Zippora Cahn
CMSC 313
Professor Teixeira
4/16/18

P3 Report

At first, I tried modifying the 3x3 tic-tac-toe code that was provided at the beginning of the assignment. I then decided to try writing my own code to make it easier to keep track of all the different components of the code.

To create the boards from any given input, I had to create a function that would be able to take the given input and convert it to the desired size board. I used the debugger prompt from the code given to us. I then created the prompt for users to pick a size of board they wished to play. I stored their response into buffer2 and had the program call minePlacer twice to allow each player to place a mine. This function has the same validation requirements that the regular play function has except that nothing is visibly displayed on the board. The valid values are stored, based on whose turn it is, in either mine1 or mine2. Then call_print board is called. In the print_board function I created a variable named dim that would take the input from buffer2 and convert it from ascii into decimal so I could use it in my calculations for the dimension of the board. This allows me to generate the board for any size board imputed by the user. The print_board function works like most tic-tac-toe print boards. There are two for loops, one nested in the other. The first for loop deals with the rows and the second deals with the columns. Since I would have to compare the rows and columns with the dimension that was stored in dim, I created a variable named dim which is just the dim – 1. Until the loop condition is complete the board will continue to create “spaces” and “pipes”. I had to create a separate for loop to handle the dashes that were required to make the underline between the rows.

Once the board was created, a function named play is activated. Play looks at the character the user inputted for the size of the board that is still stored in buffer2. Based on the character inputted, either play1, play2, or play3 will be activated.

Each play functions similarly. The user is prompted with the appropriate valid inputs, 0-8, 0-15, and 0-24. The user is then prompted that player X goes first. Whatever the user inputs is then stored in buffer3. I created a variable named player that keeps track of each player’s turn. I also created a variable to handle whether the place value is a valid input. If the value is not valid, the proper invalid function is called and the user is prompted to input a valid input.

To check whether a player chose for their next turn the other player’s bomb spot I have a check mine function that is called before the players mark is added to the board.

The check mine function compares the value(space) the player chose against the value of the opposing player’s mine. If they are equal, a message will show stating that the opposing player won. It also displays the board with the exploded bomb and the unexploded bomb. Then exits the game.

Zippora Cahn
CMSC 313
Professor Teixeira
4/16/18

After each turn, checkwin1 is called. Checkwin1 checks by rows first, then columns, then diagonal spots top left to bottom right, then finally diagonal spots bottom left to top right. If at any point one of the conditions for winning is met, the game produces the correct win notification for the player who won. The game then exits.