Using Greenfoot and Games to Teach Rising 9th and 10th Grade Novice Programmers

Mohammed Al-Bow[2], Debra Austin[4], Jeffrey Edgington[2] Rafael Fajardo[1,3], Joshua Fishburn[3], Carlos Lara[2], Scott Leutenegger[2], Susan Meyer[1]

[1]Art and Art History, [2]Computer Science, [3]Digital Media Studies, [4]Education

University of Denver

ABSTRACT

In a two-week residential game camp we used the Greenfoot IDE to teach java programming to rising 9th and 10th graders. Students created their own computer games which required learning how to write java programs, create a game design, and create art assets. In this paper we focus on the computer science pedagogy used and describe the initial design of an augmented game development framework for the Greenfoot environment. This framework includes classes for the following useful game elements: Animation, Projectiles, Side Scrolling Worlds, Text Boxes, Clocks and Timers. We describe these classes, discuss the effectiveness of each, and describe potential improvements to their implementation and design. We also report the results of a survey conducted during each of the camps.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer Science Education. D.m [**Software**]: Miscellaneous – *games.*

General Terms

Design, Human Factors, Languages.

Keywords

Computer Science Education, Introductory Programming, Novice Programmers, Game Programming, Game Development.

1.INTRODUCTION

During July 2007 we delivered two-week residential summer game camps for rising 9th and 10th graders supported by National Science Foundation grant ESI-0624767. One camp was for young men the other for young women. The goal of each camp was to use student interest in games to increase interest in Science, Technology, Engineering, and Math (STEM) disciplines. Our camps piloted a new interdisciplinary project-based learning approach drawing together art, design, and computer programming.

Each day included three 2.5 hour learning sessions: one in art, one in game design, and one in programming. The last 4 days were spent workshopping to realize the student games. These last few days also included some additional programming concepts driven by specific student need. A more modest camp using Flash/Actionscript was piloted in 2006 with results reported in a paper by Fajardo and Leutenegger [5].

During the art sessions, students were introduced to visual asset development using traditional and digital studio practices. The curriculum focused on the figure and the scene. First, students made three-dimensional wire figures. Using these figures, two-dimensional space was addressed and gesture drawing introduced, resulting in picture planes loosely depicting the illusion of space through line weight, proportional shifts, placement, and horizon line. Gesture drawings of the human figure followed, as a live model enacted short, multi-step action poses. A move to the digital realm resulted in jointed characters ready to be programmed in an animated sequence.

Students then focused on the scene, using construction paper and scissors to create narrative compositions taken from "Picture This" [2]. Relying on shape, placement, proportion and color to depict narrative, the collages offered a model easily transferred to the digital environment. Instruction on the basics of one-point, two-point and three-point perspective followed. Studio work concluded with an introduction to value, light and shadow.

In the game design sessions, we employed the rapid creation and iteration of paper-based games to teach fundamental skills. Students were given a working definition of games taken from Salen and Zimmerman [9] and then challenged to create a game in the first thirty minutes of the first design session. They were then introduced to the concept of critical and reflective play as a way to assess the quality of the play experience. Students were encouraged to develop helpful critical vocabularies by play testing and commenting on each others' games. Students then were given opportunities to refine their games and have them replayed. Students were given a simple and clear set of quality standards so that "good" games could be identified. We also modeled and described the concepts behind Humane Games to encourage the broad exploration of kinds of games.

In addition to the student camps, we also offered a two week Teacher Game Institute, a professional development program for high school teachers who will implement game development courses in their schools. Every day, teachers had 1.5 hours of instruction in each of art, design, programming, and pedagogical theory for delivering this interdisciplinary curriculum. The teachers read David Williamson Shaffer [10]. The pedagogy sessions focused on the changing context of teaching and learning; what it means to be educated in the 21st century; technology and

information literacy standards; project-based learning and performance-based assessment; and fostering creativity in digital-age students.

2.USING GREENFOOT AND INKSCAPE

Before choosing our tools we evaluated commercial, open source, and freeware tools. We decided the benefits of freeware tools outweighed any weaknesses. We chose tools for their functionality and facilitation of teaching introductory programming and game design, but also because the developers of both tools are committed to keeping the tools free. This allows schools and other institutions to begin implementing our program at very little to no cost. It also allows students to continue development of their games at home or at school, as well as design new games.

For our programming IDE we chose to use Greenfoot [1]. Greenfoot is a free, cross-platform programming and simulation environment built on BlueJ environment [3]. The Greenfoot environment allows users to easily create and explore classes, instances, and members using a graphical interface. The latest version of Greenfoot (version 1.4.1 as of this writing) is compatible with any system that runs Java 5 or later.

A Greenfoot scenario (or program) divides user created classes into three categories: World, Actor, and Utility. Users customize the scenario by creating a subclass of the World class. Users then create subclasses of Actor for each type of game entity. Utility classes can be used to accomplish functionality such as specific mathematical calculations needed by several classes.

When the Greenfoot scenario is executed, Greenfoot starts a main loop. Each iteration of the main loop invokes the act() method of every object instantiated from subclasses of the Actor class.

For visual asset creation we chose the Inkscape drawing tool [7]. Inkscape is an open source, cross-platform vector graphics editor. Inkscape was also chosen for its ease of use and simplicity.

3. PROGRAMMING CONCEPTS TAUGHT

The ultimate goal of the camp was to get students interested in pursuing studies in STEM disciplines. One way to do this is to give students a positive and fun experience where they felt empowered. Mastering elementary programming concepts is one way for students to feel empowered. During the camp students were taught many of the standard elementary programming concepts: control structures (if/else, for loops), classes, objects, members, methods, and arrays. Each of these concepts is illustrated in our Little Red example described below.

Greenfoot's elegant design allows users to create new classes, objects, and invoke methods simply by clicking with the mouse. This greatly assists teaching the difference between classes and objects. The concepts of class, object instance, and methods were taught using the Actor class.

A Greenfoot World is an invisible grid of cells. Each cell may contain one or more Actor objects. This grid corresponds exactly to a coordinate system already familiar to students from math classes but with the y-axis pointing down instead of up. We taught the grid/coordinate system two ways: via a kinesthetic exercise and using many simple Greenfoot scenarios. The kinesthetic exercise consisted of creating a "real-world" grid on the floor of the classroom. A grid of masking tape was placed on the floor with cells large enough to accommodate two students. Students were then "created" and added to the grid at specified grid locations. Each student was then "moved" about the grid by invoking "methods" on the student. This exercise was quite useful for teaching concepts of grid location, movement, velocity and the Greenfoot World class boundaries. This exercise also helped solidify the concepts of class, object instance, and method invocation. In conjunction with the kinesthetic exercise we also used our Little Red scenarios to demonstrate how Actor objects are located on the grid and how they are moved. Students were quickly able to create their own Actor classes with student created movement methods. The combination of both approaches seemed to result in a deeper understanding of the coordinate system. Many of the students exhibited that "a-ha!" moment as they moved their classmates around the grid with "code".

4. LITTLE RED RIDING HOOD EXAMPLES

To provide the students a complete learning example we created a game based on Little Red Riding Hood and informed by Molly Bang's excellent design principle book "Picture This" [2]. A goal was to use simple art so as to not intimidate the students and show them that fun games can be created with modest visual assets. A screen shot of the main play screen is in Figure 1. Little Red is the small triangle in the top left corner. We started the camp by having the students play the game. The game contains a "splash" screen with play instructions, static objects (apples) to be collected, dynamic objects (wolves) to be avoided, a goal (collect a certain number of apples) and appropriate end screens (success for collecting enough apples; failure for being eaten by a wolf). We separated the various game aspects such as keyboard events, collision detection, world position, and character animation into individual Greenfoot scenarios and corresponding programming lessons for the students.

FIGURE 1: Little Red Screen

Conference'04, Month 1–2, 2004, City, State, Country.

By first playing the game students quickly learn the Greenfoot interface. Further, the example game sets the stage for a series of increasingly more complete Little Red Riding Hood scenarios to explain step-by-step how the game is created. The scenario progression matches the order we presented the programming lecture topics. The series of Little Red scenarios are as follows:

1. Little Red moving: to the right and left, back and forth, bouncing off world boundaries (both dimensions), and keyboard controlled. Keyboard control is made easy with the Greenfoot keyboard handling methods. The boundary reflection and direction changing necessitates if/else statements.
2. Add a background image and apples, which are static objects. Add collision detection code using the Greenfoot Actor's getOneIntersectingObject() method so Little Red can collect apples.
3. Add a populate() method to the World constructor to place apples in the world at fixed locations using the Greenfoot addObject() method.
4. Spell words with apples: Use for-loops to increment x or y-locations to place apples to spell out words like "HI" or "IF". Although not part of the game the students like this example which helped explain for-loops and further reinforced their understanding of the grid cell ordering.
5. Use for loops and the Greenfoot getRandomNumber() method to place apples at random inside our populate() method (which is called by the World constructor).
6. Add wolves with a simple patrolling movement behavior. The wolves move 10 spaces forward and then change directions by 90, 180, or 270 degrees with equal probability. This code requires keeping track of the direction a wolf is heading using a data member so that on the next movement the (x,y) location can be set correctly. This scenario also makes use of the setRotation() method to rotate the wolf image to face the direction it is headed. We also added an intersection test between Little Red and a wolf so that if they collide, the game stops.
7. Modify the wolf to be a "heat seeking wolf". The wolves have the same patrolling motion as above, but, if Little Red is within a certain distance, specified via a method parameter, the wolf then moves towards Little Red. This requires teaching students how to calculate the distance between two points thus making math they have been exposed to relevant.
8. Create data members to hold the number of apples collected so far and the number of apples needed to win. Update the number of collected apples and move to a win screen when a sufficient number is collected, or, move to a lose screen when a wolf gets Little Red.
9. Give Little Red a magic wand to make wolves disappear if they are close by. We store the wolf objects in an array and when Little Red uses the wand, by pressing the "W" key, we loop through the array to calculate the distance to each wolf. If the distance is below a threshold value the wolf is removed from the game.

This series of Little Red examples illustrates many of the concepts that most of the students would need to complete their own game. Of course many of the students came up with much more grandiose design ideas that necessitated more sophisticated programming. To make this possible we have added several classes for students to use and/or modify as described in the next section.

5. GAME DEVELOPMENT FRAMEWORK

Since many games involve similar concepts: timers, projectiles, etc. we provided the students with classes which implement these ideas. This also helped mitigate the short time span of the camp itself. We wanted the students to concentrate on implementing their game rather than trying to implement details of projectiles for example.

We also wanted to provide the students with as many examples as possible. This was important for two reasons: the students could study the examples during their own time and they could modify our examples rather than creating a new solution from scratch. The examples served two purposes: as samples of good coding practices and as foundations from which the students could build their games.

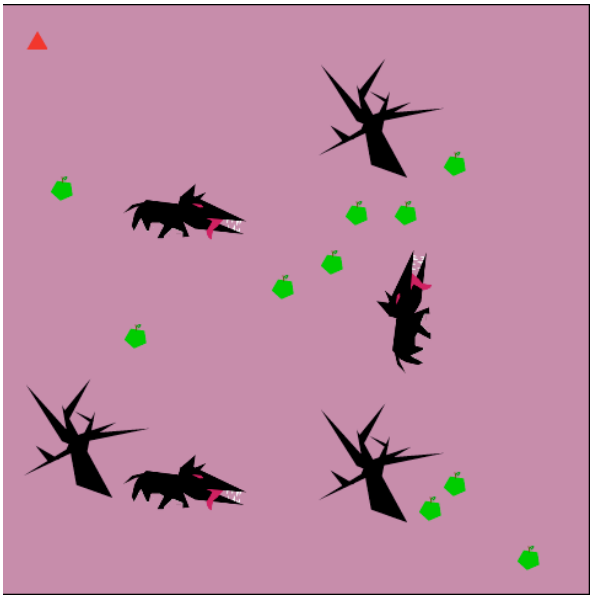
A number of game frameworks have been developed using Java. We discuss two that we considered using rather than Greenfoot and creating our own framework. The Java Instructional Game Engine (JIGE) [11] met many of our requirements: it uses Java, it targets the development of two dimensional games, it is open source, and free to use. The only two drawbacks (from our perspective) are that it is somewhat complex for ninth and tenth grade students and would have required us to teach the students a development environment or how to use the command line interface to Java. The Labyrinth framework is also implemented in Java but is aimed at creating one particular genre of two dimensional game (i.e. a simple fantasy role-playing game with a "torch bearing hero" and a "fire breathing monster") [4]. We were concerned that many of our students might not be interested in this sort of game. Labyrinth also has the same complexity issues as JIGE.

We ultimately decided to use Greenfoot and build our own game development framework. Our framework of example classes is quite simple because the Greenfoot environment provides much of the foundation required to

implement many games and simulations [6]. We provided students with six main classes: Timer, Text Box, Projectile, Bounce Movement Actor, Animation, and Side Scrolling World.

We also provided the students with color-coded class diagrams. We used different colors for the data member declarations, the act() method, any others methods, any any constructors. We purposely chose a separate color for the act() method to highlight it as the key method for Greenfoot Actor objects. We found that the color coding greatly helped students understand the parts and roles of the classes.

All example Greenfoot scenarios, class implementations, documentation, and other materials are freely available from our project website [8].



5.1 Bounce Movement Actor

This new type of actor, allows a game object to move in either the x or y dimension according to data member velocities. The game object also respects the boundaries of the world by reflecting off of them. The class included only four methods: act(), atXboundary() which returns true if at the left or right-most boundary of the world, atYboundary() which checks for the top/bottom boundaries, and moveXY(). The class contains two data members: xVelocity and yVelocity. The moveXY() sets the new location of the object to its current location plus the x/y velocity values. The act() method simply checks if the object is at a boundary, and if so negates the corresponding velocity value, and then calls moveXY().

The Bounce Movement Actor class was provided in the first of the Little Red example scenarios. This class was used by many students to make (non-keyboard controlled) enemies or obstacles bounce around the world.

5.2 Timer Class

The timer class is used in games that require a timed task by displaying a clock on the screen. The clock counts down at a rate based on number of acts specified by the student. Once an object of the timer class is created and added to the world, the timer can be started or paused using the methods start() and pause() respectively. The timer class was used in many of the female students' games especially games involving mazes.

5.3 Text Object Class

The text object class allows adding text to the world by specifying the location of a text object and it's contents. The changeText() method is used to set the contents of the text object. The text object class was used in almost all games during both the boys' and girls' camps.

5.4 Projectile Class

The projectile class was used to facilitate the inclusion of projectiles such as bullets, balls, etc. The constructor allows the students to set the speed and the lifetime of the projectile where lifetime determines how long before the projectile expires and is removed. The projectile class provides three usage types: 1) launching of a projectile at a specified angle; 2) launching at a target object; or 3) launch a projectile that tracks a target as the target moves. The class includes methods: shootInDirection(int rotation), shootAtTarget(Actor Target), shootTracking(Actor

Target). These methods will place the projectile object at the xy-coordinate of the shooting object and the projectile will move in the direction specified or towards the target as described above. This class was developed during the boys' camp and was used in 10 games (out of 17) during that camp. The projectile was also used in one game to compute line of sight.

5.5 Animation Class

The Greenfoot Actor class provides methods for changing the location of the actor object and changing the actor's current image but does not provide a method for animating an actor (i.e. displaying a sequence of images). Since animation is a common element of computer games, we created a new simple class which provides this capability.

The animation class stores a list of images. An Animation is constructed by passing the base name of a set of numbered image files. For example, if one creates an animation of a flying bird with 7 frames the image files could be name bird1, bird2, … bird7 and the base name is "bird". The class has four key methods: initAnimation(); startAnimation(), stopAnimation(), and animate() which gets the next frame of the animation.

Many of the boy's games (13 out of 18) used this simple animation or side scrolling animation (described below) in their game. Few of the girls used this class (only 2 out of 17). We believe the low usage by the girls was due to the fact that this class was presented very late during their camp. Everybody that used this class thought it was easy to use. The Animation class is used by the Side Scrolling World class described below.

5.6 Side Scrolling World

Many of the students wanted advanced features similar to side scrolling games (including multiple animation cycles such as walk, run, and jump triggered by different key strokes and health bars). To our knowledge Greenfoot does not support continuous scrolling of a world. Our initial design of the class was rather complicated to use because its methods combined movement, animation, and the scrolling of the world. Scrolling was implemented in a discrete way: a new "background" class holds an array of background images. Methods such as setNext() and setPrevious() allowed the change from the current background image to the specified background image.

Nearly half of the student games used the Side Scrolling World class, but getting it to work correctly necessitated significant instructor assistance which was contrary to our goal of making the students feel empowered. We are in the process of simplifying this class to improve its utility and usability.

6. SURVEY RESULTS

Seventeen students completed surveys both before and after the boys' camp. The survey asked questions regarding attitudes toward attending college, computer science, the camp experience, and whether basic programming concepts were understood by the student. The questions and responses were:

Q: How strong is your programming? (Asked first day of camp): "none" 4, "minimal" 7, "average"4, "strong" 1.

Q: Rate your understanding of Classes and Objects: "A little shaky" 2, "okay" 8, "strong" 6.

Q: Give an example of an if/else statement: "No Answer" 7, Examples of if with incorrect code: 8 (?), 2 correct answers.

Q: Rate your understanding of how to use Greenfoot: "No Answer" 1, "Weak" 1, "Okay" 11, "Strong" 3.

Q: Describe your programming comfort level now: "No Answer" 1, "Vague Idea" 3, "Reasonable Idea" 7, "Feel Confident" 6.

Q: Rate how your comfort level with technology changed: "No Answer" 1, "Remained the Same" 4, "Improved" 10, "Jumped by Leaps and Bounds" 2.

Q: Rate your understanding of the Coordinate System: "No Answer" 1, "Shaky" 1, "Okay" 10, "Strong" 5.

Q: Rate your understanding of For Loops: "No Answer" 1, "Weak" 4, "Shaky" 4, "Okay" 7, "Strong" 1.

Q: Give an example of a for loop: "No Answer" 9, "I don't know" 2, two examples ("Waive your hand and say hi 8 times", "doing something over"), three said something similar to "for (i=0; i<20i)", one said "Like having 4 different classes".

In every case the attitudes improved as we had hoped. Our main concern lies with whether the students understood basic programming concepts. Most of the students believed that they had a stronger understanding of classes, objects, and how to use Greenfoot. They also believed that they had a higher comfort level with programming in general and technology in general. Yet few of the students produced a correct if statement or for-loop on the post-surveys. Students correctly used these structures many times in their games. The post-survey was rushed due to a time limitation and we conjecture this was the cause for low quality answers on these two questions. From these survey

results, and the enthusiasm and pride shown by both the young women and men when demonstrating their games at the final family celebration, we conclude that this method has been quite successful in teaching elementary programming concepts to novice programmers at a young age. Most important, students were more comfortable and interested in technology after the camp.

7. FUTURE WORK

First, we plan to conduct deeper statistical survey of the students in future camps and in the high school programs. Second, we are building additional classes to add to the framework. One will provide the ability to create worlds which are larger than one screen and allow Actors to move seamlessly in this larger world (i.e. provide "true" scrolling or transition from one portion of the world to another). This new class will replace the Side Scrolling World class described earlier. We would also like to allow students to create worlds with non-square grids (hexagons and triangles) and general graph connectivity. We are also creating a "talking bubble" class which will allow narrative animations for Actors. Finally we are creating a simple network interface which would allow students to create multi-player games and simulations.

8. ACKNOWLEDGMENTS

Our work is supported by National Science Foundation grant ESI-0624767.

9. REFERENCES

[1] The Greenfoot Programming Environment, www.greenfoot.org

[2] Bang, Molly, PICTURE THIS, How Pictures Work, SeaStar Books, 2000, ISBM 1-58717-030-2

[3] The BlueJ Programming Environment, www.bluej.org

[4] Distasio, Joseph, Way, Thomas, P., Inclusive Computer Science Education Using a Ready-made Computer Game Framework, ITiCSE'07, June 23-27, 2007, Dundee, Scotland, United Kingdom, ACM 2007.

[5] Fajardo, R., Leutenegger, Scott T., "Programming, Pixels and Play: A University Summer Game Camp To Attract Under-represented Populations to Game Development and Computer Science", Proc. of Future Play, 2006.

[6] Henriksen, Poul, Kolling, Michael, greenfoot: Combining Object Visualisation with Interaction, OOPSLA'04, Oct 24-28, 2004, Vancouver, British Columbia, Canada, ACM.

[7] The Inkscape drawing tool, www.Inkscape.org.

[8] Pixels, Play, Programming, and Pedagogy, www.p4games.org

[9] Salen, Katie, Zimmerman, Eric, Rules of Play, MIT Press, 2004, isbn 0-262-24045-9

[10] Shaffer, David Williamson. How Computer Games Help Children Learn. Palgrave Macmillan. 2006

[11] Wallace, Scott A., Nierman, Andrew, Addressing the Need for a Java Based Game Curriculum, Journal of Computing Sciences in Colleges, Volume 22, Issue 2 (December 2006), pp. 20-26.