Oliver Flynn – 19336592 – Minesweeper Report

PART 1 – Game implementation

The first step I took to implementing the game was deciding how the UI would look. I decided to base the look of my UI off of the minesweeper, which is built into the google search engine, starting by drawing the board and bellow it any buttons needed to control the game. Once I had decided how my UI would look, I started by defining my own module which I would import to help abstract some of the functions and make my main code a bit more readable.

To design the GameBoard module I started by defining a data type of the same name which would store key attributes in record notation used to query the board. I also defined some other key data types to help throughout the program like the "State" data type which represents what state the game is in, namely still in play or finished. Once I had the data types initialised I then started writing the functionality. I used functions to get and set statuses and values of each square on the board, I also used simple functions to check things like whether a coordinate set is on the board or not. The main functions of the GameBoard module were the plantMines, uncover and placeFlag functiona. Which would plant a specified number of mines randomly on the board, change a specified square from hidden to visible and change a specified square from hidden to flagged, respectively. These are called in the main code depending on user input.

Once I had the GameBoard module created (Figure 1 The final GameBoard UI) I then went about implementing the main body of code, the game itself. I started by defining the setup function which is used as the starting function when using Threepenny GUI module. In setup I decided the first step would be to select the difficulty, so I first made three buttons, one for each difficulty option (easy,medium,hard). Because I struggled with displaying these on the same page as the board I decided that once the difficulty was set I would clear the window and then draw the board. To do this I made another function called play which would take parameters to set the board size and number of mines as per the difficulty. Then once a button was pressed and unpressed I would call play with specified parameters matching the difficulty selected.

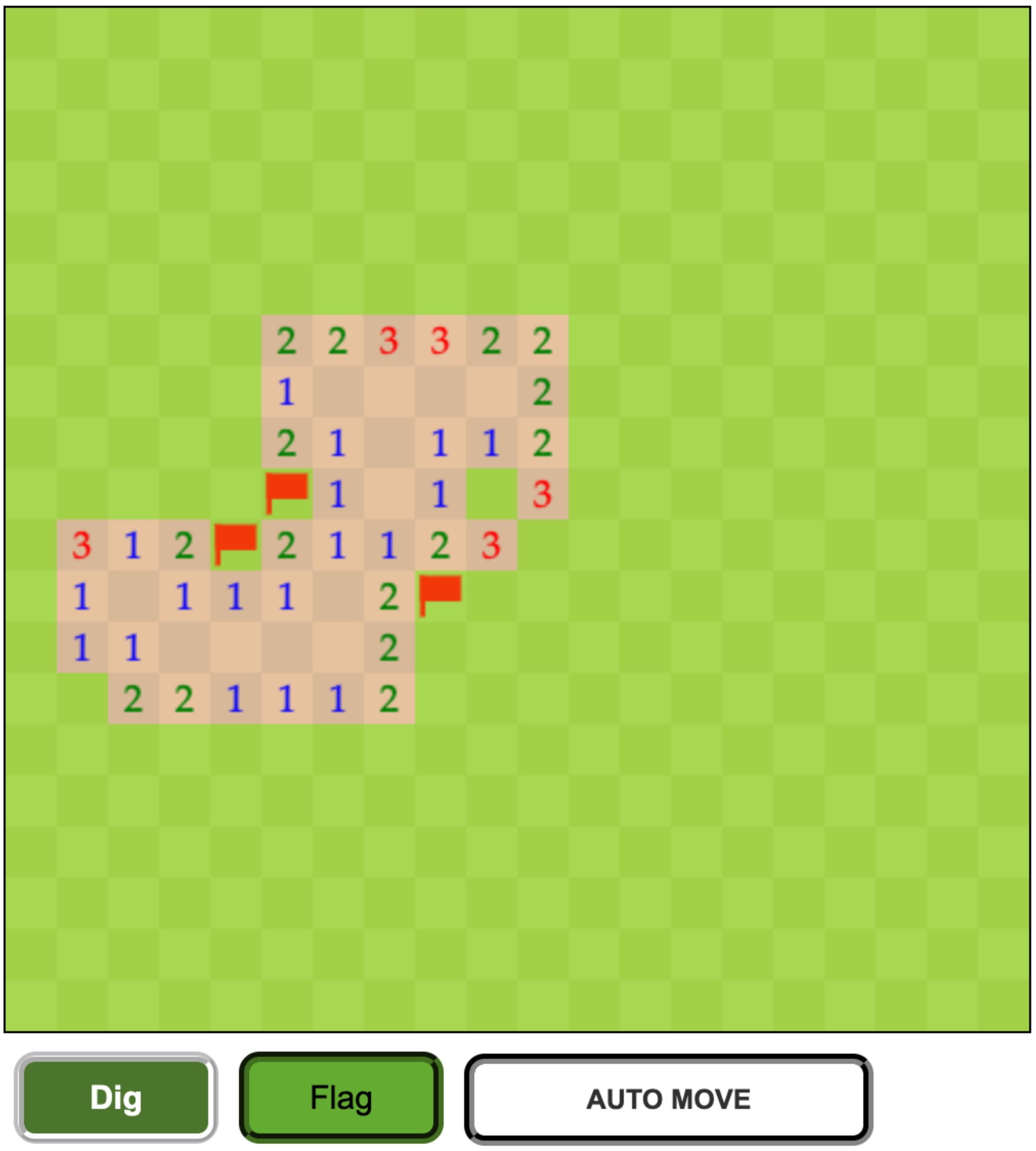

Figure 1 The final GameBoard UI

The play function is the biggest function in the code and is the heart of the program. I start by making the DOM elements; the canvas for the board, two buttons for the mode selection (dig and flag) and a final button to be used in the auto player mechanism. I then initialise several Mutable IORefs which I use in this function to get things like the current gameboard object, current mode, mouse position etc. I then draw the board on the canvas and the buttons bellow the canvas. After I add the elements to the window I wrote several on Event function for each button and the canvas. The on click events function for the mode buttons would change respective variables so that when the canvas is clicked the player either digs up (reveals) a square or plants a flag. The next big function was the on click for the canvas, this would run when the user clicks anywhere in the canvas. If the game is still in play and the current mode is dig and the user has clicked on a hidden square it will become revealed. If the user has clicked on a mine the game ends, otherwise it reveals the square and if the square has 0 adjacent mines it reveals all its adjacent squares recursively. If the user is in flag mode it will place a flag on the square to mark it as a suspected mine. After this is all evaluated the canvas is redrawn with the new display. If the game status has changed to Won the canvas will be drawn with a transparent box on top saying “WINNER” and if the status has changed to loss the canvas will be redrawn with a transparent box on top saying “GAME OVER”.

PART 2 – Auto Player implementation

To implement the auto player I started by making a module AutoPlayer which passed only one function to the main program, “autoMove” which is called when the earlier defined “AUTO MOVE” button is clicked. This function starts by iterating over all squares and checking if a value is equal to the number of hidden squares adjacent to it, if so it flags the first adjacent square and stops itterating. The function would also check if a square had the same number of flags adjacent to it as its value then any hidden adjacent squares could not be mines and it would choose one to uncover. If neither of these checks were successful my implementation would choose a random square to reveal as there is logical way to make a correct move next, as a human you can make an educated guess but since there is no guarantee that it will be right its hard to program this, so I just left it as a random selection.