PROBLEM SOLVING THROUGH INFORMED SEARCH

Adversarial Search

Min-Max Procedure

Static Evaluation Function

Alpha Beta Pruning

ADVERSARIAL SEARCH

Adversarial search is used when dealing with problems involving multiple agents with conflicting goals. In adversarial settings, the agents are competitors or opponents, such as in two-player games like chess or tic-tac-toe. In these cases, each agent aims to maximize their own success while minimizing the success of the opponent.

Key Concepts:

Zero-sum games: In adversarial search, we often focus on zero-sum games, where one player's gain is another's loss.

ADVERSARIAL SEARCH

Game tree: A game is represented as a tree where each node represents a possible game state, and each branch represents a player's action.

Ply: Each player’s move is called a ply.

Adversarial search is typically implemented using a **Min-Max algorithm** to make the best move, which we'll now explore in detail.

MIN-MAX PROCEDURE

The Min-Max procedure is a recursive algorithm for choosing the optimal move in a two-player game. One player (MAX) tries to maximize the score, while the other player (MIN) tries to minimize it. The goal of MAX is to choose a move that leads to the best possible outcome, assuming the opponent MIN will also play optimally to minimize MAX's outcome.

Procedure:

Tree Construction: Starting from the current game state, a game tree is built, where MAX and MIN alternate turns.

MIN-MAX PROCEDURE

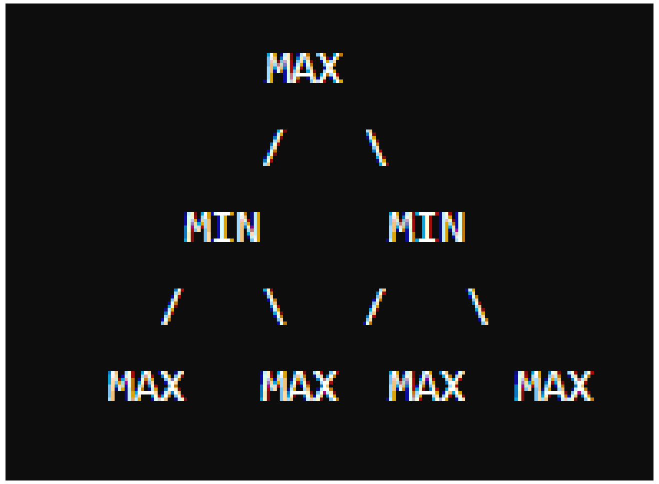
Maximization for MAX: MAX's goal is to maximize the minimum payoff, anticipating MIN’s best possible counter-moves.

Minimization for MIN: MIN's goal is to minimize the maximum payoff, assuming MAX will play optimally.

Recursive Calculation: The algorithm evaluates the terminal states (leaves) of the tree using a static evaluation function (discussed next). Then, it propagates the values back up the tree, alternating between MIN and MAX.

EXAMPLE

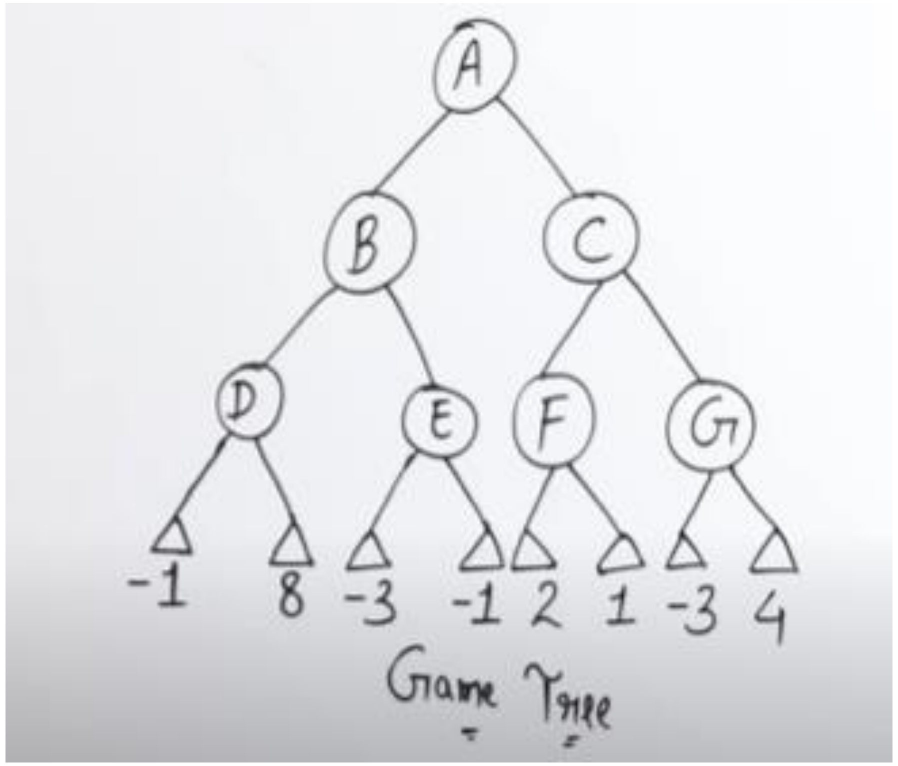
In a simplified game with two players, the game tree might look like this:



```
          MAX
         /   \
      MIN     MIN
     /   \   /   \
  MAX   MAX MAX   MAX
```

MAX's goal is to find the path that leads to the highest possible score, while MIN will try to minimize this score. The Min-Max algorithm helps MAX choose the best move by evaluating the worst-case scenarios for each branch of the game tree.

EXAMPLE



STATIC EVALUATION FUNCTION

The Min-Max algorithm evaluates game outcomes based on terminal nodes in the game tree, which represent final states (like winning or losing a game). However, most games are too complex to evaluate every possible outcome because the tree grows exponentially. This is where the **static evaluation function** comes into play.

The static evaluation function provides a heuristic estimate of the value of a non-terminal state (a state that isn’t the end of the game). In chess, for example, the static evaluation function might count material (pieces on the board) or evaluate the positional advantage. It allows the Min-Max algorithm to evaluate incomplete game states and make informed decisions without fully expanding the game tree.

STATIC EVALUATION FUNCTION

Characteristics of a Good Evaluation Function:

Speed: It should be fast to compute.

Accuracy: It should provide a value that reflects the likelihood of winning or losing from the current state.

Domain-specific: It's usually designed with specific features of the game in mind.

ALPHA-BETA PRUNING

While the Min-Max algorithm is effective, it can be computationally expensive since it has to search the entire game tree. **Alpha-Beta pruning** optimizes the Min-Max algorithm by eliminating branches in the game tree that won’t affect the final decision. This significantly reduces the number of nodes evaluated and speeds up the search process.

Key Terms:

Alpha (α): The best (highest) value that MAX can guarantee up to that point.

Beta (β): The best (lowest) value that MIN can guarantee up to that point.

ALPHA-BETA PRUNING

Pruning Process:

Alpha-beta values are propagated: As the search progresses, alpha and beta values are updated for each node.

Cutoff (Pruning): If at any point, a node's value becomes worse than the current alpha or beta value, the algorithm prunes (cuts off) that branch. This means that it stops evaluating that branch since it won’t influence the decision.

EXAMPLE

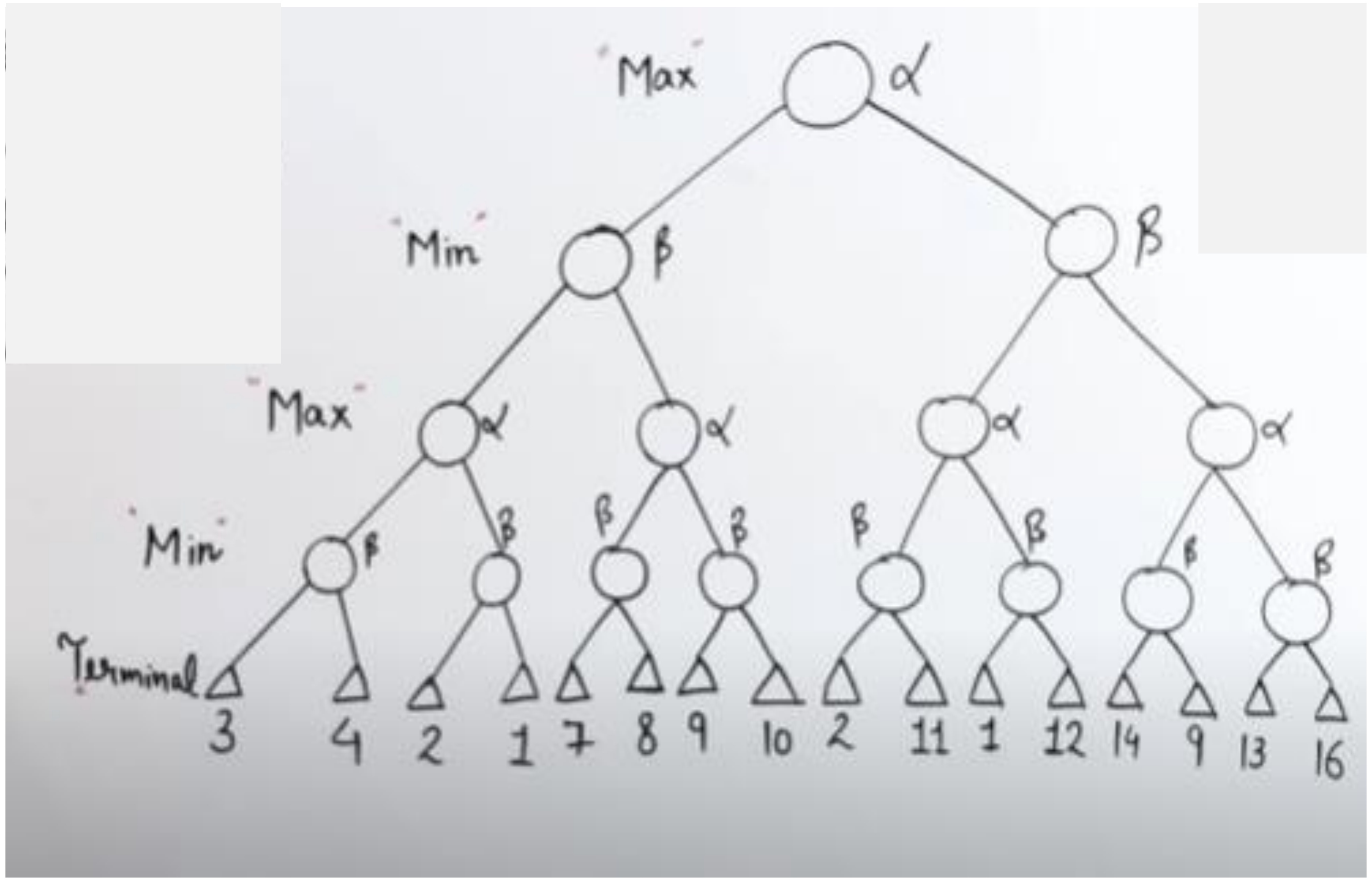
Consider a game tree with MAX at the top. If a MIN node has a value that is less than the current alpha, then MAX would never allow that node to be chosen, and the rest of the subtree under that MIN node can be ignored.

Benefits:

Efficiency: Alpha-beta pruning makes the Min-Max algorithm more efficient by reducing the number of nodes that need to be explored.

Same result as Min-Max: Importantly, alpha-beta pruning does not affect the outcome—it still returns the same result as the full Min-Max procedure but faster.

EXAMPLE


Max
α
Min
β
β
Max
α
α
α
α
Min
β
β
β
β
β
β
β
β
Terminal
3
4
2
1
7
8
9
10
2
11
1
12
14
9
13
16

SUMMARY OF KEY POINTS

Adversarial Search: This search method is applied to environments where agents (like players in a game) are competing with each other.

Min-Max Procedure: A recursive strategy used by agents to maximize their minimum guaranteed payoff in a game.

Static Evaluation Function: A heuristic tool to evaluate non-terminal game states, allowing the Min-Max algorithm to make decisions without exploring the entire game tree.

Alpha-Beta Pruning: A technique that optimizes the Min-Max procedure by pruning branches that cannot affect the final decision, greatly improving efficiency.