

Assignment 1

SOFTWARE DEVELOPMENT

KIERAN PERRIN – PER09098703

Contents

Introduction.....	2
Exercise 1.....	2
Exercise 2.....	2
Exercise 3.....	3
Appendix	3
Exercise 1.....	3
Exercise 2.....	6
Exercise 3.....	9

Introduction

In this document I will record the process used when developing my code by listing the problems that needed to be solved, evidencing the problems being solved and describing the tests used to make sure the code gives us the required output.

Exercise 1

The first exercise was a simple game of higher or lower. The problems that are needed to be solved regarding this program include:

1. Asking the user to enter a number from 1 to 9.
2. Making sure that what the user inputs is a number and is from 1 to 9.
3. Repeating that 7 times.
4. Showing the second user the first number.
5. Asking the second user to guess if the next number is higher or lower with "h" or "l".
6. Checking that what the second user has entered is either a h or l.
7. Check whether the next number is higher or lower, if the second user guessed correctly, move on to the next number. If the guess was incorrect end the game.
8. Repeat 7 times.
9. If the second user guessed right all 7 times, they win the game and the game ends.

To test that this program functions correctly I would first run the program and enter numbers that the programs asks for and enter higher or lower correctly to see if the game can be completed.

The next test would be to enter higher or lower incorrectly to see if player 2 can lose the game.

After that I would test to see if I entered a number below 0 or above 9 or not a number at all. This is to test that the program will ask the user to re enter a number that is within the parameters.

The last tests are to check if player 2 enters anything other than a h or l, it will ask the user to enter the right thing.

*evidence
of tests?*

Exercise 2

The second exercise was a two-player treasure hunt game where the first player sets treasure locations in a 7 by 7 grid. The second player then gets 3 chances to guess the treasure location. The problems that I had to solve to make this program include:

1. Set up a grid for the game board.
2. Ask player 1 for the x and y co-ordinates of the first treasure.
3. Make sure that the co-ordinates given are within the grid boundaries and are a number.
4. Repeat that 3 times.
5. Clear the screen so the second player doesn't know the treasure locations.
6. Ask the second player for his x and y co-ordinate guess.

7. Check to make sure that the guesses are a number and is within the grid boundaries.
8. Check to see whether the guess matches up with any of the co-ordinates from player 1.
9. If the co-ordinates match place an x on the board in that location. If it's not a match take 1 life away from player 2
10. If player 2 finds all the treasure locations, they win the game
11. If player two gets to 0 lives, it's game over.

To test that this program is functioning properly I would first test that the game board prints on the screen correctly. I would then test the program while entering in correct numbers to see if the game runs through and can be completed. The next test would be to enter wrong co-ordinates as player one to see whether the player is asked to re enter co-ordinates within the parameters. After that I should test to make sure if player 2 enters a wrong number or not number at all, they are asked to re-enter.

Order of testing?

Exercise 3

The last exercise was another two-player game of connect 4 but with additional functionality in the form of power ups that clear a column and a time bomb that clears the counters surrounding it in all directions after two turns. The problems that needed to be solved to make this work are:

1. A grid needs to be laid out for the game board
2. Player 1 chooses a column number
3. A piece will be placed in the lowest free slot on the grid in the column that the player chose.
4. If the column is full then the player will be asked to choose another column
5. If the player enters anything but a valid column number, they will be asked to choose again.
6. Player 2 will go through the same process.
7. The game will continue until a player gets four pieces in a row vertically, horizontally or diagonally. If the grid is full and no one has got 4 in a row it will be a draw.
8. The player that got 4 in a row will win the game.

I didn't manage to get the functionality for diagonal connect 4 and the power ups in the game.

My first test for this program is to make sure the game board prints to the screen. My next test is to check when players choose a column their piece is placed in that column on the most bottom available slot. I would then test to make sure that if a player has put a piece in a slot it isn't overwritten by the other player. I would then test to make sure if a player enters anything other than a viable column number, the program will ask them to choose again. After that I would then test to see if a column is full the program tells the user and asks them for a different column. The last test would to make sure that the program can recognise when pieces are 4 in a row vertically or horizontally and if so the player whose pieces are, win the game.

Order of testing?

Appendix

Exercise 1

using System;

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace exercise1
{
```

```
    class Program
    {
```

```
        static void Main(string[] args)
        {
```

```
            int[] player1 = { 0, 0, 0, 0, 0, 0, 0 };
            bool? higher = null;
            bool? lower = null;
            bool? answer = null;
            string playerAnswer = "";
```

```
            Console.WriteLine("Player 1 please enter 7 values:");
            Console.WriteLine();
            for (int i = 0; i < player1.Length; i++)
            {
```

```
                Console.Write("Enter number " + (i + 1) + " > "); //Asks player 1 to
                //enter a number (i+1) is because the array counts from 0
```

```
                bool success = int.TryParse(Console.ReadLine(), out player1[i]);
                //takes number entered by player 1 and put it into the array and converts to integer
                if (success == false) //If
                the user entered something other than in integer will show error message
```

```
                {
                    Console.WriteLine("Please enter a number between 1 and 10");
                    i--;
                }
                //to get the program to ask the user to re enter the number, minus 1 from i
```

```
                if (success == true && (player1[i] < 1 || player1[i] > 10))
                //If the user enters a number smaller than 1 and larger than 10 will show error
                message
```

```
                {
                    Console.WriteLine("Please enter a number between 1 and 10");
                    i--;
                }
```

```
            }
```

```
            Console.WriteLine();
            Console.Clear(); //so player 2 doesn't see the numbers entered by player 1
```

```
            Console.WriteLine("Player 2 the first value is " + player1[0]);
```

```

for (int i = 0; i < player1.Length; i++)
{
    if (i == 6) //win condition
    {
        Console.WriteLine("Player 2 wins");
        Console.ReadLine(); //pause
        Environment.Exit(1); // if the user press enter exit program
    }
    Console.WriteLine("Higher (H) or lower (L) > ");
    playerAnswer = Console.ReadLine().ToLower(); //player 2 input is
converted to lowercase so validation is easier
}

if (playerAnswer == "h" || playerAnswer == "l") //If player 2 input is
h or l, rest of the program will work
{
    if (playerAnswer == "h")
    {
        answer = true;
    }
    if (playerAnswer == "l")
    {
        answer = false;
    }

    if (player1[i] > player1[i + 1])
    {
        higher = false;
        lower = true;
    }
    else
    {
        higher = true;
        lower = false;
    }

    if (higher == true && answer == true)
    {
        Console.WriteLine("Correct, the next value is " + player1[i +
1]);
    }

    if (lower == true && answer == false)
    {
        Console.WriteLine("Correct, the next value is " + player1[i +
1]);
    }

    if (higher == true && answer == false)
    {
        Console.WriteLine("Incorrect, sorry Player 1 wins");
        Console.ReadLine();
        Environment.Exit(1);
    }

    if (lower == true && answer == true)

```

What if not H or L?
 Over-
 Complex use of
 variables. Logic
 could be much
 simpler

```

        {
            Console.WriteLine("Incorrect, sorry Player 1 wins");
            Console.ReadLine();
            Environment.Exit(1);
        }
    }

    else //if player 2 entered anything other than h or l, show error
message
    {
        Console.WriteLine("Please enter h for higher or l for lower");
        i--;
    }
}

Console.ReadLine(); //pause
}
}

```

Exercise 2

```

using System;

namespace TreasureHunt
{
    class Program
    {
        static string[,] p1Grid = new string[8, 8] {
            {" 0 ", " ", " 1 ", " ", " 2 ", " ", " 3 ", " ", " 4 ", " ", " 5 ", " ", " 6 "},
            {" 7 ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "},
            {" 1 ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "},
            {" 2 ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "},
            {" 3 ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "},
            {" 4 ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "},
            {" 5 ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "},
            {" 6 ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "},
            {" 7 ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "},
        };

        static void Main(string[] args)
    }
}

```

```

{
    Title();
    GameBoard(); //Uses the method gameboard to write the gameboard to the
screen

    int[] treasureX = new int[3];
    int[] treasureY = new int[3];
    int p2Lives = 3;
    int p2GuessX;
    int p2GuessY;
    int p2Win = 0;
    bool validate = false;

    for (int i = 1; i <= 3; i++) //loop runs 3 times so it asks p1 to enter
three locations
    {
        Console.WriteLine("Player 1 place treasure {0} now:", i);
        Console.Write("X: ");
        validate = int.TryParse(Console.ReadLine(), out treasureX[i - 1]);

        while (!validate | treasureX[i - 1] > 7) //loop runs if the try parse
isn't completed or the user enters a number above 7
        {
            Console.WriteLine("Please enter a valid X co-ordinate");
            Console.Write("X: ");
            validate = int.TryParse(Console.ReadLine(), out treasureX[i - 1]);
        }

        Console.Write("Y: ");
        validate = int.TryParse(Console.ReadLine(), out treasureY[i - 1]);

        while (!validate | treasureY[i - 1] > 7)
        {
            Console.WriteLine("Please enter a valid Y co-ordinate");
            Console.Write("Y: ");
            validate = int.TryParse(Console.ReadLine(), out treasureY[i - 1]);
        }

        Console.WriteLine("Treasure {0} placed", i);
        //p1Grid[treasureY[i-1], treasureX[i-1]] = "  X  "; used this line
of code to check if p1 co-ordinates are inputted correctly, it draws an x in the co-
ordinates

    }

    Console.Clear();

    while (p2Lives > 0 & p2Win != 3)
    {
        GameBoard();
        Console.WriteLine("Player 2 please guess the treasure location");
        Console.Write("X: ");
        validate = int.TryParse(Console.ReadLine(), out p2GuessX);

        while (!validate | p2GuessX > 7)
        {
            Console.WriteLine("Please enter a valid X co-ordinate");
            Console.Write("X: ");
            validate = int.TryParse(Console.ReadLine(), out p2GuessX);
        }
    }
}

```

Treasure Hunt

```

        foreach (string line in title)
            Console.WriteLine(line);
    }

    public static void GameBoard() //method that draws the game board using the
array
    {
        Console.WriteLine("");
        for (int i = 0; i < 8; i++)
        {
            Console.WriteLine("{0} | {1} | {2} | {3} | {4} | {5} | {6} | {7} |",
p1Grid[i, 0], p1Grid[i, 1], p1Grid[i, 2], p1Grid[i, 3], p1Grid[i, 4], p1Grid[i, 5],
p1Grid[i, 6], p1Grid[i, 7]);
            Console.WriteLine("");
        }
    }
}
}
}

```

Exercise 3

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Connect4
{
    class Program
    {
        public static string player1 = " X ";
        public static string player2 = " O ";
        public static bool winner = false;

        static string[,] gameBoard =
        {
            { " 0 ", " 1 ", " 2 ", " 3 ", " 4 ", " 5 " },
            { " - ", " - ", " - ", " - ", " - ", " - " },
            { " - ", " - ", " - ", " - ", " - ", " - " },
            { " - ", " - ", " - ", " - ", " - ", " - " },
            { " - ", " - ", " - ", " - ", " - ", " - " },
            { " - ", " - ", " - ", " - ", " - ", " - " },
            { " - ", " - ", " - ", " - ", " - ", " - " },
        };

        static void Main(string[] args)
        {
            int rowNum;
            int colNum = 6;

```

```

Title();

Console.WriteLine("");

GameBoard();

while (!winner)
{
    bool validate = false;

    Console.Write("Player 1 select row > ");
    validate = int.TryParse(Console.ReadLine(), out rowNum);

    while (!validate | rowNum > 6)
    {
        Console.WriteLine("Please enter a valid row number");
        Console.Write("Player 1 select row > ");
        validate = int.TryParse(Console.ReadLine(), out rowNum);
    }

    while (colNum == 1 & gameBoard[colNum, rowNum] == " X " | colNum ==
1 & gameBoard[colNum, rowNum] == " O ")
    {
        Console.WriteLine("This column is full, please choose another.");
        Console.Write("Player 1 select row > ");
        validate = int.TryParse(Console.ReadLine(), out rowNum);
    }

    colNum = 6;
    for (int i = 0; i < gameBoard[i, 0].Length; i++)
    {
        if (gameBoard[colNum, rowNum] == " X " | gameBoard[colNum,
rowNum] == " O ")
        {
            colNum--;
        }
    }

    gameBoard[colNum, rowNum] = player1;
    Checkwinner();
    GameBoard();
    if (winner == true)
    {
        Console.WriteLine("Player 1 has won");
        break;
    }

    Console.Write("Player 2 select row > ");
    validate = int.TryParse(Console.ReadLine(), out rowNum);

    while (!validate | rowNum > 6)
    {
        Console.WriteLine("Please enter a valid row number");
        Console.Write("Player 2 select row > ");
        validate = int.TryParse(Console.ReadLine(), out rowNum);
    }
}

```

Connect

```
        Console.WriteLine(line);
    }

    public static void Checkwinner()
    {
        for (int i = 6; i >= 1; --i)
        {
            for (int ix = 0; ix < 3; ix++)
            {
                if (gameBoard[i, ix] == player1 &&
                    gameBoard[i, ix + 1] == player1 &&
                    gameBoard[i, ix + 2] == player1 &&
                    gameBoard[i, ix + 3] == player1)
                {
                    winner = true;
                }

                if (gameBoard[i, ix] == player1 &&
                    gameBoard[i - 1, ix] == player1 &&
                    gameBoard[i - 2, ix] == player1 &&
                    gameBoard[i - 3, ix] == player1)
                {
                    winner = true;
                }

                if (gameBoard[i, ix] == player1 &&
                    gameBoard[i - 1, ix + 1] == player1 &&
                    gameBoard[i - 2, ix + 2] == player1 &&
                    gameBoard[i - 3, ix + 3] == player1)
                {
                    winner = true;
                }
            }

            for (int j = 1; j < 4; j++)
            {
                if (gameBoard[j, ix] == player1 &&
                    gameBoard[j + 1, ix - 1] == player1 &&
                    gameBoard[j + 2, ix - 2] == player1 &&
                    gameBoard[j + 3, ix - 3] == player1)
                {
                    winner = true;
                }
            }
        }
    }
}
}
}
```

