**Assignment 5: Implement Minimax Algorithm for Game Playing**

**Problem Statement**

The goal of this assignment is to implement the Minimax algorithm to create an AI for a two-player game, such as Tic-Tac-Toe. This involves building a decision-making system that evaluates potential moves and selects the optimal one based on game theory.

**Objectives:**

* Understand the principles of the Minimax algorithm.
* Implement the Minimax algorithm for strategic gameplay in a two-player game.

**Theory**

**What is the Minimax Algorithm?**  
The Minimax algorithm is a recursive strategy used in two-player turn-based games. It is designed to minimize the possible loss in the worst-case scenario. The goal is to maximize the player's minimum gain (hence the name Minimax), assuming that the opponent also plays optimally.

**Methodology**

1. **Generate a Game Tree:**
   * A game tree is constructed where each node represents a game state, and edges represent possible moves. The root node corresponds to the current game state.
2. **Evaluate Terminal Nodes:**
   * Terminal nodes are game states where the game has ended (i.e., a win, loss, or draw). Each terminal node is assigned a score based on the outcome:
     + Win for the AI player: +1
     + Loss for the AI player (win for the opponent): -1
     + Draw: 0
3. **Backpropagate the Scores:**
   * Starting from the terminal nodes, the Minimax algorithm backpropagates the scores up the tree. The AI player selects the move that maximizes its score at each level, while the opponent tries to minimize the AI's score.

**Working Principle / Algorithm**

A simplified outline of the Minimax algorithm for a game like Tic-Tac-Toe:

1. **Define the Game State:**
   * Represent the current game board and the player to move.
2. **Generate Possible Moves:**
   * For the current game state, generate all legal moves (empty spots on the board).
3. **Recursively Evaluate Moves:**
   * For each possible move:
     + Make the move and switch players.
     + If the new game state is a terminal state (win, loss, or draw), return the score.
     + Otherwise, recursively call the Minimax function to evaluate the opponent's best response.
     + Undo the move to explore the next possible move.
4. **Choose the Optimal Move:**
   * Once all moves have been evaluated, choose the move with the highest score for the AI player (Maximizer). Conversely, the opponent (Minimizer) selects the move with the lowest score.

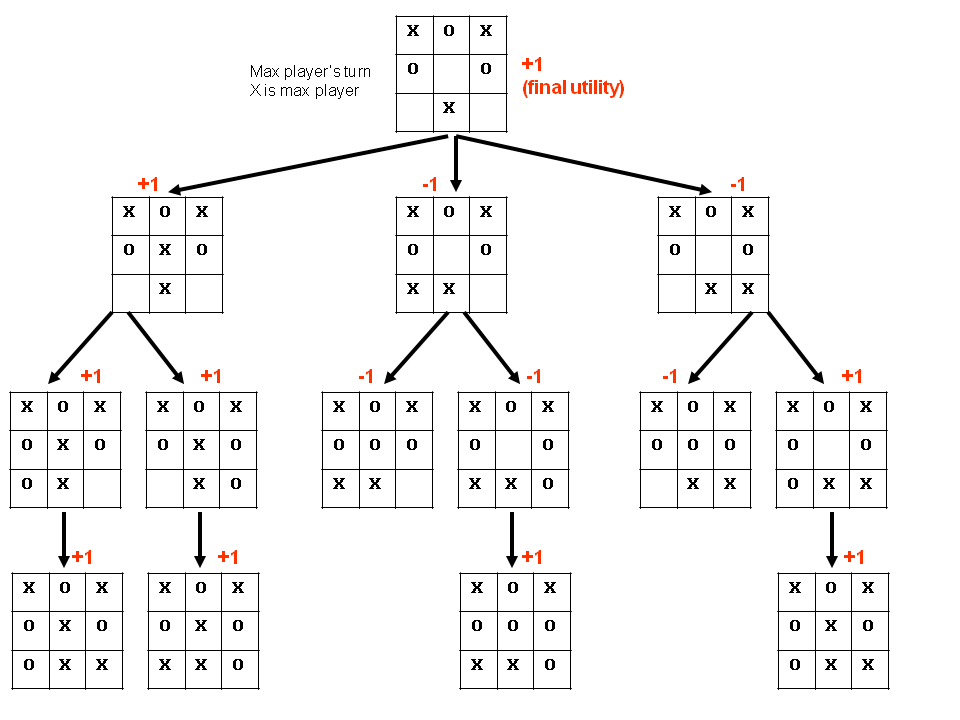
**Advantages**

* **Optimal Strategy:**  
  The Minimax algorithm ensures that the player makes the best possible move if both players play optimally.
* **Simple Implementation:**  
  It is straightforward to implement for smaller games like Tic-Tac-Toe, where the game tree is manageable.

**Disadvantages / Limitations**

* **Computationally Intensive:**  
  The algorithm can become slow for more complex games, as the number of possible game states increases exponentially.
* **Memory Usage:**  
  The game tree can become large, especially in games with bigger boards or longer move sequences, making it resource-intensive.

**Diagram**



**Conclusion**

The Minimax algorithm provides a strong foundation for developing game-playing AI, ensuring optimal moves in strategic games like Tic-Tac-Toe. By evaluating all possible outcomes and choosing the best move, the AI can effectively counter any strategy used by its opponent, making it a powerful approach to game AI design. However, as games become more complex, optimizations like alpha-beta pruning are often needed to reduce computational overhead.