SENIOR PROJECT: GAME DEVELOPMENT USING GREENFOOT

Karen Villaverde, Bretton Murphy
Computer Science Department
New Mexico State University
1290 Frenger Mall SH 123
Las Cruces, NM 88003
575-646-1609
kvillave@cs.nmsu.edu, hiei@nmsu.edu

ABSTRACT
In this paper we describe our very positive experience in teaching a senior project course—in one semester—with game development as the subject matter. In this course a 2D game engine called Greenfoot was used as the development platform. We describe why Greenfoot was chosen as our 2D game engine, what material was covered, and how the course was conducted. We also describe how the grading was performed, the quality of the students’ game projects, features of Greenfoot that students liked for game development, features that students wished Greenfoot had, and future work.

INTRODUCTION
We always wanted to teach a senior project course where 2D game development was the focus. However, in our university, senior project courses are given in just one semester. Therefore, our main concern was whether the students would have enough time to learn to use a 2D game engine, and then design, prototype, develop, and test a complete senior project game in one semester. Giving the students an already familiar programming language was another problem. Because of these concerns we decided upon Greenfoot [6] as our game development platform.

In the spring of 2010, we taught a senior project course with Game Development as the subject matter and used Greenfoot as our development platform. The course was fourteen weeks long and was taught in the Computer Science department of New Mexico State University. This was the first time that a senior project on game development has been taught in our department. We were very satisfied with the quality of the senior project games produced by the students. The students enjoyed the course very much, learned a lot, and felt a strong sense of accomplishment in having developed a complete 2D game—even though none of them had developed any game before.

In this paper, we describe our very positive experience in teaching this course with Greenfoot as our game engine. Greenfoot has been used to teach how to make very simple games to high school and university students [9], but to our knowledge nobody has used Greenfoot in a senior project course before. We hope that this paper encourages other faculty to give Greenfoot an opportunity as a powerful 2D game engine for senior project courses in the future.

GREENFOOT AS A GAME ENGINE FOR GAME DEVELOPMENT
Greenfoot is a free, 2D educational game engine developed in 2006 with the goal of teaching programming at high school level or above. It can be effectively used at school, college, and university levels, and even in advanced university courses. Greenfoot is a project of the University of Kent at Canterbury (UK) and Deakin University, Melbourne (Australia) with support from Sun Microsystems, Oracle, and Google. Greenfoot is a multiplatform game engine which runs on Windows, Mac OS, Linux, and Unix. Students write their games in Java. Greenfoot supports the full Java language and games can be run as applications, on a web browser, or on the Greenfoot environment.

Some of the best benefits of using Greenfoot is that Greenfoot has excellent resources for learning the engine (e.g., video tutorials by one of the creators of Greenfoot—a professor), Greenfoot is
regularly updated, and the Greenfoot community is extremely active. The Greenfoot community posts, comments, ranks games in the Greenfoot gallery, participates in discussions, and asks/replies to questions in the Greenfoot forum. The Greenfoot gallery is where users can post their games and there are literally hundreds of them. They can be run directly on the website and the games’ source code is available for download.

There are other game engines that use a general purpose language for 2D game development and have been used in educational settings, e.g., Microsoft XNA [11], DarkBasic [3], and Pygame [13]. However, most of these platforms have a very steep learning curve. The learning curve becomes even steeper if students have not programmed extensively in the platform’s language. This makes them very challenging to learn for students in one semester while at the same time completing a senior project. On the other hand, Greenfoot has an extremely shallow and straightforward learning curve with no major learning bumps. Students program their games in Java, a language that almost all computer science students know very well and have used it extensively by their senior year. This enables students to learn Greenfoot in a few weeks and gives them enough time to design, prototype, develop, and test a complete 2D game in the same semester.

REASONS FOR CHOOSING GREENFOOT

We chose Greenfoot as our 2D game engine for the following five reasons. First, our positive previous experience of using Greenfoot in another course. We taught a summer game programming course using Greenfoot as our 2D game engine the previous year. The only pre-requisite for this course was a data structures course in Java. Even though our summer semester is only 10 weeks, our students were able to learn the Greenfoot platform, learn how to use image and sound editors (Gimp [4] and Audacity [1]), and design, develop, and test two small but interesting 2D games. Therefore, we knew that it was possible for students to learn the Greenfoot platform in a short amount of time and develop some decent games in the same semester. Second, we wanted to try an experiment with our senior project students who love to play games but who have never actually developed a game before. Given our success with the games developed by our summer game programming students, we wanted to see the quality of games that could be produced by our senior project students working on teams. Third, we wanted a 2D game engine that could be learned in a short amount of time—a few weeks—so the students could have enough time to design, prototype, develop, and test one complete 2D game during our fourteen weeks course. Fourth, we wanted a 2D game engine with a general purpose language in which the students had programmed extensively (e.g., Java). Fifth, we wanted a free 2D game development platform where lots of different types of games could be implemented.

MATERIAL COVERED AND HOW THE COURSE WAS CONDUCTED

The main material that we covered in class came from the excellent videos tutorials and the Greenfoot manual available on the Greenfoot website. The video tutorials explain the use of the Greenfoot engine in a step by step and very educational way as they are written by the professor who created Greenfoot. This material was covered in a laptop equipped classroom with three projectors and lectures that were 100% hands on and extremely interactive.

Other material that we covered in class were game techniques such as color masking for collision detection, background scrolling, and handling levels to efficiently manage the different stages of a game; how to increase Greenfoot’s memory heap in order to run large games; how to use Audacity and Gimp—including transparencies and scaling; and some game optimization techniques. We learned some of this material by studying some of the most popular games and demos in the Greenfoot website [2, 12]. We have now actually developed video tutorials that teach all these techniques and concepts step by step and are available at [8]. We also covered some material from the Greenfoot book [10]. Some of the material covered includes the use Greenfoot’s helper classes SmoothMover and Vector for
creating movement, how to add gravitational forces, apply gravity, how to handle proton waves for fire power, and differences between sound and image file formats and sizes.

The class met three days per week for a total of 185 minutes per week. In the first four weeks, we covered the use of the Greenfoot engine, the game techniques, the use of Audacity and Gimp, and the material from the Greenfoot book. Also, in the first four weeks while the students were learning the Greenfoot engine, they were given as homework to explore the Greenfoot gallery. They also brainstormed ideas for the type of game, theme, and game play mechanics that they would like to develop for their senior project game. The students were free to select their own game type, game mechanics, theme, characters, sound effects, background music, etc. They were also free to make their own teams. We just made sure that each team had a good programmer. We had three teams of three students each.

The remaining ten weeks of the semester were spent on the design, prototype, development, and testing of the senior project games. The students did about half of their senior project work in our laptop equipped classroom while we supervised them, encouraged them, assisted them with their questions, and offered them suggestions. The decision for letting students work on their senior project in class worked extremely well to prevent student procrastination and greatly increased team communication. Also, the students appreciated the fact that we let them work in class because for them it was very difficult to meet outside class—all of our students had part-time and full-time jobs and some of them families of their own. However, our students did work (individually) on their project outside of class. During class, students divided among themselves some of the work so they could advance individually outside of class as well. The students were not required to complete software development documents other than the Java Doc documentation of all their code. This decision allowed our students to have extra time to spend for development, testing, and debugging of their games. Due to the time constraints of completing a game in one semester, the students followed a rapid application development approach. Our students developed their games in approximately 1850 minutes in class and 1800 minutes outside of class.

**GRADING**

Two criteria were used to grade the students: class attendance and participation (30%) and their senior game project (70%). The main criteria used in the evaluation of the senior game projects were quality, effort, creativity, Java Doc documentation of all their source code, as well as meeting the project deadlines: prototype version, alpha version, beta version, and final version. These project deadlines were evenly distributed in the last ten weeks of the semester. We observed that students worked extra hard when a deadline was approaching because they wanted to show that their game was the best. All the students were males and enjoyed proving that their game was the best during the games’ presentations after each deadline.

**GAME PROJECTS’ QUALITY**

All three senior projects were side-scrolling platformer games. Figures 1 through 3 show the three games developed by our students: Toby, Star Wars, and Retro All Stars.

In Toby [17] (Figure 1) the player plays the role of a robot called Toby. As Toby goes through the world, he must fight evil bees, turtles, birds, and snails that constantly attack him. At the same time Toby must avoid precipices and other obstacles. All of the art work was designed and created just for this game and consists of all unique work done by one of the team members who is an artist. One of the most interesting aspects of this game is the parallax technique used in the scrolling background which is composed of several layered images. This technique allows images in the background to scroll slower the farther away they are from Toby, creating a kind of 3D effect. All the game elements—mechanics, programming, story, and esthetics—work together to form a single unified theme. The final product is a
unique and very challenging game consisting of 36 different Java classes, 5582 lines of code, 90 sprites, and 16 sound effects. Toby is a great example of what Greenfoot can accomplish. The executable jar file is 17.8 MB.

In Star Wars [16] (Figure 2) the player plays the role of Luke Skywalker who has to fight his way through the game destroying evil droids and stormtroopers. The player can use three different types of weapons: a light saber for close quarters combat, and a blaster pistol and machine gun for long range combat. There is also a force attack that can be used from a distance. The game consists of three levels. To complete a level, all enemies have to be defeated. The excellent character animation of this game enhances very well its game play and story. Star Wars consists of 28 Java classes, 4293 lines of code, 278 sprites, and 12 sound effects. The executable jar file without background music is 8.6 MB and with background music is 26 MB.

Retro All Stars [14] (Figure 3) is a parody game where the player plays the role of Castlevania’s Simon. Game play is almost exact to that of the classic game Castlevania. But as Simon progresses through the game he finds himself in the worlds of Super Mario Brothers and Zelda fighting bosses from Mega Man, Teenage Mutant Ninja Turtles, and South Park’s Chris Hansen (of NBC’s Dateline). There are also references to Metal Gear Solid and lots of one-liners from Duke Nukem (including his rendition of the famous line from John Carpenter’s “They Live”). Simon has to fight his way using only a whip as a weapon. Each level is progressed by making it to the end of the level in which case the next level begins immediately. The game consists of three levels. Simon dies when all his health is depleted. Retro All Stars consists of 30 Java classes and 3487 lines of code, 86 sprites, and 14 sound effects. The executable Jar file is 5.3 MB.

According to an in-class vote, the best game was Toby, followed by Star Wars, and then Retro All Stars. The three games where voted for in the Greenfoot gallery by the Greenfoot community and obtained the same rankings as the in-class vote. All three games were among the best 10 games in the Greenfoot gallery at one time and received lots of comments by the Greenfoot community. Toby was chosen as a Greenfoot showcase scenario by the Greenfoot creators.

FEATURES OF GREENFOOT THAT STUDENTS LIKED

Table 1 below lists all the features of Greenfoot that our students liked.

<table>
<thead>
<tr>
<th>Greenfoot features that students liked</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Greenfoot has a very clean and simple user interface that is easy, fast, and intuitive to learn.</td>
</tr>
<tr>
<td>(2) Greenfoot’s 2D world grid can be adjusted to any number of cells with each cell adjusted to any number of pixels down to a single pixel size.</td>
</tr>
<tr>
<td>(3) Greenfoot’s user interface includes a class diagram browser which provides a very nice way to view all the game’s classes and their inheritance relations.</td>
</tr>
<tr>
<td>(4) Greenfoot supports Java Doc and it is very easy to switch a file’s format—in the Greenfoot’s code editor—between Java code and Java Doc by just clicking a button.</td>
</tr>
</tbody>
</table>
Greenfoot has an extremely easy to use and very complete Integrated Development Environment (IDE) where class diagram browsing, coding, compilation, debugging, execution control, and accessing Java Doc documentation of user’s code—as well as the Greenfoot API and Java library documentation—are very pleasant to use.

Greenfoot’s API is small—it consists of only six classes, with a relatively modest number of methods, easily learnable, and very well documented in Java Doc format.

Greenfoot’s API provides very useful collision detection methods, e.g., there is a collision method that returns a list of all objects that intersect a given object.

Greenfoot’s API provides very useful methods for handing keyboard and mouse input.

There are several very useful support classes in the Greenfoot website for animations, explosions, smooth movements, handling vectors, and GIF animations.

Greenfoot programs can be paused, single stepped, speed up, and slow down at run-time using a scroll bar.

Objects can be manually added to a game by right clicking their class in the class diagram.

An object’s methods can be executed by right clicking an object and selecting one of its methods for execution.

An object’s instance variables and their values can be examined at run time.

Greenfoot provides automatic thread management. Greenfoot executes the main method—act method in Greenfoot’s terminology—of each object in a round robin fashion. And even though it is not possible to specify the order of execution of individual objects, it is possible to specify the order of execution for classes of objects.

Greenfoot’s performance is good in that games with several hundreds of objects, all checking collisions between each other, run smoothly.

Greenfoot handles most computer graphics on its own, leaving the programmer to concentrate on the game logic.

### Table 1. Features of Greenfoot that students liked

<table>
<thead>
<tr>
<th>FEATURES STUDENTS WISHED GREENFOOT HAD</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Even though the students enjoyed working with Greenfoot (version 1.5.6) very much, they did express their desire for Greenfoot to have the following features. Fortunately, most of these problems have been fixed in Greenfoot version 2.0.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Greenfoot features wished by students</td>
</tr>
<tr>
<td>(1) The Greenfoot code editor did not have code completion, scope highlighting, and navigation view features. These features have been implemented in version 2.0.</td>
<td>(1) The Greenfoot code editor did not have code completion, scope highlighting, and navigation view features. These features have been implemented in version 2.0.</td>
</tr>
<tr>
<td>(2) Greenfoot does not have version control. The students used Google wave to keep track of their game versions.</td>
<td>(2) Greenfoot does not have version control. The students used Google wave to keep track of their game versions.</td>
</tr>
<tr>
<td>(3) The Greenfoot tracing and debugging feature that our students used did not work properly so the feature was not used at all and had to improvise to trace and debug their games. This feature now works very well in Greenfoot version 2.0.</td>
<td>(3) The Greenfoot tracing and debugging feature that our students used did not work properly so the feature was not used at all and had to improvise to trace and debug their games. This feature now works very well in Greenfoot version 2.0.</td>
</tr>
<tr>
<td>(4) Greenfoot did not have mp3 or midi support and wav support was not very good. We had to code wav and midi players for our students to use. Fortunately, Greenfoot 2.0 now has good mp3, wav, and midi support.</td>
<td>(4) Greenfoot did not have mp3 or midi support and wav support was not very good. We had to code wav and midi players for our students to use. Fortunately, Greenfoot 2.0 now has good mp3, wav, and midi support.</td>
</tr>
<tr>
<td>(5) The Greenfoot game engine had some annoying bugs suspected to be memory leaks. Most of these bugs have been removed in version 2.0.</td>
<td>(5) The Greenfoot game engine had some annoying bugs suspected to be memory leaks. Most of these bugs have been removed in version 2.0.</td>
</tr>
<tr>
<td>(6) Greenfoot does not do game optimizations by itself. All optimizations must be planned, studied, and programmed.</td>
<td>(6) Greenfoot does not do game optimizations by itself. All optimizations must be planned, studied, and programmed.</td>
</tr>
</tbody>
</table>
The Greenfoot gallery only accepts games of 20MB or less. Fortunately, there is a way around this restriction when posting big games: students can post the URL to their own websites where their game is located and can be played as well. Unfortunately, most Greenfoot users do not click on these URLs—they prefer to play games that can be played directly on the Greenfoot gallery.

Table 2. Greenfoot features wished by students

FUTURE WORK

In the future we plan to include the teaching of game design concepts in this course. We plan to use the book The Art of Game Design [15] which has been proven very useful in a game design and development course that we just finished teaching. We believe that the knowledge from this book will encourage, help, and guide students to design and develop innovative games that have better flow channels (skills vs. challenges progressions), better interest curves (interest vs. time progressions), polished game play mechanics, well designed interfaces, and good game balancing.

We plan to require students to complete a design and optimizations document where they study, plan, and analyze the memory and time complexities of their data structures and algorithms. The students will also study and plan as many optimizations as possible. We felt that this document was very much needed to avoid game performances from lagging. We also plan to require students to learn and use a code versioning program like Github [5]. This is a very useful skill that they should know before they graduate.

In order to achieve all the above, we will require students to learn Greenfoot at home instead of us teaching it in class. It is indeed possible for students to do this due to the excellent video tutorials in the Greenfoot website and the additional video tutorials that we have created since we taught this class [8].

REFERENCES