THE CONSTRUCTION OF A C++ TEACHING AIDE

by

Elizabeth Heithoff

A thesis submitted in partial fulfillment of the requirements for graduation with Honors in the Department of Computer Science

---

Ramon Lawrence
Honors Thesis Supervisor

May, 2003

All requirements for graduation with Honors in the Department of Computer Science have been completed.

Douglas Jones
Undergraduate Chair of Computer Science

ABSTRACT

Thesis: Constructing the hardware, software, and architecture to write a dynamic web application to be used as a C++ teaching tool.

This paper discusses the details of an undergraduate honors project involving the construction of a complete application system including the hardware, operating system, and software. The application system is used as a teaching tool for a computer science course, in order to motivate students to take a more active role in the course. The first goal of the project was to build a computer on both the hardware and software level that could be used as a web server for the application. The next goal was to actually create the web site that would allow students to upload their code to the server, compile the code, and use the code to host a tournament between the students in the class.

The application is dynamic and utilizes a database. The application also requires many different web-programming languages to implement the different features of the site. An important design challenge is making the web site look professional, but still be appealing to the students who use it. The result was a complete application built from the ground-up that was a valuable C++ teaching tool and interesting project that motivated and encouraged students.

TABLE OF CONTENTS

SECTION 1: INTRODUCTION

Computer programmers are essential to the technological advances of a company. However, they require system administrators who are knowledgeable about the systems on which the programs run. Introductory computer science courses primarily focus on programming. In order to gain experience with commercial hardware and software, it is necessary to perform an independent honors project or work in a business environment. This honors project provides practical experience in hardware setup and configuration. It also provides exposure to infrastructure-related software applications such as Apache, MySQL, and the Linux operating system. Lastly, it provides an opportunity to develop a complete web application that will directly use the infrastructure. While providing many additional educational experiences to supplement course work, the project also has practical importance. It is used for teaching C++ to Computer Science III students by creating a challenging and exciting project as the final assignment in that class. The remainder of this thesis will examine motivation for the project, a project summary, background information, a system description, the results, the conclusions, and future work.

SECTION 2: MOTIVATION

In an ideal situation, programmers within a company that create and maintain applications would completely understand the platform on which the programs run. However, in many large corporations, there are two completely separate sides of the IT department; the developer side and the architecture side. One of the major problems that employees on the architecture side encounter is that programmers do not fully understand the web architecture. This can result in many problems related to security, efficiency, and reliability. This project allows the ability to learn how the separated tasks of the IT department can be brought together.

Currently, students of Computer Science III write several projects in C++. The web application constructed in this work helps to motivate the students to create more efficient projects and gives them something to look forward to once their project is completed. By providing an interesting assignment involving games, competition, and interaction, the web application motivates students to do their best and makes the course more interesting. The competition provides an opportunity for student-to-student constructive criticism. This project provides a valuable teaching tool for the students of Computer Science III.

SECTION 3: PROJECT SUMMARY

Thesis: Constructing the hardware, software, and architecture to write a dynamic web application to be used as a C++ teaching tool.

There are two goals associated with this project. The first is to gain knowledge and experience with the infrastructure associated with web development. This involved gathering the hardware for a powerful machine, assembling the parts, and then installing the software necessary to host a web site. The software included Linux, Apache, MySQL, and PHP.

The second goal is to develop a web application that allows students of Computer Science III to upload their code for *Critical Mass* and play their classmates on the Internet. This was accomplished using HTML, PHP (necessary for communication to the MySQL database), C++, and a JAVA applet. The applet allows the students to go back and watch the games they played. The C++ program is used to "glue" together the programs of the two students who wish to challenge each other.

Contributions

The contributions of this work include experience with building and assembling hardware, installing the Linux operating system and related software, constructing a game site, learning HTML and PHP, and creating a valuable project and learning experience for Computer Science III students.

SECTION 4: BACKGROUND

Development Process and Goals

There are three components of this project. Project ideas are generated when a user makes a request. In this situation, there was a need for a web application to be used as a teaching tool. So the first component is the application, design, and development. It is this component that is visible to the students. The students are using the application for a game they programmed called *Critical Mass*. *Critical Mass* is a variation of *Minesweeper* and is played by placing bombs on a game board, attempting to blow up all of your opponent's pieces. The students and the professor of this course needed a web application that would allow the students to log on to the site. Then the students needed to be able to upload their code to be saved to the web server. They also needed to be able to challenge other student's code, in a tournament environment. The games that were played when one student challenged another also needed to be available for later viewing. One design limitation is that student code is written in C++, but it is difficult to use dynamic C++ code on the World Wide Web and do graphics. The application also has other common options such as changing their password.

Once the design goals have been determined, it is necessary to get a view of what the application will actually look like. When a student clicks on a link to this application, the first screen they see is a login screen. Once properly logged in, the student is presented with a variety of choices, including uploading their code, playing a game, changing their password, and viewing previous games. They are able to see directly where they stand compared to other students in the class as far as who has won the most games. They are able to challenge students within five rankings of themselves. In order to challenge other students, they click on the student's name. This will trigger a game to be played. The application allows a student to challenge another person's program,

computer versus computer. Once the game is finished, the student can request to see what went on during the game. The game will then be played back for them with a similar GUI as in their real program. One other option for the students is the ability to play human versus human.

Once a problem is identified and specifications are made about what must be done in order to suit the user's needs, it is necessary to identify what tools are required to support the application. The second component to this project is the hardware and software infrastructure that supports the application. This involves all the components of the application that need to be created, and also how they work together.

While developing this application, many challenges were encountered. The first was to find out how to communicate between a program written in C++ and a MySQL database. The research done showed that it was necessary to use the MySQL++ library (see Appendix G). While the syntax of database communications varied slightly from that used with PHP, the queries are essentially the same. Another challenge was to allow students to play each other manually. This challenge was accomplished by using Java GUI within the Java Applet.

Lastly, once it has been decided what languages the program will use, it is necessary to choose the appropriate platform on which to run the program. The first step is to acquire and assemble the parts necessary for a high-performance machine. The infrastructure configuration will consist of many parts. It was necessary to use open source software for the development of this application since no other funding was available. In order to write the programs, a basic editor was used that comes with Linux (gedit). Apache is an open source program that is used as a web server for this project. The database is MySQL. Finally, the operating system on which the whole project runs is Linux. Linux is also open source software.

Open Source Software and Languages

A major part of this project is to only use open source software. In order to determine which software was appropriate, some research was necessary. This section is provided to give the reader some background information on the tools used to complete the project. The operating system used is Red Hat Linux. The web server software used is Apache. The database used is MySQL. The different web languages used include HTML and PHP.

The operating system used is Red Hat Linux version 7.3. Linux is a free operating system that was created by Linus Torvalds in 1991. Linus started Linux by writing a kernel, which is the heart of the operating system, mostly from scratch, but partly by using publicly available software. After that, he released it out to his friends and other computer hackers on the Internet in order for them to help him enhance it. Documentation on Red Hat Linux can be found at www.redhatlinux.com.

The web server used is Apache. Apache is an open-source HTTP server that can be used with modern operating systems such as UNIX, LINUX, and Windows NT. The goal of Apache is to provide a secure, efficient, and extensible server that provides HTTP services in sync with the current HTTP standards. Apache has been the most popular web server on the Internet since April of 1996. The August 2002 Netcraft Web Server Survey found that 63% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined. More documentation on Apache can be found at http://httpd.apache.org.

As a base language for this web site, HTML is used. HTML stands for Hyper Text Markup Language. There are two essential features of HTML, hypertext and universality. Hypertext allows you to create a link in a web page that takes a visitor to

any other web page on the Internet. HTML is also universal because since HTML documents are saved as ASCII or Text Only files, virtually any computer can read a web page, regardless of the operating system or browser. HTML allows the developer to format text, add graphics, sound, and video to a web page in such a way that all users can view it. The key to HTML is the use of tags, which are keywords enclosed in tags. For example, to make a word a link you would do the following:

```
<a href="mywebpage.com">Go to My Web Page </a>
```

Since HTML is only good for formatting web pages and linking several pages together, it was necessary to use another web development language to communicate with the database. For this, PHP is essential. PHP is a scripting language similar to languages such as C, Perl, and Pascal. This web site uses version 4.1.2 of PHP. PHP stands for Hypertext Preprocessor. PHP is generally embedded or combined with HTML. PHP code that is embedded in HTML pages is processed by the PHP module of Apache. PHP has many libraries that make it possible to connect to databases. More documentation on PHP can be found at www.php.net.

The database chosen is MySQL. MySQL is open source computer software that allows the user to create, maintain, and manage electronic, relational databases. A key feature of MySQL's performance design goals is multithreading, or the ability to perform many tasks at the same time. For this website, MySQL version 3.23 is used. The Structured Query Language (SQL) is used to read and write to MySQL databases. SQL allows a person to search for, enter, modify, and delete data in a database. PHP uses SQL commands to communicate with the MySQL database. In addition, MySQL++ allows the communication between the MySQL database and a C++ program. More documentation on MySQL and MySQL++ can be found at www.mysql.com.

Architecture Overview

The student login information is stored in a MySQL database. Also stored in the database is information including their number of wins, losses, and their rank in relationship with the rest of the class. Students upload code written in C++. That code is then stored on the web server for future use. The web server serves pages of HTML and Java Applets, which are embedded with PHP code. The PHP code is used for user interface and allows users to dynamically challenge other players and connect and update the database. PHP also communicates with the command line of Unix in order to compile and run the game controller. The game controller interacts with both of the students' code, prompting each separate C++ program for the next move. After the game is played, by using MySQL++, the game controller inserts a string to the database containing all the moves that were made throughout the game and the outcome of the game. This way, once the game is finished, the student can opt to play back the game.

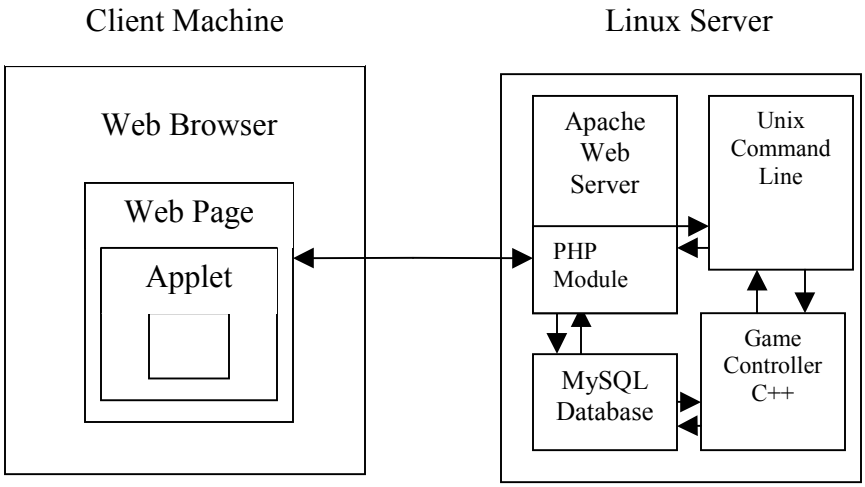

Figure 1: System Architecture

All basic web pages are done in HTML and PHP, and the student can view game results using the Java applet. This Java applet has also been extended to allow human-vs.-human games to be played. The way in which the C++ code and the Java code is linked is unique, as well as the dynamic way the C++ code is compiled. This architecture is illustrated in Figure 1.

Critical Mass Rules and Examples

It is important to understand some information about the course Computer Science III and the game Critical Mass. Computer Science III is a course offered at University of Iowa and it is part of the required curriculum of the computer science major. Computer Science III involves teaching C++ and data structures such as linked lists, stacks, and queues. Another topic covered in Computer Science III is the concept of game trees and game intelligence. As a final project for the course Computer Science III, students are asked to write the code for the game Critical Mass.

Critical Mass is a game of position, territory, and material. It is a game of pure skill as are other games such as go, chess, checkers, and tic-tac-toe. It is a challenging game that requires a great deal of concentration to play well. It can also be a frustrating game as position, territory, and material can be drastically altered with a single move. The object of the game is to completely remove all of your opponent's pieces or bombs from the game board. There are several rules of the game. The board consists of a rectangular array of cells, five rows and six columns. 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. At the start of the game, all cells on the board contain zero pieces. There are two players, who make moves alternately. To move, a player must place a piece of 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 wave-like 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.

Platform Configuration

The first task of this project was to accumulate the parts in order to build a machine that could serve as a web server. The parts were all purchased from www.mwave.com and were recommended by an employee of Union Pacific Railroad. For a detailed listing of the parts that were purchased, see Appendix A.

Once all of the parts were purchased and gathered together, it was necessary to assemble them together to make the computer work. For a detailed step-by-step process of how the computer was assembled, see Appendix B. The machine was used for the majority of the development of the project, but the final live version of this application is hosted at IDEA lab.

Once the hardware is set up and seems to be working properly, it is necessary to first install the operating system, and then the rest of the software necessary to complete this project. For details instructions on how to install the software, see Appendices C, D, E, F, and G.

SECTION 5: SYSTEM DESCRIPTION

Once all the parts were assembled and the software components were installed, it was time to begin developing the web pages that would later become a dynamic game playing application. In order to start with a base in order to build on, the first step was to build a database to store information. Then, use HTML to develop the look and feel of the application.

Database Design

The database used for this project is entitled CriticalMass and contains two tables. The two tables can be visualized as the following:

Student											
USER_ID	PSWD	F_NAME	L_NAME	WINS	LOSSES	TIES	RANK	TOTAL_GAMES	FILE_NAME	PLAYING	LAST_CHAL

GamesPlayed				
PLYR_1	PLYR_2	G_RESULT	GAME_STRING	G_NBR

The first table, *Student*, is used for identification purposes, such as for the login page, and will be used for statistics, which will be used when students want to challenge each other. The field `PLAYING` is used to enforce the rule that a player can only be in one game at a time. The field `LAST_CHAL` is used to store the last time that a player challenged another player. This ensures that we can monitor how often students challenge other students. Currently, the delay is 1-2 minutes between challenges. The second table, *GamesPlayed*, is used for storing the games that have been played so that students can

see a listing of the games they have played and can chose to watch them after they have been played.

HTML Design

The layout of the web pages was modeled after the web site www.yahoo.com, and more specifically, the finance page. The first page that you are introduced to is the index page, hosted at www.cs.idealab5.uiowa.edu (see Figure 2), which uses an HTML image tag to include a picture of a Critical Mass game board. Once the user logs in to the application, they are presented with many options, including uploading code, playing games, and replaying games.

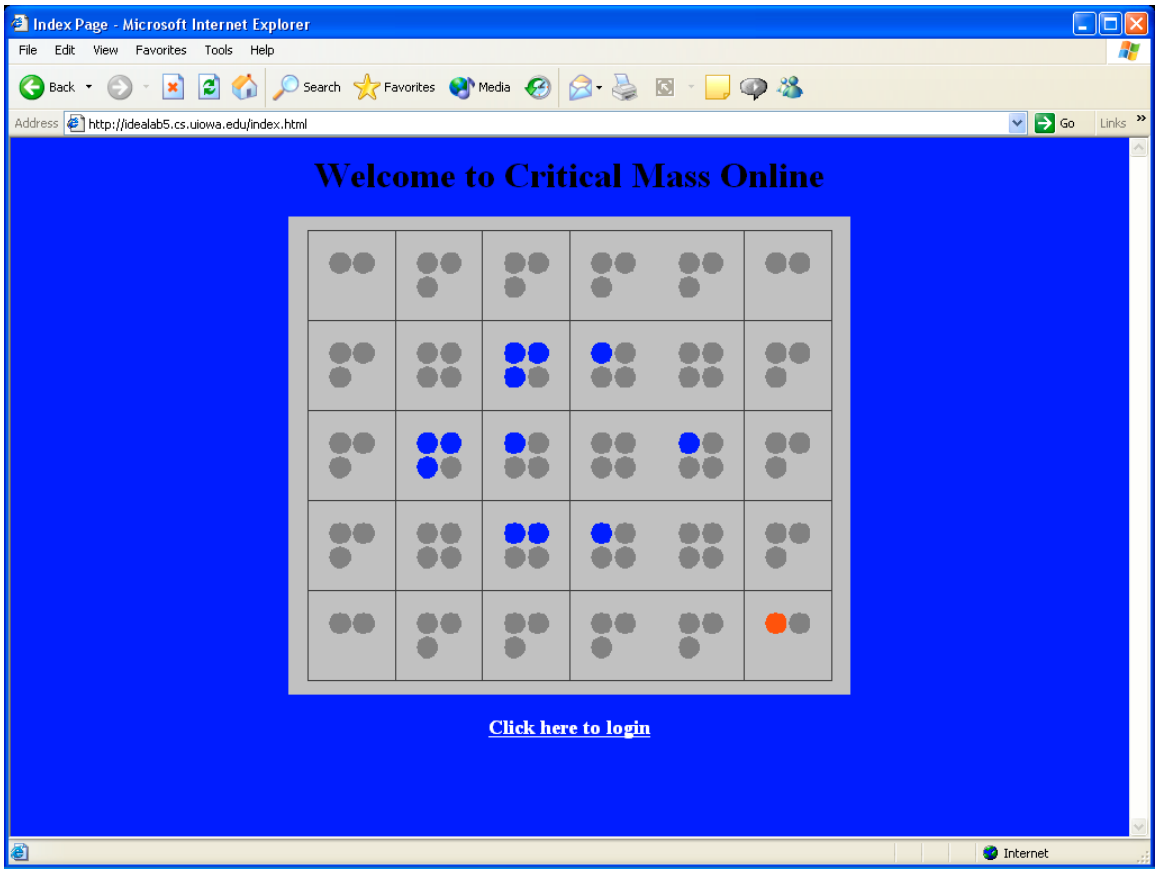

Figure 2: Index Page

Authentication

The user is prompted to click on a link in order to login to the website. In order to ensure that this application is secure, a PHP script is included at the top of every secure page so that a user cannot access the URL unless they are properly logged in. If a user does try to access a URL without being logged in, then they are redirected to the login page by the authentication script, `auth.inc`. The code for the authentication script was taken from Web Database Applications with PHP & MySQL book, as well as the code for the login and logout pages. The logic of the code for the authentication pages centers on using HTTP session variables and headers. HTTP session variables are variables that are created by registering them with the session, and then discarded as the user logs out by unregistering the variables. The headers are used for the redirects, which send the user to the login page if they are not already registered with the session.

When a user is taken to the login page, they get the first sight of the look and feel of the website (see Figure 3). If no session variables have been set yet, then the standard login page is displayed. Once the user properly enters their user id and password and clicks on the button to login, they are taken to the main page of the application. If the user login fails for whatever reason, the HTTP session variables "errorMessage" or "loginMessage" are set, and an error message is displayed to the user. Then, they are asked to login again. Examples of messages that could be displayed to the user include "Could not connect to the critical mass database under that user name" or "User *(user name here)* has logged out" or "You have not been authorized to access the URL *(URL they tried to access here)*." The logic used to correctly authenticate the user centers around looking up the username that the user entered and comparing the password they entered with the password in the database. If they match, they are authenticated, if they do not match, the user is prompted with an error message and asked to try again.

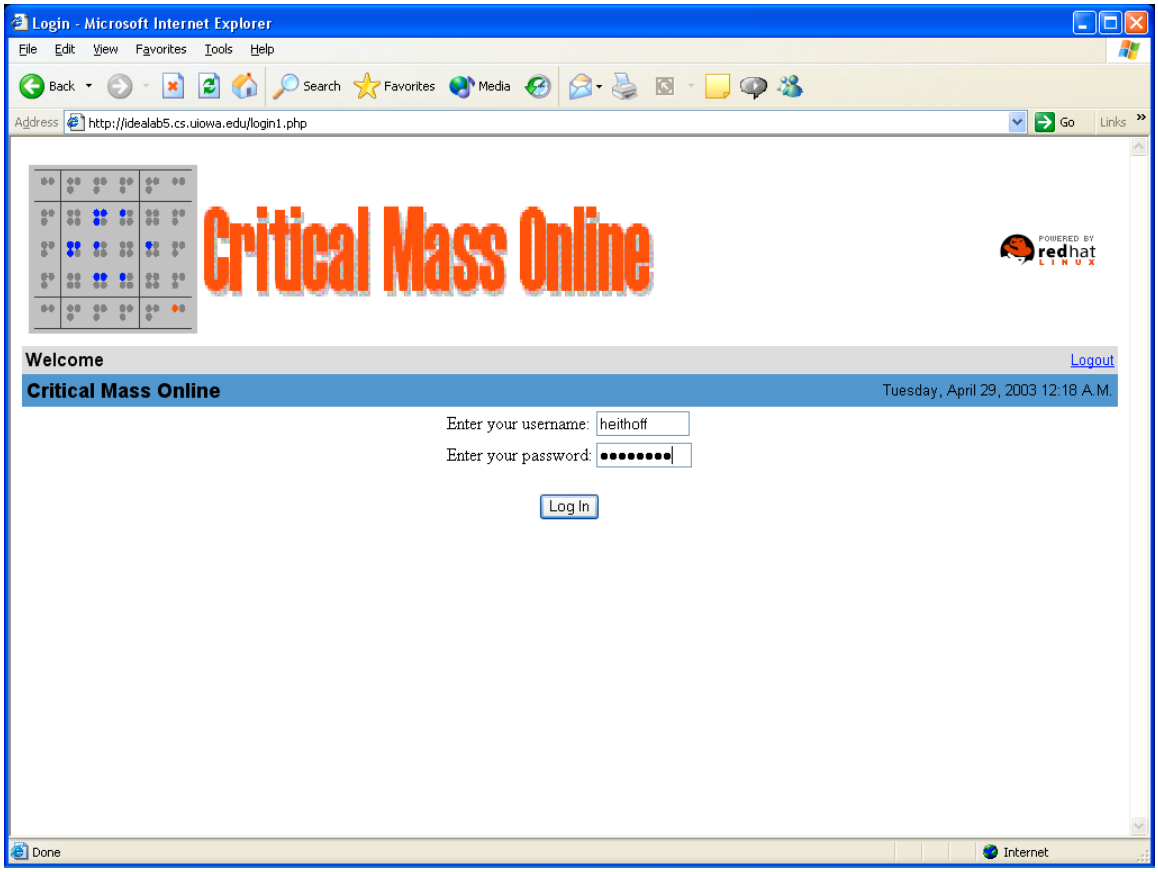

Figure 3: Login Page

Since PHP is used for the majority of the pages on this web site, headers allow both the login prompt and the main page to be in one file of code. Depending on the different cases, appropriate functions are called to display the necessary code. For instance, once the user's login information has been accepted, the function, `logged_on_page`, is called with the parameter "currentLoginName." The page displayed has five major components (see Figure 4). The first component is the top of the page. The user's name is displayed, along with graphics of the Critical Mass game board, and a link to the Red Hat Linux website. There is also a link to logout from the site, and the date and time is displayed.

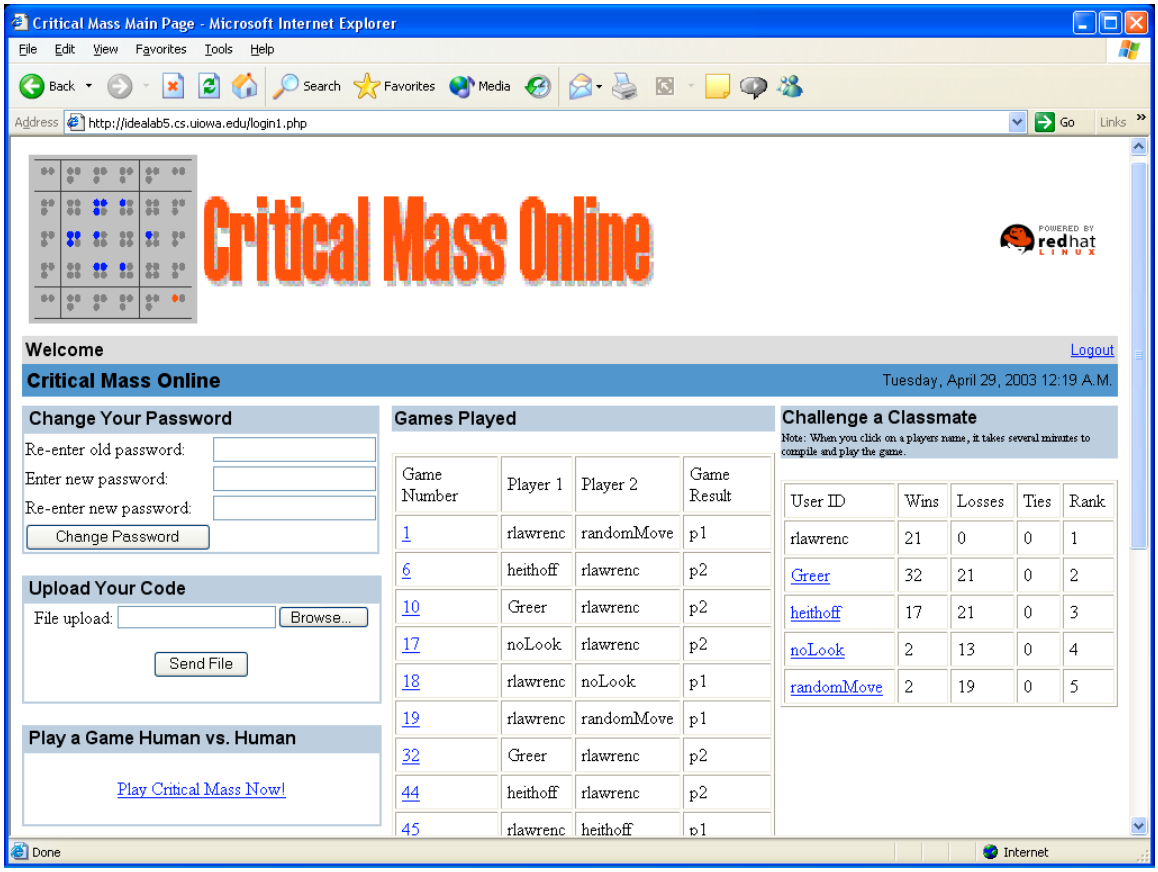

Figure 4: Main Page

The second component of the page is the change password functionality. This allows the user, at any time, to change their password. Since it is not necessary for this site to be high-security, there are no limitations set for the passwords for this site. The only limitation results from the way the database is set up, which requires that the password be 10 characters or less. The authentication process does require that your password is case sensitive. On the main page there is a form that asks the user for their old password, and to enter their new password twice. When the user clicks on submit, the form takes them to the next page `chpass.php`. This page connects to the database and compares the old password that the user entered with the password in the database. If the old password is correct, then it compares the two new passwords that the user entered. Again, if those match up correctly, then the database is updated with the new

password. For each case, messages are displayed to the user accordingly. At the bottom of the page, there is a link to go back to the main page to perform another task, try changing their password again, or to logout.

Student Code Upload

Uploading game code is the third task that a user can complete on this web site. This operation also uses a form, but includes a browse button that allows the user to search on their computer for the file they want to upload. Once they have found the file, it is automatically entered into the field, and the user just has to click on the send file button. Clicking on send file sends the user to another page, `codeUpload.php`. `CodeUpload.php` first connects to the database, and then if a connection is made, it attempts to upload the file specified and save it to a temporary directory. An important note is that the user is only allowed to upload files that are of type application/octet-stream, which includes files with the extension .h. If the user uploaded the correct type of file, then the file is copied to the student code directory where it is combined with a sample C++ program, `testStudentCode.cxx`, which verifies that their code compiles correctly. If the code has errors when it is compiled, then those errors are printed to the screen and the temporary file is deleted (see Figure 5). If there are no errors, then the file is kept in the directory.

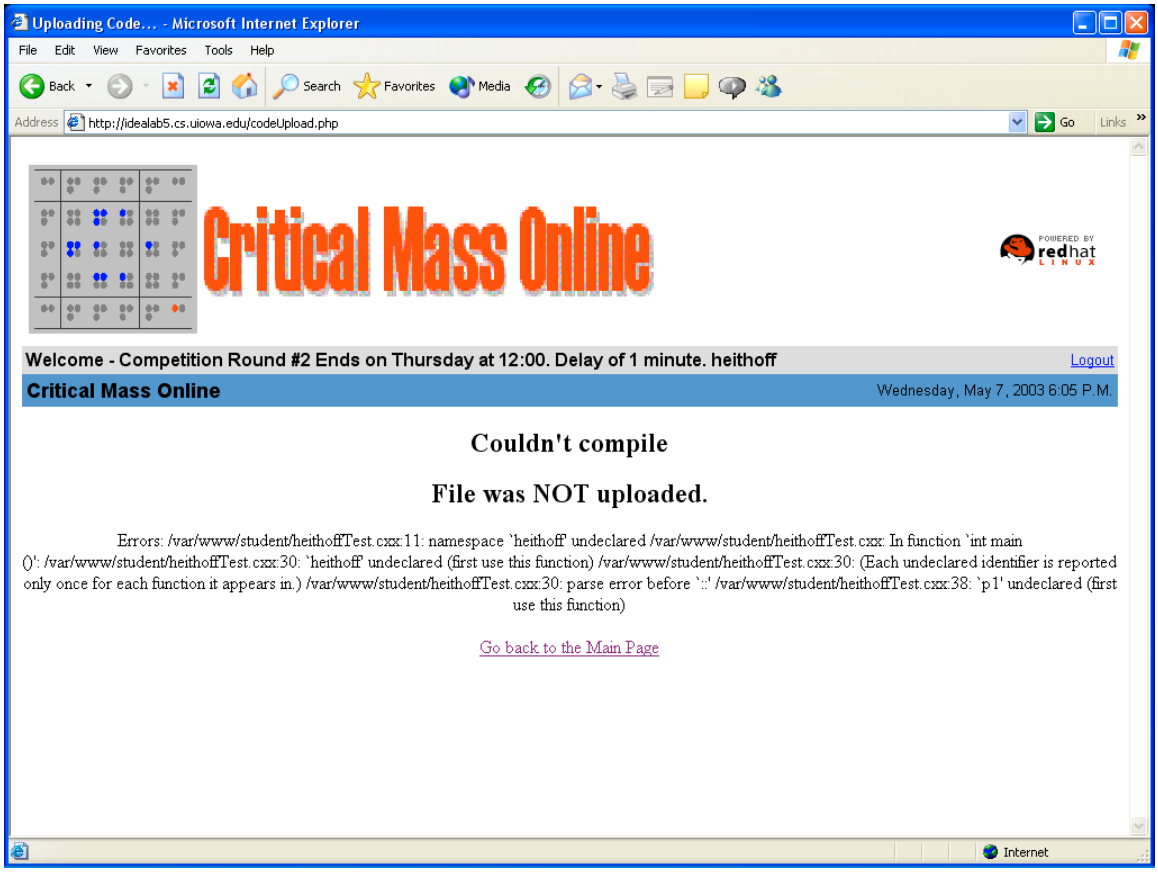

Figure 5: Code Upload Errors

Now is also when the user gets their initial rank in the tournament. If the user has not yet been assigned a rank, then they get a rank one higher than the highest rank given out so far. This is justified by the fact that the first student to upload their code should be given the rank number one. However, if the user has already been given a rank when they upload their code, then their rank just stays the same as it was. This way, if a user wants to make code changes and re-upload their code, they do not loose their spot in the tournament. This also ensures that there will always be one student who is ranked number one. If the user's code does not compile, then the errors that occurred are displayed for the user and the file that was uploaded is deleted. Similar to the change password operation, there is a link back to the main page on the bottom of `codeUpload.php`.

Playing Games

The final and most important component of this web site is the part that actually plays the games. The main page displays a table that lists users and their ranks. A player can challenge any player within 5 ranks of the user. This is done to ensure that the user does not challenge classmates that are ranked far below or above him or her. All user ids within playing range, other than the user who is currently logged in, are linked to the next page. All users are listed, except for those that have not yet uploaded their code. By clicking on a classmate's user id, the user is sent to the page `play.php`. `Play.php` first queries the database to make sure that the student is not currently involved in a game and then makes sure that they have waited the appropriate amount of time between challenges. Then `play.php` searches for the two player's code, compiles them with the `gameController` and then runs the `gameController`. Figure 6 shows the basic code for the `gameController`. See Appendix H for the complete `gameController`. Once the game is finished, the result is displayed (see Figure 7).

```
while (b.getStatus() ==0)
{     startclock = clock(); //Start sys clock
      row = -1; col = -1;
      temp = b;
      p1.makeMove(temp,1, row, col);

      endclock = clock(); //End sys clock
      if(endclock-startclock > timeAllowance)
      {     cout<< "PLAYER1 is taking too long to make moves!"<<
                  endl;
            cout << (endclock-startclock)/1000000 << " seconds"
                  << endl;
            winner = 2;
            break;
      }
      if(insertBomb(b,1, row, col))
      {
            moves++;
            std::string Srow = convertToString(row);
            std::string SCol = convertToString(col);
            gameString = gameString + Srow + SCol;
      }
      else
      {     winner = 2;
            cout << "Tried to insert invalidly player 1" << endl;
```

```
                cout << "Game string: " << gameString << endl;
                cout << "Invalid move attempted - row: " << row << "
                      col: " << col << endl;
                break;
          }

          if(b.getStatus()==0)
          {     startclock = clock(); //Start sys clock
                row = -1; col = -1;
                temp = b;
                p2.makeMove(temp,2,row,col);
                endclock = clock(); //End sys clock
                if(endclock-startclock > timeAllowance)
                {     cout<< "PLAYER2 is taking too long to make
                            moves!"<< endl;
                      cout << (endclock-startclock)/1000000 << "
                            seconds" << endl;
                      winner = 1;
                      break;
                }

                if(insertBomb(b,2, row, col))
                {
                      moves++;
                      std::string Srow = convertToString(row);
                      std::string SCol = convertToString(col);
                      gameString = gameString + Srow + SCol;
                }
                else
                {     winner = 1;
                      cout << "Tried to insert invalidly player 2" <<
                            endl;
                      cout << "Game string: " << gameString << endl;
                      cout << "Invalid move attempted - row: " << row
                            << " col: " << col << endl;
                      break;
                }
          }
    }
```

Figure 6: Game Contoller Main Loop

The `gameController` is a template written to put together two student's code. The `gameController` includes the `.h` files uploaded by the players. This code also creates the board, and uses namespaces to call methods from alternating student's code. As long as either player has not yet won the game, alternating players are asked for the move they want to make. Each player has only five seconds to choose their move. If the player exceeds five seconds when making a move, then the game is aborted, and there is

no winner. Once a move has been selected by the player, in order to actually insert the bomb, the `insertBomb` method within `CM.h` (a file included into `gameController`) is used. This ensures that the player's will not attempt to insert a bomb in an invalid location. As the game is being played, a game string is created.

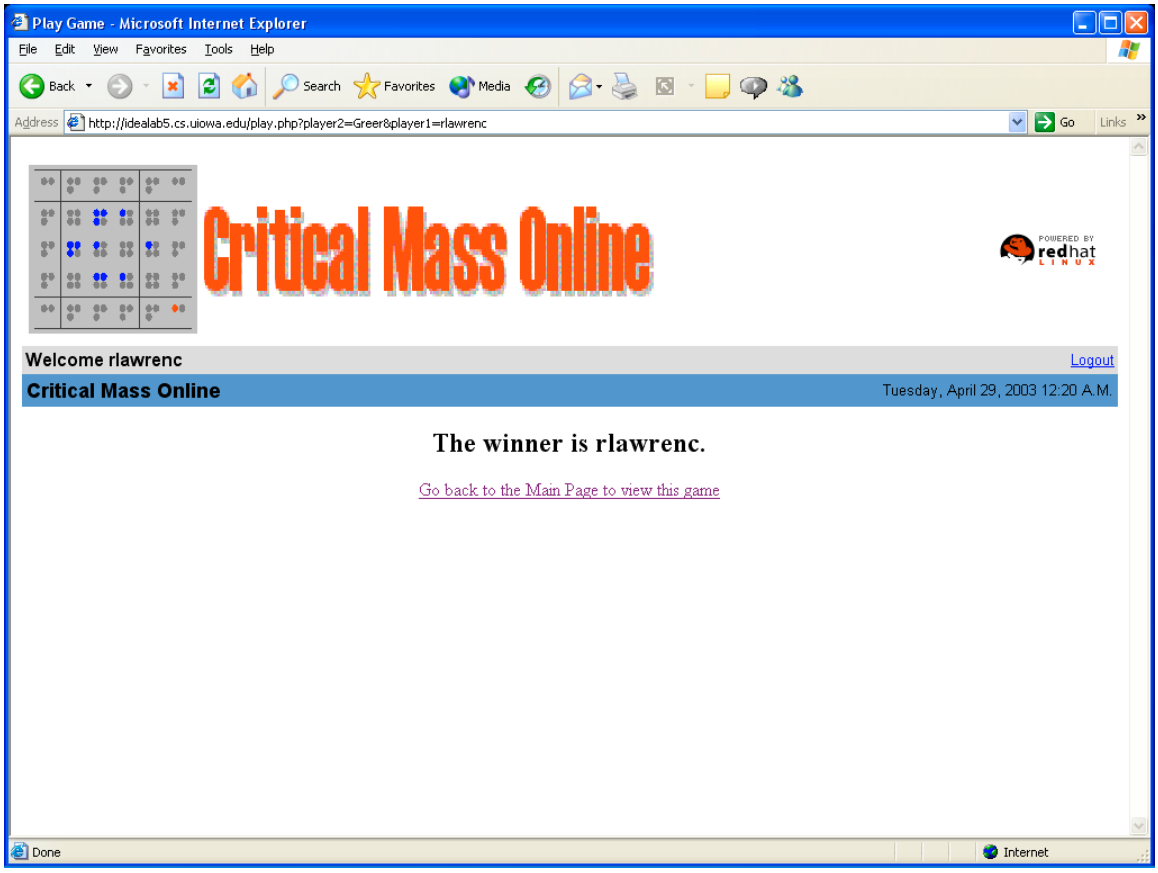

Figure 7: Play Page

Once a winner has been determined, the C++ code connects to the database using MySQL++. Once a connection is made, the player's names, the outcome of the game, and the game string are inserted into the database. Also, it is at this point where the rank of the two players is changed. If the winner has a lower rank than the loser, then they take over the loser's rank and the loser's rank is decreased by one. Also, all players

whose rank was lower than the loser and greater than the winner's initial rank get their rank decreased by one. This concludes the process of actually playing the game. See Appendix H for the complete game controller.

If the `gameController` properly compiles and runs, then a message is displayed to the user and they are able to follow a link back to the main page, where they can watch the game they just played. A necessity for the `gameController` to work is that the students upload code with the appropriate methods. Since `gameController` uses a template to call the student's code for the next move, all of the student's must follow a strict outline for their program. Certain methods must be included in the student's `.h` file and must produce the desired result. The primary and most important method is `makeMove()`. The method `makeMove()` is called by the game controller to get the next move from the player. `makeMove()` should return the row and column where the next move should be made. In order for the student to determine this row and column, they will need to write methods such as `insertBomb()` and `evaluate()` which would give them valuable information in order to make the decision of where to make the next move. While it is necessary for the student to write these methods for their own use, the game controller is only concerned with the student having a method which returns the move to be made. The Java applet and the game controller have their own versions of `insertBomb()` and `evaluate()` which are guaranteed to be correct.

Visualizing Games Played

The fourth functionality of the web site is the ability to save and replay games that are completed. This allows the users to see exactly where their code went wrong if they lost their game, or what their code is doing right if they won the game. On the main page, a table is displayed, listing all of the games that the user has played. This includes games where they challenged a classmate, and also games they had been challenged to play. If the user clicks on the appropriate game number, they are taken to a page that displays a game board. The front-end page for this is `GResult.php`. `GResult.php`

connects to the database to lookup the game string associated with the game number on which the user clicked. The game string is a list of moves in the form "row column." An example string is "0012001200." This can be translated as player one plays in row 0, column 0. Then player two makes a move in row 1, column 2. Player one now makes another move in row 0, column 0, and so on. Once the last move of the games string is played, the game is over. The game string is inserted into the database by the game controller at the end of the game. If the game string is found, then a Java applet is called and passed the game string as a parameter. The Java applet is what actually displays the game board and plays the game (see Figure 8).

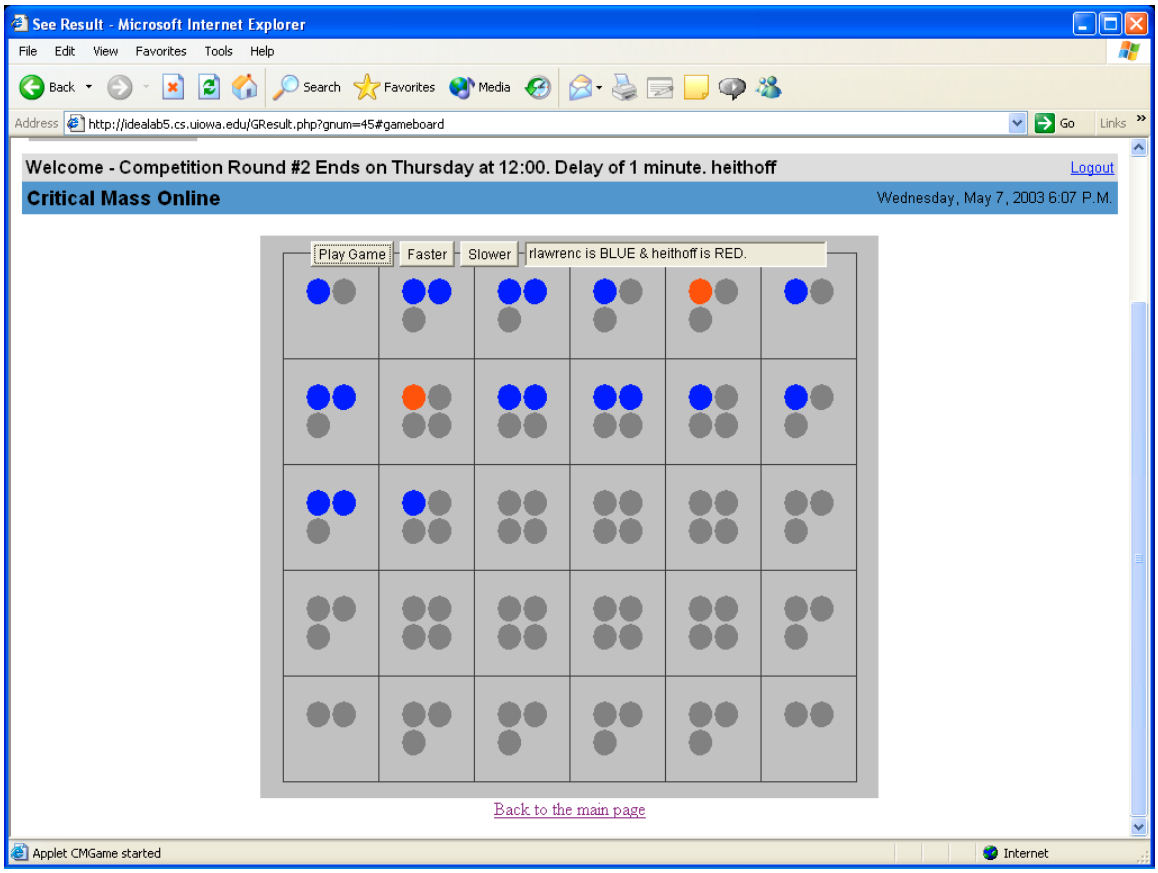

Figure 8: Java Applet

The Java applet uses a Graphical User Interface (GUI), to visually display a game. The applet reads in the game string, extracts one move from the string at a time, and displays the moves onto the screen. Each move is displayed on a timed interval, so that the user can easily watch the game being played. Displaying the moves without a timer would display the moves so fast that all that the user would actually see would be the final board. The applet supports aspects similar to the student's code, such as inserting a bomb, and exploding the pieces. In order to make sure that the user is ready to view the entire game, there is a button on the top of the game board. The game is not started until the user clicks that "Start Game" button. Once the user is done viewing their game, there is a link on the bottom of the page to take the user back to the main page.

SECTION 6: RESULTS AND USAGE

The application was first launched when the Critical Mass project was assigned to the students of Computer Science III. The students were given a listing of the game rules, an outline for the code they were to write, and some challenges. The challenges centered on the online application. First, they were challenged to upload their code to the website as soon as they could, which would give them a higher initial ranking than those who uploaded their code after them. Next, the students were informed of goals that they were to meet in order to get appropriate grades. If the student could get their code uploaded to the web server and play any game, they were awarded 25 points. If the student could do the previous and beat the code of randomMove, then they were awarded 35 points. In a similar manner, if the student beat the code of noLook, Heithoff, or rlawrenc, they were awarded 45, 50, and 60 points respectively. The code for randomMove simply used a random number generator to select a move. The code for noLook used level two intelligence and no game tree. The code for Heithoff used a game tree of depth five. rlawrenc also used a game tree of depth five, but had a better board evaluator than Heithoff. Students also got bonus marks for building a game tree, and for being ranked in the top ten.

During the implementation of this application, several problems were encountered. First of all, some student's code had infinite loops, and their code had to be manually terminated. Also, if a student had segmentation faults in their code, then no result would be displayed to the user and it was hard to track down this problem. See the section Future Work for more information.

After being online for two weeks, the application hosted over 5,000 games. Some students in the course played over 500 games. As measured by the student survey, the application was a positive experience for those involved. All 42 students surveyed either agreed or strongly agreed that the assignment was interesting. Further, almost 75% strongly agreed that it was challenging as well. The Critical Mass assignment was the

best assignment in the class according to over 78% of the students, and furthermore, it was the best assignment in any computer science course for 81% of the students. Eighty-nine percent of the students felt that this assignment increased their interest in the course material. This assignment helped to make 75% of the students better programmers. The tournament feature of the assignment motivated more effort into doing the assignment from 88% of the students, and 90% of the students felt the tournament feature made the project more interesting than a stand-alone game with no student competitions. Eighty-three percent of the students found the website user-friendly and over 40% of the students logged on to the site more that 20 times.

Comments from the students included suggestions to do things which are mentioned in the section, Future Work, such as "find a way to interrupt games that have infinite loops," "allow one to watch the progress of their current game while they are playing," and "have the ranking system based off of points, and not by whole won the last game at that spot." However, the majority of students only had positive things to say, such as "If all assignments were like this, I would learn so much more, this kind of programming project is the perfect combination of fun mixed with knowledge," "This was possibly the most interesting project I have ever worked with in my career at Iowa. It has definitely sparked my interest to learn more about game intelligence programming in the future," "I really liked the challenge of this assignment and the opportunity to have a chance to put my code up against others," and "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 that 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." For a complete listing of the survey, see appendix H.

SECTION 7: CONCLUSION

In order for a programmer to be successful, it is useful to fully understand the architecture on which programs run. This project provided the opportunity to completely create a program, from building the computer to installing the software that it runs on. Most importantly, this project provided a teaching aide for Computer Science III and C++, which may be used for years to come. It helped to teach students the effectiveness and efficiency of their programming in an interesting way.

SECTION 8: FUTURE WORK

As with most projects, many other additions to the application were sought after. There are several more features that could be added to this application to make it more functional and more appealing. First of all, is would be useful for students to be able to visualize the game as it is being played, computer versus computer, in real-time. A goal that is somewhat in sync with the real-time computer versus computer feature is the ability to play computer versus human in real-time. The Java applet could also be improved to include a mp3 player-type slider which would allow students to "rewind" or "fast-forward" through a game. Also, the GUI could be improved so that it is more aesthetically pleasing. In order to make this application last, it would be necessary to make it more versatile so that it could be used for other games as well. Lastly, in order to make this application more "user-friendly," it would be beneficial to improve the timer in the game controller that regulates how long students have to select a move. Currently, that task is done after the fact, creating a problem, especially if the student's code goes into an infinite loop.

Some challenges that were encountered when attempting these goals that are now part of "future work" mostly centered on the ability to have the Java applet running and communicating to the C++ program. Since the applet is started with HTML, and the C++ program is run by command line, this created a problem of how to get the move selected from the C++ program to the web browser. The task of turning Critical Mass Online into a web site for another game would be slightly less challenging. This would include mostly changing the Java applet to be the correct board, using the correct C++ code for the new game, and changing the logos. While researching to find a method to improve the timer for the game controller, results were found that indicated that interrupts would need to be used at the hardware level in order to accomplish this task. This would be a tedious task and would require someone who is an expert with C++ interrupts.

REFERENCES

[1] Peter Wainwright. Professional Apache 2.0. Wrox Press Ltd, Birmingham, UK, May 2002.

[2] Ben Laurie & Peter Laurie. Apache – The Definitive Guide. O'Reilly, Sebastopol, CA, February 1999.

[3] Christopher Negus. Red Hat Linux Bible. IDG Books Worldwide, Foster City, CA, January 1999.

[4] George Reese, Randy Jay Yarger, & Tim Kind. Managing & Using MySQL. O'Reilly, Sebastopol, CA, April 2002.

[5] Hugh E. Williams & David Lane. Web Database Applications with PHP & MySQL. O'Reilly, Sebastopol, CA, March 2002.

[6] Elizabeth Castro. HTML For the World Wide Web. Peachpit Press, Berkeley, CA, 2000.

APPENDIX A: ACCUMULATING THE PARTS

- **Motherboard:** ASUS A7V333
- **Monitor:** STYLEPRO Series 17 FD Trinitron CRT
- **Video Card:** Verto GeForce4 MX 420 AGP 64MB DDR SDRAM
- **Modem:** Diamond SupraMax V.92 56K PCI W/ Voice Internal (OEM)
- **Case Fan:** MWAVE 80mm ball bearing fan for case w/ 4 pin connector
- **Network Card:** Netgear FA311TX 10BT/100BTX PCI RJ45
- **Case:** Audi XP
- **CD-ROM drive:** Sony 52X IDE
- **Processor:** AMD Athlon XP Processor
- **Floppy drive:** Mitusmi 1.44 Floppy
- **Power Supply:** Antec SL350 350W
- **Hard drive:** Maxtor 60.0 GB DX6L060 EIDE ULTRA-ATA 133 8.5NS 7200RPM 2MB Buffer
- **RAM:** Corsair CM64SD256-2700CX2H 32x64 333MHZ 256MB CAS2 DDR DIMM W/ Heat Spreader

APPENDIX B: ASSEMBLING THE PARTS

1. Find the small metal stands for motherboard. The purpose of the stands is to attach the motherboard to the case, while keeping the two from touching. It is important to spread these out well, because they keep the motherboard from bending when you are installing things to it. Put some stands around the edges as well as in the middle of the motherboard.
2. Attach the motherboard onto the stands. Use non-magnetic tools and wear a static guard wristband (plug into wall) when in contact with the motherboard, memory, and processor. You should really always use it, but in those cases especially.
3. Put the memory into slot one of the memory slots. The memory should click in easily if you rock it from side to side.
4. Put processor in by lifting out and up on the lever, then matching up pin 1 with the correct corner. The processor should just fall right in; you should not have to press down. Once it is in, you will have to push pretty hard to get the lever back down.
5. To put the fan on top of the CPU, remove the sticky seal, then hook the one end onto the clips (you may have to remove the memory in order to do this), and then stretch it over to the clip on the other side. You might have to use a flathead screwdriver to push down the other side. Be careful not to slip the screwdriver off the clip and hit the motherboard.
6. Plug in the CPU fan and the power supply fan into the motherboard. These are wires with white plugs on the end. They plug into the motherboard as labeled.
7. Put in the floppy drive by sliding it into the appropriate slot from within the case. Make sure it matches up by looking at the front of the case.
8. Put in the hard drive by sliding it into the appropriate slot from within the case. This should go into a slot that does not open from the front. Set it as the master.

9. Put in the CD-ROM from the front of the case. You will have to take off the front first. It should slide in and pop into place. Set it as the master.
10. Put power supply in. Plug it into the motherboard and into all the drives. The big fat collection of wires plugs into the motherboard and the smaller sets of wires with several plugs on them plug into the drives. Plug the CD-ROM and the floppy into the same set of cords and the hard drive and front fan (later) into the same set of cords.
11. Plug IDE cables into drives and motherboard. The red wire should always be on the same side as the power plugs in the drives. The least wide cable is for the floppy drive. The cable with the most (smallest) wires is for the hard drive, and the other one is for the CD-ROM. The floppy plugs in away from the hard drive cable and the CD-ROM cable, which are right by each other.
12. Take the front cover off and screw in the front fan. Make sure that the airflow is pointing into the case. Plug the fan into the power supply (previously stated).
13. Plug wires from the case into the Panel (lower right-hand corner of motherboard). Color usually represents +5V and black or white usually represents ground.
14. Plug in USB port cables to motherboard. The USB 1.1 cables plug into the motherboard where it says USB2_3, using the rules from above and the manual. We could not find a place to plug in the USB 2.0 cables.
15. Put in the video card, modem, network card, and 2-port USB bracket. They go in the slots near the back of the case, and they should be screwed on. You will have to pop out the cut outs in the back of the case for the ports to show.
16. Plug the audio cable from the CD-ROM to the motherboard.

APPENDIX C: INSTALLING THE LINUX OPERATING SYSTEM

1. Insert the 1st Linux disc
2. Restart your computer
3. The CD-ROM will automatically start up, prompting you to press enter at the "Welcome to Red Hat Linux" screen.
4. Choose what language you want to install Linux in.
5. Choose what keyboard type you have.
6. Choose the Workstation configuration.
7. Allow Linux to partition the disk automatically.
8. Choose to install all software components.
9. Wait approximately 30 minutes for the software installation to complete. You will probably have to insert disc 2 at this point.
10. Choose what mouse you have and if you are connected to a Local Area Network.
11. Have the software create a boot floppy disk.
12. Choose the appropriate video card and monitor.
13. Take out the installation disc and reboot the computer.

APPENDIX D: INSTALLING APACHE

1. Download source code for apache off of www.apache.org.
2. Put the source code in the directory /usr/local/
3. cd /usr/local
4. gunzip httpd-2.0.39.tar.gz
5. tar xvf httpd-2.0.39.tar
6. cd httpd-2.0.39
7. ./configure –prefix=/www –enable-module=so
8. make
9. make install
10. /www/bin/apachectl start
11. Go to http://localhost/ to view the sample Apache page to make sure you installed correctly.

APPENDIX E: INSTALLING MYSQL

1. Download the following RPM files for MySQL into /tmp
    a. MySQL-3.23.52-1.i386.rpm
    b. MySQL-client-3.23.52-1.i386.rpm
    c. MySQL-devel-3.23.52-1.i386.rpm
    d. MySQL-shared-3.23.52-1.i386.rpm
    e. MySQL-bench-3.23.52-1.i386.rpm (needs Perl, I couldn't install)
2. Use these commands to install these
    a. rpm –i /tmp/MySQL-3.23.52-1.i386.rpm
    b. rpm –i /tmp/MySQL-client-3.23.52-1.i386.rpm
    c. rpm –i /tmp/MySQL-devel-3.23.52-1.i386.rpm
    d. rpm –i /tmp/MySQL-shared-3.23.52-1.i386.rpm
    e. rpm –i /tmp/MySQL-bench-3.23.52-1.i386.rpm
3. In order to connect to the database directly enter the command
    a. mysql –u root –p
    b. Enter password: (password starts out as blank if you don't change it)
4. Once you are in the mysql prompt, you can begin to create your database

```
CREATE DATABASE CriticalMass;
USE CriticalMass;
CREATE TABLE Student (
   USER_ID   CHAR(10)  NOT NULL  PRIMARY KEY,
   PSWD           CHAR(10),
   F_NAME         CHAR(25),
   L_NAME         CHAR(40),
   WINS           INT,
   LOSSES         INT,
```

```
   TIES           INT,
   RANK           INT,
   TOTAL_GAMES    INT,
   FILE_NAME      CHAR(30)
   PLAYING        CHAR(10)
   LAST_CHAL      DATETIME  );
CREATE TABLE GamesPlayed(
   PLYR_1    CHAR(10),
   PLYR_2    CHAR(10),
   G_RESULT  CHAR(2),
   GAME_STRING    TEXT,
   G_NBR     INT  NOT NULL  PRIMARY                KEY
        AUTO_INCREMENT);
```

APPENDIX F: INSTALLING PHP

1. Download the source code from www.php.net and put into /usr/local/
2. cd /usr/local/
3. gunzip php-4.2.3.tar.gz
4. tar xvf php-4.2.3.tar
5. cd php-4.2.3
6. ./configure –with-mysql –with-apxs2=/www/bin/apxs
7. make
8. make install

Problems encountered while installing php

1. In order to get php version 4.2.3 to work with Apache 2.0.39 you must do the following:
    a. In php_functions.c, comment out the following bit of code
        i. /* #if !MODULE_MAGIC_AT_LEAST(20020506,0)
            ADD_STRING(boundary);
            #endif */
    b. Also in php_functions.c, you need to check under the method php_register_hook for:
        i. ap_register_output_filter(“PHP”, php_output_filter, NULL, AP_FTYPE_RESOURCE);
            1. Remove the entire 3rd argument; Remove NULL from the parameters.
            2. If NULL is not removed, this will generate an error when using “MAKE” & “MAKE INSTALL”

ii. ap_register_input_filter("PHP", php_input_filter, NULL, AP_FTYPE_RESOURCE);

1. Remove the entire 3rd argument; Remove NULL from the parameters.
2. If NULL is not removed, this will generate an error when using "MAKE" & "MAKE INSTALL"

c. In httpd.conf,

i. Instead of "AddType application/x-httpd-php .php"

ii. Put <Files *.php>

SetOutputFilter PHP

SetInputFilter PHP

</Files>

APPENDIX G: INSTALLING MYSQL++

1. Download and install zlib (www.gzip.org/zlib/)
2. Download and install MySQL++ v.1.7.9 source distribution
    a. ./configure
    b. make
    c. make install
3. Libraries go to /usr/local/lib when installed
4. Compile line for a C++ program using MySQL++:
    a. g++ -Wall test.cxx –o test –I /usr/include/mysql –L /usr/local/lib –lsqlplus
5. In C++ program:
    a. #include <sqlplus.hh>
6. Problem: When running test.cxx, and error was occurring that stated that it could not find file.
7. Solution: Edit /etc/ld.so.conf by adding the line "/usr/local/lib" and run the program ldconfig.

APPENDIX H:

GAMECONTROLLERTEMPLATE.CXX

```
#include "../student/PLAYER1.h"
#include "../student/PLAYER2.h"
#include "CM.h"
#include <iostream>
#include <cstdlib>
#include <sstream>
#include <string>
#include <stdio.h>
#include <sqlplus.hh>
#include <custom.hh>
#include <time.h>

using namespace PLAYER2;
using namespace PLAYER1;
using namespace std;

// Forward declarations
bool insertBomb(CMboard &bd, int p, int r, int c);
void fixOverflow(CMboard &bd, int p, int r, int c);


std::string convertToString(int x)
{
   std::ostringstream o;
   if (o << x)
     return o.str();
   // some sort of error handling goes here...
   return "conversion error";
}

// Ultimate authority code for inserting bomb in game board
bool insertBomb(CMboard &bd, int p, int r, int c)
{
        if (r < 0 || r >= bd.getMaxRow() || c < 0 || c >=
                  bd.getMaxCol())
                return false;

        if( p != bd.getPlayer(r,c) && bd.getPlayer(r,c) != 0 )
                return false;

        int pieces = bd.getNumPieces(r,c);
        bd.setNumPieces(r,c,pieces+1);
        bd.setPlayer(r,c,p);
        if (pieces+1 >= bd.getMaxPieces(r,c))
                fixOverflow(bd, p,r,c);

        return true;
}

void fixOverflow(CMboard &bd, int p, int r, int c)
{
        queue<CMsquare*> q;
        q.push(bd.getSquare(r, c));
```

```
        while(!q.empty())
        {
                if (bd.getStatus() != 0)
                        break; // Chain of explosions killed other plyr
                               // Prevent possible infinite loop

                if(q.front()->bombs >= q.front()->max_bombs)
                  //must explode the square
                {
                        q.front()->bombs = 0;
                        q.front()->player = 0;
                        //find where the exploded bombs move to..
                        if (q.front()->row < bd.getMaxRow()-1)
                        //at row before last row
                        {
                                q.push(bd.getSquare(q.front()->row+1,
                                    q.front()->col));
                                ++q.back()->bombs;
                                q.back()->player = p;
                        }
                        if (q.front()->col < bd.getMaxCol()-1)
                        {
                                q.push(bd.getSquare(q.front()->row,
                                    q.front()->col+1));
                                ++q.back()->bombs;
                                q.back()->player = p;
                        }
                        if (q.front()->row > 0)
                        {
                                q.push(bd.getSquare(q.front()->row-1,
                                    q.front()->col));
                                ++q.back()->bombs;
                                q.back()->player = p;
                        }
                        if (q.front()->col > 0)
                        {
                                q.push(bd.getSquare(q.front()->row,
                                    q.front()->col-1));
                                ++q.back()->bombs;
                                q.back()->player = p;
                        }
                }
                q.pop();
        }
}

int main()
{
      int timeAllowance = 5000000;
      //number of milliseconds allowed per move = 5 seconds
      int verybeg, veryend, startclock, endclock;
      string win;
      string gameString = "";
      PLAYER2::CMGame p2;
      PLAYER1::CMGame p1;
      int row, col;
      int winner = 0;
```

```
int moves = 0;
verybeg = clock();
CMboard b(5,6), temp;

while (b.getStatus() ==0)
{     startclock = clock(); //Start sys clock
      row = -1; col = -1;
      temp = b;
      p1.makeMove(temp,1, row, col);

      endclock = clock(); //End sys clock
      if(endclock-startclock > timeAllowance)
      {     cout<< "PLAYER1 is taking too long to make moves!"<<
                  endl;
            cout << (endclock-startclock)/1000000 << " seconds"
                  << endl;
            winner = 2;
            break;
      }
      if(insertBomb(b,1, row, col))
      {
            moves++;
            std::string Srow = convertToString(row);
            std::string SCol = convertToString(col);
            gameString = gameString + Srow + SCol;
      }
      else
      {     winner = 2;
            cout << "Tried to insert invalidly player 1" << endl;
            cout << "Game string: " << gameString << endl;
            cout << "Invalid move attempted - row: " << row << "
                  col: " << col << endl;
            break;
      }

      if(b.getStatus()==0)
      {     startclock = clock(); //Start sys clock
            row = -1; col = -1;
            temp = b;
            p2.makeMove(temp,2,row,col);
            endclock = clock(); //End sys clock
            if(endclock-startclock > timeAllowance)
            {     cout<< "PLAYER2 is taking too long to make
                        moves!"<< endl;
                  cout << (endclock-startclock)/1000000 << "
                        seconds" << endl;
                  winner = 1;
                  break;
            }

            if(insertBomb(b,2, row, col))
            {
                  moves++;
                  std::string Srow = convertToString(row);
                  std::string SCol = convertToString(col);
                  gameString = gameString + Srow + SCol;
            }
            else
```

```
                {     winner = 1;
                      cout << "Tried to insert invalidly player 2" <<
                            endl;
                      cout << "Game string: " << gameString << endl;
                      cout << "Invalid move attempted - row: " << row
                            << " col: " << col << endl;
                      break;
                }
          }
     }
     if (winner == 0)
          winner = b.getStatus();
     if(winner ==2)
     {    cout << "The winner is PLAYER2. " << endl;
          win = "p2";
     }
     else if(winner ==1)
     {    cout << "The winner is PLAYER1." << endl;
          win = "p1";
     }


     int onewins, onelosses, onerank, onegames;
     int twogames, twowins, twolosses, tworank;
     int maxrank;

     Connection con("CriticalMass","localhost","root","");
     //Insert game data into database
     Query query = con.query();
     query << "INSERT INTO GamesPlayed (PLYR_1, PLYR_2, G_RESULT,
          GAME_STRING) VALUES ('PLAYER1','PLAYER2' "<< "," <<"'" <<
          win <<"'"<< "," <<"'"<< gameString << "'"<<")";
     query.execute();

     //find the maximum rank
     Query query0 = con.query();
     query0 << "Select MAX(RANK) from Student";
     Result res = query0.store();
     Row rowInfo0;
     Result::iterator k;
     for (k = res.begin(); k != res.end(); k++)
     {
          rowInfo0 = *k;
          maxrank = rowInfo0["MAX(RANK)"];
     }


     //Obtain player one's information
     Query query2 = con.query();
     query2 << "Select WINS, LOSSES, RANK, TOTAL_GAMES FROM Student
          WHERE USER_ID = 'PLAYER1'";
     res = query2.store();
     Row rowInfo;
     Result::iterator i;
     for (i = res.begin(); i != res.end(); i++)
     {    rowInfo = *i;
          onewins = rowInfo["WINS"];
          onelosses = rowInfo["LOSSES"];
```

```
      onerank = rowInfo["RANK"];
      onegames = rowInfo["TOTAL_GAMES"];
}
//Obtain player two's information
query << "Select WINS, LOSSES, RANK, TOTAL_GAMES FROM Student
      WHERE USER_ID = 'PLAYER2'";
res = query.store();
Row rowInfo2;
Result::iterator j;
for (j = res.begin(); j != res.end(); j++)
{     rowInfo2 = *j;
      twowins = rowInfo2["WINS"];
      twolosses = rowInfo2["LOSSES"];
      tworank = rowInfo2["RANK"];
      twogames = rowInfo2["TOTAL_GAMES"];
}

// Player 1 is the challenger
// Player 2 is the program the was challenged

// P1 challenged a lower ranked player P2 and won - do not update
      any rankings
if(winner == 1 && onerank < tworank)
{
      query << "Update Student Set WINS = (" << onewins +1 << ")
            , TOTAL_GAMES = (" << onegames +1 << ") WHERE USER_ID
            = 'PLAYER1'";
      query.execute();

      query << "Update Student Set LOSSES = (" << twolosses +1 <<
            "), TOTAL_GAMES = (" << twogames +1 << ") WHERE
            USER_ID = 'PLAYER2'";
      query.execute();

}

// P1 challenged a higher ranked player p2 and won - update
      rankings
else if(winner == 1 && onerank >= tworank)
{
      query << "Update Student Set WINS = (" << onewins +1 << ")
            , TOTAL_GAMES = (" << onegames +1 << "), RANK = (" <<
            tworank << ") WHERE USER_ID = 'PLAYER1'";
      query.execute();

      //update all ranks greater than that of the loser and less
            than winner
      query << "UPDATE Student SET RANK = (RANK + 1) WHERE RANK
            >= " << tworank + 1 << " AND RANK <= " << onerank <<
            endl;
      query.execute();

      query << "Update Student Set LOSSES = (" << twolosses +1 <<
            "), TOTAL_GAMES = (" << twogames +1 << "), RANK = ("
            << tworank + 1 << ") WHERE USER_ID = 'PLAYER2'";
       query.execute();
}
```

```
	// P1 challenged a higher ranked player and lost - no update to
		rankings
	else if(winner == 2 && onerank >= tworank)
	{
		query << "Update Student Set WINS = (" << twowins +1 << "),
			TOTAL_GAMES = (" << twogames +1 << ") WHERE USER_ID =
			'PLAYER2'";
		query.execute();
		query << "Update Student Set LOSSES = (" << onelosses +1 <<
			"), TOTAL_GAMES = (" << onegames +1 << ") WHERE
			USER_ID = 'PLAYER1'";
		query.execute();

	}

	// P1 challenged a lower ranked player and lost - update rankings
		by switching places between two
	else if(winner == 2 && onerank < tworank)
	{
		query << "Update Student Set WINS = (" << twowins +1 << "),
			TOTAL_GAMES = (" << twogames +1 << "), RANK = (" <<
			onerank << ") WHERE USER_ID = 'PLAYER2'";
		query.execute();
		query << "Update Student Set LOSSES = (" << onelosses +1 <<
			"), TOTAL_GAMES = (" << onegames +1 << "), RANK = ("
			<< tworank << ") WHERE USER_ID = 'PLAYER1'";
		query.execute();
	}
	veryend = clock();
	//cout << "Total time: " << veryend-verybeg << endl;

	return EXIT_SUCCESS;
}
```

APPENDIX I: STUDENT SURVEY

1. The Critical Mass assignment was an interesting assignment.
    a. Strongly Agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly Disagree
2. The Critical Mass assignment was a challenging assignment.
    a. Strongly Agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly Disagree
3. The Critical Mass assignment was the best assignment in this class.
    a. Strongly Agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly Disagree
4. The Critical Mass assignment was one of the best assignments so far in any computer science course.
    a. Strongly Agree
    b. Agree
    c. Neutral
    d. Disagree
    e. Strongly Disagree
5. The Critical Mass assignment increased my interest in the course material.

a. Strongly Agree

b. Agree

c. Neutral

d. Disagree

e. Strongly Disagree

6. Game trees and game-related projects make the data structures course more interesting.

a. Strongly Agree

b. Agree

c. Neutral

d. Disagree

e. Strongly Disagree

7. The tournament feature of this assignment motivated more effort into doing the assignment.

a. Strongly Agree

b. Agree

c. Neutral

d. Disagree

e. Strongly Disagree

8. The tournament feature of the assignment is more interesting than developing a stand-alone game with no student competitions.

a. Strongly Agree

b. Agree

c. Neutral

d. Disagree

e. Strongly Disagree

9. The design of the game website is user-friendly.

a. Strongly Agree

b. Agree

c. Neutral

d. Disagree

e. Strongly Disagree

10. Approximately how many times did you log into the game site?

a. 1-5

b. 6-10

c. 11-15

d. 16-20

e. 20 or more

11. Approximately where did you rank against your classmates?

a. 1-10

b. 11-20

c. 21-30

d. 31-40

e. 4 or above

12. What is your expected grade in this course?

a. A

b. B

c. C

d. D

e. F

13. Please enter any general comments about the assignment and the Critical Mass website here.