Intra-Class Competitive Assignments in CS2: A One-Year Study

Ryan Garlick and Robert Akl
Visiting Assistant Professor and Assistant Professor, University of North Texas, Department of Computer Science and Engineering, Denton, TX 76205 garlick@unt.edu, rakl@cse.unt.edu

Abstract – **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 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.**

Index Terms – CS2, Curriculum, Game Programming, Pedagogy.

INTRODUCTION

Nationwide, from 2003-2004, the number of newly enrolled students declaring a major in computer science declined by 10%, from 17,706 to 15,950 [1]. This follows a 23% decline in newly enrolled students from 2002-2003 [2]. As the majority of students leaving computer science do so by the end of their freshman year [3], 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 60 students engaged in a novel CS2 programming assignment over the course of two academic semesters 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. The presentation of several such avenues of further exploration would hopefully encourage students to pursue one or more.
- **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. Again, stressing the application of the techniques learned to a variety of problems would be an important task for the instructor.

- **Fostering collaboration among students.** Companies have reported that students graduating with Computer Science degrees are often under-prepared for group work [4]. Collaborative work in the early years of a computer science program for educational and social development can benefit retention as well [5]. 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 [6]. 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. This has traditionally been a problem in software development courses, as time limits prohibit the creation of a project 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 [7, 8]. Tools such as Game Maker [9] have been developed to capitalize on this topic, and several game ideas have appeared in the "nifty assignments" section of the SIGCSE bulletin [10, 11].

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 [12]. As mentioned in the previous section, drawing from multiple domains can also influence retention, as a student may discover an area that peaks 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, with details presented later. Gumhold [8] 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."

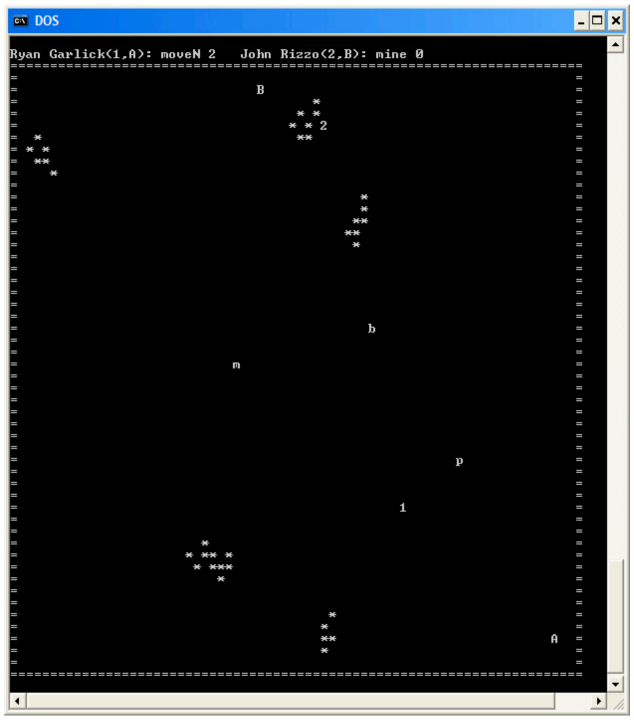
ASSIGNMENT DESCRIPTION

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.

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. Many students found creative and unforeseen ways to gain an extra edge in the game, even creating some modifications that were not allowed in the tournament, such as manipulating the random number generator and creating instances of other student's classes in order to call their action methods to see what they would do with the current world state. These loopholes were not anticipated by the instructor, but demonstrated the considerable effort students were investing to explore the game engine.



```
Ryan Garlick(1,A): moveN 2   John Rizzo(2,B): mine 0
```

FIGURE 1
TANKWAR SCREENSHOT

The assignment was designed to require the use of static methods, inheritance, stacks, and other topics from the lectures. A very basic agent was included with the source files given to each student (albeit one unlikely to do well in the tournament) as a sample player class. Students then extended this class, overriding methods to perform their chosen functionality. 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. Students were encouraged to read about obstacle avoidance, pattern matching and prediction of opponent moves, and storage and analysis of move information. These topics were discussed briefly in class, along with information on where to look for more details.

Aside from this direction, few guidelines were given for expected functionality of the finished product. No minimum acceptable solution was described. Similarly, the details of grading were undefined at the time of assignment, other than the assurance that the winner of the tournament would receive the highest possible score on the assignment and bonus points on the final exam. 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.

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 I presents statistics for the submitted programs in terms of code length. For comparative purposes, the minimally functional agent provided was 51 lines of code and the winning solutions for each semester were 1134 and 1716 code lines, respectively.

TABLE I
STUDENT SUBMISSION METRICS N=60

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

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. Figure 2 presents a scatter plot of student project submissions by lines of source code and eventual overall course grade.

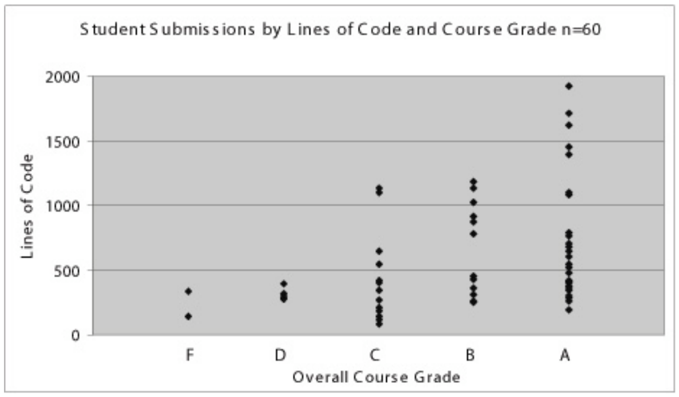


FIGURE 2
STUDENT SUBMISSION SCATTER PLOT.

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.

Figure 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.

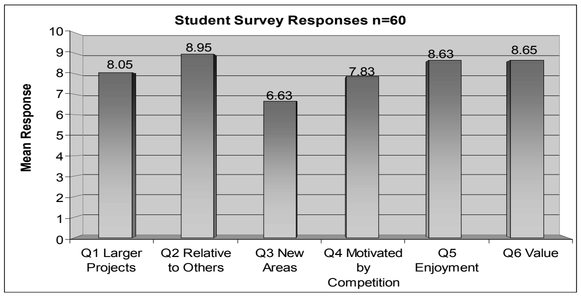


FIGURE 2
STUDENT SURVEY RESPONSES N=60.

TABLE II
CORRELATION BETWEEN SURVEY RESPONSE AND COURSE GRADE

	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 presents the results of correlating survey responses with the 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 lab-style programming assignments encountered in their academic careers.
- **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.

  Several students placing highly in the tournament rated Q3 lower, despite incorporating strategies outside the scope of CS2. A possible explanation for this response is students' desire to over-represent the degree of their experience.

  After the assignment was due, several students voluntarily modified the game engine to include sound, graphics, and extended features. Some student modifications are now incorporated into the engine, and these updates render existing player agents obsolete, preventing the extant code base from being passed around and "shortening the development cycle" in future semesters.
- **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 questions, 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 [4, 13], 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 [14]. 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 [15, 16]. Students rated the assignment 8.05/10 for improving their understanding of larger software projects.

CONCLUSIONS

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.

I. 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 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 III presents statistics for those students participating in this study in comparison to the Department of Computer Science and Engineering and University student bodies as a whole. 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 these figures are approximate.

TABLE III
RETENTION STATISTICS FALL 2003 – SPRING 2005 N=60

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

REFERENCES

[1] Zweben, S., “2003-2004 Taulbee Survey”, *Computing Research News*, May 2005.

[2] Zweben S. and Aspray, W., “2002-2003 Taulbee Survey”, *Computing Research News*, May 2004.

[3] Seymour, E, and Hewitt, N., “Talking about leaving: why undergraduates leave the sciences”, Westview Press, 2000.

[4] Waite, W., and Leonardi, P., “Student culture vs. group work in computer science”, *Proceeding of the thirty-fifth SIGCSE technical symposium on computer science education*, 2004, pp. 12-16.

[5] Chase, J. D., and Okie, E. G., “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.

[6] Sheard, J., Dick, M., Markham, S., Macdonald, I, and Walsh, M., “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.

[7] Lewis, M. C. and Massingill, B., “Graphical Game Development in CS2: A Flexible Infrastructure for a Semester Long Project”, *SIGCSE 2006*.

[8] Gumhold, M. and Weber, M., “Motivating CS Students with Game Programming”, *Proceedings of the 6th International Conference on New Educational Environments (ICNEE)*, Sept. 2004.

[9] Overmars, M., “Teaching Computer Science through Game Design”, *Computer*, Vol. 37, No. 4, April 2004, pp. 81-83.

[10] Parlante, N., Popyack, J, Reges, S., Weiss, S., Dexter, S. et al, Nifty Assignments. *Proceedings of the 34th SIGCSE technical symposium on Computer Science education*, 2003, pp. 353-354.

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

[12] Claypool, K. and Claypool, M., “Teaching Software Engineering Through Game Design”, *Proceedings of the Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, June 2005.

[13] Drury, H., Kay, J. M. and Losberg, W., “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.

[14] Schleimer, S., Wilkerson, D. S., and Aiken, A., “Winnowing: local algorithms for document fingerprinting.” *Proceedings of the 2003 ACM SIGMOD international conference on data management*, 2003, p. 76-85.

[15] Deimel, L. E., “The uses of program reading.” *SIGCSE Bulletin*, VOl 17, No. 2, 1985, pp 5-14.

[16] Raymond, D. R., “Reading source code”, *Proceedings of the 1992 conference on the Centre for Advanced Studies on Collaborative Research*, 1992, pp. 3-16.