Tic-Tac-Toe Solver in Python

Problem Statement

Tic-Tac-Toe is a classic two-player game where players take turns marking spaces in a 3x3 grid. The objective of this project is to develop an AI-based Tic-Tac-Toe solver using the Minimax algorithm. The AI should always make optimal moves, allowing a human player to compete against it. The game will determine the winner, draw, or continuation based on board states.

Student Details:

- **Name:** Satyam Kumar
- **Roll No.:** 202401100400167
- **Branch:** CSE(AIML)
- **Institution:** KIET Group of Institutions

Introduction

Tic-Tac-Toe is a simple yet strategic game that has been played for centuries in various forms. It is commonly used as an introductory problem in artificial intelligence and game theory because of its small state space and clear winning conditions. The goal of this project is to implement an AI-driven Tic-Tac-Toe solver using Python and the Minimax algorithm. The Minimax algorithm allows the AI to analyze all possible board states and select the optimal move to maximize its chances of winning while minimizing the opponent's chances.

This project is significant because it provides insights into decision-making algorithms, recursion, and optimization techniques. The application of Minimax in Tic-Tac-Toe is a fundamental example of AI-driven gameplay, demonstrating how machines can make rational decisions based on available data. Additionally, the project strengthens problem-solving skills by requiring efficient handling of board states, checking win conditions, and designing an interactive user experience.

By implementing this project, we aim to:

- Create a fully functional Tic-Tac-Toe game that allows human vs. AI gameplay.
- Ensure the AI plays optimally using the Minimax algorithm.
- Provide an interactive experience where users can input moves and receive responses from the AI.
- Gain a deeper understanding of algorithmic game strategies and their real-world applications.

Methodology

1. Board Representation

- The game board is represented as a 3x3 NumPy array.
- Empty spaces are initialized with " ".

2. Game Logic

- A function checks for winning conditions (rows, columns, diagonals).
- The board is updated based on the player's input.

3. Minimax Algorithm

- The AI evaluates every possible move using **Minimax**, which assigns scores to board states:
  - AI win: **+1**
  - Human win: **-1**
  - Draw: **0**
- The AI recursively searches for the best move by maximizing its score while minimizing the human's score.

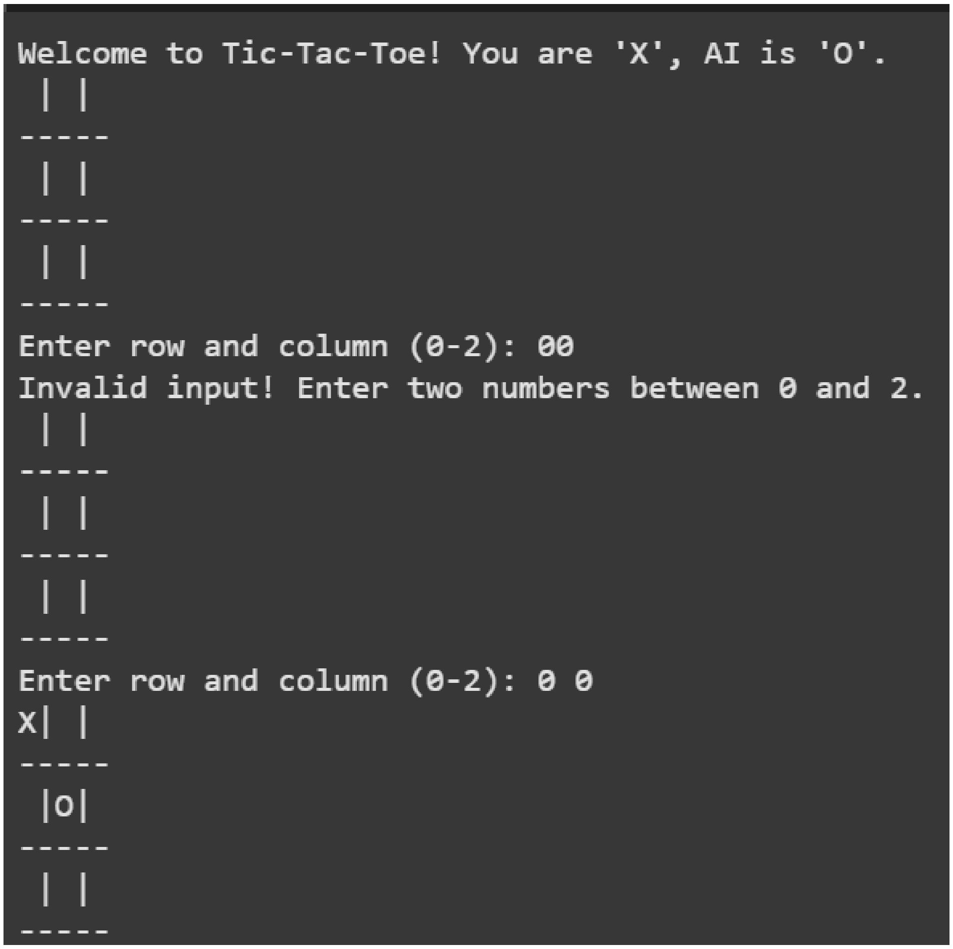
4. User Interaction

- The human player selects row and column indices (0-2) to make a move.
- The AI calculates and makes its best move.
- The game ends when a player wins or the board is full (draw).

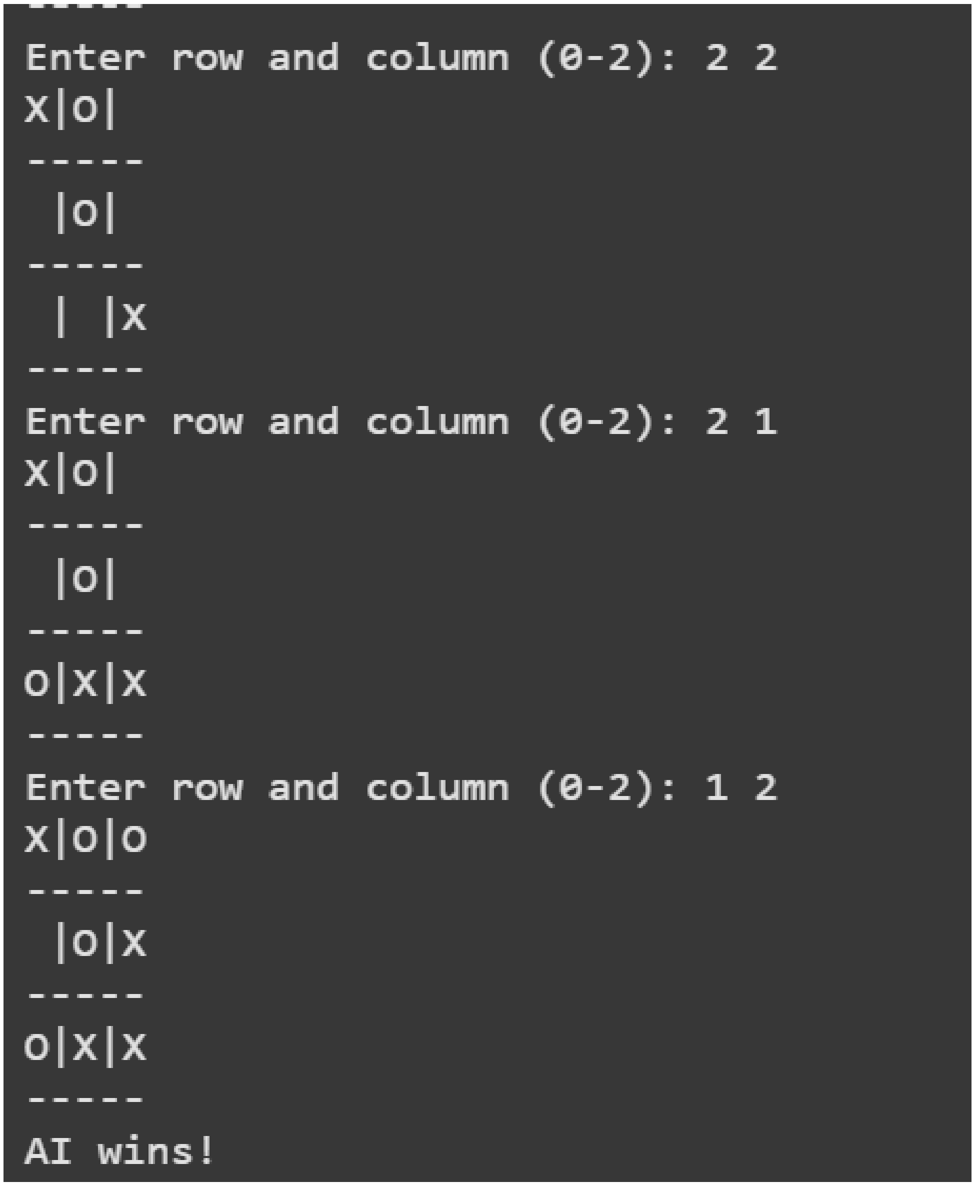
CODE

```python
import numpy as np

# Tic-Tac-Toe game class
class TicTacToe:
    def __init__(self):
        """Initialize a 3x3 Tic-Tac-Toe board and define player markers."""
        self.board = np.full((3, 3), " ")  # Create a 3x3 board filled with spaces
        self.human = "X"  # Human player marker
        self.ai = "O"  # AI player marker

    def print_board(self):
        """Prints the Tic-Tac-Toe board with a visual separator."""
        for row in self.board:
            print("|".join(row))  # Print row elements separated by '|'
            print("-" * 5)  # Print horizontal separator between rows

    def is_winner(self, player):
        """Checks if a given player has won the game by forming a line."""
        # Check all rows and columns for a winning line
        for i in range(3):
            if all(self.board[i, j] == player for j in range(3)) or \
               all(self.board[j, i] == player for j in range(3)):
                return True
        # Check both diagonals for a winning line
        return all(self.board[i, i] == player for i in range(3)) or \
               all(self.board[i, 2 - i] == player for i in range(3))

    def is_full(self):
        """Returns True if the board is full and no more moves are possible."""
        return not any(self.board[i, j] == " " for i in range(3) for j in range(3))

    def get_empty_positions(self):
        """Returns a list of available empty positions (row, col) on the
board."""
        return [(i, j) for i in range(3) for j in range(3) if
self.board[i, j] == " "]

    def minimax(self, is_maximizing):
        """Implements the Minimax algorithm to determine the best move for
AI."""
        # Evaluate the current board state
        if self.is_winner(self.human):
            return -1  # Human wins, AI loses
        if self.is_winner(self.ai):
            return 1  # AI wins
        if self.is_full():
            return 0  # It's a draw

        if is_maximizing:
            best_score = -float("inf")
            # Iterate through all empty positions and simulate AI move
            for i, j in self.get_empty_positions():
                self.board[i, j] = self.ai
                score = self.minimax(False)  # Call minimax for opponent's
turn
                self.board[i, j] = " "  # Undo move
                best_score = max(score, best_score)  # Maximize AI's score
            return best_score
        else:
            best_score = float("inf")
            # Iterate through all empty positions and simulate human move
            for i, j in self.get_empty_positions():
                self.board[i, j] = self.human
                score = self.minimax(True)  # Call minimax for AI's turn
                self.board[i, j] = " "  # Undo move
                best_score = min(score, best_score)  # Minimize AI's loss
            return best_score

    def best_move(self):
        """Determines the best move for AI using the Minimax algorithm."""
        best_score = -float("inf")
        move = None
        # Iterate through all empty positions to find the optimal move
        for i, j in self.get_empty_positions():
            self.board[i, j] = self.ai  # Simulate AI move
            score = self.minimax(False)  # Evaluate move using minimax
            self.board[i, j] = " "  # Undo move
            if score > best_score:
                best_score = score  # Update best score
                move = (i, j)  # Store the best move
        return move

    def play(self):
        """Main game loop for playing Tic-Tac-Toe."""
        print("Welcome to Tic-Tac-Toe! You are 'X', AI is 'O'.")
        while not self.is_full():  # Continue until the board is full
            self.print_board()  # Display current board state
            try:
                row, col = map(int, input("Enter row and column (0-2):
").split())
                # Validate the move
                if self.board[row, col] == " ":
                    self.board[row, col] = self.human  # Place human's
move
                else:
                    print("Invalid move, try again.")
                    continue
            except (ValueError, IndexError):
                print("Invalid input! Enter two numbers between 0 and 2.")
                continue

            # Check if human wins
            if self.is_winner(self.human):
                self.print_board()
                print("You win!")
                return

            # AI makes a move if the board is not full
            if not self.is_full():
                ai_move = self.best_move()
                self.board[ai_move] = self.ai  # AI places its move
                # Check if AI wins
                if self.is_winner(self.ai):
                    self.print_board()
                    print("AI wins!")
                    return

        self.print_board()
        print("It's a draw!")  # Game ends in a draw if the board is full

# Run the game
if __name__ == "__main__":
    game = TicTacToe()
    game.play()
```

Output



```
Welcome to Tic-Tac-Toe! You are 'X', AI is 'O'.
 | |
-----
 | |
-----
 | |
-----
Enter row and column (0-2): 00
Invalid input! Enter two numbers between 0 and 2.
 | |
-----
 | |
-----
 | |
-----
Enter row and column (0-2): 0 0
X| |
-----
 |O|
-----
 | |
-----
```



```
-----
Enter row and column (0-2): 2 2
x|o|
-----
 |o|
-----
 | |x
-----
Enter row and column (0-2): 2 1
x|o|
-----
 |o|
-----
o|x|x
-----
Enter row and column (0-2): 1 2
x|o|o
-----
 |o|x
-----
o|x|x
-----
AI wins!
```

References/Credits

- Minimax Algorithm: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
- NumPy Documentation for array operations.
- Python Official Documentation for basic I/O and loop structures.