Chapter XX

Motivating and Retaining CS2 Students with a Competitive Game Programming Project

RYAN GARLICK[1] and ROBERT AKL[2]
[1,2]University of North Texas, Department of Computer Science and Engineering, Denton, TX 76203, USA. E-mail: garlick@unt.edu, rakl@cse.unt.edu

The widespread goals of student retention, introducing larger programming projects, and fostering collaboration among students in computer science courses has led to the inclusion of group projects in many curricula, with task division and collaboration as motivation for students to complete assignments. This paper presents a study in a first-year programming assignment with similar goals, but with methods adopting the contrarian view – having students directly and individually compete with one another in a tournament of their respective software agents. This paper presents the results of a year-long experiment in an intra-class competitive assignment in the second C++ programming course at the University of North Texas in Denton. Metrics of student performance on the assignment, correlation with course grade, student surveys of the project, and retention statistics are presented. Results demonstrating overwhelmingly positive response and high levels of effort among students are submitted, along with remarks on application to student recruiting, retention, and curriculum design.

INTRODUCTION

Nationwide, from 2004-2005, the number of newly enrolled students declaring a major in computer science declined by 21%, from 15,950 to 12,532. This follows a 10% decline in new majors last year and a 23% drop the year before [1]. As the majority of students leaving computer science do so by the end of their freshman year [2], designing assignments for introductory programming courses during this formative year is a critical component of student retention. The introduction of recursion, polymorphism, and other challenging concepts in the second semester of a computer science program has contributed to high attrition rates.

The course in question for this paper is the second C++ programming course (hereafter CS2). This course begins with arrays, pointers, and an introduction to object-oriented programming and concludes with elementary data structures. The design of a large programming project for the course would have to consider that discouraged students have a good chance of leaving the field altogether.

The goals of student retention, introducing larger programming projects, and fostering collaboration among students in computer science courses has led to the inclusion of group projects in many curricula, with collaboration and task division as motivation for students to complete assignments.

This paper presents a study in a first-year programming assignment with similar goals, but with methods adopting the contrarian view – having students directly compete with one another in a tournament of their respective software agents. This study observes the performance, effort, and feedback of a total of 60 students engaged in a novel CS2 programming assignment over the course of two academic semesters (30 students per semester) for evaluation of the goals outlined in the following section.

Later sections describe the assignment designed to address these goals, literature relevant to using game development in introductory CS courses, quantitative results of student effort on the project, results of a survey to measure student response, and evaluation of the degree to which the goals of the assignment were met. Conclusions are presented along with notes on student retention.

GOALS

In formulating a prospective project for the course, the goals of an ideal assignment were considered (in no specific order):

- **An assignment that students enjoy.** This aspect would be crucial for student retention, word of mouth recruiting to the department, and course satisfaction.
- **An assignment that students will gladly put more effort into than is strictly necessary.** As a corollary to their enjoyment, the willingness of students to put significant effort into a project was an important goal. It would be desirable to include techniques applicable to a range of real-world problems.
- **Encouraging exploration of areas of computing outside the scope of the assignment.** A project encompassing a broad range of computer science topics would be ideal, motivating students to explore those aspects they find most interesting. This goal would have to be carefully balanced with the need to not overwhelm students or frustrate them by requiring extensive knowledge outside the scope of the class, while rewarding self-study.
- **A rigorous and beneficial assignment.** Enjoyable assignments can certainly lead to complimentary course evaluations, but are of questionable value if little benefit is bestowed upon students.
- **Fostering collaboration among students.** Companies have reported that students graduating with Computer Science degrees are often under-prepared for group work [3]. Collaborative work in the early years of a computer science program for educational and social development can benefit retention as well [4]. The goal was to create an assignment that students would complete individually, while encouraging an environment of collaboration and idea

exchange. As discussed in the evaluation section, although counter-intuitive, direct competition served to fulfill both of these aims.

- **Innately discouraging code copying.** Academic dishonesty in programming courses is well documented. Surveys have indicated as many as 80% of students cheat during their academic careers [5]. An assignment that intrinsically encourages cooperation yet discourages code sharing would be ideal.
- **Providing experience with larger software projects.** An initial motivation for the assignment was course exit surveys from previous semesters highlighting the lack of experience in larger software projects of the scope and size that one would typically encounter in the workplace. The solution has often been to incorporate a group project in which tasks can be divided among several group members.

In short, an assignment was sought that students would work harder on, learn more with, and tell you they love. A different approach to the traditional group project was undertaken, focusing on incorporating each individual's work in extending a given larger project. This was accomplished in the context of a game engine, with each student creating an autonomous "player agent" incorporated into the engine. The focus of this study was the competitive nature of the assignment, with student's autonomous agents competing against one another, two at a time in a single elimination tournament.

RELEVANT LITERATURE

Incorporating game programming as a method of teaching introductory programming is well established [6, 7]. Tools such as Game Maker [8] have been developed to capitalize on this topic, and several game ideas have appeared in the "nifty assignments" section of the SIGCSE bulletin [9, 10]. Although several studies have focused on cooperation in game development, concentrating strictly on the competitive nature of game assignments was not found in the existing literature.

The inter-disciplinary nature of computer games makes them effective assignments both for introducing direct application to an industry, and presenting a breadth of computer science topics [11]. As mentioned in the previous section, drawing from multiple domains can also influence retention, as a student may discover an area that piques his or her interest.

A survey of the literature also revealed increases in the level of active participation in gaming related assignments. This behavior was duplicated in this study, as detailed later. Gumhold [7] discovered more complex than expected submissions to a game-related assignment and noted that, "students were actually doing more homework just for the fun of it."

ASSIGNMENT DESCRIPTION

The assignment was to create a C++ class that would interface with a game engine created by the instructor. The game was “TankWar,” with each player providing the code to autonomously operate his or her tank on the field of battle. The game engine would pass information on the state of the game to each player in turn (in the form of “sensor” data), and receive the move of each player’s agent based on this state. Each player’s tank, in the course of one move, could fire at the enemy, drop a mine, sweep and remove mines adjacent to the player, move one space, or sit. There were 2 players in each match.

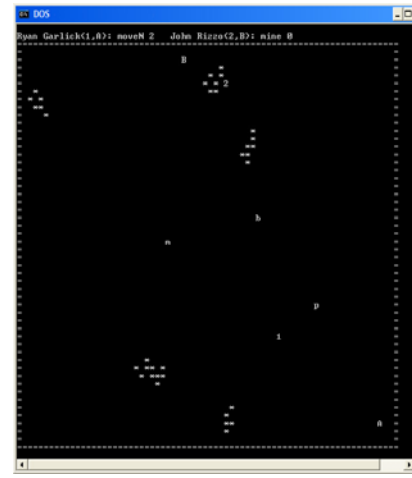

FIGURE 1
TANKWAR SCREENSHOT

Players were awarded points for successfully firing upon their opponent and sitting in the opponent’s base area (pillaging). Any shots fired at a certain game grid coordinate do not land until the opponent has a chance to move. This requires a player to predict where his or her opponent will be during the next turn, and allows for more complex behavior by storing an opponent’s move history and attempting to discern a pattern so as to more accurately aim.

Points are given to the opponent for moving over obstacles scattered randomly throughout the field. Each player has the ability to detect the obstacles so as to avoid the penalty of occupying the same square, or may choose to “take cover” in the obstacle. In this case, the player accepts the small penalty, but significantly reduces their opponent’s chance of successfully landing a shot. The first player to a pre-defined point total wins the match. Tournament results were posted on the web, with daily updates as each match progressed.

The game engine itself was approximately 1250 lines of code, requiring a reasonable effort to understand the structures passed to each player, and providing experience in the interaction between software objects. The game display was not based on calls to a graphics library, but rather simple text output (as shown in Figure I), trading off the flash of a graphical display for promoting familiarity of the code. At the time of the game’s introduction, all of the code concepts had been at least tangentially discussed, allowing students to effectively explore and modify the engine code in its entirety.

The assignment was designed to require the use of static methods, inheritance, stacks, and other topics from the lectures. The code was carefully designed so as not to require knowledge of any C++ language constructs outside the scope of the class, yet suggestions were provided to new areas that students could explore to enhance the performance of their agent.

Aside from this direction, few guidelines were given for expected functionality of the finished product. No minimally acceptable solution was described. Although the grading of assignments was ultimately independent of tournament placement, students were told to craft their solution with the goal of winning the tournament. The competitive nature of the assignment was stressed throughout, with a popular feature of the game being the ability to programmatically taunt one’s opponent at the beginning of the match.

EVALUATIONS

Although evaluations of any classroom assignment are partially anecdotal, metrics of student effort, survey responses, and evaluation of the objectives presented previously served as measures of the degree to which these goals were met and the success of the project as a whole.

The University of North Texas has a sequence of well known and popular upper-division game programming courses. Many students in CS2 are anticipating enrolling in these classes, and students were generally very receptive to game programming assignments in CS2. Response to the project was very positive.

	Code Length
Mean	598
Min	86
Max	1922
St. Dev	432

TABLE 1
STUDENT SUBMISSION METRICS N=60

Although the number of lines of code is a poor estimate of the sophistication, efficiency, or relative performance of a solution (some short but well-performing submissions were received), it is a rough estimate of the amount of effort invested in the assignment. Table 1 presents statistics for the submitted programs in terms of code length. For comparative purposes, a minimally functional agent provided as an example was 51 lines of code and the winning solutions for each semester were 1134 and 1716 lines, respectively.

The largest programs received were more code than the engine itself, and seemed to validate the idea that students would spend more time than was strictly necessary. The average submission was a good effort, often encompassing one or more of the auxiliary techniques described in class.

Survey results were used to measure student opinions and attitudes toward the project. Results were obtained from 60 students participating in the assignment. The survey consisted of six questions, with a possible score in each ranging from one (worst) to ten (best). Questions on the survey were as follows:

Q1. Rate the assignment for improving your understanding of larger software projects.

Q2. Rate the assignment in terms of programming projects you have encountered in other classes.

Q3. Rate your exploration of areas of programming that you were not familiar with in an effort to win the tournament.

Q4. Rate your motivation to work and learn more due to the competitive nature of the assignment.

Q5. Rate your overall enjoyment of the assignment.

Q6. Rate the value of the assignment.

Table 2 presents the results of the non-anonymous written survey conducted after the assignment was due and grades were assigned. The overall value of the assignment was rated a mean value of 8.65 out of 10. Table 2 also correlates survey responses with the

	Q1	Q2	Q3	Q4	Q5	Q6
Mean	8.05	8.95	6.63	7.83	8.63	8.65
St. Dev	1.6	1.9	2.7	2.4	1.7	1.8
r^2	0.04	0.11	0.01	0.06	0.05	0.09
P	0.116	**0.009**	0.811	0.051	0.073	**0.017**

TABLE 2
CORRELATION BETWEEN SURVEY RESPONSE AND COURSE GRADE

overall course grades for each student. Course grades were assigned a numerical equivalent with A=5, B=4, etc. to obtain the correlation figures. P-Values less than 0.05 are highlighted in boldface. Students with a higher course grade generally rated the project higher relative to assignments in other classes, and gave the assignment higher scores on overall value. Correlation between course grades and other survey questions was not established.

Anonymous feedback from the end of semester course evaluations relating to the assignment was positive without exception, with one student remarking, “This was the only programming assignment I have worked on after it was already turned in.”

At the conclusion of the project, goals from section 2 were revisited in evaluation of the assignment.

- **An assignment that students enjoy.** Survey results indicate a median enjoyment rating of 8.63 / 10. During the weeks leading up to the tournament, discussions before class among students were dominated by strategy comparison. Students also preferred the assignment relative to more traditional non-game related programming assignments that do not culminate in a large project.
- **An assignment that students will gladly put more effort into than is strictly necessary.** The survey question regarding motivation based on competition received a mean score of 7.83 / 10. Although effectively provided with an answer to the assignment, submissions from students ranged from 86 to 1922 lines of code. This seemed to validate the initial goal of eliciting high levels of effort from students.
- **Encouraging exploration of areas of computing outside the scope of the assignment.** Despite being the lowest rated survey question, many student submissions included some reasonably sophisticated techniques for attempting to predict the moves of the opponent. After the assignment was due, several students voluntarily modified the game engine to include sound, graphics, and extended features, as detailed in the conclusion.
- **A rigorous and beneficial assignment.** Along with a high median effort on the project, and some truly exceptional submissions, survey results were positive in this regard as well. The value of the assignment was the second highest rated survey question, with an average of 8.65/10.
- **Fostering collaboration among students.** Observations for this goal are strictly anecdotal. Although students in this study and other studies have expressed misgivings about group work [3, 12], students were encouraged to test their agent by challenging classmates before the due date (and many did). Agents were then fine tuned based on mistakes made in these preliminary matches.

- **Innately discouraging code copying.** MOSS (Measure of Software Similarity) is software for detecting plagiarism from a corpus of source files [13]. MOSS found no significant similarities between any programs submitted for this assignment. As a deterrent to copying code, students knew MOSS was in use at the beginning of the course. The nature of the assignment discourages code sharing, as no one wants to reveal the inner workings of their strategy and give their opponents a competitive advantage.
- **Providing experience with larger software projects.** Students learned to read the existing classes and extend them via inheritance. The process of reading existing software is often overlooked as a valuable tool for teaching the effective writing of software [14, 15]. Students rated the assignment 8.05/10 for improving their understanding of larger software projects.

CONCLUSION

This study has presented a non-traditional competitive CS2 programming assignment that was successful in motivating quality work, fostering collaboration, and minimizing code sharing. Based on anecdotal evidence, student performance on the assignment, and survey responses, students in this study expended considerable effort to win the tournament. Although extrapolation to larger groups is difficult, the assignment was successful among the sampled students, some of whom are anticipating a career in game programming. Demonstrations of the game have also been used as a visual aid at recruiting events for high school students.

Following the initial year-long study presented in this paper, the assignment was ported to Java and used in a subsequent CS2 class. Confirming the initial goals of eliciting more work than necessary and encouraging study of related areas, two students (of their own volition) created a graphical version of the game, shown in Figure 2. Student modifications are selectively incorporated into new versions, obsolescing the previous agents and preventing code re-use from semester to semester.

For this class, the tournament winning agent was an incredible 6,375 line program by a Physics major pursuing a minor in Computer Science. This impressive submission included obstacle avoidance and enemy behavior prediction algorithms.

FIGURE 2
MODIFIED TANKWAR SCREENSHOT

Notes on Retention

Although much focus is given to recruiting students to engineering, as the statistics in the introduction demonstrate, retaining existing students is also critical. A goal of any large assignment is to present worthwhile practice to a student, ideally encouraging retention through enjoyment and practical application to a range of computer science topics. If a student is interested in graphics, algorithms, simulation, artificial intelligence, or gaming in general, the assignment presented herein can offer something to keep him or her

Metric	Quantity
Freshman Retention Rate	74.3%
University Freshman Enrollment	3534
Department Retention Rate	71.4%
Mean Departmental Enrollment	734
Study Participant Retention Rate	82.4%
Study Participant Enrollment	60

TABLE 3
RETENTION STATISTICS FALL 2003 – SPRING 2006 N=60

interested. Statistics were gathered in the semesters following this study to compare retention rates among students exposed to gaming curricula compared to the departmental and overall University retention rates.

Table 3 presents statistics for those students participating in this study in comparison to all students in the Department of Computer Science and Engineering and all University freshmen. The course covered by this study is a freshman level course. Retained students for the department are defined as those who graduated from the College of Engineering or were still active in the College as of the writing of this paper. Other students had switched to non-Engineering majors, transferred to another school, or quit school altogether. Some of these students may have transferred to an engineering program at another institution - tracking students across multiple institutions is difficult, and thus the figures are approximate.

REFERENCES

1. S. Zweben, "2004-2005 Taulbee Survey," *Computing Research News*, May 2006.
2. E. Seymour, and N. Hewitt, *Talking about Leaving: Why Undergraduates Leave the Sciences*, Westview Press, 2000.
3. W. Waite, and P. Leonardi, "Student Culture vs. Group Work in Computer Science," *Proceeding of the thirty-fifth SIGCSE technical symposium on computer science education*, 2004, pp. 12-16.
4. J. D. Chase, and E. G. Okie, "Combining Cooperative Learning and Peer Instruction in Introductory Computer Science," *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education*, 2000, pp. 372-376.
5. J. Sheard, M. Dick, S. Markham, I. Macdonald, and M. Walsh, "Cheating and Plagiarism: Perceptions and Practices of First Year IT Students," *Proceedings of the Seventh Annual Conference on Innovation and Technology in Computer Science Education*, 2002, pp. 183-187.
6. M. C. Lewis, and B. Massingill, "Graphical Game Development in CS2: A Flexible Infrastructure for a Semester Long Project," *SIGCSE 2006.*
7. M. Gumhold, and M. Weber, "Motivating CS Students with Game Programming," *Proceedings of the 6th International Conference on New Educational Environments*, ICNEE, Sept. 2004.
8. M. Overmars, "Teaching Computer Science through Game Design," *Computer*, Vol. 37, No. 4, April 2004, pp. 81-83.

9. N. Parlante, J. Popyack, S. Reges, S. Weiss, S. Dexter, et al., "Nifty Assignments," *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, 2003, pp. 353-354.

10. N. Parlante, D. Matuszek, J. Lehman, D. Reed, J.K. Estell, et al., "Nifty Assignments," *Proceedings of the 35th SIGCSE technical symposium on Computer Science education*, 2003, pp. 46-47.

11. K. Claypool, and M. Claypool, "Teaching Software Engineering through Game Design," *Proceedings of the Conference on Innovation and Technology in Computer Science Education*, ITiCSE, June 2005.

12. H. Drury, J.M. Kay, and W. Losberg, "Student Satisfaction with Groupwork in Undergraduate Computer Science: Do Things Get Better?" *Proceedings of the Fifth Australasian Conference on Computing Education*, 2003, pp. 77-85.

13. S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: Local Algorithms for Document Fingerprinting," *Proceedings of the 2003 ACM SIGMOD International Conference on Data Management*, 2003, p. 76-85.

14. L. E. Deimel, "The Uses of Program Reading," *SIGCSE Bulletin*, Vol. 17, No. 2, 1985, pp 5-14.

15. D. R. Raymond, "Reading Source Code," *Proceedings of the 1992 Conference on the Centre for Advanced Studies on Collaborative Research*, 1992, pp. 3-16.

Ryan Garlick received the B.B.A. degree in finance from the University of Texas in 1995. He received his M.S. and Ph.D. degrees in Computer Science from Texas State University and Southern Methodist University in 1998 and 2003, respectively. Since 2005, he has been a Visiting Assistant Professor in the Department of Computer Science and Engineering at The University of North Texas. Dr. Garlick also consults in the field of electronic commerce.

Robert Akl received the B.S. degree in computer science from Washington University in St. Louis, in 1994, and the B.S., M.S. and D.Sc. degrees in electrical engineering in 1994, 1996, and 2000, respectively. He also received the Dual Degree Engineering Outstanding Senior Award from Washington University. He is a senior member of IEEE. Dr. Akl is currently an Assistant Professor at the University of North Texas, Department of Computer Science and Engineering. In 2002, he was an Assistant Professor at the University of New Orleans, Department of Electrical and Computer Engineering. From October 2000 to December 2001, he was a senior systems engineer at Comspace Corporation, Coppell, TX.