Ontario Tech University (UOIT)
Faculty of Engineering and Applied Science (FEAS)
Department Of Electrical, Computer, And Software Engineering

Principles of Software and Requirements (SOFE 2720)

Major Project | Deliverable #3 | Document of Unit Tests

Dr. Sukhwant Kaur Sagar
TA: Md Maruf
Winter 2019

<u>Group Members</u>
Aryan Kukreja (100651838)
Sunil Tumkur (100620430)
Amin Khakpour (100669547)
Armando Cuesta Leyva (100652479)

Purpose of This Document

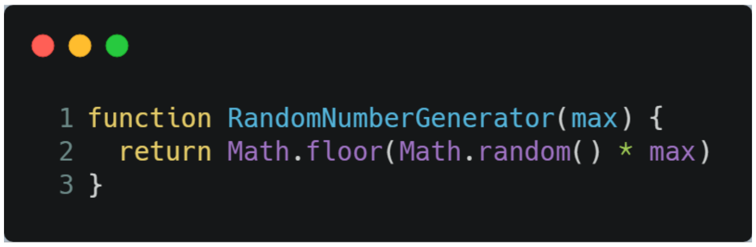
This document contains a list of unit test-cases that could not be implemented using a Javascript test library like `Mocha`. There are two reasons for this:

1. The Javascript function has a Random Number Generator
2. The Javascript Function writes to an HTML page

`Mocha` cannot test for both of these cases. Hence, as an alternative, each function will be described in detail below as a unit test:

Unit Test #1: RandomNumberGenerator(max)

The main purpose of this function was to generate a random number between 0 and a max value (inclusive). This function was implemented because random number generation was an essential component of the Sudoku code, and was used extensively in the Javascript file.

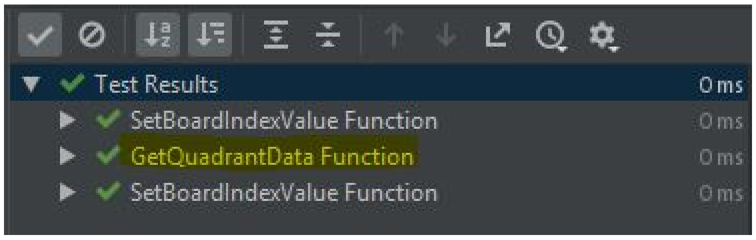


```javascript
function RandomNumberGenerator(max) {
  return Math.floor(Math.random() * max)
}
```

Figure 1: Image of RandomNumberGenerator() function

Unit Test #2: GetQuadrantData()

This function is used to get the quadrant-location related data of a particular location. It is tested using Mocha, and the test results were successful. Here is a snapshot of the output from Mocha when the unit test was run:



```
Test Results                          0 ms
  SetBoardIndexValue Function         0 ms
  GetQuadrantData Function            0 ms
  SetBoardIndexValue Function         0 ms
```

Image 1: Snapshot of Output

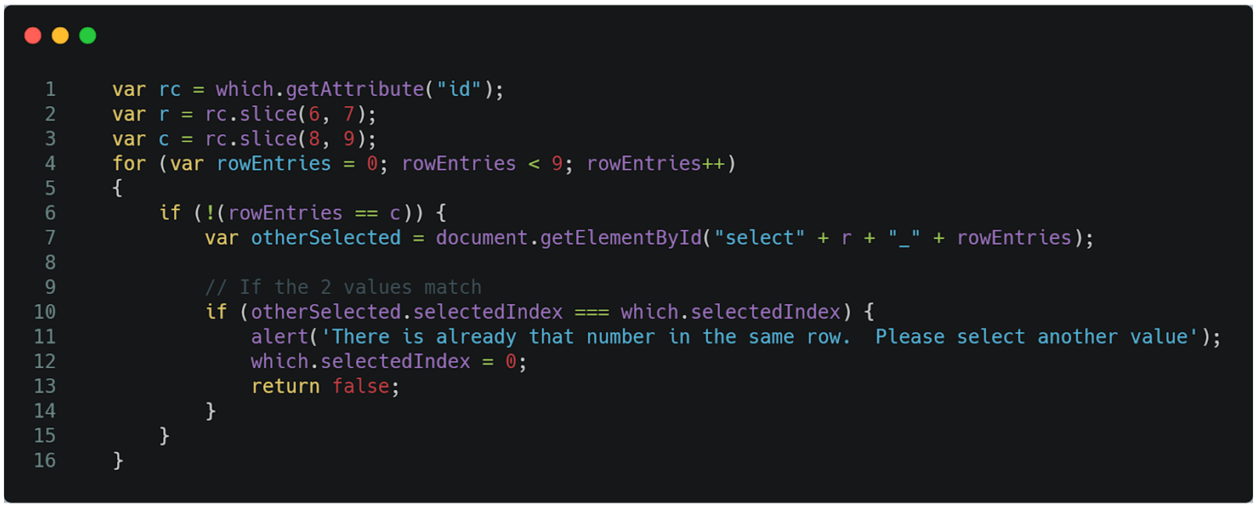
Unit Test #3: Changes()

This is the largest function in the entire Javascript file. Since it is large, it will be described in separate parts below. In general, however, it is used to detect a change in the board made by the user through an input into one of the cells. After detecting the change, it checks for 2 things:

- If the user move is invalid (the same number exists within the row, column or quadrant already).
- If the user has won the game

Code Part #1

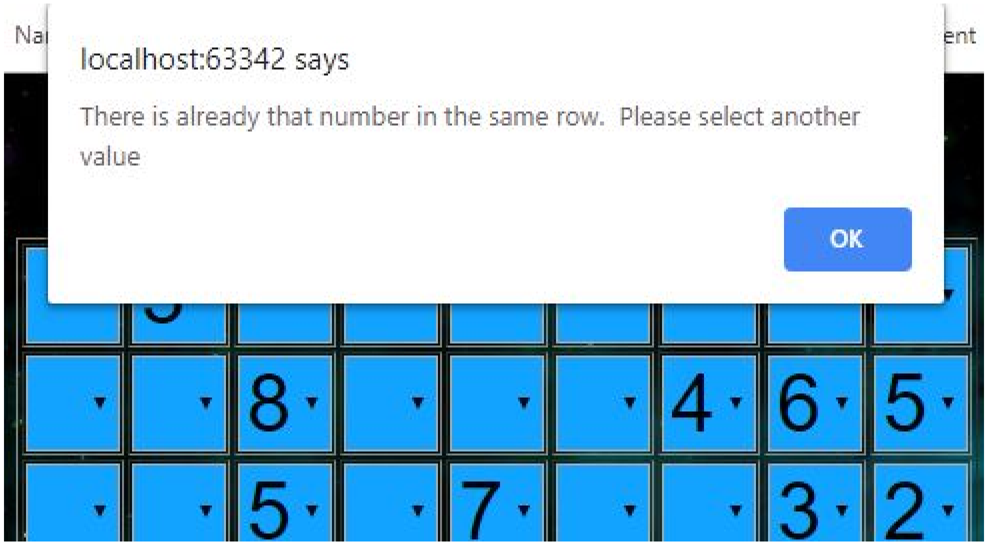
This section of the code declares three variables rc, r and c. Variable rc acquires which row and column of which attribute is selected through a particular id which is from the "select id" code block in the generateTable function. The next two variables declared r, and c. These two variables are involved with the slice method which select and return the elements 7 for variable r and element 9 for variable c. After this, it leads to the for loop which involve multiple if statements and they all coordinate with one another to check if a user enters tries to enter the same number twice within that same row. If this occurs the alert method is activated and displays an error message which will then return a boolean value of false.



```javascript
var rc = which.getAttribute("id");
var r = rc.slice(6, 7);
var c = rc.slice(8, 9);
for (var rowEntries = 0; rowEntries < 9; rowEntries++)
{
    if (!(rowEntries == c)) {
        var otherSelected = document.getElementById("select" + r + "_" + rowEntries);

        // If the 2 values match
        if (otherSelected.selectedIndex === which.selectedIndex) {
            alert('There is already that number in the same row.  Please select another value');
            which.selectedIndex = 0;
            return false;
        }
    }
}
```

Figure 2a: Part 1 of Change() Function

This is how it looks when implemented. Since it works, the unit test-case can be said to be successful:

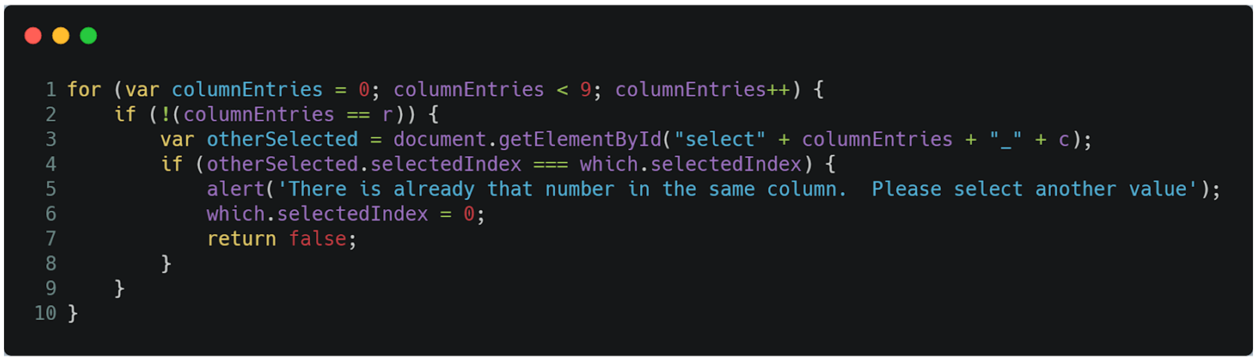

localhost:63342 says

There is already that number in the same row. Please select another value

Figure 2b: Visual Output

Code Part #2

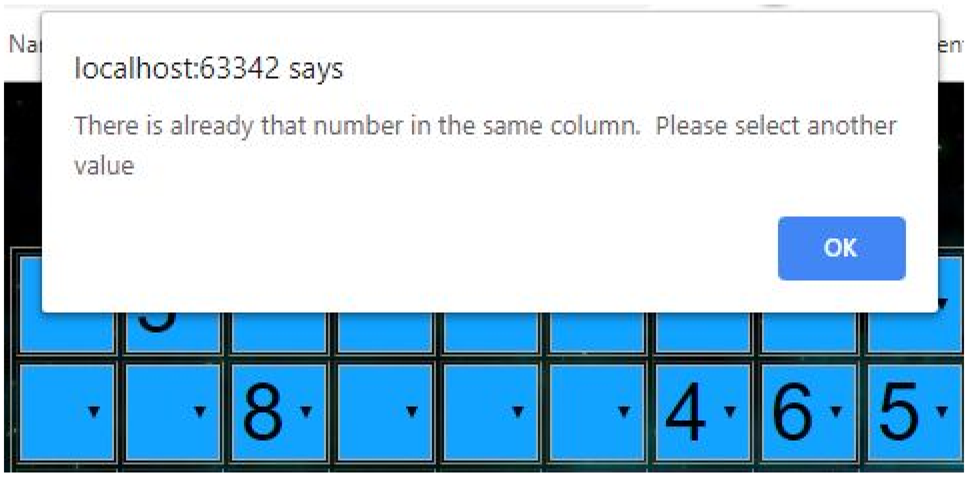
This section of the code contains one for loop and multiple if statements within it. It is a loop made to test the cells in a column being entered by the user. It’s purpose is to see if a user tries to enter the same number twice, the second if statement will then catch that error and activate the alert method where an error message will be outputted, along with returning a boolean value of false.



```javascript
for (var columnEntries = 0; columnEntries < 9; columnEntries++) {
    if (!(columnEntries == r)) {
        var otherSelected = document.getElementById("select" + columnEntries + "_" + c);
        if (otherSelected.selectedIndex === which.selectedIndex) {
            alert('There is already that number in the same column.  Please select another value');
            which.selectedIndex = 0;
            return false;
        }
    }
}
```

Figure 3: Part 2 of Change() Function

This is how it looks when implemented. Since it works, the unit test-case can be said to be successful:

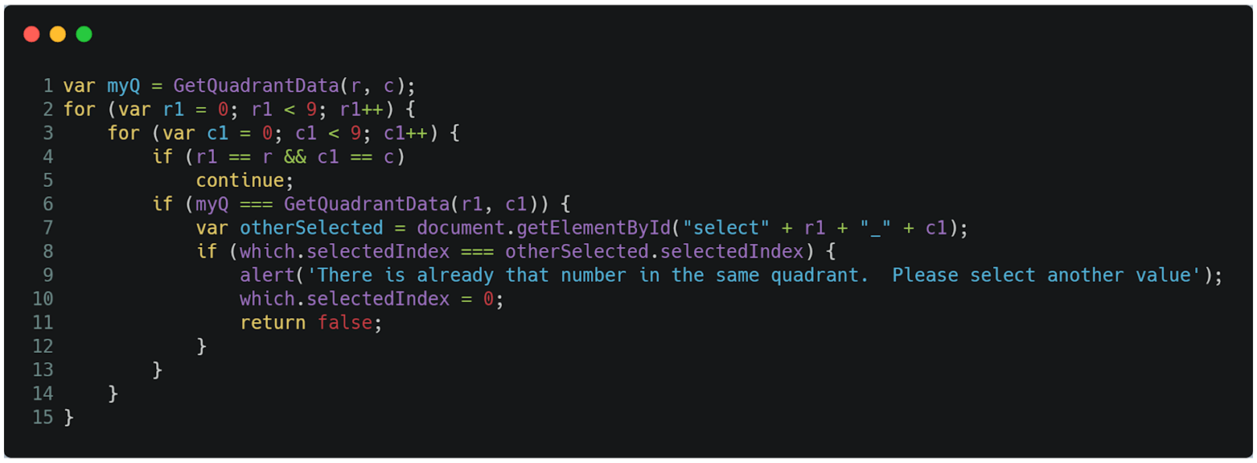

localhost:63342 says

There is already that number in the same column. Please select another value

Figure 2b: Visual Output

Code Part #3

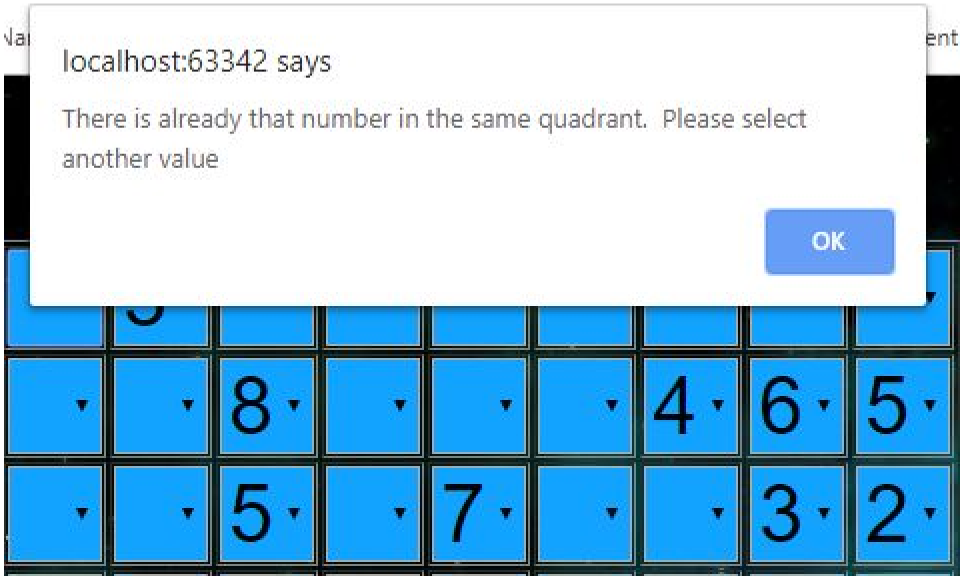
This section of the code contains the a variable called myQ which acquires the particular quadrant within a particular row and column, and also multiple for and if loops within each other. The two for loops are for both row and columns and then within the if statements, it is checked to see if the user tries to enter the same number twice within that quadrant which is a 3x3 box. If this happens, the alert method is activated and an error message is displayed along with the statement returns a boolean of false.



```
var myQ = GetQuadrantData(r, c);
for (var r1 = 0; r1 < 9; r1++) {
    for (var c1 = 0; c1 < 9; c1++) {
        if (r1 == r && c1 == c)
            continue;
        if (myQ === GetQuadrantData(r1, c1)) {
            var otherSelected = document.getElementById("select" + r1 + "_" + c1);
            if (which.selectedIndex === otherSelected.selectedIndex) {
                alert('There is already that number in the same quadrant.  Please select another value');
                which.selectedIndex = 0;
                return false;
            }
        }
    }
}
```

Figure 4: Part 3 of Change() Function

This is how it looks when implemented. Since it works, the unit test-case can be said to be successful:


localhost:63342 says

There is already that number in the same quadrant. Please select another value

Figure 2c: Visual Output

Code Part #4

This section of the code contains multiple for loops and if statements. What this section does is that it checks to see if the user fills each column and row correctly for the game. If this happens then the alert method activates and displays a “You win the game!” message. However, if this does not occur, the variable win will be set to false and we break out of the for loop until we get the correct arrangement of numbers.

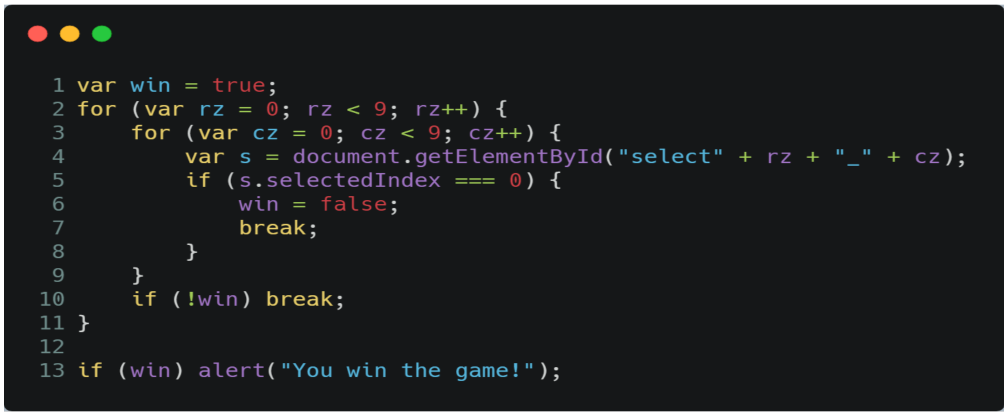

```
var win = true;
for (var rz = 0; rz < 9; rz++) {
    for (var cz = 0; cz < 9; cz++) {
        var s = document.getElementById("select" + rz + "_" + cz);
        if (s.selectedIndex === 0) {
            win = false;
            break;
        }
    }
    if (!win) break;
}

if (win) alert("You win the game!");
```

Figure 5: Part 4 of Change() Function

This is how it looks when implemented. Since it works, the unit test-case can be said to be successful:

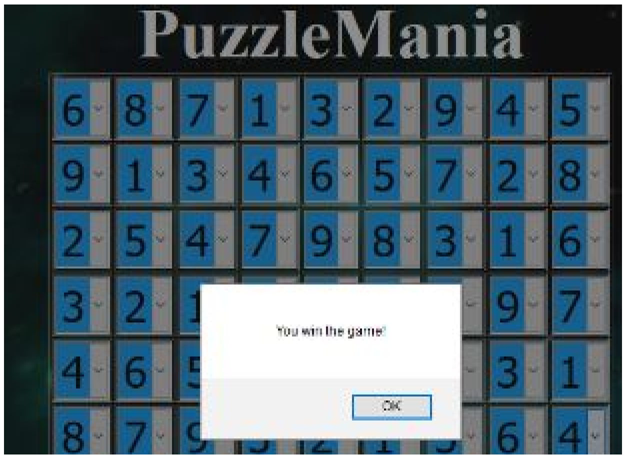
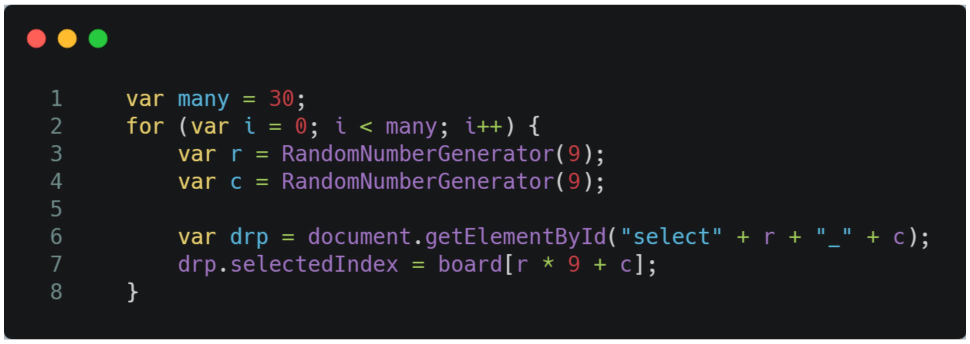


Figure 2c: Visual Output

Unit Test #4: OpenSeveral()

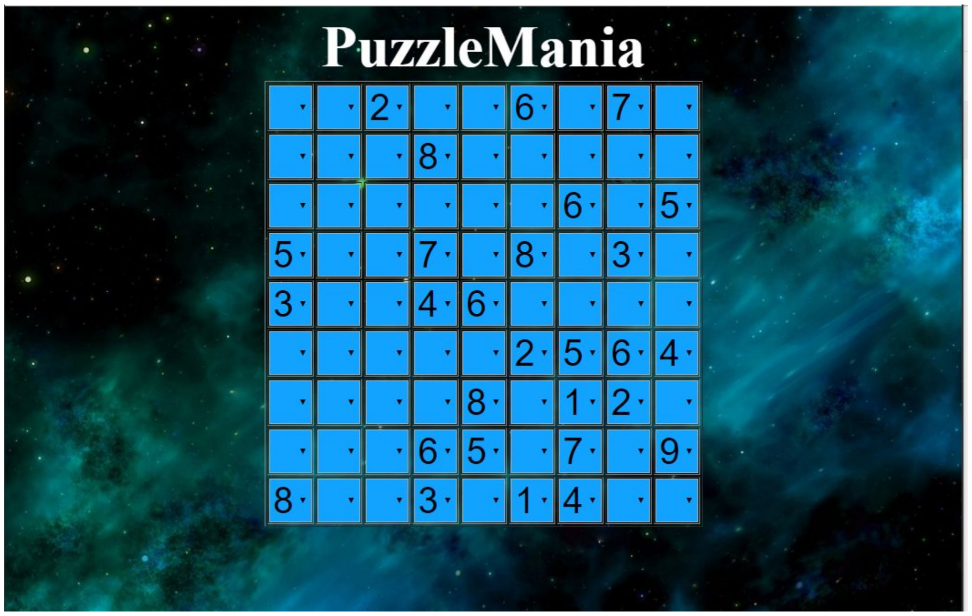
The main purpose of this function is to populate 30 cells of the Sudoku Puzzle with random values upon creation. The cells picked will be chosen by random as well. This is to give the user a set of pre-set values in the puzzle that cannot be changed.



```
var many = 30;
for (var i = 0; i < many; i++) {
    var r = RandomNumberGenerator(9);
    var c = RandomNumberGenerator(9);

    var drp = document.getElementById("select" + r + "_" + c);
    drp.selectedIndex = board[r * 9 + c];
}
```

Figure 6: OpenSeveral() Code

The following image is an output of the Sudoku Game when it is just loaded by the user. The process of filling up the initial values in the cells here are done by the OpenSeveral() function. Hence, this function works properly:



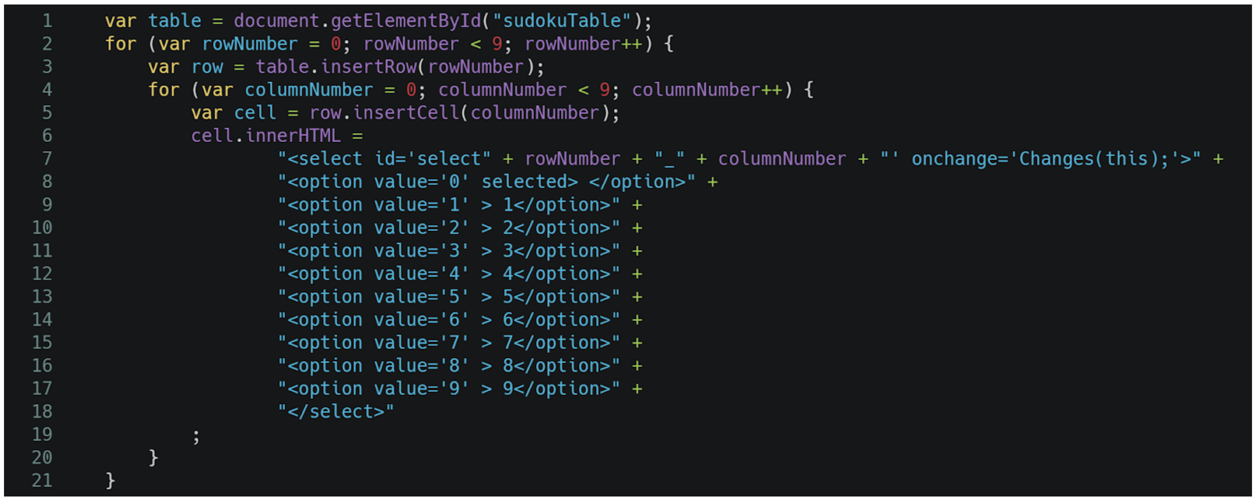
		2			6		7	
			8					
						6		5
5			7		8		3	
3			4	6				
					2	5	6	4
				8		1	2	
			6	5		7		9
8			3		1	4		

Figure 7: Output of the Game when Loaded. Notice the numbers are initially loaded; this is performed by the OpenSeveral() function

Unit Test #5: GenerateTable()

This function can be seen as the link between the HTML page that displays the board and the Javascript engine. Essentially, this function iterates across each row and column of the Sudoku Puzzle, and for each entry, it creates a cell in the table that is show the user on the Web-Page.

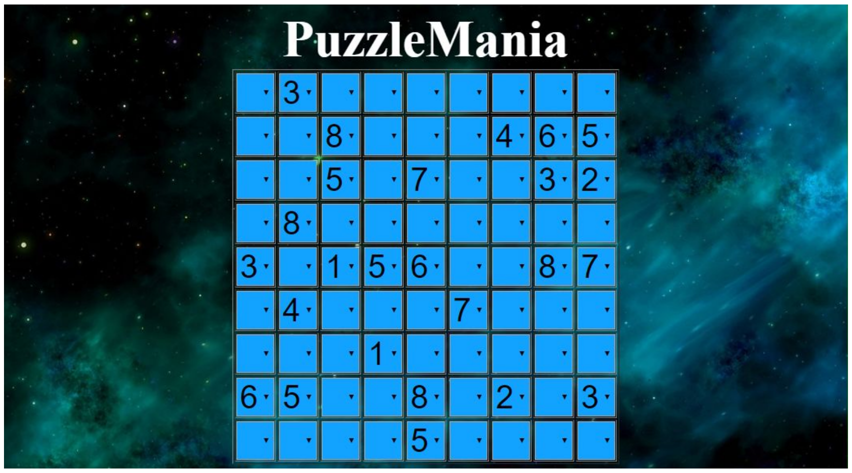
This function iterates over each column within each row. Within the for-loop for each row, it creates a new row in the table. Within the for-loop for each column, it creates a new entry and writes to it in the table.



```
var table = document.getElementById("sudokuTable");
for (var rowNumber = 0; rowNumber < 9; rowNumber++) {
    var row = table.insertRow(rowNumber);
    for (var columnNumber = 0; columnNumber < 9; columnNumber++) {
        var cell = row.insertCell(columnNumber);
        cell.innerHTML =
                "<select id='select" + rowNumber + "_" + columnNumber + "' onchange='Changes(this);'>" +
                "<option value='0' selected> </option>" +
                "<option value='1' > 1</option>" +
                "<option value='2' > 2</option>" +
                "<option value='3' > 3</option>" +
                "<option value='4' > 4</option>" +
                "<option value='5' > 5</option>" +
                "<option value='6' > 6</option>" +
                "<option value='7' > 7</option>" +
                "<option value='8' > 8</option>" +
                "<option value='9' > 9</option>" +
                "</select>"
        ;
    }
}
```

Figure 8: GenerateTable()

The unit-test can be considered successful, given that the function operates and outputs as desired.



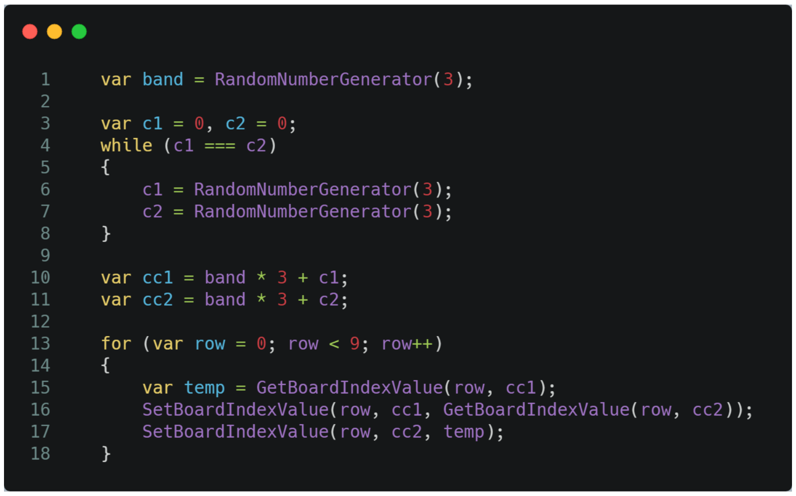
	3							
		8				4	6	5
		5		7			3	2
	8							
3		1	5	6			8	7
	4				7			
			1					
6	5			8		2		3
				5				

Figure 9: Output of Table when Written to from GenerateTable()

Unit Test #6: ColumnSwap()

This function is used to swap the 2 columns of a Sudoku Puzzle. This function is instrumental when creating a puzzle as it randomizes the puzzle completely through a series of column swaps across the entire board.

There is no specific unit test for this function, as the randomness factor makes it impossible to demonstrate through screenshots and the `Mocha` library.



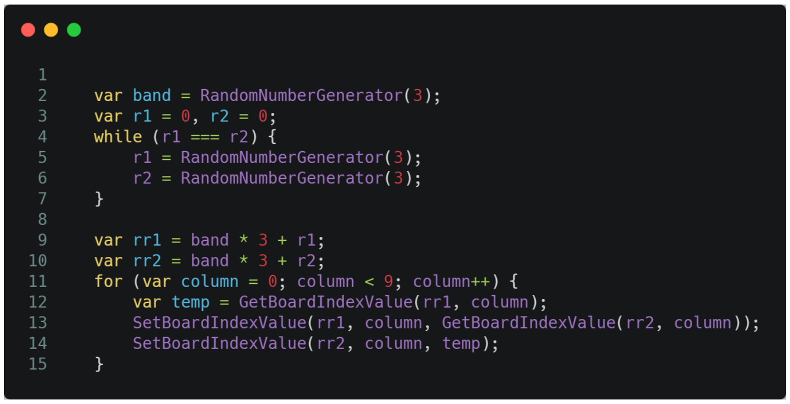
```
var band = RandomNumberGenerator(3);

var c1 = 0, c2 = 0;
while (c1 === c2)
{
    c1 = RandomNumberGenerator(3);
    c2 = RandomNumberGenerator(3);
}

var cc1 = band * 3 + c1;
var cc2 = band * 3 + c2;

for (var row = 0; row < 9; row++)
{
    var temp = GetBoardIndexValue(row, cc1);
    SetBoardIndexValue(row, cc1, GetBoardIndexValue(row, cc2));
    SetBoardIndexValue(row, cc2, temp);
}
```

Figure 10: ColumnSwap() Code

Unit Test #7: RowSwap()

This function is used to swap the 2 rows of a Sudoku Puzzle. This function is instrumental when creating a puzzle as it randomizes the puzzle completely through a series of row swaps across the entire board. *This function can be seen as a sister function of the **ColumnSwap()** function.*

There is no specific unit test for this function, as the randomness factor makes it impossible to demonstrate through screenshots and the `Mocha` library.

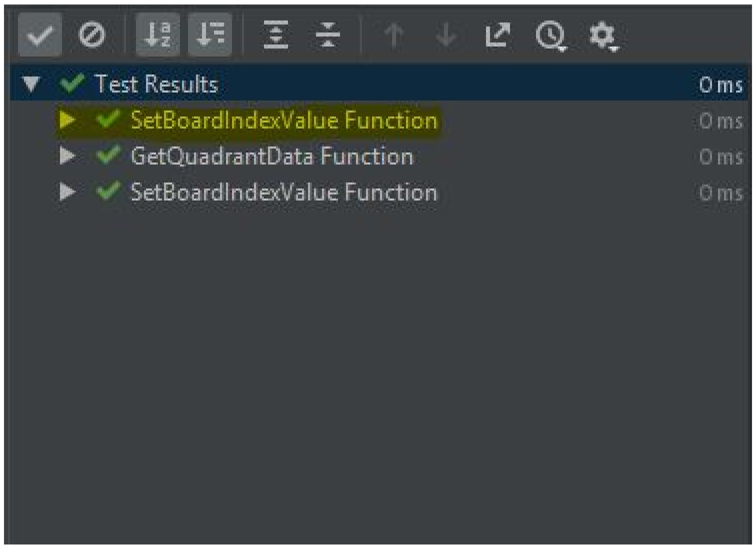


```

var band = RandomNumberGenerator(3);
var r1 = 0, r2 = 0;
while (r1 === r2) {
    r1 = RandomNumberGenerator(3);
    r2 = RandomNumberGenerator(3);
}

var rr1 = band * 3 + r1;
var rr2 = band * 3 + r2;
for (var column = 0; column < 9; column++) {
    var temp = GetBoardIndexValue(rr1, column);
    SetBoardIndexValue(rr1, column, GetBoardIndexValue(rr2, column));
    SetBoardIndexValue(rr2, column, temp);
}
```

Figure 11: RowSwap() Code

Unit Test #8: SetBoard()

This function sets a value on the board. It is tested using Mocha, and the test results were successful. Here is a snapshot of the output from Mocha when the unit test was run:

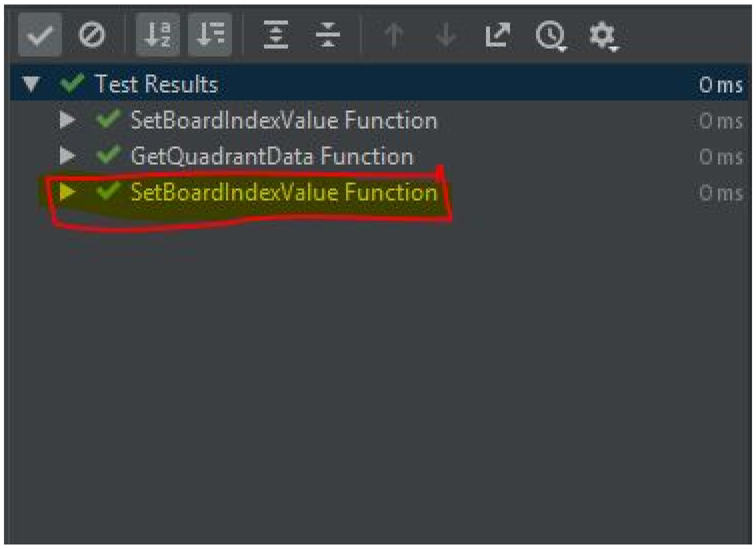


```
▼ ✓ Test Results                      0 ms
    ▶ ✓ SetBoardIndexValue Function    0 ms
    ▶ ✓ GetQuadrantData Function       0 ms
    ▶ ✓ SetBoardIndexValue Function    0 ms
```

Figure 12: Snapshot of Output

Unit Test #9: GetBoard()

This function gets a value from the board. It is tested using Mocha, and the test results were successful. Here is a snapshot of the output from Mocha when the unit test was run:

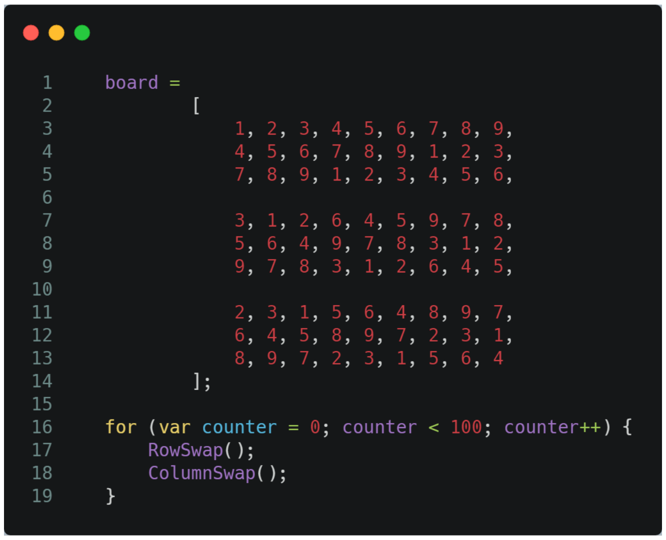


```
▼ ✓ Test Results                          0 ms
  ▶ ✓ SetBoardIndexValue Function         0 ms
  ▶ ✓ GetQuadrantData Function            0 ms
  ▶ ✓ SetBoardIndexValue Function         0 ms
```

Figure 13: Snapshot of Output

Unit Test #10: GenerateBoard()

This function generates a board with an initial set of arcane pre-set values. The randomization of these values occurs to a repeated call of RowSwap() and ColumnSwap() functions over the number of cells in the table. This function works properly, as the output comes as expected on the final web-page.



```
board =
        [
            1, 2, 3, 4, 5, 6, 7, 8, 9,
            4, 5, 6, 7, 8, 9, 1, 2, 3,
            7, 8, 9, 1, 2, 3, 4, 5, 6,

            3, 1, 2, 6, 4, 5, 9, 7, 8,
            5, 6, 4, 9, 7, 8, 3, 1, 2,
            9, 7, 8, 3, 1, 2, 6, 4, 5,

            2, 3, 1, 5, 6, 4, 8, 9, 7,
            6, 4, 5, 8, 9, 7, 2, 3, 1,
            8, 9, 7, 2, 3, 1, 5, 6, 4
        ];

for (var counter = 0; counter < 100; counter++) {
    RowSwap();
    ColumnSwap();
}
```

Figure 15: Snapshot of Code