VIETNAM NATIONAL UNIVERSITY

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL MODELLING (CO2011)

Assignment:

BATTLESHIP

Class: CC07

Instructor: Phạm Quốc Cường

Băng Ngọc Bảo Tâm

Student: Đoàn Anh Quang-2252666

Hồ Chí Minh City – 2023

Contents

I. Introduction:

1.1 MIPS and MARS stimulator:

MIPS (Microprocessor without Interlocked Pipeline Stages) is a reduced instruction set computer (RISC) architecture that was developed in the 1980s and is widely used in embedded systems and education1. MIPS assembly language is a low-level programming language that allows programmers to manipulate the hardware directly and perform operations such as arithmetic, logic, branching, and memory access.

Although MIPS has lost its relevance and influence in the current computing world, it still plays a role in teaching and learning computer organization and design, as well as in developing applications for embedded systems.

MARS (MIPS Assembler and Runtime Simulator) is a software tool that provides an integrated development environment (IDE) for MIPS assembly language programming3. MARS has several features that make it suitable for learning and teaching MIPS assembly language. MARS is also a lightweight and portable tool that can run on any platform that supports Java. It is designed to be easy to use and understand, and to provide a realistic and interactive experience of MIPS assembly language programming

1.2 BattleShip:

Battleship is a popular two-player game that involves strategy and guessing. It involves two players where each player tries to sink the other's ships by guessing their locations on a grid. Each player has five ships of different sizes that they secretly place on their own grid. Then, they take turns calling out coordinates to fire at the enemy's grid. If a coordinate hits a ship, the player marks it with a red peg. If it misses, the player marks it with a white peg. The first player to sink all of the enemy's ships wins the game.

Figure 1.2: Battleship in real life

The goal of this project is to create a battleship game in MIPS assembly language. The game has a 7x7 grid where the players place their ships. Each ship has a bow and a stern, and their coordinates are (rowbow, columnbow, rowstern, columnstern). Before the game starts, the players need to enter the coordinates of their ships. The grid cells have a value of 1 if there is a ship, or 0 if there is not. Each player has to place 3 ships of size 2x1, 2 ships of size 3x1, and 1 ship of size 4x1. The ships cannot touch or overlap each other.

II. Implementation:

2.1. Set-up phase:

Main menu:

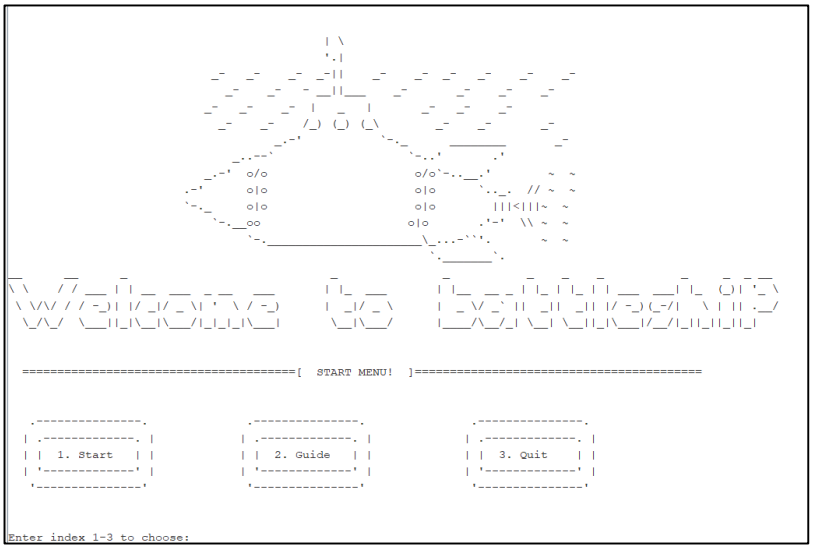


Figure 2.1: Main Menu

When the game starts, you will see a welcome screen with three options. You can press 1, 2, or 3 on your keyboard to choose what you want to do. If you press 1, you will begin the game and enter the preparation phase. If you press 2, you will see the instructions page where you can learn the basic rules and controls of the game. If you press 3, you will exit the program and end the game.

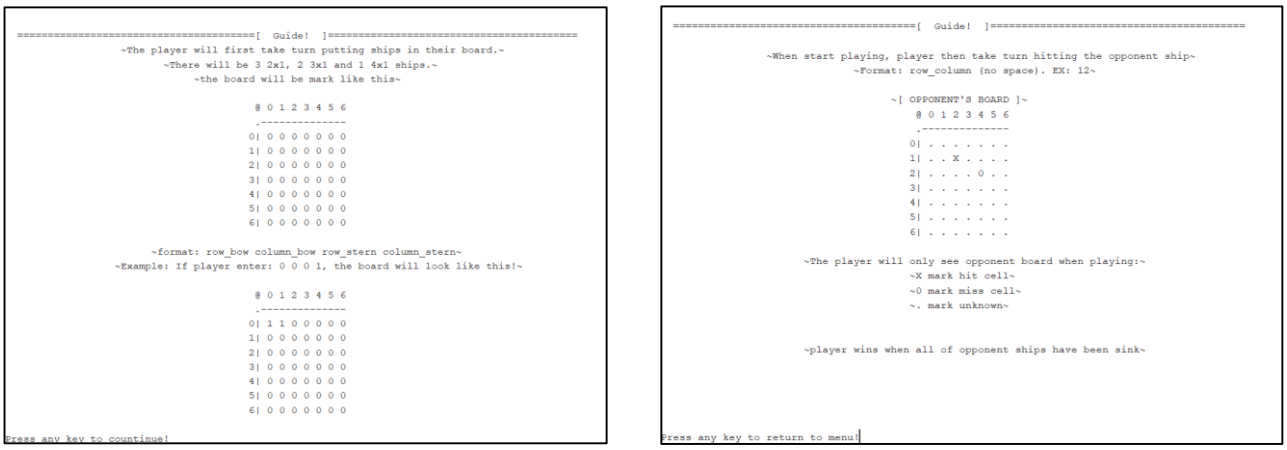


Figure 2.2: Instruction page

We use a array size 49 words with all 0s to represent two maps for two players. Each 0 means an empty square. The program change the array elements from 0 to 1 to show the player's ships. To find the index in the array, we use the formula row × 7 + column.

Player 1 places their ships first. The program will show to player's current board and ask them to input their ships. They have to enter 3 ships of 2x1, 2 ships of 3x1, and 1 ship of 4x1. The program then checks the input and updates the map.

The player's input is an 8-character string, where the program uses the characters at 0, 2, 4, and 6 as the ship indices. The program verifies the input and displays the current map in Run I/O. If the input is wrong, the program shows an error message with the reason.

The program then print out the board 1 last time after player 1 places their last ship.

```
=========================[  INITIALIZING!  ]=========================

                        ~[ PLAYER 1'S BOARD ]~
                            @ 0 1 2 3 4 5 6
                            .--------------
                          0| 0 0 0 0 0 1 1
                          1| 0 1 1 1 1 1 1
                          2| 0 0 0 1 1 1 1
                          3| 0 0 1 1 1 1 0
                          4| 0 0 0 0 0 0 0
                          5| 0 0 0 0 0 0 0
                          6| 0 0 0 0 0 0 0
```

Figure 2.3: An example the board after input.

After player 1 is done, the same process is done for player 2.

2.2. Playing phase:

The game is a two-player battle where each player has a fleet of ships hidden on a grid. The players take turns to enter coordinates and try to hit and sink their opponent's ships. The program shows the opponent's board for each player's convenience, so they can keep track of their hits and misses. A cell that has been hit is marked with an X, a cell that has been missed is marked with a 0, and a cell that has not been targeted is marked with a '.'. Whenever a player enters a coordinate, a pop-up message will

appear to indicate whether they have hit a ship or not. The message will say “Hit!!!” if the coordinate hits a ship, and “Miss!!!” if it does not

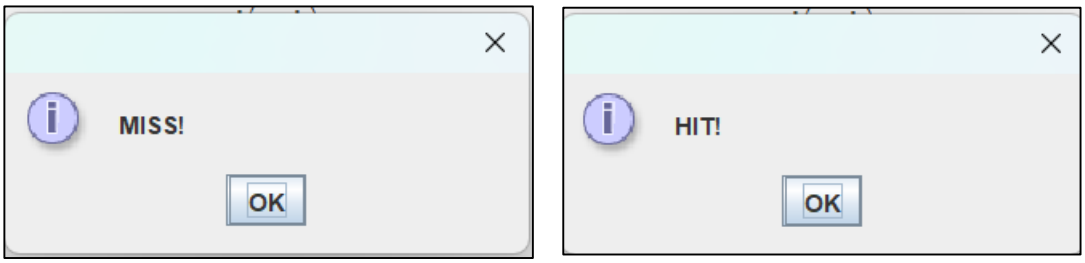


Figure 2.4: hit, miss pop-up.

The game will continue until there is no more ‘1’ on one of the players board, meaning the game ends when one player has sunk all of their opponent’s ships. When that happened, the pop-up will shown that the other player has won and the game will end.

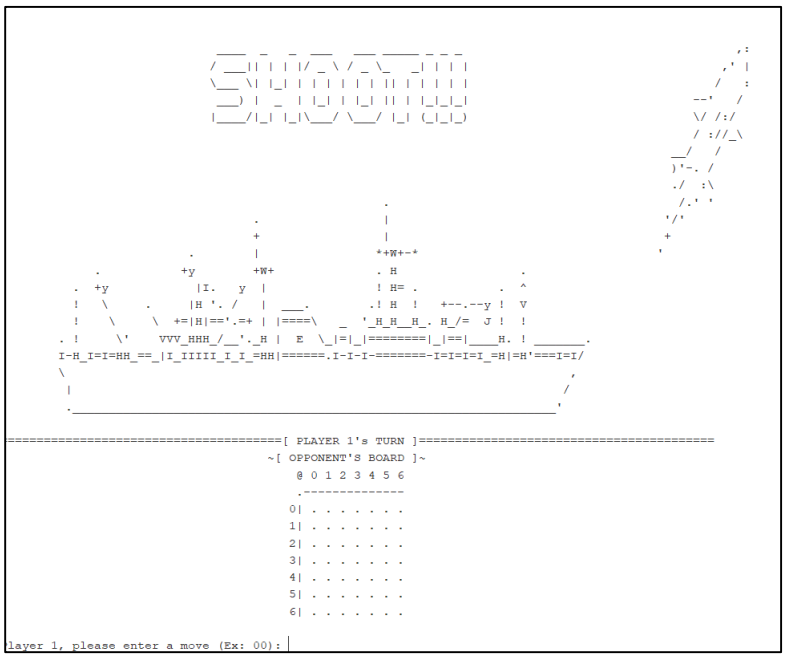


Figure 2.5: Playing phase UI.

2.3. Error checking:

If the player’s input isn’t valid, the program shows an error message with the reason and ask the player to input again. There are 5 types of error checking:

- Invalid input: The program will check each character the input string if it is in valid form (valid form:”row_bow column_bow row_stern column_stern” when initializing, row_column (no space) when playing and there is no symbol other than number and space). If not, the error pop up will appear.

- Out of bound: The error will shown when a number in the input is less then 0 or larger then 6. Since the board only contain number from 0 to 6.
- Diagonal ship: happened if both row of the ship's stern, bow and column of the ship's stern, bow are different.
- Invalid length: happened when the input length isn't match the ship length. The program check this by calculate the index of stern and bow and use the formual |column_bow-column_stern| if on the same row, otherwise |row_bow-row_stern| to find the length of input ship.
- Overlapped ship: Before putting the ships in, the program check if whether is is any 1's on cell that are about to be inputed.

2.4. Write to file:

The program will write to file with name "test.txt" the input of each player with format: "player X plays: 'Input' "

In addition:

- After player finish placing their ship, the program will print our their board for ease of checking.
- When finish initalizing, it will print "game start!!" in the file.

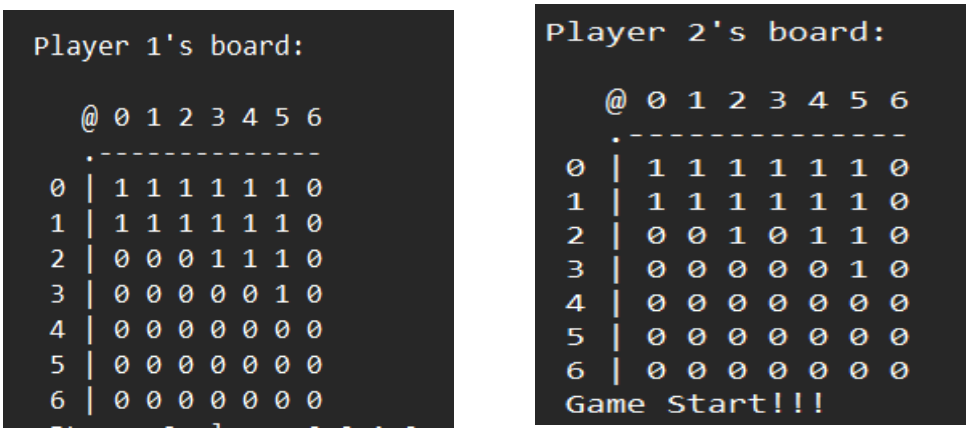
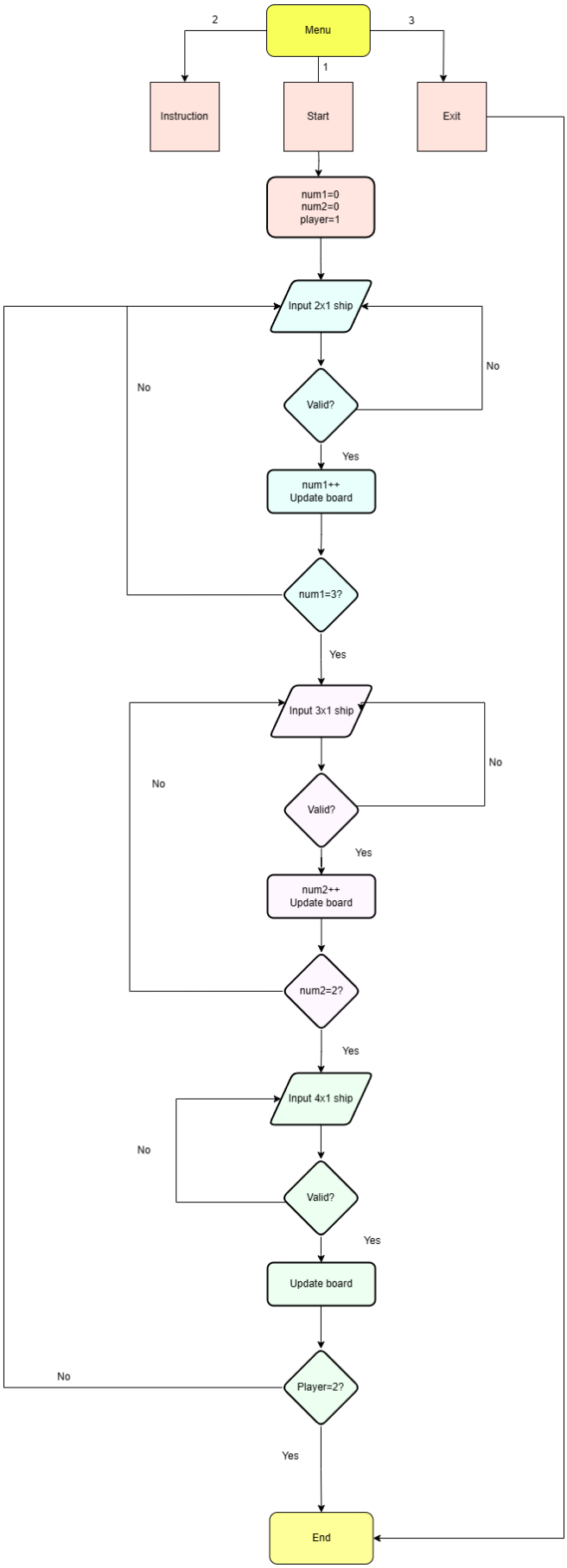


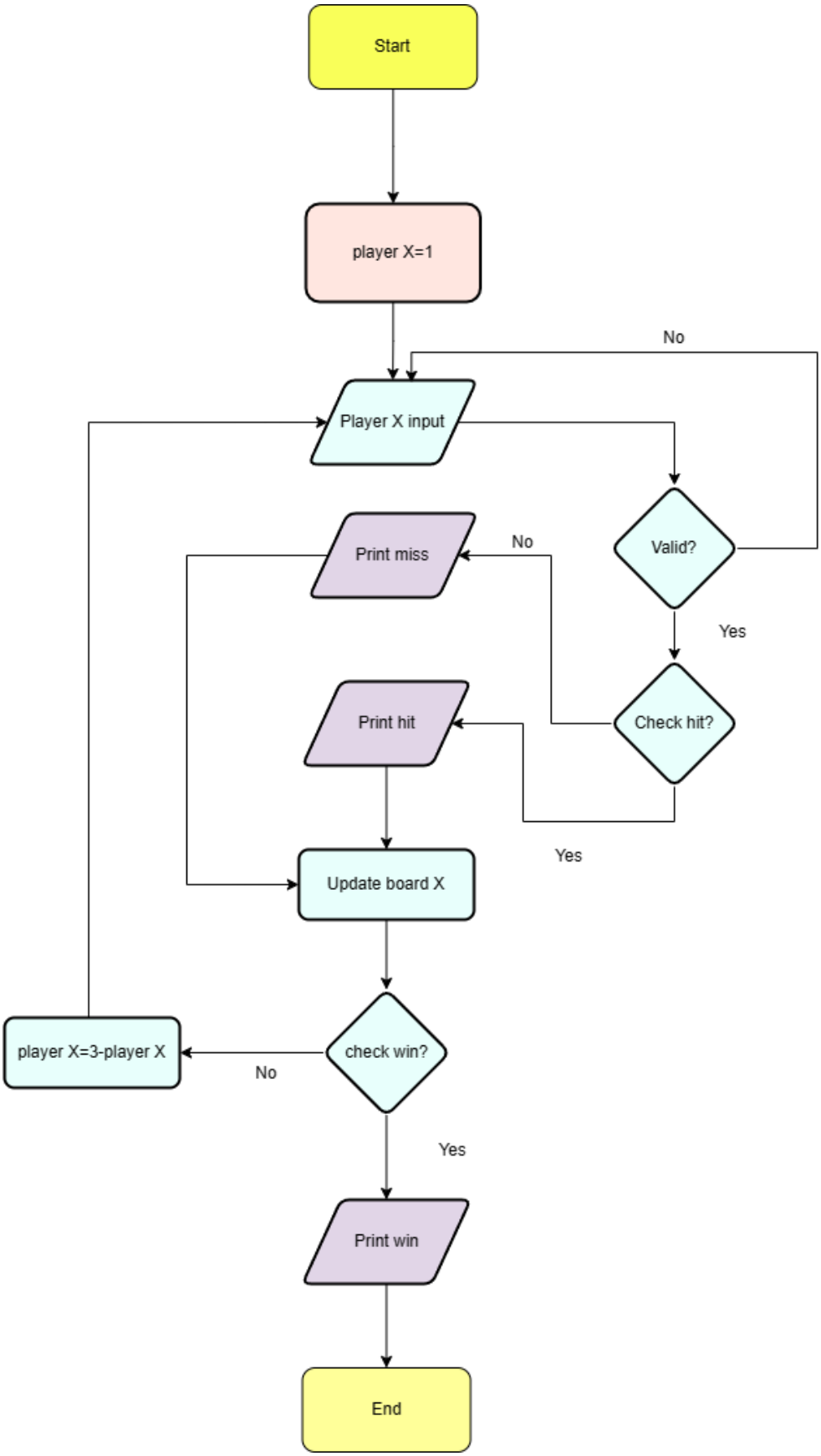
Figure 2.6:Example of writing to file.

2.5. Algorithms:

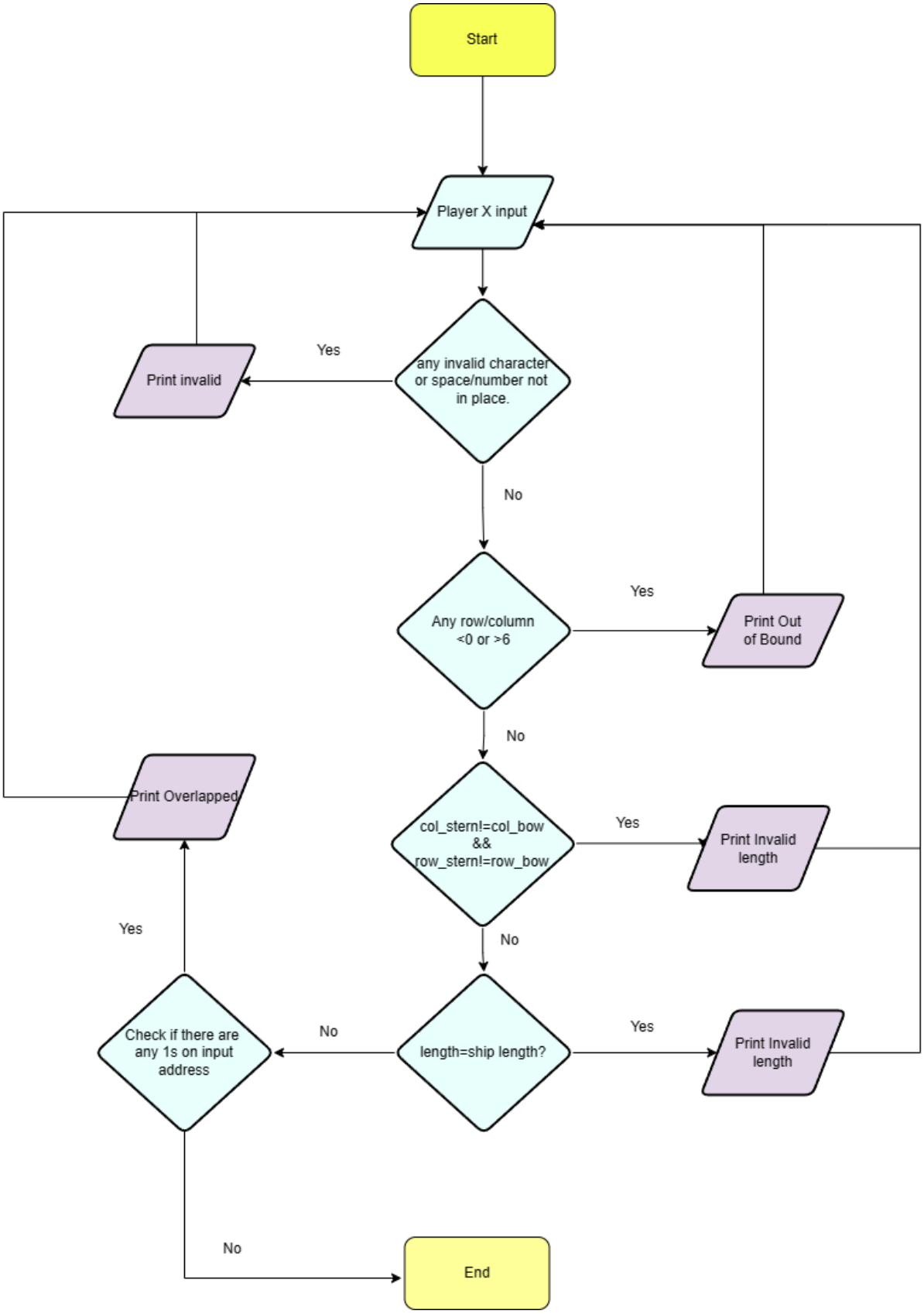
Flow chart for initial phase:



Flow chart for playing phase:

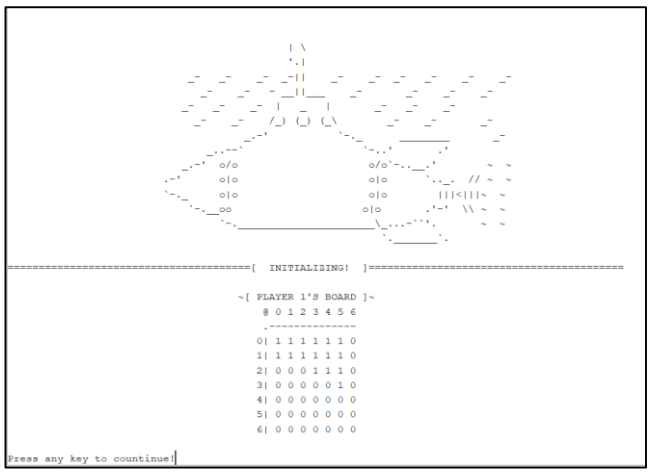


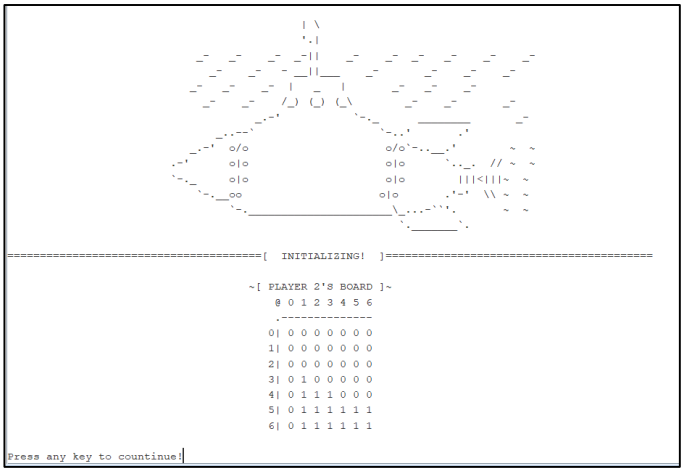
Flow chart for error checking:



III. Demo:

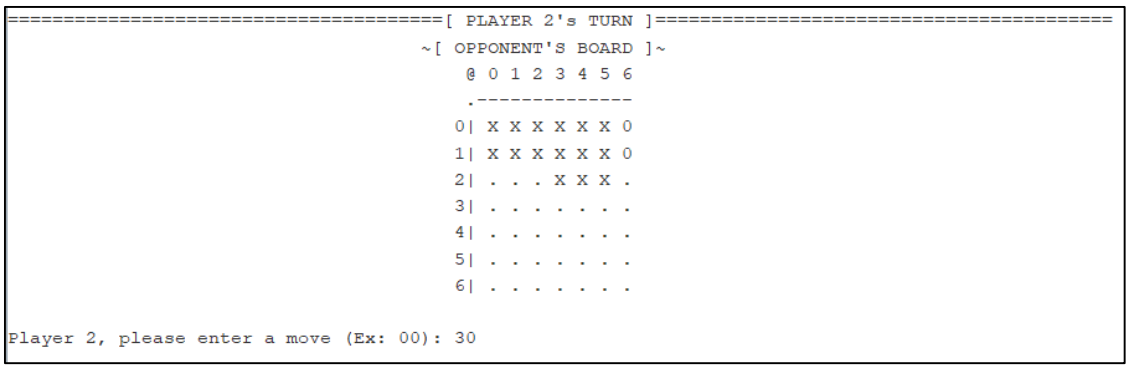
Intial board of 2 player:

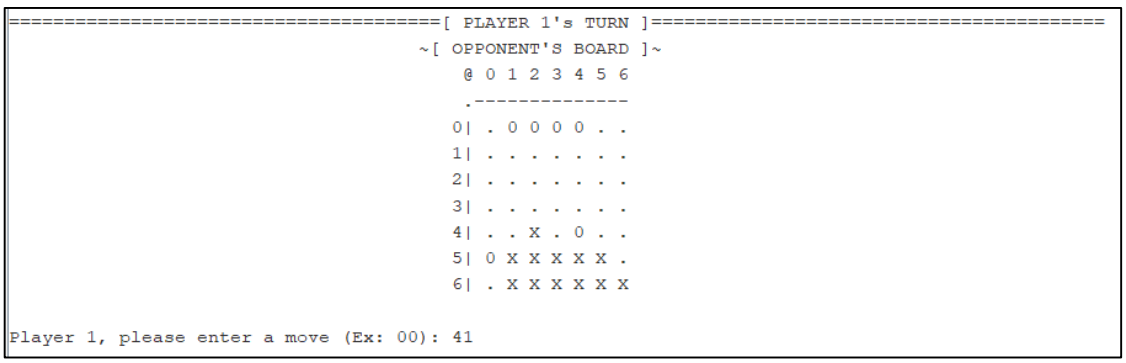




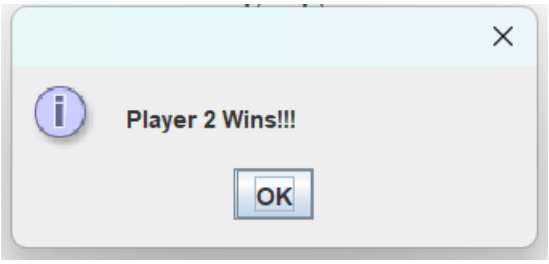
Each player move:																			
Player 1	01	02	03	04	66	61	62	63	64	65	55	54	53	52	51	50	44	42	41
Player 2	00	01	02	03	04	05	06	10	11	12	13	14	15	16	23	24	25	30	35

Each player move before the last move:





We can see that player 2 has won:



IV. References:

1. Wikipedia, “Battleship (game).” [Online]. Available: https://en. wikipedia.org/wiki/Battleship (game).

2. Assigment-v1.