Problem Set 3

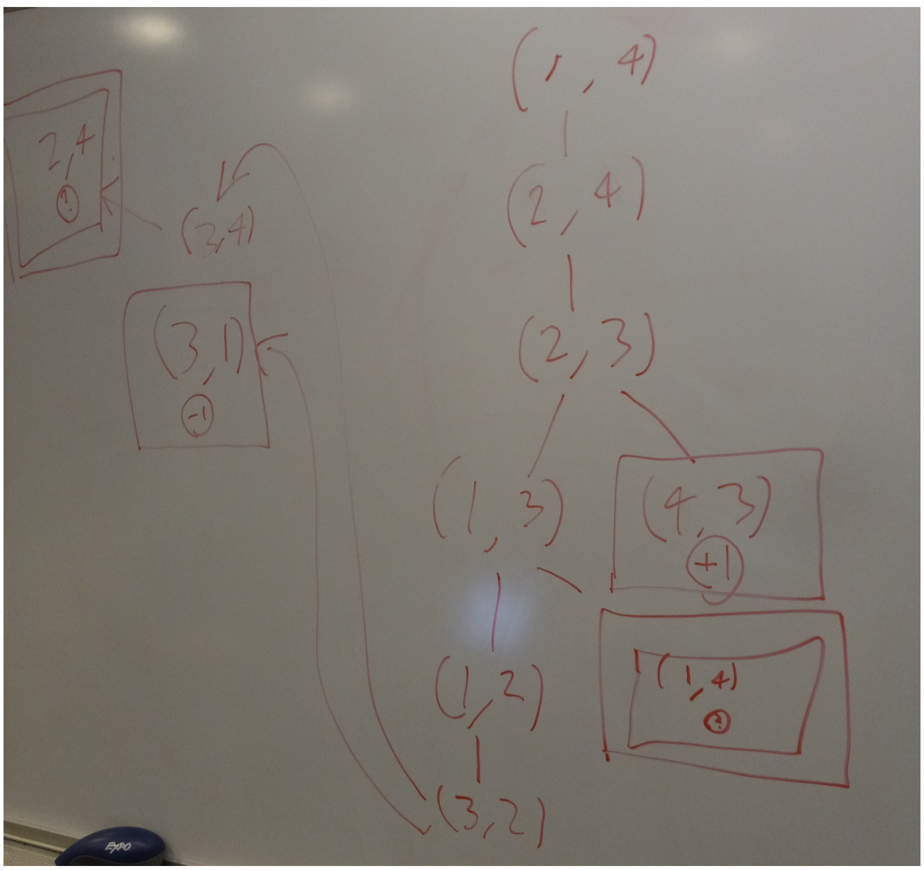
By Ben Nguyen

5.8 (7+3+7+6 = 23 pts)

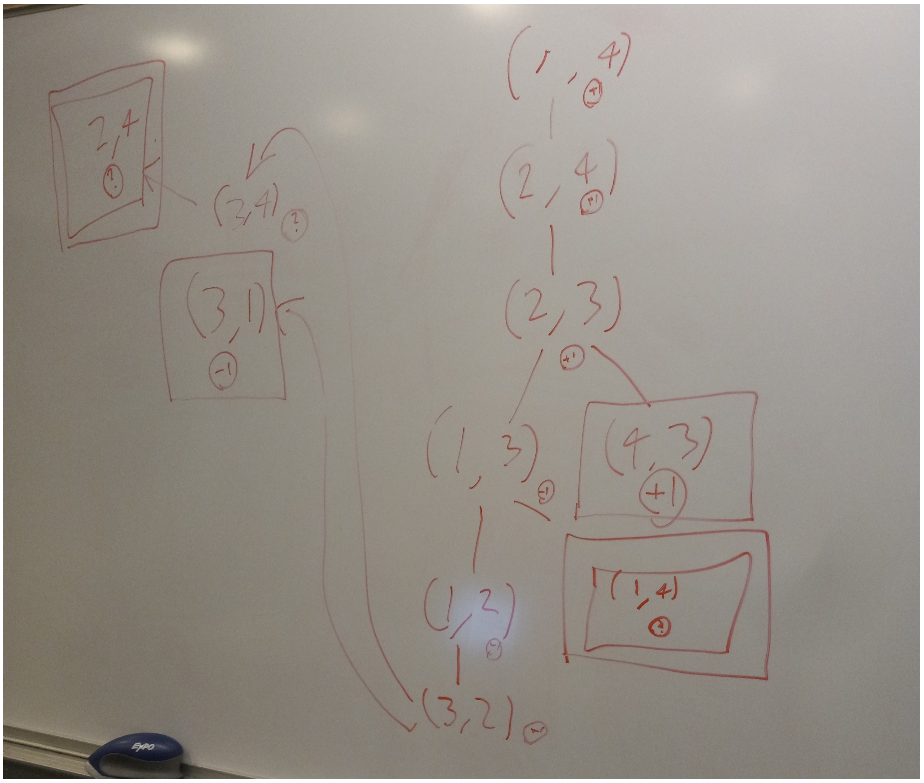
Consider the two-player game described in Figure

1. Draw the complete game tree, using the following conventions:

- Write each state as (sA,sB), where sA and sB denote the token locations.
- Put each terminal state in a square box and write its game value in a circle.
- Put loop states (states that already appear on the path to the root) in double square boxes. Since their value is unclear, annotate each with a "?" in a circle.



2. Now mark each node with its backed-up minimax value (also in a circle). Explain how you handled the "?" values and why.



The “?” values where discarded when they would be overwritten anyway no matter what they would have given back. For example, when the minimax algorithm wants to pick a node that has the lowest value, and one of the two nodes it is considering is a -1, then there is no way for the other node to influence the outcome, so it does not matter if it is a “?”. All of the “?” values has this happen to them and got overwritten.

3. Explain why the standard minimax algorithm would fail on this game tree and briefly sketch how you might fix it, drawing on your answer to (b). Does your modified algorithm give optimal decisions for all games with loops?

This game does not take into account who’s turn it is in the game state, so there could be a 1,3 or 1,3 and the game tree would think that they are the same thing, but in reality in each one it is a different person’s move

Also while minimax will loop the states back on itself, while the alpha beta exploration will recreate the entire tree at that node, instead of looping back on itself

4. This 4-square game can be generalized to n squares for any $n > 2$. Prove that A wins if n is even and loses if n is odd.

Assuming that both players play optimally, the optimal move is always to progress toward the goal, which is always forward and never backward.

Therefore, if n is even, if both players always go forward, then it will take $n-1$ steps for player A to get to the end, and n for player B to get to the end because player A will always jump over B.

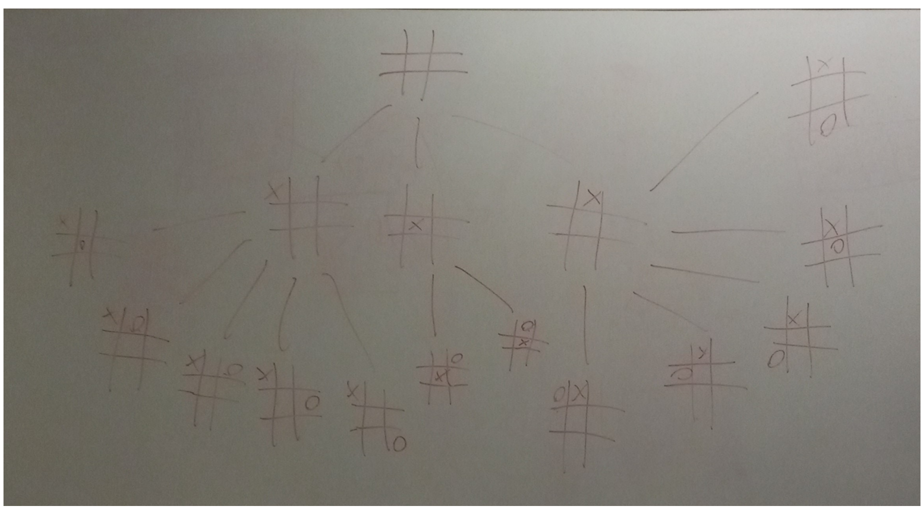
Therefore, if n is odd, if both players always go forward, then it will take n steps for player A to get to the end, and $n-1$ for player B to get to the end because player B will always jump over A.

5.9 (3+4+3+4+4 = 18 pts)

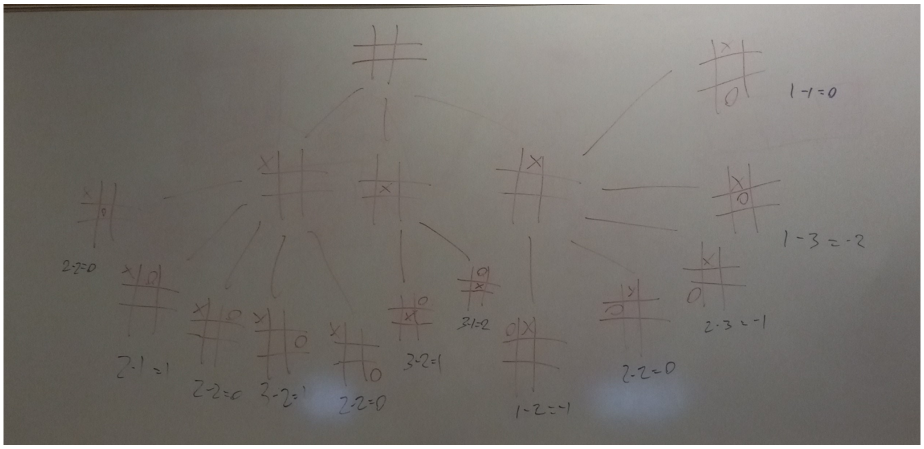
1. Approximately how many possible games of tic-tac-toe are there?

9! because each person can pick $K = 9$ places at the start, then $K - 1$ for each following move

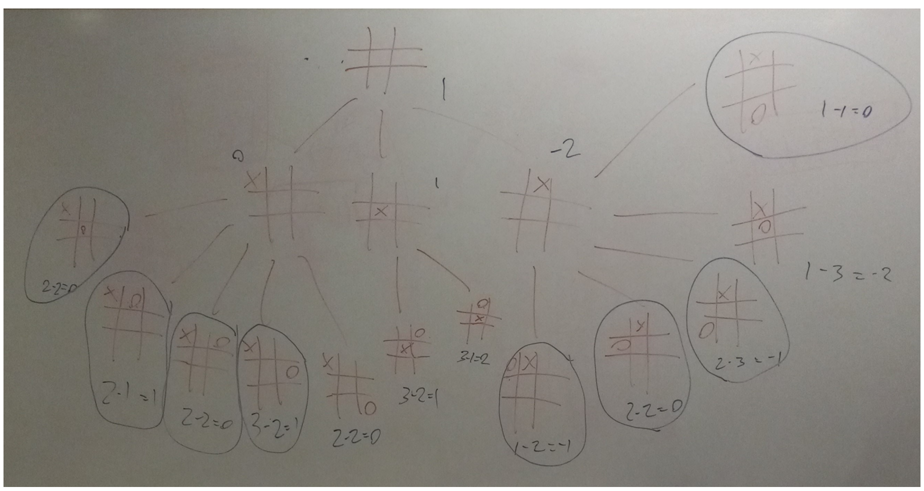
2. Show the whole game tree starting from an empty board down to depth 2 (i.e., one X and one O on the board), taking symmetry into account.



3. Mark on your tree the evaluations of all the positions at depth 2.

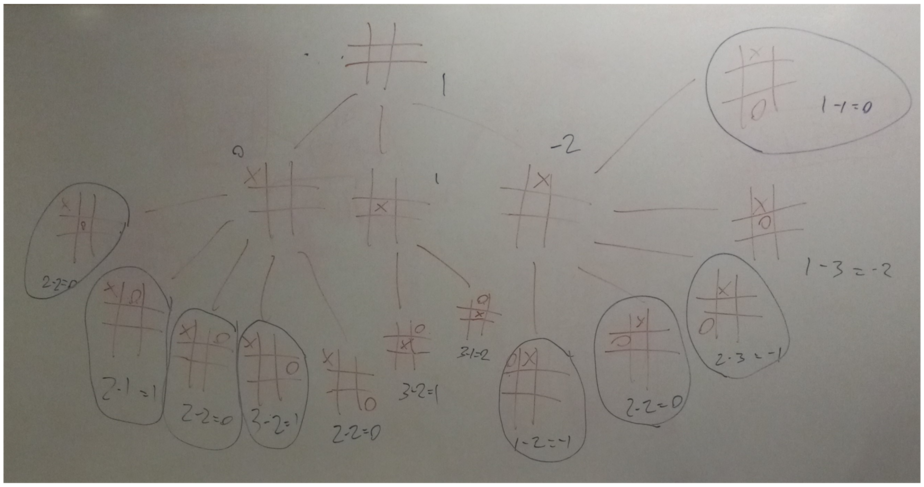


4. Using the minimax algorithm, mark on your tree the backed-up values for the positions at depths 1 and 0, and use those values to choose the best starting move.



Best starting move would be in the middle

5. Circle the nodes at depth 2 that would not be evaluated if alpha–beta pruning were applied, assuming the nodes are generated in the optimal order for alpha–beta pruning.



5.14 (4+4+5+4 = 17 pts)

Develop a formal proof of correctness for alpha–beta pruning. To do this, consider the situation shown in the Figure above. The question is whether to prune node n_j, which is a max-node and a descendant of node n_1. The basic idea is to prune it if and only if the minimax value of n_1 can be shown to be independent of the value of n_j.

1. Mode n_1 takes on the minimum value among its children: $n_1 = min(n_2, n_{21}, \ldots, n_{2b_2})$. Find a similar expression for n_2 and hence an expression for n_1 in terms of n_j.

$n_1 = min(n_2, n_{21}, \ldots, n_{2b_2})$

$n_1 = min(max(n_3, n_{31}, \ldots, n_{3b_3}), max(n_3, n_{31}, \ldots, n_{3b_3}), \ldots, max(n_3, n_{31}, \ldots, n_{3b_3}))$

$n_1 = min(max(min(max(...max(...n_j))), min(...), min(...), ...), max(...), \ldots, max(...))$

2. Let l_i be the minimum (or maximum) value of the nodes to the left of node n_i at depth i, whose minimax value is already known. Similarly, let r_i be the minimum (or maximum) value of the unexplored nodes to the right of n_i at depth i. Rewrite your expression for n_1 in terms of the l_i and r_i values.

$n_1 = min(n_2, n_{21}, \ldots, n_{2b_2})$

Now we only have to evaluate a single thing every time we go up a level, which is just check either if the $n_i > l_i$ if it's a max node, or if $n_i < l_i$

$n_1 = min(l_2, n_2, r_2)$

$n_2 = max(l_2, n_3, r_3)$

Therefore, n_1 can be expanded out until the entire tree is expanded to this:

$n_1 = min(l_2, max(l_3, min(l_4, ..., r_4), r_3), r_2)$

3. Now reformulate the expression to show that in order to affect n_1, n_j must not exceed a certain bound derived from the l_i values.

$n_1 = min(l_2, max(l_3, min(l_4, ..., r_4), r_3), r_2)$

To affect n_1, it must be more than ever single r odd value, and less than ever single l even value

n_j must be $< min(l_2, l_4, l_6, ..., l_{j-1})$ because the even values are the ones that get used in the min function, so n_j must be lower than all of them to get through all of the min expressions and affect n_1

n_j must also be $> max(l_3, l_5, l_7, ..., l_j)$ because the even values are the ones that get used in the min function, so n_j must be higher than all of them to get through all of the max expressions and affect n_1

Therefore the final expression for n_j to affect n_1 is

$max(l_3, l_5, l_7, ..., l_j) < n_j < min(l_2, l_4, l_6, ..., l_{j-1})$

4. Repeat the process for the case where n_j is a min-node.

$n_1 = min(l_2, max(l_3, min(l_4, ..., r_4), r_3), r_2)$

To affect n_1, it must be more than ever single r odd value, and less than ever single l even value

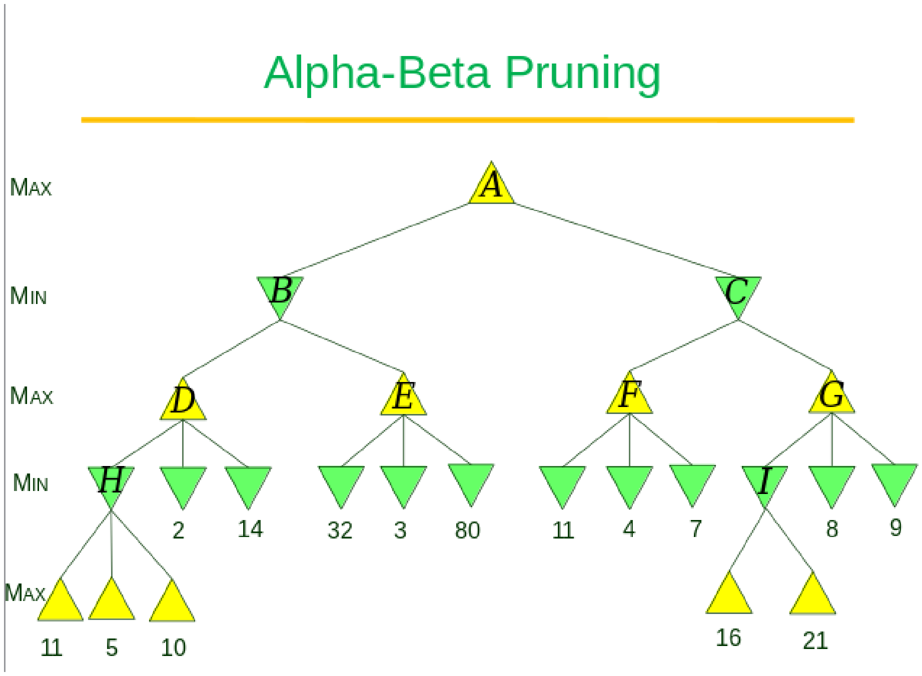
n_j must be $< min(l_2, l_4, l_6, ..., l_j)$ because the even values are the ones that get used in the min function, so n_j must be lower than all of them to get through all of the min expressions and affect n_1

n_j must also be $> max(r_3, r_5, r_7, ..., r_{j-1})$ because the even values are the ones that get used in the min function, so n_j must be higher than all of them to get through all of the max expressions and affect n_1

Therefore the final expression for n_j to affect n_1 is

$max(l_3, l_5, l_7, ..., l_{j-1}) < n_j < min(l_2, l_4, l_6, ..., l_j)$

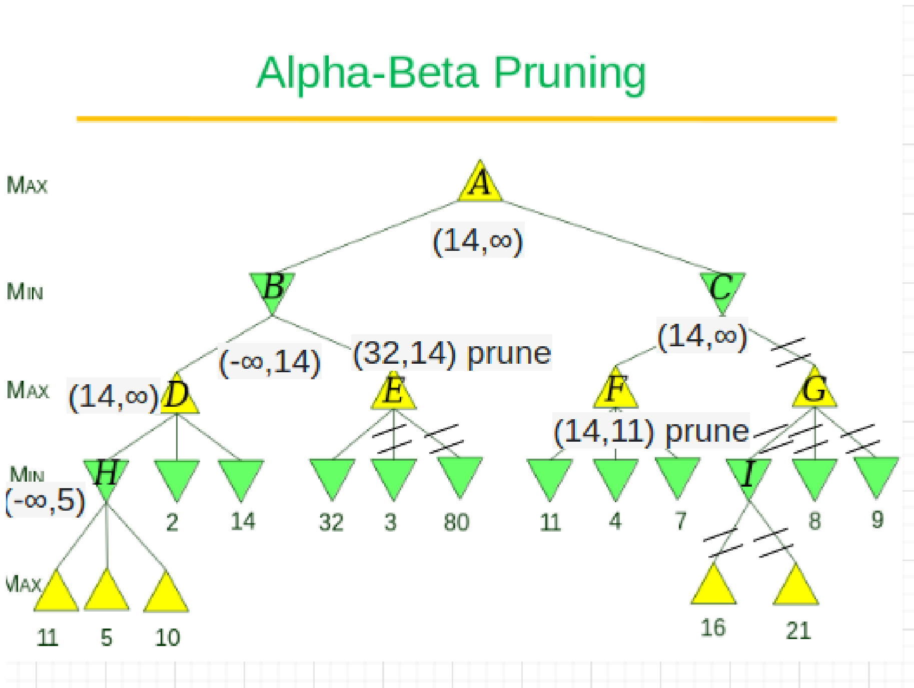
Extra problem (17 pts)



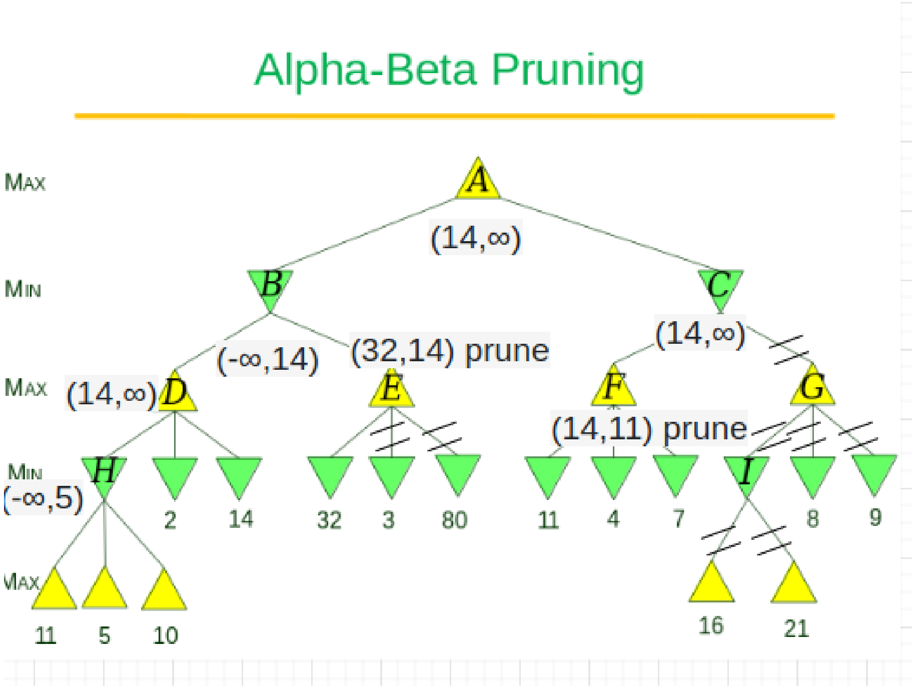
You are given a minimax search tree as shown on the next page. The tree has nine internal nodes . Not all terminal states (leaves) are at the same depth.

Execute the alpha-beta pruning algorithm (use the version from the 3rd edition of the textbook in the lecture notes on February 15).

a. (6 pts) Mark all the subtrees (including leaves) that have been pruned. You may, for instance, simply put double slashes \ or // across the edge entering the root of such a subtree from the above.



b. (8 pts) Next to each visited internal node, write down the two values just before the return from the call MAX-VALUE or MIN-VALUE invoked on the state represented by the node.



c. (3 pts) What is the final value for MAX at the root?

14 is the MAX value at the root