VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Computer Architecture Lab (CO2008)

Assignment (Semester: 231, Duration: 06 weeks)

Battle Ship
Report

Students: Vũ Minh Quân - 2212828

HO CHI MINH CITY, OCTOBER 2023

Contents

1 Game Requirements and Design

1.1 Gameplay

In this project, we will replicate the battleship game in **MIPS assembly** language. Since the resource is limited and to keep things simple, we will work with a smaller grid size which is 7x7. So first of all, initialize the two grid for each player with all 0s. Each player needs to enter enough ships require to start the game. There will be **3 ships size 2x1**, **2 ships size 3x1**, **1 ship size 4x1**. After player A has input all the ships, player B will do the same. After the intial phase, the game will start by shooting at each other's ships. The winner is the one who destroy all the opponent's ships first.

1.2 Read a ship

Here is the detail input of the game:

1. First of all, the game will ask the user to play or not by typing **Y/y** for yes or **N/n** for no respectively.
2. Then, it will ask the type of input that the user want to implement: **1** for manual input or **2** for file input.
3. If the user choose **1**, the game will ask the user to input the ship's position, else if the user choose **2**, the game will run through all the file.
4. Whenever the user's input has the correct form as the author's format and the correct logic as the Assignment mentioned, adding the ship to the pegboard by replace index **0** with index **1**.
5. The user's input must follow the rules: "**Row1 Column1 Row2 Column2**", whereas **Row2** >= **Row1** and **Column2** >= **Column1** (e.g. **2 2 1 4**), to consider to be correct form.
6. The user's input must follow the rules: "**exactly 1 couple (Row = Row or Col = Col)**" (e.g. "**1 2 1 4**"; "**2 1 2 3**"), to consider to be correct logic.
7. The ship input meets the reqirements, but the ship's position is overlapped with the other ship, consider to be incorrect logic.

1.3 Read a shot

1. The game will ask the user to input the shooting's position.
2. The user's input must follow the rules: "**Row Column**" (e.g. "**2 2**"), to consider to be correct form.
3. The shooting input meets the reqirements, but the shooting's position is overlapped with the other shooting, consider to be incorrect logic.

1.4 Constraints and Error Handling

1. If the input is not in the correct form (e.g. "**1210101**"; "**21 2 3** "), the game will throw an Error message " **- Wrong Syntax!**" and ask the user to input again.
2. If the input's value is out of range (e.g. "**8 2 7 4**"; "**-1 0 -4 0**"), the game will throw an Error message " **- Out of range!**" and ask the user to input again.
3. If the input's value is overlapped with the other ship, the game will throw an Error message " **- Overlapped Ship!**" and ask the user to input again.
4. If the number of ships in a type exceeds the number of ships in the rule, the game will throw an Error message " **- Full slot for ship type ...!**" and ask the user to input again.
5. If the shoot is not in the correct form (e.g. "**121**"; " **01**"), the game will throw an Error message " **- wrong shoot syntax**" and ask the user to input again.
6. If the shoot is out of range (e.g. "**1 8**"; "**-1 4**"), the game will throw an Error message " **- wrong shoot size**" and ask the user to input again.
7. Beside that, the output file will throw additional Error message " **incorrect shoot syntax** " or " **shoot out of range**" respectively.

2 Code Structure and Additional Features

2.1 Code Structure

I have divided the game into 5 main parts: **User interface**, **Read Input**, **Security**, **Print Out**, **Start Game** for easy checking and fixing bugs. This also boost the code progress some how!

1. The **User interface** is the part that interact with the user by asking the user to input the ship's position or the shooting's position. It also ask the user to choose between manual input or file input and to play or not.
2. The **Read Input** shall read the input from the user or the file then call the **Security** part.
3. The **Security** part will check the validation of the input, if the input is valid, process it, else recall the **Read Input** part.
4. After we have enough ships, the game will call **Print Out** part to print the pegboard.
5. The **Start Game** will then start the game by taking turn shooting each others ship, and with every shot, the game will save it to the output file.

Now, with the shoot input, we just need to check the range, the syntax the same as the ship input. If the input is valid, then pointing to that index cell in the pegboard. If that cell is 0, throw **Miss**, else throw **Hit** and continue.

2.2 Input and output file

I let the 2 files **input.txt** and **output.txt** in the same directory with the **.asm** file and put them right after the **.data** section, please create 2 files and replace their directory the same way as I did. Please open the 2 files in **MARS** for the greatest experience. In the input file, each line is a ship or a shoot with no extra space after the last column are made. If you have not entered enough ship (12 ships total) then every shoot after that will consider to be a wrong ship. If the input file ended with not enough ship, in the terminal, it will throw **Not Enough Ships to encounter, please refill !**. If there are not enough shot to find the winner, in the terminal, it will throw **Not Enough Shoots, no winner !**. The output file contains a **round counter**, **A turn** or **B turn**, **shot** and **status** of each shot.

The general between user input and file input

The same idea and code implement is that read a string of ship's position, then split it into 4 parts, then convert it into integer, then check the validation of the input. If the input is valid, then add the ship to the pegboard. Now, how to check the validation of the ship input?

1. Firstly, check all the character in odd position, if it's not a space ascii, then throw an error message.
2. Secondly, check all the character in even position, if it's not a digit ascii or exceeding the range ($\mathbf{0} - \mathbf{6}$), then throw an error message.
3. Thirdly, if there is not exactly a couple of (Row = Row or Col = Col) which means the ship is not a straight object, then throw an error message.

4. Then iterate throught the pegboard to check if the ship's position is overlapped with the other ship, if it is, then throw an error message.

5. If the input gets here, that means the input is valid, then add the ship to the pegboard

The different between user input and file input

The user input is read from the keyboard, so you can easily follow steps by steps and see the errors or correct result by real time. Whereas the file input only return the result after all. For user input, we declare a string before and store input in that string. With the file input, we read the file and put in into string direclty. But after processing a ship, in user input it will ask the user to input again, but in file input, it will continue to the next ship.

2.3 Additional Features

Here are some bonus features that I have done so far:

1. Bonus 1: I moderated the user interface (UI) as friendly as possible and add the additional **file input** features for easy testing.

2. Bonus 2: I added the **menu** for quitting, reseting or reading instruction during the game play.

3. Bonus 3: I let the string for reading in shipssize 7, and the shoot size 3 so when it gets the last column, it will **automatically enter for you** so you do not need further press "Enter button".

4. Bonus 4: Instead of replace 0s with 1s, I have used **size of the ship type** to add to the pegboard.

5. Bonus 5: I added the **remaining ship number of each ship type** for easily tracking and input the desired ship type.

6. Bonus 6: I also added the **output file** as well as the **4 status of a shoot**, not only show hits but also show misses, wrong syntax and wrong size.