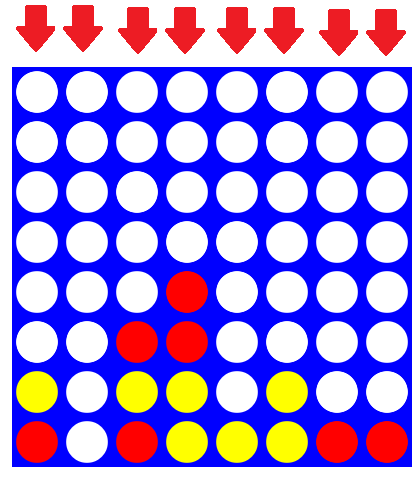
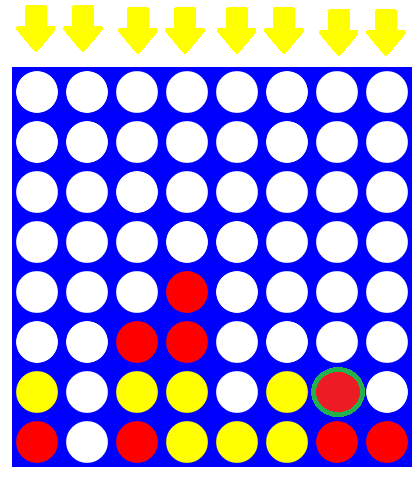
# Connect 4 AI

The [connect 4 playing program](http://www.roadtolarissa.com/javascript/connect-4-AI/) uses a [minmax algorithm](http://en.wikipedia.org/wiki/Minimax).

Every time the computer decides what move to make next, it considers all of its possible moves:



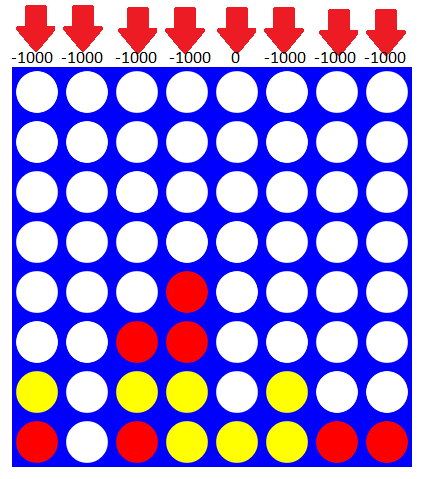
The computer then pretends that each of the moves it has considered has actually taken place. For every possible move, it looks at all the moves the other player could make in response. For example, while considering what would happen if it played a red piece down the 7th column, the computer would examine the following yellow plays:



Each of these yellow moves is analyzed in the same way. 8 initial moves each have 8 responses for a total 64 distinct board positions after two moves. After three moves, there are 64*8 = 512 possibilities. The amount of times this process repeats is determined by the \*search depth* of the algorithm. When the program is called the first time, it is passed a depth parameter. Each time another layer of moves is considered, the depth is decremented by one.

Once the depth is equal to 0, all of the moves on the current layer are evaluated. Winning moves for red are given a score of 1000, while winning scores for yellow are given a score of -1000. Every time a layer of 8 possible moves is considered, if the red player was placing a piece, the move with the highest score is picked, while the yellow player does the opposite. After a move has been picked, it is sent back up the chain and used to evaluate the move before it along with the 7 other evaluations which have also been sent back up. The algorithm is called minmax because the computer is looking to minimize its maximum losses.

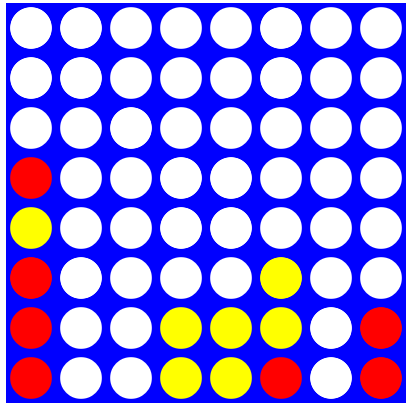
In the above picture, the computer would test all 8 yellow moves and conclude that the best move for the yellow player would be the 5th column since that results in a yellow win and a score of -1000. That score is sent back up the chain of moves with the scores of the other 7 moves, resulting in the following:



From the computer's perspective, it has the red player move to the 5th column to minimize its maximum losses (since 0 is less of a loss than -1000), but to us humans it looks like it is blocking yellow from completing a four in a row.

None of the parts of the algorithms discussed so far are specific to connect 4. Any two player game with alternating moves and terminal wins and losses can be played by considering chains of future moves looking for wins to force and losses to avoid. Unfortunately, a connect 4 program that only looks at wins and losses which takes its turn in a reasonable time frame isn't a very good player. Since examining all the possible move sequences of consisting of n moves requires 8^n board evaluations, looking farther ahead requires exponentially more time. If there was a computer fast enough to look ahead all the way to the end of the game, it would be possible to play a perfect game of connect 4 (in fact, [a computer has done so for the 7×6 version](http://www.connectfour.net/Files/connect4.pdf) of the game, which is part of the reason why I've been working on the 8×8 version).

With reasonably good play by both players, ending conditions can be many more moves father ahead than the computer can examine. Because of this, the computer needs a **heuristic function** to evaluate board states without a winner on them. Without one, during the opening plays the computer moves almost randomly, just avoiding positions that would allow their opponent to force a win within the next few moves. To the computer, red and yellow are essentially tied in this board state:



But it should be obvious to any human player that yellow has established a superior board position over red. This inability to distinguish nonterminal board states is corrected with the addition of above mentioned heuristic function. Currently, to the compare non terminal states, the computer counts the number of open 3 in a rows each player has. This simple rule allows the computer to understand the previously inscrutable board state like this instead:



The computer now knows that in this board position yellow is ahead; if the computer was playing red it would avoid move sequences that resulted in this position while if the computer was playing yellow it would try to set up as many of these three in a rows as possible. This heuristic seems to work fairly well – with the current set up, allowing the computer to make move searches with a depth of 4 produces an AI that beats me about half the time, while against a depth 5 opponent I lose most of the time.

There isn't anything special about the heuristic I've chosen. Any function that takes a board state as a input and returns a number will work as a heuristic There are three things I considered when picking this one:

1. Computational cost. Since every move requires running the heuristic thousands of times if it is too complicated, the depth of searches will be adversely affected.
2. Correspondence to wins. If a heuristic doesn't provide useful about the game, there isn't any reason to use one.
3. Ease of Implementation. More complicated heuristics take more time to program so they run quickly and without errors.

Since I mostly wanted to get a version of the AI up and running, my decision was heavily weighted towards the third option. In the code, the heuristics algorithm only has to check all the empty spaces to see if they are in line with a 3 in a row. This was simple to implement because I had already written to code to check for 4 in a rows. In the future, experimenting with other heuristics will probably improve the AI. Counting 3 in a rows still struggles at differentiating between early game positions and values unusable 3 in rows late game:

The computer thinks that the red player has an fairly favorable position since it has two more 3 in a rows than the yellow player even though the game must end before they can be completed. Having already created a baseline heuristic, once I learn more about connect 4 and can come up with better heuristics, I can have the AI play thousands of slightly randomized game against another AI using a different AI evaluate which heuristic is best.

In my next post, I'll analyze the impact of changing the heuristic. I'll also discuss the implications of using a variation on the minmax algorithm, [alphabeta pruning](http://en.wikipedia.org/wiki/Alpha-beta_pruning) with history based move ordering.