ASSIGNMENT NO: 9

	Marathwada Mitra Mandal's	
	Institute of Technology, Lohgaon Pune - 47	
	Department of Artificial Intelligence and Data Science	

Semester -I A.Y.2025-26 Sub.: - Artificial Intelligence Lab Class: SE

Aim : Understand and implement the basic Minimax algorithm for two-player deterministic, zero-sum games and apply it to a simple game (Tic-Tac-Toe). Evaluate the algorithm's behavior and discuss limitations and improvements.

2. Learning Objectives :

By the end of the lab the student should be able to:

● Explain the minimax decision rule and game trees.

● Implement minimax using recursion to choose optimal moves for perfect-play agents. ● Apply minimax to Tic-Tac-Toe and verify correct play.

● Analyze complexity and discuss pruning (alpha-beta) and depth-limiting.

3. Background / Theory

Two-player, deterministic games with perfect information (e.g., Tic-Tac-Toe, Chess at a conceptual level) can be modeled as a game tree. Each node represents a game state and edges represent legal moves. Players alternate turns; one is called MAX (tries to maximize utility)

and the other MIN (tries to minimize utility). In a zero-sum game, one player's gain is the other's loss.

Minimax idea: Starting from the current state, explore possible moves to terminal states and evaluate each terminal state with a utility function (win = +1, draw = 0, loss = -1 for MAX). Propagate utilities upward: at MAX nodes choose the child with maximum utility; at MIN nodes choose the child with minimum utility. The root decision yields the best move assuming perfect play by both.

Complexity: Time complexity is O(b^d) where b = branching factor, d = search depth. Tic-Tac-Toe is small enough to be solved fully. Larger games require depth-limiting and heuristics.

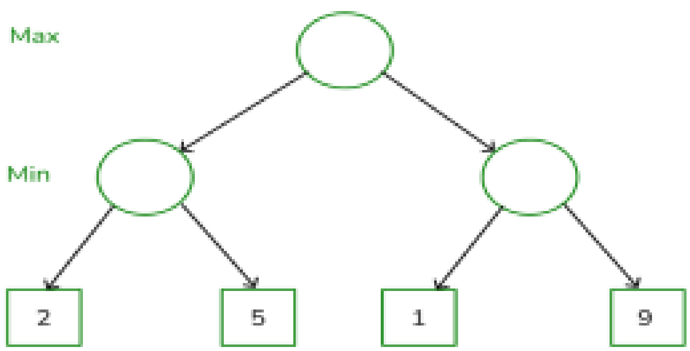
4. Algorithm

```
minimax (node n, depth d, player p)
 1. If depth = 0 then
        return value(node)
2. If player = "MAX"
        set α = - ∞
        for every child of node
                value = minimax (child,depth-1,'MIN')
                α = max(α, value)
        return (α )
else
        set α = +∞
        for every child of node
                value = minimax (child,depth-1,'MAX')
```

```
        α = min(α, value)
    return (α )
```

5. Python Implementation

Example graph



6. Sample Output

7. Observations :Optimal Play Assumption:

- Minimax assumes that both players play optimally. MAX tries to maximize the score, and MIN tries to minimize the score.
- Observation: The algorithm guarantees the best possible outcome if both play perfectly.

Exponential Growth of Tree:

- For a game tree with branching factor bbb and depth ddd, the number of nodes is O(bd)O(b^d)O(bd).
- Observation: The algorithm becomes computationally expensive for large games like Chess.

Deterministic and Perfect Information:

- Works only for games with no chance element and complete knowledge (like Tic-Tac-Toe, Chess).
- Observation: Cannot handle games with randomness or hidden information efficiently.

Decision at Root:

- The algorithm evaluates all possible future moves and assigns a value to the root node representing the best achievable outcome.
- Observation: MAX always chooses the move leading to the highest minimax value.

8. Conclusion: The Minimax algorithm is a fundamental

decision-making strategy in Artificial Intelligence, particularly in two-player, turn-based games like Chess, Tic-Tac-Toe, and Checkers. It ensures that a player makes the best possible move by assuming that the opponent also plays optimally.

```
def minimax(node, depth, maximizingPlayer, values, index=0)

 # Leaf node condition

 if depth == 0 or index >= len(values):

        return values[index]




    if maximizingPlayer:

        best = float('-inf')
```

```
    for i in range(2):  # Two children for each node
        val = minimax(node*2+i, depth-1, False, values, index*2+i)
        best = max(best, val)
    return best
else:
    best = float('inf')
    for i in range(2):
        val = minimax(node*2+i, depth-1, True, values, index*2+i)
        best = min(best, val)
```

```
        return best

# Example: Game tree with depth = 3

values = [3, 5, 6, 9, 1, 2, 0, -1]  # Leaf node values

depth = 3

result = minimax(0, depth, True, values)

print("Optimal value (using Minimax):", result
```

OUTPUT :optimal value (using minmax):5