Algoritmus vizualizáció a tanítási gyakorlatban

Törley Gábor

pezsgo@inf.elte.hu
ELTE IK

Absztrakt. Ez a cikk bemutatja az algoritmus vizualizáció (AV) történetét, külön kiemelve az oktatásmódszertani kérdéseket. Egy alapozó programozás kurzus hallgatóival elvégzett kétcsoportos pedagógiai vizsgálat is közlésre kerül, amely az AV eredményességét és az absztrakt gondolkodásra gyakorolt hatását mérte. A vizsgálat eredményei szerint eredményesebbek voltak az AV-t használó hallgatók.

Kulcsszavak: programozás, oktatás, algoritmus vizualizáció, multimédia, algoritmus, algoritmikus gondolkodás, absztrakció

1 Bevezetés

A magyar közoktatás céljai között egyre nagyobb hangsúlyt kap a tanulók kognitív képességének fejlesztése, értelmének kiművelése. Ahhoz, hogy bárki elboldoguljon az „Élet útvesztőjében", szüksége van tudatos gondolkodásra, hogy a napi problémákra gyors és hatékony megoldást találjon. Az algoritmikus gondolkodás elsajátítása ebben is nyújt segítséget. [23]

Szántó szerint [23] az algoritmikus gondolkodásnak a legfontosabb céljai a következők:

- Tudatos, tervező magatartás kialakítása
- Önkontroll kialakítása
- Értékelés – tudatosítás

A tervezés folyamán a tanuló sorrendbe fűzi a konkretizált gondolatokat (*algoritmus*), majd osztályozza ezeket a gondolatokat, mérlegeli saját stratégiáját, végül értékeli megoldását, hogy teljességben lássa a megoldandó problémát, és akár rossz irányú próbálkozásokon keresztül eljusson a megoldásig. Felidézi tapasztalatait, elraktározza, majd felhasználja azokat a következő tervezésnél. Az algoritmusok tudatos alkotása tehát fejleszti a tanulók kognitív képességeit.

A programozásoktatás a programozási tételek tanításával szerepet tud vállalni a tanulók kognitív képességeinek fejlesztésében, mégis, a tapasztalatok szerint – külföldön is [3, 28] –, nehéz tanulni és tanítani ezeket. A tanítási módszerek fejlesztése ezen a területen fontos feladata a mai informatikaoktatásnak. [25, 26]

Tanári tapasztalataim szerint a legnehezebb része a programozási tételek tanulásának, mikor arra a kérdésre keressük a választ, hogy miért is lesz jó az adott algoritmus, miért fogja az adott tétel „megoldani" a problémát. Tehát az adott algoritmus helyességének „bizonyítása" – ezen a szinten természetesen nem matematikai eszközökkel – nehézségekbe ütközik, mert ehhez a tanulónak át kell látni az algoritmus lépéseit, és a lépések közötti összefüggéseket. További probléma (sőt probléma-kettős) az, hogy ha már megértette a tanuló a tétel algoritmusát, akkor abból hogyan is „keletkezik" a konkrét feladathoz tartozó konkrét algoritmus, majd azon is továbblépve a helyesen működő kód. Azaz egyetlen problémába sűrítve mindezt: az *absztrakt* tételt hogyan fogja az adott feladatra alkalmazni, *konkretizálni*.

Tanítási gyakorlatom alatt programozási tételeket tanítottam egy fővárosi szakközépiskolában. Akkor egy prezentációba ágyazott animáción keresztül mutattam meg a diákoknak az

adott tétel működését, tehát a diákok látták az animációt, olvasták az algoritmus szövegét, és hallották a magyarázatot. Sokkal többet tudtam így átadni, mint ha csak az algoritmus szövegét tanulmányoztuk volna végig, mert több csatornán és dimenzióban tudtam „közvetíteni" a tananyagot. A *vizuális* és *audiális* csatornán túl az *idő* dimenzióját is felhasználtam, ugyanis az algoritmus időbeli változását is figyelemmel kísérhették a hallgatók. Ezt a hatást hívja Mayer „Multimédia hatásnak" a Multimédia tanulás kognitív elméletében (*Cognitive Theory of Multimedia Learning*), amely öt alapelvet fogalmaz meg [13]:

- Többszörös ábrázolás elve: jobb a magyarázatot megjeleníteni szavakban és képekben, mint csak szavakban.
- Egyidejűség elve: Magyarázat közben a megfelelő képet és szöveget együtt („egy időben") jelenítsük meg, ne külön-külön.
- Megosztott figyelem elve: A képi magyarázat mellé élőszóban adjuk a szóbelit, ne írásban.
- Egyedi különbségek elve: A fenti alapelvek sokkal fontosabbak alacsony tudásszintű tanulóknak, mint magasabb tudásszintűeknek.
- Koherencia alapelve: Multimédiás magyarázatnál használjunk minél kevesebb, a tárgyhoz nem tartozó fogalmakat vagy képeket.

A fenti elmélet – amely nagy hangsúlyt fektet a vizuális, képi elemekre –, oktatási környezetben pozitív eredményeket hozott. [12] Tapasztalataimból azt a következtetést vontam le, hogy ha olyan eszközöket használok a tanítás alatt, amelyek segítenek *elképzelni* az algoritmus belsejében történteket, akkor ezek meggyorsítják a megértés folyamatát. Az algoritmus-vizualizációs (AV) eszközök használata ebben nyújthat segítséget.

Az AV egyik erőssége lehet több érzékszerv bevonása a tanulásba. Kátai és munkatársai tanulmányai [8, 10] megerősítik azt a feltevést, hogy az olyan tanítási módszer, amely különböző érzékszervekre hat, hatékonyan támogatja algoritmusok tanítását és tanulását. Sőt, AV eszközöket nem csak informatikus hallgatók algoritmikus gondolkodása fejlesztésében lehet hasznosítani. [9]

2 Az algoritmus vizualizáció története

Az algoritmus-vizualizáció (AV) a szoftver-vizualizáció alosztályaként számítógépes algoritmusok magas szintű működésének illusztrálásával foglalkozik, általában abból a célból, hogy a programozást tanulók jobban megértsék az algoritmus eljárásainak működését. [7]

Egy folytonosan változó folyamatot nagyon nehéz bemutatni statikus eszközökkel, mint például szövegekkel és képekkel egy tankönyvben. Ugyanez igaz egy „klasszikus" prezentációra is. Bár prezentációs szoftverekben van lehetőség animációs betéteket készíteni, ez azonban igen munkaigényes, sőt általában lehetetlen újra felhasználni az animációt egy másik feladathoz. A tanítási gyakorlatban a táblai magyarázatnál egy folyamatot próbál illusztrálni a tanár: az algoritmus legjellegzetesebb adatainak állapotváltozásait dokumentálva hajtja végre az algoritmust. A legtöbb tanulónak még így is nehéz minden részletében megérteni az algoritmust. Emiatt irányult a külföldi oktatók figyelme a vizualizációra, amellyel könnyebben, problémamentesebben lehet átadni az algoritmus dinamizmusát. [5]

Az AV a múlt század 70-es éveinek végén a batch-oritentált szoftverekből – amelyek lehetővé tették az oktatóknak animációs filmek készítését [1] – mára magas szintű interakcióval rendelkező rendszerekké fejlődött, amelyekkel a tanulók felfedezhetik, beállíthatják, dinamikusan megváltoztathatják az algoritmus animációját a saját igényeiknek, kíváncsiságuknak megfelelően [4, 21], vagy maguk képesek megalkotni saját vizualizációjukat [6, 22]. Ezt a történelmi folyamatot tekintem át röviden.

Az 1990-es évek közepén, az Internet és a Java programozási nyelv alkalmazása egyre nagyobb mértékben elterjedt. Az 1990-es évek végén és a 2000-es évek elején új generációja született meg az AV rendszereknek. Példák ezekre a rendszerekre: JSamba [I9], JAWAA [18], JHAVÉ [16], ANIMAL [19], és TRAKLA2. [I12] Egy közös tulajdonsága volt ezeknek a rendszereknek: arra szánták őket, hogy teljes eszközkészletet adjanak az AV-k építéséhez, tehát követték a Java előtti rendszerek tradícióit. Fentebb említettem, hogy a kezdeti korszak rendszereit nehéz volt implementálni, a „Java-korszakban" legalább megvalósíthatóvá tették a fejlesztési folyamatot azok számára, akik készek voltak több-kevesebb munkát fektetni egy látványos rendszer kifejlesztésébe. A JAWAA üdítő példát jelent az ilyen szoftverek között, ugyanis az oktatónak könnyű volt vizualizációt készítenie, viszont ez a rendszer nem támogatta az interaktivitást.

A fő változás, amely egybeesett a Java alkalmazásával, az olyan AV rendszerek megjelenése volt, amelyek függetlenek voltak az AV fejlesztői rendszerétől és az operációs rendszertől. Ugyanis külön rendszerben fejlesztették a vizualizációt, és külön rendszerben játszották azt le. A fejlesztőnek megváltozott a szerepe. Ennek hatására olyan projektek láttak napvilágot, ahol az AV-k egy különálló csomagot képviseltek, anélkül, hogy külön fejlesztői rendszert szolgáltattak volna melléjük. Erre példák: Data Structure Navigator (DSN) [I3], Interactive Data Structure Visualization (IDSV) [I6], Algorithms in Action [I10], Data Structure Visualization [I4] és TRAKLA2. [I7]

A Java használatának másik hatása az volt, hogy oktatók és hallgatók is elkezdtek fejleszteni AV rendszereket. Sok év kitartó munkájának gyümölcseként születtek olyan Java kisalkalmazások, amelyek bemutattak egy-egy témakört, pl. különböző keresések fákban, térbeli indexelés, stb.. Ilyenek voltak például: Examples include Binary Treesome [I5], JFLAP [I8], Marylandi Egyetem Spatial Index Demos [I1] és a Virginiai Egyetem Algoritmus-vizualizációs Kutatócsoportja (2011) által készített rendszer. [I11]

Naps és munkatársai tanulmányukban [17] kiterjesztették Hundhausen és munkatársainak következtetéseit [7] arról, hogy a tanulónak minél aktívabban kell részt vennie a vizualizációban. Definiáltak egy taxonómiát, amely meghatározza a tanuló aktivitásának szintjét és a típusát. Hat szintet állapítottak meg:

1. *Nincs megtekintés*: ezekben az esetekben nem használják az AV-t egyáltalán.
2. *Megtekintés*: ilyenkor a tanuló csak figyeli az AV végrehajtását és lépéseit.
3. *Válasz*: Ilyenkor a tanuló válaszol kérdésekre, miközben megtekinti a vizualizációt.
4. *Változtatás*: ilyenkor a tanuló a vizualizáció közben tudja megváltoztatni az adatokat.
5. *Építés*: ilyenkor a tanuló megépíti maga az algoritmus vizualizációját.
6. *Prezentálás*: a tanuló, az AV-t használva elmagyarázza az algoritmus működését és visszajelzést kér.

Ez a taxonómia szorosan kötődik a Bloom hierarchiához. [3]

Urquiza–Fuentes és Velázquez–Iturbide [27] egy új alapkutatást végzett az AV rendszerek pedagógiai értékéről. A tanulmány azokra a területekre fókuszált, ahol az oktatásban hasznot hoztak az AV rendszerek. A kutatás megerősítette Hundhausen és munkatársai 2002-es eredményeit (és ezzel némileg ellentmondott a fenti taxonómiának), mi szerint a 2. szint sem hoz hasznot a tanulói eredményekben. Sajnos, sok AV rendszer csak a 2. szinten tart ma. [21]

A fenti alapkutatás arra az eredményre is jutott, hogy minél magasabb szinten képes alkalmazni a tanuló az AV rendszert, annál több tanulási hasznot hoz az AV.

Myller, Bednarick, Sutina, és Ben–Ari végzett kutatást [15] az aktivitás hatásairól, kollaboratív tanulási környezetben. Ők kiterjesztették a fenti taxonómiát, hogy jobban behatárolják a tanulók viselkedését. Négy újabb szintet adtak hozzá a taxonómiához, ebből kettőt említek meg: *irányított megtekintés* és *adatok bevitele*. Ez a két új szint a 2. és a 3. közé került be. Az *irányí-*

tott megtekintés folyamán a tanuló irányítja a vizualizációt. Pl. meg tudja állítani, vissza tud lépni az előző lépésre vagy gyorsítani és lassítani tudja azt. Az *adatok bevitele* szintnél a tanuló képes bemeneti adatokat megadni a vizualizáció lefutása előtt vagy közben is, ez azokon a pontokon lehetséges csak, ahol maga a program adatot kér be.

Myller és munkatársai kísérlettel igazolták [15] elméletüket: párokban dolgoztak hallgatókkal, akik a BlueJ [20] és a Jeliot 2000 [2] rendszereket használták. Megfigyelték és rögzítették a tanulók kommunikációját a kísérlet közben. Arra a következtetésre jutottak, hogy a kibővített taxonómia pozitívan korrelál az interakció mértékével. Az interakció az egyik alapeleme a sikeres számítógéppel segített csoportmunkának. [14]

3 A kísérlet bemutatása, eredményeinek, tapasztalatainak kiértékelése, elemzése

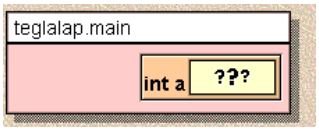
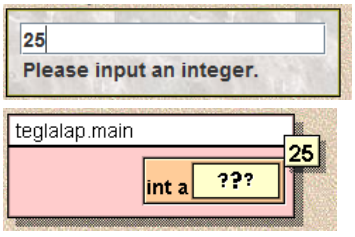
A 2012/2013-as tanév őszi félévében egy kétcsoportos pedagógiai vizsgálatot folytattam az ELTE Informatikai Karán, ahol a Programozási alapismeretek kurzusban résztvevő programtervező informatikus hallgatók egész féléves munkáját vizsgáltam. Az évfolyam hallgatói közül 5-5 csoport vett részt a kísérleti és a kontroll csoportban, összesen 153 fővel.

3.1 Programozási alapismeretek kurzus tematikájának bemutatása

A kurzus alapozó tantárgy első évfolyamon, a hallgatók fele ekkor találkozik –a felméréseim szerint– először programozással, ezért a tematika is az alapoktól indul. A kurzus célja, hogy a hallgatókat egy szintre hozza, illetve az elvárt középiskolai programozási tudás rendszerezése, és formalizálása,. A hallgatók a Code::Blocks [I2] fejlesztői környezetet használják, és C++ nyelven kódolják az órai feladatokat.

3.1.1 Programozási környezet, I/O műveletek, deklaráció

Az első témakör feldolgozása során a programozási környezet megismerése és a beolvasás és kiíratás parancsai mellett a hallgatók megismerkednek a deklaráció fogalmával. A deklarációnál azt az alapelvet kell megérteni, hogy akkor és csak akkor tudok valamilyen adatot (változót) felhasználni (azaz értéket adni neki, számolni vele), ha azt először deklaráltam. A Jeliot rendszer segítségével ez a sorrendiség vizuálisan is megmutatható. A „/” jel után a Java parancsot írtam, ahogy a Jeliotban megjelenik.

Programkód	Vizuális reprezentáció
`int a;`	
`cin >> a; / a = Input.readInt();`	

Ebből az animációból könnyen megérti a hallgató, hogy a változónak mindig legelőször létre kell jönnie, majd értéket kell adni neki, mielőtt értékét használhatnánk.

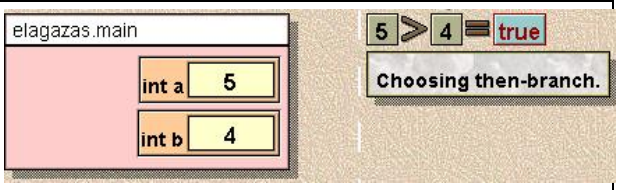
3.1.2 Vezérlési szerkezetek

A vezérlési szerkezetek működésének bevezetésékor animációval támogatva láthatóvá válik az, hogy milyen folyamat megy végbe, amikor az adott vezérlési szerkezet lefut, illetve, hogy miért úgy fut le az adott szerkezet.

A második témakör: az elágazás. Azért ez az első vezérlési szerkezet a sorrendben, mert ennek a legegyszerűbb a működési elve, másrészt a gyakorlati előnyeit – pl. előfeltétel ellenőrzését – hamar ki lehet használni.

Akkor értette meg az elágazás működését a hallgató, ha tudja, hogy melyik ága fog lefutni (természetesen a konkrét értékek ismeretében).

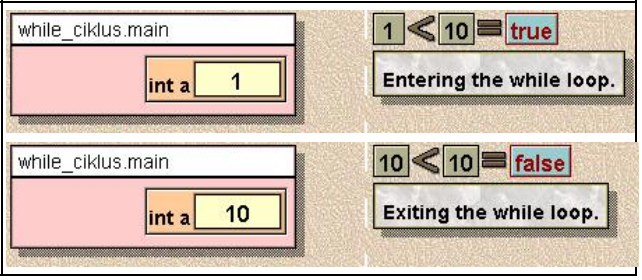
Programkód ♣	Vizuális reprezentáció

`if (a>b)`
`  {`
`   cout << "a nagyobb,↵`
`  mint b";`
` / Output.println("a nagyobb,↵`
`  mint b");`
`  }`
`else`
`  {`
`   cout << "a nem nagyobb, ↵`
`  mint b";`
` / Output.println("a nem↵`
`  nagyobb, mint b");`
`  }`
	

A Jeliot kiírja a feltétel logikai vizsgálatának az eredményét, majd a választ is, hogy a logikai vizsgálat eredménye miatt melyik ágon folytatódik a program futása. A hallgató számára mindig egyértelmű, hogy miért azon az ágon halad tovább a program futása.

A harmadik témakör: a ciklus. A ciklus fogalmának és működésének megértése nehezebb, mint az elágazásé, mert itt már egy időben „elhúzódó” folyamatról van szó. A ciklus működését akkor értette meg a hallgató, ha 1) a ciklus algoritmusát értelmezve meg tudja határozni, hogy melyik az az eset, amikor a program nem fogja már újra végrehajtani a ciklusmagot, illetve 2) ha képes egy saját maga által leírt működési elvű ciklus algoritmusát és kódját megírni.

Programkód	Vizuális reprezentáció

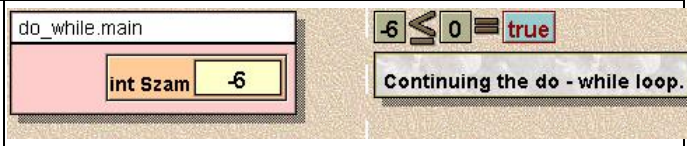
`while (a<10)`
`{`
`   ++a;`
`}`
	

Az AV program szöveges magyarázatából a hallgató megkapja a választ arra a kérdésre, hogy miért lép be a ciklusba vagy éppen ki a ciklusból a program.

♣ A csupán szerkesztési okok miatti sortöréseket a ↵ jel jelzi. Az így több sorra tört utasítások valójában egy-egy sorba kerülnek.

A hátul tesztelős ciklusnál a Jeliot a ciklus folytatását írja ki a ciklus feltétel igaznak bizonyulásakor, így markánsan meg lehet különbözetni a működését és a szerepét az elöl tesztelős ciklusétól.

Programkód	Vizuális reprezentáció

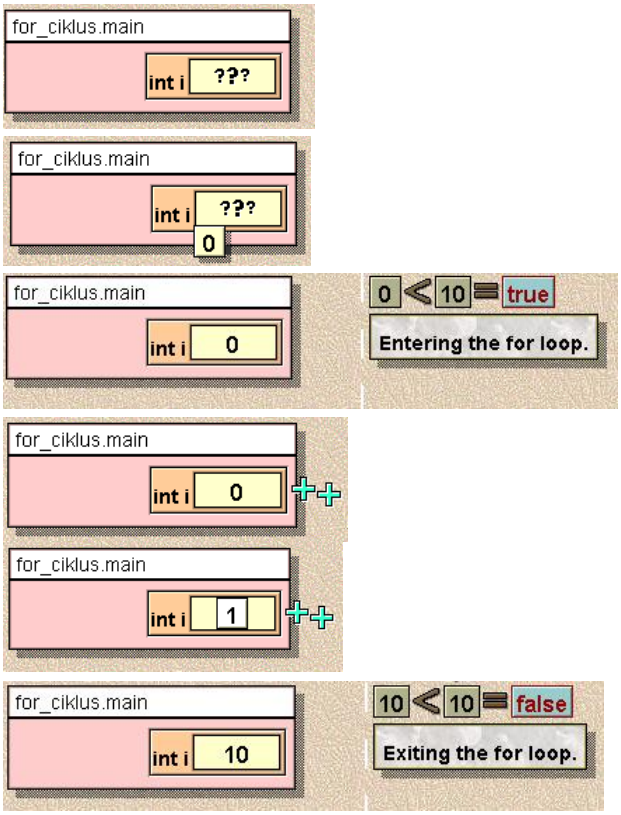
do
{
cout << "Kérem a↵
pozitív számot!")
/ Output.println("↵
Kérem a pozitív↵
számot!");
cin >> Szam;
/ Szam = Input.readInt();
if (Szam<=0)
{
cout << ("Hibas szam,↵
irj be egy pozitiv↵
szamot!";
/ Output.println("Hibas↵
szam, irj be egy ↵
pozitiv szamot!");
}
}while (Szam<=0);
	

A számlálós ciklus működésének megértése már egyszerűbb, az elől tesztelős ciklus megértését követően. A Jeliot ekkor a számlálós ciklus kódját, az abban levő kifejezések szerepének megértését tudja segíteni.

Az alábbi animációban jól követhetők a lépések: az i változó létrejön, majd értéket kap, aztán a ciklus belépési feltételének ellenőrzése és kiértékelése, lefut a ciklusmag és végül a ciklusváltozó eggyel megnő.

A vezérlési szerkezetek használata és megértése elengedhetetlen a programozási tételek magabiztos használatához.

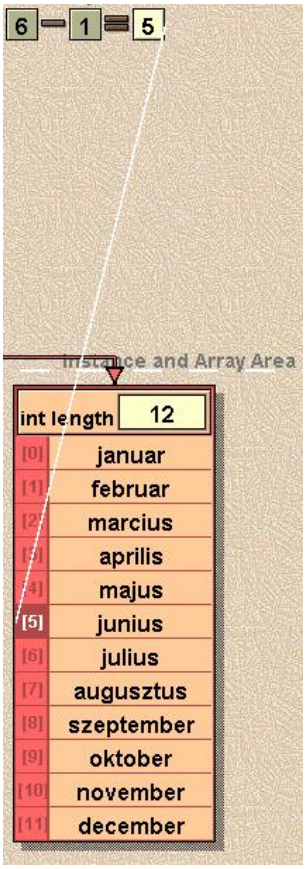
Programkód	Vizuális reprezentáció

`for (int i=0; i<10; ++i)`
`{`
`  cout << i;`
`  / Output.println(i);`
`}`
	

3.1.3 Tömbök használata

A következő témakör a tömbökkel foglalkozik. A tömb definícióját és célját még nem nehéz megérteni, viszont nehézséget jelenthet, hogy az első elemet 0-val indexeljük, a másodikat 1-gyel és így tovább, ugyanis ilyen módon egy nem C-alapú nyelven tapasztalatot szerzett hallgató könnyen zavarba jöhet az indexek miatt.

A Jeliot itt ismét a megjelenítésben, illetve a hibakeresésben tud segítséget nyújtani. A megjelenítés abban segít, hogy láthatóvá válik a tömb aktuális állapota, így a programozni tanuló nyomon tudja követni a változásokat, amely segíthet később a hibák felderítésében. Az indexelés műveletének működését is be lehet mutatni az AV programmal.



1. ábra. Jeliot 3: Tömb aktuális állapotának vizsgálata és az indexelés művelete

3.1.4 Programozási tételek

A félév utolsó nagy témaköre a programozási tételeket dolgozza fel. Tekintsük bemutató példaként a lineáris keresés tételét, amely specifikációja, az ELTE informatika tanári szakán alkalmazott konvenciói szerint [24, 25]:

Bemenet: $N \in \mathbf{N}$, $X \in \mathbf{H}^*$, $T: H \rightarrow \mathbf{L}$ [$\mathbf{L}$={igaz, hamis} – Logikai értékek halmaza]

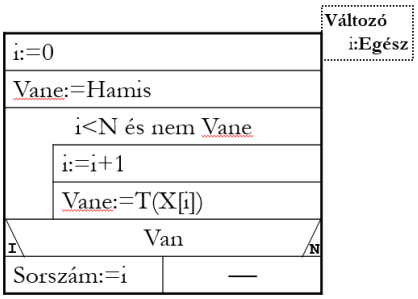
Kimenet: $Vane \in \mathbf{L}$, $Sorszám \in \mathbf{N}$

Előfeltétel: $Hossz(X)=\mathbf{N}$

Utófeltétel: $Vane \equiv \exists i \in [1..N]: T(X_i) \wedge Vane \Rightarrow Sorszám \in [1..N] \wedge T(X_{Sorszám})$

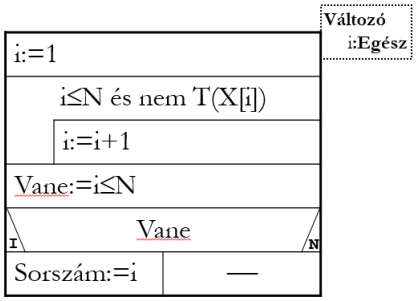
Ez a fajta konvenció jól épít a középiskolai matematika tudásra, és tovább is fejleszti azt, az előfeltétel mindenkori meggondolás és dokumentálás „kényszerével”. (Az előfeltétel fogalma valójában nem újdonság, hiszen a függvények értelmezési tartományának vizsgálata, amely ugyancsak része a középiskolai tananyagnak, ugyanerről szól. Tehát a tanuló által a matematikában begyakorolt gondolkodást kell a programozás e fázisában újraéleszteni.)

A fent specifikált lineáris keresés tétele –az alábbi algoritmus „filozófiája” szerint– egy tömb adatszerkezetben keresi a T tulajdonságú elem helyét, már ha van ilyen.



2. ábra. Lineáris keresés megoldása struktogrammal

Tekintsünk egy másik, a fenti feladatot megoldó algoritmust:



3. ábra. Lineáris keresés egy másik megvalósítása

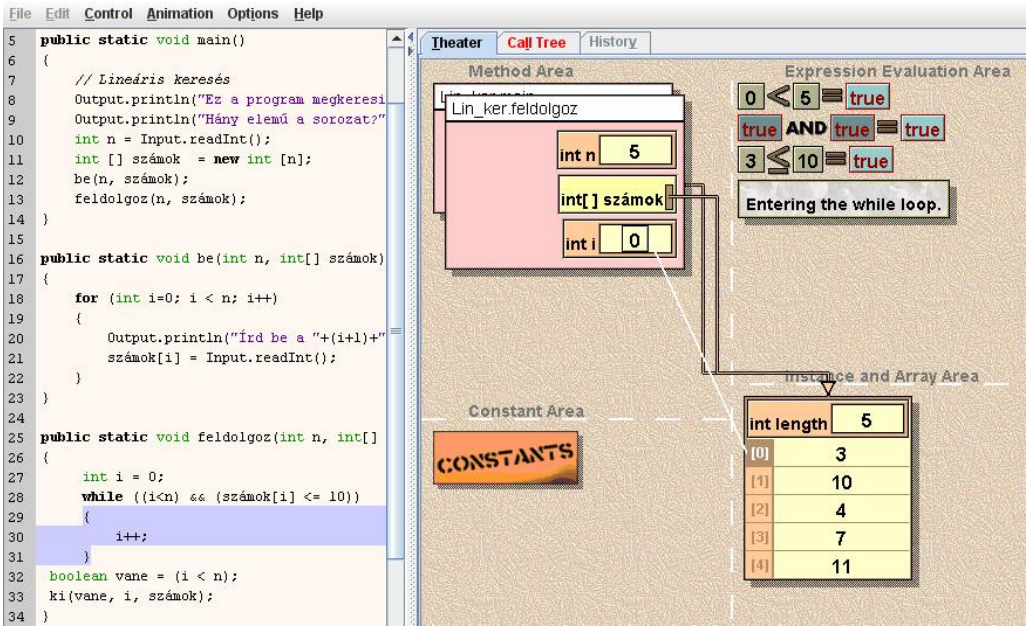
Miért érdemes egy adott programozási tételhez több absztrakt algoritmus változatot készíteni? Például azért, mert így gyakorolni lehet az algoritmus értelmezését, a két algoritmus párhuzamos vizsgálata segíti az absztrakt gondolkodás fejlődését, valamint a helyességbizonyítás nem formális elvégzésére készteti a hallgatót.

Akkor értette meg a hallgató a fenti algoritmust, ha helyesen tud felelni arra a kérdésre, hogy milyen feltétel teljesülésekor fog az kilépni a ciklusból, és miért fogja az I≤N állítás kiértékelése megmondani azt, hogy megtaláltuk-e a keresett tulajdonságú elemet.

A Jeliot 3 kétféle módon segíti a helyes válasz megadását a feltett kérdésekre, ahogy a 4. ábra mutatja.

Az ábrán a következő feladat megoldása látható, ahol a lineáris keresés tételt kellett felhasználni: „Adott egy N elemű számsorozat. Keresd meg és írd ki az első 10-nél nagyobb számot. Ha nincs ilyen szám, arról is tájékoztasd a felhasználót!”

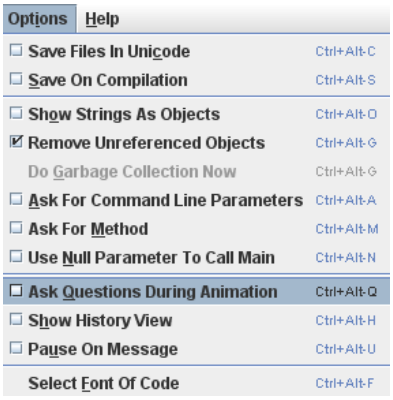
Korábban láttuk, Jeliot 3 minden elágazásnál és ciklusnál kiértékeli a feltételt, és ilyen módon megindokolja, miért arrafelé halad a végrehajtás, amerre halad. A fenti lépést felhasználva a tanár rá tud mutatni, mi a szerepük ciklusfeltételeknek, illetve tovább tudja vinni a gondolatot afelé, hogy mik azok az esetek, amikor kilépünk a ciklusból és miért, illetve mit is jelent majd a feladat megoldása szempontjából az, hogy melyik feltétel hamissá válásakor lépünk ki a ciklusból.



4. ábra. Ciklus bennmaradási feltételének kiértékelése

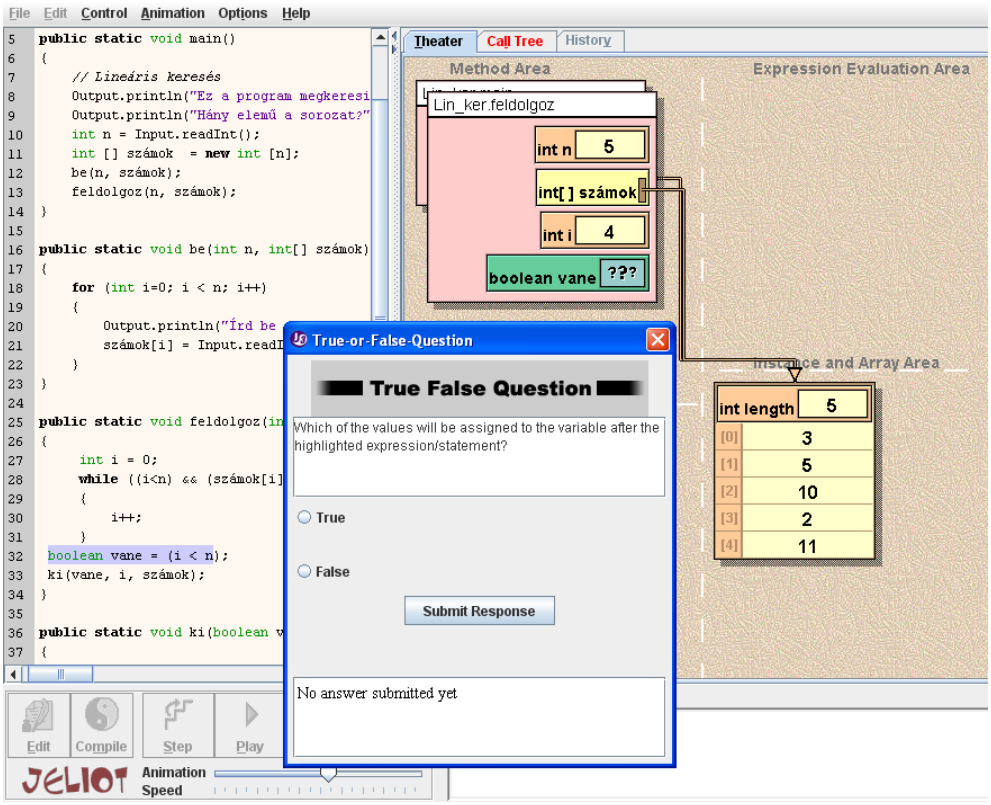
A program belső állapota mindent elmond: éppen melyik tömbelemet vizsgáljuk, mik a helyi változók értékei, illetve hogy állunk a ciklus belépési feltételének kiértékelésével. A C++ nyelvnél ez a különbözőség csak kismértékű, gyakorlatilag csak az input-output műveleteknél van különbség, az adatszerkezetek és a lényegi algoritmus kódolása szinte betűről betűre ugyanaz (a függelékben található a C++-Jeliot „szótár").

A Jeliot 3-at be tudjuk úgy állítani, hogy kérdéseket tegyen fel a változók értékeit illetően. Ennek beállítását az alábbi ábra mutatja.



5. ábra. Ide pipát téve kérdéseket kapunk animáció közben

A kérdések „egysíkúak", sajnos: mindig csak a változók/kifejezések értékére, értékváltozására vonatkoznak. Azonban, programkódunkat tekintve, az egyik feladat az algoritmus megértésének kulcskérdését feszegeti (6. ábra), végül is kijelenthetjük kitűnő a környezet otthoni, egyéni tanulásra, illetve hibakeresésre is.



6. ábra. A belső állapotot leolvasva helyesen tudunk felelni a kérdésre

A többi programozási tételnél hasonló módon lehet felhasználni az AV nyújtotta segítséget.

3.2 A kutatás módszere

A kutatás első mozzanata egy előzetes tanulmányokat felmérő kérdőív kitöltetése volt a kísérleti és a kontroll csoport hallgatóival (a kérdőív a cikk Függelékében található). A kérdőív célja volt egyrészről számszerűsíteni azt, hogy ki, mennyi órában tanult informatikát, illetve programozást középiskolai tanulmányai alatt, másrészt felmérni a hallgatók algoritmikus gondolkodásának állapotát még azelőtt, hogy a tantárgy hatással lenne arra.

A kutatás folyamán az algoritmikus gondolkodásról szóló 3-5. kérdésre a hallgatók pontszámot és jegyet kaptak, ez alapján lehet mérni a hallgatók fejlődését. Az 5. kérdés a hallgató absztrakciós készségét mérte. A kérdések kiértékelésénél az azokra adott válasz helyességét és részletességét vizsgáltam. A részletesség azért fontos a helyesség mellett, mert ezáltal lehet mérni a gondolkodásmód sokrétűségét: mennyire figyel a hallgató az apróbb részletekre. Annál több pont járt a feladat részletességére, minél fontosabb előfeltételt adott meg a feladat megoldója. A három feladat megoldására maximum 14 pontot lehetett kapni.

Ahhoz, hogy a feladatok eredményéhez képest meg lehessen határozni a fejlődést, ötfokozatúvá kellett alakítani az értékelést, hasonlóan a későbbi számonkérések eredményeihez. Pontozási rendszer:

alsó pont	jegy	arány
0	1	0%
5,5	2	39%
7,5	3	54%
10	4	71%
12	5	86%
14	(max)	

A bemeneti feladat jegyének meghatározása után már van kiindulópontunk ahhoz, hogy meghatározzuk a fejlődés mértékét. A félév folyamán a hallgatók négy számonkérésen adnak számot a tudásukból:

- 1. csoportzárthelyi: két egyszerű programozási tétellel megoldható feladat kódolása, specifikáció és algoritmus alapján (a számonkérés célja: dominánsan algoritmusértés + kódolás ismeret),
- 2. csoportzárthelyi: három egyszerű programozási tétellel megoldható feladat algoritmizálása (struktogrammal), specifikáció alapján (a számonkérés célja: specifikációértés + algoritmuskészítés, ezek együtt jóval több absztrahálási képességet kívánnak, mint az előbbi),
- beadandó feladat készítése, ahol négy részfeladatot kellett megoldani, amelyek közül az egyikben előfordultak összetett programozási tétellel megoldható feladatok (a számonkérés célja: komolyabb méretű feladat igényes megoldása, „termék-gyártás”),
- Évfolyam zárthelyi dolgozat, amelynél a négy részfeladatból 2 volt olyan, amelyet összetett programozási tétellel kellett megoldani (komolyabb méretű probléma teljes megoldása).

A fejlődés meghatározásánál nem vettem figyelembe a beadandók eredményét, ugyanis ennél a számonkérésnél kevésbé garantálható, hogy a hallgató önálló munkát adott be, ilyen módon a hitelesség megkérdőjelezhető. A többi három számonkérést az időbeli sorrend alapján súlyoztam az értéküket, ugyanis minél jobban távolodunk az első gyakorlat idejétől, annál mélyebbnek kellene lennie a megértésnek. A fejlődés mértékét (*F*) a következő képlettel határoztam meg:

$$F = [(CsZH_1 - E) + 1{,}5 * (CsZH_2 - E) + 2 * (\acute{E}vfZH - E)]/18,$$

ahol $CsZH_1$ az 1. csoportzárthelyi jegye, $CsZH_2$ a 2. csoportzárthelyi jegye, *ÉvfZH* az évfolyam zárthelyi jegye, illetve *E* az előzetes tanulmányok 3-5. kérdésére kapott osztályzat. A fenti képlet eredménye egy -1 és 1 közötti valós szám lesz. Az eredmény előjele mutatja a fejlődés irányát, az értéke pedig a nagyságát.

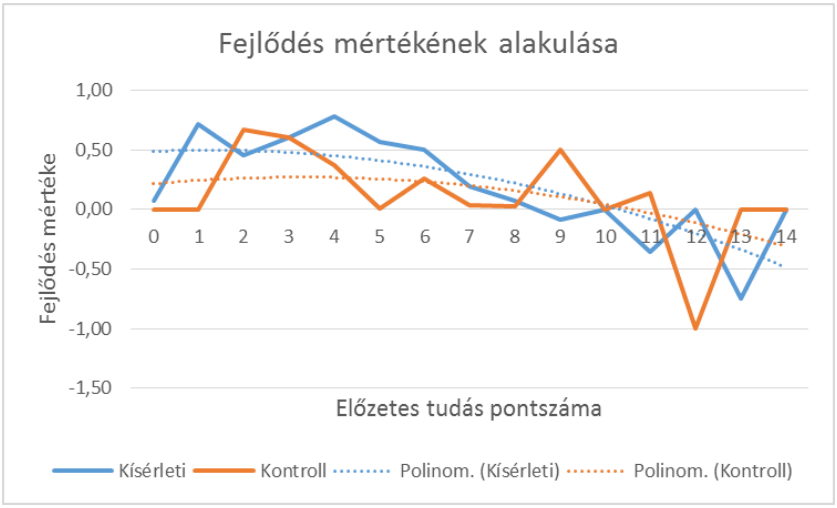
Az algoritmikus gondolkodást felmérő 3-5. kérdések eredményei hasonlóan alakultak mindkét csoportnál, nem tapasztalható jelentős eltérés, tehát kimondható, hogy a két csoport közel azonos szintről indul, tehát a későbbiekben közlendő fejlődést meghatározó értékek közötti különbség nem a bemeneti többlettudásnak köszönhető.

3.3 A vizsgálat eredményei

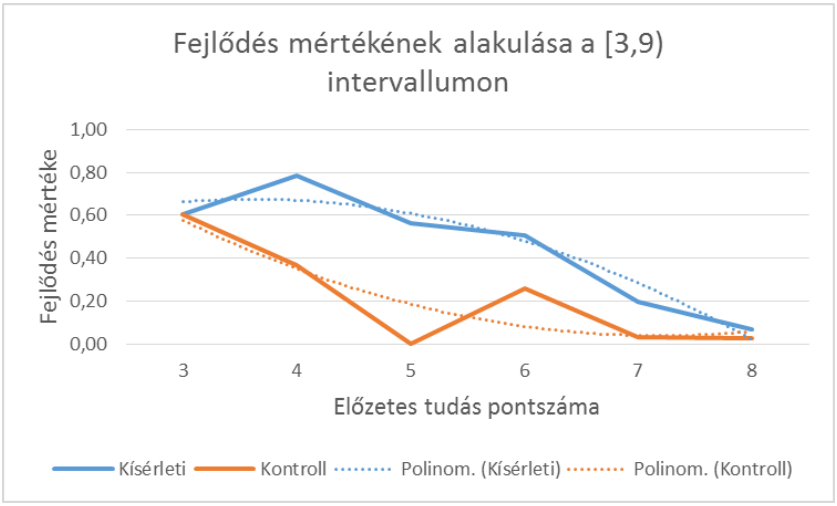
A következő táblázatból kiderül, hogy az AV rendszerrel való oktatásnak mekkora hatása volt a hallgatók fejlődésére és hogy a kísérleti vagy a kontroll csoport fejlődött-e többet.

Előzetes tanulmányok pontszám	Darabszám		%		Átlagos fejlődés	
	Kísérleti csoport	Kontroll csoport	Kísérleti csoport	Kontroll csoport	Kísérleti csoport	Kontroll csoport
0	2 db	0 db	3%	0%	0,07	-
1	3 db	0 db	4%	0%	0,71	-
2	3 db	4 db	4%	5%	0,45	0,67
3	***6 db***	***7 db***	***8%***	***9%***	***0,60***	***0,60***
4	***14 db***	***16 db***	***18%***	***21%***	***0,79***	***0,37***
5	***14 db***	***7 db***	***18%***	***9%***	***0,56***	***0,00***
6	***12 db***	***23 db***	***15%***	***31%***	***0,51***	***0,26***
7	***6 db***	***4 db***	***8%***	***5%***	***0,20***	***0,03***
8	***9 db***	***10 db***	***12%***	***13%***	***0,07***	***0,03***
9	5 db	1 db	6%	1%	-0,09	0,50
10	0 db	0 db	0%	0%	-	-
11	3 db	2 db	4%	3%	-0,36	0,14
12	0 db	1 db	0%	1%	-	-1,00
13	1 db	0 db	1%	0%	-0,75	-
14	0 db	0 db	0%	0%	-	-

A hallgatók túlnyomó része (kiemelve a többi adat közül), 84%-a a [3,9) intervallumban szerzett pontot az előzetes tudást felmérő feladatokon. Az eredmények alapján a kísérleti csoport 78%-a, míg a kontroll csoport 89%-a esik bele ebbe az intervallumba. A fejlődés mértékét az alábbi két diagram mutatja, a második a kiemelt [3,9) intervallumét:



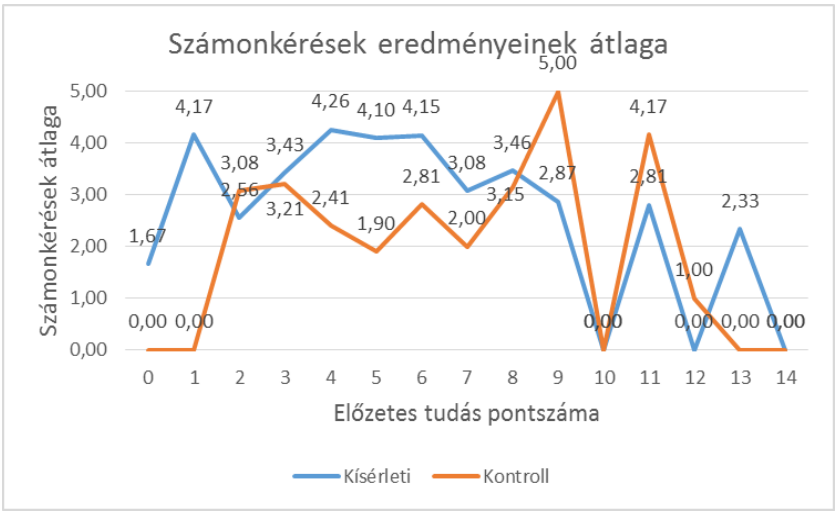
7. ábra. Fejlődés mértékének alakulása.



8. ábra. Fejlődés mértékének alakulása a [3,9) intervallumon.

A fenti diagramok alapján levonható az a következtetés, hogy a kísérletben résztvevő hallgatók 84%-ánál az AV-t használó csoportok nagyobb fejlődést értek el, mint a kontroll csoport hallgatói. Különösen az átlagos, illetve az alatti hallgatók felzárkózását segítette igazán az AV rendszerrel való tanítás. A kísérlet alapján azokat a hallgatókat nevezzük átlagos képességűnek, akiknek az előzetes tudást felmérő feladatokra kapott pontszámuk az [5,6) intervallumban volt. Ennek oka, hogy a kísérletben résztvevők átlagos pontszáma 5,74 volt. A legmagasabb fejlődést (0,79) azonban a [4,5) intervallumban pontszámot kapó hallgatók érték el a kísérleti csoportban.

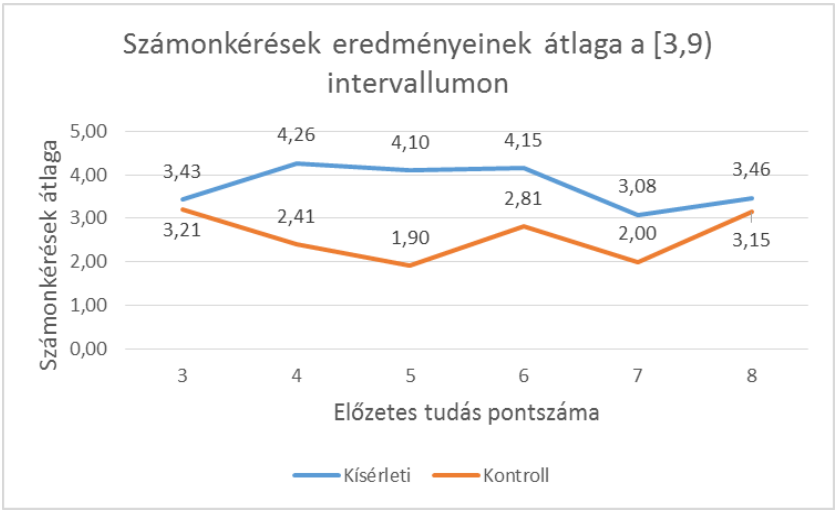
Külön vizsgálat tárgyát képezte a félév végi eredmények összehasonlítása.



9. ábra. Számonkérések eredményeinek átlaga.

Az eredmények vizsgálatánál nem a gyakorlati jegyeket vettem figyelembe, ugyanis a számonkérések átlaga jobban tükrözi a hallgató teljesítményét.

Látható, főként a hallgatók jelentős többségét magába foglaló [3,9) intervallumban, hogy különösen az átlagos hallgatóknál volt szignifikáns a különbség a kísérleti csoport javára. Ez az eredmény összefüggésben van a fejlődéssel is, tehát az átlagos előzetes tudású hallgatók felzárkózására és eredményeire volt pozitív hatással az AV eszközzel való tanítás.



10. ábra. Számonkérések eredményeinek átlaga a [3,9) intervallumon.

Az alábbi táblázatból látható, hogy az AV programot használók értek el jobb átlagot, majdnem egy jeggyel jobbat, mint a kontroll csoport hallgatói.

	Kísérleti csoport	Kontroll csoport
Számonkérések átlaga	3,58	2,65

Az AV programmal való tanulás hatással volt bukási arányra, ugyanis szignifikánsan kisebb arányú hallgató bukott meg a kísérleti csoportban, mint a kontroll csoportban. A kísérleti csoportban a hallgatók kétharmada legalább 3,5-ös átlagot ért el a számonkéréseken (a kontroll csoportban csak 37%).

Az AV absztrakt készségre gyakorolt hatását az 5. kérdés és a 2. csoportzárthelyi dolgozat eredményei mutatják meg. Az 5. kérdésre kapott pontszám mutatja meg, hogy a félév első hetében hol állt ez a készség, illetve a 2. csoportzárthelyi pedig azt, hogy fejlődött-e.

	Kísérleti csoport	Kontroll csoport
5. kérdés átlagos pontszáma	1,60	1,67
2. csoportzárthelyi átlaga	4,13	3,18

A fenti táblázat szerint a kísérleti csoport valamivel hátrébbról indult, mint a kontroll csoport, mégis majdnem 1 jeggyel jobb átlagot produkált, mint a kontroll csoport, ami azt jelenti, hogy az AV használata jobban fejleszti az absztrakciós készséget, mint ha nem került volna használatra ez az eszköz.

4 Összefoglalás

Bemutattam az AV történetét a múlt század 70-es éveitől napjainkig, külön kiemelve a vizualizáció oktatásban betöltött szerepének fejlődését.

A Jeliot 3 AV rendszert a Programozási alapismeretek kurzus tematikájához illesztettem, kiemelve azt, hogy az adott tudáselemek elsajátításában miként segít ez az AV rendszer.

2012 őszén egy összetett kétcsoportos pedagógiai kísérlet került megvalósításra az ELTE Informatikai Karán. A Programozási alapismeretek kurzus hallgatói közül 5-5 gyakorlati csoport vett részt a felmérés kísérleti és kontroll csoportjában.

Az első gyakorlaton a hallgatók kitöltöttek egy kérdőívet, amelyben az előzetes tudásukra kérdeztem rá. Két kérdés vonatkozott arra, hogy hány órában tanultak informatikát, illetve programozást középiskolában, illetve 3 egyszerű algoritmizálási feladatot kellett megoldaniuk, amellyel az algoritmikus gondolkodásuk szintjét határoztam meg.

Ez a bemeneti teszt szolgáltatta az alapot ahhoz, hogy a félév számonkérései alapján meghatározzam azt, hogy a hallgatók mennyit fejlődtek a félév eleje óta.

A kísérleti csoportok jobban teljesítettek, mind a fejlődés mértékét, mind a kurzus eredményeit tekintve. Ez különösen igaz volt az átlag, vagy kissé az alatti képességű hallgatókra. Ezért bizonyítva lett, hogy az AV eszközöket érdemes használni a programozás oktatásban, különösen kezdőknél.

5 Irodalomjegyzék

[1] Baecker, R. (1975) *Two systems which produce animated representations of the execution of computer programs*. SIGCSE Bulletin 7, 158-167.

[2] Ben-Bassat Levy, R., Ben–Ari, M., & Uronen, P. A. (2003). *The Jeliot 2000 program animation system.* Computers & Education, 40, 1–15.

[3] Bloom, B. S., & Krathwohl, D. R. (1956). *Taxonomy of educational objectives: The classification of educational goals*. Handbook I: Cognitive domain. Harlow, England: Longmans.

[4] Brown, M. H. (1988) *Algorithm Animation.* The MIT Press, Cambridge, MA.

[5] Fouh, E., Akbar, M. & A. Shaffer, C. (2012): *The Role of Visualization in Computer Science Education*, Computers in the Schools, 29:1-2, 95-117

[6] Hundhausen, C. D. (1999) *Toward effective algorithm visualization artifacts: designing for participation and communication in an undergraduate algorithms course*. Unpublished Ph.D. dissertation, Department of Computer and Information Science, University of Oregon.

[7] Hundhausen, C., Douglas, S. A., and Stasko, J. T.: *A meta-study of algorithm visualization effectiveness.* Journal of Visual Languages and Computing, 2002.

[8] Kátai, Z., Juhász, K., Adorjáni, A., K., *On the role of senses in education*, Computers & Education (2008), Vol. 51, No 4, 1707-1717, ISSN: 0360-1315.

[9] Kátai, Z., Kovács, L. I., Kása, Z., Márton, Gy., Fogarasi, K., Fogarasi, F.: *Cultivating algorithmic thinking: an important issue for both technical and HUMAN sciences*, Teaching Mathematics and Computer Science, 9 (2011) 1, 1-10.

[10] Kátai, Z., Toth L., *Technologically and artistically enhanced multi-sensory computer programming education*, Teaching and teacher education 26 (2010), 244-251.

[11] Lattu, M., Meisalo, V., Tarhio, J., *A visualisation tool as a demonstration aid*, Computers & Education, v.41 n.2, p.133-148, September 2003.

[12] Mayer, R. E. & Moreno, R. (1998, April). *A Cognitive Theory of Multimedia Learning: Implications for Design Principles*. Paper presented at the annual meeting of the ACM SIGCHI Conference on Human Factors in Computing Systems, Los Angeles, CA.

[13] Mayer, R. E. (1997). *Multimedia learning: Are we asking the right questions.* Educational Psychologist, 32, 1-19.

[14] Meier, A., Spada, H., & Rummel, N. (2007). *A rating scheme for assessing the quality of computer-supported collaboration processes.* International Journal of Computer Supported Collaborative Learning, 2(1), 63–86.

[15] Myller, N., Bednarik, R., Sutinen, E., & Ben–Ari, M. (2009). *Extending the engagement taxonomy: Software visualization and collaborative learning.* Transactions on Computing Education, 9(1), 1–27.

[16] Naps, T. L., Eagan, J. R., & Norton, L. L. (2000). *JHAVÉ—An environment to actively engage students in Web-based algorithm visualizations*. SIGCSE Bulletin, 32, 109–113.

[17] Naps, T. L., Rossling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C. D., Velazquez–Iturbide, J. (2003). *Exploring the role of visualization and engagement in computer science education.* SIGCSE Bulletin, 35(2), 131– 152.

[18] Pierson, W. C., & Rodger, S. H. (1998). *Web-based animation of data structures using JAWAA.* SIGCSE Bulletin, 30, 267–271.

[19] Rossling, G., Schuler, M., & Freisleben, B. (2000). *The ANIMAL algorithm animation* tool. In J. Tarhio, S. Fincher, & D. Joyce (Eds.),Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education (pp. 37–40), Helsinki, Finland.

[20] Sanders, D., Heeler, P., & Spradling, C. (2001). *Introduction to BlueJ: A Java developmentenvironment.* Journal of Computing Sciences in Colleges, 16(3), 257–258.

[21] Shaffer, C. A., Cooper, M., Alon, A., Akbar, M., Stewart, M., Ponce, S., & Edwards, S. H. (2010). *Algorithm visualization: The state of the field.* ACM Transactions on Computing Education, 10, 1–22.

[22] Stasko, J. T. (1997) *Using student-built animations as learning aids.* In: Proceedings of the ACM Technical Symposiumon Computer Science Education. ACM Press, New York, pp. 25-29.

[23] Szántó Sándor: *Az algoritmikus gondolkodás fejlesztése az általános iskolában*. Új Pedagógiai Szemle 2002/05

[24] Szlávi Péter, Zsakó László: *Módszeres programozás: Programozási tételek*. ELTE TTK Informatikai Tanszékcsoport, 1996.

[25] Szlávi Péter: *A programkészítési didaktika kérdései.* (Doktori disszertáció) ELTE, 2004.

[26] Szlávi Péter: *Programkészítés és gondolkodás.* Informatika a felsőoktatásban 2008 Konferencia, Debrecen

[27] Urquizza–Fuentes, J., & Velazquez–Iturbide, J. (2009). *A survey of successful evaluations of program visualization and algorithm animation system.* ACM Transac- tions on Computing Education, 9(2), 1–21.

[28] Urquiza-Fuentes, J., and Velázquez-Iturbide, J. (2013): *Toward the effective use of educational program animations: The roles of student's engagement and topic complexity*, Computers & Education, vol. 67, pp. 178 - 192

Internetes hivatkozások:

[I1] Brabec, F., & Samet, H. (2003). *Maryland spatial index demos Weboldal.* http://donar.umiacs.umd.edu/quadtree - Letöltve: 2013. június 18.

[I2] Code Blocks weboldal. http://www.codeblocks.org – Letöltve: 2013. július 23.

[I3] Dittrich, J.-P., van den Bercken, J., Schafer, T., & Klein, M. (2001). DSN: *Data structure navigator.* http://dbs.mathematik.uni-marburg.de/research/projects/dsn/index.html.bak - Letöltve: 2013. június 18.

[I4] Galles, D. (2006). *Data structure visualization.* http://www.cs.usfca.edu/~galles/visualization - Letöltve: 2013. június 18.

[I5] Gustafson, B. E., Kjensli, J., & Vold, J. M. (2011). *Binary treesome Weboldal.* http://www.iu.hio.no/~ulfu/AlgDat/applet/binarytreesome – Letöltve: 2013. június 18.

[I6] Jarc, D. J. (1999). *Interactive data structure visualization.* http://nova.umuc.edu/~jarc/idsv - Letöltve: 2013. június 18.

[I7] Korhonen, A., Malmi, L., Silvasti, P., Nikander, J., Tenhunen, P., Mård, P., Karavirta, V. (2003). *TRAKLA2.* http://www.cse.hut.fi/en/research/SVG/TRAKLA2 - Letöltve: 2013. június 17.

[I8] Rodger, S.H. (2008). *JFLAP Weboldal.* http://www.jflap.org – Letöltve: 2013. június 18.

[I9] Stasko, J. T. (1998). *JSamba.* http://www.cc.gatech.edu/gvu/ii/softvis/algoanim/jsamba - Letöltve: 2013. június 17.

[I10] Stern, L. (2001). *Algorithms in action.* http://ww2.cs.mu.oz.au/aia - Letöltve: 2013. június 18.

[I11] *Virginia Tech Algorithm Visualization Research Group Weboldal.* (2011). http://research.cs.vt.edu/AVresearch - Letöltve: 2013. június 18.

6 Függelék

6.1 Előzetes tanulmányokat felmérő kérdőív

1. Heti hány órában tanultál informatikát középiskolában?
   - 0
   - 1
   - 2
   - 3
   - 4
2. Hány órát tanultál középiskolában programozást összesen?
   - 0
   - 1-4

- 5-8
- 9-14
- 15 vagy annál több

3. Írj algoritmust (saját szavaiddal) arról, hogy hogyan mész át egy jelzőlámpás kereszteződésen!
4. Írj algoritmust (saját szavaiddal) arról, hogy hogyan kell használni egy kávéautomatát!
5. Írj algoritmust (saját szavaiddal) a következő feladathoz: Gondolj egy pozitív egész számra, és olvasd be! Nem pozitív egész esetén jelezd a hibát és lépj ki a programból. Helyes szám beolvasása után írd ki a beolvasott számot a képernyőre!