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:**

The user must input a space to clear. 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. For example, suppose the only neighboring space to A1 that is a bug is A2, then the A1 space will display a “1” after it is cleared. If a space is entered and it is a bug, a “B” will display and the game will be over! The game is won when the user has successfully cleared all spaces except for bugs.

**Summary of Development:**

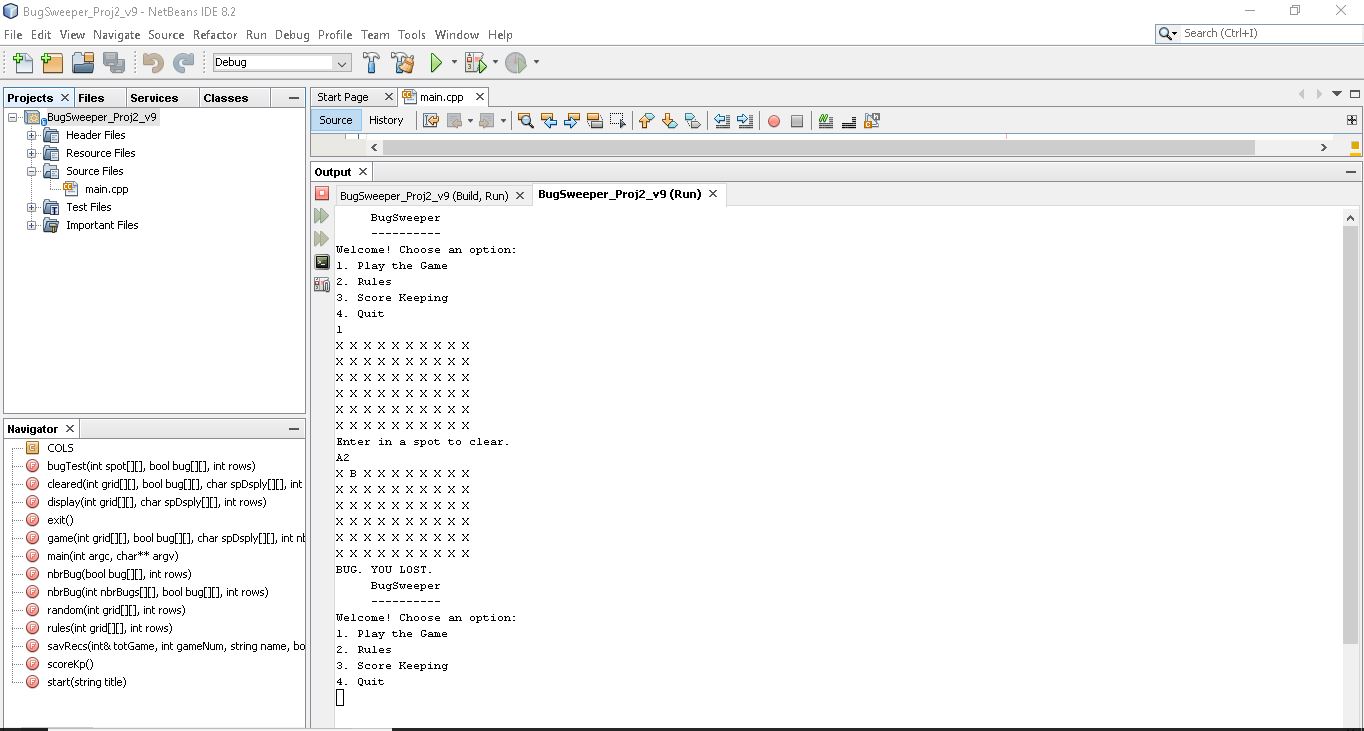
Project size: Approximately 575 lines of code.  
Number of Variables: Approximately 20, excluding the array and vector variables.

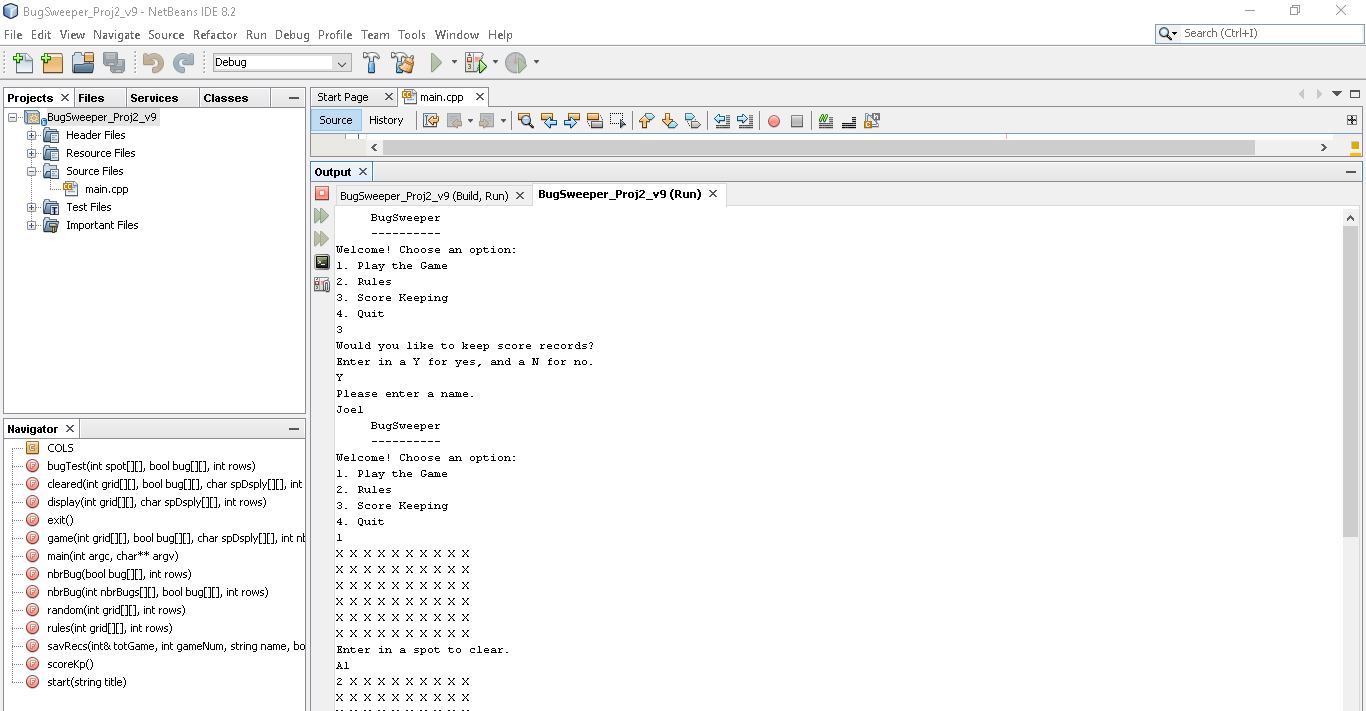
Differences from MineSweeper:

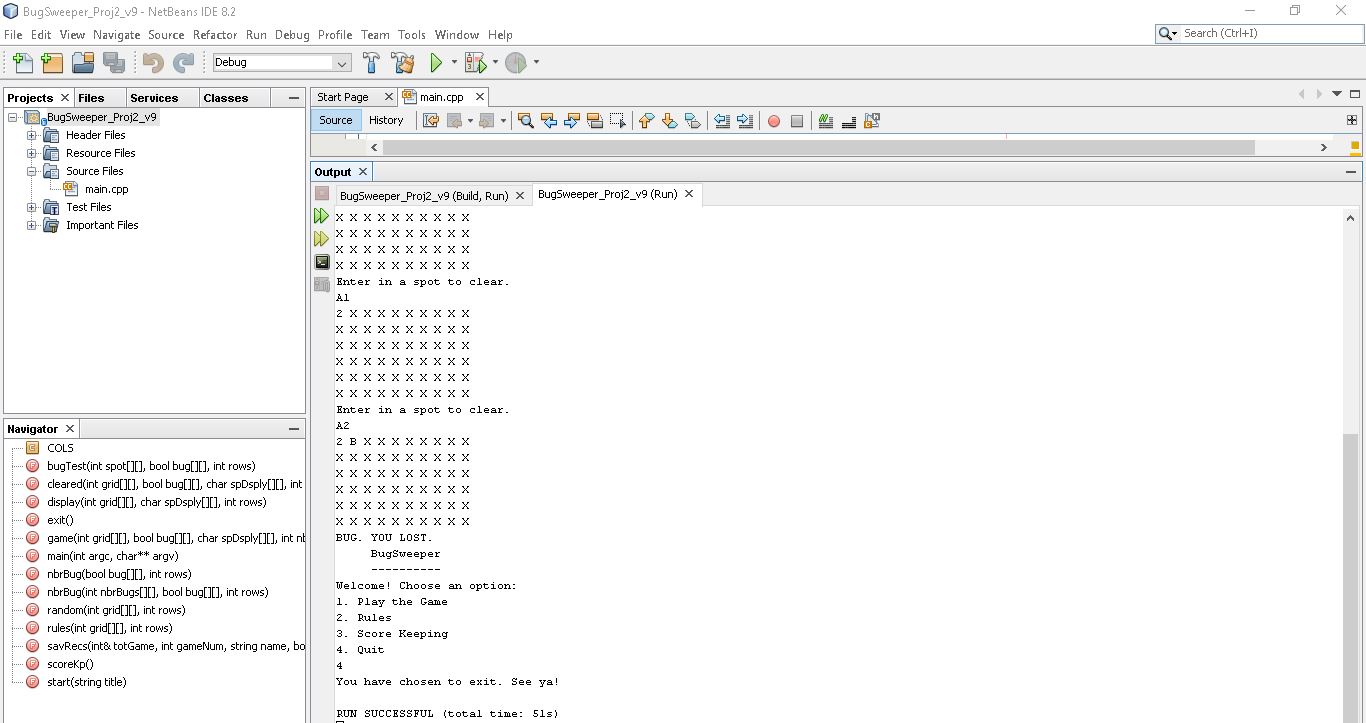
Most of the game is very similar to MineSweeper. The key differences are that there is no way for the user to “flag” mines, and that the first space entered by the user is not automatically a cleared space.

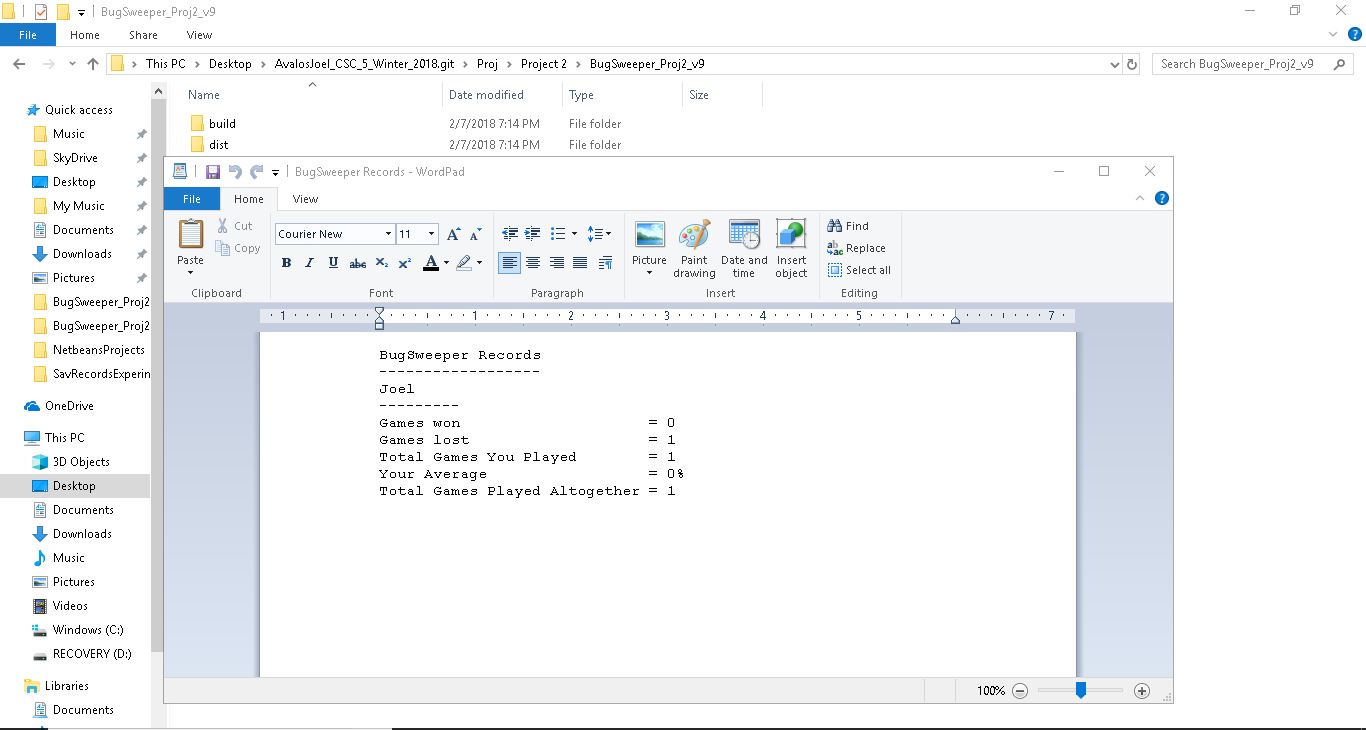
Coding the Game:

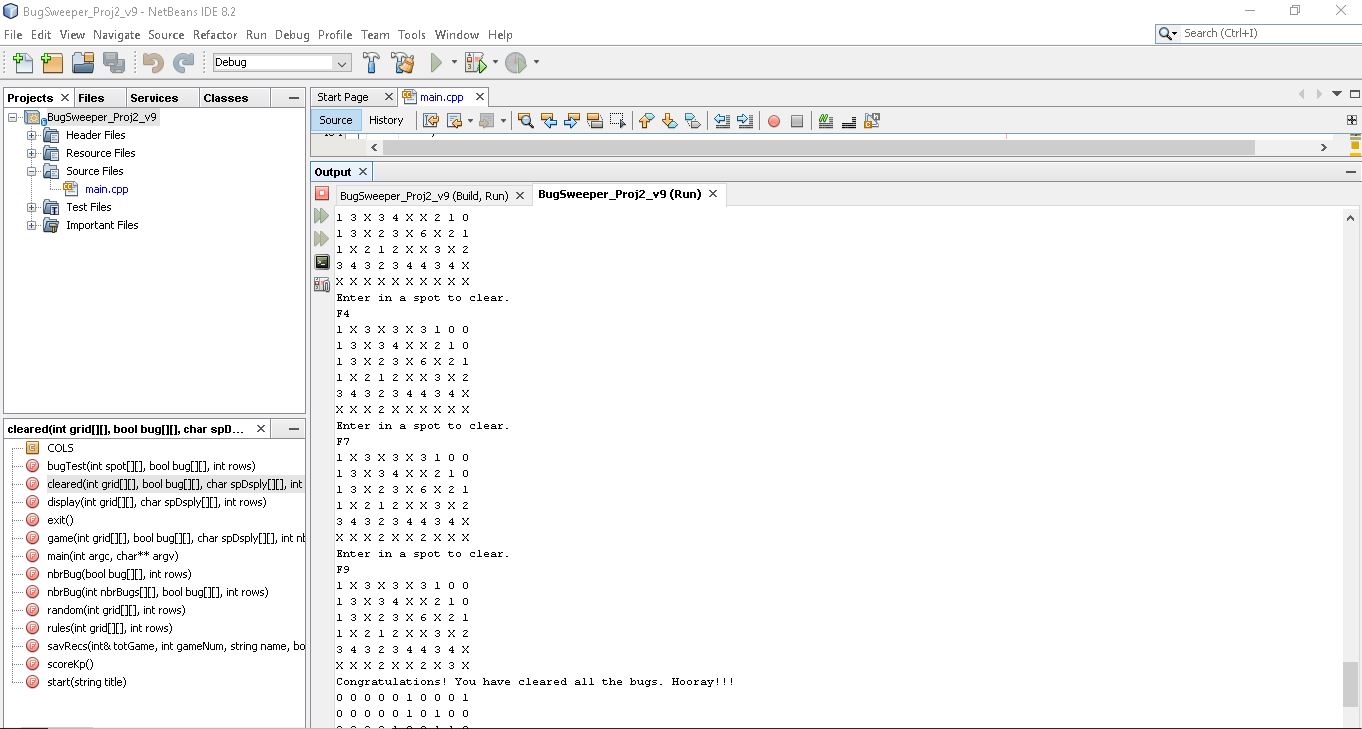
Bringing in functions and arrays simplified the code tremendously. Again, the most difficult part of coding the game was displaying the number of adjacent mines to a cleared space. I wondered how I would be able to code such a function without having to individually take every space and test all of the adjacent spaces. Sixty spaces means doing that process sixty times! 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 saving system where the user-inputted names were remembered and recalled after every time the program closed. Due to time constraints, I had to settle for a saving system that could keep track of the total number of games, as well as output a few stats for a single program run.

**Example inputs/outputs: **

****

****

****

****

**Pseudocode:**

*Constant for the number of columns in the 2-D Array*

*Function Prototypes*

*Function to randomize each spot on grid.*

*Function to test whether a spot is a bug.*

*Function to calculate number of Bugs on grid*

*A well as to activate Bug spaces.*

*Function to calculate neighboring bugs around each space*

*Function to run the game's Start Menu*

*Function to run the game*

*Function to clear each user-inputted space*

*Function for displaying the grid.*

*Function for displaying the Rules of the game.*

*Function for keeping records*

*Function for saving records*

*Function to exit the program*

*Execution Begins Here*

*Seeding the random number generator*

*Declaring Game, Grid, and Array Variables*

*Win condition of the game*

*Losing condition*

*Flag for keeping scores*

*Game title*

*Rows in the grid.*

*Grid*

*Bug or not*

*The total number of bugs in the game*

*Max number of games to record keep is 100*

*Array to hold number of games won*

*Vector to hold number of games lost*

*Count of how many games user has played*

*Total games played*

*Declaring Display Variables*

*What a space will display after being cleared.*

*The number of bugs neighboring a space on the grid*

*Declaring User-inputted Variables*

*User-inputted choice for the row that the space they want to clear is in*

*User-inputted choice for the space they want to clear*

*Menu choice*

*Name for record keeping*

*Start the Game loop*

*Initializing Variables*

*Set the initial win condition to false*

*Set the initial lose condition to false*

*Title of Game!*

*Open file for keeping total games played*

*Initializing random numbers and display values*

*Call random function to randomize the grid*

*Loop to set each space's initial Bug value to false*

*Loop to set each space's initial display to an X*

*Loop to set game wins to false*

*Initializing the Bugs*

*Call the bugTest function to arm bugs*

*Loop to display the grid with bug locations for debugging purposes*

*Call nBug function to add up the total number of bugs in the game.*

*Start Menu*

*Call start menu function*

*Ask user for input*

*This is the switch case for the menu*

*Run game*

*Display rules*

*Option to keep records*

*Or, exit program*

*Display menu while user inputs menu choices 1 or 2*

*Exit stage right!*

*Randomizing the grid.*

*Going through every element in array to randomize*

*Testing each space for bugs.*

*Bug criteria: if the random number*

*Assigned to a space is cleanly divisible by 3*

*(no remainder), then that space is a bug*

*Set bug test to true*

*Adding up the number of bugs on the grid*

*Checking bool value for bug*

*Counting number of bugs for the win condition*

*Testing neighboring spaces on grid for bugs*

*Testing neighboring spaces of column 0 spaces for bugs*

*Testing neighboring spaces of column 1 spaces for bugs*

*Testing neighboring spaces of column 2 spaces for bugs*

*Testing neighboring spaces of column 3 spaces for bugs*

*Testing neighboring spaces of column 4 spaces for bugs*

*Testing neighboring spaces of column 5 spaces for bugs*

*Testing neighboring spaces of column 6 spaces for bugs*

*Testing neighboring spaces of column 7 spaces for bugs*

*Testing neighboring spaces of column 8 spaces for bugs*

*Testing neighboring spaces of column 9 spaces for bugs*

*Displaying start menu to user*

*Game function*

*Clearing a space function*

*Calling nbrBugs function to calculate display values*

*Static local variable for number of spaces the user has cleared*

*The win condition of the game*

*If the number of cleared spaces plus number of bugs is equal to 60*

*Then the user has won the game.*

*Function to display grid.*

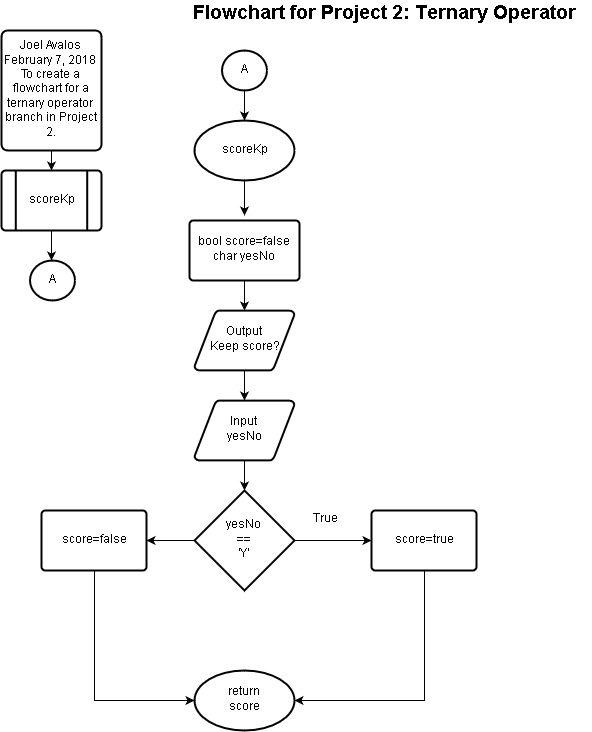
*Function to display rules.*

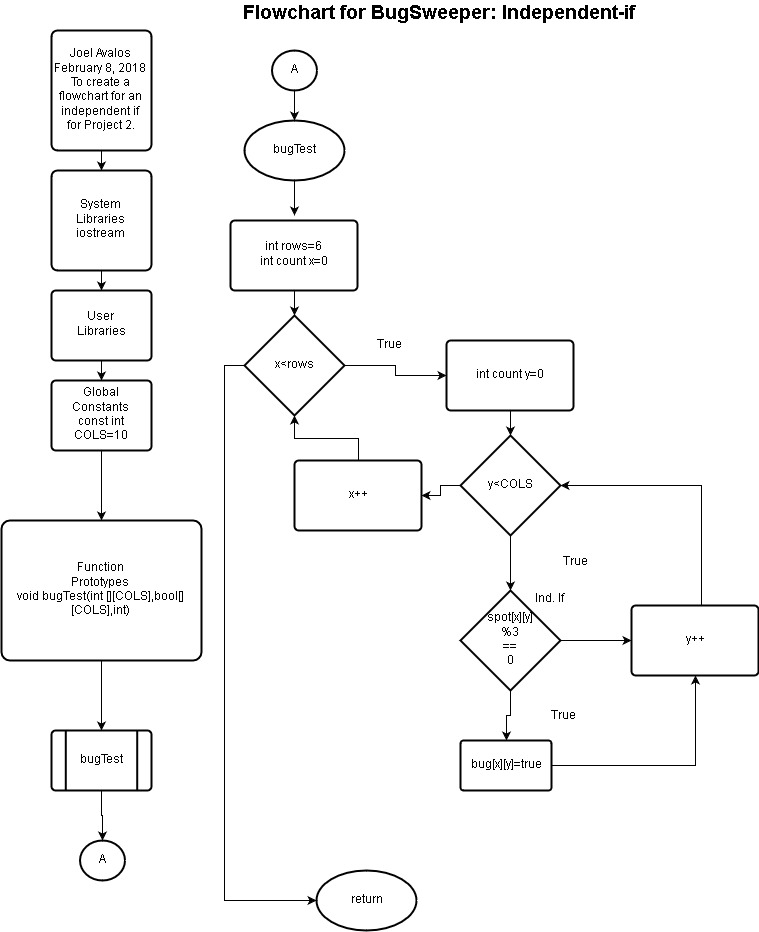
*Function for prompting user to keep scores*

*Function to save records*

*Exit function*

**Flowcharts:**

****

**

