INFODIDACT'2018

11. Informatika Szakmódszertani Konferencia

Előadáskötet

2018

Szerkesztette: Dr. Szlávi Péter, Dr. Zsakó László
Megjelenés: 2019. április

ISBN 978-615-80608-2-0

Tartalom

A micro:bitek felhasználási lehetőségei az oktatásban

Dr. Abonyi-Tóth Andor

abonyita@inf.elte.hu
ELTE IK

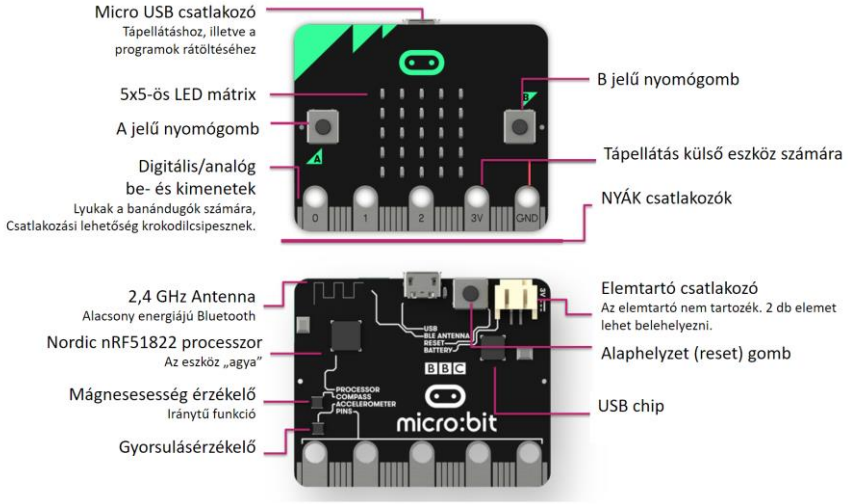
Absztrakt. A micro:bit egy oktatási célra kifejlesztett, egylapkás mikrovezérlő (SBC), amely a BBC ötlete és koordinálása alapján került kifejlesztésre, annak érdekében, hogy a diákok minél korábban betekintést nyerjenek a programozás, illetve a mérnöki tudományok alapjaiba, ezzel is ösztönözve őket, hogy később a STEM területekkel kapcsolatos pályát válasszanak. A mikrovezérlő kiválóan használható arra, hogy a gyerekek játékosan elsajátíthassák a programozás alapjait, a kiegészítőkkel pedig betekintést nyerjenek a robotika alapjaiba, de az eszköz lehetőségeit minden más műveltségterületen is sikerrel lehetne használni. Cikkemben bemutatom az eszköz felhasználási lehetőségeit, valamint az oktatásban szerzett tapasztalatokat.

Kulcsszavak: micro:bit, STEM, szenzorok, játékos programozás, mérnöki tudományok

1. A micro:bit jellemzői

A micro:bit egylapkás mikrovezérlő igen sokrétű alkalmazást tesz lehetővé az oktatás területén. Az 5x5-ös LED kijelzőjén megjeleníthetünk számokat, szövegeket, ikonokat, animációkat, illetve olyan rajzobjektumokat (sprite), amelyek segítségével egyszerűen készíthetünk különböző játékokat is.

Az alkalmazások fejlesztése során nem csak a kijelzőre támaszkodhatunk, hanem akár hangokat, dallamokat is lejátszhatunk. Fülhallgató csatlakoztatására alkalmas Jack formátumú kimenet ugyan nincs az eszközön, azonban a NYÁK csatlakozók segítségével (krokodilcsipesszel, vagy egy megfelelő átalakítóval) megoldhatjuk, hogy a fülhallgató Jack dugóját az eszközhöz illesszük.



1. ábra: A micro:bit felépítése [1] [1]

[1] Az ábra magyarítását a szerző végezte

Az eszközzel többféle módon is interakcióba léphetünk, egyrészt használhatunk két nyomógombot (A és B jelű), amelyeket külön-külön, illetve egyidőben is lenyomhatunk. Másrészt a beépített szenzoroknak köszönhetően különböző gesztusokkal is irányíthatjuk az eszközt, legyen az az eszköz különböző irányokba (balra, jobbra, fel, le) billentése vagy akár megrázása.

Az eszköz tápellátását egy a számítógéphez csatlakoztatott USB kábellel is megoldhatjuk, de akár külső tápellátást is biztosíthatunk az eszköz számára 2 db AAA elem segítségével. Így a számítógéptől független használatra is lehetőségünk van, akár szabad térben is (például hőmérséklet kijelzésére), vagy éppen viselhető eszközöket is készíthetünk, amely lehet akár lépésszámláló, ugrás számláló, okosóra vagy egy távirányító egy mobil eszközhöz.

Az eszközök rádió/bluetooth kapcsolattal is el vannak látva, így olyan alkalmazásokat is készíthetünk, amelyekben több eszköz is kommunikál egymással. Egyszerűen készíthetünk többfelhasználós játékokat, vagy akár megoldhatjuk, hogy az egyik eszköz által mért értékeket (hőmérséklet, gyorsulás, iránytű, fényerősség) egy másik eszközre továbbítsuk, és azon jelenítsük meg, akár grafikon formájában.

A pin csatlakozók segítségével analóg és digitális kimeneteket és bemeneteket is használhatunk, vagyis lehetőségünk van külső szenzorok, vagy akár szervómotorok csatlakoztatására és vezérlésére is, így az eszköz segítségével akár a robotika témakörébe is bevezethetjük a diákokat, vagy pontosabb méréseket is elvégezhetünk, mint amelyet a lapkára épített érzékelők lehetővé tesznek.

A micro:biteket többféle csomagban is meg lehet vásárolni, oktatási intézmények számára a *club* elnevezésű csomag ajánlott, amely 10 db micro:bit lapkát, ugyanennyi USB kábelt, valamint elemtartót (a szükséges elemekkel) tartalmaz.

2. Az eszköz programozási lehetőségei

A micro:bit egy mikrovezérlő, amelyre a már lefordított programot tudjuk rátölteni. Ez megtörténhet USB kábelen, vagy akár Bluetooth kapcsolaton keresztül. Utóbbi esetben az Android, illetve iOS eszközökre telepíteni kell a *micro:bit companion* nevű applikációt, amellyel a micro:bitet párosíthatjuk a tablettel. Az eszközre (többek között) az alábbi környezetekben fejleszthetünk alkalmazásokat [1][2]:

- JavaScript Blocks Editor[2]: Az online elérhető környezetben egy vizuális blokknyelvet használhatunk, vagy Javascript nyelven is írhatjuk a kódot, amennyiben erre a nézetre váltunk át.
- Makecode for microbit[3]: A Windows 10 operációs rendszerre telepíthető applikáció ugyanazt tudja, mint a JavaScript Blocks Editor, néhány kényelmi funkcióval kiegészítve.
- Scratch 3.0[4]: a környezet ezen változatában már a microbit kiterjesztést is telepíthetjük, így a diákok akár a megszokott Scratch környezetet is használhatják az alkalmazások fejlesztésére.
- Python editor[5]: Online elérhető felületet, melyben a Python nyelven fejleszthetjük az alkalmazásokat.
- Swift[6]: az applikációt iOS eszközökre telepíthetjük, majd a Swift nyelvben programozhatjuk az eszközt.

[2] https://makecode.microbit.org/

[3] https://www.microsoft.com/store/productId/9PJC7SV48LCX

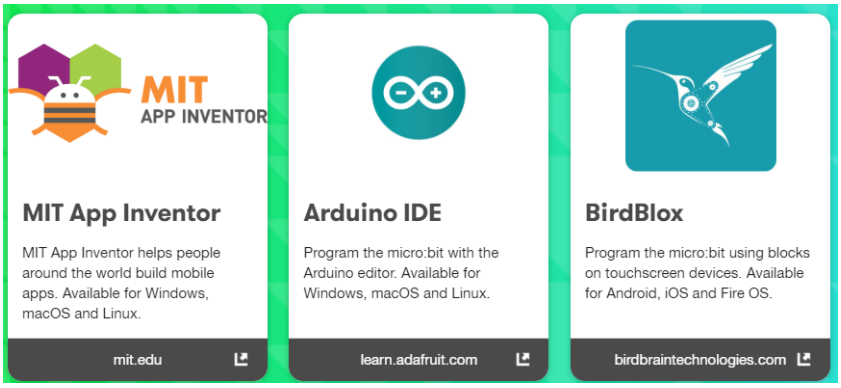
[4] https://scratch.mit.edu/microbit

[5] https://python.microbit.org/v/1.1

[6] https://microbit.org/guide/mobile/#swift

- mBed OS 5[7]: ebben a környezetben C++ kódot fordíthatunk le olyan IoT eszközökre (köztük a micro:bitre is), amelyek az ARM által gyártott processzorokon alapulnak.

A fenti környezeteken kívül számos más, harmadik fél által fejlesztett szerkesztőfelület is rendelkezésre áll [3].



2. ábra: Harmadik fél által fejlesztett fejlesztői környezetek listájának egy részlete

3. Kompatibilis hardver eszközök

A micro:bit mikrovezérlőhöz rengeteg hardver kiegészítő érhető el a piacon. Ezeket leginkább a következő kategóriákba lehet besorolni:

- *Fejlett multimédiás eszközök*: a micro:bit 25 piros LED-et tartalmaz, azonban külső kiegészítőkkel akár színes LED sorokat is használhatunk, vagy 7 szegmenses LED kijelző használatára is van lehetőség. A hangok megszólaltatásához pedig egyszerűen csatlakoztatható hangszóró panelt, vagy akár zongora modult is beszerezhetünk.
- *Robotikai kiegészítők:* ezek között találhatunk autókat (buggy), oktató robotokat, illetve robot karokat, amelyeket a micro:bit lapkával irányíthatunk, vezérelhetünk.
- *Játék kiegészítők:* a micro:bitek alkalmasak egyszerűbb játékok készítésére, a vezérlési lehetőségeket terjeszthetik ki azon játék konzolok, amelyekben elhelyezhetjük a micro:bit lapkát, így az eszközön elérhető két gomb helyett sokkal több áll rendelkezésre, vagy akár botkormányt is használhatunk.
- *Kísérletező készletek:* beszerezhetünk olyan modulokat (pl. élcsatlakozó), amelyek segítségével a külső eszközöket egyszerűbb módon csatlakoztathatjuk a micro:bit lapkához. Ezek természetesen akár külső szenzorok is lehetnek.
- *Eszköz rögzítése:* mivel a micro:bit viszonylag kicsi eszköz, akár ruhára csíptetve, karóraként, vagy nadrágövre húzva is használható, ezen alkalmazásokat könnyítik meg a különböző rögzítő és tartó elemek.
- *Eszköz védelme:* A micro:bit csomag nem tartalmazza azon műanyag tokokat, amelyekkel az eszközöket megvédhetjük a különböző sérülésektől, ezért ezeket külön tudjuk csak beszerezni.

Érdekességként megemlítjük, hogy már hazai gyártású kiegészítőket is lehet vásárolni a micro:bit lapkákhoz.

[7] https://os.mbed.com/

4. A micro:bitek használata – hazai helyzet

Ahhoz, hogy az eszköz elterjedhessen a hazai oktatásban, szükség van arra, hogy a tanárok továbbképzéseken megismerkedhessenek az eszközzel. Szerencsére egyre több konferencián találkozhatunk a microbitekkel, akár frontális előadás, akár műhelyfoglalkozás formájában, magyar nyelvű webináriumokat is megnézhetünk a témában, valamint az eszköz processzorát gyártó ARM cég magyarországi irodája is élen jár abban, hogy a diákok és tanárok megismerhessék az eszközben rejlő lehetőségeket műhelyfoglalkozások és továbbképzések formájában.

Az eszköz iránt érdeklődő tanárok és diákok számára magyar nyelvű facebook csoportok[8] is indultak, melyekben meg lehet osztani az eszközzel szerzett tapasztalatokat, illetve hasznos ötletekhez lehet hozzá jutni.

A továbbképzések és szakmai fórumok mellett nagy szükség van olyan magyar nyelvű segédanyagokra is, amelyek segítik a tanárokat az eszközben rejlő lehetőségek kiaknázásában. Ezért döntöttünk úgy, hogy kidolgozunk és szabadon hozzáférhetővé teszünk egy olyan szakköri anyagot [4], amelyben az animáció készítés témakörétől, a játékfejlesztésen át, a szenzorok használatáig a barkácsolásig bezárólag bemutatjuk az eszközben rejlő lehetőségeket.

A szakköri anyag 14 alkalomra került kidolgozásra, ahol az egyes alkalmak 90 percesek. A szakköri anyagot 2018. májusi publikálása óta 3220 alkalommal töltötték le a honlapunkról, de mivel a segédanyag az eszköz hazai forgalmazójának weblapjára is kikerült, a valós letöltési szám ennél magasabbra tehető.

A szakköri anyag készítése során egyértelművé vált, hogy a Makecode blokk szerkesztő felület magyarítása nem teljes, illetve a fordítások mögött nincs egységes koncepció. Ezért a felület magyar nyelvű fordításába is rengeteg energiát kellett bevonnunk ahhoz, hogy jól használható anyag szülessen, és minden gyakori funkciót magyarul is használhassanak a diákok.

Az ELTE Informatikai Karán az informatika tanárképzésben résztvevő hallgatókat is bevonjuk a micro:bitekkel kapcsolatos tevékenységekbe, és számos olyan szakdolgozati témát írtunk ki, amelyek olyan segédanyagok készítését célozzák, amelyek a hazai oktatásban felhasználhatóak lesznek. Így a közeljövőben további oktatási segédanyagokat publikálunk a weboldalainkon.

Szerencsére egyre több tanár fedezi fel az eszközben rejlő lehetőségeket és készítenek olyan segédanyagokat, amelyek segítik az oktatásban történő felhasználást. Ezen anyagokra mutató linkeket folyamatosan gyűjtjük és a portálunkon ezeket is közzé tesszük[9].

Az oktatási segédanyagok persze csak akkor hasznosak, ha az eszközök elérhetővé válnak a hazai iskolákban. Ezen is próbálunk segíteni (lehetőségeinkhez mérten) a következőkben ismertetett kezdeményezésünkkel.

4.1. A micro:bit botorkálás program

A „*Micro:bit botorkálás*[10]" nevű programot az NJSZT Közoktatási Szakosztálya, valamint az ELTE T@T laborja indította el 2017. októberében. A kezdeményezés célja, hogy a BBC micro:bit eszköz (ha csak átmeneti használatra is), minél több iskolába eljuthasson, hogy a diákok játékos programo-

[8] micro:bit tanári csoport (https://www.facebook.com/groups/898764273601915/), micro:bit műhely (https://www.facebook.com/groups/1194017457408416/)

[9] http://microbit.inf.elte.hu/segedanyagok/

[10] http://microbit.inf.elte.hu/

zási/kódolási tevékenységeken keresztül megismerhessék, kipróbálhassák az eszközben rejlő lehetőségeket, ezzel is fejlesztve az informatikai gondolkodásukat.[11]

Kezdetben két micro:bit csomagot vásároltunk (10-10 db eszközzel), amelyeket elindítottunk a programban részt vevő iskolák között. Az iskolák az eszközöket díjtalanul használhatták (kezdetben 2-3 hét, most már 4 hét időtartamig). Az egyetlen elvárás az iskolák felé az, hogy a kipróbálási időszak után az eszközöket postán kell továbbítaniuk a következő iskolába, illetve a használatról egy rövid beszámolót kell elkészíteniük. A programba folyamatosan várjuk az iskolák jelentkezését.

2018. márciusában a micro:bit hazai forgalmazója (Málna PC) adományozott egy csomagot a program számára. 2018 májusában pedig az eszköz processzorát gyártó ARM cég hazai irodája 20(!!!) csomaggal bővítette a programot, így jelenleg 23 micro:bit csomagot használunk folyamatosan az iskolákban, amely nem kis szervezési feladattal jár.

A program indulása óta már közel 80 iskolában jártak a készletek. A kollégák a kipróbálási időszakban hagyományos tanórákon, délutáni foglalkozásokon, vagy akár projekt napokon is felhasználták a készleteket, egy-egy iskolában. Az ELTE T@T laborja is rendszeresen tart micro:bit foglalkozásokat, így a készleteknek köszönhetően már legalább 3000 diákot értünk el a különböző tanórákon, foglalkozásokon.

5. Az eszköz felhasználási lehetőségei az oktatásban

Az eszköz sokoldalúsága miatt bármelyik műveltségi területhez kapcsolódóan felhasználható. Az egyes alkalmazásokat projektmunkában, a gyerekek aktív közreműködésével érdemes elkészíteni.

Az eszköz legsokrétűbb felhasználási módja az informatika tantárgyban van, hiszen az algoritmizálás, adatmodellezés, a programozás eszközei, alkalmazói feladatok megoldása, alkalmazói rendszerek kezelése, adatvizualizáció, problémamegoldás számítógéppel, infokommunikáció, informatikai eszközök működési elvei, az informatika matematikája témakörök [5] mindegyikéhez tudunk élvezetes, micro:bit alapú tevékenységeket társítani. Az alábbiakban néhány egyszerű felhasználási ötletet mutatunk be a további műveltségi területekhez kapcsolódóan.

- *Magyar nyelv és irodalom*: digitális történetmesélés a micro:bit lehetőségeire/korlátaira hangolva.
- *Matematika*: számtani műveletek elvégzése, kijelzése; játékos számolási gyakorlatok; adatvizualizáció, grafikonok megjelenítése; koordináta geometria bevezetése, ezen alapuló egyszerű játékok készítése.
- *Ember és társadalom*: kvíz játékok (pl. a kijelzőn megjelenő évszámhoz tartozó eseményt kell megadni)
- *Ember és természet*: mérések elvégzése beépített és külső szenzorok segítségével; a mérési adatok továbbítása és rögzítése majd feldolgozása.
- *Földünk – környezetünk*: kincskereső játék az iránytű felhasználásával; mérések elvégzése és kiértékelése; időjárás állomás készítése.
- *Művészetek*: ikonok készítése; animációkészítés; dallamok lejátszása, komponálása.
- *Életvitel és gyakorlat*: közlekedéssel kapcsolatos szimulációk (pl. közlekedési lámpa, robotok vezérlése); okosotthon alkalmazások; tárgyak tervezése, barkácsolás micro:bit alapokon.

[11] Ez már nem az első ilyen jellegű kezdeményezés Magyarországon, 2015-ben Fári János indította el a nagy sikerű Robotcsámborgás programot, amelyben Bee-Bot robotméhecskét igényelhetnek az iskolák kipróbálásra.

- *Testnevelés és sport*: viselhető lépésszámláló, ugrásszámláló; labdabirtoklás számláló; eredményki-jelző alkalmazások.
- *Idegen nyelvek*: az eszköz programozására használható blokk környezetet már számos nyelvre lefordították (vagy fordítás alatt áll)[12], így akár idegen nyelven is készíthetünk egyszerű alkalmazásokat. Idegen nyelvű weboldalakon számos projektötletet is találhatunk; a gyerekek külföldi partneriskolák diákjaival közösen is fejleszthetnek alkalmazásokat.
- *Mindegyik műveltségi terület*: micro:bit alapú szavazórendszer használata; vetélkedő játék (csapatok jelezhetik, hogy tudják a választ, a tanári micro:bit jelzi a leggyorsabb csapat sorszámát)

6. A micro:bit használatának tapasztalatai

A micro:bit használatának tapasztalataival több tanulmány is foglalkozik. 2016-ban Angliában és Walesben minden hetedik osztályos, Észak-Írországban minden nyolcadik osztályos, Skóciában minden középiskolai első osztályos diák ingyenesen hozzájutott az eszközhöz. Az eszköz használatának 1 éves tapasztalatait összegző tanulmány [6] a következő megállapításokat tette.

A diákok

- 90%-a egyetértett azzal, hogy az eszköz megmutatta nekik, hogy bárki képes kódolni,
- 88%-a nyilatkozott úgy, hogy a micro:bit megmutatta számukra, hogy a kódolás nem annyira bonyolult, mint azt előtte gondolták,
- 45%-uk jelezte, hogy szívesen választanának az informatikával kapcsolatos szakmát. Ez az arány a micro:bit használata előtt 36% volt. Ha a lányok válaszait külön vizsgáljuk, még nagyobb a növekedés, a 23%-ról 39%-ra nőtt azok száma, akik hivatásuknak is szívesen választanák az informatikát.

A tanárok

- 70%-a nyilatkozott úgy, hogy már a második félév végén az oktatásban is szeretnék használni az eszköz lehetőségeit,
- 85%-uk egyetértett azzal, hogy az eszköz az IKT/Informatika órákat élvezhetőbbé tette a diákok számára,
- 80%-uk szerint a micro:bit megmutatta a gyerekeknek, hogy a kódolás nem annyira bonyolult, mint azt korábban hitték.

Azon tanárok közül, akik már használták az eszközt, közel 50% nyilatkozott úgy, hogy tanárként magabiztosabbá váltak.

Magyarországon a micro:bit botorkálás program keretében elsőként jutott el az egyik készletünk a Kispesti Puskás Ferenc Általános Iskolába, ahol a lelkes informatika tanárok a micro:bit használatával kapcsolatban egy kutatást is végeztek 3-8. osztályos diákok (N=170) bevonásával [7].

A kutatás az alábbi főbb megállapításokkal zárult:

„*Az eredmények azt mutatják, hogy a programozási készségek az életkor előrehaladtával fejlettebbek, azonban a növekedés viszonylag kicsi (H1); jelentősebb teljesítménybeli eltérés egyedül a 3. évfolyam eredményeiben mutatkozott. A tanulók által elért eredmények nemek szerinti összehasonlítása nem mutatott szignifikáns különbséget, a fiúk és lányok eredményeinek eltérése egyik évfolyam esetében sem jelentős (H2), a vizsgált évfolyamok átlagos eredményeinél a nemek között kevesebb mint 2% differencia volt. A tanulók kutatás közben programozás során elért*

[12] https://crowdin.com/project/kindscript (A Makecode környezet fordítási felülete az egyes nyelvekkel)

eredményei nem mutattak összefüggést az előző évvégi matematika osztályzatokkal(…). A tanulóknak a micro:bithez és programozásához való viszonyulása pozitív, a tanulók több mint 90%-a jól vagy nagyon jól érezte magát a BBC micro:bites órákon; az alsó és felső tagozatos diákok viszonyulását összehasonlítva láthatóvá vált, hogy az alsó tagozatos diákok jóval nagyobb arányban válaszolták azt, hogy „nagyon jól érezték magukat" az eszközök programozása közben. A vizsgált mintánk esetében egyértelműen kijelenthető, hogy a 4. évfolyam a legoptimálisabb a programozás oktatásának elkezdéséhez, hiszen egyfelől jó eredménnyel teljesítették a programozási feladatokat, másfelől azt örömmel is tették." [7/199. o.]

A micro:bit botorkálás program kapcsán szervezőként magunk is kíváncsiak vagyunk a tanárok véleményére, ezért örömmel olvassuk az elküldött beszámolókat. Ezekben sokszor a gyerekek véleményét is olvashatjuk:

- *"Tetszik, hogy kézben elfér és könnyű kezelni"*
- *"Sok mindent ki lehet belőle hozni, és kiegészítőkkel még többet."*
- *"Otthon is tudtam folytatni a játékot és a fejlesztést, eszköz nélkül is"*
- *"Nagyon jó, mert sok olyan dolgot meg tudtam valósítani, amit az internetes felületeken nem lehet!"*
- *"Nekem személy szerint nagyon tetszett, kreatív játék, csak azt hiányoltam, hogy nincs hozzá egyéb kiegészítő."*
- *"Annyira megtetszett, hogy saját magam készíthetek játékokat, amik működnek is, hogy kértem a szüleimtől micro:bitet ajándékba és most már otthon is játszhatok vele!"*

Ezen vélemények (is) jól mutatják az eszközben rejlő lehetőségeket, a könnyű kezelhetőség, egyszerű programozás, a kézzel foghatóság és az, hogy a szimulátor segítségével is tovább lehet fejleszteni az alkalmazásokat, mind nagyon tetszenek a gyerekeknek, de az is látszik, hogy ahhoz, hogy továbbléphessenek a fejlesztésben, szükség van további kiegészítőkre is, amelyekkel akár a robotika témakör felé is nyitni lehetne.

Az is figyelemre méltó, hogy a gyerekek szívesen foglalkoznak a micro:bitek programozásával otthon is, és ajándékba is ezt az eszközt kérik a szülőktől, hogy maguk (és társaik) szórakoztatására készítsenek alkalmazásokat. Nem sok olyan oktatási eszköz van, amelyről ugyanez elmondható és az áruk is lehetővé teszik, hogy a családok ténylegesen beszerezzék azokat.

7. Összefoglalás

Érezzük, látjuk, tapasztaljuk, hogy a micro:bitek oktatási alkalmazásában rendkívül nagy potenciál van, így szükség lenne arra, hogy az iskolák saját eszközparkkal rendelkezzenek, ne csak egy rövid időszakban próbálhassák ki az eszköz lehetőségeit.

Több olyan visszajelzést is kaptunk, hogy az eszközök annyira megtetszettek a gyerekeknek és tanároknak, hogy alapítványi forrásból, vagy akár pályázat útján az iskola beszerzett néhány készletet. Nagyon indokolt lenne, hogy minél előbb, célzott, államilag támogatott pályázatok kerüljenek kiírásra, amelyek lehetővé teszik az eszközök felhasználását a magyar közoktatásban is.

Angliában a micro:bitek népszerűsítéséhez – a Kirklees Library kezdeményezésére – iskolai könyvtárak is csatlakoztak. Ezen könyvtárak lehetővé teszik a diákoknak, hogy a micro:bit eszközöket éppúgy kikölcsönözhessék, mint a könyveket. Hasonló szolgáltatást érdemes lenne bevezetni az iskolai könyvtárakban is, amennyiben az eszközök nagyobb számban megjelennének az adott intézményben.

Irodalom

1. *Microbit hardware description*
   https://tech.microbit.org/hardware/ (utoljára megtekintve: 2018.11.05.)
2. *Micro:bit editors*
   https://microbit.org/code/ (utoljára megtekintve: 2018.11.05.)
3. *Micro:bit Third Party Editors*
   https://microbit.org/code-alternative-editors/ (utoljára megtekintve: 2018.11.05.)
4. Abonyi-Tóth Andor: *Programozzunk micro:biteket!* (2018)
   http://microbit.inf.elte.hu/szakkori-anyag/ (utoljára megtekintve: 2018.11.05.)

   ISBN 978-963-284-992-8
5. Szlávi Péter, Zsakó László: *Informatika oktatása* (2012)
   https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0052_34_informatika_oktatasa/index.scorml (utoljára megtekintve: 2018.11.05.)
6. *BBC micro:bit celebrates huge impact in first year, with 90% of students saying it helped show that anyone can code* (2017)
   https://www.bbc.co.uk/mediacentre/latestnews/2017/microbit-first-year (utoljára megtekintve: 2018.11.05.)
7. Czékmán Balázs, Kiss József (2018): *Digitális eszközök használata az osztálytermekben. Egy BBC micro:bites projekt tapasztalatai,* Educatio 27 (1), pp. 111–120 (2018), DOI: 10.1556/2063.27.2018.1.9

Algoritmusok Oktatása Algoritmus Vizualizációval

Bende Imre

beiraai@inf.elte.hu
ELTE IK

Absztrakt. Útmutatást szeretnék adni arra, hogy az egyes algoritmusokat, milyen módon célszerű bemutatni, megtanítani algoritmus vizualizációs eszközökkel, hogy a diákok számára az algoritmusok elsajátítása hatékonyabb, látványosabb, élvezhetőbb legyen. Illetve kitekintek pár példával arra, hogy mely algoritmusoknál, mire kell odafigyelni, egy-egy eszköz kiválasztásakor. Időrendben kategorizálva vizsgálom az eszközöket, illetve azok újításait, tulajdonságait. Az egyes kategóriák hatékonyságát már publikált tanulmányokkal is alátámasztom.

Kulcsszavak: algoritmus vizualizáció, programozásoktatás, algoritmus

1. Bevezetés

A cikk során négy nagyobb részre osztom szét az algoritmus vizualizációs eszközöket (röviden: AV). Ezeket a kategóriákat a megjelenés időpontja, jelentősebb újítás bejövetele és az interaktivitási szintjétől függően alkottam meg. Az utóbbi, mostanában is releváns kategóriákát részletesebben is áttekintem vizsgálva azt, hogy hogyan lehetne az ezen csoportbéli eszközöket oktatásban is minél hatékonyabb oktatásmódszertani eszközként felhasználni, kiemelve pár konkrét algoritmuscsoport oktatási tapasztalatával, egyes esetekben megnézve annak „mérhető" eredményességet (a mérhetőt idézőjelbe tettem, mivel ez nem egy átfogó, minden esetben igaz eredményt ad, hanem csak a vizsgált csoportok sikerességét mutatja).

2. Algoritmus vizualizációk csoportjainak bemutatása

Az 1980-as években jelent meg [1] először az algoritmus vizualizáció a programozásoktásban. Ez idő alatt rengeteg eszköz jelent meg és nagy változások mentek végbe, tanulva az előzők hibáiból, hiányosságaiból. A következőkben négy részre szétosztva, időrendben tekintem át, részletezem az egyes időszakokat.

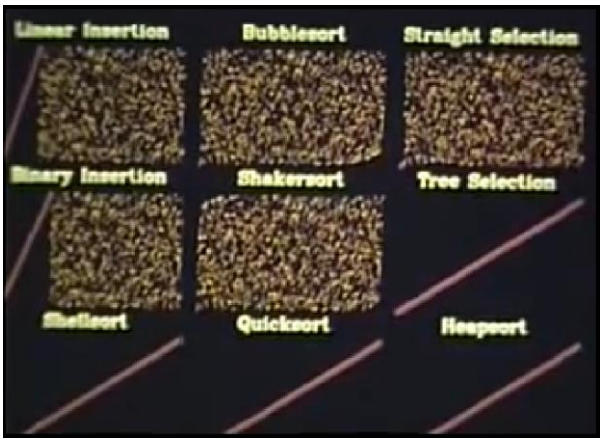
2.1. Statikus vizualizációk

Az algoritmusokat oktató tankönyvekben statikus vizualizációkat találhatunk meg. Ez a módszer az algoritmus lépéseit statikus illusztrációkkal próbálja elmagyarázni, melyekhez társulnak magyarázatok, illetve megjelenik egy leíró nyelven az algoritmus maga is (struktogramban, pszeudokódban, vagy egy adott programozási nyelvben vett implementációjában). Ezek bizonyos mértékben könnyebbé teszik az algoritmus működésének elképzelését, de összetettebb algoritmusoknál ez nem tud elégséges segítséget adni a működés megértéséhez, statikus mivolta miatt.

2.2. Animált/dinamikus vizualizációk

Az 1980-as évektől megjelentek az első algoritmus animációk, melyek egy-egy algoritmus működését próbálták ábrázolni, bemutatni. Az előző részhez hasonló illusztrációk most már folyamatában jelentek meg, így az algoritmus működése átláthatóbbá vált, gördülékenyebbé téve az algoritmus egész működésének a megtekintését. Az első ismertebb példa a „Sorting out Sorting" című videó volt [1. ábra], mely a rendezéseket mutatta be egy-egy példán keresztül, amelyekhez szóbeli magyarázat is társult [1]. Többségében az animációk értelmezéséhez azonban szükséges megfelelő előzetes

tudás az algoritmusról, így egyéb segédanyag megléte is szükséges az oktatáshoz. Az efféle vizualizációk nem mutattak szignifikáns javulást a tanulás hatékonyságában [3]. Ez főleg amiatt lehetett, hogy a tanulók csak az animáció nézői lehettek, nem pedig a tanulási folyamat résztvevői. Az algoritmus animációk azonban mindenképpen fontos alapot szolgáltak a későbbi vizualizációk létrejöttéhez.



1. ábra: Részlet a „Sorting Out Sorting" nevű algoritmus animációból, amelyen a rendezéseket hasonlítja össze (többek között a beszúró, buborékos, gyors, shell rendezéseket)

2.3. Interaktív vizualizációk

2.3.1. Interaktív vizualizációk bemutatása

Az 1990-es évektől, köszönhetően a személyi gépek elterjedésének, illetve később az Internet térhódításának megjelentek az interaktív algoritmus vizualizációk, ahol a hallgatók már résztvevőkké válhattak (nézőkből az algoritmus léptethetőségével, bemenet módosíthatóságával és egyéb módszerekkel a vizualizáció irányítóivá váltak). Az Internet segítségével könnyen, szabadon, ingyenesen elérhetővé váltak mások munkái, így a megfelelő forrás megtalálása után könnyen felhasználhatóvá váltak az eszközök.

A vizualizációk léptethetővé, visszatekerhetővé váltak (az egyes lépéseket az algoritmus megjelenítésénél is követhetjük). A bemenet állítható lett, így bármely inputra megnézhetővé vált az algoritmus működése. Egyes eszközök pedig az interakciót úgy próbálták növelni, hogy kérdéseket tettek fel azzal kapcsolatban, hogy mi lehet a következő lépés, mi változhat (ez az elem, ahogyan a diákoktól való interaktivitás szükségességét növelte, úgy a tanulás hatékonyságát is javította, ez részletesebben Naps tanulmányában is látszik [5]).

Ennél a fázisnál már érvényesülhettek a Myller által megfogalmazott kiterjesztett interaktivitási szintek [4] (ezek a szintek a megtekintés, ellenőrzött megtekintés, bemenet beírása, válaszadás, változtatás, módosítás, összerakás, bemutatás, reagálás).

Emellett egyre több olyan eszköz, környezet, keretrendszer is létrejött, amelyekkel a hallgatók önmaguk is létrehozhatnak vizualizációkat, így a kiválasztott algoritmus működését mélyebben megvizsgálva, még jobban megérthetik azt, sőt társaiknak bemutatva fejleszthetik egyéb kompetenciáikat is (kreativitás, kommunikációs, előadói készség).

2.3.2. Általános algoritmusok vizualizációi

A tanulók elsőként a programozási tételekkel találkoznak oktatási környezetben (például: összegzés, megszámolás, maximum-kiválasztás, keresés). Ezeken keresztül sajátítják el aztán az algoritmikus gondolkodást, így fontos, hogy már az elején átlássák ezeknek az algoritmusoknak az alapjait, működését. Nagy segítség lehet az algoritmus vizualizációs eszközök használata ahhoz, hogy tisztán lássák azt, hogy mi történik, milyen változások történnek az egyes lépésekben.

2. ábra: Az általam készített AV eszközből[13] egy minta, a megszámolás tétel bemutatására

A tanári szakdolgozatomhoz [2] készített algoritmus vizualizációs honlapot igyekeztem minél teljesebben létrehozni, hogy minden szükséges elem fent legyen azon, hogy a diákok könnyebben megérthessék, megtanulhassák az adott algoritmust. Így tehát nézzük meg, hogy milyen komponensekből állhat egy AV [2. ábra], illetve, hogy ez az én eszközömön, hogyan jelennek meg ezek:

- Vizualizáció. Az algoritmus működése, illetve a hozzá tartozó adatszerkezet látványos, könnyen értelmezhető formában kerül ábrázolásra. Jelen esetben a bemeneti tömböt végig láthatjuk, ezen különböző színekkel vannak jelölve a speciális elemek (megszámolás tételnél zölddel a feltételt teljesítő elemek, kékkel pedig az éppen vizsgált elem).
- Bemenet állíthatósága. Az algoritmus bemenetét tudjuk módosítani, így különböző eseteknél is meg tudjuk vizsgálni az algoritmus működését, kimenetét. A weboldalamnál a bemeneten kívül a megszámolás feltétele is állítható.
- Algoritmus. Az algoritmus megjelenik valamilyen leíró nyelven. Lehet az pszeudókód, struktogram, vagy akár valamilyen konkrét programozási nyelven megírott implementációja. Az aktuális lépés utasítását a szoftver jelzi (egy fekete háromszöggel).
- Lépések állíthatósága. Az egyes utasítások között megvan a lehetőség, hogy (előre/visszafele) léptethessük a vizualizációt. Sőt folyamatában is meg tudjuk tekinteni azt.
- Változók követése. Látjuk az aktuális lépésnél, hogy milyen változók vannak, illetve milyen értékkel rendelkeznek. Ez a vizualizációnál és az algoritmusnál is leolvasható.
- Leírás. A felületen van egy rövid leírás az algoritmusról, illetve az egyes lépések változásait is rövid megjegyzésekkel látja el.

[13] http://ermi46.web.elte.hu/dev/algotan/

2.3.3. Algoritmus felfedezés

Az algoritmus felfedezés célja, hogy „megengedjük a diákoknak, hogy tapasztalati úton felfedezzék a teljesítménybeli jellemzőket és relatíve különböző lehetőségeket egy-egy témakörön belül, ilyen témák lehetnek például a keresőfák, hash, rendezési algoritmusok" [9]. Így nem csak egy-egy algoritmust tudnak megismerni, hanem annak futásidejű és memóriabeli hatékonyságát is, illetve össze is tudnak hasonlítani többfajta megoldást hatékonyság szempontjából.

A Shaffer és munkatársai által írt tanulmányban [9] ennek eredményességét is vizsgálták a hash algoritmusok oktatásában. Két szemeszterben két-két csoportot vizsgáltak. Az elsőt hagyományos módszerekkel tanították, a hallgatók kaptak az anyaghoz külön felhasználható irodalmat, míg a másodiknál rövid bemutatás után a diákok egyedül, vagy párokban dolgozhatták fel az előre elkészített tutorial tartalmát, amiben algoritmus vizualizációkkal mutatták be az egyes algoritmusokat, amin keresztül egy probléma megoldását is összehasonlíthatták (az oktatótól persze lehetett közben kérdezni) több különböző elvvel, koncepcióval, így az egyes esetek teljesítményét, hatékonyságát is láthatták. Ezután a diákok egy tesztet töltöttek ki, melynek eredménye a következő táblázatban látszik:

Módszer	2008 ősz	2009 ősz
eszköz nélkül	67.8% (22 fő)	68.6% (31 fő)
eszközzel	77.3% (19 fő)	73.4% (48 fő)

1. táblázat: Összegzés a hash kvíz teljesítményéről [9]

Az eredményekből látszik, hogy akiket a tutoriallal tanítottak jobb pontszámot értek el a teszten, mint akiket a hagyományos módszerekkel. Viszont nagyon óvatosan szabad csak használni az ilyesfajta megközelítést, hiszen megvan a maga kockázata is. Ahogy Pinker írta: „Felfedezés útmutatás nélkül nagyon kockázatos és túl sok időt vesz igénybe. Útmutatás felfedezés nélkül pedig csak irányítás, ami jobban támaszkodik a memorizálásra, mint a megértésre." [6]

2.4. Komoly játékok

Habár nem feltétlenül sorolnám be időrendben ezt a csoport az előző kategóriabesorolásba, de mindenképpen említést szeretnék tenni a komoly játékokról (serious games). A komoly játékok: „bármilyen formájú interaktív számítógép-alapú játékszoftverek, egy vagy több játékos játszhatja valamilyen platformon, amelyet azért hoztak létre, hogy a használata a játékosoknak több legyen, mint szórakozás" [7]. A gamifikáció fogalom népszerűvé válása előtt, már igyekeztek a programozásoktatást is élvezhetőbbé és így hatékonyabbá tenni azáltal, hogy a feladatokat számítógépes játékokba ágyazták, illetve azok elveivel építették össze.

Miért is volt kézenfekvő alap a számítógépes játékok, illetve azok világának felhasználása a tanulásban, illetve az oktatásban? (itt alapul szolgáltak a [8]-ban már megfogalmazott pontok)

- A számítógépes játékok népszerűek. Közismert, hogy a fiatalok (sőt az átlagéletkor egyre magasabbá válik) nagyon sok időt töltenek számítógépes játékokkal, így kézenfekvő, hogy célszerű oktatási eszközként felhasználni a mai világban.
- A számítógépes játékok interaktívak. Teljes egészében a felhasználó irányítja, az ő általa véghez vitt műveletek kontrollálják az eseményeket a szoftverben.
- A számítógépes játékok kompetitívek. A játékosok rengeteg időt fektetnek nem csak a játékokba, hanem azok megértésébe, megismerésébe is. Mindezt azért, hogy jobbak legyenek abban, javítsanak eredményeiken, illetve legyőzhessék önmagukat, vagy akár más játékosokat is.

- A számítógépes játékok szórakoztatóak. A számítógépes játékok használata az algoritmus tanulást élvezhetővé, érdekessé teszi.

A komoly játékok témakörében a már meglévő játékoknál vizsgálták a tanulási lehetőségeket, majd ebből kiindulva hoztak létre olyan játékokat, amelyek a diákok valamilyen készségeit fejlesztette. Ezek célja többnyire a nyelvi készségek fejlesztése, ügyességfejlesztés, valamilyen témakör tudásának átadása (többek között használtak ilyen szoftvereket a katonaságban és az egészségügyben is). Így a programozásoktatásban nem, vagy csak közvetetten használhatók fel ezek a játékok, azonban a következő részben látjuk meg azt, hogy mi lesz akkor, ha vegyítjük a programozásoktatást és a számítógépes játékokat oktatási környezetben.

2.5. Játékos vizualizációk

2.5.1. Játékos vizualizációk bemutatása

A 2010-es években vált népszerű kifejezéssé az oktatásban a gamifikáció [10]. Ez az algoritmusoktatást is érinti, megjelennek az olyan vizualizációs eszközök, amely a tanulási folyamatot igyekeznek játékos környezetbe ültetni, ezzel növelve az interakció mélységét, ami pedig ezáltal a tanulás hatékonyságát is fejleszti (több eredmény is született erre vonatkozóan, többek között Karavirta disszertációjában gyűjtötte össze a kísérletek eredményeit, amiből szépen látszik ez az eredménye [3]).

3. ábra: Feladat a „Sort Attack" nevű játékban [10]

A „Sort Attack" nevű alkalmazás [3. ábra] igyekszik az algoritmustanulást játékossá tenni különböző elemekkel, illetve emellett algoritmus vizualizációt használ közvetítő közegként (emiatt is került bele ebbe a részbe). A program rendezési algoritmusokat mutat be, illetve egyre nehezedő pályákon kell az algoritmusok lépéseit végig követni megfelelő hibázási lehetőséggel. A következő felsorolásban áttekintem, hogy milyen módon jelennek meg a Yohannis és munkatársai által [11]-ben már megfogalmazott pontok, amelyek alapként szolgálnak, hogy milyen elemek, gondolatok szükségesek ahhoz, hogy egy játékos eszközről beszélhessünk:

- Játékosok. A tanulók, az eszköz felhasználói, akiknek jelen esetben a feladata a négyzetek sorba rendezése.
- Környezet. Az eszköz adja meg a környezetet, amelyen keresztül sorba lehet rendezni a négyzeteket, illetve valamiféle információt, visszajelzést is ad (helyes/helytelen lépés, állapot mutatása, eltelt idő).

- Szabályok. Az adott rendezési algoritmus követése, amelyet a felhasználóknak be kell tartania, ahhoz, hogy teljesíthessék a pályákat/játékot. Illetve megfelelő „élettel" (hibázási lehetőséggel) rendelkezhetnek egy-egy feladat megoldása alatt. Ha ez idő/lehetőség alatt nem sikerül, akkor újra kell kezdeni a felhasználónak a pályát.
- Kihívások. Kihívásnak vehetjük az előbb említett szabályok betartását. Tehát, ebben az esetben a négyzetek megfelelő helyére húzását az adott lépéskorláton belül (algoritmus működésének/lépéseinek megfelelően).
- Interakció. Az alkalmazásban megjelennek az algoritmus főbb elemei (algoritmus egy leíró nyelven, adat, reprezentáció), amelyek a játékosok cselekedetei, interakciói révén módosulnak Jelen esetben az adott rendezési algoritmus lépése szerint kell a diákoknak a megfelelő elemet kicserélniük, áthúzniuk a megfelelő helyre, amiről aztán visszajelzést is kapnak (egy felvillanó piros/zöld színnel).
- Célok. A cél, hogy sikeresen végig kövesse egy rendezési algoritmus lépéseit a megfelelő hibázási lehetőséggel, ezáltal elsajátítsa az algoritmus működését, majd azt saját maga is fel tudja használni, le tudja programozni különböző feladatokban.
- Érzelmi tapasztalok. A kihívás, hogy teljesítse a rendezést a megfelelő szabályok szerint az adott környezetben növeli a diák motivációját. Ha nem sikerült egy feladat, akkor az felkeltheti az érdeklődését, plusz erőt adhat arra, hogy újra próbálkozzon. Míg siker esetén örömet, elégedettséget érezhet, ezután kihívást kereshet a következő, nehezebb pályákban.
- Számszerűsíthető kimenet. Ilyenfajta kimenet lehet egy-egy jutalom, pontszám, pecsét, amelyeket saját teljesítményük alapján kapnak. Például: sikerül hamarabb/kevesebb élet felhasználásával teljesíteni a rendezést. Ezek alapján látjuk, hogy hol tartanak, illetve felkészültek-e nehezebb feladatok megoldására.
- Lehetséges következmények. Végtelen számú újra próbálkozási lehetőség van, így lehetőségük nyílik a tanulóknak, hogy felfedezzék, megértsék az algoritmus működését.

2.5.2. Hatékonyságbeli eredmény

Egy egyetemen megvizsgálták a játékos vizualizációk és a statikus vizualizációk hatékonyságbeli összehasonlítását [10]. Két csoport volt: az elsőnél először hagyományos módon tanítottak a rendezési algoritmusokat (jelen esetben a buborékos, beszúró és a kiválasztó rendezéses algoritmusokat), a másodiknál játékos vizualizációval tették ezt meg. Ezután egy tesztet töltettek ki, melyben egy-egy számsorozatot kellett rendezni a tanult rendezési algoritmusokkal. A teszt befejezte után fordult a szerep, és a csoportok a másik csoport eszközeit használták fel a tanuláshoz. Végül az előző teszthez hasonló feladatsor zárta a kutatást.

Csoport	Könyv-Első			Játék-Első			Különbség	
Típus	1. Teszt	2. Teszt	Kül.	1. Teszt	2. Teszt	Kül.	1. Teszt	2. Teszt
Buborékos	8.71	10.00	1.29	8.93	10.00	1.07	0.22	0.00
Beszúró	7.31	10.00	2.69	9.64	10.00	0.36	2.33	0.00
Kiválasztó	7.42	10.00	2.58	10.00	10.00	0.00	2.58	0.00
Összeségében	7.81	10.00	2.19	9.52	10.00	0.48	1.71	0.00

2. táblázat: Rendezéses tesztek eredményei az egyes részek végén [10]

Az első teszt végeredménye alapján az látszik, hogy akik először a játékos eszközöket használták jobb eredményeket értek el, mint, akik a hagyományos eszközöket használták. A második teszt végeredményéből csak annyit szűrhetünk le, hogy a végére mindenkinek sikerült elsajátítani a kiválasztott rendezéses algoritmusok működését.

3. Összegzés

Igyekeztem bemutatni minél többféle módszert arra, hogy hogyan lehetne minél sokszínűbb, minél hatékonyabb oktatási segédeszközként felhasználni az algoritmus vizualizációt. Megnézve az interaktív, játékos vizualizációk alapköveit, illetve egy-egy példán keresztül ezek, hogyan is valósulnak meg. Láttuk, hogy a megfelelő módszerekkel a megfelelő eszközöket használva kimutatható hatékonyságbeli eredményt érhetünk el az algoritmusoktatásban. Azt már előző cikkemben is kifejtettem, hogy minél interaktívabb egy AV, annál hatékonyabb lehet egy eszköz a tanulásban. Láthattuk az idő haladtával egyre interaktívabbakká váltak az eszközök (egyre több ehhez kapcsolódó komponens tartalmaztak, illetve a Naps által meghatározott szintekből is egyre több jelent meg bennük), ezáltal kimondhatjuk azt is, hogy tanuláshatékonyságban is javult, fejlődött a módszer. Most ezt kiegészítve láthatjuk, hogy ha a diákoknak a tudást csak rejtve, játékba ágyazva próbáljuk átadni, akkor még inkább hasznos eszközzé válhat. A tanulók felfedezni vágyásból, illetve a játék örömééért ássák bele magukat egyre jobban az eszköz megértésébe, annak felhasználásába, és így az algoritmus megértésébe is. Érdemes minél többfajta AV-t bevinni az osztályba (persze csoporttól függően), így az algoritmusoktatás érdekessé, látványossá, változatossá válhat.

Terveim között szerepel, hogy a jövőben én magam is készítek egy webes játékos algoritmus vizualizációs eszközt, amelyen keresztül megmérném az eszköz tanuláshatékonyságát. Ezt az eszközt úgy valósítanám meg, hogy több feladat/pálya is legyen különböző algoritmusokhoz, a játékosok eredményei utólag is visszanézhetők, valamint a játékosok/tanulók is folyamatosan nyomon követhetik az előrehaladásukat, fejlődésüket. Ezt az ELTE Informatika Karán a „Programozás" című tárgy keretein belül tenném meg.

Köszönetnyilvánítás

EFOP-3.6.1-16-2016-00023: Kutatás-fejlesztési tevékenység megvalósítása az Eötvös Loránd Tudományegyetem szombathelyi kampuszán – A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Irodalom

1. R. Baecker: *Sorting out Sorting: A Case Study of Software Visualization for Teaching Computer Science*. Software Visualization: Programming as a Multimedia Experience, pp369-381, 1998.
2. I. Bende: *Algoritmusok lépésenként*. ELTE, 2017.
3. Ville Karavirta: *Facilitating Algoritm Vizualization Creation and Adoption in Education*. Helsinki University of Technology, 2009.
4. Niko Myller, Roman Bednarik, Erkki Sutinen, Morechai Ben-Ari: *Extending the engagement taxonomy: Software visualization and collaborative learning*. ACM Transactions on Computing Education, New York, USA, 2009.
5. L. Naps, G. Rössling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, J. Á. Velázquez-Iturbide: *Exploring the role of visualization and engagement in computer science education*. ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE'02). ACM Press, pp131–152, 2002.

6. R. Pinker: *How do students learn from models?* Concord Consortium Newsletter 11, pp14-15, 2007.
7. U. Ritterfield, M. Cody, P. Vorderer: *Serious Games: Mechanism and Effects.* Routledge, London, 2009.
8. S. Shabanah, J. X. Chen: *Symplifying algorithm learning using serious games.* Proceedings of the 14th Western Canadian Conference on Comuting Education, 2009.
9. Clifford A. Shaffer, Arpit Kumar, Mayank Agarwal, Alexander Joel D. Alon, Stephen H. Edwards: *Going Beyond Algorithm Visualization to Algorithm Exploration*. Virginia Tech, 2009.
10. Alfa R. Yohannis, Yulius D. Prabowo: *Sort Attack: Visualization and Gamification of Sorting Algorithm Learning*. VS-GAMES, pp1-8, 2015.
11. Alfa R. Yohannis, Y. D. Prabowo, A. Waworuntu: *Defining Gamification: From lexical meaning and process viewpoint towards a gameful reality*. International Conference on Information Technology System and Innovation, Bali, 2014.

Sprego avatárok a 3D térben

Dienes Nikolett[1], Gulácsi Ádám[2], Csernoch Mária[3]

[1]dienesniki@gmail.com, [2]adam.gulacsi@gmail.com,
[3]csernoch.maria@inf.unideb.hu
DEBRECENI EGYETEM, INFORMATIKAI KAR

Absztrakt. A kompetencia-alapú mérőeszközök egyértelműen bizonyítják, hogy a táblázatkezelés oktatása során jelenleg alkalmazott módszerek nem elég hatékonyak. Ezek a hagyományosnak tekinthető táblázatkezelési megközelítések az egyes szoftverek speciális funkcióira, jellemzőire, az eszközhasználatra fókuszálnak ahelyett, hogy az algoritmikus, számítógépes vagy absztrakt gondolkodás fejlesztését, valamint a számítógépes problémamegoldást helyeznék előtérbe.

Munkánk során egy olyan semi-unplugged eszközt hoztunk létre, amely programozási alaptételek Sprego-implementációját teszi elérhetővé 3D térben, demó és interaktív módban. A program alapkoncepciója szerint avatárokkal játszatjuk el az algoritmust, mellyel párhuzamosan futtatjuk a szöveges képletkiértékelőt. Ezzel célunk, hogy növeljük a diákok lelkesedését és hatékony támogatást tudjunk nyújtani a számítógépes gondolkodás, mint alapkészség fejlesztéséhez.

Kulcsszavak: Sprego, táblázatkezelés, összetett függvények, programozás, algoritmikus gondolkodás, 3D oktatószoftver

1. Motiváció

Célunk olyan módszertani megközelítések, eszközök fejlesztése, amelyekkel növelhető az informatikaoktatás hatékonysága, a tanulók számítógépes gondolkodásának, ezen belül is kiemelten az algoritmikus készség és a számítógépes problémamegoldás fejlesztése. Fontosnak tartjuk a koncepció alapú problémamegoldást [1], valamint a sémaépítést [2][3], amely technikák nagyban segíthetik a tanulói gyors és lassú gondolkodás megfelelő és megbízható alkalmazását, elősegítve ezzel a hatékony és problémamentes megoldások létrehozását, azok helyességének ellenőrzését [4][5][6].

Ezen felül növelni szeretnénk a diákok motivációját látványos és modern reprezentációkkal. Fontos szempont volt továbbá, hogy a való életből vett példákkal operáljunk, hiszen így a tanulók jobban bele tudják élni magukat a megteremtett helyzetekbe. Hosszú távú célunk pedig az, hogy egy olyan alkalmazást implementáljunk, amely egy egyedülálló szerepet tölthet be az informatika oktatásában. Ez a semi-unplugged megoldás ugyanis alkalmassá válhat arra, hogy egy hidat képezzen az unplugged (például ha matrjoska babákkal szemléltetjük az összetett függvényeket) [7][8] és a plugged-in (például ha táblázatkezelő feladatokat oldanak meg a tanulók a számítógépeknél) módszerek között.

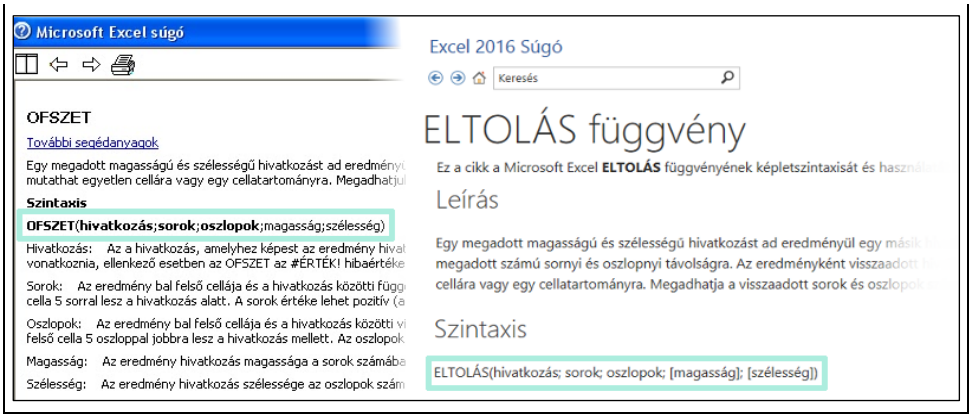
2. Elméleti és gyakorlati háttér

Kutatásunk során egy olyan 3D semi-unplugged szemléltetőeszközt hoztunk létre, amely várakozásainknak megfelelően hatékonyan fogja segíteni a diákok tanulási folyamatát a Sprego keretrendszerben, amely eltér a fentebb említett hagyományos megközelítéstől [9]. A szoftver jelenlegi verziójában két algoritmus vizualizációja érhető el, interaktív és demó módban. Az egyik egy erdei környezetben játszódó feltételes számlálás, a másik pedig egy városi környezetbe helyezett lineáris keresés.

2.1. Elméleti háttér – Sprego

Alkalmazásunk bemutatása előtt érdemes részletesen kifejtenünk azt, hogy mi is a Sprego módszer. Ez egy új, alternatív megközelítés, amely egy algoritmusalapú keretrendszert biztosít a táblázatkezeléssel kapcsolatos valódi problémamegoldáshoz [9]. A név a **Spre**adsheet és a L**ego** szavak kombinációjából alakult ki. Ahhoz, hogy megértsük a Sprego alapelveit, először részletesen meg kell vizsgálnunk a tradicionális megközelítésű táblázatkezelés buktatóit és nehézségeit.

Az első ilyen hátrány magából a táblázatkezelő szoftvercsaládokból ered, mivel a különböző programcsaládok csak a legelemibb szinten kompatibilisek. Továbbá egy programcsaládon belül is előfordulnak kompatibilitási problémák az egyes verziók között. Megtörténhet ugyanis az, hogy a táblázatkezelő függvényeket kicserélik vagy átnevezik. Erre egy példa az OFFSET() függvény átnevezése a Microsoft Excel különböző verzióiban (1. ábra).



1. ábra: Az OFFSET() függvény neve, szintaxisa és működése a Microsoft Excel 2003 (bal) és 2016 verziójában (jobb).

A második ilyen probléma az argumentumok sorrendjének következetlensége a hasonló célú függvények között. Ezt szemlélteti például az, hogy a SZUMHA() függvény argumentumai teljesen más sorrendben következnek, mint a SZUMHATÖBB() függvényé, ami növeli a szemantikailag inkorrekt formulák építésének kockázatát. [9][10].

A következő nehézség a mértéktelen mennyiségű probléma-specifikus függvényből ered. Ezek memorizálása a speciális funkciójukkal és argumentumaikkal együtt nagyon megterhelő feladat nemcsak a diákok, hanem a tapasztaltabb felhasználók számára is. Gyakori kérdés táblázatkezelői környezetben a lineáris keresés, ahol egy konkrét értéket egy vektorban, vagy egy mátrixban keresünk. A felhasználó találhat számos megoldást a problémájára, KERES(), FKERES(), VKERES() függvényeket, vagy az INDEX(HOL.VAN()) összetett függvényt alkalmazva [11][12][13][14][15]. Az első három függvénynek azonban vannak olyan limitációi, amelyek tovább korlátozzák a használatukat az összetett függvénnyel szemben (1. táblázat).

Függvény(ek)	Keresési vektor	Keresési vektor rendezettsége
KERES()	sorban és oszlopban	növekvő
VKERES()	csak sorban	növekvő / nem rendezett
FKERES()	csak oszlopban	növekvő / nem rendezett
INDEX(HOL.VAN())	sorban és oszlopban	növekvő (1, alapértelmezett) / nem rendezett (0) / csökkenő (-1)

1. táblázat: A KERES(), VKERES(), FKERES() és az INDEX(HOL.VAN()) függvények összehasonlítása.

A túl sok speciális függvény alkalmazása és tanítása nemcsak szükségtelen, hanem kevésbé hatékony is, ugyanis egy átlagos felhasználó körülbelül 12 függvényt használ normál problémamegoldó keretek között [16]. Ezen felül nem elhanyagolható tény az sem, hogy leghatékonyabban úgy fejleszthetjük a diákok számítógépes gondolkodását, ha fokozatosan építjük fel az instrukciók halmazát [17].

A Sprego egy olyan alternatíva, ami sikeres megoldást jelent a korábban említett nehézségekre. A módszer alapelve, hogy csupán 12 alapvető, általános célú táblázatkezelő függvényre van szükségünk, amelyeket mint Lego elemeket összekombinálva tudunk problémákat megoldani [9]. A 12 függvény további három csoportba sorolható, csoportonként 4-4 függvénnyel. Ez a rendszerezés lehetőséget biztosít egyfajta szemantikai és tudásszint szerinti csoportosításra (2. táblázat). Fontos megjegyeznünk, hogy ez a 12 függvényt tartalmazó halmaz nyitott, műveltségi területtől és a problémától függően bővíthető. Az egyetlen megkötés, hogy egyszerű, könnyen értelmezhető függvényeket használhatunk a bővítéshez, mint például: KICSI(), NAGY(), MEDIÁN() [9].

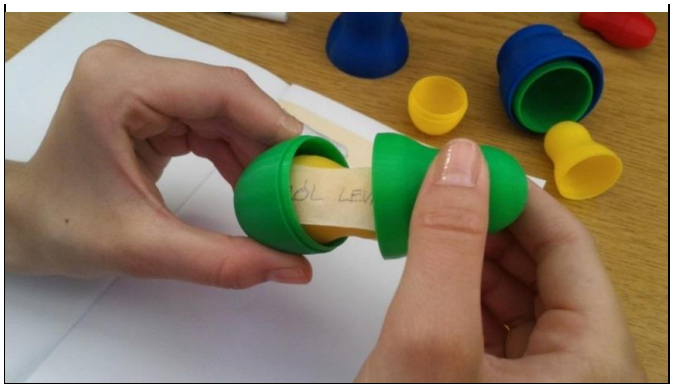
Sprego szöveg	Sprego szám	Sprego pro
BAL()	MIN()	HA()
JOBB()	MAX()	INDEX()
HOSSZ()	SZUM()	HOL.VAN()
SZÖVEG.KERES()	ÁTLAG()	HIBÁS()

2. táblázat: A 12 alapvető Sprego függvény és azok csoportosítása.

A 12 függvény megtalálható az összes táblázatkezelő szoftverben már a legelső verzióktól kezdve úgy, hogy az elnevezésük és az argumentumaik is konzisztensek az első megjelenésük óta. A Sprego felhasználónak mély megértéssel kell rendelkeznie ezen függvényeknek a működésével kapcsolatban szintaktikai és szemantikai szempontból is, hogy feladatmegoldáshoz tudja használni azokat. Ez azonban sokkal kevésbé nehéz feladat a mértéktelen mennyiségű probléma-specifikus függvény memorizálásához képest [18].

A Sprego módszer alkalmazásakor elengedhetetlen az összetett függvények és a tömbképletek használata. Az összetett függvények használatának egyik legfőbb előnye, hogy fejleszti a diákok algoritmikus és számítógépes gondolkodását, valamint tágítja a matematikai tudásukat a függvények, többváltozós függvények, tömbök, azaz n dimenziós vektorok témaköreiben [6][9].

Az eljárás másik nagy előnye, hogy új lehetőségeket nyit az unplugged eszközökkel való tanításban [7][8]. Erre egy szemléletes példa a matrjoska babák használata az osztálytermekben. A babák ugyanis tökéletesen demonstrálják az összetett függvények működését és azt, hogy ez a fajta problémamegoldás mennyire hasonlít a szöveg-alapú magas szintű programozási nyelvekhez (2. ábra).



2. ábra: Unplugged tanítási módszerek, matrjoska baba.

Megállapítható továbbá az is, hogy a módszer tökéletesen alkalmas kisiskolás korban az alkalmazói szoftverek és a programozás témakör lefedésére is, hiszen fejleszti az algoritmikus készséget és a számítógépes gondolkodást. Ezek az átfedések azért bírnak nagy jelentőséggel, mert így megvalósíthatóvá válik a kerettantervben sokszor megemlített tudástranszfer. Ezáltal a gyerekek könnyebben tudnak befogadni egy-egy új témakört, valamint kevesebb nehézséggel abszolválják a felmerülő problémákat, feladatokat. További előnye a módszernek, hogy egyetlen programozási eszközzel a kerettanterv két nagy témaköre – táblázatkezelés és programozás – is lefedhető általános iskolában és a középiskolai bevezető szakaszban, valamint teret enged a további informatikai témakörök közötti tudástranszfer megvalósításához [19].

2.2. Sprego algoritmusok 2D-s vizualizációja

Szoftverünk tervezése során egy korábbi kutatás eredményeként elkészült alkalmazásból indultunk ki [20]. A kutatócsoport egy több platformot is támogató, 2D vizualizációs applikációt készített el a különböző Sprego algoritmusok reprezentálásának céljából. Az alkalmazás két gyakran felmerülő probléma algoritmusainak ábrázolását implementálta, az egyik a feltételes számlálás, a {=SZUM(HA())} tömbképlettel, a másik pedig a lineáris keresés, az =INDEX(HOL.VAN()) összetett függvénnyel.

Az animációk fő célja, hogy a diákok jobban megértsék az algoritmusok és a Sprego függvények működését, amelyet az algoritmusok műveleteinek lépésről lépésre történő bemutatásával tettek szemléletessé.

A szoftver a való életből hozott példákkal operál, ahol az avatárok szerepét színes matrjoska babák töltik be. Az oktatásban történő felhasználás szempontjából fontos, hogy az alkalmazás a multiplatform elérhetőségnek köszönhetően támogatja az Android, Windows, Macintosh és Linux felületeken történő lejátszást is [20][21][22].

3. Kutatásunk

A legmeghatározóbb feladatunk az volt, hogy a már létező 2D-s vizualizáció koncepcióját átültessük 3D környezetbe. Ez a transzformációs folyamat két fő részre osztotta a munkafolyamatot: az első rész a grafikai feladatok kivitelezése, ami magában foglalja a 3D modellek, textúrák és UI elemek tervezését és elkészítését az új környezet igényeinek megfelelően, a második részt pedig a programozási feladatok teljesítése képezi.

A munkafolyamat megtervezésekor fontosnak tartottuk, hogy megtartsuk a 2D applikáció fő aspektusait, például a matrjoska babákat, a való életből vett példákat és a multi-platform elérhetőséget.

Ezen felül új funkciókkal is bővítettük alkalmazásunkat kihasználva a 3D adta lehetőségeket (3. ábra).

3. ábra: A 2D (bal) és a 3D (jobb) környezet összehasonlítása.

3.1. Fejlesztői környezetünk

Szoftverünket a Unity3D fejlesztőkörnyezetben implementáltuk [23]. A Unity3D egy játékfejlesztő engine, amely használható 2D és 3D applikációk létrehozásához is.

Ebben a környezetben a fejlesztéshez két fő eszköztár áll rendelkezésre: a beépített eszközök, valamint a programozó API, amelyen keresztül a projektben megtalálható objektumok működését tudjuk módosítani C# szkriptek segítségével. A beépített eszközök sokszínűsége és széleskörű felhasználhatósága nagyban megkönnyítette a kezdeti prototípusok előállítását. A C# szkriptekkel tetszőlegesen testreszabható az objektumok viselkedése a rendszerben. Ez a kettős eszközrendszer biztosította az optimális munkafolyamatot a projekt fejlesztési ideje alatt.

Fontos érv volt még a Unity3D engine mellett, hogy non-profit használat esetén teljesen ingyenesen alkalmazható. Ezen felül elmondható, hogy egy nagyon jól dokumentált rendszerről van szó [24]. Rendkívül népszerű, segítőkész és aktív fejlesztői közösséggel rendelkezik [25]. Ezek mind hasznos segítségnek bizonyultak a fejlesztés folyamán, és nagyban megkönnyítette a tanulási folyamatot, valamint a felmerülő problémák, bugok kiküszöbölését és megoldását.

3.2. Grafikai feladatok

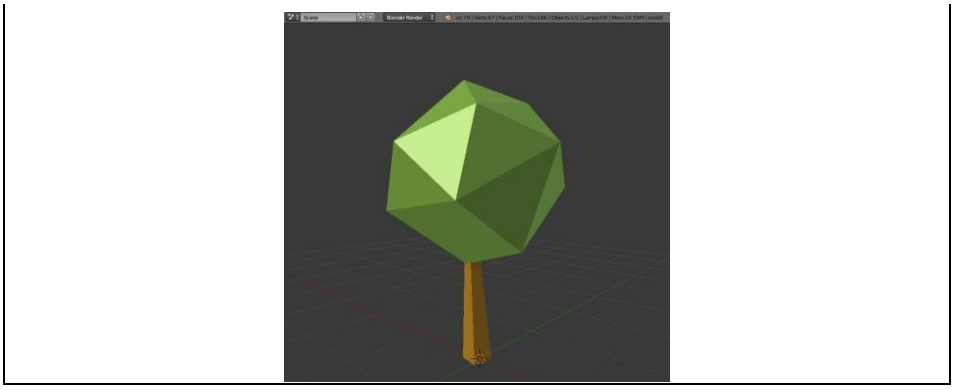
A fejlesztési munkálatok egyik része a grafikai kivitelezés volt. A 3D modellezéshez két különböző szoftvert használtunk fel: a MagicaVoxel-t kizárólag a városi jelenetben megtalálható házak építéséhez [26], az összes többi modell létrehozásához pedig a Blender szoftvert tartottuk a legalkalmasabbnak [27]. Továbbá szükség volt a projektben 2D grafikai elemekre is, ami a 3D modellek textúráit, illetve a UI elemeket jelenti. Ezeknek az elkészítéséhez a GIMP programot választottuk ki [28]. Általánosan elmondható mindegyikről, hogy ingyenes, megbízható, népszerű és jól dokumentált, ami lényegesen megkönnyítette a használatukat.

A 3D modellezési munkák első lépése egy absztrakt környezeti modell létrehozása volt. Ez azt jelentette, hogy szét kellett bontani a már adott 2D környezetet atomi elemekre. Az absztrakció eredményeként előállt egy egyértelmű lista azokról a 3D modellekről, amelyeket egyesével el kellett készíteni. Ez a lista egy rendkívül hatékony módja volt annak, hogy a modellezéssel kapcsolatos feladatainkat és a munkafolyamatot előre megtervezzük (3. táblázat).

Egy 3D alkalmazást nagyon sokféle különböző grafikai ábrázolási móddal el lehet készíteni. A mi szoftverünk esetében a low poly stílust találtuk a legmegfelelőbbnek több különböző okból kifolyólag is.

A hardver limitációja az egyik sarkalatos pont. Sok iskolában nem állnak rendelkezésre a legkorszerűbb hardverrel rendelkező számítógépek és egyéb IKT eszközök, így ezt mindenképpen figyelembe kellett vennünk. További hardverrel kapcsolatos korlátokat jelent az is, hogy a szoftvernek a korábbi terveinknek megfelelően mobil platformokon is elérhetőnek kell lennie. A low poly kétségtelenül egy hatékony stílus ebből a szempontból. Ugyanis az ilyen módon elkészített 3D modellek egy részletgazdag 3D modellhez képest jelentősen kevesebb poligonból épülnek fel. Kevesebb sokszög kevesebb számítást igényel a grafikus kártya részéről, ami hatékonyabb, gyorsabb végrehajtást és alacsonyabb rendszer követelményt igényel magától az applikációtól.

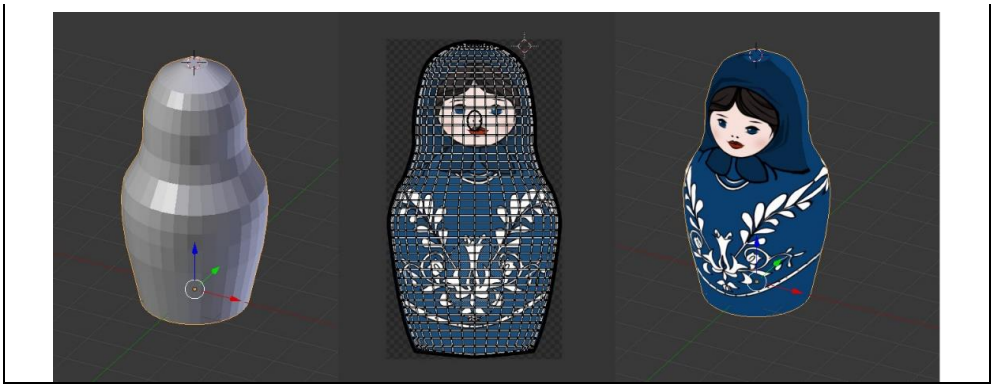
A második fontos a szempont a vizualizáció befogadásával kapcsolatos. Arra törekedtünk, hogy a videojátékok világából ismert játékosság érzetét alakítsunk ki a felhasználókban annak érdekében, hogy a Sprego tanulási folyamatának hatékonyságát növelni tudjuk. Mivel a low poly stílus mind a két elvárásunknak eleget tett, és ezen felül még rövidebb kivitelezési időt is igényel alternatíváinál, ezért végül a modelljeinket ilyen módon alkottuk meg (4. ábra).



4. ábra: Egy low poly fa 3D modellje Blender-ben.

A matrjoska babák és a település táblák modellezése egy teljesen más típusú munkafolyamatot és technikai megközelítést igényelt, hiszen textúrákat is kellett feszíteni a geometriai alakzatokra. A babák elkészítése komolyabb kihívást jelentett, így ezt a folyamatot fogjuk bemutatni.

A speciális modellek elkészítésének az első lépése egy olyan geometriai alakzat elkészítése volt, ami alkalmas avatárként működni az applikáció keretein belül úgy, hogy közben hasonlít a babák eredeti, a való életben megtalálható kinézetére is. Ezek után a következő feladat az volt, hogy a babák 3D alakzatát kicsomagoljuk (unwrapping) a síkba. Ezt követően meg kellett rajzolni az összes különböző színű textúrát GIMP-ben. A fényes, részletgazdag és színes textúrák biztosították, hogy az avatárok kellőképpen kitűnjenek majd a környezetükből az alkalmazásban, ezzel is biztosítva a minél jobb felhasználói élményt a tanulók számára. Az alakzatok kicsomagolása és a textúrák elkészítése után össze kellett rendelni a síkra kifeszített formát és a textúrákat, hogy azok megfelelően jelenjenek meg. Ezt a bemutatott módszert UV mapping-nek nevezik (5. ábra).



5. ábra: Textúra ráfeszítése egy geometriai alakzatra UV mapping módszerrel.

3.3. Programozási feladatok

A programozási feladatokat a két különböző környezet (jelenet) felépítésével kezdtük, amihez először az elkészített 3D modelleket kellet beimportálni a projektbe. Az építési folyamat atomi elemek, úgynevezett Game Objec-tek hierarchiájának felépítésén keresztül valósult meg. Ezekhez az objektumokhoz lehet hozzárendelni például olyan tulajdonságokat, hogy hol helyezkednek el a világon belül (a koordináta-rendszerben), milyen 3D modell tartozik hozzájuk, hat-e rájuk gravitáció stb.

A környezet kezdeti felépítése után egy finomhangolási folyamat következett, amely alatt számos beállítást megváltoztattunk a jelenetre vonatkozóan, hogy a végeredmény megfeleljen a megjelenéssel kapcsolatos elképzeléseinknek. Ez többek között a jelenetek világításának és az objektumok árnyékainak megváltoztatását, finomhangolását jelentette.

Ezeknek a munkálatoknak a kapcsán fontos megemlíteni a Unity-nek a Prefab nevű eszközét, ami az újrafelhasználható objektumok létrehozásáért felelős. Az ilyen speciális objektumok fő előnye, hogy amikor megváltoztatjuk a Prefab egyik példányának egy-egy attribútumát, akkor eldönthetjük, hogy ezek a változások vonatkozzanak-e az összes többi példányra is vagy sem.

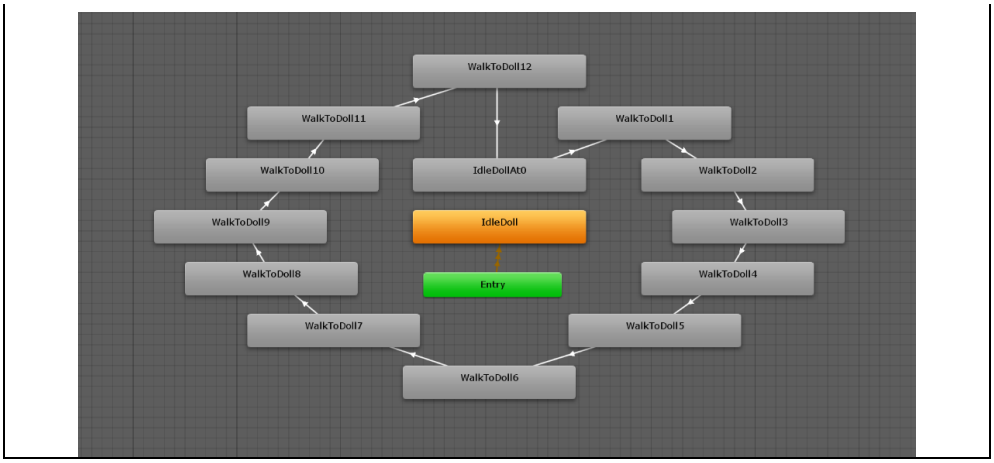
A jelenetek felépítése után az animációk implementálása volt a következő programozási feladat. Az animációval kapcsolatos munkafolyamat szemléltetésére az erdős jelenet (feltételes számlálás algoritmus) elkészítésén keresztül mutatjuk be. Az algoritmus animációja hasonlóképp működik, mint a már említett 2D alkalmazásban [20]. A babák egy erdőben mozognak a tábortűz körül, és az algoritmus megszámolja, hogy a felhasználó által kiválasztott színű babából hány darab van összesen. Az animáció előállításáért felelős folyamat azonban meglehetősen különbözik a két alkalmazásban, a két fejlesztői környezet, a Construct és a Unity3D eltérő természete és célja miatt. Unity-ben két adott eszközre van szükség az animációk implementálásához: animációs fájlokra és a Unity Animator komponensre.

Az animációs fájlok segítségével lehet képkocka alapú animációkat definiálni az adott jelenetben megtalálható objektumokra vonatkozóan. Ez specifikusan ennél a jelenetnél azt jelentette, hogy a 12 babához tartozóan 24 darab animációs fájlt kellett létrehozni, ugyanis a babák a tűz körül egy nagyobb és egy kisebb kör körül is tudnak mozogni.

Ezek után C# szkripteket és a Unity Animator komponenst használtuk arra, hogy irányítani tudjuk az animáció működését. Ezt az Animator komponenst elképzelhetjük úgy, mint egy véges automatát, mert a kettő között hasonlóság fedezhető fel a következő tekintetben:

- az állapotok tükrözik az animált objektum jelenlegi helyzetét (az állapot általában megegyezik egy animációval, amit egy animációs fájlban definiálunk),

- vannak speciális, kitüntetett szerepű állapotok (például: az animáció belépési pontja),
- az (állapot)átmenetek definiálhatók úgy, hogy megadjuk, hogy melyik animációból melyik másik animációba léphet az objektum, és milyen feltételek teljesítésével.



6. ábra: A matrjoska babák animálásáért felelős véges automata egy részlete.

Ezeknek az ismereteknek a tudatában egy olyan véges automatát terveztünk, amely képes irányítani a teljes animációs folyamatot az összes babára vonatkozóan (6. ábra). Ez a tervezési döntés később nagyon kifizetődőnek bizonyult, ugyanis így tökéletesen kontrollálható volt a teljes animáció. Ezen felül a teljes irányítás lehetővé tette azt is, hogy az animáció lejátszását bármikor szüneteltetni és folytatni tudjuk, ami elengedhetetlen volt az interaktív modulok beépítésekor a későbbiekben (7. ábra).

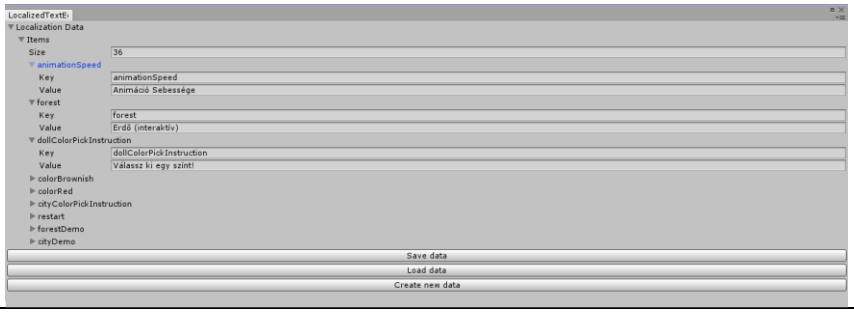
Az animációk előállítása után a fókusz a felhasználó-barát UI összeállítására irányult. A Unity3D egy kiterjeszthető UI rendszerrel rendelkezik, ami magában foglal számtalan előre legyártott ilyen elemet, például gombokat, csúszkákat, paneleket és bemeneti mezőket. Ezeknek nagy előnye, hogy a megjelenésük és a viselkedésük is teljesen testreszabható. Ennek következtében olyan 2D grafikai elemeket hoztunk létre GIMP-ben, amelyek beleillenek a prezentált környezetbe, ezzel is növelve az alkalmazás grafikai vonzerejét a tanulók számára (7. ábra).

A technikai részleteknél oda kellett figyelnünk arra is, hogy a program interaktív elemei és a felhasználói felület kompatibilisek legyenek a különböző bemeneti eszközökkel (egér, érintőképernyő). Továbbá implementáltunk egy extra világítást biztosító funkciót is, amely egy jelölőnégyzettel kapcsolható ki és be. Ennek oka az, hogy számos osztályteremben alkalmatlan körülmények merülhetnek fel a vetítéshez, például ha túl világos van és nincs lehetőség besötétíteni a termet. Ennek a funkciónak a beépítésével biztosítottuk azt, hogy az alkalmazásunk ilyen körülmények között is jól használható legyen az oktatásban.

7. ábra: Az erdős jelenet interaktív felhasználói felülete az alkalmazásban.

Fontos jellemzője a szoftverünknek, hogy a szöveges elemek támogatják a többnyelvű megjelenítést az alkalmazáson belül (az alkalmazás jelenleg a magyar és angol nyelven érthető el). Ehhez implementálni kellett egy lokalizációs rendszert, amelyben elérhetőek a megjeleníthető szövegek az összes támogatott nyelven. A rendszer tervezésénél az elsődleges szempontunk az volt, hogy a bővítés minél egyszerűbben megvalósítható legyen, amennyiben egy új szöveg vagy akár egy teljesen új nyelv bevezetésére kerülne a sor a jövőben. Ehhez a nyelvi szövegeket vezérlő rendszer JSON állományokat használ a kifejezések tárolására. A lefordítható kifejezések kulcs-érték párokként vannak eltárolva a JSON fájlokban, ahol a kulcs azonosítja a kifejezést az alkalmazáson belül, az érték pedig sztringként tárolja a kifejezést az adott nyelven. Annak érdekében, hogy az értékek könnyen szerkeszthetők és hozzáférhetők legyenek, fejlesztettünk egy saját Unity plugint, ami kezeli ezeknek a JSON állományoknak a betöltését, szerkesztését és a mentését (8. ábra).

Ezt követően implementáltunk a kezdőképernyőben egy legördülő listát, amely segítségével ki tudja választani a felhasználó az általa preferált nyelvet. A kiválasztást követően az alkalmazás futási időben betölti a fordításokat a JSON állományból és annak megfelelően egyből megváltoztatja a megjelenített szövegeket.



8. ábra: Többnyelvű kifejezések szerkesztése a saját Unity pluginunkban.

A programozási feladatok végső részét a felhasználói interaktivitás megvalósítása jelentette. Ezt két lépésben tudtuk megtenni. Először ki kellett találnunk azt, hogy hogyan is tudnánk egy olyan interaktivitást biztosítani a felhasználók számára, ami nem befolyásolja az algoritmus animációjának lejátszásának a sikerességét, de mégis érdekesebbé és hatékonyabbá teszi a folyamatot. A második

lépésben pedig fel kellett térképeznünk azt, hogy egy ilyen megoldást hogyan tudunk integrálni a meglévő rendszerekbe.

Első lépésként azzal az ötlettel álltunk elő, hogy a felhasználónak interaktív kérdéseket teszünk fel az animáció lejátszása során. Ezeket a kérdéseket párbeszédablakok formájában prezentáljuk a tanulóknak, akik egyszerű nyomógombokkal tudnak válaszolni a kérdésre (7. ábra). Amíg az adott párbeszédablak aktív, addig az animáció lejátszása szünetel, így a diákok is érzik, hogy a program futására befolyással vannak. A nyomógombok azért bizonyultak szerencsés választásnak, mert így a szoftver minden támogatott platformmal megőrizte a kompatibilitását, a bemeneti eszköztől függetlenül. Itt fontos megemlítenünk, hogy le kellett kezelnünk azt az esetet is, amikor a tanuló helytelen választ ad az egyszerű eldöntendő kérdésekre. Ekkor feldobunk egy „Biztos vagy benne?" üzenetet a diáknak, majd visszatérünk az eredeti kérdésre, ahol a felhasználónak újra lehetősége nyílik a kérdés megválaszolására. Ez oktatási környezetben kifejezetten előnyös lehet, ugyanis ilyenkor a tanárnak jut ideje arra, hogy a csoportos coaching módszerével újra átbeszéljék az algoritmus működésének részleteit.

A második lépésben kellett megvizsgálnunk az interaktivitás megvalósításához szükséges technikai hátteret. Ehhez az animáció különböző állapotai közötti átmeneteket kellett megszakítanunk, azáltal hogy az átmenetek közé beillesztettünk egy új fázist. Az új fázis alatt kellett láthatóvá tennünk az aktuális eldöntendő kérdést figyelve arra, hogy az animáció működése mindaddig szüneteljen, amíg a felhasználó nem adott elégséges választ a kérdésünkre. Az implementációhoz az Animator komponenseket, illetve specifikusan erre a célra megírt C# szkripteket használtunk.

4. Összefoglalás

Az általunk megépített alkalmazás célja egy olyan semi-unplugged 3D vizualizációs oktatóprogram elkészítése volt, ami hatékony módon segíti a Sprego programozás tanulási folyamatát.

A Sprego módszer egy alternatív megközelítése a tradicionális táblázatkezelői oktatási folyamatnak. Fő jellemzője, hogy a feladatmegoldáshoz az algoritmusépítésre, az összetett függvények és a tömbképletek használatára támaszkodik. Tagadhatatlan előnye a Sprego módszertannak, hogy fejleszti a tanulók számítógépes gondolkodását és algoritmikus készségét, ezáltal a diákok olyan tudáshoz jutnak hozzá, amely az informatika bármely területén hasznosítható.

A módszer tanításának hatékony támogatásához egy olyan látványos, színes low poly stílusú grafikai megoldást terveztünk, ami nemcsak felkelti a tanulók figyelmét, hanem hatékonyan működik limitált specifikációval rendelkező hardverek esetén is. Ezen felül biztosítjuk a különböző platformok közötti átjárhatóságot, hogy a szoftver minél szélesebb körben elérhető legyen (interaktív táblán, asztali számítógépen, okostelefonon). Továbbá beépítettük az interaktív lejátszás lehetőségét az egyes vizualizációkhoz kapcsolódóan.

Szoftverünk egy korábbi kutatás eredményeként elkészült programon alapszik, amelyben a szerzők egy olyan 2D vizualizációs applikációt készítettek, amely animációkon keresztül szemlélteti lépésről lépésre a különböző alapvető Sprego algoritmusokat [22]. Az alkalmazás legfontosabb jellemzői, hogy több platformon elérhető, matrjoska babákat használ avatárokként és a való életből vett, szemléletes példákkal operál.

A fő feladatunk az volt, hogy a feltételes számlálás és a lineáris keresés algoritmusának vizualizációját implementáljuk a már meglévő alkalmazásból kiindulva. Ehhez úgy kellett átültetnünk a környezetet, hogy a 2D megvalósítás fentebb említett fő előnyeit és aspektusait megtartjuk, miközben a 3D környezet által biztosított lehetőségek kihasználásával új funkciókat is implementálunk.

Az alkalmazás jövőjével kapcsolatban már vannak terveink és kitűzött céljaink. Az egyik, hogy szeretnénk új funkciókkal és új algoritmusok vizualizációjával bővíteni alkalmazásunkat. A nyelvi támogatás bővítését is tervezzük további fordítások hozzáadásával. Ezen túlmenően szeretnénk

minél több különböző forrásból visszajelzéseket gyűjteni annak érdekében, hogy megfelelő adataink legyenek az alkalmazás használatával kapcsolatban. A kapott adatokat felhasználva tudnánk biztosítani, hogy a szoftver evolúciója a megfelelő irányba menjen végbe. Tervezzük továbbá a Google Play online felületre is publikálni munkánkat a könnyű elérhetőség biztosításának érdekében. A legfontosabb célunk pedig a szoftver minél szélesebb oktatási környezetben történő kipróbálása és hatékonyságának vizsgálata kontrollált keretek között az általános és középiskolákban, valamint a felsőoktatásban egyaránt.

Irodalom

1. Gy. Polya: *How To Solve It. A New Aspect of Mathematical Method.* (2nd Edition 1957), Princeton University Press, Princeton, New Jersey. (1954)
2. J. J. G. Merriënboer, J. Sweller: Cognitive Load Theory and Complex Learning: Recent Deve-lopments and Future Directions. *Educational Psychology Review*, 17(2), 147–177. DOI= http://doi.org/10.1007/s10648-005-3951-0. (2005)
3. R. R. Skemp: *A matematikatanulás pszichológiája.* Edge 2000 Kiadó, Budapest. (1975)
4. D. Kahneman: *Thinking, Fast and Slow*. New York: Farrar, Straus; Giroux. (2011)
5. R. Panko: The Cognitive Science of Spreadsheet Errors: Why Thinking is Bad. *Proceedings of the 46th Hawaii International Conference on System Sciences*, January 7-10, 2013, Maui, Hawaii. (2013)
6. M. Csernoch: *Thinking Fast and Slow in Computer Problem Solving.* Journal of Software Engineering and Applications, (2017) 10(1). http://file.scirp.org/pdf/JSEA_2017012315324696.pdf. (utoljára megtekintve: 2017. 07.)
7. T. Bell, H. Newton: *Unplugging Computer Science.* Improving Computer Science Education, Routledge (2013).
8. P. Biró, M. Csernoch: *Unplugged tools for building algorithms with Sprego.* International Conference on Education and New Development, Lisbon, Portugal, (2017).
9. Csernoch Mária: *Programozás táblázatkezelő függvényekkel – Sprego*. Budapest, Műszaki Könyvkiadó (2014).
10. K. Sebestyén, G. Csapó: *Visualising Sprego Inequality Problems with 2D Representations.* The Turkish Online Journal of Educational Technology. Accepted (2018).
11. Microsoft: LOOKUP function (2018a). https://support.office.com/en-us/article/lookup-function-446d94af-663b-451d-8251-369d5e3864cb. (utoljára megtekintve: 2018. 08.)
12. Microsoft: HLOOKUP function (2018b). https://support.office.com/en-us/article/hlookup-function-a3034eec-b719-4ba3-bb65-e1ad662ed95f. (utoljára megtekintve: 2018. 08.)
13. Microsoft: VLOOKUP function (2018c). https://support.office.com/en-us/article/vlookup-function-0bbc8083-26fe-4963-8ab8-93a18ad188a1. (utoljára megtekintve: 2018. 08.)
14. Microsoft: INDEX function (2018d). https://support.office.com/en-us/article/index-function-a5dcf0dd-996d-40a4-a822-b56b061328bd. (utoljára megtekintve: 2018. 08.)
15. Microsoft: MATCH function (2018e). https://support.office.com/en-us/article/match-function-e8dffd45-c762-47d6-bf89-533f4a37673a. (utoljára megtekintve: 2018. 08.)
16. J. Walkenbach: *Microsoft Excel 2010 Bible*. Wiley Publishing, Inc. Indianapolis, (2010) 202
17. K. Freiermuth, J. Hromkovič, B. Steffen: *Creating and Testing Textbooks for Secondary Schools.* In: R. T. Mittermeir, M. M. Sysło (Eds.), Informatics Education - Supporting Computational Thinking Berlin Heidelberg, Germany: Springer (2008) 216–228 http://link.springer.com/chapter/10.1007/978-3-540-69924-8_20, pp. 219. (utoljára megtekintve: 2016. 01.)

18. M. Csernoch, P. Biró, K. Abari, J. Máth: *Programozásorientált táblázatkezelői függvények.* XIV. Országos Neveléstudományi Konferencia, Debrecen, Hungary, (2014).
19. OFI: *Informatika kerettanterv.* (2013)
http://kerettanterv.ofi.hu/02_melleklet_5-8/2.2.15_informat_5-8.doc. (utoljára módosítva: 2018.08.)
20. G. Csapó, K. Sebestyén: *Educational Software for the Sprego Method.* The Turkish Online Journal of Educational Technology, INTE 2017 October, (2017) 986-999
http://www.tojet.net/special/2017_10_1.pdf. ISSN 2146-7242 (utoljára megtekintve: 2018. 07.)
21. G. Csapó: *Sprego Virtual Collaboration Space.* 8th IEEE International Conference on Cognitive Infocommunications, (2017) 137-142
http://ieeexplore.ieee.org/document/8268230/. ISBN 978-1-5386-1264-4.
DOI=10.1109/CogInfoCom.2017.8268230 (utoljára megtekintve: 2018. 01.)
22. G. Csapó, K. Sebestyén: *Sprego – Spreadsheet Lego* (2018).
https://play.google.com/store/apps/details?id=hu.sprego.oktatoprogram. (utoljára megtekintve: 2018. 05.)
23. Unity Technologies: *Unity - The world's leading content-creation engine* (2018).
https://unity3d.com/unity. (utoljára megtekintve: 2018. 08.)
24. Unity Documentation: *Unity Scripting API* (2018).
https://docs.unity3d.com/ScriptReference/index.html. (utoljára megtekintve: 2018. 08.)
25. Unity Forums: *Unity Forum topics* (2018).
https://forum.unity.com/. (utoljára megtekintve: 2018. 08.)
26. MagicaVoxel: *Magicavoxel by ephtracy* (2018).
https://ephtracy.github.io/. (utoljára megtekintve: 2018. 08.)
27. Blender Foundation: *Free and Open 3D Creation Software* (2018).
https://www.blender.org/. (utoljára megtekintve: 2018. 08.)
28. GIMP: *About GIMP* (2018).
https://www.gimp.org/about/. (utoljára megtekintve: 2018. 08.)

Rekurzió tanítása LOGO-ban programozási fogalmak előkészítésével

Erdősné Németh Ágnes

erdosne@blg.hu
Batthyány Lajos Gimnázium, Nagykanizsa
ELTE IK Doktori Iskola

Absztrakt. A rekurzió erőteljes eszköz, de a legtöbb diák és tanár egyetért abban, hogy nehéz megtanulni, megérteni és tanítani. A LOGO ábrákon keresztül, a rekurzió vizualizálásával könnyebben felfedeztethetővé és megérthetővé válik ez a nehéz koncepció. A cikk a felső tagozatosok tanításakor alkalmazható felépítést mutatja be. A számítógépes gondolkodás elvei alapján felépítve a későbbi, programozással kapcsolatos fogalmak megalapozását vetíti előre.

Kulcsszavak: LOGO tanítása, rekurzió, számítógépes gondolkodás, felsőtagozat, LOGO verseny

1. Áttekintés

A rekurzió a számítástudomány egyik alaptétele, erőteljes eszköz bizonyos problématípusok megoldásakor [2]. A diákok a kezdő programozás kurzusokon és a matematikai tanulmányaik során is találkoznak vele. Nehéz, mély gondolat és ugyanakkor absztrakt, nehezen magyarázható és nehezen határozható meg [3]. Sokan egyetértenek azzal, hogy bár erőteljes és jelentős elv, nehéz megérteni és megtanulni [4]. Úgy tűnik, hogy a taníthatóság nehézségeit minden iskolai szinten tapasztalták már [8].

A programozás tankönyvekben három fő példát találunk a rekurzióra: a Hanoi tornyait, a faktoriális számítását és a Fibonacci sorozatot [10]. Ezek a példák erőteljesek, de nem elég kifejezők az általános iskolások számára, akiknek manipulációra és/vagy vizualizációra lenne szükségük a megértéshez. Ezeken kívül a legtöbb programozás tankönyv ciklusos példákat mutat be, a valódi rekurzív példák helyett.

Hromkowitz tankönyvében [1] a rekurzió LOGO-val történő tanítása egy egyszerű definícióval kezdődik: „A rekurzió az ismétlődő elemek önmagukhoz hasonló módon történő feldolgozása." Az első példa egy végtelen, paraméterek nélküli rekurzió, mely nem csinál semmit, mivel az első utasítás már maga a rekurzív hívás. Ezután különböző szöggel és hosszúsággal rajzolt spirálok variációi következnek, egyre növekvő paraméterekkel. Csak ezután mutat példát a feltételes megállításra, majd elemzi a hívások mélységét. Ezeket az elméleti példákat követik a hagyományos feladatok: a zárójelezés helyessége, a Koch-görbe, a Sierpinski-háromszög, fák és hópihék. Ez az elméleti felépítés felnőttek számára megfelelhet, de gyerekek számára a LOGO vizualitásának kihasználása nélkül elég használhatatlan.

2. Rekurzió tanítása LOGO-val

Gimnáziumunkban ötödiktől kezdve a tanórákon a LOGO programozási nyelv segítségével gondolkodásfejlesztés és problémamegoldás zajlik: analizálás, elemekre bontás és újraépítés, az algoritmikus gondolkodás megismerése. A LOGO-ban természetes módon elérhető vizualizáció, a hétköznapi élettel való kapcsolat, a sokféle példa segít a megértésben, növeli a motivációt, segíti a kon-

cepciók kialakulását és erős alapot nyújt a további tanulmányokhoz. Órákon a legfontosabb a számítógépes gondolkodás erősítése, míg a LOGO szakkörökön a későbbi programozási tanulmányok előkészítése is zajlik.

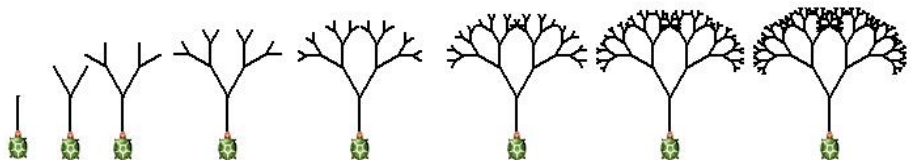
A rekurziók tanítására – a számítógépes gondolkodást szem előtt tartva – 11-12 éves kortól kerülhet sor. A rekurzió tanulása előtt a gyerekeknek nagy biztonsággal kell tudniuk kezelni a fejlesztőfelületet, ismerniük kell az előre, hátra, jobbra, balra alaputasításokat, az elágazás fogalmát és használatát, az ismétléses ciklus fogalmát és kezelését. Kell tudniuk eljárásokat írni és azokat paraméterezniük. Ezek biztos használata után kerülhet sor erre a nem könnyű gondolatra, a rajzokon keresztül viszont – tapasztalataim alapján - könnyedén megérthető és alkalmazható alapismeretre. A LOGO programozási nyelv egy természetes eszköz a rekurzió tanítására, jól működik a képekben a kép részeinek újrafelfedezése.

A cikkben a rekurzió tanításának egy, a számítógépes gondolkodást és a későbbi tanulmányok előkészítését célzó, tudatos felépítését írom le. Természetesen az egyes lépések megértése után mélyíteni és rögzíteni kell az épp megtanult ismereteket, sok-sok feladaton keresztül.

Az ábrák alapjai LOGO versenyfeladatokból származnak, saját megvalósításban jelentek meg az ISSEP 2015 konferencia kiadványában, a doktori dolgozatomban és itt is.

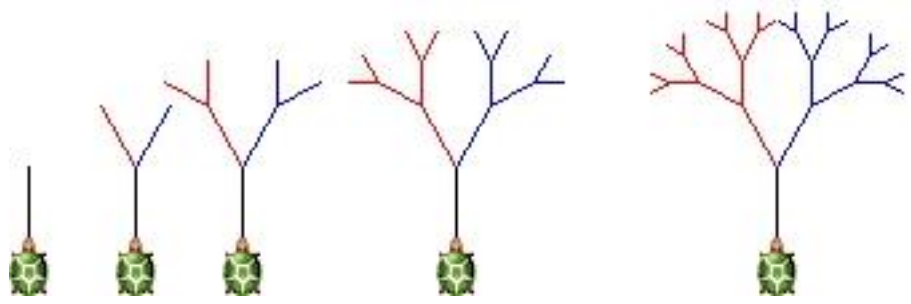
3. A rekurzív lépés megértése

Az ismerkedést a bináris fákkal kezdhetjük. A példa a valós életből való, a gyerekek emlékeznek arra, hogyan nő évről-évre egy fa – ezt először egy, a növekedést bemutató programmal szemléltethetjük. (1. ábra)



4. ábra Bináris fa növekedése

A következő kép (2. ábra) a módszer bemutatására szolgál, érdemes előre elkészíteni és megbeszélni a rajzot, megfogalmaztatni a következőket: „A fa piros része pont ugyanolyan, mint a teljes fa egy évvel korábbi állapota. A kék rész is az egy évvel fiatalabb teljes fa, csak más irányban nőtt, de ugyanazon a helyen, mint a piros. Rajzolj egy egyenest, fordulj balra, rajzolj egy évvel fiatalabb fát, fordulj jobbra, rajzolj újra egy évvel fiatalabbat, fordulj vissza balra és tolass vissza a teknőccel a kiindulási helyzetbe!”



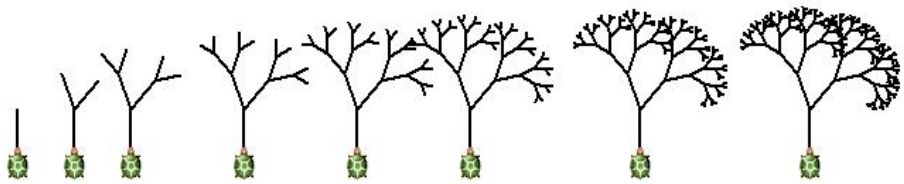
5. ábra Bináris fa színesen, magyarázathoz

A gyerekeknek saját szavaikkal, egyszerűen, maguknak kell megfogalmazniuk a szigorú csökkenést és az állapotátlátszóság követelményét. A rekurzív mintát nekik kell részekre bontani, amiben

újra felfedezik az eredeti mintát, kisebb verzióban. Az ismerkedés során fel kell velük fedeztetni a rekurzió alapalgoritmusát:

```
mintarajzolás
    alapelem megrajzolása
    mozgás/fordulás
    kisebb állapotú minta rajzolása
```

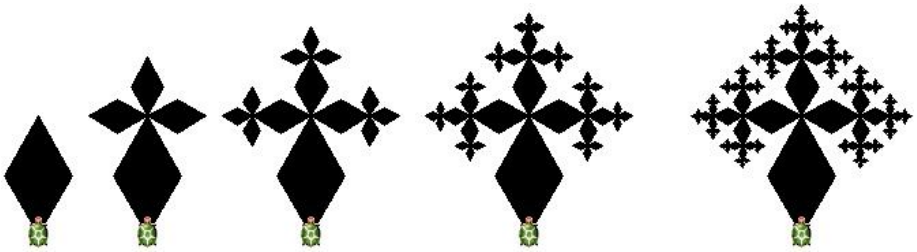
A vizualizálás segít megfogalmazni a rekurzív minták elvét. Az eljárás megfogalmazása után LOGO-ban implementálni tudják a rajzolás folyamatát. A megértést különböző számú ágakkal, szélfútta fára hasonlító más szögű forgásos fákkal, illetve ezek kombinációjával tudjuk ellenőrizni. Például a 3. ábrán látható fával.



6. ábra Szélfútta bináris fa

4. Kiinduló állapot

Az első mintában a kiinduló állapot egy nagyon egyszerű alakzat volt, egy szakasz. Így a kiinduló lépésben csak előre és hátra utasításokat kellett használni. Következő gondolati elem a kiinduló állapot bonyolítása, például rombuszrajzolással. (4. ábra)



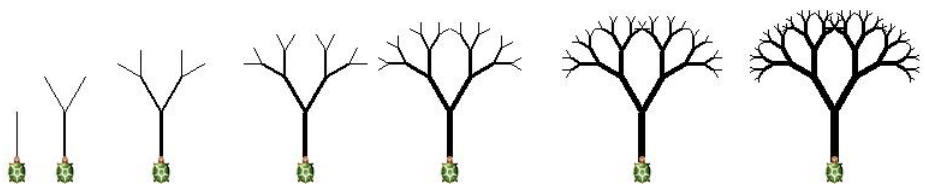
7. ábra Növekvő gyémánt

Ebben a mintában már kilépési feltételt is meg kell fogalmazni és alapállapotként egy trapézt kell rajzolnia a teknőcnek. Azaz fel kell ismerni és meg kell fogalmazni a rekurzív eljárások módosított sablonját:

```
alap_eljárás
    alapállapot esete
    rekurzív eset
```

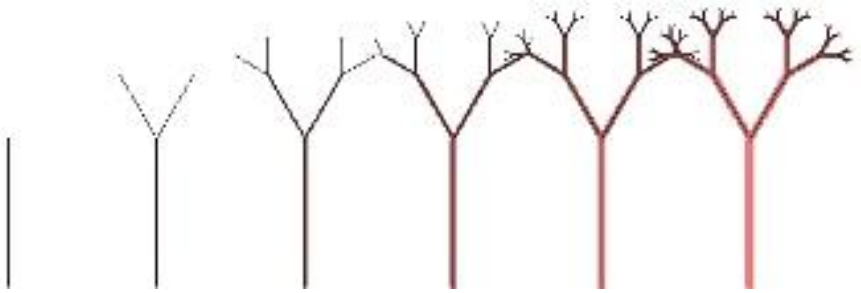
5. Szintek

A rekurzió végrehajtásának folyamán minden állapotban tudnunk kell, hogy épp hányadik rekurziós szinten vagyunk. A következő két példában a szintre vonatkozó információt kell felhasználni. Sokkal jobban hasonlít a fa az igazi növekedésre, ha az évek számával arányos az ág vastagsága. (5. ábra)



8. ábra Vékonyodó bináris fa

Minden lépésben megvizsgáltathatjuk a verem állapotát és akár ki is írathatjuk a rekurzió szintjét. Vizuálisan színezésre és a fa vastagságának megadására használhatjuk ezeket az információkat. (6. ábra)



9. ábra Színes bináris fa

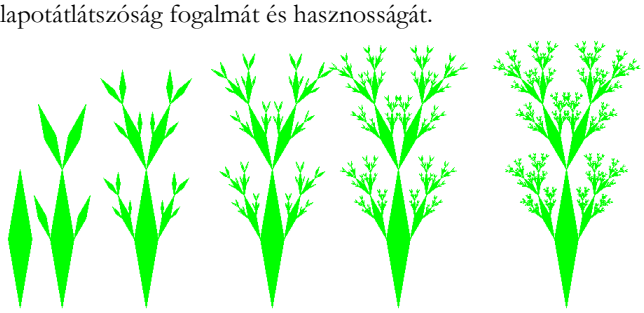
Eközben egy nagyon fontos új fogalommal is megismerkedhetnek a diákok, a verem fogalmával – itt még csak gyakorlati megvalósításban.

Ezután – akár megszakítva a rekurziókkal kapcsolatos ismereteket – kerülhet sor a funkcionális programozás alapjainak, ezen belül a listakezelés megértésére.

6. Állapotátlátszóság

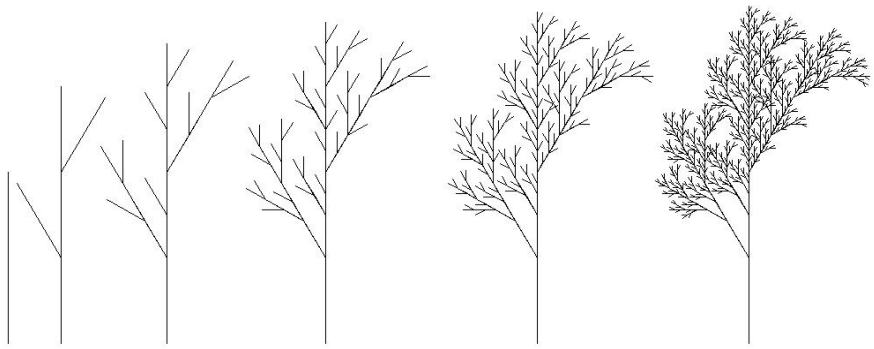
A rekurziók felfedezésében a következő lépés az állapotátlátszóság elvének pontosítása. A rekurzió nagyon rossz ábrákat adhat, ha nem tisztázzuk előre, hogy a teknőc hol van és milyen irányban áll a rekurzív lépés meghívása előtt és után.

A következő ábrák rajzolásakor a rekurziós lépések meghívása előtt a teknőcnek bonyolult mozgást kell tennie. (7. ábra, 8. ábra) Ezek és hasonló ábrák rajzolása közben, a gyakorlati megvalósítás során értik meg az állapotátlátszóság fogalmát és hasznosságát.



10. ábra Növekvő kaktusz

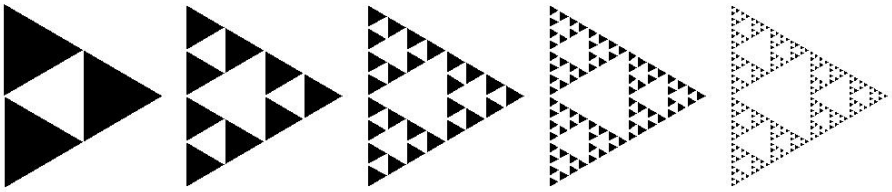
A megvalósítás során a kulcs és az első lépés a helyzetek és irányok tisztázása, az alapábra megtalálása. Pontosítani kell a rekurzió mélységét is az egyes esetekben. Csak ezek után lehet nekilátni a teljes eljárás kódolásának.



11. ábra Szélfútta fa variáció

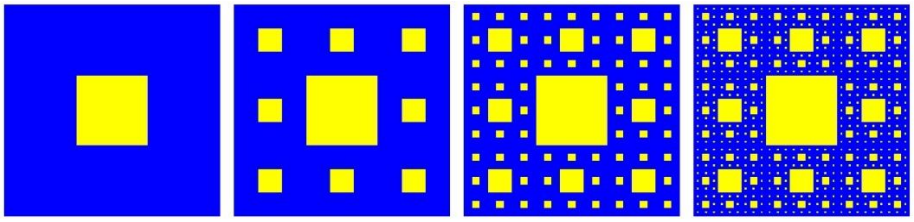
7. Az adatszerkezetek határai

A következő két klasszikus rekurzív mintán a rekurziós hívások számának korlátait tudják a diákok megtapasztalni. A Sierpinski-háromszöget (9. ábra) és a Sierpinski-szőnyeget (10. ábra) többféleképpen is lehet kódolni, attól függően, hogy a színes háromszögeket rajzoljuk meg vagy egy alapállapot színes ábráján törléssel hozzuk létre a mintát.



12. ábra Sierpinski-háromszög

Ezeken az ábrákon lehet beszélgetni a gyerekekkel az adatszerkezetek véges voltáról is. Ha a rekurziós hívások száma nagyobb, mint 5-6, akkor az ábrán már semmilyen változás nem észlelhető, hiszen a következő lépésben rajzolandó rész már túlságosan kicsi. A rajzolandó négyzet/háromszög mérete hamar kisebb lesz, mint egy pixel a képernyőn, így megjeleníthetetlen. Tipikus ötlet ennek áthidalására a gyerekek részéről, hogy akkor legyen az első ábra sokkal nagyobb – kipróbálással megtapasztalhatják, hogy ez maximum egy szinttel bővíti a lehetőséget.



13. ábra Sierpinski-szőnyeg

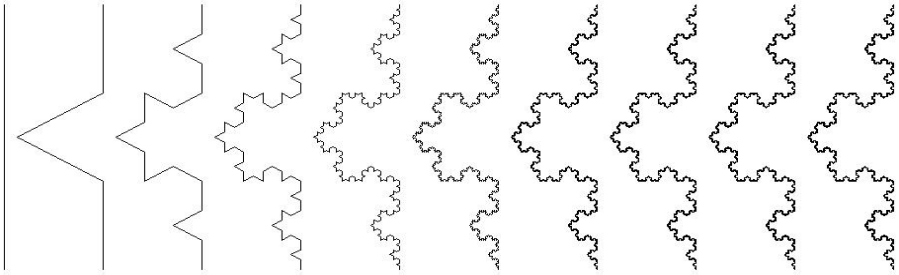
Elérve a megjelenítési határt, a számolás megy tovább, de a teknőc gyors forgásán kívül semmi változást nem látunk a képernyőn. Könnyen kiszámolható az egyes esetekben az a határ, amikor a rajz eltűnik ($h/3^n<1$).

A számítógépes gondolkodás tanításában ez egy nagyon fontos lépés – lenyűgöző lehetőség az adatstruktúrák végességének vizualizálására – ne hagyjuk ki! A későbbi tanulmányokban, amikor a

különböző adatszerkezetek (egész, hosszú egész, valós, szöveg) határairól tanulnak, könnyen tudunk ide visszahivatkozni.

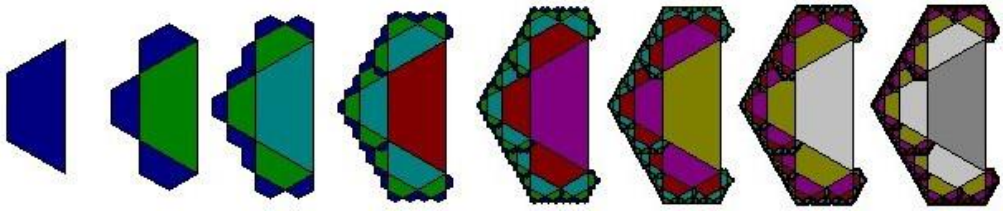
8. Futási idő vizsgálata

A Koch-görbe (11. ábra) megrajzoltatása az exponenciális futási idő szemléltetésére alkalmas. A diákok azt gondolják, a számítógépek hihetetlenül gyorsak, bármit képesek kiszámolni egy szempillantás alatt. Az első néhány szint még elég gyorsan megy, de a 6. szinttől a rajzolás egyre lassabb lesz. Láthatjuk, hogy a teknőc hihetetlen gyorsan forog és számol. A 10. szint megrajzolása már perceket vesz igénybe. Érdemes mérni a futási időt, feljegyezni és megtapasztalni a futási idő növekedését.



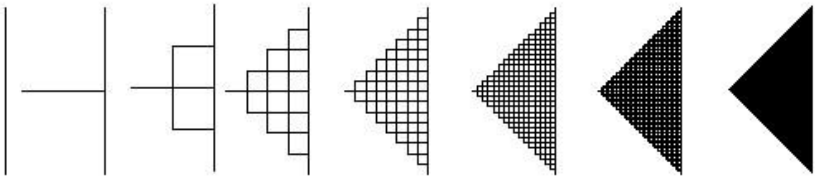
14. ábra Koch görbe

A trapézos feladaton (12. ábra) jól lehet gyakorolni az összes eddigi ismeretet, s újra megtapasztalni a futási idő és az adatszerkezet határait.



15. ábra Trapézok

Egyenesekből álló, egyszerű alakzattal (13. ábra) is lehet a területet teljesen lefedő alakzatot generálni, néhány lépéssel.

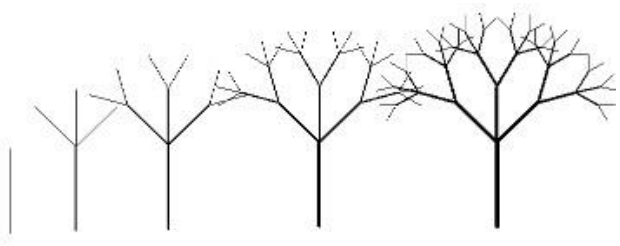


16. ábra Területlefedő alakzat

9. Egymást hívó rekurziók

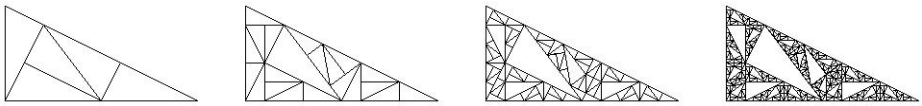
A rekurziók következő kognitív szintje, amikor egyik eljárás hívja a másikat, majd a másik visszahívja az elsőt. A LOGO vizualitásával ez az elv könnyen látvánnyá tehető és így könnyebben megérthető.

Első példaként újra visszatérhetünk a legegyszerűbb ábrához, a jól ismert, sokféle variációban már programozott fához. Ebben még akár egy elágazással meg tudjuk oldani a kétféle ábrát (14. ábra), az elvi lépés ezután következhet.

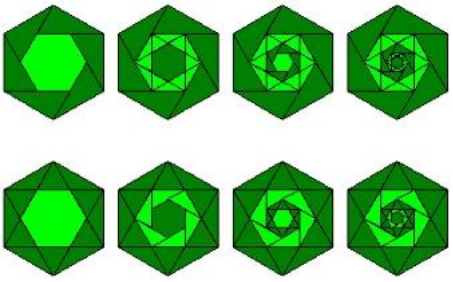


17. ábra Felváltva kettő-három fa

A következő bonyolultabb ábra (15. ábra) gondos tanulmányozásával a gyerekek felfedezhetik, hogy a háromszögek némely esetekben balsodrásúak, míg máskor jobbsodrásúak. A jobbsodrású háromszögek belsejében három jobbsodrású és egy balsodrású, míg a balsodrásúak belsejében három balsodrású és egy jobbsodrású ábra van – ennek megvalósításához már két eljárást kell írni, amik egymást hívják.



18. ábra Egymást hívó háromszög eljárások



19. ábra Hatszögek – szimultán rekurzió

A hatszöges példában (16. ábra) egy újabb paramétert is be kell vezetni annak tárolására, hogy melyik fajta eljárással kezdődjön a szimultán rekurzió.

Az egymást hívó eljárásokat – tapasztalatom szerint – már csak a kiemelkedő képességű diákok tudják önállóan lekódolni. A többiekkel elegendő addig eljutni, hogy megértsék és saját szavaikkal meg tudják fogalmazni, hogyan lehet ezeket az ábrákat megrajzolni.

10. Összefoglalás

A cikkben tanári szempontból mutattam be a rekurzió alapfogalmainak bevezetését és a további programozási ismeretek előkészítését. A legfontosabb, hogy tudatosan és egyszerűen, a gyerekek által megfogalmazva, a saját szintjükön legyenek kimondva a rekurzióval kapcsolatos fogalmak. Ha ezek megvannak, akkor később, amikor vizualizáció nélkül kerül elő a felülről-lefele történő építkezés elve, majd ebből a tudásból kiindulva lehet megfogalmazni a memorizálást s aztán áttérni a dinamikus programozás lentről-felfele építkezési elvére.

A megértést ellenőrizni és az elv használatának gyakorlását sok-sok további feladattal lehet. Ehhez elég sok példát találhatunk a LOGO versenyfeladatok gyűjteményeiben. A tapasztalatom az, hogy a gyerekek nagyon kreatívak abban is, hogy új feladatokat találjanak ki, illetve egy-egy elrontott feladatmegoldásból is lehet újabb ötleteket meríteni.

Irodalom

1. Hromkovic, J.: *Einführung in die Programmierung mit Logo: Lehrbuch Informatik*, Vieweg+Taubner GWv Fachverlage GmbH, Wiesbaden, 2010
2. Kalas, I., Blaho, A.: *I Beg Your Pardon Turtles: Don't Forget About Data Structures*, Eurologo'97 Proceedings, Budapest
3. Levy, D.: *Classification, Discussion, Recursion: Helping the Development of Computer-Science Concepts*, Eurologo'97 Proceedings, Budapest
4. Leron, U.: *What makes recursion hard*, Proceedings of the Sixth International Congress on Mathematics Education (ICME6), 1988, Budapest, Hungary
5. Murnane, J.: *To iterate or to recurse?* Computers & Education, Volume 19/4, 1992, pp. 387–394.
6. Er, M. C.: *On the complexity of recursion in problem-solving*, International Journal of Man-Machine Studies, Volume 20/6, 1984, pp. 537–544.
7. Wilcocks, D., Sanders, I.: *Animating recursion as an aid to instruction*, Computers & Education, Volume 23/3, 1994, pp. 221–226.
8. Levy, D.: *Collaborative Conceptual Change: The Case of Recursion*, Journal of Intelligent Systems 01/2002; 12(2)
9. Close, J., Dicheva, D.: *Misconceptions in Recursion: Diagnostic Teaching*, Eurologo'97 Proceedings, Budapest
10. Wirth, M. A.: *The far side of recursion*, Teaching mathematics and computer science, 2015/1, pp. 57-71.
11. Heizlerné Bakonyi, V., Zsakó, L.: *Strategy of guessing exercises – Variations of drawing trees*, Proceedings of the 9th International Conference on Applied Informatics Eger, 2014. Vol. 1., pp. 285–294.
12. Wing J. M.: *Computational thinking*, Communications of the ACM, 49(3), p. 33-35, 2006. https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf
13. Rónai O. R., Vörösss V.: *Lógós ecsetvonások*, NJSZT 2008, http://tetlabor.inf.elte.hu/logosecsetvonasok/lecke6.html & lecke7.html
14. *LOGO versenyfeladatok tára 1998-2002*, Mészáros Tamásné, Zsakó László, NJSZT, 2002, http://logo.inf.elte.hu/peldatar/LogoPeldatar1998_2002.pdf
15. *LOGO versenyfeladatok tára 2003-2007*, Heizlerné Bakonyi Viktória, Zsakó László, NJSZT, 2008, http://logo.inf.elte.hu/peldatar/Logo2003_2008.pdf
16. *LOGO versenyfeladatok tára 2008-2012*, Heizlerné Bakonyi Viktória, Zsakó László, NJSZT, 2013, http://logo.inf.elte.hu/peldatar/LogoPeldatar2008_2012.pdf
17. *NJSZT Logo Országos Számítástechnikai Tanulmányi Verseny archívum*, http://logo.inf.elte.hu/logo_archivum.html
18. Erdősné Németh Á.: *Introducing recursion with LOGO in upper primary school*, ISSEP Proceedings 2015, Ljubljana, pp. 121-129.

Tapasztalatok a programozási feladatok visszavezetéssel történő megoldását támogató osztály-sablon könyvtár használatáról

Gregorics Tibor[1], Nagy András[2]

[1]gt@inf.elte.hu, [2]nagyandras95@inf.elte.hu
ELTE IK

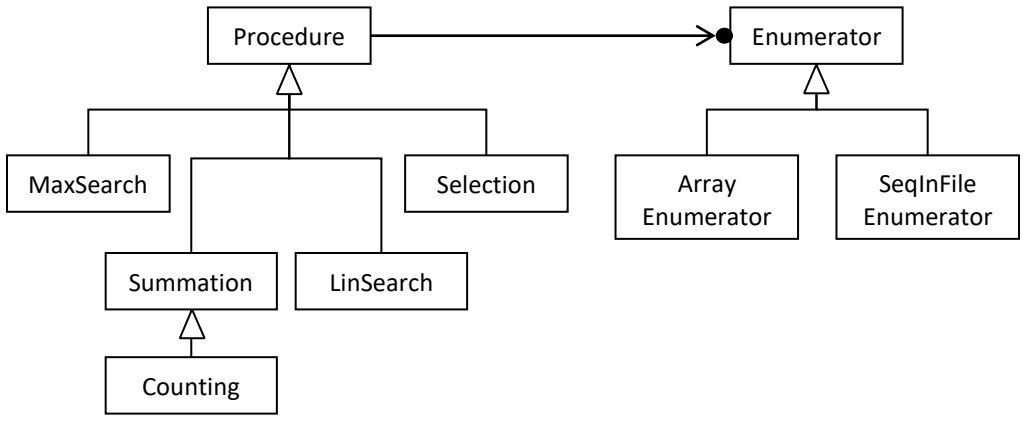
Absztrakt. Eltelt néhány év azóta, hogy az ELTE IK PTI BSc szakon a programozás oktatás része lett annak az újrahasznosítható osztály-sablon könyvtárnak az ismertetése és felhasználása, amely a programozási tételekre történő visszavezetéssel tervezett programok C++ nyelvű megvalósítását támogatja. Ebben a cikkben az oktatásba történő bevezetéssel kapcsolatos tapasztalatokról számolunk be, és bemutatjuk az ezek hatására módosított osztály-sablon könyvtárban található újításokat.

Kulcsszavak és kifejezések: visszavezetés, programozási tétel, felsoroló, objektum orientált programozás

1. Bevezetés

2009-ben került bevezetésre az ELTE IK programozó informatikus szak objektum-elvű alkalmazások fejlesztése kurzusának tananyagába egy olyan osztály-sablon könyvtár, amelynek segítségével a programozási tételekre visszavezethető feladatok C++ nyelvű megoldását lehet elkészíteni. A könyvtár és annak használata elsősorban objektum-orientált technológiára épült [1], de sablon-paramétereket is használt.

A könyvtárban három féle osztályt találunk: a felsorolót használó programozási tételeket [2] általánosan leíró osztály-sablonokat és ezek ősosztályát (*Procedure*); néhány nevezetes felsorolást [4] definiáló osztályt és ezek ősosztályát (*Enumerator*), valamint két segédosztály a maximum keresés tételének maximumot, illetve minimumot kereső változatainak előállításához. [3]



1. ábra. Programozási tételek osztály-sablon kód-könyvtára

Nemcsak a könyvtár felhasználása épül objektum-orientált technológiára, de maga a könyvtár is ennek szellemében készült. A *Procedure* osztály-sablon az összes programozási tétel őse, amely egy általános feldolgozási stratégiát fogalmaz meg: nevezetesen, végig megy egy felsoroló (erre mutat az *enor* pointer) által előállított elemeken, hogy azokat feldolgozza.

```
init();
for (enor->first(); !enor->end(); enor->next()) {
    body(enor->current());
}
```

Ezt a feldolgozást a *Procedure* ősosztály *run()* metódusa tartalmazza, és végsősoron ezt hajtja végre minden programozási tételre visszavezetett megoldás is. A végrehajtás attól válik egyedivé, hogy az egyes programozási tételeket leíró osztályok egyedi módon implementálják (felülírják) az *init()* és a *body()* virtuális metódusokat. Mindeközben újabb, a konkrét tételre jellemző virtuális metódusokat vezetnek be, amelyeket a konkrét alkalmazásoknál kell/lehet felüldefiniálni.

Egy programozási tételre visszavezethető feladat megoldása ezen könyvtár megfelelő programozási tételt leíró osztály-sablonjának felhasználásával készül. Előbb megadjuk ennek sablon-paramétereit helyettesítő értékeket, majd egy egyedi osztályt származtatunk belőle, és az így nyert osztály objektum-példányára meghívott (öröklött) *run()* metódus oldja meg a feladatot. Ennek a tevékenység-objektumnak az egyedi viselkedését egyrészt (fordítási időben) a származtatás során felülírt virtuális metódusok biztosítják, másrészt annak a speciális felsoroló objektumnak hozzáadása (futási időben) az (öröklött) *addEnumerator()* metódus segítségével (függőség befecskendezés), amelyik a megoldás során feldolgozandó elemeket sorolja fel.

A felsorolókat [2] [3] leíró osztályok őse egy interfészsablon (*Enumerator*), egy teljesen absztrakt osztály, amely bevezeti, de nem definiálja a felsoroló műveleteket. Ebből származnak a konkrét felsorolók osztályai, amelyek már reprezentációt és implementációt is tartalmaznak.

Ebben a cikkben ennek az osztály-sablon könyvtárnak a bevezetésével kapcsolatos tapasztalatokat gyűjtöttük össze, valamint a negatív kritikák nyomán végrehajtott módosításokat, amelyekkel sikerült megújítani a könyvtárat.

2. Korábbi verzióval kapcsolatos tapasztalatok

Az osztály-sablon könyvtárnak a bevezetése a programozás egyetemi tanításába az eredetileg kitűzött célon túl, miszerint az objektum-orientált programozást gyakoroltató újrafelhasználható könyvtárra volt szükség, több előnnyel is járt.

1. Nagyszámú olyan feladatot tudunk megfogalmazni, amelyek a könyvtár segítségével megoldhatók. Hosszú listája született a gyakorló feladatoknak, házi feladatoknak és zárthelyi feladatoknak. A könyvtár igen robosztus lett. Még a nem kifejezetten programozási tételekre történő visszavezetéssel megoldható feladatok is megoldhatók a segítségével (pl. láncolt lista felépítése a dinamikus memóriában, láncolt lista elemeinek feldolgozása).
2. A könyvtár használata bemutatja az objektum-orientált technológiákat, mint az egységbe zárást, az elrejtést, az öröklődést, a futási idejű polimorfizmust, függőség befecskendezést, de jó példákat szolgáltat a tervezési mintákra is. Ezen kívül a sablonok (*template*) hasz**nálatával is megismertet. Mivel maga a könyvtár is osztály-hierarchiára épül, jól** megértethető vele az objektum-orientált újrafelhasználási technika is, valamint a megvalósításhoz szükséges C++ nyelvi elemek (*class*, *virtual*, *protected*, …) is.

3. A könyvtár használata elmélyíti a programozási tételekre történő visszavezetés módszerének megértését és alkalmazását, azaz azt, amikor a programozási tétel nem csak egy laza ajánlás az algoritmikus gondolkodás során létrehozandó kódhoz, hanem egy sablon, amelynek kitöltésre váró paraméterei vannak, és ezáltal a programhelyességet garantáló módszer. Ezzel kapcsolatban mindenképpen pozitív változást hozott, hogy a 2017-es tantervváltozás során közös tantárgyba került a visszavezetéses programozási módszer és az objektum elvű programozás tanítása, és ezekkel együtt az osztály-sablon könyvtár használata.
4. A könyvtárra támaszkodó programozással sikerült rámutatni az elemzés és tervezés fontosságára már az egyszerű programozási feladatok megoldásánál is. Kikényszeríti, hogy egy feladat megoldásánál ne azon gondolkodjon a hallgató, hogy hol írjon ciklust, hogyan olvassa be a bemenő adatokat, hanem azon, hogy milyen programozási tétellel oldható meg a feladat, ehhez milyen felsorolást kell biztosítani, melyek lesznek a programozási tétel paraméterei, miközben az újrafelhasználható elemek implementálásával és tesztelésével nem kell foglalkoznia.
5. A könyvtárra támaszkodva szép megoldásokat tudunk előállítani. Programozói szemmel például nagyon érdekes, hogy az előállított megoldásokban mindössze egyetlen ciklus lesz, mégpedig a programozási tételek ősosztály-sablonjának *run()* metódusában. Ez kerül felhasználásra mindenféle paraméterezés mellett, a saját kódban pedig egyáltalán nem kell ciklust írni.

Természetesen a könyvtár oktatásba történő bevezetése negatív kritikákat is kapott.

1. A hallgatóknak nem tetszett, hogy rájuk kényszerítünk egy számukra idegen kódolási stílust (például, hogy ciklust egyáltalán nem írhatnak), amelyet az alkotói szabadságukba történő beavatkozásként éltek meg.
2. Gyakran hangoztatott kifogás, hogy a könyvtár nem életszerű feladat-megoldást támogat, ipari alkalmazásokban ilyen könyvtárat nem használnak, jobb lenne egy valódi könyvtár használatát tanítani helyette.
3. Az egyszerű feladatok megoldása túl erős technológiákra épül (származtatott osztályok, futási idejű polimorfizmus, függőség befecskendezés, tervminták), és a kapott kód mérete nem lesz kisebb a feladat elemi eszközökkel előállított megoldásánál. Például egy számlást megoldó ciklussal szemben itt egy megoldó objektumot kell egy olyan osztályból példányosítani, amelyet a számlálást leíró osztály-sablonból származtatunk, de felül kell írni a számlálás feltételét adó metódust, majd ehhez a megoldó objektumhoz egy felsoroló objektumot kell kapcsolni, és végül meghívni rá a run() metódusát.
4. A könyvtár helyes használatát csak az arra vonatkozó korlátozások betartatásával lehet kikényszeríteni. A hallgatók igencsak kreatívnak bizonyultak a könyvtár nem szakszerű használatában (a run() metódust felülírták; egy programozási tételből származtatott osztályban új adattagokat vettek fel, és ezeket módosították a felüldefiniált metódusokban, rekurzív hívásokat alkalmaztak stb.). A könyvtár helyes használatát egy évről évre bővülő tiltásokat tartalmazó szabály kódexszel próbáltunk kivédeni.
5. Nem sikerült jól az összegzés programozási tételének osztálya. Ugyanis a tétel igen robosztus, nagyon sokféleképpen lehet paraméterezni, és ezt a szabadságot olyan osztállyal sikerült biztosítani, amely különösen sok lehetőséget adott a hallgatóknak ahhoz, hogy a konvenciókat felrúgva, ne programozási tételekre épülő megoldásokat készítsenek.

3. Új elemek az osztály-sablon könyvtárban

A hallgatói visszajelzések, az oktatás és számonkérés alapján szerzett tapasztalatok után felmerült az igény a könyvtár megújítására. Ehhez felhasználtuk az új C++ nyelvi szabvány elemeit, melyek részben javítottak a könyvtár kódjának minőségén, részben pedig a könyvtár szakszerű használatát mozdították elő. Azt a szabályrendszert, melyet a zárthelyiknél eddig ki kellett kötni és külön kellett ellenőrizni,

most olyan nyelvi elemekkel biztosítottuk, melyeknek köszönhető a C++ fordító garantálja a szabályok betartását a könyvtár használata során.

3.1. Az összegzés programozási tételre épülő osztály módosításai

Az összegzés programozási tételét leíró osztály korábbi változata túl általánosra sikerült. Például eredetileg a *body()*-t alkotó *add()* művelet nem volt konstans, ezért meglehetősen szabadon lehetett felüldefiniálni. Akár egy tetszőleges algoritmus is leírható volt vele, ha megfelelő adattagokat vettünk fel a *Summation* osztályból származtatott osztályba, és az *add()* függvény felüldefiniálásákor, amely gyakorlatilag az ősosztály ciklusmagját helyettesítette, bármit csinálhattunk ezekkel az adattagokkal.

```
#pragma once

#include "procedure.hpp"
#include <iostream>

template < typename Item, typename Value = Item >
class Summation : public Procedure<Item, Value>
{
private:
    Value _result;
protected:
    void init() final override { _result = neutral(); }
    void body(const Item& e) final override {
        if(cond(e)) _result = add(_result, func(e));
    }
    virtual Value func(const Item& e) const = 0;
    virtual Value neutral() const = 0;
    virtual Value add( const Value& a, const Value& b) const = 0;
    virtual bool  cond(const Item& e) const { return true; }
public:
    Summation(){}
    Summation(const Value &v) : _result(v) {}
    Value result() const { return _result; }
};
```

2. ábra. Összegzés programozási tételt leíró új osztály-sablon

Hiányzott a korábbi *Summation* osztályból az összegzés programozási tételnek [2] azon függvénye, amely az aktuálisan felsorolt elemeket az eredmény típusára képezi le. Ezt az *add()* művelet felülírásának keretében kellett a felhasználónak megadnia.

A korábbi változatban az *init()*-et nem a *Summation* osztály definiálta felül, hanem az abból származtatott konkrét osztályban kellett azt a felhasználónak megadni, pedig az összegzés programozási tétele szerint kezdetben kizárólag az eredményt kell beállítani a neutrális értékre.

Az új változatban (2. ábra) a *Summation* osztály definiálja felül a *Procedure* osztály *init()* és *body()* metódusait, és ennek konkrét leszármazottjaiban kell majd megadni a felsorolt elemhez értéket rendelő *func()* leképezést, a *neutral()* függvényben a neutrális értéket, valamint a két érték összeadását végző *add()* műveletet.

Ettől a módosítástól azt várjuk, hogy a hallgatók a *Summation* használata során kizárólag az általunk elvárt módszert, a megtanult visszavezetési technikát tudják csak alkalmazni.

Az összegzés kódja számos új nyelvi módosításon is átesett, melyek nem kötődnek szorosan az összegzés módosításaihoz, így majd a későbbiekben fogjuk kielemezni őket.

3.2. Új felsoroló típusok bevezetése

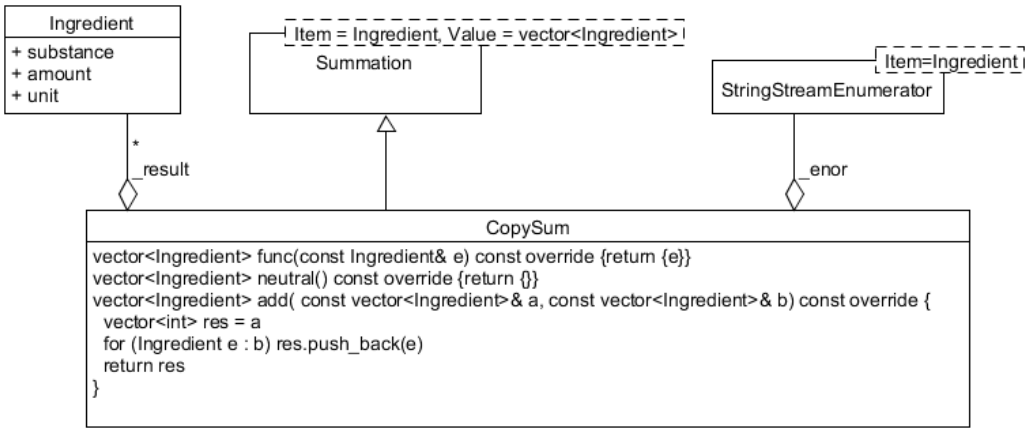
A könyvtár korábbi változata nem támogatta azon feladatok megoldását, amelyekben szöveges állományokat soronként kell feldolgozni (például felfűzni egy-egy sor adatait egy listába), de az egy sorban található adatok száma változó. Az ilyen feladatokat csak ciklus beépítésével lehetett megoldani úgy, hogy a sort szövegfolyammá alakítottuk, és addig olvastunk ebből, míg vége nem lett az adott sornak. Ez a megoldás szembe ment azzal az elképzeléssel, hogy tiltsuk meg a ciklusok használatát, mert ez hatékony kényszerítő eszköz ahhoz, hogy a hallgatók a megoldásaikban a programozási tételekre történő visszavezetésre támaszkodjanak. Ennek a dilemmának a feloldása kézenfekvő: mivel egy sor feldolgozását is valamelyik programozási tételre lehet visszavezetni, csak egy olyan speciális felsorolóra van szükségünk, amely szövegfolyam elemeit képes bejárni. Ezért vezettük be a *StringStreamEnumeratort*. A forráskód [10]-ben megtekinthető.

Nézzünk meg egy példát a *StringStreamEnumerator* használatára: Tekintsük azt a feladatot, amelyben egyetlen sor szövegesen (tehát sztringként) megadott adatait szeretnénk eltárolni egy listában (C++-ban vectorban). Az adatok egy étel receptjében szereplő összetevők. Minden összetevő három részből áll: az összetevő anyagának neve (sztring), a recepthez szükséges mennyiség (int), és annak mértékegysége (sztring).

A megoldáshoz (azaz a listába történő összefűzéshez, ami egy másolás) az összegzés programozási tételét használjuk egy olyan felsorolóval, mely egy szövegből sorolja fel egy recept összetevőit. Először létre kell hoznunk a recept egy összetevőjét leíró *Ingredient* struktúrát, és ehhez definiálni kell egy beolvasó operátort:

```
struct Ingredient {
    std::string substance;
    int quantity;
    std::string unit;
};
std::istream& operator>>(std::istream& in, Ingredient &e)
{
    in >> e.substance >> e.quantity >> e.unit;  return in;
}
```

Ezután egy saját *CopySum* osztályt kell származtatnunk a 3. ábra szerint a Summation osztálysablonból.

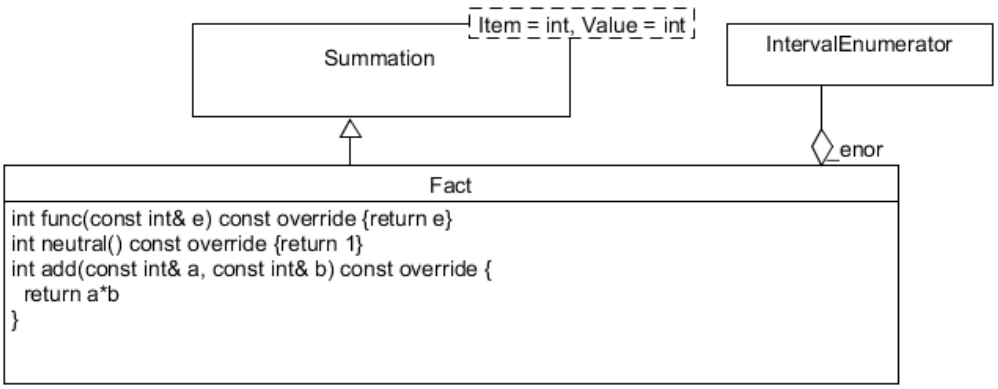


3. ábra. Összetevők összegyűjtésének osztálydiagrammja

Végül az alábbi kódot kell végrehajtani:

```
CopySum sum;
StringStreamEnumerator enum(std::stringstream(
  "tej 1 liter búzadara 13 evőkanál vaj 6 dkg cukor 5 evőkanál"));
sum.addEnumerator(&enum);
sum.run(); // sum.Result() visszaadja az eredmény vectort
```

Egy másik nevezetes felsoroló osztály is beépült az új osztály-sablon könyvtárba. Az *IntervalEnumerator* segítségével, egész számok tetszőleges intervallumának elemeit tudjuk felsorolni [10]. Ennek létrehozásához csak az intervallum elejét és végét kell megadni.



4. ábra. Faktorális kiszámolásának osztálydiagrammja

Erre a felsorolóra van szükségünk például akkor, amikor egy *n* természetes szám faktoriálisát számoljuk ki. Ehhez ugyanis elég az egész számokat kell felsorolni 2-től *n*-ig, és közben a felsorolt számokat összeszorozni. Ez utóbbi visszavezethető az összegzés programozási tételére, ahol az *add()*

művelet speciálisan két egész számot szoroz össze, a neutrális elem pedig az 1 lesz. Jól illusztrálja ez a példa az összegzés robusztusságat [5].

```
Fact sum;
IntervalEnumerator enum(2,n);
sum.addEnumerator(&enum);
sum.run();     // sum.Result() visszaadja a faktoriális értéket
```

3.3. Final kulcsszó bevezetése virtuális metódusoknál.

A korábbi verzióban probléma volt, hogy a hallgatók olyan metódusokat akartak felül definiálni, melylyel felborították a sablon metódus tervminta alapján kialakított könyvtárat, és eltértek a programozási tételekkel történő konvencionális megoldástól. Ezt a problémát tudja kiküszöbölni a *final* kulcsszó, mellyel a fordító biztosítani tudja, hogy mely metódusuk felüldefiniálása nem megengedett.

A *final* kulcsszót virtuális metódusoknál használhatjuk C++11-től kezdve [8], amely azt fejezi ki, hogy az adott metódust a leszármazott osztályokban már nem definiálhatjuk felül. Például a *Procedure* osztály *run()* metódusa, vagy a *Summation* osztály *init()* és *body()* metódusai előtt láthatunk ilyet. Ez biztosítja, hogy a *run()* metódus mindig az ősosztályban megadott algoritmus sémát hajtsa vége, illetve az összegzés mindig az inicializáló értékadással kezdődjék, a ciklusmag pedig csak a *_result* értékét változtathassa meg, és azt is csak a megadott módon.

3.4. Öröklődés korlátozása a barát osztályokra

A korábbi verziókban lehetőség nyílott arra, hogy a hallgatók elkerüljék, hogy programozási tételekkel oldják meg a kitűzött feladatot, mert a *Procedure*-ből közvetlenül lehetett egy saját algoritmust leíró osztályt készíteni. Ezt ugyan tiltottuk, de manuálisan kellett ellenőrizni.

Ha viszont a *Procedure* ősosztályt egyetlen privát konstruktorral látjuk el, akkor ezzel megakadályozhatjuk, hogy új osztályokat lehessen ebből közvetlenül származtatni, hiszen a leszármazott osztályok példányosítása a *Procedure* konstruktorát is igénylik. De hogyan lehet a könyvtárban azokat az osztályokat definiálni (*Summation*, *LinSearch*, *MaxSearch*, stb.), amelyeknek valóban a *Procedure* leszármazottjainak kell lenni? Ezeket az osztályokat barát osztályként jelöljük meg (**friend** `Summation<Item, Value>`), így számukra hozzáférést biztosítunk a *Procedure* konstruktorához.

Ez a fordítás idejű korlátozás megoldást ad a felvetett problémára, elősegíti a könyvtár szakszerű használatát, rákényszerítve a hallgatókat, hogy valamelyik nevezetes programozási tétellel dolgozzanak, és ne használják közvetlenül a *Procedure* ősosztályt.

3.5. Override kulcsszó bevezetése virtuális metódusoknál

Az *override* kulcsszót a virtuális metódusoknál használhatjuk a C++11-es szabványban [8], mely azt a szándékot fejezi ki, hogy egy virtuális metódust szeretnék felüldefiniálni. Ha elrontjuk a felüldefiniálandó metódus szignatúráját, vagy az adott metódus *final* az ősosztályban, akkor fordítási hibát kapunk. Például az *init*() és a *body()* metódusnál jelezzük a Summation-ban, hogy az *Procedure* osztálybéli megfelelő virtuális metódust felüldefiniáltja.

Ennek didaktikai szempontból két előnye van. Egyrészt láthatóvá válik a könyvtárban, hogy mikor definiálunk felül egy metódust, másrészt, ha rászokunk ennek használatára a saját osztályaikban is,

akkor fordítási időben kijönnek az olyan hibák, mikor valaki elront egy szignatúrát a felüldefiniálás során. (pl. kihagy egy *const* kulcsszót a paraméterlistából)

3.6. Template specializáció használata

Az osztály-sablon könyvtár segítségével egyedi felsorolókat is lehet definiálni, de néhány nevezetes felsoroló használatát közvetlenül is támogatja. Ezek közül az egyik a fájlfelsoroló (*SeqInFileEnumerator*). Ez egy szöveges állomány azonos típusú adatait képes felsorolni. Az egyetlen elvásárunk felé az, hogy létezzen az a beolvasó operátor, amely egy adatot olvas be. Ez az olvasás általában figyelembe veszi az elválasztó jeleket (szóköz, tabulátor jel, sorvége jel), de amikor a szöveges állomány tartalmát karakterenként kell olvasni (azaz egy adat egy karakter), akkor többnyire az elválasztó jeleket is be kell olvasnunk. Ezért a karakterek olvasásához a fájlfelsoroló működését specializálnunk kell.

Korábban ez egy futás idejű ellenőrzéssel volt megoldva. Megvizsgáltuk, hogy a beolvasandó adat típusa karakter-e, és ennek megfelelőnek állítottuk be az olvasó operátor működését. Az új verzióban ezt *template* specializációval oldottuk meg. [9]

A *template* specializáció az a mechanizmus, amikor egy sablon osztályt vagy annak egy metódusát bizonyos típusokra teljesen másképp kezelünk, mint ahogy azt az általános sablon definícióban leírtuk. Mivel ez sablon specializáció fordítás idejű ellenőrzés, így ez egy szebb konvenció, ráadásul a hallgatók is találkozhatnak ezzel az új fogalommal a könyvtárral való ismerkedés során.

A konkrét megvalósításban a *SeqInFileEnumerator*-hoz hozzáadtunk egy létrehozó metódust. Ezt a metódust specializáltuk arra az esetre, amikor a felsoroló karaktereket sorol fel, ilyenkor a fájl létrehozó metódus definícióját kiegészítettük az olvasó operátor átállításával arra, hogy az elválasztó jeleket se ugorja át.

A template specializáció hasznos mechanizmusnak bizonyult a *Summation* definíciójánál is.

Az összegzés tételének ugyanis gyakran előforduló speciális alkalmazásai a másolás, ki- vagy szétválogatás, összefuttatás jellegű feladatokat megoldása [5]. Ilyenkor az eredmény egy sorozat, ezért az összeadás műveletét a sorozatok összefűzése helyettesíti, a neutrális elem pedig az üres sorozat. A gyakorlati feladatokban azonban ezt az eredményként keletkező sorozatot a szabványos kimenetre vagy egy szöveges állományba szokták kiírni, azaz a *Value* sablon-paraméter az *std::ostream*. Az ilyen esetekben hatékonyabb, ha átadjuk az összegzésnek a kiírás helyét jelző *ostream* típusú referencia értéket (például *cout*), hogy az összegzés közvetlenül ide helyezze el az eredmény-sorozat elemeit, ne egy ideiglenes sorozatba. Ekkor viszont nincs értelme az eredményt (*ostream* típusú referencia értéket) inicializálni, emiatt az *init()* törzsét üresen kell hagyni, és ennél fogva a *neutral()* metódusra nincs szükség. Az eredményt, azaz a *Summation std::ostream* típusú *_result* adattagjának új értékét közvetlenül a `_result << func(e)` utasítással állíthatjuk elő, és ezt a *body()* törzsében kell elhelyezni az *add()* hívása helyett. Tehát az *add()* metódusra sincs szükség.

Ezt a speciális esetet írja le a *Summation* 5. ábra szerinti változata.

```
template < typename Item >
class Summation<Item, std::ostream>
                    : public Procedure<Item, std::ostream> {
private:
    std::ostream *_result;
protected:
    void init() override final { }
    void body(const Item& e) override final {
        if(cond(e)) *_result << func(e);
    }
    virtual std::string func(const Item& e) const = 0;
    virtual bool cond(const Item& e) const { return true; }
public:
    Summation(std::ostream *o) : _result(o) {}
};
```

5. ábra. Összegzés programozási tétel speciális esetét leíró új osztály-sablon

3.7. Egyéb új nyelvi elemek bevezetése

3.4.1. nullptr

C++11 előtt nem volt nyelvi kulcsszó a semmire nem mutató pointer kifejezésére, helyette a 0 konstanst vagy a NULL makró definíciót használhattuk. A könyvtár korszerűsítésével elkezdtük használni a *nullptr* kulcsszót, mely nem igényel „includot" és a fordító nem keveri össze semmilyen esetben sem a 0 konstans számértékkel. [7]

3.4.2. #pragrma once direktíva

Egy ugyan egy nem szabványos, de széles körben támogatott processzor direktíva. Az „include" őrfeltétel kiváltására szolgál.

4. Összefoglalás

Az osztály-sablon könyvtár előző fejezetben bemutatott kiegészítései és módosításai megnyugtató választ adnak a második fejezetben felvetett 4. és 5. kritikai észrevételekre.

2018-tól új tanterv szerint folyik az ELTE IK-n a programtervező informatikus képzés, és ennek keretében ugyanazon tantárgy tanítja a felsorolókra épített programozási tételek és az azokra visszavezethető (gyűjtemények feldolgozását végző) feladatok megoldását, valamint az objektum elvű programozás. Ezen témák természetes összefonódásaként találta meg a helyét a képzésünkben a felsorolót használó programozási tételek osztály-sablon könyvtára, és annak objektum-orientált felhasználása. Ez talán tompítja az első és harmadik kritikai észrevétel élét is.

Az első három kritikai észrevétellel kapcsolatban az alábbiakat mondhatjuk el.

Az első egy szubjektív álláspont, de nem akarunk vele vitatkozni. Ugyanakkor le kell szögezni, hogy az oktatásnak nem célja, hogy a hallgatókat ne mozdítsa ki a komfortzónájukból. Egy diplomás szakembertől általában is elvárható, hogy megváltozott körülményekre is képes legyen adaptálni a tudását, és ez különösen érvényes a nagyon gyorsan változó informatika területén.

A második kritikának érthető az oka, de egy ipari alkalmazásokhoz használt könyvtár oktatása egyrészt sokkal több időt venne igénybe, másrészt a tartalmi bőség miatt kevésbé lehetne fókuszálni arra a képzési célra, mely szerint az objektum-orientált technikákat (származtatás, függőség befecskendezés, példányosítás, futási idejű polimorfizmus kihasználása stb.) kellene a hallgatóknak megismerni és elsajátítani. Az általunk adott könyvtár a tantárgy anyagára épül, így azt nem kell külön megismertetni a hallgatókkal szemben egy az ipari alkalmazásokhoz hasznos könyvtárral.

A harmadik kritikát is elfogadjuk, de úgy gondoljuk, hogy az összetett technológiákat éppen az egyszerű feladatok megoldásán keresztül kell bemutatni.

Köszönetnyilvánítás

A projekt az Európai Unió támogatásával, az Európiai Szociális Alap társfinanszírozásával valósult meg (EFOP-3.6.3-VEKOP-16-2017-00002).

Irodalom

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. 1st edition, ISBN 0201633612, Pearson Education Inc., Addison Wesley Professionals, 1995.
2. Gregorics, T.: Programming theorems on enumerator, Teaching Mathematics and Computer Science, Debrecen, 8/1 (2010), 89-108
3. Gregorics, T.: Analogous programming with a template class library, Teaching Mathematics and Computer Scinece, Debrecen, 10/1 (2012), 135-152.
4. Gregorics, T.: Programozás – Megvalósítás. ISBN 978-963-312-065-1, Elte Eötvös Kiadó, Budapest, 2013.
5. Gregorics, T.: Force of Summation, Teaching Mathematics and Computer Scinece, Debrecen, 12/2 (2014), 185-199.
6. Stroustrup, B.: A C++ programozási nyelv. ISBN 963-9301-18-3, Kiskapu Kft. 2001.
7. Deitel, P. J., Deitel, H. M.: C++11 for Programmers. 2nd Edition, ISBN-13: 978-0133439854, Pearson Education Inc., 2014.
8. Az override és final használata: https://arne-mertz.de/2015/12/modern-c-features-override-and-final/ (2018.10.31.)
9. A template specializáció: https://en.cppreference.com/w/cpp/language/template_specialization (2018.10.31.)
10. Az osztály-sablon könyvtár: https://people.inf.elte.hu/gt/oep/library.zip (2018.10.31.)

Valós idejű oktatási rendszer

H. Bakonyi Viktória[1], Illés Zoltán[2]

{[1]hbv, [2]illes}@inf.elte.hu
ELTE IK

Absztrakt. Napjaink diákjai digitális bennszülöttek, akik párhuzamosan több különböző eszközt használnak (mobil telefon, tablet, laptop, stb.) Mit is tehetnénk, hogy az órákon ne kalandozzon el a figyelmük és hatékonyan bekapcsolódjanak a tanulási folyamatba? A válaszunk egyszerű, tiltás helyett használjuk fel ezeket az eszközöket az iskolai munkában is. Tegyük az óráinkat vonzóbbá és interaktívabbá a gyerekek saját eszközeinek felhasználásával! Beszámolunk a saját készítésű rendszerünkről illetve kutatásunkat azzal kapcsolatban, hogy milyen tanáraink affinitása az újdonságok bevezetéséhez.

Kulcsszavak: valós idejű, CRS, E-Lection, interakció, tanulás, BYOD, okos eszköz, oktatás

1. Bevezetés

Az utóbbi évtizedben a technika fejlődése soha nem látott gyorsaságú változást hozott a hétköznapi életünkbe is. Az adóbevallástól kezdve, a netbankoláson, a vásárláson át a szórakozásig mindent az interneten végezhetünk. Vége azoknak az időknek, amikor csak a néhány magasan kvalifikált szakmában volt szükség a számítógépek értő használatára. Ma már nehezen találunk olyan területet, ahol a munkavégzés közben ne lenne szükség valamilyen szintű informatikai ismeretre. Eközben újabbnál-újabb szakmák jelennek meg, amelyeket az internet és az okos eszközök megjelenése, elterjedése hozott létre (pl. blogger, mobil-alkalmazás fejlesztő, okos otthon építő).

A mai gyerekek születésük óta ebben a közegben élnek, így teljesen természetesen kezelik a hétköznapjaikat megkönnyítő vagy szórakoztató alkalmazásokat. Ez azonban nem jelent egyet a rendszerezett informatikai ismeretekkel, amelyek átadása továbbra is az iskola feladata marad. A 2019-től várhatóan életbelépő NAT tervezete (https://bit.ly/2ql6Nbd) ebbe az irányba kíván lépni és az eddigieknél hangsúlyosabban kívánja kezelni a digitális kultúra, a számítógépes gondolkodás, a problémamegoldás területét.

Tudomásul kell vennünk azt is, hogy nem elég a tananyag tartalmat megváltoztatni, hanem az iskolai, oktatási környezetet is radikálisan át kell alakítani az új elvárásoknak megfelelően: Az iskola szerves része a társadalomnak, nem lenne szabad egy attól idegen, elzárt világot teremteni az oktatási intézményekben. Nem tiltani kell a modern eszközök és lehetőségek használatát, hanem okosan felhasználni azokat!

"Students inhabit a 21st-century world for 18 hours a day, and, all too often, educators put them in a 19th-century classroom for six hours of that day, and the students feel a tremendous disconnect. We have a responsibility to teach them the skills to optimize these tools." (https://bit.ly/2FNB3m5) (Szabad fordításban: A diákok a nap nagy részében a XXI században élnek, csak az iskolában kényszerülnek gyakran visszarepülni a XIX. századba. Abban áll a felelősségünk, hogy megtanítsuk őket a modern eszközök helyes használatára!)

Hazánkban már régóta kísérleteznek elkötelezett tanárok, hogy a tanítási módszereiket, gyakorlatukat különböző eszközök, tabletek, telefonok, számítógépek és a velük megjelenő új lehetőségek (pl. fénykép, videó készítés) bevonásával színesítsék, élményszerűvé tegyék. A cikkben bemutatjuk az olvasóknak saját készítésű rendszerünket és az első eredményeket.

Supported by organization EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies The Project is supported by the Hungarian Government and co-financed by the European Social Fund.

2. Okos osztályok, okos eszközök

Ha kinyitjuk az újságot – akarom mondani, rákattintunk a neten egy hírközlő oldalra – csupa „okosságról" hallunk, ma minden okos, okos telefon, okos ház, okos óra, okos pelenka, okos fazék vagy okos osztály. Néha nem értjük, hogy mitől okos egy pelenka, egy fazék vagy akár mitől okos egy osztály? A közös mindegyikben az, hogy a megszokott hagyományos funkcióikat különböző informatikai lehetőségekkel bővítik ki. Nyilván egy-egy megoldásnak jelentős pénzbeli vonzata is lehet. Nem véletlen, hogy hazánkban vannak ugyan szép számmal okos osztályok, tele tehetséges diákokkal és elkötelezett tanáraikkal, de viszonylag kevés informatikai értelemben vett „okos osztály"-ról tudunk (https://bit.ly/2JvH9cv).

Egy modern „okos osztály" rengeteg olyan eszközzel van felszerelve, amelyek együttesen professzionális tanulási környezetet teremtenek. Említhetjük a padokba beépített mikrofonokat, kivetítőket, kártyaolvasókat, szavazógépeket, érintőképernyős eszközöket, okos táblákat, okos asztalokat, videó kamerákat vagy az órák streamelhetőségét. (1. ábra)

1. ábra: Okos osztály, okos előadóterem

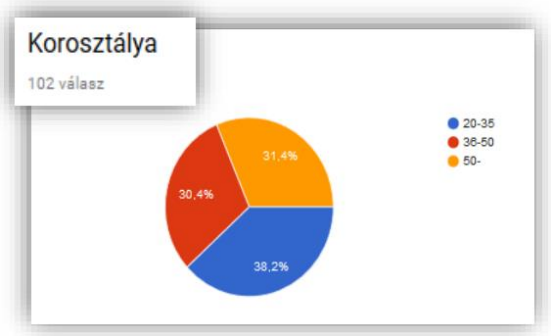
Az IoT (Internet of Things) elterjedésével azonban újabb lehetőségekkel bővülhetnek az okos osztályok például olyan helyi szolgáltatásokkal, amelyik az osztályterembe lépő tanárhoz kapcsolódó alkalmazásra vált, vagy automatikusan a belépőt felveszi a katalógusba, üzenetet küld a házi feladatról stb. Csak a képzelet szabhat határt a lehetőségek körének! (https://bit.ly/2EVWkNw)

Nyilván lehetetlen egyik napról a másikra okos osztályok, iskolák ezreit kialakítani, de arra azért van lehetőségünk, hogy a diákok saját eszközeivel „okosítsunk" a tantermünkön. Egy laptoppal, kivetítővel és a diákok saját okos telefonjaival már jókora lépést tehetünk előre – kipróbálhatjuk az iskolai szavazórendszerek (CRS-Classroom Response System) használatát. Ingyenesen elérhető például a Kahoot rendszer (https://kahoot.it/), amely széleskörűen használt az amerikai iskolákban is, de itthon is vannak már lelkes hívei. (http://tanarblog.hu/search?term=kahoot) Miért is érdemes ilyen rendszert használni? Egyfelől a diákok motiváltak az okos telefonjuk használatánál (mások szerint jó néhányan már függővé is váltak), másfelől a tanár azonnali visszajelzést kaphat a diákok aktuális tudásáról és a gyerekek is folyamatos visszacsatolást kaphatnak, ami az önértékelésüket segíti. [4]

Két lényeges kérdésre kell válaszolnunk. Az első kérdés az, hogy az újdonság varázsán túl van-e valamilyen mérhető haszna az alkalmazásuknak? [5,6] A második kérdésünk pedig az, hogy széleskörűen teret nyerhet-e a hazai tanári gyakorlatban ezek használata? Tegyünk kísérletet a kérdéseink megválaszolására!

3. A CRS rendszerek ismertsége

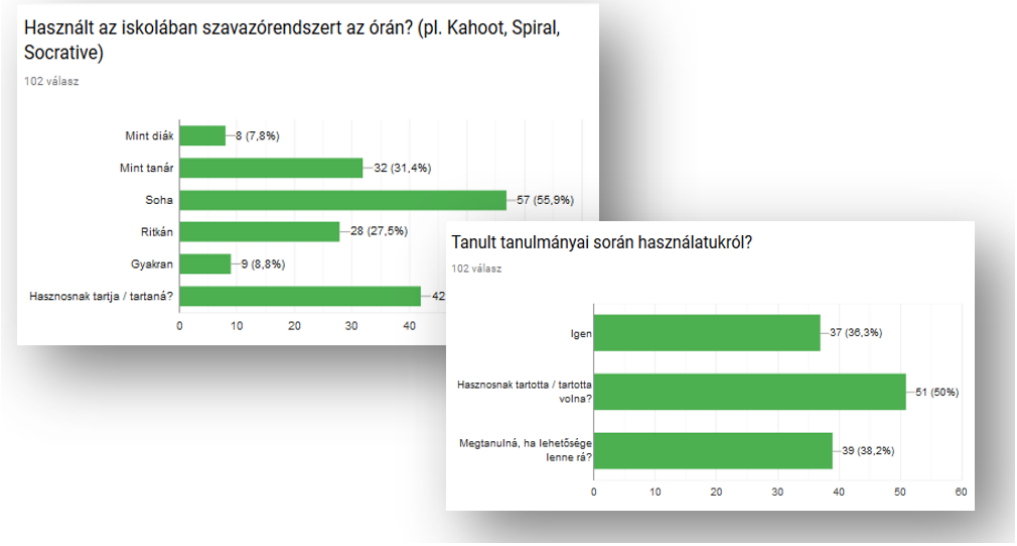
Egyetemi óráinkon van szerencsénk tanár szakos hallgatókkal is foglalkozni. Személyes beszélgetések során derült arra fény, hogy jó néhányan bizonytalanok voltak abban is, hogy használtak-e egyáltalán már CRS rendszert. Ez indított bennünket arra, hogy felmérést készítsünk arra vonatkozóan, hogy a tanári társadalom mennyire van felkészülve például az újonnan megjelenő szavazórendszerek használatára. A kérdéseinket régi tanár kollégák, illetve jelenlegi és korábbi tanítványaink felé juttattuk el megkérve őket, hogy saját ismerőseikkel is vegyék fel a kapcsolatot ez ügyben. Összesen 102 válasz érkezett, nagyjából kiegyensúlyozott mértékben három különböző korcsoporttól (20-35, 36-50, 50-) többségében gyakorló informatika tanároktól. (A korcsoportokat a tanári pályán várhatóan eltöltött idő alapján bontottuk fel, nagyjából azonos hosszúságú, 15 éves intervallumokra. Figyelembe vettük a 35 éves kort, mint a fiatal felnőttkor szakirodalomban meghatározott végét, illetve az OECD statisztikák 50 év felettiek csoportját.)



2. ábra: Korosztályi megoszlás a válaszadók között

Célunk, hogy a válaszadók körét tovább bővítsük. Az anonimitás biztosítására egy Google űrlapot használtunk, amely ma is elérhető ezen a címen:

https://docs.google.com/forms/d/1D-QYkCdRx1Gr4wUsuEUmXr7XtfIFZjeyR07BMF79gYU/

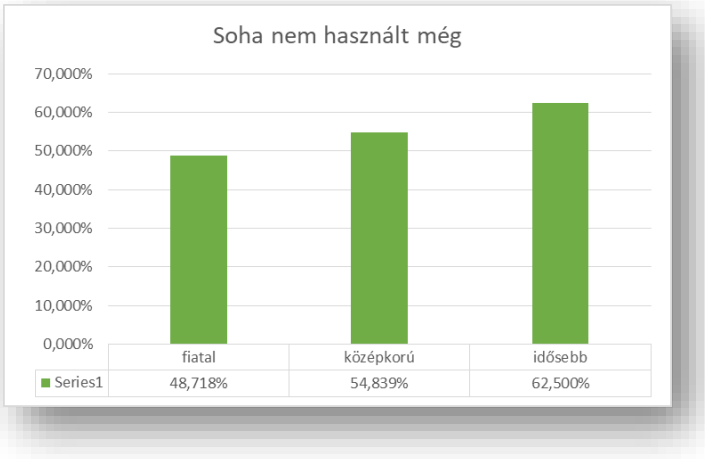


3. ábra: Használt-e szavazórendszert, tanult-e róla

A további kérdések arra vonatkoztak, hogy használt-e szavazórendszert mint diák, mint tanár és ez gyakran, ritkán vagy soha nem történt-e meg továbbá hasznosnak tartja-e, tanult-e róla.

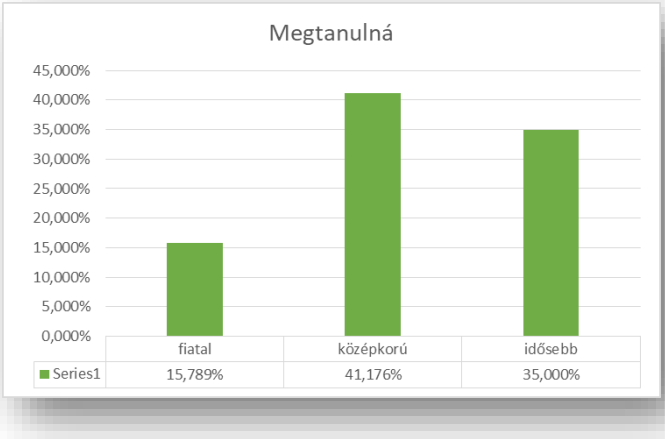
A feladott kérdést és az összesített válaszokat a 3. ábrán lehet megfigyelni. (Nem választottuk szét külön csoportokra a diákként vagy tanárként használókat, hiszen diákjaink nagy része már egyetemi évei alatt is tanít, illetve a már gyakorló tanárok is járnak továbbképzésekre.)

Az általános kiértékelésen túl a korcsoporti megoszlás került az érdeklődésünk fókuszába. Ahogy várható volt, az idősebb kollégák nagyobb százaléka nem használta még ezt a lehetőséget sohase.



4. ábra: Soha nem használtak még szavazórendszert

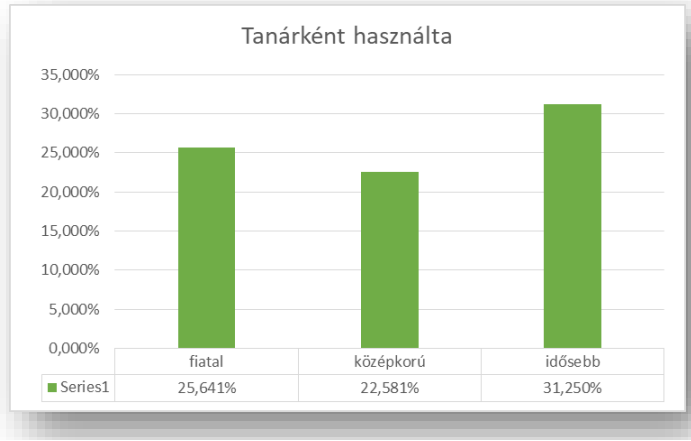
Annál meglepőbb, hogy azok közül, akik soha nem használták még a fiatal korcsoport a legkevésbé motivált ennek megtanulására. Ezt az eredményt figyelhetjük meg az 5. ábrán. További vizsgálatot igényelne az okok feltárása. Jelenleg csak vélelmezni tudjuk, hogy a fiatalabb generáció számára a digitális eszközök használata esetleg már nem olyan érdekes, hiszen természetes a számukra. Más feltételezésünk szerint a motivációjuk esetleg nem olyan erős, mint a már gyakorlottabb kollégáké. Nagyobb minta és részletesebb kérdésfeltevés esetén az esetleges fals válaszok is kiszűrhetőek lennének.



5. ábra: Motiváció az eszköz használatának megtanulására

Ennek fényében érdekes, az is hogy az egyes korcsoportba tartozók hány százaléka használta már tanárként ezt a lehetőséget. Itt is az derül ki, hogy a középkorú és idősebb tanárok bátrabban nyúlnak újabb eszközökhöz. 6. ábra

Összevetve az 5. ábrán és 6. ábrán megadott adatokat, a fiatal korcsoport tanulási kedve és kipróbálásra való hajlandósága messze elmaradnak (41%) az idősebbeké mögött (66%)



6. ábra: Tanárként már használta

Az adatokat áttekintve kijelenthetjük, hogy a pedagógusaink nagy általánosságban nyitottak új módszerek, eszközök megtanulására. Igaz, a felmérés válaszadói jobbára az informatika tanárok köréből származnak és ők nyilván az átlagosnál jobban viszonyulnak a modern eszközökhöz, de azt is látni kell, hogy ők lehetnek egy-egy iskolában a modernizáció katalizátorai.

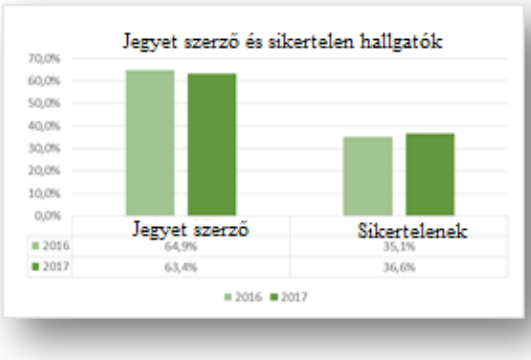
4. E-Lection, útban az interaktív okos előadóterem felé

Célunk az volt, hogy az egyetemi előadásokba is jobban bevonjuk a hallgatóságot, ezért tervbe vettük egy CRS rendszer használatát. Több létező megoldás által nyújtott lehetőséget áttekintettünk, de végül egy saját fejlesztésű rendszer létrehozása mellett döntöttünk, amiről több ízben is beszámoltunk már. [1] (7. ábra) Röviden ez egy web-alapú, valós idejű előadás-menedzselő alkalmazás, amely felhasználja a hallgatói saját okos eszközöket (telefon/tablet/laptop). Az alkalmazásnak több verziója született. A jelenlegi főbb tulajdonságai: hallgatói azonosítás egyetemi lap rendszerrel, katalógus készítés, szükség esetén IP cím szűrés, oktatói illetve hallgatói kérések, jelzések kezelése, logolása adatbányászati célokra illetve az előadások sztreamelése.[2]

Korábbi tesztelések után az első éles használat a 2017/2018-as tavaszi félévben az Operációs rendszerek tárgy keretében debütált. A kicsit módosított jelenlegi verzió az őszi félévben került használatba az átalakult BSC tanrend újragondolt Számítógépes Rendszerek című tárgyában.

7. ábra: http://election.inf.elte.hu

Megvizsgáltuk, hogy vajon van-e valamilyen mérhető változás az eredményekben pusztán amiatt, hogy megjelent egy új motivációs erővel bíró eszköz az oktatásban. (További kutatást igényel majd annak meghatározása, hogy milyen jellegű és mértékű interakció vezet a legoptimálisabb eredményhez.) [3] Először a 2016-os és 2017-es tavaszi féléves Operációs rendszerek tárgybeli eredményeit hasonlítottuk össze. A tárgy előadója, gyakorlatvezetői ugyanazok a személyek voltak, ami feltételezi, hogy nagyjából azonos módon történt a csoportonkénti értékelés. Az évfolyamok mérete is közel

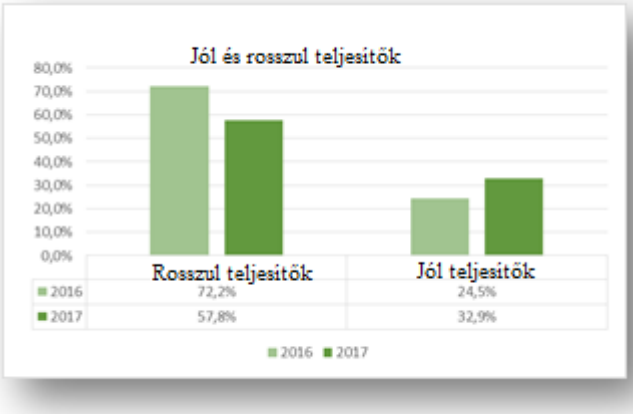


8. ábra: Jegyet szerző és sikertelen hallgató arányok összehasonlítása

azonos 200-220.) Először azok arányát hasonlítjuk össze, akik sikeresen átmentek, illetve akiknek nem sikerült legalább elégségest kapniuk a félév végén (elégtelen vagy feladta félév közben). A 8. ábra mutatja, hogy ebben az esetben releváns különbség nem mutatkozik a félévek között, sőt egy hajszállal még az E-Lection-t nem használó szemeszter produkált jobb eredményt.

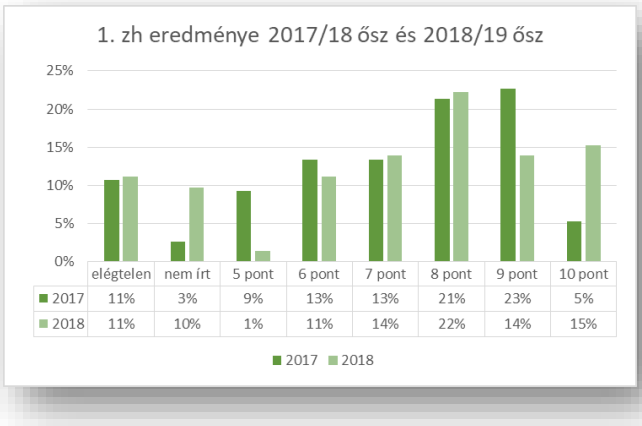
Ha azonban azt vizsgáljuk, hogy a sikeresen átment hallgatók között milyen arányban fordulnak elő a jól teljesítők, akkor már egészen más helyzettel találjuk magunkat szembe. A 9. ábra azt mutatja, hogy a jól teljesítők aránya magasabb az E-Lectiont használók körében. Különböző okokra gondolhatunk, az egyik az, hogy a mobil eszközök használata a várakozásnak megfelelően motivációt jelent, másfelől a monotonitás megtörésével gátolja a figyelem elkalandozását, harmadsorban ráirányítja a

figyelmet egy-egy fontosabb területre, ami így jobban rögzül. A későbbiekben további vizsgálatot igényel, hogy erre pontosabb választ tudjunk adni.



9. ábra: Jól teljesítők és rossz eredményt elérők aránya Operációs rendszerekből

Az idei év eredményeiről a cikk megírásának idejében még nem áll rendelkezésre túl sok információ. Így csak az első dolgozat eredményeit tudtuk összehasonlítani. Az adatokat azoktól a gyakorlatvezetőktől kértük el, akik mindkét évben tartottak gyakorlatot, hogy minél inkább kiküszöbölhessük a szubjektivitást. Az előadó személye mindkét évben ugyanaz volt, a vizsgálatban résztvevő (5 csoportnyi) hallgatóság létszáma is hasonló (95-100).



10. ábra: Első eredmények összehasonlítása Számítógépes rendszerekből, (kerekített értékek)

Az 10. ábrán látható, hogy az első zárthelyi után az már látszik, hogy három esetben van kiugró különbség az eredményekben (nem írta meg, éppen hogy elérte a kettest (5 pont), hibátlan lett). Nyilván a félév végén a teljes eredményeket összehasonlítva pontosabb képet fogunk kapni, de az a korábban levont hipotézis, hogy az elégtelenek számát egyelőre nem befolyásolja az E_Lection használata az itt is beigazolódik. A 9-10 pontosok is összességében nagyjából kiegyensúlyozott képet mutatnak, bár a hibátlan teljesítményűek aránya megugrott a 9 pontosok kárára. Nyilván valamilyen komolyabb következtetést korai lenne még levonni egyelőre, de biztatónak látjuk a helyzetet.

5. Összefoglalás

Napjaink IT technológiai robbanásának eredménye a hétköznapjaink részévé váltak. Az iskola sem vonhatja ki magát ezen társadalmi hatások alól. Az új eszköztár új eszközöket, új módszereket kíván még hatékonyabb tanítás, tanulási folyamatot tudjunk elérni. Megjelentek a különböző technikai lehetőségekkel rendelkező okos osztályok (okos eszközökkel (okos telefon, tablet, laptop) rendelkező diákok), amelyek szolgáltatásai, lehetőségei még néhány évvel ezelőtt is elképzelhetetlenek voltak. A pedagógus társadalomnak az amúgy is nehéz terhei mellett ezekkel a kihívásokkal is meg kell birkóznia. Felmérésünk megnyugtató eredményt hozott, a tanár kollégák zömében középkorú és idősebb tagjai is jórészt motiváltak az újdonságok elsajátítására, ugyanakkor elgondolkodtató (és ez további magyarázatot igényel), hogy az a fiatalabb korosztály, akiktől sokkal nagyobb aktivitást várnánk, kevésbé agilis. Cikkünkben szóltunk még a saját fejlesztésű E-lection szavazórendszerünkről is és annak első mérhető eredményeiről, amelyek szintén egy „okosodó", interaktívabb oktatásról számoltak be.

Irodalom

1. H. Bakonyi Viktória, Illés Zoltán: *Real-Time Tool Integration for Lectures*, 15th IEEE International Conference on Emerging eLearning Technologies and Applications: ICETA 2017.Konferencia helye, ideje: Starý Smokovec, Szlovákia, 2017.10.26-2017.10.27. Denver: IEEE Computer Society Press, 2017. pp. 31-36. (ISBN:978-1-5386-3294-9)
2. Dr. Zoltán Illés, Victoria H. Bakonyi, Jnr. Zoltán Illés: *Modern environment inspired education,* ICAI (International Conference of Applied Informatics) 2017. Eger (2017.01.29-02.1) előadás (2017)
3. Victoria H. Bakonyi, Dr. Zoltán Illés: *Experiences of Using Real-Time Classroom Response Systems*, ICETA 2018 Conference, Konferencia helye, ideje: Starý Smokovec, Szlovákia, 2018.10.10-2017.10.12., megjelenés alatt.
4. Pšenáková, I. (2016): *Interactive applications in the work of teachers*. In: XXIXth DIDMATTECH 2016. Budapest: Eötvös Loránd University in Budapest - Faculty of Informatics. sz. 92-100. ISBN 978-963-284-800-6.
5. Binghamton University, Center for Learning and Teaching: "Student Response System Pilot", 2016 spring, elérhető http://bit.ly/2A5Jhlq (utolsó elérés : 30/10/2018)
6. M. Ortiz: "*The effects of student response systems students achievement and engagement*", thesis, 2014, elérhető https://bit.ly/2KtE8K3 2A5Jhlq (utolsó elérés: 30/10/2018)

Álprofilok használata az etikus és biztonságos internethasználat tanításában

Holló Csaba

chollo@inf.u-szeged.hu
SZTE TTIK Informatikai Intézet

Absztrakt. Az etikus és biztonságos internethasználat tanításának legfőbb kihívása az, hogy az ismeretek átadásának mindaddig kevés haszna van, amíg nem sikerül a diákokban kialakítani azt az attitűdöt, hogy a tanultaknak megfelelően járjanak el. Ugyanakkor, az ismeretek alkalmazása többnyire olyan körülmények között történik, amire a tanárnak nagyon kevés rálátása van, a diák pedig ritkán szembesül a nem megfelelő magatartásának következményeivel, amelyek ugyanakkor nagyon súlyosak lehetnek. A cikkben azt elemezzük, hogy mit érdemes elmondani a diákoknak az álprofilok kapcsán, milyen kompetenciákat fejleszthetünk ezzel, és hogyan használható ez arra, hogy a diákokat meggyőzzük az etikus és biztonságos internethasználat szükségességéről.

Kulcsszavak: álprofilok, etikus és biztonságos internethasználat, nevelés

1. Bevezetés

Minden eszköznek vannak használati szabályai, melyeket a lehetőségek kihasználása, és a veszélyek elkerülése érdekében meg kell tanulni, és a nem megfelelő használatból származó károkért nem az eszközt kell okolni. Az internetes szolgáltatások is *eszközök*, melyek használati szabályait a fiatal generációval is meg kell ismertetni, továbbá azt is el kell érni, hogy azokat be is tartsák.

Az első kérdés, ami ezzel kapcsolatban felmerül, hogy ennek megvalósítása kinek a feladata? A gyermeket nevelni kell arra, hogy a szabályokat betartsa, amit fontos már kiskorban elkezdeni, tehát a gyermek neveléséért felelős személyek (tipikusan szülők) szerepe információ átadóként és mintaként is megkérdőjelezhetetlen. Csakhogy, sok esetben ezeknek a szabályoknak kisebb vagy nagyobb részét a szülők sem ismerik, vagy legalábbis nem alkalmazzák. A gyermek oktató-nevelő tevékenységében a másik fontos szereplő az iskola. A jelenlegi NAT, és – digitális kompetenciaként – a 2018-as NAT tervezet is tartalmazza az etikus és biztonságos internethasználat kialakítását, melyben számos tantárgy részt vehet, továbbá – különösen a viselkedési normák kialakításában – az osztályfőnöknek is nagyon fontos szerepe van. Ugyanakkor, a témában legkomolyabb szakmai ismeret és szakmódszertani felkészültség mégiscsak az informatikatanártól várható el, így az iskola részéről első sorban az ő feladata ennek megvalósítása. Ez azonban szakmódszertani szempontból is új kihívások elé állítja az informatikatanárt, ugyanis ez esetben nem elegendő az, hogy a diák megértse és megtanulja a tananyagot, a tanárnak azt is el kellene érnie, hogy a diák annak megfelelően viselkedjen, olyan környezetben is, amikor erre a tanárnak nincs rálátása. Ez pedig csak úgy lehetséges, ha a diákot sikerül *meggyőzni* arról, hogy ezeknek a szabályoknak a betartása neki is érdeke. A meggyőzésben pedig sokat segít az, ha a diákot szembesítjük a lehetséges következményekkel, akár az érzelmeire is hatva, olyan történetekkel, melyek megtörténtek, vagy megtörténhetnek. Az álprofilok természetesen nem az egyetlen terület, amelynek megbeszélése erre is lehetőséget ad, de mivel a közösségi hálózatok használatának elterjedése miatt ennek tárgyalása úgysem kerülhető ki, nézzük meg, hogy mit érdemes a témában elmondani és ez milyen tanulságokkal járhat.

2. Mi az álprofil és milyen a készítője?

Kamuregnek fogjuk nevezni az olyan felhasználót, aki olyan valótlan adatokat jelenít meg magáról, amelyek következtében valódi kiléte nem, vagy csak nehezen kideríthető. A valótlan adatokat tartalmazó profil lesz az *álprofil* vagy *kamuprofil*.

Álprofilokat sokféle okból és céllal készítenek, ám közös tulajdonságként azt kijelenthetjük, hogy az álprofil készítője valamilyen okból nem meri, vagy nem akarja felvállalni önmagát. Ez lehet például félelem (előítéletektől, kihasználástól, szülőktől, hatóságoktól, kommentelőktől stb.), a tulajdonosának olyan vágya is, hogy a valóságnál jobb képet mutasson magáról, de lehet mögötte üzleti vagy rosszindulatú cél is.

A kamuregekről további jellemzést a különböző célú álprofilok részletesebb leírása kapcsán fogunk adni.

3. Az álprofil „hitelesítése"

Az álprofil annál hatékonyabban használható, minél inkább sikerül arról másokkal elhitetni, hogy igazi. A következő trükkök ismerete azért lehet hasznos, hogy legyen néhány szempontunk, melyeket érdemes átgondolni, amikor valaki ismerősnek jelöl minket.

Egy valódinak tűnő profilhoz ismerősök kellenek. Hogyan tehet szert a tulajdonosa ismerősökre? Bár egy álprofilnak is jó esélye van arra, hogy akadjanak olyanok, akik visszajelölik [1], a legnehezebb mégis a kezdet, amikor egyetlen ismerőse sincs, hiszen egy idegen számára ez nagyon gyanús lehet. Megoldás: a kamureg készít még néhány álprofilt, és ezek egymást elfogadják ismerősnek. A több álprofilnak egyébként is sok haszna lesz még.

Az álprofilokhoz fényképek kellenek, melyek akkor növelik a profil valódiságának látszatát, ha (más) emberek fényképei lesznek (tehát nem állatok, tájak és egyebek). Az álprofilok későbbi használhatóságánál fontos lehet, hogy a fényképek valódiságát a kamureg minél hihetőbben bizonyítani tudja. Ennek érdekében a fényképekkel megteheti a következőket. Valamelyik másik álprofiljáról megírja a fénykép igazi tulajdonosának, hogy látott az ő képével egy másik profilt is, és nem tudja eldönteni, hogy melyik profil az igazi, és ezért kér tőle egy általa (például saját kézírással) hitelesített képet. Hogyha sikerül megszerezni a képet, akkor azt a kamureg képszerkesztő programmal módosítja, aminek segítségével a későbbi ismerőseinek azt állíthatja, hogy ő az igazi, és a kép igazi tulajdonosa a csaló.

Hogyha ily módon van már néhány ismerős, akkor a kamureg próbálkozhat azzal, hogy ismerősnek jelöl ismeretlen embereket is, tipikusan olyanokat, akik várhatóan kevésbé figyelmesen fogják ellenőrizni az ő profilját. Ilyenek lehetnek például a nagyon sok ismerőssel rendelkező profilok, vagy olyan fiatalok is, akik egy jóképű srácot vagy csinos hölgyet (ami természetesen lehet hamis kép) az átlagosnál kevesebb mérlegeléssel jelölnek vissza [1].

Elképzelhető azonban az is, hogy a kamuregnek konkrét „kiszemelt" áldozata is van. Ebben az esetben célravezetőbb lehet, ha először az áldozat ismerőseit jelöli be a más hamis profiljairól [1], és megpróbál minél több információt megtudni az áldozatról (például kedvencek, meglátogatott helyek, vallás). Ezután jöhet az áldozat bejelölése, akinek, ha gyanakszik, mondhatja, hogy találkoztak már egy olyan helyen, ahol az áldozat is járt, és ahol sok ember ismerkedik egymással (buli, nyaralás, konferencia, vallási szertartás). Továbbá, az áldozatról szerzett információkkal könnyebben meg tudja nyerni annak szimpátiáját azáltal, hogy az áldozat által kedvelt dolgokról azt állítja, hogy ő is azokat, szereti, olyan vallású stb. Ha az áldozat (később) mégis gyanút fog, akkor pedig elküldheti neki a fentiekben leírt módon hamisított képet.

4. Az álprofil használata

Az álprofilok használatát céljaik szerint fogjuk csoportosítani és tárgyalni.

4.1. Társkeresés

Az álprofilok használatának egyik leggyakoribb célja a társkeresés [7].

Mielőtt rátérnénk az álprofilok tárkeresésben történő használatára, érdemes megemlítenünk egy olyan szempontot, ami minden társkeresésben szerepet játszik. Minél kevesebb tényleges tapasztalásunk van a másikkal kapcsolatban, annál valószínűbb, hogy fel fogjuk ruházni olyan tulajdonságokkal is, amelyekkel nem rendelkezik, emiatt pedig erősebben szerethetjük (vagy akár gyűlölhetjük) a másikat, mint a valóságban. Ez a szempont az internetes társkeresésben pedig annál nagyobb mértékben jelentkezhet, minél korlátozottabb kommunikáció folyik a résztvevők között.

A társkeresés lehet többé vagy kevésbé komoly, de feltételezni fogjuk, hogy alapvetően jó szándékú. A kifejezetten mások kihasználására irányuló „társkeresést" a későbbi alfejezetekben fogjuk tárgyalni.

Az álprofillal történő társkeresés általában először jó szórakozásnak tűnik, a kamureg nem is gondolja, hogy tényleg komolyra fog fordulni valamelyik kapcsolat. Hogyha csak magányos és szórakozik, de a másik nem kell neki eléggé ahhoz, hogy iránta érzelmeket tápláljon, akkor ez a másik számára rossz lehet akkor is, ha a kamuregben nem volt kifejezetten ártó szándék. Hogyha pedig a kamuregben lesznek érzelmek, és a kapcsolat komolyabbra fordul, akkor nem meri bevallani, hogy bizonyos információk nem felelnek meg a valóságnak, mert nem akarja elveszíteni a másikat.

4.1.1. A jó szándékú álprofilos társkeresés haszna

Feltevődik a kérdés, hogy miért készít valaki jó szándékú társkereséshez álprofilt? Ennek több oka is lehet.

Egyrészt, lehet az, hogy fél az előítéletektől, nem akarja, hogy mások megtudják, hogy ilyen módon keres társat magának, vagy azt, hogy milyen társat keres (például azonos neműt).

Másrészt, lehet az is, hogy (valótlanul) negatív énképpel, vagy legalábbis (indokolatlanul) csekély önbizalommal rendelkezik, valószínűleg azért, mert korábban sokat bántották, tárkeresési kezdeményezései kudarcosok voltak, és azt reméli, hogy ha a „feljavított" profillal sikerül olyan társat találnia, aki jobban megismeri és kötődni fog hozzá, akkor az majd az igazi személyiségét is el fogja fogadni. Ennek aleseteként előfordulhat, hogy konkrét célszeméllyel akar közelebbi kapcsolatot létesíteni, akit a valóságban ismerhet.

Tehát, a kamureg többnyire lelki problémákkal küzd, és valószínűleg olyan társsal fogják egymást elfogadni és jól megérteni, aki hasonló tapasztalatokon ment át, és úgyszintén lelki problémákkal küzd. Már önmagában annak is terápiás hatása lehet, hogy a kamureg az álprofil mögül elmondhat olyan dolgokat, amiket a való életben nem vállalna fel, vagy másnak nem mondhatna el. Ugyanakkor, egymást kölcsönösen átsegíthetik kritikus helyzeteken, és mindkét fél részéről könnyen kialakulhat valamiféle „hiánypótló szerelem" a másik iránt, aki őt végre látszólag elfogadja. A kamureg részéről ez nyilvánvalóan önámítás, de pillanatnyilag jobbnak tűnik, mint a valóság. Viszont önámítás lehet a másik fél részéről is, akinek annál fájóbb lenne beismernie, hogy ez tévedés volt a részéről, minél több időt, érzelmet és pénzt ölt bele a kapcsolatba, ezért nem akarja meglátni a nyilvánvaló gyanús jeleket sem.

4.1.2. A jó szándékú álprofilos társkeresés csapdája

A kamureg sok esetben azt gondolja, hogy ő tulajdonképpen semmi rosszat nem tesz, különösen ha „csak" a kép és a név nem igazi, de egyébként szereti a másikat, támogatja azt, és komolyan akarja a kapcsolatot. Mi itt a probléma?

Egy valós fénykép is a másik környezetre gyakorolt benyomásának csak töredékét fejezi ki, hiszen többnyire nem átlagos körülmények között, hangulatban, ruhában készült, lehet retusált is, ugyanakkor nem tudja tükrözni megfelelően a másik „kisugárzását". Az egymásról ismert képi és személyiségi információkat a felek képzeletükkel egészítik ki, ami nagyon messze lehet a valóságtól. Ily módon, az egymást kizárólag fényképről ismerőkben, nagyon mély virtuális kapcsolat esetén is, az első találkozás alkalmával könnyen lehet olyan érzés, mintha a másik egy idegen lenne, és komoly csalódás lehet az is, hogy a másik nem egészen olyan, mint amilyennek képzelték. Hogyha a fénykép hamis, akkor mindez fokozottan jelenik meg, ráadásul a másik fél becsapottnak fogja érezni magát, és kicsi a valószínűsége annak, hogy még bízni akar és tud abban, akiről már az első találkozáskor kiderül, hogy hazudott neki. Ily módon az első találkozás jó eséllyel a kapcsolat végét jelentheti akkor is, ha egyébként a kamureget a társa a (talán kevésbé vonzó) valós képével és tulajdonságaival elfogadta volna. Az önámítás pedig annál rosszabb, minél tovább tart, hiszen közben mindketten tényleges lehetőségekről maradhatnak le.

4.1.3. Tisztességtelen eszközök a társkeresésben

Többé vagy kevésbé jó szándékú társkeresés során is előfordulhat olyan, hogy a kamureg tisztességtelen eszközöket vet be annak érdekében, hogy magához láncolja a másikat. Érdemes ezek közül is nézni néhány példát.

Korábban már említettük, hogy a több álprofil különböző célokból használható. Jelen esetben a kamureg megteheti azt, hogy más álprofiljairól sértegeti az áldozatot, majd a fő álprofiljáról látványosan kiáll mellette és megvédi a támadásoktól azért, hogy mutassa, hogy ő milyen jó barát, és megnyerje a másik bizalmát.

A kamureg a konkurencia kiiktatása céljából a következővel is próbálkozhat. Készít az áldozatról egy álprofilt, azzal ismerkedik, majd ezt az információt eljuttatja az áldozat párjának, aki azt fogja hinni, hogy az áldozat őt megcsalja. Hogyha ki is derül az igazság, pillanatnyilag ez okozhat egy olyan válságot az áldozat kapcsolatában, ami a kamureg számára megkönnyíti az áldozat elcsábítását.

Elképzelhető, hogy a „szerelem hevében" egymással intim információkat vagy meztelen képeket cserélnek, vagy legalábbis a kamureg ilyeneket kér. Ezzel viszont az áldozat zsarolhatóvá válik. Például, később hiába jön rá arra, hogy a kamureg hazudik magáról, nem meri elhagyni abbéli félelmében, hogy a kamureg a megszerzett információkat és képeket közzéteheti.

Az is megtörténhet, hogy a kamureg megszállott lesz, aki a másikat „megfojtja a szerelmével", vagy legalábbis féltékeny lesz, ebben az esetben nem meglepő, ha igyekszik megakadályozni, hogy az áldozat másokkal ismerkedjen. Tesztelheti a másik hűségét úgy, hogy egy másik kamuprofiljáról megpróbálja elcsábítani, vagy más álprofilokon keresztül az áldozat ismerőseitől érdeklődik az áldozatról. Az is lehetséges, hogy valaki csak azért készít álprofilt, hogy a párja hűségét tesztelje.

Hogyha a kamureg úgy érzi, hogy veszélyben van a kapcsolat, akkor érzelmi zsarolást is bevethet. Például, azt írja, hogy valamilyen súlyos betegségben szenved azért, hogy az áldozat úgy érezze, hogy tisztességtelen lenne, ha éppen most hagyná el, amikor a kamuregnek szüksége van rá.

4.2. Másik személyiség használata

Vannak, akik azért készítenek álprofilt, mert ezáltal élik ki személyiségük azon oldalát, amilyenek lenni szeretnének, de valamiért nem lehetnek, vagy nem mernek lenni.

Ennek talán leggyakoribb alesete, amikor valaki különböző oldalakon álprofillal kommentel. Az nyilvánvaló, hogy nem meri saját nevével felvállalni a véleményét, ettől függetlenül a kommentelő tevékenysége lehet jó vagy ártó szándékú, és ezt megvalósíthatja mások tiszteletben tartásával, vagy erőszakos és provokatív módon trollként, vagy zaklatással.

4.3. Kihasználás, rosszindulat

A teljesség igénye nélkül felsorolunk néhány olyan lehetőséget, ahogyan a kamureg mások kárára visszaélhet az álprofiljaival.

Lehet, hogy a kamureg „csak" kihasználni akarja az áldozatot, az is lehet, hogy uralkodni akar felette, és az is, hogy kifejezetten ártani szeretne neki. Az utóbbi származhat abból, hogy korábban csúfolták, megalázták, megcsalták, és bosszút akar állni úgy általában „az embereken" (ebben az esetben gyűjtheti is az áldozatokat), vagy kifejezetten az áldozaton, akire személyesen is haragudhat.

4.3.1. Adathalászat

A kamureg az álprofiljait arra is felhasználhatja, hogy másoktól olyan adatokat tudjon meg, amikkel később visszaélhet. Ennek érdekében gyakran valakinek a profilját másolja le, akinek a nevében megpróbálja megszerezni a kívánt adatokat.

Példaként említjük, hogy a 2014-es INFO ÉRA konferencián Szilágyi András egy emlékezetes előadásában ismertetett egy saját kísérletet, melynek segítségével bemutatta, hogy saját magáról készített álprofilja segítségével (amit bárki más is készíthetett volna), bárki más által megszerezhető információk segítségével, hogyan tudott diákjaitól személyes adatokat megszerezni. Ehhez az álprofilja mellett egy telefonra kiküldött programot, és elvileg névtelen, de bizonyos információk együtteséből azonban névvel beazonosítható kérdőívet használt. A kísérlet eredményét utólag ismertette a diákjaival, akiknek ez valószínűleg egy életre szóló leckét jelentett.

4.3.2. Lájkvadászat, marketing

Az ilyen álprofilokat a kamureg azért hozza létre, hogy lájkokat gyűjtsön vagy – akár egy cég megbízásából – termékeket reklámozzon. Ennek érdekében igyekszik minél több ismerőst gyűjteni, akik ily módon nagyobb bizalommal fogják fogadni a felkéréseit lájkokra vagy vásárlásokra.

Vannak olyan cégek is, amelyek ellopják vagy megvásárolják más profilok adatait és azok felhasználásával akár több ezer álprofilt készítenek, melyek jók arra, hogy reklámozzanak termékeket, illetve arra is, hogy segítségükkel célzott oldalakon valótlan követőszámot vagy lájkolásszámot hozzanak létre. Az igazi tulajdonosok számára az is meglehetősen kellemetlen lehet, hogyha a hamis profiljuk által őket olyan termékekkel, reklámokkal hozzák összefüggésbe, amilyeneket nem szeretnének [19].

4.3.3. Adományszámla

Ebben az esetben a kamureg az álprofiljában támogatásokat kér.

Egy megvalósítási lehetőség, hogy a kamureg lemásolja egy tényleg beteg ember profilját, vagy valaki olyanét, aki beteg volt, de meggyógyult, valós adatokkal és képekkel. Azért, hogy a profil hitelesnek tűnjön, és ennek érdekében legyenek ismerősei, készíthet álprofilokat a beteg ember ismerőseiről is, akiket bejelöl ismerősöknek. Ezek után ismerkedik (akár az eredeti profil ismerőseivel is), majd azt állítja, hogy kiújult a betegsége. Célja, hogy érzelmileg befolyásolja az ismerősöket (és azokat, akikkel az ismerősök megosztják az információkat), és egy saját adományszámlára (aminek természetesen semmi köze nincs az igazi beteghez) támogatásokat kérjen.

Egy másik megvalósítási lehetőség, hogy a kamureg ismert emberekről készít másolt álprofilt, és a nevükben gyűjt adományokat különböző célokra, de természetesen saját számlájára [6].

4.3.4. Zsarolás

Ebben az esetben van egy vagy több konkrét áldozat, akit, vagy akiket a kamureg az álprofilján keresztül megzsarol.

A zsarolás eszköze többnyire valamilyen információ, kép vagy videó, amiről nyilvánvaló, hogy az áldozat számára kellemetlen lenne, ha a kamureg azt nyilvánosságra hozná. Ez származhat egy bizalmas viszonyból, ami a kamureg és az áldozat között (korábban) létrejött, vagy az áldozat manipulálá-

sából. A cél többnyire pénzszerzés, de előfordulhat az is, hogy a kamureg az áldozattól valamilyen más jellegű szolgáltatást kér azért, hogy az áldozatot ne szégyenítse meg.

Fontos, hogy nem csak zsarolás áldozatai lehetünk, hanem tudtunk nélkül személyünk, vagy az általunk kiadott információk, felhasználhatók lehetnek mások megzsarolásában is. A következő példa erre is rávilágít.

Az áldozatok manipulálására példaként kivételesen megemlítünk egy ténylegesen megtörtént, tanulságos átverést. A kamureg álprofilt készített Chris "Birdman" Andersen híres kosárlabda játékosról, illetve egy Paris (Roxanne) Dylan nevű 17 éves lányról. Paris-al Chris álprofiljáról, Chris-el pedig a Paris álprofiljáról beszélgetett, akit Chris idősebbnek hitt. Meztelen képeket szerzett róluk, melyeket továbbított is, összehozott kettőjük között egy találkozót, majd Paris anyja nevében pénzt követelt Christől azért, mert „megrontotta a lányát". A rendőrség megtalálta a 17 éves Paris meztelen képeit Chris számítógépén, amiért gyermekpornográfiával vádolták és szakmai karrierje kettétört. A nyomozás végül kiderítette, hogy a kamureg egy 6 osztályt végzett, 30-as éveiben járó Shelly Chartier volt, aki 11 éven keresztül beteg édesanyját ápolta és emiatt nem hagyta el az otthonát, egyetlen kapcsolatát a külvilággal az internet jelentette. Shelly számos további híresség is becsapott, 18 hónap börtönt kapott, amiből 12 hónapot ült le. [4, 13, 15]

4.3.5. Szex, pedofília

Ebben az esetben a kamureg megpróbál mások bizalmába férkőzni, személyes vagy virtuális találkozást létrehozni, és a bizalmat felhasználni vágyai kielégítésére.

Mivel a gyermekek többnyire könnyebben átverhetők, ezért számukra különösen veszélyes lehet, tehát fontos erre külön felhívni a figyelmüket. A gyermekek viselkedésének tesztelése lehetséges úgy, hogy készítünk egy álprofilt és megpróbáljuk őket arról elcsábítani.

4.3.6. Bűncselekmények az áldozat nevében

Ebben az esetben a kamureg az áldozatról készít álprofilt, és azzal követ el bűncselekményeket.

Bár valószínűleg többnyire kiderül majd, hogy az áldozatnak nincs köze az álprofilhoz, azt hosszú nyomozás, gyanúsítgatás előzheti meg, aminek az áldozatra nézve súlyos következményei lehetnek (iskolai problémák, barátok elvesztése stb.).

4.3.7. Az áldozat lejáratása

A kamureg ebben az esetben is az áldozatról készít álprofilt, de a célja most kifejezetten az, hogy ezzel az áldozatnak ártson.

Az intelligensebb megvalósításban a kamureg az áldozat nevében annak kollégáit, tanárait, főnökeit csak finoman sértegeti, vagy indirekt módon kritizálja (például úgy, hogy a kritika az érintettekhez pletyka formájában jusson vissza, vagy közösségi oldalon, de nem a kritizált személy oldalán). A trükk abban van, hogy ha a kritika nem nagyon durva, akkor a célszemélyek nem fogják az áldozatot kérdőre vonni, tehát ő nem is fog tudni a történtektől, csak azt fogja észlelni, hogy a kollégái fokozatosan elhidegülnek tőle és „kiutálják" a közösségből, annak esetleges további következményeivel együtt.

A durvább változatban az áldozat nevében szexuális ajánlatokat tesz, vagy agresszíven sérteget másokat, amiért az áldozatot mások lenézik, utálják, esetleg megtámadják [8].

4.3.8. Internetes zaklatás

Az internetes zaklatás célja az áldozat lejáratása, megalázása, és ismétlődő, felzaklató, kiközösítő, kegyetlenkedő, agresszív, gyűlölködő üzeneteken, hívásokon, internetes kommunikáción keresztül valósiul meg. Ilyen zaklatás része lehet internetes oldal létrehozása az áldozat a lejáratására, azt nevetségessé tevő vagy annak jó hírét, kapcsolatait romboló (valótlan) tartalmak, átszerkesztett személyes képek vagy videók terjesztése, az áldozat által szégyellt vagy titkolt információk közzététele, az áldozat személyes oldalainak, fiókjainak feltörése, módosítása, és annak nevében rossz cselekedetek elkövetése

is. Az internetes zaklatás nem csak álprofillal valósulhat meg, de az esetek körülbelül felében az áldozat nem tudja, hogy ki az elkövető. [17]

4.4. Következmények

Fontossága miatt érdemesnek gondoljuk összefoglalni a legfontosabb lehetséges következményeket.

Társkeresés esetén fontos tisztában lenni azzal a jelenséggel, hogy minél kevesebb információt ismerünk a másikról, a képzelet annál inkább ki fogja tölteni lehetséges valótlan információkkal, ezért annál inkább csalódás lehet a vége. Hogyha egy ilyen kapcsolat hosszabb ideig eltart, akkor miatta más esélyes, valós kapcsolatok is tönkremehetnek. Hogyha a kamureg tisztességtelen eszközöket is bevet, akkor az áldozatnak más kellemetlenségei is lehetnek (bekavar más kapcsolataiba, nem tud a kamuregtől megszabadulni stb.). Nagyon fontos lehetséges következmény, hogy a kamureg az áldozat által kiadott (intim) információkkal visszaélhet. Hogyha az álprofil tulajdonosának szemszögéből nézzük, akkor könnyen megtörténhet, hogy a társa éppen a furcsa, gyanús jelek miatt nem köteleződik el teljesen, hanem közben másik kapcsolatot is kialakít, és a találkozás már késő lesz ahhoz, hogy a kamureget válassza. Továbbá, nagy esély van arra, hogy a társa a találkozásnál a kamureget idegennek fogja érezni (hiszen más képet alakított ki magáról), és ha egyébként elfogadta volna, akkor is annyira csalódott lesz és átvertnek fogja érezni magát, hogy nem akar további kapcsolatot olyannal, akiről az első találkozásnál kiderül, hogy hazudott neki.

Hogyha az álprofil célja kihasználás, rosszindulat, akkor az áldozatnak még komolyabb lehetséges következményekkel kell szembenéznie. Egyrészt bűncselekmények (lopás, szex) áldozata lehet. Másrészt, a tudomása nélkül, a nevében megvalósított sértések miatt megromolhatnak a kapcsolatai, a nevében megsértett személyek komolyabb bosszút is állhatnak rajta (megverik, kiteszik a munkából), illetve a megromlott közösségi viszonyai miatt önként is kénytelen lesz közösséget (munkát, iskolát) váltani. Mindez depresszióhoz, öngyilkossági kísérlethez is vezethet. A történtek miatt az áldozatnak a jövőben is nehézségei lehetnek a későbbi kapcsolataiban, munkakeresésben, hiszen például a munkaadó nem fogja nyomozni, hogy hamis volt-e a profil, amin a nevében szörnyű dolgokat műveltek.

A közösségi oldalak szabályzata tiltja a hamis profilok létrehozását. A közösségi oldalak sok esetben automatikusan is igyekeznek az álprofilokat kiszűrni és törölni, de ennek a célpontjai jelenleg első sorban az üzletszerűen működő oldalak, ezért ha álprofillal találkozunk, akkor ezt célszerű minél gyorsabban jelenteni. Ugyanis, a kamureg is megpróbálhatja letiltatni az általa lemásolt eredeti profilt, és ennek sikere attól is függ, hogy a közösségi oldal mennyire lelkiismeretesen fogja kezelni az esetet.

Az álprofil elsődlegesen polgári jogba ütköző jogsértést valósíthat meg, melynek szankciója a jogsértéstől való eltiltás mellett a sérelemdíj fizetésére kötelezés lehet. Ehhez azonban a sértettnek kell perelnie, és ehhez sok esetben neki kell megtalálnia az elkövetőt. Hogyha viszont az álprofillal már bűncselekményt is elkövetnek (például rágalmazás, zaklatás), akkor annak büntetőjogi szankciói is lehetnek. Bármilyen későbbi lépés érdekében célszerű az oldalt (képernyőképet) még annak jelentése (és remélhető eltávolítása) előtt lementeni. Hogyha jogi eljárást szeretnénk indítani, akkor be kell szereznünk egy közjegyzői ténytanúsítványt arról, hogy abban a pillanatban a jogsértő tartalom azon az oldalon szerepelt [6, 11, 16].

5. Védekezés a visszaélések ellen

A leírt trükkök ismerete segít abban, hogy hasonló átverési kísérletek esetén gyanakodjunk, ugyanakkor érdemes összefoglalni a legfontosabb lehetséges védekezési lépéseket is.

5.1. Biztonsági intézkedések

Az álprofilokkal megvalósítható visszaélések elleni védekezés során is szükséges az alapvető biztonsági intézkedések betartása.

Gondoskodjunk adataink védelméről, közösségi oldalainkat védjük erős jelszóval, és használjuk ki a közösségi oldalak más programok által nyújtott biztonsági funkciókat (például adatvédelmi beállítások, kétfaktoros hitelesítés). Jelszavunkat semmilyen körülmények között ne adjuk ki másnak, még a „szerelem hevében" sem.

Gondoljunk bele abba, hogy mennyire bajban leszünk akkor, ha a kamureg bejut a profilunkba, megváltoztatja a jelszavunkat, nekünk írni sem enged oda, így nem tudjuk értesíteni az ismerőseinket, ugyanakkor az előző fejezetben leírt cselekményeket az igazi profilunkkal hajtja végre. Természetesen ezt lehet és kell is jelenteni, de amíg a profil letiltásra kerül, addig a kamureg visszafordíthatatlan károkat okozhat nekünk.

5.2. Védjük magunkat és adatainkat

Minél kevesebb személyes információt osztunk meg, annál kevesebb támpontja lesz annak, aki vissza akar élni az adatainkkal. Egy álprofil készítésénél a kamuregnek még olyan adatok az is hasznosak lehetnek, amikkel egyébként nem lehet nekünk ártani, ezért megosztásainkat is korlátozzuk azokra, akikkel tényleg közölni szeretnénk a tartalmat. Olyan információkat pedig végképp ne tegyünk közzé, amiket mások felhasználhatnak ellenünk.

Hogyha nem vagyunk biztosak az ismerősünk kilétében, első alkalommal kizárólag nyilvános helyen találkozzunk vele, ahol nem eshet bajunk, továbbá jó, hogyha valakinek még tudomása van a találkozás részleteiről.

Hogyha visszaélést tapasztalunk, akkor ne hagyjuk magunkat zsarolni, hanem jelentsük az esetet a közösségi oldal üzemeltetőjének és a rendőrségen.

5.3. Regisztráció közösségi oldalakon

Sokan azért nem regisztrálnak közösségi oldalakra, mert úgy gondolják, hogy akkor jobban védve vannak a közösségi oldalak segítségével elkövetett visszaélésekkel szemben.

Hogyha végiggondoljuk a *Kihasználás, rosszindulat* alfejezetben leírtakat, a visszaélésnek egyik esetben sem feltétele az, hogy az áldozatnak igazi profilja legyen. Hogyha a kamureg tud szerezni valahonnan egy fényképet az áldozatról, és van némi információja is róla, akkor a kamuprofilt igazi hiányában is el tudja készíteni. Sőt, ebben az esetben a kamureg könnyebben be tudja cserkészni az áldozat igazi ismerőseit, akik azt fogják hinni, hogy az áldozat végre regisztrált a közösségi oldalon, míg hogyha annak már lenne egy profilja, akkor talán akadna az ismerősök között olyan, aki gyanakodni kezdene, és legalább megkérdezné (ezzel pedig értesítené) az áldozatot.

5.4. Kommunikáció

Első megközelítésben furcsa lehet ebben a fejezetben a kommunikáció fontosságát hangsúlyozni. Pedig bizony fontos, ugyanis a nevünkben elkövetett cselekményekről többnyire csak akkor fogunk értesülni, ha valaki jelzi nekünk, ehhez viszont olyan viszonyban kell legyünk a – nem csak közösségi oldalas, hanem valós életbeli – ismerőseinkkel is, hogy részünkről szokatlan megnyilvánulások esetén ne vonakodjanak megbeszélni velünk a tapasztaltakat.

5.5. Visszajelölés

Lehetőleg ne jelöljünk vissza ismeretleneket. Persze lehet olyan, hogy valaki tényleg ismer, csak nem emlékszünk rá, vagy sok év után nem ismerünk rá, ilyen esetben érdemes lehet visszakérdezni, hogy honnan is ismerjük egymást.

Hogyha úgy gondoljuk, hogy a visszakérdezés túlságosan kínos helyzetbe hozhat, vagy a válasza nem volt eléggé meggyőző, de a jó viszony érdekében inkább visszajelöljük az illetőt, akkor legalább azt tegyük meg, hogy helyezzük el egy gyanús ismerősök számára készített felhasználói listában, a megosztásoknál figyeljünk arra, hogy ezzel a listával minél kevesebb információt osszunk meg, közben meg figyeljük ezen ismerősök tevékenységét.

5.6. Mikor gyanakodjunk álprofilra?

Hogyha valakiről kevés adat van, vagy kevés ismerőse van, az gyanús. Lehet persze, hogy – éppen a fentebb említett elővigyázatosságból – csak kevés adatot közöl magáról, és ismerősei azért nincsenek, mert új a profilja, de azért nem árt óvatosnak lenni.

Bejelölésnél, vagy később kérdezzünk rá azoknak a körülményeknek a kevésbé nyilvánvaló vagy közismert részleteire is, ahonnan a másik állítólag ismer. Hogyha ezeket nem ismeri megfelelően, akkor joggal gyanakodhatunk álprofilra.

Érdemes megvizsgálni a másik által feltöltött képeket is. Hogyha a profil tulajdonosának vannak közös képei (különösen beteggelt) más személyekkel, akkor ez jó jel, viszont vegyük figyelembe, hogy a beteggelt ismerősök is lehetnek álprofilok, illetve álprofilon is el lehet helyezni (akár beteggelve) egy máshonnan lemásolt képet is. Hogyha a (profil)képek rosszabb minőségűek, ez lehet annak is a következménye, hogy sokszor voltak (más profilokról) le-fel töltögetve. Legyen gyanús továbbá az is, hogyha a képek többsége nyilvánvalóan azt a célt szolgálja, hogy imponáljon a másik nemű felhasználóknak (drága autó, kihívó női profilkép stb.).

Gyanús esetben érdemes a profil tulajdonos képeire rákeresni a Google képkeresőben, a https://www.tineye.com/ oldalon vagy más képkeresőkben. Hogyha ugyanaz a kép több oldalhoz is tartozik, akkor vagy a profil tulajdonosa lopta a képet, vagy róla másolták, ezt ki kell deríteni.

Hogyha a profil ismerőseinek nemi aránya nagyon eltorzult, akkor lehet arra gyanakodni, hogy ez nem egy valós profil, hanem kizárólag ismerkedésre lett létrehozva. Vegyük figyelembe azt is, hogy népszerű emberek, nagyon vonzó pasik vagy nők, ritkán keresnek társat közösségi oldalon. Hogyha társkeresés esetén a profil tulajdonosa hosszú időn keresztül nem akar találkozni, vagy videochatelni, különböző ürügyekkel (elromlott a kamera, rossz az internet kapcsolat, minden találkozásnál lebetegedik ő vagy valamelyik hozzátartozója stb.), akkor nagyon valószínű, hogy valami rejtegetnivalója van. Hogyha telefonon sem akar velünk beszélni, akkor akár a neme vagy az életkora is lehet más, mint aminek mondja magát.

Érdemes megnézni a profil tevékenységét is. Hogyha az üzenőfalon kevés bejegyzés található, vagy ezek tipikusan reklámok, vagy a profil tulajdonosa gyakran ír mások üzenőfalára reklámszövegeket, akkor valószínűleg (ál)profillal van dolgunk.

Akkor is gyanakodjunk álprofilra, hogyha egy ismerősünk adatokat akar rólunk megszerezni, például egy teszt kitöltetésével.

5.7. Nyomozás

Leírunk néhány ötletet, melynek segítségével megpróbálhatunk információkat szerezni a kamuregről.

Hogyha megvan a kamureg telefonszáma, akkor meg lehet próbálni rákeresni, hogy az kihez tartozik, bár hogyha a szám nem nyilvános, akkor nem fogjuk megtalálni. Vegyük figyelembe továbbá, hogy ha megtaláljuk, a szám akkor is lehet a kamureg valamely rokonának nevére regisztrálva.

Érdemes megnézni a kamureg azonosítóját (többnyire az URL sorban megtaláljuk), és megnézni az adott azonosítóval rendelkező oldalakat más közösségi hálózatokon. Előfordulhat ugyanis, hogy a kamureg mindenhova ugyanazzal az azonosítóval regisztrált, de csak az egyikből csinált álprofilt például más képpel és névvel.

Megpróbálhatjuk megtudni a kamureg tartózkodási helyét is a következőképpen. Valamilyen IP Logger programmal (pl. https://grabify.link/) létrehozunk egy linket az álprofil oldalhoz, a kapott linket elküldjük a kamuregnek azzal a kérdéssel, hogy az-e a közösségi oldala, aki remélhetőleg rákattint, az IP Logger programban pedig megnézzük az IP címet és az annak megfelelő tartózkodási helyet.

Érdemes számításba venni azt is, hogy a kamureg lehet egy valódi ismerősünk is (persze más név alatt), vagy olyan valaki, akit nekünk ismerőseként tűntetett fel. Ilyen és hasonló esetekben érdemes lehet elemezni a kamureg ismerőseit, az azokkal kapcsolatos aktivitásait (például https://www.facebook.com/x?and=y).

Megkérdezhetünk a közösségi oldalon a feltételezett kamureghez közelinek tűnő barátokat arról, hogy a valóságban találkoztak-e vele, és egyes feltételezett adatokról mit tudnak, de érdemes számításba venni, hogy a barát lehet „beavatott", vagy akár a kamureg által készített másik álprofil is.

Hogyha a képkereső nem dobott ki semmi gyanúsat, de mi mégis gyanakszunk, a fénykép ellenőrzésére kérhetünk a kamuregtől olyan képet, amit a valóságban könnyű, képszerkesztővel viszont nehéz elkészíteni. Például, kérhetjük tőle, hogy egy általunk kért szöveget kézzel írjon le egy papírra, gyűrje össze, nyissa ki, fényképezze le és úgy küldje el.

Hogyha a kamureg azt látja, hogy gyanakszunk, elképzelhető, hogy a profiljáról letilt minket, hogy nehezítse a nyomozást. Ilyenkor megkérhetjük egy barátunkat, hogy ismerkedjen meg vele, vagy létrehozhatunk mi is egy álprofilt, amivel bejelöljük és esetlegesen lebuktatjuk.

Végül, az is megtörténhet, hogy a gyanús jelek ellenére a profiltulajdonos mégsem kamureg, lehet, hogy vannak olyan körülmények, amiket mi nem látunk át, ezért addig ne hozzunk végleges döntést, amíg nem bizonyosodtuk meg a valóságról.

6. Mit tanulhatunk mindebből?

1. Az álprofil létrehozása etikátlan, szabálytalan és jogsértést valósít meg. Az, hogy mennyire elítélendő, attól függ, hogy a kamureg mire használja.
2. Az álprofilos átveréseknek nagyon sok fajtája, az ezek elleni védekezéseknek pedig számos lehetősége van, melyeket alkalmazni kell, és melyekben alapvető fontosságú az adatvédelmi és biztonsági intézkedések betartása.
3. Az álprofilos cselekedeteknek úgy a kamureg, mint az áldozat szempontjából súlyos következményei lehetnek.
4. Nagyon fontos, hogy a minket ért kisebb vagy nagyobb sérelmeket az (állítólagos) elkövetővel megfelelő módon, a másikra odafigyelve, megbeszéljük. Mindenkinek érdeke, hogy ne legyenek indokolatlan ellentétek, és együttműködjünk az esetleges visszaélések kivédésében.
5. Az internetes társkeresés során vannak olyan kevésbé nyilvánvaló szempontok, melyeket fontos figyelembe venni annak érdekében, hogy csökkentsük a csalódásunk és mások indokolatlan megbántásának az esélyét. Általában társkeresésre sem érdemes álprofilt készíteni, mert ezzel nem csak másokat, hanem magunkat is becsapjuk.
6. A kamureg nem feltétlen rossz ember, lehet, hogy csak menekül a valóságból, ahol bántják, és nem kap (megfelelő) támogatást. Ugyanakkor, mindenki jobban járna, ha nem készítene álprofilokat. Tehát, nem csak emberségünk követeli meg, de például ezért is közös érdekünk, hogy ne hagyjuk, hogy valakit érdemtelenül bántsanak.
7. A valóság nem fehér vagy fekete, ugyanazt más nézőpontokból is érdemes megvizsgálni, ami újszerű felfedezésekhez vezethet.

7. Módszertani következtetések

Az etikus és biztonságos internethasználat tanításának valószínűleg a legnagyobb kihívása az, hogy nem elegendő az ismereteket átadni, hanem meg is kell győzni a diákokat arról, hogy az ismereteket megfelelően használják akkor is, amikor a tanár ezt nem látja.

A cikk során láttuk, hogy mennyi következménye lehet az álprofilos visszaéléseknek, tehát az nyilvánvaló, hogy az etikus és biztonságos internethasználat tanításának keretében erről is beszélnünk kell. Egy lehetséges megközelítés, hogy a beszélgetést eleve az álprofilokkal kezdjük, amit valószínűleg a diákok sokkal érdeklődőbben fognak fogadni, mintha az a benyomásuk lenne, hogy szabályokat akarunk rájuk zúdítani. A példákkal bemutatott álprofilos történetek érzelmileg is hathatnak a diákokra, melynek következtében lehet remélni, hogy befogadóbbak lesznek, és az ezek során megbeszélt ismereteket mélyebben, hosszabb távra meg fogják jegyezni, és alkalmazni fogják.

Az előző fejezetben láttuk azt is, hogy milyen sokféle tanulsága lehet az álprofilok megbeszélésének, melyek között vannak olyanok is, amik az internetes biztonság témakörén is túlnyúlnak, ugyanakkor rendkívül fontosak a gyermekek nevelésében, és más kompetenciákat is fejlesztenek. Például, annak elsajátítása, hogy a gyerekek képesek legyenek ugyanazt különböző nézőpontokból is elemezni, a kreativitásnak is egyik fontos feltétele [14].

Ugyanakkor, vannak korlátai is a módszer használatának. Egyrészt, az etikus és biztonságos internethasználat tanítását már korábbi korosztályokban el kell kezdeni, mint amikor az álprofilokról ilyen szinten érdemes beszélni, tehát az információk álprofilok kapcsán történő megbeszélése azok kiegészítésére és megerősítésére használható. Másrészt, az álprofilokkal kapcsolatosan elmondható információk csak töredékét képezik az etikus és biztonságos internethasználat témájában tanítandó tananyagnak. Viszont, az álprofilok megbeszélése, a fenti tanulságokon keresztül, természetes módon vezethet át az etikus és biztonságos internethasználat más aspektusainak megbeszélésére.

Felmerül a kérdés, hogy ha a diákoknak megmutatjuk a lehetséges álprofilos (és úgy általában az internetes) visszaélési lehetőségeket, akkor nem történhet-e meg, hogy ezzel a tudással ők is visszaéléseket fognak elkövetni? Természetesen ezt nem lehet kizárni, de mivel mások úgyis elkövetnek ilyen visszaéléseket, ezért fontos, hogy diákjainkat felvértezzük ezekkel az ismeretekkel ahhoz, hogy védekezni tudjanak. Továbbá, nyilvánvalóan igyekeznünk kell meggyőzni a diákokat arról, hogy nem érdemes ilyen visszaéléseket elkövetni, mert ezzel hosszabb távon rosszul fognak járni.

Végül megjegyezzük, hogy az etikus és biztonságos internethasználat témakör tanításának sok érdekes megközelítése lehet, ebből bizonyos ismeretek álprofilok megbeszélésén keresztül történő tanítását nem az egyetlen, hanem – a fentiekben leírtak miatt – egy jó megközelítésnek gondoljuk a sok közül.

Irodalom

1. Gayer Zoltán, Balog Barabás Tibor: Flinder Boyd: Adatbiztonság, adattudatosság a közösségi hálózatokban, Médiakutató 12. évf. 3. sz., 2011, http://epa.oszk.hu/03000/03056/00044/EPA03056_mediakutato_2011_osz_01.html (utoljára megtekintve 2019.02.06.)
2. Alan S. Weber: Considerations for social network site (SNS) use in education, International Journal of Digital Information and Wireless Communications (IJDIWC) 2(4): 37-52, The Society of Digital Information and Wireless Communications, 2012 (ISSN: 2225-658X), http://www.academia.edu/8343492/CONSIDERATIONS_FOR_SOCIAL_NETWORK_SITE_SNS_USE_IN_EDUCATION (utoljára megtekintve 2019.02.06.)
3. S Adikari, K Dutta: Identifying Fake Profiles in LinkedIn, Pacific Asia Conference on Information Systems (PACIS), 2014, https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1110&context=pacis2014 (utoljára megtekintve 2019.02.06.)
4. Flinder Boyd: The Birdman's Vengeful Ghost, Newsweek, 2014.05.28., https://www.newsweek.com/2014/06/06/birdmans-vengeful-ghost-252517.html (utoljára megtekintve 2018.10.29.)
5. Fehér Katalin: Milyen stratégiák mentén épül fel a digitális identitás? Feltáró kutatási szakasz: a munkavállalás előtt álló egyetemisták, Médiakutató, 15. évf. 2. sz., 139-154, 2014,

http://epa.oszk.hu/03000/03056/00055/pdf/EPA03056_mediakutato_2014_nyar_139-154.pdf (utoljára megtekintve 2019.02.06.)

6. Bella Brigitta: Milliókat is fizethet egy hamis Facebook-profil miatt, 2015.04.30., origo.hu, http://www.origo.hu/teve/20150421-sok-magyar-hiressegnek-van-alprofilja-a-kozossegi-oldalakon.html (utoljára megtekintve 2018.11.02.)
7. MTV: Kamureg, http://www.mtv.co.hu/kamureg, (utoljára megtekintve 2018.11.04.)
8. MTV: Catfish: the tv show | 412: falesha & jacqueline, 2015.07.21., http://www.mtv.co.uk/catfish-the-tv-show/videos/catfishthe-tv-show-412-falesha-jacqueline (utoljára megtekintve 2018.11.02.)
9. Monica T Whitty: Anatomy of the online dating romance scam, Security Journal, October 2015, Volume 28, Issue 4, pp 443–455, Palgrave Macmillan, https://link.springer.com/article/10.1057/sj.2012.57 (utoljára megtekintve 2019.02.06.)
10. A. Madhavi ,V. Surya Narayana Reddy: Automated detection of fake profiles using simple framework: SVM, International Journal of Advance Computing Technique and Applications (IJACTA), ISSN : 2321-4546, Vol 4, Issue 1, June, Pages 176-181, 2016, http://ijacta.in/index.php/ojs/article/view/46/38 (utoljára megtekintve 2019.02.06.)
11. Magyarorszag.hu: A személy elleni bűncselekmények (szabadság, emberi méltóság elleni bűncselekmények), 2016. augusztus 18., https://ugyintezes.magyarorszag.hu/ugyek/410006/420012/420013/A_szemely_elleni_buncselekmenyek_szabadsag_emberi_meltosag.html?ugy=adatvedelem.html#topicissue (utoljára megtekintve 2018.11.04.)
12. Ali M. Meligy, Hani M. Ibrahim, Mohamed F. Torky: Identity Verification Mechanism for Detecting Fake Profiles in Online Social Networks, I. J. Computer Network and Information Security, 2017, 1, 31-39, DOI: 10.5815/ijcnis.2017.01.04, http://www.mecs-press.org/ijcnis/ijcnis-v9-n1/IJCNIS-V9-N1-4.pdf (utoljára megtekintve 2019.02.06.)
13. Andrew Paparella, Marc Dorian, Jonathan Balthasar Lauren Effron: How NBA star, aspiring model became victims of a massive catfishing scheme out of Canada, ABCNews, 2017.04.13., https://abcnews.go.com/Technology/nba-star-aspiring-model-victims-massive-catfishing-scheme/story?id=46755887 (utoljára megtekintve 2018.10.29.)
14. Dr. Gyarmathy Éva: Kreatív gyermek, kreatív felnőtt, Pedagógiai Esték, Szeged, 2017.05.02., https://www.youtube.com/watch?v=fz2iQMyuwys&t=116s (utoljára megtekintve 2018.10.31.)
15. Lauren Weigle: Shelly Chartier Vs. Chris Andersen & Paris (Roxanne) Dylan on 'Catfish', 2017.05.17., https://heavy.com/entertainment/2017/05/shelly-chartier-catfish-chris-andersen-paris-roxanne-dylan-instagram/ (utoljára megtekintve 2018.10.29.)
16. Dr. Danyi Richárd: Így védekezzünk az álprofilok ellen!, rtl.hu, 2017.07.31., http://rtl.hu/rtlklub/reggeli/igy-vedekezzunk-az-alprofilok-ellen (utoljára megtekintve 2018.11.02.)
17. Tari Annamária: Bullying – mit tehet a gyerek, a szülő, a tanár?, Pedagógiai Esték, Szeged, 2017.09.25., https://www.youtube.com/watch?v=fo6sK5BY960 (utoljára megtekintve 2019.02..10.)
18. Devakunchari Ramalingam, Valliyammai Chinnaiah: Fake profile detection techniques in large-scale online social networks: A comprehensive review, Computers & Electrical Engineering, Volume 65, January 2018, Pages 165-177, https://www.sciencedirect.com/science/article/pii/S004579061731279X (utoljára megtekintve 2019.02.06.)
19. Berta Sándor: Hatalmas üzlet az álprofilok létrehozása, sg.hu, 2018.01.30., https://sg.hu/cikkek/it-tech/129517/hatalmas-uzlet-az-alprofilok-letrehozasa (utoljára megtekintve 2018.11.02.)

Agilis módszertan kutatás-fejlesztés laborban

Ilyés Enikő

ilyese@inf.elte.hu
ELTE IK

Absztrakt. Az agilis módszertanok iparban való elterjedtségét látva felmerülhet a kérdés: eredményesen alkalmazhatók ezek a módszerek az egyetemi kutató-fejlesztő munka keretében végzett szoftverfejlesztés esetében is? Mivel korábban sikeresen próbálkoztunk a Scrum használatával egy szoftverfejlesztést bemutató egyetemi kurzuson, 2018-ban megkíséreltük az agilis módszertanok alkalmazását a „txtUML" kutató-fejlesztő labor keretei között is. Ebben a cikkben e sajátos módszerünket mutatjuk be, rávilágítva azokra a kihívásokra, melyeket a végzett munka kutatás-fejlesztő jellege támasztott egy klasszikus szoftverfejlesztés gyakorlattal szemben.

Kulcsszavak: agilis, kutatás-fejlesztés labor, csapatmunka

1. Bevezetés

Az Eötvös Lóránd Tudományegyetem kutatóegyetem jellegének megfelelően az intézmény Informatikai Karán is több kutatás-fejlesztés csoport működik. Ezek keretén belül az egyetem munkatársai és hallgatói is rész vesznek kutatómunkában. A hallgatóknak van lehetőségük önkéntesen, ösztöndíj ellenében vagy tanulmányi kredit érték megszerzése céljából bekapcsolódni e laborok működésébe.

Egy-egy kutatás-fejlesztő labor fókuszában általában egy nagyobb termék áll, melynek fejlesztése több kutatási kérdést vet fel. A kutató csoport tagjai ezt a terméket fejlesztik olyan módon, hogy egy-egy kutatási kérdést, -ágat egy-egy csoporttag/alcsoport jár jobban körbe, meglátásait megvitatja a többiekkel, majd eredményeit visszacsatolja a közös nagy termékbe.

A „Model Driven Development" kutatócsoport 2014 óta működik az Eötvös Loránd Tudományegyetemen. Egyik fő projektje a txtUML nevet viseli. A kivitelező csoport tagjainak rövid leírása a termékről: „A txtUML név egy mozaikszó, jelentése textual, executable, translatable – vagyis szövegalapú, végrehajtható és fordítható – UML. Az eszköz egy nyílt forráskódú szoftver, melynek célja a modell-vezérelt alkalmazásfejlesztés támogatása.

Szöveges (ld. forráskód) és grafikus (ld. UML diagramok) modellezési módszerek előnyeit egyesíti. Közvetlenül végrehajtható, diagramokon megjeleníthető és debuggolható, valamint más programozási nyelvekre fordítható UML modelleket lehet definiálni segítségével. A modellezés jól használható a szoftverfejlesztés korai, prototipizálási szakaszában. A támogatott nyelvek esetén az elkészült modellek akár eredeti formájukban is beágyazhatók meglévő rendszerekbe. Ilyen módon az alapul vett paradigmát akár a szoftverfejlesztés következő, általános célú programozási nyelvek feletti absztrakciós szintjeként is értelmezhetjük." [1]

A txtUML csoport működésére az volt jellemző a megalakulást követő időszakban, hogy az ötletadó (Dévai Gergely) nagy lendülettel toborzott maga köré néhány hallgatót (4-5), akik rendszerint szakdolgozat, diplomamunka, TDK dolgozat megvalósítása keretében dolgoztak a termék egy-egy ágán. Ilyen módon a csoporttagok nagyon motiváltak voltak, az irányítás pedig erős, az ötletadónál centralizálódott. 2016-ban az ötletadó távozása és a más motivációval rendelkező csapattagok érkezése megváltoztatta a csoport működésének jellegét. A hallgatók nagy része „Szoftvertechnológia labor" tárgy teljesítése céljából kezdett el a csoportban dolgozni, ami azt eredményezte, hogy: fél évente sokkal több új tag csatlakozott a csoporthoz; akik projekt iránti elköteleződésére alapvetően fél év távlatában lehetett számítani (a tárgyfelvétel fél évre szól); és 4 kredit értéknek megfelelő munka volt tőlük

elvárható (4 x 30 óra, azaz félévente 120 óra). A korábban alkalmazott, erős motivációra, hosszútávú elköteleződésre és erősen centralizált vezetésre alapuló módszer már nem bizonyult a munkaszervezés hatékony eszközének.

Mivel korábban sikeresen alkalmaztunk Scrum módszert klasszikus szoftverfejlesztés gyakorlati órán, az az ötlet született, hogy kezdjünk el agilis módszereket alkalmazni a txtUML kutatócsoport munkájának megszervezése céljából is. Az újszerű munkaszervezés megtervezése során azonban nyilvánvalóvá vált a kutatás-fejlesztés labor néhány olyan sajátossága, mely kétségessé tette, hogy a szoftverfejlesztés gyakorlat során már bevált módszerek módosítás mentes átörökítése lenne a leghatékonyabb.

1.1. Kutatás-fejlesztés labor sajátosságai a szoftverfejlesztés gyakorlattal szemben

A kutatás-fejlesztés labor legkiemelkedőbb jellemzője a szoftverfejlesztés gyakorlattal szemben már a megnevezésben szembetűnik: maga a kutatás. Míg a fejlesztési feladatok térben és időben viszonylag jól körülhatárolhatók, becsülhetők, a kutatási feladatok sokkal nagyobb szabadságot kívánnak és több bizonytalanságot rejtenek. Ebből kifolyólag csak fölösleges nyomást okozhat az, ha szigorúan rögzített időkeretek közé próbáljuk szorítani teljesítésüket. Eszerint, míg a szoftverfejlesztés gyakorlat során alkalmazott Scrum és ezáltal a futamok rögzítettsége lendületet, ritmust és motivációt ad a csoporttagok számára, a kutatás-fejlesztés labor esetében ez csak plusz terhet rak rájuk és korlátozza a kutatás esetében fontos egyéni szabadságot. Mivel korábban sikeresen alkalmaztunk Scrum módszert klasszikus szoftverfejlesztés gyakorlati órán, az az ötlet született, hogy kezdjünk el agilis módszereket alkalmazni a txtUML kutatócsoport munkájának megszervezése céljából is. Az újszerű munkaszervezés megtervezése során azonban nyilvánvalóvá vált a kutatás-fejlesztés labor néhány olyan sajátossága, mely kétségessé tette, hogy a szoftverfejlesztés gyakorlat során már bevált módszerek módosítás mentes átörökítése lenne a leghatékonyabb.

Egy másik sajátosság, hogy a szoftverfejlesztés gyakorlat hallgatói fél évet/egy évet vannak együtt, mindenki egyszerre csatlakozik a csoporthoz és egyszerre válik ki a csoportból a tárgy első, illetve utolsó óráján. A terméket általában az ötlettől kezdve ők építik fel, meglévő kódbázisra nem alapoznak. A kivitelezni kívánt szoftver teljességben megvalósítható a tárgy időkeretein belül. Ugyanez nem mondható el a kutatás-fejlesztés labor kapcsán. A hallgatók rendszerint egy már meglévő kutatócsoporthoz csatlakoznak és egy nagyobb kódbázissal való megismerkedéssel kezdik a munkát. Nem belátható, hogy mettől-meddig maradnak a kutató csoport tagjai (a minimum rendszerint fél év, maximum pedig a hátralévő tanulmányi éveik száma). Legtöbben nem látják a kutatás témáját egészében kibontakozni, csak a kutatás egy szakaszának lehetnek tanúi. Munkaszervezés szempontjából ez azért fontos, mert erőteljesen tekintettel kell legyünk arra, hogy a csapat egy-egy tagja saját folyamatait, szakaszait (pl. becsatlakozás a csapatba) kell összehangolja a teljes csapat folyamataival és ehhez kellő segítséget és motivációt szükséges biztosítanunk számára.

Látni kell azt is, hogy a szoftverfejlesztés gyakorlat tárgy hallgatói általában hasonló szintű előképzést követően kerülnek egy csoportba és a korábban elsajátított elméleti anyag gyakorlattá váltása a feladatuk. A kutatócsoportba elég különböző tudással rendelkező hallgatók kerülhetnek be és motivációjuk az elméleti és gyakorlati tudásban való fejlődés is lehet. Amikor tehát munkaszervezés módszert választunk a kutatás-fejlesztés labor számára, kellő lehetőséget kell biztosítsunk arra, hogy a tagok tanulhassanak egymástól, akár közösen gyarapodjanak elméleti és gyakorlati tudásban is.

Katrin Greßer és Renate Freisler „Agilis és sikeres vezetés" [1] című könyvéből olvashatjuk: „A vezetés nem más, mint csapatom számára keretet, s ezáltal biztonságot nyújtani, ugyanakkor a csapattagok kibontakozásához kellő szabadságot biztosítani."; „A vezetők moderátorként funkcionálnak. A csapat kollektív intelligenciájára, szakértelmére és kompetenciáira támaszkodnak." Tekintve a korábban bemutatott sajátosságokat, beláthatjuk: az agilis vezetés találó lehet a kutatás-fejlesztő csoport

jellegéhez, mindössze a megfelelő (szoftverfejlesztési gyakorlat esetében alkalmazottól némileg eltérő) munkaszervezési kereteket kell kialakítanunk.

2. Szakirodalmi kitekintés

Az agilis módszertanok egyetemi környezetben való alkalmazására részben klasszikus szoftverfejlesztési gyakorlatok módszertanának bemutatása kapcsán találunk példákat. Az Iowai Egyetem [2], Texasi Egyetem [3] oktatói Scrum módszer alkalmazásáról számolnak be. A Calgary Egyetemen [4] már Kanban és Lean módszerrel is bővül a paletta. A Marylandi Egyetemen [5] extrém programozással ötvözték a Scrumot, a Zurichi Egyetemen [6] hasonlóan. Saját Scrum-os tapasztalatainkat az „Esettanulmány: Agilis szoftverfejlesztés egyetemi kurzuson" című cikkünkben foglaltuk össze. [7]

A másik szakirodalmi csoport az agilis módszertanok kutatás-fejlesztés csoportban való alkalmazását taglalja. A „researchgate" honlapon [9] fórum keretében jelzik többen, hogy érdeklődnek Scrum módszer kutatócsoportban való használata iránt, vagy éppen már ki is próbálták. A vélemények megoszlanak a Scrum módszer és a kutatás-fejlesztés projekt jellegének összeegyeztethetőségét illetően. A Cseh Technikai Egyetem oktatója [10] a Scrum használatát az ipari partnerekkel való együttműködés szempontjából találta praktikusnak. A Minho Egyetem iFloW porjektjében [11] ugyancsak ilyen céllal alkalmaztak Scrum-ot, viszont ők UML diagramok hangsúlyos használatával bővítették a módszertant, mert ezek értékes hozadékáról nem kívántak lemondani. A Lavras Egyetem [12] hét projekt kapcsán is hasznosnak nyilvánult a Scrum használata, bár ennek kimutatására még nem álltak rendelkezésre megfelelő mérőszámok. A legnagyobb kihívás viszont már megfogalmazásra került: a csapattagokban való megfelelő hozzáállás (kommunikáció, együttműködés, közös felelősségvállalás) kialakítása.

3. Saját módszer

Ebben a fejezetben azt a sajátos agilis módszertant mutatjuk be, melyet a txtUML kutatás-fejlesztés laborban alkalmazunk 2018 tavaszi félévében. Ebben az időszakban összesen kilenc hallgató volt a csoport tagja, ezek közül három senior hallgató – vagyis olyan hallgató, aki már több mint egy éve dolgozik a kutató csoportban. Három hallgató teljesen új volt a csapatban, további három pedig már fél évvel korábban csatlakozott. A fejlesztő csapat munkáját egy projekt felelős (dr. Gregorics Tibor) és jómagam, mint módszertan felelős segítettük.

A továbbiakban szerepkörök, események és termékek mentén ismertetjük az alkalmazott módszert. Minden elem esetén tisztázzuk, hogy milyen célt szolgált.

3.1. Szerepkörök

Csapatunkban öt szerepkör érvényesül: projekt vezető, Scrum mester, technikai vezető, alcsoport vezető, fejlesztő.

Projekt vezető: Képviseli a kutatócsoport céljait és létjogosultságát az egyetem vezetősége felé. Kapcsolatember a csoport eredményeinek széleskörű terjesztése érdekében (például: az egyetemi „Nyílt nap", „Kutatók éjszakája" rendezvények esetén). Részt vesz a csoport találkozóin és ezen kívül is könnyen elérhető a csoport munkájával kapcsolatos döntések meghozása szempontjából. Szerepköre által biztos keretet és irányt ad a kutatócsoport működésének.

Scrum mester: A szerepkör megnevezése a Scrum módszertanból inspirálódott. A csoport működési értékeinek a védelmét hivatott biztosítani. Ezt különböző események kezdeményezésével, koordinálásával és példamutatással képes elérni. (például: kezdeményezi és koordinálja a napi Scrumot a hatékony megbeszélések érdekében; tréningeli a tagokat az elakadások hatékony jelentésére, stb.) Részt vesz a csoport találkozóin és figyelemmel kíséri az egyes tagok csapatba való működésének folyamatát, valamint az egész csoport működésének dinamikáját.

Technikai vezető: Átfogó képe van a szoftverről, ugyanakkor sok részletben is jártas. Ő fogja össze a csapattagok munkáját technikai szempontból. Minden fontos technikai döntés vele is egyeztetendő. Mentoráló szerepe is van, vagyis arra hivatott, hogy átadja a többi fejlesztőnek a tudását, tapasztalatát, főleg az őket érintő feladatok kapcsán. Tetszés szerint fejlesztési-kutatási feladatokat is vállalhat. Elsődlegesen a csoport találkozóin elérhető, de gyakran ezen kívül, online kommunikáció által is válaszol a fejlesztők kérdéseire.

Alcsoport vezető: A txtUML csoport tevékenysége kapcsán négy különböző kutatás-fejlesztési feladatcsoport különíthető el. A következő megnevezésekkel hivatkozunk témájukra: nyelvi front-end; megjelenítés; C++ export; modelltesztelés. Azonos témakörű feladatokon dolgozó hallgatók alcsoportot alkotnak, melyet egy senior hallgató vezet, aki a témakörben leginkább otthon van. Ő jelöli ki azokat a kisebb kutatási-fejlesztési kérdéseket, melyen egy-egy csapattag dolgozhat és mentorálja a fejlesztőket ebben a folyamatban a csoport találkozók és online kommunikáció során. Rendszerint maga is fejleszt. A csoport találkozói alkalmával bemutatót és vitát szervezhez a saját témakörét illetően, hogy átadja tudását illetve alcsoportja eredményeit visszacsatolja a teljes kutatócsoporthoz.

Fejlesztő: A csapatba való csatlakozás első lépéseként minden új tag megismerkedik a txtUML-el felhasználói nézőpontból. Ezt követően kiválasztja azt a területet, mely kapcsán szeretne hozzájárulni a kutató-fejlesztő munkához és ezáltal egy alcsoport tagja lesz. Alcsoport vezetőjétől feladatokat kap, melyeken egyénileg (az alcsoport vezető mentorálásával) dolgozik. Részt vesz a csoport találkozóin, ahol figyelemmel kíséri a csoport haladását, tanul a többiektől és ő is átadja tapasztalatait. Kutató-fejlesztő munkával járul hozzá a szoftver termék bővítéséhez, finomításához.

3.2. Események

A txtUML kutatócsoport tagjaival hetente találkozunk egy két órát tartó ülés keretein belül. Ezek alkalmával valósulnak meg az agilis módszertanhoz tartozó események. A köztes időben mindenki egyénileg dolgozik a saját feladatán. Online konzultációra a slack (https://slack.com/) eszközt használjuk.

Az események kategóriába négy nagyobb egység tartozik: Előkészítés, Heti Rutin, Visszatekintés, Bemutató.

3.2.1. Előkészítés

A félév első időszakában előkészítés folyik. Ilyenkor a projekt vezető, technikai vezető és alcsoport vezetők megbeszélik, hogy milyen fő irányok érvényesüljenek a félév során. Az új tagok ismerkednek a rendszerrel, a régebbi fejlesztők pedig jelzik, hogy milyen konkrét feladatokat vállalnának szívesen. A Scrum mester kiselőadás/tréning keretében ismerteti azokat az értékeket, melyek a csoport működésének hagyományához tartoznak és azokat a módszereket, melyeket a csoport a félév során használhat az értékek aládúcolására (Példa értékekre: hatékony csapatmegbeszélések, motiváció fenntartása, tudás átadása; példa ezeket alátámasztó módszerre: napi Scrum rituáléja.) Az előkészítés szakasz rendszerint az első 2-3 találkozó idejét veszi igénybe.

3.2.2. Heti rutin

Az előkészítés szakasz után egy sor olyan találkozás következik, melynek a felépítési váza a következő: hírek, napi Scrum, hét témája, vita közösen, vita kiscsoportban.

A *hírek* részlegben elhangzik a csapattagok köszöntése; a Scrum mester jelzi a közösségnek, ha valaki hiányzást, késést jelentett; rövid reflektálás történhet a beküldött heti beszámolókkal kapcsolatban; a csoporttal kapcsolatos rövid hírek hangzanak el (Példa rövid hírre: "Felkértek bennünket, hogy vegyünk részt a kari Nyílt napon." "Két csoporttársunk elnyerte a Kar Kiváló Hallgatója címet. Gratulálunk nekik!".) Maximum 5 percet vesz igénybe ez a kis rituálé és segíti a hatékonyságot, fókuszálást indít el a tagokban.

A *napi Scrum* maximum 15 percet tartó rituálé. A technikai vezetők, alcsoport vezetők és a fejlesztők kötelező módon részt vesznek rajta. Minden tag három kérdésre válaszol röviden: „Mit valósítottam meg az elmúlt találkozó óta?"; „Mit tervezek megvalósítani a következő találkozóig?"; „Van-e valamilyen elakadásom?". A kérdések és az időkeret szem előtt tartása abban segít, hogy lendületesen, hatékonyan haladjunk végig a csoporttagokat érintő kérdéseken. Ha ugyanis belemegyünk olyan technikai részletekbe, melyek egy-egy tag számára fontosak, de a többi tag nem érintett bennük, akkor az utóbbiak könnyen nagyon unottá és demotiválttá válnak. Így tehát az ilyen jellegű tisztáznivalókra fény derül a napi Scrum folyamán, kulcs szavait a Scrum mester feljegyzi, de megbeszélésükre a találkozó második felében kerül sor, az érintettek, esetleg egyéb érdeklődők körében (vita közösen, vita kiscsoportban). A napi Scrum rövid tanulságok továbbadására viszont alkalmas terep, hiszen mindenki kiélezett figyelemmel vesz részt rajta. Másik hozadéka, hogy mivel a csapattagok hangosan fogalmaznak meg haladási ritmusukat illető gondolatokat, saját magukat és egymást is motiválják, lendületben tartják.

A *hét témája* rész a Scrum mesternek biztosít keretet arra, hogy a csoport értékeinek mélyítésén dolgozhasson a jelenlévőkkel. Általában 10-15 percet vesz igénybe és kiselőadás vagy tréning formájában valósul meg. A 2018-as első félév során olyan témák merültek fel, mint: egyéni motiváció; közös cél; közös felelősségvállalás; heti beszámoló hatékony megfogalmazása; munka becslése; bemutató tervezése; stb. Ezek átbeszélése hozzásegít ahhoz, hogy az új és régi tagok is elsajátítsanak olyan értékeket és módszereket melyek lehetővé tudják tenni, hogy egymás mellett dolgozó egyének helyett valódi csoportmunka alakuljon ki.

A *vita közösen* illetve *vita kiscsoportban* olyan kérdések körbejárására ad lehetőséget, melyek a napi Scrum során feljöttek, de a hatékony csoport-összhelyzet felmérés érdekében későbbre halasztottuk kibontásukat. Amennyiben mindenkit érintenek a csoportban (például egy komolyabb technikai döntés, mely a termék összképére van hatással) a leginkább bevonódott tagok felvezetik a megvitatandó kérdést, majd mindenki együtt érvel annak váza mentén. Ilyen kérdésekből egy találkozó alkalmával több is feljöhet, de akár egy sem. Ha csak egy pár személyt, esetleg csak egy fejlesztő-alcsoport vezető párost érint egy kérdés, akkor ők külön elvonulnak és megtárgyalják azt. Példa erre, amikor a fejlesztő saját feladatát illetően elakad, vagy éppen befejezi azt és új feladatot kér. A vita rész kiváló lehetőséget nyújt a csapat tagoknak arra, hogy megosszák egymással tapasztalataikat, tudásukat egy adott téma kapcsán és együtt, egymás által fejlődjenek.

3.2.3. Visszatekintés

A visszatekintés során a csapattagoknak lehetőségük van reflektálni a közös munkára és együtt olyan irányokat, módszereket megfogalmazni, melyek bevetése vélhetően pozitív hatással lesz a következő munkaszakaszokra. A 2018.-as első félév során egy visszatekintő alkalom volt, a félév közepén, a hetedik találkozó alkalmával. A visszatekintést a Scrum mester koordinálta, aki két különböző színű, kis méretű papírlap csomagot osztott szét a csapattagok között. Az egyik szín a pozitív, a másik szín a negatív visszajelzéshez társult. Minden papírlapra szigorúan egy kulcsszót, gondolatot írhattak fel a csoporttagok, anonim módon. A visszagyűjtött papírlapokat „negatív/pozitív" illetve „technika/módszertan/csapat" tengelyek mentén rendszerezte a Scrum mester a táblán, egymás tetejére helyezve azokat a papírlapokat, melyek ugyanazt a gondolatot képviselték. Az így megjelenő kupacok arra utaltak, hogy egy-egy visszajelzést nagyobb súllyal érdemes kezelni, hiszen több csapattagot is hangsúlyosan érint. A visszajelzések áttekintését követően megbeszélés zajlott, mely során a csoport körbejárta a problémás területeket és ötleteket fogalmazott meg javításukat illetően. Egy konkrét példa: többen jelezték, hogy redundánsnak érzékelik az adminisztrációs feladatokat (heti jelentések írása, GitHub használata, napi Scrum). A megbeszélés során sikerült tisztáznunk, hogy mindegyiknek van külön hozadéka is, illetve a redundáns információkat hivatkozások használatával fel tudjuk oldani a továbbiakban.

3.2.4. Bemutató

A bemutató a félév záró alkalma, mely során a kutatás-fejlesztés labor minden tagja összegzi a félév során megvalósított munkáját és bemutatja eredményeit. Az alkalmon külső meghívottak is részt vehetnek. A menetrend logikusan felépített: a projekt vezető ismerteti a fő irányokat, a Scrum mester a munka menetének módszerét, a fejlesztők pedig -élen az alcsoport vezetőkkel- a megvalósított fejlesztéseket.

2018. július 2.-án került sor az idei bemutatóra, mely egyben a txtUML 0.7.0 release-ének bemutatója is volt. Az alkalmon régi csapattagok is részt vettek és nem maradt el a tortás-pezsgős ünnep sem, mely a közös sikert, mint motivációt hivatott erősíteni.

3.3. Termékek

A csoport munkájának elsődleges terméke a fejlesztett *kódbázis*, mely nyílt forráskód formájában elérhető a https://github.com/ELTE-Soft/txtUML/ honlapon.

Ugyanezen a honlapon megtalálhatóak a szoftvertermékkel kapcsolatos feladatok, melyek *termékkívánságlistaként* is felfoghatók (issues fül alatt). Ezekhez adhatnak hozzá vagy ezekből válogathatnak a fejlesztők, alcsoport vezetők, technikai vezető – természetesen a projekt vezető által kijelölt iránynak megfelelően.

Az alcsoportok mindegyikéhez tartozik egy-egy *tábla* (ugyancsak az említett honlapon, projects fül alatt), mely megjeleníti az alcsoportban éppen aktuális feladatokat. Olyan oszlopai vannak, mint: várakozó feladatok („to do”); fejlesztés alatt („in development”); tesztelés alatt („under testing”); felülvizsgálatra készen („pull request”); kész(„done”). A tábla átfogó képet nyújt tehát az alcsoport munkájának jelenlegi állapotáról.

Ezeken kívül létezik egy belső honlap, ami tulajdonképpen egy űrlapot jelenít meg, ezen keresztül gyűjt egy háttér táblázatba információkat a tagok heti haladását illetően. A „*Heti beszámoló*” nevet viseli. Felületén egy elvégzett feladattal kapcsolatban a következő adatok adhatók meg: kivitelező neve; feladat típusa (pl. kutatás, fejlesztés, mentorálás, adminisztrálás, stb.); feladat rövid leírása; feladat kifejtett leírása (ha a rövid leírás nem elegendő); kapcsolódó GitHub feladat kódja (ha van ilyen); kiemelendő tanulság (ha van ilyen). A heti beszámoló kitöltésére minden héten, a találkozót megelőző időpontban kell a fejlesztők időt szánjanak és az előző találkozó óta végzett összes feladat kapcsán kitöltsék a megfelelő adatokat. Ennek kapcsán többször fogalmaztak meg a csapattagok ellenérzést, leginkább a túlzott adminisztráció vádja merült fel. Ennek ellenére megtartottuk a heti beszámolót a következő pozitív hatásaira tekintettel: mivel teljesítése megelőzi a napi Scrumot, elősegíti, hogy a csoporttagok összeszedetten, fókuszáltan számoljanak be a napi Scrum során; kutatáshoz és pályázat elszámoláshoz alkalmas adatsorokat állít elő; a később becsatlakozó csapattagok felzárkózásához nyújthat segítséget; önreflektáláshoz gyűjt adatokat – a személyes teljesítményt naplózza; olyan gyakorlat, mely fejleszti a csoportmunkához szükséges készségeket (együttműködés, szervezés, kommunikáció) és ráhangol a munkahelyi adminisztrációs követelményekre is.

4. Visszajelzések, eredmények

2018 tavaszi félévében 18 héten át alkalmaztuk a 3. fejezetben bemutatott módszertant. Megfigyeléseink a következők voltak: a heti találkozók hatékonyabbak, gördülékenyebbek és lendületesebbek lettek, mint korábban voltak; nőtt a kommunikációs interakciók száma a csapattagok között; az önálló munkavégzés mellett kezdett kialakulni a csoportos felelősség tudata; a rendszeres adminisztrálás kezdett elfogadottá, természetessé válni; tisztábbak lettek az egyéni és a saját célok.

A 7. heti visszatekintés során a csapattagok hasonló észrevételeket fogalmaztak meg. Pozitív jellemzőként emelték ki a csapatmunkát, mely ezúttal az alcsoportok között is jobban érvényesült. Hárman is utaltak arra, hogy a célok összeszedése, világos megfogalmazása pozitívan hatott rájuk; a haté-

konyabb kommunikációt is többen kiemelték. Rövidebb és jó hangulatú találkozókról számoltak be. Nagyon pozitívnak élték meg a csapattagokban lévő kompetenciát, mely a segítségnyújtással társulva sok tanulási lehetőséget nyújtott, főleg az új tagok számára. Ez utóbbi pozitív visszajelzés az alkalmazott módszertant annyiban illetheti, hogy az igyekezett teret biztosítani a tudás átadására a csoport struktúra (hierarchikus szerepkörök) és a heti rutin (főleg a napi Scrum és a közös/kiscsoportos viták) által.

Negatív visszajelzések a heti beszámoló kapcsán érkeztek elsősorban, melyet a csapattagok redundánsnak címkéztek. Panasz volt még a „folyamat túlformalizálása". Mindkettőt némileg kezdeti kritikának éltük meg, vagyis az újdonságokkal szembeni alapvető emberi ellenállásnak. Reakciónk rá az volt, hogy adjunk még időt ezen elemek kapcsán, hogy pozitív hatásuk nyilvánvalóbbá válhasson hosszabb távon, addig pedig vita keretében mutassunk rá értékeikre. Ha későbbi visszatekintések is rávilágítanak még demotiváló hatásukra, akkor közös döntés nyomán természetesen elhagyhatók, módosíthatók.

A 17. héten a csapattagok kérdőíveket töltöttek ki a félévre vonatkozóan. Az ezek által begyűjtött információk is elemezhetők az alkalmazott módszertanra vonatkoztatva. A kérdőíveket 7 személy töltötte ki, vagyis a csapat 77 %-a. A számadatot bekérő kérdések esetén 1-től 5-ös skála állt a kitöltők rendelkezésére, ahol az "1" a legkevésbé, az „5" a leginkább szavaknak feleltethető meg. A módszertanra legdirektebb módon utaló kérdés volt: „Mit gondolsz, milyen mértékben függ össze a félév során megvalósított érték a projekt menedzselésével?". A válaszok átlaga 3,71 volt. A „Mennyire jöttek át számodra az agilis módszertan értékei a közös munka során?" kérdésre ennél alacsonyabb átlagú válaszok érkeztek: 3,41. Ezeket mi nem tarjuk nagyon alacsony értékeknek, viszont ennél magasabbakat szeretnénk elérni. (Az agilis értékek átadására például ilyen céllal dolgoztunk ki egy külön tréninget, melyet a következő félévben már be is vetettünk.)

Voltak egyéb kérdések, amelyekre kapott átlagokat nagyon pozitívnak tekintünk és amelyek hátterében módszertani hatásokat is látni vélünk. Ezeket [kérdés-válaszok átlaga-magas átlag lehetséges módszertani alapja] hármasokban foglaljuk össze:

- „Mennyire tartottad (számodra) megfelelőnek a gyakorlat során általad megvalósított feladatot?" – 4,28 – A vita kiscsoportban és az alcsoport vezetők mentoráló szerepe segítette, hogy a tagok személyre szabottan válasszanak feladatot és elégedettséget éljenek meg ennek kapcsán.
- „Mennyire volt megfelelő számodra a munkádra kapott visszajelzések száma, minősége (a félév folyamán)?" – 4,57 – A napi Scrum, vita közösen, vita kiscsoportban és az alcsoport vezetők, technikai vezető mentoráló hozzáállása teret és keretet adott az egyénre vonatkozó visszajelzésekre.
- „Mennyire működött csapatként a csoport?" – 4,14 – A heti találkozók eleji közös hírfogadás (például közös gratuláció egy csapat tagnak), a napi Scrum mindenkit bevonó jellege, a közös értékeket tudatosító hét témája rész segítette, hogy kialakuljon a csapatszellem. A technikai vezető és az alcsoportvezetők mentoráló hozzáállása is erősíthette az összetartozás érzetét.
- „Milyen volt a közérzeted a csapatban a félév során?" – 4,71 – A heti rutin menete és a szerepkörök megfelelő kialakítása segítette, hogy mindenki elegendő figyelmet kapjon a csoportban és gördülékenyen tudjon a részeként működni.

Kiemelünk néhány szöveges formában megfogalmazott visszajelzést is. A legjobb tapasztalatok kapcsán írt kulcsszavak: „a csapatmunka, fejlődés, egymás ösztönzése"; „Közösen dolgoztunk egy érdekes feladaton." Hisszük, hogy a módszertan hatékonysága is hozzájárult ahhoz, hogy a csapattagok legjobb tapasztalatként éljék meg a közös munkát. A „legnagyobb kihívás"-okat illetően felfigyeltünk arra, hogy a fejlesztést és mentorálást együttesen végző csoporttagok nehézségekkel küzdenek az egyensúly megtartása kapcsán. Ehhez több segítséget szükséges nyújtanunk nekik a továbbiakban. A „min változtatnál" kategóriában a senior hallgatók eltávozásával (befejezik egyetemi tanulmányaikat)

kapcsolatos félelem is megjelent, mely utalás volt számunkra arra, hogy még több figyelmet kell szenteljünk a jövőben a projekt-tudás átadásának (seniortól fejlesztőnek irányba leginkább). Ugyancsak ebben a kategóriában fejeződött ki az igény a módszertan mélyebb megismerésére is mely további érv volt egy/több teljeskörű agilis módszertan tréning bevezetése mellett.

5. Összefoglalás

Az egyetemi környezetben működő kutatás-fejlesztés csoportok munkájának megszervezése érdekében inspirálódhatunk a szoftverfejlesztés iparban bevált agilis módszerekből. Több példát láthattunk arra, hogy ezek alkalmazása klasszikus szoftverfejlesztési gyakorlat órán sikereket hozott. Amikor viszont kutatás-fejlesztési laboron szeretnénk bevetni őket, szem előtt kell tartanunk néhány sajátos jellemzőt: a kutatás jellege nagyobb szabadságot kíván; kevésbé becsülhetővé teszi a feladatok méretét, a megvalósításukhoz szükséges időt; a kutatócsoportokba nagyobb rugalmassággal csatlakozhatnak be és válnak ki tagok; egy már meglévő kódbázissal és csoporttal szembesülnek az új tagok és egyéni ritmusukban kell tudjanak ehhez alkalmazkodni; nagyobb teret és konkrétabb lehetőségeket kell adni a tagoknak arra, hogy egymástól tanulhassanak, együtt fejlődhessenek a kutatás témáját illetően.

Ezeket a sajátosságokat szem előtt tartva dolgoztunk ki egy agilis módszertant a txtUML kutatócsoport számára. A projektvezető többnyire iránymutató szerepe kellő szabadságot és biztonságot ad a kutatáshoz, míg a Scrum mester a gördülékeny, ugyanakkor egyének számára rugalmasságot kínáló csapatmunkát segíti. A technikai vezető - alcsoport vezetők – fejlesztők hierarchikus lánc a tudás hatékony áramlását hivatott előidézni. Az előkészítés, hírek, hét témája, visszatekintés munkaszakaszok leginkább a csapat munka értékeinek tudatosítására biztosítanak terepet. A napi Scrum, a termék kívánságlista, a táblák és a heti beszámolók az átláthatóságot és ezáltal a hatékonyságot, motivációt kívánják erősíteni. A közös vita, vita kiscsoportban a tudás átadásának, az együttes fejlődésnek a bázisa.

A kipróbált módszertan kapcsán olyan visszajelzések érkeztek, miszerint hatékonyabb lett a kommunikáció a csoportban, tisztábbak a célok, kellemes a légkör, jobban érvényesül a csapatmunka, egyéni fejlődést élnek meg a tagok a tudás csapatban való áramlása által. A visszajelzések olyan továbbfejlesztési lehetőségekre is rámutattak, mint például az agilis értékek elmélyítését szolgáló tréning bevezetése. Összességében elégedettséget érzünk a módszertanunk kapcsán és továbbra is agilis vezetéssel kívánunk „hozzájárulni ahhoz, hogy emberek egy csoportja egyéni képességeit szenvedéllyel és kreativitással értékek teremtésére használja.” [1]

Köszönetnyilvánítás

Köszönettel tartozunk a txtUML kutatócsoport minden tagjának, akik nyitottan és kísérletező kedvvel fogadták minden ötletünket saját módszerünk kialakítása érdekében. Készséges hozzáállásuk nélkül ez a tanulmány nem jöhetett volna létre.

A kutatási projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg (EFOP-3.6.3-VEKOP-16-2017-00002).

Irodalom

1. *Kutatócsoport: txtUML* https://people.inf.elte.hu/ilyese/txtUML.html (utoljára megtekintve: 2018.11.04.)
2. K. Greßer, R. Freisler.: Agilis és sikeres vezetés. Z-Press, Miskolc (2018)
3. J. G. Kuhl: *Incorporation of Agile Development Methodology into a Capstone Software Engineering Project Course*. In: College of Engineering at Iowa Research Online (ed.): 2014 ASEE North Midwest Section Conference, Iowa (2014)

4. T. Smith, K.M.L. Cooper, C.S. Longstreet: *Software Engineering Senior Design Course: Experiences with Agile Game Development in a Capstone Project*. In: ACM New York (ed.): ICSE 2011, Waikiki, Honolulu (2011)
5. C. Anslow, F. Maurer: *An Experience Report at Teaching a Group Based Agile Software Development Project Course*. In: ACM (ed.): SIGCSE'15, Kansas (2015)
6. D. F. Rico, H. H. Sayani: *Use of Agile Methods in Software Engineering Education*. In: IEEE Computer Society Washington (ed.): 2009 Agile Conference, Hannover (2009)
7. M. Kropp, A. Meier: *Teaching Agile Software Development at University Level*. In: IEEE (ed.): CSEE&T '13, San Francisco (2013)
8. E.Ilyés: *Esettanulmány: Agilis szoftverfejlesztés egyetemi kurzuson*. In: Webdidaktika alapítvány (ed.): INFODIDACT'2017, Zamárdi (2017)
9. *ResearchGate* https://www.resarchgate.net/post/Is_anyone_using_Scrum_in_research_projects_at_universities (utoljára megtekintve: 2018.10.04.)
10. M. Ota: *Scrum in Research*. In: Y. Luo (ed.): CDVE 2010, Calvia (2010)
11. N. Santos, J. M. Fernandes, M. S. Carvalho, P. V. Silva, F. A. Fernandes, M. P. Rebelo, D. Barbosa, P. Maia M. Couto, R. J. Machado: *Using Scrum Together with UML Models: A Collaborative University-Industry R&D Software Project*. In: O. Gervasi et al. (eds.): ICCSA 2016, Zhuzhou (2016)
12. I. R. Lima, T. de Castro Freire, H. A. X. Costa: *Adapting and Using Scrum in a Software Research and Development Laboratory* Revista de Sistemas de Informação da FSMA, 9. (2012) Trilha Principal (2012) 15-23

Architektúrális gondolkodás fejlesztése valós idejű rendszerekkel

Korom Szilárd

korom.szilard@gmail.com
ELTE IK

Absztrakt. A középiskolai programozás oktatásban az algoritmikusan nehéz feladatok megoldása jóval hangsúlyosabb, mint az architektúrális gondolkodás fejlesztése. Az előadásom célja ezen probléma bemutatása, részletezése, valamint egy olyan tananyagba építhető modul bemutatása, mely a valós idejű rendszereken alapszik, s a fent említett kompetencia fejlesztésére szolgál.

Kulcsszavak: programozás, informatikai kompetenciák, rendszerszintű gondolkozás, architektúra, architektúrális gondolkozás, valósidejűség, real-time, beágyazott rendszer

1. Informatikai kompetenciák

A programozás középiskolai oktatásának fontosságával már számtalan cikk és előadás foglalkozott [1]. Ezt tekinthetjük tehát axiómának még akkor is, ha a Nemzeti alaptantervben ez csak csekély mértékben van jelen. [2] Ha viszont ezt fontosnak tartjuk, meg kell vizsgálnunk mit értünk programozás oktatásán, pontosan mit- és miért akarunk fejleszteni. A Nemzeti alaptanterv alapján az informatika oktatást érintő kulcskompetenciákat megfogalmazta dr. Szlávi Péter és dr. Zsakó László egy korábbi cikkben. A következőket sorolták fel:

1. Algoritmikus gondolkodás
2. Adatmodellezés
3. A valós világ modellezése
4. Problémamegoldás
5. Kommunikációs képesség
6. Alkalmazói képesség
7. Csoportmunka, együttműködő-képesség
8. Alkotó képesség
9. Információs tájékozódási és tájékoztatási képesség
10. Rendszerszintű gondolkodás

Ezek közül nyilván az *Alkalmazói képesség* kerül elő leggyakrabban a jelen kor oktatásában, s a nemzeti alaptanterv is erre helyezi a hangsúlyt. [2] Ez a cikk azonban csak azokkal a kompetenciákkal kíván foglalkozni, melyek szorosan kapcsolódnak a programozáshoz. Így például bár a programozás a valós életben igen gyakran összefonódik a *csoportmunkával* (pl.: agilis szoftverfejlesztés), mégsem tekinthetjük a programozás részének. A cikk nem foglalkozik a *problémamegoldással* sem, hiszen ez egy olyan alapvető kompetencia, mely bármelyik másikkal összevonható, lényegében egy ős képességnek nevezhetjük, melyből tetszőlegest leszármaztathatjuk (akár más tantárgyakon belül is). Ezek alapján a következő kompetenciák maradtak:

EFOP-3.6.3-VEKOP-16-2017-00001: Tehetséggondozás és kutatói utánpótlás fejlesztése autonóm járműirányítási technológiák területén – A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

1. Algoritmikus gondolkodás
2. Adatmodellezés
3. A valós világ modellezése
4. Rendszerszintű gondolkodás

A kiválasztás fő szempontja tehát azon a kérdésen alapszik, hogy **milyen jellegű feladatokkal** érdemes tanítani a programozást. Ebből a nézőpontból a *problémamegoldás*, *kommunikációs képesség*, *csoportmunka*, *alkotó képesség*, *információs tájékozódás* kompetenciák egy feladat jelleg részét képezik, egy módszertani szempontot egy konkrét projekt tanítása során.

Jelen cikkben az *adatmodellezés*, valamint a *valós világ modellezése* kompetenciákkal nem kívánok foglalkozni. Ezen téma kidolgozása egy teljes dolgozatot igényelne. Másfelől, már született magyar nyelvű cikk a témakörben, melyeket a források között meg is említek. [3]

2. Rendszerszintű gondolkodás

A *rendszerszintű gondolkodás* meghatározására számtalan tanulmány készült. [4, 5, 6, 7, 8, 9] Ezeket kívánta összefogni, s egységesíteni Arnold és Wade A *rendszerszintű gondolkodás* olyan szinergikus analitikus készségek csoportja, melyek javítják egy rendszer azonosításának és megértésének képességét, megjósolják viselkedésüket és módosítják azokat, a kívánt hatások elérése érdekében. Ezek a készségek rendszerként működnek együtt. [10]

Röviden tehát a fenti szerzők a *rendszerszintű gondolkodást* úgy azonosították, mint egy rendszert, a rendszerekről való gondolkodásról. A probléma tehát az, hogy a komponensek működése, a részproblémák megoldása nem okozza a rendszer működését. Az *algoritmikus gondolkodás* egy komplex rendszer esetében nem mindig vezet megoldásra (vagy nem kellően gyorsan), hiszen ilyenkor a jelenségek kimenetelét sokszor a rendszer struktúrája, nem pedig az egyedi összetevők állapotai határozzák meg. [11] Például, a szükséges algoritmusok megírása önmagában nem elegendő ahhoz, hogy egy program működjön, sőt algoritmikusan nem is biztos, hogy azonosítható a struktúra. A feladatra, mint rendszerre kell tekinteni, akként kell azonosítani és elhelyezni, szükség esetén pedig a részegységeken kell módosításokat végrehajtani. Arnold és Wade a számos definíció és tanulmány alapján a következő készségelemeket fogalmazták meg (Veres Gábor fordításában): [12/227. o.]

2.1. A részek és kapcsolatok azonosítása

- *A rendszerszemlélet alapvető készsége, de megfelelő gyakorlat nélkül nehezen fejleszthető, illetve könnyen elveszthető.*

Egy komplex rendszer részeinek azonosításához rendelkezni kell sémákkal, amikre rá lehet illeszteni a komponenseket. Ilyen módon egy rendszer megtervezése, módosítása lehetetlen, ha nem vagyunk tisztában azzal, mik az előfordulható lehetőségek. Ehhez számtalan potenciális tudáselemre lehet szükség a definíciókon, tételeken, lexikális tudáson át egészen a gyakorlati alkalmazásig.

Mivel a *rendszerszintű gondolkodás* legalapvetőbb eleme is megkíván előismereteket, ezért kijelenthető, hogy a kompetencia fejlesztését meg kell előznie valamilyen ismeretátadás (pl: algoritmusok). Ez azonban nem jelenti azt, hogy „csak akkor lehet bevinni a tanórára, ha mindennel végeztünk". Módszertanként is elhelyezhető például úgy, hogy egy-egy új ismeret megszerzése után időt fordítunk annak rendszerben való működésére. Másik példa lehet a probléma-alapú oktatás projektjei. [13]

2.2. Visszacsatolások azonosítása és megértése

- *Bizonyos kölcsönhatások ok-okozati visszacsatolásokat alakíthatnak ki, amelyek alapvetően befolyásolják az adott rendszer viselkedését.*

Ez az elem tehát nem magukról a kapcsolatokról szól, hanem azok egymáshoz való viszonyairól, a kapcsolatok hatásairól és működési mechanizmusairól. Ennél a szintnél beszélhetünk arról a tézisről (amit korábban említettünk), hogy a komponensek működése nem (feltétlenül) okozza a rendszer megfelelő viselkedését. A rendszer struktúráját tekintve ettől lesz egységes egész, s a megfelelő működéshez rendszerint a részelemek módosítása is szükséges.

Az egyszerűbb programozói feladatok esetében a megoldás általában automatikusan, a tényleges feladat realizálása nélkül megtörténik, hiszen természetesen adódik az elemek megfelelő összekapcsolása, az ok-okozati visszacsatolások elkészítése. Módszertanilag erre könnyen lehet építeni, hiszen ha a diák valóban képes erre, akkor pár feladat után levonható egy egységes tanulság.

2.3. A rendszer szerkezetének megértése

- *Látszólag az 1. és 2. ponttal megegyező készségeket igényel, azonban a kutatások szerint inkább azok speciális kombinációját jelenti.*

Úgy is tekinthetünk erre, mint az első 2 pont egyfajta ötvözetére. Bonyolultsága, összetettsége a konkrét rendszertől függ, ezért tanításmódszertani szempontból ennek tanítása spirálisan képzelhető el egyre komplexebb rendszerekkel.

Ide tartozhatnak a kész architektúrák is, melyek sémákat biztosítanak az egyén számára a rendszer működésével kapcsolatban. A komplex rendszerek gyakorlati előállítása, kezelése szorosan összefügg ezzel, hiszen biztosra kell mennünk, hogy a szerkezeten mindenki ugyan azt érti. A sémák kialakítása kiválóan működhet a gyakorlatban *csoportmunkában*, hiszen ekkor a tagok rá vannak kényszerítve az egységességre.

2.4. Állandók és változók, folyamatok azonosítása

- *Állandó lehet például valamely fizikai erőforrás készlete (pl. tartalék tápanyag), vagy akár érzelmi, mint a bizalmi tőke egy kapcsolatban. A változók módosíthatják a készletek értékeit, így folyamatokat hívhatnak életre. Ez a rendszerszemlélet magasabb fokú készségeleme.*

A szerző a listát egy hierarchiában képzelte el, ahol ezt az elemet mindenképpen meg kell előznie az előző pontoknak. Az informatikában azonban ez véleményem szerint nem ilyen szigorú. A folyamatok (vagy angol nyelven: *flow*) végig követése sokszor igen hasznos az egész rendszer megértésében. Különös tekintettel összetett alkalmazások esetében. Az azonban bizonyos, hogy ezt el kell különítenünk az előző 3-tól, hiszen azok a szerkezetre vonatkoznak, míg ez egyfajta dinamikus, konkrét folyamat esetében válik érdekessé.

Ennél a pontnál külön szeretném hangsúlyozni a programozásban betöltött szerepet, ugyanis egy alkalmazás esetében nagyon meghatározó a *data flow*. Gondoljuk például az egyszerű konzolos alkalmazásokra (amikkel általában bevezetik a programozást). Ezek rendre az input, feldolgozás, output szekvenciát követik. Azonban nem csak ilyen alkalmazásoknál, de komplex informatikai rendszereknél, hálózatoknál is nagyon meghatározó az adat, annak végigfolyása, változók értékeinek megváltozása, hiszen gyakran az utóbbi esemény a „belépési pont", azaz erre reagál számos másik folyamat.

2.5. Nem lineáris folyamatok azonosítása

- *Ez a készségelem a fontossága és a félreértelmezhetőség elkerülése miatt került külön pontba az előzőtől.*

A párhuzamosság számtalan problémát felvet, hiszen ha egy folyamat „működik" egy lineáris rendszerben, korán sem biztos, hogy egy másikba is fog. A párhuzamos folyamatok egymás viselkedésére is hatással lehetnek, így a rendszer alapvető működési mechanizmusa változik meg.

Az informatikában különösen nem szabad megfeledkezni erről a pontról. Gondoljuk például a *Scratch*-re, mely pont a párhuzamos eseménykezelése miatt válik olyan érdekessé, dinamikussá és sokrétűvé.

2.6. A dinamikus viselkedés megértése

- *A kölcsönhatások és visszacsatolások befolyásolják a készleteket és a változókat, az időbeli folyamatok a rendszer dinamikus viselkedését alakítják ki. Ez a készségelem az előzőeket is feltételezi, és megfelelő gyakorlattal fejleszthető.*

Ehhez a szinthez már kiemelten szükség van a gyakorlat, sőt a rendszerek működésével kapcsolatos tapasztalatokra is. Ezen a szinten az egyén már képes a struktúra átlátására, részegységeinek azonosítására, egy lehetséges folyamat végig követésére, az általa beindított egyéb viselkedésmegváltozások azonosítására. Ezen elem megértése után a rendszer struktúrájáról egy absztraktabb képet kaphatunk.

Véleményem és tapasztalataim szerint a programozás oktatásában ez a legnehezebb szint. Meglévő struktúra és a megfelelő elemek ismeretével sem biztos, hogy a diák meg tud oldani egy feladatot, vagy képes a dinamikus viselkedés megmagyarázására még akkor is, ha ismeri a forráskódot. Még ha tervezni tud is, sőt érzi, hogy milyen algoritmuselemet kellene használni, nem tudja ráilleszteni a konkrét folyamatra. Ennek elsajátításához nagyon sok gyakorlat és tapasztalat szükséges.

2.7. A komplexitás csökkentése a rendszermodell megfelelő tervezésével

- *A komplexitás különféle intuitív technikákkal (pl. redukció, transzformáció, absztrakció és homogenizáció) csökkenthető a modellekben. Lényegében az adott cél szerint felesleges rendszerelemek kizárásának képességét jelenti.*

Ez a pont a gyakorlatban már nem mindig jelenik meg (főleg a középiskolai oktatásban), de a *rendszerszintű gondolkodás* feltétlen része. A kérdés, hogy a komplexitás csökkentése, mint meglévő rendszer módosítása, vagy mint a tervezésnél fontos szempont jelenik meg. Az utóbbira feltétlenül szükség van mind a diák, mint a tanár szempontjából, hiszen a lehető legegyszerűbben szeretnénk egy problémát megoldani.

2.8. Egymásba épülő rendszerszintek megértése

- *Ez a készségelem a rendszerek egymásba épülésével kialakuló hierarchia, a rendszer-alrendszer összefüggés megértése, az anyagi világ szerveződési szintjeinek átlátását foglalja magában;"*

Ez a képesség már igen bonyolult. Nagyon sok sémára, gyakorlatra és elméleti tudásra van szükség ahhoz, hogy a rendszert, mint egységes egészet tudjuk látni, hiszen Mér László már megfogalmazta, hogy a rövid távú memóriában egy időben tárolható kognitív sémák maximális száma. 7 ± 2 [10/12. o.] Ami tehát fontos az a lényeglátás, tehát a feladat szempontjából érdektelen komponensek kiszűrése.

További érdekes kérdés, hogy hogyan tudjuk megkülönböztetni az egyszerű komponens egy alrendszertől. Ha ugyanis a rendszer alrendszerekből épül fel, akkor a felsorolt hierarchia rekurzívan alkalmazható mindegyikre.

Az informatika középiskolai tanításában meglátásom szerint nem szükséges ilyen komplex rendszerekkel találkoznia a diáknak. A NAT-ban [2] sincsen rá utalás, hogy erre szükség lenne, s az érettségi feladatokat, vagy a versenyeket tekintve sem tűnik relevánsnak.

3. Rendszerszintű gondolkodás a NAT-ban

A rendszerszintű gondolkodás, mint elvárt kompetenciát az informatikai műveltségi területnél a Nemzeti alaptanterv [2] explicit nem fogalmazza meg. Utalás azonban több helyen is van rá. Például az adatmodellezés, algoritmizálás, információ tárolás, hálózati ismeretek s ezek komplex alkalmazásai elvárt készségek, amik együttes használata, kapcsolatuk megértése már igényli a rendszerszintű gondolkodást.

4. Rendszerszintű gondolkodás programfejlesztési szempontból

Az algoritmikus gondolkodás fejlesztésének fontosságáról és szerepéről számtalan tanulmány született. Hogy pontosan miről van szó, a következő idézettel magyarázható:

„Az algoritmizálás először nem számítógépes megvalósításról szól. Csak egy klasszikus, több ezer éves algoritmusra, a két egész szám legnagyobb közös osztóját meghatározó Euklideszi algoritmusra kell gondolnunk! Az algoritmus végrehajtója – a processzor – sok esetben lehet maga az algoritmust megalkotó, azt értelmező ember. (Sőt egy új probléma megoldásánál a rutinos programozó is magát „képzeli" a számítógép helyébe, s így próbálja ki a megoldás működőképességét.) Algoritmusokat mindenki hajt végre nap, mint nap, sőt az emberek többsége alkot is algoritmusokat saját maga és mások számára." [1/4. o.]

Összességében tehát ez a kompetencia sem kapcsolódik feltétlenül a programfejlesztéshez sőt, még az informatikához sem.[14]Persze ez nem annyira meglepő, ami azonban ebből feltétlenül következik, hogy az algoritmikus gondolkodás egyáltalán nem fedi le a valós programozói problémák megoldásához szükséges képességeket. A felvetés tehát az, hogy egy diák hiába tud algoritmikusan viszonylag nehéz problémákat megoldani (alapvető algoritmusok, rendezési algoritmusok, algoritmikus stratégiák), egy valós életben (például az iparban) használt program megírása igen nagy gondot okozhat neki még akkor is, ha eltekintünk a technológiai ismeretek hiányától. Tanárként már számtalanszor találkoztam azzal az anomáliával, hogy tehetséges, programozni tudó, algoritmikus gondolkodásukat tekintve fejlett diákok valójában nincsenek tisztában saját képességeikkel. Nincsenek tisztában azzal sem, hogy amit képesek megalkotni hogyan kapcsolódik a valós élet problémáihoz. Például hiába ismernek több rendezési algoritmust is, nincsenek tisztában azzal, hogy ebből hogyan lesz egy épkézláb alkalmazás. Valójában ez nem a diák hibája, hiszen mint tanár és programozó tudom, hogy az algoritmikus gondolkodása elég fejlett ahhoz, hogy egy átlagos programozói állásban előkerülő problémákat megoldjon (legalábbis nagy részüket), mégis távol áll még ennek megvalósításától. Természetesen az, hogy még nem alkalmas programozónak, számtalan aspektusa van a lexikális tudástól a pszichológiai érettségen át a kompetenciákig. Annak érdekében, hogy csak az informatikához kapcsolódó képességekre szűkítsük a kört, nem érdemes az iparban szükséges kompetenciákat vizsgálni, elegendő kellően összetett programozói feladatokkal foglalkozni. A cél tehát az, hogy kellő rálátást adjuk a diákoknak, mi is az a programozás, valamint, hogy tapasztalatokat szerezzenek arról, hogyan készül el egy összetett szoftveres rendszer. Ennek érdekében a *rendszerszintű gondolkodás* kompetenciát kell vizsgálnunk.

A programozás és a *rendszerszintű gondolkodás* egyvelegét **architektúrális gondolkodásnak** nevezem. Ennek az az oka, hogy az informatikában általánosan és a szoftverfejlesztésben is egy rendszert gyakran architektúrának neveznek, sőt az ezzel foglalkozó szakember beosztása is „architect". Fontos azonban visszautalni arra, hogy ennek fejlesztésével is egy informatikától független kompetenciát fejlesztünk.

Programozási szempontból tehát az az állítás, hogy az *algoritmikus-* és *architektúrális gondolkodás* együttes fejlesztésével a diák teljesebb képet kap szoftverfejlesztésről, valamint tapasztalatot szerez annak gyakorlati hasznáról, hiszen komplex szoftverrendszerek megalkotására is képes lesz. Ez persze motivációként is szolgál, hiszen az egyszerűbb, kevés funkcionalitással bíró alkalmazások és játékok gyártása egy idő után unalmassá, céltalanná válik még akkor is, ha olyan kiváló környezeteket használunk, mint a Logo, vagy a Scratch. Az *architektúrális gondolkodás* fejlesztése tehát meg kell előz-

[14] Természetesen létezik megközelítés, mely az algoritmikus gondolkodás fejlesztésénél a programfejlesztést helyezi középpontba. Erre az alábbi tanulmány szolgál példaként:
Fluent With Information Technology by the National Research Council. National Academy Press. June 1999. http://www.nap.edu/html/beingfluent/

nie az *algoritmikus gondolkodás* fejlesztésének. A diákoknak gyakorlati tapasztalatokat kell szerezniük apróbb alkalmazások, algoritmusok elkészítése terén[15] (ahogyan azt a részképességelemek első pontjában is láttuk).

5. Architektúrális gondolkodás fejlesztése jelenleg

A NAT-ban az informatikai műveltségi területnél az alábbi témakörök kerültek felsorolásra:

Az informatikai eszközök használata

Alkalmazói ismeretek

Problémamegoldás informatikai eszközökkel és módszerekkel

Infokommunikáció

Az információs társadalom

Könyvtári informatika [2]

Ez a gyakorlatban a tanmeneteknél általában úgy jelenik meg, hogy az irodai alkalmazások használata igen hangsúlyos, [15/6. o.][16] szerint 50%, a többi pedig meglehetősen fel van darabolva (többek között a kerettanterv [2] alapján). Ennek részletes elemzése most nem a cikk tárgya, ami azonban fontos, hogy a megszerzett tudás és készségek integrálására egy komplex rendszerben általában nem jut külön idő. Ennek hiányában pedig az *architektúrális gondolkodás* fejlesztése sem jelenik meg. A kompetencia fejlesztése tehát csak akkor kerül elő az informatika tantárgy keretében, ha a szaktanár az adott témakört egy olyan módszer szerint építi fel, melyben az összegzés, összekapcsolás, rendszerbe helyezés kérdése előfordul. Erre természetesen az érettségi feladatokon keresztüli tanítás teljes mértékben alkalmatlan, hiszen ott az irodai alkalmazásokkal foglalkozó feladatok külön egységet képeznek. Például, ha az adatbáziskezelést kizárólag az MS Access segítségével oktatjuk, a diák nem találkozik azzal, hogy az adatbázis rendszerek mikor- és miért hasznosak. Feladatokkal találkozik, melyet egy programmal kell megoldani, azaz „l'art pour l'art" módon tanulja meg a témakört.

Jó motivációt adhatnak a diákoknak a programozói versenyek. A Dusza Árpád Országos Programozói Emlékverseny [17] például egy kiváló lehetőség, hogy ne csak az algoritmikus gondolkodás kompetenciáját fejlesszük a diákoknak. A verseny lényege röviden annyi, hogy 3 fős csapatokban kell komplex alkalmazásokat[17] fejleszteni körülbelül 4 óra alatt (korosztálytól és fordulótól függően). A feladatok általánosságban úgy néznek ki, hogy MVC (Model-View_Controller) architektúrában felírhatók, azaz van egy vagy több adatforrás (jellemzően szöveges fájlokban), egy üzleti logikai, valamint valamilyen megjelenés a felhasználó számára. Érdekes, hogy bár a feldolgozandó egység algoritmikusan nehéznek mondható, a verseny kihívását mégsem ez a rész jelenti, hanem sokkal inkább az időhiány és a csapattagok közötti feladatmegosztás. Tapasztalataim szerint, mind a kettőt megkönnyíti, ha a diákoknak van valamilyen szoftverarchitektúra sémája. Ez azt jelenti, hogy ha meglátnak egy feladatot, a programra mint rendszerre tudnak nézni, s annak analizálásával a konkrét implementálási feladatok világosabbak és a megoldás gördülékenyebben születik meg.

[15] [17] tanulmány kiváló példákat hoz *Scratch* játékokra

[16] A cikk igen érdekes elemzést mutat az új kerettantervről úgy, hogy összehasonlítást ad a brit megfelelőjével.

[17] Ami érdekes továbbá, hogy nem csak asztali alkalmazásfejlesztő kategória van, hanem web programozás, mobil programozás kategóriák is léteznek.

6. Architektúrális gondolkodás fejlesztése valós idejű rendszerekkel

A továbbiakban azzal kívánok foglalkozni, hogyan érdemes az *architektúrális gondolkozást* fejleszteni a középiskolai informatika oktatásban. Ezzel kapcsolatban először egy vázlatos összefoglalót kívánok adni a témával kapcsolatban, amelyeket korábban a cikk keretein belül előkerültek:

1. A NAT [2] nem szán külön modult a képesség fejlesztésére az informatikai műveltségi területen belül, így a tanmenetben külön témakört általában nem képezhet.
2. Az előző probléma megoldása az lehet, ha a *rendszerszintű gondolkodást* a tanításmódszertan módosításával fejlesztjük.
3. Különböző hazai versenyek18 jó alapanyagot (feladatokat), motivációt szolgáltathatnak a kompetencia fejlesztésére.
4. Az *algoritmikus gondolkozás* és az *architektúrális gondolkozás* fejlesztésével a diák alkalmassá válik összetett programok megírására, ami sikerélményt, motivációt adhat számára, valamint tapasztalatot szerezhet a valós programozói tevékenységéről19.

A kompetencia fejlesztéséhez elsőrendűen olyan problémákra, projektekre van szükség, melyek sikeres megoldása a készség fejlődését vonják maguk után. Ehhez komplex, alrendszereket, alprojekteket, alkomponenseket tartalmazó feladatokra van szükség. A cikk a *programozásra* koncentrál, de nem tartom kizártnak, hogy például *alkalmazói rendszerekkel* is lehet ezt fejleszteni.[20] Azt persze érezzük, hogy az alapvető programozási tételeket, érettségi feladatokat, alapvető programozási stratégiákat gyakoroltató feladatok erre nem igen alkalmasak. Fontos szempont továbbá, hogy bár összetett projekteket keresünk, azok megoldása ne igényeljen túlságosan sok új lexikai tudást, megismerésükre ne menjen el sok időt.

A cikk példaként a valós idejű, beágyazott rendszereket használja[21]. A *valósidejűség* jelentőségére a következő bekezdésben fogok részletesebben kitérni. A lényeg azonban az, hogy a közoktatásban is egyre gyakrabban alkalmaznak *beágyazott-*, *IoT rendszereket* (Raspberry Pi, Arduino, Micro:bit stb.) programozás oktatásra. Ennek nagyszerűsége és hasznossága számtalan cikkben és tananyagban előkerült már [19, 20, 21], én csak egy rövid összefoglalót kívánok adni:

1. Gyors, látványos, gyakorlati tapasztalattal járó élményt kínál a diákoknak
2. Egy tananyag modulárisan, spirálisan felépíthető az elérhető környezetek miatt
3. Számtalan programozási környezet elérhető a blokk alapú nyelvektől a kódolásig, pl.: Blockly, Microsoft MakeCode, Scratch, Logo, Python, C# stb.
4. Az általános iskola alsó tagozatától kezdve lehet oktatni, egészen a középiskola végéig és mindig új, izgalmas feladatokat lehet adni
5. A diákok többféle hardverrel, operációs rendszerrel, programozási környezettel ismerkednek meg tanulmányai végére, ami által teljesebb képet kapnak az informatikáról, az *informatikai rendszerekről*

---

[18] Zsakó László egyik előadásában [18] elkészített egy összefoglalót a hazai programozást érintő versenyekről

[19] Az [1] cikk részletesen megvizsgálta a kérdést, hogy a diákok menyire nincsenek tisztában azzal, mit csinál egy informatikus

[20] Ennek lehetőségére talán egy másik cikk majd válaszol.

[21] Bár nem tartom kizártnak, hogy más is legalább ennyire alkalmas lehet. Például a https://www.netsblox.org/ kollaborációra alkalmas környezetet biztosít még mindig blokkos elemekkel.

A cikk szempontjából talán az utolsó pont a legérdekesebb. Ez ugyanis azt jelenti, hogy ezeket az eszközöket használva egy olyan környezetet kapunk, mely a szokványos asztali-, vagy konzolos alkalmazások felépítésétől eltérő. Másrészt, mivel hardverekről is szó van, fizikailag létező komponensekről beszélhetünk, melyeket már csupán össze kell kötnünk valahogy, **rendszert kell csinálnunk belőlük**. Ez azért hasznos, mert ha pusztán szoftveresen szeretnénk összetett rendszert alkotni, ahhoz nagyon bonyolult feladatra lenne szükség (pl.: Dusza verseny feladatai). A valós idejű-, beágyazott rendszerek alkalmazásával kevés lexikai tudással, rövid forráskóddal, jelentős eredményeket érhetünk el. Mivel a fizikai komponensek + szoftveres környezet alkotja majd a rendszerünket a *csapatmunka* is hangsúlyossá válik, hiszen a megvalósítandó részelemek nem csak logikailag, de a valóságban is teljesen elkülönülnek.

7. Valós idejű rendszerek az oktatásban

A fentebb felsorolt rendszerek attól válnak *valósidejűvé*, hogy megszorításaink vannak arra, hogy egy-egy folyamatot mennyi idő alatt hajtsanak végre. Ezeket folyamatokat ezután rendszerint a valameddig megismételjük. Például egy szenzoradat begyűjtését valós időben szeretnénk továbbítani, vizualizálni, de egy motor vezérlése is valós időben kell, hogy megtörténjen (pl.: drón). Ha a *valósidejűség* szempontjából közelítjük meg a hardvereinket az osztályteremben, akkor lényegében okos-otthon projekteket kapunk. *„Az IoT (Internet of Things) elterjedésével, a smart otthonok, smart city program kiteljesedésével még inkább előtérbe kerül ez a terület, ahol az adatgyűjtés és azok azonnali feldolgozása szükségszerű."* [22] Amennyiben ezeket az interneten keresztül vezéreljük, például egy központi alkalmazásból, már is kaptunk egy IoT rendszert. Összességében tehát a projektünk a következő elemekből állnak:

1. A csoport létszámától függő a beágyazott rendszerek száma
2. A hardverek néhány kifejezett funkciót látnak el valós időben (pl.: szenzoradat begyűjtése, motor vezérlése stb.)
3. A hardverelemek az információt az interneten keresztül valós időben kapják vagy adják
4. Létezik egy központi alkalmazás, mely képes a modulok vezérlésére, adatok vizualizációjára

7.1 Az ötlet megvalósíthatósága

A programozás oktatása beágyazott rendszereken egyre elterjedtebb. Egyre több iskola engedheti meg magának, hogy Raspberry Pi vagy Arduino segítségével oktasson, ezzel ablakot nyitva az IoT világára. Az ötlet előfeltétele ezen eszközök elérhetősége. Ha van ilyen, akkor ez tulajdonképpen egy módszer, egy projekt annak alkalmazására.

Azért jól alkalmazható ez a rendszer, mert a programozási környezetben nagy mozgásterünk van. Például a Raspberry Pi-re készíthetünk Python-ban és C#-ban[22] is szoftvereket. Mivel egy szenzoradat, vagy egy motor vezérlése algoritmikusan nem nehéz feladat, ezért a plusz információ egy asztali alkalmazás elkészítéséhez képest (ha feltételezzük, hogy a diák a fenti nyelvek valamelyikében már tud programozni) csak pár utasítás, mely egy rövid dokumentációban összegyűjthető.

A valós idejű kommunikáció érdekében szükségünk lesz még egy *valósidejű* adatbázisra is. Ehhez a *Firebase* [24] rendszert ajánlom. Ez ugyanis egyéni felhasználásra, limitált kommunikációra ingyenes, használata pedig nagyon egyszerű. Az „adatbázis" szó talán ijesztően hangozhat, pedig egyáltalán nem az. Valójában nem is kommunikálunk közvetlenül adatbázissal, hiszen a szerviz API-án keresztül tudjuk elérni a platformot. A gyakorlatban ez pár új parancsot jelent, ahol szöveges adatokat lehet küldeni. Fontos megjegyezni, hogy a *Firebase* egy nem-relációs adatbázis, hanem egy JSON

[22] További nyelvek: C#, C++, Javascript, Visual Basic, Node.js[23]

fa. Amikor hozzáadunk egy új adatot, az egy csomópontként jelenik meg a JSON-ben egy kulccsal összekapcsolva. Ezek ismerete egyébként egyáltalán nem szükséges a diákok számára.

A következőkben 2 mintakódot mutatok be, demonstrálva a rendszer használatának egyszerűségét. A cikk formája nem alkalmas egy konkrét projekt részletes bemutatására, de számtalan példa elérhető a *Firebase* használatához. [25] Például az adatbázis Python kódból való inicializálása a következő sorokból adódik:

```
from firebase import firebase

firebase = firebase.
FirebaseApplication('https://adatbazisom.firebaseio.com/', None)
```

Egy adat pld. hőmérséklet lekérése a következő eljárással lehetséges. Ha az adatok folyamatosan frissülnek is, nekünk csak az elsőre van szükségünk. Az adatok a „message" csomópont alatt vannak.

```
def updateMessage():
    tempMessage = firebase.get('/message', None)
    if tempMessage is not None:
        tempMessage = tempMessage.values()[0].values()[0]
        print "Beerkezett uzenet: ", message
```

A *valósidejűség* azért jelentős, mert egy-egy adat frissülésének eseményére fel tudunk iratkozni egy eljárással, amely, az adatbázis adatainak megváltozásakor képes végrehajtani valamilyen műveletet.

7.2 A projekt és az architektúrális gondolkodás kapcsolata

Összességében tehát miért kapcsolódik ez a cikkben tárgyalt kompetenciához, miért alkalmas annak fejlesztésére? A fentebb felsorolt elemek önmagában egy komplex rendszert alkotnak. Persze az implementálás legalján az szerepel, hogy hogyan lehet például szenzoradatot beolvasni, de nyilván nem ez a feladat oroszlánrésze. Ha eltekintünk attól, hogyan lehet a konkrét technológiákat alkalmazni (ami egyáltalán nem nehéz, ráadásul az adatbázisok használata például elő kell, hogy kerüljön), a feladat a rendszer megtervezése (Ki/Mikor/Mit/Kivel kommunikál? Mik történnek, ha egy esemény bekövetkezik?).

A központi alkalmazás összességében adatok rendszerez, vezérel és ütemezési feladatokat lát el. Az architektúrája azonban egyértelműen megegyezik a leggyakrabban használt szoftveres architektúrával, az MVC-vel. Az érdekessége az, hogy ha lecsupaszítanánk a többi komponenstől, akkor egy igen egyszerű alkalmazást kapnánk. Képzeljük el például, hogy az adatokat egyszerű szöveges fájlban kapjuk, s a feladat ennek a fájlnak a menedzselése. Ha nem szeretnénk túl bonyolult vizuális megjelenítést, ez gyakorlatilag egy informatika érettségi programozói feladatával egyenértékű probléma.

8. Összegzés

A *rendszerszintű gondolkodás* kompetencia a jelenlegi középiskolai oktatásban nem igazán hangsúlyos. Ez azért probléma, mert a NAT-ban is előkerül közvetve ez a képesség, megléte pedig igen fontos a későbbi életben. A programozók esetében ez különösen igaz, hiszen bonyolult szoftver rendszerekkel is kell majd foglalkozniuk. Az *algoritmikus gondolkodás* fejlesztése pedig nem elegendő ahhoz, hogy valós képet adjunk erről a teendőről, hiszen rengeteg olyan probléma van, ami nem algoritmikusan nehéz. A programozás és a *rendszerszintű gondolkodás* ötvözetét *architektúrális gondolkodásnak* nevezem, ezzel is utalva a szoftverarchitektúrákra és az architect munkakörre. Fontos azonban kihangsúlyozni, hogy a *rendszerszintű gondolkodás* nem csupán az informatikus számára hasznos és fontos, hiszen számtalan más műveltségi területnél előkerül ez a fogalom (pl.: természettudományok, társadalomtudományok), ami igen jól definiálható és körül írható konkrét tantárgy vagy terület nélkül is.

Az oktatásba való integrálása ennek a kompetenciának egyáltalán nem egyszerű. A szakirodalomban a természettudománnyal kapcsolatban találtam csak cikket [12]. Jelen tanulmány az informatika oktatásban való megjelenítésére kíván ötletet adni. Véleményem szerint a *mikrokontrollerek*, *beágyazott rendszerek* elterjedésével, valamint az *IoT* térhódításával (oktatásban is) logikusan adódik ennek alkalmazása egy *valósidejű* megközelítésben.

Irodalom

1. Németh Tamás, Sárkány Rita, Tornai Henrietta, Wiandt Zsófia, Németh-Szabados Klára Viktória, Holló Csaba: A programozás oktatásának motivációi a közoktatásban In: Szlávi Péter, Zsakó László (szerk.) INFODIDACT 2016 ISBN: 978-615-80608-0-6
2. Nemzeti Alaptanterv. http://www.okm.gov.hu/
3. Szlávi Péter, Zsakó László: Informatikai kompetenciák - A valós világ modellezése In: Szlávi Péter, Zsakó László (szerk.) INFODIDACT 2013 ISBN:978-963-08-8387-0
4. Richmond, B. (1994). Systems Dynamics/Systems Thinking: Let's Just Get On With It. In International Systems Dynamics Conference. Sterling, Scotland
5. Benson, H., Borysenko, J., Comfort, A., Dossey, L., & Siegel, B. (1985). Economics, Work, and Human Values: New Philosophies of Productivity. The Journal of Consciousness and Change, 7(2), 198
6. Sweeney, L. B., & Sterman, J. D. (2000). Bathtub dynamics: initial results of a systems thinking inventory. System Dynamics Review, 16(4), 249–286. doi:10.1002/sdr.198
7. Stave, K. A., & Hopper, M. (2007). What Constitutes Systems Thinking? A Proposed Taxonomy. In 25th International Conference of the System Dynamics Society. Boston, MA
8. Kopainsky, B., Alessi, S. M., & Davidsen, P. I. (2011). Measuring Knowledge Acquisition in Dynamic Decision Making Tasks. In The 29th International Conference of the System Dynamics Society (pp. 1–31). Washington, DC.
9. Squires, A., Wade, J., Dominick, P., & Gelosh, D. (2011). Building a Competency Taxonomy to Guide Experience Acceleration of Lead Program Systems Engineers. In 9th Annual Conference on Systems Engineering Research (CSER) (pp. 1–10). Redondo beach, CA.
10. Arnold, R. D. és Wade, J. P. (2015): A Definition of Systems Thinking: A Systems Approach Procedia Computer Science, 44, 669–678.
11. Szlávi Péter, Zsakó László: Informatikai kompetenciák: Algoritmikus gondolkodás https://people.inf.elte.hu/szlavi/InfoDidact10/Manuscripts/ZsL_SzP.pdf (utoljára megtekintve: 2018.11.05)
12. Veres, G. (2017): Gondolkodási készségek azonosítása és fejlesztése a biológia tantárgyban tankönyvelemzés. In. Zsolnai Anikó és Kasik László (szerk.): A tanulás és nevelés interdiszciplináris megközelítése. Budapest, Magyar Tudományos Akadémia Pedagógiai Tudományos Bizottsága
13. Kovácsné Pusztai Kinga: A probléma–alapú oktatás az informatika órán In: Szlávi Péter, Zsakó László (szerk.) INFODIDACT 2017 ISBN: 978-615-80608-1-3
14. Mérő László: A mesterséges intelligencia és a kognitív pszichológia kapcsolata. Tankönyvkiadó (1989)
15. Mahler Attila: A magyar és a brit informatika kerettanterv 2012-es megújítása In: Szlávi Péter, Zsakó László (szerk.) INFODIDACT 2013 ISBN:978-963-08-8387-0
16. Dusza Árpás Országos Programozói Emlékverseny: https://isze.hu/dusza-arpad-orszagos-programozoi-emlekverseny/ (utoljára megtekintve: 2018. 11. 05.)
17. Bernát Péter: Feladattípusorientált játékfejlesztés a Scratchben In: Szlávi Péter, Zsakó László (szerk.) INFODIDACT 2017 ISBN: 978-615-80608-1-3
18. Zsakó László: Tehetséggondozás az informatikából: https://people.inf.elte.hu/szlavi/InfoOkt/Eloadasok/Tehetseggondozas.pdf (utoljára megtekintve: 2018. 11. 05.)
19. Módi József (2015): A Raspberry Pi számítógép a gyakorlati oktatásban

20. Michael Kölling (2016), Educational Programming on the Raspberry Pi (elérhető: https://www.researchgate.net/publication/304455824_Educational_Programming_on_the_Raspberry_Pi, utoljára megtekintve: 2018. 11. 05.)
21. Brian W. Evans írása alapján fordította, kiegészítette és frissítette Cseh Róbert (2011): Előszó, Arduino, Szerkezet – struktúra In: Arduino Programozási kézikönyv, TavIR (5-12. oldal) (elérhető: http://www.peschka.hu/userfiles/7/files/tavir_arduino_notebook.pdf, utoljára megtekintve: 2018. 11. 05.)
22. Heizlerné Bakonyi Viktória, Ifj. Illés Zoltán, Illés Zoltán: Valós időben, valós világban In: Szlávi Péter, Zsakó László (szerk.) INFODIDACT 2015 ISBN: 978-963-12-3892-1
23. Windows 10 IoT Core, Developing foreground applications: https://docs.microsoft.com/en-us/windows/iot-core/develop-your-app/buildingappsforiotcore (utoljára megtekintve: 2018. 11. 05.)
24. Firebase: https://firebase.google.com (utoljára megtekintve: 2018. 11. 05.)
25. Firebase samples: https://firebase.google.com/docs/samples/ (utoljára megtekintve: 2018. 11. 05.)

Játékosítás (gamification) az oktatásban

Kovácsné Pusztai Kinga

kinga@inf.elte.hu
ELTE IK

Absztrakt. A gyors technológiai változások megjelenésével a generációk közötti különbségek egyre markánsabbá válnak. A most felnövekvő korosztály gondolkodásmódja és életmódja gyökeresen megváltozott. A néhány évtizede még eredményes pedagógiai módszerek egy része mára már elavulttá vált, helyükre újakat kell keresnünk; olyanokat, amelyek utat találnak a Z, illetve az alfa generációk szemléletéhez. Erre kínál egy lehetőséget a játékosítás (gamification, gamifikáció) módszere, a játékelemek alkalmazását jelenti valamely célterületen, esetünkben az oktatás folyamatában. A játékosítás 2010-től kezdődően vált egyre ismertebbé. Elsődlegesen az online térben alkalmazzák, de az eredeti koncepció nem zárja ki az offline alkalmazási lehetőségeket sem. A cikkemben a játékosítás bemutatásán, illetve a közoktatásban betöltött szerepén túl azzal foglalkozom, hogy hogyan lehetne azt egy egyetemi kurzusba is beépíteni.

Kulcsszavak: gamification, informatika oktatás, számítógépes gondolkodás, edutainment

1 A generációs különbségek

Az első *betűvel jelölt korosztály*[23], az *X generáció* tagjait, a mai 36-50 éveseket, a „*digitális bevándor*ló" névvel illetik. Ők azok, akik felnőtt korukban kerültek közel az internethez. A *Z* generáció tagjai jórészt a középiskolások vagy felső tagozatosok, de első képviselőik már megkezdték egyetemi tanulmányaikat. Az alsó tagozatos és kisebb gyerekek alkotják az alfa generációt. Az *alfa* és a *Z generáció* tagjait illetik a *„digitális bennszülött"* névvel is. Ők már úgy nőttek fel, hogy gyermekkoruktól elérhető volt az internet. Számukra magától értetődő a személyes kommunikációs eszközök használata, okostelefonnal kelnek és fekszenek, mindig elérhetők és folyamatosan kapcsolatban vannak egymással az online térben. Könnyen kezelik az információ gyors áramlását, tevékenységeiket gyakran váltogatják „multitasking" során. Így a hagyományos, frontális eszközökkel nehéz lekötni a figyelmüket. A vizuális megjelenítést részesítik előnyben, szemben a hosszú, tagolatlan szövegekkel.

Az egyes korosztályok között napjainkban egyre gyorsabban mélyül a generációs szakadék, ami egyre nehezebb feladat elé állítja a pedagógusokat. A tanárok nagy része az X generációhoz tartozik; ők azok, akik már rendelkeznek tapasztalattal, de nem szívesen közelednek az innovatív módszerekhez. Az *X* és *Z* közötti különbséget legkönnyebben a közbülső Y generáció tagjai, a fiatal tanárok tudják áthidalni, akik megértik mindkét korosztály kommunikációs sajátosságait.

A mai pedagógustársadalomra jelentős feladat hárul. Meg kell érteni a netgeneráció új nyelvezetét, kommunikációs és motivációs struktúráját, el kell fogadni, hogy megváltozott az információ befogadásának és közlésének, illetve a figyelemnek a kultúrája. Az oktatási módszereket is ennek megfelelően – *új didaktikai eszközök* bevezetésével - változtatni kell. Erre ad ötletet az oktatás gamifikálása, vagyis a játékosság elemeinek bevitele az ismeretátadás folyamataiba.

2 A gamification definíciói

A *gamification*[1] kifejezés a game (játék) és a fication (valamilyenné alakítás) szavak összetételével keletkezett és magyar nyelven *játékosításnak*, illetve *gamifikációnak* is szokták nevezni.

Az új fogalmat először Nick Pelling [22] definiálta 2002-ben a következőképpen: „elektronikus eszközök játékszerű felhasználói felületekkel való felgyorsítása és élvezhetőbbé tétele". Ez a megha-

tározás és jelentése azóta több lépésben fejlődésen ment keresztül és ettől eltérő jelentéssel került be a köztudatba.

Napjainkban Deterding 2011-ben alkotott definícióját[2] idézik és alkalmazzák leggyakrabban, amely szerint a gamification jelentése: *„a játéktervezési elemek használata játékon kívüli kontextusban”*.

Manapság jónéhány további meghatározást ismerünk; egyesek bővítették, mások szűkítették az értelmezést. A fogalom egyes részeit tovább finomították a szerzők[3]…[8]. A legérdekesebb és talán a legnagyobb változtatás az, amely a gamification szisztematikus felfogása helyett annak kísérleti (tentative) jellegét hangsúlyozza (Huotari és Hamari, 2012)[9].

3 A gamification kialakulása, elterjedése

A gamification fogalma a köztudatban a 2000-es években jelent meg és 2010-től vált elterjedtté, eredete azonban sokkal korábbra tehető. Fuchs[4] egészen a római korig visszanyúlóan talált példákat a gamifikáció hadászatban történő alkalmazására. Zichermann és Linder[5] arra mutatott rá, hogy *Napóleon* is a gamifikáció szemléletéhez fordult 1795-ben, amikor a háborús élelemszállítmány tárolásának megoldására kiírt egy országos feladványt (12.000 frankos jutalommal). A játéknak köszönhetően Nicolas Appert serfőzőmester és cukrász – már Pasteur előtt – feltalálta a pasztörizálási eljárást, amely által az ételek több, mint 4 hónapon keresztül megőrizhették szavatosságukat.

A manapság elterjedt hűségprogramok egyik ősének az *S&H Társaság* által 1896-ban megvalósított *Green Stamps*[10] tekinthető. A program keretén belül a vásárlók zöld bélyegeket gyűjthettek egy pontgyűjtő könyvbe. Amikor a könyv betelt, a bélyegek értékét a vásárlók különféle érdekes termékekre válthatták.

Manapság is számos cég alkalmazza a gamification módszerét. Például a *Nissan Leaf* [14] a vezetés játékosabbá tételére kitalálta, hogy a sofőr útja során fákat nevelhet a helyes vezetési stílus által. Finomabb vezetéssel üzemanyagot spórol meg, így gyorsabban nő a fája. Amennyiben a fa felnőtt, akkor újabb fa növesztésébe kezdhetünk. A Nissan közösségi hálóján (Eco ranglista) versenyezhetünk is egymással a legbiztonságosabb vagy a legzöldebb autóvezető címekért.

20. ábra:
A Duolingo ikonja [26]

Egyre többen ismerik és használják a *Duolingo-t*[26], az egyik legismertebb online nyelvoktató applikációt, amely több mint 60 nyelvkombinációban kínál online kurzusokat. Az egyik nyelv mindenképpen az angol, ami azt jelenti, hogy vagy angolul tanulunk egy idegen nyelvet, vagy valamelyik nyelven tanuljuk az angolt. A Duolingo-val minden nap kell foglalkoznunk; versenyezhetünk is társainkkal és tanulócsoportba tömörülhetünk. A tananyag tematikus részekre tagolódik. Az egyes ismeretanyagokon belül szinteket találunk és minden szint több leckéből áll.

A szintenkénti leckék sorrendjét magunk választhatjuk meg, de újabb ismeretanyagra csak az összes lecke teljesítése után léphetünk. Mód van arra, hogy szintfelmérővel a saját tudásszintünkön kezdjünk, illetve ellenőrző pontok teljesítésével egyszerre több leckét átugorhatunk. Tanulásunk során *„lingot”-ot* gyűjtünk, (ez a Duolingo virtuális valutája,) amit azután a hálózat boltjában különleges eszközökre válthatunk be.

4 A gamification jellemzése

A gamifikáció meghatározásaiban két fontos fogalom jelenik meg: a játékelemek és a játékmechanizmusok. A kettő együttes alkalmazása vezet a játéktervezési technikákhoz. A *játékelemeken* a ha-

gyományos és videojátékokból vett eszközöket értjük, a *játékmechanizmusok* pedig a játékok működési elvének alkalmazását jelentik. Rigóczki[25] a következő összetevőket sorolja fel:

- A történet (eseménysor és cél)
- A megjelenítés (látvány)
- Elemekre bontás (szakaszok, feladatok és a hozzá kapcsolt jutalmak, pl. pontozás)
- Azonnali és állandó visszacsatolás
- Küldetések (független, de jutalmat érő elágazások)
- Pontok, jelvények, kitűzők, ranglisták (eredményesség jelző elemek)
- Szintek (fejlődés, határok)

Az eszközök csak akkor működnek hatékonyan, ha a játék mechanizmusai jó minőségben adottak: a játék önkéntes, sikert ígérő, átlátható és kellően lehatárolt („ideje van").

A definícióban szereplő „játékon kívüli kontextus" pedig arra utal, hogy más a célja a játéknak és más a játékosításnak. Játék, és játékosítás között az a legnagyobb különbség, hogy a játékban az örömet és az élvezetet keressük, a játékosított alkalmazásban pedig a való élet egy szegmensében, egy előre meghatározott cél elérésére törekszünk, újszerű, rugalmasabb feltételek mellett.

5 Az oktatás játékosítása

5.1 Mit lehet játékosítani?

5.1.1 Az órák menete

Egy óra sokkal érdekesebb lehet, ha a tananyag frontális leadása helyett a hallgatókat engedjük *versenyezni*, vagy valamilyen *szerepjátékot* alkalmazunk. Ezen módszerek alkalmazásának célja az, hogy a diákok a passzív befogadás helyett aktívan részt vegyenek az órán.

Ezeknél még eredményesebb módszer lehet az, ha a hallgatókat valamely *publikus szoftver használatával* vonjuk be az órába. Ilyen applikációk például a *Socrative*, vagy a magyar nyelvű *Redmenta*, de sokan használják már az *Edmodót* is. Ezekkel a programokkal könnyen előállíthatunk gyorstesztet vagy feladatot, amelyet a diákok az okostelefonjukon oldanak meg, így kedvenc eszközüket az órai aktivitásra használják. Az ilyen *„app"-ok* alkalmazásával a diákok motiváltabbá válnak, nő az interaktivitásuk, továbbá gyorsabban és pontosabban kapunk visszacsatolást a tananyag megértéséről. Ezek az eszközök nem csak a csoport munkáját támogatják, hanem fontosak lehetnek az egyéni tanulási élményben is.

5.1.2 A számonkérés

Talán ez az a terület, amelyről a legtöbb forrás található. A magyar közoktatásra jellemző, hogy a tanulók egy félévben írnak néhány dolgozatot, amelyekre jegyet kapnak és a néhány (van olyan tárgy, ahol mindössze kettő) jegy átlagából alakul ki a félévi értékelés. Az egyetemen még kevésbé rugalmas módon, általában két zárthelyi képezi a féléves gyakorlati jegy alapját. A vizsgajegy pedig mindössze egyszeri teljesítmény eredménye. A zárthelyi dolgozatoknak és a vizsgáknak túl nagy lesz a súlyuk, amit a mai diákok „stresszesen" élnek meg.

Ehelyett már elterjedőben van a *pontozásos módszer*[24], amely mely Prievara Tibor nevéhez fűződik. Javaslata szerint a tanítás folyamatát célszerű bizonyos egységekre, például havi periódusokra osztani. Az egyes időszakok alatt a tanulók pontokat szerezhetnek dolgozataikkal, de a feleleteik, továbbá a házi feladatok, a szorgalmi feladatok, a beadandóik vagy az egyéb értékelhető teljesítmények is pontszerző lehetőséget jelentenek. A pontokat a periódus végén válthatjuk át osztályzatra, így minden hónapban egy több forrású jegyet kapnak a munkájukra.

A pontokon felül még *szinteket* is el lehet érni. A pontok folyamatos tájékoztatást adnak, a szintek azonban csak egy adott pontmennyiség elérése után lépnek életbe, így középtávú visszajelzést nyújtanak. A *jelvények*[17] vagy *tanúsítványok* egy adott értékelhető eseményt igazolnak vissza. Például egy hallgató részt vesz egy konferencia előadáson, vagy egy általa választott témából kiselőadást tart, a rendszer megjutalmazza egy speciális jelvénnyel.

A pontozásos módszer előnyeiről több szerző is beszámol – mind a diákok, mind a tanárok szemszögéből –, így például Fromann és Damsa[12], illetve Kenéz[13]. A diákok az órán motiváltabbak és aktívabbak voltak, ennek hatására jobban átlátták saját tevékenységeiket, illetve tudatosabban tudtak maguk elé állítani célokat. A tanárok sűrűbb és értékelhetőbb visszacsatolást kaptak, így jobban látták a tananyag elsajátításának mértékét, illetve a tanulói igényt.

A pontrendszerek egyik nagy előnye az, hogy elsősorban a *fejlődés folyamatára* fókuszálnak, amit a pontok gyűjtögetése mentális szinten is megjelenít a tanulók számára. Míg a jegy alapú értékelés átlagokat számít, addig a pontok lehetőséget adnak a gyarapodás és a haladás érzetének átélésére. Egy ilyen környezetben a diák a rosszabb jegy miatt nem a kudarcot fogja érezni, hanem azt, hogy valamennyivel még így is közelebb került a következő szint eléréséhez.

A pontrendszer további nagy előnye, hogy a diákoknak *döntési lehetőségeket* ad. Egy-egy szorgalmi feladattal, beadandóval pontot tud szerezni, így javíthatja egy elrontott dolgozatát. Azonban úgy is dönthet, hogy ha valamelyik feladatot nincs kedve megcsinálni, akkor azt kihagyhatja, mérlegelve, hogy abban a feladatban nem szerez pontot.

5.2 Mire figyeljünk?

Fromann[18] három tényezőt emel ki, amelyeket alapvetőnek tart, így ezekre – ajánlása szerint – mindenképpen figyelni kell.

5.2.1 Az optimális terhelés

A játék akkor ad sikerélményt, ha az *megfelelő nehézségű*. A túl könnyű feladatok unalmassá, a túl nehezek frusztrálóvá válnak. Természetesen, nem az egyes feladatokról van szó, hanem a játék egészéről. Egy-egy könnyű feladattal például sikerélményt adhatunk és a bekapcsolódást segíthetjük a lemaradt diákoknál, míg a nehéz feladatok inspirálják a legjobb diákokat. Gyakori mechanizmus a játékokban is, hogy a szinteket egy különösen nehéz szinttel zárják le („boss level”).

5.2.2 Az ideális szintezés

Ez a feladatok és a jutalmak rendszerbe illesztését jelenti. Fontos, hogy minden komoly játéknak van egy elérendő nagy célja, de a végső cél mellett szükséges kisebb célokat is felállítani, a motiváció fenntartása érdekében. A közbülső célok elérése is jutalommal jár. Ezt nevezzük *szintezésnek*. Minél több szintre tagolódik a játék, annál több olyan kis célt tartalmaz, amelyek pozitív élményhez juttatják a tanulókat.

5.2.3 Az ideális jutalmazási rendszer

Minden kisebb, értékelhető teljesítés után azonnali, pozitív visszacsatolás, *jutalmazás* történik, ami a tanulót megerősíti és nem engedi elbizonytalanodni. Természetesen, ezeknek a jutalmaknak arányosnak kell lenniük a teljesítménnyel.

5.2.4 Döntések, választások

Kenéz[13] fontosnak tartja még kiemelni a következő szempontot.

A tanulók számára értelmes választásokat és döntési lehetőségeket kell biztosítani a játékosítás keretein belül. Nem eléghetünk meg azzal, ha csak sodródik az eseményekkel, hanem résztvevő módon, lehetőleg alakítania is kell azokat.

Minden olyan területen, ahol különböző személyiségű résztvevők kerülnek alkotó munkakapcsolatban (ilyen pl. a team-munkában végzett informatikai fejlesztés is), érdemes figyelembe venni a résztvevők lehetséges orientációit. Ez az oktatás játékosítására is érvényes ajánlás.

5.2.5 A tanulók orientációja

Az oktatás közösségi, tantermi színtereire is alapvetően érvényes *Bartle taxonómiája*[16], amely négy jellegzetes csoportra bontja a játékosokat, tekintettel eltérő céljaikra, viselkedési mintáikra és a követett motivációikra. Ezek típusok a következők:

- Teljesítők, akik az eredményességet tekintik meghatározó célnak.
- Felfedezők, akik a lehetőségeket, a nem mutatkozó ismereteket, a „titkokat" keresik.
- Társaságiak, akik számára a játék élménye a másokkal való együttléthez köthető.
- Itt is jelen vannak a „killerek", akik általában a többiek ellen játszanak.

5.3 Mit használhatunk fel a tanításban?

A gamifikáció alkalmazása során a játékok rendszeréből *átveszünk olyan elemeket*, amelyek segítségével motiváltabbá tehetjük diákjainkat, csökkenthetjük a rájuk nehezedő stresszt, valamint segíthetünk nekik, hogy önállóbbá váljanak és részesei legyenek a tanulás során meghozandó döntéseknek. Nádori[21] a következő elemeket nevezi meg:

5.3.1 Önállóság

A játék során a tanulók kaphatnak ugyan segítséget, de a megoldást mégis nekik kell önállóan megkeresni. Ez az *önállóság*, bár több időt vesz igénybe, mégsem hagyható el, mivel ez teremt lehetőség a kísérletezésre és újra tervezésre.

5.3.2 Unalom ellenszere

Sokan és sokat panaszkodnak arra, hogy a diákokat manapság „semmi sem érdekli", ami az iskolában történik. Ha azonban a szárazabb feladatokhoz – a gépiesség és az *unalom ellen* – játékosságot tudunk kapcsolni, akkor a tanulók sokkal szívesebben vesznek részt az órákon.

5.3.3 Célok

Fontos, hogy legyenek rövid-, közép-, és hosszú távú *céljaink* is, amikor gamifikáció projekjét tervezünk. Nem elég azt mondani például, hogy „el kell foglalni egy várat, és erre van három hónapotok", hanem folyamatosan közelebbi, kisebb célokat is meg kell határoznunk, továbbá még azt is világossá kell tennünk, hogy ezek rendre hogyan viszonyulnak a végső célhoz.

5.3.4 Siker és kudarc

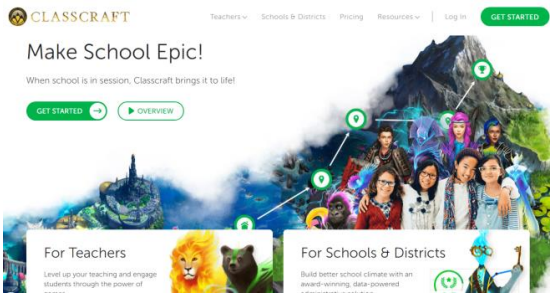
A játékok alapvetően másként viszonyulnak a *siker és kudarc* kérdéséhez, mint a hagyományos iskolai értékelés, és ezt érdemes kihasználni. Tanári és szülői beszámolókból ismert jelenség, hogy egy rossz érdemjegy annyira elkedvetlenítheti jelen korszak kisdiákjait, hogy azt nehéz helyrehozni. A játékok, amelyekkel már jóval az iskola előtt találkoztak, még vesztés esetén is arra ösztönöznek, hogy ismét indítsuk el a játékot és próbálkozzunk újra.

5.3.5 Azonnali visszajelzés

Meghatározó eleme a jól megalkotott játékoknak az is, hogy folyamatosan, a kisebb célok elérésénél is jutalmat, azaz sikerélményt kapunk. A játék szemlélete a *pozitív visszajelzés* előtérbe helyezése, ugyanis nem a hibát büntetjük, hanem az erőfeszítést értékeljük. Természetesen így is el kell érnünk az eredményt, ebből nem engedhetünk, de egészen más az oda vezető út légköre.

6 Példák a gamification megjelenésére az oktatásban

6.1 Classcraft



21. ábra: Classcraft kezdőoldala[27]

A *Classcraft* [20] egy olyan ingyenesen letölthető alkalmazás, melyben a diákok belecsöppenhetnek egy *szerepjátékba*, ám a játék a *való életben* zajlik. A tanulóknak be kell regisztrálniuk, avatárt kell csinálniuk és választani kell a három meglévő kaszt (tkp. *szereptípus*) közül (harcos, mágus, gyógyító), melyeknek különféle képességük van. A tanár csoportba rendezi a diákokat úgy, hogy minden csapatba kerüljön a mindhárom kasztból résztvevő.

Mindenkinek vannak *életpontjai* és tapasztalati pontjai, melyeket a különféle, a való életben zajló feladatokért szerezhetnek, illetve elveszíthetnek. Ilyen például a házi feladat elkészítése, vagy az órán való figyelés. Ha valaki nem készíti el időben a házi feladatot, vagy zavarja az óra menetét, akkor életpontokat veszíthet, de ha jó jegyeket szerez, akkor azokért bónuszokat kaphat. A csoportban lehetőség van gyógyításra is. A bónuszpontokat különböző jutalmakra is be lehet váltani, például több időt kaphatnak egy dolgozatra. Ez az alkalmazás nagyban segíti a csoportmunka fejlődését. A *Classcraftot* jelenleg több, mint 25 országban és több, mint 20.000 iskolában használják.

6.2 Bee the Best

22. ábra: Egy osztály értékelése a Bee the Best-ben[28]

2015-ben szintén Prievara Tibor módszere alapján egy magyar oktatókból álló csapat létrehozta a *BeeTheBest*[19] nevű oldalt, amely tökéletesen alkalmazható a *pontrendszert* használó tanárok számára.

A weboldal egy olyan online felület, ahol a tananyagot *szintekre* osztva tanítják a tanárok. Ezek időtartamát is a tanár határozza meg. A szintekhez kerettörténet is tartozik. A szint alatt a diákok feladatok elvégzésével pontokat gyűjtenek. Azt, hogy mire hány pont jár, a tanár dönti el. Az összegyűjtött pontok a szint végén jegyre válthatók, a hozzá kapcsolódó ponthatárokat a tanár határozza meg. Pontokon túl, különféle jelvényekkel is jutalmazhatók a diákok.

A diákok egy játékos, *interaktív felületen* követhetik nyomon, hogy hol tartanak. Ez a felület a tanárok számára is visszacsatolást nyújt, mivel könnyen leszűrhető belőle ki az, aki aktívan tanul és ki az, aki könnyedebben veszi az egész tanulási fázist.

7 Edutainment

A gamification egyik érdekes alesete az, amikor különféle koncepciókat játékok segítségével tanítunk meg. Az olyan játékokat, melyek oktatási céllal készültek, (az oktatás és a szórakozás szavak összekapcsolásával keletkező szóval) *edutainmentnek* nevezik. Az ilyen oktató játékok nagyon hatékonyak abban az estben, amikor unalmas rutinfeladatokat kell gyakoroltatnunk (például matematikában műveletek törtekkel), de akkor is, ha nagyon bonyolult koncepciókat szeretnénk megértetni (például egy ökoszisztéma működését).

Az edutainment elnevezés kifejezetten oktatási célra készített játékokra utal, nem pedig már meglévő, szórakoztató videókjátékok alkalmazását jelenti. Ilyen edutainment játékokat gyűjtött össze Nádori Gergely és Prievara Tibor az ingyenesen elérhető *Kis-nagy IKT könyvükben*[15].

Az interneten számos ilyen alkalmazás ingyen elérhető, egyre többször találkozunk például az idegen nyelvet oktató tankönyvek, vagy nyelviskolák részére fejlesztett online gyakoroltató játékos programokkal. Magunk is könnyen létre tudunk hozni a *Quizlet* weboldalon[11] digitális tanítókártya csomagokat bármilyen tantárgyhoz. Ezekből aztán néhány kattintással generálhatunk dolgozatokat vagy interaktív táblán (otthon is) játszható fejlesztő játékokat.

8 A gamification megjelenése az algoritmusok és adatszerkezetek I. egyetemi kurzuson

Az *Algoritmusok és adatszerkezetek* c. tantárgy az ELTE *Programtervező informatikus* képzés reformjában egy félévvel előbbre kerül, azaz a korábbi 3. és 4. félév helyett a 2. és 3. félév része lesz. A tárgy elméleti jellege miatt a jövőben várhatóan nehezebben fogják azt a hallgatók teljesíteni, éppen ezért, erre készülve, elkezdtem ebben a kurzusban a *gamification* lehetőségeit használni. A változtatásokat a hallgatókkal anonim kérdőív formájában véleményeztettem is. A kérdőíveket – nem-kötelező módon – 35 hallgató töltötte ki. A részvétel felülmúlta az általam várt eredményt, mivel 42 hallgatót tudtam megszólítani, azaz 83%-os volt a hallgatók önkéntes részvétele, annak ellenére, hogy ezért mindösszesen 2 jutalompontra számíthattak. A részvételi arány és a válaszok azt mutatták, hogy a hallgatók is látnak fantáziát a kurzus gamifikálásában.

Kezdeti lépésként, ebben a félévben bevezettem a *pontrendszert,* azaz értékelésnél nem csak a zárthelyi eredmények számítanak. A hallgatóknak két zárthelyi dolgozatot kell írniuk, mindegyik 60 pontos. Mindkét zárthelyin a kötelezően elérendő minimum a 20 pont. Továbbá, minden órán adok házi feladatot, amit nem kötelező megoldani, azonban plusz pontszerzési lehetőséget jelent. Összesen 20 kiegészítő pontot szerezhetnek a házi feladatokból. Ezen felül kapnak programozási feladatokat, melyekből hasonlóan maximum 20 pontot szerezhetnek. Továbbá pontokkal jutalmazom azokat is, akik a tananyaghoz kapcsolódó témában komolyabb kutatásokat végeznek, és azt velünk valamilyen formában megosztják. A félév végi jegyüket ezen pontszámaik összesítése alapján határozom meg. (Ha valaki nem éri el a kötelező zárthelyi minimumot, akkor pótzh-t kell írnia; ezt nem lehet egyéb pontozással kiváltani.)

Továbbá, a tárgy egyes témaköreihez, a tudásuk elmélyítését szolgáló *Quizlet segédletet*[11] dolgoztam ki. Az alkalmazást, amelyet otthoni gyakorlásra szánok, egyaránt tartalmaz játékokat és teszteket is. Fontosnak tartom az otthoni gyakorlás értékelését is, azonban ennek jutalma nem pontban, hanem ún. „*lehetőségekben*" történik. A hallgatóknak el kell küldeni egy legalább 80%-ra megoldott tesztet, és kapnak érte 5 perc „lehetőséget"; a 90%-nál jobb teszt elküldése 10 percet ér. A lehetőségek beválthatók a zárthelyin, ha még több időt szeretnének, de beváltható bármely órán későbbi érkezés, vagy korábbi távozásra is. Sok jó teszt kitöltésével akár eggyel több órai hiányzás is engedélyezett. A „lehetőségek"-kel történő értékelést nemcsak azért tartom fontosnak, hogy változatos értékelési módszert használjunk. Azt gondolom, hogy e tevékenységeket nem lenne jó pontokra váltani, hiszen ekkor túlértékelnénk egy „játéktevékenységet".

A *hallgatók véleménye* a pontozási rendszerről pozitív, mindösszesen két hallgatónak (6%) tetszik jobban a hagyományos, jegy alapú értékelési mód.

A hallgatók véleménye a *Quizlet* segédletről pozitív, egy hallgató kivételével mindenkinek tetszett a használata, két hallgató kivételével pedig azt nyilatkozták, hogy segített a tanulásban az alkalmazás használata. Számomra érdekes volt, hogy akinek nem tetszett a *Quizlet* segédlet, annak ellenére azt is segítette a tanulásban, illetve az a két hallgató, akit nem segített a *Quizlet* a tanulásban, annak ellenére úgy nyilatkoztak, hogy tetszett nekik a használata.

Bár nem volt kötelező véleményt írni róla, ebből is több válasz született, mint amennyire számítottam. Ezekből néhány: „Hasznos és egyszerűen lehet vele tanulni.” „Nagyon jó, így vettem észre, hogy vannak hiányosságaim.” „Lényegre törően összefoglalja, amit tudni kell, nagyon hasznosnak és klassznak tartom.” „Ötletes, és rengeteget segít, hogy gyakorlatban is elsajátítsuk a megszerzett tudást :-)”

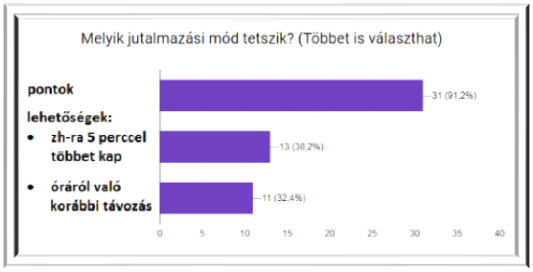
A *Quizlet* használata számomra is váratlan segítséget jelentett, mivel statisztikát készít a hallgatók megoldásairól. Könnyen láthatóvá válik az, hogy melyek azok a példák, amelyeket a hallgatók gyakran elrontanak, és melyek azok, amelyeket szinte sosem.

A *Quizlet* használatának értékeléséről már közel sem volt ilyen egyértelmű a hallgatók véleménye. Kevesebb, mint a hallgatók 2/3-a vélte jónak (55%) a lehetőségekkel történő jutalmazást, a többiek (45%) maradtak volna itt is inkább a hagyományos pontoknál.

4. ábra: Quizlet használatának értékelése

Ennek ellenére, a fentebb bemutatott és részletezett okoknál fogva, elkötelezettnek érzem magam a didaktika korszerű eszköztára irányában, és nem áll szándékomban változtatni az értékelési módon.

A hallgatók véleménye a jutalmazási módok értékelésénél is megoszlott valamennyire. Míg a pontozásos jutalmazási mód csaknem mindenkinek tetszett (92%), addig a „zh-n több időt kapnak” már csak 38%-uk, az „óráról való korábbi távozás” lehetősége pedig csak a hallgatók 32%-a találta vonzónak.



5. ábra: Jutalmazási módok

A kérdőívek kiértékelése alapján – külön tekintettel a nagyarányú részvételre – elmondható, hogy *a kurzus gamifikálása egyértelműen elnyerte a hallgatók tetszését*.

9 Összegzés

Rohamosan változó világunkban a tanárok sem tudnak megmaradni a hagyományos módszerek használatánál, ha sikeresen szeretnének tanítani. *Új irányokkal* és *elemekkel* kell bővíteniük didaktikai eszköztárukat. Mivel a ma felnövekvő generáció tagjai már egy online világba születtek, az oktatásnak is nyitnia kell az *okos eszközök* felé. Erre a szakmódszertan támogatása mellett, egy lehetséges új szemléletet nyújt a *gamification*, a játékosítás bevezetése és alkalmazása az oktatásban. A cikkemben – a gamifikáció bemutatásán és rövid jellemzésén túl – a módszer iskolai alkalmazására ismertettem néhány lehetőséget.

Végül pedig *saját egyetemi kurzusom* „gamifikálásának” lépéseit és hallgatóim tapasztalatait tekintettem át. A hallgatók magas részvétele a felmérésben, illetve válaszaik tartalma egyértelműen igazolták azt, hogy nem csak a közoktatásban, hanem a *felsőoktatásban* is helye van az oktatás gamifikálásának, vagyis a megfelelően megválasztott játékelemek alkalmazása növeli az ismeretátadás folyamatának hatékonyságát és minőségét.

10 Irodalom

1. *Wikipedia* https://hu.wikipedia.org/wiki/Gamification (utoljára megtekintve: 2018. 10. 19.)
2. Deterding, S., Sicart, M., Nacke, L., O'Hara, K., & Dixon, D: *Gamification. Using Game-Design Elements in Non-Gaming Contexts.* In CHI'11 Extended Abstracts on Human Factors in Computing Systems (2011) (pp. 2425-2428). ACM. http://dl.acm.org/citation.cfm?id=1979575 (utoljára megtekintve: 2018. 10. 19.)
3. Zichermann, G., & Cunningham, C: *Gamification by Design: Implementing Game Mechanics* (2011) in Web and Mobile Apps. Sebastopol: O'Reilly Media.
4. Németh, T.: *English Knight: Gamifying the EFL Classroom* (Unpublished master's thesis).(2015) Pázmány Péter Katolikus Egyetem Bölcsészet- és Társadalomtudományi Kar, Piliscsaba, Hungary. https://ludus.hu/gamification/ (utoljára megtekintve: 2018. 10. 19.)
5. Zichermann, G., & Linder, J.: *The Gamification Revolution: How Leaders Leverage Game Mechanics to Crush the Competition.* (2013) McGraw-Hill.
6. Kapp, K. M.: *The Gamification of Learning and Instruction: Game-based Methods and Strategies for Training and Education.* (2012) San Francisco, CA: Wiley.
7. Werbach, K., & Hunter, D: For the Win: *How Game Thinking Can Revolutionize Your Business.* (2012) Philadelphia: Wharton Digital Press.
8. Burke, B.: Gamify: *How Gamification Motivates People to Do Extraordinary Things*. (2014) Brookline, MA: Bibliomotion.
9. Huotari, K., & Hamari, J.: *Defining Gamification - A Service Marketing Perspective.* In Proceedings of The 16th International Academic Mindtrek Conference, Tampere, Finland, October 3-5, 2012.
10. Nagy Júlianna Roberta: *A gamification megjelenése a magyar középiskolákban* (2017) https://www.ecosim.hu/ecosim_www/data/downloads/51/nagy_julianna_a_gamification_megjelenese_a_magyar_kozepiskolakban.pdf (utoljára megtekintve: 2018. 10. 19.)
11. Prievara Tibor: *Mindent a Quizlet-ről* (2011) in Tanárblog http://tanarblog.hu/internet-a-tanoran/1977-mindent-a-quizlet-rl (utoljára megtekintve: 2018. 10. 19.)
12. Fromann Richárd, Damsa Andrei: *A gamifikáció (játékosítás) motivációs eszköztára az oktatásban* (2016) http://folyoiratok.ofi.hu/uj-pedagogiai-szemle/a-gamifikacio-jatekositas-motivacios-eszkoztara-az-oktatasban (utoljára megtekintve: 2018. 10. 19.)
13. Kenéz András: *A játékosítás (gamification) a felsőoktatásban*. (2016) in Fehér András, Kiss Virág Ágnes, Dr. Soós Mihály, Dr. Szakály Zoltán (szerk.): Hitelesség és Értékorientáció a Marketingben. Debreceni Egyetem Gazdaságtudományi Kar: Debrecen. ISBN: 978 963 472 8 pp. 276–288.
14. Gal Rimon: *Great Gamification Examples* (2014) https://www.gameffective.com/7-great-gamification-examples/ (utoljára megtekintve: 2018. 10. 19.)
15. Nádori Gergely, Prievara Tibor: *Kis-nagy IKT könyv* (2011) in TanárBlog (http://mek.oszk.hu/15900/15960/15960.pdf) (utoljára megtekintve: 2018. 10. 19.)
16. *Bartle taxonomy of player types*, in Wikipedia https://en.wikipedia.org/wiki/Bartle_taxonomy_of_player_types (utoljára megtekintve: 2018. 10. 19.)
17. Suzanne Holloway: *Gamification in Educations: 4 ways to bing games to your classroom* (2018) in TOP HAT (https://tophat.com/blog/gamification-education-class/)(utoljára megtekintve: 2018. 10. 19.)
18. Fromann Richárd: *Gamification jelentősége és működési mechanizmusa* (2012) in Digitalisidentitas https://digitalisidentitas.blog.hu/2012/06/04/fromann_richard_gamification_jelentosege_es_mukodesi_mechanizmusa (utoljára megtekintve: 2018. 10. 19.)
19. *Legyél a legjobb* (2017), in Boglárka http://blog-larka.blogspot.com/ (utoljára megtekintve: 2018. 10. 19.)

20. *Classcraft in Microsoft Education*
https://www.microsoft.com/hu/education/partners/showpartnersdetails.aspx?id=2033311&i=false&t=0&p=1&ps=24 (utoljára megtekintve: 2018. 10. 19.)
21. Nádori Gergely: *Gamification* (2012) in PIL Akadémia 7
http://tanarblog.hu/attachments/3010_7_gamification.pdf (utoljára megtekintve: 2018. 10. 19.)
22. Nick Pelling: *The (short) prehistory of Gamification* (2011) online:
http://nanodome.wordpress.com/2011/08/09/the-short-prehistory-ofgamification/ (utoljára megtekintve: 2018. 10. 19.)
23. Gelencsér Dóra: *Generációk különbségei : X, Y, Z és alfa az iskolában* (2018) in Tantrend
http://tantrend.hu/hir/generaciok-kulonbsegei-x-y-z-es-alfa-az-iskolaban (utoljára megtekintve: 2018. 10. 19.)
24. Nádori Gergyely és Prievara Tibor: *IKT módszertan: Kézikönyv az info-kommunikációs eszközök tanórai használatához* (2012) http://mek.oszk.hu/15900/15959/15959.pdf (utoljára megtekintve: 2018. 10. 19.)
25. Rigóczki Csaba: *Gamifikáció (játékosítás) és pedagógia.* (2016) in Új Pedagógiai Szemle, 2016/3-4.
(http://folyoiratok.ofi.hu/uj-pedagogiai-szemle/gamifikacio-jatekositas-es-pedagogia) (utoljára megtekintve: 2018. 10. 19.)
26. *Az igazság a Duolingo-ról* (2014) in Angolplusz Magazin
(https://www.angolnyelvtanitas.hu/angolnyelvtanitas-blog/az-igazsag-a-duolingo-rol) (utoljára megtekintve: 2018. 10. 19.)
27. https://www.classcraft.com/(utoljára megtekintve: 2018. 10. 19.)
28. http://blog-larka.blogspot.com/ (utoljára megtekintve: 2018. 10. 19.)

Interdiszciplináris műszaki gyakorlatok az informatikatanár szakon

Makan Gergely[1], Antal Dóra[2], Mingesz Róbert[3], Gingl Zoltán[4], Kopasz Katalin[5], Mellár János[6], Vadai Gergely[7]

{[1]makan,[3]mingesz,[4]gingl,[6]mellar,[7]vadaig}@inf.u-szeged.hu,
[2]antal74dora@gmail.com
Szegedi Tudományegyetem, Műszaki Informatika Tanszék

[4]kopaszka@titan.physx.u-szeged.hu
Szegedi Tudományegyetem, Optikai és Kvantumelektronikai Tanszék

SZTE Gyakorló Gimnázium és Általános Iskola

Absztrakt. A technikai fejlődésnek köszönhetően az eszközök egyre nagyobb része elektronikus, működésükért szoftverek felelősek. Mivel az eszközök a külvilág jeleit érzékelik, és ezek feldolgozásával végeznek beavatkozást, az informatika egyre inkább interdiszciplináris jellegűvé válik. A hétköznapi eszközök technológiája, az autóipar, az Ipar 4.0, az IoT – Internet of Things, dolgok internete – az informatika mellett épít a műszaki, elektronikai, fizikai, biológiai és más területekre is. Ennek megfelelően fontos, hogy a képzésekben is megjelenjen ez az interdiszciplináris jelleg, a hallgatók alapvető ismeretekhez juthassanak kapcsolódó műszaki megoldásokban, aminek később, a tanári pályájuk során is hasznát vehetik. Fontos kiemelni, hogy bár az eszközök fejlődése továbbra is igen gyors, a működési elvek nem változnak, így az utóbbiak oktatása egyre fontosabb, szemben egy-egy adott fejlesztőkörnyezet, szoftver vagy hardver használatának tanításával. A fenti elvek követésével különböző területekhez tartozó gyakorlatokat dolgoztunk ki az informatikatanár-képzésünk támogatására. A laboratóriumi gyakorlatok során létrehozzák az áramköri kapcsolásokat, megírják a szoftvereket és gyakorolják a beágyazott szoftverfejlesztés alapvető módszereit is. Egy hétköznapi, egy iparhoz kötődő és egy orvosi méréshez és jelfeldolgozáshoz tartozó példán keresztül mutatjuk be, hogyan értik meg és tanulják meg a hallgatók a legfontosabb elveket. Az élményszerű tanulás jobban motiválja a hallgatókat, segíti a rendszerszintű megértést, a kreativitás fejlesztését. Tapasztalataink alapján a hallgatók magabiztossága is erősödik a modern műszaki rendszerek használatakor, amit különösen fontosnak tartunk.

Kulcsszavak: műszaki informatika, Arduino, interdiszciplináris oktatás

1. Bevezetés

Az oktatás egyik legfontosabb kérdése az, hogy hogyan képes alkalmazkodni a gyors technikai fejlődéshez. Ehhez tartozik a mai, modern eszközeink használata és ismerete mellett az is, hogy a tanulók jelentősen megváltozott körülmények közt élnek, más módon jutnak ismeretekhez, így a hozzáállásuk is más. Komoly gondot okoz, hogy az iskolai közeg egyre idegenebb a tanulók számára, nem értik, mi szükség van számos tananyagrész megtanulására. Ez a közgondolkodásban is egyre elterjedtebb szemlélet: az iskola olyat oktasson, amire a gyakorlatban szükség van. Szakmai körökben gyakori a vita, hogy szükséges-e ma is univerzálisabb érvényű ismereteket tanítani ahelyett, hogy közvetlenül a mai eszközök alkalmazását tanítanánk. Szükséges-e fejben számolni, a Pitagorasz-tételt ismerni, ráadásul bizonyítani, szükséges-e külön tárgyként fizikát, biológiát és más természettudományos tárgyat oktatni.

Az informatika területén különösen jól látszik a probléma: nem tudjuk megmondani, hogy pár év múlva milyen eszközök, programozási környezetek jelennek meg, mivel fognak dolgozni a végzettek

egy informatikai cégnél. Így tehát a legjobbnak az látszik, ha olyan ismeretekhez jutnak a tanulók, melyek alapján könnyen lesznek képesek alkalmazkodni a változó környezethez. Fontos, hogy megtanuljanak logikusan gondolkodni, hogy értsék a legfontosabb működési elveket, hiszen ezek változnak a legkevésbé. Mivel egyre több területen jelennek meg az informatikai megoldások (oktatás, ipar, gyógyászat, kommunikáció, navigáció, szórakoztatás stb.), igen hasznos egy adott szintű interdiszciplináris tudás, nyitottság is.

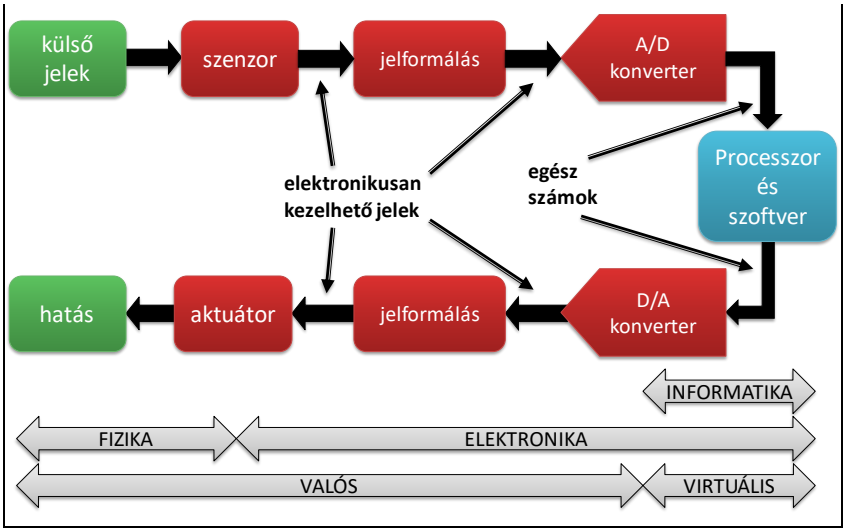
Sokan úgy látják, hogy reménytelen a modern eszközök működési elveinek megértetése, egyre inkább "fekete dobozokként" használjuk eszközeinket: nem látunk bele a működés részleteibe. Ezek az eszközök sok esetben a legújabb eredményeket alkalmazzák, működésükben igen sok szakterülethez kötődnek. Ugyanakkor, ha a megoldások részletei helyett megpróbáljuk a háttérben meghúzódó elveket megtalálni, akkor megfelelő általánosításhoz juthatunk, ami sokak számára érthetővé válik, az eszközök működése sokszor a hétköznapi életben is észlelhető elveken alapul. Jól segíti ezt az is, hogy az oktatás számára is széles körben elérhetővé váltak univerzális építőelemek, melyekkel egészen komoly eszközöket lehet akár játékosan készíteni [1, 2, 3, 4].

Fontos tehát, hogy a tanárszakos hallgatók képzésük során igen alapos tudást szerezzenek és kialakuljon a megfelelő szemléletük. A tanár magas szintű és széleskörű szakmai képzettsége nélkülözhetetlen a mai oktatásban, amit nem pótolnak, csak hatékonyabbá tesznek az oktatási módszerek és megoldások.

Az oktatás fejlesztésével foglalkozó kutatócsoportunk [5] a fentieknek megfelelően alakította ki tevékenységét. Az informatikatanárok képzésében kurzusfejlesztéssel, oktatási anyagok, eszközök és módszerek kidolgozásával veszünk részt [6]. Súlyt helyezünk a gyakorlatias képzésére, a megfelelő elméleti tudás és általános elvek ismeretének átadására, melyek laboratóriumi gyakorlatokon mélyíthetők el.

2. Műszaki módszerek és eszközök az informatikatanár-képzésben

A mai eszközök túlnyomó többsége már elektronikus, processzort tartalmaz, szoftverek vezérlik a működését. Hamarosan lényegében az összes körülöttünk levő eszköz ilyen lesz, a mérettől és az alkalmazási területtől függetlenül. A felépítés és működési elv ugyanakkor viszonylag egyszerű és egységes, ahogy az 1. ábra is mutatja [részletesebb leírás: 6]. A külvilág jelei átalakulnak olyan elektronikával kezelhető jelekké, melyeket számokká konvertálhatunk és így processzorokkal kezelhetünk. A feldolgozás során információt nyerhetünk, ami felhasználható akár beavatkozásra is. Az ilyen felépítésű eszközöket beágyazott rendszereknek is nevezik, a fő működtető egység, azaz a beépített processzor jelenléte miatt [7].



1. ábra: A modern elektronikus eszközök felépítésének blokkvázlata.

Több olyan oktatási eszköz kapható, melyeknél jól látható a fentebb részletezett felépítés, a tanárok és diákok maguk rakhatják össze az egyes univerzális építőelemeket, megtervezhetik a vezérlő algoritmust és elkészíthetik az ezt megvalósító programot.

Fontos kiemelni, hogy az eszközöktől (beágyazott rendszerektől) nagyfokú megbízhatóságot várunk el. Előzéskor, egy légzsák kinyitásakor, betegőrző monitor működése közben nem keletkezhet szoftver- vagy hardverhiba, de egy mobilalkalmazástól is elvárjuk, hogy ne navigáljon szembe egyirányú utcában, megfelelő számlára utaljon, biztonságosan kezelje adatainkat. Sarkos példaként szokták említeni, hogy egy egész szám túlcsordulása okozta egy rakéta katasztrófáját és dollár százmilliók füstté válását [8].

Már tanulókorban fontos tehát az egyfajta igényes, műszaki szemlélet, hozzáállás kialakítása, amit a felsőoktatási képzési és kimeneti követelmények közvetlenül meg is jelölnek. Tanulságos megnézni, hogy milyen előírások érvényesek az informatikatanárok esetén. Ebből pár dolgot emelünk itt ki, melyek közvetlenebbül kötődnek a fentebb említettekhez [9]:

- „*Ösztönzi a tanulók önálló véleményalkotását, törekszik a kritikus gondolkodásmód kialakítására, különös tekintettel az informatikai alkalmazás veszélyeinek figyelemfelhívására.*”
- „*Rendelkezik azokkal az ismeretekkel, amelyek lehetővé teszik, hogy szaktárgyának új eredményeit megismerhesse, értelmezhesse. Ismeri a szaktárgy alapvető kutatási módszertanát.*”
- „*Képes - elsősorban a természettudományokkal és a matematikával - a különböző szakterületek tudás- és ismeretanyaga közötti összefüggések felismerésére, integrációjára.*”
- „*Képes a szaktárgyában elsajátított elméleti ismeretek gyakorlati alkalmazására, ennek közvetítésére a tanulók felé.*”
- „*Tisztában van azzal, hogy a szaktárgyában közvetített tudás, kialakított kompetenciák más műveltségterületen is hatnak.*”
- „*Szakszerűen tudja használni az iskola informatikaoktatási eszközeit, bevonni oktatómunkájába az informatikai eszközöket, távoktatási anyagokat. Alkalmas informatikai tananyagfejlesztésre, más szakos tananyagfejlesztés informatikai megvalósításának támogatására.*”
- „*Együttműködik a szaktárgyával rokon tárgyak tanáraival. Képes arra, hogy a rokon tárgyakban is megjelenő, egymásra épülő ismeretanyagok ütemezését egyeztesse.*”

- „*Elkötelezett az igényes tanári munkára, a folyamatos önművelésre.*"

A szakmai tudás előírása igen bőséges és szerepel benne közvetlenül műszaki terület, robotika is.

2.1. Népszerű műszaki informatikai oktatási eszközök

Egyre jobb, nagyobb tudású és egyre olcsóbb eszközök állnak rendelkezésre, melyekkel jól segíthető egy mai eszköz felépítésének, működési elveinek oktatása játékos, élvezetes, gyakorlatias módon. Ilyenek a Lego robotok [1], az egylapos számítógépek is, a micro:bit [2], az Arduino [3] és a Raspberry Pi [4]. Ezekhez sokféle szenzor köthető, számos különféle jelet lehet velük mérni, melyek adataihoz közvetlenül hozzáférhetnek a tanulók, műveleteket végezhetnek rajtuk, különféle jeleket, hatásokat állíthatnak elő. Ennek megfelelően tehát számottevő átláthatóságok biztosítanak, a szenzorok fontossága és szerepe kiderül anélkül is, hogy az alkalmazott elektronikai kapcsolásokat ismerni kellene. Természetesen sokféle szintje van az ismereteknek, a tanároknak alaposabb műveltségre van szükségük, nem jöhetnek zavarba, ha például egy diák megkérdezi, hogyan képes egy robot érzékelni a közelében levő tárgyakat.

„Az informatika műszaki alkalmazásai" tantárgy oktatása során az alapvető elméleti és gyakorlati ismereteket egy félévben 15 illetve 30 órában tanítjuk informatikatanár-szakos hallgatóknak [10]. A képzés a robotika, a Raspberry Pi mellett jelentős mértékben épít a világszerte is rendkívül népszerű Arduino platformra.

2.2. Arduino alkalmazások – előnyök és hátrányok

Az Arduino egy egylapos számítógépre és programozási környezetre épülő platform. Rendkívül átlátható, gyakorlatilag egy mikrovezérlő (processzort és perifériákat tartalmazó csip) kivezetései vannak kényelmesebben használható csatlakozókra kötve. Sokféle kiegészítő áramkör, szenzor kapható hozzá, így a diákok közvetlenebbül találkoznak elektronikával, kapcsolásokkal. Ezek mellett a programozást is nagyon jó arányérzékkel oldották meg a fejlesztők. Az egyik leggyakrabban alkalmazott programozási nyelv a C++, az Arduino használatához azonban a legtöbb esetben elegendő a szinte minden nyelvben előforduló alapok elsajátítása. A környezet egyszerűségre törekszik, kevés, de jól használható függvényt, építőelemet biztosít a hardver kezeléséhez és a programok felépítése is jól megfelel az operációs rendszer nélkül működő mikrovezérlők általános alkalmazási szokásainak (a main helyett setup és loop függvények). A mikrovezérlő egy fogyasztási, ipari, gyógyászati és más területeken is használt professzionális műszaki komponens, alapvető szerepű a beágyazott rendszerekben, ahol kiemelten fontos a megbízhatóság, a kapcsolódó elvek, szabványok követése is. Természetesen az oktatás során nem lehet ezek maradéktalan követését elvárni, de el kell kerülni a helytelen berögződéseket, a nem kellően tudatos, szakszerűtlen használatot.

Az interneten szinte mindenféle feladathoz található megoldás, letölthető forráskóddal, az összeállítást részletező receptekkel, melyeket szakemberek mellett diákok, hobbisták adnak meg. Sok jó ötlettel lehet találkozni, de az előnyök mellett sajnos komoly hátrányok is mutatkoznak. Egyrészt ez arra ösztönözheti a tanulókat, hogy saját munka helyett inkább reprodukáljanak, másoljanak, sokszor nem is értik, miért kérik őket a tanárok olyan feladatok megoldására, amelyeket könnyen meg lehet találni. Másrészt a nem elég alapos tudás miatt rengeteg látszólag jól működő, műszaki szempontból mégis hibás megoldás terjed. Az egyetlen probléma esetén is rendkívül sok találatból ráadásul igen nehéz lehet megtalálni a megfelelőt még képzettebb tanárok számára is. Sajnos még az Arduino hivatalos oldalán, szakkönyvekben is találhatók helytelen iránymutatások, melyek később nehezen alakítható rossz beidegződéseket okozhatnak.

Fontos feladatunknak tartjuk ezért a működés és elvek minél jobb megértetését, a szakmailag korrekt alkalmazások segítését, az önállóságra és gondosságra nevelést. Nem célunk receptszerű megoldások megadása, az esettanulmányokat példának szánjuk, melyekben a megfelelő módszerek alkalmazását és a helyes hozzáállást szemléltetjük [6]. A laboratóriumi gyakorlatokon élményszerű tanulást jelenthetnek a beágyazott eszköz és szoftverfejlesztési elemek, segíthetik a magabiztosság

kialakulását az egyes problémákkal való szembesülés és azok megoldása útján. A képzési és kimeneti követelményekkel is összhangban fejlődhet az igényes gondolkodásmód és hozzáállás is, aminek főbb elemei közé tartozik a feladat gondos mérlegelése, megfelelő megoldás megadása, az ismert elvek és módszerek helyes, szakszerű alkalmazása, a megoldás elkészítése, tesztelése, javítása is.

3. Arduino gyakorlatok

A laboratóriumi gyakorlatok során többféle területet érintő feladatokat oldanak meg az informatikatanár hallgatók. Ennek megfelelően többféle szenzorral, megoldással, szakterülettel és szemlélettel is találkozhatnak. A munka során tűz- és munkavédelmi ismereteket szereznek, a feladataik elvégzéséről jegyzőkönyvet készítenek, gyakorolják a közös munkát is. Az elkészült programokat és jegyzőkönyveket az oktató értékeli. Az előadás és gyakorlati órák 1:2 aránya jól szolgálja a gyakorlatorientáltságot.

A hallgatók szabadon választhatják meg a megoldást, használhatják ehhez az interneten elérhető segédanyagokat is. Olyan feladatokat kapnak, melyekhez szükség van a személyes problémamegoldó képességeikre is. Néhány feladatra kezdetben egyszerűbb, elterjedtebb megoldást adhatnak, majd ezek után kerülhet sor a szakmailag igényesebb kidolgozásra, melyhez az oktató útmutatást és magyarázatot adhat.

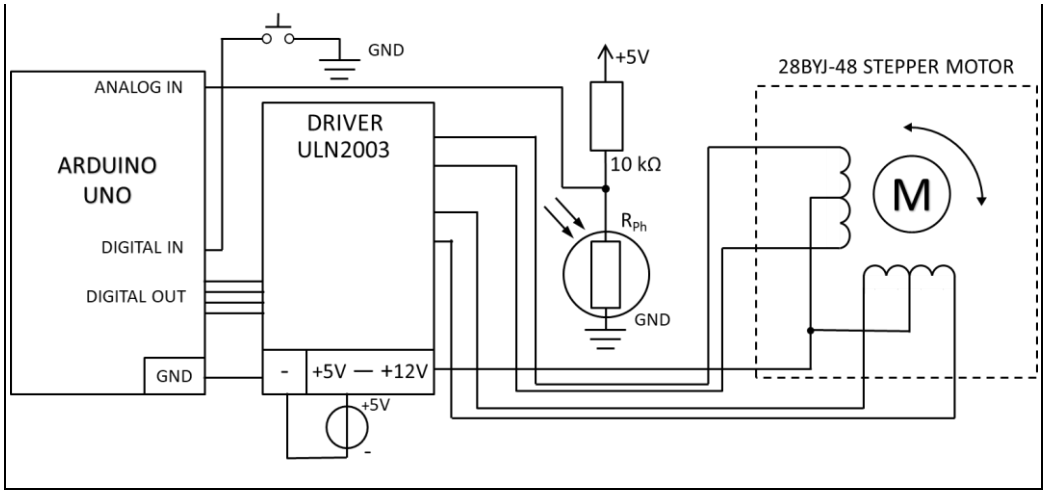
Fontos kiemelni azt is, hogy tapasztalatokat szereznek arról, hogy kell egy laboratóriumi/műszaki környezetben dolgozni, milyen főbb szabályokat, előírásokat kell betartani a megbízhatóság és minőség érdekében, függetlenül akár attól is, hogy ezek céljai számukra nem feltétlen egyértelműek.

A következő példák azt is szemléltetik, hogy a megoldásokhoz alkalmazni kell bizonyos matematikai, fizikai és akár más szakterületi ismereteket is, melyek szükségességére máshogy nem feltétlen gondoltak volna a hallgatók.

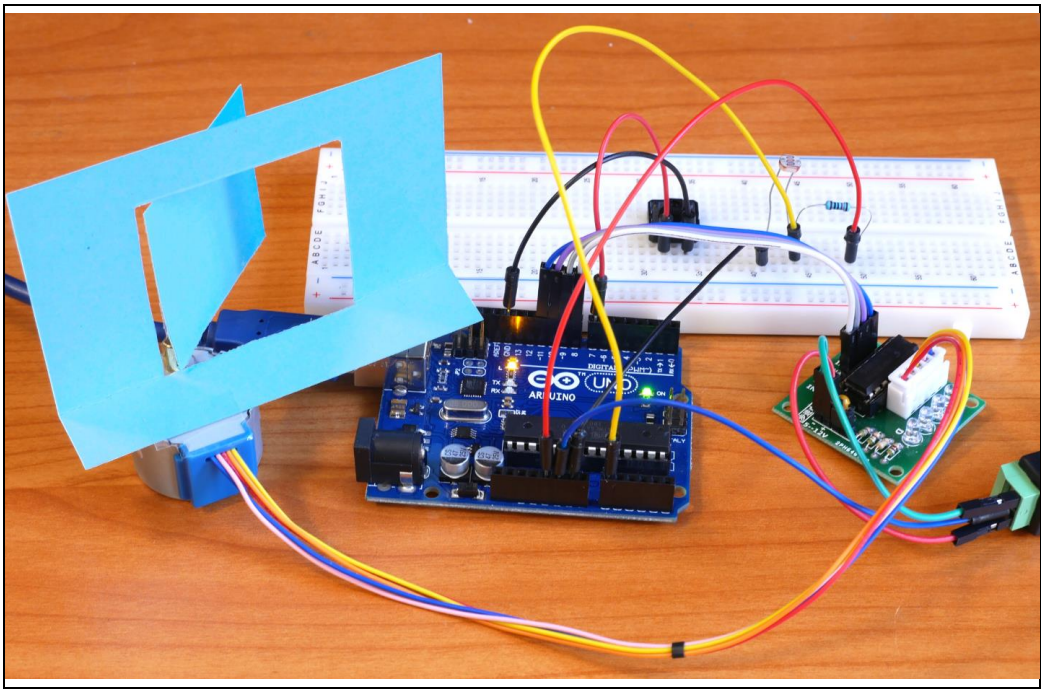
3.1. Kapuvezérlő rendszer készítése

Igen hasznosak az olyan gyakorlatok, melyek egy hétköznapi eszköz működését modellezik. Ez közvetlenül mutatja a gyakorlati hasznosságot és a mögöttes elveket is. Ilyen feladat egy garázskapu-mozgató berendezés makettjének összeállítása és a működtető szoftver megírása.

A munka során egy léptetőmotor mozgatását kell megoldani, amihez a megfelelő algoritmust ki kell dolgozni az elméleti alapismeretek alapján. A motor forgása egy kapu mozgatásának felel meg, a nyitás és zárás 90 fokos elfordulásokat jelent. Jó példa az elvi elvonatkoztatási lehetőségekre is a rendszer, ugyanis a valóságban nem léptetőmotort alkalmaznak ilyen célokra, ugyanakkor ez csak egy részegység közvetlen kezelését jelenti, a főbb elveket nem érinti. A feladat része az is, hogy egy optikai szenzor segítségével vészleállítás történjen, azaz ha a kapu útjába akadály kerül, akkor a mozgás álljon le. Erre a célra egy fotoellenállást kapnak a hallgatók, amit az Arduino áramkörhöz kell illeszteniük és az érzékeléshez szükséges kódrészletet is meg kell írniuk. A kapunyitás egy digitális bementre kötött nyomógomb segítségével indítható.



2. ábra: A kapuvezérlő rendszer felépítése. Az Arduino a meghajtó áramkörrel vezérli a léptetőmotort. A kapu nyitása gombnyomásra indul, valamint a mozgás szünetel, amíg a fényérzékelő akadályt észlel.



3. ábra: A képen az összeállított kapuvezérlő rendszer látható. A kapu egyszerű makettje kartonlapból van kialakítva.

A feladat változatos lehetőségeket ad az egyes részegységek működtetésének megoldására, számos elv alkalmazására. A motor vezérlésében használható fél vagy egész lépés, szabályozhatóvá tehető a lépési sebesség, detektálhatók a végállások. Fotoellenállás helyett használható fototranzisztor, látható helyett infravörös fény. A nyitás vezérelhető távirányítóval is, a működési logika is sokféleképp variálható. Ki lehet egészíteni a feladatot a motorok sebességének változtatásával a végpontok körül

vagy egy további LED-el, ami villog, ha mozgásban van a kapu. A hallgatók az adott számú léptetést tipikusan for ciklussal oldják meg, mely helyett használható maga a loop függvény is megfelelő feltételvizsgálatokkal. A lépések időzítését meg lehet oldani precízebben is a timer periféria használatával. Ezek nagy szabadságot adnak a nehézségi szint megválasztásában, a különböző tempóban haladó hallgatók megfelelő terhelésében is.

Tapasztalataink alapján a feladat megoldása során a hallgatóknak leginkább a következők jelentettek nehézséget: a léptetőmotor áttételének megértése és a mozgatás szüneteltetése arra az időre, amíg nincs elindítva a nyitás, és amikor akadály került az érzékelő elé. A léptetőmotor működésében a mechanikai felépítés és mozgatási elv hathatott újdonságként. A mikrovezérlő programozása operációs rendszer nélküli környezetben történik, ami szintén szokatlan a hallgatók számára. A léptetőmotor működésének megértésében több előzetes segédanyag, szimuláció lehet segítség [11, 12], míg a mozgatás megállításának ésszerű megoldásához a mikrovezérlőprogramozásban alkalmazott végtelen ciklus (az Arduino környezetben a loop függvény) szükségességének alaposabb elmagyarázása illetve több példa segíthet.

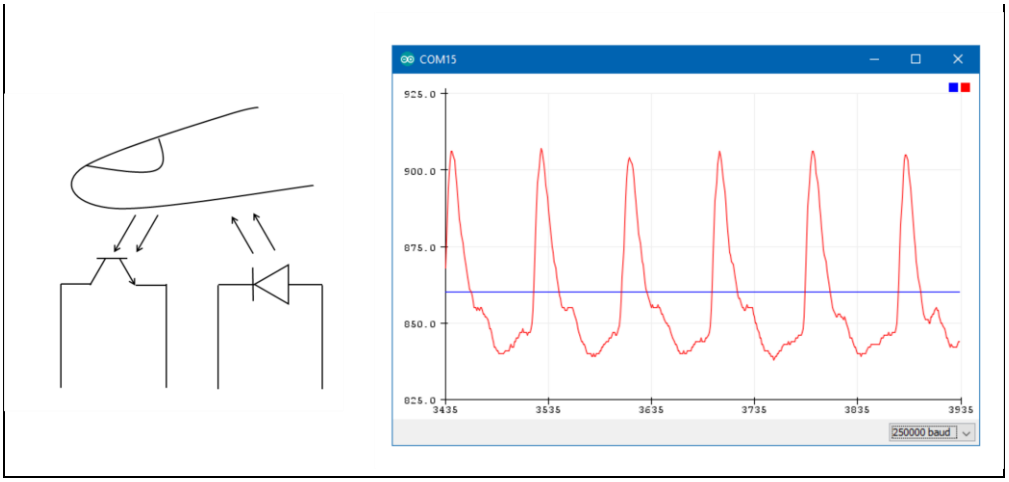
3.2 Fotopletizmográf alapú szívritmusmérés

Ma már okostelefonok és okosórák is képesek szívritmusmérésre, melynek egyik egyszerű alapelve az úgynevezett fotopletizmográfia. Egy izgalmas, élményszerű gyakorlat építhető fel ennek megvalósítására.

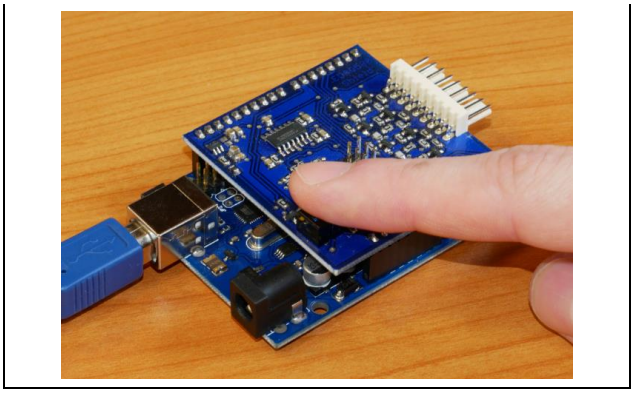
A mérési elv egyszerű: az ujjba infravörös fényt bocsátunk, majd mérjük a visszavert vagy átjutó fényt, aminek erőssége függ az ujj pillanatnyi vértelítettségétől [13]. Ennek mérésével lényegében a vérnyomással arányos időfüggő jelet kaphatunk, amit felhasználhatunk a szívműködés megfigyelésére is. A jel változó komponense igen kicsi, amely egy sokkal nagyobb átlagértékhez adódik [14]. A hallgatók megismerik, hogyan lehet a váltókomponenst megfelelő tartományba erősíteni, mely így közvetlenül az Arduino áramkör analóg bemenetére köthető és digitalizálható. A megoldási elv igen általános, sok más (pl. mikrofon) jel kezelésében is elterjedten alkalmazzák.

A hallgatók a feladat során az általunk fejlesztett EDAQ530-as szenzorinterfész [15] Arduino alapú megvalósítását használják. Az EDAQuino nevű Arduino „shielden" [6, 16, 17] többek között található egy infra fényű fotodetektor a pletizmográfiához szükséges előerősítéssel és szűréssel. Az áramkör fotopletizmogáf részét akár próbapanelen össze is lehet állítani, aminek során fontos áramkörépítési alapismeretekre tehetnek szert a hallgatók. Ezt követően mintavételezéses mérést kell végezni, ami igen egyszerűen is megoldható, de gondosabb megoldások esetén számos megfontolást igényel. Az Arduino integrált környezet valós idejű grafikus megjelenítést biztosít az úgynevezett „serial plotter" segítségével, így a hallgatóknak hamar lehet komoly sikerélményük a saját szívműködésük grafikus szemléltetésével.

Valós idejű jelfeldolgozással lehetséges egy LED-et a szívműködésnek megfelelő ritmusban kapcsolni, a megfelelő detektáló algoritmus megvalósításával az egyes szívdobbanások közti időtartamot meg lehet mérni és meg lehet jeleníteni. A feladat jelentősen bővíthető az orvosi gyakorlatban is használatos szívritmus-variabilitás (heart rate variability, HRV) mutatók kiszámításával. Ez tulajdonképpen az egymást követő szívütések között eltelt idő ingadozásának statisztikai analízisét jelenti. Ezek közé tartozik a szívütések közti idő (RR) átlagának (Mean RR), szórásának (SD RR), valamint a pNN50 nevű indikátornak a meghatározása, ami százalékosan azt adja meg, hogy az előző RR értéktől hány szívütés tér el 50 ms-nál jobban [18, 19]. Továbbá az RR értékekből hisztogram is számolható.



4. ábra: Az ábra bal oldalán a fotopletizmográf működési elve látható, a jobb oldalán pedig a fototranzisztor által mért jel a „serial plotter”-ben megjelenítve.



5. ábra: A pulzus mérésére az általunk fejlesztett EDAQuino nevű Arduino shieldet használják a hallgatók.

A tapasztalataink alapján a feladatnál a szintmetszés-detektálás algoritmusának megtalálása okozott kisebbfajta nehézséget a hallgatóknak, amit indokol, hogy digitális jelfeldolgozással még nem találkoztak. Ez rávilágít arra, hogy milyen fajta jelfeldolgozási alapokat érdemes a tananyag részévé tenni. Felmerül annak a lehetősége is, hogy a programozási alapkurzusokon olyan jellegű gyakorló feladatokat kapjanak a hallgatók, melyek hasonló algoritmusokkal oldhatók meg.

3.3 Hőmérsékletszabályzás

Az egyik legelterjedtebb és legegyszerűbb szabályozási elv az úgynevezett on-off szabályzás, mely számos hétköznapi eszköz működésében is kulcsszerepű. A szabályozások általános módszere szerint mérni kell a kívánt értéktől való eltérést, és ennek megfelelően növelő vagy csökkentő hatást kell gyakorolni. Az on-off szabályzás esetén a hatás nagysága előre adott, nem függ a kívánt értéktől való eltérés nagyságától.

A szabályzás megvalósításához egy teljesítményellenállást használnak a hallgatók, melyre feszültséget kapcsolva fűteni lehet. A fűtőteljesítmény a következőképpen függ a fűtőfeszültségtől:

$$\boldsymbol{P = \frac{U^2}{R}}, \tag{1}$$

ahol *U* a fűtőfeszültség, *R* a fűtőellenállás étéke.

Ez jelentős áramot igényelhet, amit az Arduino digitális kimeneteivel vezérelt meghajtó áramkörrel vagy egyszerűen az egyik digitális kimenetére megfelelően kötött tranzisztorral biztosíthatunk. A hűtést egy kisméretű ventilátorral lehet megoldani. Az ellenállás aktuális hőmérsékletét a hallgatók egy termisztorral [20] (hőmérsékletfüggő ellenállással) mérik meg. A hőmérsékletet a termisztor képlete segítségével számíthatjuk ki:

$$\boldsymbol{T = \frac{1}{\frac{1}{T_{25}} + \frac{1}{B} \cdot \ln\left(\frac{R}{R_{25}}\right)}}, \tag{2}$$

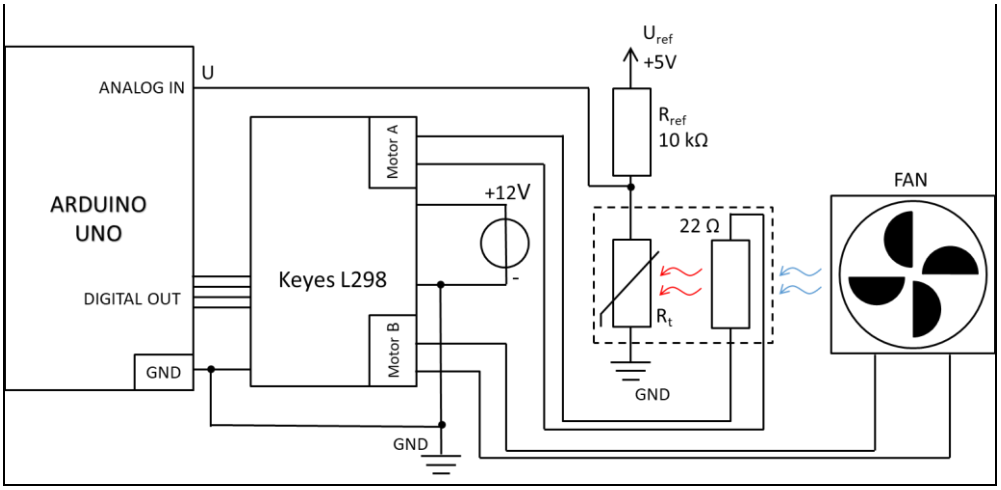
ahol *T* a hőmérséklet kelvinben, T_{25} a szobahőmérséklet szintén Kelvinben (25 °C = 298,15 K), a *B* az adott termisztorra jellemző konstans, R_{25} pedig a termisztor ellenállása 25 °C-on. Mivel az analóg-digitál konverterrel csak feszültséget lehet mérni, ezért szükség van egy olyan kapcsolásra, ami az ellenállásmérést feszültségmérésre vezeti vissza. Ezt legegyszerűbben egy feszültségosztóval tehetjük meg. A feszültségosztó bemenete az A/D konverter referenciafeszültsége (U_{ref}), a kimeneti feszültséget (*U*) mérve kifejezhetjük a termisztor ellenállását (R_t):

$$\boldsymbol{R_t = R_{ref} \frac{U}{U_{ref} - U}} \tag{3}$$

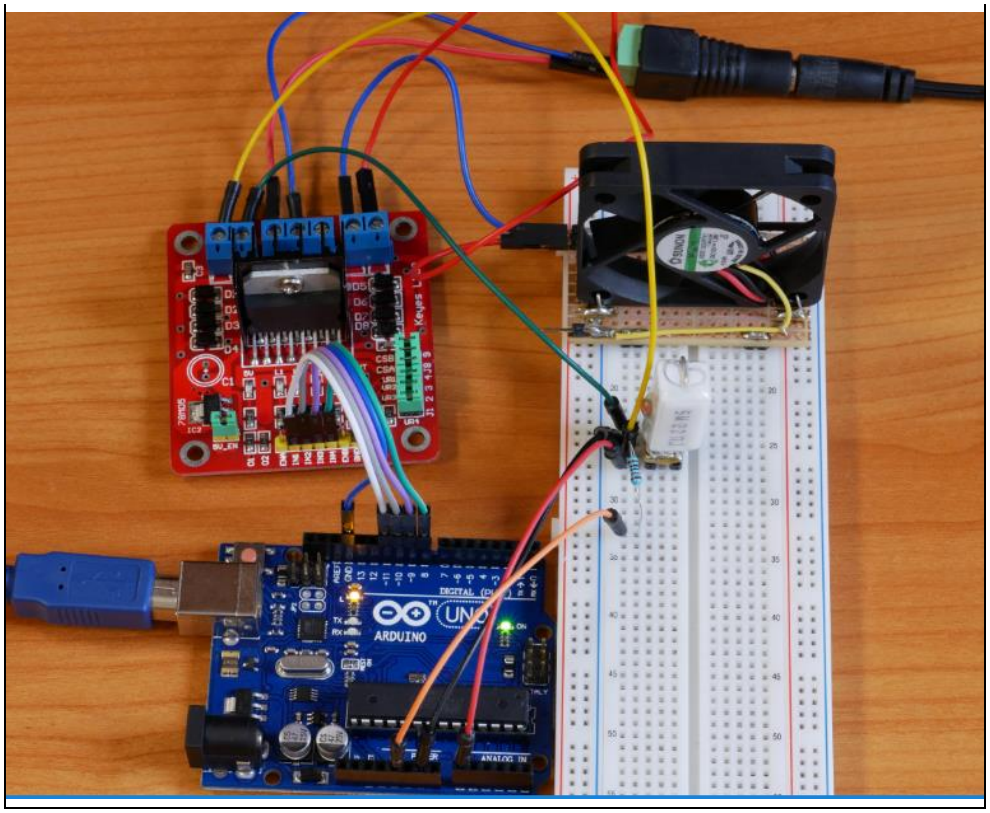
A feladat egy olyan program megírása, ami az ellenállást egy adott hőmérsékletre fűti fel, majd ezen az értéken tartja. A ki- és bekapcsolási küszöbszinteket (hiszterézist) be kell állítani, ezek hatását a hőmérséklet valós idejű grafikus megjelenítésével (serial plotter) lehet megfigyelhetővé tenni.

Pusztán a program módosításával pulzusszélesség-modulációs (PWM) módban, azaz változtatható erősséggel is vezérelhető a fűtés és hűtés, így általános elvet láthatnak a hallgatók arra, hogy kétállapotú jelekkel hogyan lehet sokféle szintet létrehozni.

A kísérlet könnyen átvihető más területekre is, alkalmazható például egy kondenzátor feszültségének adott szinten tartására, melyet egy ellenálláson keresztül lehet tölteni és kisütni. Ehhez elegendő az Arduino kimeneti árama is, így az ellenálláson és kondenzátoron kívül nincs szüksége semmilyen más alkatrészre. A pillanatnyi feszültség A/D konverterrel mérhető.

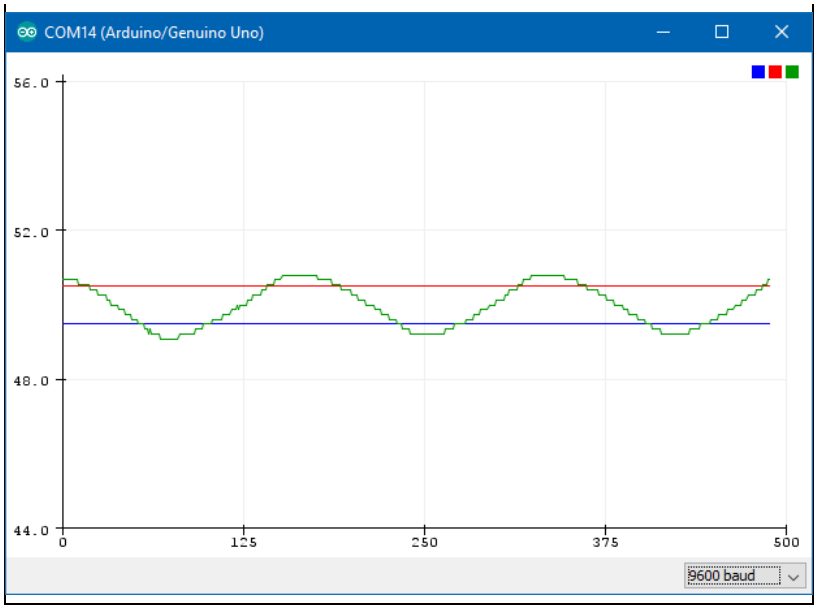


7. ábra: A hőmérsékletszabályzási feladat diagrammja. Az Arduino egy meghajtó áramkör segítségével tudja ki-, bekapcsolni a fűtést és a hűtést is.



8. ábra: A képen a próbapanelen összeállított hőmérsékletszabályzás látható. Az Arduino csak a meghajtó áramkör segítségével tud elegendő áramot kiadni a fűtés és a hűtés számára.

Ennél a feladatnál azt tapasztaltuk, hogy a hiszterézis (a két küszöbszint) megvalósítása okozott nehézséget, illetve nehezen volt érthető számukra, hogy a hőmérséklet miért nem marad a küszöbszinteken belül (9. ábra). Ennek jobb megértését akár hétköznapi példák említésével is lehet segíteni.



9. ábra: A szabályzott hőmérséklet időfüggése. A fűtés be- és kikapcsolási küszöbszintjeit (49,5 °C és 50,5 °C) kék és piros vonalak jelzik.

A hőmérséklet képletének kódolásánál gyakran futottak bele a hallgatók egész osztási problémákba. Nem mindig vették figyelembe, hogy C és C++ nyelvekben két egész szám osztásakor az eredmény is egész és ez igen nagy hibát okozhat, könnyen lehet az osztás eredménye nulla is. A mikrovezérlőprogramozás során gyakran előforduló hibákra, a fontosabb programozási elvek ismertetésére a bevezető órákon érdemes kitérni.

Mivel a hallgatóink az egyetemi tanulmányaik során most először találkoztak az elektronika alkalmazásával, ezért az egyszerű kapcsolások összeállítását és a próbapanel használatát is meg kellett ismerniük.

4. Összefoglalás

Az informatikatanárok képzésének fontos része az egyre elterjedtebb modern, processzorokat tartalmazó, szoftvereket futtató eszközök működési elveinek alapvető megértése. Az informatikai, műszaki megoldások egyre nagyobb szerephez jutnak más területeken is, így az interdiszciplináris jelleg is erősödik. Az eszközök, szoftverek igen gyors fejlődése miatt alapvető az univerzálisabb elvek megismerése, alkotó alkalmazása, melyet az elméleti oktatás mellett laboratóriumi gyakorlatokon lehet a leghatékonyabban elmélyíteni. A hallgatók az élményszerűbb gyakorlatias tanulás révén magabiztosabbá, igényesebbé válhatnak, számos szakterület módszereit ismerhetik meg, jelentősen fejlődik a problémamegoldó képességük is.

Bemutattunk három, Arduino platformmal elvégezhető laboratóriumi gyakorlatot, melyet a tanárszakos képzésünkben használtunk. A tapasztalatok alapján a hallgatók sikeresen el tudják végezni a feladatokat és élvezetesnek találják a munkát. A feladatok igen sokrétűen bővíthetők, alkalmazhatók középiskolai és egyetemi környezetben, különböző nehézségi szinteken. Az Arduino áramkörök

és kiegészítőik olcsón hozzáférhetők, így tantermek kis ráfordítással felszerelhetők, akár minden diák számára juthat külön készlet. A diákokok otthoni használatra is beszerezhetik az eszközöket, ami a gyakorlási lehetőség mellett segíti saját ötleteik megvalósítását is.

Köszönetnyilvánítás

A tanulmány elkészítését a Magyar Tudományos Akadémia Tantárgypedagógiai Kutatási Programja támogatta.

Irodalom

1. *Lego robotok információs oldalai*
https://www.lego.com/en-us/mindstorms , (utoljára megtekintve: 2018.10.24.)
2. *Micro:bit információs oldalak*
https://microbit.org/hu/ (utoljára megtekintve: 2018.10.24.)
3. *Arduino információs oldalak*
https://www.arduino.cc/, (utoljára megtekintve: 2018.10.24.)
4. *Raspberry Pi információs oldalak*
https://www.raspberrypi.org/ , (utoljára megtekintve: 2018.10.24.)
5. *Az MTA-SZTE Műszaki Informatika Szakmódszertani Kutatócsoport honlapja*
http://www.inf.u-szeged.hu/miszak/ (utoljára megtekintve: 2018. 10. 24.)
6. Gingl Zoltán, Kopasz Katalin, Makan Gergely, Mingesz Róbert, Mellár János, Szépe Tamás, Vadai Gergely, *Műszaki informatikai megoldások a modern középiskolai oktatásban,* INFODIDACT 2017 konferencia, Zamárdi, 2017. november 23-25. (2017)
7. *Beágyazott rendszer*
https://en.wikipedia.org/wiki/Embedded_system
8. *Rakétakatasztrófa*
https://hownot2code.com/2016/09/02/a-space-error-370-million-for-an-integer-overflow/
9. *KKK Informatika műveltségi terület*
https://net.jogtar.hu/jogszabaly?docid=a1300008.emm
10. *Az informatika műszaki alkalmazásai" tantárgy oktatása*
http://www.noise.physx.u-szeged.hu/Education/IMA/
11. *Mechatronika jegyzet*
https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011_0104_SZTE-2_Mechatronika/adatok.html, (utoljára megtekintve: 2018.10.28.)
12. *Léptetőmotor*
https://en.wikipedia.org/wiki/Stepper_motor, (utoljára megtekintve: 2018.10.28.)
13. Nagy, Tamás, and Zoltán Gingl., *Low-cost photoplethysmograph solutions using the Raspberry Pi*, Computational Intelligence and Informatics, CINTI 2013, IEEE 14th International Symposium on. IEEE Budapest, Hungary (2013), pp. 163-167.
14. Zoltán Gingl, *A photoplethysmograph experiment for microcontroller labs*, INTERNATIONAL JOURNAL OF ELECTRICAL ENGINEERING EDUCATION 49:(1) pp. 42-60. (2012)
15. Katalin Kopasz, Péter Makra and Zoltán Gingl, *Edaq530: a transparent, open-end and open-source measurement solution in natural science education*, Eur. J. Phys., (2011), **32** 491
16. Zoltán Gingl, János Mellár, Tamás Szépe, Gergely Makan, Róbert Mingesz, Gergely Vadai, Katalin Kopasz, *Universal Arduino-based experimenting system to support teaching of natural sciences* pp. 1-2. Spanyolország Extended abstract, GIREP-MPTL 2018 - Research and Innovation in Physics education:two sides of the same coin. 9th-13th July 2018, Donostia-San Sebastian, Spain (2018)

17. *EDAQuino*
http://www.inf.u-szeged.hu/miszak/projektjeink/edaquino, (utoljára megtekintve: 2018.10.28.)
18. *McKinley P S, Shapiro P A, Bagiella E, Myers M M, Meersman R E D, Grant I and Sloan R P 2003 Deriving heart period variability from blood pressure waveforms Journal of Applied Physiology 95 1431–8*
19. *Acharya U R, Joseph K P, Kannathal N, Lim C M and Suri J S 2006 Heart rate variability: a review Med Bio Eng Comput 44 1031–51*
20. *Termisztor*
https://en.wikipedia.org/wiki/Thermistor, (utoljára megtekintve: 2018.10.28.)

Backtrack-es feladat-variációk

Menyhárt László[1], Zsakó László[2]

{[1]menyhart, [2]zsako }@caesar.elte.hu
ELTE IK

Absztrakt. Ezt a cikket olyan informatika tanároknak írtuk, akik az óráikon, házifeladatnak, esetleg versenyeken backtrack-es feladatokat szeretnének kiadni. Bemutatunk pár módszert, amivel egy egyszerűbb feladatból bonyolultabb probléma generálható, melynek így komplexebb, ötletet igénylő megoldása lesz. Egy példából kiindulva egyre nehezedő feladat-variációkon vezetjük végig az olvasót.

Kulcsszavak: oktatás, feladat, variáció, backtrack

Bevezetés

Jelen cikkünk olyan informatika tanároknak szól, akik különböző nehézségű, visszalépéses kereséssel megoldható feladatokat szeretnének kiadni megoldandó feladatként óráikon, házifeladatnak, vagy versenyeken. Szükség lehet ilyen átalakításokra, ha nem szeretnénk ugyanazt a feladatot megoldatni különböző képességű csoportokkal, egy adott évben különböző osztályokkal vagy ha nem szeretnénk félévente-évente ismét számonkérni ugyanazt. Versenyeken a korosztályok különböző nehézségi szintű feladatai így is eltérhetnek.

Egy konkrét példát és annak variációit elemezzük végig, majd ezekből általánosan használható módszereket fogalmazunk meg.

A visszalépéses keresés (backtrack) a problémamegoldás igen széles területén alkalmazható algoritmus, amelynek lényege a feladat megoldásának megközelítése rendszeres próbálgatással. Néha ez a legjobb megoldás! Ennek ellenére az algoritmusokról szóló könyvek jelentős része nem foglalkozik a visszalépéses kereséssel. Néhányban található egy-egy példa backtrack-re a nyolc vezér, fa bejárás, játékok, logikai formulák kiértékelése témakörökben [1][2]. Egyetemi tananyagokban már több helyen szerepel a 8 vezér, térképszínezés, solitaire, sudoku témakörökkel [3][4][5][6]. Az ELTE Informatikai Karán a Mesterséges intelligencia tárgyban találkoznak vele a hallgatók [7]. Gyakoroltatáshoz kiadható rokon feladatokat azonban, mint amilyen feladat-sorozatot ebben a cikkben bemutatunk, egyik sem tartalmaz.

A visszalépéses keresés egy olyan általános módszer, mely az összes lehetséges eset kipróbálása (brute force) helyett egy ötlet felhasználásával nagyban csökkenti a lépésszámot. Megfelelő adatábrázolásra és ügyes feltételek megfogalmazására van szükség.

Adott N sorozat, amelyek rendre M[1], M[2], ... M[N] elemszámúak, de előfordulhat, hogy azonos elemszámúak (M). Ki kell választani mindegyikből egy-egy elemet úgy, hogy az egyes sorozatokból való választások másokat befolyásolnak. Ez egy bonyolult keresési feladat, amelyben egy adott tulajdonsággal rendelkező szám N-est kell megadni úgy, hogy ne kelljen az összes lehetőséget végignézni.

E feladatok közös jellemzője, hogy eredményük egy sorozat. E sorozat minden egyes tagját valamilyen sorozatból kell kikeresni, de az egyes keresések összefüggenek egymással (például a vezért nem lehet oda tenni, ahol egy korábban letett vezér ütné; egy munkát nem lehet két munkásnak adni; ha egy pékségnek elfogyott a kenyere, akkor attól már nem lehet rendelni).

- A visszalépéses keresés olyan esetekben használható, amikor a keresési tér fastruktúraként képzelhető el, amiben a gyökérből kiindulva egy csúcsot keresünk.

- Az algoritmus lényege, hogy a kezdőpontból kiindulva megtesz egy utat a feladatot részproblémákra bontva, és ha valahol az derül ki, hogy már nem juthat el a célig, akkor visszalép egy korábbi döntési ponthoz, és ott más utat – más részproblémát választ.

Először megpróbálunk az első sorozatból kiválasztani egy elemet, ezután a következőből, s ezt addig csináljuk, amíg választás lehetséges. Y[i] jelölje az i. sorozatból kiválasztott elem sorszámát! Ha még nem választottuk, akkor értéke 0 lesz. Ha nincs jó választás, akkor visszalépünk az előző sorozathoz, s megpróbálunk abból egy másik elemet választani. Visszalépésnél természetesen törölni kell a választást abból a sorozatból, amelyikből visszalépünk. Az eljárás akkor ér véget, ha minden sorozatból sikerült választani, vagy pedig a visszalépések sokasága után már az első sorozatból sem lehet újabb elemet választani (ekkor a feladatnak nincs megoldása).

```
Keresés(N,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i³1 és i≤N  {lehet még és nincs még kész}
    Jóesetkeresés(i,Van,j)
    Ha Van akkor Y[i]:=j; i:=i+1  {előrelépés}
        különben Y[i]:=0; i:=i-1  {visszalépés}
  Ciklus vége
  Van:=(i>N)
Eljárás vége.
```

Egy lineáris keresés kezdődik az i. sorozatban:

```
Jóesetkeresés(i,Van,j):
  j:=Y[i]+1
  Ciklus amíg j≤M[i] és (rossz(i,j) vagy tilos(j))
    j:=j+1
  Ciklus vége
  Van:=(j≤M[i])
Eljárás vége.
```

Megjegyzés: az i-edik lépésben a j-edik döntési út nem választható, ha az előzőek miatt rossz, vagy ha önmagában rossz.

Megállapíthatjuk tehát, hogy minden egyes új választás az összes korábbitól függhet (ezt formalizálja az rossz függvény), a későbbiektől azonban nem! Egyes esetekben nemcsak a korábbiaktól, hanem saját jellemzőjétől is függhet a választás (ezt írja le az tilos függvény).

```
rossz(i,j):                {1. változat}
  k:=1
  Ciklus amíg k<i és szabad(i,j,k,Y[k])
    k:=k+1
  Ciklus vége
  rossz:=(k<i)
Függvény vége.
```

Amikor az összes lehetséges megoldást végig kell nézni, akkor a visszalépést és a keresés folytatását rekurzív függvényhívással a legegyszerűbb megoldani. Itt viszont a sok függvényhívás miatt a memóriahasználat megnő, nagymennyiségű adat esetén problémás lehet.

```
Összes_megoldás(i,N,Db,Y,x):
  Ha i>N akkor Db:=Db+1; Y(Db):=x
  különben Ciklus j=1-től N-ig
             Ha nem rossz(i,j) és nem tilos(j)
                akkor x[i]:=j
                      Összes_megoldás(i+1,N,Db,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

Feladat-variációk

Az alapfeladat

Egy vállalkozás `N` különböző állásra keres munkásokat (`M`). A jelentkezők *bizonyos* információkat osztanak meg a jelentkezéskor. A feladat annak a meghatározása, hogy ki melyik állást töltse be.

1. variáció

Az `N` állásra pontosan `N` jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért, illetve amihez ért, arra mennyi fizetést kérne.

N állás, N jelentkező, összegeket adtak meg, mindenkit fel lehet venni, legolcsóbb kell.

Legyen `F[i,j]` értéke 0, ha az `i.` jelentkező a `j.` munkához nem ért, illetve `F[i,j]=A>0`, ha ért hozzá és `A` forintot szeretne kapni érte.

A vállalkozás vezetője azt szeretné, ha az összes állást betöltenék, de úgy, hogy számára ez a lehető legkisebb költséggel járna.

A következőt gondolta ki: első lépésként állítsuk elő az összes lehetséges állásbetöltést (ha van egyáltalán olyan).

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	100	0	0	100	0
2. jelentkező:	0	200	0	0	0
3. jelentkező:	200	100	0	0	0
4. jelentkező:	0	0	200	0	400
5. jelentkező:	500	0	400	0	200

Ez azt jelenti, hogy minden jelentkezőhöz hozzárendelünk egy állás sorszámot két feltétellel:

- olyan állást választhatunk számára, amihez ért (`F(i,j)>0`);
- olyan állást választhatunk számára, amit még nem adtunk másnak (`Volt(i,j,x)` függvény értéke hamis).

```
Megoldás(N,F,Db,Y):
  Összes_állás(1,N,F,Db,Y,x)
Eljárás vége.
```

Az `Összes_állás` eljárás első paramétere az aktuálisan vizsgált jelentkező `i` sorszáma legyen, a második paramétere pedig a jelentkezők (és egyben állások) `N` száma!

A megoldásban használt fontos változók:

- `x` – az aktuálisan számolt megoldás: `x[i]` az `i.` jelentkezőnek adott munka sorszáma

- `Db` – a megoldások száma;
- `Y` – a megoldásokat tartalmazó vektor.

```
Összes_állás(i,N,F,Db,Y,x):
  Ha i>N akkor Db:=Db+1; Y[Db]:=x
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
                akkor x[i]:=j; Összes_állás(i+1,N,F,Db,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.

Volt(i,j,x):
  k:=1
  Ciklus amíg k<i és x[k]≠j
    k:=k+1
  Ciklus vége
  Volt:=(k<i)
Függvény vége.
```

Ezután nincs más hátra, mint az `Y` vektorban összegyűlt megoldásokból kiválasztani a vállalkozó számára leggazdaságosabbat. Nagyon hamar kiderülhet azonban, hogy túlságosan sok elem lehet az `Y` vektorban.

A megoldási ötlet: felesleges az összes lehetséges megoldást tárolni, elég csupán minden lépés után a leggazdaságosabbat.

A megoldásban használt fontos változók:

- `x` – az aktuálisan számolt megoldás: `X[i]` az `i`. jelentkezőnek adott munka sorszáma
- `Y` – a legjobb megoldás.

```
Legjobb_állás(i,N,F,Y,x):
  Ha i>N akkor Ha Költség(N,F,x)<Költség(N,F,Y) akkor Y:=x
                Elágazás vége
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
                akkor x[i]:=j; Legjobb_állás(i+1,N,F,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.

Költség(N,F,x):
  s:=0
  Ciklus i=1-től N-ig
    s:=s+F[i,x[i]]
  Ciklus vége
  Költség:=s
Függvény vége.
```

Itt egy kicsi probléma léphet fel: az első megoldást mivel hasonlítjuk? Vegyünk fel egy új változót, ami az eddigi legjobb megoldás költségét tartalmazza! Állítsuk ennek az értékét a program elején a legnagyobb egész számra! Ha egy megoldást találunk, akkor az ennél biztosan jobb lesz, azaz lecserélhetjük rá.

```
Megoldás(N,F,Maxkölt,Y):
  Maxkölt:=+∞
  Legjobb_állás(1,N,F,Maxkölt,Y,x)
Eljárás vége.
```

A megoldásban használt fontos változók:

- `x` – az aktuálisan számolt megoldás: `x[i]` az `i`. jelentkezőnek adott munka sorszáma
- `Maxkölt` – az eddigi legjobb megoldás költsége;
- `Y` – a legjobb megoldás.

```
Legjobb_állás(i,N,F,Maxkölt,Y,x):
  Ha i>N akkor Ha Költség(N,F,x)<Maxkölt
                  akkor Y:=x; Maxkölt:=Költség(N,F,x)
               Elágazás vége
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
             akkor x[i]:=j; Legjobb_állás(i+1,N,F,Maxkölt,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

Előfordulhat természetesen, hogy a feladatnak egyáltalán nincs megoldása, azaz legjobb megoldás sincs:

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	0	0	100	100	0
2. jelentkező:	0	200	0	0	100
3. jelentkező:	200	100	0	0	0
4. jelentkező:	0	0	200	400	0
5. jelentkező:	0	0	400	200	0

Már csak egy apróságra gondolhatunk: ha van egy megoldásunk és a most készülő megoldásról látszik, hogy már biztosan rosszabb lesz – többe fog kerülni –, akkor azt már nem érdemes tovább vinni.

Legyen az eljárás paramétere az eddigi költség, s az eljárást csak akkor folytassuk, ha még nem érjük el a korábban kiszámolt maximális költséget. Emiatt nem a megoldások elkészültekor kell számolni költséget, hanem menet közben, folyamatosan.

A megoldást is módosítanunk kell, a `Legjobb_állás` eljárásnak új paramétere lesz, az aktuális előtti választások `költ` költsége.

A megoldásban használt fontos változók:

- `x` – az aktuálisan számolt megoldás: `x[i]` az `i`. jelentkezőnek adott munka sorszáma
- `költ` – az aktuálisan számolt megoldás költsége;
- `Maxkölt` – az eddigi legjobb megoldás költsége;
- `Y` – a legjobb megoldás.

```
Megoldás(N,F,Maxkölt,Y):
  Maxkölt:=+∞
  Legjobb állás(1,0,N,F,Maxkölt,Y,x)
Eljárás vége.

Legjobb_állás(i,költ,N,F,Maxkölt,Y,x):
  Ha i>N akkor Ha költ<Maxkölt akkor Y:=x; Maxkölt:=költ
               Elágazás vége
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
                és költ+F[i,j]<Maxkölt
                akkor x[i]:=j
                      Legjobb_állás(i+1,költ+F[i,j],
                                    N,F,Maxkölt,Y,x)
    Ciklus vége
  Elágazás vége
Eljárás vége.
```

2. variáció

Egy vállalkozás N különböző állásra keres munkásokat. A hirdetésre M jelentkező érkezett (M<N), ahol minden jelentkező megmondta, hogy mely munkákhoz ért, illetve amihez ért, arra mennyi fizetést kérne.

N állás, M(<N) jelentkező, összeget mond, mindenkit fel lehet venni, legolcsóbb kell.

Legyen `F(i,j)` értéke 0, ha az `i`. jelentkező a `j`. munkához nem ért, illetve `F(i,j)=A>0`, ha ért hozzá és `A` forintot szeretne kapni érte.

A vállalkozás vezetője azt szeretné, ha az összes jelentkezőt fel tudná venni, de úgy, hogy számára ez a lehető legkisebb költséggel járna.

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	100	0	0	100	0
2. jelentkező:	0	200	0	0	0
3. jelentkező:	200	100	0	0	0
4. jelentkező:	0	0	200	0	400

A megoldás nagyon hasonló az előzőhöz: itt nem akkor van kész egy megoldás, ha N jelentkezőnek adtunk munkát, hanem akkor, ha az M jelentkezőnek adtunk munkát:

```
Megoldás(N,M,F,Maxkölt,Y):
  Maxkölt:=+∞
  Legjobb állás(1,0,N,M,F,Maxkölt,Y,x)
Eljárás vége.
```

```
Legjobb_állás(i,költ,N,M,F,Maxköltt,Y,x):
  Ha i>M akkor Ha költ<Maxköltt akkor Y:=x; Maxköltt:=költ
               Elágazás vége
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
                és költ+F[i,j]<Maxköltt
                akkor x[i]:=j
                      Legjobb_állás(i+1,költ+F[i,j],
                                    N,M,F,Maxköltt,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

3. variáció

Egy vállalkozás N különböző állásra keres munkásokat. A hirdetésre M jelentkező érkezett (M>N), ahol minden jelentkező megmondta, hogy mely munkákhoz ért, illetve amihez ért, arra mennyi fizetést kérne.

N állás, M(>N) jelentkező, összeget mond, mindenkit fel lehet venni, legolcsóbb kell.

Legyen `F(i,j)` értéke 0, ha az `i`. jelentkező a `j`. munkához nem ért, illetve `F(i,j)=A>0`, ha ért hozzá és `A` forintot szeretne kapni érte.

A megoldási ötlet: ne jelentkezőhöz keressünk állást, hanem álláshoz jelentkezőt! Így a feladat megoldása az előzőével majdnem megegyezik, csupán a két index szerepét kell felcserélni.

Állások:	1.	2.	3.	4.
1. jelentkező:	100	0	0	100
2. jelentkező:	0	200	0	0
3. jelentkező:	200	100	0	0
4. jelentkező:	0	0	200	0
5. jelentkező:	500	0	400	0

```
Legjobb_állás(i,költ,N,M,F,Maxköltt,Y,x):
  Ha i>N akkor Ha költ<Maxköltt akkor Y:=x; Maxköltt:=költ
               Elágazás vége
  különben Ciklus j=1-től M-ig
             Ha nem Volt(j,i,x) és F[j,i]>0
                és költ+F[j,i]<Maxköltt
                akkor x[j]:=i
                      Legjobb_állás(i+1,költ+F[i,j],
                                    N,M,F,Maxköltt,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

4. variáció

Egy vállalkozás N különböző állásra keres munkásokat. Pontosan N jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért, illetve amihez ért, arra mennyi fizetést kérne.

N állás, N jelentkező, összeget mond, nem lehet mindenkit felvenni, legolcsóbb kell.

Legyen `F(i,j)` értéke 0, ha az `i`. jelentkező a `j`. munkához nem ért, illetve `F(i,j)=A>0`, ha ért hozzá és `A` forintot szeretne kapni érte.

A vállalkozás vezetője azt szeretné, ha az összes állást betöltenék, de úgy, hogy számára ez a lehető legkisebb költséggel járna.

Nem biztos azonban, hogy ez lehetséges. Akkor is érdekes lehet azonban egy olyan megoldás, amiben a lehető legtöbb munkát adhatjuk ki.

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	100	0	0	100	0
2. jelentkező:	0	200	0	0	0
3. jelentkező:	200	100	0	0	0
4. jelentkező:	0	0	200	0	400
5. jelentkező:	500	400	0	0	0

A megoldás ötlete: Vezessünk be egy N+1. ún. fiktív állást, amihez azt gondoljuk, hogy mindenki ért! Legyen ennek az ára nagyobb, mint minden más összeg a táblázatunkban! Az N+1. állásra engedjük meg, hogy többen is válasszák!

Belátható, hogy a leggazdaságosabb megoldásban ekkor a lehető legkevesebb fiktív állás lesz, azaz a legtöbb állást tudjuk betölteni.

Állások:	1.	2.	3.	4.	5.	6.
1. jelentkező:	100	0	0	100	0	1000
2. jelentkező:	0	200	0	0	0	1000
3. jelentkező:	200	100	0	0	0	1000
4. jelentkező:	0	0	200	0	400	1000
5. jelentkező:	500	400	0	0	0	1000

```
Legjobb_állás(i,költ,N,F,maxért,Maxkölt,Y,x):
  Ha i>N akkor Ha költ+maxért<Maxkölt
                  akkor Y:=x; Maxkölt:=költ+maxért
               Elágazás vége
  különben Ciklus j=1-től N-ig
             Ha nem Volt(i,j,x) és F[i,j]>0
                és költ+F[i,j]<Maxkölt
                akkor x[i]:=j
                      Legjobb_állás(i+1,költ+F[i,j],
                                    N,F,maxért,Maxkölt,Y,x)
           Ciklus vége
  Elágazás vége
Eljárás vége.
```

5. variáció

Egy vállalkozás `N` különböző állásra keres munkásokat. Pontosan `N` jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért.

N állás, N jelentkező, nem mond összeget (csak ért-e hozzá), mindenkit fel lehet venni

Legyen `F[i,j]` értéke `hamis`, ha az `i`. jelentkező a `j`. munkához nem ért, illetve `igaz`, ha ért hozzá.

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	igaz	hamis	hamis	igaz	hamis
2. jelentkező:	hamis	igaz	hamis	hamis	hamis
3. jelentkező:	igaz	igaz	hamis	hamis	hamis
4. jelentkező:	hamis	hamis	igaz	hamis	hamis
5. jelentkező:	igaz	hamis	igaz	hamis	igaz

```
Munkák(N,F,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N  {lehet még és nincs még kész}
    Jóesetkeresés(i,F,Y,Van,j)
    Ha Van akkor Y[i]:=j; i:=i+1  {előrelépés}
        különben Y[i]:=0; i:=i-1  {visszalépés}
  Ciklus vége
  Van:=(i>N)
Eljárás vége.
```

Jól látható, hogy a fő eljárásban konkrét feladat miatt semmi teendőnk nincs, egyszerűen csak lemásoljuk az általános sémát.

```
Jóesetkeresés(i,F,Y,Van,j):
  j:=Y[i]+1
  Ciklus amíg j≤N és (rossz(i,j,Y) vagy nem F[i,j])
    j:=j+1
  Ciklus vége
  Van:=(j≤N)
Eljárás vége.
```

A második szinten egyszerűsítenünk kellett, `M[i]` helyébe `N` került, a `tilos(j)` függvényt pedig egy mátrix elemre hivatkozással helyettesítjük.

```
rossz(i,j,Y):
  k:=1
  Ciklus amíg k<i és Y[k]≠j
    k:=k+1
  Ciklus vége
  rossz:=(k<i)
Függvény vége.
```

A harmadik szinten semmi teendőnk nem volt.

6. variáció

Egy vállalkozás `N` különböző állásra keres munkásokat. Pontosan `N` jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért.

N állás, N jelentkező, nem mond összeget (csak ért-e hozzá), mindenkit fel lehet venni. Módosított adatábrázolással.

Legyen `D[i]` az i. ember által vállalható munkák száma, `E[i,j]` értéke pedig az általa vállalt `j.` munka sorszáma!

A vállalkozás vezetője azt szeretné, ha az összes jelentkezőt fel tudná venni és minden munkát elvégeztetni.

	Darab	Állások: 1.	2.	3.
1. jelentkező:	2	1	4	
2. jelentkező:	1	2		
3. jelentkező:	2	1	2	
4. jelentkező:	1	3		
5. jelentkező:	3	1	3	5

A megoldásban használt fontos változók:

- `N` – a munkák és a munkások száma
- `D` – a munkások hány munkához értenek (`D[i]` – az `i`. munkás ennyi munkához ért)
- `E` – az adott munkások mely munkákhoz értenek (`E[i,j]` – az `i`. munkás által elvégezhető `j`. munka)
- `Y` – az aktuálisan számolt megoldás: `Y[i]` az `i`. jelentkezőnek adott munka sorszáma

```
Munkák(N,D,E,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N  {lehet még és nincs még kész}
    Jóesetkeresés(i,D,Y,E,Van,j)
    Ha Van akkor Y[i]:=j; i:=i+1  {előrelépés}
        különben Y[i]:=0; i:=i-1  {visszalépés}
  Ciklus vége
  Van:=(i>N)
  Ha Van akkor Ciklus i=1-től N-ig
                 Y[i]:=E(i,Y[i])
               Ciklus vége
Eljárás vége.
```

Jól látható, hogy a fő eljárásban konkrét feladat miatt majdnem semmi teendőnk nincs, egyszerűen csak lemásoljuk az általános sémát.

Az egyetlen módosítandó: mivel a megoldásban a `j` a választás sorszáma, emiatt a végén ebből elő kell állítanunk a munka sorszámát.

```
Jóesetkeresés(i,D,Y,E,Van,j):
  j:=Y[i]+1
  Ciklus amíg j≤D[i] és rossz(i,j,Y,E)
    j:=j+1
  Ciklus vége
  Van:=(j≤D[i])
Eljárás vége.
```

A második szinten egyszerűsítenünk kellett, nincs szükség a `tilos(j)` függvényre.

```
rossz(i,j,Y,E):
  k:=1
  Ciklus amíg k<i és E[k,Y[k]]≠E[i,j]
    k:=k+1
  Ciklus vége
  rossz:=(k<i)
Függvény vége.
```

Mivel j és Y[k] sem munka sorszám, hanem adott munkás munka felsorolásbeli sorszáma, ezért a hasonlítás kissé bonyolultabb lett.

7. Variáció

Egy vállalkozás N különböző állásra keres munkásokat. Pontosan N jelentkező érkezett, ahol minden jelentkező megmondta, hogy mely munkákhoz ért, illetve amihez ért, arra mennyi fizetést kérne. A vállalkozás vezetője azt szeretné, ha az összes jelentkezőt fel tudná venni és minden munkát elvégeztetni, de úgy, hogy számára ez legfeljebb X összegbe kerüljön.

N állás, N jelentkező, összeget mond, mindenkit fel lehet venni, egy megoldás kell, de meghatározunk egy maximális fedezeti összeget

Legyen `F[i,j]` értéke `=0`, ha az `i`. jelentkező a j. munkához nem ért, illetve `>0`, ha ért hozzá, és akkor ez az érték jelentse azt, hogy a munkát ennyiért végezné el.

Ha X=1300, akkor a példában vastagon szedett megoldás helyett a dőlten szedett megoldás is helyes.

Állások:	1.	2.	3.	4.	5.
1. jelentkező:	100	0	0	*100*	0
2. jelentkező:	0	*200*	0	0	0
3. jelentkező:	*200*	100	0	0	0
4. jelentkező:	0	0	200	0	*400*
5. jelentkező:	500	0	*400*	0	200

A példában az is látszik, hogy bármely munkás megbízásával a költség folyamatosan emelkedik, amit a megoldásban ki is használunk.

A megoldásban használt fontos változók:

- N – a munkák és a munkások száma
- F – az adott munkások mely munkákhoz értenek (F[i,j] – az i. munkás által elvégezhető j. munka ára)
- `Y` – az aktuálisan számolt megoldás: `Y[i]` az `i`. jelentkezőnek adott munka sorszáma
- `X` – a rendelkezésre álló összeg

```
Munkák(N,X,F,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N {lehet még és nincs még kész}
    Jóesetkeresés(i,Van,j)
    Ha Van akkor Ha Költség(N,i,j,F,Y)≤X
                    akkor Y[i]:=j; i:=i+1  {előrelépés}
                 különben Y[i]:=0; i:=i-1  {visszalépés}
         különben Y[i]:=0; i:=i-1           {visszalépés}
  Ciklus vége
  Van:=(i>N)
Eljárás vége.
```

Jól látható, hogy a fő eljárásban kell figyelembe venni a költség függvényt: ha az adott lépésig a költség nem haladja meg az X értéket, akkor a megtalált munka hozzárendelés jó, ha meghaladja, akkor pedig rossz, visszalépést okoz.

Megjegyzés: hatékonysági szempontból a két különben ág összevonható:

```
Munkák(N,Van,Y):
  i:=1; Y[]:=[0,…,0]
  Ciklus amíg i≥1 és i≤N {lehet még és nincs még kész}
    Jóesetkeresés(N,i,F,Y,Van,j)
    Ha Van és Költség(N,i,j,F,Y)≤X
                    akkor Y[i]:=j; i:=i+1  {előrelépés}
                 különben Y[i]:=0; i:=i-1  {visszalépés}
  Ciklus vége
  Van:=(i>N)
Eljárás vége.
```

A második szinten egyszerűsítenünk kellett, `M[i]` helyébe `N` került, a `tilos(j)` függvényt pedig egy mátrix elemre hivatkozással helyettesítjük.

```
Jóesetkeresés(N,i,F,Y,Van,j):
  j:=Y[i]+1
  Ciklus amíg j≤N és (Rossz(i,j,Y) vagy F[i,j]=0)
    j:=j+1
  Ciklus vége
  Van:=(j≤N)
Eljárás vége.
```

A Rossz függvénnyel pedig semmi tennivalónk nem lesz:

```
Rossz(i,j,Y):
  k:=1
  Ciklus amíg k<i és Y[k]≠j
    k:=k+1
  Ciklus vége
  Rossz:=(k<i)
Függvény vége.
```

A költségszámítás nagyon egyszerű, összegzés tétellel elvégezhető.

```
Költség(N,i,j,F,Y):
  s:=0
  Ciklus k=1-től i-1-ig
    s:=s+F[i,Y[k]]
  Ciklus vége
  s:=s+F[i,j]
  Költség:=s
Függvény vége.
```

A megoldásban a `Költség` függvény figyelembevétele volt a **korlátozás**, emiatt hamarabb lehetett szükség visszalépésre, ebből derült ki ugyanis, hogy már nem találhatunk jó megoldást.

A korlátozás lényege tehát: a teljes megoldás elkészülte előtt visszalépést okozni!

8. További variációk

Álljon itt pár példa megoldás nélkül, ami a feladat komplexitását változtatja.

- Minden állásra kell ember
- Van, aki több mindenhez ért, ilyen legyen a prioritás
- Több helyszínen vannak az állások, közelben lakók prioritás
- Ha nincs hozzáértő, jöhet olyan is, aki vállalja a betanulást
- Tanulás költségét is bevezethetjük.
- *Betanuló* fizetését plusz a *tanítás* költségét hasonlítsuk össze a *Szakértő* fizetésével

9. Az alapfeladat módosítása

További variálási lehetőség az alapfeladat módosítása. Erre pár lehetőség:

- Konferencián N idősávra M előadó jelentkezik.
- Iskolában N napon tanulmányi versenyek, M tanuló melyiken induljon
- N cég, M ellenőrt küldenek ki, de legfeljebb K helyre, lehetőleg közeli helyekre
- N témavezetőhöz M szakdolgozatos jelentkezik. Osszuk be őket attól függően, hogy ki melyik témához ért, de maximum K-an lehetnek egy témavezetőnél.
- N üzletben M termék, hol vásároljunk, hogy legjobban járjunk. Bevezethetünk az ár mellé szállítási és/vagy utazási költségeket is.

Módszerek

A bemutatott feladatban az állások számát és a jelentkezők számát változtattuk, módosítottunk a megkapott információkon aszerint, hogy tudjuk-e a fizetési igényeket. Úgy változtattuk az adatokat, hogy a legjobb megoldást kellett megtalálnunk vagy egy bármilyet, illetve korlátoztuk az adatokat.

Általánosságban tehát elmondhatjuk, hogy visszalépéses keresés alapjául szolgáló többdimenziós adatainak a nagysága (`N` és `M`), azok egymáshoz való viszonya szolgálhat a variáció elkészítésére.

A megadott adatok típusa, jelentése és egyéb kiegészítő, pontosító vagy esetleg új információ, mint például a korlát bevezetése újabb lehetőségeket ad egy másik változat elkészítéséhez.

Az alapfeladat cseréjével is egyszerűen módosítható a probléma, így további nagyszámú feladat-variáció-csomaghoz jutunk. Feladattól függetlenül megfogalmazhatjuk a következő kombinatorikai kérdéseket:

Összes ismétlés nélküli permutáció

Backtrack: $\forall i\ (1 \le i \le n)$: $\forall j\ (1 \le j < i)$: $X_j \neq X_i$ (alesete: olyan permutációk, ahol $X_i <> i$)

Összes ismétléses permutáció

Backtrack: $\forall i\ (1 \le i \le \Sigma L_i)$: $\forall j\ (1 \le j < i)$: $X_j = X_i$ legfeljebb L_i-szer

Például az egyik előző feladatban: ugyanarra a munkára több jelentkező is felvehető.

Összes ismétlés nélküli variáció

Backtrack: $\forall i\ (1 \le i \le k)$: $\forall j\ (1 \le j < i)$: $X_j \neq X_i$

Összes ismétlés nélküli kombináció

Backtrack: $\forall i\ (1 \le i \le k)$: $\forall j\ (1 \le j < i)$: $X_j < X_i$

Összes ismétléses kombináció

Backtrack: $\forall i\ (1 \le i \le k)$: $\forall j\ (1 \le j < i)$: $X_j \le X_i$

Összes partíció

Olyan K-jegyű számok, ahol a számjegyek összege pontosan N: $\forall i\ (1 \le i \le k)$: $X_j \ge 0$ és $\Sigma X_i = N$

Összes diszjunkt felbontás

N felbontása pozitív (>0) számok összegére: $\forall i\ (1 \le i \le m)$: $Xj > 0$ és $\Sigma Xi = N$

Ezen feladatok hatékony megoldása azonban egyes esetekben nem backtrack-es.

Összefoglalás

A módszer matematikai megalapozásával Fóthi Ákos és kollégái foglalkoztak [8], ehhez azonban a közoktattásban másképp kell hozzáállni.

Cikkünket azzal a céllal írtuk, hogy különböző komplexitású backtrack-es feladatok elkészítésének lehetőségeit felmérjük és bemutassuk. Egy példán keresztül bemutattuk, hogy egy alapfeladat módosításai milyen variációkra ad lehetőséget.

A felsorolt példák tanulságaiból általánosan megfogalmaztuk a módszereket, melyek segítségével még ennek a feladatnak a további variálására is lehetőség lenne. Sőt az alapfeladat módosításával, akár csak a téma lecserélésével is már egy másik problémát kap a feladatmegoldó. Fontos azonban megemlíteni, hogy a feladat módosítása után mindig ellenőrizzük vissza a megoldást, mert bizonyos esetekben előfordulhat, hogy a feladat megoldása oly mértékben eltér, hogy azt már nem is az eredeti módszer szerint kell megoldani. Például előfordulhat, hogy a backtrack helyett dinamikus programozásra lesz szükség, amit esetleg még nem ismernek a diákjaink.

Bízunk benne, hogy az itt bemutatott minták és módszerek segítik olvasóinkat későbbi munkájuk könnyebb elvégzésében, amikor különböző komplexitású backtrack-es feladatok kitűzését végzik.

Hivatkozások

1. Alexander Shen. Algorithms and Programming: Problems and Solutions, Springer, 2010
2. S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani: Algorithms. 2006
3. David G. Sullivan, Recursion and Recursive Backtracking, Harvard Extension School (2012) https://sites.fas.harvard.edu/~cscie119/lectures/recursion.pdf (utoljára megtekintve: 2018. október 31.)
4. Douglas Wilhelm Harder, Backtracink, University of Waterloo, Ontario, Canada, https://ece.uwaterloo.ca/~dwharder/aads/Algorithms/Backtracking/ (utoljára megtekintve: 2018. október 31.)
5. Steven Skiena, Analysis of Algorithms, Backtracing, Stony Brook University New York, (2017) https://www3.cs.stonybrook.edu/~skiena/373/newlectures/lecture15.pdf (utoljára megtekintve: 2018. október 31.)

6. J Zelenski: Exhaustive recursion and backtracking (2008) https://see.stanford.edu/materials/icspacs106b/h19-recbacktrackexamples.pdf (utoljára megtekintve: 2018. október 31.)
7. Lőrentey Károly; Fekete István; Fóthi Ákos; Gregorics Tibor: On the wide variety of backtracking algorithms. pp. 165-174. In: Kovács, Emőd; Winkler, Zoltán (szerk.) Proceedings of the 5th international conference on applied informatics : Education and other fields of applied informatics, computer graphics, computer statistics and modeling. Eger, Magyarország : Molnár és Társa 2001 Kft., (2001) p. 250.
8. Harangozó Éva; Nyékyné Gaizler Judit; Fóthi Ákos; Konczné Nagy Márta: Demonstration of a Problem-Solving Method ACTA CYBERNETICA 12 : 1 pp. 71-82. , 12 p. (1995)

Számítógépes problémamegoldás mérése az informatikaórán

Nagy Tímea Katalin[1], Csernoch Mária[2]

[1] timcs06@gmail.com, [2] csernoch.maria@inf.unideb.hu
DEBRECENI EGYETEM, INFORMATIKAI KAR

Absztrakt. Az Oktatáskutató és Fejlesztő Intézet megállapításából kiindulva, mely szerint „A hazai mérés-értékelési rendszerből hiányzik a Nat-ra és a kerettantervekre épülő fejlesztő értékelés, amely segíthetné a tanulót, pedagógust és szülőt annak megítélésében, hogy hol tart a tanuló a Nat, a kerettantervek és a helyi tantervek által előírt ismeretek, tudás, készségek és kompetenciák elsajátításában.", összeállítottunk egy olyan tesztet, amellyel mérhető, hogy a tanulók milyen mértékben képesek a mindennapi életben előforduló IKT szituációkhoz hasonló feladatokban alkalmazni a tudásukat. Ennek megfelelően, kutatásunk elsődleges célja, hogy felmérjük az általános iskola 7-8. és a középiskola 9-10. évfolyamos tanulóinak számítógépes problémamegoldó készségét tantárgyközi kapcsolatokra épülő feladatok alapján. Felmérésünk további célja egy olyan objektív mérőeszköz megszerkesztése, amely hatékonyan alkalmazható tantervi és oktatástámogató eszközök tervezésében. A témaválasztás időszerűségét még inkább indokolja a véleményezésre megjelent új NAT-tervezet, melyben a tanulók digitális kompetenciájának és számítógépes gondolkodásának fejlesztése kiemelt szerepet kapott. Ez utóbbi cél azért is kiemelt jelentőségű, mert a teszt előkészítése során azt tapasztaltuk, hogy a jelenleg érvényben lévő Nemzeti Alaptanterv és az informatika kerettantervek nem támogatják kellő hatékonysággal a tanulók számítógépes gondolkodásának, algoritmikus, valamint számítógépes problémamegoldó készségének fejlesztését. Az eszközorientáltságon túl a kerettantervek számos nehezen vagy többféleképpen értelmezhető kifejezést tartalmaznak, melyek nem nyújtanak elegendő útmutatást a tanároknak ahhoz, hogy a tanórát megtöltsék valós tartalmakkal. Nem adnak továbbá egyértelmű támogatást arra vonatkozóan sem, hogy hogyan valósíthatóak meg valódi tudástranszfer és tantárgyközi kapcsolatok az informatika egyes tematikai egységei, fejlesztési területei, valamint az informatika és más tárgyak között.

Kulcsszavak: informatika, Nemzeti Alaptanterv, kerettantervek, számítógépes gondolkodás, számítógépes problémamegoldás

Bevezetés

A témával foglalkozó elemzők a Debreceni Egyetem elsőéves informatika szakos hallgatóival végzett mérések alapján megállapították, hogy a hallgatók programozói tudása, algoritmikus és számítógépes problémamegoldó készsége jóval alacsonyabb szintű, mint ami az egyetemi tanulmányaik megkezdéséhez és a sikeres előre haladáshoz szükséges [5], valamint a hallgatók attitűdje, informatikáról alkotott fogalma nem egyezik meg a felsőoktatás bemeneti elvárásaival [7]. Az informatikai felsőoktatási eredménytelenség, valamint a felsőoktatási bejutást lehetővé tevő vizsgáztatási rendszerek mérési módszerei és a felsőoktatási követelményrendszer közötti különbségek [11][12] vezettek jelen méréseket megelőző pilot-mérésekhez.

Mivel a felsőoktatási bemenetnél a hallgatók tanulmányi előéletét kell feltételként vizsgálni, ezért kézenfekvő, hogy a hallgatók nehézségeinek egy része az általános és középiskolai tanulmányokra vezethető vissza. A magyar oktatási rendszerben az informatika, mint tantárgy az általános tantervben 6–10. évfolyamon kötelező [30]–[34], heti egy órában. Korábbi kezdés és/vagy magasabb óraszám a speciális osztályokban vagy a szabadon felhasználható órakeret terhére lehetséges. Függetlenül azonban a kezdés idejétől és a heti óraszámtól, a különböző kerettantervek hasonló tudástartal-

makat fogalmaznak meg [23]–[30]. A Nemzeti Alaptanterv [18]–[21] és a kerettantervek [36],[2]–[30] elemzése során arra a megállapításra jutottunk, hogy az alapdokumentumokban foglaltak megfogalmazása többféle módon értelmezhető, ezáltal nehézségek elé állítva az informatikaszakos pedagógusokat az értelmezésben, a tanmenet összeállításában és a tanításban.

A kerettantervekben továbbá hiányosságok mutathatók ki az informatika különböző területei között megvalósítható tudástranszfer, valamint az informatika és más tantárgyak közötti kapcsolatokban is. Jelen keretek között tudástranszfer fogalom alatt a tématerületek közötti tudásátadást, az interdiszciplinaritást értjük; bizonyos tudományterületeken megszerzett tudás alkalmazhatóságát más tudományterületeken felvetett problémák megoldására – a tudományos precizitás megtartásával –, valamint azt, hogy az így megszerzett ismeret hogyan csatolható vissza az eredeti tudásforráshoz [16].

Ebből adódóan elsődleges célunk volt annak feltárása, hogy az informatika szakos tanárok hogyan értelmezik a kerettantervet, milyen tartalommal és módszerekkel töltik meg az óráikat, továbbá, hogy ezen módszerek mennyire mondhatók hatékonynak. Ennek érdekében egy tanulói tesztet állítottunk össze, melyben az informatikai tudástranszfert, a számítógépes gondolkodást, a számítógépes problémamegoldást és a tantárgyközi kapcsolatokat mérjük általános iskola 7–8. és középiskola 9–10. évfolyamán. Figyelembe véve az Oktatási Hivatal weboldalán olvasható részletet [35], mely szerint a kompetenciamérés „[…] célja nem az adott év tananyagának számonkérése, hanem azt vizsgálja, hogy a diákok az adott évfolyamig elsajátított ismereteiket milyen mértékben tudják alkalmazni a mindennapi életből vett feladatok megoldása során.” [35] azt állapítottuk meg, hogy az általunk összeállított feladatok mindegyike megfeleltethető egy rövidebb, „mini” kompetenciamérő feladatnak. A különbség csupán a bevezető szövegek terjedelmében található meg, ugyanis feladataink utasításai a kompetenciamérés feladataival szemben rövid, egy-két soros utasítások.

A teszt feladatainak bemutatása

A teszt (11–16. ábra) összeállítása során az elsődleges szempont az volt, hogy a teszttel az informatika kerettantervekben megfogalmazott fejlesztési területeket lehetőség szerint lefedjük, egy-egy feladattal pedig a lehető legátfogóbb ismeretre kérdezzünk rá. A kerettantervek elemzése során azt tapasztaltuk, hogy a kerettantervek mindegyike hasonló tudástartalmakat fogalmaz meg az egyes tematikai egységekben, ezért mind az általános, mind pedig a középiskolai évfolyamokon ugyanazon tesztet töltötték ki a tanulók, így vizsgálatainkkal arra is rá tudunk mutatni, hogy a hasonló tudástartalmú tematikai egységeket tekintve milyen tudásbeli különbségek jelentkeznek a különböző évfolyamos tanulók között. A feladatokban nemcsak a kerettantervben implicit módon megfogalmazott ismereteket kértük számon, hanem igyekeztünk olyan ismeretek meglétét is felmérni, melyek nem feltétlenül a feladathoz kapcsolódó tematikai egységben kerültek említésre az informatika órán, ezáltal igyekeztünk a tudástranszfert is vizsgálni. A feladatok mindegyike saját összeállítású, vagy a korábbi évek méréseiből átvett feladatok továbbfejlesztése, melyek mindegyike a valós életből vett problémákra kérdez rá számítógépes környezetben. A tesztben összesen 66 pont volt szerezhető.

A teszt összetételét tekintve a kérdések két részre oszthatók. Az első részben általános kérdéseket (11-12. ábra: Alapinformációk), míg a második részben informatika feladatokat (13–16. ábra: Informatikai ismeretek) fogalmaztunk meg. Ahogyan az 1. táblázatban is látható az „Alapinformációk” fejezetben a nem, a kor és az osztály adatok megadása után a tanulóknak 12 kérdésre kellet választ adniuk. A feltett kérdésekből 11 kérdés volt feleletválasztós, az utolsó kérdésben pedig a tanulók által fontosnak tartott táblázatkezelő függvények felsorolása volt a feladat. Az „Informatikai ismeretek” című fejezetben 17 feladatot adtunk a diákoknak a fájlkezelés, táblázatkezelés, szövegkezelés, algoritmizálás és programozás, forráskezelés témakörökből, valamint 2 kérdés a perifériákról tanult ismeretekre vonatkozott. A feladatokban feltett kérdésekből 12 feleletválasztós, teszt jellegű

kérdés található, a maradék 9 pedig néhány szóban megválaszolható, szöveges választ igénylő feladat volt (1. táblázat).

A teszt kitöltésére biztosított egy tanóra természetesen nem tette lehetővé, hogy a kerettanterv valamennyi témakörét teljesen lefedjük. Egyfelől ez nem is volt célja a mérésnek, mivel jelen keretek között a tudástranszfer-elemek aktualizálására fókuszáltunk, másfelől egyetlen teszt nem lehet alkalmas a teljes informatika tananyag lefedésére.

	Feladat	Feladat típusa
Alapinformációk	Kor	Rövid választ igénylő (szám)
	Nem	Feleletválasztós
	Osztály	Rövid választ igénylő (szám)
	G1–G5	Feleletválasztós
	G6	Rövid választ igénylő (szám)
	G7–G11	Feleletválasztós
	G12	Rövid választ igénylő
Informatikai ismeretek	F1–F3	Feleletválasztós
	F4–F6	Rövid választ igénylő
	F7–F13	Feleletválasztós
	F14–F15	Rövid választ igénylő
	F16	Feleletválasztós
	F17	Feleletválasztós és rövid választ igénylő kérdések

1. táblázat: A teszt feladattípusai.

Összességében a teszt feladataiban feltett kérdések típusát tekintve 70% a feleltválasztós és 30% a rövid választ igénylő kérdések aránya. Ezzel igyekezetünk azt elősegíteni, hogy a teszt feladatai 45 perc alatt, azaz egy tanóra alatt kitölthetők, megoldhatók legyenek. Mindenképpen fontosnak tartjuk megemlíteni, hogy a feladatok nem tématerületek szerinti csoportosításban jelennek meg a tesztben. A kérdések között több olyan is szerepel, amelyek hasonló tudástranszfer-elemet mérnek, de más

szöveg- és problémakörnyezetben. Ezzel a megoldással is azt próbáltuk mérni, hogy az eltérő környezetekben, problémákban hogyan tudják a tanulók felismerni és alkalmazni az egyes tudáselemeket.

Perifériák

Az 5–8. évfolyamra és a 9–12. évfolyamra vonatkozó informatika kerettantervekben egyaránt megjelenik a perifériás eszközök használata, működési elve, ezért a teszt első két feladatát ezen ismeretek mérésére állítottuk össze.

Az F1 feladatban megadtunk 6 darab képet IKT eszközről, melyekről külön-külön el kellett döntenie a tanulóknak, hogy perifériás eszközök-e, vagy sem. Ebben a feladatban azon túl, hogy a kitöltők perifériáról alkotott fogalmát, valamint annak meglétét vizsgáltuk, azt is tanulmányoztuk, hogy a diákok felismerik-e a képeken látható eszközöket. A kérdés fontosságát az adja, hogy az informatika tankönyvek között nincs konszenzus a merevlemezről illetően, azaz egyes tankönyvek háttértárolóként, egyes tankönyvek pedig perifériás eszközként említik a merevlemezt. Miután a tanulók eldöntötték, hogy melyik csoportba sorolják a képeken található eszközöket – perifériás-e, vagy sem –, a perifériásokról meg kellett jelölniük, hogy bemeneti, kimeneti, vagy be- és kimeneti eszközök-e.

Az F2 kérdésben az első feladatra utaltunk vissza, melyben a résztvevő diákok feladata az volt, hogy egy 0–5-ig terjedő Likert-skálán értékeljék, önbevallásos módon, hogy mennyire ismerik az első feladatban látható eszközök működési elvét. Az F1 feladat megoldását a 2. táblázat tartalmazza.

	merevlemez	egér	monitor	headset	nyomtató	szkenner
Perifériás-e?	igen/nem	igen	igen	igen	igen	igen
Beviteli		×				×
Kiviteli			×		×	
Be-és kiviteli	×		×	×		

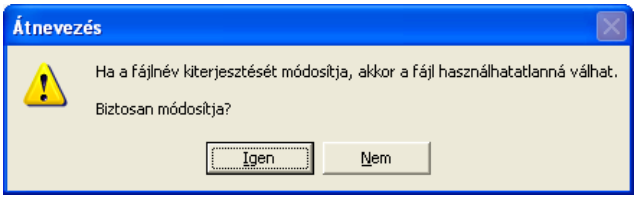
2. táblázat: Az F1 feladat megoldása.

A feladat pontozásakor figyelembe vettük a tankönyvek tartalmának különbözőségeit, ezért a merevlemez esetében teljes értékű válasznak tekintettük, ha a tanuló be-és kiviteli perifériás eszköznek nevezte meg a merevlemezt, de arra is maximális pontszámot szereztek, ha nem perifériás eszköznek nevezték és nem jelölték be egyik adatforgalmi irányt sem. Az egér, a mikrofonos fejhallgató (headset), a nyomtató, valamint a szkenner esetében a táblázatban megadott válaszokat tekintettük helyesnek, a monitor esetében figyelembe vettük az érintőképernyős monitorokat is, így a kiviteli és a be-és kiviteli perifériás eszköz is teljes értékű válasznak számítottak. Az első feladatban minden eszköznél egy pontot adtunk arra a kérdésre, hogy perifériás-e vagy sem, illetve egy pontot arra, hogy jól döntött-e az adatforgalmi irányt illetően, így erre a feladatra 12 pontot szerezhettek a tanulók, a második feladat a teszt javításakor nem került pontozásra.

Fájlkezelés

A tesztben három feladat került összeállításra a fájlkezelői ismeretek mérésére. Ahogyan az 1. táblázatban is látható, az F3 feladat típusát tekintve a feleltválasztós kategóriába sorolható. A tanulók feladata az volt, hogy megválaszolják, mit jelent az 1. ábrán látható üzenet adatfájlok esetén. A válaszlehetőségek között egy helyes és öt helytelen választ jelöltünk meg. Ez a feladat a korábbi években nyitott kérdésként szerepelt a Debreceni Egyetem első éves informatika szakos hallgatóival végzett felmérésekben (TAaAS projekt –Testing Algorithmic and Application Skills) [5][7], így a helytelen válaszok mindegyike az ott szerepelő kérdésre adott válaszok közül került kiválasztásra. A

feladatra a helyes válasz a „Megváltozik a társítás, de a fájl továbbra is használható lesz.” válaszlehetőség, mely a teszt javításakor 1 ponttal volt értékelve.



1. ábra: Az F3 feladatban szereplő minta.

Ugyancsak a fájlkezelés témakörbe tartozik a teszt F6 feladata, azonban az előzőnél komplexebb ismeretekre kérdeztünk rá. „Mi történik, ha duplán kattintunk egy dokumentum fájlra (például: zz.jpg, zz.html, zz.ods, zz.xls)?” A feladat a fájlkezelési ismereteken túl a számítógépes gondolkodást is méri, hiszen a feladat teljes értékű megoldása egy négy lépésből álló algoritmus leírása, helyes informatikai terminológiát használva. Azt kívántuk megvizsgálni, hogy a tanulók el meg tudják-e fogalmazni azt az algoritmus, ami végrehajtódik a dupla kattintás és a kiválasztott adatfájl megnyitása között Ez egy olyan lépéssorozat, amit naponta többször is végrehajtunk, azonban a legtöbb esetben nem tudatosítjuk, hogy mi történik a két szakasz között.

A feladatban megjelenő tudástranszfer-elemek:

- adatfájl vs. programfájl,
- adatfájl vs. dokumentumfájl,
- duplakattintás,
- kiterjesztés,
- társítás,
- program futtatása,
- adatfájl megnyitása.

Feladatot végző	Esemény
felhasználó	dupla kattintás
operációs rendszer	kiterjesztés ellenőrzése annak ellenőrzése, hogy van-e a kiterjesztéshez társított program társított program futtatása adatfájl megnyitása (dokumentumfájl)

3. táblázat: A résztvevők és az algoritmus lépései az F6 feladatban.

A feladat megoldása tehát egy algoritmus, melyben fontos szerepet játszanak a folyamat szereplői. A felhasználó duplán kattint, ezután az operációs rendszer átveszi az irányítást és az algoritmus összes hátralévő lépéséről a rendszer gondoskodik. Fontos megjegyezni, hogy maga a dokumentumfájl definíciója biztosítja, hogy a fájlkiterjesztés és a program között létrejöjjön egy társítás, ami a legtöbb felhasználó számára nem egyértelmű.

A teszt pontozáskor az algoritmus egyes lépéseinek meglétét figyeltük, minden helyesen leírt lépésért 1 pontot adtunk, így ebben a feladatban az összesen szerezhető pont 4.

A fájlkezelés témakör utolsó feladata a tesztben az F16 feladat. A feladat szorosan kapcsolódik a teszt F3 feladatához és a táblázatkezelés témakörhöz is, ezért került a másik két fájlkezelési feladattól távolabb. A feladat kérdése: „Hogyan tudnál egy táblázatkezelő dokumentumot szövegfájllá (.csv vagy .txt) alakítani?". Az F16 feladat – hasonlóan az F3 feladathoz – a TAaAS projekt keretein belül végzett felmérésben szerepelt nyitott kérdésként, így a válaszlehetőségek is azokból a válaszokból kerültek kiválasztásra. A feladat megoldásához a következő fogalmakkal kapcsolatos ismeretekre van szükség: fájltípus, kiterjesztés, mentés másként, dokumentumfájl. A kérdésre a helyes válasz: „Mentés másként, kiválasztjuk az új típust.", mellyel 1 pont szerezhető.

Szövegkezelés

Ebben a fejezetben az F7–F11 és az F13 feladatok kerülnek bemutatásra. A teszt F7 feladatában 5 mintát adtunk a tanulóknak (2. ábra) és az volt a kérdésünk, hogy melyik számozás helyes. A csak helyes számozás kiválasztásáért 2 pontot, amennyiben a kiválasztott számozások között a jó válaszon kívül helytelen válaszok is szerepeltek 1 pontot szerezhettek a tanulók. A feladatra a helyes válasz az E-vel jelölt válaszlehetőség.

A feladatban megjelenő tudástranszfer-elemek:

- automatikus számozás,
- kurzorpozíció,
- nem nyomtatódó karakterek,
- tipográfia.

A 1.A·növény·részei¶

B 3.·Termodinamikus·kölcsönhatás¶

C b.·)·Gabonafélék¶

D ♥ → Petőfi·szerelmi·költészete¶

E 4.→Magyarázza·el·a·ciklusok·működését!

2. ábra: Az F7 feladatban szereplő minták.

Az F8 feladat nyelvi problémákat mutat be digitális környezetben, melyhez a mintaszöveg kiválasztásakor (3. ábra) elsődleges szempont volt, hogy tesztelje a tanulók ismereteit a szóköz, az idéző-jel és a zárójel használatával kapcsolatban [17]. Feladatuk volt, hogy keressék meg és karikázzák be az összes hibát a mintán, majd jelöljék meg, hogy hány hibát találtak.

A·gyermekek·kitárt·karral·a·repülő·méhecskét·utánozzák·,·méhecske·hangját·hangoztatják:·"z'·szaladgálnak·,·ha·leoltjuk·a·villanyt·,·leszállnak·virágport·gyűjteni·(·leguggolnak)·Felkapcsoljuk·a·villanyt,·s·folytatódik·tovább·a·játék.¶

3. ábra: A teszt F8 és F9 feladataihoz csatolt dokumentumrészlet.

A feladatban megjelenő tudástranszfer-elemek:

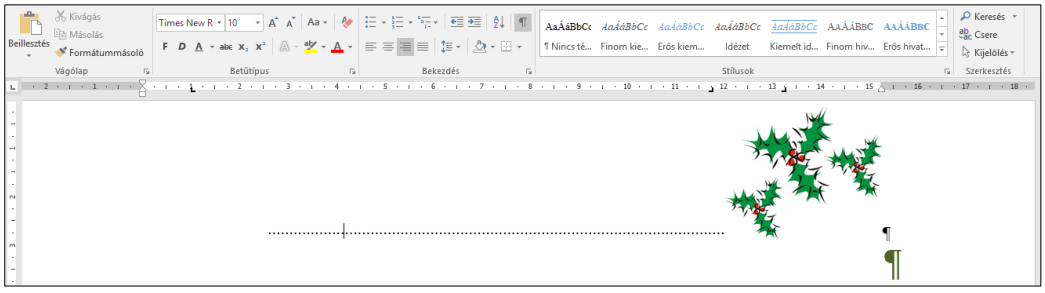
- vessző,
- mondatvégi írásjel,
- idézőjel,
- zárójelek helyes használata.

Az első sorban az „utánozzák” szó után, a második sorban a „szaladgálnak”, a „villanyt” szavak és a nyitózárójel után felesleges szóközök vannak. Az első sor végén a nyitó és a záró idézőjel is rosszul szerepel, valamint hiányzik a mondatvégi írásjel. A harmadik sorban a „felkapcsoljuk” szó nagy betűvel kezdődik, tehát a második sor végéről is hiányzik egy mondatvégi írásjel, így összesen 8 hiba szerepel a szövegben. Minden helyesen bekarikázott helyesírási hibáért 1-1 pontot adtunk, valamint, ha a minta alatti sorban a helyes választ jelölte meg és a helyesen karikázott hibák száma 6-nál több, akkor további 1 pontot szerezhettek a kitöltők. Ugyanezen mintához kapcsolódik az F9 feladat, melyben a zárójelekhez tartozó belső szóközök helyességéről kérdeztük a diákokat. A helyes válaszért, miszerint „A záró zárójelnél helyes, hogy nincs belső szóköz. A nyitó zárójelnél nem, mert van belső szóköz.” 1 pontot kaphattak a tanulók.

Az F10 feladat során a tanulók vizuális – eszköztárról és vonalzóról történő – olvasási képességeit teszteltük. A kérdésünk a következő volt: Az aktuális bekezdésben milyen bekezdésformázás olvasható le a mintáról (4. ábra)? A feladatra a helyes válasz: „Jobbra igazítás és 3 tabulátor.”, melyre 1 pontot volt szerezhető.

A feladatban megjelenő tudástranszfer-elemek:

- bekezdésformázás,
- nem nyomtatódó karakter,
- jobbra igazítás,
- tabulátor,
- bal behúzás.



4. ábra: A bekezdésformázásra vonatkozó kérdéshez tartozó minta az F10 feladatban.

Tipográfiai ismereteket igényel – az előző feladatokhoz hasonlóan 1 pontot érő – F11 feladat, melyben a sorkizárt igazítás által keletkező „utcák” problémáját vetettük fel a tanulóknak egy mintaszövegen keresztül (5. ábra). A feladat az volt, hogy döntsék el, mi okozza a mintán szereplő „utcákat”, melynek megoldásához szükséges ismeretek: igazítás, sorkizárt.


Magyarországra, de csak műszerészként kapott állást az Egyesült Izzóban. A háború minden poklát megjárva, munkaszolgálatot, deportálást, betegséget elszenvedve, 1945 augusztusában került vissza Budapestre.

hazai nyelvstatisztikai automata elkészítésével folytatta. 1960 és 1963 között a **Villamosmérnöki Kar dékánja** volt. Nagy hozzáértéssel irányította a kar első oktatási reformját. Az első tíz év tapasztalatai alapján, a fejlődés irányának jó felismerésével, sikerült a tanterv akkori korszerűsítését

5. ábra: Az F11 feladatban szereplő minta.

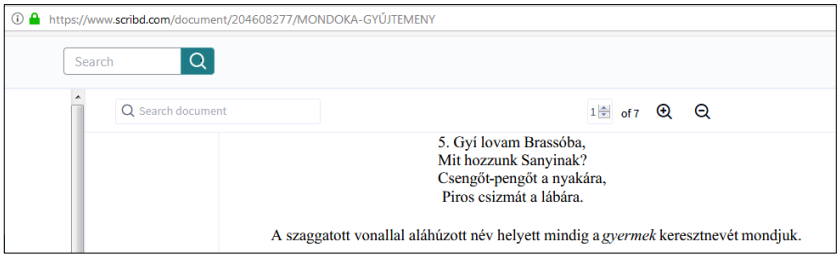
Fontos megjegyezni, hogy a bemutatott F8–F11 feladatokra egyaránt igaz, hogy a minták mindegyike olyan dokumentumokból származik, melyek mindenki számára elérhetőek és letölthetők az internetről [9][14][15].

A szövegkezelés témakörének utolsó feladata a teszt F13 kérdése, mely a forráskezelés témakörének feladatában szereplő mintához (6. ábra) kapcsolódott, a helyes válasz pedig 1 pontot ért. A kérdés a következő volt: Milyen tartalmi (szemantikai) hibát találsz a szövegrészletben? A megoldás: „A név nincs aláhúzva szaggatott vonallal.”

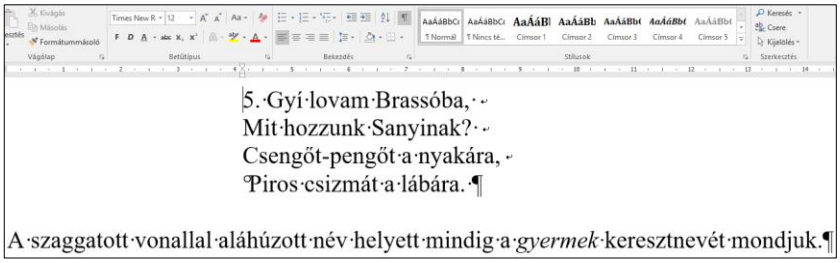
A feladatban megjelenő tudástranszfer-elemek:

- tartalmi (szemantikai) hiba,
- kézi számozás,
- tördelési hiba,
- fölösleges, helytelen szóköz,
- nem törhető szóköz.

A



B



6. ábra: Az F12 és F13 feladatokhoz tartozó minták.

Források kezelése, hitelesség

Az informatika kerettantervekben kiemelt szerepet kap a források kezelése, hitelessége témakör, ezért fontosnak tartottuk, hogy a teszt tartalmazzon egy feladatot ezen fejlesztési terület mérésére is. Az általunk összeállított feladat a tesztben F12 sorszámmal szerepel. Ebben a feleletválasztós kérdésben négy lehetőség közül kellett a tanulóknak eldönteni, hogy melyik a hiteles forrás (6. ábra). A helyes válasz a következő: „Nem tudjuk megmondani, mert egyik sem tartalmaz forrásmegjelölést." A feladathoz felhasznált dokumentumok az interneten, két különböző weboldalon találhatóak meg, azonban egyik sem tartalmaz forrásmegjelölést [15][38]. A feladat pontozásakor a helyes válasz 1 pontot ért.

Táblázatkezelés

Ebben a fejezetben a táblázatkezelés témakörébe sorolható feladatok kerülnek bemutatásra. Fontos megemlíteni, hogy a feladatok között szerepelnek olyanok, melyek nem csak egy csoportba sorolhatók, ilyen például az F15 feladat, valamint az F17 feladat utolsó kérdése, így ezeket a következő fejezetben tárgyaljuk.

A tanulóknak a teszt F14 feladatában szintaktikailag helyes képletekké kellett az ábrát kiegészíteniük.

A feladatban megjelenő tudástranszfer-elemek:

- informatika: minden táblázatkezelői képletet = jellel kezdünk,
- matematika és informatika: függvény fogalma,
- matematika és informatika: argumentumok kezelése (sorrend, zárójelek),
- informatika: összefüggő és nem összefüggő tartomány,
- tartományok megnevezésekor a -tól és -ig, valamint az „és" kulcsszavak szerepe.

A feladat megoldását a következő táblázat tartalmazza (4. táblázat). A pontozáskor minden beírt elemet 1-1 ponttal értékeltük: a két egyenlőségjel, a két zárójel, a két tartományírásjel, a két tartományhivatkozás, így összesen 8 pont volt szerezhető.

	A
1	=SZUM(A2:A20)

A2-től A20-ig

	A
1	=SZUM(A2;A20)

A2 és A20

4. táblázat: A teszt F14 feladatának megoldása.

Az F17 feladatban a probléma gyökere a különböző nyelvekben eltérő szemantikájú vesszőben rejlik. A táblázatot a „Top 250 YouTubers Channels in Hungary" című weboldalról töltöttük le és konvertáltuk (YouTubers, 2018; 7. ábra). Az eredeti weblapon a tartalom hozzáigazodik a választott országhoz, hiába választjuk ki azonban a számunkra megfelelő országot és nyelvet, a szoftver a nyelv szintaxisát figyelmen kívül hagyja, így az angol ezres elválasztó karaktert, azaz a vesszőt alkalmazza elválasztó karakterként. A fájl magyar (és a legtöbb európai) táblázatkezelőben történő megnyitásakor a táblázatkezelő a vesszőt decimális karakterként értelmezi és ha a számban egy vessző

szerepelt, akkor az egész számot valós számmá konvertálja (F17 B6 és B7 cellák és C3–C251 tartomány), vagy ha két vessző is szerepelt a számban, akkor azt string adattípusra alakítja a táblázatkezelő (F17 C2 cella és D2–D251 tartomány).

Rank	Grade	Username	Uploads	Subs	Video Views
1st	A	LetsGoMartin - Nursery Rhymes	194	2,050,447	493,359,935
2nd	B+	REAL TRILL MUSIC	372	328,644	263,907,715
3rd	B+	KerekMese	143	353,603	662,791,261
4th	B	#MISSHMUSIC	124	340,360	201,745,398

7. ábra: A YouTubers csatorna magyar változata a vesszővel, mint ezres elválasztó karakterrel.

Az F17 feladat a táblázatok adattípusainak felismerésére, az ezres elválasztó és decimális karakter helyes / helytelen használatára, valamint a különböző adattípusokra vonatkozó ismeretek adatbáziskezelés és programozás témakörökbe való áthelyezésére irányul.

A tudástranszfer egy másik aspektusa is kiemelt szerepet játszik: a számok szintaktikája, amikor a tudást anyanyelvi és idegennyelvi tanórákból hozzuk. Ebben a tudástranszfer-láncban a táblázatkezelés átmenetként funkcionál a természetes és a mesterséges számítógépi nyelvek között. Fontos megemlíteni a táblázatkezelési megközelítések egy másik jellemzőjét, miszerint a táblázatkezelést általában – csak néhány kivétellel [4][13] – tartalom nélkül tanítják, kizárólag az interfész eszközeire összpontosítva [6]. Ezzel szemben az autentikus táblázatok hiteles adatokat tartalmaznak, amelyek megfelelnek a tanulók érdeklődési körének, és mint ilyenek kiemelkedő motiváló szerepet játszanak a tanítási-tanulási folyamatokban. Következésként, az autentikus táblázatok olyan adatokat szolgáltathatnak, amelyek motiválják a tanulókat a táblázatkezelői eszközök használatára is [3][13]. Kutatások egyértelműen bizonyítják, hogy a táblázatkezelés-oktatás fiaskójának egyik magyarázata a dekontextualizált és eszközcentrikus oktatási megközelítések [6][8].

Ebben a feladatban tehát meg kellett a tanulóknak határozni az adattípusokat az egyes oszlopokban, majd a rekordok számát, továbbá a B oszlopban látható legkisebb és legnagyobb értéket (8. ábra).

	A	B	C	D
1	Felhasználó	Feltöltés	Feliratkozó	Megtekintés
2	VamosART	484	1,107,555	226,195,766
3	Videómánia	338	833,23	254,545,702
4	PamKutya	120	809,866	223,441,355
5	LetsGoMartin	176	725,638	162,798,559
6	TheVR	1,062	592,675	213,550,948
7	luckeY	1,183	561,13	150,341,428
8	Peter Gergely	100	548,241	79,713,757
9	Scribble Netty	159	546,049	74,234,471
248	Szilvaglam	87	61,899	3,918,538
249	rance flow	524	61,863	53,275,385
250	KIS GRÓFO (official)	9	61,65	30,712,031
251	KODIAK	736	61,467	14,599,194

8. ábra: Az F17 feladatban szereplő minta.

A feladatban megjelenő tudástranszfer-elemek:

- a vessző szemantikája,
- programozás: adattípus,
- adatbáziskezelés: adattípus,
- matematika: egész szám, valós szám, tizedes karakter, ezres elválasztó karakter,
- automatikus típusfelismerés és az igazítás kapcsolata.

A feladat első kérdésének megoldását az alábbi táblázat tartalmazza (5. táblázat).

Oszlop	Adattípusok
A oszlop	szöveg
B oszlop	egész szám: (B2:B5) és (B8:B251) valós szám: (B6:B7)
C oszlop	szöveg: C2 valós szám: (C3:C251)
D oszlop	szöveg

5. táblázat: Adattípusok a teszt F17 feladatában szerepelő táblázat egyes oszlopaiban.

A táblában rejtett sorok vannak, így összesen 251 sor található benne, de az első sor a mezőnevekre van fenntartva, így a táblázat 250 adatrekordot tartalmaz. A B oszlopban a mintán szereplő legnagyobb szám 736, míg a legkisebb szám 1,062.

Mint már említettük, a legtöbb európai nyelvben a decimális karakter és az ezres elválasztó karakter eltér az angol nyelvterületeken használttól. A vesszőt decimális karakterként használják az európai nyelvek, de ez az angol nyelvben ezres elválasztó karakterként szolgál, míg a pont a decimális karakter. Az európai nyelvekben a szóköz az ezres elválasztó karakter, bár a digitális világban a nemtörhető szóköz jobb választás [6][3]. A különböző nyelvi környezet és beállítások ismerete fontos, amikor el akarjuk dönteni, hogy milyen adattípusokat és értékeket tartalmaznak a táblázat oszlopai.

Ha a tanulók ismerik a különböző adattípusokat a táblázatkezelésben, akkor ezt a tudást később más környezetekben is alkalmazhatják, például az adatbáziskezelésben és a programozásban is. Ennek az állításnak a megfordítása is igaz. Az adatkezelést igénylő egyéb interfészek tanítása előtt táblázatkezelést érdemes tanítani a tanulóknak, mivel a táblázatkezelő felületek könnyebben kezelhetők, továbbá, a funkcionális nyelvek hatékonyabbnak bizonyultak első szövegalapú programozási nyelvként, mint a „hagyományos" szövegalapú programozási nyelvek [2]. Ezenkívül, az MS (Microsoft) Excel az adatokat a felismert adattípus szerint igazítja (amennyiben az alapértelmezés szerinti igazítást nem írja felül a felhasználó): a cellában balra igazítja a karakterláncokat, jobbra a számokat, középre a logikai értékeket, így az adattípusok minden cellában egyértelműen látszanak. Ezzel a vizualizációval még a nem szándékosan különböző típusú adatok is könnyebben felfedhetők. Azonban tudatában kell lennünk az automatikus adatfelismerésnek a táblázatokban, mivel nem minden adatot ismer fel a szoftver eredeti szándékunknak megfelelően (például a teszt F17 feladatában), illetve meg kell tanulnunk a vizualizáció olvasását. A teszt eredményei megmutathatják, hogy a Z-generáció [37] diákjai ezzel a tudással születnek-e vagy sem, továbbá hogyan képesek ezek a diákok a nem szöveges adatok olvasására és a vizuális reprezentáción alapuló információk elérésére.

Algoritmizálás, programozás

A kerettantervek elemzésekor megállapítottuk, hogy az algoritmusok írása mellett algoritmusok tervezése és elemzése is szerepel a tananyagban, ezt figyelembe véve választottuk ki a teszt F4 és F5 feladatait.

Az F4 feladatot (9. ábra) egy francia matematika órán kapták a 8. osztályos gyerekek. A feladatuk, hogy eldöntsék, mit csinál az algoritmus. Mivel a feladat nyelv és országfüggetlen, ezért döntöttünk úgy, hogy szerepeltetjük a teszt feladati között, továbbá a matematikai órákon a pszeudokódok bevezetése nagymértékben támogatja a matematika és az informatika közötti tudástranszfert, a készségek fejlesztését és új megközelítést nyit meg a matematikai problémamegoldásban [39][40].

A, B, C
Ha $A^2=B^2+C^2$ vagy $B^2=A^2+C^2$ vagy $C^2=A^2+B^2$
    akkor "igen"
    egyébként "nem"
Ha vége.

9. ábra: Az F4 feladat, amelyben azt mértük, hogy a tanulók egy pszeudokód alapján hogyan tudják megfogalmazni, hogy mit csinál az algoritmus.

A feladatban megjelenő tudástranszfer-elemek:

- matematika: Pitagorasz-tétel,
- programozás: változók fogalma,
- programozás: feltételes utasítás fogalma,
- programozás és matematika: a VAGY logikai operátor.

A feladat megoldása: Az algoritmus eldönti, hogy derékszögű-e a háromszög. A pontozáskor a megértés SOLO-kategóriáit vettük figyelembe [5][1]. Ennek alapján a feladatra maximum 4 pont szerezhető.

Az F5 feladat a HÓD verseny Benjamin feladatai közül lett kiválasztva: Virágok és napok (2016-CH-12) [10]. A feladat lényegét a kódfejtés adja, megoldása: Abby. Ha a válaszban a tanulók legalább két jó betűt megadtak, akkor 1, jó válasz esetén pedig 2 pontot szerezhettek (10. ábra).

Barbara két pecsétet kapott. Az egyik virágot, a másik napot nyomtat. Elgondolta, hogy tudná a nevét csak a virágokkal és napokkal lepecsételni. A különböző betűket virágok és napok különböző sorozatával határozta meg.

Betű	B	A	R	E	Y
Sorozat					

A saját nevét, „Barbara” tehát így pecsételte le:

Ezután lepecsételte angol barátja nevét is:

Hogy hívják a barátját? .....

10. ábra: A teszt F5 feladata: kódfejtési probléma a HÓD versenyből vett feladat alapján.

Az F15 feladatban arra voltunk kíváncsiak, hogy meg tudják-e határozni a tanulók, hogy milyen sorrendben hajtja végre a táblázatkezelő a következő képletben szereplő műveleteket:

=HA(ÁTLAG(D2:D58)-50<A5;"a";"")

A feladat érdekessége, hogy alapjaiban egy algoritmizálási, programozási feladat, de besorolható a táblázatkezelési feladatok közé is, mivel egy táblázatkezelői képlet megfejtése a feladat.

A feladatban megjelenő tudástranszfer-elemek:

- matematika: műveletek végrehajtási sorrendje,
- matematika: átlag,
- informatika: ha és átlag függvények,
- informatika: D2:D58 (tartománykijelölés).

A feladat megoldását a következő táblázat mutatja (6. táblázat).

1. lépés	Átlag kiszámítása.
2. lépés	A kapott eredményből kivonunk 50-et.
3. lépés	Kérdés: kisebb-e mint A5?
4. lépés	Ha igen kiírunk egy a betűt, ha nem, akkor kiíratjuk az üres sztringet.

6. táblázat: A teszt F15 feladatának megoldása.

Összesen 6 pontot ért a feladat a következők alapján: amennyiben a tanuló első lépésnek az átlagot írta, akkor 2 pontot kapott, ha az átlag bármely más helyen szerepelt, akkor 1 pontot. Ha az átlag számítása utáni lépésben a kivonás következett, arra további 1 pont járt. 1 pont járt arra is, ha a kivonás után a kérdés megfogalmazása következett, majd arra is, ha a kérdés után a HA() függvényt jelölte meg a soron következő lépésként, végül további 1 pontot kapott, ha ez utolsó lépésként került feltűntetésre.

Ugyancsak az algoritmizálás és a táblázatkezelés csoportjába is sorolható az F17 feladat utolsó kérdése, ezért ennek a megoldását is ebben a fejezetben tárgyaljuk.

A kérdés: mit csinál a következő képlet?

{=SZUM(HA(BAL(A2:A251)="L";1))}

A feladatban megjelenő tudástranszfer-elemek:

- matematika: műveletek végrehajtási sorrendje,
- matematika: átlag,
- informatika: HA(), SZUM(), BAL() és ÁTLAG() függvények,
- matematika: összetett függvény, többváltozós függvény,
- programozás: függvényhívás,
- programozás: függvényírás.

A kérdésben egy tömbképlet eredményét kellett a tanulóknak meghatározni, mely a következő: a tömbképlet meghatározza az L betűvel kezdődő felhasználók számát. A feladat pontozásához az F4 feladathoz hasonló módon SOLO-kategóriákat használtunk. A következő táblázat a feltételes számlálási probléma algoritmusát tartalmazza, amely egy háromszintű tömbképlettel kódolható (7. táblázat).

1.	Input: szöveg Meghatározzuk a szöveg első karakterét. Output: a szöveg első karaktere
2.	Input: a szöveg első karaktere és egy l/L betű Felteszünk egy eldöntendő kérdést: Az előző lépés eredménye egyenlő-e l-lel vagy L-lel? Output: egy vektor IGAZ és HAMIS értékekkel
3.	Input: egy vektor IGAZ és HAMIS értékekkel Ha igaz, akkor a rekordot megjelöljük egy 1-essel, ha nem, figyelmen kívül hagyjuk. Output: egy vektor 1-es és HAMIS értékekkel
4.	Input: egy vektor 1-es és HAMIS értékekkel Összeadjuk az 1-eseket. Output: egy egész szám

7. táblázat: Algoritmus az F17 feladat feltételes számlálás problémájára tömbképlettel történő megoldás esetén.

Összegzés

A teszt feladatainak összeállítása során elsődleges célunk volt, hogy a feladatok mindegyike a kerettantervben megfogalmazott előírásoknak, követelményeknek eleget tegyen. Ennek alapján készítettük el a feladatokat a perifériák, a fájlkezelés, a szövegkezelés, a táblázatkezelés, a forráskezelés, továbbá az algoritmizálás, programozás témakörökből. A teszt feladatainak mindegyike olyan, melyet a kerettanterv szerint már az általános iskolás tanulónak is meg kell tudni oldani, ezért tölthették ki mind az általános iskolás, mind pedig a középiskolás tanulók ugyanazon tesztet. Másik kiemelt szempontunk volt a teszt összeállításakor, hogy mindenki számára érthetően fogalmazzuk meg a feladatok kérdéseit és a válaszlehetőségeket is, ezáltal elősegítve azt, hogy a tanulók minél könnyebben értelmezzék és megoldják a feladatokat, elkerülve az esetleges szövegértési problémákat. A teszt feladatainak mintái valós dokumentumokból kerültek kiválasztásra.

A teszt kitöltése 2018 májusában és júniusában zajlott, papír alapon és online formában, mely során 8880 tanuló vett részt. A tesztek javítását követően minden iskolának megküldtük a saját tanulói eredményeit, amelyek elemzését, értékelését az iskolákon belül is el tudják végezni. A teljes adatbázis részletes elemzésére a következő időszakokban kerül sor. Az előzetes értékelések alapján az alábbi következtetéseket fogalmazhatjuk meg. A fiúk és a lányok teljesítménye között nem találtunk eltérést, tehát a nemek nem befolyásolják a tudástranszfer alapú mérési eredményeket. Ezzel szemben az életkor az osztály és az iskola típus hatással van az eredményekre, valamint az önértékelési véleményezésekre, mely alapján a tapasztaltabb tanulók reálisabban tudják megítélni tudásukat. A táblázatkezelési ismereteket elemezve bizonyításra került, hogy az egymásba ágyazott függvények tanórai használatával növekszik a tanulók táblázatkezelői eredménye, ugyanakkor a tanulók problémamegoldó készsége táblázatkezelői környezetben nem függ az ismert táblázatkezelői függvények számától.

Irodalomjegyzék

1. Biggs, J.B., Collis, K.E. (1982) *Evaluating the Quality of Learning: The SOLO Taxonomy*. Academic Press, New York.
2. Booth, S. (1992) *Learning to Program: A Phenomenographic Perspective*. Goteborg Studies in Educational Sciences 89. Acta Univesitatis Gothoburgenis, Gothenburg.
3. Bujdosó., Gy. & Csernoch. M. (2014) *Digitális írástudás, digitális nyelvhelyesség*. Tudomá-nyos és Műszaki Tájékoztatás 61:(10), pp 1–10.

4. Csernoch, M. (2014) *Programozás táblázatkezelő függvényekkel – Sprego*, Műszaki Könyv-kiadó, Budapest.
5. Csernoch, M., Biró, P., Máth, J. & Abari, K. (2015). *Testing Algorithmic Skills in Traditional and Non-Traditional Programming Environments*. Informatics in Education: an international journal 14: (2) pp 175–197.
6. Csernoch, M. (2017) *Thinking Fast and Slow in Computer Problem Solving*, Journal of Soft-ware Engineering and Applications. Vol.10 No.01 (2017), Article ID: 73749, 30 pages 10.4236/jsea.2017.101002 pp 1–31.
7. Csernoch, M. & Biró, P. (2017). *First year students' attitude to computer problem solving*, IEEE 8th International Conference on Cognitive InfoCommunications: CogInfoCom. Debre-cen, Piscataway (NJ): IEEE Computer Society, pp 225–230.
8. Csernoch, M. & Biró, P. (2018) *Edu-Edition Spreadsheet Competency Framework*, Pro-ceedings of the EuSpRIG 2017 Conference "Spreadsheet Risk Management" ISBN : 978-1-905404-54-4 Copyright © 2017, EuSpRIG European Spreadsheet Risks Interest Group (www.eusprig.org) & the Author(s), https://arxiv.org/ftp/arxiv/papers/1802/1802.00496.pdf (utoljára megtekintve: 2018. július 20.)
9. *Dinner menu €42 per person*. http://isaacsrestaurant.ie/wp-content/uploads/2017/08/%E2%82%AC42-christmas-dinner-menu.doc (utoljára megtekintve: 2018. március 5.)
10. ELTE, NJSZT (2016) *Hódítsd meg a biteket – Benjamin feladatok*, pp 16 http://e-hod.elte.hu/archiv/feladatok/hod2016_benjamin.pdf (utoljára megtekintve: 2018. március)
11. Gombos, E (2014) *Hallgatói eredményesség az adatok tükrében*. Interdiszciplináris pedagógia és a fenntartható fejlődés a VIII. Kiss Árpád Emlékkonferencia, Debrecen. pp. 170–181. http://www.kissarpadkonf.unideb.hu/2013/downloads/kissarpad2013_kotet.pdf (utoljára megtekintve: 2018. július 20.)
12. Gombos, E. és Csernoch, M. (2015) *Knowledge and Result*. END 2015 International Conference on Education and new Developments. 27–29 June Porto, Portugal pp. 44-49. http://end-educationconference.org/wp-content/uploads/2015/09/END2015_Book-of-Proceedings.pdf (utoljára megtekintve: 2018. július 20.)
13. Gross, D., Akaiwa, F. and Nordquist, K. (2014*) Succeeding in Business with Microsoft Excel 2013: A Problem-Solving Approach*. Cengage Learning, Delhi.
14. Informatika-Számítástechnika Tanárok Egyesülete, *Kozma László Országos Informatika Alkalmazói Tanulmányi Verseny*, 2006-2007, 1. forduló, 5-6. osztály
15. Jenei Erika http://www.fejlesztoklapja.hu/files/fejlesztes_jatekok.doc (utoljára megtekintve: 2018. március 5.)
16. Klein, J. T. (1990) *Interdisciplinarity*: History, Theory, and Practice. Wayne State University Press.
17. MTA (2017) *A magyar helyesírás szabályai - Új magyar helyesírás - 12. kiadás*, Akadémia Kiadó, Budapest.
18. NAT 1995 (1995) *130/1995. (X. 26.) Korm. rendelet a Nemzeti alaptanterv kiadásáról.* http://njt.hu/cgi_bin/njt_doc.cgi?docid=24382.38666 (utoljára megtekintve: 2018. június 20.)
19. NAT 2003 (2003) *243/2003. (XII. 17.) Korm. rendelet a Nemzeti alaptanterv kiadásáról, bevezetéséről és alkalmazásáról.* http://www.nefmi.gov.hu/letolt/kozokt/nat_070926.pdf (utoljára megtekintve: 2018. június 20.)
20. NAT 2007 (2007) *202/2007 (VII. 31.) Korm. rendelet a Nemzeti alaptanterv kiadásáról, bevezetéséről és alkalmazásáról szóló 243/2003. (XII. 17.) Korm. rendelet módosításáról.* http://janus.ttk.pte.hu/tamop/tananyagok/curriculum/a_nemzeti_alaptanterv_2007es_vltozat.html (utoljára megtekintve: 2018. június 20.)
21. NAT 2012 (2012) *110/2012. (VI. 4.) Korm. rendelete a Nemzeti alaptanterv kiadásáról, bevezetéséről és alkalmazásáról.* http://ofi.hu/sites/default/files/attachments/mk_nat_20121.pdf (utoljára megtekintve: 2018. június 20.)
22. OFI (2013) *Kerettanterv az általános iskola 5–8. évfolyamára*. Kötelező tantárgyak 2.2.15 http://kerettanterv.ofi.hu/02_melleklet_5-8/2.2.15_informat_5-8.doc (utoljára megtekintve: 2018. július)
23. OFI (2013a) *Kerettanterv az általános iskola 1–4. évfolyamára*. Szabadon választható tantár-gyak 1.3.3 http://kerettanterv.ofi.hu/01_melleklet_1-4/1.3.3_informat_1-4.doc (utoljára megtekintve: 2018. július 20.)

24. OFI (2013b) *Kerettanterv az általános iskola 5–8. évfolyamára.* Kötelező tantárgyak 2.2.15 http://kerettanterv.ofi.hu/02_melleklet_5-8/2.2.15_informat_5-8.doc (utoljára megtekintve: 2018. július 20.)
25. OFI (2013c) Kerettanterv az általános iskola 5–8. évfolyamára. Emelt óraszámú kerettantervek 2.3.2 http://kerettanterv.ofi.hu/02_melleklet_5-8/2.3.2_informat_5-8.doc (Letöltve: 2018. július 20.)
26. OFI (2013d) *Kerettanterv a gimnáziumok 9–12. évfolyama számára.* Kötelező tantárgyak 3.2.16 http://kerettanterv.ofi.hu/03_melleklet_9-12/3.2.16_informat_9-12.doc (utoljára megtekintve: 2018. július 20.)
27. OFI (2013e) *Kerettanterv a gimnáziumok 9–12. évfolyama számára.* Emelt óraszámú kerettan-tervek 3.3.6 http://kerettanterv.ofi.hu/03_melleklet_9-12/3.3.6_informat_emelt_9-12_u.docx (utoljára megtekintve: 2018. július 20.)
28. OFI (2013f) *Kerettanterv a szakközépiskolák 9–12. évfolyama számára.* Kötelező tantárgyak 6.2.11 http://kerettanterv.ofi.hu/06_melleklet_9-12_szki/6.2.11_informat_9-10_sz.doc (utoljára megtekintve: 2018. július 20.)
29. OFI (2013g) *Kerettanterv a szakközépiskolák 9–12. évfolyama számára.* Emelt óraszámú kerettantervek 6.3.7 http://kerettanterv.ofi.hu/06_melleklet_9-12_szki/6.3.7_informat_emelt_9-12_sz_u.docx (utoljára megtekintve: 2018. július 20.)
30. OFI (2013h) *Kerettanterv a szakgimnáziumok 9-12. évfolyama számára.* http://kerettanterv.ofi.hu/20160825_szakgimnazium.doc (utoljára megtekintve: 2018. július 20.)
31. OFI (2013i) *Kerettanterv az általános iskola 1-4. évfolyamára* http://kerettanterv.ofi.hu/01_melleklet_1-4/index_alt_isk_also.html (utoljára megtekintve: 2018. július 20.)
32. OFI (2013j) *Kerettanterv az általános iskola 5-8. évfolyamára.* http://kerettanterv.ofi.hu/02_melleklet_5-8/index_alt_isk_felso.html (utoljára megtekintve: 2018. július 20.)
33. OFI (2013k) *Kerettanterv a gimnáziumok 9-12. évfolyama számára.* http://kerettanterv.ofi.hu/03_melleklet_9-12/index_4_gimn.html (utoljára megtekintve: 2018. július 20.)
34. OFI (2016) *Rövid távú, átmeneti intézkedések a tartalmi szabályozók eredményesebb alkalmazására.* http://docplayer.hu/23037997-Rovid-tavu-atmeneti-intezkedesek-a-tartalmi-szabalyozok-eredmenyesebb.html. (utoljára megtekintve: 2018. augusztus 12.)
35. OFI (2018) *Országos kompetenciamérés.* https://www.oktatas.hu/kozneveles/meresek/kompetenciameres/jogszabalyok. (utoljára megtekintve: 2018. szeptember 21.)
36. OKM (2008) *Kerettantervek. Az oktatási és kulturális miniszter 2/2008. (II. 8.) OKM rendelete a kerettantervek kiadásának és jóváhagyásának rendjéről, valamint egyes oktatási jogszabályok módosításáról szóló 17/2004. (V. 20.) OM rendelet módosításáról.* Magyar Közlöny. 20. szám II. kötet. 2008. február 8., Budapest. (utoljára megtekintve 2018. július 20.)
37. Prensky, M. (2001) *Digital Natives, Digital Immigrants, From On the Horizon* (MCB Univer-sity Press, Vol. 9 No. 5, October 2001), http://www.marcprensky.com/writing/Prensky%20-%20Digital%20Natives,%20Digital%20Immigrants%20-%20Part1.pdf. (utoljára megtekintve: 2018. március 5.)
38. Szomolyai Ovi (2014) *Mondóka gyűjtemény*, https://www.scribd.com/document/204608277/MONDOKA-GY%C5%B0JTEMENY (utoljára megtekintve: 2018. március)
39. Wolfram, C. (2010, July 15). *Stop Teaching Calculating, Start Teaching Math—Fundamentally Reforming the Math Curriculum.* Transcript: Wolfram Technology Conference 2010 Talk. TED Global 2010. http://www.computerbasedmath.org/resources/Education_talk_transcript.pdf. (utoljára megtekintve: 2015. október 4.
40. Wolfram, C. (2015) *Evidence: Let's promote not stifle innovation in education.* http://www.conradwolfram.com/home/2015/5/21/role-of-evidence-in-education-innovation. (utoljára megtekintve: 2015. október 4.

A teszt feladatai

☐ Lány ☐ Fiú Osztály: ☐ Életkor: ☐

Alapinformációk

G1. Milyen típusú iskolába jársz? (egyet jelölhetsz meg)

☐ Általános iskola ☐ Szakgimnázium ☐ Gimnázium ☐ AMI
☐ Szakközépiskola ☐ Szakiskola ☐ egyéb:.....

G2. Informatika órán kívül, más tanórákon HETENTE mennyi időt szoktál SZÁMÍTÓGÉP (asztali számítógép, laptop) használatával tölteni? Húzd alá a választ!

kevesebbet kb. 1 óra kb. 2 óra kb. 3 óra kb. 4 óra kb. 5 óra többet

G3. Informatika órán kívül, más tanórákon HETENTE mennyi időt szoktál MOBIL eszközök (táblagép (tablet), okostelefon, e-book olvasó) használatával tölteni? Húzd alá a választ!

kevesebbet kb. 1 óra kb. 2 óra kb. 3 óra kb. 4 óra kb. 5 óra többet

G4. Összesen mennyi időt szoktál SZÁMÍTÓGÉP (asztali számítógép, laptop) használatával tölteni NAPONTA? Húzd alá a választ!

kevesebbet kb. 2 óra kb. 4 óra kb. 6 óra kb. 8 óra kb. 10 óra többet

G5. Összesen mennyi időt szoktál MOBIL eszközök (táblagép (tablet), okostelefon, e-book olvasó) használatával tölteni NAPONTA? Húzd alá a választ!

kevesebbet kb. 2 óra kb. 4 óra kb. 6 óra kb. 8 óra kb. 10 óra többet

G6. Heti hány órában tanulod/tanultad az informatikát az iskolában?

1. oszt.	2. oszt.	3. oszt.	4. oszt.	5. oszt.	6. oszt.	7. oszt.	8. oszt.	NYEK	9. oszt.	10. oszt.

G7. Mivel foglalkoztok informatika órán? (többet is megjelölhetsz)

☐ ECDL feladatok megoldása.
☐ Önállóan dolgozunk, feladatsor alapján.
☐ Érettségi feladatok megoldása.
☐ Programozás.
☐ Tankönyvi, munkafüzeti feladatok.
☐ Prezentációtervezés, diaminta.
☐ Algoritmusok megfogalmazása.
☐ Források megnevezése.
☐ Fájlkonverziók.
☐ Speciális keresés böngésző programokban.
☐ Számítógép részeinek működési elve.
☐ Dokumentumtervezés.
☐ Egyéb: .....
☐ Táblázatokba adatok gépelése.
☐ Algoritmusok leírása, rajzolása.
☐ Keresés eredményének értelmezése.
☐ Szöveg gépelése.
☐ Adatbázis gépelése.
☐ Játék, szabad foglalkozás.
☐ Hibafelismerés szöveges dokumentumokban.
☐ Elektronikus könyvtárban keresés.
☐ Tömörítési módok és eljárások megismerése.
☐ Elektronikus adatbázisban keresés.
☐ Függvények egymásba ágyazása.
☐ Eredmények helyességének ellenőrzése.

DETeP Debreceni Egyetem Tehetséggondozási Program

11. ábra: A teszt első oldala

G8. Milyen nyelven tanultok/tanultatok programozni? (többet is megjelölhetsz)

☐ C	☐ C++	☐ C#
☐ Logo	☐ Python	☐ Java
☐ Scratch	☐ robot nyelvek	☐ blokk nyelvek
☐ PHP	☐ HTML	☐ CSS
☐ Pascal	☐ Nem tanultunk programozni.	☐ egyéb:.....

G9. Milyen tankönyvetek/munkafüzetetek van informatikából? (többet is megjelölhetsz)

☐ Nincs tankönyvünk/munkafüzetünk. ☐ Egyéb:.....

☐ Nem tudom milyen könyvet/munkafüzetet használunk.

☐ Informatika munkatankönyv 7-8. osztály (Műszaki Könyvkiadó)

☐ Informatika 9-10. (Műszaki Könyvkiadó)

☐ Informatika 7. (Mozaik Kiadó) ☐ Informatika 7. mf. (Mozaik Kiadó)

☐ Informatika 8. (Mozaik Kiadó) ☐ Informatika 8. mf. (Mozaik Kiadó)

☐ Informatika középiskolásoknak (Mozaik Kiadó) ☐ Informatika középiskolásoknak mf. (Mozaik Kiadó)

☐ Informatika 9-10. a gimnáziumok számára (Eszterházy Károly Egyetem)

☐ Nyolcadikos informatika (Pedellus Tankönyvkiadó)

☐ Informatikai ismeretek a középiskolák részére (Jedlik Oktatási Stúdió)

☐ Informatikai feladatgyűjtemény (Jedlik Oktatási Stúdió)

G10. Amennyiben van, milyen gyakran használjátok a tankönyvet/munkafüzetet informatika órán? Karikázd be a megfelelő választ! (0=soha, 5=minden órán)

0 1 2 3 4 5

G11. Hogyan értékelnéd ismereteidet az alábbi témakörökben? (0=egyáltalán nem tudom, 5=nagyon jól tudom)

	Mennyire ismered a témakört? Karikázd be a megfelelő számot!						Iskolában tanultad? tanultam	nem tanultam
fájlkezelés	0	1	2	3	4	5	☐	☐
szövegkezelés	0	1	2	3	4	5	☐	☐
táblázatkezelés	0	1	2	3	4	5	☐	☐
adatbáziskezelés	0	1	2	3	4	5	☐	☐
algoritmizálás, programozás	0	1	2	3	4	5	☐	☐
források kezelése, hitelessége	0	1	2	3	4	5	☐	☐

G12. Sorold fel azokat a táblázatkezelő függvényeket (maximum 15), amelyekről úgy gondolod, hogy ismeretükre feltétlenül szükség van!

1.	4.	7.	10.	13.
2.	5.	8.	11.	14.
3.	6.	9.	12.	15.

DETeP Debreceni Egyetem Tehetséggondozás Program

2/6

SZÉCHENYI

12. ábra: A teszt második oldala

Informatikai ismeretek

F1. Karikázd be a nem perifériás eszközök betűjelét! A perifériás eszközök esetén jelöld meg a képek alatt, hogy bemeneti (B), kimeneti (K) vagy be- és kimeneti (BK)!

Z ☐B ☐K ☐BK

Y ☐B ☐K ☐BK

X ☐B ☐K ☐BK

V ☐B ☐K ☐BK

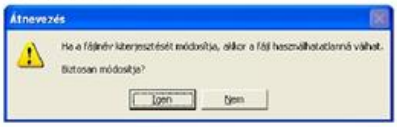
U ☐B ☐K ☐BK

T ☐B ☐K ☐BK

F2. Mennyire ismered ezen eszközök működési elvét? (0=egyáltalán nem ismerem, 5=nagyon jól ismerem)
Karikázd be a megfelelő számot!

Z	0	1	2	3	4	5
Y	0	1	2	3	4	5
X	0	1	2	3	4	5
V	0	1	2	3	4	5
U	0	1	2	3	4	5
T	0	1	2	3	4	5

F3. Mit jelent az alábbi üzenet adatfájlok esetén?



☐ Ha a kiterjesztés változik a gép képtelen lesz felismerni a fájlt.
☐ Kiterjesztés módosításnál adatok elveszhetnek.
☐ Megváltozik a társítás és a fájl használhatatlan lesz.
☐ Megváltozik a társítás, de a fájl továbbra is használható lesz.
☐ Ha rossz formátumra írjuk át a fájlt, akkor az megsérül és használhatatlan lesz.
☐ Az új kiterjesztés más kódolása miatt a tartalom a felhasználó által olvashatatlan lesz.

F4. Mit csinál a következő algoritmus?

A, B, C
Ha $A^2=B^2+C^2$ vagy $B^2=A^2+C^2$ vagy $C^2=A^2+B^2$ .....
  akkor "igen"
  egyébként "nem"
Ha vége. .....

13. ábra: A teszt harmadik oldala

F5. Barbara két pecsétet kapott. Az egyik virágot, a másik napot nyomtat. Elgondolta, hogy tudná a nevét csak a virágokkal és napokkal lepecsételni. A különböző betűket virágok és napok különböző sorozatával határozta meg.

Betű	B	A	R	E	Y
Sorozat					

A saját nevét, „Barbara" tehát így pecsételte le:

Ezután lepecsételte angol barátja nevét is:

Hogy hívják a barátját? .....

F6. Mi történik, ha duplán kattintunk egy dokumentum fájlra (például: zz.jpg, zz.html, zz.ods, zz.xls)?

.....

.....

F7. Melyik számozás helyes? Karikázd be a helyes válaszokat! (többet is megjelölhetsz)

A 1.A·növény·részei¶

B 3.·Termodinamikus·kölcsönhatás¶

C b.·)·Gabonafélék¶

D ♥ · Petőfi·szerelmi·költészete¶

E 4.·Magyarázza·el·a·ciklusok·működését!

F8. Találd meg a helyesírási hibákat a szövegben! Karikázd be ezeket a mintán!
Hány hibát találtál? Karikázd be a megfelelő számot!

A·gyermekek·kitárt·karral·a·repülő·méhecskét·utánozzák·,·méhecske·hangját·hangoztatják:·"z"·szaladgálnak·,·ha·leoltjuk·a·villanyt·,·leszállnak·virágport·gyűjteni·(·leguggolnak)·Felkapcsoljuk·a·villanyt,·s·folytatódik·tovább·a·játék.¶

0 1 2 3 4 5 6 több

F9. Helyesek-e a mintán (F8 feladat) a nyitó és a záró zárójelekhez tartozó belső szóközök?

☐ A nyitó zárójelnél helyes a belső szóköz. A záró zárójelnél nem, mert nincs belső szóköz.

☐ Egyik zárójelnél sem helyesek a belső szóközök.

☐ A záró zárójelnél helyes, hogy nincs belső szóköz. A nyitó zárójelnél nem, mert van belső szóköz.

☐ Mindkét zárójelnél helyesek a szóközök.

DETeP Debreceni Egyetem Tehetséggondozás Program

4/6

14. ábra: A teszt negyedik oldala

F10. Az aktuális bekezdésben milyen bekezdés-formázás olvasható le a mintáról?

☐ Nem látszik, hogy melyik az aktuális bekezdés, mert nincs szöveg a mintán.

☐ Jobbra igazítás és 3 tabulátor.

☐ Szóközökkel a kép jobbra igazítása.

☐ Bal behúzás, ami onnan látszik, hogy a szóközök nem a margónál kezdődnek.

F11. Mi okozza a nagy lyukakat („utcákat") a szavak között?

Magyarországra, de csak műszerészként kapott állást az **Egyesült Izzó**ban. A háború minden poklát megjárva, munkaszolgálatot, deportálást, betegséget elszenvedve, 1945 augusztusában került vissza **Budapest**re.

Ihrtai nyelvtanításának automata elkészítésével folytatta.
1960 és 1963 között a **Villamosmérnöki Kar dékánja** volt. Nagy hozzáértéssel irányította a kar első oktatási reformját. Az első tíz év tapasztalatai alapján, a fejlődés irányának jó felismerésével, sikerült a tanterv akkori konzervatívait

☐ Tabulátor. ☐ Sorkizárt igazítás. ☐ Sok szóköz.

☐ A szövegszerkesztő program hibája, mert széthúzta a szöveget.

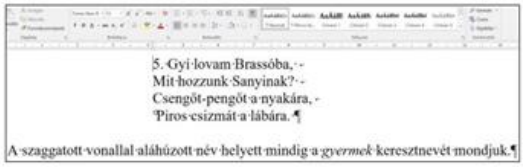
F12. Döntsd el az alábbi szövegrészekről, hogy melyik lehetett az eredeti dokumentum!

A

5. Gyi lovam Brassóba,
Mit hozzunk Sanyinak?
Csengőt-pengőt a nyakára,
Piros csizmát a lábára.

A szaggatott vonallal aláhúzott név helyett mindig a gyermek keresztnevét mondjuk.

B



5. Gyi lovam Brassóba, -
Mit hozzunk Sanyinak? -
Csengőt-pengőt a nyakára, -
Piros csizmát a lábára. ¶

A szaggatott vonallal aláhúzott név helyett mindig a *gyermek* keresztnevét mondjuk.¶

☐ A, mert az az interneten található.

☐ B, mert az egy Word dokumentum.

☐ Nem tudjuk megmondani, mert egyik sem tartalmaz forrásmegjelölést.

☐ Mindkettő, mert a mondókák szájhagyomány útján terjedhetnek.

F13. Olvasd el a szövegrészletet (F12 feladat)! Milyen tartalmi (szemantikai) hibát találsz a szövegrészletben?

☐ Kézi számozás. ☐ Minden sor végén extra szóköz.

☐ Nem-törhető szóköz a mondóka utolsó sorában. ☐ A név nincs aláhúzva szaggatott vonallal.

15. ábra: A teszt ötödik oldala

F14. Hogyan egészítenéd ki az A1 cella képletét úgy, hogy ne kapjunk hibaüzenetet (szintaktikailag helyes legyen)? Javítsd ki a képleteket az ábrán, majd írd az ábra alá, hogy mely tartományokra hivatkozol!

	A
1	SZUM(A2 A20

	A
1	SZUM(A2 A20

F15. Írd le, hogy milyen sorrendben hajtja végre az alábbi képletben szereplő műveleteket a táblázatkezelő!

=HA(ÁTLAG(D2:D58)-50<A5;"a";"")

1. lépés	
2. lépés	
3. lépés	
4. lépés	
5. lépés	

F16. Hogyan tudnál egy táblázatkezelő dokumentumot szövegfájllá (.csv vagy .txt) alakítani? (egyet jelölhetsz meg)

☐ Konvertálás. ☐ Importálás. ☐ Exportálás.
☐ Társítás. ☐ Átírjuk a kiterjesztést. ☐ Mentés másként, átírjuk a kiterjesztést.
☐ Google-ben rákeresek. ☐ Online konverter. ☐ Mentés másként, kiválasztjuk az új típust.

F17. A következő kérdések az alábbi MAGYAR nyelvű táblázatra vonatkoznak. Milyen típusú adatok szerepelnek az egyes oszlopokban a 2. sortól kezdődően? (többet is megjelölhetsz)

	A	B	C	D
1	Felhasználó	Feltöltés	Feliratkozó	Megtekintés
2	VamosART	484	1,107,555	226,195,766
3	Videománia	338	833,23	254,545,702
4	PamKutya	120	809,866	223,441,355
5	LetsGoMartin	176	725,638	162,798,559
6	TheVR	1,062	592,675	213,550,048
7	luckeY	1,183	561,13	150,341,428
8	Peter Gergely	100	548,241	79,713,757
9	Scribble Netty	159	546,049	74,234,471
248	Szilvaglam	87	61,899	3,918,538
249	rance flow	524	61,863	53,275,385
250	KIS GRÓFO (official)	9	61,65	30,712,031
251	KODIAK	736	61,467	14,599,194

	Egész szám	Valós szám	Szöveg	Logikai	Dátum	Egyik sem
A oszlop	☐	☐	☐	☐	☐	☐
B oszlop	☐	☐	☐	☐	☐	☐
C oszlop	☐	☐	☐	☐	☐	☐
D oszlop	☐	☐	☐	☐	☐	☐

Hány adatsort (rekordot) tartalmaz a táblázat? .....

Írd le a B oszlopban látható LEGNAGYOBB számot! .....

Írd le a B oszlopban látható LEGKISEBB számot! .....

Mit csinál a képlet? {=SZUM(HA(BAL(A2:A251)="L";1))} .....

.....

DETeP Debreceni Egyetem Tehetséggondozás Program

6/6

16. ábra: A teszt hatodik oldala

Scratch-től JavaScript-ig

Németh Tamás[1], Tornai Henrietta[2]

[1]tnemeth@inf.u-szeged.hu
Szegedi Tudományegyetem
[2]tornai.henrietta@stud.u-szeged.hu
SZTE TTIK

Absztrakt. Az elmúlt években egyre több olyan tananyag íródott és képzés indult, ami a Scratch használatára épül. Az eredetileg gyerekeknek szánt környezetben bármely korosztály könnyen és szórakozva szerezheti meg a programozáshoz szükséges alapokat. Azt tapasztaltuk, hogy Scratch-ben jól boldogulnak és szívesen programoznak a gyerekek, ám amikor áttérnek valamely programozási nyelv tanulására, elveszik a motiváció és sokan lemorzsolódnak. A kutatásunk fő célja feltárni, hogy pontosan melyik tananyag-egységnél és konkrétan mi okozza ezt a lemorzsolódást, és hogyan lehet ezt csökkenteni.

Kulcsszavak: gondolkodás, algoritmusok, programozás, informatika, Scratch, Java, JavaScript, lemorzsolódás, oktatás, közoktatás

1. Bevezető

A következőkben szeretnénk ismertetni a Scratch és a Java programozási nyelv működését. A Java egy széleskörűen elterjedt objektumorientált nyelv, a Scratch pedig szintén objektumorientált alapokra épít. A cikkben külön kitérünk a két programozási nyelv az közoktatásban betöltött szerepükről, milyen előnyei és hátrányai mutatkoznak meg. A cikk elsősorban arra fókuszál, hogy bemutasson egy lehetséges megközelítést, melynek segítségével a absztraktabb objektumok megértését könnyítik meg.

2. A Scratch és oktatása

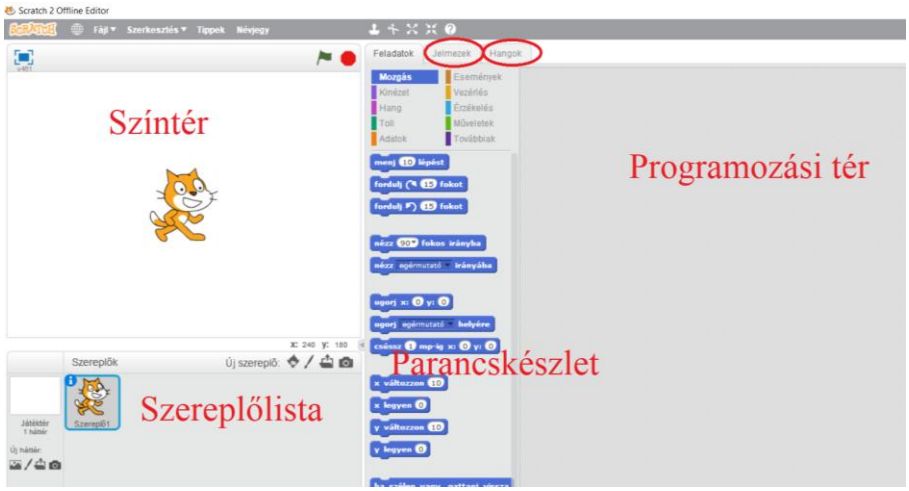
Napjainkban széleskörűen elterjedt, az eredetileg gyerekeknek (8-18 éves korosztály) készült programozási környezet. Izgalmas, kreatív, azonnal látható eredménnyel, ennek köszönhetően szórakozva tanul vele gyerek és felnőtt egyaránt. Már több tanterv is íródott, ami ezen a felületen keresztül oktatja a programozás alapjait.

2.1. Felépítése

A tervezőknek sikerült egy jól átlátható, könnyen kiismerhető platformot kialakítani. Összesen négy fő részt különíthetünk el az indításkor megjelenő ablakban.

- *Színtér:* Itt láthatjuk a programunk eredményét. A felső sávban elhelyezett *zöld zászló* és *piros kör* a program futtatására és megállítására szolgál, az integrált fejlesztői környezetekben használt gombokhoz hasonlóan.
- *Szereplőlista:* A programunkban használt szereplők listája található itt, valamint a háttereket is itt tudjuk kezelni. A program által kínált lehetőségek közül is válogathatunk, de saját képeket is betölthetünk szereplőnek, vagy háttérnek, sőt rajzolhatunk is saját mintát a programon belül.
- *Parancskészlet:* A *Feladat* fül alatt kategorizálva találhatók a blokkok, melyekkel a kiválasztott szereplőnek (vagy a háttérnek) utasításokat adhatunk.
- *Programozási tér:* Erre a területre kell behúzni a használni kívánt blokkokat.

A kiválasztott szereplő jelmezét szerkeszthetjük, és ugyanazon szereplőnek lehet több jelmeze is. Ezeket a változtatásokat a *Jelmezek* fülre kattintás után tehetjük meg, ami a *Feladatok* fül mellett található. Az előzőek mellett lévő *Hangok* fül alatt pedig a szereplő által kiadott hangokat szerkeszthetjük.

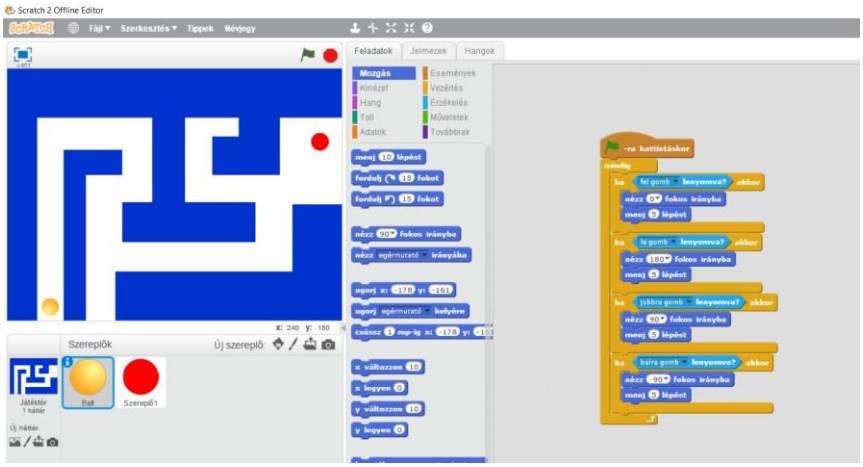


1. ábra: Scratch kezdőképernyő

2.2. Oktatása

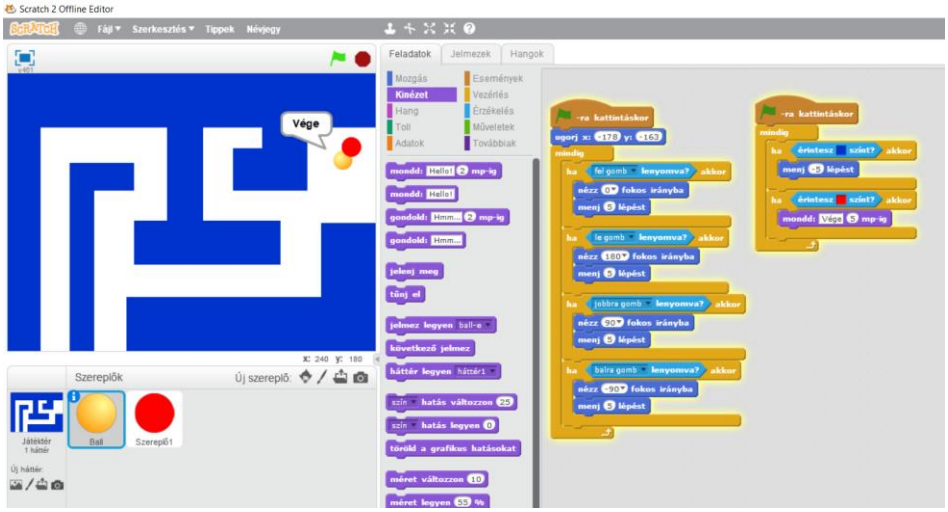
Az első lépés a környezet megismertetése a gyerekekkel. Magyar nyelven is elérhető, könnyen átlátható, így gyorsan felfedeztethető a diákokkal. Igazából az előző pontban ismertetett főbb egységeket elég megmutatni nekik, a későbbiek során ezen információk alapján ők is megkereshetik az egyéb funkciókat, vagy a feladatok megoldásánál rávezethetők.

A tantervek többségében az első lépés a környezet megismerése után, hogy valamit mozgásra bírjunk a színtéren. A legegyszerűbben ezt az indításkor alapértelmezetten megjelenő macska szereplő megfelelő utasításával elő is idézhetjük. A Scratch megnyitásakor a *Feladatok* fül alatt található *Parancskészlet* a *Mozgás* kategória blokkjait mutatja, ahol az ehhez szükséges utasítások vannak. Így sokáig keresgélni sem kell, máris látványos eredményt érhetünk el. Szinte az első percek után sikerélményt szerezhetnek vele. A kezdeti sikerek után érdemes hagyni, hogy egy kicsit önállóan fedezzék fel a lehetőségeket, próbálkozzanak. Hirtelen nagyon sok kérdésük lesz azzal kapcsolatban, hogy mit és hogyan lehet megvalósítani, ötletek repkednek a fejekben, és ezeket nem tartják magukban. Ezután egy olyan játékot érdemes közösen megvalósítani, amit a csoport ismer, és egyszerűen kivitelezhető Scratch-ben. Ilyen például egy labirintus játék. Az alapjátékhoz megfelelő hátteret, cél mezőt és szereplőt kell alkotni. Könnyen be lehet állítani, hogy a szereplőt billentyűkkel irányíthassuk.



2. ábra: Labirintus játék - első rész

Amennyiben ezzel végeztünk, szinte mondani sem kell, már rögtön ki is próbálják a diákok, majd meglepődnek azon, hogy a szereplő átmegy a falon. Ez egy remek alkalom felhívni a figyelmüket arra, hogy a program csak azt tudja végrehajtani, amire utasították korábban. Rögtön látják az eredményen, hogy a program nem gondolatolvasó, és hiába szeretnék, ha a labda megütközne a falnál, az mégis átmegy rajta. Sajnos gyakori hiba a programozás alapozó kurzusokon, hogy a hallgatók elvárják a programtól azt, amit egyáltalán nem, vagy nem úgy programoztak bele, ahogy szeretnék, hogy működjön. Egy néhány perces felvezetés, majd közös ötletelés után a gyerekekkel együtt már orvosolható a falnak ütközés problémája. Hasonló gondolkodással az is megoldható, hogy célba érés esetén történjen valami.



3. ábra: Labirintus játék – második rész

Az alapjátékot érdemes közösen elkészíteni. Fejleszteni sokféleképpen lehet, ahogy egyre több funkciót megismernek a Scratch-ben. Akár a következő órákon együtt is megvalósíthatóak, de ajánlottabb egy új játékot készíteni, és azáltal fedezni fel újabb lehetőségeket, így változatos marad a képzés, és a kreativitásukat sem gátoljuk. Mivel minden diák más, így nem szerethetik ugyanazt a

játékot, és nem élvezik egyformán bármelyik elkészítését. A számukra legkedvesebb játékot úgy is tökélyre fejlesztik otthon a saját kedvük szerint.

A Scratch-nek van online és offline változata is. Az offline verzió ingyenesen letölthető, és percek alatt telepíthető. Az online változatot regisztráció nélkül is ki lehet próbálni, regisztrációt követően pedig online bárhonnan elérhetjük előző munkáinkat, és mások nyilvános projektjei között is nézelődhetünk. Nemcsak közvetve, hanem közvetlenül is módon megvalósíthatóak a következő programozási problémák:

- elágazás (egy vagy kétágú)
- ciklus (számlálós, végtelen, feltételes)
- szekvencia
- változó
- lista
- eseményvezérlés
- többszálúság
- rekurzió
- eljárás
- függvény

3. A probléma kifejtése

A Java egy széleskörűen elterjedt, tisztán objektumorientált programozási nyelv. A fejlesztéshez, a fordításhoz, valamint a futtatáshoz szükséges program és környezet ingyenesen elérhető az interneten. A programkód hordozható, és a számítógépes alkalmazások mellett, telefonon és szervereken futó alkalmazások készítésére is alkalmas. A tanmenetek az első órát a fejlesztői környezet megismertetésére szánják, illetve a nyelv alapinformációinak ismertetésére. Az első program, amit rögtön ezen az órán, a motiváció felkeltése érdekében ajánlott megmutatni, esetleg megíratni a diákokkal, a szokásos "Hello Világ!" kiíratása. Ha a "Hello Világ!" típusú feladaton túl vagyunk, akkor a továbbiakban folytathatjuk a képernyőre írással kapcsolatos ismeretek átadásával. Ez azért is hasznos, mert rögtön látható a munkájuk eredménye. A további egységeket absztrakció szerinti sorrendben sorakoztatjuk fel:

- azonosítók, adattípusok, változók
- konstansok, kifejezések, operátorok
- utasítás, blokk, vezérlési szerkezetek (elágazások, ciklusok)
- adatszerkezetek (egy- és többdimenziós tömbök, karakterlánc)
- osztályok, objektumok, alaposztályok
- *private* és *public* hozzáférés
- öröklődés, polimorfizmus, absztrakt osztályok
- kivételkezelés
- fájlkezelés
- dátum és időkezelés
- *Collection* interfész (halmaz, rendezett halmaz, lista)
- hashtábla, dinamikus vektor, verem

Ezen ismeretek elsajátítása és gyakorlása során a legtöbbször felöltöztetett feladatokkal találkozik a diák, amelyek nem táplálják a motivációt, és nem látják, hogy miért is lenne ez számukra hasznos. Ha nem felöltöztetett, akkor valamilyen fizikai jelenség modellezése, vagy matematikai probléma megoldása kerül feladatként kitűzésre, azonban az ezen tárgyakkal szembeni ellenszenvük miatt, a motiváció csökkenése jellemző.

4. A Java és a Scratch összehasonlítása az absztrakció szempontjából

Az ismertetések alapján már érezhető miben adnak más élményt, de néhány gondolatban szeretnénk kiemelni a diákok által szerintünk fontosnak gondolt különbségeket. Míg a Scratch-ben folyamatosan látják a munkájuk eredményét, ami színes, mozgó, ingergazdag, addig Java programozás közben kevésbé szembetűnő az eredmény. A másik nagy különbség, hogy Scratch-ben elég csak a már meglévő blokkokat a megfelelő helyre húzni, Java-ban pedig már a diáknak kell begépelnie a kódot, ami sokkal több hibázási lehetőséget rejt a pontos gépelés miatt. Egy elfelejtett írásjel, vagy egy elgépelt szó következtében már vagy nem úgy, vagy egyáltalán nem fog működni a kódunk. A fejlesztői környezet azonban sokszor a hozzájárul ezen hibák minimalizálásához. A fejlesztői környezet segítsége ellenére is könnyű tabulálatlan, vagy rosszul tabulált kódot gépelni. A Scratch blokkjainak kapcsolódásai megoldják a tabulálási problémát.

```
import java.util.*;
public class fa {
    int elemszam ;
    class fapont{
        int key, value ;
        fapont bal, jobb, apa ;
    }
    fapont nil, gyoker ;
    void fa() {
        fapont q = new fapont() ;
        nil=q ;
        gyoker=q ;
        elemszam=0 ;
    }
    boolean beszur(int x, int v) {
        fapont p,papa ;
        p=gyoker ;
        papa=nil ;
        while (p!=nil && p.key!=x) {
            papa=p ;
            if (x<p.key) p=p.bal;
            else p=p.jobb;
        }
        if (p==nil) {
            fapont ujpont = new fapont();
            if (gyoker==p) gyoker=ujpont ;
            ujpont.key=x ;
            ujpont.value=v ;
            ujpont.bal=nil;
            ujpont.jobb=nil;
            ujpont.apa=papa;
            if (papa!=nil) {
                if (x>papa.key) papa.jobb=ujpont;
                else  papa.bal=ujpont;
```

4. ábra: Java kódrészlet

5. ábra: Scratch kódrészlet

Egy Java kód megírásához ismerniük kell a nyelv szintaktikáját is, nem elég ha tudja az algoritmust, amivel meg lehet oldani a problémát. Scratch-ben csak azt kell tudnia, hogy hol találja a meg-

oldáshoz szükséges blokkot. Látja az összes lehetőséget, és kikeresheti a megfelelőt, így az algoritmikus gondolkodásra nagyobb hangsúlyt lehet fordítani. A Scratch-ről Java-ra való áttérésben a legnagyobb buktatót a programkód írása jelenti. Kezdve azzal, hogy nem elég csak a megfelelő blokkot, a megfelelő helyre húzni, és helyette gépelni kell a kódot, de még arra is figyelniük kell, hogy megfelelő írásjelet használjanak, ott ahol kell. Ha még ezekkel a nehézségekkel sikeresen meg is küzd a diák, sokkal ingerszegényebb eredményt lát viszont a képernyőn a kezdetekben.

5. Ötletek a távolság csökkentésére

Olyan megoldást szeretnénk találni, amivel könnyebbé tehetjük ezt a váltást, motiváltak maradhatnak a gyerekek. Egy lehetséges megoldást látunk a következőkben ismertetett rendszerekben.

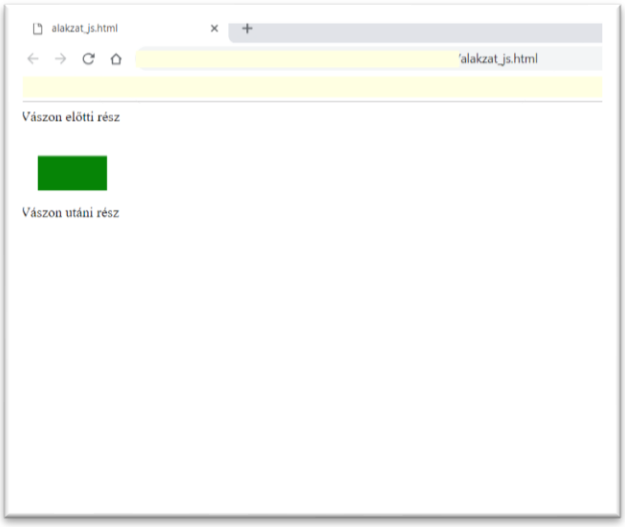
5.1. JavaScript

A JavaScript egy oylan scriptnyelv, melyet weboldalak dinamikussá tételére használják elsősorban. A felhasználó mindamellett, hogy információt szerez a weboldalról, ő is küldhet információt kattintással, szöveg bevitelével. Ezeket az eseményeket kezelhetjük JavaScript-tel. Legfőbb alkalmazási területe az űrlapok kezelése. A JavaScript kódhoz nincs szükség speciális fejlesztési környezetre, csak egy szövegszerkesztőre és egy böngészőre. A böngésző sorról sorra haladva hajtja végre a parancsokat. A JavaScript kódot HTML állományhoz kell kapcsolni, erre több lehetőség is van. Beágyazhatjuk a HMTL kódunkba, vagy külön fájlban is írhatjuk (.js kiterjesztés), ilyenkor csatolni kell a HTML kódba. Egy egyszerű weblapot már pár sor kóddal létre lehet hozni, az eredményt mentés után a böngészőben megtekinthetjük. Ha ehhez JavaScript kódot is társítunk, interaktívvá tehetjük az oldalt,ami egy frissítés után a megváltozott weblapot mutatja, így azonnali sikerélményt garantál. Nem csak űrlapok kezelésére képes, szép színes minták rajzolására is, aminek köszönhetően a Scratch-hez hasonlóan a munkánknak látványos kimenetele lesz. Lehetőség van minden apró változtatás esetén megfigyelni miben különbözik a frissített weblap. A grafikus elemek megjelenítéséhez a *canvas* HTML elemre van szükségünk.

alakzat_js.html

```
1  <html>
2  <head>
3  </head>
4  <body>
5   <p>Vászon előtti rész</p>
6   <canvas width="120" height="60"></canvas>
7   <p>Vászon utáni rész</p>
8  <script>
9     let canvas = document.querySelector("canvas");
10    let context = canvas.getContext("2d");
11    context.fillStyle = "green";
12    context.fillRect(20, 20, 80, 80);
13 </script>
14 </body>
15 </html>
```

6. ábra: Grafikus elem megjelenítése - kód



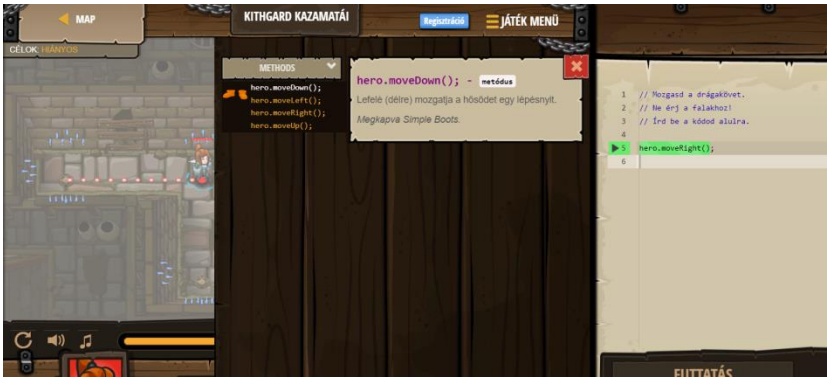
7. ábra: Grafikus elem megjelenítése - weblap

Mindamellett, hogy a látványvilágában majdnem olyan szintű élményeket nyújt, mint a Scratch, a JavaScript a C++ és a Java szintaxisán alapszik, így átvezetésként jól működhet a két programozás között.

A JavaScript oktatásának kezdésére kiváló lehetőség a CodeCombat. A későbbiekben a látványos eredmény vonalon maradva remek környezetek és kisegítő lehetőségek várnak ránk. Ezek közül is ismertetnénk néhányat.

5.1.1. CodeCombat

A CodeCombat egy olyan platform, ami játékkal motivál programozásra. Python mellett JavaScript nyelv is elérhető. A feladat pedig az, hogy a választott programozási nyelven adott utasításokkal segítsük hősünket abszolválni a pályákat. A sikeres teljesítés pontokat ér, szinteket léphetünk. Van fejlődési lehetőség, a diákok között versenyhelyzet alakulhat ki. Mint mindenhol, itt is több jó megoldása lehet egy problémának, ilyenkor a rövidebb kód több pontot ér, ezzel is ösztönzi a felhasználót arra, hogy megtalálja a legjobb algoritmust.



8. ábra: CodeCombat felület játék közben

Ez ideális felület, a JavaScript játékos bevezetésére, még jobban csökkentve a szakadékot a Scratch blokk-alapú kódolása, és a Java között.

5.1.2. HTML SVG

Az SVG egy XML alapú leírónyelv. Segítségével felbontástól függetlenül készíthetünk álló és mozgó képeket egyaránt. A *canvas* elem ezzel ellentétben csak előre meghatározott felbontással rendelkező grafikákat készíthetünk.

5.1.3. Node.js

A Node.js újabb lehetőséget ad a JavaScript programunk futtatására, nem webszerver. Ingyenes letölthető telepítők installálása után vagy shell-ben futtatjuk a kódunk, vagy pedig a JavaScript fájlt adjuk át neki.

5.1.4. Vue-CLI

A Vue.js egy reaktív keretrendszer. Egyszerű, könnyen elsajátítható lépésről lépésre, és akár komolyabb alkalmazásokat is készíthetünk vele. A Vue-CLI pedig ezt a rendszert támogató eszköz. Teljes grafikus környezetet biztosít Vue.js alkalmazásunk készítéséhez.

9. ábra: Vue-CLI logó

6. A módszer tesztelésének menete

Terveink között szerepel a módszer kipróbálása, és tesztelése. Jelenleg még a megfelelő tesztcsoport keresése zajlik. Szeretnénk egy előzetes ismeret és motiváció-felmérést végezni, a folyamat közben és végén pedig megismételni. Jelenleg is vannak olyan iskolák, ahol a bevezető képzés Scratch-ben zajlik, majd JavaScript-re térnek át. Velük együttműködve, a tapasztalatok megosztással tökélesíthető a módszer. Minél szélesebb körben próbálhatjuk ki, annál több tényező által javítható, pontosítható a tanmenet.

7. Összegzés

A mai világban már más igényei vannak a diákoknak. Jobban vágynak az erős és állandó külső impulzusokra, így ha első programozással kapcsolatos élményük a többsoros kódgépelés, nem lesz számukra elég érdekes ahhoz, hogy továbbiakban is azzal foglalkozzanak. Szerencsére egyre több lehetőség van a kreatív programozás oktatásra. Már csak kísérletező szellemű oktatókra, diákokra, vezetőkre és megfelelő környezetre van szükség ezek közoktatásba történő beillesztéséhez.

Irodalom

1. Takács Valéria: *Tananyagkészítés a Sratch programozási környezethez*, ELTE IK, Budapest (2009) http://grafit.netpositive.hu/download/scratch/szakdolgozat.pdf (utoljára megtekintve: 2018.11.03.)
2. https://pcworld.hu/szoftver/scratch-suli-kezdjunk-el-programozni-175636.html (utoljára megtekintve: 2018.11.03.)
3. https://scratch.mit.edu/about (utoljára megtekintve: 2018.11.03.)
4. Kovács Zsuzsanna: *JAVA programozási nyelv NetBeans fejlesztőkörnyezetben*, Budapest (2009) http://www.petrik.hu/files/tamop/SZINFO13/SZINFO13_TK.pdf (utoljára megtekintve: 2018.11.04.)
5. Tömösközi Péter: *Programozás Javában*, Eger (2013) http://mek.oszk.hu/14200/14282/pdf/14282.pdf (utoljára megtekintve: 2018.11.04.)
6. Horváth László András: *A Java nyelv tanítása középiskolában*, DE IK, Debrecen (2009) https://dea.lib.unideb.hu/dea/bitstream/handle/2437/88910/Szakdolgoat.pdf?sequence=1 (utoljára megtekintve: 2018.11.04.)
7. http://www.inf.u-szeged.hu/~tnemeth/source/faclassjava.txt (utoljára megtekintve: 2018.11.04.)
8. Horváth Győző , Menyhárt László Gábor: *Oktatási környezetek vizsgálata a programozás tanításához* In: Szlávi Péter, Zsakó László (szerk.) INFODIDACT 2014: Informatika Szakmódszertani Konferencia. Konferencia helye, ideje: Zamárdi, Magyarország, 2014.11.20-2014.11.21. Budapest: Webdidaktika Alapítvány, 2014. (ISBN:9789631206272)
9. Németh Tamás, Széll Réka, Tornai Henrietta: *Az algoritmikus gondolkodás fejlesztésének fontossága a közoktatásban* In: Szlávi Péter, Zsakó László (szerk.) INFODIDACT 2017: Informatika Szakmódszertani Konferencia. Konferencia helye, ideje: Zamárdi, Magyarország, 2017.11.23-2017.11.25. Budapest: Webdidaktika Alapítvány, 2017. (ISBN: 978-615-80608-1-3)
10. http://www.inf.u-szeged.hu/~tnemeth/source/faclassjava.txt (utoljára megtekintve: 2018.11.04.)
11. https://codecombat.com/ (utoljára megtekintve: 2018.11.04.)
12. https://html5.ugyesen.com/2013/06/a-html5-inline-svg-hasznalata/ (utoljára megtekintve: 2018.11.04.)
13. https://eloquentjavascript.net/17_canvas.html (utoljára megtekintve: 2018.11.04.)
14. https://schonherzbazis.hu/hirek/reszletek/Node.js-teljesen-kezdoknek (utoljára megtekintve: 2018.11.04.)
15. https://ithub.hu/blog/post/Bevezetes-a-Vuejs-keretrendszerbe/(utoljára megtekintve: 2018.11.04.)
16. https://cli.vuejs.org/ (utoljára megtekintve: 2018.11.04.)

Várható tanári szerepváltozások informatikához kapcsolódó területeken

Örkényi Virág[1], Holló Csaba[2]

[1] viragorkenyi@gmail.com
SZTE TTIK Informatikai Intézet

[2] chollo@inf.u-szeged.hu
SZTE TTIK Informatikai Intézet

Absztrakt. Az informatikai ismeretekre épülő szolgáltatások bővülése olyan társadalmi változásokat eredményez, melyek szükségessé teszik nem csak a technológia, hanem a felhasználói magatartás tanítását is, nem csak ismeret- és készség-, hanem attitűdfejlesztés szinten is. Ezzel szemben, egyes oktatáskutatók olyan véleményeket is megfogalmaznak, hogy megfelelő szervezéssel tanárra nem is lenne szükség. Ezen szélsőségesnek tűnő megállapítás mellett az tény, hogy a multimédiás szolgáltatások, a robotika és a mesterséges intelligencia fejlődésével egyre több tudás átadása legalábbis részben automatizálhatóvá válik a robotok és az egyre bővülő multimédiás digitális tananyagok segítségével. Cikkünkben megvizsgáljuk, hogy egyes informatikához kapcsolódó iskolai szintű ismeretek tanítása várhatóan milyen módon lesz automatizálható, hogyan változhat meg ennek következtében a tanár szerepe, és ily módon melyek azok a kompetenciák, amelyek fejlesztését valószínűleg célszerű lenne az informatikatanári képzésben kiemelten kezelni.

Kulcsszavak: automatizált oktatás, oktatási módszerek, informatikatanári kompetenciák

1. Bevezetés

A jelenlegi Nemzeti Alaptantervhez és kerettantervhez, illetve a 2018-as NAT tervezetéhez igazodva szeretnénk egy lehetséges képet kialakítani arról, hogy mi vár az informatika területen tanító tanárokra, egy olyan világban, amelyben a technológia rohamosan fejlődik. Milyen szempontból lehet automatizálni, segíteni, és milyen módon kell az esetleges történések hatására megváltoztatni a tanárok szerepét, és ehhez igazítva, a leendő tanárok oktatását?

Abból indulunk ki, hogy globális szinten olcsóbb lesz digitális tananyagokat legyártani, és használni minden olyan tevékenységre, amiben a tanár ezzel helyettesíthető. Ennek megfelelően feltételezzük, hogy rendelkezésre fognak állni, az eszközöktől kezdve a szoftveres megoldásokig (például mesterséges intelligencia) a kor minden technikai eredményét felhasználó, tartalmilag és módszertanilag a lehető legjobban összeállított digitális tananyagok, melyek lehetőségeiknek megfelelően igyekeznek motiválni is a diákokat a tananyagok megtanulására.

Áttekintjük a fejlesztési területeket, az azokhoz tartozó tananyagok jelentős részét, és esetenként megvizsgáljuk, hogy jelenlegi ismereteink szerint mely módszerekkel, stratégiákkal lehetne az adott témakört többnyire a legjobban megtanítani a diákoknak, és ezekben a módszerekben melyek lehetnek azok a pontok, amelyeket automatizálni lehet. Nem is törekedünk a teljességre, hiszen a tanítandó anyagmennyiség sokkal részletesebb elemzést indokolna, mint amire egy cikk keretein belül lehetőségünk van, ezért inkább igyekszünk a fontosabb témakörök kapcsán többféle pedagógiai módszert és tanári feladatot megvilágítani, melyeket az oktatás átalakulása szempontjából fontosnak gondolunk. A pedagógiai módszerek választásánál eltekintünk attól a helyzettől, hogy jelenleg a tanárok a tanítandó anyagmennyiség, a rendelkezésre álló idő, illetve technikai korlátok miatt kényte-

lenek sokszor a kevésbé kommunikatív, kevésbé diákközpontú módszereket választani, feltételezzük, hogy a jövőben rendelkezésre fognak állni a módszerek használatának ideális körülményei is.

Az új NAT tervezetben *Informatika* helyett *Digitális technológia és kultúra* tantárgy található, melyben megmaradtak *Az informatikai eszközök használata* és *Problémamegoldás informatikai eszközökkel és módszerekkel* fejlesztési területek, az *Alkalmazói ismeretek* helyett hasonló tartalommal *Digitális írástudás*, az *Infokommunikáció* és *Az információs társadalom* helyett hasonló tartalommal *Információs technológiák* jelent meg, a *Könyvtári informatika* pedig törlésre került, ezért az utóbbi tartalmi egységet nem fogjuk érinteni.

2. Fejlesztési területek

2.1. Az informatikai eszközök használata

Az egyik ismeret a területen belül a számítógép főbb egységei, azok jellemzői, a perifériák, digitalizáló eszközök, és használatuk. Talán a legjobb módszer a témakör átfogó, mélyreható elsajátításához, az elméleti ismeretek átadásán túl, a szemléltetés. Például, ha a tanár a diákokkal együtt összeszerel egy számítógépet, és mindemellett plusz információkat közvetít a tanulók felé, akkor valószínűleg a diákok maradandóbb élményt kapnak, így nagyobb érdeklődéssel sajátítják el a témakörhöz tartozó ismereteket. Érdekes lehet továbbá egy olyan beszélgetés is, mely során azt beszélik meg, hogy ha a diákok gépet szeretnének vásárolni, akkor a paraméterek és ár szempontjából mely gépeket lenne érdemes megvásárolniuk, hiszen ebben érdekeltek lehetnek a diákok, ezért a paraméterek jelentéseire is kíváncsiak lesznek. Mindezen ismereteket digitális tananyagban is le lehet írni, a gép összeszerelését is lehetséges videón bemutatni, illetve ezt a tevékenységet applikációval is lehet modellezni. Viszont, a videó, illetve az applikáció használata nem adja meg azt a *közösségi élményt*, ami a tudást maradandóbbá teheti. Továbbá, amíg a tanár a felmerülő kérdésekre *azonnal megfelelő választ tud adni*, egy applikáció, vagy egy digitális tananyag nem biztos, hogy minden felmerülő kérdésre választ ad, vagy legalábbis könnyű lesz benne a válaszokat megtalálni, az pedig kérdéses, hogy a diákok hajlandóak lesznek-e az ezek megkereséséhez szükséges plusz munkára.

A témakörhöz kapcsolódóan még fel kell hívni a tanulók figyelmét az ergonomikus körülmények megteremtésének fontosságára, valamint arra, hogy felismerjék a digitális eszközök kártékony hatásait. Rengetek tananyag, videó és különböző elektronikus felület van, amely az ezekkel kapcsolatos információkat tartalmazza, de arra nincs garancia, hogy ezt a diákok tudatosan hasznosítanák. Ezzel iskolai keretek között foglalkozni kell, és meg kell próbálni meggyőzni a diákokat arról, hogy ezek az ismeretek és használatuk hasznos. A diákok meggyőzésére, érdemes vitát kezdeményezni, meggyőző érvekkel alátámasztani a tanár nézőpontját, valamint meghallgatni a diákok szemszögéből az adott helyzetet. A vita módszer pozitív következményei közé tartozik, hogy a tanár jobban megismeri a diákok nézeteit, látásmódját, így a tanulnivalót és *az érvelést az ő véleményükhöz tudja igazítani*, személyre szabni. Fontos tudatos felhasználókat nevelnünk, akik a digitális eszközök és az internet használata közben tisztában vannak az elvégzett folyamatok következményeivel, és egy adott tevékenységsorozathoz ki tudják választani a megfelelő lépéseket olyan módon, hogy a hibalehetőségek számát minimalizálják. Az ismeretek lényegében átadhatók tananyagokkal, ám véleményünk szerint a tananyagok feldolgozása *kevésbé meggyőző lehet a közösen lefolytatott kommunikációhoz képest*.

Az eszközökhöz szorosan kapcsolódó fejlesztendő terület még az operációs rendszerek alapszolgáltatásai, mappaműveletek, állománykezelés, felhőszolgáltatások, mobileszközök, számítógépes hálózatok alapszolgáltatásainak használata, a digitális jelek minőségével, kódolásával, továbbításával kapcsolatos problémák kezelése, valamint a technológiai fejlődés nyomon követése. Úgy gondoljuk, hogy a legfontosabb ilyen jellegű ismereteket alapvetően meg lehet tanítani digitális tananyagokkal.

Az eszközök, operációs rendszerek és más szoftverek használata közben fontos a digitális kártevők elleni védekezés, az eszközök, fájlok, hálózatok illetéktelenek által történő hozzáférésének meg-

akadályozása, az adatok biztonságos tárolásának szoftveres és hardveres biztosítása, mely ismereteket úgyszintén meg kell tanítani a diákoknak. Az információk átadása itt is történhet digitális tananyagokkal, azonban a tudatos használatra nevelés ennél sokkal többet jelent, ugyanis személyiségformálásra is szükség van ahhoz, hogy a diákok tényleg használják is ezeket az ismereteket. Természetesen a digitális tananyag is törekedhet a diákok meggyőzésére (például megfelelő példák használatával), de ennek hatása jelentősen növelhető azzal, ha a diákok jó példákat is látnak (például a tanárok személyében), vagy a tanár irányításával *egymás között* a témához kapcsolódó *ismereteket, tapasztalatokat osztanak meg*.

A tömörítési ismeretek használatával kapcsolatosan érdemes megtanítani a diákoknak a tömörítés pozitív hatásait (például egy nagyméretű multimédiás fájl memóriafoglalása), valamint a tömörített fájlok kicsomagolását. A tömörítés és kicsomagolás ismeretanyag tanítása alapvetően elméleti információk átadásából és a gyakoroltatásból áll, mely valószínűleg meglehetősen jó hatásfokkal megvalósítható lesz digitális tananyagokkal is.

Végül, beszélnünk kell arról a kompetenciáról is, hogy a diákok összetett feladatokhoz képesek legyenek informatikai eszközöket választani, és azokat rendeltetésszerűen használni. Az egyes eszközök tanítása során a diákok már megismerkedtek azok céljaival és alkalmazási területeivel, így ennek a kompetenciának a kialakításához inkább a tanultak összesítésére, szintetizálására van szükség. Elméletileg ez is leírható egy digitális tananyagban, ugyanakkor hasznos lehet, ha a diákok tényleg megpróbálják az ismereteiket komplexebb feladatok megoldására felhasználni, közben *eligazítást*, a végén pedig *visszajelzést* kapnak a tanártól arról, hogy ezt mennyire sikerült jól kivitelezniük. Tehát itt a tanár személyes részvétele nem elengedhetetlen, de általa jobb eredmény érhető el.

2.2. Digitális írástudás

Egy másik fejlesztendő terület a digitális írástudás témaköre, melyen belül több részterületen fogunk végighaladni.

A szöveges dokumentumok szerkesztése magában foglalja egyszerűbb dokumentumok (például szórólap, órarend), bonyolultabb dokumentumok (például kérvény, önéletrajz, körlevél) és nagyobb méretű dokumentumok hierarchikusan strukturált részeinek (tartalomjegyzék, hivatkozások, szójegyzék, ábrajegyzék) elkészítését, többletinformációk (akadálymentesítés, feliratozás, lábjegyzet, térképi pontokhoz rendelhető információk) elhelyezését, nem WYSIWYG jellegű szövegszerkesztést, és dokumentumok átalakítását is [1].

Ehhez a témakörhöz tartozik még a grafikus ábrák készítése, a képszerkesztés, a multimédiás és webes dokumentumok létrehozása, illetve szerkesztése. A cél az, hogy a diákok megismerkedjenek a grafikonok helyes használatával, az információk grafikus feldolgozásával, és elsajátítsanak olyan készségeket, melyek ahhoz szükségesek, hogy meglévő információkból azokhoz alkalmas diagramokat, ábrákat tudjanak készíteni, illetve elkészült ábrákat adott szempontok szerint ki tudjanak értékelni. A multimédiás tartalmakkal kapcsolatosan ismerniük kell ezek minőségével és formátumával kapcsolatos legfontosabb információkat, tisztában kell lenniük a jó minőségű felvételekhez szükséges környezeti, valamint eszközbeli követelményekkel, az esetleges hibák javításainak lehetőségeivel, és ily módon képessé kell válniuk videó, hang, és képanyagok megfelelő minőségű rögzítésére.

Minden felsorolt részterület tanulása akkor nyer igazán értelmet, ha konkrét feladatok megoldására használjuk, és mindegyik messzemenően alkalmas is arra, hogy más tantárgyak keretében is gyakorolható és hasznosítható legyen. A diákok képessé kell, hogy váljanak adott feladat megoldásához szükséges szöveges dokumentumok, ábrák, multimédiás elemek gyűjtésére és készítésére, azok összeállítására, és az információs környezetbe történő szakszerű beillesztésére. Ezt, illetve a korszerű kompetenciafejlesztési célokat figyelembe véve minden ide tartozó részterület esetén a problémaorientált megközelítés tűnik leginkább célravezetőbbnek, minél több együttműködést igénylő feladatokon (például iskolaújság készítés, rendezvényszervezés) keresztül. Ennek megfelelően nagyon ajánl-

lott a projektmódszer alkalmazása, melyben a diákoknak önállóan kell egy témakörnek utánajárni, akár kutatómunkát végezni, és azt kisebb csoportokban feldolgozni, majd az osztály előtt, vagy beadandó formájában bemutatni. A projektmódszert természetesen megelőzi valamilyen szintű ismerettanítás, melyhez a szemléltetést használjuk, amelyben végigvesszük lépésről lépésre a tananyag fontosabb elemeit, ezáltal rendszerezett ismereteket is átadunk, és mindemellett biztosítjuk a gyakorlási lehetőséget arra, hogy az ismereteket alkalmazni tudják, melyre megfelelő a projekt, vagy tanulói kiselőadás. Jó megoldásnak gondoljuk az elkészült munkák tanulókkal közösen történő értékelését előre kiadott szempontok szerint, ahol a csoport minden tagjának tevékenykednie kell ahhoz, hogy társaik elismerjék a munkájukat.

Ami az automatizálást illeti, egy tananyag be tudja mutatni az elméleti tudnivalókat, és akár életszerű feladatokon keresztül is el tudja magyarázni egyes funkciók működését, tartalmazhat példákat majdnem bármire, de nehezen tud választ adni a tanulókban aktuálisan felmerülő „hogyan lehet azt úgy megcsinálni, hogy …" jellegű kérdésekre. Kizárólag tananyag birtokában a tanulók a választ legfeljebb az előre elkészített példákból tudnák kigyűjteni, ami az általuk megoldandó feladathoz képest *jelentős plusz munkát igényelhet*, és ennek következtében a diákok *motivációjának csökkenését* is eredményezheti. A tananyag továbbá nem tudna segíteni *az együttműködés bizonyos kérdéseiben* (például konfliktusok kezelésében), és nem tudná *értékelni* az elkészült munkát.

2.3. Problémamegoldás informatikai eszközökkel

Ebbe a témakörbe tartozik az algoritmizálás, programozás, adatkezelés táblázatkezelő és adatbáziskezelő alkalmazásokkal, és a számítógépes szimuláció.

Az algoritmikus gondolkodás fejlesztésénél fontos, hogy a pedagógus ráébressze a diákságot arra, hogy az életükben rengeteg dolgot algoritmikusan végeznek úgy, hogy erről sok esetben nincs is tudomásuk. Ennek érdekében szükséges, hogy a tanár életszerű példákkal mutassa be az algoritmikus gondolkodást és annak elemeit, valamint azt, hogy miként írhatjuk le ezeket a lépéseket, mely ismereteket a diákok később a programozás alapjainak elsajátításakor már programok írásához is fel fognak tudni használni. A konkrét példákkal szemléltetett magyarázat és ismeretátadás után hasznos lehet, ha a tanulók is keresnek hasonló, életből kiragadott helyzeteket, és munkáltatással elkészíttethetjük velük egy általuk sokszor végzett, vagy legalábbis jól ismert tevékenység algoritmusát. A felismerés segítéséhez és a diákok munkájának értékeléséhez mindenképpen szükséges a pedagógus. Természetesen digitális tananyagban is számos életszerű példát le lehet írni, de ezek valószínűleg kevésbé lehetnek hatékonyak, mint ha *a tanár a diák saját gondolatmenetére reagálva segíti a megértést*. A folyamat jól automatizálható része lehet viszont az algoritmusok formális elemeinek ismertetése, valamint bizonyos típusalgoritmusok megismerése és azok előre megadott módon történő gyakoroltatása [6].

Ehhez a témakörhöz tartoznak a számítógép által használt egyszerű adattípusok, és azok közötti különbségek is. Ezt az anyagrészt magyarázattal célszerű ismertetni, melynek foganatja akkor lesz, amikor a diákoknak ezeket programokban kell tudniuk felhasználni. Szükséges, hogy programnyelv specifikusan is végignézzük azt, hogy milyen módon kell használni az adott adattípusokat, és hogy a számítógépen ezek hogyan vannak reprezentálva. Ezeket az ismereteket oktatási tananyagokból is el lehet sajátítani. Valamivel bonyolultabb a kérdés az összetett adattípusok esetén, különösen, hogyha azokból többfélét tanítunk, hiszen a helyes adattípus kiválasztása már egy komplex megközelítést igényel, melynek elveit, példákkal együtt, le lehet írni a tananyagban, de a megfelelő használatához olyan jellegű gyakorlás szükséges, amelyben nagyon hasznos lehet *a tanárnak a diák megoldásához igazított visszajelzése*.

Ezen a területen belül tanítandó a programozás alapjainak elsajátítása, fejlesztőkörnyezetek használata is. Ebben építeni lehet az elsajátított algoritmikus ismeretekre, azok programozási felhasználására. A diákok a programozás elsajátítása közben képessé válnak fejlesztői környezeteket (IDE-ket) használni, valamint ezeken belül fordítani, futtatni, és fordítási hibákat javítani. A prog-

ramozás elméleti tudásra épít, hiszen egy feladat megoldásához különböző matematikai, informatikai ismeretekre van szükségünk, valamint algoritmikus látásmódra, és problémamegoldó képességre. Úgy gondoljuk, hogy a tananyagot magyarázattal egybekötött szemléltetéssel, valamint munkáltatással lehet a legjobban megtanítani. Először megtanítjuk az adott programozási nyelv sajátosságait, majd egyszerűbb példákon keresztül átvesszük az alapokat, ezek után pedig feladatokat oldunk meg, először közösen, majd további feladatokat osztunk ki, melyeket a tanulóknak önállóan, vagy csoportban kell megoldaniuk. Ehhez kapcsolódóan beszélnünk kell még a LEGO robotok használatáról. A LEGO robotok használatával a gyerekek játékosan fejlesztik programozási készségeiket, illetve szemléletesen láthatják a programozás eredményét. A robotok programozása fejleszti a gyerekek kreativitását, valamint olyan fejlesztői környezetet biztosít, melyben a diákok grafikus elemek felhasználásával könnyen tudnak programozni. A programozáshoz külön korcsoportokra lebontva használhatunk különböző robotokat. A LEGO robotok tanmenetbe illesztéséhez már oktatási tananyagok is elérhetők. Automatizálhatóság szempontjából egy tananyag (vagy robot) képes lehet arra, hogy az elméleti ismereteket bemutassa és előre megadott módon elmagyarázza, példákat is szemléltessen, de arra már valószínűleg csak korlátosan, hogy *a diákok sajátos kérdéseire válaszoljon*.

A következő vizsgált téma az adatkezelés. Ennek első részeként a táblázatkezelést tárgyaljuk, melyet gimnáziumban táblázatkezelő alkalmazásokkal tanítunk. A diákoknak meg kell tanulniuk az alkalmazás használatát, adatokat táblázatba rendezni, azokon műveleteket, függvényeket alkalmazni, valamint ezekből az adatokból grafikonokat, ábrákat létrehozni. Adatok rendszerezésére még adatbázisokat is használhatunk, melyek logikailag, valamint tartalmilag egységbe zárják az adatokat. Az ismeretelsajátítás során a tanulók megtanulják, hogyan kell adatbázist létrehozni, mi az adattábla, rekord, mező, valamint, hogy milyen kapcsolatok lehetnek az adattáblák között. Ezen témakörhöz tartoznak még az adattábla kulcsok, a külső forrásból történő adatbázis importálás, illetve az adattáblákon végezhető különböző szűrések. Ebben a témakörben célszerű végigvenni az adatbáziskezelő alkalmazás funkcióit, valamint példákat áttekinteni a különböző szűrésekhez kapcsolódóan. Az adatbázisok alkalmazhatóságánál tanítandó ismeretként felmerülnek az útvonalkereső applikációk is, melyek különböző adatbázisokat használnak helyadatok tárolásának céljából.

Egy digitális tananyag alkalmas lehet arra, hogy absztrakt eszköz orientált, fogalomorientált, menüorientált, vagy funkcióorientált módszerek bármelyikével megtanítsa a fogalmakat, a szoftverek használatát, illetve az ismeretek érdekesebb vagy fontosabb alkalmazásait, viszont a diákok számára valószínűleg legközelebb álló és leghasznosabb problémaorientált tanítási módszerre már valószínűleg csak korlátozottan lehet alkalmas a más esetekben is említett diákokhoz való alkalmazkodás nehézségei miatt.

Végül, a számítógépes szimulációkat illetően, egy digitális tananyag sok érdekes szimulációt mutathat be, ezeket megfelelő leírással és felhasználóbarát megvalósítással a diákok önállóan tanulmányozhatják és megérthetik, ezért úgy gondoljuk, hogy ennek a témakörnek a tanítása teljes mértékben automatizálható, vagy más tantárgyakhoz delegálható lehet.

2.4. Információs technológiák

Az információs technológiák témakörébe tartozik az információk keresése, azok etikus felhasználása, az online kommunikáció, annak normái, mobiltechnológiai ismeretek, illetve az e-Világ ismerete.

Kezdjük az információkereséssel. Az olyan információk kereséséhez, melyek valószínűleg valamilyen tematikus oldalakon találhatók meg könnyebben, a diákoknak ismerniük kell néhány ilyen oldalt, illetve rendelkezniük kell iránymutatásokkal arra vonatkozóan, hogy milyen oldalakon próbálkozzanak, és adott esetben – hogyha az például egy könyvtár vagy adatbázis oldala -, akkor tudniuk kell kitölteni az ott felkínált adatlapokat. A keresőprogramok használatához fontos a keresőkérdések megfelelő megfogalmazása, az információk szűrése keresőoperátorok segítségével, a források vizsgálata hitelesség, az információk elemzése pedig megbízhatóság és minőség szempontjából. A diákoknak tisztában kell lenniük azzal, hogy a keresőmotorok nem feltétlen relevancia és megbíz-

hatóság szerint rendezik a találatokat, ismerniük kell azokat a technikákat, amelyekkel valószínűsíthetik az információk megbízhatóságát. Végül, a diákoknak vizsgálniuk kell a megtalált információk felhasználhatóságának feltételeit is, és ismerniük kell az etikus forrásmegjelölés szabályait, melyek nem csak a digitális, hanem a nyomtatott dokumentumokra is vonatkoznak. Érdemes példákat mutatni helyes és helytelen forrásmegjelölésre, valamint hangsúlyozni az etikátlan forrásmegjelölés jogi következményeit. Módszertani szempontból, hogyha arra törekszünk, hogy mindezeket a diákok ne csak megtanulják, de a későbbiekben alkalmazzák is, akkor ennek fontosságáról meg is kell győznünk őket, amihez hasznos lehet, ha konkrét példákon megtapasztalják, hogy az általuk megtalált és igaznak gondolt információ lehet hamis, illetve példákat mutatunk az etikátlan vagy jogellenes információ felhasználás következményeire. A tanultak gyakorlására adhatunk olyan feladatot a tanulóknak, hogy valamilyen témában többen gyűjtsenek információkat (például olyanban, amiből tudjuk, hogy az első találatok egy része hamis lesz), készítsenek azokból egy közös beadandó dolgozatot vagy kiselőadást (ezen a ponton ki kellene derülnie az ellentmondásoknak, így esetleges további kutatásokkal elemezniük kell az információk helyességét is), a felhasznált források megfelelő feltűntetésével. Érdemes végül az elkészült munkákat közösen kiértékelni. Az információkereséshez hasznos oldalak, keresési technikák tanítása, valamint az etikus forrásmegjelölés tanítása kivitelezhető oktatási tananyagokkal vagy robotokkal, részben az információk hitelességi és megbízhatósági vizsgálatának tanítása is, bár utóbbi esetben *a logikai ellentmondások felismerésére és tanítására* valószínűleg a tanár alkalmasabb, ahogyan arra is, hogy olyan *aktuális* példákat találjon, amelyekre éppen akkor keresve ellentmondásos találatok lesznek.

A diákok nagy része már kisebb kortól kezdve napi szinten használ mobiltelefont, ily módon rendelkezik alapvető felhasználói ismeretekkel, viszont egyrészt az iskola arról szól, hogy a tanítandó ismeretekkel nem csak a diákok egy részének, hanem mindenkinek rendelkeznie kell, másrészt a diákok ismeretei különbözőek és hiányosak lehetnek, ezért a mobiltechnológiai ismeretek tanítása nem kerülhető ki. Ezen ismeretek tanítási módszertana hasonló az informatikai eszközök tanításához, ezért az oktatás automatizálásának szempontjából is az ott leírtak érvényesek.

Az online kommunikációra vonatkozóan tanítani kell annak technikai lehetőségeit, előnyeit, veszélyeit, és védekezési lehetőségeit. Napjainkban rengeteg online kommunikációs program létezik, melyek lehetővé tesznek páros vagy csoportos, írott, hanghívásos, vagy videóhívásos beszélgetéseket, írott vagy képi információk megosztását, esetlegesen sajátos szabályozásokkal, korlátozásokkal. Fontos a diákokkal megértetni, hogy ezek a programok eszközök, melyeket lehet nagyon hasznosan alkalmazni, ugyanakkor nem megfelelő használatuk komoly veszélyeket jelenthet. Feltételezzük, hogy egy modern oktatási rendszerben a diákok egyre gyakrabban fognak találkozni a kommunikációs programok és eszközök hasznos alkalmazásaival az oktatásban ([2, 4]), amely helyzetekben sokszor szükség van a tanárra, mint a *szervezőre*, aki az *adott körülményekhez, adott diákokhoz igazítva*, a használat közben kialakult *helyzetekre reagálva* irányítja az oktatást. Mindez természetesen érvényes az informatika tananyagok oktatására is, beleértve a kommunikációs programok és eszközök oktatását. Az online kommunikációs lehetőségek használatával kapcsolatosan talán a legnehezebb feladat a diákok *nevelése* ezek etikus és biztonságos használatára (az adatvédelemtől az elektronikus kommunikáció együttműködési szabályaiig), ugyanis ez esetben nem elegendő, ha a diák tudja (és egy számonkérésnél képes elmondani), hogy hogyan *kellene* ezeket alkalmazni, hanem nagyon fontos, hogy *meggyőzzük* arról, hogy ezeket akkor is a tanultaknak megfelelően használja, amikor azt a tanár nem látja. Nyilvánvaló, hogy az etikai, jogi és biztonsági ismereteket online tananyaggal is meg lehet tanítani, és valamilyen szinten meggyőzni is lehet (érvekkel, tanulságos videókkal), ugyanakkor *a beszélgetés, a résztvevők tapasztalatai, a tanár diákok gondolataira reflektáló érvei valószínűleg ebben további jelentős hozzáadott értéket tudnak képviselni* [11].

Végül, az e-világ tanítása magában foglalja az e-állampolgársági ismereteket, e-szolgáltatá-sokat, e-ügyintézéseket, és e-kereskedelmet. Ebben az esetben is, a szolgáltatások elméleti és technikai ismereteinek megtanításán túl, nevelő tevékenységet is kell folytatni annak érdekében, hogy a diákok

ezeket etikusan és biztonságosan használják. Rá kell mutatni az egyes szolgáltatások tipikus veszélyeire, és meg kell győzni a diákokat arról, hogy használják a rendelkezésre álló védekezési lehetőségeket. Módszertani szempontból ugyanolyan megközelítések és megfontolások érvényesek, mint az online kommunikáció tanításánál, így az oktatás automatizálásának lehetőségei is hasonlóak.

3. Következtetések

Először foglaljuk össze, hogy véleményünk szerint mit várhatunk általában az oktatásban az online tananyagoktól, a robotizálástól és a mesterséges intelligencia fejlődésétől?

Egy módszertanilag jól összeállított digitális tananyag, illetve egy képzett robot lehet nagyon informatív, videókkal, interaktivitással, és egyebekkel (például játékosítással) kiegészítve nagyon sok ismeret akár élvezetes megtanítását is lehetővé teheti. Képes lehet arra is, hogy figyelje, hogy a diákok a tananyaggal foglalkoznak-e, olyan jellegű tevékenységet végeznek-e, mint amit a számukra kiadott feladat előír ([5,9,10]), a diákok minden tevékenységét naplózza, és ezáltal sokkal pontosabb visszajelzést adjon arról, hogy a diákok mit értettek meg nehezebben, mint amit a tanár a hagyományos óratartás alkalmával észlelni tudna. Tehát, vannak olyan oktatási feladatok, melyeknek elvégzésére ezek az eszközök a tanárral egyenértékűen vagy akár a tanárnál is jobban képesek, és jelenleg úgy tűnik, hogy ezek közül is kevés olyan tevékenység van, amelyhez feltétlen osztálytermi megjelenésre lenne szükség. Ily módon az ismeretek megtanításának egy része a jövőben rábízható lehet otthon megtanulható online tananyagokra, ami az iskolai óraszán csökkenését eredményezheti, melynek mértéke tantárgyanként nagyon eltérő lehet. Megjegyzendő, hogy az még külön kidolgozandó, hogy hogyan lehet a diákokat rávenni arra, hogy a házi feladatként kiadott tananyagokat megtanulják, és ez mennyiben szaktanári, illetve általános pedagógiai feladat.

Az előző fejezetekből azt is láthatjuk, hogy vannak olyan tevékenységek is, amelyekhez a tanár nélkülözhetetlen, vagy legalábbis jobb eredmény várható tőle. Tekintsük át most összegezve a legfontosabb ilyen feladatokat!

Először is, a tanár mutatott értékrendje és viselkedése mintaként szolgálhat a diákok számára, különösen olyan helyzetekben, amikor nevelő tevékenységet kell kifejtenie. Az informatika esetében ilyen jellegű tevékenység a diákok meggyőzése arról, hogy a tanultakat akkor is alkalmazzák, amikor ezt a tanár nem tudja ellenőrizni. A meggyőzés hatékonyságát pedig jelentősen növelheti az, ha az ilyen témákat közösen megbeszélik, megvitatják, a résztvevők elmondják tapasztalataikat, melyek sokkal életszerűbben hathatnak, mint a tananyagban ismeretlenekről bemutatott példák, a tanár pedig a diákok aggályaira, ellenérveire reflektálhat, szemben a digitális tananyaggal, amelyben csak egy általános, előre megadott érvsorozat lehet leírva. Ugyanakkor, a tananyag közös megbeszélése olyan esetekben különösen hozzájárul a tanultak maradandóbb megjegyzéséhez, amikor ez közösségi élményként a diákok érzelmeire is hatással van [7,8]. Hasznos lehet továbbá közösen megbeszélni és értékelni a diákok által elkészített feladatokat is, egyrészt, mert tanulhatnak egymás hibáiból, másrészt, mert az, hogy a diák össze tudja mérni a tudását a többiekével, ösztönzőleg hathat a teljesítményére.

A digitális tananyagokkal szemben a tanár azonnal választ tud adni a tananyaggal kapcsolatosan a tanulókban aktuálisan felmerülő kérdésekre, melyekre a válaszokat bizonyos esetekben a tanulók vagy csak jelentős plusz munkával találnák meg, vagy ha megtalálnák is, nem biztos, hogy megértenék. Továbbá, vannak olyan ismeretek (például algoritmusok és programozás témakörében), amelyeket véleményünk szerint problémaorientáltan célszerű tanítani, vagy még egy jó minőségű digitális tananyagból is nehéz lehet megérteni, illetve a megértéshez szükség van a tanulók által készített munkák olyan értékelésére, melynek keretében a diák *a saját megoldásához, gondolatmenetéhez illeszkedő* visszajelzéseket, tanácsokat és válaszokat kap.

A digitális tananyagokkal szemben a tanár olyan ismeretek tanításában is hozzáadott értéket képviselhet, amikor bizonyos tapasztalati készségek átadásáról van szó (mint például a hamis hírek felismerése), illetve a tananyagnak napra pontosan aktuálisnak kell lennie (mint például az internetes kereséshez használt példák), hiszen egy digitális tananyag aligha fog (néhány) naponta frissülni.

Végül, de nem utolsó sorban a tanárra szükség van úgy is, mint szakmai szervezőre, aki a tananyaghoz, az adott körülményekhez, adott diákokhoz igazítva, a használat közben kialakult helyzetekre reagálva irányítja az oktatást. Különösen fontos ez olyan pedagógiai módszerek alkalmazásánál, melyek szervezést igényelnek, hiszen jelenleg nehéz elképzelni, hogy akár egy robot (vagy digitális tananyag) megfelelően elvégezné a tanulók szervezését, és segítene az együttműködés során kialakuló problémák, konfliktusok megoldásában.

A fentiekből azt valószínűsítjük, hogy az informatika területén, ugyan valamivel kevesebb óraszámban, de tanár által tartott órákra akkor is szükség lesz, ha rendelkezésre fognak állni nagyon jó minőségű digitális tananyagok és tanító robotok. A tanár szerepe viszont valószínűleg megváltozik, tárgyi ismeretek helyett tapasztalati, megértést segítő ismereteket fog átadni, szerepe inkább az oktatási folyamat megfelelő megszervezése, az érdeklődés felkeltése, az ismeretátadás élményszerűvé tétele, a motiválás, a megértés segítése, és a diákok gondolkodásának és viselkedésének fejlesztése lesz. Természetesen ehhez a tanárnak is rendelkeznie kell kritikus gondolkodásmóddal, és képesnek kell lennie alkalmazkodni a folyamatosan változó oktatási feltételekhez és lehetőségekhez.

Mi következik mindebből? Úgy gondoljuk, hogy egyrészt, fontos lenne az informatika módszertan terén intenzív kutatásokat folytatni abban az irányban, hogy hogyan lehet a különböző témakörökhöz tartozó ismereteket érdekesebben, élményszerűen, különböző (például együttműködést fejlesztő) módszerekkel tanítani, és - különösen az információs technológiák kihívásaira való tekintettel – a diákok gondolkodását és viselkedését megfelelően fejleszteni. Másrészt, ezekre támaszkodva, a tanárok képzésében *sokkal* nagyobb szerepet kellene kapnia a szakmódszertan, kommunikáció (különösen magyarázás, érvelés, előadásmód, konfliktuskezelés), a szervezés, és az információs technológiák (és azok használati lehetőségeinek) oktatásának.

Irodalom

1. Szlávi Péter, Zsakó László: *Informatika oktatása: Alkalmazói feladatok megoldása*, https://people.inf.elte.hu/szlavi/TAMOP-2/EgybenGeneralva/lecke5_lap4.html#hiv2 (utoljára megtekintve: 2018.10.27.)
2. Dr. Abonyi-Tóth Andor, Dr. Turcsányi-Szabó Márta: *A digitális írástudás fejlesztésének lehetőségei*, Educatio Társadalmi Szolgáltató Nonprofit Kft., Digitális Pedagógiai Osztály, IKT Módszertani Iroda, 2015, http://dl-sulinet.educatio.hu/download/letoltheto-dokumentumok/Digitalis-irastudas.pdf, (utoljára megtekintve: 2018.11.01.)
3. Megan Poore: *Hogyan használjuk a közösségi médiát az oktatásban?*, Wolters Kluwer, Budapest, 2015.
4. Dr. Abonyi-Tóth Andor, Dr. Turcsányi-Szabó Márta: *A mobiltechnológiával támogatott tanulás és tanítás módszerei*, Educatio Társadalmi Szolgáltató Nonprofit Kft., Digitális Pedagógiai Osztály, IKT Módszertani Iroda, 2015, https://www.educatio.hu/pub_bin/download/tamop311_II/eredmenyek/m_learning/mlearning_kotet.pdf (utoljára megtekintve: 2018.11.01.)
5. Natasha Smerling: Why teachers will never be replaced by robot, Study International, 2017 október 16., https://www.studyinternational.com/news/robot-teachers/ (utoljára megtekintve: 2019.02.03.)
6. Zoltán Kátai, Erika Osztián, Géza Károly Vekov: *Promoting computational thinking by artistically enhanced algorithm visualization*, SAPIENTIA University, INFODIDACT 2016, Zamárdi.
7. Vass Vilmos: *Személyre szabott tanulást támogató tanári kompetenciák*, VI. nemzetközi Inkluzív iskola, inkluzív társadalom tudományos szimpózium tanulmánykötete, 2017. november 8–9-én Komárom.

8. Dr. Freund Tamás: *Tanulási folyamatok és belső világunk*, Magyar Szemle, Új folyam XV. 11-12. szám, 2017. november 20.
9. NCTEFL India: *Human Teachers Vs Robot Teachers: Who Are The Best For The Changing Times?*, 2018. május 9., https://medium.com/@NcTeflIndia/human-teachers-vs-robot-teachers-who-are-the-best-for-the-changing-times-f9368b5796aa (utoljára megtekintve: 2019.02.03.)
10. Lina Narbutaité, Robertas Damasevicius, Egidijus Kazanavicius, Sanjay Misra: *Using Collaborative Robotics as a Way to Engage Students*, Towards Extensible and Adaptable Methods in Computing, pp.385-397, DOI: 10.1007/978-981-13-2348-5_29, 2018 július.
11. Holló Csaba: *Álprofilok használata az etikus és biztonságos internethasználat tanításában*, INFODIDACT 2018.

Tesztelési módszerek webes tárgyak tanításában

Horváth Győző[1], Visnovitz Márton[2]

[1]horvath.gyozo@inf.elte.hu
[2]visnovitz.marton@inf.elte.hu
ELTE IK

Absztrakt. A tesztelés fontos részét képezi a módszeres feladatmegoldás lépéseinek. Ahogy az iparban is, úgy a programozásoktatásban is egyre hangsúlyosabban jelenik meg ez a témakör kezdve már a bevezető kurzusoktól. Az ELTE Informatikai Karának új képzési tervének keretében megújuló webes tárgyak vonatkozásában is szeretnénk a tesztelést markánsabban megjeleníteni. Ebben a cikkben azt járjuk körül, hogy az iparban ezen a területen használatos eszközök és módszerek közül mi és hogyan jelenhet meg az oktatásban úgy, hogy minél kisebb energiabefektetéssel minél jobban lássák a hallgatók a tesztelés előnyeit.

Kulcsszavak: programozásoktatás, tesztelés, webfejlesztés

1. Bevezetés

Az egyetemi programozásoktatásban a feladatmegoldást módszeres lépésekre bontjuk annak érdekében, hogy a tanulók segítő eszközt kapjanak tetszőleges számítógépes probléma megértéséhez, feldolgozásához és sikeres teljesítéséhez. Ezeknek a módszeres lépéseknek egyik fontos eleme a tesztelés. A feladat meghatározása (specifikáció, követelményspecifikáció), tervezése (algoritmusok, modellek) után implementált program helyességéről a tesztelés révén győződhetnek meg a hallgatók. Többek között azért lehet fontos ez számukra, hiszen elkészült munkáik helyessége általában jelentősen befolyásolhatja érdemjegyeiket. Hosszabb távlatokban gondolkodva az egész informatikai iparág számára fontos, hogy olyan szakemberek kerüljenek ki a felsőoktatási intézményekből, akik egy alkalmazás teljes életciklusát képesek végig követni, és minőségi munkát tudnak végezni a határidők betartásával. Ennek részét képezi – a megfelelő tervezési lépések és az implementációs technológia magabiztos ismerete mellett – a tesztelés is.

Az ELTE programtervező informatikus képzésében a tesztelés a kezdő, bevezető programozási kurzusoktól jelen van különböző formákban. Az első féléves programozás (korábban programozási alapismeretek) tárgyban a tesztelés témakör különböző formákban és eltérő hangsúlyokkal van jelen. Az előadásban külön hetet szánnak a tesztelés, hibakeresés, hibajavítás témakörének, ahol a hallgatók megismerik az elméleti tudnivalókat [1]. Gyakorlatokon azonban a módszeres tesztelés sokszor kisebb hangsúlyt kap a többi ismeret mellett. A hallgatóknak ebben a fázisban sokféle új ismerettel kell megküzdeniük: a specifikációs nyelvvel, az algoritmusleíró nyelvvel, a programozási nyelvvel. Mire a feladatmegoldásban idáig érnek, a módszeres tesztelésre sokszor se idő, se energia nem marad. Ha a tesztesetek átbeszélésre kerülnek is, a tesztelés manuális volta akadályozza ezt a lépést. Ezek mellett a hallgatók hangsúlyosan a számonkérésben találkoznak a teszteléssel, hiszen ebben az esetben egy értékelő rendszer az elkészült programjaikat input-output tesztelésnek veti alá, azaz az egyes bemenetekhez tartozó helyes kimeneteket veti össze a hallgató programja által generált kimenetekkel. Ez a megközelítés a hallgató programjának egészét tekinti egy inputról outputra képező tiszta függvénynek. A kezdő tárgyra épülő objektumelvű programozás (korábban programozás) tárgyban megjelenik a függvények egység- és integrációs tesztelése is [2]. A hallgatók további elméleti instrukciókat kapnak a tesztesetek előállítására, és egy egyszerű eszközön keresztül az automatikus tesztelést is elvárják tőlük.

Ilyen alapokkal érkeznek a hallgatók az Informatikai Kar megújult BSc képzésének egyik szakirányában kötelezően helyet kapó webprogramozás tárgyakra. Ezek közül az egyik a webprogramozás alapjaival foglalkozik, bemutatva a kliens- és szerveroldali dinamikus weboldalak készítéséhez szükséges technológiák alapjait. A későbbi tárgyak erre az alapra építve ismertetik meg a hallgatókkal a modern kliens- és szerveroldali alkalmazások elveit, eszközkészletét, technológiáit, ami végül egy integráló tárgy keretében zárul le, ahol egy full-stack alkalmazás fejlesztésével tehetnek tanúbizonyságot a hallgatók mindkét oldalon szerzett ismereteikről.

Ugyan a képzés során a hallgatók dedikált tárgy keretében találkoznak a modern szoftverfejlesztési eszközökkel és módszertanokkal, azonban ahogy a fentebb felsorolt programozási tárgyak esetében is megjelentek ezek az ismeretek megfelelő mértékben, úgy szeretnénk, ha az új webes tárgyak oktatásának is szerves részét képezné a tesztelés témaköre. Ennek a cikknek az a célja, hogy bemutassa, milyen megoldásokat használnak az iparágban webes alkalmazások teszteléséhez, és ezek hogyan jelenhetnek meg az egyes tárgyak oktatásában.

2. Professzionális tesztelési megoldások

Szoftverek tesztelésének rengeteg megközelítése, típusa és technikája van, ezeknek a részletes ismertetésére ez a cikk nem vállalkozhat. A tesztelést legtöbbször az *alkalmazás tesztelendő szintje* szerint szokták kategorizálni. Ezeket is figyelembe véve a helyes alkalmazás készítését a következő tesztek biztosítják:

1. *Statikus kódelemezés* használata: a megfelelő eszközök segítségével kiszűrhetők a szintaktikus hibák, az elírások és a rossz típusokból fakadó hibák.
2. *Egységtesztelés*: az alkalmazás önálló, izolált részei helyes működésének szeparált ellenőrzése. A tesztelés legelterjedtebb, leggyakrabban használt formája. Egységtesztből nagyon sok lehet, ezért nagyon fontos, hogy gyorsan lefussanak. A tesztelendő kis egységek általában függvények vagy osztályok. Az elkülönült viselkedés ellenőrzése érdekében az a jó, ha a függvények tiszta függvények és az osztályoknak nincsen külső függősége. Ha mégis megjelenik ilyen függőség, akkor azokat helyettesíteni kell. Bevált gyakorlat osztályok esetében, hogy direkt függőségek kialakítása (pl. öröklés vagy adattagként megjelenő példány) helyett, a függőségeket paraméterként tudjuk az osztály számára átadni (kompozíció).
3. *Integrációs tesztelés*: annak ellenőrzése, hogy az előzőekben tesztelt kis egységek jól működnek együtt. Itt már nem cél a függőségek helyettesítése, sőt éppen ezek együttes eredményét szeretnénk vizsgálni. Az integrációs tesztek általában szintén gyorsak, ha azonban az integrációba hálózati forgalmat, adatbázis működést is tesztelnek, akkor ezek a tesztek már lassabbak lehetnek. Az integrációs tesztek írásához sokféle megközelítést lehet alkalmazni, ezek közül emeljünk ki kettőt, amely az oktatásban is megjelenhet:
   a) *lentről felfele tesztelés*: itt először a legalsó szinteken lévő komponensek helyes működéséről győződnek meg, majd felhasználják ezeket a magasabb szinten lévő komponensek tesztelésére;
   b) *fentről lefele tesztelés*: az előző fordítottja, először a legmagasabb szinten lévő komponens helyes működéséről győződnek meg, majd azokat a komponenseket ellenőrzik, amelyektől az előző komponens függ. Hasonló megközelítést alkalmaztak a programozási tételek tesztelésénél [2].
4. *Funkcionális tesztelés* (end-to-end tesztelés): felhasználói tevékenységeket ellenőrző folyamat, azt vizsgálja, hogy az alkalmazás helyesen működik-e a felhasználó szempontjából. Tekinthetjük a legmagasabb szintű integrációs tesztnek, hiszen ebben az esetben az alkalmazás egésze kerül vizsgálat alá. Jellegéből fakadóan nagyon lassú a többi teszthez képest, ugyanakkor felderíthet olyan hibákat, amelyeket az előzőek nem képesek. Míg az egységteszteket fejlesztés közben folyamatosan futtatják, addig a funkcionális tesztek általában a publikálás előtt futnak.

5. *Hatékonysági tesztek*: mennyire hibatűrő az alkalmazás, hogyan viselkedik terhelés alatt.

Sokféle *tesztelési módszertan* terjedt el az utóbbi időben. A klasszikus *vízesés modell* szerint a tesztelést a fejlesztés után végzik el. Az agilis módszertanok legtöbbje a *tesztvezérelt fejlesztést* propagálja több ok miatt is: egyrészt jobban megfogalmazható a tesztelendő funkció publikus interfésze használat oldalról, másrészt csak az kerül lefejlesztésre, amire szükség is van, végül sokkal nagyobb tesztlefedettség érhető el ezáltal, sok teszt pedig az alkalmazás helyes működését biztosítja.

Az *eszközök*et illetően már konkrétan a webes megoldásokat kell vizsgálnunk. Két nagy területe van a webprogramozásnak: a kliensoldali és a szerveroldali programozás. Ezt a két területet érdemes külön vizsgálni, mert nagyon eltérő megközelítésűek lehetnek. A *szerveroldali technológiák* és programtervezési minták történeti okok miatt sokkal érettebbek, így tesztelésük is kiforrottabb. Nyelvtől függően léteznek statikus kódelemzők és xUnit alapú könyvtárak. Terheléses tesztek és HTTP kérések tesztelése is könnyen lehetséges.

A *kliensoldal* mindig is gyorsabban változó és így ingoványosabb terület volt. Tesztelés szempontjából ez nemcsak a technológia miatt alakult így, hanem azért is, mert a felhasználóifelület-tesztelés eleve bonyolultabb egy backend tesztelésnél. A JavaScript nyelvhez léteznek *statikus kódelemző eszközök* (pl. ESLint), dinamikusan típusos nyelv lévén azonban a típushibákat ez felderíteni nem tudja. Éppen ezért sokan ajánlják a JavaScript nyelv típusos kiegészítéseit, ilyen a TypeScript és a Flow. Az *egységtesztelés* régebben kizárólag böngészőben történt, az egyetlen olyan platformként, ahol JavaScript fut. A Node.js projekt megjelenésével azonban a kód tesztelése parancssorban is elérhetővé vált. Sokat lendített a JavaScript egységtesztelésén az is, hogy a nyelv ma már támogatja a modulokat, így a kódot funkcionális egységekre bonthatjuk. Sajnos a modulkezelés egyelőre nem egységes a böngészőben és a parancssorban, így egy köztes átalakítási fázis szükséges, amely bonyolítja a használt eszközkészletet.

Az utóbbi időben a felhasználói felület kialakítására az ún. komponensalapú fejlesztés terjedt el. Ennek használatával a *felhasználói felület tesztelése* is egyszerűbbé vált. A komponensek ugyanis úgy viselkednek, mint a tiszta függvények: a kapott adathoz HTML-t rendelnek hozzá. Tesztelésükhöz böngésző sem kell, elég csak a generált struktúrát valamilyen könnyen kezelhető adatszerkezetben tárolni, és csak ezt kell ellenőrizni. Az utóbbi időben a felületek ellenőrzésére elterjedt az ún. pillanatkép tesztelés, ahol korábban generált struktúrát vetnek össze az újabb futásokkor.

Funkcionális tesztelés során az alkalmazást különböző operációs rendszerek különböző böngészőiben kell futtatni. Ennek automatizálásához sokszor egy bonyolult szerver-kliens rendszer kiépítésére van szükség, ahol a szerver az egyes klienseken elinduló böngészőket távolról vezérli. Ez korábban szinte kizárólagosan a Selenium Webdriver segítségével történt. Az utóbbi időben ezt több alternatíva váltotta fel. A gyors funkcionális tesztelés érdekében egyes megoldások szimulált böngészőkörnyezettel dolgoznak (jsdom), mások meglévő böngészők grafikus felület nélküli (headless) változatát használják tesztelésre (puppeteer), megint mások a tesztelendő kódot a vizsgálandó oldal kontextusába injektálják bele és ott futtatják (TestCafe).

3. Tesztelési lehetőségek a kliensoldali technológiák oktatásakor

Az alábbiakban áttekintjük, hogy a webes tárgyak oktatása során milyen eszközöket milyen szinten vagy milyen előismerettel használhatunk. A vezérelvet az alapozó tárgy tematikája szolgáltatja, de látni fogjuk, hogy az ottani tapasztalatok a komplexebb tárgyakra is tanulságul fognak szolgálni.

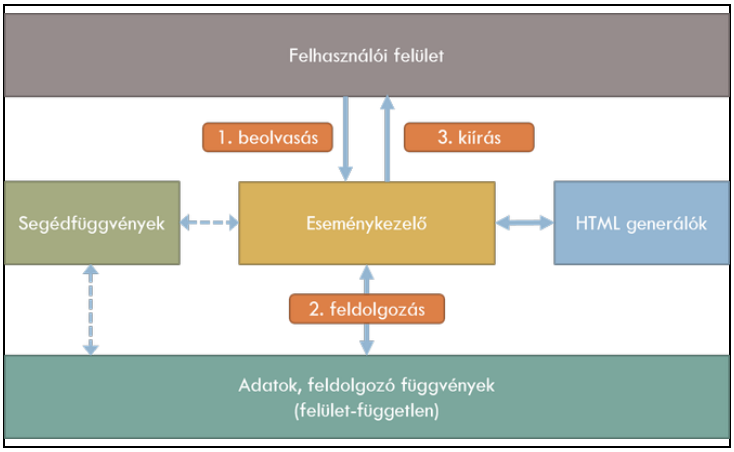
Vizsgálataink során számos módszertani vezérlőelvet alkalmazunk a tesztelésre [3]:

- a tanulók értsék meg, hogyan működik egy tesztrendszer, mielőtt használnának egyet;
- próbálják meg az elérhető nyelvi eszközökkel megoldani a tesztelést a tesztkeretrendszerek használata előtt;

- eleinte ne ők maguk írják a teszteket (oktató tesztek), hanem tanulják meg, hogyan kell használni őket és dolgozni velük;
- fokozatosan vezessünk be új eszközöket.

A bevezető kurzus egymásra épülő ismeretkörökből áll. A pusztán nyelvi elemektől fokozatosan jutunk el komplex kliensoldali alkalmazások fejlesztéséig [4]. Egy ilyen összetettebb alkalmazás a következő elemekből állhat (ld. 1. ábra) [5]:

- *eseménykezelő függvények*: mini-programokként tekintünk rájuk; általában a felhasználói tevékenység következtében futnak le, bennük a *beolvasás-feldolgozás-kiírás* általános struktúrája figyelhető meg;
- *adatok és feldolgozó függvények*: megközelítéstől függően ezek egyben is lehetnek (OOP), vagy a feldolgozó függvényeket tekinthetjük állapottér-leképezéseknek is (funkcionális);
- *segédfüggvények*: az alkalmazás központi logikájától független kisegítő függvények;
- *HTML generáló függvények*: speciális segédfüggvények, amelyek az adatokból HTML szöveget állítanak elő, általában felhasználói felület generálására használjuk.



1. ábra: Kódszervezés JavaScriptben.

3.1. Függvények tesztelése

Az előző meggondolásokból látható, hogy alkalmazásunkban sok függvény kap helyet. Ezek jelentős részét megírhatjuk tiszta függvényként. A segédfüggvények és HTML generáló függvények szinte biztosan ilyenek, egyszerűbb alkalmazások esetén az üzleti logika is ilyen függvényekből áll, az állapottér módosításához használt függvények esetén pedig a funkcionális megközelítést alkalmazva is ilyen függvényeket írunk.

```
// segédfüggvény
function range(n) { return Array.from({length: n}, (e, i) => i+1); }
// HTML generáló függvény
function genList(list) {
    return `<ul>${list.map(e => `<li>${e}</li>`).join('')}</ul>`
}
```

```
// üzleti logikai függvény
function factorial(n) {
    let f = 1;
    for (let i = 1; i <= n; i++) { f *= i; }
    return f;
}
// állapottér módosítás
function novel(state) {
    const { db, ...maradek } = state;
    return { db: db+1, ...maradek };
}
```

Az órákon ezeknek a függvényeknek a tesztelését kézzel elvégezhetjük a fejlesztői konzolban. A függvények ezekben az egyszerűbb programokban a globális névtérben helyezkednek el, így a konzolból elérhetjük őket, meghívhatjuk különböző bemenetekkel, a kimenet pedig azonnal látszik. A következő lépés ennek a folyamatnak az automatizálása lenne: a függvénydefiníciók után egy elágazásban vizsgáljuk meg, hogy a függvény helyes eredményt adott-e vissza. A logika egy `assert` függvényben általánosítható. Végül be lehet mutatni, hogy az `assert` függvényt a konzol is támogatja a `console.assert()` híváson keresztül.

```
// konzol használata
> factorial(5)
< 120

// saját ellenőrzés
console.log('0! === 1');
if (factorial(0) === 1) { console.log('OK')          }
                   else { throw new Error('failed') }

// saját assert függvény
function assert(name, actual, expected) {
    console.log(name);
    if (expected === actual) { console.log('OK') }
                        else { console.warn(`failed! ${actual} !==
${expected}`) }
}
assert('1! should be equal 1',   factorial(1), 1);
assert('5! should be equal 120', factorial(5), 120);

// console.assert
console.assert(factorial(1) === 1,   '1! should be equal 1');
console.assert(factorial(5) === 120, '5! should be equal 120');
```

Ennek használatával mindenféle plusz eszköz bevezetése nélkül helyben ellenőrizhetővé válik a kód jelentős része [8]. Az ellenőrző kódot írhatja akár a hallgató is, de tanárként elő is készíthetjük, így építve az utat a tesztvezérelt fejlesztés felé.

Miért érdemes mégis tesztfuttató környezetet és tesztkeretrendszert használni? A keretrendszerek általában jól bevált gyakorlatokat alkalmaznak a gyakran előforduló problémákra, szabályok közé szorítják a csapongó lehetőségeket, és sok kényelmi szolgáltatást nyújtanak. Következő lehetséges lépésként egy böngészőben futó tesztrendszert lehet alkalmazni. A rengeteg lehetőség közül egy, az iparban is elterjedt eszközt, a *Jasmine*-t választottuk nemcsak az érettsége és a jó dokumentációja miatt, hanem amiatt is, mert nem kell további könyvtárakat alkalmazni az ellenőrzésre vagy a helyet-

tesítések készítésére. Így is az előkészítéshez pár forrást be kell tölteni az oldal elején. Először azt a megközelítést követhetjük, hogy az alkalmazással egy dokumentumba töltjük be a Jasmine-t az oldal `<head>` részében. Az oldal törzsében pedig először a tesztelendő alkalmazáskódot emeljük be (*app.js*), majd a tesztelő kódot (*app.test.js*).

```
<!-- keretrendszer betöltése -->
<link rel="stylesheet" type="text/css" href="jasmine.min.css">
<script src="jasmine.min.js"></script>
<script src="jasmine-html.min.js"></script>
<script src="boot.min.js"></script>
<!-- az alkalmazás és teszt betöltése -->
<script src="app.js"></script>
<script src="app.test.js"></script>
```

A Jasmine a behaviour-driven development (BDD) által használatos kulcsszavakat használja a tesztesetek leírásához. Egy tesztesetet az `it` függvénnyel vezet be, a vizsgálatot az `expect` függvény oldja meg, a tesztesetek csoportosítását pedig a `describe` függvényre bízza. Gondot a névtelen függvény jelentheti kezdetben a hallgatók számára, de ezt helyettesíthetjük lambda függvényekkel, ami ismerős lehet a funkcionális programozás világából.

```
// app.test.js
describe('factorial with test array', () => {
    it('0! should be 1', () => {
        expect(factorial(0)).toBe(1);
    })
})
```

Az alapozó tárgyakon túllépve szükséges ismertetni a modul fogalmát JavaScriptben. Ekkor az üzleti logikai függvényünket külön modulba szervezhetjük ki, és önállóan, az alkalmazáson kívül tesztelhetjük. Megtehetjük ezt a böngészőben egy külön tesztfuttató HTML állományban:

```
<!-- testrunner.html -->
<script type="module" src="app.test.js"></script>
```

```
// app.js
export function factorial(n) { /* ... */}


// app.test.js
import { factorial } from './app.js';
it('0! should be 1', function () {
    expect(factorial(0)).toBe(1);
});
```

Másik lehetőség, hogy a tesztet parancssorban futtatjuk, de ekkor egy átalakítási fázist kell közbeiktatnunk a *babel* csomag segítségével. Parancssori futtatáshoz a legkényelmesebb eszköz a *Jest*, ezt kell felkészíteni az JavaScript modulok értelmezésére (*babel-jest* és *babel-core* csomagok). Ezek után az `npx jest app.test.js` futtatásával lehet a kódot tesztelni. A parancssor egyik előnye, hogy figyelő módban is elindítható a Jest, így a kód változásával a tesztek automatikusan lefutnak.

Ha egy függvény egy másik függvényt hív meg feladatának elvégzéséhez, akkor két út kínálkozik. Az egyik az, hogy a meghívott függvényt helyettesítjük egy saját implementációval. Ehhez több kód írása szükséges, és koncepcionálisan is nehezebb témakörről van szó. Példát az eseménykezelők tesztelésénél láthatunk. A másik út, hogy integrációs tesztként tekintünk az esetre, és például lentről felfele módszerrel előbb a belső függvény helyességéről győződünk meg, majd a külső függvényt teszteljük. Ez a megközelítés nem kíván újabb ismereteket a hallgatóktól.

3.2. Az állapottér tesztelése

Az alapozó webes kurzuson az alkalmazás állapotterét kezdetben globális változókban tároljuk és globális felületfüggetlen feldolgozó függvények végeznek el rajtuk módosításokat. Később az összetartozó adatokat és metódusokat objektumokba zárjuk. (Megjegyzés: az első megközelítés is objektumba zárja az adatokat és a függvényeket, ugyanis a globális névtér a `window` objektum maga.) Ez az egységbe zárás utat nyit több funkcionális rész kialakítására. JavaScriptben lehetőség van a `class` kulcsszóval objektumkonstruktorokat kialakítani, amivel érdemes is élni, hiszen az új tárgy az objektumelvű alkalmazások után következik.

Privát adattagok hiányában ki nem mondott szabályként megfogalmazhatjuk, hogy az állapotteret csak dedikált függvényeken, metódusokon keresztül módosíthatjuk (üzenetek). Az állapottér lekérdezését viszont bárhonnan megtehetjük.

```
// globális változokkal
let szamlalo = 0;
let nev = 'Anonymous';
function novel(n = 1)     { this.szamlalo += n; }
function csokkent(n = 1) { this.szamlalo -= n; }
// egységbe zárva
class Allapotter {
    constructor(kezdetiAllapot) {
        this.szamlalo = 0;
        this.nev = 'Anonymous';
        Object.assign(this, kezdetiAllapot);
    }
    novel    (n = 1) { this.szamlalo += n; }
    csokkent(n = 1) { this.szamlalo -= n; }
}
const allapot = new Allapotter({nev: 'Valami'});
```

Tesztelés során előkészítjük az állapotteret, elvégezzük a tesztelendő műveletet, majd ellenőrizzük, hogy helyes változások történtek-e. A globális megközelítésnél a globális változók beállítása történik meg, osztályba zárás esetén tisztább a helyzet, hiszen itt tesztenként új példányt hozhatunk létre az alkalmazástól teljesen szeparált módon.

```
// globális esetben
it('a novel() paraméter nélkül eggyel növeli a darabszámot', () => {
    db = 10;              // előkészít
    novel();              // elvégez
    expect(db).toBe(11); // ellenőriz
})
// egységbezárás esetén
it('a novel() paraméter nélkül eggyel növeli a darabszámot', () => {
    const allapot = new Allapotter({db: 10}); // előkészít
    allapot.novel();                          // elvégez
    expect(allapot.db).toBe(11);              // ellenőriz
})
```

3.3. A felhasználói felület és az eseménykezelők tesztelése

Noha a felhasználói felület működésének a tesztelése a funkcionális tesztelés hatáskörébe esik, az eseménykezelő függvények tesztelése pedig az egység- vagy az integrációs tesztekébe, mégis együtt kezeljük őket a továbbiakban. Ennek oka az, hogy a két terület nagyon közel esik egymáshoz. A felhasználói felületi tevékenységei eseményeket váltanak ki, a böngésző pedig meghívja az ezekhez

regisztrált eseménykezelőket. Az eseménykezelők ráadásul sok szállal kapcsolódnak a felület dokumentum objektum modelljéhez (DOM): működésükhöz szükséges adatokat onnan olvassák be, a feldolgozás eredményét pedig oda írják ki. Ezektől függetleníteni az eseménykezelőket igen nagy munka lenne, és az esetek többségében felesleges is.

A következőkben egy egyszerű példán keresztül mutatjuk be ezt a témakört: egy sugarával adott kör kerületét szeretnénk kiszámolni. Az ehhez tartozó HTML és JavaScript kód a következő:

```
<!-- index.html -->
<input  id="sugar"  >
<button id="gomb"   >Számol</button>
<span   id="kimenet"></span>
<script src="app.js"></script>
```

```
// app.js
// segédfüggvény
function $(sel) { return document.querySelector(sel); }
// feldolgozó függvény
function kerulet(r) { return 2 * r * Math.PI; }
// eseménykezelő függvény
function kattintas() {
    //beolvasás
    const r = parseFloat($('#sugar').value);
    if (isNaN(r)) { return; }
    //feldolgozás
    const ker = kerulet(r);
    //kiirás
    $('#kimenet').innerHTML = ker;
}
$("#gomb").addEventListener("click", kattintas);
```

A tesztelési lépések meghatározásánál itt is abból indulunk ki, hogy mi történik kézi tesztelés esetén: megnézzük, hogy az oldal tartalmazza-e a szükséges elemeket, majd különböző beviteli értékek mellett „megnyomkodjuk" a felületet, és ellenőrizzük a megjelenített értéket. Ezt a működést programozottan is elő tudjuk állítani. Első megközelítésben a tesztfájlokat az alkalmazás mellett helyezzük el és futtatjuk. Látható, hogy a teszteléshez új ismeretre nincsen szükség.

```
describe('felület működése', () => {
    // saját $ függvény
    function $(sel) { return document.querySelector(sel); }
    // minden teszt előtt készítsük elő a felületet
    beforeEach(() => {  $('#sugar')  .value     = '';
                        $('#kimenet').innerHTML = ''; });
    // tesztek
    it(`Minden megjelenik`, () => {
        expect($('input#sugar') ).not.toBeNull();
        expect($('button#gomb') ).not.toBeNull();
        expect($('span#kimenet')).not.toBeNull();
    });
    it(`Üresen hagyva nem jelenik meg semmi`, () => {
        $('#gomb').click();                          // elvégez
        expect($('#kimenet').textContent).toBe(''); // ellenőriz
    });
```

```
    it(`Számot írva be, jó eredményt ad`, () => {
        const n = 10, ker = kerulet(n);              // előkészít
        $('#sugar').value = n.toString();
        $('#gomb').click();                          // elvégez
        // ellenőriz
        expect($('#kimenet').textContent).toBe(ker.toString());
    });
    it(`Szöveget írva be nem jelenik meg semmi`, () => {
        $('#sugar').value = 'alma';                  // előkészít
        $('#gomb').click();                          // elvégez
        expect($('#kimenet').textContent).toBe(''); // ellenőriz
    });
});
```

A teszteléshez további függvénykönyvtárak vehetők igénybe (*jasmine_dom_matchers*, *dom-testing-library*), amelyekkel a felület kezelése és az ellenőrzés kényelmesebb lehet.

Az eseménykezelőket leválaszthatjuk a feldolgozó függvényekről, mégpedig úgy, hogy a feldolgozó függvényeket egy helyettesítő implementációval látjuk el, ami figyeli az adott függvényt. Jasmine-ban ezt a `spyOn` függvénnyel lehet megtenni. A következő példában megnézzük, hogy meghívódna-e a `kerulet` függvény:

```
it('eseménykezelő, kerulet vizsgálata', () => {
    $('#sugar').value = '10';                    // előkészít
    spyOn(window, 'kerulet');
    $('#gomb').click();                          // elvégez
    expect(kerulet).toHaveBeenCalled();          // ellenőriz
    expect(kerulet).toHaveBeenCalledTimes(1);
    expect(kerulet).toHaveBeenCalledWith(n);
});
```

Későbbi tárgyakban jogosan merülhet fel az igény a felületi elemek és funkciók izolált vizsgálatára. Erre több lehetőség adódik:

- A *jasmine-fixture* könyvtár használatával egy külön tesztoldalon belül hozhatunk létre izolált HTML elemeket, és ezeken futtathatjuk meg a teszteket. Hátránya, hogy az eseménykezelőket külön kell alkalmazni a dinamikusan beszúrt elemekre, így az eredeti alkalmazást kicsit módosítani kell.
- Az izolált tesztelést elérhetjük úgy is, hogy külön osztályba zárjuk az adott felületi elemekhez tartozó logikát: az eseménykezelőket, ezek regisztrálását és felszabadítását. (Ez nagyon hasonlít a Backbone keretrendszer nézetére.)
- Továbbvíve az előző gondolatot, egy egységbe zárt osztály akár saját maga megjelenítéséért is felelős lehet. A kirajzolás hatékonysága érdekében valamilyen DOM diffing könyvtár használata javasolt.
- Hamar elérkezünk így a komponensekig, amiket a modern keretrendszerek használnak. A komponensek önmagukban megálló, az adott felületi elemért felelős funkcionális egységek.

Lehetőség van a felület parancssori tesztelésére is. A Jest tesztrendszer fel van készítve erre is a jsdom függvénykönyvtár segítségével. A tesztbe csak be kell tölteni az adott JavaScript állományt és a tesztek ugyanúgy működnek, mint a naiv esetben.

```
beforeAll(() => {
    document.body.innerHTML = `<input id="sugar">
                               <button id="gomb">Számol</button>
                               <span id="kimenet"></span>`;
    require('./app.js');
});
```

A felületi tesztelés egy másik alternatívája egy független Chrome böngésző programozott irányítása a *puppeteer* függvénykönyvtár segítségével. Lehet böngésző megjelenésével és anélkül is futtatni a teszteket. Sajnos a programozási interfésze teljesen aszinkron, így a benne való programozás kezdőknek nem ajánlott. Létezik hozzá egy *puppeteer-recorder* nevű felvevő eszköz, de az általa generált kód sem egyszerű. Parancssorból használható, a Jesttel jól működik együtt, akár saját magunk által konfigurálva, akár a *jest-puppeteer* csomag segítségével. Elsősorban számonkérések automatikus ellenőrzéséhez ajánlott használni. Teszteléskor kétféle megközelítést alkalmaznak:

- az alkalmazás végig kattintható-e, az oldalon egyéb állapotának nincs ellenőrzése; ezzel nagyon egyszerűen írhatók tesztek;
- az alkalmazás jó adatokat jelenít-e meg, azaz nemcsak a felületi elemek megléte, hanem azok tartalma is számít.

4. Tesztelési lehetőségek a szerveroldali technológiák oktatásakor

Az alapozó tárgyban a szerveroldali webprogramozást PHP-ban tanítjuk. A tananyagot a kliensoldalhoz hasonlóan lépésről lépésre, a nyelvi elemektől a technológiai ismereteken keresztül a tervezési mintákig építjük fel [10]. A szerveroldali kód szervezésében is ügyelünk arra, hogy az adatokat és az üzleti logikát képviselő feldolgozó függvényeket elválasszuk az I/O-t jelentő HTTP kérésektől, valamint a fájl- és adatbázis-műveletektől [6].

4.1. Egységtesztek

A feldolgozó függvényeket a kliensoldalhoz hasonlóan egységteszteknek, illetve lentről felfelé építkező integrációs teszteknek vethetjük alá. A PHP-ban is van egy `assert` függvény, amely használható elemi tesztek írására. Tesztrendszer használatakor bár PHP-ban a legelterjedtebb az xUnit alapú PHPUnit, ennek formája eltér a kliensoldalon bemutatott BDD stílustól. Mivel fontosnak tartjuk, hogy minél kevesebb energiabefektetéssel tudjuk a tesztelést tanítani, így PHP-ban is kerestünk egy BDD stílusú könyvtárat, a *Kahlan*t. A tesztelés hasonlóan történik a JavaScripthez képest: vagy globálisan definiált változókat és függvényeket húzunk be a teszteseteket tartalmazó fájlba `include`-dal, vagy osztályokat példányosítunk:

```
include('./functions.php');
describe('Pozitív számok kiválogatása', function () {
    it('üres tömbre üreset ad vissza', function () {
        expect(kivalogatas([]))->toBe([]);
    });
    it('csupa pozitív tömbre az összeset visszaadja', function () {
        expect(kivalogatas([1, 3, 5, 7]))->toBe([1, 3, 5, 7]);
    });
});
```

4.2. Felületi tesztek

A szerveroldali alkalmazások célja HTML oldalak generálása. Ennek a tesztelése igazából funkcionális tesztelés: egy programmal megszólítjuk a HTTP végpontot, és megnézzük, milyen válasz jön vissza. Ehhez például a puppeteer projekt kiválóan használható. A teszteseteket ebben az esetben

JavaScriptben írjuk, de léteznek PHP-ban megírt end-to-end tesztelést végrehajtó rendszerek (pl. Codeception).

5. Esettanulmányok

5.1. Tesztvezérelt fejlesztés

Az ELTÉn oktatott JavaScript technológiák nevű speciálkollégiumon egy hallgató gyakorlatvezető segítségével kipróbáltuk, milyen az, amikor a hallgatóknak előre megírt teszteket kell kielégíteniük. A hallgatók számára új volt a tesztelés, új volt a tananyag, sőt még az elvárt parancssori környezet is gondot okozott többségüknek, így az anyag ilyetén való oktatása nehézkessé vált. Tanulságként levonható, hogy a hallgatóknak szükségük van a fokozatos építkezésre és az egyes eszközök átlátására.

5.2. Zárthelyi dolgozat értékelése

Az alapozó tárgyat évente kb. 200 hallgató veszi fel az egyik félévben. Korábban a félév végi zárthelyi gépes dolgozatot manuálisan javítottuk. Ez azonban sok energiát vett el, és szubjektivitásra adott lehetőséget. 2018 tavaszi félévében a tárgyra jelentkezett közel 200 hallgató vizsgáztatását automatikus tesztrendszerrel végeztük el. Mivel az implementációs részleteket nem vizsgálhattuk (hiszen vagy JavaScripttel vagy PHP-val oldott meg egy adott feladatot), a tesztelés funkcionálisan történt a *mocha* keretrendszer és a puppeteer használatával. A beadás egy webes felületen történt. A hallgatók év közben nem tudták a rendszert használni, a zh előtt kaptak egy tesztalkalmazást, ahol a tesztrendszer visszajelzéseivel megismerkedhettek. A zh-n főleg az ismeretlenségnek köszönhetően sok kérdés volt, de ettől függetlenül a teljesítés nem maradt el a manuális jegyektől. Az automatikus rendszer egyik előnye, hogy differenciáltabb eredményt lehet látni, hiszen a manuális ellenőrzés során adott 1-5 osztályzatok helyett itt egy nagyobb skálán mozgó pontszámot látunk, és az is kideríthető, hogy a hallgató mely ismeretkörrel nem boldogult.

A tesztek írásában érdekes kihívás, hogy az adatok tárolásának módját sem tudjuk előre meghatározni, hiszen a vizsgázó fájlban, adatbázisban egyaránt tárolhat adatot. Az ellenőrzést csak felületen keresztül tehetjük meg. Erre kétféle megoldást is kidolgoztunk: az egyik esetben véletlenszerű adatokat állítunk elő, azt mentjük el a felületen és keressük meg a megjelenítő oldalon; a másik esetben pedig előírjuk a hallgatóknak, hogy állítsanak be fix adatokat, amiket nem változtathatnak meg, mivel azoknak a helyes működését vizsgálja a program. Mivel a puppeteer API-ja az aszinkronitás miatt nehezen kezelhető, ezért egy felhasználóbarátabb teszt API kialakítását kezdtük el.

6. Módszertani kiértékelés és kitekintés

A fentiek alapján jól látható, hogy a webes tárgyak oktatása során a tanult ismeretekhez képest viszonylag kevés többlettel elérhető az alkalmazások automatikus tesztelése. Módszertani szempontból az alábbiakat érdemes megjegyezni:

- Ahogy a szoftverfejlesztő cégeknél igaz, úgy az oktatásban is vegyük figyelembe, hogy a tesztelés elsajátításához, megírásához időre van szükség. Ez kb. még egyszer annyi idő, mint az alkalmazás írása, az egyszerű programok esetében. Ha fontosnak tartjuk a tesztelés oktatását, akkor szánjunk rá kellő időt, különben a hallgatól azt fogják érezni, hogy ez a lépés kevésbé fontos az implementálásnál.
- A fent bemutatott módszer szerint a tesztelés fokozatosan is bevezethető a kézi teszteléstől a már ismert nyelvi elemek és vezérlési szerkezetek alkalmazásával kapott automatikus teszteken keresztül egészen a keretrendszerek használatáig. Ha szükséges, akkor a keretrendszereket akár el is hagyhatjuk.

- A tesztek is programok, használjuk ki a programozásoktatásban!
- A fenti eszközök alkalmasak arra, hogy különböző fejlesztési módszereket adjunk meg. A tanár által írt teszteknek való megfelelés már előre vetíti a tesztvezérelt fejlesztést. Ezt követheti a tesztek utólagos írása, végül egyszerűbb esetekben a tesztek írását is előre vehetjük [7]. Ez olyan tapasztalatot eredményez, amelyet manapság széles körben elvárnak a szoftverfejlesztésben.
- A tesztelés jótékonyan hathat a program minőségére is: tiszta függvények, függőségek megfelelő kezelése és egyéb hasznos programozási minta használatára világíthat rá.
- Az előre megírt tesztelést kiválóan lehet számonkérések objektív kiértékelésekor használni. Nagyon fontos azonban, hogy a visszajelzés a tesztekből egyértelmű és segítő legyen.
- A tesztvezérelt fejlesztést akár tutorialszerűen lehet alkalmazni online feladatmegoldó rendszerekben [8, 9]. A teszteseteket részekre bontva lépésről lépésre lehetne a tanulókkal a feladatot megoldani, ezzel irányítva és gyakoroltatva őket a feladatmegoldás lépéseiben.
- A tesztek képesek kódlefedettség információt is adni, ezt kihasználhatjuk a hallgatóktól elvárt tesztek értékelésében.
- Érdekességképpen meg lehet mutatni könnyen kezelhető hatékonysági tesztelő programokat, mint például a majomtesztelésre használt *gremlins.js* vagy a terheléses teszthez használt *ab*.

Ebben a cikkben elsősorban az alapozó tárgy tematikája volt fókuszban, az sem teljes egészében. Érdemes lenne megvizsgálni a *canvas* alapú alkalmazások tesztelését, illetve az alapozó tárgyra épülő további tanegységek esetében egyre inkább az iparban használatos eszközöket használni.

Köszönetnyilvánítás

EFOP-3.6.1-16-2016-00023: Kutatás-fejlesztési tevékenység megvalósítása az Eötvös Loránd Tudományegyetem szombathelyi kampuszán – A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Irodalom

1. A Programozás tárgy előadásainak tematikája az ELTE IK-n, http://progalap.elte.hu/?El%C5%91ad%C3%A1s (utoljára megtekintve: 2018.11.20.)
2. Gregorics Tibor, Mózsi Krisztián, Szendrei Rudolf: *Hogyan tanítsunk tesztelni?*, INFODIDACT 2017, Zamárdi, 2017.
3. David S. Janzen, Hossein Saiedian: *Test-Driven Learning: Intrinsic Integration of Testing into the CS/SE Curriculum*, SIGCSE 2006, Houston, Texas, USA, 2006
4. Horváth Győző, Visnovitz Márton: *Egy bevezető webfejlesztési kurzus módszertani megfontolásai*, Informatika a Felsőoktatásban, Debrecen, 2017
5. Horváth Győző, Visnovitz Márton: *A böngésző mint alkalmazás-fejlesztési platform*, http://webprogramozas.inf.elte.hu/tananyag/kliens/ (utoljára megtekintve: 2018.11.20.)
6. Horváth Győző, Visnovitz Márton: Dinamikus weboldalak előállítása szerveroldali technológiákkal, http://webprogramozas.inf.elte.hu/tananyag/szerver/ (utoljára megtekintve: 2018.11.20.)
7. David S. Janzen, Hossein Saiedian: *Test-driven learning in early programming courses*, SIGCSE 2008, Portland, Oregon, USA, 2008
8. Horváth Győző: *A web-based programming environment for introductory programming courses in higher education*, ICAI2017, Eger, 2017
9. Horváth Győző, Menyhárt László: *Webböngészőben futó programozási környezet megvalósíthatósági vizsgálata*, INFODIDACT 2016, Zamárdi, 2016.

A táblázatkezelés is problémamegoldás?

Papp Petra[1], Csernoch Mária[2]

[1]papppetra14@gmail.com, [2]csernoch.maria@inf.unideb.hu
DEBRECENI EGYETEM, INFORMATIKAI KAR

Absztrakt. Az érvényben lévő informatika kerettantervek és a 2018-as NAT-tervezet (2018 szeptemberi verzió) is egyértelműen magában hordozza azt az ellentmondást, miszerint a számítógépes problémamegoldás egyenértékű a programozással és a programozásoktatással. Jelen vizsgálatainkban, a táblázatkezelő eszközök programozás és adatorientációs funkcióira koncentrálva arra kerestük a választ, hogy a tankönyvek vajon követik-e a fent említett ellentmondásokat magában hordozó szemléletmódot vagy ezzel szakítva a hatékony számítógépes problémamegoldásra helyezik a hangsúlyt. Azt találtuk, hogy a vizsgált tankönyvek a felületi megközelítést preferálják az eszközökre fókuszálva, amely módszerekről már bizonyításra került, hogy nem képesek teljesíteni a kerettantervekben meghatározott követelményeket. Ezzel szemben egy olyan magas-mathability megközelítést ajánlunk, amely kiemelten támogatja az informatikán belüli és a tantárgyak közötti tudástranszfert, a hatékony számítógépes problémamegoldást.

Kulcsszavak: számítógépes problémamegoldás, táblázatkezelés, tankönyvek, tudástranszfer

1. Táblázatkezelés helye az informatika kerettantervben

1.1. Problémamegoldási megközelítések

A jelenlegi tantervi előírások [1][2][3] és a NAT2018 tervezet egyértelműen szétválasztják a programozást és az egyéb informatikai tevékenységeket azzal a kategorizálással, hogy a problémamegoldást egyenlővé teszik a programozással.

> „…az algoritmizálási készségek formális keretek közötti fejlesztése, amelyre a problémamegoldás informatikai eszközökkel és módszerekkel témakörben kerül sor. … A problémamegoldás informatikai eszközökkel és módszerekkel rész elsajátítása során a tanulók megismerkednek az algoritmizálás elméleti módszereivel, a szekvenciális és vezérléselvű programok alapvető funkcióival, majd az elméleti megalapozást követően a gyakorlatban készítenek és tesztelnek számítógépes programokat. Az elkészített programok segítségével más műveltségi területek problémái tanulmányozhatók, illetve különböző jelenségek szimulálhatók. A problémamegoldási ismeretek tanítása a mások által készített programok algoritmusainak értelmezését, az alkalmazói képesség kialakítását és a kritikus szemléletet is támogatja.” [2][3]

A számítógépes problémamegoldást ily módon leszűkítve a programozásra, ezek a dokumentumok teret biztosítanak a nem-programozói ismeretek problémamegoldás-mentes megközelítésére. Kutatások azonban egyértelműen bizonyítják, hogy a nem problémaorientált megközelítések az eszközhasználatra helyezik a hangsúlyt [4][5], mely megközelítsek kevésbé hatékonyak, sokkal inkább magukban hordozzák a hibalehetőséget [6], szemben azokkal a módszerekkel, amelyek egyenrangú partnerként kezelik a programozást a többi számítógépes tevékenységgel [7][8].

A rendelkezésre álló dokumentumok alapján vizsgálataink során kiemelt szerepet kap annak elemzése, hogy hogyan valósul meg az informatikaórákon és az informatika tankönyvekben a TPCK [9] hatékony alkalmazása, a tankönyvi megközelítések mely Meaning System modellnek felelnek meg [10], a tanulói adat és információ ellátottság hogyan viszonyul a Cognitive Load Theory [11] és a hatékony matematikatanulás pszichológiájához [12]. Vizsgáltuk továbbá, hogy milyen probléma-

megoldási (high- and low-mathability) [13][14] és gondolkodási megközelítések (gyors- és lassú gondolkodás) [15][16] azonosíthatóak a tankönyvi tartalmak alapján.

1.2. Tankönyvi elemzések szempontrendszere

Kutatásunk során a jelenleg tankönyvlistán szereplő tankönyvek és azokat teljes sorozatra kiegészítő további tankönyvek elemzését végeztük el, a kerettantervi táblázatkezelés, adatfeldolgozás témakörre fókuszálva. A tankönyvlista alapján a mintába kerültek a Mozaik, a Pedellus, a JOS, Műszaki Kiadó (MK) és az EKE (korábbi nevén NTK) 5–8. osztályos és 9–10. osztályos tankönyvei, munkafüzetei. Jelen tanulmányban az NTK 7. és 8. osztályos általános iskolás tankönyvek (NTK7 és NTK8), valamint a 9−10. osztályos tankönyv (NTK910) került elemzésre.

A tankönyvi tartalmak objektív elemzéséhez előzetesen összeállítottunk egy kérdéssorozatot (1. táblázat), valamint felhasználtuk a kutatócsoportunk további tagjai által az előző tanévben végzett kerettantervi felmérés eredményeit [17]. A kérdéseinkkel arra kerestük a választ, hogy a táblázatkezelés témakör hogyan illeszkedik az egyéb informatikai tématerületekhez, mennyiben épít a korábban szerzett ismeretekre és hogyan készíti elő a magasabb szintű adatfeldolgozást. A teszteléssel pedig azt vizsgáltuk, hogy a tankönyvi tartalmak mennyire illeszkednek a kerettantervi elvárásokhoz.

	Szempontok	**Vizsgált tartalmak**
K1.	tankönyvi táblázatok forrásai	fájlkezelés: megnyitás, mentés, fájlkonverzió forrás: adatok gyűjtése, adatok keresése, forrásmegnevezés, adatok hitelessége
K2.	rekord, mező, elválasztó karakterek	adatok csoportosítása, értelmezése adatbáziskezelés, szűrés, programozás, szövegkezelés, sémaépítés
K3.	sorok száma, sorok elrejtése, adattábla rögzítése	nagy mennyiségű adat kezelése
K4.	mintatáblázatok tartalma, témája	tantárgyközi kapcsolatok, motiváció, adatok keresése, csoportosítása, értelmezése (TPCK)
K5.	tématerület bevezetése	elméleti vagy problémaorientált
K6.	függvények száma	Cognitive Load Theory, sémaépítés
K7.	összetett függvények	matematika, programozás, adatbáziskezelés
K8.	adattípusok, automatikus típusfelismerés, adatformázás, tizedes és ezreselválasztó karakterek	adatok értelmezése, adatbáziskezelés, programozás, természetes nyelvek, matematika, sémaépítés,
K9.	képletek másolása, tömbképlet	változó, vektor, programozás, sémaépítés

1. táblázat: Tankönyvi elemzések szempontrendszere a táblázatkezelés témakörében.

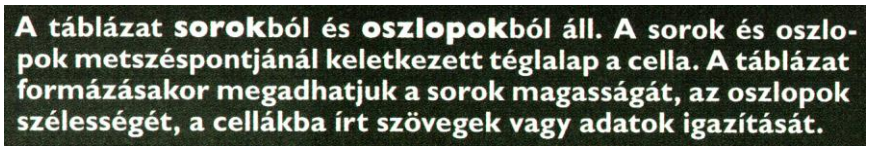
2. Tankönyvi elemzések

Jelen tanulmányban a tankönyvlista NTK tankönyvsorozat 7., 8. és 9–10. osztályos tankönyveinek táblázatkezelési fejezeteit mutatjuk be. Az 1. táblázatban ismertetett szempontrendszeren túl fontosnak tartottuk annak elemzését is, hogy a középiskolai tankönyv hogyan épít az általános iskolai tankönyvre, hogyan használja fel az ott szerzett tudást.

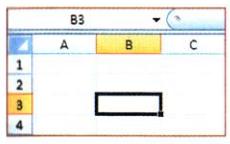
Kitérünk továbbá annak elemzésére is, hogy milyen formában történik a témakör bevezetése, különös tekintettel az „elméleti" bevezetésekre. Ennek az elemzési szempontnak a jelentőségét az adja, hogy az „elméleti" bevezetések bizonyítottan nem alkalmasak egy-egy új fogalom megértéséhez, tehát az oktatási folyamatban kerülendő gyakorlat:

> „Még egy következményét vezethetjük le annak az alapelvnek, hogy egy személy számára az általa ismerteknél magasabb rendű fogalmakat definíció segítségével nem közvetíthetünk, nevezetesen azt, hogy a fogalom maga nem definiálható: minthogy minden egyes fogalom ennek a fogalomnak egy példája, ezért ez a fogalom magasabb rendű minden más fogalomnál. … Hasonlóan, úgy hiszem, a matematika sem definiálható, csupán példákkal világítható meg." [12]

Ezt az ellentmondást szemlélteti a 7. (1. ábra) és a 9–10. osztályos (2. ábra) tankönyvek próbálkozása a fogalomalkotásra. Mindkét tankönyv definíció-bevezetéssel próbálkozik, a középiskolás könyv figyelmen kívül hagyva az általános iskolás könyvet, mindkét esetben példák és feladatmegoldások nélkül.



1. ábra: Cella, sor, oszlop fogalmi bevezetés a 7. osztályos tankönyvben.



Az aktív cella (B3)

A táblázat cellákból áll, az egymás melletti cellák sorokat, az egymás alatti cellák oszlopokat alkotnak. Az **oszlopokat betűkkel** (pl. A, B, C, …), a **sorokat számokkal** (pl. 1, 2, 3, …) jelöljük. Egy cellát az oszlop jelével és a sor számával azonosíthatunk (pl. A1, A2, B1, B2, …). A cellák közül megkülönböztethetjük azt, amelyikkel éppen dolgozunk, ezt **aktív cellának** nevezzük. Például a B3 cella a második oszlop harmadik sorában levő cellát jelzi. Az aktív cella körül egy vastag szegély látható, a cella jele a *Név* mezőben jelenik meg.

2. ábra: Cella, sor, oszlop fogalmi bevezetés a 9–10. osztályos tankönyvben.

2.1. K1: Tankönyvi táblázatok forrásai

A tankönyvi táblázatkezelési fejezetek elemzésének szempontrendszerében kiemelt fontosságú a táblázatok tartalma és azok forrása, mivel korábbi kutatások egyértelműen bizonyítják, hogy a táblázatkezelés-oktatás sikertelensége nagyban magyarázható a szövegkörnyezet nélküli táblázatok, eszközcentrikus [9][16][18], valamint „elméleti" megalapozási [12] módszerek alkalmazásával.

A 7. osztályos tankönyv a szövegszerkesztésben szerzett táblázatos tudáselemekkel – tabulátorok, táblázatok, szövegből-táblázat konverzió eszközöket használva – próbálja bevezetni a táblázatkezelési ismereteket. A tudástranszfer láncolat azonban megszakad, mivel a szövegszerkesztőben létrehozott táblázatok nem kerülnek át táblázatkezelői környezetbe. Ennek a tudástranszfer elemnek a hiánya is nagyban közrejátszik abban, hogy mind a 7., mind a 8. osztályban a táblázatkezelővel feldolgozott valamennyi adat gépeléssel kerül bevitelre, egyéb más lehetséges módszer említésre sem kerül. Mindez történik annak ellenére, hogy a gépelés valamennyi hátrányával tisztában vagyunk (például: időigényes, pontatlan, eltérő gépelési sebességek, nem hiteles, kevés adat, rövid táblázatok, unalmas, nem az informatikaóra feladata), szemben a minimális előnyével (adatbevitel) (2. táblázat).

A gépelés, gépeltetés további hátránya, hogy elveszítjük a más informatikai témakörökben tárgyalt tudáselemek gyakoroltatását (például: fájlkezelés, mozgatás, másolás, internetes tartalmak keresése, speciális keresés), a valós tantárgyközi kapcsolatokat, valamint a tanulók motivációját. A középiskolás könyv megemlíti a gépelésen túli szövegbeviteli módokat is, de egyetlen mintatáblázatot és ehhez köthető feladatot sem mutat. Feladat ötleteket találunk a tankönyvi fejezet utolsó oldalán (18 feladat, NTK910 104.o), amelyekben az adatrögzítés többségében gépeléssel történik, míg két feladathoz, a gépelésen túl, az internetes forrás is meg van adva (kipróbáláskor egyik link sem működött, és további alternatív, elérést lehetővé tevő adatok nem állnak rendelkezésünkre: hiányzik a weblap címe, szerzője, az elérés dátuma).

tankönyv	**tartalmak**
NTK7	könyvtári nyitva tartás: gépelés (szöv.szerk.), szövegből táblázat konverzió (szöv.szerk.) nincs cím: papírgyűjtés: gépelés közvélemény-kutatás: gépelés költségvetés-tervezet: gépelés
NTK8	internethozzáférés: gépelés Kisfenyő Panzió: gépelés Informatika Alkalmazói Verseny (IAV): gépelés dátumok: gépelés felvételi: gépelés szakkör1: gépelés szakkör2: gépelés felvételi2: korábbi felvételi táblázat
NTK910	üres táblázat (2. ábra) üres táblázat (újabb) nettó és bruttó ár: gépelés a, b, c: gépelés 3–7. feladat: gépelés 1–31. feladat: üres, ismeretlen cella tartalmak, gépelés 32. feladat: gépelés, képlet 1–18. feladat: gépelés, unplugged, tetszőleges internetes források

2. táblázat: Tankönyvi adatforrások (üres táblázatok többsége kihagyva).

Összességében megfogalmazhatjuk, hogy az általános iskolás NTK tankönyvek táblázatai valós, ugyanakkor szinte kizárólagosan iskolai témákat dolgoznak fel, amely, mint motivációs eszköz erősen megkérdőjelezhető. Ezzel szemben a középiskolás tankönyv nem használ egyetlen valódi táblázatot sem. Csak „elméleti" megközelítésben találkozunk a feladatok között hiteles táblázatokkal, de sem ezen táblázatok konverziója, sem az ezekből történő információ-lekérdezés nem kerül részletes feldolgozásra.

2.2. K2: Rekord, mező, elválasztó karakterek

A következő elemzési szempont alapján azt vizsgáltuk meg, hogy a feldolgozott táblázatok milyen méretűek, tehát hány adatrekordot tartalmaznak, valamint azt, hogy ezen környezetben szerzett tudáselemek hogyan készítik elő az adatfeldolgozást, hogyan transzferálható más adatfeldolgozási közegbe.

Az általános iskolás tankönyv táblázatai rövid, maximum 10–15 rekordot tartalmaznak. A táblázatok többségében találni mezőneveket és ezek száma többségében megegyezik az adatoszlopok számával. Ezek a tankönyvek azonban nem térnek ki a címsor és az adatrekordok közötti különb-

ségre. A középiskolás tankönyvben említésre kerül a címsor, de egyetlen példát sem láthatunk. Mivel a tankönyv egyetlen valós táblázatot sem tartalmaz, így a tankönyvet nem tudjuk értékelni a nemlétező táblázatai alapján (3. táblázat).

tankönyv	tartalmak
NTK7	könyvtári nyitva tartás: 3 adat, 2 oszlop, 4 rekord, mezőnevek hiányoznak, elválasztó karakter: tabulátor nincs cím: 3 adat, 3 oszlop, mezőnevek hiányoznak, elválasztó karakter: tabulátor, 3 rekord papírgyűjtés: 7 adat, 7 oszlop, 6 mezőnév, 1 hiányzik, 8 rekord közvélemény-kutatás: 3 adat, 3 oszlop, 3 mezőnév, 4 rekord költségvetés-tervezet: 1 munkalapon több független táblázat
NTK8	internethozzáférés: 5 adat, 5 oszlop, 5 mezőnév, 3 rekord Kisfenyő Panzió: egy munkalapon két egymástól független táblázat IAV: 6 adat, 6 oszlop, 6 mezőnév, 10 rekord dátumok: egy-egy cella, 1 rekord felvételi: 8 adat, 8 oszlop, 8 mezőnév, 10 rekord szakkör1: 6 adat, 6 oszlop, 6 mezőnév, 5 rekord szakkör2: 4 adat, 4 oszlop, 3 mezőnév, 15 rekord
NTK910	címsor: említés szintjén, nincs példa mutatva üres táblázatok nettó és bruttó ár: 2 adat, 2 oszlop, 2 mezőnév, 1 rekord a, b, c: 4 adat, 4 oszlop, 4 mezőnév, 1 rekord napok: 1 adat, 1 oszlop, címsor nincs, 2 rekord (96.o.) napok2: 2 adat, 2 oszlop, címsor nincs, rekord nincs (100.o.) tartalom nélküli táblák (102.o.)

3. táblázat: A tankönyvi minta-táblázatok mérete.

2.3. K3: Sorok száma, sorok elrejtése, ablaktábla rögzítése.

Az előző pontban, a tankönyvi táblázatok mérete alapján elvégzett elemzés egyértelművé tette, hogy minimál vagy üres táblázatok esetén a jelen alfejezetben taglalt elemzési szempontok nem relevánsak. A tankönyvi szövegkörnyezetek alapján arra a következtetésre juthatunk, hogy a rekordok számának meghatározására, azok hatékony kezelésére alkalmas eszközök nem kerülnek gyakorlásra egyik évfolyamon se (4. táblázat).

tankönyv	tartalmak
NTK7	nincs
NTK8	nincs
NTK910	nincs

4. táblázat: Nagy mennyiségű adatok kezelése a tankönyvi táblázatokban.

2.4. K4: Mintatáblázatok tartalma, témája

Ahogy az már a 2.1 fejezetben említésre került, az általános iskolás tankönyvek egyetlen kivétellel csak és kizárólag iskolai témájú táblázatokat használnak és ezek adattartalmához gépeléssel jutnak el. A fiktív iskolai tartalmú táblázatok azonban kevésbé motiválók és hitelesek, mint az adekvát, valós,

valóságalapú és algoritmusalapú, a tanulók életkori sajátosságainak és érdeklődési körének megfelelő tartalmak [28] (5. táblázat). A középiskolás tankönyv egyetlen mintatáblázatot sem tartalmaz, tehát ebből a szempontból sem értékelhető. Ugyanakkor, a táblázatkezelői fejezet végén felsorol 18 lehetséges tématerületet, amelyek alkalmasak táblázatkezelői problémamegoldásra. Ezek a feladatok témáikban változatosak, de egyhez sem kapunk konkrét feladatokat és azok megoldását sem csatolták. Hiányzik továbbá a megadott két weblap konverziója, tehát az az algoritmus, amely alapján a webtáblából adattáblát készíthetünk [27]. (A cikk írásának időpontjában egyik weblap sem elérhető, így nem állt módunkban ezek tesztelése.) (5. táblázat)

tankönyv	**tartalmak**
NTK7	könyvtári nyitva tartás: iskolai (fiktív) nincs cím: iskolai (fiktív) papírgyűjtés: iskolai (fiktív) közvélemény-kutatás: iskolai (fiktív) költségvetés-tervezet: iskolai (fiktív)
NTK8	internethozzáférés: MTI (forrásmegnevezés nélkül) Kisfenyő Panzió: utazás (fiktív) IAV: iskolai (fiktív) dátumok: saját adatok (fiktív) felvételi: iskolai (fiktív) szakkör1: iskolai (fiktív) szakkör2: iskolai (fiktív)
NTK910	nincsenek mintatáblázatok 32. feladat: szorzótábla (103.o.) 1–18. feladatok: feladatötletek, táblázat és megoldás nélkül (104.o.)

5. táblázat: A tankönyvi táblázatok témája, tartalma.

2.5. K5: Tématerület bevezetése

A táblázattartalmak bevezetése, tehát ahogy a tanár, a tankönyv felveti a problémát szintén erős motivációs eszköz lehet (6. táblázat). Az általános iskolás tankönyvek az esetek többségében egy hosszabb bevezető szöveggel készítik elő a táblázatokat, amely szövegek egyrészt adatforrásként szolgálnak, másrészt az érdeklődés felkeltése a cél. Kérdéses azonban, hogy ezek az iskolai témájú, gépelésre fókuszált bevezető szövegek hogyan tudják az általános iskolás tanulók érdeklődését felkelteni. Nem tartalmaznak továbbá olyan feladatokat, amelyekre azt tudnák mondani a gyerekek, hogy ezért érdemes volt 20-25 percet gépelni. A minimál táblázatokkal sokkal inkább azt érjük el, hogy a gyerekek úgy gondolják, hogy fejben, kézzel sokkal hamarabb meg tudnák válaszolni a minimál kérdéseket.

A középiskolás tankönyvek nem tartalmaznak minta táblázatokat, feladatokat, a teljes tananyag „elméleti" megközelítésű, ami biztosan nem fejleszti a tanulók problémamegoldó képességét [10], a sémaépítési folyamatokat [11][12], illetve nem segíti a tanulókat a megértés folyamatában [12].

tankönyv	tartalmak
NTK7	könyvtári nyitva tartás: szövegszerkesztő: tabulátor nincs cím: szövegszerkesztő: tabulátor, szövegből táblázat konverzió papírgyűjtés: táblázatkezelő (nincs kapcsolat az előző két táblázattal) közvélemény-kutatás: táblázatkezelő (nincs kapcsolat az előző táblázatokkal) költségvetés-tervezet: közvélemény-kutatáshoz témájában kapcsolódik, adattartalomban nem
NTK8	internethozzáférés: szövegből adatgyűjtés Kisfenyő Panzió: rövid fiktív szövegkörnyezet IAV: rövid bevezető szöveg dátumok: eszköz orientált felvételi: bevezető szöveg, eszköz orientált szakkör1: bevezető szöveg, eszköz orientált szakkör2: bevezető szöveg, eszköz orientált
NTK910	nincsenek témák, eszköz centrikus tárgyalás 32. feladat: szorzótábla (103.o.) 1–18. feladatok: feladatötletek, táblázat és megoldás nélkül (104.o.)

6. táblázat: A tématerületek bevezetése, mint motivációs eszköz.

2.6. K6: Függvények száma

Mindig nagy kérdés, hogy hány darab függvényt érdemes tanítani. Kutatások egyértelműen bizonyítják, hogy a hétköznapi életben egy átlag felhasználó 12 függvénynél többet nem alkalmaz [19]. Ez a megállapítás teljesen összecseng azzal, amit a Logo programozási nyelv tanítása során tapasztaltak a zürichi egyetem kutatói (ETH Zürich), mely szerint 5 utasítással indíthatunk és kb. 15 utasítás elegendő összetett feladatok megoldásához is [20]. Ezen kutatási eredmények ismeretében mindenképpen érdemes megnézni, hogy az egyes tankönyvek hány függvényt tanítanak, milyen feladatokat adnak és hogyan engednek teret a gyakorlásra.

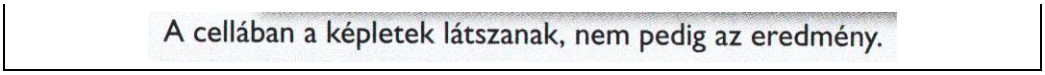
A 7. osztályos tankönyv 4 függvényt említ – SZUM(), ÁTLAG(), MIN(), MAX() –, ám ezek közül egyetlen egyre mutat példát, a SZUM() függvényre (7. táblázat). A függvények darabszáma teljesen megfelel a tanulók életkori sajátosságainak, arra viszont nem tudunk magyarázatot adni, hogy a többi függvényre miért nem oldanak meg feladatokat. További magyarázatra szorul a 3. ábrán bemutatott megoldás is.

tankönyv	tartalmak
NTK7	papírgyűjtés: 4 darab függvény, feladat 1 függvényre
NTK8	internethozzáférés: nincs IAV (56.o.): 4 ismétlés, 4 új függvény, 5 függvényre egy-egy feladat, ezek közül 1 függvényre 3 feladat, konstanssal használva dátumok (58.o.): 5 darab függvény, ezek közül 3 függvényre feladat felvételi: 1 darab függvény, erre 1 feladat
NTK910	felsorolt függvények száma csoportosítva: 5+7+6+4+5+5+5+4=41

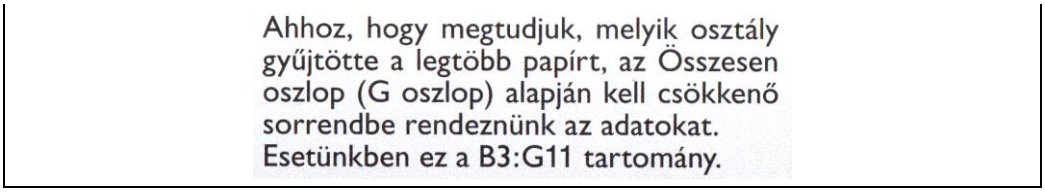
7. táblázat: A tankönyvekben felsorolt függvények száma.

A függvények számának elemzésén túl (7. táblázat) mindenképpen fontos megemlítenünk a függvények bevezetéséhez köthető tankönyvi megjegyzéseket is. Ezekre mutatnak két példát a 3. és a 4. ábra mintái. A 3. ábra egy speciális, nem alapértelmezés szerinti beállításra utal, amely inkább

„elméleti" jellegű, és kezdő felhasználóknál fölösleges ilyen részletekre kitérni. A 4. ábra mintája egy olyan feladatot mutat, amely függvényekkel, a tanulók addig szerzett ismeretei alapján nem oldható meg, és a tankönyv egy nem túl szerencsés alternatív megoldást kínál, mint egyetlen megoldási lehetőséget.



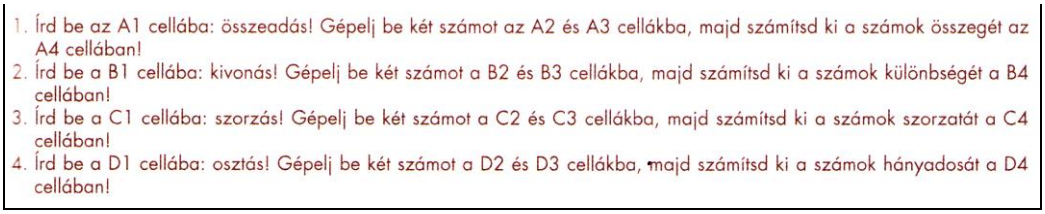
3. ábra: A 7. osztályos tankönyv egy megjegyzése, amely a képlet bevitelekor vagy a képlet kiértékelését követően csak speciális beállítások mellett (Képletek→Képletek) érvényes. Alapértelmezett beállítások szerint a képletek kiértékelését követően az értékek jelennek meg a cellákban.



4. ábra: A 7. osztályos tankönyv egy feladata, amely közvetlenül a MAX() függvény bevezetése után következik. A tankönyv nem említi, hogy ez a feladat függvényekkel is megoldható, való igaz, hogy a tanulók még nem rendelkeznek az ehhez szükséges ismeretekkel.

A 8. osztályos tankönyv ismét felsorolja a 7. osztályos 4 függvényt, ezekre hoz egy-egy példát, felsorolja példák nélkül a GYÖK() és a HATVÁNY() függvényeket, majd bevezeti a DARAB() és a DARABTELI() függvényeket. A DARAB() függvényre nincs feladat, míg a DARABTELI() függvényre csak konstans-példákat hoz (7. táblázat). A konstans-példák jelzik, hogy a DARABTELI() függvény nem egy szerencsés választás, mivel használata rendkívül komoly korlátokba ütközik [22]. Ezt követően felsorolásra kerül 5 dátum függvény, amelyek közül háromra ad egy-egy minimál példát a könyv (7. táblázat). 8. osztályban az utolsó függvény a HA(), amelyre megoldanak egy példát (7. táblázat).

A középiskolás tankönyv bevezet 41 függvényt (7. táblázat), amelyek mögött egyetlen valós táblázat sincs. A feladatok unalmasak, gondolkodást nem várnak el a tanulóktól (5. és 6. ábra), csak tartalom nélküli (7. ábra) vagy konstans példák. A függvényleírások az olykor pontatlan súgók kimásolása, melyre klasszikus példa a HOL.VAN() függvény hibás szintaktikai leírásának kimásolása (NTK910 101.o.) [16].



5. ábra: Értelmetlen gépelések, unalmas feladatok matematikai operátorok gyakoroltatására.

1. Gépelj be számokat az A2:A5 cellákba, majd számítsd ki az elemek összegét a SZUM függvénnyel az A6 cellában!
2. Gépelj be számokat a B2:B5 cellákba, majd írasd ki az elemek közül a legkisebbet a MIN függvénnyel a B6 cellában!
3. Gépelj be számokat a C2:C5 cellákba, majd írasd ki az elemek közül a legnagyobbat a MAX függvénnyel a C6 cellában!
4. Gépelj be számokat a D2:D5 cellákba, majd írasd ki az elemek átlagát az ÁTLAG függvénnyel a D6 cellában!
5. Gépelj be számokat és szövegeket az E2:E5 cellákba, majd a megfelelő függvénnyel írasd ki az E6 cellában, hogy az elemek között hány szám található!

6. ábra: Értelmetlen gépelések, unalmas feladatok függvények gyakoroltatására.

2.7. K7: Összetett függvények

Az összetett függvények nem kerülnek említésre egyetlen tankönyvi feladatban sem. A középiskolás tankönyv a logikai függvények gyakoroltatásakor, minden magyarázat nélkül prezentál három összetett függvényt, amelyek komoly kihívást jelenthetnek a tanulóknak (7. ábra). Elsőéves informatika szakos hallgatókkal végzett mérések egyértelműen bizonyítják, hogy az összetett függvények használata a tanulóknak nem veleszületett képessége [21]. Sok-sok gyakorlásra van szükség, hogy a tanulók átlássák az értékátadás menetét és önállóan is képesek legyenek feladatmegoldásokban alkalmazni az összetett függvényeket.

tankönyv	tartalmak
NTK7	nincs
NTK8	nincs
NTK910	9. feladat: 3 összetett függvény, tartalom nélkül (98.o.)

8. táblázat: Összetett függvények említése és használata a tankönyvekben.

9. Milyen értéket adhatnak eredményül az alábbi logikai függvények?
   a) =ÉS(A2>B2;C2>D2)
   b) =VAGY(A2>B2;C2>D2)
   c) =NEM(IGAZ)
   d) =NEM(HAMIS)
   e) =NEM(NEM(IGAZ))
   f) =NEM(NEM(HAMIS))
   g) =HA(A2>B2;"igen";"nem")
   h) =HA(C2>D2;C2;D2)
   i) =HA(E2<100;E2*100;"")
   j) =HA(ÉS(A2<0;B2<0);"mindkettő";"legfeljebb az egyik")

7. ábra: Logikai függvények gyakoroltatására adott feladatok táblázatok és magyarázat nélkül.

2.8. K9: Adattípusok, automatikus típus felismerés, adatformázás, tizedes és ezreselválasztó karakterek

A táblázatkezelő programok automatikus típusfelismerése nagyban megkönnyíthetné a kezdők számára az adatok típusainak felismerését. A két leggyakoribb adattípus a szöveg, amely balra igazított, és a szám, amely jobbra igazított. Ezek az igazítások teljes összhangban állnak a magyar nyelvtanból és a matematikából szerzett ismeretekkel. Ezzel szemben a tankönyvek mindezen háttérismeretet és az automatikus igazításból származó előnyöket figyelmen kívül hagyva az adatok tetszőleges formázására helyezik a hangsúlyt. Teszik ezt úgy, hogy olyan formázásokat tanítanak új ismeretként, amelyek nem táblázatkezelési ismeretek. Ezzel a megközelítéssel figyelmen kívül hagyják a lehetséges tudástranszfer tartalmakat: szövegkezelési és tipográfiai ismeretek.

tankönyv	tartalmak
NTK7	számok igazítása nincs említve szöveg alapértelmezés szerint balra, de középre igazítva formázással számok igazítása középre
NTK8	internethozzáférés: szöveg balra, szám középre igazítva, egy adatmezőben eltérő formátumok Kisfenyő Panzió: szöveg balra, szám középre igazítva IAV: szöveg balra és középre igazítva, szám középre igazítva dátumok: szám és dátum középre igazítva, szám jobbra igazítva felvételi: szöveg balra és középre igazítva, szám középre igazítva szakkör1: szöveg balra, szám jobbra igazított szakkör2: szöveg középre és balra igazított, szám és dátum középre igazított
NTK910	tartalom nélkül, elméleti bevezetés (88.o.) tartalom nélkül minták, minden adattípus balra igazítva (91.o.) a, b, c: számok és szövegek középre igazítva

9. táblázat: A tankönyvi minta-táblák adattípusai.

Ezen megközelítések egyik lehetséges következménye, hogy a tanulók nem ismerik fel az automatikus adattípusokat és helytelen outputokat fogadnak el eredményként. Erre mutat példát a 8. ábra.

	A	B	C	D
1	Felhasználó	Feltöltés	Feliratkozó	Megtekintés
2	VamosART	484	1,107,555	226,195,766
3	Videómánia	338	833,23	254,545,702
4	PamKutya	120	809,866	223,441,355
5	LetsGoMartin	176	725,638	162,798,559
6	TheVR	1,062	592,675	213,550,948
7	luckeY	1,183	561,13	150,341,428
8	Peter Gergely	100	548,241	79,713,757
9	Scribble Netty	159	546,049	74,234,471
248	Szilvaglam	87	61,899	3,918,538
249	rance flow	524	61,863	53,275,385
250	KIS GRÓFO (official)	9	61,65	30,712,031
251	KODIAK	736	61,467	14,599,194

8. ábra: A helytelen ezreselválasztó karakterek következménye, hogy az eredeti egész számok a magyar nyelvű táblázatkezelőben lehetnek egész (B2:B5, B8:B9, B248:B251) és valós számok (B6:B7, C3:C9, C248:C251), valamint szövegek (C2, D2:D9, D248:D251) [17].

A 8. ábra minta-táblázata a YouTube Top 250 magyar felhasználók listáját tartalmazza. A weblap érdekessége, hogy felismeri, hogy melyik országból kezdeményeztük a keresést, és a felismert ország listáját jeleníti meg a webtáblában. Nem igazodik azonban az ország és a nyelv helyesírási szabályaihoz. Ennek következménye, hogy az angol nyelvterületen használatos ezreselválasztó karaktert, a vesszőt használja a magyar táblázatban is. A vesszők meghagyásával az adattábla tartalmazni fog egész és valós számokat, valamint számoknak látszó szövegeket. Az adattípusok tehát nem felelnek meg a szemantikai tartalomnak, mely szerint a felhasználók, a feliratkozók és a megtekintők száma is egész szám kellene, hogy legyen. Ennek a feladatnak a megoldásához az egyetlen segítség az automatikus felismerés szerinti igazítások ismerete.

A középiskolás tankönyv példákat mutat különböző adattípusokra, de ezek a példák is szövegkörnyezetből kiragadott, „elméleti" minták, melyekről nehezen dönthető el, hogy milyen szándékkal kerültek bemutatásra. A táblázat további óriási problémája, hogy valamennyi minta-érték balra igazítva jelenik meg, figyelmen kívül hagyva az automatikus igazításokat (9. ábra).

adattípus	példa
szám	2013
pénznem	1500 Ft
százalék	25%
szöveg	informatika
dátum	2013.02.14
dátum	2013. február 14.
idő	8:00
logikai	IGAZ
tudományos	5,00E+06
egyéni	# ##" m"

9. ábra: A középiskolás tankönyv szövegkörnyezetből kiragadott példái adattípusokra, valamennyi adattípus-minta balra igazítva.

2.9. K9: Képletek másolása, tömbképlet

Az eredmények sokszorosítására a tankönyvek csak és kizárólag a másolást használják, annak ellenére, hogy így elengedhetetlen a hivatkozástípusok bevezetése. A különböző típusú hivatkozások az egyik legnehezebb fogalomegyüttes a táblázatkezelésben, tehát általános iskolában mindenképpen érdemes elkerülni.

Annak ellenére, hogy képletek másolásával szemben a tömbképleteknek számtalan előnye van, egyik tankönyv sem említi ezt a lehetőséget. Az alábbiakban röviden összegezzük a tömbképletek előnyeit, szemben a képletek másolásával:

- egyetlen képlet keletkezik ⇒ nincs szükség másolásra ⇒ képlet módosítása egy helyen történik ⇒ biztonságosabb, mint a másolás
- output: egy vektorban vagy egy változóban tárolt értékek/érték,
- elkerülhető a különböző típusú hivatkozások korai bevezetése,
- grafikus felületen tömb deklarálás ⇒ magas szintű programozási nyelvek oktatásához alapozó ismeretek,

- tömb deklarálása rendkívül egyszerű: tartomány kijelölése,
- egyetlen képlet ⇒ többszörös output ⇒ ciklus fogalmának bevezetése,
- beépített függvények helyettesíthetők algoritmus alapú tömbképletekkel ⇒ programozási tételek kódolása funkcionális programozási nyelven.

A tömbképletek felsorolt előnyei kiemelten fontosak abból a szempontból is, hogy a táblázatkezelő programokat programozási eszközként kezeli, amellyel elő lehet készíteni más szövegalapú magasszintű programozási nyelvek tanítását [19][22][24][26].

tankönyv	tartalmak
NTK7	papírgyűjtés: nincs közvélemény-kutatás: nincs
NTK8	internethozzáférés: nincs Kisfenyő Panzió: korábbi másolásra hivatkozik, de erre nem volt példa, relatív és abszolút hivatkozás, nem derül ki, hogy melyik irányba másoljuk a képleteket és hol hozunk létre új képletet IAV: másolás abszolút hivatkozással dátumok: nincs felvételi: másolás
NTK910	üres táblázatokban: másolás, relatív, abszolút és vegyes hivatkozásokkal szorzótábla: másolás vegyes hivatkozással

10. táblázat: Tömb eredmények megjelenítésére használt tankönyvi eszközök.

3. Tudástranszfer alapú adatkezelés – Sprego

A tankönyvek elemzéséhez összeállított szempontrendszer, valamint az adatkezelés problémamegoldási megközelítése egyértelműen mutatja, hogy melyek azok az irányelvek, amelyek követése lehetővé teszi a hatékony, tudástranszfer alapú adatkezelés tanítását táblázatkezelői környezetben. Méréseink egyértelműen bizonyítják, hogy az általunk használt magas-mathability megközelítések lényegesen hatékonyabbak, mint az elterjedt, széleskörben elfogadott, a szoftvergyártók által is preferált felületi megközelítések.

Fontos hangsúlyozni, hogy ezen megközelítés elsődleges jellemzője a programozás-orientáltság és ennek támogatása oly módon, hogy kihasználjuk a táblázatkezelői környezet funkcionális programozási [24][26] lehetőségeit autentikus adatokat használva valódi problémamegoldásra. A függvény fogalmát matematikából vesszük kölcsön és a táblázatkezelő függvényeit, valamint a Sprego [22] módszert használva bővítjük ezt a fogalmat a többváltozós és összetett függvények gyakorlati, ismételt alkalmazásával. Remélve, hogy ezt a bővített tudást, más tantárgyak is fel tudják használni, ahol kiemelten gondolunk a matematika irányába történő visszacsatolásra.

Az autentikus táblázatok elsődleges forrása a tanulók más tanórákon gyűjtött adatai, vagy ennek hiányában, az internet. Az internet adatgazdagsága lehetővé teszi olyan webtáblák elérését és letöltését, amelyek tartalomban megfelelnek a tanulók érdeklődési körének, korának, tantárgyközi kapcsolatainak. Más szavakkal, olyan tartalmak elérését és feldolgozását tesszük lehetővé, amelyek motivációs faktora jelentős. Az internetes webtáblák további előnye, hogy a webtábla→adattábla konverziós folyamat [27][28] lehetővé teszi az informatikai algoritmus fogalmának bevezetését, az algoritmus-megvalósítás felhasználói eszközeinek alkalmazását, valamint azt, hogy különböző formában prezentálható táblázatokat tudjunk előkészíteni a tanórákra. A táblázatok sokfélesége és különböző

formája lehetővé teszi, hogy a tanulók háttérismereteinek és az óra céljainak leginkább megfelelő verziót hozzuk létre mind tanórai, mind egyéni feldolgozásra, problémamegoldásra.

Külső adatok konverziója, előkészítése adatfeldolgozásra, a programozási és adatbáziskezelési ismeretek bevezetésén túl, kiváló teret biztosít fájlkezelési, adatelemzési, adattípusok, szöveges és numerikus adatok kódolásához köthető ismeretek gyakorlására, ezekben a témákban sémák kialakítására [22][27][28].

A tartalmi motivációs eszközökön túl a Sprego módszert támogató számos unplugged és semi-unplugged oktatási eszköz került fejlesztésre, megvalósításra [28]–[34]. Az unplugged eszközök elsősorban az összetett függvényeken belüli értékadás folyamatának megértését segítik [29]. A semi-unplugged 2D és 3D alkalmazások [32]–[34] avatárok mozgatásán és egy egyedi fejlesztésű képletkiértékelőn keresztül algoritmusok Sprego-implementációit mutatják be demó és interaktív üzemmódokban.

Mindezen lehetőségeket és kutatási eredményeket figyelembe véve, napjainkra sikerült olyan megközelítéseket, módszereket, eszközöket kidolgozni, amelyekkel hatékonyabbá tehető a táblázatkezelés-oktatás, teret adva egy minimál eszközigényű programozási környezet aktiválására.

Összegzés

A táblázatkezelés is problémamegoldás? Igen, ahogy az az elemzett középiskolás tankönyv utolsó táblázatkezelési oldalán egyértelművé válik. A kérdés igazából tehát az, hogy miért csak az utolsó oldalon kerül felsorolásra néhány lehetséges adatfeldolgozásra alkalmas valódi tartalom, miért kellett megelőzze ezt az oldalt sok-sok felesleges „elméleti” bevezetés. Egy olyan megközelítés, amelyről évtizedekkel korábban már bizonyításra került, hogy nem fejleszti a tanulók problémamegoldó képességét, nem segíti a megértést.

Az informatikaoktatás elsődleges célja a tanulók számítógépes gondolkodásának hatékony fejlesztése. Ennek a célnak a megvalósításához azonban mindenképpen arra van szükség, hogy a különböző számítógépes tevékenységek mindegyikét koncepció- és algoritmusalapú [23][24][25], alapjaiban azonos szemléletű megközelítéssel tanítsuk. Ezeknek a módszereknek a lényege, az adott tématerületen belüli hatékonyságnövelésen túl, hogy széles körben támogatja a tudástranszfert az informatikán belüli és a valódi tantárgyközi kapcsolatok megvalósításában.

Mindenképpen fontos kiemelni, hogy a fiktív és a valódi adatfeldolgozás között az alapvető különbség, hogy míg az első esetben az eszköz megtanítása a cél, addig a második esetben a problémamegoldás a cél, míg a szoftver csak egy eszköz, amit használhatunk a cél megvalósítása érdekében [13].

Kutatási eredmények egyértelműen mutatják, hogy az oktatási segédanyagoknak már a bevezető szakasztól kezdődően fel kellene vállalnia a valódi, adekvát adattartalmakat, amelyek egyrészt sokkal inkább hitelessé teszik a tanítási folyamatot, másrészt jóval magasabb motivációs erejük van, mint a színlelt adatfeldolgozásnak. A valós tartalmak felkeltik a tanulóknak az érdeklődését, és a szoftvereket – jelen szövegkörnyezetben a táblázatkezelő programokat – eszközként fogják használni az adatfeldolgozási problémáik megoldásához.

A táblázatkezelés is problémamegoldás? Igen, amennyiben szakítunk a felületi megközelítésekkel. Helyettük, a röviden bemutatott (3. fejezet), magas-mathability megoldásokat helyezzük előtérbe, a TPCK valamennyi aspektusának szem előtt tartásával, a sémaépítésen alapuló gyors és lassú gondolkodási módok megfelelő helyeken történő alkalmazásával, valamint a Meaning System Model hatékony tanári megközelítését támogatva és alkalmazva a gyakorlatban.

Irodalom

1. NAT 2012 (2012) 110/2012. (VI. 4.) Korm. rendelete a Nemzeti alaptanterv kiadásáról, be-vezetéséről és alkalmazásáról. http://ofi.hu/sites/default/files/attachments/mk_nat_20121.pdf (Letöltve: 2018. június)
2. OFI (2013) Kerettanterv az általános iskola 5–8. évfolyamára. Kötelező tantárgyak 2.2.15 http://kerettanterv.ofi.hu/02_melleklet_5-8/2.2.15_informat_5-8.doc (Letöltve: 2018. július)
3. OFI (2013) Kerettanterv a gimnáziumok 9–12. évfolyama számára. Kötelező tantárgyak 3.2.16 http://kerettanterv.ofi.hu/03_melleklet_9-12/3.2.16_informat_9-12.doc (Letöltve: 2018. július)
4. T. Bell, H. Newton: Unplugging Computer Science. *Improving Computer Science Education*, Routledge (2013).
5. M. Gove: *Michael Gove speech at the BETT Show 2012*. Published 13 January 2012. Digital literacy campaign. http://www.theguardian.com/education/2012/jan/11/digital-literacy-michael-gove-speech. (Letöltve: 2018. július) (2012)
6. R. R. Panko: The Cognitive Science of Spreadsheet Errors: Why Thinking is Bad. *Proceedings of the 46th Hawaii International Conference on System Sciences*, January 7-10, 2013, Maui, Hawaii. (2013)
7. E. Soloway: Should we teach students to program? *Communications of the ACM*, 36(10), 21–24. DOI=://doi.org/10.1145/163430.164061. (1993)
8. M. Ben-Ari: Non-myths about programming. *Communications of the ACM*, 54(7), 35. DOI= http://doi.org/10.1145/1965724.1965738. (2011)
9. P. Mishra, M. J. Koehler: Technological Pedagogical ContentKnowledge: *A Framework for Teacher Knowledge. Teachers College Record.* Volume 108, Number 6, June 2006, pp. 1017–1054. (2006)
10. J. A. Chen, D. B. Morris, N. Mansour: Science Teachers' Beliefs. Perceptions of Efficacy and the Nature of Scientific Knowledge and Knowing. In *International Handbook of Research on Teachers' Beliefs.* (Eds.) Fives, H. & Gill, M. G. Routledge, 370–386. (2015)
11. J.J.G. Merriënboer, J. Sweller: Cognitive Load Theory and Complex Learning: *Recent Deve-lopments and Future Directions. Educational Psychology Review*, 17(2), 147–177. (2005)
12. R. R. Skemp: A matematikatanulás pszichológiája, Edge 2000 Kiadó, Budapest. (1975)
13. P. Baranyi, A. Gilanyi: Mathability: Emulating and enhancing human mathematical capabilities. In 2013 IEEE 4th International Conference on Cognitive Infocommu-nications (CogInfoCom) (pp. 555–558). (2013)
14. P. Biró, M. Csernoch: The mathability of spreadsheet tools. In 2015 6th IEEE International Conference on Cognitive Infocommunications (CogInfoCom) (pp. 105–110). DOI= http://doi.org/10.1109/CogInfoCom.2015.7390573. (2015)
15. D. Kahneman: *Thinking, Fast and Slow*, New York: Farrar, Straus; Giroux. (2011)
16. M. Csernoch: Thinking Fast and Slow in Computer Problem Solving, Journal of Software Engineering and Applications. Vol.10 No. 01(2017), Article ID:73749, 30 pages 10.4236/jsea.2017.101002. (2017)
17. T. Nagy, M. Csernoch: The paradox of the hungarian frame curricula in informatics. *The Turkish Online Journal of Educational Technology, INTE 2018*. (accepted)
18. C. Angeli: Teaching Spreadsheets: A TPCK Perspective. In *Improving Computer Science Education*. (Eds.) D. M. Kadijevich, C. Angeli, and C. Schulte. Routledge. (2013)
19. J. Walkenbach: *Microsoft Excel 2010 Bible*. Wiley Publishing, Inc. Indianapolis, (2010) 202
20. K. Freiermuth, J. Hromkovič, B. Steffen: Creating and Testing Textbooks for Secondary Schools. In: R. T. Mittermeir, M. M. Sysło (Eds.), *Informatics Education - Supporting Computational Thinking*. Berlin Heidelberg, Germany: Springer (2008) 216–228 http://link.springer.com/chapter/10.1007/978-3-540-69924-8_20, pp. 219. (Letöltve: 2016. június 1.)
21. P. Biró, M. Csernoch: Deep and surface metacognitive processes in non-traditional programming tasks. 2014 5th IEEE Conference on Cognitive Infocommunications (CogInfoCom). DOI: 10.1109/CogInfoCom.2014.7020507. https://ieeexplore.ieee.org/document/7020507. (Letöltve: 2018. szeptember 12.) (2015)
22. Csernoch Mária: *Programozás táblázatkezelő függvényekkel – Sprego*. Műszaki Könyvkiadó, Budapest. (2014)

23. Gy. Pólya: *How To Solve It. A New Aspect of Mathematical Method.* Second edition. Princeton University Press, Princeton, New Jersey. (1957)
24. S. Booth: *Learning to program: A phenomenographic perspective*. Gothenburg, Sweden: Acta Universitatis Gothoburgensis. (1992)
25. Csernoch Mária, Biró Piroska: Számítógépes problémamegoldás, *TMT, Tudományos és Műszaki Tájékoztatás, Könyvtár- és információtudományi szakfolyóirat*, (62) 3, 86–94. (2015)
26. P. Sestoft: *Spreadsheet technology*. Version 0.12 of 2012-01-31. IT University Technical Report ITU-TR-2011-142. IT University of Copenhagen, December 2011. (2011)
27. M. Csernoch, E. Dani: Data-structure validator: an application of the HY-DE model. *8th CogInfoCom, Debrecen,* 2017, pp. 197–202, ISBN: 978-1-5386-1264-4, IEEE. (2017)
28. P. Biró, M. Csernoch: Semi-Unplugged Tools for Building Algorithms with Sprego. TOJET: Turkish Online Journal of Educational Technology Spec. Issue for INTE:(2) pp. 946–957. (2017)
29. P. Biró, M. Csernoch: Unplugged tools for building algorithms with Sprego. *International Conference on Education and New Development*, Lisbon, Portugal, June 2017. (2017)
30. G. Csapó: Sprego Virtual Collaboration Space. *8th IEEE International Conference on Cognitive Infocommunications*, CogInfoCom 2017 (ed. Péter Baranyi, Anna Esposito, Péter Földesi, Tamás Mihálydeák), pp. 137-142. http://ieeexplore.ieee.org/document/8268230/. ISBN 978-1-5386-1264-4. DOI=10.1109/CogInfoCom.2017.8268230. (2017)
31. G. Csapó: Sprego Virtual Collaboration Space: Improvement Guidelines for the MaxWhere Seminar System. *8th IEEE International Conference on Cognitive Infocommunications*, CogInfoCom 2017 (ed. Péter Baranyi, Anna Esposito, Péter Földesi, Tamás Mihálydeák), pp. 143-144. http://ieeexplore.ieee.org/document/8268231/. ISBN 978-1-5386-1264-4. DOI= 10.1109/CogInfoCom.2017.8268231. (2017)
32. G. Csapó, K. Sebestyén: Educational Software for the Sprego Method. *The Turkish Online Journal of Educational Technology*, INTE 2017 október, pp. 986-999. http://www.tojet.net/special/2017_10_1.pdf. ISSN 2146-7242. (2017)
33. K. Sebestyén, G. Csapó, M. Csernoch: Visualising Sprego Inequality Problems With 2D Representations. *The Turkish Online Journal of Educational Technology*, INTE 2018 november (2), pp. 888-898. http://www.tojet.net/special/2018_12_3.pdf. ISSN 2146-7242. (2018)
34. Á. Gulácsi, N. Dienes: 3D Software Environment for Educational Sprego Programming. *The Turkish Online Journal of Educational Technology*, INTE 2018 november (2), pp. 837-844. http://www.tojet.net/special/2018_12_3.pdf. ISSN 2146-7242. (2018)

Hallgatói teljesítményértékelés az algoritmikus gondolkodás tükrében

Pluhár Zsuzsa[1], Torma Hajnalka[2], Törley Gábor[3]

{[1]pluharzs, [2]thajni, [3]pezsgo}@inf.elte.hu
ELTE IK

Absztrakt. Az informatikai gondolkodás (computational thinking) egyik alapköve az algoritmikus gondolkodás. Ennek mérésére, fejlesztésére több irányzat is létezik. Egyik igen komoly és sokrétű elgondolás a számítógép nélkül, a gondolkodási sémákon keresztül történő megközelítés. Ilyen a nemzetközi Bebras kezdeményezés is. Kutatásunkban az Eötvös Loránd Tudományegyetem Informatika Karának angol nyelvű BSc-s hallgatóin keresztül arra kerestük a választ, hogy az első szemeszter tantárgyainak hatására mennyire változik a hallgatók algoritmikus gondolkodása, illetve az alapozó programozást támogató (Programming) kurzuson nyújtott teljesítmény milyen összefüggéseket mutat ezen változásokkal. A felméréshez a Bebras kezdeményezés feladatainak segítségével készítettük el felmérésünket, melynek alapjait és elsődleges eredményeit a cikkünkben foglaltuk össze.

Kulcsszavak: algoritmikus gondolkodás, programozás oktatás, felsőoktatás

1. Algoritmikus gondolkodás

Az informatikai gondolkodás (Computational Thinking) kifejezést Jeanette Wing[1] definiálta újjá és terjesztette el , majd 2010-ben újragondolta, és megalkotta a ma talán leggyakrabban idézett definíciót:

> „*Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent*" [2/1.o.]

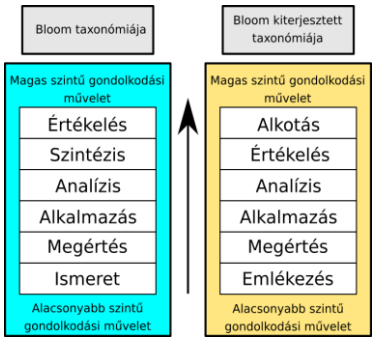
Wing és az őt követő kutatók definíciói ugyan nem egységesek, de az általuk megfogalmazott meghatározások sokszínűségében bizonyos területek egyaránt kiemelkednek. Ilyen a logikus gondolkodás, a problémamegoldás, az algoritmikus gondolkodás, az elemzés, rendszertervezés, általánosítás, és az egyik legfontosabb elem az absztrakció képessége, amely lehetővé teszi a komplex problémák megoldását is.

Selby[3] értelmezésében a következő képességek szükségesek az informatikai gondolkodáshoz: absztrakt fogalmakban gondolkodás, részekre bontás a gondolkodás során, algoritmikus gondolkodás, értékelésben való gondolkodás és általánosítás képessége a gondolkodás során. Selby[3] és munkatársainak munkássága azért is jelentős, mivel kimondták, hogy ahhoz, hogy értékelni lehessen, hogy mennyire fejlődött a tanulók informatikai gondolkodása, először fontos az informatikai gondolkodást alkotó elemek azonosítása.

1.1. Az algoritmikus gondolkodás és szintjei

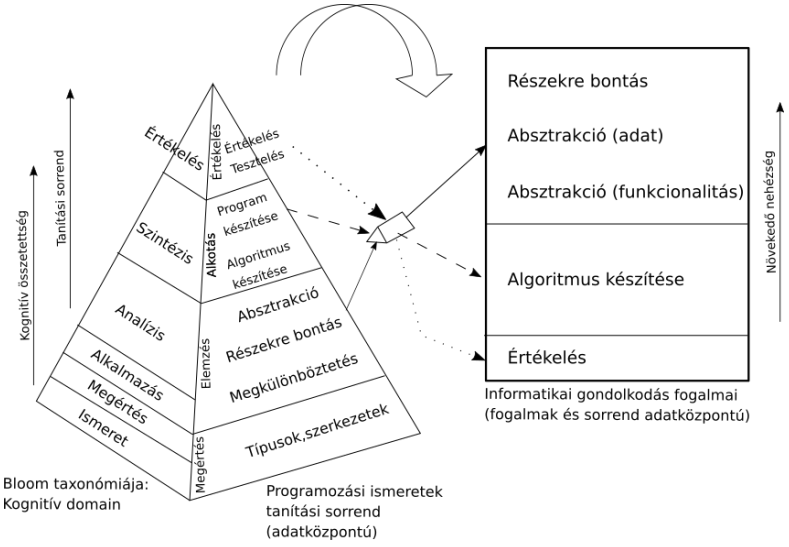
Az algoritmus egy probléma megoldásához vezető lépések sorozata. Az algoritmusok jelen vannak a hétköznapjainkban, de algoritmusokat használunk informatikai problémák megoldásakor is. Az algoritmikus gondolkodás kompetenciája egy fontos informatikai kompetencia, definiálására, szintekre bontására több törekvést ismerünk. Zsakó és Szlávi [4] szerint az algoritmikus gondolkodás szintjei az algoritmus felismerése, megértése, az algoritmus végrehajtása, az algoritmus elemzése, algoritmus alkotása és az algoritmus megvalósítása. Ez a felosztás összhangban van Bloom oktatási

célokat osztályozó taxonómiájával [5], illetve a módosított Bloom-i taxonómiával (1. ábra). A kiterjesztett Bloom-i taxonómiában [6] a kognitív folyamatok dimenziója az emlékezés, megértés, alkalmazás, elemzés, értékelés és alkotás szintekből áll. Jelen munkánk során a Bloom kiterjesztett taxonómiájában található kognitív szinteket vettük alapul.



1. ábra: Bloom eredeti és kiterjesztett taxonómiája

Selby az informatikai gondolkodás fejlesztése kapcsán a programozás oktatásával is foglalkozott. A Bloom-i taxonómiát vetítette le a programozás és az informatikai gondolkodás elemeinek oktatására. Az általa készített ábrákon a tanulási sorrendre koncentrált, mivel lehetnek olyan elvárások, amelyeknek egy tanuló aktuális tudásállapotban nem képes megfelelni a kognitív összetettség miatt, ugyanakkor az informatikai gondolkodásban esetlegesen egyszerűbb lehet a felsőbb szintű művelet, mint egy alsóbb szintű. [7] A 2. ábra azt mutatja, hogyan viszonyulnak egymáshoz Bloom taxonómiájának kognitív szintjei, az (adatközpontú) programozásoktatás tanítási sorrendje, és az informatikai gondolkodás fogalmainak tanítási sorrendje. Az ábrán jól látható, hogy a programozás-oktatás és az informatikai gondolkodás fogalmainak tanítási sorrendje nem egyezik meg.



2. ábra: Bloom taxonómiája, a programozási ismeretek és informatikai gondolkodás fogalmainak tanítása közötti kapcsolat

2. Algoritmikus gondolkodás fejlesztése

Az informatikai gondolkodás, illetve ezen belül az algoritmikus gondolkodásának fejlesztésére több kezdeményezés is kialakult. Ezek közül három főbb csapásirányt különböztethetünk meg:

Az első valamely kiemelt részterület más ismeretkörbe való beépítésével, általában projektmunka keretében épít be bizonyos célzott tevékenységeket.

A második irányzatban az informatikai gondolkodást programozás oktatásával, szimulációs játékok alkalmazásával fejlesztik és vizsgálják. Ezek között találhatunk új fejlesztéseket játékok, keretrendszerek tervezésével és megvalósításával, vagy meglévő alkalmazások integrálását [pl. 8, 9, 10].

A harmadik megközelítés a programozástól, esetleg a számítógéptől is elszakadva konkrét aktivitásokon keresztül valósítja meg a fejlesztést [pl. 11, 12, 13, 14, 15, 16].

2.1. Hód

Ezt a harmadik megközelítést tartja szem előtt a nemzetközi Bebras kezdeményezés [13] és ennek magyar megvalósulása a Hód [14], melynek informatikai gondolkodással kapcsolatos fejlesztés melletti célja, hogy

- felkeltse az érdeklődést az informatika iránt;
- feloldja az informatikával kapcsolatos félelmeket, negatív érzéseket;
- megmutassa az informatika területének sokszínűségét, felhasználási lehetőségeit és területeit.

A kezdeményezés elsődleges terepe egy verseny, melyhez a feladatokat a nemzetközi csoport dolgozza ki - és melyek megoldásához csak strukturált és logikus gondolkodásra van szükség, semmilyen különleges informatikai tudás nem szükséges a megválaszolásukhoz. A feladatok érdekes problémákat mutatnak be. Nem tesztek, inkább szórakoztató gondolkodtató feladványok, melyek át- és továbbgondolásával új ismeretekre tehetnek szert a résztvevők, illetve meglévő ismereteiket mélyíthetik el.

A feladatok előkészítését, pontosítását a nemzetközi csapat egy műhelykonferencia keretein belül végzi, kiválogatva és módosítva a résztvevő országokból beküldött kérdéseket. Ezután az egyes országok honosítják a kérdéseket és a náluk megrendezett versenyhez testre szabják azokat.

Magyarországon az ELTE Informatika Karával 2011-ben csatlakoztunk a kezdeményezéshez, és azóta évről évre megszervezzük a megmérettetést.

A versenyen szereplő kérdések egy jelentős része algoritmikus gondolkodással kapcsolatos feladatokat rejt magában.

2.2. Programozás oktatása

Az ELTE Informatikai Karának angol nyelvű Computer Science BSc képzésében részt vevő hallgatók az első félév során vesznek részt a Programming elnevezésű kurzuson. A kurzus 2 óra előadást és 3 óra laborgyakorlatot foglal magában hetente. A tantárgy, a magyar nyelvű megfelelőjéhez hasonlóan, programozás-módszertani bevezetésként szolgál. A kurzus célja az alapvető problémamegoldó algoritmusok megismerése és használata különböző egyszerű és összetettebb adatszerkezetekkel. A megoldandó problémákra elsődlegesen egy algoritmus-leíró nyelven vár el megoldást, majd ezt követi az algoritmus kódolása C++ programozási nyelven. Olyan általános sémákat igyekszik a hallgatók számára elérhetővé tenni, amely segítheti őket a problémák megértésében, a problémák részekre bontásában, az algoritmusok megalkotásában, majd végül a programok elkészítésében.

A korábbi évek tapasztalata alapján az angol nyelvű BSc képzésben résztvevő hallgatók számára a kurzus nem egyforma nehézségű, sokaknak már az egyszerű algoritmusok megértése, felhasználása, majd később nehezebb feladatokban történő alkalmazása problémákat okoz. A bonyolultabb algoritmusok, illetve a komplexebb bemeneti adatstruktúrával rendelkező problémák pedig óriási

kihívást jelentenek számukra. Ezek a tapasztalatok visszavezetnek Selby [7] megállapításaihoz az informatikai, illetve algoritmikus gondolkodás szintjeinek és a tanítási nehézség kapcsolatáról (lásd újra 2. ábra).

3. Kutatási célok és eszközök

Kutatásunkban célul tűztük ki, hogy megvizsgáljuk, mutatkozik-e kapcsolat a hallgatók BSc képzésünkbe való belépésekor mutatott algoritmikus gondolkodási szintje, illetve a tárgy teljesítése/teljesíthetősége között.

Ha van valamilyen kapcsolat, akkor az informatikai gondolkodás mely dimenziói szükségesek a sikeres hallgatói előmenetelhez, illetve tapasztalható-e változás a hallgatók informatikai gondolkodásában az egyetemi tanulmányaik előrehaladtával. További vizsgálati kérdés, hogy milyen egyéb tényezők hatnak a teljesítési szintekre: kimutatható-e bármilyen kapcsolat az angol nyelv ismerete, az előtanulmányok, illetve a különböző gondolkodási szintek megléte között.

3.1. Kérdőív

A vizsgálathoz és a változások követéséhez elő- és utókérdőív változatot készítettünk. Az összekötésekhez a hallgatói azonosítókat használtuk, de a kérdőívben egyértelműen leírtuk, hogy azok eredménye semmilyen módon nem befolyásolja a kurzuseredményeiket.

Az előkérdőív tartalmazta a szükséges háttérváltozókra vonatkozó kérdéseinket. Önbevallás alapján felmértük, hogy a hallgatók milyennek értékelik a saját angol nyelvtudásukat, illetve, hogy milyen sokat/mélyen tanultak algoritmizálást, algoritmus-elméletet. Rákérdeztünk továbbá néhány programozási nyelv/környezet (C, C++, PHP, Javascript, HTML5/CSS, Java, Scratch) előismeretének mértékére. A nyelvek kiválasztásának fő szempontja az volt, hogy általában ezek a legelterjedtebb nyelvek a közoktatásban, valamint a fenti nyelvekkel fognak találkozni a hallgatók az egyetemi képzésük folyamán. Az önértékeléshez 5 fokú Likert-skálát állítottunk össze a választható fokozatokhoz. Emellett rákérdeztünk a nemükre és nemzetiségükre.

Az előkérdőív második felében és az utókérdőívben a Hód kérdésekből [17] válogattunk a kiterjesztett Bloom taxonómiának megfelelő szintenként 2-2 kérdést. A kiválasztás során arra törekedtünk, hogy az életkori és nehézségi paraméterek minél közelebb álljanak a résztvevőkhöz, az idősebb korosztálynak (10-12. osztály) megfelelő szintekről a nem egyértelműen nagyon könnyű feladatok közül kerültek ki a feladatok. Válogatási szempontot jelentett az is, hogy a különböző országokból, kultúrákból érkezők számára a feladatban szereplő szituációk, történetek ne jelentsenek előnyt vagy hátrányt a megoldás során.

Szint	Előkérdőív	Utókérdőív
Alkalmazás (applying)	Számfordító (2012-HU-01a) Pöttyök és parancsok (2013-SK-09)	Az utókérdőív kérdései még nem publikusak annak lefolytatásáig.
Analizálás (analyzing)	Alakzatjáték (2016-CA-09) Véletlen képek (2013-DE-02)	
Szintézis (evaluating, synthesis)	Játék a golyókkal (2016-IT-02B) Ebéd (2015-DE-06)	
Alkotás (Creating)	Fogpiszkálós (2017-HU-06) Kalózvadászat (2015-SI-07)	

1. táblázat: A kérdőívben szereplő feladatok és bloom-i kognitív domain szintjeik

A kérdéseket Google űrlapként készítettük el. Nem használtuk ki az általános kiértékeléseket, illetve a kitöltők nem kaptak visszajelzést a megoldásuk helyességéről.

Általánosan meghagytuk azt a lehetőséget, hogy négy lehetséges válasz közül kelljen kijelölniük az általuk helyesnek ítéltet. Ez több célt szolgált: az egyik, hogy minél pontosabban össze tudjuk majd vetni a feladatok megoldásainak sikerességét a Hód versenyeken elért eredményekkel. Az egyes „rossz" megoldások megadásánál törekedtünk arra, hogy téves gondolkodási folyamatokat ábrázoljanak, és így ezeket is ki tudjuk értékelni, nyomon tudjuk követni. Továbbá a nyelvi nehézségeket a megfogalmazások területén és az elgépelések lehetőségét is igyekeztünk így a minimálisra csökkenteni.

4. Elsődleges eredmények

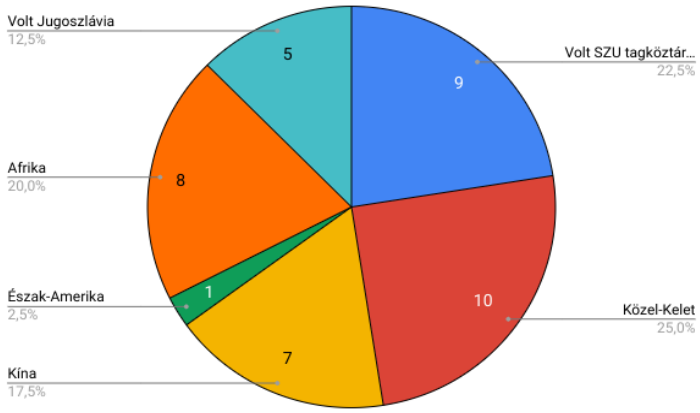
A kutatás jelenlegi fázisában az előkérdőívek kitöltésén vagyunk túl. Az ehhez köthető statisztikai eredményeket foglaljuk össze a következőkben.

Az előkérdőív kitöltéséhez nem szabtunk időkorlátot és önálló munkaként, úgymond bárhonnan kitölthették a kérdőívek a szemeszter első 3 hetében.

4.1. Minta

Az előkérdőívet a Programming alapozó kurzuson résztvevő hallgatók 60%-a (N=42) töltötte ki. A külföldi hallgatói arányainkkal megegyezően 12 nő és 30 férfi.

A hallgatók nemzetiségének eloszlását az 3. ábra mutatja.

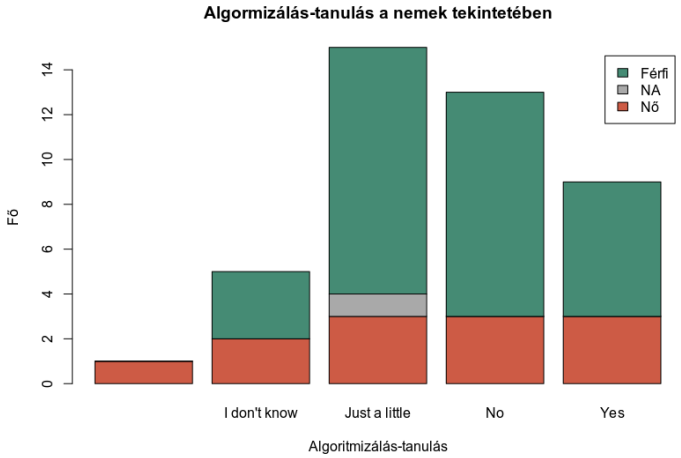


3. ábra: A mintában szereplő hallgatók nemzetiségei

4.2. Háttérváltozók alakulása

Angol tudását tekintve a hallgatók 7%-a erősebb alapfoknak, 32%-a közepesnek, 59%-a jónak értékelte azt.

A megkérdezett hallgatók mindössze 21%-a állította magáról, hogy tanult korábban algoritmusokról, és 41%-uk azt, hogy nem tudja vagy egyértelműen nincs előismerete a témát illetően (4. ábra). Kimondhatjuk, hogy a programozás elméleti alapjainak tanulása a megkérdezett hallgatók jelentős többsége számára új tananyag.

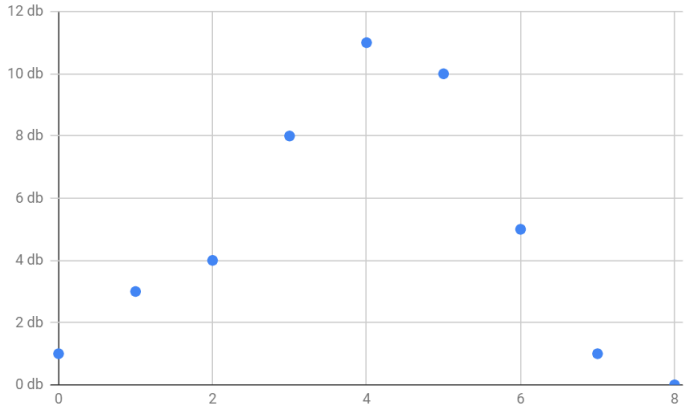


4. ábra: A hallgatók válaszai a korábbi algoritmizálás-tanulásról a nemek eloszlásában

A programozási nyelvek és környezetek területén a C++ nyelv ismerete volt a hallgatók számára a legerőteljesebb: 30%-uk vallotta, hogy segítséggel képes egyszerű programot írni benne, és 25%-uk, hogy képes önállóan programot is írni ezen a nyelven, de csupán 5%-uk nevezte magát képzettnek. A C és a HTML nyelvek kerültek a második legismertebb kategóriába úgy, hogy a hallgatók 30%-a nem is hallott róluk és 33%-uk éppen csak említés szintjén. A legismeretlenebb nyelv a PHP volt, a hallgatók 84%-a nem is hallott róla.

4.3. Hód kérdések eredményei

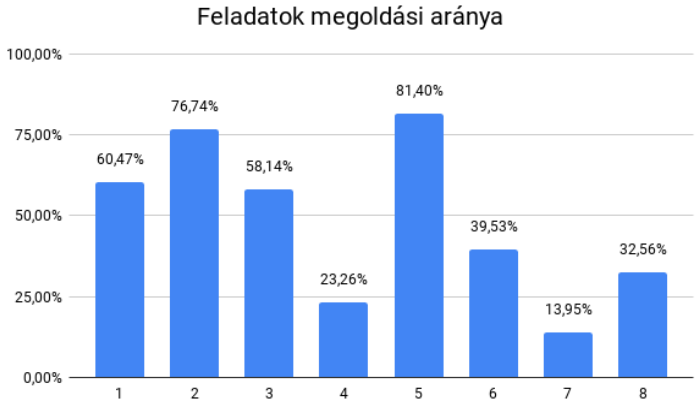
A hallgatóknak minden alkategóriából kettő, azaz összesen nyolc feladatot kellett megoldaniuk. Helyes válasz esetén ezekre 1-1 pontot adtunk, így maximum nyolc pontot lehetett elérni. A hallgatók átlagos pontszáma 3,86 volt. A hallgatók 72%-a csak 2, 3 vagy 4 pontot szerzett.



5. ábra: Hallgatók által elért pontok számának alakulása

A feladatok nehézség szerint ugyan közel azonosként kerültek kiválasztásra, de az elért eredményeket tekintve (ld. 5. ábra) megállapíthatjuk, hogy a 4. (2013-DE-02 Véletlen képek) és a 7. (2017-HU-06 Fogpiszkálós) feladat nehezebbnek bizonyult. Ezeket a feladatokat hallgatók kevesebb, mint negyede tudta megválaszolni. Mindkét feladat esetében kiemelhető egy jellemzően választott (hallgatók 41%-a), helytelen megoldási stratégiát mutató helytelen válasz. A „Véletlen képek” feladatban

egy teljesen egyértelműen megfelelő válasz került kiválasztásra. Ennek oka lehet az, hogy a feladat egy tagadással kérdezett rá a megoldásra („melyik ábra NEM nyomtatható…"), amit a hallgatók nem vettek figyelembe.

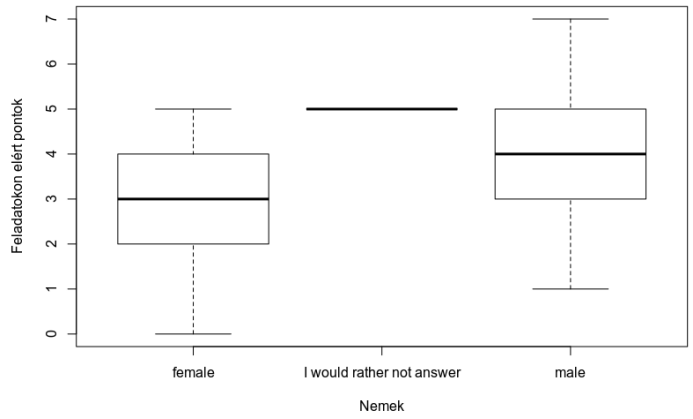


6. ábra: Az egyes feladatok megválaszolásának alakulása

Az 5. feladat kivételével egyértelműen meghatározható, hogy az algoritmikus gondolkodás komplexebb szintjeihez tartozó feladatok általában nehezebbeknek bizonyultak.

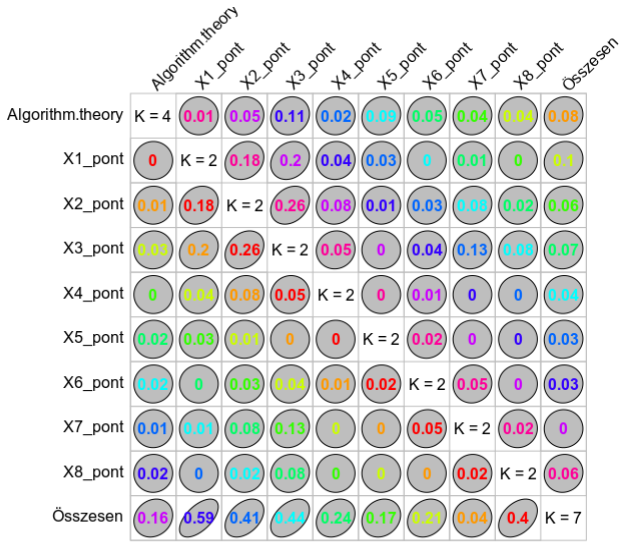
4.4. Összefüggések

Az előkérdőívben szereplő háttérváltozók és az algoritmikus gondolkodást vizsgáló kérdések közötti összefüggés-vizsgálatok csak a nem esetében mutattak gyenge összefüggést (lásd 7. ábra). A férfiak jobb eredményt értek el, mint a nők (p=0.03, átlagok különbsége: 1,16).



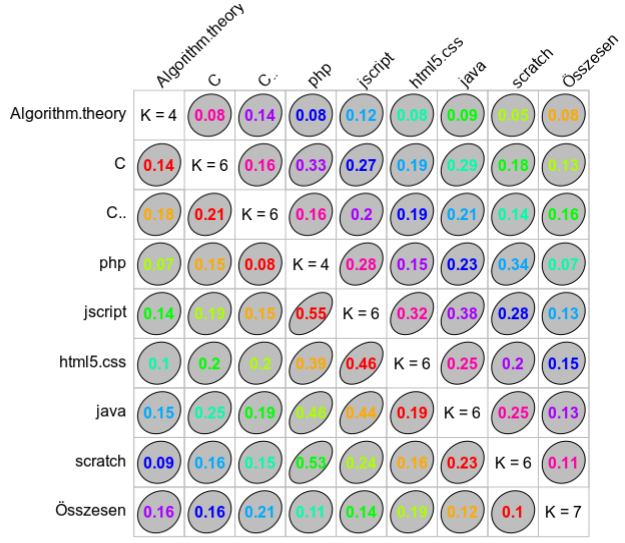
7. ábra: A nemek és az elért összpontszámok összefüggése

Az algoritmizálás korábbi tanulására adott válaszok nem állnak összefüggésben a kapott pontszámokkal (8. ábra).



8. ábra: Az algoritmus-tanulás és az elért összpontszámok összefüggése (Goodman-Kruskal tau számítás)

A korábban tanult nyelvekre adott válaszok nem állnak összefüggésben azzal, hogy valaki tanult-e algoritmus-elméletet, illetve hány pontot ért el az algoritmikus gondolkodás szintjét mérő feladatokon (9. ábra).



9. ábra: Az algoritmus-tanulás, a tanult nyelvek és az elért összpontszámok összefüggése (Goodman-Kruskal tau számítás)

Összességében elmondható, hogy az előkérdőív alapján a vizsgált változók és az elért pontszám között összefüggés nem mutatható ki.

5. Konklúzió

A felmérésünk célja az volt, hogy képet alkothassunk az ELTE Informatikai Karának Programming kurzusára járó hallgatók előképzettségéről, és a belépéskori algoritmikus gondolkodás szintjéről. Az előkérdőív alapján a vizsgált változók és az algoritmikus gondolkodást felmérő feladatokra kapott pontszám között összefüggést nem találtunk.

Jövőbeli terveink szerint összevetjük az algoritmikus gondolkodást felmérő feladatokon elért pontszámokat a tanulók kurzusteljesítési adataival. Továbbá az utókérdőív kitöltését követően elemzéseink kiterjeszthetőek lesznek, mélyebb összefüggés vizsgálatokat is el tudunk majd végezni. A jelenlegi felmérés lezárásával az eredmények összehasonlíthatóak lesznek a Hód verseny eredményeivel.

A későbbiekben érdemesnek tartjuk a vizsgálatunkat több kérdéssel is kiegészíteni az egyes algoritmikus gondolkodási kategóriákban – ezzel erősítve a kapott eredményeket. Illetve a kapott mutatók tekintetében kimondottan algoritmikus gondolkodásra irányuló tevékenységekkel bővíteni a programozó képzésünket.

Köszönetnyilvánítás

EFOP-3.6.1-16-2016-00023: Kutatás-fejlesztési tevékenység megvalósítása az Eötvös Loránd Tudományegyetem szombathelyi kampuszán – A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Irodalom

1. Wing, J. (2006). Computational thinking. Commun. ACM, 49, 33-35.
2. Wing, J. (2011). Research Notebook: Computational Thinking - What and Why? The Link. Pittsburgh, PA: Carneige Mellon. - elérhető: https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf (utoljára megtekintve: 2018.10.15.)
3. Selby C. C. (2013) Computational Thinking: The Developing Defenition. Submitted for ItiCSE Conference 2013. - elérhető: http://people.cs.vt.edu/~kafura/CS6604/Papers/CT-Developing-Definition.pdf (utoljára megtekintve: 2018.10.15.)
4. Zsakó, L., Szlávi P. (2010): Informatikai kompetenciák: Algoritmikus gondolkodás. InfoDidact 2010. - elérhető: https://people.inf.elte.hu/szlavi/InfoDidact10/Manuscripts/ZsL_SzP.htm (utoljára megtekintve: 2018.11.07.)
5. Bloom, B.S., Krathwohl, D. R. (1956) Taxonomy of Educational Objectives: The Classification of Educational Goals, by a committee of college and university examiners. Handbook I: Cognitive Domain. NY, NY: Longmans, Green
6. Anderson, L. W., Krathwohl, D. R., et al (Eds.) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. Allyn & Bacon. Boston, MA (Pearson Education Group)
7. Selby C. C. (2015) Relationships: computational thinking, pedagogy of programming, and Bloom's Taxonomy. In Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE '15). ACM, New York, NY, USA, 80-87. DOI=http://dx.doi.org/10.1145/2818314.2818315
8. Brennan, K, Resnick, M. (2012) New frameworks for studying and assessing the development of

computational thinking, AREA.

9. Brennan, K. (2011) Creative computing: A design-based introduction to computational thinking. – elérhető: http://scratched.media.mit.edu/sites/default/files/CurriculumGuide-v20110923.pdf (utoljára megtekintve: 2016. 10. 25.)
10. Aiken, J. M., Caballero, M. D., Douglas, S. S., Burk, J. B., Scanlon, E. M, Thoms, B. és Schatz, M. F. (2012) Understanding Student Computational Thinking with Computational Modeling, PERC Proceedings.
11. Bell, T., Witten, I. H., Fellows, M. (2010) Computer Sciene Unplugged. – elérhető: http://csunplugged.org/books (utoljára megtekintve: 2016. 10. 25.)
12. Dagiene, V. (2006) Information technology contests – introduction to computer science in a attractive way, Informatics in Education, 5. 1.s., 37–46.
13. Cartelli, A., Dagiene, A., Futschek, G. (2010) Bebras Contest and Digital Competence Assessment: Analysis of Frameworks. International Journal of Digital Literacy and Digital Competence. Január-Március. 24-39.
14. Pluhár, Z., Gellér, B. (2018) International Informatic Challenge in Hungary. In: Teaching and Learning in a Digital World : Proceedings of the 20th International Conference on Interactive Collaborative Learning. Berlin, Germany: Springer, .pp. 425-435.
15. CS Unplugged weboldal. Elérhető: http://csunplugged.org (utoljára megtekintve: 2018.11.10.)
16. Computer Science for Fun weboldal. Elérhető: http://cs4fun.org (utoljára megtekintve: 2018.11.10.)
17. Magyar e-hód feladatok archívuma: http://e-hod.elte.hu/archivum (utoljára megtekintve:2018.11.10.)

Az R-rel szebb a statisztika

Pšenák Péter[1], Pšenáková Ildikó[2]

{[1]petkoneo@gmail.com, [2]ildiko.psenakova@gmail.com
[1]Univerzita Komenského Bratislava, [2]Trnavská univerzita v Trnave

Absztrakt. A pedagógiai gyakorlatban sokszor szükséges különböző statisztikai számítások, elemzések használata. Mivel legtöbbször csak néhány adatról van szó a pedagógusok az MS Excel vagy a Calc programot használják. Ha azonban sok adatok kell feldolgozni, más rendszer után kell nézni. Itt jöhet képbe az RStudio statisztikai program, amely nemcsak a sok adat feldolgozásával tud megbirkózni, de hipotézisek tesztelésében, varianciaanalízis számításában és más statisztikai tesztek végrehajtásában is segítségére lehet a pedagógusnak.

Kulcsszavak: matematikai szoftver, RStudio, statisztika, oktatás

1. Bevezetés

A statisztikai számítások nem tartoznak sem a diákok, sem a legtöbb tanár kedvenc időtöltése közé. Ha munkájukban mégsem tudják ezeket elkerülni, leggyakrabban az általuk legjobban ismert szoftvert használják, és azzal igyekeznek minél előbb elvégezni a szükséges számításokat. A leggyakrabban ez az MS Excel, amely szinte már szabvánnyá vált, mert az irodai programcsomag (MS Office) részeként valamelyik verziója szinte bármely munkahelyen megtalálható.

Az iskolarendszerben is az MS Office használata az elfogadott és már az általános iskola alsó tagozatán kezdik oktatni az szövegszerkesztőt (MS Word). Később társul hozzá a táblázatkezelő (MS Excel) és a prezentációkészítő (MS Power Point) is. A középiskolákon folytatódik ez a trend és aprólékosabban oktatják a programcsomag részeinek használatát. Ebből is látható, hogy gyakorlatilag a diákok szinte az iskolai éveik alatt végig ezzel az „irodai csomaggal" ismerkednek, és egyes programjaival dolgoznak. Ebből kifolyólag természetes az is, hogy ha a tanárok ezeket oktatják, akkor ők is ezeket használják.

Mivel az MS Excel tartalmaz statisztikai funkciókat is, egyértelmű, hogy felhasználói elsősorban ezeket fogják szükség esetén használni. De mi tévők legyünk akkor, ha az Excel funkciói nem tartalmazzák a számításhoz megfelelőt, vagy túl sok adatot kell feldolgozni? Ebben az esetben más programot kell használni, amely speciálisan statisztikai számításokra orientált. Ilyen például az R és környezete az RStudio is.

2. Statisztika használata a pedagógiai kutatásban

A pedagógiai kutatás a kutatási tevékenységeknek egy specifikus típusa. Ez egy célzott és szándékos analitikus-szintetikus tevékenység, amely megfelelő módszertani eszközökkel és kutatási technikákkal tanulmányozza az objektív (tárgyilagos) pedagógiai (oktatási) valóságot. Célja, hogy új tudományos ismereteket szerezzen a pedagógiai valóság különféle jelenségeiről, tulajdonságairól, jellemzőiről és más nézőpontjairól. Eredményeként általános törvények, ismeretek vagy tudományos pedagógiai elméletek fogalmazódnak meg, melyek hozzájárulnak a pedagógiai elmélet és gyakorlat fejlődéséhez.

A pedagógiai kutatás *sajátosságai* az oktatási valóság bonyolultságából, sokrétűségéből és a pedagógia és egyéb társadalomtudományok pontatlanságából adódnak. Ezek közé sorolhatjuk a következőket:

- a pedagógiai jelenségek nem mérhetők pontosan (pl. a tanár és a tanuló oktatási tevékenységének jellemzői, eredményei, a tanár és a diák részének aránya az elért eredményekben, ...),
- a pedagógiai jelenségek dinamikusak és megismételhetetlenek – a tulajdonságok, képességek, feltételek fejlődnek és változnak, ezért szinte lehetetlen, hogy egy módszert teljesen azonos feltételekkel tudjunk használni kétszer (pl. ismételt vizsgálatoknál a kutatás csak közel azonos; a válaszadók és az eljárás lehet ugyanaz, de a kutatás során sok megismételhetetlen tényező is fennáll (pl. fizikai és lelki állapot, élmények, tapasztalat, időjárás, ...)
- a pedagógiai kutatás nem valósítható meg, ha bármilyen módon sértene vagy kárt okozna a válaszadónak, vagy ha nem felel meg az erkölcsöknek.

Ezért a pedagógia számos jelensége csak a hipotézisek szintjén marad, feltételezhetők, de nincs lehetőség pontos ellenőrzésre és egzakt megerősítésre.

A hipotézisvizsgálatokra ezekben az esetekben jól használható a következtetési (inferenciális) statisztika, melyre jól megfelel az R szoftver.

3. R

Az R egy szabad, nyílt forráskódú, ingyenesen használható, professzionális statisztikai szoftvercsomag, amelyben már kidolgozott eljárásokat tartalmazó függvények és munkakörnyezetek állnak rendelkezésre. [1]

Az R nyelv a https://www.r-project.org/ honlapról letölthető és telepíthető. Használata elsajátítása alap programozási készségekkel rendelkező felhasználóknak nem okoz gondot. Segítségként ajánlhatjuk a https://www.statmethods.net/, https://www.r-bloggers.com/, weboldalakon található tutoriálokat. A tanuláshoz további segítségül szolgálhatnak a https://www.udemy.com/ weboldalon található videók.

Az R lehetőséget ad tetszőleges célú további számolási algoritmusok kifejlesztésére, programozáshoz és alkalmas a kidolgozott algoritmusok tesztelésre is. Tehát az R egy programozási nyelv, de egyúttal programkörnyezet is. [2]

Ebből ered az R egyik legnagyszerűbb tulajdonsága, az aktív közösség, amely a szabadon írható és közzétehető forráskódjának köszönhető. A számtalan algoritmus szabadon elérhető csomagokban a webről letölthető (https://cran.r-project.org/) és ismételten felhasználható különböző adathalmazokkal. Gyakran előfordul, hogy elegendő csak a szükséges referenciákat átállítani más adathalmazra, és már meg is kaptuk a keresett eredményeket. A csomagok témakörönként vannak a honlapon feltűntetve, amelyek közül nagyon sok használható a pedagógiai gyakorlatban is, például adatbázis kezelés, idősorok számítása, grafikonok stb.

Néhány ok, miért érdemes R-t használni:

- több adatot képes kezelni, mint az MS Excel, mivel az R nem dolgozik grafikus környezetben, így nem kell az összes adatott megjeleníteni a felhasználó számítógép képernyőjén, ezért több adattal képes számolni,
- platform-független, ezért a program telepíthető Windows, Linux és Mac OS operációs rendszert használó számítógépekre is, sőt ma már szerver-oldali megoldások is elérhetőek,
- gyors és stabil, a parancsok írása viszonylag egyszerű és azonnal végrehajtódnak, a grafikonok ábrázolása is nagyon gyors. A számítások végeredményeit általunk létrehozott változók-

ban menthetjük el, amiket bármikor megnézhetünk vagy használhatunk további számításokhoz.

- használható mátrixok és vektorok számolására, egyenletek megoldására stb.
- szabadon használható oktatásban, vállalati környezetben és otthon is,
- ingyenesen letölthető az RStudio is, amely megkönnyíti az R használatát, mivel tartalmaz egy kódszerkesztőt valamint hibakeresési és megjelenítési eszközöket. [3]

Az R alapjában egy interaktív statisztikai/adatelemző környezet, ahol a felhasználók utasításokat adnak ki az R konzolnak a parancsok végrehajtására és az eredmények is itt jelennek meg. Például (1. ábra) a „summary" parancs alap leíró statisztikákat számol a megadott adathalmazból (datacomp). Elég egyetlen parancs és egyszerre megjelenik több eredmény, ellentétben például az MS Excellel, ahol leggyakrabban minden értéket külön funkcióval kell kiszámítani.

```
> summary(datacomp)
     Female          Male
 Min.   : 3333   Min.   : 3508
 1st Qu.: 5153   1st Qu.: 4965
 Median : 9252   Median : 8953
 Mean   :12808   Mean   : 9856
 3rd Qu.:22525   3rd Qu.:14714
 Max.   :26383   Max.   :17489
```

1. ábra: Példa leíró statisztikai eredményre

Az alap statisztikák számolására az R-ben elég néhány egyszerű parancsot használni és ezeket a konzol segítségével megadni. De ha egy összetett kísérlet eredményeit szeretnénk statisztikailag kiértékelni már ez nem elegendő és R szkript-et kell használni. A szkript fájlok kiterjesztésükről (*.R*) ismerhetők fel. Több egymást követő parancsot tartalmaznak, melyeket egyszerre lehet indítani és a szoftver egymás után hajtja végre őket. (2. ábra)

A megírt kódokat (szkripteket) külön mappában is tárolhatjuk, az R szoftver csak a futtatásukhoz szükséges. Ugyanazt a szkriptet többször is használhatjuk, például az ismételt kísérletnél, az adathalmaz változásánál vagy akár teljesen más adatokkal.

Az R komplex adat vizualizáció lehetőséget is szolgáltat. A grafikus és ábrázolási képessége vitathatóan felülmúlja más programokét. Például a *dplyr* és a *ggplot2* ingyenesen letölthető csomagok nagy mennyiségű lehetőséget adnak az adatok manipulálásához és ábrázolásához, amelylyel könnyítik megoldani a vizualizációt az analízis elvégzése közben. A grafikonokat külön ablakban nyitja meg (3. ábra) és külön fájlokba is elmenthetők. [4]

Az R az adatokat és az elemzést külön mutatja be, ami a felhasználó számára lehetővé teszi az adatok pontosabb ellenőrzését, a felmerülő hibák javítását vagy az adatok megtekintését különböző elemzési pontokban. [5]

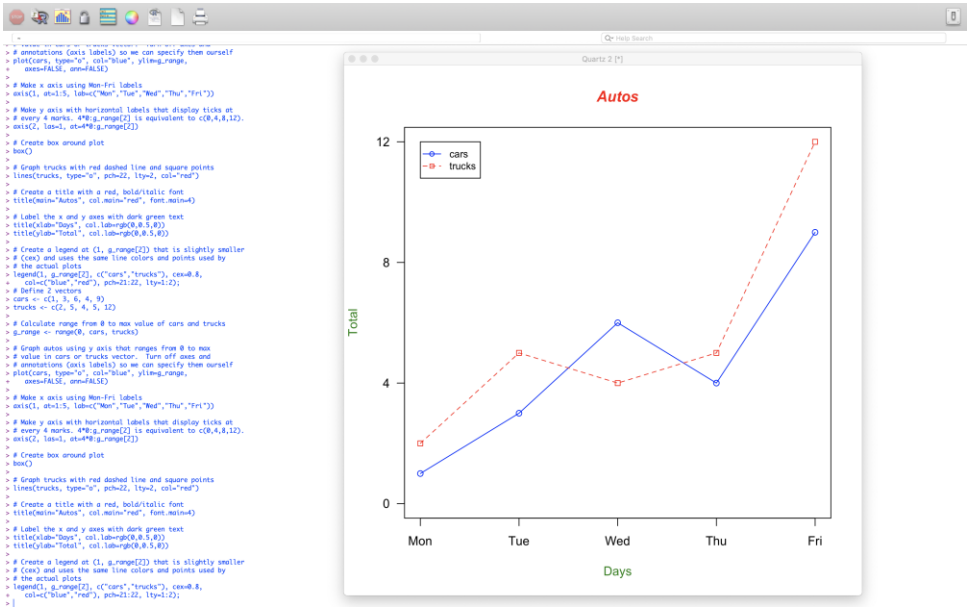
Persze, mint minden programnak, az R-nek is vannak negatívumai. Mivel az R alapja az 1960-as években használt programozási nyelvekből volt átvéve, egy viszonylag régi technológiának mondható. Jellemző rá, hogy az adatokat az operációs memóriában tárolja. Szerencsére a mai számítógépek memória kapacitása sokkal nagyobb, mint régebben, így ez a tulajdonsága már kevésbé (vagy egyáltalán nem) jelent problémát. [4]

```
# Importing Dataset
dataset = read.csv(file.choose(),sep=";")
# Changing to date
dataset$Rok <- as.Date(dataset$Rok, format="%d/%m/%Y")
# Renaming Columns
colnames(dataset) <- c('Year', 'Female', 'Male')
# Removing Date - or kkeping it
datacomp <- dataset[2:3]
datacomp <- ts(datacomp, start=c(1989), end=c(2017), frequency = 1)
autoplot(datacomp)
datafemale <- dataset[2]
datafemale <- ts(datafemale, start=c(1989), end=c(2017), frequency = 1)
autoplot(datafemale)
lines(lowess(time(datafemale), datafemale), lwd=3, col=2)
datamale <- dataset[3]
datamale <- ts(datamale, start=c(1989), end=c(2017), frequency = 1)
plot(datamale)
lines(lowess(time(datamale), datamale), lwd=3, col=2)

# Trend is statistically significant?
library(tseries)
maletrend = factor(ifelse( datamale >= median(datamale), 1, 0))
femaletrend = factor(ifelse( datafemale >= median(datafemale), 1, 0))
runs.test(maletrend, a = 'less')
# trend significant Standard Normal = -5.1056, p-value = 1.649e-07
runs.test(femaletrend, a = 'less')
# trend significant Standard Normal = -5.1056, p-value = 1.649e-07


# Finding Trend statistically changes
# install.packages('changepoint')
require(changepoint)
plot(datamale)
#detect change in trend at different points of time
cm=cpt.mean(datamale) #abrupt change in mean
print(cm)
plot(cm)
autoplot(cpt.meanvar(datamale))
```

2. ábra: Példa egy szkript részletre a benne elhelyezett megjegyzésekkel.



3. ábra: Példa egy grafikon megjelenésére.

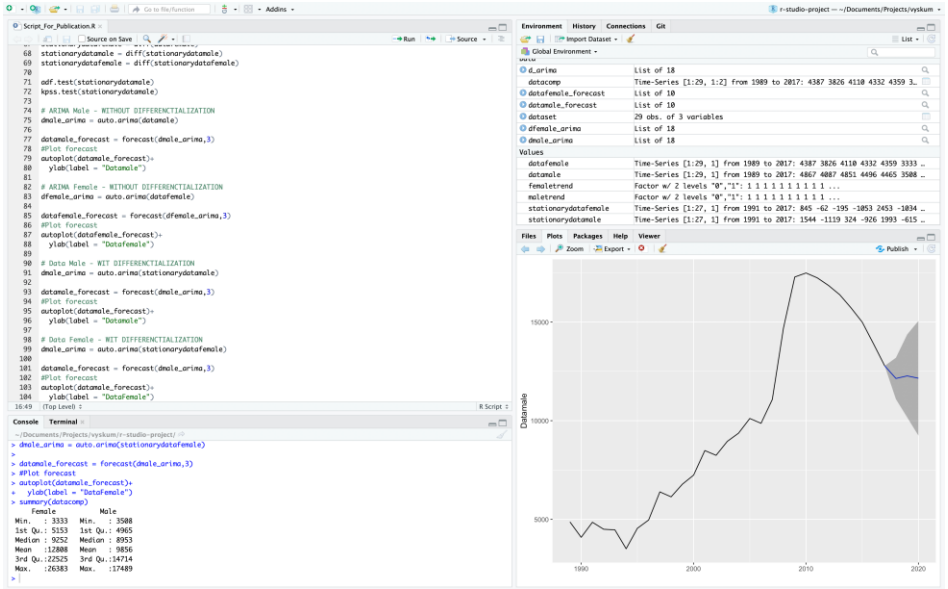
Az R nem biztonságos rendszer és nem lehet böngészőkbe vagy weboldalakba beágyazni. Az R-t nem lehet önmagában back-end szolgáltatásként sem használni. Ahhoz, hogy például az R számításokat végezzen el szükséges, hogy egy szerveren futó programozási nyelvből legyen meghívva. Ilyen nyelv lehet a PHP, Python, JavaScript stb.

Az említett szabadon elérhető R csomagjai között hibásak is találhatóak, melyek nem működnek tökéletesen, de ebben az esetben, sajnos, nincs kihez panasszal fordulni, hogy javítsa a hibákat.

Az R negatívumként említhető az a tulajdonsága is, ha egy szkriptnek csak pár sorát kell lefuttatni, akkor külön ablakot kell megnyitni és ezért az ablakok között gyakran kell váltogatni. Ezt a problémát küszöböli ki a már említett RStudio.

RStudio Integrált fejlesztői környezet, amely intuitíven használható. A képernyő felülete ablakrészekre van osztva, melyek mérete változtatható (4. ábra):

- baloldalon felül: szövegszerkesztő, fájlba elmentendő, szkriptek (kódok) számára,
- baloldalon alul: konzol a parancssoros módhoz,
- jobb oldalon felül: fejlesztői környezett (workspace), előzmények (history),
- jobb oldalon alul: segítség (help), csomagok (packages), grafikonok az ablakok.



4. ábra: RStudio felülete

4. Az R használata kutatásban

Az R lehetséges használatát egy kutatás eredményein keresztül mutatjuk be. A kutatás azt akarta felmérni milyen a munkakereső egyetemi végzetségének hatása a munkaadó választására.

A kutatáshoz szükséges adatokat a profesia.sk weboldal szolgáltatta, amelyen azok a személyek helyezik el életrajzukat, akik munkahelyet keresnek. Az életrajzok alapján a munkaadók válogatnak munkaerőt. A kutatáshoz 5 269 életrajz adatai voltak felhasználva. [6]

Szlovákiában jelenleg 35 felsőoktatási intézmény működik. A vizsgálat az informatika szakterületére koncentrált, vagyis olyan munkavállalók elhelyezkedésének eredményességét figyelte, akik a diplomájukat különböző egyetemeken, de valamelyik informatika ágazatában szerezték.

A kutatás statisztikai klasszifikáció segítségével igyekezet felmérni és bizonyítani vagy cáfolni többek között azt a hipotézist is, hogy a munkaadók előnyben részesítik azt a munkavállalót, aki egy jobb minősítéssel rendelkező egyetemen szerzett végzettséget.

Az adatok alapján a következő leíró statisztikákat lehet megjeleníteni (5. ábra). A táblázatok az R-Studio eredményei alapján készültek. A leíró statisztikák a már említett „summary" funkcióval készültek el. A táblázat első része az egyes egyetemeken végzett személyek a weboldalon közzétett életrajzainak számát tartalmazza, amit az R a „count" funkció segítségével készített el, mivel azok szövegként voltak reprezentálva az adathalmazban. Hogy ezeket hány munkaadó tekintette meg, vagyis a megtekintések értékelésénél automatikusan elvégezte a leíró statisztikákat. A táblázatban megjelenik a minimum, maximum, és az adatokból látható, hogy a maximális érték és a harmadik kvartilis nagyon messze vannak egymástól, ezért feltételezhető, hogy az adathalmaz extrém értékeket is tartalmazhat.

Egyetem	*Mennyiség*
Ekonomická univerzita - Ökonómiai Egyetem (EU)	439
Többi (11 egyetem)	895
Slovenská technická univerzita - Szlovák Műszaki Egyetem Pozsony	1864
Technická univerzita v Košiciach - Kassai Műszaki Egyetem (TUKE)	964
Univerzita Komenského - Comenius Egyetem Pozsony (UK)	396
Žilinská univerzita - Zsolnai Egyetem (ZU)	711

Megtekintések száma	
Min:	1.000
1. kvartilis:	2.000
Medián:	4.000
Átlag:	6.553
3. kvartilis:	9.000
Max:	64.000

5. ábra: Leíró statisztika eredmények

A lineáris regresszió számításánál az R automatikusan kiszámolja a maradványértékeket, koefficienseket és más adatokat, mint például az R^2 értékét, az F statisztikát, p értékét, stb. (6. ábra).

A különböző koefficiensek t- és p- értékei a táblázatban külön jelennek meg, így könnyedén kiszűrhetőek azok, amelyek statisztikailag szignifikánsak.

A modellben szignifikáns értéket érnek el a TUKE, ZU és a Többi (11) egyetem. Annak ellenére, hogy az R^2 értéke alacsony, ezek az egyetemek hatással vannak a megtekintések számára, mivel a referencia egyetemhez (a táblázatban a korrekciós tényezőbe van jelen) viszonyítva mindegyikük alacsonyabb mennyiségű megtekintést ért el.

A 6. ábrán feltüntetett statisztikai eredmények alapján, bizonyítható a felvetett hipotézis, hogy a munkavállaló egyetemi végzetsége hatással van a munkában való elhelyezkedésre, mivel a munkaadók előnyben részesítik azt a munkavállalót, aki egy jobb minősítéssel rendelkező egyetemen szerzett végzettséget. [6]

A lineáris regresszió eredményei

Maradványértékek:	Min	1. kvartilis	Medián	3. kvartilis	Max
	-6.86	-4.30	-2.05	2.07	58.07

Koefficiensek:	Becslés	Std. hiba	t-érték	p-érték
Korrekciós tényező*	7.304	0.151	48.25	$<2e^{-16}$***
EU	-0.427	0.347	-1.23	0.22
Többi	-1.984	0.266	-7.47	$9.6e^{-14}$***
TUKE	-1.371	0.259	-5.29	$1.3e^{-7}$***
UK	0.552	0.362	1.53	0.13
ZU	-1.253	0.288	-4.35	$1.4e^{-5}$***

Standard hiba	6.45, 4 209 szabadságfoknál
R^2	0.0161
Beállított R^2	0.0152
F statisztika	17.2, ami 5 és 4 209 szabadságfoknál
p érték	$<2e^{-16}$
Szignifikancia kódolása: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1	

6. ábra: A lineáris regresszió

5. Befejezés

Lehet, hogy az R felhasználói környezete kevésbé barátságos, mint az MS Excel-é, de minden tulajdonságát és funkcióit összegezve az R „statisztikai tudása" messze meghaladja az Excel funkcióit. Nagyon könnyen és jól használható program és használatát gyorsan el lehet sajátítani.

A pedagógiai gyakorlatban nagyon hasznosnak bizonyulhat, ha gyorsan szeretnénk sok adatot elemezni és statisztikailag kiértékelni.

Irodalom

1. *GNU R: szoftver, programozási nyelv, közösség* (2018) http://r-projekt.hu/mi-az-r/ (utoljára megtekintve: 2018.10.30.)
2. Tóthmérész Béla: *Bevezetés az R használatába* http://biodiversity.unideb.hu/files/oktatas/Tothmeresz_Bevezetes-az-R-hasznalataba.pdf (utoljára megtekintve: 2018.10.13.)
3. *RStudio* (2018) https://www.rstudio.com/ (utoljára megtekintve: 2018.10.3.)
4. Krill Paul: *Why R? The pros and cons of the R language* (2015), InfoWorld, https://www.infoworld.com/article/2940864/application-development/r-programming-language-statistical-data-analysis.html (utoljára megtekintve: 2018.10.13.)
5. Gebhart Kendra: *Understanding R programming over Excel for Data Analysis*, (2016) https://www.gapintelligence.com/blog/2016/understanding-r-programming-over-excel-for-data-analysis (utoljára megtekintve: 2018.10.3.)
6. Pšenák Peter, Káčer Ján: *A munkakereső egyetemi végzetségének hatása a munkaadó választására.* PEME 16. Budapest (Maďarsko): Professzorok az Európai Magyarországért Egyesülete, 2018. ISBN (online) 978-615-5709-03-6, s. 239-247

Interaktivitás az informatika tanításában

Pšenáková Ildikó[1], Heizlerné Bakonyi Viktória[2], Illés Zoltán[3]

{[1]ildiko.psenakova@gmail.com, [2]hbv@inf.elte.hu,[3]illes@inf.elte.hu}
[1]Trnavská univerzita v Trnave, [2,3]ELTE Budapest

Absztrakt. A mai modern iskola modern módszereket és tanítási formákat igényel. A modern iskolából nem hiányozhatnak a számítógépek és az információs és kommunikációs technológiák sem. Ismert tény, hogy az informatikával, számítástechnikával és programozással foglalkozó tantárgyak oktatásában a számítógépnek különleges a helyzete, mivel az nemcsak segédeszközként szerepel a tanórán, de maga a tantárgy tárgya is. A számítógépek lehetőséget teremtenek interaktív alkalmazások használatára is. Cikkünkben azt tárgyaljuk, hogyan lehet kihasználni az interaktivitást az informatika oktatásában.

Kulcsszavak: informatika, interaktivitás, oktatás,

1. Bevezetés

A digitális technológia elterjedése jelentősen megváltoztatta az emberek viszonyát a munkához, szórakozáshoz, ismeretszerzéshez. Ahhoz, hogy a fiatalok az iskolai tanulmányaikat befejezve be tudjanak illeszkedni a társadalomba és érvényesülhessenek is benne, erre a mindent átható változásra az oktatásnak is reagálnia kell. Meg kell újítani a hagyományos oktatással kapcsolatos elképzeléseket: új célkitűzésekkel, módszerekkel, tanítási gyakorlattal és eszközökkel. Modern idők modern iskolát, modern módszereket és tanítási formát igényelnek. Ez azonban nem jelentheti azt, hogy teljesen el kellene vetnünk a klasszikus iskolákban évszázadokon keresztül kiérlelődött jó gyakorlatot és mindent teljesen elölről kellene kezdeni. Használjunk fel minden kipróbált gondolatot, módszert és innovatív ötletekkel, tartalmi és pedagógiai elemekkel, jól megválasztott IKT eszközök használatával ezt a biztos alapot fejlesszük tovább! Gyakori tévedés, hogy egy-egy új eszköz megjelenése túlzott reményeket támaszt és szinte, mint csodaszerre tekintenek rá a kutatók. Sohase feledjük azonban, hogy a modern technológia használata önmagában nem jelent, nem jelenthet megoldást. A tanár személyisége, motiváló ereje, a tanítási módszerek helyes megválasztása kulcsfontosságú szerepet játszik az oktatásban.

Cikkünkben azokra az infokommunikációs eszközökre, lehetőségekre kívánunk fókuszálni, amelyek nélkül ma már el sem lehet képzelni egy iskolát. Az oktatásban az IKT eszközök igen széles skálája fordul elő a régimódi írásvetítőtől kezdve a számítógépen keresztül az interaktív tábláig vagy szavazó gépekig, hogy ne menjünk el a ma még szinte futurisztikusnak ható VR (Virtual Reality), AR (Augmented Reality) eszközök vagy akár az intelligens helyi szolgáltatások használatáig (https://bit.ly/2EVWkNw). Valamennyinek megvan a maga szerepe a tanítás során. Az írásvetítő remekül használható olyan egyedi vázlatok, rajzok bemutatására, amelyet a tanár előre elkészíthet és évről-évre felhasználhat. Animációs lehetőségei korlátozottak, interakció megvalósítására pedig nincs is lehetőség. Tanári számítógép és kivetítő esetén a lehetőségeink jelentősen megnőnek, hiszen szinte tetszőleges multimédiás anyag is beilleszthető az óra menetébe, de ebben az esetben is a tanár az, aki egy, a megszokottnál színesebb, de jószerével továbbra is „frontális tanítást” végez. Egy számítógépes teremben, ahol mindenki saját géppel rendelkezik, megfelelő szoftver alkalmazása (pl. NetSupport School Tutor) esetén megvan a közös és az egyéni munkára is a lehetőség. Természetesen egyéni számítógépek alkalmazásával interaktív, személyre szabott tanulás is megvalósítható. Ebben az esetben azonban egyre hangsú-

lyosabbá válik az ember-gép kapcsolat, a személyek közötti kapcsolat pedig ezzel egyidejűleg gyengül. Az interaktív táblák és a szavazógépek esetében viszont erősebben jelen van a diákok együttes munkájának élménye, az egészséges versenyszellem kialakításának és a folyamatos önértékelés lehetősége.

2. Informatika az oktatásban, informatika oktatása

Az előbbiekben áttekintettük, hogy milyen IKT eszközök jelentek/jelennek meg a közel jövőben az iskolákban, de nem szóltunk arról részletesebben, hogy milyen pozitív hatása lehet ezen eszközök használatának. Az interaktív eszközök megfelelő használata segítheti a diák-tanár kommunikációját, növelheti a tanulási motivációt, az oktatás szemléletességét, megkönnyítheti a tananyag megértését és érdekesebbé teheti a tanulási fázisokat. [1]

Az infokommunikációs eszközök használatának egyik vitathatatlan előnye az a vizualizációs képesség, amely megkönnyíti és felgyorsítja a tanulási folyamatot. A vizualizáció aktivizálja a tanuló egész személyiségét, felkelti a diák figyelemét és érdeklődését és közel hozza, életszerűbbé teszi a tanulást. Azt állítjuk, hogy az infokommunikációs eszközök használata már önmagában is motiváló erővel bír. Az ezekben rejlő lehetőségek, a motiváció ereje megkönnyíthetik a tovább lépést, sőt a független gondolkodás kialakítását is. [2]

Másrészről az IKT eszközök biztosíthatják a megfelelő hatékonyságú interaktivitást is.

- Interakciót a tanár és az osztály vonatkozásában például a különböző szavazórendszerek használatával. Ezek a rendszerek lehetővé teszik, hogy a tanár azonnal felmérje a gyerekek aktuális tudását, és szükség szerint változtasson az előzetes terveihez képest. Nem véletlen, hogy az egyik ilyen rendszer, a Kahoot (https://kahoot.it/) olyan népszerű már az Egyesült Államokban, hogy a diákok mintegy 50% használja is az iskolában. [3]
- Interakcióról beszélhetünk azonban az emberek és a számítógép vonatkozásában is, amennyiben az ember reakcióitól függően a gép különbözően reagál. A program az irányítójától (aki lehet tanár vagy diák is) mindig valamilyen cselekvést vár. Jó példa erre az interaktív tábla, amely például egy beprogramozott alap fizikai kísérletet, mondjuk, a lejtőn gyorsuló tárgy mozgását a gyerekek által megadott különböző kezdőértékek esetében a valóságban tapasztaltaknak megfelelően jeleníti meg. A közös megélt kísérletezés elvezetheti a tanulókat a körülöttük levő világ pontosabb megfigyeléséhez, jobb megértéséhez.
- Végezetül interakció épülhet ki egyetlen ember és a számítógép között is, amikor az egyén reakcióira válaszol a gép, amely újabb reakciót vált ki az emberből és így egy csak az egyénre jellemző tapasztalási, tanulási utat járhat be. Ez az, amit személyre szabott oktatási tartalomnak nevezhetünk.

Hála a technika fejlődésének ma a piacon a digitális eszközök igen széles skálája megtalálható a kicsi gyermekek számára készített tanuló mikrokontrollerektől a professzionális szerverekig. Ezeken az eszközökön különböző operációs rendszerekkel és változatosnál változatosabb szoftvercsomagokkal találkozhatunk. Nem lehet csodálkozni azon, ami sajnos, gyakran előfordulhat, hogy ezek közül némelyikkel a diákok állnak „szorosabb kapcsolatban" nem pedig a tanárok. Ne feledjük, a fiatalok digitális bennszülöttek, az őket oktatók jó része pedig csak digitális bevándorlónak számít. Sokat ronthat a tanítás hatékonyságán, ha a diákok azt érzékelik, hogy tanáraik nem használják magabiztosan azokat az eszközöket, amelyeket ők teljesen magától értetődőnek tartanak! [4]

Nyilvánvaló, hogy a mai iskola tanárai számára már nem elegendőek az alapfokú számítógépes ismeretek, és itt természetesen nemcsak az informatikát oktató tanárokra gondolunk. A tanároknak mélyebb IKT ismeretekre van szükségük, ezért alaposan meg kell ismerniük a különböző

digitális eszközöket és azok használatát, elsajátítani a megfelelő terminológiát, és az általuk használt különböző alkalmazásokat ebben nagy szerepe van a tanárképzésnek, továbbképzéseknek.

Az informatika és informatikával, számítástechnikával kapcsolatos tantárgyak oktatásának speciális a helyzete, mivel ebben az esetben a számítógép nem csak egy oktatási segédeszköz a sok közül, hanem jórészt maga a tananyag tárgya is. Másrészről miközben a világunkban szinte mindenben számítógép található, még a bojlerekben, hűtőgépekben, kazánokban is égető hiány alakult ki az informatikus munkaerőpiacon, így komoly gazdasági érdek is fűződik a hatékony oktatás kialakításához. Mind a közoktatás, mind pedig az ipari partnerek sokat tesznek azért, hogy a fiatalok számára kellően vonzó legyen az informatikus pálya. Ez az oka annak, hogy sokan, sokféleképpen közelítik meg ezt a kérdést.

- Egyfelől egyértelművé vált a számítógépes gondolkodás fontosságának kialakítása akár számítógép nélkül is, ilyen lehetőségeket lehet találni a https://csunplugged.org/en/ oldalon, de különböző versenyek is ezt a területet erősítik, példaként említhető a BEBRAS (https://www.bebras.org/).
- Másfelől az érdeklődés felkeltését szolgálja azoknak a foglalkozásoknak a szervezése, ahol azokat az új eszközöket is kipróbálhatják, amivel a hétköznapokban talán még nem is találkozhattak. Az így szerzett élmények is erősítik az informatika iránti érdeklődést. (http://tet.inf.elte.hu/tetkucko/)
- A következő szint a gyermekek bevonása, érdeklődésének felkeltése a programozás iránt, alapozva a korosztály érdeklődésére. A művészetek, a rajzolás, a mesék iránti szeretetet használják ki amikor Imagine, Scratch vagy akár Microbit programozást tanítanak a kisebbeknek. [5,6,7]
- A nagyobb diákok számára egyre több lehetőség nyílik olyan programozási nyelvek elsajátítására akár az iskolai órák keretében is, amelyek már az iparban is meghatározó sztenderdekké váltak. Ilyen például a C#, a (ma az oktatásban is olyan népszerű) Python, vagy a most repülő rajtot vett Swift nyelv, amelyek vonzereje legfőképpen a könnyű tanulhatóság, és a motivációt erősítő webes és mobilos platform jelenléte. [8,9,10]

3. Az interaktív tananyag

Ma a pedagógiai kutatások azt támasztják alá, hogy a személyre szabott oktatásnak, a saját ütemben haladásnak különlegesen fontos szerepe van a hatékony tanulásban. Nem véletlen, hogy az angolszász tutor rendszerű oktatás olyan elismert világszerte. A probléma ennek átvételével az, hogy ennek rendkívül magas az erőforrás igénye (igen sok segítőt, tutort, tanárt kellene foglalkoztatni), amely sok esetben nem megvalósítható. Az interaktív tananyagok használata azonban sokszor megoldhatja ennek az igénynek a kielégítését. Mindösszesen egy számítógép és egy megfelelő szoftver szükséges ahhoz, hogy a felhasználótól érkező jeleket tudja fogadni, értelmezni és annak megfelelően létrehozza a következő lépést. Az interaktív tananyag azonban csak akkor fogja biztosítani a funkcióját, ha átgondolt, jó terv alapján hozzák létre! Gyakran több szakemberből álló team együttes munkája alakítja ki az elkészült elektronikus tananyagot, amelyben pedagógustól kezdve, grafikusok és különböző területeken jártas informatikusok is részt vesznek, de az a helyzet sem példa nélkül való, hogy a pedagógusnak magának kell ilyen anyagokat előállítani. A folyamatos tananyagfejlesztésre szükség van, mivel idővel a tudományterületek változnak, új felfedezések, felismerések születnek, változnak a pedagógiai alapelvek, fejlődnek a taneszközök. **Ezért feltétlenül szükségesnek tartjuk, hogy a pedagógusképzésben a leendő tanítók és tanárok is elsajátítsák az interaktív tananyag készítésének alapjait és az interaktív tábla használatát!**

Az interaktív tananyagok az elektronikus tananyagok közé sorolhatóak, melyeknek nagy előnye, hogy több médiumot is magukba foglalhatnak (szöveget, képet, hangot, videót), így több oldalról is bemutathatják az adott problémát és annak megoldási lehetőségeit. Egy interaktív tananyag értelemszerűen különböző interakciós elemeket is tartalmazhat:

- Leggyakrabban az oktatás-tanulás irányítását maga a felhasználó (aki legtöbb esetben a diák) végezze visszalépve vagy továbblépve a következő anyagra.
- Előfordul, hogy a tanulási útvonalat a diák aktuális tudása is befolyásolja, mert az alkalmazás nem is engedi tovább, amíg a köztes kérdéseket nem tudja helyesen megválaszolni.
- Végezetül a tanulási útvonalat befolyásolhatja a tanuló tanulási módszere, ahogy például a CooSPace-ben írt anyagok esetében történik (http://portal.coospace.hu/). Nem mindegy, és egyénenként változik, hogy miből érdemes kiindulni, az általános elvekből a gyakorlati példák irányába, vagy fordítva!

A pedagógus szerepe megváltozik és a tanítás-tanulás folyamatát szerveznie, felügyelnie kell!

3.1. Tananyag készítési alapelvek

Az elektronikus tananyag helyes elkészítése nem egyszerű, de ismerve az alapvető követelményeket és betartva a tervezés jól meghatározott lépéseit, jó eredményeket lehet elérni.

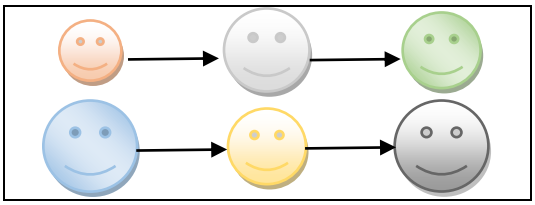
- Kezdjük a célkitűzés megfogalmazásával! Milyen tartalmat kívánunk átadni? Milyen korosztály számára készülne a tananyag? Milyen kompetenciákat kívánunk vele fejleszteni?
- Ha megvan a tananyag tartalma, akkor azt célszerű kisebb didaktikai részekre osztani figyelembe véve a tananyag logikai összefüggéseit és a tanulási folyamat optimalizálását, amely magába foglalja a korosztállyal kapcsolatos pedagógiai és pszichológiai jellegzetességeket. A tananyagszerkezet didaktikai alapegysége a *lecke*, amely néhány oldal terjedelmű. Tartalmazza az elsajátítandó tartalmat, példákat, kvízeket, játékokat, melyet a motiváció és figyelemfenntartására szolgálnak. A lecke végére összefoglaló kérdések vagy tesztek is beilleszthetők. A leckéken belül célszerű megkülönböztetni a törzsanyagot és a kiegészítő anyagot, például a kiegészítő anyag képezhet mögöttes oldalakat, melyekhez hivatkozások vezetnek és megjelenhet külön ablakban stb. [11,12]

1. ábra: Leckék

A *modul* leckékből tevődik össze. A modulok szerepe a leckéknek a tananyag tartalmi logikája szerinti rendszerbe foglalása. Leggyakoribb a lineáris szerkezet, amely a hagyományos tananyagok esetében szokásos felépítést követi. Egy modul általában egy jól körülhatárolt témakört, területet dolgoz fel. [11,12]

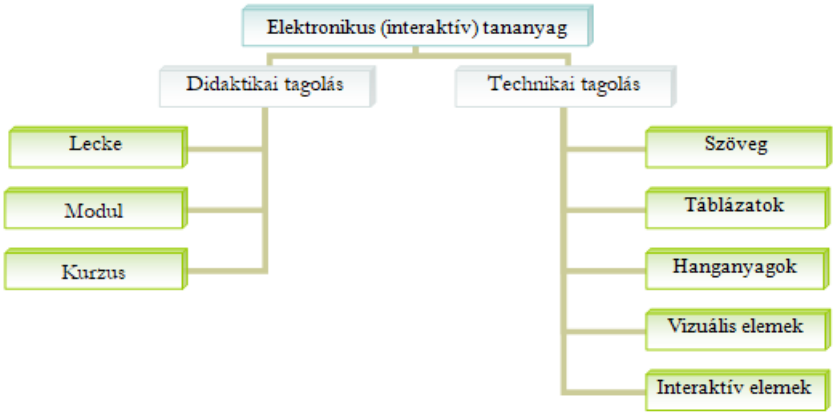
2. ábra: Modulok

A *kurzus* az elektronikus tananyag, önállóan tanulható formátumú verziója, általában modulokból áll. A kurzus terjedelme és szerkezete a tananyag jellegétől függően változhat. [11,12]



3. ábra: Kurzus (két modult tartalmaz)

- Az elektronikus tananyagok általában többféle médiumot foglalnak magukba, ezeket célszerű adattípusok szerint meghatározni. A technikai tagolás a tananyagot felépítő elemek, adatok különböző típusait jelenti. Szövegként megjelenhet az alapszöveg, ami tartalmazza a megtanulandó anyagot, definíciókat, példákat, feladatokat, ellenőrző kérdéseket, megoldásokat, de szöveg a tartalomjegyzék, összefoglaló, lábjegyzet és a megjegyzés, stb. is. A táblázatok is tartalmazhatnak szöveget, de számértékeket is. A táblázat sokszor segít az adatok rendszerezésében, átláthatóbban jelennek meg és így megkönnyíti az olvasónak a megértésüket. A hanganyagok valójában akusztikai elemek, amelyeket formailag megkülönböztethetünk, például lehet az beszéd, ének, zene. A vizuális elemek közé sorolhatjuk a képeket, animációkat, videókat és prezentációkat. Vázoljuk fel a megtervezett média elemeket, hogy a későbbiekben könnyebben átlássuk a tananyagot és megkönnyítsük különböző kombinációk létrehozását (4. ábra) az irányításra szolgáló interaktív elemekkel. Ezek leggyakrabban, mint nyomógomb, nyíl, link vagy más formában jelennek meg az eszközök (számítógép, mobiltelefon) képernyőjén vagy az interaktív táblán. Aktiválásukkal elindulhat egy másik animáció, más úton haladhat tovább a magyarázat, visszacsatolást kaphatunk, vagy vissza nézhetünk újra egy régebben látott tananyag részt, újra indíthatjuk a magyarázatot, stb. Ne hagyjuk figyelmen kívül az ergonómiai kérdéseket sem, hiszen a kényelmes használat nagyban befolyásolhatja az eredményességet!



4. ábra: Az elektronikus tananyag tagolása

- Végezetül a színpadképekből hozzuk létre az informatikában felhasználói diagramnak nevezett kapcsolati leírást.

Az elektronikus tananyag felépítésének ismerete sokban segítheti a tervező munkáját az interaktív tananyag elkészítésében. Mindenképpen javasoljuk, hogy mielőtt belekezd az olvasó

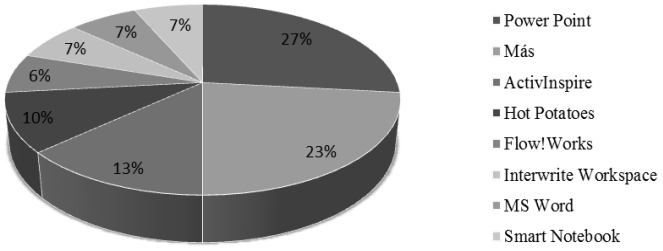
egy nagyobb lélegzetű munkába, olvasson utána részletesebben is az előbbiekben felvázoltakra. [13]

3.2. Tananyagkészítő szoftverek

Interaktív oktatási tananyagok, alkalmazások és tudásszintmérő tesztek fejlesztése nem könnyű feladat annak ellenére, hogy a készítés támogatására különböző szoftvercsomagok léteznek. Sajnos, több közülük nem ingyenes. A tanároknak, de gyakran még az iskoláknak sincs elegendő pénzük arra, hogy megvegyék ezeket. Vannak azonban szabadon letölthető programok is, vagy a piacon levő fizetős programok ingyenes változatai, de ezek legtöbbször csak korlátozott parancskészlettel rendelkeznek, így nem lehet kihasználni minden funkciót. Sok tanár éppen ezért az interaktív táblát sem tudja megfelelően kihasználni, mert elérhető szoftverek híján nem tud létrehozni jó minőségű, didaktikai és módszertani szempontokból is megfelelő tananyagot.

Nemrégiben egy projekt keretén belül egyszerű kérdőíves felmérést folytattunk, amelynek az volt a célja, hogy feltérképezzük az interaktív tábla használatát a szlovákiai oktatási intézményekben. A kérdőívet elektronikus formában terjesztettük a tanárok körében különböző iskolákban. A kérdőív kitöltése önkéntes volt, ezért a válaszadók nem alkotnak reprezentatív mintát, de a beérkezett válaszokból ennek ellenére számos érdekes megállapítást vonhatunk le. A kérdőív kérdéseiből itt csak egyet idéznénk.

Arra a kérdésre, hogy „Milyen szoftvert használ a saját interaktív tananyag elkészítéséhez?”, több mint 15 fajta válasz érkezett. A következő grafikonban csak a leggyakrabban előfordultakat tüntettük fel és a többit egybevéve a „más” jelzővel láttuk el. A magyarázatban csökkenő sorrendben vannak feltüntetve az egyes alkalmazások. (5. ábra)



5. ábra: Válaszok a „Milyen szoftvert használ a saját interaktív tananyag elkészítéséhez?” kérdésre.

Az eredményből is látható, hogy interaktív tananyag készítéséhez sokfajta szoftver használható. De jogosan felmerül a kérdés, vajon lehet-e minden fentebb felsorolt szoftverrel a követelményeknek megfelelő interaktív tananyagot/alkalmazást készíteni? Ennek megválaszolása további vizsgálatokat kívánna.

4. Értékelés, interaktív tesztelés

Az informatika órákon (de nemcsak azokon) egyre nagyobb szerepet kap az értékelés, sőt az IKT eszközök segítségével történő értékelés. Ebben nagy segítségre vannak a szavazó rendszerek is, de sajnos, nem mindenütt és nem mindig állnak rendelkezésre. Igaz, a tanulók saját eszközeit bevonva használhatunk néhány szabad felhasználású BYOD rendszert, például a korábban említett Kahoot-ot. Igaz, a nem megfelelő felhasználói azonosítás szummatív értékelés használatát nem igazán támogatja.

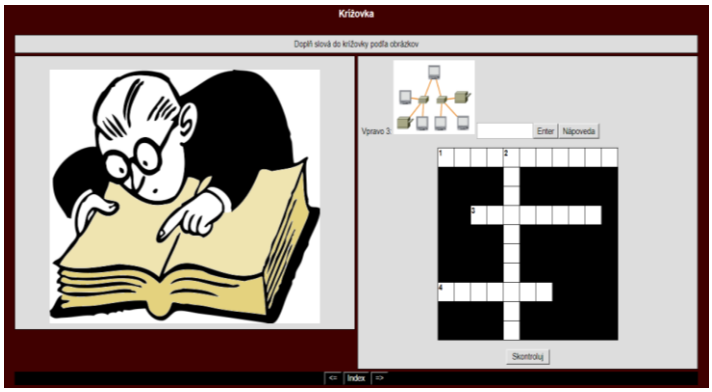
Szerencsére azonban számítógépen vagy akár interaktív táblán is lehet tesztelni a diákok tudását és különböző típusú kérdésekkel, interaktívvá és élvezetesebbé tenni a tanórát. Az interaktív gyakorló tesztek segítenek az ismeretek elmélyítésében, míg az egyes tananyag egységek lezárásánál, felhasználva az iskolai gépek beléptető rendszerének authentikálási módszereit, ezek feleltetős változatát is használhatjuk. A tesztek automatikus kiértékelése megkönnyíti a tanári munkát, bár egyúttal jól tudjuk azt is, hogy a tesztek nem mindig tükrözik a diákok tényleges tudását, hiszen a nem jól megfogalmazott kérdések szinte rávezethetik a tanulót arra, mit kell beikszelni.

A teszteket azonban létre is kell hozni, ami munkaigényes folyamat. Szerencsére ennek segítésére már léteznek szoftverek, melyekben a tesztkérdések egyszerűen elkészíthetők, csoportosíthatók, lejátszhatók, kiértékelhetők. Az ilyen típusú szoftverek igen nagy segítséget jelentenek az interaktív kérdések létrehozásában.

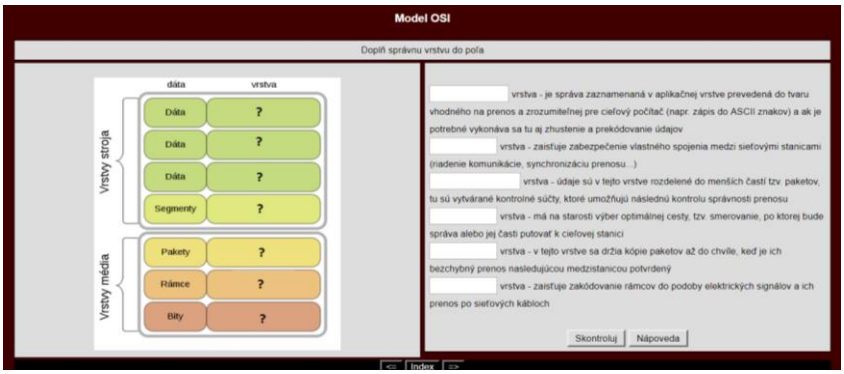
Egyik ilyen program a HotPotatoes, amelynek létezik ingyenes verziója is és szabadon letölthető (https://hotpot.uvic.ca/). Többfajta tesztet lehet benne elkészíteni különböző típusú kérdésekkel, például:

- többszörös választás egy helyes válasszal,
- több válasz kiválasztása,
- párosítás
- kifejezések helyes sorrendbe rendezése
- képrendezés
- keresztrejtvény (6. ábra)
- hiányzó szavak pótlása. (7. ábra)

A program számolja a helyes válaszokat és százalékban adja meg az eredményt. Be lehet állítani a tesztelésre szánt időt, a megjelenő színeket, képeket, hangot, animációt használni a kérdésekben, de a válaszokban, ill. az értékelésben is.



6. ábra: Keresztrejtvény (diák munka - Košalko)



7. ábra: Számítógép hálózatokkal foglalkozó teszt (diák munka - Košalko)

5. Interaktív szótár

A számítástechnika és informatika terminológiája nagyon sok angol, ill. angol eredetű szavat tartalmaz, ami gyakran megnehezíti a tananyag megértését. Elsősorban gondolunk itt az alsó tagozatos tanulókra, akik, ha tanulnak is angol nyelvet, még nem ismerik az angol szakkifejezéseket, de az idősebb generációnak is gondot okozhatnak a szakszavak. Ilyenkor nagyon jól ki lehet használni az elektronikus interaktív szótárakat.

A szótár lehet egyszerű, amely megjeleníti például a keresett angol szó magyar/szlovák megfelelőét, de lehet akár értelmező szótár is, amely magyarázatot is ad a keresett szóhoz. Az ilyen típusú szótárak tartalmazhatnak képeket, animációkat és természetesen interaktív elemeket is. (8. ábra)

8. ábra: Példa egy számítógép hálózatokkal foglalkozó értelmező szótárból (Záró dolgozat - Košalko)

6. Befejezés

Mint látható az informatika szerepe mind a hétköznapokban, mind pedig az oktatásban is egyre erősödik. Ma már nemcsak az informatika tanárnak, hanem minden pedagógusnak értenie kell az infokommunikációs eszközök használatához, hiszen olyan plusz lehetőségeket rejt magába, amely nagyban emeli az oktatás hatékonyságát. A cikkünkben kiemelt helyen foglalkoztunk az

IKT eszközök órai interaktivitást lehetővé tevő formáival, amely elősegíti a diákok bevonását a tanulási folyamat egészébe. Miközben az informatika órákon a számítógépek természetszerűleg adottak az interakció megvalósításához, láthatjuk, hogy erről máskor sem kell lemondanunk. Egyre több osztályban található interaktív tábla és az okos telefonok elterjedésének köszönhetően a gyermekek saját eszközeinek felhasználása is kiváló lehetőségeket teremt ezen a téren is. Ne féljünk használni ezeket az eszközöket és az eredmény nem fog elmaradni!

Köszönetnyilvánítás

A tanulmány megjelenését a KEGA 015TTU-4/2018: „Interaktivita v elektronických didaktických aplikáciách." (Interaktivitás az elektronikus didaktikai alkalmazásokban) című projekt támogatta.

Irodalom

1. Pšenáková, I. (2016): *Interactive applications in the work of teachers*. In: *XXIXth DIDMATTECH 2016*. Budapest: Eötvös Loránd University in Budapest - Faculty of Informatics. sz. 92-100. ISBN 978-963-284-800-6.
2. Námesztovszki Zsolt, Glušac Dragana, Branka Arsović: *A tanulók motiváltsági szintje egy hagyományos és egy IKT eszközökkel gazdagított oktatási környezetben*. (2013) In: Oktatás – Informatika, ELTE, ISSN 2061-1870, http://www.oktatas-informatika.hu/2013/03/namesztovszki-zsolt-glusac-dragana-branka-arsovic-a-tanulok-motivaltsagi-szintje-egy-hagyomanyos-es-egy-ikt-eszkozokkel-gazdagitott-oktatasi-kornyezetben/ (utoljára megtekintve: 2018.10.20.)
3. H. Bakonyi Viktória, Illés Zoltán: *Interactive talks*, EDUKACJA TECHNIKA INFORMATYKA / EDUCATION TECHNOLOGY COMPUTER SCIENCE 11:(1) pp. 298-303. (2015)
4. Suplicz Sándor (2012): *Tanárok pszichológiai jellemzői diákszemmel*. Doktori (PhD) értekezés. https://dea.lib.unideb.hu/dea/bitstream/handle/2437/161946/Suplicz_Sandor_Ertekezes-t.pdf?sequence=5 (utoljára megtekintve: 2018.10.12.)
5. Abonyi-Tóth Andor, Pluhár Zsuzsa, Dr. Turcsányi-Szabó Márta: *Játékos kódolás micro:bit-ekkel* az NJSZT és az ELTE T@T labor támogatásával, https://bit.ly/2yKZqhH (utoljára megtekintve: 2018.10.10.)
6. Rozgonyi-Borus Ferenc: *Képzeld el!,* algoritmusok, játékok. Szeged, 2008, ISBN: 978-963-06-2496-1
7. Bernát Péter: *Scratch portál*, http://scratch.elte.hu/ (utoljára megtekintve: 2018.10.20.)
8. Illés Zoltán: *Programozás C# nyelven*. Jedlik kiadó 2008, ISBN: 9789638762948
9. Peter Wentworth, Jeffrey Elkner, Allen B. Downey és Chris Meyers: *Hogyan gondolkozz úgy, mint egy* informatikus: tanulás Python3 segítségével https://bit.ly/2ql0ep0 (magyar verzió), (utoljára megtekintve: 2018.10.20.)
10. *Apple: Everyone can code*, https://www.apple.com/everyone-can-code/ (utoljára megtekintve: 2018.10.20.)
11. Mišút, M.: *IKT vo vzdelávaní*, Pedagogická fakulta Trnavskej univerzity v Trnave, (2013), ISBN: 978-80-8082-695-6
12. Antal, P., Forgó, S.: *A pedagógus mesterség IKT alapjai.* HUNline, http://okt.ektf.hu/data/forgos/file/tananyag/forgo/index.html (utoljára megtekintve: 2018.10.20.)
13. Antal Péter: *Interaktív elektronikus tananyagok tervezése*, http://mek.oszk.hu/14100/14163/pdf/14163.pdf (utoljára megtekintve: 2018.10.20.)

Informatika tehetséggondozás a Debreceni Fazekas Mihály Gimnáziumban

Simon Gyula, Kiszely Ildikó

sgy@fmg.hu, ki@fmg.hu
Debreceni Fazekas Mihály Gimnázium

Absztrakt. Fő cél, hogy minél több diák szeresse meg az algoritmizálást, programozást. A 2007-ben kidolgozott módszer 3 alapelvre épül: a korai felismerésre, hogy egy új ismeretet csak akkor kell megtanítani, ha feltétlenül szükségessé válik és az egyszerűség. A programozás kapcsolódik más területekhez is, többek között a fizikához (myDAQ, LabVIEW), az elektronikához (Arduino, C++) és a robotikához (LEGO robot, NXC) is, így ezek is motivációs forrást jelenthetnek a diákok számára. A Nemes Tihamér Programozói Versenyen kívül más versenyeken is megmérethetik magukat a tanulók, mint például a Pendroid mobil programozó versenyen, Neumann Programtermék Nemzetközi Versenyen; myDAQ pályázatokon.

Kulcsszavak: tehetséggondozás, programozás, új módszerek, robotika, elektronika, fizika

1. Bevezetés

Iskolánknak nagy hagyományai vannak a programozás tanításának a területén, még a 80-as években kezdődött, először szakköri formában, majd a 90-es évek elejétől a speciális matematika tagozatos osztályokban, heti két órában informatikát tanultak, jelentős részben programozást. Mint más iskolákban is, az első időkben házi számítógépek voltak, amelyeket a játékokon kívül lényegében csak programozásra lehetett használni. Commodore-64, ZX Spectrum, Primo, HT-1080Z, VTC voltak talán a legelterjedtebbek. Nagyon sok tanár abban az időben a fizika felől közelített a számítástechnika felé (talán mert fizika szakos is volt…), így könyvek, cikkek jelentek meg a házi számítógépek fizikai alkalmazására, például a felezési idő mérése Geiger-Müller számlálóval, amelyet a HT számítógépre írtak. (Ugyanezt a műszert és radioaktív forrást felhasználva két diákunk 30 év múlva első díjat nyert az NI Hungary Kft. pályázatán.)

1988-ban korrigált tantervek jelentek meg, amelyben a gimnáziumok I. és II. o tananyagába számítástechnikai ismeretek kerültek be, a technika óraszáma 1/3-nak megfelelő óraszámban. A 90-es években a technika tárgy gyakorlatilag kizárólag számítástechnikai ismereteket tartalmazott nagyon sok iskolában – hallgatólagosan.

A NAT 1998-as bevezetésével megjelent a közismereti informatika tantárgy, amelyben már könnyebben elhelyezhető volt a programozás tanítása.

A 2007-ig terjedő időszakban is sok tehetséges diákkal dolgozhattunk együtt, akik közül aztán sokan országos sikereket értek el.

2. Új koncepció

2007-ben, az addigi módszereket és eredményeket elemezve jutottunk arra az elhatározásra, hogy megújítjuk a programozás tanításának az *általunk alkalmazott* módszertanát. Fő célként azt jelöltük meg, hogy *minél több diákkal szerettessük meg a programozást* és lehetőleg minél több tanuló jusson el valamelyik országos programozó verseny döntőjébe (Nemes Tihamér OITV, OKTV), a helyekre

nem voltak előzetes elvárásaink, úgy gondoltuk, hogy a nagyszámú döntőbeli részt vétel magával hozza a kiemelkedő eredményeket is.

Az új módszer rendkívül sikeresnek bizonyult, a programozó versenyeken diákjaink nagy számban jutottak döntőbe, több esetben nyertek is. 2013 és 2017 között a Nemes Tihamér OITV és az OKTV programozás kategóriájában iskolánk tanulói vettek *a legnagyobb számban részt az országos döntőben.*

2018-ban például mindhárom nemzetközi diákolimpián volt tanítványunk (IOI, CEOI, EJOI).

Az akkor kidolgozott elképzeléseinkkel az iskolánk lehetőségeihez próbáltunk alkalmazkodni, elsősorban a speciális matematika tagozatos osztályok (4 évfolyamos és 6 évfolyamos), illetve a reál nyelvi előkészítős osztályok tanulóira támaszkodtunk. *Ezek a módszerek az iskolánkra és ránk szabottak, ebben a kombinációban lett sikeres az új koncepció.*

2.1. Eredmények

Nemes Tihamér OITV-n országos első 10-be került és OKTV programozás kategóriában döntős helyezést elérő versenyzők

1. korcsoport	2. korcsoport	OKTV
2008-2009-os tanév:		
2. hely: VJ 8. o		5. hely: VL 12. o
7. hely: GyM 8. o		7. hely: ÉA 11. o
2009-2010-os tanév:		
1. hely: SzB 8. o	6. hely: VJ 9. o	12. hely: ÉA 12. o
3. hely: NV 8. o		
2010-2011-os tanév:		
4. hely: AP 8. o	7. hely: NV 9. o	
5. hely: HP 8. o	8. hely: NV 10. o	
5. hely: SzG 8. o		
2011-2012-os tanév:		
2. hely: BZS 7. o	5. hely: VZ 10. o	40. hely: NV 11. o
4. hely: AN 7. o	7. hely: NV 10. o	
5. hely: NM 7. o		
6. hely: PSZ 7. o		
7. hely: PS 8. o		
2012-2013-os tanév:		
1. hely: AN 8. o	6. hely: VZ 11. o	5. hely: NV 11. o
2. hely: BZS 8. o	8. hely: AP 10. o	30. hely: UM 11. o
3. hely: PSZ 8. o		35. hely: NV 12. o
		38. hely: VJ 12. o

2013-2014-es tanév:		
3. hely: JF 8. o	10. hely: AN 9. o	1. hely: NV 12. o
		4. hely: VZ 12. o
		7. hely: AP 11. o
		17. hely: HI 11. o
2014-2015-os tanév:		
1. hely: DB 7. o	1. hely: AN 10. o	6. hely: AP 12. o
3. hely: SZM 8. o	1. hely: BZs 10. o	8. hely: VZ 13. o
9. hely: CSÁ 8. o		14. hely: SzG 12. o
9. hely: BG 8. o		20. hely: HI 12. o
		37. hely: KM 11. o
2015-2016-os tanév:		
1. hely: NÁ 8. o		6. hely: SzJ 11. o
		10. hely: AN 11. o
		14. hely: KM 12. o
		33. hely: VD 12. o
		39. hely: VB 11. o
2016-2017-os tanév:		
1. hely: GL 7. o	1. hely: NÁ 9. o	8. hely: BZs 12. o
2. hely: VB 8. o		12. hely: VB 12. o
4. hely: VBe 8. o		13. hely: SzJ 12. o
5. hely: K.SZM 8. o		19. hely: AN 12. o
9. hely: FR 7. o		42. hely: NM 12. o
10. hely: NL 7. o		
2017-2018-os tanév:		
5. hely: AL 7. o		5. hely: NÁ 10. o
5. hely: GL 8. o		33. hely: ML 12. o
9. hely: FR 8. o		

1.táblázat: versenyeredmények[23] [24]**.**

2.2. A megvalósítás

A hat évfolyamos osztályban a tanulók 7. és 10. osztály között heti 2 órában tanulnak informatikát. Az első két évben elsősorban programozást tanulnak a gyerekek, szeptembertől márciusig. Hetedik osztályban a programozás alapjaival ismerkednek meg a tanulók, sok-sok példán keresztül, nagyon

[23] http://nemes.inf.elte.hu/nemes_archivum.html

[24] https://www.oktatas.hu/kozneveles/tanulmanyi_versenyek/oktv_kereteben/dijazottak_eredmenyek

kevés elmélettel, mintegy felfedezve a programozást. Az első években magunk is tartottunk attól, hogy ez nem lehet hatékony egy olyan nyelvvel, mint a C++, de bebizonyosodott, hogy lehetséges. A hatékonyságra jellemző, hogy ebben az időszakban (2007 óta, az első eredmények 2009-ben jelentkeztek) két alkalommal fordult elő, hogy iskolánk hetedikes tanulója lett első (legutóbb 2017-ben), további 3 esetben nyolcadikos tanulónk szerzett első helyt, összesen ebben az időszakban 30 tanuló ért el az első tízbe eső eredményt.

A tanórai foglakozások mellett az érdeklődő, motiváltabb diákok szakkörre is járhatnak, szintén heti két órában (amelyet a kötelező óraszámon kívül, „ingyen" tartunk…).

Nyolcadik osztályban az első három hónapban (a már jelentősen előbbre járókon kívül, ők külön feladatokat kapnak), újra az alapokat vesszük át, mintegy lehetőséget adva azoknak, akik hetedikben lassabban haladtak, nem éreztek rá az algoritmikus gondolkodásmódra, vagy egyszerűen most kezdenek érdeklődni a programozás iránt. Mi sem mutatja jobban ennek létjogosultságát, hogy volt olyan tanulónk, akik nyolcadikban kezdett érdeklődni a programozás iránt, abban az évben már országos 3. lett (később OKTV győztes is, az IOI-n pedig bronzérmes), de rajta kívül másoknak is segített a „második merítés".

A továbbiakban (az egyéni tanuláson kívül) már csak szakkörön van lehetősége a diákoknak, hogy programozást tanuljanak, a tanórákon az érettségi követelményeinek megfelelő ismeretekkel foglalkoznak. Van egy „harmadik kör" is a programozással való ismerkedésre, 11-12. osztályban az informatika fakultáció.

2.3. A három alapelv

Az új „módszertanunk" 3 alapelvre épül:

- *A tehetséges gyerekek korai felismerése.* Iskolánkban 6 évfolyamos matematika tehetséggondozó osztályok vannak, akiknél már hetedik osztályban elkezdjük a programozás tanítását heti két tanórában és heti 2 óra szakköri órában. Tapasztalataink szerint ez a korai kezdés egyáltalán nem túlzó, tehetséges és motivált tanulóink 8. osztály végére gond nélkül meg tudják oldani az emelt szintű informatika érettségi programozási feladatát. Sok esetben a fiatalabb diákok nyitottabbak az új befogadására, könnyebben sajátítják el az új gondolkodásmód alapjait.
- A tananyag felépítésében azt az elvet követjük, hogy *egy új ismeretet csak akkor kell megtanítani, ha feltétlenül szükségessé válik.* Például nem tanítjuk az egymást követő órákban mindhárom ciklus utasítást, hanem először csak a for() ciklust és a másik kettőt csak akkor, amikor egy probléma megoldásához feltétlenül szükséges, vagy célszerűbb eszköz, például a rendezés esetén elég csak egy algoritmust megmutatni eleinte. Felesleges az úgynevezett foglalt szavakat tanítani, ahogy a legtöbb programozási könyv teszi a legelején, vagy az összes operátort. Tehát a nyelv referenciája helyett egy *induktív jellegű megismerést* sajátíthatnak el a tanulók, apró lépésekben haladva.
- *Egyszerűség.* Csak azt tanítjuk meg, amire az adott szinten feltétlenül szükség van. Az érdeklődő, motivált tanuló egy idő után képes lesz arra, hogy ezen túlmutató ismereteket szerezzen, például Open GL-t a grafikai megjelenítéshez. Ez az elv abból indul ki, hogy a versenyek, illetve az emelt szintű érettségi feladatai egészen szűk nyelvi eszköztárat igényelnek. Egy hetedikes tanulónak teljesen felesleges objektum orientált eszközöket tanítani, a feladatok mérete sem indokolja, de az emelt szintű érettségire is felesleges. Algoritmusok szintjén is hasonló a helyzet, nem kell mindent félév alatt megtanulni. Időt kell hagyni arra, hogy az algoritmikus gondolkodást elsajátíthassa a tanuló.

3. Programozás alkalmazásai

Motivációs célból vetődött fel iskolánkban, hogy olyan területeket keressünk, ahol a programozás kulcsszerepet játszik, de valamilyen kézzelfogható eredményt, visszajelzést kap a programozó, a való élettel közvetlen kapcsolatban van.

3.1. LEGO robot programozása

A legkézenfekvőbb választás, minden utasításnak azonnal látható az eredménye, kevésbé absztrakt maga feladat is, és annak megoldása is. Ráadásul támaszkodhatunk a gyerekekben (és a bennünk) megbújó kisgyerekre, szeretnek „legózni", amellyel kiélhetik kreativitásukat. Talán ez a hátránya is a dolognak, az idősebb diákok kevésbé értékelik azt, hogy legálisan játszhatnak az iskolában, és talán már fejlettebb is a gondolkodásuk annyira, hogy nem igénylik a kézzelfogható visszacsatolásokat.

A választásban az is szerepet játszott, hogy a LEGO robot akkori verziójához elérhető volt több C nyelvű fordító is, köztük ingyenes is (NXC). Ezzel a választással a C nyelv alapjait, adattípusait, vezérlési szerkezeteit gyakorolhatják a tanulók, ami erősíti a tanórákon tanultakat.

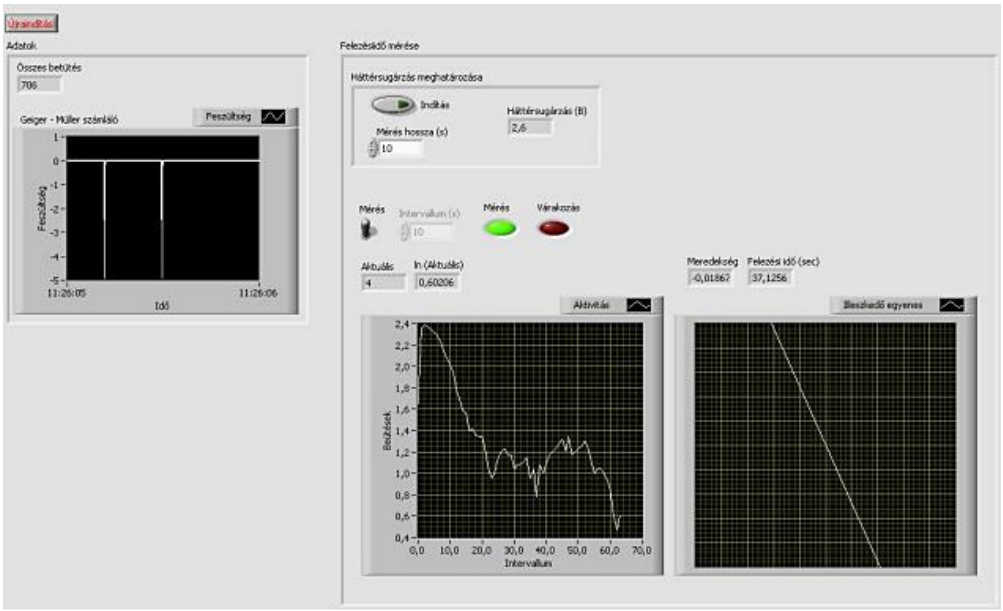
3.2. myDAQ

A LEGO robot versenyeken keresztül kerültünk kapcsolatba az NI Hungary Kft. mentor programjával. Mi is részt vettünk a tanárok számára szervezett *LabVIEW* továbbképzésen, ahol megismerhettük a nyelv alapjait és lehetőségeiről is képet kaptunk.

Sikerült érdeklődő diákokat „felkutatni", akik lelkesen vetették bele magukat a *myDAQ mérési adatgyűjtő* programozásába. Az NI által kiírt LabVIEW országos programozó versenyeken diákjaink négy alkalommal szerezték meg az első helyezést az évek során.

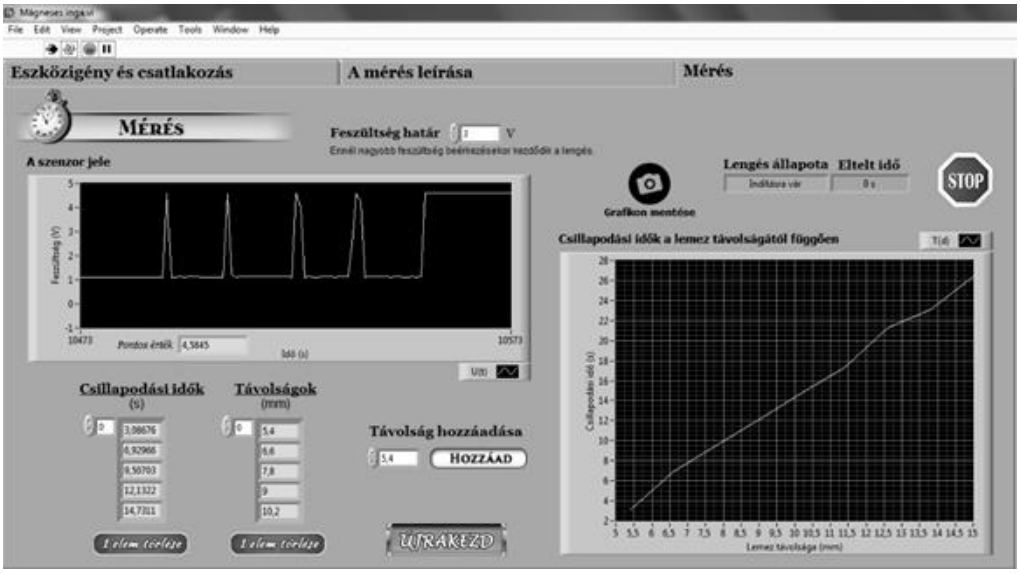
Az egyik csapatunk, a már említett NI pályázaton lett első a felezési idő mérésével.

Geiger-Müller számláló



A mérés programja

Néhány év múlva egy másik csapat Simon Gyulával az Eötvös Lóránd Fizikai Társulat és az NI Hungary Kft. által *fizikatanároknak* kiírt pályázaton szerezte meg az első helyezést. *http://fizikaiszemle.hu/archivum/fsz1610/FizSzem-201610.pdf*



A mágneses inga csillapodása

3.3. Arduino mikrovezérlő programozása

2015-ben az NTP-KKI-B kódjelű tehetséggondozást támogató pályázaton eredményesen pályáztunk, *„Fizikai mérések számítógép segítségével*" címmel. Az eredeti elképzelés szerint a már ismerős myDAQ adatgyűjtővel végezzük a méréseket, de felmerült két probléma. Az egyik az, hogy a myDAQ készülékek ára igen magas, a pályázatból nem tudtunk volna vásárolni megfelelő számút. A másik akadály a gyerekek érdeklődésének a hiánya volt, csak néhány gyereket érdekelt (akik egyébként a LabVIEW versenyeken sikeresen szerepeltek).

Felmerült egy másik lehetőség, az Arduino mikrovezérlő használata, amely ingyenes fejlesztő környezettel rendelkezik (szemben a LabVIEW borsos árával), maga az eszköz pedig fillérekbe kerül a myDAQ-hoz képest. Ráadásul C++ nyelven programozható, ami számunkra kifejezetten jó lehetőség, hiszen ezt tanulják a diákok tanórán is. Az Arduino platform nagyon sok gyereket érdekelt, hetedikesek is jártak a szakkörökre. Így került be a szakkörök közé az Arduino programozás is.

Egy szakköri projekt volt a következő:

Távolságmérés ultrahang szenzorral

Az Arduino impulzust küld a HC-SR04 felé (*trigger* jel), 10 µs ideig.

```
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
```

Ennek hatására az ultrahang modul egy 40 kHz-es jelcsomagot sugároz ki tárgy felé.

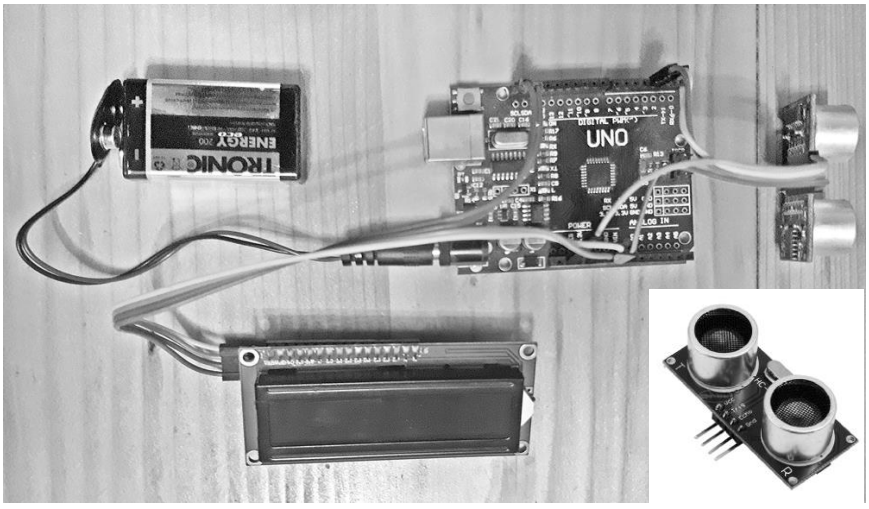
Ezután a visszavert impulzus *„hosszát",* idejét (ez az *echo* jel) méri µs-ban:

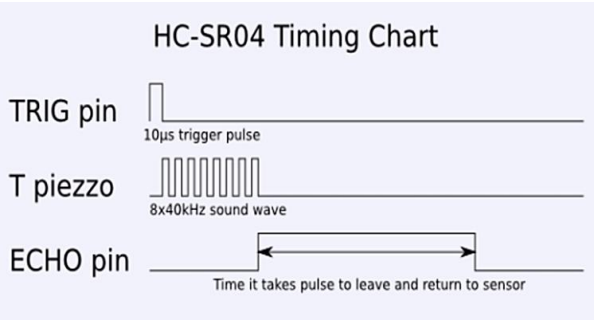
```
idoTartam = pulseIn(echoPin, HIGH);
```

Ez az időtartam megegyezik azzal az idővel, amit az *ultrahang tett meg a szenzorból a tárgyig, majd arról visszaverődve a szenzorig* – ezt számítja át a távolság meghatározásához.

```
s = v*t
```

$$d = \frac{s}{2} = \frac{v*t}{2} = \frac{340*100}{2} * \frac{t}{1000000}$$





Távolságmérés ultrahangos szenzorral

Összegzés

Egy olyan tehetséggondozó módszert próbáltunk bemutatni, ami az iskolánkban bevált, az eredmények igazolják, és jól tükrözik az eddigi pedagógiai munkánkat és szakmai tevékenységünket. Nem minden diákra és korosztályra lehet hatékonyan alkalmazni a 2.3 pontban megfogalmazott alapelveket, de a legtöbb motivált diáknál igen. A tudatosan megtervezett és jól bevált módszert folytatni szeretnénk a jövőben is.

Renzulli elmélete három területet jelöl meg a tehetség megalapozására:

- átlagon felüli képességek
- feladat iránti elkötelezettség
- kreativitás

Ha ezek közül legalább egy tulajdonsággal rendelkezik a tanuló, akkor van remény a tehetségének kibontakoztatására.

A pedagógusi pálya több pilléren nyugszik, ezek közé tartozik az elhivatottság, a szakmai és módszertani fejlődés. Nem szabad hagyni, hogy a lelkesedésünk az előttünk álló években lankadjon és a szakmai fejlődésünk megálljon. Arra törekszünk, hogy minél több diákkal szeretessük meg a programozást a továbbiakban is. Minél több területre legyen rálátásuk, ahol a programozásnak nagy szerepe van.

Felhasznált irodalom:

http://www.mateh.hu/tehetsegkonyvtar/Dr_Balogh_konyvek/Tehetseg_2011.pdf

Hivatkozási cikk:

http://fizikaiszemle.hu/archivum/fsz1610/FizSzem-201610.pdf

A versenyeredményeink következő helyeken találhatók:

http://nemes.inf.elte.hu/nemes_archivum.html

https://www.oktatas.hu/kozneveles/tanulmanyi_versenyek/oktv_kereteben/dijazottak_eredmenyek

Didaktikai számítógépes játékfejlesztés jelentősége a programozás tanításában

Stoffová Veronika

vstoffova@upol.cz / NikaStoffova@seznam.cz
Faculty of Education, Palacký University, Olomouc, CZ

Absztrakt. A tanulmány a programozás tanítására fókuszál projekt és problémamegoldás alkalmazásával. Az egyes projektek témája egy kiválasztott játék számítógépes verziója, amely bizonyos kívánt képességeket fejleszt. Például logikai, stratégiai vagy algoritmikusgondolkodást. A számítógépes játék implementálásához a tanulónak szüksége van elsajátítani egy bizonyos szinten a programozást, és a választott programozási környezetben tapasztalatokat és jártasságokat kell szereznie az alkalmazáskészítésből.

Kulcsszavak: programozás, programozás-tanulás, számítógépes játék, játékos programozás

1. Bevezető

A számítógépek és a digitális eszközök használata az iskolás gyerekek életében már napi szükségletté vált. Mobileszközöket nemcsak a kommunikációhoz, információszerzéshez, tanuláshoz, különböző szolgáltatások eléréséhez, hanem pihenésre és játékra is használják (Stoffová, 2016).

A kutatásaink eredményei azt mutatják, hogy a fiatalok (és nemcsak a fiatalok) leggyakoribb számítógépes foglalkozása a számítógépes játék (Basler, 2016). A számítógépes játékokat sem a felnőttek sem az idősek nem utasítják el. Minden korosztály kedvenc pihentető tevékenységei közé tartoznak a társasjátékok. Ide sorolhatók a tábla játékok, mint a dáma, sakk, malom és mások; a kártyajátékok (pl. hetes, máriás, pasziánsz), kirakójátékok, párosító játékok (pexeso), kitöltő játékok (keresztrejtvény, sudoku) stb. A játékok java részének a számítógépes verziója is implementálva van. Vannak olyan játékok, amelyeket egy játékos is játszhat és a számítógép csak játékos teljesítményét és a játékszabályok betartását értékeli. Gyakran a számítógép az ellenfél, partner, játéktárs, vagy egy a játékosok közül.

2. Didaktikai – oktató játékok

A tanulási célokat szolgáló játék, az (oktató játék) nemcsak a játékos szórakoztatására, mulatságára, pihenésére szolgál, hanem bizonyos didaktikus küldetése is van. Ezek közé sorolhatjuk a motivációt, figyelemfelkeltést, figyelemfenntartást, összpontosítást, gondolkodást és döntéshozás támogatását, spontán tanulást és sokféle képesség fejlesztését. Tehát a játékos tudásának, képességeinek és komplex személyiségének fejlesztése a cél. Ezért a didaktikus játékoknak és játékos tanulásnak és tanításnak pótolhatatlan helye van nemcsak az egyén fejlődésében, hanem a tanítási folyamatban is (Basler–Dostál, 2016; Chráska, 2016a; Chráska–Basler, 2016). A didaktikai játék megváltoztathatja a lecke egyediségét és sztereotípiáját, és szórakozássá, kikapcsolódássá, játékká varázsolhatja. Tehát a játék a tanítást aktív önkéntes, spontán nem terhelő tanulásra és aktív ismeretszerzésre változtathatja. A játékos tanulási folyamat a tanuló számára erős motiváció. A játékba beépített tananyag így könnyen megérthető, és a problémák megoldására spontán módon alkalmazható. Megfelelő didaktikai játékok hozzájárulhatnak a tanulók összes kulcskompetenciájának fejlesztéséhez (Chráska, 2016b, 2016c). A didaktikai játékok és a didaktikai céloktól függően néhány kulcskompetenciát is fejleszthetnek (Dostál, 2009). A didaktikai játékok az aktív tanulás módszerei közé sorolhatók (Stof-

fa–Végh, 2010; Végh–Stoffová, 2016, 2017). Ezért az informatika tanárképző programokban a didaktikai játék megprogramozását jelentős és hasznos témának tartjuk a jövendő tanítók és tanárok programozás tudásának elmélyítéséhez (Czakóvá, 2016a, 2016b).

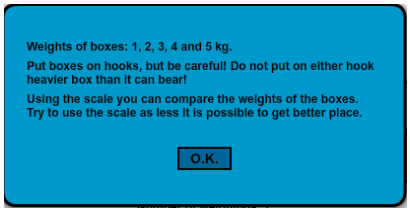
3. Számítógépes oktató játékok

Minden számítógépes játék egy egyedi grafikus környezetben játszódik le. A környezet bizonyos statikus és dinamikus elemekből épül fel – ezeket meg kell programozni. A dinamikus elemeknek bizonyos tulajdonságokat kell biztosítani, a viselkedésüket és reakcióikat vizualizálni vagy animálni kell Udvaros–Gubán, 2016). Minden játéknak saját szigorú szabályai vannak, amelyeket a játékosoknak be kell tartaniuk. Magának a játéknak rendelkeznie kell (rendelkeznie lehet) győztes stratégiával (amit a számítógép, mint játékos vagy ellenfél alkalmaz). A játék lefolyását, az egyes lépések megvalósítását szintén vizualizálni kell, hogy olvasható legyen a képernyőn a játék állása. Nem szabad megfeledkezni teljesítményértékelési szabályokról sem és a (legjobb) teljesítmények megőrzéséről. A lehetőségek bővíthetők a játék megállításával, megszakításával, folytatásával, ismétlésével, elemzésével, visszajátszásával, visszalépéssel stb.). Az alkalmazás továbbá színesebbé tehető háttérzenével, hangszekvenciákkal és különböző hang- grafikus- és egyéb figyelemkeltő és vonzó elemekkel.

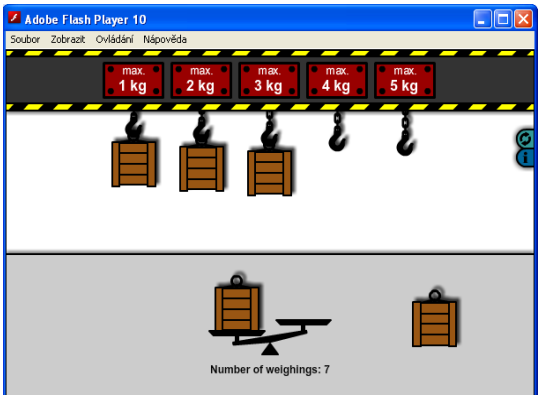
A didaktikai játékok implementálása (megtervezése, megprogramozása és megvalósítása) megfelelő feladat problémamegoldó- és projekttanításhoz. Az egyetemi tanulmányok első éveiben a számítástechnika, informatika, alkalmazott informatika, de főleg informatika tanár szakokon sikeresen beépíthető a bármely programozásra és szoftveralkalmazás készítésre orientált tantárgyba. A (didaktikai) játék implementálására a hallgató felhasználhatja bármely magasabb szintű programozási nyelvet, programozási környezetet, szoftveralkalmazás készítést támogató eszközt és ezeket kombinálhatja is (Stoffa–Végh, 2006a, 2006b; Végh, 2017). A játékszabályok és a játékmenet algoritmusba foglalása és megprogramozása, a játékstratégiák alkalmazása és az előbb említett részfeladatok beépítése, lehetőséget ad nemcsak az algoritmikus gondolkodás fejlesztésére, de az adattípusok és az adatstruktúrák, a grafikus könyvtárak és a grafikus motorok (engin-ek) effektív kihasználására, a játék grafikus felhasználói felületének kiépítésére és az erre szükséges jártasságok megszerzésére (Kožlej, 2018; Horváth–Stoffová, 2016).

A didaktikai számítógépes játékoknak bizonyos speciális jelei és jellemzői vannak. A didaktikai játékfejlesztőnek nem szabad megfeledkeznie:

1. a játékszabályok magyarázatáról, (általában szóban megfogalmazva (Lásd a 1. képet) esetleg vizualizálva, vagy animációval bemutatva), és lehetőséget kell teremteni a szabályok a játék bármely pontján való megtekintéséhez (Végh, 2011; Stoffová, 2003, 2004, 2008);
2. a játék kezelésének és irányításainak a szabályairól, még akkor sem, ha ezek standardokhoz idomulnak és a játék intuitív módon kezelhető. Itt is fontos a játékszabályok a játék bármely fázisában való elérhetősége;
3. arról a lehetőségről, hogy legyen a játék bármikor megszakítható úgy, hogy később tovább lehessen folytatni;



1. kép: Játékszabályok a ládarendező játékhoz



2. kép: ládarendező játék intuitív interaktív kezeléssel

4. a játék legyen interaktív (Pšenáková, 2016; Lapšanská, 2018);
5. a játék legyen egyszerű, intuitív módon vezérelhető (Stoffová–Kožlej, 2017; Stoffová–Horváth, 2017);
6. animációk segítségével bemutatható legyen a játék folytatásának lehetősége és a részprobléma megoldása – a játékos következő optimális lépésének meghatározását szimulálva (Végh–Stoffová, 2016; Végh–Stoffová, 2017);
7. lehetőség legyen a játékos lépéseinek elemzésére, hogy a játékos saját hibáiból is tanulhasson stb. (Stoffová, 2017; Stoffová, 2018);
8. lehetőség legyen a kommunikációs nyelv megválasztására;
9. lehetőség legyen a különböző nehézségi szint megválasztására;
10. a játékosok számára biztosítva legyen a tanácsadás és segítség;
11. rendelkezésre álljon a játékos sikerének és teljesítményének értékelése (pl. győztesek sorrendje táblázat formájában, vagy szóban a játék folyamán vagy a végén);
12. a számítógép átvehesse az egyik játékos szerepét (legtöbb esetben az ellenfél szerepét);
13. lehetőség legyen a játékos fejlődésének és hibáinak elemzésére, összehasonlítva előrehaladást „mester" teljesítménnyel (mintamegoldással). (Végh, 2011, Koreňová–Veress-Bágyi, 2017);
14. az időkövetésről (prémium odaítéléssel) ha a döntés (megoldás) ideje lerövidül vagy jóval az átlag alatt van stb.

Így folytathatnánk tovább, de reméljük, hogy az előbbi 14 pontban kifejeztük a lényeget és semmilyen fontos elvárás a didaktikai játékokkal szemben nem maradt le a listáról.

4. Számítógépes játékok a programozás tanításában

Az előbbi részben felsorolt tényekből kiindulva jó néhány didaktikai játékot és ötletes alkalmazást implementáltunk az algoritmikus gondolkodás fejlesztésére, az algoritmusok megértésére, a programozás tanításának támogatására és programozási jártasságok megszerzésére. Az implementált alkalmazások programozás tanításába való bevonását és az elért eredményeket már több alkalommal is bemutattuk és több cikkben publikáltuk (Stoffová–Végh, 2006a, 2006b; Végh –Stoffová, 2016). A játékos programozás tanulás, didaktikai játékok és az effektív tanítást és tanulást támogató szoftver alkalmazások fejlesztése és érvényesítése több informatika doktori iskola záródolgozat témájául is szolgált (Végh, 2017; Udvaros, 2017).

A játék számítógépes verziója nemcsak a játék szabályainak ellenőrzését tartalmazza, hanem egy bizonyos beépített játékstratégiát is, amely a számítógép győzelmét vagy jó teljesítményét eredményezi az ellenfél szerepében. Ezért a magasabb szintű programozás tanításában jelentős szerepet játszhat a különböző játékok számítógépes verziójának implementálása. Egy számítógépes játék megprogramozása nem egyszerű feladat. De maga a megválasztott téma a saját kreativitás és fantázia érvényesítése nagy motivációs erővel bír és segít az akadályok leküzdésében és a programozót remek teljesítményre serkenti. A kezdő programozó vágya (aki profi programozó szeretne lenni), hogy programokat saját elképzelése alapján készítsen, kiválasztott témában. A választott didaktikai játék megvalósítását jó ötletnek találtuk annak ellenőrzésére is, hogy a hallgató elsajátította-e az algoritmizálás és a programozás alapjait, és hogy képes-e egyéni vagy csoportos szoftver-készítésre. A hallgató egyben bizonyos tudást szerezhet a szoftverfejlesztés (software engineering) területén is.

5. A számítógépes játék implementálása

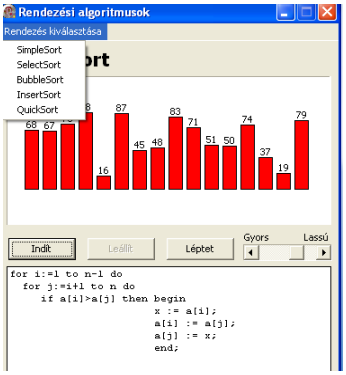
A számítógépes játék implementálása folyamán a diákok saját elképzelésük alapján dolgozzák ki a játék grafikus formáját, bebiztosítani kezelését, az interaktivitást, a játékszabályok beágyazását, nyerő stratégiák beépítését, a játék állását grafikusan kifejezni, az egyes jelenségeket, és a dinamikus eseményeket. A grafikus ábrázolást, az események vizualizálását és az animációkat, a projekt

követelményei szerint, a paraméterek értéke alapján algoritmussal kell irányítani (Végh, 2017). A szerzők a programkód megírásával valamilyen programozási nyelvben (pl. Java, Javascript, Delphi, C++), vagy valamilyen vizualizációs rendszerben (pl. JHAVÉ, Jeliot, BlueJ, ALVIS Live!), vagy kimondottan számítógépes játékkészítést támogató rendszerben (Unity3D Game Engine, Game Maker stb.) ennek eleget tehetnek. Egy másik megoldás lehet, valamilyen grafikus editor és animáció készítéséhez használható szoftver alkalmazása (pl. Adobe Flash, HTML5). Az Adobe Flash környezetben interaktív módon hatásos attraktív animációkat tudunk készíteni. (Lásd a 2. képet). Ezen eszközök felhasználásával készült jó néhány didaktikai alkalmazás, amely az interaktív konstruktivizmusra a felfedezésre, a kíváncsiság által vezérelt, a kellemes élményekre épülő tanulást támogatja. A tanulók és tanítók rendelkezésére áll a https://phet.colorado.edu/ címen. A diákok számára, mint motiváció a tanárok számára, mint módszertani eszköz. Manapság az előbb felsorolt megvalósításhoz használt eszközöket nagymértékben kiszorította a HTML5 és az előbbi weboldalon sok Flash és Java alkalmazást HTML5 alkalmazás váltott fel (Végh, 2017).

A grafikus módszertani alkalmazásokban a környezet és animációk elkészítéséhez elégséges a programozási nyelv grafikus könyvtára is, hisz azt akarjuk, hogy a hallgató programozni tanuljon meg és programozási technológiákat fejlesszen. Például a meglévő könyvtárat bővítheti (ha ezek még nem léteznek) a játék kezelésére szolgáló függvényekkel és procedúrákkal, így jártasságot szerez könyvtárépítésből is. Ilyenek például a menükészítés, a játék egérrel vagy billentyűzettel való kezeléséhez szükséges funkciók, különböző célokra szolgáló navigációs elemek stb. Hasznos a megoldásokat és lehetőségeket több programozási környezetben is bemutatni, ezeket összehasonlítani, elemezni és értékelni.

A programozás gyakorlaton olyan részproblémák megoldásával foglalkoznak, amelyek minden számítógépes játékfejlesztő számára hasznos. Ilyen feladatok a grafikus környezet megoldása (kialakítása, megprogramozása). A játék kezelése, beállítása, a játékos teljesítményének megőrzése, tárolása frissítése stb. A hosszú időtartamú játékoknál hasznos függvény a játék leállítása, megszakítása, folytatása, újraindítása.

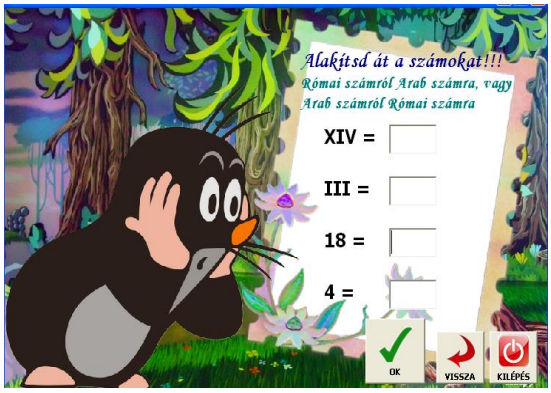
A következő képeken több ötletes megoldást találunk navigációs irányítógombok használatára és menü megprogramozására, annak grafikus formájára. A 3–5. képek két Delphi környezetben programozott alkalmazásból vannak képernyő másolattal készítve. A 3. kép egy rendezési algoritmusokat szemléltető alkalmazásból származik. A rendezési algoritmus kiválasztására legördülő menüt használ. Az üzemmód beállítását/megválasztását, menet közben a animált szimulációs kísérlet alatt lehet interaktív módon beállítani/kezelni. Ezen elemeket 2 kapcsoló Indít vagy Leállít. Ezek közül mindig csak egy lehet érvényes/aktív, az animáció fut, vagy le van állítva (hogy a tanító magyarázattal bővíthesse a látottakat).



3. kép: Rendezési algoritmusok

4. kép: Római számok – aktivitást ki választó gombjai

A következő az algoritmus léptetésére szolgáló nyomógomb, amely benyomásával 1 lépéssel tovább jutunk az algoritmus/program megvalósításában. Majd a nyomógombsor végén a csuszkával folyamatosan megváltoztatható animáció sebesség érték beállítására szolgáló elem foglal helyet. Még említésre méltó az algoritmus állását mutató sorkurzor, amely a végrehajtott program sorokat, kék színnel festi alá (Végh, 2011).



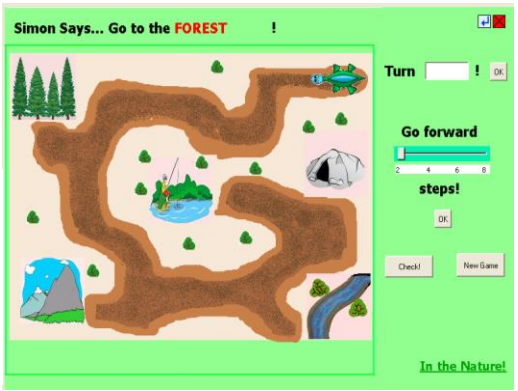
A 4. és 5. kép egy játékos római és arab számok leckét mutat be. Érdekes módon történik az aktivitás megválasztása (4. kép). Egy téglalap formájú érintőfelületre kell klikkelni az aktivitás megválasztására. A feleletek és megoldások megadását ablakocskába várja a rendszer, ahová klikkeléssel jutunk el. A feleleteket bármilyen sorrendben megadhatjuk és helyességét a játék bármely pontjában ellenőrizhetjük. Vigyázni kell arra, hogy ne találgasson a tanuló, főleg akkor ne, amikor ezért büntetve van.

5. kép: Átalakítási feladat

A 6. képen egy nyomógombokból összeállított menüt látunk, amely segítségével kiválasztjuk a lecke témáját. Az előbbi képek azt árulták el, hogy a hallgatók gyakran a második szaktantárgyuk támogatására szolgáló témát választanak. A 7. és 8. kép egy játékos angol nyelvtanulásra szolgál az alapiskola alsó tagozatán. Az 7. képen a nehézségi szint egyszerű megválasztását látjuk. A 8. kép a második nehézségi szinthez tartozó feladatot mutatja be. Az alkalmazás az Imagine mikrovilág-környezetben készült (Czakóová, 2016a, 2016b, 2017; Stoffová–Czakóová, 2015).

6. kép: Lecke kiválasztás menüből

7. kép: A nehézségi szint megválasztása



8. kép: A második nehézségi szinthez tartozó feladat

6. Befejezés

A cikk leírja az egyetem programozás tanításának problémáit. Rámutat arra, milyen jelentőséggel bír a probléma és a projektoktatás, ahol a projekt témája a választott didaktikai számítógépes játék, vagy a játékos tanulásra szolgáló szoftveralkalmazás készítése. A projekt megvalósítható egyéni vagy csoportos projektként. A programozási környezet vagy a programozási nyelv a számítógépes játék megvalósításához szabadon választható. A diákok bármely magasabb programozási nyelven dolgozhatnak, gazdag grafikus könyvtárral, vagy olyan programozási környezetet használhatnak, amely támogatja a számítógépes játékok létrehozását. Attól függ, hogy a programozók / fejlesztők / csapattagok hogyan tudják megosztani a munkát. A játék egyéni funkcionális részei külön készíthetők, vagy a munka rétegekre/részegységekre bontható (grafika, interaktivitás, algoritmusok stb.). A csapat minden egyes tagja azt csinálja, amihez ért, ami érdekli és ahol megfelelő teljesítményt tud nyújtani. Így teljesülnek azok a feltételek, amelyek a projekt sikeres befejezéséhez szükséges. Az egyik hallgató a játék grafikai tervezésére összpontosít, a másik algoritmusba foglalja a játékszabályokat és kidolgozza ezek betartásának ellenőrzését – a játék szabályainak való megfelelést a játékosok lépéseinél. A csapat többi tagja megosztja a további feladatokat, a győztes stratégiák kidolgozását, a játék leállításának és folytatásának módját, a játék nehézségi szintjének növelését, a játékosok teljesítményének kifejezését, rögzítését, frissítését stb. A projektben részt vevő hallgatók a projekt megoldása során tanulnak csapatban dolgozni, megismerkednek a team munkával, és gyakorlatilag felhasználják a szoftverfejlesztés területén szerzett elméleti tudásukat.

A számítógépes játékok és a játékalapú interaktív tanulás jelenleg nagyon népszerű. A felhasználók elvárják, hogy a digitális alkalmazásokat mobil eszközön is használhassák, és „útközben" is tanulhassanak/játszhassanak – játékkal tanuljanak. Saját játékok létrehozásának vágya elég nagy motiváció úgy a középiskolás diákok mind az egyetemi hallgatók számára, hogy programozzanak, és ezt önkéntesen és szívesen tegyék. Sok ilyen projektmegoldás sikeres TDK versenymunkává forr, vagy záródolgazat témaként szolgál.

A tanulmány a Palacký University Olomouc Tanárképző Karának IGA_PdF_2018_030 „An analysis of the use of educational computer games and online educational courses in secondary schools in relation to potential addictive behaviour in students in relation to gaming" támogatásával készült

Felhasznált irodalom

1. Basler, J. (2016). Počítačové hry a způsob jejich využívání u žáků základních škol. Trendy ve vzdělávání, 9(1), 10-19.
2. Basler, J., Dostál J. (2016). Analysis of studies focused on research of computer games' influence with an accent on education and people's psychics. In: ICERI2016 Proceedings. Seville, SPAIN: 9th International Conference of Education, Research and Innovation, s. 33-40. ISBN 978-84-617-5895-1. ISSN 2340-1095.
3. Chráska, M. jun. (2016a). Žáci gymnázia a míra jejich závislosti na počítačových hrách. Trendy ve vzdělávání, 9(1), 110-114.
4. Chráska, M., Basler J. (2016). Research of Computer game addiction among the 18-year- old students of general upper secondary schools in the Czech Republic. In: ICERI2016 Proceedings. Seville, SPAIN: 9th International Conference of Education, Research and Innovation, s. 69-78. ISBN 978-84-617-5895-1. ISSN 2340-1095. Indexováno ve Web of Science.
5. Chráska, M. (2016b). Computer Games – Preferred Way of Using ICT by Grammar School Students. In The European Proceedings of Social & Behavioural Sciences EpSBS. London: Elsevier Ltd., pp. 606–616. ISSN 2357-1330.DOI 10.15405/epsbs.2016.11.63
6. Chráska, M. (2016c). Grammar School Students and Their Typology According to Dependence on Computer Games. In SGEM 2016 Conference Proceedings. Sofia: STEF92 Technology Ltd., 2016, pp. 795–802. ISBN 978-619-7105-70-4. ISSN 2367-5659.DOI 10.5593/SGEMSOCIAL2016/B11/S03.101
7. Czakóová, K. (2016a). Tvorba vlastných aplikácií v Imagine. In: Úvod do programovania v prostredí mikrosvetov : vysokoškolská učebnica. Komárno : Univerzita J. Selyeho, 2016. 34-55. s. ISBN 978-80-8122-170-5.
8. Czakóová, K. (2016b). Creation small educational software in the micro-world of small languages. In: Teaching Mathematics and Computer Science. 14th volume, issue one, 2016/1, p. 117. Debrecen : University of Debrecen, 2016. ISSN 1589-7389
9. Czakóová, K. (2017). Élmény alapú programozás oktatás. In. : *Zborník medzinárodnej vedeckej konferencie Univerzity J. Selyeho – 2017* : *„Hodnota, kvalita a konkurencieschopnosť – výzvy 21. storočia"*. Sekcia informatických vied a IKT. Komárno : Univerzita J. Selyeho, 2017. s. 27 - 33. ISBN 978-80-8122-221-4.
10. Dostál, J. (2009). Výukový software a didaktické počítačové hry - nástroje moderního vzdělávání. Journal of Technology and Information Education. 1(1), 24-28.
11. Horváth, R., Stoffová, V. (2016). The Graphical support for the didactic games creation. In: *XXIXth DIDMATTECH 2016 : New methods and technologies in education and practice :2nd Part.* Ed. V. Stoffová, L. Zsakó, 1. vyd. Budapest : Eötvös Loránd University in Budapest : Faculty of Informatics, 2016, s. 67-82. ISBN 978-963-284-800-6
12. Koreňová, L. I., Veress-Bágyi, I., (2017). A kiterjesztett valóság alkalmazása az általános iskolai matematika tanulásban (Inquiry-Based Mathematics Learning by Applying Augmented Reality in the Primary School). In: *XXXth DidMatTech 2017 : New Methods and Technologies in Education and Practice : 2nd par*t. Ed. V. Stoffová a R. Horváth. 1. vyd. Trnava : Trnava University in Trnava, Faculty of Education, 2017, s. 75 – 86. ISBN 978-80-568-0073-7
13. Kožlej, J. (2018). Didaktické počítačové hry v projektovom a problémovom vyučovaní programovania. [Bakalárska práca]. – Trnavská Univerzita v Trnave. Pedagogická fakulta; Katedra matematiky a informatiky. - Vedúci: Prof. Ing. Veronika Stoffová, CSc., 2018. 58 s.

14. Lapšanská, Š. (2018). *Animačno-simulačné modely na podporu vyučovania základov algoritmizácie a programovania,* Diplomová práca, Trnavská univerzita v Trnave, Pedagogická fakulta, Katedra informatiky. Vedúci diplomovej práce: prof. Ing. Veronika Stoffová, CSc. Trnava: Pedagogická fakulta TU, 2018, 72 s.

15. Pšenáková, I. (2016). Interactive applications in the work of teacher. In *XXIXth DidMatTech 2016*. Budapest : Eötvös Loránd University in Budapest, Faculty of Informatics, 2016. ISBN 978-963-284-800-6. pp. 92–100. https://www.mii.lt/informatics_in_education/htm/infedu.2017.07.htm. WOS:000399818000007

16. Stoffa, V. (2004). Modelling and simulation as a recognizing method in the education, Educational Media International 40 (2), 2004. Taylor and Francis, London.

17. Stoffa, V., Végh, L. (2006a). A programozás tanításának és tanulásának elektronikus támogatása. Komárno :Selye János Egyetem, Eruditio-Educatio, I. évf., 3. szám, 2006/3, 105-113. ISSN 1336-8893.

18. Stoffa, V., Végh, L. (2006b). Guided animation of dynamic data structures. In *Third Central European Multimedia and Virtual Reality Conference*. Eger, Hungary, 2006, s. 175-179. ISBN 963-9495-89-1.

19. Stoffová, V. (2003). Počítač – univerzálny didaktický prostriedok 1. vyd. Nitra : Fakulta prírodných vied UKF v Nitre, 2003. 172 s. ISBN 80-8050-450-4

20. Stoffová, V. (2016). The Importance of Didactic Computer Games in the Acquisition of New Knowledge In: *The European Proceedings of Social & Behavioural Sciences EpSBS*. pp. 676-688. eISSN. 2357-1330. *http://dx.doi.org/10.15405/epsbs.2016.11.70*

21. Stoffová, V. (2017). Conceptual cybernetic model of teaching and learning. In: *Mathematical Modeling,* year 1, 2017, issue 2, p. 80 – 83. ISSN (WEB) 2603-2929, Print 2535-0986

22. Stoffová, V., Czakóová K. (2015). Prostredie mikrosveta v práci učiteľa základnej školy. In: *Kwartalnik naukowy NR.* 1 1(11(2015, s. 281-286. ISSN 2080-9069

23. Stoffová, V., Horváth, R. (2017). Didactic computer games in teaching and learning process Else Bucurest Bukurešť, The 13th International Scientific Conference, eLearning and Software for Education, Bucharest, April 27-28, 2017,10.12753/2066-026X-17-000

24. Stoffová, V., Kožlej, J.(2017). Didactic Computer Games. In: *New Methods and Technologies in Education and Practice : XXXth DIDMATTECH 2017 : 1st part*. Ed. V. Stoffová a R. Horváth. 1. vyd. Trnava : Trnava University in Trnava, Faculty of Education, 2017, s. 89 – 96. ISBN 978-80-568-0029-4

25. Stoffová, V., Végh, L. (2006a). Szemléltető animációk a programozásban. In *INFODIDACT 2010*, *3. Informatika Szakmódszertani Konferencia.* Editor: Zsakó, Szombathely, 2010. (príspevok na CD 6 s.)

26. Stoffová, V., Végh, L. (2006b). Počítačové hry a projektové vyučovanie programovania. (Computer games and project oriented teaching of programming). *XXIV International Coloquium on the Acquisition Process Management* : Proceeding of abstracts and electronic version of reviewed contributions on CD-ROM. Editori Eva Hájková a Rita Vémolová. Brno : University of Defence, Faculty of Economics and Management, 2006, s. 51 (abstrakt), celý príspevok na CD-ROM. ISBN 80-7231-139-5

27. Udvaros, J., Gubán, M. (2016). Demonstration the class, objects and inheritance concepts by software. ACTA DIDACTICA NAPOCENSIA 9:(1) Paper 3. 2016, ISSN 2065-1430

28. Végh, L. (2010). *Vyučovanie algoritmizácie a programovania hravou formou (Teaching algorithmization and programming playfully).* Paper presented at the XXVIII International Colloquium on the Management of Educational Process, Brno, Czech Republic.

29. Végh, L. (2011a). *From Bubblesort to Quicksort with Playing a Game (Hravou formou od bublinkového triedenia po rýchle triedenie).* Paper presented at the XXIX. International Colloquium on the Management of Educational Process, Brno, CZ.

30. Végh L. (2011b) From Bubblesort to Quicksort with Playing a Game. In: 29. International Colloquium on the M anagement of Educational Processes : Proceedings, Part 2. - Brno : University of Defence, Faculty of Economics and M anagement, 2011. - ISBN 978-80-7231-812-4, S. 539-549

31. Végh, L. (2017) Creating Interactive JavaScript Animations for Demonstrating Algorithms on One-Dimensional Arrays, In: New Methods and Technologies in Education and Practice : 30. DIDM ATTECH 2017. – Trnava : Trnava University in Trnava, 2017. - ISBN 978-80-568-0029-4, P. 75-80.
32. Végh, L., Stoffová, V. (2016): An interactive animation for learning sorting algorithms: How students reruced the number of comparisons in a sorting algorithm by playing a didactic game. In: *Teaching Mathematics and Computer Science*. Debrecen : Institute of Mathematics – University of Debrecen, 14th volume, issue one, 2016/1, s. 45–62. ISSN 1589-7389.
33. Végh, L., Stoffová, V. (2017). Algorithm Animations for Teaching and Learning the Main Ideas of Basic Sortings. In: Informatics in Education, Lithuania : Vilnius University. Vol. 16, No. 1, 2017, p. 121-140. ISSN: 1648-5831 (printed), 2335-8971 (online). DOI: 10.15388/infedu.2017.07 URL: https://www.mii.lt/informatics_in_education/htm/infedu.2017.07.htm. WOS:000399818000007
34. http://anim.ide.sk/ladakrendezese.php
35. http://ani.ide.sk/
36. https://phet.colorado.edu/

A programozás tanítása az alapiskolában – robotprogramozás

Stoffová Veronika[1], Katarína Pribilová[2]

{[1]veronika.stoffova, [2]katarina.pribilova}@truni.sk
Faculty of Education, University of Trnava, SK

Absztrakt. A cikk a Szlovákiában és Csehországban használt robotkészletekkel, robotépítéssel és robotprogramozással foglalkozik. Olyan programozható játékrobotokat és játékszereket mutat be, amelyek alkalmasak az algoritmikus gondolkozás fejlesztésére és a programozás alapjainak elsajátítására már az iskolára való felkészülés idejében. Áttekintést ad az oktatásban használható robotokról és programozható robotos játékszerekről. Bemutat néhány robotépítés és robotprogramozás versenyt is, amelyek motivációs ereje serkenti az általános iskolásokat saját kreatív ötleteik megvalósítására nemcsak a robot építésénél, de a szoftver megoldásoknál is. A diákok igyekeznek eredeti megoldásokat találni nemcsak a rendelkezésükre álló robotkészlet elemeinek felhasználásával, hanem más digitális berendezések és eszközök bevonásával is. A programozás tanulás ilyen módon élvezetesebbé válik, hisz az „életre kelő" objektumok nemcsak nagyon szórakoztatóak, de megfelelő motivációs erővel is bírnak.

Kulcsszavak: robot, robotika, oktatás, informatika, programozás

1. Bevezető

Az oktatási robotika az utóbbi időben nemcsak rohamosan fejlődik, de a népszerűsége is egyre inkább növekedik már az általános iskolások körében is. A robotok megjelenése az oktatásban erősen indokolt. Manapság sok nemzetközi és világ oktatástechnikai vásáron és kiállításon a főszerepet a programozható robotok játsszák. A játéküzletek polcain egyre több programozható játékszer és robotkészlet jelenik meg. Sok közülük árban is megfizethető a hétköznapi emberek számára is. Az első csoport a programozható, tehát programmal irányítható játékrobotok alkotják, szimpatikus állat, figura, autó, vagy más jármű formájában. A másik csoportot a gazdag kínálatból választható robotkészletek adják. Ebben az esetben a robotot előbb meg kell tervezni, fel kell építeni és csak ez után indulhat a programozás. A kezdők számára a programozás elsajátítását megkönnyíti a robotika bevezetése. A robotprogramozás a tanulást játékossá és élvezetessé varázsolja és így az iskola egy nagy játszótérré válik.

2. Oktatási játékszerek, programozható játékszerek és játékrobotok

Az oktatási játékok nem csak nagyszerűek és szórakoztatók, de alkalmasak bizonyos készségek fejlesztésére is. A játéknak nagyon fontos és pótolhatatlan helye van egy-egy személy életében és fejlődésében, amely születés pillanatától kezdve végigkíséri az életútján. Az újszülöttek különböző tárgyakkal, majd egyszerű játékokkal foglalkoznak, hogy megismerjék őket. A gyerek először a szüleivel kezd játszani, majd később, ahogy a gyermekek nőnek, egymással játszanak. Az iskola előtti intézményekben, a bölcsődékben, óvodákban a játék az egyik leggyakoribb gyermeki tevékenység, de nem hagyja el a gyerekeket sem az iskolában, sem a felnőttkorban. Az egyén fejlődése során a játék megváltoztatja jellegét, alkalmazkodik a személy korához, érdeklődéséhez, társas viszonyaihoz, de

mindvégig megtartja fontos szerepét és jelentőségét az egyén fejlődésében, akár társadalmi szinten, akár tudás szintjén, akár lelki és testi egészsége szintjén.

A játéknak fontos szerepe van az általános iskolások oktatásában is. A játék a diákok számára, hatékony tanulási segédeszközként szolgálhat. Ebbe a kategóriában sorolhatjuk az interaktív oktatási játékszereket és az ezen eszközökkel való játékokat, amelyek segítenek a tanulóknak különböző tevékenységek elvégzésében és kívánt képességek megalapozásában és fejlesztésében. Ezeknek a játékoknak köszönhetően a tanulók szórakoztató, játékos spontán módon megtanulják azt, amire szükségük van (elérik azt, ami a tanulás célja). Önkéntesen tevékenykednek, nem érzik a fáradságot, nem tekintik tanulásnak a teendőket, és így könnyedén elérik az oktatási célokat. (Koreňová, 2015; Koreňová–Veress-Bágyi, 2017).

2.1. Oktatási játékszerek

Ebben a kategóriában nagyon hosszú, szinte kimeríthetetlen listát lehetne felállítani, ezért csak 3 reprezentatív játékszert mutatunk be példaként.

2.1.1. Beatbo Cz

Olyan játékszer, amely gyerekeket nemcsak elszórakoztatja, hanem több oldalról is fejleszti. Három funkcióval rendelkezik, amelyekkel a gyermek megtanulhatja megérteni az ok-okozati összefüggéseket, fejleszti az érzékeit és finom és nagy motoros mozgáskoordinációját. Alkalmas már 9 hónapos korú gyermekek számára.

1. kép: Beatbo Cz (Fotó: www.alza.sk, 2018)

2.1.2. Kubo – mackó

A Kubo egy kölyök maci. Énekelni tud és történeteket mesélni. Segítségével egyszerű megoldani a szórakoztató rejtvényeket és matematikai játékokat. Programozható tablet vagy mobiltelefon használatával lehet irányítani. 3 éves kortól kezdve használható a gyermekek számára.

2. kép: Kuba - mackó (Fotó: www.alza.sk, 2018).

2.1.3. True4kids MagicPen – Varázstoll

True4kids MagicPen – varázstoll, a SmartPark alkalmazások vezérléshez használható tablet toll. A SmartPark egy forradalmian új tablet alkalmazás, amelyet az iskoláskor előtti és az alapfokú oktatásra terveztek. Segít a gyerekeknek nemcsak olvasni, hanem írni, rajzolni és számolni is. Számos tananyag, oktatási program, oktatási játék, hangos könyv, gyermekdal és interaktív könyv tartozik a készlethez. A matematika, a logika és a nyelv oktatása mellett az alkalmazás a művészi képességek és a gyermekek kreativitásának fejlesztését is segíti. Alkalmas 3-12 éves gyermekek számára.

3. kép: True4kids MagicPen – varázstoll (Fotó: www.alza.sk, 2018)

2.2. Programozható játékszerek

A programozható játékszerek és robotok lehetővé teszik a kritikus és kreatív gondolkodás fejlesztését, a különböző összetettségű problémák megoldásának megtervezését, a megoldás rendszeres tesztelését számítógép használata nélkül. A tanító számára lehetőség nyílik az informatikai tudás más szakterületeken való érvényesítésére. Így a számítástechnika, a számítógép az informatika információs és oktatás technológiák a tanítás eszközévé válnak (Stoffová, 2004).

Sok programozható játékrobotot, edukációs célokra szolgáló robotot és intelligens programozható robotot ismerünk. A különböző típusú robotoknak egyedi tulajdonságai, funkciói és vezérlési lehetőségei vannak.

A következő részben egy rövid áttekintést adunk a programozható játékrobotokról. A kiválasztott programozható játékrobotok listája koránt sem lehet teljes, hiszen mindig újabb és újabb játékrobotok jelennek meg a piacon.

2.2.1. Robot állatkák

A programozható vagy másmódon irányítható robot állatkák nagyon szimpatikusak, vonzóak és lekötik a kisiskolások figyelmét különböző dinamikus mozgás és kellemes hanghatásokkal. A gyártók sokat közülük már az iskola előtti korosztálynak, az óvódásoknak is javasolják. A robot állatkák különböző módon programozhatók, leginkább mobilalkalmazásokkal vagy számítógéppel működnek együtt (Čajkovič, 2017).

Robot állatkák példái:

2.2.1.1. Zoomer Bubblegum

Zoomer Bubblegum – hang- és fényeffektusokkal rendelkező kutyakölyök. 4 év fölötti életkor számára ajánlott.

4. kép: Zoomer Bubblegum (Fotó: www.alza.sk, 2018)

2.2.1.2. Zoomer Dino

Zoomer Dino – egy játék dinoszaurusz. Hihetetlenül reális mozgás benyomását kelti a „True Balance" technológiának köszönhetően. A dinoszaurusz érintéssel vagy hanggal vezérelhető. 5 év fölötti gyerekek számára ajánlott.

5. kép: Zoomer Dino (Fotó: www.alza.sk, 2018)

2.2.1.3. WowWee Chip

WowWee Chip – egy robot-kutyus. Gesztusokkal vagy Android vagy iOS. operációs rendszerrel rendelkező mobil eszközzel irányítható. 8 évesnél idősebb gyermekek számára ajánlott.

6. kép: WowWee Chip (Fotó: www.alza.sk, 2018).

2.2.1.4. Wonder Workshop Dash

A Wonder Workshop Dash egy olyan robot, amelyet csak Android vagy iOS operációs rendszerek alatt futó mobileszközzel lehet programozni. Reagál hangokra, táncra és énekre. A 6 éves kortól ajánlott.

7. kép: Wonder Workshop Dash (Foto: www.alza.sk, 2018).

2.2.2. Programozható játékok és robotok

2.2.2.1. Bee-bot

8. kép: Bee-boot.

A Bee-Bot egy aranyos méhecske, amelyet kifejezetten az iskoláskor előtti gyermekek számára terveztek. Díjnyertes robotról van szó, aminek programozásához nincs szükség számítógépre. Az robot irányítása a rajta lévő gombokkal történik. A gomb lenyomása parancsot közvetít. Négy vezérlő parancsot ismer: előre, hátra, jobbra, balra. Itt létezik egy ötödik parancs is, a szünet.

A Bee-Bot minden elvégzett utasítás után villog, és amikor befejezte a programot, akkor zenél.

Ezen egyszerű eszköz segítségével a különböző területeken játékos formában tudnak a kisiskolások új ismereteket szerezni. Alkalmas betűk és számok azonosítására, a síkban való tájékozódás begyakorlására stb. A gyerekek ezt használva játékosan fejlesztik a logikus gondolkodásukat, térbeli orientációjukat és tervezési képességeiket. A kezelőfelület egyszerűen használható (Zboran, 2017).

A gyerekek a méhecske hátán lévő négy gomb segítségével megtervezik a célhoz vezető lépéseket egy négyzethálón, majd végrehajtják azokat. Miután a felhasználó megadott egy lépéssorozatot, a parancsok végrehajtásához megnyomja a GO gombot, és a játék elindul. A teljes parancssorozatban legfeljebb 40 parancs állhat. Óvodásoknak és kisiskolásoknak ajánlott.

2.2.2.2. Pro-bot

A Pro-bot a Bee-bot nagytestvére, LCD kijelzős, LOGO teknőchöz hasonló rajzoló robot, az elfordulás mértékét fok pontossággal lehet beállítani. Számítógépre köthető és így a két eszköz előnyeit kihasználva irányítható, de számítógép nélkül is programozható. A csomag tartalmaz egy szimulációs programot is, amely segítségével megtervezhető a bejárandó útvonal, vagy a kirajzolandó ábra. A programot USB kábel segítségével lehet betölteni a robotba.

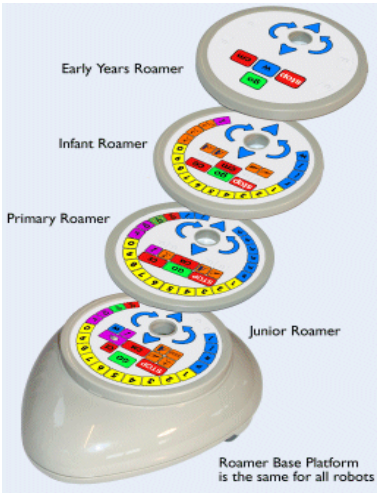
2.2.2.3. Compurobot

Ma már nem kapható, az 1980-as években készült oktatási célra a kisiskolások számára. LOGO-szerű parancsokkal volt irányítható. Rajzolni és zenélni is tudott.

10. kép: Compurobot

2.2.2.4. Roamer

A Roamer egy barátságos programozható robot, amely bevezeti a gyereket a programozás világába. A padlón gurulva járja be a beprogramozott útvonalat. Tollat adva hozzá, rajzolni is tud, mint a LOGO teknőc és zenélő funkcióval is rendelkezik. A robot tetején található panel segítségével lehet programozni. Ez a panel cserélhető nehézségi fokozatok szerint. Az első szint a bee-bot-hoz hasonló, a következő a pro-bot-hoz, majd a bonyolultabb programok megvalósítására is alkalmas panellal zárul a sor. Akárcsak az előzőekben, itt sincs szükség feltétlenül számítógéphez a programozásához. Különböző programok állnak rendelkezésre, melyek segítségével számítógépen, így aktív táblán is szimulálni tudják a választott út bejárását. Az általános iskolák alsóbb évfolyamaiban, főleg Angliában ez az egyik legelterjedtebb robot. Számos feladatgyűjtemény készült a vele való tanulás támogatására (Čajkovič, 2017). Több típusa is létezik (Standard Roamers, Activity Roamers, Turtle Roamer, SEN Roamer, Robotic Roamer).



11. kép: A Roamer robot különböző változatai

2.2.2.5. Code-a-Pillar

Code-a-Pillar egy programozható hernyó, amely alkalmas a gyermekek számára már az iskolára való felkészülés ideje alatt is. Természetesen csak azoknak, akik képesek elsajátítani a programozás alapjait már ebben a korban. A hernyó építésének elve az, hogy a felhasználó (játékos) könnyen beilleszthető szegmenseket szervezzen különböző kombinációkba, hogy a hernyót előre meghatározott cél felé irányítsa. Ez a játék támogatja a kísérletezést és olyan fontos készségeket fejleszt, mint a problémamegoldás, tervezés, menedzsment és kritikai gondolkodás. Az alapcsomag motoros fejet és nyolc könnyen kapcsolható szegmenset tartalmaz. A motoros fej 2 villogó szemmel, vicces hangokkal van felszerelve, amelyek megnyilvánulása biztosan magával ragadja a tanulókat. Amikor a gyerekek olyan szegmenseket csatolnak össze, amelyek a hernyót mozgásba hozzák és áthelyezik egy másik pozícióra, valójában egy parancssorozatot (programot) hoznak létre. A programozás azon szekvenciák összeállításából áll, amelyek egy előre meghatározott cél elérését jelentik.

12. kép: Code-a-Pillar (Fotó: Čajkovič, 2018).

2.2.2.6. WowWee - Robosapien X

Robosapien X olyan robot, amelyet egy ergonómikus vezérlővel vagy mobilkészülékkel vezérelhetünk (Android vagy iOS operációs rendszeren). A diákok figyelmét leköti a „kung fu" finom mozdulatainak és gesztusainak utánzata, a tánc és a rapping. A robot fejlett érzékelőinek köszönhetően, mozgásra, hangokra, érintésre, emelésre vagy rúgásra reagál. 67 előre beállított funkcióval rendelkezik, beleértve a dinamikus sétát, futást, vagy forgatást. A játékos ajánlott életkora 8 év fölött.

13. kép: WowWee - Robosapien X (Fotó: Čajkovič, 2018).

2.2.2.7. Ozobot (2.0 BIT)

Az Ozobot egy programozható robot mindössze 17 gramm tömeggel és nagyságát nézve belefér egy 2,5 x 2,5 x 2,5 cm^3 méretű dobozba. Kis méretének ellenére számos lehetőséget kínál a tanításhoz. A robot egyedülálló színkombinációt – színkódot (az úgy nevezett Ozokódot) ismer fel és ennek alapján tudja meghatározni a mozgása irányát, sebességét és végrehajtani speciális parancsokat. Az integrált érzékelőnek köszönhetően képes egy vonal mentén mozogni. A színsorok kombinálásával olyan parancsokat hozhatunk létre, amelyeket az Ozobot el tud végezni. Felismeri a fekete, zöld, piros és kék színeket (Fojtík, 2017).

Az Ozobot robot az ozoblockly.com weboldalon keresztül is irányítható. Az oldal tartalmaz egy parancsszerkesztőt, amely hasonló a Scratch programozási eszközhöz. Az összeállított program közvetlenül a robotra tölthető fel annak fényérzékelőjével. A robot a színváltozások sorrendjét egyszerűen beolvassa, ha a képernyő elé helyezzük. Még egy (harmadik) lehetőség is van az Ozobot robotot használatára, egy ingyenes applikáció. Mivel, hogy a robot a displayen fog mozogni, legjobb minél nagyobb képernyős (kilenc vagy több hüvely átlós) tablettet használni. Szerkesztői mini alkalmazások állnak rendelkezésre, amelyek az Android és az Apple Inc. operációs rendszerek (iOS) alatt működnek. A programozás egyik érdekes témája a tánc koreográfia.

14. kép: Ozobot 2 változata (Fotó: Čajkovič, 2018)

Az Ozobot alapvető előnyei a következők: könnyen kezelhető, szemléltető, motoros készségek fejlesztését elősegítő. Egyaránt alkalmas fiatalabb és idősebb diákok tanulásának támogatására. Részletes használati utasítás, sok feladat és egy ötletgyűjtemény áll az érdeklők rendelkezésére.

2.3. Az oktatásban használható robotok és robotkészletek

Nem egyszerű a programozható robotok és robotos játékok között határvonalat húzni. Ugyanúgy nem könnyű a különbséget tenni játék és játékos tanulás között. Már ez előbb bemutatott irányítható játékoknak is van (vagy lehet) didaktikus hozadéka és hasznos módszertani eszközként szolgálhatnak például az algoritmus és logikai gondolkodás fejlesztése területén.

Az alap elektronikai robotkészletek lehetővé teszik a tanulók számára, hogy sokféle modellt építsenek fel forrasztás nélkül, színes lámpákkal, hangjelzőkkel és egyéb más szenzorokkal és különböző módon csatlakozó berendezéssekkel. Az idősebb tanulók saját robotjukat is felépíthetik és beprogramozhatják irányítását. Az elektronikus készletek támogatják a tanulók alaptudásának kialakítását, fejlesztését fizikából, elektrotechnikából, mechanikából stb., és a jelenségek és összefüggések megértését.

Az oktatási robotkészletek használata hozzájárulhat a számítástechnikai és informatika oktatásához és növeli a programozási nyelvek varázsát (Veseloská–Mayerová, 2015, 2017; Veselovská, 2015). A programozás vonzerejét nagymértékben befolyásolja az is, hogy a robotprogramozás interaktív, játékos és tanítása összekapcsolható az iskolán kívüli tevékenységekkel, például különböző robotversenyekre való felkészüléssel (Mayerová–Veselovská, 2016;. Ezek lehetővé teszik a diákok számára, hogy összehasonlítsák tudásukat nemzeti és nemzetközi szinten is. Az edukációs robotika bevezetése lehetőséget ad a tanítás különböző formáinak, módszereinek használatára, mint például a projekt munka, a probléma megoldás, a kooperatív tanulás, de a video-oktatóanyagok és elektroni-

kus-oktatóanyagok használatát is lehetővé teszi, amelyek a programnyelvek tanítására rendelkezésre áll, vagy a csatolt programcsomag szerves részét képezi (Gujberová–Mayerová, 2016).

2.3.1. Ma-vin – robot (készlet)

15. kép: Ma-vin – robot

A Ma-vin egy könnyen összeépíthető, számítógép segítségével egyszerűen programozható robotkészlet. A mellékelt vizuális programozói környezet kezelése a használati utasítás mintafeladataival könnyen megtanulható. Ezáltal megismerhető a programozás alapvető logikája, valamint a C-be átfordított programok is tanulmányozhatók. A haladók WINAVR segítségével közvetlenül C-ben is programozhatják a robotot (Takáč, 2016; Stoffová–Takáč, 2013). A hat beépített érzékelő segítségével életszerű alkalmazások hozhatóak létre, melyek segítenek megismerkedni a mikroelektronikával, programozással és robotikával. *Főbb jellemzői:* Összeszerelése egyszerű, forrasztást nem igényel (az összeszerelés kb. 10 percet vesz igénybe); Egyszerű, vizuális programozási környezet;

WINAVR segítségével C-ben is programozható; Érzékelői segítségével reagál a környezetében történt változásokra; Moduláris felépítésű, bővíthető; Remek eszköz a tanuláshoz és oktatáshoz;

A Ma-vin alkalmazásai: Felismer és követ egy rajzolt vonalat; Reagál a fényre, hangra, érintésre; Kikerüli az útjába kerülő tárgyakat;

A készlet tartalma: Alaplap: Atmel processzor (TTMEGA64L), hangérzékelő, 1 LCD kijelző foglalat, 5 Ma-vin modul foglalat; 6 fényérzékelő;

Ma-vin érzékelő modulok: LCD kijelző, berregő, hangszóró, LED modul, érintés érzékelő, kapcsoló

Alapelemek: 2 motor, 2 kerék, USB kábel, váz, elemtartó, csavarok, szoftver CD, használati útmutató magyar, angol, német, francia, olasz, spanyol)

2.3.2. Asuro robot

Az Asuro egy mini mobilrobot, melyet a Német Űrkutatási Intézet (DLR) tervezett oktatási célokra.

16. kép: Asuro robot

Számítógép segítségével C nyelven programozható. Összeállítása gyakorlattal rendelkezőknek egyszerű feladat, de kezdők által is megvalósítható. Összeszerelése forrasztást igényel. A nyomtatott áramkörökön kívül szabványos alkatrészeket tartalmaz, programozásához freeware alkalmazások használhatók. Kitűnően alkalmas hobbi célokra, különösen ajánlott iskolai-, egyetemi projektekhez és felnőttoktatáshoz is.

A termék felépítése: Atmel AVR RISC mikroprocesszor; Két függetlenül vezérelt motor; optikai vonalkövető; 6 ütközésérzékelő kapcsoló; 2 odometer (megtett távolságmérő) szenzor; 3 LED; Infravörös adóvevő a PC-vel való programozáshoz és irányításhoz.

Említést érdemelnek a micro:bit és arduino is, mint programozható mikroprocesszorok/mikrovezérlők. Ezek fokozatosan teret hódítanak főleg a szakközépiskolai képzésben. Programozásukhoz szintén van vizuális felület és egyre gyakrabban megjelennek az általános iskolában is min későbbi alternatív lehetőségek (URL5, URL6, URL7)

2.3.3. LEGO Mindstorm NXT

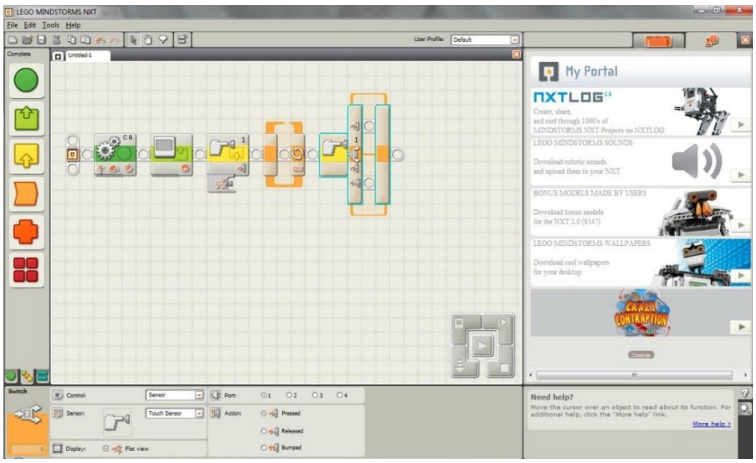
Robotok komolyabb programozásának tanulásához a LEGO Mindstorm NXT az egyik legjobb választás. A teljesen kezdőtől a profi programozóig mindenki megtalálja a számára kihívást jelentő feladatot. Tehát mindig van továbblépési lehetőség. Nagyon sok programozási nyelvvel programozható (NXT-G, NBC, NXC, RobotC, LeJoS,…) Az alapkészlet bővíthető további szenzorokkal (hangérzékelővel, iránytűvel, gyorsulásmérővel, okos telefonnal, GPS-sel…), napelemes panellel, távirányítóval, további építőelemekkel (LEGO Technic),…

Az alapkészlet tartalma: 612 alkatrész, NXT LEGO kocka, 32 bites mikroprocesszorral, nagy mátrix kijelzővel, 4 bemeneti és 3 kimeneti port, Bluetooth és USB csatlakozás; 3 interaktív szervo motor; 4 szenzor: ultrahang; 2 db érintés, szín (szín- és fényérzékelés).

Jellemzői: Blokk-alapú drag-and-drop programozási felület (NXT-G). Építési útmutatók több nehézségi szintű megépítendő modellel (PC és MAC kompatibilis). LEGO Mindstorm NXT programozása NXT-G-vel a legegyszerűbb. A robot építéséhez hasonlóan az irányító program grafikus építőkockákból – ikonokból rakható össze (Price–Barnes, 2017; Zhang, 2016).

Felhasználóbarát felület: Könnyen megérthető; Rendelkezésre állnak lépésről-lépesre leírt példaprogramok; Bonyolultabb programok már nem annyira áttekinthetőek; A lefordított kód a többi programozási nyelven készült programokhoz képest lassan fut és sok memóriát igényel (Park, 2014).

17. kép: NXT LEGO kocka



18. kép: LEGO Mindstorm NXT-G programozás környezet

19. kép: Robot LEGO Mindstorm NXT-ből

2.3.4. LEGO Mindstorms EV3

A Lego Mindstorms EV3 jelenleg a Lego robotkészletek legújabb verziója. Nagyon népszerű a robotika tanulási célok elérésére. A legfontosabb előnye az, hogy a Lego építő készletet a gyerekek már korai gyermekkorukból ismerik. További előnye különösen a modularitás, az egyszerűség és a hozzáférhetőség (Zboran, 2017).

A Lego cég robotfejlesztései 1998-ban RCX robotkészletekkel kezdődtek. 2006-ban az NXT új verziója került piacra. Jelenleg a legújabb verziója az EV3 használt, amely 2013-ban volt piacra bocsájtva. Minden készlet tartalmaz egy programozható intelligens kockát, amelyen (portok) bemenetek (érzékelők) és a kimenetek (motorok hozzácsatolásához) találhatók. A legújabb LEGO Mindstorms EV3 oktatási verziója tartalmazza az „okos" kockát Bluetooth technológiával, két nagy és egy közepes motort, érintés, szín és ultrahangos érzékelőket, a kerekek és nagy műanyag abroncsokat és feltölthető akkumulátort. Az akkumulátor miatt az intelligens kocka körülbelül 0,8 mm-rel magasabb, mint a Home Edition verziója. Lego Mindstorms legismertebb programozási környezete az EV3-G, EV3 Programmer APP, a Scratch, az OpenRoberta és így tovább. (www.lego.com). Használata 10 éves kortól javasolt.

20. kép: Robot LEGO Mindstorm EV3-ból (Fotó: www.alza.sk, 2018).

3. Mi a helyzet az iskolákban?

Annak ellenére, hogy a robotépítés és robotprogramozás iránt a tanulók körében nagy az érdeklődők száma, jelenleg még nem reális, hogy ezek a témakörök az informatika tantárgy szerves részei lehessenek. Az iskolák ugyanis nincsenek kellőképpen felszerelve robotkészletekkel. Bizonyos állami támogatásból, sikeres pályázatok alapján az általános iskolák hozzájutottak robotkészletekhez, de az említett úton csak 1-2 robotkészletet tudtak szerezni iskolánként. Szerettük volna feltérképezni a helyzetet, hogy milyen mértékben vannak Szlovákiában az iskolák ellátva robotkészletekkel és hogyan érvényesülnek ezek a tanításban. Egy kérdőíves kutatás érdekes eredményeivel szeretnénk megismertetni a továbbiakban az olvasót.

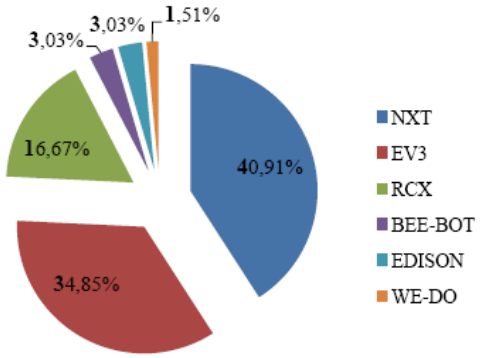
A kidolgozott kérdőív olyan iskolákra volt szétküldve, ahol megtudtuk, hogy hivatalosan birtokolnak robotkészletet, amelyet valamelyik nyilvános projekt megoldásába való bekapcsolódás alapján szereztek. A 56 kérdőívből, csak 32 kaptunk vissza kitöltve. A környékbeli iskolák személyes magánlátogatásával a kitöltött kérdőívek száma 42- re növekedett.

Megkérdeztük milyen robotkészletek vannak az adott iskolában és mire használják őket. Léteznek-e az iskolában robotikára orientált szakkörök és a tanulók részt vesznek-e robotépítő és robotprogramozó versenyeken. Érdekelt bennünket az is, hogy milyen módon befolyásolja a diákokat a robotkészletek használata a pályaválasztásban és változik-e a továbbtanulási szándékuk és az életpálya orientáció a szakkör munkájába bekapcsolt és a robotversenyeken sikeresen szereplő tanulóknál.

4. A kutatás eredményei

A kérdőív kérdéseire kapott válaszokból, csak néhány kiválasztottat kérdést értékelünk, amelyeket érdekesnek és értékesnek tartunk.

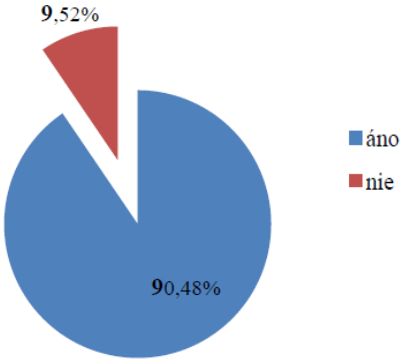
Érdekes például, hogy milyen robotkészletek vannak az iskolákban. A 1. ábrából világos, hogy a legelterjedtebb robotkészlet a szlovákiai alapiskolákban (általános iskolákban) a LEGO Mindstorm NXT (40,91 %). A LEGO Mindstorms EV3 a második pozíciót foglalja el (34,85 %). Majd a régebbi LEGO Mindstorms RCX a harmadik a sorban (16,68 %). A többi listára került 3 robotkészlet (BeeBot, Edison, WepDo) összesítve csak 1-2 példányban fordul elő. A robotkészletek finanszírozásának forrásai különbözőek. Az táblázat 1 áttekintést ad arról, hogy sikerült robotkészletet szereznie az iskolának. A 2. ábra mutatja százalékban kifejezve, hány iskolában működik informatika szakkör.



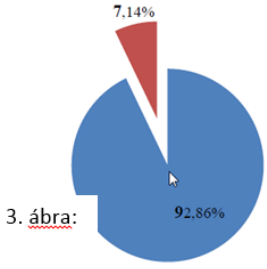
1. ábra: A robot típusok elosztása az alapiskolákon

A finanszírozás forrása	Létszám	%
Az iskola saját költségvetése	19	32,76
Szülői társulás	11	18,97
Projektek	10	17,24
Támogatók	7	12,07
Grantok	4	6,90
Alapítványok PONTIS	3	5,17
Ajándék	3	5,17
Szülők	1	1,72

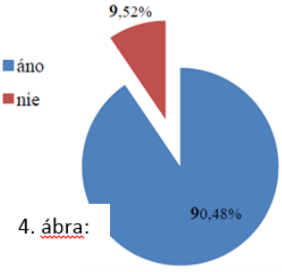
1. táblázat: A robotkészletek finanszírozásának forrásai



2. ábra: Hány iskolában van informatika szakkör (áno-igen, nie-nem)



3. ábra:



4. ábra:

3. ábra: Hány iskolában foglalkoznak robotikával az informatika szakkör keretein belül (áno-igen, nie-nem)

4. ábra: Hány iskola vesz részt robotprogramozási versenyeken azok közül, amelyek robotikával foglalkoznak (áno-igen, nie-nem)

A 3. ábra azt mutatja, hogy hány iskolában foglalkoznak robotikával az informatika szakkör keretin belül.

A 4. ábra pedig látjuk, hogy hány informatika szakkör készíti fel a tanulókat versenyekre és hány iskola vesz részt robotprogramozási versenyeken.

5. Robotépítő és robotprogramozó versenyek

Amikor a diákok robotokat tervezni, építeni és programozni tanulnak, szívesen mérik össze tudásukat nemcsak csoportonként szakkörtársaikkal, hanem nemzeti és nemzetközi versenyeken is. Szlovákiában évente sok olyan versenyt szerveznek, amelyek robotépítéssel és robotprogramozással foglalkoznak. A versenyzők, összehasonlítják tudásukat és készségeiket a versenytársaikkal, és egyúttal új inspirációt is nyernek. A versenyeken való részvétellel kapcsolatos kérdésekre adott válaszok eredményeit a 4. ábra szemlélteti, amelyből világos, hogy majdnem minden robotikával foglalkozó iskola, kihasználja a tanulók versenyszellemét és a szakkör aktivitásainak keretein belül egyben felkészülnek a robotépítő és robotprogramozó versenyekre. Sokszor a versenyen való részvétel szolgál motivációként és hajtómotorként a szakkörök munkájában.

Ezen versenyek közé sorolhatjuk a First Lego League, Robohranie (Robojátszás), RoboCup Junior, Legobot stb. Sok műszaki szakközépiskola és egyetem is szervez ilyen versenyeket. Sokszor az iskola „nyílt napok" keretein belül valósítja meg az ilyen akciót és egyben ezt jó alkalomnak tartja az iskola bemutatására (és sokszor új diákok toborozására is).

Istrobot – Pozsonyban a Szlovák Műszaki Egyetem Villamosmérnöki és Számítástudományi Karán szervezett nemzetközi robotverseny.

RoboCup Junior – robotépítés és robotprogramozás verseny az általános és középiskolások számára.

Trencsényi robotikus napok – nemzetközi robot verseny az általános és középiskolák számára.

SzaKI öttusa (Soškársky päťboj) Dubnica nad Váhom-i Műszaki Szakközépiskola által szervezett robotverseny.

A leggyakoribb feladatok ezeken a versenyeken a robot egy előre meghatározott pályán történő mozgatása, különféle akadályokkal való megküzdése, egy ismeretlen pálya felismerése és bejárása a lehető legrövidebb idő alatt és egyéb más feladat elvégzése.

Ezek a versenyek nagy motivációt és ösztönzést jelentenek a tanulók számára, mivel bemutathatják technikai és programozási képességüket, szembenézhetnek és megküzdhetnek az akadályokkal és összemérhetik erejüket versenytársaikkal. Pozitív hatással vannak a diákok kreativitására, az adott problémák azonosítására és megoldására. A projekt bemutatása, a megoldás indoklása és megvédése a kommunikációs képességek fejlesztésére, a szókincs javítására, az új technológiák használatára, az önbizalom erősítésére, a felelősségérzet fokozására, a technikai készségek fejlesztésére valamint a csapatmunkára szolgálhat.

21.kép: A SzaKI öttusa (Soškársky päťboj) **verseny támogatói**

Mivel Szlovákiában jelenleg virágzik az autógyártó ipar, ahol sok ipari robot érvényesül, és ahol vonzó munkalehetőség vár az érdeklődőkre és a szakmailag felkészültekre, nincs hiány az olyan cégekből, akik hajlandóak a tanulók és iskolák támogatására az edukációs robotika fejlesztésére és az iskola felszereltségének jobbítására. A 6. ábra mutatja, hogy egy regionális jelentőségű szakközépiskola mennyi támogatót szerzett egy ilyen verseny megszervezésére, a győztesek jutalmazására, a tehetségek felkutatására és ezek képességeinek fejlesztésére.

6. Befejezés

A kutatási eredmények azt mutatják, hogy az iskolákban a robotkészletek használata az algoritmikus és logikus gondolkozás fejlesztésére rendkívül alkalmasak. A tanulók szívesebben programoznak olyan feladatokat, amelyek saját fantáziájuk alapján épített objektumok (robotok) irányítására, mozgatására, egyszerű majd összetettebb parancsok elvégzésére irányul. A szakkörök munkájában előnyösen összekapcsolható a konstruáló képesség, a kreativitás, a fantázia a saját ötletek megvalósítása, a csoportmunka és az akadályok közös legyőzésének öröme. Maga a vizualizált (ikonikus) programozás rendkívül szemléltető. Egyszerűen interaktív módon, „az irányító szerkezetek" és funkcionális egységek összeépítésével megkapjuk a programot. A leggyakrabban használt robotkészletek a LEGO sorozatból származnak. A tanárok válaszai alapján elmondhatjuk, hogy a technikai készségek és a programozás készségek fejlesztése a robotépítés és a robotprogramozás segítségével hatékony, spontán és játékos. Ezt bizonyítják a robotversenyek résztvevői és az elért eredmények is.

A robotépítés és a robotprogramozással kapcsolatos versenyek motivációs ereje és az ezekbe való bekapcsolódás serkenti az általános iskolásokat saját kreatív ötleteik megvalósítására nemcsak a robot megépítésénél, de a szoftver megoldásoknál is. A tanulók igyekeznek eredeti megoldásokat találni nemcsak a rendelkezésükre álló robotkészlet elemeinek felhasználásával, hanem más digitális berendezések és eszközök beépítésével is. A programozás-tanulás ilyen módon az iskolások számára sokkal nagyobb élménnyé válik, hisz életre keltik a saját elképzeléseik alapján felépített robotokat.

A tanulmány a Szlovák Kulturális és Edukációs Ügynökség (KEGA) 012TTU 4/2018: Interactive animation and simulation models in education és 015TTU 4/2018: Interactivity in electronic didactic applications projektjeinek támogatásával készült.

Felhasznált irodalom

1. Čajkovič, M. (2017): Vybrané aktivity realizované robotom Ozobotom v predmete informatika (Písomná práca k druhej atestácii), Trnavská univerzita. Pedagogická fakulta, katedra matematiky a informatiky. Trnava: Pedagogická fakulta TU, 2017. 79 s.
2. Fojtík, R. (2017): Výuka programování pomocí robota Ozobot. Zborník z konferencie DidInfo&DidactIG 2017, FPV, Banská Bystrica, 2017, ISBN 978-80-557-1216-1
3. Gujberová, M., Mayerová, K., Veselovská, M.(2014): Edukačná robotika na 2. stupni ZŠ a zručnosti pre 21. storočie. Zborník z konferencie DidInfo 2014, FPV UMB, Banská Bystrica, Slovensko, 2016, ISBN 978-80-55 ISBN 978 - 80 -557-0698-6
4. Koreňová, L. (2015): What to use for mathematics in high school: PC, tablet or graphing calculator. In *International Journal for Technology in Mathematics Education*, vol. 22, No. 2, 2015, pp. 59–64. ISSN 1744-2710. WOS:000389187500004.
5. Koreňová, L. I., Veress-Bágyi, I. (2017): A kiterjesztett valóság alkalmazása az általános iskolai matematika tanulásban (Inquiry-Based Mathematics Learning by Applying Augmented Reality in the Primary School). In: *XXXth DidMatTech 2017 : New Methods and Technologies in Education and Practice : 2nd par*t. Ed. V. Stoffová a R. Horváth. 1. vyd. Trnava : Trnava University in Trnava, Faculty of Education, 2017, s. 75 – 86. ISBN 978-80-568-0073-7
6. Mayerová, K., Veselovská, M. (2016): Aktivity s LEGO WeDo pre 1. stupeň ZŠ. Zborník z konferencie DidInfo2016, FPV UMB, Banská Bystrica, Slovensko, 2016, ISBN 978-80-557-1082-2
7. *Nabídka LEGO Education pro základní, střední a vysoké školy. Vzdělávací programy pro starší školní věk.* [online]. Dostupné na internete: < http://www.eduxe.cz/les/> [utoljára megtekintve: 2018.11.25.]
8. Ozobot [online<http://cdn.shopify.com/s/files/1/1059/8266/products/bit-white-2_grande.png?v=1448268411>]. [utoljára megtekintve: 2018.03.29.]
9. Park, E. J.: Exploring *LEGO LEGO ® Mindstorms ® EV3: Tools and Techniques for Building and Programming Robots*. Printed in USA 2014. ISBN 978-1-118-87974-0.
10. Petrovič, P.: *Výuka programovania pomocou grafických robotických programovacích jazykov pre začiatočníkov a pokročilých.* [online]. <http://dai.fmph.uniba.sk/~petrovic/pub/didinfo2012/didinfo_petrovic_2012.pdf> [utoljára megtekintve: 2018.11.20.]
11. Price, T., Barnes, T.: *Comparing Textual and Block Interfaces in a Novice Programming Environment* 2015 [online]. internete:<http://www4.ncsu.edu/~twprice/website/files/ICER%202015%20Slides.pdf > [utoljára megtekintve: 2018.10.21.]
12. Stoffa, J., Stoffová, V. (1997): Medziodborové vzťahy technickej výchovy a informatiky. In Zborník 5 z vedeckej konferencie Vzdelávanie v meniacom sa svete, Univerzita Konštantína Filozofa v Nitre, jún 1997. ISBN 80-967339-9-0.
13. Stoffová, V. a kol. (2001).: *Informatika, informačné technológie a výpočtová technika. Technologický a výkladový slovník*. Vydanie. Nitra , Univerzita Konštantína filozofa, 219 s. ISBN 80-8050-450-4
14. Stoffová, V., Takáč, O. (2013): Robotické stavebnice v príprave učiteľov informačnej výchovy (Robot kits in teachers preparation for information education). In Havelka, M., Chráska, M., Klement, M., Serafín, Č. (ed.): *Trendy ve vzdělávání 2013*. Olomouc : agentura GEVAK s.r.o., 2013. 315-322. s. ISBN 978-80-86768-52-6 / ISSN 1805-8949
15. Takáč, O. (2016): Výučba robotiky pomocou lego MINDSTORMS NXT = Teaching robotics through Lego MINDSTORMS NXT 2016. DOI 10.15584/eti.2016.1.31. In: DIDMATTECH 2015 = Education - technology - computer science : Edukacja – Technika – Informatyka, Rzeszów : Uniwersytet Rzeszowkiego, 2016. ISSN 2080-9069, No. 1 (2015), p. 219-223.
16. Veselovská, M„ Majerová, K. (2017): Aktivity s LEGO WeDo vo vyučovaní informatiky. Zborník zkonferencie DidInfo&DidactIG 2017, FPV UMB, Banská Bystrica, 2017, ISBN 978-80-557-1216-1

17. Veselovská, M„ Majerová, K. (2015): Aktivity s LEGO WeDo na 2. stupni ZŠ. Zborník z konferencie DidInfo 2015, FPV UMB, Banská Bystrica, 2015, ISBN 978-80-557-0852-2
18. Veselovská, M.: *Uplatnenie robotiky v informatike na základných školách.* Bratislava 2010. Bakalárska práca. [utoljára megtekintve: 2018.03.25.] Dostupné na internet <http://www.miska.own.sk/prace/BakalarskaPracaVYSLEDNA1.pdf>
19. Zboran, M. (2017): Využitie robotických stavebníc na základných školách. Rigorózna práca, Trnavská univerzita. Pedagogická fakulta, katedra matematiky a informatiky. Vedúci rigoróznej práce: prof. Ing. Veronika Stoffová, CSc. Trnava: Pedagogická fakulta TU, 2017. 133 s.
20. Zhang, J. (2016): *Teaching Artificial Intelligence Using Lego*, SESSION THE USE OF ROBOTS AND GAMIFICATION IN EDUCATION, Las Vegas, 2016, ISBN 1-60132-435-9, [online]. [utoljára megtekintve: 2018.3.12.] Dostupné na internete:< http://worldcomp-proceedings.com/proc/p2016/FEC3679.pdf >
21. URL1: Európai Tanács. Lisszaboni Európai Tanács, 2000. március 23–24. Az elnökség következtetései. (European Council. Lisbon European Council 23 and 24 March 2000. Presidency conclusions.)
22. URL2: http://www.beebot.org.uk/
23. URL3: http://www.active-robots.com/products/robots/mavin.shtml
24. URL4: http://mindstorms.lego.com/en-us/default.aspx
25. URL5: https://www.ucimeshardverom.sk/
26. URL6: https://navody.arduino-shop.cz/arduino-projekty/
27. URL7: https://navody.arduino-shop.cz/zaciname-s-arduinem/

Valós idejű funkcionalitás Windows-ban

Szabó Dávid[1], Dr Illés Zoltán[2], Heizlerné Bakonyi Viktória[3]

{ [1]sasasoft, [2]illes, [3]hbv }@inf.elte.hu
ELTE IK

Absztrakt. Az elmúlt években tapasztalhatjuk, hogy az időzítések, határidők szerepe az élet minden területén egyre jelentősebbé vált, így van ez a számítógépek világában is, elterjedt a valós idejű alkalmazások iránti igény. Szoftverek, melyeknek nem elég gyorsan reagálniuk, a választ határidőre kell előállítaniuk. Az átlag felhasználók között egyik legelterjedtebb operációs rendszer, a Windows, ennek ellenére teljes funkcionalitásában nem támogatja a valós idejű alkalmazásokat, így azok ütemezését sem. Megvizsgáljuk, hogy milyen szolgáltatásai és programozható felületei érhetők el a rendszernek, melyekkel mégis alkalmassá tehető valós idejű alkalmazások fejlesztéséhez.

Kulcsszavak: valós idő, folyamat, szál, ütemező, Windows, .NET, C#

1. Bevezetés

A számítógépek sebességének növekedésével párhuzamosan a mindennapjaink is egyre gyorsabbak lesznek. Korábban percek, órák vagy akár csak napok alatt elvégezhető tevékenységekre nem várunk néhány másodpercnél többet. Számítógépek, telefonok és egyéb okos eszközök vesznek minket körül a nap 24 órájában és egy olyan világba nyújtanak belépést, melyben minden mindennel össze van kapcsolva. Ezt a változást nem csak a használt hardverek, de a használt szoftverek is követik.

Egy külső szemlélő ebből mindössze annyit láthat, hogy a szoftvereknek még gyorsabbnak kell lenniük, de ez több ennél. A válaszokat és reakciókat határidőre kell a rendszernek előállítania. Nem elég gyorsan vagy még gyorsabban reagálni, valós időben mérhető időbeli követelményeket támasztunk a szoftverrel szemben.

Több időbeli követelmény fajtát is megfigyelhetünk:

1. Bankkártya használata során elvárjuk, hogy a fizetés folyamata gyorsabb legyen, mintha készpénzzel fizetnénk, ezért a terminál és a bank közötti hálózati kommunikációra, tranzakció végrehajtására csupán néhány másodperce van a rendszernek.

2. Online videojátékok esetében meghatározó tényező a ping, avagy a szerver válaszideje. Minél rosszabb a válaszidő annál rosszabb lesz a szinkronizáció a játékosok között, ugyanazt a jelenséget nem ugyanakkor fogják látni a képernyőkön. A manapság feltörő e-sport világában akár néhány milliszekundumos késés is hátrányba juttathat versenyzőket.

3. Az előző problémák előfordulása jórészt kellemetlenséggel és bosszankodással jár, de egyes szoftverkörnyezetekben a határidők elmulasztása katasztrofális következményekkel járhat. Egy gyógyszergyár gyártósorát vezérlő szoftverének milli-, vagy akár mikroszekundumos pontossággal kell a műszereit vezérelnie. Egy apró hiba és a hibásan előállított termék, gyógyszer akár emberi életeket is veszélybe sodorhat!

2. Valós idejű szoftverfejlesztés

Valós idejű szoftverfejlesztésről akkor beszélhetünk, amikor a fejlesztés alatt álló szoftver követelményrendszerében az időtől függő feltételeket és követelményeket adunk meg. Követelmények,

melyek megszabják, hogy egy eseményre előállított válasz elkészítésére mennyi ideje van a rendszernek. Megadott határidővel vagy időkorláttal kell a szoftvernek feldolgoznia az eseményt és végrehajtania az esemény által kiváltott feladatokat a válasz előállításához. Egy követelmény akkor jó követelmény, ha minden eshetőségre kitér. Mi a teendő, ha ezt a határidőt mégis elmulasztja a rendszer? Két megközelítést alkalmazunk [1].

Laza valós idejű rendszer esetében (soft real-time) a rendszer törekszik a határidők betartására. A késés elkerülendő, de ha mégis bekövetkezik nincs túl nagy probléma, csupán az előállított válasz hasznossága csökken. Ezzel szemben a szigorú valós idejű rendszerek (hard real-time) határidő elmulasztása rendszerhibát vagy egyéb súlyos következményeket okozhat, a késés nem tolerálható.

Adott egy követelmény, melyben szoftverünknek egy eseményre adott határidőn belül kell a választ előállítania, végrehajtania egy kódrészletet. A beérkező eseményhez egy adat tartozik szorosan, a határidő és egy adat lazán, a válasz kódrészlet végrehajtási ideje. Sajnos attól, hogy a szoftverünk elég gyorsan (pl. valamennyi milliszekundum alatt) le tudja futtatni a szükséges kódot, még nem jelenthetjük ki, hogy ez a szoftver minden esetben tartani fogja az előírt határidőt. Ehhez garantálni kellene, hogy a végrehajtáshoz szükséges processzor időt meg is kapja a programunk még a határidő lejárta előtt.

2.1 Ütemezők általánosságban

Egy számítógépen egyszerre sokkal több alkalmazást és folyamatot futtatnak a felhasználók, mint amennyi feldolgozó egység elérhető az adott rendszeren. Az operációs rendszer feladata, hogy ezen folyamatokat úgy ütemezze a processzorokra, hogy a felhasználó számára ne tűnjön föl, hogy igazából ezek az alkalmazások nem egyszerre futnak, hanem felváltva. Az Ütemező az Ütemezési Algoritmus alapján dönti el, hogy melyik folyamat, melyik szála kaphat processzort és mikor kell lemondania arról [2]. Az ütemezésnek pártatlannak kell lennie, minden szál ugyanazon szabályrendszer szerint kell, hogy ütemezésre kerüljön. Az évek során használt rendszerek több Ütemezési Algoritmust is bemutattak, mind más és más környezetben alkalmazható kényelmesen.

Korábbi többfeladatos rendszerek kooperatív ütemezést alkalmaztak, melynek során az ütemező nem dönthet úgy, hogy egy végrehajtás alatt álló folyamattól elveszi a processzort. A folyamatoknak maguktól kell lemondaniuk a processzorról (pl. egy várakozó utasítás, vagy yield parancs használatával). Ennek a Run-to-Completition elvű ütemezőnek előnye, hogy a futás alatt álló folyamat a lehető leggyorsabban végre tudja hajtani feladatát, mert az ütemező nem szakítja meg munkáját váratlan kontextus váltásokkal. Cserébe a rendszer válaszideje és interaktivitása rendkívül gyenge, a futó folyamatok kiéheztetik a többi várakozó folyamatot.

A mai átlag felhasználók által használt rendszerek időosztásos, prioritásos preemptív ütemezőket használnak. Az ütemező nem csak egy processzor szálhoz rendelését teheti meg, a processzort el is veheti a szálaktól (preemption). Egy processzor felszabadulásakor egy várólistában várakozó szál kapja meg a processzort, előre megadott időegységnyi processzor időt kap az ütemezőtől. Az időkorlát lejárásakor az ütemező elveszi a processzort a száltól, a szál visszakerül a várólistába a processzort pedig a következő várakozó szál kapja meg [3]. A felhasználók (és fejlesztők is) prioritásokat rendelhetnek a szálakhoz, ezzel tippet adva az ütemezőnek, hogy mely folyamatainkat szeretnénk nagyobb vagy kisebb sűrűséggel végrehajtás alatt tudni.

Egy interaktív rendszerben ez az algoritmus jól működik, viszont valós idejű környezetben nem garantál semmit! Egy esemény beérkezésekor nem garantált, hogy a folyamatunk ütemezésre kerül, sem az, hogy elegendő processzor időt kap a válasz előállításához szükséges feladatok végrehajtásához. Egy ténylegesen valós idejű rendszer figyelembe veszi az idő szerepét is a folyamatok végrehajtása során! Az operációs rendszer garantálja, hogy egy esemény bekövetkezésekor a rá várakozó

folyamat processzorhoz jut és a szükséges utasítások lefutnak. Ezen funkcionalitást támogató valós idejű rendszerek lehetnek például a beágyazott rendszerek, illetve a Linux rendszernek érhetők el valós idejű bővítményei (pl. Suse Linux Enterprise Real-Time Extension). A Linux kernel 3.14 verziótól kezdve támogatja a SCHED_DEADLINE ütemezést, mely egy EDF (Earliest Deadline First) ütemezés [4]. A folyamatokhoz hozzárendelhető határidőt és szükséges végrehajtási időt figyelembe véve képes a rendszer folyamatokat ütemezni.

A helyzet tovább bonyolódik többmagos/többprocesszoros rendszerek esetében. Futó folyamatok váltása során az ütemezőnek kontextus váltást (context switch) kell végrehajtania. Az eddig futó szállnak az állapotát (utasításszámláló, stack, regiszterek, cache állapotok stb.) el kell tárolni, a következő szál állapotát pedig be kell tölteni. Ez egy drága, időigényes feladat, ha pedig a szál állapotát a különböző processzormagok között is meg kell osztani, akkor még több időt fog igénybe venni. Éppen ezért szeretnénk irányítani, hogy az ütemező, mely processzorokra ütemezheti egyes folyamatainkat. Ezt az opciót processzor affinitásnak nevezik. Valós idejű rendszerek a processzor shield opciót is támogatják, mellyel kijelölhetjük processzoroknak egy halmazát, melyekre csak valós idejű folyamatokat ütemezhetnek, ezzel csökkentve egy beérkező valós idejű folyamat várakozási idejét.

2.2 Windows ütemező

Windows rendszerben megkülönböztetjük a folyamat és szál fogalmát [5]. A folyamat egy kontextus az alkalmazásunknak, menedzseli az erőforrásait, memóriaterületét és nyilvántartja a szálait. A szálak a különböző végrehajtási útjai egy folyamatnak, az ütemező a szálakat ütemezi a processzorra. Két lépésben rendelhetünk prioritást a szálakhoz: először egy prioritási osztályt kell a folyamathoz rendelni (Realtime, High, Above Normal, Normal, Below Normal, Idle), mely minden a folyamatban lévő szálnak a prioritását a megfelelő szintre módosítja, majd az osztályon belül a szálaknak az egymáshoz képesti relatív prioritását is megadhatjuk.

A Windows kliens, illetve szerver változata sem támogatja a valós idejű ütemezéshez szükséges szolgáltatásokat. Habár a prioritási szintek (0-31) között megjelennek a valós idejű prioritások (16-31) ezek nem igazi valós idejű algoritmusokat rejtenek maguk mögött. A rendszer nem garantálja a határidők betartását, csak annyit, hogy a legtöbb szál előtt ütemezésre kerülhet a valós idejű prioritású szálunk. Használata nem veszélymentes: több rendszerszintű szál is a valós idejű szinteken dolgozik, ha egy számításigényes szállal blokkoljuk ezeket a folyamatokat a rendszer instabillá válhat. Egy szál csak akkor lehet valós idejű, ha a szálat tartalmazó folyamat valós idejű prioritási osztályba tartozik, melynek beállításához rendszergazdai jogosultságra van szükség. Valós idejű prioritási osztályban csak valós idejű szálak lehetnek.

Quantum néven hívják az időegységet, mely a folyamatok által felhasználható időszeletet határozza meg. A Quantum hosszát a kernel számítja ki a rendszer indulásakor. Egy szál processzorhoz jutáskor kap valamennyi Quantum-ot, végrehajtási időt. A kapott Quantum-ok száma változó, kliens Windows-on kevesebb Quantum-ot kap egy szál mind szerver Windows-on, illetve az előtérben lévő alkalmazás további Quantum-okat kap.

Válaszidők csökkentésére és elosztott rendszerekben felmerülő erőforrás és blokkolási problémák feloldására a Windows rendszer a Priority Boost funkciót használja. A felhasználó (vagy fejlesztő) által megadott prioritás a szálaknak az alap prioritása (base priority), a Priority Boost felülbírálhatja ezt a prioritást, melyet jelenlegi prioritásnak (current priority) nevezünk. Egyes események hatására a rendszer ideiglenesen megnövelheti egyes szálak prioritását. Például hosszabb várakozás után, egy erőforrás elérhetővé válásakor a várakozó szál prioritás növelést kaphat, így az ütemező nagyobb eséllyel fogja ütemezni és gyorsabban hozzáláthat a munkához. Az előtérben lévő alkalmazás (mellyel a felhasználó aktívan foglalkozik) is automatikusan prioritás növelést kap. A Priority

Boost Quantum növeléssel is járhat, tehát a szálak nagyobb időszeletet kapnak a processzor használatára.

A Windows is biztosít lehetőséget processzor affinitás beállítására, viszont a rendszer rávilágít egy lehetséges hátrányára az affinitásnak: Egy magasabb prioritású szálat nem fog másik processzorra ütemezni egy alacsonyabb prioritású szál affinitása miatt. Éppen ezért, lehetséges, hogy a szálunknak tovább kell várakoznia, mert egy nagyobb prioritású szál dolgozik a szálunknak megfelelő processzoron. Ennek feloldására vezették be az Ideális és Utolsó Processzor fogalmát. Ideális processzorral megadhatjuk, hogy melyik processzoron szeretnénk futtatni a szálunkat és az alábbi módon befolyásolhatjuk vele az ütemező döntését:

1. Amikor a szálunk processzorhoz jut, először az ideális processzort próbálja az ütemező biztosítani a számára.
2. Ha az ideális processzor nem elérhető akkor az előzőleg a szálat futtató processzor elérhetőségét ellenőrzi.
3. Ha az előző processzor sem elérhető akkor a többi processzor közül fog választani egyet a szál végrehajtására.

Nem garantált, hogy a szálunk mindig az ideális processzoron fog dolgozni. Ennek az algoritmusnak a cache miss minimalizálása a célja, a szál állapot processzorok közötti átvitelének minimalizálása.

Az asztali Windows rendszerek nem támogatják a valós idejű funkcionalitást, de a korábbi Windows CE rendszerek közelebb állnak egy teljes valós idejű rendszerhez. Windows CE rendszert kis teljesítményű, főleg beágyazott rendszerekre fejlesztettek, régebbi Windows telefonok és hordozható eszközök használták főleg a rendszert. A valós idejű feladatok végzését a pontos és determinisztikus megszakítások és megszakításokat kezelő szál segíti. Utolsó verziója 2013-ban készült (illetve a Windows 10 Mobile rendszer tartalmazza egyes részeit), utódja a Windows IoT Core, mely a modern kis teljesítményű Internet-of-Things eszközökre juttatja el a Microsoft ökoszisztémát és valós idejű környezetet együtt.

3. Valós idejű .NET API

Az említett ütemező funkcionalitások elérhetők a Microsoft folyamatos fejlesztése alatt álló .NET C# programozási nyelven keresztül is. Mivel egy virtuális gépben futó nyelvről van szó ezért fontos megemlíteni, hogy vannak veszélyei a C# alapú szigorú valós idejű alkalmazások fejlesztésének. A lefordított programot alkotó IL kód (Intermediate Language) futás közben Just-In-Time fordító segítségével fordul natív gépi utasításokra. Ezek a fordítási fázisok futás közben késleltethetik alkalmazásunkat. Szerencsére a modern .NET Core .NET Native előfordítási technológiájával és CoreRT futtatókörnyezetével eliminálható a JIT fordító használata [6]. A másik veszélyforrás a szemétgyűjtő használata. A memória rohamos csökkenése esetén a futtatókörnyezet szemétgyűjtést végez, melynek során a programunk szálait szünetelteti. Egy szerencsétlen pillanatban kiváltódó szemétgyűjtés a határidők elmulasztásával járhat.

A futtatókörnyezet menedzseli az alkalmazásunk szálait is, éppen ezért szükséges megkülönböztetni a virtuális gép felügyelt szálait és az operációs rendszer számára ütemezhető natív végrehajtási szálakat. Egy a Thread osztállyal létrehozott felügyelt szálat egy natív szálnak feleltet meg a futtatókörnyezet. Habár a .NET Core dokumentációja felhívja a figyelmet: a felügyelt szálakat a futtatókörnyezet áthelyezheti másik natív szálakra, jelenleg ezt a funkciót még nem támogatja a virtuális gép. ThreadPool és egyéb beépített párhuzamos könyvtárak használata során a virtuális gép külön-

böző felügyelt szálakat ütemez ugyanazon natív szálakra (ezzel lecsökkentve az operációs rendszer ütemezőjének terhelését), tehát ilyen esetekben a natív szálak tulajdonságainak módosítása nem ajánlott. Natív szálak működésének konfigurálását csak akkor érdemes végezni mikor a szálat mi magunk hoztuk létre a Thread osztály segítségével [7].

```
static void Main(string[] args)
{
     Thread myThread = new Thread(new ThreadStart(ThreadJob));
     myThread.Start();
}

static void ThreadJob()
{
     //Do something...
}
```

A Thread osztálypéldány birtokában az adott szál prioritását állíthatjuk. Ezzel a tulajdonsággal a szálak prioritás osztályon belüli relatív prioritását adhatjuk meg.

```
static void Main(string[] args)
{
     Thread.CurrentThread.Priority = ThreadPriority.Highest;

     Thread myThread = new Thread(new ThreadStart(ThreadJob));
     myThread.Priority = ThreadPriority.BelowNormal;
     myThread.Start();
}
…
```

További konfiguráláshoz hozzá kell férnünk az alkalmazásunk Process osztálypéldányához. Ügyeljünk rá, hogy a Process osztály megvalósítja az IDisposable interfészt ezért használjuk a using blokkot az erőforrások helyes kezelésére! A Process példányon keresztül hozzáférhetünk és módosíthatjuk az alkalmazásunk prioritás osztályát, affinitását és Priority Boost engedélyét [8].

```
static void Main(string[] args)
{
     using (Process process = Process.GetCurrentProcess())
     {
          process.PriorityClass = ProcessPriorityClass.RealTime;
          process.ProcessorAffinity = new IntPtr(0x0002);
          process.PriorityBoostEnabled = true;
     }
}
```

A Process példányon keresztül hozzáférhetünk az alkalmazásunk szálaihoz és bővebb konfigurálásához ProcessThread példányokon keresztül. A Thread és ProcessThread két különböző osztály és

különböző funkcionalitást is biztosítanak. Előbbivel a felügyelt szálunk állapotát és élettartamát kezelhetjük, még utóbbival natív szálak tulajdonságaihoz férhetünk hozzá. Több szálat láthatunk, mint amennyit létrehoztunk, mert a futtatókörnyezet is használ szálakat az alkalmazásunk futtatásához (például a szemétgyűjtő vagy a Just-In-Time fordító végrehajtásához). Ahhoz, hogy az aktuális végrehajtás alatt álló szálunk ProcessThread példányához hozzájussunk szükségünk van a szálunk natív azonosítójára. Ehhez a feladathoz a C# API nem biztosít metódusokat, P/Invoke segítségével kell a kernel32.dll-ben elérhető GetCurrentThreadId API vetületet használnunk. A visszatérő érték az aktuális végrehajtási szál natív azonosítója, melyet felhasználva hozzájuthatunk a szál ProcessThread példányához.

```
public static class ThreadExtension
{
     public static ProcessThread GetProcessThread(this Thread thread)
      {
            uint threadID = GetCurrentThreadId();
            using (Process process = Process.GetCurrentProcess())
                  return process.Threads.OfType<ProcessThread>()
                        .FirstOrDefault(pt => pt.Id == threadID);
      }

      [DllImport("kernel32.dll")]
      static extern uint GetCurrentThreadId();
}
…
static void Main(string[] args)
{
     using (ProcessThread pt = Thread.CurrentThread.GetProcessThread())
            pt.IdealProcessor = 7;
}
```

A fenti API-k elérhetők .NET Framework és .NET Core keretrendszerek használatával is. .NET Core-t használva még a .NET Native előfordítási technológiát alkalmazva is elérhetjük a szükséges metódusokat. A Windows 10 rendszerben támogatott Universal Windows Platform biztonsági okokból az API-t csak minimális mértékben támogatja, és legtöbb esetben a támogatott részek használata a Microsoft Store áruházi szabályzatába ütközik.

4. Összegzés

Manapság a valós idejű alkalmazások fejlesztésének igénye már nem csak az ipari, hanem az átlag felhasználói környezetben is jelentős. Az átlag felhasználói operációs rendszerek közül a Linux egyes változatai támogatnak valós idejű funkcionalitást. Azt vizsgáltuk, hogy Windows rendszerben, milyen ütemezési opciók és konfigurálási lehetőségek elérhetők, melyek mégis alkalmassá tehetik a Windows rendszert valós idejű szoftverfejlesztéshez.

A .NET C# programozási nyelv hatalmas fejlődéseken ment át az elmúlt években. Az elért sebesség növekedésnek és technológiai újításoknak köszönhetően sokkal magabiztosabban használhat-

juk valós idejű környezetben. Megfelelő könyvtárakat használva a rendszer szintű szálak működésének konfigurálását is elvégezhetjük, ezzel tanácsokat adva az ütemezőnek az alkalmazásunk helyes ütemezéséhez.

Köszönetnyilvánítás

EFOP-3.6.3-VEKOP-16-2017-00001: Tehetséggondozás és kutatói utánpótlás fejlesztése autonóm járműirányítási technológiák területén – A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Hivatkozások

[1] Dr. Illés Zoltán, Heizlerné Bakonyi Viktória, Illés Zoltán: Valós időben, valós világban: INFODIDACT 2015, Zamárdi (2015), ISBN: 978-963-12-3892-1

[2] Dr. Illés Zoltán: Operációs rendszerek 4. előadás, http://os.inf.elte.hu/index.php?link=ea (utoljára megtekintve: 2018.11.06.)

[3] Dr. Illés Zoltán, Szabó Dávid: Operációs rendszerek 5. előadás http://os.inf.elte.hu/index.php?link=ea (utoljára megtekintve: 2018.11.06.)

[4] Yoshitake Kobayashi: Deadline Miss Detection with SCHED_DEADLINE: ELC 2013. Embedded Linux Conference, San Francisco, California, 2013.02.20-22.

[5] Mark Russinovich, David A. Solomon, Alex Ionescu: Windows Internals, Part 1, 6th Edition, Microsoft Press, 2012, [359-485], ISBN: 978-0-7356-4873-9

[6] Szabó Dávid: C# Multi-Platform Környezetben, MSc Diplomamunka, 2018, https://edit.elte.hu/xmlui/bitstream/handle/10831/38929/Diplomamunka.pdf (utoljára megtekintve: 2018.11.06.)

[7] Bart de Smet: C# 4.0 Unleashed, Pearson Education, 2011, [1431-1452], ISBN: 978-0-672-33079-7

[8] Andrew Troelsen, Philip Japikse: Pro C# 7: With .NET and .NET Core, Apress, 2017, [631-641], ISBN: 978-1-4842-3017-6

Mit (nem)tanultunk informatikából a középiskolában

Szabó Tibor[1], Pšenáková Ildikó[2], Pribilová Katarína[3]

[1]tszabo@ukf.sk, [2]ildiko.psenakova@gmail.com,
[3]katarina.pribilova@truni.sk

[1]Univerzita Konštantína Filozofa v Nitre, [2,3]Trnavská univerzita v Trnave

Absztrakt. Szlovákiában az informatika oktatása már az általános iskolában elkezdődik és választható érettségi tantárgyként fejeződik be. Némely középiskola ECDL tanfolyamokat kínál diákjainak, melyek révén az itt megszerzett tudásról bizonyítványt is szereznek. A középiskola jellegétől függően, illetve attól, hogy a diák milyen irányban folytatja majd egyetemi tanulmányait, gyakran még egy programozási nyelv elsajátítása is része az iskolai tananyagnak. Az előzőekre építve, az egyetemeken természetes alapelvárás lehetne, hogy a diák tudja kezelni a szövegszerkesztőt, táblázatkezelőt, tudjon prezentációt készíteni és ne okozzon neki gondot az egyszerű adatbáziskezelés. Sajnos, a valóság egészen más! Sok első évfolyamos egyetemi hallgatónak az informatikai tudása a szociális hálók használatára korlátozódik. Feltételezéseink alapján ennek oka a közoktatásban kereshető, ahol a diákok nem kapják meg a megfelelő szintű és tartalmú oktatást. Az általunk készített felméréssel erre keressük a válaszokat.

Kulcsszavak: informatika, oktatás, tartalom, felmérés

1. Bevezetés

A felsőoktatási intézmények szinte bármelyik tanulmányi programját tekintve a diákoknak abszolválniuk kell legalább egy vagy két informatika órát az általános alapozó tantárgyakon belül. Az említett tanórák tartalma legtöbbször magába foglalja a szövegszerkesztést, táblázatkezelést, prezentációkészítést stb., tehát ténylegesen az alapokat. Az iskolarendszerben az MS Office használata az elfogadott, és már az általános iskolában elkezdik oktatni, például a szövegszerkesztőt már az alsótagozaton. Több esetben a felsőoktatási intézménybe jelentkező diákok már rendelkeznek ECDL bizonyítvánnyal is, mivel néhány középiskolában lehetőségük nyílik ezek megszerzésére. Így az említett témakörök már ismeretesek kellenének, hogy legyenek a diákok számára a középiskolás és részben az általános iskolás tanulmányaikból. Azonban a tapasztalatunk azt mutatja, hogy ez gyakran nem így van, sőt néha az ECDL bizonyítvány birtokában lévő diákok sem állnak a helyzet magaslatán. Napjainkban már a Z generációs diákokkal dolgozunk, akiknek ez a modern világ már természetes környezetük, de úgy látjuk, hogy ez sem jelent elegendő előnyt a számukra. Ez a tény sarkallt minket arra, hogy megpróbáljunk válaszokat keresni a probléma okára, illetve felmérni azt, hogy milyen informatikai ismeretekkel rendelkeznek a diákok a középfokú tanulmányaik befejezése után.

2. Az informatika oktatásának történelmi áttekintése

Az informatika, mint tudomány még fiatalnak számít. Ennek ellenére I. Kalaš szerint már három koncepción is átesett az informatika oktatása iskoláinkban:

- *algoritmizálás és programozás* – 70-es, 80-as évek és a 90-es évek eleje. Lényegében a kezdetek kezdetéről beszélünk, amikor még csupán néhány gimnáziumban került be az oktatásba a programozás tanítása, majd ezekhez folyamatosan csatlakoztak a többiek.

Ezekben az időkben még nagy problémát jelentett az iskola hiányos felszereltsége IKT eszközökkel, a tanulók csak nagyon nehezen juthattak hozzá, hogy számítógépen dolgozzanak. Ezért a programozást Pascal programozási nyelven gyakran csak papíron oktatták. 1988 és 1991 az első tankönyvek megjelenése, melyek a következők voltak: Informatika és számítástechnika (Informatika a výpočtová technika), Algoritmusok és Informatika és Számítástechnika (Algoritmy a Informatika a Výpočtová technika), Programozás Pascal nyelven (Programovanie v jazyku Pascal). Az említett tankönyvek valójában egyetemi jegyzetekhez hasonlítottak.

- *felhasználói informatika* – 90-es évek. Az oktatás már kiterjedt a számítógép működési elvének megismertetésére, az operációs rendszer, ill. alkalmazói programok használatára, például: DOS, Windows, szövegszerkesztők (T602), Norton Commander, AutoCAD. Érdekességként megemlíthető, hogy ezekben az években nem jelentek meg új tankönyvek, holott az oktatás tartalmát tekintve a kínálat bővült, ezért gyakran ezek hiányát bizonyos mértékben a programok dokumentációja pótolta.
- *az informatika tanításának modernizálása* – 90-es évek végétől kezdődött. Ekkor jöttek létre az új tantervek, új tankönyvek, munkafüzetek és megjelent az érettségi lehetősége. Az oktatás öt alaptémakört tűzött ki célul:
    - a minket körülvevő információk,
    - számítógépes rendszerek,
    - algoritmusok és algoritmizálás,
    - az informatika alkalmazásának területei,
    - információs társadalom.

  Az informatika az oktatásban arra törekszik, hogy ne csak a számítógép használatra korlátozódjon, hanem egy komplexebb megértésre összpontosítson. Eközben a középiskolákban már jelentős változás történt a számítógépekkel való felszereltséget illetően, így a programozás sem rekedt meg kizárólag csak elméleti szinten. Főként Pascal / Delphi (a Delphi ma már nem programozási nyelv, csak programozási környezet az Object Pascalt alkalmazva) nyelveken történt az oktatás. [1, 2]

Az informatika oktatása Szlovákiában, mint a legtöbb fejlett országban, már az általános iskola alsó tagozatán megkezdődik. Az alsó tagozat 2 órát, a felső 4 órát ír elő kötelezően. Bár valójában már az óvodai nevelésben is megjelenik az algoritmikus gondolkodás fejlesztésének feladata, illetve bevezetés a digitális játékok használatába.

Szlovákiában az oktatás tartalmát minden iskolai szinten az állami oktatási program határozza meg. A programok az Állami pedagógiai intézet weboldalán érhetők el: http://www.statpedu.sk/. Az Intézet közvetlen a Szlovák Köztársaság Oktatási, tudományos, kutatási és sportminisztériuma által irányított szervezet alá tartozik. [3] Esetünkben az ISCED 3 (International Standard Classification of Education), pontosabban az ISCED 3A és ISCED 3B nemzetközi klasszifikációnak megfelelő állami oktatási programok az érdekesek.

3. ISCED3

Az Informatika tantárgy a Matematika és munka az információkkal (Matematika a práca s informáciami) műveltségi területbe van besorolva. [3] A jelenlegi oktatási program 2015 szeptemberétől van érvényben, azt ezt megelőző 2011 szeptemberétől volt érvényben, mindössze négy év kellett a változtatásra. A négy(öt) és a nyolcéves gimnáziumok számára (az utóbbinál az 5-8 évfolyamokat értve) a kerettanterv Szlovákiában 3 kötelező órát ír elő, ez vonatkozik a Szlovákiában élő nemzetiségi kisebbségek nyelvén oktatott iskolákra is. Az óraszám meglehetősen alacsony, ezért az iskoláknak jogában áll még további választható órákat is beiktatniuk a helyi tantervükbe, de legfeljebb 20 órát.

Az informatika területét érintő megújult oktatási program a következő területeket célozta meg: informatikai eszközök használata, kommunikáció és együttműködés, algoritmikus problémamegoldás, szoftver és hardver, információs társadalom. A tantárgy célja, hogy a tanulók:

- elsajátítsák az ismert alkalmazások használatát, hatékonyan keressék az információkat külső adathordozókon, ill. a hálózaton;
- képesek legyenek a hálózaton keresztüli kommunikációra;
- fejlesszék együttműködési és kommunikációs képességeiket;
- elsajátítsák a kutatási munkához szükséges készségeket;
- fejlesszék a személyiségüket, kreativitásukat, logikus gondolkodásukat, felelősségérzésüket, önkritikájukat és törekedjenek az önképzésre;
- tiszteletben tartsák a szellemi tulajdont, a szerzőijogokat az információs termékek, szoftverek és alkalmazások tekintetében.

Amennyiben szeretnénk összehasonlítani az említett két utolsó oktatási programot, akkor tartalmi szempontból lényegi különbségekbe nem igazán ütközünk, kivéve a robot készletek beiktatását az oktatásba. A különbség leginkább talán abban tűnik ki, hogy az előzőhöz képest az újabb program jóval részletesebben van kidolgozva.

Elszórtan történhetnek apró kiegészítések az állami oktatási programban, például egy a 2016-os évben kiadott módszertani ajánlás, ami ugyan nem kötelez minket konkrét programozási nyelv tanítására, de a Pascal programozási nyelv helyett már ajánlja például a Python nyelvet is.

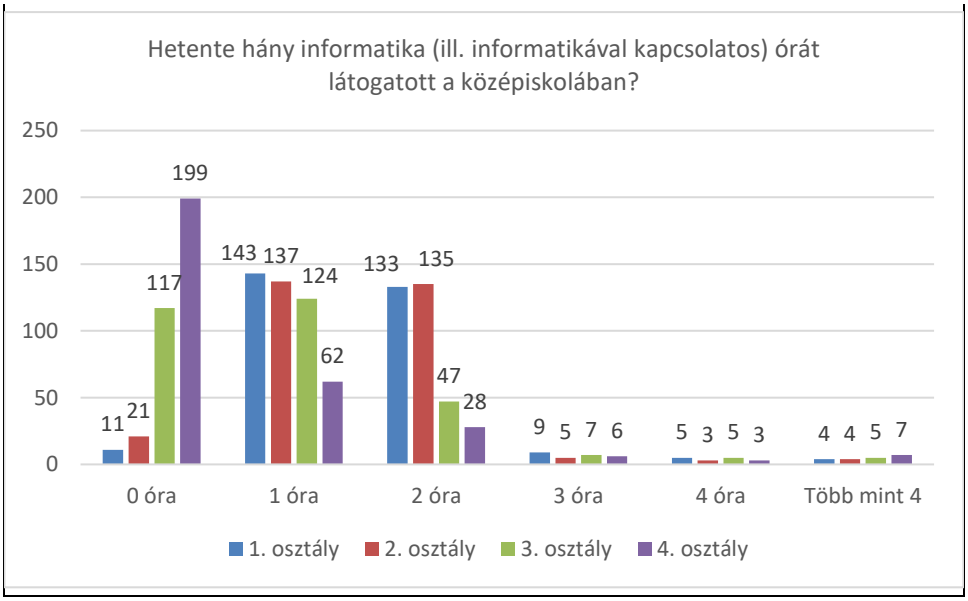
4. Felmérés

A felmérésben első évfolyamos egyetemi diákok vettek részt, számszerűen 305. Abban bízva, hogy rövidebb kérdőív, zárt kérdésekből összeállítva nagyobb arányú kitöltést eredményez számunkra, végül csak 7 kérdést tettünk fel. A rövid kérdőívnek a másik oka az volt, hogy nem szerettünk volna túl sok időt szánni annak kitöltésére a tanóra különben is rövid idejéből. A kérdések teljes mértékben összhangban vannak a 2011. szeptemberében érvénybe lépett oktatási programmal, ugyanis a válaszadók korosztályát tekintve még az ennek megfelelő oktatásban részesültek. A válaszadók összetételét tekintve tanárszakosokból, óvópedagógusokból és regionális idegenforgalom szakosokból tevődik össze.

A kutatás a Nyitrai Konstantin Filozófus Egyetem Közép-európai Tanulmányok Karának és a Nagyszombati Egyetem Pedagógia Karának diákjai között zajlott. A megkérdezettek elenyésző része végzett nyolcéves gimnáziumot csak 2,62 %, a többi 97,38 % pedig szakközépiskolát (Szlovákiában a szakközépiskola érettségivel végződik és lényegében a magyarországi szakgimnáziumoknak felel meg) vagy négyéves gimnáziumot végzett (lásd az 1. táblázatot). Az 1. ábrából leolvasható, hogy a megkérdezettek a középiskolai tanulmányaik folyamán évente (csak az utolsó négy évet tekintve) mennyi informatika, ill. informatikával kapcsolatos órát látogattak. Azonnal látható, hogy a 4. évfolyamban a diákok 2/3-ának egyáltalán nem volt informatika órája, ellenben az 1. és 2. évfolyamokban szinte mindenkinek volt legalább egy órája. Érdekes adatokhoz jutunk, ha vesszük az átlagóraszámot az egyes iskolatípusok szerint, a szakközépiskolák átlagóraszáma 4,56 óra, négyéves gimnáziumoknál 4,61 óra és a nyolcéves gimnáziumok esetében 4,56 óra az utolsó négy évet figyelembe véve. Az utóbbi csoportot figyelmen kívül hagyhatjuk a kis létszám miatt, de a két nagy csoport azt jelzi, hogy a minimálisan előírt három órát megközelítőleg másfél órával múlták felül. Meg kell még jegyeznünk azt, hogy a több mint 4 óra válaszok esetében öttel számoltunk, de ez szinte jelentéktelen eltérést okozhat csak az eredményben a kis mennyiség miatt. A továbbiakban 5 különböző témakör szempontjából tettük fel kérdéseinket, ahogy azt az alfejezetek címei is jelzik.

Iskola típusa	Válaszok száma	Válaszok száma (%)
Szakközépiskola	155	50,82
Négyéves gimnázium	142	46,56
Nyolcéves gimnázium	8	2,62

1. táblázat: A válaszadók összetétele iskolai végzettség szerint



1. ábra: Informatika órák száma az utolsó négy év tekintetében.

4.1. Alkalmazói ismeretek

Kíváncsiak voltunk, hogy milyen alkalmazásokat oktattak a középiskolákban, az eredmények megtekinthetők a 2. táblázatban. Meglehetősen érdekes, hogy a diákok majd 20 %-a nem tanult szövegszerkesztést, körülbelül 25 % nem tanult táblázatkezelést, és megközelítőleg 15 % nem tanult prezentációkészítést, holott ezek ismerete teljesen alapkövetelménynek tekinthető, és alapelvárásnak számít a munkáltatók részéről is. Megközelítőleg ezt az alapkövetelményt, azaz mindhárom alkalmazástípust a diákok 62 %-a tanulta. Legalább öt alkalmazástípust a 2. táblázatban felsoroltak közül megközelítőleg 16 % ismer. Az egyéb alkalmazásoknál említésre került számviteli szoftver, hotel információs rendszer és videó szerkesztésre alkalmas szoftver ismerete egyaránt.

Alkalmazás típusa	Válaszok száma	Válaszok száma (%)
Szövegszerkesztő	246	80,66
Táblázatkezelő	227	74,43
Prezentációkészítő	259	84,92
Adatbáziskezelő	46	15,08
Grafikus szerkesztő	101	33,11
Weblapszerkesztő	106	34,75
Egyiket sem	6	1,97
Egyéb	5	1,64

2. táblázat: Az egyes alkalmazástípusok oktatása

4.2. Programozási nyelvek ismerete

2. ábra: Programozási nyelvek oktatása a középiskolákon.

A következő kérdésünkben kíváncsiak voltunk arra, hogy milyen programozási nyelveket tanultak a középiskolában, melynek eredménye a 2. ábrán található. Az egyszerűség kedvéért döntöttünk úgy, hogy csak a programozási nyelvek ismeretére hagyatkozunk és magára az algoritmizálásra jelenleg nem. Talán ez a legérdekesebb kérdés, hiszen a fentiekben is már említettük, hogy ez a témakör került be elsők között a középiskolai informatika oktatásába. Jól látni, hogy több mint a diákok fele

(56,39 %) tanult legalább egy programozási nyelvet. További érdekesség, hogy a 80 diákból, akik gyermek programozási nyelvet (Baltík, Imagine, Scratch...) tanultak, 63 % már más programozási nyelvet nem tanult. Az ábrán feltüntettük a HTML jelölőnyelvet is, ugyanis néhányan feltüntettek az egyéb kategóriában. Többen az egyébnél feltüntették a Delphi-t is, ezt manuálisan korrigáltuk és hozzászámoltuk az Pascal / Object Pascal kategóriához. Egyértelmű, hogy ezeknek a nyelveknek még mindig jelentős szerepe van a programozás oktatásában.

4.3. Hardver és szoftver ismerete

A 2. táblázat azt mutatja, hogy a diákok milyen hardver és szoftver ismeretekkel kapcsolatos témakörökkel találkoztak a középfokú tanulmányaik folyamán. Az eredmények alapján 12,13 % az említett témakörök egyikével sem találkozott. Majdnem 18,03 % hallott az összes témakörről, négyről 16,72 %, három, ill. két témakörről egyaránt 19,02 % és csupán egy témakörrel 15,08 % találkozott.

Témakörök	Válaszok száma	Válaszok száma (%)
Számítógép működési elve	158	51,88
Számítógép felépítése (mikroprocesszor, alaplap, operációs tár, merevlemez, külső és belső tárolóeszközök, …)	171	56,07
Ki és bemeneti berendezések (billentyűzet, nyomtató, monitor...)	216	70,82
Operációs rendszer használata	159	52,13
Számítógép-hálózatok	111	36,39
Egyikről sem	34	11,15

2. táblázat: Hardver és szoftver ismertével kapcsolatos témakörök

4.4. Számítógépes biztonság

Fogalmak	Válaszok száma	Válaszok száma (%)
Szerzői jog	102	33,44
Malware	69	22,62
Phishing	53	17,38
Kiberbűnözés	76	24,92
Észlelés, megelőzés	45	14,75
Egyikről sem	135	44,26

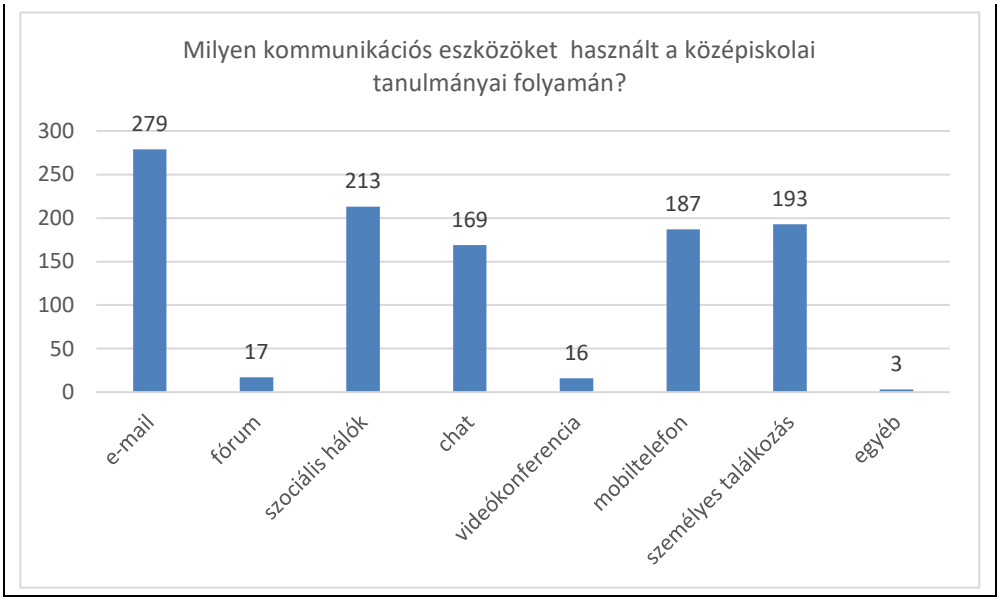
3. táblázat: Számítógép biztonsággal kapcsolatos fogalmak

Manapság az egyik legégetőbb problémát feszegettük ezzel a kérdéssel, a számítógépes biztonság oktatási kérdéseit. Kíváncsiak voltunk, hogy az általunk felsorolt öt fogalomból melyekről hallottak már a diákok a középiskolai tanulmányaik folyamán. A 3. táblázat meglehetősen szomorú helyzetképet mutat, hiszen a tanulók 44 %-a egyik fogalommal sem találkozott, ez meglehetősen aggasztó a jövőre nézve. A megkérdezettek majdnem 5 %-a mondta azt, hogy már hallotta mind a 4 fogalmat, 8,52 % három fogalmat felismert, két fogalmat 14,43 % és egy fogalmat már a diákok harmada hallott. Ez esetben a probléma gyökerei a tanárképzésre egyértelműen visszavezethetőek. „*A rosszindulatú szoftverek elleni védelem és a számítógép biztonságos használatának oktatására a leendő pedagó-*

gusok számára nincs az egyetemeinken a jelenleg akkreditált tantervekben kijelölve külön tantárgy. Az sincs megszabva, hogy melyik tantárgy tartalmába és mennyi óraszámban kellene, ill. lehetne beilleszteni az oktatásba." [7]

4.5. Kommunikációs szokások

A vizsgált csoportunk tagjai mindannyian a Z generációba sorolhatók. Köztudott, hogy a modern kommunikációs eszközöket előszeretettel használják, sőt sok esetben túlzásba is viszik. Kíváncsiak voltunk, hogy mely kommunikációs eszközöket, ill. a kommunikáció mely formáját használták középiskolai tanulmányaik folyamán a tanulást elősegítendően. Úgy gondoltuk, hogy a szociális hálók (69,84 %), ill. a chat (55,41 %) lesz a leggyakrabban használt eszköz, de ehelyett azt látjuk a válaszokból, hogy az elektronikus levél az, melyet a diákok több mint 91% használt (lásd a 3. ábrán).



3. ábra: Kommunikációs eszközök használata.

4.6. További vizsgálatok

Megvizsgáltuk az órák száma és a témakörök száma közti összefüggést. Ehhez szükséges volt tanulónként elkészítenünk az órák számának és a témakörök számának összegeit, így eredményként előállítottunk két összetartozó mintát. Elsősorban korrelációt számoltunk, a korrelációs együtthatóra kapott érték 0,3372228, mely Cohen szerint gyenge lineáris összefüggést mutat a két minta között. Az eredmény korántsem biztató, így további vizsgálatot végeztünk.

A Shapiro-Wilk teszt alkalmazása segítségével elvetjük, hogy a minták eloszlása normális lenne, mindkét esetben a *p*-értékek sokkal kisebbek 0,05-nél. A további vizsgálathoz a nem parametrikus Mann-Whitney (-Wilcoxon)-féle U-próbát kellett használnunk, melynél a *p*-értéke $2{,}597916.10^{-30}$ lett. A kapott érték kisebb mint 0,05, így elmondhatjuk, hogy statisztikailag nincs szignifikánsan kimutatható összefüggés a két minta között. Nem erre az eredményre számítottunk, de nyilván a válaszok alapján nem tudjuk eldönteni, hogy az egyes témákkal mennyire mélyrehatóan, azaz mennyi órán át foglalkoztak az iskolában - az ilyen szinten pontosított válaszok a diákok részéről elvárhatatlanok lennének.

5. Összegzés

A felmérés egyértelműen mutatja, hogy az informatika terén a diákok meglehetősen nagy tudáshiánnyal érkeznek a felsőoktatásba. Az alkalmazói ismeretek, a hardver és szoftver ismerete és a számítógép biztonsággal kapcsolatos kérdések ezt a hiányt egyértelműen alátámasztják. Holott „*ma már szinte minden munkakör betöltéséhez szükség van alkalmazói ismeretekre*." [8] Az algoritmizálást és programozást nem tekintjük alapelvárásnak a nem informatika szakosok esetében.

Az informatikaoktatás történelmi áttekintésében láttuk, hogy a kezdetek nem voltak egyszerűek, de kijelenthetjük, hogy ez jelenünkben sem az. Nemcsak informatikus szakemberekből van hiány a munkaerőpiacon, hanem informatika tanárokból is, az utóbbi esetben nem csak az érdeklődés hiánya következtében. V. Stoffa szerint: „*Be kell vallanunk azt is, hogy Szlovákiában, az utóbbi években egyre több informatika tanárnak készülő hallgató nem tervez tanári pályán maradni.*" [9] Amennyiben alacsonyabb az informatika tanári szak iránti érdeklődés és a diplomát szerző tanárok egy része ráadásul nem is marad tanári pályán, akkor végül ki is tanítja az informatikát?

Szomorú tény, hogy gyakran a tanári szakot csak nagy nehézségekkel elvégző tanárok maradnak a tanári pályán, és emellett még informatikatanár hiányról is beszélünk, így néha a nem informatika szakos tanárokra is hárul a feladat, hogy informatikát oktassanak. Az említett problémák az oktatás minőségét nem emelik. Talán az ilyen esetekben hallhatunk néha olyan válaszokat a diákoktól, arra a kérdésre, hogy „Mit tanultak informatika órán?", hogy „csak játszottunk".

Természetesen további tényezők is befolyásolják a diákok tudásszintjét. A kerettanterv csak három tanórát ír elő kötelezően, vajon elégséges ez az idő, amennyiben az adott oktatási intézmény nem dönt úgy, hogy további órákat is beiktat? Mint kiderült, átlagban megközelítőleg 4,5 óra informatikát tanultak a felmérésben szerepelt diákok, vajon az a mennyiség már elégséges? További tényező, hogy a számítógép biztonság témaköre már a tanárképzéstől kezdődően gondokkal indul. Gyakran viszont a legbizonytalanabb tényező maga a diák.

Bízunk benne, hogy a felsorolt problémák folyamatosan megoldásra találnak, melyek majd elősegítik a diákok digitális írástudásának jövőbeli pozitív javulását is.

Köszönetnyilvánítás

A tanulmány megjelenését a KEGA 015TTU-4/2018: „Interaktivita v elektronických didaktických aplikáciách." (Interaktivitás az elektronikus didaktikai alkalmazásokban) című projekt támogatta.

Irodalom

1. I. Kalaš: *Čo ponúkajú IKT iným predmetom (3. časť): Informatika a informatizácia.* Zborník konferencie Infovek. (2001)
2. J. Šišková: *Metodika prípravy na maturitu z informatiky*. Disszertációs munka, UK v Bratislave. (2012) https://people.ksp.sk/~julka/doktorantura/dizertacka.pdf (utoljára megtekintve: 2018.10.24.)
3. I. Pšenáková, T. Szabó: *Pedagógusok versus hackerek, vírusok és hasonló férgek*, INFODIDACT 2017 – 10. Informatika Szakmódszertani Konferencia, Zamárdi (2017).
4. *Kerettantervek - ISCED3* (2011) http://www.statpedu.sk/sk/svp/statny-vzdelavaci-program/statny-vzdelavaci-program-gymnazia/ramcove-ucebne-plany (utoljára megtekintve: 2018.10.28.)
5. *Kerettantervek - ISCED3* (2015) http://www.statpedu.sk/sk/svp/inovovany-statny-vzdelavaci-program/inovovany-svp-gymnazia-so-stvorrocnym-patrocnym-vzdelavacim-programom/ramcovy-ucebny-plan/ (utoljára megtekintve: 2018.10.28.)

5. *Állami oktatási program / Informatika – négy és öt éves gimnáziumok számára(ISCED3)* (2011) http://www.statpedu.sk/files/articles/dokumenty/statny-vzdelavaci-program/informatika_isced3a.pdf (utoljára megtekintve: 2018.10.28.)
6. *Állami oktatási program / Informatika – négy és öt éves gimnáziumok számára(ISCED3)* (2015) http://www.statpedu.sk/files/articles/dokumenty/inovovany-statny-vzdelavaci-program/informatika_g_4_5_r.pdf (utoljára megtekintve: 2018.10.28.)
7. I. Pšenáková: *Számítógépes biztonság oktatása a leendő tanároknak*. In: A magyar tannyelvű tanítóképző kar 2017- es tudományos konferenciáinak tanulmánygyűjteménye. Subotica: University of Novi Sad, Hungarian Language Teacher Training Faculty, (2017)
8. V. Heizlerné Bakonyi, Z. Illés, L. Menyhárt: *Szempontok az informatika oktatási tartalmak kialakításához*. Info-Didact'12 Konferencia, (2012).
9. V. Stoffová: *Az informatika tanításának elmélete és gyakorlata Szlovákiában*, INFODIDACT 2016 – 9. Informatika Szakmódszertani Konferencia, Zamárdi (2016).

Informatika (verseny)feladatok matematikája

Szabó Zsanett

szabo.zsanett@inf.elte.hu
ELTE IK

Absztrakt. A programozás feladatok egy része matematika. A kérdés az, hogy mekkora ez a rész és milyen szintű, milyen jellegű matematikai ismeretek szükségesek egyes programozás feladatok megoldásához? Ebben a cikkben áttekintem, hogy az elmúlt évek programozás versenyein a „papíros" fordulók feladatai milyen jellegű és milyen szintű matematikai ismereteket vártak el a versenyző diákoktól. Mindezt összehasonlítom a nemzetközi programozási versenyeken elvárt ismeretekkel, illetve a matematika órákon tanultakkal.

Kulcsszavak: programozás versenyek, versenyfeladatok, algoritmikus gondolkodás, matematika, gráfok, rekurzió, algoritmusok értelmezése

1. Bevezetés

A programozás versenyeken alapvetően a programozásé a főszerep annak rendje és módja szerint. Az első fordulók azonban több verseny esetében is úgynevezett papíros fordulók, ahol a feladatokat számítógép nélkül, papíron kell megoldaniuk a versenyzőknek.

A papíros fordulók létrejöttének oka többek között az, hogy így több tanuló kap lehetőséget a versenyen való részvételre. A versenyt szervező iskolák számára könnyebbség, hogy nem szükséges technikai háttér az első fordulóhoz, a tanulóknak pedig könnyebbség, hogy nem kell attól tartaniuk, hogy egy kódolási hiba miatt nem lesz értékelhető a munkájuk.

A programozás versenyek papíros fordulóinak sajátossága, hogy a feladatok megoldásához nincs szükség konkrét programozási, kódolási tudásra. Ezek a feladatok leginkább az algoritmikus gondolkodást mérik. Megoldásukhoz azonban (ha nem is konkrétan kimondott, nevesített formában) szükség van bizonyos matematikai ismeretekre. A feladatok egy része ebből adódóan előkerülhetne akár matematika órán vagy szakkörön is.

2. Programozás versenyek

Az általános- és középiskolás tanulók többféle versenyen mérhetik össze a programozással kapcsolatos tudásukat. Ezek közül négy, országos szinten minden tanuló számára elérhető versennyel, illetve azok feladataival foglalkoztam.

Az egyik ilyen verseny a HÓDítsd meg a biteket! informatika verseny. Ez a 4-12. osztályos tanulóknak szóló verseny a nemzetközi BEBRAS-kezdeményezés, vagyis a Nemzetközi informatikai és számítógép-készség verseny (*International Contest on Informatics and Computer Fluency*) magyar partnere. A Dr. Valentina Dagiene litván professzor által létrehozott kezdeményezés célja, hogy megmutassa az informatika sokszínűségét, felhasználási lehetőségeit és területeit, hogy felkeltse a tanulók érdeklődését az informatika iránt, és feloldja az informatikával kapcsolatos félelmeket, negatív érzéseket. Mindezek mellett célja, hogy segítse az oktatásban résztvevőket minél színesebb, érdekesebb, motiválóbb informatikai gondolkodást segítő feladatokkal, lehetőségekkel [1]. Ezen az egyfordulós versenyen a tanulóknak számítógépen kell megválaszolniuk a feladatokat, amelyeknél négy válaszlehetőség közül kell kiválasztaniuk a helyeset. A feladatok nem kódolással, hanem algoritmikus és logikus gondolkodással kapcsolatosak.

Nem ritka, hogy ezen a versenyen egy-egy iskola vagy osztály minden tanulója indul, ez egy tömegversenynek mondható. Az e-HÓD verseny alkalmas arra, hogy az első pozitív élményt, motivációt megadja a diákok számára, ami után érdemes továbblépniük az érdeklődő tanulónak a többi programozással kapcsolatos verseny, például a Logo vagy a Nemes Tihamér versenyek irányába.

Egy másik verseny a Logo Országos Számítástechnikai Tanulmányi Verseny, ahol a 3-12. évfolyamos (korábban 3-10. évfolyamos) tanulók mérhetik össze tudásukat három, illetve az alsó tagozaton két fordulóban. Ezen a versenyen az első forduló két részből áll: egy számítógép nélküli és egy számítógépes feladatsorból. A számítógép nélküli feladatok egy részének megoldásához szükség van a Logo programozási nyelv ismeretére, de a feladatok többsége anélkül is megoldható. A számítógép nélküli feladatsorban a versenyzők 3-4 kisebb feladatot (algoritmus- vagy programrészletet, működési vázlatot) kapnak, és olyan kérdésekre kell válaszolniuk, mint pl. Mit rajzol? Milyen feltételek mellett működik? Mi hiányzik belőle? Mire használjuk a paramétereket? Megoldja-e a kitűzött feladatot? [2].

A harmadik verseny, melynek feladatait vizsgáltam, az a Nemes Tihamér Nemzetközi Informatika Tanulmányi Verseny programozás kategóriája [3]. Ezen a versenyen 5-10. évfolyamos tanulók vehetnek részt. A verseny három fordulója közül az elsőben itt is számítógép nélkül kell megoldaniuk a feladatokat a versenyzőknek, illetve az 5-8. osztályosok az utolsó feladatnál választhatnak, hogy egy számítógépes vagy egy számítógép nélküli feladatot oldanak meg. A számítógép nélküli feladat általában egy algoritmus értelmezése és hibakeresés, a számítógépes feladatnál pedig kódolniuk kell. Ennél a versenynél a számítógép nélküli feladatok egy része nem igényel különösebb programozási ismereteket, egy részükhöz viszont szükség van arra, hogy a versenyen résztvevő tanulók értelmezni tudjanak bizonyos algoritmusokat.

A negyedik programozáshoz kapcsolódó verseny az Országos Középiskolai Tanulmányi Verseny informatika tantárgyának II. kategóriája [4]. Itt a 11-12. évfolyamos tanulók mérhetik össze programozói tudásukat szintén három fordulóban. Ahogyan a Nemes Tihamér Informatika Tanulmányi Verseny programozás kategóriájában, úgy az első fordulóban itt is számítógép nélküli feladatmegoldást várnak el a tanulóktól. A feladatok összetétele ennél a versenynél is hasonló az ott leírtakhoz. A feladatok egy része megoldható konkrét programozási tudás nélkül, algoritmikus gondolkodással, míg a feladatok másik részének megoldásához szükség van arra, hogy a versenyzők értelmezni tudjanak egy-egy algoritmust, és meg tudják mondani, hogy azok bizonyos bemenő adatokra milyen eredményt adnak.

3. Papíros fordulók

A számítógép nélküli, vagyis papíros fordulók a tanulók algoritmikus gondolkodását mérik, nem a kódolási képességüket. A tanulók egy része kimondottan örül annak, hogy bár programozás versenyen indulhatnak, mégsem kell ténylegesen programkódot írniuk. Nem kell attól tartaniuk, hogy egy hiba miatt nem fognak tudni működő programkódot írni, illetve attól, hogy nem fognak tudni értelmezni egy-egy hibaüzenetet vagy attól, hogy bár lefordul a programjuk, de az a futás során végtelen ciklusba kerül.

A tanulók (és felkészítő tanáraik) szempontjából a másik érv a papíros fordulók létjogosultsága mellett az, hogy a versenyek első fordulóira általában még a tanév első félévben sor kerül, addigra viszont az előzetes ismeretek nélkül az iskolába kerülő tanulók az alacsony óraszámok mellett nem tudnak megtanulni magabiztosan kódolni, az algoritmusok értelmezését viszont könnyebben el tudják sajátítani. A később megrendezésre kerülő második, illetve harmadik fordulóig pedig több lehetőségük van arra, hogy kódolásban is fejlődjenek.

A szervező iskolák számára könnyebbség, hogy ezekhez a számítógép nélküli, papíros fordulókhoz nem kell technikai hátteret biztosítaniuk, így korlátozniuk sem kell a versenyen induló tanulók

számát az informatikaterem befogadóképessége miatt.

A felsorolt előnyök mellett a versenyek papíros fordulói alkalmasak a tehetségek felismerésére, illetve a programozás tanulásának ösztönzésére is. Emiatt nem csak Magyarországon, hanem más országok programozás versenyein is vannak ilyen típusú feladatok.

4. A feladatok kategorizálása

A programozás versenyek kiírásaiban, illetve azok honlapjain szerepelnek rövid témafelsorolások a versenyzőktől elvárt ismeretekkel kapcsolatosan. Ezek azonban meglehetősen rövidek, szűkszavúak. Az OKTV általános követelményeit például három pontban sorolták fel, ezek közül pedig csak egy vonatkozik az első fordulóban előkerülő ismeretközökre: *„rendszerszemléletű feladatmegoldás, algoritmusok kidolgozása”* [4]. A Nemes Tihamér és a Logo programozási versenyeknél ennél konkrétabb a versenyzőktől elvárt alapvető ismeretek listája, de itt is csak a nagyobb témakörök kerültek felsorolásra [2, 3].

A programozás versenyek számítógép nélküli versenyfeladatait vizsgálva úgy gondoltam, hogy a feladatok a nagy témakörökön belül vagy néhány esetben azoktól függetlenül egy másfajta szemlélet szerint is csoportosíthatók. Azt vettem észre, hogy időről időre hasonló feladattípusok térnek vissza a versenyfeladatok között, ezért a feladatok megoldásához kapcsolódó ismeretek, megoldási módszerek szerint csoportosítottam a Logo, a Nemes Tihamér és az OKTV versenyek számítógép nélküli feladatait.

A típusfeladatok időről-időre való visszatérése bizonyos szempontból nem meglepő. A középiskolás korosztálynak még igénye van arra, hogy a versenyfeladatok valamilyen szinten ismerősek legyenek számukra, emiatt nem változhatnak meg gyökeresen a feladatok. Mindez pedig az első fordulók, vagyis a számítógép nélküli feladatok esetén még inkább igaz. Természetesen ez nem azt jelenti, hogy a korábbi évek feladatainak ismerete elegendő az új versenyfeladatok sikeres megoldásához. Az új feladatok némelyike – illetve az azokhoz szükséges megoldási út – ugyan több esetben hasonlíthat egy-egy korábbi feladathoz, de eléggé különböznek is egymástól ahhoz, hogy a korábbi feladat megoldása ne legyen egy az egyben átültethető az újra.

A feladatok kategorizálása során minden feladatnál feltüntettem a megoldásukhoz szükséges módszert, a feladattípust vagy a feladat megoldásához köthető ismeretet, fogalmat. Így a feladatok megoldásához szükséges ismeret szerint határoztam meg a kategóriákat. Külön jelöltem (a későbbi táblázatokban sötétkék háttérrel) azokat a feladatokat, amelyek megoldásánál konkrét algoritmizálási tudásra is szükség van. Ilyenek például azok a feladatok, ahol pszeudokódot kellett értelmezniük, futtatniuk a versenyzőknek, illetve az abban előforduló hibákat kellett kijavítaniuk. Megkülönböztettem azokat a feladatokat is (világoskék hátteret kaptak a táblázatokban), amelyeknél szükség van ugyan valamennyi algoritmizálási képességre, de a feladat szövegében konkrétan leírásra vagy példán keresztül bemutatásra kerül az, hogy mit jelentenek az algoritmus egyes elemei.

A Logo Országos Számítástechnikai Tanulmányi Verseny előző négy tanévi számítógép nélküli feladatai az 1. táblázat szerinti besorolást kapták.

	2017-2018		
	Logo 5-6	**Logo 7-8**	**Logo 9-12**
1.	szabály értelmezése	szabály értelmezése	szabály értelmezése
2.	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása
3.	út négyzetrácson	út négyzetrácson (45 fokkal)	algoritmus értelmezése, futtatása
4.			út négyzetrácson (45 fokkal)

2016-2017			
	Logo 5-6	**Logo 7-8**	**Logo 9-12**
1.	szabály értelmezése, hibakeresés	szabály értelmezése, hibakeresés	szabály értelmezése, hibakeresés
2.	algoritmus értelmezése, futtatása, hibajavítás	algoritmus értelmezése, futtatása, hibajavítás	algoritmus értelmezése, futtatása, hibajavítás
3.	legkevesebb lépéses út keresése négyzetrácson	legkevesebb lépéses út keresése, szabály értelmezése	algoritmus értelmezése, futtatása
4.			legkevesebb lépéses út keresése, szabály értelmezése
2015-2016			
	Logo 5-6	**Logo 7-8**	**Logo 9-12**
1.	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása
2.	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása
3.	navigáció négyzetrácson	navigáció négyzetrácson	navigáció négyzetrácson
4.			algoritmus értelmezése, futtatása
2014-2015			
	Logo 5-6	**Logo 7-8**	**Logo 9-10**
1.	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása
2.	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása
3.	navigáció négyzetrácson	navigáció négyzetrácson	navigáció négyzetrácson, algoritmus értelmezése, futtatása
4.			algoritmus értelmezése, futtatása

1. táblázat: Logo Országos Számítástechnikai Tanulmányi Verseny feladatainak kategorizálása

A Logo verseny feladatai között a vizsgált tanévek mindegyikében előfordult valamilyen négyzetrácson történő navigációval kapcsolatos feladat, ami szorosan kötődik a Logo programozási nyelv sajátosságaihoz. Emellett szembetűnő, hogy az utóbbi két évben a korábbinál kevesebb olyan feladat volt mindegyik korosztályban, ahol szükség volt a Logo nyelv ismeretére vagy algoritmizálási ismeretekre.

A Nemes Tihamér Nemzetközi Informatika Tanulmányi Verseny és az Országos Középiskolai Tanulmányi Verseny programozás kategóriájának számítógép nélküli feladatait is hasonlóképp vizsgáltam. A két verseny feladatait egy táblázatban összegeztem (2. táblázat) a versenyek hasonlósága miatt, illetve amiatt, hogy bizonyos szempontból az OKTV a Nemes Tihamér verseny folytatása, feladataik többször ugyanazok vagy ugyanannak a feladatnak különböző variációi. Gyakran előfordul, hogy a különböző korosztályok feladata csak annyiban különbözik, hogy milyen mennyiségű vagy összetettségű bemeneti adat, illetve hány paraméter esetén kell megoldaniuk a feladatot a tanulóknak.

2018-2019			
	Nemes 5-8	**Nemes 9-10**	**OKTV**
1.	gráf (körmentesség)	gráf (körmentesség)	gráf (körmentesség)
2.	algoritmus értelmezése	szabályból gráf (fa)	szabályból gráf (fa)
3.	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása
4.	szabály értelmezése	szabály értelmezése	szabály értelmezése
5.	gépes	szabály értelmezése	szabály értelmezése
6.	hibakeresés algoritmusban	szabály értelmezése	szabály értelmezése
7.		algoritmus értelmezése	algoritmus értelmezése

2017-2018			
	Nemes 5-8	**Nemes 9-10**	**OKTV**
1.	rekurzió, szabály alkalmazása	rekurzió, szabály alkalmazása	gráf (elvágó pontok)
2.	rekurzió, utak száma	algoritmus futtatása, gráf	vizualizáció alapján algoritmus
3.	algoritmus értelmezése, futtatása	gráf (elvágó élek)	algoritmus értelmezése, futtatása
4.	titkosítás, kód helyreállítása	titkosítás, kód helyreállítása	hibakeresés algoritmusban
5.	gépes	algoritmus értelmezése, futtatása	rekurzió, szabály alkalmazása
6.	hibakeresés algoritmusban	rekurzió, szabály alkalmazása	titkosítás, kód helyreállítása
7.		algoritmus, ciklusból rekurzió	algoritmus, ciklusból rekurzió
2016-2017			
	Nemes 5-8	**Nemes 9-10**	**OKTV**
1.	szabályból gráf (legrövidebb kör)	mohó stratégia	mohó stratégia
2.	rekurzió, utak száma	rekurzió	rendezés
3.	úthálózat	gráf (topologikus rendezés)	gráf (topologikus rendezés)
4.	algoritmus értelmezése, futtatása	titkosítás	titkosítás
5.	gépes	algoritmus értelmezése, futtatása	vizualizáció alapján algoritmus
6.	hibakeresés algoritmusban	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása
7.		szabály értelmezése	szabály értelmezése
2015-2016			
	Nemes 5-8	**Nemes 9-10**	**OKTV**
1.	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása, hibajavítás	hibakeresés algoritmusban
2.	szabály értelmezése (véges automata)	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása
3.	szabály értelmezése (optimalizálás)	szabály értelmezése (véges automata)	szabály értelmezése (véges automata)
4.	gépes	szabály értelmezése (optimalizálás)	szabály értelmezése (optimalizálás)
5.	algoritmus kiegészítése	algoritmus értelmezése, futtatása	algoritmus értelmezése, futtatása

2. táblázat: Nemes Tihamér Nemzetközi Informatika Tanulmányi Verseny és Országos Középiskolai Tanulmányi Verseny (programozás kategória) számítógép nélküli feladatainak kategorizálása

A feladatok között itt is vannak olyanok, amelyeknél fontos, hogy a versenyzők értsék a legtöbbször pszeudokódként megadott algoritmusokat, de a feladatok jelentős része, körülbelül fele ilyen jellegű ismeretek nélkül is megoldható. Az algoritmusok megértését igénylő feladatok száma közel állandónak mondható az előző négy tanév feladatsorai alapján. A feladatok száma viszont nőtt, így az algoritmusok megértését igénylő feladatok aránya összességében csökkent a Nemes Tihamér programozási verseny és az OKTV első fordulóinak feladatsoraiban.

Az általános- és középiskolás korosztálynak szóló, már említett négy programozási versenyen a versenyzőktől elvárt ismeretek körét érdemes összehasonlítani a Nemzetközi Informatikai Diákolimpián (IOI) elvárt ismeretekkel, mivel a Nemes Tihamér verseny és az OKTV is az IOI válogatóversenyei közé tartozik. Az IOI-n elvárt ismereteket részletes Syllabus-ban foglalták össze [5]. Ebből kiderül, hogy bizonyos aritmetikai és geometriai ismeretek mellett fontos szerepet kapnak a gráfelmélet témakör egyes ismeretei, algoritmusai is a versenyfeladatokban. Így nem meglepő, hogy ezek a témakörök a Nemes Tihamér versenyen és az OKTV-n is előfordulnak.

5. Legjellemzőbb feladattípusok

A négy programozási verseny számítógép nélküli feladatai két nagy kategóriába sorolhatók. Az egyik kategóriához az algoritmusok értelmezését igénylő feladatok tartoznak, a másikhoz pedig a matematikai jellegű feladatok.

5.1. Algoritmus értelmezését igénylő feladatok

Az algoritmusok értelmezését igénylő feladatoknál három gyakori feladattípus van. Eszerint további három alkategóriát határoztam meg az algoritmusok értelmezését igénylő feladatok esetén:

- Mit csinál az algoritmus?
- hibakeresés
- algoritmus átalakítása

5.1.1. Mit csinál az algoritmus?

Ehhez az alkategóriához soroltam azokat a feladatokat, amelyeknél a versenyzőknek meg kell határozniuk azt, hogy mit csinál az adott algoritmus, illetve milyen eredményt kapunk, ha a megadott kiinduló adatok esetén végrehajtjuk az algoritmus lépéseit.

Ilyen feladat például a következő *(Nemes, 2017-18, 5-8. osztály)*:

3. feladat: Mit csinál (30 pont)

Az alábbi algoritmus egy N elemű X vektort dolgoz fel, eredményét az N elemű Y vektorba írja.

```
Valami(N,X,Y):
  D[1..N]:=0
  Ciklus i=1-től N-1-ig
    Ciklus j=i+1-től N-ig
      Ha X[i]>X[j] akkor D[i]:=D[i]+1
                   különben D[j]:=D[j]+1
    Ciklus vége
  Ciklus vége    {*}
  Ciklus i=1-től N-ig
    Y[D[i]+1]:=X[i]
  Ciklus vége    {**}
```

Eljárás vége.

A. Mi lesz a D vektorban {*}-nál és az Y vektorban {**}-nál, ha N=6, X=(3,8,1,2,9,7)?

B. Mi lesz a D vektorban {*}-nál és az Y vektorban {**}-nál, ha N=9, X=(3,9,3,1,2,1,1,8,7)?

C. Fogalmazd meg általánosan, mi lesz a D vektorban {*}-nál és az Y vektorban {**}-nál tetszőleges N és X esetén!

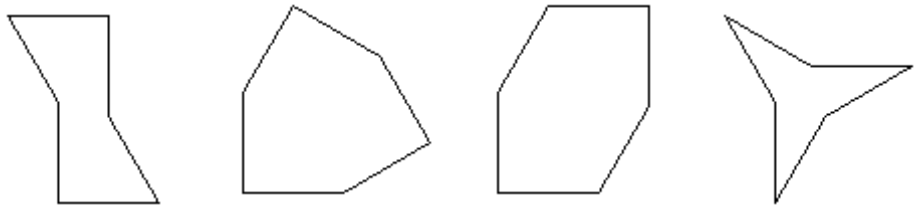
Egy másik példa ilyen típusú feladatra *(Logo, 2017-18, 5-6. osztály)*:

2. feladat: Teknős Tádé (16 pont)

Teknős Tádé hatszögeket próbált rajzolni. Sikerült négyféle változatot készítenie és az elkészült rajzokat képként is kimentette, de elfelejtette, hogy melyik rajzot melyik programmal készítette. Segíts neki párosítani a rajzokat a programokkal!

Melyik programhoz melyik ábra tartozik?

1. `ismétlés 3 [előre 50 jobbra 30 előre 50 jobbra 90]`
2. `ismétlés 3 [előre 50 balra 30 előre 50 jobbra 150]`
3. `ismétlés 2 [előre 50 jobbra 30 előre 50 jobbra 60 előre 50 jobbra 90]`
4. `ismétlés 2 [előre 50 balra 30 előre 50 jobbra 120 előre 50 jobbra 90]`



A B C D

5.1.2. Hibakeresés

A másik, algoritmus értelmezését igénylő feladattípusnál a megadott algoritmusban kell megkeresniük a hibát a versenyzőknek. Ilyen feladat például a következő *(OKTV, 2017-18, 11-12. osztály)*:

4. feladat: Összefésüléses rendezés (28 pont)

Az összefésüléses rendezés algoritmusa a következő elven működik:

- az egyelemű sorozat rendezett, nincs vele tennivaló;
- ha a sorozat több elemű, akkor
  - középen két részre osztjuk;
  - mindkét részt rendezzük az összefésüléses rendezés algoritmusával;
  - végül a két kapott rendezett sorozatot összefésüljük.

Az alábbi algoritmus ezt csinálná (az X tömb. E. és U. eleme közötti részt rendezné), azonban sajnos hibák kerültek bele. Jelöld, melyek a hibák!

```
Rendez(E,U):
  Ha E<U akkor K:=(E+U)/2
               Rendez(E,K); Rendez(K,U)
               Összefésül(E,U,K)
  Elágazás vége
Eljárás vége.
Összefésül(E,K,U):
  i:=E; j:=K+1; D:=E; Y:=X
  Ciklus amíg i≤K és j≤U
    D:=D+1
    Ha Y[i]<Y[j] akkor X[D]:=Y[j]; i:=i+1
                 különben X[D]:=Y[j]; j:=j-1
  Ciklus vége
```

```
  Ciklus amíg i≤U
    D:=D+1; X[D]:=Y[i]; i:=i+1
  Ciklus vége
  Ciklus amíg j<U
    D:=D+1; X[D]:=Y[j]; j:=j+1
  Ciklus vége
Eljárás vége.
```

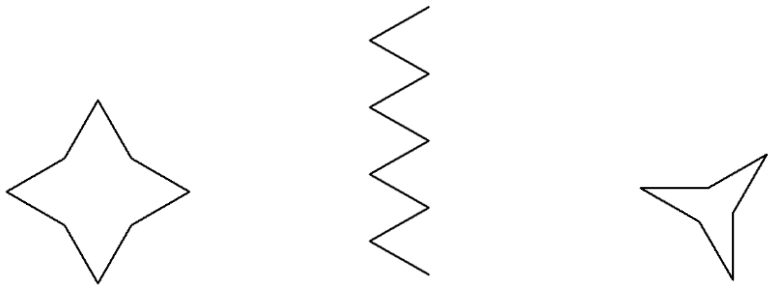
Ilyen hibakereséses, hibajavítós feladatra a Logo verseny feladatai között is van példa *(Logo, 2016-17, 5-6. osztály)*:

2. feladat: Teknős Tádé (15 pont)

Teknős Tádé szeret Logo programokat írni, viszont nem mindig tudja megkülönböztetni a bal és a jobb kezét. Ezért a Logo programjaiban időnként a BALRA helyett JOBBRA, a JOBBRA helyett pedig BALRA utasítást ír (de nem mindig).

1. `ismétlés 4 [balra 60 előre 100 jobbra 120 előre 100 balra 30]`
2. `ismétlés 4 [jobbra 60 előre 100 jobbra 120 előre 100 balra 60]`
3. `ismétlés 3 [jobbra 60 előre 100 balra 150 előre 100 jobbra 30]`

Ezekkel ezt a három ábrát szerette volna rajzolni:



1 2 3

A. Mit rajzol Tádé három programja?

B. Hogyan kell kijavítani, hogy azt rajzolja, amit várt!

5.1.3. Algoritmus átalakítása

A harmadik, algoritmus értelmezését igénylő gyakori feladattípusnál a tanulóknak meglévő algoritmust kell átalakítaniuk bizonyos feltételek szerint.

Ilyen feladat volt például a következő *(Nemes, 2017-18, 9-10. osztály)*, ahol a versenyzőknek meglévő algoritmusokat kellett rekurzívvá alakítaniuk. A feladat nem várja el a rekurzió előzetes ismeretét a tanulóktól. A feladat szövegében leírásra kerül, hogy mit jelent az, ha egy függvény rekurzív. Emellett pedig egy példán keresztül bemutatásra kerül az is, hogy hogyan lehet átírni egy ciklussal megírt algoritmust rekurzívvá. Ennek ellenére természetesen igaz az is, hogy a rekurzió ismerete, illetve az, ha a versenyző látott korábban ilyen jellegű feladatot, az nagyban hozzájárul a feladat könnyebb megértéséhez, illetve megoldásához.

7. feladat: A Rek bolygó programozói (26 pont)

Nemrég rendezték a programozók intergalaktikus találkozóját, ahol a földi programozók találkoztak a Rek bolygóról érkező kollégáikkal. A földiek meg akarták mutatni a kedvenc programjaikat, de kiderült, hogy a Rek bolygón nem ismerik a ciklusokat és az értékadást. Nincs is rá szük-

ségük, mert mindent rekurzióval valósítanak meg (vagyis olyan függvényeket írnak, amik meghívják saját magukat). A függvények értéke náluk egyszerűen az utoljára kiszámolt kifejezés értéke lesz.

Egy programot már sikerült átírnia a tolmácsnak úgy, hogy az idegenek is értsék, de a többiben néhány helyen bizonytalan. Írd be a hiányzó kifejezéseket! Szerencsére már csak a függvény paramétereket, számokat és műveleteket kell használnod. A programok paraméterei mind pozitív egészek lehetnek.

földi program	Rek program

`faktoriális(N):`
`  a ≔ 1`
`  Ciklus i=1-től N-ig`
`  a ≔ a * i`
`  Ciklus vége`
`  faktoriális ≔ a`
`Függvény vége.`
	`faktoriális(N):`
	`  ha N = 1`
	`  1`
	`  különben`
	`  N * faktoriális(N-1)`
	`Függvény vége.`

Írd át az alábbi algoritmusokat a Rek bolygó nyelvére az alábbi kódrészek kiegészítésével!

```
valami(F):
  a ≔ 1
  b ≔ 1
  Ciklus i=3-tól F-ig
    a ≔ a + b
    b ≔ a - b
  Ciklus vége
  valami ≔ a
Függvény vége.
```

```
valami(F):
  ha [          ] < 3
     [          ]
  különben
     valami([          ])+valami([          ])
Függvény vége.
```

```
valami(A,B,C):
  s ≔ 0
  Ciklus i=A-tól B-ig
    ha i mod C = 0
      s ≔ s + 1
  Ciklus vége
  valami ≔ s
Függvény vége.
```

```
valami(A,B,C):
  ha [          ] > [          ]
     [          ]
  különben ha A mod C = 0
     [          ]+valami([          ],[          ],[          ]
)
  különben
     valami([          ],[          ],[          ])
Függvény vége.
```

```
valami(A):
  c ≔ 0
  b ≔ 1
  Ciklus amíg b*b ≤ A
    c ≔ c + b * b
    b ≔ b + 1
  Ciklus vége
  valami ≔ c
Függvény vége.
```

```
valami(A):
  valami2([          ],1,0)
Függvény vége.
valami2(A,B,C):
  ha [          ] > [          ]
     [          ]
  különben
     valami2([          ],[          ],[          ])
Függvény vége.
```

5.2. Matematikai jellegű feladatok

Az algoritmusok értelmezését igénylő feladatok mellett a vizsgált versenyek számítógép nélküli feladatainak másik nagy csoportját képezik azok a feladatok, amelyek megoldásához bizonyos szintű matematikai ismeretek szükségesek. Ezek a feladatok jellegükből adódóan matematika órákon vagy szakkörön is előfordulhatnának. A matematikai jellegű kifejezés nem feltétlenül takar konkrét matematikai ismereteket. Azokat a feladatokat is ide soroltam, ahol a megoldáshoz logikus gondolkodásra, szabályalkalmazásra van szükség. Az ide csoportosított feladatok között vannak olyanok is, amelyek megoldhatóak matematikai ismeretek nélkül is, viszont a matematikai háttérismeret nagyban meg tudja könnyíteni, le tudja egyszerűsíteni a megoldás menetét.

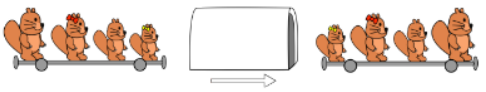
5.2.1. Megadott szabály alkalmazása

A programozási versenyek számítógép nélküli feladatainál a matematikai jellegű feladatok első csoportját azok a feladatok alkotják, amelyekben egy-egy adott szabályt kell alkalmazni a megadott adatokra. Az ilyen feladatok a HÓDítsd meg a biteket! informatika verseny feladatai között is gyakoriak. Egy ezek közül például a következő *(e-hód, 2015., benjamin – közepes, kadét – könnyű, 2013-JP-02-B, Varázslatos alagút)*:

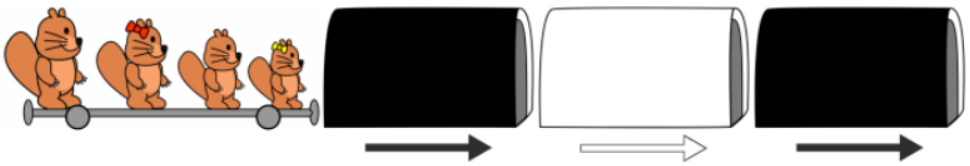
A hódvasút kétféle alagutat használ:

Ha a szerelvény egy fekete alagúton megy keresztül, az utasok fordított sorrendben jönnek ki a túloldalon.

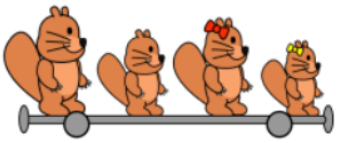
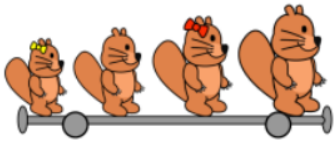
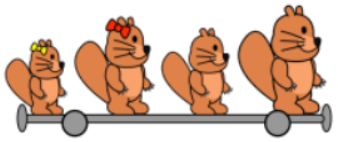
Ha a szerelvény egy fehér alagúton megy keresztül, az első és a hátsó utas helyet cserél.



A mi szerelvényünk három alagúton megy keresztül:



Milyen sorrendben jönnek ki az utasok az utolsó alagútból?

A		C	
B		D	

Egy meghatározott szabály alkalmazását várta el a versenyzőktől ez a feladat is *(Logo, 2017-18, 5-6. osztály)*:

1. feladat: 10x4-es kijelző (20 pont)

Van egy négyzetekből összeállított, 10 oszlopból és 4 sorból álló színes kijelzőnk. Ennek minden négyzetét különböző színűekre festhetjük egy speciális teknőc segítségével. A teknőcnek oszloponként egy listában kell megadnunk, hogy mely mezőket fesse ki és milyen színnel. A következő színkódokat használhatjuk:

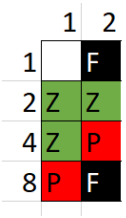
P: piros

Z: zöld

F: fekete

A szabály az, hogy a listában egy szín nem szerepelhet kétszer. A színkód előtt egy számnak kell állnia, amelyet úgy kapunk, hogy az adott sorok előtt szereplő számokat összeadjuk. A fehér négyzeteket figyelmen kívül kell hagynunk.

Nézzük az alábbi ábra részletet.



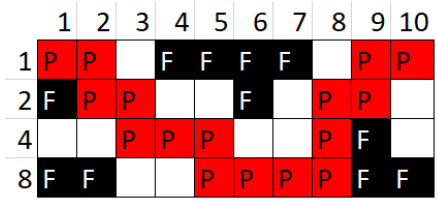
Itt az első oszlopban a piros csak a 8-as sorszámú sorban szerepel, ezért ezt a számot leírjuk, majd mögé írjuk a színkódot. Az eredmény: 8P. A zöld mező előtti sorokban a 2 és 4 szerepel, vagyis ezek összegét (6), és a Z színkódot írjuk. Az eredmény: 6Z. A fehér négyzetet figyelmen kívül hagyjuk, vagyis az oszlopok kódjai:

oszlop kódja: 8P 6Z

oszlop kódja: 9F 2Z 4P

A színeket tetszőleges sorrendben is leírhattuk volna, vagyis az 1. oszlopnál a 6Z 8P is teljesen jó megoldás.

A fenti szabályok alapján határozd meg az oszlopok kódjait az alábbi ábrára, amely egy kígyós játék egy részletét ábrázolja:



A feladatok között előfordulnak olyanok is, ahol nem a szabály alapján kell meghatározni azt, hogy mi lesz az eredmény, hanem a kezdeti érték és az eredmény alapján magát a szabályt kell kitalálniuk a tanulóknak. Ilyen jellegű feladat volt például a következő *(Nemes, 2016-17, 9-10. osztály)*, ahol még annyival nehezítették a titkosítás szabályának kitalálását, hogy a számok és a kódolt változatuk nem ugyanabban a sorrendben voltak megadva, így a versenyzőknek több lehetőséget kellett vizsgálniuk.

4. feladat: Titkosítás (40 pont)

Egy titkosítási eljárás úgy működik, hogy a kapott számok kettes számrendszerbeli bitjeit összekeverik. A keverést egy vektor írja le, amely i. elemében megadjuk, hogy a titkos kód i. bitjét a szám hányadik bitjéből kell venni (balról 0-tól sorszámozva a biteket). Például, ha a keverést a (2,1,3,0) vektor írja le, akkor a 0000, 1100, 0111 számok titkos kódjai a 0000, 0101, 1110 számok lesznek.

Add meg a keverést leíró vektort, ha a számok és a kódjuk az alábbi (a számok és a kódok sorrendje nem feltétlenül azonos):

A. Számok: 0001, 0011, 0100. Kódjuk: 0010, 1010, 0100.

B. Számok: 0001, 0011, 0111. Kódjuk: 0010, 1010, 1110.

C. Számok: 0001, 0101, 1011, 0011. Kódjuk: 1110, 1001, 1000, 1100.

D. Számok: 0101, 0111, 1011. Kódjuk: 1010, 1011, 0111.

Az előző két feladat nem csak a szabályok alkalmazásáról szól, hanem akár egy titkosítás, titkosírás felé vezető feladatsor feladatai is lehetnének. A titkosítás és a titkosírás éppúgy informatikai probléma, ahogyan matematikai is. A versenyfeladatok között talán épp emiatt szerepeltek más olyan feladatok is, amelyek valamilyen módon ehhez a témakörhöz kapcsolódtak. Ilyen feladat volt például a következő, ahol a titkosítás hibalehetőségeivel, a kódolás és dekódolás hibáinak javításával kellett foglalkozniuk a tanulóknak *(Nemes, 2017-18, 5-8. osztály)*.

4. feladat: Fényjelek (44 pont)

Ádám és Éva sötétedés után fényjelekkel kommunikálnak. Évának van egy zseblámpája, amivel piros, zöld és kék fényeket tud kiadni. Minden üzenetet azonos hosszú jelsorozattal kódolnak. Például előző héten a (piros, zöld, zöld) jelentette az igen szót, a (piros, zöld, piros) pedig a nem szót. Sajnos azonban azt tapasztalták, hogy a távolból néha nagyon nehéz megkülönböztetni a látott színt, így hibák fordulnak elő. Úgy döntöttek ezért, inkább hosszabb jelsorozatokat használnak, hogy néhány hiba esetén még a vevő ki tudja javítani a félrenézett színeket, ezzel az eredeti üzenetet helyreállítva. Ezt úgy teszik, hogy ha egy a kódtáblában nem található jelsorozatot észlelnek, akkor megkeresik azt a jelsorozatot a kódtáblában, ami a legkevesebb helyen különbözik tőle, és erre javítják.

A. A következő négy jelből álló jelsorozatokat használja Ádám és Éva: [P,Z,K,Z], [Z,K,Z,K], [K,P,P,Z], [Z,P,K,P]. Ádám a következő jelsorozatokat jegyzi fel: [K,Z,K,Z], [Z,P,K,Z], [Z,P,P,P], [P,Z,Z,Z], [K,P,P,K], [Z,K,Z,K]

A1. Helyreállítás segítségével határozd meg mi lehetett Éva üzenete!

A2. Legfeljebb hány színt nézhet félre Ádám jelsorozatonként, hogy biztosan helyes legyen a helyreállítás után az üzenet?

B. A következő öt jelből álló jelsorozatokat használja Ádám és Éva: [Z,K,P,K,K], [K,Z,Z,Z,P]. Ádám a következő jelsorozatokat jegyzi fel: [P,K,P,K,Z], [Z,K,P,K,K], [Z,Z,Z,K,P], [Z,Z,P,Z,K], [Z,K,Z,Z,P], [K,Z,P,P,P]

B1. Helyreállítás segítségével határozd meg mi lehetett Éva üzenete!

B2. Legfeljebb hány színt nézhet félre Ádám jelsorozatonként, hogy biztosan helyes legyen a helyreállítás után az üzenet?

B3. Ádám és Éva szeretne egy harmadik üzenetet felvenni a kódtáblájukba. Segíts nekik találni az új üzenethez egy új jelsorozatot úgy, hogy Ádám 2 hibát még biztosan javítani tudjon!

B4. Az előző feladatban talált jelsorozatot is a kódtáblához adva Ádám a következő jelsorozatokat észleli: [Z,Z,Z,K,K], [Z,P,P,K,Z], [P,K,Z,Z,P], [P,P,K,Z,K], [K,P,K,P,P], [K,Z,Z,Z,P]. Mi lehetett az eredeti üzenet?

5.2.2. Rekurzióval kapcsolatos feladatok

A matematikai jellegű feladatok második nagyobb kategóriáját a rekurzióval kapcsolatos versenyfeladatok alkotják. Már az algoritmusok értelmezését igénylő feladatok között is volt olyan, ahol előkerült a rekurzió, de a pszeudokód értelmezése nélkül megoldható, matematikai jellegű feladatok között is találunk olyanokat, amelyek rekurzióval oldhatók meg.

Ilyen feladat volt például a következő *(Nemes, 2017-18, 9-10. osztály)*, ahol egy-egy fa kidöntése újabb és újabb fakidöntéseket eredményezett.

1. feladat: Favágás (20 pont)

Egy fasorba N fát ültettek balról jobbra, egy vonalba. Mindegyik fának ismerjük a bal szélső fáról vett távolságát és a magasságát. Ha egy fát kivágunk, akkor az a jobboldali szomszédja felé dől, s amelyik szomszédjára rádől, az is kidől. Az 1 távolságra levő fára az 1 magasságú fa nem dől még rá, a 2 magasságú viszont igen.

Az alábbi fák esetén add meg, hogy mely fákat kell kivágni, hogy az összes fa kidőljön! A számpárok első tagja a fa távolsága a bal szélső fától, a második pedig a magassága.

A. (0,6), (3,1), (5,2), (8,1), (15,10)

B. (0,3), (2,3), (4,1), (6,3), (8,2), (10,1)

C. (0,4), (3,4), (4,3), (6,1), (7,5), (9,2), (11,4), (12,2), (14,1), (15,1), (16,5)

Egy másik rekurzióhoz kötődő feladat *(Nemes, 2017-18, 5-8. osztály)* volt az, ahol egy különleges növényfaj egyedeinek száma attól függ, hogy a korábbi évben hány faj volt, és azok hány évesek voltak.

1. feladat: Növény (36 pont)

Egy különleges növényfajt fedeztek fel az egyenlítői dzsungelben. A növény 5 évig él, élete első három évében egy-egy magjából újabb növény kel ki (azaz pl. az első évben ültetett növény a 2., 3. és 4. évben hoz magot, amit újra elültetünk, az ötödik évben még él, a hatodikban pedig elpusztul). Beszereztünk egy egyéves növény és elültettük egy arborétum üvegházába.

Töltsd ki az alábbi táblázatot, amiből kiderül, hogy a következő 10 évben hány új növény fog kikelni és melyik évben hány növényt láthatnak az üvegház látogatói!

Év	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Új növény	1									
Összes növény	1									

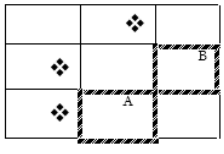
5.2.3. Kombinatorikai jellegű, útkeresős feladatok

Egy másik feladatcsoportot alkotnak a matematikához kapcsolódó, kombinatorikai jellegű, lehetséges utak számával kapcsolatos versenyfeladatok. Ezeknél a feladatoknál egyértelműen jobb helyzetbe kerülnek azok a tanulók, akik matematikai tanulmányaik során már találkoztak ilyen jellegű feladatokkal. A lehetséges utak számát a legtöbb esetben jól meg lehet határozni logikai úton is, de a matematikai tanulmányaikból ismert séma alkalmazásával egyszerűbben juthatnak el a versenyzők a jó megoldáshoz, ráadásul kisebb az esélye annak, hogy kihagynak bizonyos eseteket.

Ilyen feladat volt például a következő *(Nemes, 2017-18, 5-8. osztály)*:

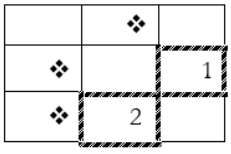
2. feladat: Kincsek (40 pont)

Egy jobbra-lefelé lejtő hegyoldalon kincseket helyeztünk el, amelyek egy részét egyetlen szánkóval szeretnénk összegyűjteni. A szánkóval a bal felső sarokból indulhatunk, és lejtő irányba (azaz vagy jobbra, vagy lefelé) haladhatunk. Amelyik mezőn átmegyünk, az ott levő kincset felvesszük.



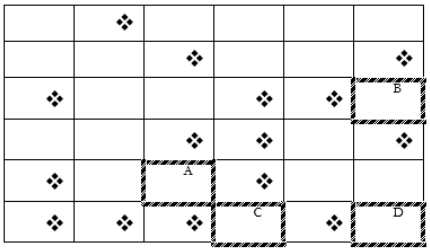
A hegyoldalon kijelöltek néhány gyűjtőpontot, a szánkóval valamelyikhez el kell jutnunk (és onnan tovább nem mehetünk).

A baloldali ábrán három kincset és két gyűjtő helyet látunk. A jobboldali ábrán látható, hogy



melyik helyre maximum hány kincset vihetünk, ha a bal felső sarokból indulunk. Egy lehetséges út az alsó gyűjtő helyhez: le, le, jobbra; a jobboldali gyűjtő helyhez: le, jobbra, jobbra.

Add meg, hogy az alábbi ábrán szereplő kincsek és gyűjtőhelyek esetén melyikbe maximum hány kincs vihető, továbbá mindegyik gyűjtőhelyhez adj is meg egy ilyen utat!

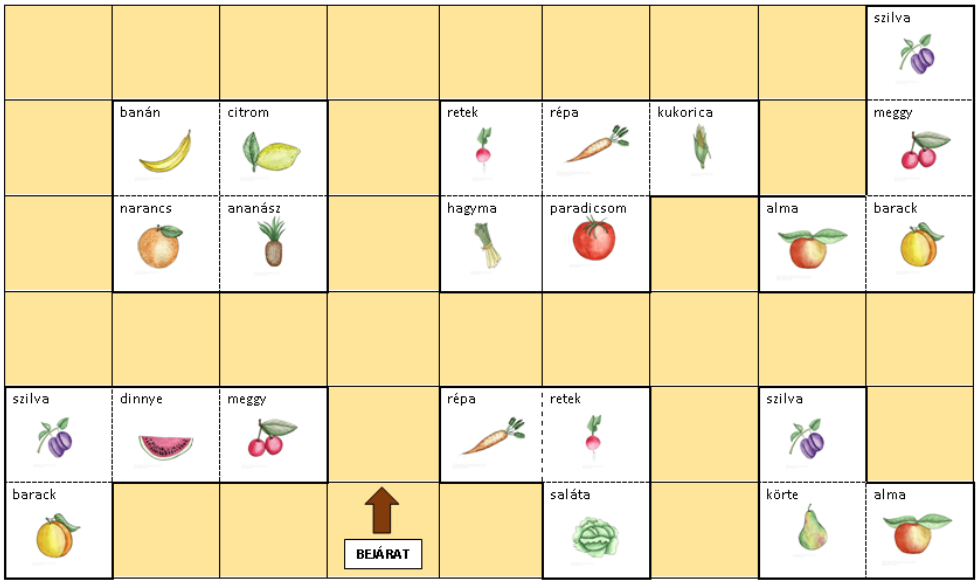


A Logo versenyen a vizsgált évek mindegyikében szerepelt olyan feladat a számítógép nélküli feladatok között, ahol egy robottal kellett bejárni egy megadott útvonalat vagy összegyűjteni valamit. Ezek a feladatok a Logo programozási nyelvhez állnak közel, de emellett a matematikához is kapcsolódnak, mert több esetben a lehető legkevesebb lépésből álló vagy valamilyen más szempontból legjobbnak számító útvonalat kell megkeresniük a versenyzőknek. Az ilyen, négyzetrácson való közlekedéssel kapcsolatos feladatok egyike például a következő *(Logo, 2016-17, 5-6. osztály)*:

3. feladat: Piac (20 pont)

Egy piacon a bevásárlás megkönnyítéséhez hordár robotokat használnak, akik az utat is megmutatják egy-egy zöldséghez, gyümölcshöz. A robot az alábbi térkép szerint tud egy vagy több mezőnyit előre (E) vagy hátrafelé (H) mozogni, illetve szükség szerint tud 90 fokot jobbra (J) vagy balra (B) fordulni. A robot alapállapotban mindig a bejárati mezőn áll és az azon szereplő nyíl által meghatározott irányba néz.

A piacon csak a kijelölt útvonalakon szabad közlekedni (a vastag körvonalú mezőkre nem lehet lépni). Egy-egy zöldséget, gyümölcsöt egy vele szomszédos mezőn állva és a zöldséget, gyümölcsöt tartalmazó mező felé fordulva lehet megvásárolni. A zöldségeket, gyümölcsöket tartalmazó mezőkön keresztülmenni nem lehet (hiszen az azokat tároló ládákon átmászni tilos).



Add meg a következő feladatokhoz tartozó robotprogramot, amivel a robot a lehető legkevesebb utasítással megoldja a feladatot!

A. Retket szeretnék vásárolni!

B. Almát szeretnék vásárolni, de a hagyma illatát nem szeretem, nem szeretnék elmenni mellette (de a sarkánál el lehet menni)!

C. Barackot, almát és meggyet szeretnék vásárolni valamilyen sorrendben!

D. Az összes szilvát meg szeretném venni!

5.2.4. Mohó stratégia, optimalizálás

A vizsgált versenyek feladatai között előfordultak olyanok, amelyek mohó stratégiával kapcsolatosak, illetve olyanok is, amelyek optimalizálási problémát rejtenek magukban. Ahogyan az utak számának meghatározását elváró feladatoknál, úgy ezeknél a feladatoknál is jelentős előnyhöz jutnak azok a versenyzők, akik korábbi (matematikai vagy informatikai) tanulmányaik során találkoztak már ilyen jellegű problémákkal, és felismerik, hogy a versenyfeladatban is ugyanazt a megoldási stratégiát kell alkalmazniuk.

Az elmúlt néhány év feladatai között mohó stratégiával kapcsolatos feladat volt például a következő *(Nemes, 2016-17, 9-10. osztály)* a Nemes Tihamér versenyen a 9-10. osztályosok feladatsorában, illetve ugyanennek a feladatnak egy kicsit módosított változata a 11-12. évfolyamosoknál is előkerült ugyanebben az évben az OKTV számítógép nélküli feladatai között.

1. feladat: Fesztiválok (36 pont)

Magyarországon sok fesztivált rendeznek. Ismerjük mindegyik első és utolsó napja éven belüli sorszámát. A fesztiválok minden nap reggeltől éjfélig tartanak, s ha egy fesztivált meglátogatunk, akkor az elejétől a végéig ott kell lennünk.

Add meg, hogy az alábbi fesztiválok közül maximum hányat tudunk meglátogatni és adj is meg egy látogatási tervet (mettől meddig milyen sorszámú fesztiválon leszünk)! (Több megoldás esetén bármelyik megadható.)

A. 6 fesztivál, 1: 2-3, 2: 2-4, 3: 5-7, 4: 3-4, 5: 2-2, 6: 1-2

B. 6 fesztivál, 1: 1-100, 2: 95-105, 3: 101-120, 4: 121-131, 5: 132-200, 6: 131-132

C. 8 fesztivál, 1: 1-10, 2: 2-6, 3: 3-7, 4: 3-3, 5: 13-13, 6: 12-13, 7: 10-11, 8: 7-9

Optimalizáláshoz kapcsolódó feladat már az 5-8. osztályosok feladatsorában is előfordul. Az ilyen feladatoknál a feladatok szövegében általában szerepel az optimális szó vagy az, hogy a legjobb vagy legkedvezőbb esetet kell megtalálni.

Ezek a feladatok könnyen érthetőek, megoldásuk azonban meglehetősen nehéz is lehet, ezért jól mérik a versenyző tanulók gondolkodási módszereit. A verseny jellegéből adódóan igaz, hogy a rövid megoldást igénylő feladatok esetében legtöbbször nem derül ki az, hogy pontosan milyen gondolatsor mentén jutott el a tanuló az általa helyesnek vélt megoldáshoz, hiszen ehhez a versenyzőknek a teljes gondolatmenetüket részletesen rögzíteniük kellene. Az első fordulót javító pedagógusnak viszont jó visszajelzést adhatnak az ilyen jellegű feladatok a tanulók gondolkodásával kapcsolatosan, mert a tanulók megoldásaikban azonban az esetek többségében egy-két gondolati elem felismerhető, így következtetni lehet a feladatmegoldás során használt módszerre is.

Optimalizálási problémával kapcsolatos feladat például a következő *(Nemes, 2015-16, 5-8. osztály)*, ahol egy kétszemélyes játéknál kellett optimális stratégia alapján gondolkozniuk a versenyzőknek. Ennek a feladatnak a módosított, nehezített változatai az idősebb korosztályok feladatsoraiban is előkerültek.

3. feladat: Játék (40 pont)

Jancsi és Juliska a következő kétszemélyes játékot játsszák. Leraknak egymás mellé 2*K kupacban gyöngyöket. Felváltva lépnek, előbb mindig Jancsi, utána Juliska. A következő lépő elveheti a valamelyik szélső kupacban levő összes gyöngyöt. Maximum mennyi gyöngy lehet az alábbi esetekben Jancsié, ha feltételezzük, hogy Juliska is optimálisan játszik, azaz a lehető legtöbb gyöngyöt akarja megszerezni? Írd le a játék menetét is!

A. 20 10

B. 1 2 8 4

C. 1 7 8 4

D. 1 2 1 2 1 2 1 2 1 2

E. 1 2 1 2 1 2 2 1 2 1

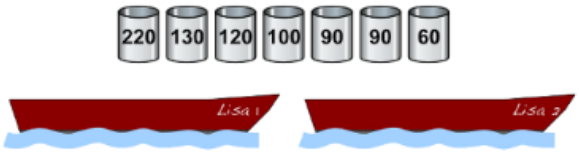
F. 1 5 3 8 4 7

G. 5 1 3 4 8 7

Az ilyen feladatok könnyű érthetőségéből és gondolkodtató jellegéből adódóan a HÓDítsd meg a biteket! verseny feladatai között is gyakoriak azok, amelyeknél optimalizálni kell valamilyen szempont szerint. Ilyen feladatra példa a következő *(e-hód, 2015., kadét – nehéz, junior – közepes, 2014-DE-08, Lisák feltöltése)*:

Bertalané és Barnabásé, a két halászé a „Lisa1” és „Lisa2” hajó – a két Lisa.

Bertalannak és Barnabásnak a két hajóval el kell szállítania pár hordó halat. A szállítást súly alapján fizetik.



Mindkét hajó legfeljebb 300kg-ig terhelhető.

Maximum mennyi halat tudnak egyszerre a két hajóval elszállítani?

A	810 kg
B	600 kg
C	590 kg
D	530 kg

5.2.5. Gráfokkal kapcsolatos feladatok

A programozás versenyek matematikához is kapcsolódó számítógép nélküli feladatainak többsége a gráfok témaköréből kerül ki. A feladatok között előfordulnak olyanok, amelyek útkereséssel (legrövidebb, leghatékonyabb), elvágó élekkel vagy elvágó pontokkal, összefüggőséggel, körmentességgel vagy topologikus rendezéssel kapcsolatosak.

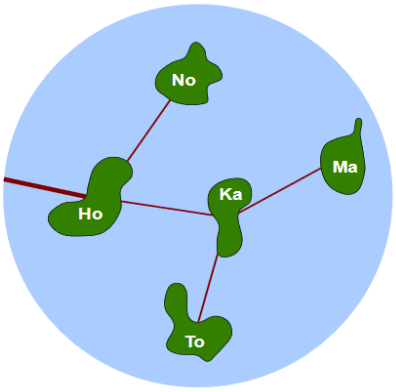
Fontos megemlíteni, hogy annak ellenére, hogy a programozás versenyek papíros versenyfeladatai között gyakran fordulnak elő gráfokkal foglalkozó feladatok, a tanulók a matematika órákon csak meglehetősen későn, 11. évfolyamon foglalkoznak ezzel a témakörrel, ráadásul akkor is csak röviden. Már az Euler-vonal, Euler-kör és a fagráfok is csak kiegészítő anyagként jelennek meg az egyik széles körben használt 11. osztályos tankönyvben [6]. A 11. osztályban elsajátítandó ismereteken kívül az egyik új kiadású, 9. évfolyamosoknak szóló matematika tankönyvben [7] egyetlen anyagrész erejéig ugyan előkerül a gráfok témaköre, de az ott előforduló ismeretanyag még kevés ahhoz, hogy a tanulók a gráfokhoz kapcsolódó versenyfeladatok megoldása során a matematika órán tanult gráfelméleti ismereteikre hagyatkozzanak.

A programozás versenyek gráfokkal kapcsolatos feladatainak szövegében talán éppen a gráfokkal kapcsolatos matematikai ismeretek, illetve azok rendszerezettségének hiánya miatt általában nem is szerepel a gráf kifejezés. A tanulók számára így a gráf csak a feladatban leírt probléma ábrázolásának eszköze. A gráfokkal kapcsolatos feladatok a szövegkörnyezet miatt általában könnyen érthetőek, megoldásukhoz azonban precíz, szisztematikus és logikus gondolkodásra van szükség. A versenyzők ilyen módon a helyes megoldáshoz is el tudnak jutni gráfelméleti ismeretek nélkül is, mégis rengeteg lehetőség van a tanulók algoritmikus gondolkodásának fejlesztésére gráfelméleti ismereteik fejlesztésének segítségével.

Gráfokkal kapcsolatos versenyfeladat gyakran előfordul a Hódítsd meg a biteket! verseny feladatai között is. Ezekben nem kerül leírásra a gráf szó, helyette például szigetek közötti utakról, kapcsolatokról van szó, ahogyan a következő példában is *(e-hód, 2017., benjamin – nehéz, kadét – közepes, junior – könnyű, 2017-DE-06, Honomakato)*, amelyben a gráf összefüggőségét kellett vizsgálniuk a versenyzőknek. Ezen a versenyen könnyebbséget jelenthet a tanulók számára az is, hogy a négy megadott válaszlehetőség közül kell kiválasztaniuk a megfelelőt. Emiatt nem szükséges minden lehetőséget megvizsgálniuk, hanem elegendő csupán a négy válaszlehetőséget végiggondolniuk, leellenőrizniük.

A Honomakato szigetcsoport öt szigetből áll: Ho, No, Ma, Ka és To.

A Ho fősziget üvegszálas kábellel csatlakozik az internethez. Ezen kívül optikai kábelek kötik össze a következő szigetpárokat: Ho és No, Ho és Ka, Ka és Ma, illetve Ka és To. Így minden kábel összeköttetésben áll a Ho szigettel és csatlakoztatva vannak az internethez.



Honomakato lakosai szeretnék stabilabbá tenni a kapcsolatot azáltal, hogy ha egy optikai kábel megsérülne, akkor is kapcsolatban maradjon minden sziget.

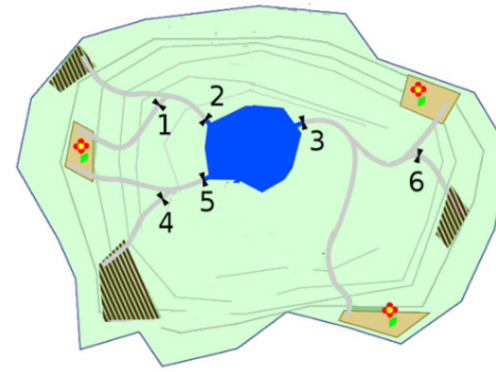
Az alábbi kábelpárok melyike teszi stabilabbá a kapcsolatot?

A) A Ho-To és No-Ma kábelek.
B) A Ho-To és Ma-To kábelek.
C) A Ka-No és No-Ma kábelek.
D) Két kábel nem elég a hálózat stabillá tételéhez.

Egy korábbi év gráfokhoz kötődő versenyfeladatában *(e-hód, 2015., kishód – könnyű, benjamin – könnyű, 2017-AT-03, Takarékos öntözés)* pedig a tóból a földekhez vezető lehetséges utakat kellett vizsgálni. Ennél a feladatnál is nagy könnyebbséget jelent a megoldás során az, hogy a négy válaszlehetőség közül kell választani.

A Nyírfa családnak van egy tava, a tó körül pedig földjei. A vizet a tóból csatornákon keresztül vezetik a földekre. Ehhez mindig a megfelelő zsilipeket nyitják ki vagy zárják le.

A Nyírfa család takarékosan bánik a tó vizével. Most csak a virággal beültetett földeket akarják megöntözni. Az üres földeknek száraznak kell maradniuk.



Segíts a Nyírfa családnak! Melyik zsilipeket nyissák meg?

A	Az 1., a 3. és a 4. zsilip legyen nyitva.
B	A 2., a 3. és a 6. zsilip legyen nyitva.
C	Az 5. és a 3. zsilip legyen nyitva.
D	A 2., a 3. és az 5. zsilip legyen nyitva.

A Nemes Tihamér programozási verseny korábbi gráfokkal kapcsolatos számítógép nélküli feladatai között is vannak a fentihez hasonló feladatok. A következőnél például bizonyos utak lezárásáról, a gráfban lévő elvágó élekről van szó. Lényeges különbség viszont az előző két példához képest a gráf megadásának módja. Az e-hód verseny példaként említett feladataiban a tanulók a gráf rajzát kapták meg, a Nemes Tihamér verseny feladatánál *(Nemes, 2017-18, 9-10. osztály)* viszont csak az élek listáját, vagyis itt már a gráf megfelelő ábrázolása is a versenyzők feladata. (Ez a Nemes Tihamér verseny és az OKTV gráfokkal kapcsolatos feladatainál általában így van.)

3. feladat: Városok (20 pont)

Ismerjük egy megye települései közötti utakat (a két település sorszámával, amelyeket összeköt-nek). Tudjuk, hogy el lehet jutni bármely településről bármely településre. Egyes utakat felújítás idejére lezárnak. Add meg, hogy melyek azok az utak, amelyek közül bármelyiket lezárva, nem lehet eljutni bármely településről bármely településre!

A. (5,2), (1,3), (1,4), (2,3), (5,6), (2,6), (3,4)

B. (9,5), (6,5), (10,6), (10,9), (3,2), (3,10), (8,3), (1,8), (7,8), (4,1), (4,7), (7,1)

Egy másik gráfokhoz kötődő feladat *(Nemes, 2016-17, 9-10.osztály)* a topologikus rendezéssel foglalkozott. Itt az élőlények táplálkozási kapcsolatainak átlátását megkönnyíti, a megoldáshoz vezető utat pedig lerövidíti, ha irányított gráf segítségével ábrázoljuk az összefüggéseket. A helyes megoldáshoz azonban más úton, gráfos ábrázolás nélkül, következtetés útján is eljuthatnak a tanulók.

3. feladat: Rendezés (31 pont)

Egy tóban sokféle élőlény él, melyekről tudjuk, hogy melyik melyiket eszi meg: egy számpár első tagja az evő, a második tagja pedig az általa megevett élőlény sorszáma). Add meg az alábbi táplálkozási kapcsolatok alapján egy olyan sorrendjüket, amiben minden párból előbb kell szerepelnie annak, aki eszik, annál, amit megeszik. (Több megoldás esetén bármelyiket.)

A. (5,2), (1,3), (1,4), (2,3)

B. (9,5), (6,5), (10,6), (10,9), (3,2), (3,10), (8,3), (1,8), (7,8), (4,1), (1,10)

A következő példaként említendő feladatnál *(Nemes, 2016-17, 5-8. osztály)* valamivel nehezebb észrevenni azt, hogy a megadott adatok gráf segítségével történő ábrázolása vezethet el könnyen a megoldáshoz. Ennél a feladatnál a gyanúsítottak állításai alapján érdemes elkészíteni egy gráfot, majd ebben a gráfban kell megkeresni a legrövidebb kört, mert a legrövidebb körhöz tartozó csúcsoknak megfelelő személyek alkotják a gyanúsítottak legkisebb létszámú csoportját.

1. feladat: Hazudósok (40 pont)

Egy bűncselekmény helyszínén N gyanúsított járt. A rendőrségi kikérdezésre kétféle választ adhattak:

- Együtt(i,j): az i. állítása szerint az i. és a j. találkozott egymással a bűncselekmény helyszínén
- Előbb(i,j): az i. állítása szerint az i. előbb elment, mint a j. megérkezett

A gyanúsítottak állításai ellentmondóak, pontosan egy valaki hazudott. Add meg az alábbi állítások csoportjára a legkisebb létszámú gyanúsított csoportot, amiben biztosan van hazudós!

A. Együtt(1,2), Előbb(2,4), Előbb(3,1), Előbb(4,3), Együtt(1,4)

B. Együtt(1,2), Együtt(4,3), Előbb(3,2), Előbb(1,3)

C. Előbb(4,2), Előbb(1,2), Előbb(3,4), Előbb(4,1), Előbb(2,3)

6. Összegzés

Összességében elmondható, hogy a programozás versenyek számítógép nélküli feladatainak jelentős részénél előkerülnek bizonyos matematikai ismeretek. Az ilyen feladatok megoldását általában megkönnyíti, leegyszerűsíti, ha a tanulók rendelkeznek a feladathoz kapcsolódó matematikai ismeretekkel, de általában szisztematikus próbálgatással, megfelelő következtetés útján is megoldhatóak ezek a feladatok.

A programozás versenyek feladataiban gyakran előforduló matematikához kapcsolódó feladattípusokat a megadott szabály alkalmazását igénylő, a kombinatorikai jellegű, a rekurzióval, mohó stratégiával vagy optimalizálással, illetve a gráfelmélettel kapcsolatos feladatok jelentik. A versenyfeladatok vizsgálata és kategorizálása alapján úgy gondolom, hogy a tanulók programozási versenyekre – főleg azok első fordulóira – való felkészítése során az algoritmizálási (pszeudokóddal, hibakereséssel, algoritmusok átalakításával kapcsolatos), programozási feladatok megoldása és gyakorlása mellett érdemes a feladatokban előforduló matematikai témakörökkel is megismertetni a tanulókat. A versenyen nagy előnyt jelenthet számukra, ha nem akkor találkoznak először például mohó stratégiára épülő vagy optimalizálással kapcsolatos feladattal. Ezek mellett pedig talán a legfontosabb az, hogy a verseny előtt lássanak gráfokhoz kötődő feladatokat, és bátran használják a gráfokat a szemléletes ábrázolás eszközeként bizonyos problémák esetén. Mindez már az általános iskolás korosztály esetében is hasznos, hiszen már a nekik szóló versenyfeladatokban is előfordulnak gráfokkal kapcsolatos feladatok. A középiskolásoknál pedig különösen fontosak lehetnek ezek a matematikai ismeretek a számítógép nélküli programozás feladatok megoldásánál, mert nekik általában nagyobb, összetettebb példákon kell megoldaniuk az ilyen feladatokat, ezek átlátása pedig jóval nehezebb a megfelelő matematikai háttérismeretek nélkül.

7. Hivatkozások

1. „e-hód - Hódítsd meg a biteket!,” http://e-hod.elte.hu/ utoljára megtekintve: 2018.11.20.
2. „Logo Országos Számítástechnikai Tanulmányi Verseny,” http://logo.inf.elte.hu/index.html (utoljára megtekintve: 2018.11.20.)
3. „Nemes Tihamér Nemzetközi Informatikai Tanulmányi Verseny - Programozás kategória,” http://tehetseg.inf.elte.hu/nemes/index.html (utoljára megtekintve: 2018.11.20.)
4. „A 2018/2019. tanévi OKTV-vel kapcsolatos tudnivalók,” https://www.oktatas.hu/pub_bin/dload/kozoktatas/tanulmanyi_versenyek/oktv/oktv2018_2019_vk/116_informatika_1819.pdf (utoljára megtekintve 2018.11.20.)
5. „International Olympiad in Informatics,” https://ioinformatics.org/files/ioi-syllabus-2018.pdf (utoljára megtekintve: 2018.11.20.)
6. J. Kosztolányi, I. Kovács, K. Pintér, J. Urbán és I. Vincze, Sokszínű matematika 11., Tizenötödik kiadás, Szeged: Mozaik Kiadó, 2017, pp. 38-62.
7. J. Kosztolányi, I. Kovács, K. Pintér, J. Urbán és I. Vincze, Sokszínű matematika 9., Hatodik kiadás, Szeged: Mozaik Kiadó, 2018, pp. 38-42.

Nevezetes felsorolók funkcionálisan

Visnovitz Márton[1], Horváth Győző[2]

{[1]visnovitz.marton, [2]horvath.gyozo}@inf.elte.hu
ELTE IK

Absztrakt. A programozás oktatásában klasszikusan a programozási tételekre alapozva az objektumelvű programozás irányába haladunk. Az objektumelvű programozás egyik megközelítése a felsorolók, felsorolható adatszerkezetek implementációja, a tanult programozási tételek alkalmazása ezen felsorolókra. Ez a cikk bemutatja, hogyan lehetséges az objektumelvű és funkcionális programozás paradigmáinak együttes használatával olyan felsorolókat létrehozni, melyekre alkalmazhatóak a hajtogatás elvén alapuló, funkcionálisan megvalósított programozási tételek.

Kulcsszavak: funkcionális programozás, objektumelvű programozás, felsorolók

1. Bevezetés

Korábbi cikkünkben [1] bemutatásra került, hogy hogyan lehetséges a magyarországi programozásoktatásban meghatározó programozási tételeket [2] a funkcionális programozásban jellemző módszerek segítségével megvalósítani. A kutatás célja azt vizsgálni, hogy a jelenleg széles körben oktatott ismeretanyag milyen mértékben valósítható meg különféle paradigmák (és azok kombinációjának) segítségével, ezzel szélesítve a bemutatott programozási paradigmák körét. Az Eötvös Loránd Tudományegyetemen a *Programozás* tárgy (korábbi nevén *Programozási alapismeretek)* foglalkozik a programozási tételek megismertetésével. Erre a tanegységre épül az *Objektumelvű programozás* (korábbi nevén *Programozás*), mely általánosítja a programozási tételeket felsoroló típusokra. Ebben a cikkben azt mutatjuk be, hogy hogyan lehet megvalósítani a felsorolókat (illetve néhány nevezetes felsorolható típust) a funkcionális programozás és az objektumelvű programozás mintáinak egyesítésével. Az így megvalósított típusokra alkalmazhatóak a korábbi cikkben [1] megvalósított, hajtogatáson (*folding*) alapuló programozási tételek.

2. Funkcionális felsorolók általánosan

Klasszikus értelemben *felsorolóknak* (vagy *felsoroló objektumoknak*) nevezzük azon típusokat, melyek képesek egy adatnak valamilyen értelemben vett első elemére ráállni, majd a soron következőre egészen addig, amíg van ily módon felsorolható elem [3]. Ez a definíció az imperatív, ciklusok segítségével történő felsorolást támogató meghatározás. A funkcionális programozás segítségével meghatározott programozási tételeket alapul véve más fajta definíciót kell adnunk, hogy felsorolás segítségével fel tudjunk dolgozni adatokat. A klasszikus meghatározáshoz hasonlóan szükségünk van arra, hogy el tudjuk kérni a felsorolónk valamilyen értelemben vett következő elemét, de mivel a további feldolgozás nem léptetéssel, hanem a maradék elemek rekurzív feldolgozásával történik (hajtogatás módszer), ezért a másik fő műveletünk a maradék elemek lekérdezése. Emellett szükség van a felsoroló ürességét (végét) jelző műveletre is. Ennek megfelelően mondhatjuk, hogy hogy a *funkcionális felsoroló* alatt olyan adatszerkezetet értünk, melyről *eldönthető, hogy üres-e, valamint felbontható első elemre és az azt követő elemeket felsoroló adatszerkezetre*. A klasszikus értelemben vett, és a funkcionális felsorolók műveleteit az **1. táblázat** mutatja.

„Klasszikus” felsorolók		Funkcionális felsorolók	
Első elemre állítás	*.First()*	Végére ért-e	*.vegeE()*
Végére ért-e	*.End()*	Következő elem	*.kovetkezo()*
Aktuális elem	*.Current()*	Maradék elemek felsorolója	*.maradek()*
Következő elemre állítás	*.Next()*		

1. táblázat: „Klasszikus” és funkcionális felsorolók műveletei

Fontos, hogy mivel funkcionális programozással dolgozunk ezért a felsorolókat úgy kell megvalósítani, hogy azoknak a belső állapota ne legyen megváltoztatható (*immuatable*), vagyis minden műveletet (metódus) úgy kell implementálni, hogy azok ne az eredeti adatszerkezetet módosítsák, hanem egy egyedi értéket, vagy egy új felsorolót adjanak eredményként.

2.1. Gyűjtemények

A felsorolók egyik speciális esetei a *gyűjtemények* (továbbiakban általános típusként *G*-vel jelöljük a gyűjtemények általános típusát). Ezen felsorolók valamilyen értékeket tárolnak, ezeket lehet felsorolni. Gyűjtemények esetében lehetséges a $G \rightarrow G$ típusú, gyűjteményről gyűjteményre leképező programozási tételeket alkalmazni, mint amilyen a *másolás* vagy a *kiválogatás*. Ahhoz, hogy ezeket a tételeket meg tudjuk valósítani szükség van egy további műveletre, a gyűjteményhez új elemet hozzáadó *.beszur()* metódusra.

3. Funkcionális felsorolók megvalósítása

A funkcionális felsorolók megvalósításában ötvözzük az objektumorientált és a funkcionális programozás módszereit. Ezáltal megvalósul az egységbe zárás, illetve a típusmegvalósítás osztályok segítségével, de ezeket az osztályokat és példányaikat úgy kell használnunk, hogy azok ne sértsék a funkcionális programozás elveit. Az implementációban az objektum-példányok nem megváltoztathatók és a függvények tiszta függvényként működnek, vagyis adott bemenetre mindig ugyanazon kimenetet produkálják. Ez utóbbi megkötést úgy értelmezzük, hogy egy osztály példányának esetében egy metódusnak magát az objektumot (*this*) is bemenetének tekintjük. Az alább adott megvalósítás olyan, hogy egy egyszer létrejött objektumnak nem tud megváltozni a belső állapota, ezért ez a tiszta függvény tulajdonság teljesül a metódusaira is. A megvalósításhoz a TypeScript [4] programozási nyelvet választottuk, mivel ez támogatja az objektumorientált és a funkcionális paradigmát is, így nem okoz gondot a programozási koncepciók együttes használata sem.

Az objektumorientált programozás lehetőséget nyújt arra, hogy olyan implementációt készítsünk a felsorolóinkhoz, mely egyszerre szolgál *interfészként* a konkrét felsorló-típusokhoz, illetve tartalmazza metódusként a programozási tételek megvalósítását is a megfelelő interfészre. Ennek megfelelően a *felsorolók* és a *gyűjtemények* általános osztályát *absztrakt osztályok* segítségével valósítottuk meg. Az absztrakt osztályokon belül a szükséges műveletek metódusai megvalósítás nélkül, kizárólag típusszignatúrával szerepelnek. Az absztrakt osztályokat sablonosztályokként, tetszőleges *H* típusú felsorolt elemekre implementáltuk:

```
abstract class Felsorolo<H> {
  abstract vegeE:     () => boolean
  abstract kovetkezo: () => H
  abstract maradek:   () => Felsorolo<H>
}
```

1. ábra: A felsorolók absztrakt osztályának megvalósítása

```
abstract class Gyujtemeny<H> extends Felsorolo<H> {
  abstract maradek: () => Gyujtemeny<H>
  abstract beszur:  () => Gyujtemeny<H>
}
```

2. ábra: A gyűjtemények absztrakt osztályának megvalósítása

A gyűjtemény osztályban felüldefiniáljuk a maradék művelet szignatúráját (2. ábra). Erre azért van szükség, hogy a gyűjteményekre definiált műveletek láncolva típushelyesek maradjanak. A példában (3. ábra) látható kifejezés típushelytelen lenne, ha *maradek* metódus *Felsorolo* típusú értékkel térne vissza, hiszen arra nincs definiálva a *beszur* művelet.

```
gyujtemeny.maradek().beszur(ertek)
```

3. ábra: A gyűjtemények absztrakt osztályának megvalósítása

4. Programozási tételek funkcionális felsorolókra

A programozási tételek korábbi cikkünkben [1] bemutatott funkcionális megvalósításai általánosíthatók az előbbiekben bevezetett *Felsoroló* és *Gyűjtemény* absztrakt típusokra. A tömbökre megvalósított tételek egyes műveletei párhuzamba állíthatóak a fent meghatározott metódusokkal. A megfeleltetést a 2. táblázat mutatja.

Művelet	TypeScript tömbre	Funkcionális felsorolókra
Üresség vizsgálata	`x0 === undefined` *a következő elem definiálatlan*	`x.vegeE()`
Következő elem	`[x0, ...xs]` *a tömb felbontása első és további elemekre, ezek közül az előbbi*	`x.kovetkezo()`
További elemek	`[x0, ...xs]` *a tömb felbontása első és további elemekre, ezek közül az utóbbi*	`x.maradek()`
Elem beszúrása	`[e, ...xs]` *új tömb konstruálása egy elem és a mögé kibontott sorozat segítségével*	`x.beszur(e)`

2. táblázat: A gyűjtemények absztrakt osztályának megvalósítása

Ezen megfeleltetés alapján bármelyik tétel átírható úgy, hogy a működjön a *Felsorolo* vagy *Gyujtemeny* típusú adatszerkezeteken is. A továbbiakban három konkrét tétel implementációját mutatjuk be, a *sorozatszámítás*, *másolás* és *kiválogatás* tételét. Azért erre a három tételre esett a választás, mert ezen tételek köré szerveződik a többi tétel magasabb rendű függvényekkel (*higher order functions)* történő megvalósítása a funkcionális programozásban [1]. A tételek közül a *sorozatszámítás* tetszőleges *Felsorolo* objektumon végrehajtható, ezért azt a *Felsorolo* osztály metódusaként valósítjuk meg, míg a *másolás* és a *kiválogatás* tételt – mivel azok építenek a *.beszur()* műveletre – a *Gyujtemeny* absztrakt osztály metódusaként implementáljuk. Ebből a tulajdonságból következik, hogy a *Felsorolo* típusba tartozó, de nem *Gyujtemeny*-ből származtatott osztályok esetén az egy konkrét értékre képező tételek mindegyikét (*megszámolás, eldöntés, kiválasztás, lineáris keresés)* a *sorozatszámítás* tételre szükséges visszavezetni.

4.1. Sorozatszámítás

```
abstract class Felsorolo<H> {
  // ...
  public sorSzam: <M> (f: (s: M, e: H) => M, k: M): M =
    (f, k) => this.vegeE()
      ? k
      : f(this.maradek().sorSzam(f, k), this.kovetkezo())
}
```

4. ábra: A sorozatszámítás tétel szignatúrája és megvalósítása a *Felsorolo* absztrakt típusban

A *sorozatszámítás* tételt megvalósító metódust (`sorSzam`) a 4. ábra mutatja be TypeScript nyelven. A megvalósítás egy egyszerű esetszétválasztáson alapszik, mely attól függően, hogy végére értünk-e az adott felsorolónak, a k kezdőértéket adja eredményül, vagy a maradék részre rekurzívan kiszámolt részeredmény és a következő elemre alkalmazott f függvény eredményét. A tétel egyértelmű, 2. táblázat szerinti átirata a tömbökre történő megvalósításnak.

4.2. Másolás

```
abstract class Gyujtemeny<H> {
  // ...
  public masol: <M> (f: (e: H) => M): Gyujtemeny<M>
    (f) => this.vegeE()
      ? this.constructor()
      : this.maradek().masol(f).beszur(f(this.kovetkezo()))
}
```

5. ábra: A másolás tétel szignatúrája és megvalósítása a *Gyujtemeny* absztrakt típusban

A *másolás* tételt megvalósító metódust (`masol`) (5. ábra) a *Gyujtemeny* osztályon belül valósítjuk meg, eredménye szintén egy *Gyujtemeny*, de már nem H, hanem tetszőleges M típusú elemek gyűjteménye, amennyiben a leképezést végző f függvény $H \rightarrow M$ típusú.

A tétel megvalósításában megjelenik a TypeScript nyelv egy sajátossága a *this.vegeE()* feltételes kifejezés igaz ágában: *this.constructor()*. A TypeScript nyelvben az úgynevezett prototípusos objektumorientált [5] jelleg miatt egy osztály konstruktora egy példányon keresztül is meghívható. Ez a konstruktor paraméter nélkül meghívva létre fog hozni egy új, üres *Gyujtemény* objektumot. A sablonparaméter (M) a metódus szignatúrájából következik, mivel ott kikötöttük, hogy a metódus M típusú elemek gyűjteményére képez.

4.3. Kiválogatás

```
abstract class Gyujtemeny<H> {
  // ...
  public kivalogat: (T: (e: H) => boolean): Gyujtemeny<H> =
    (T) => this.vegeE()
      ? this.constructor()
      : T(this.kovetkezo())
        ? this.maradek().kivalogat(T).beszur(this.kovetkezo())
        : this.maradek().kivalogat(T)
}
```

6. ábra: A kiválogatás tétel szignatúrája és megvalósítása a *Gyujtemeny* absztrakt típusban

A *kiválogatás* tételt megvalósító metódusban (`kivalogat`) (6. ábra) szintén megjelenik az új üres gyűjtemény létrehozása, de ebben az esetben H típusú elemekből jön létre az új gyűjtemény. A logi-

ka hasonló, mint a *másolás* tételnél, csak kiegészül egy további elágazással, melyben azt vizsgáljuk, hogy a soron következő elemre teljesül-e a *T* tulajdonság. E szerint ágazik el a függvény, hogy beszúrja-e a keletkező gyűjteménybe az aktuálisan feldolgozott elemet vagy nem.

5. Nevezetes felsorolók

A felsorolókat és gyűjteményeket leíró, tételeket megvalósító absztrakt osztályok segítségével lehetőségünk van konkrét, nevezetes *felsoroló típusokat* [3] megvalósítani. A továbbiakban a teljesség igénye nélkül néhány fontos felsoroló típus esetén mutatjuk meg a *Felsorolo* vagy a *Gyujtemeny* absztrakt metódusainak megvalósítását.

5.1. Egész-intervallumot felsoroló típus

Az egész intervallumot felsoroló típus egy adott [*m*..*n*] intervallum elemeit sorolja fel *m*-től, *n*-ig egyesével. Ennek mintájára sok hasonló felsoroló készíthető. Ez a felsoroló nem tekinthető gyűjteménynek, mivel csak a két végpontja és egy szabály (+1 hozzáadás) alkotja. Egész számokat sorol fel, ezért a *Felsorolo* osztály altípusa *number* típusértékkel.

```
class Intervallum extends Felsorolo<number> {
  constructor (private m: number, private n: number) { super() }

  public vegeE: () => boolean =
    () => this.m > this.n
  public kovetkezo: () => number =
    () => this.m
  public maradek: () => Intervallum =
    () => new Intervallum(this.m + 1, this.n)
}
```

7. ábra: Az egész-intervallumot felsoroló típus megvalósítása

A megvalósítás alapja, hogy az *Intervallum* objektumokat mindig egy *m* és egy *n* intervallum-határ értékkel hozzuk létre. Ezeket az értékeket kapja az osztály konstruktora paraméterként. A konstruktor paraméterei mellett szereplő, láthatóságot jelölő *private* kulcsszó a TypeScript-ben egyben azt is jelöli, hogy az így kapott értékek egyből tárolásra is kerülnek a létrejövő objektum belső állapotában.

A *maradek* metódus implementációjában fontos, hogy nem az aktuális objektum m és n értékeit módosítjuk, hanem egy új *Intervallum* felsorolót adunk vissza a már módosított határoló értékekkel. Ez azért szükséges, mert enélkül változna a belső állapot, ami sértené az immutábilis tulajdonságot.

5.2. Sorozatot felsoroló típus

A *sorozatot felsoroló típus* valamely adott típusú (*H*) elemek olyan *gyűjteményét* sorolja fel, melyben lehetséges index alapján egy elemet olvasni vagy módosítani, melynek ismerjük a hosszát, és mely sorozat bővíthető és melyből elemeket ki lehet venni. Az alább megadott implementáció azt az egyszerű esetet mutatja be, amikor a sorozatot az elejétől a végéig járjuk be, habár más fajta bejárások is lehetségesek lennének. Az ilyen sorozatok – a belső reprezentációtól függetlenül – az alábbi műveleteket kell megvalósítsák:

```
class Sorozat<H> extends Gyujtemeny<H> {
  public olvas:   (index: number) => H = //...
  public modosit: (index: number, ertek: H) => Sorozat<H> = //...
  public hossz:   () => number = //...
  public hozzaad: (ertek: H) => Sorozat<H> = // ...
  public kivesz:  () => Sorozat<H> = // ...
}
```

8. ábra: A sorozat típus műveletei

Amennyiben a sorozat műveletei (8. ábra) megvalósításra kerülnek, akkor a felsoroló műveletek ezek segítségével az alábbi módon implementálhatók.

```
class Sorozat<H> extends Gyujtemeny<H> {
  // ...
  public vegeE: () => boolean =
    () => this.hossz() === 0
  public kovetkezo: () => H =
    () => this.olvas(0)
  public maradek: () => Sorozat<H> =
    () => this.kivesz()
  public beszur: (e: H) => Sorozat<H> =
    () => this.hozzaad(e)
}
```

9. ábra: A sorozat-felsoroló műveletei

Ebben az általános implementációban nincs megadva a belső reprezentáció, illetve a *Sorozat* típus saját műveleteinek megvalósítása. Ennek oka, hogy a felsoroló műveletek ezektől függetlenek. Azt azonban elmondhatjuk, hogy a *beszur* és a *kivesz* műveletek működésétől függően (a sorozat elejével vagy végével dolgozik) a sorozat lehet *verem* vagy *sor* adatszerkezet is.

5.3. Halmazt felsoroló típus

A *halmaz* olyan adatszerkezet, melyben azonos típusú értékeket tárolunk sorrendiség és multiplicitás nélkül. Egy ilyen adatszerkezen az alábbi műveletek léteznek: *halmaz ürességének vizsgálata*, *egy elem vizsgálata, hogy benne van-e a halmazban*, *egy elem hozzáadása a halmazhoz* és *egy elem kivétele egy halmazból*. Ezen műveletek szignatúráját a **ábra** mutatja.

```
class Halmaz<H> extends Gyujtemeny<H> {
  public uresE:     () => boolean = //...
  public tartalmaz: (ertek: H) => boolean = //...
  public belerak:   (ertek: H) => Sorozat<H> = //...
  public kivesz:    (ertek: H) => Sorozat<H> = // ...
}
```

10. ábra: A halmaz típus műveletei

Ezek mellett szükség van egy olyan műveletre, mely kiválaszt egy tetszőleges elemet a halmazból. Ez a művelet sokféleképpen implementálható, de a legtöbb megvalósításban egy determinisztikus műveletet kapunk.

```
  public kivalaszt: () => H = //...
```

11. ábra: A halmaz típus *kivalaszt* művelete

Ha ez a metódus is adott, akkor a felsoroló műveletei az alábbi módon valósíthatóak meg:

```
class Halmaz<H> extends Gyujtemeny<H> {
  // ...
  public vegeE: () => boolean =
    () => this.uresE()
  public kovetkezo: () => H =
    () => this.kivalaszt()
  public maradek: () => Sorozat<H> =
    () => this.kivesz(this.kivalaszt())
  public beszur: (e: H) => Sorozat<H> =
    () => this.belerak(e)
}
```

12. ábra: A halmaz felsoroló műveletei

6. Konklúzió

A fentiekből látszik, hogy a funkcionális paradigma segítségével bevezetett programozási tételekre építkezve az ismeretek tovább bővíthetők a típusok, típuskonstruckiók, felsorolható adatszerkezetek irányába. Ez a megközelítés kombinálja a funkcionális és az objektumorientált programozási paradigmákat. Módszertani szempontból ez azért lehet előnyös, mert így többféle programozási módszer is bemutatásra kerül, nem szükséges ezeket külön-külön tárgyalni. Ezen az irányvonalon továbbhaladva bemutatható a felsorolók hagyományos, imperatív mintákra épülő megvalósítása is, valamint a különféle paradigmák kombinálásának lehetőségei. Végső soron ez a módszer azt a célt szolgálhatja, hogy kevés eszközzel mutathassunk meg minél több lehetőséget és a tanulókat arra neveljük, hogy a megfelelő célra a megfelelő eszközt válasszák.

A fenti példákban a TypeScript nyelvet választottuk a példákhoz, mivel ez számos paradigmát támogat. Természetesen sok esetben sem a módszer, sem pedig a megvalósítások nem adnak optimális megoldást az egyes problémákra, de a cél itt a tanulási folyamat támogatása.

Köszönetnyilvánítás

EFOP-3.6.1-16-2016-00023: Kutatás-fejlesztési tevékenység megvalósítása az Eötvös Loránd Tudományegyetem szombathelyi kampuszán – A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Irodalom

1. Visnovitz Márton: *Programozási nyelvek funkcionálisan.* INFODIDACT 2017.
2. Szlávi Péter, Zsakó László: *Módszeres programozás: Programozási tételek.* ELTE Informatikai Kar, 2008.
3. Gregorics Tibor: *Programozás 1. kötet Tervezés.* ELTE Eötvös Kiadó, 2013.
4. *TypeScript – JavaScript that scales.* Microsoft, 2018. https://www.typescriptlang.org (utoljára megtekintve: 2018. 11. 15.)
5. *Prototype-based programming - MDN* Web *Docs Glossary: Definitions of Web-related terms | MDN.* MDN, 2018. https://developer.mozilla.org/en-US/docs/Glossary/Prototype-based_programming (utoljára megtekintve: 2018. 11. 15.)