Assignment No: - 5

Name :- Isha Ghorpade

Enrollnment :- 22420020

Roll :- 381066

Problem Statement

Implement the Minimax Algorithm for game playing applications such as Tic-Tac-Toe or Chess (simplified).

Objectives

- To understand the working of the Minimax search algorithm.
- To apply Minimax for two-player, turn-based, zero-sum games.
- To evaluate possible moves and choose the optimal strategy.
- To study decision-making in Artificial Intelligence using adversarial search.

Theory

Introduction

The Minimax algorithm is a backtracking-based decision-making algorithm used in game theory and AI. It is used to minimize the possible loss in a worst-case scenario. The algorithm is widely applied in two-player games, where one player tries to maximize the score (MAX player) and the other tries to minimize it (MIN player).

Minimax Principle

- MAX player → Tries to maximize the score.
- MIN player → Tries to minimize the score.
- Players alternate turns until a terminal state (win/loss/draw) is reached.
- The utility (evaluation) function assigns values:
  - +1 → MAX wins
  - -1 → MIN wins
  - 0 → Draw

Working of Minimax Algorithm

1. Generate Game Tree → Expand all possible moves up to terminal states or a defined depth.
2. Apply Utility Function → Assign values (+1, -1, 0) at terminal states.
3. Backpropagate Values → From leaf nodes to the root, using max for MAX's turn and min for MIN's turn.
4. Choose Optimal Move → At the root, MAX chooses the move with the maximum value.

Example: Tic-Tac-Toe

- Variables: Board states.
- Players: MAX (X) and MIN (O).
- Constraints: A move must be in an empty cell.
- Utility:
  - +1 if MAX wins.
  - -1 if MIN wins.
  - 0 for draw.

Methodology

1. Represent the game as a tree of states.
2. Apply Minimax recursively.
3. Define an evaluation function for non-terminal states.
4. Use backtracking to propagate optimal values upward.
5. Select the best move for MAX player at the root.

Advantages

- Provides optimal strategy for deterministic, two-player games.
- Works for small games like Tic-Tac-Toe, Nim, etc.
- Forms basis for advanced algorithms like Alpha-Beta pruning.

Limitations

- Computationally expensive for large games (e.g., Chess).
- Explores the entire game tree → exponential time complexity.
- Requires pruning/heuristics for efficiency.

Applications

- Board games (Tic-Tac-Toe, Chess, Checkers).
- Decision-making in adversarial environments.
- Robotics (competitive strategy formulation).
- AI-based opponents in video games.

Algorithm (Pseudocode)

```
function minimax(node, depth, isMaximizingPlayer):
    if node is a terminal state or depth == 0:
        return evaluation(node)

    if isMaximizingPlayer:
        best = -∞
        for each child of node:
            val = minimax(child, depth-1, False)
            best = max(best, val)
        return best
    else:
        best = +∞
        for each child of node:
            val = minimax(child, depth-1, True)
            best = min(best, val)
        return best
```

Conclusion

The Minimax Algorithm is a fundamental AI technique for adversarial game playing. It guarantees the best strategy assuming the opponent also plays optimally. Though limited by computational complexity, it provides the foundation for more advanced algorithms like Alpha-Beta pruning, which make it practical for large-scale games like Chess and Go.