Project 1

BugSweeper

(MineSweeper clone)

CSC-17A Spring 2018

Joel Avalos

**Introduction:**

BugSweeper is a MineSweeper clone, replacing mines with “bugs” to avoid triggering. There are multiple game grids, dependent on a user selected difficulty, and each grid begins with “X”s representing individual spots. 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 (this can be anywhere from 1-15) 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 the cleared spot. For example, suppose the only neighboring space to A1 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 1100 lines of code.  
Number of Variables: Approximately 30, excluding the array and vector variables, as well as structures.

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” spaces they suspect to be mines, and that the first space entered by the user does not automatically clear an area to assist the user and give a “heads-start”, so to speak.

Coding the Game:

I modified this project from CSC-5, and the concepts included from Chapters 9-12 of Gaddis simplified the code tremendously. Structures were used to condense game variables into, dynamic memory allocation helped to introduce a difficulty mechanic to the game. String library functions assisted in verifying user inputs. The most challenging part of coding the game was introducing the option to modify the difficulty of the game. Along with this, I did not have the time to tinker with the record saving functions of the game, and add concepts from Chapter 12. As a result, there is a heavy lack of concepts from that chapter. Aside from this, I feel that there is a lot of new concepts used in this version of the game.

**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 change difficulty, show the menu to change the difficulty.*

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

*If the user wishes to keep track of their scores, ask them to enter their name.*

*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.*

*…And so on for rest of the grid rows.*

*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.*

**Flowcharts: see project Folder**