Title: Development of an Advanced Naval Battle Simulator

Introduction

The objective of this project was to develop an advanced simulator for naval warfare training. The project was divided into several stages, each introducing additional complexity and realism to the simulation. This report provides an overview of the work completed up to and including Part 1, Sections A, B, and C. This report provides a detailed breakdown of each component of the simulator, highlighting its functionality and implementation.

Menu Options

After initiating the program, the user is presented with a menu offering several options: initiating the simulation, viewing instructions, reviewing past statistics, or exiting the program. (performed by a switch statement). The menu options are managed through a switch statement

The "view instructions" option provides an initial overview of the simulation rules, followed by a detailed breakdown of the rules and conditions for each level. After reviewing the instructions, the user can go back to the main menu. The 'View Statistics' option allows the user to review the results of past simulations for each level, which are saved as text files

Simulation Initiation

The 'starting simulation' function will again ask the user if they want to view the details of the available escortships battleships in the program before starting the simulation. If the user chose to start the simulation they are presented with three levels to choose from, each corresponding to a different stage of the project. (performed by a switch statement)

1. Basic Training (includes Part 1-A)
2. Path of Brave (includes Part 1-B)
3. Advanced Warfare (includes Part 1-C)

The program handles invalid inputs properly, prompting the user to re-enter their selection if the input is not recognized.

Initial Conditions and Declarations

Both battleship and escortship properties will be stored in a structure with the data type Btype and Etype respectively Arrays are declared to store the properties of each type of ship, with predefined constants used to limit the array size. Dynamic memory allocation is used to allocate memory for these structures, and the array values are then copied to the structures.

For escortship in addition to the predefined properties minimum velocity, maximum velocity is randomly generated within the specified range for each type. The angle is also converted to radians to facilitate further calculations

```
E->min.Angle = ((rand () / (double)RAND_MAX) * (90 - range)) * PI / 180.0
```

The maximum angle is calculated by adding the range of each type to the randomly generated minimum angle, ensuring that both the minimum and maximum angles remain within the predefined range. Similarly, min velocity was also randomly generated between 0-50 and a conditional statement is used to check if the escort ship is of type A and generate the maximum velocity according to the predefined instructions. Finally, the launching escortship velocity is randomly generated using the maximum and minimum velocities as the range.

To calculate the attack range of the escortship which is shaped like an annulus. both min angle max angle along with the generated velocity was implemented to the projectile motion equation to calculate the range:

```
(velocity * velocity * sin (2 * angle)) / GRAVITY
```

This generates the maximum and minimum ranges of the escort ships, and by subtracting the maximum range from the minimum, the annulus-shaped attack range is created.

For battleship's the velocity was taken as a user input, the angle was randomly generated between 0-90

Canvas Creation

The program will ask the user to enter the upper x and y coordinates. allowing them to define the size of the battlefield within a maximum range error handling is implemented to ensure that the user doesn't exceed the maximum value of the canvas)

The program then prompts the user to input how many escort ships they want on the battlefield, and the program randomly generates coordinates with the x, y, and ship type in a structure, which is stored in a dynamically allocated array. Error handling is done to make sure the coordinates doesn't exceed the user input size of the canvas. The type and location of each escort ship are displayed so the user can place the battleship strategically.

Then the program will display the properties of each battleship type and ask the user to select one. (Again proper error handling is done to ensure the user doesn't input an invalid selection) Then the user is also asked to specify the coordinates for the placement of the battleship

The canvas is created as a two-dimensional array. To store the x and y coordinates respectively. But the array is accessed as a one-dimensional array throughout the program, with each coordinate assigned a unique index. This is done with the help of pointer arithmetic. '*' was used to denote battleship and escortship were symbolized with the notation given. The rest of the battle field (empty spaces of the canvas) was shown with ~ symbol

These initial conditions will be written to a file, so each time the simulation runs you can access the past battlefield conditions.

Projectile Calculations and Combat Outcomes

Using projectile motion equations, the attack range of the Battleship is calculated. It also calculates the distance from the battleship to each escort ship. If this distance is less than the attack range of any escort ship, the battleship is declared sunk and the program will display which escortship sank your battleship. if the battleship survived the program will similarly compare the battleship's attack range with the distance to each escort ship to determine if any escort ships are within range. if they are the program will count the number of escortship's that were hit and display their notation.

The properties of the escortships that sunk along with their velocities and attack ranges as well as the final state of the battlefield(canvas) is saved in 2 separate files.

Memory Management

Upon completion of the simulation, the program frees the memory allocated for the battlefield and the ships.

Part 1B - Simulation 1

Apart from the basic functions of part A, in Part B the battleship will move across the battlefield along a user generated number of points. The number of points are taken as a user input and error handling is done to ensure it's less than the (x*y number of coordinates) At each point, the program checks if the battleship has been sunk or if it has sunk any escort ships. The properties of any sunk ships, along with the final battlefield conditions, are saved to a file.

Part 1B - Simulation 2

Here after a certain number of iterations the battleship's firing angle is limited to between 0-60 so now the angle generated will drop from (0-90) to (0-60) However, as the range of a projectile remains the same for angles greater than 45 degrees, and since 60>45 this limitation does not affect the simulation.

Part 1C

Here the impact is also considered. The cumulative impact is calculated with each attack and an escort ship is sunk if the cumulative impact reaches 1. The user can choose to initiate this mode either by keeping the battleship still or by moving it across generated points. The functions used in the initial simulation are modified with a Boolean parameter named `consider Impact`, which is set to false for the previous simulations. The cumulative impact is also set to 0 for the previous two simulations and the calculated amount is sent in the last simulation.

Challenges and Solutions

One of the main challenges faced during the development of the simulator was handling the various random elements such as the positions of the ships, the type of battleship and allocating memory for them methodically. Ensuring that the user input values doesn't exceed the maximum values of the program to avoid it crashing. This was addressed by implementing robust error checking and validation to ensure that the generated values were within the acceptable ranges.

Additionally, Calculating the attack range for escortship was somewhat of a struggle as well. Moreover, adapting predefined functions to incorporate impact calculations within simulations without the need for separate functions to be declared presented further complexities Accurate parameter passing was crucial to ensure that simulations 1 and 2 excluded impact considerations and simulation 3 did included them (which was explained previously in detail)

Conclusion

The development of the Advanced Naval Battle Simulator has been a challenging but rewarding experience. The simulator is now capable of mimicking realistic naval warfare scenarios with 3 different levels that gets harder with each simulation. Future work will involve adding more complex features to the simulator like making sure pre generated values aren't taken into account. Limit certain capabilities (such as the number of ships it can attack) of the battleship after a given time period to make it even more realistic