

|  |  |
| --- | --- |
| **Student Name** | **Rugved Myakal** |
| **SRN No** | 31230051 |
| **Roll No** | 23 |
| **Program** | **Department of Computer Engineering** |
| **Year** | 2024-2025 |
| **Division** | G |
| **Subject** | Artificial Intelligence lab |
| **Assignment No** | 7 |



**Experiment Number - 07**

**Title /Problem Statement: Implement the Minimax Algorithm to solve the Tic Tac Toe problem.**

# Description: The minimax algorithm is a backtracking algorithm used in decision-making and game theory to find the optimal move for a player, assuming that the opponent also plays optimally. In Tic Tac Toe, minimax can be used to predict the opponent's best responses and choose moves that maximize the chances of winning or drawing the game.

**Theory: The minimax algorithm works by simulating all possible moves for both players and choosing the move that maximizes the player's minimum gain (hence the name "minimax").**

* **Maximizing Player (AI): Tries to maximize the score.**
* **Minimizing Player (Human/Opponent): Tries to minimize the score.**

**Each board state is evaluated:**

* **Win = +1**
* **Loss = -1**
* **Draw = 0**

**The AI recursively explores all possible game moves, assuming that the opponent will also make the best possible moves, and selects the move that ensures the best achievable outcome. In Tic Tac Toe, since the game space is small, minimax can explore the entire tree of possibilities to guarantee a win or at least a draw.**

Part A:

**import math**

**def print\_board(board):**

**for row in board:**

**print("|".join(row))**

**print("-" \* 5)**

**def check\_winner(board):**

**# Rows, columns and diagonals**

**for i in range(3):**

**if board[i][0] == board[i][1] == board[i][2] != ' ':**

**return board[i][0]**

**if board[0][i] == board[1][i] == board[2][i] != ' ':**

**return board[0][i]**

**if board[0][0] == board[1][1] == board[2][2] != ' ':**

**return board[0][0]**

**if board[0][2] == board[1][1] == board[2][0] != ' ':**

**return board[0][2]**

**return None**

**def is\_draw(board):**

**for row in board:**

**if ' ' in row:**

**return False**

**return True**

**def minimax(board, depth, is\_maximizing):**

**winner = check\_winner(board)**

**if winner == 'O':**

**return 1**

**elif winner == 'X':**

**return -1**

**elif is\_draw(board):**

**return 0**

**if is\_maximizing:**

**best\_score = -math.inf**

**for i in range(3):**

**for j in range(3):**

**if board[i][j] == ' ':**

**board[i][j] = 'O'**

**score = minimax(board, depth + 1, False)**

**board[i][j] = ' '**

**best\_score = max(score, best\_score)**

**return best\_score**

**else:**

**best\_score = math.inf**

**for i in range(3):**

**for j in range(3):**

**if board[i][j] == ' ':**

**board[i][j] = 'X'**

**score = minimax(board, depth + 1, True)**

**board[i][j] = ' '**

**best\_score = min(score, best\_score)**

**return best\_score**

**def best\_move(board):**

**best\_score = -math.inf**

**move = None**

**for i in range(3):**

**for j in range(3):**

**if board[i][j] == ' ':**

**board[i][j] = 'O'**

**score = minimax(board, 0, False)**

**board[i][j] = ' '**

**if score > best\_score:**

**best\_score = score**

**move = (i, j)**

**return move**

**def tic\_tac\_toe():**

**board = [[' ' for \_ in range(3)] for \_ in range(3)]**

**print("Tic Tac Toe: You are X, AI is O")**

**print\_board(board)**

**while True:**

**# Human move**

**row, col = map(int, input("Enter your move (row and column 0-2): ").split())**

**if board[row][col] != ' ':**

**print("Invalid move. Try again.")**

**continue**

**board[row][col] = 'X'**

**if check\_winner(board) == 'X':**

**print\_board(board)**

**print("You win!")**

**break**

**if is\_draw(board):**

**print\_board(board)**

**print("It's a draw!")**

**break**

**# AI move**

**ai\_move = best\_move(board)**

**if ai\_move:**

**board[ai\_move[0]][ai\_move[1]] = 'O'**

**print(f"AI chose position {ai\_move}")**

**print\_board(board)**

**if check\_winner(board) == 'O':**

**print\_board(board)**

**print("AI wins!")**

**break**

**if is\_draw(board):**

**print\_board(board)**

**print("It's a draw!")**

**break**

**tic\_tac\_toe()**

**Output Screenshot:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Conclusion:**

The Minimax algorithm provides an optimal strategy for playing Tic Tac Toe, ensuring that the AI either wins or forces a draw against any human opponent. By recursively exploring all possible game states and assuming that both players play perfectly, the AI can always predict the best possible move at each turn. This experiment demonstrates how decision trees, recursion, and game theory principles can be combined to create intelligent, unbeatable AI players for small-scale deterministic games. The implementation also highlights the importance of evaluation functions and efficient search techniques for solving two-player zero-sum games.