Vietnam National University, Ho Chi Minh City
University of Technology
Faculty of Computer Science and Engineering

Computer Architecture (C02007)

Four-in-a-row by MIPS

Instructor: Băng Ngọc Bảo Tâm
Student: Vương Hồng Lĩnh - 2152728

December, 2022

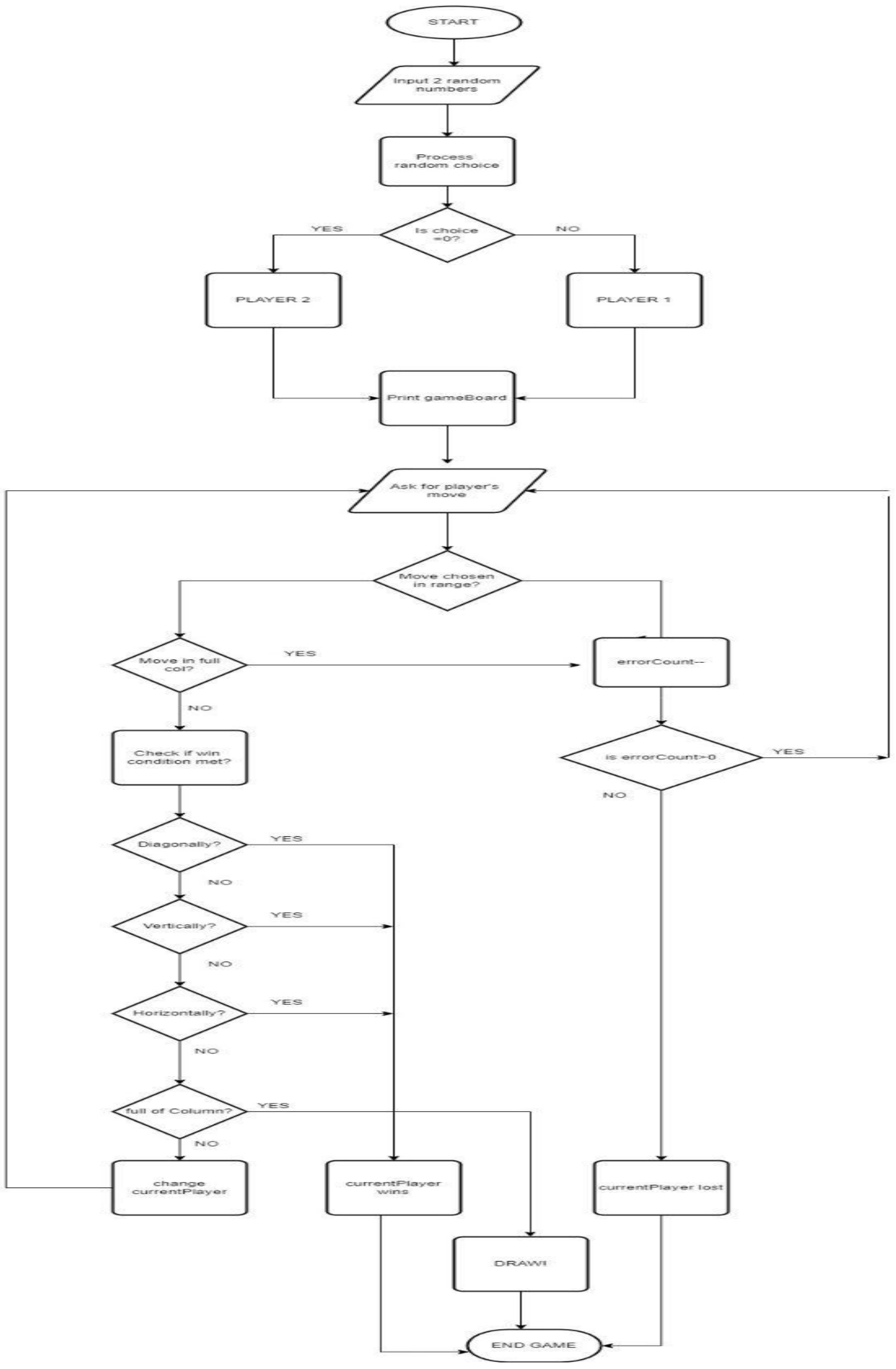
I. Gameplay explanation

- At the beginning of the game, each player enters a number they like. Then the system will automatically choose a player to go first.
- For each move, only enter a valid column index (1-7). If the input is invalid, or the chosen column is full, the player has to enter again. Once completed, the player's character of move (either X or O) will be "dropped".
- If a player's character makes consecutive four diagonally, vertically, and horizontally, that player wins the game.

II. Algorithm explanation

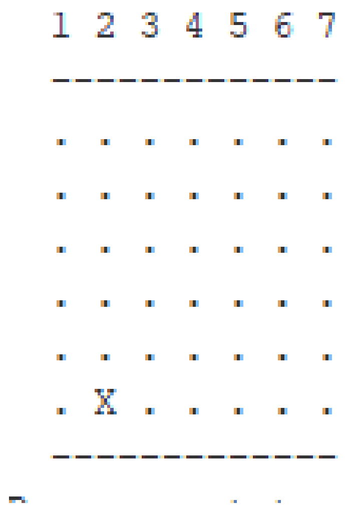
1. General algorithm:

-For full and sharp image quality, please visit:General Algorithm Diagram


START
Input 2 random numbers
Process random choice
Is choice =0?
YES
NO
PLAYER 2
PLAYER 1
Print gameBoard
Ask for player's move
Move chosen in range?
Move in full col?
YES
NO
errorCount--
is errorCount>0
YES
NO
Check if win condition met?
Diagonally?
YES
NO
Vertically?
YES
NO
Horizontally?
YES
NO
full of Column?
YES
NO
change currentPlayer
currentPlayer wins
currentPlayer lost
DRAW!
END GAME

2. **Form a gameboard (printGameBoard):**

a. Gameboard interface

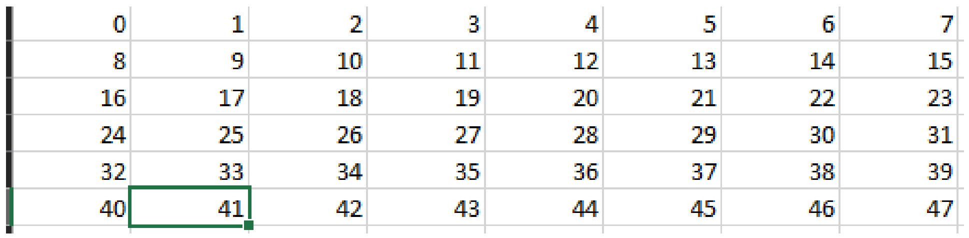


```
1 2 3 4 5 6 7
-------------
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. X . . . . .
-------------
```

b. Heap allocation

-In the first step, allocate 6 and 8 bytes respectively for rows and columns (1 exceeds bytes for columns containing “ ” character). To sum up, the board needs 48 bytes.

-The board should be visualized as indexed elements shown below



0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47

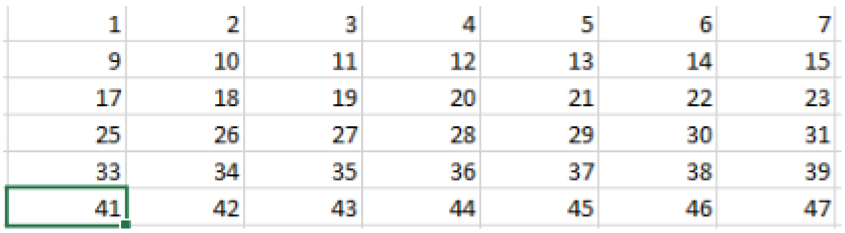
3. **Storing primary “.” character for each gameBoard[i][j]**

-At this point, create 2 “for-loop” (called outer-loop and inner-loop) to access each element in the “array”

-The access algorithm could be presented in the following pseudocode

```
for(int i = 0; i < row, i++)
    for(int j = 0; j < col; j++)
        address = i *COL
    Now  address = &gameBoard[i][j] (base+offset)
     Then load character
```

- The storing area in the heap is



1	2	3	4	5	6	7
9	10	11	12	13	14	15
17	18	19	20	21	22	23
25	26	27	28	29	30	31
33	34	35	36	37	38	39
41	42	43	44	45	46	47

4. **Storing space character to the left most column**
   -This process is similar to storing "." to each gameBoard element. Starting by accessing the first "row element" (0), gradually increase the index by 8, and assign that memory slot with space character.
5. **Random (random_in_range_2)**
   - This function uses system call number 42 to generate a number with lower bound is 0 and takes the value stores in register $a1 as the upper bound.
   - In the game, if the random number is 1, player 1 takes turn, else player 2
6. **User's move principal (UserMove & U2Move)**
   - When players input their dedicated column index, their move will be checked if valid, if not valid, check if it is their 3rd invalid move, and return the win for the other player. Once a move is valid, call isGameOver function to check if the move leads to that player's win.
   - Consider the userMove function for a player below:

   a. Check if valid:
      i. Check if out of bound: If move is not in the range (1-7), return error message.
      ii. Check if the move fills a full column: by keeping track of the column by colSpaceTrack (this is a 7-elements array, each cell's value is 5) , once a valid move will make colSpaceTrack[userCol -1] value decrease. If it comes to 0, it counts as an invalid move (column is full).

   b. Insert move to column:
      - Locate the position in the heap that the player wants to move by formula (colSpaceTrack[userInput - 1]) * COL+userInput (base + offset).
      - Load the current player character into that position.

7. **Check for winner(isGameOver)**

- This function uses a brute-force approach to check for character duplicates, hence returning the winner .
- There are 3 nested-loops in the function:
  + The first loop will be used to traverse through rows, assigned by i = 0
  + The second inner will be used to traverse each of the column, assigned by j = 0
  + The third loop will be used to count the number of duplicate elements (check for 4 consecutives). If this value reaches 4, jump to isWin function.
- Inside the third loop, check for elements that consecutively places horizontally, vertically, diagonally (left and right as well) respectively
- For each diagonal, horizontal, vertical check, if reach characters rather than the current player character, jump to the next type check. Else if k reaches 4, then jump directly to isWin, or the winCounter
- Approach for consecutive elements is different on each kind of check, with k running from 0 to 4
  + With horizontal ,check for element [i+k][j]. Each time that element is equal to the current player character, increase horizontalWinCounter by 1.
  + With vertical, check for element[i][j+k]. Each time that element is equal to the current player character, increase verticalWinCounter by 1.
  + With left diagonal, check for element [i+k][j-k]. Each time that element is equal to the current player character, increase leftDiagonalWinCounter by 1.
  + With right diagonal, check for element[i+k][j+k]. Each time that element is equal to the current player character, increase rightDiagonalWinCounter by 1.

8. **Check for draw:**
   Loop all over the heap space, if loop reaches character “.”, return to the main function. If all elements are visited without reaching any, return draw.

III. **Summary**:

All functions run properly, except for the missing undo function.

IV. **Reference**

1. Connect four: Connect Four (mit.edu)
2. Four in a Row, AI Gaming, https://help.aigaming.com/game-help/four-in-a-row.