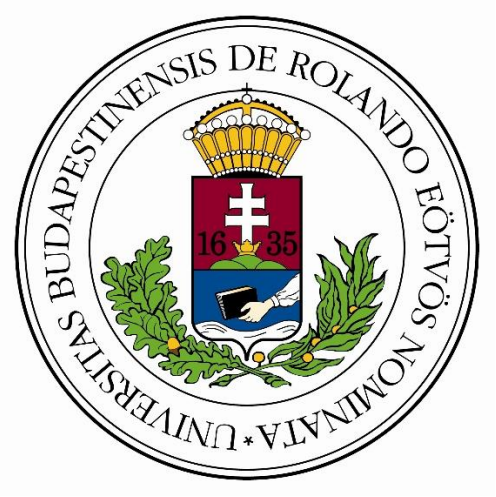


VIZUALIZÁCIÓ A PROGRAMOZÁSTANÍTÁSBAN

DOKTORI ÉRTEKEZÉS

TÖRLEY GÁBOR

2013. (2014-ben nem hivatalosan javítva)

Eötvös Loránd Tudományegyetem Informatika Doktori Iskola
Az informatika alapjai és módszertana Doktori program

Iskolavezető: Dr. Benczúr András
matematikai tudományok doktora, egyetemi tanár

Programvezető: Dr. Demetrovics János
matematikai tudományok doktora, egyetemi tanár

Témavezető:
Dr. Szlávi Péter
egyetemi docens

Budapest, 2013.

Tartalomjegyzék

1 Bevezetés

A kutatásom oktatás-módszertani jellegű, és az algoritmikus gondolkodás, a problémamegoldó képesség fejlesztésére fókuszál.

A kutatás kiindulópontjai a következők:

1. A PISA-jelentések [10] és más hazai mérések [27] szerint a mai magyar tizenévesek a problémamegoldó-képesség, a kreativitás tekintetében csak a középmezőnybe tartoznak a nemzetközi összehasonlításokban. Ezért aktuális oktatási cél a tanulók problémamegoldó gondolkodásának mielőbbi és minél hatékonyabb fejlesztése.
2. A jelenlegi középiskolai tantervek a programozásra legfeljebb 20%-nyi óraszámot szánnak az amúgy sem nagyon sok informatika-órákon belül. Néhány gimnázium-ban van, de jellemzőbben a szakközépiskolákban tanítanak programozni.
3. A programozáshoz használt nyelvek is – többnyire – valamely, DOS-os környezetben futó magasabb szintű programnyelvet jelent, pl. Pascal, C, C++, BASIC.
4. A programozás köztudottan a legnehezebb fejezete az informatikának, különösen, ha az illető tanuló nem is motivált ennek elsajátításában.
5. A programozásoktatás egyik alkalmas eszközének látszik az 1.-ben megfogalmazott probléma megoldására.
6. 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.

Szántó szerint [40] 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 végig gondolja, 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 [22, 47] –, nehéz tanulni és tanítani ezeket. A tanítási módszerek fejlesztése ezen a területen fontos feladata a mai informatikaoktatásnak. [42, 43]

1.1 A témaválasztás indoklása

2005-ben, a diplomamunkám keretein belül végeztem egy kérdőíves vizsgálatot 70 tanuló részvételével. Ennek egyik eredménye: a megkérdezett diákok alig harmada gondolta azt, hogy hasznos számára a tankönyv a programozás tanulásában. Ez felveti azt a kérdést, hogy miért van ez így, továbbá, hogy egy másik eszköz bevetésével ki lehet-e egészíteni a tankönyv szöveges, illetve a tanár szóbeli információit?

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 *hallható* 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 [24]:

- 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. [23] 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.

A programozástanítás egyik szegmense, az objektum orientált programozás (OOP) az elmúlt évtizedekben a leghatásosabb programozási paradigmává vált. Széles körben használják az oktatásban, az iparban, és majdnem minden egyetem tantervében előfordul az objektum orientáltság fogalma. A programozói közösségek szerint is hasznos OOP-t tanítani, mert ez a paradigma támogatja a strukturált programozás, a modularizáció és a programtervezés koncepciójának tanítását.

A fent említett strukturáltság mellett az OOP-s szemlélet természetes módon képes a valós világot leképezni, hiszen a körülöttünk levő „dolgok” is leírhatók tulajdonságokkal, állapotokkal és állapotváltozásokkal (metódusokkal), ilyen módon az objektum orientált paradigma fejleszti az absztrahálás képességét.

Mégis, az objektum orientált programozást nehéz tanítani, egyrészt az absztrakt gondolkodás szükségessége miatt, illetve azért is, mert az objektum, amit a programozás során létrehozunk, csak a programozási folyamat végére konkretizálódik, ölt testet.

Egy objektum működésének megértése, és aztán kódolása, a fenti tapasztalat miatt ugyancsak kevesebb időbe kerülhet, ha vannak olyan eszközeink, amelyek segítenek abban, hogy elképzeljük az adott objektumot.

Kölling szerint [21] néhány hallgatónak azért nehéz megérteni az OOP terminológiáját, mert mindaz, amit a képernyőn és a tanár prezentációjában lát, az maga a programkód. Ha elvárás a hallgatótól, hogy gondolkodni és beszélni tudjon az osztályokról, a kapcsolataikról, akkor vizuálisan célszerű azokat megjeleníteni. Ugyanez az állítás igaz, amikor a program futási idejében vizsgáljuk az objektumokat. Vizuális reprezentációval könnyebben meg lehet érteni az OOP terminológiáját.

1.2 Előzmények és célkitűzések

A 2007/2008-as tanév őszi szemeszterét Helsinkiben töltöttem, az Erasmus ösztöndíjprogram keretein belül. Kutatást végeztem a finnországi informatika- (és azon belül) programozásoktatási módszerekről. Órát látogattam, és középiskolában, illetve egyetemen tanító tanárokkal készítettem interjúkat. Magam is részt vettem programozásoktatásban, mint hallgató. Itt találkoztam egy új eszközzel, a Jeliot-tal, és egy rajta nyugvó oktatási elképzeléssel, az algoritmus vizualizációval. Ezzel az eszközzel és ezen a módon támogatják a programozás tanítását és tanulását középiskolai szinten is Izraelben és az Amerikai Egyesült Államokban. Tapasztalataimról jelentést írtam. [I18]

A kutatás célja, hogy bemutasson egy algoritmus vizualizációs (AV) eszközt, és egy ehhez kapcsolódó vizsgálati és tanítási módszert, amely különösen a kezdő vagy az átlagos vagy az alatti képességű hallgatókat segíti a felzárkózásban. Hipotéziseim a következők:

- Hipotézis 1.: Az AV beilleszthető a közoktatási informatikába.
- Hipotézis 2.: Az AV eszközzel való tanítás az átlag közeli tudású, *felsőfokú* oktatásban részt vevő hallgatók felzárkózását segíti.
- Hipotézis 3.: Az AV-t használó *felsőfokú* oktatásban részt vevő hallgatók többet fejlődnek és jobb eredményeket érhetnek el, mint a vizualizációt nem használók.
- Hipotézis 4.: *Felsőfokú* oktatásban részt vevők esetén az AV használata hatékonyabban fejleszti a formalizált algoritmusok megértését és alkalmazását, mintha nem kerülne használatra ez az eszköz.

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. [18]

Egy változó, dinamikus folyamatot nagyon nehéz bemutatni statikus eszközökkel, mint például szövegekkel és képekkel egy tankönyvben. Ugyanez igaz egy „klasszikus" prezentációra is. Bár prezentációs szoftverekben, mint ahogy fentebb saját korábbi kísérletemre utaltam, van lehetőség animációs betéteket készíteni, ez azonban nagyon időigényes munka, és igen munkaigényes, sőt általában lehetetlen újra felhasználni (azaz pl. az algoritmust egy kicsit megváltoztatni és aztán „újraanimálni"). A tanítási gyakorlat is azt mutatja, hogy amikor táblánál magyarázom az algoritmus működését, akkor is egy folyamatot próbálok illusztrálni: az algoritmus legjellegzetesebb adatainak állapotváltozásait dokumentálva hajtom végre az algoritmust. A legtöbb tanulónak még ilyen módon magyarázva is nehéz minden részletében megérteni az algoritmust. Emiatt irányult a külföldi oktatók figyelme a vizualizációra, amellyel könnyebben, problémamentesebben lehet átadni az algoritmus dinamizmusát. [11]

Az AV a múlt század 70-es éveinek végén a batch-oritentált szoftverekből – amelyek lehetővé tették az oktatóknak animációs filmek készítését [2] – mára magas szintű interakcióval rendelkező rendszerekké fejlődött, amelyekkel a tanulók felfedezhetik, beállíthatják, dinamikusan megváltoztathatják az algoritmus animációját a saját igényeiknek, kíváncsiságuknak megfelelően [6, 35], vagy maguk képesek megalkotni saját vizualizációjukat [17, 38]. Ezt a történelmi folyamatot tekintem át röviden.

2.1 A kezdetek

Az első rendszerek, mint pl. BALSA [7], Tango [36], XTango [37], Samba [39] és Polka [I16], nem terjedtek el széles körben, az implementációs technológiájuk bonyolultsága miatt. A Massachusetts Institute of Technology (MIT) Athena Projektje [8] nemcsak egy interaktív eszközt hozott létre, hanem megoldotta az elterjedés problémáját is azzal, hogy az X Windows megjelenítő rendszert használta. Emiatt sok AV rendszer épült az X Windowsra a 1980-as évek végén, 1990-es évek elején. Ennek a hátránya az volt, hogy a kor oktatásra használt

számítógépes laborjai általában nem rendelkeztek olyan hardverű számítógépekkel, amelyek biztonsággal tudták volna futtatni az X Windows rendszert.

Hundhausen és munkatársai összefoglaló tanulmánya [18] vizsgálta először az AV pedagógiai hatásait. E szerint, az intuitív vonzereje ellenére az AV nem terjedt el pedagógiai eszközként az informatikaoktatásban. Az egyik ok a sok közül, hogy a tanárok a befektetett szellemi beruházáshoz képest csekély eredményt kaptak vissza. Igazság szerint, valóban, az eddig publikált eredmények nem nyújtanak egységes képet az AV eredményességéről, nem minden esetben bizonyosodott be, hogy *szignifikánsan* jobb tanulási eredmények születtek az AV használatával. A cikk több empirikus vizsgálat eredményét elemezte és kiemelte, hogy a kognitív konstruktivista elméletet támogató vizsgálatokból publikálták eddig a legtöbb eredményt, és ezek 71%-a mutatott ki szignifikáns különbséget az AV-t használó csoport javára a kontrollcsoport eredményeihez képest. Várhatóan azoknál a csoportoknál, ahol a feladat több erőfeszítést kíván, ott fognak szignifikáns különbségek kijönni a kontrollcsoporthoz képest. Általánosságban a tanulmány hatékonynak találta az AV technológiát, de nem abban az értelemben, hogy „egy kép felér ezer szóval”. Inkább a tanulási feladat formája, amely végrehajtása során az AV-t használják, a fontosabb, mint a vizualizáció minősége. Természetesen ez nem jelenti azt, hogy nem számít a minőség. A jól tervezett vizualizáció támogatja a sikeres tanulási tevékenységet.

2.2 Algoritmus vizualizáció az Internet korában

Sok minden változott az 1990-es évek közepén, az Internet és a Java programozási nyelv alkalmazásának egyre nagyobb elterjedésének következtében. Így sokkal több oktatóhoz és hallgatóhoz jutott el az AV. Ezeken kívül az elterjedéshez az is hozzájárult, hogy egyre több hallgatónak volt saját számítógépe az egyetemi kampuszokon, illetve otthon. Az imént említett változások hatására a 1990-es évek végén és a 2000-es évek elején új generációja született meg az AV rendszereknek. Példák ezekre a rendszerekre: JSamba [I15], JAWAA [32], JHAVÉ [30], ANIMAL [33], 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, amelyek fejlesztői környezetet kínáltak azoknak, akik AV-kat akartak fejleszteni. 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) [I4], Interactive Data Structure Visualization (IDSV) [I10], Algorithms in Action [I17], Data Structure Visualization [I5] és TRAKLA2. [I12]

A Java használatának másik hatása az volt, hogy oktatók és hallgatók is elkezdtek fejleszteni AV rendszereket. Egy AV építése jó módszernek látszott arra a 1990-es évek közepén, hogy a hallgatók megtanulják a Java nyelvet. Sajnos, a legtöbb hallgató által írt AV rendszer pedagógiai értéke igen csekély volt. [35] Szerencsére vannak pozitív példák is: általában sok év kitartó munkájának gyümölcseként születtek olyan Java kisalkalmazások, amelyek bemutattak egy-egy témát, területet. Ilyenek voltak például: Examples include Binary Treesome [I6], JFLAP [I13], Marylandi Egyetem Spatial Index Demos [I2] és a Virginiai Egyetem Algoritmus-vizualizációs Kutatócsoportja (2011) által készített rendszer. [I19]

2.3 Vizualizáció és aktivitás

Naps és munkatársai tanulmányukban [31] kiterjesztették Hundhausen és munkatársainak következtetéseit [18] 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. [5]

Urquiza–Fuentes és Velázquez–Iturbide [46] 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. [35]

A fenti alapkutatás arra az eredményre is jutott, hogy minél magasabb szinten van egy AV rendszer, illetve minél magasabb szinten alkalmazza a tanuló az AV rendszert, annál több tanulási hasznot lehet „kinyerni” az AV-val.

Myller, Bednarick, Sutina, és Ben–Ari végzett kutatást [28] 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 kéz új szint a 2. és a 3. közé került be. Az *irányított 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 [28] elméletüket: párokban dolgoztak hallgatókkal, akik a BlueJ [34] és a Jeliot 2000 [3] 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. [25]

2.4 További általános megállapítások

Az AV program segítette

- az oktatót az algoritmus illusztrálásában, [6]

- a tanulókat abban, hogy megértsék az alapvető algoritmusok működését, [14]
- a hibakeresést konzultáción, [15]
- a tanulókat megérteni egy absztrakt adattípus műveleteinek működését (tehát a típus lényegét). [29]

Milyen tulajdonságokkal jellemezhető egy jó demonstrációs eszköz? [22]

- Rugalmasság
    - Platform független: nem szükséges átírni a programot a vizualizáció számára
    - Lehetőség van különböző magyarázat-stratégiákat alkalmazni
    - A felhasználónak lehetősége van a kód csak egy részét vizualizálni
- Programszerkezet, adatszerkezet, objektumok
    - A programszerkezetet jelenítse meg
    - Kövesse a program végrehajtását
    - Jelenítse meg az adatszerkezetet, és az adatok változását
    - Jelenítse meg az objektumokat és az átadott paramétereket

Előnyös, ha az AV szoftver teljes és folyamatos vizualizációt nyújt, tehát minden elemnek (konstans, változó, adatszerkezet, objektum) lesz vizuális megfelelője, illetve világosan megmutatja a program tevékenységei, eljárásai közötti kapcsolatokat. [3]

Kehoe és munkatársai tanulmányukban [19] házifeladat-szerű, valóságosabb tanulási szituációban vizsgálták a tanulókat. Ez azt jelenti, hogy a kurzus teljes idejét figyelték, nem csak a vizsgaeredmény alapján értékeltek. Két csoport tanult a binomiális kupacról, azzal a különbséggel, hogy az egyik csoport animációkhoz is hozzáférhetett tanulás közben. Sokat segített az analógia- és a fogalomalkotásban, hogy nemcsak az animációt, hanem a kódot is látták az animációval egy időben. Jobb eredmények születtek az animációt használó csoportnál, de a szerzők megjegyezték, hogy a vizsgálatot jó képességű tanulókkal végezték, gyengébb képességűeknél, valószínűleg, más eredményt kaptak volna. Az egyik legszembetűnőbb különbség a két csoport motivációja között volt. Az animációt használt csoport teljes szívvel vett részt az órákon, sokkal nyugodtabb és biztosabb volt a tudását illetően, sokkal nyitottabbak voltak a tanulásra, átlagosan több időt foglalkoztak a

tananyaggal, mint a másik csoport. Az algoritmus animáció tűnik a leginkább alkalmasnak, hogy segítsen továbbítani egy algoritmus műveleteit lépésről lépésre, így világos vizuális reprezentációt nyújt egy amúgy absztrakt folyamatról.

Az algoritmus-vizualizációs eszközök használata Magyarországon szinte ismeretlen. A Google keresője szerint mindössze csak egy magyar nyelvű és vonatkozású írás [44] jelent meg ebben a témában, holott, nemzetközi kutatások sok pozitív eredményt mutattak fel az elmúlt 40 évben.

2.5 Néhány eszköz bemutatása

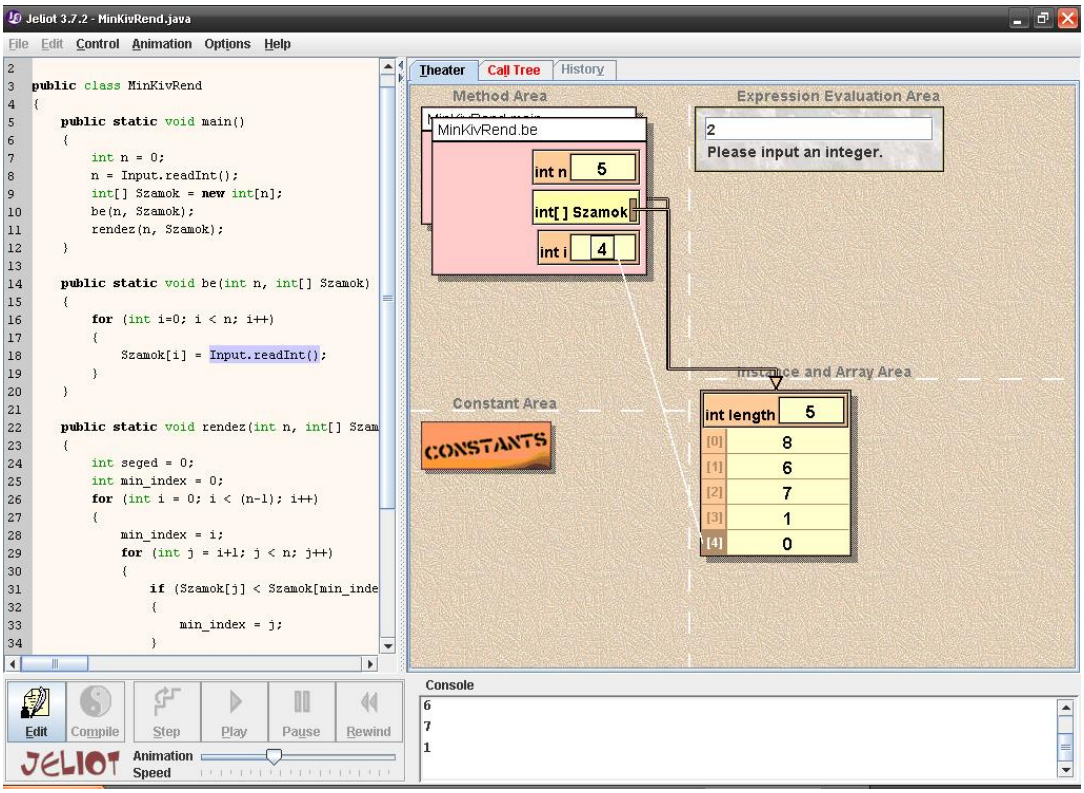
Három, korábban már említett eszközt mutatok be röviden.

2.5.1 Jeliot 3

A Jeliot 3 egy ingyenes, Java alapú AV rendszer [I11]. A Jouensuui Egyetem (Finnország) kutatói fejlesztik 1997 óta. A jelenlegi 3-as verzió 2004-ben készült el.

Ez a program-vizualizációs környezet a kezdő programozók számára készült, amelyben animációk reprezentálják Java programok lépésenkénti végrehajtását (1. ábra). A program végrehajtásának minden lépése világosan látható, és az animáció azt szimulálja, hogy a virtuális gép hogyan interpretálja a programkódot. Az animáció helyszíne a „színház” (Theater), amelyet négy részre osztottak: a középső és a fő rész a „Kifejezés kiértékelés” (Expression Evaluation), amelyhez üzenetek, metódushívások, értékek és hivatkozások érkeznek a többi vizualizációs területről. A „színház” bal alsó részén láthatóak a kódban deklarált konstansok (Constant Area), valamint a jobb alsó részen a objektumpéldányok és a tömbök reprezentációi (Instance and Array Area). A tanulóknak lehetőségük van programozni a Jeliot 3-ban és később annak működését követni tudják az animáción keresztül.

Találóan nevezték el „színháznak” a program készítői azt a területet, ahol az animáció történik. A program (az algoritmus) a forgatókönyv, amely szerint a változók, adatszerkezetek eljátsszák a szerepüket, és a tanuló a rendező. Így az animáció a programozási folyamat és nem csak a tanulási folyamat része. Nagyon előnyös tulajdonsága, hogy a kód és az animáció egyszerre látható. A környezet képes a folyamatos és a lépésenkénti animációra, sőt a tanuló ki tudja jelölni azt a részt, amit animálva akar látni.



1. ábra. *Jeliot*

A könnyebb érthetőség kedvéért – eltérve a Java nyelvtől – a main eljárásból elhagyták a kötelező `String[] args` paramétert. A Java nyelv elemeinek igen sok részét elfogadja a környezet (egyik kivétel pl. a `Scanner` osztály).

A környezet nagyon hatékonyan használható hibakeresésre is, mivel minden aktuális változó értékét nyomon lehet követni, illetve lehetséges módosítani a változók értékét.

Előnye a rendszernek, hogy platform független, többek között Windows, Linux és Mac OS rendszereken is használható.

Sok bíztató vizsgálati eredménnyel rendelkezik a Jeliot 3. Egy teljes kurzust megfigyelve [3] több időpontban mérték a tanulók teljesítményét. Kiderült, hogy a rendszer támogatja a frontális előadást, segít megérteni a tanár magyarázatait, és a közepes képességű tanulók eredménye fejlődött a legtöbbet, különösképp abban, ahogyan definiálták és elmagyarázták a fogalmakat.

A tanulási folyamat a figyelemmel kezdődik. [20] Erre a kijelentésre alapozva végzett Ebel és Ben-Ari vizsgálatokat. [9] A vizsgálat a tanulók figyelmére összpontosított, ugyanis a figyelem mértéke jól korrelál a tanulás hatékonyságával. Magatartás- és figyelemzavaros

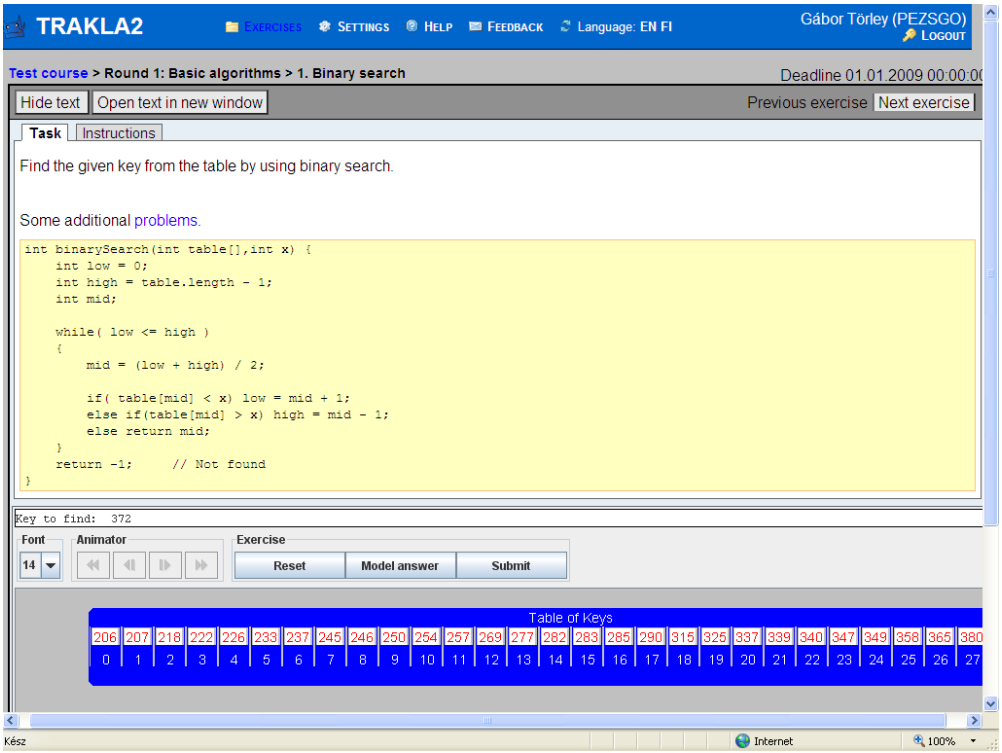
gyerekeket tanítottak a Jeliot 3 segítségével, és azt tapasztalták, hogy az órák azon részén, amikor a Jeliot-tal tanította, magyarázta a tanár az órai anyagot, nem volt magatartás- vagy figyelemzavar a tanulók részéről.

Moreno és Joy vizsgálata [26] a rendszer határaira mutatott rá: a Jeliot 3 nem elég flexibilis ahhoz, hogy különböző tudásszinten levő tanulókat vagy használati sablonokat támogasson. Általánosan igaz, hogy az átlag alatti vagy feletti tanulók teljesítményénél nem tapasztaltak szignifikáns különbséget. [3]

2.5.2 Trakla2

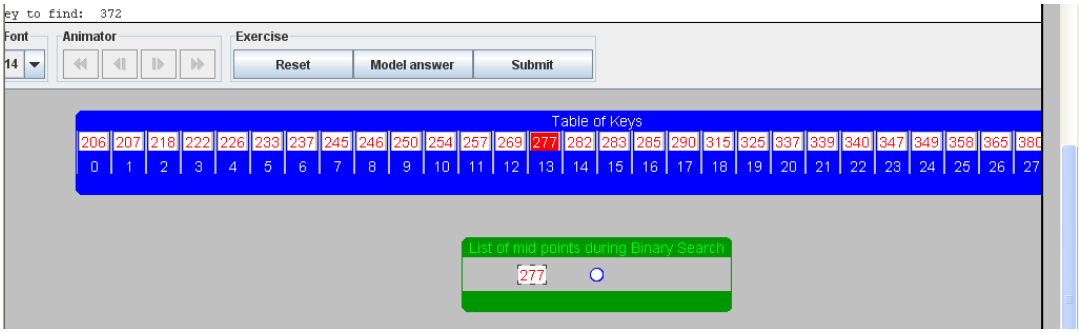
A TRAKLA2 [I12] algoritmus-vizualizációs (AV) rendszer egy teljesen más stratégiát és módszert követ. Abból a célból fejlesztették, hogy elsőéves egyetemisták programozás-tanulmányait segítsék.

Ez a rendszer nem animációval támogatja a tanulást, pontosabban az animációt magának a tanulónak kell „eljátszania” az algoritmus alapján, majd a rendszer lepontozza azt. Mostani példánkban a logaritmikus vagy bináris keresés algoritmusát fogjuk felhasználni.



2. ábra. *TRAKLA2*

Hasonlóan a Jeliot 3-hoz, ez a rendszer is egyaránt támogatja a tanár előadását és a tanuló önálló tanulását. Gyakorlatilag, mindketten eljátsszák az algoritmust, olyan módon, hogy a lenti ábrán látható (3. ábra) zöld „dobozba” húzzák bele a középső elemeket.



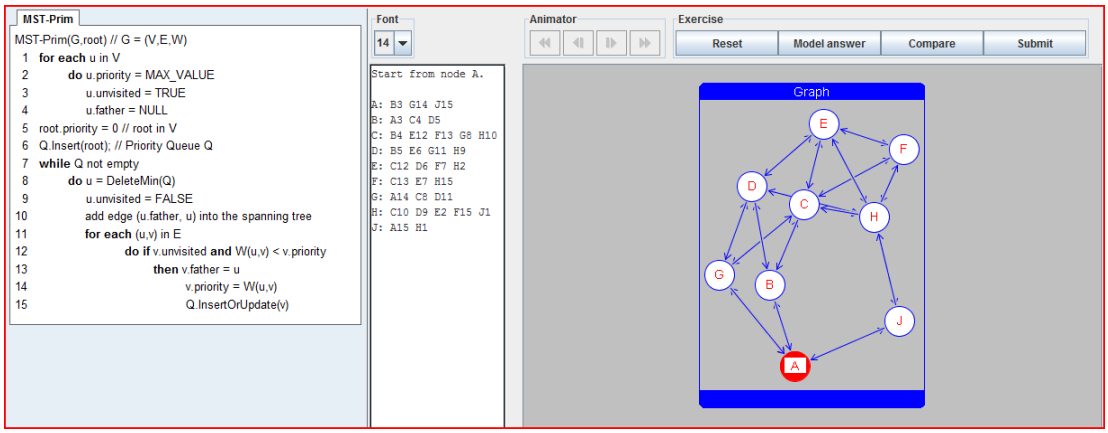
3. ábra. *Feladatmegoldás közben*

Miután megadtuk a megoldásunkat, a rendszer visszajelzi az eredményt, és lehetőséget ad arra, hogy visszanézzük a megoldásunk lépéseit, illetve, ugyancsak lépésenként, megmutatja a helyes megoldást is, ha mi közben nem jöttünk volna rá.

Ez a rendszer alkalmas arra, hogy házi feladatokat adjon fel a tanár, amelyek eredményeit meg tudja nézni, így nyomon tudja követni a tanulók teljesítményét.

A TRAKLA2 nagyban támogatja az otthoni, önálló gyakorlást, tanulást, hiszen a tanuló több bemeneten tudja „eljátszani” az algoritmus működését, és a megoldás után azonnal tájékoztatást kap annak sikerességéről. A környezet nagy előnye, hogy így magát az algoritmust (illetve annak működését) gyakorolja, és nem kódol, tehát tudása nem fog valamelyik programnyelvtől függeni. Ez az állítás annak ellenére igaz, hogy a rendszer által használt algoritmikus nyelv nagyon hasonlít a Java nyelvhez. Valójában ennek oka nem több, mint a rendszer által használt angol nyelv. Mivel a TRAKLA2 is ingyenes és szabadon fejleszthető, magyarítás után hasznos eszköz lehet a hazai felsőoktatásban Algoritmusok és adatszerkezetek tantárgynál, illetve egyszerűbb algoritmusokkal kiegészítve a közoktatás hasznos segédeszközévé is válhat.

Jól látható az ábrákon, hogy a környezet nemcsak az algoritmus működését mutatja meg, hanem az adatszerkezetet is jól szemlélteti (a bináris keresés esetében tömb). Ilyen módon, pl. gráfalgoritmusok esetén, a gráf adatszerkezetén lehet „végig játszani” az algoritmust (4. ábra).

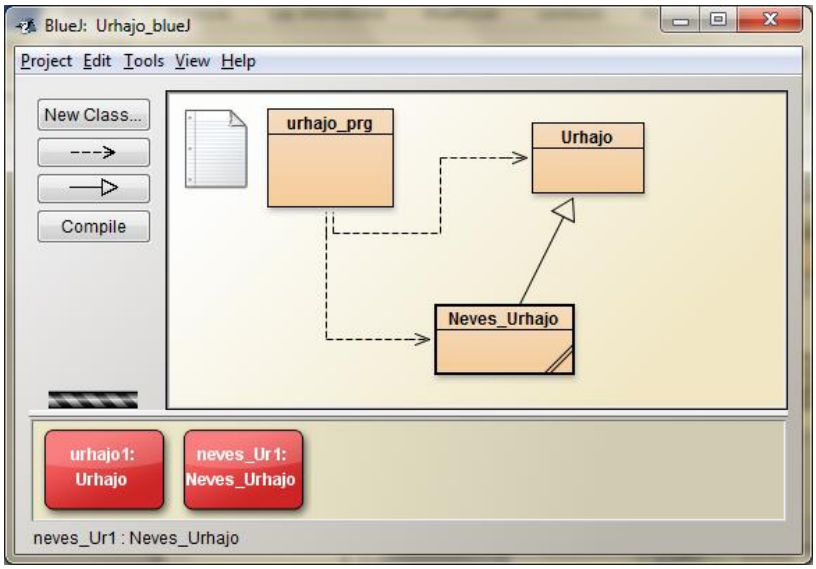


4. ábra. *Példa gráfalgoritmusra*

2.5.3 BlueJ

A BlueJ [I1] programozási környezetet egy egyetemi kutatási projekt részeként fejlesztette Michael Kölling és John Rosenberg, abból a célból, hogy egy könnyen használható, oktatásra alkalmas szoftvert hozzanak létre, amellyel kezdő, első évfolyamos hallgatóknak tanítottak objektum orientált programozást. A készítők nagy hangsúlyt fektettek a vizualizációra és az interakciós technikákra, hogy olyan programozási környezetet teremtsenek, amely elősegíti kísérletezést és a felfedezést.

A BlueJ környezet előnye, hogy grafikus reprezentációt kínál a projekt által használt objektumokról és az osztályokról, amelyekkel a hallgatók egyszerűen felugró menükön keresztül tudnak kommunikálni. Ezen kívül lehetőség van arra is, hogy az objektumok aktuális állapotát is megtekintsük.



5. ábra. *BlueJ*

A környezet lehetővé teszi azt, hogy azok a kapcsolatok, amelyeket grafikusan megjelenítünk, a kódban is megjelenjenek, pl. az 5. ábrán látható öröklési kapcsolat az Űrhajó és a Neves_Űrhajó osztály között.

A fenti két szoftvert a BlueJ-hez készített Jeliot 3 kiegészítő [I1] köti össze, amely segítségével a BlueJ-ből futtatni lehet a Jeliot 3-at.

3 A vizualizáció alkalmazásának lehetősége a programozástanításban

Az 51/2012. (XII. 21.) EMMI rendelet szól a kerettantervek kiadásának és jóváhagyásának rendjéről, illetve mellékletként tartalmazza a kerettanterveket. A kerettanterv definiálja azokat az ismeretköröket, amelyekkel rendelkeznie kell a tanulónak az adott osztály elvégzésekor, illetve meghatározza az egyes műveltségi területekhez tartozó heti minimális óraszámot. Az óraszámokat, illetve a százalékos eloszlásokat az alábbi táblázat szemlélteti.

Óraterv a kerettantervekhez – 5-12. évfolyam, gimnázium											
Tantárgyak	**5. évf.**	**6. évf.**	**7. évf.**	**8. évf.**	**9. évf.**	**10. évf.**	**11. évf.**	**12. évf.**	**Össz./hét**	**Össz.**	**%**
Testnevelés és sport	5	5	5	5	5	5	5	5	40	**1440**	**15,44%**
Magyar nyelv és irodalom	4	4	3	4	4	4	4	4	**31**	**1116**	**11,97%**
Matematika	4	3	3	3	3	3	3	3	25	**900**	**9,65%**
I. idegen nyelvek	3	3	3	3	3	3	3	3	24	**864**	**9,27%**
Történelem, társadalmi és állampolgári ismeretek			2	2	2	2	3	3	14	**504**	**5,41%**
II. idegen nyelv					3	3	3	3	12	**432**	**4,63%**
Biológia-egészségtan			2	1		2	2	2	9	**324**	**3,47%**
Fizika			2	1	2	2	2		9	**324**	**3,47%**
Osztályfőnöki	1	1	1	1	1	1	1	1	8	**288**	**3,09%**
Kémia			1	2	2	2			7	**252**	**2,70%**
Földrajz			1	2	2	2			7	**252**	**2,70%**
Ének-zene	1	1	1	1	1	1			6	**216**	**2,32%**
Vizuális kultúra	1	1	1	1	1	1			6	**216**	**2,32%**
Informatika		1	1	1	1	1			5	**180**	**1,93%**
Erkölcstan	1	1	1	1					4	**144**	**1,54%**
Társadalomi, állampolgári és gazdasági ismeretek / Latin örökségünk*	2	2							4	**144**	**1,54%**
Természetismeret	2	2							4	**144**	**1,54%**
Művészetek**							2	2	4	**144**	**1,54%**
Technika, életvitel és gyakorlat	1	1	1					1	4	**144**	**1,54%**
Etika							1		1	**36**	**0,39%**

Dráma és tánc/Hon- és népismeret*	1								1	**36**	**0,39%**
Dráma és tánc/Mozgóképkultúra és médiaismeret*					1				1	**36**	**0,39%**
Szabadon tervezhető órakeret	**2**	**3**	**3**	**3**	**4**	**4**	**6**	**8**	**33**	**1188**	**12,74%**
Rendelkezésre álló órakeret	**28**	**28**	**31**	**31**	**35**	**36**	**35**	**35**	**259**	**9324**	**100,00%**

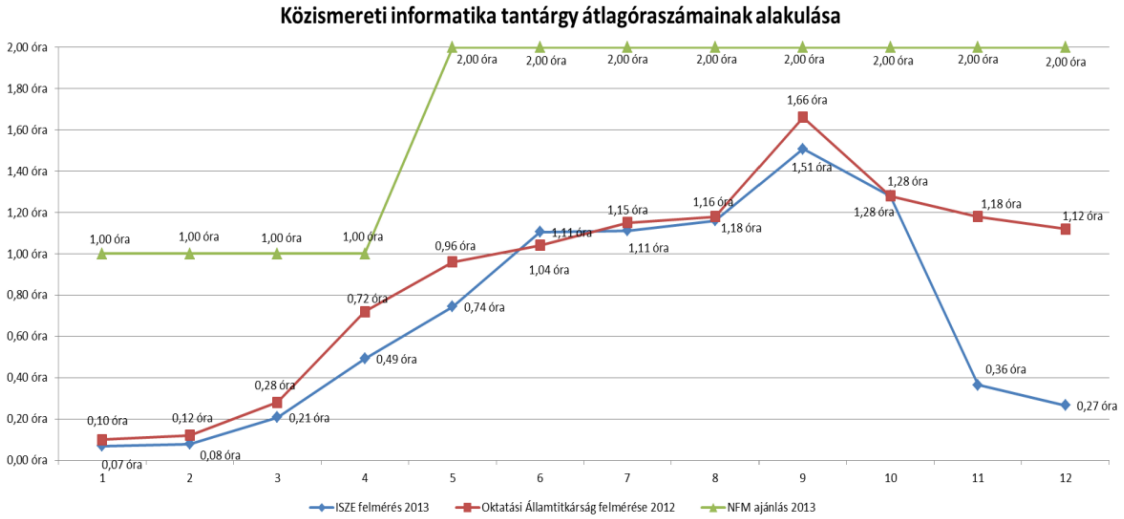
Az adatokból látszik, hogy egy tanuló minimum 180 tanítási órát tölt el informatika tantárgy tanulásával az iskolában, ez a teljes minimális órakeret kevesebb, mint 2%-a.

A Nemzeti Fejlesztési Minisztérium (NFM) Infokommunikációért Felelős Államtitkársága ajánlást [I8] adott ki az önálló informatika tantárgyról. Az ajánlás lényege az, hogy mivel az informatika a magyar gazdaság motorja, ezért a szabadon tervezhető órakeret kb. felét-harmadát használják informatika oktatásra (az alábbi táblázatból kiolvasható). Ez a dokumentum nem számol a Társadalmi, állampolgári és gazdasági ismeretek / Latin örökségünk tárggyal, ezért van 2-vel több szabadon választható időkeret 5. és 6. évfolyamban. A táblázatból látható. hogy az ajánlás megháromszorozná az informatika tantárgy minimális óraszámát.

Óraterv a kerettantervekhez – 5-12. évfolyam, gimnázium											
Tantárgyak	**5. évf.**	**6. évf.**	**7. évf.**	**8. évf.**	**9. évf.**	**10. évf.**	**11. évf.**	**12. évf.**	**Össz./hét**	**Össz.**	**%**
Informatika		1	1	1	1	1			5	**180**	**1,93%**
Szabadon tervezhető órakeret	**4**	**5**	**3**	**3**	**4**	**4**	**6**	**8**	**37**	**1332**	**14,29%**
Ebből informatika	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**2**	**10**	**360**	**3,86%**
Informatika	1	2	2	2	2	2	2	2	15	**540**	**5,79%**
Rendelkezésre álló órakeret	**28**	**28**	**31**	**31**	**35**	**36**	**35**	**35**	**259**	**9324**	**100,00%**

Az Informatika-Számítástechnika Tanárok Egyesülete és az Oktatási Államtitkárság felmérést [I7] készített arról, hogy az iskolák átlagosan hány órát szánnak a helyi tantervükben informatikára. Az alábbi ábrán látszik, hogy az NFM dokumentumát valóban csak ajánlásként használták. Megjegyzem, hogy az Oktatási Államtitkárság felmérése reprezentatív volt, míg egyesületé nem volt az. Az egyesület felmérése alapján a rendelkezésre álló órakeret 2,92%-át, míg az Államtitkárság adatai alapján a 3,69%-át használnák informatika oktatására.

Megfigyelhető az óraszámok drasztikus csökkenése 11-12. évfolyamon, ami azt valószínűsíti, hogy kevesebben fogják választani az informatikát érettségi tárgyként.



6. ábra. *A felmérés eredménye*

A fenti átlagos óraszámokra azért van szükség, hogy megbecsüljem, különböző helyi tantervek, tantervjavaslatok alapján, hogy az óraszámokból mennyi maradna programozásra. A vizsgálatot az 5-12. évfolyamra végzem el.

Öt[1] szervezet helyi informatika tanterv ajánlását vizsgáltam meg, annak tükrében, hogy az óraszám hány százalékát szánja a „Problémamegoldás informatikai eszközökkel és módszerekkel” c. tematikai egységre. A „pesszimista” számításokat vettem alapul, tehát azokat a tanterveket vettem bele a vizsgálatba, amelyek a legkevesebb óraszámot szánták informatikára. A számszerű adatokat az alábbi táblázat mutatja.

Név	Óraszám	Informatika óraszám	Százalékos arány
AK	42	212	20%
Mozaik	59	250	24%
KPSZT	38	180	21%
JOS	34	252	13%
NTK	38	180	21%

Az adatokból látszik, hogy a KPSZT-n és az NTK-n kívül mindenki felhasznált órákat a szabadon tervezhető órakeretből. Elmondható tehát, hogy fenti tanterv javaslatok alapján

[1] Apáczai Kiadó (AK), Mozaik Kiadó (Mozaik), Katolikus Pedagógiai Szervezési és Továbbképzési Intézet (KPSZT), Nemzeti Tankönyvkiadó (NTK), Jeldik Oktatási Stúdió (JOS)

átlagosan az informatika órák ötödén tanulnak a diákok programozást az 5-12. évfolyamokban.

A „Problémamegoldás informatikai eszközökkel és módszerekkel” c. tematikai egység célja az algoritmizálási készségek fejlesztése, mert a valós élet tevékenységei, kezdve a bevásárló listától az utazásig, döntések sorozatából épül fel.

A tematikai egység elsajátítását követően, a kerettanterv szerint, 5-12. évfolyamon „*a tanulók a problémamegoldás alapvető folyamatával és elemeivel ismerkednek meg. A problémamegoldás előtt információkat gyűjtenek, és megtervezik a folyamatot. A tanulók kezdetben közösen értelmeznek kész algoritmusokat. Eleinte tanári segítséggel, majd egyre önállóbban készítenek egyes tevékenységeket leíró algoritmusokat és folyamatábrákat. Később megismerkednek az algoritmizálás elméleti módszereivel, a szekvenciális és vezérléselvű programok alapvető funkcióival, majd az elméleti megalapozást követően a gyakorlatban készítenek és tesztelnek számítógépes programokat.*” [1] Továbbá a bonyolultabb, összetettebb problémák megoldása során megtapasztalják, hogy egy ilyen feladat akkor oldható meg hatékonyan, ha több kisebb részre bontják, és a feladat megoldása közben csoportokban dolgoznak. A tanulók a problémákhoz algoritmusokat készítenek, az algoritmusokat programozási nyelven kódolják, a kódolás során megismerik a program működését, alkalmazzák a megismert utasításokat. Az alulról felfelé építkezés és a lépésenkénti finomítás elve alapján a tanulók több oldalról megközelíthetik a problémát, feltárják a probléma szerkezetét, értelmezik az adatok közötti összefüggéseket, a strukturált megoldás érdekében eljárásokat készítenek.

Az alábbi felsorolásban a kerettanterv fent említett tematikai egységének azon elemeit tűntetem fel, amelyek elsajátításához fel lehet használni az AV-t.

- **5-6. évfolyam**
    - **3.1. A problémamegoldáshoz szükséges módszerek és eszközök kiválasztása**
        - *Az algoritmus informatikai fogalmának megismerése*
            - A megoldás lépéseinek szöveges, rajzos megfogalmazása, értelmezése.
            - Folyamatábrák készítése.
        - *Problémák megoldása önállóan, illetve irányított csoportmunkában*

- A problémamegoldás különböző fázisaiban az informatikai eszközök és módszerek alkalmazási lehetőségeinek tanulmányozása.

▪ *A robotika alapjainak megismerése*

- Algoritmusok megvalósítására alkalmas programok használata.
- A folyamatos beavatkozást, vezérlést igénylő problémák megoldási módjának megismerése. Például a „teknőc" utasításokkal történő irányítása.

○ **3.2. Algoritmizálás és adatmodellezés**

▪ *Adott feladat megoldásához tartozó algoritmusok megfogalmazása, megvalósítása számítógépen*

- Egyszerűbb feladatok megoldási algoritmusának megvalósítása Logo vagy más automata elvű fejlesztői rendszer segítségével.
- Egyszerű programok írása közösen.

▪ *Feladatok megoldása egyszerű, automata elvű fejlesztőrendszerrel*

- Az algoritmizálási készségek fejlesztésére alkalmas szoftverek tanulmányozása.
- Problémamegoldás folyamatának értelmezése.
- Grafika készítése teknőccel.

○ *A tanuló a problémamegoldás informatikai eszközökkel és módszerekkel témakör végére*

▪ ismerje a problémamegoldás alapvető lépéseit;

▪ képes legyen önállóan vagy segítséggel algoritmust készíteni;

▪ tudjon egyszerű programot készíteni;

- 7-8. évfolyam

○ **3.1. A problémamegoldáshoz szükséges módszerek és eszközök kiválasztása**

▪ *A problémák megoldásához szükséges eszközök és módszerek megismerése*

- Az algoritmusleírás eszközeinek mélyebb elsajátítása (pl. folyamatábra elemeinek bővítése).
- Egyszerű algoritmusok leírása algoritmusleíró nyelven.

- A feladatmegoldást segítő lehetőségek megismerése.

▪ *A robotika alapjainak megismerése, egyszerű vezérlési problémák megoldása*

- Alakzatok rajzolása, vagy egyszerű vezérléses játék készítése valamely fejlesztői környezetben.
- A paraméterértékek változtatása, a változtatások hatásának tanulmányozása.

○ **3.2. Algoritmizálás és adatmodellezés**

▪ *Adott feladat megoldásához algoritmuselemek, algoritmusok tervezése, végrehajtása*

- Algoritmus kódolása a számítógép számára egyszerű programozási nyelven.
- Összetett algoritmusok készítése az alulról felfelé építkezés és a lépésenkénti finomítás elve alapján.

▪ *A problémamegoldáshoz szükséges adatok és az eredmény kapcsolata*

- A bemenő adatok, a kimenő adatok és a változók értékeinek megadása, a bemenő adat és eredmény kapcsolatának megfigyelése.

○ *A tanuló a problémamegoldás informatikai eszközökkel és módszerekkel témakör végére*

▪ lássa át a problémamegoldás folyamatát;
▪ ismerje és használja az algoritmusleíró eszközöket;
▪ ismerje egy programozási nyelv alapszintű utasításait;
▪ tudjon kódolni algoritmusokat;
▪ tudjon egyszerű vezérlési feladatokat megoldani fejlesztői környezetben;

- 9-10. évfolyam

○ **3.2. Algoritmizálás és adatmodellezés**

▪ *Adott feladat megoldásához tartozó algoritmusok megfogalmazása, megvalósítása számítógépen, a feladat megoldásához algoritmuselemek, algoritmusok tervezése, végrehajtása, elemzése*

- Tantárgyi problémák megoldási algoritmusainak tanulmányozása.
- Algoritmusok alkotása különböző tervezési eljárások segítségével, az alulról felfelé építkezés és a lépésenkénti finomítás elvei. Algoritmusok megvalósítása.
- Néhány típusalgoritmus vizsgálata.

▪ *Elemi és összetett adatok megkülönböztetése, kezelése, használata. Adatmodellezés, egyszerű modellek megismerése*

- Különböző adattípusok használata a modellalkotás során.

o *A tanuló a problémamegoldás informatikai eszközökkel és módszerekkel témakör végére*

▪ tudjon algoritmusokat készíteni,

▪ legyen képes a probléma megoldásához szükséges eszközöket kiválasztani;

▪ legyen képes tantárgyi problémák megoldásának tervezésére és megvalósítására;

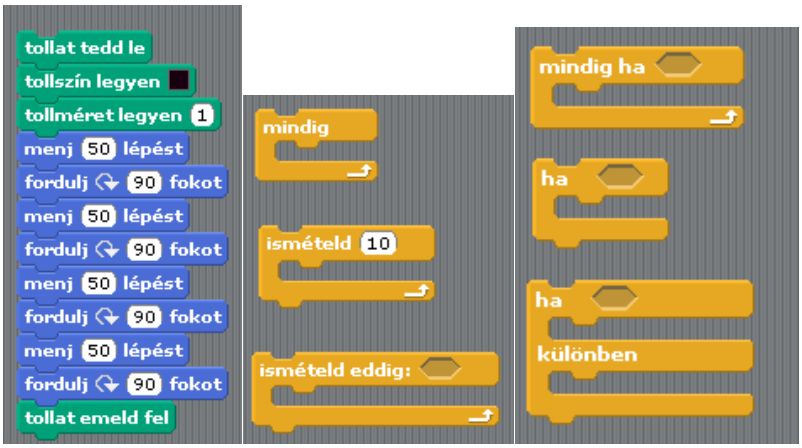
- 11-12. évfolyam = 9-10. évfolyam
  - Ebben a korosztályban az informatikai eszközök használata a többi témakör alkalmazása közben valósul meg.

3.1 Alkalmazási példák 5-12. évfolyamon, a kerettantervre alapozva

5-6. évfolyamban az algoritmus animálása segítséget nyújthat a megoldás lépéseinek szöveges, rajzos megfogalmazásában, értelmezésében, illetve a folyamatábrák késztésében, olyan módon, hogy szemben a folyamatábrával, amely csak az egyes lépéseket tartalmazza, az algoritmus animálása folyamatában mutatja meg az algoritmust, tehát a *vizualitás* mellé az *időbeliség*et is reprezentálni lehet. Ezzel segíteni lehet azt, hogy a tanuló fejében a megoldandó probléma egy folyamatként jelenjen meg, és amely folyamat lépésenkénti reprezentációja lesz a szöveges vagy rajzos megfogalmazás, illetve a folyamatábra.

A robotika alapjainak megismerését automata-elvű rendszerrel képzeli el a kerettanterv, erre utal például a „teknőc” utasításokkal történő irányítása. A „teknőc” kifejezés egyértelműen az Imagine Logo környezetre utal, de más eszközt is lehet használni.

Véleményem szerint sarkalatos kérdés, hogy milyen eszközt választunk, mert nagy valószínűséggel ekkor fog először a programozás folyamatával találkozni a tanuló, és olyan rendszerre van szükség, amely az eredményessége mellett sikerélményt is nyújt. A Scratch nevű fejlesztői rendszer [I14] előnye a Logo környezethez képest, hogy nem történik hagyományos értelemben kódolás, hanem a rendszer által felkínált „építőkockákból" kell összerakosgatni az algoritmust (7. ábra). Ilyen módon minden összeállítható program helyes lesz szintaktikailag, és az algoritmus, a gondolkodás helyességére lehet fektetni a hangsúlyt.



7. ábra. *A Scratch építőkockái: szekvencia, ciklus, elágazás*

A Scratch és a Logo egyértelmű előnye az, hogy a feladat megoldása közben vizuális reprezentációt kapok a program működéséről (a program kimenete lehet maga az animáció). A fenti ábra bal oldali programjának látható eredménye egy négyzet lesz. Az alakzatok rajzolása során könnyen meg lehet tanítani a ciklus fogalmát, illetve a Scratch szereplői által eljátszott történeteken keresztül az elágazás fogalmát is.

7-8. évfolyamban az algoritmus leírásának elmélyítéséhez is segítséget nyújt az AV. Előre elkészített animációk alapján lehet az adott folyamat algoritmusát leírni, aztán együtt figyelni a kész algoritmust és az animációt. Utóbbi esetben létrejön a multimédia hatás, és az animációt lépésenként futtatva az is kiderül, hogy a tanuló helyesen írta-e le az algoritmust.

Az egyszerűbb programok írásánál itt már jó eszköz az Imagine Logo, ugyanis ilyenkor a tanuló már tisztában van a vezérlési szerkezetek működésével, illetve az adott feladat algoritmusának leírásával, tehát be lehet hozni új elemként a kódolást, azaz hogyan lehet a számítógép „nyelvén" megvalósítani a kitűzött feladatot.

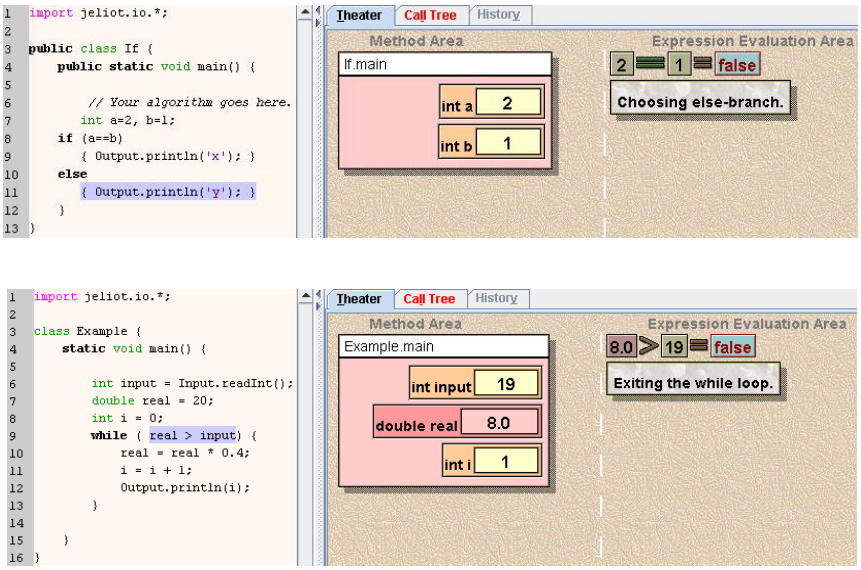
9-10. osztályban ismerkedhetnek meg a tanulók az első imperatív programozási nyelvvel, hogy tantárgyi problémákat oldjanak meg vele, illetve alkalmazzák a típusalgoritmusokat. Alapvető kérdés, hogy melyik legyen ez a nyelv. Oktatási szempontból előnyős, ha a választott nyelv könnyen megjegyezhető alapszavakból áll, és egyszerű programszerkezetet használ. Fontos megvizsgálni, hogy milyen könnyű megírni az első *értelmes* programot, tehát az első olyan programot, amelynek –a tanuló szerint is– gyakorlati haszna is van. Egy olyan nyelvre van szükség, amely közel áll az algoritmikus nyelvünkhöz, a tanuló megtanulja és megérti a programozás alapvető elemeinek helyét, szerepét és használatát. A Pascal erős (és szigorú) típusossága, átlátható és memorizálható programszerkezete „rászoktatja" a tanulót, hogy ne felejtsen el változót deklarálni, típust definiálni, hogy programírás közben mindig gondolja végig, miket és hogyan akar használni az éppen aktuális eljárás megvalósításakor. A Pascal akkurátussága jó alap a továbblépéshez az objektum orientált programozás és/vagy egy 4GL fejlesztőrendszer felé. Utóbbi, vizualitása miatt, segítheti az objektum orientált paradigma megértését. [45]

A korábban tanult környezethez képest a Pascalnak az a hátránya, hogy nincs „vizuális kimenete", ezért különösen érdemes ebben az időszakban AV eszközt használni.

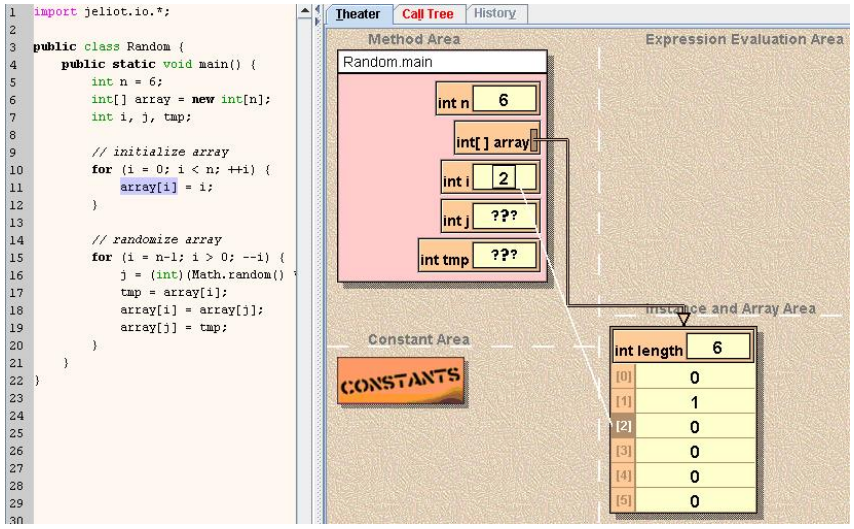
A Jeliot 3 alkalmas a vezérlési szerkezetek (elágazás, ciklus), illetve az osztályok metódusainak belső működésének implementálására, így tudja támogatni a tanár magyarázatait és az önálló tanulást is.

Nem jelentős hátrány a környezet „nyelvfüggősége", bár nyílván leginkább a Java nyelven tanulók tudják kihasználni a rendszer szolgáltatásait. A vezérlési szerkezetek, az algoritmus működésének, az osztályok metódusainak és a közöttük levő kapcsolatok megértése nem nyelvfüggő probléma.

A rendszer mindig megindokolja, miért az adott ágban halad tovább a program futása elágazás esetén, miért maradunk a ciklusban vagy lépünk bele, illetve ki belőle (8. ábra).



8. ábra. *Vezérlési szerkezetek működésének reprezentálása Jeliotban*



9. ábra. *Tömb megjelenítése Jeliotban*

Gyakran használt adatszerkezet a tömb, ennek belső állapotát is képes megjeleníteni a program (9. ábra).

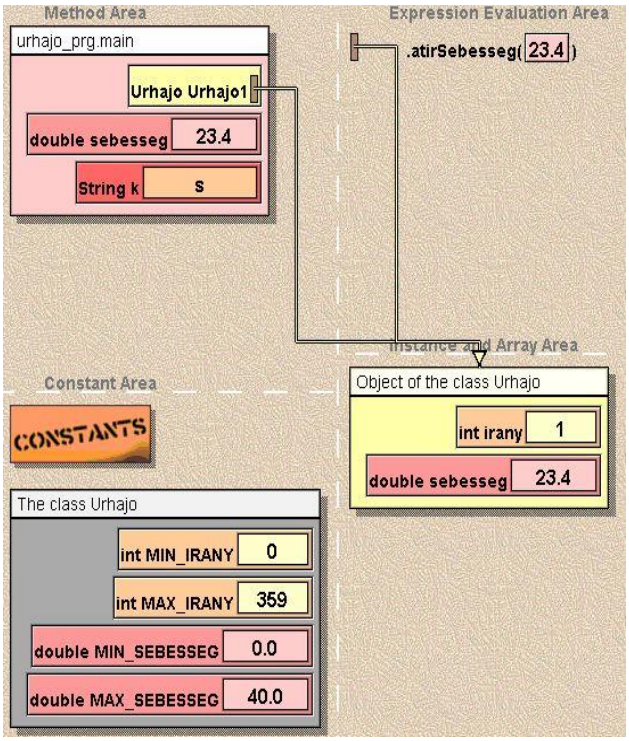
Tekintsünk egy objektum orientált példát, az Űrhajó és a Neves Űrhajó osztály megvalósítását:

Egy Űrhajónak két tulajdonsága van: a sebessége és az iránya. Ismerjük a minimum és maximumsebességet, illetve iránynak sem lehet megadni 359 foknál nagyobb értéket. Négy metódust kell megvalósítani: a sebesség megváltoztatását, a balra fordulást, a jobbra fordulást és a hajó állapotának kiírását.

A Neves Űrhajó az Űrhajó osztályból öröklődik. Annyiban különbözik az ősétől, hogy már nevet is lehet adni az egyedeinek. Egyetlen új tulajdonságunk lesz: a hajó neve, és emiatt felül kell definiálni a hajó állapotát kiíró metódust. A többi tulajdonság és metódus az ősosztályból származik.

Ha nem volna vizualizációt megvalósító eszközünk, akkor egy szövegszerkesztő program és a parancssor állna csak a rendelkezésünkre.

A Jeliot 3 alkalmas arra, hogy futási időben megmutassa az objektum aktuális állapotát (10. ábra).

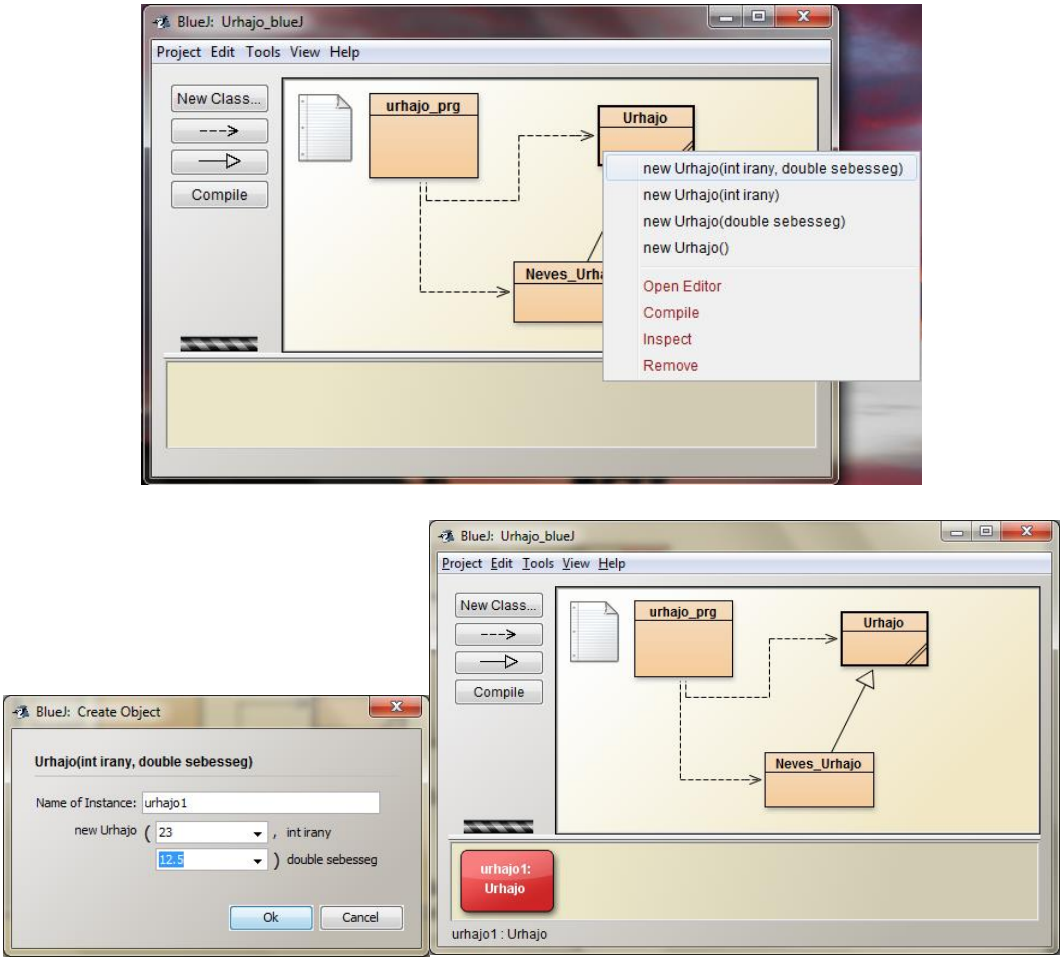


10. ábra. *Az Űrhajó osztály objektumának állapota Jeliotban*

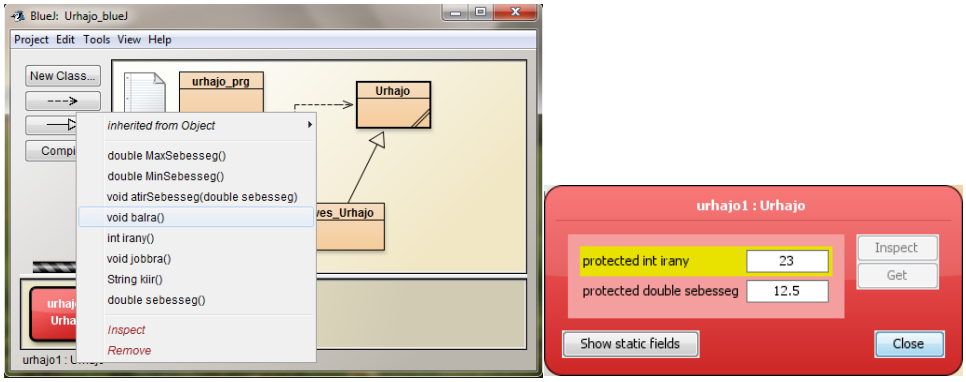
A BlueJ ugyanezt egy kicsit másként valósítja meg. Nem az animáció közben van lehetőségünk az objektum állapotát vizsgálnunk, hanem miután az osztály forrásfájlja le lett fordítva, a benne levő konstruktorokat és egyéb metódusokat le tudjuk futtatni, és ezek eredményeit meg tudjuk tekinteni (11-12. ábrát).

Ilyen módon a tanár meg tudja mutatni a hallgatóknak a végeredményt, és a grafikus reprezentációval kézzelfoghatóbbá válik az objektum, ezért annak működése könnyebben

elmagyarázható és megérthető, melynek köszönhetően a megvalósítás (a kódolás) könnyebb lesz.



11. ábra. *BlueJ: objektum létrehozása*

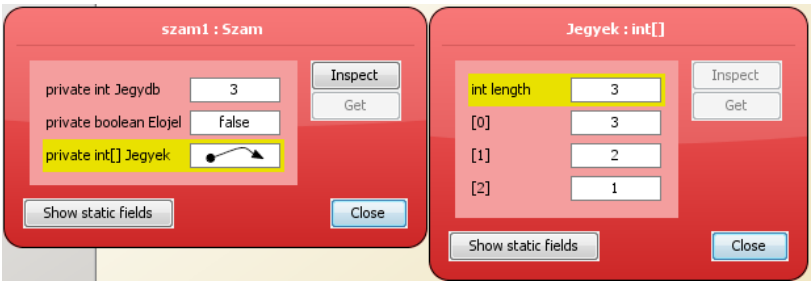


12. ábra. *BlueJ: Objektum metódusa és állapota*

A következő példa már bonyolultabb feladat megoldását tűzte ki célul: létre akarunk hozni egy új Szám nevű osztályát az egész számoknak, amellyel akár több ezer jegyű számokat is meg tudunk jeleníteni, illetve az egészekre érvényes műveleteket is végre tudjuk hajtani rajtuk. Egyértelmű, hogy a számítógép a standard típusaival erre már nem lenne képes.

A feladatot tetszőleges számrendszerre meg lehet oldani, az egyszerűség kedvéért azonban a példákat 10-es számrendszerben fogom közölni.

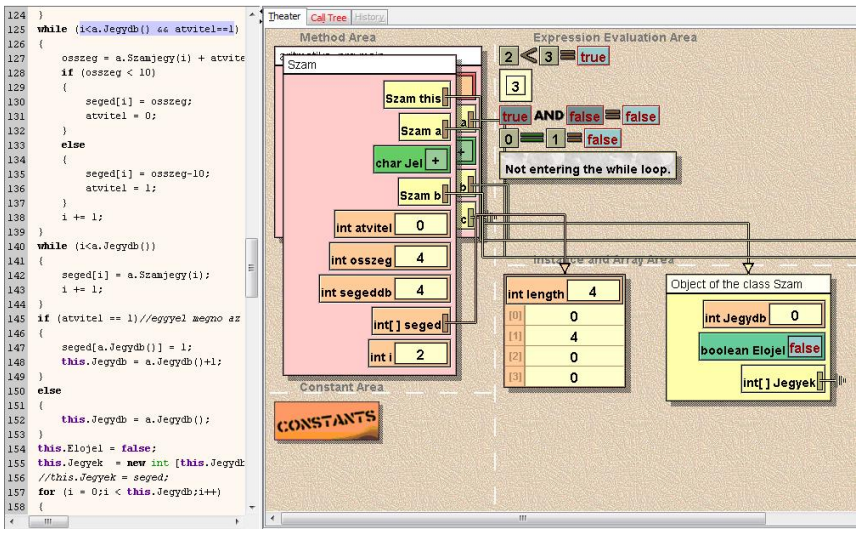
A nagypontosságú egész szám számjegyeit tömbben tároljuk. A Szám osztály tulajdonságai között fogjuk számon tartani továbbá a számjegyek számát, az előjelet és a számrendszer alapját. A megvalósítandó műveletek (metódusok): Összeadás, kivonás, szorzás, maradékos osztás, egész osztás, beolvasás, kiíratás, illetve az = , ≠ , < , > , ≤ , ≥ relációk. A 13. ábra mutatja az objektum megvalósítását.



13. ábra. *BlueJ: A 123-as szám megvalósítása a Szám osztállyal*

Habár bonyolultabb ez a példa ez előzőnél, mégsem igényel speciális ismereteket, mély gondolatokat, csillogó ötleteket, „csak” szigorú fegyelmet. A hallgatóknak mindössze fel kell idézniük, hogy miképp működnek azok a matematikai műveletek, amit már általános iskola második osztálya óta használnak, és kellő akkurátussággal kell rendezniük az ismert apró lépéseket egy összefüggő tevékenységsorrá. Pl. az összeadás és a kivonás művelete megegyezik azzal a „kézi” módszerrel, ahogyan papíron használjuk ezt a két műveletet.

A Szám osztály működésének bemutatásánál érdemes animációval illusztrálni a műveletek algoritmusát Jeliot 3-mal, így a tanár nemcsak állapotokat tud megmutatni, hanem magát a folyamatot is (erre már nem képes a BlueJ).



14. ábra. *Jeliot 3: Ciklus bennmaradási feltételének kiértékelése*

A fenti ábra az összeadás művelet algoritmusának egy pillanatnyi állapotát mutatja. Első látásra nagyon kuszának tűnhet, mert, ez csak egy állapotnak a „fényképe”. Olyan, mint ha egy mozifilm egyetlen képkockáját tekintenénk csak meg: ez lehet akár életlen, a film lejátszásakor csöppet nem zavaró. Az animáció nézése során ugyanez a helyzet: egy-egy kiragadott állapot lehet kusza, a nézési folyamat közben azonban tökéletesen érthető: az animáció közben az a többlet információ segíti a nézőt, hogy milyen folyamaton, mely állapotokon, lépéseken keresztül jutottunk el az adott állapotba. Látható, hogy a Jeliot 3 minden elágazásnál és ciklusnál kiértékeli a feltételt, és ilyen módon megindokolja, miért arra megy tovább a program, amerre menni fog. A fenti lépést (14. ábra) 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.

Mit lát a tanuló? Látja a programkódot és a program belső állapotát, tehát van szöveges és képi információja, ilyen módon teljesül a Multimédia-hatás. A kód elrejthető, ha zavart okoz, pl. olyan esetben, ha a tanulók nem Java-t, hanem egy másik programnyelvet használnak. Ebben az esetben a saját kódját tudja olvasni az animáció mellett, feltéve, hogy az szinkronban van az animáció menetével.

Felmerülhet kérdésként, hogy mennyire zavaró az a tény, hogy a Jeliot 3 környezetet, szemmel láthatóan Java programozási nyelvvel való oktatásra tervezték. A környezettel történt korábbi vizsgálatok [4] alkalmával Turbo Pascalt használtak a tanulók, és az idézett

tanulmány szerint az alapelemek tekintetkében (pl. értékadás, elágazások, ciklusok) a szintaktikus különbségek nem nagyok. Mindössze a kiíró műveletek megértése és használata okozott problémát, amit a tanulmány szerzői a két nyelv közötti különbségre vezettek vissza. A többi művelettel (értékadás), vezérlési szerkezettel (elágazások, ciklusok) kapcsolatban nem okozott problémát a programozási nyelvek közötti különbség.

3.2 Javasolt óraszámok

Tekintsünk egy olyan példát, ahol az adott közoktatási intézmény megemeli az informatika órák számát a szabad órakeretből. A Mozaik Kiadó emelt óraszámú kerettantervi ajánlása a következő óraszámokat javasolja, amely gyakorlatilag megfelel a korábban említett NFM ajánlásnak:

Óraterv a kerettantervekhez – 5-12. évfolyam, gimnázium											
Tantárgyak	**5. évf.**	**6. évf.**	**7. évf.**	**8. évf.**	**9. évf.**	**10. évf.**	**11. évf.**	**12. évf.**	**Össz./hét**	**Össz.**	**%**
Informatika	1	2	2	2	2	2	2	2	15	**540**	**5,79%**

Az általam leginkább kidolgozottnak tartott Mozaik Kiadó ajánlása alapján a következő témakörönkénti óraelosztást gondolom kivitelezhetőnek:

Évfolyam	Témakör	Óraszám	AV használatának módja
5.	Az algoritmizálási készségek fejlesztésére alkalmas szoftverek tanulmányozása.	1	Scratch bemutatása
	Problémamegoldás folyamatának értelmezése	2	Kész mintafeladatok bemutatása Scratch-csel, megoldási lehetőségek átbeszélése
	Grafika készítése teknőccel	3	Scratch használata
6.	A megoldás lépéseinek szöveges, rajzos megfogalmazása, értelmezése, folyamatábrák készítése	2	Kész mintafeladatok bemutatása Scratch-csel, ez alapján folyamatábra készítése

Évfolyam	Témakör	Óraszám	AV használatának módja
	A problémamegoldás különböző fázisaiban az informatikai eszközök és módszerek alkalmazási lehetőségeinek tanulmányozása, Algoritmusok megvalósítására alkalmas programok használata, A folyamatos beavatkozást, vezérlést igénylő problémák megoldási módjának megismerése. Például a „teknőc” utasításokkal történő irányítása	2	Scratch önálló használata
	Egyszerűbb feladatok megoldási algoritmusának megvalósítása Logo vagy más automata elvű fejlesztői rendszer segítségével	4	Scratch önálló használata
	Egyszerű programok írása közösen	4	Scratch önálló használata
7.	Az algoritmusleírás eszközeinek mélyebb elsajátítása (pl. folyamatábra elemeinek bővítése).	3	Példák bemutatása Logo-val, ezek alapján algoritmus írása
	Egyszerű algoritmusok leírása algoritmusleíró nyelven	3	TRAKLA2 önálló használata, algoritmus futtatása a rendszerrel
	A feladatmegoldást segítő lehetőségek megismerése	2	
	Alakzatok rajzolása, vagy egyszerű vezérléses játék készítése valamely fejlesztői környezetben	5	Logo önálló használata, animáció értelmezése
	A paraméterértékek változtatása, a változtatások hatásának tanulmányozása	3	Logo önálló használata, animáció értelmezése
8.	Algoritmus kódolása a számítógép számára egyszerű programozási nyelven	4	Logo önálló használata
	Összetett algoritmusok készítése az alulról felfelé építkezés és a lépésenkénti finomítás elve alapján	4	Logo, TRAKLA2 és algoritmusleíró eszközök önálló használata

Évfolyam	Témakör	Óraszám	AV használatának módja
	A bemenő adatok, a kimenő adatok és a változók értékeinek megadása, a bemenő adat és eredmény kapcsolatának megfigyelése	2	Logo önálló használata, animáció értelmezése
10.	Tantárgyi problémák megoldási algoritmusainak tanulmányozása	2	Jeliot önálló használata. problémák átbeszélése az animáció alapján
	Algoritmusok alkotása különböző tervezési eljárások segítségével, az alulról felfelé építkezés és a lépésenkénti finomítás elvei. Algoritmusok megvalósítása	2	Jeliot, TRAKLA2 önálló használata, bemutatás, ez alapján algoritmus írása és kódolása
	Néhány típusalgoritmus vizsgálata	2	Jeliot, TRAKLA2 önálló használata, bemutatás, ez alapján algoritmus írása és kódolása
11.	Tantárgyi problémák megoldási algoritmusainak tanulmányozása	2	Jeliot önálló használata. problémák átbeszélése az animáció alapján
	Algoritmusok alkotása különböző tervezési eljárások segítségével, az alulról felfelé építkezés és a lépésenkénti finomítás elvei. Algoritmusok megvalósítása	2	Jeliot, TRAKLA2 önálló használata, bemutatás, ez alapján algoritmus írása és kódolása
	Néhány típusalgoritmus vizsgálata	7	Jeliot, TRAKLA2 önálló használata, bemutatás, ez alapján algoritmus írása és kódolása
	Különböző adattípusok használata a modellalkotás során	1	Jeliot önálló használata összetett adattípusok vizualizációjára

A fenti felosztásnál annyiban eltértem a Mozaik Kiadó által írttól, hogy a folytonosság fenntartása végett 7. évfolyam tanulói számára is legyen lehetőség a programozás AV-vel való tanulására, ugyanis sokkal nehezebb lenne egy év kihagyás után újra átismételni és „belerázódni” a programozásba, tekintettel arra, hogy az informatika egyik legnehezebben megérthető területéről van szó. Igaz, ez a folytonosság megszakad 8. évfolyam után, ennek ellenére hasznosnak gondolom, mert ha, tegyük fel, egy hallgató általános iskolában nem vagy kevés informatikát tanult, akkor a 9. osztály jó alkalom a számára, hogy elmélyedjen az

informatika elméletében és az alkalmazói programok kezelésében, majd utána már kellő elméleti háttérrel és készséggel fog rendelkezni, hogy megismerkedjen a programozással.

A fenti óraszámok esetén van arra idő, hogy a tanulók megismerjék az idézett AV eszközök használatát, és hogy 10-11. évfolyamon akár már maguk tudjanak Jeliot-tal animációt készíteni, mert – ahogy az előző fejezetben láthattuk – minél aktívabban használja a tanuló az AV eszközt, annál nagyobb tanulási haszon. Így igazolva lett a Hipotézis 1. (Az AV beilleszthető a közoktatási informatikába).

4 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 csoport vett részt a kísérleti csoportban és 5 csoport vett részt a kontroll csoportban, összesen 153 fővel. A létszámokat az alábbi táblázat mutatja:

	Kísérleti csoportok	Kontroll csoportok
	16	18
	15	16
	16	14
	16	15
	15	12
Összesen:	78	75

A kísérleti és a kontroll csoport azonos tanmenetet használt, a különbség a tanítás eszközében volt. A kísérleti csoport oktatói és hallgatói a kurzus alatt a Jeliot 3 AV rendszert használták, a kurzuson használt fejlesztői környezet mellett, tehát a gyakorlat idejében több eszközt kellett megismerniük a hallgatóknak a kísérleti csoportban, mint a kontroll csoportban, tehát valójában kevesebb ideje volt az oktatónak és a hallgatónak egyaránt a kísérleti csoportokban.

Azért a Jeliot 3-ra esett a választás, mert ez a rendszer képes a kódot valós időben animálni, másrészt, a C++ és a Pascal a leggyakrabban használt nyelv a közoktatásban [I9], ugyanis ezek a nyelvek szerepelnek a Nemzetközi Informatikai Diákolimpia programozási versenyén használható nyelvek között, és a Diákolimpia programozási nyelveit a közoktatás által használt nyelvek közül választják. A Jeliot 3 követi ezeknek a nyelveknek a konvencióit, így alkalmas eszköz azok számára, akik a fent említett nyelvek valamelyikén ismerkedik a programozással. Sőt, a Jeliot 3 közel áll a C++-hoz, amely a kurzuson alkalmazott programozási nyelv.

Felmerülhet kérdésként, hogy miért nem a közoktatásban végeztem el ezt a vizsgálatot, hiszen az AV eszközök a programozás kezdeti lépéseihez nyújtanak segítséget. Az előző fejezet elején kitértem arra, hogy általában nagyon kevés óraszámot kap az informatika, és így a programozás is a helyi tantervekben, másrészt a későbbiekben ismertetett vizsgálat

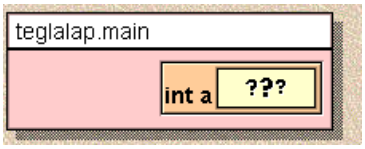
eredménye arra enged következtetni, hogy a közoktatási intézmények jelentős részénél nem tanítanak programozást, annak ellenére, hogy a téma része a hatályos kerettantervnek.

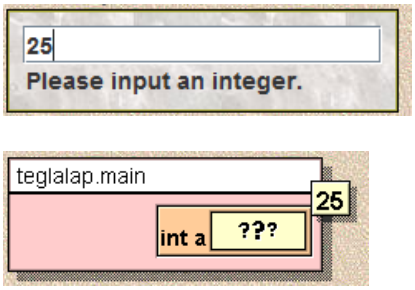
4.1 Programozási alapismeretek kurzus tematikájának bemutatása

A kurzus alapozó tantárgy első évfolyamon, a hallgatók fele ekkor találkozik először programozással, ezért a tematika is az alapoktól indul. A kurzus célja, hogy a hallgatókat, akiknek később a programozás lesz a munkájuk, egy szintre hozza, illetve az elvárt középiskolai programozási tudás rendszerezése, és némileg formalizálása is, valamint a további programozás tárgyak fogalmi és rutinbeli előkészítése; tehát azok, akik nem, keveset vagy régen tanultak programozást, felzárkóztassa azokhoz, akik már megszerezték középiskolában a programozás alapjaihoz szükséges tudást és gyakorlatot. A hallgatók a Code::Blocks [I3] fejlesztői környezetet használják, és C++ nyelven kódolják az órai feladatokat.

4.1.1 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, kvázi definiáltam. A Jeliot rendszer segítségével ez a sorrendiség vizuálisan is megmutatható. Példának nézzünk egy egyszerű programot, amely a téglalap területét számolja ki. Az alábbi táblázat megmutatja az adott kifejezés vizuális reprezentációját is. A „/” jel után a Java parancsot írtam, ahogy a Jeliotban megjelenik.

Programkód	Vizuális reprezentáció
int a;	

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

A képeken jól látszik, hogy a változó először létrejön, majd a felhasználó által megadott bemeneti adat „belemozog” a változóba. Ebből az animációból azt is megérti a hallgató, hogy a változónak mindig egyértelmű értéket kell adni, hogy biztonsággal hivatkozzunk rá.

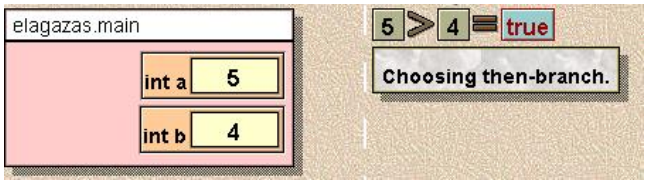
4.1.2 Vezérlési szerkezetek

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 feldolgozása során az elágazás működésével ismerkednek meg a hallgatók. 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).

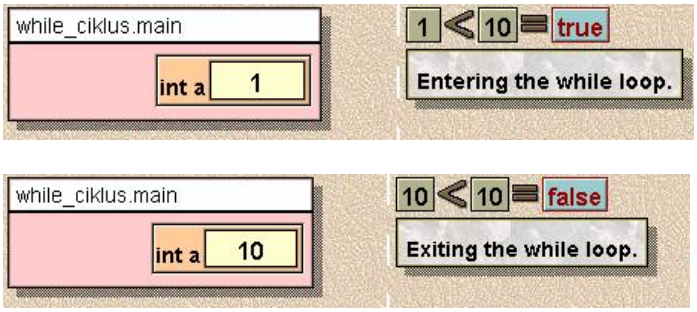
Programkód	Vizuális reprezentáció

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 (sárga kiemelés a kódban), majd a választ is, hogy a logikai vizsgálat eredménye miatt melyik ágon folytatódik a program futása (zöld kiemelés a kódban). Ezért a hallgató mindig kielégítő választ kap arra a kérdésre, hogy miért azon az ágon halad tovább a program futása.

A harmadik témakör a ciklusokról szól. A ciklus fogalmának és működésének megértése nehezebb, mint az elágazásé, mert itt már egy ismétlődő folyamatról van szó és nem csak arról, hogy egy adott feltétel függvényében merre halad a végrehajtás. 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.

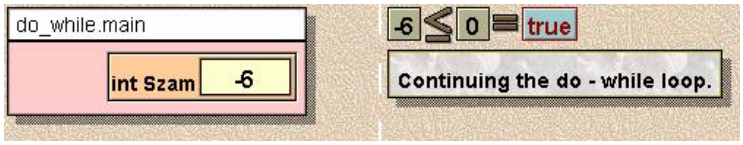
Programkód	Vizuális reprezentáció

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

Az elöl tesztelős ciklus belépési feltétele a ciklusmag lefutása előtt ellenőrzésre kerül, ennek kiértékelését és következményét kiírja a program. Az AV program magyarázatából a hallgató megkapja a választ arra a kérdésre, hogy miért lép be a ciklusba vagy éppen ki a ciklusból a program.

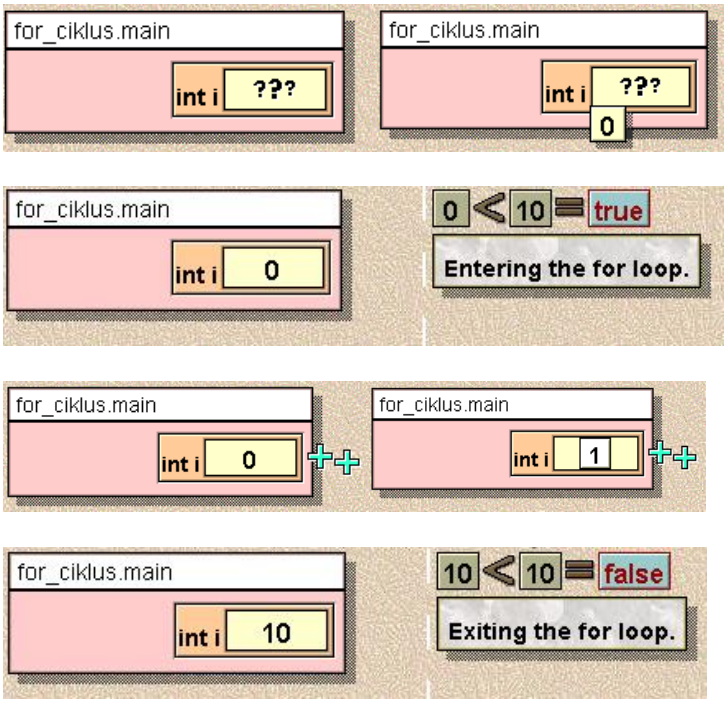
A hátul tesztelős ciklusnál a Jeliot helyesen a ciklus folytatását írja ki a ciklus feltétel igaznak bizonyulásakor, így markánsan meg lehet különbözetni a működését és a szerepét az elöl tesztelős ciklusétól.

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, ird be ujra!"; / Output.println("Hibas szam, ird be ujra!");
}
}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.

Programkód	Vizuális reprezentáció

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

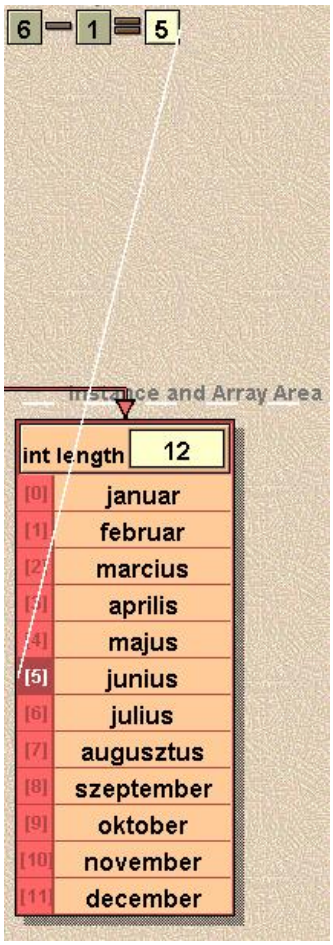
Az animációból látható, hogy első lépésben az i változó fog létrejönni, így kiderül, hogy miért szerepel az „int” az i előtt (deklaráció, sárga kiemelés), majd értéket kap (világoszöld kiemelés), aztán megtörténik a ciklus belépési feltételének ellenőrzése és kiértékelése (világoskék kiemelés), lefut a ciklusmag és végül a ciklusváltozó eggyel megnő (lila kiemelés).

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

4.1.3 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.



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

4.1.4 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 [41, 42]:

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: $Hossz(X)=\mathbf{N}$

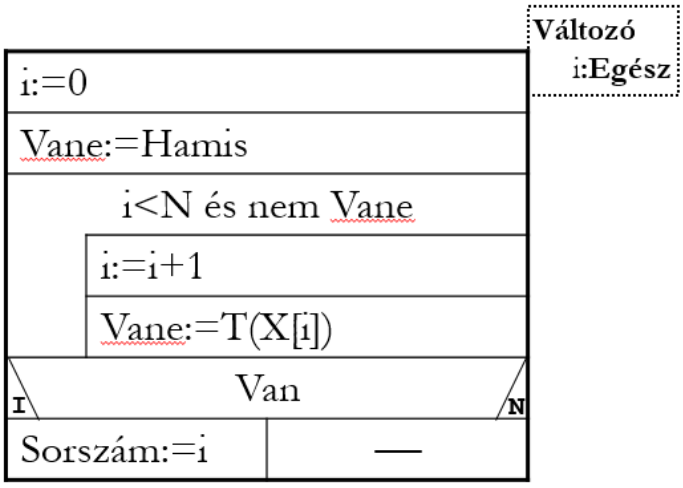
Utófeltétel: $Vane \equiv \exists i \in [1..N]: T(X_i) \wedge Vane \Rightarrow Sorszám \in [1..N] \wedge T(X_{Sorszám})$

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.)

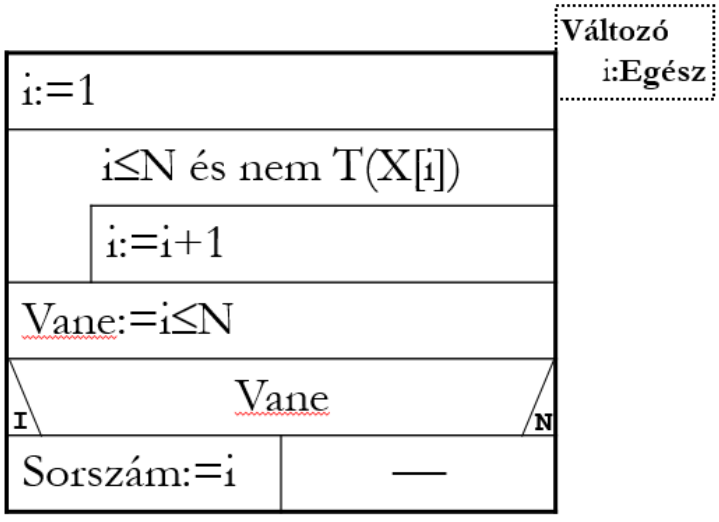
A fent specifikált lineáris keresés tétele egy tömb adatszerkezetben keresi a T tulajdonságú elem helyét, már ha van ilyen.

Lent látható algoritmus megoldja a fenti absztrakt feladatot.



16. ábra. *Lineáris keresés megoldása struktogrammal*

Tekintsünk egy másik, a fenti tételt megoldó algoritmust:

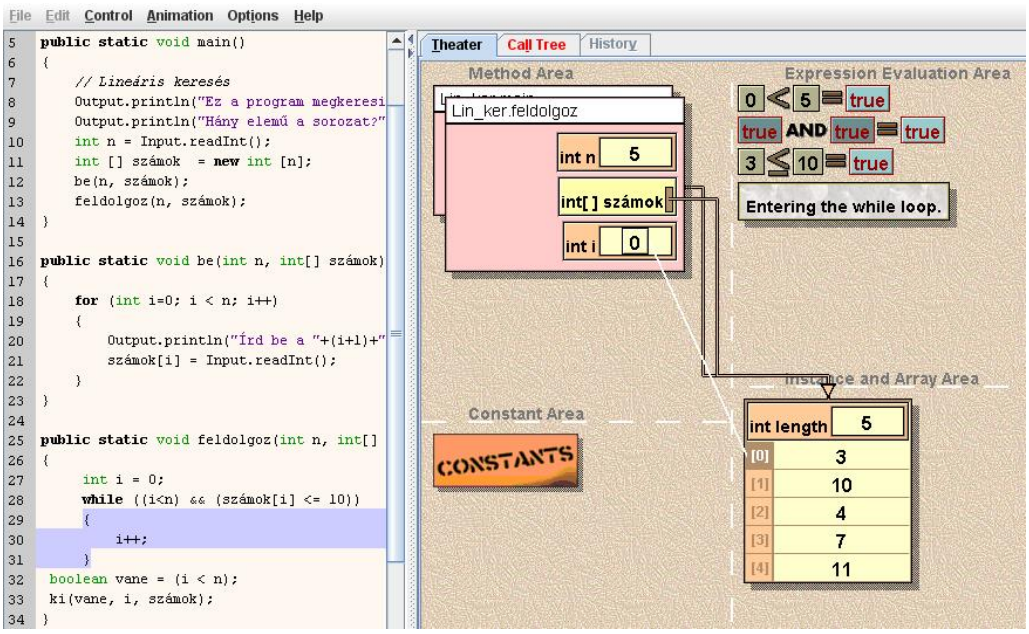


17. ábra. *Lineáris keresés egy másik megvalósítása*

Miért érdemes egy adott programozási tételhez több absztrakt algoritmus változatot készíteni? 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 a program 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:



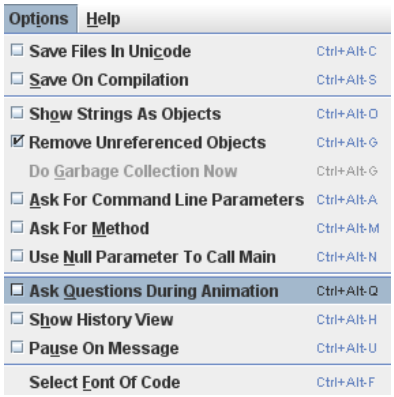
18. ábra. *Ciklus bennmaradási feltételének kiértékelése*

A fenti 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!”

Mint láthattuk korábban, Jeliot 3 minden elágazásnál és ciklusnál kiértékeli a feltételt, és ilyen módon megindokolja, miért arrafelé halad a végrehajtás, amerre halad. A fenti lépést (18. ábra) 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.

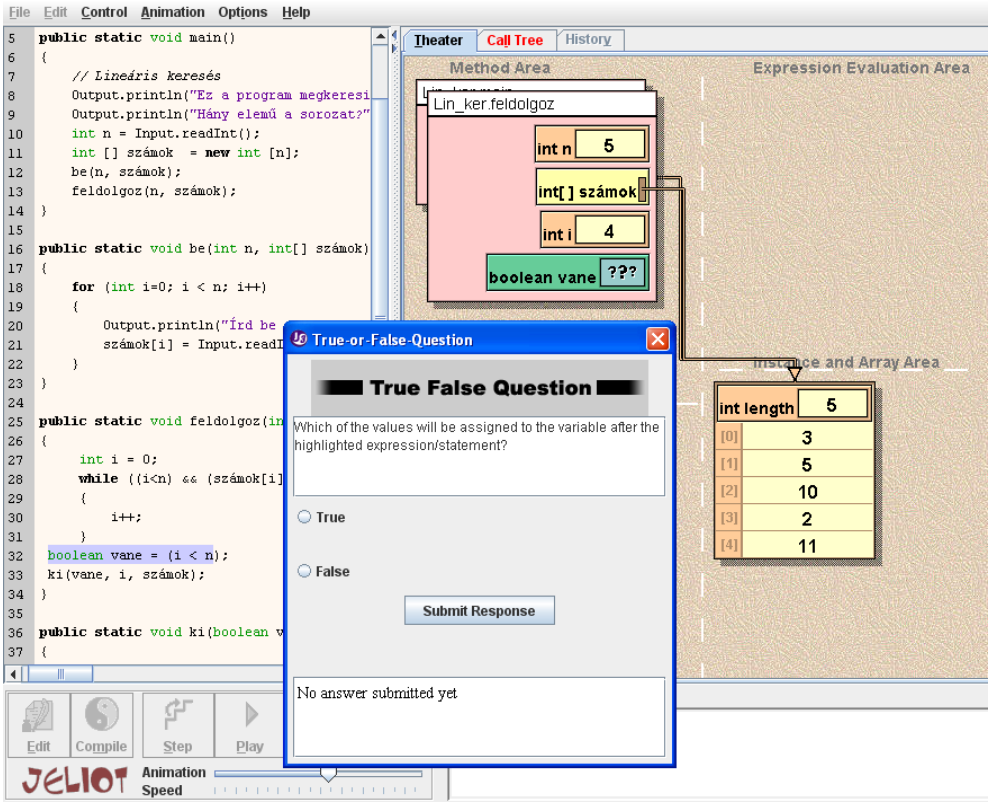
A program belső állapota mindent elmond: éppen melyik tömbelemet vizsgáljuk, mik a helyi változók értékei, illetve hogy állunk a ciklus belépési feltételének kiértékelésével. A C++ nyelvnél ez a különbözőség csak kismértékű, gyakorlatilag csak az input-output műveleteknél van különbség, az adatszerkezetek 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 20. ábra mutatja.



19. ábra. *Ide pipát téve kérdéseket kapunk animáció közben*

A kérdések „egysíkúak”, sajnos: mindig csak a változók/kifejezések értékére, értékváltozására vonatkoznak. Azonban, algoritmusunkat tekintve, az egyik feladat az algoritmus megértésének kulcskérdését feszegeti (20. ábra), végül is kijelenthetjük kitűnő a környezet otthoni, egyéni tanulásra, illetve hibakeresésre is.



20. ábra. *A belső állapotot leolvasva helyesen tudunk felelni a kérdésre*

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

4.1.5 A vizsgálatnál alkalmazott tematika

A vizsgálatnál a következő tematika szerint tanult a kísérleti és a kontroll csoport:

- 1. gyakorlat
  - Bemutatkozás
  - Code::Blocks letöltési és telepítési tudnivalók
  - Első tapasztalatok szerzése a Code::Blocks környezetről
  - Forrásfájl, futtatható kód
  - Egyszerű program megírása (Téglalap területének kiszámítása)
    - Célok:
      - Annak megtanítása, hogy a hallgatók megértsék azt, hogy akkor és csak akkor tudok valamilyen adatot (változót) felhasználni (azaz értéket adni, számolni vele, ha azt először deklaráltam, kvázi definiáltam.
      - int, cin és cout használatának megtanulása
  - Jeliot megismerése, a futtatható kiadás letöltése

        - A Jeliot mintaprogramba a lényegi C++ kód beírása, majd a program futtatása animációval, amely megmutatja ezt a sorrendiséget: először létrejön a változó, majd azután kap értéket, azután lehet felhasználni.
        - Lehetőleg a hallgatók állapítsák meg ezt a sorrendiséget
    - Házi feladat: Téglatest térfogatának kiszámítása
- 2. gyakorlat
    - Alaptípusok bemutatása: int, float, double, char, string, bool
    - Vezérlési szerkezetek közül az elágazás bemutatása Jeliot-tal
        - A Jeliot animációja végig „kommentezi" a folyamatot, hogy melyik ágat választja a program és miért (lépésenkénti végrehajtás).
    - Alap feladatsor, 1-es feladat bemutatása: Gondoljon egy pozitív egész számra és olvassa be! Nem pozitív egész esetén jelezze a hibát és lépjen ki a programból. Helyes szám beolvasása után írja ki a beolvasott számot a képernyőre! (egyszerű verzió, hibakezelés nélkül) Jeliot-tal
        - Gyakorlati feladat: kódolni a programot, majd a kezdetleges hibakezelésről beszélni.
    - Logikai és, ill. logikai vagy működésének bemutatása Jeliot-tal
    - Alap feladatsor, 6-os feladat lekódolása: Mennyi pénzt költhet el szórakozásra? Elhatározza, hogy ha 5000 Ft-nál kevesebbet keres (készpénzt kap), akkor azt azonnal elkölti – ennél nagyobb összeg esetében azonban 5000Ft egész számú többszörösét a számlájára utalja és csak a maradékot költi el (maradékosztály).
    - Házi feladat: Alap feladatsor, 10-es feladat: Reggeli hőmérséklet értékelése, átszámítása Fahrenheit-re. [°F] = [°C] · 9/5 + 32 Olvassuk be a hőmérsékletet - a Földön eddig előfordult leghidegebb és legmelegebb hőmérsékletek a kódban adottak (-89° C, 58° C). Hibás beolvasás esetén jelezzük a hibát és lépjünk ki a programból. Helyes érték esetén adjuk meg és írjuk is ki, hogy fagypont alatti vagy feletti-e a hőmérséklet és számoljuk azt át Fahrenheitre!
        - Amennyiben elakadnak a diákok, nézzék át a Jeliotos fájlokat otthon
- 3. gyakorlat
    - Többirányú elágazás bemutatása, Jeliot-tal is
    - Ciklusok bemutatása, Jeliot-tal is
        - Elől tesztelős
        - Hátul tesztelős
        - Számlálós
        - Itt is érdemes lépésenként futtatni a Jeliot-tal és értelmezni az animációt
    - Szintaktikus és szemantikus ellenőrzés jelentése, gyakorlati megvalósításuk
        - Jeliotban nincsen erre szükség
    - Elágazások feladatsor, 3-as feladat: Határozzuk meg az együtthatóival megadott, ax + b = 0 formátumú elsőfokú egyenlet megoldását!
    - Elágazások feladatsor, 4-es feladat: Határozzuk meg az együtthatóival megadott, ax2 + bx + c = 0 formátumú másodfokú egyenlet megoldásait!

    - Házi feladat: Ciklusok feladatsor, 3-as feladat: Határozzuk meg két természetes szám szorzatát úgy, hogy nem használjuk a szorzás műveletét!
- 4. gyakorlat
    - Előző órai házi feladat átbeszélése, Jeliotos animációval együtt
    - Tömbök bevezetése
    - Feladat 1: Olvassunk be egy 1-12 közötti számot, és írjuk ki a hozzá tartozó hónap nevét!
        - Animáció bemutatása a közös megoldás előtt
    - Feladat 2: Olvassunk be N (>0) egészet, és adjuk meg közülük a legnagyobbat!
    - Jeliot animáció
    - Házi feladat Írjunk programot, amely megadja egy hónapról, hogy melyik évszakba tartozik!
- 5. gyakorlat
    - Ciklusok és tömbök gyakorlás
    - Ciklusok feladatsor, 6-os feladat (Euklideszi algoritmus): Határozzuk meg két pozitív egész szám legnagyobb közös osztóját!
        - Algoritmus írása, Jeliot-tal „ellenőrizve”, azaz lépésenként végrehajtva
    - Házi feladat (előző feladatra épül)
        - Olvassunk be egy „a” egész számot és n db (n > 0) egész számot! Írjuk ki azokat a számokat, amelyek a-hoz relatív prímek!
- 6. gyakorlat
    - Programozási tételek I.
        - Eldöntés, kiválasztás, keresés
    - Eldöntés tétele
        - Algoritmus és Jeliot mutatása párhuzamosan
        - Feladat: E1. feladat: Írjunk programot, amely egy szabászat személyi nyilvántartása (személyi számok) alapján eldönti, hogy dolgozik-e férfi ezen a munkahelyen!
    - Kiválasztás tétele
        - Algoritmus és Jeliot mutatása párhuzamosan
        - Órai munka vagy házi feladat: Ki2-es feladat: Írjon programot, ami megadja, hogy egy magyar kártya mennyit ér a 21-es játékban!
    - Keresés tétele
        - Algoritmus és Jeliot mutatása párhuzamosan
        - Házi feladat: Ke1-es feladat: Nyelvvizsgán a nyelvtani tesztek pontszámait (0..maxP, maxP>0) ülési sorrendben jegyezték föl. Keressünk olyan vizsgázót, aki ugyanannyi pontot kapott, mint valamelyik szomszédja!
- 7. gyakorlat
    - Programozási tételek II.
        - Megszámolás, Sorozatszámítás
    - Megszámolás tétele

        - Algoritmus és Jeliot mutatása párhuzamosan
        - Órai munka vagy házi feladat: Egy T fős baráti társaság egy nyaralás minden napján feljegyezte, hogy aznap mennyit költöttek. Készíts programot, ami meghatározza, hogy hány olyan nap volt, amikor fejenként 100 eurónál is többet költöttek.
    - Sorozatszámítás tétele
        - Algoritmus és Jeliot mutatása párhuzamosan
    - Házi feladat: Egy papírgyűjtési akcióban mindenkiről feljegyezték, hogy ki hány kiló papírt hozott. Írj programot, ami megadja, hogy összesen mennyi papírt gyűjtöttek a résztvevők!
- 8. gyakorlat
    - 1. csoportzárthelyi
    - Programozási tételek III.
        - Maximum kiválasztás (Max. java)
    - Eljárások és függvények bevezetése
    - Maximum kiválasztás
        - Algoritmus és Jeliot mutatása párhuzamosan
        - Feladat: 4. gyakorlat feladatát átírni eljárásokba
    - Házi feladat: Készítsünk hazugságvizsgáló gépet! A gép érzékeli a vizsgált személy pulzusát, s jelez, ha a pulzusszám K fölé emelkedik.
- 9. gyakorlat
    - Összetett programozási tételek I.
    - Kiválogatás tétele
        - Jeliot-tal meg lehet mutatni, hogy mely egyszerűbb tételekből épül össze ez a tétel
    - Feladat és házi feladat: „Adott egy N létszámú osztály, amely 5 db tárgyat tanult a tanév folyamán. Egy táblázatban tároljuk az év végi jegyeket. Írd ki a kitűnő tanulók sorszámait és darabszámukat!”
- 10. gyakorlat
    - Összetett programozási tételek II.
    - Szétválogatás és Másolás
        - Jeliot-tal meg lehet mutatni, hogy mely egyszerűbb tételekből épül össze ez a tétel
    - Feladat: „Ismert egy iskola tanulóinak névsora a tanulók személyi számával. Válasszuk szét a lányokat és a fiúkat!”
    - Metszet tétel
    - Házi feladat: Egy lóversenyen ugyanazoknak a lovaknak két fordulóban kell helytállniuk. Adjuk meg azokat a lovakat, amelyek mindkét fordulóban célba értek!
    - (Házi feladat 2: Unió tétel megértése
- 11. gyakorlat
    - 2. csoportzárthelyi

  - Rekordok használata (a Jeliot nem képes rekordot megjeleníteni, mert a Java sem ismeri a rekordot, rekordhoz hasonló megjelenítéshez a java fájlban kell új osztályt létrehozni)
- 12. gyakorlat
  - Ha marad rá idő és a csoport is „kellő szinten van”: Rendezések
  - Egyébként: Minta évfolyam ZH
- 13. gyakorlat
  - Minta évfolyam ZH, gyakorlás

4.2 A kutatás módszereinek, eszközeinek bemutatása

4.2.1 Előzetes tanulmányokat felmérő kérdőív

A kutatás első mozzanata egy előzetes tanulmányokat felmérő kérdőív kitöltetése volt a kísérleti és a kontroll csoport hallgatóival (a kérdőív a Függelékben 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 szolgáltatta annak az alapját, hogy mérni lehessen a hallgatók fejlődését. 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, azaz mennyire figyel a hallgató az apróbb részletekre.

Az 3. kérdés (Írj algoritmust (saját szavaiddal) arról, hogy hogyan mész át egy jelzőlámpás kereszteződésen!) egy hétköznapi példát hoz fel, egy naponta akár többször használt algoritmust. A kérdésre adott válasznál kideríthető, hogy a ciklusok használata mennyire természetes a hallgató számára, hiszen a lámpánál való várakozás algoritmusa így írható le a legkönnyebben: várakozás, amíg a lámpa zöld nem lesz. A feladatot ciklus használata nélkül is meg lehetett oldani, bár a hibázás lehetősége ekkor nagyobb. Az algoritmus helyességét 0 vagy 1 ponttal értékeltem, míg az algoritmus részletességére adott pontszámok így alakultak:

- 0 pont – A hallgató nem írt semmit vagy értékelhetetlen a válasz.
- 1 pont – alap megoldás: Megállok a zebra előtt. Megnézem a jelzőlámpát. Ha piros, akkor várok, amíg zöld nem lesz. Ha zöld, akkor elindulok a zebrán. Megyek, amíg át nem érek.

- 2 pont – a hallgató felételezi, hogy van nyomógombos lámpa is, és/vagy figyelembe veszi a jelzőlámpa villogó zöld állapotát is.
- 3 pont – a hallgató körülnéz az úton és/vagy foglalkozik azzal is, hogy a lámpa nem működik.

Ennél a feladatnál a részletesebb algoritmusok az adott lépés előfeltételét ellenőrizték. A feladatra kapott pontszám így alakult: (helyesség*részletesség)+(részletesség/2). Egy helytelen, de részletesebb algoritmus differenciáltabb gondolkodást mutat, mint egy helytelen, de nem részletezett algoritmus, ezért értékesebb, és ezt a fenti módon vettem bele a számításba.

A 4. kérdés (Írj algoritmust (saját szavaiddal) arról, hogy hogyan kell használni egy kávéautomatát!) is egy hétköznapi példa megoldását kéri. A válaszból kideríthető, hogy a válaszadó milyen akkurátusan figyel az előfeltétel ellenőrzésre, illetve mennyire precízen fogalmaz. Ugyanis a kért termék árától függ, hogy mennyi pénzt dobok be az automatába, illetve rengeteg beállítási lehetőség van egy kávéautomatánál. A feladat helyességénél a legfontosabb szempont az volt, hogy a hallgató fogalmazza meg egyértelműen, hogy legalább annyi pénzt dob be, amennyi szükséges az általa választott ital elfogyasztásához. Nem vettem hibának, ha pl. nem az automatánál itta meg az italt vagy ott felejtette (vagy meg sem említette) a visszajáró pénzt, mivel a cél az volt, hogy igyon valamit. A helyesség pontozása ugyanolyan, mint a fenti feladatnál, a részletesség pontszámai így alakultak:

- 0 pont – A hallgató nem írt semmit vagy értékelhetetlen a megoldás.
- 1 pont – Majdnem helyes megoldás, az elegendő pénz meghatározása hiányzik.
- 2 pont – Alap megoldás: 1. Kiválasztod, milyen italt szeretnél fogyasztani 2. Előveszel annyi pénzt, amennyit kell 3. Bedobod az automatába 4. Megnyomod a kiválasztott italhoz tartozó gombot. 5. Megvárod, míg elkészül 6. Kiveszed a gépből 7. Megiszod
- 3 pont – Az algoritmus megemlíti a visszajáró pénzt.
- 4 pont – Cukor, tej mennyiségének külön bekérése.
- 5 pont – A hallgató számol azzal, hogy a gép rossz, és/vagy a gép nem tud visszaadni

Annál több pont járt a feladat részletességére, minél fontosabb előfeltételt adott meg a feladat megoldója: ahhoz, hogy tudjak kávét inni, működnie kell az automatának, ahhoz, akkor tudok biztosan tejet, cukrot az italba tenni, ha ellenőriztem azok mennyiségét, stb. A feladat egyszerűbb volt, abból a szempontból, hogy nem volt szükséges ciklust használni, viszont nehezebb volt, mert több részletet kellett végiggondolni, akkurátusabban kellett bizonyos lépéseket megfogalmazni, pl. mennyi pénzt dobok bele az automatába. A feladatra kapott pontszám ugyanúgy alakult, mint az előző feladatnál.

A 5. kérés (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!) különbözött az előző kettőtől. Egyrészt nem a „hétköznapok" egy problémáját kellett megoldani, másrészt a feladat szövege lényegében tartalmazta az algoritmust „hétköznapi nyelven" megfogalmazva. A hallgató feladata az volt, hogy a saját nézőpontjából oldja meg a feladatot, illetve bontsa le „elemi" lépésekre. Ily módon ez a feladat valójában a hallgató absztrakciós készségét igyekszik mérni: egy általánosan megfogalmazott feladatot kellett saját szemszögéből megvalósítani, azaz konkretizálni. Tehát a feladatszöveg gyakorlatilag specifikációt adott a hallgató kezébe, bemenettel, kimenettel, elő- és utófeltétellel, és ehhez kellett a hallgatónak precíz algoritmust írnia. A feladat megoldását nem lehetett nagyon részletezni. 0 pont járt az értékelhetetlen megoldásért, és 1 pont járt az alap megoldásért: 1. Gondolj egy pozitív egész számra! 2. Olvasd be! 3. Vizsgáld meg a számot! 4. Jelenítsd meg: a) ha nem pozitív egész, jelezd a hibát és lépj ki! b) ha helyes a szám, írd ki a képernyőre!

A feladat pontjainak számítása változott az előzőekhez képest: helyesség*részletesség+részletesség. Ennek oka az volt, hogy ez a feladat nagyobb súllyal szerepeljen az összesített pontszámban.

Ilyen módon maximum 14 pontot lehetett kapni a három feladatra.

Ahhoz, hogy a feladatok eredményéhez képest meg lehessen határozni a fejlődést, ötfokozatúvá kellett alakítani az értékelést, hasonlóan a többi eredményhez. A pontozási rendszerhez a kurzus évfolyam zárthelyi dolgozatának ponthatárait alkalmaztam:

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

Tehát, ha valaki mindegyik feladat alap megoldását írta le, akkor 6,5 ponttal 2-est kapott, ilyen módon a megfelelő diszkutálásnak nagy szerepe van, hasonlóan a későbbi számonkérésekhez.

4.2.2 Az előrehaladás mérése

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,
- 2. csoportzárthelyi: három egyszerű programozási tétellel megoldható feladat algoritmizálása (struktogrammal), specifikáció alapján,
- beadandó feladat készítése, ahol négy részfeladatot kellett megoldani, amelyek közül az egyikben előfordultak összetett programozási tétellel megoldható feladatok,
- Évfolyam zárthelyi dolgozat, amelynél a négy részfeladatból 2 volt olyan, amelyet összetett programozási tétellel kellett megoldani.

A fejlődés meghatározásánál nem vettem figyelembe a beadandók eredményét, ugyanis ennél a számonkérésnél kevéssé garantálható, hogy a hallgató önálló munkát adjon be, ezért ennek hitelesség erősen 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, illetve a súlyozás mértéke arányos a számonkérések időbeli távolságával, ugyanis a hallgatók az első csoportzárthelyit a félév felénél, a másodikat a félév háromnegyedénél és az évfolyam zárthelyit pedig a félév végén írják. A fejlődés mértékét (*F*) a következő képlettel határoztam meg:

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

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 18-cal való osztásnak az a szerepe, hogy pl. ha valaki 1-est kap az előzetes tanulmányok 3-5. kérdéseire és az összes jegye ötös, akkor 18 pontot érhet el maximum. Tehát a fenti képlet eredménye egy -1 és 1 közötti valós szám lesz. Az eredmény előjele mutatja a fejlődés irányát, az értéke pedig a nagyságát. Minél későbbi feladatban kapott jobb osztályzatot a hallgató a bemeneti feladathoz képest, annál nagyobb lesz a fejlődését meghatározó szám.

4.3 A kutatás eredményeinek értékelése

A Függelék F1-F24. ábráin láthatóak az előzetes tanulmányokat felmérő kérdőív eredményei, kördiagramon ábrázolva.

Informatika óraszámok	Összesített eredmény	Kísérleti csoport	Kontroll csoport
0	7%	9%	4%
1	14%	10%	18%
2	38%	40%	36%
3	7%	8%	7%
4	34%	33%	35%

A fenti táblázatból látható, hogy a kísérletben résztvevők 72%-ának heti 2, illetve 4 informatika órája volt egy héten. A hallgatók jelentős arányánál nincs szignifikáns különbség a két csoport között.

Programozás óraszámok	Összesített eredmény	Kísérleti csoport	Kontroll csoport
0	47%	51%	44%
1-4	17%	16%	19%
5-8	6%	6%	5%
9-14	2%	1%	3%
15 vagy annál több	28%	26%	29%

A fenti táblázat meglepő eredménye az, hogy a hallgatók majdnem fele, saját bevallásuk szerint, a középiskolába nem tanult programozást, pedig a hatályos kerettanterv szerint kellett volna. Ez az eredmény indokolja, hogy miért nem a közoktatásban végeztem el ezt a vizsgálatot. Azt is le lehet vonni tapasztalatként, hogy a kontroll csoport tagjai valamivel nagyobb arányban tanultak programozni, tehát feltehetően valamivel nagyobb gyakorlatuk és több előzetes tudásuk van az induláskor, mint a kísérleti csoportnak.

Helyesség	Összesített eredmény		Kísérleti csoport		Kontroll csoport	
	Helyes	Helytelen	Helyes	Helytelen	Helyes	Helytelen
3. kérdés	75%	25%	72%	28%	77%	23%
4. kérdés	58%	42%	56%	44%	60%	40%
5. kérdés	76%	24%	76%	24%	76%	24%

Az algoritmikus gondolkodást, és az absztrakciós készséget felmérő feladatoknál tekintsük először a helyesség eredményét közlő táblázatot. Itt is az látszik, hogy a kontroll csoport kitöltői alig jobb eredményt értek el, viszont az absztrakciós készséget mérő 5. kérdés esetében a két csoport eredménye között semmilyen különbség nem volt. A leggyakoribb hiba a lámpánál váró ciklus szervezésével volt. Vagy nem is volt egyáltalán ciklus, vagy hibás volt a ciklusfeltétel megadása. 31 hallgató (a vizsgált hallgatók kb. ötöde) ott ragadt volna a lámpánál, mert nem számolt azzal a lehetőséggel, hogy úgy ér oda a lámpához, hogy az éppen piros. Lássunk egy példát egy tipikus hibás, ciklus nélküli algoritmusra: „Odaérek a lámpához. Ha zöld, átmegyek a túloldalra.”

A 4. kérdésnél a legmagasabb a rossz válaszok aránya. A leggyakoribb hiba az volt, hogy a hallgató nem határozta meg, hogy mennyi pénzt is dob bele az automatába, hanem az algoritmus ide tartozó lépését úgy fogalmazta meg csak, hogy „bedobom a pénzt”, de nem derült ki, hogy mennyit is pontosan, illetve, hogy elég lesz-e az általa kiválasztott ital elfogyasztásához.

Az 5. kérdés legjellemzőbb hibája az volt, hogy lemaradt a hibaüzenet kiírása, azaz a feltétel egyébként ágát már nem fogalmazta meg a hallgató.

3. kérdés részletesség	Összesített eredmény	Kísérleti csoport	Kontroll csoport
0 pont	3%	1%	4%
1 pont	74%	78%	71%
2 pont	8%	4%	12%
3 pont	15%	17%	13%

Elenyésző volt azok száma, akik értékelhetetlen módon oldották meg a feladatot. Ez a feladat hétköznapiságának köszönhető. A százalékokból leolvasható, hogy a kísérleti csoport valamivel részletesebben oldotta meg ezt a feladatot.

4. kérdés részletesség	Összesített eredmény	Kísérleti csoport	Kontroll csoport
0 pont	1%	1%	1%
1 pont	35%	37%	32%
2 pont	39%	35%	44%
3 pont	20%	33%	19%
4 pont	2%	2%	1%
5 pont	3%	3%	3%

A fenti táblázat megmutatja, hogy a válaszadók kb. harmada nem gondolt arra, hogy pontosan meghatározza a bedobott pénz mennyiségét. Ennél a feladatnál, a részletesebb megoldásokat tekintve jobban diszkutált a kísérleti csoport a kontroll csoportnál.

5. kérdés részletesség	Összesített eredmény	Kísérleti csoport	Kontroll csoport
0 pont	12%	15%	9%
1 pont	88%	85%	91%

A fenti táblázat adataiból az következik, hogy a kontroll csoport jobban megértette az utolsó feladatot.

Átlageredmények	Kísérletben részt vett hallgatók	Kísérleti csoport	Kontroll csoport
3. kérdés	1,75	1,74	1,75
4. kérdés	2,36	2,36	2,36
5. kérdés	1,63	1,60	1,67
Összpontszám	5,74	5,71	5,78
Osztályzat	1,88	1,90	1,87

Az állításaim statisztikai bizonyításához a legkisebb négyzetek módszerével lineáris regressziós modelleket becsülök, a számításokat a Gretl nevű programmal végeztem el.

Az Experimental egy ún. dummy változó, amely 0-t vesz fel, ha az adott tanuló a kontroll csoportban volt, illetve 1-et, ha a kísérleti csoportban. Ezzel egy adathalmazba kerülhet a két összevetendő adattömeg, a megkülönböztethetőség megtartása mellett.

Hipotézis: A bemeneti tesztek eredményében nincs különbség.
Model: OLS, using observations 1-153
Dependent variable: EloTanSzum

	Coefficient	*Std. Error*	*t-ratio*	*p-value*	
const	5,78	0,273533	21,1309	<0,00001	***
Experimental	-0,0748718	0,383096	-0,1954	**0,84531**	

Ebben a modellben az EloTanSzum változót becsültem az Experimental változóval. Az EloTanSzum az előzetes tudást felmérő kérdőív 3-5. kérdéseire adott válaszok összegeit tartalmazza.

A fenti táblázatok adataiból leolvasható, hogy az algoritmikus gondolkodást felmérő 3-5. kérdések eredményei hasonlóan alakultak mindkét csoportnál, nem tapasztalható szignifikáns eltérés. Ezt bizonyítja a magas *p* érték (vastagon kiemelve). Tehát kimondható, hogy a két csoport közel azonos szintről indul, tehát a későbbiekben közlendő fejlődést meghatározó értékek közötti különbség nem a bemeneti többlettudásnak köszönhető.

Fontos megjegyeznem, hogy a kérdések közül az absztrakciós készséget mérő 5. kérdésnél alakult ki a legnagyobb arányú különbség a kísérleti és a kontroll csoport között, az utóbbi javára.

Az előzetes tudást felmérő feladatok pontszámai és osztályzatainak meghatározása után ki lehetett számolni az előző alfejezetben ismertetett képlet alapján a fejlődést.

	Kísérletben részt vett hallgatók	Kísérleti csoport	Kontroll csoport
Átlagos fejlődés	0,33	0,41	0,24

A fenti adatokból egyértelműen következik, hogy az AV rendszert használó csoportok fejlődésének átlaga nagyobb volt, mint a kontroll csoporté, akik nem használták a Jeliot rendszert a gyakorlatok során.

Fontos megvizsgálnunk azt is, hogy mely hallgatók fejlődtek a legtöbbet. Ehhez szükségünk van arra, hogy meghatározzuk az előzetes tanulmányok kérdéseire kapott pontszámok alapján a hallgatók eloszlását és az adott pontszámot elért hallgatók átlagos fejlődését. Megjegyzem, hogy a fejlődést csak azoknál a hallgatóknál tudtam meghatározni, akik az összes számonkérésen szereztek értékelhető osztályzatot.

Előzetes tanulmányok pontszám	Darabszám	%	Átlagos fejlődés
0	2 db	1%	0,07
1	3 db	2%	0,71
2	7 db	5%	0,56
3	13 db	8%	0,60
4	30 db	20%	0,58
5	21 db	14%	0,38

6	35 db	23%	0,35
7	10 db	7%	0,13
8	19 db	12%	0,05
9	6 db	4%	0,01
10	0 db	0%	-
11	5 db	3%	-0,16
12	1 db	1%	-1,00
13	1 db	1%	-0,75
14	0 db	0%	-

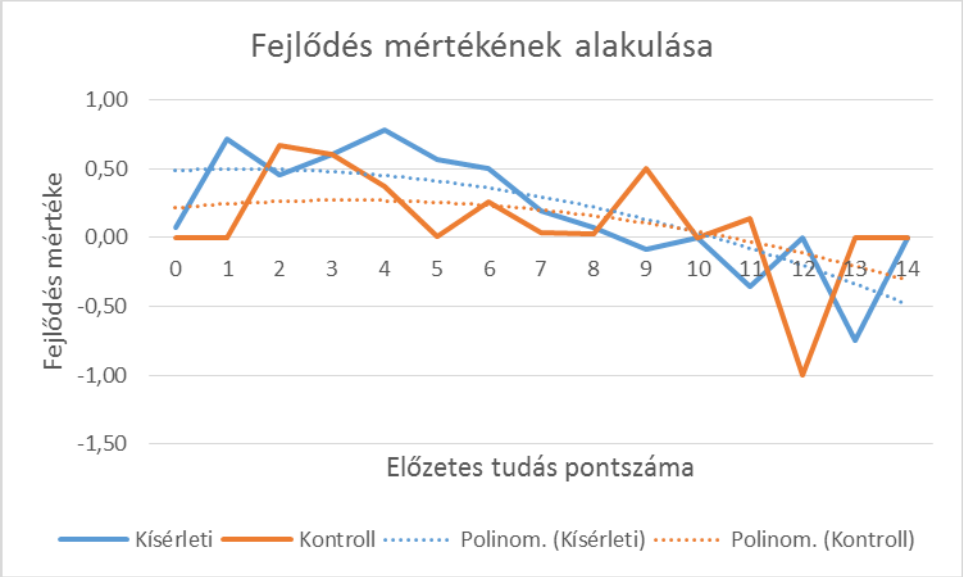
A hallgatók túlnyomó része, 84%-a (sárgával kiemelve), a [3,9) intervallumban szerzett pontot az előzetes tudást felmérő feladatokon. Kitűnik az is, hogy a kísérletben résztvevőknél – kevés kivételtől, a minta mintegy 9%-tól eltekintve – minél kevesebb tudással érkezett egy hallgató, annál többet fejlődött. Ebből az következik, hogy a kurzus betölti felzárkóztató szerepét, hiszen általában a gyengébb képességű hallgatók fejlődését segíti a leginkább.

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

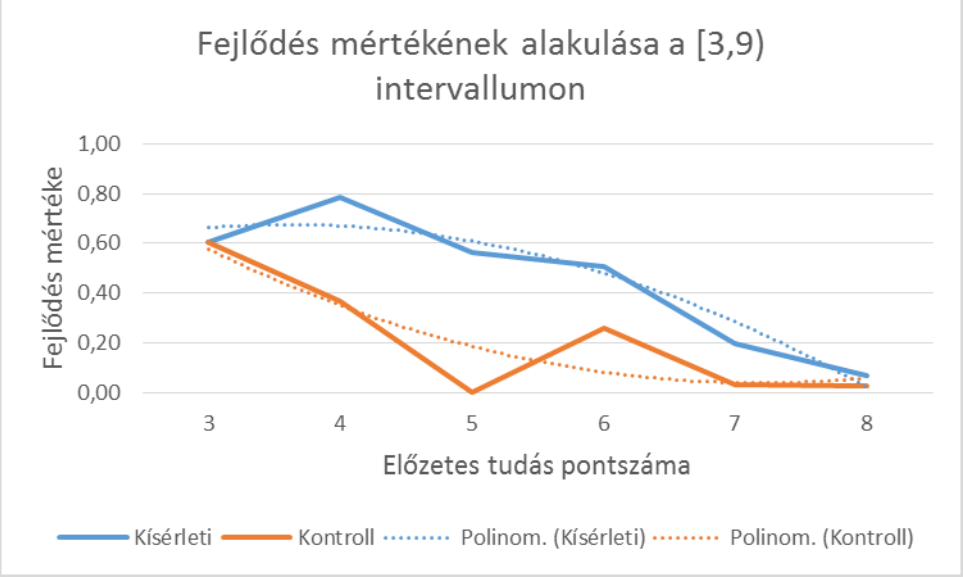
Előzetes tanulmányok pontszám	Darabszám		%		Átlagos fejlődés	
	Kísérleti csoport	Kontroll csoport	Kísérleti csoport	Kontroll csoport	Kísérleti csoport	Kontroll csoport
0	2 db	0 db	3%	0%	0,07	-
1	3 db	0 db	4%	0%	0,71	-
2	3 db	4 db	4%	5%	0,45	0,67
3	6 db	7 db	8%	9%	0,60	0,60
4	14 db	16 db	18%	21%	0,79	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%	-	-

Az eredményekből látszik, hogy a kísérleti csoport 78%-a, míg a kontroll csoport 89%-a esik bele a sárga háttérrel kiemelt intervallumba. A táblázat azt is megmutatja, hogy ebben a részben, a [3,4) intervallumot kivéve mindig magasabb volt a fejlődés mértéke a kísérleti

csoportnál. Ezt a tényt szemlélteti az alábbi két diagram is, a második a kiemelt [3,9) intervallum fejlődési értékeit mutatja:



22. ábra. *Fejlődés mértékének alakulása.*



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

Hipotézis: A kísérleti csoportban nagyobb a fejlődés
Model: OLS, using observations 1-153
Dependent variable: Fejlodes

	Coefficient	*Std. Error*	*t-ratio*	*p-value*	
const	0,224444	0,0513625	4,3698	0,00002	***
Experimental	**0,183319**	0,0719357	2,5484	**0,01182**	**

Hipotézis: A kísérleti csoportban nagyobb a fejlődés [3,9) intervallum
Model: OLS, using observations 1-128
Dependent variable: Fejlodes

	Coefficient	*Std. Error*	*t-ratio*	*p-value*	
const	0,22471	0,0486261	4,6212	<0,00001	***
Experimental	**0,274152**	0,0704385	3,8921	**0,00016**	***

Hipotézis: A kísérleti csoportban nagyobb a fejlődés [4,7) intervallum
Model: OLS, using observations 1-86
Dependent variable: Fejlodes

	Coefficient	*Std. Error*	*t-ratio*	*p-value*	
const	0,225845	0,0512929	4,4031	0,00003	***
Experimental	**0,39846**	0,0752102	5,2980	**<0,00001**	***

A statisztikai számítást elvégeztem a teljes mintára, a [3,9) és a [4,7) intervallum (előzetes tudást felmérő kérdőív pontszáma alapján) mintájára is. Az előzetes tudást felmérő kérdőív átlagos pontszáma 5,74 lett, így a [4,7) intervallumba eső hallgatók tekinthetőek az átlag közeli tudású hallgatóknak.

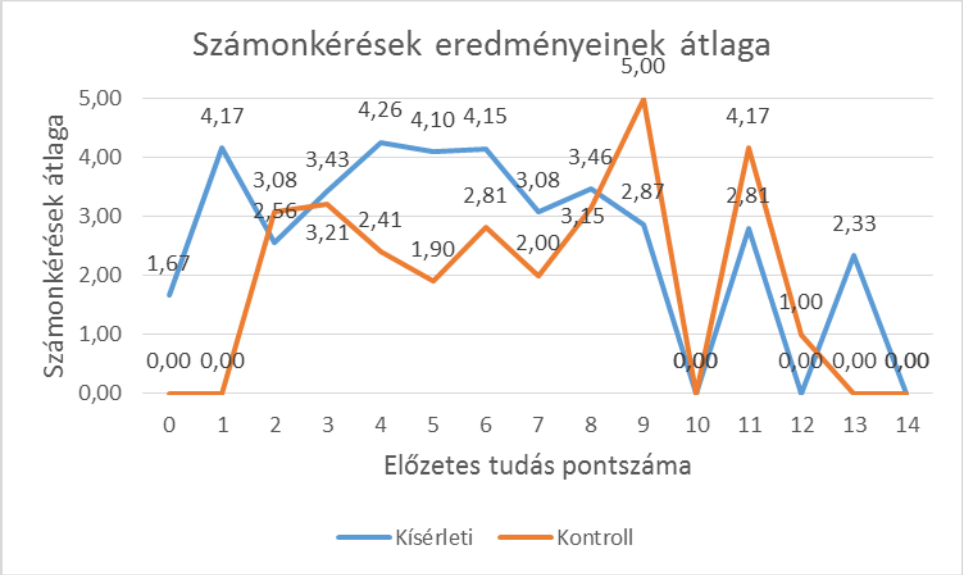
Ezekben a modellekben a Fejlodes változót becsültem az Experimental változóval. A Fejlodes a hallgatók fejlődésének mértékét tartalmazza. A *p* értékek mutatják, hogy a kísérleti csoportban a fejlődés mértéke szignifikánsan nagyobb volt, mint a kontroll csoportban, illetve a koefficiens arról árulkodik, hogy a teljes mintában átlagosan 0,183319, a [3,9) intervallumban 0,274152, illetve a [4,7) intervallumban, azaz az átlag közeli tudású hallgatóknál 0,39846 ponttal fog többet fejlődni az a hallgató, aki a kísérleti csoportban szerepelt.

Megjegyzem, hogy a hipotézis csak az átlag közeli tudású hallgatókról beszél, így az első modell eredménye azt mutatja, hogy a hipotézis állítása a teljes mintára is érvényes. Látható az is, hogy minél inkább átlag közeli hallgatókra szűkítem le a mintát, úgy nő a koefficiens mértéke a kísérleti csoport javára, illetve az is észre vehető, hogy eközben a kontroll csoport átlagos fejlődése (const koefficiense) alig változott. Tehát általánosan is kimondható, hogy azok a hallgatók, akik tanulmányaik során használták az AV rendszert, szignifikánsan többet fejlődtek, mint a kontroll csoportban levő hallgatók.

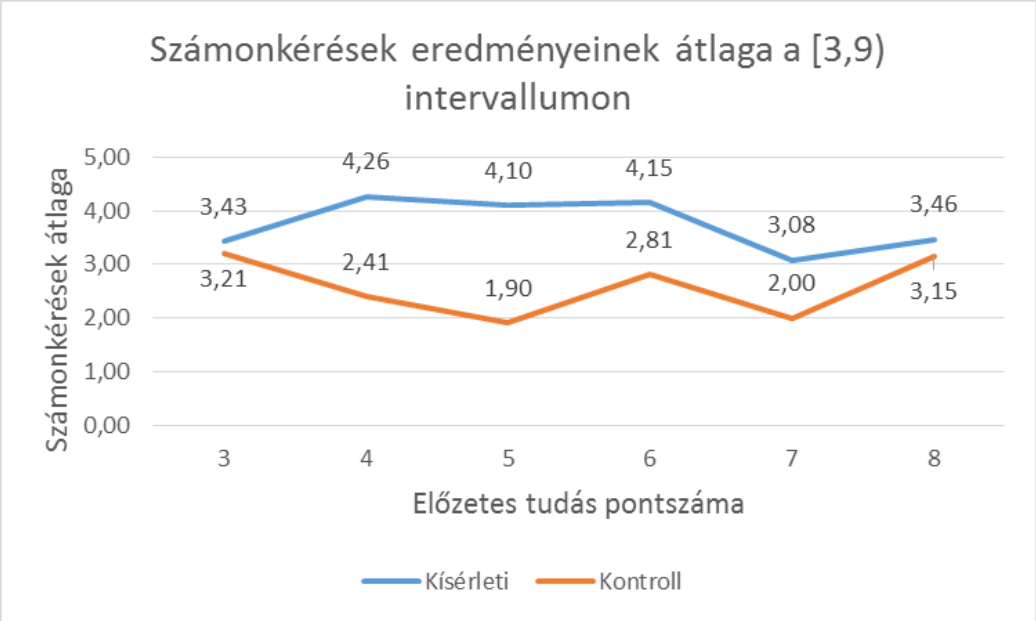
A fenti diagramok és statisztikai bizonyítás alapján levonható az a következtetés, hogy a kísérletben résztvevő hallgatók 84%-ánál az AV-t használó csoportok nagyobb fejlődést értek el, mint a kontroll csoport hallgatói. Különösen az átlagos, illetve az alatti hallgatók felzárkózását segítette igazán 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.

A fentiek alapján a Hipotézis 2. (Az AV eszközzel való tanítás az átlag közeli tudású hallgatók felzárkózását segíti) igazolva lett.

Külön vizsgálat tárgyát képezte az félév végi eredmények összehasonlítása.



24. ábra. *Számonkérések eredményeinek átlaga.*



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

Az eredmények vizsgálatánál nem a gyakorlati jegyeket vettem figyelembe, ugyanis a számonkérések átlaga jobban tükrözi a hallgató teljesítményét. A grafikonon a 0,00 átlagú pontok azt jelenti, hogy az adott intervallumban nem volt hallgató.

Hipotézis: Az AV használók jobb eredményeket érhetnek el (átlagosan), mint a vizualizációt nem használók
Model: OLS, using observations 1-153
Dependent variable: AtlagEredmeny

	Coefficient	*Std. Error*	*t-ratio*	*p-value*	
const	2,73556	0,163447	16,7367	<0,00001	***
Experimental	**0,926838**	0,228915	4,0488	**0,00008**	***

Ebben a modellben az AtlagEredmeny változót becsültem az Experimental változóval. Az AtlagEredmeny a hallgatók számonkéréseikre kapott osztályzatok átlagait tartalmazza. A *p* értéke azt mutatja, hogy a kísérleti csoportban levő hallgatók szignifikánsan jobb eredményt értek el, mint a kontroll csoportban levők. Tehát a kísérleti csoportban levők átlagosan 0,926838 jeggyel értek el magasabb átlagot a számonkérések alapján, mint a kontroll csoport hallgatói.

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. Ilyen módon a Hipotézis 3. (Az AV-t használó hallgatók többet fejlődnek és jobb eredményeket érhetnek el, mint a vizualizációt nem használók) is igazolást nyert.

Az évfolyamzárthelyi alapján végzett előrehaladás-vizsgálat is azonos eredményt mutat a fent idézettekkel. Ez azért fontos a fenti két hipotézis igazolása szempontjából, mert az évfolyamzárthelyi a leginkább hiteles közös mérce, ugyanis ez a számonkérés van a legkevésbé kitéve az egyes csoportokon belüli szubjektivitásnak.

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.

Azért állítható párba az 5. kérdés és a 2. csoportzárthelyi eredménye, mert az 5. feladat szövege lényegileg specifikációt adott a hallgató kezébe, bemenettel, kimenettel, elő- és utófeltétellel, és ehhez kellett a hallgatónak algoritmust írnia, hasonlóan a 2. csoportzárthelyi feladatához, ahol a specifikáció már formálisan volt megadva. Kutatások szerint [12, 13, 43] kapcsolat létezik a formalizált gondolkodás, az absztrakciós képesség és az algoritmikus gondolkodás között.

	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.

Az AV használata jobban fejleszti az absztrakciós készséget, mint ha nem kerülne használatra ez az eszköz
Model: OLS, using observations 1-153
Dependent variable: ZH2

	Coefficient	*Std. Error*	*t-ratio*	*p-value*	
const	2,84	0,189036	15,0236	<0,00001	***
Experimental	**0,967692**	0,264755	3,6551	**0,00035**	***

Ebben a modellben a ZH2 változót becsültem az Experimental változóval. A ZH2 a hallgatók második csoportzárthelyi dolgozatukra kapott jegyeit tartalmazza. A *p* értéke azt mutatja, hogy a kísérleti csoportban levő hallgatók szignifikánsan jobb eredményt értek el, mint a kontroll csoportban levők. Tehát a kísérleti csoportban levők átlagosan 0,967692 jeggyel kaptak magasabb osztályzatot a második csoportzárthelyin, mint a kontroll csoport hallgatói. A fenti bizonyítás alapján kimondható, hogy az AV használatának következtében a formális algoritmusok megértése javult, ezért a bátrabban alkalmazták ezeket a feladat megoldása közben, így a Hipotézis 4. is igazolásra került.

4.4 A kurzus eredményeinek értékelése

Érdemes megvizsgálni a kurzus kimenetét, azaz milyen eredmények születtek a kísérleti csoportokban. Ebben az összehasonlításban csak a csoportok által elért félév végi eredményt vizsgálom.

A kísérleti csoport induló létszáma 88, a kontroll csoporté 84 fő volt. A kísérletben részt vevők közül nem mindenki volt ott az első gyakorlaton, ezért alacsonyabb az előzetes tudást felmérő kutatásban részt vettek száma (19 fővel).

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

A fenti táblázatból látható, hogy az AV programot használók értek el jobb átlagot, majdnem egy jeggyel jobbat, mint a kontroll csoport hallgatói. Ez az eredmény is megerősíti a Hipotézis 3.-at.

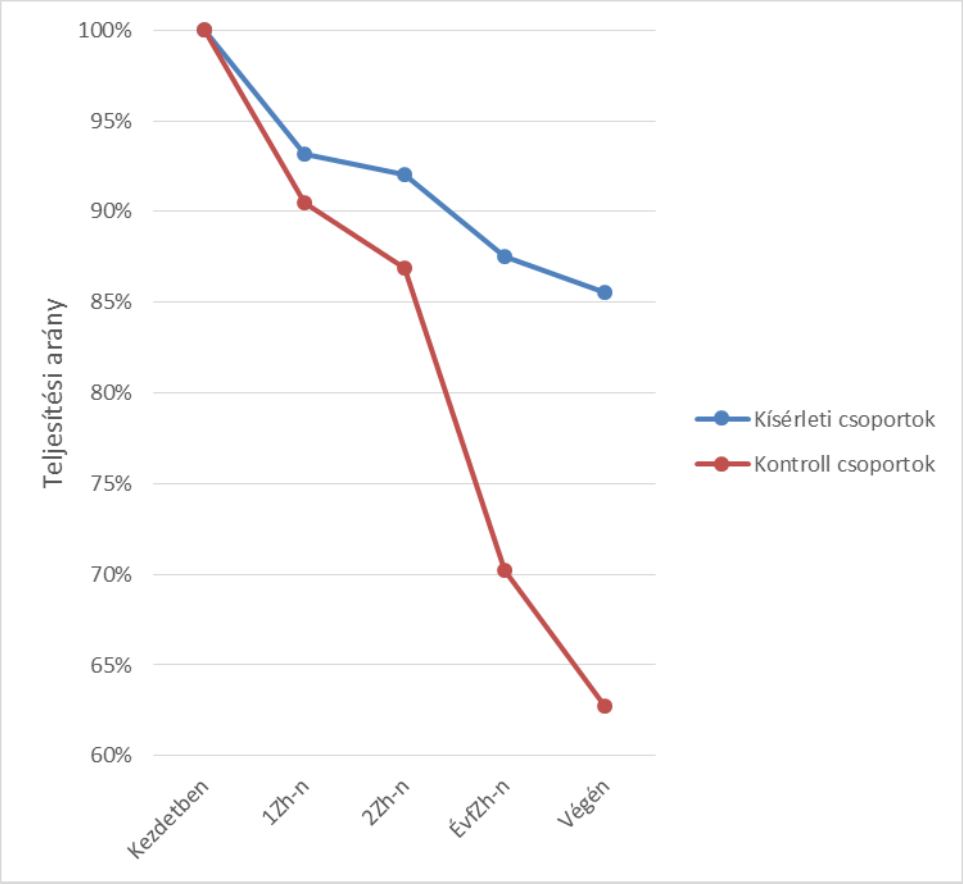
	Számonkérések átlaga	Kísérleti csoport		Kontroll csoport	
Számonkérések átlagainak eloszlása	1	16 db	18%	35 db	42%
	1,5	0 db	0%	0 db	0%
	2	2 db	2%	3 db	4%
	2,5	6 db	7%	3 db	4%
	3	6 db	7%	12 db	14%
	3,5	9 db	10%	9 db	11%
	4	21 db	24%	7 db	8%
	4,5	9 db	10%	9 db	11%
	5	19 db	22%	6 db	7%

A táblázat eredményei megmutatják, hogy 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%). Ez az eredmény is megerősíti a Hipotézis 3.-at, illetve mutatja, hogy az AV rendszerrel való tanulás/tanítás nagyobb fejlődést ért el a hallgatóknál.

Teljesítési arányok	Kísérleti csoport	Kontroll csoport
Kezdetben	100%	100%
1Zh-n	93%	90%
2Zh-n	92%	87%
ÉvfZh-n	88%	70%
Végén	86%	63%

Vizsgáltam azt is, hogy a két csoportnál mekkora volt a teljesítők aránya a számonkérések (és így az idő) függvényében. Látható, hogy a kísérleti csoportban kisebb volt a lemorzsolódás, amelyből arra lehet következtetni, hogy az AV eszköz használata pozitívan hatott a motivációra.



26. ábra. *Teljesítési arányok grafikonja*

4.5 A vizsgálat eredményének összevetése a nemzetközi irodalommal

Az algoritmus vizualizáció történeténél utaltam már külföldi eredményekre, alább azok ismertetésére szorítkozom, amelyek a legközelebb állnak kutatásom témájához és módszeréhez.

Hansen és munkatársai [16] arra keresték a választ, hogy az AV eszközzel tanuló hallgatók vagy a munkafüzetből tanulók tanulnak többet, azaz érnek el jobb eredményt a kísérletet lezáró teszten. A két kísérlet között az volt a különbség, hogy az elsőt kezdő hallgatókkal végeztették el, míg a másodikat tapasztaltakkal.

A két kísérlet abban hasonlított, hogy volt egy előzetes tudást felmérő teszt, illetve egy teszt a kísérlet végén. Ugyanígy építettem föl a fentebb említett vizsgálatomat én is, viszont abban eltértem tőlük, hogy a felmérésemben a fejlődést *több számonkérés* eredménye, és *nem csak egy* alapján határoztam meg.

Hansenék kísérletében az összefésülő rendezés (merge sort) algoritmusát tanították a hallgatóknak. Az AV csoport a HalVis fantázianevű AV eszközt használta a téma feldolgozásához, a kontroll csoport pedig egy 6 oldalas tananyagot.

A következő eredményekről számoltak be a szerzők:

	Kísérlet kezdő hallgatókkal		**Kísérlet tapasztalt hallgatókkal**	
	Előzetes teszt	Kísérlet utáni teszt	Előzetes teszt	Kísérlet utáni teszt
Kontroll csoport átlagos eredménye	27%	43%	44%	53%
AV csoport átlagos eredménye	28%	74%	48%	71%

A táblázatból kiolvasható, hogy az előzetes tudásuk alapján hasonló szintről indultak a csoportok, viszont a kezdőknél sokkal nagyobb volt a fejlődés mértéke a kontroll csoporthoz képest (46%-os növekedés a 16%-kal szemben, azaz majdnem 3-szoros az előrehaladás mértéke). Az is látható, hogy tapasztaltabb hallgatóknál kisebb a hatása az AV eszköznek („csak” 23% a 9%-kal szemben, ami *relatív* fejlődés szempontjából alig marad le a kezdőknél tapasztaltaktól).

2003-ban Ben-Bassat Levy és munkatársai végeztek kísérletet a Jeliot 2000 AV eszközzel [3] (a Jeliot 3 elődjével). Hasonlóan az én vizsgálatomhoz, egy bevezető jellegű programozási kurzusnál használták az algoritmus animációt. Minden új témakört két hétig tanítottak, heti 2 óra elméleti és heti 1 óra gyakorlati oktatás mellett (a gyakorlatokon a Turbo Pascal környezet használták). Minden témakör elején volt egy előzetes teszt (maximum 100-100 ponttal), majd a gyakorlatokon a kontroll csoport csak a Turbo Pascal szolgáltatásait használta, míg a kísérleti csoport a Jeliot 2000-et is felhasználta a tanulás folyamán. Minden témakört egy feladatsor megoldása zárt.

Az év végén egy újabb teszt megírására került sor, hogy értékeljék hosszú távú tanulás eredményeit. Legvégül egy utólagos tesztet is megoldattak a tanulókkal a következő év folyamán, hogy a kutatók megállapítsák, mennyire volt időtálló a tudás, amit az előző évben megszereztek.

A kutatóknak nem volt lehetőségük a csoportok résztvevőit meghatározni – hasonlóan az én vizsgálatomhoz, viszont az lent látható eredmények mutatják, hogy a kontroll csoport általában jobb eredményeket ért el, mint a kísérleti csoport, ezért a kontroll csoportnál nem is tudtak szignifikáns különbséget kimutatni. Az én vizsgálatomnál a két csoport előzetes tudása hasonló volt.

Témakör	Bemeneti tesztek átlaga		Kimeneti tesztek átlaga	
	Kísérleti csoport	Kontroll csoport	Kísérleti csoport	Kontroll csoport
I/O műveletek	92	92	89	96
Elágazások	77	95	89	96
Elől tesztelős ciklus	76	83	87	88
Számlálós ciklus	67	79	80	87

Az adatokból kimutatható, hogy a kísérleti csoportnál szignifikánsan jobb eredmények az adott témakör kimeneti tesztjén, mint a bemenetin, kivéve az I/O műveletek témakört, amelynek oka a Turbo Pascal és a Jeliot 2000 nyelvi különbsége volt. Tehát kimondható az, hogy a Jeliot 2000 alkalmazásával többet fejlődtek a hallgatók, bár hozzá kell tenni, a kísérleti csoport hallgatóinak volt is miből fejlődniük a kontroll csoporthoz képest.

A fenti vizsgálat az adott témakörök előtt és után kitöltött tesztek eredményei alapján határozta meg a fejlődést, azonos súllyal, míg a saját kísérletemnél mindig az előzetes tudást

felmérő teszt eredményeihez viszonyítottam a számonkérések eredményét, és az idő múlását, illetve a tudás elmélyülését az eredmények súlyozásával fejeztem ki. Véleményem szerint az utóbbi módszer pontosabb képet ad a hallgatók fejlődéséről.

5 Összefoglalás

Dolgozatom témája az AV, mint tanítási/tanulási eszköz programozás oktatásának a módszertana.

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

Három AV eszközt mutattam be, majd az AV közoktatási informatikában való alkalmazhatóságát vizsgáltam, ahol konkrét ajánlást tettem, óraszámokkal.

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 megoldottak 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.

6 Irodalomjegyzék

[1] 51/2012. (XII. 21.) számú EMMI rendelet a kerettantervek kiadásának és jogállásának rendjéről, 5. melléklet

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

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

[4] Ben-Bassat Levy, R., Ben-Ari, M., Uronen, P. A.: *An Extended Experiment with Jeliot 2000.* First International Program Visualization Workshop, Porvoo, Finland, 2000.

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

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

[7] Brown, M. H., & Sedgewick, R. (1985). *Techniques for algorithm animation.* IEEE Software, 2(1), 28–39.

[8] Champine, G. A. (1991). *MIT project Athena: A model for distributed campus computing.* Newton, MA: Digital Press.

[9] Ebel, G., Ben-Ari M.: *The affective effects of program visualization*, ICER'06, September 9–10, 2006, Canterbury, United Kingdom

[10] Executive Summary PISA 2006: *Science Competencies for Tomorrow's World* © OECD 2007

[11] 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

[12] Futschek, G. (2006) *Algorithmic Thinking: The Key for Understanding Computer Science*. In Lecture Notes in Computer Science 4226, Springer, pp. 159 - 168.

[13] Futschek, G., Moschitz, J.: *Developing Algorithmic Thinking by Inventing and Playing Algorithms*, In: Constructionist approaches to creative learning, thinking and education, (2010)

[14] Gloor, P. (1998) *Animated algorithms.* In: Software Visualization: Programming as a Multimedia Experience (Brown, M., Domingue, J., Price, B. & Stasko, J., eds) The MIT Press, Cambridge, MA, pp. 409-416.

[15] Gurka, J. S.,& Citrin W. (1996) *Testing effectiveness of algorithm animation.* In: Proceedings of the 1996 IEEE Symposium onVisual Languages. IEEE Computer Society Press, Los Alamitos, CA, pp. 182-189.

[16] Hansen, S., Narayanan, N. H. & Hegarty, M. (2002). *Designing Educationally Effective Algorithm Visualizations.* J. Vis. Lang. Comput., 13, 291-317.

[17] 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.

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

[19] Kehoe, C. M., J. T. Stasko and A. Talor, *Rethinking the evaluation of algorithm animations as learning aids: an observational study*, International Journal of Human Computer Studies 54 (2001), pp. 265–284.

[20] Kindlon, J. D.: *The measurement of attention*. Child Psychology & Psychiatry Review, 3(2):72–78, 1998.

[21] Kölling, M., *"The Problem of Teaching Object-Oriented Programming, Part 2: Environments,"* Journal of Object-Oriented Programming, 11(9): 6-12, 1999.

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

[23] 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.

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

[25] 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.

[26] Moreno, A. and M. Joy, *Jeliot 3 in a Demanding Educational Setting*, in: Proceedings of the Fourth International Program Visualization Workshop, Florence, Italy, 2006, pp. 48–53.

[27] Molnár Gyöngyvér: *A komplex problémamegoldó képesség fejlettségét jelző tényezők*, In. Magyar Pedagógia, 103. évf., 1. szám pp. 81-103., Szeged, 2003

[28] 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.

[29] Naps, T. (1990) *Algorithm visualization in computer science laboratories.* In: Proceedings ofthe 21st SIGCSE Technical Symposium on Computer Science Education. ACM Press, New York, pp. 105-110.

[30] 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.

[31] 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.

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

[33] 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.

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

[35] 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.

[36] Stasko, J. T. (1990) *TANGO: a framework and system for algorithm animation.* IEEE Computer 23, 27-39.

[37] Stasko, J. T. (1992). *Animating algorithms with Xtango.* SIGACT News, 23, 67–71.

[38] 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.

[39] Stasko, J. T. (1997). *Using student-built algorithm animations as learning aids.* In C. M. White, C. Erickson, B. J. Klein, & J. E. Miller (Eds.), Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education (pp. 25–29), San Jose, CA.

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

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

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

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

[44] Törley Gábor: *Algoritmus-vizualizáció a programozásoktatásban*, in. Alkalmazott Multimédia Vol. 4, No. 3., pp. 81-94., 2009., Apple Magyarországi Képviselet, Budapest

[45] Törley, Gábor: *Expressiveness of programming languages and environments: a comparative study*, in. Teaching Mathematics and Computer Science Vol. 6/Infodidact, pp. 111-141., 2008., Institute of Mathematics, and Faculty of Informatics University of Debrecen, Hungary

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

[47] 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] BlueJ webodlal - http://www.bluej.org/extensions/extensions.html - Letöltve: 2012. október 30.

[I2] Brabec, F., & Samet, H. (2003). *Maryland spatial index demos Weboldal.* http://donar.umiacs.umd.edu/quadtree - Letöltve: 2013. június 18.

[I3] Code Blocks weboldal - http://www.codeblocks.org – Letöltve: 2013. július 23.

[I4] 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.

[I5] Galles, D. (2006). *Data structure visualization.* http://www.cs.usfca.edu/~galles/visualization - Letöltve: 2013. június 18.

[I6] 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.

[I7] Informatika-Számítástechnika Tanárok Egyesülete weboldal: Mi lesz veled, informatika tárgy? - http://isze.hu/download/Informatika_felmeres_honlapra.pdf – Letöltve: 2013. június 24.

[I8] Informatika-Számítástechnika Tanárok Egyesülete weboldal: Tájékoztatás az önálló informatika tantárgyról közoktatási intézmények számára - http://www.isze.hu/download/KIKTAJ.pdf - Letöltve: 2013. június 24.

[I9] IOI 2013 Rules - http://www.ioi2013.org/competition/rules - Letöltve: 2013. augusztus 28.

[I10] Jarc, D. J. (1999). *Interactive data structure visualization.* http://nova.umuc.edu/~jarc/idsv - Letöltve: 2013. június 18.

[I11] Jeliot weboldal - http://cs.joensuu.fi/jeliot - Letöltve: 2013. június 12.

[I12] 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.

[I13] Rodger, S.H. (2008). *JFLAP Weboldal.* http://www.jflap.org – Letöltve: 2013. június 18.

[I14] Scratch weboldal - http://scratch.mit.edu – Letöltve 2013. július 9.

[I15] Stasko, J. T. (1998). *JSamba.* http://www.cc.gatech.edu/gvu/ii/softvis/algoanim/jsamba - Letöltve: 2013. június 17.

[I16] Stasko, J. T. (2001). *Polka animation system.* http://www.cc.gatech.edu/gvu/ii/softvis/parviz/polka.html Letöltve: 2013. június 17.

[I17] Stern, L. (2001). *Algorithms in action.* http://ww2.cs.mu.oz.au/aia - Letöltve: 2013. június 18.

[I18] Törley, Gábor: *ICT in Finland* - http://pezsgo.web.elte.hu/publikaciok/ict_in_finland.pdf - Letöltve: 2013. augusztus 7.

[I19] *Virginia Tech Algorithm Visualization Research Group Weboldal.* (2011). http://research.cs.vt.edu/AVresearch - Letöltve: 2013. június 18.

7 Függelék

7.1 Programozási alapismeretek kurzus számonkéréseinek feladatai, egy-egy példa

7.1.1 1. csoportzárthelyi

Egy benzinkút napi forgalmát jegyezték fel.

A benzinkutas feljegyezte minden autósnál, hogy hány liter benzint tankolt. Tudjuk, hogy egy autóba maximálisan 40 liter üzemanyagot lehet tölteni. A bemeneti sorozat minden sorában egy 0 és 40 közötti ((0,40]) valós szám található. Adjunk választ az alábbi kérdésekre!

a) Hányadik autóba tankolták a legtöbb benzint?

b) Volt-e olyan autó, amely üres tankkal érkezett (Igen/Nem)?

Példa (a lényegi részhez, amely a „kísérő” szövegeket nem tartalmazza, de magyarázó szöveget igen):

Bemenet [*magyarázat*]	**Kimenet [*magyarázat*]**
5 [1≤*autók száma*≤10] 20,9 [*1. autó tankolása*] 10,0 [*2. autó tankolása*] 5,4 [*3. autó tankolása*] 39,4 [*4. autó tankolása*] 0,5 [*5. autó tankolása*]	4 [*az a) részfeladat válasza*] Nem [*a b) részfeladat válasza*]

Specifikáció

Bemenet: N:**Egész**,
Autok:Tömb[1..N:**Valós**]

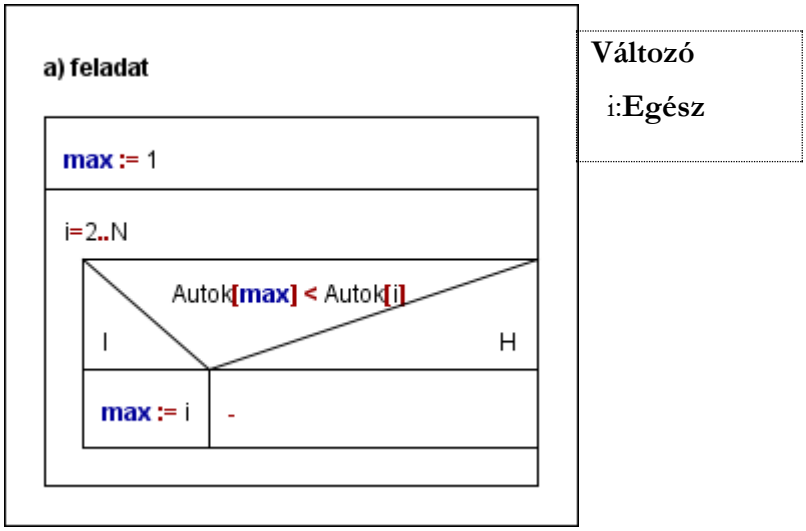
Kimenet: max:**Egész**
VanUres:**Logikai**

Előfeltétel: N∈[1,10] és ∀i∈[1..N]: 0 < Autok[i] <= 40

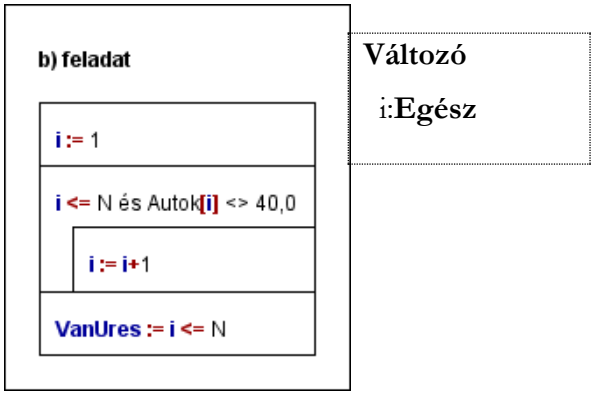
Utófeltétel: max ∈[1..N] és ∀i∈[1..N]: Autok[max] >= Autok[i] és
VanUres=LétezikE(1..N,Autok[•], • =40,0)

Algoritmus

a)



b)



Feltöltendő …

… a teljes Code::Blocks projekt tömörítve, amely neve legyen a szerző Neptun-kódja! A forrás szöveg elejére –kommentárként– írja a saját és a gyakorlatvezetője nevét!

A tömörített fájlt a Moodle-be kell feltölteni.

Értékelés

a. beolvasás 20 pont; levonások: hiányzó kérdés szöveg; hiányzó szintaktikus / szemantikus ellenőrzés

b. a) részfeladat 15 pont; levonások: nem szabályos kódolás; hiányzó vagy nem barátságos eredménymegjelenítés, algoritmusbelitől eltérő azonosítók

c. b) részfeladat 15 pont; levonások: nem szabályos kódolás; hiányzó vagy nem barátságos eredménymegjelenítés, algoritmusbelitől eltérő azonosítók

A 2-es alsóhatára: 50% (25 pont).

7.1.2 2. csoportzárthelyi

A Sarki Kisközért mindig pontosan feljegyzi, hogy milyen márkájú teából hány csomaggal rendelt egy-egy alkalommal. A Teák tömbben tároljuk a teamárkákat, a Csomagok tömbben pedig azt, hogy az i. teamárkából hány csomaggal rendeltek.

Válaszolj az alábbi kérdésekre az algoritmust megadó struktogrammal (7 pont)! Írjad az algoritmus mellé a megoldáshoz legjobban illő tétel nevét is (1 pont), ügyelj a szabályos tételalkalmazásra (2 pont).

Az alábbi kérdésekre keressük a választ:

a) Melyik teamárkából rendelte a legtöbbet a bolt egy alkalommal? (7+1+2 pont)

b) Volt-e „Teekanne” márkájú tea a rendelések között (Igen/Nem)? (7+1+2 pont)

c) Hány csomaggal rendelt a bolt „Greenfield” márkájú teát? (7+1+2 pont)

Specifikáció

Bemenet: N:**Egész**,
Teák:Tömb[1..N:**Szöveg**]
Csomagok[1..N:**Egész**]

Kimenet$_a$: maxTea:**Szöveg**

Kimenet$_b$: VoltTeekanne:**Logikai**, Volt:**Szöveg**

Kimenet$_c$: dbGreenfield:**Egész**

Előfeltétel: N>0

Utófeltétel$_a$: maxTea ∈Teák és ∀i∈[1..N], maxInd∈[1..N]: Csomagok[maxInd] >= Csomagok[i] és maxTea:=Teák[maxInd]

Utófeltétel$_b$: VoltTeekanne=LétezikE(1..N,Teák[•], • =”Teekanne”) VoltTeekanne esetén Volt:=”Igen”, egyébként Volt:=”Nem”

Utófeltétel$_C$: dbGreenfield = $\sum_{\substack{i=1 \\ \text{Teák}[i]=\text{"}Greenfield\text{"}}}^{N} \text{Csomagok}[i]$

Értékelés

A tétel megnevezése helyes: 1 pont, a specifikáció → algoritmus precíz: 2 pont; az algoritmus helyes és szabályszerű: 7 pont. A 2-es alsóhatára: 40% (3*10*40%= 12 pont).

7.1.3 Évfolyam zárthelyi

A mai napon (május 29-én) számos nevezetes esemény történt az emberiség története során. Számosakat ezek közül feljegyeztek: megadva az évet (pozitív egész), az esemény típus sorszámát (pozitív egész), és az esemény megnevezését (nem üres, akár szóközöket is tartalmazó szöveg). Az adatoktól elvárjuk (és a bemenő adataink ilyenek is lesznek), hogy év szerint (szigorúan) rendezve következzenek. Nem tudjuk azonban, hogy növekvően vagy csökkenően! Írjon programot, amely ilyen adatok ismeretében megválaszolja/megoldja az alábbiakat:

a) Legalább hány év telt el két feljegyzett esemény között? Adja meg a legkisebb követési időt, és a két –egymást követő– év$_1$, év$_2$ évet (év$_1$<év$_2$). Ha több eseménypár is van ugyanennyi távolságra, akkor az elsőként beolvasottat adja meg! 1+2

b) Két adott év között hány esemény volt? A határoló évek is számítanak! 1

c) Volt-e ókori (≤476), középkori (≤1492), újkori (≤1642), ill. modernkori (>1642) esemény a felsoroltak között? A válasz minden esetben 4 szó („IGEN” / „NEM”) egy sorban szóközökkel elválasztva! 2

d) Hány esemény típus van, amelybeli eseményből egy sincsen, és melyek ezek (a bemenet sorrendjében)? 1+3

A standard bemenet első sorában az események száma (egész szám: 2..100), a második sorban az események lehetséges típusának száma található. A harmadik sortól az egyes események adatait található: egy esemény mindig 3 sorban van. Az eseményeket leíró háromsoros blokkokban elsőként az év (egész szám: 1..2013), másodikként az esemény típusának sorszáma (egész szám: 1..típusok száma), végül az esemény megnevezése (nem üres, szóközöket is tartalmazható szöveg). Az események leírását követő sorban a b) részfeladathoz tartozó év-intervallum két határoló éve (egész számok, egy szóközzel

elválasztva; $1 \le év_1 < év_2 \le 2013$). Végül a típusok felsorolása következik, ahány típus, annyi sor (a típus mindig egy szó).

Segítségképpen:

Ha az alábbi formátumban kell beolvasni adatokat:

```
szám<sorvég>
mondat<sorvég>
```

akkor ennek beolvasása így történik:

```
string tmp;
cin >> számVáltozó;//a számVáltozóba kerül az "első" sorbeli szám
getline(cin,tmp,'\n');//"lenyeli" a <sorvég>-jelet
getline(cin,szövegVáltozó,'\n');//a szövegVáltozóba kerül a mondat
```

Minta:

	Input (billentyűzet)		**Output (képernyő)**
#	**Sortartalom [*magyarázat*]**	**#**	**Sortartalom [*magyarázat*]**
1.	4 [2≤*események szám*≤100]	**1.**	69 1453 1522 [*az a) részfeladathoz*]
2.	4 [0<*típusok száma*≤10]	**2.**	2 [*a b) részfeladathoz*]
3.	862 [*1. esemény éve*]	**3.**	NEM IGEN IGEN NEM [*a c) részfeladathoz*]
4.	1 [*1. esemény típus-sorszáma*]	**4.**	1 szerencsetlenseg [*a d) részfeladathoz*]
5.	Jutland… [*1. esemény*]		
6.	1205 [*2. esemény éve*]		
7.	2 [*2. esemény típus-sorszáma*]		
8.	II. Andras… [*2. esemény*]		
9.	1453 [*3.esemény éve*]		
10.	3 [*3. esemény típus-sorszáma*]		
11.	Konstantinapoly… [*3. esemény*]		
12.	1522 [*4. esemény éve*]		
13.	3 [*4. esemény típus-sorszáma*]		
14.	Elesik… [*4. esemény*]		
15.	1000 1500 [*évintervallum a b) részfeladathoz*]		
16.	alapitas [*1. típus*]		
17.	koronazas [*2. típus*]		
18.	haboru [*3. típus*]		
19.	szerencsetlenseg [*4. típus*]		

A standard kimenetre tehát **4** sort kell kiírni! A válaszok **egy-egy sorba** írandók, a válaszelemeket a soron belül **egy-egy szóköz** válassza el egymástól! Ha a részfeladatok valamelyikét nem tudja megoldani, akkor az eredménye helyett egy üres sort írjon ki! Ezeken kívül semmi mást nem szabad kiírni! A program végleges változatában **ne** maradjon **billentyűre várakozás** (a tesztrendszer nem képes billentyűket nyomogatni ☺)!

Csak a feladat érdemi megoldását célzó programokat értékelünk, a tesztelő rendszer próbára tételét célzó megoldások 0 pontosak, a belefektetett munka ellenére! ☺

Értékelés

Értékelés 10 teszt-adatfájl alapján:

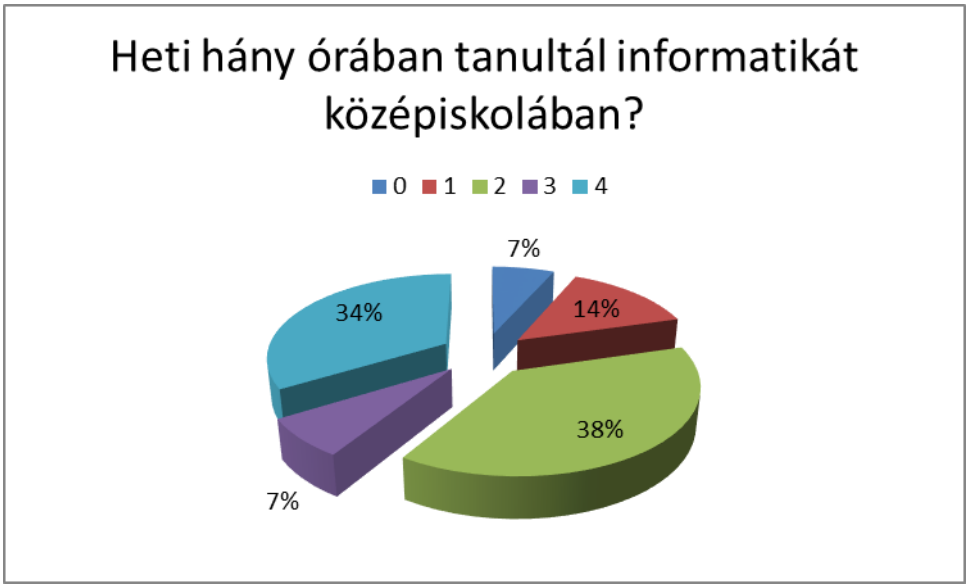
Összpont: **10*(1+2+3+4)=10*10=100 pont**

Alsópont:	**40**	**55**	**70**	**85**
Jegy:	**2**	**3**	**4**	**5**

7.2 Előzetes tanulmányokat felmérő kérdőív

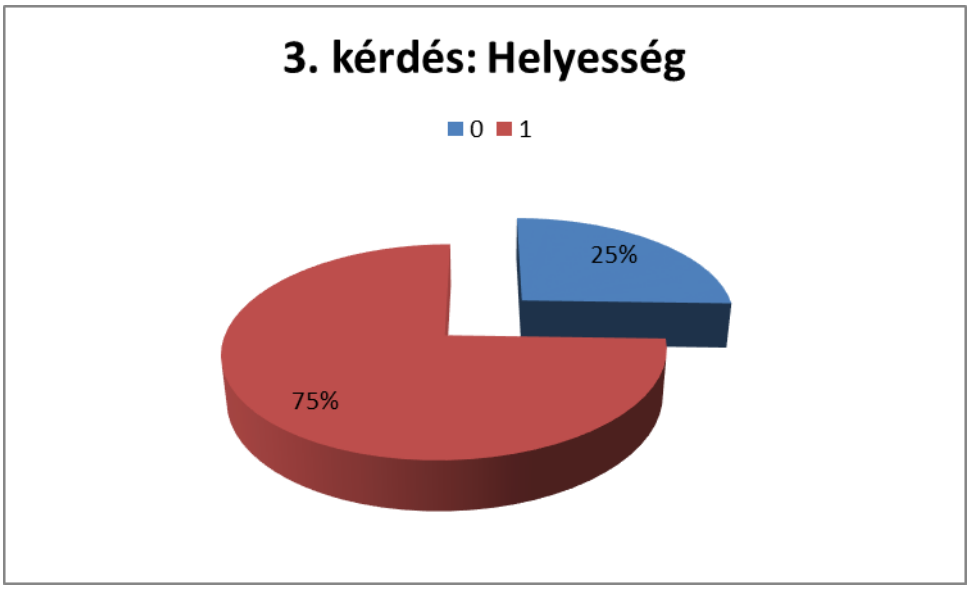
1. Heti hány órában tanultál informatikát középiskolában?
   - 0
   - 1
   - 2
   - 3
   - 4
2. Hány órát tanultál középiskolában programozást összesen?
   - 0
   - 1-4
   - 5-8
   - 9-14
   - 15 vagy annál több
3. Írj algoritmust (saját szavaiddal) arról, hogy hogyan mész át egy jelzőlámpás kereszteződésen!
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!

7.3 Az előzetes tanulmányokat felmérő kérdőív eredménye – összesített eredmény

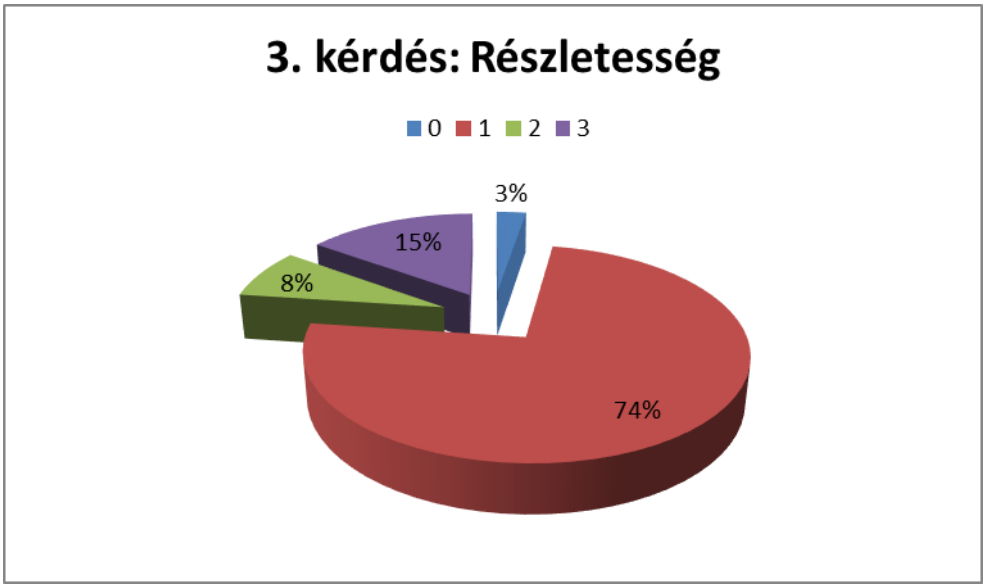


F1. ábra. *Informatikaórák eloszlása – összesített eredmény*

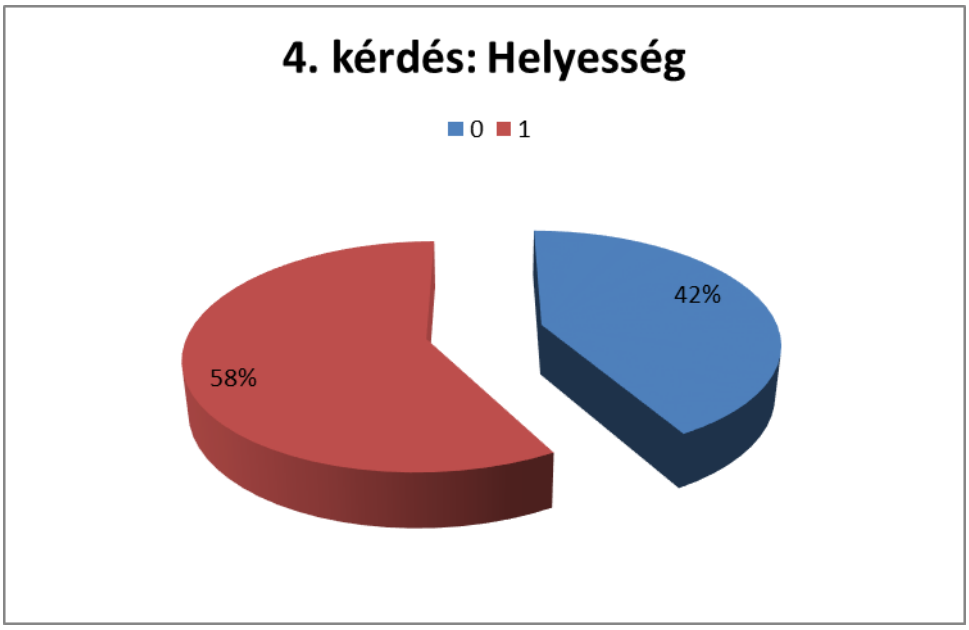
F2. ábra. *Programozásórák eloszlása – összesített eredmény*



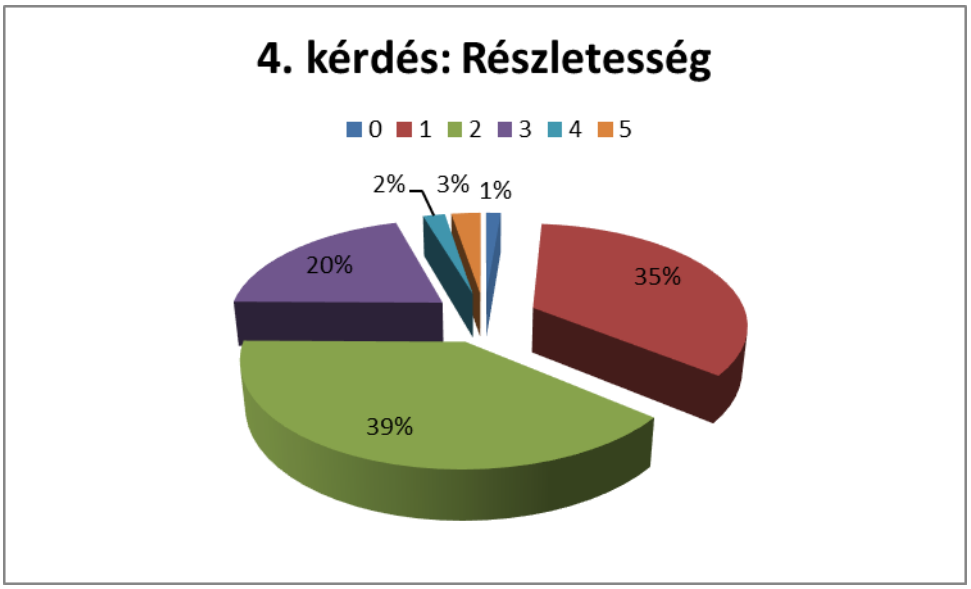
F3. ábra. *3. kérdés: Helyes-helytelen megoldások eloszlása – összesített eredmény*



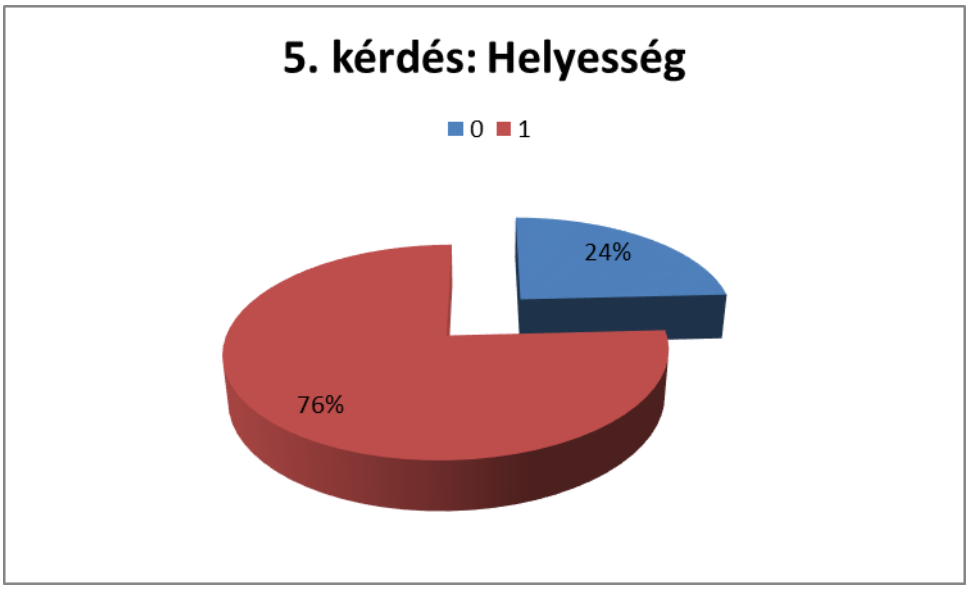
F4. ábra. *3. kérdés: Részletesség szerinti eloszlás – összesített eredmény*



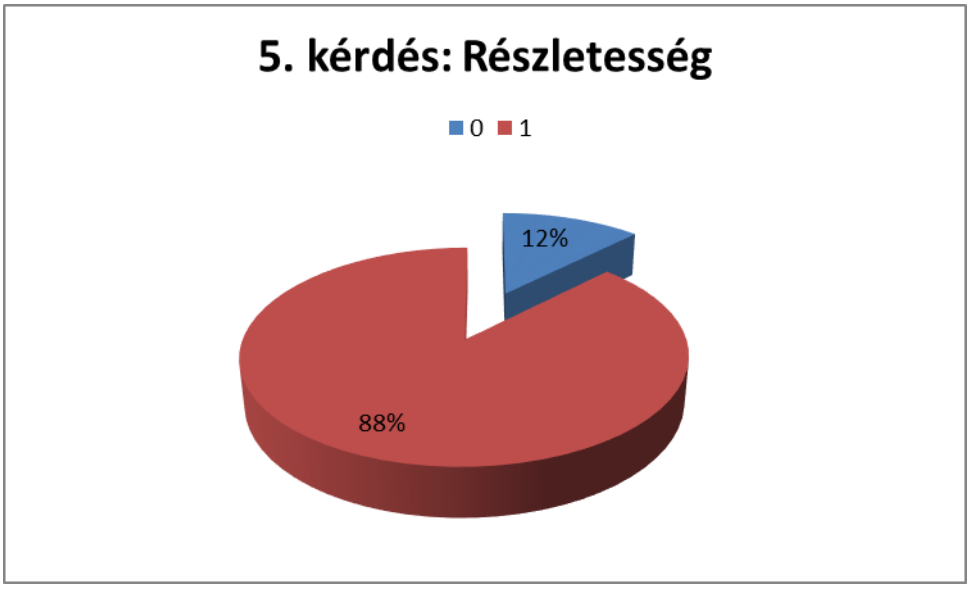
F5. ábra. *4. kérdés: Helyes-helytelen megoldások eloszlása – összesített eredmény*



F6. ábra. *4. kérdés: Részletesség szerinti eloszlás – összesített eredmény*

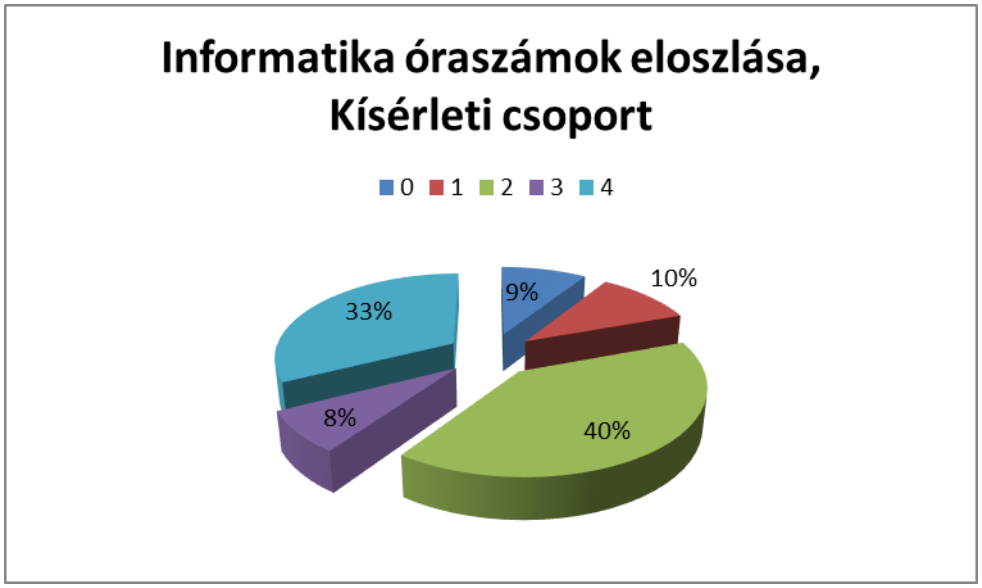


F7. ábra. *5. kérdés: Helyes-helytelen megoldások eloszlása – összesített eredmény*

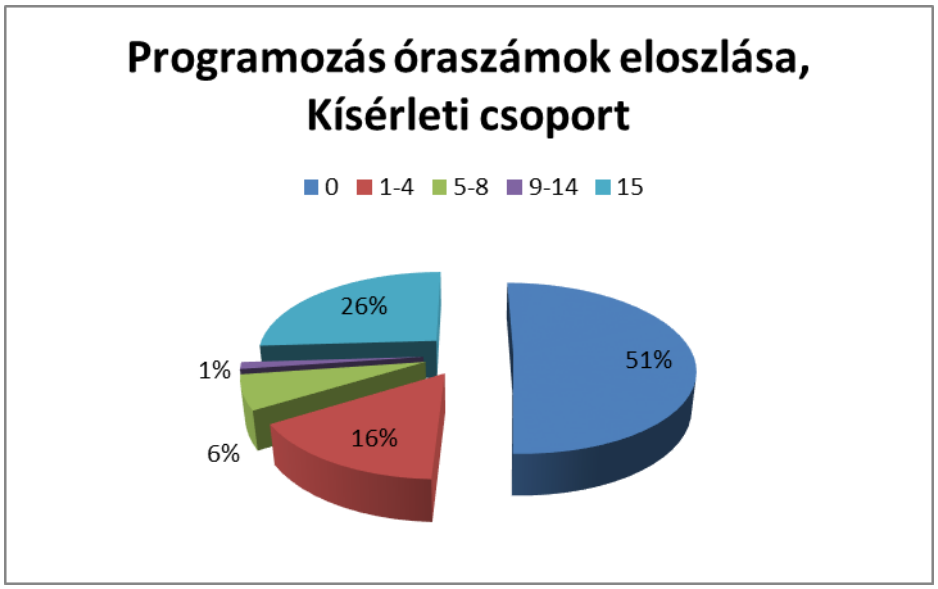


F8. ábra. *5. kérdés: Részletesség szerinti eloszlás – összesített eredmény*

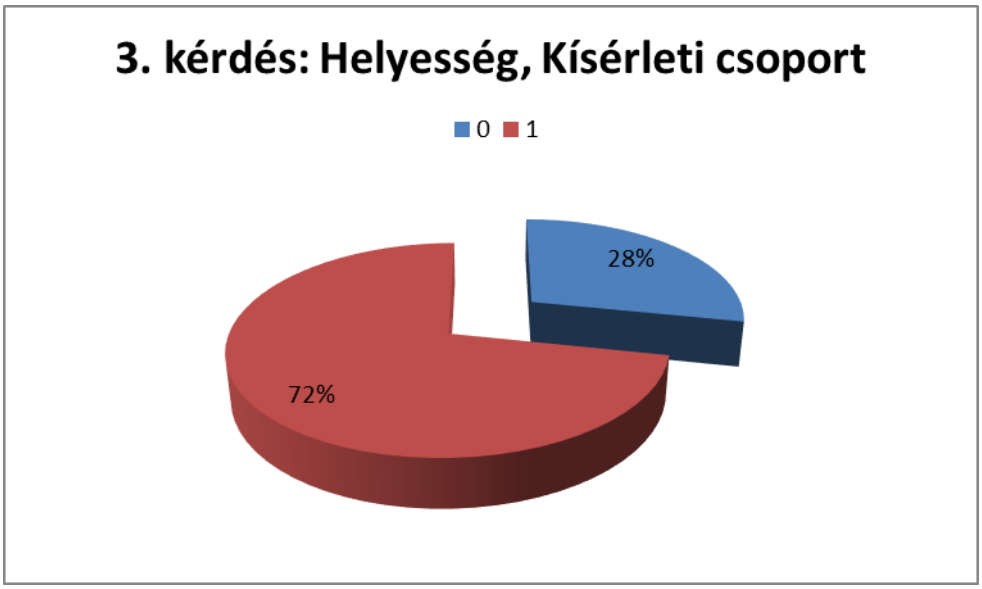
7.4 Az előzetes tanulmányokat felmérő kérdőív eredménye – kísérleti csoport



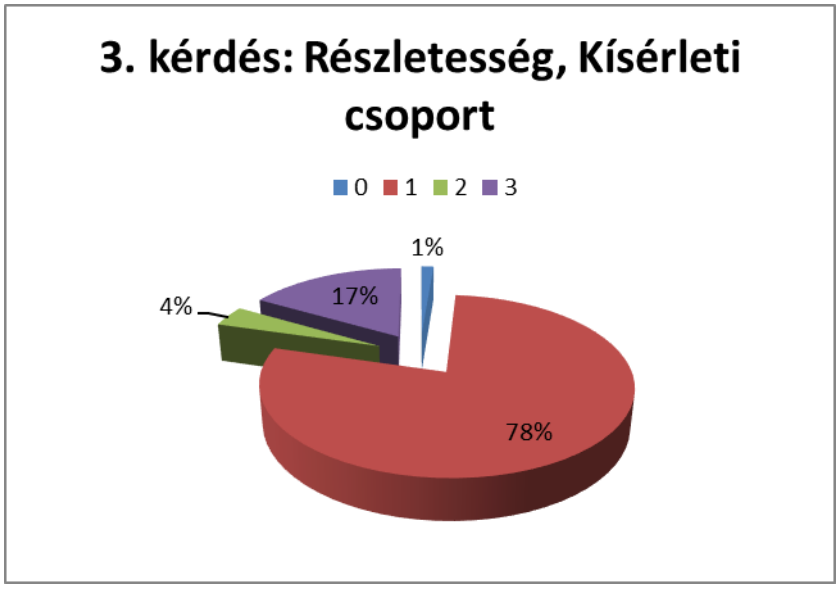
F9. ábra. *Informatikaórák eloszlása – kísérleti csoport eredménye*



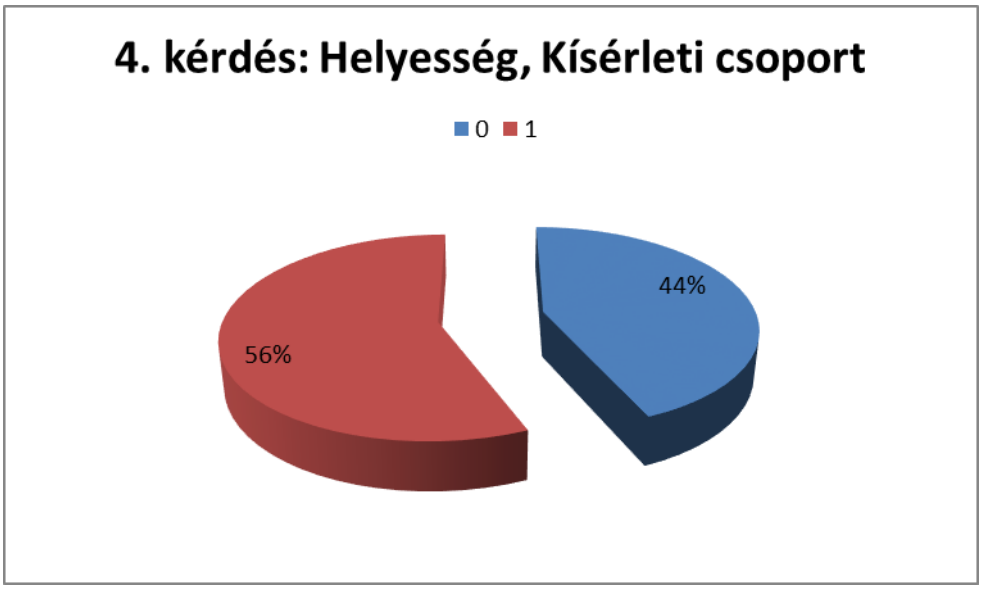
F10. ábra. *Programozásórák eloszlása – kísérleti csoport eredménye*



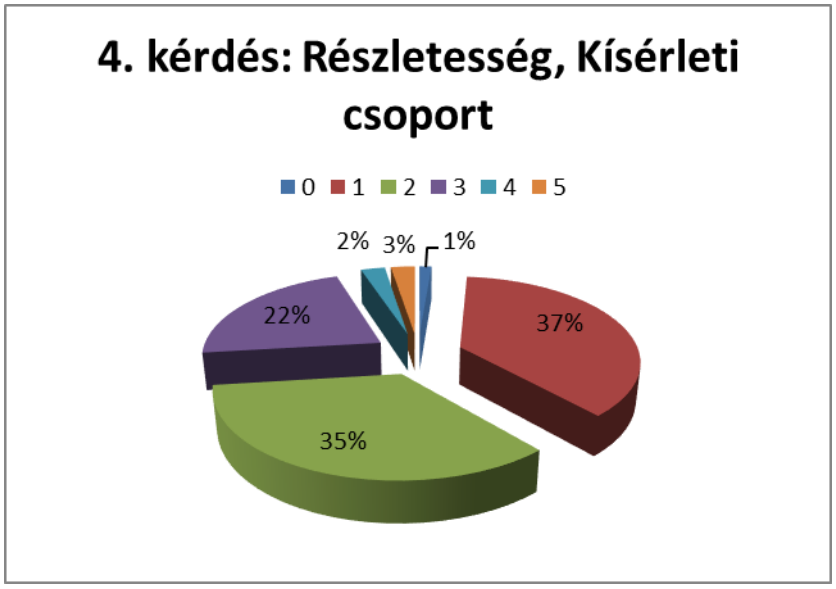
F11. ábra. *3. kérdés: Helyes-helytelen megoldások eloszlása – kísérleti csoport*



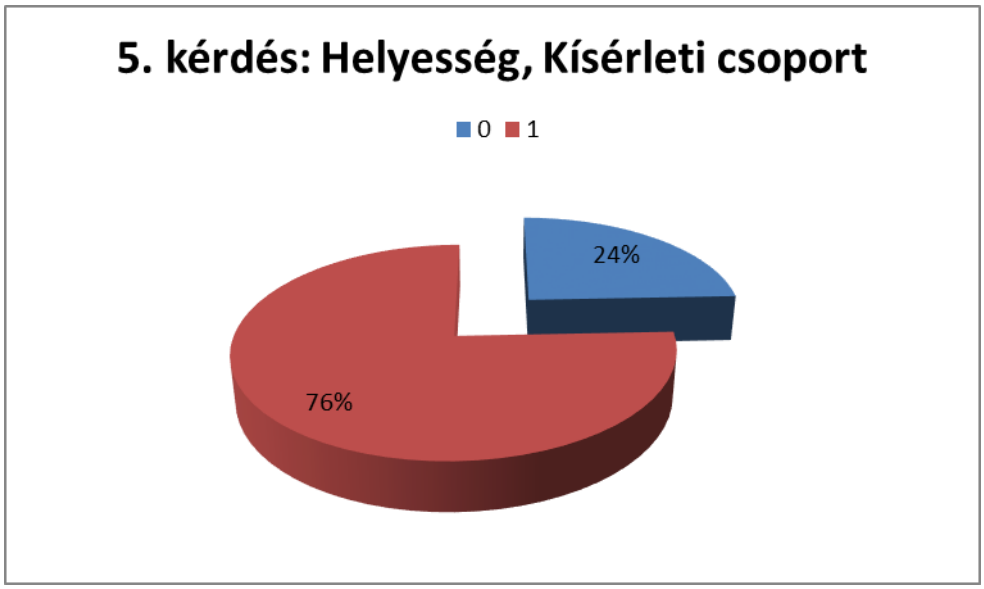
F12. ábra. *3. kérdés: Részletesség szerinti eloszlás – kísérleti csoport*



F13. ábra. *4. kérdés: Helyes-helytelen megoldások eloszlása – kísérleti csoport*



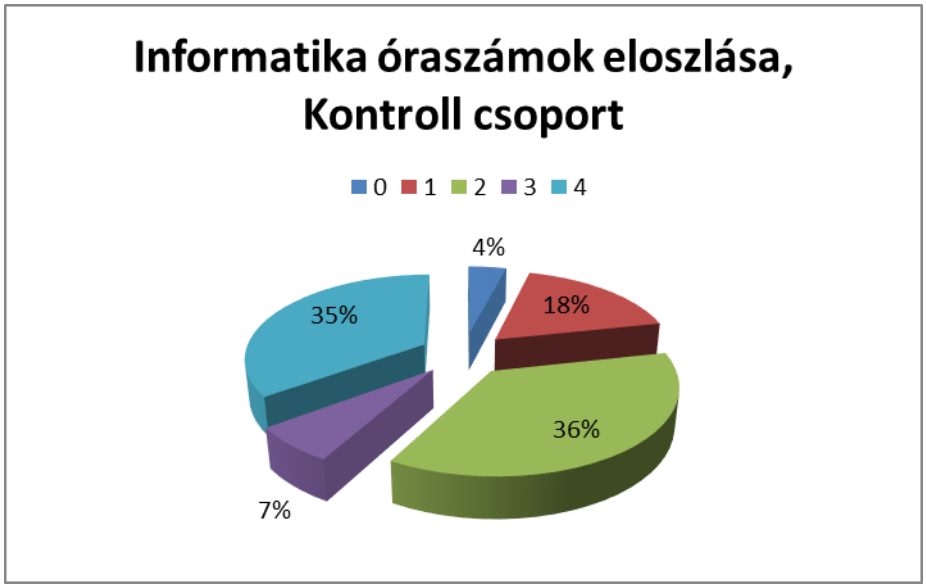
F14. ábra. *4. kérdés: Részletesség szerinti eloszlás – kísérleti csoport*



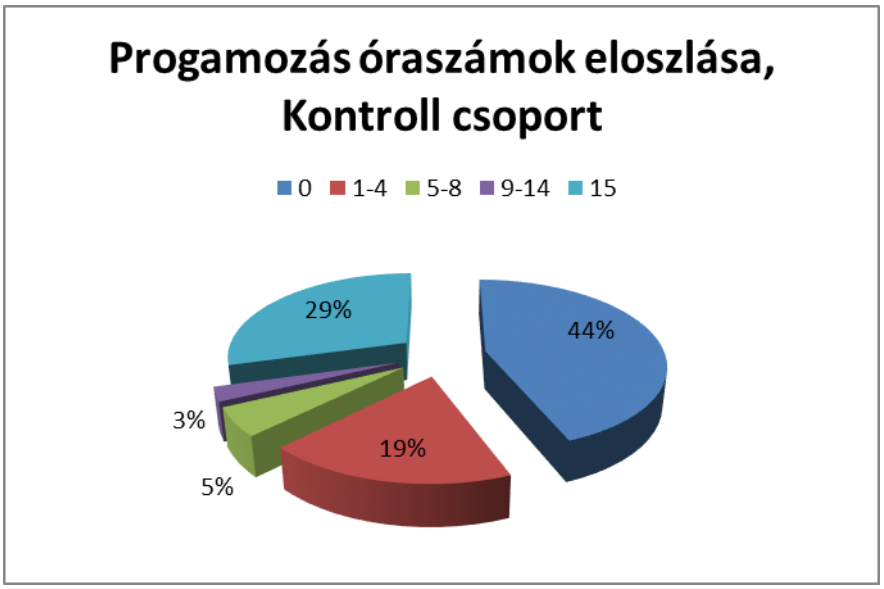
F15. ábra. *5. kérdés: Helyes-helytelen megoldások eloszlása – kísérleti csoport*

F16. ábra. *5. kérdés: Részletesség szerinti eloszlás – kísérleti csoport*

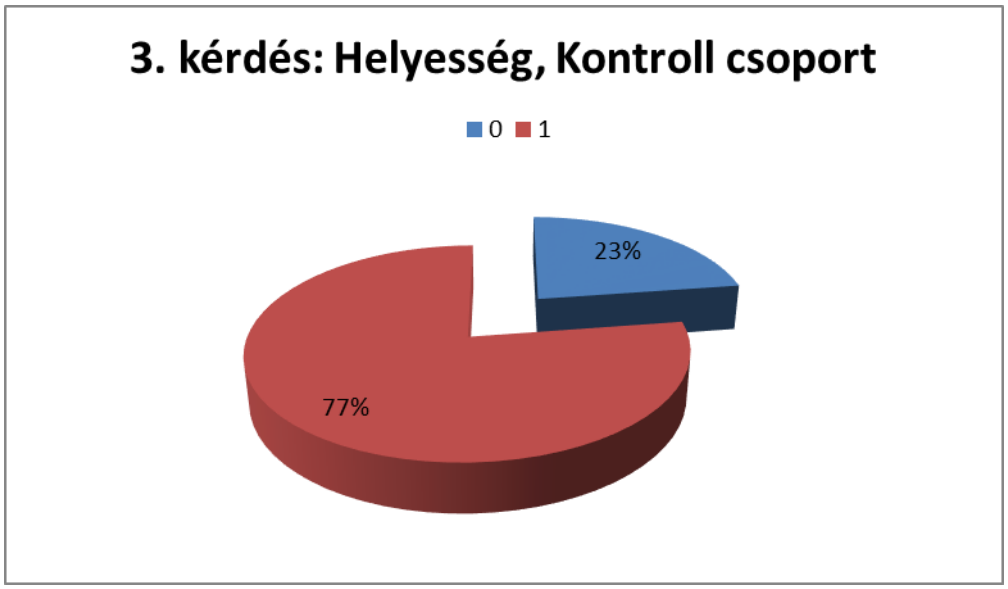
7.5 Az előzetes tanulmányokat felmérő kérdőív eredménye – kontroll csoport



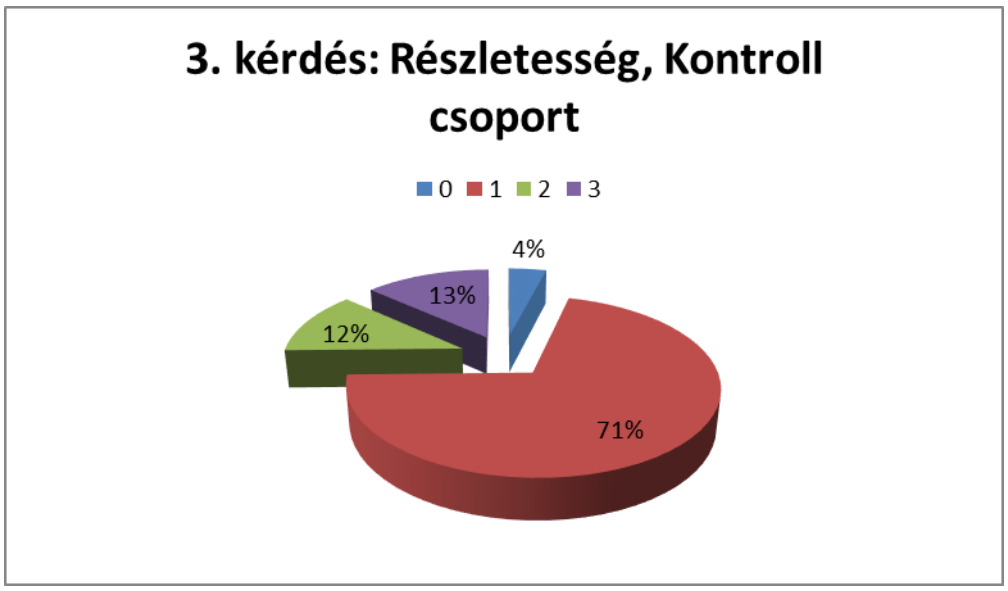
F17. ábra. *Informatikaórák eloszlása – kontroll csoport eredménye*



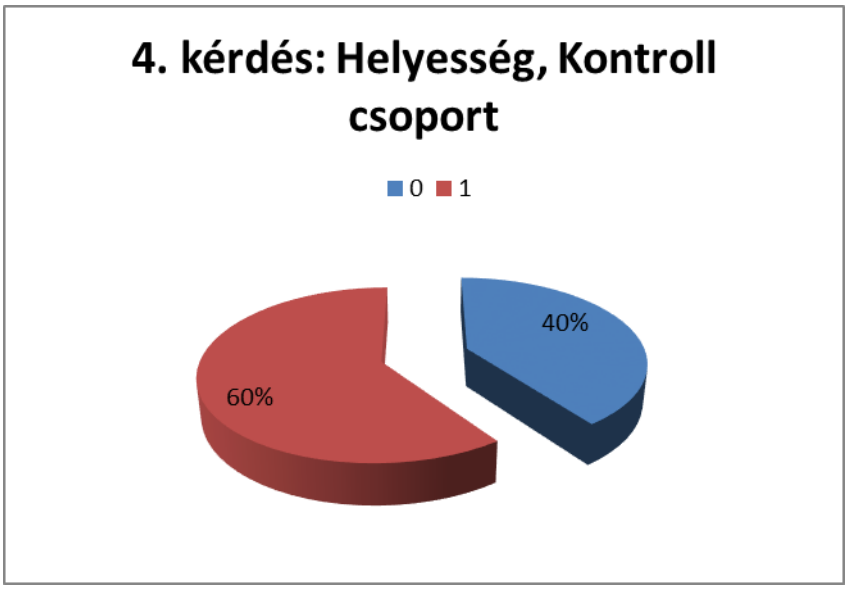
F18. ábra. *Programozásórák eloszlása – kontroll csoport eredménye*



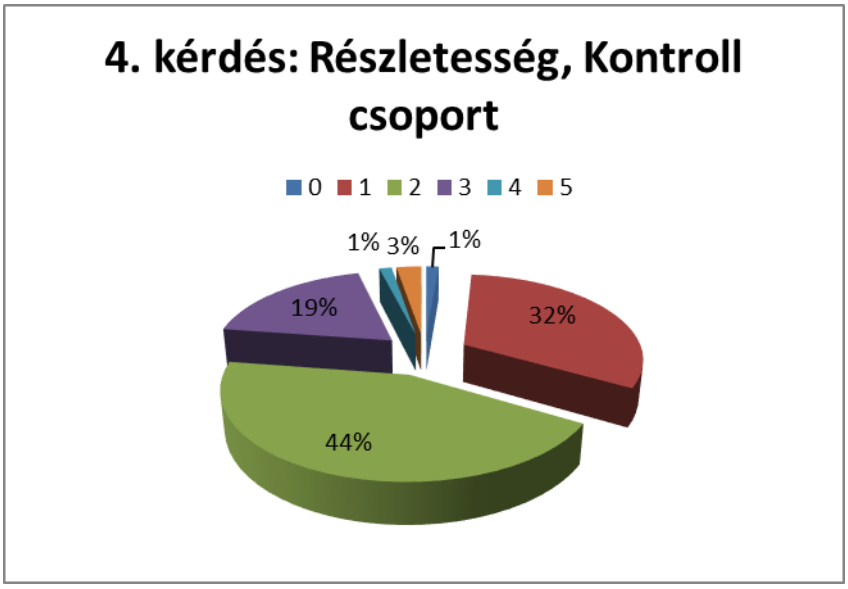
F19. ábra. *3. kérdés: Helyes-helytelen megoldások eloszlása – kontroll csoport*



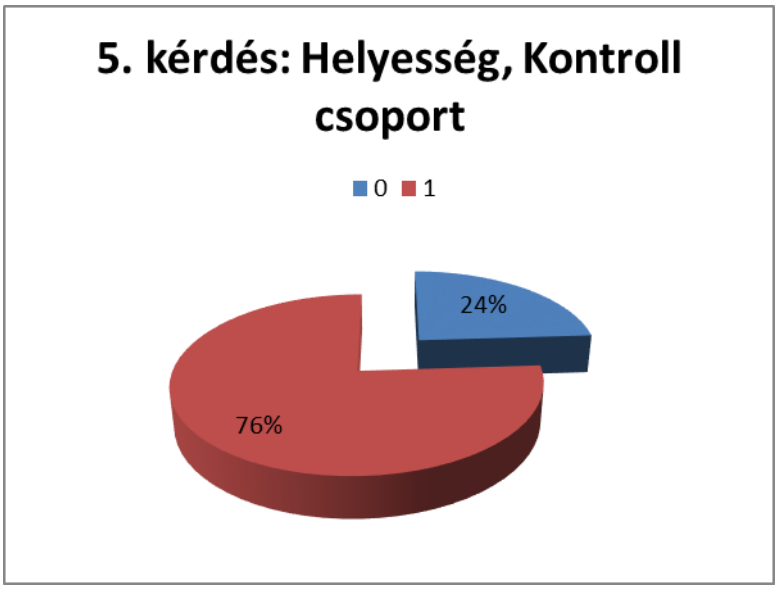
F20. ábra. *3. kérdés: Részletesség szerinti eloszlás – kontroll csoport*



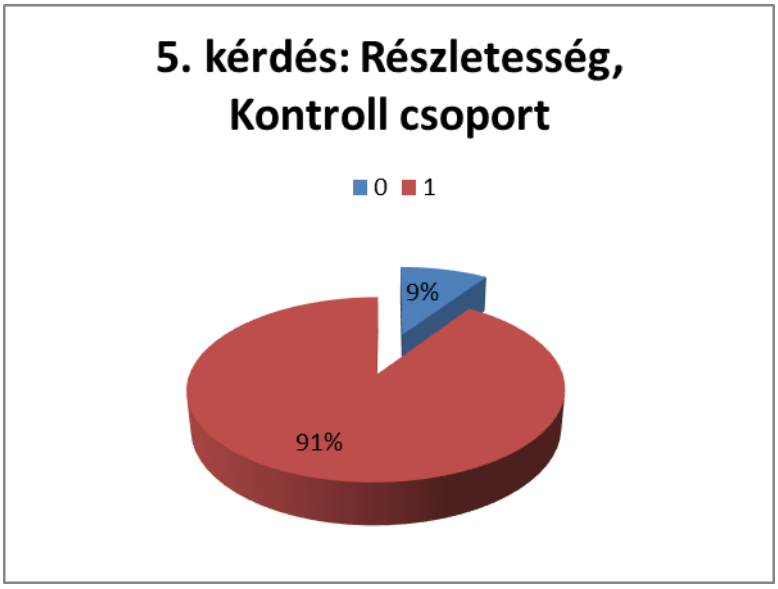
F21. ábra. *4. kérdés: Helyes-helytelen megoldások eloszlása – kontroll csoport*



F22. ábra. *4. kérdés: Részletesség szerinti eloszlás – kontroll csoport*



F23. ábra. *5. kérdés: Helyes-helytelen megoldások eloszlása – kontroll csoport*



F24. ábra. *5. kérdés: Részletesség szerinti eloszlás – kontroll csoport*

7.6 Kurzus eredményei – kísérleti csoport

Átlagok	Kísérleti csoportok					
#	11	12	13	14	18	Átlag
Gyak. jegy átlagok:	3,20	3,94	4,43	4,41	3,87	3,97
Sikeres gyak. jegy átlagok:	3,54	3,94	4,43	4,63	3,87	4,09
Számonkérések átlagai átlaga:	2,96	3,81	3,76	4,20	3,08	3,58
Min. 2-es számonkérések átlaga	3,72	4,14	4,36	4,60	3,78	4,15

Teljesítés (létszámok)	Kísérleti csoportok					
#	11	12	13	14	18	Összesen
Kezdetben	18	19	17	18	16	88
1Zh-n	17	18	15	16	16	82
2Zh-n	16	18	15	16	16	81
ÉvfZh-n	15	17	14	16	15	77
Végén	13	17	14	16	15	75

7.7 Kurzus eredményei – kontroll csoport

Átlagok	Kontroll csoportok					
#	8	15	25	29	30	Átlag
Gyak. jegy átlagok:	3,67	3,30	2,92	2,80	3,36	3,21
Sikeres gyak. jegy átlagok:	4,08	3,88	3,50	4,38	4,30	4,02
Számonkérések átlagai átlaga:	3,00	2,55	2,46	2,53	2,66	2,65
Min. 2-es számonkérések átlaga	3,92	3,52	3,33	4,25	4,15	3,84

Teljesítés (létszámok)	Kontroll csoportok					
#	8	15	25	29	30	Összesen
Kezdetben	19	13	16	17	19	84
1Zh-n	19	10	14	14	19	76
2Zh-n	19	10	13	16	15	73
ÉvfZh-n	13	8	11	17	10	59
Végén	13	8	10	9	10	53

7.8 C++-Jeliot „szótár”

Feladat	C++	Jeliot
Alapvető típusok deklarálása	`int` *változónév;*	
	`double` *változónév;*	
	`bool` *változónév;*	`boolean` *változónév;*
	`char` *változónév;*	
	`string` *változónév;*	`String` *változónév;*
Karakterlánc (string) típusú változó (a példában str) mérete	`str.size()` `str.length()`	`str.length()`
Karakterlánc (string) típusú változó (a példában str) i. karaktere	`str[i]` `str.at(i)`	`str.charAt(i)`

<table>
<tr><th>Feladat</th><th>C++</th><th>Jeliot</th></tr>
<tr><td>Karakterlánc (string) típusú változó (a példában str) i. karakterével kezdődő és a j.</td><td>str.substr(i,j)</td><td>str.substring(i,j)</td></tr>
<tr><td>Értékadás</td><td colspan="2">változó = kifejezés;</td></tr>
<tr><td>1-gyel való növelés</td><td colspan="2">i++, ++i</td></tr>
<tr><td>1-gyel való csökkentés</td><td colspan="2">i--, --i</td></tr>
<tr><td>Elágazás</td><td colspan="2"><pre>
if (feltétel)
{
   utasítások_1;
}
else
{
   utasítások_2;
}
</pre></td></tr>
<tr><td>Többirányú elágazás (if-fel)</td><td colspan="2"><pre>
if (feltétel_1)
{
   utasítások_1;
}
else if (feltétel_2)
{
   utasítások_2;
}
else if (feltétel_3)
{
   utasítások_3;
}
...
else if (feltétel_n)
{
   utasítások_n;
}
else
{
   utasítások_n+1;
}
</pre></td></tr>
<tr><td>Többirányú elágazás (switch-csel)</td><td colspan="2"><pre>
switch (kifejezés)
{
   case konstans1:
                {
                    utasítások_1;
                    break;
                }
   case konstans2:
                {
                    utasítások_2;
                    break;
                }
   case konstans3:
                {
                    utasítások_3;
                    break;
                }
   default:
                {
                    utasítások;
                    break;
                }
}
</pre></td></tr>
</table>

Feladat	C++	Jeliot

Elöltesztelős ciklus	`while (ciklusfeltétel)`
	`{`
	`  ciklusmag_utasításai;`
	`}`	

Hátultesztelős ciklus	`do`
	`{`
	`  ciklusmag_utasításai;`
	`} while (ciklusfeltétel)`	

Számlálós ciklus (a példában n-szer fut le)	`for (int i = 0; i < n; ++i)`
	`{`
	`  ciklusmag_utasításai;`
	`}`	

Egész szám beolvasása	`int egesz;` `cin >> egesz;`	`int egesz;` `egesz=Input.readInt();`
Valós szám beolvasása	`double valos;` `cin >> valos;`	`double valos;` `valos=Input.readDouble();`
Karakter beolvasása	`char karakter;` `cin >> karakter;`	`char karakter;` `karakter=Input.readChar();`
Karakterlánc beolvasása	`string szoveg;` `cin >> szoveg;`	`String szoveg;` `szoveg=Input.readString();`
Változó kiíratása, sortöréssel	`cout << szam << endl;`	`Output.println(szam);`
Szöveg és változó kiíratása, sortöréssel	`cout << "Az eredmény: " << szam << endl;`	`Output.println("Az eredmény:" + szam);`

Tömb deklarálása (egész számokat tartalmaz), rögzített elemszámmal	`int n;`
	`cin >> n;`
	`int tomb[n];`
		`int n;`
		`n = Input.readInt();`
		`int [] tomb  = new int [n];`

Tömb i. elemének elérése	`tomb[i]`	

Mátrix deklarálása (egész számokat tartalmaz), rögzített elemszámmal	`int n,m;`
	`cin >> n;`
	`cin >> m;`
	`int matrix[n][m];`
		Nem lehetséges!

Mátrix i. sorában és j. oszlopában levő elem elérése	`matrix[i][j]`	**Nem lehetséges!**

Rekord deklarálása	`struct Ember`
	`  {`
	`  string Nev;`
	`  char Nem;`
	`  int Kor;`
	`  ...`
	`  }`
		Nem lehetséges!

Rekord elérése	`Ember Jozsi;`
	`Jozsi.Nev = Kovacs Jozsef;`
	`Jozsi.Nem = 'F';`
	`Jozsi.Kor = 32;`
		Nem lehetséges!

Vizualizáció a programozásoktatásban c. értekezés összefoglalása

Dolgozatom témája az algoritmus vizualizáció (AV), mint tanítási/tanulási eszköz programozás oktatásának a módszertana.

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

Három AV eszközt mutattam be, majd az AV közoktatási informatikában való alkalmazhatóságát vizsgáltam, ahol konkrét ajánlást tettem, óraszámokkal. Ilyen módon igazoltam az első tézisemet: ***Az AV beilleszthető a közoktatási informatikába.***

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 megoldottak 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, tehát a vizsgálat igazolta a második és harmadik tézist: ***Az AV eszközzel való tanítás az átlag közeli tudású hallgatók felzárkózását segíti*** és ***az AV-t használó hallgatók többet fejlődnek és jobb eredményeket érhetnek el, mint a vizualizációt nem használók.***

A negyedik tézist ***(Az AV használata jobban fejleszti az absztrakciós készséget, mint ha nem kerülne használatra ez az eszköz)*** a bemeneti kérdőív 5. kérdésének és a 2. csoportzárthelyi dolgozatnak az eredményei bizonyították, ugyanis a két feladat az absztrakciós készség szintjét méri és a kísérleti csoport hallgatói jobb eredményeket értek el a 2. csoportzárthelyi dolgozaton, mint a kontroll csoport.

Summary of the dissertation Visualization in teaching programming

The topic of my paper is the programming teaching methodology of algorithm visualization (AV) as a teaching/learning tool.

I introduced the history of AV from the 70's to nowadays, and I highlighted the development of the AV's role in programming education.

I presented three AV tools, and I examined the AV's application in the Hungarian informatics education. I did a recommendation about a curricula and numbers of classes.

I joined the Jeliot 3 AV system to the syllabus of Basics of programming course. I highlighted that in which part of the syllabus does support this AV system the students' learning and understanding. That is why the first thesis was proved: ***AV can be included to informatics in secondary education.***

In the autumn of 2012, I performed a combined, two-group pedagogical experiment at ELTE Faculty of Informatics, where I investigate the work of 1st grader students of informatics in the Basics of programming course. 5 seminar groups took part in the experimental group and 5 seminar groups in the control group.

On the first class, students filled a questionnaire about their prior knowledge. Two questions were about how much classes they had in secondary school on informatics and programming. Three other questions contained simple algorithmic tasks, with which I defined the students' level of algorithmic thinking. This input test gave the basis for the definition of how much progress reached the students from the beginning of the semester.

The experimental group performed better, according to the measure of the progress and the final results of the course. This was especially true to the students with nearly average ability. That is why, it was proved that it is worthy to use AV tools in teaching programming, especially for novice students. So the experiment proved the second and third thesis***: Teaching with AV tool supports the catching up of students with nearly average knowledge*** and ***Students who used AV improve more and reach better results than the students who do not use AV.***

The fourth thesis ***(Use of AV improves the abstraction ability better than if this tool were not used)*** was proved by the results of the question no. 5 from the input questionnaire and the seminar group test no. 2, because this two tasks measures level of the abstraction ability of the students, and the experimental group reached better results on the seminar group test no. 2 than the control group.