Róisín Ní Bhriain - 18326577

Minesweeper

Introduction

The minesweeper I created is a beginner game with 10x10 squares and 10 bombs. It could easily be modified to include other modes such as intermediate or difficult mode but for the purpose of this project I decided to keep it simple so I could test it more easily. Thus there are some hard coded dimensions in the code which could be changed to work for other modes.

Phase 1

Randomly distributing mines (complete):
This was achieved using a random number generator. I first generated a list of random numbers in the main then used this as input to the setup of the GUI. This list of random numbers was the thing that made the distribution of the mines random. Each time the game was reinitialised it would create a new seed for the generator using this list of random numbers.

Uncovering (Revealing) and flagging mines (complete):
To reveal a mine you must choose the reveal button. Then you click one of the squares on the canvas which will provide the game with an x and y position. The program creates a command with the mode chosen r for reveal and f for flag and the x and y position. If the square has been uncovered already nothing happens. If a bomb is uncovered then the game is lost and the status of the board is changed. If an empty square is uncovered with zero bomb neighbours then each of the squares around it are uncovered. This is done by generating a list of valid neighbours to check that have not already been checked.

Flagging mines is done in a similar fashion. The user must click the flag button and then one of the squares on the canvas. The system will record it as either a flag empty or flag bomb but will display the same colour to the user. If the user tries to flag an already uncovered square then nothing will happen.

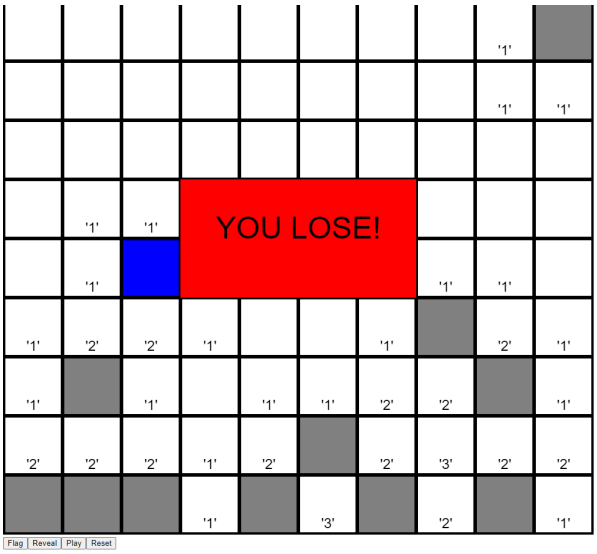
The endgame condition (complete):
The endgame condition for a win was simple - after a move if there were no empty squares or empty squares that were flagged as bombs then the user had won. The endgame condition for a loss was if the user chose to reveal a cell that had a bomb in it. This would change the status of the board to true - to indicate that the game had been lost.

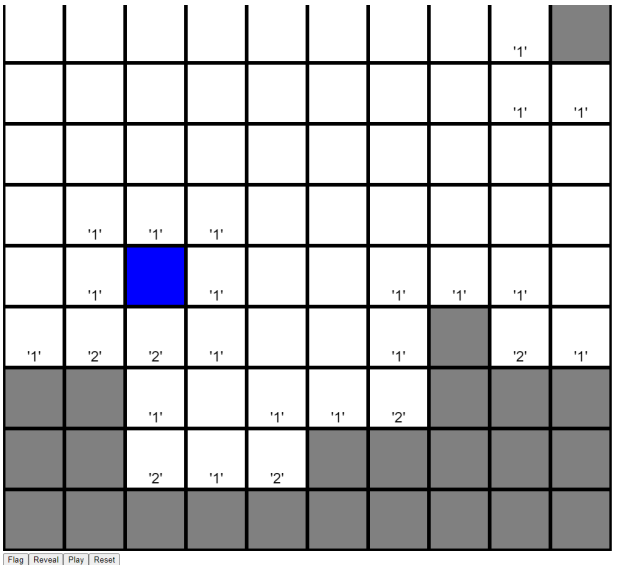
Threepenny GUI (complete):
The buttons flag and reveal both set a different mode in the game. That way when the user clicks the canvas the either flagging or revealing functions will be used. On clicking the reset button the board will reset with bombs in different locations. The play button will use the autoplayer to make a move. After each move or each reset the canvas will redraw the board.

Róisín Ní Bhriain - 18326577

I used IOref to keep track of the state of the board which was then updated any time the user flagged a cell or revealed a cell. The mode that the user was in was also kept in the IORef - either OPENING for revealing mines or FLAGGING for flagging mines.

An unknown square is grey, an empty square is white with the number of bomb neighbours in it. A flagged square is blue.





Phase 2

This attempts to use the basic patterns in minesweeper to reveal a square. If no squares have been revealed it simply reveals the first square - this might be improved by making it reveal a random square if there are no unambiguously safe moves available. I found it too difficult to implement revealing squares based on probability on time so I removed my functions for this. Although I suspect if I were to do this I would need some sort of list of probabilities to compare.

I made the squares secret from the player using my two functions. Remove details changes both Bomb and Empty to Unknown and FlagEmpty and FlagBomb to Flag. Neighbours stay the same as they have already been revealed. This means the player is not sure which is a bomb and which is empty.

I decided not to allow the player to open already flagged squares as the user thinks that there is a mine in these squares.

The patterns that my design can find are: currently none - hopeful I shall get corners - maybe 1 - 1 - 1.

Corner pattern - when the bold pattern is found the user should be able to clear dashes (highlighted). When they are not already cleared of course.

-	-	-	-

-	**-**	**1**	**0**
-	**1**	**1**	**0**
-	**0**	**0**	**0**

1 - 1 - 1 pattern - when the bold pattern is found the user should be able to clear the 2nd dash (highlighted square). When they are not already cleared of course.

-	-	-	-
1	**1**	**1**	1
0	**0**	**0**	0

I did not end up getting the autoplayer working but the above describes how I wanted my design to work.

Design Process

I originally started making the game by using a command line based UI. I found this was an easy way of ensuring that the game logic worked before starting to create the react based elements of the game. In terms of the react based elements I originally thought that the way to do it would be to create a button for each of the cells in the grid but this was far more complicated than I thought so I decided to use the canvas from the sample threepenny project as this provided an x and y coordinate when clicked on. This way I could keep the buttons simpler.

Something I struggled with during the initial implementation was clearing the cells. I kept getting infinite loops and I couldn't figure out the end condition. My solution in the end was creating an extra findNeighbours which found neighbours which had not been processed already. The data types were very easy to work with for the cells of the board and I also thought that the easily recursive functions were good to work with particularly for clearing cells.

In terms of debugging the player I found this was hard to do without any print statements. I had no idea what coordinates the functions were returning so when the play button did not reveal any squares I had to assume that the player had reached the end of the list of cells without finding a square to reveal. This made it difficult to find the correct logic for the player. I tried several different ways to get the player working but I couldn't get it functioning the way I wanted. The commented code in my Player.hs was another way I was attempting to get this working, using aforementioned patterns - corners and 1 to 1 to 1.

Conclusion

This was an enjoyable albeit quite challenging project to work on. I enjoyed creating the initial implementation of the game but I struggled with debugging the autoplayer.