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

COMPUTER ARCHITECTURE - CO2007

ASSIGNMENT REPORT

FOUR IN A ROW - A NEW VERSION

Advisor: Bang Ngoc Bao Tam
Students: Hoang Tien Duc - 2152520

HO CHI MINH CITY, APRIL 2023

Contents

1 Assignment Requirements

Using MIPS to create a game named "Connect Four" with the following requirements:

- Using the text-based User Interface (input and output in the I/O section).
- Includes a report explaining how the program works.

2 Introduction

Four in a Row is the classic two player game where you take turns to place a counter in an upright grid and try and beat your opponent to place 4 counters in a row.

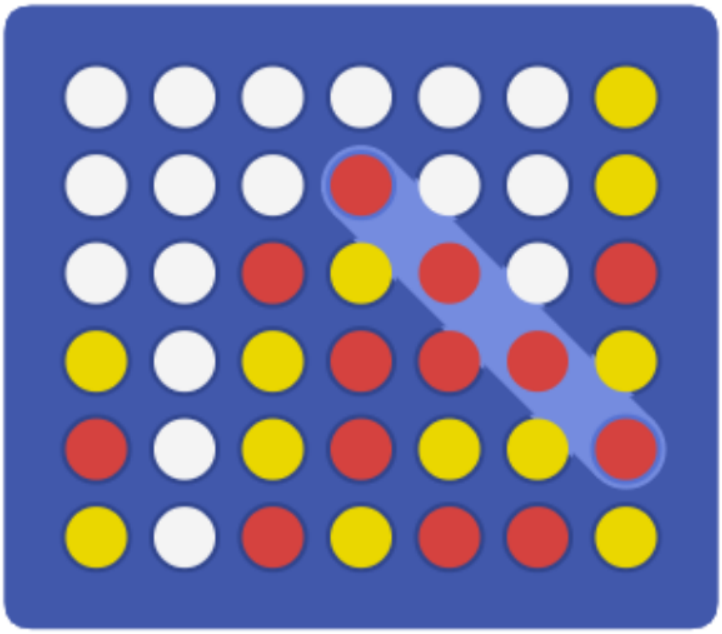

Figure 1: Four in a Row.

The game is played with a **seven-column** and **six-row** grid, which is arranged upright. The players choose their name, and a game piece (X or O) is assigned to them randomly. Then they will place a piece in column. Each player alternately takes a turn placing a piece in any column that is not already full. The piece fall straight down, occupying the lowest available spot within the column or be stopped by another piece. The aim is to be the first of the two players to connect four pieces of the same colour vertically, horizontally or diagonally (an example is shown in Figure 1). If each cell of the grid is filled and no player has already connected four pieces, the game ends in a draw, so no player wins.

3 Implementation

In this section, I would like to briefly illustrate how the program works in general, mostly explaining the algorithms and handling exceptions.

3.1 Create blank board - Random pieces

3.1.1 Input Username - Random pieces

On execution, the program first prints out a welcome message and requires 2 players to enter names. Next, the program randoms a number and stores in $\$a0$:

- $\$a0 = 1$ player 1 plays with O, player 2 plays with X.
- $\$a0 = 0$ player 1 plays with X, player 2 plays with O.

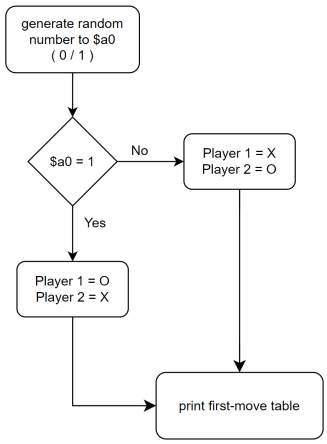

Figure 2: Flow Diagram for choosing pieces.

```
======================================================================
==================    [Welcome to Connect Four]     ==================
======================================================================

This is a turn-based 2-player game.
The rule is:
+ Each player put their designated piece into a 6 x 7 Board.
+ The game ends when either one of the player makes 4 consecutive.
+ Additional function for players: remove, undo and block.

Player One's name: Hoang Tien Duc
Player Two's name: Tien Duc Hoang

NOTE: Player 1 use 'X', and player 2 go with 'O'.
```

Figure 3: First output executed by the program.

3.1.2 Board Implementation

Since there are only 2 types of piece in the game, we can simplify the data type of every element inside our data array into integers (so the data type we use in MIPS will be **.word**). Therefore, I create a one-dimension array named **boardGame**, which consists of 42 elements and each element stores 1 of 3 integers:

- 0: No piece / empty cell
- 1: Piece of player 1
- 2: Piece of player 2

Index	0	4	8	..	160	164
Value	0	2	1	..	0	0

Figure 4: One-dimension **boardGame** array.

Initially, each element is assigned to 0, except the middle column (in the first move, players must drop in the 4th column).

140	144	148	152	156	160	164
112	116	120	124	128	132	136
84	88	92	96	100	104	108
56	60	64	68	72	76	80
28	32	36	40	44	48	52
0	4	8	12	16	20	24

a. Index of each element in the board

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	2	0	0	0
0	0	0	1	0	0	0

b. Data of each element in the board

Figure 5: **boardGame** array is represented in two-dimension.

To access the array, we need to get the value in 2D and solve in 1D. For a simpler procedure in usage for later algorithms, I form a relationship in index between 2 arrays. Let $a[m][n]$ be a normal 2D-array, and $b[m * n]$ is a linear array of the same size, and they share the exact same data in the same order. Then:

$$a[i][j] = b[164 - 28i + 4j]$$

3.1.3 Board Printing

Since our array is linear, the idea is to iterate through every element and print them out normally, and only break down to a new line every 7 elements. The implement flow is shown in Figure 6. Notice that, since each word is 4 bytes away each other in MIPS, so I actually need to decrease i by 4 every iteration to access the next element.

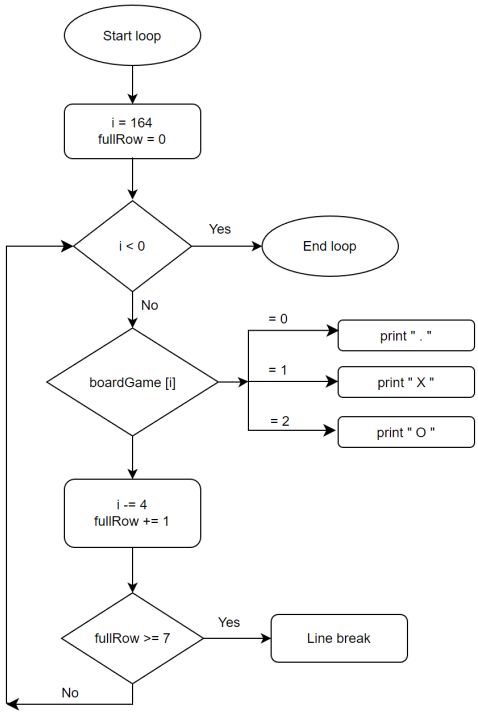

Figure 6: Flow Chart for printing board if player 1 choosing X.

```
REQUIREMENT: In the first move, players MUST place pieces in the middle column.


                -- BOARD GAME --

  | 1  2  3  4  5  6  7
-------------------------
1 | .  .  .  .  .  .  .
2 | .  .  .  .  .  .  .
3 | .  .  .  .  .  .  .
4 | .  .  .  .  .  .  .
5 | .  .  .  .  .  .  .
6 | .  .  .  .  .  .  .
```

Figure 7: Print table before starting the game.

3.2 Pre-dropping

3.2.1 Player Information

Before each player drops a piece, the program is required to print an overview of the player, including name, faults, and undo. The output is similar to Figure 8.

```
========================== [In Progress] ===========================
Player 1: Hoang Tien Duc
Piece: O            |           Fault: 0           |           Undo: 3
You have only ONE chance to block. Do you want to REMOVE one piece of your opponent? (0 = NO  |  1 = YES):
```

Figure 8: Player information is printed in each turn.

- If player wants to remove, Removing is proceed.
- Otherwise, he/she is asked to drop piece into the chosen column, which is implemented in Placing Piece.

3.2.2 Removing

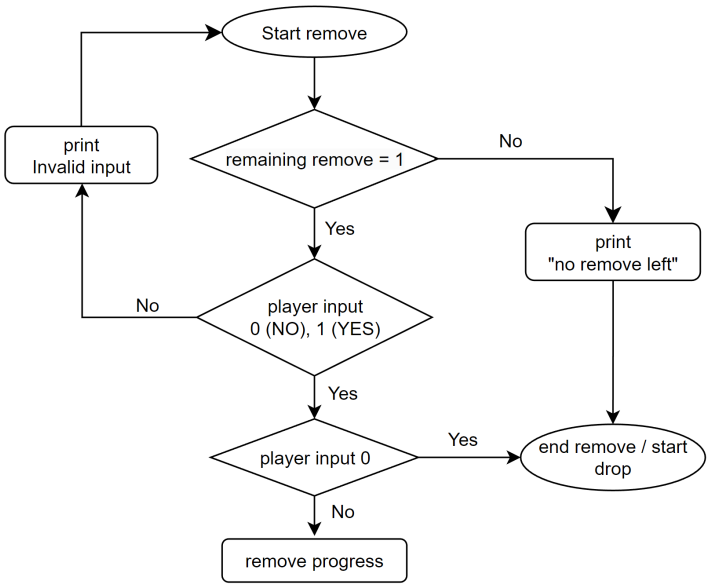

Figure 9: Flow Diagram whether player want to remove or not.

Figure 9 illustrates steps by steps for a player to remove an opponent's piece or not:

- When player chooses an inappropriate cell (a cell that contains his/her piece or an *empty cell*), he/she is required to choose position again.

- If either removal is ignored or player used removed, player will start placing pieces, which we will discuss in 3.3.
- Else, jump to remove progress.

```
You have only ONE chance to remove. Do you want to REMOVE one piece of your opponent? (0 = NO  |  1 = YES): 1
Please choose the ROW of piece you want to remove: 3
Please choose the COLUMN of piece you want to remove: 4

The place you choose do not have piece of opponent.
```

Figure 10: Player input incorrect position.

To begin the removal process, we first calculate the index of the element to be removed in the 1D array using the relationship established in section 3.1.2. Once the cell at the current index is set to 0, the subsequent step is to shift any cells above it downwards. To achieve this, I implement a loop that traverses the rows above the current row by incrementing the current index by 28, retrieving the value of the adjacent cell at the current index, and moving the current index up one row. This procedure is repeated until we approach the penultimate row. Finally, we set the value of the top cell in the column to 0 to indicate the empty cell. Nonetheless, after falling down, each cell above it must be sent to the Connect-Four Check to determine if there are any new *connect four* or *connect three*.

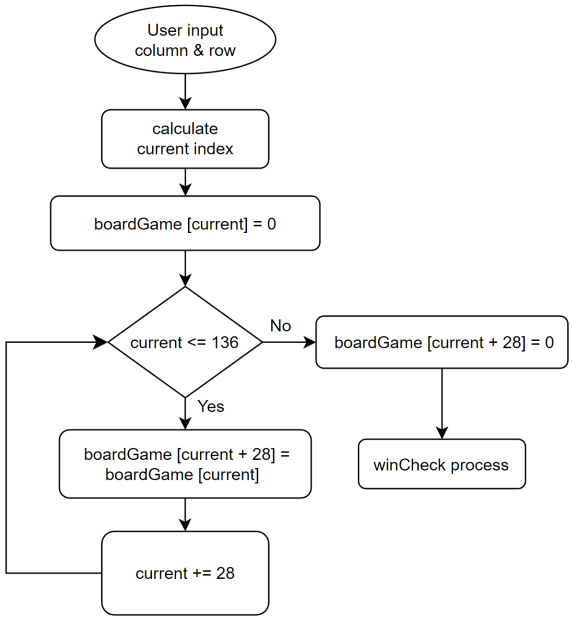

Figure 11: Flow Diagram for removing process.

An example of the whole process is described in Figure 12 and Figure 13.

```
----- BOARD GAME ------

  | 1  2  3  4  5  6  7
-----------------------
1 | .  .  .  .  .  .  .
2 | .  .  .  .  .  .  .
3 | .  .  .  O  .  .  .
4 | .  .  .  X  .  .  .
5 | .  .  .  O  .  .  .
6 | .  .  .  X  .  .  .
```

Figure 12: Before removing.

```
You have only ONE chance to remove. Do you want to REMOVE one piece of your opponent? (0 = NO  |  1 = YES): 1
Please choose the ROW of piece you want to remove: 5
Please choose the COLUMN of piece you want to remove: 4

 ----- BOARD GAME ------

   | 1  2  3  4  5  6  7
 ----------------------
 1 | .  .  .  .  .  .  .
 2 | .  .  .  .  .  .  .
 3 | .  .  .  .  .  .  .
 4 | .  .  .  O  .  .  .
 5 | .  .  .  X  .  .  .
 6 | .  .  .  X  .  .  .
```

Figure 13: After removing.

3.3 Placing Piece

Every turn, the player going in that turn has to input an appropriate number as the column number to make his/her move. Additionally, as stated in the rules, if the number of faults exceeded 3, that player is forced to lose and the game ends. Input is invalid when:

- In the first turn, players do not drop pieces in the middle column.
- It is not a valid column number (it is outside the range [1, 7] or is not even an integer).
- It is a valid column but the column is already full (There are 6 pieces in that column).

Hence, we have the flow diagram based on such information, described in Figure 14. As the diagram illustrated, we first checked the first requirement (input is in range) and only when it is satisfied, we move on to examine the second one (column is not full). With the valid input, we then update the value at the suitable location, *current index* is loaded and then print the board.

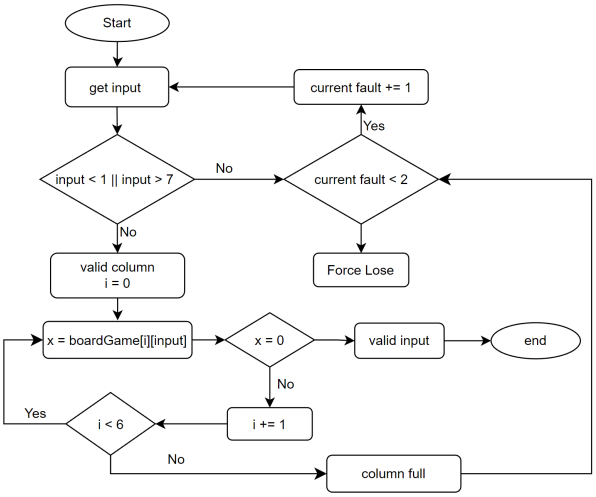

Figure 14: Flow Diagram validate input.

```
========================== [In Progress] ============================
Player 1: Hoang Tien Duc
Piece: O            |           Fault: 0           |           Undo: 3
You have only ONE chance to remove. Do you want to REMOVE one piece of your opponent? (0 = NO  |  1 = YES): 0
Please choose a column from 1 to 7 to DROP your piece: 9

WARNING: You chose INVALID column. Let's restart your turn.

========================== [In Progress] ============================
Player 1: Hoang Tien Duc
Piece: O            |           Fault: 1           |           Undo: 3

Please choose a column from 1 to 7 to DROP your piece: 1

                        -- BOARD GAME --

                      | 1  2  3  4  5  6  7
                    ------------------------
                    1 | .  .  .  .  .  .  .
                    2 | .  .  .  .  .  .  .
                    3 | .  .  .  .  .  .  .
                    4 | .  .  .  .  .  .  .
                    5 | .  .  .  X  .  .  .
                    6 | O  .  .  O  .  .  .
```

Figure 15: Program output when dropping pieces.

```
========================== [In Progress] ============================
Player 1: Hoang Tien Duc
Piece: O            |           Fault: 0           |           Undo: 0

Please choose a column from 1 to 7 to DROP your piece: 5

WARNING: You chose INVALID column. Let's restart your turn.

========================== [In Progress] ============================
Player 1: Hoang Tien Duc
Piece: O            |           Fault: 1           |           Undo: 0

Please choose a column from 1 to 7 to DROP your piece: 4

                          -- BOARD GAME --

                        | 1  2  3  4  5  6  7
                      ------------------------
                      1 | .  .  .  .  .  .  .
                      2 | .  .  .  .  .  .  .
                      3 | .  .  .  .  .  .  .
                      4 | .  .  .  .  .  .  .
                      5 | .  .  .  .  .  .  .
                      6 | .  .  .  O  .  .  .
```

Figure 16: Dropping pieces in the first turn.

3.4 Post-dropping

3.4.1 Undo

The first task in **postDrop** is undo function.

- Control user to input valid number (0 for *ignore* and 1 for *accept*).
- If a player does not want to undo movement, or the remaining undo equals to 0, move to Connect-Four Check.
- Otherwise, the process for undoing the previous move is as follows:
  1. Set element of at *current index / latest* to be empty.
  2. The remaining undo of the current player decreases by 1.
  3. Print the board to view the latest update.
  4. Since the board does not change until the next round, Connect-Four Check is examined to check after each player's turn.
- However, please note that undo can only be utilized during the middle game. Therefore, during the first turn of the two players, the number of undos will be 0, and it will be set to 3 from the second round onwards.

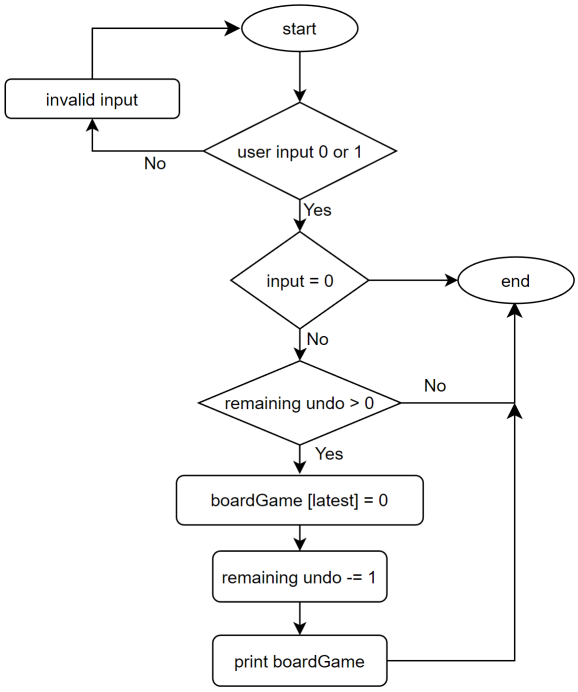

Figure 17: Flow Diagram for Undo.

```
Do you want to UNDO your move? (0 = NO  |  1 = YES): 2

WARNING: Invalid input. Please enter again.


Do you want to UNDO your move? (0 = NO  |  1 = YES): 1

                          -- BOARD GAME --

                        | 1  2  3  4  5  6  7
                       -----------------------
                       1 | .  .  .  .  .  .  .
                       2 | .  .  .  .  .  .  .
                       3 | .  .  .  .  .  .  .
                       4 | .  .  .  .  .  .  .
                       5 | .  .  .  X  .  .  .
                       6 | .  .  .  O  .  .  .
```

Figure 18: Undo movement of Figure 15.

3.4.2 Connect-Four Check

In this part, my idea is: Every time the board receives change (remove, drop), the latest change position (or at *current index*) must create/eliminate a win chance. In detail:

- Placing a piece at a precise location can be a winning move.
- A piece that is removed can lead to losing a winning chance or creating a new opportunity.

Hence, we only need to check the sequences around *current index*. The region that we need to check is illustrated in Figure 19. There is a total of 4 directions, meaning 4 subtasks that I will

140	144	148	152	156	160	164
112	116	120	124	128	132	136
84	88	92	96	100	104	108
56	60	64	68	72	76	80
28	32	36	40	44	48	52
0	4	8	12	16	20	24

Figure 19: Range checking at index = 72.

examine respectively: **Horizontal, Vertical, Backward Slash Diagonal** (from Upper-left to Lower-right) and **Forward Slash Diagonal** (from Lower-left to Upper-right). In each subtask, every direction has a left part and a right part with the *current index* in the middle. Hence we will start from *current index* and moving away from it, counting the number of elements storing the same pieces with *current index*. The loop will stop when:

- It counted 3 valid pieces, which means we meet the conditions, hence terminate the loop and jump to the endGame section.
- It either encounters a piece of a different type or meets the border of the Board (e.g. when the iterator is at column 7 but wants to move to the right). We then terminate the loop and reset the counting variable to 1, and move to the next subtask.

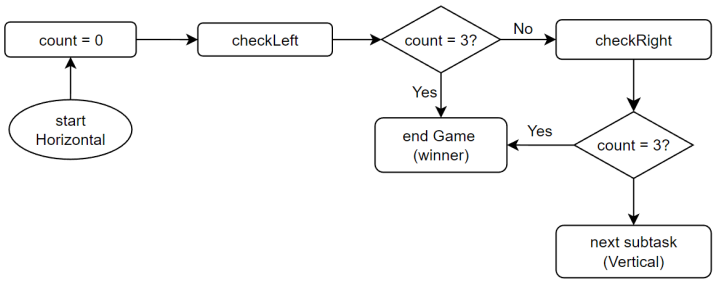

Figure 20: Flow Diagram for Horizontal subtask.

It has no matter where the location is on the 2D array, if we increase or decrease its index in 1D array representation by a specific value, we can have the same location as if we iterate it by 2 parameters - row and column. Figure 21 shows how to access each element for subtask process:

- **Horizontal**: simply minus 4 or add 4 to move to the left or to the right.
- **Vertical**: add or subtract 28 to go down/up.
- **Foward Slash Diagonal**: add 32 to go to the right, and subtract 24 to go to the left.
- **Backward Slash Diagonal**: add 24 to go to the left, and subtract 32 to go to the right.

...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	+24	+28	+32	...
...	...	...	-4	current	+4	...
...	...	...	-24	-28	-32	...
...	...	...	...	...	...	...

Figure 21: How to traverse index in each subtask.

Following the completion of the aforementioned four subtasks, the program proceeds to check if the board is completely filled. This is achieved by examining the top row of the game board, where the program checks the seven elements to determine if they are empty or not. If all seven elements are not empty, then the game is considered a draw and the program proceeds to the Game Over! phase.
Furthermore, whenever the counting variable encounters the value of 2, it indicates that the player with the game piece at *current index* has a chance to win, necessitating a prohibition on the opponent's block. Figure 21 demonstrates the procedure for obtaining the index of the elements to be checked in the four subtasks.

3.4.3 Block

The final task before the opponent's turn is block function.

- If the opponent has a chance to win, which is recognized in 3.4.2, skip to the next turn.
- Control user to input valid number (0 for *ignore* and 1 for *accept*).
- If a player does not want to block, or the remaining block equals to 0, the opponent is allowed to play.
- Otherwise, the current player can play one more turn.

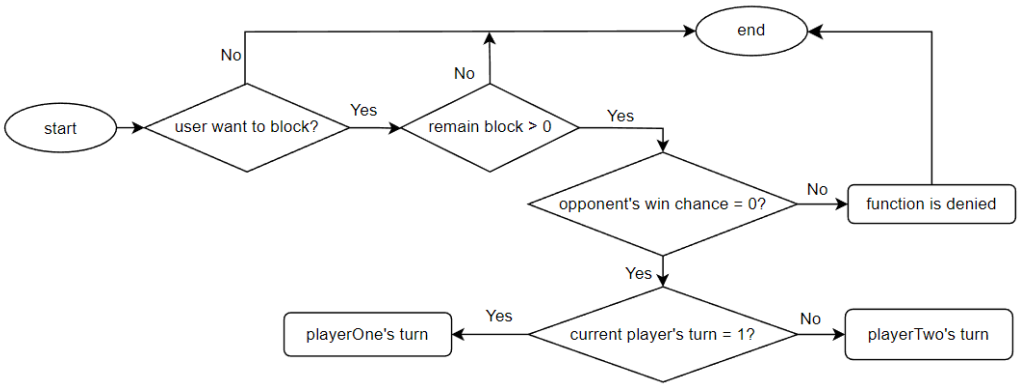

Figure 22: Flow Diagram for blocking opponent.

3.5 Game Over!

Once Connect-Four Check has determined a winner, the program will print out the name of the winner, the type of game piece (if applicable), and the number of the winning piece. Figure 23 provides an illustration of what this output will look like.

```
========================== CONGRATULATION! ==========================

Player 2 is the winner.
- Player's name: Tien Duc Hoang
- Type of piece: O
- Number of pieces: 4
  ----- BOARD GAME ------

    | 1  2  3  4  5  6  7
  ----------------------
  1 | .  .  .  .  .  .  .
  2 | .  .  .  O  .  .  .
  3 | .  .  .  O  .  .  .
  4 | .  X  .  O  .  .  .
  5 | .  X  .  O  .  .  .
  6 | .  X  .  X  .  .  .
```

Figure 23: Program finds a winner.

On the other hand, if both players fill the entire board but neither has achieved a winning position, the game is considered a draw, and the final game board will be printed to provide a visual representation as Figure 24.

```
This is a Tie game. Let's start a new game to find the winner.

----- BOARD GAME ------

  | 1  2  3  4  5  6  7
-----------------------
1 | O  X  O  X  O  X  O
2 | O  X  O  X  O  X  X
3 | O  X  O  X  O  X  O
4 | X  O  X  O  X  O  X
5 | X  O  X  O  X  O  O
6 | X  O  X  O  X  O  X
```

Figure 24: An example of a tie game.

3.6 Handling Exception

To handle exceptions, it is advisable to avoid using system calls that allow users to input incorrect data types that may cause the program to stop. Instead, I use the syscall 12 for user inputs (except entering names), which limits the user to input only one character, significantly reducing the validation process and the player also do not need to press ENTER. If the user input is invalid, the program can jump back to the previous stage and prompt the user to perform the action again until the program receives the desired input. Once we need to convert a character to an integer, we can follow the ASCII rule by subtracting 48 from the character's ASCII value.

References

[1] *Four in a Row*, AI Gaming, https://help.aigaming.com/game-help/four-in-a-row.

[2] *Connect Four*, Wikipedia, https://en.wikipedia.org/wiki/Connect_Four.

[3] *ASCII*, Wikipedia, https://en.wikipedia.org/wiki/ASCII.