CAS CS 440
Lec 7 & 8

Adversarial Search I

1. The Core Idea
    a. Let's say you are playing a game
        i. Used to be playing a 1 player game
            1. (Even if >1 player, agent's "search" didn't consider other player(s) actions)
            2. When it was our turn to search, the world paused and just considered the soldiers as just obstacles that are static
    b. What if we know the other player(s) move(s)?
        i. Can we model their goals?
        ii. Can we predict what actions they will take?
        iii. If so, can we make better choices?
2. Updating Some Terms
    a. Now there are multiple players (now consider other perspective as well)
    b. Transition function has to consider the other player's action as well
    c. Utility function $u(s) \rightarrow u(s,p) \rightarrow$ only works on the terminal state
        i. It just took the state previously since it was always from our perspective
        ii. But now there are multiple players so we need to consider who's perspective are we taking right now since there are multiple players in the game
        iii. It figures out how good was the game from my perspective
    d. Transition function $t(s,a) \rightarrow t(s,p,a)$
        i. Used to say if I am in state s, take action 'a' that will produce the new state
        ii. Since there are multiple perspectives, add player
    e. Previously talked about goal function g(s)
        i. More general term: terminal test function terminal – test(s)
        ii. Since the game can end in bad state (I can lose), update the word to terminal - test(s)
    f. Game is a 6 tuple now:
        i. Initial state
        ii. Set of players
        iii. Actions available to each player in each state (assume same actions for now)
            1. There can be player specific actions

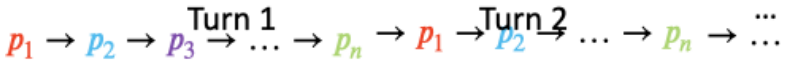
2. There are state specific actions – if you are in certain state, you can do certain things

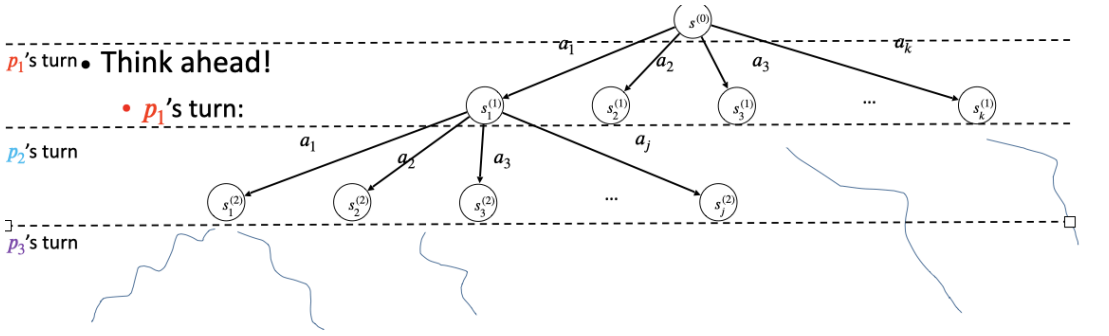
iv. Terminal - test(s)

v. u(s,p) – utility

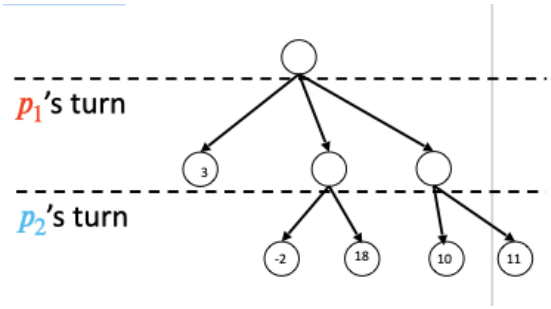
1. I do not know the true goodness of state only when the game ends
2. Hard to know the true goodness of state since it depends on the future and the actions of the players done

vi. t(s,p,a)

3. How can We Make Better Choices?
   a. Lets assume game has known order of turns
      i. Each player goes one at a time (if one player moves, other players cannot move – monopoly, chess, ordering at the same time, etc.)
      ii. 

      iii. Static world (world pauses while each agent thinks) – for now
   b. Think ahead! (p1's perspective)
      i. 

      ii. s(0) is the current state
      iii. p1 can take options of k actions
      iv. Now it is p2's turn
      v. p2 can take options of j actions
      vi. This process continues for p3, p4, and etc.
4. What to do?
   a. Can expand this tree all the way down to terminal states (until the leaf nodes of the trees are at the terminal state)
      i. Pick the path that ends with a terminal state that's good for us?
   b. Why doesn't this work?
      i. Practical problems:
         1. Tree is massive!
         2. Takes too long to expand the whole thing
         3. I do not know what the opponents will do - do not control the actions of other players

4. 

5. For example, here, if I move to the middle move, p2 has the potential to move to 18, which is good for me (bad for p2) but p2 might not move to 18 and might also move to -2 (which is bad for me) → this is not controlled by me
6. And also, p2, who's goal is go against p1, will probably go to -2

ii. Theoretical problems:
1. We only know utility value of terminal states:
a. How do we determine utility values of non-terminal states?
b. Combinatorial ways of combinations (b/c of lots of players)

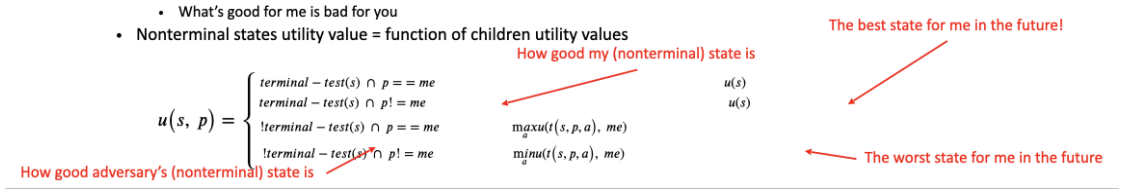
c. Need:
i. Faster tree algorithms
ii. Polynomial time ways of combining other player's move(s)

5. Simplify (for now)
a. Consider a 2-player game
i. agent1 vs agent 2 (competitive/adversarial game)
1. Adversary wants to force me to go to bad states for me
2. I want to force adversary in bad state (for them)

b. Zero sum game (a major simplifying assumption)
i. My loss is your gain (and vice versa)
ii. "true utilities" always sum to same constant (may not be zero!)

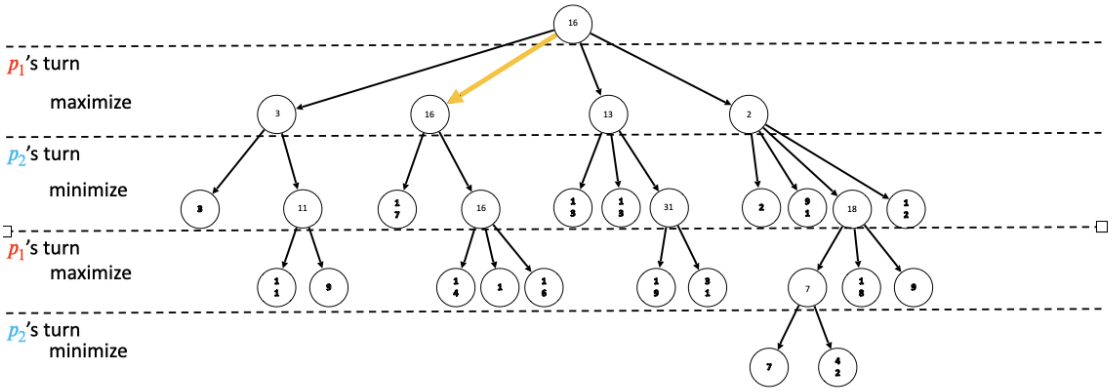
c. With these two assumptions:
i. Tree is much smaller
ii. Don't need to worry about combining multiple adversaries together (do not need to think about the enemy since what is good for me is bad for them) – only need my perspective
1. What's bad for me is good for you
2. What's good for me is bad for you

iii. Nonterminal states utility value = function of children utility values

iv. 

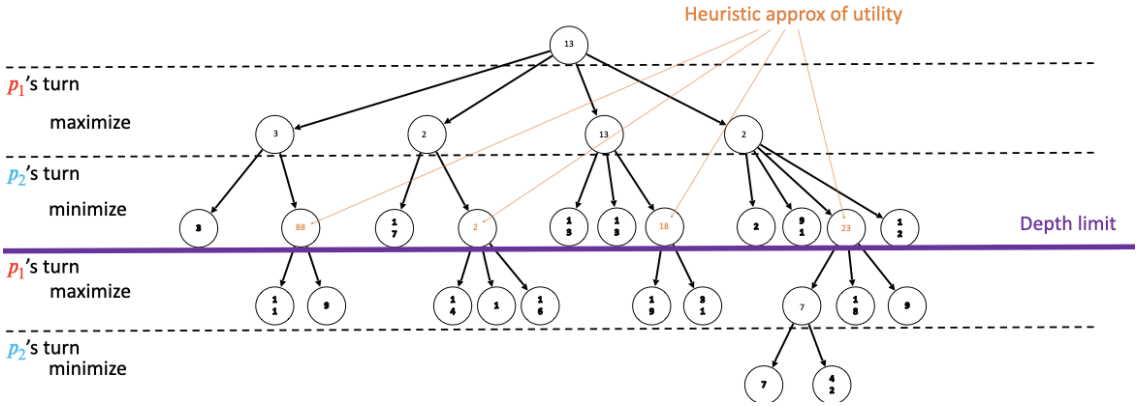
v. My max state = enemy's worst state
vi. My min state = enemy's best state
vii. This is because of zero sum

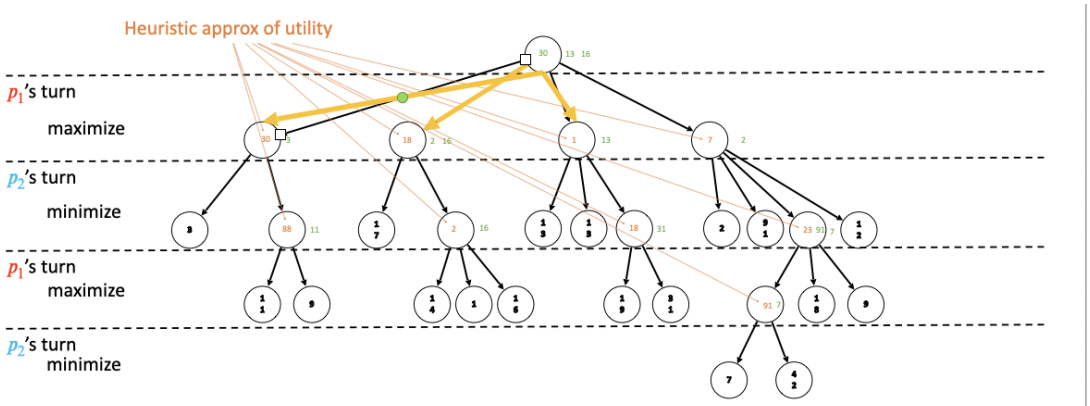
6. The Minimax Algorithm
    a. Expands tree using DFS
    b. Once you get to the terminal state, then you can go back to the parent node and assign utilities
    c. At root, pick action which leads to the best child state
    d. Only run minimax when it is my turn (I am always at the root node)



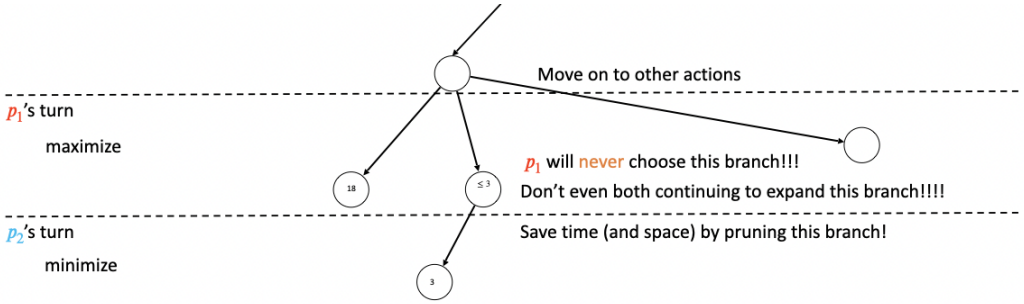
    e.
    f. I do not know what the enemy will do but they will want to do what is worst for me

7. Is this a good model?
    a. Assumes adversary always makes the "best" choice
        i. Humans are notorious for not doing this (humans cannot figure out relative goodness in a good fashion if they are somewhat equal)
        ii. How you beat chess bots!
            1. Hint: don't play chess with them (you will lose)
            2. Instead:
                a. Take advantage of time limits: make them think
                    i. If they run out of time they might play sub-optimally
    b. Major problem:
        i. Even these trees can be too big
        ii. Idea: don't go all the way to terminal states!
            1. Ex. only plan "3 moves ahead" or something
            2. Need a stand in for utility values of leaf nodes in the tree
                a. Heuristics
                b. Heuristics can also be wrong but we do not have choice

8. Depth-Thresholded Minimax
    a. 

    b. The heuristic function provides approximate answer
    c. Heuristic is wrong but I need to know somehow
    d. Therefore, ask heuristic and hope it is good
    e. The quality of heuristic matters when there is a lot of future events that could have happened
9. Iterative Deepening
    a. 

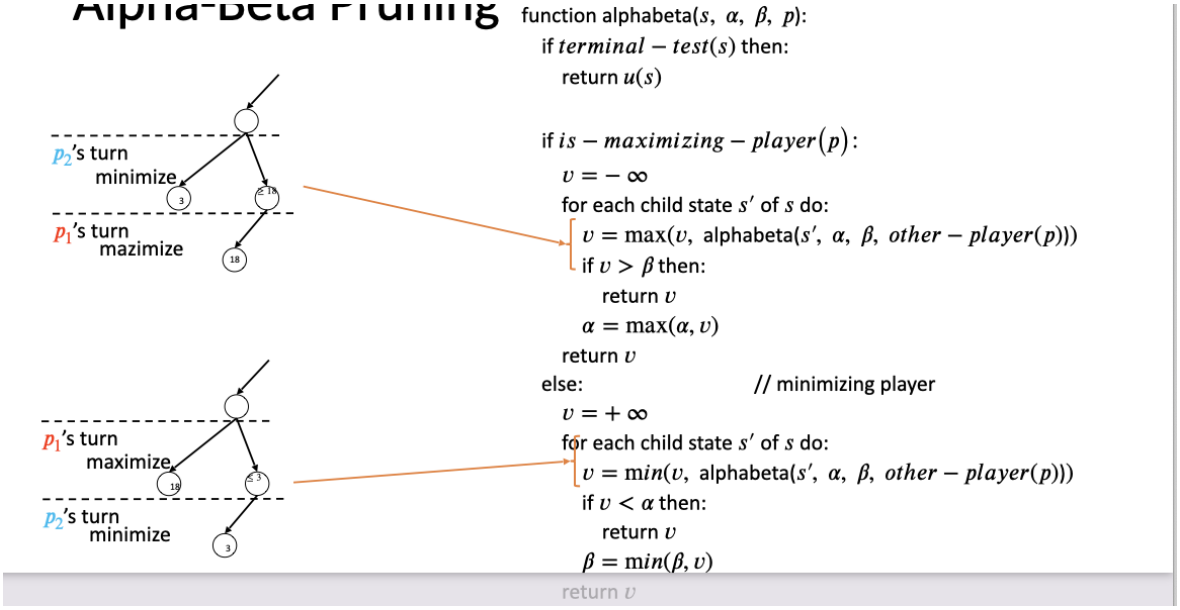
    b. Expand the tree using BFS
        i. Use heuristics to determine best move at a level
        ii. If you have more time/resources: do next level
        iii. Whenever resources run out: return best choice so far
    c. Gives us insurance that if we made bad decisions due to the heuristic function giving bad state, it does not affect the choice we make unless we run out of memory
10. Problem
    a. Minimax is inefficient
        i. Inefficiency is subtle
            1. Iterative deepening / depth-thresholding don't solve it

b. 

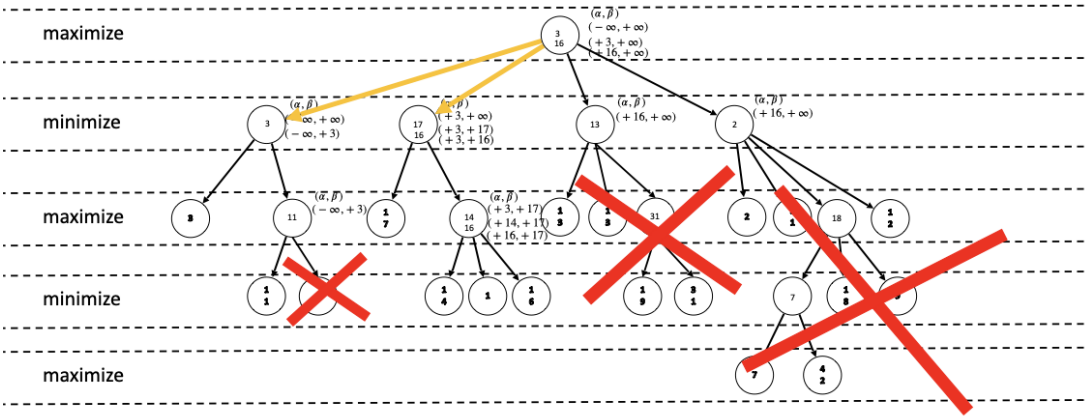
   i. Since the opponent's best answer here is 3 (minimize the goal), the opponent will always choose 3 since it is the minimum.
   ii. Then, p1 will choose 18 instead of 3 (maximizing the goal) and in that case, we do not need to expand the branch with 3 since 18 will always get chosen
   iii. Therefore, we can save time and space by getting rid of the branch since we will never choose that

11. Alpha-Beta pruning
   a. How do we detect scenarios like this?
      i. Need to keep track of "best choice" for each player
      ii. a = min score of the maximizing player (me) so far
         1. Max player is guaranteed at least this score
         2. When min player:
            If I encounter child state < a:
               Max player will never get here
               Stop expanding this branch!
      iii. Beta = max score of the minimizing player (adversary) so far
         1. Min player is guaranteed at most this score
         2. When max player:
            If I encounter child state > Beta:
               Min player will never get here
               Stop expanding this branch
      iv. Every node gets its own pair of values (a, Beta)
      v. Child nodes inherit values of parents
         1. Child can change it's own values of (a, Beta) values
      vi. Alpha-Beta pruning is Minimax with "early stopping"

b.



c.



12. Variants of Alpha-Beta Pruning
    a. Even though we prune: Alpha-Beta still examines all the way to terminal states
    b. Solved this with Minimax:
        i. Depth threshold
        ii. Iterative deepening
    c. These solutions work with Alpha-Beta too
13. Problems with Alpha-Beta Pruning
    a. Even with Depth-Thresholding/Iterative Deepening:
        i. Order in which child states are enumerated matters!
            1. Pruning only occurs when we know a better option already exists!
            2. What if we see better options last?
                a. Never prune!
                b. Without pruning, Alpha-Beta is just Minimax!

b. How can we speed this up?
    i. Transposition table:
        1. Cache utility value for states we've seen
        2. When we encounter the same state in the future, no need to expand!
        3. How many states should we cache?
    ii. How can we encourage Alpha-Beta to prune?
        1. More heuristics!
            a. Impose an order on child enumeration
            b. Children we think are better choices should come first in the order!