# INFODIDACT'2019

## 12. Informatika Szakmódszertani Konferencia


## Előadáskötet

2019

Szerkesztette: Dr. Szlávi Péter, Dr. Zsakó László
Megjelenés: 2020. január



**ISBN** 978-615-80608-3-7

## Tartalom

# Egy tábor margójára

Aszalós László[1], Bakó Mária[2]

[1]aszalos.laszlo@inf.unideb.hu, [2]bakom@unideb.hu
Debreceni Egyetem, IK és GTK

**Absztrakt.** Az informatika iránti érdeklődés felkeltésének egyik eszköze a nyári informatikai tábor. A Debreceni Egyetem Informatika Kara egy pályázat[1] révén jutott olyan lehetőséghez, mellyel ilyen táborokat tudott szervezni. Az első nyáron csak kezdők részére hirdetett tábort, de többen olyanok is érdeklődtek, akik már túl voltak az alapozáson. Épp ezért a pályázat második nyarán már mind a kezdő, mind a haladó programozói tábor elindult. Ebben a cikkben ez utóbbit kívánjuk bemutatni, a feldolgozni kívánt tematikát, a felhasznált eszközöket, illetve a tapasztalatainkat.

**Kulcsszavak:** tehetséggondozás, funkcionális paradigma, Racket programozási nyelv

## 1. Tervezési fázis

Az informatika mára igen szerteágazóvá vált. A nyolcvanas években a BASIC volt az egyetlen széles körben elérhető programozási nyelv (a maga sokféle nyelvjárásával), így minden kezdő ezzel ismerkedett meg. A sokak számára elérhető *home computer* is ezt a programozási nyelvet ismerte alapból, ezt lehetett használni pár másodperccel annak bekapcsolása után. Csak a haladó felhasználók érezték meg ennek a nyelvnek a korlátait, és fordultak más programozási nyelvek iránt, melyek között hazánkban a Forth és az assembly igen előkelő helyet szerzett meg. A későbbiekben a Basic helyét átvette a Pascal, majd ezt követően már nagyon hosszúvá vált az elsőként tanítandó programozási nyelvek listája. Szinte vallásháború van a mai napig az egyes nyelvek evangélistái között, senki nem kíván engedni a saját kedvencéből. Ennek eredményeképpen a haladó táborunkra jelentkezettek előismereteinél előfordult a C#, C++, Python, JavaScript, sőt a LabView is.

Ha a tábor programozási nyelvének ezek egyikét választottuk volna, akkor az a pár személy – aki már jól ismeri ezt a nyelvet – unatkozott volna, amíg az összes többit sikerül az ő szintjére felhozni. Persze egy akár több éves előnyt nem lehet egy hét alatt behozni, így mindenképpen szétszakad a társaság, amit felettébb megnehezíti a tábor megszervezését és lebonyolítását úgy, hogy minden résztvevő azt érezze a végén, hogy hasznosan töltötte az időt.

Egy huszárvágással elintéztük, hogy minden táborozó azonos szinten legyen, ez pedig a kezdő szint volt. Habár az előbb felsorolt nyelvek mindegyikében elérhető az objektum-orientált paradigma, jellemzően ez nem kap szerepet az általános iskolai, illetve középiskolai oktatásban. (Most tekintsünk el a szakirányú szakgimnáziumi képzéstől.) Azaz a hagyományos, imperatív megközelítés az általános – még ha az adott nyelv más paradigmákat is támogat –, azaz az imperatív megközelítésben szocializálódott minden résztvevő. Ebben a szellemben fogalmazzák meg az egyes problémákat megoldó algoritmusokat. Annak érdekében – hogy a komfortzónájukból kimozgassuk őket –, egy teljesen más paradigmával kívántuk megismertetni a diákokat, ez pedig a funkcionális programozás volt.

[1] EFOP-3.4.4-16-2017-00023 AZ MTMI szakokra való bekerülést elősegítő innovatív programok megvalósítása a Debreceni Egyetem vonzáskörzetében

Fontosnak tartottuk, hogy más módon, más szemszögből lássák az esetleg korábban már megismert feladatokat, kérdéseket. Érzésünk szerint nem volt elrugaszkodott célunk, csupán egy kis betekintést kívántunk adni egy számukra addig ismeretlen területre. Az már az adott diák érdeklődésén, kíváncsiságán múlik, hogy a tábor egy hete után szeretne még ezzel a paradigmával foglalkozni – akár ilyen irányú programozási nyelve(ke)t használva – vagy az általa eddig használt nyelvben fedezi fel és veszi birtokba az ott található funkcionális eszközkészletet, vagy csak egy rossz élményként tekint majd vissza a táborra. Véletlenül sem kívántunk térítőként viselkedni – amit többször is hangsúlyoztunk számukra a tábor folyamán –, ám meg akartuk mutatni, hogy a programozásban is vannak különféle irányzatok.

Az egyetemi oktatásban nyert tapasztalatok révén már tudjuk, hogy a hallgatók készek bármilyen nyakatekert szerkezetek használni, csak hogy ne kelljen eltávolodni a már elsajátított tudástól. Ezért készek megerőszakolni a programozási nyelveket, sokkal hosszabb, gyakran követhetetlen megoldásokkal előállni, mintsem átvegyék az adott rendszer filozófiáját és rövid, lényegre törő programot készíteni. A legeklatánsabb példákkal Prolog nyelven találkoztunk, ahol nincs sem hagyományos értelemben vett ciklus, sem feltételes utasítás. Ugyanez a hozzáállás – az új elsajátításának elutasítása – buktatta meg a kilencvenes években az Oberon nyelvet [1], melynek az a volt nagy bűne, hogy nem szerepelt benne `for`-ciklus, csak `while`.

Habár egyes szerzők szerint szinte bármilyen programnyelven lehet a funkcionális paradigmát követve programozni [2] [3] [4] [5] [6]; olyan nyelvet kívántam választani, melynek már a tervezésekor is ez a paradigma állt a középpontjában. Ilyen nyelv bőséggel található, a Lisp-familia igencsak tekintélyes. A Scheme elnevezésű variánst sikerült annyira leegyszerűsíteni, hogy több könyv is született azzal a céllal, hogy az olvasója elkészítse a saját Scheme variánsát [7] [8] [9], másrészt évtizedekig ezen tanulták meg a programozás alapjait a legnevesebb amerikai és európai egyetemek diákjai. A programozási nyelv és a hozzá kapcsolódó programozási környezet kiválasztásánál arra is kellett ügyelni, hogy itt középiskolás és még fiatalabb diákok fogják használni a programozói környezetet, akik nem biztos, hogy minden hibaüzenetet tökéletesen megértenek angolul, illetve nem feltétlenül a parancssor megszállottjai. Mindezeket mérlegelve esett a választásunk a Racket programozási nyelvre [10], és annak DrRacket programozási környezetére. Ez a nyelv is egy Scheme variáns, korábbi változatai még DrScheme néven futottak. Viszont pont oktatási szempontok miatt tértek el annyira a Scheme nyelvtől, hogy már illendő volt más elnevezést választani helyette.

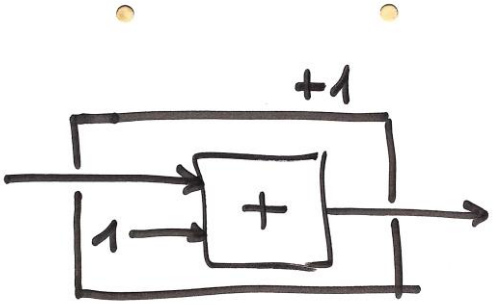
Ha valaki utánanéz ennek a nyelvnek, akkor az azzal hirdeti magát, hogy programozható programozási nyelv, azaz valójában egy keretrendszernek tekinthető, melyben mindenki elkészítheti a saját programnyelvét [11]. A rendszer telepítésekor már több tucat ilyen nyelv megjelenik a számítógépünkön, kezdve prezentációkat, dokumentációkat leíró nyelvektől (slideshow, scribble) különböző bonyolultságú, kezdőknek szánt programozási nyelvekig (htdp, sicp, regex). Egy új programozási nyelv, vagy akár csak egy DSL megalkotása is önmagában megtöltene egy nyári tábort, de ahhoz már nagyon haladó diákokra lenne szükség. Ezért erről az irányzatról, erről a lehetőségről nagyon mélyen hallgattunk. A tábor során csupán a funkcionális programozás alapjait kívántuk megmutatni, semmi többet.

Szerencsére a Racket könnyedén telepíthető, így az egyik géptermünket bő egy óra alatt elő tudtuk készíteni a táborra.

Miután a táborban 20 gyerekkel foglalkoztunk, és tíztől tizennyolc évig terjedt az életkoruk, ezekből az adatokból nem lehet statisztikailag megalapozott következtetéseket levonni, így csak tendenciákat tudunk megfogalmazni a – nem mindenki által kitöltött – kérdőívünk, valamint személyes tapasztalataink során. Miután ezekkel a diákokkal csak a tábor idejére találkozunk, és legközelebb majd csak a következő táborban, alapvetően szemléletmódot, egy más nézőpontot kívántunk átadni nekik.

## 2. Elméleti alapozás

Több olyan tantárgyat tartottunk már egyetemisták számára, ahol egy teljesen új, esetleg eltérő paradigmájú nyelvvel kellett megismertetni őket. Miután ebben az esetben sincs királyi út, elegendő időt kell számukra hagyni, hogy akklimatizálódjanak az új paradigmához, átálljon a gondolkodásuk. Ezt a periódust mi *agymosásnak* nevezzük, mert meg kell szabadulni a régi berögződésektől, hogy el lehessen sajátítani az új ismereteket, az új látásmódot.




**1. ábra:** Az eggyel növelés függvényének rajza

A [12] könyvhöz hasonlóan a funkcionális paradigma tanításakor az első géptermi gyakorlatok valójában táblás gyakorlatok, mert a gépeket nem kapcsoljuk be. *Táblán programozunk*, azaz oda rajzoljuk fel a nagyon egyszerű programjainkat. Például az összeadás egyik argumentumát fixálva elkészülhet az 1. ábrán szereplő *inc* függvény. A rajzolt függvényekkel találkozás minden egyes életkorban meglepi a diákokat, de elég gyorsan bemelegednek, és onnantól kezdve igen gyorsan tudunk haladni. Pár példa után már önállóan is elkészülnek a *programok*, és ekkor térhetünk át azok *futtatására*. A táborban kezdetben aritmetikai függvényeket kellett megfogalmazni, majd mikor ezek mindenki számára már jól mentek, ezután megismerkedtünk a lista adatszerkezettel. Természetesen ezt a tudást is az adatszerkezetekhez kapcsolódó függvényekkel mélyítettük el. Miután a lista egy rekurzív adatszerkezet, így az ezt feldolgozó függvényeknek is rekurzívaknak kell lenniük. Nyilvánvalóan ez a diákok többségének nehézséget jelentett, mert korábban a rekurzióval nem igazán találkoztak. Rendszerint ciklusok segítségével oldották meg mindazt, amire itt alapesetben rekurziót kell használunk. Arról, hogy különféle kiegészítések révén a Scheme nyelveknél is találhatóak ciklushoz hasonló szerkezetek – pontosabban lista-értelmezések – mélyen hallgattunk, mert az egyik célkitűzésünk a rekurzió megismertetése, és használatának begyakorlása volt. Ennek megfelelően nagyon sok feladatot, és annak megoldását mutattuk be, melyeknél rekurzióval igen könnyen megfogalmazható a megoldás.

További újdonságot jelentett a Lisp-típusú nyelveknél elterjedt prefix jelölés, és az a rengeteg zárójel. Miután a függvényszimbólumok és operátorok a megelőzik az argumentumaikat (prefix jelölés), a megszokott `1+2-t` `(+ 1 2)` formában kell írni. Ez a szokatlan sorrend sok panasznak volt forrása, sok időre volt szükség az átálláshoz. Viszont annyi ideig nem tartott a tábor, hogy a diákok értékelni tudják ennek a sorrendnek előnyös vonásait, például amikor elemzőt kellene írni a forrásnyelvi program feldolgozásához. A megoldások bemutatása *live coding* formájában ment, így nem egy kész programot töltöttünk be az IDE-be, és magyaráztuk el sorról-sorra, hogy mit is látnak maguk előtt; hanem ott született meg a megoldás előttük, esetleg több lépésben. A Lisp-típusú nyelvekre jellemző módon a Racket is rendelkezik egy REPL elnevezésű parancssorral, ahol zenész módjára lehet improvizálni: rövidebb hosszabb kifejezéseket kipróbálni, amit a rendszer azonnal ki is értékel. Így be tudtuk mutatni, hogy bizonyos kifejezések hogyan viselkednek, és ezekből – esetleg több lépésben – állítottuk össze a megoldást.

A vicc szerint a Lisp a *lots of irritating superflous parentheses* mozaikszava. Egy-egy függvénydefiníció végén valóban nem ritka a tucatnyi egymás mellett álló záró zárójel. De ha összehasonlítanánk egy-egy méretes C/C#/Java programot, hogy abban hány (kerek, szögletes és kapcsos) zárójel van, és a neki megfelelő Lisp/Scheme programban hány zárójel szerepel, meglepetéssel vennénk tudomásul, hogy az utóbbiban van kevesebb. Egy kis gyakorlat után, esetleg erre kihegyezett szövegszerkesztő használata esetén (paredit, parinfer, rainbow-brackets) már egyáltalán nem zavaróak a zárójelek, sőt az újabb Scheme verziók – a Racket-hez hasonlóan – megengedik a szögletes és kerek zárójelek keverését, így még nyomtatott formában is jól olvashatóak a programok. Persze egy hét a megszokáshoz kevés, a diákoknak nem vált ennyi idő alatt a kedvencévé a Łukasiewicz féle prefix jelölésrendszer. A többszintű listák ellaposításának programja így néz ki nyomtatásban, míg egy arra alkalmas szövegszerkesztőben a különböző szinten szereplő zárójelek automatikusan más színűek lesznek, illetve amikor mozgunk, folyamatosan jelzi a szövegszerkesztő a megfelelő zárójelpárt.

```
(define (flatten lista)
  (cond
    [(empty? lista) '()]
    [(list? (first lista))
     (append (p07 (first lista)) (p07 (rest lista)))]
    [else  (cons (first lista) (p07 (rest lista)))]))
```

A prefix jelölésrendszer megismerése, és egy kis jártasság megszerzése után a felrajzolt definíciókat elkezdtük Racket nyelven is megfogalmazni. Az előkészítésnek hála, ez nem okozott gondot senkinek sem. Ezzel véget is ért az első nap. A második nap a legalapvetőbb programozási szerkezeteket ismertük meg, hogy hosszabb, bonyolultabb programokat írhassunk: `if`, `cond`, `lambda`, `let`, `let*`, `letrec`, `map`, `filter` és `foldl`. Nem vettük ezek mindegyikét használatba, de bemutattuk őket, mert ez volt az, amire építkezni kívántunk a tábor során.

Miután a szó elszáll, az írás megmarad, az elhangzott elméleti ismeretek már a tábor kezdete előtt elérhetőek voltak a tábor Moodle kurzusában, így ha valakinek valami volt teljesen érthető, vagy hiányzás miatt nem hallotta, utólag elolvashatta azt. Sőt, aki többre vágyott, az további kapcsolódó anyagokra (honlapokra, könyvekre) mutató hivatkozásokat talált a Moodle kurzusban.

## 3. Gyakorlati feladatok

Miután a tábor programozói tábor volt, minden napra, minden témához egy-egy feladatsor társult, melyet a Moodle kurzusban, magyar nyelven tettünk elérhetővé. Az elméleti alapozás során a bevezető Scheme könyvek anyagából válogattunk, mint pl. [13]. Majd a *99 Prolog feladat* Lisp-re átírt változatából [14] szemezgettünk, és fogalmaztuk át a feladatokat és megoldásaikat Racket-re. Ebben a feladatsorban a listakezelési és aritmetikai feladatok voltak a központban, innen származik az előbb bemutatott program is.

Ezután komolyabb vizekre eveztünk, a Euler Projekttel [15] ismertettük meg a diákokat. Itt többnyire aritmetikai, számelméleti feladatok vannak százszámra, és regisztráció után be lehet küldeni a saját megoldásunk által generált eredményt (mint egy számot). A rendszer nyilvántartja, hogy mely feladatokat sikerült megoldanunk, ami hatalmas ösztönzés volt a diákjaink számára. Amint elkészültek egy önálló megoldással, vagy elkészültünk egy közösen elkészített megoldással, azonnal töltötték is fel az eredményeket, és boldogan fogadták a jó választ jelző hatalmas zöld pipát. Az Euler Projekt különféle szintű feladatot tartalmaz, és gyakran nem mindegy, hogy milyen programnyelven próbáljuk megoldani azt. Volt egy feladat, ahol 100 darab ötvenjegyű szám összegének a kezdete (első tíz számjegye) volt a kérdés. Ez a méret már nagyobb annál, hogy egy hagyományos

egész típusú változóban elférjen. Tehát a megoldáshoz rendszerint implementálni kell egy megfelelő adatszerkezetet, valamint ezen az adatszerkezeten az összeadást. Nem így nálunk. A parancssorba át kellett másolni a számokat, egy zárójelpárt kellett írni köréjük, majd a nyitó zárójel után be kellett szúrni egy + jelet. Ezzel el is készültünk, a rendszer már adta is vissza a megoldást. Az jó kérdés, hogy azzal, hogy megadtunk egy kifejezést, valójában programoztunk-e, vagy sem; de a kívánt eredményt minimális erőbefektetéssel megkaptuk. Hasonló meglepetést okozott a diákok számára a 1000 faktoriális kiszámítása, ami több sorból álló számot adott eredményül, szinte egy szempillantás alatt. Az ehhez szükséges program az alábbi, amely veszi *n*-ig a számokat, melyekből elhagyja a kezdő nullát, majd a megmaradt számokat mind összeszorozza. Természetesen használható a megszokott rekurzív összefüggés is, amellyel majd dupla ilyen hosszú lesz a megoldás. A Racket képességeinek hála nekünk nem kell foglalkozni a túlcsordulással.

```
(define (f n) (apply * (rest (range (add1 n)))))
```

Talán ez a pár példa néhány diáknak felnyitotta a szemét, és tudatosította, hogy nem biztos, hogy elegendő csak egy programnyelvet ismerni, és minden egyes feladatot azon megoldani, mert létezhetnek olyan rendszerek, ahol egyes dolgok egyszerűbben, könnyebben fogalmazhatóak meg.

A tábor kezdetén alapvetően mi oldottuk meg a feladatot, folyamatosan magyarázva, hogy mit miért is csinálunk, időnként táblára rajzolt szerkezetek segítségével. Ahogy haladtunk előre az időben, egyre gyakrabban már a diákok mutatták meg saját megoldásaikat, melyeket átbeszéltünk, hogyan lehetne esetleg még javítani rajta.

A tábor utolsó napján – egyfajta kitekintésként – a 2048 játék [16] egy Racket nyelvű megoldását mutattuk be lépésről lépésre, rámutatva, hogy a grafikus felület kezelése a programkód szinte elenyésző részét képezi. A program túlnyomó részét a játék szabályainak megfogalmazása, az adatszerkezetek definiálása, a hozzá kapcsolódó függvények elkészítése tette ki. A tábor utolsó óráiban – kifogyva az előkészített feladatokból – túlléptünk a korábban megszabott korlátokon és egy prímszámokhoz kapcsolódó feladatot próbáltunk megoldani. Ebben az esetben fontos volt a korábban kiszámolt prímszámokhoz való visszatérés, azaz az azokat tároló adatszerkezet. Miután a lista elérése lineáris, hosszú listák esetén ez jelentős lassulást okoz. Ezért megmutattuk a asszociatív tömbök (hash) használatát, ami már kicsit körülményesebb, mint az addig látottak; de nem kívántuk azt a hamis látszatot kelteni, hogy ebben a nyelvben minden egyszerűen megoldható.

## 4. Vélemények, tapasztalatok

Fontos körülmény, hogy a tábor teljes mértékben ingyenes volt a diákok számára. Emiatt néhány diák lemorzsolódott a hét során. Kicsit más lett volna a helyzet, ha jelentős összeget kellett volna fizetni a szülőknek, mert akkor valószínűleg futottak volna a pénzük után, és nem engedik, hogy a gyerekük ellógjon egy napot, vagy napokat. Viszont több mint kilencven százaléka a diákoknak kitartott, és néhányuk csak az iskolai évnyitó miatt fejezte be az utolsó napot korábban. A többiek még péntek délután négykor is ott verték a billentyűzetet, és gépelték a Racket nyelvű programokat. Úgy véljük, hogy ez a legnagyobb dicséret.

A feladatokat próbáltuk úgy megfogalmazni, hogy egymásra épüljenek, időnként visszatértünk egy korábbi megoldáshoz, és azt kellett továbbfejleszteni, hogy megoldjunk egy későbbi feladatot. Viszont volt olyan diák, melynek az új feladat kitűzésekor már kész volt a megoldása, mert a korábbi feladat megoldása során nem elégedett meg a minimális megoldással, próbálta továbbgondolni, új feladatot tűzött ki saját maga részére, és meg is oldotta. A tábor Benjaminja hasonló módon elkalandozott, és megszállottan próbált egy általa kitalált – az általunk megoldott feladatoktól teljesen eltérő – feladatot megoldani. Az egyik szünetben – pár speciális programozási szerkezettel megismertetve – sikerült begyújtani a rakétáit, és még megszállottabban haladt a saját feladata megoldása felé.

A *generációs* különbségek természetesen itt is kiütköztek. Az általános iskolások jellemzően szívesen begépelték a kivetítőn megjelenő teljes megoldásokat, de ha csak hiányos megoldással találkoztak, akkor vártak volna a sült galambra. Ilyenkor mindenkihez külön-külön leülve, átbeszélve a megoldandó feladatot rendszerint már képesek voltak kitölteni a hiányzó részeket. Meglepő módon esetükben a hét túlnyomó részében sikerült megküzdeni a mobil és a netes játékok csábításával, csak a napok végén, illetve az utolsó napon – mikorra szellemileg már teljesen elfáradtak – választották a kikapcsolódásnak ezt a formáját. A középiskolások viszont a tempónkat jól bírták, gyakran még egymással is versenyeztek, hogy ki milyen gyorsan tud megoldani egyes feladatokat, illetve érdeklődéssel nézték egymás programjait, hogy kinek milyen eszközt használva sikerült megoldani a feladatot.

Miután ez volt az első ilyen táborunk, a jövő nyári tábor sikeressége céljából elkészítettünk egy kérdőívet, melyben anonim módon megfogalmazhatták a véleményüket a táborozók. Sajnos kevesebb, mint a diákok fele válaszolt, éppen a kemény mag. Ők elégedettek voltak a táborral, tetszett nekik a választott programozási nyelv, szívesen jönnének jövőre is; és úgy néz ki, hogy sikerült megfertőzni őket, mert egy megszokott és széles körben ismert programnyelvbeli tudásuk elmélyítése helyett egy újabb, a fősodortól távoli nyelvet szeretnének megismerni. Ezzel rendesen feladták a leckét számunkra, mert valami újabb, különc paradigmát kellene keresni számukra. Hiányosságként a szünetek száma és terjedelme lett megfogalmazva, amit meg kell szívlelnünk; valamint az étkezést érte még panasz (nem a minőséget, nem a mennyiséget, hanem inkább a szerkezetét), amin szintén könnyen lehet javítani.

## 5. Összefoglalás

Először szerveztünk most nyáron haladó programozói tábort a Debreceni Egyetemen. Emiatt volt bennünk egy kis bizonytalanság, mert ismeretlen terepre tévedtünk. Ezt tetéztük azzal, hogy a szervezés megkönnyítése érdekében egy, a diákok számára ismeretlen nyelvet választottunk, melyben azért mi sem voltunk teljesen otthonosak. Mindenesetre a felkészülés során igen sok feladatot lefordítottunk magyarra és meg is oldottuk, majd megosztottuk a feladat kiírását és a megoldását a tábor Moodle kurzusában. Ezzel sikerült olyan gyakorlatot szerezni, mellyel a tábor megtartása zökkenőmentes volt.

Mint a diákok tábor során nyújtott teljesítménye, mint a kitöltött kérdőívek alapján sikeresnek tekinthető a tábor, úgy érezzük sikerült egy új világot kinyitni előttük. Ezzel a saját magunk számára kitűzött célunkat elértük. Ennek megfelelően a soron következő tábort ebben a szellemben kívánjuk megszervezni jövőre, s már csak egy kérdésre kell választ találni addig, hogy mi legyen az az egzotikus nyelv, mellyel jövőre megtámadjuk a diákokat?

## Irodalom


1. M. Reiser és N. Wirth: *Programming in Oberon: Steps Beyond Pascal and Modula*, Addison-Wesley (1992)
2. P.-Y. Saumont: *Functional Programming in Java: How functional techniques improve your Java programs*, Manning Publications (2017)
3. I. Cukic: *Functional Programming in C++: How to improve your C++ programs using functional techniques*, Manning Publications (2018)
4. L. Sheehan: *Learning Functional Programming in Go: Change the way you approach your applications using functional programming in Go*, Packt Publishing (2017)
5. S. F. Lott: *Functional Python Programming: Discover the power of functional programming, generator functions, lazy evaluation, the built-in itertools library, and monads*, Packt Publishing (2018)

6. P. Hartel, H. Muller: *Functional C*, Addison-Wesley (1997)

7. C. Queinnec: *Lisp in Small Pieces*, Cambridge University Press (2003)

8. N. M. Holm: *Compiling Lambda Calculus*, Lulu Press (2018)

9. N. M. Holm: *Scheme 9 from Empty Space*, Lulu Press (2014)

10. PLT Research Group: *Racket.* (2019)
http://racket-lang.org. (utoljára megtekintve: 2019.9.15.)

11. M. Butterick: *Beautiful Racket*, szerzői kiadás (2019)
https://beautifulracket.com.

12. D. S. Touretzky: *Common LISP: A Gentle Introduction to Symbolic Computation*, Dover Publications (2013/1989).

13. B. Harvey, M. Wright: *Simply Scheme: Introducing Computer Science*, MIT Press (1999)

14. J. Meidanis: *L-99: Ninety-Nine Lisp Problems*, (2006)
https://www.ic.unicamp.br/~meidanis/courses/ mc336/2006s2/funcional/L-99_Ninety-Nine_Lisp_Problems.html (utoljára megtekintve: 2019.8.6.)

15. *Project Euler* (2019)
https://projecteuler.net/ (utoljára megtekintve: 2019.8.25.)

16. *2048*, (2019)
https://2048game.com/ (utoljára megtekintve: 2019.5.10.)

# Diákközpontú oktatás

Bakonyi Viktória[1], Illés Zoltán[2]

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

**Absztrakt.** A XXI. század fiataljai már hozzászoktak a valós idejű informálódási lehetőségekhez, amelyet az őket körülvevő modern technika nyújt számukra. Amellett, hogy rendelkezésükre áll az interneten található ismeretek szinte végtelen tárháza, felismerték és elsődlegessé vált számukra a tudás megosztása. Míg korábban az egyén, ma a közösség tudásán van a hangsúly. Nem engedhetjük meg magunknak, hogy figyelmen kívül hagyjuk ezt a szükségletet. Előadásunkban néhány olyan szoftvert mutatunk be, amelyek ezt a munkát segíthetik a tapasztalataink alapján.

**Kulcsszavak:** valós idejű, smart eszköz, szavazatszámláló rendszer, CRS, felügyelő és menedzselő alkalmazás, oktatás, informatika

## 1. Bevezető

A pályán lévő tanároktól egyre gyakrabban hallani, hogy a jelenleg az iskolapadban ülők radikálisan máshogy viselkednek akárcsak a néhány évvel korábbi diákokhoz hasonlítva is. Kevesebb ideig tudnak figyelni ugyanarra a dologra, hiszen ahhoz szoktak hozzá, hogy egyszerre több készülék is a kezük ügyében van – ezt a jelenséget a szakirodalom **hiper-figyelemn**ek [1] hívja, ami pedig az elmélyülést kívánó tanulmányok kárára vannak. Sokan képtelenek elszakadni még rövid időre is a mobil eszközüktől és ezen keresztül a közösségi hálók által kínált színes információáradattól. (https://vip-phone.hu/phonedorlat-blog/fuggoseg-eredmenyek) Valljuk be az internet gazdagságával, a virtuális világ csodáival a hagyományos iskola nem tudja felvenni a versenyt, a gyerekek motivációja érezhetően lecsökkent.

Próbálkozhatunk azzal, hogy kitiltjuk az iskolákból a XXI századi eszközöket, de ez a lépés nem kecsegtet túl nagy sikerrel – hiszen az iskola kapuján kívül az ő világuk erről szól! A mai diákok már valamennyien digitális bennszülöttek, nekik a modern technológiák használata már a megszokott, sőt elvárt környezetet jelenti. *„Ha a hegy nem megy Mohamedhez, Mohamed megy a hegyhez!*" Ha a gyerekeket nem köti le a hagyományos oktatás, az oktatási módszereket kell megváltoztatni és közelebb vinni az elvárásaikhoz.

Egyre több oktató vallja, sőt a gyakorlatban alkalmazza, hogy modernizálni kell az oktatást, felhasználva a digitális eszközöket is a gyerekek érdeklődésének a felkeltésére, hogy újra élettel teljen meg az együtt töltött idő és minél nagyobb hozadéka legyen az új lehetőségeknek. Ezt a munkát hivatott elősegíteni a Digitális Pedagógiai Módszertani Központ (https://dpmk.hu/), illetve a 2012-től évente megrendezett Digitális Pedagógus konferencia sorozat, ahol a legjobb gyakorlatokat, ötleteket is megoszthatják egymással a pedagógusok. [2]

Az eszközhasználat azonban nem minden: „*Természetesen ehhez nem elég, ha bevisszük az óráinkra és használjuk a legújabb IT lehetőségeket. Ha a tananyag nem jó, ha a tanár nem felkészült, ha a módszertani eszköztár szegényes, ha az óra nem tanulócentrikus, ha a tanulók nagyrészt passzívak, akkor jöhet bármilyen app vagy tablet…"* [3]

## 2. Tanulócentrikus szemlélet, motiváció

„*A megfelelő motiváló erő nélkül szinte képtelenség rávennünk magukat a tanulásra, legyen szó bármilyen érdekes tananyagról, bármelyik életkorban… Nem csupán a gyermek céljai, a környezete és a pedagógus, de a tanuló maga, az adott tanagyag, valamint a tanítás módszere is befolyásolja a motiváltsági szintet.*" [4]

A kérdés az marad, hogyan tudunk javítani a meglevő motiváltsági szinten? Melyek azok a főbb elemek, amelyek meghatározzák ezt?

- A tanár szerepe, tudása, módszerei, lelkesedése, reflexiós képessége
- A tantárgy maga – mennyire áll közel a diák egyéni céljaihoz, gyakorlati élethez való közelsége
- A diák tanulási stílusa, ambíciói, céljai
- A megfelelő iskolai és otthoni tanulási környezet (az otthoni elvárásoktól a digitális lehetőségek eléréséig)

Az első három nem tartozik az iskola hatáskörébe. Időzzünk is el egy pillanatig az utolsó pontnál: az iskolai és otthoni tanulási környezetnél! Megszívlelendő a következő idézet: *"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) (Az idézet szabad fordításban a következő*: A tanulók a nap 18 órájában XXI. századi környezetben élnek, de az iskolában eltöltött 6 óra alatt XIX. századi körülmények közé kerülnek – ezt szörnyű ellentmondásként élik meg.*)

Végezzünk egy gyors összehasonlítást az iskolai és az otthoni tanulási környezet, lehetőségek között – természetesen erősen karikírozva mindkét oldalt, hogy még szemléletesebb legyen a különbség! Természetesen ma már jónéhány iskola rendelkezik modern digitális eszközökkel – és messze nem mondható el, hogy minden diák ideális otthoni tanulási környezetben tud fejlődni.

Egyáltalán nem lehet csodálkozni az 1. táblázat áttekintése után, hogy a diákok érdeklődése drasztikusan megcsappant az utóbbi időben!

| Tulajdonságok | Klasszikus iskolai | Otthoni környezet |
|---|---|---|
| Fizikai környezet | Előadóterem, osztályterem, kivetítő és tábla. | Körül van véve az okos telefonnal, tablettel, okos órával, TV-vel stb. Színes multimédia folyam. |
| Tanulás kötöttsége | Megadott időben, helyen, tematikával – iskola határozza meg, mindenkinek egyformán | Bármikor, bárhonnan, amiről csak akar, *azonnal* |
| Eszközök | Tilos a saját eszközök használata | Szinte „hozzánőttek" az okos eszközeikhez |
| Használt IT technológia | Nem túl modern eszközök | Legmodernebb eszközök |
| A kommunikáció típusa | Klasszikus órai kommunikáció, figyeli a tanári magyarázatot, vagy felel. | Jórészt a virtuális világban zajlik *valós időben*, de üzeneteket is lehet hagyni. Térben és időben kötetlen. |
| Tudás forrása | A tanár vagy egy másik diák. Egy időben egyszerre csak egy. | Pl. Szociális hálókon keresztül bárki, bárhonnan… Közösségi tudás, a *tudás megosztása* |

| | | |
|---|---|---|
| A kommunikáció iránya | Általában egyirányú, a tanár beszél. A tanár irányít. | Akár egyidőben több párhuzamos kétirányú kommunikáció. Egyenrangú felek. |
| Figyelem | Ugyanarra hosszú ideig | Párhuzamosan több mindenre figyel [1] |

**1. táblázat:** Az iskolai és az otthoni tanulási körülmények összehasonlítása

Láthatjuk, hogy az *azonnali reakciók lehetősége, a tudás megosztása, a modern eszközök használata* kiemelkedően fontos lenne, hiszen ehhez szokott a XXI. századi hallgatóság! Ezután evidensnek mondható, hogy érdemes új lehetőségeket is keresni, amelyekkel a diákok elvárásaihoz közelebb vihető az iskolai élet. Az otthoni körülményeket, környezetet leginkább az interaktivitást, a közös munkát segítő eszközökkel biztosíthatjuk. Tehát az a jól használható eszköz, ami az órai munkát is közös gondolkodássá alakítja, ezt segíti.

# 3. Szavazórendszerek

A szavazórendszerek először a marketing, a vásárlói preferenciák feltérképezése területén jelentek meg. A vállalati, közösségi eseményeken tapasztalt sikeres használat eredményeként, ahogy minden más új eszköz esetében is, ezek hamarosan bekerültek az oktatás eszköztárába is (az oktatásban CRS – Classroom Response System). [5][6][7] Mára az okos telefonok elterjedésével elkerülhető a drága clicker rendszerek megvásárlása. A BYOD (Bring Your Own Device) felhasználásával már lehetséges ezen lehetőségek bevezetése egyetemeken [8][9], a középiskolákban, sőt az általános iskolákban is, hiszen ezek az erőfeszítések nem hiábavalók, aktivizálják a diákokat.

Nézzük meg újra a kiindulási 1. táblázatunkat, hogy mely pontjainak felel meg egy CRS rendszer, hol lehet közelíteni használatukkal a ma elvárt környezet felé az egyetemi oktatást! Soroljuk fel az ott megemlített fontosabb szempontokat:

- használhatja a saját, megszokott eszközét (nem tiltjuk, inkább bevonjuk az oktatási térbe)
- nem egyirányú a kommunikáció (nemcsak hallgatnia kell az előadást), kérdezhet, véleményt nyilváníthat *azonnal*, amely az előadás menetét befolyásolhatja
- *valós idejű* kommunikáció történik, azonnali visszajelzés történik
- együtt gondolkodás a többiekkel, a tudás megosztása – látva a *valós idejű* visszajelzéseket

Számtalan kész alkalmazás van a piacon, amelyik CRS tulajdonságokkal rendelkezik, ilyen például a Sli.do is, amelynek free változatával bárki kipróbálhatja a szavazórendszerek használatát. [5]

## 3.1. A szavazórendszerek elfogadottsága, felmérés

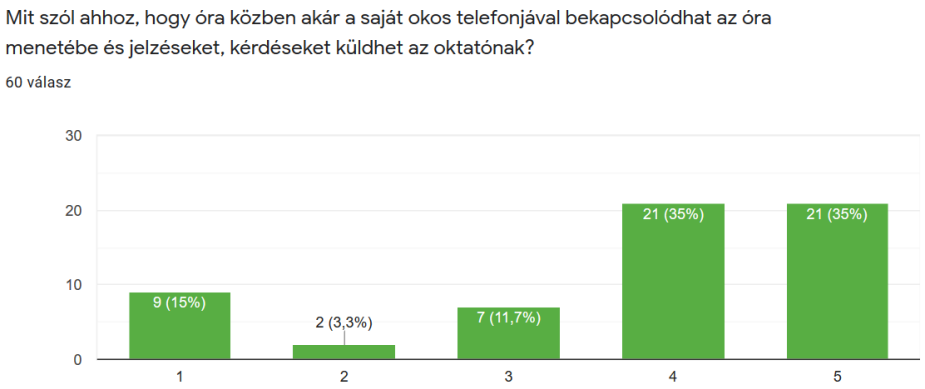
Többek között mértük a CRS rendszerek elfogadottságát a hallgatók körében egy anonimitást biztosító Google Űrlap segítségével. (https://bit.ly/2CDsn10)

A 2019 tavaszi félévben az *Operációs rendszert*, a keresztféléves *Számítógépes rendszereket* és a *Számítógépes problémamegoldást* hallgató tanulók kaptak felkérést ennek kitöltésére. A kitöltés nem volt kötelező. 60 válasz érkezett a magyar nyelvű űrlapra (a külföldi hallgatók számára ugyanez angol nyelven volt elérhető). (A *Számítógépes rendszerek* tárgyat a tavaszi félévben hallgatók többsége ismétlő.)

A vonatkozó kérdés a következő volt, amit 1-5 skálán értékelhettek a hallgatók:

*Mit szól ahhoz, hogy óra közben akár a saját okos telefonjával bekapcsolódhat az óra menetébe és jelzéseket, kérdéseket küldhet az oktatónak?*

Az 1. ábrán jól látható, hogy a hallgatók 70 %-a pozitívan fogadja a lehetőséget (4-5) és csak 15 % utasítja el erősen (1-2)




**1. ábra:** A CRS elfogadottsága

## 3.2. Sli.do

A Sli.do egy olyan általános célú szavazórendszer, amelyet konferenciákon, vállalati továbbképzéseken és az oktatásban is használnak. Az alapötlete éppen egy egyetemi oktatóhoz kötődik, aki hallgatói véleményeket gyűjtött (https://bit.ly/2r9cnB1). Bárki használhatja tanárként, még regisztrálni sem kell, elég gmail-es azonosítást választani. Ezentúl az eszköz tudását tekintve az ingyenesen elérhető kategóriában (pl. Kahoot, Voxvote, Socrates stb.) gyakorlatilag a legjobbak között van, így mi is ezt vizsgáljuk meg közelebbről.

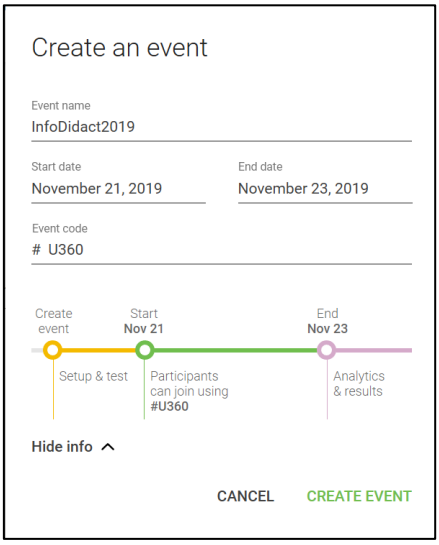
A használatához létre kell hozni egy eseményt, megadva a kezdő és végdátumot, ami alatt a hallgatóság kapcsolódhat az eseményhez lásd 2. ábra.

A hallgatóság egy az esemény létrehozásánál generált kóddal tud csatlakozni. Az ingyenes verzióban csak publikus szobák vannak, ahol a felhasználók anonim módon kapcsolódnak.

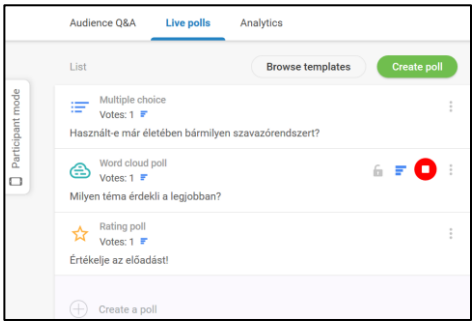
A kommunikáció kétirányú: a hallgatóság is kezdeményezhet kérdés küldést, illetve az előadó is szavazatra bocsáthat valamit, amire az eseményre felcsatlakozott hallgatóság a saját készülékét használva válaszolhat.

a) Nézzük először az előadó lehetőségeit: többszörös választás, szófelhő, kvíz, szabad szöveg, értékelés. A rendszer ehhez megfelelő sablonokat is ad, sőt az azonnali kipróbálást is felajánlja. Az ingyenes verziónál egy eseményhez három kérdést készíthetünk.
   - Az esemény előtt előkészíthetjük a kérdéseinket (lásd 3. ábra) és tesztelhetjük,
     - Egyszerre láthatjuk a kérdés teszt aktivizálásakor az előadói és egy másik ablakban a hallgatói mobil telefon nézetet
     - Küldhetünk teszt adatokat a fiktív hallgatói eszközről
   - Az esemény alatt valós időben követhetjük a kommunikációt,
     - A kérdés típusának megfelelő sablonok alapján prezentál (lásd 4. ábra)
   - Majd az esemény befejeztével az eredményeket analizálhatjuk, exportálhatjuk.

b) Legalább ilyen érdekes a másik irány, ahol a hallgatóság szabad szövegű kérdéseket küldhet az esemény teljes időtartamában. Az ingyenes verzióban nincs moderálási lehetőség.
   - Az alkalmazás itt is teljeskörű kipróbálási lehetőséget nyújt előadói és hallgatói nézettel együtt.
   - A hallgatóság a többiek által feltett kérdéseket is látják a készülékükön, lájkolhatják is azokat.




**2. ábra:** Esemény létrehozása




**3. ábra:** Különböző típusú kérdések


**4. ábra:** A szófelhőben ábrázolt válaszok

A Sli.do egy rendkívül tetszetős, professzionális elkészített alkalmazás, amelynek azonban megvannak az oktatásban a korlátjai:

- Ilyen probléma az, hogy az anonim módon, moderátor nélkül küldött kérdések, amelyeket mindenki más is láthat igencsak rosszul is elsülhet - bármely órát, különösen előadást teljesen szétzilálhat.
- Hiába vennénk azonban meg a moderálási lehetőséget, aligha van jelen az órán még egy ember, aki a moderálási feladatokat fel tudná vállalni, ahogy az minden eseten történik egy nagyobb konferencián. A hallgatóság felől érkező kérdések kikapcsolhatók – így ez a veszély elhárul, bár a kommunikáció egyik irányát teljesen eltávolítja.
- A fizetős változatoknál az azonosítás is megoldható, ami egy értékelésnél fontos lenne, ez viszont aránytalan adminisztrációs terhet róhat az előadóra, hiszen neki kellene a konkrét azonosítókat létrehoznia, kiosztania.
- A fizetős verzió sem nyújtja a kérdésbank összeállításának lehetőségét.

## 4. Terem menedzselő és vezérlő rendszerek

Az interaktivitást segítő rendszerek, mint például a korábban tárgyalt Sli.do előadás jellegű események lebonyolítását segíti elsősorban. Természetesen egy iskolai gyakorlati óra segítésében is hasznos tud lenni, de erre a feladatra jobban illeszkednek a számítógép felügyelő, menedzselő rendszerek. Az informatika órákon vagy más olyan órán, ahol számítógép előtt ülnek a diákok, bevethető egy ilyen rendszer is.

Egy ilyen rendszer egyfelől arra alkalmas, hogy a tanár átvegye a vezérlést a diák gépe felett, bekapcsolhatja, lekapcsolhatja azokat, befagyaszthatja a működésüket, megakadályozva azt, hogy elkalandozzon a gyerekek figyelme. Ugyanakkor kiküldheti a képernyőkre az általa kívánt tartalmat, üzenetet vagy akár megnézheti bárkinek a munkáját valós időben és kivetítőre kapcsolva közzé teheti azt.

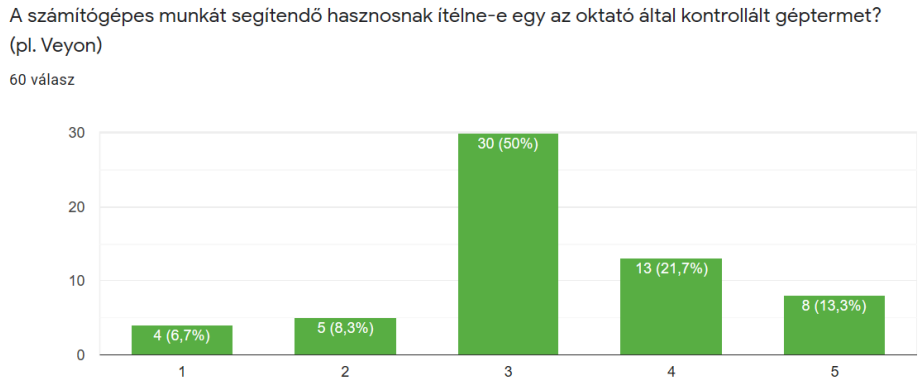
Ha visszapillantunk az 1. Táblázatra az abban foglaltakból teljesül néhány elvárás, bár kevesebb, mint a szavazórendszerek esetében.

- Valós idejű kommunikáció – igaz, csak egyirányú, egyszerre egy személlyel
- Tudás megosztása a közösségen belül – ez hatványozottabban így van, mint a CRS-nél.

## 4.1. A menedzselő és vezérlő rendszerek elfogadottsága, felmérés

A 2.2.1. pontban már említett felmérésben rákérdeztünk a felhasználók osztálytermi vezérlő rendszerekkel kapcsolatos véleményére is. A vonatkozó kérdés a következő volt:

*A számítógépes munkát segítendő hasznosnak ítélne-e egy az oktató által kontrollált géptermet?* (pl. Veyon)
Az értékelés 1-5 közötti volt. Az eredmény az 5. ábrán látszódik.

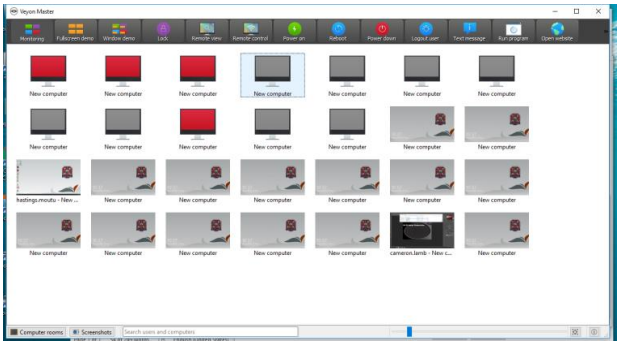



**5. ábra**

Az eredmény már nem olyan fényes – látszik, hogy kevésbé fogadják el ezt a fajta megoldást. Nyilván az ellenérzés egyik oka az, hogy lehetőséget ad a tanárnak a korlátozásra és a folyamatos ellenőrzésre és ez nem mindenkinek szimpatikus. Okosan használva azonban a tanári munkában nagy segítség lehet.

## 4.2. Veyon

Ajánlani tudjuk az ingyenesen használható Veyon alkalmazást, amely erre a célra való lásd 6. ábra. A Sli.do-val szemben itt telepítésre van szükség, hiszen a tanári géphez csatlakoztatni kell a tanulói gépeket. A telepítés után viszont minden korábban említett lehetőség fentáll, vagyis az oktató a szoftver segítségével teljesen ellenőrzése alatt tarthatja az összes kapcsolódó számítógépet.



**6. ábra:** Veyon tanári kezelő felület

Tapasztalatból állíthatjuk, hogy „tapintatos” használat után a hallgatói ellenérzések csökkennek.

## 5. Összegzés

Ahogy az élet számos területe, az oktatás sem tudja magát kivonni a környezet, a technológia hatások alól. Ha hatékonyan, sőt jól akarunk tanítani, folyamatosan változnunk kell a hallgatói igényeknek megfelelően. Napjainkban a technológiai újítások az emberek közötti kommunikációs módokat szinte teljesen átrajzolták. A valós idejűség, az azonnali kétirányú kommunikáció a fókuszba került, növelve az órai interaktivitást. A szociális hálók népszerűségének köszönhetően a tudásmegosztás is központi szerepet kapott a mindennapi élet apró-cseprő problémáinak megoldásában is – fel kell használnunk az oktatásban is ezeket a trendeket. Cikkünkben két olyan alkalmazástípust mutattunk be, amelyekkel közelebb kerülhetünk a hallgatósághoz.

## Irodalom


1. Dani, E.: *E-létezés és „hiperfigyelem*”, In: Könyv és Nevelés, 2013/4 (Online) http://bit.ly/1OsFALM (utoljára megtekintve: 2019.10.30.).
2. *Digitális Pedagógus*. http://digitalispedagogus.hu/ (utoljára megtekintve: 2019.10.31)
3. Főző Attila: *A Kahoot jelen*ség. https://komposzt.wordpress.com/2018/07/23/a-kahoot-jelenseg/
4. *Miért nem tanulsz.* https://mindsetpszichologia.hu/2018/08/23/miert-nem-tanulsz-a-tanulasi-motivacio/ (utoljára megtekintve: 2019.10.31)
5. Heizlerné. Bakonyi Viktória, Ifj. Illés Zoltán, Illés Zoltán, *Valós idejű oktatást segítő rendszerek*. InfoDidact 2017 konferencia, Zamárdi, https://people.inf.elte.hu/szlavi/InfoDidact17/betolt.html (utoljára megtekintve: 2019.10.31)
6. J. L. Brown, *Quick, click: Student response systems evolve in higher ed, New student response systems offer increased versatility"* University Business, November 2016 http://bit.ly/2fnJMRw (utoljára megtekintve: 2019.10.31.)
7. Dangel, H. L. & Wang, C. X. (2008). *Student response systems in higher education: Moving beyond linear teaching and surface learning***.** Journal of Educational Technology Development and Exchange, 1(1), 93-104. http://www.sicet.org/journals/jetde/jetde08/paper08.pdf (utoljára megtekintve: 2019.10.31.)
8. R. Žitný, T. Szabó, I. Pšenáková, Z. Illés and V. H. Bakonyi, "*Education Using Mobile Technologies*". ICETA2016.11.24-25. Stary Smokovec, pp. 387-393. IEEE, ISBN:9781509046997
9. H. Bakonyi Viktória, Illés Zoltán: "*Real-Time Tool Integration for Lectures*", 15th IEEE International Conference on Emerging eLearning Technologies and Applications: ICETA 2017.Conference place and time: Starý Smokovec, Slovakia, 2017.10.26-2017.10.27. Denver: IEEE Computer Society Press, 2017. pp. 31-36. (ISBN:978-1-5386-3294-9)

# Smart eszközök a tanórákon

Bakonyi Viktória[1], Illés Zoltán[2], Pšenáková Ildikó[3], Heizler Adina[4]

{[1]hbv, [2]illes}@inf.elte.hu, [3]ildiko.psenakova@truni.sk,
[4]adina.heizler@hotmail.com
ELTE IK, Trnavská univerzita v Trnave, ELTE TÓK

**Absztrakt.** Napjainkban az iskolai oktatásban is megjelentek az okos eszközök. Egyre több helyen van okos tábla és már egyre fiatalabb diákok is rendelkeznek okostelefonnal. A feladat már csak az, hogy ezeket az eszközöket úgy használjuk fel, hogy minél változatosabb és interaktívabb módon vonjuk be a gyerekeket a tanítási órákba, a közös munkába. Oldjunk meg feladatokat az okostáblán, tanuljunk algoritmikusan gondolkodni vagy akár programozni a telefonunkkal! Az előadás során bemutatunk néhány olyan lehetőséget, amely sikeres lehet már a kisebbek körében is.

**Kulcsszavak:** okostelefon, okostábla, algoritmizálás, programozás

## 1. Bevezető

1977-ben Ken Olson a DEC alapítója és elnöke mondta: *"Nincs ok, amiért bárki akarna egy számítógépet az otthonába."* (https://bit.ly/32yvAsO) Nem is tévedhetett volna nagyobbat! 5 éven belül megjelent a Zx Spectrum és hódító útjára indult. 1990-ben elindult a hazai mobil szolgáltatás, mára már az okostelefonoké a főszerep – 5.3 millió okostelefon felhasználót tartanak nyilván (https://bit.ly/2p23Vmt) és egyre nő a Microbit, a Raspberry vagy az Arduino-t használók tábora is.

Miért **lettek** ezek az eszközök ilyen népszerűek? Nyilvánvaló, hogy mind a számítógépekkel, mind pedig az okostelefonokkal egy olyan eszközt kapott kezébe az emberiség, amely a hétköznapi feladatok megoldásának megkönnyítésére, információ szerzésre, kommunikációra és szórakozásra is páratlan lehetőségeket ad. Ma már sokan függőségről beszélnek – vannak, akiket annyira magába szippant ez a virtuális világ, amely már káros is lehet. (https://bit.ly/36OMdnx) No, de mi a helyzet az általános és középiskolákban? Van, ahol tiltják ezeknek a használatát mondván, hogy a tanulók nem figyelnek, hanem inkább játszanak. Mások inkább amellett kardoskodnak, hogy vonjuk be a számítógépeket, okos eszközöket a tanulásba.

*„Eszerint a technika kétféleképpen jelenhet meg az iskolában vagy bővíti vagy átalakítja a tanítás menetét. Mindegyiknek két szintje képzelhető el: a bővítés esetében a helyettesítés és a kiterjesztés, az átalakításnál a módosítás és az átértelmezés."* (tanárblog: https://bit.ly/34Jqrj7) Ennek a gondolatnak a vezérfonalát követve tekintjük át cikkünkben az iskolában elterjedt vagy elterjedőben lévő eszközök, alkalmazások hatását.

## 2. Számítógép

A magyar közoktatás abban a szerencsés helyzetben van, hogy évtizedek óta van informatika óra az iskolákban – az azonban erősen vitatott a szakemberek körében, hogy ennek mennyisége elegendő-e. Az órákon főleg az irodai szoftverekkel, illetve az adat modellezés és a programozás alapjaival foglalkoznak a gyerekek. Elkötelezett tanárok a tehetséges diákjaikat szakkörökön különböző versenyekre készítik fel: ilyen például az e-Hód, az Országos Logo Verseny és a Nemes Tihamér Informatikai Verseny is [1]. A számítógépek használata azonban nemcsak az informatika órákon jelenik meg. A mindig megújulni képes, tapasztalt oktatói gárda mellett, mára már felnőtt és munkába állt egy olyan tanári generáció, amely digitális bennszülöttként, értőn és magától értetődően használja az IT eszközöket a mindennapokban. Természetes, hogy az iskolai munkában is megjelentek az info-

kommunikációs eszközök – *módszertani bővülést* is eredményezve. (Ezt a folyamatosan fejlődő területet segíti a Digitális Pedagógiai Módszertani Központ (https://dpmk.hu/) munkája.) Nem számít kuriózumnak, ha gyerekeinknek házi feladatként irodalomból blogot kell írniuk, történelemből a facebook idővonalát felhasználva kell végezni a gyűjtőmunkát, ha a geometria feladatot a Geogebra alkalmazással kell megoldani vagy ha a tanár a gyerekekkel és a szülőkkel az interneten tartja a kapcsolatot. A különböző játékos oktató programok élményszerűvé tehetik az egyébként könnyen unalmassá váló gyakorlást is (https://bit.ly/32w5OWb).

## 3. Okostábla

Egyre növekszik azoknak a tanároknak a száma, akik a technika segítségével akarják bevonni diákjaikat a tanulásba. Egyre gyakrabban használják az oktatásban az okos interaktív táblát vagy asztalt, amely lehetővé teszi, hogy az interaktív alkalmazások segítségével kisebb és nagyobb diák csoportok közös tanulási élményben részesüljenek. A tanár és a tanulók is együtt dolgozhatnak a táblánál, miközben a többiek a számítógépeknél tevékenykednek és a csoport közösen megbeszélheti a felmerülő problémákat. Az azonnali interakció és a beépített multimédiás lehetőségek olyan újfajta közös munkára adnak lehetőséget, amely ezek nélkül az eszközök nélkül nem voltak lehetségesek. *Bővítették* a csoportos munka szervezését, segítik a felfedeztető tanítást és a diákok aktivizálásával majd minden esetben *helyettesíthetik* a frontális módszert.

Feltételezzük, hogy az interaktív tábla fogalmát már nem szükséges magyarázni. Még ha van is olyan tanár, aki a gyakorlatban nem használja, szinte biztos, hogy már legalább hallott róla. Azok a kollégák, akik már használtak vagy használják jelenleg is az interaktív táblákat, reméljük, hogy egyetértenek velünk, hogy megfelelő tervezéssel, felkészüléssel és használattal ez egy nagyon hatékony oktatási eszköz. A tábla segítségével sokkal könnyebb felhívni és fenntartani a diákok figyelmét, sokkal érdekesebben, látványosabban lehet a tananyagot bemutatni és hatékonyabbá tenni a magyarázatát. Az interaktív tábla használható elméleti előadásokon és gyakorlati tanórákon is és a bemutatott tartalom nehézsége szinte minden korosztályhoz alakítható és alkalmazható. Az okos tábla használatával a tanórák szórakoztatóbbak lehetnek, a diákok és pedagógusok egyaránt motiváltabbá, az órák pedig érdekesebbé válhatnak [2][3].

Az általános iskolai tanítóképzésben is szerepet kapott az okostáblák használata mind a magyar, mind pedig a szlovák felsőoktatásban, sőt az ezekre való alkalmazások készítése is a képzés része a Trnavai Egyetemen.

### 3.1 Alkalmazások készítése

A Nagyszombati Egyetem Pedagógiai Kar tanítóképző szakon az interaktív tananyag készítését a Hot Potatoes szoftver segítségével oktatjuk. Ennek elsősorban az az oka, hogy ez egy szabadon elérhető és ingyen letölthető szoftver (http://web.uvic.ca/hrd/hotpot/), amely akkor is használható, ha nincs összeköttetésben a számítógép a táblával. Ez megadja a lehetőséget arra, hogy a diákok otthon is létre tudnak hozni interaktív tananyagot és felkészülni a tanteremi órákra. A szoftvernek ezt a tulajdonságát használhatják ki később a gyakorlatban, illetve a ma már gyakorló pedagógusok is, mert a szükséges anyagokat a táblától függetlenül tudják előkészíteni.

Másodsorban a szoftver használata olyan egyszerű, hogy a diákok nagyon gyorsan elsajátítják a kezelését. Saját feladatokat készítenek, miközben nemcsak a szoftver kezelését tanulják, hanem ismétlik a már megtanult tananyagot, analizálják tudásukat és így tökéletesítik tanulási képességeiket.

A Hot Potatoes öt részből áll, melyek különböző típusú feladatok elkészítésére szolgálnak. Ezek lehetnek különállóak vagy tömbösített egymással összefüggő feladatsorok. Ezek elkészítésére a The Masher program rész szolgál, amely a különálló részekből létrehoz egy .html formátumú web oldalt.

Ennek segítségével a feladatsorok elhelyezhetők egy web oldalra, ahol elérhetőek a diákok számára és interaktív gyakorlatokként is használhatók.

A feladatoknál be lehet állítani a tesztelésre szánt időt, a betűtípust, a megjelenő színeket, képeket, hangokat. A feltett kérdésekben, válaszokban, visszacsatolásokban és az értékelésben is animációt és videót is lehet használni. A program számolja a helyes válaszokat és százalékban adja meg az eredményt. A diák a feladat megoldása közben segítséget is igénybe vehet, de ezt a program az értékelésnél figyelembe veszi és csökkenti a százalékokat.

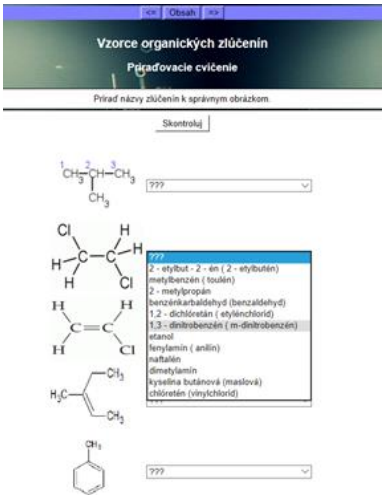
Karunkon minden leendő pedagógusnak el kell végeznie a tantárgyat, amelyben megtanulja a program és az interaktív tábla kezelését és az interaktív tananyag helyes készítésének "törvényeit", fortélyait. Természetesen saját szakjuknak megfelelően interaktív feladatokat is készítenek, melyekből néhányat bemutatunk.

## 3.2 Interaktív feladatok

Az óvópedagógus hallgatóknak figyelembe kell venni azt a tényt is a feladatok készítésekor, hogy a kicsik még nem tudnak írni és olvasni. Ilyen feladatokban céltudatosan kell használniuk a képeket, animációkat, hangokat. Lásd 1. ábra.

**1. ábra:** Okostáblára készített feladat – kicsik számára

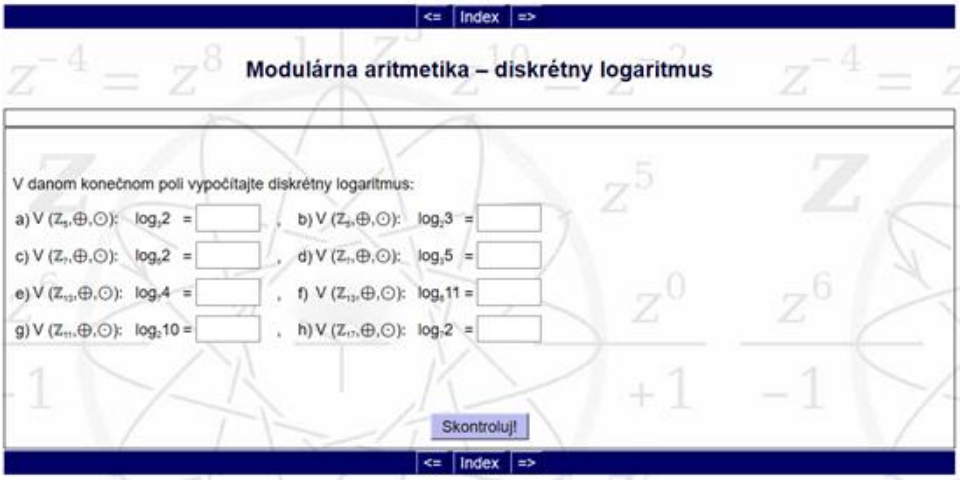

**2. ábra:** Irodalom




**3. ábra:** Kémia

A szlovák irodalom tantárgyra készült feladatban a diáknak a megadott címekből ki kell választani a szlovák írók fényképei mellé a hozzájuk tartozó irodalmi művet. Lásd 2. ábra.

A kémia órára ugyanolyan struktúrájú feladatban a megadott képletekhez kell megtalálni a helyes választ. Lásd 3. ábra.




**4. ábra:** Matematika

Egy példa matematikai feladatra, amely a logaritmusok tesztelésére, ismétlésére használható. Az üres négyzetekbe kell beírni a helyes választ. Lásd 4. ábra.

A néhány példa is jól mutatja, hogy a diákok megtanulják a program kezelését és ötletes feladatokat készítenek.

# 4. Okostelefon

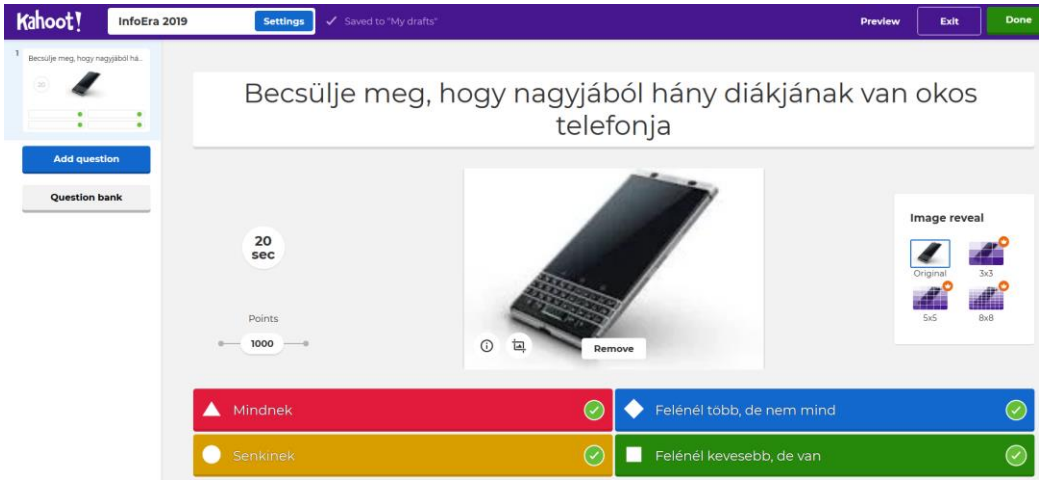
## 4.1. Mobil oktatóprogramok

Mint már a bevezetőben is írtuk, mára az emberek elsöprő többsége rendelkezik okostelefonnal – különösen igaz ez a fiatalabb korosztályra. Legtöbben szinte mindent ezen keresztül intéznek, kapcsolatot tartanak ismerőseikkel, információt keresnek, játszanak vagy éppen zenét hallgatnak, filmet néznek. Szinte természetszerűleg merül fel a kérdés, hogyan is lehetne ezeket az eszközöket is az oktatás szolgálatába bevonni? Nagyszerű alkalmazások állnak rendelkezésre például a nyelvtanulásnál, ilyen a *Duolingo*, amit kicsiktől az idősebbekig mindenki elvezettel használhat. Gyakorolhatjuk a szorzattá alakítást a Factor Monsters-sel, mobil eszközön is GeoGebrá-zhatunk vagy akár felépíthetünk saját világokat a MineCraft-tal. Bármelyiket is tesszük – *kibővítjük* vagy akár át is *alakítjuk* vele a tanulási módszereinket! A Duolingo, a GeoGebra esetében a szótanulást és a szerkesztést egyszerűsítjük le – a MineCraft viszont már egy teljesen *új* lehetőséget jelent, azzal hogy felépíthetjük, modellezhetjük az általunk kitalált dolgokat!

## 4.2. Szavazórendszerek

A szavazórendszerek megjelenése az iskolában gyökeresen változtatta meg a tanórák menetét. Az addig főleg frontálisan vezetett óra helyett lehetővé vált a gyerekek hatékonyabb bevonása, az egész osztály aktivizálása – a technika ebben az esetben jelentősen *átalakíthatja* az óraszervezést. Igen előnyös tulajdonság lehet, ha egy alkalmazás eszközt ad a csoport munka szervezéséhez is. A munkaerőpiac ma egyik talán legfontosabb elvárása, hogy olyan embereket kíván a legtöbb terület, akik képesek közösen dolgozni.

A *Kahoot* (https://kahoot.com/) egyike a legelterjedtebb ilyen jellegű, oktatásra kifejlesztett alkalmazásnak, amely bevonja és aktivizálja a gyermekeket. [4] [5] [6] Mind online, mind pedig letölthető formában elérhető. Az ingyenes változata is jól használható - létrehozhatunk kvízeket vagy felhasználhatjuk a már a rendszerben levő kész kérdéseket. (Aki kvízt szeretne létrehozni, annak regisztrálnia kell a https://kahoot.com/ oldalon) A kérdésekhez időlimit beállítható, képek, zenék, sőt videók is beszúrhatók. Lásd 5. ábra. A gyerekek a rendszer által generált 6 számjegyből álló PIN kód begépelésével csatlakozhatnak. A személyesebb hangvételhez megadhatják a becenevüket vagy a tanár kérhet a rendszertől generált neveket a bejelentkezőkhöz. (Fegyelmezetlenebb diákoknál gondot jelenthet az anonimitás)




**5. ábra:** Kahoot – kérdés készítése

A tanár kivetíti a kérdést, majd a válaszlehetőségeket is. A diákok eszközein már csak a válaszokhoz rendelt jelek jelennek meg, a kérdés és a válaszlehetőségek nem. Lásd 6. ábra.


**6. ábra:** Kahoot kérdés a diák eszközén és a tanári oldalon ennek kiértékelése

Amikor a tanár elindítja a kvízt, választhat, hogy *Challenge*-t (kihívás), vagy *Host Live-ot* (élő játék) szeretnének játszani. A kihívásnál megadhatjuk, hogy mennyi ideig legyen elérhető a játék és meghívhatunk tagokat e-mail-en keresztül vagy link segítségével. Legfőbb előnye, hogy az otthoni munkát is a megszokott felületre teszi át, változatos multimédiás tartalmakkal bővítve – ezzel nemcsak az órai munkát változtatjuk meg, hanem jelentősen *bővíthetjük* a házi feladatok repertoárját is. Amennyiben a *Host Live*-ot választjuk, akkor azon belül *Classic* (klasszikus), vagyis, hogy mindenki egyedül játszik, vagy *Team mode* (csapatjáték) érhető el. A *Team mode*-ban a csapattagoknak 5 másodpercük van megbeszélni a helyes választ.

A kvíz befejezésekor lehetősége van a tanárnak visszajelzést kérni arról, hogy tetszett-e a játék, akarják-e folytatni. Ezenkívül az alkalmazás pontozza a válaszokat és győztest is hirdet – több helyes válasz esetében a gyorsaság is számít. Az eredményeket xls fájlban letölthetjük, vagy a szerveren tárolhatjuk. Lásd 1. táblázat.

**1. táblázat:** Kahoot kvíz eredménye

| InfoEra 2019 | |
|---|---|
| Played on | 9 Nov 2019 |
| Hosted by | teacher |
| Played with | 1 player |
| Played | 1 of 1 |
| | |
| **Overall Performance** | |
| Total correct answers (%) | 100,00% |
| Total incorrect answers (%) | 0,00% |
| Average score (points) | 633,00 points |
| | |

| Feedback | | | | | | |
|---|---|---|---|---|---|---|
| Number of responses | 1 | | | | | |
| How fun was it? (out of 5) | 0,00 out of 5 | | | | | |
| Did you learn something? | 0,00% Yes | | 0,00% No | | | |
| Do you recommend it? | 0,00% Yes | | 0,00% No | | | |
| How do you feel? | ◉ | 100,00% Positive | ◉ | 0,00% Neutral | ◉ | 0,00% Negative |
| | | | | | | |
| Switch tabs/pages to view other result breakdown | | | | | | |

Kisebb korosztályok esetében fontos szempont lehet az alkalmazás kiválasztásánál, hogy az anyanyelvén tudja-e elérni. Szerencsére a *Kahoot*-ot át lehet állítani magyar nyelvűvé, bár az eredmény nem tökéletes – pár szó így is angolul marad, mint például a *True* és a *False* felirat. Ennél nagyobb probléma, hogy az ingyenes változat egyetemi oktatásban nagy létszámú előadásokon biztosan nem használható – maximum 50 diák csatlakozhat egy kvízhez.

## 4.3. Algoritmizálás

2006-ban Jennet Wing robbantotta be a köztudatba a számítógépes gondolkodás fogalmát, az általános műveltség szintjére emelve azt. [7] Ráérzett arra, hogy mai világunkban mindenkinek szüksége van informatikai ismeretekre, számítógépes gondolkodásra. Az új Nemzeti Alaptanterv Tervezete (https://www.oktatas2030.hu/) már fókuszba helyezi a digitális eszközök használatán kívül a robotikát, amellyel az algoritmizálást, programozást kívánják megalapozni. Az algoritmikus gondolkodásra tanítást már alsóban el lehet kezdeni. [8] Ezt a célt maguk elé tűzve, az iskolákban egyre több helyen kísérleteznek a számítógép nélküli számítógépes gondolkodás feladatainak felhasználásával (http://csunplugged.org ), indítanak robot szakköröket [9] vagy programozó szakköröket (Logo, Scratch).

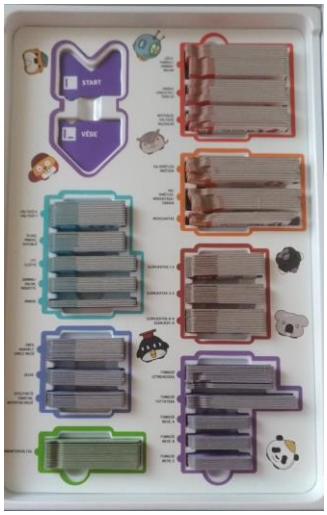
Ezen lehetőségek, a választható módszerek körét *bővíti* egy új kezdeményezés. 2018-ban megjelent a *Scottie Go*! (http://www.ebt.lt/lt/produktai/scottie-go ) társasjáték, amely a blokkprogramozás rejtelmeibe vezeti be a tanulókat. Amiben ez a programozást bevezető játék különleges és innovatív, az az, hogy egyszerre az *összes érzékszervre hat*. Fontos, hogy lássák, hallják és személyesen is tapasztalják a felmerült problémát. Ráadásul azonnali visszacsatolást kapnak a játéktól, így folyamatosan tudnak haladni.

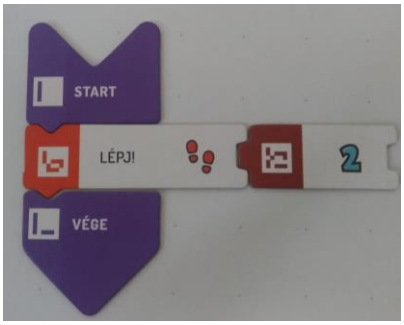
*Scottie* a világok között utazik, azonban egy váratlan pillanatban meghibásodik az űrhajója, és a Földön száll le, hogy beszerezhesse az összes alkatrészt – a gyerekeknek ebben kell őt segíteniük! A társasjáték rendelkezik kézzelfogható és virtuális elemekkel is. Az előbbi a játék puzzle (kódrészletek) darabjai, míg az utóbbi egy alkalmazás (Scottie Go! Edu) tableten vagy okostelefonon futtatható. A lényeg, hogy a „kirakott" algoritmust lefényképezve az alkalmazás végre tudja hajtani és így azonnali visszacsatolást tud adni a gyerekeknek.

A társasjátékban a logikusan összekapcsolódó darabjai ugyanazzal a színnel vannak jelölve, mint pl.: mozgás = narancssárga. A dobozban (7. ábra) nyolc részre vannak bontva a puzzle darabok, így jól lehet tájékozódni benne. A játék 10 szintből, (amelyek a földrészek meglátogatása pl.: Európa1, Európa2) és azok 8-10 pályából állnak. Minden szint sikeres elvégzése után kapunk egy alkatrészt, így Scottie meg tudja javítani a meghibásodott űrhajóját.

A feladatot az alkalmazás adja, a tanulónak viszont „kézzel" kell a kódot megírni, vagyis kiraknia. (Akárcsak a robotoknál, itt is a fizikailag létező darabokat kell megfogni, mozgatni.) A kód nagyon egyszerűen írható, hiszen a kirakóhoz hasonlóan csak a megfelelő helyre lehet illeszteni őket (7. ábra) – akárcsak a blokk alapú nyelvek bármelyikénél például a Scratchnél. Amikor ezzel a tanuló készen van, az applikáció segítségével beolvassa azt, majd *Scottie* teljesíti a feladatot, amit a képernyőn követhet (7. ábra) – így azonnali visszacsatolást kap a megoldásról. (Ne felejtsük el, hogy a személyes tanári dicséret milyen sokat jelent a gyerekeknek!)

A kártyákon különböző egyszerűsített QR kódok találhatóak, így szinte kivétel nélkül, mindig *helyesen* dolgozik. A kódsorainkat kétféleképpen tudjuk beolvasni: fényképezéssel és kamerázással. Ezeket a beolvasáskor ki tudjuk választani az ikonoknál. A fényképezés rövidebb programsorok esetében ajánlható. Ilyenkor a fényképen kis pipák jelennek meg, így ellenőrizhetjük, hogy helyesen történt-e a beolvasás. A felvétel készítést akkor érdemes használni, ha már hosszabb sorokat írtunk. Ekkor a képernyőn megjelennek a kártyák másai és le tudjuk ellenőrizni, hogy helyesen dolgozott-e.






**7. ábra:** Scottie Go doboza (balra). Egy egyszerű parancs (jobbra fent). Az alkalmazás képe (jobbra lent).

Egy Scottie-s óra folyamán a tanulók együttműködő és kommunikációs képessége is fejlődik, hiszen csoportokban célszerű alkalmazni. A csapatokba érdemes különböző alkalmazói szinten lévő tanulókat válogatni, hogy aktívan tudjanak egymásnak segíteni és egymásnak is magyarázni. Ennek köszönhetően ők maguk is fejlődnek, nem csak a társuk. Természetesen a problémamegoldó és kreatív képesség is fejlődik a játék során, hiszen, ha valamit nem sikerül elsőre maximális csillagszámmal teljesíteni, akkor egy újabb programot kell írniuk, amely hatékonyabb, de a végeredmény ugyanaz.

*A tanulók nagyon könnyen ráéreznek a Scratch programozásra a játék segítségével, hiszen apró lépésekben magyarázza az egyre nehezedő feladatokat.* [10]

## 4.4. Programozás

Az eddigiekben láttuk, hogy milyen sok új lehetőséget adott az oktatás számára a számítógépek, az okostáblák és most a mobil telefonok elterjedése. Egyetlen dologról nem beszéltünk még, arról, hogy programozhatunk is velük! 2013-ban Barack Obama mondta: *„Don't Just Play on Your Phone, Program It":*
https://obamawhitehouse.archives.gov/blog/2013/12/09/don-t-just-play-your-phone-program-it

Már akkor is úgy látták, hogy mindenki rendelkezik egy „mini" számítógéppel, a telefonjával és létkérdés a programozás alapjainak megtanulása – erre pedig ez az egyik legjobb út! Valljuk be a telefon ilyen jellegű használata teljesen újszerű lehetőségeket teremt az oktatásban.

A Microsoft *TouchDevelop* app segítségével mobil készüléken lehetett alkalmazásokat fejleszteni. [11] Nemcsak ebben volt egyedi, hanem abban is, hogy egyszerre három különböző szinten lévő programozót tudott kiszolgálni. A legkisebbek, a kezdők blokk alapú vezérlőkkel dolgozhattak, a következő szinten váltani lehetett a grafikus és a mögöttes kód között, hogy könnyebb legyen az átállás és a leggyakorlottabbak már közvetlenül egy script nyelvet használhattak. Az alkalmazás elérhető volt Windows, Android és IPhone készülékeken.

2019-ben ennek az eszköznek a fejlesztése leállt, helyette a *CodeMaker Arcade* áll csatasorba, ahol a fókusz már azon lesz, hogy ugyanazt a kódot különböző eszközökre lehet letölteni például Arduino-ra, Microbit-re vagy Raspberry-re. Itt is a TouchDevelop-ból ismerős editorral találkozunk. Jelenleg még csak béta verzióban létezik, de kíváncsian várjuk a végleges megoldást.

# 5. Összegzés

Mint ahogy azt a saját bőrünkön is érezzük, az informatika napról napra viharos gyorsasággal fejlődik. Újabb technológiák, eszközök, alkalmazások jelennek meg. Mára a mennyiségi változás minőségi változásba kezd átfordulni az oktatás területén. Nemcsak a régi módszerek használatát frissíthetjük fel IT eszközökkel, hanem gyökeresen új tanítási módszereket is láthatunk, amelyek elképzelhetetlenek lettek volna akár húsz évvel ezelőtt is. Elmondhatjuk, hogy a szemünk előtt formálódik a modern, XXI. századi iskola, amely már informatikai alapokon nyugszik!

## Köszönetnyilvánítás

A tanulmány megjelenését a KEGA 015TTU-4/2018: „Interaktivita v elektronických didaktických aplikáciách." (Interaktivitás az elektronikus didaktikai alkalmazásokban) című projekt támogatta.

## Irodalom

1. Erdősné Németh Ágnes: *A LOGO-tól az Informatikai Olimpiáig – Informatikai tehetséggondozás a középiskolában* (PhD dolgozat)
2. Pšenáková Ildikó, Heizlerné Bakonyi Viktória, Illés Zoltán: *Interaktivitás az informatika tanórákon*. InfoDidact 2018, 2018. nov. 22-24. Zamárdi, ISBN 978-615-80608-2-0
3. Pšenáková, I.: *A digitális tananyag.* In: Képesség-fejlesztés digitális tananyaggal. Debrecen: Kocka Kör, 2010. ISBN 978-963-87488-9-8, p. 9-54.
4. Papia Bawa: *Using Kahoot to Inspire*. In: Journal of Educational Technology Systems 47(2):004723951880417, October 2018, DOI: 10.1177/0047239518804173
5. Manjet Kaur Mehar Singh, Malini Ganapathy, Debbita Than: *Kahoot!: Enhancing Creativity in Classroom Learning*, In book: Creativity in Education, Publisher: USM Press, 2018, ISBN: 978-967-461-400-3

6. *Why Kahoot is one of my favourite classroom tools*. http://tomorrowslearners.com/why-kahoot-is-one-of-my-favourite-classroom-tools/ (utoljára megtekintve: 2019.11.09.)
7. J.M. Wing, *"Computational Thinking,"* Communications of the Association for Computing Machinery Viewpoint, March 2006, pp. 33-35.
8. Dr. Lénárd András: *Az algoritmikus gondolkodás fejlesztésének fázisai az alsó tagozaton módszertani megközelítésben*. InfoDidact2018, 2018. nov. 22-24. Zamárdi, ISBN 978-615-80608-2-0
9. Fári János: *Robot csámborgás*. konferencia előadás, InfoÉra 2016 nov. 24-26.
10. Maja Videnovik, Vladimir Trajkovik: *USING SCOTTIE GO! AS GAME BASED LEARNING TOOL FOR COMPUTATIONAL THINKING COURSE*. https://library.iated.org/view/VIDENOVIK2018USI (utoljára megtekintve 2019.11.09.)
11. Viktória, Heizlerné Bakonyi: *Touchdevelop as the Tool for Novice Programmers.* AASCIT JOURNAL OF EDUCATION 3: 6 pp. 54-60., 7 p. (2017)

# Feladatok Algoritmus Vizualizációval

Bende Imre

beiraai@ludens.elte.hu
ELTE IK

**Absztrakt.** Arra szeretnék választ adni a következőkben, hogy algoritmus animációval, vizualizációval milyen feladatokat, módszereket lehet felhasználni a tanórákon, valamint a számonkérés során. Illetve milyen feladatok segíthetik a programozásoktatást, amelyeket mintákkal, példákkal is szemléltetek.

**Kulcsszavak:** algoritmus animáció, algoritmus vizualizáció, programozásoktatás, számonkérés

## 1. Bevezetés

Az algoritmus vizualizációs eszközök alapkövei lehetnek az informatikaoktatásnak. Hatékonyságát, helyét az oktatásban már számos helyen bizonyították (például: Karavirta disszertációjában összegyűjtött több kutatást is, amelyek erre az eredményre jutottak [3]). Ha a lehetséges problémákra választ, megoldást tudunk adni, akkor nem lehet kérdés, hogy érdemes-e felhasználniuk a tanároknak a módszert a tanóráikon. Azonban hiába szeretnénk használni, kérdés lehet az, hogy érdemes-e, és ha igen milyen feladatokat, módszereket lehet felhasználni a számonkérés során. A következőkben erre szeretnék választ adni, úgy, hogy leírom én milyen módszereket tudnék elképzelni, majd ezeket példákkal is bemutatom. Megjegyezném, hogy az algoritmus animáció és a vizualizáció között van különbség, azonban a cikk során ezeket nem különítem el és mindkettőt felhasználom a feladattípusok létrehozása közben.

## 2. Feladattípusok

### 2.1. Algoritmus leírása papíron

Papír alapú számonkérésként azt tudom elképzelni, hogy le van írva az algoritmus maga, vagy csak annak megnevezése, majd egy minta teszteset segítségével le kell írniuk a diákoknak, hogy az egyes lépésekben mi történik (gondolok itt arra, hogy egy-egy közbenső utasításnak mi a célja; mik a változók értékei, hogyan módosulnak), illetve az is egy lehetséges feladat emellett, hogy valamilyen módon a papíron ábrázolják a főbb lépéseket vizuálisan. Az értékelés főbb szempontja ebben az esetben az, hogy minden lényegesebb utasításnál tett-e valamilyen féle megjegyzést, illetve a főbb változók (lehet olyan feladat, melyben mondjuk egy ciklusváltozót nem kötelező jelölni, mivel nem játszik nagyobb szerepet) értékei leírásra kerültek-e.

Ha nem is az online világot nézzük, szorosan az algoritmus vizualizációhoz köthetjük az egyes feladatok életben vett reprezentációit. Ezekre nagyon jó mintákat, konkrét feladatokat kínál a CSUnplugged című könyv [1], mely feladatonkénti leírással, felhasználhatósággal, anyagokkal rendelkezik. Mindemellett a feladatok valósághűvé teszik az algoritmus működését, használatát, így az értelmezése is könnyebbé válik a fiatalabbak számára, mindezt játékosan próbálták megalkotni a könyv készítői. A kötetben találhatunk többek között kereséses (lineáris, logaritmikus, hasheléses), rendezéses, gráfalgoritmusos gyakorlatokat is.

Az Algorithms Unplugged [7] hasonló feladatokkal rendelkezik, mint az előző bekezdésben említett CSUnplugged, viszont komplexebb algoritmusokat (keresés, rendezés, gráfalgoritmusok, dinamikus programozás, hatékonyság javítás) mutat be, amelyek idősebb, több alapismerettel rendelkező diákoknak lehet megfelelő. A megközelítés hasonló: a könyv motivációt akar adni a diákoknak a

programozástanulásra offline feladatokkal. Részletes magyarázattal és többfajta vizualizációs példával mutatja be az algoritmusokat. Több esetben találkozhatunk olyan tananyagokkal is, amik csak egyetemen fordulhatnak elő, viszont megfelelő illusztrációkkal ezeket is jól és érthetően mutatja be.

Egy másik érdekes megközelítés a Dynamicland [I5] környezete, ahol a Lua-ban megírt programoknak fizikai megjelenítése van. A papírlapok funkcionálnak programként, illetve fizikai tárgyakat tudunk bemenetként, módosító tényezőként használni. Jelenleg ez még elég új, de erre a technológiára a későbbiekben mindenképpen érdemes odafigyelni, illetve amennyiben elterjedtebb lesz, mindenképpen érdemes lehet kipróbálni, használni az informatika órákon.

## 2.2. Algoritmus vizualizáció gyűjtése

Előre kiszabott, vagy saját érdeklődés alapján kiválasztott algoritmusról lehetne algoritmus vizualizációkat keresni gyűjtőmunkaként, majd ezeket a diákság osztályozza, értékeli, s végül összehasonlítja őket egymással. A keresést lehet támogatni azzal, hogy vizualizációs gyűjteményeket, kollekciókat, tárházakat mutatunk nekik, így már eleve tudják hol kell kezdhetik el a keresést (példa kedvéért: AlgoViz [I1], VisuAlgo [I8]). A tanórákon elmondjuk mely szempontok lehetnek fontosak egy vizualizációval kapcsolatban (gondolok itt Myller által meghatározott pontokra [4]), melyek alapján tudják értékelni az egyes eszközöket. A végtermék egy dokumentum, linkgyűjtemény lenne, amely összehasonlít egy algoritmushoz tartozó vizualizációkat, sőt akár végeredmény lehet egy vizualizáció is, amelyet aztán az osztály többi tanulójának is meg lehetne mutatni tanulás céljából, ezáltal a tanár feladatainak száma is csökkenhet.

## 2.3. Algoritmus vizualizáció gyűjtése

Lehetséges otthoni feladat egy algoritmus vizualizáció megtervezése, elkészítése, lefejlesztése. Első feladatként a tanár feladhatja, hogy készítsenek a diákok egy dokumentumot, amelyen részletesen megfogalmaznak egy algoritmust (lehet szó olyanról, amit órán már vettek, vagy olyanról is amit máshonnan ismernek és érdekesnek találtak), illetve egy hozzá tartozó animációt, vizualizációt, bemutatva azt, hogy mely esetekben, mikor, mi történik rajta. Ha vannak jobb képességű, tehetségesebb tanulók, akkor pedig folytatásként, akár szorgalmi feladatként lehet az, hogy a dokumentum alapján elkészítik ténylegesen is az AV-t (opció lehet, hogy ekkor más diákok által elkészített tervezetek közül is választhatnak, ha nekik az jobban tetszik, szívesebben foglalkoznak vele). Könnyítés lehet, hogy ha mutatunk nekik egy olyan keretrendszert, library-t, mely részletes, jó leírást, dokumentációt tartalmaz, így egyszerűbb lesz a helyzetük a feladat elkészítésében. Mivel ezt főként otthoni, szorgalmi feladatnak szánnám ezért, ha a munkán és annak részletességén látszik a megfelelő vele eltöltött időmennyiség, akkor azt valamiféle jutalmazással értékelném. Azáltal, hogy maguk készítenek egy vizualizációt sokkal jobban megértik az általuk választott algoritmus működését, illetve plusz programozói tudásukat, kompetenciáikat is fejlesztik környezettől függően (például egy webes felület esetén HTML, CSS, JavaScript készségeiket is fejlesztik).

Algoritmus vizualizáció készítése lehetséges lenne más eszközökkel is. Lehet egyszerűen állapotokat készíteni, rajzolni akár papíron, akár gépen, ezeket lefotózva diavetítésszerűen lehetne animálni, bemutatni az algoritmus egyes lépéseit. Minél részletesebben mutat be egy-egy lépést, állapotot (ami fontos az algoritmus működése szempontjából), minél több lépést tartalmaz az animáció, annál jobban magyarázza el, mutatja be az algoritmust. Ennél összetettebb egy Flash készítése, ami hasonló elven működhet, de közben a Flash készítést is gyakorolják a diákok. Flash alapúhoz hasonló megoldásokat találhatunk például Végh Ladislav által alkotott weblapon is, mely referenciaként szolgálhat hasonló animációkhoz [6], de az AlgoViz oldalán is találhatunk ilyeneket [I1].

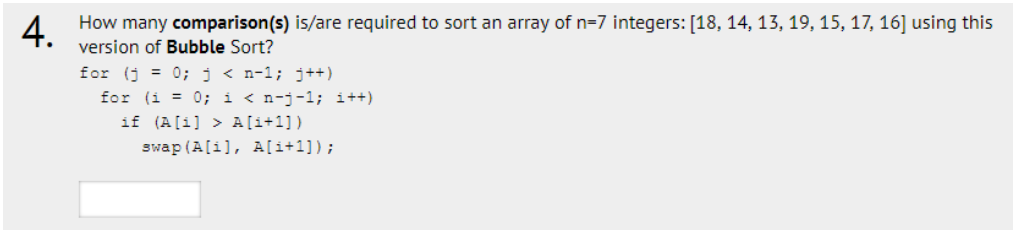
Emellett persze csak a képzelet szab határt annak, hogy egy-egy diák milyen megközelítésből is tud, vagy szeretne megcsinálni egy vizualizációt. Itt mindenképpen meg szeretném említeni Kátai Zoltán és Tóth László által rendezett néptáncokat, melyek egy-egy algoritmust mutatnak be táncosokkal (példa kedvéért buborékos rendezés, gyors rendezés, összefésülés rendezés stb.) [I7]. De

nagyon látványos animációkat is lehet találni az interneten, ilyen például egy kis robot által bemutatott részletes videó, mely a jelentősebb rendezéseket mutatja be, majd hasonlítja össze őket [I6].

## 2.4. Algoritmikus feladatok megoldása AV segítségével

Számonkérés lehetséges lenne gépen is, online, interaktív tesztek formájában. Ebben az esetben fel lehetne tenni működésbeli kérdéseket az algoritmussal kapcsolatban (például: egy konkrét bemenet alapján a buborékos rendezés hány cserét igényel, adott állapotban a változók milyen értékekkel rendelkeznek, mely algoritmus a leírt pszeudokód). A válaszokat megadhatjuk kérdés típusától függően sima bemeneti mezőbe, feleletválasztósként, illetve igaz/hamis formában is. Az algoritmus működését érintő kérdéseknél a bemeneti értékeket random függvénnyel is generálhatjuk, így nem kell több különböző kérdéssort összeállítani, hiszen a tanulóknak nagy valószínűséggel különböző feladatok jönnek létre. Mivel a teszt online környezetben jönne létre, így a tanárnak az értékeléssel egyáltalán nem kellene foglalkoznia, vagy csak kevés feladata lenne vele. Tanártól és iskolától függően pedig állítható, hogy a százalékos eredmények alapján, milyen érdemjegy jár a dolgozatot kitöltő diáknak.

Végh Ladislav hasonló feleletválasztós feladatsorokkal próbálta vizsgálni, hogy a diákok mennyire értették meg az órákon szereplő algoritmusokat [6]. De az interneten is találhatunk erre hajazó, hasonló példákat, ahol a megoldások a teszt kitöltését követően kiértékelésre kerülnek. Ilyenek vannak például az VisuAlgo honlapján [I8], mely nem csak vizualizációkat, hanem a témakörökhöz tartozó feladatsorokat is tartalmaz.




**1. ábra:** Példa VisuAlgo honlapján lévő tesztfeladatra

Emellett el tudok képzelni olyan feladatokat is, amelyeknél egy animációt nézve kell meghatározni, leírni a bemutatott algoritmust (a megoldásoknál pszeudokód alapú algoritmusleírás lehetséges). Alapvetően gondolok itt arra, hogy az algoritmus megjelenik a felületen, de néhány változó, utasítás ki van szedve belőle, melyeket a diákoknak kell kiegészíteni a vizualizáció lépéseinek megfigyelésével, megvizsgálásával. Összetettebb, nehezebb lehet, ha az animáció alapján az egész algoritmust kell meghatározni, de ezt csak részletesebben ábrázolt, könnyebben felismerhető algoritmusoknál lehetne feladni, megoldani.

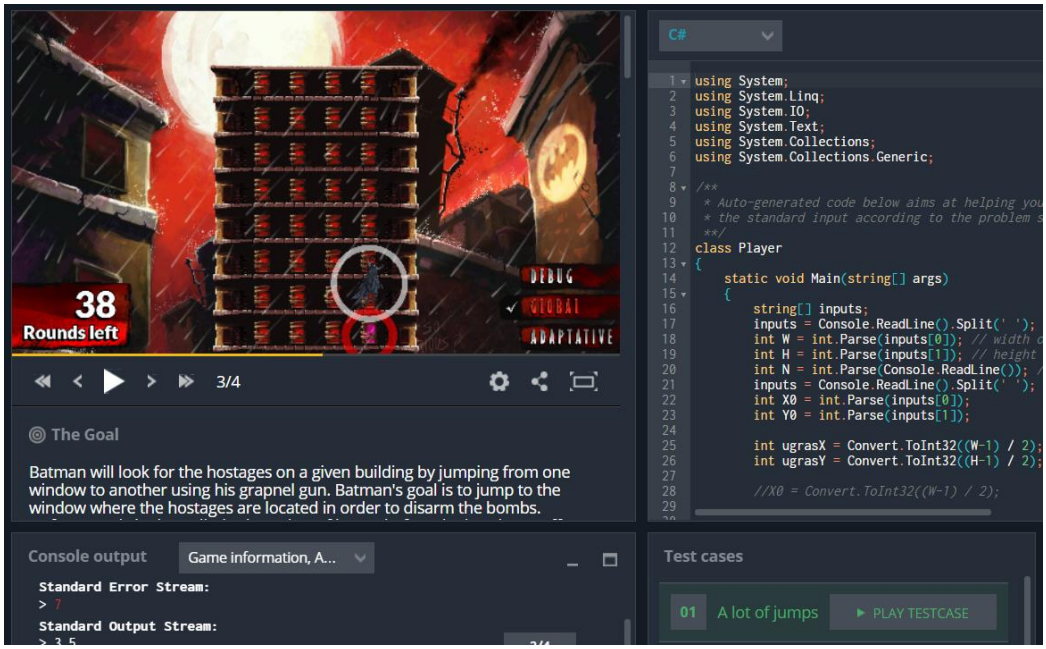
Egy másik megközelítése lehet az előző fejezetben említett feladattípusnak, hogy az algoritmus leírását és/vagy algoritmus nevét ismerjük, majd a feladata az lenne, hogy a diákok az animációt irányítsák pszeudokód, vagy saját tudásuk alapján. Interaktivitási eszközök lehetnek az elemek helyükre húzása, értékek beírása. Ebben az esetben a feladat megoldása sikeres, ha az algoritmus végeredményét megkaptuk, de úgy, hogy az algoritmus működését követtük.

## 2.5. Programozási feladatok AV támogatással

Alapvetően nem minden diák számára motiváló, hogy amikor programozási feladatokat oldanak meg kódolással, akkor visszacsatolásként, tesztelésként mindösszesen egy konzolt látnak, illetve csak azt tudják használni. Érdekesnek tartom azt az ötletet, hogy ha fogjuk ezeket a hagyományosnak vett programozási feladatokat és a feladat megoldását, tesztelését algoritmus animációval, vizualizációval támogatnánk meg. Ezt úgy tudom elképzelni, hogy az algoritmus eredménye irányit egy animációt, illetve különböző előre megírt tesztesetek léteznek a feladatokhoz, majd a mi algoritmusunk

ezeket próbálja megoldani, mely során látványos eszközökkel ad egyfajta visszaigazolást a program helyességére vonatkoztatva.

Példaként a CodinGame oldalán [I4] ez egy megvalósított eszköz. Többféle típusú, nehézségű feladat van, melyek jelentős része vizualizációval van támogatva. Ez társul azzal, hogy nemcsak a módszert használja, hanem a feladatok is érdekesnek mondhatók, így összeségében ez motiváló erőként hathat azok számára is, akik kevésbé ilyen beállítottságúak, kevésbé szeretnek konzolos programokat írni algoritmus tanulás szempontjából.

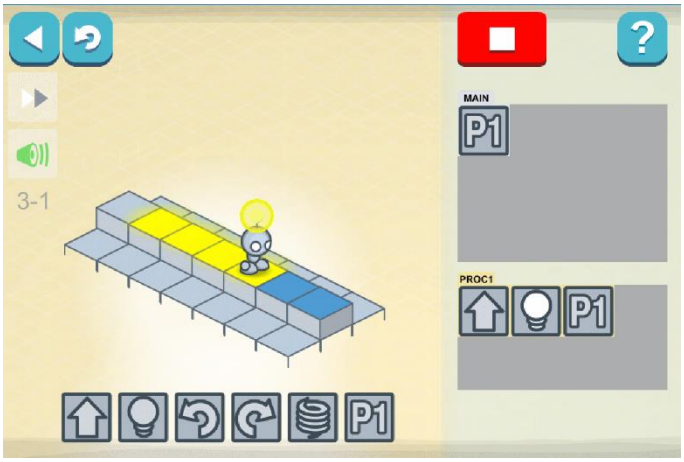



**2. ábra:** Egy feladat a CodinGame oldalán

Az ábrán jól látszik a felület összes eleme. Bal oldalon helyezkedik el a vizualizáció (léptethetőséggel, videó lejátszó funkciókkal), a feladat szövege és a konzol bemenete/kimenete, míg jobb oldalon a megírt forráskód (több programozási nyelv közül is lehet választani), illetve a tesztesetek elindító gombjai vannak.

Egyszerűbb feladatok mellett mesterséges intelligencia alapú játékok is megjelennek az oldalon, melynek elkészültekor lehetőség van más emberek munkáival versenyezni, összecsapni.

Emellett egyszerűbb utasításalapú feladatokat, „játékokat" is találhatunk az interneten. Többek között ilyen a CodeCombat [I3], vagy a Lightbot [I5]. Míg az elsőben egy harcost kell irányítani a várbörtönben különböző nehézségeket leküzdve, a másikban egy robottal kell végig menni a pályákon úgy, hogy mindeközben a kék mezőket kivilágítja (utasítások, lépések kiválasztásával).




**3. ábra:** Lightbot egyik feladánya

## 3. Amőba játék

Pár gondolat, játék a feladattípusok kutatása közben megihletett, így elkészítettem egy egyszerűbb kódolásos játékot. Az alapelgondolás azonos az amőba játékkal azonban ahelyett, hogy a diákok egymással csapnak össze mesterséges intelligenciákat (későbbiekben MI) készítenek, fejlesztenek (jelenleg ezt JavaScript nyelven), melyek helyettük játszák le a játszmákat. Habár az amőba közismert játék, de azért röviden leírom a lényegét: Két játékos van, az egyik az X-szel, a másik a O-rel, majd egy táblázatban egymás után felrajzolják egy-egy négyzetbe a jelüket. Az a játékos nyer, aki vízszintesen, függőlegesen vagy átlóba tud öt azonos jelölést tenni (ennek rövidebb verziója a 3x3-as tábla, ahol csak hármat kell összegyűjteni).

A diákok egy-egy mérkőzés után átgondolhatják mi az, amit másképpen csinálhatna a programjuk, hol, miért is veszített az, ezáltal motiválva vannak a folyamatos fejlesztésre, hogy minél „jobb" amőba MI-t készítsenek. Nem csak másokkal állíthatják szembe kódjukat, hanem saját maguk is játszhatnak ellene (kézi irányítással), így egy „belső" környezetben is tudják tesztelni, mérni annak képességeit, határait, hiányosságait.

Mivel van beépített MI is (mivel ez JS-en íródott, így a forráskódját a diákok elérik, illetve az egész oldal is letölthető a GitHub-ról), amit (direkt) viszonylag egyszerűbb legyőzni ezért általános iskola 7. osztályától felhasználható, akár órai keretek között is bemutatásra. Azonban az egymás közti mérkőzéseket nem célszerű órán is használni, hiszen aki kevésbé ügyes, gyengébb kódot készít demotivált lesz, kevésbé lesz kedve ezzel foglalkozni. Ehelyett azonban órán kívüli tevékenységként, szakkörök alkalmával, illetve akár versenyként is tökéletesen fel lehetne használni ezt a játékot.

# 4. Feladattípusok kategorizálása

Kérdés lehet, hogy melyik típust, kiknek, milyen körülmények között lehet, érdemes használni. A következőkben különböző szempontok szerint kategorizálom az egyes feladattípusokat. Korosztály, módszer és interaktivitás alapján.

## 4.1. Koronként vizsgálva

Első lépésben korosztályonként vizsgálom a kérdést, figyelembe véve a jelenlegi Kerettantervet [I6].

- A fiatalabbaknak (általános iskolásoknak) mindenképpen a szórakoztatóbb, egyszerűbb feladatokat ajánlom, gondolok itt a CSUnplugged-ban található mintákra, illetve egyszerűbb utasításalapú játékokra (ezek elkezdése minél előbb lehetséges, hiszen sok esetben olvasni, írni sem kell tudni ahhoz, hogy a robotot vagy az adott figurát irányítani lehessen).
- A középkorosztálynak általános iskola végén, gimnázium elején, mikor tanulnak már programozási tételeket, egyszerűbb rendezéses algoritmusokat, akkor már képesek arra, hogy fényképekből, diákból készítsenek animációkat, melyeket bemutathatnak a többieknek is. Idősebbeknek, gimnazistáknak az összetettebb, de mégsem a legnehezebb AV-alapú programozós játékokat (például CodinGame) lenne célravezető feladni, illetve akár komplexebb algoritmusokról szóló AV-k készítését is rájuk bízhatjuk.
- Végül egyetemistáknak lehetséges módszer, hogy a nehezebb programozási feladatokat feladjunk, illetve ekkor már rendelkeznek olyan szintű programozási, algoritmikus, technológiai tudással, ismerettel, gondolkodással, hogy saját maguk is képesek legyenek algoritmus animációt, vizualizációt létrehozni (gondolok itt Flash alapú animációra, weboldalra, vagy akár egy Unity-vel létrehozott interaktív animációra is).
- A tesztalapú feladatok nehézségét könnyen lehet állítani, így azt kortól függetlenül lehet használni számonkérés során. Animáció mutatásával, annak részletességével, kérdések összetettségével, komplexitásával, bemutatott algoritmusok bonyolultságával lehet állítani a kérdéseken úgy, hogy a kiválasztott csoport, osztály előzetes vagy tanult tudásának, ismereteinek megfelelő legyen. Fiatalabbaknál inkább már tanult algoritmusokról érdemes feladatokat összeállítani, míg később új, eddig nem ismerteket is fel lehet használni, hiszen ekkor a diákok már képesek pszeudokód alapján értelmezni egy algoritmus működését.

Soron kívül pedig megemlíteném még a Logo-t, illetve a Scratch-et melyet főleg kicsiknek, általános iskolásoknak lenne érdemes használniuk. Manapság már a legtöbb helyen beépült a tananyagba, illetve rendelkezik olyan dokumentációval, segédeszközökkel, amelyek segítik a tanárok munkáját.

## 4.2. Módszer szerint

Ebben a fejezetben aszerint osztom szét az egyes feladattípusokat, hogy mely módszernek, számonkérési formának, melyet tartom ideális választásnak.

- Órai, otthoni gyakorlásra a gép nélküli offline tartalmakat, illetve a programozási feladatokat ajánlom, melyek megoldási menete, hosszúsága bele tud férni a tanórák hosszúságainak.
- Szorgalmi feladatnak a vizualizációkészítés, vizualizáció keresés lehet jó választás, hiszen az órai keretek, óraszámok rövidségébe ez nem férhet be, így pedig annyi időt foglalkozhatnak, tölthetnek vele, amennyit csak szeretnének.

- Számonkérésként érdemjegyért az online tesztfeladatokat és a komplexebb programozási feladatokat érdemes használni, hiszen ezek objektíven értékelhetők és könnyen javíthatók.
- Versenyfeladatként az algoritmus vizualizációval támogatott programozási feladatokat tudnám elképzelni, hiszen komplexebb feladatok is létrehozhatók a módszerrel, illetve segítséget tud nyújtani abban, hogy a diákok megértsék, majd megoldják azt. Az MI (mesterséges intelligencia) megoldást igénylő feladatokra bajnokságszerűen fel lehet építeni egy-egy versenyt, melynek menetét többféleképpen is felépíthetjük (egyenes kieséses, csoportmérkőzéses). Így ezáltal egyértelműen lehet kiválasztani a legjobbakat és nem, vagy csak nagyon ritkán fordulhat elő holtverseny a fontosabb helyezéseken.

Az előbb említett feladattípus nem csak egy megrendezett verseny keretein belül megvalósítható, hanem tanórákon (plusz akár otthoni munkával, továbbfejlesztéssel) is létre lehet hozni egy ilyen környezetet, mely motiválhatja a diákokat, hogy a többiek munkájával versenyezhessen. Ebben az esetben azonban oda kell figyelni azon hallgatókra is, akiket demotiválja a „vereség", lemaradás. Ennek elkerülése végett megoldás lehet az, hogy gyengébb MI-ket, megoldásokat is feltöltünk, melyeket az ő programjaik is le tudnak győzni.

### 4.3. Interaktivitás szerint

Már több kutatás is kimutatta, hogy minél interaktívabb egy algoritmus vizualizációs eszköz, annál hatékonyabb az a programozásoktatás során (Kavarirta által összegyűjtött eredmények [3]). Így figyelembe véve a Myller által meghatározott kiegészített taxonómiát [4] interaktivitási szint szerint is kategorizáltam az egyes feladattípusokat.

- Válaszadás: A különböző tesztekkel, vizsgafeladatok az interaktivitás ezen szintjét érjük csak el, viszont ezekben az esetekben a cél a tudás vizsgálata, nem pedig annak a megszerzése.
- Változtatás: A játékos feladatokkal a kimenet és maga a vizualizáció módosítható válik. Mindenképpen érdemes ezeket használni a programozással, algoritmizálással való ismerkedés során.
- Összerakás: Dynamicland-del és hasonló eszközökkel a diákok maguk is létrehozhatnak algoritmusokat, valamint a hozzá tartozó vizualizációikat.
- Bemutatás: Az interaktivitás legmagasabb fokán az van, ha a diákok maguk készítik el a vizualizációt, majd bemutatják azt társaiknak. Vagy akár csak az utóbbi, hiszen már ekkor is mélyebben meg kell ismernie a diáknak az algoritmus működését, hogy el tudják azt magyarázni.

## 5. Zárszó

A cikkben igyekeztem minél többféle feladatot, feladattípust, módszert felsorolni, amiket az informatika órákon, programozásoktatás során fel lehetne használni úgy, hogy mindeközben algoritmus vizualizációs (vagy animációs) eszközöket veszünk igénybe támogatásként. Több esetben is látszik az, hogy a felhasználása ezeknél a feladatoknál hatékonyabbá, élvezhetőbbé teszi a tanulást, illetve motiváló a diákok számára. Érdemes lehet változatosan, időkerettől függően minél több fajtát kipróbálni a diákokkal, hátha megszeretik ezeket az alkalmazásokat, módszereket és így szabadidejükben is fogják használni azokat önfejlesztésre (amelyet sok esetben csak játéknak tekintenek). Természetesen a feladattípusok listája nem teljes, technológiai, módszertani újítások folyamatosan jönnek létre és terjednek el, csak a képzelet szab határt annak, hogy még miként lehetne beépíteni a tanórákba az algoritmus vizualizációs eszközöket és az ehhez kapcsolódó feladatokat, módszereket. Ami biztos: mindenképpen használjuk az informatika órákon probléma-, feladatmegoldásra az AV-t, és folyamatosan figyeljük azt, hogy mivel lehetne még élvezhetőbbé tenni a diákok számára a prog-

ramozástanulást a módszer segítségével, mivel egyre könnyebben elérhetőek mindenki számára ezek az eszközök.

## Irodalom


1. Mike Fellows et al.: *Computer Science Unplugged*, Computer Science Unplugged, 2015.
2. Christophet D. Hundhausen, Sarah A. Ddouglas Andjohn T. Stasko: *A Meta-Study of Algorthm Visualization Effectiveness*. Journal of Visual Languages and Computing, pp259-290, 2002.
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. Törley Gábor: *Vizualizáció a Programozástanításban*, ELTE, 2013.
6. Végh Ladislav: *A Programozás Tanulásának és Tanításának Támogatása Elektronikus Tananyagba beépíthető Interaktív Animációs Modellekkel*, ELTE, 2017.
7. Vöcking B. et al.: *Algorithms Unplugged*, Springer-Verlag Berlin Heidelberg, 2011.


## Internetes hivatkozások


I1. The Algorithm Visualization Portal – http://algoviz.org

I2. Algorithm Animations and Visualizations – http://www.algoanim.ide.sk

I3. CodeCombat oldala – https://codecombat.com/

I4. CodinGame oldala – https://www.codingame.com/

I5. Dynamicland oldala – https://dynamicland.org/

I6. Kerettanterv – http://kerettanterv.ofi.hu/

I7. Lightbot oldalat – http://lightbot.com/

I8. Youtube: Merge Sort vs Quick Sort – https://www.youtube.com/watch?v=es2T6KY45cA

I9. Youtube: Merge-sort with Transylvanian-saxon (German) folk dance – https://www.youtube.com/watch?v=XaqR3G_NVoo

I10. VisuAlgo weboldal – https://visualgo.net/training

# A blokkalapú és a szövegalapú kódolás értékelése

Bernát Péter

bernatp@inf.elte.hu
ELTE IK

**Absztrakt.** A kezdő programozók számára nemcsak a szemantikailag, de már a szintaktikailag helyes program létrehozása is kihívást jelentő feladat. Utóbbit esetenként a szöveges programkód bevitelében segítő szolgáltatásokkal, más esetekben a szöveges kódolást kiváltó blokkalapú programozással kívánják megkönnyíteni. Cikkemben a kétféle kódlétrehozási lehetőség összehasonlításával és értékelésével igazolom, három informatikatanárral készített interjúval pedig alátámasztom, hogy a kezdők számára könnyebb a blokkalapú programozást elsajátítani, a haladó szintű programozáshoz viszont szükséges áttérni a szöveges kódolásra. Majd rámutatok arra, hogy az áttérés változatlan programozási témakörön belül is lehetséges, és utalok az interjúalanyaimnak az áttéréssel kapcsolatos tapasztalataira.

**Kulcsszavak:** programozástanítás, blokkalapú kódolás, szövegalapú kódolás, interjú

## 1. A kétféle kódolás

A kezdőknek szánt oktatási célú programozási nyelvek megalkotói számos megoldással igyekeznek a programozás legelső lépéseit minél érthetőbbé és motiválóbbá tenni. A Carnegie Mellon egyetem oktatói által létrehozott taxonómia [1] a törekvéseket három nagy kategóriába sorolja aszerint, hogy a programozási paradigmával, a program létrehozásával, vagy annak futtatásával kapcsolatosak-e.

A kezdő programozók számára nemcsak a szemantikailag, de már a szintaktikailag helyes program létrehozása is kihívást jelentő feladat. Utóbbit esetenként a szöveges programkód bevitelében segítő szolgáltatásokkal, más esetekben a szöveges kódolást kiváltó blokkalapú programozással kívánják megkönnyíteni. A blokkalapú programozás során a programkód paraméterezhető grafikus blokkok összeillesztésével állítható elő. A blokkok funkcióját szöveges vagy képi tartalmuk fejezi ki (1. ábra).


**1. ábra:** Programkód egy képes (ScratchJr) és egy szöveges (Alice) blokkalapú programozási nyelven

Cikkemben a kétféle kódlétrehozási lehetőség összehasonlításával és értékelésével igazolom, három informatikatanárral készített interjúval pedig alátámasztom, hogy a kezdők számára könnyebb a blokkalapú programozást elsajátítani, a haladó szintű programozáshoz viszont szükséges áttérni a szöveges kódolásra. Majd rámutatok arra, hogy az áttérés változatlan programozási témakörön belül is lehetséges, és utalok az interjúalanyaimnak az áttéréssel kapcsolatos tapasztalataira.

Az összehasonlítás során mindig zárójelben tüntetem fel, hogy valamely tulajdonsággal a példaként választott ScratchJr (scratchjr.org), Scratch (scratch.mit.edu) és Alice (alice.org) blokkalapú, illetve Imagine Logo (logo.sulinet.hu), RoboMind (robomind.net) és Small Basic (smallbasic.com) szövegalapú oktatási célú programozási környezetek közül melyek rendelkeznek.

## 2. A kétféle kódolás összehasonlítása

### 2.1. A nyelvi elemek beillesztése a programkódba

A blokkalapú programozási környezetekben a programozási területre behúzható nyelvi elemek egy blokk-készletben érhetők el funkció szerinti csoportosításban (ScratchJr, Scratch és Alice) (2. ábra). Ezáltal nem szükséges pontosan emlékezni a nevükre, és gépelési hibáktól mentesen lehet azokat beilleszteni a programba.

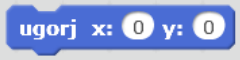

**2. ábra:** A Scratch blokkjainak csoportjai, és a Mozgás csoport néhány utasítása

A szöveges programkód bevitelét egyes fejlesztői környezetek az automatikus kódkiegészítéssel segítik (Small Basic). Ez a funkció a kontextus és a már beírt karakterek alapján egy a kurzornál megjelenő legördülő listában kínálja fel a lehetséges kulcsszavakat és azonosítóneveket, amelyeket egy vagy legfeljebb egy-két billentyű lenyomásával be lehet szúrni (3. ábra). Ez is segít a nyelvi elemek felidézésében, és a helyes bevitelükben, ugyanakkor az elírási hibák vele együtt sem zárhatók ki, és nem is mindegyik fejlesztői környezet kínálja fel ezt a szolgáltatást.


**3. ábra:** Automatikus kódkiegészítés a Small Basicben

## 2.2. A nyelvi elemek paraméterezése

A blokkalapú programozási nyelvekben a nyelvi elemek és a paramétereik együtt alkotnak egy blokkot, ezért csak a megfelelő *számú* paraméter adható meg a számukra a megfelelő *helyeken* (prefix vagy infix alakban) (ScratchJr, Scratch és Alice), és több paraméter esetén a szöveges tartalmukból kiderül az egyes paraméterek *funkciója* (Scratch és Alice) (1. táblázat, első oszlop).

| Scratch | Imagine Logo |
|---|---|
|  | `lenyomat` |
|  | `tollszín!` |
|  | `xypoz!` |
|  | `szó` |
|  | `és` |

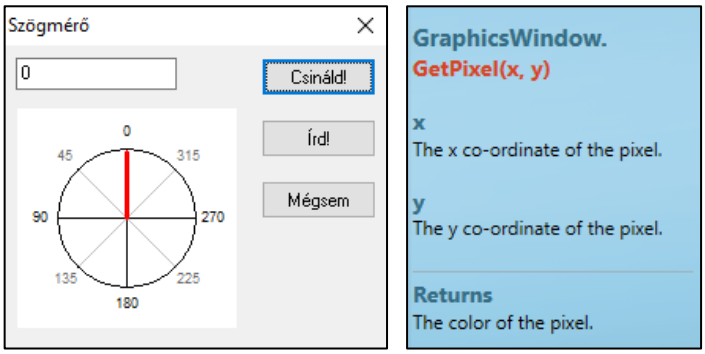
**1. táblázat:** Ugyanazon funkciókat betöltő nyelvi elemek a Scratch-ben és az Imagine Logóban; az előbbi nyelvben kiderül a paraméterek darabszáma, helye és funkciója, az utóbbiban nem

Az alábbi lehetőségek valamelyikével pedig azt is biztosítják, hogy csak megfelelő *típusú* értéket lehessen megadni paraméterként:

- a paraméterhely alakja megszabja a beilleszthető elem típusát (például a táblázat első oszlopában az *és* művelet paramétereiként csak két logikai értékű kifejezés adható meg) (Scratch);
- csak a megfelelő típusú érték gépelhető be (például a táblázat első oszlopában az ugorj utasítás paraméterhelyeibe csak számok írhatók) (ScratchJr, Scratch és Alice);
- egy lenyíló listából választható ki a paraméter lehetséges értékei közül valamelyik (Scratch és Alice).

A szövegalapú programozási nyelvekben csupán a nyelvi elemekből nem derül ki, hogy szükségük van-e paraméterre, és ha igen, akkor hány *darabra,* milyen *típusúra,* és több paraméter esetén milyen *pozícióban* és *szereposztásban* (1. táblázat, második oszlop). A fejlesztői környezetek egy részében ezért előhívható egy olyan ablak, amelyben a kurzornál található nyelvi elem egy űrlap segítségével paraméterezhető fel, így – a blokkalapú kódoláshoz hasonlóan – csak a megfelelő számú és típusú paraméter adható meg, amelyeknek kiderül a funkciója, és amelyeket a környezet másol be a programkód megfelelő helyeire (Imagine Logo) (4. ábra, bal oldal).

A szöveges fejlesztői környezetek egy másik része a kurzornál található nyelvi elem paraméterezéséről egy helyzetérzékeny súgóban tájékoztat (Small Basic), ebben az esetben azonban megadható tévedésből túl sok vagy túl kevés, esetleg hibás típusú vagy rossz helyre írt paraméter (4. ábra, jobb oldal). És vannak olyan fejlesztői környezetek is, amelyek egyik lehetőséget sem kínálják fel (RoboMind).




**4. ábra:** A kurzornál található utasítás paraméterezését segítő űrlap az Imagine Logóban, illetve helyzetérzékeny súgó a kurzornál lévő utasítás paraméterezéséről a Small Basicben

## 2.3. A vezérlési szerkezetek létrehozása

Hasonló a helyzet a vezérlési szerkezetek létrehozásával kapcsolatban. A blokkalapú nyelvekben a vezérlési szerkezetek kulcsszavai és utasításblokkjai együtt alkotnak egy blokkot, ezért csak a megfelelő mennyiségű utasításblokk adható meg a számukra, amelyekbe csak utasítások illeszthetők (ScratchJr, Scratch és Alice) (5. ábra).


**5. ábra:** Elágazás a Scratch-ben

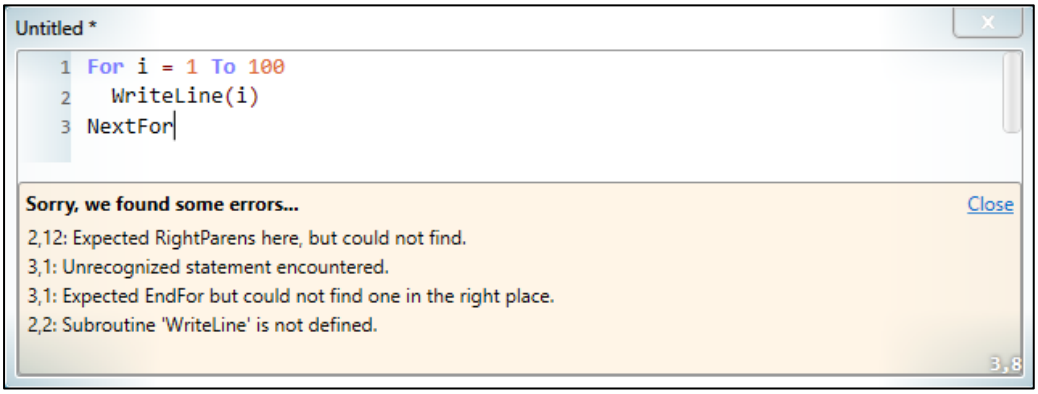
A szövegalapú kódolás során helyzetérzékeny súgó informálhatja a programozót a szükséges utasításblokkok számáról (Small Basic), de ennek ellenére megadható tévedésből túl kevés vagy túl sok utasításblokk (például a ha–akkor–különben szerkezet számára csak egy), és azokba nemcsak utasítások írhatók.

## 2.4. Szintaktikai helyesség és a szintaktikai hiba jelzése

A blokkalapú programozás során tehát csak létező és jól paraméterezett nyelvi elemek használhatók, a vezérlési szerkezetek is csak a megfelelő számú, és kizárólag utasításokat tartalmazó utasításblokkokból állíthatók össze, továbbá nincsen szükség azokra a határolójelekre, amelyekkel a szöveges programozási nyelvekben például az utasításokat és a paramétereket tagolják. Ezeknek köszönhetően a blokkokból csak szintaktikailag helyes programkód állítható össze. A programozó azzal, hogy valamely lépése szintaktikai hibát okozna (például számadatot szeretne felhasználni logikai feltételként), anélkül szembesülhet, hogy a hiba ténylegesen létrejönne, és később a futó program hibaüzenettel leállna (ScratchJr, Scratch és Alice). Így a tévesztés nem jár kudarccal, a visszacsatolás mégis azonnali (6. ábra).


**6. ábra:** Számérték nem, de logikai értékű kifejezés lehet az elágazás feltétele a Scratch-ben

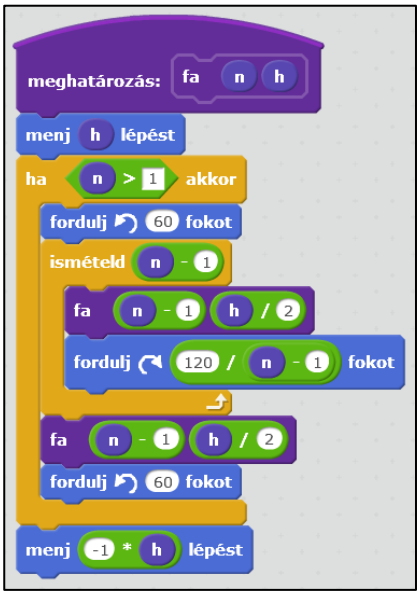
A szövegalapú programozás közben a fentebb ismertetett segítségek ellenére is (ha egyáltalán azok rendelkezésre állnak) szintaktikai hibák véthetők. A tévesztésekre csak a programfutás indításakor vagy megszakadásakor megjelenő hibaüzenetek hívják fel a figyelmet (Imagine Logo, RoboMind és Small Basic). Mivel a visszajelzések késnek, a hiba helye és oka utólag nem mindig egyértelmű, a sok hibaüzenet pedig frusztráló lehet (7. ábra).




**7. ábra:** Megjelenő hibaüzenetek a program indításakor a Small Basicben

## 2.5. Olvashatóság

A blokkalapú programozási nyelvekben még az összetett kifejezések esetén is jól látható, hogy az egyes értékek és részkifejezések mely nyelvi elemnek a paraméterei (Scratch és Alice) (8. ábra, bal oldal). A szövegalapú kódolásban általában a zárójelezés segíti az összetett kifejezések kiolvasását (RoboMind és Small Basic), a zárójelpárokat azonban meg kell találni. Ráadásul az oktatási célú nyelvek között található olyan is, amelyben a paraméterek – a nyelv más szempontú egyszerűsítése érdekében – nincsenek zárójelbe téve (Imagine Logo) (8. ábra, jobb oldal).



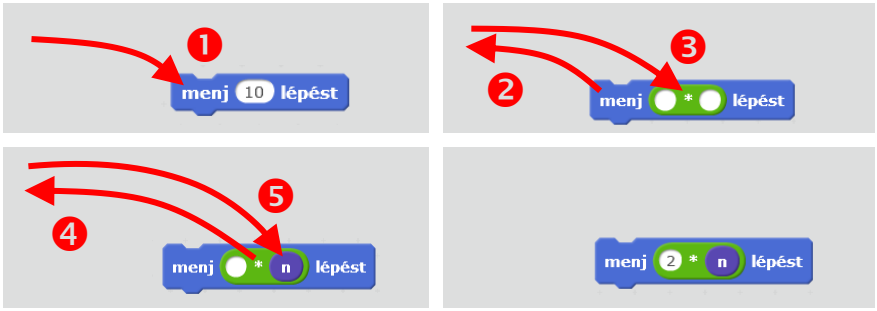

```
eljárás fa :n :h

előre :h
ha :n > 1 [
balra 60
ismétlés :n - 1 [
fa :n - 1 :h / 2
jobbra 120 / (:n - 1)]

fa :n - 1 :h / 2
balra 60]

hátra :h
vége
```

**8. ábra:** Ugyanazon feladatot végrehajtó program egymásnak megfeleltethető sorai a Scratch-ben és az Imagine Logóban

A blokkalapú nyelvekben az utasításblokkok egymásba ágyazásai is jól áttekinthetők, programozói beavatkozás nélkül (ScratchJr, Scratch és Alice) (8. ábra, bal oldal). Ugyanezt a szövegalapú kódolás során a különböző mértékű behúzások alkalmazásával szokták elérni. Ha azonban a programozó nem igazítja a sorokat, az utasításblokk-struktúra nehezebben áttekinthetővé válik (8. ábra, jobb oldal). Néhány fejlesztői környezet ezért felkínálja a sorok utólagos automatikus igazításának a lehetőségét (Small Basic és RoboMind).

## 2.6. A bevitel sebessége

A blokkalapú programozási környezetekben egy új nyelvi elem beszúrásához először ki kell választani a blokk-készlet megfelelő kategóriáját, majd azon belül a szóban forgó nyelvi elemet, amelyet végül be kell húzni a programkód megfelelő helyére (ScratchJr, Scratch és Alice). Így kell eljárni minden egyes kulcsszó, azonosító, sőt operátor esetén is (9. ábra).




**9. ábra:** A jobb alsó fázisban látható utasítás összeépítéséhez szükséges egérmozgások a képernyő bal szélén található (a képen nem látható) blokk-készlet és a programozási terület között a Scratch-ben

Ugyanezek a nyelvi elemek egy szövegalapú programozási nyelv fejlesztői környezetében gyorsabban begépelhetők (Imagine Logo, RoboMind és Small Basic), még akkor is, ha nem áll rendelkezésre az automatikus kódkiegészítés funkció.

## 2.7. Terjedelem

A blokkokból összeállított programkód lényegesen több helyet foglal el a vele egyenértékű szöveges kódnál. Ez a blokkok méretén túl annak is köszönhető, hogy a blokkokat csak egymás után lehet csatlakoztatni (ScratchJr, Scratch és Alice), miközben a legtöbb szövegalapú programozási nyelvben az utasítások egy sorba is írhatók (Imagine Logo és RoboMind).

## 2.8. Szerkesztés más környezetben

A blokkalapú programkódok csak a fejlesztői környezeten belül szerkeszthetők, és azon kívül csak képként használhatók fel (ScratchJr, Scratch és Alice). Ezzel szemben a szöveges programkódok tetszőleges kód- vagy szövegszerkesztőben módosíthatók, majd onnan vissza is másolhatók a fejlesztői környezetbe (Imagine Logo, RoboMind és Small Basic).

# 3. A kétféle kódolás értékelése

A kezdő programozók számára a blokkalapú programozás a megfelelőbb: elsősorban azért, mert nem lehet szintaktikai hibát ejteni, a blokkok formái pedig fokozzák a programkód olvashatóságát. További előnyt jelenthet, hogy a blokkokon szerepelhetnek piktogramok – ezáltal elérhetővé téve a kódolást a legkisebbek számára –, vagy pedig az élőbeszédhez közelebb álló bővebb kifejezések is, mivel azokat nem kell begépelni. Az önálló kísérletezést támogatja az is, hogy az összes nyelvi elem elérhető a blokk-készletben. Kutatások szerint is a kezdő programozók számára könnyebb a blokkalapú programozás [2], és a szövegalapú programozásban mérhetően jobb eredményeket érhetnek el azok, akik korábban blokkalapú programozást tanultak [3].

Ugyanakkor nem véletlen, hogy a professzionális programozásra a szöveges kódolás a jellemző: a billentyűzethasználatban gyakorlott programozók gyorsabban viszik be a szöveget gépelés útján. A haladó programozók számára, akiknek a szintaktikai szabályok betartása már kisebb gondot jelent, és akik már jelentős mennyiségű programot állítanak elő, a szövegalapú kódolás a hatékonyabb.

# 4. Az interjúalanyok véleménye a kétféle kódolásról

Az interjú olyan kvalitatív kutatási módszer, amely mélyre ható kikérdezést tesz lehetővé, és ezáltal nézőpontokat, indoklásokat és összefüggéseket tárhat fel a kutató számára [4]. Az első interjúalanyom négy éve oktat 2–8. osztályos gyermekeket egy programozóiskolában egy olyan oktatási platformon, amelyben a szövegalapú és a blokkalapú kódolás is lehetséges. A második hatosztályos elitgimnáziumban tanít a normál órákon szövegalapú, programozószakkörein azonban blokkalapú programozási nyelveket is. A harmadik egy szintén jónevű, nyolcosztályos gimnázium informatikatanára, aki azonban csak szövegalapú programozási nyelveket tanít.

Anélkül, hogy alanyaimnak konkrét szempontokat adtam volna, arra kértem őket, hogy hasonlítsák össze a kétféle beviteli módszert, saját tapasztalataik alapján. Az interjúk anonim módon hangrögzítéssel készültek, feldolgozásukhoz pedig a kategorizáció [5] módszerét használtam, amelynek megfelelően először a blokkalapú, majd a szövegalapú kódolás mellett szóló érveket csoportosítottam.

## 4.1. A blokkalapú kódolás mellett szóló érvek

Interjúalanyaim megfogalmazták, hogy a blokkalapú programozás során **a nyelvi elemek könnyebben elérhetők.** „Nem szükséges tudni, hogy milyen utasítások léteznek, azok a menüben megtalálhatók, tanári segítség nélkül is." (3. interjúalany)

Kiemelték, hogy a blokkokból könnyebben állítható elő **szintaktikailag helyes programkód.** „A blokkos programozás azért jó, mert szintaktikai hibát nem lehet ejteni, csak szemantikait; ebből a szempontból jobb, mint a szöveges kódolás." (1. interjúalany) „Könnyebb vele hibátlan kódot előállítani kezdő programozóként." (2. interjúalany)

A blokkokból összeállított **kód jobb olvashatóságáról** is beszéltek. „[A blokkalapú programozás] véleményem szerint áttekinthetőbb is [mint a szövegalapú]: például jobban képes vizualizálni az egymásba ágyazott ciklusok vagy elágazások viszonyát. A kód igazítása minden esetben hibátlan és egységes, míg például a szöveges kódolás igazítására többféle szokás terjedt el, amelyek közül néhány gyakran megnehezíti számomra a kód olvasását." (1. interjúalany)

Lényegesnek tartották, hogy a blokkokkal **kiváltható a gépelés.** „Egy alsó tagozatos tanuló számára nagy könnyebbséget jelent, ha nem kell begépelnie az utasításokat, legfeljebb csak a paramétereiket." (1. interjúalany) „[A blokkalapú nyelvek] képesek elérhetővé tenni a programozást a betű-

ket és a számokat még nem ismerő kisgyermekek, valamint a rendesen még gépelni nem tudó fiatalok számára, körülbelül 4-6. osztályig." (2. interjúalany)

### 4.2. A szövegalapú kódolás mellett szóló érvek

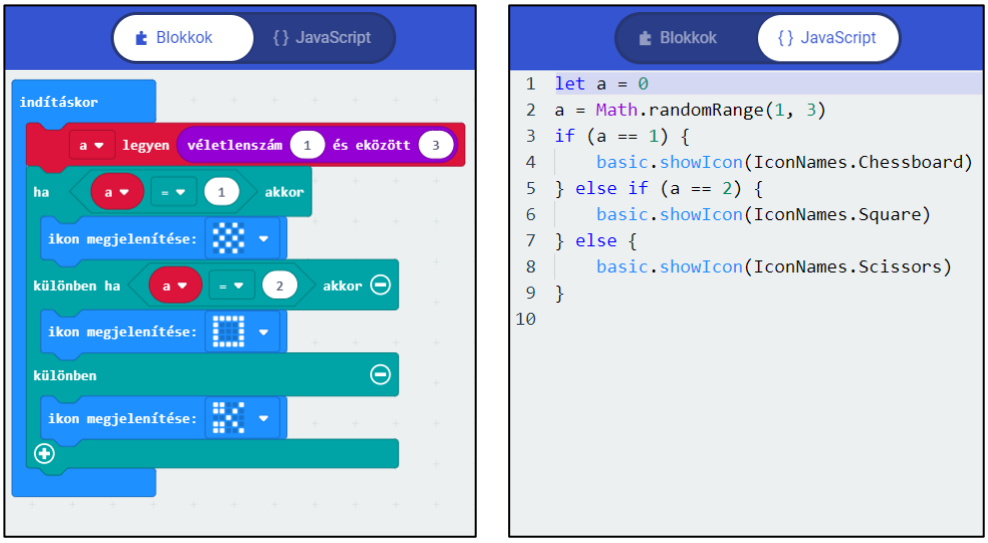
Interjúalanyaim rámutattak a szövegalapú kódolás előnyeire is, így például a **gyorsabb bevitelre.** „Egy nagyon egyszerű problémát nyilván nagyon könnyű blokkokkal megoldani. Ahogy azonban nő a feladat komplexitása, úgy csökken az, hogy mennyire hatékony ez a kódlétrehozási módszer, hiszen időigényessé válik a megoldás »összelegózása« az egérrel, a gépelés helyett." (1. interjúalany) „Egy »e« betűt leírni, és utánaírni a számot, sokkal gyorsabb, mint egy panelon kikeresni az előreutasítást, egérrel behúzni a képernyőre, majd utána ugyanúgy begépelni a megfelelő helyre a paramétert." (3. interjúalany)

Rávilágítottak továbbá, hogy praktikusabb a szöveges programkódok **kisebb terjedelme.** „Egy nagyobb alkalmazás kódját összerakni és áttekinteni is nagyon nehéz a blokkalapú környezetekben." (2. interjúalany) „Ha már nagyon sok doboz van a képernyőn, akkor sem a távoli, sem a közeli nézetben nem lehet jól áttekinteni a programot, és gyakran nem világos, hogy mikor mi történik. Természetesen a szöveges kód is lehet hosszú, és az sem feltétlenül fér el egy képernyőn, ugyanakkor ott motiváltabb az eljárások bevezetése, amelyekre a megfelelő helyen elég hivatkozni. Azt gondolom tehát, hogy amikor már egy bizonyos szint felett van egy tanuló, a szöveges programozás átláthatóbb." (3. interjúalany)

Interjúalanyaim tapasztalatai, érvei és ellenérvei alátámasztják a korábbi értékelést.

## 5. Áttérés a blokkalapú kódolásról a szövegalapúra

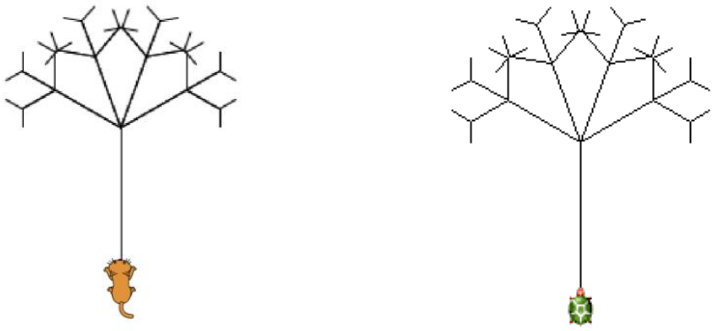
A blokkalapú programozásról a szövegalapúra áttérni egy olyan fejlesztői környezetben a legkönnyebb, amely támogatja a kétféle kódolási mód közötti folyamatos konvertálást, vagyis amelyben ugyanaz a (helyes) programkód bármikor megtekinthető és szerkeszthető az egyik vagy a másik nézetben. Ilyen például a Micro:bit (microbit.org) miniszámítógép Microsoft MakeCode (makecode.microbit.org) elnevezésű kódszerkesztője, amelyben egy blokkalapú, valamint egy a JavaScript szintaktikájára épülő szöveges programozási nyelv között lehet váltogatni (10. ábra).




**10. ábra:** Ugyanaz a program blokkos és szöveges nézetben a Microsoft MakeCode-ban

Természetesen a szintaktikailag hibás szöveges programkód nem futtatható és nem is konvertálható vissza a blokkalapúba. Ebben az esetben kérhetjük a legutóbbi hibátlan változat visszaalakítását blokkokká.

De a blokkalapú programozásról a szövegalapúra áttérés akkor is elképzelhető, ha a blokkalapú fejlesztői környezet nem biztosítja a kétféle kódbevitel közötti váltogatást. Annak érdekében, hogy egyszerre csak egy újdonság legyen – tudniillik a megváltozott programozási nyelv –, célszerű a programozást ugyanazon témakörön belül folytatni. Erre biztosít lehetőséget például a Scratch és az Imagine Logo programozási nyelve, az előbbi ugyanis blokk-, az utóbbi pedig szövegalapú, és mindkettővel megoldható a teknőcgrafika számos feladattípusa (11. ábra). Elérhetők bennük az automata elven irányítható és tollal rendelkező objektumok (szereplők, illetve teknőcök), mozgatóutasításaik és a tollhasználat parancsai pedig javarészt megfeleltethetők egymásnak.



**11. ábra:** A 8. ábrán látható, egymásnak utasításonként megfeleltethető programkódok futtatásának eredménye a Scratch-ben és az Imagine Logóban

A Logo versenyfeladatok megoldása a Scratch programozási nyelven [6] című kiadványom segítséget nyújthat a két nyelv közötti váltásban, amelyben összehasonlítottam a két programozási nyelv teknőcgrafikai lehetőségeit, és számos Logo-versenyfeladat megoldását ismertettem a Scratch-ben, feladattípus szerinti csoportosításban.

## 6. Az interjúalanyok tapasztalatai az áttérésről

Első interjúalanyom arról számolt be, hogy programozóiskolájuk diákjai „a szöveges kódolást és annak kulcsszavait nagyon izgalmasnak találják, és nagyon hamar meg akarják tanulni az összeset”, és „nem szokták »visszasírni« a blokkalapú kódbevitelt”. Ez vélhetően annak köszönhető, hogy jórészt a programozásra nagyon fogékony tanulók járnak abba az iskolába.

A gimnáziumban tanító két beszélgetőpartnerem azonban arról vallott, hogy a blokkosan programozó diákjaikat sokszor nekik kell motiválttá tenniük. „Akinek a blokkalapú programozás jól megy, és azt szereti, az önmagától nem igazán szeretne áttérni a szöveges kódolásra. […] Néhányan rájönnek például egy egyszerű prímszámkeresési feladat megoldása során arra, hogy azt a Scratch-ben nem lehet hatékonyan megvalósítani. És akkor közösen kereshetünk egy olyan programozási nyelvet, amelyen hatékonyabban lehet programozni. Ugyanakkor az a tapasztalatom, hogy magától a Scratch-et használók többsége nem jut el eddig a gondolatig.” (2. interjúalany) „Fájdalmas volt a számára [az áttérés], ugyanis a blokkok használatában már nagyon rutinos volt. Gépelni pedig nagyon nem szeretett.” (3. interjúalany) Második interjúalanyom viszont kiemelte, hogy a szövegalapú kódolás során a korábban blokkalapon programozó diáknak „a programozásról és a programok felépítéséről kialakult szemlélete nagyon jól jön, azt nem kell újra tanítani. Könnyen megérti a vezérlési szerkezeteket, és nagyobb nehézségek nélkül megszokja a szintaktikai szabályokat is.”

## 7. Konklúzió

Cikkemben a kétféle kódlétrehozási lehetőséget az oktatási célú programozási nyelvek más adottságaitól függetlenül hasonlítottam össze és értékeltem, amelynek alapján megállapítottam, hogy a kezdők számára a blokkalapú, a haladók számára viszont a szövegalapú kódbevitel az előnyösebb.

Úgy gondolom, hogy a kezdőknek életkortól függetlenül is a blokkos programozás javasolható, amelyet nemcsak a köznevelésben, de a felsőoktatásban is egyre bővülő körben használnak a bevezető programozástanításban [7]. Személyes meggyőződésem, hogy habár a haladó programozás során a szöveges kódolás egyértelműen hatékonyabb, bármely programkód blokkalapú nézete és szerkesztése közelebb áll a programozási nyelv struktúrájához az egyszerű szövegnél.

## Irodalom


1. Kelleher, C. & Pausch, R., 2005. *Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers.* ACM Computing Surveys, 37(2), pp. 83-137.
2. Weintrop, D. & Wilensky, U., 2015. *To block or not to block, that is the question: students' perceptions of blocks-based programming.* 14th International Conference on Interaction Design and Children, pp. 199-208.
3. Armoni, M., Meerbaum-Salant, O. & Ben-Ari, M., 2015. *From Scratch to "real" programming.* Transactions on Computing Education, 14(4).
4. Falus, I. & Ollé, J., 2008. *Az empirikus kutatások gyakorlata – Adatfeldolgozás és statisztikai elemzés.* Oktatáskutató és Fejlesztő Intézet, Budapest.
5. Schleicher, N., 2007. *Kvalitatív kutatási módszerek a társadalomtudományban.* Századvég, Budapest.
6. Bernát, P., 2017. *Logo versenyfeladatok megoldása a Scratch programozási nyelven*. ELTE Informatikai Kar, Budapest.
   http://logo.inf.elte.hu/tanaroknak/logo-scratch%20v11.pdf (utoljára megtekintve: 2019.11.10.)
7. code.org, 2014. *Why top universities teach drag and drop programming.*
   https://www.youtube.com/watch?v=_Mwc1gc77dc (utoljára megtekintve: 2019.11.10.)

# Programozás tanítási módszerek – stratégia a kezdetekre

Bernát Péter[1], Zsakó László[2]

[1]bernatp@inf.elte.hu, [2]zsako@caesar.elte.hu
ELTE IK

**Absztrakt.** A programozás tanítását alapvetően feladattípus-orientáltan képzeljük el [5], de nagyon sok függ a kiválasztandó feladattípusoktól. E cikkben azt járjuk végig, hogy az egyes választások milyen előnyökkel és hátrányokkal járhatnak. A következő stratégiákat nézzük végig: hétköznapi algoritmusok; teknőcgrafikára épített programozás, animációkészítésre épített programozás, játékfejlesztésre épített programozás, robotikára épített programozás; matematikára épített programozás.

**Kulcsszavak:** programozástanítás, feladattípus-orientált módszer, hétköznapi algoritmusok, teknőcgrafikára épített programozás, animációkészítésre épített programozás, játékfejlesztésre épített programozás, robotikára épített programozás; matematikára épített programozás

## 1. Bevezetés

A programozási módszertan az informatika oktatása egyik legrégibb területe, így többféle, ma is használatos módszer alakult ki tanítására. Ezek közül egyesek az általános, illetve középiskolai oktatásban használhatóak eredményesen, mások pedig a felsőoktatásban.

A 70-es évek végén más megközelítésben is előkerült a programozás tanításának módszertana a gyermeki gondolkodásra alapozva, ennek kialakítója elsősorban S. Papert [1]. Hasonló, bár más feladatkörre alapozott módszerről ír J. Hvorecky és J. Kelemen [2].

Koncepciójuk alapján a következők terjedtek el:

A. **Módszeres, algoritmusorientált** (a módszer az algoritmus megtervezését, megértését teszi a programozási folyamat középpontjába);
B. **Adatorientált** (a módszer az adatstruktúra megtervezését, megértését teszi a programozási folyamat középpontjába, ehhez alkot algoritmust, majd kódot);
C. **Specifikációorientált** (a módszer a specifikálást teszi a programozási folyamat középpontjába, s mindent ebből vezet le);
D. **Feladattípus-orientált** (a módszer egyre bővülő feladatsorokon keresztül vezeti be az új programozási fogalmakat, módszereket);
E. **Nyelvorientált** (a módszer egy programozási nyelv elemein keresztül tanítja meg a programozást);
F. **Utasításorientált** (a módszer egy általános programozási nyelv osztály elemein keresztül tanítja meg a programozást);
G. **Matematikaorientált** (a módszer egy másik tantárgy – pl. a matematika – lehetséges tananyagához vezet be programozási feladatokat, melyeket felfedezés-szerűen old meg);
H. **Hardverorientált** (a módszer alapgondolata, hogy programozási nyelv nélkül nem érthető a programozás, assembly nyelv nélkül nem érthető a programozási nyelv, … és így tovább egészen a számítógép működés fizikai alapjaiig);
I. **Mintapélda alapján** (a módszer kész programok elemzésén keresztül tanítja meg a programozást).

P. Szlávi és L. Zsakó szerint [5] a mindenkinek szóló informatika oktatásban a *D. Feladattípus-orientált* módszer az igazán jól használatos.

## 2. Feladattípus-orientált programozás tanítási módszer

A feladattípus-orientált programozás tanítási módszer sikeressége nagyban függ attól, hogy milyen feladattípusokat választunk, milyen stratégiával állunk hozzá a programozás tanításához.

A klasszikus programozás tanításában ez tipikusan matematikai feladatkör, legtöbbször számelméleti feladatokkal (pl. oszthatóság, prímszámok, prímtényezős felbontás).

Más stratégiát is választhatunk, de mindegyik lényege az, hogy egy egymásra épülő példákat tartalmazó feladatsort kell megoldanunk. A feladatsor egyes feladatai megoldásához van szükségünk új programozási fogalmakra, elemekre, s ezeket azért vezetjük be, mert a konkrét feladatmegoldáshoz kellenek. Ennek előnye, hogy az új ismeretet természetes igényekből kiindulva vezethetjük be, s nem kinyilatkoztatásként. Ugyanakkor azonnal alkalmazzuk is a feladat megoldására, s mint közismert, a megértés egyik legmagasabb szintje az, amikor az új ismeretet alkalmazni is tudjuk.

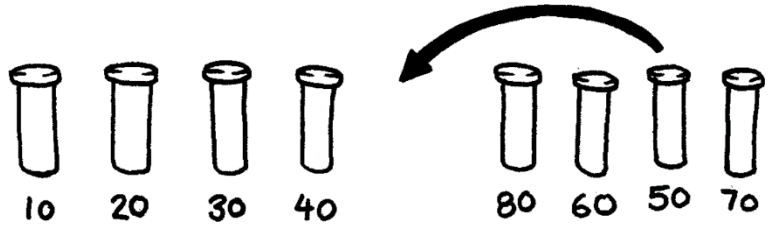
Ebben a cikkben a következő – feladattípus-orientált koncepcióra építő – stratégiákat fogjuk vizsgálni:

1. Hétköznapi algoritmusok
2. Teknőcgrafikára épített programozás tanítás
3. Animációra épített programozás tanítás
4. Játékfejlesztésre épített programozás tanítás
5. Robotikára épített programozás tanítás
6. Matematikára épített programozás tanítás

Az egyes stratégiákat elsősorban a következő szempontok szerint tárgyaljuk: a megtanítható programozási fogalmak és módszerek köre; kapcsolódási lehetőségek az informatika egyéb témaköreihez és más műveltségi területekhez; a továbblépés lehetőségei más nyelvek és az informatikus szakma irányába; a szociális kompetenciák fejlesztésének lehetősége; motivációs szempontok.

## 3. Hétköznapi algoritmusok

Mindennapi tevékenységeink során algoritmusokat hajtunk végre, amelyekben gyakran felfedezhetők az algoritmizálás és az adatmodellezés alapvető fogalmai és módszerei. Ezek az ismeretek megfelelően kiválasztott hétköznapi algoritmusokkal számítógép használat nélkül is bevezethetők.




**1. ábra:** Különböző súlyú skatulyák rendezése a minimum-kiválasztás módszerével [12]

Az óvodákban és az általános iskolákban hétköznapi algoritmusok megértését és végrehajtását már nagyon régóta tanítják, csak ennek sokáig semmi köze nem volt az informatika algoritmizálás tanításához [3].

Megjegyezzük, hogy azt az elképzelést, hogy az algoritmus úgy működik, ahogyan magunk is végrehajtanánk, sok programozás tanítási módszer alkalmazza.

A hétköznapi algoritmusokra építő stratégiával bevezethetők a következő, a mindennapi tevékenységeinkben is előforduló algoritmikus elemek és adatstruktúrák:

- szekvencia, elágazás, ciklus;
- eljárás;
- programozási tételek;
- elemi adatok, összetett adatok;
- sor, prioritási sor, lista, fa, gráf;
- objektum.

Nemigen találni példát azonban a változófogalomra és az értékadásra.

Különösen ez a stratégia támogatja a programozással való kezdeti ismerkedést (általános iskola alsó tagozata, sőt: óvoda), hiszen a mindenkiben meglévő természetes ismeretekre épít, azokból vezeti le a programozási fogalmakat, módszereket. [6]

Játékos, gyakran párban vagy csoportban végrehajtandó feladatok kapcsolódhatnak hozzá [12], amelyeknek nemcsak a megértésben és a begyakorlásban, de a tanulók szociális készségeinek fejlesztésében is szerepük lehet.

A módszer nagy előnye tehát, hogy természetes, hétköznapi tevékenységeken keresztül vezet be a programozás gondolatvilágába, de nagy problémája, hogy számítógép nélkül gyorsan unalmassá válik.

## 4. Teknőcgrafikára épített programozás tanítás

A teknőcgrafikát, mint a programozás tanításában jól használható módszert S. Papert [1] vezette be, már több évtizeddel ezelőtt. Újabban J. Hromkovic is hasonlókat fogalmaz meg [9].

A teknőc utasítható a számára „érthető" feladatok elvégzésére: adott távolsággal előre vagy hátra tud menni, adott szöggel jobbra vagy balra elfordulni, tollát (ami a hasára van erősítve) felemelni, leereszteni, más színűre cserélni, ezáltal mozgásával érdekes nyomokat hagyni a képernyőn. A teknőcgrafika fokozatosan egy új tudományterület, a teknőc-geometria megszületéséhez vezetett.

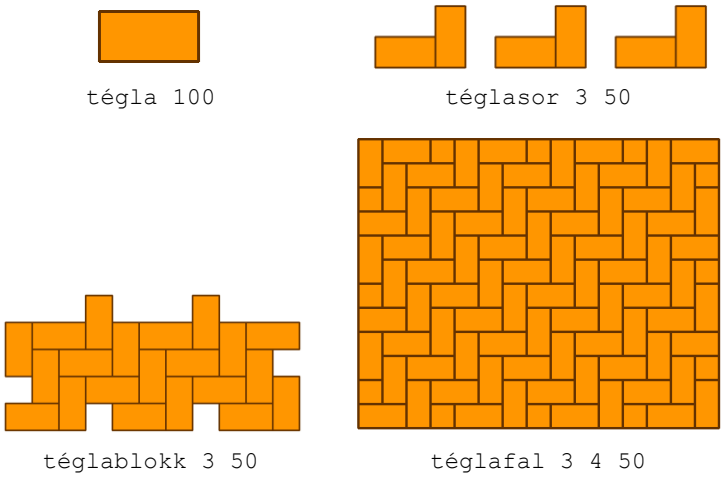
Alapgondolatai

- a programozás itt automataelvű – a végrehajtó teknőc egy automata, aminek helyébe saját magunkat is beleképzelhetjük;
- a specifikáció lényegi része egy rajz, a rajz elkészítése úgy történik, ahogy magam csinálnám.

Teknőcgrafika nagyon sok nyelvben van, így megvalósítása nem igényli speciális nyelv megtanulását, bár nyilvánvalóan a Logo programozási nyelvben vezették be, a nyelv erre a fogalomvilágra épül. A többi nyelvet általában ezzel a nyelvi eszközkészlettel bővítették.

A specifikáció itt pontos, betartása ellenőrizhető, hiszen a specifikáció maga egy rajz. Emiatt a feladatok látványosak, a szekvencia, az elágazás, a ciklus a rajzon látszik.

A teknőcgrafikában az eljárás a strukturálás alapeszköze, ahol az eljárások a rajzos specifikáció alapján megfogalmazhatók, mint részrajzok. (2. ábra)




**2. ábra:** Logo verseny mozaikos feladat

A rajz alapján a (paraméteres) eljárásokra bontás logikus, elvégezhető (még a rekurzió is), a paraméterek megjelenése következik a rajzos specifikációból. (3. ábra)

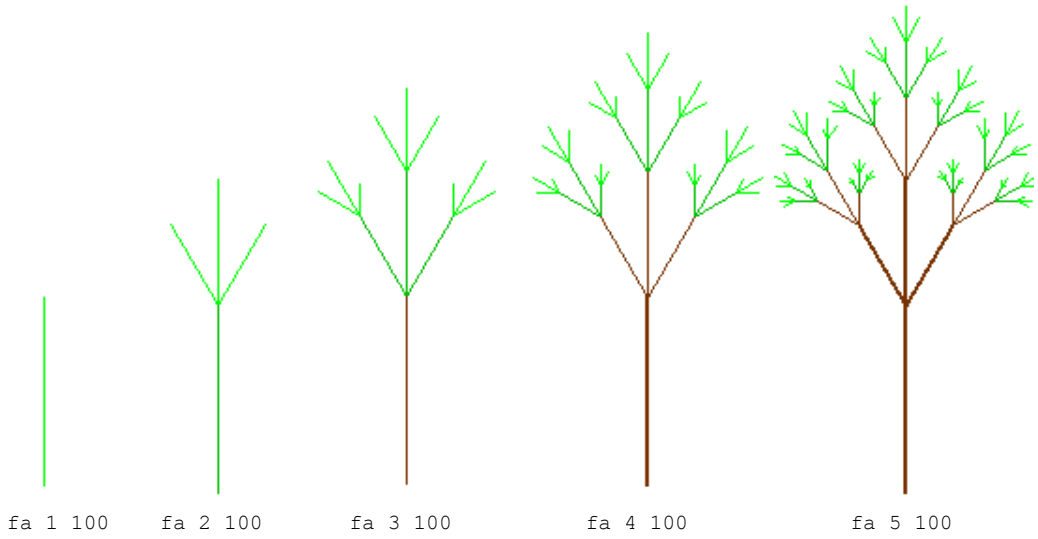
A témakör nagyon kevés nyelvi elemmel megvalósítható (Logo-ban: `előre`, `balra`, `toll-szín!` a minimális utasításkészlet, de a teljes készlet sem túlságosan nagy). Emiatt a gondolkodást helyezi előtérbe a lexikális ismeretekkel szemben. A program nem utasítások hosszú lineáris sorozata, hanem – mint a szakmában általában – mélyen struktúrált szerkezet (azaz a sok gondolkodás mellett kevés gépelés kell hozzá). Alkalmas kevés tanulás után is „komoly" feladatok megoldására.

A rajzos programozás pozitív hatása – ami nem csak a teknőcgrafikára igaz -, hogy a hibás algoritmus lehet érdekes, a hibás rajz segítheti a hibakeresést. Akkor különösen jó ez a technológia, ha a felülről lefelé tervezett megoldásokat ki lehet alulról felfelé próbálni.

A megoldható feladatok miatt jól használható matematikai fogalmak tanítására, gyakorlására (a matematikát segítheti), kapcsolható koordinátageometriához.

A témakör jellegzetessége miatt sokféle feladattípus megvalósítható vele (pl. sokszögek, spirálok, sorminták, mozaikok, fraktálok, ásványszerkezetek, optikai csalódások, intarziák, mandalák, frízek stb.).

Továbblépési lehetőség a párhuzamos programozás felé: lehetőség több automatára (teknőcök). A színkezelés megoldása funkcionálisan, összetett típus kezelésével lehetséges, bevezethető vele a funkcionális programozás. A Logo alapú szimulációra remekül használható egy továbbfejlesztése, a netlogo.




**3. ábra:** Logo verseny fás feladat

Hátrányai közé tartozik, hogy logikusan nincs változó és értékadás, nincsenek típusok, nincsenek összetett típusok (csak, ami funkcionális nyelvekben lehet), továbbá pixelgrafikus feladatok megoldására nem jó.

Az objektumorientáltság ugyan megjelenik, de gyengén objektumorientált (teknőc).

## 5. Animációra épített programozás tanítás

Ennél a stratégiánál a programozási feladat egy animáció létrehozása. (4. ábra) Az animáció egyfajta forgatókönyv (mint specifikáció) alapján készülhet akár jelentős mennyiségű képi és hangi alapanyag felhasználásával, amelyek elkészítése is a feladathoz tartozhat. [7]

**4. ábra:** Egy jelenet az Alice programozási környezetben [7]

Animációk célszerűen objektumorientált programozással készíthetők. Ebben az esetben objektumok a szereplők, a díszlet elemei és az egyéb összetevők is (például kamerák és fényforrások). Az objektumok sokféle látványos tulajdonsággal és művelettel rendelkezhetnek, így segítségükkel szem-

léletes és közérthető módon vezethetők be az objektumorientált programozás alapfogalmai. (A kialakult képet azonban a későbbiekben nyilvánvalóan árnyalni kell, hiszen a programozás során használt objektumok az előzőeknél gyakran elvontabbak.)

Ebben a koncepcióban az eljárások a jelenetek, valamint a jelenetekben szereplők műveletei. Praktikus, ha a jelenetek önállóan is kipróbálhatók, a teljes film megalkotása nélkül. A jelenetek lineárisan követik egymást, ezért a program szekvenciális felépítésű. Nem jellemző a mély strukturálás, nincs rekurzió. Emiatt fennáll a „spagetti kód" alkotásának veszélye.

A szereplők tevékenységeit a jeleneteken belül folyamatosan össze kell hangolni, ami a pontos időzítésen kívül a szereplők közötti üzenetváltásokkal történhet.

Az új funkciókhoz sokszor új nyelvi elemek kellenek. Ez egyfelől nagy lexikális tudást igényel, ugyanakkor a sokféle különböző típusú (például a szereplők mozgatásával, a kinézetük megváltoztatásával vagy a hangokkal kapcsolatos) műveletnek köszönhetően a vezérlési szerkezeteket is látványosan eltérő célokra lehet használni. Így ezek a fogalmak általánosabban mutathatók be, mint például a teknőcgrafikában.

A stratégia jól támogatja a felülről lefelé kifejtés, illetve az alulról felfelé építkezés módszerét is: például bevezethetők jelenetek és objektumok a kifejtésük későbbre halasztásával, de önmagukban megvalósított jelenetek és objektumok is beilleszthetők a történetbe.

Az animáció készítéssel a kép- és hangrögzítő eszközök használatán, valamint a kép- és a hangszerkesztésen kívül könnyen kapcsolódhatunk bármely műveltségi területhez: megeleveníthetők történetek, készíthetők szemléltetések, bemutatók bármilyen témakörben.

Az objektumok elhelyezéséhez (ideértve a szereplőket és a kamerákat is) koordinátageometriai ismeretekre van szükség (amihez logikusan tartoznak változók és értékadás).

A feladatok motiválhatnak és gyors sikerélményt adhatnak, amelyek különösen a művészetekre (irodalom, filmművészet, zene) nyitott, és esetleg az elvontabb problémák iránt kevésbé érdeklődő tanulók számára lehetnek vonzók. Ezenkívül kellően összetettek lehetnek ahhoz (főleg, ha a képi és hangi alapanyagokat is el kell készíteni), hogy érdemes legyen rajtuk csapatban dolgozni. Kérdéses, hogy mennyire „lányos" az animációkészítés, történet kitalálás, illetve hogy mely korosztály fogja azt mondani, hogy unják már ezt.

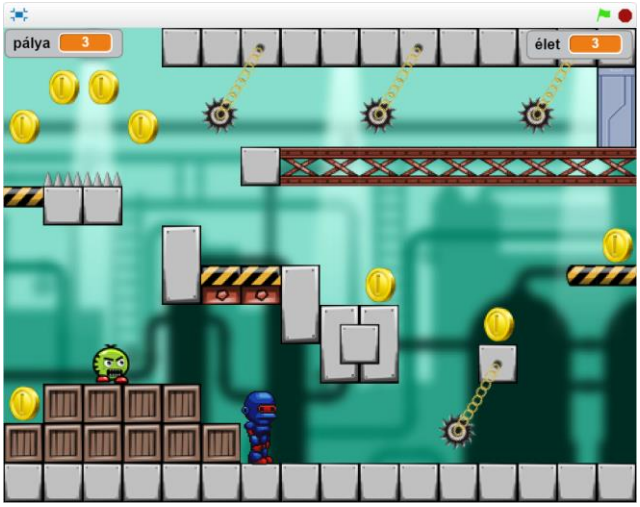
A stratégia problémái közé tartozik azonban, hogy bár az eredmény látványos lehet, de nem biztos, hogy megéri a látványosság adott szintje a befektetett munkát. Az igazán hatásos eredmény eléréséhez általában magas szintű (nem informatikai) ismeretekre van szükség. Következésképpen ez a stratégia az informatikai világból elvisz a filmkészítés világába, azaz nem sokáig segít az informatikus szakmára motiválásban.

## 6. Játékfejlesztésre épített programozás tanítás

A játék is bizonyos szempontból animáció, és az animáció készítésre alkalmas programozási nyelvekkel általában készíthetők játékok is. Az utóbbiak esetén azonban meghatározó a játékos és a játékprogram, továbbá a játékos és a számítógép által irányított szereplők közötti folyamatos interakció. Sőt, akár több játékos is játszhat ugyanazzal a játékkal, ami az online játékokra jellemző.

A játékfejlesztés tehát az animáció készítésre épül, ennek megfelelően az előző stratégia legtöbb előnye és hátránya itt is érvényes. Ebben a szakaszban elsősorban a különbségeket emeljük ki.

Mivel a számítógépes játékoknak a kezdetektől napjainkig nagyon sokféle műfaja alakult ki, könnyen meghatározhatók feladattípusok (például táblás- és kártyajátékok, autóversenyzős játékok, platformjátékok, építkezős játékok stb.) (5. ábra)




**5. ábra:** Egy platformjáték a Scratch programozási környezetben (saját játék)

A folyamatos interakciók miatt kiemelt szerephez jut az eseménykezelés. A játék és az egyes szereplők mindenkori állapotának nyilvántartásához pedig számos változóra és akár összetett adatszerkezetre is szükség lehet.

A játékfejlesztés módszerei az animáció készítésen alapulnak. Például a játékok is felbonthatók olyan jelenetekre, amelyekben a játék működése a többitől lényegesen eltérő (egy tipikus felbontás: főcím, menü, a játék futtatása, a játék vége). A játékkészítésnek azonban további játéktípus független, illetve játéktípus függő módszerei alakultak ki, amelyek ismerete szükséges az összetettebb játékok elkészítéséhez. Egyes játéktípusok kivitelezése csak jelentős matematikai és fizikai apparátussal lehetséges. Szükség lehet nagyon bonyolult, akár a mesterséges intelligencián alapuló viselkedésű szereplőkre.

A fentiek miatt a játékfejlesztés jó alapot és továbblépési lehetőséget biztosíthat az informatikus szakmák irányába.

A játékfejlesztés a számítógépes játékok megvalósítása iránt érdeklődők számára nagyon motiváló lehet. Mint ahogyan az is, hogy az elkészült játékokkal más is játszhat (például barátok). A játékkészítés esetén még indokoltabb lehet a feladat megosztása egy csoporton belül. Azt azonban figyelembe kell venni, hogy a szükséges ismeretek miatt a komolyabb játékok elkészítése a középiskolás korra, sőt annak is a végére tehető. Gy. Molnár és P Nyírő [11] egyetemi oktatásban próbálta ki a játékfejlesztésen alapuló programozás tanítási módszert, Game Maker-t használva.

Meggondolandó azonban, hogy a játékfejlesztés mennyire „fiús” tevékenység, segíti e a lányok nagyobb arányú megjelenését az informatikus szakmákban?

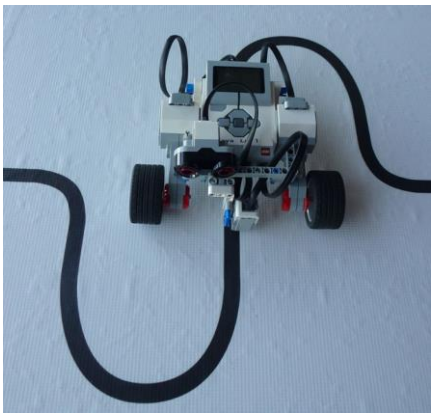
Ugyanennek a témának web-böngészőbeli megvalósításával foglalkozik Gy. Horváth, L. Menyhárt, L. Zsakó cikke [10].

## 7. Robotikára épített programozás tanítás

Ebben a stratégiában robotikai problémákat kell megoldani egy programozható robottal. Arra az alapgondolatra épít, hogy a fizikai eszközök programozása (az eszközök működésének megfigyelése) elősegíti az absztrakt programozási ismeretek kialakítását [8].

Az oktatási célú robotok jellemzően egyenesen haladni és kanyarodni képes szerkezetek, amelyekhez érzékelők is (elsősorban távolság-, ütközés-, fény- és színérzékelők) tartozhatnak. Az egyes

feladattípusokat az ilyen robotokkal megoldható problémakörök határozhatják meg (például területbejárás, átpakolás, útvonalkövetés, kijutás labirintusból stb.). (6. ábra)



**6. ábra:** Útvonalkövető LEGO robot [14]

A szóban forgó robotok néhány fajtája készre szerelt és a felépítésük nem módosítható, más típusok azonban a rendelkezésre álló elemekből mindig az aktuális problémának megfelelően építhetők össze. Szerelhető robot esetén a robotikai problémák szélesebb körével lehet foglalkozni, és a programozást jelentős mértékben kiegészítheti az elektromechanikai ismereteket igénylő robot- és pályaépítés. Nem szerelhető robot esetén a problémák szűkebb körével lehet foglalkozni, de a figyelem legnagyobb része a programozásra irányulhat.

Ezenkívül az oktatási robotok lehetnek valódiak vagy szimuláltak. A valódi robotok szó szerint kézzel foghatók és kapcsolatba kerülhetnek a valós világ tárgyaival, ezért működésük látványosabb. Egyszerre viszont csak kevesen tudják használni őket, és a program és a pálya módosítása körülményes lehet. Habár a szimulált robotok csak a virtuális térben léteznek, mindenki egyszerre használhatja őket, és a pálya és a program gyorsan változtatható és azonnal kipróbálható.

Programozási nyelvük a teknőcgrafikai programozási nyelvekhez hasonlóan automata elvű és eljárásorientált. A szenzorok állapotát jellemzően elágazásokkal lehet lekérdezni, de az eseménykezelés is elképzelhető. A változók és az adatszerkezetek használata azonban nem jellemző.

A robotikai problémák a tantárgyak közül elsősorban a matematikával és a fizikával lehetnek kapcsolatban. Bizonyos tanulók számára a fizikai világban mozgó robotok a többi stratégia virtuális tevékenységeinél motiválóbbak lehetnek. Ennél a stratégiánál is felmerül a párban vagy esetleg a kis csoportban dolgozás lehetősége: például a robot (és a pálya) megépítése, illetve a program elkészítése két külön részfeladat lehet.

A stratégiával kapcsolatban felmerülő probléma azonban, hogy az ipari robotok jelentős része nem mozog, a mozgó robotok pedig sokszor nem teknőcszerűen mozognak, hanem a mozgásállapotukat változtatják meg (pl. önvezető autók).

## 8. Matematikára épített programozás tanítás

Ez a stratégia a legrégibb, elsősorban a számelmélet témaköréből vett feladatokra épít, számelméleti feladatokat old meg (oszthatóság, prímszámok, számrendszerek), majd erre általában kombinatorikai feladatokat épít (faktoriális, Fibonacci számok, binomiális együtthatók). Újabban megjelentek a geometriai algoritmusok is.

Alapgondolata

- a matematikából ismert algoritmusok, módszerek megvalósítására koncentrál.

A matematika sok feladat algoritmusával, kiszámítási szabályával foglalkozik. Ezek sokasága alapvetően számtípusokra, valamint haladóbb szinten sorozatokra épít, amelyek megvalósítása nagyon könnyű bármely programozási nyelven.

A matematikai fogalmak használatából (osztó, prímszám, …) egyértelműen következik, hogy hogyan kell a programokat eljárásokra, függvényekre bontani.

Matematikai összefüggések alapján sokszor előkerülhet a rekurzió, mint a kiszámítás módszere, majd a rekurzió helyettesítése táblázatkitöltéssel (egyik tipikus példája a Fibonacci számok rekurzív összefüggése, majd táblázatkitöltéssel való megvalósítása.

Előnye és egyben hátránya is, hogy szorosan kapcsolódik a matematikához. Aki a matematikát nem szereti, azt ez a világ nem fogja meg. Aki viszont érti a matematikát, annak számára a matematikai algoritmusok megvalósítása nem okoz nehézséget, ezeken keresztül könnyedén megtanulja az algoritmizálási ismereteket.

Bár itt a „matematika" a címszó, természetesen más tantárgy is szóba jöhet, mint pl. P. Szlávi és L. Zsakó legelső tankönyvében [4], amely biológusoknak készült. Ebben olyan példák szerepelnek, hogy egy adott egyedről a genotípusa alapján állapítsuk meg, hogy heterozigóta-e, a vércsoportját megadó génpár szerint mondjuk meg, hogy A, B, AB vagy 0-s vércsoportú-e, két szülő vércsoportja alapján adjuk meg, hogy gyerekeik milyen vércsoportúak lehetnek, …

## 9. Záró gondolatok

Úgy gondoljuk, hogy nincs igazán üdvözítő módszer, amit kizárólagosan lehetne alkalmazni, minden esetben az előnyöket és hátrányokat mérlegelve kell dönteni.

Egyes módszerek kiválóan alkalmasak motiválásra, de a továbbhaladás velük nagyon nehézkes lehet. Más módszerek ezzel szemben a kezdeti lelkesedés után nagyon messzire is elvihetnek a programozás világában, a motiváció fenntartásával. Vannak módszerek, amelyek a gyerekek egyes rétegeinél nagyon sikeresek lehetnek, másoknál viszont csak kudarcba fulladhatnak.

Mindenképpen szem előtt kell tartanunk, hogy a programozás tanulás akkor lehet sikeres, ha érdekes és motiválhat az informatikus szakmákban való elmélyedésre, továbbá építkezhetünk rá a későbbi (akár szakmai) tanulmányok során.

## Irodalom


1. Seymour Papert: *Mindstorms. Children, Computers and Powerful Ideas*, Basic Books, Inc, Harper Colophon Books, 1981.
2. Josef Hvorecky, Josef Kelemen: *Algoritmizácia, Elementárny Úvod*, Alfa, Bratislava, 1983.
3. C.H.A. Koster: *Systematisch Leren Programmeren*, Educaboek, 1984.
4. Péter Szlávi, László Zsakó: *Bevezetés a számítástechnikába.* Tankönyvkiadó, Budapest, 1987.
5. Péter Szlávi, László Zsakó: *Methods of Teaching Programming.* Teaching Mathematics and Computer Science, Vol.1. 2003.
6. Juraj Hromkovic: *Contributing to General Education by Teaching Informatics.* In: Mittermeir, R.T. (Ed.), ISSEP 2006, Lncs, 4226, 25–37.
7. Caitlin Kelleher, Randy Pausch: *Using Storytelling to Motivate Programming.* Communications of the Acm, July 2007/Vol. 50, No. 7., pp 59-64.
8. Attila Pásztor, Erika Török, Róbert Pap-Szigeti: *Innovatív informatikai eszközök és módszerek a programozás oktatásban.* Gradus, Vol.1., No.1, pp: 22-27, 2014.
9. Juraj Hromkovic: *Einführung in die Programmierung Mit Logo*, 2009, ISBN: 3834810045

10. Győző Horváth, László Menyhárt, László Zsakó: *Viewpoints of Programming Didactics at a Web Game Implementation.* XXIX. Didmattech 2016, Eötvös Loránd University, Faculty of Informatics, Budapest, 2016, pp. 79-88.
11. György Molnár, Péter Nyírő: *A gyakorlati programozás tanításának játékfejlesztésen alapuló, élménypedagógiai alapú módszerének bemutatása.* In: Karlovitz János Tibor: Pedagógiai és szakmódszertani tanulmányok, pp: 89-98, 2016.
12. Bell, T.; Witten, I. H.; Fellows, M.: *Computer Science Unplugged, an Enrichment and Extension Programme for Primary-Aged Children*; http://Csunplugged.Org/
13. *Alice – An Educational Software That Teaches Students Computer Programming in a 3d Environment*; http://www.Alice.Org/Index.Php/
14. *Drgraeme.Org – Free Tutorials Lego Ev3 Mindstorms*; http://Www.Drgraeme.Org

# A robotika témakör integrálásának lehetőségei a természettudományos tantárgyak oktatásában

Gaál Bence

gaalbence@inf.elte.hu
ELTE Informatikai Kar

**Absztrakt.** A 21. század kihívásai miatt a természettudományok és az informatika ismerete még hangsúlyosabbá válik. Ennek ellenére sajnos a hazai trendek azt mutatják, hogy a diákok egyre kevésbé tartják fontosnak ezen ismereteket és egyre kisebb a motivációjuk, illetve kevesen választanak olyan szakmákat, amelyek ezen tudományterületekre épülnek. Erre a problémára megoldást nyújthat az, ha a különböző tantárgyakat ötvözzük az informatikával, azon belül is a robotikával, kiaknázva a STEM tárgyak azon belül is a természettudományos tantárgyak és a robotika kapcsolatának előnyeit. Cikkemben kitérek ennek a kapcsolatnak a mivoltára, valamint áttekintem, hogy a magyarországi kerettantervek alapján, melyek azok az anyagrészek, ahol érdemes és lehetséges a robotika alkalmazása, hogy a diákok játékosan, kézzel fogható eszközök és valós problémák segítségével sajátíthassák el a kapcsolódó ismereteket.

**Kulcsszavak:** természettudományok, robotika, tantervek, oktatás, természettudományos oktatás, informatikaoktatás

## 1. Hazai természettudományos tárgyak helyzetének, népszerűségének bemutatása

A téma szempontjából elkerülhetetlen kitérnünk a természettudományos tantárgyak hazai helyzetére, ezért megvizsgáljuk ezek helyét a tantervekben, valamint statisztikai adatok alapján következtetéseket vonunk le a diákok motiváltságával kapcsolatban. A statisztikai elemzés alapjául a tanulmányi versenyekkel és a felsőoktatási felvételik eljárásokkal kapcsolatos adatok szolgálnak.

### 1.1. Jogszabályi környezet

Hazánkban jogilag többféle meghatározás is életben van, abban a tekintetben, hogy mely tárgyak tartoznak a természettudományi tantárgyakhoz, attól függően, hogy közép-, vagy felsőoktatásról van szó. Jelen esetben a meghatározás alapjául a felsőoktatási felvételi eljárásról *423/2012. (XII. 29.) Korm. rendeletet* vettük, amely a következő tárgyakat sorolja a természettudományos tantárgyak csoportjába: *biológia, fizika, kémia, földrajz (a földünk és környezetünk), természetismeret és a természettudomány*. Ez a beosztás szinkronban van az úgynevezett STEM[2] területek természettudományokat érintő részével, amiről a későbbiekben még szót ejtünk.

A Nemzeti Alaptantervben a természettudományos kompetencia a technikai kompetenciával együtt alkotja az egyiket, a kilenc kulcskompetencia közül. Birtokában az egyén képes felismerni az őt körülvevő természeti jelenségek, és mechanizmusok működését és kimenetelét, ismeri a fenntartható társadalom folyamatait és feltételeit, valamint képes a tudását és ismereteit a hétköznapi életbe integrálni, azok által új dolgokat alkotni, létrehozni. Kritikus attitűddel áll a természettudományokkal kapcsolatba hozható álhírekkel szemben, miközben nagy nyitottságról tesz tanúbizonyságot az őt körülvevő műszaki- és tudományos világ megismerése során.

[2] Magyarul MTMI (matematikai, természettudományos, műszaki és informatikai) tudományok.

A műveltségi területeket vizsgálva a természettudományos tantárgyak „Az ember és természet", valamint a „Földünk – környezetünk" területhez tartoznak. Előbbiben a közműveltségi tartalmak lefedik a technika, kémia, biológia, fizika tantárgy ismereteit, míg az utóbbi műveltségi terület egésze a földrajz tantárgyat foglalja magába, átfedésben a természet-és környezetismeret tárgyakkal.[1]

A kerettanterveket megvizsgálva már sokkal átfogóbb képet kaphatunk arról, hogy melyik tantárgynak mennyi a kötelező minimális óraszáma. Ezekhez az óraszámokhoz viszonyítva minden tantárgy ismeretanyaga túlmutat az adott óraszámban teljesíthető követelményeken. Ez a megállapítás az informatika tantárgyra is átültethető. Az alábbi táblázatokban a kötelező minimális óraszámokat láthatjuk.

| Tantárgyak | 1. évf. | 2. évf. | 3. évf. | 4. évf. |
|---|---|---|---|---|
| Magyar nyelv és irodalom | 7 | 7 | 6 | 6 |
| Idegen nyelvek | | | | 2 |
| Matematika | 4 | 4 | 4 | 4 |
| Erkölcstan | 1 | 1 | 1 | 1 |
| **Környezetismeret** | **1** | **1** | **1** | **1** |
| Ének-zene | 2 | 2 | 2 | 2 |
| Vizuális kultúra | 2 | 2 | 2 | 2 |
| Technika, életvitel és gyakorlat | 1 | 1 | 1 | 1 |
| Testnevelés és sport | 5 | 5 | 5 | 5 |
| Szabadon tervezhető órakeret | 2 | 2 | 3 | 3 |

**1. táblázat**: Kötelező tantárgyak és minimális óraszámok az 1-4. évfolyamon.[2]

| Tantárgyak | 5. évf. | 6. évf. | 7. évf. | 8. évf. |
|---|---|---|---|---|
| Magyar nyelv és irodalom | 4 | 4 | 3 | 4 |
| Idegen nyelvek | 3 | 3 | 3 | 3 |
| Matematika | 4 | 3 | 3 | 3 |
| Történelem, társadalmi és állampolgári ismeretek | 2 | 2 | 2 | 2 |
| Erkölcstan | 1 | 1 | 1 | 1 |
| **Természetismeret** | **2** | **2** | | |
| **Biológia-egészségtan** | | | **2** | **1** |
| **Fizika** | | | **2** | **1** |
| **Kémia** | | | **1** | **2** |
| **Földrajz** | | | **1** | **2** |
| Ének-zene | 1 | 1 | 1 | 1 |
| Vizuális kultúra | 1 | 1 | 1 | 1 |
| Dráma és tánc/Hon- és népismeret* | 1 | | | |
| Informatika | | 1 | 1 | 1 |
| Technika, életvitel és gyakorlat | 1 | 1 | 1 | |
| Testnevelés és sport | 5 | 5 | 5 | 5 |
| Osztályfőnöki | 1 | 1 | 1 | 1 |
| Szabadon tervezhető órakeret | 2 | 3 | 3 | 3 |

**2. táblázat:** Kötelező tantárgyak és minimális óraszámok az 5-8. évfolyamon.[2]

| Tantárgyak | 9. évf. | 10. évf. | 11. évf. | 12. évf. |
|---|---|---|---|---|
| Magyar nyelv és irodalom | 4 | 4 | 4 | 4 |
| I. idegen nyelv | 3 | 3 | 3 | 3 |
| II. idegen nyelv | 3 | 3 | 3 | 3 |
| Matematika | 3 | 3 | 3 | 3 |
| Történelem, társadalmi és állampolgári | | | | |
| ismeretek | 2 | 2 | 3 | 3 |
| Etika | | | 1 | |
| **Biológia - egészségtan** | | **2** | **2** | **2** |
| **Fizika** | **2** | **2** | **2** | |
| **Kémia** | **2** | **2** | | |
| **Földrajz** | **2** | **2** | | |
| Ének-zene | 1 | 1 | | |
| Vizuális kultúra | 1 | 1 | | |
| Dráma és tánc/Mozgóképkultúra és médiaismeret * | 1 | | | |
| Művészetek * | | | 2 | 2 |
| Informatika | 1 | 1 | | |
| Technika, életvitel és gyakorlat | | | | 1 |
| Testnevelés és sport | 5 | 5 | 5 | 5 |
| Osztályfőnöki | 1 | 1 | 1 | 1 |
| Szabadon tervezhető órakeret | 4 | 4 | 6 | 8 |

**3. táblázat:** Kötelező tantárgyak és minimális óraszámok a 9-12. évfolyamon.[2]

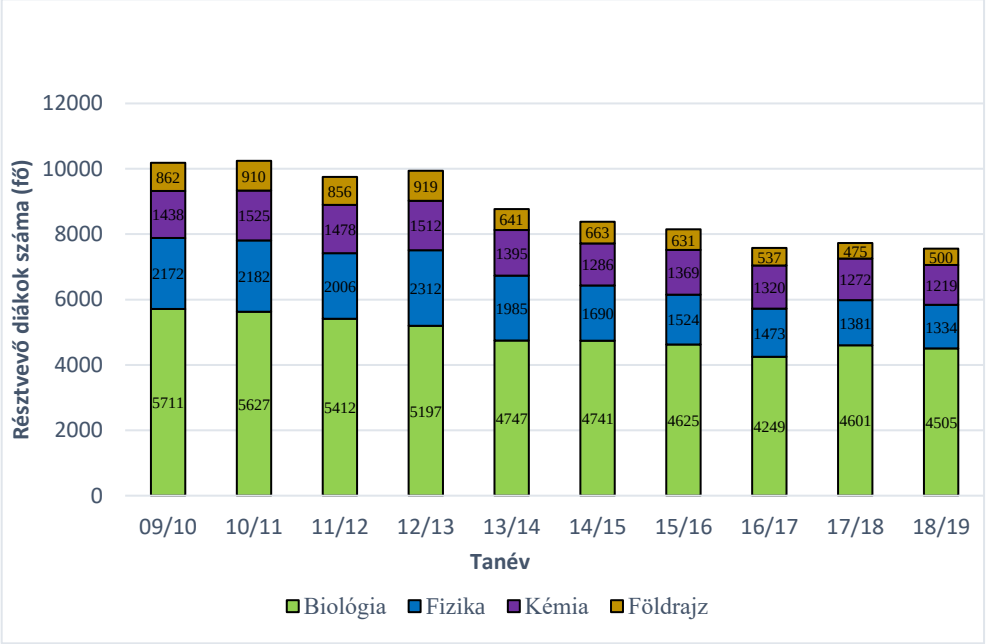
Magyarországon a természettudományos tantárgyak, a teljes éves összóraszámhoz viszonyítva a 24%-át teszik ki a tanóráknak. Ha kevesebb óraszám tartozik egy adott tárgyhoz, mint amivel le lehetne fedni annak ismeretanyagát, akkor felléphetnek bizonyos problémák. Egyik legfőbb ilyen nehézség a kísérlet vagy megfelelő mértékű szemléltetés, illetve vizsgálódások hiánya a tanóra menetében, ami jelentősen hathat a diákok motivációjára.

A TIMSS-2015-ös méréséből kiderül, hazánkban jóval a nemzetközi átlag alatt van az egy tanévben a természettudományok tanításra fordított órák száma (200-328 óra)[3], valamint a tanórai kísérletek mennyisége a 8. osztályban. Továbbá, a magyar pedagógusok általában kevesebb szerepet szánnak az oktatás során olyan módszertani megoldásoknak, mint például a felfedeztető tanulási technika, a kísérletek, valamint a modellek, vagy akár a szimulációk. Ennek megoldásaként, nem csak az óraszámokat kellene növelni, hanem a megfelelő infrastrukturális-, és humánerőforrást biztosítani. A jelentés alapján hazánk a fentebb említett két erőforrás kapcsán is átlagon aluli értékeket produkál, valamint a diákok sem tartják különösebben fontosnak a természettudományokat.[3]

## 1.2. Természettudományos versenyeken történő részvételi adatok alakulása

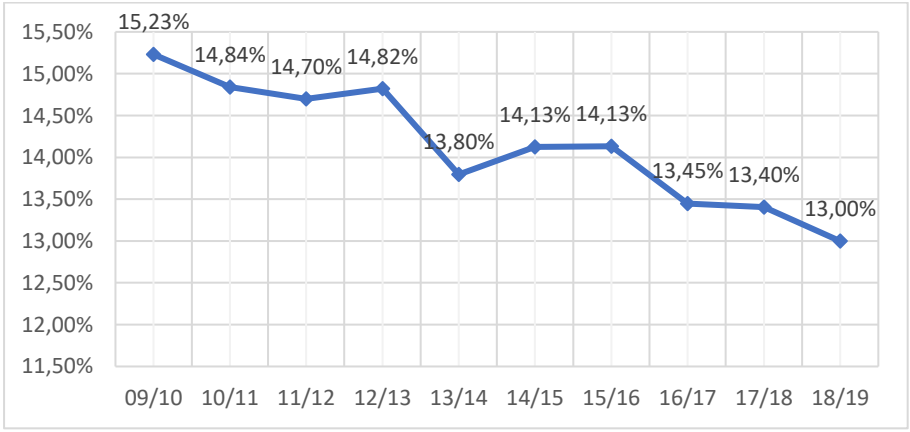
Elsőként a természettudományos tanulmányi versenyeken való részvételt vizsgáltam (1. ábra). A statisztika szempontjából fontos, hogy olyan versenyre volt szükség a méréshez, amely tárgyanként viszonylag egyforma típusú feladatokra épül fel. Ezért és az adatok hozzáférhetősége miatt választottam az OKTV versenyek eredményeit. A vizsgált időintervallum 10 évet ölel fel, kezdve a 2009/10-es tanévvel, a 2018/19-es tanévvel bezárólag.[4]

[3] Az olyan országokat vizsgálva, ahol tantárgyanként tanítják a különböző területeket.




**1. ábra:** OKTV-n résztvevő diákok számának alakulása, a természettudományos tantárgyak tükrében

Mivel ezeken a versenyeken nem kötelező a tanulók részvétele, ezért a jelentkezők száma szoros összefüggésben áll a tantárgyhoz kapcsolódó motivációval. A jelentkezők száma a 2009-es évhez képest 74%-ra csökkent a legutóbbi mérés időpontjában. Ez a csökkenés folyamatosnak mondható a 2010-es évtől kezdődően, kivételt ez alól csak a 2012/13-as tanév képez csak (2. ábra). Ez a tendencia azonban nem csak a természettudományos tantárgyakat érintette, hanem az össze tantárgynál megfigyelhető, ugyanis a természettudományos tantárgyak OKTV versenyén résztvevő diákok részesedése az egész mezőnyből, csupán 1%-ot csökkent. Meg kell azonban azt is vizsgálnunk, hogy valóban arányaiban kevesebb diák jelentkezik-e ilyesfajta versenyekre, vagy csak a populáció csökkenése jelenik meg az adatokon. Megállapítható, hogy a csökkenés arányaiban véve is jelen van, igaz a mértéke így csak körülbelül 2,3%-os a megfigyelt időszakban, azonban a tendencia itt is csökkenő irányú.




**2. ábra:** Az OKTV-re jelentkezett hallgatók száma a középiskolai tanulók számához viszonyítva

A tantárgyak megoszlásainak arányai nem mutatnak 3-4%-os értéknél nagyobb kilengést. Megállapítható, hogy arányaiban a fizika tantárgy iránti érdeklődés zuhant a legtöbbet, mégpedig 3,68%-ot. Földrajz esetében ez az érték ugyan csak 1,85%, de a legutóbbi mérés során már csak 500 tanuló vett részt a versenyen a 7558 indulóból, ami így mindössze 6,62%-ot jelent. Talán ennél a két tárgynál nagyban befolyásoló tényező lehetett az előrehozott érettségik eltörlése is, az adott tárgyakból.

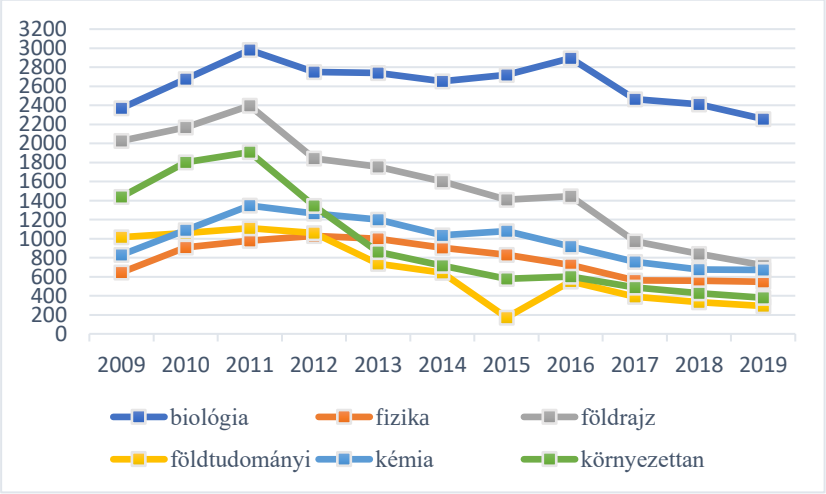
A kémia tantárgy indulószáma kisebb nagyobb kilengésekkel, de csökkent, azonban arányaiban 2%-ot növekedett az összes résztvevőt nézve. A vizsgált időtartam alapján a biológia az a tárgy, amelyen a jelentkezők több mint fele elindult és 3,53%-os erősödést mutatott a teljes indulólétszámhoz mérve.

Összességében tehát megállapítható, hogy egyre kevesebb diák vesz részt ezeken a versenyeken és a részvételi létszámok alakulása a kémia és a biológia esetében ezzel lehetnek összefüggésben. Emellett az is látszik, hogy a fizika és földrajz népszerűségének erős visszaesése az érettségi rendszer változásához köthető. Gondolunk itt arra, hogy a diákok nem tudták magukat tehermentesíteni 1-1 tantárgy alól az előrehozott érettségi kapcsán, ami miatt kevesebb idejük volt foglalkozni különkülön a tantárgyakkal.

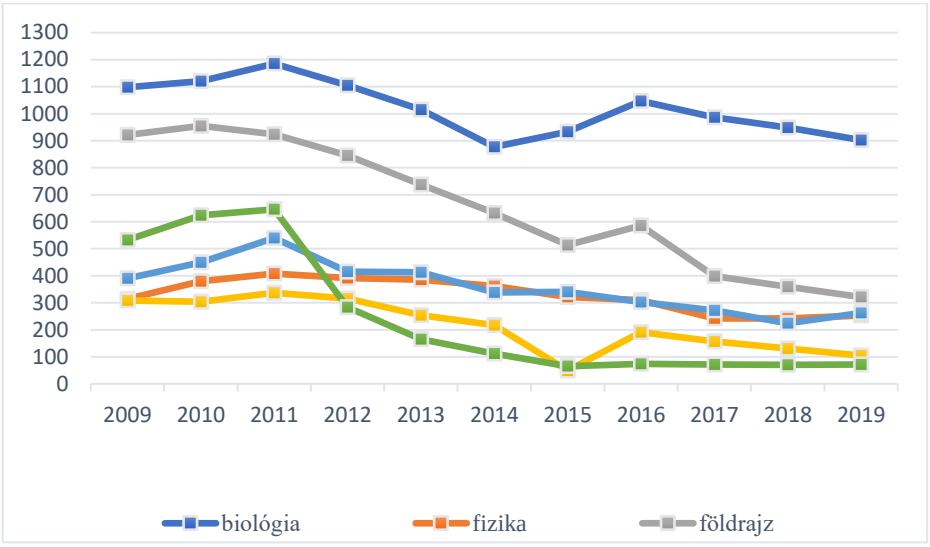
## 1.3. Felvételi statisztikák

A felvételi statisztikák készítésénél figyelembe kell vennünk, hogy több részre tagolódik a földrajz ismeretköre, ezért nem csak a fentebb említett négy tárgy alapszakját érdemes vizsgálni. 2013-tól pedig bevezetésre kerültek az osztatlan tanárszakok, amelyeket a szakpárok miatt szintén külön kell megvizsgálni. Az adatok a *felvi.hu* oldalon csak a matematikával közösen szerepelnek, de a jogszabályi értelmezés szerint az említett tárgy nélkül a következő szakokat vizsgáltam meg: biológia, fizika, földrajz, földtudományi, kémia, környezettan.[5] Ezek a természettudományos alapszakokat biztosító tantárgyak. Az osztatlan képzés adatainál pedig a szakpár első tagjához soroltuk a diákokat, tehát aki biológia-földrajz szakpárral rendelkezik, az a biológia tárgyhoz lett sorolva. A vizsgálatot szintén a 2009-es felvételi eljárástól kezdve végeztük és csak az általános eljárások szerepelnek az adatok között.

Előzetesen megállapítható, hogy mind a jelentkezett (3. ábra) és mind a felvett hallgatók (4. ábra) szempontjából a csökkenő tendencia érvényesül. A legnépszerűbb tantárgy ebben az esetben is a biológia, azonban az azt követő sorrendben már jelentős eltérések vannak, és a földrajz lesz a második helyen.

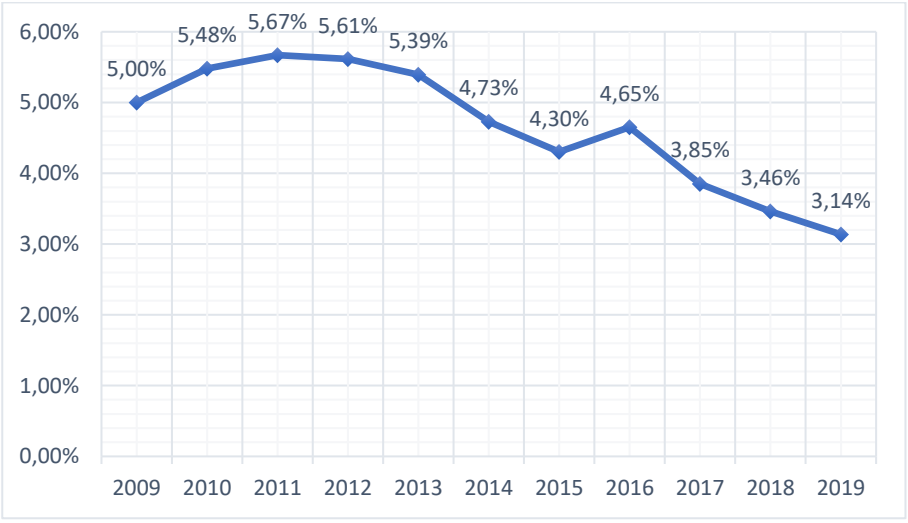



**3. ábra:** Természettudományos szakokra jelentkezett hallgatók számainak alakulása, tantárgyanként



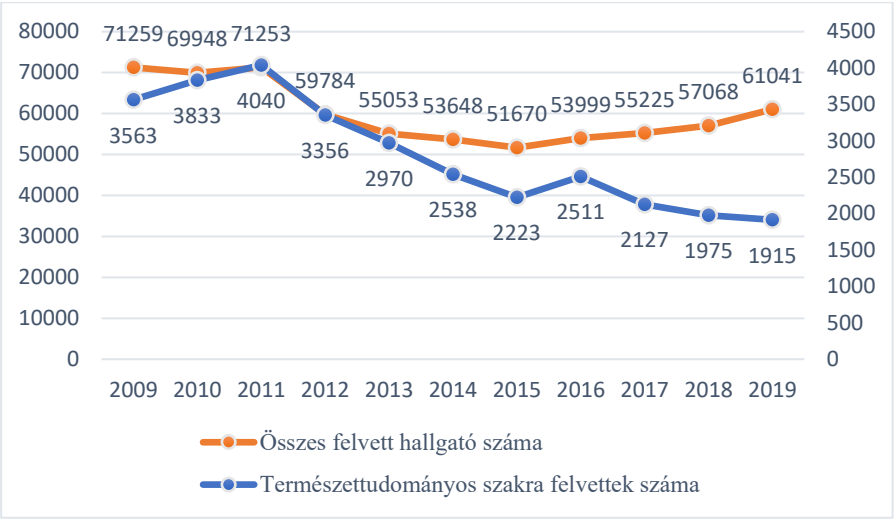

**4. ábra:** Természettudományos szakokra felvett hallgatók számának alakulása, tantárgyanként

A csökkenés mértéke önmagában nem elég bizonyíték arra, hogy kevesebben jelentkeztek a természettudományos szakokra. Ahhoz, hogy a teljes képet megkapjuk az arányokat kell megvizsgálni.




**5. ábra:** Természettudományos szakokra felvett hallgatók aránya, a felvett diákokhoz viszonyítva

Itt már jól látható, hogy tényleges csökkenésről beszélhetünk. A vizsgált időszakban a természettudományos szakokra felvett hallgatók száma az összes alapszakra/osztatlan mesterszakra felvett hallgatóknak az immáron kevés 5%-ról csupán 3,14% százalékra csökkent (5. ábra). Ez a tendencia azonban nem feltétlenül jellemző a felvett hallgatók számának alakulására (6. ábra). Az utóbbi években növekedés figyelhető meg, ezáltal az olló úgymond egyre jobban ketté nyílik, valamint kimondható, hogy a tendencia nem a népesség csökkenésének tudható be. Fontos azt is leszögezni, hogy az alacsony számokban benne vannak a tanárszakokra jelentkezett hallgatók is, így a rendkívüli munkaerőhiány, amely jelenleg már érezteti hatását, a pedagógus szakmában is rövidesen égető problémává válik a természettudományi tantárgyak kapcsán.




**6. ábra:** A felvételt nyert hallgatók tendenciája

## 2. A STEM és a robotika kapcsolata

A cikk ezen fejezetében röviden áttekintjük a STEM mivoltát, történetét és jelenlegi helyzetét az oktatásban, valamint megvizsgáljuk a robotikával való kapcsolatát és a robotika alkalmazásainak előnyeit hátrányait.

A STEM egységesíti a minket körülvevő folyamatok leírására szolgáló tudományokat. Ezek megismerése kulcsfontosságú a jelen kor szempontjából, mivel a gazdasági és munkapiaci átrétegződés jelenleg is folyamatban van, amely által munkakörök szűnnek meg és teljesen más területek születnek meg. Az adatok alapján arra lehet következtetni, hogy ezen tárgyak oktatása nem megfelelő, amiből diákok motivációjának csökkenése következhet. Olyan oktatási módszerekre van szükség, amely nem a konkrétan a probléma megoldását adja meg, hanem az oda vezető algoritmust tanítja meg úgy, hogy azt máshol is alkalmazni tudják majd a diákok, mindezt játékosan, érdekesen, új dolgot létrehozva, alkotva. Ehhez tökéletes lehet a robotika integrálása, amellyel az absztrakt problémákból kézzel fogható szimulációkat hozhatunk létre.

### 2.1. Mi az a STEM? Melyik részéről lesz szó?

Mint ahogy korábban említettük a STEM egy rövidítés, amely magában foglalja a természet-, a műszaki-, a matematikai-, és informatikai tudományok oktatását és az oktatásba való integrálásukat. Az USA kiemelt és vezető szerepet játszik a STEM tárgyak oktatásában és ezen tudományterületeket magas prioritással kezeli, valamint a mozaikszó megszületése és múltja is oda köthető. Mindennek az elindítója a hidegháborús űrverseny volt, konkrétan a Szovjetunió első műholdjának felbocsájtása. Ez a mozzanat kellett, ahhoz, hogy az amerikai oktatási politikában gyökeres változás álljon be és előtérbe kerüljenek ezen tudományterületek. 2001-től a Nemzeti Tudományos Alap az ezekkel a területekkel foglalkozó tudományoknak a STEM nevet adta.[6]. A 2010-et megelőző évektől kezdve az USA-ban több átfogó intézkedés is indult a tudományterületek fellendítésére és kiemelt helyen való kezelésük érdekében. A különböző törekvések és a hozzáállás azonban sok másik országban ugyanúgy fellelhető (jellemzően 2010-es évek után), valamint az Európai Unión belül is a European

Round Table of Industrialists[4] is kulcsfontosságúnak tartja a STEM tárgyak oktatását a gazdasági fejlődés érdekében.[7]

A STEM-en keresztül a tudásszerzés mellett a diákoknak több olyan fontos kompetenciája is fejlődik, amelyek a mindennapi életben elengedhetetlen készségekké váltak mára már. Ilyenek például a hatékony problémamegoldás és helyzetfelismerés, a kritikus gondolkodás, a kommunikáció és a csapatban való együttműködés képessége. Ezek gyakorlatilag nevezhetők kulcskompetenciáknak is, ha azt vizsgáljuk meg, hogy a jelenlegi versenyszférában az adott terület lexikális tudásán kívül mivel is kell rendelkeznie egy embernek.

A cikkben főként a természettudományos területeket, tantárgyakat vizsgáljuk meg, ami már kiderülhetett a különböző adatsorokból is. A későbbiek folyamán kitérünk a STEM területeire is, de a gyakorlatban az összes terület egyidejűleg jelenik meg, hiszen például a programkód létrehozásához matematikai ismeretek is szükségesek, tehát a kódolás folyamán ez a készség is fog fejlődni. A legfőbb ok, hogy a természettudományos tantárgyak kerültek a fókuszba, az az, hogy Magyarországon ezeknek a tárgyaknak a kifutási számai egyre alacsonyabbak és arányaiban is rendkívüli mélységekben vannak, miközben egyre több területen lenne szükség a munkaerőre a versenyszférában, a tanári pályáról nem is beszélve.

## 2.2. Robotika általi oktatás előnyei, hátrányai

A robotika oktatásban való használatának az egyik leginkább kézzelfogható előnye a diákok figyelmének lekötése, valamint a motiváció felkeltése és fenntartása. Robotika segítségével a gyerekek az adott tantárgy témakörét játékos formában, szórakoztatóan tudják elsajátítani. Több tanulmány is készült, amelyben a diákok figyelmét vizsgálták a foglalkozások során, amelynek központjában a robotika volt. Az egyik ilyen tanulmányt szeretném bemutatni a teljesség igénye nélkül.[8]

Az előbbiekben említett kutatás Amerikában készült és több fázisra osztva vizsgálta a diákokat. Ezek a fázisok a következőképpen alakultak: robotépítés, programozás, önálló feladatmegoldás (az első kettő fázisban pármunka volt). A robotépítés fázisban a diákok közül mindösszesen egy olyan tanuló volt, aki ránézett egyszer a mobiltelefonjára, ami véleményem szerint nem feltétlen kezelendő problémának. A programozás fázisban már több olyan diák volt, akinek a figyelme más irányába kalandozott, azonban még itt is elenyésző számok voltak egy normál tanórához képest. Ezek az eredmények megegyeznek a saját tapasztalataimmal. Amikor foglalkozást tartottam a legérdekesebb rész a robotok összeszerelése és a feladat megoldásának megtervezése volt, majd a kódolásnál kicsit visszaesett a figyelem szintje, amely a kipróbálás szakaszában megint csak teljesnek mondható volt.

Visszatérve a kutatáshoz, az egész folyamat során 1 olyan diák volt, aki feladta az egészet és nem tudta végrehajtani az önálló feladatot. A kutatás a figyelem mellett érzelmi állapotot is vizsgált. A visszajelzések alapján az első érzelmi reakciók az idegesség, frusztráció, valamint nyugtalanság volt. Ennek oka az volt, hogy sok diák nem is találkozott korábban robotokkal, valamint az önbizalom hiányának is felróható, mivel a diákok többsége nem hitte el magáról, hogy igenis képes robot programozásra. Ezekben az esetekben az első benyomások fokozatosan átalakultak egy vidám élménnyé. Utólagosan többek már nem is találták nehéznek a feladatokat.

Velük ellentétben, az 5. évfolyamon, a diákjaim reakciója a robotika említése terén pont az ellenkező volt. Talán ennek az ellentmondásnak a magyarázata az életkori sajátosságokban rejlik és ők még inkább játéknak fogják fel a robotprogramozást. Náluk izgatottság, kíváncsiság és öröm társult a bejelentés mellé, amikor megtudták, hogy a tanév folyamán robotprogramozást fognak tanulni. Az 5. osztályos diákok, akikkel együtt dolgozunk, ugyan még csak Scratch nyelven készítenek egyszerű animációkat, de már itt is megjelent az a visszajelzés, hogy mennyire élvezetes olyan dolgot csinálni a

[4] Gyáriparosok Európai Kerekasztala

számítógépen, amelynek azonnali visszajelzése van és örömmel tölti el őket az alkotás folyamata, mint ahogy a kutatásban résztvevő diákok is ezt nyilatkozták.

További pozitív élmény volt még a párban való közös alkotás, valamint többen beszámoltak arról is, hogy az adott témakörben mélyebb ismereteket sajátítottak el úgy, hogy könnyebbnek ítélték meg az anyag feldolgozását azáltal, hogy problémaalapú oktatás valósult meg.

Mindenféleképpen ejtenünk kell pár szót a problémaalapú/projektalapú oktatásról is. Véleményem szerint a robotika hagyományos frontális oktatás keretei közt nem tud kiteljesedni. Szükség van egy olyan környezetre, ahol párban vagy csoportokban dolgozva saját maguk felfedezik a jó megoldáshoz vezető utat, ami által hatékonyabb lesz a tanulási folyamat és az anyagrész elmélyítése. Eközben azonban tanárként ott kell lennünk és mindenben a legnagyobb támogatást nyújtani a diákok számára, de nem úgy, hogy konkrét megoldásokat adunk a kezükbe, hanem csak terelgetjük őket. Ez azért is fontos, mert a legnagyobb motiváció az, amikor a diák saját maga rájön valamire és sikerélménye lesz, ami által boldogság tölti el, ami később pozitív visszacsatolást alakít ki nála az adott tananyagrészhez kapcsolódóan.

A sok pozitív dolog mellett nem szabad megfeledkeznünk arról, hogy lehet negatív oldala is robotika túlzott mértékű használatának. Megeshet, hogy a diákok motivációja csökken a robotika nélküli tananyagrészek feldolgozása kapcsán, azonban fontos, hogy nem fogunk tudni minden témakört feldolgozni robotok segítségével, ezért a tananyag elosztásánál kiemelten kell figyelmet fordítani arra, hogy váltakoztassuk a témaköröket. Amellett, hogy biztosan nem minden tananyagegység felel meg annak a kritériumnak, hogy robotikával oktatható legyen, nem minden típusú eszköz alkalmas minden egyes olyan témakör oktatására, ahol egyáltalán alkalmazhatók ezek a szerkezetek. Mindegyik fajta robotnak megvannak azok a tulajdonságai, amelyek által behatárolható, hogy mely feladatokra alkalmas. Például egy micro:bit-tel sokkal jobban be lehet mutatni az áramköröket, mint egy LEGO robottal. Utóbbit azonban könnyebb kiterjeszteni és bővíteni.

Hátrányokhoz sorolható a robotok költségei is. Itt is nagy eltéréseket tapasztalhatunk az árak alakulásában. Vannak olcsóbb (4-6000 Ft) egyszerűbb eszközök és drága (~100.000 Ft) kiterjeszthető eszközök, ha már az előző példánál maradunk. Sajnos a tapasztalatom az, hogy a legtöbb iskola még az olcsóbb eszközöket sem tudja mindig megengedni magának. Erre talán megoldást jelenthetnek olyan kezdeményezések is mint például *a „micro:bit botorkálás[5]”*. Az ilyen programok keretei között az iskolák kölcsönözhetnek eszközöket, ahol az egyik feltétel, hogy egy bizonyos használati idő után továbbküldik másik iskola részére az eszközöket.

Nem szabad megfeledkezni még arról sem, hogy az ilyesfajta tananyagátadásnak a jelenlegi oktatási rendszerünkben magas követelménye az, hogy a különböző tantárgyakat oktató pedagógusok egymással szinkronban dolgozva adjanak ki olyan feladatokat, amelyek akár több órán átívelő projektekké futnak ki. Emellett a pedagógusoknak rendelkeznie kell megfelelő szintű programozási tudással. Itt az informatika tanárok tekintetében is beszélhetünk hiányosságokról. Kevés az olyan jól programozó tanár, aki a jobb megélhetés reményében nem az informatika szektor irányába mozdul el és emiatt akár el is hagyja a pályát. 2011-ben a tanárként végzettek elhelyezkedése 1–3 évvel a végzés után a mérnöktanárok (79 végzett hallgató) és a számítástechnikatanárok (197 végzett hallgató) voltak azok, akik a legnagyobb százalékban hagyták el a pályájukat.[9] Ez a helyzet azóta sem javult. Az új tanárok hiánya és az egyre inkább elöregedő tanári társadalom pedig nehezíti az új módszerek és új programozási nyelvek elterjedését. Több megkérdezett pedagógus azt nyilatkozta, hogy tisztában van vele, hogy elavult az a programozási nyelv, amit ismer, de már nem akar/képes

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

megtanulni újat. Ezért is van az, hogy sokan kételkedve fogadják a robotprogramozást, mivel nincs kellő magabiztosságuk vagy attitűdjük az új módszerek és eszközök ismeretének elsajátításához.

# 3. Az alkalmazhatóság áttekintése, konkrét tantervi elemekkel

Az alábbiakban röviden áttekintjük a robotika alkalmazásának lehetőségeit, amelyeket a gimnáziumi kerettantervek alapján rendszereztünk a korábban már említett természettudományos tárgyak körében.[10] Az áttekintést tantárgyak szerint került felosztásra, és mindegyik kapcsolódási pontnál egy rövid kifejtésre került sor a lehetséges alkalmazhatósági lehetőségekkel kapcsolatosan. Fontos kiemelni, hogy az adott tantárgyak elemei általános iskolában is megjelennek természet- és környezetismeret órák keretei közt. Az átfedések értelmében a robotok alkalmazhatósága itt is véghez vihető, csak a feladatokon kell alakítani az életkori sajátosságoknak megfelelően.

Fontos megjegyezni, hogy most kifejezetten a lehetőségek kerültek fókuszba, későbbi munkáink során ezen területek részletesen kifejtésre kerülnek. A cél az, hogy az oktatásban szerzett tapasztalatok alapján meghatározzuk, hogy az adott célcsoport, illetve korosztály számára melyek a leginkább megfelelő robotok és kiegészítők, valamint egy olyan pedagógia programot alkossunk meg, amelyet követve a diákok élményszerűen sajátíthatják el a programozás alapjait, felhasználva a természettudományos területekkel való kapcsolódási lehetőségeket.

## 3.1. Földrajz

| Tematikai egység | Alkalmazhatóság | Rövid ismertető |
|---|---|---|
| **A Föld kozmikus környezete** | *Tellúrium készítése* | Föld, Hold, Nap hármas modellezése LEGO robotokkal. |
| | *Kepler 3. törvényeinek modellezése* | Akár az előző is továbbfejleszthető, vagy kiterjeszthető egész Naprendszerre. |
| | *Holdjáró készítése* | Komplex projekt, egy összetett holdjáró elkészítése, amelyen robotkar és különböző szenzorok találhatók. Alkalmazható fizika órán is. Megvalósításhoz könnyen bővíthető robot ajánlott. |
| **A földi tér ábrázolása** | *Iránytű/GPS készítés* | Egyszerűbb robotokkal iránytű készítése. Kapcsolódhat mellé terepi munka vagy tájékozódási verseny, esetleg kincskeresés. |
| **A Föld, mint kőzetburok szerkezete és folyamatai** | *Talajnedvesség és tápanyag mérése* | Különböző talajtípusok tápanyagtartalmának megmérése, illetve nedvességtartalom mérése különböző talajmodellek esetében. |
| **A légkör földrajza-Időjárás** | *Komplex mérőállomás készítése* | Egy komplex mérőállomás megépítése és az általa gyűjtött adatok alapján elemzéssel egybekötött projektmunka. Itt a választandó robot függ attól, hogy milyen méréseket akarunk végezni. Szennyezettségmérő esetén az Arduino eszközök ajánlottak. |
| **Megújuló energiaforrások** | *Megújuló energiával működő modellek készítése.* | Nap és szél által hajtott miniatűr erőművek építése. Izzók segítségével szimulálható már egy kisebb település áramellátottsága. Gazdaság- és energiaföldrajznál is megvalósítható. |
| **A világgazdaság jellemző folyamatai** | *Gyártósorok modellezése* | Projektként kiadható különböző gyártósorok modellezése vagy egy energiahordozó életútjának automatizálása. Kapcsolódhat a környezetvédelemhez is. |

**4. táblázat:** Alkalmazhatóság a földrajz tantárgyban

## 3.2. Fizika

A fizika tantárgy kivételes helyzetben van a robotok alkalmazhatóságának tekintetében. Bármely komplexebb robot megfelelő lehet az áramkörök működtetésének szemléltetésére. A robotika segítségével a diákok egy érdekes, és játékos környezetben végezhetnek méréseket, hozhatnak létre áramköröket, valamint köthetnek be áramköri elemeket. A digitális visszajelzésnek hála pedig tökéletesen szemléltethetők, különböző mért adatok is.

| Tematikai egység | Alkalmazhatóság | Rövid ismertető |
|---|---|---|
| **Tájékozódás égen-földön** | *Radar/GPS készítés* | Egyszerűbb robotokkal radar/GPS készítése. Kapcsolódhat mellé terepi munka vagy tájékozódási verseny, esetleg kincskeresés, akárcsak a földrajznál. |
| **A közlekedés kinematikai problémái** | *Gyorsulásmérés és sebességmérés* | Robotok beépített szenzoraival sebesség és gyorsulásmérés, mérések kiértékelése. |
| | *Fotocellás mérőrendszer készítése* | Komplex projekt, egy összetett mérőállomás készítése, amely analóg tárgyak mozgásához köthetően mér értékeket. |
| **A közlekedés dinamikai problémái** | *Mérőrendszerek készítése, erőhatások mérése* | Az erőhatások irányának, mértékének elemzése, értelmezése konkrét gyakorlati példákon: erőhatások mérése/ütköztetések/szabadesés. |
| **A tömegvonzás** | *Kepler 3. törvényeinek modellezése* | Bolygómodell megalkotása több bolygóval. Földrajzban is hasznosítható. |
| | *Nehézségi gyorsulás és erő mérése* | Szenzorokkal felszerelt robottal szabadeséshez kapcsolódó adatok mérése. |
| **Hidro- és aerodinamikai jelenségek, a repülés fizikája** | *Szélerőmű készítése* | Egy szélerőmű modellezése, pozitív negatív jelenségek demonstrálása ennek segítségével. A szél, mint potenciális energiaforrás megvizsgálása. |
| **Áramkörök, áramtermelés** | *Áramkörök létrehozása* | Bármely olyan megvalósítás jó ide, amelynek elkészítéséhez áramkört kell létrehoznunk. Egyik tökéletes példa a talaj nedvességtartalmának vizsgálata, mint elektromos vezető. |
| **A fény természete** | *Színfelismerés, színkeverés, infravörös sugárzás* | Infravörös szenzor alkalmazása, színfelismerő robotok építése, látható szín befolyásolásának demonstrálása, különböző megvilágítás hatására. |
| **Az űrkutatás hatása a mindennapjainkra** | *Holdjáró készítése* | Komplex projekt, egy összetett holdjáró elkészítése, amelyen robotkar és különböző szenzorok találhatók. Alkalmazható földrajz órán is. Megvalósításhoz könnyen bővíthető robot ajánlott. |
| | *GPS készítés* | Lásd. a táblázat első sorában! |

**5. táblázat:** Alkalmazhatóság a fizika tantárgyban

## 3.3. Biológia

| Tematikai egység | Alkalmazhatóság | Rövid ismertető |
|---|---|---|
| **Kapcsolatok az élő és élettelen között** | *Méh- és hangyarajok viselkedésének szimulálása* | Az apró „swarm" robotok segítségével, különböző parancsokat beprogramozva szimulálható egy raj viselkedése munka vagy külső támadás esetén. |
| **Jó a levegő? – A légzés** | *Levegőszennyezettséget mérő robot készítése* | Levegőszennyezettséget mérő eszközön dohányzás károsságának bemutatása, mérések |

| | | |
|---|---|---|
| | | végzése a városban terepi munka formájában. |
| **Szívből szívbe – nedvkeringés, belső környezet** | *Mesterséges érrendszer megépítése* | Egy érrendszer kialakítása és a „vér” áramlásának modellezése, érsérülések hatásának szemléltetése. |
| | *Pulzusmérő készítése, adatok gyűjtésével* | Robotra csatlakoztatható pulzusmérő szenzor alkalmazása, saját pulzus megmérése, grafikonkészítése belőle → nagyon egyszerű EKG készülék modellezése |
| **Védelmi vonalaink** | *Vérsejtek munkájának szimulálása* | Ugyancsak „swarm” robotokkal egy támadási szimuláció a különböző idegen behatolásokkal szemben. A védelmi rendszernek az idegen testhez való irányítása. |
| | *Immunrendszer modellezés* | |
| **Gazdálkodás és fenntarthatóság** | *Mérőrobot készítése* | Különböző talajtípusok tápanyagtartalmának megmérése, illetve nedvességtartalom mérése különböző talajmodellek esetében. |
| **Laboratóriumi környezet** | *Automatizált rendszerek és segédrobotok létrehozása* | A biológiában a mindennapi életben rengeteg robotot használnak (áramoltató berendezések, pumpák, mintavételi eszközök, centrifugák), ezek megvalósítása betekintést nyújthat egy biológus munkájába. Ez megvalósítható a kémia tantárgy keretei között is. |

**6. táblázat:** Alkalmazhatóság a biológia tantárgyban

## 3.4. Kémia

A kémia tantárgy szempontjából a robotika alkalmazásának nincs túl sok lehetősége, de ez nem is baj, hiszen a tárgy sajátossága, hogy rendkívül látványos és izgalmas kísérleteket lehet elvégezni és demonstrálni a diákok irányába.

| Tematikai egység | Alkalmazhatóság | Rövid ismertető |
|---|---|---|
| **Milyen részecskékből állnak az anyagok, és ezek hogyan kapcsolódnak?** | *Molekularácsok, részecskék, anyagok, vegyületek modellezése* | Molekulák kapcsolódásának szimulálása. Ehhez egy speciális robotra van szükség ('M-blocks'[11]), amely kockákból áll és szabadon mozoghatnak, vagy kapcsolódhatnak egymáshoz mágnes segítségével. |
| **Kémiai folyamatok a környezetünkben** | *Mérőrobot készítése* | Különböző talajtípusok tápanyagtartalmának, illetve nedvességtartalmának mérése. |
| **Laboratóriumi környezet** | *Automatizált rendszerek és segédrobotok létrehozása. Veszélyes kísérletek elvégzése robotkarok irányításával.* | A biológiában a mindennapi életben rengeteg robotot használnak (áramoltató berendezések, pumpák, mintavételi eszközök, centrifugák), ezek megvalósítása betekintést nyújthat egy biológus munkájába. Ez megvalósítható a biológia tantárgy keretei között is. |

**7. táblázat:** Alkalmazhatóság a kémia tantárgyban

# 4. A STEM többi területére gyakorolt hatásokról röviden

A STEM tárgyak közül részletesen megvizsgáltuk a természettudományos tárgyakat, azonban nem hagyhatjuk szó nélkül a robotika pozitív hatását a többi területére sem a STEM-nek. A robotok programozása nagyban elősegíti a problémaalapú, kritikus és algoritmikus gondolkodás fejlődését, valamint gyakorlatot biztosít a diákok számár a programozási alapok elsajátítására, vagy elmélyítésére, attól függően, hogy mennyire komplex robotról beszélhetünk. Kisgyermekek esetében a blokkos környezetben programozható robotok nagyban hozzájárulnak a programozás megszerettetéséhez, valamint a későbbiekben szükséges alapok bevéséséhez. A mai világunkban pedig egyre több

helyen elvárás az ilyesfajta gondolkodásmód és a különböző informatikai rendszerek ismerete. Ezeken felül a robotok számítógépre történő csatlakoztatásakor a diákok a fájlműveletek és a fájlrendszerek tekintetében is gyakorlatra tesznek szert, ezáltal ismereteik elmélyülését biztosíthatjuk ezen a területen. A tapasztalatok azt igazolják, hogy nagyon sok diák esetében itt hiányosságok mutatkoznak.

A matematika szempontjából a programozás az érdekesebb, függetlenül, hogy az robotprogramozás, vagy csak virtuális programok létrehozása. A matematika szinte minden kódban jelen van és ezáltal a diákok egy életszerű példát tapasztalhatnak meg abból, hogy mire is jó a matematika mélyebb ismerete. Már egy egyszerű feltétel megállapításánál logikai kijelentéseket, szabályokat alkotnak a diákok és gyakorlatilag észre sem veszik azt, hogy programozás közben matematikával is foglalkoznak. Integrálásra itt is számos lehetőség van, de úgy gondolom hatásosabb lenne olyan együttműködések létrehozása, ahol a matematika órán kiadott projektekre a megoldásokat az informatika órán keresik a diákok a programozás segítségével. Ha mindez robotikával történik, az külön kiemelendő példa, de az eszközök nélkül is, az interneten kutatva rengeteg érdekes példa és feladat található.

A műszaki részről sem megfeledkezve megállapítható, hogy a robotok összeszerelése, több robotrendszer összekapcsolása nagyban elősegíti a kreativitás és a finommotoros mozgások fejlesztését, valamint az alkotóvágy erősebbé válását. Rengeteg olyan eszköz van, amelyet a diákok kedvük szerint építhetnek vagy szerelhetnek. Minél komplexebb a robot, annál több műszaki ismeretre lesz szükség a kiterjesztéséhez. Ezáltal a diák belső motivációja is növekedni fog, hogy utánanézzen egyes műszaki dolgoknak, ami pedig felkelti az érdeklődését a mérnöki területek irányába és megszeretteti vele azokat. A műszaki rész szoros kapcsolatban van a fizika és matematika tantárgy integrálási lehetőségeivel és gyakorlatilag, ami hasznosítható fizikában az biztosan fejleszti a műszaki, mérnöki kompetenciáit is a gyermekeknek.

## 5. Összegzés

Hazánkban a trendek és számok alakulása negatív a természettudományos tárgyak megbecsültsége, valamint az ezeket a tárgyakat választó, felsőoktatásban továbbtanuló diákok számának tekintetében. Ennek egyik oka lehet a motiváció és érdeklődés hiánya az ilyen területek felé annak ellenére, hogy a modern világban egyre nagyobb igény mutatkozik ezen szakemberek foglalkoztatásában. Megállapítható, hogy a természettudományos tantárgyak körében a tantervnek megfelelően lenne lehetőség a robotika integrálására és egyes tananyagrészek oktatására, modellezésére és demonstrálására robotok segítségével. Ezek természetesen tantárgyanként eltérő súllyal és gyakorisággal alkalmazhatók.

A robotok használatának előnyei megkérdőjelezhetetlenek, azonban rengeteg akadály van még, amely meggátolja az integrálását. Ezek elhárítására a jelenlegi oktatási rendszer hibáinak kiküszöbölése jelenthetne megoldást. Fontos azonban azt is megjegyezni, hogy egy tantárgy teljes anyagának oktatására nem lehet csak és kizárólag a robotikára támaszkodni, hiszen rengeteg más látványos, motiváló, izgalmas és érdekes szemléltetési módszer létezik, akár digitálisan, akár analóg módon.

A robotika STEM tárgyakban való integrálásával kapcsolatban megfigyelhetők különböző átfedések. Ezek és a XXI. század tudományos szakemberek iránti kereslete felvetik a kérdést, hogy talán létjogosultsága lehetne egy olyan tantárgy bevezetésének, amely több diszciplínán átívelő témaköröket foglalna magában, mindezt úgy, hogy maximálisan kiaknázza a projektalapú modern oktatási elveket és formákat, miközben az informatika, azon belül pedig a robotika lehetőségeinek, magasfokú használatával köti egybe azokat. A modern eszközök használata nem a régiek teljes eltűnését jelenti, hiszen több olyan fontos tudáselem van, amelynek átadására a klasszikus módszerek a legjobbak. A cél az lenne, hogy megtaláljunk egy arany középutat és kialakítsunk egy olyan környezetet,

ami a diákokat a jelenlegi világhoz szükséges tudással ruházza fel és felkészíti őket annak a kihívásaira.

## Irodalom


1. Nemzeti Alaptanterv [2012]. http://www.okm.gov.hu/
2. EMMI rendelet a kerettantervek kiadásának és jóváhagyásának rendjéről 51/2012. (XII. 21.)
3. TIMSS 2015 [2016]: *Összefoglaló jelentés.* Oktatási Hivatal.
4. OKTV eredmények [2009/10-2018/19] https://www.oktatas.hu/kozneveles/tanulmanyi_versenyek_/oktv_kereteben/eredmenyek (utoljára megtekintve 2019. 10. 01.)
5. Felvételi statisztikák [2009-2019] https://www.felvi.hu/felveteli/ponthatarok_statisztikak/elmult_evek/!ElmultEvek/index.php/elmult_evek_statisztikai/ (utoljára megtekintve 2019. 10. 02.)
6. Heather B. Gonzalez, Jeffrey J. Kuenzi [2012]: *Science, Technology, Engineering, and Mathematics (STEM) Education: A Primer*. Congressional Research Service Reports 1-5.
7. European Round Table of Industrialists https://www.ert.eu/ (utoljára megtekintve 2019. 10. 08.)
8. ChanMin Kima, Dongho Kima, Jiangmei Yuana, Roger B. Hilla, Prashant Doshib, Chi N. Thai [2015]: *Robotics to promote elementary education pre-service teachers' STEM engagement, learning, and teaching.* Computers & Education 91. kötet. 14-31.
9. Veroszta Zsuzsanna [2012] *A tanári pályaelhagyás szaktárgyi mintázata.* Educatio, 2012/4. 607-611.
10. 51/2012. (XII. 21.) számú EMMI rendelet 3. melléklete, Kerettanterv a gimnáziumok 9-12. évfolyama számára.
11. John W. Romanishin, Kyle Gilpin, Sebastian Claici, and Daniela Rus [2015] *3D M-Blocks: Self-reconfiguring Robots Capable of Locomotion via Pivoting in Three Dimensions.* IEEE International Conference on Robotics and Automation (ICRA)

# Az informatikatanár lehetőségei az internetes zaklatás megelőzésében és kezelésében

Grünfelder Borbála[1], Holló Csaba[2]

[1]grunfibori@gmail.com
SZTE TTIK
[2]chollo@inf.u-szeged.hu
SZTE TTIK Informatikai Intézet

**Absztrakt.** A társadalmi változások az informatika gyors fejlődésével együtt az internetes zaklatások elterjedését is eredményezték, melyeknek nagyon súlyos következményei lehetnek. Az informatikatanárnak ennek megelőzésében kettős szerepe van, egyrészt az informatikai ismeretek oktatójaként, másrészt pedagógusként, hiszen hatása lehet az osztálybeli viszonyok alakulására, és segítséget nyújthat a résztvevőknek. Cikkünkben, a jelenség elemzése mellett, szeretnénk erre ötleteket és módszereket mutatni, azok alkalmazási feltételeivel és nehézségeivel együtt.

**Kulcsszavak:** nevelés, módszertan, zaklatás, biztonságos internethasználat

## 1. Bevezetés

Gyorsan változó világunkban mindig újabb és újabb eszközt kap az internetes zaklatás, hiszen folyamatosan újabb közösségi médiaplatformok jelennek meg, melyeket a digitális bennszülött generációk hamar használatba vesznek, míg a tanáraik vagy a szüleik nem feltétlen ismerik azok használatát és trükkjeit, sőt esetenként nem is tudnak a létezésükről. Nem csoda, hogy a gyermekeknek olyan gondolataik vannak, mint „az én anyám azt se tudja, hogy mit csinálok, mikor Snapchetezek, mégis hogyan érthetné meg a problémám?", vagy „a tanárnő azt se érti, hogy miről beszélünk a Facebookon, miért akar beleszólni?". Ily módon a diákok úgy vélik, hogy a digitális eszközökhöz és internethez kötődő kérdésekhez ők jobban értenek, ami természetesen nem minden esetben igaz. Viszont, azt valószínűsíthetik, hogy az informatikatanár technikai kérdésekben a többi tanárhoz képest jobban tájékozott, szükség esetén tőle informatikai szempontból hatékonyabb segítséget kaphatnak, könnyebb lesz neki elmagyarázni a problémájukat, és nem fog túlzott technikai korlátozásokat előírni, például nem fogja őket indokolatlanul eltiltani az internettől vagy a telefon használattól. Emiatt lehetséges, hogy internetes zaklatás esetén is a diákok hamarabb fognak az informatikatanártól segítséget kérni, mint más tanároktól, feltéve, hogy a személyes kapcsolat is megfelelő. Hogyha pedig mégis más pedagógustól kérnek segítséget, akkor is szükség lehet az informatikatanár tudására, ahogyan az iskola tőle várja a – zaklatásra is vonatkozó - szabályzatok informatikai részének a fiatalok digitális szokásait figyelembe vevő kidolgozását, folyamatos aktualizálását, oktatóanyagok elkészítését és tanítását is. Emellett természetesen rá is vonatkoznak a megelőzés érdekében minden pedagógusra vonatkozó olyan elvárások is, mint a támogató bizalmi környezet kialakítása. Összességében tehát, internetes zaklatással kapcsolatosan az informatikatanárnak kiemelkedő szerepe lehet, ehhez próbálunk a téma áttekintésével és gyakorlati ötletekkel hozzájárulni.

## 2. Mit érdemes tudni az internetes zaklatásról?

### 2.1. Az internetes zaklatás jellemzői

A zaklatás valamely személlyel szemben hosszabb időn keresztül, erőfölénnyel történő, emberi méltóságot sértő, fizikai és/vagy lelki, közvetlen vagy közvetett agresszió, melynek célja vagy hatása

megfélemlítő, ellenséges, megalázó, megszégyenítő vagy támadó környezet kialakítása [1]. A zaklatási folyamatban érintettekre a fenyegetettség, kiszolgáltatottság, tehetetlenség érzése jellemző. A zaklatásnak súlyos következményei lehetnek. Iskolai környezetben a zaklató célja szociális státusz, hatalom szerzése és annak megtartása, és hogyha azt tanulja meg, hogy az agresszív viselkedés sikeres, akkor később a konfliktusokat is agresszíven próbálja megoldani, ami deviáns, antiszociális viselkedést eredményezhet. Az áldozat el szeretné kerülni a megalázó helyzeteket, ezért gyakran nem megy iskolába, ami az iskolai teljesítményének romlását eredményezi. Többnyire kapcsolati problémái vannak vagy lesznek, magányos vagy azzá válik, bizalomhiány, szorongás, depresszió alakulhat ki. Pszichoszomatikus tünetek is megjelenhetnek, az áldozat agyában olyan strukturális változások jöhetnek létre, melyek megnövelik a mentális betegségek kialakulásának valószínűségét [25], illetve jellemző a CRP szint növekedése, mely sok év múlva is fennmarad, és későbbi szív- és érrendszeri, gyulladásos betegségek, depresszió, illetve cukorbaj kialakulásának kockázatát hordozza magában. [7]

Az internetes zaklatás internet segítségével történik, többnyire közösségi oldalakon és azonnali üzenetküldő szolgáltatásokkal, ugyanakkor társulhat hozzá más elektronikus (például telefonos), vagy hagyományos zaklatás is. Ebben az esetben az erőfölény abból fakad, hogy az áldozat nehezen tudja megvédeni magát, az eseményeket leállítani, és törölni az információkat. Látszólag a fizikai erő, vagy magas szociális státusz ez esetben nem jelent hatalmat, ugyanakkor az offline térben – akár ezek által - megszerzett státusznak lehet hatása a társak online térbeli viselkedésére is. Internet használata esetén a zaklatás számos speciális jellemzővel rendelkezhet [6, 18]:

- a személyes érintkezés és a metakommunikációs jelek hiánya miatt a zaklatónak könnyebb hazudnia, valótlanságokat elhitetnie másokkal, továbbá csökkenhet a morális kontrollja, szégyenérzete és bűntudata, szemében az áldozat „dehumanizálódhat", nem szembesül az áldozat szenvedésével, ezért nem is feltétlen tudatosul benne a cselekedetének súlya;
- a csökkent morális kontroll és a megnövekedett érzéketlenség a - fiatalokban amúgy is erős - csoportnorma követés esélyét megnöveli olyan helyzetekben is, amikor az egyén más körülmények között nem csatlakozna; ebből következik, hogy ha a zaklatónak jelentős számú támogatója van, akkor olyanok is csatlakoznak, akik egyébként nem tennék;
- a zaklató anonim módon is támadhat, ezért olyasvalaki is lehet zaklató, aki egyébként ezt nem vállalná fel;
- a zaklatás személyes találkozás nélkül is megvalósítható, továbbá a zaklató tartalmak olyankor is gyűlhetnek, amikor az áldozat nem is elérhető; ily módon nehezebb elmenekülni előle, ezért érzelmileg megterhelőbb [2, 3, 4];
- a zaklató tartalmak gyorsan és nagy számban terjedhetnek, a nagyszámú nyilvánosság miatt az áldozat még inkább megalázottnak érezheti magát;
- a beavatottakon kívülieknek (szülőknek, tanároknak is) sokkal nehezebb észrevenni, ezért amikor a környezet észleli, akkor már nagy lehet a mértéke.

A fentiek miatt az internetes zaklatás a hagyományoshoz képest súlyosabb következményekkel járhat, akár öngyilkossághoz is vezethet.

## 2.2. Kiből lesz áldozat?

A [9] szakirodalom szerint az áldozattá válást nem önmagában a külső megjelenési jegyek (kövér, szemüveges, tájszólás, szokatlan beszédstílus, szokatlan öltözék, eltérő rasszjegyek stb.) okozzák, hanem sokkal inkább személyiségjegyekre, tipikus reakciómintákra és (különösen a fiúk esetében) a fizikai erőre, illetve gyengeségre vezethetők vissza. A szorongóbb, érzékenyebb, csendesebb, visszahúzódóbb, alacsonyabb önbecsüléssel rendelkező, magányosabb gyermekek gyakrabban válnak áldozattá. Ugyanakkor, a [9] tanulmány felmérése szerint, míg a tanárok többsége a bántalmazás

okának az áldozat másságát, eltérő viselkedését gondolja, addig a diákok szerint a bántalmazás oka nem annak alanya, hanem a csoportnyomás, a megfelelési vágy. Ez is azt mutatja, hogy kiemelt szerepe van a megfelelő csoportnorma kialakításának. Továbbá, a [30] szerint a csoportegység megtartása érdekében a csoport megpróbálja a legkevesebb kapcsolati szálon kötődő tagját kivetni magából, azaz „megsemmisíteni". Ez korrelál a diákok véleményével, ahogyan azzal a korábbi tapasztalattal is, hogy általában a magányosabb gyermekek lesznek áldozatok.

## 2.3. A környezet szerepe

A gyerekek manapság folyamatosan online kapcsolatban vannak egymással, ugyanakkor nehéz létezni egy olyan térben, melyben folyamatosan figyelik egymást. Ráadásul, a kevesebb személyes kapcsolat miatt nincs elegendő tapasztalatuk arra, hogy hogyan kellene viselkedni egymással úgy, hogy ne legyenek bántók még akkor sem, ha a másikat nem kedvelik. Ilyen körülmények között az agresszió könnyen gerjed, és csökken a másik személyiségének, érzelmeinek és a tettek következményeinek súlya. Sokan, minőségi kapcsolatok hiányában, az elfogadásuk vágyát mennyiségi visszajelzésekkel (pl. lájkgyűjtésekkel) próbálják kielégíteni, és könnyebben követik a csoportnormát még akkor is, ha az esetleg nem egyezik az elveikkel. Agresszív közösségben a zaklató „menő" lehet, akihez mások is csatlakoznak, és akinek a tekintélye, a zaklatás nem megfelelő kezelése esetén, tovább nőhet. Más nézőpontból viszont többnyire a zaklató is segítséget igényelne ahhoz, hogy agressziójét kezelni tudja, aminek forrása az is lehet, hogy más körülmények között (régebben, vagy otthon) ő is agresszió elszenvedője (volt). Közvetlen agresszió hiányában is a gyermekek valamilyen szinten átveszik a felnőtt világra jellemző szorongást és agressziót, ennek jele, hogy a zaklatás már az óvodában is jelentkezik [22].

Nagyon fontos kiemelni, hogy nem egyszerű gyerekcsínyről van szó, sok esetben nem várható el a gyerektől, hogy egy ilyen helyzetet önállóan megoldjon. A külső szemlélőknek (szülők, tanárok, nézőközönség, média) felelőssége van abban, hogy észrevegyék és tegyenek ellene. Prof. dr. Kósa Éva szerint Magyarországon magas a gyerekek veszélyeztetettsége, mert elég jó az eszközellátottság, ugyanakkor nagyon rossz a szülők informáltsága [18, 29]. A felnőttek sok esetben nem ismerik a gyerekek által használt felületeket, nem tudják, hogy mire kellene figyelni, sőt egyes történéseknek a súlyát sem értik, másfelől a gyerekek sem számolnak be arról, ami az online közösségeikben történik. A kortárs szemlélők sok esetben félnek az áldozattá válástól, ezért nem védik meg az áldozatot, hanem felelősségük csökkentése érdekében magukat győzik meg arról, hogy a zaklatás „csak poén volt", vagy áldozathibáztatással élnek.

Az áldozathibáztatás egy többnyire nem tudatos önvédelmi stratégia, aminek az a lényege, hogy az áldozattal történtekkel kapcsolatos tehetetlenségünk vagy nem megfelelő cselekedetünk felelősségét az áldozatra hárítjuk, ily módon csökkentve az azzal kapcsolatos negatív érzéseinket. Minél inkább sikerül magunkkal elhitetni, hogy az áldozat (is) hibás, és megérdemelte azt, amit kapott, annál kevésbé érezzük magunkat hibásnak vagy sebezhetőnek. Másfelől, az áldozat sok esetben magát is hibásnak érzi, és emiatt nem kér segítséget, az áldozathibáztatás pedig ezt tovább erősíti. Összességében az áldozathibáztatásnak köszönhetően csökken annak az esélye, hogy az áldozat a környezetétől segítséget kapjon.

## 2.4. A segítségkérés problémái

Az áldozat annál nagyobb eséllyel fog segítséget kérni, minél jobban sikerül eloszlatnunk az aggodalmait, melyek tipikusan a következők [17, 18]:

- árulkodónak minősítik, ami miatt még jobban kiközösítik;
- éppen a nyilvánosságot szeretné felszámolni, nem szeretne több embert bevonni;
- a kényes tartalmat nem akarja másokkal is megosztani;
- mivel magát is hibásnak tartja, fél attól, hogy elítélik, megbüntetik;

- fél attól, hogy eltiltják az internettől, telefontól, ami elszigetelődést, a kiközösítésének fokozódását jelentené.

Ahhoz, hogy az áldozat segítséget kérjen, pontosan tudnia kell, hogy ilyen esetben mi fog történni, és nem kell félnie az áldozathibáztatástól vagy a kortársak reakciójától. Ehhez azonban meg kell arról győzni, hogy a környezete képes lesz megfelelően kezelni a helyzetet.

# 3. Szabályozási és pedagógiai lehetőségek

## 3.1. Közösségfejlesztés a megelőzés érdekében

### 3.1.1. Pozitív osztálylégkör kialakítása

A szakirodalom szerint a leghatékonyabb a megelőzés, melyben fontos szerepe van az intézményi környezetnek, az intézmény értékrendjének, az intézmény által közvetített elvárásoknak és ezek eljárásokban, szabályokban való direkt és indirekt megjelenésének. A [9] tanulmány szerint, a sértegetést és a kritizálást a diákok sokkal nagyobb arányban gondolják bántalmazásnak, mint a tanárok, ami azt jelenti, hogy ezek előkészítői vagy részei lehetnek egy zaklatásnak. A tanárnak észlelnie kell, ha a csoport megnyilvánulása többnyire – még ha ironikusan is – negatív, és jellemzőek a „savazások", azaz a sértő poénok és viccelődések. A diákoknak meg kell tanulniuk, hogy nem kell valakit bántani csak azért, mert nem szimpatikus, és egymás segítése közös érdekük. Fontos a toleráns, megértő és barátságos környezet kialakítása, melyben egymás segítése a norma, és amelybe egymás kisebb (de idővel összegyűlő) sértegetése sem fér bele. Ennek érdekében szükséges a kommunikációs, empátiás készségek, és impulzuskontroll fejlesztése is, melynek során rutint, megoldási módszereket, forgatókönyveket kell kialakítani.

### 3.1.2. A potenciális áldozat segítése

Jó lenne minél hamarabb észlelni a kialakuló negatív helyzeteket, hiszen könnyebb és kevésbé fájdalmas lehet leállítani, mint utólagosan kezelni. Viszont manapság a tanár az osztálybeli viszonyokat nehezebben látja át, hiszen a diákok a kommunikációjuk nagyobb részét az interneten folytatják, így akár az is megtörténhet, hogy az online zaklatók valamelyike és az áldozat egy padban ülnek, és a tanárnak barátoknak tűnnek. Ezért jó, ha van olyan diák, akitől a tanár értesülni tud a problémás helyzetekről, ilyen szempontból is hasznos lehet a kortárssegítők jelenléte, amire későbbi fejezetben visszatérünk.

De mit tehet a pedagógus akkor, ha észleli, hogy az osztályban valakit a többiek folyamatosan gúnyolnak, sértegetnek?

Először is, ne hagyja figyelmen kívül. Nem neki egy személyben, de a pedagógusok közösségének felelőssége van abban, hogyha hagyják a helyzetet tovább romlani. Érthető, hogyha egy túlterhelt pedagógus úgy érzi, hogy egyszerűen képtelen minden ilyen helyzettel foglalkozni, viszont nem is minden ilyen helyzetet neki kell egyedül megoldania. Az iskolában vannak más pedagógusok is, a lényeg, hogy ilyen helyzetekben mindig legyen valaki, aki megpróbál segíteni, és ehhez nagyon hasznos lehet, ha a tanárok az ilyen helyzetekről is kommunikálnak egymással.

Lehetőség szerint meg kell próbálni úgy segíteni, hogy ebből a sértett gyermek társai minél kevesebbet érzékeljenek, mert azért is rászállhatnak, ha úgy érzik, hogy vele kivételezünk.

De mit lehet tenni? A 2.2 fejezet alapján csökkenteni kellene a gyermek magányosságát és növelni kellene az önbecsülését.

Ehhez első körben valószínűleg érdemes vele elbeszélgetni. A tanár adhat neki tanácsot arra vonatkozóan, hogy hogyan javíthatja a kapcsolatait. Például, menjen oda egy olyan gyerekhez, aki nem véleményvezér, de nem is a leggyengébbek közül való, és próbáljon vele beszélgetni. Kérdezze meg,

hogy mit csinált a hétvégén, mivel játszott, milyen filmet nézett stb., és ezt rendszeresen tegye meg több gyerekkel is [30].

A beszélgetés során a tanár tájékozódhat a diák körülményeiről, illetve megpróbálhatja felmérni, hogy milyen területeken lehetne őt segíteni abban, hogy sikerélményei legyenek, és ezáltal is nőjön az önbizalma és megbecsülése a társai körében. Hogyha a tanár tantárgya esélyes ebben a tekintetben, akkor például segíthet neki személyre szabott feladatokkal és korrepetálással, más tárgyak esetében beszélhet kollégákkal. Természetesen minden gyermek esetén igyekezni kell a sikerélmények kialakításában, de ennek ebben az esetben kiemelt fontossága van.

A tanár az órán használhatja a mozaik módszert [11], melynek során építő egymásrautaltsági helyzetet teremt. A közös munka során a diákok kénytelenek egymást jobban megismerni és elfogadni, ily módon a feladattal a viszonyok változására is tudunk hatni. Ebben az esetben sokat nőhet a sértett gyermek megbecsülése a társai körében, ha rendszeresen neki köszönhetően érnek el jó eredményt, és tudatosítja a többiekben, hogy ez csak akkor működik, ha tisztelettel bánnak vele.

A tanár további tippeket is adhat az önbizalom növelésére. Az önbizalom növelése segít a gyereknek abban, hogy az esetleges kellemetlen megnyilvánulásokat kevésbé fenyegetőnek élje meg, és ne reagálja túl, amivel súlyosbítaná a helyzetet.

## 3.2. Szabályozás

### 3.2.1. Alapvető szabályok

Számos zaklatás esetére is érvényes jogszabály létezik. Ilyenek:

- a Btk. számos pontja: 222. § Zaklatás, 195. § Kényszerítés, 216. § Közösség tagjai elleni erőszak, 204. § Gyermekpornográfia, 226. § Rágalmazás, 226/A. + B. § Becsület csorbítására alkalmas hamis hang- / képfelvétel, 227. § Becsületsértés;
- a Ptk. néhány pontja: 2:45 § Jó hírnév megsértése, 2:48 § A képmáshoz és a hangelvételhez való jog, 2:51 és 2:52 § Személyiségi jogok megsértése;
- a Köznevelési Törvény, mely kimondja, hogy biztonságos környezetet kell kialakítani, amelyben a gyerekeknek módjuk van szabadon, lelkileg, szellemileg és fizikailag is harmonikusan fejlődni;
- a Nemzeti Alaptanterv, mely előírja a gyerekek erkölcsi nevelését, önismeret és társas kultúra fejlesztését, és a testi és lelki egészségre nevelést;
- a Gyermekvédelmi Törvény, mely szerint az iskolaköteles a gyermek veszélyeztetettségét jelezni a gyermekjóléti szolgálatnak, hatósági eljárást kezdeményezni a gyermek bántalmazása, súlyos elhanyagolása, más veszélyeztető ok, illetve a gyermek önmaga által előidézett súlyos veszélyeztető magatartása esetén;
- az Emberi Erőforrások Minisztériuma által kiadott Módszertani útmutató ([15]) a gyermekvédelmi észlelő és jelzőrendszer működésére.

Ezek azonban a zaklatásnak csak bizonyos eseteit fedik le, viszont az ezekbe nem tartozó esetek sem elfogadhatók, ezért az iskolának is feladata a Szervezeti és Működési Szabályzatában és Házirendjében szabályozni a zaklatás kérdését is.

Fontos, hogy ezeket a szabályokat a közösség minden tagja (diákok, tanárok, szülök) ismerje, ennek érdekében pedig az iskola megfelelően tájékoztassa őket.

Szabályszegések esetén fontos a körülmények alapos kivizsgálása, mely során akár az is kiderülhet, hogy az elkövető maga is áldozat. Továbbá, a cél nem a büntetés, hanem az áldozat fájdalmának enyhítése, az elkövető viselkedésének normalizálása, és a további zaklatások megelőzése, ezért a szakemberek első sorban a resztoratív sérelemkezelést javasolják.

### 3.2.2. A szabályzatok kialakítása

A cél nyilvánvalóan a kölcsönös tiszteleten és bizalmon alapuló iskolai környezet kialakítása, melyben azt, aki ez ellen vét, a társai is elítéljék. A szabályzatnak mindenki által könnyen érthetőnek kell lenni, és egyértelműen tartalmaznia kell azt, hogy az iskolai közösség számára mi az, ami elfogadhatatlan, annak mi a büntetése, és az egyes szereplőknek mi a feladata. Szabályozni kell azt is, hogy a diákok hogyan kérhetnek segítséget, és ilyen esetben az iskola dolgozóinak mit lehet és kell tenni. Annak érdekében, hogy a szabályzatot mindenki magáénak érezze, illetve a különböző résztvevők szempontjai is beépüljenek, érdemes a szabályzat készítésébe mindenkit (az iskola dolgozóit, szülőket, és a diákokat is) bevonni. A szabályzat kialakításához számos hasznos ötletet találunk a [17] projekt Tanári kézikönyvében.

## 3.3. Tájékoztatás, oktatás és érzékenyítés

A tanórákon érdemes a diákokat megismertetni az őket fenyegető veszélyekkel, az online zaklatással is. Hasznos lehet továbbá, ha tanórákon kívüli előadásokat, programokat is szervezünk a témában, vagy az iskola pedagógiai programjában már bent lévő projektnapokon, témaheteken a témához kapcsolódó veszélyekre is felhívjuk a figyelmet. Az is számíthat, ha nem csak az áldozat felé próbáljuk érzékenyíteni a diákokat, vagy azt hangsúlyozni, hogy ők is bármikor áldozattá válhatnak, hanem azt is kifejteni, hogy milyen következményei (büntetőjogi és lelki értelemben is) lesznek annak, ha ők válnak elkövetővé. Ez hatásosabb lehet, ha rendőröket vagy ügyvédeket kérünk meg, hogy tartsanak a témában előadást. Például a digitális témahét keretein belül is szervezhetünk az iskolánkba előadásokat, melyeken rendőrök beszélnek a digitális bűncselekményekről, és arról, hogy ezekért milyen büntetés jár. Lehetnek a héten kiscsoportos beszélgetések, ahol a diákok saját tapasztalataikat, véleményüket osztják meg. Alkalmazhatunk drámapedagógiás órákat, ahol a diákok szerepjátékok és szituációk segítségével jobban megérthetik az áldozatokat, az elkövetőket és a szemtanúkat.

A tanároknak fontos feladata az is, hogy a diákokkal ismertessék azokat a lehetőségeket, ahova segítségért fordulhatnak. Bizalommal fordulhatnak a tanárok felé is, az iskolán kívül pedig a családsegítő hivatalhoz és különböző alapítványokhoz (pl. Kék Vonal) is. Ezekről kitehetünk plakátokat is, melyen szerepelnek a telefonszámok, honlapok melyeken tájékozódhatnak és segítséget kérhetnek. Emellett persze azt is fontos tisztázni, hogy ezek hívása nem játék, tényleg csak akkor hívják, ha szükségük van rá.

## 3.4. Szülők oktatása

Annak ellenére, hogy a felmérések szerint a diákok egyre nagyobb része érzi magát szüleinél tájékozottabbnak az interneten, mégis a diákoknak körülbelül fele a szülőkhöz fordulna, ha online zaklatás áldozatává válna [17]. Gyakran előfordul, hogy a szülők tényleg nem értenek a számítógéphez, internethez olyan szinten, hogy a gyermekükkel az internetezés veszélyeiről beszélni tudjanak, felügyelni tudják a gyermekük számítógép használatát. A felmérések azt is kimutatták, hogy a szülők nem csak nem értik, hogy mit csinálnak a gyermekeik az interneten, de nem is feltételezik róluk, hogy internetes zaklatásba, gúnyolódásba kerülnének „támadó félként" [23], és nem is feltétlen érzékelik ennek a súlyát. Hasznos lehet, ha az iskola szervez a szülőknek egy képzést, melyben elsősorban azokra a veszélyekre hívják fel a figyelmet, mely a diákokat fenyegeti az interneten, így az internetes zaklatásra is. Budapest XII. kerületében a 2016-ban létrejövő kezdeményezésben a biztonságos netezésért, nem csak a gyermekeket és a tanárokat oktatták, hanem a szülőket is. Az volt a tapasztalatuk, hogy míg ez az önkormányzat szervezésében folyt egy közösségi házban, addig kevesebben vettek rajta részt, mint mikor átkerült az iskolákba, ahova a gyerekeik jártak [23]. Ebből is látszik, hogy a tanároknak és az iskoláknak mekkora szerepük lehet a tájékoztatásban. Fontos, hogy ne csak a veszélyekkel ismertessük meg a szülőket, hanem azzal is, hogy ezeket hogyan tudják megbeszélni a gyermekeikkel, mit tudnak tenni az ő biztonságuk érdekében, illetve, hogy az internetet és a digitális eszközöket hogyan lehet pozitív célokra is használni. Fontosnak gondoljuk, hogy a képzésben ne csak az osztályfőnök vegyen részt, hanem az informatikatanár is, akit a szülők esetleg informatikai

szempontból képzettebbnek, jobban hozzáértőnek tartanak, és ezért a segítséget is könnyebben fogadhatják el tőle.

## 3.5. Bizalom és rendelkezésre állás

Fontos, hogy a gyerekek, ha bármilyen bántalmazás éri őket, akkor azt ne tartsák magukban, legyen kivel megbeszélniük [27]. Mivel a napjuk jelentős részét az iskolában töltik, ehhez szükséges, hogy kialakuljon egy olyan bizalmi légkör, melyben legalább valamelyik tanáruknak el merik mondani a problémájukat, tudnak segítséget kérni. A gyerek akkor fog segítséget kérni, ha bízhat abban, hogy a tanár megérti és hatékonyan segíteni fog. A tanárok jelenlegi leterheltsége mellett nehéz, de fontos, hogy ha a diák beszélgetni szeretne, a tanár ne utasítsa el, mert valószínűleg nem fog újra próbálkozni. A tekintélytiszteleten alapuló rendszerben felnőtt tanár is képes kell, hogy legyen elbeszélgetni a diákokkal, és a tudomására jutott legdurvább dolgokra is megfelelően reagálni. Itt megjegyzendő, hogy ehhez szükséges a tanárok érzelmi támogatása is ahhoz, hogy a stresszt kezelni tudják.

## 3.6. A megtörtént eset kezelése

Ha már megtörtént a zaklatás, a cél akkor sem a zaklatók kriminalizálása, hanem a résztvevők visszaterelése a „normális" iskolai életbe oly módon, hogy lehetőleg elejét vegyük további zaklatásoknak. Ily módon az, hogy mit kell és lehet tenni online zaklatás után, egy meglehetősen komplex kérdés, aminek csak a legfontosabb, az informatikatanár számára is hasznos lépéseit fogjuk áttekinteni a [18] és más irodalmak alapján.

### 3.6.1. Beszélgetés az áldozattal és az elkövetővel

Hogyha a tanárnak tudomására jutott a zaklatás, fontos, hogy minél hamarabb beszéljen külön az áldozattal és az elkövetővel is. Mindkét esetben érdemes keresni egy nyugodt helyet, ahol a diákot figyelmesen végighallgatja úgy, hogy nem befolyásolja, és nem von le elhamarkodott következtetéseket. A gyermek csak akkor lesz őszinte, ha látja, hogy komolyan veszik, és nem (most) kritizálják. Jó, ha a tanár nyugodtan, vádló szavak mellőzésével beszél vele, és biztosítja arról, hogy segít megnyugtató megoldást találni. Súlyosabb esetben érdemes megbeszélni a diákkal, hogy mit fognak tenni, és az mivel jár, mint például szülők, pszichológus, esetleg tapasztaltabb kolléga vagy más szakember bevonása, iskolai, rendőrségi feljelentések, illetve jogi lépések. Fontos, hogy az áldozat érezze, hogy a tanár együttérez vele és nem hibáztatja azért, mert megfélemlítés áldozata lett. Hogyha a tanárnak volt része megfélemlítésben, akkor azt érdemes elmesélnie. Érdemes az áldozatot megnyugtatni, hogy „minden csoda három napig tart", a mai információs áradatban a társaik is rövid időn belül új témát fognak találni, és a történtek fokozatosan halványulni fognak. Az elkövető esetében nyilvánvalóvá kell tenni, hogy a megfélemlítő magatartás elfogadhatatlan, és az esetnek következményei lesznek, továbbá fontos, hogy belássa tettének következményeit és bocsánatot kérjen.

### 3.6.2. A zaklató üzenetek kezelése

Először is, a későbbi dokumentálás céljából, az áldozat készítsen másolatot a zaklatást tartalmazó oldalakról, az üzenetekre ne reagáljon (csak rontana a helyzeten), hanem amiket lehet, jelentse a szolgáltatónak, majd tiltsa le. Ezután néhány napig egyáltalán ne használja a zaklató online felületeket, ugyanis utólag, amikor már egy kicsit lenyugodott és megkapta a szükséges támogatást, talán valamivel kevésbé felkavaró élmény lesz számára elolvasni a bántó üzeneteket.

A tanároknak, családtagoknak és más támogatóknak is nagyon meg kell fontolniuk, hogy belefolynak-e a zaklató beszélgetésekbe, és ha igen, akkor hogyan. Ugyanis, az ő válaszaikat (még ha támogatóak is) az ismerőseik is látni és kommentelni fogják, a támogatók válaszaira további áldozatot sértő üzenethad is érkezhet, így tovább fog bővülni a bevont emberek és sértő üzenetek száma, így az áldozat szégyenérzete is. Ugyanakkor az is lehet, hogy ezek (egy részét) az áldozat már nem olvassa el. Ezért ajánlott az áldozattal privát üzenetben, telefonon vagy személyesen felvenni a kapcsolatot, illetve a családdal is beszélni, hogy ők is ennek megfelelően járjanak el. Ebben az esetben is

érvényes, hogy minden beavatkozást érdemes előtte az áldozattal megbeszélni. Ha mégis úgy döntenénk, hogy reagálunk a sértő üzenetekre, akkor sem érdemes támadni a zaklatókat, csupán az áldozat értékeit kiemelni.

### 3.6.3. Nyilvános megbeszélés

Először is fontos tisztázni, hogy nem mindig jó a gyermeknek, ha az esetet az osztályban beszélik meg, vagy szülői értekezletet hívnak össze. Előfordulhat, hogy a többi szülő nem érti, és eleve haragudni fog, hogy néhány gyerek rosszalkodásáért miért rángatták be, utána otthon (a gyereke előtt) méltatlankodni fog, és másnap az osztály még jobban bántani fogja az áldozatot. A szülők és a kortársak reakciója függ attól is, hogy mennyire sikerült megfelelő környezetet kialakítani, illetve felvilágosítani a szülőket az online zaklatás működéséről és következményeiről. A helyzet felmérésének fényében a gyerekkel meg kell beszélni, hogy akarja-e, hogy a történteket az osztályban vagy bármi más (nyilvános módon) megbeszéljék, hiszen lehet, hogy nincs olyan állapotban, hogy ezt fel tudja vállalni, ha pedig nem képes erre, akkor iskolaváltásra lehet szükség amellett, hogy pszichológus segítségét is kérni kell. Viszont jó esély van arra, hogy az áthelyezés önmagában nem fogja megoldani a problémákat: az áldozatot az új helyen is zaklathatják, és az agresszorok az osztályban is kiszúrhatnak újabb áldozatot. Ezért fontos a korábban leírt lépések megtétele is a további esetek megelőzése érdekében. A továbbiakban feltételezzük, hogy az áldozat beleegyezett a nyilvános megbeszélésbe.

A megbeszélés során nem büntető, hanem resztoratív igazságszolgáltatást érdemes megvalósítani, melynek célja a helyreállítás, a közös problémamegoldás, és az áldozat fájdalmának elismerése mellett az elkövető motivációinak megértése. Ennek során először az áldozat mondja el, hogy mit kér a társaitól, majd a zaklatók is elmondják, hogy ők mit éreznek, és mit kérnek. Érdemes a beszélgetésbe megfigyelőket is bevonni, akiknek nem az a szerepük, hogy szemtanúként megalapozzák az elkövetők megbüntetését, hanem hogy elmondják, ők mit tettek volna az adott helyzetben, hogyan tudták volna megelőzni a bántást. Bizonyos esetekben érdemes lehet a szülőket is bevonni a megoldásba.

Néhány hét után érdemes érdeklődni a diákoknál, hogy megoldódott-e a helyzet, milyen a kapcsolatuk, és jól érzik-e magukat.

## 3.7. Kortársmentor program

A gyerekek, különösen a tinik szívesebben és könnyebben beszélik meg a problémájukat közel azonos korú társaikkal. Ez derül ki a kortárs szexuális zaklatást vizsgáló deSHAME program felméréséből is, melyben az áldozatok az esetek 67%-ban beszélnék meg a történteket a barátaikkal [17]. Egy amerikai kutatás felmérése szerint (nem feltétlen online) zaklatás esetén a kortárs alábbi cselekedetei a következő mértékben váltak be (segítettek) [8]:

- Konfrontálódott az áldozat miatt: 36%
- Tanácsot adott: 49%
- Felhívta, bátorította az áldozatot: 51%
- Segített, hogy az áldozat elhagyja a helyszínt: 54%
- Csatlakozott az áldozathoz (időt töltött vele) az iskolában: 56%.

Látható, hogy a legfontosabb az áldozat lelki támogatása, tehát egy kortárs akkor is nagyon sokat tud segíteni, ha nem rendelkezik szakmai tudással.

A kortársmentor programnak lehet ennél komplexebb célja is, tipikusan egy olyan légkör kialakításának segítése, mely nem tartalmaz bántalmazást sem az iskolában, sem az iskolán kívül, sem az interneten. Természetesen ez nem valósítható meg néhány alkalmas felvilágosítással, hanem egy hosszú távú folyamat.

A program minden résztvevőjének hasznára válik. A kortársmentorok (más néven kortárssegítők) fejlődnek a kommunikáció, az empátia, és a felelősségvállalás területén. A diákok közössége segítséget kap, több lehetőségük lesz a segítségkérésre. A mentortanároknak a program lehetőséget biztosít a fejlődésre, a szakmai kihívásokra.

### 3.7.1. A kortárssegítők feladatai

A kortárssegítők feladatait többféleképpen is definiálhatjuk, attól függően, hogy milyen szerepet szánunk nekik. Így persze a programban résztvevő tanárok feladatai is többfélék lehetnek. A következőkben a Kék vonal *Kortárs részvétel és közösség a jó és biztonságos iskolai légkörért* projektjében [14] megfogalmazott kortárssegítő programból kiindulva határozzuk meg a diákok és tanárok feladatait.

A kortárssegítők olyan önkéntes diákok, akik vállalják, hogy részt vesznek az iskolában olyan programok szervezésében, melyek hozzájárulnak a megfelelő iskolai környezet kialakításához, és segítő hozzáállással fordulnak a társaik felé. A feladatra jelentkező diákoknak először egy képzésen kell részt venniük, ahol megtudják, hogy mi lesz pontosan a feladatuk, és megismerkednek a csoportdinamika alapjaival.

A kortársmentorok feladatai közé tartozik az újonnan érkezettek fogadása, akár iskolát váltó diákokról, akár az iskolát most kezdő osztályokról legyen szó. A mentoroknak szemmel kell tartaniuk az iskolai élet mindennapjait, segíteniük kell a diákok szabadidős tevékenységekbe kapcsolását. Fontos, hogy észrevegyék, ha valaki problémával küszködik és meghallgassák őt, segítsenek neki. A fentiekben láttuk, hogy a kapcsolati háló jelentősen csökkenti az áldozattá válás esélyét, ezért a kortársmentoroknak észlelniük kell azt, ha valaki magányos, és segíteniük kell neki bekapcsolódni a környezetbe. A kortárssegítőknek feladatuk, hogy a különböző programok szervezésében részt vegyenek, Ilyen programok célja lehet például a diákok érzékenyítése egymás és a diákokat fenyegető problémák iránt.

Emellett vannak olyan feladatok is, melyekről első hallásra úgy tűnhet, és a kortárs segítők is úgy gondolhatják, hogy ez is az ő feladatuk lenne, pedig nem az. Ilyenek például a szabályok betartatása, igazságtétel, mások problémájának megoldása, vagy a pszichológus helyettesítése.

A bevezető tréning után lesznek olyan diákok, akik mégsem vállalják így a feladatot, illetve akadhatnak olyanok is, akiket a tréning végére a program vezetői nem fognak alkalmasnak találni arra, hogy betöltsék ezt a feladatot. Fontos, hogy ilyenkor keressenek nekik olyan feladatokat, melyek jobban illenek hozzájuk.

### 3.7.2. A kortársmentor programban résztvevő tanárok feladatai

A program működéséhez szükség van néhány tanárra és más szakemberre (iskolapszichológus, pedagógiai asszisztens, gyermekvédelmi szakember), akik segítik a kortársmentorokat. Az ő feladatuk meghirdetni a programot, kiválasztani a jelentkezők közül a kortárs mentorokat, a kezdő tréninget megszervezni és megtartani. A munkájukhoz tartozik a mentorok felügyelete és bátorítása is, fontos, hogy a kortárssegítők bármikor tanácsot tudjanak kérni tőlük. Ehhez szükséges, hogy a tanárok magukat is továbbképezzék.

## 4. Az informatikatanár speciális lehetőségei

Az eddig felsorolt lehetőségekből az informatikatanár ugyanúgy kiveheti a részét, mint bármely más tanár. A továbbiakban azt vizsgáljuk, hogy ennél többet mit tehet informatikatanárként.

A legtöbb iskolában az a gondolatmenet merülhet fel, hogy mivel online történik a zaklatás, ezért az informatikatanár feladata a megelőzés. Felmerülhet a kérdés, hogy a mai világban tényleg szükséges a diákokat tanórán internetezni tanítani? Nem ismerik elég jól, ha már folyamatosan online vannak? Bár az internet bizonyos lehetőségeinek kezelését valóban jól ismerik a diákok, de a szabályokról, normákról és veszélyekről, illetve az internet számos hasznos lehetőségéről is általában kevesebb-

bet tudnak. Ez picit olyan mintha KRESZ ismerete nélkül vezetne valaki autót. Tudja, hogy hogyan kell megállni, de nem tudja, hogy melyik táblánál kell megállnia. A diákok is tudják, hogy mit hogyan kell elrejteni az interneten, hogyan kell letiltani valakit, de nem biztos, hogy minden esetben felismerik a veszélyt és azt, hogy most kell lépni ellene. Ezért fontos informatika órán is a veszélyhelyzeteket és elkerülésüket ismertetni velük.

## 4.1. A kerettanterv által biztosított lehetőségek

A kerettanterv alapján általános iskolában és középiskolában is van két témakör, melyek technikai információk tanítása mellett jelentős mennyiségű értékformáló, viselkedésalakító, azaz nevelő tevékenységet igényelnek. Ezek az Infokommunikáció és Az információs társadalom témakörök, melyek 2018-as NAT tervezetben összevontan Információs technológiák néven szerepelnek. Az ezekhez tartozó témákat a gyakorlati alkalmazások mellett más módszerekkel is lehet tanítani, például beszélgetésekkel, szituációkkal, drámapedagógiai módszerekkel vagy esetleg tréning formájában. Ezekben a témakörökben a kerettantervben vannak olyan témák, melyek keretében részletesebben lehet beszélni az internetes zaklatásról, és melyekre arányosan az alábbi időkeret fordítható:

**5.-6. osztály**

| | |
|---|---|
| Az online kommunikációban rejlő veszélyek elleni védekezés. | 0,5-1 óra |
| Az informatikai eszközök etikus használatára vonatkozó szabályok megismerése. Az információk megosztásának etikai kérdései. | 0,5 óra |
| A hálózat használatára vonatkozó szabályok megismerése, értelmezése. | 0,5 óra |

**7.-8. osztály**

| | |
|---|---|
| Az információforrások etikus felhasználása. | 0,5 óra |
| Az információ szerepe az információs társadalomban. | 0,5 óra |
| Az informatikai eszközök használatának következményei. | 0,5 óra |

**9.-10. osztály**

| | |
|---|---|
| Informatikai eszközök etikus használata.<br>Az informatikai eszközök használatának következményei a személyiségre és az egészségre vonatkozóan. | 0,5-1 óra |

Ily módon, összességében szerintünk az internetes zaklatással kapcsolatos ismeretekre körülbelül 2 óra jut, amit a diákok fejlettségi szintjétől függően érdemes az egyes évfolyamokra szétosztani.

Mivel egyes internetes zaklatással kapcsolatos ismeretek tanítása ténylegesen része a tananyagnak, az informatikatanár különösebb feltűnés nélkül megteheti, hogy kiadja a diákoknak (vagy legalábbis az agresszorokat tartalmazó csoportnak), hogy megfelelően összeállított szakirodalomból dolgozzák fel a témát, ami úgy megelőzés, mint esetkezelés szempontjából hasznos lehet, ugyanakkor nem kelti azt a látszatot, mintha a történtekről a tanár tudna és beavatkozna.

## 4.2. Kapcsolatok más informatikai témakörökkel

A kerettanterv számos további olyan témakört is tartalmaz, melyeknek ugyan nem képezi részét az internetes zaklatás, de nagyon fontos szerepet játszanak annak megelőzésében és megfelelő kezelésében. Ilyen témakörök a következők.

- Adatvédelem, az adatokkal, különösen a személyes adatokkal való visszaélések, veszélyek és következmények megismerése, azok kivédése, a védekezés módszereinek és szempontjainak megismerése.

- Az információk elemzése hitelesség szempontjából.
- Az információ értékként való kezelése, megosztása. Például, a megosztott ártatlannak látszó információ is segíthet a zaklatóknak.
- Infokommunikációs ismeretek. Például, nem kívánt üzenetek letiltása.
- Netikett ismerete. A kommunikáció írott és íratlan szabályai.

Ezen tartalmak megtanulásában és a tanultak alkalmazásában motivációt jelenthet annak megbeszélése, hogy hogyan segíthetnek az internetes zaklatás megelőzésében is.

## 4.3. További módszertani megfontolások

A hagyományos informatikatanítási módszerekhez képest az információs technológiák tanítása új módszertani kihívást jelent az informatikatanár számára, hiszen ez esetben nem elegendő az információkat megtanítani, hanem azt is el kellene érni, hogy a diákok olyankor is a tanultaknak megfelelően viselkedjenek, amikor azt a tanár nem látja, hiszen a tanárnak nagyon kevés rálátása van a diákok online tevékenységére. A diákok meggyőzésében pedig fontos szerepe van annak, hogy a diák értse, hogy milyen mechanizmusok játszódnak le, és miért fontos a tanultak betartása.

Ennek érdekében érdemes lehet olyan megközelítést is alkalmazni, hogy kifejezetten a potenciális veszélyekből indulunk ki (adathalászat, álprofilokkal lehetséges visszaélések, internetes zaklatás stb.) és azok kapcsán tárgyaljuk az internet etikus és biztonságos használatával kapcsolatos kérdéseket [21].

Mivel a témára igen kevés idő jut, úgy is érzékenyíthetjük a diákokat és új ismeretekkel láthatjuk el őket, ha nem az internetes zaklatás az óra témája. Más témakörökben bevihetjük a témát a feladatokhoz használt anyagokként. Például, szövegszerkesztésnél használhatunk olyan forrásfájlt, ami az adatvédelemről vagy a netikettről szól.

## 4.4. Kapcsolódás más tantárgyakkal

A zaklatás témaköre több tantárgyi óra során is felmerülhet. Például, több film is készült, ami a zaklatást, vagy konkrétabban az internetes zaklatást dolgozza fel [19, 24, 28]. Ezek közül egyet (vagy többet) mozgóképkultúra és média órán meg lehet nézni és beszélgetni róla. Magyar órán is felmerülhet olyan mű, ami a zaklatás témakörét taglalja.

Mielőtt bármilyen órán, fórumon beszélgetnénk is az internetes zaklatásról a diákokkal, mindenképpen avassuk be az iskolapszichológust is. Fontos, hogy felkészüljön rá, hogy esetleg lesznek diákok, akik az órák után ebben a témában fordulnak majd hozzá. Az iskolák többségében az iskolapszichológus nincs jelen napi rendszerességgel az iskolában, hiszen több iskola is tartozik egy pszichológushoz.

Mielőtt az informatika óra keretein belül is felmerülne ez a téma, érdemes lehet beszélni a kollégákkal, hogy hogyan igazodjunk egymáshoz a témát illetően. Többféle módon is megközelíthető a probléma. Megbeszélhetjük, hogy mindnyájan nagyjából ugyanabban az időszakban tárgyaljuk ezt a témát, hogy a diákok még intenzívebben halljanak róla. Az is egy lehetőség, hogy szándékosan nem egyszerre tárgyaljuk, hogy többször halljanak róla a diákok, többször szembesüljenek a problémával. Az is megvalósítható, hogy szervezünk egy tematikus hetet, ahol bár néhány tanórát elhagyunk, mégis mindegyik tantárgyhoz kapcsolódó ismeretanyag szóba kerül egymástól kevésbé elválasztva, mintha külön tanórákon beszélnénk róluk.

## 4.5. Szabályzat kialakítása

Az informatikatanárnak jelentős szerepe lehet a zaklatással kapcsolatos szabályzat technikai részének megalkotásában, az elfogadhatatlan viselkedés és eszközhasználat definiálásában és folyamatos aktu-

alizálásában oly módon, hogy ugyanakkor ez ne akadályozza indokolatlanul a technika pozitív használatát.

### 4.6. Beszélgetések szervezése a fiatalok online tevékenységeiről

Hasznos lehet, ha az informatikatanár irányításával a fiatalok időnként beszámolnak az aktuális digitális szokásaikról, online trendekről, népszerű alkalmazásokról, játékokról és oldalakról. Könnyen megtörténhet, hogy ezek egy része a tanárok számára is ismeretlen lesz, továbbá lehetőséget ad arra, hogy megbeszéljék a potenciális veszélyeket.

### 4.7. Szülők képzése

A második fejezetben tárgyaltuk, hogy mit tehet a szülő a gyermeke biztonságáért. A szülők viszont nem mind ismerik ezeket a lehetőségeket, vagy esetleg a számítógépet és az internetet nem tudják olyan szinten kezelni, hogy használják gyermekük megóvására. Ezért szükség lehet a szülők képzésére, ahol ezeket a lehetőségeket megmutatják nekik. A tantestületben nem feltétlenül csak az informatikatanár ismeri ennek az informatikai hátterét, de a szülők lehetséges, hogy tőle jobban elfogadják a segítséget, mert a technikai ismereteit hitelesebbnek tekintik.

### 4.8. Kortársmentorok képzése

Korábban már említettük, hogy a kortársmentoroknak többféleképpen is meghatározhatjuk a feladatait. Amennyiben úgy határozzuk meg kortárssegítők feladatait, hogy az informatikai háttértudást igényel, indokolt lehet az informatikatanár részvétele a mentortanárok között. Az informatikai részét mindenképp neki kellene megtanítani a programban részt vevő tanulóknak, de a többi részt is magára vállalhatja.

## 5. Összefoglalás és következtetések

Az internetes zaklatás valós problémát jelent a diákok számára, melynek megelőzése és megfelelő kezelése szempontjából alapvető fontosságú, hogy ismerjük a folyamat jellemzőit és működését, a beavatkozás lehetőségeit, illetve buktatóit is. Cikkünkben első sorban azokra az ismeretekre tértünk ki, melyek egy informatikatanár szempontjából hasznosak lehetnek, és igyekeztünk lehetőségeket, ötleteket bemutatni az ilyen helyzetek megelőzésére és kezelésére.

Láttuk, hogy a megelőzésben sokat segíthet olyan környezet megteremtése, mely elítéli egymás bántását, a diákok, szülők és tanárok képzése, a zaklatás jeleinek minél korábbi észlelése és megfelelő intézkedések foganatosítása. Tárgyaltuk azt is, hogy mit lehet tenni akkor, ha már megtörtént a zaklatás, és az informatikatanárnak milyen speciális további lehetőségei vannak. Valószínűleg tökéletes megoldás nincs, de az meggyőződésünk, hogy ha ilyen történik, az áldozatnak segíteni kell.

## Irodalom


1. 2003. évi CXXV. törvény az egyenlő bánásmódról és az esélyegyenlőség előmozdításáról, 10. § (1) bekezdés, módosítva a 2006. évi CIV. törvény által, 2006.XII.5., https://net.jogtar.hu/jogszabaly?docid=a0300125.tv (utoljára megtekintve 2019.11.10.)
2. Ybarra M. L. és Mitchell K. J. (2004): Online aggressor/targets, aggressors, and targets: a comparison of associated youth characteristics, Journal of Child Psychology and Psychiatry, Vol. 45(7) pp 1308–1316.
3. Sameer Hinduja és Justin W. Patchin (2007): Offline Consequences of Online Victimization: School Violence and Delinquency, Journal of School Violence, Vol. 6(3), 89.-111.
4. Slonje R. és Smith P. K. (2008): Cyberbullying: Another main type of bullying? Scandinavian Journal of Psychology, 2008, 49, 147–154.
5. 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 Informa-

tion 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.)

6. Domonkos Katalin: zaklatás, Osztályfőnökök Országos Szakmai Egyesületének kiadványa, https://osztalyfonok.hu/cikk.php?id=1033, 2012. február 8. (utoljára megtekintve 2019.10.26.)
7. Gimes Júlia: Az iskolai bántalmazás és a gyulladás, Magyar Tudomány, 2014.06.19., http://www.matud.iif.hu/2014/06/19.htm (utoljára megtekintve 2019.10.26.)
8. Dr. Parti Katalin, PhD: "Álmaimban Amerika": A bullying kezelése Massachusettsben, előadás "Az internet hatása a gyermekekre és fiatalokra" című konferencián, Budapesten, 2014.október 2-án, https://youtu.be/F_i7vaUyQhw (utoljára megtekintve 2019.11.03.)
9. Simon Dávid, Zerinváry Barbara, Velkey Gábor: Az iskolai bántalmazás megjelenése az 5-8. évfolyamos diákok körében: jelenségek és magyarázatok a normál és alternatív tantervű iskolákban, Oktatáskutató és Fejlesztő Intézet, TÁMOP-3.1.1-11/1-2012-0001 XXI. századi közoktatás (fejlesztés, koordináció) II. szakasz, 2015, http://iskon.opkm.hu/admin/upload/Osszegzo_tanulmany.pdf, (utoljára megtekintve 2019.11.02.)
10. Dr. Lénárd Kata, PTE BTK: Iskolai zaklatás – A gyermekek közötti terrorizálás. megfélemlítés pszichológiája, Nyitott Egyetem, Pécs, 2015, https://www.youtube.com/watch?v=M76_uwhOE4w (utoljára megtekintve 2019.11.02.)
11. Dr. Makó Ferenc: Tanulásmódszertan, 2.4. Kooperatív tanulási módszerek, Óbudai Egyetem, 2015.09.19., https://www.tankonyvtar.hu/hu/tartalom/tamop412b2/2013-0002_tanulasmodszertan/tananyag/JEGYZET-20-2.4._Kooperativ_tanulasi_mods.scorml (utoljára megtekintve 2019.11.10.)
12. Mendi-Kozma Laura: A kiberbűnözés egyes aspektusai. Az online zaklatás. Károli Gáspár Református Egyetem, ŰJTK., 2016., http://nmhh.hu/dokumentum/192062/MendiKozma_Laura__A_kiberbunozes_egyes_aspektusai__online_zaklatas.pdf, (utoljára megtekintve 2019.11.02.)
13. KiVa program, 2016, http://www.kivaprogram.net/hu, (utoljára megtekintve 2019.11.02.)
14. Szűcs Katalin, Reményiné Csekeő Borbála: Mentori kézikönyv iskolai kortárssegítő programokhoz, Kék Vonal Gyermekkrízis Alapítvány, 2016
15. Az Emberi Erőforrások Minisztériuma által kiadott Módszertani útmutató a gyermekvédelmi észlelő és jelzőrendszer működésére, 2016, https://www.kormany.hu/download/c/72/b0000/modszertani_utm_bantalmazas_megelozes_2.pdf (utoljára megtekintve 2019.11.02.)
16. Villányi Gergő: Cyberbullying, megalázás – Mit tehetünk a zaklatók ellen?, iPon, 2016.04.10., https://ipon.hu/elemzesek/cyberbullying_megalazas_%E2%80%93_mit_tehetunk_a_zaklatok_ellen/2772/1 (utoljára megtekintve 2019.11.02.)
17. deSHame projekt, 2017, https://www.kek-vonal.hu/index.php/hu/szolgaltatasok/projektek/490-14, (utoljára megtekintve 2019.11.02.)
18. 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.)
19. 7 figyelemfelkeltő filmes alkotás az iskolai erőszakról, Nők Lapja, 2017.10.04, https://noklapja.nlcafe.hu/ajanlo/2017/10/04/61-figyelemfelkelto-filmes-alkotas-az-iskolai-eroszakrol/ (utoljára megtekintve 2019.11.09.)
20. Galán Anita, Rákó Erzsébet, Szabó Gyula (2018): A virtuális világ veszélyei és a gyermekvédelem aktuális kérdései. Különleges Bánásmód, IV. évf. 2018/4. szám, 61–72. DOI 10.18458/KB.2018.4.61
21. Holló Csaba, Álprofilok használata az etikus és biztonságos internethasználat tanításában, INFODIDACT 2018 (2018. 11. 22-24), Informatika Szakmódszertani Konferencia, elektronikus kiadványa, 59-70, Zamárdi,

Hungary, április, 2019, ISBN: 978-615-80608-2-0., https://people.inf.elte.hu/szlavi/InfoDidact18/Manuscripts/HCs.pdf (utoljára megtekintve 2019.11.10.)

22. Óvónők.hu: Óvodai bullying?, 2018.02.16, https://ovonok.hu/2018/02/19i-bullying/, (utoljára megtekintve 2019.11.02.)
23. Rácz Johanna: A magyar gyerekek ötöde bántalmazott már egy másik gyereket az interneten, Qbit, 2018.10.04., https://qubit.hu/2018/10/04/a-magyar-gyerekek-otode-bantalmazott-mar-egy-masik-gyereket-a-neten?fbclid=IwAR3zoOYM60Nt8hhjqABOc5T83jG9YQOvPtd-61tKDXmkZnuZ5aksV3wnnAs (utoljára megtekintve 2019.11.02.)
24. Fábián Sebestyén: Behálózott gyermeklelkek – 1. rész, 2018. 11. 22., https://vasarnap.hu/2018/11/22/behalozott-gyermeklelkek-1-resz/ (utoljára megtekintve 2019.11.09.)
25. How Bullying Affects the Brain, Neuroscience News, 2018.12.12., https://neurosciencenews.com/brain-bullying-10331/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+neuroscience-rss-feeds-neuroscience-news+%28neuroscience+News+Updates%29&fbclid=IwAR29fYijcLHGGpQ-w2LJFj8tDV2e8Agn-YlbR9E7OBC5_sGCN14csqgTBbs (utoljára megtekintve 2019.10.26.)
26. Allen D. Truell, Jensen J. Zhao, Edward J. Lazaros, Christopher Davison, Dakota L. D. Nicley: Cyberbullying: important considerations, Issues in Information Systems Volume 20, Issue 2, pp. 83-88, 2019, http://www.iacis.org/iis/2019/2_iis_2019_83-88.pdf (utoljára megtekintve 2019.10.26.)
27. Gál Diána Babett: A gyermekeket is érinti az internetes zaklatás – Ne hagyjuk szó nélkül, OrosCafé, 2019.03.20., https://oroscafe.hu/2019/03/20/a-gyermekeket-is-erinti-az-internetes-zaklatas-ne-hagyjuk-szo-nelkul/?fbclid=IwAR3u_Nv12lAD2HZjwWMtDQ5o9zlZfW__BKtHX33cF2aojT4 (utoljára megtekintve: 2019.11.02.)
28. Villányi Gergő: Az internetes zaklató – filmkritika és tanulságok, Digitális Család, 2019.09.23, https://www.digitaliscsalad.hu/biztonsag/az-internetes-zaklato-filmkritika-es-tanulsagok (utoljára megtekintve 2019.11.09.)
29. Dr. Kósa Éva: Mindenki (előbb-utóbb) szülő lesz, X. Nemzetközi Médiakonferencia - Budapest, Magyar Tudományos Akadémia - 2019. október 1-2., https://youtu.be/HrU9m5FVx7U?list=PL54fQF5Awd_kuWG7BGlCqVuJmEQtUtItr (utoljára megtekintve 2019.11.02.)
30. Fekete Zsombor: Néhány gondolat a cyberbullyingról..., 2019.10.16., https://www.facebook.com/permalink.php?story_fbid=110403640376307&id=110338667049471 (utoljára megtekintve 2019.11.10.)
31. ISKON Tudásközpont, http://iskon.opkm.hu/, (utoljára megtekintve 2019.11.02.)

# A formális szemantika interaktív oktatása tételbizonyító rendszerben

Horpácsi Dániel[1], Németh Dávid János[2], Kaposi Ambrus[3]

[1]daniel-h@elte.hu, [2]ndj@inf.elte.hu, [3]akaposi@inf.elte.hu
ELTE IK

**Absztrakt.** Az ELTE informatikai képzéseiben gazdag múltra tekint vissza a programozási nyelvek formális szemantikájának oktatása. Karunkon, más elméleti tárgyakhoz hasonlóan, hagyományosan a Formális szemantikát is tábla és papír köré szerveztük, a 2008-ban indult programtervező informatikus mesterképzésben pedig a tárgy gyakorlata sajnos egyáltalán nem kapott helyet. Sokéves kihagyás után az előző félévben ezt géptermi kurzusként indítottuk újra, amely keretében immáron interaktív és végrehajtható módon mutatjuk be a programozási nyelvek tételbizonyító rendszerben történő modellezését. Cikkünkben ismertetjük a gyakorlat megtervezése és lebonyolítása során szerzett tapasztalatainkat; kiértékeljük a tételbizonyító rendszer használatának látható következményeit; végül szót ejtünk a módszerünk más kurzusokra való kiterjeszthetőségéről is.

**Kulcsszavak:** formális szemantika, tételbizonyító rendszer, Coq, interaktív gyakorlat

## 1. Bevezetés

Egyetemi szintű programozó-képzésben elengedhetetlen a matematikai, formális leírások használata, oktatása. Az ELTE programtervező mesterképzésen kötelező tantárgy a Formális szemantika, amelynek keretében a hallgatók elsajátítják a programozási nyelvek matematikai alapokon nyugvó definíciós módszereit. A tárgy nagy mennyiségű absztrakt fogalommal és formális jelölésrendszerrel dolgozik, amelyet a kevésbé matematikus beállítottságú hallgatók nehezen fogadnak be. Mint minden „elméleti" tárgynál, itt is szükséges a matematika „csomagolása", amellyel könnyebben emészthetővé tesszük az egyébként száraz anyagot. Részletes, gyakorlatias példákkal lényegesen könnyíthető az absztrakt fogalomrendszer megértése, ám a fiatal generáció akkor fogadja be a legkönnyebben az elvont fogalmakat, ha megérinthetővé, kipróbálhatóvá, kísérletezhetővé tesszük azokat. A formális szemantika kurzus gyakorlatán nemcsak prezentálunk szemléletes példákat, de bevezettük a tételbizonyító rendszer használatát is, amelynek segítségével a definíciókon túl a tételek és bizonyítások is interaktívan oktathatóak.

## 2. A formális szemantika kurzus

A programozási nyelvek a természetes nyelvekkel ellentétben mesterségesek: azért alkotta őket az ember, hogy a számítógépet irányítsa. A természetes nyelvekkel szemben a programozási nyelvek mondatainak egyértelmű és pontos jelentéssel kell rendelkezniük, amely alapján a számítógép mérlegelés és további kérdések nélkül végre tudja hajtani őket, ahol ezen végrehajtás hatása matematikai precizitással megfogalmazható. Számítógépes programok jelentésfogalmával, és általában a programozási nyelvek formális definícióival foglalkozik a Formális szemantika.

Az ELTE programtervező matematikus és programtervező informatikus MSc szakok tanulmányi hálójának mindig fontos eleme volt a Formális szemantika tantárgy, aminek keretében a programozóhallgatók elsajátíthatják a programozási nyelvek definícióinak megértéséhez és formalizációjához szükséges matematikai eszközöket. A kurzus körbejárja a szintaxis, statikus szemantika és dinamikus szemantika matematikai leírásának lehetőségeit, beleértve az operációs, denotációs és axioma-

tikus stílusban megadott szemantikákat. A különböző definíciós módszerek minden esetben valós, elterjedt (főleg imperatív) programozási absztrakciók szemantikájának leírásával kerülnek bemutatásra, ami elősegíti a megértést, kapcsolja a megszerzendő tudást a korábbi tanulmányokhoz, és hasznos a meglévő tudás elmélyítésében is.

## 2.1. A kurzus célja

A kurzusnak több célja is van. Az első és legfontosabb a programozási nyelvek precíz, formális definiálásának népszerűsítése, továbbá a formális definíciók szerepének tisztázása a nyelvek megértésében, összehasonlításában, valamint a programok funkcionális tulajdonságainak, helyességének vizsgálatában. A hallgató a különböző általa ismert programozási absztrakciók (pl. vezérlési szerkezetek) formális megadása útján mélyebben megérti ezen absztrakciók működését, kölcsönhatásait, korlátait, valamint átfogóbb képet kap a különböző nyelvekben különbözőképp leírt absztrakciók hasonlóságairól és különbségeiről (pl. elágazások és ciklusok, függvényabsztrakciók és kivételkezelés megannyi formája a modern nyelvekben).

A formális szemantikadefiníció motivációja és közvetlen gyakorlati haszna mellett legalább ugyanolyan fontos a matematikai eszközkészlet megismertetése és megtanítása. A hallgató átismétli a fordítóprogramok tanulmányozása során megismert szintaxisdefiníciós módszereket, majd megismeri az operációs és denotációs definíciós módszert a jelentés megadására. Az operációs szemantikákat (small-step és big-step) a konfigurációknak, valamint azok átmeneteit induktívan megadó levezetési szabályok formátumának és megtervezésének megértésével, a denotációs szemantikákat pedig a szemantikus függvények felírásával, a kompozicionalitás elvének megismerésével sajátítja el a hallgató.

Az alkalmazott matematikai eszközkészlet felhasználja a halmazelmélet és a csoportelmélet egyes elemeit, induktív definíciókat és azokon végzett mintaillesztést, indukciót, a fixpontelmélet releváns részeit. A tárgy által feldolgozott téma, valamint a feldolgozás módja igen absztrakt, amit a hallgatók egy része nehezen tud befogadni. A példák sokat segítenek egy-egy új fogalom megértésében, ám a gyakorlat tudja csak igazán elmélyíteni az előadás anyagát.

## 2.2. A kurzus korábbi megvalósítása

A Formális szemantika tárgy a programtervező matematikus képzésben előadás és gyakorlat keretében került oktatásra, ahol a gyakorlaton az előadás által tárgyalt matematikai eszközök használatával foglalkoztak a hallgatók. Habár volt gyakorlat, az papír alapú volt, nem alkalmazott semmilyen interaktív környezetet a definíciók és tételek kipróbálására, a szemantikákkal való kísérletezésre. 2008-ban aztán a programtervező informatikus képzésben a tárgy gyakorlata nem is kapott helyet, így az osztott képzésben már csupán előadás keretében tanulhatták az MSc hallgatók a Formális szemantikát. Ez nyilvánvalóan nem tett jót a kurzus hatékonyságának és népszerűségének, hiszen a tantárgy kiterjedt és absztrakt fogalomrendszerét gyakorlás nélkül nehéz teljes mértékben megérteni és elsajátítani.

A 2008-2018 közötti tízéves időszak közepén oktatóváltás történt, aminek következményeként gyakorlatiasabb stílust kapott az előadás. Az órák elején és végén népszerű programozási nyelvek bonyolult konstrukciói kerültek bemutatásra és elemzésre, a kedvcsinálás mellett pedig a hallgatók bevonása is nagyobb hangsúlyt kapott: a denotációs szemantikákat implementáltuk a Haskell programozási nyelvben, az operációs szemantikákat pedig formalizáltuk egy termátíró keretrendszerben, ezzel végrehajtható, módosítható, kipróbálható definíciókat adva a hallgatók kezébe. Ezek a lépések érezhetően javítottak a tárgy megítélésén, de a következő természetes lépés a gyakorlat újbóli bevezetése volt.

## 2.3. Az új gyakorlat

A 2018-ban újragondolt programtervező informatikus mesterképzési tantervbe visszakerült a Formális szemantika gyakorlat, amely újabb lendületet tudott adni a szemantika gyakorlatias oktatásá-

nak. A tárgy oktatói – akik egyúttal a cikk szerzői – nekiláttak a gyakorlat megtervezésének és lebonyolításának. A cikk a sokéves kihagyás utáni első gyakorlati csoportok új tematika szerinti oktatásának tapasztalatait mutatja be.

A gyakorlat tematikáját és számonkérést úgy kellett megalkotnunk, hogy bármelyik elsőéves programtervező mesterhallgató képes legyen elsajátítani a tudást és elvégezni a kurzust. Az oktatók között abban teljes egyetértés volt, hogy a Formális szemantika gyakorlatias oktatásnak elengedhetetlen eleme a definíciókkal való kísérletezés, újabb és újabb nyelvi elemek megadása; ahogy abban is egyetértettünk, hogy ennek a kísérletezésnek interaktívan, számítógépen kell történnie. Mivel a definíciók mellett a tételek kimondását és bizonyítását is szerettük volna gyakoroltatni a hallgatókkal, olyan rendszert kellett választanunk, amely tételbizonyításra és/vagy ellenőrzésre is alkalmas. A következő fejezet részletesen tárgyalja, hogy mely funkcionális nyelvek és tételbizonyítók közül választottunk, és hogy milyen múltra tekint vissza a gépi tételbizonyítás bevonása az egyetemi gyakorlatokba.

## 3. Tételbizonyító rendszerek az oktatásban

A tételbizonyító rendszerek olyan szoftverek, amelyek lehetőséget adnak:

- absztrakt modellek minden részletre kiterjedő definiálására,
- ezen modellek felett értelmezett állítások kimondására,
- majd az igaz állítások matematikailag megalapozott szabályrendszer mentén történő, gépileg ellenőrizhető, interaktív bizonyítására.

A Formális szemantika kurzus géptermi gyakorlatának tervezése során több ilyen rendszert is fontolóra vettünk, figyelembe véve az irodalomban hozzájuk köthetően fellelhető oktatási tapasztalatokat, valamint a körvonalazott céljainkhoz való illeszthetőségüket.

A Haskell programozási nyelv [6] szigorú értelemben véve ugyan nem tekinthető tételbizonyító rendszernek, viszont tisztán funkcionális jellege és egyszerű szintaxisa alkalmassá teszi programozási nyelvek modellezésére. Benne az absztrakt szintaxist algebrai adattípusokkal, a szemantikát pedig függvényekkel írhatjuk le, denotációs stílusban, végrehajtható módon. A gyakorlatban építettek már rá szemantika kurzust például az Umeåi Egyetemen (Svédország) [11], viszont az alkalmazását mi, dedikált tételbizonyítási funkcionalitás hiányában, elvetettük.

Az Agda [1] egy, a Haskellénél erősebb, függő típusrendszert támogató funkcionális programozási nyelv. Az ilyen típusrendszerek sajátossága, hogy a bennük előforduló típusokat értékektől tehetjük függővé, leírva például a statikusan 4-hosszú vektorok típusát. Egy ennyire erős típusrendszer, a típusok és a matematikai állítások között kapcsolatot teremtő Curry-Howard izomorfizmus révén lehetőséget ad arra, hogy benne állításokat típusok, bizonyításokat pedig futtatható programok formájában, konstruktívan írjunk le. Az Agda nyelvet jelenleg is aktívan használják szemantika oktatására többek között az Edinburgh-i Egyetemen (Egyesült Királyság) és a Vermonti Egyetemen (Amerikai Egyesült Államok) [10], mi viszont célszerűbbnek láttuk egy olyan rendszer bevezetését, amely élesebben leválasztott, a hallgatók által papíron megszokottakhoz közelebb álló tételbizonyító funkcionalitással rendelkezik.

Ennek a követelménynek eleget tesz az Isabelle/HOL tételbizonyító [7], mely szintén, gyakorlatban alátámasztottan alkalmas programozási nyelvek szemantikájának oktatására (Müncheni Műszaki Egyetem, Németország) [2]. Ugyanakkor a típusrendszere nem támogat függő típusokat, így benne szükségessé válhat, hogy egy erősebb típusrendszerben egyébként a típusokba (állításokba) kódolt, és így a rendszer által automatikusan kezelt információkat a tételkimondások és bizonyítások során manuálisan kelljen karban tartani. Az ezzel járó komplexitást más eszközben potenciálisan elkerülhetjük.

A választásunk végül a Coq tételbizonyító rendszerre [5] esett. Az Agda nyelvezetéhez hasonló, funkcionális és függő típusos leírónyelve kellően kifejező és konstruktív, viszont dedikált, imperatív stílusú bizonyítási résznyelvvel is rendelkezik, ami szerencsés összhangban áll az általánosságban imperatív paradigmához szokott hallgatók előismereteivel. Széleskörű pedagógiai alkalmazása (például a svédországi Chalmers Műszaki Egyetemen és az amerikai egyesült államokbeli Pennsylvaniai Egyetemen is felhasználták szemantika oktatására) [3], részletesen kidolgozott segédanyagai [9] és aktív támogatottsága [4] tovább növelték a belé vetett bizalmunkat.

# 4. Interaktív formális szemantika gyakorlat

A Formális szemantika gyakorlat feladata, hogy segítse az előadás anyagának megértését az ott tárgyalt definíciók, példaprogramok, valamint további példák formalizálásával. Mivel a hallgatók túlnyomó többségének nincsen gyakorlata semmilyen tételbizonyító rendszerrel, a gyakorlati óráknak gyorsan és hatékonyan kell a hallgatót rutinos tételbizonyító (Coq) felhasználóvá tenniük anélkül, hogy a hangsúly a szemantika fogalmairól a tételbizonyító technikai részleteire tolódna el.

## 4.1. Irányelvek

**Előadáson megértés, gyakorlaton kódolás.**

Az előadások során a tananyag esetenként kifejezetten összetett példákon kerül bemutatásra, hogy a példák minél inkább mintázzák a valódi programozási nyelvekben előforduló konstrukciókat. Ezeket részletesen tárgyalja az előadás, de a formalizációjuk komplexebb annál, mint amit egy kezdő magától le tud írni a tételbizonyító rendszerben. A kész formalizációkat kiadhatnánk a hallgatóknak (a korábbi végrehajtható szemantikákkal ezt tettük), de az új gyakorlatokon még inkább a hallgató bevonása a cél. A gyakorlati órák többségén nem meglévő formalizációkat olvasnak vagy módosítanak a hallgatók, hanem informális és fél-formális definíciókat, tételeket visznek be a rendszerbe önállóan, "üres lappal" indulva. Ezzel biztosítjuk, hogy az idő nem a komplex példák megértésére és magyarázatára, hanem – még ha kicsi példákon is, de – a formalizáció gyakorlására kerül felhasználásra. Ennek köszönhetően a szemantikán marad a fókusz, viszont a folyamatos önálló kódolással viszonylag rövid idő alatt gyakorlatot szerez a hallgató a tételbizonyító rendszer használatában.

**Először megértés, aztán kódolás.**

Az előző gondolat persze nem jelenti azt, hogy a hallgatónak a definíció, tétel vagy bizonyítás megértése nélkül kellene elkezdenie kódolni. Sőt, a gyakorlatoknak feladata, hogy az anyag elméleti megértését különválassza a Coqban való formalizációjától. A gyakorlat kisebb feladatokat próbál teljes egészében megoldatni a hallgatókkal, de a tételbizonyítóban való formalizációt minden esetben megelőzi az elmélet tökéletes megértése. Ha szükséges, ehhez a táblára vagy füzetbe is írunk, de ez az „analóg" kidolgozás jóval kisebb hangsúlyt kap, mint más matematikai tantárgyak gyakorlatain.

**Fokozatosság a tételbizonyító megismerésében.**

A Coq megannyi nyelvi elemmel és funkcióval segíti a kényelmes és hatékony formalizációt, de ha kezdettől fogva arra törekszünk, hogy a hallgató jó Coq programozó legyen, nem szemantikát fogunk tanítani neki, hanem Coqot szemantika példákkal. A tételbizonyító nyelvét és funkcióit fokozatosan vezetjük be gyakorlatról gyakorlatra három-négy újabb elemmel (hasonló sorrendben, mint a Programming Language Foundations könyv [9]), ennek köszönhetően a hallgató fokozatosan szerez jártasságot a Coq fogalomrendszerében és nyelvezetében, nem veszik el annak részleteiben.

**Fokozatosság a tételek bonyolultságában.**

A hallgatóknak sokszor nemcsak a tételbizonyítókkal, a tételbizonyítással sincs elég tapasztalatuk. A Formális szemantika gyakorlatoknak ezért sokszor bizonyításelméletet is kell tanítaniuk, és ezzel egy időben a bizonyítás formalizációjának módszerét is tárgyalniuk kell. Ahhoz, hogy a hallgató ne ve-

szítse el idejekorán a kedvét és lelkesedését a tételbizonyító rendszer használata iránt, kellő körültekintéssel kell egyre hosszabb és bonyolultabb bizonyításokat készíteni és készíttetni a gyakorlatok során. Tapasztalataink szerint egyszerű esetszétválasztással megadott függvények formalizációja például nem jelent gondot egyik mesterszintű hallgatónak sem, de a levezetési szabályok induktív definíciókká alakítása már esetenként fejtörést okoz. Kellő előkészítéssel folyamatos sikerélményt tudunk biztosítani a hallgatónak, aminek következményeként a tételbizonyító rendszer használatát pozitív élményként fogja elkönyvelni, és ez fontos mérföldkő lehet a szakmai fejlődésében.

**Folyamatos munka és számonkérés.**

A gyakorlat számonkérését úgy terveztük meg, hogy hétről hétre dolgoznia kelljen a hallgatónak: a 90-90 percnyi előadás és gyakorlat mellett heti egy házi feladat és heti egy rövid zárthelyi dolgozat tartja folyamatos fejlődésben a kurzus résztvevőit. Minden kiadott feladat a Coq rendszerben kerül megoldásra, tovább gyorsítva a tételbizonyító használatának elsajátítását. A feladatokat úgy találtuk ki, hogy a heti házi feladat mindig a következő zárthelyit mintázza, tehát a házi feladat megoldása elősegíti a gyakorlati számonkérés jó teljesítését; ez tovább növeli a hallgatói motivációt a folyamatos munkában.

## 4.2. Tematika

A gyakorlat tematikája elsősorban az előadás meglévő anyagára és a már említett könyv [9] megközelítésére lett felépítve. Szerencsére a könyv kitűnő alapot ad a Coq gyors és gördülékeny bevezetésére, miközben az általa tárgyalt szemantika példák sok helyen kitűnően egybecsengnek az előadáson tárgyaltakkal, mert mindkét forrás az imperatív programozási paradigma absztrakcióit dolgozza fel. Rövid mérlegelés után úgy állítottunk össze a tematikát, hogy a félév első felében a könyvet [9], a félév második felében pedig elsősorban az előadásanyagot követi és formalizálja. A gyakorlat tematikája a következő fontosabb elemekből áll össze:

- Bevezető, a tételbizonyító rendszer használata. Egyszerű típusok Coqban: bool és nat, kapcsolódó lemmák kimondása, bizonyítás esetszétválasztással.
- Szintaxis megadása induktív definícióval (mély beágyazás), egy kifejezésnyelv leírása denotációs szemantikával, kiértékelés szerkezeti rekurzióval. Statikus szemantika: kifejezésen értelmezett egyszerű leképezések (literálok száma, műveletek száma), induktív bizonyítások a kifejezésnyelv szerkezete alapján. Kifejezések leképezése ekvivalens kifejezésekre (optimalizációs lépések), a szemantika megőrzésének induktív bizonyítása.
- Állapot (változókörnyezet) bevezetése, a kifejezésnyelv szintaxisának és szemantikájának bővítése. Állapotokkal kapcsolatos lemmák kimondása és bizonyítása kifejezések speciális eseteire. Statikus elemzés: szabad és kötött változók, zárt kifejezés fogalmának megadása függvénnyel és induktívan definiált relációval. Definíciók ekvivalenciája, zártságra vonatkozó lemmák kimondása és bizonyítása.
- A kifejezésnyelv small-step és big-step operációs szemantikájának megadása induktívan definiált relációval. Levezetési szabályok formalizálásának gyakorlása. A megadott denotációs és operációs szemantikák ekvivalenciájának bizonyítása a kifejezésnyelvre.
- Imperatív utasítások nyelvének formalizálása. Szintaxis, denotációs szemantika. Fixpontelmélet, a terminálás kérdése. Big-step operációs szemantika formalizálása, levezetések, determinisztikusság. A denotációs és operációs szemantikák ekvivalenciája a While nyelvre. Programminták ekvivalenciája (egyszerű optimalizációs lépések helyességének belátása). További vezérlési szerkezetek formalizálása operációs és denotációs szemantika megadásával: for ciklus, repeat-until ciklus.
- Haskell kód exportálása a helyes Coq programból, gyakorlati alkalmazások.

**Bonyolultabb fogalmak elmélyítése**

Amint említettük, vannak olyan fogalmak és módszerek, amelyeket még mesterhallgatók is lassan, fokozatosan tudnak csak elsajátítani. A gyakorlati kurzus talán abból a szempontból a legfontosabb, hogy ezeket a bonyolult, absztrakt fogalmakat fokozatosan nehezedő példákon mutatja és gyakoroltatja be. Az eddigi tapasztalataink szerint a következő fogalmak tanítása lényegesen hatékonyabb lett a gyakorlatoknak köszönhetően:

- *Levezetési szabályok és levezetés*: sokszor nem világos a hallgatók számára, hogy egy operációs szemantikát leíró szabályrendszert mért pont úgy formalizálunk, ahogyan; mit lehet a (levezetési szabály premisszáját és konklúzióját elválasztó) vonal fölé, a vonal mellé, és a vonal alá írni. A Coqban megadott induktív típus rákényszeríti a hallgatókat, hogy pontosan megértsék a szabályokat és a formalizmus elemeit. A Coq szemantikájából adódó mintaillesztés kapcsán a hallgató könnyebben el tudja képzelni, miért és hogyan kerülnek alkalmazásra a levezetési szabályok.
- *Strukturális indukció*: bonyolultabb szerkezetű típusok esetén nem mindig érthető a hallgató számára az induktív bizonyításhoz használt séma – nem világos, mit mi után, milyen hipotézis mentén kell belátni. A papíros bizonyítások sokszor könnyebben olvashatóak, de sajnos bármit „eltűrnek"; ezzel ellentétben Coqban interaktívan tudnak bizonyítani, a tételbizonyító rendszer lépésről lépésre vezeti őket a bizonyításon, világosan megadva, hogy mik a hipotézisek és mit kell azokból belátni, valamint hibás lépéseket nem enged meg.
- *Kompozicionalitás*: a kompozicionalitás elvét sokszor csak bemagolják a hallgatók, de nem válik teljesen világossá számukra, hogy miért is kell így megadnunk a denotációkat. Amikor a programok vagy kifejezések szerkezete szerinti indukcióval bizonyítanak, jól látszik a tételbizonyító rendszerben, hogy a kompozicionalitásnak köszönhetően az indukciós hipotéziseik egybevágnak azokkal a belátandókkal, amelyek az összetett szerkezetekre adódnak. További következmény, hogy a denotációs szemantikák rekurzív szerkezetekre való megadásánál kompozicionális definíciók esetében egészen biztosan szerkezetileg csökkenő (primitív) rekurzív hívások adódnak, amit a Coq biztosan terminálló rekurziónak fog értelmezni.

## 5. A tételbizonyító rendszer használatának kihívásai

A tételbizonyító rendszer tudáselmélyítés szempontjából előnyösnek tekinthető tulajdonságai között akadtak olyanok is, amelyek megoldandó pedagógiai problémákat is felvetettek. Ezek egy része a Coq alapnyelvének jellemzőiből következett, míg mások mögött a papírra vetett és a gépbe kódolt bizonyítások közötti részletszigorúság állt.

A programozási nyelvek szintaxisának és operációs szemantikájának megadásához használt induktív típuskonstrukciók helyenként nehezen voltak értelmezhetők az objektumorientált nyelvekhez szokott hallgatók számára – ezt a Coqban definiált típusokhoz Java-beli megfelelőt mutatva igyekeztünk ellensúlyozni.

A hallgatók korábban technikai részleteket mellőző vagy átugró bizonyításokat írtak papíron. Ehhez képest Coqban a bizonyítások legkisebb részleteit is ki kellett fejteni, ez gyakran nehézséget okozott. Ilyen jelenség például, hogy míg többszörös esetszétválasztást szükségessé tevő bizonyításokban papíron elegendő csak az érdekes ágakat kiemelni a triviálisak közül, addig egy tételbizonyítóban akkurátusan foglalkozni kell minden egyes lehetőséggel. Szerencsére ez a probléma sem bizonyult megoldhatatlannak: bizonyítási taktikákat ismétlő nyelvi elemekkel az érintett gépi bizonyítások újra közel hozhatók a megszokott papíros megfelelőikhez.

# 6. Eredmények

A gyakorlat bevezetésének eredményességét objektíven a gyakorlati és a kollokviumjegyekkel tudjuk mérni. A gyakorlati számonkérés módjából és értékelési skálájából fakadóan az első gyakorlatot végző évfolyam gyakorlati jegyei nagyon jók lettek, így a képzés eredményességét, a minőség javulását inkább az előadás-vizsgajegyeken kívánjuk szemléltetni.

A Formális szemantika vizsga hagyományosan szóbeli felelet, amely egyrészt a hallgató lexikális tudását, másrészt az anyag elsajátításának módját, a megértés mértékét vizsgálja. A hallgatók nagy számára való tekintettel évek óta egy írásbeli beugró dolgozat is része a vizsgának, amely egyrészt kiszűri a felkészületlen hallgatókat a szóbeli vizsgáról, másrészt a sikeres beugróval szóbeli felelet nélkül elégséges jegyet szerezhetnek a téma iránt kevésbé lelkesedő hallgatók.

## 6.1. Vizsgaeredmények

A gyakorlat egyértelműen javította a vizsgaeredményeket. Ez valószínűleg annak köszönhető, hogy a hallgatók az elméleti anyag bemagolása helyett a fogalmak folyamatos megértésének köszönhetően értve tudtak felkészülni az előadás tematikájából. A következő számszerűsített javulást könyvelhettük el a 2017/2018/2 és a 2018/2019/2 félévek között.

**Átlagok javulása**

Az összes vizsgázót tekintve az érdemjegyek átlaga 2,6-ről 3,14-ra nőtt a beugró feladatok jellegének vagy a tételsornak a megváltoztatása nélkül. A vizsgáztató személye is változatlan maradt. Az átlagba minden érdemjegyet (1-5) beleszámítottunk.

**Több szóbeliző**

Habár a sikeres írásbeli beugró után a hallgatók megszakíthatják a vizsgájukat egy elégséges érdemjeggyel, szóbeli vizsga tétele nélkül, a szóbelire jelentkező hallgatók aránya 33%-ról 48%-ra emelkedett. Tehát a gyakorlat olyan mértékben javította a hallgatók magabiztosságát az anyag megértését illetően, hogy az összes hallgatónak majdnem fele szóbeli vizsgán szerezte az érdemjegyét, a korábbi egyharmad helyett.

**Szóbelizők átlagának javulása**

A szóbeli felelet valamely anyagrész részletes bemutatását jelenti, ahol a hallgatónak arról kell bizonyságot tennie, hogy nemcsak megtanulta, de érti is az előadáson elhangzottakat. Nem kis meglepetésünkre a szóbeli feleletek érdemjegyeinek átlaga is emelkedett, 3,9-ről 4,2-re, ami annak is köszönhető, hogy sokkal többen tudták jól elmagyarázni a kihúzott tételüket, és szereztek jó vagy jeles érdemjegyet.

## 6.2. Szubjektív eredmények

Oktatói szemmel egészen szembetűnő javulás mutatkozott a szóbelik gördülékenységében, a hallgatók magabiztosságában. Jobban megértették a fogalomrendszert, a bizonyítások lépéseit és szerkezetét. A vizsgajegyekben mutatkozó javulás szubjektíven is érezhető volt.

A hallgatók részéről is kifejezetten pozitív visszajelzéseket kaptunk a kurzussal kapcsolatban. Ugyan a Formális szemantika előadás nem volt kifejezetten népszerűtlen, a gyakorlatnak és az interaktív oktatási módszernek köszönhetően a hallgatók lelkesebben álltak hozzá a tárgyhoz. Úgy gondoljuk, hogy a fent vázolt mód az egyetlen, amellyel matematikai tárgyakból hosszú távú, használható tudást tudunk átadni az átlagos mesterhallgatónknak.

## 6.3. Coq más kurzusokban

Célunk, hogy a Formális szemantika kurzus esetében szerzett tapasztalatainkat a képzésünk más, hasonló jellegű tárgyainak átalakításához is felhasználjuk. Az aktuális, 2019/2020/1 félévben a Nyel-

vek típusrendszere című, a formális típuselméletbe gyakorlatiasan és szemléletesen bevezető tárgyat egészítettük ki egy Coqra alapozott gyakorlattal, részben a Formális szemantika mintájára. Az átállás ugyan nem zökkenőmentes (a gépi modellezés által megkívánt alaposság miatt a gyakorlat az előadásnál lassabban tud csak haladni), de a Formális szemantikánál érzékelhető (a félév végéig egyelőre jórészt szubjektív) pozitívumok itt is mutatkoznak.

Az eddig említett két kurzuson kívül a jövőben más potenciális tárgyak is szóba jöhetnek a programozáselmélet, a logika és a matematika témaköréből. Az előző félév során (2018/19/2) például az Osztott rendszerek specifikációja és implementációja című kurzusunkhoz született egy hallgatói formalizáció [8]. Hosszabb távon gondolkozva meggyőződésünk, hogy a gépileg ellenőrzött modellezés releváns tárgyakon átívelő szabványosítása számottevő mértékben járulna hozzá a hallgatók szakmai jártasságának elmélyítéséhez, és így az informatikai képzések minőségének, színvonalának emeléséhez.

## 7. Összefoglalás

Cikkünkben bemutattuk, hogy miként indítottunk el egy tételbizonyító rendszerre épülő, interaktív géptermi kurzust egy korábban sokáig csak elméleti előadást magában foglaló, programozási nyelvek formális szemantikájával foglalkozó tárgyhoz.

A mögöttes motivációnkat az eredeti kurzus képzésünkön belüli pozíciójának, történetének ismertetésével magyaráztuk, kiemelve azt az előnytelen helyzetet – fontos elméleti anyag a megértését segítő gyakorlat nélkül –, ami sok éven át korlátozta a tárgy céljainak megvalósítását.

Az új gyakorlat alapelvének, a tételbizonyító szoftver használatának rögzítése után számba vettük a szóba jövő rendszereket, majd ezek közül kiválasztottuk a Coqot, amely a jól szeparált modellező és tételbizonyító résznyelvei, kifejező típusrendszere, valamint gazdag segédanyag-irodalma révén ideális jelöltnek bizonyult.

A tananyag megalkotásának irányelveit és a meglévő előadásanyaggal való kapcsolatát, illetve a gyakorlati kurzus optimális lebonyolítását szerintünk leginkább segítő számonkérési módszert három pontban foglaltuk össze: először megértés, aztán kódolás; fokozatosság; folyamatos munka és számonkérés. Ezekre építve egy tematikatervezetet is felvázoltunk, amelyet (többek között) abban a szellemben állítottunk össze, hogy pusztán előadással nehezen átadható fogalmak (levezetés, indukció, kompozicionalitás) elsajátíthatóságán javítani tudjunk.

A tételbizonyító rendszer bevezetése természetesen technikai és pedagógiai kihívásokkal is járt, melyek közül a sok hallgató számára szokatlan leírónyelvet, továbbá a gépi bizonyítások papíros megfelelőik után helyenként (logikailag indokolt, de a gyakorlatban) csak nehezen védhető komplexitásnövekedését emeltük ki. Ezen problémákra megoldásokat is javasoltunk rendre a megszokott nyelvekkel való párhuzamvonás, valamint az automatizáció használatának formájában.

A bevezetett géptermi gyakorlat eredményességét az objektív összehasonlításra lehetőséget adó vizsgajegyek mentén értékeltük. Örömmel tapasztaltuk, hogy ezekben számottevő javulás volt kimutatható: a korábbi félévekhez képest nemcsak az átlagok emelkedtek, de a sikeres beugrót követő fakultatív, elégségesnél jobb jegy szerzéséhez szükséges szóbeli feleletre is több hallgató vállalkozott. Kísérletünk sikerességét látva módszerünket az aktuális félévben alkalmazzuk egy másik tárgyra is, a jövőben pedig tervezzük több tárgy irányelveink szerinti átalakítását is.

A programok magas szintű, precíz modellezése azontúl, hogy elengedhetetlen részét képezi a szoftverhelyesség matematikai eszközökkel történő vizsgálatának, hozzájárul az informatikushallgatók programnyelvi jártasságának, absztrakciós készségének fejlesztéséhez is. Bízunk abban, hogy az itt megosztott tapasztalataink legalább bátorítást, de akár tanácsokat, segítséget is adnak ahhoz, hogy ezt és a hozzá kapcsolódó, alapvetően elméleti tudományterületeket gyakorlatias és interaktív módon oktatva hozzuk közelebb a hallgatókhoz.

# Irodalom


1. Az Agda programozási nyelv (2007-től)
   https://wiki.portal.chalmers.se/agda/pmwiki.php (utoljára megtekintve: 2019.11.04.)
2. Tobias Nipkow, Gerwin Klein: *Concrete Semantics with Isabelle/HOL*. (2014-től)
   http://concrete-semantics.org/ (utoljára megtekintve: 2019.11.04.)
3. Coq In The Classroom (2017-től)
   https://github.com/coq/coq/wiki/CoqInTheClassroom (utoljára megtekintve: 2019.11.04.)
4. A Coq tételbizonyító GitHub repozitóriuma (2007-től)
   https://github.com/coq/coq (utoljára megtekintve: 2019.11.04.)
5. A Coq tételbizonyító rendszer (1989-től)
   https://coq.inria.fr/ (utoljára megtekintve: 2019.11.04.)
6. A Haskell programozási nyelv (1990-től)
   https://www.haskell.org/ (utoljára megtekintve: 2019.11.04.)
7. Az Isabelle tételbizonyító rendszer (1986-tól)
   https://isabelle.in.tum.de/ (utoljára megtekintve: 2019.11.04.)
8. Donkó István: *orsi-formalization*. Az ELTE-IK Osztott rendszerek specifikációja és implementációja című tárgyának hallgatói Idris-formalizációja. (2019)
   https://github.com/Isti115/orsi-formalization (utoljára megtekintve: 2019.11.04.)
9. Benjamin C. Pierce et al.: *Programming Language Foundations*. Software Foundations series, volume 2. Elektronikus könyv. (2018)
   http://www.cis.upenn.edu/~bcpierce/sf (utoljára megtekintve: 2019.11.04.)
10. Philip Wadler: *Programming Language Foundations in Agda*. Konferenciaanyag, 21st Brazilian Symposium (SBMF 2018). Salvador, Brazil, November 26–30, 2018.
11. O. Johansson: *Programming language semantics*. Kurzusanyag. Umeå Egyetem, Umeå, Svédország (1993-1998)
    http://oldwww.cs.umu.se/local/kurser/TDBC05/ (utoljára megtekintve: 2019.11.04.)

# A *Szoftvertechnológia* tárgy agilisra vált

Ilyés Enikő[1], Pintér Balázs[2], Szendrei Rudolf[3], Cserép Máté[4]

{[1]ilyese, [2]pinter, [3]swap, [4]mcserep}@inf.elte.hu
ELTE IK

**Absztrakt.** Az ELTE Informatikai Karán a *Szoftvertechnológia* tárgy keretein belül olyan témakörökről tanulhatnak az alapképzéses hallgatók, mint követelményelemzés, tervezési minták, verifikáció-validáció, tesztelés stb. A gyakorlati órák során a cél az volt, hogy a hallgatók alkalmazzák az előadáson bemutatott témaköröket és egyénenként egy-egy szoftvert fejlesszenek egy nagyobb programozási feladat megoldására. A 2018-as tanterv-módosítás során a tárgy oktatói úgy döntöttek, hogy agilis csoportmunka bevezetésével támogatják a tárgy keretein belül megszerzett elméleti tudás elmélyítését. A cikk beszámol azokról a kihívásokról, melyeket az agilis transzformáció okozott, feltárja a kihívásokra adott válaszokat, ismerteti a 2019 tavaszi félévében szerzett tapasztalatokat és mérési adatok formájában mutatja be a csoportmunkára vonatkozó visszajelzéseket.

**Kulcsszavak:** agilis, oktatás, csoportmunka

## 1. Bevezetés

Az informatika jellegű felsőoktatási képzések nagyrésze a képzések elejére az algoritmikus gondolkodás kialakítására irányuló tárgyakat ütemez. Elsőként egyszerű feladatok megoldására irányuló rövid programokat írnak a hallgatók a gyakorlati órák során és a beadandókban – rendszerint önállóan. A képzés előrehaladtával viszont sor kell kerüljön egyre komplexebb feladatok megoldására – valódi szoftver létrehozására, hiszen az informatika jellegű felsőoktatási képzések kimeneteként elvárt, hogy a végzettek képesek legyenek üzleti értéket képviselő szoftverterméket előállítani. Az is fontos, hogy olyan munkaszervezési eljárásokat, szoftverfejlesztési módszertanok is elsajátítsanak, melyek segítségével hatékonyan tudnak bekapcsolódni egy szoftverfejlesztéssel foglalkozó vállalat életébe.

Az informatikai jellegű felsőoktatási képzésekben rendszerint megtalálható legalább egy olyan tárgy, melynek gyakorlatához tartozó követelmény: a hallgatók fejlesszenek egy szoftvert egy nagyobb programozási feladat megoldására valamely megfelelő szoftverfejlesztési módszertant alkalmazva. Az már változó, hogy ezt a szoftvert egyénileg vagy csoportban kell fejleszteniük. A választott módszertanok közül a vízesés modell és az agilis modellek a legelterjedtebbek. Amennyiben korábban vízesés modellt használtak az adott tárgy esetében, az agilis modellre való váltás egy érdekes kihívás. Az agilis transzformáció kérdései ugyanúgy felmerülnek az oktatási intézmények esetében, akár az ipari intézmények esetében.

A szakirodalomban megjelennek olyan cikkek, melyek beszámolnak agilis transzformációkról. Az oktatók gyakran sikert látnak az átállást illetően. [1] részletesen számol be arról, hogy milyen hátrányai voltak a vízesés modellnek saját kurzusán való alkalmazása esetében, és hogyan oldotta fel ezek egy részét az agilis modellre való váltás. [2], [3] ugyancsak esettanulmányokat tartalmaz agilis módszertanok szoftverfejlesztés témájú kurzusokon való alkalmazására vonatkozóan. [4] részletesen tárgyalja, hogy miért indokolt napjainkban olyannyira, hogy a szoftverfejlesztési képzéseken a tananyag része legyen az agilis módszertanok témaköre.

Az Eötvös Loránd Tudományegyetemen mi is fontosnak ítéltük meg az agilitás témakörének gyakorlati megismertetését a hallgatókkal. Cikkünk egy esettanulmányt tartalmaz: Az ELTE programtervező informatikus alapképzéséhez tartozó *Szoftvertechnológia* tárgy agilis transzformációjáról számolunk be részletesen. Az olvasó tájékozódhat arról, hogy milyen módon, milyen szerepkörök és

eszközök bevezetésével hajtottuk végre a tárgy megváltoztatását, és ezáltal milyen eredményeket, visszajelzéseket sikerült elérnünk. Beszámolónk és elemzésünk által ötleteket meríthet felsőoktatásbeli szoftverfejlesztési kurzusok agilis transzformációjához.

A második fejezetben beszámolunk arról, hogy milyen helyzetből indítottuk az átállást. A harmadik fejezetben az átállás kivitelezésének módjáról számolunk be. A negyedik fejezet a tapasztalatokat és visszajelzéseket tartalmazza. Az ötödik fejezet összegzés jellegű.

# 2. A módosítás előtti állapot

## 2.1. Az új *Szoftvertechnológia* tárgy előadásának elődje

A *Szoftvertechnológia* tárgy előadásának elődjét a már több mint egy évtizede futó Programozási technológia 2. tárgy előadása jelentette. A tárgy első félévében főként a statikus és részben a dinamikus UML modellezés volt a téma. Az előadás bemutató anyagai Java nyelven íródtak, és fő szerepet kapott az objektumorientáltság.

A tárgy második félévében a hangsúly az objektumorientált tervezésre került, ahol egy szoftver teljes életciklusának bemutatása volt a cél. A szoftver életciklusából külön kiemeltük a megvalósíthatósági tanulmány készítését, a követelmény-elemzést, a specifikáció készítést, valamint ezen információk birtokában a tervezést, megvalósítást, és végül, de nem utolsó sorban a verifikációt és a validációt. A tervezéshez, valamint az azt megelőző fázisokhoz bemutatásra kerültek a kapcsolódó UML diagramok, megtámogatva azokat esettanulmányokkal.

A félév során jelentős hangsúly került a tervmintákra, melyek ma már elválaszthatatlan részét képezik egy objektumorientáltan implementált szoftver minőségi kivitelezésének.

Az előadás a gyakorlatot megtámogatva olyan további Java ismereteket is nyújtott, melyek a szálkezeléssel, illetve adatbázis-kezeléssel kapcsolatosak, és amelyek elengedhetetlenül fontosak voltak a tárgy gyakorlatához kapcsolódó házi feladatok, illetve év végi zárthelyik teljesítéséhez.

A kurzust elvégző hallgatók a szoftverfejlesztéssel kapcsolatban tehát megszerezhették azokat a kompetenciákat, amelyek egy szoftver megtervezéséhez, illetve kivitelezéséhez kapcsolódnak.

## 2.2. Az új *Szoftvertechnológia* tárgy gyakorlatának elődje

Az új *Szoftvertechnológia* tárgy elődjének gyakorlata szervesen kapcsolódott az előadáshoz oly módon, hogy azon egy szoftvert kellett kifejleszteni objektumorientáltan, Java nyelven, végigkövetve a termék teljes életciklusát.

A szoftver kifejlesztése négy fő fázisból állt, melyekre a hallgatók egy-egy érdemjegyet kaptak oly módon, hogy azok átlaga lett végül a félév végi érdemjegyük, amennyiben a vizsga zárthelyijük sikerült.

A négy fő fázis a követelmény-elemzés, a specifikáció, a tervezés, valamint az implementáció volt, ahol az utolsó fázisnak tartalmaznia kellett a tesztelés forgatókönyvét is. A gyakorlat a fázisokra egyenként kb. három-négy hetet biztosított a fázisok nehézségének függvényében. A gyakorlatban ez azt jelentette, hogy főként az implementáció igényelt több időt a hallgatók részéről.

A hallgatók által megoldandó házi feladatokhoz a gyakorlatokon példák kerültek bemutatásra, illetve a felmerült kérdések megvitatása volt még kiemelt jelentőségű. Az egyes fázisok lezárását a gyakorlaton történő házi feladat bemutatás jelentette. Ez gyakorlatvezetőktől függően történhetett közvetlen módon, vagy pedig kivetítőn a teljes gyakorlati csoport előtt. Utóbbi megoldás mind az interakció, mind pedig a közös tapasztalatszerzés okán hasznosnak bizonyult.

A félév során megoldandó házi feladat egy adatbázis-kezelőre épülő információs rendszer megvalósítása volt. A konkrét beadandó feladatot a gyakorlatvezetők választották ki egy négy-öt információs rendszert bemutató listából, és adták ki egyesével a hallgatóknak véletlenszerűen. Az elkészí-

tendő szoftverrel kapcsolatos közös kritériumokat, valamint a szükséges funkciók halmazát írott formában előre megkapták a hallgatók a félév első gyakorlatán.

# 3. Az új *Szoftvertechnológia* tárgy

## 3.1. Az újítás indokai

A valós szoftveripari projektben a résztvevőknek egyre ritkábban kell csak önállóan dolgozniuk, jellemzően kisebb-nagyobb csapatokban kell feladataikat teljesíteniük. Ezen kihívásokra és elvárásokra reagálva a programtervező informatikus alapképzésben célként jelent meg, hogy a kimeneti feltételeket teljesítő hallgatók a szoftvertechnológiai képzésük részeként csapatmunkában végzett szoftvertervezést és -fejlesztést is végezzenek. Mindezek során elméletben, de gyakorlatban is megismerkedjenek az elterjedt szoftverfejlesztési módszertanokkal (különös hangsúlyt fektetve az agilis módszerekre), valamint a munkájukat támogató projekteszközökkel.

## 3.2. Az új *Szoftvertechnológia* tárgy előadása

Az előadások kiegészítésre kerültek a csapatmunkában történő specifikálást, implementálást és tesztelést segítendő projekteszközök elméleti hátterének és gyakorlati példáinak bemutatásával, összesen 4 témát érintve: projekt menedzsment eszközök, verziókezelés, build rendszerek (ANT, Maven), folyamatos integráció és kiadás. Az egyes témák nem tömbösítve, hanem a szemeszter során egyenletesen kerültek a tematikába beillesztésre, hogy a gyakorlatokat kísérve mindig az éppen aktuális projektfeladatokat támogassák. A módosított előadás tematika így az 1.táblázat szerint alakult.

| Hét | Előadás tematika |
|---|---|
| 1. | Szoftverfejlesztési folyamat |
| 2. | Követelmény specifikáció |
| **3.** | **Projekt menedzsment eszközök, verziókezelés** |
| 4. | Objektumorientált tervezés |
| 5. | Objektumorientált tervezési szempontok és minták |
| **6.** | **Build rendszerek** |
| 7. | Verifikáció és validáció, Egységteszt JUnit-tal |
| 8. | Szálkezelés – 1. |
| 9. | Hálózatkezelés |
| **10.** | **Continuous integration and delivery** |
| 11. | Szálkezelés – 2. |

| | |
|---|---|
| 12. | TDD, Clean Code |
| 13. | Agilis szoftverfejlesztési módszertanok, Scrum |

**1. táblázat:** A módosított előadás tematika

## 3.3. Módosítás mibenléte a gyakorlatot illetően

A gyakorlatok tekintetében a tantárgy revíziójának elsődleges célja az volt, hogy a hallgatók ne egyénileg, hanem kisebb, jellemzően 3-4 fős csapatokba szerveződve készítsenek el egy már komplexebb alkalmazást, amely tipikusan perzisztálási és hálózati funkcionalitással is rendelkezik. Az alkalmazás elkészítésének annak teljes életciklusára ki kellett terjednie a követelmény-analízist, a specifikációt, az implementációt és a tesztelést is magába foglalva. A gyakorlati munka a Scrum módszertan elemeit használva került megszervezésre, ahol a hallgatók felelőssége volt, hogy az egyben Scrum masterként is eljáró gyakorlatvezető segítségével a feladatokat megfelelően felosszák egymás között és önálló csapatként kivitelezzék a szoftvert. A csapatok a heti gyakorlati órákon Scrum meetinget tartottak, 3 hetente, összesen 4-szer a félév során pedig bemutatóra került sor.

A projektmunka a GitLab [5] webes projektvezető szolgáltatás kötelező használatával került megtámogatásra. A választás azért erre a termékre esett, mert integráltan tartalmazza mindazokat a projekteszközöket (projektmenedzsment, verziókezelés, folyamatos integráció), amelyeket a kurzus keretében a hallgatókkal meg kívántunk ismertetni, így az ezekkel az eszközökkel jellemzően még csak most először találkozó hallgatók figyelme és időráfordítása nem több különböző eszköz elsajátítása között oszlott meg.

A hallgatók gyakorlati jegyet kaptak a tárgyból. A négy mérföldkő (egybeesik a bemutatók időpontjával) teljesítését követően az adott munkaszakaszra vonatkozóan az oktatók pontszámokat osztott ki a csoportoknak. Ha nagyon szembetűnő volt, hogy egy csoporton belül egyes tagok sokkal többet dolgoztak, akkor az oktató ellenőrizte azt is, hogy a csapatoknak kiosztott pontszámokat körülbelül megfelelő arányban osszák szét a hallgatók egymás között. A négy mérföldkő során összegyűjtött pontszámok összege képződött le a tárgy gyakorlati érdemjegyeként.

## 3.4. A módosítás kihívásai és az azokra adott válaszok

Mivel első ízben váltottunk a tárgy gyakorlatai óráin belül csoportmunkára és agilis munkaszervezésre, külön figyelmet kellett szentelnünk annak, hogy a váltás minél gördülékenyebben menjen. Erre a célra vezettünk be a tárgy szervezői között egy „csoportműködés mentora” szerepkört.

A csoportműködés mentora elsőként a tárgy gyakorlatvezetőivel vette fel a kapcsolatot a félév megkezdése előtt, és felkészítőt tartott nekik arra vonatkozóan, hogy pontosan mi az ő szerepük a csoportok kísérését illetően a gyakorlatok során. Fontos volt ezt alaposan tisztázni, mivel a tárgy felépítése a megszokottnál bonyolultabb helyzetbe hozta a gyakorlatvezetőket. Ők voltak ugyanis egyszemélyben a tárgy gyakorlatvezetői, a csoportok termékgazdái és Scrum mesterei is. Ahhoz, hogy ezek a szerepek ne keveredjenek szétválaszthatatlan módon össze – esetleg zavart képezve a hallgatók fejében is e fogalmakat illetően – előre leszögeztük, hogy melyik órán, milyen szerepkörben lesznek jelen jellemző módon a gyakorlatvezetők. A szerepkörök érvényesülése a 2. táblázat szerint alakult.

| Hét | Gyakorlat | Gyakorlatvezető szerepköre |
|---|---|---|
| **1.** | Feladat és számonkérés ismertetése | Gyakorlatvezető<br>Termékgazda |
| **2.** | Napi Scrum | Termékgazda<br>Scrum mester |
| **3.** | Napi Scrum | Termékgazda<br>Scrum mester |
| **4.** | Bemutató: Követelményelemzés<br>(Használati esetek diagram, User story-k) | Termékgazda |
| **5.** | Napi Scrum | Scrum mester |
| **6.** | Napi Scrum | Scrum mester |
| **7.** | Bemutató: Megoldási terv<br>(Osztály diagram, szekvencia diagram) | Termékgazda |
| **8.** | Napi Scrum | Scrum mester |
| **9.** | Napi Scrum | Scrum mester |
| **10.** | Bemutató: Prototípus<br>(Részleges implementáció, leegyszerűsített grafika) | Termékgazda |
| **11.** | Napi Scrum | Scrum mester |
| **12.** | Napi Scrum | Scrum mester |
| **13.** | Bemutató: Végleges termék, tesztelés | Termékgazda |

**2. táblázat:** Scrum szerepkörök érvényesülése a gyakorlatok során

A három szerepkörben röviden a következő feladatok érvényesültek:

- *Gyakorlatvezető:* Tisztázza a tárgy teljesítésének követelményeit. Technikai elakadásokban segít.
- *Termékgazda:* Képviseli egy potenciális ügyfél igényeit, az üzleti szempontokat.
- *Scrum mester:* Segítséget, lendületet ad a csoport önállóságának kialakításához.

Az 1.,2.,3. órán leginkább a termékgazda szerepköre érvénysült, mert a csapatoknak ekkor az volt a feladata, hogy megértsék, hogy mit vár el tőlük „a megrendelő". Hasonlóan, a bemutató jellegű órákon (4.,7.,10.,13.) is a termékgazda szerepe érvényesült, hiszen ilyenkor a csoportok bemutatták az utolsó iteráció, futam eredményét, a gyakorlatvezetők pedig az ügyfél szempontjából kellett reagáljanak a bemutatókra. A bemutatók közötti, „napi Scrum" típusú órák során Scrum mester szerepkörben voltak leginkább jelen a gyakorlatvezetők. Minden csoporthoz egyenként mentek oda,

és levezettek egy napi Scrum-ot. Jellemzően a következő kérdések mentén folyt a napi Scrum: „Mit végeztél el az elmúlt találkozó óta?", „Mit tervezel megvalósítani a következő találkozóig?", „Van valamilyen elakadásod?". A hallgatóknak egyenként kellett válaszolniuk e három kérdésre.

A gyakorlatvezetői felkészítő során a csoportműködés mentora igyekezett megerősíteni a gyakorlatvezetőket abban, hogy milyen fontos, hogy az adott szerepkörből annak megfelelően szólaljanak meg, például: A bemutatók során valóban képviseljék az ügyfél hangját, és ne csak a kód működésére, hanem az ügyfél szempontjából általában olyan fontos „külsőségekre" (a szoftver felhasználói felülete, a bemutatók minősége, a csapat bizalomkeltő hozzáállása) is figyeljenek. A napi Scrum-ok esetében ne próbálják külső irányítással megoldani a csapat összes elakadását, hanem Scrum mesterek módjára inkább erősítsék a csapatok önállóságát, erősítsék bennük a problémamegoldó képességet, bátorítsák őket őszinte, tiszteletteljes, nyílt kommunikációra. Ennek hangsúlyozása azért volt nagyon fontos, mert ennek betartása volt az agilitás érvényesülésének záloga, tekintettel az Agilis Kiáltványban [6] megfogalmazott elvekre.

A gyakorlatvezetői felkészítőt követően a csoportműködés mentora külön kommunikációs csatornát [7] hozott létre a gyakorlatvezetők számára. Ezen a felületen lehetőség volt arra, hogy a gyakorlatvezetők kérdezzenek, ha elakadásuk volt a csoportmunka kapcsán, megtárgyaljanak eseteket, fórumszerűen beszélgessenek a csoportmunkával kapcsolatos tapasztalataikról, vagy éppen személyes konzultációt, óralátogatást kérjenek a csoportműködés mentorától.

A félév közepén, illetve végén sor került a tapasztalatok személyes, közös megbeszélésére is. A csoportműködés mentora szervezte, és vezette le ezeket a találkozásokat, illetve rögzítette az itt elhangzó kérdéseket, gondolatokat, ötleteket. A leírt jegyzőkönyvek megosztásra kerültek a csoportműködéssel kapcsolatos csatornán, így azok is értesülhettek az itt elhangzottakról, akik valamilyen okból nem lehettek személyesen jelen. A közzétett információhoz többször hozzá is szóltak utólag is a gyakorlatvezetők – véleményükkel, meglátásaikkal egészítették ki azt. Néhány példa a gyakorlatvezetők félévközepi visszajelzéseiből:

- „A csapatmunka szervezés újszerűsége (hogyan osszuk fel a feladatokat, hogyan osszuk be az időt megfelelően stb.) sokakat váratlanul ér, időre van szükségük, hogy beletanuljanak. Emellett viszont túl sok az új szakmai anyag is. Mindkettőre nem tudnak fókuszálni kellőképpen a hallgatók." Hasznos felvetés abból a szempontból, hogy erre a jövőben számítunk, legyünk türelmesek a csapatmunka kialakulásának készsége kapcsán, hiszen ez részben a tárgyban fókusz is. A félév végi következtetések során az a döntés is megszületett, hogy a tárgy célját (bemutatott témaköröket, elméleti anyagot) kicsit le kell csökkenteni.
- „Több példa kódra és a mérföldkövek kapcsán elvártak szigorúbb meghatározására vágynak a hallgatók." – A visszajelzéshez például a következő hozzászólás érkezett a fórumon: „Több tárgyunk kapcsán sajnos az a tapasztalat, hogy a túl sok példakód csak azok értelmezés nélküli összemásolását (gányolását) eredményezi többeknél, így nem biztos, hogy a több példakód jó irány. Legyen mindenre példa, de ne beadandó szintű kész megoldások."
- „A napi Scrum-ok maximum ¾ óra alatt lemennek. Így viszont úgy tűnik, hogy nem használjuk ki megfelelően a gyakorlat idejét." – Ehhez a visszajelzéshez nagyon változatos hozzászólások érkeztek a fórumon: többen ugyanis inkább azt tapasztalták, hogy nem is elegendő az óra ideje a napi Scrumokra. De arra is volt példa, hogy más gyakorlatvezető tippeket adott arra, hogyha mégis hamarabb lemegy a napi Scrum, akkor mivel lehet kitölteni a gyakorlati óra hátralévő részét (pl. példakódok mutatásával.)
- „A követelményelemzés és tervezés szakasz sok időt elvesz a félévből, kevés az implementációra maradt idő, későn kezdik el a kódolást." Ez a visszajelzés a félév során többször előjött, így a félév végén a tárgy következő iterációjára tekintettel az a döntés született, hogy kevesebb lesz a követelményelemzésre és tervezésre szánt idő, és a hallgatók korábban elkezdhetik majd a fej-

lesztést. Ez a megközelítés várhatóan az agilis szoftverfejlesztés erősebb érvényesülését is lehetővé tesz majd.

A csoportműködés mentora nem csak a gyakorlatvezetők kísérése által, hanem egy kérdőívhonlap páros működtetése által is segítette a hallgatói csoportok működését és az agilis szoftverfejlesztés érvényesülését. A gyakorlatok tematikájából szembetűnő lehet, hogy a Scrum elemek közül a „Visszatekintés" nem szerepel a gyakorlatokon. Ennek az volt az oka, hogy sem az időbe, sem a gyakorlatvezetői erőforrásba nem fért bele, hogy minden munkaszakasz után levezessenek a gyakorlatvezetők a következő gyakorlat keretén belül egy-egy „Visszatekintőt" minden egyes csapattal. Ugyanakkor a „Visszatekintő" fontos eleme a csoportműködésnek, jó lehetőség arra, hogy tudatosodjon a csapattagokban a hatékony működéshez szükséges intézkedések és hozzáállás fontossága. Különösen hasznos ez a kevés csapatmunkával kapcsolatos tapasztalattal rendelkező személyek és az újonnan alakult csapatok esetében. Ebből az okból kifolyólag a csapatműködős mentora a hagyományos Scrom-os „Visszatekintés" részinti helyettesítése céljából egy „Visszatekintő" kérdőívet hozott létre, melyet a csapattagok egyenként kellett kitöltsenek a bemutatós alkalmakat követően, illetve bátorítást kaptak arra is, hogy megbeszéljék egymással mindazokat a szempontokat, melyeket egyénenként meg tudtak fogalmazni a kérdőívek kitöltése során. A kérdőív összeállítása kapcsán szempont volt az, hogy gyorsan kitölthető legyen, motiválva arra a hallgatókat, hogy az összes kérdésre válaszoljanak. Az is fontos volt, hogy a kérdőív „ne adjon a szájukba" szavakat (pl. feleletválasztásos kérdések által), hanem inkább gondolkodásra késztesse őket meglátásaik, véleményük saját szavas megfogalmazása által. A kérdőív online volt elérhető [8]. A kérdések a következők voltak:

- Milyen volt a közérzeted a csapatban a futam során? (1 (nagyon rossz) -> 5 (nagyon jó))
- Mennyire működött csapatként a csoport a futam során? (1 (nagyon rossz) -> 5 (nagyon jó))
- Értékeld a csapat teljesítményét a futam kapcsán! (1 (nagyon rossz) -> 5 (nagyon jó))
- Értékeld a csapat kommunikációját erre a futamra vonatkozóan! (1 (nagyon rossz) -> 5 (nagyon jó))
- Értékeld a csapat lelkesedését ebben a futamban! (1 (nagyon rossz) -> 5 (nagyon jó))
- A mi csapatmunkánk erőssége volt ebben a futamban … (kifejtés néhány szóban, mondatban)
- A mi csapatmunkánk gyengesége volt ebben a futamban… (kifejtés néhány szóban, mondatban)
- Szerintem ezt tehetnénk, hogy (még) jobb legyen a csapatmunkánk … (kifejtés néhány szóban, mondatban)
- Elmondom a csapattársaimnak az előbbi folyamatjavító lépéseimet! (igen/nem/egyéb)

A „Visszatekintő kérdőív" kitöltéseit a csapatműködés mentora összesítette, elemezte, majd ezeket közzétette egy publikus honlapon, ahol úgy a hallgatók, mint a gyakorlatvezetők elérhették. A hallgatók értesítve voltak afelől, hogy ajánlott elolvasniuk a visszatekintések eredményeit, szembesülniük azzal, hogy a többiekhez képest hogyan áll csapatuk, esetleg mit tanulhatnak tőlük (főleg a folyamatjavító ötletek kapcsán, vagy a gyengeségeik és erősségeik felismerését illetően).

A félév utolsó hetében a csapatműködés mentora külön 20 percet kapott az utolsó előadás idejéből arra, hogy a visszatekintések eredményeit személyesen is visszacsatolja a hallgatóknak. A hallgatók érdeklődve hallgatták vissza, hogy „miket írtak", és remélhetőleg ez is hozzájárult ahhoz, hogy a tárgy az agilis szoftverfejlesztés és a csapatműködés szempontjait elmélyítse bennük. (A kérdőívek eredményeiről cikkünk negyedik fejezetében számolunk be részletesen.)

# 4. Tapasztalatok és visszajelzések

## 4.1. Gyakorlatvezetői tapasztalatok

Ebben a fejezetben részletesebben beszámolunk 85 hallgatóval kapcsolatos tapasztalatokról, akik négy különböző csoportban, ugyanazon oktató vezetésével végezték el a tárgyat. A 85 hallgató az előirányzott 60-nál jóval nagyobb létszámot jelentett, így a csoportlétszám meghaladta a tervezett 15 főt csoportonként. Erre azt a megoldást találtuk, hogy megnöveltük a csapatok létszámát és a követelményeket is. Így ezekben a csoportokban túlnyomórészt négy fős csapatok dolgoztak. A csapatokat a hallgatók szabadon alakították ki.

A feladat egy, a Command and Conquer [9] játékhoz hasonló valós idejű stratégiai játék megtervezése és implementálása volt. Az első órán egy váratlan probléma merült fel: a hallgatók egy része egyáltalán nem játszott stratégiai játékokkal ezelőtt, nem tudták elképzelni a feladatot. Szerencsére, mivel a játék nagyon ismert, a játékmenetet be tudtuk videókon mutatni, illetve létezik egy nyílt forráskódú implementáció is [10] így a hallgatók ki is tudtak próbálni egy hasonló játékot. Ötleteket is meríthettek ebből az implementációból, bár ez nem volt jellemző: ezt az ipari szintű megoldást sok éve fejleszti több, mint 200 fejlesztő, így a bonyolultsága jelentősen meghaladja egy féléves hallgatói projekt szintjét.

A hallgatók szabadon választhattak fejlesztési eszközöket, és szabadon használhattak kész játékmotorokat is, programozási nyelv megkötés nélkül. Három féle fejlesztési környezet alakult ki: Java és Swing, játékmotor nélkül (ez követte az előadás tematikáját), Java libGDX játékmotorral, illetve C# Unity játékmotorral.

Meglepetést okozott, hogy a Unity játékmotort a hallgatók negyede választotta, habár a C# nyelvet csak később tanulják az egyetemi képzésben. Erre egy magyarázat, hogy a Unity az iparban magasan a legelterjedtebb megoldás a három közül, így ennek a tudásnak a megszerzését találták a leghasznosabbnak, illetve hobbiprojektekben már sokan találkoztak a C# nyelvvel. Másfelől a középiskolában is sokan találkoztak már a C#-pal. Az egyik aggodalmunk, ami felmerült a félév elején, hogy a Unity-s csapatoknak könnyebb dolga lesz, mert szinte mindent megold majd a játékmotor helyettük. Meglepő módon ennek pont a fordítottja lett igaz: bár a Unity-s projektek színvonala általában meghaladta a többi projektét, egyik csapatnak sem sikerült teljesen befejeznie a játékot. Valószínűleg a szerteágazó játékmotor széleskörű megismerése és az ahhoz való alkalmazkodás okozta a nehézséget.

A legtöbb csapat magáénak érezte a feladatot. Sokan nem csak egyetemi kurzusként, hanem saját hobbiprojektként is gondoltak a projektre, így jelentősen meghaladták a kurzus követelményeit. Az egyik csapat egy teljes 3d-s játékmotort fejlesztett a játékukhoz, aminek a tematikája a Charlie és a csokigyár volt. A grafikát is saját maguk készítették, és érdekes ötletekkel töltötték meg a játékot. Az egyik ilyen ötlet a ninja karakter volt, aki helyett hálózati játékban a másik játékos egy fát lát, és így sokkal nehezebben veszi észre.

Voltak olyan jó képességű csapatok is, akik nagyon magas színvonalon kezdték el a projektet, viszont a konzisztens csapatmunka kihívást jelentett számukra, így nem sikerült teljesen befejezniük. Ezen csapatok többsége a Unity motorral dolgozott. Nagyon gondosan válogatták össze az asseteket és kezdték el a munkát, többségében teljesen 3d-s RTS-t fejlesztettek, de egy-két funkció megvalósítása a Unity környezetben gondot okozott. A legjobb példa erre a hálózati játék: ez egyetlen Unity-s csapatnak sem sikerült.

A személyes hallgatói visszajelzések többsége pozitív volt. Sokan mondták, hogy ez volt eddig a leghasznosabb tárgyuk a képzés során. Volt, aki egy cégnél interjúzott, és nagyon pozitívan fogadták, hogy már az egyetemi képzés során is csapatban old meg egy verziókezelt, nagyobb projektet.

Persze problémák is akadtak. A félév folyamán több esetben is akadt olyan csapattag, aki nem tudott kódolási feladaton dolgozni – ez a jegyszerzés követelménye volt. Ennek különböző okai

voltak. Egyrészt voltak olyan hallgatók, akik minél kevesebb munkával szerettek volna jegyet szerezni, és úgy gondolták, hogy a többiek meg fogják oldani helyettük a feladatot. Szerencsére ezeknél a hallgatóknál sikerült időben közbelépni: a verziókezelés segítségével könnyen visszakövethető volt, hogy ki min dolgozott, és amint ez a visszakövethetőség a hallgatók számára is világossá vált, jobban bekapcsolódtak a munkába. Például volt egy négyfős csapat, ahol két fő nagyon szépen dolgozott, és ők ketten majdnem befejezték a teljes játékot. Itt azt a megoldást választottuk, hogy ők ketten megkapták még félév közben az ötöst a tárgyra, a másik két főnek pedig onnan kellett befejeznie, illetve plusz funkciókat kellett beletenniük.

Egy nehezebben kezelhető eset volt, amikor a hallgatók, habár nagyon lelkesek voltak, nem voltak meg a megfelelő programozási alapjaik. Például volt egy csapat, ahol egy hallgató nagyon jó programozói tudással rendelkezett, a többieknek viszont alapvető hiányosságaik voltak. Emiatt mind a rosszul teljesítő hallgatóknak, mind a jól teljesítő hallgatónak lelkiismeretfurdalásuk volt: a hiányosságokkal rendelkező hallgatóknak azért, mert nem nagyon tudtak a projekthez hozzátenni, a jól teljesítő hallgatónak pedig azért, mert úgy érezte, hogy rajta múlik az egész projekt, és ő felel a többiek jegyéért is. Ezekben az esetekben azt a megoldást találtuk, hogy a játék egy részét is elfogadtuk kész projektként, mindenkinek találtunk olyan feladatot, amit ő is el tud végezni, és a jegyek elosztásában figyelembe vettük, hogy kinek mekkora része volt a játék elkészítésében.

Egy harmadik eset, amikor valaki szeretne, és részt is tudna venni a csapat munkájában, viszont a csapatban kialakult viszonyrendszer ezt megakadályozza. Ez az eset csak egy esetben fordult elő, viszont sok stresszt okozott az elszenvedőjének. A projekt kezdetén ő vette ki a legjobban a részét a munkából, ő készítette a követelményleírás nagy részét. Amikor a kódolásra került a sor, akkor viszont a többiek hárman kezdtek el összedolgozni, és ő kimaradt, nem kapott feladatot. Ez nem volt szándékos a többiek részéről, viszont a hallgatónak rosszul esett, illetve amiatt is aggódni kezdett, hogy nem lesz meg a jegye. Miután biztosítottuk arról, hogy lesz jegye, illetve kapott kódolási feladatot, a helyzet megoldódott.

## 4.2. A hallgatók visszajelzései saját kérdőívek nyomán

A félév két köztes pontján, egy-egy munkaszakasz lezárta után a hallgatók futam visszatekintéshez hasonló kérdőíveket töltöttek ki az elmúlt munkaszakaszra vonatkozóan. A kérdőíveket egyénileg kellett kitölteniük, majd javasolt volt, hogy megbeszéljék csapattársaikkal az arra adott válaszokat. Az évfolyam eredményeinek összesített és elemzett változata egy honlapra került ki, ahol a gyakorlatvezetők és csoporttagok egyaránt követhették, és szembesülhettek az átlaghoz képesti helyzetükkel. Cikkünk olvasói számára is rendelkezésre állnak ezek a részletek (people.inf.elte.hu/ilyese/csapatmentor.html).

A számértékeket illetően ebben a cikkben az átlagokat és azok módosulását emeljük ki a 3. táblázatban.

| Értékeld a csapat teljesítményét a futam kapcsán!<br>Az értékelés során használd az 1..5 skálát: 1 – egyáltalán nem jó, 5 – nagyon jó. | | | |
|---|---|---|---|
| **Kérdések** | **1.felmérés** | **2.felmérés** | **Változás** |
| Milyen volt a közérzeted a csapatban? | 4.62 | 4.33 | **-0.29** |
| Mennyire működött csapatként a csoport? | 4.39 | 4.33 | **-0.06** |
| Értékeld a csapat teljesítményét! | 4.47 | 4.33 | **-0.14** |
| Értékeld a csapat kommunikációját! | 4.29 | 4.15 | **-0.14** |
| Értékeld a csapat lelkesedését! | 4.08 | 3.53 | **-0.55** |

**3. táblázat:** Csapatmunka értékelése a félév közben

Megítélésünk szerint az átlag számértékek viszonylag magasak voltak mindkét felmérés során. Az első futam során a hallgatók többnyire teljesen jól érezték magukat a csoportmunkában (közérzetek átlaga: 4.62). A csapat teljesítményét is egész magasra értékelték (4.47). A csapat kommunikációjának megítélése hozta a legnagyobb szórást. Érdekes az, hogy bár közérzetük többnyire magas volt, a csapatok lelkesedését 0.54-gyel alacsonyabbra értékelték. Látható, hogy a második futam után minden átlag csökkent valamennyit, de legtöbb 0.55-öt. Ugyanakkor figyelmeztető jel lehet, hogy a kommunikáció, a csapatmunka működése nem fejlődött a közös munka során, pedig elméletileg már jobban ismerik egymást a csapattagok, jobban „összeszokta". A lelkesedés csökkent a legtöbbet. A kisebb értékek viszont annak is betudhatók, hogy a második értékelésnél kifinomultabban értékelnek már a hallgatók, esetleg a kezdeti „rózsaszín felhők lassan elvonultak" és a hallgatók reálisabban látták helyzetüket.

A két félévközi felmérés során megkérdeztük a hallgatókat, hogy mit látnak csapatunk erősségének, gyengeségének, és milyen lehetséges javító lépéseket tudnak megfogalmazni csapatuk számára.

Az első felmérés kapcsán a kommunikáció kérdésköre az erősségek és a gyengeségek esetén is hangsúlyosan visszatért. Ez rámutat arra, hogy milyen lényeges faktor a kommunikáció a csapatmunka során. Hasonlóan, a munkamegosztás és az időbeosztás is gyakran emlegetett faktor volt: a sikeresség kapcsán inkább a munka megfelelő beosztását, a sikertelenség kapcsán az idő nem megfelelő beosztását jegyezték fel a hallgatók. A korábbi hallgató-társi kapcsolataik mentén alakítani ki a csoportokat erősségként lett megjelölve, ahogyan az is, hogy közösen – egy fizikai térben dolgoztak együtt a csapattagok. A gyengeségek kapcsán a motiváció hiány is fellépett. Viszont volt több olyan hallgató, aki úgy élte meg, hogy a csapatnak nem is volt gyengesége, „minden flottul ment". A javító lépés ötletek szépen összecsengtek az erősségek-gyengeségek kapcsán kihangsúlyozott faktorokkal. Legtöbben az időbeosztás optimalizálását fogalmazzák meg lehetséges javító lépésként, a közös fizikai térben történő munkát és a kommunikációt. Vannak, akik felvetik, hogy jobban kéne ösztönözniük egymást a munkára. Akik elégedettek a csapatmunkájukkal, azok többnyire nem is fogalmaznak meg lehetséges javító lépést.

A második felmér esetében a csapat erősségek kapcsán a kommunikáció megjelölése alábbhagyott, a munkamegosztás és csapatmunka nagyobb hangsúlyt kapott. Örvendetes, hogy a csapatmunka, mint erősség megjelent az első visszatekinéshez képest – jelentheti ez azt, hogy a hallgatók jobban érzékelték már, hogy valóban egy csapat tagjai. A gyorsaság, mint erősség is feltűnt. A gyengeségeket illetően még mindig viszonylag nagy százalék van, akik nem érzékelnek különösebb gyengeséget. A többség a halogatást érzékeli, ami viszont az időbeosztással is szorosan összefügg. Már az első felmérés során is láthattuk, hogy az időbeosztás kérdése legalább a hallgatók negyedét érintette, a második felmérés esetében úgy tűnt, hogy még többet. Megjelent egy újabb tényező: „a feladat megértése". Oktatóként erre fokozottan fel kellett figyelnünk: a feladatok még tisztább meghatáro-

zása segíthet csapatainknak. A javító-lépés ötletek kategóriában kevesebb témájú visszajelzés született. A kommunikáció nagyobb arányban jelent meg, mint korábban (jelentheti ez azt, hogy a hallgatók egyre inkább felismerik, hogy a kommunikációra fektetett figyelem erős javulást hozhat a csapatmunkát illetően). Itt is megjelenik (a gyengeségekhez hasonlóan) az új szó: halogatás – úgy tűnik, hogy sokak esetében aktuális probléma ez. A hallgatók 26%-ának nincs ötlete javító lépésre – ezek többsége abban az összefüggésben jegyzi ezt meg, hogy gyengeséget sem vélt fölfedezni a csapatban.

A félév végén is visszatekintés jellegű kérdőíveket töltött ki 118 hallgató, ekkor már viszont a teljes félévre vonatkozóan. Az eredmények egy részét a 4. táblázat foglalja össze:

| Értékeld a csapat teljesítményét a futam kapcsán!<br>Az értékelés során használd az 1..5 skálát: 1 – egyáltalán nem jó, 5 – nagyon jó. | | | | | | |
|---|---|---|---|---|---|---|
| **Kérdések** | **1** | **2** | **3** | **4** | **5** | **Átlag** |
| Milyen volt a közérzeted a csapatban a félév során? | 1.7% | 5.1% | 16.1% | 20.3% | 56.8% | **4.25** |
| Mennyire működött csapatként a csoport a félév során? | 5.1% | 12.7% | 17.8% | 26.3% | 38.1% | **3.79** |
| Értékeld a csapat teljesítményét a félévre vonatkozóan! | 2.5% | 9.3% | 14.4% | 34.7% | 39% | **3.98** |
| Értékeld a csapat kommunikációját a félévre vonatkozóan! | 5.1% | 11.9% | 23.7% | 17.8% | 41.5% | **3.78** |
| Értékeld a csapat lelkesedését a félévre vonatkozóan! | 7.6% | 12.7% | 16.9% | 30.5% | 32.2% | **3.66** |

**4. táblázat:** Csapatmunka értékelése a félév végén

Az átlagokat elég magasnak értékeljük. Feltűnő, hogy a „közérzet a csapatban" a legnagyobb érték – ez alátámasztja a gyakorlatvezetőknek azt a megfigyelését, hogy a hallgatók élvezték a csapatmunkát. A legkisebb érték a csapat lelkesedése, bár ez sem kritikusan alacsony. Rámutat viszont arra, hogy bár a hallgatók kellemesen érzik magukat a csapatmunkában, a lelkesedésük, motivációjuk némileg visszafogottabb. Érdekes megfigyelés lehet, hogy a „Mennyire működött csapatként a csoport?" és az „Értékeld a csapat kommunikációját!" típusú kérdésekre mennyire hasonló az átlag és szórás is – ez utalhat arra, hogy a „csapatként működés" érzete mennyire összefügg a „működik köztünk a kommunikáció" érzetével. Míg minden más érték esetében a szavazatok százaléka egyre nő a válaszokra adott számértékek növekedésével (vagyis a „jóság" mértékével), a kommunikáció esetében ez megtörik: a közepes 3-as érték és a maximum 5-ös érték emelkedik ki a többi érték közül. Ez alapján úgy tűnik, hogy a kommunikációt leginkább vagy teljesen megfelelőnek, vagy közepesen jónak érzékelik a hallgatók, nehezebb egy „jó" szintet elérni vagy érzékelni. Utalhat ez arra is, hogy a kommunikáció jóságának megítélésében nehezebb számukra árnyaltabban gondolkodni. A csapat lelkesedésének, motivációjának értékelésében láthatjuk a legnagyobb szórást.

Arra is kíváncsiak voltunk, hogy a hallgatók mennyire értékelték hasznosnak a tárgyat a csapatmunkával kapcsolatos tapasztalatok szempontjából. Az átlagérték ebben az esetben is magas lett:

3.96, ahol az 5-ös érték a maximum. A megkérdezettek közel fele 5-ös értéket adott. Ez jó visszaigazolás abból a szempontból, hogy a tárgy azon célja, hogy a hallgatóknak hasznos tapasztalatot nyújtson a csoportműködést illetően megfelelően teret kapott. Az 5.táblázat részletesen mutatja be a felmérés eredményét.

| Kérdések | 1 | 2 | 3 | 4 | 5 | Átlag |
|---|---|---|---|---|---|---|
| Mennyire tudott hasznos tapasztalatot adni neked ez a tárgy a csapatmunka működését illetően? | 4.2% | 5.1% | 16.1% | 35.6% | 39% | **3.96** |

**5. táblázat:** Scrum szerepkörök érvényesülése a gyakorlatok során

Azon kérdések esetében, ahol néhány szóban adhattak visszajelzést a hallgatók a csoportmunkára vonatkozóan, kiemelünk néhány választ:

1. „A mi csapatmunkánk erőssége volt ...”
   - „A jó kommunikáció.”
   - „Jól osztottuk szét a feladatokat.”
   - „Ha a csapat egyik tagja elakadt, mindig kapott segítséget a másik csapattagtól.”
   - „A precizitás, a lelkesedés a feladat iránt.”
   - „Hogy le tudtunk ülni megbeszélni problémákat.”
   - „Hogy ismertük egymást.”
   - „Jó időbeosztás, jó hangulat.”
2. „A mi csapatunk gyengesége volt …”
   - „Élő kommunikáció hiánya”.
   - „Mindenki elfoglalt, kevés volt az időnk.”
   - „Több mindent a határidők végére hagytunk és nem számoltunk azzal, hogy ha másnak is dolgoznia kell az eredménnyel.”
3. „Legszívesebben másként tenném így utólag azt, hogy ...”
   - „Több energiát forgatnék bele.”
   - „Keveset teszek a projekthez az utolsó szakaszban”.
   - „A lehető legkorábban összeülnék a csapattal, hogy bőven a bemutatás előtt végezzünk a munkával.”
   - „A többi csapattagot motiváltabbá tegyem.”
   - „Gyakrabban használnám a kommunikációs csatornákat.”
   - „Határozottabban mondanám el a vélemenyem bizonyos helyzetekben.”

Láthatjuk azt, hogy a hallgatók visszajelzéseik alapján fölismerték azokat az értékeket, melyek a szakirodalom szerint is nagymértékben hozzájárulnak csapatok megfelelő működéséhez: jól struktúráltság („jó időbeosztás”, „Jól osztottuk szét a feladatokat.”), motiváció, konfliktusok és nehézségek fölvállalása, illetve a pszichológiai biztonság jelenléte a csapatban. Ezt a fogalmaz ugyan bizonyára nem ismerték a hallgatók ahhoz, hogy így nevezzék meg visszajelzéseik során, de a „jó kommuniká-

ció", „mindig kapott segítséget", „le tudtunk ülni megbeszélni problémákat", „ismertük egymást" kifejezések erősen utalhatnak erre.

A gyengeségek kategóriájában a hallgatók leginkább a jólszervezettség és a kommunikáció hiányát ismerik fel. Könnyen lehet, hogy a gyengeségek szintjén nem ismerik fel – vagy nem tudják megfogalmazni, hogy a pszichológiai biztonság hiánya mennyire megnehezíti a hatékony csoportműködést: ha nem tudják fölvállalni saját véleményüket, konfliktusaikat, önmaguk igényeit.

Amikor arról kérdeztük a hallgatókat, hogy mit tennének utólag másként, sok olyan lehetőséget fogalmaztak meg, melyek valóban hozzájárulnak a hatékonyabb csapatmunkához. Ebből is látszik, hogy amennyiben tudatosítást próbálunk előidézni a hallgatókban a csoportmunkát illetően, sok felismerés felszínre tör. Az, hogy önvizsgálatra hívtuk meg őket ezt a témakört illetően reményeink szerint hozzájárult ahhoz, hogy a további csapatmunkáik során csapataik még értékesebb tagjai legyenek.

Arról is megkérdeztük a hallgatókat, hogy a teljes tárgyra vonatkozóan mi volt a leghasznosabb és legpozitívabb tapasztalat számunkra. A csapatmunka mindkét kérdés kapcsán többször előkerült a válaszokban. A következőkben kiemelünk néhány visszajelzést:

4. „A leghasznosabb élmény számomra az volt, hogy …"
    - „Végre volt lehetőség csapatban dolgozni, így megtanulni egy verziókezelő használatát."
    - „Megtanultam milyen is egy csapatot összetartani, motiválni, koordinálni."
    - „Megtanultam, hogy a csapat akkor se működik jól, ha valaki az elején túl sok energiát fektet a munkába, hiszen a többiek így inkább hátradőlnek. Sajnálatos tapasztalat volt."
    - „Létrehoztam egy félkész játékot, kicsit giteztem, emberi természetet jobban megismertem."
    - „Rengeteg újdonságot tanulhattam a többiektől a fejlesztés során."
5. „A legpozitívabb élmény számomra az volt, hogy …"
    - „A munkámmal kapcsolatban pozitív visszajelzéseket kaptam."
    - „Végre volt lehetőség egy csapatban dolgozni."
    - „Csapatban sikerült valami működő egészet csinálni."
    - „Hogy a specifikáció/tervezés végre nem volt abszolút felesleges."
    - „látványos eredményt adott"
    - „Kiderült, hogy jól tudok csapatban dolgozni."
    - „A csapatban mindenkinek összeadódnak az erősségei."

## 4.3. A hallgatók visszajelzései az OMHV alapján

Az Oktatói Munka Hallgatói Véleményezése (OMHV) azért érdekes, mert ezt a hallgatók minden félév során kitöltik minden tárgy kapcsán így lehetőség nyílik a változások megismerésére az eredeti és a megváltoztatott tárgy értékelésének összevetésével.

Három kérdést találtunk érdekesnek:

1. Mennyire szolgált releváns és új tudással a kurzus?

2. Mennyire segítette a kurzus a szakmai fejlődését?
3. Mennyire tartotta interaktívan a foglalkozásokat az oktató?

A kérdéseket a hallgatók az előadás és a gyakorlat szempontjából külön válaszolják meg. Az értékelés lehet:

- Teljes mértékben
- Többnyire
- Kevéssé
- Egyáltalán nem
- Nem válaszolok

Az előadáshoz kapcsolódó válaszokat az 6-8. táblázatok tartalmazzák. Mivel a két félévben különböző számú hallgató töltötte ki a kérdőíveket, a válaszokat az összes válaszoló hallgató százalékában adjuk meg.

| **Válaszlehetőség** | **2018** | **2019** | **Változás** |
|---|---|---|---|
| Teljes mértékben | 51.61% | 55.26% | 3.65% |
| Többnyire | 29.03% | 28.95% | -0.08% |
| Kevéssé | 9.68% | 5.26% | -4.41% |
| Egyáltalán nem | 6.45% | 3.95% | -2.50% |
| Nem válaszolok | 3.23% | 6.58% | 3.35% |

**6. táblázat:** Válaszok a „Mennyire szolgált releváns és új tudással a kurzus?" kérdésre az előadáshoz (teljes kurzushoz) kapcsolódóan

| **Válaszlehetőség** | **2018** | **2019** | **Változás** |
|---|---|---|---|
| Teljes mértékben | 48.39% | 63.16% | 14.77% |
| Többnyire | 32.26% | 22.37% | -9.89% |
| Kevéssé | 6.45% | 2.63% | -3.82% |
| Egyáltalán nem | 6.45% | 3.95% | -2.50% |
| Nem válaszolok | 6.45% | 7.89% | 1.44% |

**7. táblázat:** Válaszok a „Mennyire segítette a kurzus a szakmai fejlődését?" kérdésre az előadáshoz (teljes kurzushoz) kapcsolódóan

| Válaszlehetőség | 2018 | 2019 | Változás |
|---|---|---|---|
| Teljes mértékben | 35.48% | 53.95% | 18.46% |
| Többnyire | 25.81% | 23.68% | -2.12% |
| Kevéssé | 9.68% | 9.21% | -0.47% |
| Egyáltalán nem | 6.45% | 2.63% | -3.82% |
| Nem válaszolok | 22.58% | 10.53% | -12.05% |

**8. táblázat:** Válaszok a „Mennyire tartotta interaktívan a foglalkozásokat az oktató?" kérdésre az előadáshoz (teljes kurzushoz) kapcsolódóan

A válaszokból kitűnik, hogy a válaszoló hallgatók nagy része összességében az előző évben is elégedett volt a kurzussal: 80 százalékuk úgy vélte, hogy segítette a kurzus a szakmai fejlődésüket. Az új kurzussal ehhez képest sikerült számottevő javulást elérni. Különösen nagy, 15 illetve 18 százalékos javulás látható a 2. és a 3. kérdésnél a „teljes mértékben" válaszoknál.

A gyakorlatok értékelése hasonló eredményt mutat (9-11. táblázatok). Ami némileg meglepő, hogy a 2. kérdésre adott válaszoknál a „teljes mértékben" növekedése nem ellensúlyozza a „többnyire" csökkenését, így összességében kis visszaesés tapasztalható. Ez ellentétes az előadásra – vagy teljes kurzusra – adott pontszám alakulásával. Azonban ha megnézzük a 2018-as adatokat feltűnik, hogy eleve 88%-ról indult azon válaszolók száma, akik elégedettek voltak a gyakorlattal, tehát nem a 2019-es adat az alacsony, hanem a 2018-as a magas. Ezenfelül az első és a második kérdésre adott válaszok 2019-es százalékai teljesen hasonlóak. Összefoglalva, a csökkenés a magas 2018-as értékkel magyarázható.

| Válaszlehetőség | 2018 | 2019 | Változás |
|---|---|---|---|
| Teljes mértékben | 57.14% | 61.90% | 4.76% |
| Többnyire | 24.49% | 20.00% | -4.49% |
| Kevéssé | 14.29% | 5.71% | -8.57% |
| Egyáltalán nem | 4.08% | 8.57% | 4.49% |
| Nem válaszolok | 0.00% | 3.81% | 3.81% |

**9. táblázat:** Válaszok a „Mennyire szolgált releváns és új tudással a kurzus?" kérdésre a gyakorlathoz kapcsolódóan

| Válaszlehetőség | 2018 | 2019 | Változás |
|---|---|---|---|
| Teljes mértékben | 59.18% | 61.90% | 2.72% |
| Többnyire | 28.57% | 20.00% | -8.57% |
| Kevéssé | 6.12% | 9.52% | 3.40% |
| Egyáltalán nem | 4.08% | 4.76% | 0.68% |
| Nem válaszolok | 2.04% | 3.81% | 1.77% |

**10. táblázat:** Válaszok a „Mennyire segítette a kurzus a szakmai fejlődését?" kérdésre a gyakorlathoz kapcsolódóan

| Válaszlehetőség | 2018 | 2019 | Változás |
|---|---|---|---|
| Teljes mértékben | 46.94% | 64.76% | 17.82% |
| Többnyire | 28.57% | 23.81% | -4.76% |
| Kevéssé | 8.16% | 3.81% | -4.35% |
| Egyáltalán nem | 2.04% | 1.90% | -0.14% |
| Nem válaszolok | 14.29% | 5.71% | -8.57% |

**11. táblázat:** Válaszok a „Mennyire tartotta interaktívan a foglalkozásokat az oktató?" kérdésre a gyakorlathoz kapcsolódóan

## 5. Összefoglalás és további terveink

A tárgy átalakítása közben legfőbb célként az lebegett a szemünk előtt, hogy az ipari szoftverfejlesztéshez hasonló élményt tudjunk adni a hallgatóknak egyetemi keretek között, valamint a hallgatók éljék meg az alkotás örömét, lehetőség szerint a jegyszerzésen túl is érdekeltek legyenek a projektmunkában. A félév során folyamatosan gyűjtöttük a visszajelzéseket, hogy képet kapjunk arról, hogy az elképzeléseinkhez képest mindez hogyan alakult. Az ebben a cikkben részletezett eredményekből azt a képet kaptuk, hogy jó úton haladunk a cél felé, ami pedig a legfontosabb, a hallgatók nagy része kellemes élményekkel gazdagodott és sokat tanult a félév során. Persze még korántsem értünk célba, a kurzust folyamatosan fejleszteni fogjuk az elkövetkező években.

Ennek első lépéseként a félév végén összesítettük a tapasztalatokat és változásokat vezettünk be, amiket már a tárgy következő félévében alkalmazni fogunk. Az egyik legfontosabb észrevétel az volt, hogy a projekt terjedelme túl nagyra sikerült. Teljesen, mindennel együtt nagyon kevés, csak néhány csapat tudta befejezni a projektet. A legtöbben a folyamatos integrációt hagyták el, illetve a hálózati játék sem sikerült a csapatok többségének, emiatt végül nem is volt szükséges a jeles eléréséhez. A hálózati játék még egy problémát felvetett: mivel a hálózatkezelést a hallgatók csak a 8. előadáson

ismerték meg, nehezen tudták megtervezni a munkát és a kezdeti architektúrát – sok csapatnál a hálózat újratervezést igényelt. A tapasztalatokból kiindulva a jövő félévben a hálózati játék nem lesz követelmény, így marad idő a folyamatos integrációra, illetve jobban tervezhető lesz a projekt.

A tervezhetőséggel kapcsolatban felmerült még egy probléma: a hallgatóknak túl sok volt a követelményleírásra és megoldási tervre biztosított 6 hét, a maradék 6 hét pedig túl kevés az implementációra. Habár a gyakorlatvezetők hangsúlyozták a prototípusok fontosságát az első gyakorlattól kezdve, sok csapat csak a 7. hét után kezdte meg a kódolást. Erre azt a megoldást találtuk, hogy a két tervezési mérföldkövet összevontuk és előrehoztuk. Így legközelebb három mérföldkő lesz a félév során: egy tervezési és két implementációs.

Ez egy másik problémát is megoldhat, mégpedig azt, hogy sok csapatnál az implementáció nagy része az utolsó hétre készült el. Habár adtunk a vizsgaidőszakban plusz időpontot a projekt bemutatására, több esetben így is le kellett vonni a végső jegyből a fontos hiányzó funkciók miatt (pl. egységek támadása). Az új tervben az implementáció nagy részét már az első implementációs mérföldkőnél számon fogjuk kérni, így lesz idő a korrekcióra.

Azt reméljük, hogy ezekkel a változtatásokkal sikerül még hasznosabbá és élvezetesebbé tenni a kurzust a hallgatók számára. Egyrészt így több időt tudnak a funkciók implementálásával tölteni és az implementáció jobb visszacsatolást nyújthat a tervezésre. Másrészt, mivel előbb kezdik az implementációt, el tudjuk kerülni azt, hogy a projektmunka hajrája és az egyéb tárgyak zárthelyi dolgozataira való készülés ütközzön a szorgalmi időszak utolsó hetében.

Összességében úgy gondoljuk, hogy az ipari projektmunka szimulációja egyetemi környezetben sikeresnek bizonyult. A hallgatók olyan, ipari sztenderdnek számító eszközöket használhattak csoportmunkában, amikkel azelőtt vagy egyáltalán nem, vagy csak külön tárgy keretében találkozhattak. A tárgyunk legfőbb értékének azt tartjuk, hogy a fontos elemeket – mint az agilis csoportmunka, követelményleírás, tervezés, verziókezelés, folyamatos integráció – egy tárgyban sikerült összefognunk egy gyakorlati projektmunka keretében. Így ezek a különálló elemek összeállhattak egy szerves egésszé, ami talán ablakot nyitott a hallgatóknak az ipari szoftverfejlesztés világába. Persze rengeteg tennivalónk lesz még, hiszen az informatika folyamatosan változik. Remélhetőleg mi is képesek leszünk változni vele.

## Irodalom


1. J.G. Kuhl: *Incorporation of Agile Development Methodology into a Capstone Software Engineering Project Course*. In: University of Iowa - Iowa Research Online (ed.): 2014 ASEE North Midwest Section Conference, Iowa city (2014)
2. 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)
3. C. Anslow, F. Maurer: *An Experience Report at Teaching a Group Based Agile Software Development Project Course*. In: ACM (ed.): SIGCSE'15, Kansas (2015)
4. Y. D. Orit Hazzan: *Why Software Engineering Programs Should Teach Agile Software Development*, In: ACM (ed.): ACM SIGSOFT Software Engineering Notes, 2007
5. Gitlab: http://gitlab.com (utoljára megtekintve: 2019. 10. 31.)
6. Agile Manifesto: http://agilemanifesto.org/ (utoljára megtekintve: 2019. 10. 31.)
7. Slack: http://slack.com (utoljára megtekintve: 2019. 10. 31.)
8. Google forms: https://www.google.com/forms/about/ (utoljára megtekintve: 2019. 10. 31.)
9. Command and Conquer Studios, Westwood. "Command and conquer." Virgin Interactive: United Kingdom (1995)
10. OpenRA: http://www.openra.net/ (utoljára megtekintve: 2019. 10. 31.)

# Úton a szakköralkalmazás felé a ProgCont API-val

Kádek Tamás[1], Biró Piroska[2]

{[1]kadek.tamas, [2]biro.piroska}@inf.unideb.hu
DE IK

**Absztrakt.** A ProgCont automatikus megoldáskiértékelő rendszert a Debreceni Egyetem Informatikai Karán már számos különböző számonkérésen, helyi, országos és nemzetközi versenyen alkalmaztuk sikeresen. A pozitív tapasztalatainkból kiindulva úgy gondoljuk, hogy a rendszert érdemes szélesebb körben elérhetővé tenni. Ez adta az ötletét az általunk újonnan kifejlesztett ProgCont API szolgáltatásnak, melyet elsősorban a középiskolában tanító pedagógusok számára kívánunk felajánlani. Az API felhasználásának első kísérleteként egy szakköröket támogató alkalmazás fejlesztése mellett döntöttünk. Cikkünkben az API-ban és a segítségével készülő alkalmazásban lévő lehetőségeket szeretnénk bemutatni.

**Kulcsszavak:** ProgCont rendszer, automatizált kiértékelés, programozás, szakköralkalmazás

## 1. Bevezetés

A ProgCont rendszer, mely egy automatizált megoldáskiértékelő rendszer, 2011 óta használjuk és fejlesztjük a Debreceni Egyetem Informatikai Karán [1], [2], [3], elsősorban programozási feladatok kiértékelésére, számonkérések és programozási versenyek lebonyolítására, támogatására készült.

A legújabb fejlesztési irányunk a ProgCont API fejlesztése és az ehhez kapcsolódó szakköralkalmazáshoz tartozó kliens program megtervezése és implementálása. Ezt a fejlesztési irányt a GDPR megjelenése és a GDPR által támasztott követelményeknek való megfelelési kényszer szülte. A megfelelés leggyorsabb módja az volt, hogy egyszerűen eltávolítottunk a szoftverből minden személyes adatot, melynek eredményeképpen bejelentkezés nélkül vált használhatóvá a rendszer. Előtte csak a Debreceni Egyetem Informatikai Karának hallgatói tudták használni hálózati azonosítóval, és az általunk szervezett verseny résztvevői a számukra létrehozott felhasználói fiókkal. Ezután a korábban csupán egy zárt közösség számára elérhető alkalmazásunk nyitottá vált. Azóta előállt a helyi GDPR-ral kompatibilis szabályozásnak megfelelő verzió is, de szoftver regisztráció nélküli használható változatát a pozitív visszajelzések miatt megtartottuk.

Jelen pillanatban a ProgCont rendszert bárki szabadon használhatja az elérhető feladatok megoldásainak kiértékelésére. Az, hogy a rendszerünk regisztráció nélkül kiértékeli a korábban karunkon összeállított programozási feladatok megoldásait, még nem feltétlenül teszi csábítóvá a középiskolák számára. Annak érdekében, hogy érdembe bevonjuk a rendszer használatába a középiskolákban tanító tanárokat is, kialakítunk a ProgCont rendszerből egy, kifejezetten a szakkörök szervezését támogató alkalmazást is.

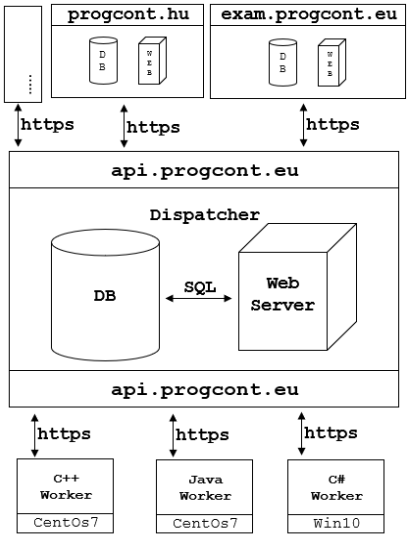
A ProgCont rendszert számos feladat megoldásra használjuk, ezek a feladatok nem jelentenek egyenletes terhelést a háttérben meglévő infrastruktúra számára. Például a nagy érdeklődést kiváltó programozási versenyek, illetve a különböző programozási tárgyak számonkérésének ideje alatt mutatkozó terhelés mellett eltörpül az, ami hétköznapi használat során tapasztalható. A ritkán jelentkező, de intenzív terhelést jelentő időszakok közt más is hozzáférhetne és használhatná a rendelkezésre álló erőforrásokat akár saját szakkörök, számonkérések szervezése céljából. Köztudott, hogy a középiskolák infrastruktúrája változó, ezért számukra fejezetten hasznos lehet a ProgCont szabad kapacitásainak kihasználása. Célunk, hogy támogassuk a szakkör szervezésének lehetőségét azáltal, hogy biztosítjuk az alkalmazás használatát a középiskolában tanító kollegák számára.

# 2. ProgCont API

A fejlesztés alapvető eleme a ProgCont API. Ezzel létrehozunk egy alkalmazói interfészt, amelyen keresztül az automatikus megoldáskiértékelő rendszert hozzáférhetővé tettük. Ezután következhetett a meglévő felhasználói interfészek (webalkalmazások) átalakítása úgy, hogy az új API szolgáltatásain keresztül lássák el faladatukat. Ennek segítségével még házon belül tesztelhettük az alkalmazói interfészt.

## 2.1. A ProgCont API felépítése

Az API egy online elérhető eszközt definiál, amely a programkódok automatikus fordításáért, futtatásáért és a kiértékeléséért felelős. Nem csupán a meglévő alkalmazásaink kaptak ezzel egységes háttérszolgáltatást, hanem megnyílt a lehetősége, hogy hasonló alapokon újabb funkciókkal bővüljön a ProgCont eszköztára, amelyek segítségével bármely érdeklődő – akár középiskolai informatika tanár – továbbfejleszthesse és saját elképzelései alapján testre szabhassa.




**1. ábra:** A ProgCont API rendszersémája [3]

A ProgCont API elérhető a következő link segítségével: https://api.progcont.eu/

Az API központi eleme a Dispatcher szolgáltatás, ez az a réteg, amellyel közvetlenül kapcsolatba kerül az, aki a ProgCont API-n fejlesztést akar végezni, ez tulajdonképpen a beérkező programkódok fordítás, futtatás feladatát elosztja azon worker-nek nevezett gépek között, amelyek képesek a beérkező programkódokat elemezni, fordítani, futtatni különböző programozási nyelveken (például: C, C++, C#, Java, Pascal, Python).

A rendszert modulárisra terveztük, az alábbi rétegeket megkülönböztetve egymástól:

- a programkódok fordításáért és futtatásáért felelős worker-ek,
- a feladatok kiosztását felügyelő Dispatcher alkalmazás (ennek funkciói jelennek meg a ProgCont API szolgáltatásaiként), és
- a felhasználói interfészek (webalkalmazások) a különféle igények kielégítésére.

Ha valakinek az általunk felajánlott kapacitás nem elegendő, akkor ezek a részek önállóan is telepíthetők saját infrastruktúrára. Természetesen ez sokkal több konfigurációs beállítást igényel, mint az általunk biztosított ProgCont API végpont – az `api.progcont.eu` – felhasználása.

## 2.2. A ProgCont API szolgáltatásai

A ProgCont API szolgáltatásai: a programozói feladatokhoz készült forráskódokat képes lefordítani különböző programozói nyelven (C, C++, C#, Java, Pascal, Python), ezen lefordított kódokat futtatja és vizsgálja a keletkezett kimenetet, a program kimeneti kódját, hiba kimenetét, standard outputját és a program futási idejét.

Az API központi eleme egy XML állomány, amely tartalmaz egy feldolgozási feladatot, a feldolgozási feladat pedig tartalmazhat néhány a feltöltött forráskódtól független részt, például hogyan akarunk értesülni a felhasználás eredményéről, vagy ha több felhasználó is használja a rendszert, akkor milyen prioritásba kerüljön a feldolgozás.

Az API két különböző XML sémát támogat:

1. az *egyszerűsített kiértékelést* leíró állomány egy CodeRunner[6] szerű [4] szolgáltatásokat leíró XML, amely tartalmazza a:
    a. forrásállományt, amelynek tesztelését el kívánjuk végezni,
    b. teszteseteket:
        i. a standard bemenettel,
        ii. a parancssori argumentumokkal,
        iii. a program futtatásakor elérhető állományokkal,
        iv. az elvárt kimenettel vagy kimenetekkel,
2. a *szakértői kiértékelést* leíró állomány, amely a fentiek mellett tartalmaz:
    a. fordítási opciókat,
    b. interpreter beállításokat,
    c. kimenetet ellenőrző programkódot,

amelyek közül a nyilvánosság számára az egyszerűsített változat az elérhető. Ez az, ami a workerek működésével kapcsolatos háttértudást – ilyenek lehetnek az operációs rendszer és a fordítóprogram verziófüggő sajátosságai – nem igényel.

A ProgCont API egyszerűsített kiértékelést leíró XML sémája nyilvános, a következő címen érhető el: `https://api.progcont.eu/docs/job.xsd`.

A séma részletes bemutatása helyett álljon itt egy példa egy kiértékelési feladat leírására (`job_in.xml`). Ebben egy egyszerű ANSI C program – amely egy darab `main.c` forrásállományban kapott helyet – kiértékelését kértük. Az állomány esetében épp úgy, ahogy később a standard bemenet és az elvárt kimenet leírásánál ASCII kódolású (`target-encoding="US-ASCII"`) szöveges állományokat használunk, ahol a sorok végét az operációs rendszernek megfelelő sorvégjelek zárják (`line-endings="platform-dependent"`). A fordítóprogramtól függően a worker kerülhet akár Linux-os vagy Windows-os gépre is, az ebből adódó különbségeket azonban kiküszöböltük. A kiértékelés eredményében a fordítóprogram kimenetét is fel szeretnénk tüntetni (`report-output="yes"`). Egyetlen teszteset segítségével ellenőrizzük a programot (`test-case-id="t1"`), amely esetében legfeljebb 5 másodperces futási időt engedélyeztünk (`time-limit="5000ms"`). A program akkor helyes, ha a megadott elvárt kimenetet produkálta.

[6] A CodeRunner a Moodle beépített plugin-ja, mely segítségével a diákok által különféle programozási feladatok megoldásaként készített programkódok futtathatók és értékelhetők [4].

Példa egy kiértékelési feladat leírására (`job_in.xml`):

```
<job-request job-id="j1" source-language="C">
   <source-code>
      <file file-name="main.c">
         <text target-encoding="US-ASCII" line-endings="platform-
         dependent">
               #include &lt;stdio.h>;
               int main(){
                   int szam, osszeg = 0;
                   while (scanf("%d", &amp;szam) > 0) {
                       osszeg += szam;
                   }
                   printf("%d", osszeg);
                   return 0;
               }
         </text>
      </file>
   </source-code>
   <compile-with report-output="yes"/>
   <test-cases>
      <test-case test-case-id="t1">
         <execute-with report-output="yes" time-limit="5000ms">
            <input>
               <text target-encoding="US-ASCII" line-endings=
               "platform-dependent">5 4 9 7 4</text>
            </input>
         </execute-with>
         <validate-with report-output="yes">
            <answer>
               <text target-encoding="US-ASCII" line-endings=
               "platform-dependent">29</text>
            </answer>
         </validate-with>
      </test-case>
   </test-cases>
</job-request>
```

Példa a kiértékelési feladat eredményére (`job_out.xml`):

```
<job-response job-id="j1">
   <compile exit-code="0"> <stdout/> <stderr/> </compile>
   <test-result test-case-id="t1">
      <execution exit-code="0" output-size="2" execution-time="6ms">
         <stdout> <text target-encoding="US-ASCII">29</text> </stdout>
         <stderr/>
      </execution>
      <validation verdict="PASS"/>
   </test-result>
</job-response>
```

A kiértékelés eredménye (`job_out.xml`) egyfelől tanúskodik arról, hogy a fordítás sikeresen lezajlott, és a program futása a megadott tesztesetre (`t1`) is sikeres volt (`exit-code="0"`). A futás eredménye a standard kimeneten 2 karakter (`output-size="2"`), amely 6 milliszekundum

alatt állt elő (`execution-time="6ms"`) úgy, hogy a hibakimenet üres volt. A kimenet megegyezett az elvárt kimenettel, így a program átment a teszten (`verdict="PASS"`).

# 3. Szakköralkalmazás

Az alkalmazás nem más, mint a ProgCont API segítségével fejlesztett informatika szakkörök lebonyolítását segítő webalkalmazás.

A fejlesztés szerepe kettős:

1. szeretnénk beemelni a középiskolákat a ProgCont rendszer felhasználóinak táborába, alternatívát biztosítva Moodle CodeRunner kiegészítője mellé [4], bízva abban, hogy a szűkebb célcsoport igényeit jobban ki tudjuk szolgálni;
2. forráskódját publikálva bemutatjuk, hogy miként lehet a ProgCont API szolgáltatásait felhasználó alkalmazásokat fejleszteni.

## 3.1. Kliens program

Az alkalmazás első elkészült komponense a ProgCont API kliens, amely a felhasználói interfész szolgáltatásainak eléréséért felelős. Egy olyan Java nyelven íródott mintakód, amely segítségével feltölthetők a kiértékelendő programozási feladatok megoldásai és tesztesetei, továbbá letölthetők a kiértékelés eredményei.

Az API felhasználásához egy felhasználóra és a hozzá tartozó két regisztrációs kulcsra van szüksége, amelyeket a kliens programba kell beépíteni. Innentől kezdve a kommunikáció a ProgCont szolgáltatásaival hiteles és titkos, annak érdekében, hogy érzékeny környezetben – például számonkérések vagy versenyek lebonyolítása során – is alkalmazható legyen.

## 3.2. Szakkörök támogatása

A felhasználói webinterfésze jelenleg tervezési szakaszban jár.

A tervezés során elsősorban a már meglévő szoftvereink funkciói közül merítettünk ötleteket:

1. `exam.progcont.eu` egyetemi programozás kurzusok számonkéréseinek menedzselésére készült zárt webalkalmazás, mely az ELTE Bíró [8] nevű webfelületéhez hasonló,
2. `deikrpcs.progcont.eu` a Debreceni Egyetem Regionális Programozó Csapatversenyéhez készült webalkalmazás, ami a PC$^2$ [6], illetve a Mooshak [5] szoftverek ötletét gondolta tovább a helyi igényekhez alkalmazkodva,
3. `progcont.hu` nyilvánosan is elérhető gyakorlófelület, amely az ELTE Mester [8] nevű és az UVa Online Judge [7] számunkra fontos jellemzőit ötvözi.

A Moodle CodeRunner [4] kiegészítőjéhez hasonló alkalmazást tervezünk megvalósítani, ebben a Moodle-ben már jól bevált alábbi funkciókkal, például:

- felhasználói fiókok kezelése,
- tananyagok feltöltésének lehetősége,
- programozói feladatok kérdésbankja,
- jelentések és statisztikák megjelenítése.

# 4. Összefoglalás

Cikkünkben bemutatásra került a ProgCont alkalmazás legújabb fejlesztési iránya, amely reményeink szerint kitágítja a felhasználók körét. Egy egyszerű példa segítségével röviden bemutattuk, hogy milyen szolgáltatásokat nyújt a ProgCont API a lehetséges felhasználók számára. Az itt bemutatott

változat már egy hosszabb fejlesztési szakasz eredménye, amellyel egy logikusan felépülő és könnyen használható interfész jött létre. Szintén bemutattuk a kliens program első változatát, amely leegyszerűsíti a felhasználók számára az API szolgáltatásainak használatát.

A fejlesztés itt nem fejeződött be, a közeljövőben olyan webalkalmazásokat szeretnénk készíteni, amelyek példaként állnak az API-t felhasználni kívánók előtt. Első ilyen alkalmazásunk a középiskolákban tanító informatikatanárok számára nyújt segítséget programozó szakkörök lebonyolításához.

## Köszönetnyilvánítás

A ProgCont rendszer további fejlesztését az „EFOP-3.4.4-16-2017-00023 AZ MTMI szakokra való bekerülést elősegítő innovatív programok megvalósítása a Debreceni Egyetem vonzáskörzetében" című projekt támogatta.

## Irodalom

**1.** R. Tóth, M. Kósa, T. Kádek, J. Pánovics: *The Development of Evaluation Systems at the Faculty of Informatics*, University of Debrecen. In: L., Gómez Chova; A., López Martínez; I., Candel Torres – INTED2019 PROCEEDINGS Valencia, Spanyolország: International Academy of Technology, Education and Development (IATED) (2019) 5552–5559.

**2.** R. Tóth, M. Kósa, T. Kádek, J. Pánovics: *A ProgCont rendszer új szolgáltatásaira épülő alkalmazások*. SZÁMOKT 2018, Tusnádfürdő, Románia: Erdélyi Magyar Műszaki Tudományos Társaság, (2018) 331–336.

**3.** T. Kádek, P. Biró: *A ProgCont API: programozási feladatok megoldásainak újszerű kiértékelése*. SZÁMOKT 2019 konferencia, Erdélyi Magyar Műszaki Tudományos Társaság (EMT) kiadó, Temesvár (2019) 191-195.

**4.** CodeRunner Documentation V3.1.0 (2019) https://coderunner.org.nz/mod/book/tool/print/index.php?id=184 (utoljára megtekintve: 2019.10.31.)

**5.** M. Rubio-Sánchez, P. Kinnunen, C. Pareja-Flores, Á. Velázquez-Iturbide: *Student perception and usage of an automated programming assessment tool*, Science of Computer Programming, 31, (2014) 453-460. https://doi.org/10.1016/j.chb.2013.04.001

**6.** A. S. Arefin, M. A. Rahman, S. A. Sharna, S. Mahmud, D. M. Kaykobad: *Secured Programming Contest, System with Online and Real-time Judgment, Capability*, 8th International Conferrence on, Computer and Information Technology (ICCIT), IUT, Dhaka, (2005) 584–586.

**7.** E. Verdú, L. M. Regueras, M. J. Verdú, J. P. Leal, J. P. de Castro, R. Queiros: *A distributed system for learning programming on-line*. Computers & Education. (58) 1, (2012) 1–10. https://doi.org/10.1016/j.compedu.2011.08.015

**8.** Győző Horváth, Gyula Horváth, László Zsakó: *A bíró és a mester – az online értékelés szerepe a programozás oktatásában, Mérési és értékelési módszerek az oktatásban és a pedagógusképzésben*. Szerk. Károly Krisztina és Homonnay Zoltán, Diszciplínák tanítása – a tanítás diszciplínái 5, ELTE Eötvös Kiadó, Budapest (2017) 89–103.

# Vizuális programozás nyújtotta lehetőségek a középiskolai informatika órán

Kelemen András[1], Árgilán Viktor Sándor[2]

{[1]kelemen, [2]gilan}@jgypk.szte.hu
SZTE JGYPK Informatika Alkalmazásai Tanszék[1,2]
Szegedi Radnóti Miklós Kísérleti Gimnázium[1]

**Absztrakt.** Az infokommunikációs technológiák látványos fejlődése és elterjedése egyre inkább megköveteli a felhasználói felületek és a mögöttes működési logika ismeretét. A Nemzeti Alaptanterv alapján megfogalmazott érettségi követelményekben az irodai felhasználói ismertek (dokumentum szerkesztés, táblázat kezelés, egyszerű adatbázis kezelés) dominálnak. Emelt szinten a programozási feladat követelményrendszere teljes egészében csak konzol alapú programozási ismereteket követel meg. A konzol alapú programozás azonban nem látványos, többször elmaradnak a sikerélmények. Előadásunkban rámutatunk arra, hogy a vizuális programozás nyújtotta felhasználói élmény mennyire segíti a programozás megszerettetését és a programozói gondolkodásmód fejlesztését.

**Kulcsszavak:** vizuális programozás, grafikus felület, eseménykezelés, projekt szemlélet

## 1. Bevezetés

Az infokommunikációs technológiák látványos fejlődése és elterjedése egyre inkább megköveteli a felhasználói felületek és a mögöttes működési logika ismeretét [4,5]. A Nemzeti Alaptanterv alapján megfogalmazott érettségi követelményekben az irodai felhasználói ismertek (dokumentum szerkesztés, táblázat kezelés, egyszerű adatbázis kezelés) dominálnak [4,5]. Emelt szinten a programozási feladat követelmény rendszere teljes egészében csak konzol alapú programozási ismereteket követel meg. Azonban a konzol alapú programozás nem látványos, nehezebben jönnek a sikerélmények. Ráadásul a diákok heterogén érdeklődése, valamint a tanórák 45 perces beosztása miatt a programozás oktatása eleve nehéz feladat. A rendelkezésre álló idő – a menet közben óhatatlanul felmerülő kódolási, technikai problémák és a diákok eltérő sebessége miatt – gyakran nem elég a tanórára kitűzött feladat teljes megoldására. A fentiek hatására a diákok jelentős része elfordul a programozástól, noha sokan szeretnének közülük informatikával foglalkozni. Megoldás lehet a programozás megszerettetésére, ha az informatika tanár programozás szakkört indít, azonban félő, hogy ide csak azok fognak járni, akik már elkötelezettek [3]. Másik út lehet, hogy a programozás oktatását nem a klasszikus utat bejárva végezzük, hanem a vizuális programozás nyújtotta azonnali sikerélményeket próbáljuk a cél érdekében felhasználni. [1-3]

A cikkben egy tapasztalatokon alapuló módszert mutatunk arra, hogy a fent említett problémát hogy lehet áthidalni és megszerettetni a programozást.

## 2. A módszer

A módszer megalkotásakor a kiindulási ötlet az volt, hogy a Szegedi Radnóti Miklós Kísérleti Gimnáziumban a 6 évfolyamos speciális matematika tagozaton a 7. osztályban LOGO-val tanulnak programozni a gyerekek. Az egyszerű utasításkészlet és a teknőc nyújtotta vizuális visszacsatolás még az informatikától idegenkedőket is arra sarkallta, hogy megpróbálják kiszenvedni az adott feladat megoldását. Amikor a 8. osztálytól kezdve a Logo-t felváltotta valamilyen komolyabb programozási nyelv (C/C++, C#, Java) kevés kivételtől eltekintve azonnal elmaradtak a sikerélmények és

jöttek a problémák, melynek egyik oka lehet, hogy programozási alapok oktatásánál elmarad a vizuális visszacsatolás.

A programozás alapjainak oktatása általában az alábbi tematika szerint szokott történni:

- Az adat és az algoritmus fogalma példákon keresztül.
- A hardver és a szoftver fogalma. Az operációs rendszer feladatai.
- A konzol (parancssor használata, fontosabb parancsok)
- Programok felépítése (adatok+algoritmusok= programok), programozási nyelvek, programozási paradigmák.
- A választott fejlesztő környezet ismertetése.
- A változó, mint adattároló fogalma. Elemi adattípusok a választott nyelvben.
- Változók deklarálása, értékadás.
- Adat bekérés és adatkiírás a konzolra.
- Matematikai és logikai műveletek.
- Iterációk,
- Elágazások
- Véletlen számok és használatuk a választott nyelvben.
- Tömbök használata.
- Keresés, kiválogatás, szétválogatás
- Unió, metszet
- Rendezés tömbökben.
- String műveletek.
- Láncolt listák, bináris fák
- File kezelés (írás, olvasás)

A tematikából már látható, hogy a programok futtatásához szükség van a parancssor használatára, és a baj itt kezdődik, nevezetesen ismerni kell néhány fájl és könyvtár kezelésre vonatkozó parancsot.

## 2.1. Programozás alapjainak oktatása vizuális programozással

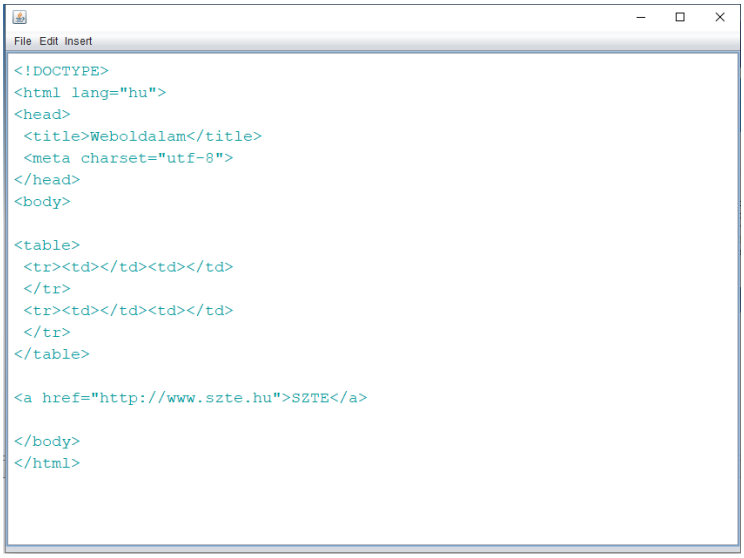
A vizuális programozást általában a felhasználói felület elkészítésével kezdjük, melyek a modern fejlesztő környezetekben (Visual Studio, Netbeans) nem igényelnek kódírást. A felület elemeit a „fogd és vidd" módszerrel lehet összerakni. A fejlesztő környezet, pedig automatikusan elkészíti felületet futtató programkódokat. Az első sikerélmény így megvan. Még nem írtunk semmilyen program kódot, de már van egy „működő" programunk. Működő, mert fut, de csak azokra az eseményekre reagál, melyeket a fejlesztő környezet automatikusan belerak. Itt el kell mondani a diákoknak, hogy minden olyan szoftver, amelynek grafikus felhasználói felülete van, ott a program a felhasználói felületen bekövetkező *eseményekre* reagál. Szerencsére a grafikus keretrendszerek az események kezelését úgy oldják meg, hogy az esemény bekövetkeztekor lefuttatnak egy üres metódust is. Ebbe a metódusba kell nekünk az adott eseményre vonatkozó logikát leprogramozni. Ez elsőre természetesen sokkoló hatású, de a diákoknak azt kell mondani, hogy az esemény kezelést fekete doboznak tekintsék, és csak azzal foglalkozzanak, hogy az esemény bekövetkeztekor hogyan reagáljon a rendszer.

Természetesen az oktatásban be kell tartani a fokozatosság elvét, ezért kezdetben csak kettő vezérlő elemet használunk. Szövegdobozt az adatbevitelre és megjelenítésre. Nyomógombot az esemény kezelésre. Mivel a szövegdoboz csak stringeket tud kezelni, ezért ez az egyszerű felület kiválóan alkalmas a string műveletek és velük együtt az iterációk és elágazások gyakorlására. Néhány példa:

- Szövegben adott betű másik betűre cserélése.

- Szövegben betűk előfordulásainak megszámlálása.
- Szöveg titkosítása.
- Szövegek darabolása, összefűzése.

Többsoros megjelenítésre alkalmas szövegdobozt, valamint véletlen számokat használva már lehetőségünk van a tömbök használatának begyakorlására a klasszikus példákon (maximum és minimum keresés, átlagszámítás, indexek keresése, rendezések, stb) keresztül. Valamint lehetőségünk van a file műveletek (mentés, betöltés) illusztrálására (1. ábra). Ennél a feladatnál a nyomógombokat menükkel helyettesítettük. Ezzel egy újabb vezérlő elem használatát ismerték meg a diákok.



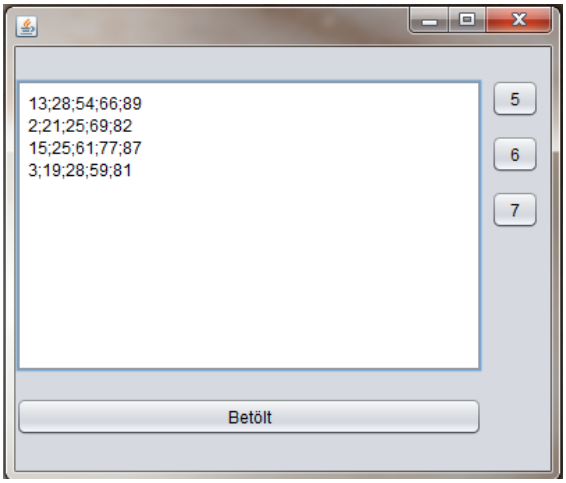

**1. ábra:** Egyszerű html editor képernyő képe

A következő órán a programot kiegészítettük a „New” az „Insert table”, és az „Insert link” parancsokkal, melyeknél string műveletekkel szúrtunk be HTML utasításokat a kurzor pozíciójába.

## 2.2. Projekt szemlélet

Természetesen, ahogy haladunk előre a vizuális programozásban egyre több vezérlő elem kerül elő és velük együtt az összetettebb adatszerkezetek (kétdimenziós tömbök, rekordok, láncolt listák). Az illusztrációul szolgáló példák pedig egyre összetettebbek, egyre inkább az alkalmazásfejlesztés irányába mutatnak [1,2]. Itt már egy 45 perces óra nem elegendő nulláról indulva egy feladat teljes megoldásához. Projektekben, projekt szemléletben kell gondolkodni. Az egyéni motiváció fenntartásához érdemes a diákokat 2-3 fős csapatokra osztani. Amennyire lehet, itt már hagyni kell őket önállóan dolgozni, természetesen szigorú határidőt szabva a projekt elkészültének. Néhány általunk használt példa projekt:

*Lottószám generáló* (2. ábra):




**2. ábra:** A Lottószám generáló alkalmazás képernyő képe

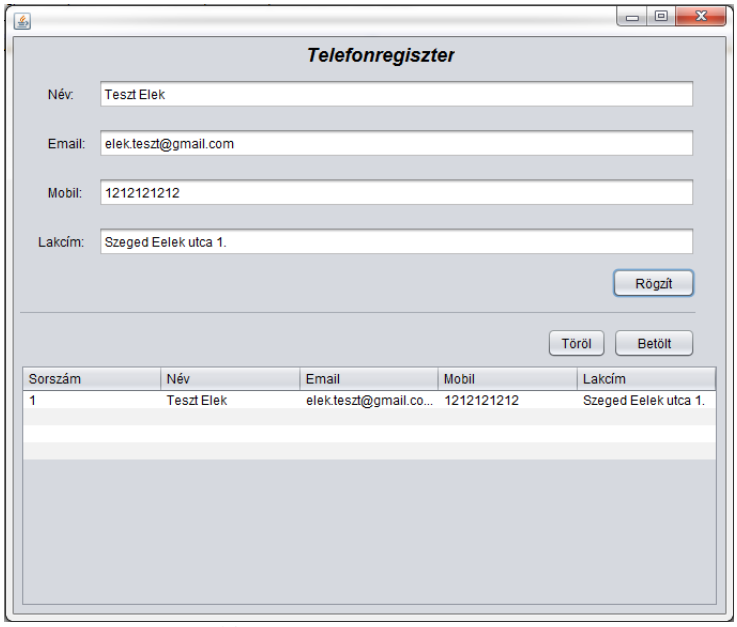
Ennek a projekt feladatnak a részletes specifikációja a következő:

Készítsen grafikus alkalmazást, amely lottószámok egyszerű kezelését teszi lehetővé.

- A program az 5-ös, 6-os és 7-es lottó szabályainak megfelelő számokat generál.
- A felhasználó az ábráknak megfelelően nyomógombok segítségével dönti el, hogy melyik lottóhoz generál számokat.
- A nyomógombokon történő minden egyes kattintáskor az adott lottó szabályainak megfelelő számsor generálódik és a megjelenítő ablakban új sorban jelenik meg, a soron belül növekvő számsorrendben.
- A generált számok automatikusan elmentődnek az adott lottó szabályainak megfelelő oszlopszámú pontosvesszővel tagolt csv formátumú fájlba. A sor végén ne legyen pontos vessző!
- Az adott lottójátékhoz tartozó első mentés alkalmával a program a fájl kiválasztó ablak segítségével kérdezzen rá a fájl névre.
- Legyen lehetőség a már elmentett lottószámok betöltésére, megjelenítésére

Ennél a feladatnál már előkerül az egészszámok tömbökben való tárolása, tömbök véletlen számokkal adott feltételek mellet történő feltöltése. További programozási feladat még a tömbelemek szövegdobozban történő megjelenítése (egészszám - string konverzió).

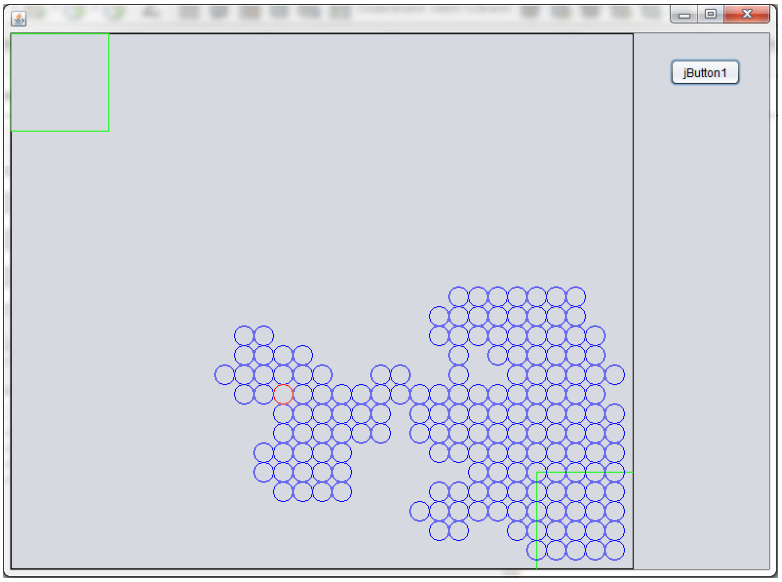
*Telefonregiszter* (3. ábra):




**3. ábra:** A Telefonregiszter alkalmazás képernyő képe

Az oktatási cél itt a rekord és a láncolt lista használata a személyek memóriában történő tárolására, a lista tárolása csv fájlban, valamint a grafikus vezérlőelemek (szövegdoboz, nyomógomb táblázat, fájlválasztó) együttes használata.

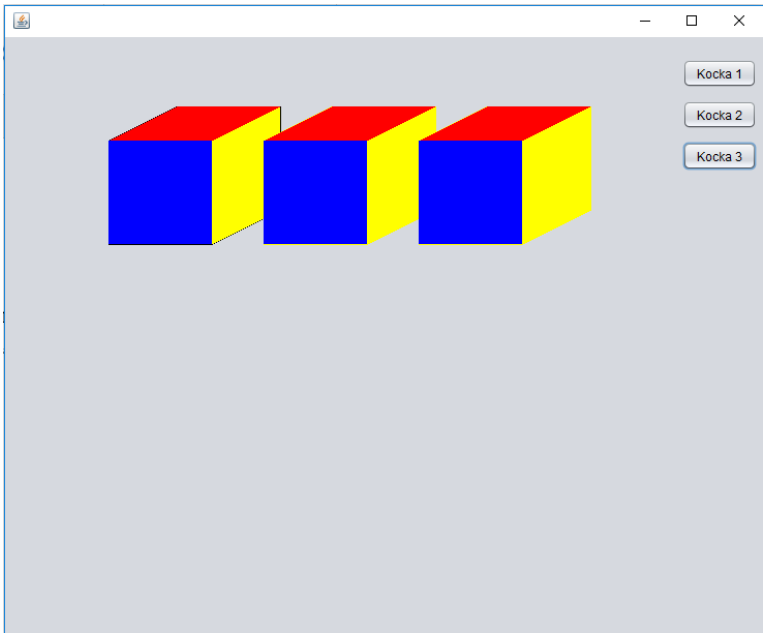
*Részeg tengerész* (4. ábra):




**4. ábra:** Részeg tengerész képernyő képe

A grafikus felületen az időzítő és a véletlen szám generátor együttes használatának bemutatására szolgál a *Részeg tengerész* nevű alkalmazás (4. ábra), mely alapvetően a Brown mozgás szimulálására szolgál. Ennél az alkalmazásnál a játéktér bal felső és jobb alsó sarkában van egy-egy zöld téglalappal jelzett „kocsma". A szimuláció kezdetén a piros körrel jelzett tengerészünk a jobb alsó kocsmában helyezkedik el, és itt alaposan felöntött a garatra. Ekkor szólnak neki, hogy a bal felső kocsmában sokkal finomabb a rum. Tengerészünk elindul, de olyan részeg, hogy minden egyes lépésénél véletlenszerűen jobbra vagy balra, vagy előre, vagy hátra lép. Az alkalmazás az időzítő minden egyes aktiválódásakor a tengerész aktuális pozícióját kék színnel kirajzolja, majd piros színnel az új pozícióba lépteti.

*Kocka rajzoló* (5. ábra):




**5. ábra:** Kocka rajzoló képernyő képe

Az oktatási cél itt a háromdimenziós ábrázolás gyakorlása. A részletes feladatspecifikáció szerint a nyomógombra kattintva mindig annyi kiszínezett kockát rajzol program, amekkora szám a nyomógombra írva van.

## 3. Tapasztalatok, eredmények

A középiskolában a tanulói és tanári munka sikerességét általában az emelt szintű érettségi és esetleg a tanulmányi versenyek eredményein szokták számszerűleg mérni. Tisztában vagyunk vele, hogy ez nem teljesen objektív mutató. Erősen függ a diákok kezdő felkészültségi szintjétől, az iskola és a tanár lehetőségeitől, a tananyag és tantervi előírásoktól.

Tapasztalataink szerint, ha az informatika órán a hangsúlyt és ezzel együtt az óraszámokat a programozásra fordítjuk, akkor az itt megszerzett készségek, gondolkodásmód más tanórán (pl. matematika, fizika, nyelvtan, egyes szakmai tárgyak) is előjönnek, hasznára válnak a diáknak. Jelentősen javul a diákok probléma átlátó és problémamegoldó képessége. A programozásba és főleg a

vizuális programozásba fektetett munka és idő busásan megtérül az alkalmazói szoftverek oktatásánál, hiszen itt már pontosan értik e szoftverek felépítésének és működésének a logikáját.

Az emelt szintű érettségin a programozási feladat „mumus" feladat szokott lenni. Tapasztalataink szerint vizuális programozás nyújtotta fejlesztői szemlélet sokkal könnyebbé teszi ezen összetett feladat megoldását.

Meglátásunk szerint a tanulmányi versenyek legfőbb hozadéka a rendszeres elmélyült tanulásra való késztetés. Amennyiben a vizuális programozással sikerült megszerettetni a programozást a diákkal, akkor sokkal könnyebb rávenni őt az algoritmusok és az adatszerkezetek mélyebb tanulmányozására és releváns tanulmányi versenyeken történő indulásra.

## Irodalom


1. Dr. Kelemen András: *Programozás oktatás projektalapon a közoktatásban*. In: Koltai, László (szerk.) Hazai és külföldi modellek a projektoktatásban Budapest, Magyarország: Óbudai Egyetem Rejtő Sándor Könnyűipari és Környezetmérnöki Kar, (2019) pp. 122-131, 9 p. ISBN 978-963-449-133-0
2. Dr. Kelemen, András; Árgilán, Viktor Sándor: *Szimulációs algoritmusok oktatása projekt szemléletben*. In: Bodáné, Kendrovics Rita (szerk.) Hazai és külföldi modellek a projektoktatásban: Nemzetközi Tudományos Konferencia tanulmánykötete Budapest, Magyarország: Óbudai Egyetem Rejtő Sándor Könnyűipari és Környezetmérnöki Kar, (2018) pp. 335-343, 9 p. ISBN 978-963-449-024-1
3. Kelemen, András; Árgilán, Viktor Sándor: *Szimulációs algoritmusok az informatikatanár szakos hallgatók képzésében*. In: Szlávi, Péter; Zsakó, László (szerk.) INFODIDACT 2017, Budapest, Magyarország: Webdidaktika Alapítvány, (2017) p. ISBN:9786158060813
4. Árgilán, Viktor; Kelemen, András: *Digitális kompetenciák a NAT-ban és a XXI. század elvárásai*. In: Zsakó, László; Szlávi, Péter (szerk.) INFODIDACT 2016: Informatika Szakmódszertani Konferencia Zamárdi, Magyarország: Webdidaktika Alapítvány, (2016) Paper: AVKA.pdf, 4 p.
5. Árgilán, Viktor Sándor; Kelemen, András: *Az elvárt digitális kompetenciák a XXI. században és az informatika oktatás gyakorlata a közoktatásban Magyarországon*. In: Bíró, Károly-Ágoston; Sebestyén-Pál, György (szerk.) XVII. ENELKO – XXVI. SzámOkt Nemzetközi Energetikai-elektrotechnikai és Számítástechnikai Konferencia Kolozsvár, Románia: Erdélyi Magyar Műszaki Tudományos Társaság (EMT), (2016)

# Kompetenciafejlesztés IoT rendszerekkel

Korom Szilárd[1], Dr. Illés Zoltán[2]

[1]korom.szilard@gmail.com; [2]illes@inf.elte.hu
ELTE IK

**Absztrakt.** A középiskolai informatika/programozás oktatás ritkán helyezi az ismereteket kontextusba. A diákok ritkán látják a *Big Picture*-t, nehezen tudják megfogalmazni, hogy egy új témakör bevezetésére miért van szükség, azzal milyen problémát akarunk megoldani. A kutatás célja ennek azonosítása az elsőéves programozó-informatikus hallgatók körében, valamint egy lehetséges megoldás megfogalmazása az *IoT* (Internet of Things – A dolgok internete) *rendszereken* keresztül.

**Kulcsszavak:** programozás, informatikai kompetenciák, valós idejű rendszerek, beágyazott rendszer, raspberry pi, rendszerszintű gondolkodás

## 1. Bevezetés

A programozás oktatás során leginkább az *algoritmikus gondolkodás kompetencia* kerül előtérbe középiskolai szintén. Azonban az még jellemzően a felsőoktatásra is igaz, hogy ritkán hangsúlyozódnak a valós életbeli, komplex problémák, ahol a különböző technológiákat, paradigmákat integráltan kell alkalmazni. Ez azért probléma, mert a diák/hallgató az oktatás keretein belül csak elszórtan tud kilépni a konkrét tananyag egységből. Például az „adatbázisok" tananyag középiskolában is [1], de az nem kerül elő hogyan és mire használják a való életben.

Egy korábbi cikkünkben kiemeltünk 4 kompetenciát az informatikát érintők [2] közül, amikor a „milyen jellegű feladatokkal érdemes tanítani a programozást" [3] kérdésre kerestem a választ:

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

Jelen esetben azt kívánjuk bemutatni, hogy létezik olyan technológia, keret, melyben a diákokhoz mérten (életkor, érdeklődés, cél) kialakítható egy tananyag úgy, hogy ezt a 4 kompetenciát érintse (főleg), azaz kilépjen az *egyedi összetevők* szintjéről, a konkrét eszköz használatából, s *rendszer szintű* szemléleted adjon, kontextusba helyezze a megtanultakat.

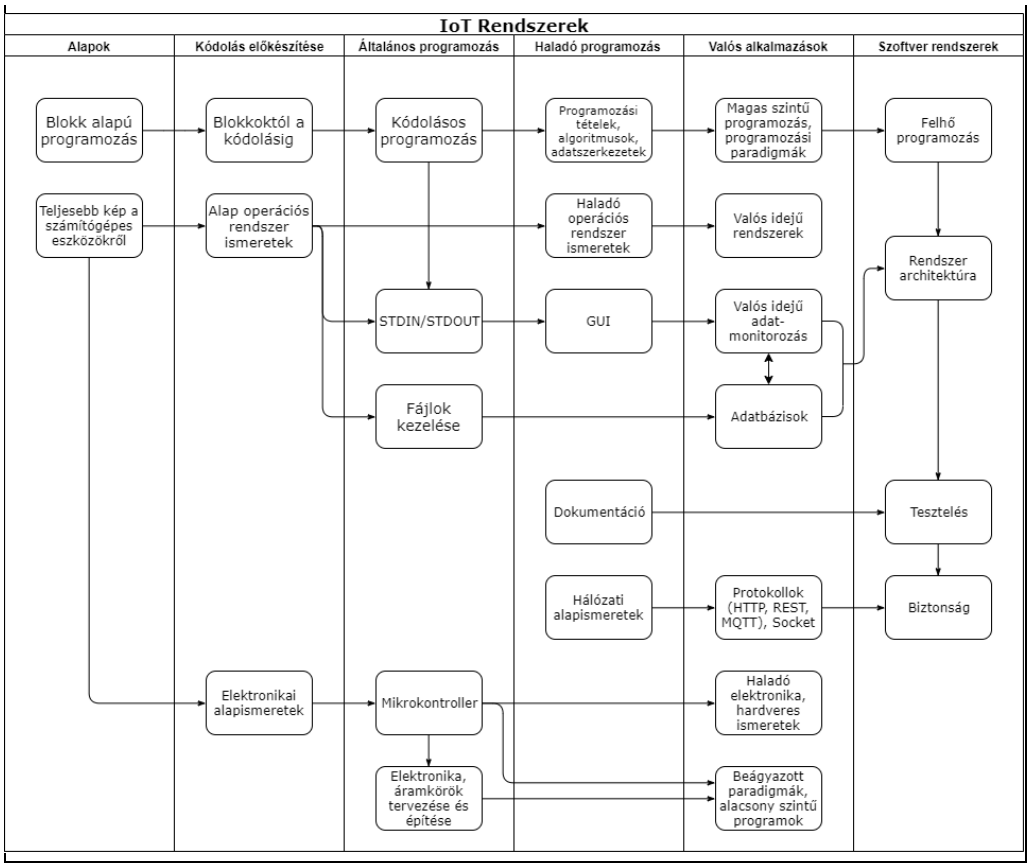
A cikkben az *IoT*[7] *rendszereket* hozzuk fel példának, megpróbáljuk pontosabban leírni mit szeretnénk/tudunk fejleszteni ezekkel az eszközökkel, majd egy kérdőív segítségével megmutatjuk, hogy a felvetett kompetenciák bizony valóban gondok okoznak a diákok számára.

## 2. Programozás oktatás moduljai IoT eszközökkel

Egy korábbi tanulmányunkban [4] a programozás oktatással foglalkozó néhány cikkek, valamint egyetemi tantárgyak leírásai [4, 5, 6] alapján az alábbi ábrában gyűjtöttük össze, milyen *modulokat*

[7] Internet of Things: A dolgok internete, vagy olyan apró céleszközök rendszere, melyek az interneten kommunikálnak egymással. Ezek jellemzően beágyazott, *valós idejű rendszerek* (de nem feltétlenül).

lehet oktatni *IoT* rendszerekkel. Azóta az ott bemutatott ábrát némileg módosítottuk, kiegészítettük, így kapjuk az 1. ábrát.




**1. ábra:** *IoT* rendszereken keresztül oktatható *modulok* és egymásra épülésük

Az 1. ábra azonban az *IoT* rendszerektől függetlenül is alkalmazható. Például GUI-t, adatbázist, vagy a tesztelési mintákat ettől teljesen függetlenül is lehet oktatni. Ezzel a szemmel az ábra az egész, és az egységek kapcsolatát tükrözi és további kérdéseket vet fel, ugyanis „a komponensek működése, a részproblémák megoldása nem okozza a rendszer működését" [3]. A cikk fő célja annak a hipotézisnek a bizonyítása, hogy még ha a diák ismer is egy-egy *modult*, nem feltétlenül tudja azt egy magasabb nézőpontból értelmezni, vagy indirekt módon alkalmazni, például, ha nem közvetlenül arra kérdezünk rá, hogy *mire jó az adatbázis*, hanem hogy azt hol és milyen körülmények között hasznos alkalmazni. Ha ezt sikerül bizonyítani, valamint a fenti ábrát elfogadjuk, mint alkalmas leírását az *IoT* eszközökkel való programozás oktatásnak, akkor egy alkalmas keretet kapunk, mely a skálázható, fókuszpontja mozgatható, s egyszerre képes *egyedi összetevők*- és *rendszer szinten* bemutatni egy problémát. Mit adnak tehát a begyázott, *valós idejű rendszerek*[8] a programozás oktatáshoz? Olyan eszköztárat adnak a tanár kezébe, mellyel a csoport életkorának, előismeretének, érdeklődési körének, s per-

[8] Itt elsősorban az olyan eszközökre gondoltunk, mely széles körben elterjedtek a programozás oktatásban, , mint pl.: micro:bit [5] Arduino, *Raspberry Pi* [6]

sze az oktató saját tudásának függvényében egy tananyagot tud felépíteni, mely látványos, bővíthető, s egy-egy probléma könnyen kilépteti a diákok az *egyedi összetevők* szintjén való gondolkodásból, azaz széleskörűbb tudást ad a tanulónak.

## 2.1. Egy példa

A fentebb leírtakat egy tananyag vázlatával (1. táblázat) szeretnénk szemléltetni, mely az ELTE-n MsC hallgatóknál egy *Embedded and Real Time Systems* nevű tárgy keretében lett kipróbálva. Itt a *Valós idejű rendszerek* elemen (az 1. ábrából kiindulva) volt a hangsúly, egészen egyszerűen azért, mert ez volt a tárggyal szemben a követelmény. A példa véleményünk szerint jól illusztrálja, hogy az *IoT* rendszereken keresztüli oktatás széles körű lehetőséget biztosítanak, rengetek technológia és programozási paradigma oktatását teszik könnyűvé úgy, hogy a diák kénytelen rendszer szinten gondolkodni, olyan megoldást adni, mely számos *modulból*, és azok együttes működésre bírásáról szólnak.

Az alapvető feltevés egyébként az volt, hogy *Raspberry Pi*[9] eszközökkel építsünk okos otthon projekteket, úgy, hogy a rendszer alkalmas legyen arra, hogy további eszközökkel bővítsük (pl.: egy új szoba felokosítása esetén), s az egészet monitorozni tudjuk egy grafikus alkalmazáson keresztül. Természetesen a fontos adatok az interneten keresztül elérhetőek legyenek (pl.: a monitorozás funkció), de építeni kellett egy lokális *IoT* átjárót (mert ez az iparban általában nagyon fontos).

| Óra | Leírás |
|---|---|
| 1. | **IoT bevezetés**<br>Témakör bevezetése, elméleti háttér áttekintése. Egy minta program bemutatása, mely egy valós életbeli *IoT* projektet demonstrál<br>**Hardverelemek összeépítése**<br>A hardverek összeépítése, összekötése, élőben kipróbálása mintakódokkal.<br>• Hogyan kommunikáljunk *Sense Hat*[10] *modullal* (szenzor adatok begyűjtése, LED mátrixra kirajzolás)<br>• Hogyan vezéreljük a kamera modult[11]?<br>*tkinter* Python GUI csomag alap funkcióinak kipróbálása |
| 2. | **MQTT protokoll**<br>MQTT broker installálása, konfigurálása és helyi hálózati kommunikációra használása. A *publish-subscribe* filozófia megismerése, események és eljárások használata az előző órai példák felhasználásával. |
| 3. | **Felhő szolgáltatások**<br>Firebase nem-relációs *valós idejű* adatbázis konfigurálása és integrálása egy okos otthon szimulációs projektbe. |
| 4-6. | **IoT Lab**<br>Önálló, valóság közeli rendszerek tervezése és implementálása 2 fős csapatokban. |

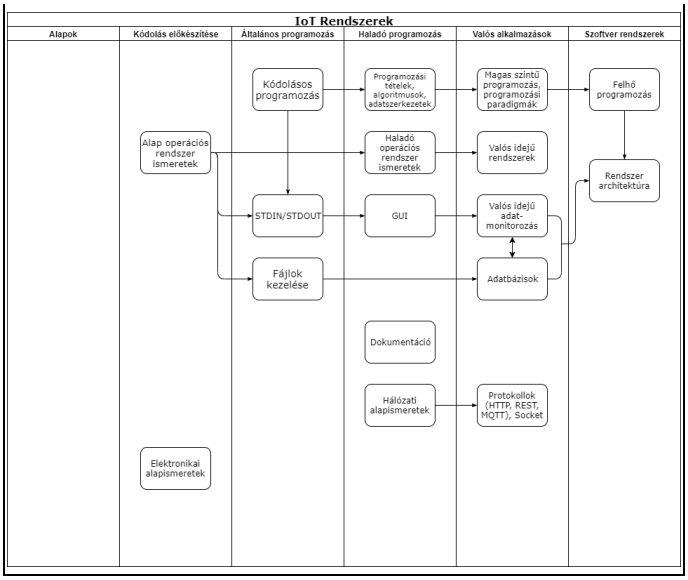
**1. táblázat:** Egy *IoT* tananyag címszavakban

[9] https://www.raspberrypi.org

[10] https://www.raspberrypi.org/products/sense-hat/

[11] https://www.raspberrypi.org/products/camera-module-v2/

Az 1. ábrából kiindulva szemléltetni kívánjuk mi mindent érintettünk ez alatt a 6*1,5 óra alatt. Látható a 2. ábrán, hogy viszonylag sokat (legalább is középiskolai szinthez mérve, így kijelenthető, hogy rövid idő alatt egy komplex rendszert kellett és tudtak alkotni a hallgatók.




**2. ábra:** Tananyag által érintett elemek

A skálázhatóság hangsúlyozása nagyon fontos. Nyilván ugyan ez nem tehető meg középiskolai körülmények között, de nem is ez a cél.

Lehetne ugyan ezt a projektet középiskolásoknak is tanítani? Véleményünk szerint igen, ha például nem kell interneten *valós idejű* adatbázison keresztül kommunikálni, vagy ha nincsen monitorozó alkalmazás, vagy lokális MQTT broker.

Lehetne ugyan ezt a projektet tovább vinni, még magasabb szinten oktatni? Meglátásunk szerint igen. Például, ha követelmény, hogy automatikus tesztek ellenőrizzék a rendszer helyes működését, ha nagyobb hangsúly van a biztonságon, vagy nem a kész *Sense Hat modult* használjuk, hanem magunk rakunk össze egy bonyolultabb áramkört.

## 3. A kérdőív

A kérdőívet elsőéves programozó informatikus hallgatók töltötték ki, pontosan 205-en. A kérdések megfogalmazásánál a következő szempontok domináltak:

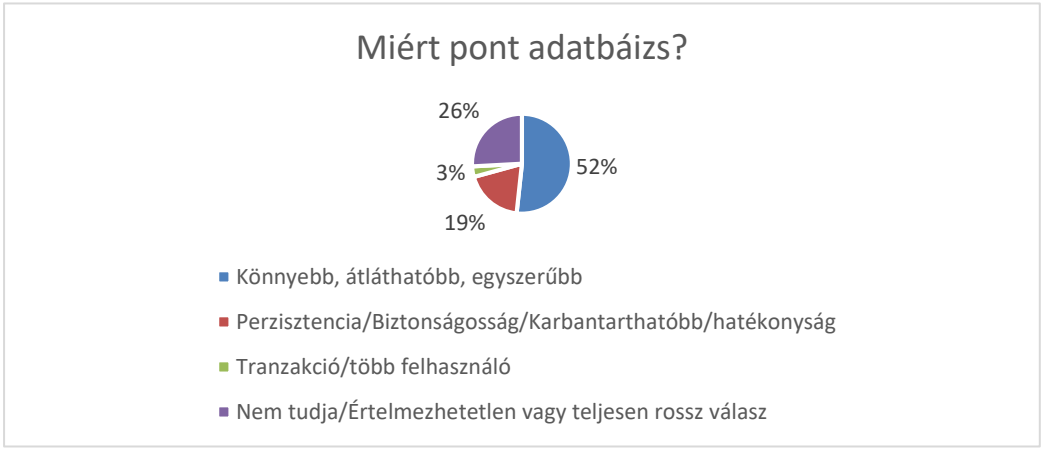
- középiskolai, esetleg aktuálisan tanult tananyaghoz kapcsoló kérdések legyenek
- attól, hogy a hallgató hibás választ ad, még elképzelhető legyen, hogy tudja a tananyagot, vagy teljesítse az adott tantárgyat (akár hibátlanul)
- a kérdések rendszerszintű kompetenciára kérdezzenek rá, vagyis a válaszok ismerete elengedhetetlenek legyenek a konkrét eszköz/technológia egy nagyobb projektben való alkalmazása során, de szükségtelenek az eszköz megtanulása során

Hogy az egyes kérdéseknél miért teljesülnek ezek a feltételek, arra egyesével ki fogunk térni. A kifejtős kérdéseknél a válaszokat próbáltuk kategorizálni. Ez sokszor nagyon nehéz volt, mert helyenként eléggé vegyes választ adtak a hallgatók. Az általunk megfogalmazott kategóriák a diagram-

moknál lesz láthatók, de igyekszünk majd példákat is adni, miket soroltunk az adott kategóriába. A továbbiakban felsoroljuk ezeket, s diagrammokon ábrázoljuk.

## 3.1. Miért tárolunk adatokat adatbázisban és nem fájlokban?

Az adatbázis-kezelés már a középszintű érettségiben is követelmény. Attól, hogy a diák tud táblákat létrehozni, azok tartalmát módosítani, hozzájuk lekérdezéseket írni a konkrét keretrendszerben (pl.: Access), még nem biztos, hogy látja az egyáltalán miért került képbe. A középszintű érettségin, valamint az egyetem első évében is jellemzően fájlokban történik az adattárolás, így nem biztos, hogy világos az adatbázis egyáltalán hogyan jött a képbe. Később azonban egy valós alkalmazásnál elengedhetetlen, hogy a diák tudja és értse a perzisztencia, a tranzakciók lényegét és értelmét.



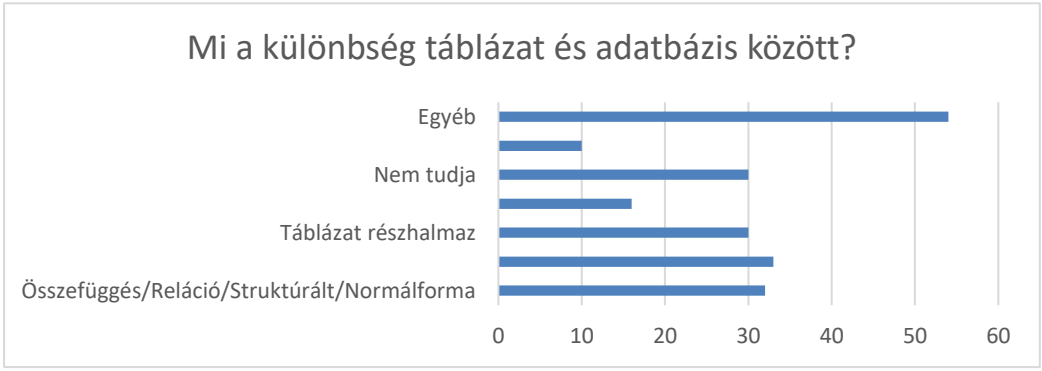

**3. ábra:** A kiértékelt, kategorizált válaszok az első kérdésre.

A *perzisztenciát*, *tranzakciót* konkrétan senki nem említette meg név szerint. Az ezeknek megfelelő kategóriákba azokat soroltuk, ahol bármiféle utalást találtunk ezekre a fogalmakra. Az utolsó kategóriába olyanok kerültek, amit nem igazán tudtunk hova tenni, például „Tömörítés".

Véleményünk szerint az eredmény eléggé lesújtó, bár a hipotézist igazolja: a diákok nem értik, az adatbázis miért és hol hasznos, egyáltalán mikor és mire használjuk.

## 3.2. Mi a különbség egy táblázat és egy adatbázis között?

Itt hasonlóan az előző kérdéshez, az adatbázis tágabb értelemben vett hasznára szerettem volna rákérdezni. A diákok a középiskolában gyakran nem értik, miért kell adatbázis-kezeléssel foglalkozni, amikor a táblázatkezelős feladatok nagyon hasonlóak, sőt, az előbbiben előkerülő feladatok nagy hányada meg is oldható az utóbbiban. Könnyen lehet tehát, hogy a hallgató ismeri és tudja használni az Excel és az Accesst anélkül, hogy tudná kívülről nézve mi a különbség a kettő között.




**4. ábra:** A kiértékelt, kategorizált válaszok a második kérdésre.

Ennél a kérdésnél a válaszok hihetetlenül vegyesek és érdekesek voltak. Nagyon nehéz volt bekategorizálni. Próbáltunk jóindulatúan következtetni arra, mire gondolhatott a hallgató. Természetesen nem konkrétan ezeket a válaszokat fogalmazták meg, pl.:

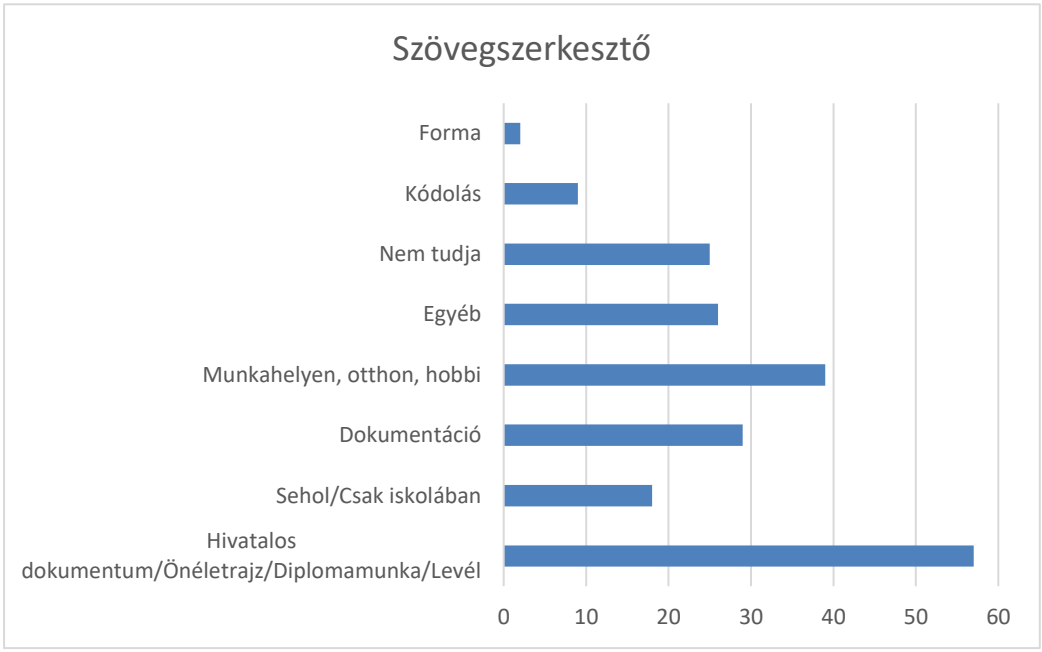
- Összefüggés/Reláció/Struktúrált/Normálforma: Az adatbázis táblái között is végezhetünk műveleteket.
- Egyéb: Adatbázisban több a funkció; Egy adatbázis sokféleképpen variálható, míg egy táblázat bizonyos szinten kötöttebb; Adatbázis olyan táblázat, amiben vannak adatok; Több dolog fér az adatbázisba
- Táblázat részhalmaz: Adatbázis táblázatok gyűjteménye; Több tábla

A kategorizálásnál tehát erősen érvényesült, hogy a diák gondolatmenetét igyekeztünk kategorizálni, s nem a konkrét választ. Még ezzel a jóindulatú módszerrel is nagyon érdekes eredmény jött ki.

Az eredmény azért nagyon meglepő (4. ábra), mert táblázatkezelést és adatbázis kezelést is biztosan tanultak a középiskolában, de ezek szerint egyáltalán nem értik a különbséget.

### 3.3. Fogalmazd meg, szerinted hol és milyen formában fogod használni a szövegszerkesztőben megtanultakat?

A szövegszerkesztés szintén követelmény már a középszintű érettségin is [1]. Fontossága pedig nem csak a programozók számára hangsúlyos. Van azonban a szövegszerkesztésnek általános szabályai, eszközei, mely szinte minden szövegszerkesztésre alkalmas környezetben előfordul (akár egy e-mail kliens esetében, akár fórumokon stb.). Például, ha ipari környezetben dokumentációt kell majd írnia a hallgatónak, akkor szinte bizonyosan muszáj lesz használnia ezeket az ismereteket, hiszen az valószínűleg nem Wordben fog történni. Hogy ténylegesen tudatában van-e a karakter-, bekezdés-, szakaszszintű formázásokkal (vagy ezen kategóriák puszta meglétéről) az más kérdés, de itt a felvetés arra irányul valójában érti-e, hogy amit megtanult az hol köszön majd vissza.




**5. ábra:** A kiértékelt, kategorizált válaszok a harmadik kérdésre.

A kategóriák meghatározásánál külön figyeltünk, hogy valaki válaszolt-e a *milyen formában* kérdésre. Itt elsősorban valamilyen általános szövegszerkesztő elemre gondoltunk, például, hogy a karakterformázások más programokban is hasonló logikát követnek, mint pl. Wordben. Ahogy látható erre 2 helyen volt összesen bármilyen utalás. Pár példa:

- Egyéb: Amikor szöveget akarok szerkeszteni
- Sehol: Ha kibukok és titkárnő leszek
- Forma: Formai szerkesztésnél

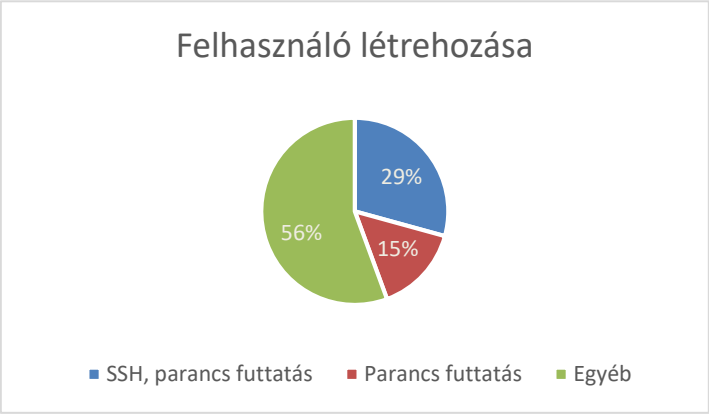
Szerintünk egyértelműen jelzik az adatok (5. ábra), hogy nem sikerült a szövegszerkesztőt általánosan értelmezni. Nagyon sok volt egy konkrét dokumentum megfogalmazása (utolsó pont), s nagyon kevés olyan, ami bármiféle módszerre, vagy logikára utalna. Az külön elkeserítő, hogyan sokan úgy gondolják, hogy semmire sem hasznos.

## 3.4. Tegyük fel az a feladatod, hogy létrehozz egy új felhasználót egy távoli Unix szerveren. Írd le általános milyen lépésekkel érnéd ezt el?

A kérdést azért mertük feltenni, mert aktuálisan a shell-scriptek tananyagot képeznek. Válaszként olyasmi várunk, hogy:

- SSH kapcsolat a távoli szerverrel admin jogosultságú felhasználóval
- Megfelelő parancs kiadása megfelelő paraméterezéssel

Mivel a felhasználók létrehozása konkrétan nem tananyag, a kérdés arra irányul, hogy érti-e mire fogja használni a shell környezetet, érti-e annak általános felépítését, hogy milyen jellegű problémákat tud majd ezen keresztül megoldani. Attól, hogy erre nem tud válaszolni, a tárgyat még hibátlanul teljesítheti, hiszen a zh keretében csak konkrétan megmutatott parancsokat kell használni, amit ettől függetlenül tudhat.




**6. ábra:** A kiértékelt, kategorizált válaszok a negyedik kérdésre.

Amikor a válaszokat kategorizáltuk, különösen igaz volt, hogy teljesen távoli válaszokat is a fenti kategóriákba soroltunk. Például:

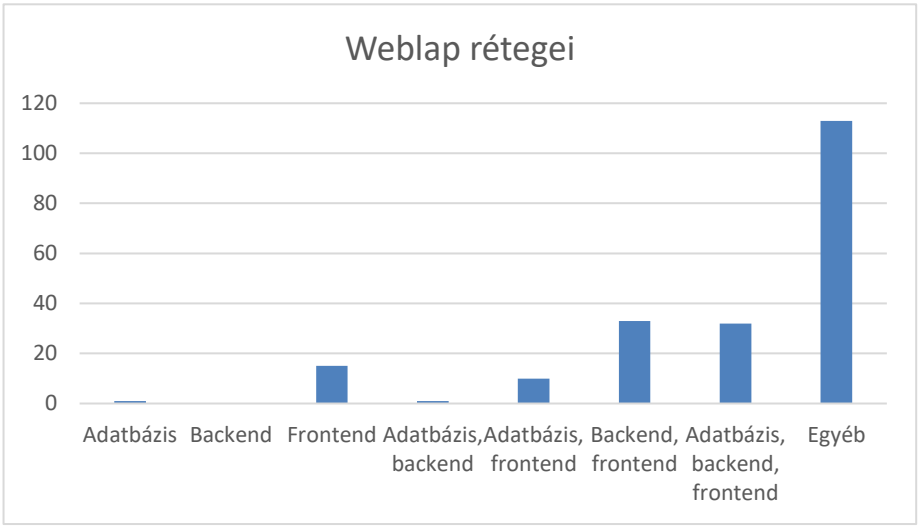
- SSH kapcsolat: Csatlakozás a szerverhez, felhasználó létrehozása
- Parancs futtatás: Google

Igyekezetem arra figyelni mire gondolhatott a hallgató. Ha az volt a válasza például, hogy „Google", szerintem arra gondolt, hogy megkeresi a megfelelő parancsot.

Ami viszont biztos, hogy még így is szembetűnő (a 6. ábra alapján), hogy ha felmerül egy probléma, azt nem tudják munkafolyamatként értékelni, hanem a konkrét megoldást várják, amit persze nem tudnak, hiszen nem tanulták a shell szkriptes tárgy keretein belül. Persze ezzel nem arra akarok utalni, hogy ha valóban kellene ne tudnák megoldani, inkább arra, hogy az oktatás keretein belül az általános, rendszerszintű gondolkodást nem sikerült átadni. Erre később lesznek rákényszerítve.

## 3.5. Tegyük fel készítened kellene egy webshopot. Sorold fel milyen rétegei vannak az alkalmazásnak!

Az adatbázis-backend-frontend hármasra vártunk, mint megfelelő válasz. Ezekből már középiskolában találkozott az adatbázissal, valamint a frontenddel. A backend réteggel még elképzelhető, hogy nem találkozott, mert egyetemen sem tananyag az első félévben. Ettől függetlenül az alkalmazás rétegelése elvárható a diáktól attól teljesen függetlenül, hogy például tud-e lekérdezéseket írni. A kutatás hipotézise persze az, hogy bár elvárható lenne, mégsem tudja megfogalmazni a rétegeket annak ellenére, hogy külön-külön ismeri azokat.



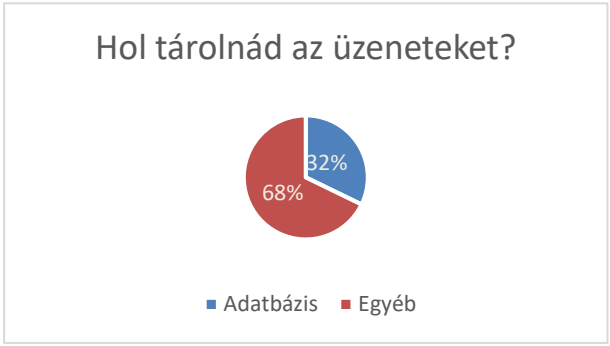

**7. ábra:** A kiértékelt, kategorizált válaszok az ötödik kérdésre.

Az utolsó adatsor a 7. ábrán jelzi azokat a válaszokat, ahol egyáltalán nem volt a rétegelésre utaló gondolat, vagy teljesen máshogyan osztotta fel (pl.: a weblap funkciói szerint). Mivel a többi adat összesen nem éri el az *Egyéb* kategóriába eső válaszok számát, ezért a felvetésünk helyes volt. Néhány példa a válaszokra:

- Adatbázis, backend,frontend: Design, szerverek, raktár az áruknak; Kinézet, háttérben futó scriptek, adatbázisok
- Adatbázis, frontend: Fizetési szoftver, képek, árak és termékek adatbázisa, adatrögzítő adatbázis..

## 3.6. Ha írnod kellene egy valós-idejű csevegő alkalmazást, hogyan tárolnád, menedzselnéd az üzeneteket?

A kérdést kifejezetten azért fogalmaztuk meg, hogy egy olyan példát hozzunk, amikor nem a technológia tanulását követi a feladat, hanem a valós problémához kell kötni a technológiát. Válaszként az adatbázisra való bármilyen utalást várunk. A kérdés egyébként azért is szerencsés a véleményünk szerint, mert kifejezetten egy programozói gondolkodás kell ahhoz, hogy a hallgató meg tudja válaszolni. Pusztán annak a felismerése, hogy egy informatikai technológia a válasz, már önmagában értékesnek mondható.

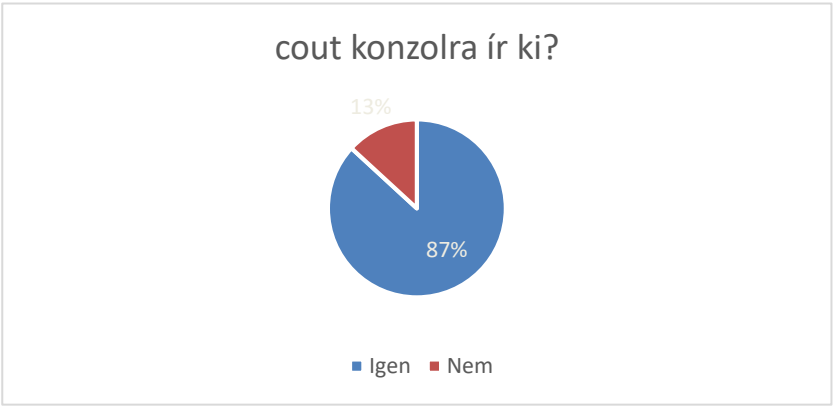



**8. ábra:** A kiértékelt, kategorizált válaszok a hatodik kérdésre.

Őszintén szólva ennél a kérdésnél arra számítottunk, hogy nem fogja a hipotézisünket igazolni, már csak azért sem, mert az első kérdés utalt rá, hogy adatbázisban érdemes információkat tárolni. Ennek ellenére nagyon szépen megmutatja a 8. ábra, hogy a probléma felől megközelítve egy tanult technológiát még nem képesek felismerni.

## 3.7. Igaz vagy hamis? C++ nyelven a cout a konzolra írja ki a megadott stringet.

A hallgatók az első félévben C++ nyelven (is) tanulnak programozni. Az eljárás tehát minden bizonnyal ismeretes a számukra. A válasz persze *hamis*, hiszen a standard kimenetre kerül a szöveg. Ha a C++ nyelvet például programozási tételek, alapvető algoritmusok, adatszerkezetek implementálására használják, a standard kimenet fontosságának megértése nem szükséges, elegendő, ha a konzolos alkalmazásban lát valamiféle eredményt. A későbbiekben azonban ez nagyon fontosság válik, hiszen, ha például a háttérben futó szervízeket kell majd írnia ipari környezetben, akkor a kimenetet minden bizonnyal át kell majd irányítania naplófájlokba. A kérdés számunkra azért is különösen érdekes, mert a shell szkript írást is most tanulják, ahol az átirányítások nagyon hangsúlyosak.



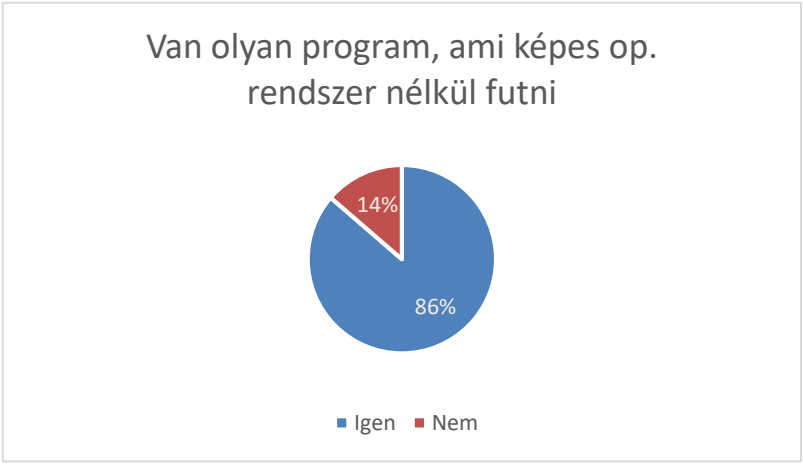

**9. ábra:** A hetedik (Igaz/Hamis) kérdés válaszai összesítve

Ennél a kérdésnél meg voltak adva az opciók, így nem kellett kiértékelni, csupán összesíteni. Az eredmény (9. ábra) szembetűnő: a hallgatók emlékeznek a parancsra, tudják használni, de a konkrét tapasztalatukra támaszkodnak, mely konkrét feladatokra épül, de nem értik általánosan az hogyan működik. Mivel tanulták már, hogy a sztandard kimenetet át lehet irányítani, ezért két összefüggő,

de apró képesség birtokában vannak, de az eredmények alapján ezt nem tudják összefűzni rendszerré.

## 3.8. Igaz vagy hamis? Van olyan program, ami képes operációs rendszer nélkül futni.

Itt valójában arra voltunk kíváncsiak, vajon értik-e mi történik, amikor futtatnak egy programot, el tudják-e különíteni, hogy a fordított állományuk éppen mivel kommunikál. Mivel tanulnak shell szkriptet írni, találkoztak olyannak, amikor egy másik program futtatja a kódot, de az imperatív programozás tárgyban C++-ban programoznak, ami a gépikódra fordul (amellett, hogy persze kommunikál az operációs rendszerrel, de nem feltétlenül van rá szüksége).

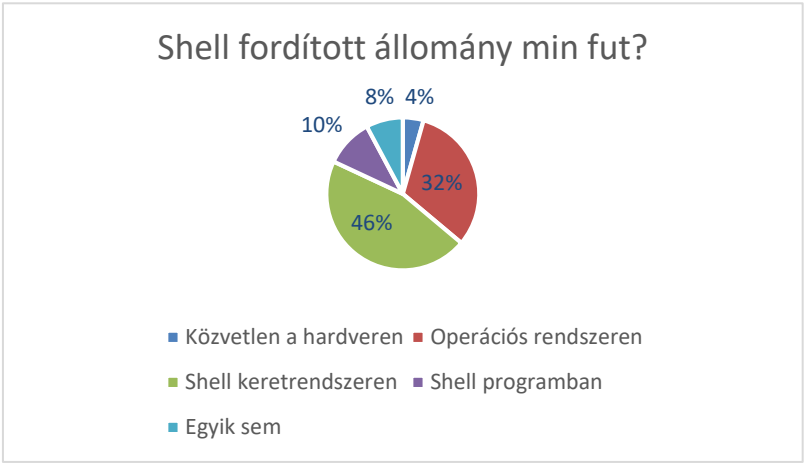



**10. ábra:** A nyolcadik (Igaz/Hamis) kérdés válaszai összesítve

Ez az eredmény (10. ábra) nem igazolja a hipotézist. A válaszok arra utalnak, hogy hallgatók értik mi történik a fordítás ideje alatt, sajnos azonban a következő kérdésre adott válaszok alapján egyáltalán nem vagyunk abban biztosak, hogy ez tényleg így van.

## 3.9. Ha egy shell fordított állományt (compilerrel) elindítasz, min fog futni? (közvetlen a hardveren, op rendszeren, shell keretrendszeren, shell programban, egyik sem)

Ez talán a legbeugratósabb mind közül. Maga a kérdés is értelmetlen, hiszen a shell szkriptet nem lehet fordítani, mert a shell program fogja értelmezni *valós időben* (ez a szkript definíciója). Amennyiben erre a hallgatók nem tudják a választ, az véleményünk szerint megvilágító erejű, hiszen ezt pont most tanulják (sőt, nagy részük ezen tárgy előadásán töltik ki a kérdőívet). Jelen állás szerint a hallgatók nagy része tud shell szkriptet futtatni, a tárgyat valószínűleg sikeresen fogják elvégezni, de az állományok futtatása elképzelhető, hogy fekete doboz maradt a számukra.



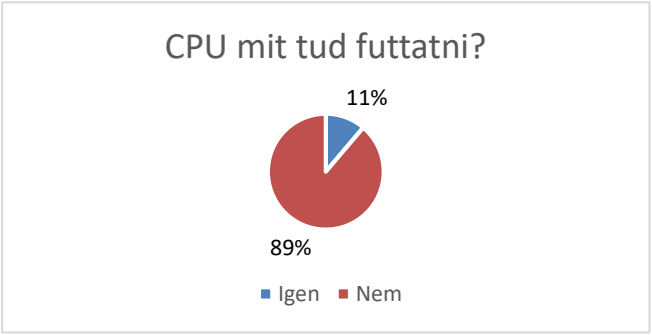

**11. ábra:** A kilencedik (feleletválasztós) kérdés válaszai összesítve

Az eredmények (11. ábra) szerintünk megdöbbentők. Igaz, a kérdés beugratós volt, de ha a hallgató még el is hiszi (beugrik a becsapásunknak), hogy létezhet shell fordítás, csak ő nem tud róla, akkor sem a shell keretrendszert (ahogyan a 11. ábra grafikonján látható) kellett volna választania (utalva az előző bekezdésre). Még a shell programban választ is meg lehetne magyarázni, hiszen itt előfordulhat, hogy a hallgató nem emlékszik mi a különbség szkript és program között, de arra igen, hogy a shell program értelmezi a kódját. A félév során a „Számítógépes rendszerek" előadást az egyik szerző tartotta (Illés Zoltán), a másik (Korom Szilárd) pedig bent volt az összesen (ahol tanulják ezt a témakört), így biztosan tudjuk, hogy az előadó többször is hangsúlyozta a megfelelő választ, azaz biztosan nem hallottak olyanról (egyetem keretein belül), hogy shell keretrendszer.

A gyakorlatokról tudjuk, hogy a többség tud shell szkriptet írni, de akkor mégis hogyan lehetséges, hogy nem tudják mi történik, amikor megfuttatják azt? Az eredményekből arra következtetünk, hogy azért, mert ha a *modul* szintjén maradnak (pl.: írj meg egy konkrét shell szkriptet; válaszold meg, mi az a keretrendszer), képesek megoldani a feladatot, mert ezt kérik tőlük számon. Egységesen az egészet azonban nem látják, nem tudják rendszerben értelmezni a megtanultakat.

## 3.10. Tud-e más kódot futtatni a CPU mint gépi kódot?

Itt a hardver és a szoftver kapcsolatára próbáltunk rákérdezni. Gyakran találkozunk azzal a kérdéssel középiskolások között, hogy a szövegszerkesztőben megírt program és az elektromos, áramkörökkel, bitekkel dolgozó hardver között mi a kapcsolat? Persze ha a hallgató a magas szintű nyelveknél marad, akkor ezzel a problémával talán soha nem találkozik. Teljesen vígan lehet kódot írni, tesztelni, futtatni, dokumentálni e nélkül s mégis, véleményünk szerint ez egy nagyon elemi kérdés, vagyis nem a rendszerre kérdez rá.




**12. ábra:** A tizedik (Igaz/hamis) kérdés válaszai összesítve

A 12. ábrán látható, hogy közel 90%-os a helyes válaszarány, vagyis ha olyan kérdést kap a hallgató, melyet tanult, s egyetlen egységre kell fókuszálnia (vagyis nem rendszer szinten, hanem az *egyedi összetevők* szintjén kell gondolkodnia), akkor azt meg tudják válaszolni.

## 4. Összegzés

A hipotézist, miszerint, ha diák ismer is egy-egy *modult*, nem tudja azt rendszer szinten igazolni, véleményünk szerint sikerült belátni. Az utolsó kérdés ilyen *modul* szintű volt, amire 90%-os aránnyal válaszoltak (12. ábra), míg a többiben (1 kivétellel) a nem helyes választ adták meg. A válaszok megítélésünk szerint további kérdéseket is felvetnek (például, hogy miért hiszik, hogy van „shell keretrendszer, ahogyan a 11. ábrán látható), hiszek meghökkentő arányok is jöttek ki.

Mivel a nem kifejtős kérdések esetén még kategorizálni lehetett a válaszokat, nem tartjuk kizártnak, hogy más szempontból is érdemes lehetne megvizsgálni az eredményeket. Például az érdekesség kedvéért a kérdőívben szerepelt a programozói tapasztalatukra is kérdés, ám a cikk keretében úgy éreztük nem fért bele a válaszok vizsgálata ennek tükrében.

A tanulmány elején igyekeztünk egy keretet, technológiát és módszert definiálni, mely alkalmas lehet a *modulok* összefűzésére, a rendszer szintű gondolkodás fejlesztésére. Az *IoT* rendszerek ugyanis olyan problémák megoldására, implementálására irányítják a diákokat, mely során muszáj több *modult* összeépíteniük, azokat egyidejűleg használniuk. Egy lehetséges tananyag pedig jól skálázható, fókuszpontja a csoporthoz és tanárhoz mérten beállítható. Annak igazolása, hogy ez valóban hatékony még nem történt meg, ám a jövőben erre is sor kerülhet.

## 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.

## Irodalom

1. *Informatika kerettanterv:* http://kerettanterv.ofi.hu (utoljára megtekintve: 2019.11.14)
2. Szlávi Péter, Zsakó László: *Informatikai kompetenciák – A valós világ modellezése.* In: Szlávi Péter, Zsakó László (szerk.) INFODIDACT 2013 ISBN:978-963-08-8387-0

3. Korom Szilárd: *Architektúrális gondolkodás fejlesztése valós idejű rendszerekkel.* In: Szlávi Péter, Zsakó László (eds.) INFODIDACT 2018 ISBN: 978-615-80608-2-0

4 Korom Szilárd: *IOT Systems in Education*. In: Marek, Smid; René, Bílik; Veronika, Stoffová (szerk.) XXXII. Didmattech 2019, Trnava, Szlovákia : Trnava University in Trnava Faculty of Education, (2019) p. 19

2. Universitat Pompeu Fabra Barcelon: *Subject syllabus – The Internet of Things.* https://www.upf.edu/pra/en/3376/22580.pdf (utoljára megtekintve: 2019.11.14)

3. University of Virginia – Angela Orebaugh, PhD: *Securing the Internet of Things.* https://collab.its.virginia.edu/syllabi/public/bcca51cc-9823-4bd4-8045-55b7b8f098cf (utoljára megtekintve: 2019.11.14)

4. Uppsala Universitet: *Syllabus for Internet of Things.* http://www.uu.se/en/admissions/master/selma/kursplan/?kKod=1DT094 (utoljára megtekintve: 2019.11.14)

5 Dr. Abonyi-Tóth Andor: *A micro:bitek felhasználási lehetőségei az oktatásban*. In: Szlávi Péter, Zsakó László (eds.) INFODIDACT 2018 ISBN: 978-615-80608-2-0

6 Módi József (2015): *A Raspberry Pi számítógép a gyakorlati oktatásban*.

7. Heizlerné Bakonyi Viktória, Illés Zoltán: *Valós idejű oktatási rendszerek*. In: Szlávi Péter, Zsakó László (eds.) INFODIDACT 2018 ISBN: 978-615-80608-2-0

8 Németh Tamás, Tornai Henrietta: *Scratch-től JavaScript-ig.* In: Szlávi Péter, Zsakó László (eds.) INFODIDACT 2018 ISBN: 978-615-80608-2-0

9 Bernát Péter: *Feladattípusorientált játékfejlesztés a Scratchben*. In: Szlávi Péter, Zsakó László (eds.) INFODIDACT 2017 ISBN: 978-615-80608-1-3

# Innovatív eszközök az algoritmusok és adatszerkezetek kurzuson

Kovácsné Pusztai Kinga

kinga@inf.elte.hu
ELTE IK

**Absztrakt**. A ma felnövő generáció már egy online világba született, ezért gondolkodásmódjuk, illetve életmódjuk gyökeresen megváltozott. A néhány évtizede még jól bevált pedagógiai módszerek egy része mára már elavulttá vált, helyükre olyan új módszert kell keresnünk, mely a Z, illetve alfa generációk szemléletéhez kapcsolódik.
A cikkemben azzal foglalkozom, hogyan lehetne az algoritmusok és adatszerkezetek kurzust úgy megújítani, hogy a hallgatók könnyebben és élvezetesebben sajátíthassák el a kurzus ismereteit.

**Kulcsszavak:** gamification, informatika oktatás, számítógépes gondolkodás, edutainment, algoritmusok

## 1. A generációs különbségek [3]

Az *X generáció* tagjait, a mai 36-50 éveseket a *„digitális bevándorló”* 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 elérhetőek és folyamatosan kapcsolatban vannak egymással az online térben. Könnyen kezelik az információk 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 generációk között napjainkban a *„generációs szakadék”* egyre gyorsabban mélyül, így egyre nehezebb feladat elé állnak a pedagógusok. 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 generációk közötti különbséget legkönnyebben az *Y generáció* tagjai, a fiatal tanárok tudják áthidalni, akik megértik mindkét generáció 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 jelentése, jellemzése

A gamification [1] szó a game (játék) és a fication (valamilyenné alakítás) szóból származik, magyarul játékosításnak, eljátékosításnak vagy gamifikációnak is szokták nevezni.

Manapság jónéhány definíció létezik, de Deterding 2011-ben alkotott definíciója [2] vált a leggyakoribbá, mely szerint a gamification **„a játéktervezési elemek használata játékon kívüli kontextusban”**.

A gamification definícióikban két nagyon fontos fogalom jelenik meg, a játékelemek és a játékmechanizmusok, melyet gyakran összevonva játéktervezési technikának is neveznek. A játékelemeken a hagyományos és videojátékokból vett eszközöket értjük, a játékmechanizmusokon a játékok működési elvének alkalmazását. Rigozky [5] (2016) a következő játékelemeket sorolta 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 természetesen csak akkor működnek hatékonyan, ha a játék mechanizmusai 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 a legnagyobb különbség, hogy valamit a játékban, a játék élvezetéért csinálunk, a játékosított alkalmazásban pedig a való életben egy előre meghatározott cél elérésének érdekében.

A gamification fogalma a köztudatban a 2000-es években jelent meg, valamint 2010-től vált népszerűvé, eredete azonban sokkal korábbra tehető. Fuchs [4] például egészen a római korig visszanyúlóan talált példákat a gamification hadászatban történő alkalmazására.

Manapság számos cég alkalmazza a gamificationt. Ki ne találkozott volna egy-egy áruházlánc hűségprogramjaival, amikor például matricákat gyűjtünk egy pontgyűjtő füzetbe. Ha a füzet betelt, akkor azt beválthatjuk egy termékre vagy egy nagyobb kedvezményre. Ennek őse az 1896-ban az S&H által megvalósított Green Stamps [7], ahol zöld bélyegeket gyűjtöttek.

Az oktatásban is számos példát találunk a gamification alkalmazására. Számos ingyenesen elérhető szoftver létezik, melyek segítségünkre lehetne az oktatás játékosításában. Ilyen app-ok például a *Socrative*, vagy a magyar nyelvű *Redmenta*, de nagyon sokan használják már az *Edmodo*-t is. Ezekkel a szoftverekkel könnyen előállíthatunk gyorstesztet, feladatot, melyet a diákok az okostelefonjukon oldanak meg, azaz a kedvenc eszközüket órai aktivitásra használják. Az ilyen app-ok alkalmazásával a diákok motiváltabbakká válnak, nő az interaktivitásuk, és sokkal gyorsabban és pontosabban kapunk visszacsatolást a tananyag megértéséről. Ezen eszközök azonban nem csak a csoport munkáját támogatják, hanem nagyon fontosak lehetnek az egyéni tanulási élményben is.

## 2.1. Edutainment

A gamification egyik érdekes alesete az, amikor különféle koncepciókat játékok segítségével tanítunk meg. Az olyan játékokat, melyek oktatási céllal készültek, edutainmentnek (az oktatás és a szórakozás szavak összekapcsolásával) nevezik. Az ilyen oktató játékok nagyon hatékonyak arra, hogy unalmas rutinfeladatokat gyakoroltassanak, (például matematikában műveletek törtekkel,) vagy nagyon bonyolult koncepciókat értessünk meg (, például egy ökoszisztéma működését). Nem már meglévő, elsősorban a szórakozás céljából készített videojátékokat használunk a tanításban, hanem kifejezetten oktatási célra készített játékokat viszünk be a tanterembe

## 2.2. Mit használhatunk fel a tanításban?

A gamification alkalmazása során átveszünk olyan elemeket a játékok rendszeréből, amelyek segítségével motiváltabbá tehetjük diákjainkat, csökkenthetjük a rájuk nehezedő stresszt, valamint segíthetünk nekik, hogy önállóbbá váljanak, és valóban részesei legyenek a tanulás során meghozandó döntéseknek. Nádory Gergely [9] a következő elemeket nevezi meg:

- **Önállóság**

  A játék során a diákok kaphatnak ugyan segítséget, de a megoldást mégis nekik kell önállóan megkeresni. Ez az önállóság, bár több időt vesz igénybe, mégis lehetőség a kísérletezésre, újratervezésre.

- **Unaloműző**

  Sokan és sokat panaszkodnak arra, hogy a diákokat manapság semmi nem érdekli, ami az iskolában történik. Ha azonban az unalmas feladatokból játékot csinálunk, a diákok sokkal szívesebben fognak részt venni az órán.

- **Célok**

  Nagyon fontos, hogy legyenek rövid-, közép-, és hosszú távú céljaink is, amikor gamifikált projektet tervezünk. Nem elég azt mondani például, hogy „el kell foglalni egy várat, és erre van 3 hónapotok", hanem folyamatosan kisebb, közelebbi célokat is meg kell határoznunk, illetve világossá kell tennünk, hogy mindez hogyan viszonyul a végső célhoz.

- **Siker és kudarc**

  A játékok alapvetően másképpen viszonyulnak a siker és kudarc kérdéséhez, mint a hagyományos iskolai értékelés, és ezt érdemes kihasználni. Míg egy rossz érdemjegy megkapása egy folyamat végzetes és végleges lezárása, addig a játék vége arra ösztönöz, hogy újból elindítsuk a játékot.

- **Azonnali visszajelzés**

  Nagyon fontos eleme a játékoknak, hogy nem kell a játék végéig várni, hogy sikerüljön eredményt elérni, hanem folyamatosan, minden kis cél elérésénél jutalmat, azaz sikerélményt kapunk. Fontos a pozitív visszajelzés, azaz, hogy nem a hibát büntetjük, hanem az erőfeszítést értékeljük. Természetesen így is el kell érnünk az eredményt a jutalom kiérdemléséhez.

# 3. Változtatások az algoritmusok és adatszerkezetek egyetemi kurzuson

Az algoritmusok és adatszerkezetek tárgy elméleti jellege, illetve korai megjelenése miatt szükségesnek találtam a gyakorlati órákon a gamification lehetőségeit használni. A változtatásokat a hallgatókkal anonim kérdőív formájában véleményeztettem is, melyet 35 hallgató töltött ki. Ez a részvétel az általam várt eredményeket jóval felülmúlta, mivel 42 hallgatót tudtam megszólítani, azaz 83%-os volt a hallgatók önkéntes részvétele, annak ellenére, hogy jutalmuk mindösszesen 2 pont volt. Mindezek a számok azt mutatják, hogy a hallgatók is látnak fantáziát a kurzus gamifikálásában.

Kezdeti lépésként a félévben behoztam a **pontrendszert** [8], azaz értékelésnél nem csak a zárthelyi eredmények számítanak. A hallgatóknak két zárthelyi dolgozatot kell írniuk, mindegyik 60 pontos. Mindkét zh-n a kötelezően elérendő minimum a 20 pont. Továbbá minden órán kapnak házi feladatot, mely megoldása nem kötelező, azonban plusz pontszerzési lehetőség. 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 ezt velünk valamilyen

formában megosztják. A félév végi jegyüket ez alapján a pontjuk összesítéséből számolom. Ha valaki nem éri el a zh-kon a kötelező minimumot, akkor pótzh-t kell írnia, ezt nem lehet egyéb pontozással kiváltani.

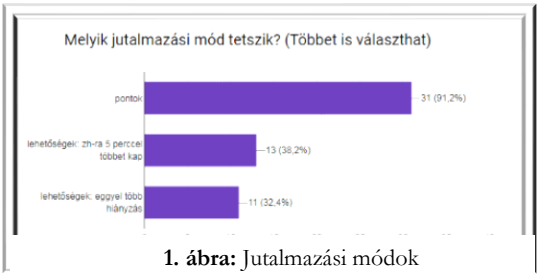
Ezen felül minden órához készítettem egy – általában játékos – edutainment alkalmazást, melyeket házi feladatként adtam fel, a honlapomról elérhető. [6] Amennyiben teljesen jól oldották meg a feladatot, egy pontot, illetve 5 perc „lehetőséget" kaptak. (Egy-egy nehezebb feladatnál megduplláztam az említett jutalmakat.) A kurzuson a zh-n összesen 120 pont érhető el, a házi feladat pontjai ehhez adódnak hozzá. Az 5 perc „lehetőség" az órai késésekre, összesítve egy óra hiányzására, illetve a zh-n több idő kapására használható fel. Elmondható tehát, hogy a jutalmazásuk elég szerény, mégis nagyon sok hallgató foglalkozott vele.

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

A „lehetőségek" jutalmazási rendszer nem volt olyan sikeres, mint a pontozás. Kevesebb, mint a hallgatók 2/3-a vélte jónak (55%) a lehetőségekkel történő jutalmazást, a többiek (45%) maradtak volna csak a pontoknál.

A hallgatók véleménye a házi feladatra szánt alkalmazásokról egyértelműen pozitív, nagyfokú részvétel (volt olyan app, melyet több mint a hallgatók 65%-a megoldott), illetve elégedettségük jellemezte (Bőven 4 fölötti átlagot kaptak a segédletek, és egy-két hallgató kivételével mindenki hasznosnak találta.) Bár nem volt kötelező véleményt írni az alkalmazásokról, mégis ezekből az általam vártnál több született. 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 klasszsnak tartom." „Ötletes, és rengeteget segít, hogy gyakorlatban is elsajátítsuk a megszerzett tudást :D"*

A hallgatók ezen véleménye a jutalmazási módok értékelésénél is meglátszott. Míg 92%-uknak tetszett a pontok jutalmazási mód, addig a „zh-n több időt kapnak" már csak 38%-uknak tetszett, és az óráról való távozás lehetősége csak a hallgatók 32%-át vonzotta. (1. ábra)




**1. ábra:** Jutalmazási módok

A kérdőív kiértékelése és a nagyarányú részvétel alapján elmondható, hogy a hallgatóknak egyértelműen tetszett a kurzus gamifikálása.

A kurzus gamifikálásán túl az órát kiegészítő segédanyagokat tettem elérhetővé a hallgatók számára a honlapomon. Ezen segédanyagok egy része az interneten elérhető youtube videó vagy animáció, melyek egy-egy algoritmus működését magyarázzák el, de természetesen saját készítésű segédanyag is található a gyűjteményben.

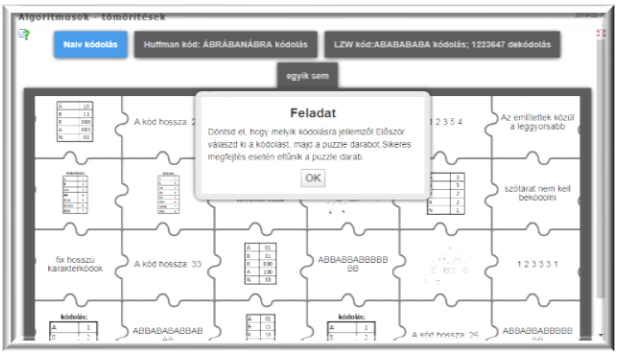
## 3.1. Algoritmusok Animációi és Vizualizációi

A www.algoanim.ide.sk honlapon számos algoritmusnak, illetve adatszerkezetnek található animációja, illetve vizualizációja. Innen kölcsönöztem a szélességi- mélységi gráfkeresés, illetve a Dijkstra algoritmus animációját.

## 3.2. Csoportos kirakó a tömörítések gyakoroltatására

A tömörítések fejezet gyakoroltatására létrehoztam Learningapps-ben egy csoportosítós kirakót (2. ábra). A kirakóban 4 *fogalomcsoport* szerepel: a *„Huffman kódolás"*, az *„LZW kódolás"*, a *„naiv kódolás"*,

illetve az *„egyik sem”*. A fogalomcsoport nevében egy feladatot is elrejtettem. A puzzle darabok az egyes „fogalomcsoportok” meghatározásai. Először ki kell választani a hallgatónak a fogalomcsoportot, majd utána a hozzá tartozó puzzle-t. Ha helyesen oldotta meg, akkor a puzzle darab eltűnik, és a háttér láthatóvá válik. A hallgatók feladata az volt, hogy találják ki, kit ábrázol a kép. A megoldás: Abraham Lempel volt.




**2. ábra:** Puzzle kezdő kép

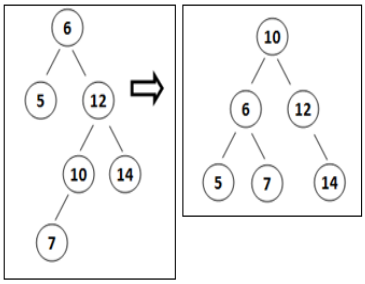
A puzzle darabok tartalmazták a fogalomcsoportok nevében elrejtett feladat megoldását, a kiszámolt kód hosszát, illetve a tömörítésekhez tartozó olyan ismereteket, amelynek tudására a vizsgán lesz szükségük.

A feladat természetesen megoldható gondolkodás nélküli kattintgatással is, de sokkal könnyebb gondolkodva, mivel ebben az esetben a 24 puzzle darab 4 fogalomcsoporthoz illesztése elég sok „próbálgatással” jár.

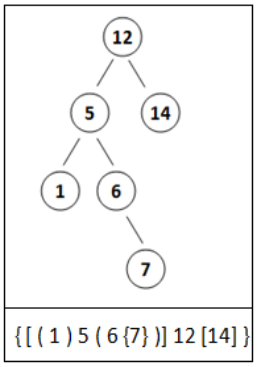
A szerény jutalmazás ellenére a hallgatók nagyfokú érdeklődést mutattak a feladat iránt, a részvétel meghaladta a 65%-ot. (49 hallgatóból 32 oldotta meg.) Az online kérdőívet 22 hallgató töltötte ki. Értékelésük alapján elmondható, hogy tetszett nekik a feladat: 5 fokú skálán 4.32-re értékelték, illetve hasznosnak is találták. Egy hallgató kivételével mindenki úgy nyilatkozott, hogy segítette őket a tanulásban.

## 3.3. Párosító játék az AVL fa műveleteinek gyakoroltatására

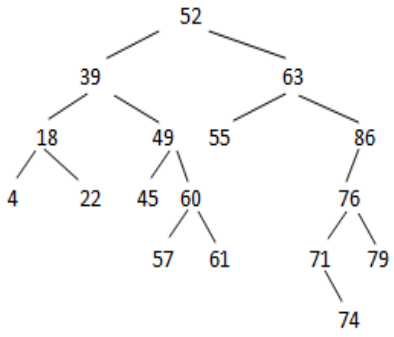
Az AVL fák műveleteinek gyakorlásához egy memória játékot hoztam létre a Learningapps-ben. A párok többsége egy *fa forgatás előtti, illetve utáni állapotát* tartalmazza. (3. ábra) A párok másik része olyan *elméleti fogalmakat és magyarázatukat* tartalmaznak, amelyeknek ismerete szükséges lesz a vizsgán. Ilyen párok például a keresőfa, illetve az AVL fa definíciója, az AVL fa magasságának meghatározása, AVL fa szöveges leírása (4. ábra), valamint az AVL fa és keresőfa közötti kapcsolat és különbség szemléltetése (5. ábra). Törekedtem arra, hogy minél több képet, ezáltal példát lássanak; ezzel elősegítve az elméleti anyag vizualizálását.




**3. ábra:** (++,-) forgatás

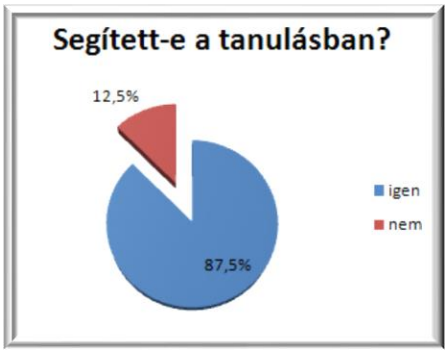



**4. ábra:** AVL fa zárójeles leírása




**5. ábra:** Keresőfa, de nem AVL fa

A hallgatói részvétel ennél a feladatnál is jóval magasabb volt, mint egy átlagos szorgalmi feladat megoldásánál. *49* hallgatóból *21* foglalkozott a feladattal (*43%*), és *20* jól oldotta meg. Az online kérdőívet *24* hallgató töltötte ki, *4.375*-re értékelték, és *87,5%*-ban (12 igen és 3 nem) találták hasznosnak a feladatot. (6. ábra) Ez a feladat kapott a legtöbb szöveges véleményt is, amelyek nagyon hasznosak voltak számomra, többek között tartalmaztak konstruktív javaslatot is. Néhány példa a szöveges vélemények közül: *„Nagyon jó, és inspiráló! Egyszer csak azon kaptam magam, hogy már a 10-edik AVL fát rajzolom a füzetembe. Úgy sikerült begyakorolni az órán vett műveleteket, hogy szinte fel sem tűnt, hogy tanulok.” „Tényleg nagyon jól lehet gyakorolni az anyagot a játékokkal, nekem nagyon tetszik ez a módszer :D” „Nagyon tetszett, hogy amíg tovább nem lépett az ember, a puzzle mutatta, mi van rajta, így rendesen végig lehetett gondolni.” „Sokat kellett görgetni és ez megnehezítette és kevésbé élvezetessé tette a feladatot. Esetleg több kisebb (3x3, 4x4) feladatra szét lehetne osztani. Maga az ötlet, hogy memóriajáték, az tetszik.”*

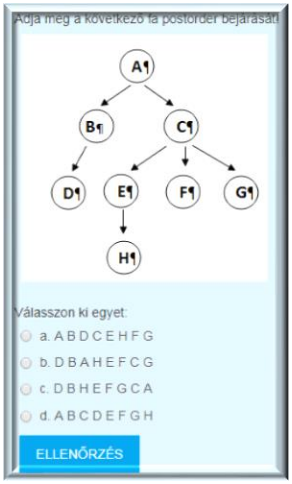



**6. ábra:** Hallgatói vélemény az alkalmazásról

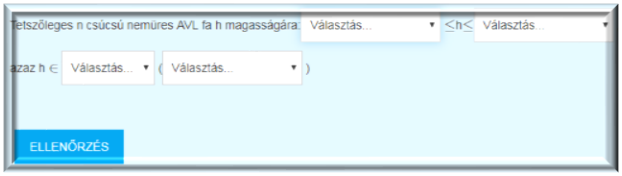
## 3.4. Moodle tesztek a fákkal kapcsolatos ismeretek elmélyítésére

A fák gyakoroltatására két moodle tesztet hoztam létre. Az első (AA_3 nevű) 6 különböző típusú feladatból állt. Két beépített választós feladat az AVL fa magasságát, illetve az AVL fa főbb műveleteinek idejét kérdezte (7. ábra), két feleletválasztós feladat az általános fa pre- illetve postorder bejárásával foglalkozott. (8. ábra) Ezen felül volt egy párosítós feladat, mely az általános fa ábrázolási módjait kérte számon, valamint egy kiegészítő kérdéses feladat, amely egy általános fa szöveges reprezentációjáról szólt. A teszt teljesen jó megoldására összesen 2 pontot lehetett kapni.

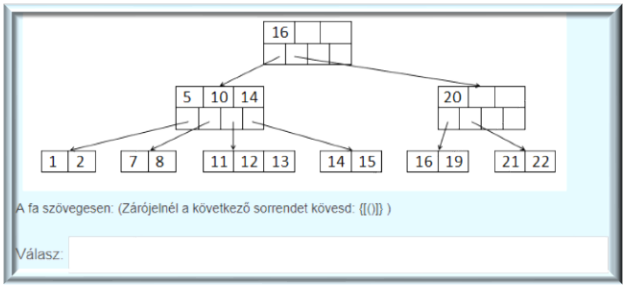
A másik moodle teszt (AA_4 nevű) 10 kérdésből állt a B+ fa témaköréből. Egy kiegészítő kérdős feladat foglalkozott a B+ fa ábrázolásával (9. ábra), két feleletválasztós kérdés szólt a B+ fa műveleteiről (keresés, beszúrás), illetve 7 igaz-hamis állítás mélyítette el a B+ fáról tanult ismereteket. Ennek a tesztnek a megoldása 1 pontot ért.




**8. ábra:** Feleletválasztós feladat: Általános fa postorder bejárása.



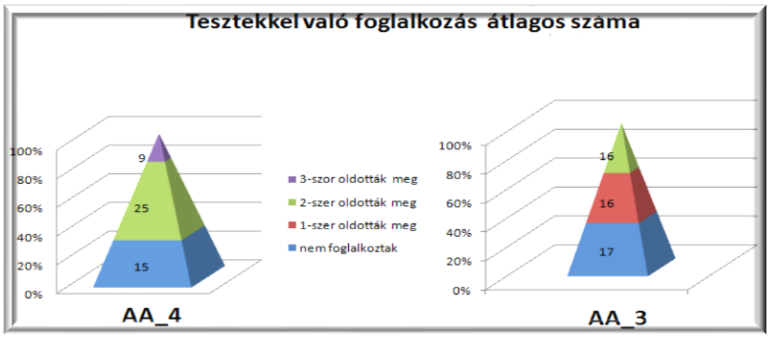

**7. ábra:** Beépített válaszos feladat: AVL fa magasságának műveletideje




**9. ábra:** Kiegészítő kérdős feladat:B+ fa ábrázolása

A hallgatói részvétel, illetve a véleményük a feladatról számomra megdöbbentőek voltak. Mivel ezek a tesztek voltak a félév során a legkevésbé látványosak, és a legkevésbé játékosak, azt vártam, hogy ezek a feladatok lesznek a legkevésbé kedveltek, illetve ezekkel fog a legkevesebb hallgató foglalkozni. Ezzel szemben azonban a diákok másképp vélekedtek, a hallgatói részvétel ezeknél a feladatoknál is kiemelkedően magas volt, az első tesztet *49*-ből *32*-en (*65%*), a második tesztet *34*-en (*69%*) oldották meg. Mivel célom nem a számonkérés, hanem a tanultak elmélyítése volt, egy hallgató akárhányszor megoldhatta bármelyik tesztet. A moodle egyik előnye a feladat automatikus naplózása, így meglepődve tapasztaltam, hogy az első tesztre összesen 48, a másodikra pedig 77 próbálkozás született. (Ez személyenként átlag 1,5, illetve 2,26 próbát jelent.) (10. ábra)

Az első feladat kérdőívét 9-en, a második teszt kérdőívét 8-an töltötték ki. Ezek ugyan kisebb részvételi arányok a többinél, ami betudható annak, hogy a félév vége felé fogy a hallgatók lelkesedése, illetve nőnek az egyéb beadandó feladatai. A hallgatók az első tesztet átlagosan 4,44-re, a másodikat átlagosan 4.375-re értékelték. Mindkét feladatot 100%-ban hasznosnak találták. Ezekre a feladatokra kevés szöveges vélemény érkezett, ezek közül az egyik így szól: *„Tetszett, mert át kellett hozzá néznem a mai anyagot. Továbbá az is jó, hogy nem volt idő limit a moodle teszten, így sokkal sokkal nyugisabb az egész."*

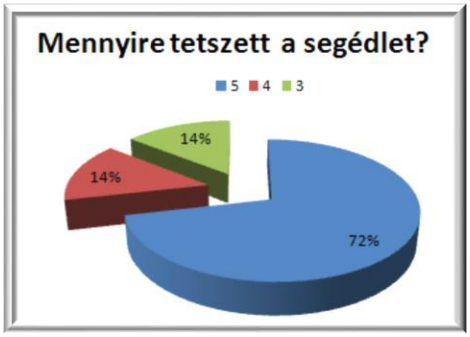



**10. ábra:** A hallgatói átlagos foglalkozás a moodle tesztekkel

## 3.5. Keresztrejtvény a gráf alapfogalmainak gyakoroltatására

A gráf alapfogalmainak gyakorlására létrehoztam egy keresztrejtvényt a HotPotatoes-ban. A keresztrejtvény összesen 15 fogalmat tartalmaz, 9-et vízszintesen, 6-ot pedig függőlegesen. Az egyes elemek kitöltésekor lehetőségünk van segítséget kérni, amely egy betű megmutatását jelenti. A 12. ábra a kitöltött keresztrejtvényt mutatja.

A kérdőívet 15 hallgató töltötte ki, átlagosan 4,4-re értékelték (11. ábra), és mindenki úgy nyilatkozott, hogy az alkalmazás segítette őt a tanulásban. A szöveges véleményekből egyértelműen kiderül, hogy a hallgatóknak tetszett a segédlet. (*„Ez volt az eddigi legjobb segédlet! :)”*)




**11. ábra**: Hallgatói vélemény az alkalmazásról

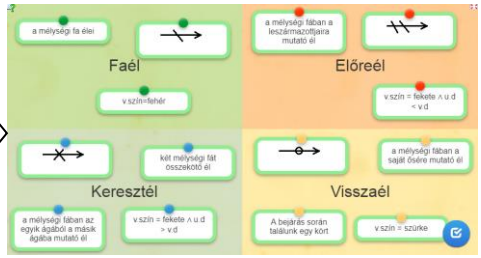

**12. ábra**: Kitöltött keresztrejtvény

## 3.6. Csoportba rendezős játék „az irányított gráf éleinek az osztályozása a mélységi keresés után” témakör gyakoroltatására

Az élek osztályozásának, illetve az egyes osztályok tulajdonságainak gyakoroltatására létrehoztam egy csoportba rendezős játékot a LearningApps-ben. A mélységi bejárás után az irányított gráf éleit négy osztályba soroljuk: faél, előreél, visszaél, keresztél. A játék kezdetén ez a négy csoport különböző színben jelenik meg (13. ábra). Középen fogjuk kapni az egyes meghatározásokat, amelyeket a megfelelő csoportba kell húznunk. Ha elfogytak a kártyák, (vagy ha abba akarjuk hagyni a játékot,) akkor

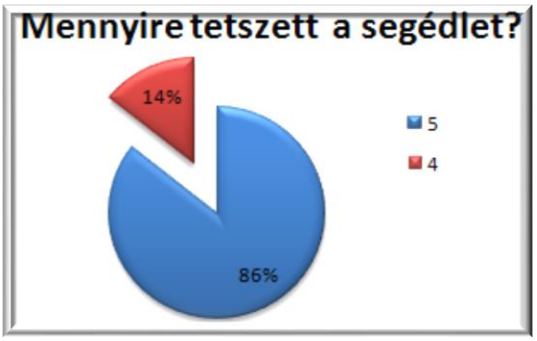
a képernyő jobb alsó sarkában található kék körben lévő pipára kattintva ellenőrizhetjük a megoldásunkat. A helyes megoldások zöld keretet kapnak, a helytelenek pirosat. (14. ábra)


**13.** ábra**:** Játék kezdete




**14. ábra:** Játék vége

Mivel ez a feladat a félév vége felé került kiadásra, a hallgatók lelkesedése fogyott, így már csak 16-an oldották meg ezt a feladatot, de 14-en véleményezték is. A hallgatóknak ez a segédlet tetszett a legjobban, átlagosan 4,86-ra értékelték (15. ábra), mindenkit segített a tanulásban, valamint 6 pozitív szöveges véleményt is kapott. Ezekből néhány: *„Nagyon kreatív játék, tetszett”*, *„Eddig ez a játék tetszett a legjobban!”*, *„szép színes volt, a barátnőmnek tetszett”*, *„Zhra készüléshez kiváló volt!”*.




**15. ábra:** Hallgatói vélemény az alkalmazásról

## 4. Összegzés

Rohamosan változó világunkban a tanárok sem tudnak megmaradni a hagyományos módszereknél, ha sikeresen szeretnének tanítani. Mivel a ma felnövekvő generáció már egy online világba született, az oktatásnak is nyitnia kell az okos eszközök felé. A cikkemben egy egyetemi kurzus innoválásával foglalkoztam. A kurzus megújításánál két lényeges dolgot követtem, kiegészítettem az órát online videókkal, animációkkal, illetve „gamifikáltam” a kurzust (, azaz pontrendszert vezettem be, kibővítettem a jutalmazási rendszert „lehetőségek” szerzésével, valamint kibővítettem a pontszerzési lehetőségeket szorgalmi feladatokkal, programírással, kutatómunkákkal). A legjelentősebb változtatás az volt, hogy minden órához készítettem egy edutainment alkalmazást, melynek célja a tanult ismeretek elmélyítése, egy otthoni játékos környezetben. Mindezen változtatásokat a hallgatókkal anonim online kérdőív formájában véleményeztettem.

A hallgatók magas részvétele, illetve válaszuk egyértelműen igazolták, hogy még a felsőoktatásban is helye van az oktatás gamifikálásának, illetve a tananyag vizualizálásának.

## Köszönetnyilvánítás

Az EFOP-3.6.3-VEKOP-16-2017-00002. számú projektben elvégzett szakmai feladat az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

## Irodalomjegyzék


1. *Gamification,* wikipedia
   https://hu.wikipedia.org/wiki/Gamification (utoljára megtekintve: 2019. 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: 2019. 10. 19.)
3. 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: 2019. 10. 19.)
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: 2019. 10. 19.)
5. 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: 2019. 10. 19.)
6. *Algoritmusok és adatszerkezetek 2.* kurzus honlapja
   https://people.inf.elte.hu/kinga/algoritmusok2/seged.htm (utoljára megtekintve: 2019. 10. 19.)
7. 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: 2019. 10. 19.)
8. 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: 2019. 10. 19.)
9. Nádori Gergely: *Gamification* (2012) in PIL Akadémia 7
   http://tanarblog.hu/attachments/3010_7_gamification.pdf (utoljára megtekintve: 2019. 10. 19.)
10. Kovácsné Pusztai Kinga: *Játékosítás (gamification) az* oktatásban (2018) In: Infodidact 2018.
11. Kovácsné Pusztai Kinga: *Edutainment is Education* (2019) In: XXXII. DidMatTech 2019.

# Az ismétlés áttekintése

Menyhárt László Gábor

menyhart@inf.elte.hu
ELTE IK

**Absztrakt.** Az ismétlés hétköznapi fogalma és az informatika általi használat, illetve a különböző implementációk eltérnek. Ebben a cikkben összeszedem a különböző szintek, paradigmák és nyelvi lehetőségeket, majd ismertetem az elvégzett mérésemet, végül didaktikai szempontok szerint is értékelem az eredményeket. Így hasznos programozó versenyre készülő diákok, illetve az őket felkészítő informatika tanárok számára.

**Kulcsszavak:** ismétlés, rekurzió, ciklus, programozási nyelvek, mérés

## 1. Bevezetés

Hétköznapi értelemben az ismétlés természetes jelenség: Gondoljunk itt a napok, hetek, évszakok stb. rendszeres váltakozására. Ugyanakkor az informatikában többféleképpen jelenik meg. Máshogy gondolkozhatunk, máshogy fordul elő a különböző paradigmákban, a különböző nyelveken más a szintaxis és máshogy lett implementálva, ezért a futásuk is különböző, vagyis a futásidejük is nagyban eltér. Ezen észrevételek és benyomások ösztönöztek arra, hogy gyűjtsem össze a jelenlegi lehetőségeket és hasonlítsam össze őket.

Ebben a cikkben a fogalmi meghatározásoktól kezdve, különböző implementációk futtatásának mérésén keresztül a didaktikai megfontolásokig áttekintem az eredményeket. A bemutatásra kerülő módszer és a jelenlegi eredmények hasznosak lehetnek programozó versenyre készülő diákok, illetve az őket felkészítő informatika tanárok számára.

## 2. Az ismétlés fogalma

Az ismétlés valaminek a többszöri elvégzése, végrehajtása. Összegyűjtöttem pár meghatározást is különböző helyekről, annak ellenére, hogy mindenki tudja, érzi, hogy mit jelent.

A Magyar Szinonima kéziszótár-ban az ismét és az ismétel szavak találhatóak meg:

„***ismét*** *újra, megint | szintén, ismételten*” [1/140. o.]

„***ismétel*** *mondogat, hajtogat, csépel| szajkóz | megismétel, elismétel, visszamond| reprodukál, megrepetál | próbál, gyakorol | folytat*” [1/140. o.]

Az Új Magyar Lexikon-ban az ismétlés meghatározása az, hogy az „***ismétlés : 1.*** *(nev) oktatási eljárás ; … Osztály~: …* ***2.*** *(nyelvt) … A magyarban jelölheti a cselekmény hosszan tartó voltát (pl. ment-ment) …* ***3.*** *a stilisztikában …*” [2/448. o.]

A Magyar néprajzi lexikon [3] csak nyelvi meghatározást, mondanivaló nyomatékosítására, mint költői kifejezés ír a többszörözésről.

[4]-ben az „***ismétlés*** *(fn.) … Cselekvés, melynél fogva valamit ismét, azaz újra, még egyszer teszünk. …*”

A Sulinet Tudásbázisában az Informatika tantárgynál az ismétlés fogalma „Többszöri végrehajtás”-ként van definiálva. [5]

A WikiSzótárban [6] az ismétlés egyik definíciója „**2.** Egy **cselekvés újra végzése**, amikor ugyanazt tesszük, amit korábban már megtettünk.”.

A Cambridge szótárban a **repeat** szóra az szerepel, hogy „*to happen, or to do something, more than once*” [7], míg az **iteration**-re az, hogy „*the process of doing something again and again, …”* [8].

## 3. Az ismétlés az informatika szintjein

Amikor egy konkrét megoldandó problémánk van, még csak azt tudhatjuk, hogy valamit többször kell majd végrehajtani. Gondolkozásunk következő absztrakt szintjén, már azt döntjük el, hogy egy műveletet, amit többször szeretnénk elvégezni azt kiszervezzük, majd meghívjuk önmagát, ez a rekurzió [10], vagy az adott műveletet ciklikus hívással ismételjük. Ez a megkülönböztetés már paradigmaként is megjelenik, hiszen a funkcionális programozás csak az előbbire ad lehetőséget.

A cikluson belül megkülönböztetünk feltételest és adott elemszámszor lefutót.

Az adott elemszámszor lefutónál két nagy csoportot különböztethetünk meg. Az egyikben egy ciklusváltozó, mint számláló értéke változik az ismétlés során, míg a másikban egy konkrét elem kap értékeket. Az előbbiben tehát egy sorozat elemein úgy szaladhatunk végig, hogy egy számláló növekszik egyesével és azzal indexeljük a sorozat elemeit a feldolgozáshoz, míg a másodikban rögtön a sorozat egyes elemeit kapjuk és dolgozzuk fel.

A feltételes ciklusokat is további két nagy csoportra oszthatjuk, az elöltesztelő és a hátultesztelő verzióra. Ezek között a különbség, hogy a hátultesztelő ciklus műveletei egyszer biztosan lefutnak, míg az elöltesztelőnél elképzelhető, hogy a műveletei egyszer sem hajtódnak végre. Mindkettő feltételeként egy logikai kifejezést adunk meg, amitől függ, hogy az műveleteket ismét végre kell-e hajtani vagy sem. Megkülönböztetjük a ciklusokat aszerint is, hogy az igaz logikai kifejezésnél mi történik. Azaz, hogy ilyenkor a műveletek ismét lefutnak vagy pedig pont ilyenkor áll le a ciklus futása. Tehát, hogy a bent maradás vagy a kilépés feltételét kell-e megadni.

<table>
<tr><th colspan="6">Ismétlés</th></tr>
<tr><td rowspan="4">Rekurzió</td><td colspan="5">Ciklus</td></tr>
<tr><td colspan="4">Feltételes</td><td colspan="2">Adott elemszámszor futó</td></tr>
<tr><td colspan="2">Elöltesztelő</td><td colspan="2">Hátultesztelő</td><td rowspan="2">Számláló</td><td rowspan="2">Sorozat elemein iteráló</td></tr>
<tr><td>bent marad</td><td>kilép</td><td>bent marad</td><td>kilép</td></tr>
</table>

**1. táblázat:** Ismétlés áttekintő táblázata

A szintaxis bezavarhat ebbe a tiszta képbe, ugyanis a legtöbb programozási nyelv, a fenti lehetőségek közül valamelyiket nem támogatja. Például a Pascalban a kilépés feltételét kell megadni a hátultesztelő ciklusnak, míg a C típusú nyelveknél a bent maradás feltételét. Az elöltesztelő ciklusnál általában a bent maradás feltételét kell megadni, de a Linux-os shell scriptben van szintaktika a kilépés feltételének megadásához is. Sőt, shell scriptben mind a négyféle feltételes ciklust le tudjuk írni. C-ben a `for` kulcsszót lehet használni elöltesztelő cikluskénti is és ott lehetőség van a `break` illetve `continue` kulcsszavakkal az elvileg adott elemszámszor futó `for` ciklus idő előtti befejezésére is. Persze a több egymásba ágyazott `for` ciklus esetén már trükközni kell, ezért érdemes elkerülni ezt a lehetőséget. Van olyan programozási nyelv is (például a Logo), ahol a ciklusváltozó meg sem jelenik, csak ha szükség van rá, akkor lehet használni (a REPEAT/ismétlés-ben a REPCOUNT/hányadik).

A magasszintű programozási nyelvekben nem csak a kulcsszavak, azaz a szintaxis térhet el, hanem különbözőféleképpen implementálják alacsony szintre, vagyis a ténylegesen futó gépi kódra a fenti lehetőségeket. Így természetesen a futásidejük is eltér. Mivel több nyelven is implementáltam

[11, 12, 13, 14, 15, 16] ugyanazon feladatok megoldását és érzésre is különböző volt a futásidejük, így felmerült bennem, hogy végezzek el egy pontosabb mérést.

# 4. A mérés ismertetése

Hat különböző programozási nyelven az ismétlés öt-hat féle implementációját készítettem el. Majd a futási időket milliszekundumban mérve összegyűjtöttem.

A nyelvek a következők voltak: C++, C#, Java, JavaScript, Pascal és Python.

Implementáltam a rekurziót, elől tesztelő feltételes ciklust, számlálós ciklusok közül pedig a ciklusváltozós for-on kívül olyan iterációs változatokat, amiket az adott nyelv támogat. Párszor előfordult, hogy a nyelv nem támogatja az adott módszert, így ott az kihagyásra került. Az elemeken való végig iterálás is többféleképpen fordul elő, így ezeket egy közös csoportba vontam össze.

## 4.1. A környezet bemutatása

A mérést egy Intel® Core™ i7-8750H 2.20GHz-es processzorú, 32GB memóriával rendelkező számítógépen futtattam, melyen 64 bites Windows 10 Pro operációs rendszer futott.

A C++ fordítója a Code::Blocks 17.12-es alapértelmezett mingw32-g++.exe-je volt, aminek a verziója 5.1.0.

A C#-hoz a Windows 10-en megtalálható .NET Framework fordítóját használtam, a „Microsoft (R) Visual C# Compiler version 4.8.3752.0”-t.

A Java-hoz a jelenleg legfrissebb 2019. október 6-án megjelent OpenJDK-t.

```
>java -version
openjdk version "13.0.1" 2019-10-15
OpenJDK Runtime Environment (build 13.0.1+9)
OpenJDK 64-Bit Server VM (build 13.0.1+9, mixed mode, sharing)
```

JavaScript kódokat a Node v10.16.0 segítségével futtattam.

Pascalhoz a 32 bites „Free Pascal Compiler version 3.0.4 [2017/10/06] for i386”-em volt.

A Python kódokat a „Python 2.7.15 (v2.7.15:ca079a3ea3, Apr 30 2018, 16:30:26) [MSC v.1500 64 bit (AMD64)]” értelmezte és futtatta.

## 4.2. Az implementált algoritmus bemutatása

Mind a hat programozási nyelven az összes lehetséges módon a következő algoritmus szerint implementáltam az ismétlést.

Mivel kiderült, hogy a rekurzió lépésszáma a vizsgált programozási nyelvekben korlátos, ezért először annak a maximális lehetséges lépésszámának a meghatározásával kezdtem. A 4.3.-as fejezetben lesznek a pontos értékek. Az így kimért legkisebb lépésszámot használtam az összes módszernél. Mivel így nagyon gyorsan lefutott a kód ezt többször kellett elvégezni, amihez az összes nyelven egy számlálós (`for`) ciklust választottam.

Az eltelt idő mérését a futás előtti és a futás utáni időpont különbségéből számoltam ki. Üres, azaz műveletet nem tartalmazó ciklust nem érdemes csinálni, ezért egy egyszerű értékadást tettem bele. Viszont engem csak a ciklusra szánt idő érdekelt, ezért az értékadásra szánt időt ki kellett találnom. Ezt pedig úgy csináltam, hogy rögtön még egyszer megismételtem ugyanazt a ciklust két értékadással is. Az absztrakt algoritmus:

```
time1
loop
  assignment1
time2
```

```
loop
  assignment1
  assignment2
time3
```

Kiszámoltam, hogy mennyi az egy értékadásra szánt teljes idő (1) és feltettem, hogy az értékadások ugyanannyi idő alatt futnak le (2). A ciklusra szánt konkrét időt (3) tehát így számoltam ki:

$$T_{a2} = (t_3 - t_2) - (t_2 - t_1) \quad (1)$$

$$T_{a1} = T_{a2} \quad (2)$$

$$T_l = t_2 - t_1 - T_{a1} \quad (3)$$

Az implementált kódok az A. mellékletben találhatóak.

## 4.3. Mérési eredmények

Először meghatároztam a különböző nyelveken történő rekurzív megvalósítás alapján egy lépésszámot, azt a legnagyobbat, amelyik mindegyiken elfut. A rekurzív függvény implementálásakor az utasítások elejére tettem a rekurzív hívást, hogy a fordító biztosan ne optimalizálja a farok-rekurziót.

| Programozási nyelv | MaxCnt<br>(Ebben a környezetben, ezen programok esetében) |
|---|---|
| C++ | 56988 |
| C# | 15918 |
| Java | 11420 |
| JavaScript | 8940 |
| Pascal | 65134 |
| Python | 4194 |

**2. táblázat:** Rekurzív hívások maximális száma

Végül 4000 ismétlést egy számlálós ciklussal, azaz az implementációkban `for` szintaktikával 10.000-szer megismételtem, hogy mérhető és összehasonlítható eredményt kapjak. Később kiderült, hogy ez jó választás volt, mert ez az egyik leggyorsabb, így nem zavar bele annyira az összehasonlításba. Így kétszer 40.000.000 szer futott minden mért ismétlés.

## 4.4. Mérés kiértékelés

A B. mellékletben elérhető nyers adatokból Excel segítségével átlagokat és szórást számoltam. Az időt ($T_l$) milliszekundumban írattam ki és az első összesen 40 millió ismétlésre vonatkozik. A szórást az átlag értékhez viszonyított százalékos értékkel adtam meg, hogy könnyebben érthető és összehasonlítható legyen. Az eredmények a következő táblázatban láthatóak:

| Programozási nyelv | While | For | ForEach [+lambda] | ForEach +named fnc | Rekurzív |
|---|---|---|---|---|---|
| C++ | 7 (127%) | 6 (113,9%) | 1142 (8,8%) | 1141 (12,7%) | 135 (17,7%) |
| C# | 20 (46,8%) | 24 (55,1%) | 117 (43%) | 113 (69,2%) | 90 (10,2%) |
| Java | 54 (42,2%) | 50 (21,6%) | 234 (22%)*<br>170 (18,7%) | 2020 (12,4%) | 109 (15,9%) |
| JavaScript | 36 (37,5%) | 41 (68,1%) | 254 (18,6%) | 242 (14,7%) | 267 (9,2%) |
| Pascal | 65 (10,1%) | 89 (36,5%) | 565 (8,6)* | notSupported | 138 (11,3%) |
| Python | 2054 (28,8%) | 1156 (55,3%) | 460 (95,4%)* | 3589 (20,8%) | 9876 (15,4%) |

**3. táblázat:** Mérési eredmények: átlag (szórás az átlaghoz viszonyítva %-ban)

### 4.5. Az eredmény véleményezése

A táblázatból látszik, hogy az elöltesztelő ciklus majdnem mindig a leggyorsabb, de az adott lépésszámú ciklusváltozós megoldás is elhanyagolható lassulást okozott három nyelvnél, míg a Python esetében kétszer gyorsabb. Sebességben a rekurzív hívás követi az előzőeket, ami Pascalban és Javaban 2-szer, C#-ban, JavaScriptben és Pythonban 4-5-ször, C++-ban pedig körülbelül 20-szor lassabb, mint az előzőek átlaga. Az eddigi idők öt programozási nyelvnél a másodperc töredéke, míg Pythonban már 10 másodperc körüli eredményt mutat. A sorozat elemein történő végig iterálás tekintetében a C# és JavaScript összemérhető a rekurzív hívás idejével, de a C++ 9-szer lassabb a rekurzív hívásnál, míg a Java 2-szer vagy 18-szor lassabb, attól függően, hogy anonymous vagy elnevezett függvényt használunk. Míg Pythonban a számlálós ciklushoz képest „csak" 3-szoros lassúlást, de a rekurzív híváshoz képest gyorsulást mértem. Az ismétlés a C++ számlálós ciklusával a leggyorsabb, míg a Pythonban a rekurzív megvalósítása a leglassabb.

## 5. Didaktikai megfontolások

Ha oktatáshoz a fenti táblázat alapján kellene nyelvet választanom, akkor a C#-ot választanám, mert az egyenletesen hoz elég jó időket. A Pascal-t nem választanám, mert nem támogat mindent. A Python meg nagyon lassú. Tehát a sorrend C#, JavaScript, C++ és Java lenne az ismétlés különböző implementációinak futási ideje és azok szórása alapján.

Természetesen mindegyik lehetőséget meg kell tanítanunk hallgatóinknak és meg kell mutatnunk, hogy mi alapján tudnak választani a paradigmák, szintaxisok között.

Vannak esetek, amikor a kód olvashatósága a fontosabb, de előfordul, hogy a futási sebességet kell előtérbe helyezni. Például már többen megállapították [9], hogy a rekurzív eljárások többnyire kevésbé hatékonyak, mint az iterációt használók, de gyakran egyszerűbb, könnyebben olvasható leírást kapunk. Egy ilyen összehasonlító táblázat hasznos lehet olyan programozási versenyeken is, ahol időlimitet használnak és a versenyző választhatja meg a nyelvet. Habár ott legtöbbször nem csak egy értékadás a művelet és a többi nyelvi elem is befolyásolhatja az futás összes idejét.

A cikkemben megtalálható tudás és mérési eredmények segíthetik az adott feladathoz történő megfelelő implementáció kiválasztását. A mérés technikája pedig később jól jöhet, hiszen a jövőben a processzorok fejlődése, illetve az újabb verziójú fordítók megjelenése változtathat az itt ismertetett időkön, vagyis újra el kell végezni ezt a Benchmark-ot [17]. Hiszen az újabb fordító verziók a kódok elemzésével egyre hatékonyabb bináris kódokat hozhatnak létre, akár a kódok különböző mértékű változtatásával. Például a Python-ból már most is elérhető a 3.8-as, sőt több alternatív implementáció is létezik hozzá [18, 19], amiket még érdemes lesz összehasonlítani.

## Irodalom


1. Bék Gerzson, Csiffáry Tamás: *Magyar szinonima kéziszótár*, Könyvmíves Könyvkiadó, Budapest (2004)
2. Akadémiai kiadó szerkesztősége: *Új magyar lexikon*, 3 G-J, Nyolcadik, változatlan lenyomat, Akadémiai Kiadó, Budapest (1962)
3. *Magyar néprajzi lexikon*, Akadémiai Kiadó, Budapest (1977-1982)
4. Czuczor Gergely, Fogarasi János: *A magyar nyelv szótára*, Emich Gusztáv magyar akadémiai nyomdásznál Pest (1862)
   https://czuczor.oszk.hu/kereses.php?kereses=ism%C3%A9tl%C3%A9s (utoljára megtekintve: 2019.11.11.) és
   https://mek.oszk.hu/05800/05887/pdf/3kotet_1.pdf (74. oldal, utoljára megtekintve: 2019.11.11.)
5. *Ismétlés fogalma a Sulinet tudásbázisban* (2016)
   https://tudasbazis.sulinet.hu/hu/informatika/informatika/informatika-2-evfolyam/algoritmusok-a-hetkoznapokban-tevekenysegek-elemekre-bontasa-helyes-sorrend-megkeresese/ismetles-szerepe (utoljára megtekintve: 2019.11.11.)
6. *WikiSzótár* (2012)
   https://wikiszotar.hu/ertelmezo-szotar/Ism%C3%A9tl%C3%A9s (utoljára megtekintve: 2019.11.11.)
7. *Repeat in Cambridge Dictionary* (2019)
   https://dictionary.cambridge.org/dictionary/english/repeat (utoljára megtekintve: 2019.11.11.)
8. *Iteration in Cambridge Dictionary* (2019)
   https://dictionary.cambridge.org/dictionary/english/iteration (utoljára megtekintve: 2019.11.11.)
9. Rónyai L., Ivanyos G., Szabó R.: *Algoritmusok*, Typotex (2004) 37. oldal
10. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest.: *Introduction to Algorithms*, MIT Press (1990) (magyar verzió: Algoritmusok, Műszaki Könyvkiadó 1997)
11. *C++ referencia* (2019)
    https://en.cppreference.com/w/ (utoljára megtekintve: 2019.11.11.)
12. *C# referencia* (2017)
    https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/ (utoljára megtekintve: 2019.11.11.)
13. *Java documentation* (2019)
    https://docs.oracle.com/javase/tutorial/java/ (utoljára megtekintve: 2019.11.11.)
14. *JavaScript reference* (2019)
    https://developer.mozilla.org/hu/docs/Web/JavaScript/Reference (utoljára megtekintve: 2019.11.11.)
15. *Free Pascal reference guide* (2017)
    https://www.freepascal.org/docs-html/ref/ref.html (utoljára megtekintve: 2019.11.11.)
16. *The Python Language Reference* (2019)
    https://docs.python.org/2.7/reference/index.html (utoljára megtekintve: 2019.11.11.)
17. *Becnhmark jelentése*
    https://pcforum.hu/szotar/Benchmark (utoljára megtekintve: 2019.11.11.)
18. *Alternative Python Implementations*
    https://www.python.org/download/alternatives/ (utoljára megtekintve: 2019.11.11.)
19. *Cython*
    https://cython.org/ (utoljára megtekintve: 2019.11.11.)

# Mellékletek

## A. Forráskódok

### A.1. C++

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <array>
#include <chrono>

using namespace std;

int a;
int b;

void recursiveFor(int &i, const int &N, std::array<int,65135> &v) {
    if (i<N) {
        i++;
        recursiveFor(i,N,v);
        a=v[i-1];
    }
}

void recursiveFor2(int &i, const int &N, std::array<int,65135> &v) {
    if (i<N) {
        i++;
        recursiveFor2(i,N,v);
        a=v[i-1];
        b=v[i-1];
    }
}

int main()
{
    auto timer_start= std::chrono::high_resolution_clock::now();
    auto timer_mid= std::chrono::high_resolution_clock::now();
    auto timer_end= std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed;
    std::chrono::duration<double> elapsed2;

    const int MaxN=65135;
    int N=4000;
    int times=10000;

    std::array<int,MaxN> v;
    for (int i=0; i<N; i++)
    {
        v[i];
    }

    auto func=[](const int&x)
    {
        a=x;
```

```
    };

     auto func2=[](const int&x)
    {
        a=x;
        b=x;
    };

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        while (i<N)
        {
            a=v[i];
            i++;
        }
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        while (i<N)
        {
            a=v[i];
            b=v[i];
            i++;
        }
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;while;cnt:"<< times*N <<";lt:" << elap-
sed.count()*1000-(elapsed2.count()*1000-elapsed.count()*1000) <<
"\n";

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        do
        {
            a=v[i];
            i++;
        } while (i<=N);
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        do
        {
            a=v[i];
            b=v[i];
            i++;
```

```
        } while (i<=N);
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;do-while;cnt:"<< times*N <<";lt:" << elap-
sed.count()*1000-(elapsed2.count()*1000-elapsed.count()*1000) <<
"\n";

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        for (int i=0; i<N; i++)
        {
            a=v[i];
        }
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        for (int i=0; i<N; i++)
        {
            a=v[i];
            b=v[i];
        }
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;for;cnt:"<< times*N <<";lt:" << elap-
sed.count()*1000-(elapsed2.count()*1000-elapsed.count()*1000) <<
"\n";

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        std::for_each(v.begin(), v.end(),
                      [](const int&x)
        {
            a=x;
        });
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        std::for_each(v.begin(), v.end(),
                      [](const int&x)
        {
            a=x;
            b=x;
        });
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
```

```
    std::cout << "Cpp;ForEach + lambda;cnt:"<< times*N <<";lt:" <<
elapsed.count()*1000-(elapsed2.count()*1000-elapsed.count()*1000) <<
"\n";

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        std::for_each(v.begin(), v.end(), func);
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        std::for_each(v.begin(), v.end(), func2);
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;ForEach + named function;cnt:"<< times*N
<<";lt:" << elapsed.count()*1000-(elapsed2.count()*1000-
elapsed.count()*1000) << "\n";

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        recursiveFor(i,N,v);
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        recursiveFor2(i,N,v);
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;Recursive;cnt:"<< times*N <<";lt:" << elap-
sed.count()*1000-(elapsed2.count()*1000-elapsed.count()*1000) <<
"\n";

    return 0;
}
```

### A.2. C#

```
using System;

public class Program
{
    static int a;
    static int b;

    static void recursiveFor(int i, int N, int[] v) {
        if (i<N) {
            i++;
```

```
        recursiveFor(i,N,v);
        a=v[i-1];
    }
}

static void recursiveFor2(int i, int N, int[] v) {
    if (i<N) {
        i++;
        recursiveFor2(i,N,v);
        a=v[i-1];
        b=v[i-1];
    }
}

public static void Main()
{
DateTime timer_start= DateTime.Now;
DateTime timer_mid= DateTime.Now;
DateTime timer_end;
TimeSpan elapsed;
TimeSpan elapsed2;

int N=4000;
int times=10000;

int[] v=new int[N];
for (int i=0; i<N; i++)
{
    v[i]=i;
}

Action<int> func = x => {
    a=x;
};
Action<int> func2 = x => {
    a=x;
    b=x;
};

timer_start= DateTime.Now;
for (int j=0; j<times; j++)
{
    for (int i=0; i<N; i++)
    {
        a=v[i];
    }
}
timer_mid= DateTime.Now;
for (int j=0; j<times; j++)
{
    for (int i=0; i<N; i++)
    {
        a=v[i];
        b=v[i];
    }
```

```
    }
    timer_end = DateTime.Now;
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    Console.WriteLine("cs;for;cnt:{1};lt:{0}",elapsed.Milliseconds-
(elapsed2.Milliseconds-elapsed.Milliseconds),N*times);

    timer_start= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        int i=0;
        while (i<N)
        {
            a=v[i];
            i++;
        }
    }
    timer_mid= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        int i=0;
        while ( i<N)
        {
            a=v[i];
            b=v[i];
            i++;
        }
    }
    timer_end = DateTime.Now;
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    Console.WriteLine("cs;while;cnt:{1};lt:{0}",elapsed.Milliseconds-
(elapsed2.Milliseconds-elapsed.Milliseconds),N*times);

    timer_start= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        Array.ForEach(v, (int x) =>
        {
            a=x;
        });
    }
    timer_mid= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        Array.ForEach(v, (int x) =>
        {
            a=x;
            b=x;
        });
    }
    timer_end = DateTime.Now;
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
```

```
    Console.WriteLine("cs;ForEach + lamb-
da;cnt:{1};lt:{0}",elapsed.Milliseconds-(elapsed2.Milliseconds-
elapsed.Milliseconds),N*times);

    timer_start= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        Array.ForEach(v,func);
    }
    timer_mid= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        Array.ForEach(v,func2);
    }
    timer_end = DateTime.Now;
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    Console.WriteLine("cs;ForEach + named func-
tion;cnt:{1};lt:{0}",elapsed.Milliseconds-(elapsed2.Milliseconds-
elapsed.Milliseconds),N*times);

    timer_start= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        int i=0;
        recursiveFor(i,N,v);
    }
    timer_mid= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        int i=0;
        recursiveFor2(i,N,v);
    }
    timer_end = DateTime.Now;
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;

Console.WriteLine("cs;Recursive;cnt:{1};lt:{0}",elapsed.Milliseconds-
(elapsed2.Milliseconds-elapsed.Milliseconds),N*times);
}
}
```

### A.3. Java

```
import java.util.Date;
import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.function.Consumer;

public class ciklus{

    static Integer a;
    static Integer b;
```

```
public static void recursiveFor(int i, int N, Integer[] v) {
    if (i<N) {
        i++;
        recursiveFor(i,N,v);
        a=v[i-1];
    }
}

public static void recursiveFor2(int i, int N, Integer[] v) {
    if (i<N) {
        i++;
        recursiveFor2(i,N,v);
        a=v[i-1];
        b=v[i-1];
    }
}

public static void main(String[] args) {
    Date timer_start=new Date();
    Date timer_mid=new Date();
    Date timer_end;
    long elapsed;
    long elapsed2;

    int MaxN=65135;//100000000;
    int N=4000;//100000000;
    int times=10000; //30;

Integer[] v=new Integer[MaxN];
    for (int i=0; i<N; i++)
    {
        v[i]=i;
    }

    Consumer<Integer> func = (x) ->
    {
        a=x;
    };
    Consumer<Integer> func2 = (x) ->
    {
        a=x;
        b=x;
    };

timer_start = new Date();
for (int j=0; j<times; j++)
{
    for (int i=0; i<N; i++)
    {
        a=v[i];
    }
}
timer_mid = new Date();
for (int j=0; j<times; j++)
{
```

```
            for (int i=0; i<N; i++)
            {
                a=v[i];
                b=v[i];
            }
        }
        timer_end = new Date();
        elapsed = timer_mid.getTime() - timer_start.getTime();
        elapsed2 = timer_end.getTime() - timer_mid.getTime();
        System.out.println("Java;for;cnt:"+times*N+";lt:"+(elapsed-
    (elapsed2-elapsed)));

        timer_start = new Date();
        for (int j=0; j<times; j++)
        {
            int i=0;
            while (i<N)
            {
                a=v[i];
                i++;
            }
        }
        timer_mid = new Date();
        for (int j=0; j<times; j++)
        {
            int i=0;
            while (i<N)
            {
                a=v[i];
                b=v[i];
                i++;
            }
        }
        timer_end = new Date();
        elapsed = timer_mid.getTime() - timer_start.getTime();
        elapsed2 = timer_end.getTime() - timer_mid.getTime();
        System.out.println("Java;while;cnt:"+times*N+";lt:"+(elapsed-
    (elapsed2-elapsed)));

        timer_start = new Date();
        for (int j=0; j<times; j++)
        {
            for (Integer x : v)
            {
                a=x;
            }
        }
        timer_mid = new Date();
        for (int j=0; j<times; j++)
        {
            for (Integer x : v)
            {
                a=x;
                b=x;
            }
```

```
    }
    timer_end = new Date();
    elapsed = timer_mid.getTime() - timer_start.getTime();
    elapsed2 = timer_end.getTime() - timer_mid.getTime();
    System.out.println("Java;forEach;cnt:"+times*N+";lt:"+(elapsed-
(elapsed2-elapsed)));

    List<Integer> vl=Arrays.asList(v);
    timer_start =new Date();
    for (int j=0; j<times; j++)
    {
        vl.forEach((x)-> {
            a=x;
        });
    }
    timer_mid =new Date();
    for (int j=0; j<times; j++)
    {
        vl.forEach(x-> {
            a=x;
            b=x;
        });
    }
    timer_end = new Date();
    elapsed = timer_mid.getTime() - timer_start.getTime();
    elapsed2 = timer_end.getTime() - timer_mid.getTime();
    System.out.println("Java;ForEach + lamb-
da;cnt:"+times*N+";lt:"+(elapsed-(elapsed2-elapsed)));

    timer_start =new Date();
    for (int j=0; j<times; j++)
    {
        vl.forEach(func);
    }
          timer_mid =new Date();
    for (int j=0; j<times; j++)
    {
        vl.forEach(func2);
    }
    timer_end = new Date();
    elapsed = timer_mid.getTime() - timer_start.getTime();
    elapsed2 = timer_end.getTime() - timer_mid.getTime();
    System.out.println("Java;ForEach + named func-
tion;cnt:"+times*N+";lt:"+(elapsed-(elapsed2-elapsed)));

    timer_start = new Date();
    try {
    for (int j=0; j<times; j++)
    {
        int i=0;
        recursiveFor(i,N,v);
    }
    } catch(Exception ex) {
    }
    timer_mid = new Date();
```

```
        try {
        for (int j=0; j<times; j++)
        {
            int i=0;
            recursiveFor2(i,N,v);
        }
        } catch(Exception ex) {
        }
        timer_end = new Date();
        elapsed = timer_mid.getTime() - timer_start.getTime();
        elapsed2 = timer_end.getTime() - timer_mid.getTime();
        System.out.println("Java;Recursive;cnt:"+times*N+";lt:"+(elapsed-
(elapsed2-elapsed)));
        }
}
```

### A.4. JavaScript

```
let N=4000;
let times=10000;
let a;
let b;
let v=[];
for (let i=0; i<N; i++)
{
    v.push(i);
}

function recursiveFor(i, N, v) {
    if (i<N) {
        i++;
        recursiveFor(i,N,v);
        a=v[i-1];
    }
}

function recursiveFor2(i, N, v) {
    if (i<N) {
        i++;
        recursiveFor2(i,N,v);
        a=v[i-1];
        b=v[i-1];
    }
}

var timer_start = Date.now();
for (let j=0; j<times; j++)
{
    for (let i=0; i<N; i++)
    {
        a=i;
    }
}
var timer_mid = Date.now();
for (let j=0; j<times; j++)
```

```
{
    for (let i=0; i<N; i++)
    {
        a=i;
        b=i;
    }
}
var timer_end = Date.now();
elapsed=timer_mid-timer_start;
elapsed2=timer_end-timer_mid;
console.log("Js;for;cnt:",(times*N),";lt:",elapsed-(elapsed2-
elapsed));

var timer_start = Date.now();
for (let j=0; j<times; j++)
{
    let i=0;
    while (i<N)
    {
        a=i;
        i++;
    }
}
var timer_mid = Date.now();
for (let j=0; j<times; j++)
{
    let i=0;
    while (i<N)
    {
        a=i;
        b=i;
        i++;
    }
}
var timer_end = Date.now();
elapsed=timer_mid-timer_start;
elapsed2=timer_end-timer_mid;
console.log("Js;While;cnt:",(times*N),";lt:",elapsed-(elapsed2-
elapsed));

var timer_start = Date.now();
for (let j=0; j<times; j++)
{
    v.forEach((item, index, arr)=>{
        a=item;
    });
}
var timer_mid = Date.now();
for (let j=0; j<times; j++)
{
    v.forEach((item, index, arr)=>{
        a=item;
        b=item;
    });
}
```

```
var timer_end = Date.now();
elapsed=timer_mid-timer_start;
elapsed2=timer_end-timer_mid;
console.log("Js;ForEach + lambda func-
tion;cnt:",(times*N),";lt:",elapsed-(elapsed2-elapsed));

function func(item, index, arr) {
  a=item;
}
function func2(item, index, arr) {
  a=item;
  b=item;
}
var timer_start = Date.now();
for (let j=0; j<times; j++)
{
    v.forEach(func);
}
var timer_mid = Date.now();
for (let j=0; j<times; j++)
{
    v.forEach(func2);
}
var timer_end = Date.now();
elapsed=timer_mid-timer_start;
elapsed2=timer_end-timer_mid;
console.log("Js;ForEach + named func-
tion;cnt:",(times*N),";lt:",elapsed-(elapsed2-elapsed));

var timer_start = Date.now();
for (let j=0; j<times; j++)
{
    let i=0;
    recursiveFor(i,N,v);
}
var timer_mid = Date.now();
for (let j=0; j<times; j++)
{
    let i=0;
    recursiveFor2(i,N,v);
}
var timer_end = Date.now();
elapsed=timer_mid-timer_start;
elapsed2=timer_end-timer_mid;
console.log("Js;Recursive;cnt:",(times*N),";lt:",elapsed-(elapsed2-
elapsed));
```

### A.5. Pascal

```
Program ciklus;

Uses sysutils;

  var
    timer_start:TDateTime;
```

```
    timer_mid:TDateTime;
    timer_end:TDateTime;
    elapsed:TDateTime;
    elapsed2:TDateTime;
    N:integer;
    times:integer;
    v:array[1..65135] of integer;
    i:integer;
    j:integer;
    a:integer;
    b:integer;
    w:integer;

procedure recursiveFor(var i:integer; const N:integer; var v:array of
integer);
begin
    if (i<=N) then begin
        i:=i+1;
        recursiveFor(i,N,v);
        a:=v[i-1];
    end
end;

procedure recursiveFor2(var i:integer; const N:integer; var v:array
of integer);
begin
    if (i<=N) then begin
        i:=i+1;
        recursiveFor2(i,N,v);
        a:=v[i-1];
        b:=v[i-1];
    end
end;

BEGIN
    N:=4000;
    times:=10000;

    for i:=1 to N do begin
        v[i]:=i;
    end;

    timer_start:=Now;
    for j:=1 to times do begin
        for i:=1 to N do begin
            a:=v[i];
        end;
    end;
    timer_mid:=Now;
    for j:=1 to times do begin
        for i:=1 to N do begin
            a:=v[i];
            b:=v[i];
        end;
    end;
```

```
    timer_end:=Now;
    elapsed:=timer_mid-timer_start;
    elapsed2:=timer_end-timer_mid;
    WriteLn('Pas;for;cnt:',times*N,';lt:',(elapsed-(elapsed2-
elapsed))*100000000:6:0);

    timer_start:=Now;
    for j:=1 to times do begin
        i:=1;
        while (i<=N) do begin
            a:=v[i];
            i:=i+1;
        end;
    end;
    timer_mid:=Now;
    for j:=1 to times do begin
        i:=1;
        while (i<=N) do begin
            a:=v[i];
            b:=v[i];
            i:=i+1;
        end;
    end;
    timer_end:=Now;
    elapsed:=timer_mid-timer_start;
    elapsed2:=timer_end-timer_mid;
    WriteLn('Pas;while;cnt:',times*N,';lt:',(elapsed-(elapsed2-
elapsed))*100000000:6:0);

    timer_start:=Now;
    for j:=1 to times do begin
        for w in v do begin
            a:=w;
        end;
    end;
    timer_mid:=Now;
    for j:=1 to times do begin
        for w in v do begin
            a:=w;
            b:=w;
        end;
    end;
    timer_end:=Now;
    elapsed:=timer_mid-timer_start;
    elapsed2:=timer_end-timer_mid;
    WriteLn('Pas;forEach;cnt:',times*N,';lt:',(elapsed-(elapsed2-
elapsed))*100000000:6:0);

    timer_start:=Now;
    for j:=1 to times do begin
    end;
    timer_end:=Now;
    elapsed:=timer_end-timer_start;
    WriteLn('Pas;ForEach + lambda;cnt:',times*N,';lt:Not supported');
    timer_start:=Now;
```

```
    for j:=1 to times do begin
    end;
    timer_end:=Now;
    elapsed:=timer_end-timer_start;
    WriteLn('Pas;ForEach + named function;cnt:',times*N,';lt:Not sup-
ported');
    timer_start:=Now;
    for j:=1 to times do begin
        i:=1;
        recursiveFor(i,N,v);
    end;
    timer_mid:=Now;
    for j:=1 to times do begin
        i:=1;
        recursiveFor2(i,N,v);
    end;
    timer_end:=Now;
    elapsed:=timer_mid-timer_start;
    elapsed2:=timer_end-timer_mid;
    WriteLn('Pas;Recursive;cnt:',times*N,';lt:',(elapsed-(elapsed2-
elapsed))*100000000:6:0);

END.
```

### A.6. Python

```
import sys
from datetime import datetime

#print sys.getrecursionlimit()
sys.setrecursionlimit(100000)
#print sys.getrecursionlimit()

N=4000;
times=10000;
v=[]
for i in range(1,N):
  v.append(i);

def func(x):
  a=x

def func2(x):
  a=x
  b=x

def recursiveFor(i,N,v):
  if i<N-1:
    i=i+1
    recursiveFor(i,N,v);
    a=v[i-1]

def recursiveFor2(i,N,v):
  if i<N-1:
    i=i+1
    recursiveFor2(i,N,v);
```

```
    a=v[i-1]
    b=v[i-1]

timer_start = datetime.now()
for j in range(1,times):
  for i in range(1,N):
    a=v[i-1]
timer_mid = datetime.now()
for j in range(1,times):
  for i in range(1,N):
    a=v[i-1]
    b=v[i-1]
timer_end = datetime.now()
elapsed=timer_mid-timer_start
elapsed2=timer_end-timer_mid
print
'Py;For;cnt:',N*times,';lt:',(elapsed.seconds*1000+elapsed.microsecon
ds/1000)-((elapsed2.seconds*1000+elapsed2.microseconds/1000)-
(elapsed.seconds*1000+elapsed.microseconds/1000))

timer_start = datetime.now()
for j in range(1,times):
  i=0
  while i<N-1:
    #std::cout << v[i] << std::endl;
    a=v[i]
    i=i+1
timer_mid = datetime.now()
for j in range(1,times):
  i=0
  while i<N-1:
    a=v[i]
    b=v[i]
    i=i+1
timer_end = datetime.now()
elapsed=timer_mid-timer_start
elapsed2=timer_end-timer_mid
print
'Py;While;cnt:',N*times,';lt:',(elapsed.seconds*1000+elapsed.microsec
onds/1000)-((elapsed2.seconds*1000+elapsed2.microseconds/1000)-
(elapsed.seconds*1000+elapsed.microseconds/1000))

timer_start = datetime.now()
for j in range(1,times):
  for x in v:
    a=x
timer_mid = datetime.now()
for j in range(1,times):
  for x in v:
    a=x
    b=x
timer_end = datetime.now()
elapsed=timer_mid-timer_start
elapsed2=timer_end-timer_mid
```

```
print 'Py;ForEach [+ lamb-
da];cnt:',N*times,';lt:',(elapsed.seconds*1000+elapsed.microseconds/1
000)-((elapsed2.seconds*1000+elapsed2.microseconds/1000)-
(elapsed.seconds*1000+elapsed.microseconds/1000))

timer_start = datetime.now()
for j in range(1,times):
  for x in v:
    func(x)
timer_mid = datetime.now()
for j in range(1,times):
  for x in v:
    func2(x)
timer_end = datetime.now()
elapsed=timer_mid-timer_start
elapsed2=timer_end-timer_mid
print 'Py;ForEach + named func-
tion;cnt:',N*times,';lt:',(elapsed.seconds*1000+elapsed.microseconds/
1000)-((elapsed2.seconds*1000+elapsed2.microseconds/1000)-
(elapsed.seconds*1000+elapsed.microseconds/1000))

timer_start = datetime.now()
for j in range(1,times):
  i=0
  recursiveFor(i,N,v);
timer_mid = datetime.now()
for j in range(1,times):
  i=0
  recursiveFor2(i,N,v);
timer_end = datetime.now()
elapsed=timer_mid-timer_start
elapsed2=timer_end-timer_mid
print
'Py;recursive;cnt:',N*times,';lt:',(elapsed.seconds*1000+elapsed.micr
oseconds/1000)-((elapsed2.seconds*1000+elapsed2.microseconds/1000)-
(elapsed.seconds*1000+elapsed.microseconds/1000))
```

## B. A mérés nyers adatai

```
Cpp;while;cnt:40000000;lt:0.056
Cpp;do-while;cnt:40000000;lt:0.658
Cpp;for;cnt:40000000;lt:31.361
Cpp;ForEach + lambda;cnt:40000000;lt:1076.23
Cpp;ForEach + named function;cnt:40000000;lt:1087.14
Cpp;Recursive;cnt:40000000;lt:124.673
cs;for;cnt:40000000;lt:22
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:92
cs;ForEach + named function;cnt:40000000;lt:89
cs;Recursive;cnt:40000000;lt:85
Java;for;cnt:40000000;lt:56
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:213
Java;ForEach + lambda;cnt:40000000;lt:153
Java;ForEach + named function;cnt:40000000;lt:1901
```

```
Java;Recursive;cnt:40000000;lt:98
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 249
Js;ForEach + named function;cnt: 40000000 ;lt: 229
Js;Recursive;cnt: 40000000 ;lt: 263
Pas;for;cnt:40000000;lt:    97
Pas;while;cnt:40000000;lt:    58
Pas;forEach;cnt:40000000;lt:   549
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   130
Py;For;cnt: 40000000 ;lt: 1140
Py;While;cnt: 40000000 ;lt: 1872
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 265
Py;ForEach + named function;cnt: 40000000 ;lt: 3176
Py;recursive;cnt: 40000000 ;lt: 9022

Cpp;while;cnt:40000000;lt:11.553
Cpp;do-while;cnt:40000000;lt:0.589
Cpp;for;cnt:40000000;lt:22.609
Cpp;ForEach + lambda;cnt:40000000;lt:1095.92
Cpp;ForEach + named function;cnt:40000000;lt:1125.63
Cpp;Recursive;cnt:40000000;lt:110.243
cs;for;cnt:40000000;lt:31
cs;while;cnt:40000000;lt:8
cs;ForEach + lambda;cnt:40000000;lt:117
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:91
Java;for;cnt:40000000;lt:37
Java;while;cnt:40000000;lt:48
Java;forEach;cnt:40000000;lt:199
Java;ForEach + lambda;cnt:40000000;lt:157
Java;ForEach + named function;cnt:40000000;lt:1961
Java;Recursive;cnt:40000000;lt:138
Js;for;cnt: 40000000 ;lt: 37
Js;While;cnt: 40000000 ;lt: 30
Js;ForEach + lambda function;cnt: 40000000 ;lt: 235
Js;ForEach + named function;cnt: 40000000 ;lt: 227
Js;Recursive;cnt: 40000000 ;lt: 251
Pas;for;cnt:40000000;lt:   101
Pas;while;cnt:40000000;lt:    94
Pas;forEach;cnt:40000000;lt:   546
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   118
Py;For;cnt: 40000000 ;lt: 1204
Py;While;cnt: 40000000 ;lt: 2665
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 270
Py;ForEach + named function;cnt: 40000000 ;lt: 3197
Py;recursive;cnt: 40000000 ;lt: 8994

Cpp;while;cnt:40000000;lt:24.209
Cpp;do-while;cnt:40000000;lt:1.789
Cpp;for;cnt:40000000;lt:4.774
```

```
Cpp;ForEach + lambda;cnt:40000000;lt:1095
Cpp;ForEach + named function;cnt:40000000;lt:1109.37
Cpp;Recursive;cnt:40000000;lt:126.22
cs;for;cnt:40000000;lt:47
cs;while;cnt:40000000;lt:2
cs;ForEach + lambda;cnt:40000000;lt:90
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:93
Java;for;cnt:40000000;lt:50
Java;while;cnt:40000000;lt:66
Java;forEach;cnt:40000000;lt:228
Java;ForEach + lambda;cnt:40000000;lt:157
Java;ForEach + named function;cnt:40000000;lt:1938
Java;Recursive;cnt:40000000;lt:112
Js;for;cnt: 40000000 ;lt: 37
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 238
Js;ForEach + named function;cnt: 40000000 ;lt: 231
Js;Recursive;cnt: 40000000 ;lt: 251
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    62
Pas;forEach;cnt:40000000;lt:   560
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   159
Py;For;cnt: 40000000 ;lt: 1042
Py;While;cnt: 40000000 ;lt: 2833
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 235
Py;ForEach + named function;cnt: 40000000 ;lt: 3176
Py;recursive;cnt: 40000000 ;lt: 9232

Cpp;while;cnt:40000000;lt:4.446
Cpp;do-while;cnt:40000000;lt:3.817
Cpp;for;cnt:40000000;lt:0.397
Cpp;ForEach + lambda;cnt:40000000;lt:1093.52
Cpp;ForEach + named function;cnt:40000000;lt:1108.86
Cpp;Recursive;cnt:40000000;lt:127.066
cs;for;cnt:40000000;lt:13
cs;while;cnt:40000000;lt:8
cs;ForEach + lambda;cnt:40000000;lt:125
cs;ForEach + named function;cnt:40000000;lt:82
cs;Recursive;cnt:40000000;lt:93
Java;for;cnt:40000000;lt:58
Java;while;cnt:40000000;lt:31
Java;forEach;cnt:40000000;lt:211
Java;ForEach + lambda;cnt:40000000;lt:166
Java;ForEach + named function;cnt:40000000;lt:1922
Java;Recursive;cnt:40000000;lt:113
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 247
Js;ForEach + named function;cnt: 40000000 ;lt: 231
Js;Recursive;cnt: 40000000 ;lt: 271
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    67
```

```
Pas;forEach;cnt:40000000;lt:   556
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   133
Py;For;cnt: 40000000 ;lt: 1131
Py;While;cnt: 40000000 ;lt: 2009
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 277
Py;ForEach + named function;cnt: 40000000 ;lt: 2980
Py;recursive;cnt: 40000000 ;lt: 7478

Cpp;while;cnt:40000000;lt:3.886
Cpp;do-while;cnt:40000000;lt:0.068
Cpp;for;cnt:40000000;lt:2.29
Cpp;ForEach + lambda;cnt:40000000;lt:1114.82
Cpp;ForEach + named function;cnt:40000000;lt:1160.19
Cpp;Recursive;cnt:40000000;lt:134.357
cs;for;cnt:40000000;lt:20
cs;while;cnt:40000000;lt:21
cs;ForEach + lambda;cnt:40000000;lt:91
cs;ForEach + named function;cnt:40000000;lt:93
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:65
Java;while;cnt:40000000;lt:61
Java;forEach;cnt:40000000;lt:222
Java;ForEach + lambda;cnt:40000000;lt:160
Java;ForEach + named function;cnt:40000000;lt:2018
Java;Recursive;cnt:40000000;lt:117
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 252
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 270
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   564
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   137
Py;For;cnt: 40000000 ;lt: 1190
Py;While;cnt: 40000000 ;lt: 1832
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 300
Py;ForEach + named function;cnt: 40000000 ;lt: 3291
Py;recursive;cnt: 40000000 ;lt: 9734

Cpp;while;cnt:40000000;lt:5.952
Cpp;do-while;cnt:40000000;lt:3.779
Cpp;for;cnt:40000000;lt:5.6
Cpp;ForEach + lambda;cnt:40000000;lt:1136.82
Cpp;ForEach + named function;cnt:40000000;lt:1141.19
Cpp;Recursive;cnt:40000000;lt:133.593
cs;for;cnt:40000000;lt:23
cs;while;cnt:40000000;lt:22
cs;ForEach + lambda;cnt:40000000;lt:93
cs;ForEach + named function;cnt:40000000;lt:91
cs;Recursive;cnt:40000000;lt:89
```

```
Java;for;cnt:40000000;lt:38
Java;while;cnt:40000000;lt:54
Java;forEach;cnt:40000000;lt:218
Java;ForEach + lambda;cnt:40000000;lt:177
Java;ForEach + named function;cnt:40000000;lt:2018
Java;Recursive;cnt:40000000;lt:110
Js;for;cnt: 40000000 ;lt: 40
Js;While;cnt: 40000000 ;lt: 40
Js;ForEach + lambda function;cnt: 40000000 ;lt: 253
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 265
Pas;for;cnt:40000000;lt:    81
Pas;while;cnt:40000000;lt:    66
Pas;forEach;cnt:40000000;lt:   565
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   131
Py;For;cnt: 40000000 ;lt: 101
Py;While;cnt: 40000000 ;lt: 1431
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 1225
Py;ForEach + named function;cnt: 40000000 ;lt: 5895
Py;recursive;cnt: 40000000 ;lt: 14090

Cpp;while;cnt:40000000;lt:5.244
Cpp;do-while;cnt:40000000;lt:0.978
Cpp;for;cnt:40000000;lt:3.862
Cpp;ForEach + lambda;cnt:40000000;lt:1139.48
Cpp;ForEach + named function;cnt:40000000;lt:1143.29
Cpp;Recursive;cnt:40000000;lt:133.426
cs;for;cnt:40000000;lt:23
cs;while;cnt:40000000;lt:21
cs;ForEach + lambda;cnt:40000000;lt:93
cs;ForEach + named function;cnt:40000000;lt:94
cs;Recursive;cnt:40000000;lt:89
Java;for;cnt:40000000;lt:38
Java;while;cnt:40000000;lt:48
Java;forEach;cnt:40000000;lt:232
Java;ForEach + lambda;cnt:40000000;lt:172
Java;ForEach + named function;cnt:40000000;lt:2012
Java;Recursive;cnt:40000000;lt:114
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 36
Js;ForEach + lambda function;cnt: 40000000 ;lt: 247
Js;ForEach + named function;cnt: 40000000 ;lt: 237
Js;Recursive;cnt: 40000000 ;lt: 272
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   565
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   139
Py;For;cnt: 40000000 ;lt: 1654
Py;While;cnt: 40000000 ;lt: 2797
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 265
Py;ForEach + named function;cnt: 40000000 ;lt: 3279
```

```
Py;recursive;cnt: 40000000 ;lt: 10467

Cpp;while;cnt:40000000;lt:5.164
Cpp;do-while;cnt:40000000;lt:2.288
Cpp;for;cnt:40000000;lt:4.173
Cpp;ForEach + lambda;cnt:40000000;lt:1149.63
Cpp;ForEach + named function;cnt:40000000;lt:1146.72
Cpp;Recursive;cnt:40000000;lt:134.877
cs;for;cnt:40000000;lt:24
cs;while;cnt:40000000;lt:22
cs;ForEach + lambda;cnt:40000000;lt:93
cs;ForEach + named function;cnt:40000000;lt:91
cs;Recursive;cnt:40000000;lt:111
Java;for;cnt:40000000;lt:59
Java;while;cnt:40000000;lt:52
Java;forEach;cnt:40000000;lt:247
Java;ForEach + lambda;cnt:40000000;lt:107
Java;ForEach + named function;cnt:40000000;lt:2035
Java;Recursive;cnt:40000000;lt:114
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 259
Js;ForEach + named function;cnt: 40000000 ;lt: 244
Js;Recursive;cnt: 40000000 ;lt: 280
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    65
Pas;forEach;cnt:40000000;lt:   646
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   152
Py;For;cnt: 40000000 ;lt: 1313
Py;While;cnt: 40000000 ;lt: 1905
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 268
Py;ForEach + named function;cnt: 40000000 ;lt: 3216
Py;recursive;cnt: 40000000 ;lt: 9942

Cpp;while;cnt:40000000;lt:0.999
Cpp;do-while;cnt:40000000;lt:2.048
Cpp;for;cnt:40000000;lt:0.98
Cpp;ForEach + lambda;cnt:40000000;lt:1216.67
Cpp;ForEach + named function;cnt:40000000;lt:1147.93
Cpp;Recursive;cnt:40000000;lt:139.67
cs;for;cnt:40000000;lt:23
cs;while;cnt:40000000;lt:21
cs;ForEach + lambda;cnt:40000000;lt:95
cs;ForEach + named function;cnt:40000000;lt:94
cs;Recursive;cnt:40000000;lt:89
Java;for;cnt:40000000;lt:41
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:279
Java;ForEach + lambda;cnt:40000000;lt:193
Java;ForEach + named function;cnt:40000000;lt:1988
Java;Recursive;cnt:40000000;lt:44
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
```

```
Js;ForEach + lambda function;cnt: 40000000 ;lt: 241
Js;ForEach + named function;cnt: 40000000 ;lt: 228
Js;Recursive;cnt: 40000000 ;lt: 272
Pas;for;cnt:40000000;lt:    81
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   583
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   134
Py;For;cnt: 40000000 ;lt: 809
Py;While;cnt: 40000000 ;lt: 2704
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 234
Py;ForEach + named function;cnt: 40000000 ;lt: 3207
Py;recursive;cnt: 40000000 ;lt: 10193

Cpp;while;cnt:40000000;lt:1.675
Cpp;do-while;cnt:40000000;lt:1.435
Cpp;for;cnt:40000000;lt:4.495
Cpp;ForEach + lambda;cnt:40000000;lt:1117.01
Cpp;ForEach + named function;cnt:40000000;lt:1149.84
Cpp;Recursive;cnt:40000000;lt:127.438
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:7
cs;ForEach + lambda;cnt:40000000;lt:118
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:57
Java;while;cnt:40000000;lt:47
Java;forEach;cnt:40000000;lt:228
Java;ForEach + lambda;cnt:40000000;lt:175
Java;ForEach + named function;cnt:40000000;lt:1965
Java;Recursive;cnt:40000000;lt:117
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 244
Js;ForEach + named function;cnt: 40000000 ;lt: 218
Js;Recursive;cnt: 40000000 ;lt: 259
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   544
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   132
Py;For;cnt: 40000000 ;lt: 75
Py;While;cnt: 40000000 ;lt: 3586
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 975
Py;ForEach + named function;cnt: 40000000 ;lt: 3906
Py;recursive;cnt: 40000000 ;lt: 8930

Cpp;while;cnt:40000000;lt:48.477
Cpp;do-while;cnt:40000000;lt:174.121
Cpp;for;cnt:40000000;lt:0.981
Cpp;ForEach + lambda;cnt:40000000;lt:1697.38
Cpp;ForEach + named function;cnt:40000000;lt:1959.92
Cpp;Recursive;cnt:40000000;lt:216.816
```

```
cs;for;cnt:40000000;lt:88
cs;while;cnt:40000000;lt:60
cs;ForEach + lambda;cnt:40000000;lt:311
cs;ForEach + named function;cnt:40000000;lt:212
cs;Recursive;cnt:40000000;lt:128
Java;for;cnt:40000000;lt:76
Java;while;cnt:40000000;lt:183
Java;forEach;cnt:40000000;lt:510
Java;ForEach + lambda;cnt:40000000;lt:324
Java;ForEach + named function;cnt:40000000;lt:3482
Java;Recursive;cnt:40000000;lt:148
Js;for;cnt: 40000000 ;lt: 206
Js;While;cnt: 40000000 ;lt: 115
Js;ForEach + lambda function;cnt: 40000000 ;lt: 529
Js;ForEach + named function;cnt: 40000000 ;lt: 448
Js;Recursive;cnt: 40000000 ;lt: 400
Pas;for;cnt:40000000;lt:   278
Pas;while;cnt:40000000;lt:    60
Pas;forEach;cnt:40000000;lt:   817
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   215
Py;For;cnt: 40000000 ;lt: 1964
Py;While;cnt: 40000000 ;lt: 3432
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 1442
Py;ForEach + named function;cnt: 40000000 ;lt: 5341
Py;recursive;cnt: 40000000 ;lt: 14436

Cpp;while;cnt:40000000;lt:13.586
Cpp;do-while;cnt:40000000;lt:1.62
Cpp;for;cnt:40000000;lt:5.154
Cpp;ForEach + lambda;cnt:40000000;lt:1106.2
Cpp;ForEach + named function;cnt:40000000;lt:1108.6
Cpp;Recursive;cnt:40000000;lt:127.709
cs;for;cnt:40000000;lt:14
cs;while;cnt:40000000;lt:31
cs;ForEach + lambda;cnt:40000000;lt:113
cs;ForEach + named function;cnt:40000000;lt:88
cs;Recursive;cnt:40000000;lt:94
Java;for;cnt:40000000;lt:34
Java;while;cnt:40000000;lt:40
Java;forEach;cnt:40000000;lt:211
Java;ForEach + lambda;cnt:40000000;lt:154
Java;ForEach + named function;cnt:40000000;lt:1977
Java;Recursive;cnt:40000000;lt:110
Js;for;cnt: 40000000 ;lt: 34
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 249
Js;ForEach + named function;cnt: 40000000 ;lt: 249
Js;Recursive;cnt: 40000000 ;lt: 260
Pas;for;cnt:40000000;lt:    81
Pas;while;cnt:40000000;lt:    69
Pas;forEach;cnt:40000000;lt:   515
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
```

```
Pas;Recursive;cnt:40000000;lt:   128
Py;For;cnt: 40000000 ;lt: 1158
Py;While;cnt: 40000000 ;lt: 1598
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 735
Py;ForEach + named function;cnt: 40000000 ;lt: 3325
Py;recursive;cnt: 40000000 ;lt: 9003

Cpp;while;cnt:40000000;lt:7.54
Cpp;do-while;cnt:40000000;lt:6.197
Cpp;for;cnt:40000000;lt:0.663
Cpp;ForEach + lambda;cnt:40000000;lt:1136.38
Cpp;ForEach + named function;cnt:40000000;lt:1125.48
Cpp;Recursive;cnt:40000000;lt:128.11
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:95
cs;ForEach + named function;cnt:40000000;lt:98
cs;Recursive;cnt:40000000;lt:102
Java;for;cnt:40000000;lt:62
Java;while;cnt:40000000;lt:49
Java;forEach;cnt:40000000;lt:219
Java;ForEach + lambda;cnt:40000000;lt:138
Java;ForEach + named function;cnt:40000000;lt:2032
Java;Recursive;cnt:40000000;lt:89
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 246
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 255
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    52
Pas;forEach;cnt:40000000;lt:   574
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   148
Py;For;cnt: 40000000 ;lt: 1175
Py;While;cnt: 40000000 ;lt: 1874
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 233
Py;ForEach + named function;cnt: 40000000 ;lt: 3260
Py;recursive;cnt: 40000000 ;lt: 9195

Cpp;while;cnt:40000000;lt:13.119
Cpp;do-while;cnt:40000000;lt:17.027
Cpp;for;cnt:40000000;lt:9.737
Cpp;ForEach + lambda;cnt:40000000;lt:1119.74
Cpp;ForEach + named function;cnt:40000000;lt:1099.06
Cpp;Recursive;cnt:40000000;lt:128.921
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:21
cs;ForEach + lambda;cnt:40000000;lt:98
cs;ForEach + named function;cnt:40000000;lt:94
cs;Recursive;cnt:40000000;lt:83
Java;for;cnt:40000000;lt:54
Java;while;cnt:40000000;lt:42
Java;forEach;cnt:40000000;lt:220
```

```
Java;ForEach + lambda;cnt:40000000;lt:174
Java;ForEach + named function;cnt:40000000;lt:1971
Java;Recursive;cnt:40000000;lt:100
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 242
Js;ForEach + named function;cnt: 40000000 ;lt: 233
Js;Recursive;cnt: 40000000 ;lt: 284
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   549
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   134
Py;For;cnt: 40000000 ;lt: 1145
Py;While;cnt: 40000000 ;lt: 1776
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 729
Py;ForEach + named function;cnt: 40000000 ;lt: 3322
Py;recursive;cnt: 40000000 ;lt: 8969

Cpp;while;cnt:40000000;lt:3.234
Cpp;do-while;cnt:40000000;lt:0.561
Cpp;for;cnt:40000000;lt:4.021
Cpp;ForEach + lambda;cnt:40000000;lt:1119.4
Cpp;ForEach + named function;cnt:40000000;lt:1111.89
Cpp;Recursive;cnt:40000000;lt:128.048
cs;for;cnt:40000000;lt:22
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:94
cs;ForEach + named function;cnt:40000000;lt:91
cs;Recursive;cnt:40000000;lt:87
Java;for;cnt:40000000;lt:53
Java;while;cnt:40000000;lt:49
Java;forEach;cnt:40000000;lt:232
Java;ForEach + lambda;cnt:40000000;lt:184
Java;ForEach + named function;cnt:40000000;lt:1972
Java;Recursive;cnt:40000000;lt:100
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 243
Js;ForEach + named function;cnt: 40000000 ;lt: 230
Js;Recursive;cnt: 40000000 ;lt: 260
Pas;for;cnt:40000000;lt:    79
Pas;while;cnt:40000000;lt:    65
Pas;forEach;cnt:40000000;lt:   553
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   130
Py;For;cnt: 40000000 ;lt: 1162
Py;While;cnt: 40000000 ;lt: 2469
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 248
Py;ForEach + named function;cnt: 40000000 ;lt: 5494
Py;recursive;cnt: 40000000 ;lt: 9184

Cpp;while;cnt:40000000;lt:0.22
```

```
Cpp;do-while;cnt:40000000;lt:0.463
Cpp;for;cnt:40000000;lt:5.005
Cpp;ForEach + lambda;cnt:40000000;lt:1107.88
Cpp;ForEach + named function;cnt:40000000;lt:1121.43
Cpp;Recursive;cnt:40000000;lt:131.073
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:22
cs;ForEach + lambda;cnt:40000000;lt:93
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:93
Java;for;cnt:40000000;lt:36
Java;while;cnt:40000000;lt:53
Java;forEach;cnt:40000000;lt:225
Java;ForEach + lambda;cnt:40000000;lt:159
Java;ForEach + named function;cnt:40000000;lt:1950
Java;Recursive;cnt:40000000;lt:95
Js;for;cnt: 40000000 ;lt: 32
Js;While;cnt: 40000000 ;lt: 31
Js;ForEach + lambda function;cnt: 40000000 ;lt: 250
Js;ForEach + named function;cnt: 40000000 ;lt: 236
Js;Recursive;cnt: 40000000 ;lt: 263
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   546
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   127
Py;For;cnt: 40000000 ;lt: 1197
Py;While;cnt: 40000000 ;lt: 1660
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 47
Py;ForEach + named function;cnt: 40000000 ;lt: 2791
Py;recursive;cnt: 40000000 ;lt: 8820

Cpp;while;cnt:40000000;lt:0.233
Cpp;do-while;cnt:40000000;lt:2.049
Cpp;for;cnt:40000000;lt:6.047
Cpp;ForEach + lambda;cnt:40000000;lt:1114.01
Cpp;ForEach + named function;cnt:40000000;lt:1118.2
Cpp;Recursive;cnt:40000000;lt:133.1
cs;for;cnt:40000000;lt:29
cs;while;cnt:40000000;lt:23
cs;ForEach + lambda;cnt:40000000;lt:92
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:55
Java;while;cnt:40000000;lt:52
Java;forEach;cnt:40000000;lt:224
Java;ForEach + lambda;cnt:40000000;lt:161
Java;ForEach + named function;cnt:40000000;lt:1966
Java;Recursive;cnt:40000000;lt:117
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 244
Js;ForEach + named function;cnt: 40000000 ;lt: 235
Js;Recursive;cnt: 40000000 ;lt: 261
```

```
Pas;for;cnt:40000000;lt:    79
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   550
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   137
Py;For;cnt: 40000000 ;lt: 81
Py;While;cnt: 40000000 ;lt: 2897
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 216
Py;ForEach + named function;cnt: 40000000 ;lt: 3273
Py;recursive;cnt: 40000000 ;lt: 11002

Cpp;while;cnt:40000000;lt:1.006
Cpp;do-while;cnt:40000000;lt:0.084
Cpp;for;cnt:40000000;lt:2.409
Cpp;ForEach + lambda;cnt:40000000;lt:1114.54
Cpp;ForEach + named function;cnt:40000000;lt:1099.17
Cpp;Recursive;cnt:40000000;lt:128.804
cs;for;cnt:40000000;lt:22
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:94
cs;ForEach + named function;cnt:40000000;lt:89
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:54
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:209
Java;ForEach + lambda;cnt:40000000;lt:160
Java;ForEach + named function;cnt:40000000;lt:1966
Java;Recursive;cnt:40000000;lt:113
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 241
Js;ForEach + named function;cnt: 40000000 ;lt: 234
Js;Recursive;cnt: 40000000 ;lt: 257
Pas;for;cnt:40000000;lt:    79
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   545
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   134
Py;For;cnt: 40000000 ;lt: 461
Py;While;cnt: 40000000 ;lt: 2933
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 1159
Py;ForEach + named function;cnt: 40000000 ;lt: 3164
Py;recursive;cnt: 40000000 ;lt: 11564

Cpp;while;cnt:40000000;lt:0.98
Cpp;do-while;cnt:40000000;lt:2.854
Cpp;for;cnt:40000000;lt:6.992
Cpp;ForEach + lambda;cnt:40000000;lt:1116.53
Cpp;ForEach + named function;cnt:40000000;lt:1115.34
Cpp;Recursive;cnt:40000000;lt:127.015
cs;for;cnt:40000000;lt:38
cs;while;cnt:40000000;lt:8
cs;ForEach + lambda;cnt:40000000;lt:102
```

```
cs;ForEach + named function;cnt:40000000;lt:93
cs;Recursive;cnt:40000000;lt:80
Java;for;cnt:40000000;lt:64
Java;while;cnt:40000000;lt:49
Java;forEach;cnt:40000000;lt:210
Java;ForEach + lambda;cnt:40000000;lt:157
Java;ForEach + named function;cnt:40000000;lt:1985
Java;Recursive;cnt:40000000;lt:105
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 240
Js;ForEach + named function;cnt: 40000000 ;lt: 233
Js;Recursive;cnt: 40000000 ;lt: 264
Pas;for;cnt:40000000;lt:    80
Pas;while;cnt:40000000;lt:    68
Pas;forEach;cnt:40000000;lt:   557
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   147
Py;For;cnt: 40000000 ;lt: 1180
Py;While;cnt: 40000000 ;lt: 1764
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 320
Py;ForEach + named function;cnt: 40000000 ;lt: 3195
Py;recursive;cnt: 40000000 ;lt: 9001

Cpp;while;cnt:40000000;lt:2.975
Cpp;do-while;cnt:40000000;lt:3.126
Cpp;for;cnt:40000000;lt:4.612
Cpp;ForEach + lambda;cnt:40000000;lt:1122.08
Cpp;ForEach + named function;cnt:40000000;lt:1115.68
Cpp;Recursive;cnt:40000000;lt:125.834
cs;for;cnt:40000000;lt:24
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:91
cs;ForEach + named function;cnt:40000000;lt:89
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:37
Java;while;cnt:40000000;lt:47
Java;forEach;cnt:40000000;lt:222
Java;ForEach + lambda;cnt:40000000;lt:148
Java;ForEach + named function;cnt:40000000;lt:1964
Java;Recursive;cnt:40000000;lt:102
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 240
Js;ForEach + named function;cnt: 40000000 ;lt: 232
Js;Recursive;cnt: 40000000 ;lt: 262
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   552
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 1146
Py;While;cnt: 40000000 ;lt: 1817
```

```
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 193
Py;ForEach + named function;cnt: 40000000 ;lt: 3313
Py;recursive;cnt: 40000000 ;lt: 11275

Cpp;while;cnt:40000000;lt:3.112
Cpp;do-while;cnt:40000000;lt:1.315
Cpp;for;cnt:40000000;lt:3.389
Cpp;ForEach + lambda;cnt:40000000;lt:1113.93
Cpp;ForEach + named function;cnt:40000000;lt:1119.23
Cpp;Recursive;cnt:40000000;lt:128.719
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:92
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:87
Java;for;cnt:40000000;lt:55
Java;while;cnt:40000000;lt:50
Java;forEach;cnt:40000000;lt:241
Java;ForEach + lambda;cnt:40000000;lt:159
Java;ForEach + named function;cnt:40000000;lt:1982
Java;Recursive;cnt:40000000;lt:104
Js;for;cnt: 40000000 ;lt: 37
Js;While;cnt: 40000000 ;lt: 30
Js;ForEach + lambda function;cnt: 40000000 ;lt: 244
Js;ForEach + named function;cnt: 40000000 ;lt: 231
Js;Recursive;cnt: 40000000 ;lt: 261
Pas;for;cnt:40000000;lt:    87
Pas;while;cnt:40000000;lt:    66
Pas;forEach;cnt:40000000;lt:   554
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   133
Py;For;cnt: 40000000 ;lt: 1135
Py;While;cnt: 40000000 ;lt: 1383
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 194
Py;ForEach + named function;cnt: 40000000 ;lt: 2675
Py;recursive;cnt: 40000000 ;lt: 8004

Cpp;while;cnt:40000000;lt:7.681
Cpp;do-while;cnt:40000000;lt:1.819
Cpp;for;cnt:40000000;lt:6.874
Cpp;ForEach + lambda;cnt:40000000;lt:1107.76
Cpp;ForEach + named function;cnt:40000000;lt:1107.76
Cpp;Recursive;cnt:40000000;lt:140.189
cs;for;cnt:40000000;lt:14
cs;while;cnt:40000000;lt:24
cs;ForEach + lambda;cnt:40000000;lt:245
cs;ForEach + named function;cnt:40000000;lt:549
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:37
Java;while;cnt:40000000;lt:59
Java;forEach;cnt:40000000;lt:224
Java;ForEach + lambda;cnt:40000000;lt:153
Java;ForEach + named function;cnt:40000000;lt:1981
Java;Recursive;cnt:40000000;lt:109
```

```
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 241
Js;ForEach + named function;cnt: 40000000 ;lt: 233
Js;Recursive;cnt: 40000000 ;lt: 259
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   549
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   133
Py;For;cnt: 40000000 ;lt: 1131
Py;While;cnt: 40000000 ;lt: 1825
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 664
Py;ForEach + named function;cnt: 40000000 ;lt: 3704
Py;recursive;cnt: 40000000 ;lt: 10454

Cpp;while;cnt:40000000;lt:1.2
Cpp;do-while;cnt:40000000;lt:1.137
Cpp;for;cnt:40000000;lt:0.599
Cpp;ForEach + lambda;cnt:40000000;lt:1111.89
Cpp;ForEach + named function;cnt:40000000;lt:1113.95
Cpp;Recursive;cnt:40000000;lt:127.868
cs;for;cnt:40000000;lt:29
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:94
cs;ForEach + named function;cnt:40000000;lt:91
cs;Recursive;cnt:40000000;lt:93
Java;for;cnt:40000000;lt:23
Java;while;cnt:40000000;lt:36
Java;forEach;cnt:40000000;lt:203
Java;ForEach + lambda;cnt:40000000;lt:158
Java;ForEach + named function;cnt:40000000;lt:1969
Java;Recursive;cnt:40000000;lt:114
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 243
Js;ForEach + named function;cnt: 40000000 ;lt: 230
Js;Recursive;cnt: 40000000 ;lt: 261
Pas;for;cnt:40000000;lt:    90
Pas;while;cnt:40000000;lt:    60
Pas;forEach;cnt:40000000;lt:   576
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   126
Py;For;cnt: 40000000 ;lt: 1141
Py;While;cnt: 40000000 ;lt: 1847
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 52
Py;ForEach + named function;cnt: 40000000 ;lt: 3655
Py;recursive;cnt: 40000000 ;lt: 8668

Cpp;while;cnt:40000000;lt:4.09
Cpp;do-while;cnt:40000000;lt:2.544
Cpp;for;cnt:40000000;lt:6.484
Cpp;ForEach + lambda;cnt:40000000;lt:1115.32
```

```
Cpp;ForEach + named function;cnt:40000000;lt:1108.97
Cpp;Recursive;cnt:40000000;lt:110.164
cs;for;cnt:40000000;lt:16
cs;while;cnt:40000000;lt:31
cs;ForEach + lambda;cnt:40000000;lt:113
cs;ForEach + named function;cnt:40000000;lt:81
cs;Recursive;cnt:40000000;lt:89
Java;for;cnt:40000000;lt:51
Java;while;cnt:40000000;lt:63
Java;forEach;cnt:40000000;lt:220
Java;ForEach + lambda;cnt:40000000;lt:148
Java;ForEach + named function;cnt:40000000;lt:1965
Java;Recursive;cnt:40000000;lt:115
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 243
Js;ForEach + named function;cnt: 40000000 ;lt: 256
Js;Recursive;cnt: 40000000 ;lt: 253
Pas;for;cnt:40000000;lt:    90
Pas;while;cnt:40000000;lt:    66
Pas;forEach;cnt:40000000;lt:   538
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 1190
Py;While;cnt: 40000000 ;lt: 1833
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 270
Py;ForEach + named function;cnt: 40000000 ;lt: 4329
Py;recursive;cnt: 40000000 ;lt: 9364

Cpp;while;cnt:40000000;lt:16.581
Cpp;do-while;cnt:40000000;lt:10.326
Cpp;for;cnt:40000000;lt:17.516
Cpp;ForEach + lambda;cnt:40000000;lt:1101.75
Cpp;ForEach + named function;cnt:40000000;lt:1132.63
Cpp;Recursive;cnt:40000000;lt:135.849
cs;for;cnt:40000000;lt:26
cs;while;cnt:40000000;lt:31
cs;ForEach + lambda;cnt:40000000;lt:101
cs;ForEach + named function;cnt:40000000;lt:90
cs;Recursive;cnt:40000000;lt:88
Java;for;cnt:40000000;lt:40
Java;while;cnt:40000000;lt:60
Java;forEach;cnt:40000000;lt:196
Java;ForEach + lambda;cnt:40000000;lt:201
Java;ForEach + named function;cnt:40000000;lt:1931
Java;Recursive;cnt:40000000;lt:108
Js;for;cnt: 40000000 ;lt: 35
Js;While;cnt: 40000000 ;lt: 32
Js;ForEach + lambda function;cnt: 40000000 ;lt: 251
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 260
Pas;for;cnt:40000000;lt:    76
Pas;while;cnt:40000000;lt:    65
Pas;forEach;cnt:40000000;lt:   561
```

```
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   126
Py;For;cnt: 40000000 ;lt: 2125
Py;While;cnt: 40000000 ;lt: 1330
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 2215
Py;ForEach + named function;cnt: 40000000 ;lt: 4767
Py;recursive;cnt: 40000000 ;lt: 9615

Cpp;while;cnt:40000000;lt:9.26
Cpp;do-while;cnt:40000000;lt:7.192
Cpp;for;cnt:40000000;lt:2.262
Cpp;ForEach + lambda;cnt:40000000;lt:1122.75
Cpp;ForEach + named function;cnt:40000000;lt:1137.29
Cpp;Recursive;cnt:40000000;lt:123.278
cs;for;cnt:40000000;lt:18
cs;while;cnt:40000000;lt:16
cs;ForEach + lambda;cnt:40000000;lt:113
cs;ForEach + named function;cnt:40000000;lt:111
cs;Recursive;cnt:40000000;lt:78
Java;for;cnt:40000000;lt:54
Java;while;cnt:40000000;lt:48
Java;forEach;cnt:40000000;lt:230
Java;ForEach + lambda;cnt:40000000;lt:161
Java;ForEach + named function;cnt:40000000;lt:1995
Java;Recursive;cnt:40000000;lt:103
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 245
Js;ForEach + named function;cnt: 40000000 ;lt: 235
Js;Recursive;cnt: 40000000 ;lt: 256
Pas;for;cnt:40000000;lt:   102
Pas;while;cnt:40000000;lt:    53
Pas;forEach;cnt:40000000;lt:   564
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   134
Py;For;cnt: 40000000 ;lt: 1201
Py;While;cnt: 40000000 ;lt: 2038
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 578
Py;ForEach + named function;cnt: 40000000 ;lt: 4587
Py;recursive;cnt: 40000000 ;lt: 8701

Cpp;while;cnt:40000000;lt:12.72
Cpp;do-while;cnt:40000000;lt:9.18
Cpp;for;cnt:40000000;lt:1.849
Cpp;ForEach + lambda;cnt:40000000;lt:1101.42
Cpp;ForEach + named function;cnt:40000000;lt:1105.52
Cpp;Recursive;cnt:40000000;lt:124.376
cs;for;cnt:40000000;lt:4
cs;while;cnt:40000000;lt:10
cs;ForEach + lambda;cnt:40000000;lt:124
cs;ForEach + named function;cnt:40000000;lt:105
cs;Recursive;cnt:40000000;lt:92
Java;for;cnt:40000000;lt:53
```

```
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:220
Java;ForEach + lambda;cnt:40000000;lt:167
Java;ForEach + named function;cnt:40000000;lt:1988
Java;Recursive;cnt:40000000;lt:93
Js;for;cnt: 40000000 ;lt: 37
Js;While;cnt: 40000000 ;lt: 32
Js;ForEach + lambda function;cnt: 40000000 ;lt: 244
Js;ForEach + named function;cnt: 40000000 ;lt: 230
Js;Recursive;cnt: 40000000 ;lt: 259
Pas;for;cnt:40000000;lt:    87
Pas;while;cnt:40000000;lt:    53
Pas;forEach;cnt:40000000;lt:   537
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   138
Py;For;cnt: 40000000 ;lt: 1129
Py;While;cnt: 40000000 ;lt: 2036
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 246
Py;ForEach + named function;cnt: 40000000 ;lt: 2955
Py;recursive;cnt: 40000000 ;lt: 9146

Cpp;while;cnt:40000000;lt:2.277
Cpp;do-while;cnt:40000000;lt:11.445
Cpp;for;cnt:40000000;lt:14.63
Cpp;ForEach + lambda;cnt:40000000;lt:1127.67
Cpp;ForEach + named function;cnt:40000000;lt:1130.66
Cpp;Recursive;cnt:40000000;lt:118.777
cs;for;cnt:40000000;lt:17
cs;while;cnt:40000000;lt:14
cs;ForEach + lambda;cnt:40000000;lt:93
cs;ForEach + named function;cnt:40000000;lt:89
cs;Recursive;cnt:40000000;lt:95
Java;for;cnt:40000000;lt:45
Java;while;cnt:40000000;lt:58
Java;forEach;cnt:40000000;lt:216
Java;ForEach + lambda;cnt:40000000;lt:175
Java;ForEach + named function;cnt:40000000;lt:1961
Java;Recursive;cnt:40000000;lt:99
Js;for;cnt: 40000000 ;lt: 37
Js;While;cnt: 40000000 ;lt: 32
Js;ForEach + lambda function;cnt: 40000000 ;lt: 248
Js;ForEach + named function;cnt: 40000000 ;lt: 259
Js;Recursive;cnt: 40000000 ;lt: 258
Pas;for;cnt:40000000;lt:    79
Pas;while;cnt:40000000;lt:    68
Pas;forEach;cnt:40000000;lt:   528
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 3913
Py;While;cnt: 40000000 ;lt: 1955
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 262
Py;ForEach + named function;cnt: 40000000 ;lt: 3428
Py;recursive;cnt: 40000000 ;lt: 9675
```

```
Cpp;while;cnt:40000000;lt:6.028
Cpp;do-while;cnt:40000000;lt:1.992
Cpp;for;cnt:40000000;lt:3.412
Cpp;ForEach + lambda;cnt:40000000;lt:1157.15
Cpp;ForEach + named function;cnt:40000000;lt:1125.75
Cpp;Recursive;cnt:40000000;lt:130.32
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:16
cs;ForEach + lambda;cnt:40000000;lt:99
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:90
Java;for;cnt:40000000;lt:53
Java;while;cnt:40000000;lt:50
Java;forEach;cnt:40000000;lt:218
Java;ForEach + lambda;cnt:40000000;lt:164
Java;ForEach + named function;cnt:40000000;lt:1934
Java;Recursive;cnt:40000000;lt:111
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 258
Js;ForEach + named function;cnt: 40000000 ;lt: 243
Js;Recursive;cnt: 40000000 ;lt: 256
Pas;for;cnt:40000000;lt:    78
Pas;while;cnt:40000000;lt:    65
Pas;forEach;cnt:40000000;lt:   559
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   137
Py;For;cnt: 40000000 ;lt: 1207
Py;While;cnt: 40000000 ;lt: 1839
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 258
Py;ForEach + named function;cnt: 40000000 ;lt: 3196
Py;recursive;cnt: 40000000 ;lt: 9268

Cpp;while;cnt:40000000;lt:0.358
Cpp;do-while;cnt:40000000;lt:2.786
Cpp;for;cnt:40000000;lt:3.322
Cpp;ForEach + lambda;cnt:40000000;lt:1107.68
Cpp;ForEach + named function;cnt:40000000;lt:1109.93
Cpp;Recursive;cnt:40000000;lt:149.029
cs;for;cnt:40000000;lt:23
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:96
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:87
Java;for;cnt:40000000;lt:37
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:218
Java;ForEach + lambda;cnt:40000000;lt:160
Java;ForEach + named function;cnt:40000000;lt:1983
Java;Recursive;cnt:40000000;lt:96
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 243
```

```
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 259
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    68
Pas;forEach;cnt:40000000;lt:   556
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   133
Py;For;cnt: 40000000 ;lt: 1177
Py;While;cnt: 40000000 ;lt: 1822
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 180
Py;ForEach + named function;cnt: 40000000 ;lt: 3663
Py;recursive;cnt: 40000000 ;lt: 9222

Cpp;while;cnt:40000000;lt:5.169
Cpp;do-while;cnt:40000000;lt:3.163
Cpp;for;cnt:40000000;lt:0.998
Cpp;ForEach + lambda;cnt:40000000;lt:1135.57
Cpp;ForEach + named function;cnt:40000000;lt:1123.54
Cpp;Recursive;cnt:40000000;lt:129.678
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:21
cs;ForEach + lambda;cnt:40000000;lt:79
cs;ForEach + named function;cnt:40000000;lt:137
cs;Recursive;cnt:40000000;lt:90
Java;for;cnt:40000000;lt:54
Java;while;cnt:40000000;lt:54
Java;forEach;cnt:40000000;lt:235
Java;ForEach + lambda;cnt:40000000;lt:161
Java;ForEach + named function;cnt:40000000;lt:1965
Java;Recursive;cnt:40000000;lt:157
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 243
Js;ForEach + named function;cnt: 40000000 ;lt: 237
Js;Recursive;cnt: 40000000 ;lt: 255
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    60
Pas;forEach;cnt:40000000;lt:   536
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 100
Py;While;cnt: 40000000 ;lt: 1208
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 498
Py;ForEach + named function;cnt: 40000000 ;lt: 3102
Py;recursive;cnt: 40000000 ;lt: 9127

Cpp;while;cnt:40000000;lt:3.758
Cpp;do-while;cnt:40000000;lt:2.042
Cpp;for;cnt:40000000;lt:6.67
Cpp;ForEach + lambda;cnt:40000000;lt:1093.56
Cpp;ForEach + named function;cnt:40000000;lt:889.726
Cpp;Recursive;cnt:40000000;lt:236.729
cs;for;cnt:40000000;lt:12
```

```
cs;while;cnt:40000000;lt:24
cs;ForEach + lambda;cnt:40000000;lt:264
cs;ForEach + named function;cnt:40000000;lt:141
cs;Recursive;cnt:40000000;lt:73
Java;for;cnt:40000000;lt:53
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:313
Java;ForEach + lambda;cnt:40000000;lt:197
Java;ForEach + named function;cnt:40000000;lt:2064
Java;Recursive;cnt:40000000;lt:106
Js;for;cnt: 40000000 ;lt: 38
Js;While;cnt: 40000000 ;lt: 32
Js;ForEach + lambda function;cnt: 40000000 ;lt: 246
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 264
Pas;for;cnt:40000000;lt:    90
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   558
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 1199
Py;While;cnt: 40000000 ;lt: 1826
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 285
Py;ForEach + named function;cnt: 40000000 ;lt: 3359
Py;recursive;cnt: 40000000 ;lt: 11310

Cpp;while;cnt:40000000;lt:4.049
Cpp;do-while;cnt:40000000;lt:1.115
Cpp;for;cnt:40000000;lt:20.573
Cpp;ForEach + lambda;cnt:40000000;lt:1223.32
Cpp;ForEach + named function;cnt:40000000;lt:1120.09
Cpp;Recursive;cnt:40000000;lt:128.421
cs;for;cnt:40000000;lt:9
cs;while;cnt:40000000;lt:18
cs;ForEach + lambda;cnt:40000000;lt:130
cs;ForEach + named function;cnt:40000000;lt:99
cs;Recursive;cnt:40000000;lt:88
Java;for;cnt:40000000;lt:54
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:216
Java;ForEach + lambda;cnt:40000000;lt:190
Java;ForEach + named function;cnt:40000000;lt:1966
Java;Recursive;cnt:40000000;lt:106
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 32
Js;ForEach + lambda function;cnt: 40000000 ;lt: 239
Js;ForEach + named function;cnt: 40000000 ;lt: 236
Js;Recursive;cnt: 40000000 ;lt: 260
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    69
Pas;forEach;cnt:40000000;lt:   553
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   141
```

```
Py;For;cnt: 40000000 ;lt: 1048
Py;While;cnt: 40000000 ;lt: 791
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 248
Py;ForEach + named function;cnt: 40000000 ;lt: 3314
Py;recursive;cnt: 40000000 ;lt: 10663

Cpp;while;cnt:40000000;lt:7.188
Cpp;do-while;cnt:40000000;lt:0.537
Cpp;for;cnt:40000000;lt:12.518
Cpp;ForEach + lambda;cnt:40000000;lt:1152.84
Cpp;ForEach + named function;cnt:40000000;lt:1147.2
Cpp;Recursive;cnt:40000000;lt:127.19
cs;for;cnt:40000000;lt:28
cs;while;cnt:40000000;lt:24
cs;ForEach + lambda;cnt:40000000;lt:107
cs;ForEach + named function;cnt:40000000;lt:91
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:57
Java;while;cnt:40000000;lt:42
Java;forEach;cnt:40000000;lt:218
Java;ForEach + lambda;cnt:40000000;lt:185
Java;ForEach + named function;cnt:40000000;lt:1984
Java;Recursive;cnt:40000000;lt:111
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 250
Js;ForEach + named function;cnt: 40000000 ;lt: 241
Js;Recursive;cnt: 40000000 ;lt: 281
Pas;for;cnt:40000000;lt:    78
Pas;while;cnt:40000000;lt:    60
Pas;forEach;cnt:40000000;lt:   550
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   120
Py;For;cnt: 40000000 ;lt: 1062
Py;While;cnt: 40000000 ;lt: 2101
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 293
Py;ForEach + named function;cnt: 40000000 ;lt: 3142
Py;recursive;cnt: 40000000 ;lt: 9430

Cpp;while;cnt:40000000;lt:9.256
Cpp;do-while;cnt:40000000;lt:1.018
Cpp;for;cnt:40000000;lt:1.817
Cpp;ForEach + lambda;cnt:40000000;lt:1123.14
Cpp;ForEach + named function;cnt:40000000;lt:1113.97
Cpp;Recursive;cnt:40000000;lt:129.866
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:22
cs;ForEach + lambda;cnt:40000000;lt:104
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:100
Java;for;cnt:40000000;lt:60
Java;while;cnt:40000000;lt:53
Java;forEach;cnt:40000000;lt:234
Java;ForEach + lambda;cnt:40000000;lt:162
```

```
Java;ForEach + named function;cnt:40000000;lt:1955
Java;Recursive;cnt:40000000;lt:111
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 240
Js;ForEach + named function;cnt: 40000000 ;lt: 234
Js;Recursive;cnt: 40000000 ;lt: 259
Pas;for;cnt:40000000;lt:    78
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   564
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 1153
Py;While;cnt: 40000000 ;lt: 1945
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 253
Py;ForEach + named function;cnt: 40000000 ;lt: 3457
Py;recursive;cnt: 40000000 ;lt: 8972

Cpp;while;cnt:40000000;lt:6.794
Cpp;do-while;cnt:40000000;lt:0.863
Cpp;for;cnt:40000000;lt:2.873
Cpp;ForEach + lambda;cnt:40000000;lt:1238.21
Cpp;ForEach + named function;cnt:40000000;lt:1205.92
Cpp;Recursive;cnt:40000000;lt:159.03
cs;for;cnt:40000000;lt:27
cs;while;cnt:40000000;lt:22
cs;ForEach + lambda;cnt:40000000;lt:167
cs;ForEach + named function;cnt:40000000;lt:180
cs;Recursive;cnt:40000000;lt:98
Java;for;cnt:40000000;lt:58
Java;while;cnt:40000000;lt:58
Java;forEach;cnt:40000000;lt:272
Java;ForEach + lambda;cnt:40000000;lt:201
Java;ForEach + named function;cnt:40000000;lt:2075
Java;Recursive;cnt:40000000;lt:110
Js;for;cnt: 40000000 ;lt: 41
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 290
Js;ForEach + named function;cnt: 40000000 ;lt: 246
Js;Recursive;cnt: 40000000 ;lt: 301
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   631
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   156
Py;For;cnt: 40000000 ;lt: 1375
Py;While;cnt: 40000000 ;lt: 2309
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 440
Py;ForEach + named function;cnt: 40000000 ;lt: 4063
Py;recursive;cnt: 40000000 ;lt: 13387
```

# A Nemes Tihamér Programozási verseny témaköreiről készült syllabus

Nikházy László

laszlo.nikhazy@gmail.com
ELTE IK

**Absztrakt.** A Nemzetközi Informatikai Diákolimpia (IOI) feladatait egy nemzetközileg elfogadott útmutató alapján tűzik ki. Ez az IOI Syllabus, amely tartalmazza, hogy milyen témakörök lehetnek, illetve nem lehetnek a versenyen. A Syllabus nagyon sokat segít a versenyre való felkészülésben is. Hasznos lenne egy ilyen dokumentum a Nemes Tihamér NITV Programozás kategóriájában mindhárom korcsoportnak. A korábbi évek feladatait rendszerezve javaslatot teszünk a „Nemes Tihamér Syllabus" tartalmára, és azt közzétesszük a szakmai közösség számára megvitatás céljából – abban bízva, hogy a 2020/2021-es tanévre kialakul hivatalos formában is. Részletezzük, hogy milyen módszerrel készült a jelenlegi javaslat, és statisztikák alapján elemezzük a korábbi versenyek anyagát.

**Kulcsszavak:** programozási verseny, algoritmusok, adatszerkezetek, versenyfelkészülés, syllabus

## 1. Bevezetés

Jelen cikk arról szól, hogy mi alapján készült a bemutatott Nemes Tihamér Syllabus (továbbiakban NT Syllabus) jelenlegi verziója, ami pillanatnyilag egy javaslat. Ez a dokumentum elérhető nyilvánosan [1]. A GitHub platform lehetőséget ad bárki számára, hogy hozzászóljon a tartalmához, és célunk, hogy a feladatkitűzőkkel és a felkészítő tanárokkal való eszmecsere útján, szakmai konszenzussal alakuljon ki a következő évre érvényes verzió.

Az NT Syllabus az IOI Syllabus [2] mintájára készül. Ha eléri célját, egy hivatalos dokumentum alakul ki belőle, amelynek aktuális verzióját a versenybizottság hagyja jóvá és teszi közzé minden évben. Mivel az informatika, és ezen belül a versenyprogramozás is folyamatosan fejlődik, ezt követni szeretnénk, vagyis az évek során a dokumentum változhat, fejlődhet. Minden évben a dokumentum gazdája felelős a dokumentum frissítéséért, amelynek a fent említett konszenzussal kell megvalósulnia.

A cikk felépítése a következő. A 2. fejezetben megfogalmazzuk, hogy milyen célok betöltésére hoztuk létre az NT Syllabus-t. A 3. fejezetben részletezzük, hogy milyen kategóriákat alkalmaz a dokumentum, amelyek szükségesek a további megértéshez. A 4. fejezetben leírjuk, hogy milyen módszert alkalmazva jött létre a jelenlegi verzió. Az 5. fejezetben bemutatjuk a választott formátumot, és egy részletet a dokumentumból a példaként a tartalmára is. A 6. fejezetben a Syllabus írása során elemzett feladatminta segítségével fogalmazunk meg észrevételeket az elmúlt évek versenyfordulóinak témaköreivel kapcsolatban.

## 2. Célok

Az NT Syllabus-nak három fő célja van:

- Meghatározza a versenyre szükséges előzetes tudást.
- Segít a versenyzők felkészülésében, a tanároknak a felkészítés tervezésében.
- A feladat kitűzőknek támpontot ad, hogy egy-egy feladat melyik kategóriában lehet.

A fenti célok elérése érdekében az NT Syllabus-ban megtalálható minden olyan téma (algoritmusok, adatszerkezetek, módszerek, nyelvi eszközök), amely egyáltalán szóba jöhet a versenyen, és ezeket kategóriákba soroljuk. A kategóriák tükrözik azt, hogy milyen módon fordulhat elő egy témakör, ezt a következő fejezet részletezi.

Az NT Syllabus jelen verziója kifejezetten a 2. és 3. forduló (gépes) feladatairól szól. Az első fordulóban szélesebb körből fordulhatnak elő feladatok, mert ott elvárt egy-egy (esetleg ismeretlen) számítástechnikai témakör alapszintű megértése a feladatleírás alapján.

## 3. Témák kategorizálása

Az NT Syllabus az IOI Syllabus-hoz hasonló kategóriákat alkalmaz, az alábbiakban felsoroljuk, majd bővebben magyarázzuk őket. A verseny három korcsoportjában különböző kategóriába sorolhatók a témakörök.

| Jelölés | Kategória |
|---|---|
| ✓ | Lehet, megkötések nélkül |
| ✓📄 | Lehet, de magyarázandó |
| ✓📄 | Lehet, de a feladatleírásban nem |
| ? | Hatáskörön kívüli |
| ✗? | Nem lehet, de nyitott a megvitatásra |
| ✗ | Kizárt |

**1. táblázat:** Témakörök kategóriái

✓ **Lehet, megkötések nélkül**

Ebben a kategóriában lévő témakörök előzetes tudásnak tekintendők. A versenyzőktől elvárt az ismeretük. A feladatleírásban magyarázat nélkül előfordulhatnak.

✓📄 **Lehet, de magyarázandó**

A versenyzőknek ismerniük kell ezt a témakört, de amikor feladatleírásban szerepel, akkor definiálni kell. Általában akkor használjuk ezt a besorolást, ha egy ✓ témát többféleképpen is lehet értelmezni.

✓📄 **Lehet, de a feladatleírásban nem**

Olyan témakörök sorolhatók ide, amelyeknek ismeretére a versenyzőknek a feladatmegoldás során van szüksége. Tehát a feladat szövegében nem szerepelhetnek.

Egy ilyen téma például az *Aszimptotikus felső becslés a komplexitásra*. Ez egy központi téma a verseny kapcsán, hiszen egy helyes program az algoritmus komplexitása alapján kap pontot. Mégsem fordulhat elő a feladatleírásban a komplexitás fogalma, a precíz definíció ismerete nem is elvárt.

? **Hatáskörön kívül**

Bármilyen téma, amit nem említ a Syllabus, ebbe a kategóriába esik. A versenyzőktől nem elvárt, hogy ismerjék ezeket. A legtöbb versenyfeladat nem kapcsolódik ilyen témakörökhöz. De ez nem zárja ki, hogy a versenyen legyen olyan feladat, ami ilyen témakörhöz kapcsolódik. A versenybizottság kitűzhet ilyen feladatot a verseny színesítése érdekében, de ebben az esetben is megoldhatónak kell lennie a fenti kategóriákba eső ismeretekkel.

✗? **Nem lehet, de nyitott a megvitatásra**

A Syllabus az évek során változik, fejlődik a versennyel együtt, így lehetőség adódik arra, hogy egy (valamely korcsoportban) kizárt téma később bekerüljön a verseny anyagába. Általában ezek a témakörök kapcsolódnak megengedett témakörökhöz, és nehéz határt szabni. Ebben az esetben a megvitatásra nyitott kategóriába kerülnek ezek a témakörök, ezzel is bíztatva a szakmai közösséget a visszajelzésre a kérdést illetően. Ha bekerül az idők során egy témakör, akkor a kizárt kategóriából először ebbe kell kerülnie.

✗ **Kizárt**

Ebbe a kategóriákba főként nehéz algoritmikus módszerek, bonyolult matematikai fogalmak tartoznak. Garantált, hogy nem lesz olyan feladat a versenyen, amelynek megoldásához *elengedhetetlen* egy ilyen témakör ismerete. Szintén kívánatos, hogy ezeknek a témáknak az ismerete ne nyújtson utat egyszerűbb, vagy több pontot érő megoldáshoz. Ugyanakkor előfordulhat, hogy egy feladat témaköre kapcsolódik egy ilyen témához, de ennek ismerete nem segít lényegesen a megoldásban.

## 4. Elemzési módszer

A következőkben leírjuk, hogy az NT Syllabus jelenlegi, első javaslat verziója milyen módszerrel alakult ki. A dokumentum alapját az IOI Syllabus képezi, viszont ezt jelentősen testre szabtuk a magyar versenyekre. A kategóriák megegyeznek az ottaniakkal.

A témakörök halmazának kialakításakor felsoroltuk az IOI Syllabus témaköreit (magyarra fordítva), és kiszűrtük belőlük azokat, amelyek túl messze állnak a magyar versenyek hatáskörétől, valamint azokat, amelyek készségekre (például hibakeresés) és eszközök használatára (például fejlesztőkörnyezet) vonatkoznak. Tehát az NT Syllabus az IOI Syllabus-hoz képest erősen rövidített, az áttekinthetőség kedvéért. Főként a konkrét tárgyi ismereteket és módszereket sorolja fel. Emellett a témaköröket kiegészítettük olyan elemekkel, amelyeket a magyar versenyeken gyakran előfordulnak, viszont az IOI Syllabus nem tér ki rájuk specifikusan, nem szerepel benne külön témakörként. Néhány esetben az IOI Syllabus egy-egy témakörét fel kellett bontanunk többre.

A fenti lépés után a témaköröket kategóriákba kellett sorolnunk mindhárom korcsoportra különkülön. Ehhez az utóbbi évekből feladataiból egy reprezentatív mintát vettünk, korcsoportonként legalább 50 feladatot (a harmadik, OKTV korcsoport esetében 80 feladatot). Készítettünk egy hatalmas megosztott táblázatot [3], amelynek oszlopaiban minden azonosított témakör szerepel, a soraiban pedig a vizsgált feladatok. Minden feladatot elemeztünk a feladatleírás és a helyes megoldások szempontjából, és bejelöltük, hogy mely ismeretetek szükségesek hozzá. Ebben nagy segítségünkre voltak tapasztalt versenyző diákok. A táblázat kitöltése közben is azonosítottunk még újabb témaköröket, amelyeket hozzávettünk az NT Syllabus felsorolásához.

A táblázatban kialakult statisztika és a versenykiírás [4] alapján minden témakörre eldöntöttük, hogy az egyes korcsoportokban mely kategóriába tartozzon. Hangsúlyozzuk, hogy ez csak egy javaslat, egy kiindulás a szakmai közösség számára. Az „igazi" változat hosszas egyeztetések után alakul ki, a versenybizottsággal, a felkészítő tanárokkal, illetve akár versenyzőkkel. Végül a versenybizottság dönti el, hogy tényleg felhasználja-e ezt a dokumentumot a verseny hivatalos dokumentumaként.

## 5. Formátum, példa

Az alábbi képen látható egy részlet az NT Syllabus jelenlegi (2019. 11. 16.) változatából. Táblázatos formában jelenítjük meg, hogy az egyes témakörök a három korcsoportban milyen kategóriába tar-

toznak. Ezzel áttekinthető az is, ahogyan nehezedik a verseny az idősebbek számára. Kialakítható belőle egy „tanterv” a versenyfelkészítésre vonatkozóan.

**AL3a. Algoritmusok**

| NT1 | NT2 | OKTV | Leírás |
|---|---|---|---|
| ✓ | ✓ | ✓ | Egyszerű számelméleti algoritmusok: számrendszer átváltás, Euklideszi algoritmus (LNKO-ra), prímteszt $\mathcal{O}(\sqrt{n})$ osztókereséssel, Eratosztenészi szita, prímfelbontés (osztókereséssel vagy szitával) |
| ✗ | ✗? | ✓ | Gyors hatványozás (négyzetre emelésekkel) |
| ✗ | ✗? | ✓ | Műveletek tetszőlegesen nagy egész számokkal (összeadás, kivonás, szorzás) |
| ✓ | ✓ | ✓ | Egyszerű programozási tételek tömbökön: összegzés, megszámolás, keresés, minimum/maximum, kiválogatás |
| ✓ | ✓ | ✓ | Programozási tételek összeépítése, pl. feltételes maximum |
| ✓ | ✓ | ✓ | Rendezett sorozatok összefésülése, metszet, unió |
| ✓ | ✓ | ✓ | Egyszerű string algoritmusok (pl. minta keresése naiv módszerrel) |
| ✓ | ✓ | ✓ | $\mathcal{O}(n^2)$ rendezések (buborék, leszámláló, minimum-kiválasztásos) |
| ✗? | ✓ | ✓ | Gyorsrendezés (quicksort), *NT1-ben a sort() függvény használata elvárt, de az algoritmus ismerete nem* |
| ✗ | ✓ | ✓ | $\mathcal{O}(n \log n)$ rendezések (kupac, összefésüléses) |
| ✗ | ✗? | ✓ | Lineáris idejű rendezések (láda/vödör rendezés, radix rendezés) [3] |
| ✗? | ✓ | ✓ | Rekurzív fabejárás |
| ✗ | ✓ | ✓ | Rendezett fák bejárásai (pre, in- és post-order) |
| ✗ | ✓ | ✓ | Szélességi és mélységi gráfbejárás |
| ✗ | ✓ | ✓ | Összefüggő komponensek meghatározása |
| ✗ | ✓ | ✓ | A mélységi feszítőfa alkalmazásai, például topologikus sorrend |
| ✗ | ✓ | ✓ | Irányított körmentes gráf emeletekre bontása |

**1. ábra:** Részlet az NT Syllabus tartalmából

## 6. Elemzési eredmények, észrevételek

Egy nagyon hasznos kimenete az elemzésnek a feladatokból készült megosztott táblázat [3], amely nem csak statisztikaként szolgál, az egyes témakörökre való célzott felkészülést is segíti, hiszen lehet a témakörhöz tartozó feladatokat választani ez alapján. Tervezzük, hogy bővítjük az elemzett feladatok körét még további, korábbi versenyfordulókkal. A táblázatból kiolvasható statisztika alapján a következő fejezetekben megmutatjuk, hogy korcsoportonként melyik az a néhány témakör, ami a

legtöbb feladatban szerepel. Hangsúlyozzuk, hogy az alábbi táblázatok csak néhány kiemelt témát mutatnak, a teljes statisztika az online táblázatban [3] látható.

A táblázatokban feltüntetjük, hogy hozzávetőlegesen a feladatok mekkora hányadához szükséges egy-egy témakör, kerekített értékekkel, hiszen a minta méretéből adódóan 4-5 % pontatlanság mindenképp előfordulhat. Megjegyzés ezzel kapcsolatban, hogy minden feladatnál a legfontosabb módszereket, algoritmusokat, adatszerkezeteket jelöltük meg, amelyek lényegében a feladat témájának tekinthetők. Bizonyos alapvető algoritmusok (például maximumkiválasztás) nagyon sok más, bonyolultabb algoritmusnak és módszernek velejárói, azoknál nem jelöltük be őket. Ezeket az alapvető algoritmusokat akkor jelöltük meg, ha a feladat erről szól, vagy legalábbis a megoldásnak egy nagyon lényeges hányadát képezi ez.

## 6.1. Az 1. korcsoport leggyakoribb témakörei

Az alábbi táblázatból látható, hogy az 5-8. osztályosoknál lényegében nem szerepelnek összetettebb, előre elsajátítandó „tankönyvi" algoritmusok. A feladatok mintegy 65%-a megoldható az úgynevezett programozási tételek [5] alkalmazásával. Ezek egyszerű algoritmus sablonok, amelyeket a diákok sok esetben a tudatosításuk nélkül is képesek alkalmazni. A feladatok nagyjából 35%-ához kell valamilyen haladóbb algoritmikus stratégia használata. Ezek rendszerint valamilyen komolyabb ötletet igénylő, nehezebb feladatok. Mivel fordulónként 3-4 feladat van, ezért a versenyzők számíthatnak arra, hogy a második fordulóban egy, a döntőben egy vagy két ilyen feladat lesz. Az utóbbi években a mezőny fejlődése miatt a döntőben már általában négy feladat van, ez esetben átlagosan két feladat várható ebből a típusból.

Adatszerkezetek között is döntő többségben csak nagyon egyszerűek szerepelnek a versenyen, természetesen ide tartozik a tömb, ritkább esetben a kétdimenziós tömb. Némileg alulreprezentált a fontosságához képest a karakterlánc (string), amely nem is feltétlenül van minden évben. Egy fontos kiemelendő viszont a hisztogram, másnéven (a szerző által bevezetve) „számláló tömb", amikor egy tömb elemeiben számoljuk meg, hogy az egyes értékek hányszor szerepelnek egy sorozatban.

| Algoritmikus stratégiák | |
|---|---|
| Mohó algoritmusok | 15% |
| Két mutató technika | 10% |
| Dinamikus programozás | 5% |
| Szimuláció | 5% |
| **Adatszerkezetek** | |
| Tömb (egy dimenziós) | 55% |
| Hisztogram, számláló tömb | 20% |
| Karakterlánc (string) | 15% |
| Két dimenziós tömb | 10% |
| **Algoritmusok** | |
| Minimum-, maximumkiválasztás | 30% |
| Programozási tételek összeépítése | 15% |
| Keresés, eldöntés | 10% |
| Számelméleti algoritmusok | 10% |
| Matematika | |
| Számrendszerek, mértékegység átváltások | 20% |
| Műveletek madadékokkal | 20% |
| Legnagyobb közös osztó | 10% |

**2. táblázat:** Az 1. korcsoport leggyakoribb témakörei

A matematika témaköréből kitüntetett figyelmet érdemel a mértékegység átváltások, illetve számrendszerek témaköre. (Azért soroljuk őket egybe, mert szinte ugyanazok az eszközök kellenek hozzájuk, a mértékek is egyfajta, vegyes alapú számrendszernek tekinthetők.) A versenybizottság előszeretettel tűz ki ilyen feladatokat, minden évben várható egy vagy kettő. A maradékos műveletek is nagyon fontosak, egyrészt a mértékegységekhez, számrendszerekhez is kell, másrészt szintén nem elhanyagolható mértékben vannak számelméleti algoritmusok a versenyen (például legnagyobb közös osztó).

## 6.2. A 2. korcsoport leggyakoribb témakörei

A 9-10. osztályosok esetében a legfontosabb különbséget a dinamikus programozás „térhódítása" és a gráfalgoritmusok megjelenése okozza. Természetesen a többi témakörből (például mohó algoritmusokból) is jóval nehezebb feladatok vannak, mint az 1. kategóriában. Az alábbi táblázatban megjelenített fa-, és gráfbejárási algoritmusok részesedése között némi átfedés van (mert bizonyos feladatoknak többféle egyenrangú megoldása van), de a táblázat áttekintésével kijelenthetjük, hogy a feladatok legalább egyharmada kapcsolódik a gráfokhoz. A gyakorlatban azt jelenti ez, hogy a második fordulóban egy, a harmadikban két ilyen feladatra lehet számítani. Egy másik nagyon fontos fejlemény, hogy a rendezés használatának mindenképp a versenyző eszköztárában kell lennie.

Az adatszerkezetekben való eltolódás is a fent említett okok miatt következik be. A gráfok szomszédsági listás reprezentálásához szükséges a nyújtózkodó tömb (C++ vector), illetve néhány egyéb helyzetben is hasznos. A karaktersorozatoknak változatlanul nem nagy a jelenléte, de mivel fordulónként 5-6 feladat van, ezért minden évben lehet számítani string-es feladatra.

A gráfok mellett megjelenik még egy nehéz matematikai téma, a kombinatorika. Ebbe nem értjük bele a gráfelméletet, vagyis döntő többségében leszámlálásokhoz köthető feladatokról van szó.

Ezzel párhuzamosan az 1. korcsoportban gyakori mértékegység átváltások sokkal ritkábban fordulnak elő.

| **Algoritmikus stratégiák** | |
|---|---|
| Dinamikus programozás | 20% |
| Két mutató technika | 15% |
| Mohó algoritmusok | 15% |
| Rekurzív kiszámítás | 5% |
| **Adatszerkezetek** | |
| Tömb (egy dimenziós) | 50% |
| Nyújtózkodó tömb | 30% |
| Gráfreprezentációk: szomszédsági lista | 30% |
| Két dimenziós tömb | 15% |
| Karakterlánc (string) | 10% |
| **Algoritmusok** | |
| Mélységi gráfbejárás | 20% |
| Minimum-, maximumkiválasztás | 15% |
| Rekurzív fabejárás | 10% |
| Rendezés | 10% |
| Szélességi gráfbejárás | 10% |
| Matematika | |
| Irányítatlan, súlyozatlan gráf | 10% |
| Irányított gráf | 10% |
| Fa gráf | 10% |
| Kombinatorika | 10% |

**3. táblázat:** A 2. korcsoport leggyakoribb témakörei

## 6.2. A 3. korcsoport, vagyis az Informatika OKTV leggyakoribb témakörei

A 11-12. osztályosok versenyében egy nagyon fontos és nehéz új témakör a geometriai algoritmusok. A statisztikából arra következtethetünk, hogy nagyjából évenként egy ilyen feladat van. Szintén fontos, hangsúlyos témakör a haladó gráfalgoritmusok, amibe sok algoritmust beleértünk, többek között a Dijkstra-algoritmust legrövidebb utak keresésére, Kosaraju algoritmusát erősen összefüggő komponensek meghatározására, Tarjan algoritmusát elvágó pontok és élek keresésére, Euler-séta keresését stb. Az optimalizálási problémák bináris kereséssel való megoldása is szignifikánsan jelenik meg. A statisztika alapján az látható, hogy alig van olyan feladat, amelynek megoldásához nem szükséges valamilyen összetettebb „tankönyvi" algoritmus, vagy pedig valamilyen algoritmikus stratégia használata.

Nagyjából minden feladathoz kell egy- vagy kétdimenziós több, vagy nyújtózkodó tömb. Fontos újdonság az előző korcsoportokhoz képest a prioritási sor gyakori alkalmazása. Feltüntettük még azt is, hogy sok feladathoz szükséges valamilyen egyszerűbb rekordot használni (C++ struct, pair stb.)

A gráfok között az irányított gráfok gyakoribbak, mint a fiatalabb korosztályban, és a súlyozott gráfok is szignifikáns részben fordulnak elő. A kombinatorika változatlanul jelen van, évenként

átlagban egy feladatban. Újdonságként jelenik meg a matematikai témakörök között a korábban említett mélyebb geometriai ismeretek, ami elsősorban koordináta-geometriát jelent.

| **Algoritmikus stratégiák** | |
|---|---|
| Mohó algoritmusok | 25% |
| Dinamikus programozás | 20% |
| Geometria: söprés, görgetés | 10% |
| Rekurzív kiszámítás | 5% |
| Szimuláció | 5% |
| Bináris keresés | 5% |
| **Adatszerkezetek** | |
| Tömb (egy dimenziós) | 60% |
| Nyújtózkodó tömb | 35% |
| Gráfreprezentációk: szomszédsági lista | 35% |
| Két dimenziós tömb | 20% |
| Egyszerű rekord | 15% |
| Prioritási sor | 10% |
| **Algoritmusok** | |
| Mélységi gráfbejárás és alkalmazásai | 20% |
| Rendezés | 20% |
| Szélességi gráfbejárás | 15% |
| Haladó gráfalgoritmusok | 10% |
| Matematika | |
| Irányított gráf | 15% |
| Irányítatlan, súlyozatlan gráf | 10% |
| Kombinatorika | 10% |
| Geometriai fogalmak és számítások | 10% |
| Fa gráf | 10% |
| Súlyozott gráf | 5% |

**4. táblázat:** Az Informatika OKTV (programozás kategória) leggyakoribb témakörei

## Köszönetnyilvánítás

A cikk és a Nemes Tihamér Syllabus elkészítéséhez nagyban hozzájárult a már sokszor hivatkozott táblázat az elmúlt évek feladatairól [3]. Ennek elkészítésében számos szakköri és különórai tanítványom segített: Bata Zsombor, Bukva Dávid, Csontos Dávid, Czanik Pál, Deák Bence, Fadgyas Péter, Gyimesi Péter, Harmati László, Hervay Bence, Horcsin Bálint, Horváth Boldizsár, Juhász-Molnár Erik, Kerekes Boldizsár, Máté Lőrinc, Molnár Bálint, Nagy Nándor, Noszály Áron, Reimann Kristóf, Sárvári Tibor, Soós Máté, Szabó Kornél, Székely Milán, Szilágyi Balázs, Tiszay Dávid, Tóth Balázs, Várkonyi Zsombor. Nekik ezúton is szeretném megköszönni a munkájukat.

Az elkészült Nemes Tihamér Syllabus javaslat pedig reményeim szerint nagy hasznára lesz a jövőben mindenkinek, aki valamilyen módon érintett a versenyben, és emeli a verseny minőségét és színvonalát.

## Irodalom


1. Nikházy László: *NT-Syllabus*, GitHub repository. https://github.com/niklaci/NT-Syllabus (utoljára megtekintve: 2019.11.16.)
2. M. Forišek: *IOI Syllabus*. (2018) https://ioinformatics.org/files/ioi-syllabus-2018.pdf (utoljára megtekintve: 2019.11.16.
3. *NT feladatok témakörökkel*, Google táblázat. http://bit.ly/nt-feladatok (utoljára megtekintve: 2019.11.16.)
4. *A Nemes Tihamér NITV tárgya, követelményei*. http://nemes.inf.elte.hu/index.html
5. Szlávi Péter, Zsakó László: *Módszeres programozás: programozási tételek*. Mikrológia 19, ELTE IK (2008)

# Az algoritmikus gondolkodás vizsgálata különböző korosztályú tanulóknál, E-learning tesztelési környezetben

Osztián Erika[1], Kátai Zoltán[2]

{[1]osztian, [2]katai_zoltan}@ms.sapientia.ro
SAPIENTIA EMTE

**Absztrakt.** A Sapientia Erdélyi Magyar Tudományegyetemen kezdeményezett többérzékszerves informatikaoktatási módszer új arculata az *AlgoRythmics* projektben mutatkozik meg. Ahhoz, hogy ez a környezet még motiválóbb és hatékonyabb legyen MINDEN tanuló algoritmikus gondolkodásának fejlesztésében, előteszteléseket végeztünk. Ennek eredményeit, következtetéseit szeretnénk ismertetni ebben a dolgozatban. A mérések alapeszközeként az AlgoRythmics kollekció lineáris keresés videóját használtuk és ennek hatását vizsgáltuk különböző *korosztályú* (3, 5, 7, 9 osztályok) és *szakirányú* (művészetek vs. reál), programozási előismeretekkel nem rendelkező diákok algoritmikus/számítógépes gondolkodására. A vizsgálatot kiterjesztettük nagyobb korosztályra is, így lehetőséget biztosítva az egyetemisták algoritmikus gondolkodásának mérésére is. Nekik a bináris keresést is el kellett sajátítaniuk. Az *E-learning* környezetben végzett mérések eredményei útmutatót adtak, hogy milyen további lépéseket tudnánk tenni az AlgoRythmics környezetet fejlesztésében, mely segítségével követhető legyen a tanulók fejlődése, és a megértés és kódolás közti szakadék is jelentősen csökkenne.

**Kulcsszavak:** algoritmusok, korosztály, nem, szakmai beállítottság, többérzékszerves oktatás

## 1. Bevezető

A számítógépes gondolkodás fogalma jónéhány éve a figyelem középpontjában van, az utóbbi években azonban egyre nagyobb hangsúlyt fektetnek arra, hogy hogyan lehetne mérni is azt. Számos tanulmány azt méri, hogy egy adott korosztályú tanulóknál milyen szintű a számítógépes gondolkodás. Mi, a Sapientia Erdélyi Magyar Tudományegyetem Matematika-Informatika Tanszék tanárai, egy olyan kutatásban veszünk részt, amelyben azt mérjük, hogy milyen ütemben fejlődik ez a készség különböző korosztályú tanulóknál. Ezért 3., 5., 7. és 9. osztályos tanulókat ugyanazon környezetben teszteltünk. Mivel az algoritmika projektünk a művészet és az informatika párosításából született meg, arra gondoltunk, hogy váll-váll mellett vizsgáljuk a művészeti és elméleti szakirányú tanulókat. A szakirodalomból merítve természetesen figyelembe vesszük, hogy a számítógépes gondolkodást fejleszteni kell már kisgyerekkorban. De felmerül az a kérdés, hogy MINDENKI számára fejleszteni kell és ha igen, HOGYAN fejlesszük ezt?

## 2. A számítógépes gondolkodást fejlesztő- és mérő oktatási környezet

Az utóbbi évtizedben folytonos erőfeszítések történtek annak az érdekében, hogy a számítógépes gondolkodás fejlesztését beépítsék a K-12, illetve a K-9 oktatásba. A *"Számítógépes Gondolkodás mindenkinek"* kezdeményezésnek köszönhetően, két komplementáris ötlet született, mely elősegítette ennek megvalósítását: (1) új számítástechnika tanfolyamok bevezetése és (2) a jelenlegi lecketervekbe a számítógépes gondolkodás fejlesztésének beépítése (Román-González, Pérez-González, Moreno-León, & Robles, 2018). Ezt követve, 2014 óta, a Számítástechnika oktatása az Egyesült Királyságban élő gyerekek számára kötelezővé vált 5 éves kortól kezdődően (Brown, Sentance, Crick, &

Humphreys, 2014) és ennek a tantárgynak a tanítása egyre növekvő elismerést élvez más országokban is (European Schoolnet, 2015). Egy másik kutatás (Mannial, 2014) azt vizsgálta, hogy jelen oktatási rendszerünkben milyen mértékben vannak jelen a számítógépes gondolkodást elősegítő eszközök. Az itt megfogalmazott gondolatok hallgatólagosan arra utalnak, hogy a jelenlegi tantervek nem járulnak elegendően hozzá a számítógépes gondolkodás fejlesztéséhez. Több kutatás nyomatékosítja, hogy a programozást tanuló hallgatók nem rendelkeznek azokkal az előzetes szakismeretekkel, amelyeket ez a tantárgy megkövetel (Ahadi, Lister, Lal, Leinonen, & Hellas, 2017; Evans & Simkin, 1989; Simon, Chen, Lewandowski, McCartney, & Sanders, 2006). Másrészt, amiatt hogy a számítógépes gondolkodás egy kombinált készség (Feaster, Ali, Zhai, & Hallstrom, 2014), arra következtethetnénk, hogy explicit számítógépes gondolkodás-oktatásra való különös figyelem nélkül is fejlődik a gyerekek számítógépes gondolkodása, a jelenlegi tantervnek köszönhetően. Jelen kutatásban is kifejezetten arra összpontosítottunk, hogy ha kérdéseket teszünk fel egy olyan feladat kapcsán, amely bizonyos szintű számítógépes gondolkodást igényel, van-e különbség a harmadik, ötödik, hetedik és kilencedik osztályos tanulók válaszai közt. Kutatásunkat bővítettük a 2019 őszén mért felmérések eredményeivel, melyet azon elsőéves egyetemisták körében végeztünk, akik nem rendelkeztek előzetes programozási ismeretekkel.

Az AlgoRythmics környezetben az algoritmusokat olyan kontextusban tálaljuk, amelyek bárki számára vonzóak. Tulajdonképpen a táncegyüttesekkel karöltve, olyan tánckoreográfiákat dolgoztunk ki, amelyek alapvető számítógépes algoritmusokat szemléltetnek. Ilyen algoritmusaink például a lineáris és bináris keresési algoritmus, amely kísérletünk kivitelezésére szolgált.

## 3. Kísérletek

A tervezett kísérletben résztvevő tanulók, a K-12 oktatás három szakaszában kell teljesítsenek: általános iskola alsó tagozat (0–4. évfolyam), általános iskola felső tagozat (5–8. évfolyam) és középiskola (9–12. évfolyam). A tanterv formája, mint az oktatás tartalmát szabályozó dokumentum, minden iskolán belül a különböző tanulmányi területek tanulmányozásának köszönhetően, lehet: elméleti, speciális szakmai (például művészeti) és technológiai. A számítógépes gondolkodást fejlesztő oktatásban, évek óta kötelező tárgyként, csak a középiskolai osztályokban (9-12 évfolyamoknál), ott is csak a természettudományok, a matematika informatika és az intenzív informatika szakosztályok számára szerepelt a tanterv keretében a programozás oktatás. A 2017-2018-as tanévtől kezdődően, a gimnáziumi szintű (az 5. osztály második felétől) az Informatika és Információs és kommunikációs technológiák név alatt szereplő tantárgy tantervét az algoritmikus-gondolkodást fejlesztő elmélettel is kiegészítették.

Jelen tanulmány résztvevőit 325 tanuló képviseli (42% lány), ebből 57 (46% lány) alsó tagozatos általános iskolás, 103 (63% lány) felső tagozatos általános iskolás, 54 (54% lány) középiskolás és 111 (20% lány) első éves egyetemista. A két tagozat tanulói a környék két, jó hírnevű állami iskolájában tanultak a 2018-2019-es tanévben. Mivel elemi szinten csak elméleti (T) és művészeti (A) szakos gyerekek tanulnak, az őket képviselt kiválasztott iskolák is ilyen irányítottságúak voltak (Elméleti iskola-T és Művészeti iskola-A). A 3., 5., 7., 9. fokozat nyolc osztályát kértük fel tehát a kísérletre, mindkét szakról kiválasztva egy-egy osztályt (3T, 3A, 5T, 5A, 7T, 7A, 9T, 9A). A Settle, Goldberg és Barr (2013) kutatók által végzett mérések arra utalnak, hogy a 3. osztály az a fokozat, amelyben a tanulók már nem olvasni tanulnak, hanem olvasással tanulnak. Ez lehet a legkisebb korosztály, akiknél az algoritmikus-gondolkodás, mint képesség, követhető. A további korosztályok kiválasztásakor azt is feltételeztük, hogy a kétéves eltolódás a felmért tanulók közt mérhető eredményeket szolgál. Ezek a feltevések harmonizálnak a CSTA[1] (2017) szabványok által meghatározott felosztással. Az

[1] CSTA – Computer Science Teachers Association

egyetemisták a Sapientia Erdélyi Magyar Tudományegyetem Informatika, Számítástechnika, Mechatronika és Gépészetmérnöki szakokon kezdték tanulmányaikat a 2019-2020-as tanévben.

## 3.1. A kísérlet leírása:

A kísérlet során az AlgoRythmics kollekció lineáris és bináris keresési videójával, valamint a világhálón megtalálható ezen algoritmusok animációjával tanítottunk különböző csoportokat. A méréseket a 2018-2019, valamint a 2019-2020-as tanévekben végeztük. Összefoglalva, a következő tanulási módszer alapján tanultak a tanulóink (1. táblázat: **Alkalmazott tanulási módszerek és algoritmusok**)

| **Tanterv** | **3, 5, 7, 9 osztályosok** | **Informatika C+D Számítástechnika C Mechatronika** | **Informatika A+B Számítástechnika A+B Gépészmérnöki** |
|---|---|---|---|
| **Algoritmus neve** | Lineáris keresés | Lineáris és bináris keresés | Lineáris és bináris keresés |
| **Tanulási módszer** | Videó | Videó | Animáció |

**1. táblázat:** Alkalmazott tanulási módszerek és algoritmusok

A kísérletet a 2018-19-es tanév első félévében kezdtük az alsó, felső és középiskolás tanulókkal. A tesztelés, a 3. osztályos tanulók kivételével, a résztvevő iskolák számítógépes laboratóriumában zajlott. Mivel a kisiskolások tantervei nem tartalmazzák az Informatika és Információs és kommunikációs technológiák tanórákat és a figyelmük lekötése is igencsak nehéz feladat, meghívtuk őket az egyetem számítógépes laboratóriumába, azt remélve, hogy az új környezet is segíteni fog a bemutatott algoritmusok megértésében.

Az egyetemisták felmérésére a 2019-2020-as tanévben, az Informatika alapok tantárgy keretén belül került sor. A felosztás szakonként, ezen belül pedig névsor szerint valósult meg.

Mind a kisiskolásoknak, középiskolásoknak, mind az egyetemistáknak ugyanazt a tananyagot tanítottuk. Tanulási eszközként használtunk szemléltető képeket, videókat, animációkat. Minden tanulási szakaszt egy-egy kérdés követett, melyre a választ a Socrative online tesztelési alkalmazásban értékeltünk ki. Az általunk összefoglalt tananyag 2 szakaszból állt. A lecke címe: *A párját kereső fiú története*. Minden csoport felmérése 30-35 percet vett igénybe. Az órából fennmaradó 10-15 percben együtt értelmeztük a diákokkal a feltett kérdésekre adott válaszaikat.

## 3.2. Első szakasz

Az első tanulási szakaszban meg szerettük volna tudni, hogy a tanulók mennyire éreznek rá a lineáris keresés algoritmusára. Segítségül egy kép szemléltette a párját kereső fiú történetét (1 fiú és 7 lány; a fiú a 7-es számot, a 7 lány pedig a következő számokat viselte: 3, 9, 0, 5, 8, 7, 4). A feladat forgatókönyve röviden megfogalmazva: a fiú keresi a párját a lányok közt. Az a párja, aki ugyanazt a számot viseli, és ugyanazt a „tükörkoreográfiát" táncolja. Mivel a fiú nem látja a lányok számát (a fiú a mellén, a lányok a hátukon viselik a számokat), a következő stratégiát kell, hogy alkalmazza: felkéri őket egy-egy táncra és tánc közben tudatosul benne, hogy a lány a párja-e vagy sem. (1. ábra: Lineáris keresés - felvezetőkérdés)


**1. ábra:** Lineáris keresés - felvezetőkérdés

A felmérés első kérdése:

*$Q_1$: Hány lánnyal kell táncolnia a fiúnak, ahhoz, hogy megtalálja a párját?*

**Helyes válaszoknak a lineáris keresés szerinti válaszokat fogadtuk el:** 6 vagy 2 (mivel a tanulók a lánysorozat bármelyik oldalától indíthatták a keresést)

## 3.3. Második szakasz


**2. ábra:** Lineáris keresés videóval

A tanulóknak megtanítottuk a lineáris keresés algoritmusát. Oktatási eszközként, az AlgoRythmics kollekció flamenco tánccal bemutatott lineáris keresés videóját alkalmaztuk. (ábra: **Lineáris keresés videóval**).

Ahhoz, hogy megvizsgáljuk mennyire értették meg az algoritmust, újabb kérdéseket tettünk fel számukra. Az első kérdés keretében egy újabb kép alapján (a fiú a 10-es számot, a 10 lány pedig a következő számokat viselte: 18, 22, 8 ,7 ,9 ,12 ,10 , 64 , 1, 19) kellett újra választ adjanak az alábbi kérdésre:

*Hány lánnyal kell táncolnia a fiúnak, ahhoz, hogy megtalálja a párját?*

**Helyes válaszok**: 7 (ha balról indul), 4 (ha jobbról indul)

Ezt követett négy, az algoritmus bonyolultságára vonatkozó kérdés. Az első két kérdés ezekből a következő ábrához kapcsolódott. Ezúttal a lányok hátán nem voltak feltüntetve a számok (3. ábra: **Milyen számokkal lássuk el a lányokat?**).


**3. ábra:** Milyen számokkal lássuk el a lányokat?

$Q_{2.2.1}$: *Milyen számokkal látnád el a lányokat, ahhoz, hogy a fiú egy tánc után megtalálja a párját? ("legszerencsésebb eset"). Sorold fel az 5 számot, szóközzel elválasztva.*

Mivel nem hangsúlyoztuk, hogy a fiú balról vagy jobbról kezdi a keresést, helyes válaszként elfogadtuk, ha a 25-ös szám a sorozat első pozícióján vagy a sorozat utolsó pozícióján volt megtalálható. Ugyancsak helyesek voltak azok a válaszok, amelyeknél a 25-ös szám a sorozat mindkét végén, vagy a megadott sorozat csupa 25-ös számokból állt.

**Egy lehetséges helyes válasz**: 25 14 89 4 3

$Q_{2.2.2}$: *Milyen számokkal látnád el a lányokat, ahhoz, hogy a fiú a "legszerencsétlenebb esetben" legyen? Sorold fel az 5 számot, szóközzel elválasztva.*

Ebben az esetben helyes válaszként a következő eseteket fogadtuk el: a 25-ös szám a sorozat utolsó pozícióján található, a 25-ös szám a sorozat első pozícióján található vagy a 25-ös szám nem volt megtalálható a sorozatban.

**Egy lehetséges helyes válasz:** 1 85 7 12 25

A feladat nehézsége az utolsó 2 kérdéssel fokozódott:

*$Q_{2.3.1}$: Hány lánnyal kell táncoljon a fiú „legszerencsésebb esetben", ahhoz, hogy megtalálja a párját, ha 5 lány helyett, x lány állna vele szemben?*

**Helyes válasz**: 1

*$Q_{2.3.2}$: Hány lánnyal kell táncoljon a fiú „legszerencsétlenebb esetben", ahhoz, hogy megtalálja a párját, ha 5 lány helyett, x lány állna vele szemben*?

**Helyes válasz**: x vagy x-1

Az egyetemisták számára a kérdések sorozatát kibővítettük a kódolást elősegítő kérdésekkel, melyek segítségével reméltük, hogy „megépül" a híd, a megértés és a kódolást elválasztó szakadék fölött. A tanár felvázolta az algoritmust leíró programrészletet, néhány hiányzó résszel. Ezeket kellett nekik kipótolniuk.

Kérdés: Feltételezzük, hogy n lány közül keresi a párját a fiú, aki a *fiú* számot viseli a hátán. Mit kell írni a pontok helyére? A lányok 0-tól vannak sorszámozva. A tömb azonosítója *lányok*. Ha a fiú megtalálja a párját, a program írja ki a lány *sorszámát*, különben a „*Nincs*" szót.

**Lineáris keresés**

```
int main()
 {
   int fiu, lanyok[100],n,i;
   //beolvasás
   for (i =  … ; i < … ; … )       //(Q3)
   {
      if(…==…) //(Q4)
      {
       cout << …; //(Q5)
       break;
      }
   }
   if(i == …) //(Q6)
          {cout << „Nincs";}
return 0;
}
```

**Bináris keresés**

```
int main()
{
int fiu, lanyok[100], n;
//beolvasás
int bal, jobb, kozep;
bal = 0; jobb = n-1;
while( ......... ) //(Q3)
{
      kozep = .....;  //(Q4)
      if(fiu == lanyok[...])       //(Q5)
       {
            cout << ...... ;       //(Q6)
            break;
      }
      else if(fiu < lanyok[....])  //(Q7)
       {
            jobb = .......;        //Q8)
      }
      else {
            bal = ..... ;          //(Q9)
       }
}
if(.......) //(Q9)
{
         cout <<„NINCS";
}
return 0;
}
```

Összefoglalva a kutatás célja a következő kérdés köré fonódik: van-e különbség az előzetes programozási ismerettel nem rendelkező 3., 5., 7. és 9. osztályos tanulók tanulási teljesítményei közt, pontosabban fogalmazva:

- Mennyire látszik az algoritmikus gondolkodás fejlődése különböző korosztályoknál ($RQ_1$)?
- Függ-e a fejlődés üteme a jelenlegi oktatási terv jellegétől (Művészeti iskola vs. Elméleti iskola) ($RQ_2$)?
- Függ-e a tanulók algoritmikus gondolkodásának fejlődése a tanulók nemétől ($RQ_3$)?
- Mennyire tudják az előzetes programozási tapasztalatokkal *nem* rendelkező egyetemisták elsajátítani az általunk bemutatott számítógépes algoritmust (lineáris és bináris keresés) ($RQ_4$)?
- Van-e különbség a videóval, valamint az animációval való oktatási módszerek közt az előzetes programozási tapasztalatokkal nem rendelkező egyetemisták eredményei között ($RQ_5$)?

## 4. Eredmények és következtetések

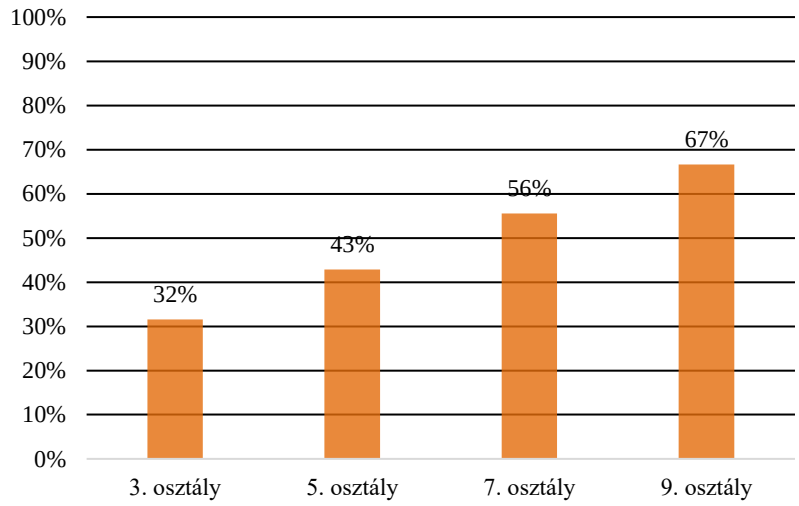
Az alsó-, felső tagozat, valamint a középiskolás tanulók száma 214 volt. Ezt a létszámot a 3. osztályosok (57 tanuló), 5. osztályosok (49 tanuló), 7. osztályosok (54 tanuló) és 9. osztályosok (54 tanuló) alkották.

Minden évfolyamon két különböző szakterületet vizsgáltunk: művészeti és elméleti (3. osztály: 25A+32T, 5. osztály: 20A+29T, 7. osztály: 25A+29T, 9. osztály: 26A+28T).

## 4.1. Első szakasz eredményei

### 4.1.1. A tanulók válaszainak feldolgozása korosztályonként (a két iskola eredményei összesítve)

Az első kérdésre adott válaszokat, mindkét iskolából, korosztályonként, a következő ábra szemlélteti (4. ábra: **Az első kérdés eredményei korosztály szerint (minden iskola)**):




**4. ábra:** Az első kérdés eredményei korosztály szerint (minden iskola)

Az online tesztben feltett kérdésekre adott válaszokat összegyűjtve, majd kódolva 0 és 1 – es értékekkel (1 – helyes, 0 – helytelen válasz), szerettük volna megtudni, hogy van-e szignifikáns eltérés a különböző korosztályú, különböző nemű, különböző tanterv szerint tanuló tanulók válaszai közt. Az adatok bináris jellege, valamint a minta viszonylag kis mérete miatt, a szakirodalommal összhangban, az adatok elemzését Fisher-egzakt teszttel végeztük. Az adatainkat 2×2-es kontingenciatáblazatokba rendeztük, és 95%-os szignifikancia szinttel dolgoztunk. Az *első kérdésre* (Q1) adott válaszok alapján az alábbi 2x2 – es kontingenciatáblákat nyertük.

| | **Helyes válaszok száma** | **Helytelen válaszok száma** |
|---|---|---|
| **3. osztály** | 18 | 39 |
| 7. osztály | 30 | 24 |
| **5. osztály** | 21 | 28 |
| **9. osztály** | 36 | 18 |

**2. táblázat:** Az első kérdésre adott helyes és helytelen válaszok száma osztályonként

A Fisher-egzakt teszt eredményei alapján nem találtunk szignifikáns különbséget a korosztályok kétévenkénti elemzésénél (3. osztály VS 5. osztály; 5. osztály VS 7. osztály; 7. osztály VS 9. osztály). Ezt követően négyévenkénti összehasonlítást végeztünk (3. osztály VS 7. osztály; 5. osztály VS 9. osztály). Ennek eredményei alapján, a 7. osztályos tanulók szignifikánsan jobban teljesítettek, mint a 3. osztályos tanulók ($p = 0.01 < 0.05$). Hasonló eredményeket értünk el az 5. és 9. osztályos tanulók teljesítményét tekintve, ahol a szignifikancia értéke $p = 0.01 < 0.05$ volt.

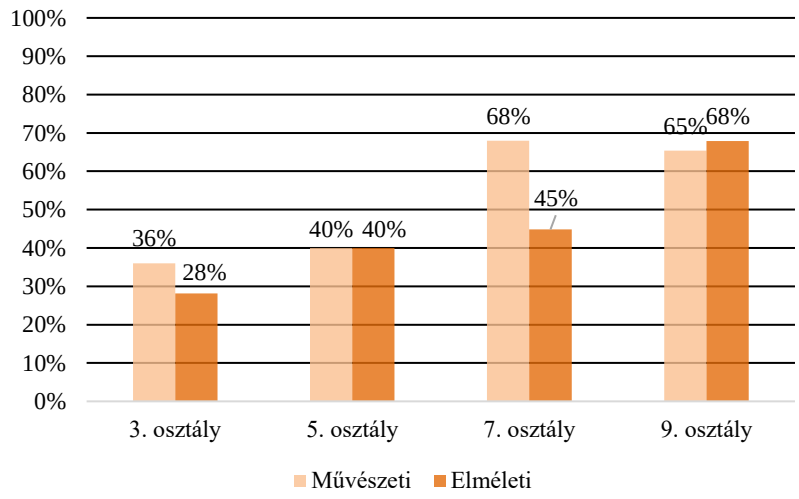
Következtetés ($RQ_1$): a diákok algoritmikus gondolkodása fejlődik, de moderáltan. A jelenlegi iskolarendszer nem járul kellőképpen hozzá ennek fejlesztéséhez.

### 4.1.2. A tanulók válaszainak feldolgozása iskolánként

A művészeti iskolában elemi szinten csak zene osztályok vannak. Rajz osztályok csak 5.-től indulnak. Ezzel összhangban a kiválasztott 3A osztály, zene osztály volt. A kiválasztott 5A, 7A és 9A osztályok viszont már rajz osztályok voltak. Mivel az 5A osztályosok csak 5.-től tanultak a művészeti iskolában, ezért az elemi osztályokat ugyancsak különböző elméleti iskolákban járták ki. Ily módon az 5A és 5T osztályok összehasonlítása nem volt releváns a két iskola összehasonlítása szempontjából. Érdekes, hogy e két osztály azonos eredményeket ért el.

Hasonló mondható el a két 9. osztály összehasonlításáról is. A 8. osztályt követő felvételi vizsga miatt kijelenthető, hogy: a művészetiben tapasztalható visszaesés 9.-ben azzal magyarázható, hogy a reál-beállítottságú tanulók közül sokan iskolát váltottak, főleg azok, akiket jobban vonzott az informatika. Az elméleti iskola 9. osztályában tapasztalható növekedés annak számlájára írható, hogy kimagaslóan teljesítő diákok kerülnek felvételre ide. Ily módon a 9. osztályos eredmények sem relevánsak a két iskola összehasonlítása szempontjából. A tanulók fej-fej mellett teljesítettek (65% és 68%) mindkét iskola esetében.

Fontos tehát szem előtt tartani azt, hogy "igazán" művészeti-s tanulóknak, csak a 3. és 7. osztályos tanulókat tekinthetjük, mivel csak ők azok, akik 2 évet már ebben az iskolában tanultak. Éppen ezért, ez iskolák szerinti eredmények kiértékelésében elsősorban erre a két osztályra összpontosítottunk.




**1. ábra:** Az első kérdés eredményeinek feldolgozása iskolák szerint

A művészeti iskolában a „3 VS 7" eltoláshoz tartozó növekedés szignifikáns (Fisher teszt alapján: $p=0.04<0.05$), az elméleti iskolában viszont nem. Látható, hogy a művészetiben tapasztalható növekedéshez főleg az 5.-ről 7.-re való eltolás járult hozzá (40%ról 68%ra). Ez a művészetoktatás előnyére írható, mert a 7.-es diákok két évet ebben a szellemben tanultak (5. és 6. osztályban).

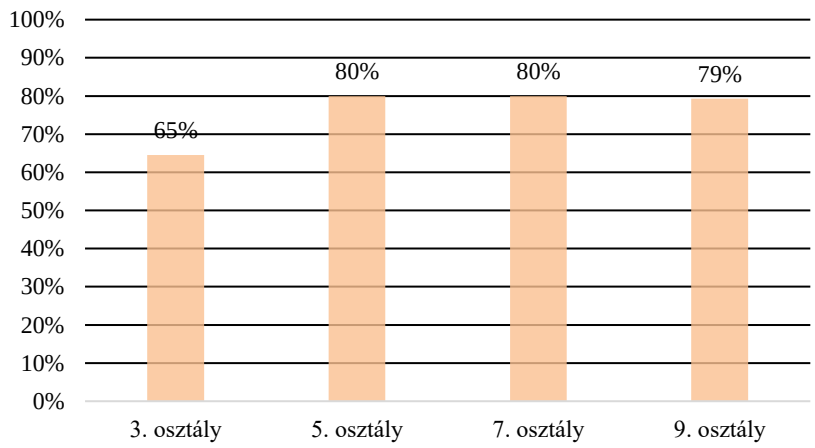
Ezzel összhangban a művészeti iskola 7. osztályos tanulói jelentősen jobban teljesítettek, mint az elméleti iskola 7.-es tanulói (36% VS. 28%), bár nem szignifikáns a különbség.

A 3. osztályos tanulók vizsgálata során is hasonló eredményeket értünk el. A művészeti iskola 3. osztályos tanulói bár jobban teljesítettek, mint az elméleti iskola 3. osztályos tanulói (36% vs. 28%), a különbség nem szignifikáns.

Következtetés ($RQ_2$): A jelenlegi oktatás mérsékelten járul hozzá a számítógépes gondolkodás fejlesztéséhez. Az eredmények azonban arra engedtek következtetni, hogy a művészetoktatásnak jelentősebb a hozzájárulása a számítógépes gondolkodás fejlesztéséhez, mint más elméleti iskolának.

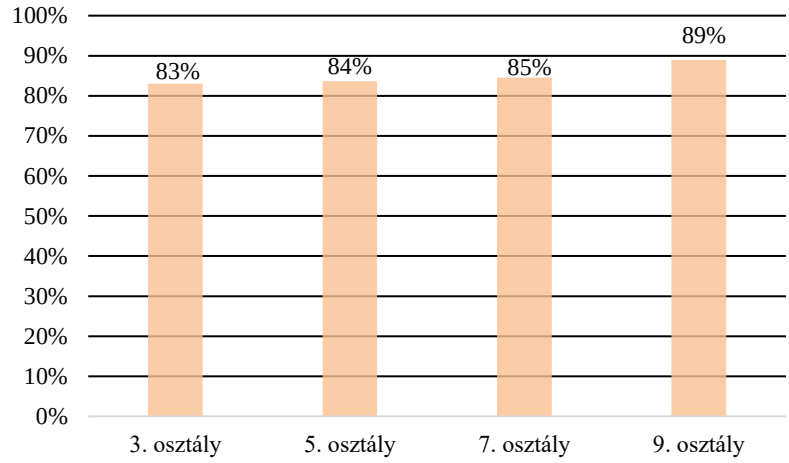
## 4.2. Második szakasz eredményei

A mérési folyamat második szakaszában a lineáris algoritmust szemléltettük videóval. Továbbra is az algoritmusban való elmélyülést, annak megértését vizsgáltuk, és azt, hogy milyen mértékben járulnak hozzá ehhez az említett szemléltető eszközök. A tanmenet öt kérdésből tevődött össze, melynek válaszait 1-5 közötti értékekkel osztályoztunk (a magyar oktatási rendszer osztályozásához hasonlóan). Az eredményt, vagyis az osztályok szerinti átlagokat, az alábbi ábra szemlélteti:




**6. ábra:** A második kérdés eredményei korosztály szerint (minden iskola)

Az egyszempontos varianciaanalízis (ANOVA) eredménye azt mutatja, hogy van szignifikáns különbség a korosztályok közt ($F(210, 3) = 7.79$, $p = 0.00$), azonban amint azt az átlagok is tükrözik, ez a különbség a 3. osztályos tanulók eredményeiből adódik (65%), hiszen az 5, 7, 9 osztályosok váll-váll mellett teljesítettek (80%, 80%, 79%). Ez további tesztek alapján is igazolódott. A (3, -1, -1, -1) kontraszt értékeket alkalmazva világossá vált, hogy a harmadik osztály tér el szignifikánsan a többiektől ($t_{210} = 4.83$, $p = 0.00$). A 3. osztály tanulói, mindkét iskolából, a $Q_{2.2.3}$ és $Q_{2.2.4}$ kérdésekre adott helytelen válaszaik alapján maradtak le a többi osztályokhoz képest, mivel a kérdésben szereplő „x" lány (valamennyi) fogalom még számukra ismeretlen információ volt. Annak érdekében, hogy megbizonyosodjunk arról, hogy valóban ez okozta a szignifikáns különbséget, újra elemeztük minden korosztály válaszait, azonban csak az első három kérdés alapján ($Q_{2.3.1}$ és $Q_{2.3.2}$ kivételével), és így a szignifikáns különbség eltűnt ($p = 0.47$).

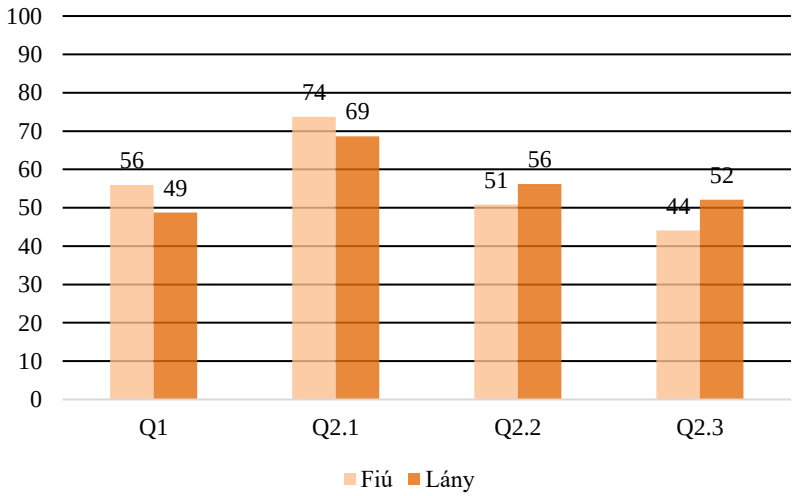



**2. ábra:** A második kérdés eredményei az utolsó két kérdés kivételével korosztály szerint (minden iskola)

Következtetés ($RQ_1$): Elmondhatjuk, hogy az algoritmus egyformán megtanítható tanulóknak az általunk használt oktatási módszerrel és eszközökkel, hiszen megvan az a potenciál a diákokban, már elemi osztálytól, hogy számítógépes gondolkodás orientált fogalmakat elsajátíthassanak.

## 4.3. A válaszok feldolgozása nemenként

Az összes választ elemezve, nemenként nem jutottunk szignifikáns különbségekhez. Ez arra engedett következtetni, hogy a módszer nem csak korosztálytól, hanem nemtől függetlenül is alkalmazható a diákok körében ($RQ_3$).




**8. ábra:** A kérdésekre adott válaszok feldolgozása nemenként

Érdekes volt az a tény, hogy annak ellenére, hogy a felmérés elején a „Szereted-e az informatikát?” kérdésre többnyire a fiúk válaszoltak pozitívan, bebizonyosodott, hogy az algoritmusra vonatkozó összes kérdés esetében csupán moderált különbség volt észlelhető a fiúk és lányok teljesítménye között ($p_{Q1} = 0.5173$, $p_{Q2.1} = 0.3954$).

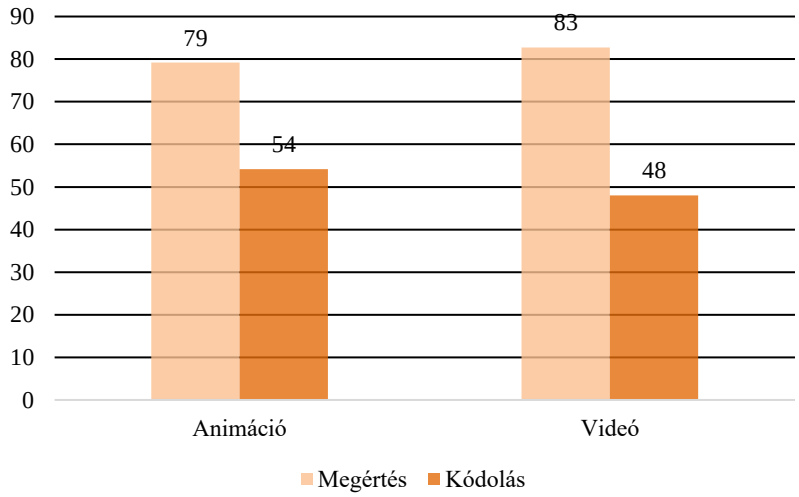
## 4.4. Videó vs. Animáció szemléltetési módszer

### 4.4.1. Lineáris keresés

A felmérésben 42 tanuló vett részt, melyet két csoportra osztottunk. A két csoport két különböző oktatási módszerrel tanult: videóval, illetve animációval.

Az egyszempontos varianciaanalízis (ANOVA) eredménye, melyet az elsőéves előzetes programozási ismeretekkel nem rendelkező egyetemisták esetében alkalmaztunk nem vezetett szignifikáns különbséghez a két csoport eredményei között ($F(40, 1) = 0.13$, $p = 0.71$).

Következtetés ($RQ_4 – RQ_5$): Összehasonlítva az eredményeket, amely az algoritmusban való elmélyülést, megértést, valamint a kódolást foglalta magába, bebizonyosodott, hogy bár többnyire sikeresen teljesítették a kódolással kapcsolatos feladatokat, egyik oktatási módszer esetén sem tűnt el teljes mértékben a megértés és kódolás közti szakadék (9. ábra: **Lineáris keresés - Animáció vs. Videó**).



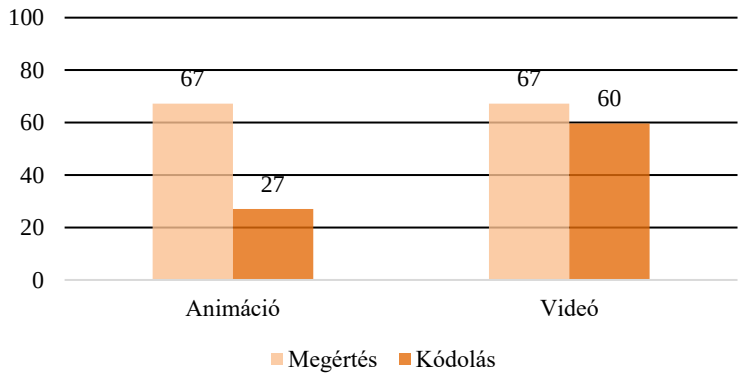

**9. ábra:** Lineáris keresés - Animáció vs. Videó

### 4.4.2. Bináris keresés

A felmérésben 39 tanuló vett részt, melyet két csoportra osztottunk. A két csoport a lineáris kereséshez hasonlóan, különböző oktatási módszerrel tanult: videóval, illetve animációval.

Az egyszempontos varianciaanalízis (ANOVA) eredménye, melyet az elsőéves előzetes programozási ismeretekkel nem rendelkező egyetemisták esetében alkalmaztunk szignifikáns különbséghez vezetett ($F(38, 1) = 25.75$, $p = 0.00$).

Következtetés ($RQ_4 – RQ_5$): Az előző algoritmussal ellentétben, a bináris keresés tanítása során az a csoport, mely a videó segítségével tanulhatott, sikeresebben teljesített a kódolás fázisát illetően. Ennek következtében a megértés és a kódolás közti szakadék jelentősen csökkent, míg az animáció oktatási módszert alkalmazva, továbbra is megmaradt (10. ábra: **Bináris keresés - Animáció vs. Videó**).




**10. ábra:** Bináris keresés - Animáció vs. Videó

## 5. Összefoglaló

A jelenlegi oktatás mérsékelten járul hozzá a számítógépes gondolkodás fejlesztéséhez. Az eredmények azonban arra engedtek következtetni, hogy a művészetoktatásnak jelentősebb a hozzájárulása a számítógépes gondolkodás fejlesztéséhez, mint más elméleti iskolának.

A több szakaszból álló mérések eredményei arra engedtek következtetni, hogy mindenkinek, korosztálytól, nemtől, iskolától függetlenül, megtaníthatunk olyan algoritmusokat, amelyek elősegítik a számítógépes gondolkodásuk fejlődését, ha megfelelő oktatási módszert választunk. Bebizonyosodott, hogy úgy a videó, mint az animáció is hozhat pozitív eredményeket a tanulásban való előre haladásban, és minden korosztálynak megvan a saját bevált tanulási módszere. Az eredmények azt is kimutatták, hogy bár jelentősen javult a diákok kódolással kapcsolatos feladatmegoldási készsége, még mindig észlelhető egy szakadék a megértés és az algoritmus valódi leprogramozása között. Úgy döntöttünk, hogy ennek javítása érdekében egy kódolást elősegítő tanulási lépést is be kell építenünk az AlgoRythmics környezetbe. Folyamatosan igény van érdekes, kreativitást elősegítő, motiváló és különleges oktatási módszerekre annak érdekében, hogy a tanulási folyamat hatékony, figyelemfelkeltő és interaktív legyen.

## Irodalom

1. Ahadi, A., Lister, R., Lal, S., Leinonen, J., & Hellas, A. (2017, January). Performance and Consistency in Learning to Program. In Proceedings of the Nineteenth Australasian Computing Education Conference (pp. 11-16). ACM.
2. Brown, N. C., Sentance, S., Crick, T., & Humphreys, S. (2014). Restart: The resurgence of computer science in UK schools. ACM Transaszámítógépes gondolkodásáhozions on Computing Education (TOCE), 14(2), 9.
3. CSTA. (2017). CSTA K-12 Computer Science Standards, Revised 2017. Retrieved from: https://www.csteachers.org/page/standards.
4. Donald, M. (1991/2001): Az emberi gondolkodás eredete. Osiris Kiadó, Budapest.
5. Donald, M. (2001): A mind so rare. The evolution of human consciousness. W. W. Norton & Company, New York.
6. European Schoolnet. (2015). Computing our future. Computer programming and coding: priorities, school and initiatives across Europe [Technical report]. Retrieved from: http://www.eun.org/resources/detail?publicationID=661.

7. Gander, W., Petit, A., Berry, G., Demo, B., Vahrenhold, J., McGettrick, A., ... & Meyer, B. (2013). Informatics education: Europe cannot afford to miss the boat. Report of the joint Informatics Europe & ACM Europe Working Group on Informatics Education.
8. Katai, Z. (2014, June). Seleszámítógépes gondolkodásához ive hiding for improved algorithmic visualization. In Proceedings of the 2014 conference on Innovation & technology in computer science education (pp. 33-38). ACM.
9. Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014, June). Computational thinking in K-9 education. In Proceedings of the working group reports of the 2014 on innovation & technology in computer science education conference (pp. 1-29). ACM.
10. Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018).
11. Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018). Can computational talent be detected? Predictive validity of the Computational Thinking Test. International Journal of Child-Computer Interaction, 18, 47-58.
12. Settle, A., Goldberg, D. S., & Barr, V. (2013, July). Beyond computer science: computational thinking across disciplines. In Proceedings of the 18th ACM conference on Innovation and technology in computer science education (pp. 311-312). ACM.

# AlgoRythmics: Tánctól a Kódig

Osztián Pálma Rozália[1], Kátai Zoltán[2]

{[1]osztian.palma, [2]katai_zoltan}@ms.sapientia.ro
SAPIENTIA EMTE

**Absztrakt.** Köztudott, hogy napjainkban jelentősen elterjedt az online oktatási környezetek használata, és bár folyamatosan bővül az E-learning környezetek tárháza a kérdés mindig megfogalmazódik bennünk: Vajon hogyan lehetne a lehető leghatékonyabban tanítani informatikát és fejleszteni az algoritmikus-gondolkodást?

Az elmúlt időszakban kifejlesztettünk egy olyan online oktatási környezetet, mely hasonló a közismert E-learning felületekhez, mégis különleges a maga módján. *AlgoRythmics*: az *algoritmus* (informatika) és a *ritmika* (ritmus, tánc), vagyis a *tudomány* és *művészet* ötvözete. Egyedisége abban rejlik, hogy különböző algoritmusok és azokhoz tartozó tanfolyamok segítségével segít a felhasználóknak az algoritmikus gondolkodás fejlesztésében úgy, hogy elvezeti őket a *tánctól, egészen a kódig*. Mindezt öt alapvető tanulási lépés teszi lehetővé: *a videó, animáció, levezénylés, kódépítés és az életre kelt kód* fázisai.

Az egyedi tanulási lépéseknek köszönhetően a teljes tanulási folyamatot változó *interaktivitási szint* jellemzi, így a tanfolyamok során a felhasználóknak lehetőségük van független megfigyelőként (*0 interaktivitás*), részleges felhasználói irányítással (*1/2 interaktivitás*), illetve teljes felhasználói irányítással (*1 interaktivitás*) végig vezetni az algoritmus lépéseit.

**Kulcsszavak:** E-learning, interaktivitás, algoritmikus-gondolkodás, tanulási lépések, tudomány és művészet

## 1. Bevezetés

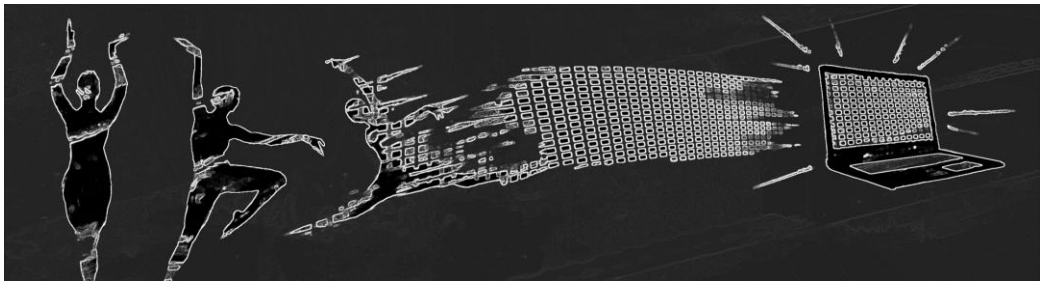
Mai digitális világunkban egyre többen, akár már gyerekkorban, találkoznak a számítógépek és az informatika nyújtotta lehetőségekkel. Nagyon sok gyerek már egészen kiskortól telefont fog a kezében, vagy számítógép elé ül, és játszik, tanul, fejlődik mely cselekvésekhez elengedhetetlen, hogy olykor használja, talán tudattalanul is, számítógépes gondolkodását. Amint azt Jeannette Wing is megfogalmazta *a számítógépes gondolkodás* a *negyedik* alapvető készség, az *aritmetika*, *olvasás* és az *írás* mellett, mellyel egy XXI. századi embernek rendelkeznie kell, és folyamatosan fejlesztenie kell azt. Ennek a készségnek a fejlesztésére az egyik legnagyobb segítséget az informatikai algoritmusok nyújtják, melyek nap, mint nap, jelentős részét képezik életünknek. Elsősorban ezek megértése feltételezi a tudásban való elmélyülést. Emellett azonban sok szakértő úgy tekinti, hogy a számítógépes gondolkodás hatékony fejlesztése magába foglalja a kódolást is.

Napjainkban egyre több iskolában vezették be a kisiskolások órarendjébe is az informatika órákat. Erdélyben már a 2017/2018 tanévtől kezdődően jelen van az informatika óra a középiskolások tantervében. Belátható, hogy az informatikai algoritmusok megértése és elsajátítása fontos szereppel bír. Ebben a szellemben vette kezdetét 2007-ben az *AlgoRythmics* projekt, mely kapcsán 6 videó és egy webalkalmazás látott napvilágot. A videók 6 *rendezési algoritmus* (buborékos-, beszúró-, kiválasztó-, összefésülő-, gyors- és shell rendezés) néptánccal eltáncolt változatát mutatták be. Amint azt a neve is mutatja ez az oktatási stratégia a **tudományt** *(algoritmus)* és a **művészetet** *(tánc, vagyis ritmus)* ötvözte, így kapta az *Algo – Ritmika*, vagy angolul *AlgoRythmics* nevet.

Az előzetes kutatásoknak és munkának köszönhetően az algoritmusokat bemutató tánc-videók rendelkezésünkre álltak, és amint az kiderült hasznosak is bizonyultak. Szerettük volna azonban ezeket egybegyűjteni egy olyan oktatási környezetben, mely más módszerekkel kiegészülve, kéz a kézben, segítséget nyújt a felhasználóknak az algoritmikus gondolkodás elsajátításában. Egy olyan E-

learning környezet implementálását tűztük ki célul, mely hasonló, de mégis a maga nemében más tanulási lépések révén vezeti el a felhasználókat a tánctól, egészen a kódig úgy, hogy közben az interaktivitás különböző szintjein nyilvánul meg.

Bár algoritmus-vizualizációval számos helyen találkozhattunk már, ez a környezet a táncra és az animációkra összpontosul. Ezek közös bemutatásával akár két eltérő tanulási stílust is megcélozhatunk, hiszen lehetőséget biztosítunk a felhasználóknak arra, hogy *dinamikus (tánc)*, illetve *statikus (animáció)* vizualizációval tanulhassanak. A dinamikus bemutatás magával hozza az emberi mozgás effektust, mely segítségével a diákok azonosulni tudnak az adott számot viselő táncossal, a statikus vizualizáció pedig egy absztraktabb bemutatást tesz láthatóvá, ahol az animáció különböző, letisztult és egyszerű mozzanatai érvényesülnek.



**1. ábra:** Tánctól a Kódig

# 2. Interaktivitási szintek

## 2.1. 0 Interaktivitás:

Ez a törzsfogalom foglalta magába a független megfigyelést, mely során a felhasználók, vagyis a diákok zavartalanul tekinthették meg egy-egy algoritmus bemutatását (videó vagy animáció által). Jelen oktatási rendszerünkben ez az előadó órákhoz hasonlítható, melyek esetében a tanár előadást tart egy adott témáról, a tanulók pedig figyelemmel kísérik azt.

## 2.2. ½ Interaktivitás:

Az úgynevezett „fél" interaktivitás, a felhasználók részleges bevonását jelentette. Jelen napjainkban mindez párhuzamba hozható az interaktív tanórákkal, mely során a tanárok kérdéseket intéznek a diákokhoz, ezáltal próbálva bevonni őket az tanítási- és tanulási folyamatba.

## 2.3. 1 Interaktivitás:

A harmadik és egyben utolsó csoportosítási osztály a teljes felhasználói irányítást foglalta magába. Ez tulajdonképpen a teljes interaktivitást jelentette, mely során a tanulóknak önállóan kellett *„levezényelni"*, vagyis irányítani az adott algoritmust. Mindennapjainkban ez önálló munkák, feladatok formájában jelenik meg, mely során a diákoknak a kitűzött feladat kezdetétől el kell jutniuk a végső termékig, vagyis az elkészített feladványig.

# 3. Szakirodalmi áttekintő

## 3.1. 0 Interaktivitás

Számos tanulmány bizonyította, hogy az algoritmusok animációval történő reprezentációja igen hatékony és elősegíti a hallgatókat abban, hogy megértsék az algoritmusok működését. A kutatók azt is megfigyelték, hogy egyszerű animációk által nem érhetünk el tartós tudást, hiszen a legtöbb esetben a hallgatók néhány hónap után már el is felejtették az algoritmust. Annak érdekében, hogy az

algoritmusokkal kapcsolatos tartós ismeretek megszerzését biztosítsunk, a nézőket be kell vonni jelentőségteljes interakciókkal is. Bebizonyosodott, hogy azon hallgatók emlékezetében, akiknek van lehetőségük megtapasztalni, *„átélni"* egy adott témával kapcsolatos tudnivalókat, sokkal hosszabb ideig megmarad az információ, mint azokéban, akikkel csak az elméleti aspektusokat szemléltették, jelentőségteljes interakciók nélkül.

### 3.2. ½ Interaktivitás

Egy módszer, amely segítségével be lehet vonni a diákokat a tanulási folyamatba, az úgynevezett „interaktív jóslás", mely azt jelenti, hogy az animációs folyamat megszakad, és a felhasználónak kell meghatároznia a következő mozzanatot [1]. Ezzel kapcsolatosan számos kutatást vizsgáltunk meg: Először Byrne, Catrambone és Stasko [2] dolgozott ki egy módszert, melynek segítségével bevonták a hallgatókat a megjelenítési folyamatba. Az előre meghatározott pillanatokban a hallgatóknak szóban kellett elmondaniuk, hogy mi fog történni. A kutatás eredményei azt mutatják, hogy a hallgatóknak sokáig megmaradt emlékezetükben mindaz, amit megtanultak. Korhonen és Malmi kutatása [3] egy olyan módszert mutat be, mely során a tanulóknak lehetőségük nyílt az algoritmus manuális levezénylésére. Adott kulcsmomentumokban válaszolniuk kellett kérdésekre, melyeket rögzítettek és azonnali visszajelzést kaptak rá diákok. A kutatás nagyon jó eredményekhez vezetett. Egy újabb ötletes módszernek számít Naps, Eagan és Norton [4] kutatása, akik az algoritmus vizualizáció során „stop-and-think" kérdéseket használtak. Ahhoz, hogy a tanulók folytatni tudják a tanulási folyamatot, válaszolniuk kellett a feltett kérdésekre. Ezesetben is automatikusan kiértékelődtek a válaszok és visszajelzést kaptak a diákok, a tanulási folyamat pedig igen hatékonynak bizonyult.

Ezek mellett, Jarc, Feldman és Heller tanulmányában bemutatott megjelenítési rendszer [5] szintén megpróbálta aktívan bevonni a hallgatókat jelentőségtejes interakciókkal. Rendszerük lehetővé tette a hallgatók számára, hogy passzív módon tekintsék meg a vizualizációt *(„Mutasd meg")*, és úgy is, hogy legyenek bevonva interaktív kérdésekkel *(„Megpróbálom")*. Meglepő módon kutatási eredményeik azt mutatják, hogy a nagyobb interaktivitású csoport következetesen, de nem szignifikánsan alulteljesítette a kontroll csoportot.

### 3.3. 1 Interaktivitás

Továbbá, egy újabb tanulmány is megerősíti az interaktív algoritmusok interaktív megjelenítésének hatékonyságát. Ennek kapcsán nyer először értelmet az *„orchestration"* algoritmus fogalma, mint magasabb szintű interaktivitás: *„Az interaktív predikció egyik különleges esete az, amikor a hallgatóknak a vizsgált algoritmust kell irányítaniuk"*. Ez tulajdonképpen azt jelenti, hogy az interaktív vizuális tanulási környezetet használó hallgatóknak, az algoritmusnak nem csak a következő lépését kell meghatározniuk, hanem végre is kell hajtaniuk azt. A feladat megértésének és teljesítésének biztosítása érdekében a környezetnek azonnali visszajelzést kell adnia minden beérkezett válaszról, és lehetőséget kell, hogy biztosítson arra, hogy a tanulók újra próbálkozhassanak, és szükség esetén segítséget kérhessenek. A kutatás egyik fő gondolata kihangsúlyozta, hogy a *„levezénylési"* folyamat elsődleges célja nem a felmérés, hanem a hallgatók megértésének javítása és finomítása a vizsgált algoritmusról [1].

Mivel megoszlanak a vélemények az oktatásban használt interaktivitás mértékéről, a fent említett tanulmányok alapján azt mondhatjuk, hogy minden egyes interaktivitási szintnek megvannak a saját erősségei és gyengeségei, és nincs univerzálisan optimális interaktivitási szint. Éppen ezért döntöttük úgy, hogy ennek hasznosságát és szerepét szeretnénk az általunk implementált AlgoRythmics oktatási környezetben is tesztel, hiszen annak egyik jellegzetessége a változó interaktivitási szint.

## 4. AlgoRythmics online oktatási környezet

Az AlgoRythmics oktatási környezet hasonló más online, tanítást és tanulást elősegítő oldalakhoz, mégis különlegesnek nevezhető a maga módján. Elsősorban a rendezési-, keresési- és visszalépéses

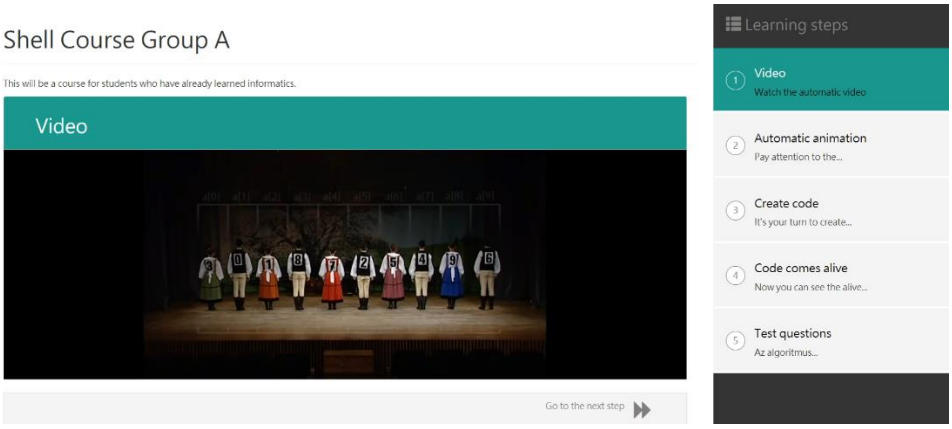
keresési algoritmusokra specializálódott. Célunk az volt, hogy egy olyan interaktív weboldalt biztosítsunk, mely elősegíti a tanulók algoritmusokban való elmélyülését, azok megértését és végül, de nem utolsó sorban leprogramozását. Mindezt, az oktatási környezetet alkotó algoritmusok, tanfolyamok és a tanulási lépések segítségével valósítja meg.

## 4.1. Tanulási lépések

Az egyik legfontosabb összetevői az oldalnak a tanulási lépések. Donald Knuth *„Egy algoritmust látnom kell ahhoz, hogy elhiggyem."* gondolatára alapozva, kijelenthető, hogy az algoritmusok vizualizációja nagymértékben hozzájárul a megértéshez, de emellett a tanulók kíváncsiságának, motivációjának felkeltéséhez is. Az *5+1 tanulási lépés* tulajdonképpen ezt a célt, valamint a lecketervek változatosságát biztosítja, hiszen ezek mindegyike más és más jellegzetességeivel járul hozzá a tanulási folyamat fázisaihoz.

### 4.1.1. Videó

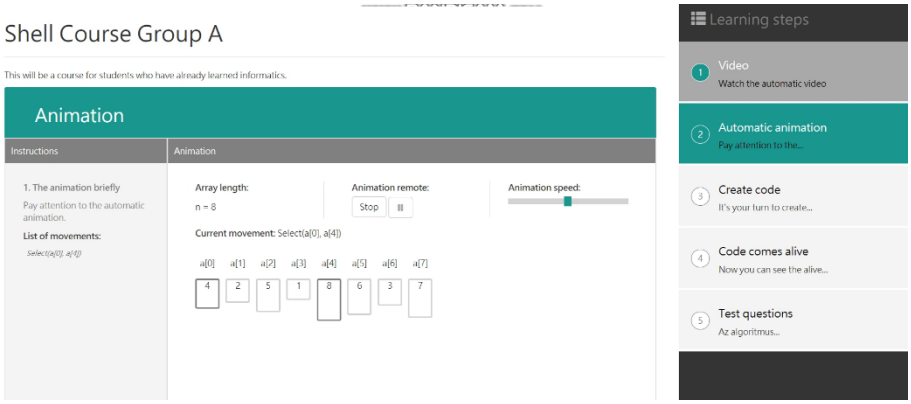
A videó tanulási lépés mutatja be az AlgoRythmics kutatócsoport által létrehozott eltáncolt algoritmusok videóit. A videók amellett, hogy egy-egy algoritmus vizualizációját jelentik, etnikai jelleggel is rendelkeznek. A különböző táncstílusok, melyek megtalálhatóak a táncvideókban, illeszkednek az adott algoritmusok típusaihoz. A rendező algoritmusok erdélyi néptánccal (pl: küküllőmenti-, szászcsávási-, mezőségi néptáncok), a kereső algoritmusok flamenco tánccal, míg a visszalépéses kereső algoritmusok (N királynő probléma) balett tánccal vannak bemutatva.




**2. ábra:** Videó tanulási lépés

### 4.1.2. Animáció

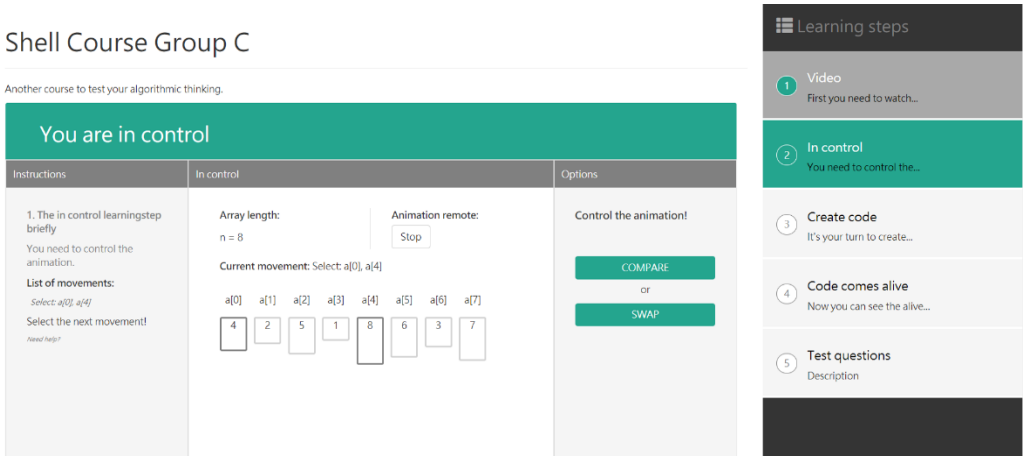
Az animációk képezik az algoritmusok bemutatásának egy absztraktabb változatát. A korábban emberek által képviselt értékek helyét, az animáció esetén számértékek veszik fel, melyek különböző magassággal rendelkező „dobozok" segítségével utalnak az adott értékek nagyságrendjére. Ez a tanulási lépés az egyik legfontosabb részét képezi az oktatási környezetnek, hiszen a videó tanulási lépésen kívül minden másik tanulási fázisban megjelenik, és ehhez illeszkednek, illetve társulnak az algoritmusvizualizáció további jellemzői.




**3. ábra:** Animáció tanulási lépés

### 4.1.3. Levezénylés

A levezénylés tanulási lépés az animáció egy komplexebb változata. Az előző tanulási lépéshez hasonlóan, itt is megjelenik egy „dobozok" által szemléltetett számsorozat, melyet rendezni, vagy melyben keresni kell. Ezúttal azonban teljes szinttű interaktivitás jellemzi a tanulási folyamatot. A felhasználónak elejétől a végéig kell irányítania az algoritmus főbb mozzanatait. Mindezt, az alapvető műveleteket segítő gombok használatával, vagyis az összehasonlítás (compare), csere (swap) gombokkal, illetve az elemek kiválasztásával teheti meg.

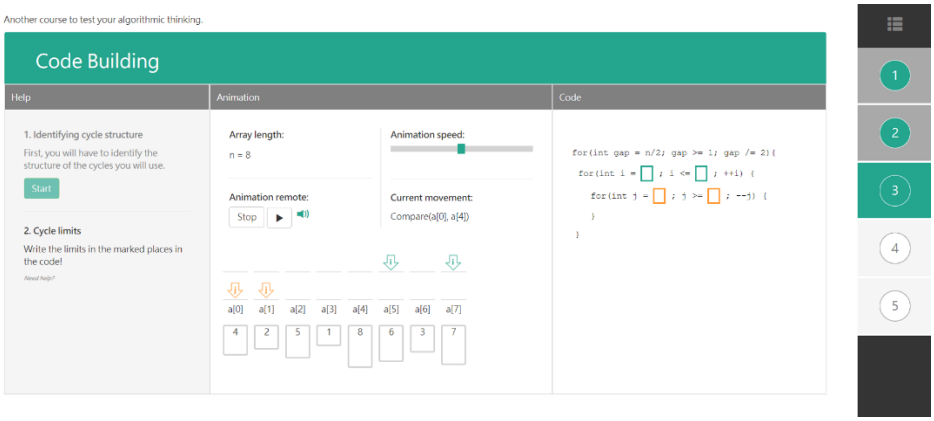



**4. ábra:** Levezénylés tanulási lépés

### 4.1.4. Kódépítés

Az első három tanulási lépés segíti elő az algoritmikus-gondolkodásban való elmélyülést, a szemléltetést és ez által az algoritmusok megértését. Ahhoz azonban, hogy a tanulók programozási készségeit is tudjuk fejleszteni a bemutatott algoritmusokat illetően, bevezettük a kódépítés fázisát is. Bár már a videó tanulási lépés során is megjelenik a *hang*, a *zene*, itt a hallással való érzékelés külön szerepet kap. Ez az adott algoritmus ciklusszerkezetének meghatározásakor lép érvénybe, amikor a felhasználónak hang alapján kell meghatározni, hogy egy, két egymásba ágyazó, vagy két külön ciklus

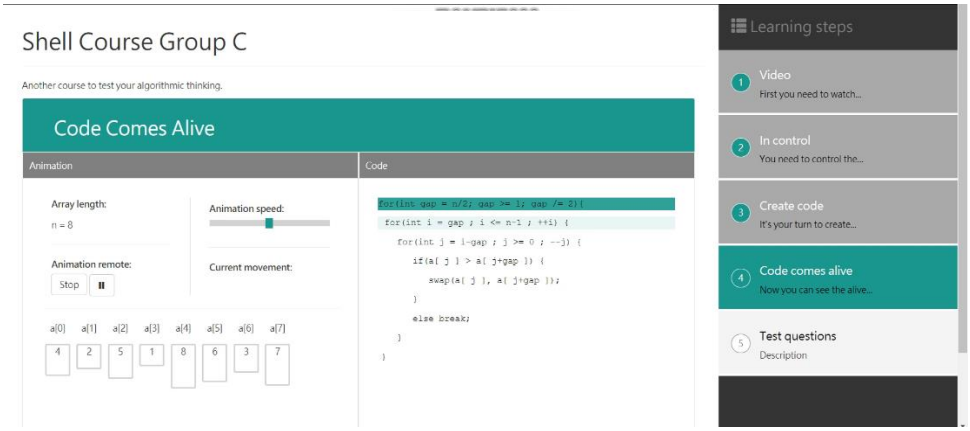
képezi az algoritmus vázát. Ezt követően az adott ciklusok határértékeit kell, hogy meghatározza a felhasználó, majd az összehasonlításra, illetve a cserére vonatkozó kritériumokat. Ha mindezt sikeresen elvégezte, elkészül az algoritmushoz tartozó kódrészlet.




**5. ábra:** Kódépítés tanulási lépés

### 4.1.5. Életre kelt kód

Ez a tanulási lépés tekinthető egyfajta összefoglalónak is. Az animáció, itt is fontos szereppel bír, hiszen a „megépített" kód egy-egy mozzanata egyszerre értékelődik ki az animáció különböző fázisaival. Tulajdonképpen olyan, mintha a kód irányítaná az animációt.

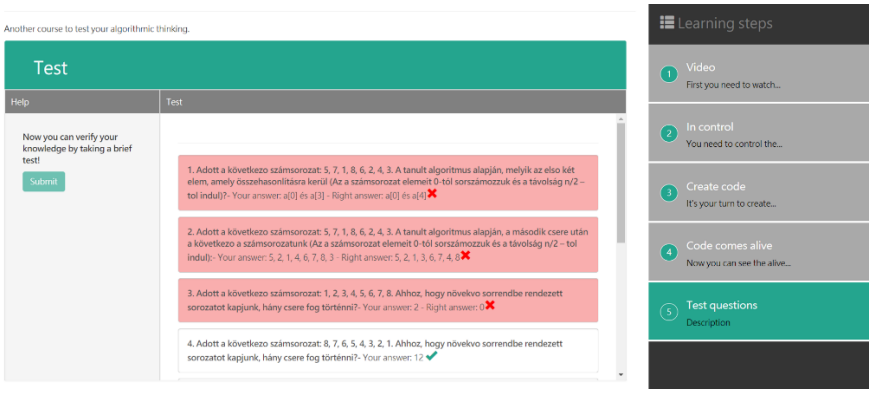



**6. ábra:** Életre kelt kód tanulási lépés

### 4.1.6. Kiértékelő tesztek

Az öt alapvető tanulási lépés segít a felhasználóknak abban, hogy elmélyüljenek az algoritmusok különböző vizualizációjában. Ahhoz azonban, hogy fel tudjuk mérni a tanulók teljesítményét is, azt hogy mennyire értették meg az adott algoritmus lényegét, és a különböző interaktivitási szintek mindegyike milyen mértékben bizonyult hatékonynak, bevezettük az ellenőrző tesztek fogalmát, vagyis egy extra (+1) tanulási lépést. Ezeket tetszés szerint határozhatják meg a tanárok különböző válaszlehetőség típusokat megadva (egy-, több- vagy véleménynyilvánító válaszok). A válasz-

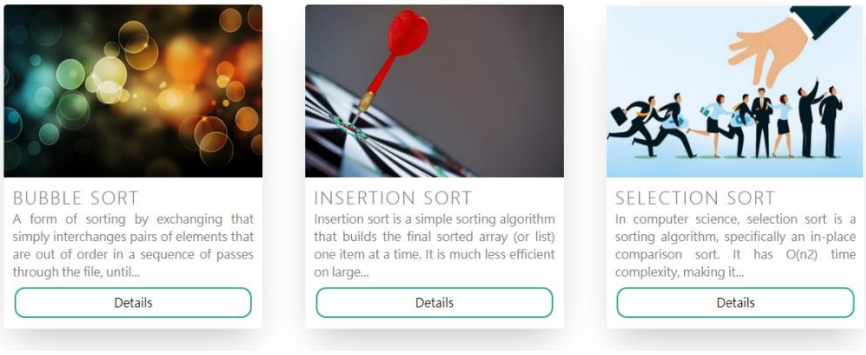
lehetőségek meghatározása mellett, a tanárok a kérdések számát is rögzíthetik. A felhasználók, amennyiben minden kérdést megválaszoltak, megtekinthetik helyes, illetve helytelen válaszaikat, hiszen a felmérő tesztek automatikusan értékelődnek ki.




**7. ábra:** Kiértékelő teszt

## 4.2. Tanfolyamok

Az oldalon megjelenő online tanfolyamok teszik lehetővé azt, hogy mérni tudjuk a felhasználók teljesítményét. A tanfolyamok tulajdonképpen a felsorolt tanulási lépések különböző változataiból és kombinációiból épülnek fel, melyeket a tanárok vagy adminisztrátorok állíthatnak be tetszés szerint. Ehhez kapcsolódik az interaktivitási szintek meghatározása is. A tanulási lépések és azok típusai megfelelnek egy-egy adott interaktivitási szintnek, így azok hozzáadása a tanfolyamokhoz egy meghatározott interaktivitással rendelkező lecketervet eredményez.

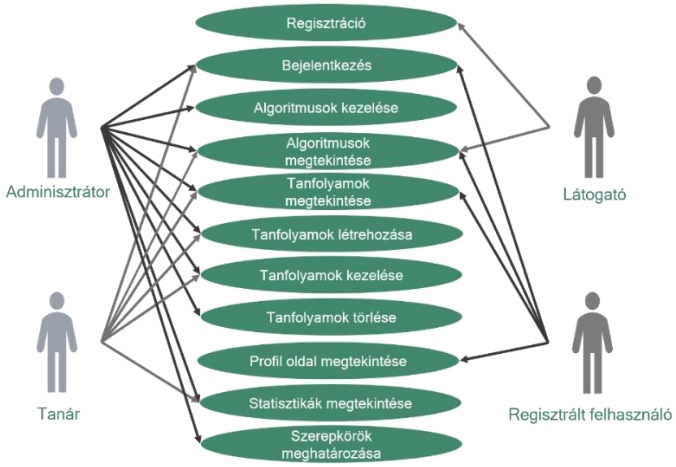



**8. ábra:** Tanfolyamok

## 4.3. Felhasználók

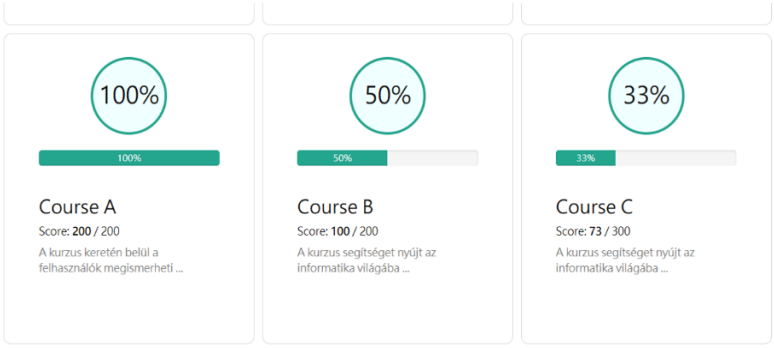
Jelen oktatási rendszerünk két legfontosabb szereplője: a diák és a tanár. Ezt a két fő kategóriát az általunk implementált E-learning környezetben kibővítettük, és így négy felhasználói szerepkört különítettünk el. Az *adminisztrátor* és a *tanár* foglalja magába a **„szerkesztő”** funkcionalitásokat, melyek a lecketervek és azok jellemzőit határozzák meg; a *látogató,* illetve a *regisztrált felhasználó* (az

adott tanuló) pedig a **„megfigyelő”** lehetőségeket, vagyis megtekinthették, és elvégezhették a rendelkezésükre álló tanfolyamokat.



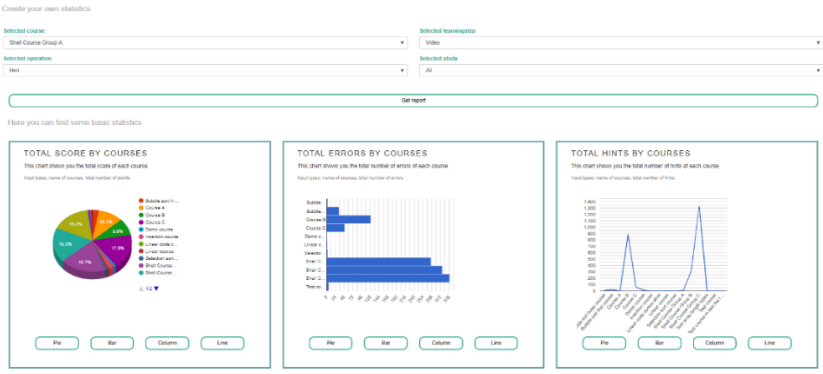

**9. ábra:** Use case diagram

Tanulási folyamatról lévén szó, úgy a tanárok, mint a tanulók is kíváncsiak elért eredményeikre. A diákok számára biztosítottunk egy *felhasználói* (profil) *oldalt*, melyen minden regisztrált felhasználó nyomon követheti az elkezdett tanfolyamokon való előrehaladást, és az elért pontszámot.




**10. ábra:** Profil oldal

A tanulási folyamat eredményességét a hibák, illetve segítségkérések összességével határoztuk meg. Minden felhasználói interakció során kérhettek segítséget a tanulók az adott lépést illetően, illetve hibázhattak. Ezek közül mindkettő pontlevonással járt. Annak érdekében, hogy a tanárok is megtekinthessék az adott lecketervek hatékonyságát, és azt, hogy az algoritmusok mely mozzanatai okozták a legnagyobb nehézséget a diákoknak (milyen gyakori volt a segítségkérések és hibák száma) lehetőséget biztosítottunk statisztikák megtekintésére, melyek az addigi felhasználói interakciók alapján értékelődnek ki.

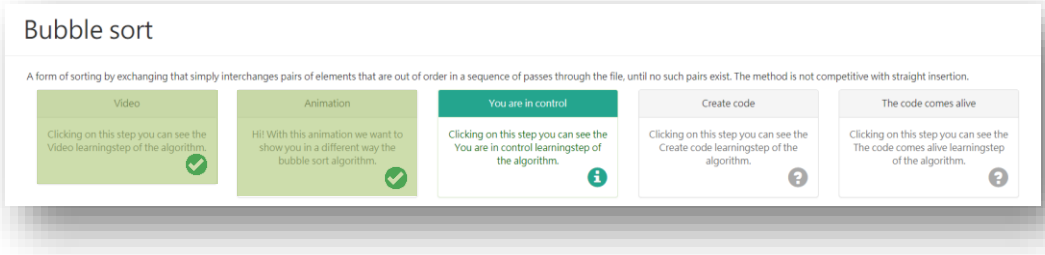



**11. ábra:** Statisztikák

## 4.4. Megjelenítési követelmények

### 4.4.1. Algoritmusok

Az algoritmusok esetében elsősorban az elméleti tudást szerettük volna megalapozni, így az öt tanulási lépés bemutatása előtt a felhasználók megtekinthetik az algoritmus nevét és a hozzá tartozó rövid leírást. Ez egyben, egy rövid ismerkedést is jelent az adott algoritmussal kapcsolatosan. A leírást a tanulási lépések öt típusa követi, melyek egy-egy kapcsolótáblával vannak szemléltetve. Bár a tanulási lépések elvégzésének sorrendje nincs meghatározva, a rendszer nyomon követi a felhasználók cselekvéseit. A még meg nem tekintett tanulási lépések kapcsolótábláit világosszürke szín és egy kérdőjel, az aktuális tanulási lépést zöld szín és egy információs ikon, míg a már megtekintett tanulási lépéseket sötétszürke szín és egy pipa jelölte.




**12. ábra:** Tanulási lépés kapcsolótábláinak megjelenítése

A tanulási lépések kapcsolótábláit követően az aktuális kiválasztott lépés jelent meg. Ez az animáció esetén két részre (feladatok és animálandó számsorozat), levezénylés esetén pedig három részre (feladatok, animálandó számsorozat és lehetséges műveletek gombjai) oszlott.

Az oldal alját az algoritmushoz tartozó speciális tanfolyamok listája zárta. Azok megtekintéséhez szükséges volt az oldalra látogató bejelentkezése.

### 4.4.2. Tanfolyamok

A tanfolyamok esetén az algoritmusokkal ellentétben nem egymás alatt követte a tanulási lépések listája és az aktuális lépés részletes bemutatása egymást, hanem egymás mellett. Ez azért volt fontos,

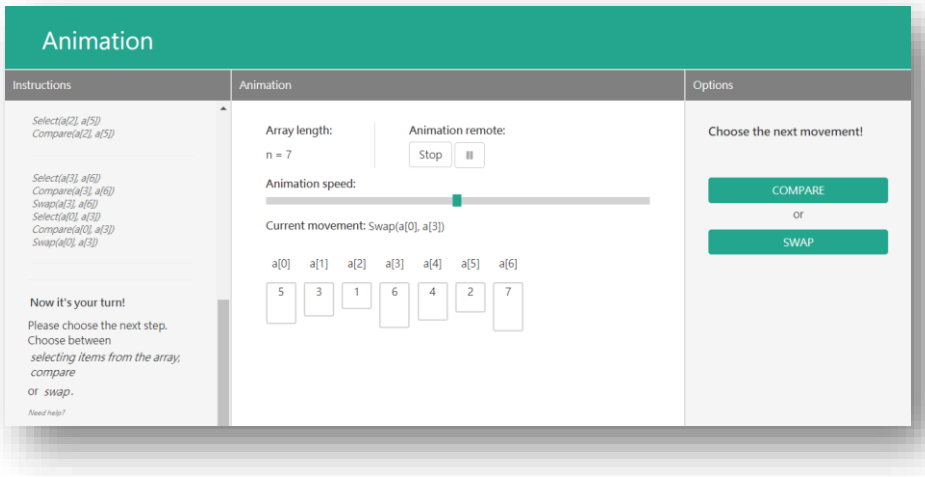
mert a felhasználóknak egyaránt szemléltetni szerettük volna az adott tanulási lépés fázisát, de azt is, hogy hol tart a tanfolyam elvégzésével kapcsolatban.

Mivel ennek következtében kevesebb hely jutott a soron következő tanulási lépés részletezésére, azt is megvalósítottuk, hogy a lépések listája „becsukható" (elrejthető) legyen. Így sikerült több helyet biztosítani az animációknak és lehetséges műveleteket jelző gomboknak.

### 4.4.3. Tanulási lépések

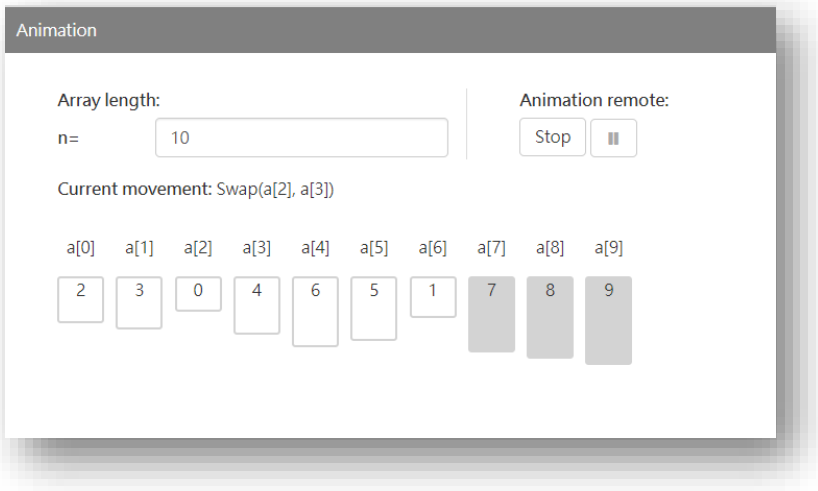
Szinte minden tanulási lépésre meg tudtunk határozni egy általános megjelenítési jellemzőt. A video kivételével minden lépés esetében legalább két részre osztottuk a képernyőt, melyek közül az első rész mutatta be a szükséges feladatok leírását, melyet a felhasználónak végig kellett vezetnie. Emellett, itt volt lehetősége a felhasználónak segítséget kérni a következő helyes műveletet illetően. Ez a rész biztosította azt is, hogy a tanuló nyomon követhesse az adott lépés fázisait (az eddig megtörtént mozzanatokat és műveleteket), és azt, hogy mikor bizonyul teljesítettnek az adott tanulási lépés.

A második rész legfontosabb részét az animálandó számsorozat képezte. Ez dobozok és azokban szereplő értékek segítségével volt szemléltetve. Az animáció fölött közvetlenül megjelenítettük a számsorozatot képviselő tömb jelét (a) és minden elem fölött az aktuális pozíciót (pl.: a[0], a[1], a[2]…). A második rész fontos jellemzőit képezte a számsorozat hosszának, illetve az aktuális művelet nevének megjelenítése. A felhasználóknak lehetőséget adtunk az animáció sebességének változtatására is, melyet egy úgynevezett sebességsáv jelezte. Kezdetben az animáció sebességét közepesre állítottuk. Emellett a felhasználó megállíthatta és újra is kezdhette az animáció futását a **STOP**, **START**, **PAUSE** és **PLAY** gombok segítségével.




**13. ábra:** Animáció megjelenítésének jellemzői

Rendezésekről lévén szó, fontosnak találtam valamiképp jelezni azt, ha egy sorozat béli elem helyére került. Ezt sötétebb szürke háttérrel rendelkező dobozzal jelöltem. Így a felhasználó számára világossá vált az, hogy az sötétebb szürke elemeket már nem szükséges tovább hasonlítani vagy cserélni, hiszen a helyükre kerültek.




**14. ábra:** Helyes pozición található elemek szemléltetése

Az adott lépések interaktivitási szintjének függvényében a megjelenítések típusa is változott. Ezek kapcsán a következőket kellett biztosítanunk:

**Videó esetén:**

- A felhasználónak látnia kell a videót teljes méretében, és ha szeretné, akkor kinagyított formában is.
- Interaktív videó esetén a kérdéseknek nem szabad eltakarniuk az épp zajlódó mozzanatokat annak érdekében, hogy a felhasználó a lehető legjobb választ tudja megadni.

**Interaktív animáció és levezénylés esetén:**

- Mindkét tanítási lépés esetében kell, hogy legyen egy harmadik rész, amely az irányításért felel. Ebben a részben találhatja meg a felhasználó az összes olyan utasítást, melyre szüksége van az adott lépés véghezviteléhez (összehasonlítás és csere műveletét). Az elemek kiválasztását a középső részben található animálandó számsorozatra kattintva hajthatta végre.

**Kódépítés esetén:**

- A kódépítés tanulási lépés az előző interaktív lépésekhez hasonlóan három részre osztotta az algoritmus bemutatását: feladatok megfogalmazása, animáció és a kiegészítendő kód. Ennek kapcsán külön szerepet kaptak a színek, melyeket az adott paramétereket jelző nyilak és keretek segítségével szemléltettünk. Az i és j mutatók az animáció fölött jelentek meg nyilak formájában. Az ezzel összhangban megjelenő ciklushatárok i és j mutatók színe függvényében váltakoztak. Az i-vel kapcsolatos kiegészítendő rész i pointer színével, a j-vel kapcsolatos információ pedig j pointer színével egyezett meg.
- A ciklusszerkezet meghatározását szolgáló kis ablak az említett három rész (feladatok, animáció és kódrészlet) előtt jelent meg.

**Az életre kelt kód esetén:**

- Az kód életre keltésének lépésénél az eddig megjelenített feladatok rész eltűnik, hiszen itt a felhasználónak csupán figyelnie kell a kód által irányított animációt. Ennek következtében a segítségkérés és a hibák lehetősége is elmarad, hiszen teljes szinttű automatikus lejátszás jellemző a tanulási lépésre.
- A színek jelen esetben is fontos szerepet kapnak, hiszen az animáció adott fázisai a kód egy bizonyos részével együtt értékelődnek ki, melyet a kódsor kiszínezett keretével szemléltettünk.

### 4.4.4. Tanulási lépések jellemzői

A tanulási lépések függvényében az animáció tulajdonságainak megjelenítése is változott.

**„Fehér mód"**

- Az animálandó sorozat elemeit képező dobozoknak a mérete nyilvánvaló kell, hogy legyen a felhasználó előtt, illetve az is, hogy milyen számot tartalmaznak.

**„Fekete mód"**

- El kell rejteni a felhasználó elől a sorozat elemeinek méretét és a bennük lévő számokat. Így, csak akkor tudja meghatározni a felhasználó a következő helyes műveletet, ha figyelmesen követi végig az összehasonlítások és cserék műveleteit.

**Tanári bemenet**

- A tanfolyam létrehozásakor a tanár előre meghatároz egy számsorozatot. Ezt megmutatjuk a felhasználóknak, viszont nem adunk lehetőséget arra, hogy megváltoztassák (mint ahogy azt más esetekben megteheti). A számsorozat hosszát jelölő bemenet ez esetben nem szerkeszthető.

**Véletlenszerű bemenet**

- Az animálandó számsorozat ebben az esetben nem állandó. A felhasználónak meg kell engedni a számsorozat hosszának változtatását, és a „START" gomb megnyomás általi új számsorozat generálást.

**Legjobb bemenet**

- Előre meghatározott számú elem jelenik meg az animálandó sorozat részeként növekvő sorrendben. A felhasználónak lehetőséget kell adnunk a számsorozat hosszának megváltoztatására, viszont ennek jellege nem változhat: rendezések esetén a számsorozat elemeinek értékei mindig növekvő sorrendben maradnak; lineáris keresés esetén, mindig a keresett elem marad az első helyen.

**Legrosszabb bemenet**

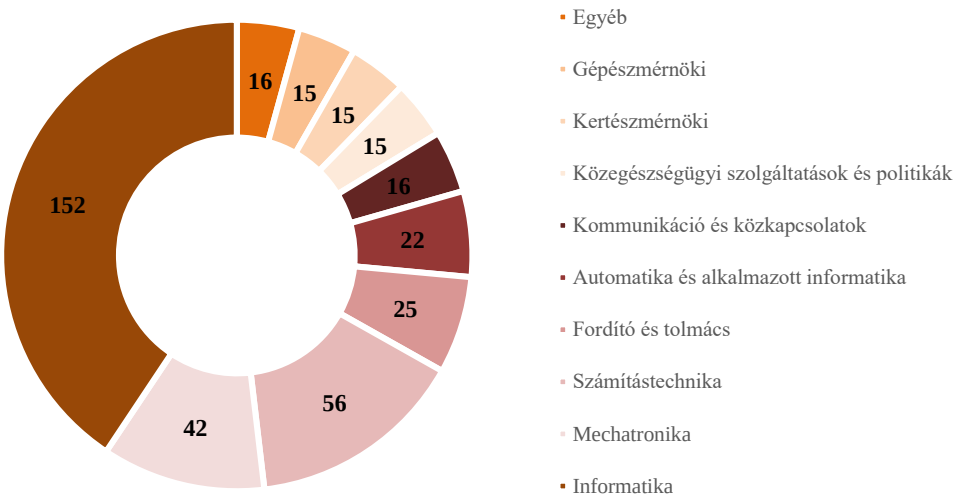
- Előre meghatározott számú elem jelenik meg az animálandó sorozat részeként csökkenő sorrendben. A felhasználónak lehetőséget kell adnunk a számsorozat hosszának megváltoztatására, viszont ennek jellege nem változhat: rendezések esetén mindig csökkenő sorrendben maradnak a számsorozat értékei; lineáris keresés esetén mindig legutolsó helyen szerepel a keresett elem, vagy egyáltalán nem szerepel a sorozat elemei között.

## 5. Összefoglalás

*„Vajon, ha egy diák nem tud úgy tanulni, ahogy tanítják, nem a tanárnak kellene úgy tanítani, hogy azt a diák is megértse?”* Sokszor elgondolkodunk Ignacio Estrada gondolatán, hiszen egy olyan oktatási technika kifejlesztése lenne a cél, amely kimagasló eredményeket érne el a diákok teljesítményét tekintve. Függetlenül a korosztálytól, a nemtől, a szakiránytól szeretnénk mindenki számára egy elérhető platformot biztosítani, ahol elsajátíthatók az algoritmikával kapcsolatos tudnivalók. Az AlgoRythmics oktatási környezet segít elsősorban a tanároknak, hiszen lehetőséget biztosít lecketervek dinamikus létrehozására, melye különböző tanulási lépések révén vezetik el a felhasználót a megértésig. Emellett, segítséget nyújt a felhasználóknak, hiszen mindenki úgy tanulhat, ahogy az számára a legkedvezőbb és legeredményesebb.

Mindenki más és más, és mindenki megérdemli, hogy tanuljon és tanítsák. Ez az E-learning oktatási környezet, annak köszönhetően, hogy az interaktivitás különböző szintjein nyilvánul meg, és a tanulási lépések különböző változatait használja az oktatásban, lehetőséget biztosít arra, hogy mindenki megtalálja a számára leghatékonyabb tanulási módszert.

Célunk, hogy minél inkább be tudjuk vonni az oktatási folyamatba a környezet használatát, és hogy ez által elősegítsük az algoritmusok megértésében való előrehaladást. 2019. májusától kezdve egyetemünkön már elérhetővé vált, és azóta közel 374 tanuló használta. Jelenleg hetente alkalmazzuk tanórákon, melyek során blended – learning oktatási formával tanítjuk diákjainkat és segítjük az algoritmusok megértését (15. ábra: **Regisztrált felhasználók száma szakok szerint**).




**15. ábra:** Regisztrált felhasználók száma szakok szerint

## Irodalom

1. Katai, Z. (2014, June). Selective hiding for improved algorithmic visualization. In Proceedings of the 2014 conference on Innovation & technology in computer science education (pp. 33-38). ACM.
2. Byrne, M. D., Catrambone, R., & Stasko, J. T. (1996). Do algorithm animations aid learning?. Georgia Institute of Technology.
3. Korhonen, A., & Malmi, L. (2000). Algorithm simulation with automatic assessment. ACM SIGCSE Bulletin, 32(3), 160-163.
4. Naps, T. L., Eagan, J. R., & Norton, L. L. (2000, May). JHAVÉ—an environment to actively engage students in Web-based algorithm visualizations. In ACM SIGCSE Bulletin (Vol. 32, No. 1, pp. 109-113). ACM.
5. Jarc, D. J., Feldman, M. B., & Heller, R. S. (2000). Assessing the benefits of interactive prediction using web-based algorithm animation courseware. ACM SIGCSE Bulletin, 32(1), 377-381.

# Interaktív webes alkalmazások használata a pénzügyi ismeretek oktatásában

Pšenák Péter[1], Pšenáková Ildikó[2], Szabó Tibor[3]

[1]peter.psenak@fm.uniba.sk, [2]ildiko.psenakova@truni.sk, [3]tszabo@ukf.sk
Univerzita Komenského, Fakulta managementu, Bratislava,
Trnavská univerzita v Trnave, Pedagogická fakulta
Univerzita Konštantína Filozofa v Nitre, Fakulta stredoeurópskych štúdií, Szlovákia

**Absztrakt.** Szlovákiában a pénzügyi ismeretek oktatása az általános és középiskolákban még eléggé gyerek cipőben jár. A múltban csak a gazdasági középiskolák és az egyetemek hallgatói találkoztak ezzel a kérdéssel. Ez az új témakör számos matematikai elemet tartalmaz, ezért nem vált a diákok kedvencévé. Számos eszköz létezik a diákok érdeklődésének fokozása iránt, az oktató feladata, hogy használja is azokat. A cikkben bemutatjuk, hogyan lehet az általunk készített interaktív webes alkalmazást felhasználni a pénzügyekkel és a pénzgazdálkodással kapcsolatos kérdések tanításához. Az alkalmazás elsősorban a felsőoktatás számára készült, de középiskolákban is használható.

**Kulcsszavak:** pénzügyi ismeretek, oktatás, interaktivitás, web alkalmazás

## 1. Bevezetés

A média egyre gyakrabban foglalkozik a pénzügyi ismeretek fogalmával, ennek ellenére még nagyok sok embernek problémái vannak a pénzügyeivel, ami azt jelenti, hogy hiányzik a pénzügyi műveltségük. Pedig a pénz hosszú távú kezelése nem nagy tudomány. Valójában meglehetősen egyszerű szabályok léteznek, melyek megértéséhez csak egy kis józan észre, önuralomra, csipetnyi bátorságra és alapvető matematikai ismeretekre van szükség. Aki gyermekkorában nem tanult meg ésszerűen viszonyulni a pénzhez, annak felnőttkorban nagyobb eséllyel lehetnek pénzügyi nehézségei. Ezért már korai gyermekkorban tanácsos kezelni ezt a kérdést, elsősorban családi környezetben, de nagyon fontos foglalkozni vele az iskolában is.

## 2. A pénzügyi műveltség fogalma

A pénzügyi műveltség az alapvető életképességek egyik elengedhetetlen eleme és egyike a legfontosabb pénzügyi fogalmaknak is. A pénzügyi műveltség valójában nem más, mint a pénzbevételi és –kiadási, hitelfelvételi és adósságkezelési ügyeknek az átlátása és a lehetőségek felismerése. [1]

A pénzügyileg művelt személy megérti a legfontosabb pénzügyi fogalmakat és képes magabiztosan kezelni személyes pénzügyeit. Mindehhez megfelelő rövid távú döntéshozásokat és hosszú távú pénzügyi tervezéseket tud létrehozni. A tervezésnél felismeri és figyelembe veszi a különböző életeseményeket és a folyamatosan változó gazdasági körülményeket egyaránt. [2]

A pénzügyi műveltséget a mindennapi életben gyakran alábecsülik. Empirikus adatok azt mutatják, hogy Szlovákiában a pénzügyi ismeretek szintje alacsonyabb, mint más közép-európai országokban. [3]

Alapjában véve a pénzügyi műveltség egy képesség, amelynek célja kihasználni tudásunkat, készségeinket és tapasztalatainkat saját anyagi erőforrásaink hatékony kezelésére annak érdekében, hogy megteremtsük az élethosszig tartó saját és háztartásunk anyagi biztonságát.

Megszerezni ezt a képességet családi nevelés és/vagy iskolai oktatás útján lehet. A gyermeknek már az első zsebpénzétől kezdve meg kellene tanulnia, hogy egy részét félre kell tenni a „rosszabb

időkre". A pénzügyi ismeretek első szabálya az, hogy „ne költekezzen többet, mint amennyit keresett". De ha egy gyermek látja a szülei pazarló életét, ugyanazokat a hibákat ismételi, akkor ezek nagy valószínűséggel a felnőttkorban még megsokszorozódnak.

Pénzügyi művelté akkor válunk, ha képesek vagyunk annyi pénzt keresni, amennyire szükségünk van saját kiadásaink fedezetére, céljaink megvalósításához és még valamennyi marad is. Mindenki azonban nem lehet pénzügyileg művelt. A pénzügyi szokások szinte megegyeznek az étkezési vagy viselkedési szokásokkal. Egyesek soha nem fogják megváltoztatni étkezési szokásaikat, még akkor sem, ha tudják, hogy azok ártalmasak egészségükre. Ez vonatkozik a pénzügyi ismeretekre is.

## 3. A pénzügyi ismeretek oktatásának jelenlegi helyzete Szlovákiában

Elsődlegesen az iskolák munkaerőt képeznek és a munkahely az egyik jövedelemforrás a felnőttkorban. Először meg kell keresni a pénzt, csak utána lehet vele gazdálkodni.

Az iskolának hozzáférést kellene biztosítani a pénzügyi piac kockázatait és előnyeit érintő gyakorlati információkhoz, hogy ez által a tanulók megérthessék a gazdaság alapelveit, ugyanakkor elegendő elméleti ismeretekre tegyenek szert az érintett problémakörből, melyeket majd a gyakorlatban is képesek lesznek alkalmazni.

Egy kutatás eredményei kimutatták, hogy a középiskolai oktatásban komoly problémák vannak a pénzügyi ismeretek terén. Egy 2018-ban végzett online kérdőíves felmérés alapján a középiskolai végzettséggel rendelkező fiatalok 65 %-a ismerte a pénzügyi ismeretek jelentését. Pozitívumként említhető, hogy a megkérdezettek 68 %-a legalább egy pénzügyekkel kapcsolatos tantárgyat tanult, ennek eredményeként ők ismerték, ill. értették az infláció fogalmát. Viszont sajnálatos az a tény, hogy a legtöbben nem ismerték a hitelkártya és a betéti kártya közti különbséget és nem tudtak különbséget tenni a folyó fizetési mérleg egyenlege és a bankszámlán rendelkezésre álló egyenleg között. A feladatok bonyolításával a résztvevők helyes válaszainak aránya gyorsan csökkent. A kutatást ismertető tanulmány az eredmények által arra a következtetésre jutott, hogy a tanulók nem rendelkeznek megfelelő pénzügyi ismeretekkel. [4]

2008-ban a Szlovák Köztársaság Oktatási, Tudományos, Kutatási és Sportminisztériuma kiadta a Pénzügyi műveltség nemzeti szabványát (Národný štandard finančnej gramotnosti – NSFG), amely a pénzügyi területre és a személyi pénzügyek kezelésére vonatkozó oktatási stratégiát határozza meg. A pénzügyi témák mellett, tartalmazza a fogyasztói nevelést, korrupcióellenes nevelést, vállalkozáshoz szükséges ismeretek oktatását és az állami források felhasználása során elkövetett csalások elleni nevelést is. [5]

Az NSFG a pénzügyi ismeretek oktatását kötelezőnek minősítette az általános és középiskolákban, továbbá meghatározta a pénzügyi ismeretek, készségek és tapasztalatok tartalmát a pénzügyi oktatás és a személyi pénzgazdálkodás területén, amelyeket a középiskolai végzettséggel rendelkező egyéneknek tudniuk kellene. A tananyag tartalma hat témakörbe van besorolva:

- Fogyasztói pénzügyi felelősségvállalás;
- Tervezés, jövedelem és munka;
- Fogyasztói döntéshozatal és menedzsment;
- Kölcsön és adósság;
- Megtakarítások és befektetések;
- Kockázatkezelés és biztosítás. [5]

Az egyes témakörök oktatásának sorrendjét az iskola határozza meg a saját oktatási programja keretein belül. Az általános iskolákban és a gimnáziumokban nincs konkrét tantárgy a pénzügyi

ismereteket oktatására, ezért az NSFG ajánlása alapján több lehetőség is alkalmazható az egyes témakörök beépítésére az oktatási folyamatba.

Az egyik lehetőség, hogy az egész témát több részre bontják és más tantárgyak tartalmához rendelik. Ezt követően a tantárgyakat oktató tanárok kidolgozzák az egyes részek tartalmát és integrálják azt az oktatásba. A másik lehetőség elemezni az oktatás aktuális tartalmát, minden tanárnál megtalálni a pénzügyi ismeretek elemeit, amelyeket az oktatásban alkalmaz és ezeket további elemekkel bővíteni, amelyek beépíthetők az adott tantárgyba. A szabvány nem írja elő, hogy melyik lehetőséget kell alkalmazni, az iskola dönti el, hogy melyiket választja.

A pénzügyi ismeretek oktatásának sikeres megvalósításához elengedhetetlen, hogy maguk az oktatók is tisztában legyenek a pénzügyi műveltség lényegével, csak akkor lesznek képesek helyesen dönteni, hogy mely tantárgyakba tudják beépíteni és milyen módszereket fognak alkalmazni az oktatására.

A folyamatosan változó környezetünkben, amelyben élünk, egyre szükségesebbé válik, hogy a tanárok modern oktatási technikákat alkalmazzanak a tanórákon. Emellett egyre növekvő az igény az emberek pénzügyi ismeretének magasabb szintre emelésére. Többféle platform létezik a pénzügyi készségek elsajátítására, ugyanakkor azon egyének számára, akik teljes körű elméleti tudásra és gyakorlati készségekre szeretnének szert tenni e területen, a legjobb választás a megfelelő egyetemi képzés igénybevétele.

A nem közgazdaságtan felé orientált középiskolákban leggyakrabban az informatika jellegű tantárgyakba integráljak a pénzügyi ismeretek oktatását. Ennek elsődleges oka az, hogy az informatika keretein belül oktatják az MS Excel táblázatkezelő programot, amely tartalmaz néhány statisztikai függvényt is. Ezek alkalmasak bizonyos pénzügyi ismeretekkel kapcsolatos elméleti kifejezések magyarázatára és pénzügyi feladatok megoldására.

Vannak azonban más platformok is a pénzügyi készségek tanítására, például az R-Studio program csomag, amelyben készült a következőben bemutatott alkalmazás is.

## 4. Felhasznált programok rövid ismertetése

Az alkalmazás létrehozására az ***R-Studio*** programkörnyezetét használtuk, mely ingyenesen letölthető az R-studio weboldaláról, de megtalálható az *Anaconda* programcsomag részeként is. Az Anaconda a világ egyik legnépszerűbb Python és R adattudomány (data science) platformja. A felhasználók száma jelenleg több mint 4,5 millió. [6]

Az R-Studio egy R-el integrált ingyenes online fejlesztői környezet (IDE). Ez magába foglal egy konzolt, szintaxiskiemelő szerkesztőt, amely támogatja a közvetlen kódvégrehajtást, valamint a rajzoláshoz, előzményekhez, hibakereséshez és a munkaterületek kezeléséhez szükséges eszközöket. A konzolban használt nyelv az R-nyelv.[7]

Az R egyik erős oldala az egyszerűség, amellyel könnyen készíthetők magas minőségű táblázatok, grafikonok, szükség esetén matematikai szimbólumokkal és képletekkel kiegészítve. Ezt a nyelvet ajánljuk azoknak az oktatóknak a figyelmébe, akik a pénzügyi ismeretek tanításához megfelelő oktatási anyagokat szeretnének készíteni, ugyanis könnyen kezelhető és viszonylag jó tanulási görbével rendelkezik.

Az R egy népszerű és aktuális nyelv. Az egyik legkedveltebb programozói fórum a TOP 20 programozó nyelv közé sorolta. A fórum által végzett kutatás világszerte zajlott. [8] Az R adat analízisre volt kifejlesztve az S nyelvből - gyakorlatilag az S nyelv ingyenes változata - és leggyakrabban statisztikai szoftverként használják.

Az R nyelv azonban önmagában nem lenne elegendő webes alkalmazások létrehozásához, mivel elsősorban script nyelvként használják. Mivel az R meglehetősen népszerű a felhasználók táborában

és „open source” tulajdonságának köszönhetően a programozók közössége, akik részt vesznek az R karbantartásában és új modulok programozásában, így létrehoztak a weblapok készítésére egy speciális modult a Shiny-t.

A Shiny egy ingyenesen letölthető könyvtár, amely lehetővé teszi egy webszerver létrehozását az R számára. A könyvtárnak köszönhetően a forráskód elküldhető egy webszerverre, és általa létrehozható egy webes alkalmazás. Ehhez viszont még több más könyvtár és webszerver-beállítás is szükséges. Összegezve, ez lehetővé teszi teljes interaktív webes alkalmazások létrehozását. [9]

A webes alkalmazások létrehozása során felmerülő problémák nagy része a gyengébb eszközökre való optimalizálás, amelyre a hivatásos programozók sok időt fordítanak. A Shiny könyvtár az optimalizálást megoldja a fejlesztő helyett, ezért nem kell foglalkoznia az alkalmazás sebességével, elég, ha az interaktív elemekre összpontosít.

A Shiny könyvtárban készült alkalmazás csak a forráskódot létrehozó felhasználó számítógépén fut, de a forráskód feltölthető a Shiny weboldalra is és így szabadon terjeszthető. Az ingyenesen feltöltött alkalmazások és látogatók száma azonban limitált. Ha több weboldal közzétételére vagy a látogatók számának növelésére van szükség, a szolgáltatás (shinyapps.io) fizetős módra vált. [10]

Egy másik lehetőség, egy saját szerver beállítása és a Shiny alkalmazások futtatása rajta. A webszerver futtatásához szükséges kód teljesen ingyenes. A szolgáltatásokkal kapcsolatos további információkért ajánljuk meglátogatni a hivatalos weboldalt vagy a megfelelő fórumokat.

A leggyakrabban használt fórumok közé az alábbiak tartoznak:

- Stack Overflow, [11]
- R-bloggers, [12]
- RStudio Community Forum, [13]
- rOpenSci fórum. [14]

A Shiny-ben írt alkalmazásnak van néhány nagyszerű tulajdonsága, amelyek nagyon praktikus eszközzé teszik. Az alkotók, többek között észrevették a mobil eszközök használatának növekvő globális tendenciáját, így a létrehozott webes alkalmazások automatikusan reagálnak a képernyő méretének változásra. Ezt a CSS keretrendszer a Bootstrap teszi lehetővé. Előre beépített kódot biztosít a webhelyek vagy webes alkalmazások fejlesztésének felgyorsításához, bebiztosítva, hogy az előre beállított stílus megmarad. A keretrendszert a webhelyek és webes alkalmazások programozói ismerik, és gyakran használják. [13]

A Bootstrap-ot a Twitter saját alkalmazásuk fejlesztésére hozta létre, majd később nyílt forrású eszközként tették közzé. Ezért, mint a Shiny és az R, a Bootstrap is ingyenesen használható licenc megvásárlása nélkül.

Az R és a Shiny kombinációjának köszönhetően egy olyan webes alkalmazást fejlesztettünk ki, amely alkalmas a pénzügyi ismeretek oktatására.

## 5. Az alkalmazás leírása

Az általunk bemutatott interaktív didaktikus webes alkalmazás közgazdasági és pénzügyi egyetemek hallgatói számára készült. Alapvetően a pénzgazdálkodás kérdéseivel foglalkozik.

A tárgyalt témák túlságosan bonyolultnak és matematikainak tűnhetnek, de alapvetően ezek a pénzügyi munka alapelvei, mint például a pénz időértéke, járadék, a befektetés megtérülése, befektetési kockázat, belső megtérülési ráta, a pénzáramok diszkontálása és pénzügyi tervezés. Az alkalmazás tartalma azokat az információkat egészíti ki, amelyeket a hallgatók már az osztályteremi oktatásban elsajátítottak és olyan információkat is nyújt, amelyeket esetleg félreérthettek a tantárgy tanulása közben. A tartalom szükség szerint bővíthető, illetve további témakörök is feldolgozhatók.

Az alkalmazás szerkezete lényegében egyszerű. Váltakoznak benne az elméleti részek, amelyek a probléma magyarázatát tartalmazzák, a gyakorlati részekkel, amelyekben megjelennek a számításokhoz szükséges képletek, majd ezeket követi a képletek szerinti számítások gyakorlati bemutatása. Ebben a részben találhatók a beépített interaktív elemek, amelyek segítségével különféle értékeket lehet a képletekbe beállítani. Ez által változnak az eredmények, melyek grafikonokon vagy más formában jelennek meg a képernyőn. A hallgatók így egyszerűen tudják követni, milyen változást hoz egy-egy paraméter értékének a változtatása.

A weboldal bevezető részében röviden, de „velősen" mutatjuk be a szükséges tényeket, amelyekkel az elméleti részben fogunk majd foglalkozni. Ezzel arra törekszünk, hogy a lehető legnagyobb figyelmet kapjuk a hallgatóktól, még akkor is, ha a figyelmük olvasás közben lankad. Ilyen módon a weboldal nem hasonlít a tankönyvhöz. Azt szeretnénk elérni, hogy a hallgatók a weboldalt úgy használják, mint egy „játékot", amely segít megtanulni és megérteni a tananyagot.

Az alkalmazást a pedagógus is használhatja, aki ebben az esetben emlékeztetheti a hallgatókat arra is, hogy az alkalmazás az egyes témák jellegének gyors megismerése szolgál, hogy csupán a pénzügyi képletek áttekintését nyújtja és a tananyag gyorsabb megértését segíti, mert szemlélteti a változásokat, viszont nem elegendő az adott téma átfogó és teljes elsajátításához.

Az alkalmazás felépítését a pénz időértékének konkrét példáján tárgyaljuk (1. ábra).


**1. ábra:** Elméleti rész

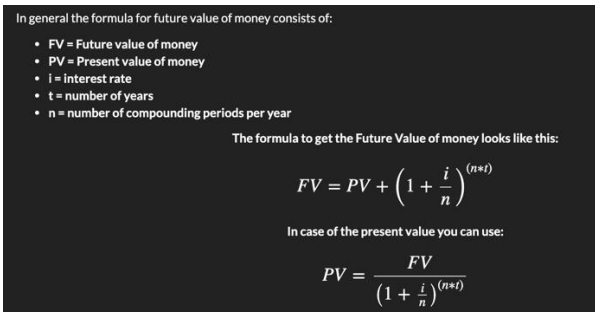

**2. ábra:** Az alkalmazás arculata az iPhone 5 képernyőén

Annak érdekében, hogy megtanítsuk a hallgatókat, arra hogy mobilkészülékeiket használják más célokra is, nemcsak csevegésre vagy fényképezésre, megkérjük őket, hogy vegyék elő mobiltelefonjukat és keressék meg az online interaktív alkalmazásunkat a weben. Ugyanazt a tartalmat fogják látni a mobilkészülékük kijelzőjén, mint az interaktív táblán vagy a számítógép képernyőjén, természetesen más elrendezésben (2. ábra). Ez a már említett alkalmazkodási (reszponzivitás) lehetősége az alkalmazásnak.

Az alkalmazás ellentétes irányban is méretezhető, ezért az interaktív táblán vagy interaktív kivetítőn is probléma és minőség romlás nélkül megjeleníthető. Az oktató előnye tehát az, hogy a tananyagot csak egyszer kell létrehoznia egyetlen forráskóddal, és nem kell foglalkoznia az optimalizálásával különféle eszközökre.

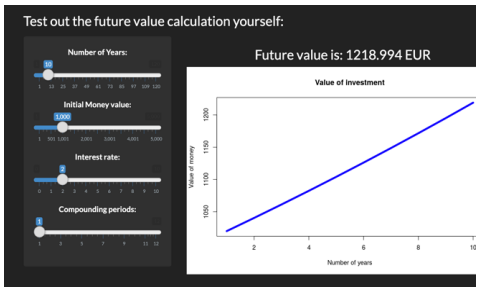
Az alkalmazás második részében találhatóak a számításhoz szükséges képletek. Ez a rész azoknál a tantárgyaknál fontos, ahol matematikai számításokra van szükség. Gyakran megtörténik, hogy a megfelelő szoftver hiánya miatt az alkalmazások készítői a képletet egy sorba írják, ami megnehezíti annak megértését, átláthatóságát. Alkalmazásunkban a megfelelő képletkészítő modulok használatá-

val a képletek formásak, láthatóak és könnyen értelmezhetők (3. ábra). A képletek a pénz jövőbeli értéke számítására szolgálnak. A pénz jövőbeli értéke egyszerű, de ugyanakkor a pénzgazdálkodás egyik legfontosabb fogalma, amelynek célja meghatározni, mennyi lesz a pénz értéke, ha valamilyen módon és bizonyos feltételek mellet beruházásra kerül. [15]

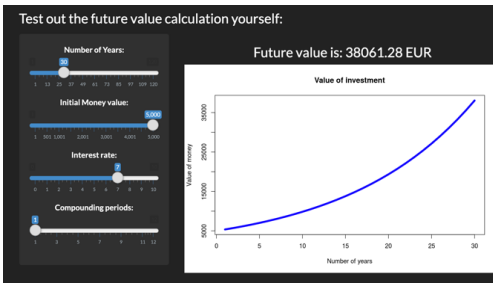



**3. ábra:** Példa a matematikai képletek megjelenésére.

Az interaktív alkalmazás legfontosabb része a harmadik. Az elméleti rész és a fenti képletek tanulmányozása után a hallgatóknak nem kell manuálisan vagy számológéppel végezniük a számításokat, hanem lehetőségük van négy csúszkával „játszani". Ezeket tetszőlegesen állíthatják be a nekik szükséges értékekre. A weboldal betöltése után megjelennek az alapértelmezett értékek (4. ábra).




**4. ábra:** Alap beállítások, értékek és grafikon




**5. ábra:** Néhány értékbeállítás után

Az oldal tartalma két részre van elosztva. A baloldalon vannak elhelyezve a csúszkák, amelyek segítségével megváltoztathatók az alapvető bemeneti változók értékei. A változások automatikusan beilleszkednek a képletbe, és megjelenik az aktuális eredmény. A jobboldali részben grafikon látható, amely a pénz jelenlegi helyzetét mutatja be a változók beállított értékeitől függően, és fölötte megjelenik a pénz jövőbeli értéke. Látható, hogy az alapértékek beállításánál a grafikon szinte lineáris. (4. ábra)

Bármely interaktív elem elmozdításával a grafikon alakja valós időben megváltozik, és a pénz jövőbeli értékei is idővel frissülnek. Megjegyezzük, hogy a görbe fokozatosan alakul ki, mint az exponenciális függvény görbéje. Ennek oka a számlánkban tartott évek számának és az általunk meghatározott kamatlábnak a növekedése. Végeredményben, ha a változók értéke megváltozik, akkor a grafikon is változik (5. ábra).

Természetesen a fenti számítások nem lennének elegendőek ahhoz, hogy egy menedzser vagy igazgató eldöntse, vajon elég jó-e a befektetés. Helyénvaló lenne legalább összehasonlítani a kamatlábat az infláció szintjével, és figyelembe venni az időtényezőt is.

Az alkalmazásba képeket, fotókat is be lehet illeszteni, ez természetesen függ a tananyag tartalmától, illetve, hogy milyen korosztálynak készül az interaktív tananyag. Az alkalmazást letöltési sebességre optimalizáltuk, ezért gyengébb internet kapcsolattal rendelkező helyeken is jól használható.

A bemutatott interaktív webes alkalmazást folyamatosan frissítjük, és újabb tananyagokkal bővítjük. A fejlesztési folyamatba megpróbáljuk bevonni a hallgatókat is. A hallgatók véleménye az alkalmazásunkról eddig pozitív, és örömüket lelik benne, hogy segítenek nekünk, akár a szövegek kiegészítésében, esetleges hibák javításában, vagy további anyagok készítésében.

Az eddigi eredmények az alkalmazásunk használata alapján azt mutatják, hogy teljesíti célunkat, segíteni a hallgatókat a tárgy sikeres elvégzésében. Az alkalmazás segíthet a jövőbeni és mostani pedagógusoknak is, akik érdekeltek a pénzügyi műveltség javításában és a pénzügyi ismeretek oktatásában, mivel továbbadják tudásukat az általános és a középiskolák tanulói számára.

## 6. Befejezés

A modern oktatás komplex problematikája nem oldható meg különálló, zárt megoldásokkal. Ahhoz, hogy a lehető legjobb eredményeket érjük el az elméleti ismeretek és a gyakorlati készségek átadásakor mind a tanulók, mind a hallgatók számára, több különböző pedagógiai gyakorlat szinergiáját kell felhasználnunk.

A szlovákiai általános és középiskolákban az alapvető pénzügyi ismeretek oktatásának jelenlegi állapotában az oktatók számára lehetővé kell tenni, hogy a pénzügy kérdését a lehető legtermészetesebb módon integrálják különböző tantárgyakba. Az interaktív tananyagok szinte természetszerűen illeszkednek elsősorban az informatika tantárgyához.

Elképzelhető, hogy a jövőben az interaktív webes alapú tankönyvek lesznek „a menők" az általános és középiskolás tanulók körében, mivel ők már egyértelműen „digitális bennszülöttek"-nek minősülnek. [16]. Már ma is sok olyan pedagógussal találkozunk a gyakorlatban, elsősorban az egyetemeken, akik digitális (.pdf) formátumban biztosítják a tananyagokat a hallgatóknak, de sajnos, ez nem biztosítja az interaktivitás lehetőségét. Pedig az interaktív elemek beépítése az anyagokba pozitív hozzáadott értéket jelenthetne az oktatók és hallgatók számára is.

Tisztában vagyunk azzal, hogy az interaktív tananyagok, különféle didaktikai alkalmazások és interaktív tesztek létrehozása, annak ellenére, hogy már különböző támogatási csomagok állnak rendelkezésre az előkészítésükhöz, nem könnyű feladat. Sok pedagógus éppen azért nem használja ezeket, mert nem képesek minőségi, tartalmilag és módszertanilag helyes interaktív tananyagot létrehozni.

A bemutatott interaktív alkalmazás csak az egyik a modern eszközök közül, amely segítheti a hallgatókat tanulmányaikban.

## Köszönetnyilvánítás

A tanulmány megjelenését a KEGA 015TTU-4/2018: „Interaktivita v elektronických didaktických aplikáciách." (Interaktivitás az elektronikus didaktikai alkalmazásokban) című projekt támogatta.

## Irodalom

1. *Pénzügyi Tudakozó: Miért fontos a pénzügyi műveltség, és hogyan fejlesztheted?* (2016). https://penzugyi-tudakozo.hu/miert-fontos-a-penzugyi-muveltseg-es-hogyan-fejleszthete d/ [utoljára megtekintve: 2019.11.13.]
2. D. L. Remund: Financial Literacy Explicated: The Case for a Clearer Definition in an Increasingly Complex Economy. Journal of Consumer Affairs, (2010) vol. 44, no. 2. 276–295
3. L. Klapper, A. Lusardi, P. van Oudheusden: *Financial Literacy Around the World* https://gflec.org/wp-content/uploads/2015/11/Finlit_paper_16_F2_singles.pdf [utoljára megtekintve: 2019.11.10.]

4. K. Rentková, Ľ. Mitková: *Finančná gramotnosť na Slovensku*. In: Horizonty podnikateľského prostredia 4. Bratislava: Univerzita Komenského v Bratislave, (2018) 156–163
5. *Národný štandard finančnej gramotnosti verzia 1.2*. (2017) Ministerstvo školstva, vedy, výskumu a športu Slovenskej republiky, and Ministerstvo financií Slovenskej republiky. https://www.minedu.sk/data/att/11358.pdf. [utoljára megtekintve: 2019.10.19.]
6. *What is Anaconda.* https://www.anaconda.com/what-is-anaconda/ [utoljára megtekintve: 2019.11.13.]
7. *R: The R Project for Statistical Computing*. https://www.r-project.org/ [utoljára megtekintve: 2019.10.1.]
8. *Stack Overflow Developer Survey* (2019) https://insights.stackoverflow.com/survey/2019/?utm_ source=social-share&utm_medium=social&utm_campaign=dev-survey-2019 [utoljára megtekintve: 2019.10.21.]
9. *Shiny*. https://shiny.rstudio.com/ [utoljára megtekintve: 2019.10.20.]
10. *shinyapps.io* https://www.shinyapps.io/. [utoljára megtekintve: 2019.10.20.]
11. *Stack Overflow Where Developers Learn, Share, & Build Careers* https://stackoverflow.com/ [utoljára megtekintve: 2019.10.21.]
12. R-bloggers|R news and tutorials contributed by hundreds of R bloggers. https://www.r-bloggers.com/. [utoljára megtekintve: 2019.10.20.]
13. *RStudio Community*. https://community.rstudio.com/. [utoljára megtekintve: 2019.10.21.]
14. *rOpenSci Discuss*. https://discuss.ropensci.org/. [utoljára megtekintve: 2019.10.21.]
15. N. Gardner: *The Time Value of Money: A Clarifying and Simplifying Approach*. Journal of College Teaching & Learning (TLC), vol. 1, Jan. (2011)
16. *M. Prensky: Digital Natives, Digital Immigrants* (2001). https://www.marcprensky.com/ writing/Prensky%20-%20Digital%20Natives,%20Digital%20Immigrants%20-%20Part1.pdf

# Az interaktivitás szerepei az idegen nyelv oktatásában

Pšenáková Ildikó[1], Godiš Tomáš[2], Štrbo Milan[3]

{[1]ildiko.psenakova, [2]tomas.godis, [3]milan.strbo}@truni.sk
Trnavská univerzita v Trnave – Nagyszombati egyetem, Pedagogická fakulta – Pedagógiai Kar, Szlovákia

**Absztrakt.** Az új kommunikációs technológiák egyre fontosabb szerepet játszanak az idegen nyelv tanításában is. Az idegen nyelvek oktatásában nemcsak a tanítási és tanulási folyamat módszertana és megközelítése változott, hanem a tanár és a tanuló hagyományos szerepe is. A tanároknak igyekezniük kell, hogy az új eszközöket beillesszék saját pedagógiai eszköztárukba, így elérve, hogy a modern technikai eszközök használata hatékony legyen a tanítási folyamatban. Cikkünkben azt tárgyaljuk, hogy miként lehet kihasználni az interaktivitást az idegen nyelvek oktatásában.

**Kulcsszavak:** interaktivitás, oktatás, idegen nyelv, multimédia

## 1. Bevezetés

Az információ gyors terjedése, a folyamatosan fejlődő számítástechnika és az új műszaki találmányok korszaka radikálisan megváltoztatta az emberek életét. Manapság alig van már olyan emberi tevékenységi terület, amelyet ez a technológiai fejlődés nem érintene. Akár a gazdaság, a tudomány, a kommunikáció, az oktatás, a munka, a mindennapi élet vagy a hobbi – mindenhol olyan új technológiákat találunk, amelyek nemcsak hatékonyabbak és tökéletesítik az emberi tevékenységet, de mindenekelőtt megkönnyítik azt.

Számos terület, amely befolyásolta a technológiai fejlődést, magában foglalja az oktatási folyamatot is. Az új technológiák a tanítási módszereket is erősen megváltoztatták. Olyan médiumok kerültek be az oktatásba, mint a televízió, a videó, a számítógép, az interaktív tábla, az internet, a szociális hálózatok és természetesen a különféle tanulási és oktatási programok és alkalmazások, amelyek nemcsak modernizálják a tanítási órákat, hanem szemléletesebbé és érdekesebbé is teszik azokat.

Az új média lehetővé teszi a tantárgy tartalmának világosabb bemutatását, az autentikus és a valós helyzetek szimulálását, a tanulók aktív részvételét, az önálló tanulás előmozdítását, a gyengébb tanulók jobb támogatását és a motiváció növelését. Ezen kívül a média egy másik információforrást is kínál, ahol a tanuló viszonylag gyorsan és különösebb nehézségek nélkül hozzáférhet a kívánt információkhoz.

De az új média nemcsak előnyöket, hanem hátrányokat is hoz. Általuk helytelen információk is terjeszthetők, és a médiumok fokozott használata az osztályban negatívan befolyásolhatja a tanár és a tanuló közötti interakciót is. A veszélyek kiküszöbölése azonban a tanár feladata. [1]

Ma már a modern média és kommunikációs technológiák az oktatási folyamat állandó társaivá váltak. Ennek ellenére továbbra is vannak olyan tanárok, akik inkább a tradicionális oktatás formáját részesítik előnyben. Ennek különböző okai lehetnek, mint például az iskolák rossz technikai felszereltsége, de gyakoribb oka az, hogy a tanár médiaműveltsége kevés vagy egyáltalán nincs.

## 2. Média az idegen nyelv tanításában

Az 1950-es években az idegen nyelv oktatása a nyelvi ismereteket a nyelvtudás elé helyezte. A tanóra alapját a nyelvtani szerkezetek megtanulása és valamilyen kulturális vagy történelmi szövegek fordítása képezte. A kiejtési képzés csak a tanár kiejtésének utánozását jelentette. [2]

Az első elektronikus média megjelenésével világossá vált, hogy ezek jelentősen hozzájárulhatnak az idegennyelv-oktatásának sikeréhez is. Így a 20. század 40-es éveiben először az auditív a 60-as években pedig az audiovizuális módszer jelent meg. Mindkét módszer célja a korábban elhanyagolt idegen nyelven való beszéd és hallgatás készségeinek továbbfejlesztése és a kommunikációs készségek előmozdítása volt. Noha ezek a fogalmak jobb tanulási hatékonyságot eredményeztek, a nyelvtan és a frontális oktatás továbbra is fontos szerepet játszott. A felhasznált anyagok didaktikai videók vagy hangfelvételek voltak (mesék, dalok, versek...). Ezek a módszerek azonban jelentősen hozzájárultak a kiejtés javításához. [2]

Az 1970-es években megváltoztak a társadalmi körülmények, amelyek befolyásolták az oktatást is és létrejött egy új didaktikai koncepció a kommunikatív didaktika, amely a tanulót helyezi az tanóra középpontjába. A kommunikációs módszer már nem a tananyagon, hanem a tanulón, mint az oktatási folyamat tárgyán alapul. A tanulási folyamatban a tanulót partnerként kell értelmezni. [2] A tanár nem csupán információforrás, hanem asszisztens, aki segíti a tanulót a tartalom megértésében. Már nem dominál a frontális tanítás, hanem a kommunikációs oktatás formái, mint például a párbeszéd gyakorlatok, csoportmunka, együttműködési munka (nyelvi feladatok közös megoldása) vagy egyéni és projektmunka. A tanítás középpontjában a kommunikatív struktúrák és maga a kommunikáció áll, bár a nyelvtani struktúrák fontosak, de nem dominálnak az osztályban. Minden készség (beszéd, hallgatás, olvasás és írás) ugyanazt a jelentőséget kapja. A kommunikációs didaktika lehetővé teszi az új média (videó, számítógépes technológia, oktatási programok és interaktív oktatási szoftverek) teljes integrálását az osztályba, ezáltal az osztályteremi oktatás érdekesebbé és hatékonyabbá válik.

A média, amelyet a 70-es években használtak az osztályban (rádió vagy videó), az idegen nyelv oktatásának ma is szerves része. A számítógépes technológia gyors fejlődése azonban az ezredforduló körül új oktatási lehetőségeket hozott létre. A közösségi hálózatok, multimédiás oktatási programok és interaktív alkalmazások egyre inkább beilleszkednek a tanítási folyamatba, és befolyásolják az oktatási módszereket és gyakorlatokat. [3]

Ismert tény, hogy a multimédia több egyedi média (szöveg, hang, animáció, videó) rendszerbe történő integrálásával jön létre. Tiňáková szlovák didaktikus hangsúlyozza, hogy a multimédiás médiumoknak a nyelvtanításban történő alkalmazása manapság sokkal hatékonyabban közvetíti az oktatás tartalmát, mint egyetlen médium, mivel a tanuló több érzéke egyidejűleg stimulálódik, miközben a tanítási tartalom érthetőbb. Jellemző példa egy számítógépes program, amely egyidejűleg szöveget, képet, beszélgetést, videót vagy klipet kombinál. [3] Véleményünk szerint a számítógépet univerzális multimédiás közegnek tekinthetjük, amely minden médiaformát egyesít.

Nyilvánvaló, hogy az új oktatási módszerek új kompetenciákat igényelnek a tanároktól. A médiakompetencia azonban nem csak azt jelenti, hogy a tanár tudja használni a számítógépet és az interaktív táblát, vagy megfelelő információt tud találni a weboldalakon, hanem azt is, hogy ezeket a médiákat módszertanilag helyes és hatékony módon tudja használni az osztályban. A média kompetencia tartalma folyamatosan bővül – mindig újabb és újabb programok jelennek meg, ezért az oktatók ismereteinek is folyamatosan bővülniük kell. [4]

Ma már nem elég, hogy a tanár a kész oktató programokat vagy alkalmazásokat mechanikusan integrálja a tanórákba, hanem előnyére válik, ha aktívan és kreatívan készít új, személyre szabott programokat is, amelyek megfelelnek a tanulók igényeinek. A tanároknak ezért szükséges aktívan részt venni saját tanítási és tanulási programok és alkalmazások elkészítésében. [5]

Úgy tűnhet, hogy a nyelvtanítás történelmében a különböző módszerek divatszerűen változnak. Egy korábbi letűnt módszer, vagy annak elemei időről-időre visszatérnek. Ezek a változások a társadalom változásának, illetve a nyelvtudással kapcsolatos társadalmi elvárások változásának köszönhetőek. Egy új nyelvtanítási módszer nem kizárólag a szakma belső változásainak következtében jön létre, hanem sokkal inkább annak következményeként, ahogy a nyelvtanítás-nyelvtanulás világa reflektál a társadalmi, politikai, pénzügyi, oktatáspolitikai, tudományfejlődési kihívásokra. [6]

Lehet nincs messze az az idő, amikor a hagyományos papíralapú munkafüzetek, feladatgyűjtemények, szótárak, könyvek kora lejár. Helyükre a 21. századi igényeknek megfelelő interaktív könyvek lépnek, amelyek nemcsak olvasható-írható feladatokat nyújtanak, hanem különböző képi és hanghatásokkal, sőt, akár háromdimenziós grafikákkal jeleníthetik meg az iskolai tananyagot. Ezek az alkalmazások sokkal életszerűbbé és élvezetesebbé tennék a tanulást, ezáltal hatékonyabbak is lehetnének, mint a hagyományos taneszközök.

## 3. Az interaktivitás szerepei az oktatásban

Az interaktív elemek használatát az elektronikus didaktikus tananyagban a számítógép és a felhasználó között létező interakció teszi lehetővé. Ha a multimédiás tananyagot egészítjük ki interaktív elemekkel, akkor az egyes multimediális elemeket is irányítani lehet, mint a tanár, mint a diák oldaláról. Gondolunk itt a hang be- és kikapcsolására, szimuláció, animáció vagy videó indítására stb.

Az interaktív tananyagok és alkalmazások a tanítási folyamatban azonnali hatékonyságot ösztönözhetnek, mivel figyelmeztethetik a diákokat az esetleges hibákra és lehetőséget adnak azok javítására is. A diákok azonnali visszacsatolást kaphatnak arról is, hogy jól vagy rosszul oldották meg a feladatot, helyesen válaszoltak-e a kérdésekre, szükség esetén igénybe vehetik a felajánlott segítséget és az azonnali végeredmény megjelenése csökkenti a stresszt, amely az eredményre várással jár, és meggyorsítja a tanár munkáját is.

Az interaktivitás lényegéből kifolyólag különböző interaktív elemeket lehet az oktatásban felhasználni. Ha a tanár hatékonyan szeretné ezeket alkalmazni, ismernie kellene a lehetséges szerepeiket. Tiňáková [3] és saját tapasztalataink alapján, következőképen foglaltuk össze az interaktív alkalmazások, tananyagok funkcióit.

- *Bemutató funkció*: a tanár multimédiás prezentáció segítségével mutatja be a tananyagot, amely világosabb és érthetőbb módon közvetíti a tartalmat a tanulók felé. A prezentációba interaktív elemekként linkeket épít be és így a tartalmat különféle információforrásokkal (enciklopédiák, cikkek, weboldalak) kapcsolja össze;
- *Motivációs funkció*: az interaktív multimédiás programok, alkalmazások érdekesebbé teszik a tanórát, tükrözik a tanulók érdekeit, és motiváló hatással vannak rájuk; A tanár a tananyag motiváló hatását, beépített gazdag képgalériával és bőséges tananyagforrásokkal is fokozhatja. [7]
- *Kommunikatív funkció*: az interaktív multimédiás programok lehetővé teszik a kétirányú egyéni (video beszélgetés) vagy csoportos (videokonferencia) kommunikációt;
- *Individualizációs funkció*: oktatási program segítségével különböző nehézségi szintű gyakorlatokat vagy feladatokat, különböző munkaidővel rendelhetünk különböző tanulókhoz - az egyéni igényeknek megfelelően;
- *Szimulációs szerep*: interaktív alkalmazás segítségével a diáknak olyan valós helyzeteket lehet teremteni (szimulálni), amelyekkel szembesülhet a jövőben. Így megtanulhatja, hogyan kell helyesen viselkedni egy konkrét helyzetben, és hogyan tudja szó szerint megoldani a megadott problémát (pl. hívja a rendőrséget és magyarázza el a bűncselekményt, vonatjegyek telefonos foglalása, áruk online áruházból történő megrendelése...);

- *Autonóm tanulás*: az interaktív elemek lehetővé teszik a diák számára, hogy eldöntse, melyik feladatot kívánja előbb megoldani, otthoni munka esetében azt is, hogy mikor és mennyi időt szentel ezek kidolgozására, azt is eldöntheti, hogy megoldjon önként valamilyen gyakorlatot, vagy csak a kötelezőket, vagyis ő vállalja a felelősséget a tanulás előrehaladásáért;
- *Ismétlési funkció*: a programok és alkalmazások lehetővé teszik a megszerzett tudás és készségek ismétlését vagy gyakorlását nem csak az órák során, hanem a tanulók egyéni felkészítésének részeként is (az oktatási intézményen kívül);
- *Ellenőrzési funkció:* a tanulási folyamat ellenőrzésére interaktív tesztek használhatók. Ha online oldják a diákok a feladatokat, akkor a tanár gyors visszajelzést kaphat a tanulók ismereteiről, tudásáról és a tananyag hasznosságáról.

Még ha tanulmányunkban az idegen nyelv oktatására próbáltunk is összpontosítani, a felsoroltak alapján szinte nyilvánvaló, hogy ezek a szerepek, funkciók nemcsak az idegen nyelv oktatására érvényesek, hanem más tantárgyakban is érvényesülhetnek.

# 4. Interaktív feladatok

A nyelvtanítás módszertana annyira összetett, hogy szinte lehetetlen megjelölni egy olyan módszert, amely univerzális, elméletileg következetes és annyira hiteles, hogy minden kontextusban használható. [7]

Ma a pedagógiai kutatások azt támasztják alá, hogy a személyre szabott oktatásnak, a saját ütemben haladásnak fontos szerepe van a hatékony tanulásban, és ez különösen érvényes az idegen nyelv tanulására is. Az interaktív tananyagok használata sokszor megoldhatja ennek az igénynek a kielégítését, mivel a diák átveheti az oktatási-tanulási folyamat irányítását. Saját igényei szerint haladhat a tananyagban, visszaléphet, vagy továbbléphet a következő részre, és annyi időt tölt el egy feladaton, amennyire szüksége van. A tanulási útvonalat a diák aktuális tudása is befolyásolhatja, mert az alkalmazás nem is engedi tovább, amíg a köztes kérdéseket nem tudja helyesen megválaszolni. [8]

A leendő pedagógus hallgatóinkat bevonjuk az interaktív tananyagok, tesztek, feladatok tervezésébe, így nemcsak az idegen nyelvi kompetenciájuk fejlődik, hanem az informatikai készségeik is. Erre az informatika tanórákon van lehetőség. Ezért az is fontos, hogy az informatika tanárnak is meglegyen legalább az alap nyelvtudása a megfelelő idegen nyelvből. Ebben is megnyilvánulnak az interdiszciplináris kapcsolatok. A leendő nyelvtanárok így tapasztalhatják, hogy szükségük van nem kevés informatikai tudásra is ahhoz, hogy modern tanárokká váljanak.

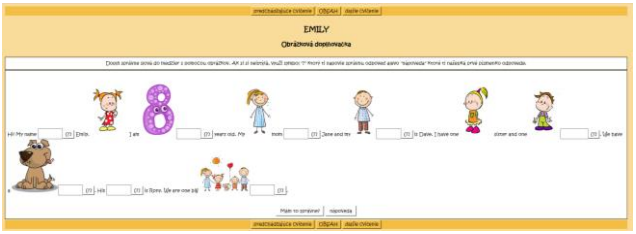
A következő néhány példán szeretnénk bemutatni, hogyan lehet az informatika órán olyan feladatokat megoldani a leendő nyelvtanárokkal, amelyek az idegen nyelv oktatását teszik érdekesebbé. A példákban elsősorban az ellenőrzés lehetőségére mutatunk néhány ötletet, de ezeket nem csak tesztelésre lehet használni, hanem a tanórákra való felkészüléshez is. A feladatokat a Hot Potatoes programcsomag segítségével pedagógus hallgatóinkkal együttműködve hoztuk létre.

A következőkben bemutatott feladattípusok angol és német nyelvből készültek, de kiválóan átültethetők bármely másik nyelv tanítására is.

## 4.1. Feladatok angolnyelvoktatáshoz

Az általános iskolák alsó tagozatos tanulói (4. osztály) számára készült feladat (Készítette: Darina Šramčíková, 1. évf., 2017). A címoldalon (lásd 1. ábra) megjelennek a feladatok, amelyekből tetszőlegesen is választhat a tanuló, vagy sorban oldja meg őket. Az első feladat – EMILY (lásd 2. ábra), amelyben az üres szövegmezőkbe kell beírni a mellettük levő kép angol megnevezését. A feladatot tanulásra és tesztelésre is használhatjuk.

**1. ábra:** A feladat címoldala

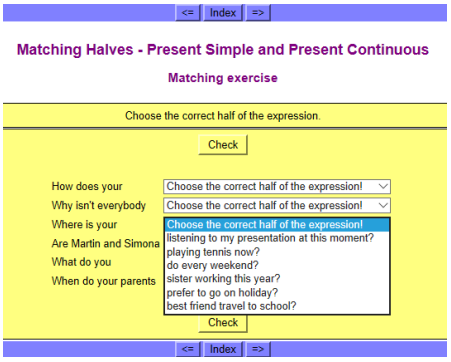



**2. ábra:** EMILY - képes kitöltő

Idősebb korosztály számára készült feladat (lásd 3. ábra), amelyben a megadott szavakat kell kategorizálni típusuk szerint. Ez a feladat egy egyszerű válaszkiválasztó tesztnek felel meg. (Készítette: Tatiana Tisovčíková, 1. évf., 2016).

A következő feladatban (lásd 4. ábra) a diáknak kell kiválasztania a mondat helyes folytatását a felsoroltak közül. (Készítette: Pavol Kadlec, távutas hallgató, 2019).

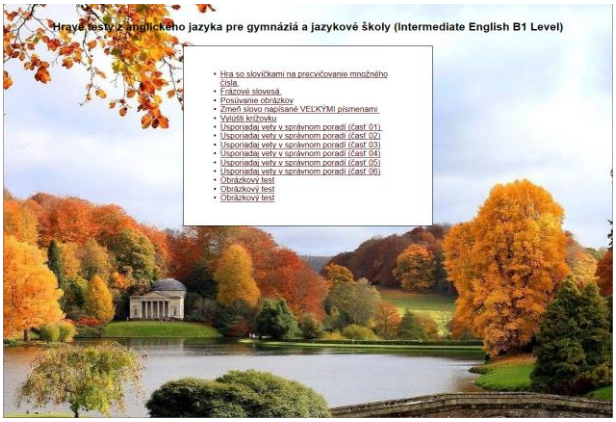

**3. ábra:** Teszt



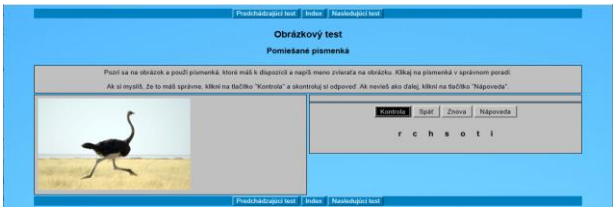

**4. ábra:** Fejezd be a mondatot

Gimnáziumok és nyelviskolák számára készült a következő feladat csomag. Megoldásukhoz más középszintű angoltudásra van szükség. A jó eredmény eléréséhez a diáknak a címoldalon (lásd 5. ábra) látható 14 feladatot kell helyesen megoldania. A megoldások sorrendje választható.

Érdekesek a képes feladatok, ahol a kép alapján a megadott betűkből kell összerakni a képen látható állat nevét (lásd 6. ábra). A feladat megoldása után megjelenik a válasz értékelése és az elért eredmény százalékosan kifejezve (lásd 7. ábra). (Készítette: Roman Langer, távutas hallgató, 2018)




**5. ábra:** A címoldal




**6. ábra:** A megoldásra váró feladat


**7. ábra:** A megoldás az értékeléssel

## 4.2. Feladatok németnyelvoktatáshoz

Hasonló példákat tudunk bemutatni németnyelvoktatáshoz is. Ebben a feladatban az üres szövegmezőkbe kell beírni a melléknévvégződések (lásd 8. ábra). A kérdőjellel segítséget lehet kérni, amelynek tartalmát természetesen a készítő, aki leggyakrabban maga a tanár, határozza meg.

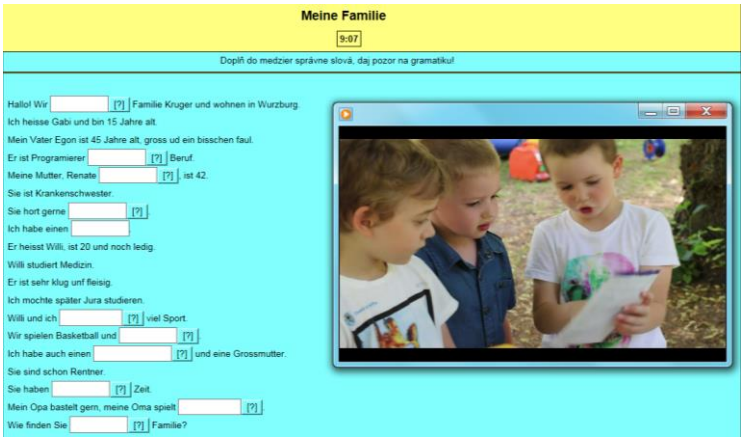

**8. ábra:** Töltsd ki

A diákok nagyon kedvelik a keresztrejtvényes feladatokat. A válaszokat a kérdés mellé a szövegmezőbe kell kitölteni és a program automatikusan megjeleníti a keresztrejtvény celláiba a szavakat

(lásd 9. ábra). A kérdésekbe képeket és hangokat is be lehet illeszteni. (Mindkét feladatot készítette: Lucia Šteinigerová, 2.évf., 2016)


**9. ábra:** Keresztrejtvény

Végül még egy lehetőséget mutatunk be az interaktív elemek alkalmazására. A feladatban a látott, hallott videó felvétel alapján kell a diákoknak kitölteniük a hiányzó szavakat. A feladat nehézségi fokát az is emeli, hogy időre kell teljesíteni (lásd 10. ábra).




**10. ábra:** Interaktív feladat (Készítette: csoport munka)

# 5. Befejezés

Ma már nemcsak az informatika tanárnak, hanem minden pedagógusnak értenie kellene az infokommunikációs eszközök használatához, mert olyan lehetőségeket kínálnak, amelyek nagyban növelik az oktatás hatékonyságát. [8]

Az új kommunikációs technológiák egyre fontosabb szerepet töltenek be az idegen nyelv tanításában is. Lehetővé teszik olyan új módszerek kifejlesztését, amelyek nemcsak a tanórákat individualizálják, hanem gyakorlatiasabbá és realisztikusabbá teszik azokat.

A tanórák egyre digitálisabbá válnak – meggyőződésünk, hogy a nyelvoktatás jövője a modern kommunikációs eszközök és a számítógépes oktatási módszerek és gyakorlatok fokozott használatában rejlik. Ha okosan használjuk ezeket az eszközöket az eredmény biztosan nem fog elmaradni.

A mai tanítás középpontjában a diák a tanítási folyamat aktív tagjaként szerepel, a tanár inkább tervező és menedzser, aki a speciális készségeken kívül infókommunikációs, mediális kompetenciával is rendelkezik. Az egyetemek és az oktatási intézmények feladata, hogy ezt a kompetenciát átadják a jövőbeli tanároknak.

## 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.: Kapitoly z mediálnej výchovy. Nitra: UKF, (2010)
2. B. Wiezorek: *Die wichtigsten Methoden des Fremdsprachunterrichts* (2019) http://www.zsozimek.wodip.opole.pl/german/artykuly/methoden.htm (utoljára megtekintve: 2019.7.1.)
3. K. Tiňáková: *Využívanie komunikačných technológií vo vzdelávaní* (2007) https://www.mtf.stuba.sk/buxus/docs/internetovy_casopis/2007/2/tinakova1.pdf (utoljára megtekintve: 2019.10.20.)
4. I. Pšenáková, O. Hegedűs: *Interactivity in educational materials for language training.* In:: Módszertani közlöny, 7 (2017) 1.Újvidéki Egyetem, Magyar Tannyelvű Tanítóképző Kar, Szabadka (2017) 219-228.
5. I. Pšenáková: *Interactive applications in the work of teachers.* In: XXIXth Didmatech 2016. Budapest: Eötvös Loránd University in Budapest – Faculty of Informatics,(2016) 92-100.
6. Bárdos, J.: Élő nyelvtanítás-történet, Nemzeti Tankönyvkiadó, Budapest (2005)
7. I. Pšenáková, A. Kelemen: *Az interaktív tábla a magyar nyelvtan oktatásában*. In: Ildikó Pšenáková, Orsolya Hegedűs (ed.): Tudomány az oktatásért – Oktatás a tudományért. Nitra: UKF (2011) 201-205.
8. I. Pšenáková, V. Heizlerné Bakonyi, Z. Illés: *Interaktivitás az informatika tanításában*. In: InfoDidact 2018. Budapest: Webdidaktika Alapítvány (2018). 177-185.

# Formatív értékelés SWOT-analízissel

Sarmasági Pál

psarmasagi@inf.elte.hu
ELTE IK

**Absztrakt.** A formatív értékelés még ma is újszerű módszer, ami konferenciákon és cikkekben népszerű téma, de az iskolai gyakorlatban szerény mértékben használják. Szükségszerűségét leggyakrabban a folyamatos visszajelzéssel, a tanuló megfelelő informálásának fontosságával indokolják, az alkalmazott módszerei között leggyakoribb a házi feladatok, beadandó feladatok, diák, vagy diákok órai prezentációja, röpdolgozat, vagy szóbeli felelet értékelése. A folyamatos informálás mellett fontos a diákokat is bevonni az értékelésbe, ez is a formatív értékelés feladata. Az említett célokat segítheti SWOT elemzések készítése, amit marketingre dolgozták ki, de hatékonysága miatt közoktatási intézmények és személyek értékelésére is használják. Mindez a választott szempontrendszer és az értékelés alanyának a függvénye.

**Kulcsszavak:** formatív értékelés, diagnosztikus értékelés, SWOT elemzés, informatikai kompetenciák

## 1. Bevezetés

Informatikai tehetségek felismerésének és kiválasztásának témakörével foglalkozom, középiskolában. A „munkaterületem" így elsősorban tagozatos osztályok, valamint szakköri foglalkozások. Főleg az utóbbi esetben a szummatív értékelés nem annyira lényeges, a diagnosztikus és formatív értékelés viszont kimondottan fontos.

## 2. Formatív értékelés

A formatív értékelés még ma is újszerű, a hazai szakirodalomban alig 20 éves múlttal rendelkező módszer. Napjainkban természetesen mind konferenciákon, mind cikkekben az egyik legnépszerűbb téma, mind ezek ellenére a gyakorlatban még csak nagyon szerény mértékben használják.

A formatív értékelés szükségszerűségét leggyakrabban a folyamatos visszajelzéssel, a tanuló megfelelő informálásának fontosságával indokolják. Ez nem is kérdéses, hiszen ez alapján tudnak az egyes diákok jobban gazdálkodni idejükkel, tanulásra fordított energiájukkal, és a tanárok is felismerhetik, mely témakörökkel kell többet foglalkozni, vagy újabb metaforákkal tanítani a jobb megértés és megtanulás érdekében.[1]

A formatív értékelés ismert, gyakorolt módszerei között leggyakoribb a házi feladatok, vagy beadandó feladatok, esetleg diák, vagy diákok órai prezentációja, röpdolgozat, vagy szóbeli felelet értékelése.

Informatika témában különösen jól használhatók az IKT eszközök, és egyre több pedagógus használja ezeket formatív értékelés támogatására. Ilyen a Kahoot, a Redmenta, a Learningapps.com és még folytathatnám a sort.[2]

Van azonban egy másik szempontrendszer a formatív és diagnosztikus értékelés mentén. A folyamatos informálás mellett fontos a diákokat is bevonni az értékelésbe. Nem csupán a relációképzés matematikai kompetencia erősítésére, hanem az önértékelési képesség fejlesztésére is. Az önképzavar ugyanis nem csak pszichiátriai kategória, középiskolás korban életkori sajátosság is, amit csoport és projektmunkákkal, diákok egymás kölcsönös értékelésével, majd önmaguk értékelésével kezelhető. Ez is a formatív értékelés feladata, még ha kevesebbet is beszélünk róla.

## 3. SWOT elemzés

A címben szereplő SWOT analízis közismert, az elmúlt két évtizedben a Magyarországi közoktatásban is elterjedt, elsősorban intézmények, kisebb részben pedig tantervek, tanmenetek értékelésére.

Nézzük tudományosan a SWOT elemzést. A Humán- és Társadalomtudományokon belül a Gazdaság- és Jogtudományok része a Marketing, ezen belül a stratégiai marketing eszközének fejlesztették ki az 1950-es években az Egyesült Államokban. A marketing történetének ezt az időszakát a vevőorientált korszaknak is nevezzük (1940-1980), és talán a SWOT elemzés is hozzájárult, hogy 1980-tól napjainkig egy új korszaka van a marketingnek, ez pedig a Társadalomorientált korszak. Érdekes, hogy annak idején elfelejtették publikálni, így tudománytörténészek feladata tisztázni, hogy Albert Humphrey a Stanford Egyetemen a 60-as években, vagy George Albert Smith Jr. és C. Roland Christensen a Harvard Egyetemen az 50-es években dolgozta ki. [3] Az utóbbiak munkáját folytatta Kenneth Andrews szintén a Harvard Business School-ban, és terjesztették el a SWOT elemzés módszerét. Jelen állás szerint ők hárman, illetve hallgatóik dolgozták ki a módszert, amit a nagyobb publicitással rendelkező, és azt kutatásában felhasználó Humphrey-nek tulajdonítanak néhányan.[4]

Az elemzés célja, hogy egy-egy terméket megfelelően pozícionáljanak a piacon. A SWOT maga egy betűszó, a Strengths mint erősségek, a Weaknesses mint gyengeségek az Opportunities mint lehetőségek és a Threats mint veszélyek szavak kezdőbetűiből. Az erősségek és gyengeségek belső tényezők, melyek a termék fizikai és pénzügyi jellemzői (ez utóbbi a cég pénzügyére is vonatkozhat), illetve a háttérben lévő emberi erőforrásokra utalnak. A lehetőségek és fenyegetése a külső tényezők, mint a gazdasági és piaci trendek. Az elemzés végén javaslatot, célt kell megfogalmazni, illetve megtalálni a cél eléréséhez szükséges stratégiát.

Az 1960-as években kezdték el használni, a módszer azóta sokat bizonyított, így egyre több területen kezdték alkalmazni. A belső tényezők között eleve szerepel az emberi erőforrás, emiatt a munkaerő piaci trénerek, coach-ok elkezdték személyekre is alkalmazni. Emberi erőforrás gazdálkodás területén a munkavállaló is egy termék, amit el kell tudni adni. Az egyéni marketingterv a személyes fejlesztési terv stratégiáját vázolja.[5]

A nemzetközi szakirodalomban is már fellelhetők olyan példák, ahol a diákok valamilyen szempontrendszer szerinti SWOT elemzésével vizsgálnak fontos kérdést. Thaiföldön orvostanhallgatók körében vizsgálták [6], mennyire lesznek jó orvosok a hallgatók? A kutatás elsődleges célja az volt, hogy feltárják azokat a veszélyeket, amelyek következtében az egyetemi képzést követően mégsem orvosi pályán helyezkednek el a hallgatók. Az eredmények ismeretében módosították a képzést, hogy minél többen vállalják az orvosi hivatást.

## 4. SWOT elemzés az iskolai értékelésben

Visszatérve ismét a munkaterületre, a középiskolai tagozatos osztályok és szakkörök tanulóihoz, a tehetségek felismerésére és gondozására hasznos a diagnosztikus és formatív értékelés, amit jól támogathat a SWOT elemzés.

Milyen szempontrendszer szerint tudjuk vizsgálni a belső tényezőket, mi szerint elemezzük a diákok erősségeit és gyengeségeit? Már évtizedes tapasztalattal rendelkezünk a kompetencia mérésekkel, az EU által meghatározott kompetencia területek jó kiindulási pontot jelentenek. Informatika szempontjából azért is jó a kompetenciákból elindulni, mert egy fastruktúrában rendezett szempontrendszeren keresztül a különböző szintek feltárásával a különböző diákoknál más-más területeket vizsgálhatunk a diák érdeklődése, erőssége vagy gyengesége alapján. Mindenki számára ismert a 9 kulcs kompetencia: [7]

- anyanyelvi (beszéd, olvasás, írás),

- idegen nyelvi (beszéd, olvasás, írás),
- matematikai kompetenciák,
- természet-, és műszaki tudományi kompetenciák,
- digitális kompetencia (számítógép- és Internet-használat),
- szociális és állampolgári kompetenciák,
- kezdeményezőkészség és vállalkozói kompetenciák,
- kulturális tudatosság és kifejezőkészség kompetenciái,
- a tanulási képesség kompetenciái

Informatikai tehetséggondozás szempontjából nem elhanyagolhatók ezen általános kompetencia területek a legfelsőbb szinten. Sok informatikai tehetség nehezen fejezi ki magát. Korosztályomból is sok ilyen embert ismerek. Középiskolában ennek hiányossága – ami komoly gyengeség – még pótolható. A tehetséges diákokat nemzetközi versenyekre is benevezzük, szükséges az idegennyelv ismeret. Nagy veszteség, ha nyelvismeret hiánya miatt nem nevezhető a legjobb diákunk. Tehetséggondozás területen erre is figyelni kell. A matematikai, természet- és műszaki tudományi, valamint a digitális kompetenciák részben magukért beszélnek, e területeket egy következő lépésben egy, vagy több szinttel mélyebben is vizsgáljuk. A szociális és állampolgári kompetenciák a tehetség kibontakozás szempontjából fontos és vizsgálandó kérdés, a kezdeményező készség, kulturális tudatosság és a tanulási képesség kompetencia területei is mind-mind feltárandó és a tanulmányok során visszatérően vizsgálandó területek.

Második szinten – fókuszálva az informatika tantárgyra – vizsgáljuk a „digitális", informatika tanárok által inkább Informatikai kompetenciának nevezett terület komponenseit:

- algoritmikus gondolkodás
- adat modell
- a valós világ modellezése (interdiszciplináris látás)
- probléma megoldás
- kommunikációs képesség
- alkalmazói képesség
- csoportmunka, együttműködő képesség
- alkotó képesség
- információs tájékozódási és tájékoztatási képesség
- rendszerszintű gondolkodás

Ezen a szinten már attól függően, hogy az egyes komponensek erősségét vagy gyengeségét képezik egy-egy diáknak, már differenciálni kell. A tehetséggondozás során már nem feltétlen elvárás, hogy minden diák minden területen egyformán eredményes legyen. Mindez nem azt jelenti, hogy csak a sikerrel kecsegtető területre kell fókuszálni, de sokkal differenciáltabb egyéni célokat kell és lehet megfogalmazni, mint sem egy általános középiskolai osztályban, csoportban. Ennek megfelelően a következő szinten már van olyan diák, akinél az alkalmazói képesség kompetencia részleteivel célszerű tovább lépni, míg más diáknál az algoritmikus gondolkodás szintjeit kell vizsgálni, mik az erősségek, és mik a gyengeségek? Példámban ez utóbbi komponenst kiválasztva a következő harmadik szinten az alábbi szempontokat vizsgálhatjuk:

- Algoritmus (tevékenységsorozat) felismerése, megértése
- Algoritmus (tevékenységsorozat) végrehajtása
- Algoritmus (tevékenységsorozat) elemzése
- Algoritmus (tevékenységsorozat) alkotása
- Algoritmus (tevékenységsorozat) megvalósítása
- Algoritmus (tevékenységsorozat) módosítása, átalakítása
- Komplex algoritmus (tevékenységsorozat) tervezése

Egy jó algoritmikus gondolkodású, programozási rutinnal is rendelkező diák esetén a versenyekre való felkészítés céljából az „Algoritmus (tevékenységsorozat) megvalósítása" területet bonthatjuk tovább. Ezen mélyebb szinteken már nagyon egyéni szempontok szerint kell vizsgálódni. Lehetséges további vizsgálati szint, hogy milyen programozási környezetekben mozog otthonosan?

- C/C++
- C#/Java
- Python

A C/C++ környezeten belül a gyakrabban előforduló programozási feladatok ismerete:

- adattípusok és adatszerkezetek ismerete
- állománykezelés
- matematikai függvények ismerete
- rekurzív algoritmusok megvalósítása
- stb.

A szintek száma diákonként és témakörönként épp úgy változhat, mint az egyes szinteken lévő értékelési szempontok. Maguk ezek az értékelési szempontok is segítenek megfogalmazni a célokat, mind a diák, mind a tanár számára.

Az elemzés azonban nem teljes, a fenti vizsgálat csupán az erősségek és gyengeségek feltárását, tehát a belső tényezők áttekintését teljesítette. A külső tényezők azonban egy személy elemzésekor szorosan kapcsolódnak a belső tényezőkhöz. A lehetőségek esetünkben az erősségekhez kapcsolódnak. Amilyen területen erős a diák, azon érdemes versenyre nevezni, amely területen pedig gyengeség tapasztalható, azt át kel beszélni. Ha a közösen megfogalmazott célokat hátráltatja, az veszélyként értékelendő, azon a területen még dolgozni kell. Amennyiben olyan területet érint, amely nem része sem a kötelező tananyagnak, sem a további céloknak, legalábbis az adott időszakban, akkor az ilyen esetben az adott gyengeség nem jelent veszélyt, az a vizsgálat során elhanyagolható.

A SWOT elemzések irodalmában az évtizedes gyakorlat során kialakult stratégiai modellek is ismertek, amelyek szintén átültethetők az iskolai értékelésbe, különös tekintettel a visszajelzési, szabályozási funkcióval bíró formatív értékelésben. Ezek a kombinált SWOT mátrixban felírt betűpárokból természetesen adódó stratégia lehetőségek.

| | | **Belső tényezők** | |
|---|---|---|---|
| | | **Erősségek** | **Gyengeségek** |
| **Külső tényezők** | **Lehetőségek** | S-O stratégiák | W-O stratégiák |
| | **Veszélyek** | S-T stratégiák | W-T stratégiák |

- S-O stratégia: Offenzív stratégia – Lehetőségek kihasználása az erősségek révén
- W-O stratégia: Válságorientált stratégia – Gyengeségek leküzdése a lehetőségek kiaknázásával
- S-T stratégia: Diverzifikált stratégia – Védelem a fenyegetettség ellen az erősségek használatával
- W-T stratégia: Defenzív stratégia – Olyan stratégia kell, ami megóv a gyengeséget célzó fenyegetéstől

Az iskolai értékelés során elsődlegesen a belső tényezők feltárására van módunk, a diákok erősségei és gyengeségei alapján tudjuk azonosítani a klasszikus SWOT elemzésben feltárandó külső tényezőket, a lehetőségeket és a veszélyeket.

A megfelelő stratégia kiválasztása egy újabb lehetőség a diákok értékelésbe való bevonására, fejlesztve ezáltal a tanulók elemzési képességüket. Ugyanakkor a tanár pedagógiai érzékére, tapasztalatára is szükség van, hogy a kiválasztott stratégia hozzájáruljon a tanuló fejlesztéséhez. Egy általánosságban gyengébb tanuló esetén, aki az elemzés során csak azt veszi észre, hogy mennyi gyengesége van nehéz kérdés eldönteni, hogy defenzív stratégia alkalmazásával minél több terület párhuzamos fejlesztésébe kezdünk, vagy adott esetben a néhány erősségére fókuszálva offenzív stratégiával a lehetőségek irányába indulunk el? Ez az adott diák személyiségétől is függ.

Ugyanez a kérdés természetesen az átlagos tanulók, akiknél nagyjából egyensúlyban vannak az erősségek és a gyengeségek, illetve a kiemelt képességű tanulók esetén is nem egyértelműen megválaszolható.

Az értékelés formatív jellegét erősíti azonban, hogy egy stratégia alkalmazását követő újabb SWOT elemzés alapján hasonló eredmények mellett is egy másik stratégiát próbálunk alkalmazni. Nem csak az elemzés értékelő részébe, hanem a stratégia kiválasztásába is bevonva a diákokat a tanulási cél pontos megfogalmazásába.

## 5. Konkrét példa egy SWOT elemzés alapú értékelésre

Nézzünk egy konkrét példát a megvalósításra a fenti szempontrendszerek alapján. Egy 12. osztályos tagozatos diákkal közösen értékeltük tudását az egyes területek alapján.

| Szempont | Diák | Tanár |
|---|---|---|
| *Kulcskompetenciák* | | |
| anyanyelvi (beszéd, olvasás, írás) | + | + |
| idegen nyelvi (beszéd, olvasás, írás) | + | + |
| matematikai kompetenciák | + | + |
| természet-, és műszaki tudományi kompetenciák | + | + |
| **digitális kompetencia (számítógép- és Internet-használat)** | + | + |
| szociális és állampolgári kompetenciák | + | + |
| kezdeményezőkészség és vállalkozói kompetenciák | + | + |
| kulturális tudatosság és kifejezőkészség kompetenciái | + | + |
| a tanulási képesség kompetenciái | + | + |
| *Informatikai kompetenciák komponensei* | | |
| **algoritmikus gondolkodás** | + | + |
| adat modell | + | + |
| a valós világ modellezése (interdiszciplináris látás) | + | + |
| probléma megoldás | + | + |
| kommunikációs képesség | + | + |
| alkalmazói képesség | + | + |
| csoportmunka, együttműködő képesség | + | + |
| alkotó képesség | + | + |
| információs tájékozódási és tájékoztatási képesség | + | + |
| rendszerszintű gondolkodás | + | + |
| *Adott komponens részletezése* | | |
| Algoritmus (tevékenységsorozat) felismerése, megértése | + | + |
| Algoritmus (tevékenységsorozat) végrehajtása | + | + |
| Algoritmus (tevékenységsorozat) elemzése | + | + |
| Algoritmus (tevékenységsorozat) alkotása | + | – |
| **Algoritmus (tevékenységsorozat) megvalósítása** | + | – |
| Algoritmus (tevékenységsorozat) módosítása, átalakítása | + | – |
| Komplex algoritmus (tevékenységsorozat) tervezése | – | – |
| *Program nyelvek ismerete* | | |
| **C/C++** | + | + |
| C#/Java | + | – |
| Python | – | – |
| *Adott nyelv ismerete* | | |
| adattípusok és adatszerkezetek ismerete | + | + |
| állománykezelés | + | + |
| matematikai függvények ismerete | – | – |
| rekurzív algoritmusok megvalósítása | – | – |

A fastruktúrába szervezett szempontrendszerek mélyén már eltért egymástól a diák önértékelése és az általam készített tanári értékelés. A diagnosztikus értékelési szakaszban ez természetes. Amennyiben egy ilyen jellegű elemzést egy tanév során négy alkalommal elvégzünk, a tanuló önértékelési képessége sokat fejlődik, és a különbségek eltűnnek, de legalább minimalizálódnak.

A belső tényezők elemzését követően megkezdődhet a stratégiaalkotás. A fent bemutatott példában az aláhúzott sorok a következő időszak célkitűzései között szerepelnek, a szürke háttérrel jelzett sorok a jelen időszakban elhanyagolhatók, ami természetesen egy időszakra vonatkozik, a későbbi tanulmányi céloknak részévé válhat.

| | | |
|---|---|---|
| *Adott komponens részletezése* | | |
| Algoritmus (tevékenységsorozat) felismerése, megértése | + | + |
| Algoritmus (tevékenységsorozat) végrehajtása | + | + |
| Algoritmus (tevékenységsorozat) elemzése | + | + |
| Algoritmus (tevékenységsorozat) alkotása | + | – |
| **Algoritmus (tevékenységsorozat) megvalósítása** | + | – |
| Algoritmus (tevékenységsorozat) módosítása, átalakítása | + | – |
| Komplex algoritmus (tevékenységsorozat) tervezése | – | – |
| *Program nyelvek ismerete* | | |
| **C/C++** | + | + |
| C#/Java | + | – |
| Python | – | – |
| *Adott nyelv ismerete* | | |
| adattípusok és adatszerkezetek ismerete | + | + |
| állománykezelés | + | + |
| matematikai függvények ismerete | – | – |
| rekurzív algoritmusok megvalósítása | – | – |

Az adott diáknak az elemzés szerint jó lehetőségei vannak. Általános műveltsége és intelligenciája kiemelkedő, szeret informatikával foglalkozni, és erős alapokkal bír. Lehetősége van indulni versenyeken, azonban a gyengeségei még veszélyeket jelentenek. Az önálló algoritmusalkotásban, és önálló programozásban kell fejlődnie, hasznos lehet a C# környezetben is nagyobb gyakorlatot szereznie, valamint a C++ nyelv matematikai függvényeit, és rekurziós lehetőségeit is meg kell tanulnia.

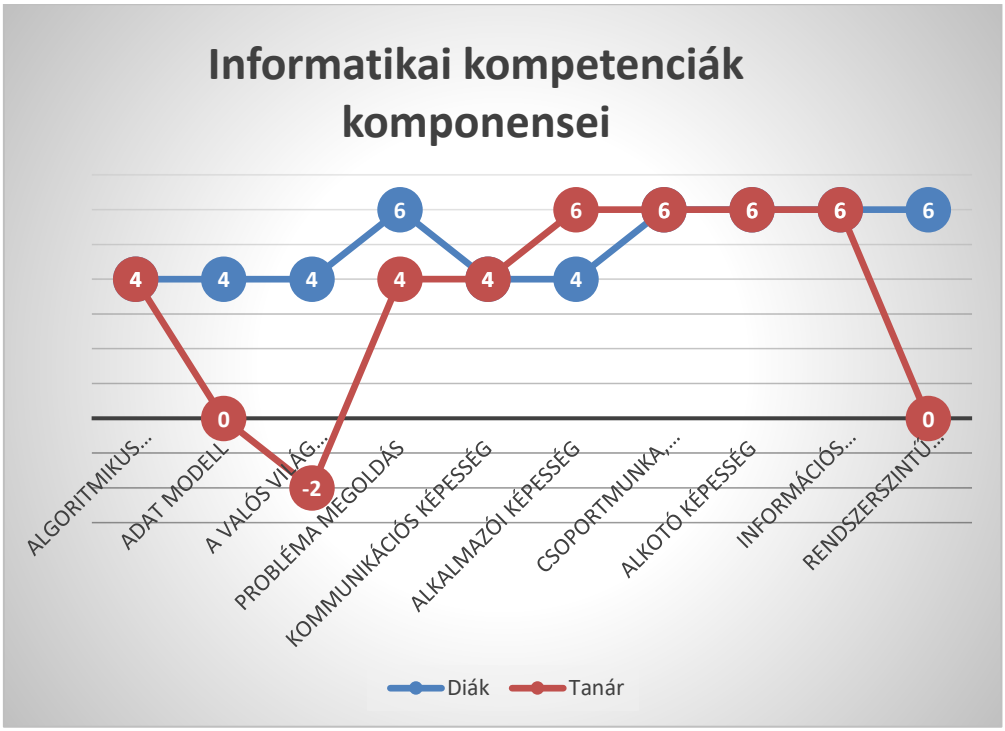
Esetében egy „W-T” stratégia javasolt, meg kell óvni a gyengeségeit célzó veszélyektől. Iskolai környezetre fordítva ez nem más, mint a következő hónapok tanulási céljainak megfogalmazása. Az egyértelmű célok megfogalmazása nagyon fontos mind a diák, mind az őt tanító tanár szempontjából. A tanár a megfelelő tananyag leadása mellett olyan gyakorló feladatokat ad a diáknak, melyek segítségével a gyengeségként feltárt területet megtanulhatja és begyakorolhatja. Természetesen a siker eléréséhez a diák megfelelő közreműködése is szükséges, a jól behatárolt cél a szükséges hozzáállást elősegíti, támogatja.

## 6. Az előbbi példa kiterjesztése

Az előzőekben bemutatott SWOT elemzést a csoport több tagjával is elkészítettük. Az eredmények többé-kevésbé hasonlóak voltak, amikor eltérés mutatkozott a tanár és diák értékelésében, a legtöbb esetben a diákok értékelték többre magukat, és csak néhány esetben fordult elő, hogy tanárként erősségnek láttam egy területet, amit a diák gyengeségnek értékelt tudásában.

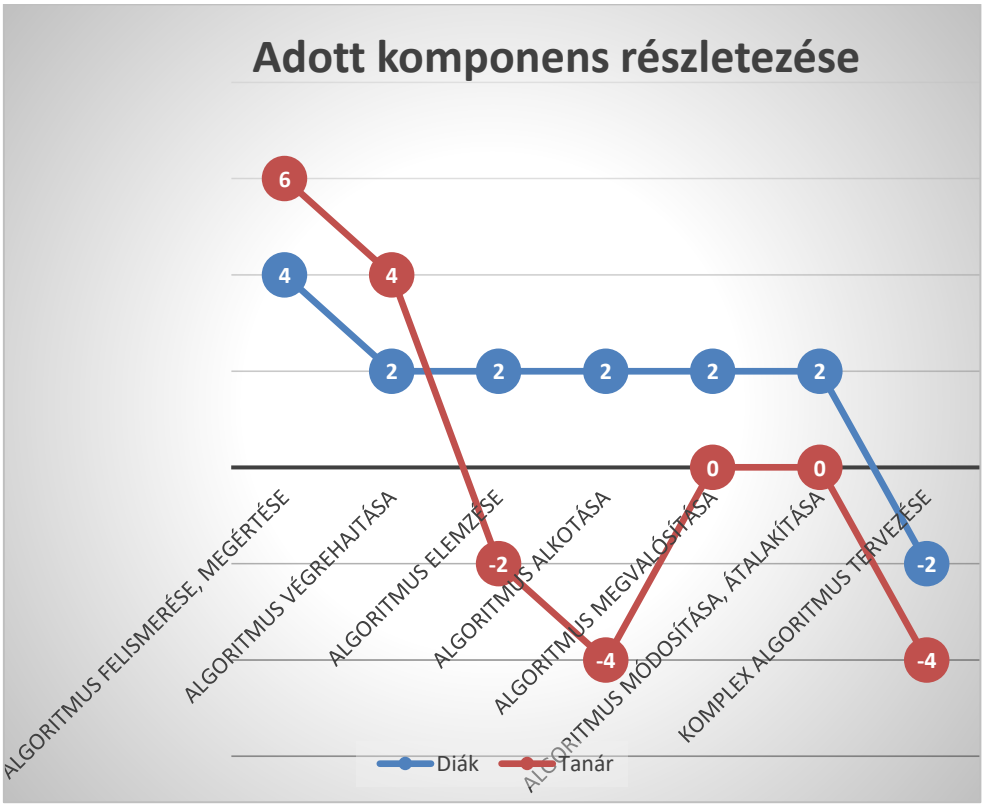
Az egyes diákokra elkészült elemzéseket a szempontrendszerek szerint csoportosítva értékeltem ki. A különböző szinteken belül szempontonként vizsgáltam a csoport erősségeit és gyengeségeit. A kimutatásokban az erősségek +1, a gyengeségek –1 értékkel szerepelnek, a vizsgálatban 6 diák vett részt. Ebből az elemzésből a kulcs-kompetenciák szempontjait kihagytam, és inkább az informatikai kompetenciák területére és annak komponenseire vonatkozóan készítettem ábrákat.

A nagyobb minta vizsgálata a tanár számára ad igazán hasznos visszajelzést egy-egy terület megértésével vagy tanulási nehézségeivel kapcsolatban. Mind a jelentősebb eltérések, mind a gyengeségek tartományban megfigyelhető korrelációk figyelmeztetik a tanárt, hogy egyes tananyagokat át kell ismételni, másokat jobban be kell gyakorolni. A konkrét vizsgálat során az elemzés hitelességének tesztelésére olyan témakör is bekerült a szempontrendszerek közé, amelyről a foglalkozások során nem volt szó.



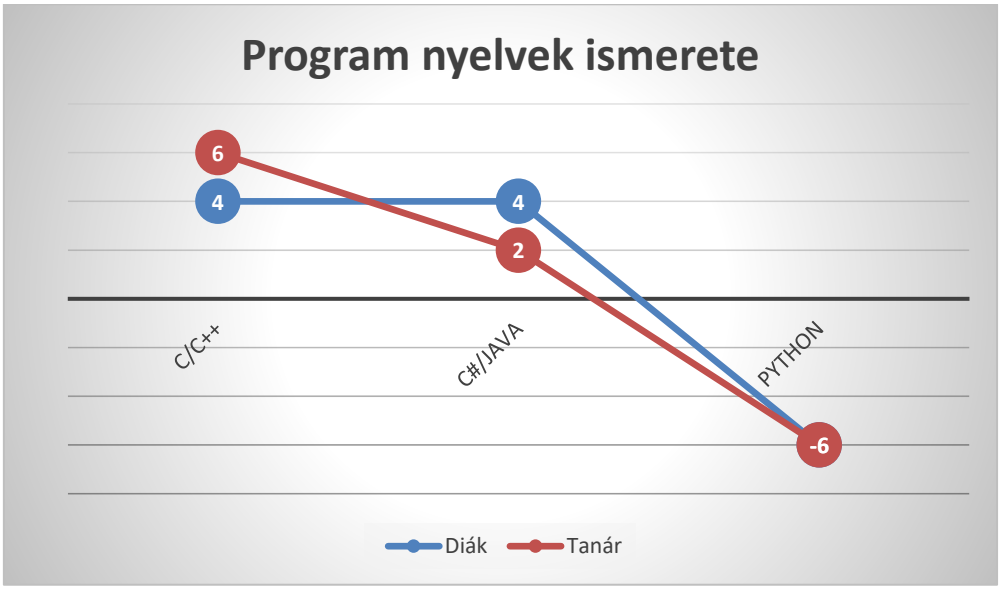

Az Informatikai kompetenciák komponensei között 3 esetben van jelentős eltérés a diákok és a tanári értékelés között. Az adott csoport esetében az eltérés egyik oka az értékelés szubjektivitása, mivel tanárként több éve tanítom a vizsgálatban részt vett diákokat, ismerem órai aktivitásukat, gondolkodásukat. Másrészről azonban az adatmodellezés, a valós világ modellezése, valamint a rendszerszintű gondolkodás olyan komponensek, amelyek elsajátítása hosszabb időt, és több gyakorlást igényel, miközben a diákok egy-egy sikerélményt követően hajlamosak egyszerűbbnek látni az adott területet.

Az elemzés következő szintjén részletezett algoritmikus gondolkodás komponens egészét vizsgálva azonban nem volt eltérés a diákok és a saját tanári értékelésemben, 5:1 arányban korrelált a két fél értékelése az erősségek és gyengeségek viszonyában.



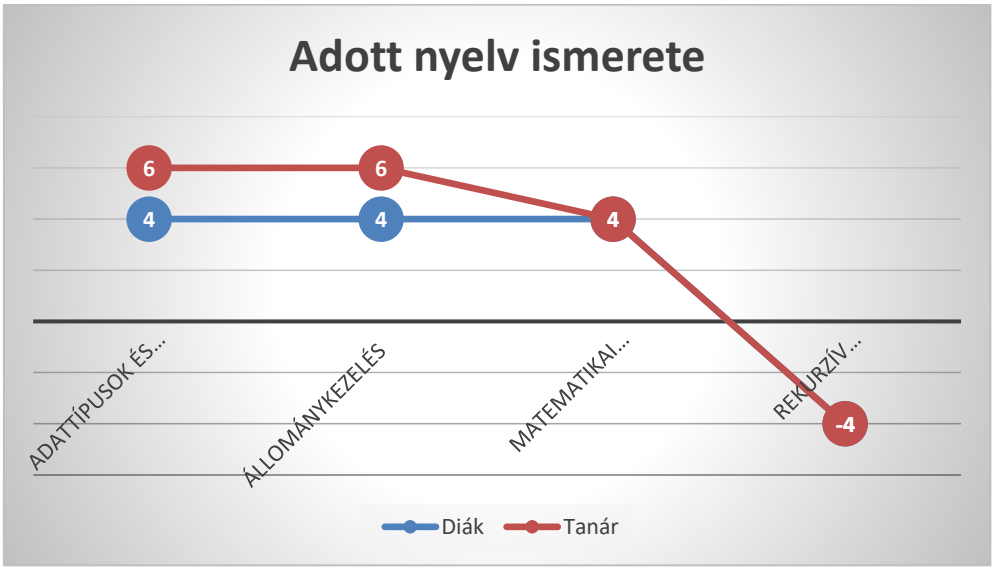

A részletes adatok között azonban ismét jelentős eltérések figyelhetők meg. Az algoritmus elemzése, még inkább az algoritmus alkotása a diákok értékelése alapján egyszerűbb feladatnak látszik, amit meglátásuk szerint készségszinten elsajátítottak. Értékelésemben azonban az órai aktivitást, és a beadandó feladatok minőségét is figyelembe vettem, így ezen területeket még fejlesztendő gyengeségnek értékeltem.

A komplex algoritmus tervezését illetően nincs jelentős eltérés az értékelésekben. Az a tény azonban, hogy az értékelés szerint ez a terület a gyengeségek negatív tartományába került mind a diák, mind a tanár számára egyértelműen jelzi, hogy az adott terület további fejlesztésre szorul.




A programozási nyelvek ismeretének vizsgálatához a Python nyelv részben diagnosztikus okokból, részben a teszt hitelessége miatt került. A csoport az iskolai foglalkozások során még nem tanult erről a nyelvről, és a vizsgálat alapján a diákok más forrásból sem rendelkeznek használható ismerettel.

Az elemzés mélyebb szintjein, a tananyag felépítésének alap összetevőit vizsgálva már jellemzőbb, hogy tanárként kiforrottabbnak értékelem az egy-egy terület ismeretét, mint a diákok.




A jelen elemzés legmélyebb szintjén a leggyakrabban használt programozási nyelv ismeretét vizsgáltuk. Ezen a szinten nagyon jól korrelál a diákok és a saját tanári értékelésem. A rekurzív algoritmusokkal még szintén nem foglalkozott a csoport, azonban az egyik diák proaktív érdeklődésének köszönhetően jól ismeri és használja a rekurzív algoritmusokat is.

## 7. Összegzés

Néhány mondatban összefoglalom, hogy véleményem szerint miért alkalmas a SWOT elemzés az iskolai értékelésre, a diagnosztikus és formatív értékelés mely szempontjait, a korszerű oktatás mely elvárásait tudja támogatni, kielégíteni az iskolai értékeléssel kapcsolatban. Egy táblázatban összegzem a SWOT elemzéssel kapcsolatos fontosabb állításokat, mellettük megjelenítve az iskolai értékeléssel kapcsolatos, az adott állítás által lefedett, kiszolgált elvárást, követelményt.

| SWOT elemzés jellemzője | Iskolai értékeléssel kapcsolatos elvárás |
|---|---|
| Összegzi a különböző forrásokból származó információkat | Iskolai értékelés is adatgyűjtésen alapul. Hagyományosan felelés, dolgozat alapján, de az adott teljesítmény mellett fontos információ a diák hozzáállása, fejlődése, illetve elmaradása is. |
| A SWOT elemzés nem azt mondja meg, hogy mit kell elemezni, hanem azt, hogyan rendezzük,<br>hogyan értékeljük a rendelkezésre álló információkat | Az iskolai értékelésben alkalmazott kritérium és normaorientált értékeléshez is alkalmazható |
| Helyzetelemzés | A diagnosztikus értékelés a tanulók aktuális felkészültségét méri |
| Stratégia alkotás | Diagnosztikus és formatív értékelés során fontos a rövid és hosszú távú célok megfogalmazása |
| SWOT elemzés készíthető más személyről és önmagunkról is | Az iskolai értékelés a tanár és a tanuló közösség közös ügye. Fontos a diákok bevonása az értékelésbe. |

Az iskolai értékelésben a SWOT elemzés tehát alkalmas diagnosztikus és formatív értékelést támogató eszköznek. A kiválasztott szempontrendszer alapján feltárhatók a tanulók erősségei és gyengesége, melyek alapján lehetőségeket és veszélyeket azonosíthatunk. A további célok, stratégia megfogalmazásának iránya a lehetőségek kihasználása és a veszélyek elkerülése. Nem fordítva! Ehhez fontos szerep jut a pedagógusnak.

A tehetség felismerés és nevelés során a szummatív értékelés kevésbé fontos, ilyen közegben a tanulmányi versenyek töltik be annak szerepét. Szakköri környezetben a versenyek előtt, de gyakran utána is a formatív értékelés nagyon fontos. Néhány diák képes „túlnyerni" magát, őket meg kell óvni a kiégéstől, míg azokat, akik semmit nem nyernek, nagyon kell a további bíztatás.

## 8. További lehetőségek

A tanári attitűd és a diákok ismerete alapján egyéb, általánosabb területek is vizsgálhatók SWOT elemzéssel.

- Hasonlóan a SWOT analízishez, az emberi erőforrás gazdálkodás használ egy egyszerűsített „személyiség tesztet", melyben különböző kérdések segítségével 4 csoportba sorolják az embe-

reket: Domináns (D), befolyásoló (I), stabil (S) és szabálykövető (C). A módszert Carl Gustav Jung introvertált és extrovertált csoportosításának kibővítésével William Moulton Marston alakította ki 1928-ban.

- A tanulás, felnőtté válási folyamat egy nagyon fontos összetevője az alkalmazkodási képesség. Ezt nagyon ritkán vizsgálják, pedig ez is egy fontos szempont, a jó alkalmazkodó képesség előny, erősség, míg a gyengébb hátrány, gyengeség.
- Informatikai tehetségek kutatása, felismerése során a különböző tehetség modellek szempontrendszerei is vizsgálhatók. Akár a Renzulli féle háromkörös modell alapján, akár a Czeizel Endre féle 2*4+1 szempontrendszer alapján. Mind két modell alapján ahhoz, hogy valaki kihasználja, és a társadalom szintjén is kamatoztatni tudja tehetségét, a lehető legtöbb szempontból erősnek kell lennie.

## Irodalom


1. Radnóti Katalin: *Milyen oktatási és értékelési módszereket alkalmaznak a pedagógusok?* In: Kerber Zoltán (Szerk.) Hidak a tantárgyak között. ISBN: 9636825726 (2005) Országos Közoktatási Intézet, Budapest (131-167)
2. Szerk: Károly Krisztina, Homonnay Zoltán: *Mérési és értékelési módszerek az oktatásban és a pedagógusképzésben*; ISSN: 2416-2957 (2017) ELTE Eötvös Kiadó
3. Tim Friesner: *History of swot analysis*; (2011) https://www.researchgate.net/publication/288958760_History_of_swot_analysis (utoljára megtekintve: 2019.10.19)
4. Emet Gürel, Merba Tat: *SWOT ANALYSIS: A THEORETICAL REVIEW*; ISSN: 1307-9581 (2017) The Journal of International Social Research http://www.sosyalarastirmalar.com/cilt10/sayi51_pdf/6iksisat_kamu_isletme/gurel_emet.pdf (utoljára megtekintve: 2019.10.21)
5. Lon Addams, Anthony T. Allred: *THE FIRST STEP IN PROACTIVELY MANAGING STUDENTS' CAREERS: TEACHING SELF-SWOT ANALYSIS*; 17 ISSN: 1528-2643 (2013), ACADEMY OF EDUCATIONAL LEADERSHIP JOURNAL
6. Thira Woratanarat, Patarawan Woratanarat: *Assessment of Prospective Physician Characteristics by SWOT Analysis*; (2012), Malays J Med Science https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3436489/ (utoljára megtekintve: 2019.10.28)
7. Zsakó László, Szlávi Péter: *Informatikai kompetenciák: Algoritmikus gondolkodás;* DOI: 10.13140 (2011) ELTE IK

# LEGO robotok felhasználási lehetőségei az oktatásban

Solymos Dóra

solymi007@inf.elte.hu
ELTE IK

**Absztrakt.** Készítünk egy programot, kiírja egy nagy fekete ablakban, hogy 4. Most akkor ez jó vagy sem? Miért is kell nekem ez? (Merültek fel bennem is anno a kérdések.) A programozás oktatása, bármilyen korosztályban is kezdjük el, nehéz, ugyanakkor fejleszti a logikát, a problémamegoldó képességet, a kreativitást, és hozzájárulhat az informatikai gondolkodás egyéb részterületeinek fejlesztéséhez is. A tanításban támogatást nyújtanak az elmúlt néhány évben megjelenő eszközök, melyek segítségével játékos formában sajátíthatják el a diákok a programozás alapjait. Ebben a cikkben bemutatom a LEGO Mindstorms EV3 felhasználási lehetőségeit az általános iskola felsős korosztályában, valamint beszámolok a nemrég indított szakkörök és táborok tapasztalatairól.

**Kulcsszavak:** LEGO Mindstorms EV3, Micro:bit, STEM, szenzorok, játékos programozás, programozástanítás

## 1. Bevezetés

A robotika és a programozás oktatása egyre nagyobb teret nyer hazánkban is. Ez részben köszönhető annak, hogy egyre több kézzel fogható és nem mellesleg olcsó(bb) eszköz került piacra, melyekhez egy-egy gyerekbarát nyelv is társult. Továbbá fontos megemlíteni a programozást népszerűsítő workshopokat, magániskolákat és a növekvő igényt a munkaerőpiacon a programozókra.

Mindezeket figyelembevéve, célszerű már az iskolában is elkezdeni a programozás tanítását, ugyanakkor a közoktatásban a programozás oktatása több nehézségbe is ütközik. Ezek között szerepel az alacsony óraszám, ami kis probléma a gyerekek érdeklődésének felkeltése és megtartása mellett. Hiszen, ha szeretnénk szakkört hirdetni, hogy valami érdemleges dolgot is csináljunk, akkor az valószínűleg (késő) délutánra esik, amikor a gyerekek már nem túl lelkesek. Azt se felejtsük el, hogy mennyi felkészülés kell a pedagógus számára egy ilyen óra megtartására. Hiszen, lehet, hogy a régi, megkopott tudást fel kell eleveníteni, gyakorolni, majd tervezni és újra tervezni, ami valljuk be hosszú folyamat. Ha esetleg valamilyen eszköz/robot segítségével szeretnénk megmutatni a programozás csodás világát diákjainknak, akkor még több problémába ütközünk. Kezdve az eszközök beszerzésével, használatával, karban-tartásával és közoktatásba való átültetésével. Ebben segítséget nyújthat, hogy már több magyar segéd-anyag elérhető a robot programozás tanításához. [1] [2]

### 1.1. Robotika az oktatásban

Külföldi, esetleg már bevált módszer után kutatva, nem kell nagyon messzire mennünk. A BBC Micro:bit 2015-ben jelent meg az Egyesült Királyságban, és egy év alatt egymillió darabot gyártottak belőle, hogy 2016-ban minden hét éves kisiskolásnak adjanak az országban. A diákok tanórákon használták az eszközöket tanáraik segítségével és tapasztalataik azt mutatják, hogy az eszköz megváltoztatta mind a diákok, mind a tanárok hozzáállását a programozáshoz. A megkérdezett diákok 90 %-a egyetértett azzal az állítással, hogy a BBC Micro:bit megmutatta nekik, hogy bárki meg tud tanulni programozni, míg a tanárok 85%-a nyilatkozott úgy, hogy az eszköz segítségével, az informatika még élvezetesebb lett diákjai számára. [3]

A brit sikereken fellelkesülve, Horvátországban közösségi finanszírozásból ezer általános és középiskolásnak, valamint 37 könyvtárnak adományoztak Micro:bit-et. Szingapúr 2017 áprilisában jelentette be, hogy 100.000 diáknak és felnőttnek adományoz Micro:bit-et, hogy felkeltse az érdeklődésüket a technológia iránt. [3] Kanadában a CanCode kezdeményezés létrehozta a Kids Code Jeunesse non-profit szervezetet, amely 100.000 Micro:bit-et osztott szét az országban. [4]

Az MIT Média Laboratórium szakemberei [5] úgy gondolják, hogy az egyik oka annak, hogy megjelentek az oktatásban a robotok az az, hogy sokféle területet összekapcsolnak. Robotokkal történő *játéknál*, például szükség van tervezésre és kivitelezésre az építéshez. Az elkészült kreáció életre keltésénél ki kell találni, hogy milyen reakciókat hajtson végre a robot, majd ezeket a robot számára is értelmezhető programmá kell alakítani, tehát programozói tudásra is szükség van. Azonban, tapasztalataik szerint a gyerekek első találkozása robotokkal olyan foglalkozásokon történik, ahol autót kell építeniük, ami abból a szempontból nem előnyös, hogy ez nem feltétlenül érdekel mindenkit. Emiatt Rusk és társai [5] teljesen más stratégiával próbálkoztak, mégpedig témák felől közelítettek. Workshopokat tartottak különböző témákban gyerekeknek, családoknak, oktatóknak. Például voltak olyan foglalkozások, ahol a művészet volt a főszerepben és az általuk képzelt jövő divatját mutatták meg a résztvevők, de volt, ahol a vidámpark volt a téma, és vidámparki játékok születtek, valamint egy történet, ami egy család egy napját meséli el a vidámparkban. A szervezők a workshopokhoz biztosították a technikai eszközöket és az egyéb szükséges kiegészítőket (mint például karton dobozokat, ping-pong labdákat, faleveleket stb.). Tapasztalataik szerint több diákot tudtak elérni így, mint más módszerekkel és kimagasló volt a részt vevő lányok száma az átlagoshoz képest.

A LEGO cég már az 1980-as években is több oktatási alkalmazásban részt vállalt. A robotok, programozható elemek mellett az okosotthonok érzékelőinek, a mechanikai és mérnöki alkalmazások oktatásához is fejlesztettek és állítottak össze csomagokat alaposan kidolgozott tanmenetekkel, segédanyagokkal.[13] [14] [15]

A LEGO Mindstorms EV3 robotok iskolai keretek között történő felhasználásáról több kisebb-nagyobb projekt is beszámol. Követendő példaként szolgál az Egyesült Arab Emírségek kormánya által létrehozott projekt, amelyben 250 állami iskolát láttak el LEGO robotokkal, annak érdekében, hogy a diákok részt vehessenek a Robot Olimpián. A programot 2007-ben indították el és az oktatási tanácsuk szerint a World Robot Olympiad-ra jelentkezők száma gyors ütemben nőtt. 2008-ban 29 csapat vett részt a válogatókon, míg 2011-ben már 602 csapat nevezett a selejtezőkre. [6]

Chaudhary és társai [7], egy indiai kilenc napos, 8-13 éves gyerekeknek tartott nyári tábor tapasztalatait foglalja össze. A táborszervezők arra voltak kíváncsiak, hogy mennyire hatékonyan tudják a gyerekeket problémamegoldásra, programozásra, csapatmunkára és projektszervezésre ösztönözni a robot segítségével. A résztvevő diákok 78%-a úgy nyilatkozott, hogy számára könnyebb volt csapatként megoldani a problémákat, viszont 89% azt mondta, hogy a programozás volt a legnehezebb számára a táborban.

További hasznos tanácsokról, valamint kevéssé sikeres ötletekről számol be Daniel C. Cliburn, a hannoveri egyetem professzora [8]. José Varela Aldás [9], pedig a LEGO Mindstorms EV3 robotok mobil applikációval történő használatáról ír.

[13] https://www.lego.com/hu-hu/aboutus/lego-group/the-lego-group-history/

[14] https://www.lego.com/en-us/lego-history/lego-education-604484c3b3de4314a678ef280d04d216

[15] https://education.lego.com/en-us/lessons

# 2. T@T Kuckó

Az Eötvös Loránd Tudományegyetem Informatikai Karának nemcsak oktatnia kell a hallgatóit, hanem mintaként és követendő példaként is kell szolgálnia más intézmények számára az oktatási tevékenységekben. A kar Média- és Oktatásinformatikai Tanszékén alakult egy egyetemi oktatók alkotta csoport, a T@T Labor. A T@T, a Tanulást (digitálisan) Elősegítő Technológia, az angol TEL (Technology Enchanced Learning) kifejezésből származik. A fő célja ennek a labornak az élményalapú tanulási környezetek és az élményalapú tanulás megvalósítása, népszerűsítése az oktatásban és honosítása hazánkban. [10]

A labor fizikai kiterjesztése, aktivitásainak helyszíne a T@T Kuckó, mely egy technológiai játszóház kicsiknek és nagyoknak, tanulóknak, hallgatóknak, tanároknak, fejlesztőknek, és mindenkinek, aki kicsit is érdeklődik a technika iránt. Fontos kiemelnem a listából a fejlesztőket, mert az alapítók egy másik törekvése, hogy a programozókat is bevonják a labor munkájába, ezzel is közelebb hozva a szoftverkészítőket a tanárokhoz és a közoktatáshoz (vagyis a felhasználókhoz). Ugyanis szükség van az (informatika)oktatás három kulcsszereplője (fejlesztő-tanár-diák) között kapcsolatot létrehozni. [10]

**1. ábra:** Kutatók éjszakája 2019-ben a T@T Kuckóban.

## 2.1. Eszközeink és felhasználásuk

A Kuckóban a legmodernebb eszközök és módszerek megismerésére és felhasználására van lehetőség, amiket a tanárképzésben és a programtervező informatikus képzésben résztvevő hallgatók egyetemi órai körülmények között használhatnak. A résztvevő hallgatók appokat, egyéb eszközöket, foglalkozásokat és tananyagokat dolgoznak ki. A labor célja az élményalapú tanulás meghonosítása hazánkban, amit (ingyenes) foglalkozások szervezésével és tartásával is próbálnak elérni. Ilyen foglalkozások keretein belül van lehetőségük a (tanárszakos) hallgatóknak kipróbálniuk magukat és foglalkozás tervüket, azaz csoportosan, ellenőrzött körülmények között megtartani az órát, amit

kitaláltak. Az óra tervezésekor a hallgatók kamatoztathatják kreativitásukat és a rendelkezésre álló eszközök felhasználásával bármilyen élményalapú órát „készíthetnek”.

A rendelkezésre álló eszközök többek között: LEGO Mindstorms EV3, LEGO WeDo, BBC Micro:bit, mBot, RasberryPi, Edbot, Arduino.

Azon hallgatók, akik sikeresen megtartják óráikat és kedvet éreznek magukban, vállalkozhatnak további foglalkozások tartására. Ilyen foglalkozás lehet a Kutatók délutánján/éjszakáján (1. ábra) pályaorientációs napokon történő „szereplés”, díjkiosztó ünnepségek előtt és után érdeklődést felkeltő foglalkozás tartása.

# 3. Robotika szakkör és tábor

Az ELTE Informatikai Karán először 2018 nyarán rendeztük meg a Kuckó Tábort egy hallgatótárssammal és közös oktatónkkal. Ez egy napközis tábor 10-14 éves diákok számára, ahol a programozással ismertetjük meg a táborozókat a LEGO Mindstorms EV3 robot segítségével. A tábort szerettük volna tovább folytatni az iskola megkezdése után is, így 2019 tavaszán elindítottuk a Kuckó Szakkört is.

## 3.1. A kezdetek

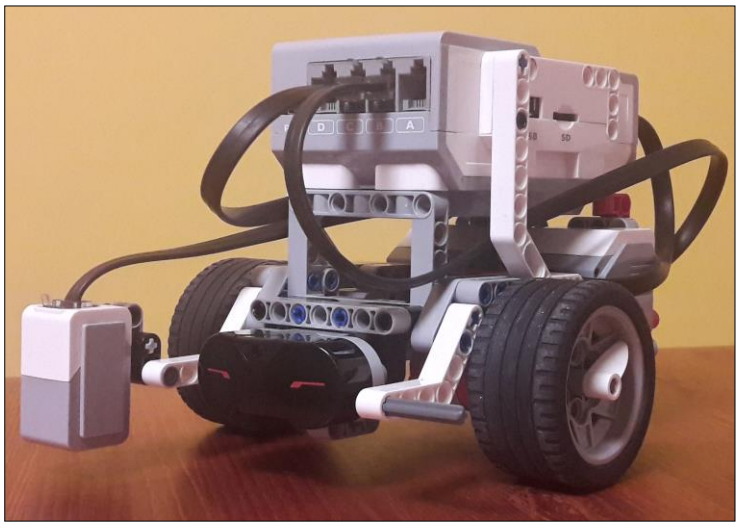
2018 nyarára két turnust terveztünk, mindkét turnusba két csoportot 10-10 fővel. A táborban délelőtt folyt a tanulás, délután pedig közösségépítő programok voltak. A tábort úgy találtuk ki, hogy egy kezdő és egy haladóbb csoportot állítunk össze az alapján, hogy a jelentkezők mit gondolnak a saját tudásukról. (Tehát a jelentkezési űrlapon bejelölhették, hogy már korábban foglalkoztak huzamosabb ideig robotokkal vagy sem.)

A tábor szervezés keretei közé tartozott a délutáni foglalkozások megtervezése, a gyerekbarát menü összeállítása és nem utolsó sorban a délelőtti tematika kidolgozása. Előzetesen nem egyeztettük részletesen a tematikát és a megoldandó feladatokat, csak nagy vonalakban beszéltük meg, hogy ki mit tervez, milyen sorrendben, milyen érzékelőket, eszközöket fog használni, aztán saját magunknak állítottuk össze a tervet. Ennek ellenére közel azonos metodikát követtünk, valószínűleg korábbi közös oktatói tapasztalunk miatt. Ami különbözött, az az volt, hogy a szaktársam a képernyő kezelését emelte ki jobban, míg én a hangok használatát.

## 3.2. Tematika és célok

Fő célunk az volt, hogy játékos formában megismertessük a gyerekekkel a programozást a LEGO Mindstorms EV3 robotok segítségével. Mindezt úgy képzeltük el, hogy a játékos feladatmegoldás során a táborozók a programozás világáról is kapnak egy képet és szereznek egy programozói alaptudást is, amit a későbbiekben alapként használhatnak, ha folytatják az ilyen irányú tanulmányaikat.

A cél eléréséhez a tananyagunkban [2] a hangsúlyt a programozásra fektettük és nem az építésre, így a táborozók nem építettek a tábor során, hanem egy előre összerakott robotot (2. ábra) **2. ábra**programoztak. A diákok egy része kezdetben hiányolta is az építő tevékenységet, de kis idő után mindenki talált olyan részt a programozásban, ami érdekelte. Még azok is érdekesnek találták a programozást, akik főként azért jöttek, mert LEGO-ról volt szó.



**2. ábra:** A táborban általunk használt LEGO Mindstorms EV3 robotok a Robot Educator Driving Base nevű útmutató alapján készültek. Az építési útmutatók online elérhetőek[16].

A robotba beépítésre került egy fény- és színérzékelő, egy nyomásérzékelő, és egy távolságérzékelő (ultrahangos vagy infravörös), méghozzá úgy, hogy a nyomásérzékelő átszerelhető a jármű másik oldalára. A járművet úgy építettük fel, hogy mind az előre, mind a hátra történő mozgás során egyenletesen mozogjon és ne 'ugráljon'. (A robot építési útmutatója megtalálható a szakköri segédanyagunkban is. [2])

A tanulás a robot külső megismerésével kezdődik, amely során a diákok intuitívan próbálják meghatározni a robot egyes moduljait, majd közösen beszéljük át azok tulajdonságait és használati módjaikat. Ezt követően a robothoz tartozó szoftver[17] részeit tekintjük át, amely segítségével egy interaktív környezetben lehet programozni az eszközt. Mivel egy autóhoz hasonló szerkezetről van szó, ezért először a robot mozgatásához szükséges blokkokkal ismerkedünk meg, különböző problémák megoldása során. Sok próbálkozás és gyakorlás után eljutunk a ciklushoz, aminek bevezetése a következőképpen zajlik: a már ismert (motort mozgató zöld) blokkokkal a robot mozgása során 'lerajzolunk' egy négyzetet, majd felvetődik a kérdés, hogy nem lehetne-e ezt a hosszú kódot valahogy leegyszerűsíteni, lerövidíteni. Általában egy-két gyerek hamar észreveszi az ismétlődéseket, de azért, hogy a többiek számára is látható legyen, elkezdjük nyilakkal lerajzolni a robot útvonalát. Ezt követően tanuljuk meg használni a várakozás blokkot, amivel nagyobb részt használjuk az érzékelőket. Először a nyomásérzékelőt, majd a távolságérzékelőt és végül a színérzékelővel szoktunk megismerkedni. A sorrendet az határozza meg, hogy a színérzékelő elég pontatlan és ekkorra már van annyi tapasztalatuk a gyerekeknek a robottal, hogy meg-értik, hogy amit csináltak az jó, és a robot a szenzorának mérési hibája miatt nem jól hajtja végre a parancsokat. Az idő és a haladás függvényében a hangok és/vagy a kijelző használatával egészítjük ki az ismereteket, bonyolítjuk, tesszük komplexebbé a feladatokat.

Az utolsó napra mindig valamilyen akadálypályát készítünk a gyerekeknek, olyan feladatokkal, mint például akadály kikerülése, komplexebb alakzat rajzolása, labda kapuba juttatása a robottal stb.

[16] https://education.lego.com/en-us/support/mindstorms-ev3/building-instructions

[17] www.lego.com/hu-hu/mindstorms/downloads

A gyerekek a feladatokat tetszőleges sorrendben hajthatják végre, több programot is készíthetnek, de nem emelhetik fel a robotot a pályáról. Új programot lehet indítani a bemutatáskor, de egyik állomásról a másikra nem helyezhetik át a robotot. A feladatok ismertetése után, mindig megijednek egy kicsit a gyerekek, főleg, amikor azt mondjuk nekik, hogy ezt önállóan kell megvalósítaniuk. A nagy ijedtség után, nehezen, de elkezdik megoldani a problémákat és ők is rájönnek, hogy nem olyan nehezek ezek a feladatok. Az utolsó kb. 30 percben mindenki bemutatja az akadálypályán, hogy mely problémákat sikerült megoldania és ezzel el is búcsúzunk a robotoktól. Ezzel a kis megmérettetéssel látjuk, hogy mennyi az a tudás, amit magukba szívtak a héten, nekik pedig egy verseny, hogy ki lesz a legtöbb feladattal készen, és nem utolsó sorban egy szép lezárása is a tábornak.

A táborban a délutáni programok során törekedtünk a csapatépítésre, de hagytunk időt a játéknak és a pihenésnek is. Egy délután, pályaorientációs céllal, a gyerekekkel ellátogattunk az EIT Digitals Bogdánfy úti telephelyére, ahol egy előadást tartottak nekünk a szervezet munkásságáról, valamint megmutatták milyen eszközökkel 'játszanak' a programozók náluk. Egy másik napon, egy délutáni foglalkozás keretében, bemutattunk a gyerekeknek egy másik eszközt, a BBC Micro:bitet. Olyannyira megtetszett a gyerekeknek az eszköz, hogy a tábor végén kérték, hogy legyen Micro:bit tábor a következő évben.

## 3.3. További tábori programok

A táborban a diákoknak számos programot biztosítottunk a délelőtti oktatáson kívül, amelyek során törekedtünk a csapatépítésre, de hagytunk időt a játéknak és a pihenésnek is. Egy délután, pályaorientációs céllal, a gyerekekkel ellátogattunk az EIT Digitals Bogdánfy úti telephelyére, ahol egy előadást tartottak nekünk a szervezet munkásságáról, valamint megmutatták milyen eszközökkel 'játszanak' a programozók náluk. Egy másik napon, egy délutáni foglalkozás keretében, bemutattunk a gyerekeknek egy másik eszközt, a BBC Micro:bitet. Olyannyira megtetszett a gyerekeknek az eszköz, hogy a tábor végén kérték, hogy legyen Micro:bit tábor a következő évben.

## 3.4. Szakkör kialakulása

A táborra külön-külön kidolgozott tananyagainkat összeraktuk a szaktársammal egy tananyaggá, kicsit finomítottunk rajta és egy szakköri segédanyag formájában egy EFOP projekt keretében ingyenesen online elérhetővé tettük 2018-ban. [2]

A tananyagba az került bele, amit a táborban megtanítottunk a gyerekeknek, valamint kiegészítettük a következő témákkal: roboton lévő led használata, elágazás használata, képernyő programozása, érzékelők értékének felhasználása, robot gombjainak programozása, változók használata, valamint hozzáfűztünk néhány információt az akadálypályáról és projekt ötleteket.

## 3.5. Új év, új ötletek

2019 nyarára a tábort már csak az oktatómmal ketten szerveztük, de így is sikerült kicsit változtatnunk a táboron. A gyerekek kéréseinek megfelelően indítottunk ténylegesen haladó csoportokat a LEGO robotok programozása terén és nemcsak a LEGO Mindstorms EV3 robotokkal lehetett foglalkozni a 2019-es Kuckó Táborban, hanem a BBC Micro:bit-tel is. Valamint az sem elhanyagolható „fejlesztés", hogy idén már négy turnusban vártuk a lelkes programozni vágyó diákokat.

Korábbi hibánkból tanulva, a haladó csoportba jelentkezőkkel, előzetesen kitöltettünk egy kérdőívet, hogy felmérjük a LEGO Mindstorms EV3 robotok programozásával kapcsolatos tudásukat. Így kevesebb olyan diák jelentkezett haladó csoportba, aki tudásszintje alapján inkább a kezdőbe került volna, mint a korábbi évben. Ugyanis a 2018-as táborban a haladóbb csoportban nem tudtunk olyan ütemben haladni, mint terveztük, mert a gyerekek tudása nem volt olyan biztos, mint ahogy azt ők gondolták. (Sajnos, ebben a korosztályban, még nem tudják olyan pontosan felmérni a saját tudásukat.) Továbbá, a teszt kitöltése azt a célt szolgálta, hogy mindig az adott csoport tudásához igazíthassuk a tananyagot. Emiatt alakult ki olyan helyzet, hogy az első turnusban a haladó csoporttal gyorsan haladtunk, így egy nehezebb akadálypályát és projekteket is meg tudtunk valósítani a tábor

során. Azonban volt olyan csoport, ahol nem haladtunk olyan jól, az alapok mélyebb átismétlésére volt szükség, így kevesebb újdonságot tudtunk megnézni és emiatt náluk a projekt rész is kimaradt. A hét megfelelő lezárása érdekében (és a projekt helyett) kaptak egy összetettebb akadálypályát a táborozók, amivel bizonyítani tudták, hogy tényleg elsajátították az alapokat.

További újítás volt a 2019-es táborban, hogy a délutáni programba beillesztettünk egy planetáriumi látogatást, egy drónos és nagyon sok szabadtéri foglalkozást, hogy ne csak szellemileg, hanem fizikailag is lefáradjanak a gyerekek a táborban.

# 4. Tapasztalatok

## 4.1. Tanári tapasztalatok

Az első évben mindenki a saját módján tanított a táborban, mindketten azt, amit gondoltunk. A tananyagban nem is volt sok különbség, de például a szaktársam minden napot egy rövid ismétléssel indított egy Kahoot![18] játék keretében. Eleinte nem tartottuk annyira fontosnak ezt a fajta ismétlést, tekintve, hogy a táborban előző nap foglalkoztunk az adott témával, de a szakkörön teszt jelleggel bevezettük. Az akkori csapat nagy része ismerte a játékot és örömmel fogadták az ilyen irányú kezdeményezést. Kérdezték rögtön a játék után, hogy lesz-e ilyen a következő órán is. A pozitív fogadtatás miatt volt, és az azt követőn is, sőt még a táborban is bevezettük következő évben. Ugyanis ezzel a módszerrel ténylegesen fel tudtuk mérni, hogy mi az, amit tudnak, mit kell még gyakorolnunk, tisztáznunk. Ezt a fajta ismétlést összesen öt csoportban próbáltuk ki a tavalyi tábor óta és mindenhol nagy lelkesedéssel fogadták a gyerekek. Tapasztalataink szerint mindig ugyanazok a problémák merülnek fel. Nehezen látják át a gyerekek, hogy hogyan működik az a program, amit ők készítettek, de rendszeresen felmerülő probléma a mozgatás meg nem értése, a ciklus használatának kérdésköre és a változókig még el sem jutottunk. Ezek miatt, a problémás rész(ek) új módszer szerinti tanítását tervezzük.

Egyre több csoportban tapasztaljuk azt, hogy a gyerekek nem mernek vagy nem akarnak kísérletezni, próbálkozni. Általában egy új anyag tanítása/tanulása úgy szokott kinézni, hogy elmagyarázzuk és bemutatjuk az új blokk működését, kiemeljük, hogy mire kell figyelni a használatakor. Ezt követően adunk néhány feladatot, amiben kipróbálhatják a működését a blokknak. A feladatokat úgy adjuk ki, hogy először csak az adott (új) blokkot kell használni, majd ha az már megy, akkor a korábban tanultakkal együtt kell felhasználni a feladat megoldásakor. A tanulás utolsó fázisában, amikor alkalmazniuk kellene a tanultakat és talán egy kis kreativitást kellene még a megoldásba csempészniük, akkor mintha leblokkolnának a gyerekek. Például, először az előre és a hátra történő mozgást tanulják meg a gyerekek a robottal, majd, ha ez sikerül, akkor a robotot úgy kell beprogramozniuk, hogy (fizikailag) egyhelyben derékszögben elforduljon. Nagyon sokféleképpen meg lehet oldani a feladatot, de ilyenkor nagyon sokan csak ülnek és várnak a megoldásra. Eleinte az új közösségbe kerülés nehézségeivel magyaráztuk a jelenséget, de miután több csoportban is tapasztaltuk ezt, más okok meglétét feltételezzük.

A 2019-es táborban két csoportban próbáltuk ki a projekt alapú robotépítést. Az első csoportról már volt szó (3.5. bekezdés), velük három nap tanultunk, a negyedik napon akadálypályát kellett megoldaniuk a gyerekeknek, és az utolsó napon építkeztünk. Ez egy olyan hét fős csoport volt, ahol a csapat öt tagja az előző évi táboron is részt vett, szinte mindannyian kiemelkedően teljesítettek akkor, és többségük már tudott programozni más nyelven (saját bevallásuk szerint), így kicsit nyu-

[18] A Kahoot! egy online, ingyenes feladatsor-készítő program, amelyben többféle feladattípus közül választhatunk. Segítségével, a korábban tanultakat egy kvíz jellegű játékkal ismételhetjük át.

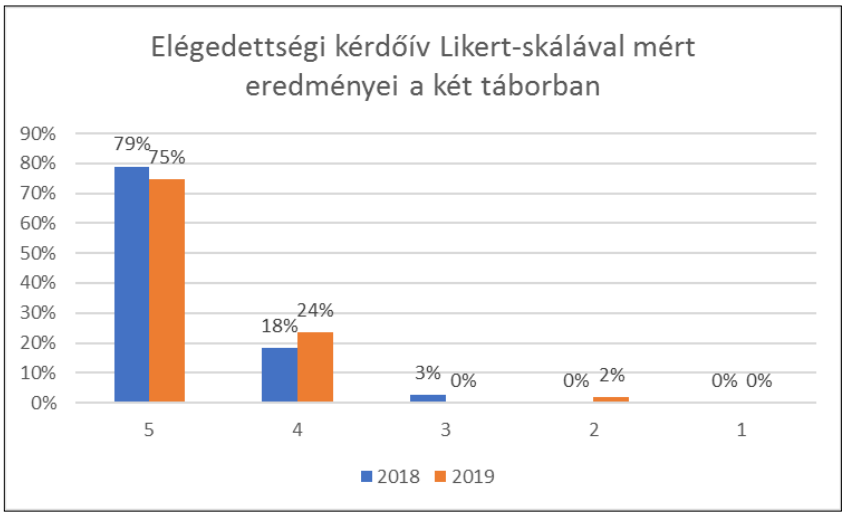
godtabban vágtunk bele a projektbe. A gyerekek előzetes feladata az volt, hogy találjanak ki maguknak egy projektet, hogy ne ezzel menjen el az idejük az építéstől. Sajnos, ezt nem tették meg, de mivel előrelátóan készültünk erre az esetre is projektötletekkel, ezt a problémát hamar orvosoltuk. A projektek megvalósításánál a gyerekeknek nagyon nehezen ment a csapatmunka, még azoknak is, akik baráti társaságként, együtt jöttek a táborba. Az egyik csoport, az általunk felajánlott ötletekből egy nagyon szép, de bonyolult projektet választott, egy elefánt felépítését és programozását. Az építést hiába osztották fel egymás között, nem haladtak jól vele, így a délutáni program helyett ők inkább befejezték a projektet. Az elefánt elkészült, még mozogni is tudott egy keveset, de picit csalódottan távoztak a csapattagok az időbeni csúszásuk miatt. Sajnos, ez egy olyan része a robotikának, ahol nehezen lehet belőni a foglalkozásra fordítandó időt. Ezt mi is megtapasztaltuk és a következő csoportnál igyekeztünk korrigálni azzal, hogy kihagytuk az akadálypályát és helyette a projekttel foglalkoztunk. Ezen a héten mindenki be tudta fejezni a projektjét, sőt, utána még szét is tudták szedni őket, amire azért volt szükség, mert a következő heti csoportnak is a korábbi tananyagok ismétlésével kellett kezdeni, amihez a korábbi felépítésre volt szükség.

A szakkörön az egyik diák, aki még csak 10 éves volt akkor, valamilyen szinten már tanult Imagine Logo-ban programozni. Annál a feladatnál, amikor egy négyzetet kell rajzoltatniuk a robottal, ez a fiúcska megjegyezte, hogy az EV3 olyan, mintha az Imagine Logo-ból a teknőst programozná, csak ez 3D-ben meg is jeleníti az alakzatokat és meg lehet fogni. A kisfiúnak nagyon jó meglátása volt, hiszen a tanulási folyamat egész hasonló. Ott is, először a teknős mozgatását tanulják meg a gyerekek, majd utána jönnek az érdekesebb dolgok, a színezések, a paraméterezések, a rekurziók stb.

## 4.2. Visszajelzések

Mindkét évben 10 és 14 év közötti gyerekek táborozhattak a Kuckó Táborban. A 2018-as táborban 39 fő, míg a 2019-es táborban 58 fő vett részt. Nemcsak a létszámok, hanem a nemek arányában is pozitív változást tapasztaltunk, míg az első évben 5 lány és 34 fiú, addig a második évben 11 lány és 47 fiú táborozott a Kuckóban.

Minden tábor végén kitöltettünk a diákokkal egy anonim elégedettségi kérdőívet, melyben értékelniük kellett, hogy hogyan érezték magukat, valamint meg kellett fogalmazniuk, hogy mi tetszett és mi nem tetszett nekik a tábor során. A kérdéseket egy Google űrlap formájában tettük fel. Arra a kérdésre, hogy hogyan érezték magukat, egy ötfokú Likert-skála segítségével válaszolhattak (ahol az egyes érték azt jelenti, hogy nem érezték jól magukat, az ötös érték pedig azt, hogy nagyon jól érezték magukat). A válaszok átlagos értéke 2018-ban 4.8, míg 2019-ben 4.7 volt. Az értékek szórása 0.37 volt az első évben, a másodikban pedig 0.4, ami egy új tábor számára kimagasló eredmény. A 2018-as kitöltők 79%-a adott 5-ös értékelést, míg a 2019-es kitöltők 75%-a tette ezt. A további válaszok eloszlása a 4. ábrán látható.




**3. ábra:** A táborozókkal a tábor végén kitöltetett elégedettségi kérdőív "Hogy érezted magad a táborban?" c. kérdésére ötfokú Likert-skála alapján adott válaszok eloszlása.

Azért, hogy átfogóbb képet kapjunk arról, hogy hogyan érezték magukat a gyerekek a táborban, megkértük őket, hogy írják le pár szóval, hogy mi tetszett nekik. A következőkben ezen válaszokból olvashatnak néhányat:

- „Az tetszett a legjobban, hogy megtanultuk használni a szenzorokat, és bonyolultabban programozhattunk robotot."
- „A programozás meg hogy sokat tanultam, és nagyon sokat segítettek nekem."
- „Kahoot!"
- „A sok izgalmas feladat meg a délutáni programok."
- „Informatív foglalkozások voltak, a közösség is nagyon jó volt"
- „a színérzékelő tanulása"
- „hogy délelőtt programoztunk és minden délután más program volt"
- „A programozás."
- „A zenélés micro:bittel."

A kérdőívbe beleraktunk egy egyéb megjegyzések/észrevételeknek szóló részt. Először voltak kételyeink ezzel kapcsolatban, például, hogy megéri-e beletenni a kérdést, elég érettek-e ehhez a gyerekek. Azt kell mondanom, pozitívan csalódtunk. Aki írt ebbe a rubrikába valamit, az építő jelleggel tette ezt. Ezek közül kiemeltünk néhányat:

- „Jövőre legyen haladó csoport is." (Ez a válasz a 2018-as tábor kérdőívére érkezett.)
- „Az, hogy folytassátok és biztos, hogy jövök jövőre." (Ez a válasz a 2018-as tábor kérdőívére érkezett.)
- „Kéne egy olyan csoport, amiben pythonban programozzák az ev3-at."
- „RasberryPI Porogramozás Csoport, C# Programozás Csoport."
- „Legyen jövőre haladóbb haladó csoport."

A szülőkkel nem töltettünk ki kérdőívet a táborral kapcsolatban, de kaptunk néhány e-mailt, amiben megerősítették azt a feltételezésünket, hogy a gyerekek jól érezték magukat a táborban. Ezek közül egyet emelnénk ki, amely így szólt (a gyermek nevét az anonimitás miatt rövidítettük): „Köszönjük a képeket! B. jól érezte magát a táborban, sok dolgot megtanult. Most már itthon is van microbitünk.". Továbbá, azt figyeltük meg, hogy kezd kialakulni egy visszajáró közösség a táborban. Emiatt is gondoljuk azt, hogy elégedettek a szülők is a munkánkkal.

## 5. Összefoglaló

Tapasztaljuk, hogy a gyerekek nagyon lelkesek, érdeklődőek a témával kapcsolatban, néha olyan váratlan kérdéseket tesznek fel 9-10 évesek, amikre egy felnőttől számítunk. Nagyon sok diák úgy hagyja el a tábort, szakkört, hogy behívja a szülőt a terembe megmutatni, hogy milyen eszközzel foglalkoztunk és kéri, hogy hadd kapjon ilyet. Több visszajáró diák mesélte már, hogy szülinapjára Micro:bitet vagy EV3-t kért, vagy esetleg a saját zsebpénzéből vásárolta meg az áhított eszközt. Ahogy a [11]-es cikkben is kifejtették, lenyűgöző, hogy mennyire vonzza a gyerekeket a robotika világa és mennyien szeretnének ezzel foglalkozni (még akár a saját zsebpénzük árán is).

Munkámat folytatva, tovább népszerűsítem a programozást a robotokkal. A tábor és a szakkör során szerzett tapasztalataimmal javítva a tananyagot, és továbbfejlesztve az oktatási módszereket. Valamint terveim között szerepel, hogy megvizsgálom, hogyan fejlődik a gyerekek programozói tudása a robotokkal történő programozás tanulás során.

## Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani oktatómnak, Pluhár Zsuzsának, akinek a támogatása és segítsége nélkül nem jöhetett volna létre a Kuckó Tábor. Külön köszönöm szaktársamnak, Barbalics Dóra Krisztinának, hogy együtt dolgozhattunk mind a tábor létrehozásban, mind a tananyag fejlesztésben.

## Irodalom

1. Abonyi-Tóth Andor: *Programozzunk micro:biteket!*, ELTE Informatikai Kar, Budapest, 2018.
2. Barbalics Dóra Krisztina, Solymos Dóra: *Lego Mindstorms EV3 robotok programozása*, ELTE Informatikai Kar, Budapest, 2018. http://tet.inf.elte.hu/tetkucko/wp-content/uploads/2018/12/legomindstorms_szakkorianyag.pdf (utoljára megtekintve: 2019. november 3.)
3. *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: 2019. november 3.)
4. *Meet the BBC micro:bit*, 2019. https://www.bcs.org/content-hub/meet-the-bbc-microbit/ (utoljára megtekintve: 2019. november 3.)
5. Natalie Rusk, Mitchel Resnick, Robbie Berg, Margaret Pezalla-Granlund: *New Pathways into Robotics: Strategies for Broadening Participation*, Journal of Science Education and Technology, USA, 2007., DOI 10.1007/s10956-007-9082-2
6. E. Afari and M. S. Khine: *Robotics as an Educational Tool: Impact of Lego Mindstorms*, International Journal of Information and Education Technology, 2017. http://www.ijiet.org/vol7/908-T108.pdf (utoljára megtekintve: 2019. november 3.)
7. Vidushi Chaudhary, Vishnu Agrawal, Pragya Sureka, Ashish Sureka: *An Experience Report on Teaching Programming and Computational Thinking to Elementary Level Children using Lego Robotics Education Kit*, IEEE, 2016., DOI: 10.1109/T4E.2016.016

8. Daniel C. Cliburn: *Experiences with the LEGO Mindstorms throughout the Undergraduate* Computer Science Curriculum, IEEE, 2006., DOI: 10.1109/FIE.2006.322315
9. José Varela-Aldás, Oswaldo Miranda-Quintana, Cesar Guevara, Franklin Castillo and Guillermo Palacios-Navarro: *Educational Robot Using Lego Mindstorms and Mobile Device,* Advances and Applications in Computer Science, Electronics and Industrial Engineering, 71-82 (2019)
10. Andor Abonyi-Tóth and Zsuzsa Pluhár: *Wandering micro:bits in the public education of Hungary*, 12nd International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2019, Larnaca, Cyprus, 2019.
11. Dr. Abonyi-Tóth Andor: *A micro:bitek felhasználási lehetőségei az oktatásban*, InfoDidact, 2018. https://people.inf.elte.hu/szlavi/InfoDidact18/Manuscripts/ATA.pdf (utoljára megtekintve: 2019. november 3.)

# Szimulációs modellekkel támogatott programozás tanítása az alapiskolában

Stoffová Veronika[1], Czakóová Krisztina[2]

[1]NikaStoffova@seznam.cz
Nagyszombati Egyetem, Tanárképző Kar
[2]czakoovak@ujs.sk
Selye János Egyetem, Gazdaságtudományi Kar

**Absztrakt.** A szimulációs modellek alkalmazása jól kidolgozott és gyakran használt kutatási eljárás az egzakt, főleg műszaki tudományok területén. Új rendszerek megismerésére, beazonosítására, működési elveiknek feltárására és így a modellezett (reális vagy fiktív) objektumokról új tudásszerzésre ad lehetőséget. A matematikai modellen alapuló szimulációs kísérletek nélkülözhetetlenek olyan dinamikus folyamatok esetében, amikor az elemzett esemény veszélyes, idő- vagy anyagi befektetésre nézve igényes, esetleg túl gyorsan játszódik le vagy szabad szemmel nem megfigyelhető stb. Az utóbbi időben a szimulációk jelentős szerepet kaptak a mély tanulásban, az úgynevezett „deep learning" -ben. A jól megtervezett algoritmussal irányított vizualizált szimulációs modellel végzett kísérletek által a tanuló (diák, hallgató) számára fontos, a tanulás témáját érintő ismeretekre tehet szert, amit hasznosíthat a témával kapcsolatos feladatok megoldásában egyaránt. A saját tapasztalatok, a megfigyelés és meggyőződés alapján szerzett ismeretek mélyebbek, tartósabbak és rendszerezettebbek. Így a tanuló könnyebben fedezi fel az összefüggéseket, aktívan tudja használni és egyúttal felépíteni saját tudásrendszerét. A tanulmány két szimulációs játékot is bemutat, mely a programozás játékos tanítását hivatott támogatni az alapiskolán.

**Kulcsszavak:** programozás, programozás tanítása, programozás tanulása, mély tanulás

## 1. Bevezető a modellezés és szimulációba

A modellezés és szimuláció erős oktástechnológiai eszközzé fejlődött. Számos tantárgyban alkalmazható, és nem utolsó sorban a mély tanuláshoz vezet. Főleg olyan témakör tudásanyagának elsajátításban érvényesíthető, ahol nélkülözhetetlen az összefüggések megértése és azok egzakt kifejezése. A viszonylag egyszerű matematikai modell alapján számos dinamikus jelenség és folyamat is bemutatható, és így könnyebben elérhetők a kitűzött didaktikai célok.

Olyan jelenségekre és folyamatokra gondolunk, amelyek már ismertek, matematikailag pontosan leírhatók és az oktatás tárgyát képezik. A paraméterek által vezérelt animációkkal be lehet szemléletesen mutatni a szimulációs kísérletek eredményeit. A jól megtervezett és pontosan végrehajtott szimulációs kísérletek lehetővé teszik a modell paraméterváltozásai által a modellezett objektum viselkedésére gyakorolt hatás tanulmányozását. Alternatív megoldásként a szimulációs kísérleteket fel lehet használni a modellezési folyamatok tulajdonságainak feltárására (felfedezésére) és tesztelésére, amelyet a felhasználó/kutató/tanuló saját megfigyelései alapján végez. A dinamikus szemléltetés, a vizuális ábrázolás és a grafikus kifejezés elősegítik a kísérlet eredményeinek helyes megértését és azok helyes értelmezését.

A szerzők a szimulációt főképp, mint tanítási és tanulási módszert kívánják bemutatni. A szimulációs modellek a kísérletek eredményei alapján a modellezett objektummal kapcsolatos új ismeretek megszerzésére szolgálnak. A műszaki, természettudományi, gazdaságtudományi és más egzakt tudományok területén a megszerzett tudás relevanciáját a tanulmányozott tárgy esetébe a rendszer pontos matematikai modellje biztosítja. A módszer korlátlan lehetőségeket ad a modellezett rendszer – a tanulás tárgyának megismerésére [1], [2], [3], [5].

A jól megvalósított matematikai modell helyettesítheti a valós objektumot, és lehetővé teszi annak tulajdonságainak tanulmányozását szimulációs kísérletek segítségével, hogy a megfigyelő/tanuló új ismereteket szerezzen a modellezett objektumról.

## 1.1. Mi is valójában a modellezés és a szimuláció?

A szimulációs modell segítségével információkat kaphatunk arról, hogyan fog viselkedni valami anélkül, hogy valóban tesztelésre kerülne a reális életben. Például abban az esetben lehet segítségünkre, ha egy versenyautót akarunk tervezni, és nem tudjuk pontosan, milyen funkcionális részeket építsünk bele. Az autó számítógépes szimulációjával képesek vagyunk felbecsülni a különböző alkatrészek, formái megoldások és egyéb paraméterek eredményre gyakorolt hatását. Hasznos betekintést nyerhetünk a különböző döntésekbe, amelyeket az autógyártásnál hozhatunk anélkül, hogy az autó prototípusát elkészítenénk.

Általánosabban fogalmazva, a szimuláció modelleket alkalmaz, beleértve emulátorokat, prototípusokat és szimulátorokat a (tervezett) objektum adatai és paraméterei kidolgozása érdekében, vezetői vagy műszaki döntések meghozatalához.

A „modellezés" és a „szimuláció" kifejezéseket gyakran felcserélhetően szinonimaként használják – ez azonban nem mindig helyes. A szimuláció a műszaki gyakorlatban a modellel végzett szimulációs kísérleteket jelenti [8].

## 1.2. A modellezés és szimuláció, mint tudományág

A modellezés és a szimuláció (M&Sz) fontos szerepet játszik a tudományos kutatásban. A valós rendszerek reprezentálása – akár fizikai reprodukciók révén (kisebb méretben), akár matematikai modellek segítségével, amelyek lehetővé teszik a rendszer dinamikájának szimuláción keresztüli ábrázolás – lehetővé teszi a rendszer viselkedésének olyan szintű feltárását, amely gyakran nem lehetséges, vagy túl kockázatos lenne a valóságban.

A M&Sz használata a mérnöki kutató, fejlesztő és irányító munkák során bevált és elismert. A szimulációs technológiák az összes alkalmazási szakterületeken felhasználhatók, beleértve a mérnöki menedzsment szakterületét is, hiszen a szimulációs modellek helyes alkalmazásával hozzájárulhatnak a költségek csökkentéséhez, a termékek és rendszerek minőségének javításához. Annak biztosítása érdekében, hogy a szimuláció eredményei alkalmazhatók legyenek a reális életben, egy menedzsernek meg kell értenie eme feltörekvő tudományág feltételeit, koncepcióit és megvalósítási korlátait egyaránt [8].

## 1.3. A modellezés és szimuláció előnyei

A szimulációs alkalmazások iránt az érdeklődés folyamatosan növekszik. Ennek okai a következők:

1. A szimulációk használata általában olcsóbb és biztonságosabb, mint a termék prototípusán végzett kísérletek.
2. A szimulációk lehetővé teszik egy virtuális környezet létrehozását, amely megengedi a szimulált rendszerek megvizsgálását, részletes elemzését, még a tervezési szakaszban. Majd vegyes virtuális rendszerek segítségével, beépítve az első prototípusú komponenseket, a tervezett rendszert tesztelni és optimalizálni lehet virtuális környezetében. És ezt mind még az az első funkcionális rész kiépítése előtt.
3. A szimulációk gyakran gyorsabban hajthatók végre, mint azok a valós időben megtörténnek. Ez lehetővé teszi a különféle alternatívák hatékony elemzését, különösen akkor, ha a szimuláció inicializálásához szükséges adatok könnyen beszerezhetők az operatív adatokból. A szimulációk gyakran valósághűbbek is lehetnek, mint a hagyományos kísérletek, mivel lehetővé teszik a végtermék operatív alkalmazási területén található környezeti paraméterek szabad konfigurálását.

4. Az M&Sz elmélete és gyakorlata szorosan összefügg és kölcsönösen támogatja egymást, és így meghatározzák a M&Sz tudományos alapjait.
5. Az M&Sz elmélete folyamatosan fejlődik és keresi az alkalmazható megoldási mintákat. A hangsúlyt az általános módszerek jelentik, melyek különféle problémakörökben alkalmazhatók.
6. Az M&Sz alkalmazások úgy oldják meg a valós világ problémáit, hogy megoldási lehetőségekre és stratégiákra összpontosítanak. Gyakran a megoldás csak egy módszer alkalmazásából származik. Sok esetben hangsúlyosan problémakör-specifikus, és a problémakör-szakértelemből vezetik le, nem pedig az általános elméletből vagy módszerből.
7. A modellek különféle egységekből (finomabb felbontású részletes modellekből) összekapcsolhatók egy adott cél elérése érdekében, ezért modellező megoldásoknak is nevezhetők [8].

### 1.4. A modellezés és szimuláció folyamata

A modellezés folyamata iteratív jellegű. Magába foglalja a következő lépéseket, amelyek szakaszonként vagy egyenként is megismételhetők (javíthatók):

- a modellezés tárgyának meghatározása;
- a modellezés céljainak meghatározása;
- az objektum, mint rendszer azonosítása (a rendszer bevezetése a kiválasztott objektumon);
- az alrendszerek és a részletek szintjének meghatározása;
- a modell matematikai leírásának meghatározása;
- matematikai modellek implementálása a számítógépen;
- szimulációs kísérletek megvalósítása;
- a szimulációs eredmények értelmezése [8].

## 2. Szimulációs játékok algoritmizálás és programozás tanítására

Ebben a fejezetben két játékot, illetve a játékos programozás tanítására szolgáló alkalmazást mutatunk be [7], [9], [10], [11], [12], melyek az alapiskolai programozás oktatására és a problémák megoldásának algoritmizálására fókuszál. A programozás tanításában és tanulásában fontos a feladat megoldását standard algoritmuselemek alapján kifejezni. Ennek a begyakorlását szolgálhatják a következő alfejezetekben bemutatott interaktív szimulációk.

### 2.1. Stratégiafejlesztés alapú játék az alapvető vezérlési szerkezetek megértésére

Az itt bemutatott játék a szekvencia (utasítások egymás után történő végrehajtása), az elágazás (a feltételtől függően a végrehajtható ágak száma akár több is lehet) és a ciklus (adott számú ismétléssel) programrészek (programnyelvi konstrukciók) megértésére és aktív használatára irányul. A számítógépes játék három nehézségi szintre épül, amelyek a játék folyamán egymást követik a kapott feladat (vizuálisan megjelenített szimulációs modell) sikeres megoldását követően. A játékfelület engedélyezi a kínált utasítások (lépés előre, hátra, forgás 90 fokkal jobbra vagy balra) szekvenciába való foglalását, majd az összeállított program megvalósítását és vizuális kiértékelését [7].

A kalózlány vagy kalózfiú vezérlésére szolgáló ikonok, amelyek a játék képernyőjének bal alsó sarkában találhatók, a programozható játékok irányítására szolgáló nyomógombokra emlékeztetnek. Így a diákok kamatoztathatják a jól ismert programozható méhecske (Bee-bot) vagy programozható autó (Pro-bot) használata során szerzett tapasztalataikat és tovább fejleszthetik képességeiket algoritmus (feladatmegoldás menet) írásból. A következő 3 képernyőképből világos a játék menete,

valamint interaktív és intuitív kezelése. A program a jobb felső sarokban kerül ábrázolásra. Itt jelenik meg a program helyességének vizuális értékelése is.


**1. ábra:** 1. pálya – szekvencia kialakítása elemi utasítások által


**2. ábra:** 2. pálya – számláló ciklusvezérlés és elágazás megvalósítása

**3. ábra:** 3. pálya – nehezítéssel, akadályok leküzdésével

Az előbbi ábrák a játék menetét és kezelését illusztrálják. Az első ábra egy egyszerű útvonal bejárását mutatja. A szekvencia begyakorlására szolgál. A második ábrán látható, hogy a játékpálya bejárását számláló ciklus alkalmazása teszi gyorsabbá. A pálya végén feltételes elágazással is szembesülhetünk, melynek feltétele a több drágakő megszerzése. Az út bejárását (programot) a jobb felső sarokban elhelyezett ablak mutatja a kiválasztott lépések grafikus megjelenítésével. Az ábrán az is látható, hogy az egyes vezérlő struktúrák (parancs szekvencia – szürke, ciklussal kiválasztott utasítássorozat– kék) más-más színnel vannak megjelenítve. A harmadik ábrán a bejárandó útvonal nehezítéssel, azaz akadályokkal (harcos kalózokkal) került kibővítésre. Ezeket jobb elkerülni, hogy ne kelljen a rossz kalózokkal szembesülni, néhányat kivéve, akikkel való megküzdés jutalma a félúton megszerezhető drágakövekben lakozik [7].

A játék (applikáció) a *Visual Studio* 2015 fejlesztőkörnyezetben készült, C++ programnyelv segítségével. A játék grafikai elemeinek szerkesztéséhez a Paint.net rajzprogram szolgált.

## 2.2. A ciklusok játékos tanítása

A következő alfejezetben bemutatásra kerülő interaktív szimulációk célja a különböző ciklus típusok értelmezése, azok megértése, különös tekintettel az algoritmusok és programozás témakör oktatása során [7], [12], [13].

Az itt leírt interaktív szimulációk egységes grafikus szerkezettel rendelkeznek (szimulációs feladat megnevezése, rövid leírása, a megoldás algoritmusának folyamatábrával és programkóddal való kifejezése és az eredmény vizualizálása). A feladat megnevezése és rövid leírása az ábrákról hiányzik, mivel e szimulációk eredetileg szlovák nyelven készültek. Így azok feltüntetésétől eltekintettünk (a szlovák nyelvet nem értő olvasó számára ezen adatokat elhanyagolhatónak ítéltük). Remélhetőleg az ábrákból így is világosan látszik az interaktív szimulációk kezelése, az információk formájának kifejezése és a rész-/eredmények vizualizálása, ami a szimulációk lényeges küldetése.

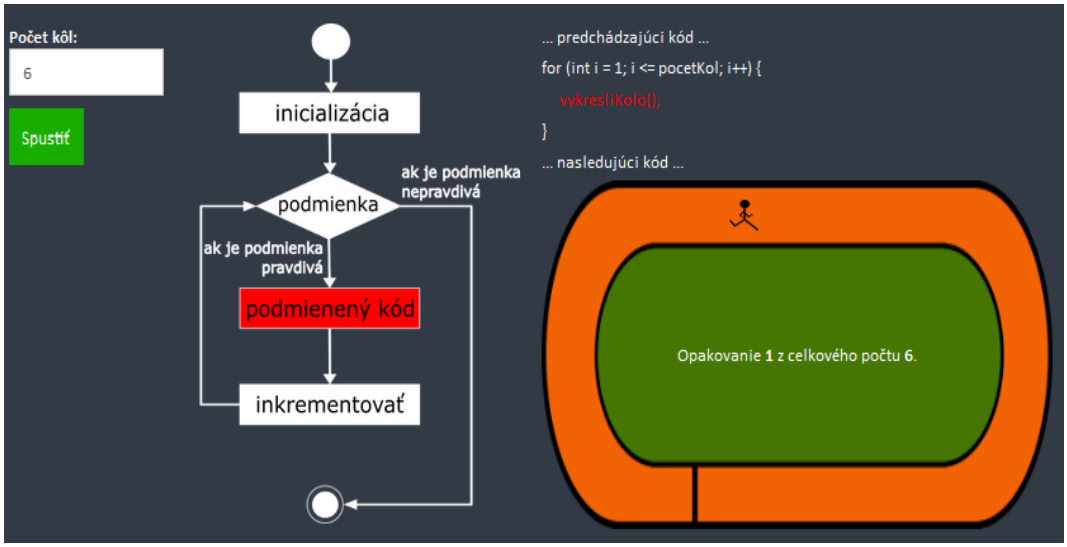
Minden animációhoz felhasználói kézikönyv is készült, amely módszertani anyagként szolgál tanítók/tanárok számára az adott szimuláció megfelelő alkalmazására az oktatásban.

### 2.2.1. Megadott számú ismétlés – for típusú ciklus

A ***for ciklus*** működési elveit egy életből való példa mutatja. A futó a felhasználó (tanuló) által megadott számú kört fut le egy futballpálya körül. A negyedik ábrából világos a szimuláció paramétereinek beállítása és a feladat elvégzésének nyomon követése [4].

A szimulációs modell paramétere egy - a megtett körök számát képviselő - pozitív egész szám, amely alapján a futó lefutja az adott számú kört. Ebben a szimulációban a végkimenet három formában került megjelenítésre: animációval, folyamatábrával és program formájában.

Az algoritmus egyes lépéseit a szimuláció pillanatnyi állapotát követve figyelhetjük meg, a kód kiemelt része, a folyamatábra, illetve a futó mozgása által. A szimuláció vezérlésére az Indítás (***Spusti***) gomb szolgál [4].

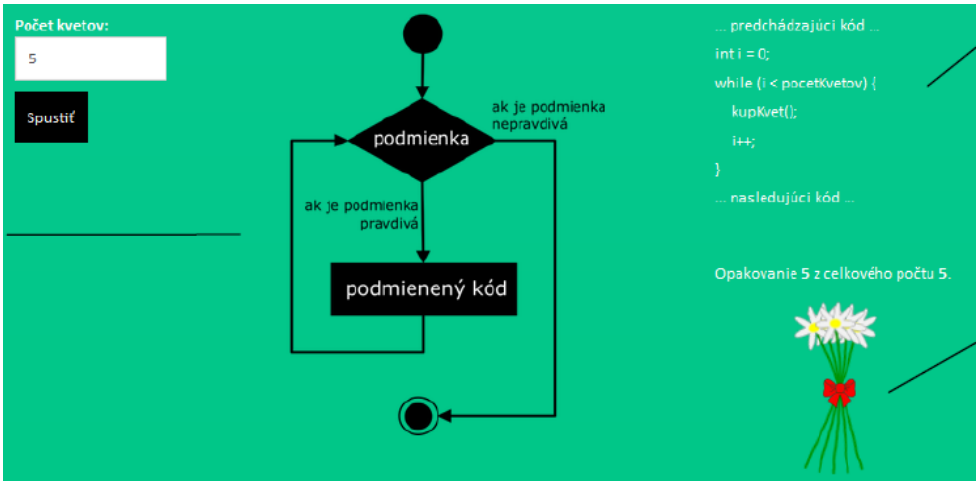



**4. ábra:** Adott számú ismétlés – *for* típusú (számláló) ciklus szemléltetése

### 2.2.2. Ismétlés előfeltétellel – elöltesztelő ciklus (while típusú ciklus)

A ***while* típusú ciklus** működési elvét a szimuláció egy csokor elkészítése példáján mutatja be. A felhasználó megadhatja, hány virágot kíván csokorba kötni. Az ötödik ábrából világos a szimuláció paramétereinek beállítása és a feladat végrehajtásának nyomon követése.

A szimuláció létrehozása és felépítése hasonló az előző szimulációhoz. Grafikus (animált) objektumként szerepelnek: a virág, a szalag, és a folyamatábra [4]. Azt a feltételt kell megadni, amellyel a csokor a virág kirajzolása által van (ismétléssel) irányítva (5. Ábra). A virágok csokorba fűzését virág alakú kép fokozatos kirajzolása biztosítja, különböző elforgatással. Az első virágot függőlegesen felfelé rajzoljuk, és minden ezt követő virágot 10 °-kal elforgatjuk (felváltva minkét irányba), figyelembe véve a virágok egyenletes eloszlását a csokron belül. A ciklus szükséges ismétlésének elvégzése után befejezzük a csokor kirajzolását és a csokrot egy szalaggal megkötjük.




**5. ábra:** Az elöltesztelő ciklus szemléltetése

### 2.2.3. Ismétlés utófeltétellel – hátultesztelő ciklus (repeat - until típusú ciklus)

A ***repeat - until*** **típusú ciklus** működési elveit a szimuláció folyadékmelegítés példáján mutatja be. Többféle előre definiált folyadékból lehet választani (pl. víz, olaj stb.). A kiválasztott folyadék alapján a program automatikusan hozzárendeli a hozzá tartozó szükséges paramétereket (sűrűség, fajhő). A felhasználó definiálja még a többi belépő adatot, mint: a kezdőhőmérsékletet, a véghőmérsékletet, a melegítő spirál teljesítményét és a folyadék térfogatát [4].

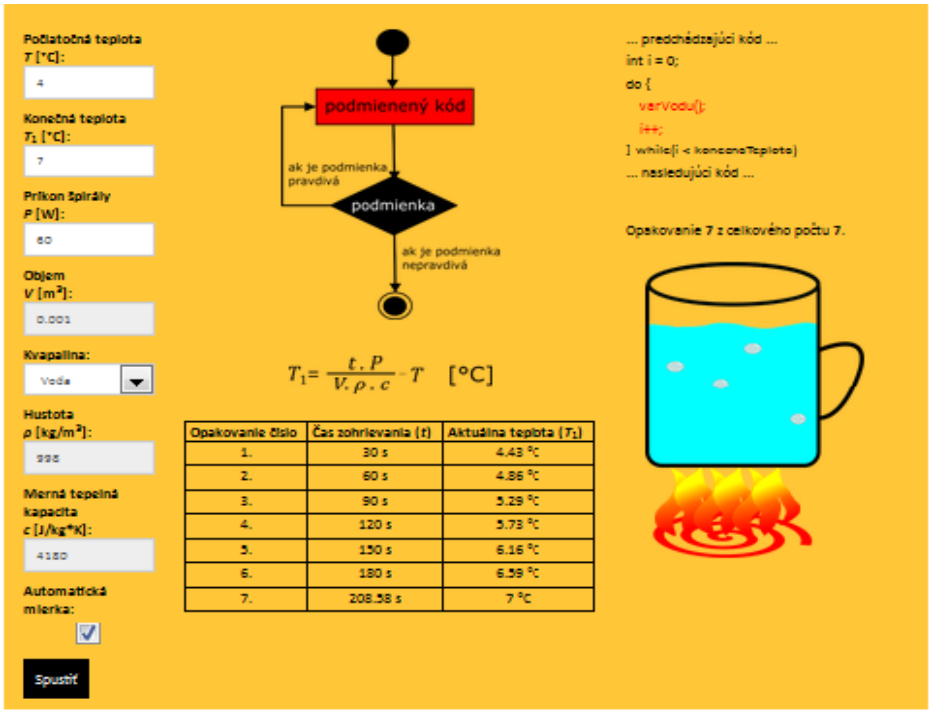
A folyadékmelegítés szimulációjával ábrázoljuk a ciklus menetének folyamatát, amely a ciklus lefutása után teszteli a befejezést/leállítást irányító feltétel teljesülését. A ciklust ugyanúgy határozzuk meg, mint az elöltesztelő ciklusnál. A különbség azonban lényeges. A hátultesztelő ciklus esetében a ciklustest rész legalább egyszer lefut, mivel az első tesztelés az első végrehajtás után valósul meg. Amint a meghatározott ciklusfeltétel teljesül, a program leáll. Az algoritmus alapján a szimuláció addig melegíti a folyadékot, amíg az el nem éri a felhasználó által meghatározott véghőmérsékletet (6. Ábra). A példa fizikai ismeretekre épül, így a programozás mellett a fizikából szerzett jártasságokat és készségeket is próbálja fejleszteni.

A 2.2 alfejezet alatt bemutatott interaktív szimulációk a HTML, CSS, JavaScript használatával készültek, így alkalmasak web böngészőbe való betöltésre és futtatásra [4], [14], [15], [16].

## 3. Befejezés

A cikkben rámutattunk az interaktív animált szimulációk jelentőségére és hozadékára a mély tanulásban. A pedagógiai megfigyeléseink, több éves tapasztalataink a programozás tanításában és az elvégzett pedagógiai kísérletek eredményei is alátámasztják azt, hogy az animációkkal kísért szimulációknak fontos szerepük van a programozás és algoritmizálás oktatásában. Algoritmussal irányított animációk segítségével a tanulók könnyebben, rövidebb idő alatt, sokszor izgalmas, játékos formában érthetik meg a szemléltetett fogalmakat, folyamatokat és a köztük lévő különbségeket egyaránt [15].

Fontos azonban megemlíteni, hogy egy új animáció megtervezésénél, ill. fejlesztésénél figyelembe kell venni a multimédiával támogatott tananyag didaktikai elveinek betartását és a megfelelő grafikus reprezentáció megválasztását. Fontos, hogy megfelelő interaktivitást iktassunk az animációkba. A gondosan megtervezett, implementált és tesztelt interaktív szimulációk hasznos oktatási segédeszközként szolgálhatnak úgy a tanítók, mint a diákok számára.




**6. ábra:** A hátultesztelő ciklus szemléltetése

## Köszönetnyilvánítás

A cikk a KEGA 012TTU-4/2018 Interactive animation and simulation models in education (Interaktív animációs és szimulációs modellek az oktatásban) nemzeti projekt támogatásával készült.

This work has been supported by the Grant Agency of the Slovak republic KEGA under the Grant No. 012TTU-4/2018 Interactive animation and simulation models in education.

## Irodalom

1. Czakóová, K. 2016. *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
2. Czakóová, K.. 2019. *Interaktív modellek és szimulációk az oktatásban.* In. XXXII. Didmattech 2019. Trnava: Trnavská univerzita v Trnave, 2019
3. Czakóová, K.. 2019. Mathematical model based interactive simulations in education. ICERI 2019, 11-13. november 2019, Seville
4. Lapšanská, Š. 2016. *Visualize dinter active simulation models for teaching,* Bachelor thesis, Trnava University in Trnava, Faculty of Education, Department of ComputerScience. Supervisor of the bachelor thesis: prof. Ing. Veronika Stoffová, CSc. Trnava: Faculty of Education TU, 2016, 56 p
5. Pokorný, M. 2019. *Blended learning can improve the results of students in combinatorics and data processing.* In: ISET 2019. Los Alamitos: IEEE Computer Society, 2019, s. 207-210. ISBN 978-1-7281-3387-4
6. Pokorný, M. – Oubrechtová, S. 2016. *Blended Learning Can Improve the Results of Students in Mathematics.* In: Information, Communication and Education Application, Advances in Education Research, Volume 94, 2016, s. 57-62. ISBN 978-1-61275-505-2, ISSN 2160-1070

7. Szalai, D. 2017. *Algoritmikus gondolkodás és problémamegoldás fejlesztése interaktív játékok segítségével az alapiskolán.* Diplomamunka, Komárno: Selye János Egyetem, Tanárképző Kar, diplomamunka vezetője: PaedDr. Czakóová Krisztina, PhD. 2017, 62 p.
8. Stoffová, V. 2004. *Počítač – univerzálny didaktický prostridok.* Nitra: Univerzita Konštantína Filozofa, 2004. 172 s. ISBN 80-8050-765-1.
9. Stoffová, V. 2016. *The Importance of Didactic Computer Games in the Acquisition of New Knowledge.* In: The European Proceedings of Social & Behavioural Sciences EpSBS. 2016a, pp. 676-688. eISSN: 2357-1330. http://dx.doi.org/10.15405/epsbs.2016.11.70
10. Stoffová, V. 2016. *Počítačové hry a ich klasifikácia.* In: Trendy ve vzdělávání. 2016b, roč. 9, č. 1, s. 243-252.
11. Stoffová, V. - Horváth, R. 2017. *Didactic computer games in teaching and learning process Else Bucurest*, The 13th International Scientific Conference, eLearning and Software for Education, Bucharest, April pp. 27-28, 2017, 10.12753/2066-026X-17-000 DOI: 10.12753/2066-026X-17-000
12. Végh, L. 2016. *Interaktívne animácie vo vyučovaní algoritmov (Interactive animations in teaching and learning programming).* Edukacja – Technika – Informatyka (Education – Technology – Computer Science), 2016. 15(1): pp. 207-211.
13. Végh, L. 2006. *Vizualizácia algoritmov vo vyučovaní programovania.* Informatika v škole a v praxi. Ružomberok: Katolícka univerzita, 2006, s. 65-69. ISBN 80-8084-112-8
14. Végh, L. 2016. *Javascript library for developing interactive micro-level animations for teaching and learning algorithms on one-dimensional arrays.* Acta Didactica Napocensia, 9(2), 23–32.
15. Végh, L. 2017. *A programozás tanulásának és tanításának támogatása elektronikus tananyagba beépíthető interaktív animációs modellekkel.* (PhD theses). Eötvös Loránd Tudományegyetem Informatika Kar, 2017
16. 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

# Valósidejű C# grafika a böngészőben

Szabó Dávid[1], Dr. habil. Illés Zoltán[2]

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

**Absztrakt.** A legfrissebb .NET Core 3.0 futtatókörnyezet felhasználásával már nem csak webalkalmazásaink szerver oldalát, de akár a böngészőben végrehajtandó kliens oldalát is fejleszthetjük C# nyelven. Felhasználhatjuk a .NET világában elérhető könyvtárakat és névtereket, továbbá a Javascript funkcionalitáshoz is hozzáférhetünk. Kutatásaimban böngészőben futó valósidejű grafikai alkalmazásokkal vizsgálom az új technológia lehetőségeit.

**Kulcsszavak:** .NET Core, C#, Blazor, valós idő, grafika

## 1. Bevezetés

Webszerverek és webszolgáltatások fejlesztésében a mai napig jelentős szereplő az ASP .NET keretrendszer. A funkcionalitásban és könyvtárakban gazdag .NET világ és C# nyelv kombinációjával egyszerűen készíthetünk statikus, vagy komplexebb dinamikus weboldalakat is. Az ASP keretrendszer rengeteg konfigurációs terhet levesz a vállunkról. Átlátható navigációt és átirányítási lehetőségeket biztosít, gyorsan implementálhatunk authentikációt és adatbázis támogatást, illetve az egyszerű IT biztonsági eszközök támogatásával és beállításával segíti a szolgáltatásaink biztonságossá tételét. A Razor szintaxis [1] segítségével C# logikát ágyazhatunk a HTML lapokba, melyet a szerver a lap előállításakor hajt végre. Dinamikusan módosíthatjuk a HTML lapunk tartalmát, feltételeket és ciklusokat használhatunk HTML elemek létrehozásához és paraméterezéséhez. A több lapos weboldalakat és webszervereket az MVC architektúra segítségével szervezhetjük egy rendezett projektté, mely megkönnyíti kapcsolatok kiépítését a Razor HTML lapok és az alkalmazásunk üzleti modellje között.

Az elmúlt években a .NET és C# technológiák fő fejlesztési iránya a multi-platform eszközökké alakítás volt. Ennek az irányvonalnak a Microsoft által fejlesztett nyílt forráskódú .NET Core futtatókörnyezet [2] az egyik legjelentősebb eredménye, mellyel C# alkalmazásainkat eljuttathatjuk nem csak Windows, de Linux és MacOS környezetekre is (az Android és iOS támogatás is fejlesztés alatt áll). Tehát webalkalmazásaink már nem csak Windows rendszereken futhatnak. A további platformokon a hosszú múltra visszatekintő Mono futtatókörnyezetet használhatjuk, mely a mai modern operációs rendszerek mindegyikén képes futtatni C# alkalmazásainkat. A Mono nyílt forráskódú környezetet a Mono-Project vállalat fejleszti és céljuk, hogy nem csak asztali, de mobil és egyéb kompakt okos eszközökre is eljuttassa a C# nyelvet.

Láthatjuk, hogy az előbbi technológiák segítségével, célplatformtól függetlenül választhatjuk a C# nyelvet. Bármilyen mai platformra fejleszthetünk C#-ban… egyetlen kivétellel: Webalkalmazásainkat kiszolgáló szerverink készülhetnek C# nyelven, de a kliens oldalon, a felhasználó eszközén futó weboldalhoz tartozó kód kizárólag Javascript lehet. A webböngésző alkalmazások világában ugyanis a Javascript a sztenderd kliens oldali nyelv, melyet minden böngésző képes végrehajtani.

A legújabb .NET Core 3.0 keretrendszerrel és Blazor könyvtárral viszont ez megváltozott, mostantól a felhasználók eszközén a webböngészőben végrehajthatunk C#-ban készült alkalmazásokat!

# 2. C# a böngészőben

Korábban, ha a weboldalunkon dinamikus működést szeretnénk elérni, azaz miután a felhasználó eszközén betöltődött a lap további kódot és logikát szeretnénk végrehajtani a Javascript programozási nyelvet kellett használnunk. Ez az a nyelv melyet beágyazhatunk a HTML lapjainkba és minden webböngésző alkalmazás képes futtatni. Mindez igaz volt a 2017-ben bemutatott WebAssembly [3] megjelenéséig.

## 2.1 WebAssembly

A WebAssembly-t a World Wide Web Consortium együttműködésével a Mozilla, Microsoft, Google és Apple fejleszti. A WebAssembly (WASM) egy új típusú kód és programozási nyelv, melyet futtatni képesek a modern webböngészők. Ezt a nyelvet nem arra tervezték, hogy közvetlenül fejlesszenek benne alkalmazásokat. A WebAssembly nyelve inkább egy fordítási célnyelv más alacsony vagy akár magas szintű programozási nyelvek számára. A modern webböngészőkben dolgozó virtuális gép már nem csak Javascript kódot, hanem WASM kódot is képes kiszolgálni és végrehajtani. Tehát a WASM és Javascript párhuzamosan futhat a weblapunkon és a két kódbázis API-kon keresztül interaktálhat egymással. A WebAssembly nem a Javascript-et leváltó nyelvnek készül, hanem a lehetőségeinek kiegészítését segíti.

A WebAssembly célja, hogy a böngészőben futó alkalmazásaink egyes részeit Javascript-től különböző nyelven készíthessük el. Célszerű a WASM-hez fordulni, ha egy meglévő más nyelven íródott kódbázist szeretnénk felhasználni, vagy kifejezetten teljesítmény igényes feladatokat egy alacsonyabb szintű nyelven, szigorúbb memóriamodell mellett szeretnénk fejleszteni. Ezeket a modulokat írhatjuk C#, C/C++, vagy bármilyen nyelven, melyhez elérhető eszköz, amely WASM kódra fordítja a forrást. A lefordított WASM kódot pedig, mint egy könyvtárat felhasználhatjuk a weboldalunkon.

## 2.2 Blazor

A C# alkalmazásaink böngészőben futtatását is a WebAssembly nyelv teszi lehetővé további .NET technológiák kombinálásával, melyet összefoglaló néven Blazor-nek neveznek.

Az új .NET Core 3.0 keretrendszerben elérhető egy új projekttípus, a BlazorWASM projekt. A sablon egy .NET Core 3.0 projektet készít, melyben konfigurált egy ASP webszervert statikus fájlok kiszolgálásához. Egy webszervert kaptunk, melyet Windows, Linux és MacOS rendszereken is képesek vagyunk futtatni. A szerver jelenleg egyetlen index.html weboldalt szolgál ki, mely a BlazorWASM webalkalmazásunk. Az index.html által hivatkozott mono.wasm állományt is letöltik a kliens böngészők, mely a weboldalunkhoz tartozó a felhasználó eszközén futó C# kódunkat és a kiszolgálásához szükséges eszközöket tartalmazza. Kliens oldalon a Blazor a Mono futtatókörnyezetet [4] használja a C# kód kiszolgálásához és végrehajtásához. A Mono-Project implementálta a Mono futtatókörnyezetet a WebAssembly nyelvre. Tehát a böngészők virtuális gépében egy Mono futtatókörnyezetet dolgozik, melyben C# kódot hajthatunk végre.

Egy Blazor alkalmazás Komponensekből áll, melyeket a .razor kiterjesztésű állományok írnak le. Ezekkel a komponensekkel a HTML elemekhez hasonlóan fel tudjuk építeni a weboldalunk megjelenését. A sablon projekt tartalmaz komponenseket, például az App komponenst, mely a különböző kliens oldali Blazor lapok között tud navigálni. Míg a .NET Core szerver egy útvonalon egy Blazor webalkalmazást szolgál ki, addig ezen webalkalmazáson belül további al-lapok is lehetnek, melyeket az App komponens keres fel és jelenít meg az URL alapján. Ezek a Blazor lapok is komponensek, melyekben a @page kulcsszóval ellátva megadhatjuk, hogy milyen relatív URL címen szeretnénk elérni azt az adott lapot. Egy komponens nem csak egy lap lehet, akármilyen egyéb HTML elemeket vagy akár másik komponenseket is tartalmazhat. Ezzel szegmentálhatjuk a HTML lapjaink leírását különböző komponensek között.

A komponensekben a @ karakterrel C# kódot ágyazhatunk be, mely a kliens oldalon a felhasználó böngészőjében fog futni a Mono futtatókörnyezetben. A kód beágyazás a már korábbról ismert Razor szintaxist követi. Az elágazásokba helyezett HTML tag-ek csak akkor kerülnek a megjelent HTML kódba, ha az elágazás ágába fut a végrehajtás. A ciklusok minden iterációja példányosítja a ciklusmagba helyezett HTML tag-eket. Akár egy teljes, csak C# kódot tartalmazó kódblokkot is létrehozhatunk @code blokk elhelyezésével. Változókat és azok értékeit is felhasználhatjuk a HTML előállításakor. A különbség a hagyományos ASP.NET Razor és a Blazor között, hogy míg az előbbi csak egyszer, a szerveren értékelődik ki, mikor a HTML lapot előállítjuk, addig a Blazor képes a megjelenített HTML állományt futás közben alterálni a felhasználó eszközén.

A háttérben minden komponenst egy ComponentBase osztályból származtatott osztállyá fog alakítani a fordító. Az ősosztályon keresztül hozzáférhetünk a komponens életciklusának metódusaihoz, melyekkel kódot futtathatunk a komponens inicializálásakor (OnInitialized) vagy a HTML leírásba behelyezéskor (OnAfterRender). A @code blokkban felülbírálhatjuk ezeket a függvényeket, illetve paramétereket és tulajdonságokat is vihetünk fel a komponensünkhöz, melyeket kívülről elérhetnek más komponensek és módosíthatják értéküket.

A Blazor projekt létrehozásakor ügyeljünk a felhasznált projektsablonra. Több Hosztolási módot is választhatunk a Blazor projektekhez. Ez a cikk első sorban a BlazorWASM hosztolási módot mutatja be, mely a kliens oldalon C# kódot futtat a Mono keretrendszer segítségével. A szerver oldali .NET Core alkalmazás kihagyható, de ebben az esetben egy saját webszerverre lesz szükségünk, mely képes a statikus Blazor fájlokat kiszolgálni a kliensek számára. Elérhető a BlazorServer mód, melyben a komponensekbe írt C# kód továbbra is a szerveren hajtódik végre és a hívások eredményét a SignalR valósidejű kommunikációs könyvtár segítségével juttatja el a szerver a klienshez. A komponensek programozása és Blazor felhasználása ugyanúgy történik mindkét projekt típusban, viszont a háttérben a webalkalmazás működése jelentősen különbözik.

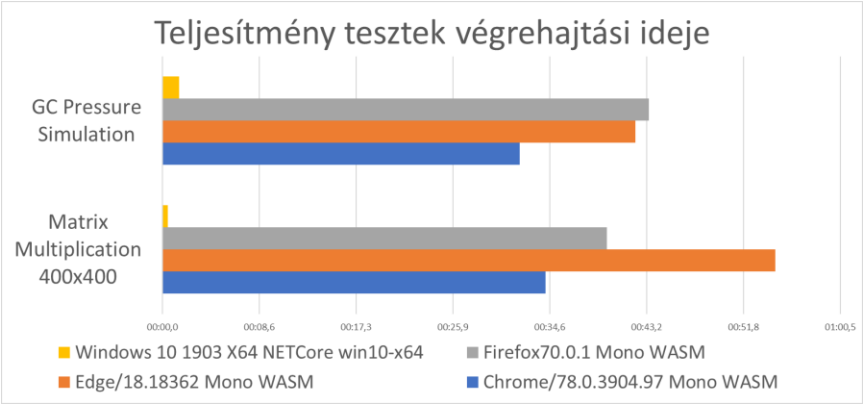
## 2.3 Javascript hívások

A webalkalmazások kliens oldali logikájában szükség lehet hozzáférni különböző Javascript API-khoz, ugyanis a webes világban a legtöbb webböngésző szolgáltatás és böngészőkben elérhető könyvtár Javascript API-kal kommunikál. Ahhoz, hogy ezeket az eszközöket használni tudjuk a Blazor rendszerben Javascript invokálást kell végrehajtanunk.

A C# kódból képesek vagyunk Javascript függvényeket meghívni, adatokat és paramétereket átadni ezeknek a függvényeknek, illetve a visszatérési értékület kiolvasni. Ehhez szükség van a komponensünkben a @inject IJSRuntime JSRuntime utasítással igényelni egy Javascript futtatókörnyezet implementációt. Ezen az implementáción keresztül tudunk a Javascript futtatókörnyezetben függvényeket hívni. A hívás során a Mono és Javascript virtuális gépek között adatkonverzió és szinkronizáció történik emiatt ezek a hívások jelentősen lassabbak, mint egy egyszerű függvényhívás. A hívások aszinkron függvények, ugyanis a Mono környezetben futó szál és a Javascript végrehajtási szála között szinkronizációra van szükség.

## 2.4 Teljesítmény

A kliens oldalon futó C# teljesítmény összehasonlítva a .NET Core 3.0 keretrendszer teljesítményével elég rossz eredményeket produkál. A tesztek során nagy méretű mátrixok szorzását, illetve a szemétgyűjtő terhelését végeztem string-ek és byte tömbök létrehozásával [5]. Jellemzően legalább 50x-es teljesítmény veszteséggel fut a böngészőben C# kód a .NET Core 3.0-hoz képest. Egyes tesztekben a különbség akár 100x-osra is növekedhet.




**1. ábra:** Teljesítmény tesztek futási idejének összehasonlítása

A mérésekhez használt számítógép konfiguráció: i7-2630QM, 16 Gb memória

# 3. WebGL

Kutatásaim során a .NET és C# nyelv lehetőségeit vizsgálom multi-platform valósidejű [6] grafikai megjelenítés témájában. A Blazor felhasználásával egy új platform és egyben új világ tárult ezen témakör opciói közé: a webböngészők és webalkalmazások világa.

Ha a web-en komplexebb valósidejű grafikai alkalmazásokat szeretnénk fejleszteni jellemzően a WebGL grafikus API vagy egy WebGL-re épülő könyvtár felhasználása közül választhatunk. A WebGL az OpenGL grafikus API mintájára készült grafikus processzorok alacsony szintű felhasználását lehetővé tévő interfész. Egy Javascript API-n keresztül hozzáférhetünk az OpenGL-ből ismert erőforrások webes változataihoz, adatokat küldhetünk a grafikus processzornak, shader programokat készíthetünk és futtathatunk, illetve ezek eredményeit megjeleníthetjük a böngészőben egy Canvas HTML elemben.

Az OpenGL-hez hasonlóan az állapotot a grafikus kontextus tárolja, illetve az állapotot módosító és rajzoló utasításokat a kontextuson értelmezzük. Ha WebGL-t szeretnénk megjelenítéshez használni, akkor először egy WebGL grafikus kontextusra van szükségünk, mely képes kezelni a grafikus erőforrásainkat és végrehajtani velük a rajzolási és megjelenítési parancsokat. OpenGL-ben egy grafikus kontextus egy végrehajtási szálhoz van kötve és ez határozza meg, hogy az adott szálon végrehajtott utasítások, melyik kontextusban hajtódnak végre. WebGL esetén az API biztosít egy WebGLContext objektumot és ezen osztályon értelmezett metódusokkal tudjuk a kontextus állapotát módosítani.

A kontextust a Canvas elemen keresztül tudjuk létrehozni. Az API automatikusan konfigurálja a megjelenítés platformfüggő részeit (a SwapChain erőforrást, melyet a WebGL el is rejt a fejlesztők elől).

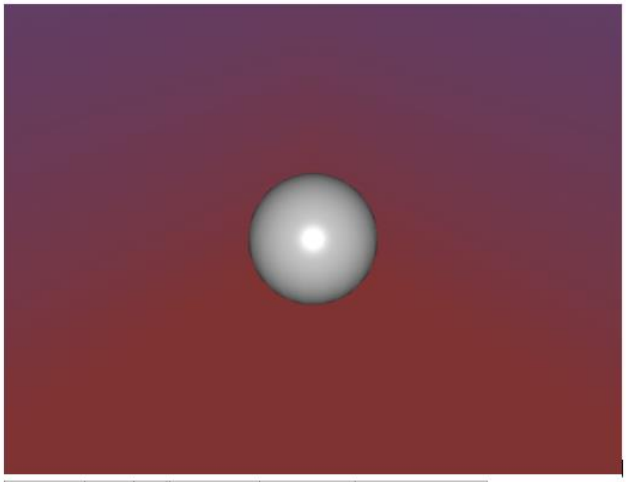
A Blazor segítségével a Canvas grafikák és WebGL elérhetővé vált a C# fejlesztők számára is. A BlazorExtensions/Canvas NuGet csomag felhasználásával egy átlátható C# API-t kapunk a HTML Canvas eleme köré, mellyel elérhetjük a Canvas rajzoló függvényeit és a WebGL grafikus API-t [7].

## 3.1 Blazor és WebGL

A BlazorCanvas könyvtáron keresztül a WebGL API első verzióját érhetjük el. Ez a legtöbb böngészőben támogatott, bár funkcionalitása felett eljárt az idő. A modernebb és gyorsabb grafikus eszközök eléréséhez használható WebGL 2 egyelőre nem támogatott a könyvtárban (ahogy minden webböngészőben sem).

Az API első látásra kényelmes WebGL osztályok és rajtuk értelmezett metódusok formájában tárja elénk a grafikus lehetőségeket. A WebGL Javascript erőforrásaihoz (Context, Buffer, Texture, stb.) kapunk C# wrapper osztályokat. A WebGLContext példányhoz hozzájuthatunk a Canvas komponenstől, mely inicializálja nekünk a grafikus kontextust minden a weboldal felületén megjelenítéshez szükséges funkciójával. A kontextus függvényei C# primitív típusokat, definiált enumerátorokat, illetve WebGL erőforrás osztálypéldányokat várnak paraméterben, így a legtöbb függvény kényelmesen és intuitívan használható.

Sajnos ez nem minden függvényre igaz. A buffer-ek feltöltése és uniform változók értékeinek megadása float, integer vagy egyéb primitív számtípusok tömbjeinek átadásával történik. Tehát, ha ezeket a változókat a System.Numerics névtér típusaiban tároljuk (Vector3, Matrix4x4 stb.) vagy akár komplexebb saját struktúrában [8], akkor minden WebGL hívásnál konvertálnunk kell az adatokat tömbökké, mely sűrűn fog szemétgyűjtést előidézni. Másik lehetőségünk, hogy kényelmetlenül tömbök formájában kell kezeljük az adatainkat. A WebGL Javascript API is tömbök formájában várja ezeket a paramétereket, ezért ez a konverzió elkerülhetetlennek tűnik.



| Time | Frame | FPS | Frame Time | Update Time | Render CPU Time |
|---|---|---|---|---|---|
| 00:09.7908 | 281 | 32,7 | 00.0316 | 00.0001 | 00.0286 |

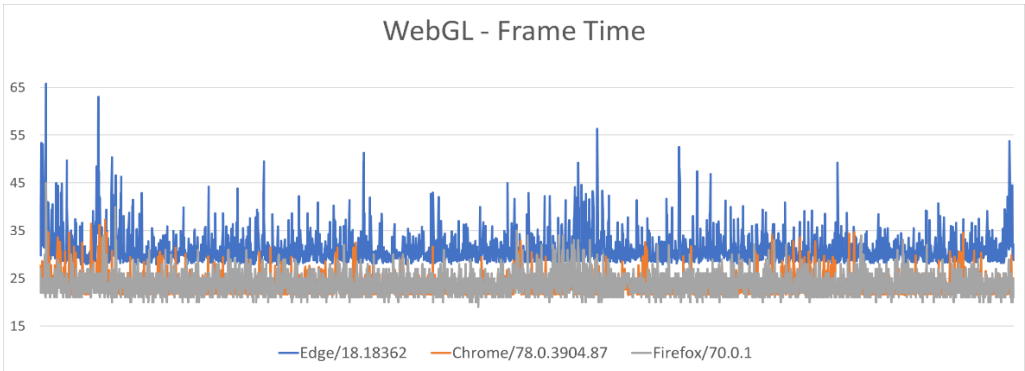
**2. ábra:** Árnyalt gömb és procedurális Skybox Blazor WebGL-ben

A .NET Standard közös kódbázist támogató lehetőségeinek köszönhetően a korábbi OpenGL grafikai alkalmazásaimhoz készített kódot kényelmesen fel tudtam használni a Blazor WebGL projektekben is. Több C# algoritmus és adatszerkezet melyeket korábban már elkészítettem néhány kattintást követően felhasználhatóvá váltak a böngészőben futó webalkalmazásban. Blazor nélkül ezt a kódbázist teljesen újra kellett volna írnom Javascript-ben.

## 3.2 Valósidejű Grafika?

Az egyébként is az elvártnál lassabb kliens oldali sebességet tovább súlyosbítja a sűrű Javascript invokáció. Az összes WebGL hívást továbbítani kell a Javascript futtatókörnyezetnek ugyanis ezeket a hívásokat mind a Canvas DOM elemen és a Javascript WebGLContext példányon kell végrehajtani. Így egy egyszerű WebGL parancs végrehajtása modern processzorokon is, akár 2-3 milliszekundumot is igénybe vehet, amely a valósidejű grafikában a másodpercenként 60 képkocka előállításához elfogadhatatlan. A WebGL-hez hasonló alacsony szintű grafikus API-nkak célja, hogy nagy részletességgel tudjuk konfigurálni a grafikus szerelőszalag adatait és működését. Ha minden erőforrást megfelelően előkészítünk, akár 15-20 WebGL utasítás (30-60 milliszekundum) is szükséges lehet egy olyan egyszerű grafika megjelenítése, mint egy 3 dimenziós gömb. Ez azt jelenti, hogy a jelenlegi

lehetőségeinket tekintve ez a technológia még nem alkalmas valósidejű grafika megjelenítésére. A Javascript invokációk csökkentése érdekében a könyvtárban lehetőségünk van csoportosítani a WebGL utasításainkat és egyszerre több utasítást leküldeni a Javascript futtatókörnyezetnek, de a két környezet közötti szinkronizáció, illetve a hívások szerializálása és deszerializálása ugyanúgy értékes feldolgozási időbe kerül. Az alábbi diagrammon láthatjuk a gömb megjelenítésének 2 perces futása során az egyes képkockák előállításához szükséges időt milliszekundumban. Láthatjuk, hogy ez az idő végig 16 milliszekundum felett áll, tehát a másodpercenként kevesebb, mint 60 képkockát állítunk elő.




**3. ábra:** Blazor WebGL képkocka előállítási idő milliszekundumban

Amint a WebGL hívások eljutottak a Javascript futtatókörnyezetbe a grafikus processzor által végzett munkák már nem szenvednek teljesítmény veszteséget. Tehát, ha nagy adatszerkezeteket dolgozunk fel vagy hosszú shader programokat kell futtatnunk és csak ritkán például felhasználói adatmódosítás esetén szükséges frissíteni a grafikát, akkor mindezt megtehetjük anélkül, hogy a C# nyelvet elhagynánk. Amennyiben valós időben folyamatosan frissülő grafikát és megadott másodpercenkénti képkockaszámot szeretnénk elérni a Blazor keretrendszer és Blazor Canvas könyvtár még nem áll készen a feladatra.

## 4. Összegzés

A Blazor segítségével egyszerűen juttathatunk C# kódot a webböngészőkben futó alkalmazásokba. A Razor szintaxissal kiegészített HTML lapok képesek dinamikusan változtatni a tartalmukat és frissíteni a megjelenített információkat. Korábbi alkalmazásaink C# kódbázisát felhasználhatjuk a böngészőkben, ezzel drasztikusan rövidítve a webalkalmazások fejlesztési idejét.

A technológia fiatalsága révén a futási teljesítmény nagyságrendekkel lassabb, mint egy hagyományos alkalmazásé. Ebből kifolyólag jelenleg csak rugalmasabb puha valósidejű határidőkkel rendelkező alkalmazások fejlesztésére alkalmas a Blazor. Szigorú határidők [9] vagy rövid időkorlátok tartása egyelőre nem konzisztens a böngészőben futtatható C# kód segítségével.

## 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. Adam Freeman: *Pro ASP.NET Core MVC*, Apress, 2016, [101-122], ISBN: 978-1-4842-0397-2
2. Andrew Troelsen, Philip Japikse: *Pro C# 7: With .NET and .NET Core*, Apress, 2017, [1245-1251], ISBN: 978-1-4842-3017-6
3. *WebAssembly*, 2019, https://webassembly.org/ (utoljára megtekintve: 2019.11.06.)
4. Laurent Sansonetti: *Mono and WebAssembly - Updates on Static Compilation*, 2018, https://www.mono-project.com/news/2018/01/16/mono-static-webassembly-compilation/ (utoljára megtekintve: 2019.11.06.)
5. Szabó Dávid: *C# Multi-Platform Környezetben*, MSc Diplomamunka, 2018, [58-59] https://edit.elte.hu/xmlui/bitstream/handle/10831/38929/Diplomamunka.pdf (utoljára megtekintve: 2019.11.06.)
6. Dávid Szabó, Dr. Zoltán Illés, Viktória B. Heizlerné: *Valós idejű funkcionalitás Windows-ban*, INFODIDACT 2018, [263-264], ISBN: 978-615-80608-2-0
7. *Blazor Canvas repository on GitHub*, 2019, Letöltve 2019. november 06., https://github.com/BlazorExtensions/Canvas
8. Illés Zoltán: *Programozás C# nyelven*, Jedlik Oktatási Stúdió 2005, [51-54], ISBN: 963-86514-1-5
9. Dr. Illés Zoltán, Heizlerné Bakonyi Viktória, Illés Zoltán: *Valós időben, valós világban*. INFODIDACT 2015, Zamárdi (2015), ISBN: 978-963-12-3892-1

# Módszertani ötletek egyetemi kurzusok és az újabb hallgatói generációk elvárásainak összehangolására

Széll Réka[1], Holló Csaba[2]

[1]szll.reka96@gmail.com
SZTE TTIK
[2]chollo@inf.u-szeged.hu
SZTE TTIK Informatikai Intézet

**Absztrakt.** Az egyetemi oktatási tevékenység során tapasztalható, hogy a hallgatók elvárásai, tanulási attitűdjei az utóbbi években jelentősen megváltoztak és folyamatosan változnak. A hatékony oktatás érdekében szükséges ezeket a változásokat megérteni, és az oktatás szervezését és módszertanát ennek megfelelően alakítani. A cikkben elemezni fogjuk a változások fontosabb jellegzetességeit, és lehetséges megoldásokat, módszertani ötleteket keresünk, melyek sikeresebbé tehetik az egyetemi oktatási tevékenységet.

**Kulcsszavak:** Z generáció, oktatásszervezés, módszerek.

## 1. Bevezetés

Az egyetemisták egyre nagyobb része a Z generációhoz tartozik, melybe a 1996-2010 között született fiatalokat soroljuk. természetesen ez ne jelenti azt, hogy minden Z generációs egyforma lenne, de az a gazdasági, szociális és digitális környezet, amelyben felnőttek, lényegesen maghatározza a szemléletmódjukat és hozzáállásukat a tanulás esetében is. A Z generációs jellegzetességekről a [14] cikkben részletesen olvashatunk, itt többnyire csak hivatkozni fogunk az abban leírtakra. Cikkünkben arra próbálunk megoldásokat találni, hogy hogyan lehet az egyetemi oktatást közelebb vinni a Z generációs igényekhez. Az iskolák már korábban szembesültek ezzel a kérdéssel, és kidolgoztak jó és átvehető módszereket is, ugyanakkor az egyetemi képzésnek vannak az iskolai képzéstől eltérő jellegzetességei is, melyekre úgyszintén megoldásokat kell találni.

Míg az iskolai oktatás esetén egyértelmű, hogy azokban gyerekek vesznek részt, hagyományos módon az egyetemistákról azt feltételeznénk, hogy felelős felnőtt emberek, akik első sorban tanulni jönnek az egyetemre. A mindennapi gyakorlat azonban azt mutatja, hogy sokan nem rendelkeznek az önszabályozás megfelelő szintjével, továbbá a hallgatók számának jelentős növekedése következtében sokkal gyengébb hallgatók is bekerülnek az egyetemre. Ennek következtében vannak olyan hallgatók is, akik nem csak tanulással akarnak (jobb) jegyet szerezni. Az nem egyértelmű, hogy az oktatói kapacitásból mennyit érdemes a csalások kiszűrésére áldozni ahelyett, hogy azt a kurzus fejlesztésére és a tényleg tanulni vágyó fiatalok segítésére fordítanánk. Valószínűleg egy kompromisszumos megoldás az, ami elfogadható, de a módszerek alkalmazásánál ezeket a szempontokat is figyelembe kell venni.

A Z generációs jellegzetességek között találjuk azt is, hogy a hallgató csak akkor hajlandó valamit megtanulni, ha annak hasznát látja, nincs türelme hosszú időn keresztül egyetlen dologra figyelni, igényli a folyamatos tevékenységet és másokkal való együttműködést. Ezek fényében nem meglepő, hogy a hallgatók a gyakorlatokat általában szívesebben látogatják, mint az előadásokat [14], ezért a cikkben kiemelt figyelmet fordítunk arra, hogy az előadásokat hogyan lehetne vonzóbbá tenni.

## 2. Hallgatói visszajelzések figyelembe vétele

A digitális bennszülöttek számára fontos, hogy elmondhassák véleményüket, hogy érezzék, hogy bevonjuk őket a tervezés folyamatába, és hogy számít a véleményük. Mégis azt tapasztaljuk, hogy mikor lehetőségük adódik hivatalos formában a kurzus értékelésére, nem élnek vele, és rejtett csatornákon adják át egymásnak a tapasztalatokat. Ily módon pedig nem jut el az oktatókhoz az információ, nem tudják esetlegesen ezekhez igazítani az óráikat, mert nincsenek tisztában azzal, hogy mi az, amit a hallgatók problémának éreznek.

Érdemes célként kitűzni, hogy megértessük, és elmagyarázzuk a hallgatóknak, hogy fontos a véleményük, és csak úgy tudjuk ezeket figyelembe venni, ha megosztják velünk. A lehetőség mindenki számára adott, hogy leírja a véleményét, viszont a többség nem él vele, így más módszert is érdemes alkalmazni. Rászánhatunk a félév elején és a félév végén valamennyi időt arra, hogy az órán – természetesen névtelenül – kitöltsenek értékelői lapokat. A félév elején rákérdezhetünk az elvárásaikra, hallott információkra az oktatóval kapcsolatban, így beleláthatunk, hogy milyen elvárásokkal, félelmekkel jönnek az óránkra. A félév végén pedig konkrét kérdések segítségével megtudhatjuk, hogy mi az, amin változtatnának, szerintük mi működött jól, és mi kevésbé.

## 3. Előadások és gyakorlatok szervezése

A hallgatói visszajelzésekből tudni lehet, hogy a hallgatók általában kevesebb előadást és több gyakorlatot szeretnének. Annak, hogy az előadásoknak kevésbé látják értelmét, oka lehet az is, hogy otthon saját sebességüknek megfelelően tudnak tanulni, míg az előadás sebessége egyeseknek túl gyors, másoknak túl lassú (ezért unalmas) lehet. A gyakorlatok unalmasságát csökkenti, hogy több aktivitást igényelnek, ugyanakkor általában a hallgatók a gyakorlatban jobban látják a tanultak alkalmazhatóságát, hasznosságát is. Ahhoz azonban, hogy kevesebb előadás és több gyakorlat legyen, számos problémát kell megoldani.

Egyrészt, a tantárgyak nagy részében elmélet nélkül nincs, amihez gyakorlatot végezni, tehát elmélet átadására szükség van, kérdés, hogy ezt az elméletet a hallgatók hogyan tanulják meg. Amint más fejezetekben is tárgyalni fogjuk, meggondolandó az a megoldás, hogy a tananyag egy részét a hallgatók önállóan megtanulják, és az előadást inkább a hallgatók által előzetesen már ismert tananyag jobb megértésére, a felmerülő kérdések megbeszélésére fordítsuk, hogyha pedig ez kevesebb időt igényel, akkor csökkenteni lehetne a kontaktórák számát, ami szükséges is akkor, ha elvárjuk, hogy ugyanannyi kreditért a hallgatók önállóan többet dolgozzanak. Természetesen ezekben jelentős különbségek lehetnek az egyes tantárgyak között. Megjegyzendő továbbá, hogy a rövidebb előadás nem feltétlen jelent kevesebb készülést az oktató számára, mivel egyrészt meg kell szerveznie azt, hogy a hallgatók az egyes alkalmakra mit olvassanak el otthon (esetleg tananyagot is kell ehhez fejlesztenie), másrészt a más jellegű előadás módszertanilag is több felkészülést igényelhet. Viszont, ily módon csökkenne az infrastruktúra használat, a hallgatók pedig így valószínűleg szívesebben tanulnának, és – ha hajlandók a kiadott tananyagot feldolgozni, akkor – az anyagot is jobban elsajátítanák.

Egy másik megoldás lehetne, hogy az előadás egy részét gyakorlatiasabban tartjuk, akár úgy is, hogy a hallgatók az előadás alatt több oktató bevonásával csoportmunkákat is végeznek ([14]). Viszont, egy előadóteremben nem mindig lehet ennek a technikai feltételeit biztosítani, továbbá, ha már úgyis több oktatót használunk, akkor felmerül az, hogy az előadások óraszámának egy részét esetleg átcsoportosíthatnánk gyakorlatokra.

Lehetne-e növelni a gyakorlatok mennyiségét? Tanrend szempontjából az előadásokból csoportosíthatunk át krediteket a gyakorlatokhoz. Viszont, egy 500 fős kurzus esetén heti 1 óra átcsoportosítása 25 fős termek esetén újabb 20 darab heti 1 órás gyakorlati csoportot jelentene, kérdés, hogy van-e ehhez elegendő infrastruktúra (terem és oktató)? És egyáltalán kell-e? Hogyha a hallgató az előadás anyagnak szívesebben utánanéz, mint hogy meghallgassa, akkor elvárható, hogy a gyakorlást

is többnyire önállóan tegye meg, és a gyakorlat is inkább a felmerülő problémák megbeszéléséről szóljon.

## 4. Tanulási eredmény alapú kurzusleírás készítése

A Z generációs fiatalok lehetőség szerint azonnali hasznos várnak. Jelen esetben tudni akarják, hogy miért érdemes számukra a kurzust megtanulni, illetve az azzal kapcsolatosan elvárt cselekvéseket elvégezni.

Az eredményalapú kurzusleírás – amennyiben ezt nem csak formálisan, látszólag tesszük meg –, rákényszerít arra, hogy ahelyett, hogy a sok éves tematikát simán lemásolnánk, a hallgató szemszögéből gondoljuk át a kurzusunkat: biztos-e, hogy 2019-ben is az a fontosa kurzusban, ami korábban volt, mit szeretnénk elérni, milyen előzetes tudásra és milyen hallgatói képességekre számíthatunk, és ezek figyelembe vételével hogyan tudunk minél eredményesebbek lenni? A tanulási eredmény alapú kurzusleírással kapcsolatosan további ötleteket a [9] cikkben olvashatunk.

## 5. Előadások érdekesebbé tétele

Ebben a fejezetben csak néhány kisebb ötletet mutatunk be az előadások érdekesebbé tételére, további ötletek részletesebben lesznek kifejtve a következő fejezetekben.

A hallgatók figyelmének fenntartásához hozzájárulhat, hogyha az oktató előadásmódja humoros, ugyanakkor a humorral óvatosan kell bánni, nehogy valakit megbántsunk. A megértést segíti, ha az előadás jól strukturált, a tananyag megjegyzését pedig az, ha az elhangzottak érzelmekhez kapcsolódnak, ezért jó, ha az előadásba érzelmi töltetet, szenvedélyt is bele tudunk vinni [13].

Az előadás vonzóbb lehet akkor, hogyha sikerül beleépíteni kevésbé közismert vagy meglepő tartalmakat, hiszen ez felkelheti az érdeklődést és változatosságot kölcsönöz az órának. Annak is hasonló hatása lehet, ha sikerül kapcsolatot teremteni különböző jellegű, vagy nem nyilvánvaló tartalmak között, ráadásul megtörténhet, hogy olyan hallgatók, akik előzetesen kedvelik a távoli kapcsolódó témakört, ezen kapcsolat miatt fokozottan érdeklődni fogna a mi tárgyunk iránt is.

A digitális bennszülöttek generációjának sajátos tulajdonsága a türelmetlenség, hogy mindent azonnal tudni akarnak, és nehezen fogadják el, hogy valamely sikerekre várniuk kell. A multitasking működéssel, és az információk folyamatos áramlásával egyetlen dologra már sokkal kevésbé, és kevesebb ideig tudnak koncentrálni, mint a korábbi generációk, így a hagyományos órákon sokkal hamarabb unatkozni kezdenek. Az internet adta lehetőségekkel élve, a honlapok közötti folyamatos váltásokkal, és az információk több helyről való gyűjtésével a hosszabb tartalmak olvasásában és az írásában kevésbé tudnak elmélyülni [3]. Az, hogy egy előadáson az írott szöveget kivetítjük, így látják azt, és közben hallják, amit mondunk, még nem jelent nekik akkora információáramlást, hogy rövid idő (7-20 perc) után ne veszítsük el a figyelmüket. Időnként érdemes bevetni olyan segítségeket, melyek kizökkentik a hallgatókat az egyhangúságból.

Az animációk használata nagyban segítségünkre lehet, kevésbé kell az írott szövegre koncentrálni, sokkal nagyobb hangsúlyt kap a mozgás, a hang, a képi információ. Hamarabb jutnak a megértés szakaszába, mint egy magyarázó szöveg segítségével, így a türelmetlenség nem üti fel olyan hamar a fejét, és közben jobban ki tudják használni a képességeiket, hogy egyszerre több mindenre kell figyelniük. Továbbá, egy kreatívan és jól elkészített animáció alkalmazása szórakoztatóvá is tudja tenni a tananyag elsajátítását. Az animációk használatának hátrányai, hogy elkészítésük nagyon időigényes, sok befektetett órát és kreativitást igényel.

# 6. A designgondolkodás használata

A designgondolkodás módszertanának oktatásban történő alkalmazása egy viszonylag új terület, ezzel együtt hasznos lehet tanárként ezzel a látásmóddal is végiggondolni a kurzusban rejlő lehetőségeket [25, 21].

A designgondolkodás szerint a design azt a szisztematikus folyamatot jelenti, ami szerint megtervezünk valamit, és nem pedig azt, hogy valami formailag jól néz ki. A módszer a hallgatót helyezi középpontba, akiről azt mondja, hogy akkor lesz motivált, hogyha a saját maga által felismert problémával foglalkozhat. Hogyha viszont ezt tanítással akarjuk összekötni, akkor szükséges, hogy a hallgató meg tudja határozni az őt foglalkoztató és ugyanakkor szakmailag releváns, a tanulási célok szempontjából is fontos problémát. Érdemes a tanulást a való világ létező problémáihoz kötni, ráadásul úgy, hogy a problémák megoldásában a tanulók lehetőleg együttműködjenek, és megtapasztalhassák, hogy a képességeik hogyan egészítik ki egymást. Annak elérése, hogy a hallgató minderre képessé váljon, a tanár részéről nagyon sok előkészítő és támogató munkát igényel, a klasszikus frontális előadó szerepkör mellett vagy helyett. Ennek a megvalósítása inkább kisebb létszámú (gyakorlati) csoportokban lehetséges.

# 7. Hallgatók közötti együttműködés támogatása

Manapság a munkaerőpiac elvárja azt, hogy a végzettek alkalmasak legyenek csapatmunkára, ugyanakkor az újabb generációk is igénylik az együttműködést. Nyilvánvaló, hogy az is hasznos, hogyha a hallgatók segítenek egymásnak. Mindezek miatt érdemes átgondolni annak a lehetőségeit, hogy hogyan tudjuk az adott kurzus esetében a hallgatók közötti együttműködést támogatni.

## 7.1. Állandó hallgatói csoportok szervezése

A bolognai rendszer bevezetése a hallgatók számára oktatási tevékenységük szabadabb megszervezését tette lehetővé, viszont ennek a szabadságnak ára van. Azon túl, hogy a hallgatóknak több önfegyelemre lenne szükségük ahhoz, hogy a szabadsággal megfelelően éljenek, egyrészt a választási lehetőségek bővülésével, másrészt azáltal, hogy kizárólag a nem teljesített tárgyakat kell újból elvégezni, megszűntek az állandó csoportok, melyekben a hallgatók jobban ismerték és segítették egymást. Látszik az a tendencia, hogy az egyetemek igyekeznek újból bevezetni az állandó csoportokat, de nyilván ennek a lehetőségei korlátozottak.

## 7.2. A projektmódszer alkalmazása

Az együttműködés nagyobb méretű megvalósítására alkalmazhatjuk a projektmódszert. A projektoktatás lényege, hogy egy nagyobb, összetettebb téma, esetleg feladat feldolgozását/megvalósítását teljes egészében a hallgatóknak kell elvégezniük, esetlegesen a téma megválasztásától a tényleges reprezentálásig. Ez történhet egyéni és páros munkában is, azonban sokkal célravezetőbb, ha 4-5 fős csoportokra bízzuk a feladat elvégzését [1]. Fontos kiemelni, hogy a projektmódszer jó eszközként szolgál, hogy ösztönözzük a hallgatókat az önálló munkavégzésre, a kreativitásuk kiélésére, az ötleteik megvalósítására.

A projektmódszer fontos elemei a komplex feladat részekre bontása, a részek hallgatók közötti szétosztása, a határidők meghatározása, a teljes munkamenet megtervezése, az elkészült produktum bemutatása. Lényeges, hogy ezek elvégzése a csoport feladata, a tanár csupán segítőként, mentorként van jelen a munka során, és a csoportnak önállóan kell dolgoznia.

Bár a projektmunka egy jól működő módszer, ha egyszerűen kiadjuk a feladatokat egy olyan csoportnak, akik még sosem működtek együtt másokkal, nem dolgoztak projekten, nem valószínű, hogy igazán eredményes lesz, ugyanis meg kell a hallgatókat tanítani arra, hogy hogyan kell csapatban dolgozni, hogyan kell felépíteni egy projektet [7]. Gyakorlati példán keresztül érdemes elmagya-

rázni, hogy milyen lépéseken haladjanak végig, segíteni őket abban, hogy kiosszák a megfelelő szerepeket, figyelni kell arra, hogy mindenkinek legyen feladata, méghozzá olyan, mellyel a csoportban hasznos taggá tud válni.

Előre közölni kell a hallgatókkal, hogy a beadáskor milyen dokumentumokat várunk el, mik a tényleges formai követelmények. Fontos, hogy jól specifikált feladatokat kapjanak, hogy már az elejétől fogva tudják, mi az, amit elvárunk tőlük. Meg kell mondanunk nekik, hogy hogyan értékeljük őket, ehhez érdemes figyelembe vennünk a következő fejezetben leírtakat.

## 7.3. Csoportmunka

A csoportmunka azt jelenti, hogy a hallgatók valamilyen módon együtt dolgoznak egy feladat megoldásán.

A csoportmunka szigorú megközelítése értelmében a feladatnak olyannak kell lennie, hogy azt kevesebben ne tudják elvégezni. Viszont, minél kevésbé tudjuk ellenőrizni a hallgatók tevékenységét, annál nehezebb ilyen feladatot adni. Hogyha a hallgatók az órán dolgoznak, akkor megfelelő időkorláttal a feladatok többségénél biztosítható, hogy azt ne tudják egymás helyett elvégezni. Otthoni feladatként kiadott projektek esetén erre már kevés lehetőségünk van, többnyire azt tudjuk tenni, hogy a teljes feladatot értékeljük, a kapott pontszám elosztását a résztvevőktől kérjük, mindenkinek nyilatkoznia kell arról, hogy a pontszámelosztás arányában mely rész elkészítésében vett részt, és a végén megpróbálunk meggyőződni arról, hogy azt megfelelően érti is (ami nem feltétlen jelenti azt, hogy ő csinálta, de ennél többet nem tudunk ellenőrizni). Úgy gondoljuk, hogy ha lesznek is olyanok, akiknek valamilyen mértékben sikerül kijátszaniuk a rendszert, összességében nagyobb haszon az, hogy akik együttműködés terén is fejlődni akarnak, annak lehetőségük van rá. A tanárok részéről plusz befektetést igényel az, hogy a hallgatóknak meg kell tanítani az együttműködés módszertanát [7], és a végén mindenki esetében el kell végezni a megértés tesztelését, ami nem feltétlen automatizálható.

## 7.4. Új helyzetek teremtése

Ebben az esetben nem egy külön módszerről van szó, hanem egy olyan szempontról, amit a csoportmunka, illetve projektmódszer alkalmazásánál érdemes figyelembe venni.

A hálózatkutatás eredményei szerint az új megoldandó helyzetek összetartóbb csoportokat hoznak létre, hidakat képeznek távoli nagyobb (fontosabb, népszerűbb) csomópontok (és ezek által csoportok) között, és ami a legfontosabb, hogy a hálózat az új helyzetekből sokkal hatékonyabban tanul [10]. Ugyanakkor, az új helyzetek teremtése és megoldása új ötleteket és alapos szervezést igényel.

# 8. Aktivizálás

A különböző tanulási stílusú hallgatók különböző módszereket igényelnének. A legtöbb tanuló – a Z generációsok még nagyobb mértékben – a vizuális kategóriába sorolható, és sokszor hajlamosak vagyunk főként rájuk koncentrálni. A vizuális hallgatók számára a vizuális információk a mérvadók, tehát a legkönnyebben a prezentációkból, ábrákból, grafikonokból, képekből tanulnak. Az auditív hallgatók az információkat hallás alapján dolgozzák fel, így számukra az órán való figyelés, a beszélgetésben való részvétel, az instrukciókra való figyelés a hasznosabb. Érdemes őket ösztönözni arra, hogy felvállaljanak egyféle mentor szerepet, ugyanis rájuk van legnagyobb hatással a másnak történő magyarázat, ők tanulnak a legtöbbet belőle. Az írott és hallott szövegeket gyorsan tudják feldolgozni. Az egyetemen a kinesztétikus vagy mozgásos tanulási stílusú hallgatókra kevésbé figyelünk. Számukra fontos a mozgás, az érintés, a fizikai tárgyakkal való foglalkozás. Nehezükre esik egyhelyben ülni, és úgy koncentrálni [2].

Az előadásokat érdemes úgy felépíteni, hogy minden típusú hallgatónak kedvezzünk, ugyanis senki nem esik egyértelműen csak az egyik kategóriába. Mindenki tartalmaz stílusjegyeket az összes tanulási típusból. Fontos lenne, hogy a hallgatók felkelhessenek, mozoghassanak kicsit az előadás közben, különben gyorsan elveszítjük a figyelmüket. A képek, animációk, grafikonok használata kiemelten fontos, ugyanis a digitális bennszülöttek sokkal inkább tanulnak a vizuális elemekből, mint szövegekből.

Érdemes az előadásra nem csak úgy gondolni, mely során feltétlenül csak a tanárnak szükséges beszélnie. Lehet audiovizuális elemeket, hangfájlokat, videókat használni. Strukturálhatjuk úgy az órát, hogy a hallgatók valóban dolgozzanak, meghatározhatunk kérdéseket, szempontokat melyek segítenek a hallott/látott anyag feldolgozásában [4]. És ezzel felmerül egy kérdés, melyen érdemes elgondolkodni. Ki az aktív az órán? Ha a tanár beszél, ő az, aki magyaráz, akkor ő az aktív résztvevője az órának, a hallgatók pedig a passzívak, ők a legjobb esetben jegyzetelnek, rosszabban pedig csak fizikailag vannak jelen. Az aktív tanulás nagyon lényeges lenne, ahhoz viszont meg kell fordítanunk, legalábbis ki kell egyenlítenünk az aktivitás és a passzivitás arányait. Az előadásba beépíthetjük az irányított megbeszélés módszerét, mellyel tulajdonképpen lebonthatjuk az egyes témakörök feldolgozását kérdésekre és válaszokra, ezzel fokozva a tanulói aktivitást az órán [15]. Az irányított megbeszéléssel nem mondjuk ki az új információkat, hanem kimondatjuk azokat a hallgatókkal, kérdéseinkkel irányítva.

A Z generációs hallgatók ahhoz szoktak hozzá, hogy az interneten mindent kimondhatnak rögtön, azonnal kapnak információt, egyik linkről a másikra ugrálhatnak, és közben véleményt nyilváníthatnak [4]. Érdemes ezt az oktatásban is beépíteni, teret hagyni nekik arra, hogy kimondják azokat a gondolatokat, melyek megfogalmazódnak a tanórán.

## 9. Szavazás

A [14] szerint a hallgatók leginkább azért nem járnak előadásokra, mert unalmasnak tartják, mivel igénylik az interaktivitást, azonnali visszajelzéseket, jutalmakat, melyeket más – első sorban online – tevékenységeikben megszoktak.

Egy további lehetőség a hallgatók aktivizálására, akár nagyobb előadások esetén is, a szavazás használata. Ebben az esetben a hallgatóknak időnként kérdéseket teszünk fel, melyekre válaszolhatnak. A válaszok begyűjtése után megmutathatjuk a helyes választ, elemezhetjük a potenciális hibákat, ily módon megtörténik a visszajelzés és esetleges jutalmazás is, és mivel a hallgatók így jobban figyelnek, az információátadás is hatékonyabb.

Az online szavazások lebonyolítására számos program alkalmas (CooSpace, Kahoot!, Socrative stb.), melyekben a tesztek elkészíthetők, a program a válaszokat kiértékeli, majd az eredmények exportálhatók. Mivel nagy előadásokra nem jellemző, hogy azokat gépteremben lehetne tartani, ezért a hallgatók saját eszközeire (jellemzően okostelefonokra) lehet építeni. A mobil eszközök használata azonban szükségessé teszi megfelelő minőségű WIFI biztosítását, ami jelenleg még nem megoldott, csak remélni lehet, hogy ha egyre többen felismerik ennek a fontosságát, és akkor javulni fog a helyzet. Jelenleg ez a lehetőség tehát jellemzően úgy használható, hogy a hallgatók saját mobil internetüket használják, aminek a használatát elvárni nem lehet, ezért a szavazást csak opcionális plusz lehetőségként lehet használni, még akkor is, ha ezek a válaszolások kevés adatforgalmat bonyolítanak.

Összességében a tapasztalat azt mutatja, hogy a szavazás lehetősége jelentősen hozzájárul az órákon való részvételi arány növekedéséhez, még akkor is, ha a technikai hiányosságok miatt sok hallgató csak saját költségén képes bekapcsolódni a szavazásba.

## 10. Fordított osztályterem

A fordított osztályterem módszer egy olyan tanulásszervezési modell, mely eltér a hagyományos módszertől első sorban abban, hogy megfordul az új tananyag megismerésének és feldolgozásának helyszíne. A hagyományos módszer alatt azt értjük, hogy a diákok az órán megkapják az új ismereteket úgy, hogy a tanár megtartja az előadást, a diákok pedig csak figyelnek és jegyzetelnek. Az otthoni munka során, pedig az órai jegyzetek és tankönyvek alapján megpróbálják elmélyíteni a tananyagot. Ezzel szemben a fordított osztályterem használata során a diákok a tananyagot megkapják előre és otthoni munkával feldolgozzák azt, majd pedig az órán ilyen módon sokkal több idő van a tananyag mélyebb rétegeinek feltárására, az ismeretek megbeszélésére csoportosan vagy akár párban. Az otthoni munka megfelelő elvégzéséhez elengedhetetlen a tananyag biztosítása. Erre számos lehetőségünk adódik. Egyrészt, használhatunk az interneten elérhető tananyagokat, dokumentációkat. Hogyha saját tananyagot szeretnénk biztosítani, akkor maradhatunk az egyszerű írott szöveg vagy prezentáció formáknál is, de készíthetünk videókat, hanganyagokat is. Számos olyan internetes felület létezik, melyen online tanagyagot hozhatunk létre, minek segítségével lépésről lépésre végigvezethetjük a tanulót a tananyagon. Érdemes a kiadott anyagokba elgondolkodtató kérdéseket, feladatokat helyezni, esetleg kvízt, mellyel ellenőrizni lehet, hogy ténylegesen elsajátították-e a tananyagot.

A fordított osztályterem módszerrel a diákok passzivitását aktivitássá lehetne formálni, mely a produktív tanulásra sokkal nagyobb hatással van. A hagyományos modellben a hallgatók sokszor úgy indulnak haza az óráról, hogy értik az új anyagot, csupán akkor jönnek rá, hogy nem világos valami, mikor egyéni munkát végeznek, legyen az házi feladat, szorgalmi vagy zárthelyi. Ezzel szemben, ha az ismereteket otthon megszerzik, az órán meg tudjuk beszélni az otthonra kiadott feladatokat, esetleg további feladatokat megoldani, bonyolultabb összefüggésekre rávezetni, így az órán a hallgatók több konkrét kérdésre, felmerülő problémára tudnak választ kapni.

Egy egyetemi előadás/gyakorlat sokszor monotonná válik, és a diákok vagy nem járnak be rá, vagy bár fizikailag ott tartózkodnak, a gondolataik egészen máshol járnak. A fordított osztályterem módszertől remélhetjük, hogy az órákon aktívabbak lesznek, és szívesebben részt vesznek. De hogyan tudjuk elérni azt, hogy ténylegesen el is végezzék az otthoni feladatokat? Egy lehetőség, hogy rendszeresen (hetente vagy kéthetente) rövid zárthelyiket írathatunk velük, ezzel ellenőrizve, hogy ismerik-e az adott anyagot. Adhatunk továbbá házi, szorgalmi feladatokat is, amivel (plusz) pontokat érhetnek el. Ezen ellenőrzések megvalósítása azonban több oktatói munkát igényel, és nagy létszámok esetén nehezebben kivitelezhető.

A fordított oktatás módszerének további előnye, hogy a hallgatók bármikor megnézhetik a közzétett tananyagot, és annyiszor hallgatják meg/olvassák el, amennyiszer nekik szükséges. Az óráról való hiányzás nem okoz akkora kiesést, és sokkal könnyebb bepótolni az anyagokat. Az otthoni tanulás mindenkinek a saját tempójában zajlik, így nem okoz akkora frusztrációt, mint mikor másokhoz kell alkalmazkodni.

A módszer hátrányai közé tartozik, hogy elvárjuk a diákoktól az aktivitást, és arra támaszkodunk. Azonban, ha a hallgatók nem teljesítik az otthoni munkát, az órán sem lehet haladni. Érdemes előre közölni velük, hogy milyen módszert alkalmazunk, és hogy mindennek az alapja, ha mindig elvégzik az otthoni tanulást. Az otthoni tananyagok megfelelő meghatározása és elkészítése alapvető fontosságú, és részletes kidolgozást igényel annak érdekében, hogy a tananyagot a hallgatók otthon önállóan is fel tudják dolgozni. Ezért ennek elkészítése nagyon sok időt igényelhet, ráadásul mindennek időben a hallgatók kezében kell lennie.

## 11. Elektronikus tananyag készítése

Úgy is dönthetünk, hogy a kurzuson belüli olyan tananyag részekhez, melyek kevesebb szóbeli magyarázatot igényelnek, elektronikus tananyagot készítünk. Ez alatt most nem egyszerű pdf-ben kiadott diákat értünk, hanem olyan környezetet, mely irányítja is a tanulás folyamatát, továbbá minél több motivációs eszközt tartalmaz. Ehhez ajánlott a tartalmat elemi lépésekre bontani, ezek megtanulásához ellenőrzési pontokat elhelyezni, a felhasználót pedig akkor jutalmazni és esetlegesen csak akkor tovább engedni, ha az adott részt megfelelően megtanulta. Nyilvánvaló, hogy egy ilyen tananyag elkészítése meglehetősen sok munka, hiszen egy előadáshoz képest további tervezést, átszervezést és ellenőrző tesztek készítését is igényli.

Egy másik lehetőség oktatóvideók készítése, ami alatt azt értjük, hogy a tanár a tananyag egy részének előadását videóban rögzíti.

Mindkét módszer előnye, hogy a hallgató saját időzítéssel tanulhat, akkor, amikor erre magát alkalmasnak érzi, az így felszabadult idővel pedig vagy csökkenthetjük a kontaktórák számát, vagy azokat nehezebb, megbeszélést inkább igénylő anyagrészekre fordíthatjuk.

## 12. Játékosítás

A játékosítás lényege, hogy a játékok motivációs mechanizmusait megpróbáljuk beépíteni az oktatási folyamatba, nem feltétlen játékok alkalmazásával. A cél az, hogy a hallgatók motiváltabbak legyenek a kurzusbeli feladatok elvégzésére, és ideális esetben azokat ne kötelezőként, hanem egy jó lehetőségként, kihívásként éljék meg.

A játékosítás csak egy kiegészítő lehetőség, ami nem fogja megoldani a funkcionális problémákat. Ha a hallgatók nem tudják, hogy miért lesz jó számukra a kurzus elvégzése, nem találják meg a segédanyagokat (vagy azokat elfelejtjük feltenni), nem érthető a tananyag, nem világosak a teljesítési feltételek, nem tudnak velünk kommunikálni (nem mernek vagy nincs erre megfelelő lehetőség), és hasonló problémák vannak, akkor ezeket a játékosítás nem fogja megoldani.

A játékosítást alaposan meg kell tervezni, ugyanakkor fontos az is, hogy legyenek alternatíváink nem várt helyzetekre.

A tervezés első lépéseként érdemes átgondolni, hogy kiket szeretnénk motiválni, azaz mit tudunk a hallgatókról [22]. Milyen céljaik vannak, és azokhoz hogyan tudunk a tárgyunkkal kapcsolódni? Milyen félelmeik, nehézségeik vannak, hogyan tudunk a kurzus keretén belül ebben nekik segíteni? Milyen szociális jellegzetességeik vannak, melyeket érdemes figyelembe vennünk? Például, vannak-e olyan helyzetben, hogy saját költségükön mobilinternetet használjanak a szavazáshoz? Milyen kommunikációs csatornákat használnak, mi lenne ezekből számunkra is megfelelő? Természetesen a hallgatók is sokfélék, és a leghatékonyabb lenne a motiválást személyre szabni, de ezt általában korlátozottan tudjuk megtenni, ilyen helyzetekben érdemes olyan mechanizmusokat használni, melyek várhatóan minél többféle vagy több hallgató esetén működnek.

A játék szempontjából hasznos lesz, ha létrehozunk egy történetet, amelybe a játékot beágyazzuk, ezt nevezzük narratívának. A hallgatók jobb érzéssel fognak részt venni a játékban, ha beleélhetik magukat egy történetbe, ami illeszkedik a gondolkodásmódjukhoz, céljaikhoz, és amiről esetlegesen még azt is gondolhatják, hogy valamilyen magasztos cél érdekében is cselekszenek. [23]

A játék során a távolabbi cél mellett szükséges rövidebb célokat is kitűzni, melyeket könnyebb elérni. Többnyire szinteket is szoktak készíteni, melyek azt jelzik, hogy a játékos vagy egy lokális célt elért (például megtanult egy anyagrészt), vagy összegyűjtött egy komolyabb mennyiségű pontot (ebben az esetben is megtanult egy komolyabb mennyiségű anyagot, csak kevésbé határoltuk be azt, hogy pontosan mit).

Érdemes lehet a játékosítás folyamatát még a kurzus előtt elindítani. A felkeltett érdeklődés következtében remélhetjük, hogy az első alkalomra többen fognak eljönni, a felfokozott várakozás pedig több energiát adhat a kezdéshez. Így nagyobb esélyünk lesz arra, hogy a hallgatókat meggyőzzük arról, hogy részt vegyenek a játékban. Ugyan – a teljesítési feltételekbe beépítve – a játékban levő részvételt kötelezővé is tehetjük, de számolnunk kell azzal, hogy ami kötelező, azt várhatóan kisebb lelkesedéssel fogják csinálni, ezért nehezebb dolgunk lehet a motiválás során.

Mindenképpen szükséges alaposan megtervezni, hogy milyen tevékenységeket várunk el a hallgatóktól a kurzus előtt, illetve annak során, és ennek megfelelően nekünk mit kell tennünk. A cél a hallgatók számára a flow élmény elérése, ehhez a feladatokat úgy kell megterveznünk, hogy a hallgatók számára kihívást jelentsenek, de ne jelentsen számukra túlságosan nagy nehézséget sem ezek elvégzése. A nehézség lehet szellemi, fizikai, anyagi, időbeli befektetés, rutintalanság vagy az elvárt cselekvés közösségi megítélése [19]. Például, hogyha a hallgatók rendkívül elfoglaltak, akkor nem a sok időt igénylő cselekvések lesznek alkalmasak, hanem inkább azok, amelyeket ugyan fokozott koncentrációval, de rövid idő alatt el tudnak végezni. De hogyha van idejük bőven, viszont nehezebben értik meg a tananyagot, akkor adhatunk sok, kevésbé nehéz (de érdekes) gyakorló feladatot.

Az elvárt cselekvésekhez jutalmakat kell rendelnünk. A jutalom ideális esetben arányos a cselekvés nehézségével, továbbá azzal, hogy mennyire fontos nekünk a cselekvés elvégzése. A jutalom lehet dicséret, pont, jelvény, csoki stb., a lényeg, hogy aki kapja, az fontosnak érezze. Sok esetben a pozitív visszajelzés is elegendő, feleslegesen ne adjunk túl sok jutalmat. Ugyanis, tudatában kell lennünk annak, hogy a külső jutalmak csak rövid távon működnek, tehát ha hosszabb távon szeretnénk játékosítani., akkor el kell érnünk, hogy a hallgatók élvezzék azt, amit csinálnak. De ha már külső jutalmakat adunk, akkor érdemes azt is figyelembe venni, hogy általában a leghatásosabb az olyan jutalom, ami nem rendszeres, hanem azt a játékos időnként kapja. [16, 19]

Mivel a Z generációnak fontos, hogy a többiekkel összemérje magát, ezért fontos jutalom lehet az is, hogyha közzétesszük, hogy adott játékos hol tart a többiekhez képest. Ezzel azonban óvatosan kell bánni, mert aki egy hosszú lista legalján látja magát, az reménytelennek érezheti, hogy bármire is vigye, és feladhatja. Ezért érdemes különböző tananyagokra, időszakokra, vagy más módon szűkített ranglistákat készíteni, vagy a játékosnak csak a lista egy részét megmutatni, amiben azt látja, hogy vállalható mennyiségű munkával több társát is megelőzheti, ugyanakkor utána is vannak, akik utolérhetik [20]. De nem ütközik-e az eredmények közzététele adatvédelmi korlátokba? Egyrészt, ha a játékban való részvétel önkéntes, akkor a szabályzatban mondhatjuk, hogy a csatlakozással a hallgató elfogadja azt (is), hogy az eredményeket általunk meghatározott és leírt módon közzétesszük. Másrészt, a játékosoktól amúgy is célszerű kérni általuk választott játékos nevet és logót, és akkor az eredményeket közzétehetjük csak ezek feltűntetésével.

Természetesen a cikk keretei között a fentiekben csak a játékosítás legfontosabb elemeit tudtuk összefoglalni, felsőoktatási kurzusok játékosításához is hasznos további ötleteket olvashatunk többek között a [18, 12, 5, 11, 17] cikkekben.

Ugyanakkor a fejezet lezárásaként szükségesnek tartjuk megjegyezni, hogy a fentiek konkrét kitalálása és megvalósítása meglehetősen munka- és időigényes tevékenység lehet, attól függően, hogy mennyire komplex rendszert tervezünk. Továbbá fontos, hogy a nehézségek ellenére lelkesek maradjunk, és a játékosokat is folyamatosan lelkesítsük, és figyelmeztessük az aktuális teendőkre.

## 13. Kapcsolódás az aktuális trendekhez

Az éppen aktuális trendek mindig tömegeket vonzanak, köztük hallgatókat is. Érdemes lehetőség szerint az oktatás során ezeket is kihasználni, hiszen ez által a hallgatók várhatóan jobban fogják magukat érezni a folyamatban, és motiváltabbak lesznek. Erre adunk most két ötletet a jelenlegi trendek figyelembevételével.

Napjainkban a szabadulószobák nagy lelkesedésnek örvendenek. Átalakítva és leegyszerűsítve megvalósíthatnánk a tanteremben is. Mivel rengeteg előkészületet és időt igényel, ezért nem lehet minden órán igénybe venni, azonban az érdeklődés fenntartása végett egyszer, kétszer érdemes megpróbálkozni vele. Csapatokra osztva a hallgatókat, kialakítható egy versenyhelyzet. Ki kell találni egy olyan történetet, mely érdekli a hallgatókat – legalábbis többségüket – legyen ez egy rejtvény megoldása, a világ megmentése, a gyilkos megtalálása stb. Majd kitalálni a lépéseket, melyekbe bele lehet építeni a tananyagot. Akár az eddig elhangzottak alkalmazását, akár új anyag elsajátítását. A lényeg azon van, hogy közelebb vigyük a hallgatók világához az ismereteket, és úgy használják, úgy sajátítsák el, hogy közben szórakoznak.

A Z generációban rendkívül népszerűek a Youtuberek. Az oktató – ha tud elég időt szánni rá – egyféle Youtuberként is működhet, hallgatói körében. A Youtubereket sokan követik, figyelik és várják a videóikat. Vannak olyanok, akik hasznos és fontos információkat osztanak meg, és vannak olyanok, akik csupán figyelemre vágynak, és megtesznek bármit a kamerák előtt. A fiatalok viszont lelkesedve várják az újabbnál újabb feltöltött videókat. Érdemes megpróbálkozni azzal, hogy a tanár videót készít minden héten, és közzéteszi azt a hallgatói számára. Ezek a videók tartalmazhatnak tananyagot is, de egyszerűen érdekességeket, párhuzamokat a tantárggyal kapcsolatban, aktuális információkat, feladatmegoldásokat.

## 14. Következtetések

Az elmúlt években a hallgatók tanulási attitűdjei jelentősen megváltoztak, ami miatt a hagyományos módszerek nem minden esetben működnek, ezért hasznos, ha a tanár olyan módszereket is használni tud, melyek jobban illeszkednek a fiatal generációk elvárásaihoz. Cikkünkben bemutattunk néhány lehetőséget arra, hogy egyes módszereket hogyan lehetne beépíteni az egyetemi kurzusokba, és néhány további ötletet az órák érdekesebbé tételére. Természetesen ezeknek a lehetőségeknek különböző korlátai vannak, de reméljük, hogy a kollégák találnak közöttük olyanokat, amelyek az adott körülmények között alkalmazhatóknak és hasznosaknak bizonyulnak.

## Irodalom


1. Gaskó Krisztina, Hajdú Erzsébet, Kálmán Orsolya, Lukács István, Nahalka István, Petriné Feyér Judit: A gyakorlati pedagógia néhány alapkérdése. Hatékony kutatás. 2.3. Kooperatív tanulás, ELTE PPK Neveléstudományi Intézet, 2006, http://mek.niif.hu/05400/05446/05446.pdf (utoljára megtekintve: 2019.11.12)
2. Gyarmathy Éva: Diszlexiás tanulókról – felsőfokon, 2. fejezet: Tanulási és tanítási stílusok, MTA Pszichológiai Kutatóintézet, Budapest 2010
3. Tari Annamária: Z generáció, Tericum Kiadó Kft., Budapest 2011
4. Ollé János, Papp-Danka Adrienn, Lévai Dóra, Tóth-Mózer Szilvia, Virányi Anita: Oktatásinformatikai módszerek, Tanítás és tanulás az információs társadalomban, Papp-Danka Adrienn: Tanulás és tanulásmódszertan az információs társadalomban, 57.o, Budapest 2013, ISBN 978 963 312 157 3, http://www.eltereader.hu/media/2013/11/Olle2_okt-inform_READER.pdf (utoljára megtekintve: 2019.11.11)
5. Rab Árpád: A gamifikáció lehetőségei a nem üzleti célú felhasználások területén, különös tekintettel a közép-és felsőoktatásra, Oktatás-Informatika, 2013.03., http://www.oktatas-informatika.hu/2013/03/rab-arpad-a-gamifikacio-lehetosegei-a-nem-uzleti-celu-felhasznalasok-teruleten-kulonos-tekintettel-a-kozep-es-felsooktatasra/ (utoljára megtekintve 2019.11.14.)
6. 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.)

7. Dr. Makó Ferenc: Tanulásmódszertan, 2.4. Kooperatív tanulási módszerek, Óbudai Egyetem, 2015.09.19., https://www.tankonyvtar.hu/hu/tartalom/tamop412b2/2013-0002_tanulasmodszertan/ tananyag/JEGYZET-20-2.4._Kooperativ_tanulasi_mods.scorml (utoljára megtekintve 2019.11.10.)
8. Pintér Dániel Gergő: Nem a Z-generáció butább, csak az oktatás ragadt a 20. században, 2015.10.09, https://media20.blog.hu/2015/10/09/a_z-generacio_szamara_alkalmatlan_a_jelenlegi_oktatas?token=b98e57085ba3008d9604e580736cf1f0 (utoljára megtekintve: 2019.11.03.)
9. Holló Csaba, Németh Tamás, Tanulási eredmények alapú egyetemi kurzusleírások készítése, tapasztalatok és módszertani hatások, INFODIDACT 2015 Informatika Szakmódszertani Konferencia elektronikus kiadványa (DVD), 1-8, Zamárdi, Hungary, November 26 - 27, 2015, ISBN: 978-963-12-3892-1., https://people.inf.elte.hu/szlavi/InfoDidact15/Manuscripts/HCsNT.pdf (utoljára megtekintve 2019.11.14.)
10. Csermely Péter: A hálózatok új tudománya, IVSZ MENTA 2015 konferencia, http://ivsz.hu/esemenyek/menta2015/ (utoljára megtekintve: 2019.11.06.)
11. Kenéz András: A mobil momentumok bevonása a tanulási folyamatba (felsőoktatási tapasztalatok), JátékosLét, 2016.05.17, http://jatekoslet.hu/news.php?extend.66 (utoljára megtekintve 2019.11.14.)
12. Pusztai Ádám: Így játékosítunk mi: beszámoló, kis túlzással esettanulmány, Kollektíva, 2016.11.04., http://kollektiva.eu/ora-jatekositas-beszamolo/ (utoljára megtekintve 2019.11.14.)
13. Freund Tamás: Agyhullámok, tanulás, kreativitás, Pedagógiai Esték, Szeged, 2016. 11. 22., https://www.youtube.com/watch?v=J13f5so7y2c (utoljára megtekintve 2019.11.15.)
14. Holló Csaba, Sárkány Rita, Németh Tamás: Hallgatói teljesítések javításának lehetőségei a felsőoktatásban informatikus szakokon, INFODIDACT 2016 Informatika Szakmódszertani Konferencia elektronikus kiadványa (CD), 1-13, Zamárdi, November 24 - 26, 2016, ISBN: 978-615-80608-0-6. https://people.inf.elte.hu/szlavi/InfoDidact16/Manuscripts/HCsSRNT.pdf (utoljára megtekintve 2019.11.11.)
15. Ollé János: Az oktatás módszertana, 2016 https://www.youtube.com/watch?v=kd1bLpPMnI8 (utoljára megtekintve: 2019.11.10)
16. Pusztai Ádám: A munka gyümölcse: 6 jutalom típus, amit a gamification használ, Kollektíva, 2017.02.06., https://kollektiva.eu/jutalmazas-rendszer/ (utoljára megtekintve 2019.11.14.)
17. Pusztai Ádám: Hogyan alakíthatunk ki hosszú távú elköteleződést a gamification-nel?, 2017.06.19., http://kollektiva.eu/gamification-elemzes-octalysis/ (utoljára megtekintve 2019.11.14.)
18. Hajba László: Gamifikáció a felsőoktatásban – egy online kurzus tapasztalatai, Tempus Közalapítvány, 2018.09.03., https://tka.hu/nemzetkozi/4149/learning-by-doing-avagy-gamifikacio--jatekositas-a-felsooktatasban--egy-online-kurzus-tapasztalatai (utoljára megtekintve 2019.11.14.)
19. Pusztai Ádám: "Ez engem nem motivál" – érted, hogy ez mit jelent?, Kollektíva, 2018.11.05., https://kollektiva.eu/kulso-belso-motivacio/ (utoljára megtekintve 2019.11.14.)
20. Pusztai Ádám: Jutalmazási rendszerek, Kollektíva, 2018.12.30, https://youtu.be/Tz7R0aEWkxE (utoljára megtekintve 2019.11.14.)
21. Schneider Ákos: „Számunkra egyértelmű, hogy mindenki designer" – Interjú Bényei Judittal és Csernátony Fannival, Designisso (DIS), 2019. 03. 26., https://designisso.com/2019/03/26/szamunkra-egyertelmu-hogy-mindenki-designer-interju-benyei-judittal-es-csernatony-fannival/ (utoljára megtekintve 2019.11.14.)
22. Pusztai Ádám: Így tervezd meg a gamification perszónádat (30 kérdés és válaszok), Kollektíva, 2019.05.28, https://kollektiva.eu/igy-tervezd-meg-a-gamification-perszonadat-30-kerdes-es-valaszok/?fbclid=IwAR1_-qtbmMHg2lFMPXJGdeQM0PNCn_tOaMCI-qnhDc_UeyIBckK_3Az7ByQ (utoljára megtekintve 2019.11.14.)
23. Pusztai Ádám: Így írj lebilincselő narratívát erőlködés nélkül (28 ötlet), Kollektíva, 2019.07.25., https://kollektiva.eu/igy-irj-lebilincselo-narrativat-erolkodes-nelkul-28-otlet/ (utoljára megtekintve 2019.11.14.)

24. Socrative, https://socrative.com/
25. Design Thinking for Educators, https://designthinkingforeducators.com/ (utoljára megtekintve 2019.11.14.)

# Nemi sztereotípiák az informatika oktatásban

Szlávi Anna

anna.szlavi@gmail.com
ELTE

**Absztrakt.** A 21. században a digitális olvasás- és írástudás elengedhetetlen az egyre inkább automatizált és digitalizált munkavégzés miatt. Az informatika oktatásnak tehát hatalmas a jelentősége és a felelőssége is, hisz a jövő rendszereinek felhasználóit és fejlesztőit kell képeznie. Éppen ezért a differenciált oktatás különösen fontos az informatika órákon: a hatékony tanulási folyamat elősegítése végett szükség van a diákok egyéni képességeinek és sajátosságainak, mint például kulturális, etnikai vagy épp nemi identitásának figyelembe vételére. A jelen tanulmány azt kívánja körüljárni, miért szükséges az informatika oktatást nemi aspektusok – különösen a nemi sztereotípiák – felől is megközelíteni. Nemcsak elméleti probléma a nők és lányok – nemi sztereotípiákból adódó – alacsony részvétele az informatikában, hanem kézzelfogható technológiai és társadalmi komplikációkhoz is vezet. Az elméleti áttekintés után a cikk konkrét módszereket sorakoztat fel arra, hogy hogyan lehet inkluzívabbá tenni az informatika órákat.

**Kulcsszavak:** gender, nem, informatika, oktatás, sztereotípia

## 1. Bevezetés

A 21. század az informatika kora, hisz az élet legtöbb színterén jelen van a technológia. A számítógép, az okostelefon, az okos háztartási gépek már legtöbbünk napjainak nélkülözhetetlen részei. Azonban nemcsak a szórakozás és a kapcsolattartás eszközei, hanem a munkáé is: az automatizált és digitalizált munkavégzés miatt manapság (és a jövőre nézve még inkább) elengedhetetlen a digitális olvasás- és írástudás.

Az informatika oktatásnak tehát hatalmas a jelentősége és a felelőssége: a jövő rendszereinek felhasználóit és fejlesztőit kell képeznie. Ezért az informatika mint szaktárgy sokrétű [1], interaktív [2], és kooperatív [3], amelynek célja nemcsak az algoritmikus gondolkodás fejlesztése [4], hanem a szociális nevelés is. Belátható, hogy a differenciált oktatás, a tanár tutori szerepvállalása [2] különösen fontos tehát az informatika órákon. A diákok egyéni képességeinek és sajátosságainak, mint például kulturális, etnikai vagy épp nemi identitásának figyelembe vétele nélkülözhetetlen; nemcsak morális és szociális szükséglet, hanem szakmai és technikai is. Mind az informatika órákon, mind a programozás során létfontosságú a kooperáció, és ez csak nyitott és elfogadó légkörben valósul meg, ahol a sztereotípiák helyett a sokszínűség a vezérlő elv.

A jelen tanulmány azt kívánja körüljárni, hogy a nemi sztereotípiáknak milyen hatása van az informatika oktatásra, és ebből következően az informatika világára. A cikk célja elsősorban, hogy felhívja a figyelmet az oktatás, különösen az informatika oktatás nemi aspektusaira, rávilágítván, hogy nemcsak elméleti probléma a nők és lányok alacsony részvétele az informatikában, hanem kézzelfogható technológiai és társadalmi komplikációkhoz is vezet. Másodsorban azt is céljául tűzte a tanulmány, hogy az elméleti áttekintés és problémafelvetés után konkrét módszereket vonultasson fel arra, hogyan lehet inkluzívabbá tenni az informatika órákat és ezzel több nőt és lányt bevonni az informatika szakmába.

## 2. Társadalmi nem

Az elmúlt évtizedekben számtalan nemzetközi és hazai kutatás fordította figyelmét a társadalmi nem vagy gender kérdésére [5][6][7][8]. A legkülönbözőbb tudományterületek, köztük a nyelvészet [9], a szociológia [10], a természettudományok [11], a politikatudomány [12] és a pedagógia [13] is feltárta, hogy a nemi alapú megkülönböztetés, illetve a nők hátrányos helyzete áthatja társadalmunkat.

Ami a magyar kontextust illeti, habár az ország 1996. óta az Gazdasági Együttműködési és Fejlesztési Szervezet (OECD), 2004. óta pedig az Európai Unió (EU) tagja, nem tud jó eredményeket felmutatni a nemek közötti egyenlőség tekintetében. Ugyan a patriarchális nemi viszonyok hosszú múltra tekintenek vissza Magyarországon, vagyis a nemi egyenlőtlenség nem új keletű jelenség [14], az utóbbi tíz évben Magyarország az EU és az OECD egyik legelmaradottabb országává vált a nemi egyenlőség tekintetében.

A legfrissebb világméretű felmérések alapján [15] a nemek közötti szakadék Magyarországon jelentősen mélyült az elmúlt évtizedben: Európán belül az egyik utolsó, globális szinten pedig az utolsó harmadban kullog az ország. A felmérés, amely a nők és a férfiak oktatási lehetőségeit, gazdasági részvételét, egészségügyi helyzetét és politikai képviseletét vizsgálja, kimutatta, hogy a nemi egyenlőtlenség a nők érdekképviselete tekintetében a legsúlyosabb: Magyarország a világ tíz legegyenlőtlenebb országának egyike. Vagyis Magyarországon a nők nagyon kevéssé vannak irányító, véleményformáló szerepben [16].

## 3. Társadalmi nem az informatikában

Az informatika területén is jelentős hátrányban vannak a nők. Ha a legfelsőbb szintet nézzük, a világ 3 legnagyobb és legbefolyásosabb informatikai cégénél, az Apple-nél, a Microsoftnál és az Amazonnál csekély számú nőt találunk a felső vezetésben. Az Apple-nél 4 nő jut 12 férfire, a Microsoftnál 3 a 12-re, míg az Amazonnál 6 a 11-re (ld. apple.com, microsoft.com, aboutamazon.com).

Azonban nemcsak a vezetőségben találunk egyenlőtlenségeket, hanem az informatikai szakma egészében. Kirkup felmérése az Egyesült Királyság, az Egyesült Államok, Kanada, Tajvan, Írország és Spanyolország IKT szektorát vizsgálva mind vertikális, mind horizontális munkamegosztást és egyenlőtlenségeket talált [17]. A felmérésből az derült ki elsősorban, hogy a nők súlyosan alulreprezentáltak az IKT szakmákban. Különösen kevés a nő az IKT szektor mérnöki/technikai pályáin. Azok a nők, akik mégis az informatikai szférában dolgoznak, tipikusan alacsonyabb státuszban vannak és/vagy kevesebbet keresnek, mint a férfiak. Ezzel együtt, bizonyos országokban az IKT szakmában dolgozó nők magasabb iskolai végzettséggel rendelkeznek, mint férfi-kollégáik. A kutatás továbbá arra is rámutatott, hogy a férfiak előléptetése gyorsabb, mint a nőké, még akkor is, ha vezetői képességeik gyengébbek.

Ugyan egyre több figyelmet fordítanak a kutatók és a szakma a nők alacsony részvételére, és egyre több kezdeményezés igyekszik a lányokat és a nőket bevonni a STEM szakmákba, vagyis a (természet)tudományba, a technológiába, a mérnöktudományba vagy a matematikába (ld. girlsintech.org, girlswhocode.com, www.womenintechnology.org), az eredmények mégis korlátozottak. Továbbra is alig látni nőket – jól kereső és/vagy befolyásos – programozó pozíciókban, ami azt eredményezi, hogy kevés a minta és a példakép a fiatal generációk előtt. Továbbra is tehát megkérdőjelezetlenül maradnak az egyenlőtlen helyzetet kiváltó nemi sztereotípiák, vagyis hogy a lányokat nem érdekli az IKT, nem értenek hozzá és nem tudják érvényesíteni magukat [17].

Az informatika és a gender kapcsolata ritkán merül fel azon a ténymegállapításon kívül, hogy kevés a nő az informatikai szakmában. A nők kimaradásának azonban nemcsak társadalmi következményei vannak – vagyis hogy egy jól kereső szakma „férfi-privilégium", ami ilyen formán súlyosbítja a nők anyagi lemaradását – vagyis nemcsak „humán" probléma. Technikai, tehát az informatika számára is releváns következményei is vannak. Az, hogy a szoftverek, programok, algoritmusok

mögött ki áll, hatással van a szoftverekre, programokra, algoritmusokra. A nők tipikusan felhasználói, nem gyártói az IKT-nak [18], amely azt eredményezi, hogy a női szemszög, vagy egyszerűen a diverzitás csorbát szenved az alkotó/fejlesztő folyamat során. A legszembetűnőbb példa a videójátékok tematikája és ember-ábrázolása, amely csak az utóbbi években vált gender-inkluzívabbá. Addig kizárólag férfi-fejlesztők férfiaknak/fiúknak készítették a játékokat, nagyon kevés, egy női játékos számára választható, „önazonos" (vagyis például nem tárgyiasított és szexualizált) perszónákat vagy tematikákat felsorakoztatva.

**1. ábra:** Videójáték nő-ábrázolásai (forrás: pixelkin.com)

A videójátékok esetében, még ha megkérdőjelezhetetlen is a nemi alapú probléma, könnyen félretehető azzal a feltevéssel, hogy csak egy szűk réteget – a videójátékokkal játszó embereket – érinti ez a torzulás. Illetve foghatjuk a szórakoztatóipar „sajátosságaira" is, amely az erőszakot és a szexualitást próbálja (többek között) eladni játékaival.

Azonban az informatika számos „objektív" területén is problémákat generál a nők, illetve a diverzitás hiánya a programozói csapatból. Hogy csak egy példán keresztül érzékeltessük, vizsgáljuk meg a gépi tanulásra épülő arcfelismerés technológiáját, amely kétségtelenül objektívként és nemsemlegesként definiálja magát, és akkor működik hatékonyan, ha valóban az is. Az MIT friss kutatása azonban arra mutatott rá, hogy a legnagyobb arcfelismerő szoftverek (Microsoft, Face++ és IBM) működése se nem objektív, se nem gender-semleges [19]. Buolamwini és Gebru vizsgálata szerint az egyes szoftverek nem működnek ugyanolyan pontossággal és hatékonysággal nőknél, különösen fekete bőrű nőknél, mint ahogy férfiaknál. 23%, 36% és 33%-os hibaaránnyal dolgoztak fekete nők arcfelismerésénél, míg fehér férfiaknál alig volt hiba.

Elméletük szerint a gépi tanuláshoz használt minták nem voltak megfelelőek: a (nemi és etnikai) diverzitás nem volt elégségesen reprezentálva a mintákban, ahogy vélhetően a programozói csapatban sem. Buolamwini arra figyelmeztet, hogy a sztereotípiák és előítéletek – vagy épp a hátrányos társadalmi státuszt sugalló láthatatlanság – könnyen átkerülhetnek a programokba is, s így továbbterjedhetnek az objektívnek tartott algoritmusokon keresztül, ha a programozók csapata nem elég sokszínű, hogy figyelhessenek egymás vakfoltjaira, – akár önkéntelenül sztereotip – berögzüléseire [20].

Szemben a videójátékokkal, az arcfelismerő szoftverek nagyon széles körben elterjedtek és használatosak. Már az okostelefonokon is megjelentek az automatizált arcfelismerő rendszerek, de ezt a technológiát alkalmazzák érzelemfelismerő programokban, autizmussal élők vagy látás sérültek segítésére, vagy a bűnmegelőzésben is [19]. Éppen ezért az algoritmus (nemi, etnikai vagy egyéb

aspektusú) téves működése súlyos következményekkel járhat; a sokszínű, inkluzív és kooperációra képes fejlesztői csapat tehát kétségtelenül szükséges.

# 4. Társadalmi nem az informatika oktatásban

Jól látható, hogy a lányok és nők jelenléte az informatikában kulcsfontosságú. A nemi egyensúly elősegítésében pedig központi szerepe van az iskoláknak és az informatika tanároknak.

Egy 2012-es hazai felmérés szerint, amely azt vizsgálta, a lányok miért választják csupán kis számban a műszaki pályát, a középiskolai reáltárgyak során szerzett rossz tapasztalatok voltak a fő okok [13]. A megkérdezett lányok nagy része negatív élményekről számolt be matematika és fizika órák kapcsán, nemcsak az órák tartalmát illetően, hanem a szaktanárok hozzáállását is. Az interjúba bevont tanárok válaszadása is sok esetben megerősítette ezt: sok tanár interjújában fellelhetőek voltak az olyan nemi sztereotípiák, hogy a lányok rosszabbul teljesítenek matematikából (vagy reáltárgyakból általában), kevésbé okosak, ötlettelenek, és a többi. A tanárok egy része tisztában volt vele, hogy másképp kezeli a fiú- és másképp a lány-diákjait.

A vizsgálat kitért az informatikára is. A műszaki-informatikai szakmákat férfiasnak címkézték a meginterjúvolt lányok, és voltak, akik azt is hozzátették, hogy úgy érzik, nehezen összeegyeztethető a nők „családi teendőivel". A nemi identitás konstrukciós szerepe [5] tehát jelentősen befolyásolja a lányok pályaválasztását.

A fentebbi vizsgálatból jól látszik, hogy a nemi sztereotípiáknak és a szaktanároknak fontos szerepe van abban, hogy a diákok milyen szakmát tudnak elképzelni maguknak, vagy mit tartanak megvalósíthatónak. Vagyis, az informatika tanár gender-semlegességre törekvő vagy akár nyíltan buzdító viselkedése kompenzálhatja az átható társadalmi nemi sztereotípiákat és pozitívan hathat lány-diákjai pályaválasztására.

## 4.1. Elmélet

Hogyan tud tehát egy informatikatanár inkluzív módon tanítani? Először lássuk az elveket!

Az általános iskolai és középiskolai informatika tanárok szerepe hatalmas, hisz pályaválasztás előtt álló diákokat vezethetnek be az informatika szépségeibe és terelhetik őket az informatikai pályára. (Ezzel szemben az felsőoktatásban tanító kollégák már csak azzal a jóval kisebb számú hallgatóval dolgozhatnak együtt, akiket erre a pályára „tereltek.") Az általános iskolai és középiskolai informatika tanár munkájában tehát különösen fontos, hogy tisztában legyen a szaktárgyához kapcsolódó nemi sztereotípiákkal, és tudatosan álljon az informatika oktatás nemi aspektusaihoz. Mindehhez elengedhetetlen az önreflexió [21]. Nemcsak a nemi egyenlőség elősegítése érdekében, hanem a sokszínű, toleráns, együttműködő csoport érdekében fontos a nyitottság, a kommunikáció, a méltányosság és demokratikusság elve, a szociális érzékenység, valamint a sztereotípia- és előítéletmentesség [21].

A tanári hozzáállás mellett a módszereknek is központi szerepe van. A feladatok sokféleségével inspirálni és motiválni képes a – sokféle – diákságot a tanár [17], vagyis ha több oldalról is bemutatjuk, mire képes az informatika és mire teszi képessé a diákot, nagyobb az esély arra, hogy érdekesnek és izgalmasnak fogják látni a diákok. A feladatok megoldása csoportokban vagy párokban, egyszóval kooperációban, sokkal hatékonyabb [3] és motiválóbb is. Ezen kívül érdemes hangsúlyt fektetni a feladatok tartalmának tudatos kiválasztására is: a multikulturalitás [22] inspiráló (és helyenként hiánypótló is) lehet. Kátai etnikai és kulturális anyagok használatát javasolja; ehhez hasonlóan nemi aspektusú sokszínűséget is tükrözhet az anyagunk.

Összefoglalva, a 21. századi középiskolai informatika tanár fontos célkitűzése kell, hogy legyen, hogy a lányokat ne felhasználókká, hanem alkotókká, programozókká, nevelje. Legyen tudatos cél megmutatni nemcsak azt, hogy mitől érdekes az informatika, hanem azt is, hogy elérhető pálya ez a

lányok előtt (is). Ehhez fontos felvonultatni példákat, sikertörténeteket és biztatni őket, hogy ez számukra is egy lehetséges jövő.

Ami a felsőoktatás informatikai területén oktató tanárokat illeti, az ő szerepük és felelősségük sem elhanyagolható. Igaz, hogy jóval kisebb számú diákra tudnak hatást tenni, mint a közoktatásban dolgozó kollégáik, de tanítási módszereiken sok múlik. Itt már nem az érdeklődés felkeltése a feladatuk, hanem az érdeklődés fenntartása, valamint a tudatos, szociálisan érzékeny, inkluzív programozók nevelése. Margolis és Fisher [23] az oktatás humanizálását és kontextualizációját emelik ki, vagyis, hogy fontos a programozás humán aspektusát láttatni, illetve összekapcsolni sokféle területtel és témakörrel. Emellett a csoportos, illetve páros munkát tartja alkalmasnak a sokszínűség, a tolerancia és az együttműködés támogatására. Továbbá, kifejezetten a női hallgatók helyzetének megszilárdítása érdekében hatékonynak tartja a példaképek és a sikertörténetek hangsúlyozását, melynek érdekében összekötné a női hallgatókat női tanárokkal és szakmabeliekkel.

## 4.2. Gyakorlat

A következőkben röviden áttekintünk egy gyakorlati példát arra, hogyan lehet egy informatikai kontextusú órát gender-inkluzív és sokszínűséget támogató módszerekkel megtervezni.

A kurzus a programtervező informatikus (felsőszintű) képzés hallgatóinak készült, azzal a céllal, hogy szakmai angol nyelvhasználatukat fejlessze és lehetővé tegye (a döntően angol nyelvű) szakmai anyagok feldolgozását a képzés programozó óráin. A kurzus a képzés első évének kötelező, két féléves tantárgya, tehát az összes informatikus hallgató részt vesz rajta. Az adott évfolyam létszámától függően több, tipikusan 12-18 fős csoportban végzik el a kurzust.

Ugyan nem szigorúan vett "informatika óráról" van szó, az itt bemutatott elvek és gyakorlatok hasznosak és alkalmazhatóak programozó tárgyaknál is, részben mert a módszertani elvek általánosak, részben mert az óra szorosan kapcsolódik az informatikai témákhoz, hisz az informatikai szaknyelv elmélyítése a cél, így a gyakorlatok nem szakma-idegenek.

A két(féléves) egymásra épülő órák módszertani és tartalmi megtervezésénél figyelembe kellett vennem, hogy (1) sok hallgató kerül egy csoportba, (2) nagyon különböző szintekkel érkeznek, (3) kicsi a csoportokon belüli – nemi, etnikai stb. – diverzitás: ami a lányokat illeti, egy 15-16 fős csoportban jó, ha 3-4 lány előfordul. Ezért módszertanilag a kevés frontális, és sok csoport- és pármunka tűnt jó megoldásnak. Így egyrészt fejlődnek a kooperációs és szociális készségek, mint például a kommunikáció, a tolerancia és a másság-elfogadás. Másrészt az egyéni (szaktudásbeli) különbségeket is könnyebb kiegyenlíteni: az erősebb tudással rendelkező segíti a gyengébbet, vagy épp kiegészítik egymás tudását, mert együtt kell megoldaniuk és leadniuk a feladatot. (Mindez nemcsak nyelvi képességekre és órákra vonatkoztathatóak, hanem ugyanúgy programozóira is.)

Ami a kurzus tartalmának kiválasztását illeti, a fentebb felsorolt körülmények miatt fontosnak tűnt, hogy a képességfejlesztés inspiráló, gondolatébresztő és szociálisan érzékenyítő – szakmai – anyagok kontextusában zajljon. Ezért a kurzus egyik alappillére, s egyben kapocs is a két félév anyaga között, a következő nyolc, informatikai témát bemutató, magas színvonalú TED-előadás[19] elemzése:

- Agüera y Arcas, Blaise: *How PhotoSynth can connect the world's images*
- Arar, Raphael: *How we can teach computers to make sense of our emotions*
- Bracy, Catherine: *Why good hackers make good citizens*

[19] A TED (www.ted.com) egy ingyenesen hozzáférhető, magas színvonalú (főleg angol nyelvű) előadások gyűjtőhelye. Ahogy a TED rövidítés sugallja, a prezentációk főleg a technológia (technology), az oktatás (education) és a dizájn (design) témájára reflektálnak.

- Buolamwini, Joy: *How I'm fighting bias in algorithms*
- Feinberg, Danielle: *The magic ingredient that brings Pixar movies to life*
- el Kaliouby, Rana: *This app knows how you feel – from the look on your face*
- Lupi, Giorgia: *How we can find ourselves in data*
- Redmon, Joseph: *How computers learn to recognize objects instantly*

**2. ábra:** A kurzushoz használt informatikai TED videók előadói

Ahogy Margolis és Fisher [23] rámutattak, fontos kontextualizálni és humanizálni az informatikát, bemutatván, milyen (és milyen sokféle) emberi kontextusban használható a programozás. A kiválasztott nyolc videó különböző területekbe enged betekintést (pl. arcfelismerés, gépi látás, animálás, adatvizualizáció). Emellett lényeges eleme mindegyiknek, hogy a programozás és az egyes technológiák humán aspektusát és társadalmi felelősségvállalását is hangsúlyozzák, ami kiemelt célja volt a kurzusnak, hisz erről ritkán esik szó. Ahogy a prezentációk felsorolása, illetve a 2. ábra szemlélteti, a sokszínűség az előadók – a szakmai példaképek – személyében is megvalósul: példát szolgáltatnak különböző származásra, különböző etnicitásra és különböző nemre (többek között). Szándékos választás volt, hogy legalább annyi, ha nem több női előadó legyen a kurzus anyagában, mint amennyi férfi, hogy ezzel is sugallni tudja az óra, hogy vannak női programozó sikertörténetek, a nők is hiteles és inspiráló szakemberek, és úgy általában: a diverzitás az informatika területén is lehetséges (és szükséges is).

A félév végén a hallgatók visszajelzést adhattak az elemzett videókra. Az írásbeli visszacsatolások alapján megállapítható, hogy inspirálónak találták őket; sokan példaképeket találtak az előadókban. A lány-hallgatók esetében különösen számottevő volt az egyik női előadó (Danielle Feinberg) méltatása: többektől elhangzott, hogy ők is ilyenek szeretnének lenni.

A kurzus tehát mind módszertanával, mind anyagválasztásával igyekezett elfogadó, sztereotípiamentes, sokszínűséget támogató kontextusba helyezni az informatikát, különös tekintettel arra, hogy a női hallgatók – a berögződött társadalmi sztereotípiákkal szemben – példaképeket kapjanak, és ezzel megerősödjenek informatikai pályaválasztásukban.

## 5. Befejezés

A jelen tanulmány célja az volt, hogy bemutassa, miért szükséges az informatika oktatást nemi aspektusok – különösen a nemi sztereotípiák – felől is megközelíteni. A nők és a lányok súlyosan alulreprezentáltak az informatikai szakmában, amelynek társadalmi, gazdasági és technológiai következményei is vannak. A szakmához (és általában a STEM területekhez) kapcsolódó nemi sztereotípiák megkérdőjelezésével és konkrét példaképeken keresztüli megcáfolásával, valamint a sokszínűséget és a kooperációt támogató módszerekkel, az informatika tanárok nagy hatással lehetnek az informatikai világ nemi egyensúlyára.

## Irodalom


1. Gál, B., Dávid, Á. *Az önfejlesztés és a reflektív pedagógusszemélyiség kialakításának lehetőségei az informatika szakos tanárképzésben*. InfoDidact. (2008) https://people.inf.elte.hu/szlavi/InfoDidact08/Manuscripts/GBDA.pdf (utoljára megtekintve: 2019.12.20.)
2. Pšenáková, I., Heizlerné, B. V., Illés, Z. *Interaktivitás az informatikatanításában*. InfoDidact. (2018) https://people.inf.elte.hu/szlavi/InfoDidact18/Manuscripts/PIHBVIZ.pdf (utoljára megtekintve: 2019.12.20.)
3. Nahalka, I. *Az általános didaktika és az informatikatanítás didaktikájának egymásra hatása.* InfoDidact. (2009) https://people.inf.elte.hu/szlavi/InfoDidact09/Manuscripts/NI.pdf (utoljára megtekintve: 2019.12.20.)
4. Zsakó, L., Szlávi P. *Informatikai kompetenciák: Algoritmikus gondolkodás*. InfoDidact (2010) https://people.inf.elte.hu/szlavi/InfoDidact10/Manuscripts/ZsL_SzP.pdf (utoljára megtekintve: 2019.12.20.)
5. de Beauvoir, S. *A második nem*. Budapest: Gondolat (1969)
6. Butler, J. *Gender trouble*. New York & London: Routledge (2007)
7. Huszár, Á. *A nő terei*. Budapest: L'Harmattan (2011)
8. Bozzi, V., Czene, G. *Elsikkasztott feminizmus*. Budapest: Osiris (2006)
9. Huszár, Á. *Bevezetés a gender-nyelvészetbe*. Budapest: Tinta (2009)
10. Milestone, K., Meyer, A. *Gender and popular culture*. Malden, Cambridge, MA.: Polity (2012)
11. Gibney, E. *Women under-represented in world's science academies*. Nature. (2016) www.nature.com/news/women-under-represented-in-world-s-science-academies-1.19465 (utoljára megtekintve: 2019.12.20.)
12. *OECD Recommendation of the Council on Gender Equality in Public Life*. (2016). http://www.oecd.org/governance/2015-oecd-recommendation-of-the-council-on-gender-equality-in-public-life-9789264252820-en.htm (utoljára megtekintve: 2019.12.20.)
13. Óbudai Egyetem. *Lányok útja a műszaki diplomáig – Középiskolai és felsőoktatási esélyek és nemi különbségek a műszaki pályaválasztás területén*. (Zárótanulmány) (2012) http://nokatud.hu/letoltes/Lanyok_utja_a_muszaki_diplomaig.pdf (utoljára megtekintve: 2019.12.20.)
14. Marinovich, S., Arpad, S. *Why hasn't there been a strong women's movement in Hungary?* (1995) https://onlinelibrary.wiley.com/doi/abs/10.1111/j.0022-3840.1995.2902_77.x (utoljára megtekintve: 2019.04.20.)
15. *Global Gender Gap Report.* (2017) http://reports.weforum.org/global-gender-gap-report-2017/dataexplorer/#economy=HUN (utoljára megtekintve: 2019.12.20.)
16. Szlávi, A. *The Construction of Gender in Hungarian Discourses*. Doktori disszertáció. ELTE (2019)
17. Kirkup, G. *ICT as a tool for enhancing women's education opportunities, and new educational and professional opportunities for women in new technologies*. United Nations Division for the Advancement of Women (UNDAW) (2002)
18. Crutzen, C. *Questioning Gender in E-learning and its Relation to Computer Science. Space for design, working, and learning*. In: Braidotti, R. & van Baren, A. (eds.) The Making of European Women's Studies Vol. VI. University of Utrecht, (2005) 40-59.
19. Buolamwini, J., Gebru, T. *Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification*. Proceedings of Machine Learning Research 81:1–15, (2018) http://proceedings.mlr.press/v81/buolamwini18a/buolamwini18a.pdf (utoljára megtekintve: 2019.12.20.)
20. *Algorithmic Justice League* https://www.ajlunited.org/ (utoljára megtekintve: 2019.12.20.)

21. Lipovits, Á., Háli, A., Kovács, E., Pozsgai, T., Gál. B. *Ötletek az informatikatanárok képzéséhez*. InfoDidact. (2010) https://people.inf.elte.hu/szlavi/InfoDidact10/Manuscripts/LA_et_al.pdf (utoljára megtekintve: 2019.12.20.)
22. Kátai, Z. *Multicultural Computer Science Education*. InfoDidact. (2011) https://people.inf.elte.hu/szlavi/InfoDidact11/Manuscripts/KZ.pdf (utoljára megtekintve: 2019.12.20.)
23. Margolis, J., Fisher, A. *Unlocking the Clubhouse. Women in Computing*. Cambridge Mass, MIT Press. (2002)

# A tanulás változó szerepe az információs társadalomban

Vetési Erika

vetesi.erika@tok.elte.hu
ELTE TÓK

**Absztrakt.** Az előadás keretein belül a változó világban az iskola változó világával ismerkedhetünk meg. Történeti áttekintés során érintjük a korábbi korok tanulás felfogásait. Megvizsgáljuk, hogy az adott neveléstörténeti korszakban a tanulás milyen szerepet töltött be. Végül a 21. század kihívásaira reflektálva értelmezzük a tanulás szerepét az információs társadalomban.

**Kulcsszavak:** tanuláselmélet, információs társadalom, 21. századi készségek

## 1. Bevezetés

A különböző korok tanuláselméleteit vizsgálva arra gondolhatunk, hogy a technológiai újítások mindig is hatással voltak az oktatási folyamatokra, bennük a tanulásra. Olyan hozzáadott értékkel bírtak, amelyek nem helyettesítették, hanem kiegészítették minden korban a tanulás folyamatát. Egészen messziről, az antik világtól indulva, a középkori és újkori történelmi időket áttekintve jutunk el a 21. század világához. Vizsgálódásunk középpontjában az áll, hogy a történelmi változások hogyan befolyásolták a tanulás szerepét a különböző neveléstörténeti korszakokban.

## 2. A szavak ereje

Nincs kétségünk afelől, hogy az európai kultúra fejlődésének szempontjából az antik görög társadalom és az antik görög nevelés kiemelt jelentőséggel bír. Görög filozófusaink közül Szókratész volt az első, aki a nevelésről fontos gondolatokat fogalmazott meg. [1]

Szókratész a Phaidrosz c. művében az írásbeliség megjelenésének kapcsán arról elmélkedik, hogy az írás az emlékezés elleni varázsszerként hat. Ugyanis, ha leírjuk a gondolatainkat, kevésbé vagyunk rákényszerítve a minél pontosabb emlékezésre, felidézésre. Az írás nem a felejtés ellen gyógyír, hanem épp ellenkezőleg, elősegíti a feledést. [2]

Érthető ez az érvelés, ha belegondolunk a korabeli viszonyokba. A tudást akkoriban a mesterektől csak néhány kivételezett szerezhette meg, kizárólag az élőszóban folytatott viták, beszélgetések során. Ne felejtsük el azonban, hogy az írás, mint technológiai újítás, óriási értékkel bírt a későbbi századok során, hiszen óriási demokratizáló hatásával nem csak földrajzilag, hanem időben is lehetővé tette az ismeretek megőrzésén túl a tudáshoz való nyitottabb hozzáférést.

## 3. A szemléltetés forradalma

Az előbbi gondolatunkat folytatva, megfigyelhetjük, hogy míg a koraközépkorban az iskolázottság továbbra is egy szűk, elit réteg kiváltsága maradt, addig a könyvnyomtatás elterjedésével a 15. századtól kezdve a tudáshoz való hozzáférés további átalakuláson ment keresztül.

Ebben az átalakulásban a 17. század szülöttének, Johannes Amos Comeniusnak további szerepe volt. Comenius alkotta meg az első részletesen kidolgozott pedagógiai rendszert. Olyan didaktikai és metodikai alapelveket alkotott meg, amelyek azóta is megfigyelhetők az iskolai gyakorlatban. Az alapelvek közül a szemléletességet kiemelve elmondható, hogy ezzel Comenius arra törekedett, hogy

minél több érzékszervet bevonjon a megismerés folyamatába. Ennek fontosságát előtte is többen hangsúlyozták, de a pedagógia történetében az ő nevéhez köthető az első képekkel illusztrált tankönyv, amelyben az ábrákon látható dolgok megnevezése is szerepel. [1]

Technológiai újításként az írásbeliség nyomtatott változata, illetve a képi vizualitás módosították a tanulás szerepét. A nyomtatott és illusztrált könyv a tanulás új eszközeként arra ösztönözte a tanulókat, hogy életükben már nem csak az emlékezés, hanem a képek értelmezése, feldolgozása is elengedhetetlenné váljon. Az antik világban és a középkorban is a nagy dogmarendszerek állításai jelentették a tanulás alapját. Ezekben az időkben a tehetséges fiatalok felemelkedésének egyik lehetséges útja éppen az egyházi oktatásban való részvétel, a klerikális előrejutás, boldogulás.

## 4. Az ipari forradalom hatása

A 18. század közepétől kezdve erőteljes és egyre gyorsabban bekövetkező változások vették kezdetüket. Egészen addig a lakosság többsége a mezőgazdaságban termelte meg az életben maradáshoz szükséges javakat. A társadalmi ranglétrán való előbbre jutás szinte lehetetlen volt, kivéve az előbb említett egyházi útvonalat.

A technológiai újítások közül az első óriási hatású, a gőzgép feltalálása volt. Ahogyan megjelentek a gépek, úgy alakult át az életvitel, a társadalmi berendezkedés, a gondolkodásmód. Innentől kezdve többször formálódott át a világtérkép is. A kort kísérő állandó háborúk idején kezdték az azonos életkorú fiúkat osztályokba rendezni és besorozni. A kor egyik legnagyobb gondolkodója, Wilhelm von Humboldt alapelveire építve megszületnek az első, modern központilag irányított állami oktatási rendszerek Európában. Az oktatás célja, hogy a növendékekből engedelmes katonákat képezzenek. Ez a poroszos oktatási stílus aztán kedvezett az ipari forradalom idején működő iskoláknak, hiszen futószalagon bocsátották ki a hatékony munkaerőt. [3]

Johann Friedrich Herbart pedagógiai elmélete elementáris erejű hatást gyakorolt a korabeli klaszszikus középiskolákra. A nevelésnek három szakaszát különítette el; a kormányzás, az oktatás és a vezetés szakaszait. Az első szakasz a porosz kaszárnyák ridegségét, lélektelen fegyelmét idézi. A második már a jövőre, az erkölcsi fejlődésre irányul. A harmadikkal, a vezetéssel válik a nevelés teljessé. A vezetés az oktatással párhuzamosan futó tevékenység. Ekkor a tapintat, a gyengédség és a jó kedély váltják föl a korábbi rideg bánásmódot, kérlelhetetlen szigort. [1]

Ellen Key 1900-ban megjelenő Gyermek évszázada c. könyve volt a nyitánya a reformpedagógia térnyerésének. Már korábban is léteztek azonban olyan iskolák Európában, melyek a nevelés korábbi (Herbart-féle) formáinak kritikájából kiindulva fogalmazták meg saját reformpedagógiai nézeteiket. A mozgalommá szerveződés ugyanakkor csak a 20. század elején következett be. Ebben a felfogásban a tanulás olyan környezetet kívánt, melyben a pedagógus kötelessége a segítés, a feltételek megteremtése, és nem elsősorban a tanulnivalók kijelölése és számonkérése. A tanulás célja, hogy a gyermek önálló kutató tevékenységet folytasson, manipuláljon, problémákat oldjon meg. [4]

Az ipari társadalomban bekövetkező egyik legnagyobb változás az, hogy a természet helyett az idő és a naptár vette át az irányítást. A gyárrendszer kialakulásával fontossá vált az ipari termelékenység, új foglalkozások alakultak ki, valamint az ipari fejlődés nyomán jelentős mértékben javultak az életfeltételek. Az iparosodást kísérő urbanizáció pedig teret adott a keresetért végzett szakmunkának. Egyre gyakoribbá és tervezhetőbbé vált az egész életre szóló foglalkozás. [5]

Az ipari társadalomban így a tanulás szerepe az élethosszig tartó foglalkozáshoz tartozó tudás megszerzésében összegezhető. Ne felejtsük el, hogy a 19.század közepére több európai országban törvényileg tiltották be a gyermekmunkát. A gyermekkor alapvetően megváltozott, hiszen a gyermekből tanuló lett, olyan iskolás gyermek, aki a kötelezővé tett intézményes nevelés keretein belül szocializálódik, ahol a megfelelő viselkedésformákat és a szükséges ismereteket szerzi meg. Ezek a feltételei a jövendő boldog életének. [6]

## 5. Az információs társadalom és a tanulás

Elsőként vegyük sorra, hogy milyen jellemzői vannak az információs társadalomnak. Az ipari társadalomban indult technológiai forradalmakra egyöntetűen az jellemző, hogy az egyes technológiai innovációk szinte exponenciális mértéket öltve, egyre gyorsabban és egyre nagyobb területen terjednek el.

Gondoljunk csak arra, hogy amíg a távközlésben a telefonnak 75 év kellett, hogy 50 millió felhasználót kössön össze a világban, addig az internet 4 év alatt érte el ezt a mennyiséget, sőt az iPhone-nak csupán 3 hónapra volt szüksége ugyanehhez. [7]

A munkavállalói szektorban további tényező, hogy egyre többször tűnnek fel korábban nem létező foglalkozások, mint például az influencerek, vloggerek vagy az adattudósok. Az is ismert jóslat a Világgazdasági Fórum által készített jelentés szerint, hogy a 2017-ben iskolapadba ülő gyerekek 65%-ának olyan foglalkozása lesz élete során, ami ma még nem létezik. [8] Az ipari társadalmakat jellemző élethosszig tartó foglalkozás egyre ritkább lesz. Helyette az élethosszig tartó tanulás, azaz a life long learning lesz általános.

Megfigyelhető már a gyermekek életében is, hogy a passzív befogadói szerep átalakult aktív felhasználói szereppé. A passzív tartalomfogyasztókból aktív tartalomfejlesztőkké és megosztókká váló gyerekeknek azonban, az információs társadalom által körülölelt digitális világban megjelenő veszélyekkel is meg kell küzdeniük. Nemcsak az online veszélyérzetük nem alakult még ki, hanem védtelenek a hamis információkkal, meghamisított tényekkel szemben is. A digitális környezet változásai nem csak feltételként, hanem kiváló lehetőségként is megjelennek a kisgyermekkori oktatásban. Nem mindegy, hogy a jövő oktatása él-e, jól él-e ezekkel a lehetőségekkel különös tekintettel a kisgyermekkorral kapcsolatban. [9]

Nehéz kérdés, hogy ebben a folyton változó világban hogyan tud az iskola alkalmazkodni az új 21. századi jelenségekhez. Egyáltalán meg tudja-e őrizni korábbi szerepét a tanulásban? Szinte minden pedagógus átélte már azt a zavarba ejtő helyzetet, amikor a rá bízott tanuló bizonyos kérdésekben igen mély tudásról ad tanúbizonyságot, sőt talán azt is, amikor az érdeklődő tanuló olyan kérdést tesz fel, amire tiszteséggel válaszolni csak további információk szerzésével lehetséges.

A konstruktivista pedagógia számol ezzel az előzetes tudással. A tanulási folyamat kimeneteleinél több típust is megfogalmaz. A problémamentes tanulás során nincs ellentmondás az elsajátítandó ismeret és a belső értelmező rendszer között. Megesik azonban, hogy nagyon nagy a szakadék a két előbb említett terület között, vagyis a tanuló a teljes közömbösség állapotába zuhanhat, vagy kizárhatja a befogadásra kínált információt. Nagyon gyakran magolás történik, amikor a tanuló iskolás tudást formál, kettős mércével tekint a megszerezhető tudásanyagra. Az információk meghamisításával, azok kreatív mentésével pedig sikerül a tanulóknak megtartani a saját, eredeti kognitív struktúrájukat. Ezek a jelenségek azt mutatják, hogy a gyerekek kreatív teljesítménye, találékonysága igen fejlett lehet. A legnagyobb változást jelentő tanulási forma a konceptuális váltás, melynek során az információ és a belső rendszer ellentmondása a belső rendszer radikális átalakulásához vezet, melyhez súlyos döntések, néha igen gyötrelmes folyamatok vezetnek. [4]

Mit kínáljuk a jövő tanulóinak, mi legyen a tanulás szerepe az életükben?

Több szervezet is összeállította a saját kompetencialistáját, melyet ezügyben érvényesnek gondol. Az OECD 2005-ben, A Defining 21st Century Skills (21. századi készségek meghatározása) 2012-ben, a Világgazdasági fórum 2015-ben. A legújabb gyűjtemény is a Világgazdasági fórum összeállításában jelent meg. Ebben az első helyen a komplex problémamegoldás, a kritikus gondolkodás mellett a kreativitás, valamint többek között a kognitív rugalmasság szerepel. Ha azonban csupán a kompetenciákra koncentrálunk, akkor könnyen figyelmen kívül hagyhatjuk a tartalom jelentőségét. Ahhoz, hogy tájékozódni tudjunk az információs társadalomban szükségünk lesz a műveltségünkre. A tanulás tartalmi kérdéseiben utat mutathat, ha átgondoljuk, hogy az egyes tudományágak mit

adhatnak hozzá az információs társadalomban való boldogulásunkhoz. Ha a társadalomtudományokra gondolunk akkor ki kell emelnünk az elemző gondolkodásra, forráskritikára, ok-okozati összefüggések feltárására való ösztönzést. A természettudományok a szabályszerűségek és mintázatok felismerésére szoktatnak. A matematika világa a racionális gondolkodásunkat, problémamegoldó képességünket fejlesztheti a saját eszközeivel. Ennek a műveltségnek ki kell egészülnie a digitális kultúra elemeivel: az információfeldolgozás, értelmezés, az algoritmikus gondolkodás és az innovációs készségeink elemeivel. A jövő fiataljait olyan problémák megoldására és olyan technológiák használatára kell felkészíteni, amelyek ma még nem léteznek. Ehhez mindenképpen szükségük lesz a korábban említett kompetenciákra. Legfőképpen pedig olyan pedagógusokra, akik megtalálják a módját, hogy az adott kultúrához, életkorhoz és kompetenciacélokhoz igazítsák a tananyag pontos tartalmát és a tanulás-tanítás módszertanát. [3]

## 6. Összegzés

A tanulmány sorra vette a különböző neveléstörténeti korokban a tanulás változó szerepét. Egészen az antik világ szóbeliségétől kezdve a könyvnyomtatásig megvizsgáltuk, hogy a technológiai újítások hogyan befolyásolták az oktatásról való gondolkodásunkat. Az ipari forradalom csak a kezdete volt annak az egyre gyorsuló exponenciális jellegű fejlődésnek, amelyet a gépekkel kapcsolatban figyelhetünk meg. Az információs társadalom benne a kommunikáció formáinak drasztikus változásaival, az egyre nagyobb teret követelő digitalizációval, alaposan felforgatta a világot, az iskola világát. Ebben a bizonytalan világban kell továbbra is tanulóink számára megőriznünk a kihívásokkal teli, izgalmas tanulás értékét.

## Irodalom


1. Pukánszky, B., Németh, A.: *Neveléstörténet.* Nemzeti Könyvkiadó, Budapest (1997)
2. Platón: *Phaidrosz.* Ford.: Kövesdi Dénes. Atlantisz Kiadó, Budapest (2005)
3. Nemes, O.: *Generációs mítoszok.* HVG Kiadó Zrt., Budapest (2019)
4. Nahalka, I.: *Hogyan alakul ki a tudás a gyerekekben? Konstruktivizmus és pedagógia.* Nemzeti Tankönyvkiadó, Budapest (2002)
5. Németh, A.: *Emberi idővilágok. Pedagógiai megközelítések.* Gondolat Kiadó, Budapest (2010)
6. Szabolcs, É.: *Gyermekből tanuló.* Az iskolás gyermek, 1868-1906. Gondolat Kiadó, Budapest (2011)
7. C. B. Frey, M. Osborne: *The Future of Innovation and Employment.* Global Perspectives & Solutions, Citi GPS. 2015. február 3.
8. T. Doran, N. Lewis: *How to prepare children for the jobs of the future* https://edition.cnn.com/2019/07/30/business/future-education-technology/index.html (utoljára megtekintve: 2019.10.31.)
9. A. Lénárd: *A digitális környezet következményei és lehetőségei kisgyermekkorban.* Iskolakultúra, 29 (2019) 99-114