CMSI 2130 – Classwork 3

Instructions:
This worksheet gives you some important practice with the basics of heuristic design, minimax search, and $\alpha - \beta$ pruning! Specific notes:

- Provide answers to each of the following questions and write your responses in the blanks. If you are expected to show your work in arriving at a particular solution, space will be provided for you.
- Place the names of your group members below:

Group Members:

1. Mike Hennessy
2. Cameron Scolari
3. _____

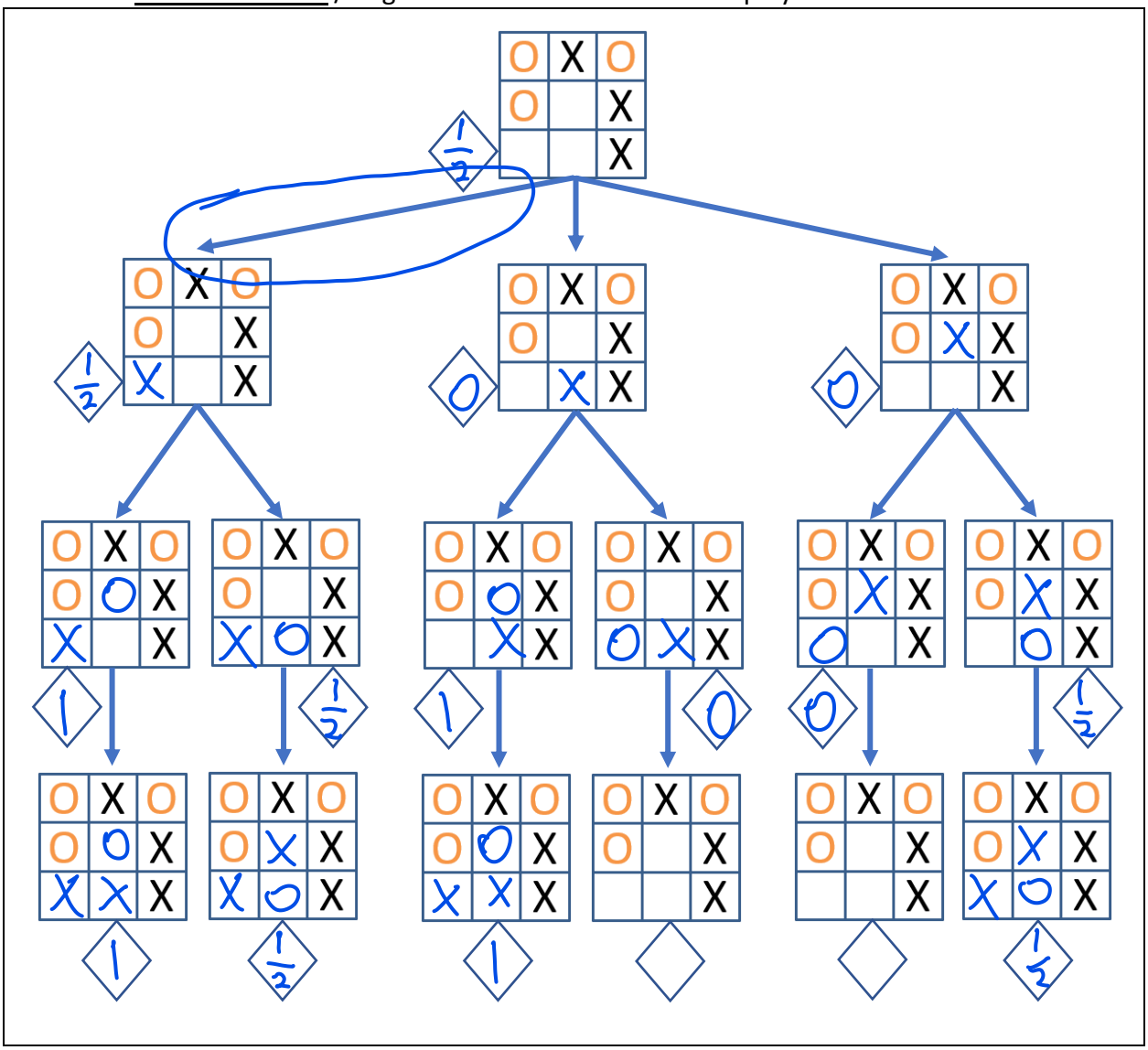
[!] NOTE: If completing digitally, since several exercises have you draw on an existing graphic, I suggest using editing tools on the PDF version (e.g., Adobe Acrobat or Mac's Preview). It is also acceptable to print this skeleton, write your solutions on the paper, and then scan your final copy *so long as* the resulting document is a single, multi-page PDF document that is legible (i.e., DO NOT submit like 10 blurry .jpgs you took with your 1998 Nokia).

Problem 1 – Adversarial Search & Minimax Algorithm

Time to get some practice with a rite of passage… adversarial search on Tic-Tac-Toe (or T^3 for the cool kids)! Suppose we consider a general conception of a 3x3 T^3 board, in which the objective of each player is to connect 3 $X's$ (Player 1) or 3 $O's$ (Player 2) in either straight or diagonal lines (you know the rules of T^3, why am I telling you?).

1.1. Consider the following partial game tree from an advanced game of T^3. You are to:

- **a.** Fill in the missing actions of each state in the game tree (some game board states may not be used, they're just there to make life easy for you; cross out any unused states).
- **b.** Score the terminal states from the perspective of Player 1 (max node at the root) with a score of 0 for losing, $1/2$ for a tie, and 1 for a victory (place scores in diamonds)
- **c.** Using mini-max search, "bubble" these scores up to find the minimax scores of all non-terminal nodes, (place minimax scores in non-terminal diamonds).
- **d.** Circle the action / edge from the root that the max player should choose.

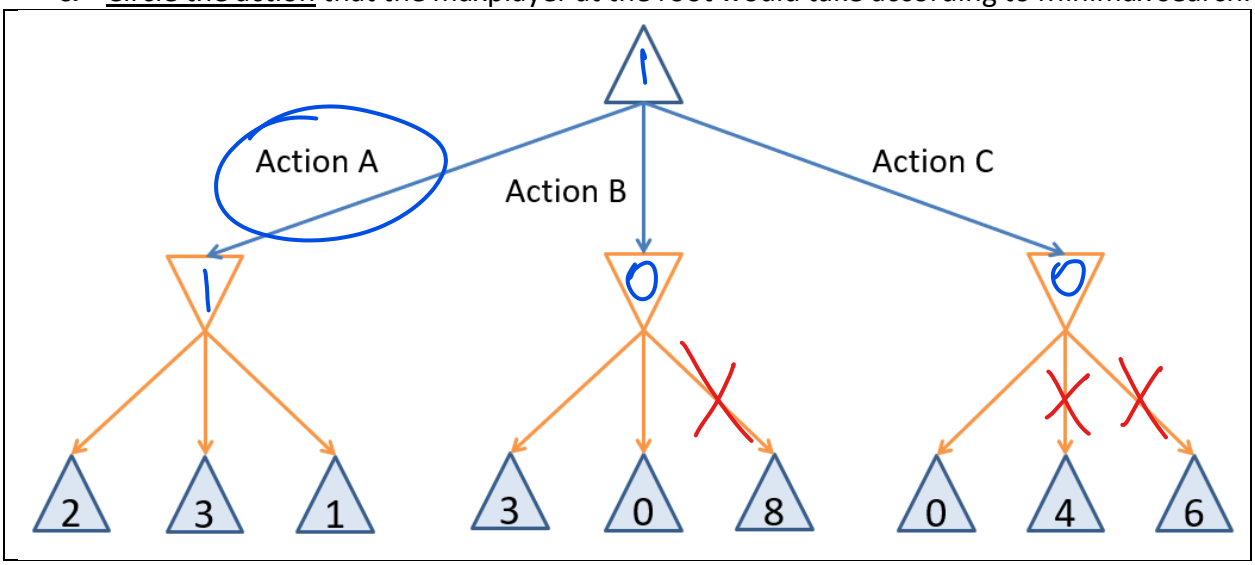


Problem 2 – $\alpha - \beta$ Pruning

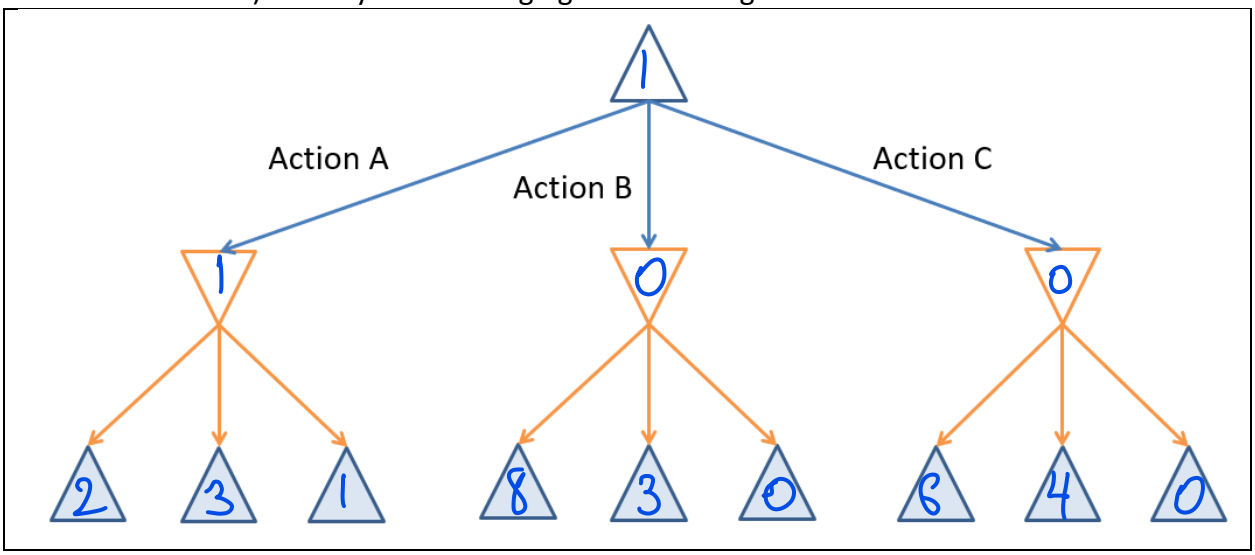
Suppose we have the following, arbitrarily-scored game tree corresponding to some abstract canonical game and that actions are explored in left-to-right order.

2.1. For the game tree specified below, utility and minimax scores are specified within each node, and upward-facing triangles represent max nodes. Your tasks:

- **a.** Indicate the Minimax score of each non-terminal node *that is not pruned*.
- **b.** Indicate which paths are pruned by $\alpha - \beta$ pruning by crossing out the pruned arrows.
- **c.** Circle the action that the maxplayer at the root would take according to Minimax search.



2.2. Using the game tree above, rearrange the order of the terminal nodes' generation such that *no subtree is pruned by $\alpha - \beta$ pruning.* E.g., instead of the middle sub-tree's terminal nodes generating in order $[3,0,8]$, consider a different ordering $[0,3,8]$ (incorrect here, but illustrative for what the task is). Show your rearranging in the blank game tree below.



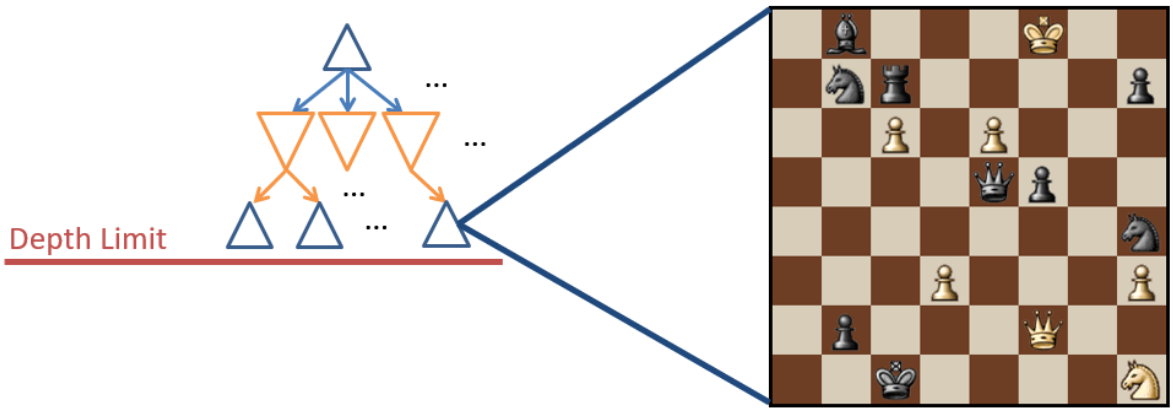
Problem 3 – Conceptual Miscellany

Here are a few questions to get the ole noggin joggin, and make sure you've got some strong roots in the theory behind adversarial search.

Consider that the full game tree for a standard game of chess has $\sim 10^{43}$ states (ignoring repeated states and those that cannot be reached through legal moves). Plainly, this is too large to consider the entire game tree, so we're instead forced to make an educated guess of terminal utilities through the following assumptions:

- We set a depth limit (like in DLFS) at which we pretend all nodes at this level are terminal.
- For each node at the depth limit, we apply an *imperfect utility heuristic* that is meant to estimate the *true* utility of the subtree rooted at that node. Note: "heuristic" here is another application of the "educated guess" paradigm for $h(n)$ we saw in A*, but not subject to the requirements we talked about in classical search like admissibility.

This strategy is depicted below:



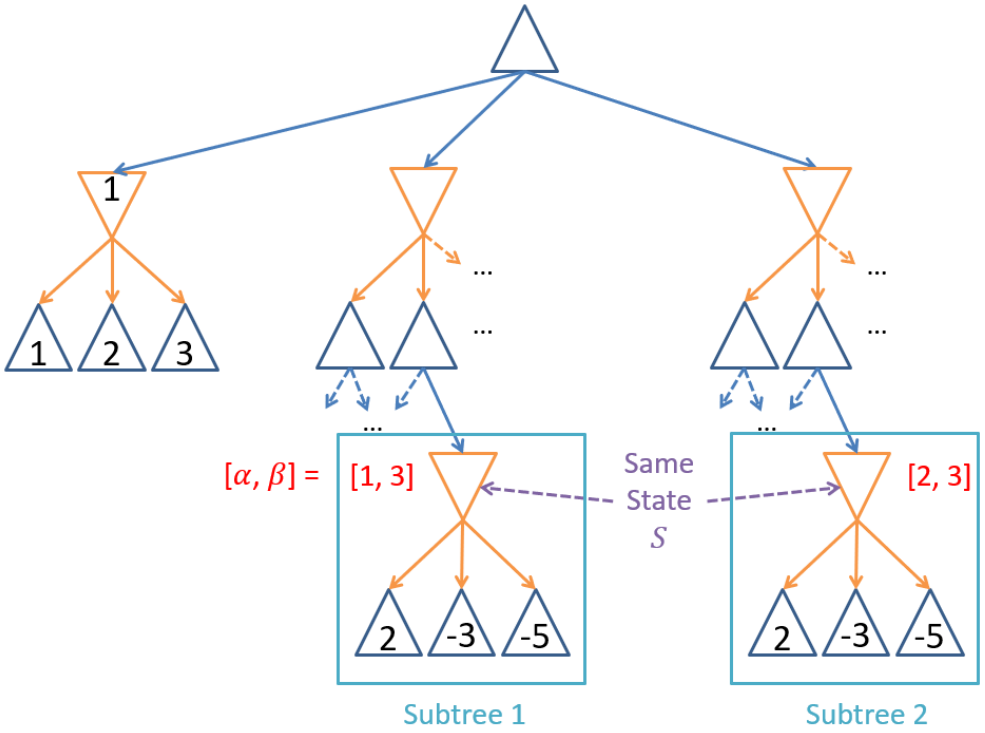
3.1. Suppose we begin with a simple imperfect utility heuristic that simply counts the difference in the number of pieces between our agent and its opponent. If our agent is playing as black what score would this approach estimate for the (non terminal) state shown above on the right?

9 - 7 = 2

3.2. A heuristic that just counts the difference in the number of pieces is pretty simplistic and won't take into account piece quality; e.g., a pawn (the worst piece) and a queen (the best) will be counted equally. Propose (in plain English) another imperfect utility heuristic that would be better than the simple one in 3.1.

It gives each piece a certain value, specifically based on the chess piece point system:
Pawn: 1, Knight: 3, Bishop: 3, Rook: 5, Queen: 9
We would find the difference in our total points and our opponent's total points.

Suppose we are tracking the execution of some arbitrary minimax tree with the arbitrary utilities and minimax values displayed within each node below. Suppose also that the depth-first search is conducted in left-to-right order in the partial game tree displayed below.



Now, suppose that during minimax search with $\alpha - \beta$ pruning, we encounter some State S(e.g., think about some configuration of a chess state where a piece is moved back and forth to replicate an earlier state) at multiple locations in the game tree.

3.3. If we were *NOT* using $\alpha - \beta$ pruning, what would be the returned minimax score of S?

-5

3.4. If we *WERE* to use $\alpha - \beta$ pruning, what would be the returned minimax score of S in each of the different subtrees in which it appears? NOTE: State S has different starting values for $[\alpha, \beta]$ based on the unshown subtrees above them.

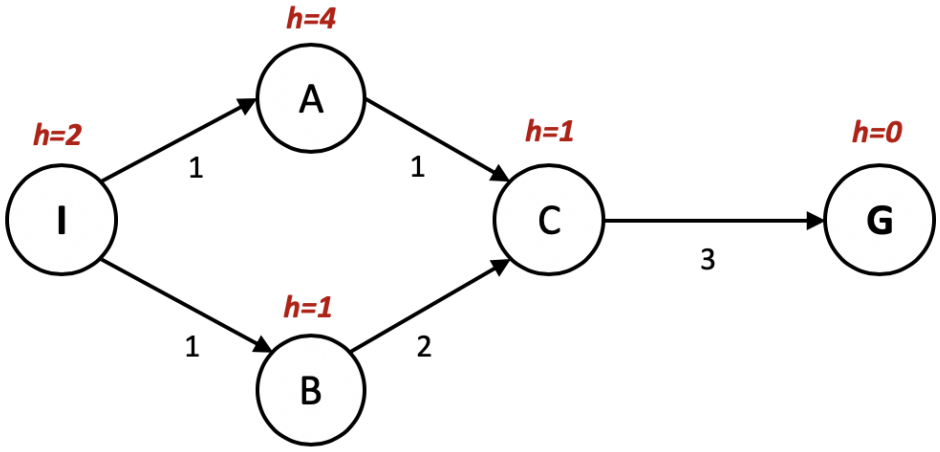
Subtree 1: -3	Subtree 2: 2

3.5. Using your answers above, if we were using $\alpha - \beta$ pruning, would it be wise to cache / store / memoize the minimax score for state S the first time it was encountered to save on computation whenever we see it later during the search (like the graveyard in search)? Why or why not?

Do not store it because the two minimax scores for state S have different paths, and therefore will not have identical alpha-beta ranges.

Problem 4 – A* Heuristic Design

Suppose we have the following State Space Graph of an arbitrary search problem with Initial State *I* and Goal State *G*. Path costs are indicated along the edges of each transition, and the heuristic estimates of some heuristic *h* are indicated alongside each state / node (in red).



4.1. Complete the following table to prove whether or not the heuristic, h, in the state space graph is *admissible (the first row is done for you):*

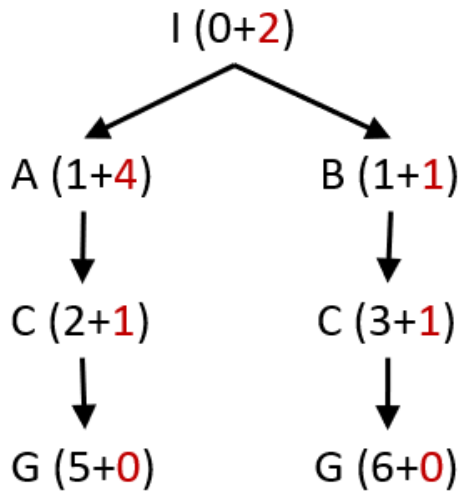
State / Node, n	$h(n)$	$h^*(n)$
I	2	5
A	4	4
B	1	5
C	1	3
G	0	0

False _____ h is Inadmissible True _____ h is Admissible

4.2. Complete the following table to prove whether or not the heuristic, h, in the state space graph is *consistent / monotonic (the first row is done for you):*

State / Node, n	Next State, n'	$h(n)$	$h(n')$	$cost(n, a, n')$
I	A	2	4	1
I	B	2	1	1
A	C	4	1	1
B	C	1	1	2
C	G	1	0	3

True _____ h is Inconsistent False _____ h is Consistent



The tree to the left represents the A* Search Tree that would be generated during A* *tree search* (i.e., without memoization for repeated states), with the *evaluation* $f(n) = g(n) + h(n)$ of each node n listed next to each ($g(n)$ is the black number and $h(n)$ the red).

Notably, the optimal solution *will* be discovered in this example, even though the state C will be expanded twice during the search.

4.3. Provide the order in which A* **Graph** Search would expand its search tree, remembering that graph search *memoizes states* that it has previously *expanded* such that it will never expand nor generate them in the future.

Order of Expansion	State / Node, n	$g(n)$	$h(n)$	$f(n)$
0	I	0	2	2
1	B	1	1	2
2	C	3	1	4
3	A	1	4	5
4	G	6	0	6

4.4. Is the solution found in 4.4 the optimal one? Why or why not? Discuss the notion of an *inconsistent heuristic* in your answer.

It is not optimal because the goal state is reached with a higher cost. The goal state could have been reach with a cost of 5, but we got 6. It is inconsistent because on the transition from A to C, the triangle inequality does not hold, that is h(n) of A, which is 4, is not less than or equal to the h(n) of C, which is 1, plus the transition cost from A to C, which is 1 (4 !<= 1 + 1).