INTEGRATING GAMES INTO THE COMPUTER SCIENCE CURRICULUM: FROM GENERAL EDUCATION TO THE GRADUATE LEVEL

Robert E. Marmelstein, Eun-Joo Lee
Department of Computer Science,
East Stroudsburg University
East Stroudsburg, Pennsylvania, USA

ABSTRACT - The use of game-based projects in the computer science curriculum has the potential to make the teaching of numerous computing courses more effective and interesting. In this paper, approaches are presented for using computer games to teach computer science at different educational stages: from general education through the graduate level. In each case, we examine how the game project supports the goals of the course. Potential pitfalls and lessons learned from these teaching experiences are also addressed.

KEY WORDS - Computer Games, Artificial Intelligence, Computer Science Education and Curriculum

I. Why Teach Computer Games?

For many of us, teaching computer game programming is not just a pedagogical exercise—it’s also personal because we love recreating the games we play. Likewise, many students become motivated to become computer scientist as a result of the computer games they play as children and teenagers. Beyond the challenge of playing the game, the underlying technology that makes these games work fascinates many students. In short, playing these games provides the motivation to learn how to program them. While most students may never become professional game programmers, it’s important for computer science educators to appreciate the power of the computer game as a mechanism for teaching many facets of the discipline. The following bullets illustrate the rationale for this assertion:

- The design, coding, and testing of most games can be very challenging. The experience forces the students to develop, utilize, and refine many different skills and processes.
- Incorporation of “play offs” can increase the motivation of students to innovate and excel.
- The process of writing a computer game often has a significant interdisciplinary component. It may require the student to employ skills in art, music, math, physics, and storytelling to varying degrees.
- Increasingly, games can incorporate technologies beyond graphics. In particular, they can make use of new input devices and sensors (such as Microsoft Kinect) which can recognize poses, movements, and speech. Additionally, game development may require web programming for multiplayer games, as well as, cross-platform development for mobile devices and game-specific platforms like the Xbox.

Obviously, there are many compelling reasons to integrate computer games into our curriculum. Often, this is accomplished by creating one or more classes specifically for computer game design. While this is the most direct approach, there are many other opportunities to integrate games into the curriculum at various levels. The remainder of this paper will explore how games might be incorporated at the general education, introductory, advanced undergraduate and graduate levels. Additionally, we will look at common pitfalls associated with teaching computer games and how avoid them.

II. General Education Level

Convincing young people to study computer science is difficult for several reasons. Those who grew up with computers often come to think of them as mere appliances. Conversely, many students also avoid the subject due to the perceived difficulty of programming. If those students take a General Education (GE) course in
For many years, both the fundamentals of procedural programming and Objected Oriented Programming (OOP) were covered in our first class in our introductory programming sequence. Even though the OOP topics have since been moved to a follow-on course, we still teach Java as our first high level programming language. While Java can be used to do procedural programming, it is an inherently OOP language. Given this, the challenge is finding an introductory project which develops both the student’s procedural and OOP skills.

In order to develop these skills in our students, we had them develop a Blackjack game. Use of the Blackjack game project to teach OOP is an approach that other computer science educators have advocated [2]. Note that the game is not developed all at once; rather it is completed in stages over the course of the semester. The nominal sequence of stages is as follows:

- **Stage 1** – Validate the string representing the value of a given card (e.g., “J-S” for Jack of Spades). Report an error if the string does not represent a valid card.
- **Stage 2** – Instantiate a Card object from a validated string.
- **Stage 3** – Create a Hand object to keep track of a series of dealt cards; as such, the Hand object is composed of Card objects. As each card is added, update and report the value of the hand (including if a bust has occurred). Note that the Hand may have a primary and an alternate value if it contains an ace.
- **Stage 4** – Create a mechanism to deal the cards from a virtual Shoe object. The Shoe can hold four decks of cards and must be shuffled at the start of the game and when it runs low. Additionally, two classes are created for different types of game participants: a Dealer class and a Player class. For each game, the program will determine the winner based on the value of the players’ hands.

Recently, our Department launched a new GE course entitled “Games, Robots, and Intelligence”. The purpose of this course is to provide non-majors an introduction to using computers to solve problems that are both fun and interesting. As the title suggests, the course is divided up into three parts. In the robots part of the course, students use the Lego Mindstorms kits to build and program robots for simple tasks. The intelligence portion examines the nature of intelligence, both biological and artificial. Some of the issues and impacts associated with advanced in machine intelligence are presented and discussed.

For the games part of the course, students are taught to build simple games using the Scratch programming environment. **Scratch** is an interactive development environment which enables novices to quickly prototype games without having to learn syntactically correct code writing first. Created by the MIT Media Lab, it is intended to motivate further learning through playfully experimenting and creating projects, such as interactive animations, games, etc. [1]. Scratch scripts are constructed using functional building blocks that can be fit together to build larger, more complex structures. Each of these blocks represents different types of functions including: control, sensing, motion, looks, sound, pens, variables, and operators. These scripts are created for each individual sprite in the game. Each sprite is an object (character) that interacts with other sprites on a common stage; this makes Scratch an inherently object oriented environment. One of the most positive aspects of Scratch is that it enables students to quickly get a visual, interactive program working.

While Scratch makes programming easier and more intuitive, it does not make it effortless. In this course, students are guided, step by step through a sample Scratch program (see Figure 1) that makes use of all the relevant features. This “Copter” game is built up in stages: first the state is created, and then the sprite is imported. The initial sprite script allows the copter to take off and fly. Next sprite animation and sound are added. The next steps are adding the logic to detect either a successful landing or a crash. Lastly, the animation associated with a crash situation is added. There are many excellent books on Scratch for beginning programmers; the one currently used is “Scratch Programming for Teens” by Jerry Lee Ford. The Scratch.mit.edu website has a wealth of free resources, including help pages, video tutorials, and a downloadable gallery of over 3 million Scratch programs.

![Figure 1 - Scratch Copter Game](image)
• Stage 5 – At the conclusion of a game, output the cards of each hand in sorted order.

As you can see from the above sequence, the first stage of the project largely involves procedural programming skills. However, with each stage, more and more OOP skills are introduced to the student. The last two stages of the project introduce the concept of arrays and sorting. Figure 2 shows a typical Java class diagram for the final version of Blackjack.

Unlike the Scratch game, this game need not make use of the GUI or other graphics-multimedia capabilities. Instead, input/output is text-based because the emphasis is on developing the student’s procedural and OOP skills. The fact that this project is a card game that many of them are familiar with makes it fun, challenging, and rewarding. At the end of the class, the students have a readily familiar accomplishment they can show their parents and friends.

![Java Class Diagram for Blackjack Game](image)

**Figure 2 - Java Class Diagram for Blackjack Game**

IV. Advanced Undergraduate

The next major opportunity to write computer games comes in our Computer Game Development course, which is an advanced undergraduate elective. In the latest iteration of this class, students are required to write two full-scale game projects in C# using XNA game development library. They also complete several minor programming assignments which build up their skills in preparation for the major projects. Both major projects are 2D, arcade type games. While 3D graphics techniques are covered later in the course, they are not required for these projects.

The first project is an “Asteroids” style game. This game is developed incrementally and due at midterm. The purpose of this project is to help the students master the basics of an arcade type game. The “Asteroids” game itself is challenging from a real-time programming perspective. The game software must create, move, and animate many different object types and their related interactions; some of the objects, like asteroids and missiles, have large populations which must be managed. Examples of the “Asteroid” game objects include: the player ship, asteroids (of varying size), missiles, explosions, and an autonomous alien ship. Likewise, inputs from the user must be processed without a noticeable delay and appropriate sounds must be played when certain events occur (e.g., explosions). The game also incorporates a fair amount of physics, including velocity, acceleration, and credible collision detection-and-response effects (e.g., conservation of momentum). To round out the experience, students are also required to implement a title screen and a high score screen as part of the game application.

The second project is a “Pacman” style game, due by the end of the term. This is likewise an arcade style game; however, the emphasis here is on constructing the maze environment, scrolling, navigation, and implementing the artificial intelligence (AI) for the bots chasing the player. The background maze is required to be much larger than the viewport (see Figure 3). When the player moves within the maze, the background must scroll to keep the player centered in the viewport, if possible.

The most elaborate part of designing the game, however, is the AI, including the behaviors of the bots [3]. The player can use the controls to guide his character through the maze; the bots must chase the player convincingly, without making it too difficult for the player to evade them. As a result, each bot must track the player in a unique way. For example, one bot may track the player by column; another bot may track the player based on the last intersection visited. Bots may also share information periodically in order to appear more intelligent and less predictable.

Once a given bot has an estimate of the player’s current position, the bot must then plan a path through the maze from where it currently is to the player’s potential location. Of course, over even a few game cycles, the player’s position will change, so the bot must continually re-plan its route based on the most current estimate. Like “Asteroids”, the “Pacman” game must run in real-time, so the continual planning that goes on cannot be allowed to slow down the game. As you can see, both games are challenging, self-contained projects that teach the students complementary sets of skills for real-time programming.

Additionally, we are currently supervising an independent study course where students are developing a “Battle Tetris” application that runs on Android mobile phones. This app will be distributed, such that students can compete against each other in real time when connected to the cellular phone network. If this course goes well, we will considering incorporating this material into a
cross-listed undergraduate/graduate advanced games course.

![Figure 3 - Student Version of Scrolling "Pacman" Game](image)

V. Graduate Level

At the graduate level, students build a Reversi game as the main project for the Data Structures and Algorithms course. In this game, the human is pitted against a computer (AI) opponent. Because a human is in-the-loop, the focus of the project is searching for the best move in a reasonable period of time (under 3 minutes). Due to the time constraint, the student must develop effective heuristic search strategies and limit search depth to maintain effective performance within the allowed time.

![Figure 4 - Student Implementation of Reversi Game](image)

The graphical user interface for a student-built Reversi game is shown in Figure 4. Most students implement the Reversi AI using a Minmax search algorithm coupled with alpha-beta pruning, as this is a well-documented approach [4] [5]. As a result, the key to designing the most effective computer player is the search heuristic. While some board positions are obviously more valuable than others (e.g., corners are considered the most valuable), the most successful students are those who use multiple factors to create a composite heuristic. Typically these factors include mobility (e.g., choosing a move that limits the mobility of the follow-on moves of the opponent), frontier (e.g., restricting the move choices the opponent has on the frontier), and stability (e.g., choosing moves that cannot be easily flipped).

Additionally, students are required to do a performance analysis of the time required to select a move based on the tree depth and move number (starting with the first move and ending with the last move). Figure 5 shows the chart generated from one such student analysis. Once the projects are complete, students participate in a “play off” tournament held on the last day of class. The student whose implementation wins the tournament gets extra credit—even more so if their implementation beats that of the professor. Of course, the biggest payoff for the student is the invaluable experience of building an AI algorithm that can beat a person at a strategic board game.

![Figure 5 - Performance Analysis Chart for Reversi Game](image)

In the computer graphics course, developing a game program has been utilized to enhance students’ fluency and understanding of the course material. Two major learning goals are established for the students: acquiring fundamental concepts and skills, and developing various problem-solving strategies. Obtaining the fundamentals is essential to a student’s learning process. In the class, students are expected to understand and implement all of the pieces of the graphics pipeline - model building, transformations, and rendering - in order to create an image. To develop students’ problem-solving strategies, a final project of creating a game program has been employed, in which students were requested to solve various types of problems based on their knowledge and skills in computer graphics. Based on our experience, the process of developing a game program often helps to
provide students with a deeper and more practical understanding of the subject.

Figure 6 shows a 3D Tetris game, developed by a graduate student in the previous class. The student expected it to represent a fully functional playable game. Also, some key concepts of computer graphics were demonstrated such as transparency, user interaction, orthographic views, double buffering, nested transformations and sub windows with window specific event handlers.

![Figure 6 - 3D Tetris Game](image)

VI. Pitfalls

Despite the many positives of assigning game-related projects, there are also significant pitfalls as well. Most of these stem from student misconceptions about the nature, and difficulty, of the task. A fair number of students who sign up for a “computer games” course may expect to do more game playing than game writing—this is true even for programming courses. Even those that expect to work are often surprised at the level of difficulty associated with fully implementing an arcade type game. On some level, they may expect computer games to be as easy to write as they were to play. Because of this, it is good practice to make clear early on that the course will involve a significant amount of work; this helps weed out those who may be looking for entertainment vs. education.

Another issue with game programming is shifting standards and tool obsolescence. Our first Computer Game Development course used Allegro, a cross-platform game library, for C++ game development. While Allegro was well supported, there were issues with some of the support tools, especially the tile-map editor (Mappy) needed for maze design and scrolling [6]. The Mappy editor came with a playback engine compatible with Allegro version 4. However, when Allegro version 5 was released, there was no Mappy upgrade for the new version. This left me with three undesirable choices: find a replacement for Mappy compatible with Allegro 5, continue to use the outdated version of Allegro, drop my Pacman clone project, or migrate to another game library.

Ultimately, Allegro was abandoned in favor of XNA [7], a cross-platform, .Net game library for C# developed by Microsoft. However, that switch was rather painful as all my Allegro C++ code had to be converted to XNA C#. Additionally, with several scrolling tools had to be tested before finding one that worked in an acceptable manner; and even then, the selected application (TIDE) only had a fraction of the functionality that Mappy possessed. While the transition to XNA was successfully completed, Microsoft recently announced they will no longer support XNA. So, at some point, the whole process will have to be repeated for some new Game library. The moral of the story here is: be very careful with standards and the tools that you are depending on. Whenever possible, choose libraries and tools with the best support and greatest probability of longevity.

Another pitfall is the sheer complexity of the many game design tools. There are many different tools that can be used to develop games. While some of these are easy to use, many of the most powerful and flexible tools (like 3DSMAX and Maya) have very long learning curves. Even some of the simpler graphic editors, like Gimp, can take a while to master. For this reason, if the focus of the course is game programming, it’s best to choose tools that deliver basic functionality without the learning curve.

Another challenge is trying to cover all essential topics in a single semester. There are many aspects of game programming that can be taught with 2D game applications. However, 3D games are increasingly the norm. The problem is that it’s tough to combine both 2D and 3D game techniques in a single course without overloading the student. In the computer games course, the focus is on 2D scrolling and AI, vs. going into 3D techniques. The process for building 3D games is a conceptual sea change from 2D, especially in how objects are designed and rendered on the screen. 3D game programming inherently involves matrix creation and manipulation, making it far more mathematically complex than 2D games. While many game libraries have features to facilitate 3D graphics programming, it is still a non-trivial exercise. For this reason, it is preferable to separate 3D game development into a follow-on course. In our case, the computer graphics course serves as the follow-on course for learning 3D graphics.

Because of the difficulty of writing games, it is very important that the instructor write a functioning version of each game assigned. Even if it’s a game that you’ve written before using another language or game library, it
is always a good idea to convert your work to match the currently assigned configuration. Having a self-written version of every assignment becomes invaluable in guiding the students through the game development process step by step. Remember that any game implementation, no matter how seemingly trivial, comes with at least one “gotcha” characteristic. Without being able to assist them with the hard-to-implement features, many of your students may succumb to the frustration factor.

Lastly, it is essential to keep the students’ expectations of the course realistic. It’s important that they understand that a single game development class won’t make them a game writing superstar; rather, it is only a starting point. Like professional athletes, only a select few programmers ever get hired by top-flight game companies like Electronic Arts. The key to success is getting experience on your own writing even more complex games or involving yourself in open source game development. Successful game developers build up a visible, accessible portfolio which will eventually cause someone to take notice.

VII. Conclusion

Computer games are an excellent vehicle for teaching a wide variety of concepts within the computer science curriculum. However, they can also be very challenging; as a result, it is important to select the right game project and development environment for a given course objective. Luckily, with the number of game tools and environments continuing to expand, this is becoming easier than ever. At the introductory level, the introduction of the Scratch environment for teaching novice programmers is extremely exciting; it is also a potential vehicle for getting many non-programmers interested in computer science. In this paper, we’ve related our experiences with injecting computer games into the computer science curriculum at all levels. Hopefully, this can be used as inspiration or a guide for readers to do likewise.

References