Adversarial Search

Adversarial Search

- The Adversarial search is a well-suited approach in a **competitive environment**, where two or more agents have conflicting goals.
- The adversarial search can be employed in **two-player** games which means what is good for one player will be the misfortune for the other.
- In artificial intelligence, adversarial search plays a vital role in **decision-making**, particularly in competitive environments associated with games and strategic interactions.
- Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.
- **Adversarial Search in Game Playing:** In which we examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.

Assumpations

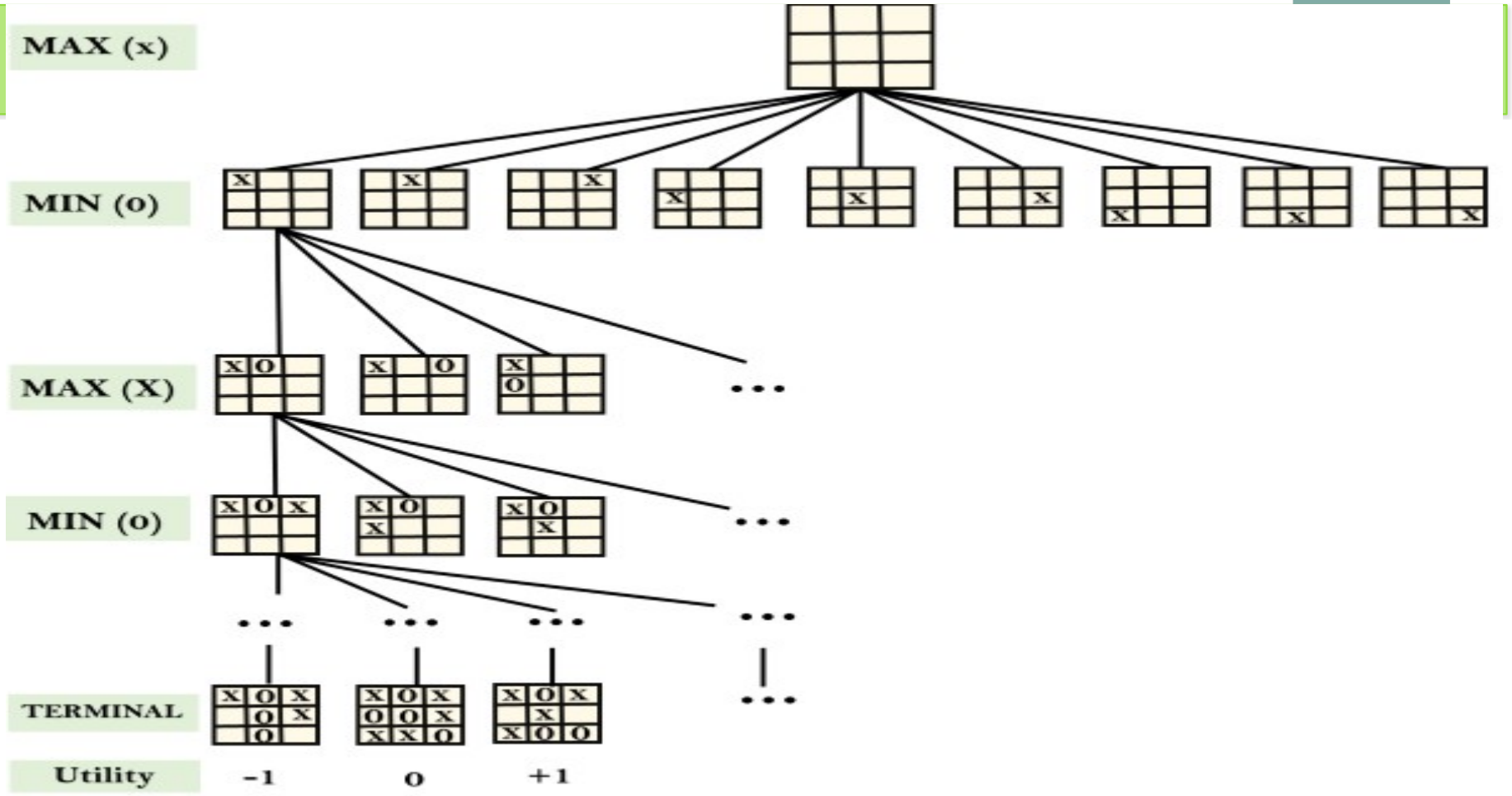
- 2 agents whose actions alternate – 2 player games with alternate moves
- Values for each agent are the opposite of the other which creates adversarial situation.
- Fully observable environments.
- Using dice is not involved
- Clear rules for legal moves
- Well defined moves
- Example: Tic Tac Toe, Checkers, Chess

How to strategize in each game?

- Cosider all the legal moves you can make
- Each move leads to a new board configuration
- Evaluate each resulting position and determine which is best
- Make that move
- Wait for the opponent to move and repeat

Game Tree

- A game tree is a tree where **nodes of the tree are the game states** and **Edges of the tree are the moves** by players.
- Game tree involves initial state, actions function, and result Function.
- Example: Tic-Tac-Toe game tree
- **Following are some key points of the game:**
  - There are two players MAX and MIN.
  - Players have an alternate turn and start with MAX.
  - MAX maximizes the result of the game tree
  - MIN minimizes the result.


MAX (x)
MIN (o)
MAX (X)
MIN (o)
TERMINAL
Utility
-1
0
+1

- From the initial state, **MAX has 9 possible moves**. **MAX place x and MIN place o**, and both player plays alternatively until we reach a leaf node where one player has three in a row or all squares are filled.
- Both players will compute each node, minimax, the minimax value which is the best achievable utility against an optimal adversary.
- Suppose both the players are well aware of the tic-tac-toe and playing the best play. Each player is doing his best to prevent another one from winning. **MIN is acting against Max in the game.**
- One layer of Max, One layer of MIN, and each layer is called as Ply. Max place x, then MIN puts o to prevent Max from winning, and this game continues until the terminal node.
- In this either MIN wins, MAX wins, or it's a draw. This game-tree is the whole search space of possibilities that MIN and MAX are playing tic-tac-toe and taking turns alternately.
-

MIN-MAX Algorithm

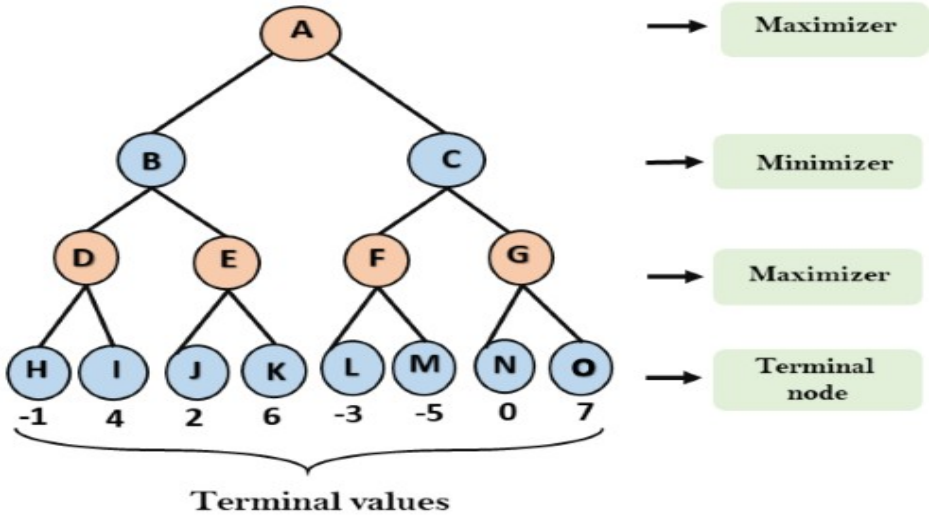
- Mini-max algorithm is a **recursive or backtracking algorithm** which is used in decision-making and game theory.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

MIN-MAX Algorithm

- **MAX will try to maximize its utility (Best Move)**
- **MIN will try to minimize its utility (Worst Move)**

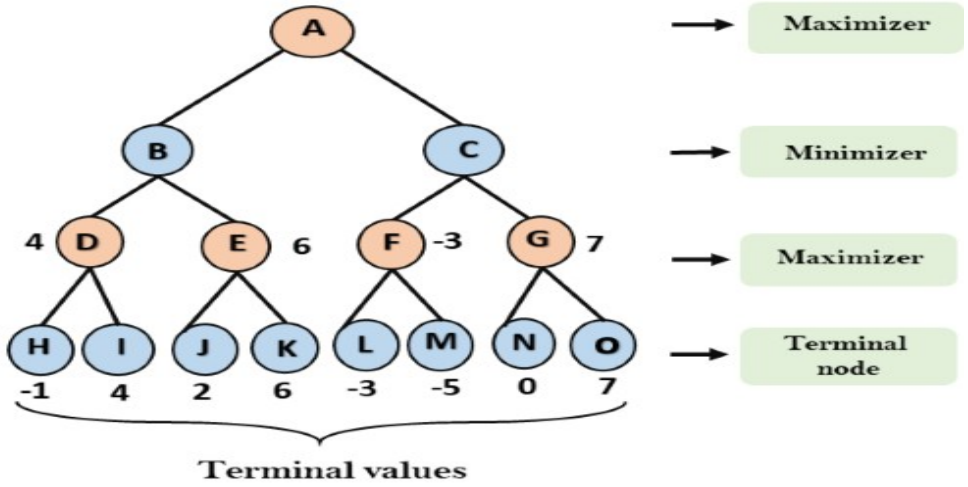
MIN-MAX Algorithm

- Step 1: the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states.



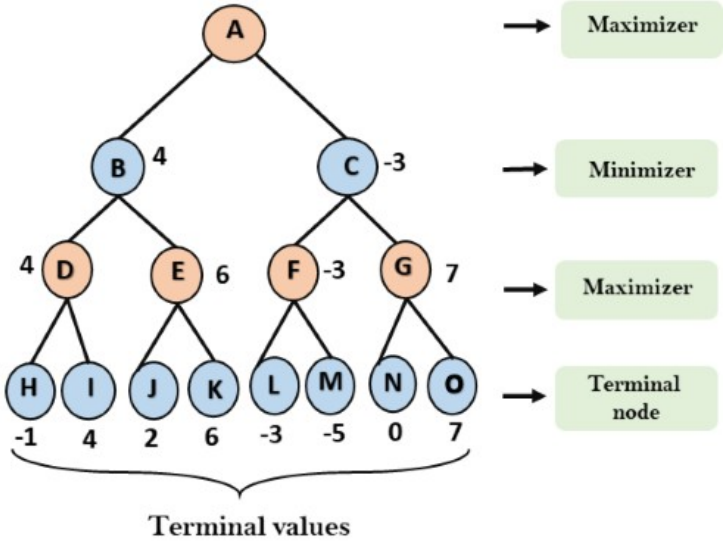
MIN-MAX Algorithm

- Step 2 -First we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.



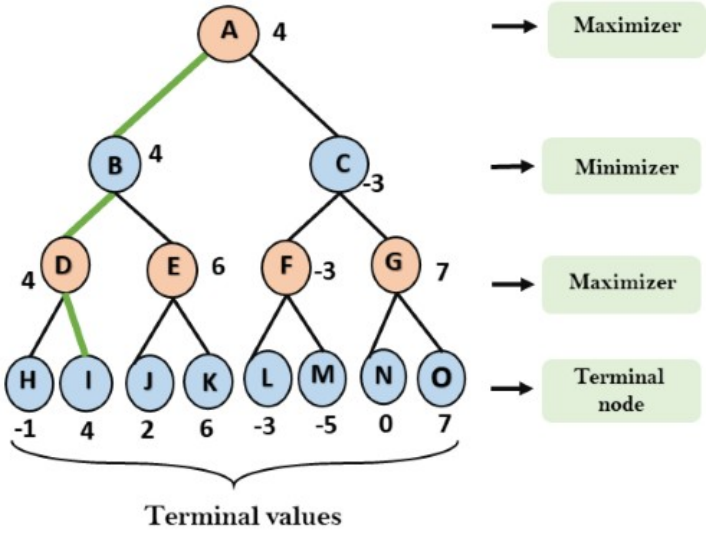
MIN-MAX Algorithm

- Step 3- it's a turn for minimizer, so it will compare all nodes value and will find the 3rd layer node values.
- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3



MIN-MAX Algorithm

- Step 4- It's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node.



Limitation

- It gets really slow for complex games such as Chess, go, etc.
- This type of games has a huge branching factor, and the player has lots of choices to decide.

Alpha-beta pruning

- It is an optimization technique for a minimax algorithm.
- It reduces computation time by a huge factor, allowing the user to traverse faster and deeper into the tree.
- It stops evaluating when at least one possibility has been found that typically proves the move to be worse than the previously examined move.
- The minimax search is based on depth-first search which considers the nodes along a single path in a tree. But Alph-Beta pruning bonds with two major parameters in **MAX-VALUE(state, alpha, beta), representing Alpha plays a maximizer role, whereas Beta plays a minimizer role.**

Alpha-beta pruning

- Alpha – denotes the value of the **best or highest value**.
- Beta – denotes the value of the **best or lowest value**.
- By efficiently **pruning away branches of the tree that are known to be irrelevant**, alpha-beta pruning dramatically speeds up the search process, making it an essential component of game-playing AI.

Alpha-beta pruning

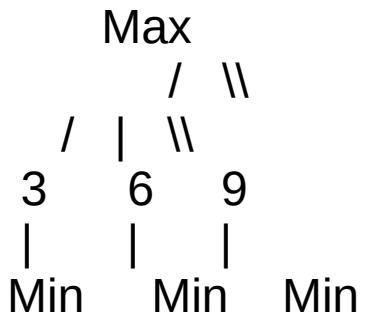
- Alpha (α): This represents the best value achievable by the maximizing player (Max) along the path from the root to the current state.
- Beta (β): This represents the best value achievable by the minimizing player (Min) along the path from the root to the current state.
- As the search progresses, the algorithm continuously updates alpha and beta to maintain the best-known values for Max and Min.
- When Max is considering a move, it updates alpha with the maximum value found so far. If alpha becomes greater than or equal to beta ($\alpha \geq \beta$), Max knows that the opponent (Min) will never allow this move, and there's no need to explore further. Therefore, the algorithm prunes the rest of the subtree under this node.
-

Alpha-beta pruning

- Similarly, when Min is considering a move, it updates beta with the minimum value found. If beta becomes less than or equal to alpha ($\beta \leq \alpha$), Min knows that Max will never allow this move, and the algorithm prunes the subtree.
- α (alpha) = 9, which is the best value Max has seen.
- The algorithm now knows that Min will not choose values greater than or equal to 9 because doing so would lead to pruning in Max's

  earlier moves. As a result, Min's options are irrelevant,

  and the remaining branches can be pruned.



Case Study: Cybersecurity Threat Detection

- **Scenario**: In the realm of cybersecurity, organizations face a constant adversarial battle against cyber threats and attackers. Adversarial search is used to model and mitigate these threats efficiently.
- **Application**: Consider an organization that employs adversarial search to enhance its threat detection system. In this case:
- **Maximizing Player (Defender)**: The organization plays the role of the maximizing player (Max) and seeks to maximize the security of its systems and protect its data.
- **Minimizing Player (Attacker)**: The attacker, who may be a hacker or malicious entity, is the minimizing player (Min), aiming to exploit vulnerabilities and breach the organization's security.