TEACHING INTRODUCTORY PROGRAMMING WITH GAME DESIGN AND PROBLEM-BASED LEARNING

Andrey Soares, Southern Illinois University Carbondale, asoares@siu.edu
Frederico Fonseca, Pennsylvania State University, fredfonseca@ist.psu.edu
Nancy L. Martin, Southern Illinois University Carbondale, nlmartin@siu.edu

ABSTRACT

Teaching and learning in the introductory programming course can be difficult. Numerous studies address this problem with none resulting in a perfect approach for either students or instructors. The authors of this paper found success in the introductory programming course by implementing a problem-based learning environment and by delivering the entire course in the context of game design. This paper discusses the course structure, results of a student survey about the course, and the instructor's reflections about various aspects of the course.

Keywords: Introductory Programming, Problem-Based Learning, Game Design, Pedagogy

INTRODUCTION

Learning to program is challenging for many students [16]. Introductory programming course failure rates are high, exceeding 30% and even 50% for some programs [3], and students report a fear of learning to program [17]. Teaching programming can also be difficult because a programming course requires much more than imparting knowledge. Rather, instructors are expected to foster a learning experience with "a project realistic enough to excite interest, while providing structured problems for them [students] to practice and learn the basic skills" [9, p.12]. Complicating matters for the instructor is the fact that different students approach the task of learning to program in different ways [6] and often have varying degrees of prior knowledge and skill level. As a result, instructors struggle with finding the perfect pedagogy for teaching introductory programming.

The authors of this paper have found success teaching the introductory Java programming course by incorporating problem-based learning and game design. This approach was repeated in three course sections over two semesters at a large Midwestern university. At the end of each course, student perceptions were recorded in anonymous surveys. This paper provides a brief summary of background material, describes aspects of the course structure, content and delivery, reports the results of the student surveys, and presents reflections of the authors' experience teaching courses with this approach.

BACKGROUND

Introduction to Programming

Learning to program is difficult because it requires the absorption of complex new knowledge, strategies, and skills [16]. Not only must students learn about programming structures, specific language syntax, and other basic concepts, they must also quickly begin to construct strategies for combining the new knowledge for solving problems via computer code. The task is then still not finished because the code must be tested, debugged, and often reformulated and optimized. The novice's principal problem area is not primarily in language or syntax, but rather in program planning [20, 24] and generation [16] which is a direct result of deficient problem solving skills. With these difficulties in mind, the authors designed their course incorporating problem-based learning and game design.

Problem Based Learning

Problem-based learning (PBL) is a method first developed in medical schools and is usually attributed to Howard Barrows [1]. In PBL students are given the basic tools necessary to tackle some real world problem, assignment, or task. The problem is often ill-structured where an obvious and easy solution is not visible. In confronting and solving ill-structured problems, students are expected to develop content knowledge, evaluate and select strategies for a solution, and develop skills in defending their preferred solutions. PBL has been shown to positively affect the withdrawal rate in introductory programming courses [14]. Students have reported increased motivation and social/emotional support from the collaborative work arrangement in PBL [14]. O'Grady [15] conducted a thorough review of PBL in computing education literature and found that both instructors' and students' experiences with PBL are broadly positive. PBL was used as the primary organizing method for this course.

Game Design

In an effort to attract and retain students in computing programs, a variety of novel approaches to the introductory programming course have been reported. Incorporating game design and programming into introductory courses is expected to build excitement and enthusiasm for computing in general [12] and game-based courses have been successful in this regard [e.g., 2, 9, 21, 22]. For game-based courses to be successful, the instructor should possess some level of prerequisite knowledge about gaming and game design, and the course must retain focus on core programming concepts [21]. The skills of the instructors of this course, along with other considerations, led the way for the redesign of the introductory programming course to center around game design.

COURSE STRUCTURE

This introductory programming course is offered in the Information Systems Technologies (IST) department at a large Midwestern university. It is a required course for majors in IST and for majors in the Electronic Systems Technologies (EST) program. The course is offered over a period of 16 weeks, and classes meet twice a week for one hour and 15 minutes. Because this is a required course in some majors, it is usually offered multiple times per year with two sections in the spring semester and one section in the fall semester.

Java and IDE

While the main emphasis of the course is on the logic used in the process of building a computer program, it is also expected that students learn the syntax of the commands for the language in use, in this case, Java. This can be a daunting task, especially for those with no prior knowledge of programming [16]. While some instructors prefer to use a simple text editor with little or no support for code completion (e.g., Notepad), others may choose to adopt some type of IDE (i.e., Integrated Development Environment) such as Eclipse or Netbeans. The IDE adopted for this course is Eclipse.

One advantage of an IDE is that all processes to compile and run applications are hidden from students, reducing the time spent with routine tasks. A second valuable feature of an IDE is the content assistant, which suggests code options based on what is being typed. This feature promotes PBL because students are stimulated to explore new code options, which can quickly be done with the help of content assistant. For instance, students may know what behavior they want for the program, but they may not remember the correct syntax. More often than not, students use content assistant to explore and learn about the different methods available for the classes needed for their application. As a result, students use some functions not covered in the course material by exploring the content assistant.

Topics Covered

This introductory programming course is designed to cover the fundamentals of programming, graphical user interface (GUI) programming, and object-oriented programming. Depending on the students' major and interests, this may be the only programming course they will take throughout the curriculum. The course is roughly divided into three blocks with each block lasting approximately five weeks. The 16th week is used for completing and presenting the final projects (i.e., games). Table 1 displays a list of the topics covered for each of the three blocks.

The topics are presented in a cumulative fashion and previous concepts are often revisited and reused with new topics or assignments. With this strategy, the class can move quickly through the basic concepts because they will be revisited later in the course in different forms [9]. For instance, condition statements are discussed over only one week, and most of the practice is left to future assignments. In other words, instead of spending another week or so practicing the different ways if/else statements can be used, students learn about them as they need to apply the concepts in different assignments. In addition, most of the topics are presented with game design in mind to help students associate how the concepts learned could be used as part of a game.

Table 1. Course Topics

Block 1: Fundamentals of Programming			
Introduction to Programming	Input/Output	JOptionPane	Increment/decrement
Variables	Operators	If/Else statement	Escape Characters
Data types	Import	For/While statement	Math.Random()
Block 2: Introduction to Object-Oriented Programming and GUI Programming			
Class – Attributes, Methods, Constructor, Instances	Inheritance	Slider	Mouse Motion (with brief intro to arrays)
Frame	Layouts	Textfield	Mouse Click
Panel	Buttons	Keyboard	Listeners
Buttons	Images	Timer	PaintComponent
Block 3: Final Project			
Game Design	Java Examples		

Arrays are briefly introduced toward the end of Block 2 when the students are learning about mouse motion to create their own PaintBrush application. At this point, students see the need to save the color of each pixel and the mouse coordinates to repaint the drawings, for example, after the screen is re-sized. Many students take on the challenge to learn more about arrays on their own in order to implement a feature in their games that uses arrays.

Assignments

A variety of activities comprise course assignments including exercises, labs, lab tests, and a final project. Exercises are in-class activities that students complete, usually with the instructor, to introduce programming concepts (Concept Learning). For example, the instructor discusses the concepts of a class Student with its attributes and methods, and then conducts a demo to illustrate the code used to create the class and some instance. Labs are in-class or outside of class activities for students to try the new concepts (Concept Practicing). In this case, a lab may require students to create classes for other objects such as a professor, a player, or a dog. Lab tests are labs that students are required to complete during a class period with closed book/notes to demonstrate their proficiency with some concepts. The course culminates with a group project to create a game that requires students to use the concepts learned throughout the semester (Concept Application). However, students also learn new concepts on their own because they want to implement certain features in their games. Project requirements are that the game should include the following features: (1) welcome panel, (2) setup panel (3 options with 3 sub-options each), (3) instructions panel, (4) game panel, and (5) score panel. The games are evaluated based on 10 criteria including understandable and explainable Java code, number of bugs, complexity, game set up options, user friendliness, functionality, graphics, game difficulty, level of entertainments, and teamwork.

COURSE EVALUATION

At the end of the Fall 2010 and the Spring 2011 semesters, students were asked to complete a paper survey to evaluate this Introduction to Programming course. The survey was exploratory in nature with no predicted outcomes and its aim was to collect students' feedback about their learning experiences and their perceptions of the course. A brief description of the survey and class demographics are presented next.

Method

A total of 44 students completed the survey, 19 in Fall 2010 and 25 in Spring 2011. Students were informed that answering the survey was voluntary and anonymous, and the survey was not related to grades for the course. The survey contained 20 questions with 65 unique data points on a variety of topics related to the course. Questions asked for demographic information, previous programming experience, perceptions of the course, use of games and the PBL approach, learning experience related to programming concepts, soft skills, interest in programming and more. Most questions, other than demographic-related and categorical ones, were answered on a 5-point Likert scale.

Demographics

Forty-four students answered the survey. The students' ages ranged from 19 to 54 years old, with 32 students of traditional college age (i.e., < 25 years old), 9 students of non-traditional college age (i.e., ≥ 25 years old), and 3 students who did not report their age. The majority of students were majors in the IST or the EST programs where the introductory programming course is a core requirement. The other students were from Health Care Management, Technical Resource Management, and Undecided. Table 2 displays the sample breakdown by gender and class level. More than half of the students, 60.6 percent, reported little or no prior experience with programming. Other relative survey results are incorporated into the next section.

Table 2. Sample Demographics

	Freshman	Sophomore	Junior	Senior	Total
Male	2	8	19	11	40
Female	1	0	3	0	4
Total	3	8	22	11	44

INSTRUCTOR REFLECTIONS

In this section, we discuss several topics that were of particular importance to the success of the course. Where relevant, data from the student surveys are reported as well.

Book or No Book

This course has been offered with and without a required textbook (i.e., Spring 2011 and Fall 2010, respectively) with no noticeable difference in the outcomes in terms of student success. However, requiring a textbook did result in some student concerns because the course is structured in a way that does not align well with current textbook offerings. While most Java courses cover the same programming topics, this course combines topics in a fashion that required students to jump from chapter to chapter to discuss specific concepts. As a result, students complained that they could not follow the book explanations or the explanations were not directly related or applicable to other course materials. This was especially true for the programming concepts needed to build the games. For example, instructors would start with a concept in chapter 3, then jump to chapter 14 to review supplementary GUI programming concepts, and then go back to chapter 2 to discuss a topic needed for implementing a feature in the game. This approach was not very popular among students, although some did prefer having a physical textbook in which to write notes. This issue is not a criticism of any particular textbook, but rather a consequence of course design.

The course has also been taught without a required textbook. In this scenario, most of the material needed was presented by the instructor in the form of PowerPoint slides or as links to online materials. Since Java is such a popular language, it is not difficult to find examples to illustrate specific concepts. However, because of the complexity of how some examples are presented, the instructors developed simplified samples to show how a specific feature works (e.g., slider, dropdown list, menu, dialog box, etc.). As the course progressed, the instructor released small code samples to demonstrate the use of some Java functions and these served as resources to later create the games.

Overall, the instructors favor the no-book approach. In a PBL environment, the lack of a textbook motivates students to find usable materials and to learn how to adapt the material to their programming needs. Learning "how to learn" is a crucial skill that students will often need in a field such as Information Technology that is constantly changing. The course survey results reflect a significant increase in students' confidence in their ability to develop a game by the end of the semester, and the instructors attribute that increase to the students' involvement in a PBL environment. Specifically, at the beginning of the semester, students were shown examples of games created by former students and only 27.3 percent believed they could learn the skills necessary to develop a similar game. However, at the end of the semester, 79.1 percent felt confident that they possessed the skills needed to develop a game.

Final Project Game Selection

In earlier offerings of the course, the instructors proposed themes for the games. For example, the theme "a day in the life of a student at the university" would likely produce games with squirrels crossing the campus, athletics games or other activities at the football stadium, or students and faculty interacting in the classroom. For the course sections described in this paper, students were allowed to select and create their own games. This approach is designed to encourage students' creativity and make it more appealing for them to get involved and to explore programming skills. It is expected that students would be more enthusiastic about the project if they are building something related to their interests as opposed to following a requested theme imposed by the instructor. Student responses in the survey support this approach as 75 percent report the course was "fun" or "awesome", and 56.8 percent said that the course increased their interest in programming.

Around week 9 or 10, when most of the essential programming concepts needed to create a game have been covered, students submit a proposal for their group's final game project. Usually students tried to recreate some of their favorite games or proposed a game that combined features from two or more known games. All games must be approved by the instructor, who assesses the complexity of the game and its feasibility to be completed within the remaining weeks of the semester. This is a crucial decision that can impact the completion of the project and the students' learning experience. A large or incomplete project may cause stress and frustration. Thus, it is important for the instructor to pay close attention to the students' skills with Java to be able to complete the proposed project, both individually and as a group. Although it is more work for the instructors, allowing students to select their own game theme is more popular with the students and results in a wider variety of creative projects.

Teaching/Learning Assistants and Instructor Support

The instructor utilizes both teaching assistants (TA) and learning assistants (LA) for this course. The TA's primary duty is to support the instructor's teaching activities and grade assignments. A TA also provides support in class and schedules office hours to help students learn concepts and complete assignments. A LA, on the other hand, is in class only to improve the students' learning experiences. LAs are students that have previously taken the course and have volunteered to assist in class as coaches and mentors for students currently enrolled in the course. One of the motivations for volunteering is that LAs understand that they can learn more and improve their own programming skills by helping others to build their games. Before the semester starts, the instructor advises the assistants on how to provide support to students in class. In particular, they are instructed to not give all the answers to students, but to encourage them to think about the problems they are facing and propose their own solutions, which in turn, reinforces the PBL environment.

The use of TAs and LAs has several benefits. When rating factors that contributed to their success in the course, students ranked the availability of TA/LAs second only to the availability of the instructor. Students commented that they liked having TAs and LAs in class for extra help. In addition, students stated that because of the quick access to help in class, they were more willing to try new things and explore the Java language. Having LAs in class also shows the students that the course may be challenging, but it is rewarding enough that some students come back as volunteers to help. This peer level support provides encouragement and reassurance to students that they will learn the content.

Lectures and In-Class Work

The lecture materials for each class period are built upon the previous ones to give students a sense of continuity and to help them realize how all the pieces are integrated to become a complete application by the end of the semester. Instructors repeatedly connect lectures and lab assignments to some part of the final game project to help students understand how even small pieces learned throughout the semester are necessary for building the final project. For example, when discussing the concept of variables, examples applicable to the game such as a variable to store the player's score, the energy level of a character in the game, the position of a bullet on the screen, the number of lives remaining, or the difficulty level are used. When discussing condition statements, the example of the bullet moving across the screen and hitting an object is offered. For example, if the object hit is one of the enemies on the screen, then show an explosion and sound, and increase the content of the variable score points. Using the context of a game to illustrate programming concepts seems to motivate the students to learn the course materials [9]. In particular, this approach is useful for emphasizing not only the implementation, but also the planning of an application.

An important aspect of this course is to provide the students numerous opportunities to learn-by-doing and to ask questions. To allocate time for practice, short lectures are delivered to introduce specific concepts relevant to the overall knowledge needed to complete the final projects. For a course that meets twice a week for one hour and fifteen minutes per class, the lecture time for introducing new material takes approximately 20-25 minutes, leaving the remaining minutes of class time for hands-on assignments. During in-class work, the instructors and TA/LAs act as coaches and tutors, providing feedback and engaging students in dialog that encourages them to think about possible solutions to the problems with their programs. Instructors and TA/LAs will, whenever possible, answer a student question with another question or redirect the question to another group member with the intent to start a discussion. Sometimes, by thinking about the instructor's question or reformulating their own questions, students find the answer, or they can identify a clue during the conversation that may help them find the answer. In this case, the discussion that started as a question may be interrupted with students saying "Never mind, I just realized X" or "I will try to adapt what we learned in lab N". The in-class work time combined with the availability of the instructor and TA/LAs allows students to work on assignments with on-demand support. In rating course success factors, 65.9 percent of students reported that time in class for hands-on work was key.

Individual and Group Work

Some course assignments are completed individually to provide opportunities for students to practice and to develop their skills as well as to help the instructor assess students' progress. But the course is also designed around collaborative learning which generally refers to a method that involves two or more people performing some learning activity together in a truly joint effort [8]. Studies of collaborative learning in the introductory programming course have had favorable results with students showing increased programming achievement and satisfaction with the course [18]. The collaborative environment of this course received favorable ratings in the student survey. Group work was listed as one of the top five course success factors by 65.9 percent of students, and 68.2 percent felt they learned teamwork skills as a result of group work.

In this course, the majority of activities are group assignments where students are placed in groups of three based on individual skill levels. The approach for creating groups is to place students with similar skills in the same group, as opposed to a mix of students with different skill levels. Students with previous programming experience tend to be more comfortable with the fundamentals of programming and spend more time trying to learn new features of Java that can be applied to the game. In contrast, a student that is new to programming spends more time practicing the fundamentals, but also tries to apply what they are learning to the game. Moreover, pairing on skill level has been shown to help preclude many of the challenges related to partner compatibility [e.g. 4, 13]. Due to the diverse skill levels of the groups, the games produced display different levels of complexity. Nonetheless, the programs are still challenging for the students and provide opportunities for learning and improving programming skills at any level.

Weekly Logs

Every week students are required to submit a weekly log in an effort to encourage active learning [5, 9]. Active learning is an educational approach that places more responsibility on students for their own learning by specifically implementing activities and causing students to think about or reflect on those activities. The weekly log is usually a paragraph that includes students' reflections about the course and learning experiences for the previous week of class. Students are instructed to describe strengths and weaknesses of the course and to provide comments or suggestions to improve the course. Typically, students would refrain from discussing the course or their progress during office hours or in class, but with the weekly log, they have a way to openly communicate with the instructor.

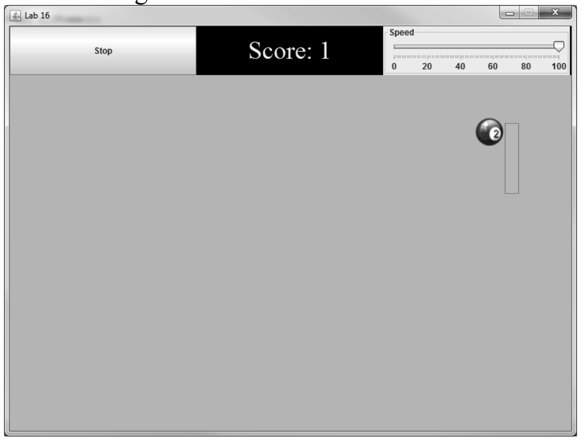
The weekly log helps the instructor to identify issues that may be interfering with the students' learning experiences both in class and outside of class. It can also provide clues about how to improve the course and help the instructor assess students' progress. For example, a log from the first or second week of class will show excitement about the possibility of creating a game as part of a course. It will also show the struggles and successes with the new material. As the course progresses to more complex structures, the logs change and reflect some calls for help, but still also document excitement about applying the knowledge to the final project. During the weeks before work begins on the final project, the logs start showing a mix of excitement (e.g., I am ready to start building my game) and frustration (e.g., No way that I will be able to finish the game). These variations are expected and can be used as check points for the instructor to gauge the course progress. In particular, the logs help to identify students or groups that need assistance and course materials that need reinforcement. The instructor must pay careful attention to the logs to avoid more frustration than planned, and to keep measuring the "pulse" of the course.

Learning by Examples

Researchers report that students prefer learning from examples and actually learn more from studying examples, and that presenting a variety of examples helps students recognize plans and strategies related to programming [7]. Others found that "examples are beneficial when provided before problems…[because] worked examples…can guide future problem solving" [23, p. 217]. By presenting examples of previous games or parts of a game prior to students attempting solutions, they are better able to connect the new material with the expected final project. Students in this course found the use of examples very helpful with 72.7 percent reporting that the use of Java examples contributed to their success in the course.

Many examples discussed in class are related to games, from tossing a coin or playing rock-paper-scissors to designing a simple "Pong" game or building a sample "Paintbrush" application (Figure 1). The Half-Pong lab was an opportunity to illustrate the basic components of a game, including the use of and communication between classes, timer control, movement, bouncing the ball on the screen, controlling the keyboard to move the paddle, starting and ending a game, using controls to change the speed of the game, scoring points, and more. Later, students were challenged to complete the full Pong game by including another paddle for a second player, setting up keys to control the second paddle, and adding a new score board for the second player. Similarly, the Paintbrush lab was used to demonstrate the use and control of the mouse and how buttons from a control panel could change the application (e.g. changing the color or the size of the brush).

A. Half-Pong Game



B. Paintbrush

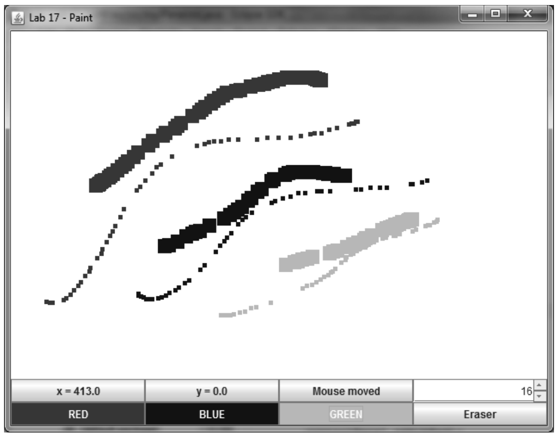


Figure 1. Sample Class Assignments

The example of the Pong game was used as the starting point for many games created as final projects. For example, one group of students changed the position of the paddle to create a breakout type of game (Figure 2). Then, the students reused the concept of a paddle (without moving it) to create several bricks on the screen. Using the random function learned earlier in the semester with the toss coin/dice labs, the application randomly assigned colors and points for each brick. Students also kept the bouncing ball created in the Pong game to bounce after hitting a brick.

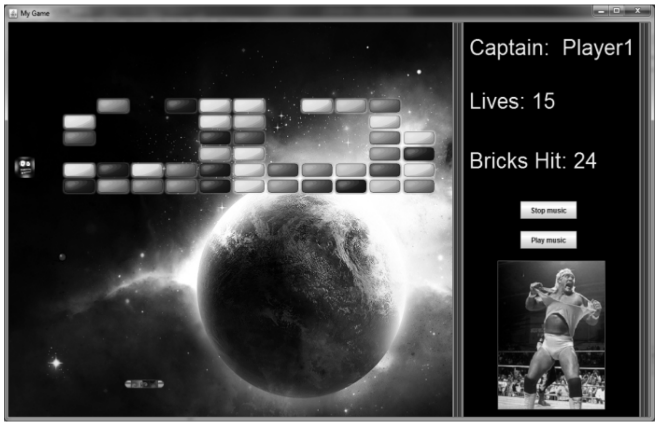


Figure 2. Breakout Game (Spring 2011)

The use of examples to illustrate programming concepts and code that could be implemented in a game indeed helped students learn and explore new uses for the things they had already completed in class. Some students stayed close to the topics discussed in class when proposing their games, while other students went above and beyond the topics discussed to create their games, requiring them to learn new materials/concepts and to adapt them to their games. It is important for the instructor to assess the proposed game and the students' skills and motivation to complete the game. We would caution that presenting the same game in examples throughout the semester to explain different topics may influence the students' choices for a final project. For instance, a shooting game was used many times to illustrate different programming concepts and to exemplify a great game created in class. As a result, some of final projects were closely related to the shooting game discussed in class. Because we did not show any code for the shooting game, we still considered that students trying to implement a similar game was a very positive learning experience and a great practice of their programming skills.

The Final Games

Each time the course is taught, the instructors are impressed with the games created, considering that it is an introduction to programming course, and that many students have little or no programming experience prior to this course. On the very first day of class, we play 5 to 10 games created by students in previous classes, and we ask students to evaluate them. First, students must be reminded that these are not Wii or Xbox-quality games. Rather, they are games created by students previously enrolled in the course, and they should provide an idea of what can be accomplished in just one semester of learning how to program with Java.

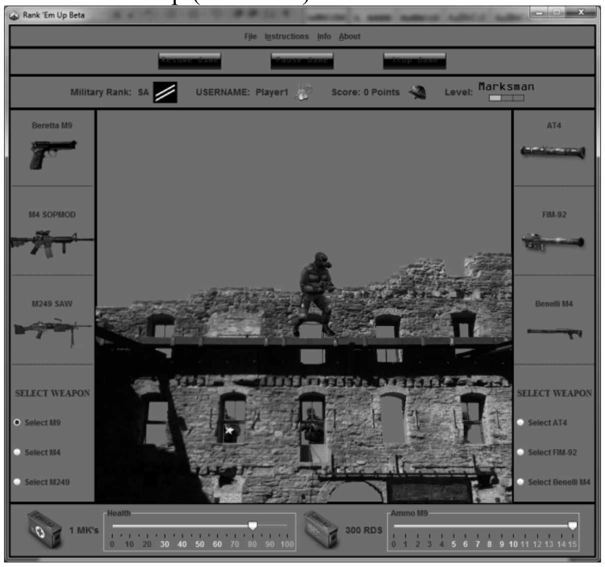
Then the instructor leads a discussion about the evaluation of the games, and students create various criteria for measurement. After evaluating the games and discussing the reasons for selecting the criteria and the relative points for game evaluation, students are informed that some of the criteria they created will be used to evaluate their own games at the end of the semester. Table 3 provides several examples of criteria used to evaluate the games.

Table 3. Sample Criteria Used to Evaluate the Final Games

Category	Points
Functionality	1 – Controls do not work as intended. Game does not respond to user actions 5 – Controls respond relatively well to user actions 10 – Game runs correctly and responds to all inputs. Players' actions directly affect gameplay visuals, sounds, etc.
Graphics	1 – No color, black and white only, bad character design 5 – Some color, no level variation of color and design 10 – Great design and variety of colors
Entertainment	1 – Quite boring, quit after the first game or even before the first game is over 5 – A bit entertaining, keeps you interested through at least one game. Some variation in each level 10 – Play multiple rounds of the game. Each level is uniquely different

Recall that students create the games working in groups that are composed of students with similar programming skills. So, inevitably the final games will have varying levels of complexity in the game itself and in the Java code. Nonetheless, as expected in a PBL approach, learning occurs during the process of building the games as students apply the knowledge acquired throughout the course and look for solutions to implement features for their proposed games. Based on our experience, students will, more often than not, propose a feature that requires more code than what was discussed during the course. As expected, students' motivation to complete their games drives their interest in learning new materials that can be applied to their projects (i.e., solving problems). Figure 3 shows some examples of the games created in the courses described in this paper.

A. Rank 'Em Up (Fall 2010)



B. Jewel Thief (Spring 2011)

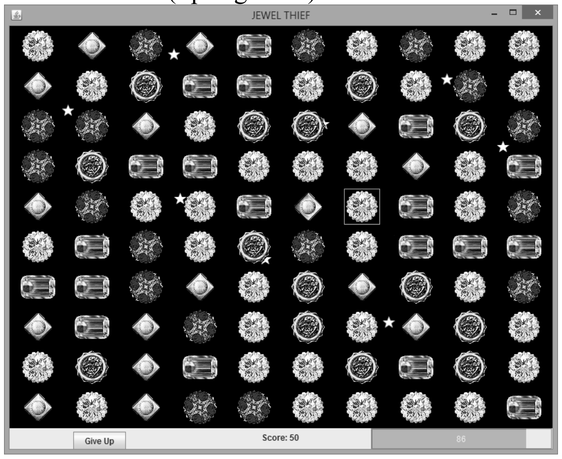


Figure 3. Sample Final Projects

CONCLUSIONS

Learning to program can be difficult for students, and finding the perfect pedagogy for teaching introductory programming can be difficult for instructors. A great deal of research focuses on this challenge [e.g., 16, 19] still providing no perfect solution. This paper has reported on an approach that incorporates game design in a problem-based learning environment for the introductory programming course. Some researchers have reported that the primary problem for novice programmers is not the acquisition of facts and concepts, but how to apply them [11], or failing to apply them [24], and a PBL environment directly addresses those problems. Moreover, the use of examples, particularly examples of prior classes' games, motivates current students to seek the necessary skills and to learn how to apply programming concepts to create their own games for the course.

Teaching programming within a game design context arouses students' interest because most of them, if not all, are interested in playing computer games. This approach fits well within a PBL environment because as Hmelo points out, "the final goal of PBL is to help students become intrinsically motivated. Intrinsic motivation occurs when learners work on a task motivated by their own interests, challenges, or sense of satisfaction" [10, p. 241].

The instructors of this course believe that this combination has been successful, and student survey results support that belief. Even though most students, 59 percent, felt the course was hard or extremely difficult, 75 percent reported that the course was either "awesome" or "fun" and 79.6 percent enjoyed creating a game for the final project. In this paper, we have offered a number of reflections about the course that hopefully will aid other instructors in finding a successful approach for their introductory programming courses.

REFERENCES

1. Barrows, H.S. and Tamblyn R.M. (1980). *Problem-based learning : An approach to medical education*. New York: Springer.
2. Bayliss, J.D. (2009). Using games in introductory courses: Tips from the trenches. *SIGCSE Bulletin*, *41*(1), 337-341.
3. Bennedsen, J. and Caspersen M.E. (2007). Failure rates in introductory programming. *SIGCSE Bulletin*, *39*(2), 32-36.
4. Bevan, J., Werner L., and McDowell C. (2002).Guidelines for the use of pair programming in a freshman programming class. Proceedings from *15th Conference on Software Engineering Education and Training*. Washington, DC: IEEE Computer Society.
5. Bonwell, C.C. and Eison J.A. (1991). *Active learning: Creating excitement in the classroom*. Washington, D.C.: School of Education and Human Development, George Washington University. 3-16.
6. Bruce, C., Buckingham L., Hynd J., McMahon C., Roggenkamp M., and Stoodley I. (2004). Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education*, *3*, 143-160.
7. Caspersen, M.E. and Bennedsen J. (2007).Instructional design of a programming course: A learning theoretic approach. Proceedings from *Third International Workshop on Computing Education Research*. Atalanta, Georgia.
8. Dillenbourg, P. (1999). What do you mean by collaborative learning? In P. Dillenbourg (Ed.), *Collaborative-learning: Cognitive and computational approaches* (pp. 1-19). Oxford: Elsevier.
9. Fonseca, F. and Spence L. (2014). The Karate Kid method of problem based learning. In J.M. Carroll (Ed.), *Innovative practices in teaching information sciences and technology: Experience reports and reflections*, (pp. 9-17). Switzerland: Springer International Publishing.
10. Hmelo-Silver, C.E. (2004). Problem-based learning: What and how do students learn? *Educational Psychology Review*, *16*(3), 235-266.
11. Lahtinen, E., Ala-Mutka K., and Jarvinen H. (2005). A study of the difficulties of novice programmers. *SIGCSE Bulletin*, *37*(3), 14-18.
12. Lewis, M., Leutenegger S.L., Panitz M., Sung K., and Wallace S.A. (2009). Introductory programming courses and computer games. *SIGCSE Bulletin*, *41*(1), 204-205.
13. McDowell, C., Werner L., Bullock H.E., and Fernald J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, *49*(8), 90-95.
14. Nuutila, E., Törmä S., and Malmi L. (2005). PBL and computer programming — the seven steps method with adaptations. *Computer Science Education*, *15*(2), 123-142.
15. O'Grady, M. (2012). Practical problem-based learning in computing education. *Transactions on Computing Education*, *12*(3), 1-16.
16. Robins, A., Rountree J., and Rountree N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137-172.
17. Rogerson, C. and Scott E. (2010). The fear factor: How it affects students learning to program in a tertiary environment. *Journal of Information Technology Education*, *9*, 147-171.
18. Sabin, R.E. and Sabin E.P. (1994). Collaborative learning in an introductory computer science course. *SIGCSE Bulletin*, *26*(1), 304-308.
19. Sheard, J., Simon S., Hamilton M., and Lönnberg J. (2009).Analysis of research into the teaching and learning of programming. Proceedings from *Fifth International Workshop on Computing Education Research*. Berkeley, CA.
20. Spohrer, J.C. and Soloway E. (1986). Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM*, *29*(7), 624-632.
21. Sung, K. (2009). Computer games and traditional CS courses. *Communications of the ACM*, *52*(12), 74-78.
22. Sung, K., Panitz M., Wallace S., Anderson R., and Nordlinger J. (2008). Game-themed programming assignments: The faculty perspective. *SIGCSE Bull.*, *40*(1), 300-304.
23. van Gog, T., Kester L., and Paas F. (2011). Effects of worked examples, example-problem, and problem-example pairs on novices' learning. *Contemporary Educational Psychology*, *36*(3), 212-218.
24. Winslow, L.E. (1996). Programming pedagogy - a psychological overview. *SIGCSE Bulletin*, *28*(3), 17-22.