**Tic-Tac-Toe Solver Report**

**Name:** Akash Kr Gupta  
**Roll No.:** 202401100300025  
**Date:** 11/03/2025  
**Branch:** CSEAI  
**Sec:** A

**Problem Statement**

Design and implement an AI-based Tic-Tac-Toe Solver that determines the best possible move for a player using the Minimax algorithm. The objective is to ensure optimal gameplay by preventing the opponent from winning and maximizing the chances of victory.

**1. Introduction**

Tic-Tac-Toe is a two-player game played on a 3x3 grid. The goal is to place three of one’s marks in a row (horizontally, vertically, or diagonally) before the opponent does. This report presents a Tic-Tac-Toe Solver using the Minimax algorithm, which allows an AI to play optimally against a human or another AI.

The Minimax algorithm evaluates all possible game states and chooses the move that minimizes the opponent’s chances of winning while maximizing the AI’s chances. This makes the solver unbeatable if implemented correctly.

**2. Methodology**

The Tic-Tac-Toe Solver is implemented using the following steps:

1. **Game Representation**: The board is represented as a 3x3 matrix with three states for each cell: empty (''), 'X', or 'O'.
2. **Move Evaluation**: The AI evaluates all possible moves and selects the best one using the Minimax algorithm.
3. **Minimax Algorithm**:
   * Recursively explores all possible moves.
   * Assigns a score to each move based on potential outcomes.
   * Chooses the move that leads to the best possible result.
4. **Game Implementation**: The program allows a human player to compete against the AI, which responds optimally.
5. **Winning Conditions**: The solver checks for a win, a draw, or an ongoing game after every move.

**3. Code**

import math

# Function to print the Tic-Tac-Toe board

def print\_board(board):

for row in board:

print(" | ".join(row))

print("\n")

# Function to check if a player has won

def check\_winner(board, player):

for row in board:

if all(cell == player for cell in row):

return True

for col in range(3):

if all(board[row][col] == player for row in range(3)):

return True

if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):

return True

return False

# Function to check if the board is full

def is\_full(board):

return all(cell != '' for row in board for cell in row)

# Minimax algorithm for finding the best move

def minimax(board, depth, is\_max):

if check\_winner(board, 'O'):

return 1 # AI wins

if check\_winner(board, 'X'):

return -1 # Human wins

if is\_full(board):

return 0 # Draw

if is\_max:

best = -math.inf

for i in range(3):

for j in range(3):

if board[i][j] == '': # Check if cell is empty

board[i][j] = 'O'

best = max(best, minimax(board, depth + 1, False))

board[i][j] = '' # Undo move

return best

else:

best = math.inf

for i in range(3):

for j in range(3):

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

board[i][j] = 'X'

best = min(best, minimax(board, depth + 1, True))

board[i][j] = '' # Undo move

return best

# Function to find the best move for AI

def find\_best\_move(board):

best\_val = -math.inf

best\_move = (-1, -1)

for i in range(3):

for j in range(3):

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

board[i][j] = 'O'

move\_val = minimax(board, 0, False)

board[i][j] = '' # Undo move

if move\_val > best\_val:

best\_move = (i, j)

best\_val = move\_val

return best\_move

# Function to play the game

def play\_game():

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

while True:

print\_board(board)

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

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

board[row][col] = 'X' # Human move

else:

print("Invalid move, try again!")

continue

if check\_winner(board, 'X'):

print\_board(board)

print("Player X wins!")

break

if is\_full(board):

print\_board(board)

print("It's a draw!")

break

ai\_move = find\_best\_move(board)

board[ai\_move[0]][ai\_move[1]] = 'O' # AI move

if check\_winner(board, 'O'):

print\_board(board)

print("AI (O) wins!")

break

if is\_full(board):

print\_board(board)

print("It's a draw!")

break

play\_game()

**4. Output/Results**

**Key Findings:**

1. **Minimax Algorithm Efficiency:** The AI successfully predicts and makes optimal moves, preventing losses and maximizing its chances of winning.
2. **Game Playability:** The solver allows a human player to compete against an AI, making it interactive.
3. **Winning and Draw Detection:** The AI correctly identifies winning conditions and draws, ensuring fair gameplay.

**5. References/Credits**

1. Algorithm implemented using the Minimax strategy.
2. Libraries used: Python built-in functions.
3. Code executed in a local Python environment.