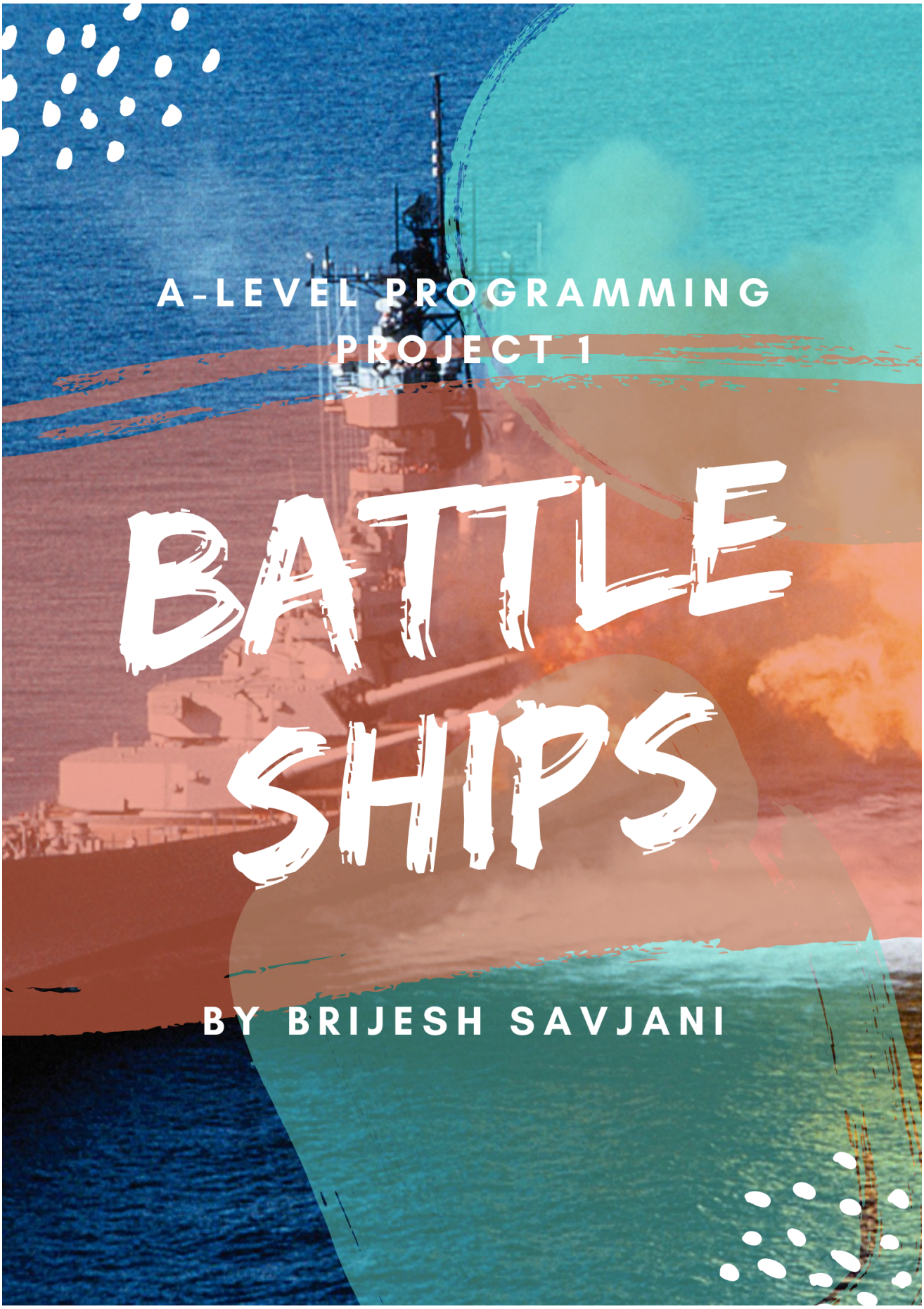
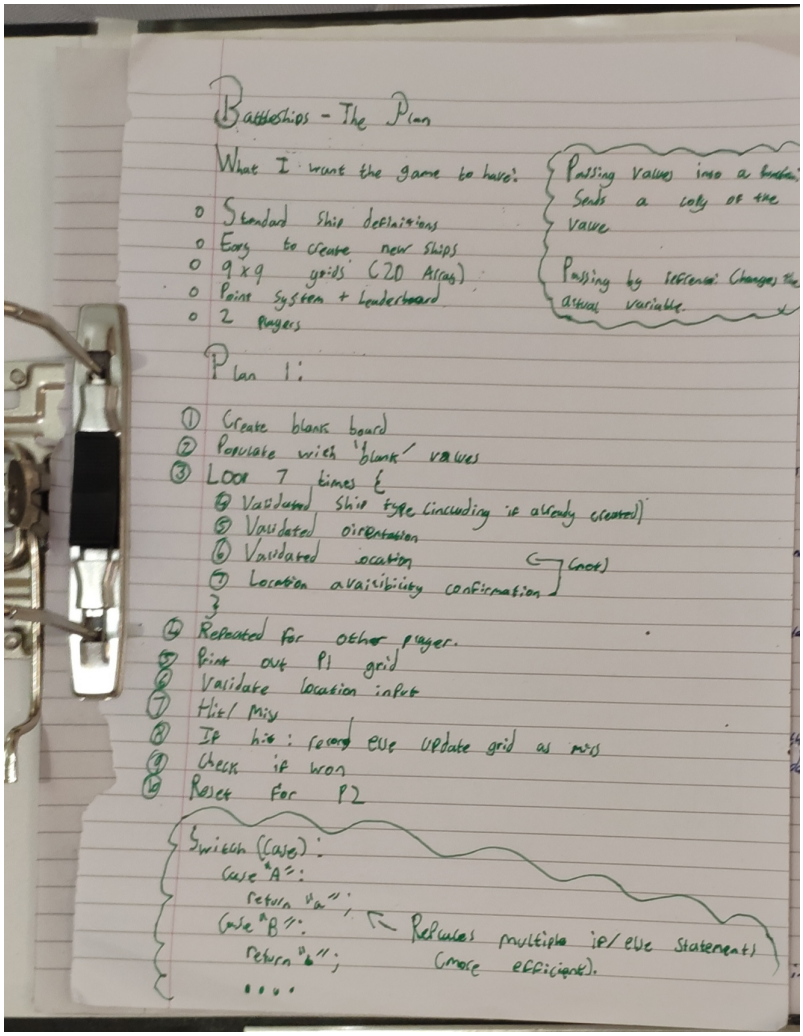

A-LEVEL PROGRAMMING
PROJECT 1
BATTLE SHIPS
BY BRIJESH SAVJANI

Planning Stage

Initial Plan

The initial plan of the Battleships program was done together as a Class. However, I did change this because I felt that it might be better to expose myself to more complicated syntax and ideas in C# (like objects) as I felt I had a good enough grip on the basics.



Success Criteria

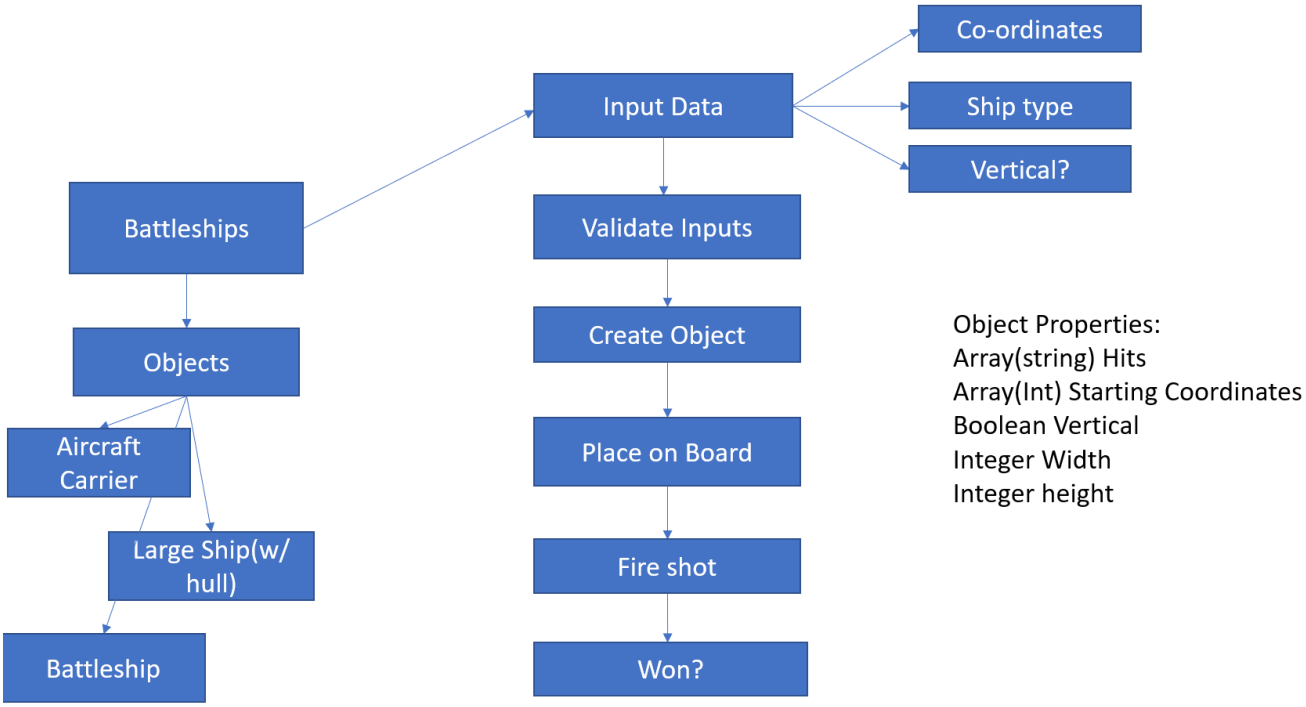
For my program to be judged as a total success I want it to pass the following goals:

- [ ] Have a functioning 2 player battleships game
- [ ] Use Object Orienated Programming
- [ ] Use a switch case statement
- [ ] Make it effiecient
- [ ] ASPIRATIONAL: Have a leaderboard
- [ ] ASPIRATIONAL: Have an 'AI'

Problem decomposition

I used Problem decomposition to break down battleships into it's key components. I then made these into individual key components into functions later on.

Here is my decomposed plan of Battleships:



Abstraction

I used abstraction whilst making my Battleships game to make the development process easier and more efficient. An example of my use of abstraction can be found in the functions. To make my functions I removed (abstracted) the context of the function in the game and simply focused on the end result of each individual function. I then simply combined the functions in my main method at the end.

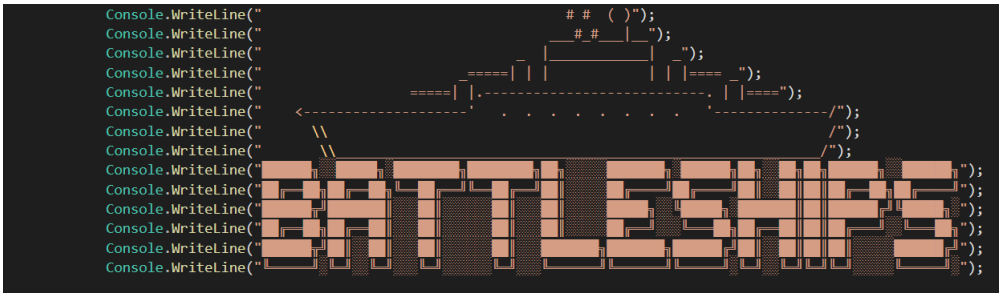
Code

The main body of code

```
    //Declared at top of code before functions
    static char[,] BoardP1 = new char[9,9];// Makes Player 1's Board
    static char[,] BoardP2 = new char[9, 9];//Makes Player 2's board
    public static List<string> CreatedShips = new List<string>();//Array of
Created ship names(Player 1)
    static ship[] shiparray = new ship[3]; // Object Array for Player 1
    public static List<string> CreatedShipsP2 = new List<string>();//Array of
Created ship names-Player 2
    static ship[] shiparrayP2 = new ship[3];//Object Array for Player 2
```

As you may be able to tell I did change the structure of Objects later on in my development process. The reasoning for this will be discussed later within the Object Orientated section of my .

Main method



//Code for ASCII art to make it look nice

```
//Subsection 1
    try
//Try to catch any Parse Errors
        {
            EmptyBoard(BoardP1);//Empties out the Player 1 board
            EmptyBoard(BoardP2);//Empties out Player 2 board
            for (int p = 1; p <= 2; p++)// For both Players
            {
                for (int i = 1; i <= 3; i++)//For every ship type
                {
```

This code is responsible for Emptying the Boards and for starting the loop so the Player has to create all the Ship types.

```
                    Console.WriteLine("For Ship #" + i.ToString() + " Please
enter your Ship Type or type Armoury for your availibe options"); //Tells the
user the ship type
                    Console.Read();//Create pause
                    string shiptype = Console.ReadLine();//Capture User input
                    string createship = ShipValidation(shiptype.ToUpper(),p);
// Validates ship
                    if (shiptype.ToUpper() == "ARMOURY")
                    {
                        Console.WriteLine("You have to place 1 Aircraft
Carrier, 1 battleship and 1 Large Ship");
                        i = i - 1;// Rewinds loop
                    }
                    else if (createship == "false")
                    {
                        Console.WriteLine("You have either already created
this ship or have entered an invalid ship name!");
                        i = i - 1;
                        //Invalid Ship
                    }
```

This code is responsible for gathering the Ship type and then Creating the ship.

```
                if (orientation.ToUpper() == "VERTICAL")//If vertical
```

```
                    {

                        switch (createship) //Use of Switch Case
                            {
                                case "AC"://Aircraft Carrier
                                    if (p == 1)//For player 1
                                    {
                                        shiparray[0].vertical = true;//Set the
ship to a vertical
orientation
                                        shiparray[0].starting_coordinate[0] =
int.Parse(coordinates.Split(',')[1].ToString()); //Set starting coordinates
inside an Array in the object
                                        shiparray[0].starting_coordinate[1] =
int.Parse(coordinate_input[1].ToString());//Set starting coordinates inside an
Array in the object
                                        Created = PlaceShip(shiparray[0],
BoardP1);//Places ship
                                        //Adds details to Ship array
                                    }
                                    else
                                    {
                                        shiparrayP2[0].vertical = true;
                                        shiparrayP2[0].starting_coordinate[0]
= int.Parse(coordinates.Split(',')[1].ToString()); ;
                                        shiparrayP2[0].starting_coordinate[1]
= int.Parse(coordinate_input[1].ToString());
                                        Created = PlaceShip(shiparrayP2[0],
BoardP2);//Repeat for Player 2
                                    }
                                    break;

                                case "BSHIP":
                                  ...//Similiar to first example
                                case "LGSHIP":
                                ...//Similiar to first example

                                case "BSHIP":
                                  ...//Similiar to first example
                                    break;
                                case "LGSHIP":
                                ...//Similiar to first example
                                    break;
                            }
                        }//Else Looks exactly the same but vertical is set to
false
```

Overall this section of code is the bit that I am least happy with as it's the most inefficient and repeats itself. This is mainly because I ran out of time and could not optimise changing the Player1 Array to a Player 2 array. Given more time I would definitely try and optimise this to the maximum. Furthermore, the bits in the Switch case are repeated and if given more time I would just make another function that takes in a Ship and the starting coordinates and then sets the coordinates for that ship. This would cut down on a huge amount of lines however I just ran out of time to do this.

```
                        if (Created == true)// If the Ship object has been
created
                        {
                            if (p == 1)//If player 1
                            {
                                OutputBoard(BoardP1);//Output Player 1 board
                            }
                            else
                            {
                                OutputBoard(BoardP2);//Otherwise output Player
2 Board
                            }
                            Console.WriteLine("Press any key to
continue");//Tells user how to continue
                            Console.Read();//Pauses so they can see where ship
is placed
                            Console.Clear();//Clears Console
                        }
                        else
                        {
                            Console.WriteLine("Ships overlap"!);//Alert user
if they're ships overlap
                            i =i -1;//Reset Loop
                        }
                    }
                    else
                    {
                        Console.WriteLine("Invalid Coordinate");//links upto
previous if;For coordinates
                        i = i - 1;//Reset loops
                    }
```

This code is responsible for checking if the object has been created and then showing where it is on the board. If the object hasn't been created then it alerts the user as to why it has not been created and reset the loops so they can correct their mistake.

```
                }//Ends for loop as all the ships would have been created.
                Console.WriteLine("Write target coordinates");
                string target = Console.ReadLine();//Takes in Traget
Coordinates
                if (p == 1)// For player 1
                {
                    ShotsFired(target, BoardP2); // Fires shot on Player2's
Board
                    bool won = WinCondition(BoardP2);//Checks if they've won
                    if (won == true)
                    {
                        Console.WriteLine("Player 1 wins");// Alerts them to
victory
                        break;//Breaks out.
                    }
                }
                else
                {//Same for player 2
```

```
                    ShotsFired(target, BoardP1);
                    bool won = WinCondition(BoardP1);
                    if (won == true)
                    {
                        Console.WriteLine("Player 2 wins");
                        break;
                    }
                }
            }
        }
```

This code handles the shots that the Players make upon each other and checks if any of the players have one.

Functions

Ship Validation Function

```
    public static string ShipValidation(string shiptype,int p)
    {
        bool created = false;// Creates Boolean to say if created
        foreach (string item in CreatedShips)//Foreach statement of all
created ships
        {
            if (shiptype == item)// If created
            {
                created = true;//sets created as true
                break;//breaks loop
            }
        }
        if (created == false)//If not created
        {
            switch (shiptype)
            {
                case "AIRCRAFT CARRIER"://Switch case that creates the ship
and sets height & width
                    //also records that the ship has been created
                    if (p == 1)
                    {
                        shiparray[0] = new ship { height = 4, width = 2 };
                        CreatedShips.Add(shiptype.ToUpper());
                    }
                    else
                    {
                        shiparrayP2[0] = new ship { height = 4, width = 2 };
                        CreatedShipsP2.Add(shiptype.ToUpper());
                    }
                    return "AC";
                case "BATTLESHIP":
                    if (p == 1)
                    {
                        shiparray[1] = new ship { height = 3, width = 4 };
                        CreatedShips.Add(shiptype.ToUpper());
                    }
                    else
```

```
                    {
                        shiparrayP2[1] = new ship { height = 3, width = 4 };
                        CreatedShipsP2.Add(shiptype.ToUpper());
                    }
                    return "BSHIP";
                case "LARGE SHIP":
                    if (p == 1)
                    {
                        shiparray[2] = new ship { height = 5, width = 3 };
                        CreatedShips.Add(shiptype.ToUpper());
                    }
                    else
                    {
                        shiparrayP2[2] = new ship { height = 5, width = 3 };
                        CreatedShipsP2.Add(shiptype.ToUpper());
                    }
                    return "LGSHIP";
                default:
                    return "false";

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

This code validates the Ship name and if the ship has been created. If it has then it creates the ship and sets the height & width of the ship. It also records that the ship has been created.

Coordinate validation

```
 public static string UserCoordinateValidation(string coordinates)
     {
     string[] valid_letters = new string[9] { "A", "B", "C", "D", "E", "F",
"G", "H", "I" };
     int[] valid_numbers = new int[9] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
     string vll = "";
     string vln = "";
     int letter_translated = 0;
     //^ Creates Variables
    foreach (string letter in valid_letters)//Foreach loop checks if letter is
valid
        {
            letter_translated = letter_translated + 1;//Records letter value
for final iteration
            if (letter == coordinates[0].ToString())//If iterated letter is
Input then mark vll(VaLidLetter)as true
            {
                vll = "valid";
                break;//break loop
            }
```

```
        }
        foreach (int number in valid_numbers)
        {
            if (number == int.Parse(coordinates[1].ToString()))//Checks if
number is input
            {
                vln = "valid";// Marks VaLid Number as true because value in
range
                break;
            }
        }
        if (vll == "valid" & vln == "valid")
        {

            return ("Valid," + letter_translated.ToString());// Return valid
and letter value
        }
        else
        {
            return ("Invalid");//Else Return invalid
        }
    }
```

This code validates any User inputted coordinates.

Place ships

```
 public static bool PlaceShip(ship ship, char[,] Board)
        {
            try//Catches exception if Ship goes of board so user can be
alerted
            {
                if (ship.vertical == false)//If not vertical
                {
                    for (int y = 0; y < ship.height; y++)//Modifies
Board[x_val,y_value] to enter board in ship.
                    {
                        for (int x = 0; x < ship.width; x++)
                        {
                            int x_val = x + ship.starting_coordinate[1];
                            int y_val = y + ship.starting_coordinate[0] - 1;
                            if (Board[x_val, y_val] == 'e')
                            {
                                Board[x_val, y_val] = 'j';
                            }
                            else
                            {
                                return false;
                        }
                        }
                    }
                }
                else//Does same except if vertical
```

```
            {
                for (int y = ship.starting_coordinate[0]; y <
ship.starting_coordinate[0] + ship.width; y++)
                {
                    for (int x = ship.starting_coordinate[1]; x <
ship.starting_coordinate[1] + ship.height; x++)
                    {
                        if (Board[x-1, y-1] == 'e')
                        {
                            Board[x-1 , y-1] = 'j';
                        }
                        else
                        {
                            return false;
                        }
                    }
                }

            }
            return true;
        }
        catch
        {
            return false;
        }
        }
```

This code places a ship on the board and returns false if it was unable to(due to Ship going out of bounds or board)

Shots Fired

```
    public static void ShotsFired(string coordinates, char[,] Board)
    {
        string valid = UserCoordinateValidation(coordinates);  //Validate
coordinates
        if (valid.Split(',')[0] =="Valid")//If coordinates are valid
        {
            int i = int.Parse(valid.Split(',')[1]);
            Board[int.Parse(coordinates[1].ToString())-1,i-1] = 'h';//Write h
on the board(Shot)
        }
        }
```

This code validates the coordinate and then places the shot on the board.

Clear board

```
    public static void EmptyBoard(Char[,] Board)
    {
        for (int Y = 0; Y < Board.GetLength(0); Y++)// For all Y values
        {
            for (int X = 0; X < Board.GetLength(0); X++)//For all X values
            {
                Board[Y, X] = 'e';//Set all values to e
            }
        }
    }
```

This code sets every value to e.

Output Board

```
public static void OutputBoard(Char[,] Board)
        {
            for (int Y = 0; Y < Board.GetLength(0); Y++)//For all lines
            {
                Console.WriteLine("");//new line
                for (int X = 0; X < Board.GetLength(0); X++)//For all
horrizontal values
                {
                    Console.Write(Board[Y, X] + " ");//Outputs cell value
                }
            }
        }
```

This code outputs the board

Win conditions

```
    public static bool WinCondition (char[,] OpponentBoard)
    {
        foreach (char space in OpponentBoard)//for all spaces in board
        {
            if (space == 'j')//If there is a ship part
            {
                return false;//if there is return false
            }
        }
        return true;//Won if no ships
    }
```

This code checks WinConditions

Class

```
using System;
namespace Ships
{
   public class ship
    {
        public bool vertical;
        public int width;
        public int height;
        public int[] starting_coordinate = new int[2];
    }

}
```

Definition of ship class. As you can tell this is different from plan in two ways. Firstly, there is only 1 item instead of 4 separate ones. This was because it's more efficient to do it this way. Secondly, the hits array is missing because it was not necessary. This is because the values stored on there can just be seen from the board.

Evaluation

I feel that my code was partially successful. In terms of my success I think I achieved 3.5/5 criteria (Switch case,Working game,use of object orientated programming & half of efficiency).I feel that all of my code except switch case in the main method was efficient to a high enough degree. That is why I gave myself a half on that criteria. The main reason I did not achieve more of my targeted criteria was because I ran out of time creating my project.

The End