HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE

ASSIGNMENT REPORT

COMPUTER ARCHITECHTURE - CO2007

FOUR IN A ROW

Instructors:	Assoc. Prof. PhD. Pham Quoc Cuong
	M.Sc. Bang Ngoc Bao Tam

Lab class:	CC08
Conducted by:	Nguyen Phan Tri Duc

(Contact information: 0938906033, duc.nguyen@hcmut.edu.vn)

Ho Chi Minh City, 11/2022

MỤC LỤC

I.IDEA

1. Game board

Base on the idea of the game is played with a seven-column and six-row grid, for this assignment, we will use an array *BOARD[42]* to represent the board. Each array element is 1 byte, the values are **0/1/2** for **[blank] / [first_player_character] / [second_player_character]** respectively where the characters can be 'X' or 'O' depends on the character the first player want to choose.

This is the index table from 0 to 41 showing how the array is being printed.

5	11	17	23	29	35	41
4	10	16	22	28	34	40
3	9	15	21	27	33	39
2	8	14	20	26	32	38
1	7	13	19	25	31	37
0	6	12	18	24	30	36

For example, the table of

[								]
[								]
[		O	O					]
[		X	X	O				]
[		O	X	X	O			]
[	X	O	O	O	X	X		]

where ‘X’ is the first player character and ‘O’ is the second player character will have the value in the memory as

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	**2**	**2**	0	0	0	0
0	**1**	**1**	**2**	0	0	0
0	**2**	**1**	**1**	**2**	0	0
1	**2**	**2**	**2**	**1**	**1**	0

Additionally, we also create an array *HEIGHT[7]* for storing the currennt height of each column. Initialize each *HEIGHT* elements to 0, the column is full when value of the element in the corresponding index is 6.

For the table above, the current values of elements in *HEIGHT* are

1	4	4	3	2	1	0

Both *BOARD* and *HEIGHT* array are updated after each user INPUT, which will be discussed in the next part.

2. Input

The game is simple on how user input data, for each move, player just need to input their desired column to drop the symbol in.

For natural language, when printing the board, instead of 0 to 6, the columns are indexed from 1 to 7, therefore user will choose a number from that range, called ***n***.

Process ***n***:

• ***n*** *=* ***n*** *- 1 (change the range from [1,7] to [0,6])*

• Check if ***n*** is in range [0;6], if false then input again *(out of range error)*.

• Check if *HEIGHT[**n**]* < 6, if false then input again *(full column error)*.

• Update *BOARD[6*n+HEIGHT[n]] (or BOARD[n][HEIGHT[n]] if use 2D array)*

to the value of corresponding symbol.

• Update *HEIGHT[**n**] = HEIGHT[**n**]* + 1

For the two error mentioned, at each move, if a player's input column raise the error, the player will have **3** more chances to input the valid column. If the player violates all of the chances, that player will be force to lose the game result in the other player's win.

After each success move, the COUNT variable of the game will increase by 1, if COUNT increase to 42 then we know that the BOARD is full without anyone winning, leads to draw between 2 players.

3. Undo after each move.

Each player also has 3 chances to undo after each move, before confirm to end their move to the next player. The order of the procedure is described in the flowchart below.

After any move, there is a variable $v1 store the index **i** on the BOARD of the recent move. The undo procedure will do its work base on that value, including:

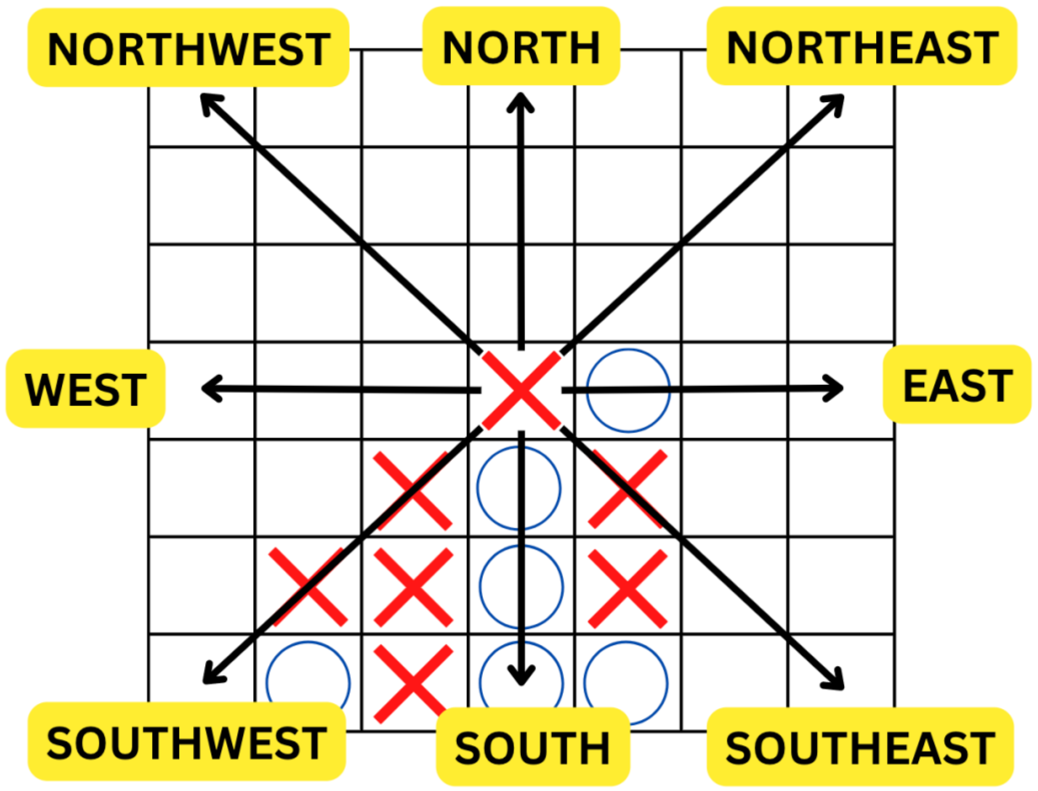
- Set BOARD[i] to 0 again.
- Do HEIGHT[column(i)] = HEIGHT[column(i)] – 1
- Do countUndo = countUndo – 1 (correspond to the player that use undo)

4. Check if the player has won on the recent move

To win the game, the player needs to be the first to connect four pieces of their symbol vertically, horizontally or diagonally.

For our program, after each move, the program will loop through all of the elements in the *BOARD* from 0 to 41, if any of the elements connects 4 vertically, horizontally or diagonally **at any direction** succesful with the other elements, that player will win the game.

As a result we have 8 directions to check on each point on the board, named **NORTH**, **SOUTH**, **EAST**, **WEST**, **NORTHEAST**, **NORTHWEST**, **SOUTHEAST**, **SOUTHWEST**. However, we don't need to check all 8 directions, because if there exist one point that successfully satisfied **NORTH** line, then surely there will also exist the point at the other end that satisfied **SOUTH** line.

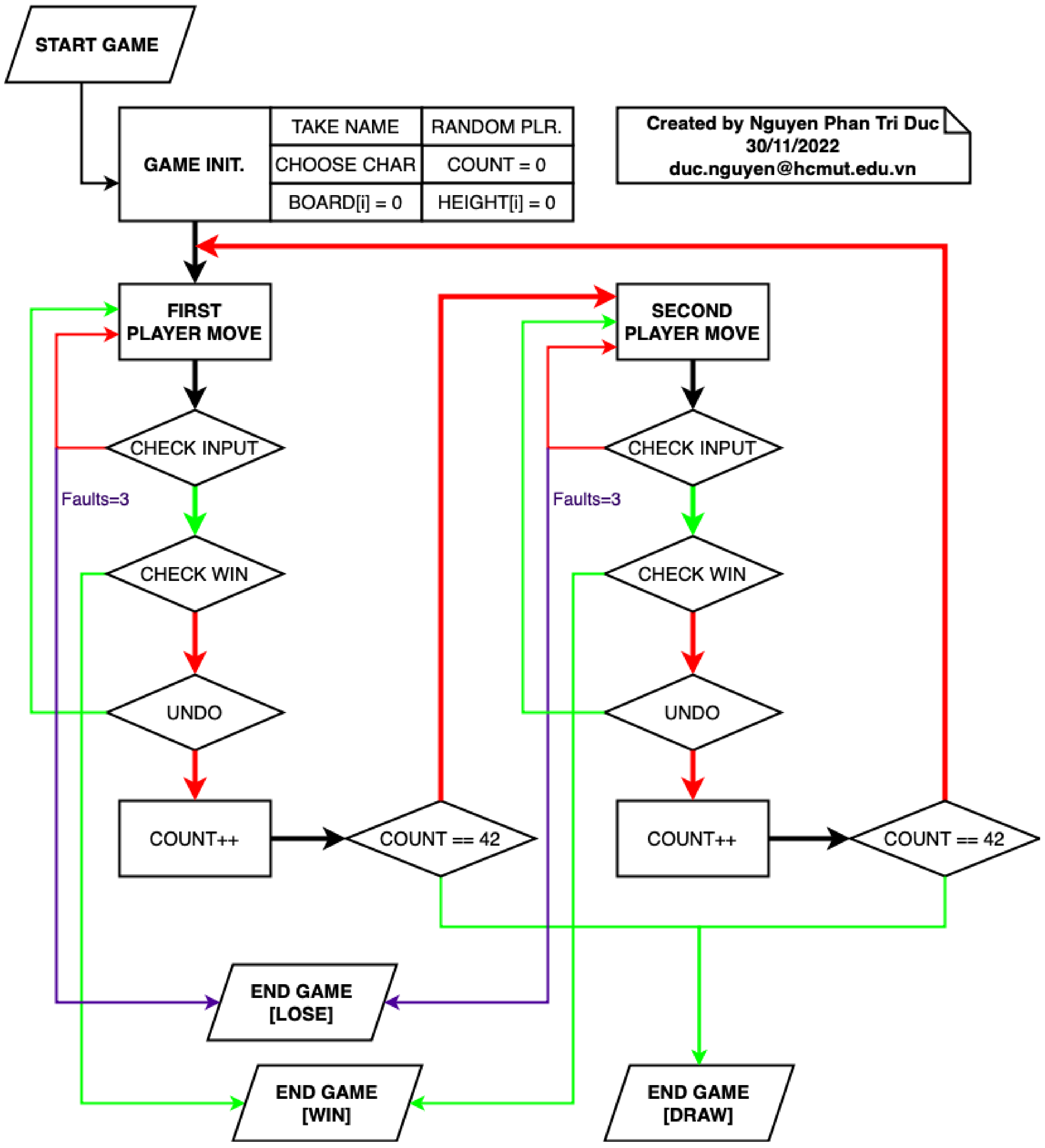


The idea above shorten the check algorithm to just 4 directions to check: **NORTH**, **EAST**, **NORTHEAST**, **NORTHWEST**, also we don't need to check if the current point have different symbol from the symbol of the recent player.

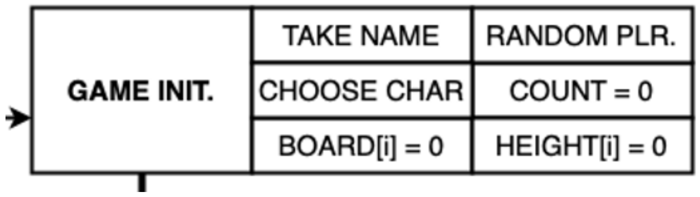
The indexing idea makes traversing on the board easy. Call the index of the current point **i**, the **NORTH** index is **i+1**, the **SOUTH** index is **i-1,** the **EAST** index is **i+6,** the **WEST** index is **i-6**.

- After traversing **NORTH**, if new index **j** % 6 == 0 means that we have gone outside of the board.
- After traversing **SOUTH**, if new index **j** % 6 == 5 means that we have gone outside of the board.
- After traversing **EAST**, if new index **j** > 41 means that we have gone outside of the board.
- After traversing **WEST**, if new index **j** < 0 means that we have gone outside of the board.

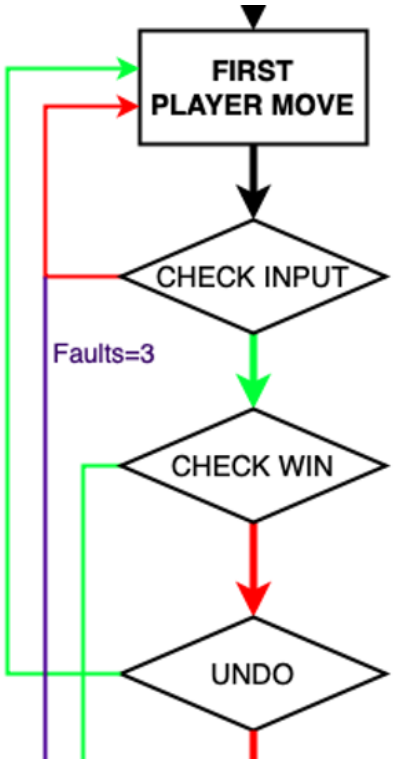
5. Flowchart


START GAME
GAME INIT.
TAKE NAME
RANDOM PLR.
CHOOSE CHAR
COUNT = 0
BOARD[i] = 0
HEIGHT[i] = 0
Created by Nguyen Phan Tri Duc
30/11/2022
duc.nguyen@hcmut.edu.vn
FIRST PLAYER MOVE
SECOND PLAYER MOVE
CHECK INPUT
Faults=3
CHECK WIN
UNDO
COUNT++
COUNT == 42
CHECK INPUT
Faults=3
CHECK WIN
UNDO
COUNT++
COUNT == 42
END GAME [LOSE]
END GAME [WIN]
END GAME [DRAW]

Detailed explanation:



First, the game will be initialized. In this procedure, the game will ask the players to input their names, then randomize the chosen player to be the first player to move, also that player can choose their character 'X' or 'O'. The other player must stick with the remain character. Also we initialize BOARD, HEIGHT and COUNT mentioned above to 0 for all values.



After that, as the player moves, the program checks the input number and check win like the idea above, then ask if the player wants to undo their recent move or confirm to end their turn.

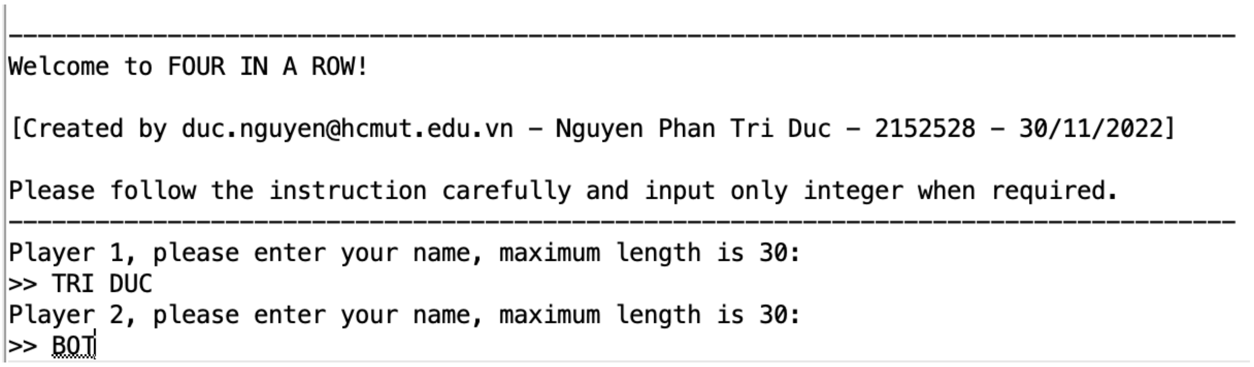
The program will update COUNT after each move, check if the game is draw, then switch to another player turn.

• If the player violates the input more than 3 times, the game will make that player **lose**. (purple branch)

• If the player move result in *true* in the CheckWin function, the game will make that player **win**.

• If the COUNT variable goes to 42, the game will result in **draw**.

II.INTERFACE



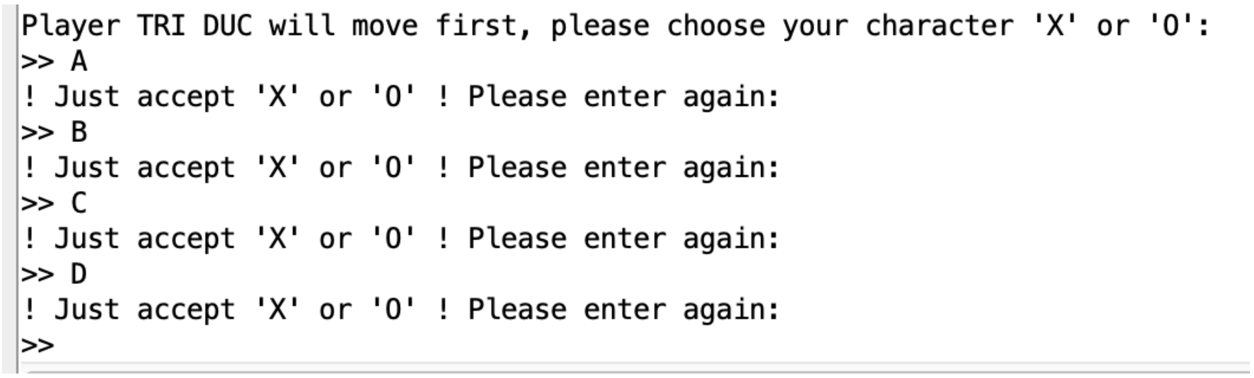
```
---------------------------------------------------------------------------------
Welcome to FOUR IN A ROW!

[Created by duc.nguyen@hcmut.edu.vn - Nguyen Phan Tri Duc - 2152528 - 30/11/2022]

Please follow the instruction carefully and input only integer when required.
---------------------------------------------------------------------------------
Player 1, please enter your name, maximum length is 30:
>> TRI DUC
Player 2, please enter your name, maximum length is 30:
>> BOT
```

A. Input UI

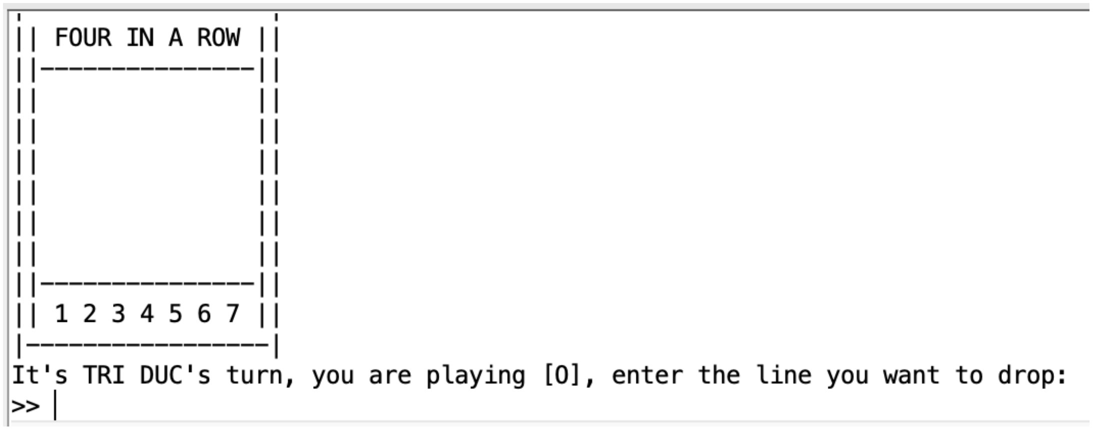
The program will display **[>>]** if it needs user input.



```
Player TRI DUC will move first, please choose your character 'X' or 'O':
>> A
! Just accept 'X' or 'O' ! Please enter again:
>> B
! Just accept 'X' or 'O' ! Please enter again:
>> C
! Just accept 'X' or 'O' ! Please enter again:
>> D
! Just accept 'X' or 'O' ! Please enter again:
>>
```

B. False Input

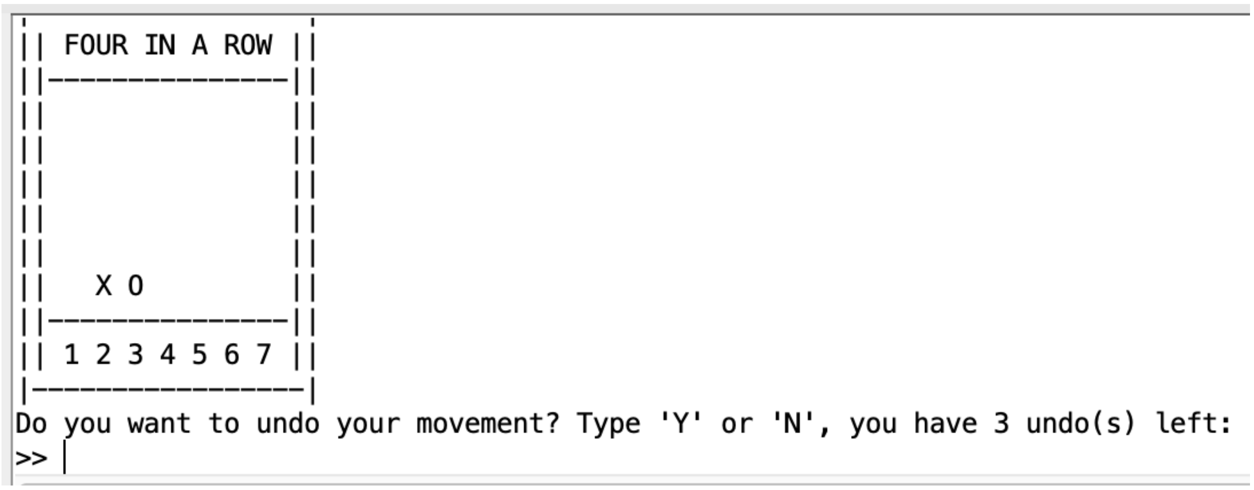
The program will ask to type again if input is invalid.



```
|| FOUR IN A ROW ||
||---------------||
||               ||
||               ||
||               ||
||               ||
||               ||
||               ||
||---------------||
|| 1 2 3 4 5 6 7 ||
|-----------------|
It's TRI DUC's turn, you are playing [O], enter the line you want to drop:
>> |
```

C. The Board showing the game

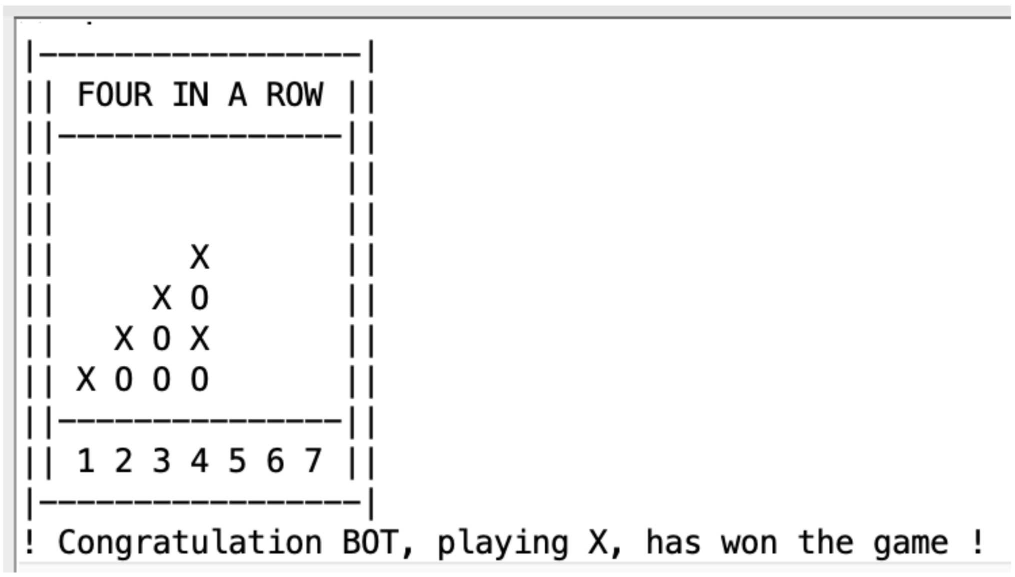
D. Undo



```
|| FOUR IN A ROW ||
||---------------||
||               ||
||               ||
||               ||
||               ||
||               ||
||   X O         ||
||---------------||
|| 1 2 3 4 5 6 7 ||
|-----------------|
Do you want to undo your movement? Type 'Y' or 'N', you have 3 undo(s) left:
>> |
```

After each move, if the player still have undo chance(s), the program will ask if they want to undo, if the user input an unvalid character, it also raise error and ask to input again.

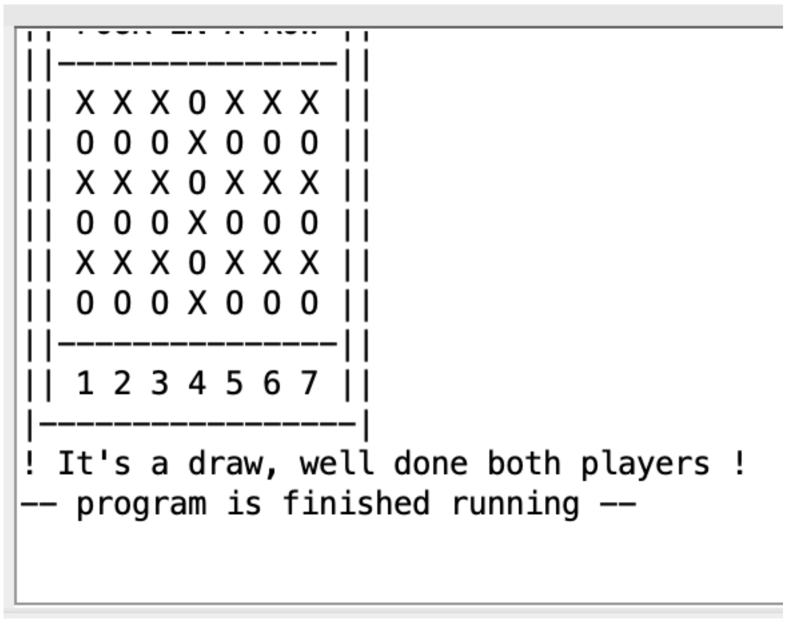
E. Win



```
|-----------------|
|| FOUR IN A ROW ||
||---------------||
||               ||
||               ||
||       X       ||
||     X O       ||
||   X O X       ||
|| X O O O       ||
||---------------||
|| 1 2 3 4 5 6 7 ||
|-----------------|
! Congratulation BOT, playing X, has won the game !
```

After a move that makes the corresponding player win, the program will display who have won then end the program.

F. Draw


```
||---------------||
|| X X X O X X X ||
|| O O O X O O O ||
|| X X X O X X X ||
|| O O O X O O O ||
|| X X X O X X X ||
|| O O O X O O O ||
||---------------||
|| 1 2 3 4 5 6 7 ||
|-----------------|
! It's a draw, well done both players !
-- program is finished running --
```

After the last move that makes the table full, the program will result in draw then finish running.

- THE END -