Oktatóprogram a Sprego táblázatkezelő módszerhez

Csapó Gábor
glerikud.strawhat@gmail.com
DE IK

Sebestyén Katalin
papircikesz@gmail.com
DE IK

Absztrakt: Jelenlegi munkánk alapját a táblázatkezelő szoftverek új oktatási módszere, a SPREGO adja. A módszer lényege, hogy néhány alapvető függvény használatával, egybeágyazásával szinte bármilyen táblázatkezelő probléma megoldható. A Sprego népszerűsítése és könnyebb megértése végett gondoltuk úgy, hogy szükség van egy oktatóprogramra. A demo célja, hogy vizuális reprezentációként szolgáljon a Sprego oktatási módszer által alkalmazott táblázatkezelői képletek működésére.

1. Bevezetés

Előadásunk a Sprego – Spreadsheet Lego – mély metakognitív megközelítésű táblázatkezelő módszerhez készülő demo program bemutatása [13], [19], [22], [24], [25]. A módszer kialakulásánál, tesztelésénél mi is jelen voltunk [21], igaz akkor még a másik oldalon, tesztalanyként, így a módszer hatékonyságát saját tapasztalatainkkal is ki tudjuk egészíteni. Leendő informatikatanárokként pedig teljes mértékben hasznosítani tudjuk.

1.1. Irodai szoftverek

Az irodai szoftverek ismerete, használata alapvető feltétel szinte minden munkakör betöltésénél, de eddig sem kell elmennünk, magánemberként is szükségünk van szöveg-, táblázat- és adatbáziskezelő programok ismeretére.

A különböző irodai szoftvercsomagok mindegyike felhasználóbarát felületet kínál, ezzel a marketingfogással hívva fel magára a figyelmet. De mit is jelent a mi szempontunkból a felhasználóbarát? Egyre több extrafunkcióval bővülő szoftververziók, minden pár kattintásra elérhető, látszólag kényelmes megoldások. A szoftverfejlesztők által készített varázslók, súgók nem nyújtanak kellő támogatást a számítógépes gondolkodás fejlesztéséhez. A szoftvergyártók ezt leginkább azzal magyarázzák, hogy a felhasználók nagy része nem rendelkezik a megfelelő számítástechnikai, matematikai vagy épp statisztikai háttérismerettel. Mindezek következménye, hogy gondolkodás, algoritmus felépítése nélkül kattintgatással, lesz, ami lesz alapon – Trial-And-Error Wizard-based [24] – használjuk ezeket a szoftvereket. És vagy sikerül, amit szerettünk volna, vagy nem, a kapott output nem feltétlenül az eredeti probléma megoldása.

1.2. Informatikaoktatás - problémamegoldás

A Debreceni Egyetem (a táblázatkezelő programok kapcsán végzett) felméréseinek eredményeként megállapítható, hogy a grafikus környezet használatára helyezett hangsúly, illetve a rosszul megválasztott – szoftverfejlesztők által ajánlott felület alapú megközelítés – módszerek okozzák a képletbeviteli hibák többségét. [14], [15], [17] [23]

A grafikus felületen történő navigáció, kattintgatás nem igényel komoly gondolkodást, mély metakognitív megközelítésű tevékenységsorozat végrehajtását a felhasználóktól.

A hazai informatikaoktatás is a szoftverhasználatra épül, kihagyva a megértés fázisát. Hiába folyik 20 éve informatikaoktatás [3], az még ma is kiforratlan Magyarországon. Európában több ország rövidebb múlttal rendelkezik az informatikaoktatás terén, ennek ellenére a PISA felméréseken [8] sokkal jobb eredményeket értek el, ahol a számítógépes írástudást, navigációs képességet tesztelték. Az egyetemi és a PISA felmérés egyértelműen megmutatja, hogy szükség van nézőpontváltásra. A hazai informatikaoktatás feladata nem a folyamatosan változó – Microsoft Office 2003, 2007, 2013… - szoftververziók felületének bemutatása lenne, hanem környezettől független, egyfajta univerzális gondolkodásmód – számítógépes gondolkodás és algoritmikus készség – kialakítása és továbbfejlesztése [9], [19].

A Sprego számítógépes gondolkodásra épülő módszer, mely táblázatkezelő programok verziószámaitól, gyártóitól független és megfelel a 2013-ban megjelent IEEE&ACM jelentésben [16] meghatározott három szigorúan egymásra épülő számítógépes ismeretszintnek:

- familiarity: a megértés szintje, vannak-e ismereteink a problémával kapcsolatban, mik azok.
- usage: a koncepció, módszer alkalmazása.
- assessment: több nézőpont közül a megfelelő kiválasztása, tudatos választás, a Miért? kérdésre ad választ.

A hazai informatikaoktatásban a második szintre fektetjük a hangsúlyt, a különböző felhasználó szoftverek használatát írja elő a Kerettanterv [11]. Vegyük példának a szövegszerkesztést, a betűk, bekezdések formázásától kezdve különböző stílusokon, wordarton át számos program adta lehetőséggel ismertetjük meg a diákokat, de nem építünk neki alapot. Hiányzik például az a tipográfiai alap, amellyel felismerhetik a problémát, hogy majd a tipográfiai szabályoknak megfelelően formázhassák a szöveget. Nem ismerik meg, hogy miért fontos, hogy egy szöveg jól formázott legyen, milyen hátrányokkal kell szembenézniük a későbbi szerkesztés során, ha teletűzdelik a szöveget felesleges bekezdésekkel, vagy esetleg kézi számozást használnak. Mivel ezek a háttérismeretek hiányoznak, így a harmadik szintre sem juthatnak el a hallgatók, nem ismerik fel a problémát, így az sem várható el, hogy különböző megoldási lehetőségek közül kiválaszthassák a megfelelőt.

A táblázatkezelés oktatása kapcsán sem a különböző formázásokra, és sok száz képlet megtanítására van szükség, ugyanis ez utóbbi szinte lehetetlen. A korábban említett felhasználóbarát környezet magába foglalja például az összevont függvényeket, mint a DARABTELI(), DARABHATÖBB(), SZUMHA(), SZUMHATÖBB(), ÁTLAGHA(), ÁTLAGHATÖBB(), és társai, melyek argumentumlistája, a hasonlóságok ellenére eltérő, valamint ezen függvények számának folyamatos emelkedését.

A felhasználó nem képes, és az igény sincs meg benne, hogy ezeknek a függvényeknek az argumentumait és az argumentumok sorrendjét megjegyezze. A Sprego módszer esetén csupán néhány elemi függvény ismeretére van szükség a középhaladó szint eléréséhez. Ez a minimális darabszám igazolhatja Booth-t [2], aki szerint a funkcionális nyelvek, melyek közé tartoznak a táblázatkezelő programok nyelvei is [7], első programozási nyelvként taníthatóak. Ahogy minden mesterséges nyelvre jellemző a táblázatkezelők nyelvére is, hogy a szókészletét és nyelvtanát tudatosan tervezték, ebből kifolyólag véges készletről van szó. Aki egy picit is jártas a programozásban, az tudja, hogy nincs sok szabály, utasítás, de azok szintaktikájára nagyon kell figyel-

ni. És ahogy a programozás oktatásánál, folyamatosan vezetjük be az új ismereteket, sosem árasztjuk el a diákokat, mindig csak annyi információt adunk közre, amennyi a probléma megoldásához szükséges, hogy ne zavarjuk meg a gondolkodási folyamatban őket. Az táblázatkezelő programok esetén ez egy nehéz feladat, mert a szoftvergyártó cégek a teljes függvénykészlet használatát javasolják, már a kezdetektől, ugyanakkor a tanítás során a hozott ismerettel is számolni kell.

2. Sprego

A minimalista elvet követve, néhány alapfüggvény ismeretével szinte bármilyen lekérdezést végre tudunk hajtani. A módszer kitalálói 3 függvénycsoportot határoztak meg (**1. táblázat**) [22].

	1. csoport	**2. csoport**	**3. csoport**
1.	SZUM()	INDEX()	KICSI()
2.	ÁTLAG()	HOL.VAN()	NAGY()
3.	MIN()	HIBÁS()	SOR()
4.	MAX()		OSZLOP()
5.	BAL()		ÉS()
6.	JOBB()		VAGY()
7.	HOSSZ()		NEM()
8.	SZÖVEG.KERES()		ELTOLÁS()
9.	HA()		TRANSZPONÁLÁS()
10.			KEREKÍTÉS()
11.			VÉL()
12.			INT()

1. táblázat Sprego 3 függvénycsoportja

A Debreceni Egyetem felmérésében az egyik feladat az volt, hogy sorolják fel a hallgatók az általuk legfontosabbnak tartott függvényeket (15 db). Ezek összegzése után a kutatást végzők 99 féle függvényt összesítettek. Ha belegondolunk ezek a hallgatók egy oktatási rendszerben, a NAT [12] és a Kerettanterv [11] szerint haladtak. Nehéz a korábbi ismeretekre alapozni, ha ennyire sokféle háttérrel rendelkeznek.

A Sprego nem várja el, hogy 99 függvényt ismerjünk, a legtöbb felhasználó igényeit bőven kielégíti az első két oszlop tizenkét függvénye (**1. táblázat**). A három kategória olyan függvényeket tartalmaz, melyeket egymásba ágyazva – akárcsak a matrjoska baba (**1. ábra**) – táblázatkezelői keretek között, szinte bármilyen kérdésre választ kaphatunk.

1. ábra Matrjoska baba

2.1 Előnyei

A Sprego legnagyobb előnye, hogy az ismeretszintek legalsó lépcsőfokán kezd, és onnan építkezve képes eljutni a legfelső szintre. A felmérések, a tapasztalatok azt mutatják, hogy az **1. táblázat**ban felsorolt függvények begyakorolt alkalmazása sokkal hatékonyabb, kevésbé időigényes, mint a számos előre definiált összevont sablonok használatánál. A Panko által végzett felmérés szerint a táblázatkezelővel készített dokumentumok 84%-a hibás [5], [6], [20], – 1995 óta létrehozott dokumentumokra értendő, a kapott százalék az évek során elvégzett különböző vizsgálatok átlageredménye –, mely a felhasználók felületes tudásán alapszik. Ez a felületes tudás onnan ered, hogy a szoftverfejlesztők, marketingesek a legújabb szoftververziók új szolgáltatásait, jelen esetben a probléma specifikus sablonok használatát szorgalmazzák. A hibás dokumentumok nem a végfelhasználóknak okoznak gondot. Ezen dokumentumok többsége nagyvállalatok tulajdonában vannak, a nem megfelelő dokumentum tervezés, a nem megfelelően használt függvények, munkalapok nagy károkat okozhatnak a gazdasági, pénzügyi szférában. 2012-ben a JPMorgan Chase & Co. vállalat 6 billió dollár veszteséget tudhatott magáénak hibás dokumentumoknak köszönhetően [10]. A hibák elkerülhetőek odafigyeléssel és a hangzatos, kényelmes összevont függvények helyett az egyszerű függvények egymásba ágyazásával.

A Spregonak és az egyszerű függvényeknek számos előnye van.

- Nem gyártó, verzió specifikusak. A három legnagyobb táblázatkezelő, a Microsoft Excel és az OpenOffice és LibreOffice Calc, bármelyik verziója között teljes az átjárás.
- Nem kell foglalkozni új függvények argumentumlistájának megértésével, debuggolásával, megjegyzésével.
- Az összevont függvények valamilyen speciális probléma megoldására készültek, lehetetlenség minden problémára egy sablont megalkotni, ráadásul ezeket megtalálni, megjegyezni is nehéz. Egyszerű elemekből építkezve sokkal több szabadsággal rendelkezik a felhasználó. A módszer neve is innen ered - Spreadsheet Lego. A Legohoz hasonlóan, általános építőelemek használva és kombinálva lehet összetett objektumokat létrehozni.
- Fejleszti a számítógépes gondolkodást, az algoritmikus készséget.
- Egyszerű logikán alapszik, elég egyszer megérteni, mondhatni olyan, mint a biciklizés. Ezt a Debreceni Egyetem felmérése is igazolja. Az Egyetem három mérést végzett el, egyet a

Sprego előtt, az érettségi után, az egyetemi tanulmányok megkezdésekor (5–10%), egyet a módszer megtanulása után (60–70%), majd 1 évvel később is megismételte (40–50%). Ezek az eredmények bizonyítják, hogy a Sprego módszer lényegesen hatékonyabb, mint a korábban alkalmazott felületi megközelítésű módszerek, és tartós tudás szerezhető vele [25].

3. A vizuális programozás

Napjainkban egyre népszerűbbé válik egy újfajta programozási módszer, amelyet használva (ahogy azt a neve is sugallja) a szoftverfejlesztő vizuális felületen építi fel a program logikáját hagyományos szöveges programozási utasítások megadása nélkül. A tapasztalat azt mutatja, hogy az objektumorientált programozást követően ez a módszer tekinthető a szoftverfejlesztés egyik következő nagy előrelépésének.

A vizuális programozás során a programozás teljesen grafikusan történik és a fejlesztő lényegében egy kódgenerátort használ, amely legtöbb esetben egy előre megírt motorhoz kapcsolódik és különböző programozási nyelvekre fordítja le a vizuálisan felépített logikát. Így a programozás folyamata gyorsabbá, letisztultabbá, valamint átláthatóbbá válik. A vizuális kódépítésből adódóan nem kell az adott nyelv szintaktikájával foglalkozni és így számos kezdő programozó könnyebbnek találja a vizuális programozás elsajátítását a hagyományos nyelvekkel szemben. Az sem ritka eset, hogy több éves programozói múlttal rendelkező fejlesztők választják a vizuális szoftverfejlesztést annak előnyei miatt. Bár a vizuális programozás egyik fő felhasználási területe a játékfejlesztés, tökéletesen alkalmas más alkalmazások fejlesztésére is.

Ezek szerint megjelent egy módszer, amely kiszorítja a hagyományos programozást a piacról? Korántsem ez a helyzet. Bár vitathatatlan előnyei vannak a vizuális programozásnak, ezek bizonyos hátrányokkal járnak együtt: a fejlesztés egy korlátolt rendszerben történik, amelyben lehetőségeink sokszor az azt készítő fejlesztőktől függ. Vannak esetek, amikor lehetőségünk van hagyományos, imperatív programozásra váltani, ha határokba ütközünk, de sajnos ez jelenleg nem tekinthető általánosnak. Jó hír azonban, hogy az újonnan megjelenő környezeteknél ennek kezelése javuló tendenciát mutat.

Mivel a vizuális programozás egy újfajta megközelítés az algoritmikus problémák megoldására és nem egy új programozási nyelv, ezért nem beszélhetünk egységes megjelenésről sem. Számos fejlesztőkörnyezetben van lehetőség vizuálisan fejleszteni, de ezek a szoftverek eltérő rendszereket használnak. Így jogosan merül fel a kérdés, hogy van-e átjárhatóság az eltérő vizuális fejlesztőkörnyezetek között? Elegendő csupán egyet megtanulni és már mindegyikben elboldogulunk? A válasz: igen is és nem is. Igen, mert a vizuális programozás tisztán az algoritmizálásra fókuszál és az algoritmikus készséget minden környezetben (akár hagyományos programozás során is) képes kamatoztatni a fejlesztő. Nem, mert az eltérő környezetek eltérő felületeken különböző modulokat alkalmaznak az építkezésre, és így ami az egyik szoftverben már egyértelmű volt a programozónak, lehet, hogy a következőben más megközelítést igényel. Jó hír azonban, hogy a megosztottság ellenére 4 fő kategóriába csoportosíthatóak a vizuális programozást képviselő módszerek, amelyek fejlesztési folyamatai közel azonosak: viselkedés alapú, eseményutasítás alapú, blokk alapú és csomópont alapú. Optimális esetben a fejlesztőkörnyezet átjárást biztosít ezek között a kategóriák között és lehetővé teszi a módszerek párhuzamos alkalmazását a fejlesztési folyamat során.

A viselkedés alapú programozás a legegyszerűbb formája a vizuális programozásnak és egyben a legkorlátoltabb is. Ezzel a módszerrel a programozó előre megírt utasításokat, viselkedéseket rendel a képernyőn megjelenő objektumokhoz és bizonyos paramétereken túl nincs módja azok módosítására. Hatékony a fejlesztési folyamat meggyorsítására és prototípusok készítésére, de önmagában ezzel a módszerrel összetettebb alkalmazások nem készíthetőek.

Az esemény-utasítás alapú megközelítés már a fejlesztő kezébe adja a szabadságot, hogy eldöntse: előre megírt eseményekre miként reagáljon a szoftver. Az egyes objektumokhoz eseményeket rendelhetünk, majd definiálhatjuk az ezek teljesülésekor végrehajtandó utasítások halmazát. Ezzel a módszerrel közel bármilyen logika felépíthető. Az ilyen rendszert alkalmazó fejlesztőkörnyezetekre jellemző, hogy az események eléggé generikusak ahhoz, hogy a legtöbb szituációban alkalmazhatóak legyenek.

A blokk alapú programozás már leképezi a hagyományos programnyelvek szintaktikáját és ezeket blokkokba rendezve teszi lehetővé az algoritmusok felépítését. Bonyolultságát tekintve ezt nevezhetjük a legnehezebb módszernek, ellenben képes az alapjául szolgáló hagyományos programnyelv valamennyi lehetőségének kihasználására. Népszerűségét tekintve ez a legkevésbé elterjed formája a vizuális programozásnak, mivel a fejlesztési folyamat lassú és sokszor átláthatatlan grafikus kódot eredményez komplexebb projektek esetén.

Az utolsó kategória a csomópont alapú vizuális programozás. Akárcsak a blokk alapú módszer, ez is törekszik az alapvető építőelemek megragadására. Nevét onnan kapta, hogy a fejlesztés során csomópontokat alakít ki a fejlesztő és azokat kapcsolja össze az algoritmusnak megfelelően, és végül a grafikus kód egy folyamatábrát mintáz. Ezt a módszert gyakran használják a professzionális környezetek annak rugalmassága és hatékonysága miatt.

Ahogy arról korábban szó volt, a fenti módszerek megoszlanak a különböző fejlesztőkörnyezetek között. A viselkedés alapú módszert általában kíséri valamilyen másik formája is a vizuális programozásnak a lehetőségek kiszélesítése végett (pl.: az esemény-utasítás). Ilyen környezetek közé tartozik a Game Maker [37] és a Construct 2 [31]. A blokk alapú megjelenítéssel az oktatásban is gyakran találkozhatunk, ugyanis a népszerű Scratch [28] program is erre épül, de a Stencyl [35] fejlesztőkörnyezet is erre alapoz. A csomópont alapú formával olyan, szakmailag is régóta elismert, környezetekben találkozhatunk, mint az Unreal Engine 4 [36], illetve a Unity 3D Playmaker [26] kiegészítője.

3.1 Construct 2

Az oktatószoftver fejlesztésére a Scirra cég Construct 2 vizuális fejlesztőkörnyezetét választottuk annak rugalmassága és hatékonysága miatt. A környezet HTML5 alapú alkalmazások fejlesztésére képes, illetve harmadik féltől származó wrapperek segítségével célplatformok széles körét támogatja. A szoftver jól dokumentált. Teljes angol nyelvű kézikönyv [33] áll rendelkezésre, amelyet kiegészítenek a fejlesztők által írt segédletek [34], valamint az aktív és segítőkész közösségi fórum [32].

A fejlesztési folyamat viselkedés- és esemény-utasítás alapú megközelítésekre épít. A piacon jelen lévő vizuális fejlesztőkörnyezetek közül ennek a szoftvernek tekinthető a legkiforrottabbnak és leghatékonyabbnak az esemény-utasítás rendszere.

A fejlesztés úgynevezett „elrendezéseken” és „eseménylapokon” történik. A programozó a létrehozott objektumokat elhelyezi az elrendezéseken, amelyek a felhasználói felület felépítésére szolgálnak. A kialakított elrendezés úgy fog megjelenni futtatás során a képernyőn ahogy az a

szerkesztőfelületen látható. Ezekhez az elrendezésekhez kapcsolódnak az eseménylapok, amelyeken a program logikáját felépíti a fejlesztő eseményeket, majd az azokhoz kapcsolódó utasításokat definiálva. Itt mutatkozik meg a környezet egyik legnagyobb erőssége: a rugalmas vizuális kód kialakítása. Lehetőségünk van úgy rendezi, egymásba ágyazni az eseményeket és feltételeket, hogy valamennyi esetet kezelni tudjuk. A környezet számos elemét alkalmazza a hagyományos programozási nyelveknek, így a változók, tömbök, függvények, I/O folyamatok kezelésének lehetősége magától értetődő.

A szoftver ingyenesen elérhető bizonyos korlátozásokkal [30], amely nem kereskedelmi célú használatra és oktatásra egyaránt alkalmas. Munkánk során saját, Personal Edition licenszet használtunk.

4. Oktatószoftverek

Évente több ezer oktatószoftver készül, leginkább a nyelvoktatás területén találkozhatunk velük. Sokan úgy gondolják, hogy ezek a programok helyettesítik a tanárt, de a tapasztalatok nem ezt mutatják, ezért is nevezik inkább számítógéppel segített tanulásnak vagy épp tanulást segítő programoknak, eszközöknek. Egyfajta kiegészítő lehetőségként tekinthetünk ezekre a szoftverekre, melyek segítik a megértést az interaktív szemléltetésnek köszönhetően. Nagyon sokféle programmal találkozhatunk melyeknek céljaik, feladataik eltérőek, de minden esetben a tanulási folyamat elősegítése az a végső cél, amelyért a program létrejött. A dolgozat címe, az oktatóprogram kifejezés kicsit félrevezető lehet. A szakma már réges-régen meghatározta az oktatószoftverek típusait, melyek céljaikban eltérőek – gyakoroltatás, szemléltetés, szimuláció stb [1]. Ezzel szemben ma már mindenre ráhúzzák ezt a kifejezést, ha bármiképpen is kapcsolódik az tanítási-tanulási folyamathoz.

> *„A jövő útját tehát részben abban látjuk, hogy a különböző, főleg mechanikus gyakorlást, sulykolást igénylő, algoritmizálható feladatokat ki kell, ki lehet vinni az osztályteremből és átruházni a komputerre.”*[1]

Ahogy azt majd 30 éve megfogalmazta Kecskés István, az még ma is irányadó. A Sprego módszer feladata és lényege az algoritmizálás, az algoritmusok megépítése, ezek gyakorlása, egy mélyebb tudás elérése, felszínre hozása, amely algoritmusok sémákká alakulhatnak. Ezen cél eléréséhez azonban szükség van a táblázatkezelők mesterséges nyelvének, logikájának megértésére és elsajátítására. Az interaktivitás az az eszköz, amely nagymértékben befolyásolja a megértést.

A kutatómunkánk céljaként azt a feladatot tűztük ki, hogy minél szélesebb körben ismerjék meg a Sprego módszert, tanárok, oktatók fedezzék fel a Sprego egyszerűségét, taníthatóságát. Csernoch Mária egy könyvet is kiadott a módszerről, mely lépésről lépésre építi fel a tömbképleteket, ennek ellenére nem túl bizalomgerjesztő, ha valaki meglát egy 2 soros függvény egybeágyazást (**2. ábra**).

```
{=HA(HIBÁS(KICSI(HA(G2:G251<>G1:G250;G2:G251);SOR()-
1));"";KICSI(HA(G2:G251<>G1:G250;G2:G251);SOR()-1))
```

2. ábra Egy többszörösen egymásba ágyazott képlet

1 Kecskés István: Mikroszámítógépek használata az idegennyelv-oktatásában. Tankönyvkiadó, Budapest (1987)

Ezért is gondoltuk úgy, hogy szükség van egy vizuális szemléltetőeszközre, amely az egyszerűtől az egyre összetettebb problémákig nyújt segítséget. A program használói közelebb kerülhetnek a táblázatkezelők hátterében meghúzódó logikai sémához, egy-egy függvény hogyan működik – lépésről lépésre, majdnem szájbarágósan jelenik meg. Különböző beépített funkciók segítik az algoritmus felépítését, tevékenységek sorrendiségét.

4.1 A Sprego szoftver működése

Az általunk fejlesztett Sprego oktató-demo szoftver kialakításánál kiemelt szempont volt a könnyű kezelhetőség és a felhasználói felület optimalizálása érintőképernyőkkel rendelkező eszközökre is. Mindemellett törekedtünk a látványos és esztétikus megjelenítésre és a webes verziónál a lehető legkisebb méretre is.

3. ábra A Sprego program főmenüje

A program indítását követően megjelenik a fő menü, ahol a felhasználónak lehetősége van kiválasztani az általa megvizsgálni kívánt problémát (**3. ábra**), a képernyő bal oldalán található gombok segítségével. Ekkor még a felület többi része le van tiltva és el van rejtve, hogy az elérhetetlen beállítások ne zavarják meg a felhasználót.

4. ábra Biró Zsuzsanna festett grafikái

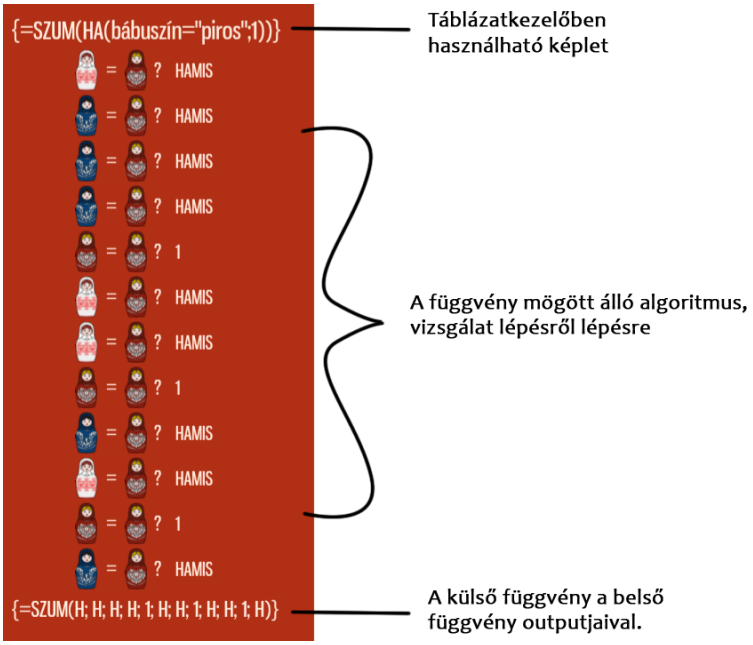
Az oktató-demo programunk a matrjoska babákat használja főszereplőként. Azért esett a választás ezekre a babákra, mert a függvények egymásba ágyazása kiválóan szemléltethető a babák összeépítésével. Kiindulásként Biró Zsuzsanna grafikus által készített babákat (**4. ábra**) használtuk, majd ezek újraszínezésével létrehoztunk egy saját baba készletet (**5. ábra**).

5. ábra Újraalkotott babák több színben

A probléma kiválasztás után a szoftver megvizsgálja, hogy milyen beállítások rendelhetőek az adott esethez és ehhez mérten engedélyezi a menü jobb oldalán található beállítások egyikét: a bábu színének kiválasztását, ahogy az **3. ábra** képén is látszódik vagy a bábu méretének változtatását (jelenleg rejtett, mert egyik probléma sem igényli). Amennyiben a program megkapta a továbblépéshez szükséges inputokat, a probléma bemutatását indító gomb – Start – megjelenik. Ez az utolsó lépés csak a színválasztás esetén áll fenn, ugyanis a méretváltoztatásnál az alapértelmezett mérettel is tud dolgozni a program, így akkor egyből megjelenik az indító gomb.

A probléma és a hozzá szükséges inputok kiválasztása után a szoftver megjeleníti az adott problémához tartozó elrendezést és kis várakozás után elindítja a demot. A felhasználó a képernyő bal felső sarkában láthatja a választott probléma megoldásához használt táblázatkezelő képletet és alatta az egyes végrehajtási lépések eredményeit (**6. ábra**). Ezek alatt kerül megjelenítésre a demo végén az eredményül kapott érték. Ez a részegysége a felületnek egyfajta, grafikus képletkiértékelőként funkcionál, vizuális segítséget és magyarázatot nyújtva az aktuális probléma megoldására szolgáló képlet működésére. A képernyő középső részén látható az animált demo, amely mindig az adott probléma jellegzetességét mutatja be, valamint a bal felső táblázatkezelő képlet működésének reprezentációjaként is szolgál. Végül pedig a képernyő jobb széle tartalmazza a

vezérlőgombokat fentről lefele haladva a következő sorrendben: vissza a menübe, újrakezdés, szünet és lassítás.

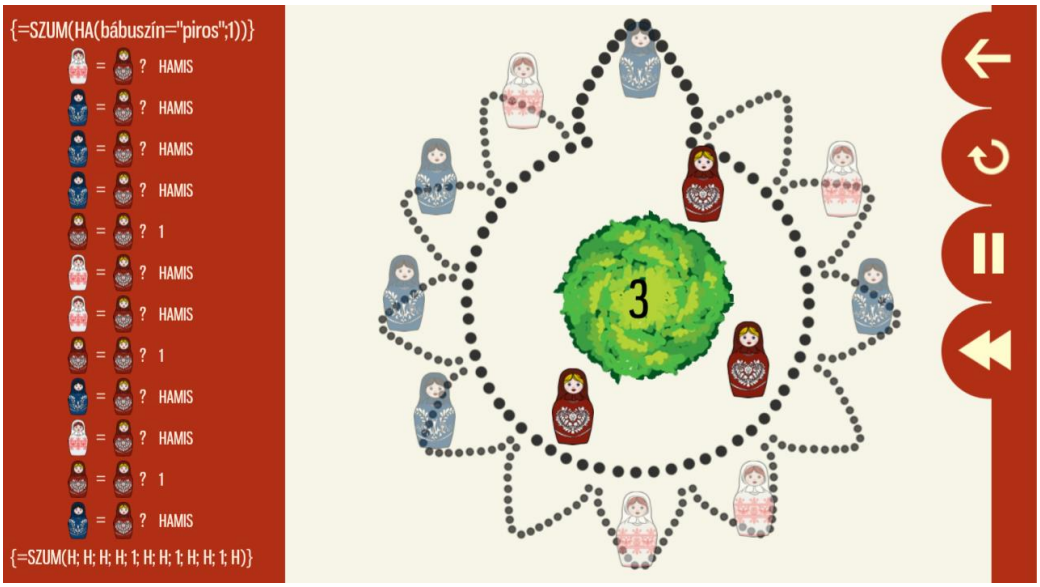


6. ábra Kiértékelő sáv jelmagyarázattal

A menübe való visszatérést követően a felhasználónak lehetősége van egy másik problémát kiválasztani vagy a jelenleg kiválasztottat (a program megjegyzi az előzőekben választott problémát) új beállításokkal indítani. Az újrakezdés gomb arra szolgál, hogy az éppen folyó demot újraindítsa a beállítások megtartása mellett. A szünet gomb segítségével megállíthatjuk a bemutatót, illetve újbóli megnyomásával elindíthatjuk. Ezt a funkciót a probléma könnyebb elemezhetősége miatt implementáltuk, hogy a felhasználónak legyen ideje akár lépésenként megállítani a prezentációt és megvizsgálni a történéseket. Amikor a bemutató szüneteltetve van, a lassítás gomb nem elérhető. A lassítás funkció a bemutató sebességét csökkenti, hogy könnyebben követhető legyen megállítás nélkül, valamint újbóli megnyomásakor az eredeti sebességre kapcsolható vissza.

4.1.1 Probléma 1.

Az első probléma (**7. ábra**), a „Hány azonos színű baba van?” megfelel a {=SZUM(HA())} egymásba ágyazott függvénynek. A vizsgálat során a középső területen körkörösen helyezkednek el a babák, amelyek körben táncolva áthaladnak egy kapun és azok a bábuk, amelyek megfelelnek a feltételnek – a felhasználó által választott színnek – belépnek egy belső körbe, ahol tovább folytatják körkörös mozgásukat (**7. ábra**).



7. ábra Hány azonos színű baba van?

A kapuba – itt történik meg a logikai kifejezés kiértékelése – érkezés pillanatában megáll a bemutató egy rövid időre, hogy éreztesse a felhasználóval, hogy most vizsgálat történik. Amikor valamennyi baba megvizsgálásra került, a program megszámolja a belső körben – feltételnek megfelel – babákat és számukat kiírja középre. Mindezek közben a bal oldali, kiértékelő sávban minden egyes vizsgálatkor megjelenik az éppen vizsgált bábu kicsiben, feltételbe foglalva a választott színű bábuval. A vizsgálat végén (a késleltetés után) a feltétel eredménye is megjelenik. A bemutató végén pedig kiírásra kerül a {=SZUM()} függvény, amely a HA() függvény külső függvénye, és a HA() függvény által visszaadott vektor értékeit kapja meg inputként. A megjelenítésre az összeadáskor kerül sor. Ezt követően a felhasználó újraindíthatja a bemutatót, vagy visszatérhet a menübe a vezérlőgombokat használva.

4.1.2 Probléma 2.

A második probléma (**8. ábra**), a „Melyik házban lakik a baba?” megfelel az {=INDEX(HOL.VAN))} egymásba ágyazott függvényeknek. A menüben 9 különböző színű baba közül választhatnak a felhasználók. A különböző babákat a HOL.VAN() függvény harmadik argumentuma – egyezés típusa, lehet 1, 0, −1, jelen esetben a 0-ás egyezés indokolja. A kiválasztás után elindíthatjuk a szemléltetést. Ahogy az a **8. ábra** mintáján is látszódik, egy utca jelenik meg 9 házzal, minden ház előtt egy-egy baba áll. Az utca bal felső sarkából egy postást szimbolizáló baba indul el, amely megáll minden ház előtt és megvizsgálja, hogy az ott lakó bábu színe megegyezik-e a választott színnel. Amennyiben nem, a vizsgálat tovább folytatódik. Ha a postás megtalálja a keresett babát, odamegy hozzá. A bal oldali képletkiértékelőben követhetőek az egyes házaknál tett vizsgálatok, illetve a célhoz érkezés után a házak vektora is megjelenítésre kerül a vizsgálatok mellett. A ház kiválasztása az INDEX() függvénnyel történik, melynek egyik argumentuma a házak vektora, a másik pedig a HOL.VAN() függvény által visszaadott sorszám.

8. ábra Melyik házban lakik a baba?

4.2 Technikai háttér

A fejlesztést a Construct 2 vizuális programozást használó környezetben végeztük el. Ebben a részegységben a szoftver technikai megoldásait részletezzük, különös tekintettel a fejlesztőkörnyezet sajátosságaira.

A program HTML5 alapú és ebből adódóan elsődleges célplatformja a web. Emellett asztali verziókat (Windows, Linux és MAC) is készítettünk belőle az NW.js [29] wrappert használva, valamint mobil eszközökre, Android alkalmazásként is futtatható Android 4+ rendszert használó készülékeken. A mobil verzió csomagolásához az Intel XDK [27] szolgáltatásra támaszkodtunk.

A grafikus megjelenés elkészítéséhez az Adobe Photoshop, Paint.NET és Gimp szoftvereket használtuk. Teszteléseink során a következő modern böngészőprogramokban futtattuk a webes változatot: Chrome, Opera, Firefox, Edge. Az asztali verziót Windows 10 x64 rendszeren, míg az Androidra készültet Samsung Galaxy Grand Prime és LG G Pad 8.0 eszközökön teszteltük.

A felhasználói felülettel történő interakciót érintéssel kezeljük. PC-n az egér kattintását a program érintésként értelmezi. Ezzel a különböző eszközök közötti inputok eseményeit leegyszerűsíthettük. A szoftver egyes elrendezéseinek betöltődésekor a szöveg objektumokra a Gnuolane Regular webfont kerül alkalmazásra, így a webes megjelenítés is tartalmazza a betűtípust.

A menü egyfajta beállításközpontként is szolgál a program számára, ugyanis az itt megadott beállításokat – választott probléma, választott szín, választott méret (ez utóbbi a jelenleg implementált probléma bemutatásokhoz nem választható) – globális változóként adja tovább a szoftver további részeinek. Minden problémához hozzá van rendelve egy tulajdonság, ami megadja, hogy a felhasználó milyen beállításokat végezhet el annak választásakor. A szoftver ez alapján engedélyezi az ezekhez tartozó felületeket (külön rétegeken tárolva), illetve a beállítások megtörténését vizsgáló algoritmus is ezek alapján dönti el, hogy mikor jeleníti meg és engedélyezi az indító gombot.

Fontos megjegyeznünk, hogy a menübe való belépéskor a korábbi beállítások (a választott függvény kivételével) nullázásra kerülnek.

4.2.1 Probléma 1.

A {=SZUM(HA())} problémát bemutató elrendezés betöltődésekor szintén nullázásra kerülnek bizonyos értékek, amelyeket a vizsgálat során a szoftver felhasznál. A vizsgált babák darabszámát változóban, a feltétel egyes eredményeit tömbben tároljuk. A betöltődéskor meghívásra kerül egy eljárás, amely a körben pozícionált babák színét véletlenszerűen választja ki. Ezt követően a bal oldali sávban megjelenik a választott probléma belső függvénye és legenerálásra kerül a babák számával megegyező számú kis babákat mutató feltétel, majd ezek láthatóságát kikapcsoljuk. Erre a lépésre egy hiba megkerülése miatt van szükség, miszerint a webfont alkalmazása az indítás után létrehozott szövegobjektumokra csak késve történik meg. Ezt kerüli meg a feltételeket jelző elemek előre legenerálása és elrejtése. A körben álló bábuk körkörös mozgása egy láthatatlan központi objektum körül történik, amelynek a pozíciójához rögzítjük a bábukat, így elegendő csak a központ forgatását kezelni.

A feltételek vizsgálatára egy szintén láthatatlan objektumot alkalmazunk, amely a kapuk között helyezkedik el. Ezzel az objektummal minden egyes baba átfedésekor meghívásra kerül a vizsgálatot végző algoritmus, amely megállítja a kör forgását, hozzáad egyet a megvizsgált bábuk számához és megjeleníti a bal oldali sávban az éppen következő feltételt a kis babákkal. Ha a feltétel teljesült, akkor a szoftver kiírja annak eredményét – 1 – a megadott sorba (ha a feltétel nem teljesül, akkor az alapértelmezett visszaadott érték HAMIS) (8. ábra), majd eltárolja ezt az értéket a tömbben és meghívja a baba belső körbe mozgatását végző eljárást, majd a mozgatás után újraindítja a központ forgását. A feltétel nem teljesülése esetén annyi eltérés van, hogy a baba nem kerül bemozgatásra a belső körbe, helyette az elhalványításra kerül, jelezve, hogy a baba nem felelt meg a feltételnek.

A bemozgatást végző eljárás feloldja a baba rögzítését a központhoz, majd a „Bullet” viselkedést használva a kör középpontja felé indítja el a babát, aminek megtett útját egy másik algoritmus figyeli és a határérték elérése után megállítja azt, majd ismét hozzákapcsolja a központ pozíciójához.

Amint az összes baba megvizsgálásra került, a középpont mozgását megállítjuk és meghívjuk az összeszámolást végző eljárást, ami megszámolja a belső körben lévő babákat és a számokat kis késleltetéssel megjeleníti. Ennek célja, hogy a számolási folyamatot a felhasználó is szemmel tudja követni. Végül pedig a bal oldali sáv aljában egy másik eljárás segítségével kiírásra kerül a feltételek eredményeit tartalmazó {=SZUM()} függvény a tömb értékei alapján.

Az egyes gombok kezelése ugyanúgy történik, mint a szoftver más területein. A vissza a menübe gomb meghívja a menü elrendezést (és annak betöltődésekor nullázódnak a beállítások). Az újraindítás gomb egyszerűen újra meghívja az aktuális elrendezést. A szünet és lassítás gombok pedig a „Set time scale” utasítást használva megállítják vagy lassítják az idő múlásának sebességét a programon belül.

4.2.2 Probléma 2

Az {=INDEX(HOL.VAN())} problémát prezentáló elrendezés betöltődésekor ugyancsak visszaállítjuk néhány változó alapértelmezett értékét. Ezt követi a lakosok (a házak előtt álló babák) színeinek megkeverése. Ebben az esetben figyelni kellett arra, hogy az előző problémától eltérő-

en itt minden bábu (mind a 9 színű) megjelenjen és mindegyik maximum egyszer forduljon elő. Ennek kezelésére tömböt használunk. Kis késletetés után elindul a postás és egészen addig halad előre, ameddig nem kerül átfedésbe egy, a már előző probléma bemutatásának működésénél ismertetett láthatatlan vizsgálati zónával. Ekkor megnézi a program, hogy mennyi bábut vizsgáltunk meg eddig és ezt az értéket összeveti a színeket tartalmazó tömb aktuális mezőjével. Amennyiben a tömbben szereplő adat megegyezik a választott színnel, megtaláltuk a keresett bábut. Ellenben a postás tovább folytatja útját a következő vizsgálati zóna irányába. A postás mozgatásában két különleges esetett kellett kezelni. Az egyik, amikor eléri a felső házsor végét. Ilyenkor a vizsgált bábuk darabszáma alapján (az 5. bábu megvizsgálása után) irányt vált és lemozog az utca alul található részére, ahol egy újabb irányváltás után hasonlóan mozog, mint a felső házsornál az ellenkező irányban. Minden mozgásváltozásnál a postás objektum külön animációt jelenít meg. A másik különleges mozgás esete akkor aktiválódik, amikor a postás megtalálta a keresett babát. Ekkor irányát felfele megváltoztatja és megközelíti a babát. Mindkét esetben a postás megállítását az általa megtett út vizsgálatával végeztük.

A kiértékelő hasonlóan működik, mint az előző probléma esetében, néhány eltéréssel: Csak 9 sor kerül megjelenítése (a babák számából adódóan), illetve már az elrendezés betöltődésekor meg van jelenítve a babák vektora. Elindul az összehasonlítás. Minden összehasonlítás után kiírjuk a megvizsgált baba vektoron belüli sorszámát. Mindaddig végezzük az összehasonlításokat, amíg meg nem találjuk a keresett babát. Miután a postás megtalálta a keresett bábut, kijelöljük a helyes baba teljes rekordját, majd az egyes vizsgálati sorok mellett megjelennek a babákhoz tartozó házak képei, majd ezt követően a kiértékelő sáv alján az eredmény adó képlet az INDEX() függvénnyel és ház képe is, ami az eredeti probléma megoldása, az INDEX(HOL.VAN()) összetett függvény outputja.

5. Konklúzió

Egyik célunk a Sprego népszerűsítése, annak bemutatása, hogy milyen egyszerű, az emberi gondolkodáshoz közeli logika szükséges a módszer elsajátításához. Mindezt egy olyan szemléltetőeszköz segítségével szeretnénk elérni, amely a tanulási folyamat megértési fázisában is segítségül hívható. Jelenleg 2 probléma érhető el, az elérhető problémák, feladatok listáját folyamatosan bővíteni szeretnénk. A legnagyobb feladat annak kigondolása, hogyan tudjuk minél egyszerűbben láthatóvá tenni a képletek, függvények mögött álló algoritmust, úgy, hogy a teljesen kezdő felhasználók is megértsék.

Nagyon sok ötletünk van a továbbhaladás szempontjából. Terveink között szerepel:

- a problémák számának bővítése
- a program minimális szöveget tartalmaz – képletek nevei –, így könnyen alakítható többnyelvűvé
- beszélő menü
- diákokkal tesztelni a program hatékonyságát, kontrolcsoport mellett

Az informatikaoktatás nézőpont váltás előtt áll, szeretnénk mi is hozzájárulni a megértésen alapuló, problémamegoldás központú szemlélet terjesztéséhez. A Sprego hatékonysága már bizonyított, de a kezdeti lépések nehezek. A felületalapú, kényelmes megoldásokat kínáló irodai szoftverek világában nem egyszerű előidézni a változást, nehéz rávenni arra a diákokat, hogy

elkezdjenek gondolkodni. A táblázatkezelő programok képleteinek létrehozása, azok egymásba ágyazásának megértése lassú folyamat a szöveges utasítások miatt. Az oktató-demo szoftverünkkel ezt a folyamatot szeretnénk megkönnyíteni és felgyorsítani.

Tervekben, ötletekben nem szűkölködünk, ennek ellenére nagyon fontosnak tartjuk, hogy még most, ebben a kezdeti stádiumban minél több visszajelzést kapjunk az elképzeléseinkkel kapcsolatban. További ötletekért, kritikákért, módszertani észrevételekért nagyon hálásak vagyunk.

A szoftver publikálásra kerül a még készülőben lévő hivatalos Sprego weboldalon (http://sprego.hu), ahol az oldalba ágyazva használható lesz a webes verzió, illetve letöltő linkeken keresztül beszerezhető majd Windows, Linux és MAC operációs rendszerekre asztali használatra. Az Androidra készített verzió terjesztése a Google Play áruházon keresztül történik majd. Tervben van továbbá a mobil platformok támogatottságának kiszélesítése, elsősorban a Windows Phone 10-re fókuszálva, azonban a szoftver módszertani fejlesztése fontosabb helyet foglal el a prioritásaink között.

Irodalomjegyzék

1. Kecskés István: *Mikroszámítógépek használata az idegennyelv-oktatásában*. Tankönyvkiadó, Budapest (1987)
2. Booth, S.: *Learning to program: A phenomenographic perspective. Gothenburg*, Sweden: Acta Universitatis Gothoburgensis (1992)
3. NAT: *Nemzeti Alaptanterv 2007*. Korona Kiadó, Budapest (1995)
4. Wing, J. M.: *Computational Thinking*. March 2006/Vol. 49, No. 3 Communications of the ACM (2006)
5. Panko, R. R.: *What We Know About Spreadsheet Errors*. Journal of End User Computing's. Special issue on Scaling Up End User Development. (10)2 (2008) 15–21
6. Panko, R., Aurigemma, S.: *Revising the Panko-Halverson taxonomy of spreadsheet errors*. Decis. Support Syst. 49, 2 (2010) 235–244
7. Sestoft, P.: *Spreadsheet technology*. Version 0.12 of 2012-01-31. IT University (2011)
8. OECD: PISA 2009 Results: Students on Line: Digital Technologies and Performance (Volume VI). (2011) http://www.ecdl.org/media/PISA_2009_Results.pdf.
9. Csernoch, M., Balogh, L.: *Algoritmusok és táblázatkezelés. Tehetséggondozás a köz-oktatásban az informatika terén*. Magyar Tehetségsegítő Szervezetek Szövetsége, Budapest (2011)
10. Report of JPMorgan Chase & Co. Management Task Force. Regarding 2012. CIO Losses. http://files.shareholder.com/downloads/ONE/2272984969x0x628656/4cb574a0-0bf5-4728-9582-625e4519b5ab/Task_Force_Report.pdf
11. Kerettanterv. 51/2012. (XII. 21.) számú EMMI rendelet – a kerettantervek kiadásának és jóváhagyásának rendjéről (2012) http://kerettanterv.ofi.hu/
12. Nemzet Alaptanterv 2012 (2012) http://dokumentumtar.ofi.hu/index_NAT_informatika.html.
13. Biró, P., Csernoch, M.: *Deep and surface structural metacognitive abilities of the first year students of Informatics*. 4th IEEE International Conference on Cognitive Infocommunications, Proceedings, Budapest (2013) 521–526
14. Biró, P., Csernoch, M.: *Elsőéves informatikushallgatók algoritmizáló készségei. XXIII. Nemzetközi Számítástechnika és Oktatás Konferencia* - SzámOkt 2013, EMT (2013) 154–159
15. Csernoch, M., Biró, P.: *Teachers' Assessment and Students' Self-Assessment on The Students' Spreadsheet Knowledge*. EDULEARN13 Proceedings July 1st-3rd, 2013 — Barcelona, Spain. Publisher: IATED (2013) 949–956
16. IEEE&ACM Report 2013. Computer Science Curricula 2013. *The Joint Task Force on Computing Curricula Association for Computing Machinery* (ACM) IEEE Computer Society http://www.acm.org/education/CS2013-final-report.pdf.
17. Biró, P. Csernoch, M., Abari, K., Máth J.: *First year students' algorithmic skills in tertiary Computer Science education*, 9th International Conference on Knowledge, Information and Creativity Support Systems, 6–8. November, 2014, Cyprus, Limassol, Ed.: George Angelos Papadopoulos, Cyprus Library (2014) 301–306

18. Biró, P., Csernoch, M.: *Deep and surface metacognitive processes in non-traditional programming tasks*. In: 5th IEEE International Conference on Cognitive Infocommunications CogInfoCom 2014 Proceedings. IEEE Catalog Number: CFP1426R-USB, Italy (2014) 49–54

19. Biró, P., Csernoch, M.: *Táblázatkezelés algoritmikus megközelítése*. Kiss Árpád Emlékkonferencia Tanulmánykötete 2013, Debrecen (2014)

20. Csernoch, M., Biró, P.: *Spreadsheet misconceptions, spreadsheet errors*. Oktatáskutatás határon innen and túl. HERA Évkönyvek I., ed. Juhász Erika, Kozma Tamás, Publisher: Belve-dere Meridionale, Szeged, 2014 (2014) 370–395

21. Csernoch, M., Simon, K., Brósch, É., and Kiss, É. (2014) *I HaveLearned Spreadsheet Management With Sprego. In Hungarian, Spregoval tanultam táblázatkezelést*. In: Zsakó Lász-ló (szerk.) INFO Éra 2014. Zamárdi, Magyarország, 2014.11.20-2014.11.22. Budapest: NJSZT, pp. 1–20. ISBN 978-963-12-0627-2.

22. Csernoch, M.: *Programozás táblázatkezelő függvényekkel*. Sprego. Műszaki Könyvki-adó, Budapest (2014)

23. Biró P., Csernoch M., Máth J., Abari K.: *Measuring the level of algorithmic skills at the end of secondary education in Hungary*. Procedia - Social And Behavioral Sciences 176 (2015) 876–883

24. Csernoch, M., Biró, P.: *Számítógépes problémamegoldás*, TMT, Tudományos és Mű-szaki Tájékoztatás, Könyvtár- és információtudományi szakfolyóirat, Vol. 62(3) (2015) 86–94

25. Csernoch, M., Biró, P.: *Sprego programming*. Spreadsheets in Education (eJSiE) Vol. 8, Iss. 1. http://epublications.bond.edu.au/cgi/viewcontent.cgi?article=1175&context=ejsie.

26. Hutong Games: *PlayMaker - Visual Scripting for Unity3D* (2015) http://www.hutonggames.com/

27. *Intel® XDK | Intel® Developer Zone (2015)* https://software.intel.com/en-us/intel-xdk

28. Lifelong Kindergarten Group: *Scratch - Imagine, Program, Share* (2015) https://scratch.mit.edu/

29. nwjs (2015) NW.js http://nwjs.io

30. Scirra: *Construct 2 - Create Your Own Games* (2015) https://www.scirra.com/store/construct-2

31. Scirra: *Create Games with Construct 2* (2015) https://www.scirra.com/11.01.

32. Scirra: *Index page - Scirra Forums* (2015) https://www.scirra.com/forum/

33. Scirra: *Official Construct 2 Manual* (2015) https://www.scirra.com/manual/1/construct-2

34. Scirra: *Top game making tutorials* (2015) https://www.scirra.com/tutorials/top

35. Stencyl: *Stencyl: Make iPhone, iPad, Android & Flash Games without code* (2015)

36. Unreal: *Unreal Engine Technology* (2015) http://unrealengine.com

37. YoYo Games: *Welcome to YoYo Games* (2015)
http://www.yoyogames.com/