

**EE4023 – Distributed Systems**

**Project Report – Tic-Tac-Toe**

**Group name: *Willbo Baggins***

William O'Leary - 15155528

Aaron Dunne - 15148602

Aidan Cleere – 15160009

**Table of Contents**

**1. Introduction 3**

**2. PHP Driven Web Interface 4**

2.1 Ajax 4

2.2 PHP 5

2.3 CSS 6

**3. Java Desktop Application 7**

3.1 Design 7

3.2 Implementation 8

**4. Conclusion 17**

**1. Introduction**

The goal of this project was to develop two interfaces for an implementation of Tic-Tac-Toe. The interfaces would consist of a Java implementation employing JFrame as a UI and PHP implementation using HTML/CSS as a UI on the user’s browser respectively. Both implementations had to interface with a Soap Web Service provided to us by our benevolent lecturer and our implementations would use this Webservice to coordinate games, retrieve data from the database and any other functionality users would need to interact with other users on the system.

Several screens were specified to be included in both interfaces these included screens such as Login, Register, Main Menu, Leaderboard, Game and User statistics screens. Threads were also required for the Java interface to work correctly and to update the UI. For the PHP side threads are not viable, so JavaScript asynchronous functions making Ajax Requests were employed on a set interval to update the User interface.

Finally, it was specified that the PHP interface should employ CSS to enhance the user experience. As CSS is quite a tricky to implement as seen in Figure 1, we decided to focus our efforts on the functionality of our interfaces and opted to use a template found on the internet and simply gut it of anything that didn’t appear to be used in our web interface.

**2. PHP Driven Web Interface**

**2.1 Ajax**

The PHP driven interface was set up to ensure as much of a separation as possible between the PHP and HTML/JS/CSS as otherwise, development can get messy. Instead, we used several PHP classes and used Ajax functions to asynchronously call on these classes Figure 2. This approach was easily extended when we needed the include functionality like updating the Tic Tac Toe board as we would only need to set up a JS worker with a *setInterval* function to routinely execute said Ajax functions Figure 3.

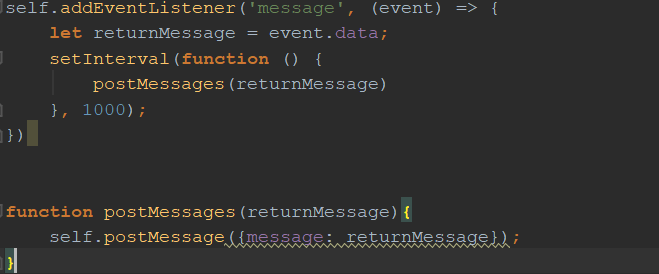
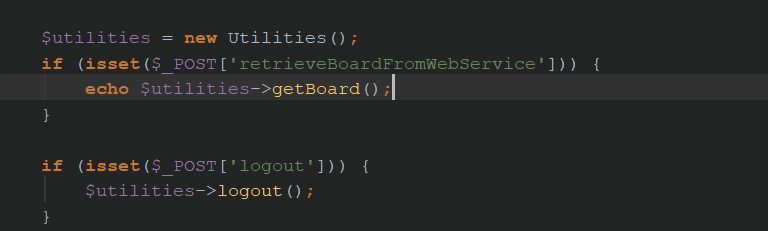
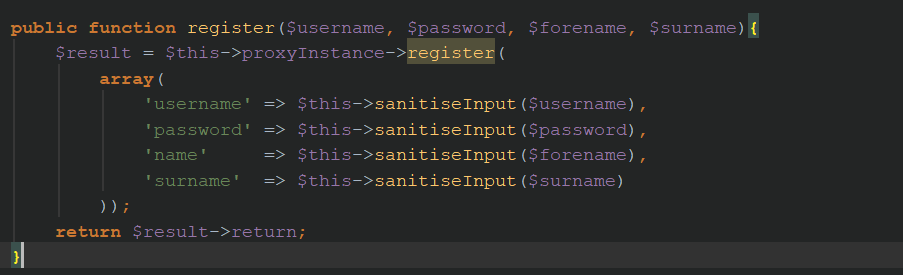
Figure 1.

Figure 2.

**2.2 PHP**

As Previously stated, the PHP was mostly kept separate from the rest of our code and resided in both the *Utilities.php* file (Handled post requests from UI Figure 3) and the *webServiceHandler.php* (Middleman between Webservice and *Utilities.php* Figure 4). We also used PHP to auto login users who had cookies in their browser already set (Figure 5) and conversely redirect users back to the index page if the browser session was invalid(Figure 6).



Figure 3.

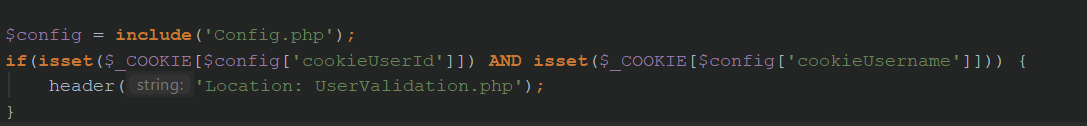
Figure 4.

Figure 5.

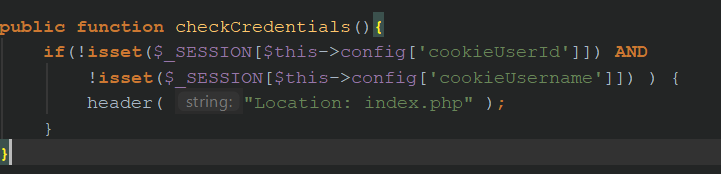


Figure 6.

**2.3 CSS**

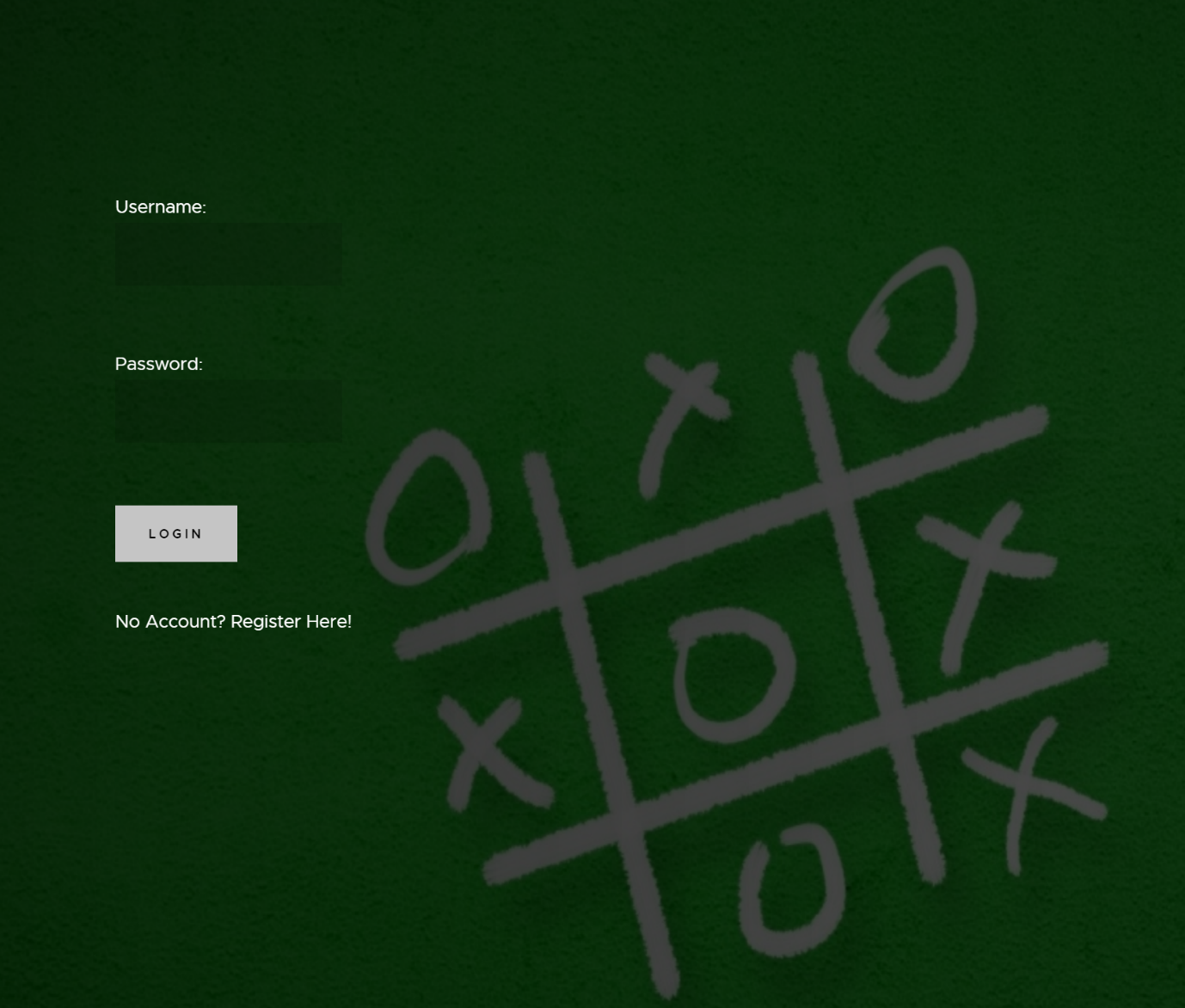
CSS was implemented from a template found online to enhance the user experience on the Web Interface. This template was stripped of anything we didn’t think was necessary (Animated Menus and Images etc.). We also used an image we used from the internet as our background.

Figure 7.

**External Resources used**

CSS Template - <https://colorlib.com/wp/template/transcend/>

Background Image - <http://wallpaperswide.com/tic_tac_toe_game-wallpapers.html>

**3. Java Desktop Application**

**3.1 Design**

For the Java side of the project, we split each window needed into its own object which created a JFrame. We used JFrame forms to design each window except the welcome window. The TTTWebApplication is instantiated in our first object then passed to every other object throughout the program.

The Login and Register windows are simple windows consisting of text fields for username, password, first and surname and buttons to execute commands to the Web service.

The program then opens the main menu which acts as a central hub any window opened off the main menu can be closed without killing the program, the main menu will stay open. It consists of buttons to create a new game, display a window of player stats, to log out and to display a leaderboard. Each button opens a new window by instantiating that object. The main menu also contains 3 JTables one is all the game a user can join the next is all that user’s games which do not have a second player yet and the third is all the user's games which they have played or are playing. The JTables have no buttons. A user can join a game by double-clicking the row which contains the game they want to join. A mouse listener is attached to each table which initializes the main game object depending on the type of game such as is the user P1 or P2. The main menu contains a thread which constantly calls the web service and checks each table for a change if a change is detected the tables are updated by their relevant method calls.

The main game window is where Tic-Tac-Toe is played it consists of 9 JButtons in a 3x3 grid and a label details which player the user is and who's turn it is. It contains 3 threads; the first Thread is triggered when the user creates a game it checks with the web service for a change of game state from -1 to 0 and when this condition is met it unlocks the board and start the next thread. The second thread checks if it's the users turn and if so unlocks the board, so they can make a move. It also makes a call to a function which updates the board when the opponent makes a move and after the first move taken kicks off the third thread. The third thread check for a change of game state to a win, lose or draw and then locks the board and displays the winner.

**3.2 Implementation**

Each button in the Main Menu, Login and Register windows have a java.awt.event.ActionEvent attached to them which execute certain commands when a button is clicked most buttons perform a call to the Web Service such as creating a new game or retrieving certain information needed like getting the league table (Figure 8).

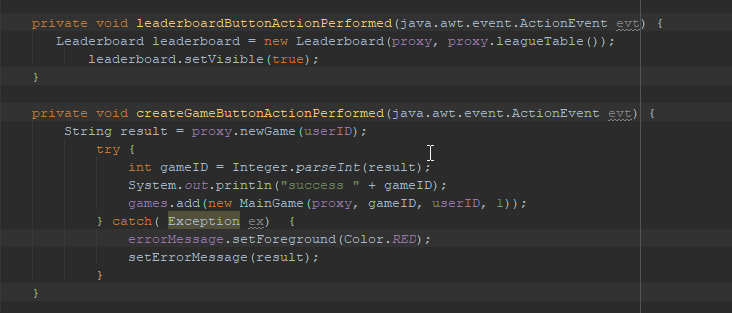
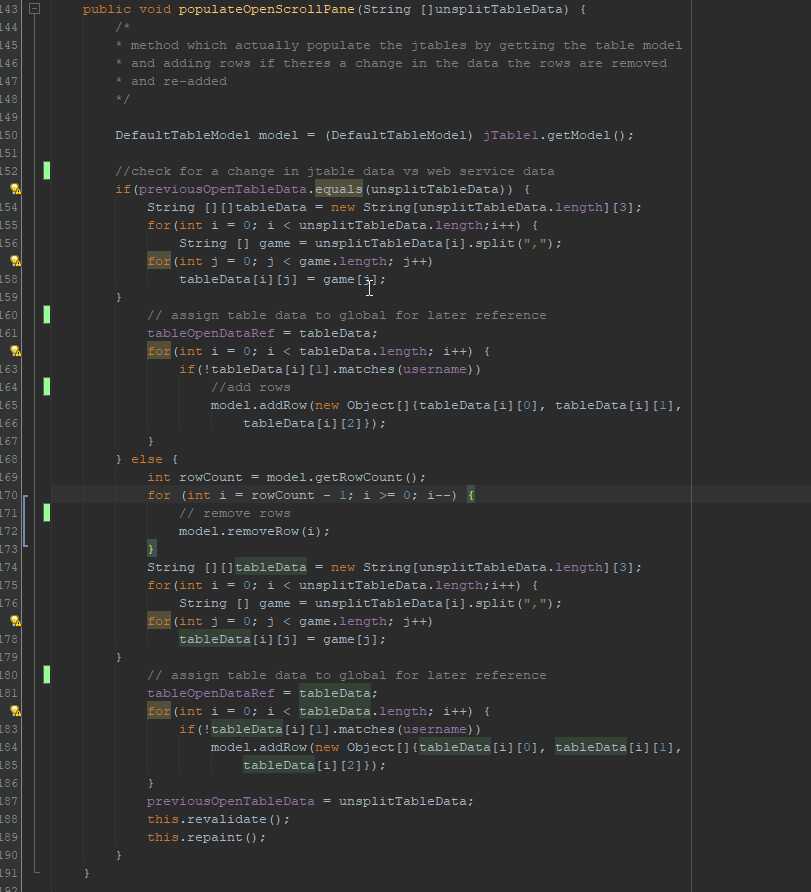


Figure 8.

The JTables in the main menu is populated by retrieving the table model of the relevant JTable and adding rows to them retrieved from the web service (Figure 9). The String return from the web service is stored in an array it is first split by each new line character getting each game. The game string is then split by a comma getting each cell. We iterate through the whole array adding each row to the table. When a change in the data coming from the web service is detecting we remove all rows and re-populate using the same method. A 2D array of the data in each table is stored for reference in the mouse listener.

Figure 9.

The JTables each have a mouseListener which listens for a double-click on a cell (Figure 10). That cells row number is then retrieved, and the relevant information is gotten from the table data reference variable. The gameID retrieved is then used to join a game. Depending on where you created the game or joined it, you will join a game as P1 or P2 and different commands will be executed depending on which player you are.

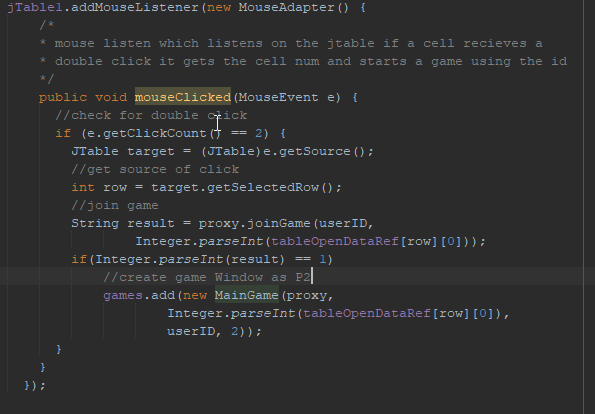


Figure 10.

The threads contained in the main menu and Main Game are implemented as anonymous inner classes this was due to the main game needing window needing 3 thread which must modify variables contained in the main game class. The updateGamesThread (Figure 11) in the main menu executed until the program is killed it retrieves the data for the three JTables splits the data then check for a change against the data currently in the JTables. If a change is detected, then the function which populates the relevant table is called.

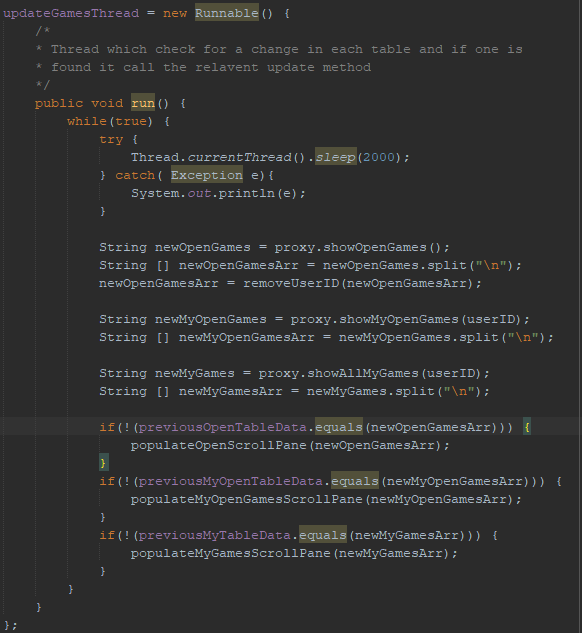


Figure 11.

The thread in the main game, unlike other classes, implements window listener this is so when a user closes the window the class can set a Boolean terminate thread to true which will terminate all running thread and then dispose of the window. Each time the main game window is opened the threads will start again (Figure 12).

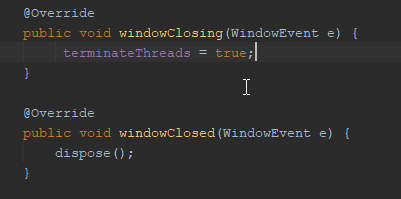


Figure 12.

The board is also locked and unlocked using a Boolean if it is the users turn the Boolean is set to false otherwise it is true this Boolean is controlled by the player turn thread which starts after the CheckPlayerJoin thread (Figure 13) detects a change in game state. The CheckPlayerJoin thread does not start if the player doesn't create a game but instead joins it, the player thread is started instead in this case.

The CheckPlayerJoin is a simple thread which keeps the board locked until the web service returns a change in game state to 0 it then unlocks the board and spins off the player turn thread. This thread will then die as it only runs while games == -1.

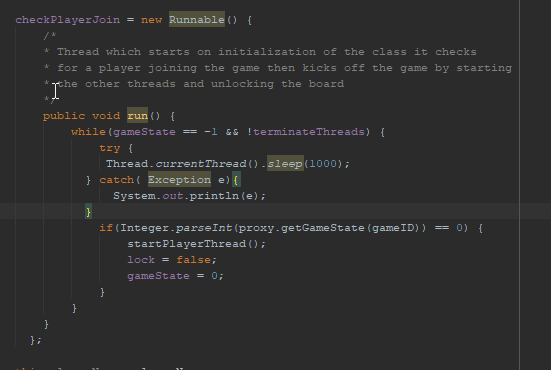


Figure 13.

The playerTurn thread (Figure 14) runs while the game state == 0 and terminate threads is false, it works by retrieving the moves made using the get board function provided by the web service keeps the board unlocked while getBoard returns nothing as anyone can make the first move when a move is made it then decides if it’s the players turn by checking the second last move and if that move was taken by the user then it unlocks the board and lets the user make a move. After it finds that it is again the users turn it calls the updateBoard function which will update the board based on the opponents to move using a switch case and swapping the JButton in the grid with the relevant X or O label (Figure15).

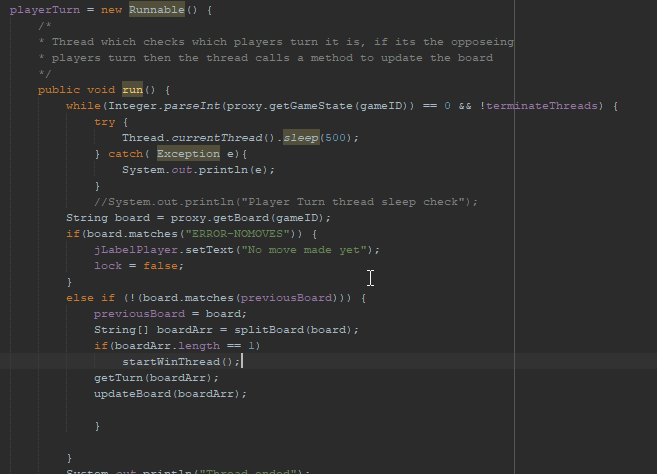


Figure 14.



Figure 15.

The last thread checkWinThread (Figure 16) has the same loop conditions as the player turn thread this thread calls the checkWin function from the web service and when it detects a change it sets the game state to the result returned by the function which can be either 1,2 or 3 meaning p1 win, p2 win and draw. Once the game state changes all threads are terminated and the winner is displayed, and the board locked.



Figure 16.

From the main menu, players can reflect on their previous games matches from the Player score screen. The player’s game statistics are provided on this window and show all of the relevant information, number of games played, number of wins, draws, and losses, and the player's win-to-game ratio.

Upon having clicked the Player Stats JButton, the occurring event a new window opens. A call to the Web Service using the leagueTable() function returns the resulting String of all the completed games that have been played to this point. This String is then broken split into its separate games wherever a line break has occurred, storing these games in an array, and is split furthermore into the important aspects of each game (players involved, the winner etc.) wherever a ‘ , ’ has occurred, storing the data in a separate array.

This action takes place in the calculatePlayerStats function (Figure 17). Each game is checked to find the results of each game the player is associated with. The player’s username is checked with both the Player 1 and Player 2 values of a game, at the respective indexes inside the game array. Switch statements then trigger if the player did take part in the game to find the result of the player for that game, with each case incrementing the number of games played and the respective outcome of a game.

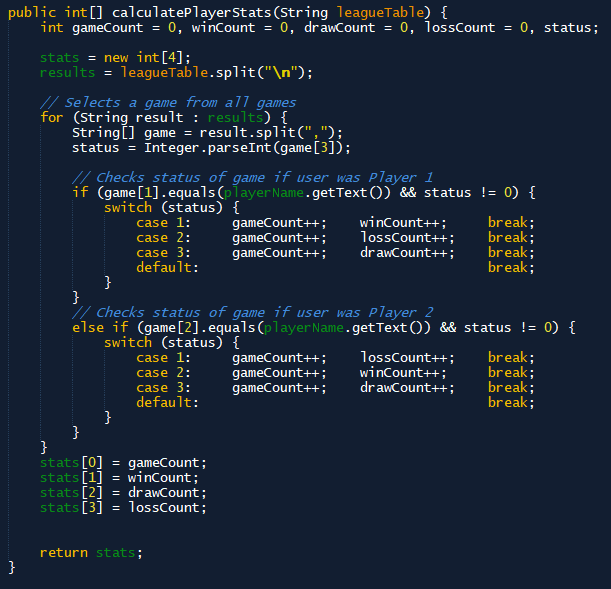


Figure 17

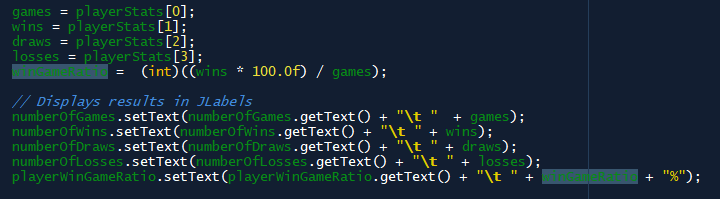
The outcome values are stored in the stats array and returned. The value of the winGameRatio variable is given by cast the result to an int. A JLabel for each variable is used to print the results to the user, where the result is latched on to the end of the text within the label (Figure 18).

Figure 18

The player leaves this window back to the main menu by pressing the dedicated JButton that disposes of the window until a new instance is created, or by closing the dialog, which does the same.

**4. Conclusion**

In conclusion, we believe we have satisfied all the specifications of this project and are happy with our implementation. We gained a lot of experience from this project especially in the realm of Threads and Ajax functions and feel it was a decent challenge for all group members.