2D Array and Recursion Practice

Lecture 23 – Spring 2019 – COMP110

Review Sessions

- Section 1 – Sunday, May 5th at 3:30pm in Sitterson 014
- Section 2 – Sunday, April 28th at 3:30pm in Sitterson 014

Warm-up Question #1

- Given a 2D array named **a**, to the right, what is the correct way to access the element in the array with a value of 8, assuming row-major order?

	0	1	2
0	1	4	7
1	2	5	8
2	3	6	9

Warm-up Question #2

- Given the 2D array *board* to the right, and the find function below, what is returned when **`find(2)`** is called?

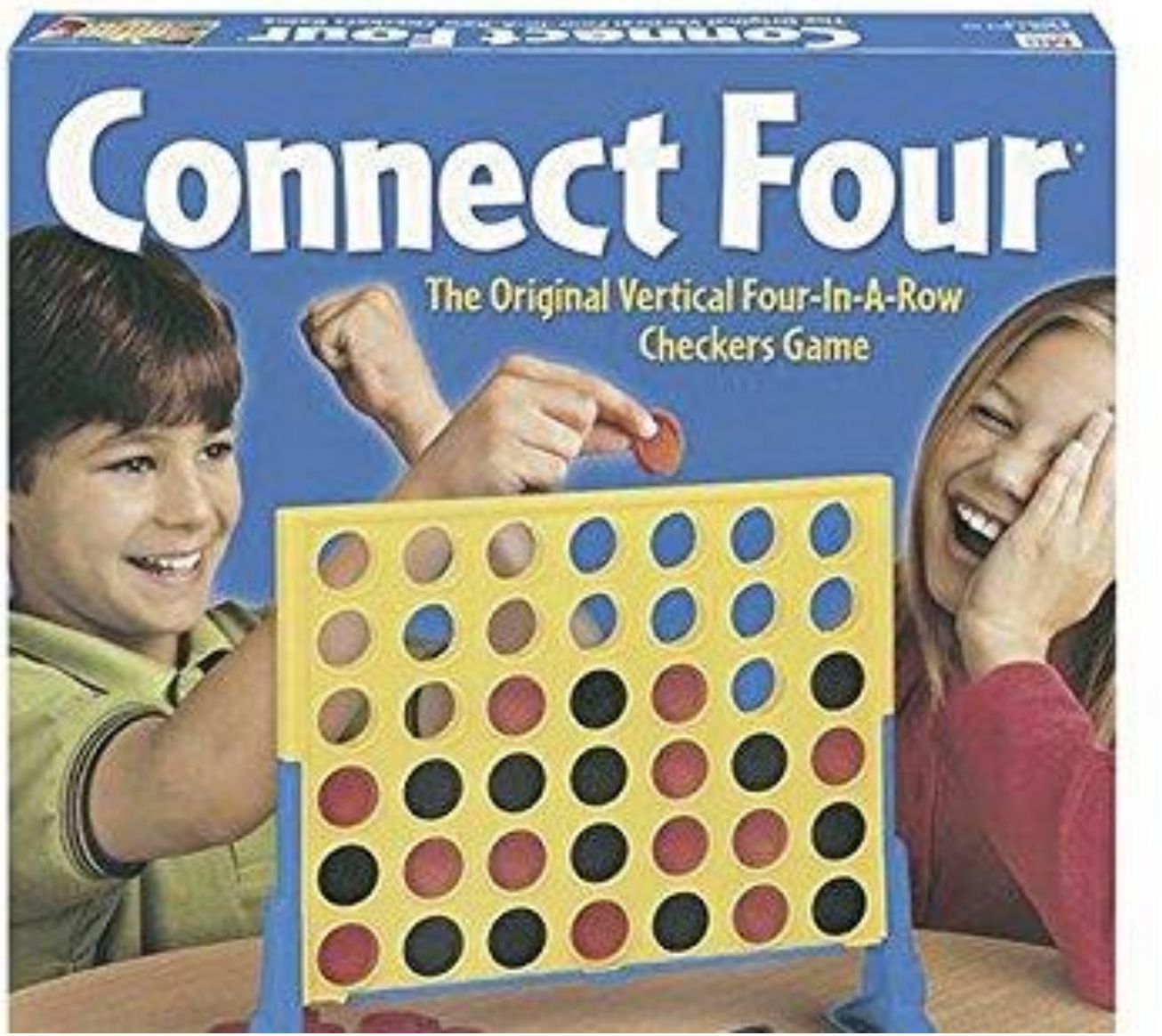
```
let find = (col: number): number => {
   for (let i: number = 2; i >= 0; i--) {
      if (board[i][col] === -1) {
            return i;
      }
   }
   return -1;
}
```

board:

	0	1	2
0	-1	-1	-1
1	0	-1	-1
2	1	0	-1

ConnectFour

- "Vertical Checkers"
- Players take turns dropping pieces in a column
- The first to create a streak of 4 of their pieces connected in a row wins
- Four-in-a-row can be:
  - Vertical
  - Horizontal
  - Diagonal



Stencil Code

- The stencil code sets up 2 global variables and some functions we'll implement for the game's logic

- Variables
  - board - 2D number array
    - -1 represents EMPTY
    - 0 represents player 0's piece
    - 1 represents player 1's piece
  - turn - number to keep track of how many turns have been played
    - turn % 2 - the player whose turn it is!

- Functions
  - play - Given a row and column, this function is called every time a position is clicked.

Follow-along #1) Simulating a "Drop"

- Currently, when a player clicks on any position their piece takes that position. We want it such that their piece gets placed at the last empty row in the selected column.

- In the next hands-on, you'll implement a function named **findEmptyRow** that is given a column and returns the index of the last empty row or -1 if the column is full.

- First, we'll modify the **play** function to call **findEmptyRow** so that it:
  1. Places pieces in the selected column's last empty row
  2. Does not use a turn if the column is already full

```
let emptyRow = findEmptyRow(col);
if (emptyRow === -1) {
    alert("Column is full!");
    return;
}

// Assign a position to the current player
board[emptyRow][col] = player;
```

Hands-on #1) **findEmptyRow**

- Implement the **findEmptyRow** function. For the given column, it should:

1. Begin a **for loop** at the *last row's index* using a **row** counter variable
2. Continue looping while the **row** is still a valid index of **board**'s rows
    1. Hint: How do you loop through every element in an array from last to first?
3. Subtract 1 from **row** each iteration
4. In the loop, check to see if the position **board[row][col]** is empty (equal to -1). If so, return the row!
5. If no empty position is found in this column, return -1.

- Check-in on PollEv.com/comp110 once you can click around your pieces stack properly.

```
/**
* Given a column, find the first "EMPTY" row starting from the last row of the
* column and working back up. If no empty row for the col is found, return
*/
let findEmptyRow = (col: number): number => {
    for (let row = board.length - 1; row >= 0; row--) {
        if (board[row][col] === EMPTY) {
            return row;
        }
    }
    return -1;
}
```

Follow-along #2) Call isWin to check for a winning move

- In the next hands-on, you'll begin implementing the algorithm that checks for a winning move in the isWin function.

- Let's modify the play function so that it calls isWin and will alert the user and reset the game if a win is encountered.

```
// Assign a position to the current player
board[emptyRow][col] = player;

// TODO: Check for wins
if (isWin(player)) {
    alert("You win!");
    reset();
}
```

Hands-on #2) Implement **isWin**

- The **isWin** function will check every element on the board and call the **isConnect4** function to test whether that position is the start of a four-in-a-row streak

1. Write a nested for loop that will iterate through every row and column in the **board**'s 2D array.
2. For every combination of row and column, call the **isConnect4** function. If **isConnect4** returns true, then **isWin** should also return true.
3. After all positions have been checked, and no win is found, return false.

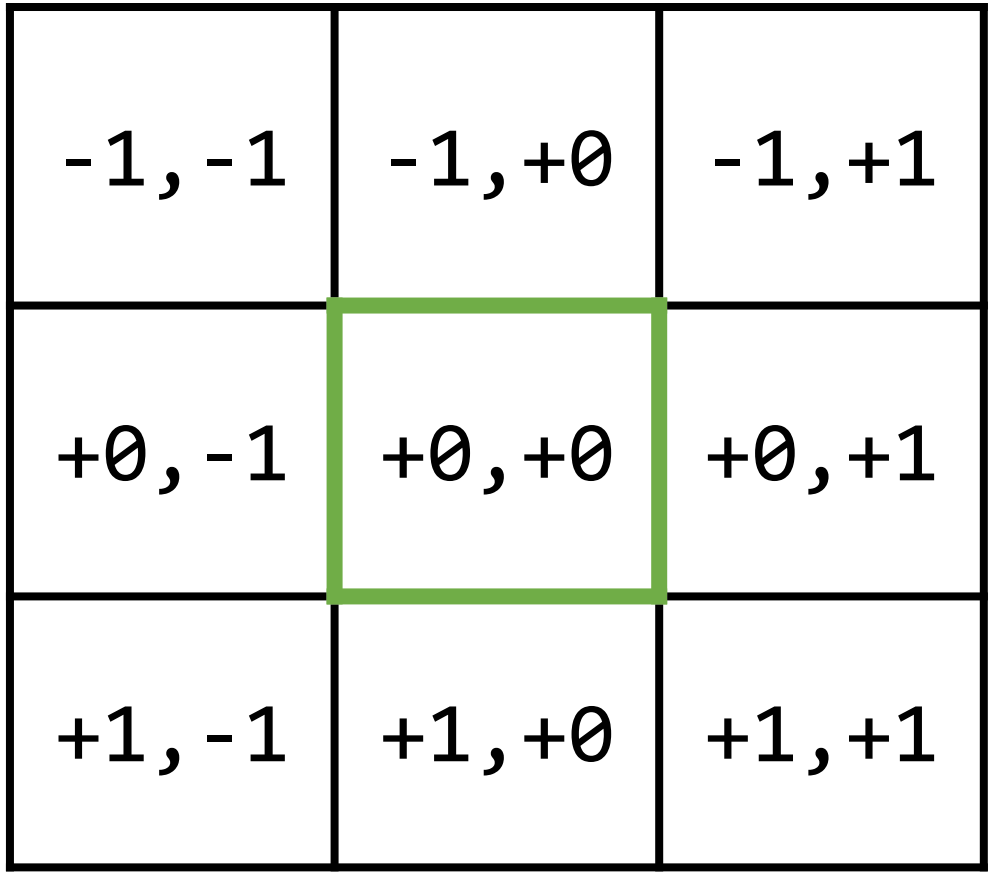
- There's a faulty, simple implementation of **isConnect4** that will allow you to test your **isWin** functionality. Try figuring out which position and player you can use to generate a win.

- Check-in on PollEv.com/comp110 when you have **isWin** working or are stuck

```
/**
* Check to see if a player has won the game. This will traverse every position
* in the 2D array and check to see if that position is the start of a winning
* streak of 4 connected positions controlled by the same player.
*/
let isWin = (player: number): boolean => {
    for (let row = 0; row < board.length; row++) {
        for (let col = 0; col < board[0].length; col++) {
            if (isConnect4(player, row, col)) {
                return true;
            }
        }
    }
    return false;
}
```

Directional Offsets

- We can think about "directions" as offsets of row, col relative to the element we are checking
- We used a similar technique in Conway's Game of Life
- For example, if we are checking for four-in-a-row in a downward, leftward diagonal, the row direction will be +1 and column direction will be -1.



Hands-on #3) Implement **isConnect4**

- You will loop through all 8 possible directions using "rowDir" and "colDir" of -1 and +1 respectively
- Write a nested for-loop with counter variables rowDir and colDir
- Each counter variable should begin at -1 and end at 1 (inclusive)
- Check to see if recurConnect4 returns true. Use 0 as the streak argument and *player* as player argument. If recurConnect4 is true, then return true from isConnect4.
- When the outer loop has completed it means this position does not begin a connect-4 streak. Return false.
- Check-in on PollEv.com/comp110

```
let isConnect4 = (player: number, row: number, col: number): boolean => {
    for (let rowDir = -1; rowDir <= 1; rowDir++) {
        for (let colDir = -1; colDir <= 1; colDir++) {
            if (recurConnect4(player, row, col, rowDir, colDir, 0)) {
                return true;
            }
        }
    }
     return false;
}
```

Hands-on #4) Implementing **recurConnect4**

- Base Cases:
  1. Four-in-a-row streak detected!
     - When streak is 4, return true
  2. Invalid col or row index
     - When col or row are < 0 or >= # of rows or columns, return false
  3. Player does not control this position
     - When the board position at [row][col] is not held by the player, return false
  4. Avoid infinite recursion
     - When rowDir *and* colDir are *both* 0, return false

- Recursive Case:
  - Return the result of calling recurConnect4 recursively on the *next* row and col. Increase streak argument by 1.

- Check-in on PollEv.com/comp110 when you've got recurConnect4 -- your algorithm should now detect four-in-a-row streaks!

Follow-along #3) Checking for Tie Game

- In the last hands-on, you'll implement the logic to see if all positions are taken. Since we're already checking for winners, in this case it means the game is tied.

- Let's modify the play function so that it uses the is isDraw function you'll implement next.

```
if (isWin(player)) {
   alert("You win!");
   reset();
} else if (isDraw()) {
   alert("Draw!");
   reset();
}
```

Hands-on #5) Implementing **isDraw**

- Check every position to see if there are empty spaces available.
- Return true if all positions belong to either player 0 or player 1.
- Return false if any positions are -1.
- Hint: Consider making reuse of the findEmptyRow function you wrote earlier.
- Check-in when your isDraw function is working.

Big Ideas in 110

- Data Types, Variables
- Boolean Logic
- Control Flow (if/while/for) and Functions
- Arrays (and Row-major 2D arrays)
- Classes
  - Properties / Fields
  - Methods
  - Constructors
- Scope (Global, Function Call Frame/Local, Block)
- Recursion
- Generics (Functions, Classes, Interfaces)
- Higher-order Functions and Functional Interfaces (Filter, Map, and Reduce)