

Eötvös Loránd Tudományegyetem Informatikai Kar

Az algoritmus vizualizáció új megközelítése

Készítette: **Németh Boldizsár**
Programtervező Informatikus szak, III. évfolyam

TDK-dolgozat

Témavezető: **Kőhegyi János** mestertanár
Algoritmusok és Alkalmazásaik Tanszék



Budapest, 2012. május

Tartalomjegyzék

1. Bevezető.....	1
2. Az Algoritmus Vizualizáció	3
2.1. Az AV helye a tudományágak között	3
2.2. A tudományág története	3
2.2.1. Milyen vizualizációk érhetőek el?	4
2.3. Az AV rendszerek hatása a tanulás hatékonyságára	6
2.3.1. A vizualizációs környezet lehetőségei és a vizualizáció hatékonysága.....	6
2.3.2. Az animációk kidolgozottságának hatása	7
2.3.3. A tanulók aktivitásának hatása	7
2.3.4. Több csatornán történő információátadás.....	8
2.3.5. Az AV használata a tanulás különböző fázisaiban	8
2.4. Létező AV Rendszerek.....	10
2.4.1. ANIMAL - Advanced Navigation and Interactive Modeling of Animations for Lectures	10
2.4.2. JFLAP - Java Formal Language and Automata Package	11
2.4.3. Jeliot 3	13
3. Az AV elméleti megközelítése	15
3.1. Az AV folyamatának résztvevői.....	15
3.1.1. A DAVIK koncepciója az AV résztvevőiről	15
3.2. Vizualizáció készítési módszerek	17
3.2.1. Animáció generálás grafikus felületen keresztül	17
3.2.2. Vizualizációs program megírása	18
3.2.3. Animáció generálás programinterfész segítségével	19
3.2.4. Leírónyelvvel történő vizuális leképezés	20
3.2.5. Animációgenerálás kódértelmezéssel	21
3.2.6. Összefoglalás.....	22
3.3. Az Algoritmus Vizualizáció részei	24
3.3.1. Az állapottér megjelenítése	24
3.3.2. Algoritmikus szerkezet megjelenítése	25
3.3.3. Az algoritmus háttere	26
3.3.4. Az információk integrációjának fontossága.....	26
3.3.5. Vezérlés.....	27
3.4. Milyen algoritmusok mutathatók be az AV segítségével?.....	28
3.4.1. Rendezések	28

3.4.2. További algoritmuscsoportok	29
3.4.3. Az AV alkalmazhatósága	29
4. A DAVIK rendszer kidolgozásának vezérelve	31
4.1. Egyszerű és gyors algoritmuskészítés	31
4.2. Algoritmus-leíró pszeudokód	33
4.2.1. Programozási nyelvek az oktatásban	33
4.2.2. A pszeudokód jellegzetességei	34
4.2.3. A DAVIK rendszer pszeudokódjának alapelvei	35
4.3. A rendszer és a felhasználó interakciója	37
4.4. Vizualizációs ablakok	38
4.5. Egységes működés és összevethetőség	39
4.6. Információk és vizualizáció integrálása	41
4.7. Hordozhatóság	41
5. Összefoglalás	42
6. Irodalomjegyzék	44

„Tökéletes a megoldási módszer akkor,
ha kezdettől fogva előre látjuk,
sőt, be is bizonyítjuk,
hogy azt követve elérjük célunkat.”

- Gottfried Wilhelm Leibniz

1. Bevezető

A rendszer létrehozásának gondolata 2010-ben merült fel bennem, amikor másodéves hallgatóként Kőhegyi János Tanár Úr Algoritmusok és Adatszerkezetek II gyakorlatára jártam. Ez a csoport a szokványostól eltérő módon tanulta meg az összetettebb, gráfokkal, tömörítéssel, szövegfelismeréssel és hasító-függvényekkel foglalkozó algoritmusok működését. Ahelyett, hogy táblán mutatta volna meg az algoritmusok működését, a Tanár Úr arra kért minket, hogy készítsünk az algoritmusokhoz egy vizuális megjelenítést, amiből elsősorban mi magunk, de mások is tanulhatnak.

Hamar rájöttem, hogy a hasonló algoritmusok vizualizációja nagymértékben épít közös elemekre. Ahogy haladtunk előre a hasonló feladatokkal, egyre kevesebbet kellett dolgoznom, mert a munka nagyobb részt már kész volt. A félév végére elkészült egy gráfalgoritmusokat megjeleníteni képes rendszer.

Persze, ez a rendszer nem volt tökéletes. A kezelőfelület nem volt ergonomikus, mind az új algoritmusok, mind a bemeneti adatok beviteléhez java kódot kellett írni, amit le kellett fordítani és fel kellett tölteni a szerverre. Amikor egy új algoritmust helyeztem el a rendszerben, a vizuális felületet úgy kellett átalakítani, hogy annak a vizualizációjára is megfeleljen. Továbbá vizualizáció megjelenését nem lehetett befolyásolni, mindenkinek úgy kellett néznie az algoritmust, ahogy én gondolatban felépítettem.

Ekkor fogalmazódott meg bennem, hogy jó lenne egy olyan rendszer, amelyben nincsenek meg ezek a korlátozások, és nem csak gráf-, hanem tetszőleges algoritmus vizualizációjára alkalmas. Ahogy a szakirodalmat és a különböző kísérletek elemzését olvastam, arra jutottam, hogy ami még nem készült el, az egy olyan rendszer, mely nem csak a jól használható vizualizációt, hanem az algoritmusok gyors létrehozását is segíti.

A szakirodalom tanulmányozása, és az elkészült programok vizsgálata során fokozatosan jöttem rá, hogy mi az, amit hiányolok belőlük. Legtöbbjük a hagyományos szemléletet követte, hogy a tanár összeállít egy vizualizációt, amit a diákok kipróbálnak. De mi lenne, ha a vizualizáció előállítása alig lenne több mint az algoritmus megadása? Így a diákok maguk is kísérletezhetnének, átírva az oktató által bemutatott algoritmust, vagy teljesen új algoritmusváltozatokat próbálgatva.

Még ha nem is születnének ezzel a módszerrel új, a gyakorlatban is használható algoritmusok (amit viszont nem tartok kizártnak), akkor is ez a kísérletezgetés megtanítaná a diákokat arra, hogy milyen elvek mentén lehet új algoritmusokat megtervezni, illetve mi miatt működnek az órán tanult klasszikus algoritmusok. Egy ilyen rendszer lebonthatná a falat, ami elválasztja a kreatív diákokat a „bemagolni” való algoritmusoktól, így ha mást nem is, legalább kedvet teremt a tanuláshoz.

Remélem, hogy ez a rendszer nem csak elméleti érdekességként, hanem hasznos tanítási segédeszközként is megállja a helyét.

2. Az Algoritmus Vizualizáció

Az algoritmusvizualizáció (továbbiakban AV) az informatika egyik tudományága, mely az általános informatikai problémák megoldásának folyamatát jeleníti meg grafikus úton.

2.1. Az AV helye a tudományágak között

Az AV a szoftver vizualizáció részterülete. A szoftvervizualizáció foglalkozik még olyan területekkel is, mint a komplex rendszerek viselkedésének megjelenítése, vagy a vizuális hibakeresés. A szoftvervizualizáció többi területe elsősorban a szoftverfejlesztést segíti, míg az AV az algoritmusok megismerését és az algoritmikus gondolkodás elsajátítását. Az AV szerepe a program szerkezetének és állapotterének bemutatására korlátozódik, míg a szoftvervizualizáció magában foglalja a szoftverrendszerek komponenseinek és az azok között lévő kapcsolatok bemutatását is.

Az AV azon formáit, ahol a felhasználó nem gyakorol hatást az algoritmus működésére, szokás algoritmus animációnak is nevezni.

Dolgozatunkban kizárólag az algoritmus vizualizációra koncentrálunk, nem térünk ki sem a komplex rendszerek vizualizációjára, sem a statikus animációkra.

2.2. A tudományág története

A történeti kutatás terén sokat merítünk az ANIMAL-FARM rendszert ismertető, a tudományág eredményeit feldolgozó könyvből [av4].

- 1947: Neumann és Goldstein folyamatábrákat használ algoritmusok bemutatására.
- 1959: Lois Haibt a Yorktown Heights-i IBM kutatólaboratóriumban elkészít egy rendszert, mellyel a folyamatábrák a forráskód alapján automatikusan generálhatók. [av10]
- 1980: Dr. Ronald Baecker a Torontói Egyetemen elkészíti az első sikeres animációt (Sorting Out Sorting) mely a rendezőalgoritmusok működését mutatja be és hasonlítja össze [av11].

- 1984: A Brown Algorithm Simulator and Animator (BALSA) az első rendszer, mely nem csak az algoritmusok egy speciális részét mutatja be, hanem általánosan használható. Marc Brown fejlesztette ki a Brown egyetemen. [av12]
- Az elérhető algoritmus vizualizációk mennyisége gyors növekedésnek indul a 90-es évek elején. [av5]
- 1995: Megjelenik a Java nyelv és platform, a vizualizációk készítői gyorsan áttérnek a használatára. Innentől a vizualizációk jellemzően a java applet technológiával készülnek. [av5]
- 2000: A Lawrence egyetemen elkészül a JHAVE nevű általános környezet, melyben különböző animációgenerálási rendszerek egységes felületet tudnak nyújtani. [av13]

A tudományág fejlődésében megfigyelhető egy tendencia, mely a passzív animációk elkészítése felől halad az általánosabb és dinamikusabban használható rendszerek felé. A DAVIK rendszert ennek a természetes fejlődésnek a folytatásaként tekintjük. Ahogy a tudományág halad a változtatható bemenetű és testreszabható algoritmusmegjelenítés felé, természetes lépésnek tűnik, hogy magukat a megjeleníthető algoritmusokat is dinamikussá tesszük.

Az AV-vel kapcsolatos saját tapasztalataink és a tudományterületen uralkodó tendenciák alapján döntöttünk úgy, hogy a megvalósítás a Java nyelven történjen, mely kellő rugalmasságot, és könnyű terjeszthetőséget biztosít a fejlesztéshez.

2.2.1. Milyen vizualizációk érhetőek el?

A Virginiai Egyetem egy kutatócsoportja több mint 350, az interneten elérhető algoritmus vizualizációt tanulmányozott [av5]. Sajnos nagy számukkal ellentétben ezeknek a vizualizációknak a minőségét alacsonynak találták.

Bár nagy számban találtak önálló vizualizációkat is, jellemző tendencia az, hogy egy adott rendszerben több algoritmust is implementálnak. Személyes tapasztalataim segítségével könnyen értelmezni tudom a jelenséget. Egy önálló algoritmus implementálásakor a munka túlnyomó része nem az algoritmus, hanem a vizualizáció módszerének megvalósításából áll. Ezért, ha már egy vizualizáció elkészült, a többi hasonló témájú (emiat hasonló vizualizációs módszerekkel szemléltethető) algoritmus

megjelenítése az eredeti erőfeszítés töredékéből megvalósítható. Ezt a jelenséget gráfokkal foglalkozó algoritmusok implementálásakor tapasztaltuk.

A vizualizációk témája meglehetősen szelektív, a többségük rendezésekkel, keresésekkel, lineáris struktúrákkal foglalkozik, jellemzően az algoritmusok oktatásának első kurzusaira illeszkednek. Kimaradnak, vagy csak nagyon gyenge minőségben jelennek meg a specializáltabb, vagy haladó szintű algoritmusok. Az elérhető vizualizációknak ez a szelektív témaválasztása arra mutat, hogy hasznos lehet egy olyan rendszer, amiben minőségi animációkat lehet gyorsan készíteni, azokról a témákról is, melyek nem a legnépszerűbbek a vizualizációk készítői számára.

2.3. Az AV rendszerek hatása a tanulás hatékonyságára

Arról, hogy a vizualizációs rendszer szolgáltatásainak bővülése mennyire segíti az algoritmusok ismeretanyagának elsajátítását, több kísérlet is történt.

A 2002 előtti 24 ilyen kísérlet eredményeit foglalja össze Hundhausen összefoglaló tanulmánya [av1]. A kísérletek eredménye vegyes, 11 kísérletben találtak jelentős különbséget a vizualizációval történő oktatás javára a csak írott anyaggal történővel szemben.

A kísérletek eredménye arra mutat, hogy nem maga a vizualizáció minősége az, ami eldönti az oktatás eredményességet, hanem a módszeré, amelyben használják. Azokban a kísérletekben, ahol a vizualizáció használata passzív volt (vagyis nem járt vele a tanulói aktivitás valamilyen formája), jellemzően nem bizonyult hatásosabbnak a szövegekkel és ábrákkal történő magyarázatnál. Tehát az algoritmus vizualizáció nem jobb olyan formában a hagyományos módszereknél, hogy „egy kép felér ezer szóval” [av1], de hatékony eszközzé válhat az informatika oktatásában. A tanulmány eredményei szerint az AV használatának pozitív hatásai jobban megmutatkoznak az algoritmusok utánzásának (és ezzel együtt megvalósításának) képességében, mint azok elméleti megértésében.

2.3.1. A vizualizációs környezet lehetőségei és a vizualizáció hatékonysága

Purvi Saraiya kísérletei [av9] például azt mutatják, hogy a több vezérlési lehetőséggel, és a több szempontból való megjelenítéssel rendelkező vizualizációk megtekintése elsősorban az elméleti jellegű kérdések megválaszolásában segít, míg a procedurális kérdések megválaszolását inkább az egyszerűbb, kevesebb lehetőséget biztosító vizualizáció segíti.

Itt azonban figyelembe kell vennünk, hogy a kísérletben résztvevő hallgatók az adott vizualizációs felületekkel itt találkoztak először, így az egyszerűbb felületek sikerességét nagymértékben növelhette a könnyű kiismerhetőségük.

Viszont egyértelműen kiderül a kísérletek eredményeiből, hogy a megtekintő által vezérelt vizualizációk pedagógiai hatása nagyobb, mint a csak lejátszható animációké,

és hogy a bemeneti adatokra adott jól megválasztott példák segítik az algoritmus megértését.

Ez megerősített minket abban az elképzelésünkben, hogy a vizualizációnak teljes mértékben vezérelhetőnek kell lennie. Nem is tettük lehetővé az automatikus lejátszást, mert az véleményünk szerint valószínűbbé tenné azt, hogy a tanuló elmulasztja észrevenni a vizualizáció fontos pontjait. Hogy egy lépést többször is meg lehessen figyelni, az egész vizualizációt teljes mértékben megfordíthatóvá tettük, így bármely lépés visszavonható.

2.3.2. Az animációk kidolgozottságának hatása

Andrea Williams Lawrence hat kísérlete a vizualizációk egyes szolgáltatásainak hatását vizsgálja a tanulás sebességére [av8]. Az eredményei azt sugallják, hogy a vizualizáció szempontjából nem lényeges a bemutatott bemeneti adatok mérete, a prezentáció pontos formája. Továbbá nem volt az eredményre jelentős hatással, hogy előbb szöveges formában mutatták be az algoritmust, vagy vizuálisan, illetve hogy milyen sorrendben mutatták az vizualizációkat be, amikor több algoritmus került bemutatásra. Bizonyos területeken az egyszerűbb megoldás vált hatékonyabbá, például, a fekete-fehér animációt megtekintők jobb eredményt értek el, mint a színeset tanulmányozók.

Ezeknek az eredményeknek a nyomán világossá válik, hogy az animációkon történő esztétikai fejlesztéseket mindenképp alaposan meg kell fontolni, mert a látvány minősége nincs szoros összefüggésben a pedagógiai hatékonysággal. Az ilyen irányú erőfeszítések lehetséges, hogy nagyobb hasznot hoznak, ha a felhasználói aktivitás növelésére fordítják őket.

2.3.3. A tanulók aktivitásának hatása

Két kísérlet eredménye [av22,av36] is arra mutat, hogy minél aktívabb szerepet kapnak a felhasználók a vizualizációban, annál könnyebben sajátítják el az algoritmus működésének ismeretét. Ezekben a kísérletekben azok a felhasználók, akik önállóan állíthatták össze a bemeneti adatokat, jobb eredményeket értek el a teszten.

Ehhez hasonlóan a felhasználó aktív tanulásának előnyeit támasztja alá Byrne és társainak kísérlete [av7] is, itt viszont a tevékenység az algoritmus lépéseinek

előrejelzése volt. Jarc és társainak kísérlete [av16] is az előrejelzés szerepét vizsgálta a vizualizáció segítségével történő tanulásban. Az eredményben jelentős különbség nem mutatkozott, azonban az előrejelzést is végző diákok több időt töltöttek el a vizualizációs szoftverrel, mint az előrejelzést nem végzők, ami a nagyobb motivációra mutathat.

Az elvégzett kísérletek kétféle felhasználói aktivitás hatását vizsgálták az eredményességre nézve. Az egyik a bemeneti adatok összeállítása, a másik az animáció lépéseinek előrejelzése. Mi a DAVIK rendszer tervezésekor egy harmadik aktivitási forma megteremtésére vállalkoztunk, melyben a tanulók magával az algoritmussal lépnek interakcióba, magán az algoritmuson végezhetnek változtatásokat, miközben tanulmányozzák.

Fest és Kortenkamp felmérése a Visage rendszerrel [av21] arra mutat, hogy a tanulók is értékelik azt a lehetőséget, hogy maguk állíthatják össze az algoritmust, ami vizualizációra kerül. Ezen a téren a különbség az, hogy a Visage rendszer esetében az algoritmus létrehozásakor a megjelenítést is meg kell valósítani, míg a DAVIK-ban ez már a rendszer szintjén megtörtént.

2.3.4. Több csatornán történő információátadás

Hansen kísérletei [av17] a HalVis tanulási környezettel különösen sikeres eredményt hoztak. A vizsgált rendszer nem kizárólag animációs rendszer, hanem a hagyományos és multimédiás oktatási módszereket egyesítő virtuális tankönyv. Továbbá más kísérletek eredményei is abba az irányba mutatnak, hogy az átadott tudás egyszerre több formában történő reprezentációja kedvezően segíti elő a tanulási folyamatot.

Az ilyen eredmények miatt döntöttünk úgy, hogy a DAVIK rendszerében egyszerre lehessen megtekinteni az algoritmus aktuális állapotát, struktúráját és szöveges magyarázatát.

2.3.5. Az AV használata a tanulás különböző fázisaiban

A vizualizációs módszerek hatékonyságának értékeléséhez ügyelnünk kell arra, hogy egy adott kísérlet milyen szituációban, milyen tanulási fázisban alkalmazza azt.

Teszt szituáció

A hallgató számára egy meghatározott időmennyiséget biztosítanak, hogy az AV rendszer használatával megértse egy algoritmus működését.

A Hundhausen-tanulmányban [av1] lévő kísérletek többsége ilyen körülmények között zajlott le, az eredmények pedig, mint azt már ismertettük, vegyesek. Viszont nem engedhetjük meg, hogy az AV hatékonyságát csak ebben a helyzetben vizsgáljuk, pusztán azért, mert egy kísérlet körülményei között ezt a helyzetet a legkönnyebb szimulálni.

Házi feladat szituáció

Itt a hallgató korlátlan ideig tanulmányozhatja az algoritmus működését.

Colleen Kehoe álláspontja [av17] szerint az AV hatékonyságának vizsgálatakor nem szenteltek elég figyelmet annak, hogy milyen tanulási szituációban történt a vizualizáció használata. Saját kísérletében, melyben házi feladat jellegű szituációt hozott létre, jelentős pontszámkülönbséget tapasztalt a vizualizációt használó alanyok javára.

Itt természetesen nem csak arról van szó, hogy a vizualizációval történő tanulás hatékonyabb, hanem a tanulók motiváltsága is nagyobb ebben az esetben. Erre utal az, hogy a vizualizációt használó csoport tagja az említett kísérleten átlagosan 10 perccel több időt töltöttek a felkészüléssel, mint azok, akik csak szöveget és illusztrációkat használhattak.

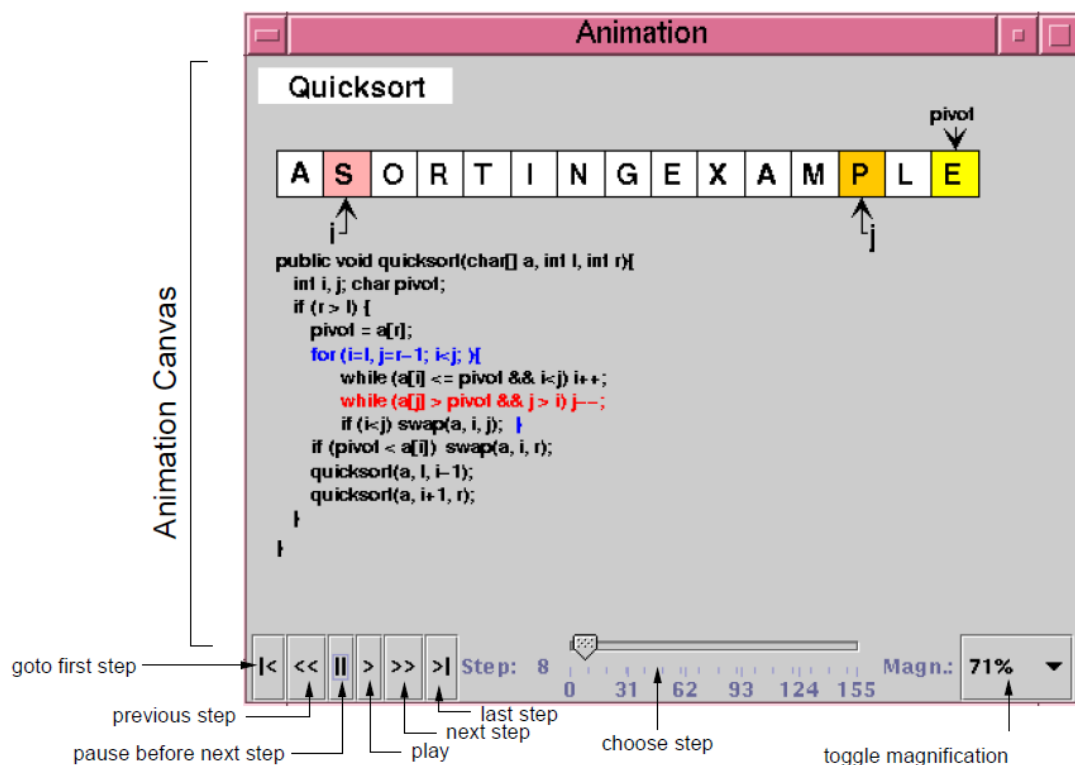
Előadás vagy konzultáció

Itt az AV rendszer használója az oktató, aki a tábla helyett a rendszerben végzi el az algoritmus bemutatását. Előnyösebb egy összeállított prezentációnál, mert tetszőleges adatokkal elvégezhető a megjelenítés. A DAVIK rendszer fejlesztésekor nem elsősorban tanári segédeszközt szerettünk volna alkotni, ezért kisebb súlyt fektettünk az ilyen jellegű használhatóságára.

2.4. Létező AV Rendszerek

2.4.1. ANIMAL - Advanced Navigation and Interactive Modeling of Animations for Lectures

[av6] Animáció megjelenítő rendszer, melyet az algoritmusok oktatására fejlesztett ki Guido Rössling, a Darmstadti Műszaki Egyetemen, 1999 és 2000 között.



1. ábra: A gyorsrendezés egy megjelenítése az Animal rendszerrel

Tartozik hozzá vizuális animáció szerkesztő, és az AnimalScript nyelv segítségével szövegesen is elkészíthető az animáció, de van hozzá egy java alapú felület is, aminek a segítségével ezt programozott módon is létre lehet hozni. Az Animal-Farm rendszer könnyebbé teszi az Animal rendszer bővítését, és egy egységes attribútumalapú testreszabási felületet biztosít számára. Készült továbbá hozzá eclipse [web2] fejlesztőkörnyezetbe illeszkedő bővítmény [av14], többféle vizuális adatbeviteli felület, és több mint ötven animáció.

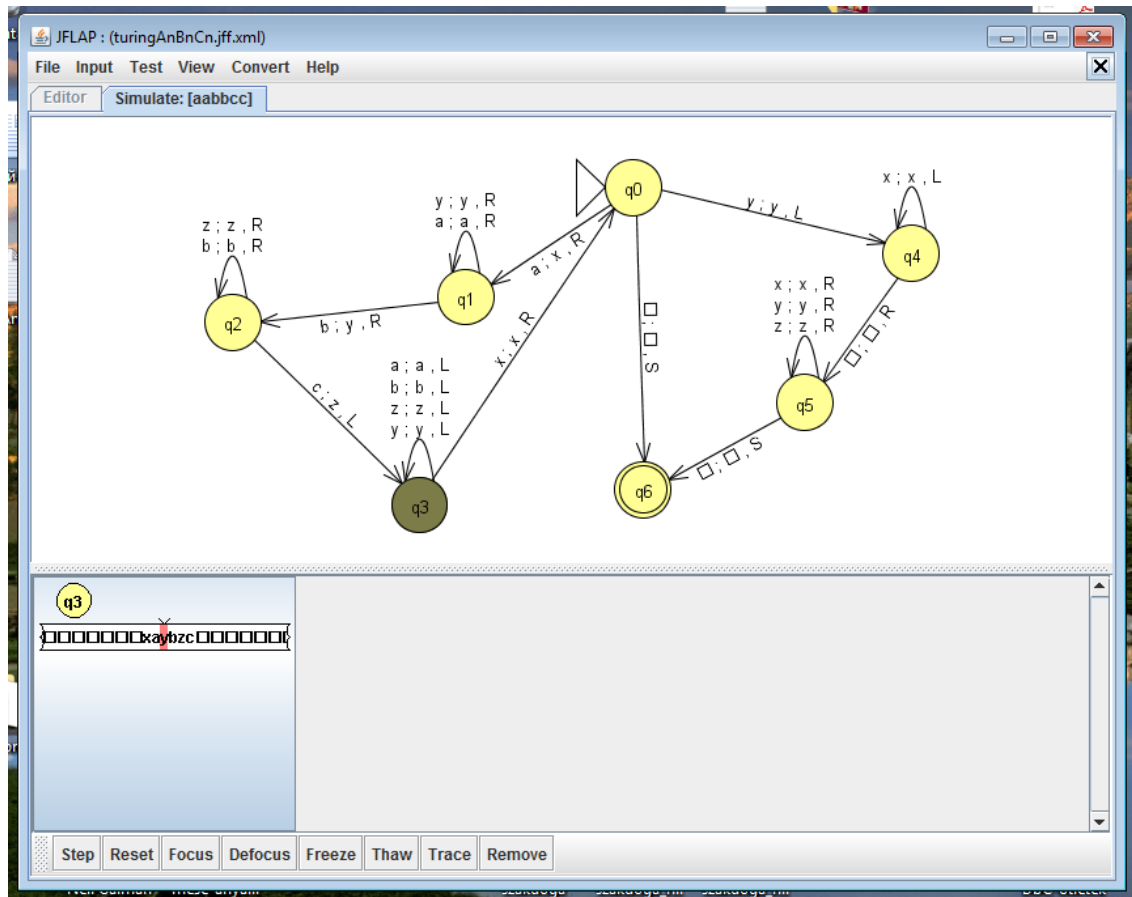
Az Animal rendszer segítségével tetszőleges animációt készíthetünk, így alkalmazására a legjobb lehetőség az olyan algoritmusok esetében van, melyek bemutatása speciális megjelenítési módszereket igényel.

Az viszont világos, hogy az animációk létrehozása az Animal rendszerrel komoly munkát igényel, mivel, ahhoz hogy dinamikus animációt tudjunk készíteni, meg kell ismernünk egy java nyelven írt rendszert, ami AnimalScript kódot generál, amit végül a rendszer lejátszik. Ennek a többlépéses folyamatnak a komplexitása jelentős.

Véleményem szerint az Animal egy bizonyos esetekben jól használható animációs rendszer, melynek sikere elsősorban a nagy támogatottságnak és a sokféle kiegészítő szoftvernek köszönhető. A DAVIK rendszer megalkotásakor is figyeltünk arra, hogy minél könnyebb legyen a programot továbbfejleszteni, illetve hozzá segédprogramokat alkotni.

2.4.2. JFLAP - Java Formal Language and Automata Package

[web3] Vizualizációs és szimulációs program a formális nyelvek témakörének tanulmányozására. A program funkcionalitása a különböző formális nyelvi modellek köré van csoportosítva (nyelvtanok, automaták, Mealy-, Moore- és Turing-gépek). A nyelvfelismerési modellek létrehozása grafikusán történik, és a tesztelés folyamatát is interaktívan lehet végigkövetni.



2. ábra: egy komplex Turing-gép működése a JFLAP rendszerrel bemutatva

Külön funkcióként szerepelnek benne a különböző modellek közötti átalakítások algoritmusai is. A tanulást segíti, hogy az algoritmusok lépésenként is elvégezhetők úgy, hogy a program csak általános utasításokat ad, amit a felhasználó hajt végre. Ezt az oktatás szempontjából jó gyakorlatnak tartom, bár összetettebb szerkezetek esetében az algoritmus végrehajtása monotonná válik.

A program használata során nyilvánvalóvá válik, hogy a fejlesztők komoly erőfeszítéseket fektettek a szoftverfejlesztés területén való használhatóságba is. Így kerülhetett bele kötegelte feldolgozás és a különböző tesztelési lehetőségek is.

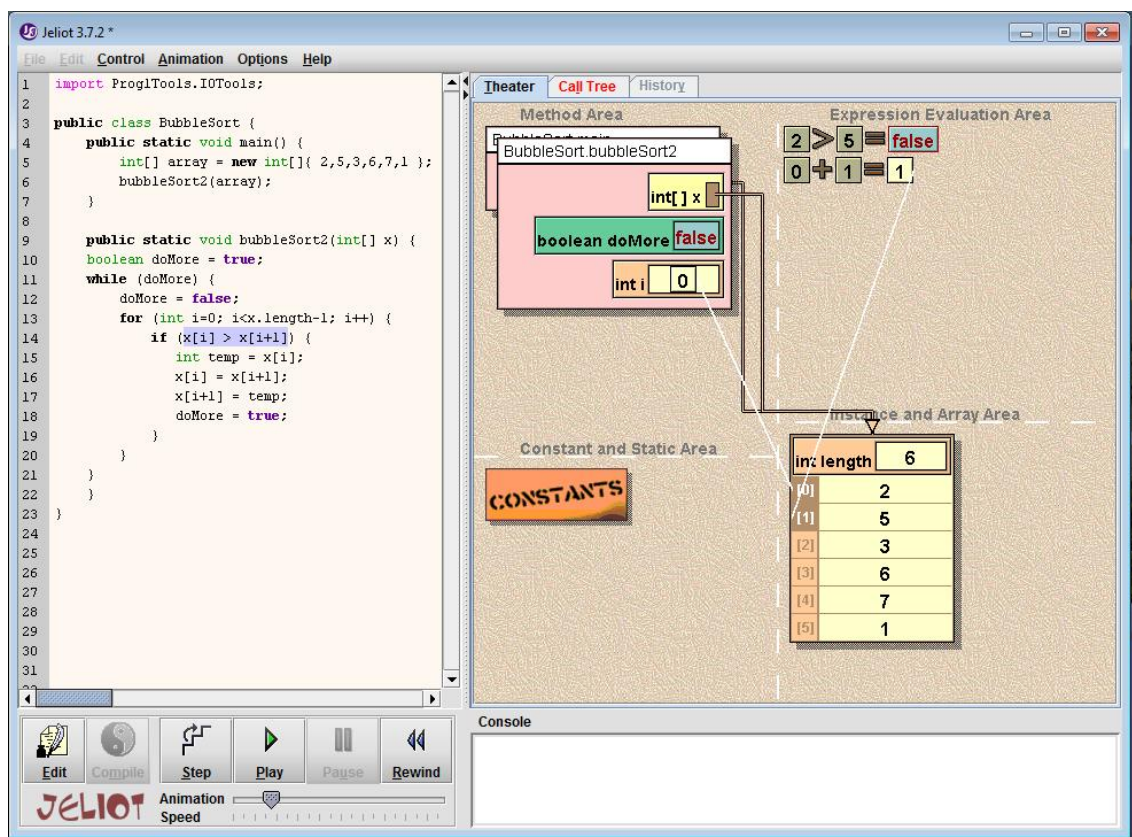
Szerintem a JFLAP tehát egy sikeres keveréke az AV programnak és az általános, a formális nyelvek használatát segítő alkalmazásnak. Sikeréhez valószínűleg az is hozzájárult, hogy nem általánosságban foglalkozik az algoritmusok bemutatásával, csak egy speciális területen. A formális nyelvek tanulmányozásának legtöbb eszköze szemléletesen irányított gráfokkal jeleníthető meg, futásuk pedig karaktorsorok megjelölt formáival.

Hasonló módon sikeres lehetne egy geometriai algoritmusok, gráfalgoritmusok, vagy a mesterséges intelligenciához kapcsolódó keresések működését bemutató program.

A speciális vizualizációs programok képességeit a DAVIK rendszerbe úgy szándékozzuk átvinni, hogy a megjelenítés rendszere elkülönített nézetablakokon keresztül zajlik. Ezeknek az ablakok biztosítják a speciális programok által is nyújtott előnyöket.

2.4.3. Jeliot 3

[web4] Vizualizációs program, mely automatikusan képezi le a program állapotát egy grafikus felületre.



3. ábra: a Jeliot rendszer bemutatja a buborékrende­zés működését

Érdekes­se, hogy a programkód megadásán kívül semmilyen lépés nem szükséges az algoritmus megjelenítéséhez. A programot java nyelven adhatjuk meg. A megjelenítési terület négy részre van osztva. A bal felső része a hívási vermet mutatja be, a bal alsó része a konstansokat és statikus adatokat. A jobb felső részen a jelenlegi kifejezés kiértékelését követhetjük nyomon. A jobb alsó részen pedig az objektumok (rekordok) és a tömbök láthatóak.

Tapasztalataim szerint a felület egyenlőtlenül van kihasználva. A konstansok és statikus adattagok területén szinte soha nem látható semmi. Véleményem szerint a kifejezések lépésenkénti kiértékelése túlzottan dagályossá teszi a futtatást. Szerintem a program használatának legnagyobb nehézsége, hogy nem lehet gyorsan haladni a program során, és nem lehet szabályozni a megjelenő adatokat.

A program vizualizációs lehetőségeinek automatikus működéséből következik az a hátránya, hogy nincs lehetőség arra, hogy kiemeljük a vizsgált algoritmus egyedi jellegzetességeit.

Az események megjelenítése tetszetősen animált, az egész egy olyan környezetet hoz létre, amiben a fiatalabbak is vizsgálhatják az algoritmusok működését. Ennek viszont ellent mond a java nyelv használata, mellyel általában csak később találkozunk az informatikusi pályára lépők. A java nyelven való programbevitel lehetővé tenné, hogy összetettebb algoritmusok is használhatóak legyenek, ezt viszont a megjelenítés szépen animált formája nehezkesse tesz. Ez a két funkció egymásnak ellentmondó elképzeléseket testesít meg a környezet használatáról.

Az előzőek ismeretében nem meglepő, hogy ahogy az Törley Gábor cikkében [alg38] is olvasható, a rendszer használata elsősorban a figyelemzavaros és a közepes képességű tanulók esetén bizonyult hatásosnak.

Működése jól példázza azt, hogy jól használható lehet egy olyan rendszer, melyben mindenféle megjelenítési utasítás nélkül megadott programkódból animációt tud készíteni.

3. Az AV elméleti megközelítése

3.1. Az AV folyamatának résztvevői

[av9, av30] Zárójelbe tettük azt a kifejezést, ahogy az angol nyelvű szakirodalomban szerepel.

Szoftverfejlesztő (visualization software developer): Elkészíti, vagy továbbfejleszti a vizualizációs rendszert. A szoftverfejlesztő számára a legfontosabb, hogy a rendszer fenntartható és bővíthető legyen, így megváltozott követelményeknek megfelelően is fejleszthető.

Algoritmusfejlesztő (programmer): Megírja az algoritmusokat, melyekhez a vizualizáló az animációt megalkotja. Számára az a lényeges, hogy az algoritmus megadása gyorsan és egyszerűen történhessen, lehetőség szerint ne kelljen egy számára idegen nyelvet megtanulni hozzá.

Animátor (visualizer): Beállítja a vizualizáció megjelenítésének paramétereit. Az animátor munkája egyesíti a szoftverfejlesztő és az algoritmusfejlesztő eredményeit. Fontos számára, hogy a vizualizációra már egy elkészült általános eszköztárt használhasson, és az algoritmusok egyedi jellegzetességeit is megjeleníthesse.

Felhasználó (user): Megtekinti a vizualizációt, szabályozza a megjelenítést és lépteti az algoritmust. Számára a rendszer lényeges tulajdonsága a könnyű használhatóság, átláthatóság és interaktivitás.

3.1.1. A DAVIK koncepciója az AV résztvevőiről

Az animátor közvetítő jellegű feladatkörét három részre osztva a másik három szereplő között felosztásra került. A szoftverfejlesztő feladatává válik a rendszert úgy kialakítani, hogy az egy absztrakt algoritmus futtatása során a megjelenítést automatikusan előállíthassa. Az algoritmusfejlesztő feladata, hogy a rendszer által adott vizualizációs megoldásokból összeállítson egy alapértelmezett vizualizációt, amely szerinte oktatási szempontból előnyös. A felhasználó pedig úgy finomítja az alapértelmezett megjelenítést, hogy az a saját személyes gondolkodásmódjához minél jobban illeszkedjen.

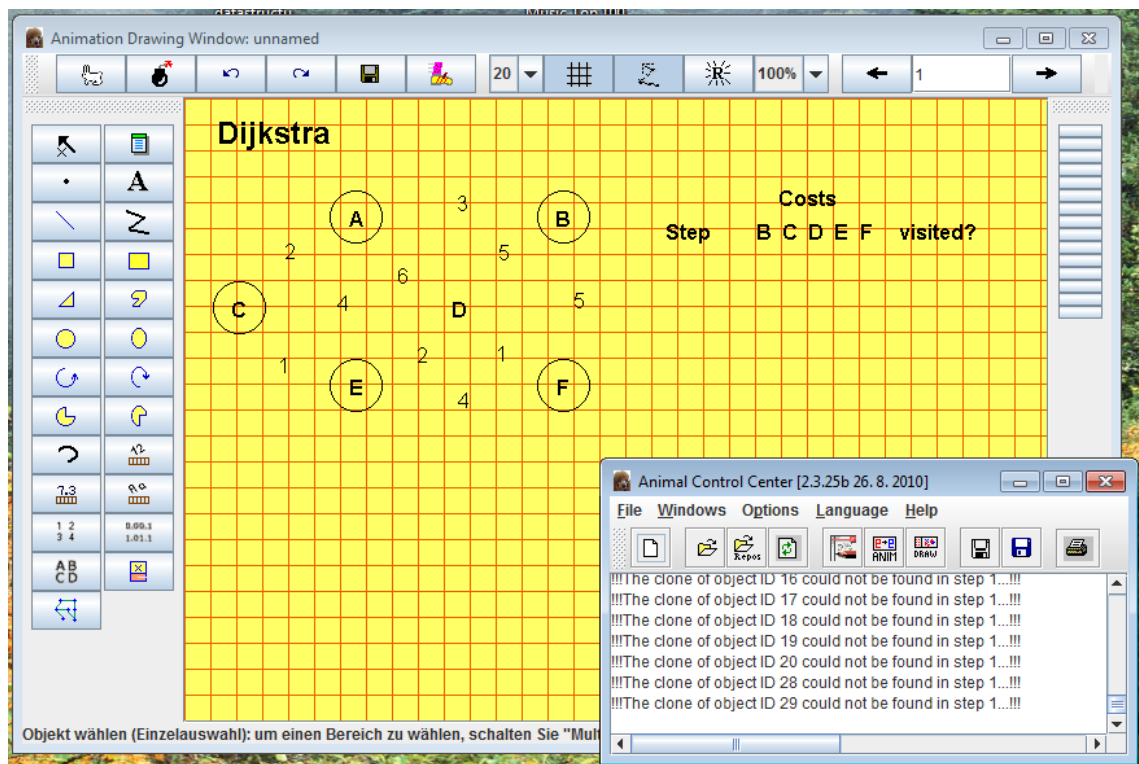
A DAVIK koncepciójának másik alapvető vonása, hogy összemossa az algoritmus készítésének és megtekintésének folyamatát. Így az algoritmus bevitele is a tanulás szerves részévé válik. Az alternatív algoritmusokkal való próbálkozás és a hibás változatok vizsgálata is fontos részévé válik a tanulási folyamatnak.

3.2. Vizualizáció készítési módszerek

Alapvető forrásunk volt Rößling összefoglaló céllal is készített műve [av4].

3.2.1. Animáció generálás grafikus felületen keresztül

Gyakran prezentációkészítő programokat alkalmaznak erre a feladatra, mint a Microsoft PowerPoint. A legtöbb ilyen program támogatja az animációk készítését, de nem ez a célzott felhasználási módjuk. Itt a **szoftverfejlesztő** alkotja meg a grafikus szerkesztőprogramot. Ami lehet speciális, vagy általános célú program. Az **algoritmusfejlesztő** elkészíti az algoritmus modelljét, amit az **animátor** a grafikus felületen láthatóvá tesz.



4. ábra: az Animal rendszer [av6] grafikus animáció szerkesztő alkalmazása

Ez a módszer az animátortól nem követel meg semmilyen speciális ismeretet, csak az irodai programcsomag vagy a célprogram használatának képességét. Ezt a módszert használva szinte korlátlan lehetőségei vannak a megjelenítést testreszabására, például szabadon megválaszthatja a kívánt absztrakciós szintet, kiegészítheti megjegyzésekkel. Mivel nem kell törődni futási követelményekkel, könnyen bemutathatók a nem működő algoritmusváltozatok, gyakori hibák is. Ebből persze következik, hogy a

helyesnek készített animációba is kerülhet hiba, és a program nem fog semmilyen jelzést adni.

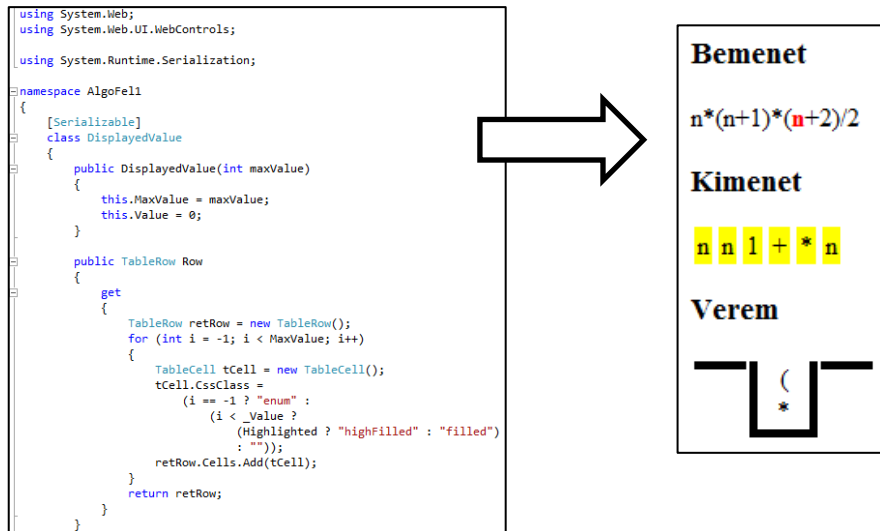
A módszer legfőbb hátránya az animációk készítésének lassúsága. A grafikus felületen való animálás jellegéből következően nem alkalmas ismétlődő, szabályokat követő folyamatok gyors megjelenítésére. Az animáció elkészítésekor a bemeneti adatok is rögzítve lesznek, és azokat később nem lehet változtatni. Így nagyon sok függ attól, hogy a kiválasztott bemenet jól reprezentálja az algoritmus lehetséges bemeneteit, és se nem túl hosszú, se nem túl rövid.

Elméletileg lehetőség lenne az ilyen programok által létrehozott állományok programmal történő generálására, de ez jellemzően kivitelezhetetlen, vagy nem praktikus, mert a fájlformátum nem nyílt, vagy áttekinthetetlen. További probléma, hogy az animátor nagy szabadsága miatt az elkészült animációk mind egyediek, a bemutatott algoritmusok összehasonlítása körülményes.

3.2.2. Vizualizációs program megírása

A programinterfészen keresztüli animáláshoz hasonlóan, itt is egy program állítja elő a vizualizációt. Azonban ebben az esetben a vizualizációs interfészt az adott algoritmus megjelenítésének igényeihez alakítjuk ki, nem különül el olyan mértékben. Az oktatási célra készített vizualizációk többsége nem valamilyen keretrendszert használva lett megírva, egy adott algoritmust (vagy több, hasonló módon megjeleníthető algoritmust) mutatnak be, ehhez tartalmazzák az algoritmikus és vizualizációs elemeket egyaránt.

Bármennyire is egyszerűnek tűnik a feladat, nem szabad lebecsülni az ilyen program algoritmusának implementációs nehézségét. Saját tapasztalatból tudom, hogy az eseményvezérelt megvalósítás (mely a léptethető algoritmus megvalósításához szükséges) nem illeszkedik ahhoz a gondolkodásmódhoz, amiben az algoritmusokat elsajátítjuk. Erre a problémára technikai megoldás lehet az, hogy a programot külön párhuzamos szálon futtatjuk, és a vizualizáció léptetését szinkronizációs műveletekkel valósítjuk meg. Ehhez viszont a párhuzamos programozásban szerzett tapasztalatok szükségesek.

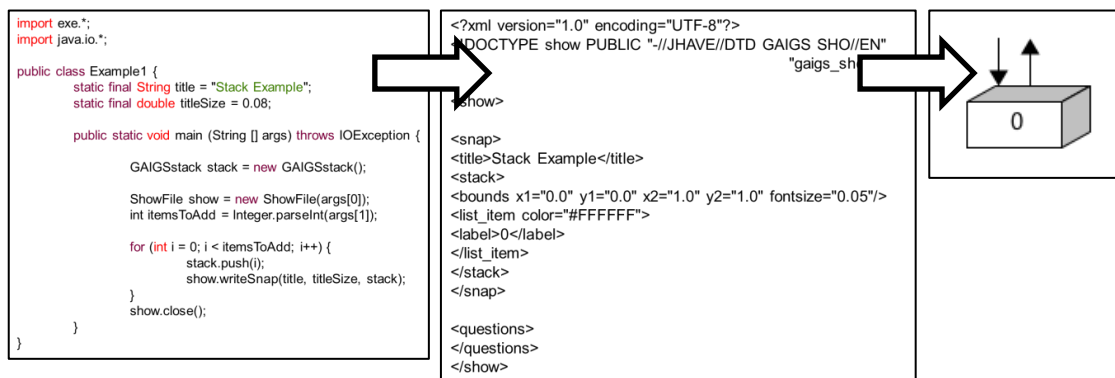


5. ábra: A lengyel formára hozás egy implementációja önálló programként (asp.net implementációval)

Mivel ebben az esetben a vizualizációs rendszer az algoritmus igényre szabott, a megjelenítési lehetőségek változatosabbak. Ugyanebből az okból kifolyólag az elkészítés lassabb, hiszen magát a vizualizációs rendszert is minden alkalommal újra kell implementálni.

A vizualizációs program megírója egy személyben egyesítheti az **algoritmusfejlesztő** és az **animátor** feladatait. Azonban az is lehetséges, hogy az algoritmus kódja és a megjelenítéshez szükséges eszközök külön-külön készülnek el.

3.2.3. Animáció generálás programinterfész segítségével



6. ábra: GAIGS kód generálás programinterfész segítségével, majd megjelenítés [av20]

Ez a módszer abból áll, hogy előállítunk egy programot, mely egy speciális vizualizációs programfelület eljárásainak meghívásával előállítja a vizualizációt. Ezek a hívások történhetnek implicit vagy explicit módon. Az implicit módszer megvalósítása úgy történik, hogy a standard objektumokat, amikkel az algoritmus dolgozik, olyan

objektumokra cseréljük, amelyeken végzett műveleteknek vizualizációs mellékhatásai is vannak. Az utóbbi esetben közvetlen függvényhívásokon keresztül végeztethetjük el a megjelenítést.

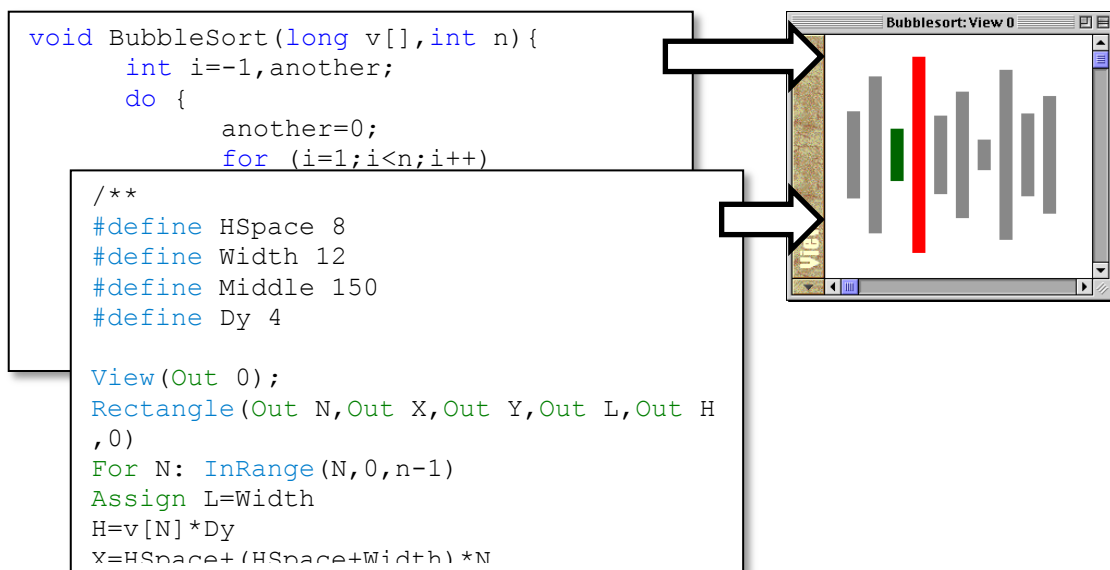
A **szoftverfejlesztő** elkészíti az animációs könyvtárat, mely az animációgenerálást végzi. Az **algoritmusfejlesztő** implementálja az algoritmust, majd az **animátor** ezt kiegészíti az animációs könyvtárhoz intézett közvetlen vagy közvetett hívásokkal.

Elvileg nincs semmilyen akadálya, hogy a program többszöri futtatásával különböző bemeneti adatokra is előállítsuk a vizualizációt, bár ennek az erőforrásigénye nagy lehet. Egy jó animációs könyvtár biztosít lehetőséget a különböző jelelölésekre, megjegyzések bevitelére.

A programfelület segítségével történő animációgenerálás jellegéből adódóan programozási ismereteket igényel. Általában az algoritmusfejlesztőnek és az animátornak azon a nyelven kell programoznia, amin a szoftverfejlesztő a programfelületet megalkotta.

3.2.4. Leírónyelvvvel történő vizuális leképezés

Ebben az esetben, a generálás során egy leképezést adunk meg, mely az algoritmus állapotterének minden állapotához hozzárendel egy vizuális megjelenítést.



7. ábra: algoritmus és vizuális leképezés megadása a Leonardo rendszer [web9] segítségével

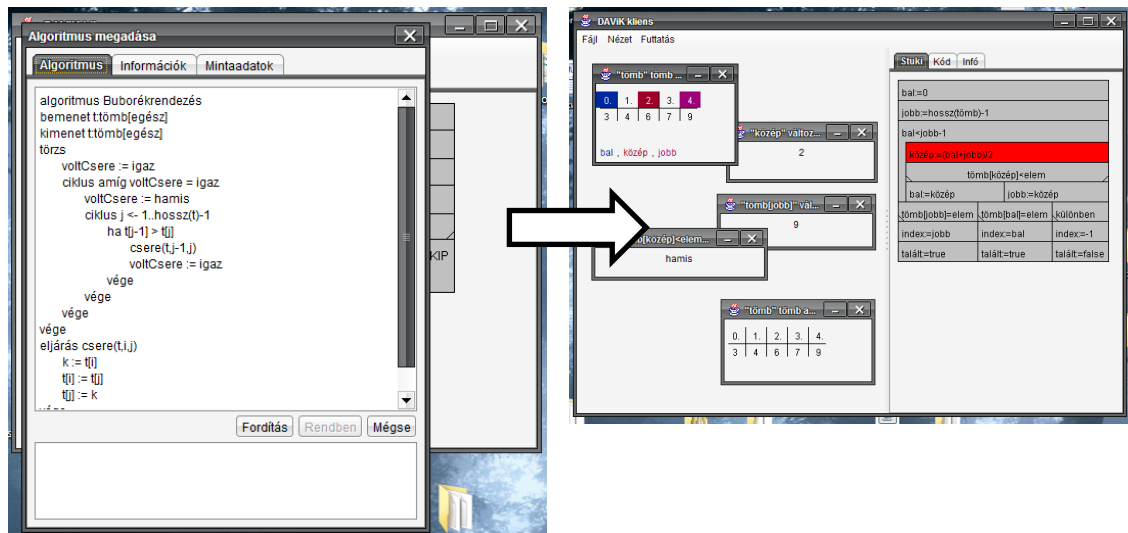
A módszer alkalmazása esetén a **szoftverfejlesztő** megalkotja a programot, mely egy leíróformátummal megadott leképezést ténylegesen elvégez. Az **algoritmusfejlesztő** implementálja az algoritmust, az **animátor** pedig összeállítja a leképezést.

A módszer elve az, hogy az animátor feladatainak egy részét a szoftverfejlesztőre helyezi át. Az olyan leírónyelv, mely a gyakorlatban is használható, sajnos meglehetősen nehezen érthető és használatának elsajátításához is komoly befektetett munka szükséges. A leképezést megadó leíróformátum gyakran az algoritmus forráskódjába kerül beágyazásra, például kommenteken belül.

A programinterfész alapján való animációgenerálástól abban különbözik, hogy itt az állapot leképezése az animátor által részletesen definiálható, és nem feltétlenül kötődik az egyes, az algoritmusban szereplő adatszerkezetek beépített megjelenítési módjától.

3.2.5. Animációgenerálás kódértelmezéssel

Az interpreter, ami a kódot értelmezi, lehet egy ismert nyelvhez készített hibakereső (debugger) alkalmazás, vagy egy teljesen új nyelvhez készített értelmező.



8. ábra: a DAVIK rendszer algoritmus szerkesztője és vizualizációs felülete

A **szoftverfejlesztő** elkészíti az interpretert. Az **algoritmusfejlesztő** megírja a kódot, amit az interpreter értelmez. Az **animátornak** itt másodlagos feladatai vannak, például a megjelenítés testreszabása, ha szükséges.

A legnagyobb előnye a kód értelmezésével végzett animálásnak, hogy a kód és a vizualizációja szoros kapcsolatban van egymással, így szinte lehetetlen, hogy a megjelenítés hibássá, vagy elavulttá válik. Továbbá az animációk készítéséhez az algoritmus, és az algoritmus megadására használt nyelv ismeretén kívül nincs szükség semmilyen gyakorlatra, a vizualizáció gyorsan előállítható.

Az értelmezett kód többféleképp is felhasználható, viszonylag egyszerűen készíthető belőle tetszőleges nyelvnek megfelelő kód, struktogram vagy más ábra, illetve megjeleníthető vizuálisan, de akár önmagában is futtatható.

A kód és az animáció ilyen szoros kapcsolatának azonban ára van, az megjelenítés absztrakciós szintje nagyon nehezen befolyásolható. Továbbá nehéz befolyásolni, hogy az algoritmus melyik lépéseit szeretnénk látni, és melyiket nem.

A DAVIK rendszer megvalósításához a kódértelmezéssel történő vizualizációt választottuk. A könnyű és sokféle felhasználási mód, illetve az algoritmusok megadásának könnyedsége kárpótol a bemeneti nyelv értelmezéséből következő megnövekedett bonyolultságért.

3.2.6. Összefoglalás

Az alábbi táblázatban összefoglalom az algoritmus vizualizációk készítési módszereit:

	dinamikus*	szoftverfejlesztő	algoritmusfejlesztő	animátor
Animáció generálás grafikus felületen keresztül	sem az adatok, sem a nézet	animációs rendszer és vizuális felület	algoritmusmodell	animáció elkészítése a vizuális felületen
Animáció generálás programinterfész segítségével	igen	könyvtár	algoritmus a könyvtár nyelvén	animációs utasítások a könyvtár ismeretében
Vizualizációs program megírása	külön munkával mindkettő lehetséges	-	algoritmus a könyvtár nyelvén	vizualizáció rendszere
Leírónyelvvél történő vizuális leképezés	az adatok igen, a nézet nem	leképezés értelmezése + vizualizáció	algoritmus a könyvtár nyelvén	megjelenítési forma megadása
Animációgenerálás kódértelmezéssel	az adatok és a nézet is dinamikussá tehető	fordító + futtatási rendszer + vizualizáció	algoritmus az értelmezett nyelven	vizualizáció testreszabása

*: Egy rendszert az adatok szempontjából akkor tekintünk dinamikusnak, ha egy adott algoritmus animációját tetszőleges bemenettel is tesztelhetjük, nézet szempontjából pedig akkor, ha lejátszáskor a megjelenítendő komponensek kiválaszthatóak, testreszabhatóak.

Forrásunk külön tárgyalja azokat a módszereket is, melyek egy szkriptet állítanak elő, amit később egy animációs rendszer megjelenít. Ez a módszer azonban bármely animáció-generálási módszerhez hozzáilleszthető. Előnye az utólagos szerkeszthetőség, és az animáció (tömörebb) elmenthetősége. Hátránya a megnövekedett komplexitás.

3.3. Az Algoritmus Vizualizáció részei

Mi az, amit meg kell jeleníteni ahhoz, hogy az algoritmus érthető legyen? Egy adott pillanatban az algoritmus futásának végeredményét két dolog határozza meg, az algoritmusban szereplő adatok értéke, és az, hogy milyen utasításokból áll az algoritmus.

3.3.1. Az állapottér megjelenítése

Az algoritmus állapotának vizualizációjához meg kell jeleníteni az algoritmusban szereplő változók (adatelemek) értékét.

Az algoritmus változóinak típusát a következők közül szoktuk választani: logikai, egész szám, valós szám, szöveges karakter vagy adatszerkezet, ha a változó skaláris. Továbbá használhatunk összetett változókat, melyek lehetnek különböző, de fix számú, névvel indexelt tag rekordjai, vagy fix elemszámú, számmal indexelt tömbjei.

Ezek azok az adatstruktúrák, amik majdnem minden számítógépes programozási nyelvben megvalósíthatók, mert a gépi kód segítségével könnyen lehet velük dolgozni. Emiatt, ha ezeket az adatstruktúrákat vesszük alapértelmezett, beépített szerkezeteknek, akkor nem veszítünk az általánosságból.

Az összetett és skaláris változók közötti különbség az, hogy az előbbinél foglalkozunk az alkotóelemekkel külön is. Tehát egy egész számot összetett típusnak kell tekinteni, ha az egyes jegyeivel (vagy biteivel) akarunk műveleteket végezni, és egy szöveges adat tekinthető skalárisnak, amennyiben a karaktereket nem vizsgáljuk, vagy változtatjuk külön.

Az adatszerkezetek viselkedését a vezérlő algoritmusai határozzák meg. Amikor egy adatszerkezet szerepel az algoritmusunkban, mint változó, a háttérben az adatszerkezetnek megfelelő adatrepresentáció helyezkedik el.

Fontos, hogy az adatok megjelenítése összhangban legyen az algoritmus szerkezetének megjelenítésével. Az egyes adatok neve egyezzen meg az adatok algoritmusban szereplő jelölésével.

3.3.2. Algoritmikus szerkezet megjelenítése

Az algoritmust strukturált formájában akarjuk megjeleníteni. Az algoritmus strukturált formája alatt olyan algoritmust értünk, ami egyrészt egyes utasításokból, elágazásokból és ciklusokból áll. Ez a definíció még nem zárja ki, hogy bizonyos utasítások hatással legyenek a vezérlésre, azaz meghatározzák, hogy mely utasítások kerülnek végrehajtásra. Például sok esetben növeli az átláthatóságot, ha egy egyszerű utasítással ki tudunk lépni egy ciklusból, ahelyett, hogy egy összetettebb vezérlési struktúrát alkalmaznánk.

Az algoritmusok szerkezetének megjelenítésére két módszer terjedt el. Az egyik az, hogy szöveges leírásként, vagy pszeudokódként jelenítsük meg az algoritmus működését, a másik pedig hogy valamilyen diagramként tesszük ezt.

A pszeudokód a program működésének meghatározott stílust követő szöveges leírása. A pszeudokód a tényleges implementációs programkódnál kevésbé formálisabb (magasabb absztrakciós szinten van), de a működés egyszerű szöveges leírásánál strukturáltabb. A pszeudokód előnye a szabad szöveges leírással szemben, hogy egy adott lépés a pszeudokódban kijelölhető, és maga a pszeudokód automatikusan előállítható.

A diagramként való ábrázolás történhet folyamatábrákkal, melyekben a működés egyes lépéseit különböző alakzatok jelölik, a közöttük lévő átmeneteket pedig nyilak. Ilyen például az UML activity diagramja [hiv1].

A másik gyakran használt módszer a Nassi–Shneiderman diagram (vagy struktogram), mely az algoritmus szerkezetét egymásba ágyazott utasítástömbökön keresztül mutatja be [web1]. A struktogram különösen alkalmas az algoritmus szerkezetének megjelenítésére, mert az utasítás horizontális pozíciójával, és a blokkok bekeretezésével is jelzi, hogy az utasítás melyik blokkhoz tartozik. Továbbá előállítása is egyszerűbb, mint a folyamatábraké.

Annak érdekében, hogy a megjelenítés interaktív legyen, hasznos funkció a program aktuális lépésének kiemelése a képernyőn. Ezáltal bármely pillanatban gyorsan meg

lehet határozni a program állapotának legfontosabb paraméterét, a következő utasítást.

3.3.3. Az algoritmus háttere

A HalVis kísérletsorozat [av17] 7. és 8. kísérletében Hansen és társai azt vizsgálták, hogy a vizualizáció egyes elemeinek hiánya mennyire csökkenti a hatékonyságot, és hogy ezek az elemek önmagukban milyen pedagógiai hatással bírnak.

Az ő rendszerüknek három fő komponense volt, a koncepcionális nézet egy analógiával magyarázta az algoritmus absztrakt működését, a részletes nézet az algoritmus részletes lejátszására adott lehetőséget az algoritmus állapotát és pszeudokódját. A benépesített nézet pedig az algoritmus gyakorlati működését mutatta, egy nagy adathalmazzal, főként a hatékonyságra figyelve.

A hetedik kísérlet eredménye azt mutatta, hogy a koncepcionális nézet hiánya okozta a legnagyobb, 34%-os teljesítménycsökkenést, szemben a másik két nézet elhagyásával, melyek csak 10%-os (koncepcionális n.), illetve 8%-os (benépesített n.) csökkenést okoztak.

Ebből látszik, hogy milyen fontos az, hogy az algoritmus részletes működésének bemutatása mellett, az algoritmus elvéről és hátteréről is közöljünk információt a tanulóval. Többek között ezen eredmények hatására döntöttünk úgy, hogy az algoritmushoz tartozó információk megjelenése a DAVIK rendszerben kiemelt fontossággal bír.

A nyolcadik kísérlet azt vizsgálta, hogy önmagában az egyes nézetek milyen hatékony pedagógiai eszközök. Itt a részletes nézet bizonyult a leghasznosabbnak, melynek használatával a teszten a teljes rendszer használatához képest csupán 10% csökkenés volt tapasztalható.

Ez az eredmény megerősített minket abban az elképzelésünkben, hogy a vizualizációt az adatok és az algoritmus szerkezetének megmutatására alapozzuk.

3.3.4. Az információk integrációjának fontossága

Hansen és társainak kísérletei [av17] egyértelműen arra mutatnak, hogy több ismeret-átviteli médium együttes használata jobb eredményhez vezet. Az ötödik kísérlet során

az egyik csoport számára a HalVis integrált tanulási környezetet biztosították, míg a másik csoportnak külön szöveges magyarázatot és animációt adtak. Az integrált környezet használói 18%-al többet értek el, mint a kontrollcsoport. Ennek az eltérésnek nagyon valószínű magyarázata, hogy a több integrált csatornán az ismeretek könnyebben eljutnak a tanulóig, mintha független forrásokból jutna hozzájuk.

Amennyiben a tanuló az algoritmus működése közben tanulmányozhatja annak leírását, akkor az elvi ismereteit és gyakorlati tapasztalatait összevetheti.

3.3.5. Vezérlés

A vizualizáció vezérlésének több módszere ismert. A vizualizáció által mutatott állapot befolyásolása többféle szinten is megvalósulhat. Elképzelhető, hogy a vizualizáció ilyen lehetőséget nem nyújt, ekkor animációként jelenik meg az algoritmus működése.

A megjelenítés irányíthatóságának első lépcsőfoka, ha van egy előre gomb, ami egy lépést végez el az algoritmus működéséből. Lehet továbbá hátra gomb, ami visszafelé lépteti az animációt, illetve még lehetnek olyan gombok, melyek bizonyos kódszakaszokat ugranak át.

A Hansen tanulmány 6. kísérlete azt az elméletet igazolja, hogy az algoritmus felhasználói utasításra történő lépésenkénti végrehajtása sokkal érthetőbbé teszi azt, mint a folyamatos, animációszerű lejátszás. Az a csoport, akiknek animációként játszották le a vizualizációt 20%-al kevesebb pontot ért el, mint akik irányítottan tekintették meg.

Ha a felhasználó megadhatja az algoritmus bemenetét, akkor ezzel tanulmányozhatja az algoritmus működését olyan esetekben is, melyekre a vizualizáció készítője nem gondolt.

A megjelenítés testreszabása segíti a felhasználót abban, hogy az algoritmust több absztrakciós szinten tanulmányozva megértse annak koncepcióját.

3.4. Milyen algoritmusok mutathatók be az AV segítségével?

3.4.1. Rendezések

Törley Gábor egy előadásában [av19] amellett érvel, hogy a rendezési problémák jó témái az informatika oktatásának. Érvei között szerepel, hogy ez az algoritmuscsoport kezdők és haladók számára is érdekes feladatnak számít, és hogy nem egy helyes megoldás létezik, hanem több olyan megoldás, melyek bizonyos helyzetekben előnyösebbek a többinél.

Valóban igaz, hogy a rendezések működésének megértése nagyon hasznos mindenki számára, aki programozással vagy algoritmusokkal akar foglalkozni. Az algoritmus-szakértő Steven S. Skiena négy indokot sorol fel arra [hiv2], hogy miért olyan jó a rendezésekkel foglalkozni:

- A rendezések fontos szerepet játszanak több algoritmusban is, mivel szabályossá teszik az adatokat, megkönnyítik az őket követő lépéseket.
- Az algoritmusokkal kapcsolatos főbb elvek közül sok a rendezésekkel kapcsolatban merült fel, és velük is mutatható be a legegyszerűbben. Ilyenek például az „oszd meg és uralkodj” algoritmusok, az adatszerkezetek és a véletlenített algoritmusok.
- A világ számítógépei az informatika története során a rendezéssel foglalkoztak a legtöbbit. A szerverek lefutott processzorciklusainak a negyede rendezésre lett fordítva.
- A rendezés az egyik legtöbbit tanulmányozott informatikai probléma. Több tucatnyi algoritmus ismert, melyek közül mind előnyös valamilyen helyzetben.

Ezek az elméleti előnyök a gyakorlatban is láthatóak, egy széleskörű felmérésben [av5] szereplő algoritmus vizualizációk több mint harmada foglalkozott a rendezésekkel. Ez elég meggyőző arány, a lehetséges problémák változatosságának fényében.

Vizualizációs szempontból is kedvező ezekkel az algoritmusokkal foglalkozni, mert nem kell hozzájuk sok különbözőféle megjelenítési funkció. Az egyszerűbb rendezések (buborékrendezés, beszúrásos rendezés, minimumkereséses rendezés) szemléltetéséhez elegendő egy tömb, amiben meg tudjuk jelölni azokat a pontokat,

amikkel éppen az algoritmus foglalkozik. A rekurzív rendezések (összefésülő rendezés, gyorsrendezés, ...) bemutatásához szükséges egy olyan megjelenítő eszköz, mely képes követni a rekurziót, azaz be tudja mutatni, hogy hogyan áll elő a részproblémák megoldásaiból a végleges megoldás. Bizonyos rendezésekhez, például a kupacrendezéshez, vagy a versenyrendezéshez a leghasznosabb, ha egyszerre tudjuk megjeleníteni a tömböt, és az annak megfelelő fát.

3.4.2. További algoritmuscsoportok

Véleményem szerint a gráfalgoritmusok is az AV rendszerekben jól megjeleníthető algoritmusok közé tartoznak. Egy gráfmegjelenítő módszer szükséges a használatukhoz, melyen hasznos, ha megjelölhetők a csúcsok és élek, illetve rájuk illeszthetők különböző jelölések, például az élsúlyok.

Geometriai algoritmusok esetében különösen fontos, hogy komplex kifejezéseket adhassunk meg rendszernek, melyeket kiértékelve jelenít meg. Így megjeleníthetünk néhány számadat alapján két vagy háromdimenziós alakzatokat, egyeneseket, síkokat, anélkül, hogy az algoritmusban az adott formában előfordulnának.

A geometriai algoritmusok vizualizációjának használhatóságát alátámasztja például az izraeli kutatók által létrehozott MAVIS rendszer [av2]. Ezzel például robotirányító algoritmusokat mutatnak be grafikus módon, meglehetősen áttekinthetően. A Vega egy korábban elkészült, szerver-kliens architektúrával rendelkező vizualizációs program, ami teljes mértékben a geometriai algoritmusok megjelenítésére koncentrál [av3].

3.4.3. Az AV alkalmazhatósága

Egyszerűbben foglалható össze, hogy milyen algoritmusokra nem használható az AV. Ott, ahol a feladatnak megoldásának nincs olyan grafikus megjelenítése, mely közelebb vinne a feladat megértéséhez. Kérdéses, hogy ilyen algoritmus egyáltalán létezik-e, hiszen a grafikus megjelenítés elég tág fogalom.

Sokkal jobban szűkíti a lehetőségeinket, ha azt vizsgáljuk, hogy milyen algoritmusok szemléltetésére nem megfelelő egy általános rendszer, ahol véges számú beépített vizualizációs módszer áll rendelkezésünkre. Egy olyan algoritmus, melynek az

eddigiektől teljesen eltérő megjelenítési módszerre van szüksége, csak nehezen illeszthető be egy ilyen rendszerbe.

Példaként az euklideszi algoritmust tudom felhozni, két egész szám legnagyobb közös osztójának meghatározására. Ennek az algoritmusnak az egyetlen olyan hatékony vizualizációját találtam, mely világosan szemléltette a módszer elvét. Ez a csempézési módszer, amiben a két szám legnagyobb közös többszörösének megtalálásához a megfelelő hosszúságú oldalakkal rendelkező téglalapot fedjük le négyzetekkel (a maradékos osztást reprezentálva) [web8]. Azonban nem tudok más olyan algoritmusról mely szintén téglalapokra helyezett négyzetekkel lenne bemutatható.

Jellemzően nehezen megjeleníthető algoritmusok a különböző matematikai eljárások, ahol az absztrakció magas foka nem segít a vizuális megjelenítésben (próbáljunk például egy n -dimenziós teret elképzelni), illetve a nagy mennyiségű adattal dolgozó statisztikai, adatbányászati algoritmusok.

4. A DAVIK rendszer kidolgozásának vezérelvei

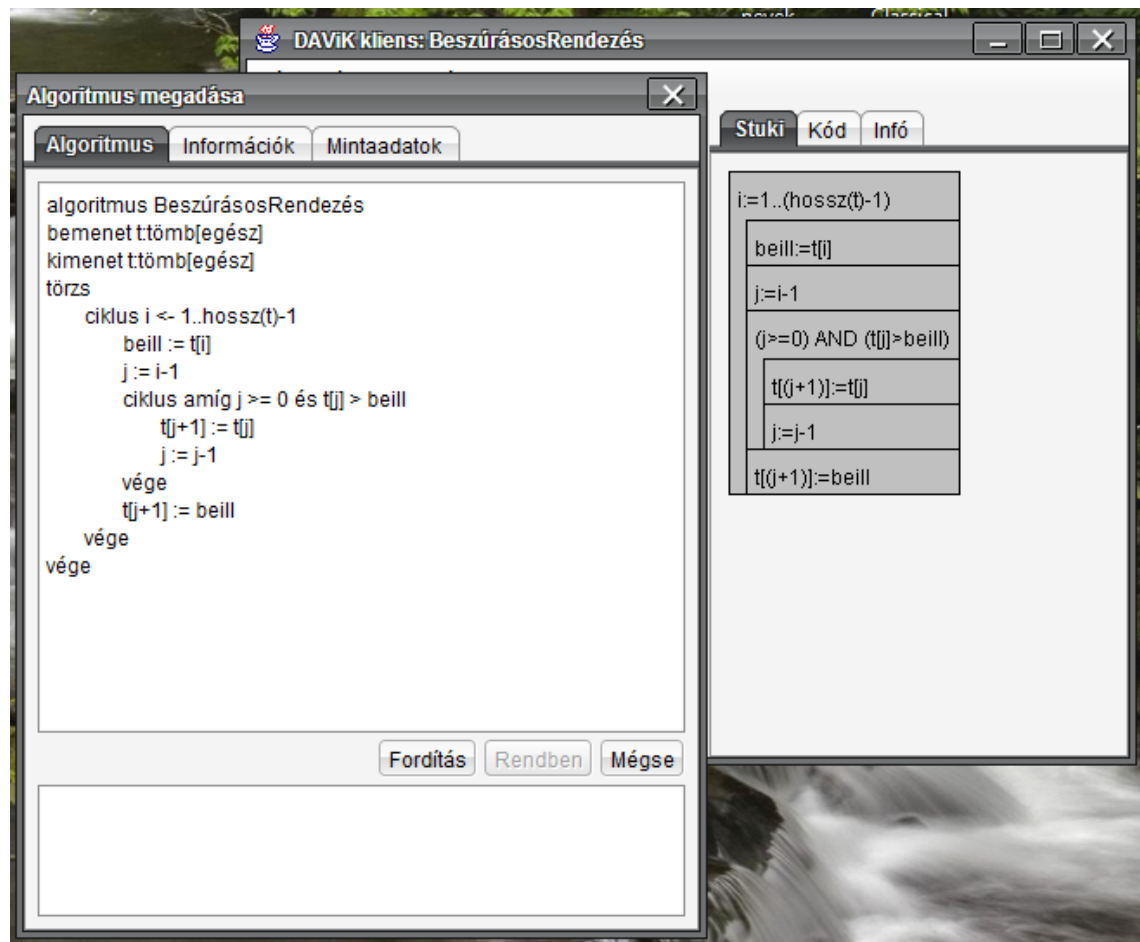
4.1. Egyszerű és gyors algoritmuskészítés

Ezt a lehetőséget még egyetlen már létező AV rendszernél sem fedeztem fel. E tekintetben két főbb ága van az AV rendszereknek.

Az első csoportnál a vizualizáció előállítója közvetlenül az animációt készíti el. Ebben az esetben nem igazán lehet könnyű algoritmuskészítésről beszélni. Az algoritmus minden lépését manuálisan kell bevinni, ami nem igazán segíti a hosszabb animációk összeállítását.

A második lehetőség az animáció generálása. Ennek módszere többféle is lehet. Amennyiben könyvtári függvények hívásával, vagy vizualizációt végző adatszerkezetek használatával érjük el, akkor ennek a rendszernek a nyelvét és használatát kell elsajátítani. A generálás történhet úgy is, hogy az algoritmust megvalósító programkódba illesztenek bele olyan annotációkat, melyek a megjelenítést elvégzik. Ez viszont az algoritmusok létrehozását egy adott nyelvhez köti, és még egy beágyazott nyelv használatát is kötelezővé teszi, ez tovább növeli az algoritmusok készítésének elsajátításához szükséges időt.

Annak érdekében, hogy ezt a problémát feloldjuk, a második lehetőséget választottuk, de azt úgy valósítottuk meg, hogy a generálás kellően absztrakt módon történhessen. Az algoritmusok létrehozására egy pszeudokód nyelv használható, melyet a rendszer szimulációval bármilyen adaton el tud végezni. Ennek előnye, hogy könnyebb az elsajátítása, és mind szintaxisában, mind szemantikájában jobban illeszkedik az algoritmikus gondolkodáshoz, mint az algoritmus valamely, adott nyelvben történő implementációja. Továbbá egy absztrakt leírása az algoritmusnak, amit pszeudokódként még az algoritmus megjelenítésének eszközeül is választhatunk.



9. ábra: Pseudokód és a belőle létrejött algoritmus stuktogramja a DAVIK rendszerben

4.2. Algoritmus-leíró pszeudokód

Az, hogy az algoritmusokat pszeudokóddal, és nem programkóddal kell megadni, lehetővé teszi, hogy egy adott nyelvben járatlan oktatók és hallgatók is készíthessenek vizualizációkat. Kiküszöböli az implementációs részleteket, így az algoritmus létrehozásának folyamata is gyorsabbá válik. A pszeudokód szerkezetét úgy határozzuk meg, hogy minél jobban illeszkedjen az algoritmikus gondolkodáshoz és a konvencionális jelölésekhez.

Sokan foglalkoztak már azzal, hogy oktatási szempontból milyen programozási nyelv a leghasznosabb. Nyilvánvaló a párhuzam az oktatási cél, és az animációk készítése között, hiszen itt is olyan nyelvet szeretnénk használni, ami olvasható, áttekinthető és könnyen megírható kódot eredményez.

Azonban nem szabad figyelmen kívül hagynunk az eltéréseket a programozás és az algoritmusok megadása között. Könnyen előfordulhat, hogy a programozással még csak ismerkedő tanulók programjai is több száz sorosak lehetnek. A bonyolultság abból ered, hogy valós használati eseteket kell lehetővé tenni. Szükség van konverziókra, kivételkezelésre, a felhasználóval való kommunikációra. Az algoritmusok viszont egy-egy probléma absztrakt megoldását jelenítik meg, így rövidebbek lehetnek. Nehéz lenne példát találni olyan, hagyományos egyetemi oktatásban tanított algoritmusra, mely pszeudokóddal csak több mint száz sorban adható meg.

A terjedelmi különbségek miatt az algoritmusok megadásához jobban illeszkedik a szkriptnyelvek rugalmassága és tömörsége, mint a programozási nyelvek szigorú szabályai és következetessége. A szkriptnyelvekhez hasonlóan itt is a lehető leggyorsabbá kell tenni a kód létrehozását.

4.2.1. Programozási nyelvek az oktatásban

Törley Gábor szakdolgozatában [av18] a pascal nyelv használata mellett érvel. Véleménye szerint az oktatásban használt nyelvnek elsősorban világos alapszavakkal kell rendelkeznie. Műveletkészletének egyszerűnek kell lennie, ellentétben például a processzorközelibb c nyelvvel, ahol sok operátor közvetlenül gépi utasításokat visz nyelvi szintre (inkrementálás, bitműveletek).

Továbbá felhívja a figyelmet a következetesség fontosságára. A következetességnek több formáját is kiemeli, például a kötött deklarációs szabályok a következetességet növelik, mert minden változót egy adott helyen kell felsorolni, így tudjuk, hogy az algoritmusban milyen változók szerepelnek. Véleményem szerint igaza van abban, hogy a szigorú deklarációs szabályok is a következetességet növeli, viszont szerintem fölöslegesen növeli a program megírásának nehézségét.

Itt ugyan ellenpéldaként szerepel a pascal, mivel benne például a vessző és pontosvessző használata összekeverhető, kezdők számára nem egyértelmű.

Véleményem szerint nem tulajdonít elég nagy jelentőséget a programok könnyű megírásának. A pascal programok szerintem túl terjedelmesek, túlságosan nagy a kódnak az információt nem hordozó része. Figyelembe kell venni, hogy Törley Gábor az általában vett programozásoktatásról beszél, mi pedig az algoritmusok megadásának legjobb módszeréről.

4.2.2. A pszeudokód jellegzetességei

A pszeudokód természeténél fogva egy kötetlen formája az algoritmus megadásának. Azonban mégis vannak alapelvei a pszeudokód írásának. Ezt jól példázza a wikipédia ajánlása pszeudokód írására [web6]. A wikipédián több mint száz algoritmus leírásához tartozik pszeudokód [web5].

Azért hasznos a pszeudokóddal foglalkozni, mert az algoritmus megadásának természetes módja, nem követeli meg semmilyen konkrét programozási nyelv ismeretét.

A leírás elsősorban a tiszta stílusra hívja fel a figyelmet, hogy kerüljük az implementációra utaló szerkezeteket, és a valamely létező nyelvvel való hasonlóságot. A magas szintű pszeudokódra koncentrál, melyben könnyen fejezi ki az algoritmus lényeges vonásait.

Az ilyen magas szintű pszeudokód már nem követi a szigorú szintaktikát, bár kifejezéseit át lehetne írni függvényhívásokká. Jellemzően felsorolósos ciklusszerkezeteket használ.

Más pszeudokód írási javaslatok [web7] a következőkre hívják fel a figyelmet: Ajánlatos a műveleteket szövegesen kiírni. Például a & jelnél érthetőbb az „and” szó, mint logikai művelet. Természetesen ezzel a technikával is vigyázni kell, mert az összetettebb formulák könnyen nagyon hosszúvá válhatnak, ha mindent szövegesen írunk ki.

Mi ezt megfontolva úgy döntöttünk, hogy a DAVIK rendszer algoritmusmegadó pszeudokódjának logikai műveleteit szöveges operátorokkal használjuk, míg az aritmetikai műveletekhez meghagyjuk azok szokványos jelölési rendszerét.

4.2.3. A DAVIK rendszer pszeudokódjának alapelvei

Az alapvető célunk, amit a pszeudokód használatával el szeretnénk érni, hogy a tanuló az algoritmust rövidebb idő alatt vihesse be a rendszerbe, mint ahogyan azt implementálni tudná, és sokkal gyorsabban el lehessen sajátítani, mint egy programozási nyelvet.

Minél egyszerűbb szintaktikát választottunk, mert így a lehető legkevesebb idő megy el a szintaktikus hibák javításával, és a nyelv is könnyebben található. Így rövidebb, áttekinthetőbb is lesz a kód, és gyorsabban is megírható.

Fontos kérdés volt a pszeudokóddal kapcsolatban, hogy milyen stílust kövessen a szintaktikája. Itt a hagyományos procedurális szintaktika mellett döntöttünk. Erre alapvetően két okunk van. Elsőként, az informatikus hallgatók elsőként a procedurális programozási modellel ismerkednek, annak egyszerűsége miatt. Másodszor pedig a klasszikus algoritmusok a tankönyvekben is a procedurális gondolkodásmódot követve kerültek leírásra. Emellett a procedurális modell implementálása is egyszerűbb, kevésbé növeli a rendszer bonyolultságát.

Hogy gyorsítsuk az algoritmusok létrehozását, a szkriptnyelvek mintájára nem vezetünk be deklarációs szabályokat, az egyes változókat egy egyszerű értékadással létre lehet hozni. Ugyanígy, nem végzünk típusellenőrzéseket sem. Az egyetlen hely, ahol jelezni kell a bemeneteket, kimeneteket és azok típusait, az maga az algoritmus.

```
algoritmus Buborékredezés
bemenet t:tömb[egész]
kimenet t:tömb[egész]
törzs
    voltCsere := igaz
    ciklus amíg voltCsere = igaz
        voltCsere := hamis
        ciklus j <- 1..hossz(t)-1
            ha t[j-1] > t[j]
                csere(t,j-1,j)
                voltCsere := igaz
            vége
        vége
    vége
vége
```

10. ábra: A DAVIK rendszer pszeudokódja Notepad++ szerkesztőprogramban

4.3. A rendszer és a felhasználó interakciója

A leginkább egyértelmű eredménye az AV témakörében folytatott kísérleteknek az, hogy az animáció önmagában nem hatásosabb oktatási médium, mint a tankönyv. Az eredmények azt mutatják, hogy a vizualizációk csak abban az esetben hatékonyak, amikor segítségére vannak a hallgatónak abban, hogy az algoritmus működésének mentális modelljét fölépítse. Tehát a vizualizáció nem a tudás hordozója, hanem eszköz, amivel a megtekintő megértheti az eljárás elvét. [av1]

A megtekintő és a vizualizáció közötti interakció lehetséges formái: az animáció futásának az irányítása (1), a bemeneti adatok felhasználó általi meghatározása (2), az animáció közben föltesztelt kérdések (3), az animáció elvégeztetése a felhasználóval (4), a vizualizáció megjelenésének, nézőpontjának meghatározása (5), és az algoritmusok felhasználói megadása (6).

Ezek közül az 1. minden kidolgozottabb vizualizációnak része. A 2.-at azok a rendszerek teszik lehetővé, ahol az algoritmus kódját és nem a végrehajtás lépéseit adja meg a vizualizáció létrehozója. A 3. lehetőséget leginkább nagyobb vizualizációs keretrendszerekben, mint például a Jhave [av13], tapasztaltam. A 4. lehetőséget egyedül a TRAKLA2 [av15] rendszerben vettem észre, ahol nagy hangsúlyt fektettek arra, hogy a tanulók tudásszintjének vizsgálatára is lehessen a rendszert használni. Az 5. pont megvalósítására csak nagyon korlátozott próbálkozásokat találtam, a nézet testreszabását a nagy rendszerek tervezői az animátor feladatának tekintik. A 6. interakciós formát például a Jeliot [web4] rendszer biztosítja.

A DAVIK rendszert úgy terveztük meg, hogy a felhasználói interakció minél változatosabb formáit támogassa. A vizualizáció léptetését (1) ezért előre és hátra egyaránt el lehet végezni. A bemeneti adatok teljesen szabadon megadhatók (2), de lehet az előre rögzített mintaadatokból is választani. A vizualizáció lejátszásakor a felhasználó kiválaszthatja, hogy milyen változók, kifejezések értéke érdeklí, és ezeket figyelni meg, tetszőleges elrendezésben (5). Végül, az éppen megfigyelt algoritmust bármikor módosíthatja, és újraindíthatja, illetve tetszőleges új algoritmust is megadhat (6).

4.4. Vizualizációs ablakok

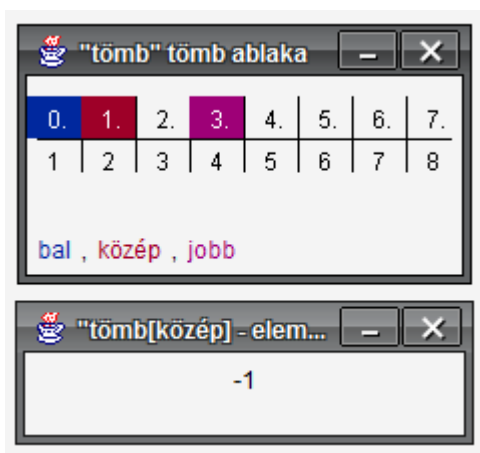
A DAVIK rendszerben a vizualizációs ablak az algoritmus megjelenítésének alapja. Ez az objektum az algoritmus állapotterének egy nézőpontját képviseli. Például külön vizualizációs ablak használható arra, hogy a egy tömbben megjelenítsük az éppen vizsgált elemeket, egy másik pedig csupán az elemeinek szöveges formáját mutatja, hogy a rendezettség mértékét követni lehessen.

A vizualizációs ablak egyszerre több változó értékét is megjelenítheti. Ennek gyakori formája az, hogy a változó megjelenítését más változókból képzett jelölések teszik érthetőbbé. Lehetőség van arra is, hogy egy változót egyszerre több vizualizációs ablakon keresztül is megjelenítsünk.

A rendszer lényeges pontja, hogy a vizualizációs ablakok elkészítése minél könnyebb legyen. Fontos, hogy a rendszer vizuális képességei növelhetőek legyenek. Ha ez lehetővé válik, akkor informatika egyes részterületeihez tartozó algoritmusok speciális vizuális igényei is kielégíthetők, és ezzel valóban univerzális vizualizációs rendszer alkotható meg.

Ahhoz hogy ez megvalósulhasson, a megjelenítést teljes mértékben függetlenné tettük az algoritmus futtatásától. A megjelenítő egységek adatváltozási események formájában értesülnek, amikre válaszul frissítik a megjelenített képet.

A vizualizációs ablakok megjelenítési lehetőségei nem csupán a változókra terjednek ki, a segítségükkel tetszőleges kifejezés értéke megjeleníthető. A megjelenített kifejezések automatikusan frissülnek, amikor egy tagjuk értéke megváltozik.



11. ábra: Vizualizációs ablakok

4.5. Egységes működés és összevethetőség

A világhálón nagyon nagy mennyiségű algoritmus vizualizációs anyag halmozódott fel a 1990 óta. Sajnos, minőségük igen vegyes képet mutat [av5]. Bár van közöttük igen sok pedagógiai értékkel bíró program is, viszont még ezek használatát is nehezíti az egységesség hiánya. Mivel mind különböző kezelőfelülettel rendelkezik, minden algoritmus megtekintése az adott program megismerésével kell, hogy kezdődjön. Továbbá az algoritmusok összehasonlíthatóságát csökkenti, hogy teljesen különböző felületeken tanulmányozhatók.

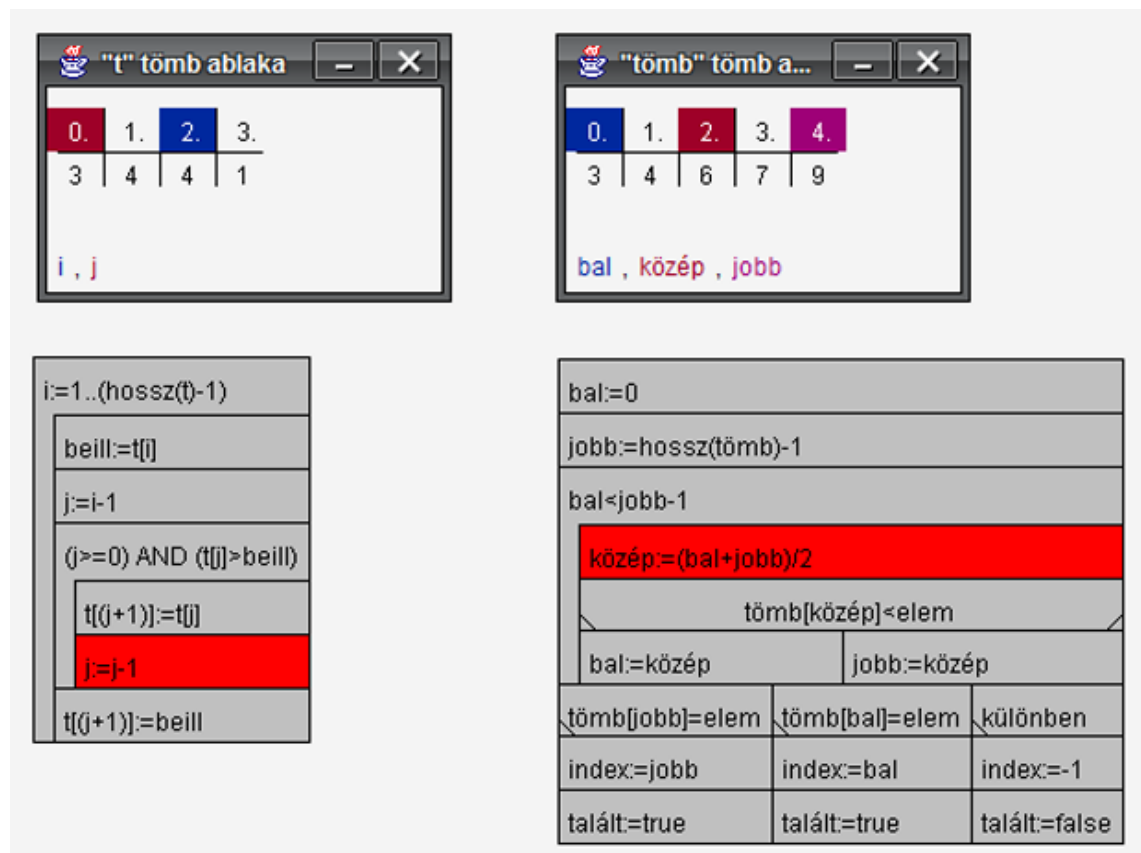
Insertion Sort i A S R T I N G E j v O <pre> public void insertionSort(int[] a){ int i, j, v; // counter; v contains an element for (i=1; i<a.length; i++){ v = a[i]; for (j=i; j>-1 && a[j-1]>v ; j--){ a[j] = a[j-1]; // Move element right } a[j] = v; // Insert element in sorted array } } </pre>	Binary Search search for : 2 Position : 4 a 1 3 4 5 6 8 9 11 12 0 1 2 3 4 5 6 7 8 <pre> public int binSearch(int l, int r, int x, int[] a) { int midElem = (l+r)/2; if (a == null a.length == 0 r < l) return -1; if (a[midElem] == x) return midElem; else if (x < a[midElem]) return binSearch(l, midElem - 1, x, a); else return binSearch(midElem + 1, r, x, a); } </pre> isSmall() solve() 9 divide() 0 divide() 8 combine() is empty
---	--

12. ábra: Egy rendező és egy keresőalgoritmus az Animal rendszerben: nehezen összevethető vizualizációk

Ahhoz, hogy az egységességet elérhessük, az algoritmus animátorának feladatait szétosztjuk a rendszer és a felhasználója között. Az olyan dolgokat, mint az alapvető megjelenítési egységek (nézetablakok), a rendszer biztosítja. Míg a megjelenés pontos szabályzását a felhasználó saját maga szabhatja testre, az alapértelmezett beállításokat az algoritmus létrehozója állítja be. (Ennek implementációja még nem készült el.) Az alapvető megjelenítési komponenseket pedig a rendszer egységes formában biztosítja. Ezzel a felosztással az adott algoritmus jellemzőinek kiemelésére, és az egységes, összehasonlítható megjelenítésre egyszerre van lehetőség.

Az összehasonlíthatóság másik fontos eleme az algoritmusok adatbázisa. Itt a megfelelő rendszerezés segíti az algoritmusokkal foglalkozó tanulót, hogy már azelőtt lásson bizonyos összefüggéseket az algoritmusok között, mielőtt azokat betöltené a

vizualizációs kliensbe. Ennek az adatbázisnak a megvalósítása jelenleg nem teljes, de az előkészületek megtörténtek ennek a teljessé tételéhez.



13. ábra: Az előző két algoritmus vizualizációja a DAVIK rendszerben: könnyen látható a megjelenítés hasonlósága

4.6. Információk és vizualizáció integrálása

A vizualizációs környezet, amit létre akarunk hozni, két lényeges vizualizációs komponenst jelenít meg. Ezek az algoritmus állapotterének vizualizációja és az algoritmus struktúrájának megjelenítése. A kettő közül az algoritmus állapotának megjelenítése a domináns, hiszen az a legteljesebb formája az algoritmus állapotának megjelenítésének. Ez a testreszabható a két megjelenítési komponens közül.

Ebbe a környezetbe még több másodlagos vizuális komponens is integrálásra került. Ilyenek az algoritmus elméleti hátterét és főbb tulajdonságait ismertető panel, de tervezzük az éppen aktuális lépést magyarázó szöveges segítség implementációját is.

4.7. Hordozhatóság

Szeretnénk, ha a rendszer a lehető legtöbb platformon lenne használható. Továbbá előnyös lenne, ha a rendszer használata minél gyorsabban elkezdhető lenne, nem kellene telepíteni.

Ezeknek a céloknak az elérésére a Java virtuális gép által nyújtott absztrakció segítségével fejlesztjük a vizualizációs rendszer kliens és szerveroldali komponenseit, melyek így minden olyan platformon futtatóak lesznek, ahol java virtuális gép rendelkezésre áll. Ezzel a törekvésünkkel nem vagyunk egyedül, a Java rendszer elterjedésétől kezdve az AV-k többsége erre a platformra lett kifejlesztve. [av5]

Külön funkció, hogy nincs elkülönített kliens és szerver program, ugyanazt az állományt más paraméterekkel futtatva működtethetünk szervert is. Ezt a funkciót a hordozhatóság és egyszerű használat segítségével valósítottuk meg.

A széles körű használhatóság érdekében a programot úgy valósítottuk meg, hogy a használt nyelv konfigurálható legyen, és könnyedén lehessen új nyelvi fájlt létrehozva a programot lefordítani. Ehhez a Java fejlesztői csomag internacionalizációt segítő szolgáltatásait használtuk fel. Jelenleg a program magyar és angol nyelvre lett lefordítva.

5. Összefoglalás

A dolgozat első részében tanulmányoztuk az algoritmus vizualizáció hatását a tanulásra és a már létező AV rendszerek használatát és működését.

A következőket állapítottuk meg:

- 1) Önmagában az algoritmusok animálása nem növeli jelentős mértékben az oktatás hatékonyságát, de az AV rendszerek képesek a tanulóval való interakcióra, melynek pozitív hatása kimutatható.
- 2) Az animáció kidolgozottságának növelése általában nem jár a hatékonyság növekedésével, a többféle interakciós lehetőség viszont igen.
- 3) Ha több nézőpontra keresztül mutatjuk be egy algoritmus működését, akkor annak megértése is könnyebb lesz.

A különböző rendszerek tanulmányozása után ezeket a rendszereket osztályoztuk, aszerint, hogy milyen módon lehet bennük vizualizációkat készíteni. Ilyen szempontok alapján beszéltünk grafikus felületen keresztüli animálásról (1), egyedi vizualizációs programokról (2), programinterfészrel történő animálásról (3), leírónyelvvél történő vizuális leképezésről (4), kódértelmezéssel történő animációról (5).

Megállapítottuk, hogy az első két osztályba tartozó programok egyedi animációkat hoznak létre, melyek nagy rugalmassággal bírnak, ám hiányzik belőlük az egységesség és a szoros kapcsolat az algoritmussal. A 3.-4. osztályokba tartozó programok jobban automatizálhatók és használatuk gyorsabb, ám ezekben az esetekben komolyabb programozói tudás kell az algoritmusok létrehozásához. Az 5. osztály AV programjainak implementációja bonyolultabb, ám a segítségükkel programozás nélkül is létre lehet hozni az algoritmussal szorosan együttműködő vizualizációkat.

A következő második részében a DAVIK rendszer tervezési elveit és főbb funkcióit ismertettük.

A rendszer tervezésének legfőbb elvei az algoritmusok könnyű elkészíthetősége, amit segít a pszeudokód, a hordozhatóság, amit a java alapon történő implementáció garantál. Továbbá a vizualizációk egységes megjelenése, amit elsősorban azzal értünk el, hogy a megjelenítés létrehozását a rendszer maga végzi el. Ez segíti a különböző

információk integrálását is. A felhasználóval való interakciót nem megjelenő tesztkérdések formájában, hanem a testreszabás és kísérletezés lehetőségeiben látjuk, és ennek megfelelően valósítottuk meg a rendszert.

6. Irodalomjegyzék

A rövidítések jelentése:

- [web]: Interneten keresztül elérhető anyagok, különböző felhasznált programok honlapjai, wikipédia cikkek az egyes fogalmak értelmezéséhez, az interneten található tartalmak változékony természete miatt dátummal jelölve.
- [av]: Az algoritmusvizualizáció tudományos irodalma.
- [hiv]: Referenciákra, specifikációkra, szakmai könyvekre való hivatkozások.

[web1]: Wikipedia: Nassi–Shneiderman diagram, URL:

http://en.wikipedia.org/wiki/Nassi%E2%80%93Shneiderman_diagram, Utolsó letöltés: 2011. 11. 20.

[web2]: The Eclipse Foundation open source community website, URL: <http://eclipse.org/>, Utolsó letöltés: 2011. 11. 20.

[web3]: JFLAP, URL: <http://jflap.org/>, Utolsó letöltés: 2011. 11. 20.

[web4]: Jeliot 3, URL: <http://cs.joensuu.fi/jeliot/index.php>, Utolsó letöltés: 2012.04.20

[web5]: Algoritmusok leírásai a wikipédián, melyekhez tartozik pszeudokód, URL: http://en.wikipedia.org/wiki/Category:Articles_with_example_pseudocode, Utolsó letöltés: 2012. 04. 20

[web6]: A wikipédia alapelve a pszeudokód írására, URL:

http://en.wikipedia.org/wiki/Wikipedia:Algorithms_on_Wikipedia#General_guidelines_for_writing_pseudocode, Utolsó letöltés: 2012. 04. 20

[web7]: Pseudocode Standard, URL: http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html, Utolsó letöltés: 2012. 04. 20

[web8]: Euklideszi algoritmus vizualizációja a Wikipédián, URL:

http://en.wikipedia.org/wiki/File:Euclidean_algorithm_1071_462.gif, Utolsó letöltés: 2012. 04. 20

[web9]: Leonardo IDE: C Compiler and Software Visualization System, URL:

<http://www.dis.uniroma1.it/~demetres/Leonardo/>, Utolsó letöltés: 2012. 04. 20

[av1]: Christopher D. Hundhausen, Sarah A. Douglas, John T. Stasko (2002): A Meta-Study of Algorithm Visualization Effectiveness, In Journal of Visual Languages & Computing, Vol. 13, No. 3. (June 2002), pp. 259-290

[av2]: Igal Koifman, Ilan Shimshoni, and Ayellet Tal. (2002): MAVIS: A Multi-Level Algorithm Visualization System within a Collaborative Distance Learning Environment. In Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02) (HCC '02). IEEE Computer Society, Washington, DC, USA, 216-.

- [av3]: Christoph A. Hipke, Sven Schuierer (1999): Vega - A user-centered approach to the distributed visualization of geometric algorithms
- [av4]: Guido Rößling, ANIMAL-FARM (2008): An Extensible Framework for Algorithm Visualization, VDM Verlag Dr. Muller Aktiengesellschaft & Co. KG
- [av5]: Clifford A. Shaffer, Matthew Cooper, Stephen H. Edwards: Algorithm Visualization: A Report on the State of the Field, In SIGCSE '07 Proceedings of the 38th SIGCSE technical symposium on Computer science education
- [av6]: Guido Rößling, Bernd Freisleben (2002): Animal: A System for Supporting Multiple Roles in Algorithm Animation, Journal of Visual Languages & Computing, Volume 13, Issue 3, June 2002, p 341-354
- [av7]: M. D. Byrne, R. Catrambone & J. T. Stasko (1999) Evaluating animations as student aids in learning computer algorithms. Computers & Education, Volume 33, Issue 4, 1 December 1999, p 253-278
- [av8]: A.W. Lawrence (1993) Empirical studies of the value of algorithm animation in algorithm understanding. Unpublished Ph.D. dissertation, Department of Computer Science, Georgia Institute of Technology.
- [av9]: Purvi Saraiya (2002): Effective Features of Algorithm Visualizations, In Proceedings of the 35th SIGCSE technical symposium on Computer science education (SIGCSE '04). ACM, New York, NY, USA, 382-386.
- [av10]: Ronald Baecker and Blaine Price. The Early History of Software Visualization. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price (editors), Software Visualization, chapter 2, pages 29–34. MIT Press, 1998.
- [av11]: Ronald Baecker and Dave Sherman (1981): Sorting out Sorting. 30 minute color film, Dynamic Graphics Project, University of Toronto. Distributed by Morgan Kaufman Publishers.
- [av12]: Marc H. Brown and Robert Sedgewick (1984): A System for Algorithm Animation Structures. In ACM SIGGRAPH '84 Proceedings, Minneapolis, Minnesota, pages 177–186, July 1984.
- [av13]: Thomas L. Naps, James R. Eagan, Laura L. Norton (2000): JHAVE - An Environment to Actively Engage Students in Web based Algorithm Visualizations, Lawrence University, SIGCSE '00, Proceedings of the thirty-first SIGCSE technical symposium on Computer science education
- [av14]: Guido Rößling, Peter Schroeder (2008): Animalipse - An Eclipse Plugin for AnimalScript, Electronic Notes in Theoretical Computer Science, Volume 224, 1 January 2009, p 3-14
- [av15]: Ari Korhonen, Lauri Malmi, and Panu Silvastu (2003): TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. In: Proceedings of Koli Kolistelut / Koli Calling – Third Annual Baltic Conference on Computer Science Education. Joensuu, Finland, pp. 48–56.

[av16]: D. J. Jarc, M. B. Feldman, R. S. Heller (2000): Assessing the benefits of interactive prediction using web-based algorithm animation courseware. In: Proceedings SIGCSE 2000. ACM Press, New York

[av17]: S. R. Hansen, N. H. Narayanan & D. Schrimsher (2000) Helping learners visualize and comprehend algorithms. Interactive Multimedia Electronic Journal of Computer-Enhanced Learning

[av18]: Törley Gábor (2005): Középiskolai programozás oktatás vizuális környezetben, szakdolgozat, ELTE IK Informatika Szakmódszertani Csoport

[av19]: Törley Gábor (2007): Rendezések oktatási lehetőségei az informatika tantárgyban, Savaria Konferencia 2007., Szombathely, URL: <http://pezsgo.web.elte.hu/publikaciok/rendezesek.zip>, Utolsó letöltés: 2012. 04. 20

[av20]: Ismeretlen szerző (2006): Generating GAIGS XML Scripts I, Grand Valley State University, URL: <http://jhave.org/workshops/Workshop06/GeneratingGAIGSscripts1.ppt>, Utolsó letöltés: 2012. 04. 20

[av21]: Andreas Fest and Ulrich Kortenkamp (): Teaching graph algorithms with Visage, In: Teaching Mathematics and Computer Science, 2009, Vol. VII, Issue I, pp. 35-50

[hiv1]: Unified Modeling Language Specification, Version 1.4.2, ISO/IEC 19501:2005, (<http://www.omg.org/spec/UML/ISO/19501/PDF/>)

[hiv2]: Steven S. Skiena (2008): The Algorithm Design Manual, Second Edition, Springer Science+Business Media