What are Games in AI?

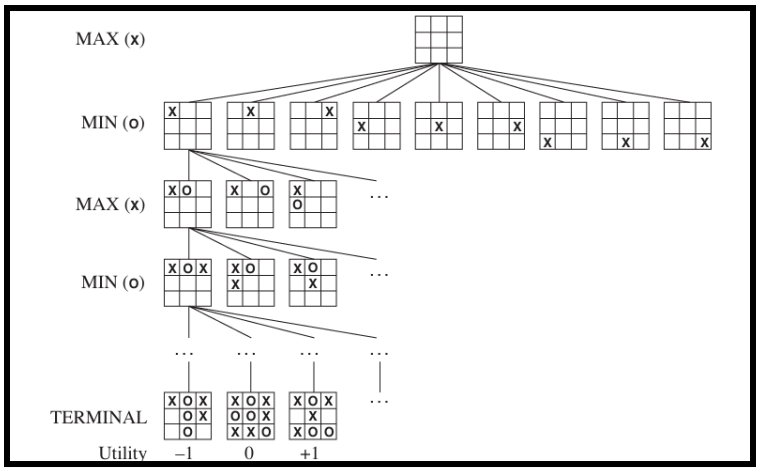
Games in AI are situations where two or more players (agents) compete, each trying to achieve their goals. These games are examples of **multiagent environments**, meaning that one player's actions can affect another player's outcomes. Think of games like chess or tic-tac-toe, where you must consider not only your moves but also your opponent's.

Competitive Environments

- In chess, if one player wins, the other loses. This makes chess a **zero-sum game**, meaning the **total score of all players adds up to zero. One player's gain is the other's loss.**

Key Concepts in Game Analysis

1. **Game Tree**:
    - A **game tree** shows all possible moves and outcomes in a game.
    - Example: In tic-tac-toe, the tree starts with the empty board, then branches out based on the available moves (like placing an "X" or "O").



2. **Players**:
    - Players take turns making moves. One player is often called **MAX** (trying to maximize their score), and the other is **MIN** (trying to minimize MAX's score).
3. **Terminal States**:
    - These are end points of the game where a winner is decided, or the game ends in a draw.
4. **Utility Function**:
    - This assigns a numerical value to terminal states. For example:
        - Win = +1
        - Loss = -1
        - Draw = 0

Games as Search Problems

In AI, games can be thought of as **search problems** where the goal is to find the best move to win. Here are the key elements that define a game:

1. **Initial State (S_0)**:
    - This describes the starting point of the game. For example, in chess, the board is set up with all pieces in their starting positions.
2. **PLAYER(s)**:
    - Specifies whose turn it is to make a move in a given state. For instance:
        - In chess, either "White" or "Black" is the active player.
3. **ACTIONS(s)**:
    - Lists all the legal moves available in a given state. For example:
        - In tic-tac-toe, if a square is empty, placing an "X" or "O" there is a legal action.
4. **RESULT(s, a)**:
    - This defines what happens after a player takes an action. It gives the **new state** after the move is made.
        - Example: In chess, moving a knight from one square to another updates the board to reflect the new arrangement.
5. **TERMINAL-TEST(s)**:
    - A test to check if the game is over (i.e., a terminal state is reached). Terminal states include:
        - A win: One player achieves victory (e.g., checkmate in chess).
        - A draw: Neither player can win (e.g., a stalemate in chess or a full tic-tac-toe board with no winner).
6. **UTILITY(s, p)**:
    - The **utility function** assigns a numeric value to terminal states for each player:
        - Win: +1
        - Loss: -1
        - Draw: 0
        - In more complex games like backgammon, utility values can range widely (e.g., 0 to +192).

What is the Minimax Algorithm?

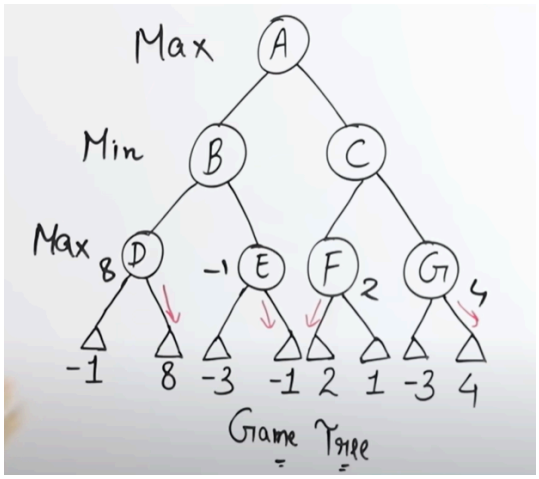
The **Minimax Algorithm** is a decision-making process used in games to find the best move for a player, assuming the opponent also plays optimally. The algorithm alternates between:

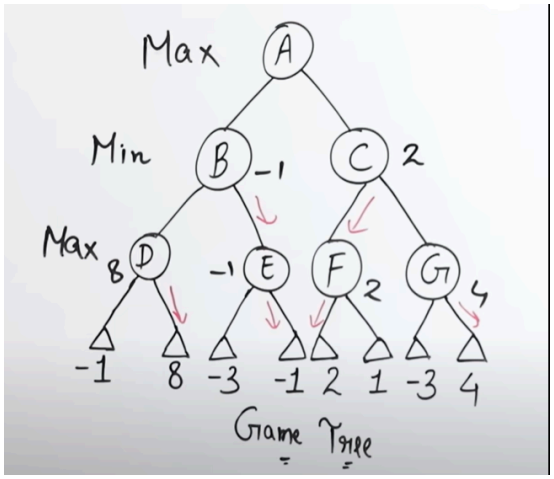
1. **Maximizing** the score for the current player (called MAX).
2. **Minimizing** the score for the opponent (called MIN).

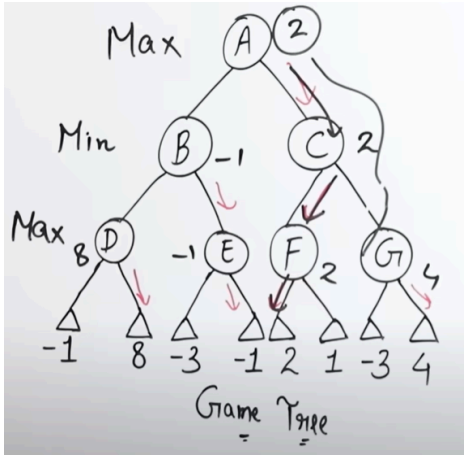
Minimax evaluates a **game tree**—a structure representing all possible moves in the game—by:

1. Starting at the **leaf nodes** (end states of the game).
2. **Backing up values** (scores) to higher levels based on the rules:

- MAX chooses the **highest score** at its turn.
- MIN chooses the **lowest score** at its turn.
- It is a backtracking algorithm.







Alpha–Beta Pruning: Simplifying Minimax Search

Alpha–Beta Pruning is an optimization technique for the minimax algorithm that reduces the number of nodes evaluated in the game tree, making the process faster without changing the final result.

What is Alpha–Beta Pruning?

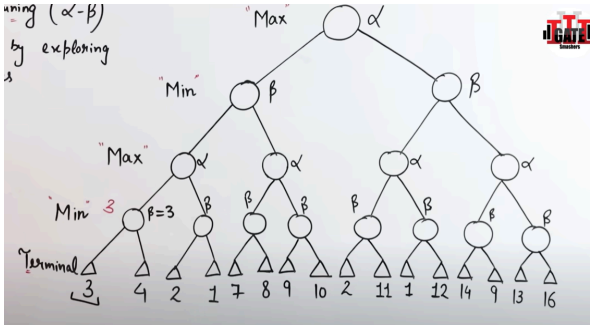
Alpha–Beta Pruning helps skip parts of the game tree that cannot influence the final decision. Instead of evaluating every node (like minimax does), it **prunes** branches that are irrelevant to the final outcome.

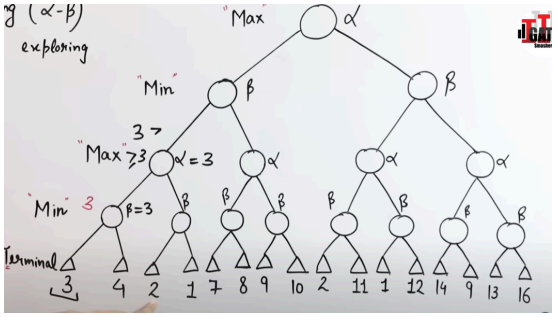
- **Alpha (α)**: The best value that MAX can guarantee so far.
- **Beta (β)**: The best value that MIN can guarantee so far.

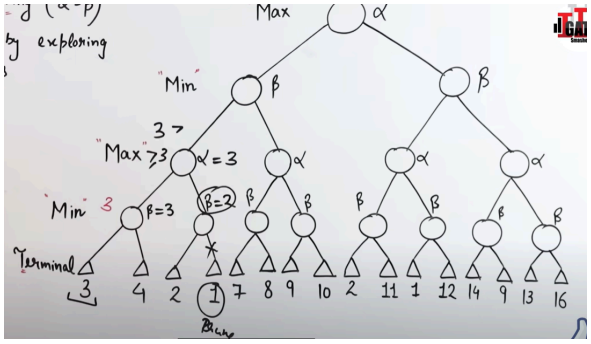
How Alpha–Beta Pruning Works

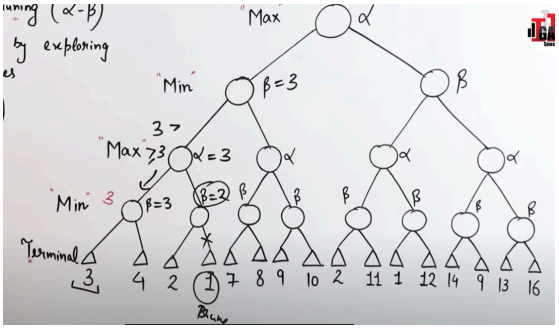
Let’s walk through the process step by step:

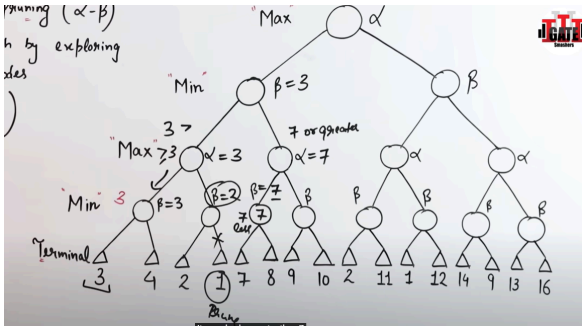
1. **Initial Setup**:
    - Start with **α = -∞** (worst-case scenario for MAX).
    - Start with **β = +∞** (worst-case scenario for MIN).
2. **Recursive Tree Search**:
    - As the algorithm explores the game tree, it updates α and β based on the best scores seen so far.
    - If a branch’s score is **worse** than the current α or β, the algorithm **stops exploring that branch** (prunes it).
3. **Pruning Logic**:
    - MAX prunes branches where it is clear that MIN will force a worse outcome than MAX's current best (α).
    - MIN prunes branches where it is clear that MAX will force a worse outcome than MIN's current best (β).

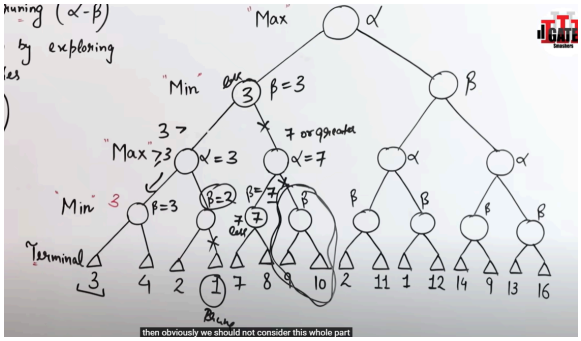


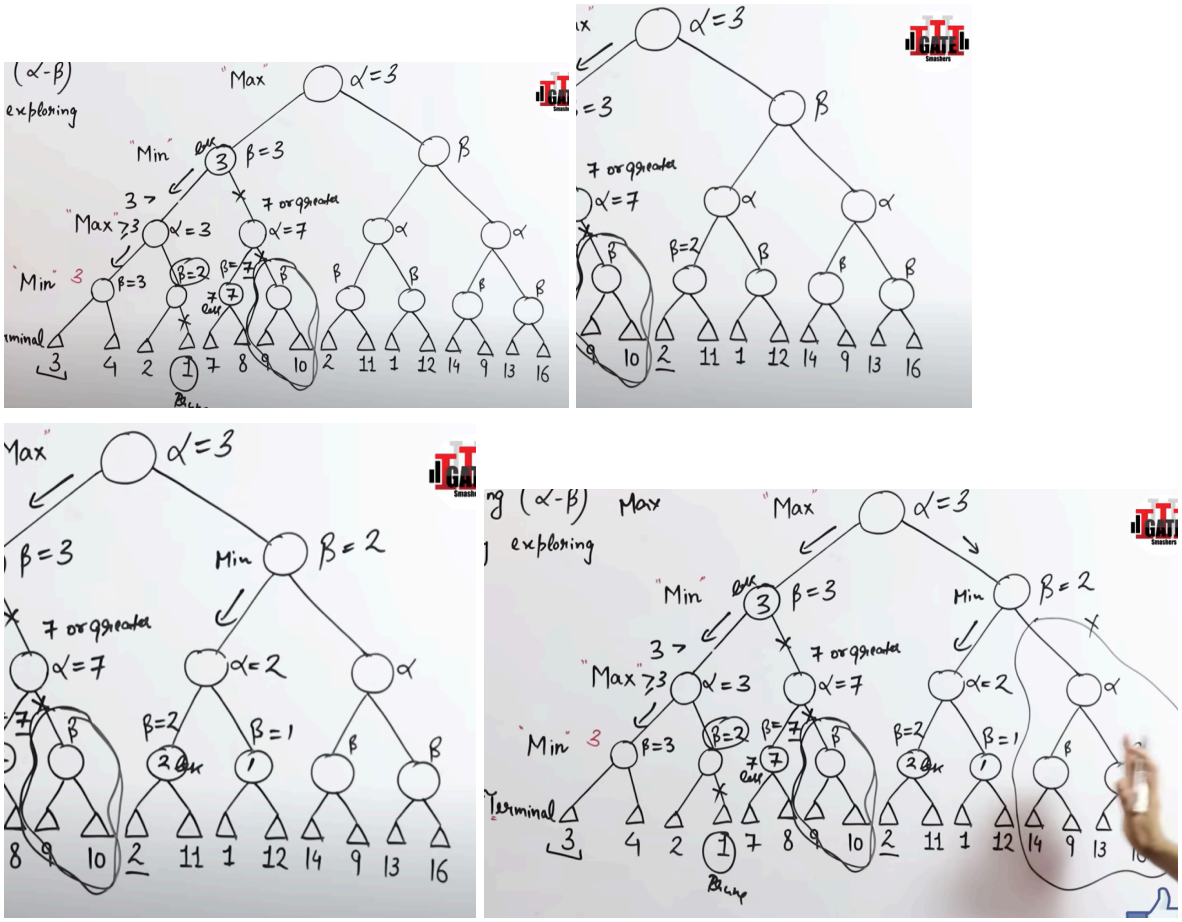


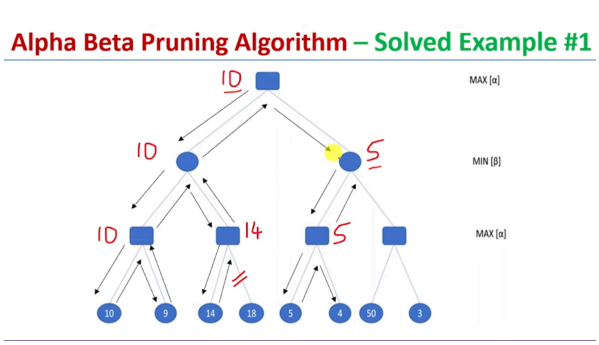


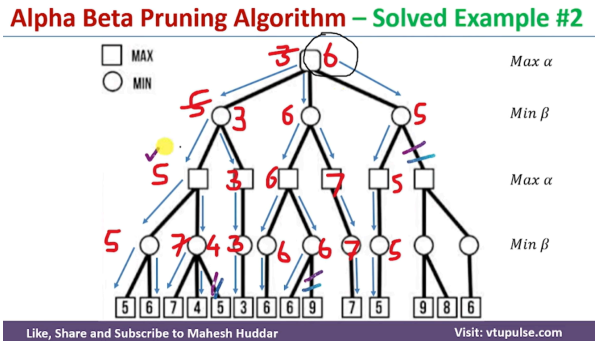












Key Benefits of Alpha–Beta Pruning

1. **Saves Time**:
   - By pruning unnecessary branches, it significantly reduces the number of nodes explored.
   - In the best case, it reduces the complexity to **O(b^(m/2))**, effectively doubling the depth of the tree you can explore compared to minimax alone.
2. **Same Result as Minimax**:

- Alpha–Beta pruning does not affect the final decision—it simply makes the process more efficient.