Software Development Project

Tic Tac Toe

Report By: Gabriel Stroe

Course: Software Engineering

Project Dates: October 10th 2016

Date Submitted: November 2th 2016

# Introduction:

The goal of this project is to create a Tic Tac Toe game that one player can play against a computer or two people can play with each other. This project will have a fully graphical user interface. In addition, since Tic the goal for the AI of this project is to have 3 different levels of difficulties, each level reducing the chance for the AI to make a mistake. For example in easy mode, the AI will make a mistake 50% of the time, on medium it will make a mistake 20% of the time, and on hard it will make a mistake 1% of the time.

The program should serve as a fun game that the user can play with a friend or by themselves against the computer. The program should also be stable and easy to use.

In the future this project could be expanded for other types of games and different sized boards for the Tic Tac Toe. Also adding better visual effects like animations or even 3D effects could really enhance the enjoyment of the program.

# Estimates/Plans:

Based on the previous projects and the length of time each section took me, my estimate for this project is that it will be about 650 lines of code and take approximately 18.15 hours to complete. Due to my estimates being shorter than the actual results on the last project, this time I have decided to slightly over estimate how long I believe each section will take me. Breaking up each section I have assumed that the base game design will take me about 3.35 hours to complete from the design phase to the functional testing phase. This will the largest functional component. All of the time and LOC estimates are in the Project Log.xlsx file in the Estimates Worksheet.

The flowing is the current plan to complete the project. The idea is that by each day the following item will be complete much like a milestone.

* 10/19/2016: Requirements and High Level Design Complete.
* 10/22/2016: Starting Menu and 2 Player Game Detail Design complete.
* 10/23/2016: Victory/Defeat Screen, New Round Interaction, 1 Player Game UI and AI Logic Detail Design complete.
* 10/24/2016: Starting Menu and 2 Player Game Coding and Functional tests complete.
* 10/25/2016: Victory/Defeat Screen and New Round Interaction Coding and Functional tests complete.
* 10/29/2016: 1 Player Game UI and AI Logic Coding and Functional tests complete.
* 10/30/2016: Integration test and Retrospective analysis complete.

With this milestone path I should have an extra 3 days to complete the project if I run into any unexpected issues.

# Requirements Analysis:

This program has the following requirements:

* Written in Java using the Intellij Studio IDE.
* Have a graphical interface to show the Tic Tac Toe board.
* Allow a player to choose between playing against a computer and playing against another player.
* Keep all of the traditional rules of Tic Tac Toe.
* Show the victory statues for each player and allow them to start another game.
* Keep score until the user returns to the main menu.
* Prompted each user when it is there turn.
* Have an AI that can play with 3 different difficulty levels (If time allows, bare minimum is to place move in next available slot).

# High Level Design:

This is the high level design for the entire system. It shows how each requirement will be completed among the different modules of this project. Below is a list of all the modules and their features followed by a diagram that shows their interaction with one another.

* Module: Main Menu
  + Must be able to Start a 1 or 2 player game using two different buttons
  + Can set the difficulty level of the AI (if time allows) with checklist style button
* Module: 2 Player Game
  + Shows which player is using which character (x/o).
  + Shows who’s Turn in currently is with sometime of Highlight or arrow pointing.
  + Shows the current scores from playing consecutive matches.
  + Displays a working Tic Tac Toe Board that players can interact with it.
  + Checks for a three in a row condition or if the map is full. If either is true then it goes to the victory/defeat module
* Module: 1 Player Game
  + Shows whether the player is x’s or o’s.
  + Shows the AI’s move and prompts the player when its their turn.
  + Shows the current score between Shows the current scores from playing consecutive matches.
  + Displays a working Tic Tac Toe Board that the player can interact with.
  + Checks for a three in a row condition or if the map is full. If either is true then it goes to the victory/defeat module
* Module: AI
  + Gets called with a 1 player game after every move from the player.
  + Minimum requirement: will place in the next available position.
  + Ideal requirement to meet: 3 levels of difficulty based on a percent chance to make a mistake.
  + Returns move to game module
* Module: Victory/Defeat Screen
  + Shows whether the match ended in a draw or who won the game.
  + Prompts the player(s) if they want to return to the main menu or to start another game and tally the wins and losses.
  + Transition to the player(s) choice.



# Low Level Designs:

The following are low level Designs for each module. As for the layout of the modules with UI elements I would create the elements in the Intellij Studo GUI environment. I am only loosely familiar with JSwing so adjusting it in the environment was the easiest method for the GUI’s design especially since there was no specified design requirement for the GUI other then it shows the game.

The first is the Main Menu Module.



This Module is fairly simple to understand. Users are presented with a choice of one or 2 players. Each choice will flag the proper game mode and begin a game.

The next is the Two Player Module.



This module waits for input and just checks if the space has already been taken. It calls the end game logic function which checks if the Victory/Defeat module needs to be called.

This is the Victory/Defeat Module.



For this module choosing rematch keeps the score and swaps the starting players. The board is reset as normal. The main menu doesn’t actually change any flags as choosing one of the game modes in the main menu defines those.

This is the 1 Player Mode Module.

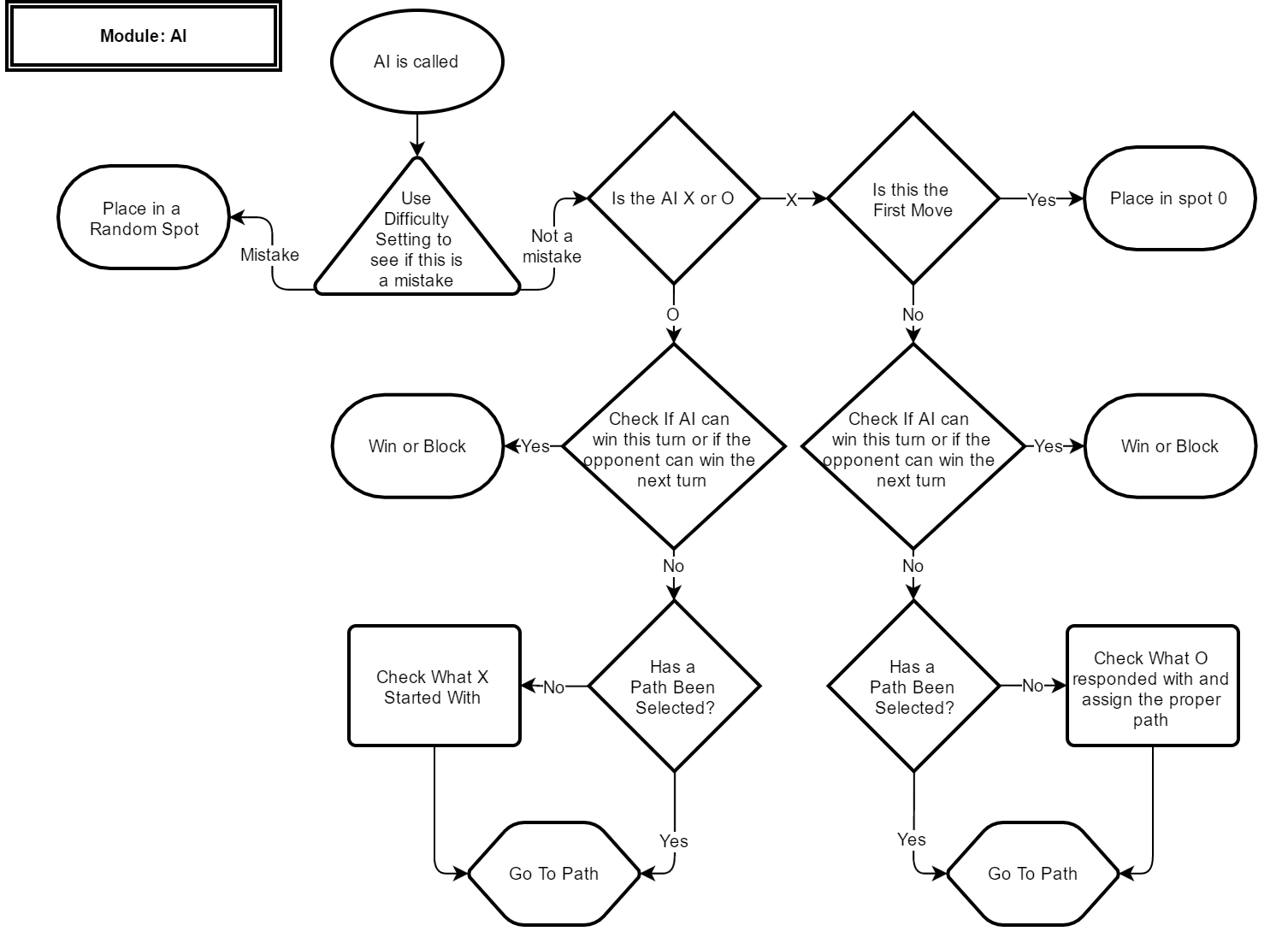


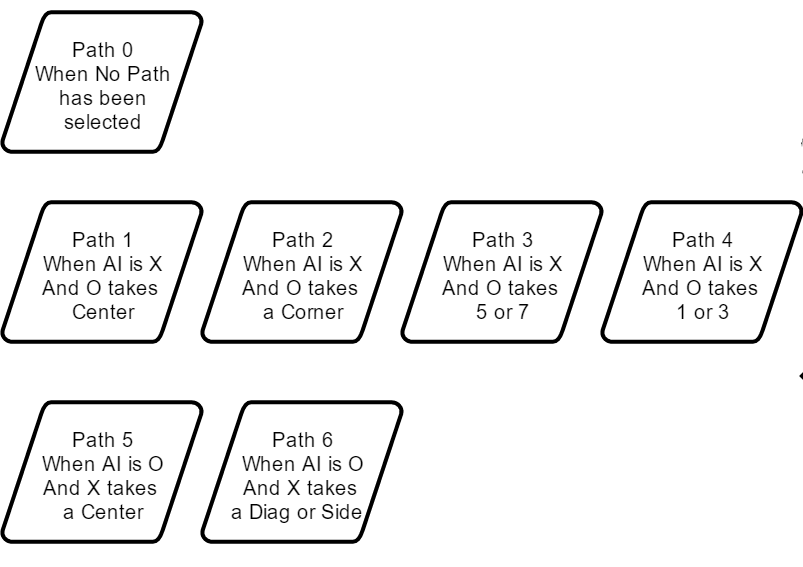
This module is similar to the 2 Player module. The main difference is it has the AI module as another input source and it always changes between the two inputs. In hindsight this should have be combined into one game module since this was really just a modification of the 2 Player module.

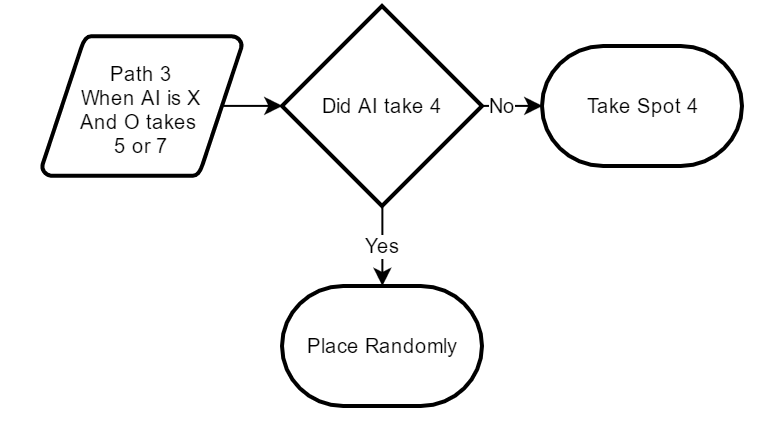
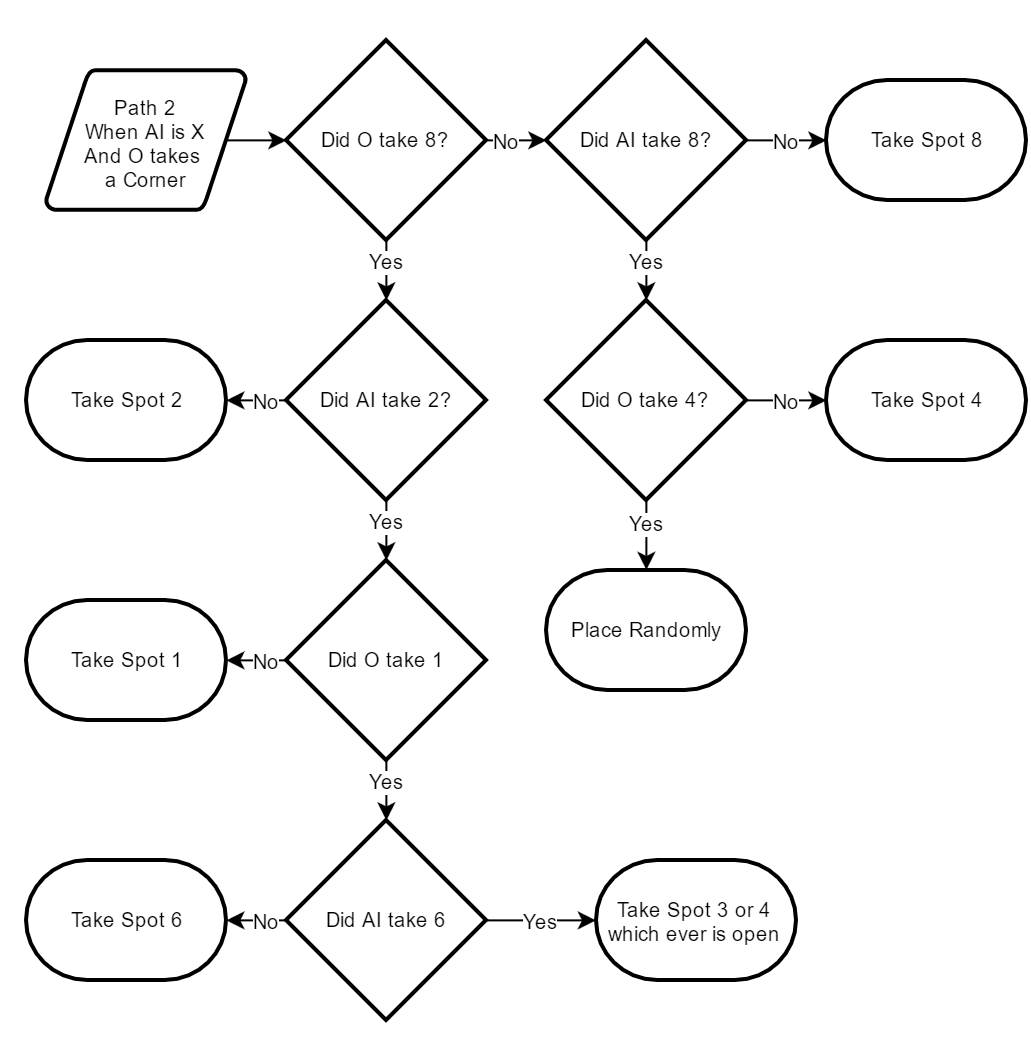
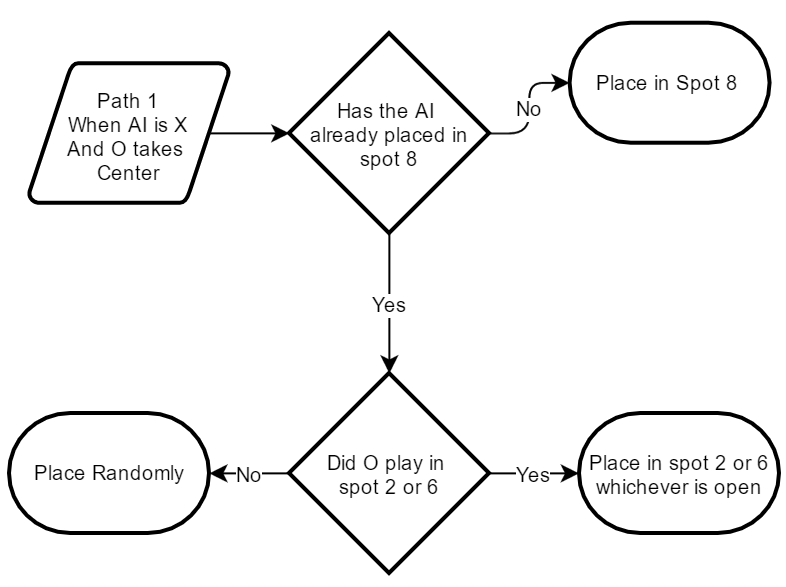
The Following is the AI module. The plan for this was to use the same basic starting move depending if they were X or O and follow a set path based on their opponent’s first move. The actual strategy was gathered from my own test runs and <http://www.learnplaywin.net/tic-tac-toe-strategy/>. Note that for Path 5 and 6 because of the repeated if-then statements I simplified them into a few larger blocks.

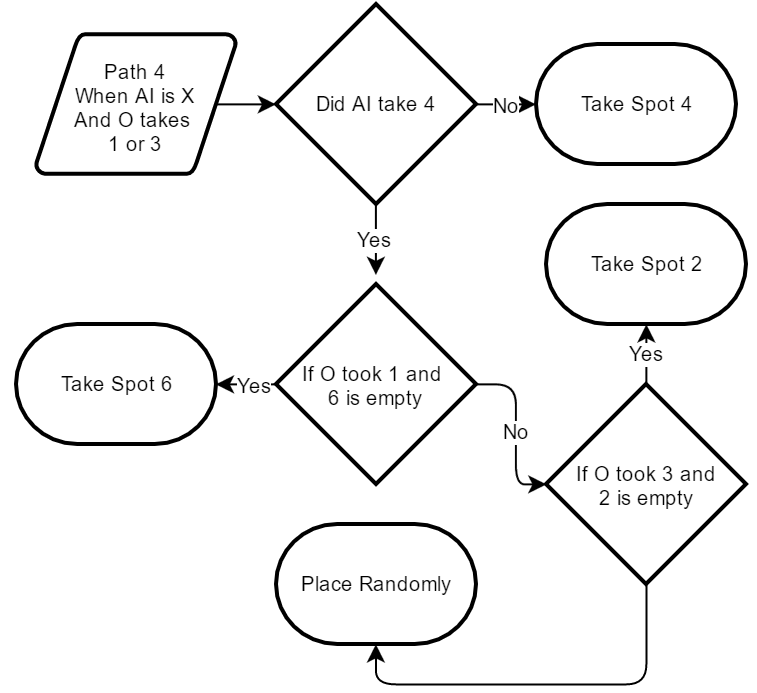
This logic was changes so that the before going into the path logic the AI will see if they can win this turn or if they need to block the opponent from wining the next turn. This greatly simplified the Paths as much of their moves used that same mechanism.

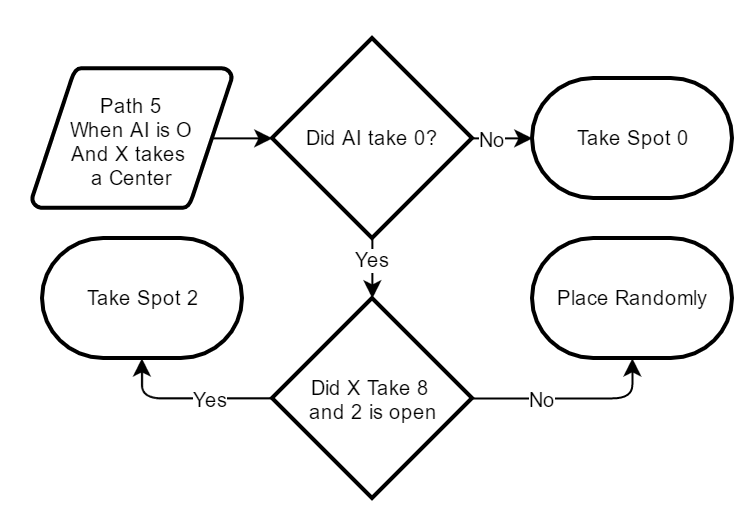
Also a note, the place randomly option only exists as there are a few spots where you choice won’t matter and the game will end in a draw either way. That’s what that endpoint is for.

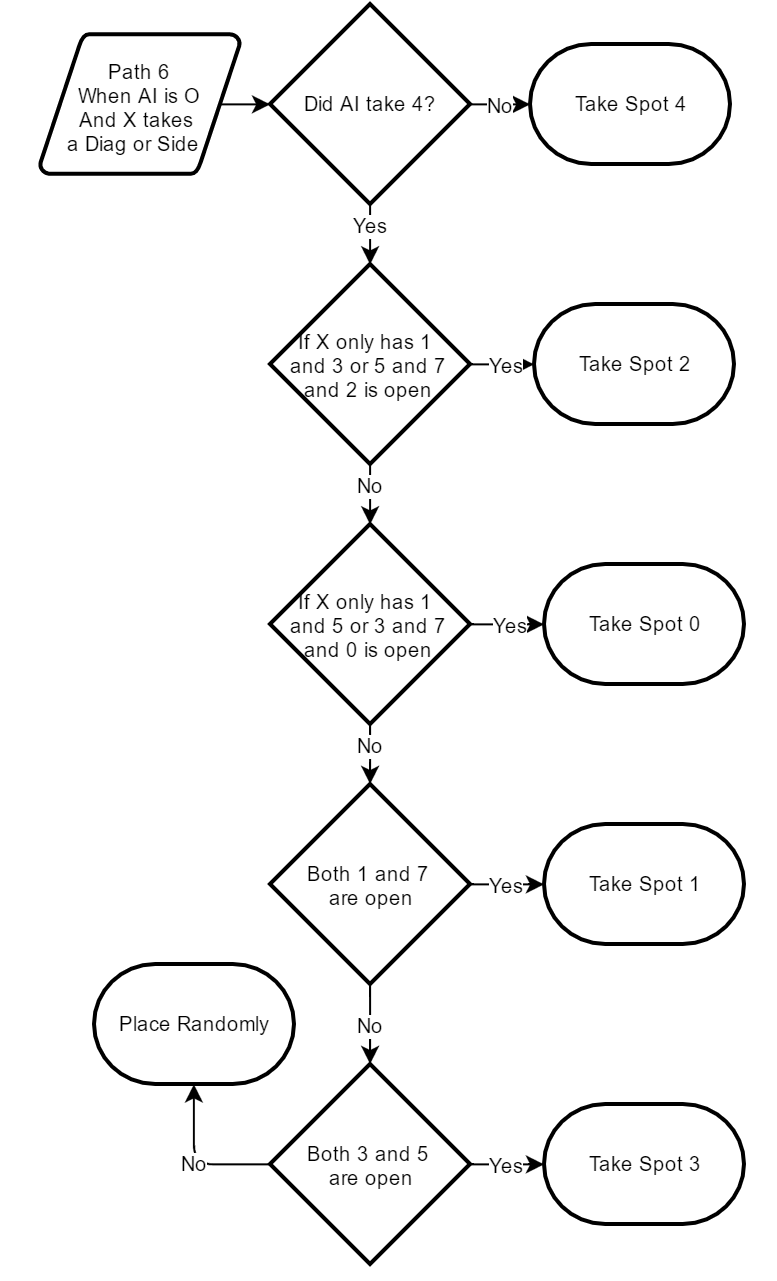












# Function Lists:

* **class TicTacToeBoard** - Contains all of the GUI elements as well as all of the program logic.
* **public TicTacToeBoard()** - Constructor for the TicTacToeBoard class. This is called by the main function and includes all of the action listeners. The application only includes 1 type of action listener which is the mouse click event. It is also applied to 2 types of objects. The first is the main menu buttons which cause it to launch the game with the correct flags for that game mode. The second is the logic for the user selecting a square, which checks to see if the square is taken and places a mark on it if it is not. It also always checks for a victory condition after the placement and launches the AI\_Logic() function if the onePlayerMode flag is set to true.
* **private void AI\_Logic()** - Performs to move for the AI. Also checks if the victory condition is met.
* **private void gameOver()** - When called this prompts the user with who the won the game and asks if they would like a rematch or to return to the main menu. It also changes the correct flags to perform the users choice.
* **private void changeTurnIndicator()** - Checks who’s turn it is and assigns the indicator to the correct user.
* **private boolean winCheck()** - Returns true if a victory condition is met, false if it is not. It also flags the winner flag with which player won the game (0 for player 1, 1 for player 2 and 2 if the game is a draw).
* **private void boardClear()** - Clears the board both visually and in the boardFill array.
* **private void startingValues()** - Sets that starting flags when starting a game from the main menu. These are the values that are the same regardless of a one or two player game.
* **public static void main(String[] args)** - The main function. Starts a JFrame and sets the main panel as its context panel.
* **private int centerLineCheck(int player) –** Checks for a move across the center for the player. The player must control the center for this function to work. Will return the spot to take or -1 if there is no spot that this matches
* **private int sideLineCheck(int player) –** checks for a move across the sides for the player. Will return the spot to take or -1 if there is no spot that this matches.
* **private int randomResult()** – return a random result from an available spot.

# Implementation:

The implementation for this program is a Java program created in Intellij Studio. The coding conventions used are traditional java coding conventions.

The naming scheme for each GUI object is as follows:   
(2 characters showing object type) + (panel on top of) + (name)   
For example: lbBoardP1 is the label for Player 1 on the Board Game panel.

When it comes to the GUI implementation, I used the Intellij GUI design environment. While all of the objects are called in the TicTacToeBoard.java the actually placement and design of the files are in the TicTacToeBoard.form file that Intellij uses.

# Testing:

Testing this time around seemed to go much smother. Although there were several functional units to test and a few test cases in each, the project seemed to only have a few very small and easily fixable bugs.

The most difficult fix was seeing that my end game logic was flawed as it only worked if you got 3 in row on your first 3 moves. I quickly adjusted my diagram then made the same adjustments in my code and it was working correctly.

I believe that the more detailed design items are the reason for this. I noticed as I was creating them I would change a lot of the things I have already written so it would work better or not cause a bug.

# Conclusion/Project Retrospective Analysis:

In conclusion after Adding my planned AI feature the line took more hours and more LOC then I had expected. In my previous submission I was under as I seemed to have over-estimated or had my estimate just about right. It does appear that I under-estimated how hard and lengthy coding AI would be.

From looking at my plan and milestones compared to my actual time log, it can be seen that I was very much off schedule for this project. Fortunately, this is why I gave myself quite a good buffer time, incase this is what ended up happening.

For the next project I will be looking to stick to my plan a bit better so I don’t need to do the project in such large blocks. Also I won’t under estimate any complex AI. Other than that I feel that my implementation worked fairly well and I would be looking to replicated that with the next project.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Estimated: | Measured: | Analyze: %Error |
| Time | 18.15 Hours | 20.5 Hours | 17.7 % |
| Size | 650 LOC | 787 LOC | 11.5 % |

Actual Project Time and Program Size vs. Estimates

# Appendices:

* Project Log.xlsx – The project log that contains estimates, time logs and defect logs.
* Test Log.xlsx – A spreadsheet of the different test performed on the code.
* Project 3.jar – The executable for this project.
* Process Checklist.xlsx – A spreadsheet with a checklist to complete each phase of the project.
* TicTacToeBoard.java – The java file for this project.
* TicTacToeBoard.form – The GUI layout for this project