Teaching Data Structures Using Competitive Games

Ramon Lawrence, *Member, IEEE*

***Abstract*—A motivated student is more likely to be a successful learner. Interesting assignments encourage student learning by actively engaging them in the material. Active student learning is especially important in an introductory data structures course where students learn the fundamentals of programming. In this paper, the author describes a project for a data structures course based on the idea of competitive programming. Competitive programming motivates student learning by allowing students to evaluate and improve their programs throughout an assignment by competing their code against instructor-defined code and the code of other students in a tournament environment. Pedagogical results indicate that the combination of game development and friendly student competition is a significant motivator for increased student performance.**

***Index Terms*—Active learning, competition, competitive programming, data structures, educational games, motivation, strategy.**

I. Introduction

MOTIVATING student interest is a challenging task. Instructors want assignments that allow students the opportunity to enhance their knowledge while being interesting enough for them to complete. In computer science courses, such as introductory data structures, students must learn the fundamental concepts and the necessary programming skills to apply them effectively. Thus, assignments must be designed that encourage the repetition of programming skills and discourage student procrastination.

This paper describes a project for data structure courses based on the concept of competitive programming. In a competitive programming project, students develop and improve their code throughout an assignment by competing in a tournament against instructor-defined code and the code of other students. The ability for students to evaluate their code against others encourages them to spend more effort in its development. In the project described in this paper, students write code that is the "artificial intelligence" for the computer to play a game. The student code is then uploaded to a server that allows students to challenge each other's code. The introduction of a competitive aspect to the assignment greatly increases student motivation and provides a useful form of evaluation.

The contributions of this work are as follows:

- a learning method called *competitive programming* that uses competition during an assignment to increase student motivation;
- a detailed description of a project for an introductory data structures course that combines competitive programming with game development;
- a supporting architecture for the game project based on open standards and software that is deployable in other courses (the supporting architecture allows for reduced grading time);
- a pedagogical evaluation of using a competitive programming project in an introductory data structures class.

The discussion begins with background information and a description of competitive programming. Then, detailed information is provided in Section III on a competitive programming project involving game development for a data structures course. Section IV covers the pedagogical results of using the project in the course, and the paper ends with some conclusions.

II. Background

One of the foundation courses for computer science and engineering students is introductory data structures, which is typically taken in the second year. In this course, students are exposed to fundamental programming constructs, such as lists, stacks, queues, and trees. There has been extensive work on creating tools for visualizing data structures to promote learning [1], and many such resources [2] are available on the Internet. At this time, students begin to solve larger programming assignments and gain valuable programming experience. One of the common reasons why students fail to complete the data structures course is that they do not complete the programming assignments or complete them in a substandard fashion. Since it is critical to the students' success that they successfully complete the assignments, instructors are challenged to developed interesting assignments that students want to finish.

The use of games to promote student learning has been well documented. Games capture student interest because they are fun and exciting, and students tend to learn more when actively engaged by the subject. Developing games as assignments has been used [3], [4] in introductory data structures and programming courses. The motivation for introducing games as projects is simple: most students have intimate contact with computer games before their formal computer education begins, and computer games are often what attracts and motivates them to learn more about programming and computers in general. Although arcade-style computer games attract most student interest, strategy games such as checkers, chess, and Go normally have more pedagogical benefit. Game projects [5] can be modified to satisfy the needs of larger introductory courses and small advanced group projects. Games have also been used in the instruction of interprocess communication [6] and operating systems [7].

Manuscript received June 13, 2003; revised October 4, 2003.

The author is with the Department of Computer Science, University of Iowa, Iowa City, IA 52242-1419 USA (e-mail: ramon-lawrence@uiowa.edu).

Digital Object Identifier 10.1109/TE.2004.825053

The unique contribution of this paper is a project that combines the implementation of strategy games with a real-time, competitive tournament environment. Although some of the previous work [3], [5], [6] discussed using tournaments, the tournaments were performed after the project was completed. In this paper, an architecture was developed for automated student competition *during* the assignment and used to motivate student success. An open-source project called CodeWars [8] allows development of competing artificial intelligence (AI) code in a game environment, but this project is thought to be the first pedagogical evaluation of using competitive gaming in an educational setting.

The project described in this paper is a particular implementation of the more general notion of competitive programming. Competitions are common in many areas such as AI, robotics, and programming. However, they are rarely used in the classroom setting to encourage learning. *Competitive programming* allows students to improve and evaluate their programming skills during an assignment by competing their code against instructor-defined code and the code of other students. The introduction of the competitive tournament increases student motivation and reduces procrastination, a common cause for students failing to complete assignments. Motivated learners [9] generally have greater success.

The competition aspect also increases student interaction, which involves students of different learning styles [10]. However, the effectiveness of competitive programming may differ by gender. Female students tend to value cooperation over competition and have different educational needs in a computer science program [11]. To this end, specialized games have been developed for girls [12], [13]. An evaluation of how introducing competition to assignments affects female learning is an interesting subject that is not the focus of this paper.

The project described in the next section combines competitive programming with game development. Thus, there are two motivational factors in this project. First, developing games is interesting to students. Second, the ability to compete against other students' code in a tournament creates another interesting challenge. A competitive programming project does not always have to involve writing code for games. As long as some method for comparing and ranking the effectiveness of code exists, using competition during the assignment is possible.

III. Project Overview

This project is a capstone project at the end of the data structures course. The students have approximately three weeks to complete it. The project involves writing code to implement the game intelligence functionality for a board game. The game itself is called Critical Mass and was chosen because it has the beneficial feature of not allowing a draw (the game always has a winner). However, the project is general, and other games are substituted for variation. Students write their code to determine the moves that their computer program, called a "bot," will make. The student code is evaluated based on whether their code plays the game correctly and how well it plays against predefined "bots."

Writing code for game intelligence [14] requires writing a board evaluator, a move generator, and a game tree. Creating a board evaluator requires manipulation of a two-dimensional array and devising cost functions based on piece locations. Although the amount of code for a board evaluation function is typically small, writing a good function requires significant effort in understanding how to play the game. A move generator requires the student to write code that determines the valid moves that can be made given a board position. For many games, this requirement is straightforward but may require a decent amount of logic if the piece movement characteristics are not uniform (e.g., chess). Developing code for a game tree is an excellent test of a student's understanding of the fundamental concepts of recursion and trees. Recursion and game trees are covered in the lecture, and sample code is given for the game of tic-tac-toe. Advanced topics such as alpha–beta pruning are also covered. Although some general discussion on board evaluation is given, the students are left to determine their own board evaluator and move generator for Critical Mass. The students are given a description of the Critical Mass rules as described hereafter.

Critical Mass uses a 5×6 board of squares. The object of the game is to remove completely all of the opponent's pieces from the game board. Each cell may contain zero or more pieces. The pieces are of two different colors, one belonging to each player. All pieces in a cell are always of the same color. The two players alternate moves. To make a move, a player must place a piece of his or her own color into any cell, which does not contain a piece of the opponent's color. If the number of pieces in a cell becomes greater than or equal to the number of adjacent cells, then that cell explodes. When a cell explodes, the pieces in that cell are removed, one additional piece is added to each adjacent cell, and all pieces in adjacent cells become the color of the player whose move caused the explosion. Explosions may cause subsequent explosions. If an explosion causes additional explosions, then the new explosions happen simultaneously, not by means of a wavelike chain reaction. The game ends when, after any move except the first, all pieces on the board are the same color. The player corresponding to that color wins.

A. Pedagogical Constraint

The project has students develop in C++, which is the language of instruction for the data structures course at the University of Iowa. The students develop using `gcc` on Linux machines. One of the weaknesses of this environment is the difficulty in developing graphical applications. An instructor whose focus is on teaching data structures does not have the class time to teach students how to use the complex graphics libraries available. Thus, most development is performed using command-line compiling and text-based interaction.

The use of text-based assignments has less appeal to students than graphical assignments, especially when developing games. One of the contributions of this project is an architecture for combining C++ programs with Java user interfaces for game development. Previous attempts at the project used Java to code a graphical front end that communicated with C++ code using Java Native Interface (JNI). This technique worked well and allowed students to develop stand-alone games with graphical

```
struct CMsquare
{
   int player;    // 0 - if empty, 1 - if player 1, 2 - if player 2
   int bombs;     // # of pieces in the square
   int row, col;  // row # and col # of square starting from (0,0)
   int max_bombs; // Maximum # of pieces in the square
};

class CMboard
{
public:
   int getPlayer(int r, int c) const       { return bd[r][c].player; }
   int getNumPieces(int r, int c) const    { return bd[r][c].bombs; }
   CMboard& operator=(const CMboard &bd);
...
private:
   CMsquare **bd;         // Game board is a dynamic matrix
   int max_row, max_col;
};
```

Fig. 1. Critical Mass board class in file `CM.h`.

```
namespace student // Replace student with your ID
{

class CMGame
{
public:
   void makeMove(const CMboard& b, int player, int& row, int& col);
   bool insertBomb(CMboard &b, int p, int r, int c );

private:
   /*
   Add appropriate methods to your class in order to determine the best move such as:

   int evaluate(const CMboard &b, int player);
      Postcondition: Returns an integer value indicating how desir-
able board b is for given player.
   */
};
} // end namespace
```

Fig. 2. Critical Mass game class in file `student.h`.

front ends. However, JNI cannot be used in applets on web pages and, thus, cannot be used for developing a competition website.

Although the implemented architecture satisfies the pedagogical constraint of allowing students to code their game intelligence in C++ and visualize the game in Java, this contribution is secondary to the fact that the architecture allows competition during the assignment period. The use of competition during the assignment is beneficial and independent of the course and development environment. Thus, this architecture has wide applicability to any course project that can be structured as a competition.

B. Student Requirements

The students are given a listing of the game rules for Critical Mass and rules for the competition. C++ code files are provided as a template that the students must follow. The students are required to follow the code template to ensure that their code is compatible with others when the competition is performed. The template files consist of `CM.h` which contains code for a Critical Mass board class and related functions, and `student.h`, which contains an empty game class with methods that the student must write. Abbreviated class headers for these files are given in Figs. 1 and 2, respectively. The student is also given the file `CMconsole.cxx`, which allows them to play against their code on the console in human versus computer mode for debugging. The code presented here is specific to the game Critical Mass, although it is straightforward to generalize the code for any game.

Given these initial code files, the student starts writing the methods that select the move to make. The two public methods that the student must write are `makeMove`, which passes in a current game board and asks them to return the row and column where they want to move, and `insertBomb`, a method that, given a row and column, makes the move and produces the resulting game board. Students can write any other methods that they want in the class `CMGame`.

The hardest part of any major project for students is knowing where to start. The sample code compiles without any modification. The instructor then encourages the students to write a very simple `makeMove` method that generates random moves. This method allows students to get going quickly without feeling totally overwhelmed. Then, students iteratively improves their code.

Using namespaces in the student code file (`student.h`) is critical to the success of the project. The students were instructed to rename the file and the namespace in the code to their unique user id. This renaming allows each student's code

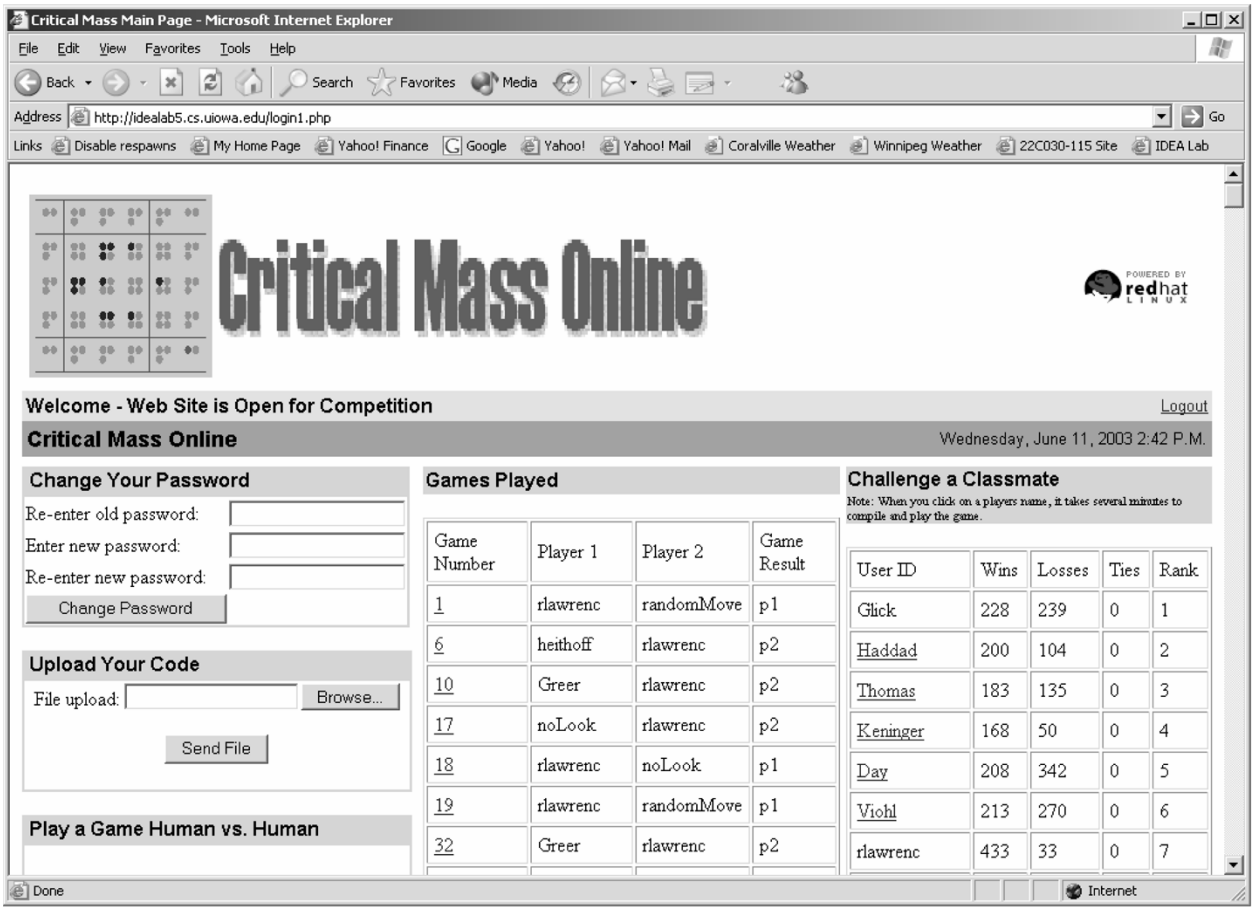

Fig. 3. Student main screen on competition server.

to be individually referenced and will be discussed more in Section III-D.

C. Evaluation and Competition

The major unique feature of this project is that students do not develop their code in isolation. Rather, once a student has his/her code working, he/she uploads it to a web server that hosts a competition between all students in the class. This competition site ranks the student based on the performance of his/her code. A student can then challenge other students' code to improve his/her ranking.

Students are evaluated on the project based on whether they can get their code to work properly and then on the success of their implemented code in the competition. The basic evaluation was as follows:

- *25 marks*—if code successfully compiled on competition server and could play one game (regardless of outcome);
- *35 marks*—if code could beat `randomMove`, a bot that performed random moves;
- *45 marks*—if code could beat `noLook`, a bot with a board evaluator but no game tree;
- *50 marks*—if code could beat `heithoff`, a bot with a board evaluator and game tree with four levels of look-ahead;
- *60 marks*—if code could beat `rlawrenc`, a bot with a very strong board evaluator and game tree with four levels of look-ahead;
- Bonus marks awarded for the top 10 ranked students.

The goal of the evaluation criteria is to provide a mechanism for efficiently and effectively evaluating student code. In practice, determining whether game tree code is working correctly with no errors is extremely difficult; therefore, the idea was to assign grades based on results. Presumably, code with major errors in game tree logic would not be able to beat some of the advanced bots, although this initial assumption is discussed further in Section IV. Pregenerated bots of increasing levels of difficulty were created by the instructor and placed in the competition. The base assignment is out of 50 marks, although students could get bonus marks by being top in the competition or beating the best bot (`rlawrenc`).

D. Website Architecture

The competition website was developed by the instructor and an undergraduate honors student who had previously completed the course. One of the major design goals was that the architecture use only open-source software. Thus, the operating system was Red Hat Linux 7.3, the web server Apache 1.3, the database MySQL 3.23, and PHP (hypertext preprocessor), Java, and HTML were used for web scripting, database access, and web page development.

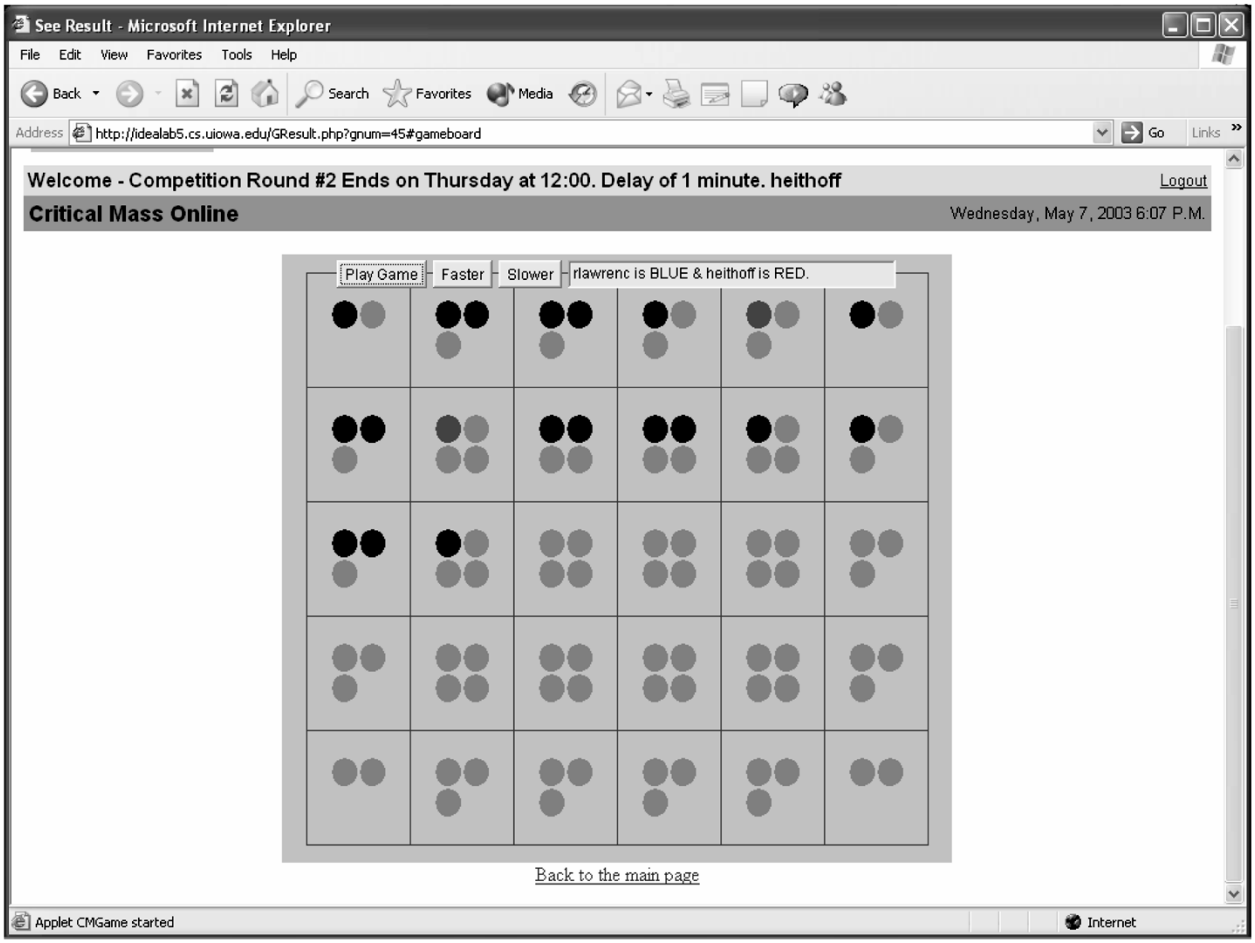

Fig. 4. Student viewing game result using Java applet.

For each student, the database contains login information, a link to the code file, and a record of the student's performance. It also contains a list of all games played, and for each game played, there is a game string that encodes in textual form all the moves made in the game.

When a student is satisfied with how his/her game code is working, he/she logs on to the competition server and uploads his/her code. During the code upload, the student's code is compiled with a C++ game controller that validated that the student's code had the proper format and could make reasonable moves. On a successful upload, the student is assigned a rank one less than the current lowest rank in the competition. Thus, students who uploaded their code first have an advantage by getting a higher initial rank.

At this point, the student has the ability to challenge other students. Typically, a student will want to challenge another student who has a ranking higher than his/her own. The simple challenge system devised is that a student could only challenge another student every minute, and if the challenger wins, he/she gets the rank of the student challenged. Further, a student could only challenge other students within five ranks (plus or minus) of his/her current position. Students could also re-upload an improved version of their code at any time and maintain their rank, view games that were previously played, and play a graphical game human versus human in order to understand the game more fully. The main user screen is shown in Fig. 3.

To allow students to play each other's code, the server must maintain the code of all students and, when a challenge is made, dynamically build a program that will play the game. The solution exploited the use of C++ namespaces. When a challenge was made by clicking on a student link, the PHP code ensures that the students do not currently have a game running and that the challenge is valid. Then, the PHP code issues SQL (structured query language) statements to the database to find the code of both students on the web server. The code for each student is in a header file with his/her login id as the file name and with his/her own namespace. Thus, a student with id `Beth` will have namespace `Beth` and file name `Beth.h` storing her code. Header files for both students are combined with a game controller template file. This C++ code creates a game board and alternates between asking each student's code to make a move. The game controller template file is copied to another file, the place holders for student names (`PLAYER1`, `PLAYER2`) are replaced with the actual student names, and the file is compiled and run. Once the game is completed, the game controller saves the result and game string to the database. The outcome of the challenge is then presented to the user, who can then replay the game result using a Java applet (Fig. 4). The game controller code is presented in Fig. 5.

For example, if student `Beth` has uploaded her code and is currently in tenth position, she can challenge students from

```
#include "../student/PLAYER1.h"
#include "../student/PLAYER2.h"
...
using namespace PLAYER2;
using namespace PLAYER1;
...
int main()
{
   PLAYER2::CMGame p2;
   PLAYER1::CMGame p1;
   CMboard b(5,6), temp;
   string gameString;

   while (b.getStatus() == 0)
   {  row = -1; col = -1;
      temp = b;
      p1.makeMove(temp,1, row, col);

      if(insertBomb(b,1, row, col))
         gameString = gameString + convertToString(row) + convertToString(col);
      else
         // Invalid move by player 1 - Player 2 wins by forfeit

      if(b.getStatus()!=0)  break;

      row = -1; col = -1;
      temp = b;
      p2.makeMove(temp,1, row, col);

      if(insertBomb(b,1, row, col))
         gameString = gameString + convertToString(row) + convertToString(col);
      else
         // Invalid move by player 2 - Player 1 wins by forfeit
   }

   // Save game information (winner and loser) and gameString to database
```

Fig. 5. Game controller code.

the fifth position to the fifteenth position. Typically, she will challenge a student in thefifth position. If `Steve` is in the fifth position, `Beth` challenges `Steve` by clicking on his link in the rank list. Clicking the link executes a PHP script that takes the two students `Beth` and `Steve` as parameters. This script retrieves both student records from the database. If the challenge is valid, the PHP script (which runs on the server), copies the game controller code to a file called `BethSteve.cxx`. A string replacement function is run on the new file to replace all instances of `PLAYER1` with `Beth` and instances of `PLAYER2` with `Steve`. The player's code is kept in a separate directory on the server that can only be accessed by the PHP code. `BethSteve.cxx` is compiled, and then the executable is run. When the game is complete, the C++ executable saves the game string to the database and assigns the game a unique number. An example game string would be "00 01 00". This game string results in a win for player 1 and corresponds to moves (P1, 0, 0), (P2, 0, 1), and (P1, 0, 0). During the time the game is running, the PHP script is displaying a wait message and then displays the final result when the game is completed. The student can then go back to the main screen to view the game. Clicking on the game number in the main screen executes PHP code which uses the game number to retrieve the game string from the database and create a new page with a Java applet that takes the game string in as a parameter and shows the moves made. The student can speed up or slow down the replay of the game.

IV. Pedagogical Results

The game competition project was tested on a data structures class consisting of 55 students. Unlike normal introductory data structures classes, this offering consisted of a larger percentage of senior (53%) and junior (25%) students than would normally be expected for a sophomore class. This student distribution is partly a result of the course being the second offering in the year (spring semester), and most computer science majors would have taken it in the fall. Thus, the class consisted of a much larger percentage of students who are taking data structures as a required part of a noncomputer science degree, such as mathematics and computer and electrical engineering. In addition, a significant number of students in the course had just previously failed or dropped the fall offering of the course.

About 85% of the students registered in the course at least got their code to compile on the server. Common complaints among students were that there was not adequate time to complete the project, especially with respect to other courses late in the semester. Eighty percent of the students beat `randomMove`; 65% beat `noLook`; 50% beat `heithoff`; and 7% beat `rlawrenc`. During the assignment period, over 5000 games were played, and some students in the course played over 500 games. The "stickiness" of the website was demonstrated by a user survey that found that more than 40% of the surveyed students logged onto the game site more than 20 times, and 88% logged on at least six times.

The marking for the project was very efficient since it was based on student performance. Simple database queries were written that returned the students who beat each predesigned bot and allowed initial marks to be assigned. Then, each student's code was evaluated electronically by having the student's code challenge the four test bots. For this particular game, random moves sometimes beat even the best bots so some marks were adjusted by manually examining the code to detect students who did not build a board evaluator and game tree. Marking the entire project took less than five hours for the professor. Interesting future work would be to automate this process and to run algorithms to detect any sharing of code among students.

The student satisfaction with the project was very high. Students were asked to fill out an anonymous survey on the project after they completed it but before marks were assigned. In the survey, students were presented with statements and asked to respond if they strongly agreed, agreed, were neutral, disagreed, or strongly disagreed. The survey had 42 responses of the 55 registered students. All 42 respondents either strongly agreed (60%) or agreed (40%) that the assignment was interesting, and all but one strongly agreed (74%) or agreed (24%) that it was challenging. When asked their impressions of the game project with respect to other assignments, 78% either strongly agreed (47%) or agreed (31%) that it was the best assignment in the class. Even more telling, 81% either strongly agreed (50%) or agreed (31%) that "the Critical Mass assignment was one of the best assignments so far in any computer science course."

Further, the percentage of students that either strongly agreed or agreed that the assignment increased interest in the course was 88%, that the game project made the course more interesting was 88%, that the project helped them become a better programmer was 86%, that the project helped them learn game trees better than just the lecture notes was 72%, and that the assignment helped them learn the challenges in building games was 88%.

The major unique factor of this project was that the tournament was run during the assignment in order to increase student participation and motivation in the project. The benefit of this approach was surveyed by asking the students to agree or disagree with two statements with the following results:

- "T*he tournament feature of the assignment motivated more effort into doing the assignmen*t." Eighty-nine percent of the students either strongly agreed (60%) or agreed (29%) with this statement.
- "*The tournament feature of the assignment is more interesting than developing a stand-alone game with no student competitions*." Ninety percent of the students either strongly agreed (64%) or agreed (26%) with this statement.

In the survey, the students had the opportunity to have general anonymous comments about the assignment. Although there were some negative comments about the amount of work, system problems with the competition server, and issues on ranking, the comments were overwhelmingly positive. One comment was especially appropriate and encompasses many of the statements echoed by the students:

> I'd just like to say that this is the most innovative method used for an assignment so far in any computer science course that I've taken. Placing students in direct competition in this way and having them build something that they can truly have fun with makes this project a much better learning experience than if we were told to simply make a game tree and turn it in. The motivation behind seeing how well my code stands in contrast to others really forced me to work harder than I otherwise would have on this project.

Although these survey results do not determine whether the students were more successful in learning game tree intelligence, obviously the students worked harder, wrote more code, and were motivated to complete the assignment. Since motivated students generally are more successful learners [9], the assignment probably met its instructional goals.

A. Architecture Improvements

The competition server architecture worked well considering the technical challenges involved. The technique exploiting namespaces to differentiate student code submissions is a useful trick that instructors can use for other projects. The author has made all source code and project materials available on the web (http://idealab.cs.uiowa.edu/teaching). Although there is an appropriate amount of setup required [15] to configure the database and web server, deploying this project is not beyond the abilities of most instructors. The template code can be modified to support different games and assignments, and construction of the website is a good senior undergraduate project.

The current competition server implementation is not very robust. During the competition, some students would upload code that would either contain a segmentation fault or an infinite loop. If the code caused a segmentation fault, the game controller would never complete and no result for the game would be recorded. In addition, the feedback to the student about the error was poor because the site was not intended for debugging. An even more serious problem was when a student's code contained an infinite loop. In this case, the game would never complete and would consume valuable server resources. Further, both students involved in the game could not continue to participate in the competition (since they could compete in only one game at a time). Thus, the site required active system administration, especially during the opening of the competition.

Future work involves handling the problems with infinite loops and segmentation faults by using interprocess communication (IPC) and a client–server architecture. Since student code was compiled into the same process as the game controller, any problems with the student code affected the game controller. By having the student code communicate with the game controller through IPC as separate processes on the server, most of the problems can be resolved. One problem with this approach is that the student code becomes more complex than just using header files and namespaces (two concepts already taught in the course), and the students have more difficulty debugging code without being on the server.

The other major improvement that did not get implemented is displaying the game as it is being played. The architecture

could allow the game to be shown in Java as it is being played by having the Java applet query the database for the moves while the C++ game controller is generating them. However, the authors ran out of time for implementing this feature.

One factor that must be considered in deploying such a project is security. Uploading and executing C++ code is an inherent security risk. Thus, a dedicated machine was used for the project, and although possible security holes exist, they had no effect on the competition.

V. Conclusion

Making introductory data structures courses interesting and challenging requires projects that motivate students to enhance their programming abilities using the new data structures. This paper has discussed how a project involving competitive gaming motivates students to learn advanced game intelligence programming and improves their opinion of the course overall. The use of competition during an assignment can be used in any course where a suitable project can be developed. The competition server architecture can be used to enable the competitive environment. The major contribution is that interactive competition during the assignment increases student effort and satisfaction compared with projects where the competition comes after the assignment is completed.

Acknowledgment

The author would like to thank E. Heithoff for her work on the competition website as well as the class of introductory data structures in spring 2003 for their participation.

References

[1] J. Stasko, J. Domingue, M. Brown, and B. Price, *Software Visualization*. Cambridge, MA: MIT Press, 1998.

[2] P. Brummund. (2003) Complete Collection of Algorithm Animations (CCAA). [Online]. Available: http://www.cs.hope.edu/alganim/ccaa/index.html

[3] J. Adams, "Chance-it: An object-oriented capstone project for CS-1," in *Proc. 29th ACM Special Interest Group on Computer Science Education (SIGCSE) Technical Symp. Computer Science Education*, 1998, pp. 10–14.

[4] K. Becker, "Teaching with games: The minesweeper and asteroids experience," *J. Computing Small Colleges*, vol. 17, no. 2, pp. 23–33, 2001.

[5] T. Huang, "Strategy game programming projects," in *Proc. 6th Annu. Consortium for Computing Sciences in Colleges (CCSC) Northeastern Conf. Computing Small Colleges*, 2001, pp. 205–213.

[6] D. Reese, "Using multiplayer games to teach interprocess communication mechanisms," in *ACM Special Interest Group on Computer Science Education (SIGCSE) Bull.*, vol. 32, 2000, pp. 45–47.

[7] J. Hill, C. Ray, J. Blair, and C. Carver, "Puzzles and games: Addressing different learning styles in teaching operating systems concepts," in *Proc. 34th ACM Special Interest Group on Computer Science Education (SIGCSE) Technical Symp. Computer Science Education*, 2003, pp. 182–186.

[8] (2003). CodeWars, CodeWars Open Source Project. [Online]. Available: http://codewars.sourceforge.net

[9] S. Fallows and K. Ahmet, *Inspiring Students: Case Studies in Motivating the Learner*. London, U.K.: Kogan Page, Ltd., 1999.

[10] R. Felder and L. Silverman, "Learning and teaching styles in engineering education," *Engineering Education*, vol. 78, no. 7, pp. 674–681, 1988.

[11] J. Margolis and A. Fisher, *Unlocking the Clubhouse: Women in Computing*. Cambridge, MA: MIT Press, 2002.

[12] C. Gorriz and C. Medina, "Engaging girls with computers through software games," *Commun. ACM*, vol. 43, no. 1, pp. 42–49, 2000.

[13] M. Klawe, M. Westrom, K. Davidson, and S. Super, "Phoenix quest: Lessons in developing an educational computer game for girls. . . and boys," in *Proc. Conf. Multimedia Technology Management*, 1996, pp. 264–274.

[14] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2002.

[15] E. Heithoff, "The construction of a C++ teaching aide (honors thesis),", Department of Computer Science, University of Iowa, Iowa City, IA, 2003.

Ramon Lawrence (M'02) received the B.C.Sc. (Hons.) and Ph.D. degrees from the University of Manitoba, Winnipeg, MB, Canada, in 1996 and 2001, respectively.

He is currently an Assistant Professor at the University of Iowa, Iowa City, with teaching interests in data structures and database systems, and the Director of the Iowa Database and Emerging Applications (IDEA) Laboratory (http://idealab.cs.uiowa.edu). He is an active supporter of undergraduate honors research and projects. His current research area is database integration and querying.