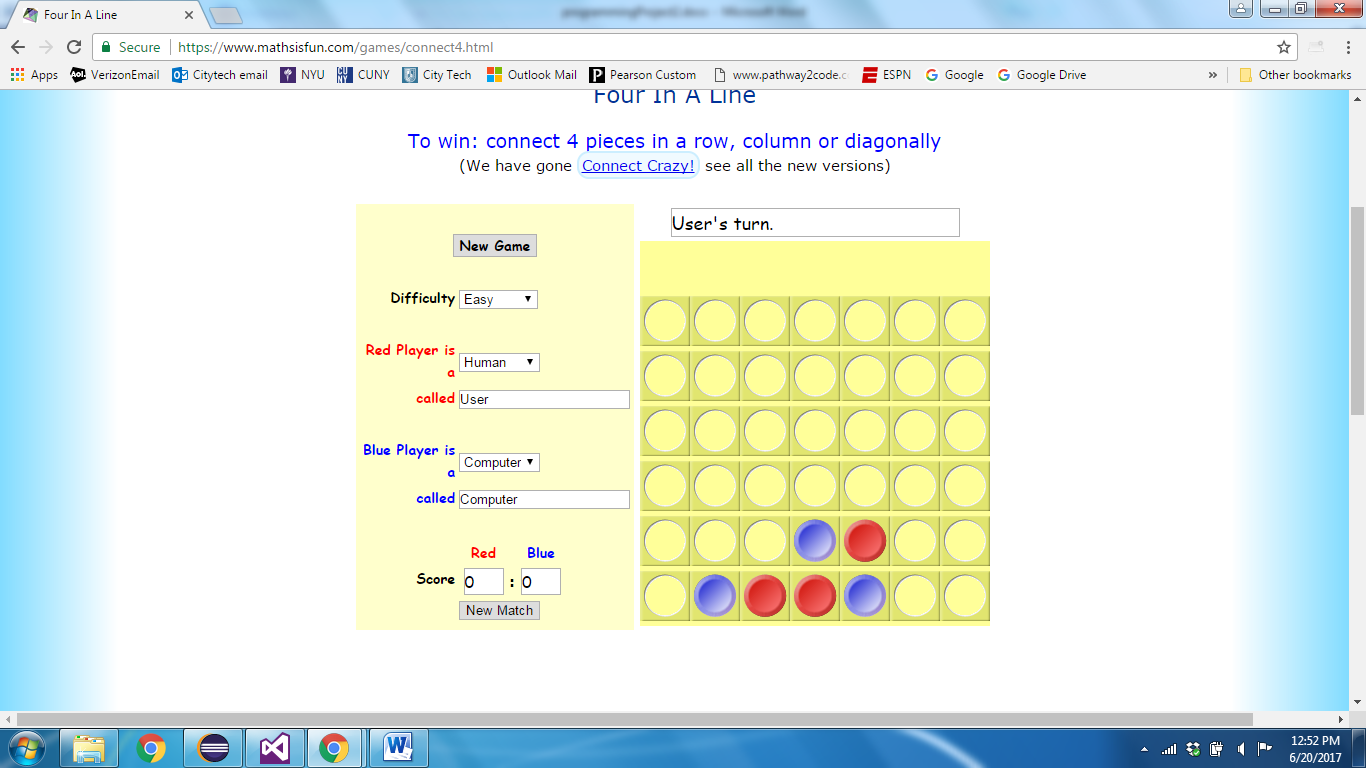
Project 3: Connect4

Due date: Friday July 20 , 11:00 PM EST.

### You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own . Any sharing or copying of assignments will be considered cheating (this includes posting of partial or complete solutions on any public forum). If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

In this project you will help determine the best moves for an abbreviated game of Connect4. The program will analyze the best starting position to make a move. Due to processing times our Connect4 board will be 4 x 4, versus a 7x7 board of the game ( the actual board is 7 x 6).



The program will explore making its first move in each of the four columns. The first action will be to make a move in one of the four columns. The program will then pass the Board, and the next player to a Play method. The Play method will analyze the board and call itself up to 4 times, representing the possible number of next moves. At times the Play method may call itself less than four times due to the condition that a column is full. The Play method will return a 1 if the game is won by the first player, -1 if won by the second player, and zero, if that moves leads to a tie. Hence Play (board, clr) gives you the Net wins for first player, given the board position represented by board, and the next move is to be taken by clr.

A game is won if 4 discs of the same color appear in a column , row or diagonal.

# Objectives

The goal of this programming project is for you to master (or at least get practice on) the following tasks:

* understanding recursion calls
* debugging and checking results
* writing classes
* working with existing code

# Helpful code:

# Code is provided to do the same type of logic as this problem, except with a Tic Tac Toe Board. In the Tic Tac Toe game, after the first move, there are 8 possible moves by the second player. In Connect4, with our abridged board, there are always a maximum of 4 next moves. Also in the Tic Tac Toe game the next move can be anywhere on the Board, while in Connect4, only the bottom of an open column can be chosen for the next move.

## Tic Tac Toe Program Logic

The logic of the program provided prints out information for X making the first move in one of the three spaces of a diagonal. In the Main method, within a loop iterating through the 3 diagonal spaces, the program makes a first move and then calls Play , passing the current board state, and next player.

Within Play, the method first checks to see if the current board has a winning position for either player. If so, 1(X) or -1(O) is returned. Also a non-winning full board is checked. If the board is full, with no winner, then zero is returned. The CheckBoard method does this analysis of the board.

If the board is not a complete game, CheckBoard returns 3. This result prompts Play to recursively call itself with all possible moves for the current player. The current board array is copied to another array. to ensure future executions of the method do not alter the current board.

## Expected Results:

It is advised to make your Connect4 program also work on a 3 by 3 matrix. This is easier to test with, since the 4x4 does take more processing. If you run a 3x3 Connect4 game , these are the results you should achieve:

NetWins for column 0: 112

Number of recursion calls: 1087

Red Wins:224 Blue Wins: 112

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NetWins for column 1: 6

Number of recursion calls: 1103

Red Wins:168 Blue Wins: 162

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NetWins for column 2: 112

Number of recursion calls: 1087

Red Wins:224 Blue Wins: 112

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Running the program

The zipped java project file, which contains all your source code, input files and Eclipse related projects, is to be submitted. If you submit an incorrect file (say , just the source file) you will lose 10% of the project value. If you are unsure you can submit a non-working early version to be sure. You may include more than one test file to indicate your program is working.

## Working on This Assignment

You should start right away!

You should modularize your design so that you can test it regularly. Make sure that at all times you have a working program. You can implement methods that perform one task at a time. This way, if you run out of time, at least parts of your program will be functioning properly.

**Grading Criteria – 20 points**

1. (5) Program shows a recursive approach to determining the next possible positions and making recursive calls.
2. (5) Required outputs, including the number of wins by player, and number of recursion calls.
3. (10) Results are accurate. Show how you can achieve the results for the 3x3 shown in this sheet.

+3 points extra credit –

When considering the next possible moves, if the player can win, have that player only make that move. For example, if all four columns have an opening, but moving in column 3 wins the game, only recursively call moving in column 3.