# Games as a "Flavor" of CS1

Jessica D. Bayliss
Rochester Institute of Technology
Rochester, NY 14623
585-475-2507
jdb[at]cs[dot]rit[dot]edu

Sean Strout
Rochester Institute of Technology
Rochester, NY 14623
585-4754599
sps[at]cs[dot]rit[dot]edu

## ABSTRACT
Introductory programming courses have a specific set of expected outcomes. One of the most often heard complaints in such courses is that they are divorced from the reality of application. Unfortunately, it is difficult to find areas for application that all students have the background knowledge for and that are engaging and challenging. One such area is computer games and we have developed a cohesive CS1 course that provides traditional outcomes from within the context of games as an application area in both the lecture and lab components of the course. This course was piloted as a ten-week distance program for incoming computer science students with the defining features that the program carried no academic credit and offered no end grades. We discuss the overwhelming interest in this course as well as objective and su$^{o}$bjective student experiences. One of the most important outcomes of the summer course was that it brought students with similar interests and goals together. We discuss this and the different ways we have found to discuss computer science course topics from within a games context.

## Categories and Subject Descriptors
K.3.2 [**Computer and Education**]: Computer and Information Science Education – *computer science education, curriculum.*

## General Terms
Algorithms, Design, Languages

## Keywords
Games, CS1, Video Games

## 1. INTRODUCTION
Introductory programming sequences teach the core concepts of the computer science discipline: object-oriented programming, software engineering, data structures, and algorithms. Much like musicians, who take music theory in order to understand and compose written music, but must also take instrument lessons in order to practice the skill of playing a musical instrument, computer scientists must learn the theory of their field, but must also practice the skill of programming. This is apparent in many courses where there are separate lecture and lab course components. Unlike the field of music, computer scientists ordinarily need to apply the theory and logic of computing to one or more application areas. These areas can range from biology to art to operating systems, but all have one thing in common: students need a certain amount of background in the application field in order understand why they are creating the program. Giving students this background takes time, unless the background is very simple or students have previous knowledge of the area.

As an example where background knowledge can help students understand, consider the many object-oriented employee payroll type of examples in CS1 Java books. Do these examples mean anything to students who have never worked in an office? Probably not. Unlike the employee example, all of our students have played some kind of game and they often have very strong opinions about games. Games cut across cultural, racial, and gender boundaries.

Games may be entertaining like an arcade game or serious like a biological simulation. All students at one point have played a game. The important concepts of win conditions and scoring systems are things that students are familiar with and thus little background is needed when using simple games in a CS1 course.

Several examples of courses with games in their CS1 content exist and academics are considering the use of games more often in their courses [5]. Some of the courses are not truly games-related, but are really visually oriented programming environments that may be used for games [1, 2, 3, 4]. Some of these courses explicitly teach game development and may not meet traditional CS1 outcomes [6, 7]. Some courses use games for homework assignments and projects. In order to avoid the common split between the content of lectures and labs, we chose to integrate game material into both our labs and the lectures while still meeting traditional CS1 outcomes using Java as an introductory programming language [13].


*SIGCSE'06,* March 1–5, 2006, Houston, Texas, USA.

## 2. THE RAPT SUMMER COURSE

The CS1 course is part of a 3 course programming sequence titled the Reality and Programming Together (RAPT) program. The sequence teaches basic programming, algorithms, and data structures with games as an application area.

## 2.1 RAPT CS1 – THE COURSE

Due to the pilot nature of the CS1 course, it was taught as a 10-week summer distance course using Java. The course's weekly structure consisted of 2 hours of on-line lecture coupled with 2 hours of required lab/homework that were started synchronously at a specific time. Students were expected to do much of the labs and to read the book off-line. Teaching assistants were available to help students with problems on-line. Three on-line quizzes and a final were given to provide feedback to students and the instructor. The final was then followed up by a placement exam upon student arrival in the fall. The placement exam was used as a pass/fail mechanism to allow students to progress into CS2. In order to keep the course linked with real industry practices, two game developers presented their work in on-line chat sessions over the summer.

All incoming CS students who wanted to participate had to fill out an application containing an essay of why they wanted to participate as well as questions about prior programming experiences. Applicants were chosen by considering beginning programmers with good essays before those with more experience. Roughly half of the students accepted were classified as beginning programmers. So that all of our entering freshmen could participate, the program was free during the summer, but carried no course credit. The only material benefits to students for participating in this program were that students could gain the knowledge to take the placement exam upon entering our university in the fall. This exam could place them out of CS1 and into the RAPT CS2 course. Placement into CS2 frees up one course slot in our degree program and allows students to take an extra CS elective to fill that slot. All students accepted into the program were slated to take CS1 in the fall. The course ran from mid-June through mid-August.

## 2.2 RAPT CS1 – COURSE RESULTS

Games are very appealing to incoming students. Incoming freshmen offered to take time out of their busy summer schedules ranging from full time employment to family trips in order to take an introductory programming course with a teacher they knew nothing about.

We received 81 applications (40% of our entering class) and were able to accept 48 into the program. Of these individuals, two were female and five (non-overlapping) individuals were ethnic minorities (out of a possible 25). The percentage of female applicants is not really as small as it appears since there was an entire pool of 9 females who received the RAPT program application. These students generally entered the program for one of two reasons:

1. They liked games and wanted to learn how to program them.
2. They wanted to get ahead in programming and take CS2 upon arrival at our university.

The distance format introduced difficulties to the course. Of the initial number, 4 never really attended the course, 3 withdrew because of work schedules that conflicted with the course, 1 withdrew due to hospitalization, and 3 withdrew after going on vacation and missing a few weeks of the course. Thus, by the end of the course there were 37 people who qualified to take the placement exam.

Success of the program was measured in two ways. First, scores on the placement exam objectively measured student performance. Additionally, students' opinions were measured over the summer with two surveys.

The placement exam contained a representative sample of CS1 materials and could be considered equivalent to a CS1 final, although it also contained CS2 and CS3 materials as well. The CS1 materials had a separate grade from the rest of the exam. The placement exam was written by a professor not associated with the RAPT program and tested CS1 and CS2 materials. It was used to place students into our introductory sequence at the CS1, CS2 or Advanced Placement level.

Of the 37 students, 3 failed the placement exam and were not placed into CS2. If we consider only those students who attended the course and took the placement exam, the overall failure rate was approximately 8%. This is on the low range for our normal CS1 courses (usually around 10%), although it is not significantly different.

Of the students who passed the exam, two have left the program. One has chosen to advance place out of CS2 as he passed the exam at a level above CS2 and one has decided not to continue with the program as his goal was to place into a regular CS2 course. This particular student stated that he is not particularly fond of programming games. Three other students could have placed into the advanced placement course and out of RAPT CS2, but have chosen to stay in the program as they say that it matches their interests more than the advanced placement course.

We tracked the subjective experiences of students with two surveys taken at different times in the course. The surveys asked students questions about distance learning, continuation of the RAPT program, their comfort levels with a variety of programming topics, their intimidation levels, and we asked some free form questions about their likes and dislikes about the course.

We asked about intimidation that students felt from other students. In a previous internal study of CS1 students, we found that a significant portion were intimidated by other students who "knew so much more", and that the intimidated students were more likely to switch majors. This is similar to the results found by Bergin and Reilly who found a strong correlation between student comfort levels with programming modules and their actual performance [8]. Thus, a measure of the intimidation students

were feeling was very important. We used a Likert scale to measure intimidation with 1 being always intimidated, 2 frequently, 3 sometimes, 4 very little, and 5 never intimidated. The results between the original survey and the RAPT survey are shown in Table 1. Only the top 3 intimidation levels are shown in order to highlight the difference between the RAPT students and our normal CS1 students. These differences could be due to the distance nature of the course or to the fact that even when the students did not know about programming, they were on equal footing with their classmates in knowledge of popular games. Of note is the fact that the one person (1 person on both surveys was frequently intimidated) who feels intimidated is not one of the poor performers in the class and passed the placement exam with no problems.

**Table 1: Student perception of intimidation due to their peers. While the numbers are significantly smaller due to the size of the RAPT course, intimidation percentages look different from a normal CS1 course.**

| | Always | Frequently | Sometimes |
|---|---|---|---|
| CS1 (n=369) | 7% | 15% | 23% |
| RAPT Survey Week 4 (n=33) | 0% | 3% | 30% |
| RAPT Survey Week 10 (n=21) | 0% | 5% | 19% |

Of all students, approximately half agreed that the distance nature of the course hurt their learning from the course, but 100% on the last survey felt that the course should be continued with 66% preferring the distance nature of the course to 34% who preferred that the course be taught in the fall as a regular CS1 course. The main reason for this preference appears to be the ability to get to know other students with similar interests before coming to college.

Over the summer, these students developed a cohesive bond that has passed into the fall. Several students commented on their disappointment at not being placed with a RAPT student as a roommate in the dorms. In a free form question asking what people liked about the course, approximately 30% cited meeting other people as their top reason for liking the course. There is a large body of research supporting the idea students in a cohesive social group are more likely to succeed. The implication for using games as an application area in CS1 is that the course may attract those with similar interests outside of Computer Science. This in turn can foster greater success.

Another 30% cited the games nature of the course as the part of the program they liked the most. The students on the whole were a very motivated group and commonly did more than they were asked to on any given assignment. This ranged from creating extra character classes for a combat simulation assignment to a student who implemented the A* algorithm for character movement in a 2D collector game called PondMaze.

These results indicate the success of the program, although they also suggest that providing a summer distance course to incoming freshmen can be problematic. It is possible for a program such as this to be used as an attractor for students who may be bored over the summer and "looking for something to do". The results indicate that not all of our introductory courses should be taught with games as an application area – some people do not like games. We are a large enough school to offer this as a specialty section of CS1, but smaller institutions (with only one small CS1 section) may have problems teaching CS1 courses in this manner.

# 3. SUGGESTIONS FOR COMBINING GAMES WITH CS1

It is common to see many game-type projects/labs in a CS1-type course. It is perhaps not quite as easy to link lecture materials to the discussion of how games are developed. We would like to offer some ideas for things that can be done within the context of CS1 material. The visual nature of games is convenient for instructors, as it is fairly easy to see how a particular syntactical construct may be used in a game and logical program errors may show up as amusing visual errors. As a simple example, students were very amused when their grid-based game characters started randomly jumping through walls due to incorrect programming of the character movement routine.

Below we outline some of the basic topic areas that are covered in a CS1 course and examples that we have or would like to use. We realize that some of them may be straight forward, but expect that some of them may be surprising to educators not involved with the game development industry.

## 3.1 SOFTWARE ENGINEERING

The top games are usually multi-million dollar software development projects with all the joys and pains of any other large software project. On top of this, game developers often operate under very harsh ship deadlines since most games must be released before Christmas in order to generate appropriate revenues. Game developers may go into "crunch time" shortly before the game is to be released and sometimes use agile development techniques such as pair programming in order to limit the number of bugs introduced into the end product during crunch time. At a recent Game Developer's Conference, Agile development was the main topic for a developer's roundtable discussion [9]. Postmortems for games are commonly published in trade magazines for developers. In addition, modern games focus much of their development efforts on extensive tool sets that support the creation of content for the game including music, art, and level design. Within a course doing a simple game such as Tetris, this takes the form of programming a data driven design where rather than students programming the movements of individual blocks, they instead read in a configuration file containing block layouts and rotation points. The blocks are then properly displayed and rotated by generic methods. In looking at a chronology of computer game development, further insights may be obtained about how software engineering has developed in the gaming industry from early assembly games through the development of the modern game engine and the use of data driven design [14].

## 3.2 ETHICS

Ethical issues are involved when certain game companies force their employees to remain in crunch time (sometimes working as many as 80 hours a week) when not close to ship dates. This has led the International Game Developers Association (IGDA) to issue a white paper that calls for a greater quality of life for game developers [10]. The IGDA has white papers on a variety of legal issues involving games [11] including the open question of whether or not massively multiplayer games are addictive in a similar manner to gambling and what developers can do for those who play games too often. Other issues are involved when individuals hack multiplayer games in order to gain unfair advantages over opponents and occasionally source code is stolen and distributed on-line as was done with the Half Life II engine.

## 3.3 PROGRAMMING

One of the primary focuses in a CS1 course is to teach programming. At a higher level, we would like students to learn how to create/use algorithms and then implement them using the syntax of a particular programming language. Here we present a sampling of ideas from games that can be used within the context of a CS1 course.

1. Expressions: Role-playing games and collectible card games often involve complicated combat systems. They range from simply rolling one 10-sided die to determine damage to a player character to full blown combat systems that take character level, strength, agility, etc. into account. One of our favorite examples comes from a commercial role-playing game where low level (numerically) characters become more and more likely to miss hitting higher level characters as the level difference increases. This game has the concept of specialty skills for each character that can increase the chance any character has to hit. One can think of a couple of scenarios that would take this into account:
    a. (ChanceToHit*levelRatio)+skillIncrease
    b. (ChanceToHit+SkillIncrease)*levelRatio

    The game in question uses a. and this means that low level characters that choose to train in a specialty skill have a much higher chance to hit higher level characters than many players think they should. Expressions were practiced in the course through a programming a portion of a combat simulator called PondCombat. Additionally, since students copied class templates in order to fill in expression details, this lab could later be used as a refactoring assignment in order to make the code use inheritance properly.
2. Modulus: One can use the modulus operator wherever there is a repeated sequence that should repeat every *n* game clock ticks. Our favorite example has to do with a line of identical horses that all swish their tails in a metronome fashion at the same time from the commercial game Rome Total War. Modulus can be used to individualize when the horses swish their tails by giving each horse a unique id that can be used to start a tail swish at a multiple of that unique id in game time. Prime numbers are better if horses should swish their tails fairly independently, since the tails will align at the least common multiples of two different horse id's.
3. Conditionals: Game logic in many games consists of hierarchies of finite state machines. This is because state machines are easy to create and easy to test. Interactive fiction is a good example of a game concept that almost entirely relies on large numbers of conditional statements. Students can usually think of many ways the logic of a finite state machine can break. We recorded a sample from the game Morrowind where the player is supposed to escort a husband back to his wife. If the player stabs the wife, the wife will attack the player. The husband (as a friend of the player) will then attack the wife until one of them dies. Strangely enough, we recorded a clip of this and when the two first meet each other, the player completes the mission even though the two them attack each other.
4. Iteration: A game loop keeps the game running in a cycle waiting for user input until the user wants to quit the game.
5. Arrays: To serve as a motivational vehicle for introducing Arrays in Java, the class looked over an essay from John Carmack of ID Software[12]. He wrote an article regarding his experiences with cell phone game programming. In the article he talked about the many hurdles he encountered when dealing with an interpreted language for game development. He talked about how a useful language feature like range checking with arrays actually caused a huge hit on the performance of his code. Most students in the class were already aware of who Carmack was and could immediately associate with what he was talking about. This in turn motivated students when they worked with our 2D grid-based game called PondWars. We also moved from programming with arrays for laying out maps in PondWars to using a data file and a map editor for construction maps. Other than map data, games with multiple players and/or opponents work well for arrays and can motivate the use of other linear data structures.
6. Objects, classes, and inheritance: Game characters are often inherited from an entity class that includes all kinds of character and game world objects. The character class may derive from this class and different types of characters further derive from the character class. It's often convenient to allow students to write their own individual 2D opponent characters within the constraints of a game. We had students rewrite PondCombat in order to use inheritance properly (there had been almost complete copying in the expressions lab).
7. Efficiency: Many games need to distribute the load caused by multiple characters within a small distance from each other that are all trying to move and perform other functions. Given the shear size of many modern games, there are a lot of trade-offs in size and memory and game developers are very careful with how they use data structures. A favorite example of how a built-in

data structure can be misused can be shown for the Java Vector class. The Java implementation for this particular class allows for the resizing of the underlying array structure when the number of data elements exceeds the size that the array can hold. Unfortunately, the automatic size increment is to multiply the current size by 2 and the array starts out to be fairly small such that it must grow quite often for large collections of objects. It is very important for a person using the Vector class to give it a default size and increment in order to avoid having to recreate the array many times as the data collection grows in size.

## 4. DISCUSSION AND FUTURE WORK

This course has shown that a large percentage of students like learning about introductory programming concepts within the context of computer games. While the program started off with one of the hardest course formats to succeed in, almost all of our students who persevered did succeed in the course. Those who did not succeed gained an important benefit: many of them still socially network with students in the RAPT program. We will be tracking the overall success of all individuals from the original group over the course of the first year in order to ascertain if there is a lasting effect of the program on the students who actually participated in the program at some point. At the time of writing this paper, RAPT CS2 has started and is being taught in the studio course model. Students are currently learning the C# language and are discussing the pro's and con's of that language in comparison to Java. We believe that using multiple languages early in CS curriculum will prove beneficial. While CS1 concentrated on small programming projects, CS2 contains a quarter long project that requires individual students to write a client side (ro)bot to play against other bots in the commercial game Unreal Tournament. This project involves writing a message parsing class that supports the client message interace, a multi-threaded TCP client connection class, and artificial intelligence for how the bot plays.

## 5. ACKNOWLEDGMENTS

This program was supported by a Microsoft Computer Gaming Curriculum grant.

## 6. REFERENCES

[1] Bergin, J., Stehlik, M., Roberts, J., and Pattis, R., Karel++: A gentle introduction to the art of object-oriented programming, John Wiley & Sons, 1997.

[2] Clements, D. H. and Meredith, J. S., Research on Logo: Effects and Efficacy, Retrieved September 1, 2005 from http://el.media.mit.edu/logo-foundation/pubs/papers/research_logo.html.

[3] Cooper, S., Dann, W., and Pausch, R., Teaching Objects-first in Introductory Computer Science, SIGCSE Technical Symp. On Computer Science Education, 2003.

[4] Carlisle, M. C., Wilson, T. A., Humphries, J. W., and Hadfield, S. M., RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving, Proc. of the 36th SIGCSE technical symposium on Computer science education (Feb. 2005), 176-180.

[5] Sweedy, E., deLaet, M., Slattery, M. C., and Kuffner, J., Computer game and CS education: why and how, Proc. of the 36th SIGCSE technical symposium on Computer science education (Feb. 2005), 256-257.

[6] Coleman, R., Krembs, M., Labouseur, A., and Weir, J., Game design & programming concentration within the computer science curriculum, Proc. of the 36th SIGCSE technical symposium on Computer science education (Feb. 2005), 545-550.

[7] Parberry, I., Roden, T, and Kazemzadeh, M. B.., Experience with an industry-driven capstone course on game programming, Proc. of the 36th SIGCSE technical symposium on Computer science education (Feb. 2005), 91-95.

[8] Bergin, S. and Reilly, R., Programming factors that influence success, Proc. of the 36th SIGCSE technical symposium on Computer science education (Feb. 2005), 411-415.

[9] Game Developer's Conference Archives. Retrieved September 1, 2005 from http://www.gdconf.com/archives/.

[10] Quality of Life Committee, Quality of Life in the Game Industry: Challenges and Best Practices, Retrieved September 1, 2005 from http://www.igda.org/qol/whitepaper.php

[11] International Game Developers Association web site. http://www.igda.org.

[12] Carmack, J., Cell Phone Adventures, Retrieved September 1, 2005 from http://www.armadilloaerospace.com/n.x/johnc/Recent%20Updates.

[13] Bayliss, J., RAPT CS1 Course Home Page, Retrieved November 1, 2005 from http://www.cs.rit.edu/~cs1.

[14] Dalmau, D. S., Core Techniques and Algorithms in Game Programming, New Riders Publishing, 2004, 1-27.