**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**

* To implement the Minimax algorithm for a turn-based two-player game.
* To evaluate all possible moves and their outcomes by simulating the opponent's responses.
* To find the optimal strategy for both players by minimizing potential losses.
* To ensure efficient decision-making using depth-based exploration of the game tree.

**Theory**

**What is the Minimax Algorithm?**

The Minimax algorithm is a recursive strategy used in two-player turn-based games to minimize the possible loss for a worst-case scenario. When playing optimally, it maximizes the minimum gain (or minimizes the maximum loss) for the player.

**Methodology**

1. Define the Game States and Moves:

* Represent the game state (e.g., board configuration) and possible moves (e.g., placing a piece, changing a state).
* Identify the terminal states (win, lose, draw) and intermediate states (ongoing game).

1. Recursive Decision-Making with Minimax:

* For each possible move, simulate both players' turns by recursively evaluating the game's outcomes.
* Assign values to game states: +1 for a win, -1 for a loss, and 0 for a draw.

1. Optimize the Move Selection:

* If it's the maximizing player's turn, choose the move that leads to the highest possible score.
* If it's the minimizing player's turn, choose the move that minimizes the opponent's score.

1. Backtracking the Optimal Decision:

* Return the best move for the maximizing player and propagate the chosen value back up the game tree to make the final decision.

**Working Principle / Algorithm**

Here’s a simplified outline of the Minimax algorithm:

1. **Define the Game State**:
   * Represent the game board and the current player.
2. **Generate Possible Moves**:
   * For the current game state, generate all possible legal moves.
3. **Recursively Evaluate Moves**:
   * For each move:
     + Make the move and switch players.
     + If the new game state is a terminal state, return the score.
     + Otherwise, call the Minimax function recursively to evaluate the opponent's best response.
     + Undo the move to restore the game state for the next iteration.
4. **Choose the Optimal Move**:
   * Based on the scores returned from the recursive evaluations, choose the move with the highest score for the AI player.

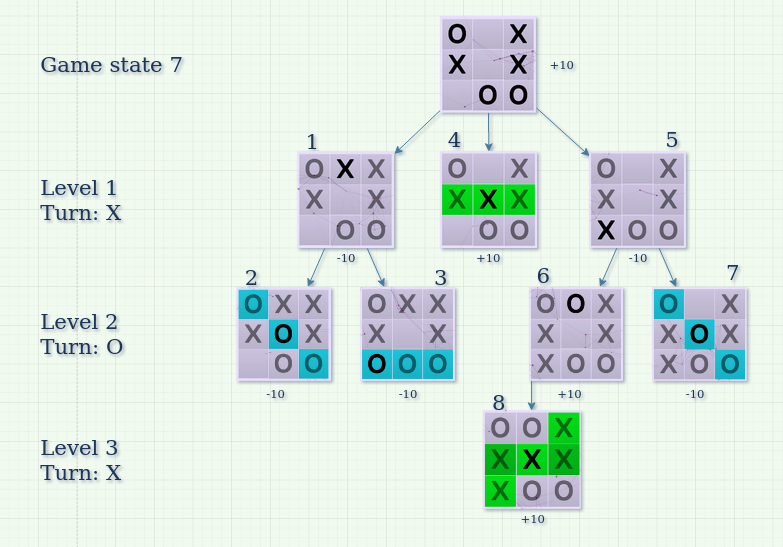
**Advantages**

* Optimal Play: Guarantees the best possible move for both players assuming perfect play.
* Game Analysis: Can be used to analyze all possible moves and strategies in small or finite games like Tic-Tac-Toe or chess.
* Simplicity: The recursive nature makes it conceptually straightforward to implement.

**Disadvantages / Limitations**

* Computational Complexity: For games with large state spaces, such as chess, the number of possible moves grows exponentially, making Minimax impractical without optimizations.
* Slow Execution: Without pruning techniques (e.g., Alpha-Beta pruning), Minimax can be slow for deep game trees due to evaluating every possible outcome.

**Diagram**



**Conclusion**

The Minimax algorithm provides a solid foundation for game-playing AI, ensuring optimal moves in strategic games. By systematically evaluating potential moves, the algorithm allows for the creation of intelligent opponents in two-player games.