An Approach to Teaching Introductory Programming Using Games

Tzvetomir I. Vassilev, Borislav I. Mutev

Abstract: *This paper addresses the difficulties of teaching introductory programming to computer science students. It reviews several existing approaches to make learning programming more attractive: using games or programmatically controlled hardware devices. A new game library, developed by the authors, is described, which controls virtual models moving in a virtual environment. The system uses appealing 3D graphics to attract attention. Several sample programs, illustrating main features of this approach, are presented. Screenshots of the sample programs and results of teaching a pilot group of students are given at the end of the paper.*

Key words: *Teaching programming, serious games.*

INTRODUCTION

In the last twenty years there has been a noticeable interest in young people to study computing specialities like Computer Science, Computer Engineering, Information Systems and Software Engineering. There are two main reasons for this: learning more about computers that penetrate all human activities nowadays and finding a satisfactory job. Even in the years of world economy recession there have been very well paid jobs for good IT specialists. However, learning programming well turns out to be a hard task. In general programming courses are considered very difficult by many students and often have the highest dropout rates [8]. It usually takes about 10 years of experience to turn a novice into an expert programmer [8]. Introductory programming used to be taught using procedural languages, but more universities acknowledge that in the recent years object-oriented programming (OOP) has become the most influential programming paradigm [5]. It is widely used in industry and almost every university teaching any computing speciality has it in its curriculum [5]. Currently introductory programming is mainly taught using C++ or Java [7] programming languages. There are also attempts of doing this with C# [1].

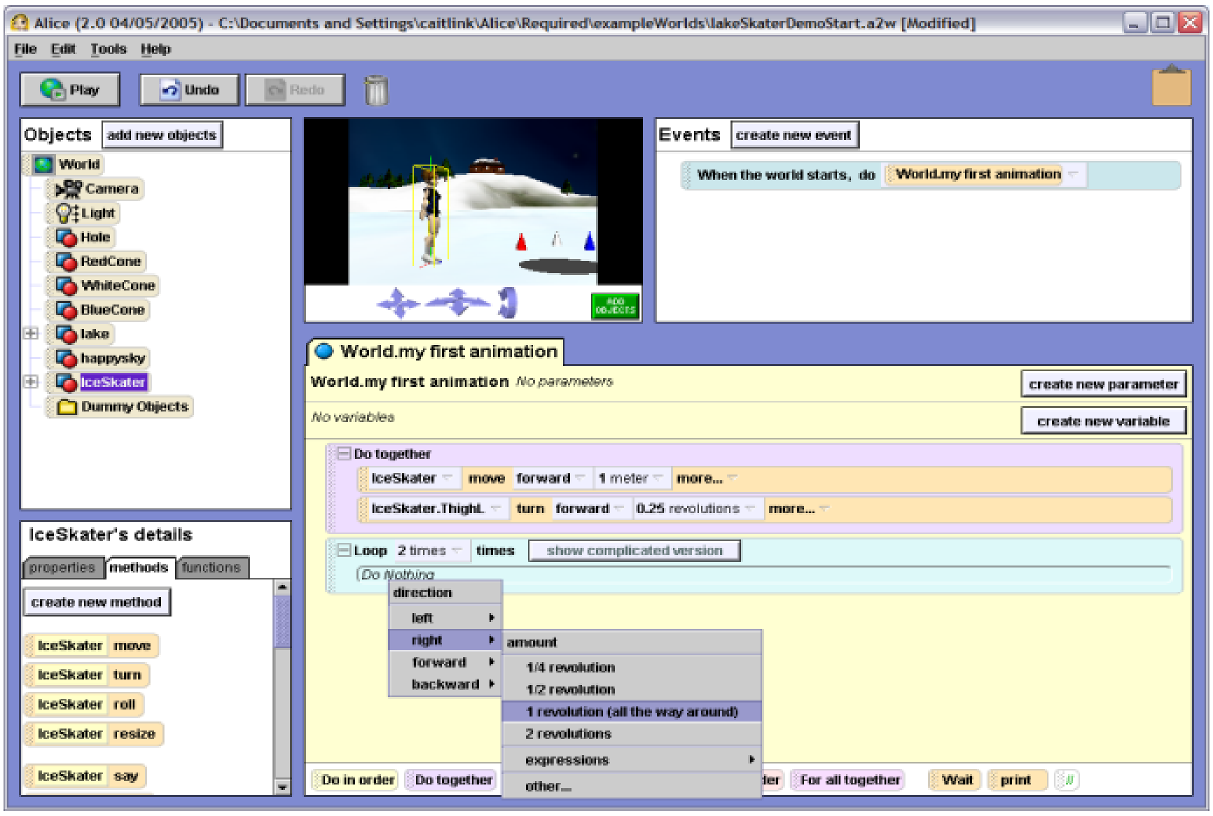
OOP is suitable to teach using real world objects, like actors, robots with their activities being methods like move, turn, jump, stop, etc. This paper reviews such approaches [2, 3, 4] and points out their strong and weak sides. It also describes a game library, which was developed using the Panda3D game engine and whose main objective is to be used for teaching introduction to programming with C++. It can also be used for teaching OOP principles for students, who have already passed the course Introduction to programming.

The rest of the paper is organised as follows. The next section describes the related work like games and programmatically controlled hardware devices to teach programming. The following two sections describe the new game library for teaching programming with C++ and give results. The last section concludes the paper and gives ideas for future work.

RELATED WORK

The Alice game

Alice is free software that can be downloaded from http://www.alice.org. It is a 3D programming environment that makes it easy to create a story-telling animation, or a video for the web. It can be used as a complementary tool for teaching programming to university or high-school students. Alice has visible data, but no syntax. The students construct programs by dragging and dropping building blocks representing world objects in a programming language. In this way Alice removes the possibility for syntax errors, which are a common source of frustration for beginning programmers. Programs can be executed, so that students see if and where they have made mistakes. The environment main screen is shown in Figure 1.



```
World.my first animation   No parameters
No variables
Do together
    IceSkater  move  forward  1 meter  more...
    IceSkater.ThighL  turn  forward  0.25 revolutions  more...
Loop  2 times  times   show complicated version
    (Do Nothing
```

Figure 1. Alice main window

Some of Alice's advantages are:

- Uses nice 3D graphics to engage students;
- Uses a drag-and-drop editor to reduce syntax errors;
- Uses storytelling, computer games to make it more interesting for students;
- Uses objects from the real world, which students can see and understand better;
- Has a Java syntax mode for easier transition to Java programming language.

In fact the main drawback of Alice is that it does not teach syntax. It is pointed out as one its advantages by its developers, and this is probably true for students non specialists in computing. In reality the syntax of a programming language is very important for computer science students and it can be studied only if it is practiced by writing programs and learning from syntax errors.

The scribbler robot

The scribbler robot is a relatively small fully programmable robot with multiple sensor systems, manufactured by the Parallax Company. It has two independent motor wheels, two obstacle sensors, two line sensors, three light sensors, three programmable LEDs and a speaker. The circuit board, connected to it, called the Fluke, is designed by the Georgia Institute of Technology. It provides a Bluetooth antenna, a low resolution colour camera and two programmable LEDs.

The Institute for Personal Robots in Education (IPRE) in the USA defined an API called Myro (for My Robot) that allows programs to control a personal robot. Myro was originally defined in the Python programming languages, but later has been ported to other languages like C++ and Java [4].

The Myro Java API is described in details in [4]. There is a collection of methods that control the robot behaviour. They include: Movement, Sensor, Camera and images, Simple speaker tones and LED control, etc.

The main advantage of Scribbler over Alice is that the students have to learn the syntax of the programming language (Java) by writing programs that control the robot. It is very interesting for the learners as they see real things happening as a result of their programs.

The main disadvantage of the Scribbler is its price. Together with the Fluke it costs about 220 USD.

Lu Yan [9] experimented with using other two game-like environments for teaching Java, called Greenfoot and BlueJ. The Greenfoot system is a framework and environment to create interactive, simulation-like applications in a two-dimensional plane. It allows implementation of and interaction with objects in the context of scenarios [9]. BlueJ has a GUI-centric design and encourages students to define classes and their relationships with an UML-like notation [9]. First the students play with objects and their properties and then they look at the source code and modify it. This approach is strongly object-oriented, and it creates objects and animations in a two-dimensional world.

Corral et al [1] decided to use Sifteo Cubes for teaching programming in C#. A Sifteo Cube is a hardware device with a 32-bit ARM CPU, 8 MB of flash memory, TFT screen and a set of sensors for motion, orientation, proximity of other cubes, etc. Using the Sifteo SDK the students write simple programs to control the cubes and detect events. This approach is suitable for teaching only object-oriented programming. Its main drawback is that it is relatively expensive: a set of three cubes costs about 200 USD.

GAME LIBRARY FOR TEACHING PROGRAMMING

We have developed a game-like library, which was designed for teaching computing students (computer science, computer engineering, information systems etc.) both introductory and object-oriented programming in C++. It is implemented using the Panda3D free game engine [6]. It possesses the advantages of Alice and Scribbler, but does not require that the students buy expensive hardware. The system exploits appealing 3D graphics to attract attention. The scenes are rendered using a directional light source set in infinity representing an artificial sun. All 3D objects are lit and throw shadows on the virtual ground to increase the realism. Shadows are implemented using a shadow map of size 1024x1024 pixels. Collision detection between moving objects and between moving and static objects is also implemented and is exposed to the programmer as described below. The student has to include the main header file of the library and link the programs to the library file to get an executable. He/she writes the main function of the C++ application and defines ordinary variables or objects of the available classes. The programs, which the students have to create, control virtual actors moving in a selected virtual environment. When the program is executed, the students can see the animation of their scene on the screen and have the possibility to navigate using the mouse: zoom, pan, rotate, etc.

The API has several main classes, described below.

CWindow. One object of this class has to be created to open a graphics window with a specified name.

CCamera. One object of this class has to be created. The constructor automatically places a camera in the scene, so students don't have to bother about it.

CEnvironment

One object of this class has to be created. A specific environment can be loaded by passing an argument to the constructor. For example:

```
CEnvironment env1(window,"env_park.egg");
```

The environments are in fact Panda3D egg files and several are provided with the samples that come with the library. When the file is loaded, it is automatically positioned, so that the origin of the coordinate system (0,0,0) is in the centre of the environment.

CActor

This is the main class, which implements virtual actors moving in the 3D scene. To place an actor in the environment one should create an object of this class

CActor actor1(window, model_name, x, y, size);

where window is an opened window (CWindow class object), model name is a Panda3D egg file describing a character, x and y are the coordinates, where the character will be positioned in the environment, and size is a scaling coefficient of the character. The default size is 1, so this argument can be omitted.

A set of models are supplied: panda, robot, Ralph (boy), Eve (girl), sphere, box, chess figures, etc.

The CActor class has several member functions:

- Movement: moveForward, moveBackward, moveLeft, moveRight, rotate, move;
- Collision detection: Collision() – returns true if there is a collision with an object in the scene and false otherwise;
- Text: setLabel – sets a text on top of the actor.

If teaching OOP, the student will be asked to implement new classes derived from CActor with additional functionality. Additional free software is provided with the library for saving the animation as a movie file, which can be viewed or publish on the web.

Many examples have been elaborated to teach different programming constructs like if, loops, arrays, pointers, etc. Three of them, used in the Introduction to programming course, are explained below.

Example 1 is a sample problem to teach if statement: enter three float number, create actors with sizes proportional to the floats, find the biggest number and make the actor corresponding to this number to move forward several steps.

Example 1. If statement

```
float step=0.2;
CWindow window( argc, argv, "IF Window" );
CEnvironment env( window, "abstractroom.egg");
CCamera camera( window );
cout<<"Enter 3 float numbers in [0.8;4.0]:"; cin>>a>>b>>c;
CActor act_a(window,"ralph.egg",-5,0,a);act_a.setLabel("A");
CActor act_b(window,"ralph.egg", 0,0,b);act_b.setLabel("B");
CActor act_c(window,"ralph.egg", 5,0,c);act_c.setLabel("C");
if (a>b)
   if (a>c) act_a.move(step,30);// a is greatest
   else act_c.move(step,30);// c is greatest
else
   if (b>c) act_b.move(step,30); // b is greatest
   else act_c.move(step,30); // c is greatest
```

The next example illustrates a while loop. The actor moves forward until it hits an obstacle in the scene. Then it tries to go round the object by moving right until there is no collision any more. Then it continues forward for 20 more steps.

Example 2. While statement

```
float step=0.2;
CWindow window( argc, argv, "While Window" );
CEnvironment env( window, "world.egg");
CCamera camera( window );
CActor actor1(window,"robot.egg", 0,0,1);
//Move forward until no obstacle
while (!actor1.Collision()) actor1.move(step);
//Go right until no obstacle
while (actor1.Collision())actor1.moveRight(step);
// Move forward again 20 steps
actor1.move(step, 20);
```

The students can also define arrays and pointers. Example 3 defines an array of 6 pointers to actors and creates an object for each pointer. The actors are first placed in the origin, and then they rotate and move away, so that they are positioned in a circle with radius 15 cm and face the centre of the circle. After that they start moving towards the centre until they collide in each other, then turn around and again move away from the centre.

Example 3. Arrays and pointers

```
const int numActors = 6, numSteps=150;
float step = 0.1;
char buf[64];
CWindow window( argc, argv, "Array Window" );
CCamera camera( window );
CEnvironment env( window, "env_park.egg");
CActor *act[numActors];
for( int i = 0; i < numActors; i++ ) {
  act[i] = new CActor( window, "smiley.egg", 0, 0);
  sprintf(buf,"Actor %d",i+1); //name for each actor
  act[i]->setLabel(buf); //set the name as label
  act[i]->rotate(i*360.0/numActors); //rotate 0,60,…
  act[i]->move(step, numSteps); //move 15 cm
  act[i]->rotate(180); //face the centre
}
for( int j = 0; j < numSteps*2; j++ )
{
  for (int i=0;i<numActors;i++) {
      if (act[i]->Collision()) act[i]->rotate(180);
      act[i]->move(step);
  }
}
```

RESULTS

The library was implemented under Windows and integrated with Microsoft Visual Studio 2008, which is used by our students. It can be easily implemented in Linux or Mac OS, as Panda3D SDK is available for all the three operating systems.

Figure 2 shows the result of running the program corresponding to example 1. The environment is a room called "abstractroom.egg" and the actor is a boy, called Ralph (ralph.egg). Each actor is assigned a label, corresponding to one of the three variables. In this case the greatest variable was B.

Figure 3 shows a picture of executing the program corresponding to example 3. The screenshot is at the moment, when all actors are situated in a circle facing each other before starting their move to the centre. Each actor is assigned a label corresponding to its number. Video clips, showing the whole animations, will be presented at the conference sessions.

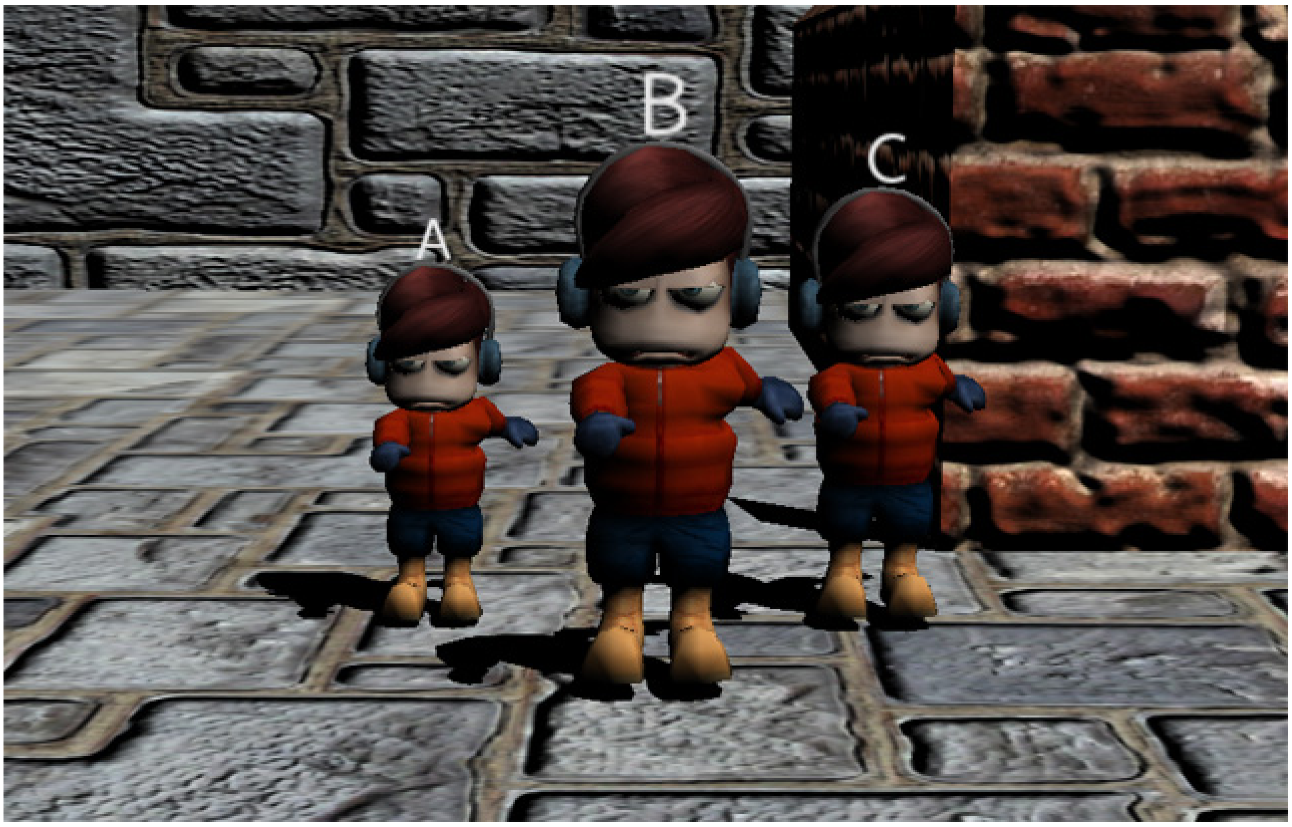


Figure 2. A screenshot of example 1 of the previous section

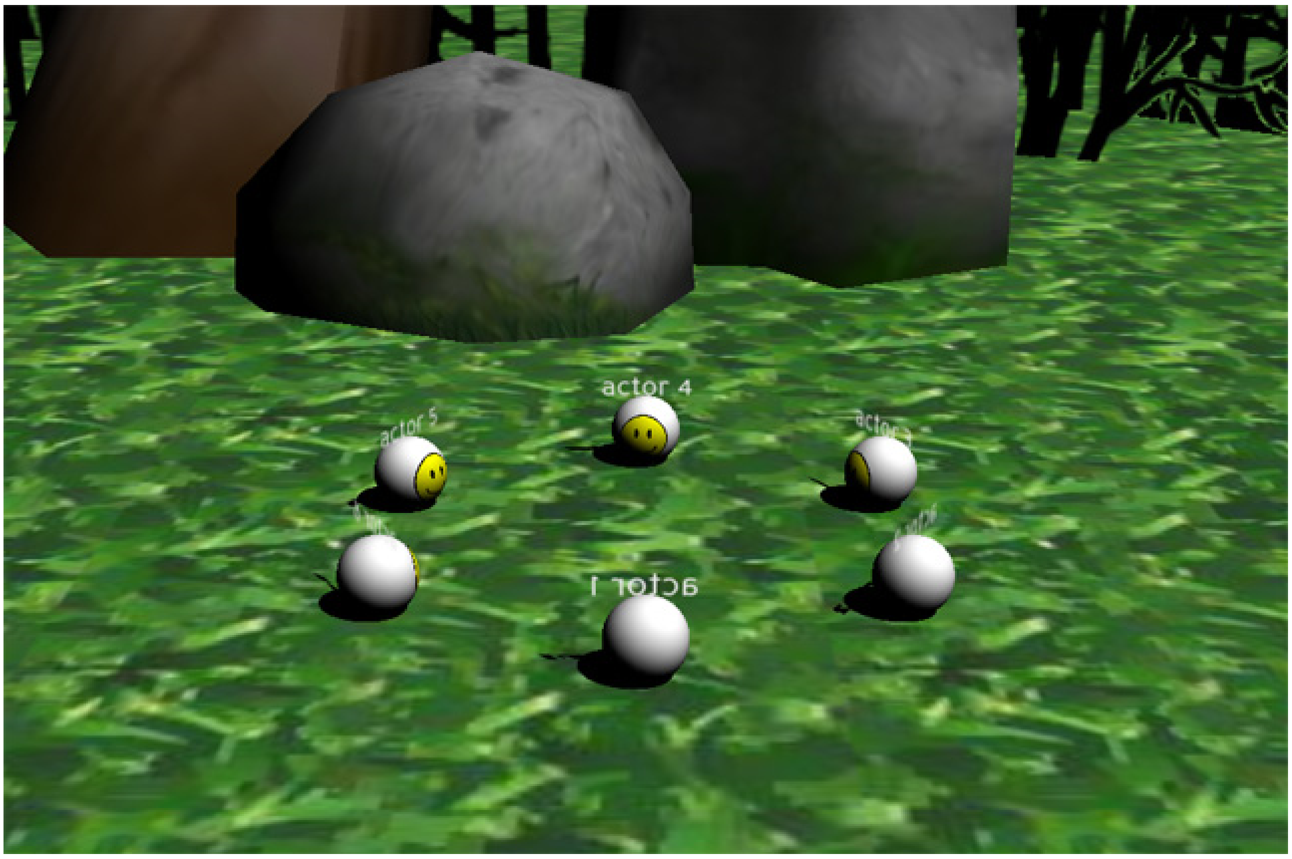


Figure 3. A picture of example 3 of the previous section

The proposed approach was experimented with a pilot group of twelve students studying the course Programming 2. In the previous semester they did the course Programming 1, which included algorithms demonstrated with Pascal. They had normal lectures as all other students, but at the workshops they were given problems and assignments to develop programs using the described game library. A sample project file was provided. At the end of the course the students were asked to fill in a short questionnaire. Eleven of them pointed out that they liked the game approach better than traditional one in the first semester and recommended that it should be used in the future. Only one noted it was more or less the same. At the final exam 8 students improved their results compared to Programming 1 while 4 repeated their grades.

CONCLUSIONS AND FUTURE WORK

The paper proposed an approach to teaching introductory programming for computing students using a game library. Unlike other approaches like Alice, the student has to learn the syntax of the C++ programming language. The library was developed using the Panda3D free game engine. The programs, which the students have to write, control virtual actors in a virtual environment. Several sample programs were presented in the previous sections. When executing their program the students will see an animation of the 3D actors moving in the 3D environment as instructed by their programs. The animation can be saved in a video file. A set of problems was developed for student assignments. Initial experiments with pilot students showed that they seem more motivated than the main group taught in the conventional way.

The following work will be done in the future:

- Add more interesting features to the library;
- Split the students in two groups. At the workshops teach one group using the conventional approach and the other one using the proposed library. Both groups will attend the same lectures;
- Compare and analyse the results at the end of the course.

REFERENCES

[1] Corral, J.M.R, Balcells, A.C., Estévez, A.M., Moreno, G.J., Ramos, M.J.F. A game-based approach to the teaching of object-oriented programming languages, Computer and Education, vol. 73, 2014, p. 83-92.

[2] Dann, W.P., Cooper, S., Pausch, R., Learning to Program with Alice – 3rd Edition, Prentice Hall, 2012

[3] Dann, W.P., Cooper, S., Pausch, R., Learning to Program with Alice, http://www.aliceprogramming.net/, 2012

[4] Harms, D. Personal Robots in CS1: Implementing the Myro API in Java. Proceedings of CompSysTech'11, ACM press, 552-557

[5] Kölling, M., The Problem of Teaching Object-Oriented Programming, Part 1: Languages, Journal of Object-Oriented Programming, 11 (8): 8-15, 1999

[6] Panda3D game engine. http://www.panda3d.org/, visited 2014.

[7] Pendergast, M.O., Teaching Introductory Programming to IS Students: Java Problems and Pitfalls, Journal of Information Technology Education, (5): p. 491-515, 2006

[8] Robins, A. Rountree, J., Rountree, N. Learning and Teaching Programming: A Review and Discussion, Computer Science Education, 13(2): 137-172, 2003

[9] Yan, L. Teaching Object-Oriented Programming with Games. Sixth International Conference on Information Technology: New Generations, Las Vegas, 2009, 970-974.

ABOUT THE AUTHORS

Dr. Tzvetomir Ivanov Vassilev, Department of Informatics, University of Ruse, Phone: +359 82 888475, E-mail: TVassilev@uni-ruse.bg.

Mr. Borislav Ivanov Mutev, Undergraduate student at the University of Ruse, e-mail: borislav_mutev@abv.bg

The paper has been reviewed.