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

COMPUTER ARCHITECTURE LAB
(CO2008)

Assignment (Semester: 221, Duration: 06 weeks)

"FOUR IN A ROW"

Instructor: MSc. Bang Ngoc Bao Tam

Student's Name: Le Trong Hieu
Student's ID: 2153342

HO CHI MINH CITY, DECEMBER, 2022

Contents

1 Introduction **2**
1.1 Four In a Row . . . 2
1.2 Assingment: Four In a Row . . . 2

2 Ideas and Algorithm **3**
2.1 How the game works? . . . 3
2.2 Functions' explanation . . . 5
2.2.1 PlayerPick . . . 6
2.2.2 InitBA and ExitInitBA . . . 6
2.2.3 main . . . 6
2.2.4 DrawFullBoard . . . 7
2.2.5 StoreInput . . . 8
2.2.6 Redo . . . 9
2.2.7 Winner . . . 9
2.2.8 WinnerCheck . . . 10

3 Conclusion **13**

1 Introduction

1.1 Four In a Row

Figure 1: Four in a row toy

Four in a row, also known as Connect Four, is a two-player connection board game, in which the players choose a color and then take turns dropping colored tokens into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the lowest available space within the column.

The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own tokens. Connect Four is a solved game. The first player can always win by playing the right moves. The game is draw if the all the element is filled but no one has won.

1.2 Assignment: Four In a Row

For the assignment, we need to implement the game using MARS mips, the rules to win/lose/draw is kept as original.

We will also implement some methods other than the game's winning conditions:

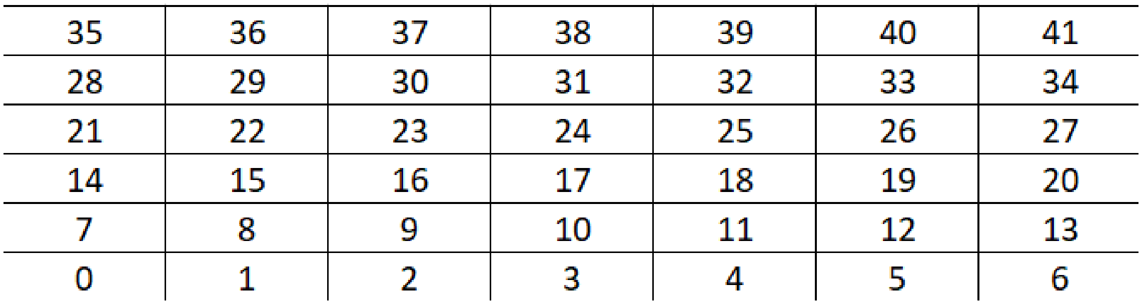
- Randomly choose X, Y for player A, player B
- Each player has 3 chances to redo their moves after deciding the column
- Each player has 3 chances to continue making moves if they violated moves(not in the range from the first to the seventh column)
- Count the scores for each players

Further explanation will be discussed further in section 2.

2 Ideas and Algorithm

2.1 How the game works?

In my work, I use an array of 42 bytes to display the Board Array. Particularly, it can be interpreted as a board with 6 rows, each rows have 7 columns as below:



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

Figure 2: Index Array

NOTE: The array above are filled with 0, the figure only described the indexing and how the array can be looked so the reader can understand easier (We see the array as a folding of 6 parts).

To traverse, store, check user's input, calculate the position, I used modulo to obtain result.

The way the code works follows the Flow chart below:

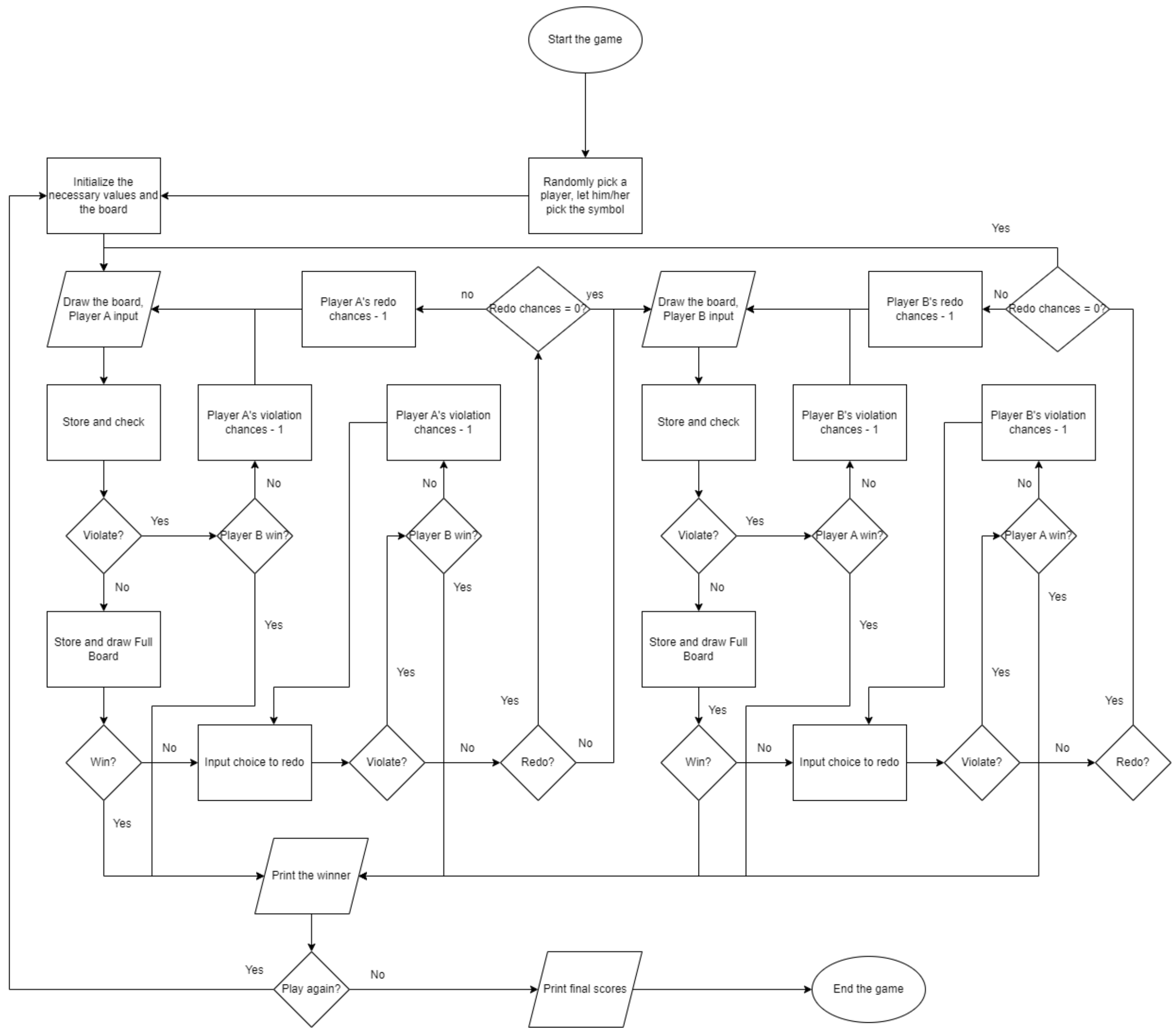
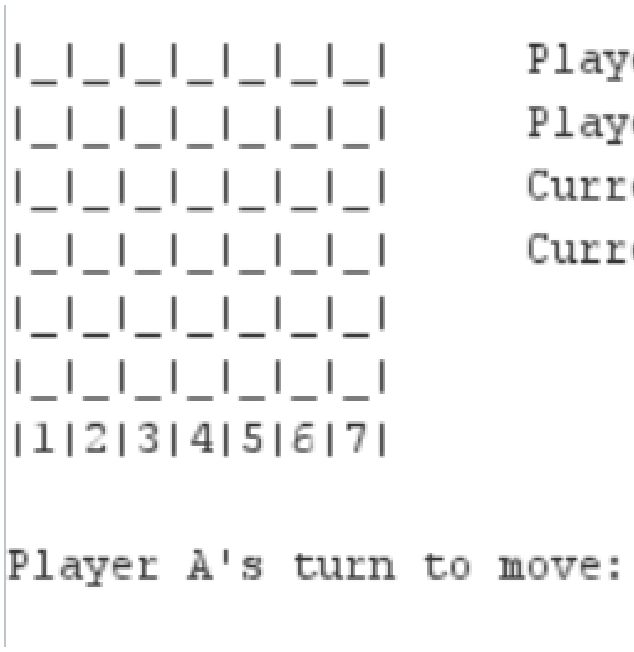

Start the game
Randomly pick a player, let him/her pick the symbol
Initialize the necessary values and the board
Draw the board, Player A input
Store and check
Violate?
Yes
No
Player B win?
Player A's violation chances - 1
Player A's redo chances - 1
Redo chances = 0?
no
yes
Store and draw Full Board
Win?
Input choice to redo
Redo?
Draw the board, Player B input
Player A win?
Player B's violation chances - 1
Player B's redo chances - 1
Print the winner
Play again?
Print final scores
End the game

Figure 3: How the code functions

For the code, it requires many functions (and sub-functions) to perform wells, I will summarize the main functions here:

- PlayerPick: Pick randomly a player and let them choose X or O.
- Initialize: To start a new game
- InitBA and ExitInitBA: Initialize Board Array with full of 0's. Also, it initializes 3 chances for violation and redoing for each player.
- main: The main function of the game that will be used, it asks for users' inputs, store them, draw the Board, check if the player that entered won, ask if they want to redo and move to the other player's turn.
- DrawFullBoard: Draw the full board as below:



```
|_|_|_|_|_|_|_|     Player A: O
|_|_|_|_|_|_|_|     Player B: X
|_|_|_|_|_|_|_|     Current Score: 0-0
|_|_|_|_|_|_|_|     Current turn: Player A
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|1|2|3|4|5|6|7|


Player A's turn to move:
```

Figure 4: Game's Table

- StoreInput: Store the input position from user to the array, check and return the latest move
- Redo: Redo the latest move of the current player
- Winner: Print the one who win and ask if they want to play a new game
- WinnerCheck: Check 4 direction to see if the current move is a win/draw.

The following section will describe each of the above functions thoroughly.

2.2 Functions' explanation

Before immersing into the functions explanation, I will quickly mention the global variable that I use in the code:

- boardArray: the array
- \$s1, \$s2: X or O randomly for Player A and Player B respectively
- \$s3, \$s4: Chances left when violating the rules for Player A and Player B respectively
- \$s5, \$s6: Chances left to redo the move for Player A and Player B respectively
- \$s0, \$s7: Points for Player A and Player B respectively
- \$a2: Current turn
- \$a3: Latest move as index of array
- Some other strings

2.2.1 PlayerPick

This is the simplest function in the code. It just simply random the player and use if else branch to assign the symbols to the players.

Listing 1: Pseudocode for PlayerPick

```
PlayerPick():
    x = random in {0,1} #random from [1,3)
    if x = 0: #Player A
        print PlayerAChoose
        y = player A input # 0 -> O, 1 -> X
        if y = 0: #Choose O
            $s1 = 0
            $s2 = 1
        else: #Choose X
            $s1 = 1
            $s2 = 0
    else:    #Player B
        print PlayerBChoose
        y = player B input
        if y = 0: #Choose O
            $s1 = 0
            $s2 = 1
        else: #Choose X
            $s1 = 0
            $s2 = 1
```

2.2.2 InitBA and ExitInitBA

Actually, The InitBA is a loop to "clean" the game table when starting a new game. The ExitInitBA is used to assign chances.

Listing 2: Pseudocode for InitBA and ExitInitBA

```
    i = 0
    InitBA():
        if i == 42:
            goto ExitInitBA()
        else:
            array[i] = 0 #set the board to 0
            i++
            InitBA()
    ExitInitBA(): #Setting the chances to player
        $s3 = 3
        $s4 = 3
        $s5 = 3
        $s6 = 3 #3 chances for all cases for each player
        $a2 = 1 #The first turn is of the playerA
```

2.2.3 main

The main function is a loop of two functions inputA() and inputB(). The sub-functions will be described later.

The flow-chart for this main function is exactly Figure 3

Listing 3: Pseudocode for main()

```
main():
    DrawFullBoard()
    x = userA input #the column
    $a3 = StoreInput(1,x-1) #check possible slots, violation,...
    #$a3 = StoreInput(1,x-1) return the lates move here
    # x - 1 because we use [0,41]
    DrawFullBoard()
    WinnerCheck(1, $a3) #Check if player A won
    if(SureA()): #check if the user is sure about their move
        inputB() #continue the turn for player B
    else:
        redo(1,$a3)
inputB():
    DrawFullBoard()
    x = userB input #the column
    $a3 = StoreInput(2,x-1) #check possible slots, violation,...
    #$a3 = StoreInput(2,x-1) return the lates move here
    # x - 1 because we use [0,41]
    DrawFullBoard()
    WinnerCheck(2, $a3) #Check if player B won
    if(SureB()): #check if the user is sure about their move
        main() #return to main() to make a loop
    else:
        redo(2,$a3)
```

2.2.4 DrawFullBoard

To implement this function, we firstly need to define the DrawRow and DrawNumber functions:

DrawRow will take in the begining position of the row i as input and and print the 7 consecutive position.

For example: DrawRow(7) will print boardArray[i], $i \in$ [7,13].

Similarly, DrawNumber will print number from 1 to 7 to make the game interface more friendly

Listing 4: Pseudocode for DrawRow and DrawNumber

```
    DrawRow(i): #i is $a0 in mips
        for i in range(i,i+7):
            print '|' #Used to seperate cells
            if boardArray[i]==0:
                print '_' #not yet reached
            else if boadrdArray[i] == 1: #X
                print 'X'
            else: #O == 0
                print 'O'
        print '|' #to enclosed
    DrawNumber():
        for i in range(1,8):
            print '|'
            print i
        print '|' #enclosed
```

Then, the code for DrawFullBoard is pretty simple, we print the array from top to bottom

Listing 5: Pseudocode for DrawFullBoard

```
    DrawFullBoard():
        DrawRow(35)
        Print Player A symbol, endl
        DrawRow(28)
        Print Player B symbol, endl
        DrawRow(21)
        Print current scores between A and B, endl
        DrawRow(14)
        Print the current turn, endl
        DrawRow(7), endl
        DrawRow(0)
        DrawNumber()
```

2.2.5 StoreInput

For this function, I firstly describe the Violate function to check for violation move. Its input in MIPS is:

- $a0: Which player

Listing 6: Pseudocode for Violation

```
Violation(player): #player is player A or B
    if player = 1:
        if $s3 = 0: #No chances left for A
            Conclude A lost the game
        else:
            $s3 =$s3 - 1 #chances--
            print Threat
            main() #return to main() for A to input
    else:
        if $s4 = 0: #No chances left for B
            Conclude B lost the game
        else:
            $s4 = $s4 - 1 #chances--
            print Threat
            inputB() #return to inputB()
```

For the StoreInput, it takes two inputs in MIPS

- $a0: which player (A = 1 or B = 2)
- $a1: which column (should be $\in$ [1,7])

Generally, this function will "rehash" to find empty slot given the column that the user has entered. It does that by adding 7 or "moving up" to the cell above until find empty cell or violation because the column is full.

Listing 7: Pseudocode for StoreInput(player, column)

```
StoreInput(player, column): #player = $a0, column = $a1
    if column < 1 || column > 7:
        Violation(player)
    else:
        i = column
        while i <= 41:
            if boardArray[i] == 0: #in MIPS, we need to load byte
                boardArray[i]=player #Store 1 or 2 to the array to
    indicate it is reached
                return i #latest move as indexes of array
            else:
                i = i+7 #go to the upper row to see if it is empty
        Violation(player) #this means that column is full(i>=42), the
    player has violated the rule
```

2.2.6 Redo

This method takes two inputs:

- \$a0 = which player (A = 1 or B = 2)
- \$a3 = latest move (Note this is array index, $\in$ [0,41])

Listing 8: Pseudocode for Redo(player, index)

```
Redo(player, index): #$a0 = player, $a3 = index
    if player == 1:
        if $s5 == 0:
            print player A has no choice left
            InputB() #B continue to input
        else:
            $s5 = $s5 - 1
            boardArray[index]=0 #Reset that index to 0
            DrawFullBoard()
            main() #go to main for A to input again
    else:
        if $s6 == 0:
            print player A has no choice left
            main() #A continue to input
        else:
            $s6 = $s6 - 1
            boardArray[index]=0 #Reset that index to 0
            DrawFullBoard()
            InputB()
```

2.2.7 Winner

This function take in one input \$a0 (A = 1 or B = 2) which is the player who won

Listing 9: Pseudocode for Winner(player)

```
Winner(player): #player = $a0
    if player == 1:
        print Player A won
        print Next game ?
        x = user input #1 = play next, else = endgame
        if x == 1:
            Initialize() #Restart the game with new values
        else:
            EndGame()
    else:
        print Player B won
        print Next game ?
        x = user input #1 = play next , else = endgame
        if x == 1:
            Initialize() #Restart the game with new values
        else:
            EndGame()
EndGame():
    print final results
    print thanks
    exit
```

2.2.8 WinnerCheck

Perhaps this is the most important functions and the core of the code. This function take in 2 inputs:

- $a0: which player
- $v0: latest move as index of array (I used this global variable instead of $a1,2,3)

This function will traverse for direction of the input Index: Vertical, Horizontal, Upward Diagonal, Downward Diagonal, and draw.

The pseudocode is long. So to summarize, the way this function works follows the flow chart below:

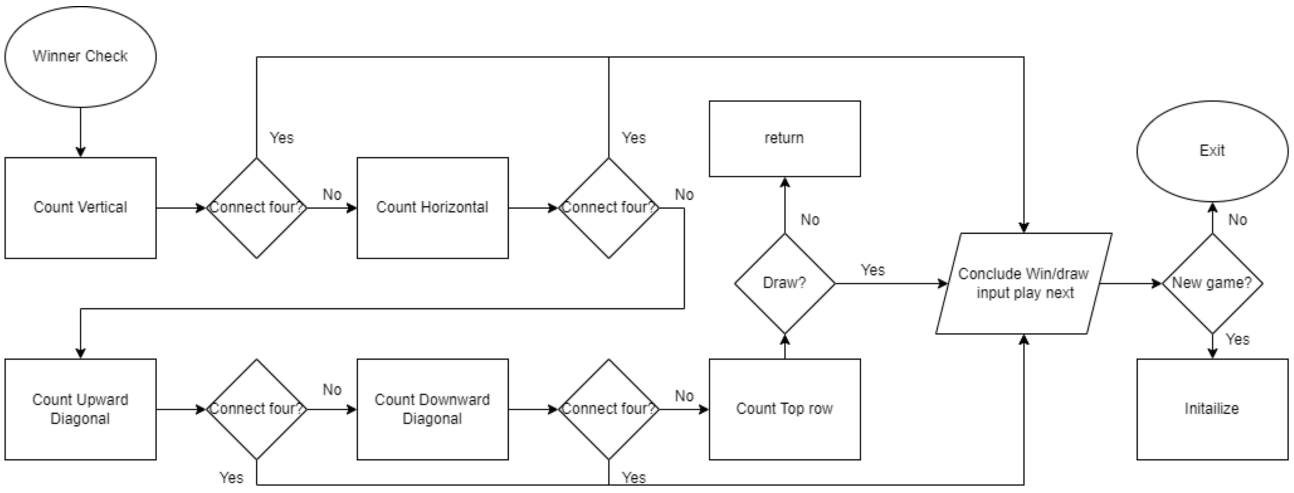


Figure 5: WinnerCheck

To calculate the index, I used modulo to make it easier. Particular description will be depicted in the pseudocode below:

Listing 10: Pseudocode for WinnerCheck(player, Index)

```
WinnerCheck(player, Index): #$a0 = player, $a3 = index
    #-----Check vertical-----#
    count = 1 #because that index is counted firstly
    i = Index
    #Check up
```

```
 while i <= 34: #we go up until the index is > 34 as this is the top
row, cannot go higher
     i = i + 7 #go to the cells above the current cell
     if boardArray[i] != player:
         break
     else:
         count = count + 1
         if count > 3:
             Winner(player)
 #Check down:
 i = Index
 while i >= 7: #This mean we are at the bottom row, cannot go down
anymore
     i = i - 7 #Go to the cell below the current cell
     if boardArray[i]!=player:
         break
     else:
         count = count + 1
         if count > 3:
             Winner(player)
 #-----Check horizontal-----#
 #Check left
 count = 1
 i = Index
 while i % 7 != 0: #if i % 7 == 0 then we are at the left most cell,
cant go left anymore
     i = i - 1 #go left
     if boardArray[i] != player:
         break
     else:
         count = count + 1
         if count > 3:
             Winner(Player)
 #Check right
 i = Index
 while i % 7 != 6: # if i%7==6, we are at the right most cell, cant
go right anymore
     i = i + 1 # go right
     if boardArray[i] != player:
         break
     else:
         count = count + 1
         if count > 3:
             Winner(Player)
 #-----Upward diagonal-----#
 i = Index
 count = 1
 #Check up right
 while !(i > 34 || i%7 == 6): #if this is satisfied, we are at
rightmost column or top row, we cannot go on that diagonal anymore
     i = i + 8 # go to the right cell of the cell above
     if boardArray[i] != player:
         break
     else:
         count = count + 1
         if count > 3:
             Winner(Player)
 #Check down left
 i = Index
 while !( i < 7 || i%7 == 0): #if this is satisfied, we are at left
most column or bottom row, we cannot go on that diagonal anymore
     i = i - 8 # go to the left cell of the cell below
     if boardArray[i] != player:
         break
     else:
         count = count + 1
         if count > 3:
             Winner(Player)
 #-----Downward diagonal-----#
 i = Index
 count = 1
 #Check Up left
 while !(i>34 || i%7==0): #if this is satisfied, we are at leftmost
column or top row, we cannot go on that diagonal anymore
     i = i + 6
     if boardArray[i] != player:
         break
     else:
         count = count + 1
         if count > 3:
             Winner(Player)
 i = Index
 #Check down right
 while !(i < 7 || i%7 == 6): #if this is satisfied, we are at
rightmost column or bottom row, we cannot go on that diagonal anymore
     i = i - 6
     if boardArray[i] != player:
         break
     else:
         count = count + 1
         if count > 3:
             Winner(Player)
 #-----DRAW GAME-----# only happen if all cells are filled
 #We only need to check the top row:
 for i in range(35, 42):
     if boardArray[i] == 0:
         return
 #Conclude Draw
 print Draw Game
 print Next game ?
 x = user input #1 = play next, else = endgame
 if x == 1:
 Initialize() #Restart the game with new values
 else:
     EndGame()
```

3 Conclusion

These are some main functions, algorithms and ideas that I implemented for my FOUR IN A ROW game using MARS mips. Althougth, truly, there are also other functions that are not mentioned, their roles are not so significant or used for breaking the loops. Therefore, I will not mention them in my report.

Thanks to the project, I had many opportunities to get used to mips code, implementing functions, nested functions and being able to classify code so that they look "clean". But most importantly, I finally knew how to play this game.

Besides, there were also many difficulties, such as coming up with the idea for drawing the game board, storing input using modulo for array. One of my regrets is that I am still afraid to use recursion in my code as I fear of not handling the $sp and $ra well. I sincerely wish to have more opportunities to deal with MARS mips again in a near future.

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.

——————END——————