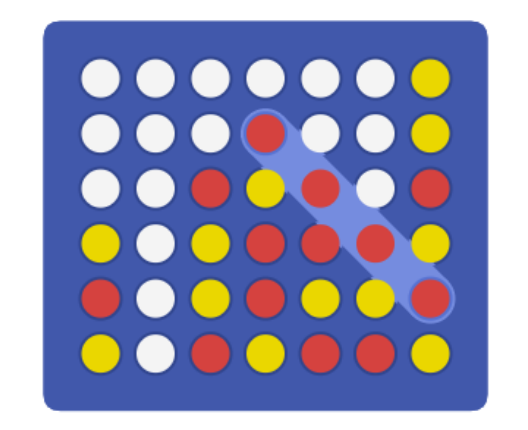
**TABLE OF CONTENT**

1. **INTRODUCTION**
2. **SOLUTION**
3. **Data Structure**
4. **Algorithms**

**A. INTRODUCTION**

**PROJECT NAME : FOUR IN A ROW**

****

Four in a row - An example

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.

**B. SOLUTION**

1. **Data structure**

The data structure in the program consists of 3 parts :

1. A set of arrays to construct the game’s board.

To my understanding, the board game is simply a 2 dimensional array. Therefore, one way to construct it is to create 6 rows of 7 words associate with a function to gain access to any location knowing its row and column number.

1. 2 variables to store the last inserted index and the current undoings/ rules violations of players.

To avoid the lack of variables when creating the main function, these 2 variables play a role as extra memory. This also help prevent the changing of their values when executing the function.

1. Strings to make the game more understandable.

These are the outputs that are used to interact with the user and help them play the game more easily.

1. **Algorithms**

The game lies mostly in the main function with the help of some external ones. In general, the game’s source code has 3 major part is initialize, user input and condition checking.

First is initializing the game. I used a syscall function to generate a random number between 1 and 2 to decide which player get to go first. Then this player will pick the piece he/she prefered and the other one will be left for the remain player. This part will also check if the player try to input any thing other than ‘x’ or ‘o’.

After that, the game begins with the first turn belongs to the player chosen in the last part. The game will use 4 variables to store the current player’s piece (symbol), number, undoings counter address and invalids counter address. The game will be a whole loop between label “play” and “otherwin” following the below steps :

1. Load all the information from the extra memory mentioned earlier into the 4 variables for the current turn.
2. Print the screen : print the board game and the 4 variables above.

The print screen function will traverse through out the game board – which is a set of 6 rows of 7 elements - using a function called get\_adrs having 2 variables indicates row and column for its input. This function, first multiply number of column by 4 after decreasing it by 1 and store to 1 variable (let’s call it var1). Then it create a temporary variable of value 0 (call var2) and traverse through all the row following these steps :

1. Increase var2 by 1.
2. Load address of row1 to a var3
3. Compare var2 to row input.
4. If they are equal go to step 9.
5. Increse var2 by 1.
6. Load address of row2 to var3
7. Compare var2 to row input.
8. If they are equal go to step 9.

-- Do the same for row3, row4, row5, row6 --

9. Add multiplication of column by 4 above mentioned to var3 and load a variable decided before.

The print function will go row by row, elements by elements and stop when traverse through all 42 cells of the table.

Another function in print screen is print other, this function basically loads all the values of violations and undoings of player given in the external source into its variabales and print them all out.

1. Print the number and piece of the current player.
2. Ask the player to choose a number in 1-7 as their desired-to-input column.

The input format is actually of char type. A normal approach will take inputs as integers but taking input as a char will prevent the player from input too many numbers, which is unnecessary because there are only 7 columns, save time and more importantly, let the program keep running even if the player input anything other than numbers.

This char will go through some code similar to that of get\_adrs to check if it is a number and return the integer value to the next block of code.

Then, it will go through a function to check for the fullness of the chosen column. The method of this function is that they will check for the top element of the column (row 1 of in the table). If there is a piece occupied here the player will get a penalty and the move will be restarted. Otherwise, we go ahead to the next steps.

Besides, as an input step, there will also be some lines of code to check the input of the user and give penalty if they try to violate.

1. Insert the piece into the game board.

The insertion into the table will consist of 2 steps. First we need to check for the available row and then we insert to that cell found available having its row and column.

To find the available row, our solution is to traverse from the last row to the first. If we encounter any cell that is empty, return its index. This function is assured to be successfully executed all the time because if the column is filled the move should have been restarted in the previous step.

Now that we have the column and row number, the function to insert the piece into the column should be easy. Having the index of the available cell, one call of function get\_adrs will gain us its address and we can immediately store the current player's piece.

1. Check if the player won the game

By the rules of the game, 4 connect pieces mean a win from the player. To resolve that problem, my solution is to traverse away from the recently input piece vertically, horizontally and diagonally in 2 directions and count for the number of pieces that are the same with the current piece. If the counter encounters any piece different to the current, stop traversing and combine the result from 2 directions. If the result is over 3 vertically, horizontally or diagonally, then the current player won, the loop is over and the game returns the result of the winner, else go to the next step.

1. Check if the table is filled.

To check fullness is an easy task. My solution to this problem is to check every column at row 1. If all of them are occupied by any word other than the empty word then the board must be filled, the game loop will be broken and bring the game to a draw result. Else go to the next step.

1. Switch the player's turn.

This solution includes combining with another function to load the address of some important variables of piece, violations and undoings of the next player. That function is placed at the beginning of the main game source so we just have to switch the player's number and jump back to the beginning of the game code.