# Tic Tac Toe Project Documentation

## 1. Team Information

Team Members:

* Member 1: Calvin White (GUI implementation, Controllers, UML, Documentation)
* Member 2: Pratham Snehi (Gameboard, Player logic, UML, Documentation)

## 2. Project Overview

* This is a two-player classical Tic-Tac-Toe game, where you would have to score three of the same symbols in a row in order to win the game.

## 3. OOP Concepts Applied

Abstraction: Usage of abstract classes in order to hide implementation of common things, like the Player classes and all the different functions that the player does like make a move, etc

Encapsulation: Use of private and protected wherever possible in order to contain the variables inside the classes if we do not need them outside the classes

Inheritance: Player class is implemented by the Abstract player class, and the Abstract player class is extended by the Human Player class. This makes it so that we can have common functionality for the entities that are able to play the game. This makes it very easy to add classes like “Robot AI” in the future that take some functionality from the parent class, but also implement some of its own.

Polymorphism: We create two player classes in order to play the game, and these classes have their separate properties like Name and Symbol in order to differentiate them from one another, despite having the same parent class (which is the AbstractPlayer)

## 4. Usage Instructions

Setup and Run:

1. Download the whole java project from github, and use the Eclipse Java IDE in order to open an existing project
2. Navigate to the “GameController.java” class and hit the run current file option. A GUI should pop up if you have installed the required dependencies.
3. Have one player play first, and the other player can move afterwards, and continue this by taking turns one at a time (Note that the first player gets the symbol of ‘X’)
4. Repeat the above until one of the player wins or the game ends in a draw, then you are free to reset the game and play it again by hitting the reset button

## 5. Testing

Test Scenarios:

|  |  |  |
| --- | --- | --- |
| Scenario | Expected Outcome | Tested Outcome |
| Player clicks a cell | Marks that cell with the user’s symbol (either X or O) | Same as expected outcome |
| Player tries to click a filled cell | Nothing happens: no cell change, no exception throwing | Same as expected outcome |
| Game ends in a win | Updates the score of the winning player by 1, and restarts the game | Same as expected outcome |
| Game ends in a draw | Resets the game, with no player score incrementing | Same as expected outcome |
| Board resetting | Empty board is displayed, resets the board derived from the GameBoard object. | Same as expected outcome |
| Invalid symbol assignment (developer level) | Throwing of exceptions in case of invalid assignments | Same as expected outcome |

## 6. Known Bugs or Limitations

* Limitations include:
  + Player name cannot be customized, and is defaulted to Player 1 and Player 2
  + You cannot customize the theme of the GUI window to match your system’s theme
  + There is no implementation of permanently storing the scores for the user. Every new run of the game is a fresh start
* We tried all possible edge cases, like quickly clicking a cell multiple times, resetting the game board twice, trying to click on boxes after the entire game board has been filled, and many more, and did not find any bugs during this process.

## 7. Future Enhancements

* AI player so that you only need one player to play
* Name customization
* Leaderboard system based on the name of the people that play