

SOFTWARE DEVELOPMENT

Bradley Martin

MAR16829326

02/04/2019



Contents

Exercise One (Higher or Lower).....	2
Methods	2
Player 1.....	3
Development.....	3
Testing.....	3
Player 2.....	4
Development.....	4
Testing.....	5
Problems.....	6
Exercise Two (Treasure Hunt)	7
Methods	7
Check Life	7
Player 1.....	8
Development.....	8
Testing.....	9
Grid.....	10
Development.....	10
Testing.....	10
Player 2.....	11
Development.....	11
Grid Updating	13
Testing.....	13
Problems.....	15
Appendix	16
Program 1	16
Program 2	18
Program 3	23

No evidence of design logic
Prior to coding.

Exercise One (Higher or Lower)

Methods

```
namespace Assignment_Exercise_1
{
    class Program
    {
        static void Main(string[] args)
        {
        }

        static void Player1()
        {
        }

        static void Player2()
        {
        }
    }
}
```

I began to create the first program by creating the necessary methods that I would need to keep my code organised. I decided to add two methods, one for player 1 input and the second method for player 2 guessing higher or lower.

```
static void Main(string[] args)
{
    int[] num = new int[7]; // array for player 1 numbers
    int numberP2 = num[0]; // beginning value for player 2
}

static void Player1(int[] num)
{
}

static void Player2(int[] num, int numberP2)
{
}
```

I then created two variables which would be placed in the Main method. The first variable will be used for storing player 1's input and comparing them to player 2's guesses later on. The second variable will store the current number that player 2 has to guess with.

Player 1

Development

```
static void Main(string[] args)
{
    int[] num = new int[7]; // array for player 1 numbers
    Player1(num);

    int numberP2 = num[0]; // beginning value for player 2
}

static void Player1(int[] num)
{
    Console.WriteLine("Player 1 please enter 7 values:");
    for (int i = 0; i < num.Length; i++) // for loop to ask the user 7 times
    {
        int number = i + 1;

        int validationloop = 0;
        while (validationloop == 0) // validation
        {
            Console.Write("Enter number " + number + " > "); // asks user for inputs
            num[i] = int.Parse(Console.ReadLine());

            if (num[i] > 10 || num[i] < 1) // checking if the number is above 10 or below 1
            {
                Console.WriteLine("Number must be between 1 and 10"); // rejecting false input from user
            }
            else
            {
                validationloop++; // close validation loop
            }
        }
    }
    Console.WriteLine();
}
```

*try Parse
not
used.*

This is the player 1 method that I created. It starts by telling the player to enter 7 values. The player will be able to enter the values because of the loop below which asks the user to enter a number and also validates the input. If the input that the user enters isn't valid then the 'validationloop' will loop back around asking the user to enter a valid number instead.

Testing

```
Player 1 please enter 7 values:
Enter number 1 > 0
Number must be between 1 and 10
Enter number 1 > 11
Number must be between 1 and 10
Enter number 1 > 1
Enter number 2 > 10
Enter number 3 > 0
Number must be between 1 and 10
Enter number 3 > 11
Number must be between 1 and 10
```

As you can see from the testing above the program works as planned. It will not accept any input below 1 or above 10.

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

Player 2 the first value is 1
```

As you can see from the testing above the program also moves on after 7 valid inputs are received and will not go any further.

Player 2

Development

```
static void Player2(int[] num, int numberP2)
{
    Console.WriteLine("Player 2 the first value is " + numberP2); // tells player 2 beginning value
    for (int i = 1; i < num.Length; i++) // loop for player2 inputs
    {
        int validationloop = 0;
        while (validationloop == 0) // validation
        {
        }
    }
    Console.WriteLine("You managed to beat Player 1! Well Done!");
    Console.ReadLine(); // pause
}
```

This is the starting point for the player 2 method. I decided to have a for loop with a while validation loop inside it checking the users input. I plan to make it so that the only way out of the loop is for the user to guess all of them correct, if he does then the program will leave the loop displaying the win message.

```
static void Player2(int[] num, int numberP2)
{
    Console.WriteLine("Player 2 the first value is " + numberP2); // tells player 2 beginning value
    for (int i = 1; i < num.Length; i++) // loop for player2 inputs
    {
        int validationloop = 0;
        while (validationloop == 0) // validation
        {
            Console.WriteLine("Is your current value Higher (H) or lower (L) than the next value? ");
            string highLow = Console.ReadLine(); // player 2 input

            if (highLow == "H" || highLow == "L")
            {
                validationloop++; // close loop
                int numberP2next = i;
                if (highLow == "H") // checks if H was the input
                {
                    if (numberP2 > num[numberP2next]) // correct guess
                    {
                        Console.WriteLine("Correct, next value is " + num[numberP2next]); // next value displayed
                        numberP2 = num[numberP2next]; // update value
                    }
                    else // incorrect guess
                    {
                        Console.WriteLine("Incorrect, sorry Player 1 wins."); // tells player 2 they are incorrect
                        Console.ReadLine(); // pause
                        Environment.Exit(1); // closes program
                    }
                }
                if (highLow == "L") // checks if L was the input
                {
                    if (numberP2 < num[numberP2next]) // correct guess
                    {
                        Console.WriteLine("Correct, next value is " + num[numberP2next]); // next value displayed
                        numberP2 = num[numberP2next]; // update value
                    }
                    else // if number isn't higher like player 1 guessed
                    {
                        Console.WriteLine("Incorrect, sorry Player 1 wins."); // tells player 2 they are incorrect
                        Console.ReadLine(); // pause
                        Environment.Exit(1); // closes program
                    }
                }
            }
        }
    }
}
```

```
    }
    else
    {
        Console.WriteLine("Please enter either H or L");
    }
}
Console.WriteLine("You managed to beat Player 1! Well Done!");
Console.ReadLine(); // pause
}
```

Case
sensitive
H h
L l?

This is what I created for my player 2 method. It's similar to the first method where it asks the user for an input and validates it before continuing. The only valid inputs are 'H' and 'L' and it will loop back around if the user doesn't enter one of them. If the input was valid then it will check if it was either 'H' or 'L', if the input is H then it will check if the users current number is higher than the next number and if it is then the current number will be replaced. If the guess was incorrect then a message will be displayed saying that player 1 won. This also applies if the input is L. If the user guesses them all correct then the program will leave the loop and display a message saying that player 2 won.

Testing

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

Player 2 the first value is 1
Is your current value Higher (H) or Lower (L) than the next value? > t
Please enter either H or L
Is your current value Higher (H) or Lower (L) than the next value? > 5
Please enter either H or L
Is your current value Higher (H) or Lower (L) than the next value? > ;
Please enter either H or L
Is your current value Higher (H) or Lower (L) than the next value? > H
Incorrect, sorry Player 1 wins.
```

This is the first set of testing that I did to make sure the game was working as intended. As you can see it doesn't allow invalid inputs and if player 2's guess was wrong then it will display a message saying that they have lost.

```
Player 2 the first value is 9
Is your current value Higher (H) or Lower (L) than the next value? > L
Incorrect, sorry Player 1 wins.
```

As you can see the message will also be displayed if the user is incorrect with L as their guess.

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

Player 2 the first value is 1
Is your current value Higher (H) or Lower (L) than the next value? > L
Correct, next value is 2
Is your current value Higher (H) or Lower (L) than the next value? > H
Correct, next value is 1
Is your current value Higher (H) or Lower (L) than the next value? > L
Correct, next value is 2
Is your current value Higher (H) or Lower (L) than the next value? > H
Correct, next value is 1
Is your current value Higher (H) or Lower (L) than the next value? > L
Correct, next value is 2
Is your current value Higher (H) or Lower (L) than the next value? > H
Correct, next value is 1
You managed to beat Player 1! Well Done!
```

This is the testing to see what happens if player 2 guessed correct. As you can see from the testing above the current value gets updated and tells player 2 what their current value is. As you can see from the testing it allows both valid inputs and will continue as intended if the guess is correct.

Problems

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

Player 2 the first value is 2
Is your current value Higher (H) or Lower (L) than the next value? > H
Incorrect, sorry Player 1 wins.
```

One problem I have found when testing the program is that player1 is able to enter the same number twice in a row. This is unfair to player2 as it's impossible to be either higher or lower. The solution I've decided to do for this is add more validation for player1 which stops them from inputting the same number twice in a row.

```
static void Player1(int[] num)
{
    Console.WriteLine("Player 1 please enter 7 values:");
    for (int i = 0; i < num.Length; i++) // for loop to ask the user 7 times
    {
        int number = i + 1;

        int validationloop = 0;
        while (validationloop == 0) // validation
        {
            Console.Write("Enter number " + number + " > "); // asks user for inputs
            num[i] = int.Parse(Console.ReadLine());

            if (num[i] > 10 || num[i] < 1) // checking if the number is above 10 or below 1
            {
                Console.WriteLine("number must be between 1 and 10"); // rejecting false input from user
            }
            else
            {
                if (i > 0) // check if its the first input
                {
                    if (num[i] == num[i - 1]) // check if same as last input
                    {
                        Console.WriteLine("you cannot enter the same number twice in a row!");
                    }
                    else
                    {
                        validationloop++; // close validation loop
                    }
                }
                else
                {
                    validationloop++; // close validation loop
                }
            }
        }
    }
    Console.WriteLine();
}
```

I added this to the player1 method which validates the latest input and prevents it from being a repeat of the last.

```
Player 1 please enter 7 values:
Enter number 1 > 2
Enter number 2 > 2
You cannot enter the same number twice in a row!
Enter number 2 >
```

As you can see it doesn't accept the repeated input.

Exercise Two (Treasure Hunt)

Methods

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

static void Player1()
{
}

static void Player2()
{
}

static void DrawGrid()
{
}

static void CheckLife()
{
}
```

*Logic Design
before code?*

I started my program by adding the four methods that I believe would be necessary to keep the code organised. The first method is player1, this will ask the user for inputs and store them by using an array. The second method is player2, this method will be asking the second user for inputs and check to see if their guess is correct. If not then a life will be removed. The third method is DrawGrid, this method will be updating the grid whenever player2 enters a new input. The final method is CheckLife, this method checks if player2 has ran out of lives.

Check Life

```
int lives = 3;
```

```
static void CheckLife(int lives)
{
    if (lives <= 0) // check lives reach 0
    {
        Console.WriteLine("You've ran out of lives! Player 1 Wins!");
        Console.ReadLine(); // pause
        Environment.Exit(1); // close
    }
}
```

In the main method I have created a variable called 'lives' which starts at the number 3. I then linked this variable to the check life method so that I'd be able to create an if statement which checks if player2 has ran out of lives or not. If player2 has then a message will be displayed saying player1 has won. I'll be testing if this method works at the same time as the player2 method.

Player 1

Development

```
int[] Xaxis = new int[4]; // array for player 1 X axis
int[] Yaxis = new int[4]; // array for player 1 Y axis
```

I first created these arrays in the main method which will store the first player's inputs.

```
static void Player1(int[] Xaxis, int[] Yaxis)
{
    Console.WriteLine("Player 1 place treasures now:");
    for (int i = 1; i < 4; i++) // player1 input loop
    {
        int repeatingValidation = 0;
        while (repeatingValidation == 0) // validation for repeating coordinates
        {
            int validationloopX = 0;
            while (validationloopX == 0) // validation
            {
            }

            int validationloopY = 0;
            while (validationloopY == 0) // validation
            {
            }
        }
        Console.WriteLine("Treasure " + i + " placed");
    }
}
```

This is the layout I've decided to go for when creating the player1 method. This contains a for loop which will loop every time player1 has entered valid inputs for both X and Y coordinates. Inside of that loop contains 3 more loops for validation. The first loop will make sure that the user doesn't repeat the same coordinates, an example of this is 4, 6 being inputted twice. The other validation loops will make sure that the numbers that the user input will be between 0 and 7 (the coordinates of the grid).

```
int validationloopX = 0;
while (validationloopX == 0) // validation
{
    Console.Write("X: ");
    Xaxis[i] = int.Parse(Console.ReadLine()); // player 1 input for X
    if (Xaxis[i] > 7 || Xaxis[i] < 0) // validation for latest X axis input
    {
        Console.WriteLine("Error, the number you input must be between 0 and 7.");
    }
    else { validationloopX++; } // close validation loop
}

int validationloopY = 0;
while (validationloopY == 0) // validation
{
    Console.Write("Y: ");
    Yaxis[i] = int.Parse(Console.ReadLine()); // player 1 input for Y
    if (Yaxis[i] > 7 || Yaxis[i] < 0) // validation for latest Y axis input
    {
        Console.WriteLine("Error, the number you input must be between 0 and 7.");
    }
    else { validationloopY++; } // close validation loop
}
```

try Parse?

These are the finished validation loops which make sure that the user input is between 0 and 7. It starts by asking the user for an input, once the user inputs a number the if statements will check to see if they are valid. I created two loops, one for X coordinates and one of Y coordinates.

```

    else { validationloop++; } // close validation loop
}

if (i == 1)
{
    repeatingvalidation++; // close repeating loop for first input
}

if (i == 2)
{
    if (Xaxis[i] == Xaxis[1] && Yaxis[i] == Yaxis[1]) // check if coordinates were repeated
    {
        Console.WriteLine("Please do not enter the same coordinates as before!");
    }
    else
    {
        repeatingvalidation++; // close repeating loop
    }
}
else
{
    if (i == 3)
    {
        if (Xaxis[i] == Xaxis[1] && Yaxis[i] == Yaxis[1]) // check if coordinates were repeated
        {
            Console.WriteLine("Please do not enter the same coordinates as before!");
        }
        else
        {
            if (Xaxis[i] == Xaxis[2] && Yaxis[i] == Yaxis[2])
            {
                Console.WriteLine("Please do not enter the same coordinates as before!"); // check if coordinates were repeated
            }
            else
            {
                repeatingvalidation++; // close repeating loop
            }
        }
    }
}

Console.WriteLine("Treasure " + i + " placed");
}

```

Beneath that is the validation which checks to see if the coordinates that the user input were new or not. If they were brand new coordinates then it would move on but if these are repeated coordinates then it will tell the user to enter different coordinates, this would then loop back to the top where it asks the user for the inputs again.

Testing

```

Player 1 place treasures now:
X: 8
Error, the number you input must be between 0 and 7.
X: -1
Error, the number you input must be between 0 and 7.
X: 0
Y: 8
Error, the number you input must be between 0 and 7.
Y: -1
Error, the number you input must be between 0 and 7.
Y: 7
Treasure 1 placed

```

This is the testing for the first set of validation in the player1 method. This checks to see if it allows inputs under 0 or over 7 for both the X and Y coordinates. The testing also checks to see if the validation allows the numbers 0 and 7. As you can see from the testing above the program is working as intended and once there are valid inputs for both X and Y then it will display a message saying that the treasure is placed.

```

X: 0
Y: 7
Please do not enter the same coordinates as before!
X: 7
Y: 0
Treasure 2 placed

```

I then entered the same coordinates again to see if the final validation would allow repeated inputs. As you can see from the testing above the program is working as intended and is rejecting repeated coordinates and asking for the inputs again.

Grid

Development

```
static public string[,] Grid = new string[8, 8];
```

I first started by creating a public multidimensional array. I made this as I believe this is the most effective way to create a grid.

```
static void DrawGrid()
{
    Console.WriteLine("  0  1  2  3  4  5  6  7"); // top of grid
    for (int i = 0; i < 8; i++) // loop to create grid
    {
        Console.Write(i + " "); // numbers for side of grid
        for (int j = 0; j < 8; j++) // fill in the grid
        {
            Grid[i, j] = "-"; // "-" will show a grid spot not guessed
            Console.Write(Grid[i, j] + " "); // space between each spot
        }
        Console.WriteLine(); // cut to next line of grid
    }
}
```

In the DrawGrid method I created a loop which will create the grid. The loop shows what to show empty inputs as.

Testing

```
Player 1 place treasures now:
X: 1
Y: 1
Treasure 1 placed
X: 2
Y: 2
Treasure 2 placed
X: 3
Y: 3
Treasure 3 placed
  0  1  2  3  4  5  6  7
0 - - - - - - - -
1 - - - - - - - -
2 - - - - - - - -
3 - - - - - - - -
4 - - - - - - - -
5 - - - - - - - -
6 - - - - - - - -
7 - - - - - - - -
```

As you can see from the testing the player1 method stops looping after 3 valid inputs which shows that it is working as intended. Another test in the screenshot above is for the grid being created. As you can see the grid is displayed properly with nothing out of place.

Player 2

Development

```
int[] player2X = new int[8]; // array for player 2 X axis
int[] player2Y = new int[8]; // array for player 2 Y axis
int lives = 3;
int[] player2Guess = new int[4];
```

I first created 3 more arrays in the main method. The first array is for storing player2 X axis inputs which will be added to the grid later on. The second array is the same except for Y axis inputs. The final array I created will be used for validating if the user has already guessed this or not. If the user has guessed then a life would be taken off as if they had gotten it wrong.

```
static void Player2(int[] Xaxis, int[] Yaxis, int[] player2X, int[] player2Y, int lives, int[] player2Guess)
{
    for (int i = 1; i < 8; i++) // player2 input loop
    {
        Console.WriteLine("Player 2 Guess");
        int validationloopX = 0;
        while (validationloopX == 0) // validation
        {
            Console.Write("X: ");
            player2X[i] = int.Parse(Console.ReadLine()); // player 2 X input
            if (player2X[i] > 7 || player2X[i] < 0) // validation for latest player2 X input
            {
                Console.WriteLine("Error, the number you input must be between 0 and 7.");
            }
            else { validationloopX++; } // close validation loop
        }

        int validationloopY = 0;
        while (validationloopY == 0) // validation
        {
            Console.Write("Y: ");
            player2Y[i] = int.Parse(Console.ReadLine()); // player 2 Y input
            if (player2Y[i] > 7 || player2Y[i] < 0) // validation for latest player2 Y input
            {
                Console.WriteLine("Error, the number you input must be between 0 and 7.");
            }
            else { validationloopY++; } // close validation loop
        }
    }
}
```

I decided to create the same validation loops as I did for player1 when checking if the inputs were between 0 and 7. As you can see from the code it is pretty much the same other than the variables being used.

```
if (player2X[i] == Xaxis[1] && player2Y[i] == Yaxis[1]) // check guess
{
    if (player2Guess[i] == 0) // check if already guessed
    {
        Console.WriteLine("You have found one of the treasures!");
        TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
        player2Guess[i] = 1; // prevent guessing answer again
    }
    else //if already guessed
    {
        TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
        Console.WriteLine("You've already guessed that!");
        lives = lives - 1; // remove life
        Console.WriteLine("Lives left: " + lives);
        Console.WriteLine();
        CheckLife(lives); // check lives
    }
}
```

Here is the first part of the if statement that I have created which checks if the guess is correct. This part only checks if it is correct with the first set of inputs that player1 entered. If the guess was correct then it will check if it was already guessed, if it was already guessed then a life will be removed and the program will check lives. If it wasn't already guessed and is correct then it will update the grid and prevent them inputs from being guessed again.

```

else
{
    if (player2X[i] == Xaxis[2] && player2Y[i] == Yaxis[2]) // check guess
    {
        if (player2Guess[2] == 0) // check if already guessed
        {
            Console.WriteLine("You have found one of the treasures!");
            TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
            player2Guess[2] = 1; // prevent guessing answer again
        }
        else // if already guessed
        {
            TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
            Console.WriteLine("You've already guessed that!");
            lives = lives - 1; // remove life
            Console.WriteLine("Lives left: " + lives);
            Console.WriteLine();
            CheckLife(lives); // check lives
        }
    }
    else
    {
        if (player2X[i] == Xaxis[3] && player2Y[i] == Yaxis[3]) // check guess
        {
            if (player2Guess[3] == 0) // check if already guessed
            {
                Console.WriteLine("You have found one of the treasures!");
                TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
                player2Guess[3] = 1; // prevent guessing answer again
            }
            else // if already guessed
            {
                TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
                Console.WriteLine("You've already guessed that!");
                lives = lives - 1; // remove life
                Console.WriteLine("Lives left: " + lives);
                Console.WriteLine();
                CheckLife(lives); // check lives
            }
        }
    }
}

```

This is then repeated twice but for the other two treasures that player1 placed.

```

else // if player2 guess is wrong
{
    Console.WriteLine("Wrong! You lose a life!");
    lives = lives - 1; // remove life
    Console.WriteLine("Lives left: " + lives);
    Console.WriteLine();
    TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
    CheckLife(lives); // check lives
}

```

I then added a final 'else' if the guess that player2 did was wrong. This will update the grid and remove a life.

```

if (player2Guess[1] == 1 && player2Guess[2] == 1 && player2Guess[3] == 1) // player 2 win condition
{
    Console.WriteLine("Player 2 Wins!!");
    Console.ReadLine(); // pause
    Environment.Exit(1); // close
}

```

This is player2's win condition. If they manage to guess all three correct then it will display a message saying that player2 has won.

Grid Updating

```

Console.WriteLine(" 0 1 2 3 4 5 6 7"); // top of grid
for (int i = 0; i < 8; i++) // loop to create grid
{
    Console.Write(i + " "); // numbers for side of grid
    for (int j = 0; j < 8; j++) // fill in the grid
    {
        Grid[i, j] = "-"; // "-" will show a grid spot not guessed
        for (int d = 1; d < 8; d++)
        {
            if (player2X[d] == Xaxis[i] && player2Y[d] == Yaxis[j]) // check if 1st treasure is guessed
            {
                Grid[player2X[d], player2Y[d]] = "X"; // change correct to X
            }
            else
            {
                if (player2X[d] == Xaxis[2] && player2Y[d] == Yaxis[2]) // check if 2nd treasure is guessed
                {
                    Grid[player2X[d], player2Y[d]] = "X"; // change correct to X
                }
                else
                {
                    if (player2X[d] == Xaxis[3] && player2Y[d] == Yaxis[3]) // check if 3rd treasure is guessed
                    {
                        Grid[player2X[d], player2Y[d]] = "X"; // change correct to X
                    }
                    else
                    {
                        Grid[player2X[d], player2Y[d]] = "N"; // change incorrect to N
                    }
                }
            }
        }
        Console.Write(Grid[i, j] + " "); // space between each spot
    }
    Console.WriteLine(); // cut to next line of grid
}

```

I added this to the grid so that it would update whenever player2 would input valid coordinates. The grid will update each time player2 has two valid inputs. If player2 finds one of the treasures then the grid spot where the treasure was will be updated from '-' to 'X'. If player2 doesn't find a treasure with that guess then that grid spot will be updated from '-' to 'N'.

Testing

```

Player 2 Guess
X: 8
Error, the number you input must be between 0 and 7.
X: -1
Error, the number you input must be between 0 and 7.
X: 0
Y: 8
Error, the number you input must be between 0 and 7.
Y: -1
Error, the number you input must be between 0 and 7.
Y: 7
Wrong! You lose a life!
Lives left: 2

```

This testing checks to see if the validation for the inputs work. As you can see from the testing above it doesn't allow numbers under 0 and above 7 which is working as intended. Another test is to see what happens if the guess is wrong, as you can see from the screenshot if the guess is wrong then it will display the amount of lives you have left as intended.

```

Player 2 Guess
X: 1
Y: 1
You have found one of the treasures!

```

This is testing to see what happens if you find a treasure. As you can see the program works as intended and tells the user that they have found a treasure.

```
Player 2 Guess
X: 1
Y: 1
You've already guessed that!
Lives left: 2
```

This is to test what happens if you guess the same as a treasure that has already been found. As you can see from the testing the program works as intended and removes a life.

```
Player 2 Guess
X: 2
Y: 2
You have found one of the treasures!
  0  1  2  3  4  5  6  7
0 N  -  -  -  -  -  -  -
1 -  X  -  -  -  -  -  -
2 -  -  X  -  -  -  -  -
3 -  -  -  -  -  -  -  -
4 -  -  -  -  -  -  -  -
5 -  -  -  -  -  -  -  -
6 -  -  -  -  -  -  -  -
7 -  -  -  -  -  -  -  -
Player 2 Guess
X: 3
Y: 3
You have found one of the treasures!
  0  1  2  3  4  5  6  7
0 N  -  -  -  -  -  -  -
1 -  X  -  -  -  -  -  -
2 -  -  X  -  -  -  -  -
3 -  -  -  X  -  -  -  -
4 -  -  -  -  -  -  -  -
5 -  -  -  -  -  -  -  -
6 -  -  -  -  -  -  -  -
7 -  -  -  -  -  -  -  -
Player 2 Wins!!
```

This testing here checks what happens if player2 manages to find all of the treasures. As you can see it will display the win message as intended. Another test on this screenshot is the updating grid. As you can see an X is added when player2 finds one of the treasures.

```

  0  1  2  3  4  5  6  7
0 N - - - - - - -
1 - X - - - - - -
2 - - - - - - -
3 - - - - - - -
4 - - - - N - -
5 - - - - - N -
6 - - - - - - -
7 - - - - - - -
Player 2 Guess
X: 6
Y: 2
Wrong! You lose a life!
Lives left: 0

  0  1  2  3  4  5  6  7
0 N - - - - - - -
1 - X - - - - - -
2 - - - - - - -
3 - - - - - - -
4 - - - - N - -
5 - - - - - N -
6 - - N - - - -
7 - - - - - - -
You've ran out of lives! Player 1 Wins!

```

This test here checks to see what happens if you run out of lives. As you can see the program works as intended and displays a player 1 wins message. Another test is to see if the grid updates as intended even if the guess is wrong which it does.

Problems

```

X: 2
Y: 2
Treasure 2 placed
X: 3
Y: 3
Treasure 3 placed
  0  1  2  3  4  5  6  7
0 - - - - - - -
1 - - - - - - -
2 - - - - - - -
3 - - - - - - -
4 - - - - - - -
5 - - - - - - -
6 - - - - - - -
7 - - - - - - -
Player 2 Guess
X: 1
Y: 1
You have found one of the treasures!
  0  1  2  3  4  5  6  7
0 N - - - - - - -
1 - X - - - - - -
2 - - - - - - -
3 - - - - - - -
4 - - - - - - -
5 - - - - - - -
6 - - - - - - -
7 - - - - - - -

```

At the coordinate 0, 0 it will always display an N if a treasure isn't there and an X if a treasure is there even if player2 is yet to guess that area.

Appendix

Program 1

```
using System;
```

```
//Bradley Martin MAR16829326
```

```
namespace Assignment_Exercise_1
```

```
{
```

```
    class Program
```

```
    {
```

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

```
        {
```

```
            int[] num = new int[7]; // array for player 1 numbers
            Player1(num);
```

```
            int numberP2 = num[0]; // beginning value for player 2
            Player2(num, numberP2);
```

```
        }
```

```
        static void Player1(int[] num)
```

```
        {
```

```
            Console.WriteLine("Player 1 please enter 7 values:");
```

```
            for (int i = 0; i < num.Length; i++) // for loop to ask the user 7 times
```

```
            {
```

```
                int number = i + 1;
```

```
                int validationloop = 0;
```

```
                while (validationloop == 0) // validation
```

```
                {
```

```
                    Console.Write("Enter number " + number + " > "); // asks user for
```

inputs

```
                    num[i] = int.Parse(Console.ReadLine());
```

10 or below 1

```
                    if (num[i] > 10 || num[i] < 1) // checking if the number is above
```

```
                    {
```

rejecting false input from user

```
                        Console.WriteLine("Number must be between 1 and 10"); //
```

```
                    }
```

```
                    else
```

```
                    {
```

```
                        if (i > 0) // check if its the first input
```

```
                        {
```

```
                            if (num[i] == num[i - 1]) // check if same as last input
```

```
                            {
```

twice in a row!");

```
                                Console.WriteLine("You cannot enter the same number
```

```
                            }
```

```
                        }
                        else
```

```
                        {
```

```
                            validationloop++; // close validation loop
```

```
                        }
```

```
                    }
```

```
                    else
```

```
                    {
```

```
                        validationloop++; // close validation loop
```

```
                    }
```

```
                }
```

```
            }
```

```
        }
```

```
        Console.WriteLine();
```

```

    }

    static void Player2(int[] num, int numberP2)
    {
        Console.WriteLine("Player 2 the first value is " + numberP2); // tells
        player 2 beginning value
        for (int i = 1; i < num.Length; i++) // loop for player2 inputs
        {
            int validationloop = 0;
            while (validationloop == 0) // validation
            {
                Console.Write("Is your current value Higher (H) or Lower (L) than
the next value? > ");
                string highLow = Console.ReadLine(); // player 2 input

                if (highLow == "H" || highLow == "L")
                {
                    validationloop++; // close loop
                    int numberP2next = i;
                    if (highLow == "H") // checks if H was the input
                    {
                        if (numberP2 > num[numberP2next]) // correct guess
                        {
                            Console.WriteLine("Correct, next value is " +
num[numberP2next]); // next value displayed
                            numberP2 = num[numberP2next]; // update value
                        }
                        else // incorrect guess
                        {
                            Console.WriteLine("Incorrect, sorry Player 1 wins.");
// tells player 2 they are incorrect
                            Console.ReadLine(); // pause
                            Environment.Exit(1); // closes program
                        }
                    }
                    if (highLow == "L") // checks if L was the input
                    {
                        if (numberP2 < num[numberP2next]) // correct guess
                        {
                            Console.WriteLine("Correct, next value is " +
num[numberP2next]); // next value displayed
                            numberP2 = num[numberP2next]; // update value
                        }
                        else // if number isn't higher like player 2 guessed
                        {
                            Console.WriteLine("Incorrect, sorry Player 1 wins.");
// tells player 2 they are incorrect
                            Console.ReadLine(); // pause
                            Environment.Exit(1); // closes program
                        }
                    }
                }
                else
                {
                    Console.WriteLine("Please enter either H or L");
                }
            }
        }
        Console.WriteLine("You managed to beat Player 1! Well Done!");
        Console.ReadLine(); // pause
    }
}

```

```
}
}
```

Program 2

```
using System;
```

```
//Bradley Martin MAR16829326
```

```
namespace Assignment_Exercise_2
```

```
{
```

```
    class Program
```

```
    {
```

```
        static public string[,] Grid = new string[8, 8];
```

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

```
        {
```

```
            int[] Xaxis = new int[4]; // array for player 1 X axis
```

```
            int[] Yaxis = new int[4]; // array for player 1 Y axis
```

```
            Player1(Xaxis, Yaxis);
```

```
            int[] player2X = new int[8]; // array for player 2 X axis
```

```
            int[] player2Y = new int[8]; // array for player 2 Y axis
```

```
            int lives = 3;
```

```
            int[] player2Guess = new int[4];
```

```
            DrawGrid();
```

```
            Player2(Xaxis, Yaxis, player2X, player2Y, lives, player2Guess);
```

```
        }
```

```
        static void Player1(int[] Xaxis, int[] Yaxis)
```

```
        {
```

```
            Console.WriteLine("Player 1 place treasures now:");
```

```
            for (int i = 1; i < 4; i++) // player1 input loop
```

```
            {
```

```
                int repeatingValidation = 0;
```

```
                while (repeatingValidation == 0) // validation for repeating
```

coordinates

```
                {
```

```
                    int validationloopX = 0;
```

```
                    while (validationloopX == 0) // validation
```

```
                    {
```

```
                        Console.Write("X: ");
```

```
                        Xaxis[i] = int.Parse(Console.ReadLine()); // player 1 input
```

for X

```
                        if (Xaxis[i] > 7 || Xaxis[i] < 0) // validation for latest X
```

axis input

```
                        {
```

```
                            Console.WriteLine("Error, the number you input must be
```

between 0 and 7.");

```
                        }
```

```
                        else { validationloopX++; } // close validation loop
```

```
                    }
```

```
                    int validationloopY = 0;
```

```
                    while (validationloopY == 0) // validation
```

```
                    {
```

```
                        Console.Write("Y: ");
```

```
                        Yaxis[i] = int.Parse(Console.ReadLine()); // player 1 input
```

for Y

```
                        if (Yaxis[i] > 7 || Yaxis[i] < 0) // validation for latest Y
```

axis input

```

        {
            Console.WriteLine("Error, the number you input must be
between 0 and 7.");
        }
        else { validationloopY++; } // close validation loop
    }

    if (i == 1)
    {
        repeatingValidation++; // close repeating loop for first input
    }

    if (i == 2)
    {
        if (Xaxis[i] == Xaxis[1] && Yaxis[i] == Yaxis[1]) // check if
coordinates were repeated
        {
            Console.WriteLine("Please do not enter the same
coordinates as before!");
        }
        else
        {
            repeatingValidation++; // close repeating loop
        }
    }
    else
    {
        if (i == 3)
        {
            if (Xaxis[i] == Xaxis[1] && Yaxis[i] == Yaxis[1]) // check
if coordinates were repeated
            {
                Console.WriteLine("Please do not enter the same
coordinates as before!");
            }
            else
            {
                if (Xaxis[i] == Xaxis[2] && Yaxis[i] == Yaxis[2])
                {
                    Console.WriteLine("Please do not enter the same
coordinates as before!"); // check if coordinates were repeated
                }
                else
                {
                    repeatingValidation++; // close repeating loop
                }
            }
        }
    }
}
Console.WriteLine("Treasure " + i + " placed");
}
}

static void Player2(int[] Xaxis, int[] Yaxis, int[] player2X, int[] player2Y,
int lives, int[] player2Guess)
{
    for (int i = 1; i < 8; i++) // player2 input loop
    {
        Console.WriteLine("Player 2 Guess");
        int validationloopX = 0;
        while (validationloopX == 0) // validation

```

```

    {
        Console.Write("X: ");
        player2X[i] = int.Parse(Console.ReadLine()); // player 2 X input
        if (player2X[i] > 7 || player2X[i] < 0) // validation for latest
            {
                Console.WriteLine("Error, the number you input must be between
0 and 7.");
            }
        else { validationloopX++; } // close validation loop
    }

    int validationloopY = 0;
    while (validationloopY == 0) // validation
    {
        Console.Write("Y: ");
        player2Y[i] = int.Parse(Console.ReadLine()); // player 2 Y input
        if (player2Y[i] > 7 || player2Y[i] < 0) // validation for latest
            {
                Console.WriteLine("Error, the number you input must be between
0 and 7.");
            }
        else { validationloopY++; } // close validation loop
    }

    if (player2X[i] == Xaxis[1] && player2Y[i] == Yaxis[1]) // check guess
    {
        if (player2Guess[1] == 0) // check if already guessed
        {
            Console.WriteLine("You have found one of the treasures!");
            TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
            player2Guess[1] = 1; // prevent guessing answer again
        }
        else //if already guessed
        {
            TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
            Console.WriteLine("You've already guessed that!");
            lives = lives - 1; // remove life
            Console.WriteLine("Lives left: " + lives);
            Console.WriteLine();
            CheckLife(lives); // check lives
        }
    }
    else
    {
        if (player2X[i] == Xaxis[2] && player2Y[i] == Yaxis[2]) // check
guess
{
            if (player2Guess[2] == 0) // check if already guessed
            {
                Console.WriteLine("You have found one of the treasures!");
                TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
                player2Guess[2] = 1; // prevent guessing answer again
            }
            else // if already guessed
            {
                TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
                Console.WriteLine("You've already guessed that!");
                lives = lives - 1; // remove life
            }
        }
    }
}

```

```

        Console.WriteLine("Lives left: " + lives);
        Console.WriteLine();
        CheckLife(lives); // check lives
    }
}
else
{
    if (player2X[i] == Xaxis[3] && player2Y[i] == Yaxis[3]) //
check guess
    {
        if (player2Guess[3] == 0) // check if already guessed
        {
            Console.WriteLine("You have found one of the
treasures!");
            TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
            player2Guess[3] = 1; // prevent guessing answer again
        }
        else // if already guessed
        {
            TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
            Console.WriteLine("You've already guessed that!");
            lives = lives - 1; // remove life
            Console.WriteLine("Lives left: " + lives);
            Console.WriteLine();
            CheckLife(lives); // check lives
        }
    }
}
else // if player2 guess is wrong
{
    Console.WriteLine("Wrong! You lose a life!");
    lives = lives - 1; // remove life
    Console.WriteLine("Lives left: " + lives);
    Console.WriteLine();
    TreasureGrid(player2X, player2Y, Xaxis, Yaxis);
    CheckLife(lives); // check lives
}
}
}
if (player2Guess[1] == 1 && player2Guess[2] == 1 && player2Guess[3] ==
1) // player 2 win condition
{
    Console.WriteLine("Player 2 Wins!!");
    Console.ReadLine(); // pause
    Environment.Exit(1); // close
}
}
}

static void DrawGrid()
{
    Console.WriteLine("  0  1  2  3  4  5  6  7"); // top of grid
    for (int i = 0; i < 8; i++) // loop to create grid
    {
        Console.Write(i + " "); // numbers for side of grid
        for (int j = 0; j < 8; j++) // fill in the grid
        {
            Grid[i, j] = " - "; // "-" will show a grid spot not guessed
            Console.Write(Grid[i, j] + " "); // space between each spot
        }
        Console.WriteLine(); // cut to next line of grid
    }
}

```

```

    }
}

static void TreasureGrid(int[] player2X, int[] player2Y, int[] Xaxis, int[]
Yaxis)
{
    Console.WriteLine("  0  1  2  3  4  5  6  7"); // top of grid
    for (int i = 0; i < 8; i++) // loop to create grid
    {
        Console.Write(i + " "); // numbers for side of grid
        for (int j = 0; j < 8; j++) // fill in the grid
        {
            Grid[i, j] = "- "; // "-" will show a grid spot not guessed
            for (int d = 1; d < 8; d++)
            {
                if (player2X[d] == Xaxis[1] && player2Y[d] == Yaxis[1]) //
check if 1st treasure is guessed
                {
                    Grid[player2X[d], player2Y[d]] = " X "; // change correct
to X
                }
                else
                {
                    if (player2X[d] == Xaxis[2] && player2Y[d] == Yaxis[2]) //
check if 2nd treasure is guessed
                    {
                        Grid[player2X[d], player2Y[d]] = " X "; // change
correct to X
                    }
                    else
                    {
                        if (player2X[d] == Xaxis[3] && player2Y[d] ==
Yaxis[3]) // check if 3rd treasure is guessed
                        {
                            Grid[player2X[d], player2Y[d]] = " X "; // change
correct to X
                        }
                        else
                        {
                            Grid[player2X[d], player2Y[d]] = " N "; // change
incorrect to N
                        }
                    }
                }
            }
            Console.Write(Grid[i, j] + " "); // space between each spot
        }
        Console.WriteLine(); // cut to next line of grid
    }
}

static void CheckLife(int lives)
{
    if (lives <= 0) // check lives reach 0
    {
        Console.WriteLine("You've ran out of lives! Player 1 Wins!");
        Console.ReadLine(); // pause
        Environment.Exit(1); // close
    }
}
}
}

```

Program 3

```
using System;
```

```
//Bradley Martin MAR16829326
```

```
namespace Assignment_Exercise_3
```

```
{
```

```
    class Program
```

```
    {
```

```
        static public string[,] Grid = new string[6, 6];
```

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

```
        {
```

```
            DrawGrid();
```

```
            Console.ReadLine();
```

```
        }
```

```
        static void Player1Turn()
```

```
        {
```

```
        }
```

```
        static void Player2Turn()
```

```
        {
```

```
        }
```

```
        static void DrawGrid()
```

```
        {
```

```
            Console.WriteLine(" 0   1   2   3   4   5"); // top of grid
```

```
            for (int i = 0; i < 6; i++) // loop to create grid
```

```
            {
```

```
                for (int j = 0; j < 6; j++) // fill in the grid
```

```
                {
```

```
                    Grid[i, j] = " - "; // "-" will show a grid spot not guessed
```

```
                    Console.Write(Grid[i, j] + " "); // space between each spot
```

```
                }
```

```
                Console.WriteLine(); // cut to next line of grid
```

```
            }
```

```
        }
```

```
    }
```

```
}
```