Implementation Test

*Please refer to the unit testing introduction prior to reading the implementation test

Since Unity does not have a strong support of unit testing due to its high front and back end integration, the boundary between the unit testing and implementation testing is quite shallow.

For example, each grid is saved as gameobjects in Unity, and due to its high dependencies in a game development environment, some of the methods have to be interconnected to make sense for the other (e.g. sending rowCheck requires the highlighted square to send in the numberSelected state).

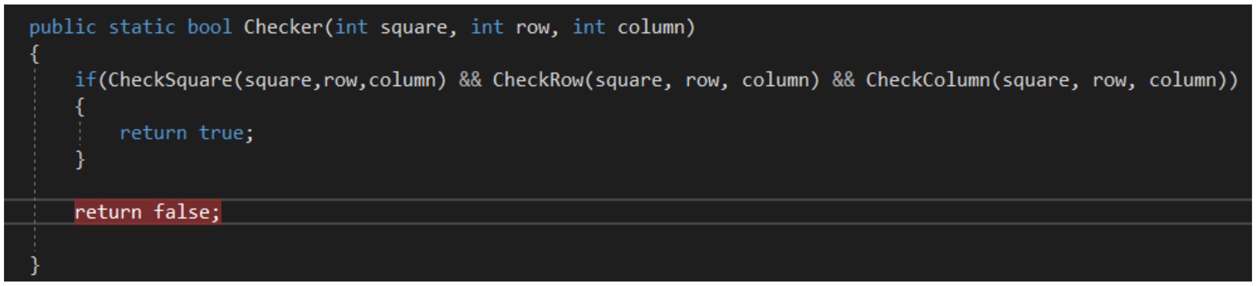
However method such as –checking row, column, sub-square duplicates can be combined in a super method, so rather than calling all individually, we call separately.

In the implementation test, we will be doing two tests

1. Checker (for checking row, column, sub-square)
2. Completed test (checking if the board is completed or not)

1. Checker (for checking row, column, sub-square)

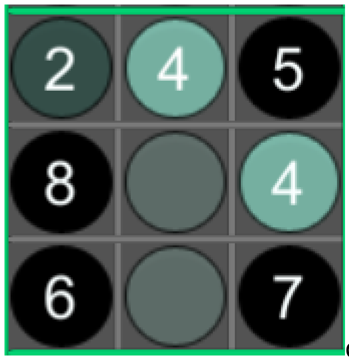
Checker method calls in check sub-square method, check row method, and check column method. If any of them returns false, the method will return false.



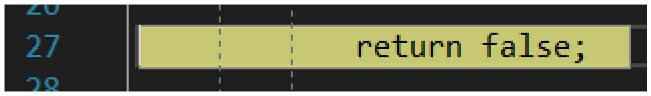
```
public static bool Checker(int square, int row, int column)
{
    if(CheckSquare(square,row,column) && CheckRow(square, row, column) && CheckColumn(square, row, column))
    {
        return true;
    }

    return false;

}
```

To test this method, we will be inserting a breaking point at the return false statement of the checker method.

a. Testing false for sub-square



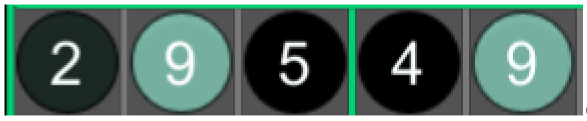
duplicated number in a sub-square



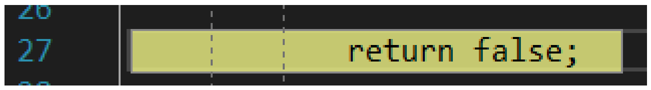
```
return false;
```

returned false

b. Testing false for row checker



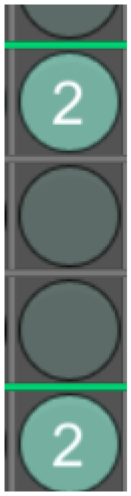
duplicated number in a row

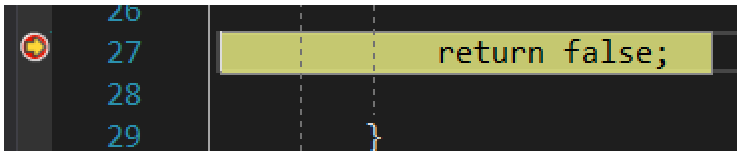


```
return false;
```

returned false

c. Testing false for column checker





```
                return false;

            }
```

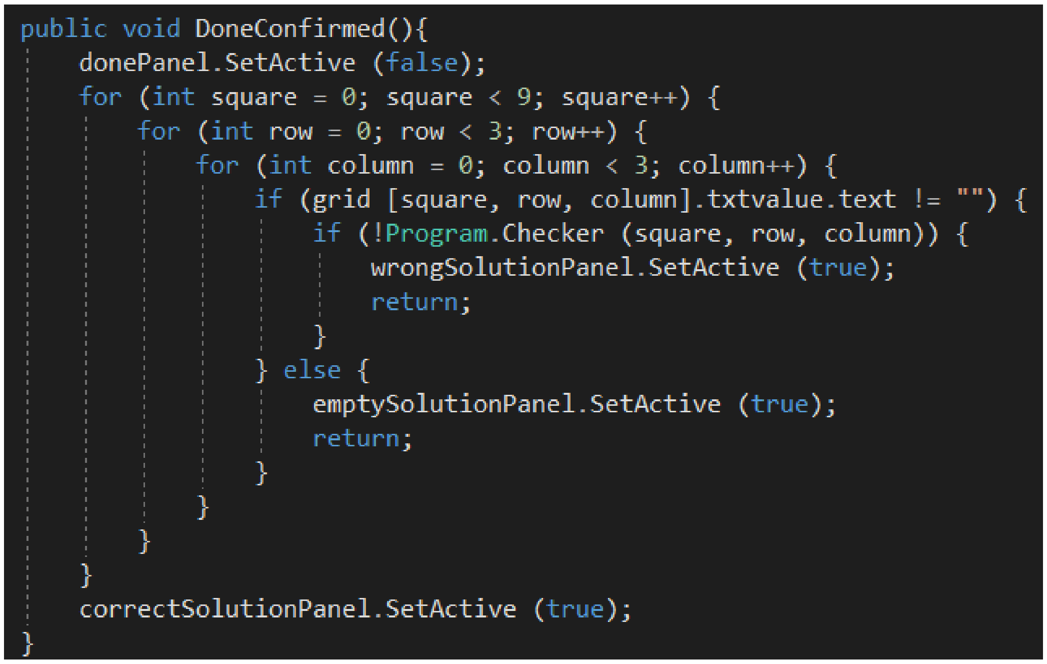
duplicated number in a column

returned false

Conclusion: the implement test passed since the checker method was successfully detect duplicated number in all 3 different scenarios

2. Completed test (checking if the board is completed or not)

In our game, completion button is available for users to complete the game. The method behind the button checks the state of the board (checks for any empty squares), and goes through the checker method once again (from step 1).

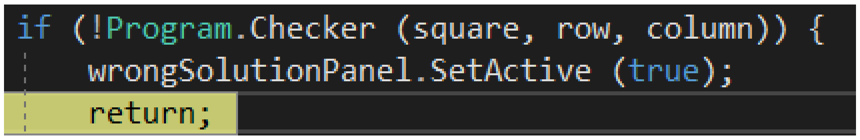


```
public void DoneConfirmed(){
    donePanel.SetActive (false);
    for (int square = 0; square < 9; square++) {
        for (int row = 0; row < 3; row++) {
            for (int column = 0; column < 3; column++) {
                if (grid [square, row, column].txtvalue.text != "") {
                    if (!Program.Checker (square, row, column)) {
                        wrongSolutionPanel.SetActive (true);
                        return;
                    }
                } else {
                    emptySolutionPanel.SetActive (true);
                    return;
                }
            }
        }
    }
    correctSolutionPanel.SetActive (true);
}
```

The three possible outcomes are possible.

a. Wrong solution
b. An empty square was spotted
c. Default(correct solution)

a. Wrong solution

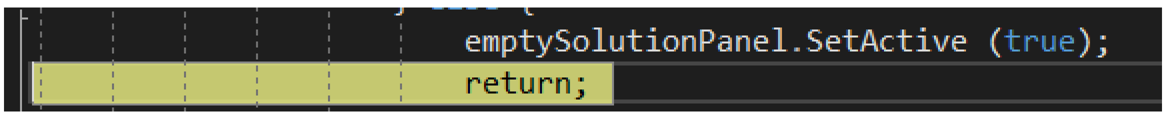


```
if (!Program.Checker (square, row, column)) {
    wrongSolutionPanel.SetActive (true);
    return;
```

When we insert the wrong solution, it must give a false from the checker method. As expected – by setting a breaking point at return statement after the if statement for wrong solution, the game paused successfully.

b. Empty square was spotted

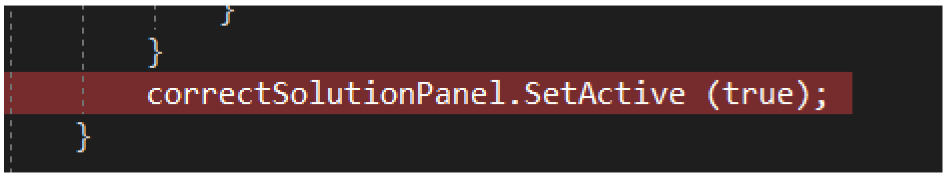
While going through a for loop, when an empty spot was detected (square did not have a value state), it must return an empty solution panel which warns a user – as expected, by setting a breaking point at return statement after the if statement for the empty solution, the game paused successfully



```
                emptySolutionPanel.SetActive (true);
                return;
```

c. Default

Since the last condition is the default, if the game passes the first two conditions, it should give a correct solution panel to the user. As expected, by setting a breaking point at return statement at the default statement, the game paused successfully



```
            }
        }
        correctSolutionPanel.SetActive (true);
    }
```

Conclusion: the implement test passed since the completion method was successfully detect duplicated number in all 3 different scenarios