Project 2

BugSweeper

(MineSweeper clone)

CSC-5 Winter 2018

Joel Avalos

**Introduction:**

BugSweeper is a MineSweeper clone, replacing mines with “bugs” to avoid triggering. A 10x6 grid with 60 “X”s representing individual spots is shown to the user. The point of the game is to clear each space without finding a bug, labelled as a “B” in the game. When the user has successfully cleared each space without encountering any bugs, then the game has been won. However, if the user uncovers a bug, then the game is over. Once you successfully clear a space, a number will display, signifying the number of mines adjacent (specifically a 3x3 grid) to the space cleared. This is a clue to aid the user in avoiding bugs, and ultimately winning the game.

**Rules:**

Entering in your choice is comprised of an “A”, “B”, “C”, “D”,”E”, or “F”, followed by the number (1-10) of the spot you wish to clear. The letters represent each row of the grid, and the number represents the column in that row. For example, entering in an A1 would clear the top left space on the grid. If that space is not a bug, then a number will display, that number is dependent on how many bugs are nearby. So say the only neighboring space with a bug is A2, then the A1 space will display a “1” after it is cleared. If the space is a bug, a “B” will display and the game will be over!

**Summary of Development:**

Project size: Approximately 575 lines of code.  
Number of Variables: Approximately 20, excluding the array and vector variables.

Differences from MineSweeper:

Most of the game is very similar to MineSweeper. The key differences are that there is no way for the user to “flag” mines, and that the first space entered by the user is not automatically a cleared space.

Coding the Game:

Bringing in functions and arrays simplified the code tremendously. Again, the most difficult part of coding the game was displaying the number of adjacent mines to a cleared space. I wondered how I would be able to code such a function without having to individually take every space and test all of the adjacent spaces. This, of course, would have taken many lines of code and much time. Fortunately, I realized there were patterns to this checking, and used them to make the function easier to code. Another problematic part of the coding was the input/output record keeping function. I found that it would take too much time to implement a

**Example inputs/outputs**

**Pseudocode:**

*Initialize*

*Display start menu*

*If the user starts the game, display the grid and wait for an input.*

*Else if the user wishes to see rules, display rules.*

*Else, quit the program.*

*If the user input has an “A”, clear the numbered space in row A.*

*Else if the user input has a “B”, clear the numbered space in row B.*

*Else if the user input has a “C”, clear the numbered space in row C.*

*Else if the user input has a “D”, clear the numbered space in row D.*

*If the space is a bug, output a “B” and end the game.*

*Else, keep asking the user for input until a bug is triggered, or all spaces except bugs have been cleared, in which case the game is won. Return to start menu.*

**Flowchart: see project Folder**