HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
OFFICE FOR INTERNATIONAL STUDY PROGRAMS
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER ARCHITECTURE (CO3009)

TEST-BASED MULTIPLAYER
BATTLESHIP GAME WITH MIPS ASSEMBLY LANGUAGE

Under the guidance of: PROF. PHAM QUOC CUONG
Accomplished by: LE GIA HUY – 1952717

Ho Chi Minh City, 10/2023

Contents

1. Introduction

1.1. What is Battleship Boardgame?

Battleship (also known as ***Battleships*** or ***Sea Battle***[1]) is a strategy type guessing game for two players. It is played on ruled grids (paper or board) on which each player's fleet of warships are marked. The locations of the fleets are concealed from the other player. Players alternate turns calling "shots" at the other player's ships, and the objective of the game is to destroy the opposing player's fleet.

The game rules can be found in [2], in summary:

- Step 1: In a classic game, each player sets up a fleet of battleships on their map (a 10x10 grid). A fleet must contain a predefined set of battleships with different sizes. For example, a fleet of ships can consist of 5 2x1 ships, 3 3x1 ships, 1 5x1 ship.
- Step 2: After the ships have been positioned, the game proceeds in a series of rounds. In each round, each player takes a turn to announce a target square in the opponent's grid which is to be shot at. The opponent announces whether or not the square is occupied by a ship. If it is a "hit", the player who is hit marks this on their own or "ocean" grid (with a red peg in the pegboard version), and announces what ship was hit.
- Step 3: If all of a player's ships have been sunk, the game is over and their opponent wins.

Figure 1: Example of the battleship game in real life

1.2. MIPS assembly language and MARS

1.2.1 MIPS assembly language

MIPS[1] is a reduced instruction set computer (RISC) instruction set architecture (ISA). It was developed by MIPS Computer Systems in the USA. MIPS processors are used in embedded systems like

[1] Microprocessor without Interlocked Pipelined Stages

residential gateways and routers but originally, it was designed for general-purpose computing. Some video games were also developed using MIPS, like those found in [2]Nintendo 64 and [3]Sony PlayStation.

Figure 2: A Nitendo 64 console

1.2.2 MARS

MARS[4] is a lightweight interactive development environment for programming in MIPS assembly language. It is easy to use and can monitor registers and memory in an intuitive fashion. My version of battleship boardgame will be developed and executed on MARS.

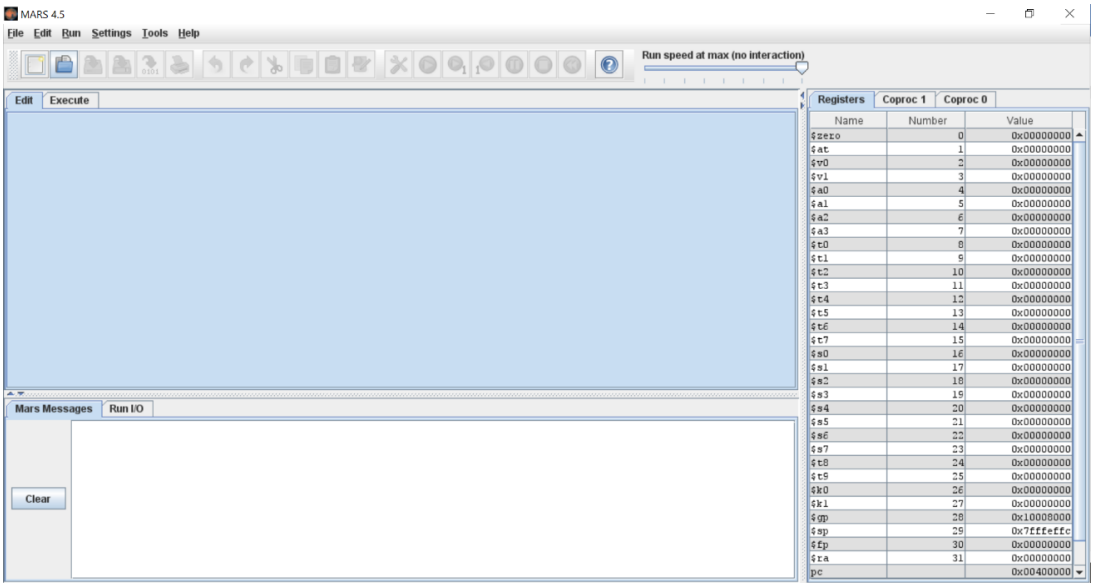

Figure 3: The default user interface of MARS

2. Battleship boardgame basics and user interface idea

2.1. Understanding the rules

It is crucial to understand the rules well in order to develop any game, not just Battleship boardgame. A firm awareness of the rules helps a lot in developing.

[2] "Main specifications of VR4300TM-series". NEC. Retrieved May 20, 2006
[3] "Sony's PlayStation Debuts in Japan!". Electronic Gaming Monthly. No. 65. Sendai Publishing. December 1994
[4] MIPS Assembler and Runtime Simulator

In this project, we will work with a smaller grid size which is **7x7**. We can deduce some key information about just Battleship boardgame as follows:

- There are two boardgame completely different, each player will have **three** of **2x1** ships, **two** of **3x1** ships, and **one** of **4x1** ship. Note that the ships can not overlap with each other
- At first, each player have to declare their own board. The player must enter 4 values when declaring each position of the boats, including the x and y positions where the boat starts and where the boat ends in order.
- The format for inputting the position of the boats is "row_{bow} $column_{bow}$ row_{stern} $column_{stern}$".
- Player only input the number from 0 to 6.
- In any case, when a player's board runs out of ships, this will be considered a loss for this player.
- There are 2 possible outcomes, which will display if we have a winner.
  For example, "[Player 1] wins!!" and "[Player 2] wins!!".

From these highlights, we can have a basic understanding of how the game should be implemented:

- At first, we begin with the preparation phase where the players set up their ship locations. The program should prompt to ask the players to insert the location of the ships. The values of the cells of the map will be either **1** or **0**. **1** means that the box is occupied by a ship, otherwise it's **0**. The players will need to setup the exact amount of ships with the same size in order to complete the setup phase. Considering to rule, each player will have **three** of **2x1** ships, **two** of **3x1** ships, and **one** of **4x1** ship.
- After the setup phase, both players will take turn targeting a box on the other player's map blindly. If the attack hits a cell that is occupied by a ship, an announcement should pop up in the terminal and says **"HIT"**, otherwise it's **"MISS"**. This will allow the attacking player to know if they hit an opponent's ship or not. The players will need to remember the cells that have been hit themselves. If a cell got hit, its value should change into **0**, as there it is no longer occupied by a ship. A ship is completely destroyed if all of the cells it's occupying are targeted. A player will lose if all of their ships got destroyed first. In other words, the game will end when a player's board contain only **0**'s.

A perfect example for inputting the boardgame, the player have to declare the coordinates for:

- The **yellow** of **4x1** ship is "0 0 0 3".
- The **blue** of **2x1** ships are "1 4 2 4", "2 1 3 1", and "4 4 4 5".
- The **green** of **3x1** ships are "5 3 5 5" and "6 0 6 2".

1	1	1	1	0	0	0
0	0	0	0	1	0	0
0	1	0	0	1	0	0
0	1	0	0	0	0	0
0	0	0	0	1	1	0
0	0	0	1	1	1	0
1	1	1	0	0	0	0

Figure 4: Example of the the ships that a player can place in this project

2.2. Welcome text and instruction

As a text-based game favors simplicity, it can be a problem for players to determine exactly what they need to do. Therefore, a clear instruction should be placed at the start of the program. By understanding the rules well, the job should be relatively easy.

In my case, I first state the rules and explain the input mechanism, and then print out the grid. Always state that ***Player1*** always declare the boardgame and play first. After the instruction, it is time for the players to enjoy the game.

```
-------------------- WELCOME TO THE BATTLESHIP BOARDGAME --------------------

Here is some rules you must follow to play this game:
      1. The values of the cells of the map will be either 1 or 0.
      --> 1 means that the box is occupied by a ship, otherwise it is 0.
      2. In detail, each player will have 3 of 2x1 ships, 2 of 3x1 ships, and 1 of 4x1 ship.
      3. Note that the ships can not overlap with each other.
      4. The player must enter 4 values when declaring each position of the boats, including the x and y positions where the boat starts and where the boat ends in order.
      5. The player must enter 2 values when playing boardgame, which means the x and y positions in order.
      6. The player 1 go first, then player 2.

--------------------------THE BATTLESHIP BOARDGAME--------------------------
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |


--------------!!!Hope you two have fun playing this boardgame!!!--------------
```

Figure 5: Game instructions: numbered grid, mark assignment, input instruction, and message (Image taken directly from my game's implementation).

2.3. The grid

The text-based part lies in the input, but things like the **7x7** grid has to be visible and constantly updated. The grid implementation can be done in two different fashions.

- Coordinate system: think of it as a small-scaled chess board: 0, 1, 2, 3, 4, 5, 6, 7 extending to the right and 0, 1, 2, 3, 4, 5,6 ,7 extending downwards in the Oxy plane. Players would enter the respective coordinates in order to place their designated mark the position of the boats on a block.
- Numbered grid: instead of the coordinate system, now the 49 blocks are numbered from 1 to 49. Players select a block by typing in the corresponding block number.

I decided to use the first method because the second method doesn't work based on the rules of the game. It will affect the player experience and make it difficult to recognize which box you have checked when there are numbers from 1 to 49 that do not change instead of 0 and 1 as required in the problem.

As a result, player will get acquainted and have fun.

3. Building the game

3.1. Storing inputs and checking for victory

Working with MIPS is a challenging task due to limited enrollment as well as unfamiliarity with the language. Luckily, we have registers that can hold arrays. Therefore, we only need 4 arrays of 49 1-byte elements because they are used to contain 1 and 0.

The first array is for the first player when designing his boardgame, similarly the second array is for the second player. The remaining 2 arrays will be in turn order to draw the boardgame in real time while playing

Specifically, when we decide to set a block to 1 or 0, we will loop through the 49-element array to arrive at the value to be modified, then we will convert it to 1 or 0 depending on demand demand of conditions.

Then, to determine the winning status, we set up a check mechanism after each player input. We need to check to see if there are any 1 elements left in the array. If there are no more 1's left, the winner will be determined immediately.

3.2. Grid visualization

The 7x7 grid can be set up easily through any text editor (I used VSCode) and then brought into MARS. First, we will declare it in the data section as 1 space with 49 elements, which means each element will contain 1 byte. Whenever we want to draw, we just need to use a loop to each character, then retrieve and print it.

At the same time as printing the character of each element in the array, we add a "|" and space to make the board easier to see and more convenient for players. At the same time, whenever a player finishes typing, we will print out the table and update it in real time so we know where they have marked.

```
------------------------THE BATTLESHIP BOARDGAME P1--------------------------
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

First player input the position of first ship 2x1: 4445

------------------------THE BATTLESHIP BOARDGAME P1--------------------------
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

First player input the position of second ship 2x1:
```

Figure 6: The boardgame always print out when it's the player's turn

3.3. Preventing the game from breaking

Everyone hates bugs and game makers are no exception. Sometimes a small mistake can ruin the whole thinggame mechanics, making the game unplayable or leading to bad decisions.

In the game of Battleship Boardgame there can also be errors and we must implement a checking mechanism to prevent that. These are possible errors and solutions:

- Trying to fill an already occupied block: preventing this is quite simple. As I have used 1 array to store the input, and fortunately by default it will always be 0 in each array. So we can check if a register is zero or not. If it is 1 then it is filled and cannot be accessed again by another input.
- Enter an input value that is not within the 7x7 matrix range: I am using an array, so if I enter more than that, it will give an error and cannot be stored inside the array. Therefore, I will use an algorithm by considering that if the input is greater than 6 or less than 7, I will print an error message and force the player to enter again.
- Entering the input value is not the correct size of the boat according to the rules: I use a string so if I enter 4 values. Therefore, I will use an algorithm to calculate the boat size. If the size is not correct, I will print an error message and force the player to re-enter it.
- Entering the input value is not the correct position of the boat according the start position and the end position: I use a string so if I enter 4 values. Therefore, I will use an smae algorithm to calculate the boat size. If the size is the positive value, I will print an error message and force the player to re-enter it.

In the mean time, I am hopeful that players strictly follows the instruction.

Some example of the exact counter-measures is demonstrated below.

```
------------------------THE BATTLESHIP BOARDGAME P1--------------------------
                       | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

First player input the position of second ship 2x1: 0102

Please enter again, you have entered the wrong value according to the rules of the game!!!
```

Figure 7: An example of bug checking.

In Figure 7, the player attempts to fill in the occupied block at the postion 01, so the game prints an error message and then immediately asks for another input.

```
------------------------THE BATTLESHIP BOARDGAME P1--------------------------
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

First player input the position of first ship 2x1: 0003

Please enter again, you have entered the wrong value according to the rules of the game!!!
```

Figure 8: An example of bug checking.

In Figure 8, the player attempts to fill in the block with wrong format according to the rule, so the game prints an error message and then immediately asks for another input.

```
------------------------THE BATTLESHIP BOARDGAME P1--------------------------
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                       | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

First player input the position of first ship 2x1: 0607

Please enter again, you have entered the wrong value according to the rules of the game!!!
```

Figure 9: An example of bug checking.

In Figure 8, the player attempts to input the value with wrong format according to the rule, so the game prints an error message and then immediately asks for another input.

4. Program flowchart

Because of the long condition for each step, we decide to draw a summary flow chart in order to clean for following. The execution of the game can be describe with the flowchart in figure below

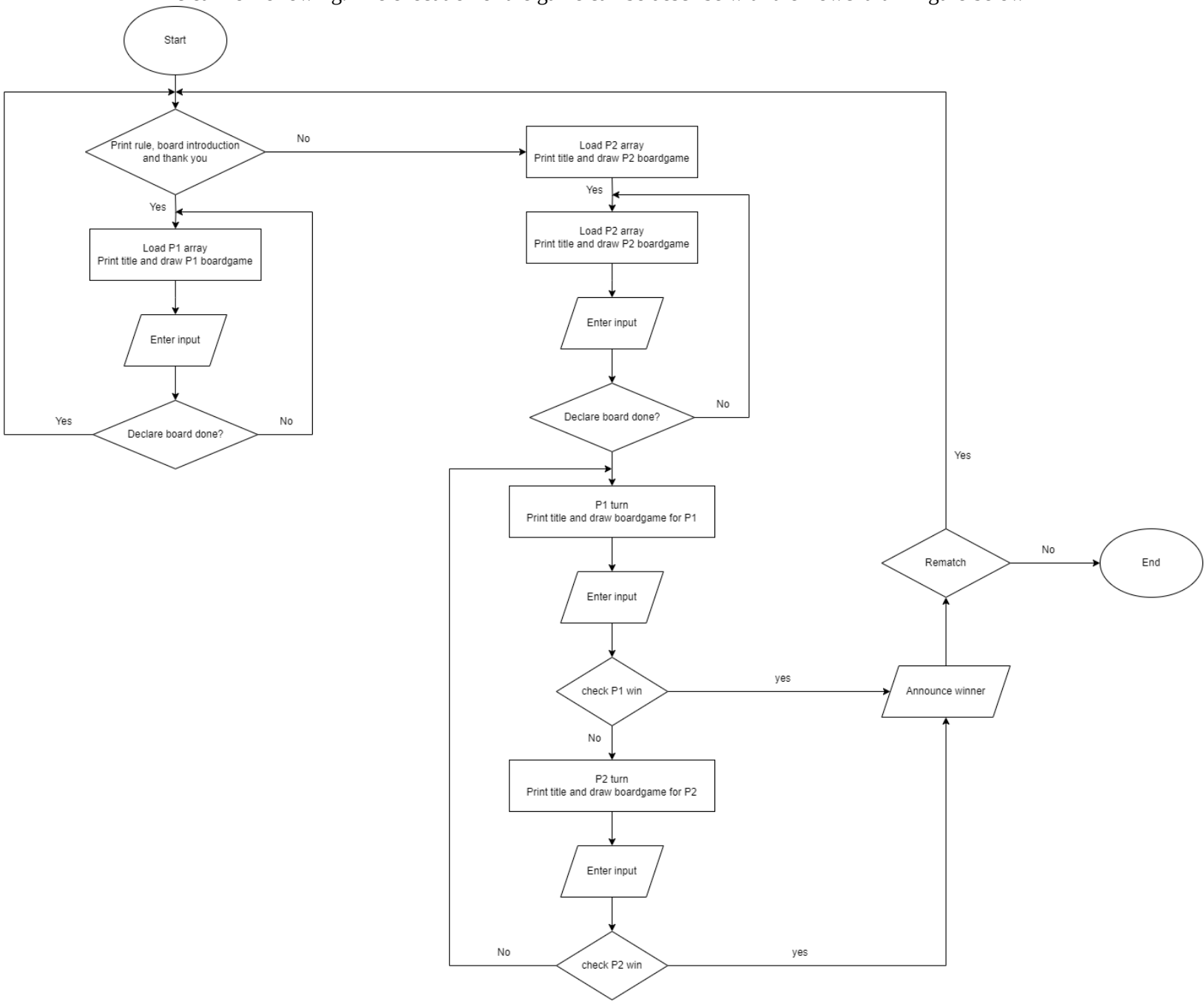

Figure 10: Summary program flowchart of Battleship Boardgame.

5. Demonstration

Here are some images taken from the game, showcasing the discussed features.

```
-------------------------THE PRESENT BOARDGAME FOR P1-------------------------
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |


[Player 1 TURN] --> input move: 14

-----------------------------------> HIT <-----------------------------------
```

Figure 11: Player 1 input the right format and hit the block that contain the Player 2's boat.

.

```
-------------------------THE PRESENT BOARDGAME FOR P2-------------------------
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
                        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |


[Player 2 TURN] --> input move: 14

-----------------------------------> MISS <-----------------------------------
```

Figure 12: Player 2 input the right format and hit the block that contain the Player 1's boat.

```
------------------------------>[Player 1] wins!!<------------------------------


Do you want to continue? Type 1 for YES / Type 2 for NO
Your option is: 2

-- program is finished running --
```

Figure 13: Player 1 win and want to rematch or not.

6. Conclusion

By taking a deep look into the game's mechanism and implementation, I have learned more about MIPS programming as well. Overall, I think this is a great way to learn about programming in general. An elegant use of branching and bit-wise logic is critical, and thanks to that, my game came to life!

7. External link

External link Raw source code can be found at this Github address: **Link**.

Download MARS MIPS here: **Link**.

Flowchart drawn with **Draw.io**.

8. Reference

[1] https://mips.com/

[2] ”Main specifications of VR4300TM-series”. NEC. Retrieved May 20, 2006

[3] ”Sony’s PlayStation Debuts in Japan!”. Electronic Gaming Monthly. No. 65. Sendai Publishing. December 1994.