**Name: Aditya Parade  
Roll No: 381047  
PRN: 22311577**

**Assignment 5**

**Problem Statement:** Implementation of Minimax Algorithm for Game Playing – Tic Tac Toe

**1. Introduction**

The **Minimax Algorithm** is a fundamental concept in **Artificial Intelligence (AI)** and **Game Theory**, primarily used in **two-player, turn-based games** such as Tic Tac Toe, Chess, and Checkers.  
It provides a systematic way for a computer to make **optimal decisions** by simulating all possible moves of both players and selecting the move that **minimizes the opponent’s advantage** while **maximizing its own**.

In this experiment, we apply the **Minimax Algorithm** to the game **Tic Tac Toe**, a simple 3×3 grid-based game where two players (X and O) alternately mark spaces. The goal is to get three marks in a row — horizontally, vertically, or diagonally.  
The algorithm enables the computer to play **intelligently** and **never lose**, ensuring either a win or a draw.

**2. Objective**

The main objectives of this experiment are:

* To understand and implement the **Minimax Algorithm** for decision-making in two-player games.
* To apply Minimax to the **Tic Tac Toe** game for creating an **AI opponent**.
* To simulate all possible game states and choose the **optimal move**.
* To understand the concept of **game trees**, **utility evaluation**, and **backtracking** in AI-based game playing.

**3. Theory**

**3.1 Game Playing in AI**

In Artificial Intelligence, a **game** is defined as a problem with:

* **Initial State:** The starting configuration (empty Tic Tac Toe board).
* **Players:** MAX (the computer) and MIN (the human opponent).
* **Actions:** Legal moves at each step.
* **Terminal State:** A state where the game ends (win, lose, or draw).
* **Utility Function:** The numerical outcome of the game:
  + +1 → MAX wins
  + -1 → MIN wins
  + 0 → Draw

The AI’s task is to select the move that leads to the **best possible outcome**, assuming the opponent also plays optimally.

**3.2 The Minimax Algorithm**

The **Minimax Algorithm** is a **backtracking** algorithm used to choose an optimal move for a player by minimizing the possible loss while maximizing the potential gain.

The game is represented as a **tree of possible moves**:

* The **MAX** player (AI) tries to maximize the score.
* The **MIN** player (human) tries to minimize the score.

At each level of the game tree, the algorithm recursively explores all possible future moves and assigns scores based on terminal outcomes (win, lose, draw).  
The AI then selects the move leading to the **maximum guaranteed utility**.

**3.3 Working Steps of the Algorithm**

1. **Generate the game tree** for all possible moves.
2. **Apply a scoring function** to evaluate terminal states (win, lose, draw).
3. **Propagate scores upward**:
   * MAX chooses the move with the **highest** score.
   * MIN chooses the move with the **lowest** score.
4. **Select the move** corresponding to the best score for MAX (AI).
5. Repeat this process for each turn until the game ends.

**3.4 Minimax Pseudocode**

function minimax(board, depth, isMaximizingPlayer):

if terminal\_state(board):

return evaluate(board)

if isMaximizingPlayer:

best = -∞

for each move in possible\_moves(board):

make move

value = minimax(board, depth + 1, false)

undo move

best = max(best, value)

return best

else:

best = +∞

for each move in possible\_moves(board):

make move

value = minimax(board, depth + 1, true)

undo move

best = min(best, value)

return best

**Evaluation Function (for Tic Tac Toe):**

* If AI wins → return **+10**
* If Human wins → return **-10**
* If Draw → return **0**

**3.5 Example – Tic Tac Toe Scenario**

Consider the following partially filled board:

X | O | X

O | X | \_

\_ | \_ | O

The AI (X) must decide its next move.  
Using Minimax, the AI simulates all possible moves, checks potential wins or losses, and finally selects the move that maximizes its chances of winning (or forces a draw if winning isn’t possible).

**3.6 Optimization – Alpha-Beta Pruning**

In large game trees, exploring all nodes is computationally expensive.  
**Alpha-Beta Pruning** is used to **cut off unnecessary branches** that won’t affect the final decision, improving efficiency without changing the result.

* **Alpha (α)** – The best value that MAX can guarantee.
* **Beta (β)** – The best value that MIN can guarantee.

Pruning is done when β ≤ α, meaning that further exploration of the branch is useless.

**3.7 Applications of Minimax**

* **Board Games:** Tic Tac Toe, Chess, Checkers, Connect 4.
* **Strategic AI Decision Making:** Used in adversarial planning and two-agent systems.
* **Game Development:** To create intelligent, unbeatable opponents.

**4. Conclusion**

In this experiment, we successfully implemented the **Minimax Algorithm** for the **Tic Tac Toe** game.  
The AI analyzes all possible moves and counter-moves to select the **optimal strategy**, ensuring it **never loses** when playing perfectly.

Key learnings include:

* Understanding the concept of **game trees** and **adversarial search**.
* Learning how **recursive evaluation** determines the best move.
* Observing how **Alpha-Beta Pruning** enhances Minimax efficiency.

This implementation demonstrates how intelligent decision-making can be achieved through **logical reasoning and search algorithms** — forming the foundation of many AI-based games and strategic systems.