Games Programming in Computer Science Education

D. Vrajitoru[1], **P. Toprac**[2]
[1]Department of Computer and Information Sciences, Indiana University South Bend, IN, USA
[2]Department of Computer Science, University of Texas at Austin, TX, USA

Abstract – *In this paper, we investigated the role of content and processes as they relate to games and programming in post-secondary computer science courses. Our examination found themes that can help computer science instructors teach both introductory and advanced programming topics. Our analysis was focused primarily on the theoretical and practical subject matter that is taught in game programming classes and, secondarily, in how game-based programming projects and assignments can help students learn core computer science concepts and improve coding skills.*

Keywords: game programming, curriculum design

1 Introduction

Game programming has been a popular topic in computer science education for many years. The popularity of game programming with computer science students is due to two causes. First, programmers and potential computer science majors are often passionate about playing games and participating in the game culture and related media, including literature, film, conventions, live action role-playing, and so on. Thus, they are driven by a desire to create games and be recognized within the gaming community. Second, the game development industry provides significant job opportunities, which also fuels students' interest in the field. Job opportunities are present for those who are interested in working for established, relatively stable, game companies, as well as for those who would rather work as independent freelancers or for small entrepreneurial startups.

Although game programming courses are often offered as electives for computer science majors, there are a large number of universities that offer entire degrees or programs in game programming. Such programs include the Bachelor's of Science in Computer Game Development from DePaul University in Chicago, IL, the Certificate of Computer Science in Game Development from the University of Texas at Austin, and the Bachelor's of Science in Game Design & Development from the Rochester Institute of Technology, NY. Moreover, computer science departments offer game design and development courses and programs in collaboration with departments or schools of fine arts, and even English departments.

Several studies have examined the curriculum of game programming courses based on their syllabi. The typical purpose is to help instructors of these courses design their assigned classes more effectively. For instance, [1] examines what topics are appropriate for game courses, what would be the best choice of language/API considering the class level, and various ways to find useful resources. In [2], the authors survey a range of game courses and describe how digital games can support the learning process in computer science education. In [8], the authors discuss how content related to computer hardware can be learned by means of a digital game. The study presented in [11] illustrates the use of game development for teaching problem solving techniques. Our paper examines in more detail some of the aspects mentioned in [2], particularly those regarding game courses in computer science at different levels of student expertise.

In [4], the authors discuss how active learning as a tool to teach critical thinking and problem solving is attractive to computer science students. The paper also touches on the issue of retention of computer science majors and how active learning methods can improve it. Our paper also discusses the course CSCI-B100 Tools for Computing mentioned in [4].

In the papers [3] and [10], the authors examine the interdisciplinary nature of some of the upper-level game development classes, which we will also discuss in Section 4 of this paper. As part of the interdisciplinary nature of game development courses, managing teams is discussed in [7].

Studies can also be found on using games or other entertaining topics in introductory computer science classes, such as Gothic novels [5], role-playing games [6], and gamification techniques in [7] and [9]. Our paper also discusses the use of games in early programming classes.

This paper is organized as follows. Section 2 examines introductory programming classes, as well as core course following them. Section 3 discusses middle level courses with specialized content in game development, while Section 4 takes a look at upper level courses. We finish the paper with some conclusions.

2 Early Programming Classes

Let us start by examining programming classes that take place early in students' computer science track, such as the CSCI-B100 Tools for Computing, taught at Indiana University South Bend (IUSB). Typically these are the first or second programming classes taken by students.

The goal of such classes is to introduce both major and non-major students to programming and possibly attract undecided students to computer science. These classes typically employ simple programming languages, such as Visual Basic or Processing, because they are easier to learn without prior experience in coding. The advantage of these programming languages is that they come with visual tools that can teach basic programing concepts without requiring students to learn the strict syntax of coding using text. Using visual programming tools lessen cognitive load and can improve motivation and learning over coding using text. Another technique that is often employed to improve motivation and learning is using project-based learning. That is, students must complete practical mini-projects throughout the semester in a lab environment under instructor supervision.

Game development projects can also be used in introductory courses to improve learning and motivation, because students can quickly see that basic data structures and programming concepts are necessary in these projects. The game development process affords immediate visual feedback of progress during programming. In addition, games provide an element of fun and entertainment, making it easier to engage students in the task at hand and making them more invested in the outcome of their programming assignments.

For example, a game of Pong [12] can teach beginner programmers about variables – notably the two paddles and the ball. Pong presents a problem requiring the solution to an iterative process where the ball moves in the stage area with a given speed and direction. In order to solve the problem, students are introduced to the idea of conditionals by checking for collisions of the ball with the stage boundaries or walls and performing a bouncing operation when this occurs.

More advanced concepts, such as functions, can be introduced through many different games. For example, in a game of Tic-Tac-Toe, a function can be written to verify the winning condition. The concept of simple arrays and string manipulation can be taught through a game of Hangman. Here, a string is needed to hold the word to be guessed. A Boolean array is used in the background to mark the letters in the word that were guessed correctly so far.

More advanced core courses can also benefit from using games as programming projects. Games provide practical applications for data structures and object-oriented programming, and are more fun and appealing to the students. For example, game assignments can be used in courses such as CSCI-C243 Data Structures taught at IUSB, which serves as a prerequisite for and gateway to most of the upper-division core and elective courses. This class teaches major data structures, including tables, trees, and graphs, and algorithms. It is important for students in this class to not only learn and understand data structures but also grasp their pervasive importance in all areas of computer science. The latter can be achieved both by providing lists of applications for each data structure, and by assigning practical programming tasks that make use of the abstract data structures that are introduced.

For instance, two-dimensional arrays serving as tables can be explained through arcade-style games such as PacMan [13] or Sokoban [14]. In these games, the world is two dimensional with discrete coordinates and can be represented as a 2D array or matrix of integers. An even better approach would be to define an enumerated type for the stage components, containing constants for walls, spaces, food, and other game elements. Then, the matrix representing the game area would use it as the base type for the cells.

A game of Breakout [15] can teach the concepts of boundary checking in data structures, as well as navigation. As a ball bounces against the walls, conditions of being within the boundaries of the arena must be checked every time. In the classic version of the game, the player destroys bricks by bouncing a ball against them, which is represented as a 2D array. Breakout also teaches students the complex relationships between integer coordinates in the table and real coordinates in the arena, in order to display the bricks properly. This leads to the concept of type conversion from a higher precision data type to a lower precision one.

More complex data structures, such as stacks and queues, can be somewhat abstract and difficult to grasp for the students. Card games are ideal applications for making abstract concepts more concrete. For example, a shuffled deck can be stored in a stack or a queue, depending on its purpose in the game. In the classic version of Klondike Solitaire, the game starts with eight “tableau” piles of cards facing down. Only the top cards can be turned face up and played from each of the piles. Other cards can be placed on top of the piles in descending order and in alternating color. Thus, tableaus can be represented as stacks. To win the game, all the cards must be moved to four foundation piles by suit and in ascending order. Again, cards can only be added to the foundations on top, which also means that they are best represented as stacks. Finally, the deck of playable cards can be recycled, so that when the cards are exhausted, playing starts over. To accomplish this, either a pair of stacks or a queue can be used, the latter being better suited for the purpose of the class.

Binary trees are also challenging for students to learn, and, again, games can make this subject less abstract. For example, one can show, in a game of Tic-Tack-Toe, how the current state of the game can be expanded into a tree where each branch is a possible move leading to another state. This can lead to explaining how artificial intelligent (AI) agents could use this tree to play the game. Even though AI is probably too early for students to implement at the time, the discussion helps to effectively introduce the concept, which can be built upon latter in the curriculum.

Object Oriented Programming (OOP) is often taught in the second or third semester of programming courses within the computer science curriculum. Developing games can help improve the students' comprehension and retention of OOP. Game projects showcase well-structured hierarchies of classes, some of them with storage purposes, such as stacks and queues, and others with more operational purposes, such as an interface class. A game project typically contains a Game Master or Manager class that oversees all the game objects, rules, and goals. This helps students understand not only the individual purpose of each class, but also the need for an overall structure. Incidentally, the term Game Master is also used when playing role-playing games. So, students who play these types of games may intuitively understand the topic better and also feel increased engagement with the subject.

3 Medium Level Classes

In this section, we will examine courses that are taken in students' sophomore or junior years. These are typically elective courses in a general computer science undergraduate degree program, or foundation or core courses in a degree program specialized in game development. Examples of such courses include the CSCI-C490 Games Programming and Design taught at IUSB and GAM 374 Fundamentals of Game Programming I taught at DePaul University, Chicago, IL.

Students in these classes have already completed many core courses, as well as probably rigorous algorithms and data structures courses. Although instructors typically find these students to be capable programmers, students still need to practice and extend their coding skills. At the same time, these courses train and hone students' programming skills and give them a solid game development experience.

These game programming electives usually teach students fundamental concepts in game development, relevant algorithms, design topics, and game-specific program structures. Examples of typical game algorithms include fair shuffling methods for card games, efficient randomized content generation, and kinematics-based object movement. Collision detection is another important topic that requires a significant amount of time during the semester to cover. Another focus is the various game loops for different types of games.

Students better appreciate the importance of algorithms when applied in concrete applications. For instance, Prim's algorithm is at the core of procedurally generated content, which allows the instructor to create a connection between game development and graph theory. Other graph algorithms have applications to games, notably path-finding algorithms for maze-like structures.

In terms of design, these courses emphasize strong program structure and object-oriented design. They also contain many topics related to game design, such as the mechanics, dynamics, and aesthetics analysis of games. In the C490 class at IUSB, students typically must write a midterm paper examining a game of their choosing from several points of view, such as genre, theme, rules and mechanics, reasons for failure or success, as well as the social context of the game. By performing class presentations of their papers, students train their communication skills while either sharing their enthusiasm for their favorite game or warning peers of bad decisions made by some game creators.

These higher-level courses are very likely to be intensive in terms of their use of libraries or APIs. These may not be entirely new concepts for the students, but they have not worked with them extensively before. A large part of the instruction may be dedicated to getting students more comfortable with using specialized game libraries or engines. For example, some classes may be taught in C++ or Java with direct use of OpenGL. As an alternative, an API such as Flash / ActionScript may be used, or even a game engine such as Unity or Unreal. The challenge in this case is to understand the tools provided by the game engine and how best to make use of them in the development process. Although it takes extra effort to learn how to use theses libraries, the gain is in being able to focus on higher-level game aspects by letting the libraries take care of lower level tasks.

An important aspect of the mid-level classes is that they are programming-intensive. As mentioned previously, although students have a good programming background, they still need to develop and improve their coding skills by working on real-world projects. In lower level classes students see programming as a goal in itself, the task to be accomplished to earn a good grade. Mid-level classes allow students to move towards seeing programming as a tool to accomplish something specific. Students are not focused anymore on just the algorithms for their own sake, but are solving a complex problem that requires solving many other smaller ones.

Furthermore, students get to experience how much of a challenge creating a good game can be and how important debugging is for the development process. Since many games are intended to be played over and over, any coding or design flaws are more likely to be revealed over time than in other programs that students may have written previously. Thus, students learn to appreciate the need for good programming practices and for extensive testing techniques. Students can also better understand their instructor's insistence on efforts to improve program performance. Generally, correctness of the program is a major goal for any coding activity, but efficiency may seem to beginner programmers as a lower priority. When working on games, students can see first-hand how an efficient algorithm performs better for their purpose than a weaker one. Good program structure and programming practices are thus seamlessly integrated into their learning objectives.

4 Advanced Level Classes

In this section, we examine advanced courses taken by students who are seniors, but may be juniors. An example of such a course is CS 354T Game Development Capstone: 3D Games taught at the University of Texas at Austin.

The focus of these courses is game development in terms of interdisciplinary teamwork, project management, and relatively complex software system development of significant size. The teams can be between 3 and 7 students, with bigger teams providing a working experience closer to that of the game industry. Oftentimes, these courses are open to students from various majors: computer science, fine arts, music, and others. The team composition reflects the variety of specialties and interests that creates a balance of the skills necessary to produce a game that approximates those in the real world.

While in the early game programming courses the instructor typically emphasize programming skills and OOP program organization, the advanced courses are used as a platform for training software engineering skills. The teams typically work on a single project for the entire semester, which requires the students to manage their time and make critical decisions during the development process. Team communication is crucial and specialized tools, such as TeamSpeak and Slack, must be learned and used throughout the semester. Source code control and sharing software, such as GitHub, TortoiseSVN, or Perforce, are necessary even for small teams. Students may apply general software engineering concepts that they learned in another class and, in addition, learn some that are specific to game development.

In earlier classes, students' efforts are concentrated on making their games work. In later classes, their assignments begin with the design process. Students must start by making design decisions about the game they want to make, such as the genre, theme, and scope of content creation. They must choose, adjust, and justify all the game aspects, from basic mechanics to storyline. Students must design an interface and choose from different ways of delivering narrative text or tutorial help. They learn to pitch their game to an audience and change plans based on feedback. Moreover, students must balance the time they have with the skills they share collectively to decide how much they can accomplish in one semester.

Along the way, students learn a considerable amount of content related to the entire game development process from concept to delivery. If the class is using the agile development methodology, as most do, students must deliver and present iterations of the game on a regular basis, as well as important milestones for the game, including the first playable version, alpha, beta, and final release by the end of the semester. Students are confronted with feedback about their games when their games are being playtested by participants with different levels of player expertise and different preferences, from the team members themselves to classmates to possibly complete strangers. The instructor must also rely on peer feedback from team members about themselves, each other, and the project in order to assign grades. This feedback must reflect not only the overall achievement of the project, but also the contribution of each team member to the effort.

These kinds of courses help students transition from solo programmers to team members and even project managers. Students learn collaborative and leadership skills that they will need later in the workplace.

5 Conclusions

In this paper we examined various ways in which game development content can help achieve learning objectives in a computer science program as students progress through the curriculum. We started with beginner courses where writing small games can make learning to code more fun and attractive. In more advanced early core classes, game programming can be used as concrete examples for many of the algorithms or data structures that students must learn. Medium level courses are where students learn fundamental concepts related to games while improving their programming skills through challenging practical exercises. Finally, advanced game development classes teach collaboration, project management, and software engineering skills. Thus, game programming can improve computer science education at all levels with content that is attractive and motivating for students.

6 References

[1] Ian Parberry, "Challenges and Opportunities in the Design of Game Programming Classes for a Traditional Computer Science Curriculum", *Journal of Game Design and Development Education*, Vol. 1, pp. 1-17, 2011.

[2] U. Wolz, T. Barnes, I. Parberry, M. Wick, "Digital Gaming as a Vehicle for Learning" In *Proceedings of the 2006 ACM Technical Symposium on Computer Science Education*, pp. 394-395, Houston, TX, Mar. 1-5, 2006..

[3] H. Bourdreaux, J. Etheridge, and A. Kumar, "Evolving Interdisciplinary Collaborative Groups in a Game Development Course"; *Journal of Game Design and Development Education*, Vol. 1, pp. 25-37, 2011.

[4] H. Hakimzadeh, R. Adaikkalavan, and J. Wolfer, "CS0: A Project Based, Active Learning Course", *International Transaction Journals of Engineering Management and Applied Sciences and Technologies*, http://www.tuengr.com/Vol 2(5), 2011.

[5] H. Bort, D. Brylow, M. University; and M. Czarnik, "Introducing Computing Concepts to Non-Majors: A Case

Study in Gothic Novels", *Proceedings of the Special Interest Group in Computer Science Education Conference (SIGCSE 2015)*, Kansas City, 2015.

[6] D. Toth and M. Kayler, "Integrating Role-Playing Games into Computer Science Courses as a Pedagogical Tool", *Proceedings of the Special Interest Group in Computer Science Education Conference (SIGCSE 2015)*, Kansas City, 2015.

[7] C. Latulipe, N. B. Long, C. E. Seminario, "Structuring Flipped Classes with Lightweight Teams and Gamification", *Proceedings of the Special Interest Group in Computer Science Education Conference (SIGCSE 2015)*, Kansas City, 2015.

[8] M. Sanchez-Elez and S. Roman, "Learning Hardware Design by Implementing Student's Video-Game on a FPGA", *Frontiers in Education: Computer Science and Computer Engineering*, Las Vegas, NE, July 27-30, 2015.

[9] M. Mejias, K. Jean-Pierre, Q. Knox, E. Ricks, L. Burge, III, and A. N. Washington, "Meaningful Gamification of Computer Science Departments: Considerations and Challenges", *Frontiers in Education: Computer Science and Computer Engineering*, Las Vegas, NE, July 27-30, 2015.

[10] D. Rosca, J. Kostiou, and C. Locke, "Video Games to Motivate a Multi-disciplinary Capstone Experience", *Frontiers in Education: Computer Science and Computer Engineering*, Las Vegas, NE, July 27-30, 2015.

[11] E. Bachu and M. Bernard, "Visualizing Problem Solving in a Strategy Game for Teaching Programming", *Frontiers in Education: Computer Science and Computer Engineering*, Las Vegas, NE, July 21-24, 2014.

[12] Pong, Atari, http://www.ponggame.org/.

[13] PacMan, Namco Bandai Games, http://www.bandainamcoent.com/game/pac-man.html.

[14] Sokoban, Thinking Rabbit, http://sokoban.info/.

[15] Breakout, Atari, www.atari.com/arcade.