**Name: Pranay Harne**

**Roll no. : 381008**

**PRN : 22310103**

**Assignment - 5**

**Minimax Algorithm for Game Playing**

**Aim -**To implement the Minimax algorithm for decision-making in two-player games, allowing the computer to make optimal moves while considering the opponent's strategy.

**Objectives -**

1. To understand the Minimax algorithm and its application in game theory.
2. To implement the Minimax algorithm in a simple game (e.g., Tic-Tac-Toe).
3. To analyze the performance and effectiveness of the Minimax algorithm in gameplay.
4. To explore enhancements to the Minimax algorithm, such as alpha-beta pruning.

**Theory -**

1. **What is minimax algorithm?**

The Minimax algorithm is a decision-making algorithm used in two-player games to determine the optimal move for a player assuming that the opponent also plays optimally. It is based on the premise of minimizing the possible loss for a worst-case scenario (hence the name "minimax"). The Minimax algorithm constructs a game tree, representing all possible moves, and evaluates them to find the best possible outcome for the maximizing player while anticipating the minimizing player's strategy.

1. **Significance of minimax algorithm**

The significance of the Minimax algorithm lies in its ability to provide a systematic approach to decision-making in competitive environments. It helps AI agents or players make optimal moves in games by evaluating potential future states of the game, thereby increasing their chances of winning. This algorithm is foundational in artificial intelligence for game playing and is widely used in developing intelligent game opponents.

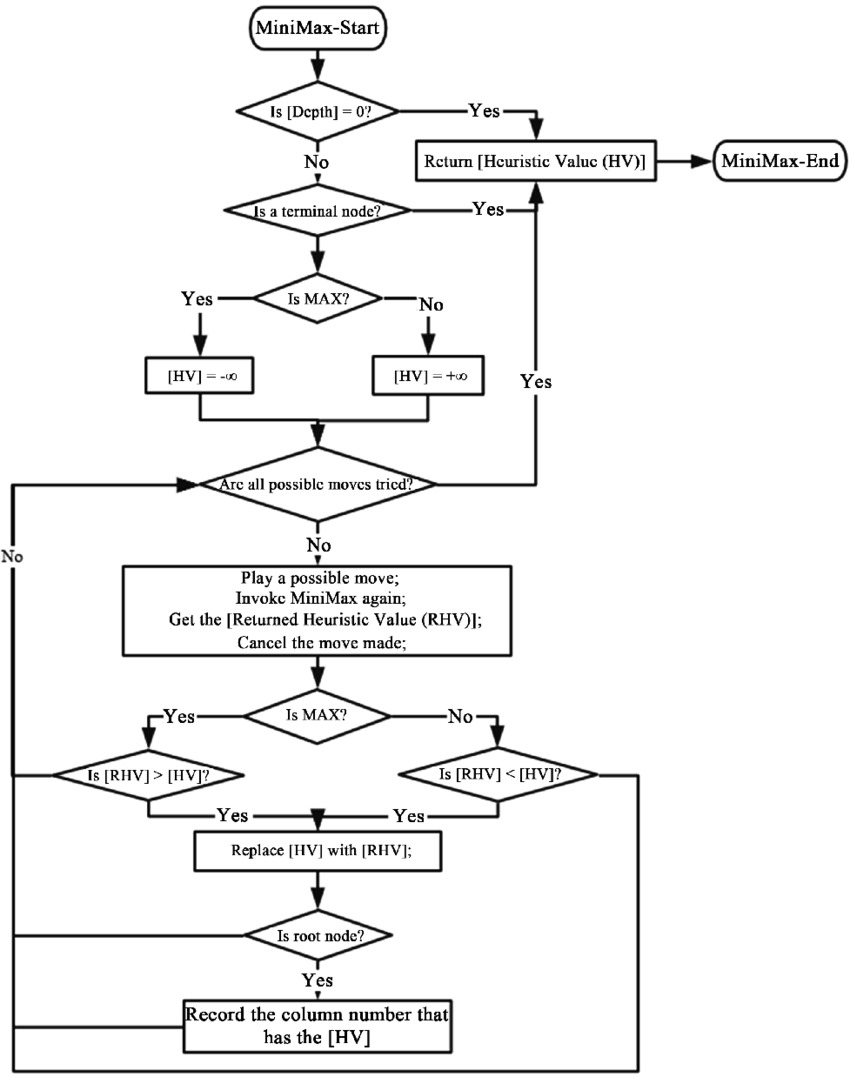
1. **Key concepts included**

* **Game Tree:** A tree structure that represents all possible moves in a game. Each node represents a game state, and edges represent player moves.
* **Nodes:** Represent game states where decisions are made.
* **Leaves:** Terminal states of the game (end conditions) that provide a utility score (win, lose, or draw).
* **Utility Function:** A function that assigns a numerical value to terminal states, representing the desirability of that state for a player.
* **Minimizing Player:** The player trying to minimize the score (usually the opponent).
* **Maximizing Player:** The player trying to maximize their score (the player using the Minimax algorithm).

1. **Algorithm**
2. Initialize the game state and define the utility function for terminal states.
3. Create a Minimax function:

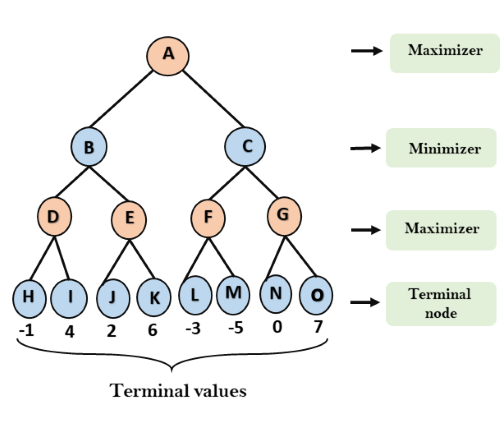
* If the game is over (terminal state), return the utility value.
* If it's the maximizing player's turn:
* Initialize the maximum value to negative infinity.
* For each possible move, recursively call the Minimax function and update the maximum value.
* If it's the minimizing player's turn:
* Initialize the minimum value to positive infinity.
* For each possible move, recursively call the Minimax function and update the minimum value.

1. Return the optimal value for the current state.
2. Implement the main function to execute the Minimax algorithm and determine the best move.
3. **Workflow diagram**

****

1. **Working of minimax algorithm through an example**

The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.



In this example, there are two players one is called Maximizer and other is called Minimizer. Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score. This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes. At the terminal node, the terminal values are given so we will compare those values and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

1. **Step-1:** In the first step, the algorithm generates the entire game-tree and applies the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A as the initial state of the tree. Suppose maximizer takes the first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.
2. **Step 2:** Now, first we find the utility value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with the initial value of Maximizer and determine the higher nodes values. It will find the maximum among them all.

* For node D max(-1,- -∞) => max(-1,4)= 4
* For Node E max(2, -∞) => max(2, 6)= 6
* For Node F max(-3, -∞) => max(-3,-5) = -3
* For node G max(0, -∞) = max(0, 7) = 7

1. **Step 3:** In the next step, it's a turn for minimizer, so it will compare all node values with +∞, and will find the 3rd layer node values.

* For node B= min(4,6) = 4
* For node C= min (-3, 7) = -3

1. **Step 4:** Now it's a turn for Maximizer, and it will again choose the maximum of all node values and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

* For node A max(4, -3)= 4

1. **Time complexity of minimax algorithm**

The time complexity of the Minimax algorithm is O(b^d), where:

* b is the branching factor (the average number of possible moves at each state).
* d is the depth of the game tree (the maximum number of moves to reach a terminal state).

This exponential complexity arises because the algorithm explores all possible moves for both players, resulting in a significant number of states being evaluated, especially in complex games.

1. **Space complexity of minimax algorithm**

The space complexity of the Minimax algorithm is O(d), where d is the depth of the tree. This is due to the storage required for the recursive calls on the call stack. If iterative deepening is implemented, the space complexity can be reduced further.

1. **Applications of minimax algorithm**
2. **Board Games:** Widely used in games such as Chess, Tic-Tac-Toe, Checkers, and Othello for AI opponents.
3. **Game AI Development:** Creating intelligent game agents that can compete against human players.
4. **Decision-Making Systems:** In situations where multiple outcomes are possible and strategic choices are required, such as in economic modeling or resource management games.
5. **Advantages of minimax algorithm**
6. **Optimal Play:** Guarantees the best possible move if both players play optimally.
7. **Simple Implementation:** The concept is straightforward and easy to implement for games with a clear win/loss condition.
8. **Complete Knowledge:** Evaluates all possible future states, providing a comprehensive strategy.
9. **Limitations of minimax algorithm**
10. **Computational Complexity:** Becomes impractical for games with large game trees due to exponential growth (O(b^d)).
11. **Time Consumption:** Can take a long time to compute optimal moves in complex games without optimizations like alpha-beta pruning.
12. **Static Evaluation:** The algorithm relies on heuristic evaluation in non-terminal states, which may not always reflect the true game state.
13. **What is alpha-beta pruning?**

Alpha-beta pruning is an optimization technique for the Minimax algorithm, used in two-player games to reduce the number of nodes evaluated in the search tree. The main goal is to eliminate branches that will not affect the final decision, allowing the algorithm to run more efficiently without affecting the final outcome. This technique enables the algorithm to make decisions faster by avoiding unnecessary evaluations of certain game states.

1. **Use cases for minimax algorithm**
2. **Tic-Tac-Toe:** A classic example where the Minimax algorithm can easily determine the optimal move for either player.
3. **Chess Engines:** High-level chess engines use Minimax, often enhanced with alpha-beta pruning, to evaluate millions of positions quickly.
4. **Game Development:** Game developers use the Minimax algorithm to create challenging AI opponents in strategy and board games.

**Conclusion-**

The Minimax algorithm is a foundational technique in game theory and artificial intelligence, providing a systematic approach to decision-making in competitive environments. While effective in simple games, its limitations necessitate enhancements like alpha-beta pruning for more complex scenarios. Implementing the Minimax algorithm fosters a deeper understanding of strategic thinking in gaming and AI development.