SDV503 Assessment 2

Minesweeper Game Implementation – Caleb Eason

Contents

[1. Implementation Workflow 2](#_Toc135605754)

[1.1. Program Objective 2](#_Toc135605755)

[1.2. Testing Methodology 2](#_Toc135605756)

[1.3. Workflow Diagrams 3](#_Toc135605757)

[1.3.1. Board generation 3](#_Toc135605758)

[1.3.2. Mine placement 4](#_Toc135605759)

[1.3.3. Mine counter 5](#_Toc135605760)

[1.3.4. Check for mine 7](#_Toc135605761)

[1.3.5. Place flag 7](#_Toc135605762)

[1.3.6. Uncover cell 8](#_Toc135605763)

[1.3.7. Recursive uncover 9](#_Toc135605764)

[1.3.8. Check win conditions 10](#_Toc135605765)

[2. Implementation Requirements 11](#_Toc135605766)

[2.1. Board Generation 11](#_Toc135605767)

[2.2. Mine Placement 12](#_Toc135605768)

[2.3. Mine Counter 14](#_Toc135605769)

[2.4. Check for Mine 15](#_Toc135605770)

[2.5. Place Flag 16](#_Toc135605771)

[2.6. Uncover Cell 17](#_Toc135605772)

[2.7. Recursive Uncover 18](#_Toc135605773)

[2.8. Check Win Conditions 19](#_Toc135605774)

[3. Game Requirements Programming 20](#_Toc135605775)

[3.1. Generating the Gameboard 20](#_Toc135605776)

[3.2. Placing Mines 20](#_Toc135605777)

[3.3. Counting Nearby Mines 21](#_Toc135605778)

[3.4. Displaying the Board 22](#_Toc135605779)

[3.5. Check For Mine 26](#_Toc135605780)

[3.6. Uncovering a Cell 28](#_Toc135605781)

[3.7. Recursive Uncover 28](#_Toc135605782)

[3.8. Toggle Flag 29](#_Toc135605783)

[3.9. Checking Win Conditions 29](#_Toc135605784)

[3.10. Game Settings 31](#_Toc135605785)

[3.11. Begin Game 34](#_Toc135605786)

[3.12. Select Action 34](#_Toc135605787)

[3.13. Root 36](#_Toc135605788)

[4. Game Program Testing and Debugging 36](#_Toc135605789)

[4.1. Testing the pre-game menu 37](#_Toc135605790)

[4.2. Testing the initialisation 37](#_Toc135605791)

[4.3. Testing game loop 40](#_Toc135605792)

[4.4. Testing end game 46](#_Toc135605793)

[5. References 47](#_Toc135605794)

# Implementation Workflow

## Program Objective

The objective of this program is to create a simple, playable game of minesweeper. The game should use a 3 by 3 game board with a random number of mines placed randomly within the board. Each turn the game board will be printed to the console. The user can interact with the game by typing commands into the console. The user should be able to uncover cells within the board or place a flag on a cell to mark a mine. The goal for the user is to uncover every cell that does not contain a mine, at which point the game will end. If at any point the user uncovers a mine, the game will end, and the user will lose.

## Testing Methodology

I will develop the program by breaking it down into core functionalities and developing each function individually. I will use the agile SLDC methodology to iteratively develop and improve each function, regularly testing the behaviour and removing any bugs. Once the functions are developed, I will combine them into one program and develop the main gameplay loop to call the different functions as required. I will continue to iterate over the program, fixing bugs and adding any additional functionality if deemed appropriate.

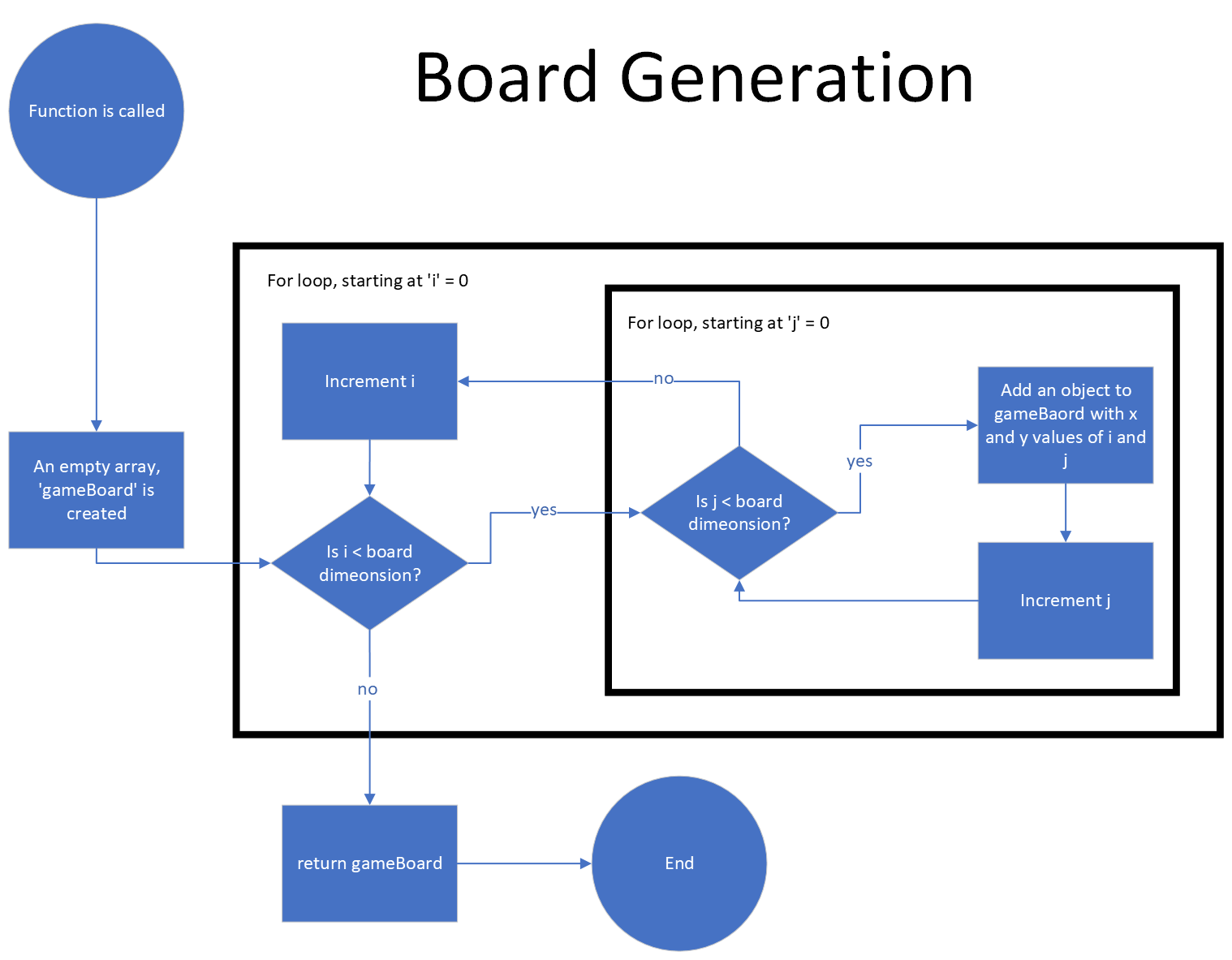
Once the program is fully developed, I will comprehensively test the program with various expected and unexpected user inputs, comparing the expected results to the actual. I will use the testing sheet I developed for assessment one to document my testing.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | |  | **Test Case Description** | |  | | | | | |
| **Created By** | |  | **Reviewed By** | |  | | **Version** | |  | |
|  |  |  |  |  |  |  |  |  |  |  |
| **QA Tester’s Log** | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Tester's Name** | |  | **Date Tested** | |  | | **Test Case (Pass/Fail/Not Executed)** | |  | |
|  |  |  |  |  |  |  |  |  |  |  |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | |
| 1 |  | | |  | 1 |  | | | | |
| 2 |  | | |  | 2 |  | | | | |
| 3 |  | | |  | 3 |  | | | | |
| 4 |  | | |  | 4 |  | | | | |
|  |  |  |  |  |  |  |  |  |  |  |
| **Test Scenario** |  | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | **Pass / Fail / Not executed / Suspended** | | |
|  |
| 1 |  | |  | |  | | |  | | |  |
| 2 |  | |  | |  | | |  | | |  |
| … |  | |  | |  | | |  | | |  |

## Workflow Diagrams

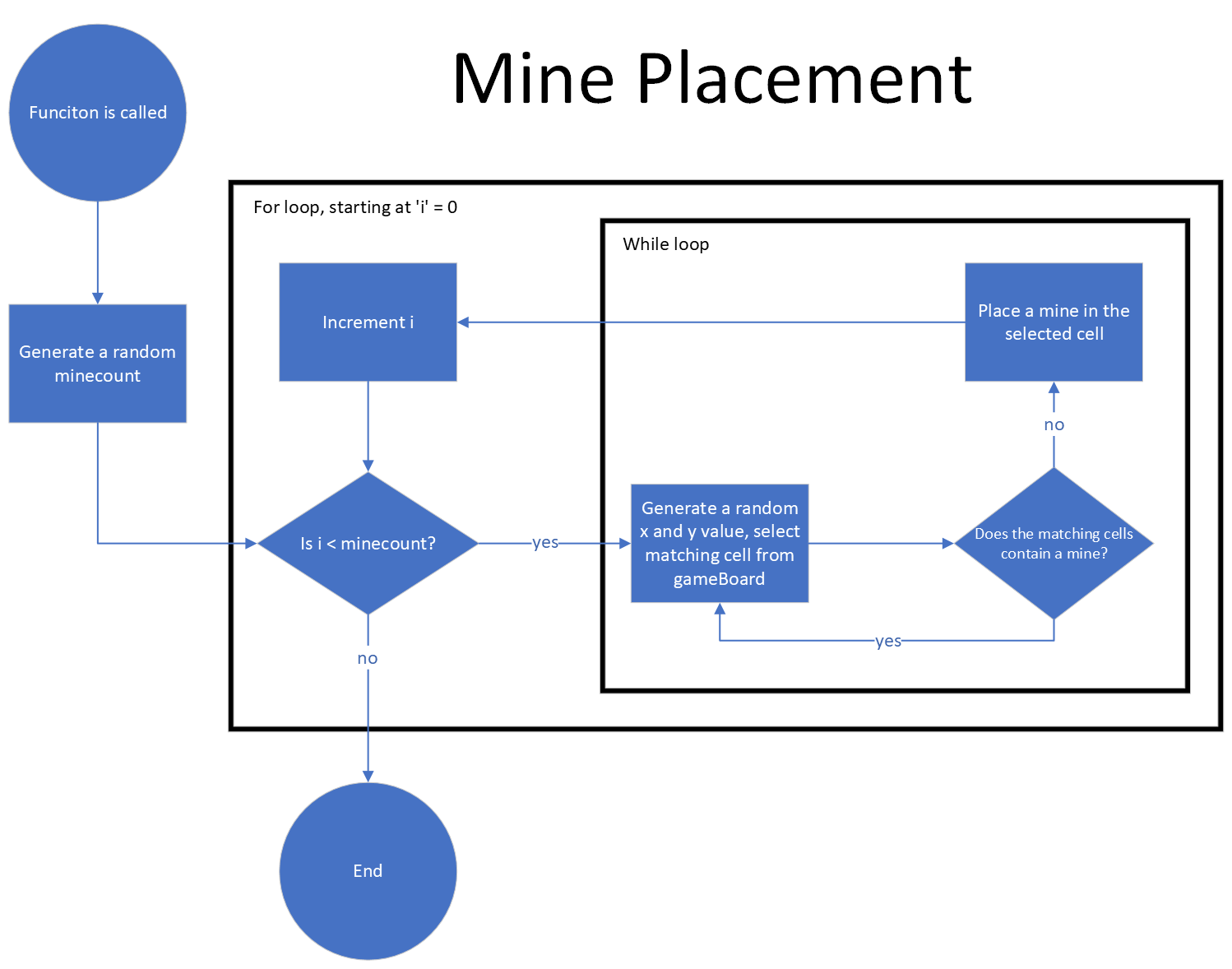
### Board generation

This function generates an array to store a variable size minesweeper game board. The generated array will have global scope, allowing other function to easily access it.



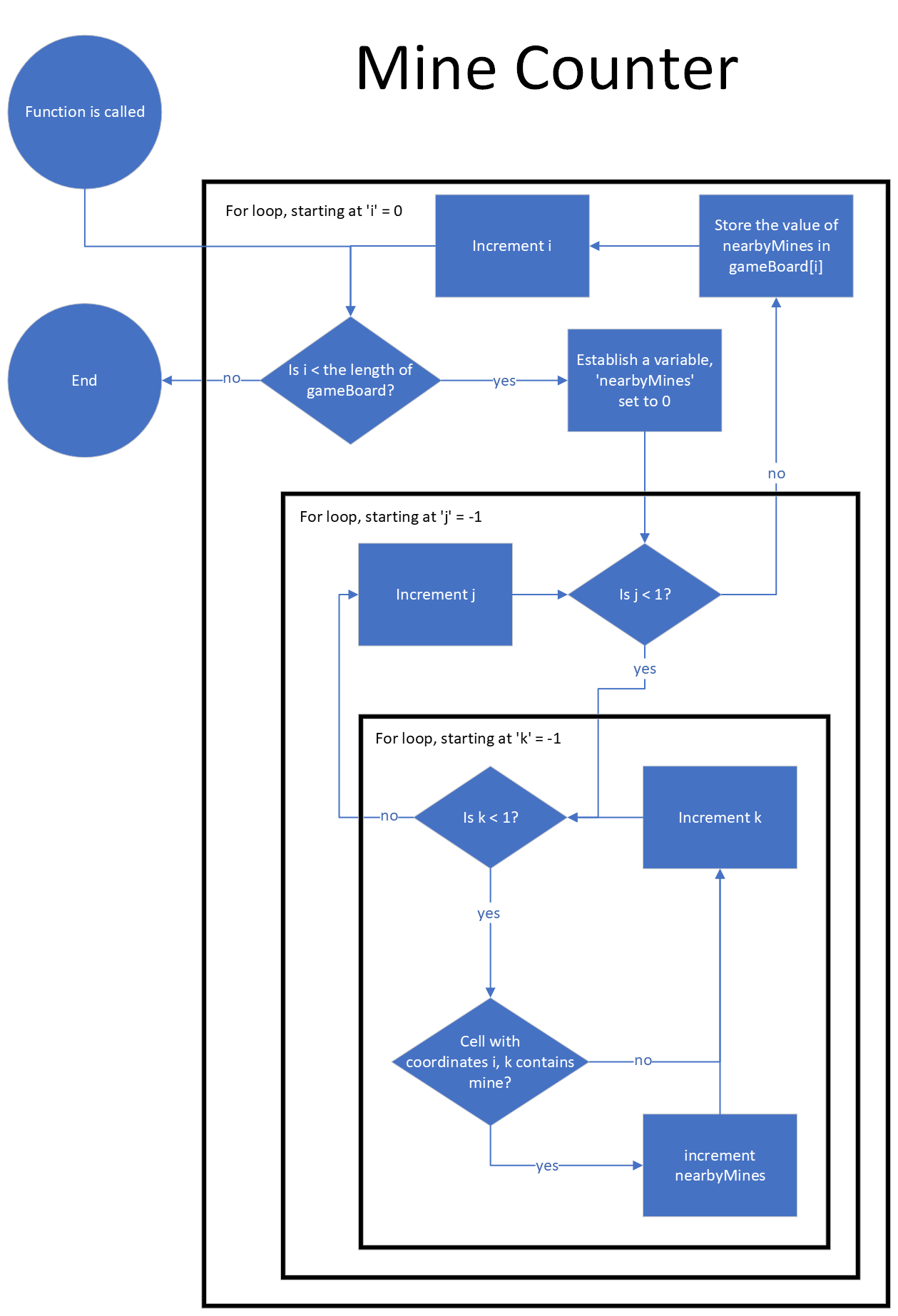
### Mine placement

This function randomly places a random number of mines on the board, avoiding duplicates. Uses the previously created global variable ‘gameBoard’.



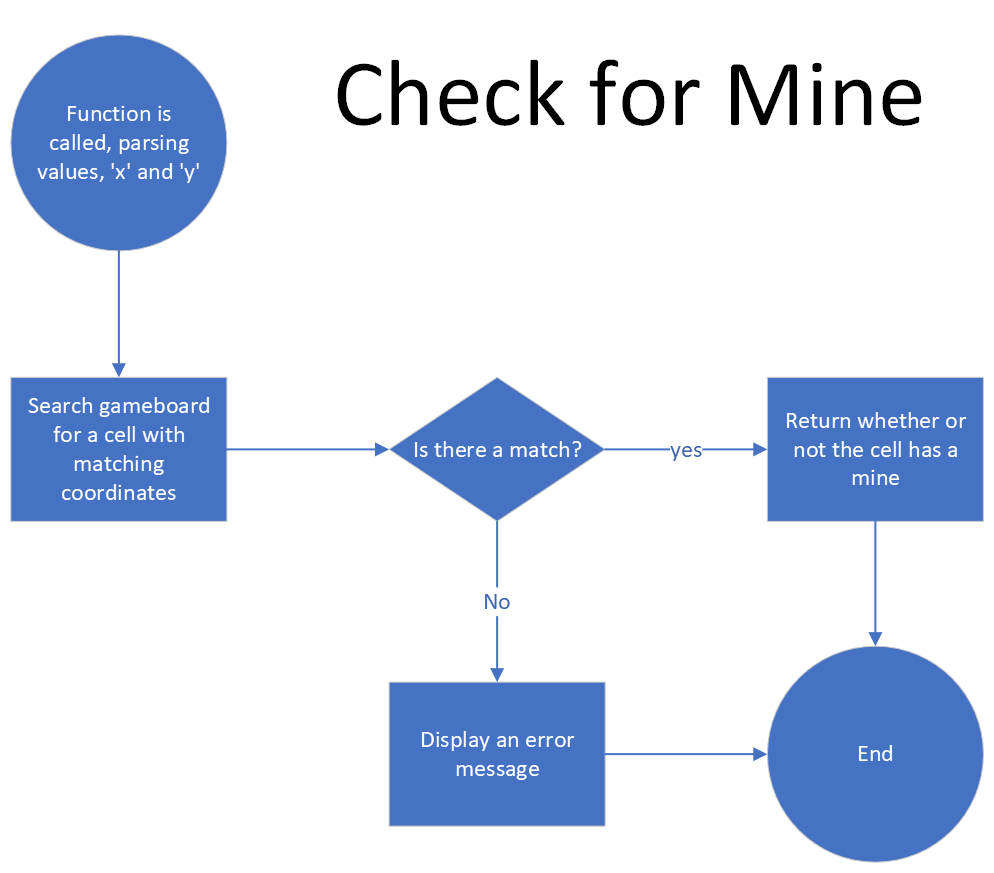
### Mine counter

This function iterates through each cell on the game board, and counts how many adjacent cells containing a mine, storing the value in gameBoard.



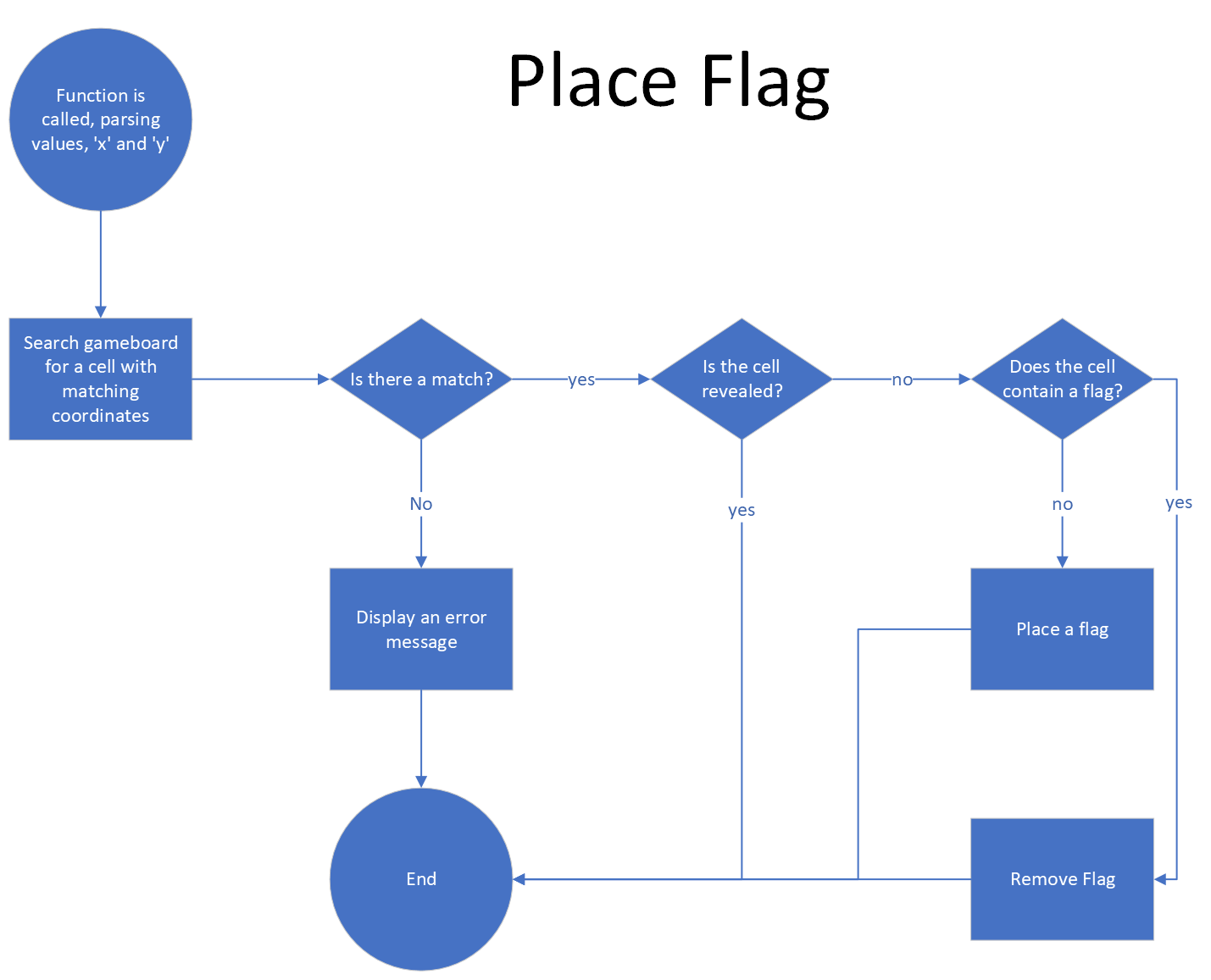
### Check for mine

This function takes an inputted x and y value, tests if there is an object in gameboard with those values and returns whether the cell contains a mine.



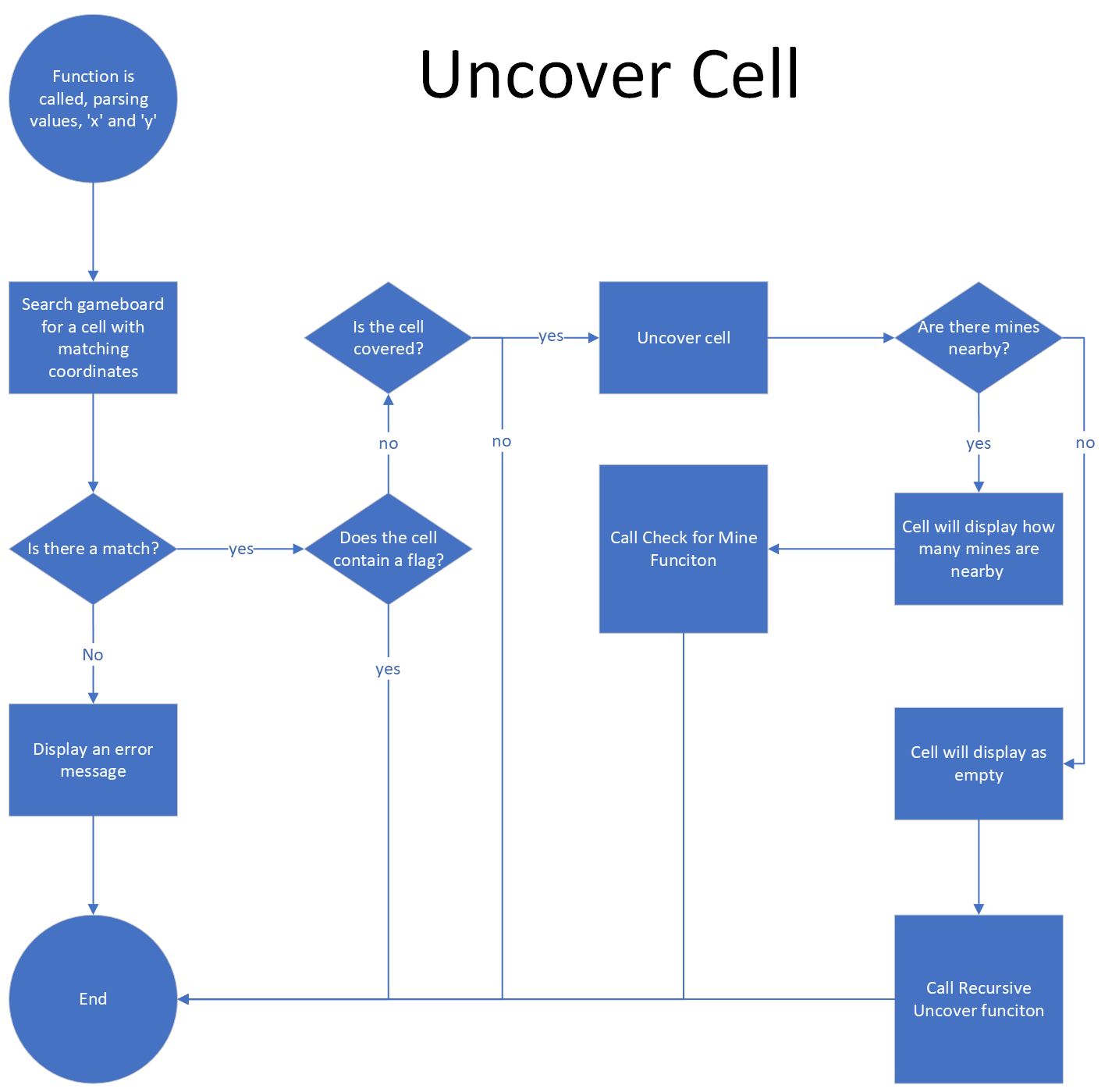
### Place flag

This function takes an inputted x and y value, tests if they match an object in gameboard, then places or removes a flag depending on the cell’s current state. Has no effect on uncovered cells.



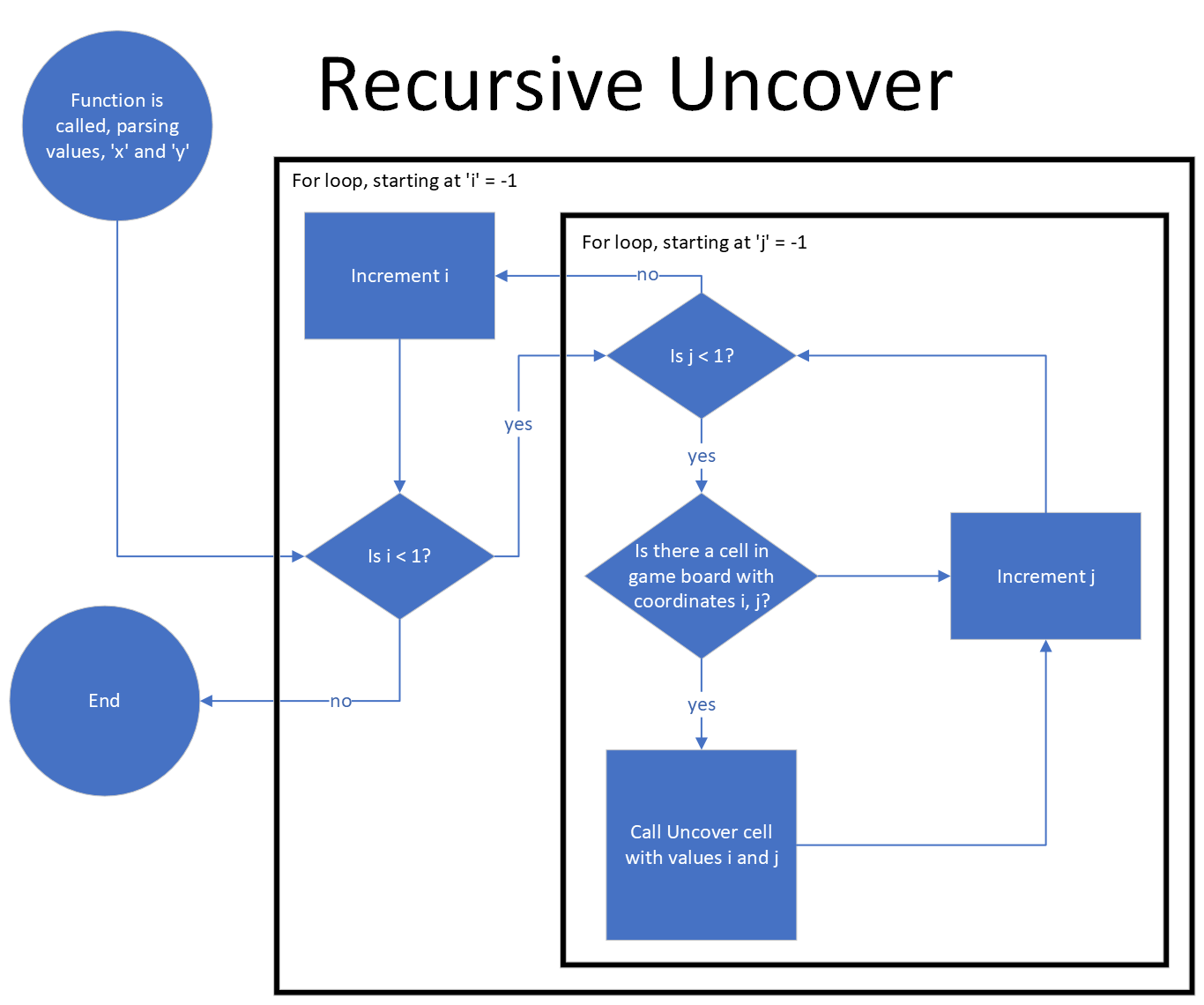
### Uncover cell

This function takes an inputted x and y value, tests if they match a cell in gameboard, then uncovers the cell if it is not flagged or uncovered already. If a cell is uncovered the number of nearby mine will be displayed. If this number is 0, the Recursive Uncover function will be called and a blank tile will be displayed instead. After a cell is uncovered the Check for Mine function is called.



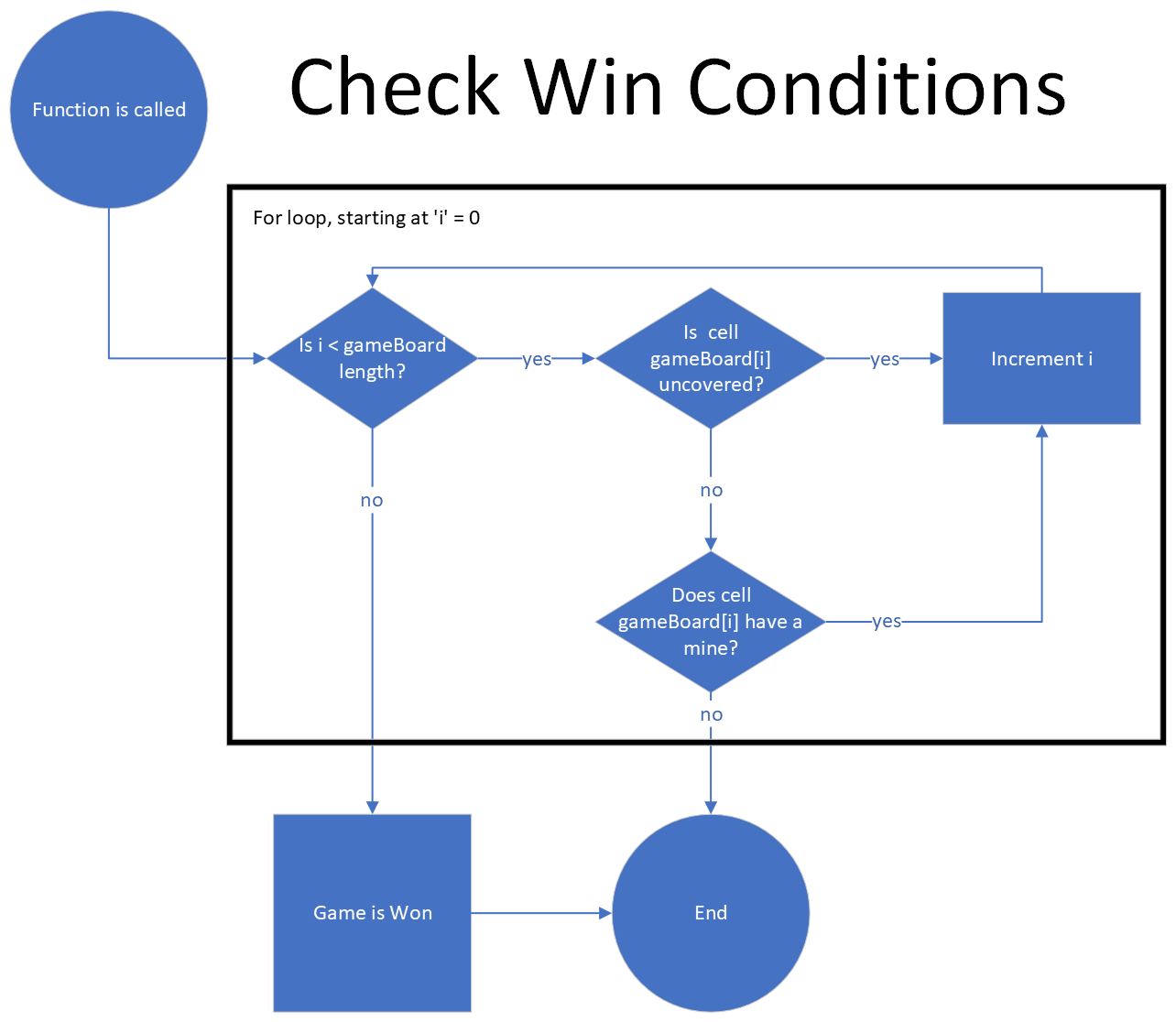
### Recursive uncover

When called by the Uncover Cell function, this function will call Uncover Cell on all cells surrounding an inputted x and y value.



### Check win conditions

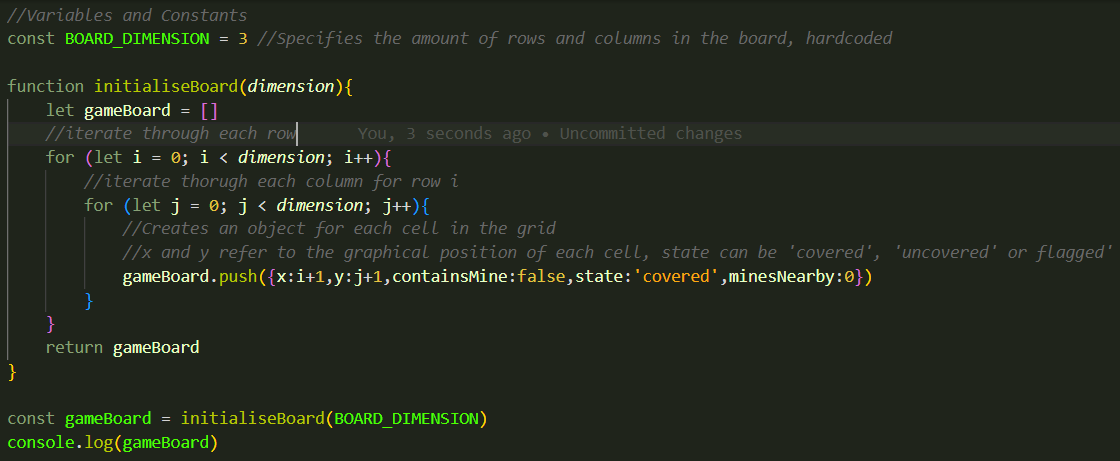
This function checks if the user has uncovered every cell that does not contain a mine, if they have, the game is won.



# Implementation Requirements

I decided to develop my code to work with variable sized game boards instead of just 3 by 3. Initially I just left a few functions open to expandability, but I decided it wouldn’t be too much more work to build the entire program to accommodate variable board sizes.

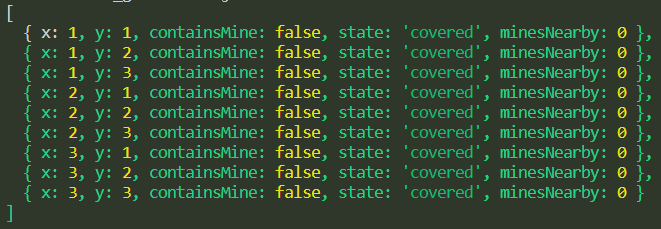
## Board Generation



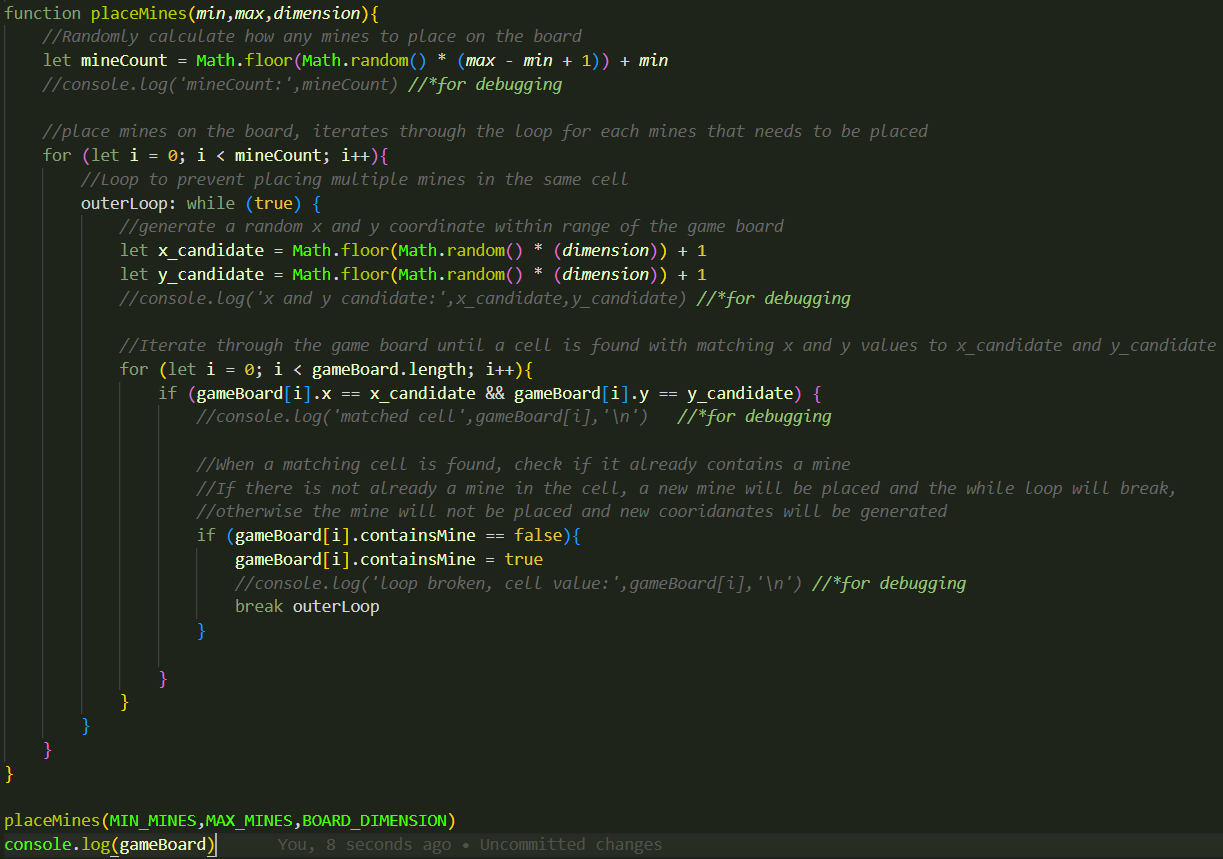
The code uses nested for loops to iterate through a specified number of rows columns. In this case the board dimension is set to 3 (meaning 3 by 3). The first for loop will create each row of the grid. For each row the second for loop is run, creating each column. Each time the 2nd for loop runs, it will append an object to an array. The iteration numbers of the two for loops (i and j) will be set to properties x and y in the object to represent grid coordinates.

I decided to use a list of objects to store my game board data as opposed to a two-dimensional array. This allows me to store all of a cell’s information in one place instead of using a different array for each property (coordinates, whether it contains mine, what state the cell is in and how many mines are adjacent to the cell).

Here is a sample output form this function.



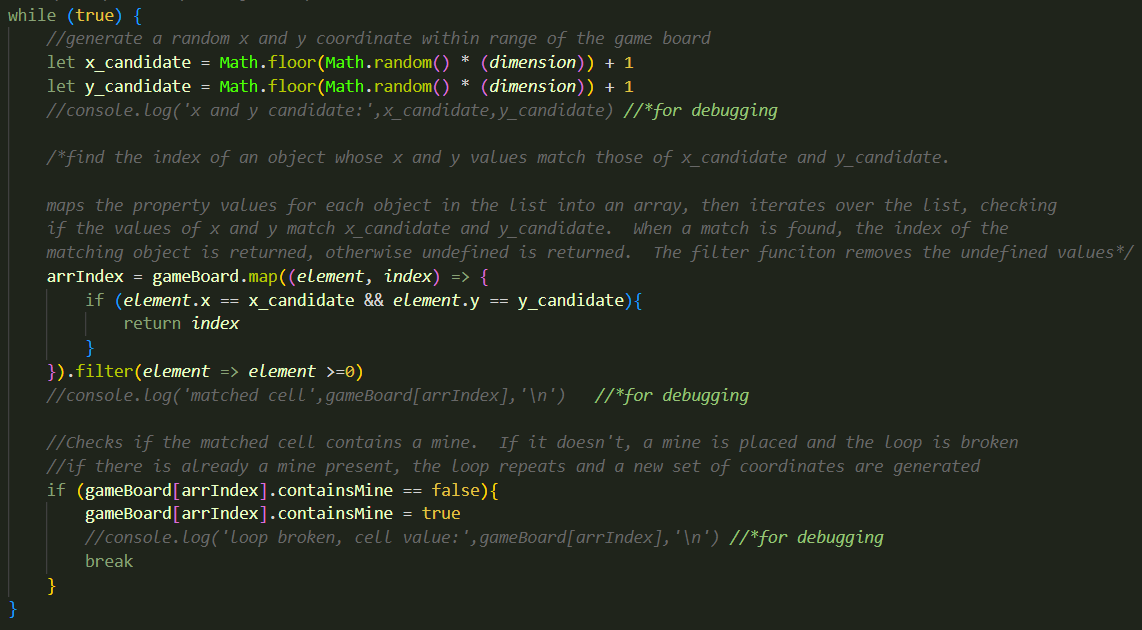
## Mine Placement



This function starts by generating a random number of mines to be placed, the range of which is specified with the variables, min and max. The function then runs through a for loop until it reaches the mine count, placing a mine on the board for each iteration. Inside the for loop the program generates a random coordinate pair then runs a nested for loop. The nested for loop runs through each cell in the game board until a cell is found with matching coordinates to the randomly generated ones. Then, the program checks if the selected cell already contains a mine, If it does not, a mine is placed and the for loop moves on to the next iteration. If there is already a mine, a duplicate mine will not be placed, and the same iteration will be run again (using a while loop).

I used this method to place mines to ensure number of mines paced is always consistent with the mine count generated. If there was no system in pace to avoid selecting the same cell multiple times, the mine count would not be accurate. I made ‘gameBoard’ a global variable so it can be easily accessed and modified from any function.

I later improved this code by using the .map function.



This version of the code allows me to find the array index of a cell that matches the generated coordinates, without the need for a for loop.

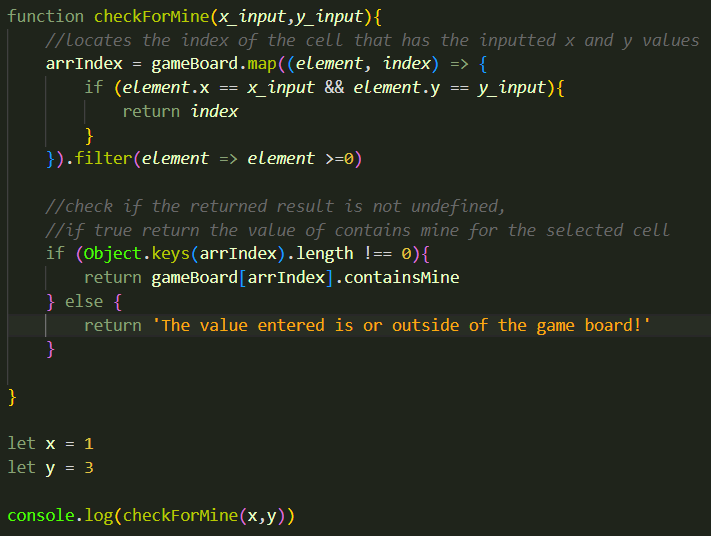
## Mine Counter



I chose to calculate the adjacent mine count for each cell at the start of the game as opposed to generating it individually each time a cell is uncovered. This way, I only need to call this function once at the start of the game, which will save a bit performance during gameplay.

This function uses a for loop to iterate over every cell in the gameboard. The function begins each iteration by declaring 3 variables. ‘nearbyMines’ keeps track of how many mines are adjacent to the cell, and ‘cx’ and ‘cy’ hold the coordinates for the current cell (gameboard[i]). The main for loop contains a nested for loop, which itself contains another nested for loop. These nested for loops iterate through all the adjacent cells to the selected cell. Each time a mine is encountered the value of nearbyMines is increased by one. A if statement is used to check the object is not null, as running this code for cells on the edge of the board will return undefined values when the for loop tries to select a non-existent cell. E.g., the cell to the left of cell 1,1 would be 0,1 which does not exist. Once the nested for loops have finished, the value of nearbyMines is stored in the gameBoard.

## Check for Mine



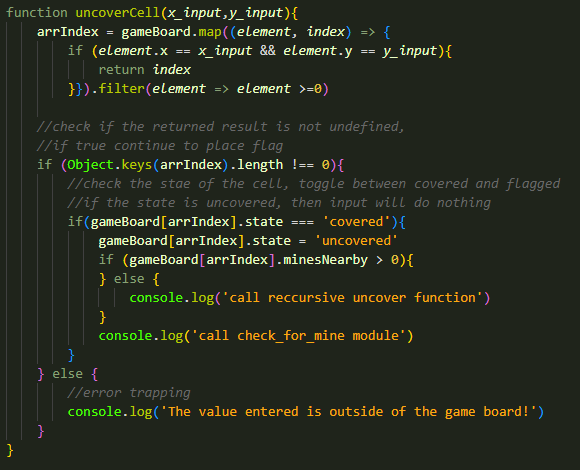
This function takes an inputted coordinate pair and finds the matching cell in the gameboard. If a matching cell is found, it will return the whether the cell contains a mine. If no match is found it will print an error message to the console.

## Place Flag



This function takes an inputted coordinate pair and finds the matching cell in the gameboard. If a matching cell is found, it will check the current state of the cell. If the cell state is covered, the program will change it to flagged. If it is flagged, it will be changed to covered. If the cell is in any other state, nothing will happen. If no matching cell is found the function will print an error message to the console.

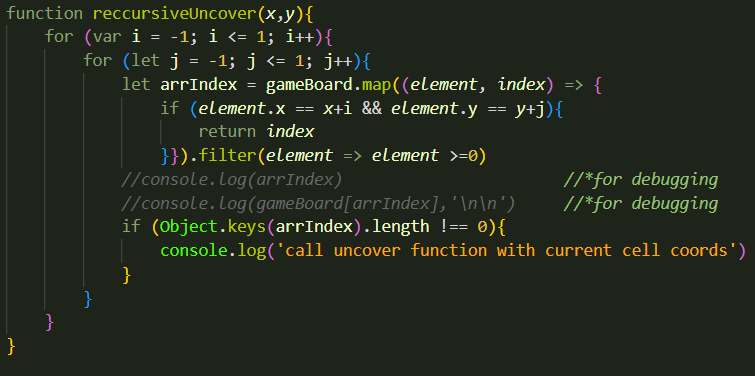
## Uncover Cell



This function takes an inputted coordinate pair and finds the matching cell in the gameboard. If a matching cell is found, it will check the current state of the cell. If the cell state is covered, the program will change it to uncovered. When a cell is uncovered, the program will check how many mines are adjacent to the cell. If there are no adjacent mines, the function will call the Recursive Uncover function, parsing the inputted coordinates. The program will then call the Check for Mine function. If no matching cell is found, an error message will be displayed in the console.

I chose to make the recursive uncovering behaviour its own function. This allows the two functions to recursively call each other, eliminating the need for any difficult logic.

## Recursive Uncover



When called by Uncover cell, this function takes the parsed x and y values and iterates over all adjacent cells using nested for loops. In each iteration of the innermost for loop, a check is performed to make sure the selected cell is not outside of the gameboard. If it succeeds, uncover cell is called on that cell. This means this function will call uncover cell on all cells adjacent to the cell first uncovered (and also on the original cell but it won’t do anything) and if any of those cells have 0 mines nearby, they will call Recursive Uncover which will run uncover cell on all of the surrounding cells to that cell and so on and so forth. This means if an empty cell is uncovered, all surrounding cells will be uncovered in a ripple effect until they hit cells that contain numbers.

## Check Win Conditions

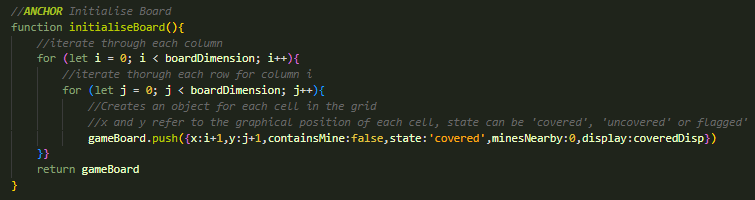
A screen shot of a computer code

Description automatically generated with low confidence

This function will be called after each turn. This function uses a for loop to iterate through each cell on the gameboard. For each cell, the program will check its state and whether it contains a mine. If the cell’s state is uncovered or if the cell contains a mine, then the loop will continue to the next iteration. If the cell does not pass the check, then the loop will be broken, and the game will continue as normal. The variable gamefinished is used to track if the game has been won. When the function is called, gamefinished is set to true, but if the loop is broken it will be set to false. If the value of gamefinsihed is true once the for loop is done, then the player has met win conditions and the game will end in a victory.

# Game Requirements Programming

## Generating the Gameboard



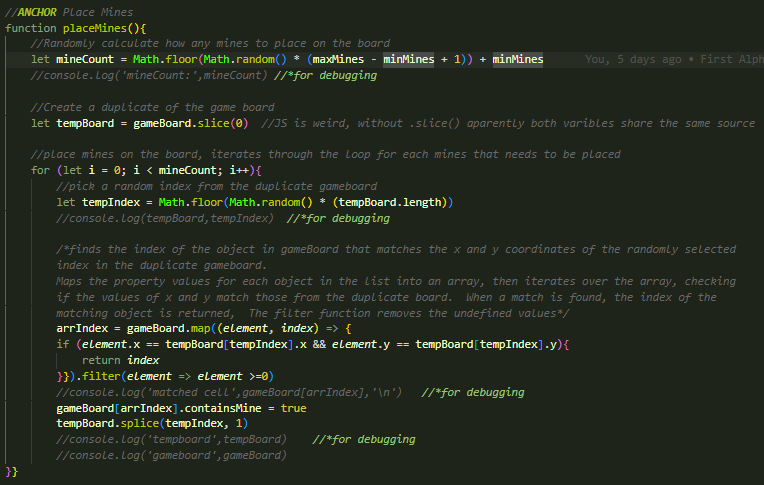
When implementing the UI display, I needed to include a character to visually represent what state each cell was in. I achieved this be adding the ‘display’ property to the cell objects. This allows me to store a character in each cell to represent different cell states which will be printed to the gameboard during rendering.

Instead of using hardcoded values for the display characters, I used variables allowing for easier customisation.

A black screen with white text

Description automatically generated with low confidence

## Placing Mines



This function did not initially receive any significant changes during implementation\*. However, testing revealed a significant problem with this function, documented in section 4, test log MSFG004. This is the updated version of the function.

This version alters the logic for selecting a location to place a mine. The original logic randomly selected a location to place a mine, then checked if the selected cell already contained a mine and placed one if not. If a selected cell already contained a mine, the program would randomly select a different cell to try to place the mine in. The problem was, the program could select cells with mines multiple times in a row, before successfully placing a mine. While this is not an issue on a small gameboard, on large gameboards with high mine counts this process could take an indefinite amount of time as the chances of selecting an empty cell got smaller and smaller each time a mine was placed.

I overcame this by creating a copy of the gameboard and removing cells each time they are selected. For each iteration of the for loop the program will randomly select a cell from tempBoard (the copy of gameboard) then find the cell in gameBoard that has matching coordinates and place a mine in that cell. The randomly selected cell will then be removed from tempBoard to prevent it being selected again.

\**the variables minimum and maximum mine counts where changed to global variables instead of being parsed into the function as they were used elsewhere, but no functionality was initially changed.*

## Counting Nearby Mines

A screen shot of a computer code

Description automatically generated with low confidence

This function did not require any significant changes during implementation.

## Displaying the Board

A screenshot of a computer program

Description automatically generated with medium confidence

A picture containing text, screenshot, software

Description automatically generated

This was a new function created during function implementation. This function generates a text-based render of the gameboard and displays it in the console. I chose to use special ascii characters to create a visual representation of the board as I found other options such as console.table to be unsatisfying. Instead of using hardcoded console.log statements to create the board, I used for loops to allow the board to be variable in size. These for loops use string concatenation to append modular board segments together to create a variable sized board. Template literal syntax is used to insert the display character for each cell into the game board.

Here are some sample outputs.

A screenshot of a game

Description automatically generated with medium confidence A screenshot of a game

Description automatically generated

During testing I discovered that for board sizes over two digits extra logic is required to decease the indent between labels. To fix this I split each for loop in two. The fist for loop will end once i exceeds 10. The second for loop will begin only if the board size is greater than 10. The second for loop uses a smaller indent value to compensate for the labels reaching two digits.

A picture containing text, screenshot, software, multimedia software

Description automatically generated

## Check For Mine

A screenshot of a computer program

Description automatically generated with medium confidence

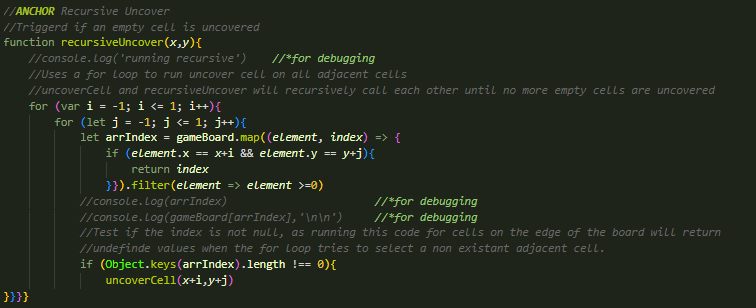
For this function I added logic to prevent the player from loosing on the first round. When a mine is uncovered, the function uses an if statement to check if the global variable gameState is equal to start. If so, instead of ending the game, the program removes the mine from the cell and calls the place mine function, setting the number of mines to be paced to 1. The uncover cell function is then called again on the cell now that the mine has been moved. The function has also been updated to make use of the new display property of the gameboard.

## Uncovering a Cell



This function has been updated to use the new display property added to the gameboard. When a cell is uncovered the function will set the display to the number of mines nearby, or to the empty cell display character if there are 0 mines nearby.

## Recursive Uncover

This function was not changed during implementation.

## Toggle Flag

A picture containing text, screenshot, software, font

Description automatically generated

This function has been updated to use the new display property added to the gameboard, no other functionality has changed.

## Checking Win Conditions

A picture containing text, screenshot, software

Description automatically generated

This function has been modified slightly to set the gameState global variable to ‘win’ and update the display if the player has met the win conditions.

## Game Settings

 A screenshot of a computer program

Description automatically generated with medium confidence A screenshot of a computer program

Description automatically generated with low confidence

This is a new function added during implementation. It alows the user to change various game settings such as board size and mine count using a commandline interface. As this is not a requirement of the program I will not go into detail on functionality.

## Begin Game

A screen shot of a computer program

Description automatically generated with medium confidence

This is a new function added during implementation. This function allows the player to open the settings menu by typing ‘settings’ or ‘s’ or start the game by pressing enter. This function uses the synchronous readline module to retrieve user input and store it as a variable. There is also unused code to open a help menu. This functionality has not been implemented so has been commented out for this release. The function uses inline functions to get user input. This way, if a user enters an invalid input, the inline function can be recursively called to get a correct input.

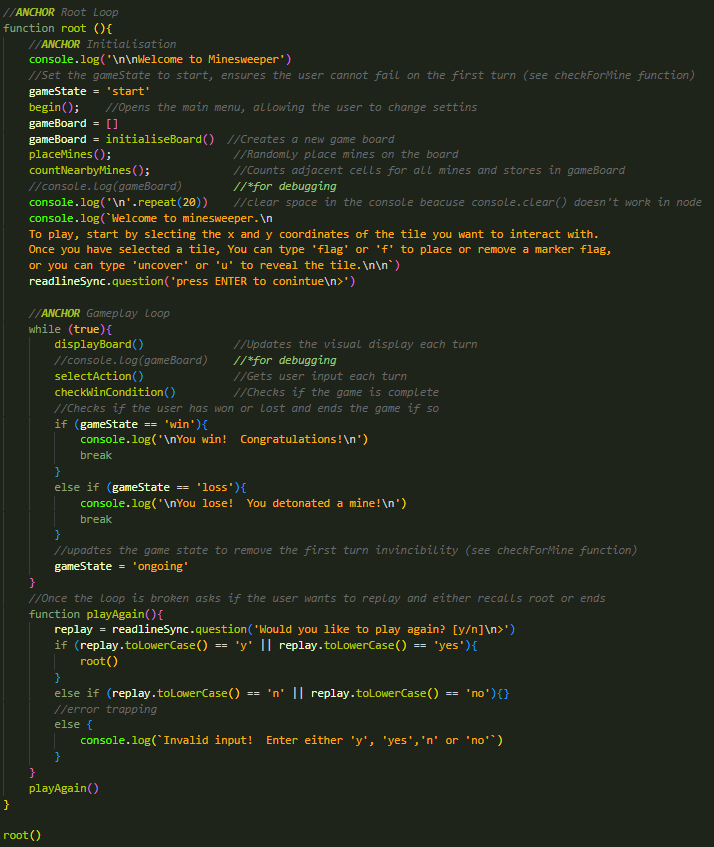
## Select Action

A screen shot of a computer program

Description automatically generated with medium confidence

This is a new function created during implementation. This function is called each turn and allows the user to interact with the game. The function uses the synchronous readline module to retrieve user input and store it as a variable. This function also utilises inline functions to allow the user to renter an invalid input. This function uses three inline functions. getX and getY retrieve the coordinate pair or the cell the user wants to interact with, getAction retrieves the action the user wants to perform on the selected cell. Valid actions are uncovering a tile or toggling a flag. If the user enters an incorrect input an error message will inform them of how they should format their input.

## Root



This is the main body of the program that is called on launch. It sequentially calls various functions to build the gameboard, run the game and then either restart or end the program depending on user input.

# Game Program Testing and Debugging

## Testing the pre-game menu

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | MSFG001 | **Test Case Description** | | Testing the functionality of the pre-game menu | | | | | | |
| **Created By** | | Caleb Eason | **Reviewed By** | | Ali Kahwaji | | **Version** | 1.1 | | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **QA Tester’s Log** | |  |  |  |  |  |  | |  |  |  | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **Tester's Name** | | Caleb Eason | **Date Tested** | | 21/05/2023 | | **Test Case (Pass/Fail/Not Executed)** | Pass | | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | | |
| 1 | readline-sync module is installed | | |  | 1 | N/A | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **Test Scenario** |  | | |  |  |  |  | |  |  |  | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | | **Pass / Fail / Not executed / Suspended** | | |
|  |
| 1 | Start program | | The program should display welcoming text and input prompt | |  | | | | Pass | | |  |
| 2 | Unexpected input ‘qwerty’ is entered | | The program should display an error message and provide the input prompt again | |  | | | | Pass | | |  |
| 3 | The following inputs are entered from the start menu:   * ‘settings’ * ‘s’ * ‘seTTingS | | The program should open the settings menu when either ‘s’ or ‘settings’ is entered. Regardless of capitalisation. | | The program successfully opened the settings menu each time.    Note due to time constraints I will not be documenting testing for the settings menu as it is not part of the program requirements. I have fully tested this menu though and it functions as intended. | | | | Pass | | |  |
| 4 | (From the start menu)  ENTER is pressed | | The program should exit the begin() function and display a short instructional text with an input prompt | |  | | | | Pass | | |  |

## Testing the initialisation

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | MSFG002 | **Test Case Description** | | Testing the initialising of the game | | | | | | |
| **Created By** | | Caleb Eason | **Reviewed By** | | Ali Kahwaji | | **Version** | 1.1 | | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **QA Tester’s Log** | |  |  |  |  |  |  | |  |  |  | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **Tester's Name** | | Caleb Eason | **Date Tested** | | 21/05/2023 | | **Test Case (Pass/Fail/Not Executed)** |  | | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | | |
| 1 | readline-sync module is installed | | |  | 1 | Default board size: 3 | | | | | |
| 2 |  | | |  | 2 | Default min mine count: 2 | | | | | |
| 3 |  | | |  | 3 | Default max mine count: 5 | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **Test Scenario** |  | | |  |  |  |  | |  |  |  | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | | **Pass / Fail / Not executed / Suspended** | | |
|  |
| 1 | Generating the board | | With a board size of 3, a gameboard with 9 cells should be produced. | | The program successfully produced the gameboard array. | | | | pass | | |  |
| 2 | Mine placement | | The program should place between 2 and 5 mines on the board. | | The program placed 3 mines in cells, [2,1], [2,2] and [3,2] | | | | pass | | |  |
| 3 | Mine Counting | | Cells should have respective nearby mine counts of 2,2,1,3,3,2,3,3,2 | | The cells had the respective nearby mine counts of 2,2,1,3,3,2,3,3,2 | | | | pass | | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | MSFG003 | **Test Case Description** | | Testing the initialising of the game | | | | | | |
| **Created By** | | Caleb Eason | **Reviewed By** | | Ali Kahwaji | | **Version** | 1.1 | | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **QA Tester’s Log** | |  |  |  |  |  |  | |  |  |  | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **Tester's Name** | | Caleb Eason | **Date Tested** | | 21/05/2023 | | **Test Case (Pass/Fail/Not Executed)** | Pass | | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | | |
| 1 | readline-sync module is installed | | |  | 1 | Default board size: 5 | | | | | |
| 2 |  | | |  | 2 | Default min mine count: 10 | | | | | |
| 3 |  | | |  | 3 | Default max mine count: 15 | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **Test Scenario** |  | | |  |  |  |  | |  |  |  | | |
|  |  |  |  |  |  |  |  | |  |  |  | | |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | | **Pass / Fail / Not executed / Suspended** | | |
|  |
| 1 | Generating the board | | With a board size of 3, a gameboard with 25 cells should be produced. | | The program successfully produced the gameboard array. | | | | pass | | |  |
| 2 | Mine placement | | The program should place between 10 and 15 mines on the board. | | The program placed 3 mines in cells, [1,2], [1,3], [1,4], [2,2], [3,4], [3,5],[4,2], [4,3], [5,2], and [5,3]  Note: the program took a while to generate the board. | | | | pass | | |  |
| 3 | Mine Counting | | Cells should have respective nearby mine counts of 2,3,4,2,1,2,3,5,4,3,2,3,4,  3,2,2,4,5,4,2,2,4,4,2,0 | | The cells had the respective nearby mine counts of 2,3,4,2,1,2,3,5,4,3,2,3,4,3,2,2,4,5,4,2,2,4,4,2,0 | | | | pass | | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | MSFG004 | **Test Case Description** | | Testing the initialising of the game | | | | | | |
| **Created By** | | Caleb Eason | **Reviewed By** | | Ali Kahwaji | | **Version** | 1.1 | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **QA Tester’s Log** | |  |  |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Tester's Name** | | Caleb Eason | **Date Tested** | | 21/05/2023 | | **Test Case (Pass/Fail/Not Executed)** |  | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | | |
| 1 | readline-sync module is installed | | |  | 1 | Default board size: 20 | | | | | |
| 2 |  | | |  | 2 | Default min mine count: 100 | | | | | |
| 3 |  | | |  | 3 | Default max mine count: 50 | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Test Scenario** |  | | |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | | **Pass / Fail / Not executed / Suspended** | | |
|
| 1 | Generating the board | | With a board size of 20, a gameboard with 400 cells should be produced. | | The program successfully produced the gameboard array. | | | | pass | | |
| 2 | Mine placement | | The program should place between 50 and 100 mines on the board. | | The program froze. | | | | Suspended.  See Section 3.2 | | |
| 3 | Mine Counting | | Cells should have appropriate mine counts | |  | | | | Not executed | | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | MSFG005 | **Test Case Description** | | Testing the initialising of the game after place mines rework | | | | | | |
| **Created By** | | Caleb Eason | **Reviewed By** | | Ali Kahwaji | | **Version** | 1.1 | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **QA Tester’s Log** | | Redesigned place mine function to me more efficient |  |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Tester's Name** | | Caleb Eason | **Date Tested** | | 21/05/2023 | | **Test Case (Pass/Fail/Not Executed)** | Pass | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | | |
| 1 | readline-sync module is installed | | |  | 1 | Default board size: 20 | | | | | |
| 2 |  | | |  | 2 | Default min mine count: 100 | | | | | |
| 3 |  | | |  | 3 | Default max mine count: 50 | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Test Scenario** |  | | |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | | **Pass / Fail / Not executed / Suspended** | | |
|
| 1 | Generating the board | | With a board size of 20, a gameboard with 400 cells should be produced. | | The program successfully produced the gameboard array. | | | | pass | | |
| 2 | Mine placement | | The program should place between 50 and 100 mines on the board. | | The successfully placed 63 mines on the board with no wait time. | | | | Pass | | |
| 3 | Mine Counting | | Cells should have appropriate mine counts | | Mine counts where accurate | | | | Pass | | |

## Testing game loop

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | MSFG006 | **Test Case Description** | | Testing the main gameplay | | | | | | |
| **Created By** | | Caleb Eason | **Reviewed By** | | Ali Kahwaji | | **Version** | 1.1 | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **QA Tester’s Log** | |  |  |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Tester's Name** | | Caleb Eason | **Date Tested** | | 21/05/2023 | | **Test Case (Pass/Fail/Not Executed)** | Pass | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | | |
| 1 | readline-sync module is installed | | |  | 1 | Default board size: 3 | | | | | |
| 2 |  | | |  | 2 | Default min mine count: 2 | | | | | |
| 3 |  | | |  | 3 | Default max mine count: 5 | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Test Scenario** |  | | |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | | **Pass / Fail / Not executed / Suspended** | | |
|
| 1 | Displaying the board | | The program should print a 3 by 3 board of covered tiles to the console. | |  | | | | pass | | |
| 2 | Entering unexpected values for x coordinate:   * f * Two * -3 | | The program should reject the input and ask again | |  | | | | Pass | | |
| 3 | Expected input, 3, is entered for x coordinate: | | The program should accept the input and ask for a y coordinate | |  | | | | Pass | | |
| 4 | Entering unexpected values for y coordinate:   * f * Two * -3 | | The program should reject the input and ask again | |  | | | | Pass | | |
| 5 | Expected input, 2, is entered for x coordinate: | | The program should accept the input and ask for an action | |  | | | | Pass | | |
| 6 | Unexpected inputs are entered for select an action:   * Flarg * 4 * (Null) | | The program should reject the input and ask again, displaying a list if valid inputs. | |  | | | | Pass | | |
| 7 | Expected inputs for placing a flag are entered into select an action:   * f * flag * F * flAg | | The program should place a flag on cell 3,2 | |  | | | | Pass | | |
| 8 | After selecting cell 1,1 the player enteres expected inputs for uncovering a cell:   * u * uncover * unCoVER | | The program should reveal cell 1,1 | |  | | | | Pass | | |
| 9 | Out of range value of 1,5 is entered with a valid input | | The program should reject the input and ask again | | The program behaved as expected, however the error message was pushed off the screen.  To prevent this I implemented a quick fix of asking the player to press enter to continue when the error is displayed. | | | | Fail | | |
| 10 | Out of range value of 1,5 is entered with a valid input (after quick fix) | | The program should display an error message then wait for input before displaying the board again. | |  | | | | Pass | | |
| 11 | Uncovering cell 3,2 | | Nothing should happen as cell 3,2 is flagged | |  | | | | Pass | | |
| 12 | Placing a flag on cell 3,2 | | As there is already a flag here, the program should remove it | |  | | | | Pass | | |
| 13 | Uncovering a mine | | The program should reveal a mine and the game should end, asking if the user wants to play again | |  | | | | Pass | | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | MSFG007 | **Test Case Description** | | Testing the main game loop | | | | | | |
| **Created By** | | Caleb Eason | **Reviewed By** | | Ali Kahwaji | | **Version** | 1.1 | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **QA Tester’s Log** | |  |  |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Tester's Name** | | Caleb Eason | **Date Tested** | | 21/05/2023 | | **Test Case (Pass/Fail/Not Executed)** | Pass | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | | |
| 1 | readline-sync module is installed | | |  | 1 | Default board size: 5 | | | | | |
| 2 |  | | |  | 2 | Default min mine count: 3 | | | | | |
| 3 |  | | |  | 3 | Default max mine count: 5 | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Test Scenario** |  | | |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | | **Pass / Fail / Not executed / Suspended** | | |
|
| 1 | Displaying the board | | A 5 by 5 board should be displayed | |  | | | | pass | | |
| 2 | An empty cell is uncovered | | The program should recursively uncover adjacent empty cells | |  | | | | Pass | | |
| 3 | The player uncovers all cells that do not contain a mine | | The game should end in a victory, the players should be asked if they want to play again | |  | | | | Pass | | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | MSFG008 | **Test Case Description** | | Testing the main game loop | | | | | | |
| **Created By** | | Caleb Eason | **Reviewed By** | | Ali Kahwaji | | **Version** | 1.1 | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **QA Tester’s Log** | |  |  |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Tester's Name** | | Caleb Eason | **Date Tested** | | 21/05/2023 | | **Test Case (Pass/Fail/Not Executed)** | Pass | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | | |
| 1 | readline-sync module is installed | | |  | 1 | Default board size: 20 | | | | | |
| 2 |  | | |  | 2 | Default min mine count: 5 | | | | | |
| 3 |  | | |  | 3 | Default max mine count: 5 | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Test Scenario** |  | | |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | | **Pass / Fail / Not executed / Suspended** | | |
|
| 1 | Displaying the board | | A 20 by 20 board should be displayed | |  | | | | Suspended  See 3.4 | | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | MSFG009 | **Test Case Description** | | Testing the generating large boards after fix | | | | | | |
| **Created By** | | Caleb Eason | **Reviewed By** | | Ali Kahwaji | | **Version** | 1.1 | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **QA Tester’s Log** | |  |  |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Tester's Name** | | Caleb Eason | **Date Tested** | | 21/05/2023 | | **Test Case (Pass/Fail/Not Executed)** | Pass | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | | |
| 1 | readline-sync module is installed | | |  | 1 | Default board size: 20 | | | | | |
| 2 |  | | |  | 2 | Default min mine count: 5 | | | | | |
| 3 |  | | |  | 3 | Default max mine count: 5 | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Test Scenario** |  | | |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | | **Pass / Fail / Not executed / Suspended** | | |
|
| 1 | Displaying the board | | A 20 by 20 board should be displayed | |  | | | | Pass | | |
| 2 | An empty cell is uncovered | | The program should recursively uncover adjacent empty cells | | (because of the small mine count player wins in one move) | | | | Pass | | |
| 3 | The player uncovers all cells that do not contain a mine | | The game should end in a victory, the players should be asked if they want to play again | |  | | | | Pass | | |

## Testing end game

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | MSFG010 | **Test Case Description** | | Testing the replay functionality | | | | | | |
| **Created By** | | Caleb Eason | **Reviewed By** | | Ali Kahwaji | | **Version** | 1.1 | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **QA Tester’s Log** | |  |  |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Tester's Name** | | Caleb Eason | **Date Tested** | | 21/05/2023 | | **Test Case (Pass/Fail/Not Executed)** | Pass | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **S #** | **Prerequisites:** | | |  | **S #** | **Test Data** | | | | | |
| 1 | readline-sync module is installed | | |  | 1 | Default board size: 3 | | | | | |
| 2 |  | | |  | 2 | Default min mine count: 2 | | | | | |
| 3 |  | | |  | 3 | Default max mine count: 5 | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Test Scenario** |  | | |  |  |  |  | |  |  |  | |
|  |  |  |  |  |  |  |  | |  |  |  | |
| **Step #** | **Step Details** | | **Expected Results** | | **Actual Results** | | | | **Pass / Fail / Not executed / Suspended** | | |
|
| 1 | Player enters a invalid input when asked to play again | | The program should reject the input and ask again | |  | | | | Pass | | |
| 2 | The player enters an expected input to restart:   * Y * Yes * yEs | | The program should restart | | (because of the small mine count player wins in one move) | | | | Pass | | |
| 3 | The player uncovers all cells that do not contain a mine | | The game should end in a victory, the players should be asked if they want to play again | |  | | | | Pass | | |
| 4 | The player enter an expected input to end the game:   * n * no * NO | | The program should end | | Program ends | | | | Pass | | |

# References

Eason, C. (2023). SDV503 Assessment 1. Nelson Marlborough Institute of Technology.