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

COMPUTER ARCHITECTURE (CO2007)

Assignment

Advisor: Phạm Quốc Cường
Lê Trọng Đức - 2152523

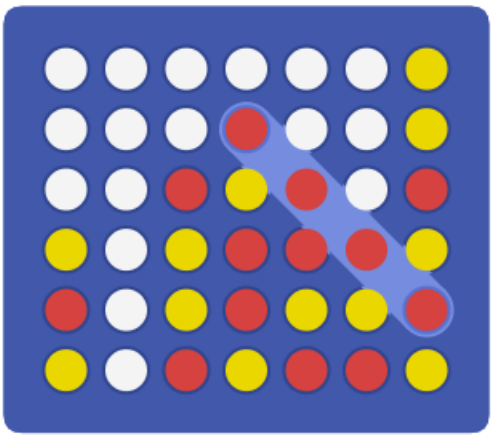
HO CHI MINH CITY, DECEMBER 2022

Contents

FOUR IN A ROW

1 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. The game is played with a seven-column and six-row grid, which is arranged upright.
The starting player is randomly chosen, pick a game piece color (yellow or red) and can place a piece in any column. Each player then 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 . 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.



Hình 1: Four in a Row

2 Interface

2.1 Briefly introduction

For the interface, we shall use the **Bitmap Display** tool to create color and board game for our "4 in a Row".
We will set up a 512×512 pixels board game. The display setting are:

- Unit Width in Pixels : 1
- Unit Height in Pixels : 1
- Display Width in Pixels : 512
- Display Height in Pixels : 512
- Base address for display : 0x10010000 (static data)

2.2 Drawing the background

Our map is like an array with 512×512 elements, hence, to color our background, we will use a loop to run from 0 to 262144, and perform store the color `blue` into each element.

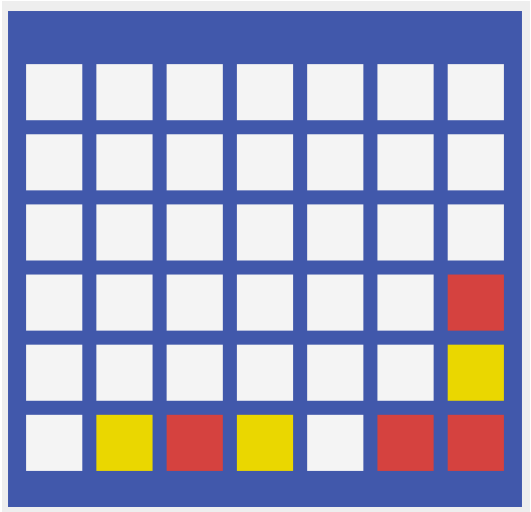
2.3 Drawing square function

To draw a square (which is a cell in our board game), the key point of here is to find where the begin pixel is. And in our case, we use the center pixel to draw our square.
And after many experiments, we choose 56 pixels as the length of an edge for our square. The upper margin is 53 pixels, the left horizontal margin is 18 and the distance from a square to the next square on the same line or on the same columns is 14 pixels.

To draw a square, first we need to choose the color. Then we perform 2 nested loops, one to iterate the rows, the other to iterate through the columns. And when iterate through each cell we perform `sw` the color to that cell.

Moreover, to leverage our draw square function, we also implement into our function a part to calculate the center pixels. Thus, our drawing function takes only 3 parameters, one for the color, others two are for the coordinate of the square need to draw (x, y) with $x \in [0, 6]; y \in [0, 5]$.

Our result:

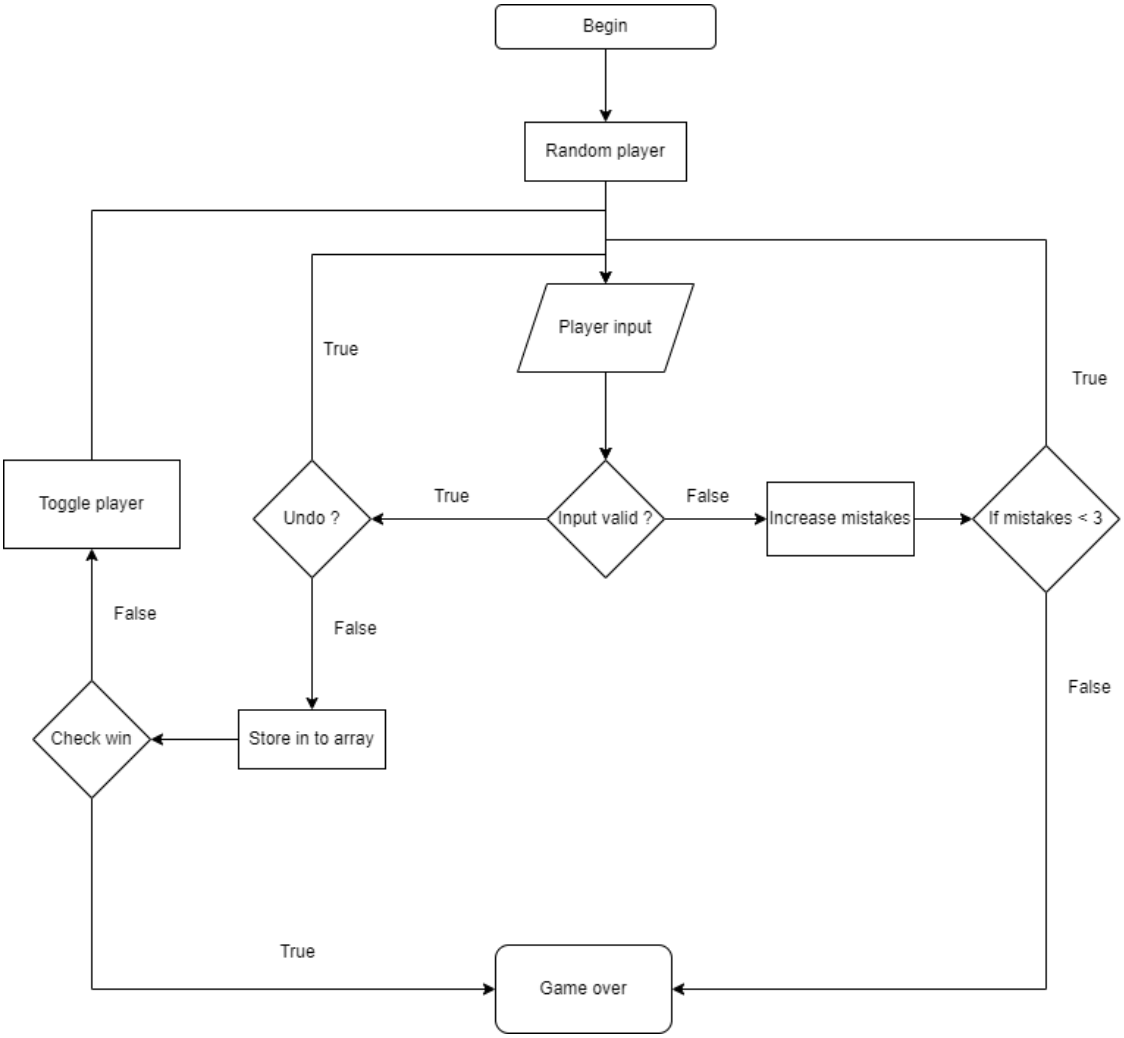


Hình 2: Result

3 Algorithm

3.1 Plotting the idea

The flowchart below illustrate some basic components of our program.



Hình 3: A simple flowchart of our program

3.2 Finding the appropriate data structure

Firstly, we need to think what appropriate data structure use to store the move of each player. In my case, I use one-dimension simple array, and our array has 42 cells. And we can imagine each cells in the table is a cell in our array, from bottom up, left to right, it will look like the figure 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

3.3 How to store the move of each player

To store all the move of both players, we use two arrays, each arrays has the capacity of 42 cells. And at each cell, we store the state of a cells according to the game board.
we use 3 states (values) to store for both array:

- State 0 : empty cell
- State 1 : that the cell we have chosen
- State -1: that the cell our opponent has chosen

Initially, we will set all the cells of both array of two players to 0. Then, each time they perform the move, we will update both array at the same time.
Every time a player input from 0 -> 6(example `x`), we will perform a loop from 0 to 5 (example `i`). In each loop we will take `i*7+x` and check if the cell `i*7+x` has the 0 value.

- If yes that cell value will change to 1 and the opponent's cell of that index will change to -1 and exit the loop.
- If no the loop will continue.
- If `i` has reached 6 but value at that cell still different from 0, we will handle the case Input in a full Column.

For instance, if player 1 goes for the cells 30, the cells 30 of player 1 will become 1, and the cell 30 of player 2 will become -1.

3.4 Handling the undo functionality

To handle the undo, we use 2 variables, initially both are set to 3, one stores the numbers of undo turns left for the first player, the other for the second player. Then, each time a player perform an undo move, his/her undo variable will be decreased by one. And when it reach -1, the player will lose.
Moreover, when a player perform a undo move, we will set the cell he/she has just chosen and the opponent's cell back to 0 (empty state), this will enable later move can go into that cell again.

3.5 Checking validity

Similar to the undo function, we also have 2 variables to store the mistakes of two players, initially store 0 and each time the player input an invalid move, we will not store it into his/her array but we will force he/she to move again, and each time like that his/her mistake variable will be increased by 1, and when it reach 3, that player will lose.

3.6 How to know the winning person

To check for the winning of each person, we check vertically, horizontally, up diagonal and down diagonal.

3.6.1 Checking vertical

To check the vertical, we will perform 2 loops, the outer loop will go from 0 to 2 (three rows from the bottom up), the inner loop will go from 0 to 6 (seven columns from left to right). Pseudo code:

```
    for(i=0 to 2)
        for(j=0 to 6)
            if(arr[7*i + j     ] == 1
            && arr[7*i + j + 7 ] == 1
            && arr[7*i + j + 14] == 1
            && arr[7*i + j + 21] == 1)
            return true;
```

3.6.2 Checking horizontal

To check the horizontal, we will perform 2 loops, the outer loop will go from 0 to 5 (six rows from bottom to top), the inner loop will go from 0 to 3 (four columns from left to right). Pseudo code:

```
    for(i=0 to 5)
        for(j=0 to 3)
            if(arr[7*i + j    ] == 1
            && arr[7*i + j + 1] == 1
            && arr[7*i + j + 2] == 1
            && arr[7*i + j + 3] == 1)
            return true;
```

3.6.3 Checking up diagonal

To check the up diagonal, we will perform 2 loops, the outer loop will go from 0 to 2 (three rows from the bottom up), the inner loop will go from 0 to 3 (fours columns from left to right). Pseudo code:

```
    for(i=0 to 2)
        for(j=0 to 3)
            if(arr[7*i + j     ] == 1
            && arr[7*i + j + 8 ] == 1
            && arr[7*i + j + 16] == 1
            && arr[7*i + j + 24] == 1)
            return true;
```

3.6.4 Checking down diagonal

To check the down diagonal, we will perform 2 loops, the outer loop will go from 0 to 2 (three rows from the bottom up), the inner loop will go from 3 to 6 (fours columns from 4th to 7th, left to right). Pseudo code:

```
    for(i=0 to 2)
        for(j=0 to 3)
            if(arr[7*i + j     ] == 1
            && arr[7*i + j + 6 ] == 1
            && arr[7*i + j + 18] == 1
            && arr[7*i + j + 24] == 1)
            return true;
```

3.7 Check draw

For checking the draw, we will iterate through an array of player 1, if 42 cells are different from 0 but the game still not end yet, the draw will happen to end the game.

4 Reference

Four in a Row, AI Gaming
Connect Four, Wikipedia
Simulating MARS Bitmap Display using Conditional Loops
CSC258 Assembly Programming Project: Doodle Jump