In the connect-4 game, states are all possible board and the number of actions per state varies from 0 to 7 which depends on how many columns are still available to fill.

What heuristic? Why

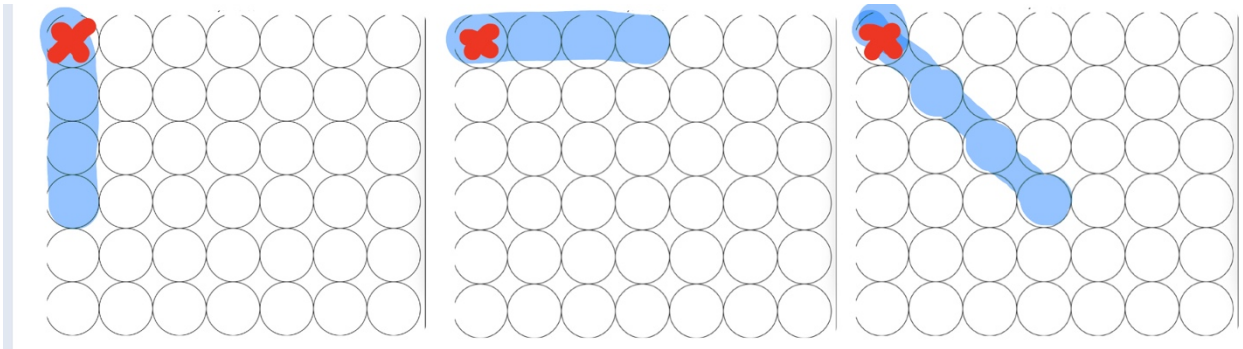
For both the implementations of alpha-beta search algorithm and expectimax search algorithm, I cut off the search at depth of 4 in order to save the time spent for each round, in other words, I use depth limit of 4. Consider the use of depth limit, probably the utilities of leaves are not accessible before we cut off the tree, so I use a heuristic evaluation function to compute the estimated desirability of a state, here in other words, the estimated value of win at the state. The heuristic evaluation function is described below:

1. Initially, for each position inside the board, I compute the number of four connected positions which include that position by hand, and I got the the table:

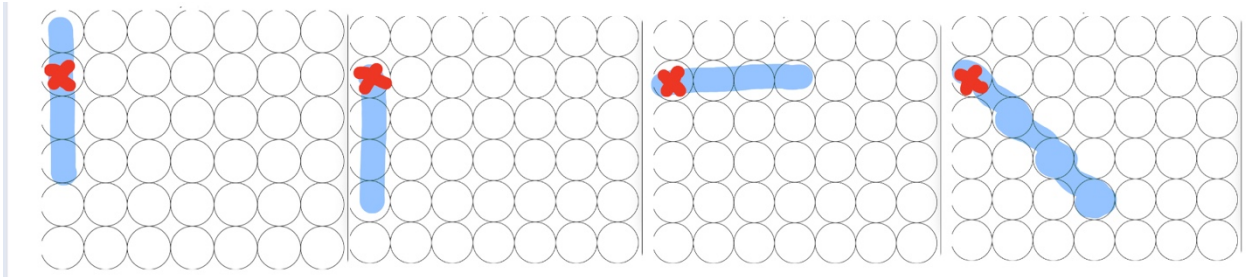
   [[3, 4, 5, 7, 5, 4, 3],
   [4, 6, 8, 10, 8, 6, 4],
   [5, 8, 11, 13, 11, 8, 5],
   [5, 8, 11, 13, 11, 8, 5],
   [4, 6, 8, 10, 8, 6, 4],
   [3, 4, 5, 7, 5, 4, 3]]

Here are two examples about how to understand the table:

For the left top position, it has a value of 3 because it has the three possibilities of connecting 4 positions as followings:



For the first column second row position, it has a value of 4 because it has the four possibilities of connecting 4 positions as followings:



In the same way, I got the whole tables.
Sum up the table, the value is 138.

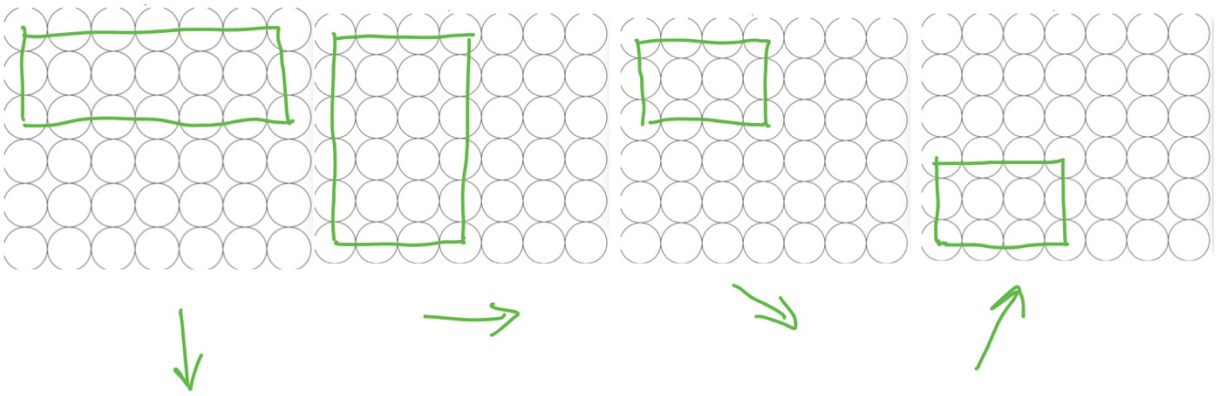
I would like to compute the evaluation of each state so that it could estimate how close the current state is to win. Obviously, when next state has more possibilities of connecting 4 positions, then the estimation of future is more optimistic.
So initial evaluation function will sum up the total number of possible four connected positions for all the current player's positions and minus the up the total number of possible four connected positions for all the opponent's positions.

Eval = (value[position] for each current player's position of current board)
- (value[position] for each opponent's position of current board)

Here value[position] is the value at position of table given above.
However, the table values need some other extensions, as followings.

2. Block opponent and Form 4 connected
   However, before compute the value as above, we need to check if the current board state has any connected 4 position whatever for opponent or the current player. So inside evaluation, firstly, I check the current board, if any connected4 exist, whatever for opponent or current player, evaluation function return what the check function return, the check function is as followings:
   a. If opponent has 4 connected position, check function return -1000
   b. If current player has 4 connected position, return 1000
   c. Otherwise, just return 0, which means no connected4 found

   The above check function is also used at alpha_beta_search in order to cut the depth if a connected4 is found at earlier depth.
   How do I implement the check function to find the connected4?
   Below is a simple sketch of four possible solutions, 4 connected positions could form lines to the right, down, and diagonal up and to right, down and to the right. So inside the picture, the positions inside the green rectangle are those possible positions which could form lines of 4 directions separately. Implementations could just iterate through those positions.



   For expectimax search algorithm, the probabilities I use is just a simple uniform distribution. Probabilities = 1/len(successors), in other words, 1/7

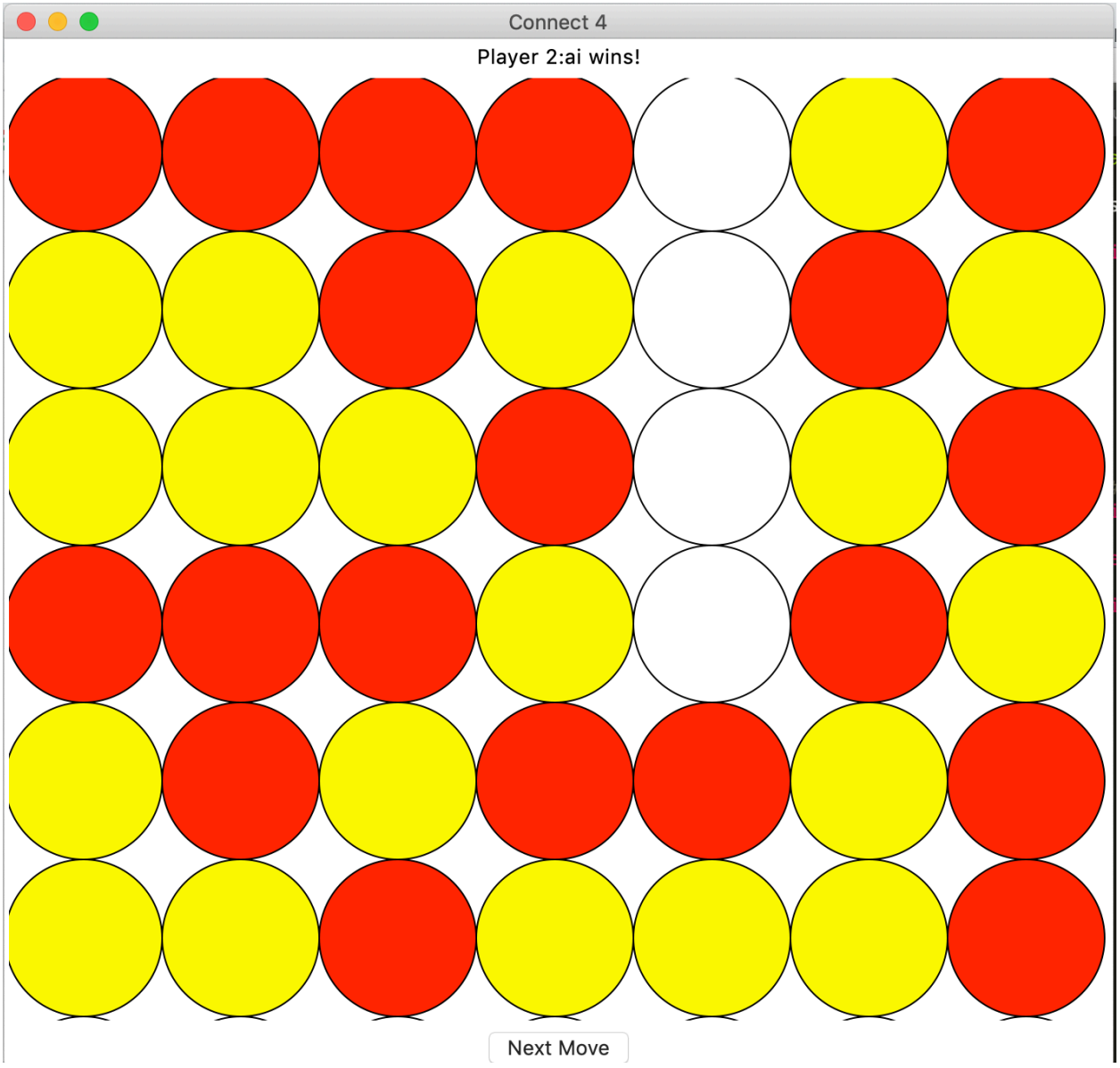
Describe how your algorithm performs given different time constraints.
How much of the tree can you explore given 5 seconds per turn? 10 seconds? 3 seconds?

For my alpha-beta-search, I list the table below to show their performance with different max_depth for 3, 5, 10 seconds. Exceed means that the player Exceeded time limit, Not exceed means that not exceed time limit and the whole tree of that max_depth could be explored.

sec\depth	5	4	3
3	Exceed	Exceed	Not exceed
5	Exceed	Not exceed	Not exceed
10	Exceed	Not exceed	Not exceed

Can you beat your algorithm?
No, I tried several times but fail, but maybe more tries will work.

If your algorithm plays itself, does the player that goes first do better or worse in general? Share some of the results.
The player that goes first always do worse, and the result board is always the same.


Connect 4
Player 2:ai wins!
Next Move