

Name: Cieran Hughes

Student Number: 1800730

Assignment Title: Computing Portfolio

Module Title: Software Development

Course Title: Applied Computing

Abstract

This report will include:

- An overview of each of the exercises completed.
- The problems that were encountered and solved when completing the exercises.
- Evidence of problem solving before coding the project. This is all notes that help the planning of the exercises.
- The stages of development on the exercises.
- A testing section to show that all aspects of the exercises work as expected too.

Contents Page

1. Introduction.....	1
2. Exercise 1	1
2.1 Overview of the Exercise	1
2.2 Problems Solved and Evidence	1
2.2.1 Validation of Numbers Entered	1
2.2.2 Passing Arrays in the Functions	1
2.3 Problem Solving Before Code	2
2.4 Stages Of Development.....	2
2.5 Testing.....	3
2.5.1 Number Size Validation.....	3
2.5.2 Whole Number Validation	3
2.5.3 Player 1 Wins	4
2.5.3 Player 2 Wins	5
3. Exercise 2	6
3.1 Overview of the Section.....	6
3.2 Problem Solved and Evidence	6
3.2.1 Displaying the Grid.....	6
3.2.3 Validating the Coordinated	6
3.3 Problem Solving Before Code	7
3.4 Stages of Development	7
3.5 Testing.....	8
3.5.1 Player 1 Number Size	8
3.5.2 Player 1 Coordinates Previously Picked	9
3.5.3 Player 1 Win	9
3.5.4 Player 2 Coordinates Previously Picked.....	11
3.5.5 Player 2 Win.....	12
4. Exercise 3	14
4.1 Overview of the Exercise.....	14
4.2 Problem Solving and Evidence	14
4.2.1 Diagonal Checks to Find Winner	14
4.2.2 Blitz	14
4.3 Problem Solving Before Code.....	15

4.4 Stages of Development	16
4.4.1 Input choice	16
4.4.2 Winner Checks	16
4.5 Testing	18
4.5.1 Player 1 Wins - Horizontal.....	18
4.5.2 Player 1 Wins - Vertical	18
4.5.3 Player 1 Wins - Diagonal Left.....	19
4.5.4 Player 1 Wins - Diagonal Right	19
4.5.5 Player 2 Wins - Horizontal.....	20
4.5.6 Player 2 Wins -Vertical.....	20
4.5.7 Player 2 Wins - Diagonal Left.....	21
4.5.8 Player 2 Wins - Diagonal Right	21
4.5.9 Blitz	22
4.5.10 Validation	23
5. Appendix.....	25
5.1 Appendix - Exercise 1 Code	25
5.2 Appendix - Exercise 2 Code	31
5.3 Appendix - Exercise 3 Code	41

1. Introduction

This report will outline three exercise tasks that have all been completed. Each task will be broken down in to sections giving an overview of the exercise, problems solved and the evidence for it, proof of problem solving before code, the stages of development that the exercise when through and testing of the exercise.

2. Exercise 1

2.1 Overview of the Exercise

This exercise is a game, player 1 needed to enter 7 numbers that are between or equal to one and ten. Then player two gets to see the first number and needs to guess if the following number is higher or lower. If they are correct they go on to the next number until they get all of the differences correct, then they win. If they guess wrong player 1 wins.

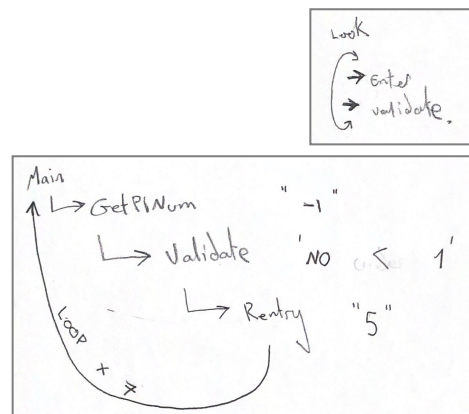
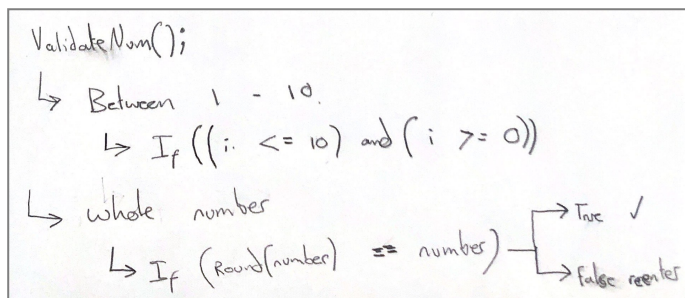
2.2 Problems Solved and Evidence

2.2.1 Validation of Numbers Entered

This exercise a problem I encountered was validating the numbers that were entered by Player 1. There are two validations that needed to be done, one of these were making sure the number entered was between or equal to one and ten, then the second validation was that they needed to be integers. This was a problem as all the validation was in a function but needed to be called for each number entered. Then if the number didn't pass the validation it needed to be reentered with the same validation to check it.

Validating the size was easy as it was only two if statements to check if the number was between the values, but the second part of the validation was slightly harder. To get around this there is a round function that can be called and the compiler will round the double to an integer and not truncate the number. To make this worked there needed to be an added header file called 'cmath' that included the function.

Problem solving before coding for this problem:



2.2.2 Passing Arrays in the Functions

The hardest problem that I encountered in this exercise was passing arrays in to functions, as its not like a variable and you cant pass it in and still have the original value still in the previous function. The array was being passed from the Main to the HighOrLower function. I researched the problem and found the correct syntax for passing the array through functions by the HigherOrLower function looking at the place in memory for the array and getting the stored values from there.

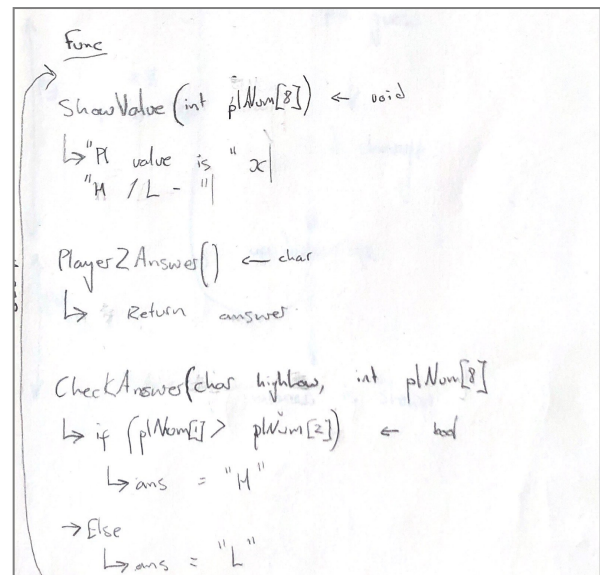
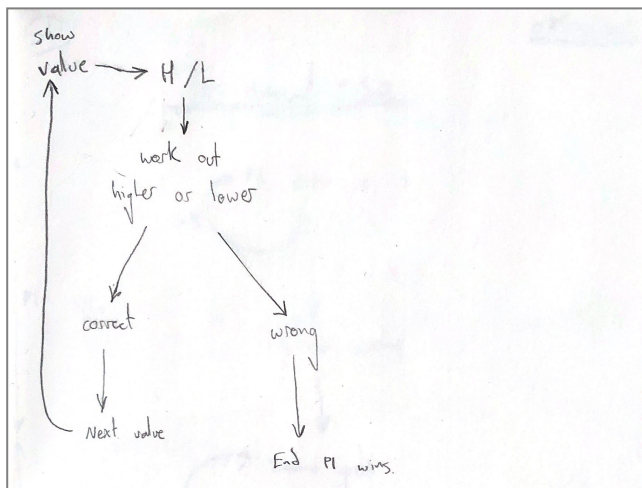
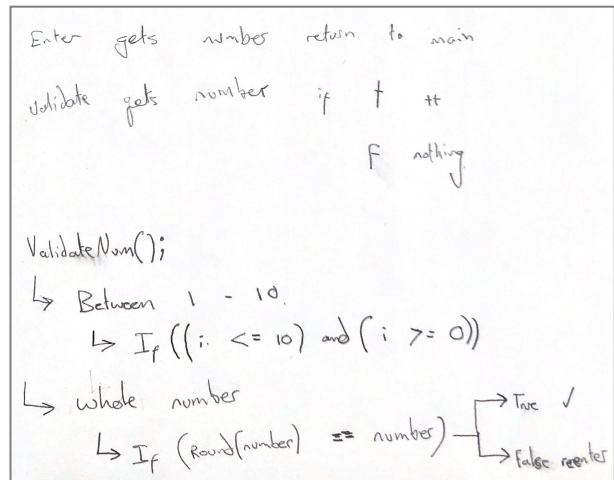
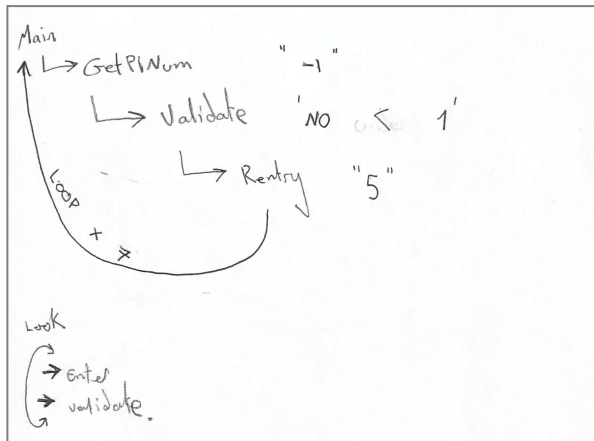
Within this problem I found that I wanted to manipulate the data in the second function, so to do this I put the array in a loop and duplicated the numbers to a local array to the second function as show to the right.

```
for(counter = 1; counter < 8; counter++)
{
    p1IntNum[counter] = p1Num[counter];
}
```

By doing this I ran in to another problem. This is that the array that was passed in has a data type of a double, and the array that was newly created needed to be an integer. But as earlier on in the code the numbers validated to make sure they are whole but still stored in an array that is a double, the loop automatically truncates the values.

2.3 Problem Solving Before Code

Below is all the screenshots of the planning that was done before coding this exercise:



2.4 Stages Of Development

When starting this game, I firstly made sure player 1 could enter the first 7 numbers. This meant making a loop to run a function 7 times that would make player 1 enter the number. After this the next stage was clearing the screen, this was another function that was added that printed 1000 new lines.

The next section to complete was making the first number that player 1 entered to show to player 2. To do this I added another function that shows the number that player 1 entered, the array that collected player 1's numbers needed to be passed in to the function. Then player 2 had to enter their guess. This was put in the loop with the function so it would keep showing and asking player 2 for their guess. This loop just kept running as there was no code to find the winner and stop the loop yet.

The next section needed to work out if player 2 was correct and end the loop if they were wrong. To do this I made a function that returned a boolean value depending on if they are correct, then the returned value can be used to see if the loop needs to stop because they were wrong or go on to the next number. Then finally the validation was added. This meant adding a function called 'ValidatonOfNumbers' in the beginning loop when user 1 enters their numbers.

2.5 Testing

2.5.1 Number Size Validation

Aim of the Test

The aim of this test is to show that the user cannot enter a number that is bigger than ten. If they do they will need to reenter the number.

Inputs

Player 1 - Numbers										Player 2 - H / L					
1	2	3	4	4	5	5	6	7		1	2	3	4	5	6
8	9	10	11	4	12	5	7	4		H	H	L	H	H	L

Expected Output

When the user entered the number that are bigger than ten they should be asked to reenter the number. This example it should happen on entry four and five

Actual Output

As expected.

```
Player one please enter 7 values:
enter number : 8
enter number : 9
enter number : 10
enter number : 11

Please enter a number that is between 1 and 10 and a whole number
enter number : 4
enter number : 12

Please enter a number that is between 1 and 10 and a whole number
enter number : 5
enter number : 7
enter number : 4
```

2.5.2 Whole Number Validation

Aim of the Test

The aim of this test is to see if the user can enter a number that isn't a whole number.

Inputs

Player 1 - Numbers										Player 2 - H / L					
1	2	3	4	4	5	5	6	6	7	1	2	3	4	5	6
1	4	3	5.5	4	10.5	8	0.1	2	4	H	L	H	H	L	H

Expected Output

On entry four, five and six player 1 will get an error message and need to reenter the number that they put in, as these numbers are all not whole.

Actual Output

As expected.

```

Player one please enter 7 values:
enter number : 1
enter number : 4
enter number : 3
enter number : 5.5

Please enter a number that is between 1 and 10 and a whole number
enter number : 4
enter number : 10.5

Please enter a number that is between 1 and 10 and a whole number
enter number : 8
enter number : 0.1

Please enter a number that is between 1 and 10 and a whole number
enter number : 2
enter number : 4

```

2.5.3 Player 1 Wins

Aim of the Test

The aim of this test is to see if player 1 can win.

Inputs

Player 1 - Numbers							Player 2 - H / L					
1	2	3	4	5	6	7	1	2	3	4	5	6
1	2	3	4	5	6	7	H	H	H	L	H	H

Expected Output

The game should not show any error messages and proceed until player 2 enters their fourth go as this go is wrong. Once this has happened the game should end showing that player 1 is the winner.

Actual Output

As expected.

```

Player one please enter 7 values:
enter number : 1
enter number : 2
enter number : 3
enter number : 4
enter number : 5
enter number : 6
enter number : 7

```

```

Player 1s value was 1
Higher (h) or Lower (l) - h
Correct
Player 1s value was 2
Higher (h) or Lower (l) - h
Correct
Player 1s value was 3
Higher (h) or Lower (l) - h
Correct
Player 1s value was 4
Higher (h) or Lower (l) - l
Incorrect
Player 1 wins.

```

2.5.3 Player 2 Wins

Aim of the Test

The aim of this test is to see that player 2 can win the game.

Inputs

Player 1 - Numbers							Player 2 - H / L					
1	2	3	4	5	6	7	1	2	3	4	5	6
7	6	5	4	3	2	1	L	L	L	L	L	L

Expected Output

The expected output of this test is the game should run all the way through. Get to the sixth entry for player 2 and recognise that they win the game and show a message to confirm this.

Actual Output

As expected.

```
Player one please enter 7 values:
enter number : 7
enter number : 6
enter number : 5
enter number : 4
enter number : 3
enter number : 2
enter number : 1
```

```
Player 1s value was 7
Higher (h) or Lower (l) - l
Correct
Player 1s value was 6
Higher (h) or Lower (l) - l
Correct
Player 1s value was 5
Higher (h) or Lower (l) - l
Correct
Player 1s value was 4
Higher (h) or Lower (l) - l
Correct
Player 1s value was 3
Higher (h) or Lower (l) - l
Correct
Player 1s value was 2
Higher (h) or Lower (l) - l
Correct
Player 2 wins.
```

3. Exercise 2

3.1 Overview of the Section

This exercise is a game, player 1 enters the coordinates of a three treasures in an eight by eight grid. Then player 2 enters coordinates to find these treasures, they start with three lives and every missed guess player 2 loses a life. The game will end if player 2 runs out of lives, meaning player 1 wins, or guesses all three of player 1's treasures, meaning player 2 wins.

3.2 Problem Solved and Evidence

3.2.1 Displaying the Grid

A problem that I encountered when doing this exercise was displaying the grid after player 2 enters their guess. Each time they guessed, the grid wouldn't show previous guesses, just shown the recent guess. This is because in the loop the grid would be initialised and set back to blank. To get around this I needed to stop the loop from running the code that sets the grid to blank. I added an extra function to show the blank grid that would only run once, then after each guess a different grid would be shown with all the guesses shown. Below is a screenshot of the 'BlankGrid' function that is shown at the beginning of the game.

```
void BlankGrid()
{
    string coor[10][10];
    int counter1;
    int counter2;

    for(counter1 = 1; counter1 < 10; counter1++)
    {
        for(counter2 = 1; counter2 < 10; counter2++)
        {
            coor[counter1][counter2] = "-";
        }
    }

    cout << "0" << " " << "1" << " " << "2" << " " << "3" << " " << "4" << " " << "5" << " " << "6" << " " << "7" << " " << "8" << endl;
    cout << "1" << " " << coor[1][1] << " " << coor[2][1] << " " << coor[3][1] << " " << coor[4][1] << " " << coor[5][1] << " " << coor[6][1] << " " << coor[7][1] << " " << coor[8][1] << endl;
    cout << "2" << " " << coor[1][2] << " " << coor[2][2] << " " << coor[3][2] << " " << coor[4][2] << " " << coor[5][2] << " " << coor[6][2] << " " << coor[7][2] << " " << coor[8][2] << endl;
    cout << "3" << " " << coor[1][3] << " " << coor[2][3] << " " << coor[3][3] << " " << coor[4][3] << " " << coor[5][3] << " " << coor[6][3] << " " << coor[7][3] << " " << coor[8][3] << endl;
    cout << "4" << " " << coor[1][4] << " " << coor[2][4] << " " << coor[3][4] << " " << coor[4][4] << " " << coor[5][4] << " " << coor[6][4] << " " << coor[7][4] << " " << coor[8][4] << endl;
    cout << "5" << " " << coor[1][5] << " " << coor[2][5] << " " << coor[3][5] << " " << coor[4][5] << " " << coor[5][5] << " " << coor[6][5] << " " << coor[7][5] << " " << coor[8][5] << endl;
    cout << "6" << " " << coor[1][6] << " " << coor[2][6] << " " << coor[3][6] << " " << coor[4][6] << " " << coor[5][6] << " " << coor[6][6] << " " << coor[7][6] << " " << coor[8][6] << endl;
    cout << "7" << " " << coor[1][7] << " " << coor[2][7] << " " << coor[3][7] << " " << coor[4][7] << " " << coor[5][7] << " " << coor[6][7] << " " << coor[7][7] << " " << coor[8][7] << endl;
    cout << "8" << " " << coor[1][8] << " " << coor[2][8] << " " << coor[3][8] << " " << coor[4][8] << " " << coor[5][8] << " " << coor[6][8] << " " << coor[7][8] << " " << coor[8][8] << endl;
}
```

3.2.3 Validating the Coordinated

This problem was the hardest problem that I encountered in this exercise. I encountered this when I was validating the coordinated entered by each player. The validations were that player 1's coordinates needed to be between or equal to one and eight and not in a place where a treasure has already been placed. Then the validations for player 2 needed to be they can guess in a place that they have already guessed.

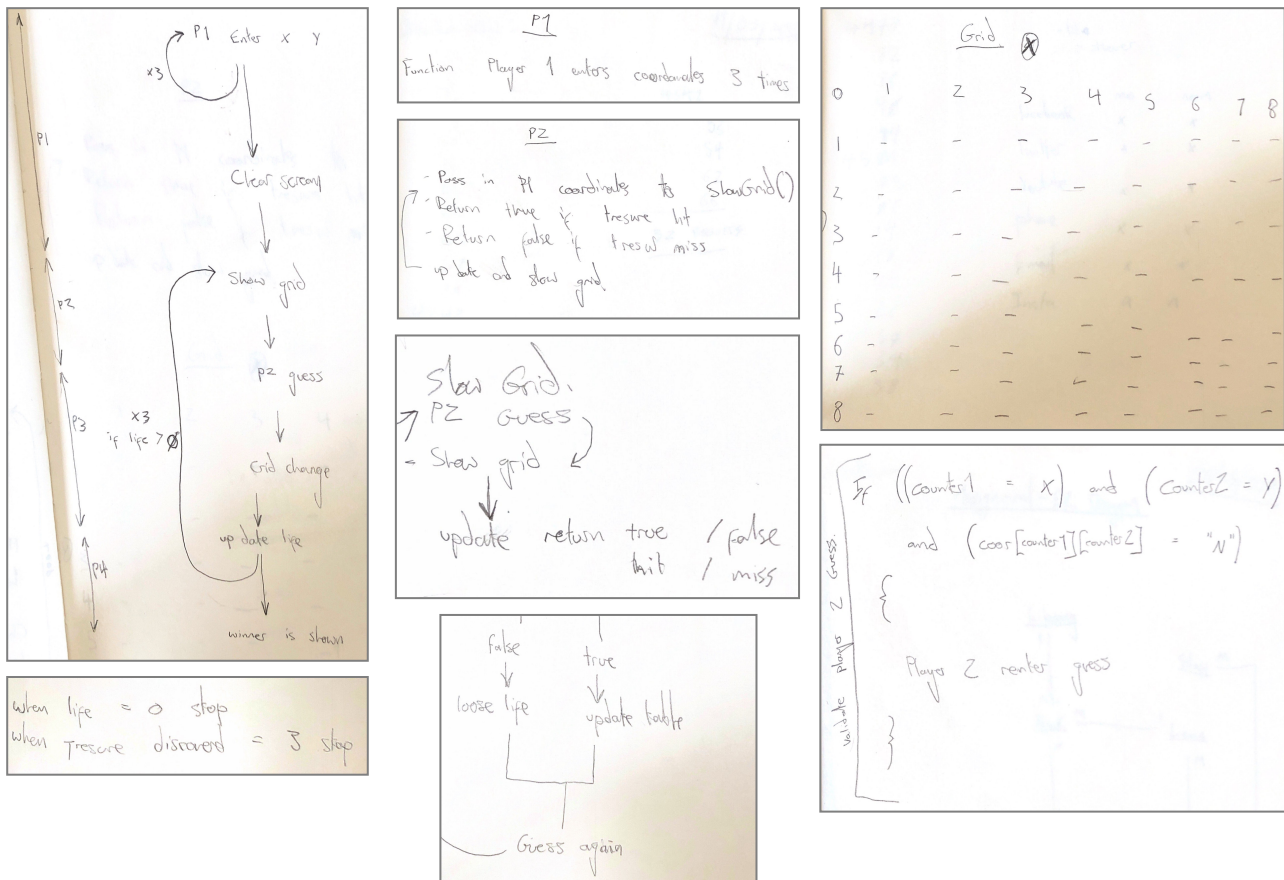
To do the validation for player 1 I added two functions. One was to check the treasure did match and the other to make sure it was in the size range. Both these functions work in a similar way, all the values get passed in the to functions, and depending on the size or if they match the functions will return a boolean value of true or false. Then the program will make the users reenter the value depending on the return value of the function. Both functions are below.

```
bool CheckTresure(int x1, int y1)
{
    if((x1 > 9) || (x1 < 1))
    {
        return true;
    }
    if((y1 > 8) || (y1 < 0))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

```
bool CheckTresureMatch(int x1, int y1, int x2, int y2)
{
    if((x1 == x2) && (y1 == y2))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

3.3 Problem Solving Before Code

Below is all the screenshots of the planning that was done before coding this exercise:



3.4 Stages of Development

The first stage of making this game was getting player 1 to enter the coordinates of the first three treasures. This was done by using a function to get the X coordinate and then another to get the Y coordinate. These same functions were used to get all three coordinates, they return the coordinate entered by player 1. The next stage to complete was clearing the screen so player 2 couldn't see where the treasures were placed and outputting a blank grid. To do this the program calls a function that outputs 1000 end line commands in a loop, then calls another function that outputs a blank grid so player 2 can choose the spot to guess.

After completing this, the next section was get player 2 to guess, after each guess an updated grid must be shown and this must be repeated until player 2 either guesses all the treasures or runs out of lives. To do this section I made a while loop that's condition was while the points were below three and lives above zero. This loop is run and inside it player 2 is made to enter their guesses, this is using the same functions that are used when player 1 is entering their coordinates. Next there is a section working out if the space that was just guessed by player 2 is a treasure then add a point or if it isn't lose a life. The grid is updated with the guess and then outputted and the loop reruns.

The final section that was added at the end was the validation, this is done in different sections. These sections are for player 1 checking the coordinates are correct and making sure the treasures aren't placed in the same place. This is done using two functions, the first one is validating the number size. The boolean function is called and the coordinates are passed in to it, these are then checked that they are above zero and less than nine. If they both are valid numbers, then the function returns true, if the numbers need to be reentered as they aren't valid, then the function returns false. To check the treasures aren't matching a function gets passed in the coordinates, checks if they are equal to each other and if they are it returns a boolean value to the main. If this value is true then player 1 is made to reenter the coordinates of the matching treasure.

The validation for player 2 is if they are guessing in a place on the grid that they have already guessed, then they need to reenter the coordinates. This was done in one function that does the check if the guesses are matching. If any of the guesses are in the same place as another then the function returns true and player 2 is made to enter the values again. The function is called 'CheckGuess' and there is screenshot showing it.

```
bool CheckGuess(string coor[10][10], int x, int y)
{
    int i = 0;
    for(int counter1 = 1; counter1 < 10; counter1++)
    {
        for(int counter2 = 1; counter2 < 10; counter2++)
        {
            if((counter1 == x) && (counter2 == y) && (coor[counter1][counter2] == "N"))
            {
                i = 1;
                break;
            }
        }
    }

    if(i == 1)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

3.5 Testing

3.5.1 Player 1 Number Size

Aim of the Test

The aim of this test is to make sure that the validation on the numbers is working. This will show if the user can enter numbers that are equal or between one and eight.

Inputs

Position	Player 1 - Treasure						Player 2 - Guess		
	1	1	2	2	3	3	1	2	3
X	0	1	8	8	11	5	1	2	3
Y	3	3	9	8	10	5	1	2	3

Expected Output

The expected outcome is player 1 should be made to reenter all the treasure placements as either the X value or the Y value is breaking the validation on each guess.

Actual Output

As Expected

```
Treasure 1
X - 0
Y - 3

Re-enter Treasure 1
X - 1
Y - 3

Treasure 2
X - 8
Y - 9

Re-enter Treasure 2
X - 8
Y - 8

Treasure 3
X - 11
Y - 10

Re-enter Treasure 3
X - 5
Y - 5
```

3.5.2 Player 1 Coordinates Previously Picked

Aim of the Test

The aim of this test is to make sure that when player 1 is entering their treasure coordinates they can't place a treasure in the same place as one previously placed.

Inputs

Position	Player 1 - Treasure					Player 2 - Guess		
	1	2	2	3	3	1	2	3
X	1	1	3	1	5	1	2	3
Y	1	1	3	1	5	1	2	3

Expected Output

The expected outcome is player 1 should be made to reenter placement of treasure two and three as they are both the same as treasure one.

Actual Output

As Expected

```
Treasure 1
X - 1
Y - 1

Treasure 2
X - 1
Y - 1

Re-enter Treasure 2
X - 3
Y - 3

Treasure 3
X - 1
Y - 1

Re-enter Treasure 3
X - 5
Y - 5
```

3.5.3 Player 1 Win

Aim of the Test

The aim of this test is to make sure that player 1 can win the game if player 2 doesn't guess any of their treasures.

Inputs

Position	Player 1 - Treasure			Player 2 - Guess		
	1	2	3	1	2	3
X	1	2	3	4	5	6
Y	1	2	3	4	5	6

Expected Output

The expected outcome is player 1 should win the game after all of player 2's wrong guesses.

Actual Output

As Expected

```
Treasure 1
X - 1
Y - 1
```

```
Treasure 2
X - 2
Y - 2
```

```
Treasure 3
X - 3
Y - 3
```

```
0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
4  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -
```

```
Player 2 enter guess
X - 4
Y - 4
```

```
0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
4  -  -  -  N  -  -  -  -
5  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -
```

You have 2 lives left

```
Player 2 enter guess
X - 5
Y - 5
```

```
Player 2 enter guess
X - 5
Y - 5

0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
4  -  -  -  N  -  -  -  -
5  -  -  -  -  N  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -
```

You have 1 lives left

```
Player 2 enter guess
X - 6
Y - 6
```

```
0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
4  -  -  -  N  -  -  -  -
5  -  -  -  -  N  -  -  -
6  -  -  -  -  -  N  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -
```

You have 0 lives left

Player 1 Wins

■

3.5.4 Player 2 Coordinates Previously Picked

Aim of the Test

The aim of this test is to make sure that player 1 player 2 can't guess a place on the grid that they have already guessed perviously.

Inputs

Position	Player 1 - Treasure			Player 2 - Guess			
	1	2	3	1	2	2	3
X	1	2	3	4	4	5	6
Y	1	2	3	4	4	5	6

Expected Output

The expected outcome is player 2 should loose the game and be made to reenter the guess two as it is in the same places as guess one.

Actual Output

As Expected

```
Treasure 1
X - 1
Y - 1

Treasure 2
X - 2
Y - 2

Treasure 3
X - 3
Y - 3
```

```
0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
4  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -
```

```
Player 2 enter guess
X - 4
Y - 4
```

```
0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
4  -  -  -  N  -  -  -  -
5  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -
```

You have 2 lives left

```
Player 2 enter guess
X - 4
Y - 4

Player 2 re-enter guess
X - 5
Y - 5
```

```
0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
```

```

4 - - - N - - -
5 - - - - N - - -
6 - - - - - - -
7 - - - - - - -
8 - - - - - - -

You have 1 lives left

Player 2 enter guess
X - 6
Y - 6

0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
4  -  -  -  N  -  -  -  -
5  -  -  -  -  N  -  -  -
6  -  -  -  -  -  N  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -

You have 0 lives left

Player 1 Wins

```

3.5.5 Player 2 Win

Aim of the Test

The aim of this test is to make sure that player 2 can win the game if they guess all of player 1's treasures.

Inputs

Position	Player 1 - Treasure			Player 2 - Guess		
	1	2	3	1	2	3
X	1	2	3	1	2	3
Y	1	2	3	1	2	3

Expected Output

The expected outcome is player 2 should win the game after all their guesses.

Actual Output

As Expected

```

Treasure 1
X - 1
Y - 1

Treasure 2
X - 2
Y - 2

Treasure 3
X - 3
Y - 3

```

```
0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
4  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -
```

Player 2 enter guess

X - 1

Y - 1

```
0  1  2  3  4  5  6  7  8
1  X  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
4  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -
```

You have hit 1 of player 1's treasure's

Player 2 enter guess

X - 2

Y - 2

```
0  1  2  3  4  5  6  7  8
1  X  -  -  -  -  -  -  -
2  -  X  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -
4  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -
```

You have hit 2 of player 1's treasure's

Player 2 enter guess

X - 3

Y - 3

```
0  1  2  3  4  5  6  7  8
1  X  -  -  -  -  -  -  -
2  -  X  -  -  -  -  -  -
3  -  -  X  -  -  -  -  -
4  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -
```

You have hit 3 of player 1's treasure's

Player 2 Wins

4. Exercise 3

4.1 Overview of the Exercise

This exercise is a connect four game, this means two players choose columns and their counter is dropped in to that column. If they get four in a vertical, horizontal or diagonal row they win the game. There is a special one use per game ability call Blitz, this can clear a column that the player chooses. The player must enter a 'B' in, then the column number.

4.2 Problem Solving and Evidence

4.2.1 Diagonal Checks to Find Winner

Finding if the winner had four of the same counters in a row diagonally was a problem that I ran in too. This problem took a while, to work it out I used two nested loops and then three if statements together to find if there was a diagonal. As there can be two different diagonals, one left and one right as show in the screenshot to the right. The left has the highest point on the left and then the right has the highest point on the right.

Diagonal Left						Diagonal Right					
1	2	3	4	5	6	1	2	3	4	5	6
-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-
x	-	-	-	-	-	-	-	-	x	-	-
-	x	-	-	-	-	-	-	x	-	-	-
-	-	x	-	-	-	x	-	-	-	-	-
-	-	-	x	-	-	-	-	-	-	-	-

1	2	3	4	5	6
[1][1]	[1][2]	[1][3]	[1][4]	[1][5]	[1][6]
[2][1]	[2][2]	[2][3]	[2][4]	[2][5]	[2][6]
[3][1]	[3][2]	[3][3]	[3][4]	[3][5]	[3][6]
[4][1]	[4][2]	[4][3]	[4][4]	[4][5]	[4][6]
[5][1]	[5][2]	[5][3]	[5][4]	[5][5]	[5][6]
[6][1]	[6][2]	[6][3]	[6][4]	[6][5]	[6][6]

After each guess by each player the program checks for a winner. So to find if there is a winner with a diagonal a two functions are called, they are 'WinnerCheckDiagonalLeft' and 'WinnerCheckDiagonalRight'. These functions have a for loop in them that count up while the counter less than seven, then in this loop is a nested for loop that counts down. This is needed as the grid is stored as a multi-dimensional array, so when finding a diagonal line one of the coordinates is a value that is getting smaller and one is getting bigger as shown on the left. In the nested loop counting down there are three if statements to find if there is a diagonal line in the grid, if there is it breaks from the loops and returns the player number from the functions that has a four diagonal counters.

4.2.2 Blitz

The hardest problem that I encountered when coding this exercise was the Blitz ability. This was the hardest as I made the game so the validation was in a function and nested in that function was another function that got the number from the user and returned it. So when the user entered a letter in it came back as an invalid entry so they needed to reenter the column number. To fix that I had to change it so that it was inputted as a string and if it wasn't 'B' then it would convert the number to a double and return that to the validation function.

If the player did enter a 'B' then there is an array that had been passed down from the 'main' in to the nested 'GetNumber' function that updates it too '2' to show being used, then to '0' to show its been used. Then the blitz function runs and collects the column number that needs to be erased and clears it.

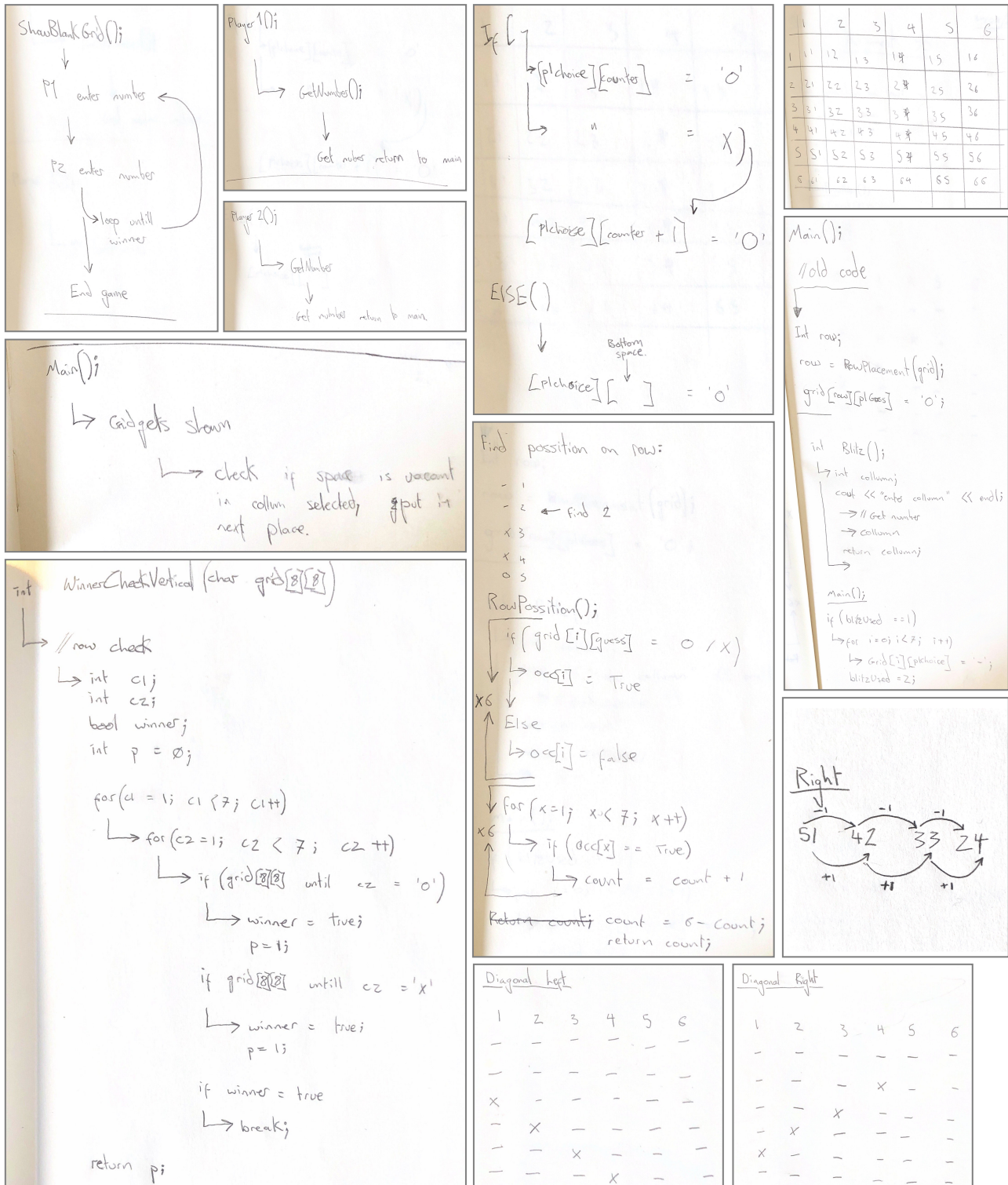
```
double GetNumber(int blitzUsed[1])
{
    double ans = 0;
    string answer;
    stringstream stream;
    string input = "";
    getline(cin, answer);
    stringstream geek(answer);
    if(answer == "B")
    {
        ans = Blitz(blitzUsed);
    }
    else
    {
        geek >> ans;
    }
    return ans;
}
```

```
int Blitz(int blitzUsed[1])
{
    int collumn;
    cout << endl << "Enter collumn number to erase:" << endl;
    stringstream stream;
    string input = "";
    getline(cin, input);
    stream.str(input);
    stream >> collumn;
    stream.clear();
    blitzUsed[0] = 2;
    return collumn;
}
```

```
//Blitz
if (blitzUsed[0] == 2)
{
    for(counter1 = 1; counter1 < 7; counter1++)
    {
        grid[counter1][p1choice] = '-';
    }
    blitzUsed[0] = 0;
}
```

4.3 Problem Solving Before Code

Below is all the screenshots of the planning that was done before coding this exercise:



4.4 Stages of Development

4.4.1 Input choice

The first sections is getting the choices from the user and outputting the grid. This is one of the main sections of the game. There is a while loop that has nested in it player 1's choice, grid output with updated choice, player 2's choice and grid output with updated choice. This is done by a function that gets the choice from the user of the column that their counter will be placed, nested in this function is another function called 'GetNumber'. After this another function runs and it works out the row that the counter needs to be placed as it can't replace a counter that is already in that column, it needs to go in the lowest vacant place. This function creates an array that if there is an occupied space it turns to true. After this the for loop count how many occupied places there are, then takes this number from 6 as the grids lowest row is sixth and top row is the first. A code snippet of the function is placed to the side. After the count is returned, it is used to update the grid.

```
int RowPlacement(char grid[8][8], int choice)
{
    bool occ[7];
    int counter;
    int count = 0;
    //Checking Rows
    for(counter = 1; counter < 7; counter++)
    {
        if(grid[counter][choice] == 'O' || grid[counter][choice] == 'X')
        {
            occ[counter] = true;
        }
        else
        {
            occ[counter] = false;
        }
    }
    for(counter = 1; counter < 7; counter++)
    {
        if(occ[counter] == true)
        {
            count++;
        }
    }
    count = 6 - count;
    return count;
}
```

After this there is a grid output section that shows the grid. As the grid gets updated in the previous section all this section does it output the grid as seen below.

```
//Grid Output
cout << endl << "1" << " " << "2" << " " << "3" << " " << "4" << " " << "5" << " " << "6";
cout << " || Player 1 = O | Blitz = " << blitzUsed[0] << " || Player 2 = X | Blitz = " << blitzUsed[0] << " ||" << endl;
for(counter1 = 1; counter1 < 7; counter1++)
{
    for(counter2 = 1; counter2 < 7; counter2++)
    {
        cout << grid[counter1][counter2] << " ";
    }
    cout << endl;
}
```

4.4.2 Winner Checks

Vertical

To do this check to see if there is a winner with four of their counters in a vertical line the 'main' calls a function that has a for loop, in this loop is a nested for loop and then nested in the second for loop is three if statements. These if statements check the grid for any vertical winners, if there is a winner then the function returns the player that has won the game. It works by the inner if statements not changing the column and scanning the rows for the counters.

```
int WinnerCheckVertical(char grid[8][8])
{
    int counter1;
    int counter2;
    bool winner = false;
    int p = 0;
    for(counter1 = 1; counter1 < 7; counter1++)
    {
        for(counter2 = 1; counter2 < 7; counter2++)
        {
            if((grid[counter2][counter1] == 'O') && (grid[counter2 + 1][counter1] == 'O') && (grid[counter2 + 2][counter1] == 'O') && (grid[counter2 + 3][counter1] == 'O'))
            {
                winner = true;
                p = 1;
            }
            if((grid[counter2][counter1] == 'X') && (grid[counter2 + 1][counter1] == 'X') && (grid[counter2 + 2][counter1] == 'X') && (grid[counter2 + 3][counter1] == 'X'))
            {
                winner = true;
                p = 2;
            }
            if(winner == true)
            {
                break;
            }
        }
    }
    return p;
}
```

Horizontal

To do this check to see if there is a winner with four of their counters in a horizontal line the 'main' calls a function. This function is the same as the vertical function, but instead of the inner if statements not changing the column and scanning the rows for the counters, it does the opposite and scans the columns for matching counters and keeps the row the same.

```
int WinnerCheckHorizontal(char grid[8][8])
{
    int counter1;
    int counter2;
    int p = 0;
    bool winner = false;
    for(counter1 = 1; counter1 < 7; counter1++)
    {
        for(counter2 = 1; counter2 < 7; counter2++)
        {
            if((grid[counter1][counter2] == 'O') && (grid[counter1][counter2 + 1] == 'O') && (grid[counter1][counter2 + 2] == 'O') && (grid[counter1][counter2 + 3] == 'O'))
            {
                winner = true;
                p = 1;
            }
            if((grid[counter1][counter2] == 'X') && (grid[counter1][counter2 + 1] == 'X') && (grid[counter1][counter2 + 2] == 'X') && (grid[counter1][counter2 + 3] == 'X'))
            {
                winner = true;
                p = 2;
            }
            if(winner == true)
            {
                break;
            }
        }
    }
    return p;
}
```

Diagonal Left

This check is to see if there is a winner with their counters in a diagonal line with the left most counter highest. It works using a loop with a countdown loop nested in it. These loops allow the program to check every combination on the for a winning diagonal line.

```
int WinnerCheckDiagonalLeft(char grid[8][8])
{
    int counter1;
    int counter2;
    int maxCount = 7;
    int countdown = maxCount - 1;
    int p = 0;
    bool winner = false;
    for(counter1 = 0; counter1 < 7; counter1++)
    {
        for(counter2 = 0; counter2 < maxCount; counter2++)
        {
            if((grid[counter1][counter2] == 'O') && (grid[counter1 + 1][counter2 - 1] == 'O') && (grid[counter1 + 2][counter2 - 2] == 'O') && (grid[counter1 + 3][counter2 - 3] == 'O'))
            {
                winner = true;
                p = 1;
            }
            if((grid[counter1][counter2] == 'X') && (grid[counter1 + 1][counter2 - 1] == 'X') && (grid[counter1 + 2][counter2 - 2] == 'X') && (grid[counter1 + 3][counter2 - 3] == 'X'))
            {
                winner = true;
                p = 2;
            }
            if(winner == true)
            {
                break;
            }
            countdown--;
        }
    }
    return p;
}
```

1	2	3	4
[1][1]	[1][2]	[1][3]	[1][4]
[2][1]	[2][2]	[2][3]	[2][4]
[3][1]	[3][2]	[3][3]	[3][4]
[4][1]	[4][2]	[4][3]	[4][4]
[5][1]	[5][2]	[5][3]	[5][4]
[6][1]	[6][2]	[6][3]	[6][4]

Diagonal Right

This check is to see if there is a winner with their counters in a diagonal line with the right most counter highest. This works by calling a function called 'WinnerCheckDiagonalRight' that has a loop then a nested countdown loop in that. This is needed as show in the grid when starting at column one and following the winning counters, the row number is decreasing by one each time. So this loop creates a variable that counts down each time the loop goes around, and this number is used to identify the row.

```
int WinnerCheckDiagonalRight(char grid[8][8])
{
    int counter1;
    int counter2;
    int maxCount = 7;
    int countdown = maxCount - 1;
    int p = 0;
    bool winner = false;
    for(counter1 = 0; counter1 < 7; counter1++)
    {
        maxCount = 7;
        countdown = maxCount - 1;
        for(counter2 = 0; counter2 < maxCount; counter2++)
        {
            if((grid[counter1][countdown] == 'O') && (grid[counter1 + 1][countdown + 1] == 'O') && (grid[counter1 + 2][countdown + 2] == 'O') && (grid[counter1 + 3][countdown + 3] == 'O'))
            {
                winner = true;
                p = 1;
                break;
            }
            if((grid[counter1][countdown] == 'X') && (grid[counter1 + 1][countdown + 1] == 'X') && (grid[counter1 + 2][countdown + 2] == 'X') && (grid[counter1 + 3][countdown + 3] == 'X'))
            {
                winner = true;
                p = 2;
                break;
            }
            if(winner == true)
            {
                break;
            }
            countdown--;
        }
    }
    return p;
}
```

1	2	3	4
[1][1]	[1][2]	[1][3]	[1][4]
[2][1]	[2][2]	[2][3]	[2][4]
[3][1]	[3][2]	[3][3]	[3][4]
[4][1]	[4][2]	[4][3]	[4][4]
[5][1]	[5][2]	[5][3]	[5][4]
[6][1]	[6][2]	[6][3]	[6][4]

4.5 Testing

4.5.1 Player 1 Wins - Horizontal

Aim of the Test

The aim of this test is to show player 1 can win horizontally.

Input

P1	P2	P1	P2	P1	P2	P1	P2
1	1	2	2	3	3	4	6

Expected Output

Player 1 wins the game.

Actual Output

As Expected

```

1    2    3    4    5    6    || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-    -    -    -    -    -
-    -    -    -    -    -
-    -    -    -    -    -
-    -    -    -    -    -
X    X    X    -    -    -
0    0    0    0    -    -

Player 1 wins

```

4.5.2 Player 1 Wins - Vertical

Aim of the Test

The aim of this test is to show player 1 can win vertically.

Input

P1	P2	P1	P2	P1	P2	P1	P2
1	2	1	2	1	2	1	6

Expected Output

Player 1 wins the game.

Actual Output

As Expected

```

1    2    3    4    5    6    || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-    -    -    -    -    -
-    -    -    -    -    -
0    -    -    -    -    -
0    X    -    -    -    -
0    X    -    -    -    -
0    X    -    -    -    -

Player 1 wins

```

4.5.3 Player 1 Wins - Diagonal Left

Aim of the Test

The aim of this test is to show player 1 can win diagonally left.

Input

P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1
4	3	3	2	2	1	2	1	1	6	1

Expected Output

Player 1 wins the game.

Actual Output

As Expected

```

1  2  3  4  5  6  || Player 1 = O | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-  -  -  -  -  -
-  -  -  -  -  -
O  -  -  -  -  -
O  O  -  -  -  -
X  O  O  -  -  -
X  X  X  O  -  X

Player 1 wins

```

4.5.4 Player 1 Wins - Diagonal Right

Aim of the Test

The aim of this test is to show player 1 can win diagonally right.

Input

P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1
1	2	2	3	3	4	3	4	4	6	4

Expected Output

Player 1 wins the game.

Actual Output

As Expected

```

1  2  3  4  5  6  || Player 1 = O | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  O  -  -
-  -  O  O  -  -
-  O  O  X  -  -
O  X  X  X  -  X

Player 1 wins

```

4.5.5 Player 2 Wins - Horizontal

Aim of the Test

The aim of this test is to show player 2 can win horizontally.

Input

P1	P2	P1	P2	P1	P2	P1	P2
6	1	1	2	2	3	3	4

Expected Output

Player 1 wins the game.

Actual Output

As Expected

```

1   2   3   4   5   6   || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-   -   -   -   -   -
-   -   -   -   -   -
-   -   -   -   -   -
-   -   -   -   -   -
0   0   0   -   -   -
X   X   X   X   -   0

Player 2 wins

```

4.5.6 Player 2 Wins -Vertical

Aim of the Test

The aim of this test is to show player 2 can win vertically.

Input

P1	P2	P1	P2	P1	P2	P1	P2
1	2	1	2	1	2	6	2

Expected Output

Player 1 wins the game.

Actual Output

As Expected

```

1   2   3   4   5   6   || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-   -   -   -   -   -
-   -   -   -   -   -
-   X   -   -   -   -
0   X   -   -   -   -
0   X   -   -   -   -
0   X   -   -   -   0

Player 2 wins

```

4.5.7 Player 2 Wins - Diagonal Left

Aim of the Test

The aim of this test is to show player 2 can win diagonally left.

Input

P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2
1	1	1	1	3	3	6	2	2	2	6	4

Expected Output

Player 1 wins the game.

Actual Output

As Expected

```

1    2    3    4    5    6    || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-    -    -    -    -    -
-    -    -    -    -    -
X    -    -    -    -    -
O    X    -    -    -    -
X    O    X    -    -    O
O    X    O    X    -    O

Player 2 wins

```

4.5.8 Player 2 Wins - Diagonal Right

Aim of the Test

The aim of this test is to show player 2 can win diagonally right.

Input

P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2
2	1	4	3	6	2	3	4	4	3	4	6

Expected Output

Player 1 wins the game.

Actual Output

As Expected

```

1    2    3    4    5    6    || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-    -    -    -    -    -
-    -    -    -    -    -
-    -    -    X    -    -
-    -    X    O    -    -
-    X    O    X    -    O
X    O    X    O    -    O

Player 2 wins

```

4.5.9 Blitz

Aim of the Test

The aim of this test is to show that the blitz function work for either players.

Input

P1	P2	P1	P2	P1	P2	P1	P1	P2	P1	P2	P1	P2	P2
1	1	1	1	1	1	B	1	3	3	3	3	B	3

Expected Output

The chosen columns are cleared by the user.

Actual Output

As Expected

```
1  2  3  4  5  6      || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
X  -  -  -  -  -
0  -  -  -  -  -
X  -  -  -  -  -
0  -  -  -  -  -
X  -  -  -  -  -
0  -  -  -  -  -
```

```
player 1 enter column number:
B
```

```
Enter collumn number to erase:
1
```

```
1  2  3  4  5  6      || Player 1 = 0 | Blitz = 0 || Player 2 = X | Blitz = 1 ||
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
```

```
1  2  3  4  5  6      || Player 1 = 0 | Blitz = 0 || Player 2 = X | Blitz = 1 ||
-  -  0  -  -  -
-  -  X  -  -  -
-  -  0  -  -  -
-  -  X  -  -  -
-  -  0  -  -  -
-  -  X  -  -  -
```

```
player 2 enter column number:
B
```

```
Enter collumn number to erase:
3
```

```
1  2  3  4  5  6      || Player 1 = 0 | Blitz = 0 || Player 2 = X | Blitz = 0 ||
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
```

4.5.10 Validation

Aim of the Test

The aim of this test is to show player 1 and player 2 can only enter valid numbers. Valid numbers are whole numbers between or equal to 1 and 6.

Input

P1	P1	P2	P2	P1	P1	P2	P2	P1	P1	P2	P2
0	1	0	1	7	6	7	6	5.5	5	5.5	5

Expected Output

Each time the player enters a value that isn't valid they will need to reenter the value.

Actual Output

As Expected

```
player 1 enter column number:
0

player 1 reenter a valid column number:
1

1  2  3  4  5  6      || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
0  -  -  -  -  -
```

```
player 2 enter column number:
0

player 2 reenter a valid column number:
1

1  2  3  4  5  6      || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
X  -  -  -  -  -
0  -  -  -  -  -
```

```
player 1 enter column number:
7

player 1 reenter a valid column number:
6

1  2  3  4  5  6      || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
-  -  -  -  -  -
X  -  -  -  -  -
0  -  -  -  -  0
```

```
player 2 enter column number:
7

player 2 reenter a valid column number:
6

1      2      3      4      5      6      || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-      -      -      -      -      -
-      -      -      -      -      -
-      -      -      -      -      -
-      -      -      -      -      -
X      -      -      -      -      X
0      -      -      -      -      0
```

```
player 1 enter column number:
5.5

player 1 reenter a valid column number:
5

1      2      3      4      5      6      || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-      -      -      -      -      -
-      -      -      -      -      -
-      -      -      -      -      -
-      -      -      -      -      -
X      -      -      -      -      X
0      -      -      -      0      0
```

```
player 2 enter column number:
5.5

player 2 reenter a valid column number:
5

1      2      3      4      5      6      || Player 1 = 0 | Blitz = 1 || Player 2 = X | Blitz = 1 ||
-      -      -      -      -      -
-      -      -      -      -      -
-      -      -      -      -      -
-      -      -      -      -      -
X      -      -      -      X      X
0      -      -      -      0      0
```

5. Appendix

5.1 Appendix - Exercise 1 Code

```
//  
// main.cpp  
// SD_Assignment_1_Exercise_1  
//  
#include <string>  
#include <iostream>  
#include <sstream>  
#include <cmath>  
  
using namespace std;  
  
//FORWARD CALLING FUNCITONS  
bool ValidatonOfNumbers(double num);  
double GetPlayer1Number();  
void ClearScrean();  
void HigherOrLower(double p1Num[8]);  
void ShowValue(int p1Num[8], int counter);  
char Player2Answer();  
bool CheckAnswer(char highLow, int p1Num[8], int counter);  
  
//FUNCTIONS  
int main(int argc, char*argv[])  
{  
    stringstream stream;  
    string input = "";  
    string firstMessage;  
    int counter = 1;  
    double user1Num[8];  
    bool val;  
  
    ClearScrean();  
    firstMessage = "Player one please enter 7 values:";  
  
    //Get player 1 numbers  
    cout << firstMessage << endl;
```

```
while(counter < 8)
{
    user1Num[counter] = GetPlayer1Number();
    val = ValidatonOfNumbers(user1Num[counter]);

    if(val == true)
    {
        counter++;
    }
    else
    {
        cout << endl << endl << "Please enter a number that is between 1 and 10 and a whole number" << endl;
    }
}

ClearScrean();
HigherOrLower(user1Num);

cin.get();
return 0;
}

double GetPlayer1Number()
{
    stringstream stream;
    string input = "";
    double user1Num;

    //User 1 enter number
    cout << "enter number : ";
    getline(cin, input);
    stream.str(input);
    stream >> user1Num;

    stream.clear();

    return user1Num;
}
```

```
}
```

```
bool ValidatonOfNumbers(double number)
```

```
{
```

```
    bool size;
```

```
    bool point;
```

```
    //Size validation
```

```
    if(number >= 0 & number <= 10)
```

```
    {
```

```
        size = true;
```

```
    }
```

```
    else
```

```
    {
```

```
        size = false;
```

```
    }
```

```
    //Whole number validation
```

```
    if(round(number) == number)
```

```
    {
```

```
        point = true;
```

```
    }
```

```
    else
```

```
    {
```

```
        point = false;
```

```
    }
```

```
    //Return if value is correct
```

```
    if(size == true & point == true)
```

```
    {
```

```
        return true;
```

```
    }
```

```
    else{
```

```
        return false;
```

```
    }
```

```
}
```

```
void ClearScreen()
```

```
{  
    int counter;  
    //Clear screen  
    for(counter = 1; counter < 1000; counter ++)  
    {  
        cout << endl;  
    }  
}
```

```
void HigherOrLower(double p1Num[8])  
{  
    int counter;  
    int p1IntNum[8];  
    int playerWins = 0;  
    char higherLower;  
    bool correct;  
  
    //Make intager  
    for(counter = 1; counter < 8; counter++)  
    {  
        p1IntNum[counter] = p1Num[counter];  
    }  
  
    counter = 1;  
  
    //Decide Winner  
    while (counter < 7)  
    {  
        ShowValue(p1IntNum, counter);  
        higherLower = Player2Answer();  
        correct = CheckAnswer(higherLower, p1IntNum, counter);  
  
        if(correct == true)  
        {  
            cout << endl << "Correct" << endl << endl;  
            playerWins = 2;  
            counter ++;  
        }  
    }
```

```
        else
        {
            playerWins = 1;
            break;
        }
    }

    if(playerWins == 2)
    {
        cout << endl << "Player 2 wins." << endl << endl;
    }
    else
    {
        cout << endl << "Incorrect" << endl << endl << "Player 1 wins." << endl << endl;
    }
}

//Show Next Value
void ShowValue(int p1Num[8], int counter)
{
    cout << "Player 1s value was " << p1Num[counter] << endl << "Higher (h) or Lower (l) - ";
}

char Player2Answer()
{
    char ans;
    cin >> ans;
    return ans;
}

//Find Correct Higher Or Lower Answer
bool CheckAnswer(char highLow, int p1Num[8], int counter)
{
    char ans;

    if(p1Num[counter] > p1Num[counter + 1])
    {
        ans = 'l';
    }
}
```

```
    }  
  
    else  
    {  
        ans = 'h';  
    }  
  
    if(highLow == ans)  
    {  
        return true;  
    }  
    else  
    {  
        return false;  
    }  
}
```

5.2 Appendix - Exercise 2 Code

```
//  
// main.cpp  
// SD_Assignment_1_Exercise_2  
//  
#include <string>  
#include <iostream>  
#include <sstream>  
  
using namespace std;  
  
int Player1XCoordinate();  
int Player1YCoordinate();  
void ClearScreen();  
void BlankGrid();  
bool ShowGrid(bool progress, int p2x, int p2y);  
int Player2XCoordinate();  
int Player2YCoordinate();  
bool CheckTresureMatch(int x1, int y1, int x2, int y2);  
bool CheckTresure(int x1, int y1);  
bool CheckGuess(string coor[10][10], int x, int y);  
  
int main(int argc, char*argv[])  
{  
    stringstream stream;  
    string input = "";  
    string coor[10][10];  
    bool progress;  
    int player1Treasuer1[3];  
    int player1Treasuer2[3];  
    int player1Treasuer3[3];  
    int p2Life = 3;  
    int p2Points = 0;  
    int p2GuessX = 0;  
    int p2GuessY = 0;  
    int counter1;  
    int counter2;  
    int initial = 0;
```

```
bool matchedTresure;  
  
bool validateTresure;  
  
bool sameCoordinate;
```

```
//TRESURE 1
```

```
cout << endl << "Treasure 1" << endl;  
  
player1Treasuer1[1] = Player1XCoordinate();  
player1Treasuer1[2] = Player1YCoordinate();
```

```
validateTresure = CheckTresure(player1Treasuer1[1], player1Treasuer1[2]);  
if(validateTresure == true)  
{  
    cout << endl << "Re-enter Treasure 1" << endl;  
    player1Treasuer1[1] = Player1XCoordinate();  
    player1Treasuer1[2] = Player1YCoordinate();  
}
```

```
//TRESURE 2
```

```
cout << endl << "Treasure 2" << endl;  
  
player1Treasuer2[1] = Player1XCoordinate();  
player1Treasuer2[2] = Player1YCoordinate();
```

```
validateTresure = CheckTresure(player1Treasuer2[1], player1Treasuer2[2]);  
if(validateTresure == true)  
{  
    cout << endl << "Re-enter Treasure 2" << endl;  
    player1Treasuer2[1] = Player1XCoordinate();  
    player1Treasuer2[2] = Player1YCoordinate();  
}
```

```
matchedTresure = CheckTresureMatch(player1Treasuer1[1], player1Treasuer1[2], player1Treasuer2[1],  
player1Treasuer2[2]);  
if( matchedTresure == true)  
{  
    cout << endl << "Re-enter Treasure 2" << endl;  
    player1Treasuer2[1] = Player1XCoordinate();  
    player1Treasuer2[2] = Player1YCoordinate();  
}
```

```
//TRESURE 3

cout << endl << "Treasure 3" << endl;

player1Treasuer3[1] = Player1XCoordinate();
player1Treasuer3[2] = Player1YCoordinate();

validateTresure = CheckTresure(player1Treasuer3[1], player1Treasuer3[2]);
if(validateTresure == true)
{
    cout << endl << "Re-enter Treasure 3" << endl;
    player1Treasuer3[1] = Player1XCoordinate();
    player1Treasuer3[2] = Player1YCoordinate();
}

matchedTresure = CheckTresureMatch(player1Treasuer1[1], player1Treasuer1[2], player1Treasuer3[1],
player1Treasuer3[2]);
if( matchedTresure == true)
{
    cout << endl << "Re-enter Treasure 3" << endl;
    player1Treasuer3[1] = Player1XCoordinate();
    player1Treasuer3[2] = Player1YCoordinate();
}

matchedTresure = CheckTresureMatch(player1Treasuer2[1], player1Treasuer2[2], player1Treasuer3[1],
player1Treasuer3[2]);
if( matchedTresure == true)
{
    cout << endl << "Re-enter Treasure 3" << endl;
    player1Treasuer3[1] = Player1XCoordinate();
    player1Treasuer3[2] = Player1YCoordinate();
}

ClearScreen();
BlankGrid();

while((p2Life > 0) or (p2Points < 3))
{
    cout << endl << "Player 2 enter guess" << endl;
    p2GuessX = Player2XCoordinate();
    p2GuessY = Player2YCoordinate();
}
```

```
sameCoordinate = CheckGuess(coor, p2GuessX, p2GuessY);
if (sameCoordinate == true)
{
    while(sameCoordinate == true)
    {
        cout << endl << "Player 2 re-enter guess" << endl;
        p2GuessX = Player2XCoordinate();
        p2GuessY = Player2YCoordinate();
        sameCoordinate = CheckGuess(coor, p2GuessX, p2GuessY);
    }
}

if((player1Treasuer1[1] == p2GuessX && player1Treasuer1[2] == p2GuessY) || (player1Treasuer2[1] == p2GuessX
&& player1Treasuer2[2] == p2GuessY) || (player1Treasuer3[1] == p2GuessX && player1Treasuer3[2] == p2GuessY))
{
    progress = true;
}
else
{
    progress = false;
}

if(initial == 0)
{
    for(counter1 = 1; counter1 < 10; counter1++)
    {
        for(counter2 = 1; counter2 < 10; counter2++)
        {
            coor[counter1][counter2] = "-";
        }
    }
}
initial = 1;
if(progress == true)
{
    coor[p2GuessX][p2GuessY] = "X";
    p2Points = p2Points + 1;
}
```

```

else
{
    coor[p2GuessX][p2GuessY] = "N";
    p2Life = p2Life - 1;
}

cout << endl << "0" << " " << "1" << " " << "2" << " " << "3" << " " << "4" << " " << "5" << " " << "6" << " " <<
"7" << " " << "8" << endl;

cout << "1" << " " << coor[1][1] << " " << coor[2][1] << " " << coor[3][1] << " " << coor[4][1] << " " << coor[5]
[1] << " " << coor[6][1] << " " << coor[7][1] << " " << coor[8][1] << endl;

cout << "2" << " " << coor[1][2] << " " << coor[2][2] << " " << coor[3][2] << " " << coor[4][2] << " " << coor[5]
[2] << " " << coor[6][2] << " " << coor[7][2] << " " << coor[8][2] << endl;

cout << "3" << " " << coor[1][3] << " " << coor[2][3] << " " << coor[3][3] << " " << coor[4][3] << " " << coor[5]
[3] << " " << coor[6][3] << " " << coor[7][3] << " " << coor[8][3] << endl;

cout << "4" << " " << coor[1][4] << " " << coor[2][4] << " " << coor[3][4] << " " << coor[4][4] << " " << coor[5]
[4] << " " << coor[6][4] << " " << coor[7][4] << " " << coor[8][4] << endl;

cout << "5" << " " << coor[1][5] << " " << coor[2][5] << " " << coor[3][5] << " " << coor[4][5] << " " << coor[5]
[5] << " " << coor[6][5] << " " << coor[7][5] << " " << coor[8][5] << endl;

cout << "6" << " " << coor[1][6] << " " << coor[2][6] << " " << coor[3][6] << " " << coor[4][6] << " " << coor[5]
[6] << " " << coor[6][6] << " " << coor[7][6] << " " << coor[8][6] << endl;

cout << "7" << " " << coor[1][7] << " " << coor[2][7] << " " << coor[3][7] << " " << coor[4][7] << " " << coor[5]
[7] << " " << coor[6][7] << " " << coor[7][7] << " " << coor[8][7] << endl;

cout << "8" << " " << coor[1][8] << " " << coor[2][8] << " " << coor[3][8] << " " << coor[4][8] << " " << coor[5]
[8] << " " << coor[6][8] << " " << coor[7][8] << " " << coor[8][8] << endl;

if(progress == true)
{
    cout << endl << "You have hit " << p2Points << " of player 1's treasure's" << endl;
}
else
{
    cout << endl << "You have " << p2Life << " lives left" << endl;
}
if(p2Life == 0)
{
    cout << endl << "Player 1 Wins" << endl;
    cin.get();
    return 0;
}
if(p2Points == 3)
{

```

```
        cout << endl << "Player 2 Wins" << endl;

        cin.get();

        return 0;

    }

}
```

```
    cin.get();

    return 0;

}
```

```
int Player1XCoordinate()
{
    stringstream stream;
    string input = "";
    int coordinate;
    cout << "X - ";
    getline(cin, input);
    stream.str(input);
    stream >> coordinate;

    stream.clear();
    return coordinate;
}
```

```
int Player1YCoordinate()
{
    stringstream stream;
    string input = "";
    int coordinate;
    cout << "Y - ";
    getline(cin, input);
    stream.str(input);
    stream >> coordinate;

    stream.clear();
    return coordinate;
}
```

```
void ClearScreen()
```

```
{
    int counter;
    for(counter = 0; counter < 500; counter++)
    {
        cout << endl;
    }
}
```

```
void BlankGrid()
```

```
{
    string coor[10][10];
    int counter1;
    int counter2;

    for(counter1 = 1; counter1 < 10; counter1++)
    {
        for(counter2 = 1; counter2 < 10; counter2++)
        {
            coor[counter1][counter2] = "-";
        }
    }

    cout << "0" << " " << "1" << " " << "2" << " " << "3" << " " << "4" << " " << "5" << " " << "6" << " " << "7" << " " << "8" << endl;

    cout << "1" << " " << coor[1][1] << " " << coor[2][1] << " " << coor[3][1] << " " << coor[4][1] << " " << coor[5][1] << " " << coor[6][1] << " " << coor[7][1] << " " << coor[8][1] << endl;

    cout << "2" << " " << coor[1][2] << " " << coor[2][2] << " " << coor[3][2] << " " << coor[4][2] << " " << coor[5][2] << " " << coor[6][2] << " " << coor[7][2] << " " << coor[8][2] << endl;

    cout << "3" << " " << coor[1][3] << " " << coor[2][3] << " " << coor[3][3] << " " << coor[4][3] << " " << coor[5][3] << " " << coor[6][3] << " " << coor[7][3] << " " << coor[8][3] << endl;

    cout << "4" << " " << coor[1][4] << " " << coor[2][4] << " " << coor[3][4] << " " << coor[4][4] << " " << coor[5][4] << " " << coor[6][4] << " " << coor[7][4] << " " << coor[8][4] << endl;

    cout << "5" << " " << coor[1][5] << " " << coor[2][5] << " " << coor[3][5] << " " << coor[4][5] << " " << coor[5][5] << " " << coor[6][5] << " " << coor[7][5] << " " << coor[8][5] << endl;

    cout << "6" << " " << coor[1][6] << " " << coor[2][6] << " " << coor[3][6] << " " << coor[4][6] << " " << coor[5][6] << " " << coor[6][6] << " " << coor[7][6] << " " << coor[8][6] << endl;

    cout << "7" << " " << coor[1][7] << " " << coor[2][7] << " " << coor[3][7] << " " << coor[4][7] << " " << coor[5][7] << " " << coor[6][7] << " " << coor[7][7] << " " << coor[8][7] << endl;

    cout << "8" << " " << coor[1][8] << " " << coor[2][8] << " " << coor[3][8] << " " << coor[4][8] << " " << coor[5][8] << " " << coor[6][8] << " " << coor[7][8] << " " << coor[8][8] << endl;
}
```

```
}
```

```
int Player2XCoordinate()
```

```
{  
    stringstream stream;  
    string input = "";  
    int coordinate;  
    cout << "X - ";  
    getline(cin, input);  
    stream.str(input);  
    stream >> coordinate;  
  
    stream.clear();  
    return coordinate;  
}
```

```
int Player2YCoordinate()
```

```
{  
    stringstream stream;  
    string input = "";  
    int coordinate;  
    cout << "Y - ";  
    getline(cin, input);  
    stream.str(input);  
    stream >> coordinate;  
  
    stream.clear();  
    return coordinate;  
}
```

```
bool CheckTresureMatch(int x1, int y1, int x2, int y2)
```

```
{  
    if((x1 == x2) && (y1 == y2))  
    {  
        return true;  
    }  
    else  
    {
```

```
    return false;

}

}
```

```
bool CheckTresure(int x1, int y1)
```

```
{
    if((x1 > 8) || (x1 < 1))
    {
        return true;
    }
    if((y1 > 8) || (y1 < 1))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

```
bool CheckGuess(string coor[10][10], int x, int y)
```

```
{
    int i = 0;
    for(int counter1 = 1; counter1 < 10; counter1++)
    {
        for(int counter2 = 1; counter2 < 10; counter2++)
        {
            if((counter1 == x) && (counter2 == y) && (coor[counter1][counter2] == "N"))
            {
                i = 1;
                break;
            }
        }
    }
}
```

```
if(i == 1)
```

```
{
    return true;
```

```
    }  
    else  
    {  
        return false;  
    }  
}
```

5.3 Appendix - Exercise 3 Code

```
//  
// main.cpp  
// SD_Assignment_1_Exercise_3  
//  
#include <string>  
#include <iostream>  
#include <sstream>  
#include <cmath>  
  
using namespace std;  
  
void BlankGrid();  
void InitialGrid(char grid[8][8]);  
int Player1Choice(int blitzUsed[2]);  
int Player2Choice(int blitzUsed[2]);  
int RowPlacement(char grid[8][8], int choice);  
double GetNumber(int blitzUsed[2], int player);  
int WinnerCheckVertical(char grid[8][8]);  
int WinnerCheckHorizontal(char grid[8][8]);  
int WinnerCheckDiagonalLeft(char grid[8][8]);  
int WinnerCheckDiagonalRight(char grid[8][8]);  
int Blitz(int blitzUsed[2], int player);  
  
int main(int argc, char*argv[])  
{  
    char grid[8][8];  
    int blitzUsed[2];  
    blitzUsed[0] = 1;  
    blitzUsed[1] = 1;  
    int winnerFinal = 0;  
    int winnerV = 0;  
    int winnerH = 0;  
    int winnerDL = 0;  
    int winnerDR = 0;  
    int p1choice;  
    int p2choice;  
    int counter1;
```

```
int counter2;

int row = 0;

BlankGrid();

InitialGrid(grid);

while(winnerFinal == 0)
{
    //Player 1 Choice
    p1choice = Player1Choice(blitzUsed);
    row = RowPlacement(grid, p1choice);
    grid[row][p1choice] = 'O';

    //Blitz
    if (blitzUsed[0] == 2)
    {
        for(counter1 = 1; counter1 < 7; counter1++)
        {
            grid[counter1][p1choice] = '-';
        }
        blitzUsed[0] = 0;
    }

    //Grid Output
    cout << endl << "1" << " " << "2" << " " << "3" << " " << "4" << " " << "5" << " " << "6" << " " << " || Player 1 = O |
    Blitz = " << blitzUsed[0] << " || Player 2 = X | Blitz = " << blitzUsed[1] << " || " << endl;
    for(counter1 = 1; counter1 < 7; counter1++)
    {
        for(counter2 = 1; counter2 < 7; counter2++)
        {
            cout << grid[counter1][counter2] << " ";
        }
        cout << endl;
    }

    //Winner Check Player 1
    //Vertical
    winnerV = WinnerCheckVertical(grid);
    if(winnerV == 1)
```

```
{
    cout << endl << "Player 1 wins";
    winnerFinal = 1;
    break;
}

//Horizontal
winnerH = WinnerCheckHorizontal(grid);
if(winnerH == 1)
{
    cout << endl << "Player 1 wins";
    winnerFinal = 1;
    break;
}

//DiagonalLeft
winnerDL = WinnerCheckDiagonalLeft(grid);
if(winnerDL == 1)
{
    cout << endl << "Player 1 wins";
    winnerFinal = 1;
    break;
}

//DiagonalRight
winnerDR = WinnerCheckDiagonalRight(grid);
if(winnerDR == 1)
{
    cout << endl << "Player 1 wins";
    winnerFinal = 1;
    break;
}

//Player 2 Choice
p2choice = Player2Choice(blitzUsed);
row = RowPlacement(grid, p2choice);
grid[row][p2choice] = 'X';

//Blitz
if (blitzUsed[1] == 2)
{
```

```

        for(counter1 = 1; counter1 < 7; counter1++)
        {
            grid[counter1][p2choice] = '-';
        }
        blitzUsed[1] = 0;
    }

    //Grid Ouput
    cout << endl << "1" << " " << "2" << " " << "3" << " " << "4" << " " << "5" << " " << "6" << " " << " || Player 1 = O |
    Blitz = " << blitzUsed[0] << " || Player 2 = X | Blitz = " << blitzUsed[1] << " ||" << endl;

    for(counter1 = 1; counter1 < 7; counter1++)
    {
        for(counter2 = 1; counter2 < 7; counter2++)
        {
            cout << grid[counter1][counter2] << " ";
        }
        cout << endl;
    }

    //Winner Check Player 2
    //Vertical
    winnerV = WinnerCheckVertical(grid);
    if(winnerV == 2)
    {
        cout << endl << "Player 2 wins";
        winnerFinal = 2;
        break;
    }
    //Horizontal
    winnerH = WinnerCheckHorizontal(grid);
    if(winnerH == 2)
    {
        cout << endl << "Player 2 wins";
        winnerFinal = 2;
        break;
    }
    //DiagonalLeft
    winnerDL = WinnerCheckDiagonalLeft(grid);
    if(winnerDL == 2)

```

```
{
    cout << endl << "Player 2 wins";
    winnerFinal = 2;
    break;
}
//DiagonalRight
winnerDR = WinnerCheckDiagonalRight(grid);
if(winnerDR == 2)
{
    cout << endl << "Player 2 wins";
    winnerFinal = 2;
    break;
}
}
cin.get();
return 0;
}
```

void BlankGrid()

```
{
    int counter1;
    int counter2;
    char grid[8][8];

    for(counter1 = 0; counter1 < 8; counter1++)
    {
        for(counter2 = 0; counter2 < 8; counter2++)
        {
            grid[counter1][counter2] = '-';
        }
    }
}
```

```
cout << "1" << " " << "2" << " " << "3" << " " << "4" << " " << "5" << " " << "6" << " " << "7" << " " << "8" << endl;
|| Player 1 = O | Blitz = 1
|| Player 2 = X | Blitz = 1 ||" << endl;
```

```
for(counter1 = 1; counter1 < 7; counter1++)
{
    for(counter2 = 1; counter2 < 7; counter2++)
    {
```

```
        cout << grid[counter1][counter2] << "  ";  
    }  
    cout << endl;  
}  
}
```

```
void InitialGrid(char grid[8][8])  
{  
    int counter1;  
    int counter2;  
  
    for(counter1 = 0; counter1 < 8; counter1++)  
    {  
        for(counter2 = 0; counter2 < 8; counter2++)  
        {  
            grid[counter1][counter2] = '-';  
        }  
    }  
}
```

```
int Player1Choice(int blitzUsed[2])  
{  
    double ans;  
    bool size = false;  
    bool whole = false;  
    int pNumber = 0;  
    cout << endl << endl << "player 1 enter column number:" << endl;  
    ans = GetNumber(blitzUsed, pNumber);  
  
    while(size == false || whole == false)  
    {  
        if(ans >= 7 || ans <= 0)  
        {  
            cout << endl << "player 1 reenter a valid column number:" << endl;  
            ans = GetNumber(blitzUsed, pNumber);  
        }  
        else  
        {  
            size = true;  
            whole = true;  
            pNumber = ans;  
        }  
    }  
}
```

```
        size = true;
    }
    if(round(ans) == ans)
    {
        whole = true;
    }
    else
    {
        cout << endl << "player 1 reenter a valid column number:" << endl;
        ans = GetNumber(blitzUsed, pNumber);
    }
    if(size == true && whole == true)
    {
        break;
    }
}
return ans;
}
```

```
int Player2Choice(int blitzUsed[2])
{
    double ans;
    bool size = false;
    bool whole = false;
    int pNumber = 1;
    cout << endl << endl << "player 2 enter column number:" << endl;
    ans = GetNumber(blitzUsed, pNumber);
    while(size == false || whole == false)
    {
        if(ans >= 7 || ans <= 0)
        {
            cout << endl << "player 2 reenter a valid column number:" << endl;
            ans = GetNumber(blitzUsed, pNumber);
        }
        else
        {
            size = true;
        }
    }
}
```

```
    if(round(ans) == ans)
    {
        whole = true;
    }
    else
    {
        cout << endl << "player 2 reenter a valid column number:" << endl;
        ans = GetNumber(blitzUsed, pNumber);
    }
    if(size == true && whole == true)
    {
        break;
    }
}
return ans;
}
```

```
double GetNumber(int blitzUsed[2], int player)
{
    double ans = 0;
    string answer;
    stringstream stream;
    string input = "";
    getline(cin, answer);
    stringstream geek(answer);
    if(answer == "B" || answer == "b")
    {
        ans = Blitz(blitzUsed, player);
    }
    else
    {
        geek >> ans;
    }
    return ans;
}
```

```
int RowPlacement(char grid[8][8], int choice)
{

```

```
bool occ[7];

int counter;

int count = 0;

//Checking Rows
for(counter = 1; counter < 7; counter++)
{
    if(grid[counter][choice] == 'O' || grid[counter][choice] == 'X')
    {
        occ[counter] = true;
    }
    else
    {
        occ[counter] = false;
    }
}

for(counter = 1; counter < 7; counter++)
{
    if(occ[counter] == true)
    {
        count++;
    }
}

count = 6 - count;

return count;
}
```

```
int WinnerCheckVertical(char grid[8][8])
{
    int counter1;
    int counter2;
    bool winner = false;
    int p = 0;
    for(counter1 = 1; counter1 < 7; counter1++)
    {
        for(counter2 = 1; counter2 < 7; counter2++)
        {
            if((grid[counter2][counter1] == 'O') && (grid[counter2 + 1][counter1] == 'O') && (grid[counter2 + 2][counter1] ==
'O') && (grid[counter2 + 3][counter1] == 'O'))
            {
```

```
        winner = true;

        p = 1;
    }

    if((grid[counter2][counter1] == 'X') && (grid[counter2 + 1][counter1] == 'X') && (grid[counter2 + 2][counter1] ==
'X') && (grid[counter2 + 3][counter1] == 'X'))
    {
        winner = true;

        p = 2;
    }

    if(winner == true)
    {
        break;
    }
}

return p;
}
```

```
int WinnerCheckHorizontal(char grid[8][8])
{
    int counter1;
    int counter2;
    int p = 0;
    bool winner = false;
    for(counter1 = 1; counter1 < 7; counter1++)
    {
        for(counter2 = 1; counter2 < 7; counter2++)
        {
            if((grid[counter1][counter2] == 'O') && (grid[counter1][counter2 + 1] == 'O') && (grid[counter1][counter2 + 2] ==
'O') && (grid[counter1][counter2 + 3] == 'O'))
            {
                winner = true;

                p = 1;
            }

            if((grid[counter1][counter2] == 'X') && (grid[counter1][counter2 + 1] == 'X') && (grid[counter1][counter2 + 2] ==
'X') && (grid[counter1][counter2 + 3] == 'X'))
            {
                winner = true;

                p = 2;
            }
        }
    }
}
```

```
    }  
    if(winner == true)  
    {  
        break;  
    }  
}  
}  
return p;  
}
```

```
int WinnerCheckDiagonalLeft(char grid[8][8])  
{  
    int counter1;  
    int counter2;  
    int maxCount = 7;  
    int countdown = maxCount - 1;  
    int p = 0;  
    bool winner = false;  
    for(counter1 = 0; counter1 < 7; counter1++)  
    {  
        for(counter2 = 0; counter2 < maxCount; counter2++)  
        {  
            if((grid[counter1][counter2] == 'O') && (grid[counter1 + 1][counter2 - 1] == 'O') && (grid[counter1 + 2][counter2 -  
2] == 'O') && (grid[counter1 + 3][counter2 - 3] == 'O'))  
            {  
                winner = true;  
                p = 1;  
            }  
            if((grid[counter1][counter2] == 'X') && (grid[counter1 + 1][counter2 - 1] == 'X') && (grid[counter1 + 2][counter2 -  
2] == 'X') && (grid[counter1 + 3][counter2 - 3] == 'X'))  
            {  
                winner = true;  
                p = 2;  
            }  
            if(winner == true)  
            {  
                break;  
            }  
            countdown --;  
        }  
    }  
}
```

```
    }  
}  
return p;  
}
```

```
int WinnerCheckDiagonalRight(char grid[8][8])  
{  
    int counter1;  
    int counter2;  
    int maxCount = 7;  
    int countdown = maxCount - 1;  
    int p = 0;  
    bool winner = false;  
    for(counter1 = 0; counter1 < 7; counter1++)  
    {  
        maxCount = 7;  
        countdown = maxCount - 1;  
        for(counter2 = 0; counter2 < maxCount; counter2++)  
        {  
            if((grid[counter1][countdown] == 'O') && (grid[counter1 + 1][countdown + 1] == 'O') && (grid[counter1 + 2]  
[countdown + 2] == 'O') && (grid[counter1 + 3][countdown + 3] == 'O'))  
            {  
                winner = true;  
                p = 1;  
                break;  
            }  
            if((grid[counter1][countdown] == 'X') && (grid[counter1 + 1][countdown + 1] == 'X') && (grid[counter1 + 2]  
[countdown + 2] == 'X') && (grid[counter1 + 3][countdown + 3] == 'X'))  
            {  
                winner = true;  
                p = 2;  
                break;  
            }  
        }  
        if(winner == true)  
        {  
            break;  
        }  
        countdown --;  
    }  
}
```

```
    }  
  
    return p;  
}
```

```
int Blitz(int blitzUsed[2], int player)  
{  
    int collumn;  
  
    cout << endl << "Enter collumn number to erase:" << endl;  
    stringstream stream;  
    string input = "";  
    getline(cin, input);  
    stream.str(input);  
    stream >> collumn;  
  
    stream.clear();  
    if(player == 0)  
    {  
        blitzUsed[0] = 2;  
    }  
    if(player == 1)  
    {  
        blitzUsed[1] = 2;  
    }  
    return collumn;  
}
```