Modelarea interactivă vizuală a algoritmilor prin forme metaforice

Liviu Daniel Safta
Universitatea Tehnică din Cluj-Napoca,
Facultatea de Automatică şi Calculatoare
Str. G.Bariţiu nr. 26-28
davonkeep@yahoo.com

Dorian Gorgan
Universitatea Tehnică din Cluj-Napoca,
Facultatea de Automatică şi Calculatoare
Str. G.Bariţiu nr. 26-28
dorian.gorgan@cs.utcluj.ro

REZUMAT

În acest articol este descrisă o metodă didactică în programarea calculatoarelor, metodă ce presupune modelarea interactivă vizuală a algoritmilor şi evaluarea bazată pe forme metaforice. Am descris pe scurt abordări similare şi am subliniat îmbunătăţirile aduse de către metoda propusă. Am evidenţiat succesiunea stărilor de reprezentare ale unui algoritm, realizând o descriere formală corespunzătoare fiecărei stări. În final, am exemplificat elementele teoretice expuse în lucrare printr-un studiu de caz.

Cuvinte cheie

Modelare interactivă vizuală, forme metaforice, proces didactic, gândire algoritmică, limbaj de programare.

Clasificare ACM

K. Computing Milieux, K.3 COMPUTERS AND EDUCATION, K.3.1 Computer Uses in Education, *Computer-assisted instruction (CAI).*

INTRODUCERE

Conform [1], "Cursurile de programare se concentrează adeseori pe sintaxă şi pe caracteristicile particulare ale unui limbaj de programare, determinând studenţii să se concentreze pe aceste detalii relativ neimportante în loc de competenţele algoritmice de bază [….] Multe dintre limbajele utilizate pentru programarea obiectuală în industrie – în special C++, dar într-o anumită măsură şi Java – sunt semnificativ mai complexe decât limbajele clasice. Dacă profesorii nu au mare grijă să îşi prezinte materia într-un mod care să limiteze această complexitate, asemenea detalii îi pot copleşi cu uşurinţă pe studenţii de la cursurile de începători."

Chiar şi în momentele când profesorul încearcă să concentreze lecţia pe concepte teoretice importante ale claselor sau algoritmilor studiaţi, o mare parte din lecţie este orientată spre clarificarea dificultăţilor sintactice întâmpinate de studenţi [2].

Felder [3] remarcă faptul că majoritatea studenţilor învaţă vizual, în timp ce profesorii tind să prezinte informaţia în mod verbal. Între 75% şi 83% din studenţi învaţă vizual [4,5]. Scanlan [6] a spus că pentru majoritatea studenţilor, limbajele de programare tradiţionale, de natură textuală, nu oferă un cadru intuitiv pentru însuşirea conceptelor de programare orientată pe obiect şi a stimulării gândirii algoritmice. Scanlan a arătat că studenţii înţeleg mai bine algoritmii prezentaţi în flowcharturi decât pe cei prezentaţi în pseudocoduri. Carlisle et. al [7] a indicat că, dacă li se oferă oportunitatea, 95% dintre studenţi aleg să exprime algoritmii cu ajutorul flowcharturilor în defavoarea limbajelor de programare tradiţionale, chiar şi atunci când majoritatea instrucţiunilor le-au fost date într-un limbaj tradiţional. Câteva studii [7,8,9] au arătat că studenţii au performanţe mai bune la curs atunci când suportul materiei predate sunt limbaje de programare iconice.

Medii didactice imersive

Majoritatea mediilor folosite în prezent în didactica programării calculatoarelor sunt medii didactice evidenţiate printr-o componentă predominant imersivă, visual-interactivă [10]. Conform [11], educaţia prin imersiune combină grafica interactivă, tehnologia de simulare (inclusiv jocuri comerciale), realitatea virtuală, voice chatul şi mediile digitale cu medii colaborative de cursuri online şi săli virtuale de clasă.

Numeroase opinii din literatură susţin că folosirea acestor medii poate stimula plăcerea, motivarea şi implicarea utilizatorilor, ajutând la reţinerea şi căutarea de informaţii şi poate încuraja dezvoltarea mai multor abilităţi sociale şi cognitive [12,13,14].

Second Life (SL) este un astfel de mediu de educaţie prin imersiune, considerat ca fiind unul dintre cele mai populare. Second Life este un mediu integral 3D, care oferă instrumente de construcţie şi scriere incorporate, ceea ce îl face un mediu de învăţare complet, bazat pe stimularea şi antrenarea capacităţilor logice ale studenţilor [15].

Programarea vizuală

Numeroase dovezi susţineau ideea că studenţii înţeleg mai bine conceptele de programare atunci când li se oferă o reprezentare vizuală [2]. Astfel s-a format o nouă direcţie în didactica programării calculatoarelor, abordare bazată pe modelarea soluţiilor problemelor prin programare vizuală.

Programarea vizuală este acel mod de programare în care, pentru a exprima elementele semantice, se foloseşte mai mult de o dimensiune [16]. Dacă sintaxa unui limbaj de programare conţine expresii vizuale, limbajul se numeşte limbaj de programare vizuală (VPL) [17].

Dintre numeroasele medii de programare vizuală sau iconică dezvoltate, cele mai importante sunt analizate pe scurt în ceea ce urmează.

SFC (Structured Flow Chart) Editor [18] este un editor structurat de flowcharturi. SFC permite dezvoltarea de flowcharturi structurate şi afişează o reprezentare în pseudocod a fiecărui flowchart, în sintaxă C++ sau Pascal, dar nu oferă posibilitatea creării de clase.

Visual Logic [19] este un instrument comercial bazat pe un proiect academic, Flint [20]. Visual Logic suportă crearea de programe cu proceduri multiple, fiecare dintre ele fiind reprezentată ca un flowchart. La fel ca SFC, Visual Logic nu suportă crearea de clase.

Alice [21], creat de Carnegie Mellon University, este un mediu de programare 3D foarte răspândit care sprijină predarea conceptelor introductive de programare într-un mod obiectual. Studenţii creează animaţii plasând obiecte într-o lume virtuală 3D, iar apoi le programează comportamentul. Cu toate că Alice utilizează terminologia obiectuală, nu suportă în mod direct moştenirea [22].

Iconic Programmer [23] şi B# [8] sunt două alte instrumente care permit studenţilor să creeze programe utilizând flowcharturile. Acestea suportă operaţii de intrare/ieşire, selecţie, iteraţie şi generarea de cod, dar nu suportă implementarea de subprograme.

Raptor [2] este un instrument open-source care suportă complet programarea obiectuală, incluzând încapsularea, moştenirea şi polimorfismul. Raptor permite studenţilor să îşi execute algoritmii în mediu, decât să trebuiască să îi compileze separat, şi să îşi execute programele. Acest lucru înseamnă că depanarea se poate realiza pe reprezentarea vizuală a algoritmului, şi nu pe cea textuală şi astfel nu sunt necesare mai multe instrumente. Conform [2], această combinaţie de componente face ca Raptor să fie unic, oferind o funcţionalitate care nu este disponibilă în nici un alt mediu de programare educaţional existent la ora actuală.

Dintre instrumentele software recente ce utilizează o abordare obiectuală, trebuie amintite: BlueJ [9], Java Power Tools [11], Karel J. Robot [2], şi diverse biblioteci grafice. Toate aceste instrumente au o puternică componentă vizuală/grafică pentru a-l ajuta pe începător să „vadă" exact ceea ce este un obiect şi să îşi dezvolte capacitatea de programare orientată pe obiect.

BlueJ [24] oferă un mediu integrat în care utilizatorul începe în general cu un set de clase definit anterior. Structura proiectului este prezentată grafic, în stil UML. Utilizatorul poate crea obiecte şi poate invoca metode pe aceste obiecte pentru a le ilustra comportamentul.

Java Power Tools (JPT) [25] oferă o interfaţă grafică (GUI) cuprinzătoare, interactivă, formată din câteva clase cu care va lucra studentul. Studentul interacţionează cu GUI şi învaţă despre comportamentul claselor GUI tocmai prin intermediul acestei interacţiuni.

Karel J. Robot [26] utilizează o microlume cu un robot pentru a-i ajuta pe studenţi să înveţe despre obiecte. La fel ca în Karel [27], roboţii sunt adăugaţi într-o grilă 2-D. Metodele pot fi apelate pe roboţi pentru diverse acţiuni.

Bruce et al. [28] şi Roberts [29] utilizează bibliotecile grafice în abordarea obiectuală. În acest caz, există o aşa numită pânză pe care obiectele (de exemplu, formele 2D) pot fi desenate. Pe obiecte se pot testa diverse metode.

Un neajuns al mediilor didactice menționate este lipsa unei asocieri clare între modelarea interactivă vizuală a algoritmilor şi pseudocod. Există o verigă lipsă tocmai în faptul că nu se evidențiază legătura dintre interacțiunile utilizatorului cu mediul didactic şi secvența de instrucțiuni generată de acțiunile sale [30].

PROCESUL DIDACTIC

Etapele procesului didactic sunt:

- transmiterea de noi cunoştiinţe, exemplificări şi analogii;
- rezolvarea problemelor;
- evaluarea soluţiilor propuse, realizarea feedback-ului.

Metoda propusă se localizează în etapa de rezolvare a problemelor. Aici apar principalele probleme în procesul didactic, în special pentru studenții începători, şi anume trecerea de la analiza cerinţelor unei probleme propuse spre rezolvare, la obținerea soluției - un algoritm. Chiar daca analiza problemei este realizată optim, cerințele problemei fiind extrem de clare, două întrebări apar mereu:

- Cum se rezolvă problema dată din punct de vedere logic, cum se modelează soluția?
- Cum se codifică modelul logic pentru a obține cod sursă executabil?

DEZVOLTAREA ALGORITMILOR PRIN MODELARE INTERACTIVĂ VIZUALĂ

Pentru a răspunde întrebărilor formulate anterior, metoda propusă presupune transformarea problemei date într-un exemplu concret, un scenariu reprezentat prin forme metaforice. De exemplu, sortarea elementelor unui vector unidimiensional poate fi transpusă metaforic în vizualizarea unor elemente grafice de dimensiuni diferite şi manipularea lor interactivă în vederea sortării în funcție de dimensiune. Un alt exemplu poate fi operația logică de comparare a două variabile. Cele două variabile pot fi vizualizate ca două pahare umplute cu apă, iar prin plasarea lor pe o balanță, sa aibă loc simularea unui test logic, indicând paharul mai greu. Exemplele pot continua cu alte forme metaforice pentru diverse elemente din domeniul programării calculatoarelor.

Conform [30], crearea scenariilor şi a formelor metaforice este o sarcină ce revine celui care crează lecția. Formele metaforice folosesc la modelarea interactivă vizuală a unei soluții logice, fără a avea vreo legătură aparentă cu scrierea algoritmului cerut. Astfel, se evită concentrarea asupra rigorilor sintaxei unui limbaj de programare, urmărindu-se doar aspectul logic, de obținere a unui model metaforic.

Scena

Mediul de lucru folosit în modelarea interactivă vizuală prin forme metaforice este scena [31]. Scena este interfața dintre aplicația didactică şi utilizator, locul în care are loc procesul de modelare.

Elementele funcționale ale scenei sunt obiectele şi zonele de interacțiune. La încărcarea lecției, scena este populată cu obiecte. Utilizatorul vizualizează forma metaforică în care este transpusă problema dată prin obiectele din scenă.

Obiectele

Obiectele sunt elementele de bază ale scenei, fiind manipulate în vederea modelării soluției problemei propuse.

Fiecare obiect este caracterizat prin atribute vizuale și tipuri de interacțiune.

Atribute vizuale. Fiecare obiect este reprezentat vizual în scenă, în primul rând, printr-o imagine. Astfel, se asigură prima interacțiune dintre utilizator și mijloacele de care dispune pentru a rezolva problema. Prin simbolul asociat obiectului, utilizatorul vizualizează mai ușor posibilele roluri pe care le poate îndeplini obiectul în modelarea soluției. Acesta este singurul rol pe care îl are imaginea asociată obiectului.

Alte atribute vizuale pot să apară în funcție de metaforele necesare. De exemplu, etichete indicând valori specifice, valori care transmit informații importante pentru utilizator în luarea deciziilor cu privire la modul în care poate manipula și plasa fiecare obiect în scenă.

Tipuri de interacțiune. Tipurile de interacțiune sunt transmise textual utilizatorului. Ele specifică ce operații sunt posibile asupra fiecărui obiect. De exemplu, operații de deplasare a obiectelor cu mouse-ul (drag-and-drop), suprapunerea obiectelor, declanșarea unor acțiuni etc.

Codificarea elementelor scenei. La nivel funcțional, fiecare obiect este reprezentat printr-un dreptunghi caracterizat de lungime și lățime. Doar aria și poziția fiecărui obiect în scenă sunt importante pentru evaluarea algoritmului modelat. Tipurile de interacțiune sunt implementate corespunzător specificațiilor fiecărui obiect, asigurându-se astfel un minim control asupra acțiunilor stundentului, o măsură de limitare a posibilităților în vederea evitării acțiunilor inutile sau greșite.

Zone de interacțiune

În scenă, tipurile de interacțiune sunt corelate cu zonele de interacțiune. De exemplu, chiar dacă un obiect are ca tip de interacțiune asociată deplasarea în scenă, deplasarea sa necesită existența unei zone în care sa fie plasat, arei numită zonă de interacțiune.

Zonele de interacțiune au o funcție asemănătoare tipurilor de interacțiune asociate obiectelor scenei, limitând spectrul interacțiunilor utilizatorului. În acest mod, se elimină o bună parte din acțiunile inutile sau greșite ale utilizatorului, fiind parțial condus spre un traseu valid.

Plasarea obiectelor în anumite zone de interacțiune este condiționată în general de atribute funcționale precum aria și poziția obiectului. Și etichetele vizuale pot avea un rol dublu, valoarea transmisă în scop cognitiv utilizatorului, având uneori și o semnificație funcțională, cooptată în setul de condiții de acceptare a obiectelor în anumite zone de interacțiune.

Modelarea soluțiilor

Modelarea formelor metaforice este de fapt strâns corelată cu scrierea unui pseudocod. Fiecare acțiune descrisă prin interacțiunile vizuale este asociată unei instrucțiuni pe baza unui set de reguli predefinite. Se disting trei etape principale de dezvoltare interactivă a unui algoritm (Figura 1):

- modelarea interactivă vizuală,
- interpretarea interacțiunilor utilizatorului,
- implementarea algoritmului.

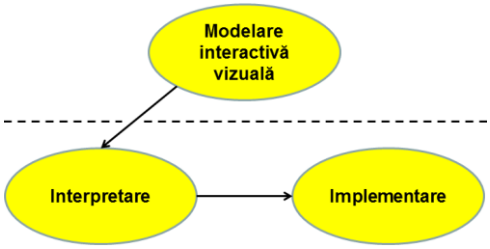

Figura 1 Etapele parcurse în dezvoltarea interactivă a unui algoritm [31]

Utilizatorul are rolul principal numai în etapa de modelare interactivă, vizuală a problemei specificate. Următoarele etape, de interpretare a acțiunilor studenților și asocierea lor cu instrucțiuni, se fac în spatele interfeței grafice, printr-un set de reguli și asocieri predefinite de către profesorul care a descris lecția [30].

Conform [31] celor trei etape ale procesului de dezvoltare interactivă a unui algorim le corespund în rezolvarea problemelor trei componente (Figura 2):

- modelarea interactivă vizuală a soluției,
- interpretarea acțiunilor intreprinse,
- transcrierea acțiunilor interpretate în instrucțiuni.

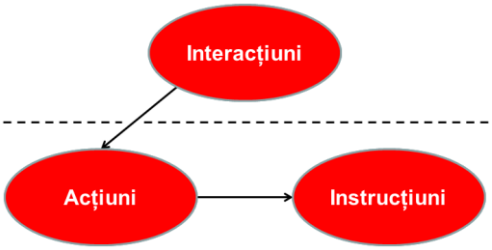

Figura 2 Componentele dezvoltării interactive a unui algoritm

REPREZENTAREA ALGORITMILOR

Urmărind interacțiunea celor trei componente în rezolvarea problemelor, se identifică patru stări posibile în reprezentarea unui algoritm (Figura 3):

1. Reprezentarea metaforică vizuală - modelarea interactivă vizuală prin forme metaforice.
2. Reprezentarea textuală prin secvența de comenzi.
3. Reprezentarea abstractă a algoritmului propus.
4. Reprezentarea codificată, respectând sintaxa unui limbaj de programare.

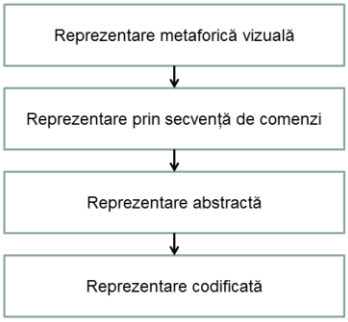

Figura 3 Succesiunea stărilor de reprezentare a unui algorim

Reprezentarea metaforică vizuală

Reprezentarea metaforică vizuală se obține prin modelarea formelor metaforice. În [32] se descrie în detaliu procesul de modelare interactivă vizuală prin forme metaforice. Pe scurt, acest proces vizual de interacțiune a utilizatorului cu scena trasează o secvență de evenimente memorate printr-o serie de interacțiuni (Figura 4).

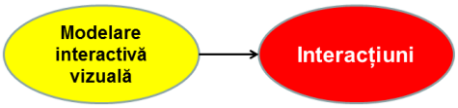

Figura 4 Reprezentarea metaforică vizuală

Un alt rezultat al acesui proces este o formă metaforică finală, un model. Aceasta reprezintă prima stare în reprezentarea algoritmului, reprezentare ce prezintă două mari avantaje:

- posibilitatea trasării logice a algoritmului prin urmărirea secvenței de interacțiuni intreprinse de utilizator în cadrul procesului de modelare interactivă vizuală,
- obținerea unei prime condiții de validare a soluției propuse, prin compararea formei metaforice finale cu un model predefinit, așteptat ca rezultat al procesului de modelare interactivă vizuală.

Reprezentarea textuală

Reprezentarea textuală se face prin secvența de comenzi (Figura 5).

Figura 5 Reprezentarea textuală

Reprezentarea textuală se obține prin interpretarea interacțiunilor utilizatorului cu scena. Fiecare interacțiune din scenariul metaforic este asociată unei acțiuni sau unui set de acțiuni. Cunoscând interacțiunile permise utilizatorului încă din momentul proiectării scenariului, apartenența unei interacțiuni la grupuri de acțiuni se face pe baza unor reguli de predefinite. Întreg procesul de modelare interactivă vizuală a soluției este un proces controlat, parțial direcționat pe traiectorii predefinite.

Rezultatul fazei de interpretare a interacțiunilor, secvența de acțiuni, este reprezentată textual ca o secvență de comenzi. Relația dintre acțiuni și comenzi este de tip 1:1, astfel încât fiecare comandă este reprezentarea textuală a unei singure acțiuni.

Reprezentarea abstractă

Reprezentarea abstractă se face prin pseudocod (Figura 6).

Figura 6 Reprezentarea abstractă

Reprezentarea abstractă se obține prin interpretarea secvenței de comenzi. Comenzile din secvența reprezentată textual sunt convertite în elemente de pseudocod, în pseudoinstrucțiuni. Relația dintre comenzi și pseudoinstrucțiuni este de tip N:M, astfel încât fiecare comanda sau grup de comenzi se asociază unei pseudoinstrucțiuni sau unui set de pseudoinstrucțiuni. Rezultatul fazei de interpretare a secvențelor de comenzi este pseudocodul.

Determinarea asocierii dintre comenzi și pseudoinstrucțiuni se face pe baza unui șablon. Șablonul este predefinit în momentul crearii scenariului, a formelor metaforice și a interacțiunilor dintre ele. Un grup de comenzi se poate regăsi în mai multe șabloane, identificarea șablonului potrivit fiind bazată pe context (etichetă, scenariu din care face parte).

Fiecare grup de comenzi, grup ce poate fi compus chiar și dintr-o singură comandă, este etichetat conform funcției modelate. De exemplu, declarații de variabile, operații de intrare-ieșire, operații matematice etc. Eticheta este criteriul principal folosit în asocierea dintre comenzi și pseudointrucțiuni.

Pseudocodul este o reprezentare abstractă a algoritmului modelat inițial prin forme metaforice.

Reprezentarea codificată

Reprezentarea codificată se face prin cod sursă executabil (Figura 7).

Figura 7 Reprezentarea codificată

Reprezentarea codificată se obține prin implementarea pseudocodului, respectând sintaxa unui limbaj de programare.

Relația dintre pseudoinstrucțiuni și intstrucțiuni este de tip 1:N, astfel încât fiecare pseudoinstrucțiune se asociază unei sau mai multor instrucțiuni din limbajul de programare ales pentru implementare. Cel mai frecvent, relația este de tip 1:1, însă caracteristicile fiecărui limbaj de programare determină adăugarea de elemente suplimentare față de cele existende în pseudocod. Astfel, succesiunea instrucțiunilor din codul sursă, echivalentă cu succesiunea pseudoinstrucțiunilor din reprezentarea abstractă, este completată cu elemente specifice limbajului de programare (setări de mediu, biblioteci, declarații de constante, variabile, structuri de date etc).

Rezultatul fazei de implementare a pseudoinstrucțiunilor este codul sursă.

Asocierea dintre fiecare pseuinstrucțiune și instrucțiunile cod sursă se face pe baza unui set de reguli predefinit. Setul de reguli este predefinit în general într-o structură comună tuturor lecțiilor și scenariilor corespunzătoare. Motivul este faptul că, fiind reprezentări abstracte, aceleași pseudoinstrucțiuni pot să apară în mai multe lecții, pot fi reutilizate în scenariile modelării mai multor soluții.

Codul sursă este o reprezentare codificată a algoritmului modelat inițial prin forme metaforice.

STUDIU DE CAZ

Studiul de caz descris în [32] ilustrează o lecție în care se studiază structura alternativă. Pentru a exemplifica

definirea şi descrierea caracteristicilor structurii alternative, s-a ales ca aplicație practică modelarea problemei de selecție a maximului dintre două numere.

Scenariul imaginat este comparația dintre două cantități şi selecția celei mai grele.

Formele metaforice alese sunt două pahare cu apă, corespunzător celor două cantități de comparat, şi un cântar de tip balanță, corespunzător structurii alternative.

Tabelul 1 conține o analiză a modelării interactive vizuale a unei soluții pentru problema dată, urmărind trecerea algoritmului modelat prin cele patru faze de reprezentare, de la forme metaforice, până la cod sursă executabil.

CONCLUZII

Scopul metodei propuse este facilitarea învățării programării calculatoarelor, printr-un mediu ce stimulează gândirea algoritmică şi evită constrângerile sintaxei unui limbaj de programare. Se pune accentul pe evidențierea asocierii dintre interacțiunile utilizatorului cu mediul, acțiunile modelate vizual, obținerea pseudoinstrucțiunilor şi codului sursă executabil.

Tabelul 1 Succesiunea stărilor de reprezentare a unui algorim

Reprezentare metaforică vizuală	Reprezentare textuală	Reprezentare abstractă	Reprezentare codificată (C++)

Vizualizarea elementelor scenei**:**
- obiecte:
• pahare
• cântar
• robinet
- atribute:
• cantitate conținută
• afişarea rezultatului comparației
- interacțiuni:
• selecție pahar
• deplasare pahar
• schimbare stare robinet
- zone de interacțiune:
• talerele balanței
• robinet
	Declarația variabilelor
	(stare predefinită):
	• declarație x
	• declarație y
	• declarație max
	
	Inițializarea unor variabile
	(stare predefinită):
	• inițializare x
	• inițializare y
		real x
		real y
		real max
		x←0
		y←0
			#include <iostream.h>
			void main()
			{
			float x;
			float y;
			float max;
			x=0;
			y=0;

Umplerea unui pahar cu apă. Vizualizarea cantității de apă din pahar.	• selecția paharului „x” • deplasarea în zona de interacțiune a robinetului • schimbarea starii robinetului (on) • actualizarea valorii etichetei cantității de apă din pahar. • schimbarea starii robinetului (off).	citire x	cin>>x;
Umplerea unui pahar cu apă. Vizualizarea cantității de apă din pahar.	• selecția paharului „y” • deplasarea în zona de interacțiune a robinetului • schimbarea starii robinetului (on) • actualizarea valorii etichetei cantității de apă din pahar. • schimbarea starii robinetului (off).	citire y	cin>>y;

Deplasarea fiecarui pahar pe cântar.	• selecția paharului “x”
	• deplasarea în zona de interacțiune a cântarului.
	• modificarea valorii etichetei talerului “x”.
	• selecția paharului “y”.
	• deplasarea în zona de interacțiune a cântarului.
	• modificarea valorii etichetei talerului “y”.
		dacă
		x>y
		atunci
		max ← x
		altfel
		max ← y
		sfârşit dacă
			if (x>y)
			max=x;
			else
			max=y;

Vizualizarea paharului mai greu.	• verificare dacă ambele zone de interacțiune ale cântarului sunt suprapuse cu obiecte. • afişare maxim.	afişare max	cout<<max; }

REFERINŢE

1. Programa şcolara la Informatică ACM 2001, http://www.acm.org/education/curricularecommendations [consultat 2013]
2. Carlisle, M.C., RAPTOR: a visual programming environment for teaching object-oriented programming, http://www.usafa.edu/df/dfe/dfer/centers/accr/docs/carlisle2009a.pdf [consultat 2013].
3. Cardellini, L. An Interview with Richard M. Felder. Journal of Science Education 3(2), p.62-65, 2002.
4. Fowler, L., Allen, M., Armarego, J., and Mackenzie, J. Learning styles and CASE tools in Software Engineering. In A. Herrmann and M.M. Kulski (eds), Flexible Futures in Tertiary Teaching, 2000, http://ccea.curtin.edu.au/tlf/tlf2000/fowler.html [consultat 2013].
5. Thomas, L., Ratcliffe, M., Woodbury, J., Jarman, E., Learning Styles and Performance in the Introductory Programming Sequence, SIGCSE, p.33-42, 2002.
6. Scanlan, D. A., Structured Flowcharts Outperform Pseudocode: An Experimental Comparison, IEEE Software, 6, 5, p.28-36, 1989.
7. Carlisle, M. C., Wilson, T. A., Humphries, J. W., Hadfield, S. M., RAPTOR: a visual programming environment for teaching algorithmic problem solving, SIGCSE, p.176-180, 2005.
8. Cilliers, C., Calitz, A., Greyling, J., The effect of integrating an Iconic programming notation into CS1, ITiCSE, p.108-112, 2005.
9. Calloni, B. A., Bagert, D. J., Haiduk, H. P., Iconic programming proves effective for teaching the first year programming sequence, SIGCSE, San Jose, California, United States, 1997.
10. Seng, J.L.K., Nalaka, E.M., Edirisinghe, S., Teaching computer science using Second Life as a learning environment, www.ascilite.org.au/conferences/singapore07/procs/lim.pdf [consultat 2013].
11. De Freitas, S., Oliver, M., How can exploratory learning with games and simulations within the curriculum be most effectively evaluated? Computers and Education Special Issue on Gaming, 46, p.249-264, 2006.
12. Ebner, M., Holzinger, A., Successful implementation of user-centred game- based learning in higher education: an example from civil engineering. Computers and Education, 2005.
13. Betz J.A., Computer games: increase learning in an interactive multidisciplinary environment. Journal of Educational Technology Systems, 24(2), p.195–205, 1995.
14. Mitchell, A., Savill-Smith, C., The use of computer and video games for learning. A review of the literature. London. Learning and Skills Development Agency, 1995.
15. Second Life Education, https://secondlife.com/whatis/?lang=en-US [consultat 2013]
16. Burnett, M., McIntyre, D., Visual programming. Computer 28(3), p.14-16, 1995.
17. Johnston, W.M., Hanna, J.R.P., Millar, R.J., Advances in dataflow programming languages, ACM Computing Surveys 36(1), p.1–34, 2004.
18. Watts, T., The SFC editor: a graphical tool for algorithm development. J. Comput. Small Coll. 20, 2, p.73-85, 2004.
19. Visual Logic Online, http://www.visuallogic.org [consultat 2013].
20. Ziegler, U., Crews, T., An Integrated Program Development Tool for Teaching and Learning How to Program, SIGCSE, p.276-280, 1999.
21. Alice Online, http://www.alice.org [consultat 2013].
22. Baldwin, R., Alice Programming Tutorial http://www.dickbaldwin.com/alice/Alice0150.htm [consultat 2013].
23. Chen, S., Morris, S., Iconic programming for flowcharts, java, turing, etc., ITiCSE, p.104-107, 2005.
24. Kölling, M., Rosenberg, J., Guidelines for teaching object orientation with Java, ITiCSE, p.33-36, 2001.
25. Proulx, V., Raab, R., Rasala, R., Objects from the beginning – with GUIs, ITiCSE, p.65-69, 2002.
26. Bergin, J., Stehlik, M., Roberts, J., Pattis, R., Karel J., Robot a gentle introduction to the art of object oriented programming in Java, http://csis.pace.edu/~bergin/KarelJava2ed/Karel++Java Edition.html [consultat 2013]
27. Pattis, R., Roberts, J., Stehlik, M., Karel the robot: a gentle introduction to the art of programming, 2nd Edition. John Wiley & Sons, 1994.
28. Bruce, K., Danyluk, A., Murtagh, T., A library to support a graphics-based object-first approach to CS 1, SIGCSE, p.6- 10, 2001.
29. Roberts, E., Picard, A., Designing a Java graphics library for CS1. In Proceedings of the 3rd annual conference on Innovation and Technology in Computer Science Education, p.213-218, 1998.
30. Safta, D., Gorgan, D., MDL - Modul de dezvoltare a lecţiilor asistate de calculator, Revista Română de Interacţiune Om-Calculator, Vol. 3, ISSN 1843-4460, 2010, pp. 75-80
31. Safta, D., Gorgan, D., Aplicații MDL în didactica informaticii, Revista Romana de Interactiune Om-Calculator, Număr special: Conferința Națională de Interacţiune Om-Calculator Ro-CHI 2011, ISSN 1843-4460, 2011, pp. 75-78.
32. Safta, D., Gorgan, D., LDS: Computer-Based Lesson Development System for Teaching Computer Science, Lecture Notes in Computer Science, Volume 6389, HCI in Work and Learning, Life and Leisure, ISBN:3-642-16606-7 978-3-642-16606-8, Springer-Verlag Berlin, Heidelberg, 2010, pp. 228-243