Advanced Naval Battle Simulator

De Zoysa T S D

Index No - IT23669062

Sri Lanka Institute of Technology

Programming Methodology - SE1012

Abstract

Advanced Naval Battle Simulator is a serious tool that helps naval personnel practice and understand the strategies of naval warfare. This report presents an in-depth analysis of an advanced naval simulator design and coding challenges that I faced when I was coding this simulator using c programming language.

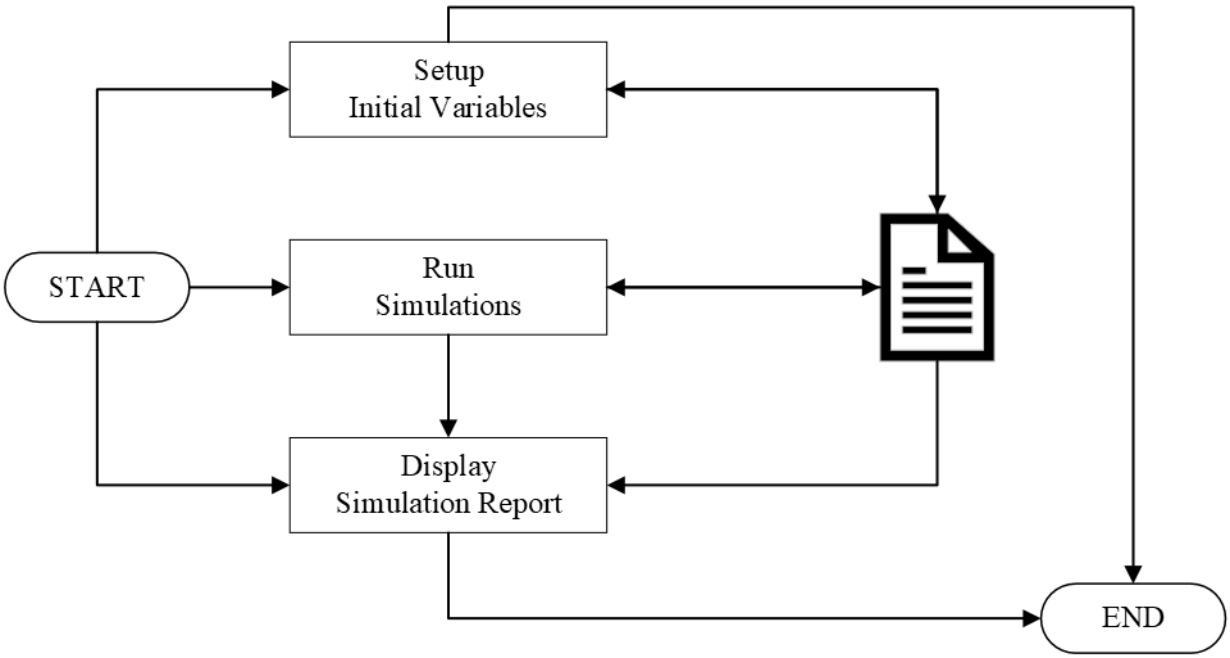
The development of the simulator is broken into different smaller parts. Firstly I added simulations containing much simpler features and gradually added complex features. This simulator has 4 different battleships and 5 different types of escort ships which are used in the real world in World War 2. At any given time, there is only one battleship on the naval battlefield and many escort ships.

In this simulator, there is only one gun in each ship. When a gun fires, the shell follows the typical parabolic projectile motion trajectory under the effect of gravity. Calculating that projectile motion each ship has its range. If a ship is on another's attack range they can impact that ship.

Design

High-level Overview

The diagram briefly explains how users can initialize variables for battalion ships and escort ships. Further explains how the simulation and simulation report display.



Setup Initial Variables

Initializing the 2D Canvas

To initialize the squared 2D Canvas user should input (d) the height of the plane. The system will determine it as x and y coordinates (d, d), which represent the upper right corner of the 2D plane. Lower left corner coordinates were predefined as (0, 0). Canvas size is saved to the system file “src/log/user_input.dat”.

Initializing the BattalianShip

First System will load the battalion ship types file from “src/resource/battalian_types.dat”. Then display it in tabular format. Users will pick the battalion ship type from the index and then set up the battalion's position, max velocity, and reloading time. The system validates the user inputs and then battalion ship Initial values are saved to a separate file “src/log/initial_condition_battalian.dat” using the CSV format.

Initializing the EscortShips

Here also escort ship types are loaded from the file from “src/resource/escorts_types.dat”. The system will prompt the user to input the escort ship count. Then it generates a random integer value between escort ship types. The escort ship type system generates random values for minimum angle, minimum velocity, and maximum velocity. Moreover, the maximum angle of the escort ship is determined by adding the escort ship type angle range to the randomly generated minimum value. Finally, it also saved to a separate file “src/log/initial_condition_escort.dat” using CSV format.

Seed Value

In c programming language generating a random value needs a unique integer number for each start-up to generate random numbers on each start-up. Unless it will be the same number as all the start-ups. In my research, most of the time developers use the raw time from the machine as the unique identifier. In this project, I enable the user to either enter an integer as the seed value or enter a value for it. However, if the user doesn’t set the seed value, the system will automatically use the time as the seed value.

Running Simulations

In this project, the System will simulate each scenario described in the assignment accordingly.

Part 2 A Simulation Design

In this project, I encountered a strategy for problem part 2 A, which is to rearrange the battalion ship's attack order according to the time it takes to attack the escort ship.

Time = Shell Travel Time + Reloading Time

The battalion ship will attack the escort ship which takes the shortest time to attack. The following figure contains the implemented code.

```
bool baseSimulation(InitialConditionsBattalian battalian, InitialConditionsEscort * escort_ships, float * health, int escort_count)
{
    if (!escortSimulation(escort_count, escort_ships, battalian.position, health))
    {
        int in_range_count = 0;
        Time * escorts_time = battalianAttackTime(&in_range_count, &escort_count, battalian, escort_ships);

        //Sorting the time in acending order
        bubbleSort(in_range_count, escorts_time, escort_ships);

        //elemenate it one by one
        while (in_range_count > 0)
        {
            eleminateEscort(&in_range_count, escorts_time,in_range_count - 1);
        }

        free(escorts_time);
        return true;
    }else{
        return false;
    }
}
```

Part 2 B Simulation Design

For the part 2 B problem, my strategy is to make the attack order according to the attack time of each ship

Attack time = Reloading Time + Shell Traveling Time

Moreover, hence the escort ships can attack multiple times battalion ship stays in one position until it sinks all the escort ships. If an escort ship's attacking time is more than the battalion ship's attacking time it, it will get attacked. If not escort ship will attack the battalion ship. In my implementation, I failed to implement it fully.

Simulation Report

At the end of each simulation system will display the simulation results. It includes the Escorts and Battalion ship initial values, battalion health, and the attack order of the battalion ship. The following figure shows an example of a simulation report.

```
----------------------
Simulation Statistics
----------------------

Canvas Size      :-  width: 100 height: 100
EscortShip Count :- 20

BattalianShip Initials Values
-------------------------------------
|Notation  |Position   |Max Velocity |
-------------------------------------
|U         |50    50   |10.540000 |
-------------------------------------

BattalianShip  Health Record
-------------------------------------------------------------------
|Position   |Status     |Cumilative Impact   |EscortShip Index    |
-------------------------------------------------------------------
|50    50   |1          |100.00              |-                   |
|60    49   |1          |96.00               |10                  |
|62    49   |1          |92.00               |10                  |
|82    49   |1          |87.00               |15                  |
|84    49   |1          |82.00               |15                  |
-------------------------------------------------------------------

EscortShips Initial Values
-----------------------------------------------------------------------------------------------------------
|Index     |Position            |Max Velocity   |Min Velocity   |Max Angle |Min Angle |type      |
-----------------------------------------------------------------------------------------------------------
|0         |88         66       |4.49           |4.30           |22.44     |2.44      |Ea        |
|1         |67         22       |8.46           |1.46           |90.00     |84.74     |Ea        |
|2         |60         91       |11.67          |1.25           |90.00     |61.62     |Ed        |
|3         |58         67       |9.70           |1.55           |90.00     |87.17     |Ec        |
|4         |77         69       |9.14           |4.39           |85.90     |60.90     |Ec        |
|5         |38         69       |10.04          |2.15           |90.00     |22.43     |Ee        |
|6         |40         56       |18.63          |13.77          |48.14     |28.14     |Ea        |
|7         |42         15       |13.29          |5.32           |46.52     |16.52     |Eb        |
|8         |70         71       |2.63           |0.58           |87.95     |57.95     |Eb        |
|9         |0          27       |15.32          |6.22           |90.00     |23.07     |Ee        |
|10        |77         49       |14.36          |6.18           |90.00     |46.17     |Ee        |
|11        |15         4        |0.50           |0.30           |90.00     |60.43     |Ee        |
|12        |83         55       |8.72           |7.37           |90.00     |84.80     |Ec        |
|13        |62         8        |1.06           |0.26           |90.00     |85.06     |Ed        |
|14        |79         27       |10.10          |0.25           |81.56     |31.56     |Ed        |
|15        |95         45       |22.08          |0.84           |90.00     |67.86     |Ed        |
|16        |53         68       |9.59           |9.11           |90.00     |23.56     |Ee        |
|17        |67         95       |14.22          |9.54           |81.45     |31.45     |Ed        |
|18        |13         100      |8.17           |2.35           |84.53     |64.53     |Ea        |
|19        |31         56       |4.20           |1.81           |32.29     |7.29      |Ec        |
-----------------------------------------------------------------------------------------------------------

EscortShips Attack Order
-------------------------------------------
|EscortShip Index     |Time to Hit          |
-------------------------------------------
|10                   |1.32                 |
|12                   |1.27                 |
|15                   |1.26                 |
-------------------------------------------
Attacked Escort Ship Count :- 3
Total Time to end the battle :- 3.85 s
```

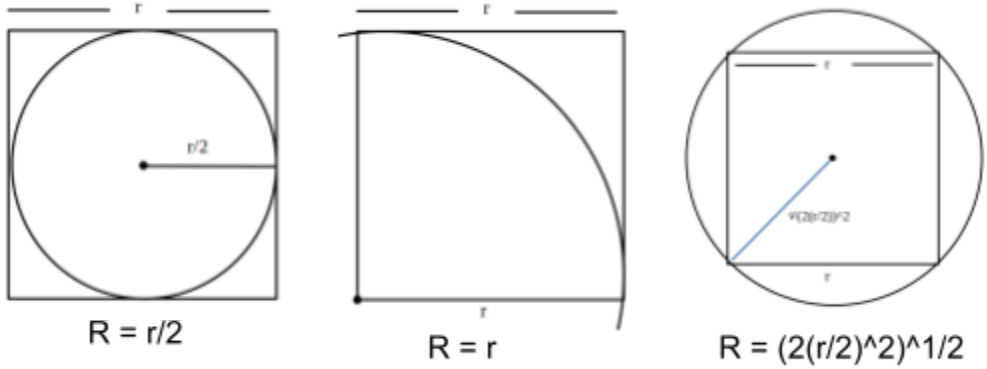
Discussion

This chapter will discuss the coding challenges I encountered when programming this simulation and how I overcame them.

Optimizing Velocity

Finding the best velocity range is one of the most hard to overcome challenges I found in this project. Because when we define a constant range, it will not work for all canvas sizes. For example, if we define the velocity range can vary between 0 to 100 meters per second, this gives us a maximum range of around 1020.4 meters. This range is ideal for canvas sizes 1000 by 1000, but not for 100 by 100. Because ships like battalion ships can attack the whole canvas even their position in a corner of the canvas.

Implementing a dynamic velocity range according to the canvas size can be one solution to overcome this problem. I encounter some strategies in my research that can apply. The below diagrams briefly explain what they are.



In this research, among these three methods, I got a fair result every time for the first diagram shown above. Ships maximum velocity of a ship should be less than the length of the center to the canvas side. The following figure shows the implementation.

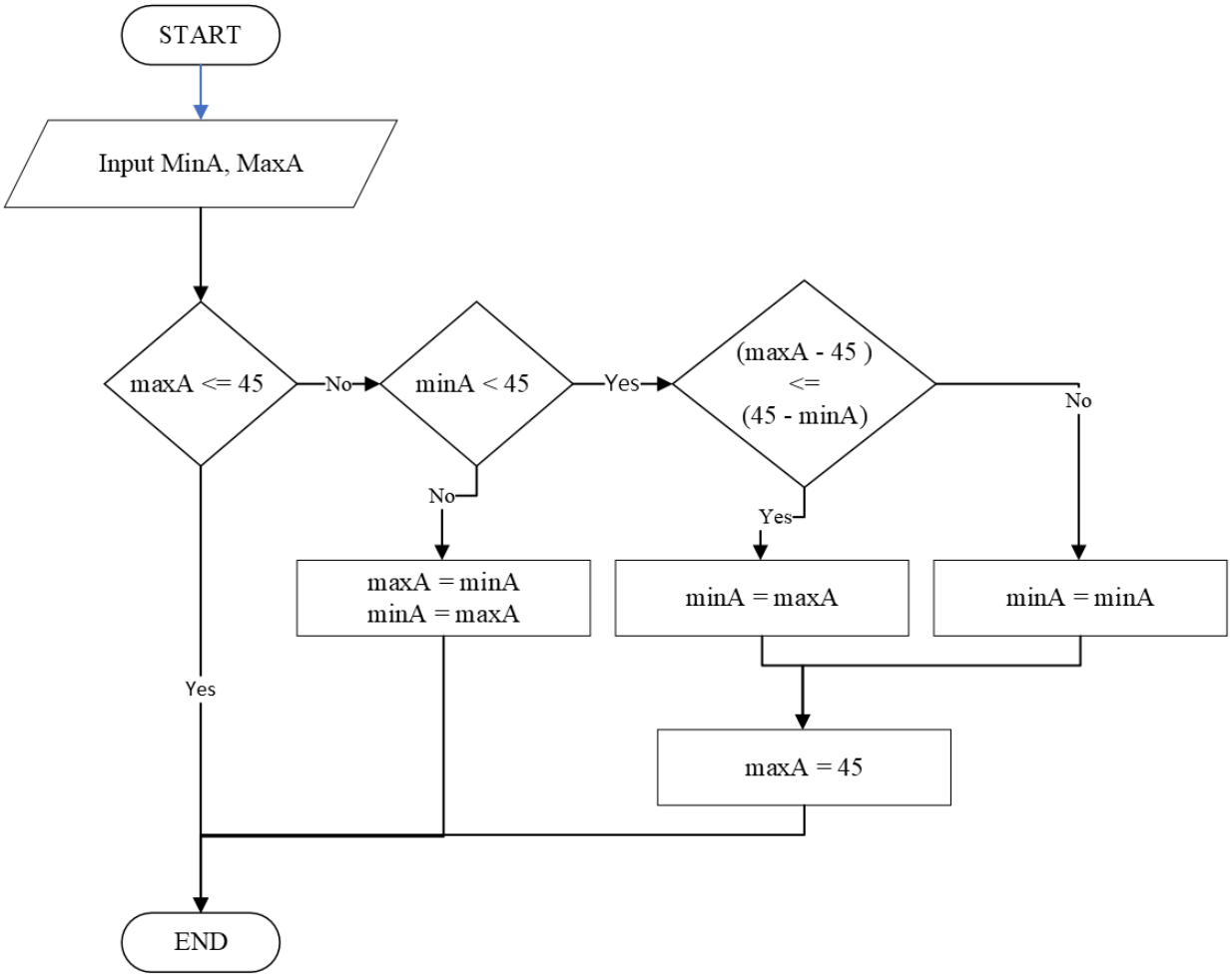
```
//Finding the best maximum velcity according to the canvas size
float max_velocity = sqrt((canvas_size.x / 2.0)* GRAVITY);
```

Optimizing Projectile Motion

To calculate the attack range of the ship we need to find the max range and the min range. To find the max range and the min range we need to find the angle which delivers the maximum and the minimum ranges. In projectile motion, a 45-degree angle gives the maximum range to launch the projectile. But when initializing system saves the ship's maximum and minimum angles.

To overcome this problem I encountered two strategies.

- One is to brute-force every angle in the range and find the max range given angle and min range given angle.
- The second one is a logical algorithm which is shown in the following flow chart.



The logical method needs less computer power than the brute-force method. The following figure shows an implementation of this algorithm in c.

```c
/**
 * Check the Maximum and Minimum Range Angle
*/
void findMaxMinAngle(float * minA, float * maxA)
{
    if (* maxA <= 45){}
    else {
        if (* minA < 45)
        {
            if ((* maxA - 45) < (45 - * minA))
            {
                * minA = *maxA;
            }
            * maxA = 45.0;
        }else{
            float temp = * maxA;
            * maxA = * minA;
            * minA = temp;
        }
    }

}
```

Navigating the Maze

In Pat1-B simulation1 describes the ship moving to a random position in the 2d canvas. Finding a random destination can done by generating random x and y values inside the canvas.

To find the path we need a strategy to implement. Here also in my research, I found two strategies that can be used. There are,

- L - shape method
- Diagonal method

Both methods are seamlessly work on the project. But the only problem is diagonal method uses fewer points than the L-shape method when the destination is not on the same axis. In my project, I use the diagonal method. The following figure shows the implementation of the diagonal method.

```
/**
 * Change Position
*/

Coordinates changeCoordinate(Coordinates current, Coordinates destination)
{
    if (current.y > destination.y)
    {
        current.y --;
    }
    else if(current.y < destination.y)
    {
        current.y ++;
    }

    if (current.x > destination.x)
    {
            current.x --;
    }
    else if(current.x < destination.x)
    {
            current.x ++;
    }

    return current;
}
```

NP-hard Combinatorial Optimization Problem

NP is a class of computational decision problems. NP stands for “Non Polynomial”. These problems give a solution but not in a polynomial time. Np-hard Combinatorial optimization problem is a sub-class of NP that has the hardest problems. For example, tic-tak-toe is a Np-hard problem because there is no specific strategy to solve the problem. There are games, simulations, traffic management systems, and encryption algorithms that have NP-hard problems to solve. This project also has an Np-hard problem. Therefore there is no optimal solution.

Optimizing Firing Order

Hence this is an Np-hard problem there is no optimal method to run this simulation to maximize damage on escort ships while minimizing the impact on battalian ships. I developed a strategy that the battalion ship would attack the escort ship which takes the shortest time to attack.

To sort the ships in the shortest to longest order I used the bubble sort algorithm. The bubble sort algorithm takes two values from the array and compares them and re-arranges them in shortest to the left and longest to the right. The following figure shows the implementation of bubble sort.

```
// Function to swap two elements
void swapT(Time *x, Time *y) {
    Time temp = *x;
    *x = *y;
    *y = temp;
}
void swapE(InitialConditionsEscort *x, InitialConditionsEscort *y) {
    InitialConditionsEscort temp = *x;
    *x = *y;
    *y = temp;
}


void bubbleSort(int escort_count, Time * escort_time, InitialConditionsEscort * escort)
{
    for (int i = 0; i < escort_count - 1; i++) {
        for (int j = 0; j < escort_count - i - 1; j++) {
            if (escort_time[j].time > escort_time[j + 1].time) {
                swapT(&escort_time[j], & escort_time[j+1]);
                swapE(&escort[j], &escort[j + 1]);
            }
        }
    }
}
```

Conclusion

In this project, we develop an advanced naval simulator that helps naval personnel practice and understand the strategies of naval warfare. Currently, this system runs only in the terminal. In future development, it can develop into to much more graphical user interface. That will help personnel to understand the strategies clearly.

Open Source project of this report can be found in the following GitHub repository:

https://github.com/sathindud/Advanced-Naval-Battle-Simulator

References

- Generating random value in c
  https://www.geeksforgeeks.org/generating-random-number-range-c/

- NP-hard Combinatorial Optimization Problem
  https://www.youtube.com/watch?v=YX40hbAHx3s

- Bubble Sort – Data Structure and Algorithm Tutorials
  https://www.geeksforgeeks.org/bubble-sort/