

1809693-MarkHuntly- Semester2Assignment-C++

by Mark Huntly

FILE	11495_MARK_HUNTLY_1809693-MARKHUNTLY- SEMESTER2ASSIGNMENT-C__178138_581303764.DOCX (23.34 M)		
TIME SUBMITTED	05-APR-2019 06:15PM (UTC+0100)	WORD COUNT	6692
SUBMISSION ID	104231439	CHARACTER COUNT	33461



ACCA4017 – Software Development Programming in C++



1809693 - MARK HUNTLY

WEB DEVELOPMENT - SCHOOL OF APPLIED COMPUTING - UWTSD

TABLE OF CONTENTS

Exercise 1	2
Planning and Design	2
Implementation	8
Testing	11
exercise 2	15
Planning and Design	15
Implementation	19
Testing	21
Exercise 3	27
Planning and Design	27
Implementation	32
Testing	32
Conclusion	34
References	34
Appendix	35
Appendix A – Exercise 1 Code	35
Appendix B – Exercise 2	38
Appendix C – Exercise 3	42

1.0 EXERCISE 1

“Build a two player game. The first player enters 7 values between 1 and 10. The first value is shown to the second player and they are asked to guess whether the next value is guess higher or lower than the one being shown. Once they enter their guess, the next value is shown and the process repeats until all 7 values are shown. When all the values are revealed, the game ends and the second player wins but the first player wins if any incorrect guesses are made”.

1.1 Planning and Design

When considering the two-player game in exercise 1, it was important to break the game down into individual challenges / problems that needed to be overcome, or functions to complete a task needed within the program.

‘A function is a group of statements that is executed when it is called from some point of the program’. [1]

This was done by making a list of everything the game had to do. At the top level, and breaking certain aspects down further as follows;

- **Player entering 7 values;**
 - The ability to enter a number.
 - Ensuring the number entered was a number and not another character from the keyboard.
 - As instructed, ensuring the number was between 1 and 10.
- **Showing first number;**
 - A display showing one number.
 - An instruction to pick whether next is higher or lower.
- **Player 2 guessing whether the value is higher or lower;**
 - The player having the ability to pick higher or lower and the computer recognising the choice.
 - Ensuring only higher or lower is chosen and recognised.
- **Player 2 guess correct or incorrect;**
 - Establishing whether the next number is higher or lower than the current number.
 - Assessing Player 2 answer.
 - Comparing answer to whether number is higher or lower and assessing whether the guess is correct.
- **Player 1 winning on a wrong guess;**
 - If the guess is wrong, the computer showing a message declaring player 1 winner.
 - The game stopping at this point.
- **Player 2 guessing all seven correctly (or the six following the first);**
 - The computer recognising a correct guess.
 - The computer declaring the guess correct and playing on.
 - The computer recognising all six correct guesses and declaring player two the winner.

The next step is to establish ease of performing each task, based on the author’s perception of ease, from easiest to hardest:

1. **A display showing the first number** – The easiest task using a 'cout' to deliver the number from variables to show player 2 their first number. This was included in the main function.
2. **The ability to enter a number** - As an exercise in tutorials a function to be able to input numbers was made. This was able to be reused within the task. This could also be used with all number entering functions.[3]
3. **Ensuring the number entered was a number** – As part of the same function, it included the ability to ensure it was a whole number.
4. **Ensuring the number was between 1 and 10** – By creating a new function and including a while statement, this would allow only numbers between 1 and 10 (inclusive) to be entered.
5. **An instruction to pick higher or lower.**
6. **Ensuring only higher or lower is chosen** – Having already created some generic functions, this was made relatively easy allowing a menu of choice and using the 'ReadNumber' function with another while loop to keep it to a binary choice of 1 or 2. As there were six numbers to run through, a 'for' loop would allow the choice to run six times.
7. **The player having the ability to pick higher or lower and the computer recognising it** – Leading on from point 6, the most logical way would be to use an integer for true or false, however on reflection, a character variable would be as effective. Boolean logic in later points would work the same.
8. **The game stopping** – Although at number 8, this is possibly the easiest thing to do. The reason it is here is that it needs to fit within the task of ensuring higher or lower is right or wrong. Once this is established, a 'break' line is put into the loop to pull the program out and move on to declaring a winner.
9. **The computer declaring the guess correct and playing on** – This should use a Boolean to show the answer as true or false. Making the program recognise the Boolean with 'if / else' statements will return the result to move on or stop.
10. **The computer recognising a correct guess.**
11. **The computer recognising all six correct guesses and declaring player two the winner.**
12. **If the guess is wrong, the computer showing a message declaring player 1 winner.**
13. **Establishing whether the next number is higher or lower than the current number** - These last six points are all linked logically and so the problem was worked through as a chain of the smaller problems. Individually with Boolean variables and if statements the challenge was straight forward. The difficulty was logically linking all these together to form a working program.
14. **Comparing answer to whether number is higher or lower and assessing whether the guess is correct** – This point combines the previous tasks to give the correct result and was the hardest to work through. Again, a function was made to deliver the result using a combination of functions to pull them together.

Following on from highlighting the tasks and ranking them easy to hard, a plan was put together as to how the game was going to work. The best way to show this was graphically and then written in a logical way.

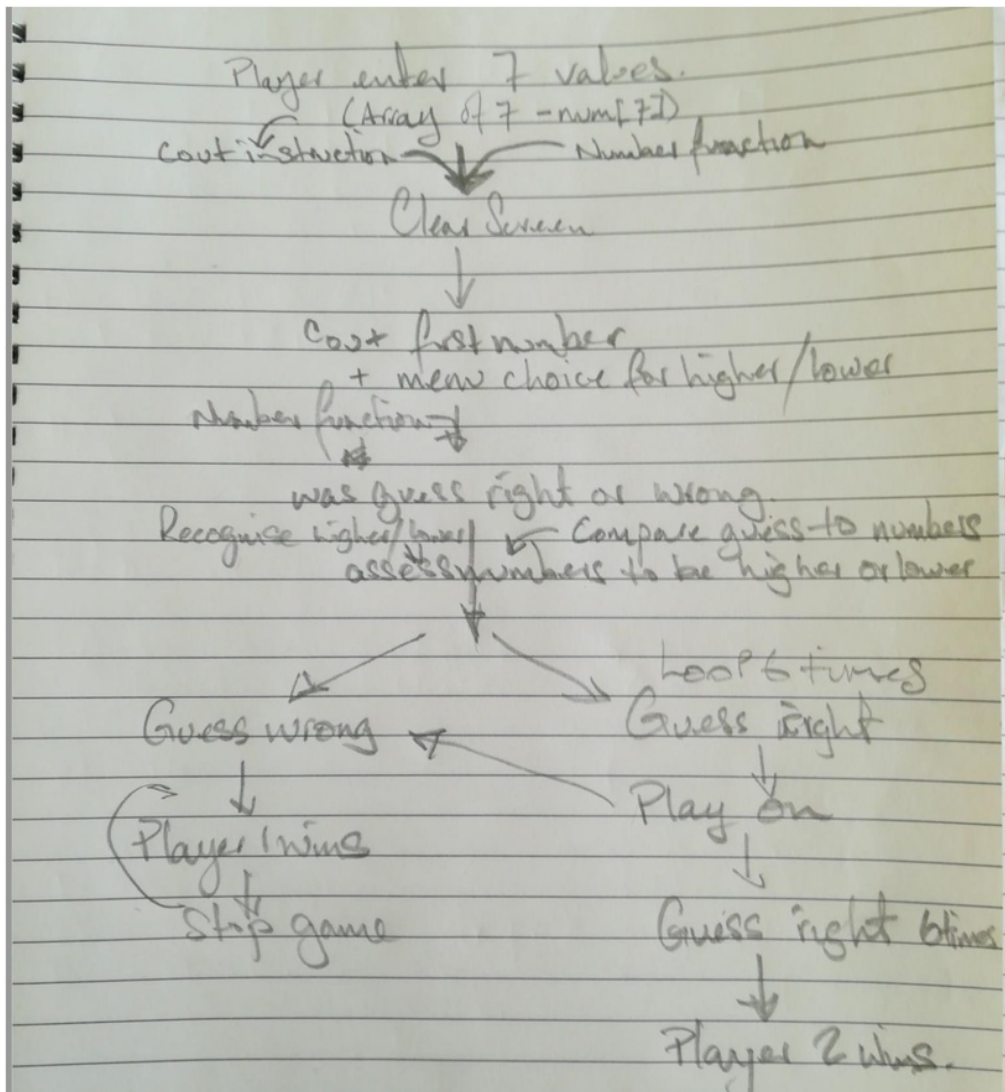


Figure 1 - First Draft game flow

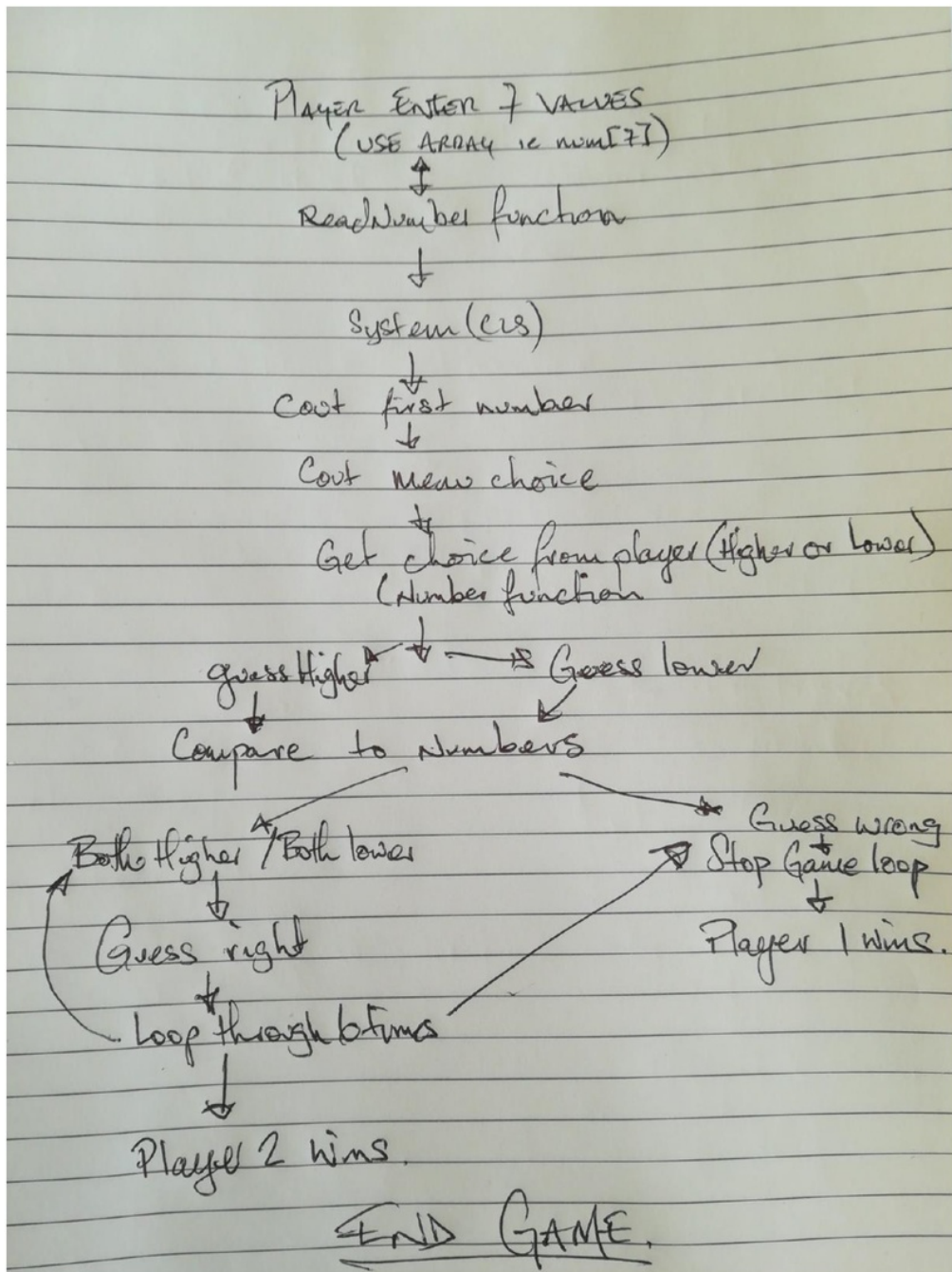


Figure 2 - second draft game flow

From this came a form of pseudocode of how the program might look.

```

Create variable num[7]

Function ReadNumber (string message)
    Display message
    input into num
    while num is not a number
        Display "make sure it is a whole number"
        input into num
    end while
    return num
end ReadNumber

Var Function ReadNumbers (int array of 7)
    Loop through 7 with variable i
    numSC[i] populate from ReadNumber
    while numSC[i] is less than 1 or more than 10
        Reuse ReadNumber and populate numSC[i]
    end while
end loop

```

Called from menu

Figure 3 - Functions created

Open with Google Docs

```

main function
    Call ReadNumbers into variable 'num[7]'
    Clear screen
    Display "First number is" num[0]

    Loop 6 times
        Display options for higher/lower
        Pass ReadNumber to variable 'choice'
        While 'choice' is not 1 or 2 (horiz)
            Pass ReadNumber again
        end While
        Switch 'choice'
            Case 1: (higher)
                if num[0] < num[1]
                    Boolean = true
                else
                    Boolean = false
                end if
            Case 2: (lower)
                if num[0] > num[1]
                    Boolean = true
                else
                    Boolean = false
                end if
        end Switch
        If Boolean = true
            Display "Correct, number is" num[1]
        else
            Display "Wrong Choice"
            break
        end if
    end Loop

    if Boolean = true
        Display "Player 2 Wins"
    else
        Display "Player 1 Wins"
    end if
end Main

```

Figure 4 - main program

1.2 Implementation

From these notes it was possible to see the logical steps the code should follow using the statements available. The code was written into the main function and then from there it was possible to create further functions and take the generic code out of the main.

As Prinz and Prinz state, “C++ supports efficient software development on the lines of the top-down principle. If you are looking to provide a solution for a more complex problem, it will help to divide the problem into smaller units.” [2]

The functions below were specifically made to tackle one problem and could call on other functions if needed.

```
int ReadNumber(string message)
{
    int num;
    string input;
    stringstream stream;

    cout << message << endl;
    getline(cin, input);
    stream.str(input);

    while ((stream >> num && stream.eof()) == false)
    {
        cout << "Please make sure it is a whole number entered" << endl;
        stream.clear();
        getline(cin, input);
        stream.str(input);
    }

    return num;
}
```

Figure 5 - Number entering function

One of the more important tasks to ask for and read a number. Figure 5 also ensured a number was input as opposed to another character.

Figure 6 passes those numbers for player one's choice and ensures the number is within the range needed.

```
void ReadNumbers(int nums[7])
{
    for (int i = 0; i < 7; i++)
    {
        nums[i] = ReadNumber("Enter a number");
        while (nums[i] < 1 || nums[i] > 10)
        {
            nums[i] = ReadNumber("Number must be in the range of 1 to 10");
        }
    }
}
```

Figure 6 - Reading the numbers function

```

for (int i = 0; i < 6; i++)
{
    DisplayMenu();
    choice = ReadNumber("Please choose option 1 or 2");

    while (choice != 1 && choice != 2)
    {
        choice = ReadNumber("Please choose again, option 1 or 2");
    }
}

```

Figure 7 - Higher or lower choice

Figure 7 also uses the function in figure 5, this time for player two, allowing the binary option of higher or lower. Figure 8 determine whether the number is higher or lower in separate functions.

<pre> bool Higher(int x, int y) { bool higher; if (x > y) { higher = true; } else { higher = false; } return higher; } </pre>	<pre> bool Lower(int x, int y) { bool lower; if (x < y) { lower = true; } else { lower = false; } return lower; } </pre>
--	---

Figure 8 - Higher and Lower Functions

Figure 9 passes the functions in figure 8 together to compare the answer to the numbers and determine a Boolean answer.

```

bool Choice (int choice, int num1, int num2)
{
    bool highLow = false;
    switch (choice)
    {
        case 1:
            highLow = Higher(num1, num2);
            break;
        case 2:
            highLow = Lower(num1, num2);
            break;
    }
    return highLow;
}

```

Figure 9 - Bringing the choice and higher or lower together

Figure 10 passes the Boolean into a function to establish the answer to be correct or wrong.

```

bool WhoWins(bool higherLower, int x)
{
    if (higherLower == true)
    {
        cout << "Correct, the number was: " << x << endl;
        return true;
    }
    else
    {
        cout << "Sorry, wrong choice." << endl;
        return false;
    }
}

```

Figure 10 - Winner Boolean function

Figure 11 takes the information to establish whether player one or 2 wins.

```
void winner(bool win)
{
    if (win == true) // COMPLETE ALL SIX - PLAYER TWO WIN!
    {
        cout << "Player Two wins!" << endl;
    }
    else // IF WRONG - PLAYER 1 WINS
    {
        cout << "Player One wins!" << endl;
    }
}
```

Figure 11 - The winner function

1.3 Testing

Each step was tested to show it working;

In the 'ReadNumbers' function, when letters and characters were input, the question was asked again, numbers above 10 and below 1 did the same and numbers 1 and 10 all worked as they should as did numbers in between. This was shown in a test scenario as shown in figure 12 with screenshots of the testing done.

ReadNumber Test				
Test	Action taken	Expected Result	Actual result	Pass / Fail
Enter Letters	Entered most letters on keyboard	Ask question again to get number	Asked question again to get number	pass
Enter symbols	Entered all symbols on keyboard	Ask question again to get number	Asked question again to get number	pass
Space bar	Entered space	Ask question again to get number	Asked question again to get number	pass
Decimal number	Entered 3 decimal numbers	Ask question again to get number	Asked question again to get number	pass
ReadNumbers Test				
Test	Action taken	Expected Result	Actual result	Pass / Fail
Enter numbers outside range	Entered 11	Ask question again to get number in range	Asked question again to get number in range	pass
	Entered 10	Break out of loop and played on	Broke out of loop and played on	pass
	Entered 9	Break out of loop and played on	Broke out of loop and played on	pass
	Entered 0	Ask question again to get number in range	Asked question again to get number in range	pass
	Entered 1	Break out of loop and played on	Broke out of loop and played on	pass
	Entered 2	Break out of loop and played on	Broke out of loop and played on	pass
Decimal number	Entered 3 decimal numbers	Ask question again to get number	Asked question again to get number	pass

Figure 12 - ReadNumbers Function Test

```

C:\Users\markh\OneDrive\Desktop\UWTSD\USB\SOFTWARE DEV\ASSIGNI
Enter a number
g
Please make sure it is a whole number entered
s
Please make sure it is a whole number entered
n
Please make sure it is a whole number entered

Please make sure it is a whole number entered
0
Number must be in the range of 1 to 10
11
Number must be in the range of 1 to 10
10
Enter a number
1
Enter a number
5
Enter a number

```

Figure 13 - testing the ReadNumbers function

The Boolean choice for player 2 was tested by including a 'cout' to see whether it was 1 or 0 (true or false) as shown in figure 11.

The Boolean that showed if player 2 had input either 1 or 2 was tested, and that the guess of higher or lower was correct. The results shown in figure 14.

HigherLower Test				
Test	Action taken	Expected Result	Actual result	Pass / Fail
Enter Letters	Entered most letters on keyboard	Ask question again to get number	Asked question again to get number	pass
Enter symbols	Entered all symbols on keyboard	Ask question again to get number	Asked question again to get number	pass
Space bar	Entered space	Ask question again to get number	Asked question again to get number	pass
Decimal number	Entered 3 decimal numbers	Ask question again to get number	Asked question again to get number	pass
HigherLower Test				
Test	Action taken	Expected Result	Actual result	Pass / Fail
Enter numbers outside range	Entered 0	Ask question again to get number in range	Asked question again to get number in range	pass
	Entered 1	Broke out of loop and played on	Broke out of loop and played on	pass
	Entered 2	Broke out of loop and played on	Broke out of loop and played on	pass
	Entered 3	Ask question again to get number in range	Asked question again to get number in range	pass
	Entered other random numbers	Ask question again to get number in range	Asked question again to get number in range	pass
Decimal number	Entered a decimal number	Ask question again to get number	Asked question again to get number	pass

Figure 14 - HigherLower Testing

```
C:\Users\markh\OneDrive\Desktop\UWTSD\USB\SOFTWARE DEV
Player Two your first number is: 1
Is the next number 1. higher or 2. lower
Please choose option 1 or 2
1
True or false = 1
Correct, the number was: 9
Is the next number 1. higher or 2. lower
Please choose option 1 or 2

Please make sure it is a whole number entered
2
True or false = 1
Correct, the number was: 2
Is the next number 1. higher or 2. lower
Please choose option 1 or 2
2
True or false = 0
```

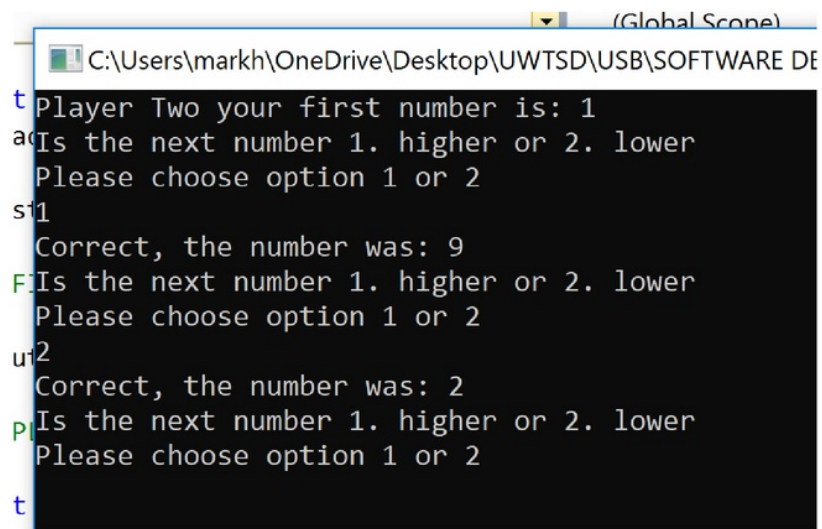
Figure 15 – Testing HigherLower Boolean

Testing of the 'ReadNumbers' function for player 2 is shown in figure 15 and 16. This also tested the choice being wrong and the 'WhoWins' and 'Winner' function. This also shows testing for player 2 wrong answer

```
C:\Users\markh\OneDrive\Desktop\UWTSD\USB\SOFTWARE DEV\ASSIGN
1 Player Two your first number is: 10
Is the next number 1. higher or 2. lower
U Please choose option 1 or 2
g
p Please make sure it is a whole number entered
j
2 Please make sure it is a whole number entered
e Please make sure it is a whole number entered
e 0
e Please choose again, option 1 or 2
5
i Please choose again, option 1 or 2
1
a Sorry, wrong choice.
e Player One wins!
Press any key to continue . . .
h
```

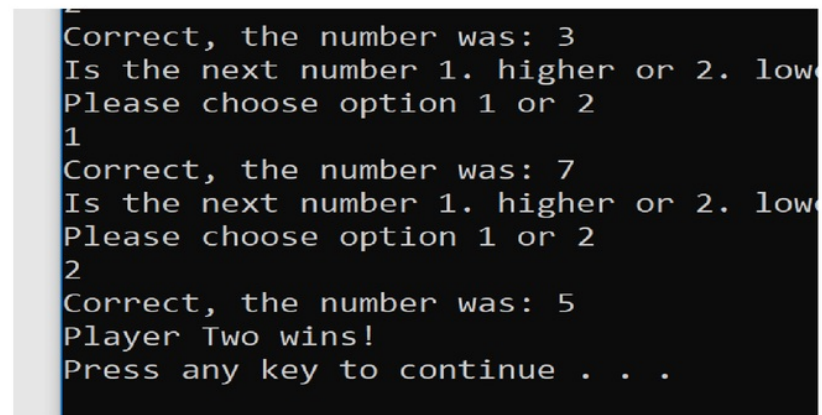
Figure 16 - testing player 2 input ReadNumbers function and wrong answer

Figures 17 and 18 shows testing for correct answers all the way through to the end of the loop and the Booleans working, with the 'WhoWins' and 'Winner' functions. This showed testing successful both ways.



```
(Global Scope)
C:\Users\markh\OneDrive\Desktop\UWTSD\USB\SOFTWARE DE
Player Two your first number is: 1
Is the next number 1. higher or 2. lower
Please choose option 1 or 2
1
Correct, the number was: 9
Is the next number 1. higher or 2. lower
Please choose option 1 or 2
2
Correct, the number was: 2
Is the next number 1. higher or 2. lower
Please choose option 1 or 2
```

Figure 17 - testing player 2 right answer



```
Correct, the number was: 3
Is the next number 1. higher or 2. low
Please choose option 1 or 2
1
Correct, the number was: 7
Is the next number 1. higher or 2. low
Please choose option 1 or 2
2
Correct, the number was: 5
Player Two wins!
Press any key to continue . . .
```

Figure 18 - testing full loop, player 2 win

2.0 EXERCISE 2

“Build a two-player game where the first player hides 3 hidden treasures and the second has to guess where they are by entering the x and y coordinates of a location on an 8x8 grid. The player starts with 3 lives. If they fail to find the treasure then an n character should be placed on the grid and they lose a life. An x should be displayed if they find the treasure, and they win if they discover all three treasures. The player grid should be re-displayed after each guess, as well as appropriate messages such as “you found the first treasure”. ”

2.1 Planning and Design

As with exercise 1, the same process was followed, starting with the list of everything the game had to do as follows;

- **Create a grid.**
 - Produce an array (possibly 2d).
 - Print the array out to look like a grid of characters.
- **Player 1 choose three coordinates.**
 - Ask for values.
 - Get values.
 - Input into the grid.
 - Hide the coordinates.
- **Player 2 guess coordinates.**
 - Ask for guesses.
 - Get values.
 - Compare values to player 1 coordinates.
- **Player 2 right or wrong.**
 - Establish compared guesses match or not.
 - Display ‘X’ or ‘N’ at the guessed coordinates.
- **Play three lives.**
 - If guess is wrong, count down lives.
 - When three lives are used, break out and end game.
- **Show winner.**
 - If all three lives used up show player 1 as winner.
 - If three correct guesses before lives used, end game and show player 2 as winner.

The tasks were again ranked in order of perceived ease to the author as follows:

1. **Hide the coordinates** – Easiest task achieved by using ‘system(“cls”)’.
2. **Ask for values**
3. **Get values** – Both of these tasks could be completed reusing the ‘ReadNumber’ and ‘ReadNumbers’ functions used in the first exercise. The message would be altered to match what was required.
4. **Compare values to player 1 coordinates**
5. **Establish compared guesses match or not** – Points four and five are connected and were perceived to be next in terms of ease as it would be an ‘if / else’ statement, possibly nested for each coordinate.
6. **If all three lives used up show player 1 as winner**
7. **If three correct guesses before lives used, end game and show player 2 as winner**
 - These two points could connect as well using ‘if / else’ statements.

8. **When three lives are used, break out and end game** – As this is linked to the above points, it falls in this position although it is slightly more complicated too.
9. **If guess is wrong, count down lives** - This should be a straight forward task of inserting a break in the 'if' statement under a variable incremented.
10. **Display 'X' or 'N' at the guessed coordinates** – Harder to do as a grid needs to be formed. After researching 2d arrays, this is something I haven't tackled before and had to understand them to make this work.
11. **Input into the grid** - As with the above task, the issue lies with the ability to make a grid before being able to input into it.
12. **Produce an array (possibly 2d)** – Having researched, it was decided a 2d array was the best route. Producing the array seemed straight forward, not dissimilar to any other array.
13. **Print the array out to look like a grid of characters** – This is perceived to be the hardest task. The research had shown it could be achieved with a nested 'for' loop and following examples would help.

The next stage was to produce a plan of what was to be done, using flow charts and a form of pseudocode to have an idea of what was to be achieved.

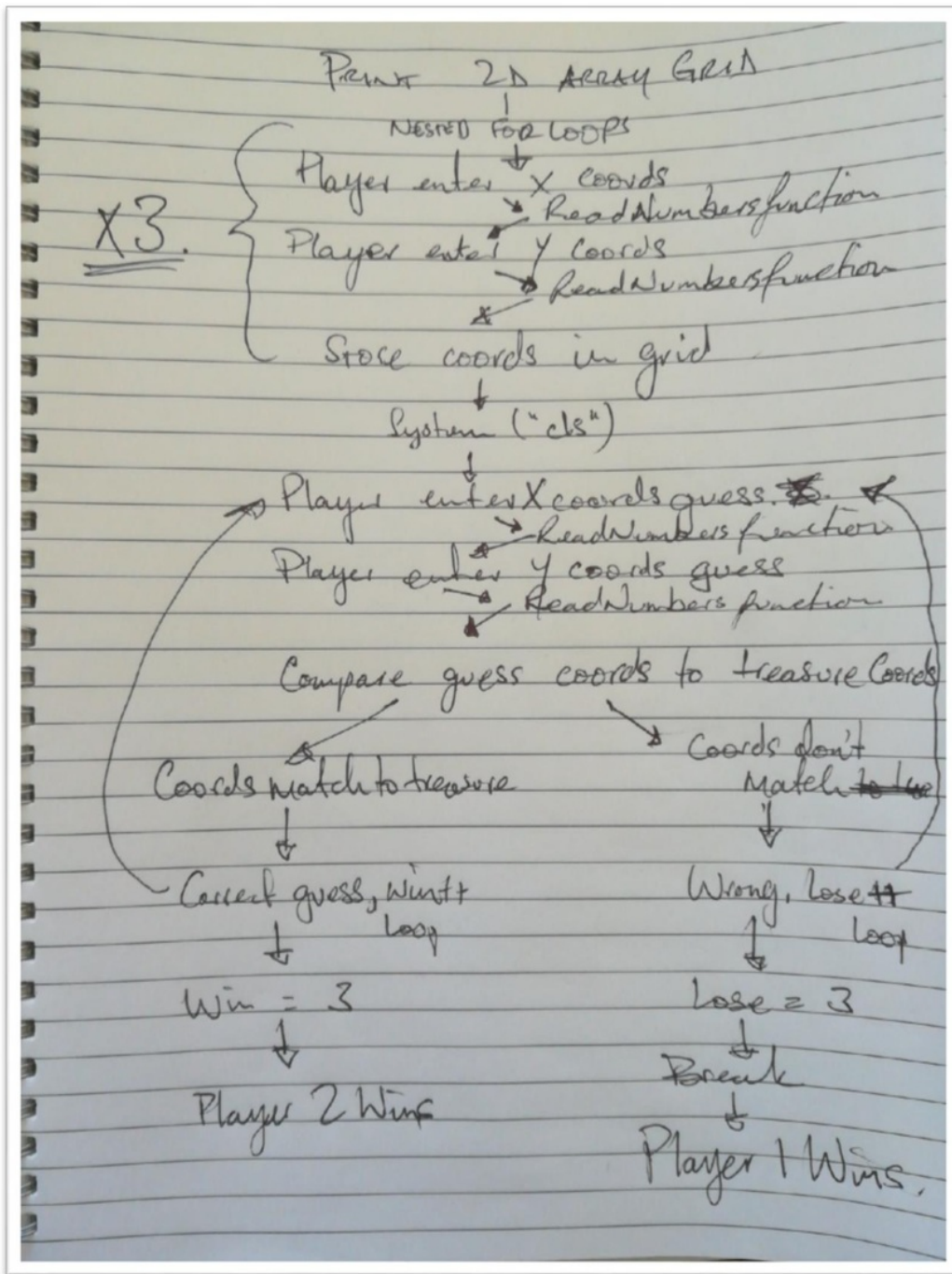


Figure 19 – Exercise 2 Game Flow

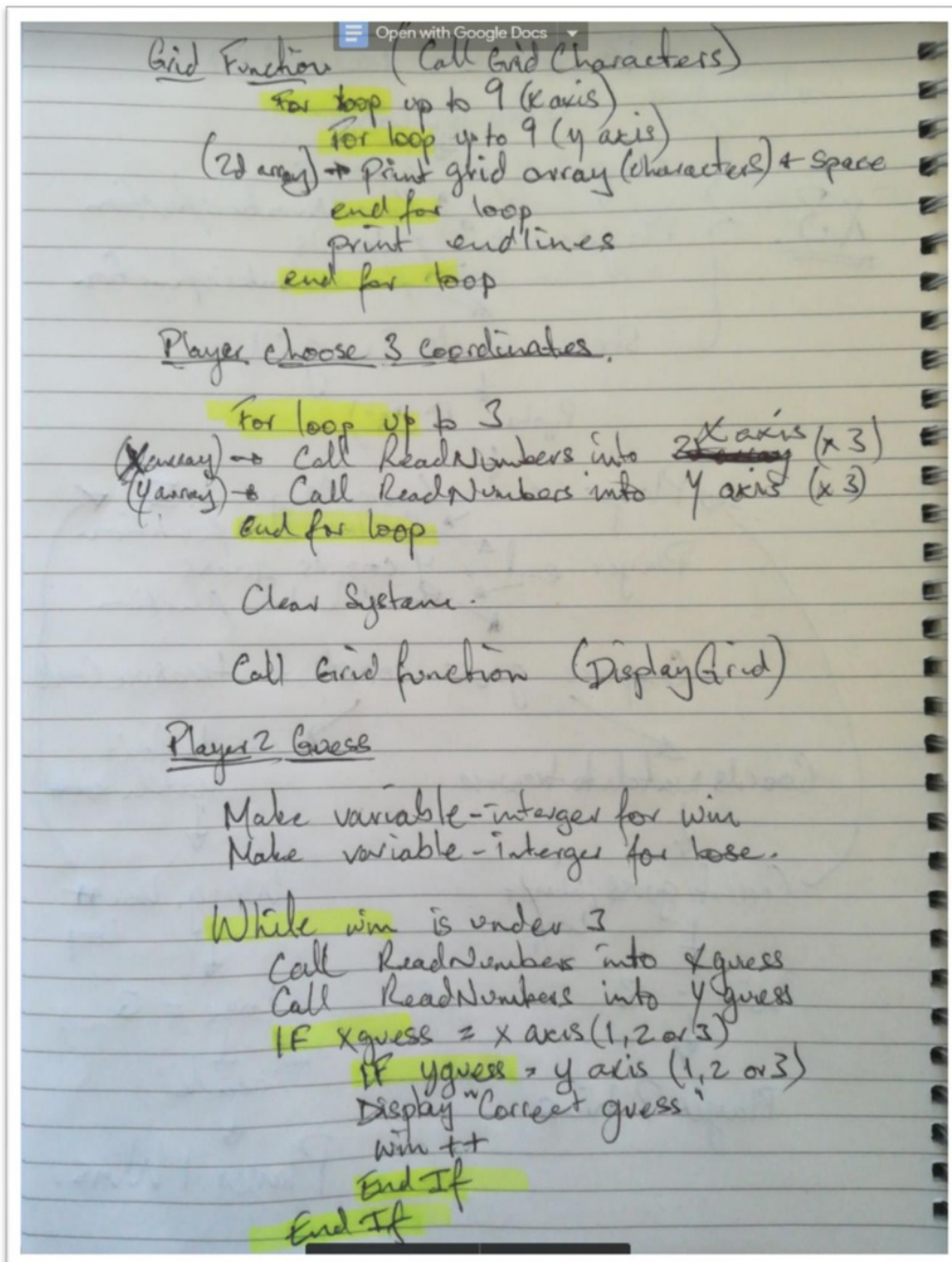


Figure 20 - Exercise 2 Main Game

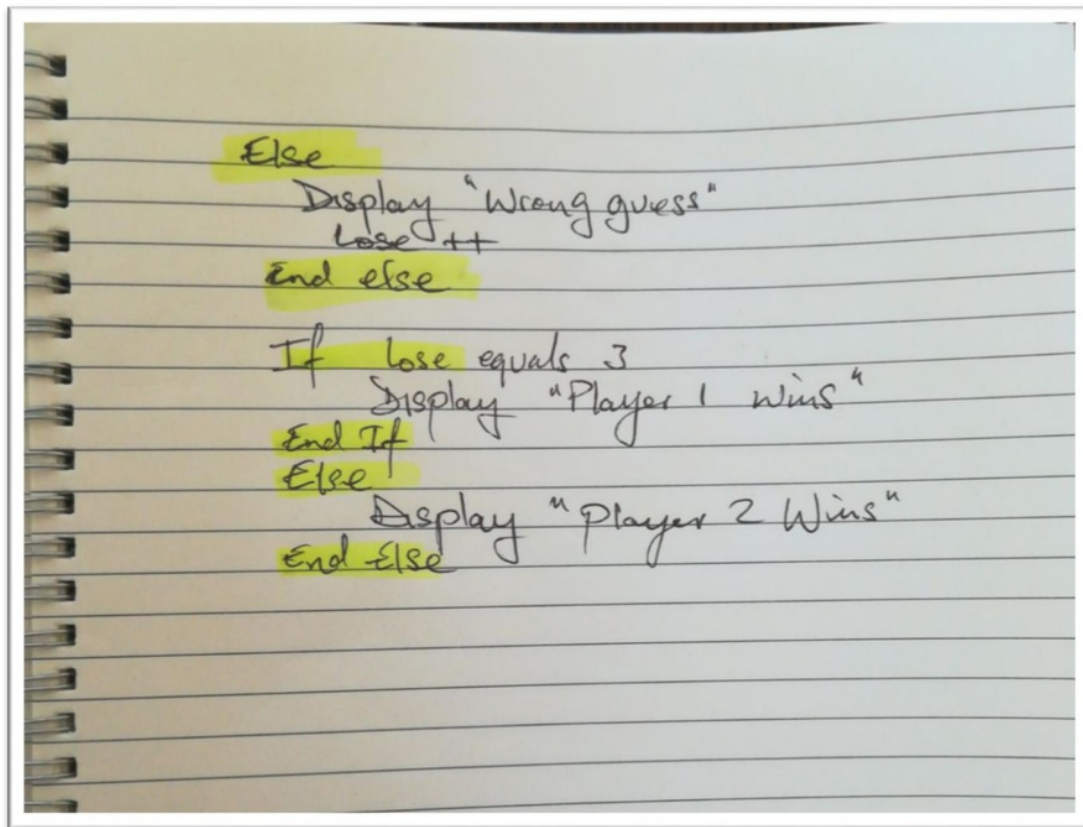


Figure 21 - Exercise 2 Main Game

2.2 Implementation

From the notes it was possible to logically follow through to code. In hindsight it is felt there should have been a little more detail in the pseudocode as there are further steps in the code itself with more functions. However, at this point it is becoming more apparent, as exercise two is worked through, that there is an importance to the logical order of a program for it to perform as required.

The hardest to understand was the 'DisplayGrid' for this exercise however with discussion and perseverance, it was possible to complete. This was completed with a two-dimensional array [2, pp. 330 - 331], as shown in figures 22 and 23. The array was declared in the main function and passed to 'DisplayGrid' for ease of changing the chars in the array when needed. This way it remained more constant.

```

void DisplayGrid(char grid[9][9])
{
    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 9; j++)
        {
            cout << grid[i][j] << "    ";
        }
        cout << endl << endl;
    }
}

```

Figure 22 – 'DisplayGrid' Function

```

159
160 int main(int argc, char *argv[])
161 {
162     char coords[9][9] = {
163         {'0','1','2','3','4','5','6','7','8'},
164         {'1','-','-','-','-','-','-','-','-'},
165         {'2','-','-','-','-','-','-','-','-'},
166         {'3','-','-','-','-','-','-','-','-'},
167         {'4','-','-','-','-','-','-','-','-'},
168         {'5','-','-','-','-','-','-','-','-'},
169         {'6','-','-','-','-','-','-','-','-'},
170         {'7','-','-','-','-','-','-','-','-'},
171         {'8','-','-','-','-','-','-','-','-'} };
172
173     int lose = 0;
174
175     DisplayGrid(coords);

```

Figure 23 - 2d Array Variable

The rest of the functions fell into place and it all appeared to work well. Full code is shown in the appendix.

2.3 Testing

Everything was tested again from scratch for this exercise. The 'ReadNumber' function went exactly as exercise 1, showing the same results as did Player 1 coordinates.

Exercise 2				
ReadNumber Test				
Test	Action taken	Expected Result	Actual result	Pass / Fail
Enter Letters	Entered most letters on keyboard	Ask question again to get number	Asked question again to get number	pass
Enter symbols	Entered all symbols on keyboard	Ask question again to get number	Asked question again to get number	pass
Space bar	Entered space	Ask question again to get number	Asked question again to get number	pass
Decimal number	Entered 3 decimal numbers	Ask question again to get number	Asked question again to get number	pass
Whole number	Entered a whole number in range	Move on to next function	Moved on to next function	pass
Player 1 Coordinates Test				
Test	Action taken	Expected Result	Actual result	Pass / Fail
Enter numbers outside range	Entered 0	Ask question again to get number in range	Asked question again to get number in range	pass
	Entered 9	Ask question again to get number in range	Asked question again to get number in range	pass
Enter numbers in range	Entered 1	Break out of loop and played on	Broke out of loop and played on	pass
	Entered 8	Break out of loop and played on	Broke out of loop and played on	pass
Decimal number	Entered a decimal number	Ask question again to get number	Asked question again to get number	pass

Figure 24 - Entering Numbers Testing

```

6  -  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -  -

Player 1 choose position 1 of 3 for your treasure
Enter X co-ordinate as a number between 0 and 8
0
Number must be in the range of 1 to 8
9
Number must be in the range of 1 to 8
1
Enter Y co-ordinate as a number between 0 and 8
0
Number must be in the range of 1 to 8
9
Number must be in the range of 1 to 8
8
Player 1 choose position 2 of 3 for your treasure
Enter X co-ordinate as a number between 0 and 8

```

Figure 25 - Entering Numbers Console

Player 2 coordinates performed as required also.

Player 2 Guess Test				
Test	Action taken	Expected Result	Actual result	Pass / Fail
Enter numbers outside range	Entered 0	Ask question again to get number in range	Asked question again to get number in range	pass
	Entered 9	Ask question again to get number in range	Asked question again to get number in range	pass
Enter numbers in range	Entered 1	Break out of loop and played on	Broke out of loop and played on	pass
	Entered 8	Break out of loop and played on	Broke out of loop and played on	pass
Decimal number	Entered a decimal number	Ask question again to get number	Asked question again to get number	fail

Figure 26 - Player 2 Input Test

```

8  - - - - -
S
Player 2 guess the position of the treasure:
Enter X co-ordinate as a number between 1 and 8
h
Please make sure it is a whole number entered
0
Number must be in the range of 1 to 8
9
Number must be in the range of 1 to 8
1
Enter Y co-ordinate as a number between 1 and 8
h
Please make sure it is a whole number entered
0
Number must be in the range of 1 to 8
9
Number must be in the range of 1 to 8
8
Good Guess, correct!
0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  X
2  -  -  -  -  -  -  -  -

```

Figure 27 - Player 2 Input Console

The first test fail was revealed when the game was played by someone else. This was the fact that coordinates could be entered more than once. There was no check to see if these coordinates had been used both for player one and player two. The fix for this was a simple one and should have been obvious, inserting a while loop into the code as in figures 29 - 32, but wasn't thought of until it was pointed out showing the importance of independent testing with an imperfect path through the program.

Player 1 Coordinates Test		Test Repetition			
Test	Action taken	Expected Result	Actual result	Pass / Fail	
Enter same coordinates again	Entered 0	Inform already chosen, ask again	Performed task and moved on	Fail	Remedied with an 'While' Loop.
Player 2 Guess Test		Test Repetition			
Test	Action taken	Expected Result	Actual result	Pass / Fail	
Enter same coordinates again	Entered 0	Inform already chosen, ask again	Performed task and moved on	Fail	Remedied with an 'While' Loop.

Figure 28 - Same coordinates test

```

while (x[i] == x[i - 1] || x[i] == x[i - 2])
{
    if (y[i] == y[i - 1] || y[i] == y[i - 2])
    {
        x[i] = ReadNumber("You can't use same coordinates twice, pick X again:");
        y[i] = ReadNumber("You can't use same coordinates twice, pick Y again:");
    }
    else
    {
        break;
    }
}
pOneCoords[x[i]][y[i]] = 'T';

```

Figure 29 - While Loop to Correct Duplication for Player 1

```

2
Player 1 choose position 2 of 3 for your treasure
Enter X co-ordinate as a number between 0 and 8
4
Enter Y co-ordinate as a number between 0 and 8
5
Player 1 choose position 3 of 3 for your treasure
Enter X co-ordinate as a number between 0 and 8
4
Enter Y co-ordinate as a number between 0 and 8
5
You can't use same coordinates twice, pick X again:

```

Figure 30 - Console showing No Duplication for Player 1

```

xGuess = PlayerTwoGuessX();
yGuess = PlayerTwoGuessY();

while (grid[xGuess][yGuess] == 'X' || grid[xGuess][yGuess] == 'N')
{
    cout << "You have already chosen that one, try again:" << endl;
    xGuess = PlayerTwoGuessX();
    yGuess = PlayerTwoGuessY();
}

if (pOneCoords[xGuess][yGuess] == 'T')
{
    cout << "Good Guess, correct!" << endl;
}

```

Figure 31 - While Loop to Correct Duplication for Player 2

```

Player 2 guess the position of the treasure:
Enter X co-ordinate as a number between 1 and 8
3
Enter Y co-ordinate as a number between 1 and 8
5
You have already chosen that one, try again:
Player 2 guess the position of the treasure:
Enter X co-ordinate as a number between 1 and 8
4
Enter Y co-ordinate as a number between 1 and 8
5
Good Guess, correct!
0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -

```

Figure 32 - Console showing No Duplication for Player 2

The next test fail was when entering guess coordinates. The system wasn't recognising some answers, and others were showing correct when they weren't. The issue was found to be in the logical flow of the code as in figure 33. The program was recognising the x coordinate guess and then looking at the y coordinate. If both matched, it was showing correct however, it was looking at them as individual entities and so producing the wrong combinations, showing them as correct.

```

while (win != 3)
{
    DisplayGrid(grid);

    xGuess = PlayerTwoGuessX();
    yGuess = PlayerTwoGuessY();

    if (grid[xGuess][yGuess] == 'X' || grid[xGuess][yGuess] == 'N')
    {
        cout << "You have already chosen that one, try again:" << endl;
        xGuess = PlayerTwoGuessX();
        yGuess = PlayerTwoGuessY();
    }

    if (xGuess == x[0] || xGuess == x[1] || xGuess == x[2])
    {
        if (yGuess == y[0] || yGuess == y[1] || yGuess == y[2])
        {
            cout << "Good Guess, correct!" << endl;

            grid[xGuess][yGuess] = 'X';

            win++;
        }
        else
        {
            cout << "Sorry, wrong guess." << endl << "Life " << wrongGuess + 1 << " from the three used up" << endl;

            grid[xGuess][yGuess] = 'N';

            wrongGuess++;
        }
    }
}

```

Figure 33 - Guess Code Causing Error

This was corrected by producing a second 'phantom' grid to hold the treasure coordinates and the grids being overlaid to compare as in figure 34.

```

while (win != 3)
{
    DisplayGrid(grid);

    xGuess = PlayerTwoGuessX();
    yGuess = PlayerTwoGuessY();

    if (grid[xGuess][yGuess] == 'X' || grid[xGuess][yGuess] == 'N')
    {
        cout << "You have already chosen that one, try again:" << endl;
        xGuess = PlayerTwoGuessX();
        yGuess = PlayerTwoGuessY();
    }

    if (pOneCoords[xGuess][yGuess] == 'T')
    {
        cout << "Good Guess, correct!" << endl;

        grid[xGuess][yGuess] = 'X';

        win++;
    }
    else

```

Figure 34 - New Code Comparing Grids

Next the incremental variables were tested to ensure the correct guesses were being registered as were the lives used.

```

Sorry, wrong guess.
Life 3 from the three used up
Win = 1
WrongGuess= 3
0  1  2  3  4  5  6  7  8
1  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  N  -  -  -
4  -  N  -  -  X  -  -  N
5  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -

****PLAYER 1 WINS!****

```

Figure 35 - Incremental Variables Working

As can be seen in figures 35 and 36, this also showed if this carried through to the Winner function and showed the correct player winning.

```

8
Good Guess, correct!
Win = 3
WrongGuess= 2
0  1  2  3  4  5  6  7  8
1  -  X  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -
3  -  -  -  -  N  -  -  -
4  -  -  -  -  X  -  -  -
5  -  -  -  N  -  -  -  -
6  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  X
8  -  -  -  -  -  -  -  -

****PLAYER 2 WINS!****

C:\Users\markh\OneDrive\Desktop\UWTSD\USB\SOFTWARE DEV
d with code 0.
To automatically close the console when debugging stop

```

Figure 36 - Incremental Variables Working

3.0 EXERCISE 3

“Build a Connect 4 style Console game which allows the players to select the column they wish to drop their counter (0 or X) down. The program should be able to detect a winning horizontal, vertical or diagonal line of 4 matching counters, then stop the game and declare the winner. However in our version of the game, we will allow each player two optional special moves that they can each play once during the game. These are:

- 1. BLITZ activated by pressing the B key and selecting a column to clear.*
- 2. TIME BOMB activated by pressing the T key and selecting a column. This places a special symbol ‘*’ in the selected column as normal. However it clears the counters surrounding it horizontally, vertically and diagonally after the next player has made 2 more moves.”*

3.1 Planning and Design

This game seems harder to create; the same process was followed breaking down what needed to be done. This breakdown, in parts, seems more daunting;

- **Produce a grid**
 - Create a 2d array
 - Display the array as a grid
- **Create two players counters, ‘X’ and ‘O’**
 - Toggle between two players
 - Ensure the right counter is placed
- **Allow players to place their counters in a column**
- **Counters drop to the bottom of the column**
- **Create a Blitz button, clearing a column**
 - Allow a choice for players between buttons or choosing a column
 - When button is chosen, clear all counters in that column
- **Create a Time Bomb button, placing an asterisk in place of a counter**
 - Allow this as a choice between buttons or choosing a column
 - Display an asterisk in the coordinates chosen
 - Set a countdown on moves as a timer
 - Clear counters around the asterisk when countdown ends
- **Recognise four in a row**
- **Declare a winner**

The same ranking process followed;

1. **Produce a grid** – The same function from exercise two could be adapted for use in this exercise.
2. **Allow a choice for players between buttons or choosing a column** – This is a case of using an ‘if / if else / else’ statement for players to choose.
3. **Declare a winner** – This should be a case of a simple Boolean. The issue here was tying it in to the recognising four in a row.
4. **Allow players to place their counters in a column** – Displaying a request for a column number and transposing that to the grid is the way to do this.
5. **When Blitz button is chosen, clear all counters in that column** – This seemed daunting however on consideration it will be like the above point.
6. **Display an asterisk in the coordinates chosen** – As above, this can be done by displaying a request for row and column and transposing these coordinates to the grid.
7. **Ensure the right counter is placed** – Using an if statement here should make sure the right counter is placed.

8. **Toggle between two players** – This will be a little more complicated than previous games as each player takes a go consecutively.
9. **Counters drop to the bottom of the column** – Although a little daunting at first, on consideration a for loop should be able to handle this.
10. **Set a countdown on moves as a timer** – This could be done using a variable and incrementing the amount of turns until timer is up.
11. **Recognise four in a row** – This could be done with 'for' loops. It should be more straight forward for horizontal matches but will be more complex for the vertical and diagonal.
12. **Clear counters around the asterisk when countdown ends** – This could be the most complex part, possibly done by using the coordinates of the Asterix and setting all coordinates around it back to original character.

The initial planned out program was a more limited than desired. Figures 37 – 42 show the plans as with previous exercises.

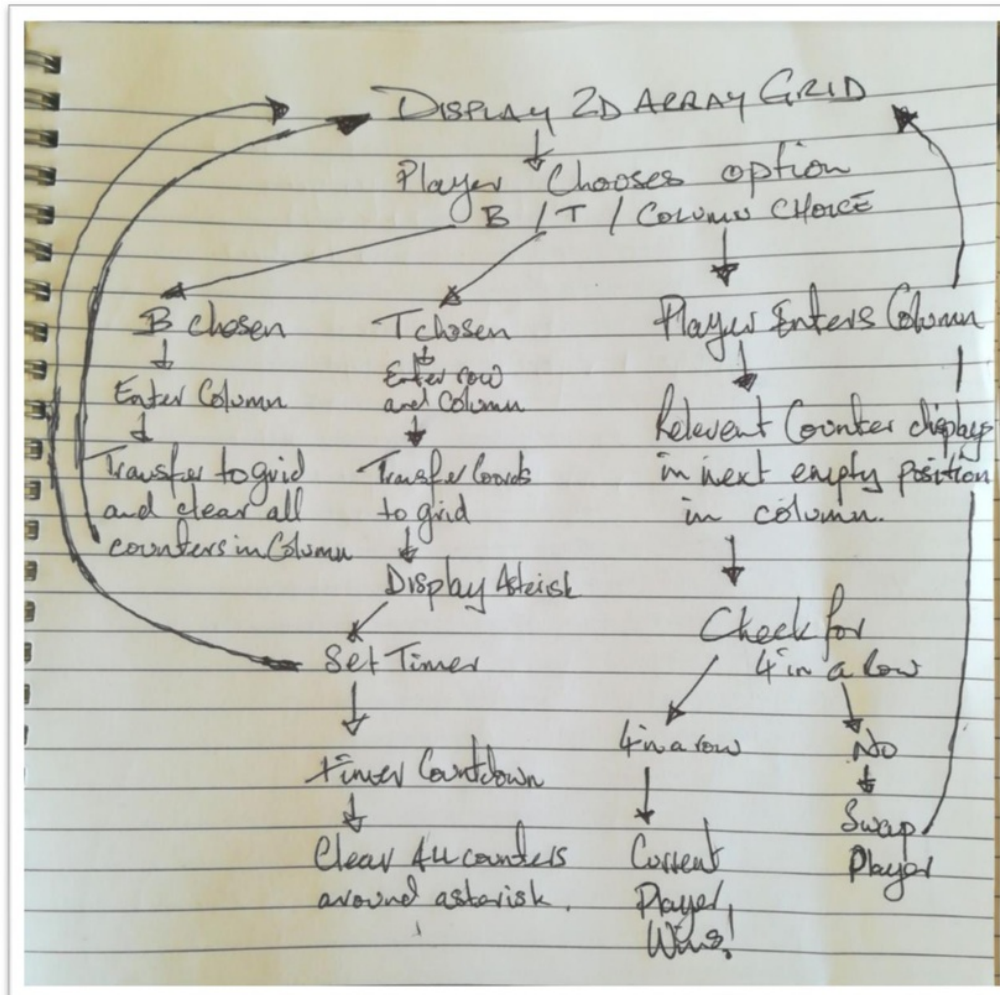


Figure 37 - Exercise 3 Game Flow

```

Use Display Grid Function (change
parameters)
Use Read Number Function
Use Read Numbers Function (change
range)

PlayerOneColumnChoice function
    if variable "column" is less than 1
        for loop starts to 1
            if grid[variable "column"] is '-'
                grid[variable "column"] is 'X'
                variable "column" incremented
                Break
            End if
        End For
    Else
        Display "This Row is Full"
        Call ReadNumbers Function
    End if/else
End Function

Repeat For Player Two
using 'O'

```

Figure 38 - Exercise 3 Program Design

```

Call DisplayGrid
for loop to 31
    Display "Current player choose to
    Blitz (B), Time bomb (T) or any
    other button to choose a column"
    Input to "Button" variable

    If "Button" is 'B'
        BlitzButton function called
    Else If "Button" is 'T'
        TimeBomb function called
        "Timer" variable set to 0
    Else
        Call ReadNumbers function for
        "ColumnChoice" variable

        Switch statement on "ColumnChoice"
        CASE 0
            if "current player" variable is 1
                call PlayerOneColumnChoice func
            End If.
            Else
                call PlayerTwoColumnChoice func
            End Else
            BREAK
        End Case 0

        Repeat to Case 5
    End If
End For loop

```

Figure 39 - Exercise 3 Program Design

```

Increment "Times" variable
If is equal to 4
    Call Time Out function
End If
Call Display Grid function
    Establish 4x4 Array
If "Current Player" variable is 1
    "Current Player" is 2
Else
    "Current Player" is 1
End If
End For loop

```

Figure 40 - Exercise 3 Program Design

```

BlitzBots Function
Display "You can now clear a column"
Call ReadNumbers function for variable "column"
For loop from 1 to 7
    grid [i][column] is '-'
End For
"variable" [column] is 6 (For column choice function)
End Function

```

Figure 41 - Exercise 3 Blitz Function

```

ChangeBombAsterish function
  for loop up to 7 (i)
    for loop up to 6 (j)
      if grid[i][j] is '*'
        grid[i][j] is '-'
        grid[i-1][j] is '-'
        grid[i+1][j] is '-'
        grid[i][j-1] is '-'
        grid[i][j+1] is '-'
        grid[i-1][j-1] is '-'
        grid[i-1][j+1] is '-'
        grid[i+1][j-1] is '-'
        grid[i+1][j+1] is '-'
      End if
    End for
  End for
End Function

TimeBomb function
  Display "Set a timer to clear Squares
           around your spot. Enter coords"
  Call ReadNumbers for variable "i"
  Call ReadNumbers for variable "j"

  grid[i][j] is '*'
End Function

```

Figure 42 - Exercise 3 TimeBomb Functions

3.2 Implementation

Having set out the functions better in the plan, it was clearer how to write the code. There was still a little consideration to be given to the recognition of four in a row, however, it was possible from the designs to write nearly all the code.

It was clear that previous functions could be used, such as 'ReadNumbers' and 'DrawGrid'. This was very useful. It was also that there were a number of patterns that could be followed within the design to aid with loops as well as a number of 'if / else' statements and binary decisions. After some consideration, it was clear how to perform the change in player one to player two using an 'if / else' statement to swap between the two on each turn.

3.3 Testing

Testing was limited for this exercise as some of the game hasn't been completed, however tests have shown most of the functions are working. 'DrawGrid', 'ReadNumber' performed as previously. The column choice function worked well and the toggle between players worked

too. The limit comes at the four in a row functions. Horizontal was the easiest to achieve as a nested 'for' loop worked well for that. Vertical was much more difficult. Both of these at first wouldn't work as the loops continued and the realisation was that it needed a reset on the variable to count up to four. Once this was put in place the vertical and horizontal worked. The only other function that wouldn't work as expected was the diagonal four in a row. This was a much more complex problem that time prevented from completing.

CONCLUSION

The key learnings from this process are as follows:

Logic and process are all important to completing a successful program. It was important to work through the logic of the decisions with each statement, to plan carefully and form the program. This gives a good picture of how the program should work before it is started.

Functions are reusable. It was good to be able to reuse a function for the same purpose in another exercise.

Testing is crucial. It is crucial to remove any bugs from a program by testing all scenarios thoroughly. It was found that an independent person testing shows more issues than the programmer following the perfect route through a program.

REFERENCES

- [1] J. Soulié, "C++ Language Tutorial," June 2007. [Online]. Available: <http://www.cplusplus.com/doc/tutorial/>. [Accessed 27th February 2019].
- [2] U. Kirch-Prinz and P. Prinz, A Complete Guide to Programming in C++, Sudbury, Massachusetts: Jones and Bartlett Publishers, 2002, p. 173.
- [3] M. Dacey, *Week 11 C++ - Functions*, Swansea, 2019.

APPENDIX

Appendix A – Exercise 1 Code.

```
#include<string>
#include<iostream>
#include<sstream>
#include<stdlib.h>

using namespace std;

int ReadNumber(string message) // Reads the input and makes sure its a number entered.
{
    int num;
    string input;
    stringstream stream;

    cout << message << endl;
    getline(cin, input);
    stream.str(input);

    while ((stream >> num && stream.eof()) == false)
    {
        cout << "Please make sure it is a whole number entered" << endl;
        stream.clear();
        getline(cin, input);
        stream.str(input);
    }

    return num;
}

void ReadNumbers(int nums[7]) //Uses ReadNumber and makes sure number choice is in
range.
{
    for (int i = 0; i < 7; i++)
    {
        nums[i] = ReadNumber("Enter a number");
        while (nums[i] < 1 || nums[i] > 10)
        {
            nums[i] = ReadNumber("Number must be in the range of 1 to 10");
        }
    }
}

void DisplayMenu() // Just to display choices before passing ReadNumbers.
{
    cout << "Is the next number 1. higher or 2. lower" << endl;
}

bool Higher(int x, int y) // Returns a boolean as true when the next number is higher.
{
    bool higher;
    if (x > y)
    {
        higher = true;
    }
    else
    {

```

```

        higher = false;
    }
    return higher;
}

bool Lower(int x, int y) // Returns a boolean as true when the next number is lower.
{
    bool lower;
    if (x < y)
    {
        lower = true;
    }
    else
    {
        lower = false;
    }
    return lower;
}

void Winner(bool win) //Declares a winner dependent on a boolean statement.
{
    if (win == true) // COMPLETE ALL SIX - PLAYER TWO WINS
    {
        cout << "***** Player Two wins! *****" << endl;
        cout << "----- GAME OVER -----" << endl;
    }
    else // IF WRONG - PLAYER 1 WINS
    {
        cout << "***** Player One wins! *****" << endl;
        cout << "----- GAME OVER -----" << endl;
    }
}

bool Choice (int choice, int num1, int num2) // compares the player's choice to the
previous booleans.
{
    bool highLow = false;
    switch (choice)
    {
        case 1:
            highLow = Higher(num1, num2);
            break;
        case 2:
            highLow = Lower(num1, num2);
            break;
    }
    return highLow;
}

bool WhoWins(bool higherLower, int x) // Determines whether to play on or end game if
wrong choice.
{
    if (higherLower == true)
    {
        cout << "Correct, the number was: " << x << endl;
        return true;
    }
    else
    {
        cout << "Sorry, wrong choice." << endl;
        return false;
    }
}

```

```

    }
}

int main(int argc, char *argv[])
{
    //PLAYER 1 ENTERS 7 VALUES (1 - 10) - LOOP WITH BOOL
    cout << "----- Welcome to Higher Or Lower -----" << endl << endl;
    cout << "        Player one will enter 7 numbers." << endl;
    cout << "Player two will then guess whether the numbers are higher or lower,"
<< endl;
    cout << "Guess wrong and player one wins. Guess all six right and you win!" <<
endl << endl;

    int num[7];
    ReadNumbers(num);

    system("cls");

    //FIRST NUMBER SHOWN

    cout << "Player Two your first number is: " << num[0] << endl;

    //PLAYER 2 GUESS HIGHER OR LOWER LOOPED 6 TIMES.

    int choice;
    bool higherLower = false;

    for (int i = 0; i < 6; i++)
    {
        DisplayMenu();
        choice = ReadNumber("Please choose option 1 or 2");

        while (choice != 1 && choice != 2)
        {
            choice = ReadNumber("Please choose again, option 1 or 2");
        }

        higherLower = Choice(choice, num[i+1], num[i]);

        if (WhoWins(higherLower, num[i + 1]) == false)
        {
            break;
        }
    }

    Winner(higherLower);

    system("pause");
    return 0;
}

```

Appendix B – Exercise 2

```
#include<string>
#include<iostream>
#include<sstream>
#include<stdlib.h>

using namespace std;

int ReadNumber(string message); // Reads input and determines if it is a number.

void DisplayGrid(char grid[9][9]); // Draws a grid based on 2d array, x is rows
downward, y is columns across.

void PlayerOneChoose(char p1coords[9][9]); // Allows player one to choose coordinates
to place the treasure

int PlayerTwoGuessX(); // Allows player two to guess the x coordinate.

int PlayerTwoGuessY(); // Allows player two to guess the y coordinate.

int IsSame(char p1coords[9][9], char grid[9][9]); // Compares the guessed coordinates
to the hidden grid with treasure

void Winner(int lose); // decides a winner based on all treasure guessed or three
wrong guesses.

int main(int argc, char *argv[])
{
    char coords[9][9] = {
        {'0','1','2','3','4','5','6','7','8'},
        {'1','-','-','-','-','-','-','-','-'},
        {'2','-','-','-','-','-','-','-','-'},
        {'3','-','-','-','-','-','-','-','-'},
        {'4','-','-','-','-','-','-','-','-'},
        {'5','-','-','-','-','-','-','-','-'},
        {'6','-','-','-','-','-','-','-','-'},
        {'7','-','-','-','-','-','-','-','-'},
        {'8','-','-','-','-','-','-','-','-'} }; // 2d array, x being rows
    (down) y being columns (across)

    char pOneGrid[9][9]; // hidden grid with treasure on.
    int lose = 0;

    cout << "----- Welcome to Treasure Hunt! -----" << endl << endl;
    cout << "Player one will pick three positions for treasure" << endl;
    cout << "Player two will guess where the treasure is buried" << endl << endl;

    DisplayGrid(coords);

    PlayerOneChoose(pOneGrid);

    system("cls"); // Clears screen so player two can't see treasure coordinates.'

    lose = IsSame(pOneGrid, coords);

    DisplayGrid(coords);

    Winner(lose);
```

1

2

```

        return 0;
    }

    int ReadNumber(string message)
    {
        int num;
        string input;
        stringstream stream;

        cout << message << endl;
        getline(cin, input);
        stream.str(input);

        while ((stream >> num && stream.eof()) == false)
        {
            cout << "Please make sure it is a whole number entered" << endl;
            stream.clear();
            getline(cin, input);
            stream.str(input);
        }

        return num;
    }

    void DisplayGrid(char grid[9][9])
    {
        for (int i = 0; i < 9; i++)
        {
            for (int j = 0; j < 9; j++)
            {
                cout << grid[i][j] << "    ";
            }
            cout << endl << endl;
        }
    }

    void PlayerOneChoose(char pOneCoords[9][9])
    {
        int x[3];
        int y[3];
        for (int i = 0; i < 3; i++)
        {
            cout << "Player 1 choose position " << i + 1 << " of 3 for your treasure" << endl;
            x[i] = ReadNumber("Enter X co-ordinate as a number between 0 and 8");

            while (x[i] < 1 || x[i] > 8) // Make sure number is in range.
            {
                x[i] = ReadNumber("Number must be in the range of 1 to 8");
            }

            y[i] = ReadNumber("Enter Y co-ordinate as a number between 0 and 8");
            while (y[i] < 1 || y[i] > 8)
            {
                y[i] = ReadNumber("Number must be in the range of 1 to 8");
            }

            while (x[i] == x[i - 1] || x[i] == x[i - 2]) // Make sure coordinates aren't picked twice.
            {

```

```

        if (y[i] == y[i - 1] || y[i] == y[i - 2])
        {
            x[i] = ReadNumber("You can't use same coordinates twice,
pick X again:");
            y[i] = ReadNumber("You can't use same coordinates twice,
pick Y again:");
        }
        else
        {
            break;
        }
    }
    pOneCoords[x[i]][y[i]] = 'T'; // inserts into grid that's compared to
main grid.
    }
}
int PlayerTwoGuessX()
{
    int x;
    cout << "Player 2 guess the position of the treasure: " << endl << " You only
get three lives! Guess carefully!" << endl;
    x = ReadNumber("Enter X co-ordinate as a number between 1 and 8");
    while (x < 1 || x > 8) // Makes sure numbers are in range.
    {
        x = ReadNumber("Number must be in the range of 1 to 8");
    }
    return x;
}
int PlayerTwoGuessY()
{
    int y;
    y = ReadNumber("Enter Y co-ordinate as a number between 1 and 8");
    while (y < 1 || y > 8)
    {
        y = ReadNumber("Number must be in the range of 1 to 8");
    }
    return y;
}

int IsSame(char pOneCoords[9][9], char grid[9][9])
{
    int win = 0;
    int wrongGuess = 0;
    int xGuess;
    int yGuess;

    while (win != 3) // Makes sure the game can continue if all the treasure hasn't
been found.
    {
        DisplayGrid(grid);

        xGuess = PlayerTwoGuessX();
        yGuess = PlayerTwoGuessY();

        while (grid[xGuess][yGuess] == 'X' || grid[xGuess][yGuess] == 'N') //
Makes sure guesses can't be made twice.
        {
            cout << "You have already chosen that one, try again:" << endl;
            xGuess = PlayerTwoGuessX();
            yGuess = PlayerTwoGuessY();
        }
    }
}

```

```

        if (pOneCoords[xGuess][yGuess] == 'T') // Compares player two guesses to
treasure grid.
        {
            cout << "Good Guess, correct!" << endl;

            grid[xGuess][yGuess] = 'X';

            win++; // counts off one treasure.
        }
        else // checks off three lives for player two on wrong guesses.
        {
            cout << "Sorry, wrong guess." << endl << "Life " << wrongGuess +
1 << " from the three used up" << endl;

            grid[xGuess][yGuess] = 'N';

            wrongGuess++;
        }

        if (wrongGuess == 3) // Breaks out if three lives are used.
        {
            break;
        }
    }

    return wrongGuess;
}

void Winner(int lose)
{
    if (lose == 3)
    {
        cout << endl << endl << "****PLAYER 1 WINS!****" << endl;
        cout << "----- GAME OVER -----" << endl;
    }
    else
    {
        cout << endl << endl << "****PLAYER 2 WINS!****" << endl;
        cout << "----- GAME OVER -----" << endl;
    }
}

```

Appendix C – Exercise 3

```
#include<string>
#include<iostream>
#include<sstream>
#include<stdlib.h>

using namespace std;

void DisplayGrid(char grid[7][6])
{
    // To produce a grid from the 2d array created. X is rows and Y is columns.
    for (int i = 0; i < 7; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            cout << grid[i][j] << "    ";
        }
        cout << endl << endl;
    }
}

// Function to read numbers and make sure they are numbers entered.
int ReadNumber(string message)
{
    int num;
    string input;
    stringstream stream;

    cout << message << endl;
    getline(cin, input);
    stream.str(input);

    while ((stream >> num && stream.eof()) == false)
    {
        cout << "Please make sure it is a number entered:" << endl;
        stream.clear();
        getline(cin, input);
        stream.str(input);
    }
    return num;
}

//Using the ReadNumber function and check the choice is in range.
int ReadNumbers()
{
    int num;

    num = ReadNumber("Please choose your column, 0 to 5:");
    while (num < 0 || num > 5)
    {
        num = ReadNumber("Number must be in the range of 0 to 5");
    }
    return num;
}

// Player 1 column choice to enable the X counter to be dropped into place.
void PlayerOneColumnChoice(int col, char grid[7][6], int i[6])
{
    if (i[col] >= 1)
    {
        for (int x = 6; x > 0; x--)
        {
            if (grid[x][col] == '-')

```

```

        {
            grid[x][col] = 'X';
            i[col]++;
            break;
        }
    }
}
else
{
    cout << "This row is full..." << endl;
    col = ReadNumbers();
}
}
// Player 2 column choice to enable the 0 counter to be dropped into place.
void PlayerTwoColumnChoice(int col, char grid[7][6], int i[6])
{
    if (i[col] >= 1)
    {
        for (int x = 6; x > 0; x--)
        {
            if (grid[x][col] == '-')
            {
                grid[x][col] = 'O';
                i[col]++;
                break;
            }
        }
    }
    else
    {
        cout << "This row is full..." << endl;
        col = ReadNumbers();
    }
}

bool CheckHorizontal(char grid[7][6])
{
    bool same;
    int playerOne = 0;
    int playerTwo = 0;

    for (int i = 0; i < 7; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            if (grid[i][j] == grid[i][j + 1])
            {
                if (grid[i][j] == 'X')
                {
                    playerOne++;
                    playerTwo = 0;
                }
                else if (grid[i][j] == 'O')
                {
                    playerTwo++;
                    playerOne = 0;
                }
                else
                {
                    playerOne = 0;
                    playerTwo = 0;
                }
            }
        }
    }
}

```

```

        }
        if (playerOne == 3 || playerTwo == 3)
        {
            same = true;
            break;
        }
        else
        {
            same = false;
        }
    }
}
return same;
}

bool CheckVertical(char grid[7][6], int col)
{
    bool same;
    int playerOne = 0;
    int playerTwo = 0;

    for (int i = 0; i < 6; i++)
    {
        if (grid[i][col] == grid[i + 1][col])
        {
            if (grid[i][col] == 'X')
            {
                playerOne++;
            }
            else if (grid[i][col] == 'O')
            {
                playerTwo++;
            }
        }
        if (playerOne == 3 || playerTwo == 3)
        {
            same = true;
            break;
        }
        else
        {
            same = false;
        }
    }
    return same;
}

bool IsDiagonalDown(char grid[7][6])
{
    int playerOneStart = 1;
    int playerTwoStart = 1;
    bool connectFour = false;

    for (int i = 1; i < 4; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (grid[i][j] == 'X')
            {
                for (int x = 0; x < 4; x++)
                {
                    if (grid[i][j] == grid[i + x][i + x])

```



```

        {
            playerOneStart++;
        }
    }
}
else if (grid[i][j] == 'O')
{
    for (int z = 0; z < 4; z++)
    {
        if (grid[i][j] == grid[i + z][i + z])
        {
            playerTwoStart++;
        }
    }
}
}
}
if (playerOneStart == 4 || playerTwoStart == 4)
{
    connectFour = true;
}
return connectFour;
}

bool IsDiagonalUp(char grid[7][6])
{
    for (int i = 0; i < 7; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            if (grid[i][j] == 'X' || grid[i][j] == 'O')
            {
                for (int x = 1; x < 4; x++)
                {
                    if (grid[i][j] != grid[i - x][j + x])
                    {
                        return false;
                        break;
                    }
                    else
                    {
                        return true;
                    }
                }
            }
        }
    }
}

bool CheckDiagonal(char grid[7][6])
{
    bool fourUp = IsDiagonalUp(grid);
    bool fourDown = IsDiagonalDown(grid);

    if (fourUp == true || fourDown == true)
    {
        return true;
    }
    else
    {

```

```

        return false;
    }
}

void BlitzButton( char grid[7][6], int choice[6])
{
    cout << "You can now clear a whole column." << endl;
    int column = ReadNumbers();

    for (int i = 1; i < 7; i++)
    {
        grid[i][column] = '-';
    }
    choice[column] = 6;
}

// Function to feed into TimeBomb function.
int ReadRow()
{
    int num;

    num = ReadNumber("Please choose your row between 1 and 6:");
    while (num < 1 || num > 6)
    {
        num = ReadNumber("Number must be in the range of 1 to 6");
    }
    return num;
}

void TimeBomb(char grid[7][6])
{
    int i;
    int j;

    cout << "You can now set a timer to clear a square and ALL adjacent squares
around it." << endl;
    i = ReadRow();
    j = ReadNumbers();

    grid[i][j] = '*';
}

// Turns all squares around the asterisk back to clear.
void TimeUp(char grid[7][6])
{
    for (int i = 0; i < 7; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            if (grid[i][j] == '*')
            {
                grid[i][j] = '-';
                grid[i - 1][j] = '-';
                grid[i + 1][j] = '-';
                grid[i][j - 1] = '-';
                grid[i][j + 1] = '-';
                grid[i - 1][j - 1] = '-';
                grid[i - 1][j + 1] = '-';
                grid[i + 1][j + 1] = '-';
                grid[i + 1][j - 1] = '-';
            }
        }
    }
}

```

```

}

int main(int argc, char *argv[])
{
    string input;
    stringstream stream;

    char coords[7][6] = {
        {'0','1','2','3','4','5'},
        {'-','-','-','-','-','-'},
        {'-','-','-','-','-','-'},
        {'-','-','-','-','-','-'},
        {'-','-','-','-','-','-'},
        {'-','-','-','-','-','-'},
        {'-','-','-','-','-','-'} };
    // 2d array for grid.

    int columnChoice;
    int stack[6] = { 6,6,6,6,6,6 };
    int currentPlayer = 1;
    int winner;
    bool horizontalEqual = false;
    bool verticalEqual = false;
    bool diagonalEqual = false;
    char button;
    int timer = 0;

    cout << "----- Welcome to Four In A Row - With a Difference -----"
    -" << endl << endl;
    cout << "          This is a two player game with bonus buttons to help you." <<
    endl << endl;

    DisplayGrid(coords); // draws the grid from the above array. x is rows (down) y
    is columns (across)

    for (int x=0; x<31; x++) // to allow for all squares to be filled if needs be.
    {
        cout << "Player " << currentPlayer << " choose to blitz (b), time bomb
        (t) or any other button to choose a column followed by enter:" << endl;
        cin >> button; // allow a choice of buttons or column choice.
        if (button == 'b')
        {
            BlitzButton(coords, stack);
        }
        else if (button == 't')
        {
            TimeBomb(coords);
            timer = 0;
        }
        else
        {
            columnChoice = ReadNumbers();

            switch (columnChoice) // switch cases for each column and current
            player.
            {
            case 0:
                if (currentPlayer == 1)
                {
                    PlayerOneColumnChoice(columnChoice, coords, stack);
                }
                else

```

```

        {
            PlayerTwoColumnChoice(columnChoice, coords, stack);
        }
        break;
    case 1:
        if (currentPlayer == 1)
        {
            PlayerOneColumnChoice(columnChoice, coords, stack);
        }
        else
        {
            PlayerTwoColumnChoice(columnChoice, coords, stack);
        }
        break;
    case 2:
        if (currentPlayer == 1)
        {
            PlayerOneColumnChoice(columnChoice, coords, stack);
        }
        else
        {
            PlayerTwoColumnChoice(columnChoice, coords, stack);
        }
        break;
    case 3:
        if (currentPlayer == 1)
        {
            PlayerOneColumnChoice(columnChoice, coords, stack);
        }
        else
        {
            PlayerTwoColumnChoice(columnChoice, coords, stack);
        }
        break;
    case 4:
        if (currentPlayer == 1)
        {
            PlayerOneColumnChoice(columnChoice, coords, stack);
        }
        else
        {
            PlayerTwoColumnChoice(columnChoice, coords, stack);
        }
        break;
    case 5:
        if (currentPlayer == 1)
        {
            PlayerOneColumnChoice(columnChoice, coords, stack);
        }
        else
        {
            PlayerTwoColumnChoice(columnChoice, coords, stack);
        }
        break;
    }
}

timer++; // for TimeBomb function.

horizontalEqual = CheckHorizontal(coords); // ok, will recognise
horizontal four in a row.

```

```

        verticalEqual = CheckVertical(coords, columnChoice); // ok, will
recognise vertical four in a row.
        diagonalEqual = IsDiagonalDown(coords); // not working yet!

        if (timer == 4)
        {
            TimeUp(coords);
        }

        DisplayGrid(coords);

        if (verticalEqual == true || horizontalEqual == true || diagonalEqual ==
true)
        {
            cout << "***** Player " << currentPlayer << " Wins! *****"
<< endl;
            cout << "***** Game Over *****" << endl << "-----
Thank you for playing! -----" << endl;
            break;
        }

        if (currentPlayer == 1) // player toggle to allow alternative turns.
        {
            currentPlayer = 2;
        }
        else
        {
            currentPlayer = 1;
        }
    }

    return 0;
}

```

FINAL GRADE

92/100

GENERAL COMMENTS

Instructor

Exercise 1 (50) 46

Exercise 2 (30) 28

Exercise 3 (20) 18

Good evidence of problem solving and testing.

Exercise 1 is well subdivided and is very clean code. In more "real-world" situations you would not produce quite so many simple functions but we are learning and this is good evidence of the correct approach being applied. Well done.

Exercise 2 is also very good.

Exercise 3 is very clean.

Excellent work.

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39



Comment 1

The main() function here is telling a nice story



Comment 2

CheckForWinner would be a better name

PAGE 40

PAGE 41



Comment 3

These two functions should be one more generic solution

PAGE 42



Comment 4

Move this up to above while loop

```
while(win != 3 && wrongGuess < 3)
```

PAGE 43

PAGE 44



Comment 5

These two functions are similar and should be replaced by a generic solution

PAGE 45



Comment 6

There looks like some code duplication that could be removed

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50
