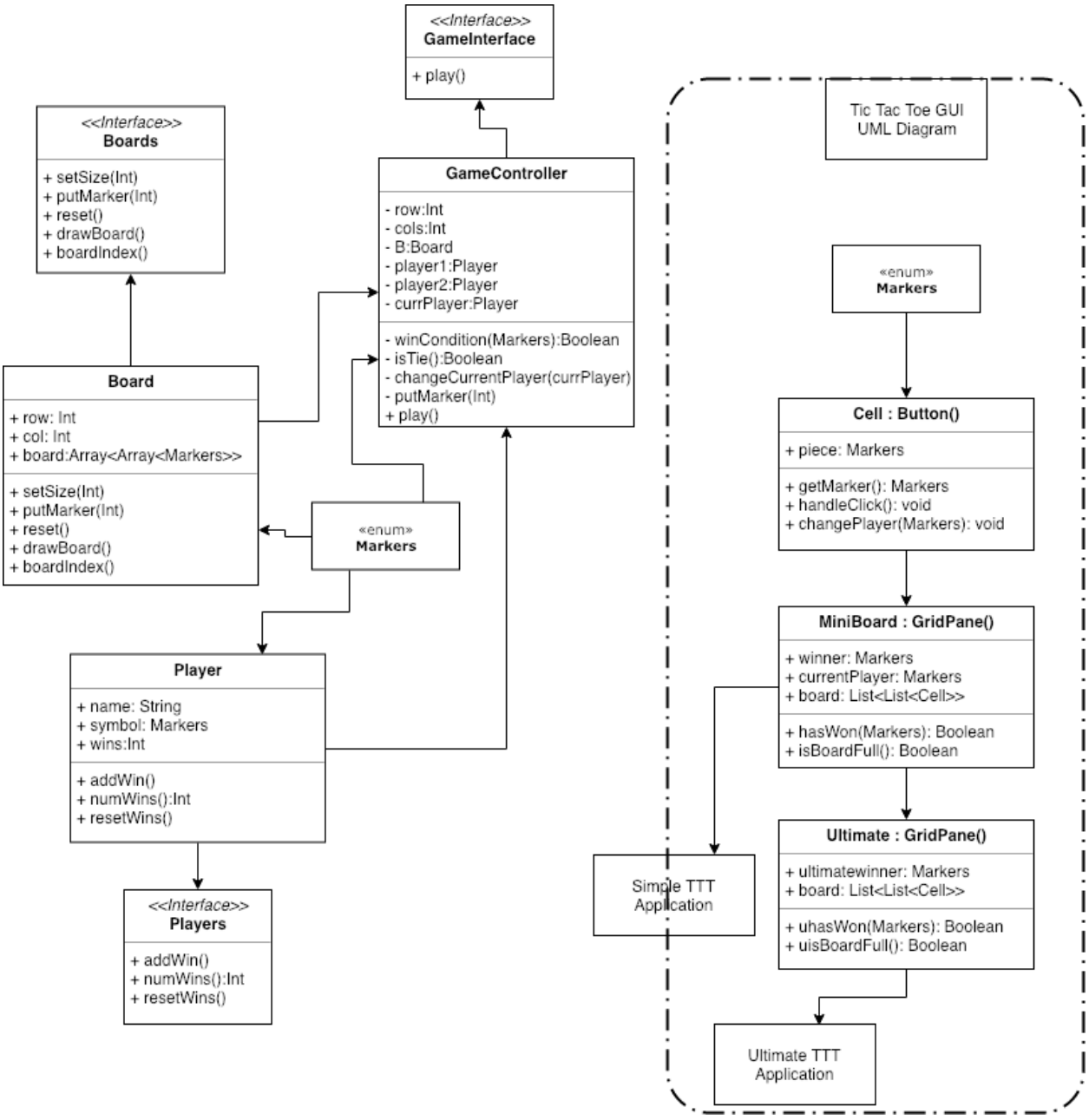
Marc Gozali

Tic-Tac-Toe Project2820

To run the game, run the main method in the Menu.kt file



To briefly explain this UML diagram, it's split up into two parts, the text version of TTT and the GUI version of TTT. On the right hand side, I tried to encapsulate it with a border, but it slightly overlaps on the "Simple TTT Application" square. The GUI UML is quite simple, as it

only utilizes three classes and one enum class. There is a Cell, which extends the button functionality of JavaFX, and the MiniBoard and Ultimate classes, that extend the GridPane shape of JavaFX. The MiniBoard creates a 3x3 array of Cells, which each in turn function as the X and O placeholders for the game. The Ultimate class also extends the GridPane shape to create a 9x9 of mini 3x3 MiniBoards to create an ultimate Tic-Tac-Toe game. On the left hand side, I have 3 classes, each with their own interface, and one enum class. First off, the board class creates a nxn array of empty Markers, which are later filled in by either an X or O Marker to play the game. The player class designates each player to a symbol and name, and it keeps track of their wins and losses. The GameController class has all private variables, so that users don't mess with the functionality of the game outside of the game controller class itself. It functions as the game state handler by using a board and players, keeping track of the currentPlayer's turn, and checking, after each move, if the game has ended.

This Tic-Tac-Toe game features the basic 3x3 Tic Tac Toe GUI board, as well as a fully functional text version of the game. There is an Ultimate GUI board, however, it wasn't fully completed as problems with using JavaFX arose throughout its creation. It also features a functional Menu, Back to Menu button, and Exit Game buttons, that were a neat feature to add. I am most proud of my regular Tic-Tac-Toe GUI as I thought of an interesting way to handle the game state through one method, handleClick(), in the Cell class. The handleClick essentially functions as the game handler compared to the GameController class stated previously. However, this time, after each click, the game will check if the last player has won or the game has ended.

Throughout this project period, I think the hardest part for me was the use of JavaFX. The text version of the game came very easily to me, and I was able to complete that part of the project in only a couple of hours. However, the JavaFX portion of creating the GUI gave me so

much trouble, it made me reconsider if I really understood how to code or not. Majority of my time spent on this project went into reconfiguring the JavaFX as I couldn't get the FXML to work for the first couple of days, so I had to learn how to create things using text only. I think for future projects, I have a much, MUCH better understanding of how FXML and JavaFX play into creating applications, and it'll be much easier to start with using FXML rather than building it up from text only JavaFX code. One of the most interesting things I learned was to extend classes so they inherit Buttons or GridPanes, so they can be directly added into the scene for the application. I learned of this method through YouTube videos, and my application of said method can be seen when using the Ultimate TTT application, where it splits up the JavaFX part from the Kotlin code. Most of the information I learned to do JavaFX came from video tutorials or stack overflow pages. In the end, I'm more frustrated that I couldn't get the Ultimate TTT board to work, even though I put 4-5 days' worth of effort into it, but I learned some interesting techniques, which is always a valuable experience. Overall, I am most proud of my ability to overcome the problems I had with JavaFX and be able to create a somewhat working application. This project was a great introduction of how to create applications using modular interfaces and classes and definitely taught me good coding practices.