Caribbean Teaching Scholar
Vol. 1, No. 2, November 2011, 93–113

ERA
EDUCATIONAL RESEARCH ASSOCIATION

Using Gaming to Improve Advanced Programming Skills

Colin A Depradine

Department of Computer Science, Mathematics and Physics, The University of the West Indies, Cave Hill, Barbados

The successful mastering of any new programming language or methodology can be a difficult task for students to achieve. The abstract and mathematical nature of computer programming has been noted as one of the major stumbling blocks students face. In fact, this issue has been identified as one of main reasons for the declines in enrollment in some Computer Science programmes around the world. To overcome these issues, a number of new teaching approaches have been formulated and applied to introductory programming courses. One such approach is electronic or computer gaming, which is used to wrap some of the mundane, abstract aspects of computer programming in a more approachable form. Computer games can also be used to create a learning environment that allows students to explore and experiment with the new concepts using representations that they can relate to. This article presents the preliminary results of using computer games as a mechanism for introducing students to fundamental object-oriented programming principles in an advanced programming course. Using this approach, the performance of students improved in both coursework and examinations, when compared to previous years.

Keywords: Object-Oriented Programming, Games, Java

Introduction

Object-oriented programming has become a required software development skill for the modern software developer. It is a computer programming technique that allows the software developer to focus on the data and the processes that act on it. This approach has been found to be highly beneficial to programming in general, and as a result, the most popular programming languages and development environments now utilize some form of object-oriented programming methodology. With this in mind, progressive Computer Science or Information Technology degree programmes include object-oriented programming in some form.

The use of the object-oriented programming paradigm brings several advantages and benefits to the software developer. It is the significance of these benefits that has led to the rapid rise of this approach in the programming world. However, there is a tradeoff and that is the difficulties encountered teaching this methodology to students, especially to novice programmers. While the advanced features of this approach do bring important benefits, the increase in complexity makes it difficult for students to grasp the underlying concepts.

At the University of the West Indies, Cave Hill Campus, in Barbados, object-oriented programming (OOP) has formed part of the Computer Science and Information Technology majors since 2001. To help ease students' introduction to this approach, the OOP course is taught as an advanced level II (second year) course. As a result, students would have spent an entire academic year learning fundamental programming concepts before undertaking the advanced OOP concepts.

Using this method, student performance remained relatively consistent throughout the time period 2004 to 2007 (the university switched to the GPA system in 2004). During this time period the number of students receiving grades A and B was higher than those receiving grades C and D. However, from 2008 to 2009, the start of a measurable decline in student performance was noted where the number of grade Bs began to decline (and to a lesser extent grade As) while grades C and D began to rise. To stem this decline, in 2010, several different teaching approaches were considered and included varying classroom teaching methods and offering different types of assignments. Based on its successful use in the teaching of introductory programming, electronic gaming was selected. One of the primary reasons for this choice was the noticeable influence electronic gaming now plays in the lives of students. No longer confined to desktop computers and specialized gaming devices, electronic gaming can now be found on mobile phones and other portable devices such as laptops. In fact, the rise in the use of laptops has resulted in the development of serious games for portable devices. For example, Sánchez and Olivares (2011) demonstrate the use of gaming on portable devices for developing problem solving and collaborative skills in eighth grade students.

With this informal evidence, a preliminary investigation on the use of gaming as an active learning strategy, to improve student performance was carried out over the 2010-2011 academic year. The objective of the study was to determine the feasibility of using active learning strategies, based on 2D gaming, to improve the results of an advanced programming course. The study focused on the actual gaming activities and how they were used in the delivery of the course. The course results for the period 2004 to 2009 are discussed and analyzed. This is then followed by a description of the teaching methodology used and the results achieved.

Background

Over the years, the use of highly visual gaming approaches to teach complex subjects has been steadily increasing (Ebner and Holzinger, 2007; Huang et al., 2010; Chang et al., 2011; El-Sheikh and Prayaga, 2011; Hainey et al., 2011). Games enable the instructor to design scenarios that expose students to specific issues in a controllable manner. Paraskeva, Mysirlaki and Papagianni (2010) provide a summary of the major benefits of game use, which include the following:

- The likelihood of transferring the knowledge and skills obtained increases.

- Further practicing leads to the automatic consolidation of the knowledge and skills in memory. Thus, enabling the learner to focus on new information.
- Learning can be achieved through trial and error, where the student remains the decision maker.
- Immediate feedback is received after each committed action, resulting in an environment that encourages exploration and experimentation.
- Games can motivate the learner and increase his or her confidence.

Also, as demonstrated by Kebritchi and Hirumi (2008), games enable the utilization of a variety of instructional strategies and theories such as direct instruction, experiential learning and discovery learning. By employing a specific instructional strategy during the creation of a game, various learning objectives can be achieved. However, as discussed by Vos, van der Meijden and Denessen (2011) greater benefits can be obtained by the construction of games as opposed to simply playing them.

Today, virtual worlds, 2D gaming environments and simulations form one of the primary gaming approaches for teaching fundamental computer programming concepts (Kelleher and Pausch, 2005; Gestwicki and Sun, 2008; Muratet et al., 2009; Phelps et al., 2009; Carbonaro et al., 2010; Maloney et al., 2010; Wang, 2010; Wang and Wu, 2011). This approach focuses on the concepts of programming, hiding the inherent complexity of the programming languages themselves.

Virtual worlds provide an environment where programming tasks can be performed and instant feedback received. Using the appropriate metaphor, students can easily relate to the tasks since they move beyond the normal abstract representations found in computer science. For example, the *ProgrammingLand MOOseum* uses a museum metaphor that enables students to explore the various concepts of computer programming (Slator et al., 2004). Another example is the *DELSYS* system, by Dagdilelis et al. (2003), which uses a simple programming language to control the elements of a microworld. In fact, the virtual world metaphor can be used to teach basic programming concepts to young children as demonstrated by *ToonTalk*, which has been used by children as young as six years old (Kahn, 2001; Kelleher and Pausch, 2005). One of the more advanced systems to date is the *Alice* system, which uses a 3D world metaphor that enables students to explore the various concepts of computer programming (Alice.org).

However, the use of virtual worlds does have some drawbacks. Firstly, the target audience is usually limited to young or non-computer science students as well as students in introductory programming courses. Secondly, the simplified development environments make transitioning to an advanced programming language difficult.

The issue of transitioning from the simplified virtual world to advanced programming language has proven to be a barrier to the adoption of these systems

by universities for teaching advanced programming courses. In fact, this very issue is being looked at in the development of the new version (version 3.0) of the *Alice* programme (Cooper, 2010). Another teaching environment, *Greenfoot* (Kolling, 2010) tries to manage this issue by providing a closer relationship between the visual approach and the underlying programming language. However, the environment allows short-cuts to be taken that simplify the task of learning programming but also makes the transition more difficult.

The approach discussed in this paper avoids the issue of transitioning by allowing students to develop their software, in the gaming environment, using the pure programming language. At the same time, the use of the gaming environment enables the programming concepts to be represented and experimented with in a form that the students can more easily relate to. In summary, this approach attempts to utilize the benefits obtained with the use of simplified gaming environments in introductory programming courses while at the same time still exposing students to the required advanced programming concepts.

Programme structure and student performance

In this section, the performance of students taking the OOP course from the year 2004 to 2009 is reviewed. This range was selected as a result of two significant changes that were made to the overall method of assessment. The first change occurred in 2004, where the university switched to the GPA (Grade Point Average) system. This changed the entire grade structure for the Faculty. For example, the D grade was introduced. The second change occurred in 2008, where the Computer Science discipline implemented a must-pass coursework and final exam rule. Students, who failed either the coursework or the final exam, failed the entire course.

In terms of the demographics, students in this course would be undertaking either the Computer Science major (single or double) or the Information Technology major (single). Generally, they would be in their second or third year of study (if registered as a full-time student), with most taking the course in their second year.

Object-Oriented Programming Course

At the Cave Hill Campus, the OOP course is an elective for the Computer Science single major and compulsory for the Information Technology single major and the Computer Science double major. It is a level II (second year) course and covers the core areas of OOP, as listed below.

- Fundamental Concepts (e.g. classes and objects, encapsulation)
- Fundamentals of Class Design (e.g. inheritance versus composition relationships)
- Design by Contract (e.g. exception handling)
- Advanced Concepts (e.g. abstract classes)
- Applications using Class Libraries

The latter topic is especially important as it enables students to utilize any programming code provided for the course and associated assignments.

The implementation programming language is the Java programming language which is widely used in the industrial, research and education sectors (Oracle Technology Network for Java Developers). The prerequisites of this course are two level I (introductory) programming courses using the C programming language as the implementation language. Upon reaching the OOP course, students would have undergone a full year of computer programming, learning the fundamental concepts that apply across all modern programming languages and methodologies. As of the year 2008, the OOP course is taught twice within a single academic year, once in each semester. It should be noted that since the course's inception in 2001, coursework is worth 40% of the overall course mark with the final exam constituting the remaining 60%.

Software tools

As in all programming courses, students are required to use a standard set of software tools for practicing and developing their programming skills. In an advanced programming course such as this one, the tools used are generally the same ones found in industry or at the very least, an educational version (lower cost) of the industry standard tools.

For the Java programming language, the standard programming tools are available free on the vendor's website (Oracle Technology Network for Java Developers). These tools are commonly referred to as the JDK (Java Standard Development Kit or Java SDK) and are used by both the commercial sector and the educational sector. While this ensures that the students are exposed to the commercial versions of the tools, there is the drawback of introducing considerable complexity early on in the learning process, making it harder for the students to grasp the concepts taught in the course.

As a result, students are only exposed to the two most important tools needed for programming in Java, the compiler and the interpreter. The compiler checks for syntax errors within the Java code and the interpreter runs the finished product. The other tools that make up the JDK are described but not utilized within the course.

However, it should be noted that some of the material generated for the course is created by two other JDK tools. The documentation that describes how to use the Java code produced for the course was created by the JDK documentation tool, known as Javadoc. This tool produces documentation in a standard web format that can be easily added to any website. The Javadoc tool is also used to document the code provided with the JDK (Javadoc Tool Home Page, 2010). This ensures consistency within the course and exposes students to the format applied in industry. The other tool is an archive tool, called Jar, which packages the course's Java code into a single package that can be easily incorporated by the students into their assignments.

Performance

Prior to 2008, the failure rate for the final exam was high with 44.9% of students failing the final exam in 2004, 43.2% in 2005, 27.3% in 2006 and 50% in 2007. This value dropped to 8.9% in semester I 2008 but climbed to 19% in semester II 2008 and remained at 18% for both semesters in 2009. The significant drop, in semester I 2008, coincided with the institution of the must-pass coursework and final exam rule.

Over the period 2004 to 2008, the average percentage of students gaining an overall grade of C was 18%. In semesters I and II 2009, this percentage rose to 40% and 33.3%. For grade B, from 2004 to semester I, 2008, the average was approximately 20%. This average then dropped to 9% during the period semester II 2008 to 2009. During the entire period, 2004 to 2009, the average number of grade As was 28% with peaks of 40% in 2006 and semester I 2008.

The number of students failing the course overall, dropped from an average of 20% to 9% in semester I 2008. However, the average climbed back to 20% in the remaining years. A similar trend occurred for the percentage of grade Ds, with an average of 17% of students obtaining this grade in 2004 to 2007. Then in semester I 2008, the percentage dropped to 7% but climbed back to an average of 14% in the following years.

To fully understand the significance of the change in the 2008-2009 period, the percentage ranges for each grade is given in table 1. Given the wide range for the A and B grades, the decline in the number of grade Bs and the increase in the grades C and D become noticeable. Also, the large range for grade A probably accounts for the relatively stable results for this grade, across the years.

Table 1. Percentage ranges for each grade.

Grade	Percentage Range
A	67-100
B	57-66
C	47-56
D	40-6
F	0-39

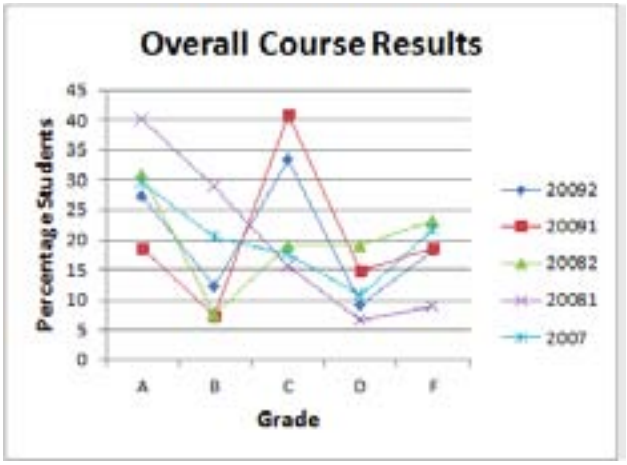

Figure 1. Overall grade results for academic years 2007 to 2009.

Figure 1 shows the percentage grades across the 2007-2009 period. At first look, it appears that semester II 2009 (i.e. 20092 on the graph), shows a reversing of the decline. However, given the large range for the A grade, it is better to view the results in terms of the actual mark obtained. Figure 2 shows the results for the 2008 to 2009 period.

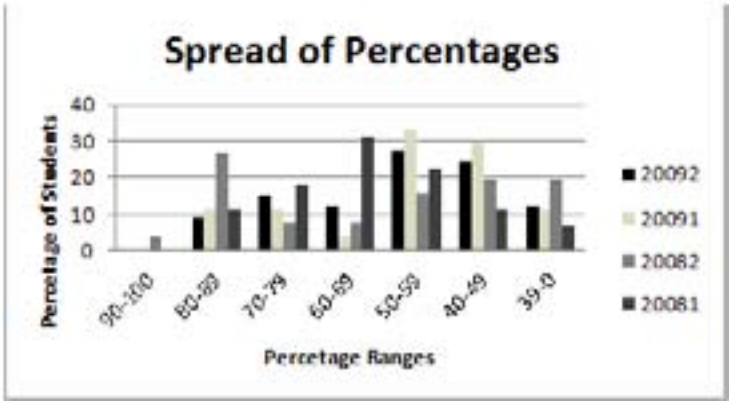

Figure 2. Detailed breakdown of grades for academic years 2008 and 2009.

In Figure 2, for grades 60% and above, semester I, 2008, saw better performances than 2009, which matches the results in Figure 1. In semester I 2008, 60% of the class got above 60%, in semester II 2008, 46%, in semester I 2009, 25% and in semester II 2009, 36%. However, for 2009, the percentage of students with a grade of 70% or above was 22% for semester I and 24% in semester II. In semester I 2009 more students got 80% or above but the difference is small, approximately 2%.

Given these results, an analysis was performed on the types of assignments set, the final examinations and the classroom interactions. One major difference noted was the type of assignments given. The semester with the worst performance, semester I 2009, was the only semester that did not have at least one assignment incorporating the creation of a game. In semester II 2009, students were asked to create a game using an internally created mini-game engine. During that semester, there was an increase in the As and Bs when compared to semester I 2009 and a corresponding decline in the number of Cs, Ds and Fs as shown in Figure 1.

Further viewing of the data in Figure 2, shows a large spike in the 80-89% range for semester II 2008. During this semester, the same gaming engine used in semester II 2009, formed a larger percentage of the programming assignments given. This provided a further indication that the use of gaming maybe more suitable to the learning styles of the students.

Additional analysis of the assignments showed that students were weakest in the application side of computer programming, that is, the actual creation of computer programs. The root cause of this issue was a result of gaps in the prerequisite knowledge needed for the course. Again, the gaming approach seemed to provide a good mechanism for introducing the students to the new concepts and at the same time helping them to revise the prerequisite concepts. Finally, informal discussions revealed that students entering the course had developed a dislike for programming in general and lacked the motivation to really apply themselves to this area of computer science. Gaming seemed to provide a needed motivational bump.

The evidence indicates that a change in the instructional method may produce improved results. The success of the gaming activities in semester I, 2008, shows that such an assessment methodology has significant potential since students are afforded the opportunity to apply their knowledge and hence develop their higher order cognitive skills. With this in mind, it was decided to undergo a preliminary evaluation of the use of gaming as an active learning approach.

The gaming system

To implement the new gaming methodology, appropriate software tools had to be developed. As discussed in previously, while there are a large variety of programming tools for creating games, an entire course is generally required before competency with the tools is achieved. However, while it is possible to teach the OOP concepts at the same time, it was felt that mixing OOP concepts with specific gaming system concepts would prevent the maximum teaching and learning benefits from being achieved. Also, simplified gaming systems created for teaching level I or introductory programming, hide the low level programming details required for an advanced course such as this. This approach results in transition difficulties for students moving from the simplified systems to the more complex industry ones. Another major problem preventing the use of these systems was that the hardware

requirements exceeded those of the computers used by the students, both in the campus labs and their own personal computers.

As a result, a new gaming engine called SIMOO (**Sim**ulator for Teaching **O**bject-**O**riented Programming) was created. The mini-gaming engine used in 2008 and 2009 was not utilized since its primary purpose was to be a single assignment application and so could not be deeply incorporated into the teaching of the course as a whole. Another benefit of creating a new system was that it was developed in-house, demonstrating to students that such tools can be developed locally and within the Caribbean region, providing another motivating factor.

Architecture

The SIMOO gaming system was created using the Java programming language, which is the same programming language taught in the OOP course. This makes it easier for students to use the system as well as ensuring that they employ the concepts taught in the course. SIMOO consists of a set of core Java classes that provide the basic functionality needed for creating simple two dimensional (2D) games. Figure 3 shows the core layers that constitute a typical SIMOO application.

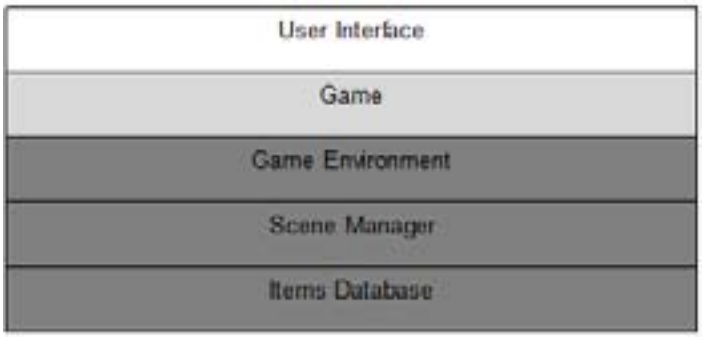

Figure 3. Core layers of a SIMOO application.

The first layer or user interface is what the student sees and interacts with. The second layer is created by the student and provides the actual game dynamics or rules. This layer utilizes the core features of SIMOO found in the lower three layers. The lower three layers provide the base SIMOO facilities for creating a 2D game.

SIMOO is a *scene* (or frame) based system. A scene is considered to be a snapshot of what the user sees at any point in time. Each time the user commits an action, a new scene is drawn (via the Scene Manager layer). The time between scenes is called the *transition* period and it is during this time that the objects of the game (Items Database layer) determine what their status will be in the next scene. For example, an object may decide to move to a new position or change its colour. Transitions between scenes can also be continuous. This is necessary for

games where several activities are occurring at once. It is the continual playback of successive frames or scenes that give the appearance of motion within a game.

Figure 4 shows an actual game created using the SIMOO environment. The large empty space on the right is the drawing canvas. It is here that students draw the graphics for the game. For example, in Figure 4, the grid with the smiley characters has been drawn on the drawing canvas. The buttons on the left side control the game action. SIMOO can run a game a single frame at a time (the Next button) or continuously (the Start and Stop buttons). This helps students perform debugging (locating errors) on their software. The Reset button returns the game to its original starting state. The bottom row of buttons contains basic functionality such as an Exit button. The Up, Down, Left and Right buttons are internally represented as generic buttons, Button1, Button2 etc. It is up to the programmer to decide what these buttons will do. For the assignments, the behavior of these buttons (or what they represent) was predetermined and fixed so as to maintain consistency in the marking scheme.

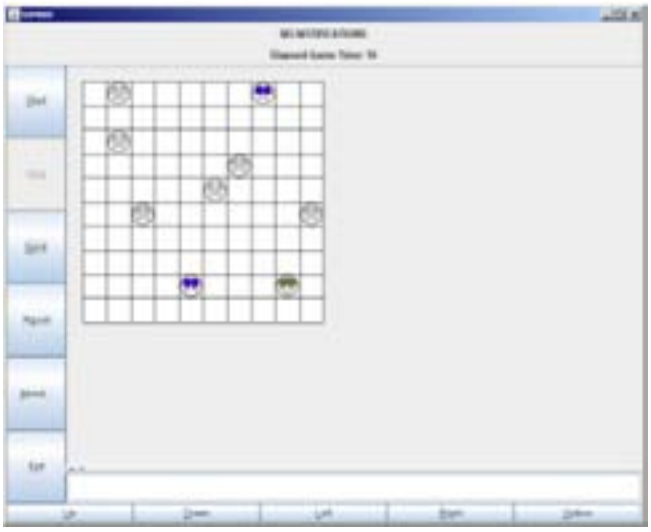

Figure 4. A sample game created using SIMOO.

Teaching approach

In the years 2008 to 2009, the teaching of the programming theory was augmented with live demonstrations of running code. For each concept taught, actual programming code was demonstrated using a laptop and projector combination. Attendance at these classes was generally high and the class discussions were quite engaging. However, it was noted that the students often did not take notes during the demonstrations. Instead they relied on verbal communication and memory. Therefore, in spite of the large attendance, the previously stated decline was witnessed.

In keeping with the change in the assignment strategy, the teaching approach was also modified. For each new concept taught, a significant code example was discussed with the use of the whiteboard. This was then followed by code implementation sessions, where students were given a programming problem to solve in class. Their solutions were then written on the whiteboard for further discussion. The concepts tackled corresponded to those in the assignments. This allowed students to complete their assignments in stages over the scheduled time period.

The use of the gaming system facilitated this approach by allowing the student to build the game in stages, with each stage corresponding to one or more object-oriented concepts. It should be noted that based on the reaction and performance of the students, the one-to-one correspondence could not always be maintained. However, any concepts needed for the gaming assignments were always covered at least one week before the deadline.

Documentation

All documentation for the SIMOO system was included on the campus's e-learning system which utilizes the Moodle e-learning platform (Moodle.org). The documentation consists of two parts, a comprehensive tutorial and Java class documentation. The tutorial takes the student through the steps needed to create a simple game using SIMOO. The Java code generated during the tutorial can be used (and is highly recommended that students do so) for any assignments that utilize SIMOO.

The Java class documentation describes the actual Java code that forms the SIMOO system. The documentation is in the form of web pages created by the standard Java documentation tool called Javadoc (Javadoc Tool Home Page, 2010). As discussed before, this approach is the same used to document the libraries provided by the JDK. To successfully use the JDK, a programmer must become familiar with this format. As a result, this documentation format was chosen for the SIMOO system so that students become comfortable with it. Figure 5 shows a sample SIMOO class documentation page.

Figure 5. Sample SIMOO class documentation page.

Methodology

The SIMOO gaming approach was tested in semesters I and II for the academic year 2010-2011. At the start of each semester, a survey was given to the class to determine their attitudes towards programming as well as their current motivational levels. The survey was also used to determine whether or not students saw gaming as a suitable approach for developing their computer programming skills.

In both semesters, three major assignments were given. The first assignment focused on core object-oriented programming areas and did not utilize the SIMOO environment. The second and third assignments used the SIMOO environment and focused on advanced areas such as inheritance and frameworks. In these assignments, students were required to modify and extend the shell of a previously created game. To prevent copying across the semesters, the underlying details of the game rules were modified between semesters.

Survey instrument

The survey instrument consisted of three parts. The first part captured demographics such as age and gender as well as prior education, current level, full-time/part-time status and currently registered major. The second part consisted of five questions and looked at what currently motivated students and what could potentially motivate them. The third part consisted of twelve questions and dealt with students' attitudes towards computer programming in general.

SIMOO assignments

The SIMOO assignments were based on a paddle, balls and blocks game, as shown in Figure 6. A completed game consisted of a set of balls that moved around the grid area. Each time a ball collided with a block, its direction changed. Depending on the colour of the ball and the block, a number of actions were taken after the collision. For example, if a green ball collided with a red block, the overall score would decrease. The paddle could be moved by the Up, Down, Left and Right buttons. The paddle was used to change the direction of a ball. During the game, the user can add and remove blocks with the computer mouse.

The initial shell of the game consisted of the paddle and a single ball only. The shell actually formed part of the tutorial, which all students were required to complete. The other balls, blocks and game rules were then added by the student across the two assignments. The programming mechanisms used to create the full game required the use of the programming concepts taught during the class.

The documentation for the tutorial and the assignments matched each objective or task to the specific concept taught. For example, the addition of more balls to the game required the use of the concept of Inheritance.

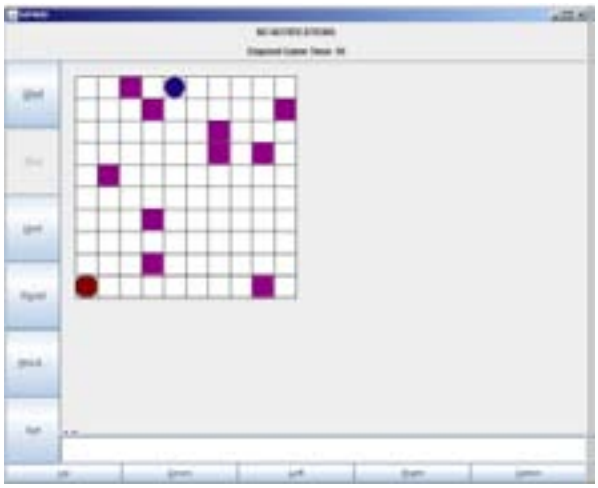

Figure 6. Paddle, balls and blocks game

Final Exams

The structure of the final exam has essentially remained the same since 2007. The exam paper has two sections, where the first section is compulsory and the second section provides students with options where they either select two out of three questions or one out of two. The first section focuses on programming where students are required to write a major object-oriented program. This section is worth 50% of the final exam mark. The second section places the emphasis on the theory and its application. This structure was maintained for the 2010-2011

academic year. This approach maintained consistency across the years and retained the recommended improvements suggested by the external examiners over the years.

Results

Attitudes Survey

A pre-test of the survey was performed using students in a level III course. Fourteen students completed the survey with no problems arising. The number of students taking the Object-Oriented Programming course in semesters I and II 2010-2011, was 42 in each semester. In semester I, 26 students completed the survey and in semester II, it was 24. In each semester, three questionnaires were completed incorrectly and so were discarded; leaving 23 (54.8%) and 21 (50%) completed questionnaires in semesters I and II respectively.

Demographics

Tables 2, 3 and 4 show the percentage spread for sex, age and level of study, for both semesters.

Table 2. Sex.

Sex	Percentages for Semester I	Percentages for Semester II
Male	78.3	71.4
Female	21.7	28.6

Table 3. Age ranges.

Age Ranges	Percentages for Semester I	Percentages for Semester II
16-19	17.4	4.8
20-25	60.9	90.5
26-29	8.7	0
30-39	8.7	4.8
40-49	4.3	0
50-59	0	0
60-69	0	0
70+	0	0

Table 4. Level of study.

Level	Percentages for Semester I	Percentages for Semester II
Preliminary Year	0	0
First Year	0	4.8
Second Year	82.6	81
Third Year	17.4	14.2

Overall, the results in Tables 2 and 3 match the departmental norms where the majority of students are in the age range 16-25 and the males outnumber the females. In Table 4, the large percentage of level II (second year) and level III students (third year) is expected since the course in question is a level II course.

Motivation

To determine what motivated students during the completion of a programming assignment, students were asked to select all that applied, from the list of options given in table 5. Students were also allowed to add any area that did not appear in the list. None of the students added any new items. As expected, getting good grades scored the highest. Meeting course and/or coursework requirements scored the second highest in semester I. In the case of semester II, interest level of the assignment formed the second highest percentage. However, the number of students that stated that gaming is a motivational factor was high, especially in semester II.

Table 5. Motivational factors that influence the completion of a programming assignment

Motivational Factor	Percentages for Semester I	Percentages for Semester II
Getting good grades.	91.3	85.7
It is part of a required course.	52.2	33.3
It is part of the final coursework mark.	60.9	42.9
The assignment is an interesting one	43.5	71.4
The assignment is based on a game.	34.8	47.6
The programming language used.	8.7	28.6
My future career choice.	30.4	47.6
The assignment is a challenging one.	39.1	47.6

When asked to rate the level of difficulty of computer programming, less than 10% of the students found programming easy or very easy (see Table 6).

Table 6. Level of difficulty.

Difficulty level	Percentages for Semester I	Percentages for Semester II
Very Difficult	8.7	4.8
Difficult	47.8	31.1
Neutral	39.1	47.6
Easy	4.3	4.8
Very Easy	0	4.8

Attitudes

Students were provided with a number of questions that determined their attitudes towards computer programming and included what they perceived their level of

enjoyment and self-confidence to be. The questions also asked students whether or not they believed that specific actions, if taken, could improve their programming skills. Each question required students to provide a rating of Strongly Disagree, Disagree, Neutral, Agree or Strongly Agree. Generally there were no surprises but three of the questions did produce some interesting results. The attitudes and corresponding charts are given below.

1. I believe I can solve any computer programming problem (see Figure 7).
2. I would like to see more gaming approaches used in my computer programming courses (see Figure 8).
3. My Level I computer programming experience influences how I feel about computer programming today (see Figure 9).

Only 21.7% and 19% of students, in semesters I and II respectively, believed that they could solve any computer programming problem that they might encounter. Interestingly, as Figure 8 shows, most students would like to see more gaming in their programming, with 65.2% and 71.4% agreeing or strongly agreeing with the assertion. Figure 9 shows that most students agreed that their experience, at level I, influenced how they felt about programming, with 78.3% and 66.6% agreeing or strong agreeing.

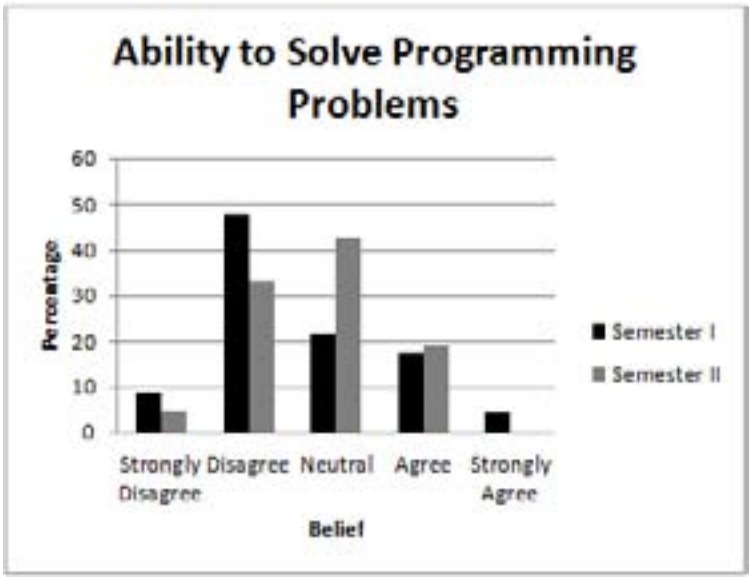

Figure 7. Ability to solve computer programming problems.

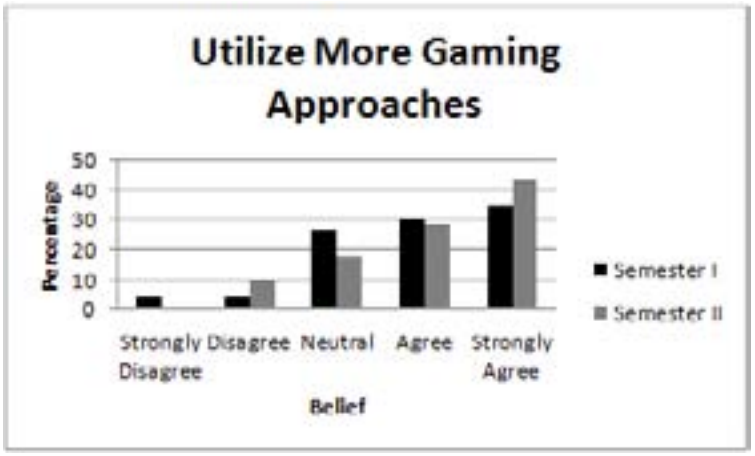

Figure 8. Utilize more gaming approaches.

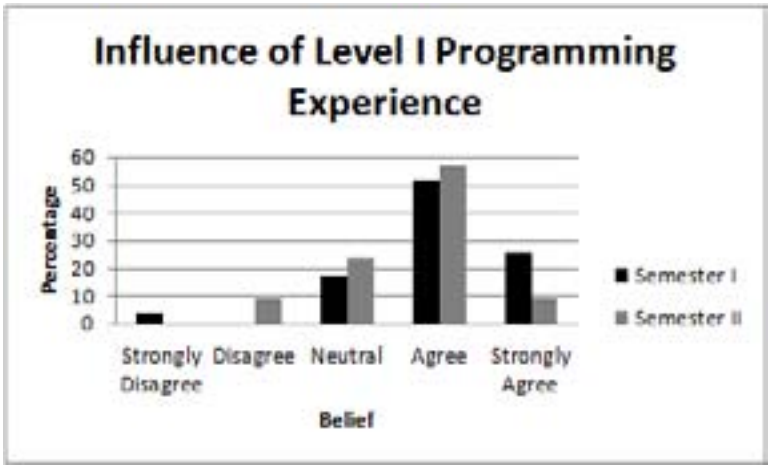

Figure 9. Influence of level I computer programming experience.

Coursework and Final Exam Results

Figures 10, 11 and 12 show the coursework, final exam and overall course grades obtained in the academic year 2010-2011, compared to previous years. As Figure 10 shows, in terms of the number of students obtaining an A grade, they have returned to levels comparable to 2007. Similar results were obtained for the final exam, where the number of students that achieved grade A, exceeded every year but 2008 semester II. Finally, Figure 12 shows that the overall results have exceeded all years including 2007 when looking at those students that obtained grade A. In all cases, similar results have been achieved for grade B. In terms of grades D and F, there has been the corresponding reduction in numbers, with 2010-2011 experiencing some of the lowest levels. It should be noted that the number of students that failed the coursework and the final exam was at its lowest during the period 2010-2011.

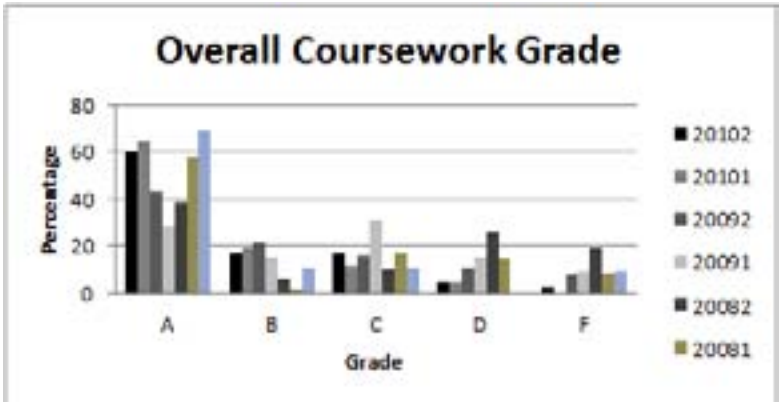

Figure 10. Coursework grades.

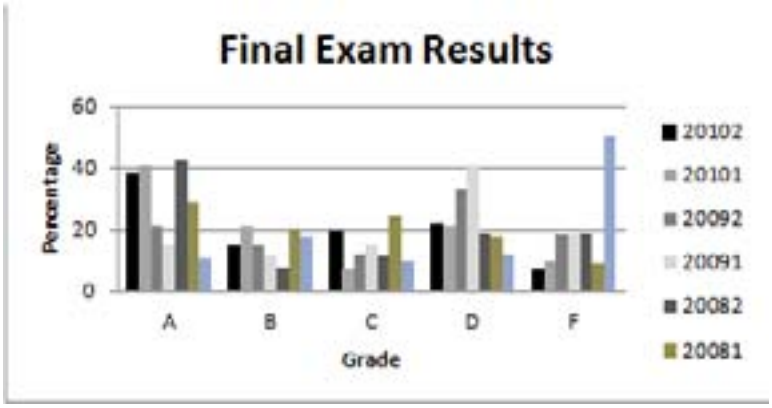

Figure 11. Final exam results.

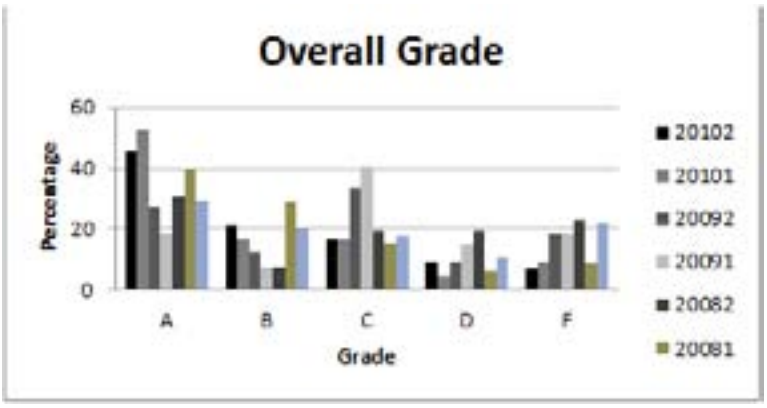

Figure 12. Overall grades.

Discussion

The overall results for the academic year 2010-2011 exceeded all prior years with significant improvements in the number of grades A and B. Corresponding reductions in grades D and F were also achieved. However, it should be noted that this is a preliminary study and is limited by the lack of statistical approaches to determine the statistical significance of the results. Also, no formal control mechanisms were implemented to ensure the statistical reliability and validity of the results.

In spite of its preliminary nature, some lessons can be gleaned from the exercise. Firstly, gaming can improve student motivation in advanced programming courses. During this exercise, students generally made a determined effort to submit completed assignments. It should be noted that two of the students who had each repeated the course at least four times, finally passed the course, obtaining a B and a C grade. The common fear of students copying coursework and then not being able to perform during the final exam did not materialize.

The improved results in the final exam, especially in the low number of fails, reveal that students were able to perform well under exam conditions, demonstrating that the material learned during the course was successfully applied. The improved results also demonstrate that students were able to overcome any lack of prerequisite knowledge as well as any self-confidence issues. However, this does not consider the effect that the must-pass coursework and final exam rule may have had on the outcome. There is the possibility that the full effect of this rule was only now being felt since most of the students in the course would have entered the university after the rule was implemented.

Another important lesson learnt was the need for the lecturer to fully develop solutions for any assignments based on the SIMOO environment. This helped identify any problems with the SIMOO system itself before students used it. This also removed the need to issue any software fixes, for the SIMOO system, during the semester. This is important as software fixes can change the behavior of the gaming system, forcing students to redo already completed work.

As stated earlier, the gaming engine used is a custom built one, developed specifically for teaching object-oriented programming. One of the primary benefits of custom built software is the removal of software licensing costs and the ability to run the software on pre-existing hardware. However, there is the downside of requiring someone to build and maintain the engine. If the software development skill set is not readily available, one can incur significant costs (both in time and money) hiring someone to build the engine.

Given the demonstrated promise of this approach, future work will involve the development and execution of a more formal study with proper statistical controls and methods applied. This would include determining if it was the use of gaming that resulted in the improvements or the adoption of active learning methods where gaming is one of many methods that can be incorporated into the teaching process.

References

Alice.org. (n.d.). Retrieved September 19, 2011 from http://www.alice.org/.

Carbonaro, M., Szafron, D., Cutumisu, M. & Schaeffer, J. (2010). Computer-game construction: A gender-neutral attractor to Computing Science. *Computers & Education, 55,* 1098–1111.

Chang, K.-E., Wu, L.-J., Weng, S.-E. & Sung, Y.-T. (2011). Embedding game-based problem-solving phase into problem-posing system for mathematics learning, *Computers & Education* (October 2011), doi: 10.1016/j.compedu.2011.10.002.

Cooper, S. (2010). The design of Alice. *ACM Transactions on Computing Education.* 10(4), Article 15 (16 pages).

Dagdilelis, V., Evangelidis, G., Satratzemi, M., Efopoulos, V. & Zagouras, C. (2003). DELYS: A novel microworld-based educational software for teaching computer science subjects. *Computers & Education, 40*(4), 307-325.

Ebner, M. & Holzinger, A. (2007). Successful implementation of user-centered game based learning in higher education: An example from civil engineering. *Computers & Education, 49*, 873–890.

El-Sheikh, E. & Prayaga, L. (2011). Development and use of AI and game applications in undergraduate computer science courses. *Journal of Computing Sciences in Colleges, 27*(2), 114-122.

Gestwicki, P., & Sun, F. (2008). Teaching design patterns through computer game development. *ACM Journal on Educational Resources in Computing, 8*(1), 21 pages.

Hainey, T., Connolly, T. M., Stansfield, M. & Boyle E. A. (2011). Evaluation of a game to teach requirements collection and analysis in software engineering at tertiary education level. *Computers & Education, 56*, 21–35.

Huang, W., Huang, W. & Tschopp, J. (2010). Sustaining iterative game playing processes in DGBL: The relationship between motivational processing and outcome processing. *Computers & Education, 55*, 789-797.

Javadoc Tool Home Page (2010). Retrieved September 19, 2011 from

http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html.

Kahn, K. (2001). Generalizing by removing detail: How any program can be created by working with examples. In H. Lieberman (Ed) , *Your Wish Is My Command Programming By Example*. San Francisco: Morgan Kaufmann.

Kebritchi, M., & Hirumi, A. (2008). Examining the pedagogical foundations of modern educational computer games. *Computers & Education, 51*, 1729-1743.

Kelleher, C. & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys, 37*(2), 83-137.

Kolling, M. (2010). The Greenfoot programming environment. ACM Transactions on *Computing Education, 10*(4), Article 14 (21 pages).

Maloney, J., Resnick, M., Rusk, N., Silverman, B. & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computer Education, 10*(4), Article 16 15 pages.

Moodle.org (n.d.). Retrieved September 19, 2011 from http://moodle.org/.

Muratet, M., Torguet, P., Jessel, J. & Viallet, F. (2009). Towards a serious game to help students learn computer programming. *International Journal of Computer Games Technology*, Article ID 470590, 12 pages doi:10.1155/2009/470590

Oracle Technology Network for Java Developers (n.d.). Retrieved September 19, 2011 from http://www.oracle.com/technetwork/java/index.html.

Paraskeva, F., Mysirlaki, S., & Papagianni, A. (2010). Multiplayer online games as educational tools: Facing new challenges in learning. *Computers & Education, 54,* 498–505.

Phelps, A. M., Egert, C. A. & Bayliss, J. D. (2009). Games in the classroom: using games as a motivator for studying computing: part 1. *IEEE Multimedia,* April-June, 5-8.

Sánchez, J. & Olivares, R. (2011). Problem solving and collaboration using mobile serious games, *Computers & Education, 57*(3), 1943-1952.

Slator B. M., Hill, C., & Del Val, D. (2004). Teaching computer science with virtual worlds. *IEEE Transactions on Education, 47*(2), 269-275.

Vos, N., van der Meijden , H. & Denessen, E. (2011). Effects of constructing versus playing an educational game on student motivation and deep learning strategy use. *Computers & Education, 56*, 127–137.

Wang, A. I. (2010). Extensive evaluation of using a game project in a software architecture course. *ACM Transactions on Computer Education, 11*(1), 28 pages.

Wang, A. I. & Wu, B. (2011). Using game development to teach software architecture. *International Journal of Computer Games Technology*, Article ID 920873, 12 pages doi:10.1155/2011/920873.