LEARNING COMPUTER PROGRAMMING WITH GAME DESIGN

Parth Dalal[1], Nikunj Dalal[2] and Subhash Kak[1]
[1]*Department of Computer Science, Oklahoma State University, Stillwater, OK 74078, U.S.A.*
[2]*Department of Management Information Systems, Oklahoma State University, Stillwater, OK 74078, U.S.A.*

Keywords: Rapid game creation, Game design, Introductory programming, Learning, Game creation software.

Abstract: We describe a teaching approach that introduces the computer science and information systems student to programming ideas using game design. Game-creation software that requires little or no programming knowledge is used for this purpose. Students construct a simple game using the software, and later they convert their game events to pseudo-code written using standard programming constructs. This rapid game creation teaching module can be implemented rather quickly and can be used as a precursor to the teaching of a formal programming language. In this paper, we overview the pedagogical model, discuss a rapid game creation tool, present a teaching plan, and outline the potential benefits of this approach.

1 INTRODUCTION

Programming is at the heart of computer science. Yet, traditional introductory computer programming courses quickly disenchant the beginning student especially when "pedagogically unsound" languages lacking in beauty and simplicity (Gries, 2006, p.81) form the context of their first exposure to programming. There are many difficulties in introducing students to formal programming and this includes the poor preparation in mathematics and abstract reasoning that students receive in high school. Furthermore, students are increasingly using computers for a variety of interesting applications at which they feel they are already reasonably proficient using graphical interfaces. In contrast, programming is often seen as boring, which is why unlike other scientific disciplines where the gender gap has closed, the gap in computer science has worsened. Clearly what is required is a new approach and for this we propose the use of game-creation software that encourages the student to take an active role in the learning process.

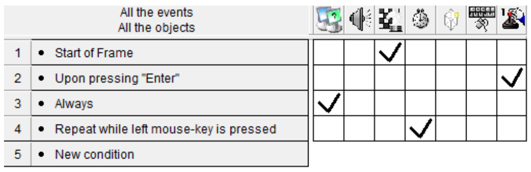
2 PEDAGOGICAL MODEL

In recent years, Digital Game Based Learning (DGBL) has received considerable attention from researchers as it has been found that playing videogames can enhance learning in both adults and children (Bowman, 1982). For students of computer science and information systems, in addition to playing a well-designed educational game, the act of constructing and designing a game itself can be a great learning experience (Rai et. al., 2006). Moreover, game creation is fun, and it appeals to all ages and to both males and females (Overmars, 2004). The pedagogical approach we propose in this paper is based on the premise that game creation develops intrinsic motivation, which influences learning in a positive way by including elements of enjoyment, fantasy, curiosity, novelty, and challenge (Malone, 1981). In addition, *rapid* game creation enables a creator to create a quick prototype and to see the effect of changes almost immediately.

3 RAPID GAME CREATION TOOLS

In recent years, several game creation tools have become commercially available. Tools such as the popular Game Maker and Multimedia Fusion typically provide an easy-to-use, powerful and visual object-oriented environment for rapidly prototyping and developing games (Overmars, 2006).

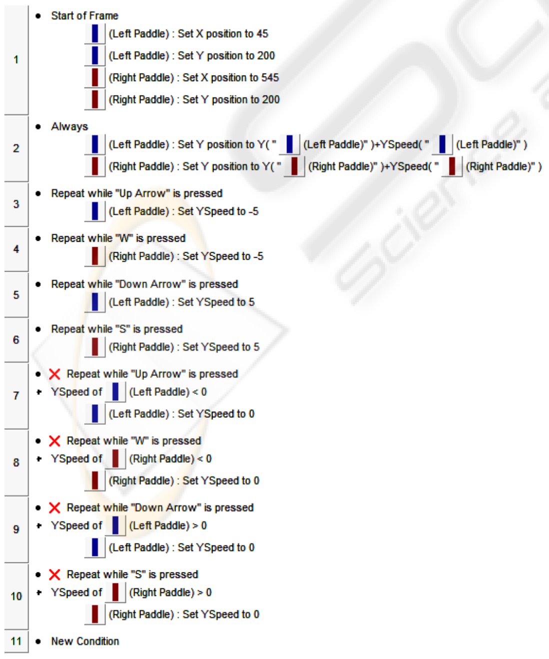
A student new to game development will perceive game creation from the perspective of the game playing process. The goal of a rapid game-creation tool, targeted at these non-programmers, is to set up developer-side interfaces that correspond to player-side interfaces, to allow the student to easily grasp the basics of the software. Generally, these interfaces consist of a scene editor, corresponding to the visual aspect of playing a game, and a condition editor corresponding to the events that occur in a game. Figure 1 shows the condition editor interface called "Event editor" in Multimedia Fusion 2 (http://www.clickteam.com/eng/mmf2.php) from Clickteam.



	All the events All the objects							
1	• Start of Frame			✓				
2	• Upon pressing "Enter"							✓
3	• Always	✓						
4	• Repeat while left mouse-key is pressed				✓			
5	• New condition							

Figure 1: The condition editor interface.

In the Event Editor in Multimedia Fusion 2, if the mouse is hovered over any one of the checkmarks in Figure 1, a list of actions is displayed, which can then be edited. The Event Editor can be viewed in another format, known as the Event List Editor, shown in Figure 2, where all actions are always visible, regardless of mouse position. It will be useful to display this format here to better understand the similarities between formal programming and game design using Multimedia Fusion 2.



1 • Start of Frame
  (Left Paddle) : Set X position to 45
  (Left Paddle) : Set Y position to 200
  (Right Paddle) : Set X position to 545
  (Right Paddle) : Set Y position to 200

2 • Always
  (Left Paddle) : Set Y position to Y(" (Left Paddle)")+YSpeed(" (Left Paddle)")
  (Right Paddle) : Set Y position to Y(" (Right Paddle)")+YSpeed(" (Right Paddle)")

3 • Repeat while "Up Arrow" is pressed
  (Left Paddle) : Set YSpeed to -5

4 • Repeat while "W" is pressed
  (Right Paddle) : Set YSpeed to -5

5 • Repeat while "Down Arrow" is pressed
  (Left Paddle) : Set YSpeed to 5

6 • Repeat while "S" is pressed
  (Right Paddle) : Set YSpeed to 5

7 • ✗ Repeat while "Up Arrow" is pressed
  + YSpeed of (Left Paddle) < 0
  (Left Paddle) : Set YSpeed to 0

8 • ✗ Repeat while "W" is pressed
  + YSpeed of (Right Paddle) < 0
  (Right Paddle) : Set YSpeed to 0

9 • ✗ Repeat while "Down Arrow" is pressed
  + YSpeed of (Left Paddle) > 0
  (Left Paddle) : Set YSpeed to 0

10 • ✗ Repeat while "S" is pressed
  + YSpeed of (Right Paddle) > 0
  (Right Paddle) : Set YSpeed to 0

11 • New Condition

Figure 2: The Event List Editor.

The events shown in Figure 2 were created in fewer than 10 minutes. Running the game with these events will allow the player controlling the 'Left Paddle' object to move the paddle up and down with the W and S keys respectively. Likewise, the player controlling the 'Right Paddle' can move his or her paddle with the arrow keys. These events form the basic structure of a Pong game. In standard programming, loops are often used to repeatedly execute a set of instructions. Typically, because games involve testing for particular conditions constantly, game programmers use a 'game loop' that contains programming statements checking for the presence of various events.

The Event List Editor, then, can be thought of as statements executed in sequence from top to bottom, and then repeated, just like a programming loop. Notice that each line (1-10) in the figure contains a condition, and the action(s) to take for that condition (actions are indented). We now rewrite some of the Event List Editor statement in pseudo-code, within a loop. This is similar to what students would do at the end of a game design module with only basic programming knowledge. The comments are marked with a '//' in front of the comment.

```
while ()  // the beginning of the game loop
{
  // incrementing a loop index
  loopindex = loopindex + 1;

  // we initialize paddle positions
  // only at the start of the program.
  // This 'if' statement corresponds
  // to Line 1 in the Event List Editor
  if (loopindex == 1) // loop just started
  {
    leftPaddle.x = 45;
    leftPaddle.y = 200;
    rightPaddle.x = 545;
    rightPaddle.y = 200;
  }

  // Moving the paddles based on key presses
  // This 'if' statement corresponds to
  // Line 3 in the Event List Editor
  if (key_down(UP_ARROW))
  {
     rightPaddle.ySpeed = -5;
  }

  // No longer moving at the rate of -5 if
  // the Up Arrow is not held down while the
  // the rate is -5.  This corresponds to
  // Line 7 in the Event List Editor.
  // Notice how the concept of ! (not) and
  // && (and) become easier to learn
  // through the red X (negation) operator
  // and the + (added condition) in
  // Multimedia Fusion 2
   if(!key_down(UP_ARROW)&& leftPaddle.ySpeed<0)
  {
     leftPaddle.ySpeed = 0;
  }

}  // end of game loop
```

Through the Event List Editor, it is much easier for students to write this pseudo-code and understand the programming constructs of sequence, loops, decisions, logic operators, classes, and objects.

4 TEACHING PLAN

4.1 The Approach

The focus of teaching programming concepts via game design should be on fun, challenge, and exploration. Learning will inevitably take place as a result. The main steps are:

1) Introduce students to a game-creation tool. The purpose at this stage is not to teach the minutiae of the game creation process or the software; the software is simply a means for the student to experience fun and challenge and to develop a liking and appreciation of programming in that process.

2) Have students come up with a simple game idea for a game they will construct using this software.

3) Each subsequent class period will be a time where students work on their games in a guided but relatively free environment amenable to exploration and creativity. In this way, students explore problem-solving concepts central to the learning of programming in an engaging gaming environment.

4) After the games are completed, the students will convert their event editor code to pseudo-code, written with IF-THEN_ELSE, DO LOOPS, and other common programming constructs.

4.2 Modular Teaching Plan

The teaching module can be implemented as a one to three credit course in game design, which could be a pre-requisite to a formal programming course, or on a smaller scale during the beginning weeks of an introductory programming course. A sample modular teaching plan is outlined below. The description is based on the use of Multimedia Fusion as the game-creation tool but any other similar tool would work. All teaching sessions are conducted in a laboratory environment.

Session 1: Show the capabilities of the game creation software and create a rapid game. Explain that students can make such games with none or little programming knowledge. Discuss the frame editor and the types of objects. Discuss the use of various movements (such as “Bouncing Ball” movement and “Eight-Directional” movement) and simple conditions for collisions and counters. Lead the students through the creation of a simple pong game. Divide students into pairs and ask students to come up with individual simple game ideas to be developed into a game in two weeks. Create a web-based asynchronous discussion forum to continue discussions among and with students.

Session 2: Discuss more features of the game creation software. Ask students to begin work on their game project. Be present to answer questions. Encourage an atmosphere of conviviality, learning, sharing, and creativity.

Session 3: Give a short lecture on common programming constructs. Then let the students continue work on their games. Be prepared to answer many questions from students who have worked on their project over the weekend.

Session 4: Ask students to convert their event editor code to programming pseudo-code, with common programming statements. Demonstrate games created by the students. Discuss examples of programming pseudo code. Conclude with lessons learned.

5 BENEFITS

We see numerous benefits to learning programming using game design. The student is exposed organically to programming concepts and constructs and develops intrinsic motivation to learn programming given the fun, challenge, and novelty of the game creation process as well as having a task to perform that offers contextualization, personalization, and choice (Cordova and Lepper, 1996) Collaborating on game design allows exchange of ideas and knowledge. Moreover, having created a game, students develop some confidence early in the programming process. Later, when concepts such as object-orientation, abstract data types, classes, attributes, methods, inheritance, and polymorphism are taught formally, students will find it easier to understand them if references to the game are made to provide instances of those concepts. As users develop a love for game-creation through this easy-to-learn structure, a few may attempt to master the advanced aspects of the tool, and in doing so, will be able to create more complex and original games with a deeper understanding of game programming. When the limitations of the tool itself seem to be reached, their interest in game programming will inspire them to learn more powerful languages.

In addition, the game creation process may involve exposure to several artistic and scientific topics that can further complement the developer's learning. In designing a complete game, there must be graphics, music, scripts and storylines, physics and much more. (Stieler, 2009). As an example, in many of the Super Mario games for the Nintendo Entertainment System, the player-controlled character, Mario, must save the Princess; this forms the storyline. There is music for every level and zone in the game. Mario's parabolic path as he jumps through the air is a result of a physics formula. These components are not just found in industry quality games but are also important in simpler two-dimensional games that students create using the type of game-development software we have discussed in this paper. The development of most complete games will include, along with the learning of programming, the learning of many other beneficial skills from a variety of disciplines and domains.

Our preliminary research shows that this method is more accessible than introducing programming through robots, which has been adopted at some schools and colleges (Calinon and Billard, 2007). Game construction and game playing provides more flexibility since it uses a variety of objects and scenarios in an interactive environment.

REFERENCES

Bowman, R.F. 1982. *A Pac-Man theory of motivation. Tactical implications for classroom instruction. Educational Technology* 22(9), 14-17.

Calinon, S and Billard, A. 2007. *Active teaching in robot programming by demonstration.* IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), 2007.

Cordova, D. I., & Lepper, M. R. 1996. *Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice.* Journal of Educational Psychology, 88, 715-730.

Gries, D 2006. *What Have We Not Learned about Teaching Programming?* 30th Annual IEEE/NASA Software Engineering Workshop SEW-30 (SEW'06), 2006, pp.175-178.

Malone, T. W. 1981. *Toward a theory of intrinsically motivating instruction.* Cognitive Science, (4), 333-369.

Overmars, M. 2004. *Teaching Computer Science through Game Design, Computer*, v.37 n.4, p.81-83, April 2004

Rai S, Wong, K. and Cole P. 2006. *Game construction as a learning tool*, Proceedings of the 2006 international conference on Game research and development, p.231-236, December 04-06, 2006, Perth, Australia.

Stieler, C 2009. *Games in Learning.* http://education. qld.gov.au/smartclassrooms/strategy/dp/games.html