Final Project: Connect Four

Introduction:

This project builds on the concepts learned thus far to cheat at the classic childhood game of Connect Four. The rules for the game are simple: using a 6 x 7 game board, get 4 of your pieces in a row (in any direction!) before your opponent does.

Your goal is to find the best move to make in the game from only its image! This code will be split into 3 separate parts: Image Processing, Board Analysis, and Finding the Best Move.

Part 1: Process Image

Approximate lines of code: 45 (does not include comments or white space)

Given various images of the game, your code must determine where the pieces are currently located; a perfect image thresholding problem! The images you have to process will always follow these rules:

- Player 1 uses yellow pieces
- Player 2 uses red pieces
- The game board is always a dark blue
- The board will always be fully visible and oriented upwards but may be different sizes

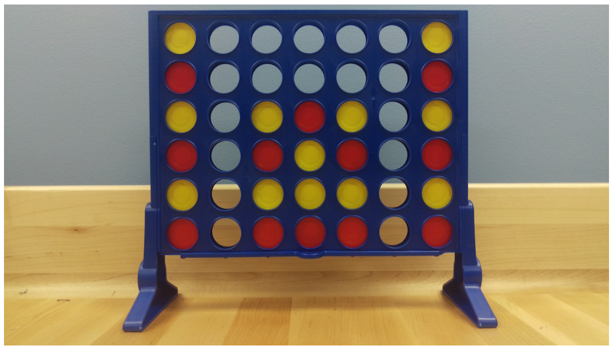

Figure 1: Example image to decode in software.

Create the following function:

- **Function Name:** ProcessImage
- **Input:** Name of Image to Open
- **Output:** Board Layout (as 2D grid of numbers)

The function is complete when it can take any of the 5 sample images provided, and produce the correct board layout for each. Use a **1** to represent Player One's yellow pieces and a **2** for Player Two's red pieces. Use a **0** for empty hole locations on the board. (**Please note that your code will be tested against different images when graded!**) For example, the board layout for image 1 show above is the following:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 1 & 2 & 1 & 0 & 1 \\ 2 & 0 & 2 & 1 & 2 & 0 & 2 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 2 & 0 & 2 & 2 & 2 & 0 & 2 \end{bmatrix}$$

Part 2: Evaluate Board Layout

Approximate lines of code: 45 (does not include comments or white space)

Once the code has identified the board layout, it must evaluate how each player is doing using a scoring system. Essentially, we need to know which player is in the strongest position to win! The scoring system looks at every **combination of 4 pieces** on the board and calculates a score for each player:

4 pieces of the same color	1000 points
3 pieces of same color and 1 open space	10 points
2 pieces of same color and 2 open spaces	1 point

Create the following function:

- **Function Name:** EvaluateBoard
- **Input:** Board Layout (as a 2D grid of numbers)
- **Outputs:** Score for yellow, Score for red (these will actually be references!)

In the following board layout, **Yellow has a score of 12 points** and **Red has 0 points!**

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Part 3: Find Best Move

Approximate lines of code: 45 (does not include comments or white space)

Finally, the code must use all this information to make the best move possible using a **2 Level Minimax** algorithm. This algorithm looks at all the possible moves it can make (Player 1) and then all the possible moves Player 2 (red) can make based on those moves! By looking ahead like this, it can decently predict the best move to make. The more levels the algorithm looks ahead, the better chance it has of making the perfect move but at a significant computation cost. At each level, the algorithm will try to minimize its maximum loss!

Create the following function:

- **Function Name:** FindBestMove
- **Input:** Board Layout (as a 2D grid of numbers)
- **Outputs:** Best Move, Best Score (these will actually be references!)

The code will simulate all the possible game variations for 2 levels by dropping pieces into every column! These game variations will be grouped together based on the starting move to reach this point. Using the EvaluateBoard function from Part 2, calculate the total score value by subtracting the yellow score from the red score for each potential board layout. Lastly, find the minimum score from all the possible groups on Level 2 and then the maximum score of those values. This score represents the best move the algorithm can make!

Extra Credit (10 bonus points on final project): Draw Picture of a Game

Instead of trying to cheat at Connect Four, pretend you are going to make your own Connect Four computer app! Given a board layout, make an image that is **500 by 500** pixels wide that shows the state of the game. There really aren't any rules here, just draw an image that looks good!

Create the following function:

- **Function Name:** DrawGame
- **Input:** Board Layout (as a 2D grid of numbers)
- **Output:** Image of Board