Algoritmus-vizualizáció a programozásoktatásban

Törley Gábor
Budapesti Corvinus Egyetem, Közigazgatás-tudományi Kar, Közigazgatási Informatikai Tanszék
gabor.torley@uni-corvinus.hu

Absztrakt: Ez a cikk bemutatja az algoritmus-vizualizáció elméletét és eddig oktatási vonatkozású eredményeit, majd példát mutat egy jól használható algoritmus-vizualizációs eszközre, végül értékeli azt. A cikk bemutatja továbbá, hogyan lehet felhasználni oktatóként és tanulóként az algoritmus-vizualizációs eszközöket, konkrétan a programozási tételek tanítása és tanulása közben, illetve értékeli a tanulási és tanítási módszert. Két eszközt fog példaként felhozni: a Jeliot-ot és a TRAKLA2-t

Kulcsszavak: programozás, oktatás, algoritmus, vizualizáció, multimédia

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. [1]

Szántó szerint [1] 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 időt hagy arra, hogy végig gondolja, osztályozza ezeket a gondolatokat, mérlegelje saját stratégiáját, ne vonjon le túl gyorsan következtetést, végül értékelje 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 tehát fejlesztik a tanulók kognitív képességeit.

A programozásoktatás a programozási tételek tanításával szerepet tud vállalni a tanulók kognitív képességeinek fejlesztésében, mégis, a tapasztalatok szerint – külföldön is –, nehéz tanulni és tanítani algoritmusokat. A tanítási módszerek fejlesztése ezen a területen fontos feladata a mai informatikaoktatásnak. [2, 21, 22]

A programozási tételek tanítása alapeleme a középiskolai programozásoktatásnak. Ezeken keresztül érti meg a tanuló, hogyan működik együtt egymással a specifikáció, az algoritmus és a kód, illetve, hogy e tételek összeépítésével tud majd bonyolultabb programokat alkotni. [21]

Tanári és tanulói tapasztalataink 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. Nyilvánvalóan, a helyességbizonyítást [23] el kell vetnünk, mint módszert, ugyanis középiskolában nincs meg ehhez a tanulók kellő tudása, ráadásul elrettentené őket. A másik indok a tisztán

matematikai eszközök mellőzése mellett, hogy a programozás célja, hogy a tanulók kognitív képességei fejlődjenek.

Tanítási gyakorlatunk alatt programozási tételeket tanítottunk egy fővárosi szakközépiskolában. Akkor egy prezentációba ágyazott animáción keresztül mutattuk meg a diákoknak az adott tétel működését, tehát a diákok látták az animációt és hallották a magyarázatot. Sokkal többet tudtunk így átadni, mint ha csak az algoritmus szövegét tanulmányoztuk volna végig. Ezt a hatást hívja Mayer „Multimédia hatásnak" a Multimédia tanulás kognitív elméletében (*Cognitive Theory of Multimedia Learning*), amely öt alapelvet fogalmaz meg [3]:

- 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. [4] Tapasztalatainkból azt a következtetést vontuk le, hogy ha olyan eszközöket használok a tanítás alatt, amelyek segítenek *elképzelni* az algoritmus belsejében történteket, akkor meggyorsítjuk a megértés folyamatát. Az algoritmus-vizualizációs (AV) eszközök használata ebben nyújt segítséget.

1. Algoritmus-vizualizáció

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. [5]

Az AV a múlt század 70-es éveinek végén a batch-oritentált szoftverekből – amelyek lehetővé tették az oktatóknak animációs filmek készítését [6] – mára magas szintű interakcióval rendelkező rendszerekké fejlődött, amelyekkel a tanulók felfedezhetik, beállíthatják, dinamikusan megváltoztathatják az algoritmus animációját a saját igényeiknek megfelelően [7, 8] vagy maguk képesek megalkotni saját vizualizációjukat [9, 10]. A következőkre használták ezeket: Az AV program segítette

- az oktatót az algoritmus illusztrálásában, [7]
- a tanulókat abban, hogy megértsék az alapvető algoritmusok működését, [11]
- a hibakeresést konzultáción, [12]
- a tanulókat megérteni egy absztrakt adattípus műveleteinek működését. [13]

Hundhausen és munkatársai összefoglaló tanulmánya [5] szerint, az intuitív vonzereje ellenére az AV nem terjedt el pedagógiai eszközként az informatikaoktatásban. Az egyik ok a sok közül, 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.

Milyen tulajdonságokkal jellemezhető egy jó demonstrációs eszköz? [2]

- 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. [14]

Kehoe és munkatársai tanulmányukban [15] 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 nem csak 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

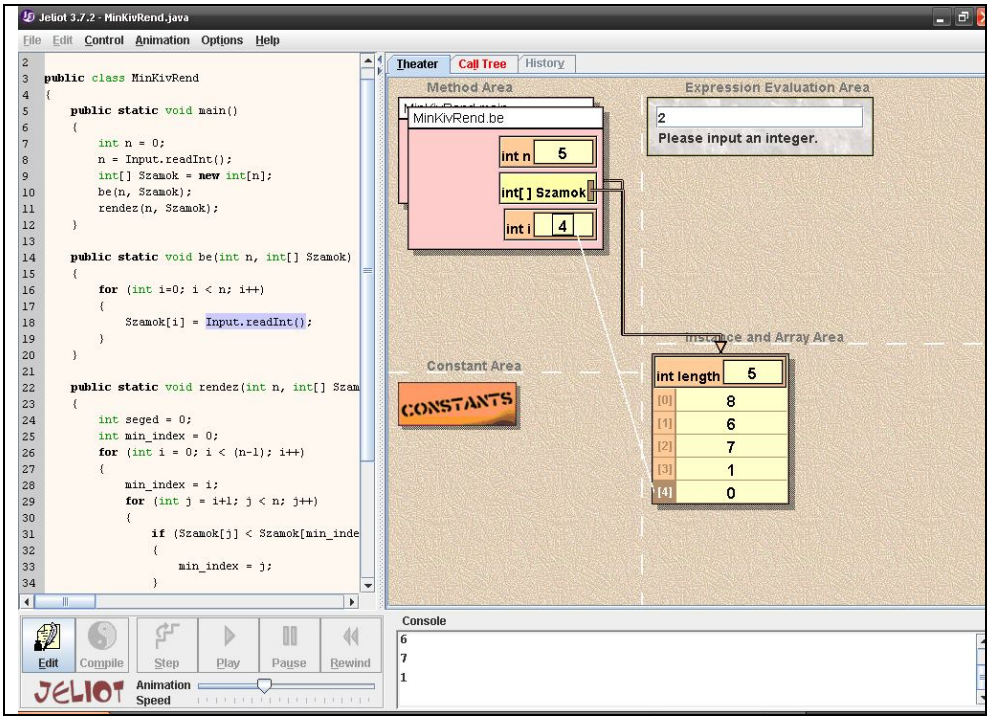
időt foglalkoztak a tananyaggal, mint a másik csoport. Az algoritmus animáció tűnik a leginkább alkalmasnak, hogy segítsen továbbítani egy algoritmus műveleteit lépésről lépésre, így világos vizuális reprezentációt nyújt egy amúgy absztrakt folyamatról.

Az algoritmus-vizualizációs eszközök használata Magyarországon szinte ismeretlen. Magyar nyelvű és vonatkozású írás a mai napig nem jelent meg ebben a témában, holott, nemzetközi kutatások sok pozitív eredményt mutattak fel az elmúlt csaknem 40 évben.

2. Jeliot 3

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 (lásd 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), amely 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 tanulóknak lehetőségük van programozni a Jeliot 3-ban és később annak működését követni tudják az animáción keresztül.



1. ábra. *Jeliot*

[1] http://cs.joensuu.fi/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 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 (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 [14] 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, ahogyan definiálták és elmagyarázták a fogalmakat.

A figyelem a tanulás első lépése [16]. Erre az elméletre 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.

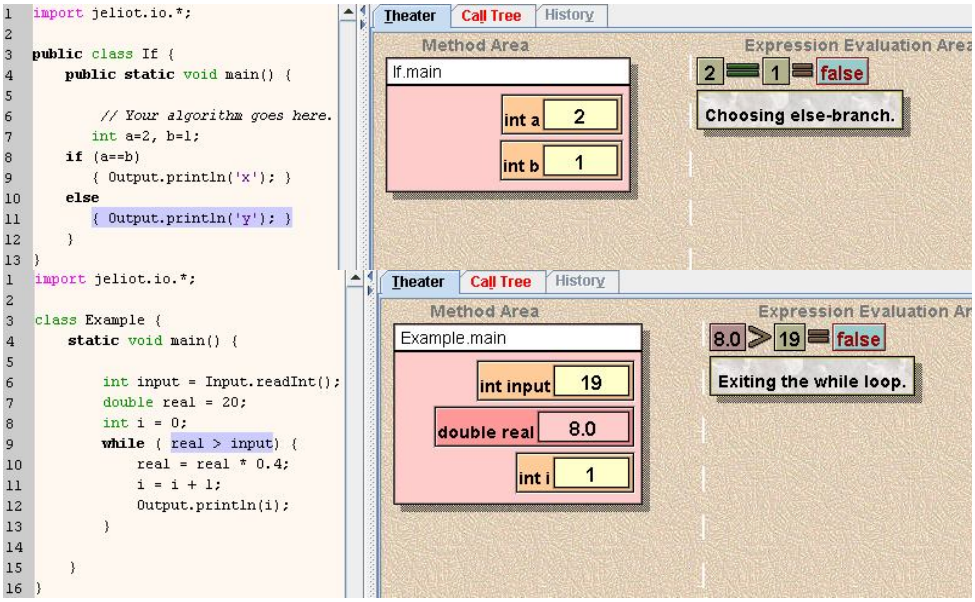
Moreno és Joy vizsgálata [18] a rendszer határaira mutatott rá: a Jeliot 3 nem elég flexibilis ahhoz, hogy különböző tudásszinten levő tanulókat vagy használati sablonokat támogasson. Általánosan igaz, hogy az átlag alatti vagy feletti tanulók teljesítményénél nem tapasztaltak szignifikáns különbséget.

3. Néhány alkalmazási példa

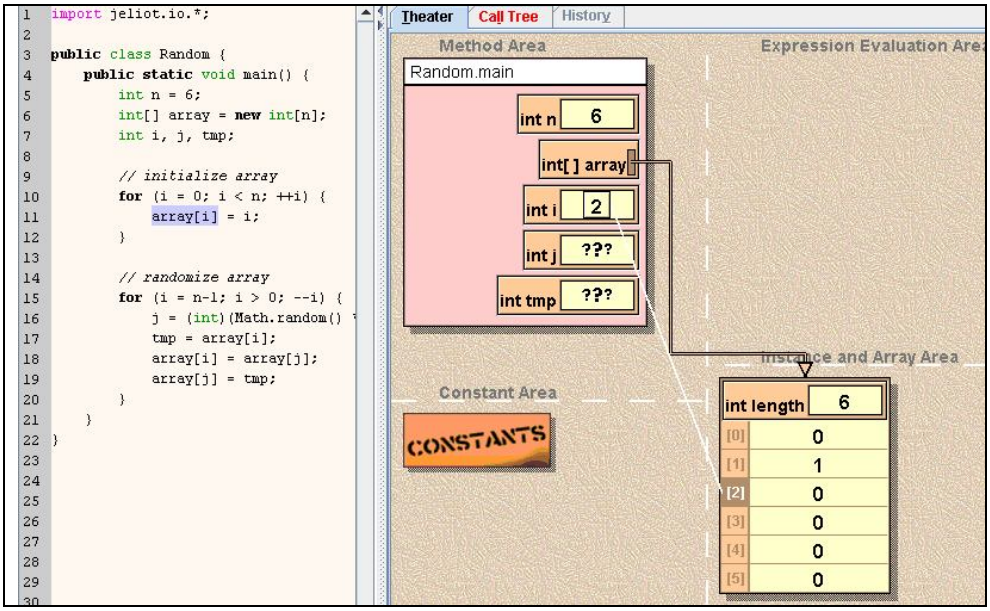
A Jeliot 3 alkalmas a vezérlési szerkezetek (elágazás, ciklus), illetve programozási tételek belső működésének reprezentálására, így támogatva a tanár magyarázatait és az önálló tanulást is.

Nem jelentős hátrány a környezet „nyelvfüggősége”, nyílván leginkább a Java nyelven tanulók tudják kihasználni a rendszer szolgáltatásait. A vezérlési szerkezetek és az algoritmus működésének megértése nem nyelvfüggő feladat.

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 (lásd 2. ábra).



2. ábra. *Vezérlési szerkezetek működésének reprezentálása*



3. ábra. *Tömb megjelenítése*

Gyakran használt adatszerkezet a tömb, ennek belső állapotát is képes megjeleníteni a program (lásd 3. ábra).

Tekintsünk egy egyszerűbb programozási tételt, a lineáris keresés tételét:

A specifikáció az ELTE informatika tanári szakán alkalmazott konvenciókat [19, 21] követi. (Ennek formális vagy informális volta most nem fontos, a lényeg, hogy a megoldáshoz egyértelmű információkkal szolgáljon.)

Specifikáció:

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

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

Előfeltétel: $Hossz(X)=\mathbf{N}$

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

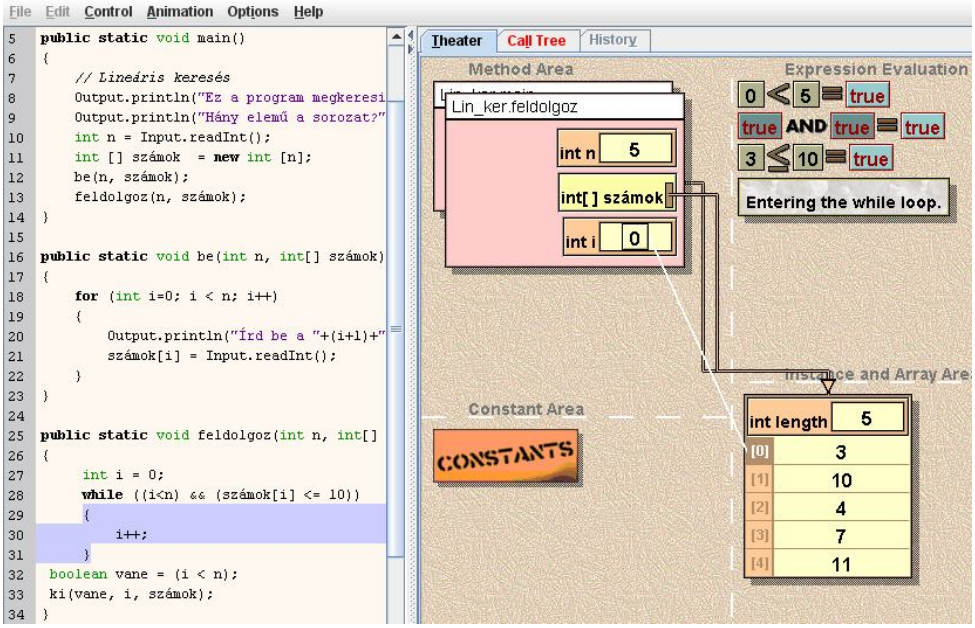
Alább csak a lényegi algoritmust közöljük és vizsgáljuk. A beolvasást és a kiíratást csak a nyelvvel, ill. a környezettel kapcsolatban fogjuk tárgyalni, hiszen ott markánsan eltérőek lesznek.

Algoritmus:

```
Eljárás Lineáris_keresés(Konstans N: Egész; X: THk;
                         Változó Vane: Logikai,
Sorszám: Egész):
    Változó
       I: Egész
    I:=1
    Ciklus amíg I≤N és nem T(X(I))
       I:=I+1
    Ciklus vége
    Vane = (I≤N)
    Ha Vane akkor Sorszám:=I
Eljárás vége
```

Akkor értette meg a tanuló az 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:



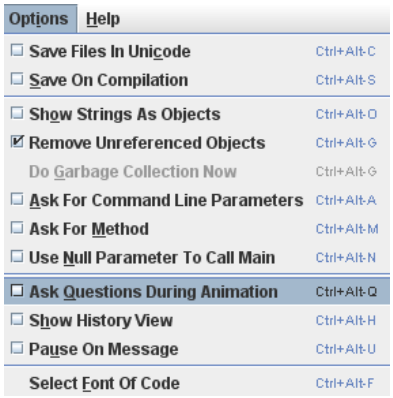
4. ábra. Ciklus bennmaradási feltételének kiértékelése

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!”

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 4. 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.

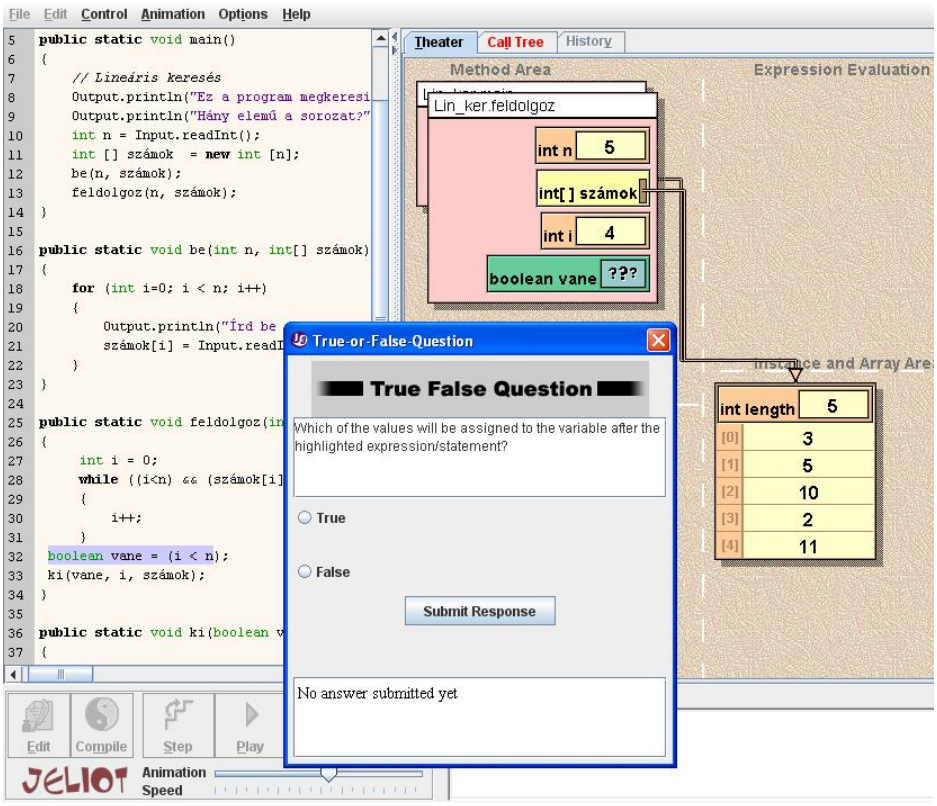
Mi lát a tanuló? Látja a programkódot és a program belső állapotát, tehát van szöveges és képi információja, ilyen módon teljesül a Multimédia-hatás. A kód elrejthető, ha zavart okoz, pl. olyan esetben, ha a tanulók nem Java-t, hanem egy másik programnyelvet használnak. Ebben az esetben a saját kódját tudja olvasni az animáció mellett. A program belső állapota mindent elmond: éppen melyik tömbelemet vizsgáljuk, mik a helyi változók értékei, illetve hogy állunk a ciklus belépési feltételének kiértékelésével.

A Jeliot 3-at be tudjuk úgy állítani, hogy kérdéseket tegyen fel a változók értékeit illetően. Ennek beállítását a 5. ábra mutatja.



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

A kérdések eléggé „egysíkúak”, sajnos, mindig csak a változók/kifejezések értékére, értékváltozására kíváncsi, azonban, algoritmusunkat tekintve, az egyik kérdést nagyon is jó helyen teszi fel (lásd 6. ábra), ilyen módon nagyon hasznos a környezet otthoni, egyéni tanulásra, illetve hibakeresésre is.

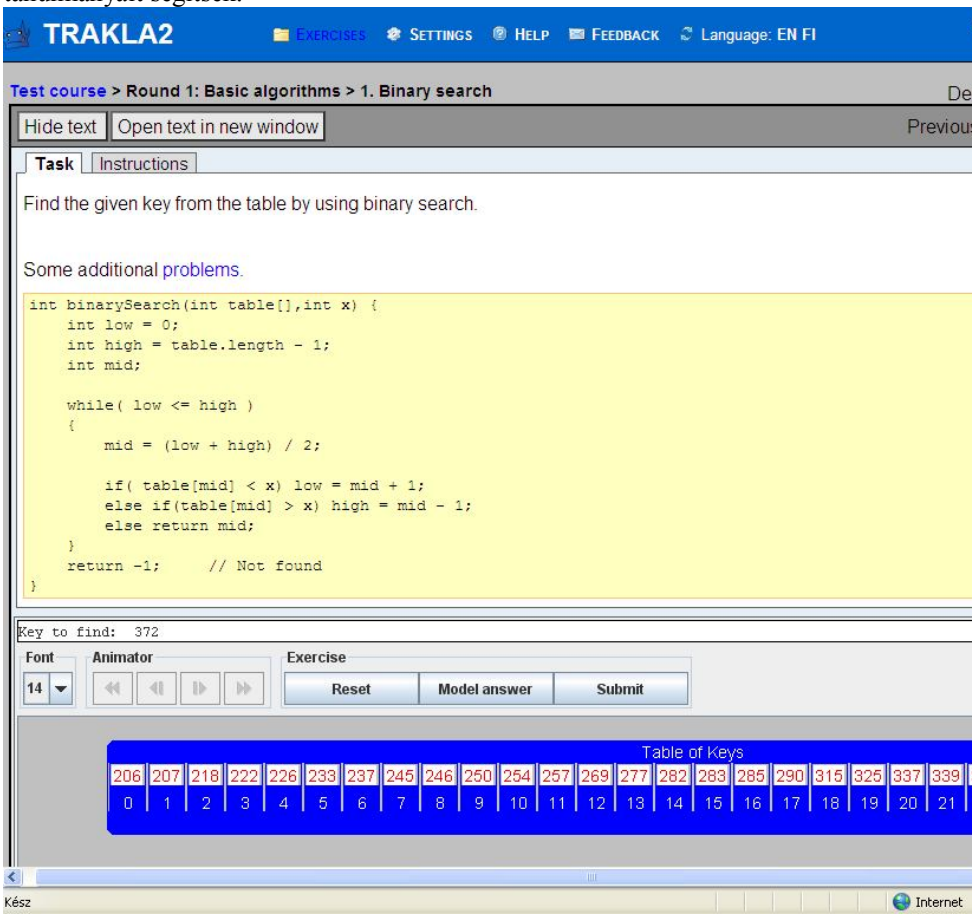


6. ábra. *A belső állapotot leolvasva helyesen tudunk felelni a kérdésre*

Felmerülhet kérdésként, hogy mennyire zavaró az a tény, hogy a környezetet, szemmel láthatóan, Java programozási nyelvvel való oktatásra tervezték. Talán egy kicsit zavaró. A környezettel történt korábbi vizsgálatok [20] alkalmával Turbo Pascalt használtak a tanulók, és az idézett tanulmány szerint az alapkoncepciókban, mint pl. értékadás, a szintaktikus különbségek nem nagyok. A tanár mindig tudott segíteni az ebből adódó problémák megoldásában.

4. TRAKLA2

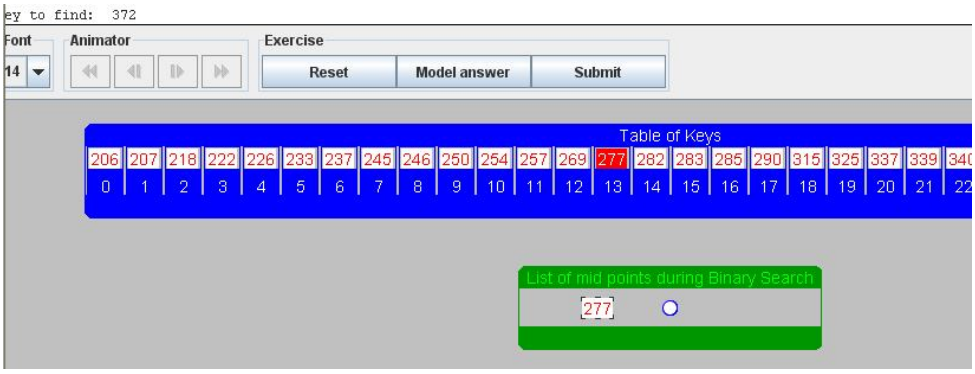
A TRAKLA2 algoritmus-vizualizációs (AV) rendszer egy teljesen más stratégiát és módszert követ. Abból a célból fejlesztették, hogy elsőéves egyetemisták programozás-tanulmányait segítsék.



7. ábra. TRAKLA2

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.

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 (8. ábra) zöld „dobozba” húzzák bele a középső elemeket.



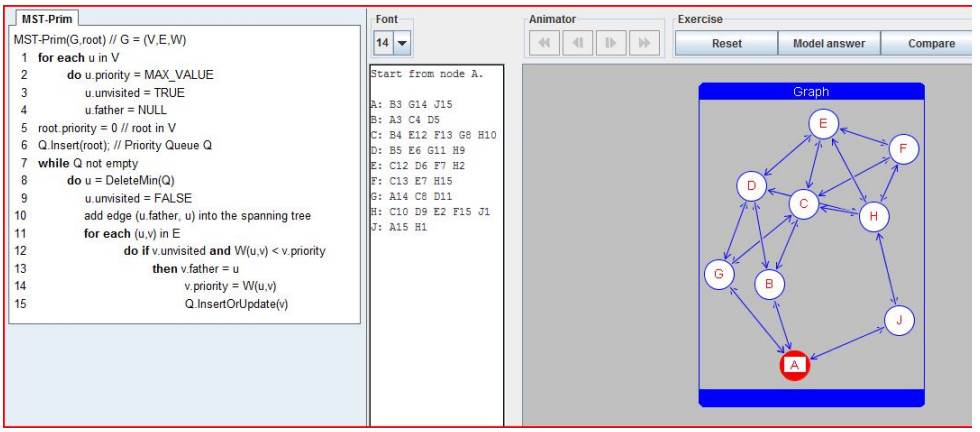
8. ábra. *Feladatmegoldás közben*

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 bementen 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, az egyszerűbb algoritmusokkal kiegészítve a közoktatás hasznos segédeszközévé is válhat.

Jól látható az ábrákon, hogy a környezet nem csak 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 (lásd 9. ábra).



9. ábra. *Példa gráfalgoritmusra*

5. Összefoglalás

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

A program támogatja a frontális oktatást, ahol a tanár szóbeli magyarázatait egészíti ki képi információkkal, illetve az egyéni tanulásnál programkódot megértését segíti vizuális elemekkel, így jól valósítja meg Mayer elméletét. [4] A rendszer rendelkezik olyan funkcióval, hogy az animáció futtatása idején, az adott változó használata közben rákérdezzen annak értékére. Sajnos, más jellegű kérdést nem tesz fel a program, de ezek a kérdések tesztelhetik, hogy mennyire érti a tanuló a program működését, illetve támogatják a hibakeresést.

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 eljárások 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.

A TRAKLA2 különösen támogatja az aktív tanulást, ugyanis a tanuló nem passzív szemlélője az animációnak, hanem ő maga „játssza el” az algoritmust, ilyen módon, ha szükséges, rossz megoldásokon keresztül eljutva, megerősíti a tanulóban az algoritmus helyes működését.

Az AV oktatásban való felhasználásáról jelenleg nincs magyar nyelvű irodalom vagy magyar vonatozású eredmény. Arra a kérdésre, hogy milyen módszerekkel és elvek betartásával lehetséges az AV-t a magyar középiskolai programozásoktatás gyakorlatába bevonni, további kutatások adhatják meg a választ. A külföldi eredmények vegyesek, de vannak reményt keltőek abban a vonatkozásban, hogy idővel a tanárok és a tanulók kezébe olyan segédeszköz kerülhet, amely növeli a tanulás hatékonyságát és a diákok motivációját.

Irodalomjegyzék

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

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

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

[4] 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.

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

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

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

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

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

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

[23]Szlávi Péter: *Programok, programspecifikációk*, In: Informatika a felsőoktatásban'99, pp. 576-582, Debrecen, 1999.