Objektum orientált programozás tanítása vizualizációs eszközökkel

Törley Gábor
Torley.Gabor@uni-nke.hu
Nemzeti Közszolgálati Egyetem, Közigazgatás-tudományi Kar,
E-közszolgálati Fejlesztési Intézet

Absztrakt. Ez a cikk bemutatja az algoritmus-vizualizációt, mint segédeszközt az objektum orientált programozás tanítása során. Az algoritmus-vizualizáció elméleti bevezetése és néhány oktatási vonatkozású eredmény közlése után, az írás példát mutat két jól használható vizualizációs eszközre: a BlueJ-re és a Jeliotra, végül értékeli azokat.

Kulcsszavak: programozás, oktatás, algoritmus vizualizáció, OOP, objektum orientált

Az objektum orientált programozás (OOP) az elmúlt évtizedekben a legbefolyásosabb 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 dolog 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 képes való világot leképezni, hiszen a körülöttünk levő „dolgok” is leírhatók tulajdonságokkal, állapotokkal és metódusokkal, ilyen módon az objektum orientált paradigma fejleszti a gondolkodás „térlátását” és az absztrahálás képességét

Röviden összefoglalva, az OOP egy jó eszköz a fenti módszerek megtanítására. Mégis, az objektum orientált programozást nehéz tanítani, egyrészt az absztrakt gondolkodás szükségessége miatt, illetve azért is, mert az objektum, amit leprogramozunk, gyakorlatilag csak a programozási folyamat végére konkretizálódik.

Tanítási gyakorlatom alatt programozási tételeket tanítottam egy budapesti szakközépiskolában. Akkor egy prezentációba ágyazott animáción keresztül mutattam meg a diákoknak az adott tétel működését, tehát a diákok látták az animációt és hallották a magyarázatot. Sokkal többet lehetett így átadni, mint ha csak az algoritmus szövegét tanulmányoztuk volna végig a tanulókkal. Ezt a hatást hívja Mayer „Multimédia hatásnak” a Multimédia tanulás kognitív elméletében (*Cognitive Theory of Multimedia Learning*), amely öt alapelvet fogalmaz meg [1]:

- 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: Pl. összegzésnél 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. [2] Tapasztalataimból azt a következtetést vontam le, hogy ha olyan eszközöket használok a tanítás alatt, amelyek segítenek *elképzelni* az algoritmus belsejében történteket, akkor meggyorsul a megértés folyamata. Az algoritmus-vizualizációs (AV) eszközök használata ebben nyújt segítséget.

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

Kölling szerint [3] néhány hallgatónak azért nehéz megérteni az OOP terminológiáját, mert mindaz, amit a képernyőn és a tanár prezentációjában látnak, az maga a programkód. Ha elvárjuk a hallgatótól, hogy gondolkodni és beszélni tudjon az osztályokról, a kapcsolataikról, akkor vizuálisan kell azokat megjeleníteni. Ugyanez az állítás igaz, amikor a program futási idejében vizsgáljuk az objektumokat. Vizuális reprezentációval könnyebben meg lehet érteni az OOP terminológiáját.

1. Algoritmus-vizualizáció

Az algoritmus-vizualizáció a program-vizualizáció alosztályaként számítógépes algoritmusok magas szintű működésének illusztrálásával foglalkozik, általában abból a célból, hogy a programozást tanulók jobban megértsék az algoritmus eljárásainak működését. [4]

Az AV a múlt század 70-es éveinek végén a batch-oritentált szoftverekből – amelyek lehetővé tették az oktatóknak animációs filmek készítését [5] – mára magas szintű interakcióval rendelkező rendszerekké fejlődött, amelyekkel a tanulók felfedezhetik, beállíthatják, dinamikusan megváltoztathatják az algoritmus animációját a saját igényeiknek megfelelően [6, 7] vagy maguk képesek megalkotni saját vizualizációjukat [8, 9]. Az AV program segítette

- az oktatót az algoritmus illusztrálásában, [6]
- a tanulókat abban, hogy megértsék az alapvető algoritmusok működését, [10]
- a hibakeresést konzultáción, [11]
- a tanulókat megérteni egy absztrakt adattípus műveleteinek működését. [12]

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. [13]

Kehoe és munkatársai tanulmányukban [14] 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 a binomiális kupacról tanult, de 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 megmutassa egy algoritmus

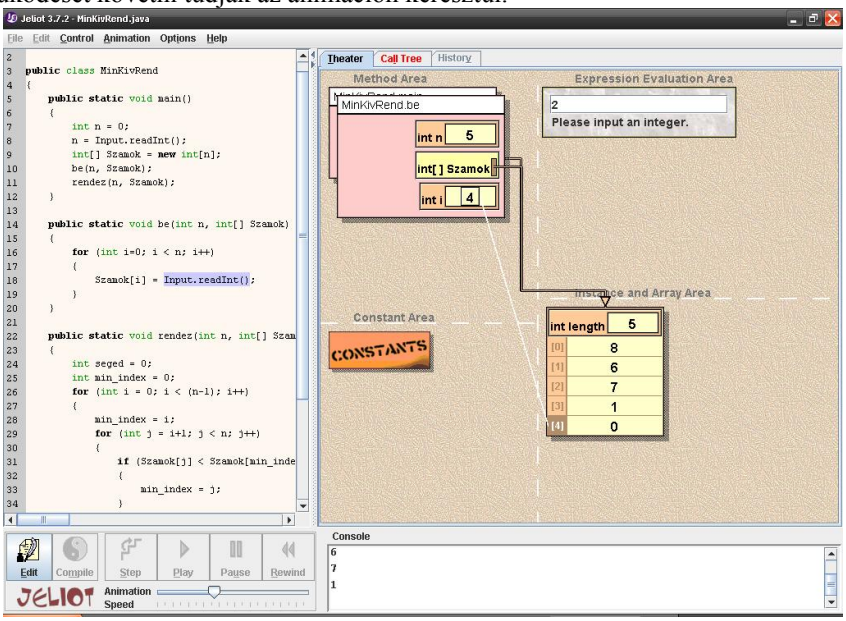
műveleteit lépésről lépésre, így világos vizuális reprezentációt nyújt egy amúgy absztrakt folyamatról.

Az algoritmus-vizualizációs eszközök használata Magyarországon szinte ismeretlen. A Google keresője szerint mindössze csak egy magyar nyelvű és vonatkozású írás [15] jelent meg ebben a témában, holott, nemzetközi kutatások sok pozitív eredményt mutattak fel az elmúlt 40 évben.

2. Jeliot 3 és BlueJ

A Jeliot 3 egy ingyenes, Java alapú AV rendszer[1]. 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. 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 (lásd az 1. ábrát). A tanulóknak lehetőségük van programozni a Jeliot 3-ban, és később annak működését követni tudják az animáción keresztül.



1. ábra. Jeliot

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

[1] http://cs.joensuu.fi/jeliot – Letöltve 2012. október 27.

egyszerre látható. A környezet képes a folyamatos és a lépésenkénti animációra, sőt a tanuló ki tudja jelölni azt a részt, amit animálva akar látni.

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.

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 [14] több időpontban mérték a tanulók teljesítményét. Kiderült, hogy a rendszer támogatja a frontális előadást, segít megérteni a tanár magyarázatait. A vizsgálat kimutatta, hogy a közepes képességű tanulók fejlődése szignifikáns volt, az övék volt a legmarkánsabb. Az előrehaladásuk különösen abban mutatkozott meg, ahogyan definiálták és elmagyarázták a fogalmakat.

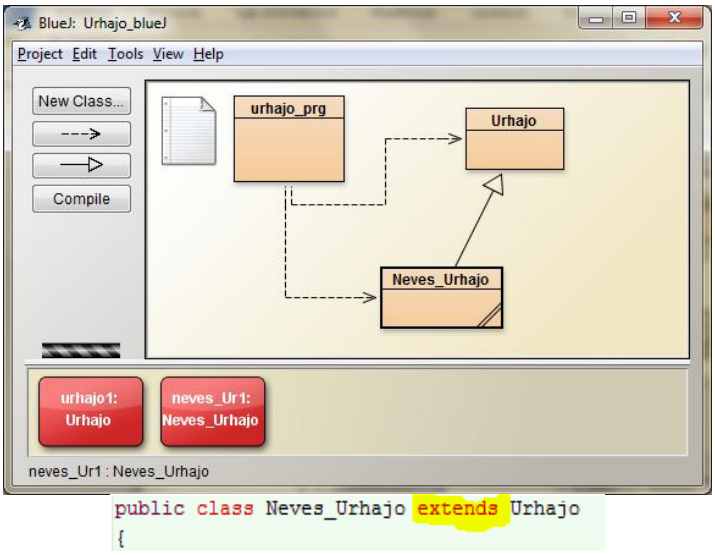
A tanulási folyamat a figyelemmel kezdődik [16]. Erre alapozva végzett Ebel és Ben-Ari vizsgálatokat. [17] A vizsgálat a tanulók figyelmére összpontosított, ugyanis a figyelem mértéke jól korrelál a tanulás hatékonyságával. Magatartás- és figyelemzavaros gyerekeket tanítottak a Jeliot 3 segítségével, és azt tapasztalták, hogy az órák azon részén, amikor a Jeliottal tanította, magyarázta a tanár az órai anyagot, nem volt magatartás- vagy figyelemzavar a tanulók részéről.

Moreno és Joy vizsgálata [18] 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.

A BlueJ[2] 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.

[2] www.bluej.org - Letöltve 2012. október 27.



2. ábra. BlueJ

A környezet lehetővé teszi azt, hogy azok a kapcsolatok, amelyeket grafikusan megjelenítünk, a kódban is megjelenjenek, pl. a 2. ábrán látható öröklési kapcsolat az Űrhajó és a Neves_Űrhajó osztály között.

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

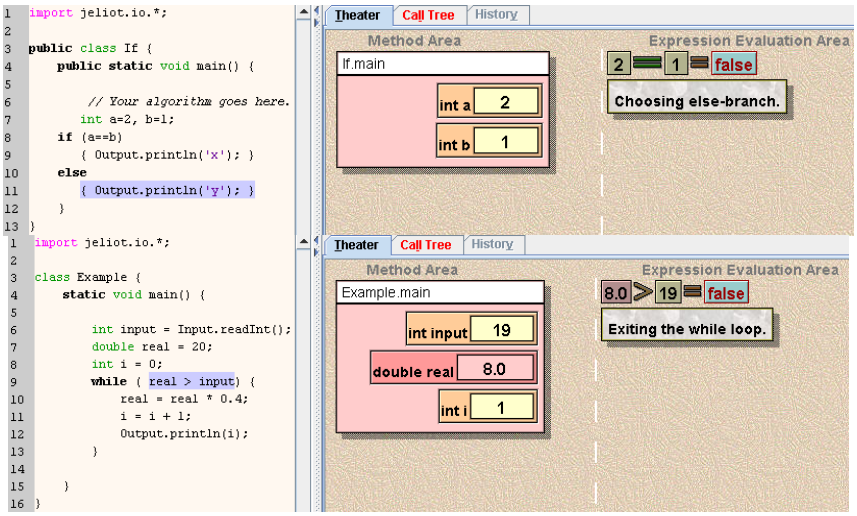
3. Néhány alkalmazási példa

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

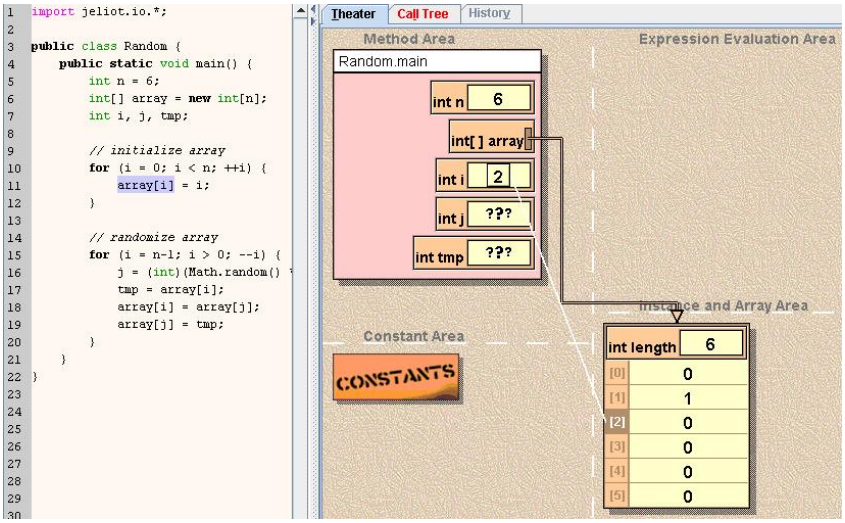
Nem jelentős hátrány a környezetek „nyelvfüggősége”, bár nyílván leginkább a Java nyelven tanulók tudják kihasználni a rendszer szolgáltatásait. A vezérlési szerkezetek, az algoritmus működésének, az osztályok metódusainak és a közöttük levő kapcsolatok megértése nem nyelvfüggő feladat.

A Jeliot 3 mindig megindokolja, miért az adott ágban halad tovább a program futása elágazás esetén, miért maradunk a ciklusban vagy lépünk bele, illetve ki belőle (lásd a 3. ábrát).

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



3. ábra. Vezérlési szerkezetek működésének reprezentálása Jeliotban



4. ábra. Tömb megjelenítése Jeliotban

Gyakran használt adatszerkezet a tömb, ennek belső állapotát is képes megjeleníteni a program (lásd a 4. ábrán).

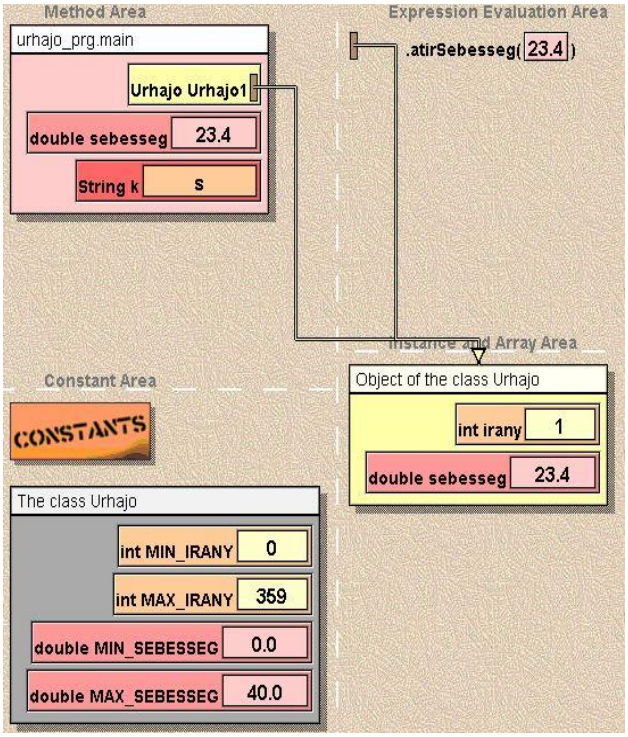
Tekintsünk egy egyszerű példát, az Űrhajó és a Neves Űrhajó osztály megvalósítását:

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.

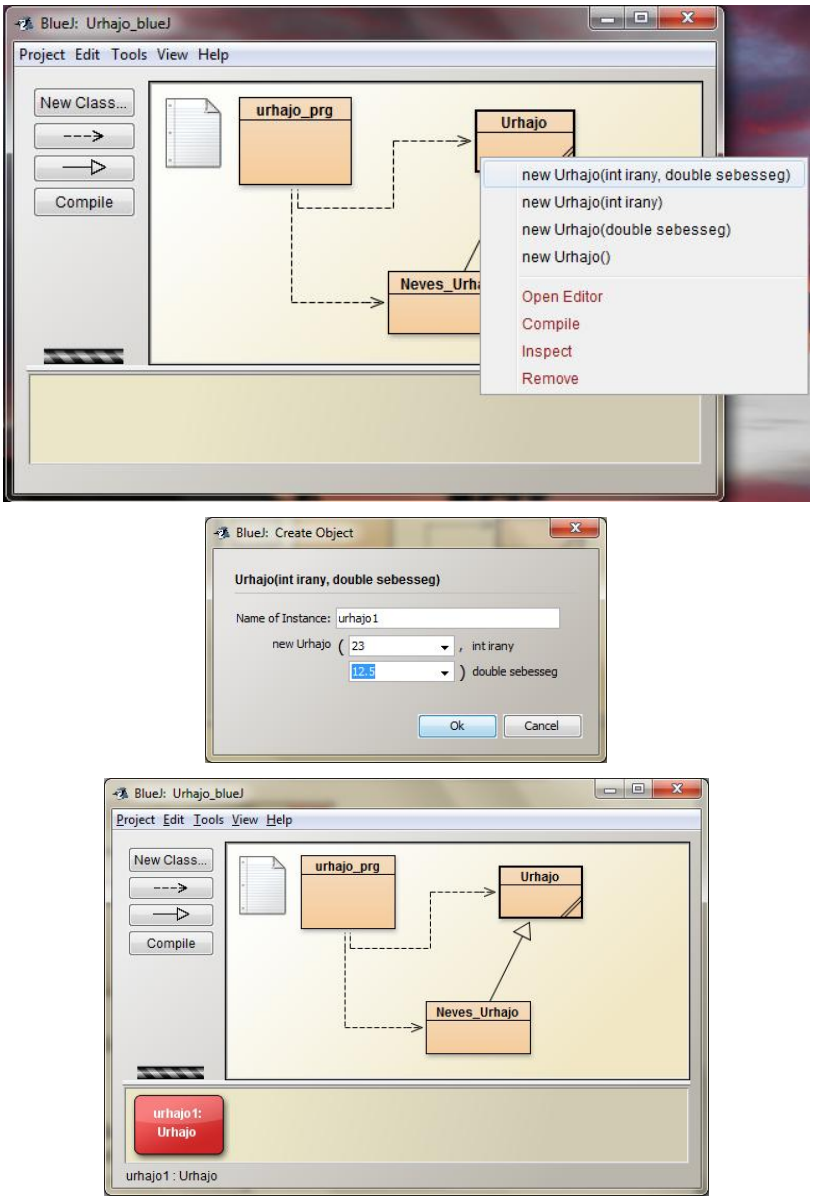
A Jeliot 3 alkalmas arra, hogy futási időben megmutassa az objektum aktuális állapotát (lásd az 5. ábrát).



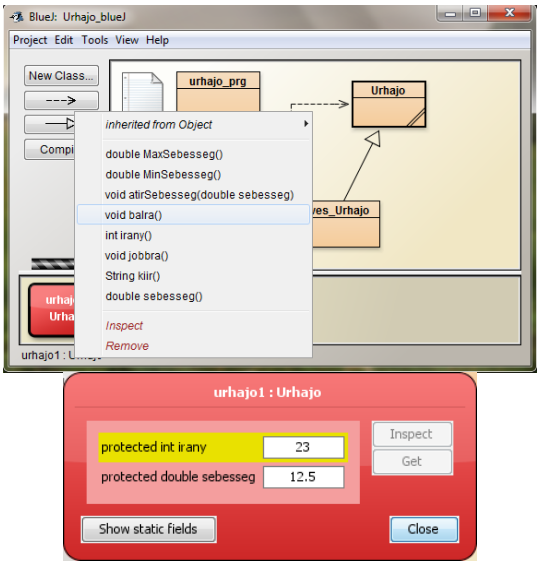
5. ábra. Az Űrhajó osztály objektumának állapota Jeliotban

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 (lásd a 6-7. 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.



6. ábra. BlueJ: objektum létrehozása

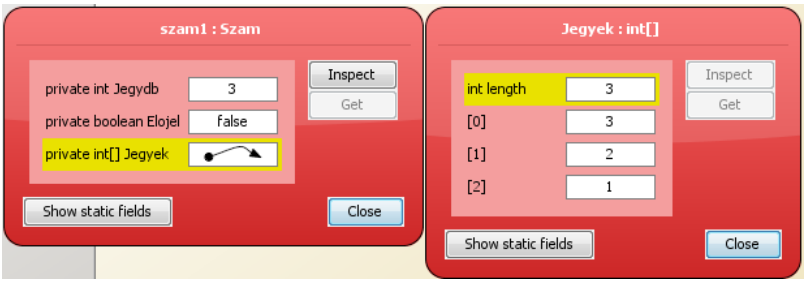


7. ábra. BlueJ: Objektum metódusa és állapota

A következő példa már bonyolultabb feladat megoldását tűzte ki célul: létre akarunk hozni egy új Szám nevű osztályt 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.

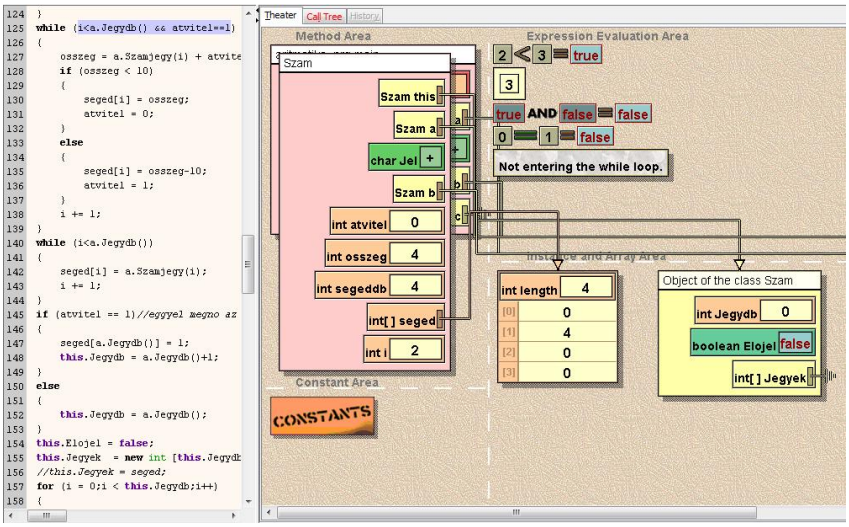
A nagypontosságú egész szám számjegyeit tömbben tároljuk. A Szám osztály tulajdonságai között fogjuk számon tartani továbbá a számjegyek számát, az előjelet és a számrendszer alapját. A megvalósítandó műveletek (metódusok): Összeadás, kivonás, szorzás, maradékos osztás, egész osztás, beolvasás, kiíratás, illetve az $=$, $\neq$, $<$, $>$, $\leq$, $\geq$ relációk. A 8. ábra mutatja az objektum megvalósítását.



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

Habár bonyolultabb ez a példa ez előzőnél, mégsem igényel speciális ismereteket. A hallgatóknak csak mélyebben bele kell gondolnia, hogy miképp működnek azok a matematikai műveletek, amit már általános iskola második osztálya óta használnak. 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).



9. ábra. Jeliot 3: Ciklus bennmaradási feltételének kiértékelése

A fenti ábra az összeadás művelet algoritmusának egy állapotát mutatja. 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 (lásd a 9. ábrán) 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. Talán egy kicsit zavaró. A környezettel történt korábbi vizsgálatok [19] alkalmával Turbo Pascalt használtak a tanulók, és az idézett tanulmány szerint az alapkoncepciókban, mint pl. értékadás, 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.

4. Összefoglalás

A Jeliot 3 és a BlueJ AV program rendelkezik azokkal a tulajdonságokkal, amelyek egy hatékonyan használható program-vizualizációs rendszert jellemez.

Mindkét program támogatja a *frontális oktatás*t, ahol a tanár szóbeli magyarázatait egészíti ki (mozgó) képi információkkal, illetve az *egyéni tanulás*nál programkód megértését segíti vizuális elemekkel, így jól valósítja meg Mayer elméletét. [2] A rendszerek támogatják a hibakeresést, illetve segítségükkel a hallgatók „egyben" látják az objektumokat.

A Jeliot 3-ban Lehetőség van *egy időben egyetlen* töréspontot a kódba helyezni, amelytől kezdődően fut az animáció, így lehetőség van csak a tanuló által kiválasztott részt animálni. A rendszer jól szemlélteti az adatszerkezetek belső működését, illetve az metódusok és azok paramétereit, lokális változóit.

A két program nagy előnye, hogy támogatja a tanárt, akár elméleti, akár gyakorlati oktatásról legyen szó, illetve a tanuló is hatékony eszközként tudja használni, mikor önállóan dolgozza fel a feladat adta problémát.

Arra a kérdésre, hogy milyen módszerekkel és elvek betartásával lehetséges az AV-t a magyar programozásoktatás gyakorlatába bevonni, további kutatások adhatják meg a választ. A külföldi eredmények alapján Hundhausen és munkatársai összefoglaló tanulmánya [4] szerint, az intuitív vonzereje ellenére az AV nem terjedt el pedagógiai eszközként az informatikaoktatásban. Az egyik fő ok az, hogy a tanárok nem tartják hatékonynak. Igazság szerint, valóban az eddig publikált eredmények nem nyújtanak egységes képet az AV hatékonyságáról, nem minden esetben bizonyosodott be, hogy szignifikánsan jobb tanulási eredmények születtek az AV használatával. A cikk több empirikus vizsgálat eredményét elemzi 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, amelyben AV-t használnak, 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. Tehát a tevékenység formája fontosabb, mint a vizualizáció formája.

A fenti elveket alkalmazva a tanárok és a tanulók kezébe olyan segédeszköz kerülhet, amely növeli a tanulás hatékonyságát és a diákok motivációját.

Irodalomjegyzék

1. Mayer, R. E. (1997). *Multimedia learning: Are we asking the right questions.* Educational Psychologist, 32, 1-19.
2. 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.
3. Kölling, M., "The Problem of Teaching Object-Oriented Programming, Part 2: Environments," *Journal of Object-Oriented Programming*, 11(9): 6-12, 1999.
4. C. Hundhausen, S. A. Douglas, and J. T. Stasko. *A meta-study of algorithm visualization effectiveness.* Journal of Visual Languages and Computing, 2002.
5. R. Baecker (1975) T*wo systems which produce animated representations of the execution of computer programs.* SIGCSE Bulletin 7, 158-167.
6. M. H. Brown (1988) *Algorithm Animation*. The MIT Press, Cambridge, MA.

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

8. J.T. Stasko (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.

9. C. D. Hundhausen (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.

10. P. Gloor (1998) *Animated algorithms*. In: SoftwareVisualization: Programming as a Multimedia Experience (M. Brown, J. Domingue, B. Price&J. Stasko, eds) TheMIT Press, Cambridge, MA, pp. 409-416.

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

12. T. Naps (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.

13. Ben-Bassat Levy, R., M. Ben-Ari and P. A. Uronen, *The Jeliot 2000 program animation system*, Computers & Education 40 (2002), pp. 1–15.

14. 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.

15. Törley Gábor: *Algoritmus-vizualizáció a programozás-oktatásban*, in. Alkalmazott Multimédia Vol. 4, No. 3., pp. 81-94., 2009., Apple Magyarországi Képviselet, Budapest

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

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

18. 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.

19. Ronit Ben-Bassat Levy, M. Ben-Ari, Pekka A. Uronen: *An Extended Experiment with Jeliot 2000.* First International Program Visualization Workshop, Porvoo, Finland, 2000.