

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



Computer Architecture (Lab) - CO2008

Assignment:

FIVE IN A ROW

INSTRUCTOR: NGUYEN THIEN AN
STUDENT: DOAN HONG HIEU KIEN - 2052555
CLASS: CC02, GROUP 9

HO CHI MINH CITY, March 2025



Contents

1	Introduction	3
2	Requirements	4
2.1	Rules of the game	4
2.2	Detailed requirements	4
3	Algorithm	5
3.1	Access an element at a specific row and column of a 2D array.	5
3.2	Initialize the board	5
3.3	Print the board	5
3.4	Get and parse user input	6
3.4.1	Check the length of input	6
3.4.2	Count the number of commas	7
3.4.3	Check digits	7
3.4.4	Check leading zeros	7
3.4.5	Check that the characters on both sides of the comma are digits. . .	7
3.4.6	Parse number	7
3.4.7	Check valid coordinates	7
3.5	Check winning state	7
3.5.1	Naive Approach: Perform a brute-force search by looping through every cell on the board.	8
3.5.2	Better Approach: Perform direction checks at specific points	8
3.6	Main program	9
3.7	Write to file	10
4	Testing	10
4.1	The showcase	11
4.2	User 1 and 2 enter a valid coordinate	13
4.3	The user inputs an invalid format or coordinate.	14
4.4	User wins	15
4.5	Write game result to file	20
5	References	21



List of Figures

1	A standard Gomoku game	3
2	Gomoku board in the terminal	6
3	Flow chart of the game	9
4	Players can view the moves they made in each turn.	10
5	The showcase (The board does not fully display in the terminal.)	11
6	The rest of the board (with a prompt for user 1)	12
7	User 1 inputs a valid coordinate	13
8	User 2 inputs a valid coordinate	14
9	Some examples of invalid format and coordinate	15
10	Five in a row vertically	16
11	Five in a row horizontally	17
12	Five in a row diagonally	18
13	Five in a row diagonally	19
14	Write the result to file	20

1 Introduction

Traditional board games have gone through many iterations throughout history. From the classic game of Chess to more recent ones like Uno or Poker, these games captivate their audiences with a wide variety of fun yet skillful gameplay. Among them, Gomoku stands out as a simple yet strategic game enjoyed by both the young and the old.

Gomoku — also known as Five in a Row, Tic-Tac-Toe in English, or Caro in Vietnamese — is an abstract strategy board game with historical records dating back to the mid-1700s during Japan's Edo period. The name "Gomoku" comes from Japanese, where it is referred to as *gomokunarabe*: *go* means five, *moku* is a counter word for pieces, and *narabe* means to line up. The game is also popular in China, Korea, and other Asian countries.

Since its creation, Gomoku has undergone many rule changes, giving rise to various game variants that keep it fresh and competitive. The World Gomoku Championships have been held since 1989, hosted by different countries with changing champions. Needless to say, the game has aged like fine wine.

In the standard Gomoku rules, players alternate turns placing a stone of their color on an empty intersection. Traditionally played with Go pieces (black and white stones), the black player moves first. The winner is the first to form an unbroken line of five stones of their color, either horizontally, vertically, or diagonally. In some rule sets, the winning line must be exactly five stones long; a line of six or more stones is considered an "overline" and does not count as a win. If the board is completely filled and no player has formed a line of five, the game ends in a draw.

The figure below shows a Gomoku game where Black wins:

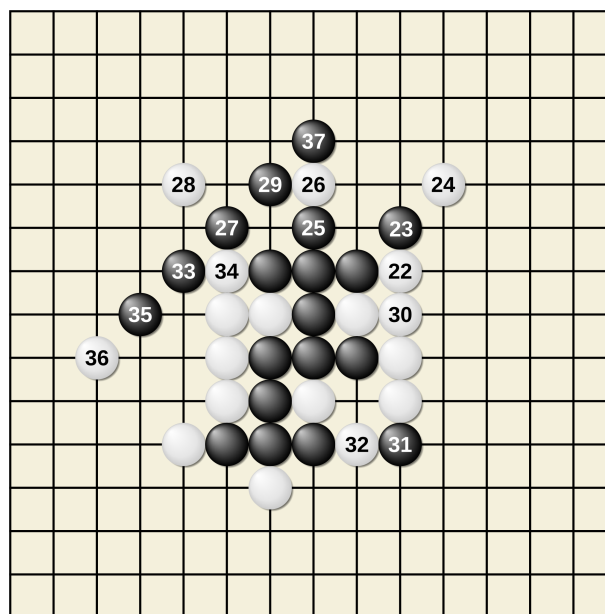


Figure 1: A standard Gomoku game

2 Requirements

As mentioned in section 1, the game has many variants, ranging from different first move to swapping rules and winning conditions. For this assignment, the game will be played with a standard 15×15 board, using "X" and "O" pieces, which are familiar to the Vietnamese people.

2.1 Rules of the game

The standard rules for a Tic-Tac-Toe game are applied, means that:

- The first player will play with the X symbol, while the second player will play with the O symbol.
- The first player to get five in a row (vertically, horizontally, or diagonally) wins.
- If the board is full and no winner can be determined, the game ends in a draw.

2.2 Detailed requirements

- An empty board is displayed in the terminal at the beginning to show case the start of the game.
- A prompt for each player is needed to begin a move. The prompt should be: **Player 1, please input your coordinates** and **Player 2, please input your coordinates**.
- The program must then receive a coordinate input in the form of x,y which are the horizontal and vertical coordinates respectively. Example: **4,5** indicates row 4, column 5. Note: coordinates start from **0** and end at **14**. It is imperative that the program automatically checks for any form of incorrect input such as out of range, wrong format, etc. and provide the warning message to the player. A wrong input should be warned and a re-input is required from the player.
- The board updated with the new move must be shown right after the player input. **X** is shown in the place of player 1 moves while **O** represents player 2.
- The winning condition is the first player to get 5 pieces in a line (horizontally, vertically, or diagonally). A tie occurs when all squares have been played yet none of the players can find the winning condition. In the case of a win, a final statement **Player 1 wins** or **Player 2 wins** is shown in the terminal. A tie would instead shows **Tie**.
- The program must also write the final version of the board (either win or tie) and the final statement into a text file name "result.txt"

3 Algorithm

In the algorithm, a board with size 225 bytes (15×15) is used, where each byte can be either 0 (ASCII code for NULL character), 79 (ASCII code for O) or 88 (ASCII code for X). This board will be referred to as **board**.

3.1 Access an element at a specific row and column of a 2D array.

MIPS does not have built-in syntax for multidimensional arrays like high-level languages, but we can implement a 2D array by managing memory access manually. Assuming row-major order (as is standard in MIPS), the address of the element at row x and column y can be calculated using the following formula:

$$\text{addr}(x, y) = \text{addr}(0, 0) + (x * \text{no_of_column} + y) * \text{elem_size}$$

Here, $\text{addr}(0, 0)$ is called the base address of the array.

Using the above formula, we can access `board[x][y]` by substituting `no_of_column = 15` and `elem_size = 1` (since `board` only contains characters).

3.2 Initialize the board

We must initialize the board at the beginning of the game to prevent any unexpected behavior caused by garbage values.

We can do this by looping through all elements of **board** and setting each element to 0.

3.3 Print the board

We can simply print the board by printing each character separated by spaces (with the NULL character replaced by a hyphen '-'), and each line separated by a newline. However, for a cleaner presentation, we use the following format as shown in the image:

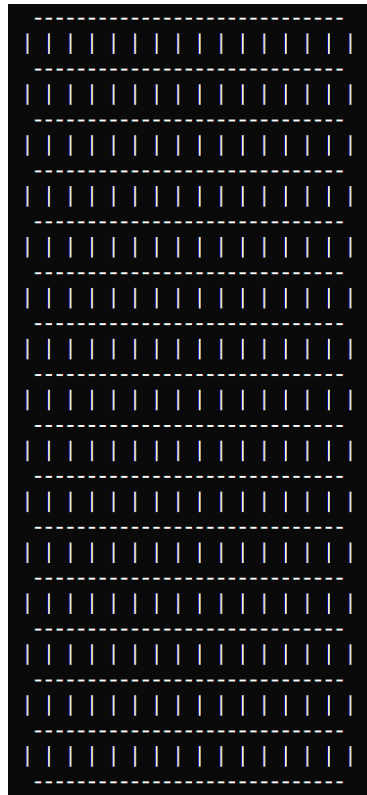


Figure 2: Gomoku board in the terminal

We can achieve this by looping through each row of `board` and drawing the horizontal line. For each column, draw a vertical line `|` followed by the character at that row and column (with the `NULL` character replaced by a space). At the end of each line, draw a vertical line and a newline character `\n`. Finally, draw a horizontal line at the end of the board.

3.4 Get and parse user input

Getting user input can be done easily using `li $v0 8`, so we will focus on parsing input. If any of the following steps fail, the program will warn the user.

3.4.1 Check the length of input

We know that the input has the form `x,y`, where `x` and `y` are within the range `[0, 14]`. Therefore, we can calculate the length of the user input and check whether it falls between 3 (if `x` and `y` are one-digit numbers) and 5 (if `x` and `y` are two-digit numbers).

Note that the user's input string ends with a newline character, not a `NULL` character.



3.4.2 Count the number of commas

We create a variable to count the number of commas in the input string and compare that variable to 1.

3.4.3 Check digits

Except for the commas, the rest of the input string must consist of digits (with their ASCII codes falling between 48 and 57).

3.4.4 Check leading zeros

We do not accept numbers like 00, 01, 02, ..., so we will check if the current character is '0' and the next character is either a comma or a newline (but we must check if the temporary value is zero or not for the case number 10).

3.4.5 Check that the characters on both sides of the comma are digits.

Simply ensure the comma is not at the beginning of the string and is not followed by a newline.

3.4.6 Parse number

We initialize a temporary variable and a 1D array of length 2 to hold the row and column indices. Using decimal representation, we parse each digit character and accumulate the value in the temporary variable. When a comma or newline is encountered, we store the accumulated value in the array. If the character is a comma, we then reset the temporary variable to 0.

3.4.7 Check valid coordinates

A coordinate (x, y) is considered valid if:

- x and y are within the range $[0, 14]$.
- The character at row x and column y of `board` is a NULL character (this check is separate from the coordinate validation and is used solely for determining a winning state, as described in the next subsection.).

After passing all the above steps, the user's symbol is placed in `board` at the position $15x + y$

3.5 Check winning state

Multiple algorithms can be used to check for a winning state. The following is a summary of each method.

3.5.1 Naive Approach: Perform a brute-force search by looping through every cell on the board.

We iterate through all 225 cells of the `board`. For each cell that contains a non-null character, we perform an 8-directional check (N, NE, E, SE, S, SW, W, N - similar to compass directions) to determine if there is a set of 5 consecutive identical symbols in any direction. This algorithm works for any state of the board.

To perform the 8-directional check, we use two arrays: one for the row indices and one for the column indices. The values in these arrays correspond to the directions mentioned above (N, NE, E, SE, S, SW, W, NW), as follows:

- Row dimension: $-1, -1, 0, 1, 1, 1, 0, -1$
- Column dimension: $0, 1, 1, 1, 0, -1, -1, -1$

For the i -th direction ($i \in [0, 7]$), we move to the next cell by adding the x and y coordinates with the i -th elements of the row and column dimension arrays, respectively (we also need to check if the resulting coordinates go outside the board).

The loop for each direction check terminates if:

- There is a set of 5 consecutive identical symbols (the symbol is not null) found on the board. In this case, we return the symbol.
- The number of consecutive identical symbols is less than 5 (either the coordinates go outside the board, the coordinate contains a different symbol, or it is a NULL character).

If the 8-directional check fails, return 0.

3.5.2 Better Approach: Perform direction checks at specific points

Instead of looping through all cells in the `board`, we only need to check the user's input point (since the element at this coordinate is always non-null).

We also reduce the size of the direction arrays to 4 (i.e., we only use the W, NW, N, and NE directions, since there are 4 pairs of x, y that negate each other). Therefore, we need to perform two checks: one for the chosen direction and one for the opposite direction.

For each direction, calculate the number of consecutive cells whose symbol matches the user's input symbol.

The condition for terminating a loop for the direction check remains the same as above. The key difference is the return value. In this approach, we return 1 if a valid set is found (i.e., a set of 5 consecutive identical symbols), and 0 otherwise.

In the code, we use this approach for better performance.

3.6 Main program

The flow of the game is as follows:

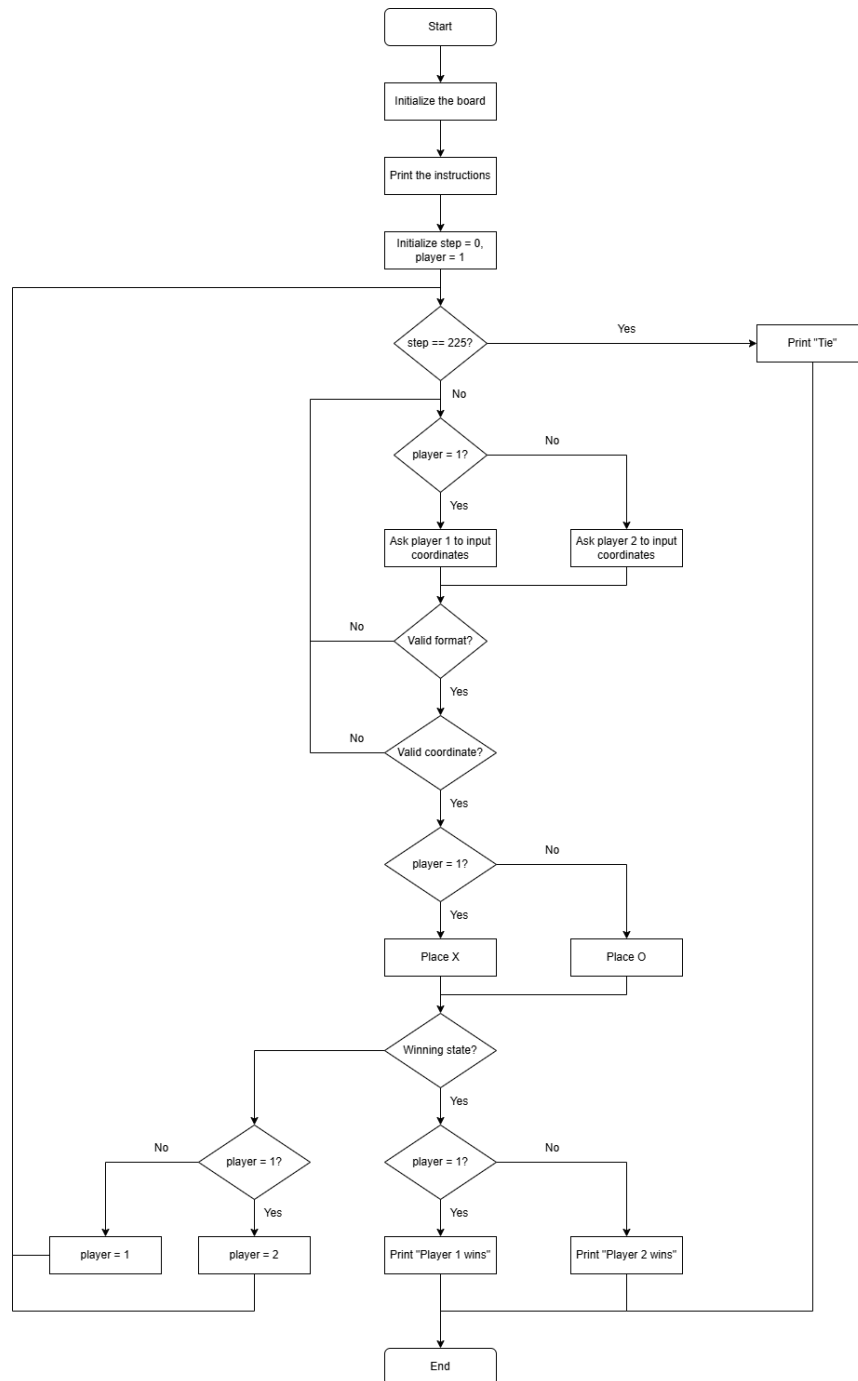
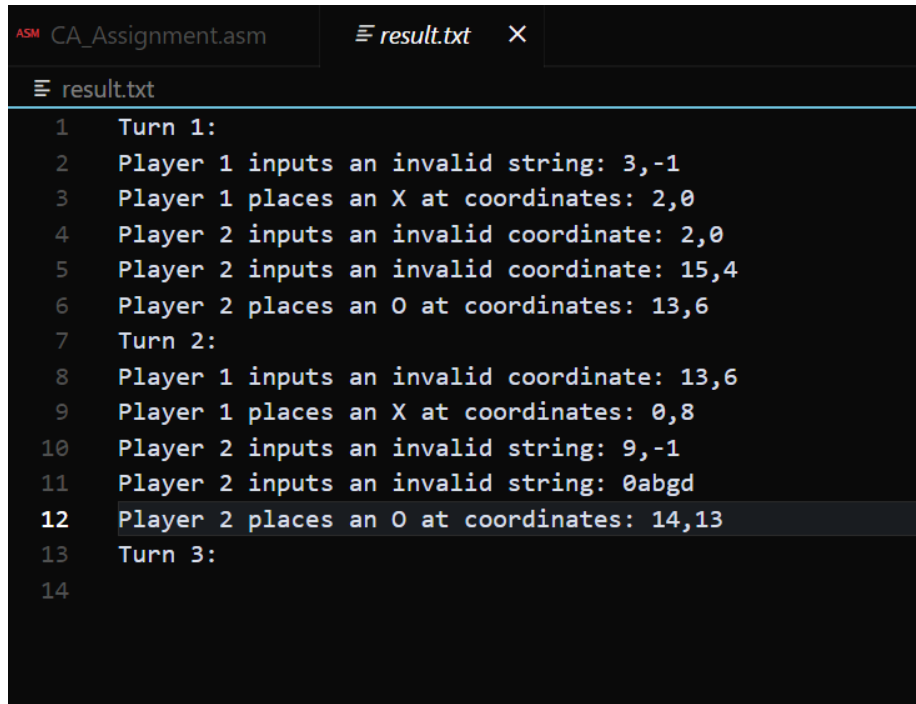


Figure 3: Flow chart of the game

3.7 Write to file

The logic for writing the board to a file differs from printing it to the terminal (although the output format is the same). We use heap allocation to allocate a 32-character string for each row of the `board`. For each column, we calculate the offset in the string by multiplying the column index by 2, then append a vertical line (ASCII code 124) followed by the character at the corresponding row and column in `board` (replacing it with a space if the character is NULL, or a newline if it's the end of the row). For each row, we write a horizontal line and the allocated string to the file. A horizontal line is also written at the end of the board. After writing the board, we append the game state message to the file.

Before writing the board to the file, we also print the list of moves - including invalid ones - made by both players, grouped by turns. A turn is counted when both players have successfully made valid moves.



```
ASM CA_Assignment.asm  result.txt X
result.txt
1  Turn 1:
2  Player 1 inputs an invalid string: 3,-1
3  Player 1 places an X at coordinates: 2,0
4  Player 2 inputs an invalid coordinate: 2,0
5  Player 2 inputs an invalid coordinate: 15,4
6  Player 2 places an O at coordinates: 13,6
7  Turn 2:
8  Player 1 inputs an invalid coordinate: 13,6
9  Player 1 places an X at coordinates: 0,8
10 Player 2 inputs an invalid string: 9,-1
11 Player 2 inputs an invalid string: 0abgd
12 Player 2 places an O at coordinates: 14,13
13 Turn 3:
14
```

Figure 4: Players can view the moves they made in each turn.

4 Testing

To ensure that our program meets the requirements, testing is necessary. Below are several test cases used to verify the correctness of our program.



4.1 The showcase

```
PS C:\Users\VOSTRO 3490\OneDrive\Desktop\CA_Lab> java -jar ../Mars4_5.jar nc CA_Assignment.asm
Welcome to Five in a Row, a game of strategy and skill.
Player 1 will be X and Player 2 will be O.
Enter the coordinates of your move in the format 'row,column'.
For example, to place an X in the top-left corner, enter '0,0'.
The board is 15x15, so the coordinates range from 0 to 14.
Good luck!

| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
```

Figure 5: The showcase (The board does not fully display in the terminal.)

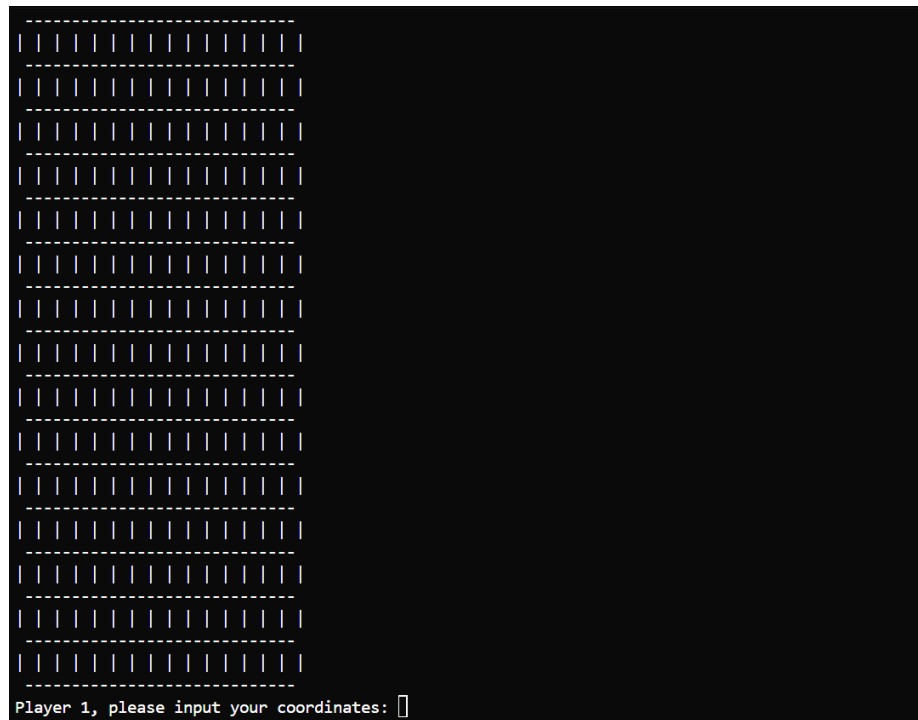


Figure 6: The rest of the board (with a prompt for user 1)



4.2 User 1 and 2 enter a valid coordinate



Figure 7: User 1 inputs a valid coordinate

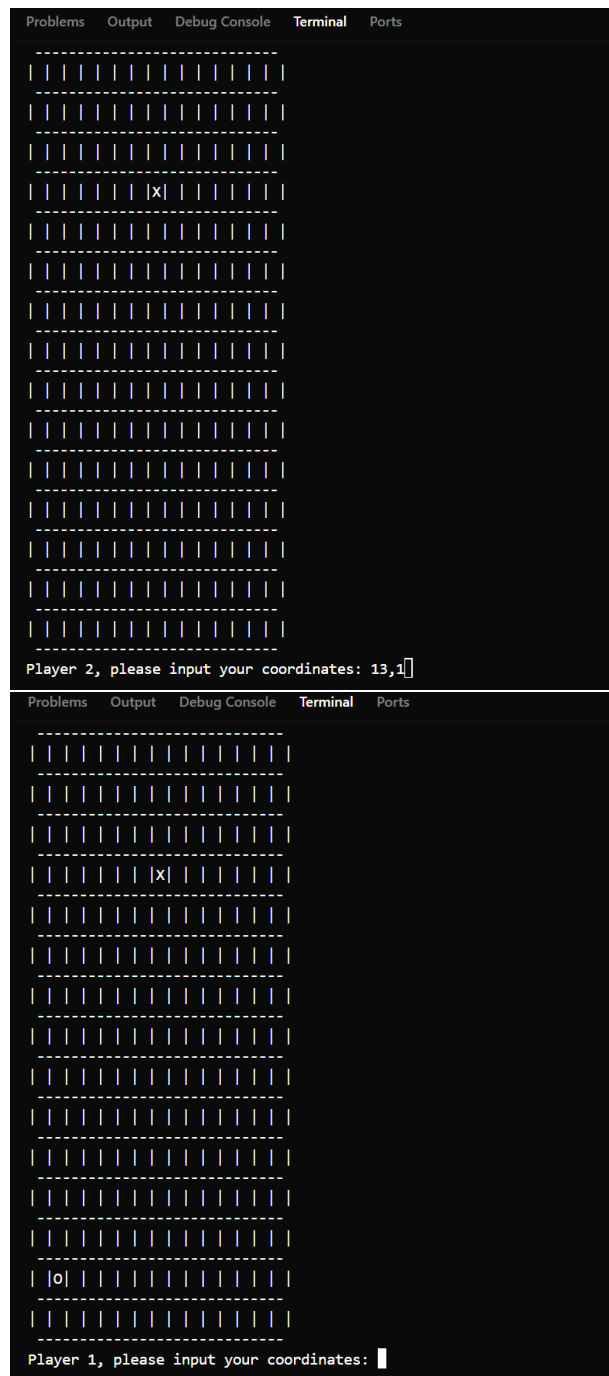


Figure 8: User 2 inputs a valid coordinate

4.3 The user inputs an invalid format or coordinate.

Assume that the cell at row 3 and column 7 is already occupied.

```
Player 2, please input your coordinates: 3,7
The cell is already occupied. Please try again.
Player 2, please input your coordinates: 00,3
Invalid input. Please try again.
Player 2, please input your coordinates: 1,15
Invalid coordinates. Please try again.
Player 2, please input your coordinates: 15,1
Invalid coordinates. Please try again.
Player 2, please input your coordinates: abcdef
Invalid input. Please try again.
Player 2, please input your coordinates: 3,00
Invalid input. Please try again.
Player 2, please input your coordinates: ,0
Invalid input. Please try again.
Player 2, please input your coordinates: ,10
Invalid input. Please try again.
```

Figure 9: Some examples of invalid format and coordinate

4.4 User wins

We test our program using the following input sequences for one user, while the other user plays randomly.

1. User 1 plays 0,0; 1,0; 2,0; 3,0; 4,0 and wins vertically

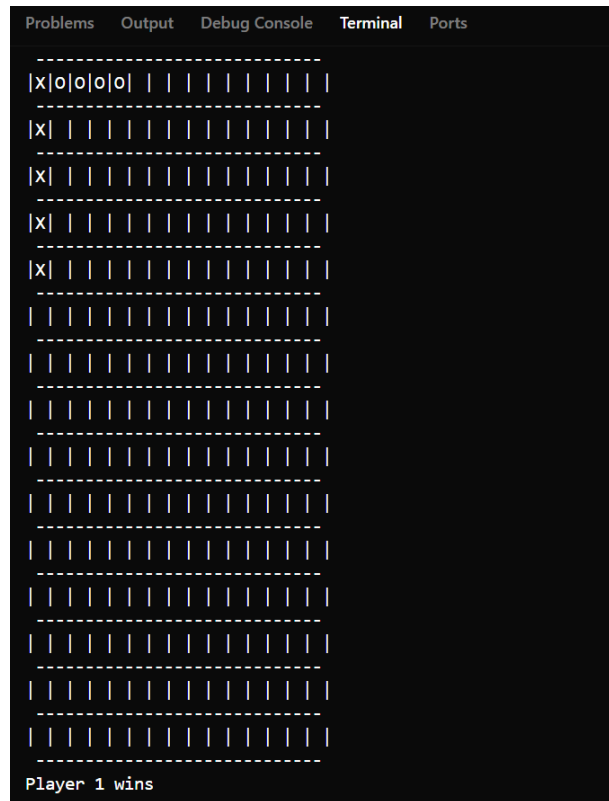


Figure 10: Five in a row vertically

2. User 2 plays 0, 4; 0, 2; 0, 1; 0, 3; 0, 0 and wins horizontally



Figure 11: Five in a row horizontally

3. User 2 plays 6, 5; 10, 9; 8, 7; 7, 6; 9, 8 and wins diagonally

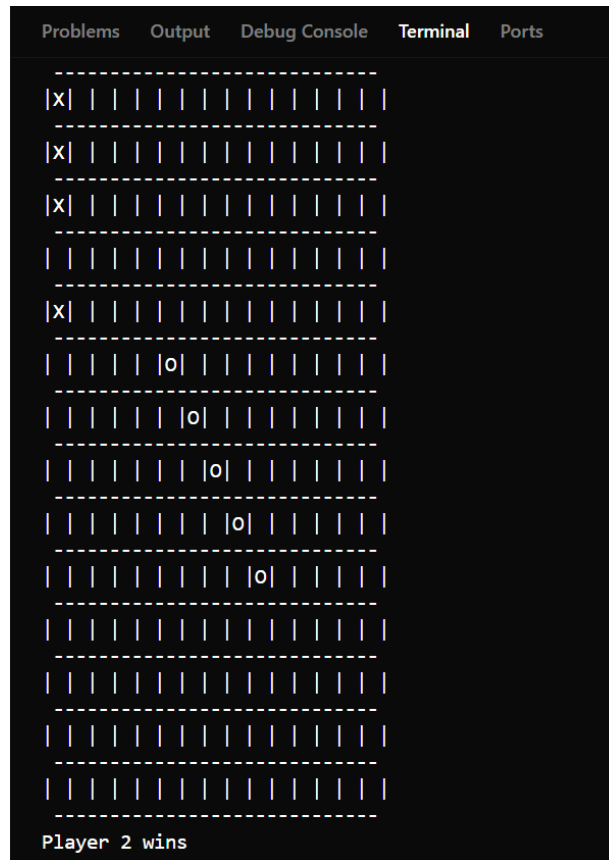


Figure 12: Five in a row diagonally

4. User 1 plays 7, 7; 5, 9; 9, 5; 8, 6; 6, 8 and wins diagonally

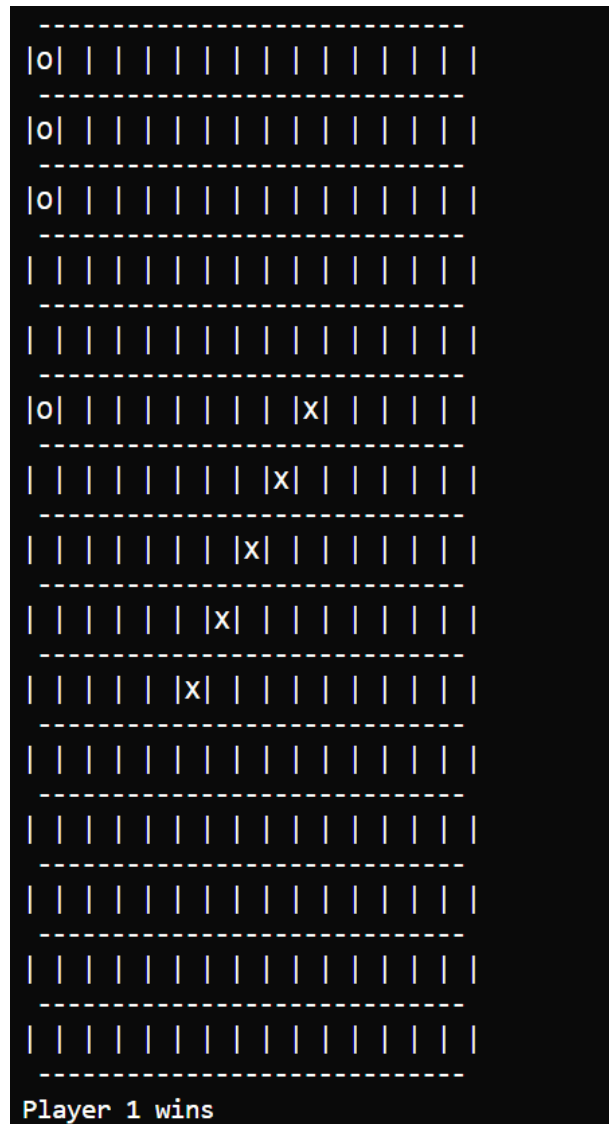


Figure 13: Five in a row diagonally

4.5 Write game result to file

```

1  Turn 1:
2  Player 1 places an X at coordinates: 0,0
3  Player 2 places an O at coordinates: 1,0
4  Turn 2:
5  Player 1 places an X at coordinates: 1,1
6  Player 2 places an O at coordinates: 2,0
7  Turn 3:
8  Player 1 places an X at coordinates: 2,2
9  Player 2 places an O at coordinates: 3,0
10 Turn 4:
11 Player 1 places an X at coordinates: 3,3
12 Player 2 places an O at coordinates: 4,0
13 Turn 5:
14 Player 1 places an X at coordinates: 4,4
15 The final board is:
16 -----
17 |X| | | | | | | | | |
18 |-----|
19 |O|X| | | | | | | | |
20 |-----|
21 |O| |X| | | | | | | |
22 |-----|
23 |O| | |X| | | | | | |
24 |-----|
25 |O| | | |X| | | | | |
26 |-----|
27 | | | | | | | | | |
28 |-----|
29 | | | | | | | | | |
30 |-----|
31 | | | | | | | | | |
32 |-----|
33 | | | | | | | | | |
34 |-----|
35 | | | | | | | | | |
36 |-----|
37 | | | | | | | | | |
38 |-----|
39 | | | | | | | | | |
40 |-----|
41 | | | | | | | | | |
42 |-----|
43 | | | | | | | | | |
44 |-----|
45 | | | | | | | | | |
46 |-----|
47 Player 1 wins
48

```

Figure 14: Write the result to file



5 References

1. [Assignment Specification](#)
2. [MIPS Reference](#)
3. [MIPS Instruction Set](#)
4. [Read file instruction](#)
5. [Phụ lục MARS](#)
6. [Gomoku](#)
7. [Cờ ca-rô](#)