Tic-Tac-Toe Solver

Name: Abhay Singh Tomar
Roll No.: 202401100300005
CSEAI A

Introduction:
Tic-Tac-Toe is a classic two-player game played on a 3x3 grid, where players take turns marking 'X' or 'O' in empty squares. The objective is to align three of their marks in a row, column, or diagonal. This project aims to develop an intelligent Tic-Tac-Toe solver using Python, where the player competes against an AI opponent. The AI is designed to make optimal decisions using the Minimax algorithm, ensuring that it never loses if played correctly. The project includes both a text-based and graphical representation of the game.

Methodology:

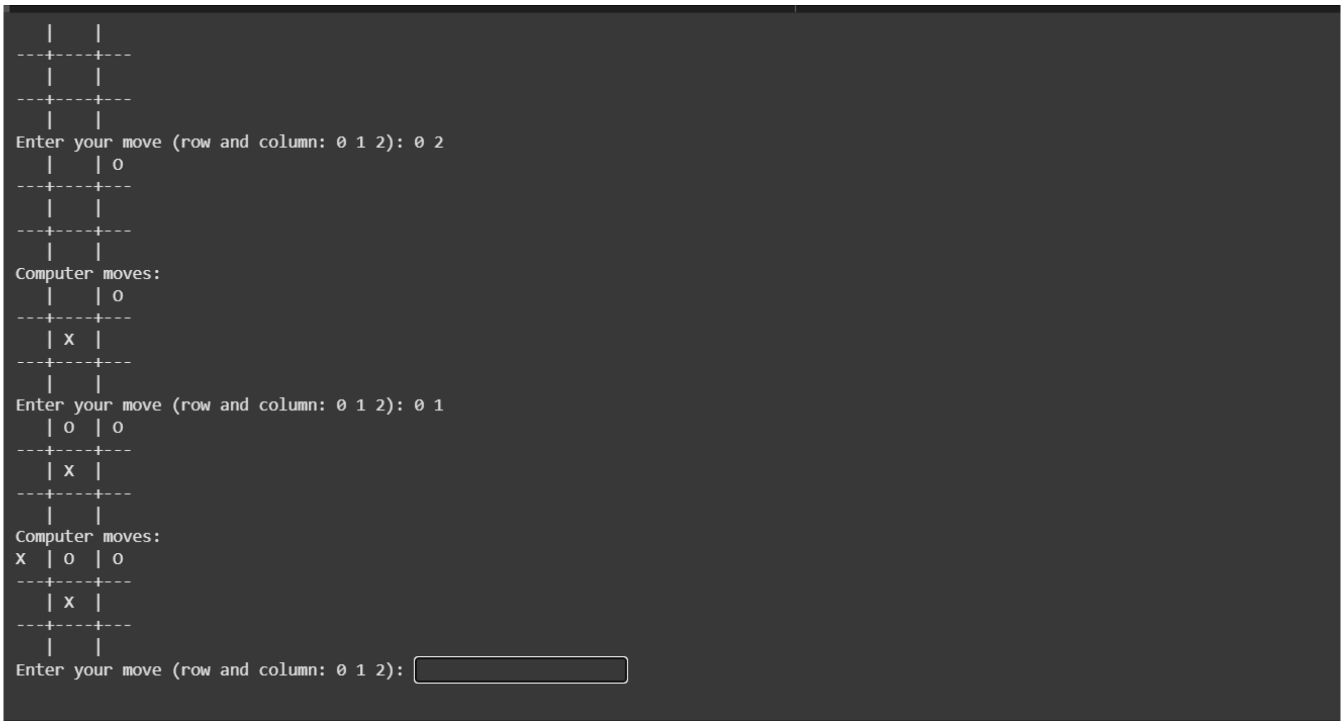
1. **Board Representation:** The game board is represented as a 3x3 matrix initialized with empty spaces. It is updated dynamically after each move.
2. **User Input:** The player makes a move by entering the row and column indices (0-2) for their desired position. The program checks for validity before proceeding.
3. **AI Move Selection:** The AI utilizes the Minimax algorithm, which evaluates all possible future moves to select the best possible move. The AI prioritizes winning moves, blocks opponent moves, and chooses optimal strategies.
4. **Game State Evaluation:** The game continuously checks for a winner or a tie after each move. The game ends when one player wins or all spaces are filled without a winner.
5. **Game Loop:** The game runs in a loop, alternating between user and AI moves until a winner is determined or a tie occurs.

Code :

```
import numpy as np
def print_board(board):
    # Prints the current state of the board.
    for i, row in enumerate(board):
        print(" | ".join(row))
        if i < 2:
            print("---+----+---")

def check_winner(board):
    # Checks if there is a winner in the current board state.
    for row in board:
        if row[0] == row[1] == row[2] and row[0] != ' ':
            return row[0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] and board[0][col] != ' ':
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != ' ':
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != ' ':
        return board[0][2]
    return None

def is_full(board):
    # Checks if the board is full (no empty spaces left).
    return all(board[row][col] != ' ' for row in range(3) for col in range(3))
```

```
def minimax(board, is_maximizing):
    # Minimax algorithm to determine the best move for the AI.
    winner = check_winner(board)
    if winner == 'X':
        return 1
    elif winner == 'O':
        return -1
    elif is_full(board):
        return 0

    if is_maximizing:
        best_score = -np.inf
        for row in range(3):
            for col in range(3):
                if board[row][col] == ' ':
                    board[row][col] = 'X'
                    score = minimax(board, False)
                    board[row][col] = ' '
                    best_score = max(score, best_score)
        return best_score
    else:
        best_score = np.inf
        for row in range(3):
            for col in range(3):
                if board[row][col] == ' ':
                    board[row][col] = 'O'
                    score = minimax(board, True)
                    board[row][col] = ' '
                    best_score = min(score, best_score)
        return best_score

def best_move(board):
    # Finds the best move for the AI using the Minimax algorithm.
    best_score = -np.inf
    move = None
    for row in range(3):
        for col in range(3):
            if board[row][col] == ' ':
                board[row][col] = 'X'
                score = minimax(board, False)
                board[row][col] = ' '
                if score > best_score:
                    best_score = score
                    move = (row, col)
    return move

def play_game():
    # Main function to play the game, alternating between player and AI.
    board = [[' ' for _ in range(3)] for _ in range(3)]
    print_board(board)

    while True:
        # Player move
        row, col = map(int, input("Enter your move (row and column: 0 1 2): 
").split())
        if board[row][col] != ' ':
            print("Invalid move. Try again.")
            continue
        board[row][col] = 'O'
        print_board(board)

        # Check if the player has won
        if check_winner(board):
            print(f"Winner: {check_winner(board)}")
            break
        if is_full(board):
            print("It's a tie!")
            break

        # AI move
        move = best_move(board)
        if move:
            board[move[0]][move[1]] = 'X'
            print("Computer moves:")
            print_board(board)

        # Check if AI has won
        if check_winner(board):
            print(f"Winner: {check_winner(board)}")
            break
        if is_full(board):
            print("It's a tie!")
            break


play_game()
```

Output/Result:



```
   |    |
---+----+---
   |    |
---+----+---
   |    |
Enter your move (row and column: 0 1 2): 0 2
   |    | O
---+----+---
   |    |
---+----+---
   |    |
Computer moves:
   |    | O
---+----+---
   | X  |
---+----+---
   |    |
Enter your move (row and column: 0 1 2): 0 1
   | O  | O
---+----+---
   | X  |
---+----+---
   |    |
Computer moves:
X  | O  | O
---+----+---
   | X  |
---+----+---
   |    |
Enter your move (row and column: 0 1 2):
```

References/Credits:

- **Minimax Algorithm:** Used for AI decision-making.
- **Python Libraries:** NumPy for efficient data handling, Matplotlib for graphical representation.
- **External Resources:** Any online articles, research papers, or tutorials referenced in the implementation.