VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Computer Architecture CO2007

Assignment

BATTLESHIP

Advisor : Pham Quoc Cuong

Students : Nguyen Xuan Son - 22252717

HO CHI MINH CITY, November 30th 2023

Contents

1. Introduction

Battleship is a strategic board game between 2 players. Both players have a map of a 7x7 grid, they initially set up a fleet of battleships of different sizes, in this game are 3 **2x1** ships, 2 **3x1** ships, 1 **4x1** ships. In each round, players alternatively target a box on the other player's map blindly. A player will lose if all of their ships got destroyed first

I implemented the code in MARS MIPS simulator. The goal of this document is to explain about my idea and the building process.

2. Main idea and Game mechanics

2.1. Interface representation

- Here I use the Bitmap Display tool provided by *MARS 4.5* for ease of observation, to set up this feature, first go to **Tools → Bitmap Display**. After that, Bitmap Display is set up in MARS at:

  Unit Width in Pixels = 32

  Unit Height in Pixels = 32

  Display Width in Pixels = 512

  Display Height in Pixels = 512

  Base address for display = 0x10008000($gp)

Select **Connect to MIPS**, then Press  (F3) →  (F5) to start. A complete Set up is shown by the figure:

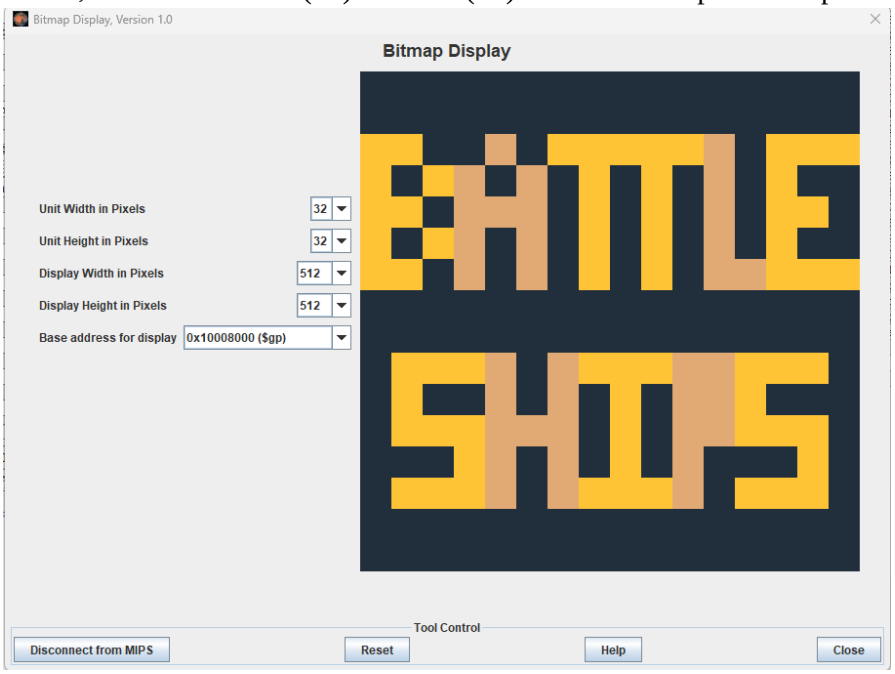

Figure 1: Welcome screen

2.2. Describe main idea by flow chart

2.2.1. Set up phase

- The game first draws a 7x7 board with coordinates 0–6 for x-axis and 0–6 for y axis. For the ease of access of players from beginner to proffessional level, I choose to display on the prompt the rules of the game, and then guide the current user step by step placing their fleets of battle ships. The idea is quite simple as the flowchart show below:

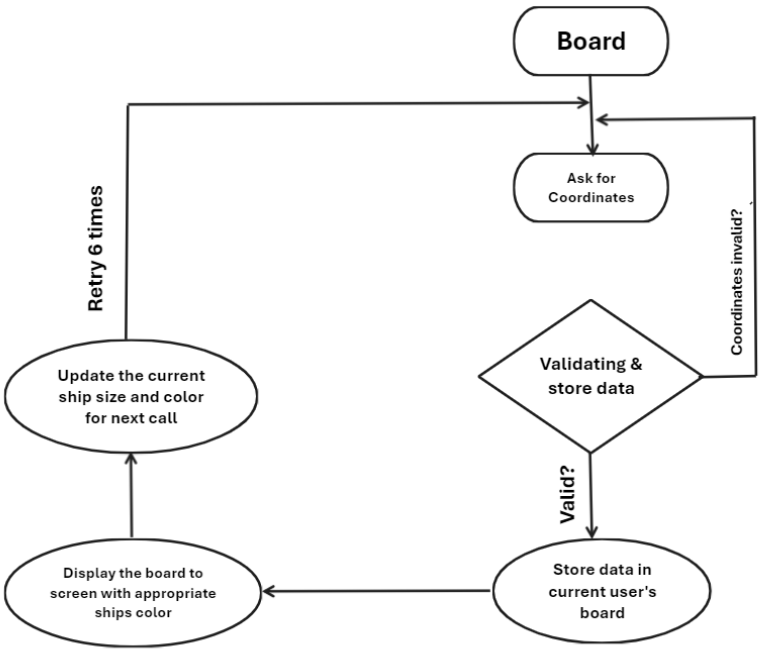

Figure 2: Flowchart of set up phase

- After coordinates are entered successfully, the program then fills the board with an appropriate ship's color so that user can track their fleets easily. Below is an example of a successful ship placement:

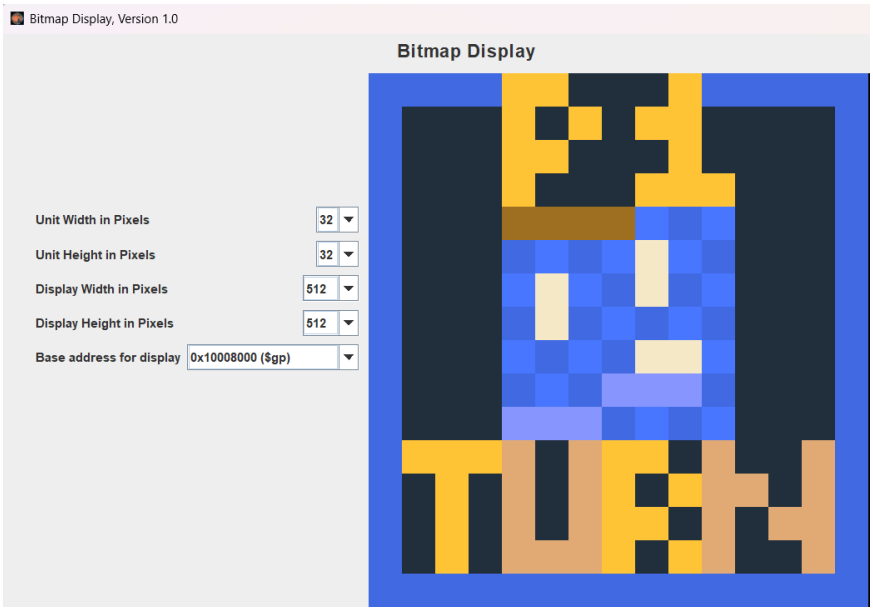

Figure 3: Turn indicator

2.2.2. Coordinate validating & Store data

- One of the most important features that has a huge impact on User Experience (UX) is Data Validation, as it makes the game fresh and minimizes bugs. For that reason, the process of checking whether a coordinate is valid or not must be implement carefully. In this game, users have to enter the value in correct format (row_{bow}, $column_{bow}$, row_{stern}, $column_{stern}$) and the allignment of the ships must be parrallel to vertical and horizontal axis. They also have to enter the correct size of the ship according to the intructions.

- My idea is quite simple as according to the above condition, the ship coordinates must satisfy one of two conditions:

$$\text{Coordinate: } \begin{cases} \boldsymbol{x_1 = x_2, y_1 < y_2} \\ \boldsymbol{x_1 < x_2, y_1 = y_2} \end{cases} \qquad \boldsymbol{otherwise}$$

- After that, the coordinates must satisfy the following equation: $\boldsymbol{cursize = |x_1 - x_2| + |y_1 - y_2|}$ (known as Mathathan distance).

- To check for duplication, the process is quite simple as all I have to do is simulate that ship and just check.

- Below is the idea represented by flowchart:

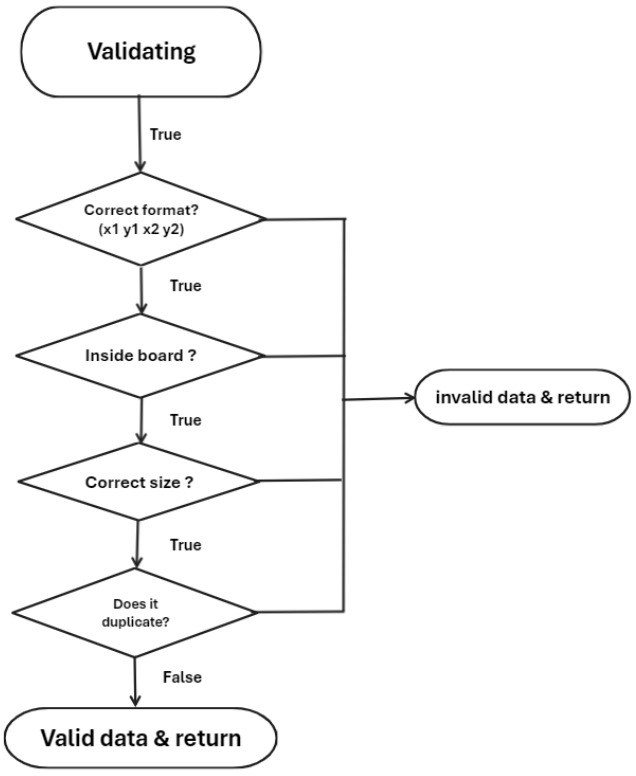

Figure 4: Data checking

- When storing data, I first try to loop from the begin location to the end location following the above condition, and in each step I apply my decode function to the current coordinate into address and store it into the user's matrix. The method of decoding is followed by this fomula:

$$\boldsymbol{address = Base\ address + (Row\ index * N + Collumn\ index) * 4}$$

here N is the number of collumn in a row, in this case, $N = 7$

2.2.3. Battle phase

- After all the fleets have been placed, the program will move to next phase – Battle phase. In this game, I divided the screen into two boards for the purpose that two players will have a clear visualization of what they are doing. Also, in order to ensure the equity of the game, all user's board will be hidden. It means that they are blindly attack each other. To make it clear, hear is the interface of the program in battle phase:

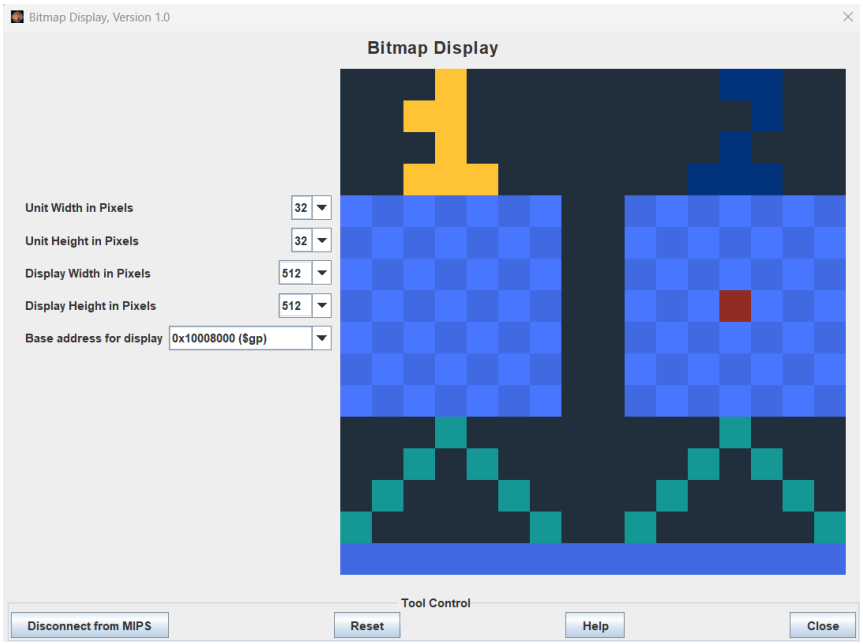

Figure 5: Battle phase UI

2.2.4. Seting up fire location & updating current score

- In this phase, two players will aternaltively choose their fire location to shoot the other's fleet and gain a score for each successful attack. Of course, to avoid unfairness for the later player, a round is calculated only when the two players completed their turn. It will avoid the case when there is only one box left in both players' board unshot.

- Players also have to enter the correct form of input. In this phase, only one coordinate should be entered, which is (x_{fire}, y_{fire}). The idea can be decribed by Figure [5] by flowchart.

- When it comes to fire location validation, the procedure is similar to placing battleships, but instead of filling the map with the coordinates, what we have to do is checking the opponent's board and define if the target is *miss* or *hit*, or check if we duplicated the fire location. The idea can be explained by Firgure [6].

- Let's talk about the Scoring system, here I noticed that the special property of this game is that each player has three **2x1** ships, two **3x1** ships and one **4x1** ship, which means that each of them has to shoot a total of 16 boxes in order to win the match. It leads to the idea of the one who score 16 points first will be the winner.

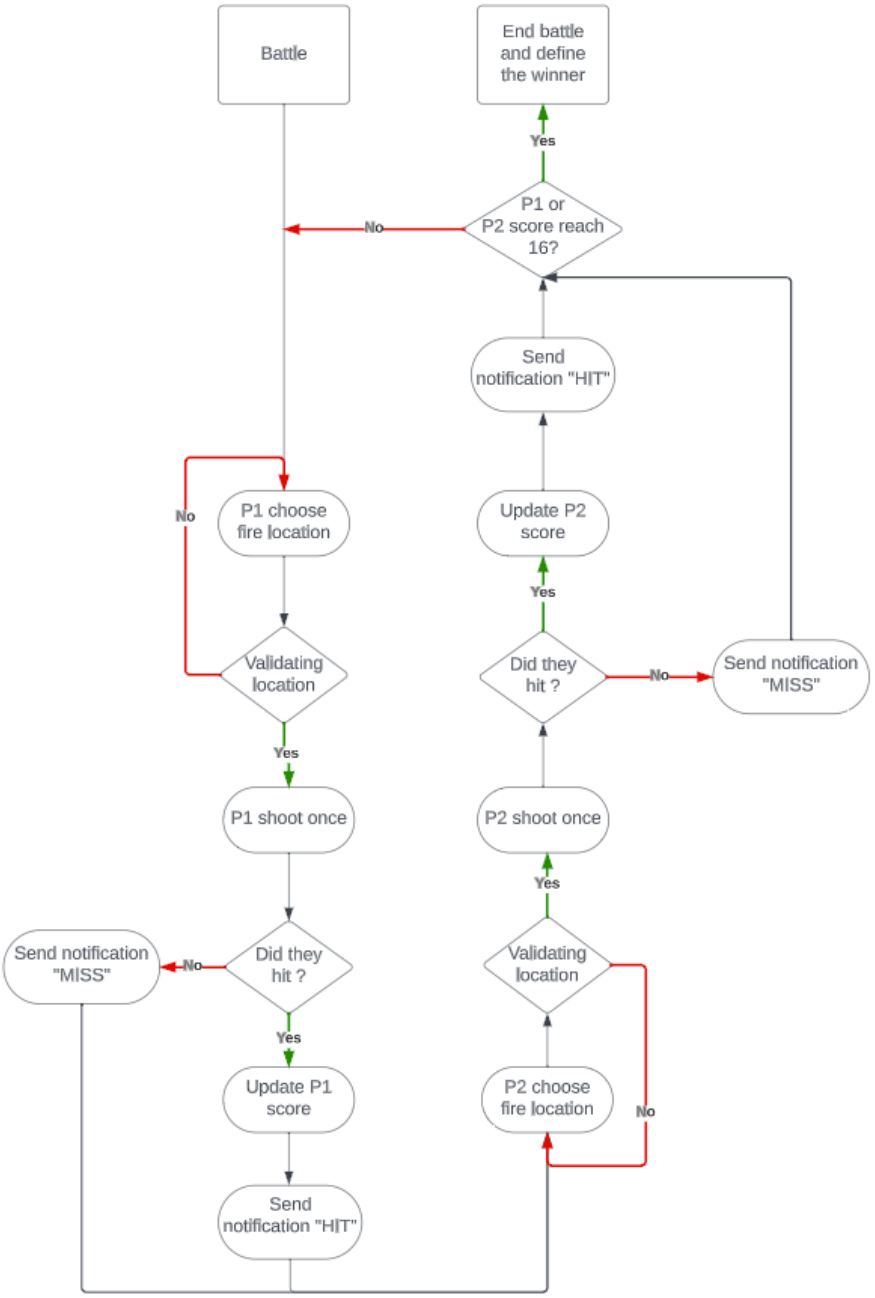

Figure 6: Gameplay flowchart

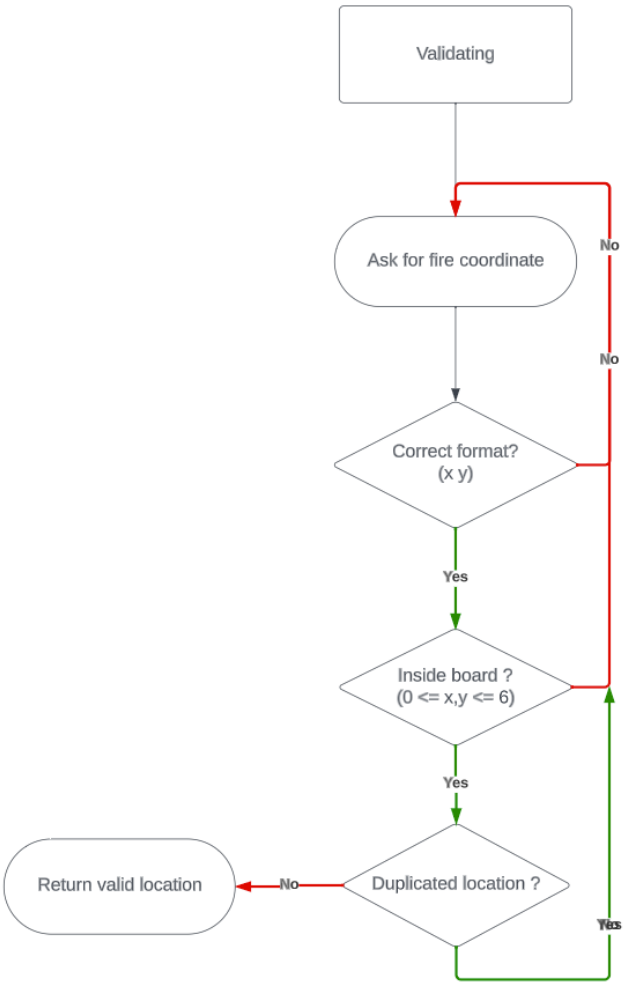

Figure 7: Data validation flowchart

- When the player chooses an area to attack, depending on the opponent's map, the program will notify whether the player has successfully shot that warship or not. If they were successful, the game will send a notification “Hit!” to terminal and color the box green, otherwise it will send “Miss!” and color the box red.

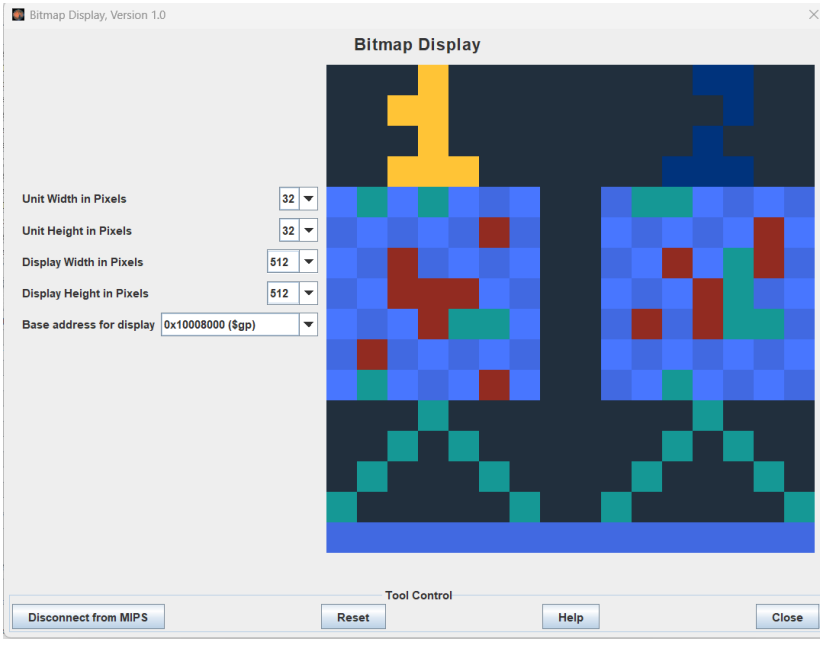

Figure 8: An example of the match in progress

2.2.5. Determining the winner

- When one of the two players has score 16 point first (or both of them) and a round is completed (when Player 2 finish his final move), the program will terminate and calculate who will be the winner.

- It leads to 3 cases when the game is terminated: $\begin{cases} Point_{P1} > Point_{P2} \\ Point_{P1} < Point_{P2} \\ Point_{P1} = Point_{P2} \end{cases}$. It can be observed that in the first two case, the winner can be determined by ***Player 1*** **wins** (Case 1), or ***Player 2*** **wins** (Case 2). The third Case can be considered **Draw** as two players simultaneously defeat the other.

3. Extention and notification

In this part, I will introduce some interface and extension of my game.

3.1. Guiding users to the game rules

- To make the game more user – friendly, I display some introductions and rules to make it less confusing at first glance so that the users can understand it easily.

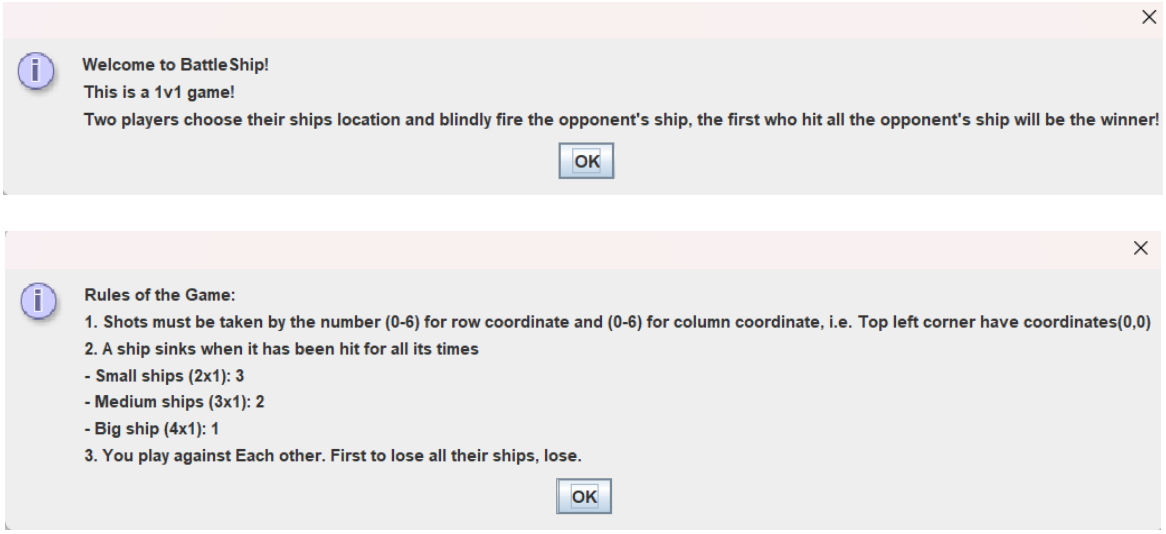

Figure 9: Game introduction

3.2. Guiding user about information of the ship they're about to place

- In this game, I won't make user randomly choose their coordinates, instead, I will guide them *step by step*, as they must input the ship in the order: 3 **2x1** ships, 2 **3x1** ships, 1 **4x1** ship. Not only does it make it accessible to beginners, but it also makes the game more coherent.

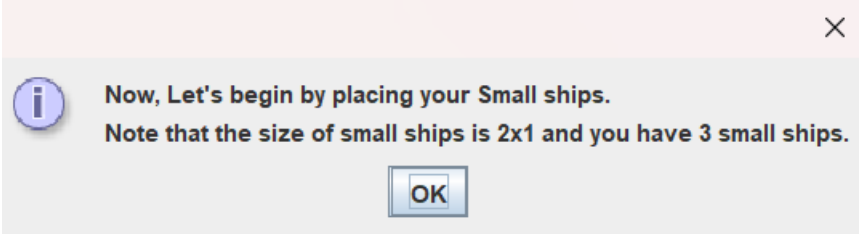

Figure 10: An example of clear explaination of the game

3.3. Error detection

- In set up phase and in battle phase, whenever coordinates input do not match the validating criteria, a prompt will display and let the user knows that they had a mistake. It then let the user try again until the conditions match.

- In battle phase, if user duplicates their attack, another error prompt will send a prompt that he/she has already shot this location before.

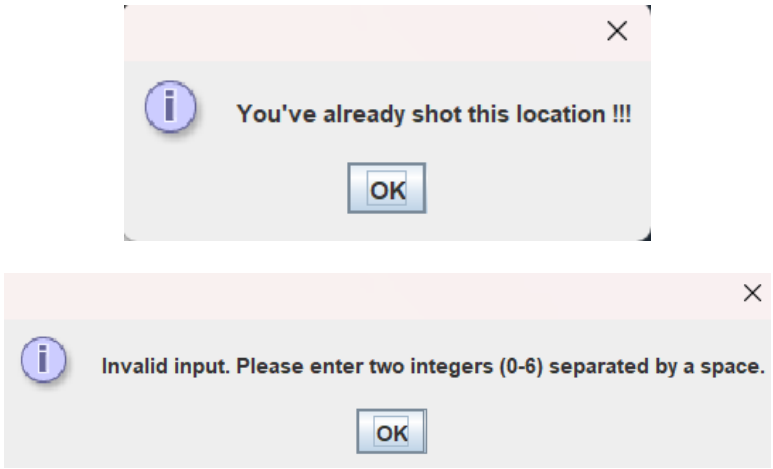

Figure 11: Error notifications

3.4. Turn indicator

To make the progress more easily to keep track of, I design my one turn indicator for both set up phase and battle phase.

- In set up phase, the players place their fleets of battle ships, they can see the full map of their ships with color brown for the big ship, purple for medium ship, and light yellow for small ship. You can see in this in *Figure 3*.

- In battle phase, when the players take turn to attack each other, There should be something to make the everyone knows who's turn is in progress, so I decide to make the number of the current player brighter and the other is darker. You can see in this in *Figure 8*.

3.5. Move Recording

- For ease of review, I also write out the moves of each player into a separated text file using I/O operations provided by MARS 4.5 MIPS. The file will be of the name ***"game_log.txt"*** in the current project files. It will look something like this:

```
Setup Phase:
--------------------------------------------------------------------------------
P1:
1 4 2 4
2 1 3 1
4 4 4 5

5 3 5 5
6 0 6 2

0 0 0 3

P2:
1 4 2 4
4 4 4 5
2 1 3 1

5 3 5 5
6 0 6 2

0 0 0 3
--------------------------------------------------------------------------------
Battle Phase:
--------------------------------------------------------------------------------
P1: 1 3 MISS!
P2: 2 6 MISS!
--------------------------------------------------------------------------------
P1: 3 3 MISS!
P2: 4 4 HIT!
--------------------------------------------------------------------------------
P1: 0 3 HIT!
P2: 5 3 HIT!
--------------------------------------------------------------------------------
P1: 6 1 HIT!
P2: 2 1 HIT!
--------------------------------------------------------------------------------
P1: 4 5 HIT!
P2: 3 3 MISS!
--------------------------------------------------------------------------------
P1: 3 2 MISS!
P2: 2 4 HIT!
--------------------------------------------------------------------------------
P1: 5 3 HIT!
P2: 4 1 MISS!
--------------------------------------------------------------------------------
P1: 5 4 HIT!
P2: 5 4 HIT!
--------------------------------------------------------------------------------
```

Figure 12: Move record for further review

4. Try an example

Link of demo is here

5. Source code

Link of source code is here