REPORT FILE

Artificial Intelligence MSE-1

Problem Statement

Tic-Tac-Toe Solver

Name :- *Preeti Tiwari*

Branch :- *CSE(AI&ML)*

Section :- *C*

University Roll No. :- *202401100400147*

Introduction

Problem Statement:- Tic-Tac- Toe solver

The ***Tic-Tac-Toe Solver*** *is a Python-based implementation of the classic Tic-Tac-Toe game, where an AI opponent plays against a human user. The AI is designed using the* ***Minimax Algorithm****, a decision-making technique used in two-player games to find the optimal move. The game is played on a 3×3 grid, where the player (X) and the AI (O) take turns placing their marks. The AI evaluates all possible moves, assigns scores based on potential outcomes (win, lose, or draw), and chooses the move that maximizes its chances of winning while minimizing the opponent's chances. This ensures that the AI never loses if it plays optimally. The program continuously updates the board, checks for a winner or a draw, and displays the game state after every move. The Tic-Tac-Toe Solver demonstrates the power of game theory and recursion in artificial intelligence, providing an unbeatable AI opponent.*

Methodology Used

The approach used to solve the ***Tic-Tac-Toe Solver*** *is based on the* ***Minimax Algorithm****, which is a recursive decision-making technique commonly used in turn-based games. The game begins with an empty 3×3 board, where the user (X) and the AI (O) take turns making moves. To determine the best possible move, the AI simulates all potential future moves using the Minimax algorithm. It evaluates each move by assigning a score:* ***+1 for an AI win, -1 for a player win, and 0 for a draw****. The AI then chooses the move that maximizes its chance of winning while minimizing the player's chance of victory. This ensures that the AI plays optimally and never makes a losing move. If the board is already in a winning or draw state, the algorithm stops further calculations.*

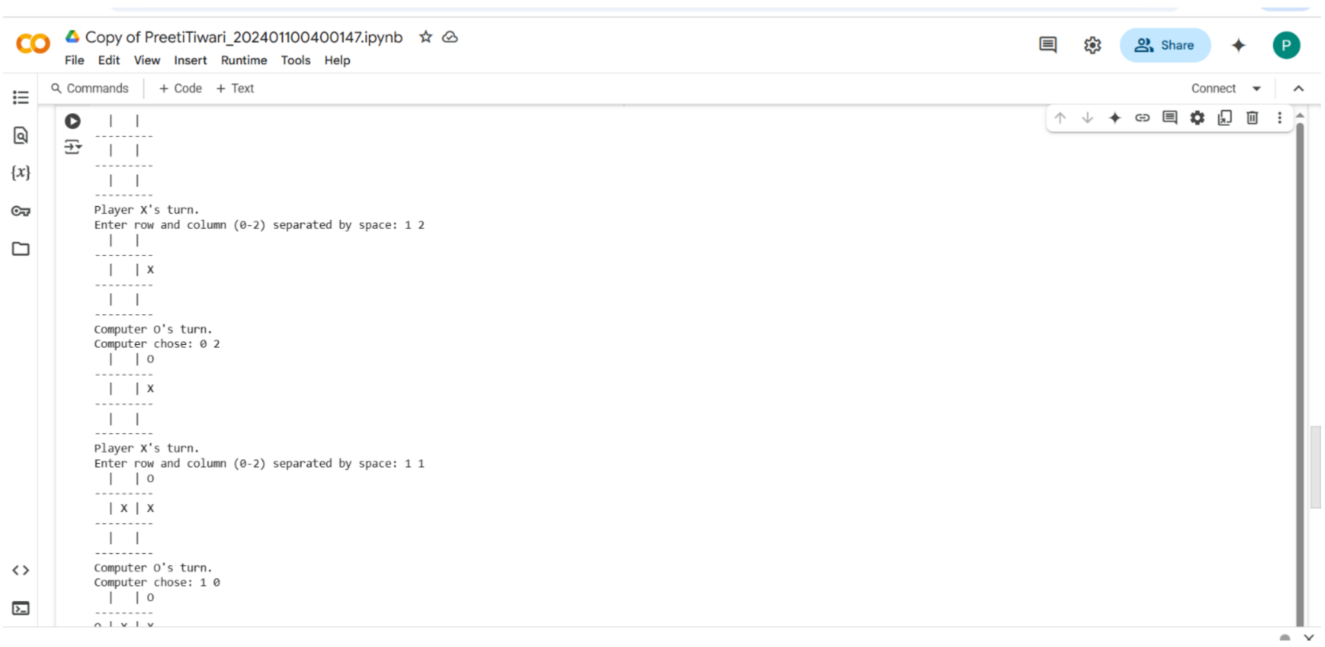
The **Minimax function** *works by recursively simulating all possible board states. If it's the AI's turn, the algorithm tries to maximize the score by selecting the move with the highest value. If it's the player's turn, the algorithm assumes the player will also play optimally and tries to minimize the score. The recursion continues until a terminal state (win, loss, or draw) is reached. The AI then backtracks, choosing the move that leads to the best possible outcome. To improve efficiency, additional techniques like* **Alpha-Beta Pruning** *can be used to eliminate unnecessary calculations, but in a simple 3×3 Tic-Tac-Toe game, Minimax alone is sufficient for optimal gameplay. This approach ensures that the AI plays perfectly and will never lose unless the opponent also plays flawlessly.*

Code

```
import math
# Function to print the current state of the board
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)
# Function to check if there's a winner
def check_winner(board):
    # Check rows and columns
    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]
    # Check diagonals
    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
# Function to check if the game is a draw
def is_draw(board):
    for row in board:
        if ' ' in row:
            return False
    return True
# Minimax algorithm to find the best move for the computer
def minimax(board, depth, is_maximizing):
    winner = check_winner(board)
    if winner == 'X':  # Player wins
        return -1
    if winner == 'O':  # Computer wins
        return 1
    if is_draw(board):  # Game is a draw
        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
# Function to determine the best move for the computer
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

# Main function to run the Tic-Tac-Toe game

def tic_tac_toe():

    board = [[' ' for _ in range(3)] for _ in range(3)]  # Initialize empty board

    turn = 0  # Keeps track of turns

    while True:

        print_board(board)

        if turn % 2 == 0:

            print("Player X's turn.")

            try:

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

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

                    print("Cell already taken! Try again.")

                    continue

                board[row][col] = 'X'

            except (ValueError, IndexError):

                print("Invalid input! Enter numbers between 0 and 2.")

                continue

        else:

            print("Computer O's turn.")

            row, col = best_move(board)  # Computer selects the best move

            board[row][col] = 'O'

            print(f"Computer chose: {row} {col}")
```

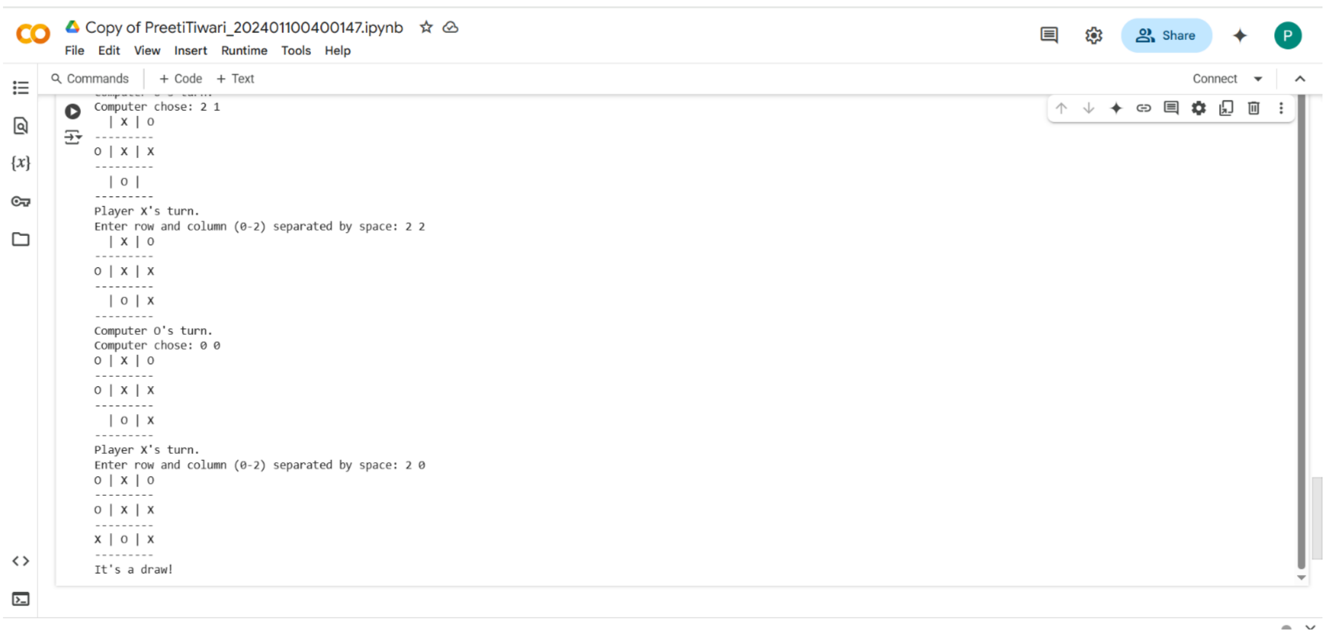
```
 # Check for a winner
        winner = check_winner(board)
        if winner:
            print_board(board)
            print(f"Player {winner} wins!")
            break
      # Check for a draw
        if is_draw(board):
            print_board(board)
            print("It's a draw!")
            break
        turn += 1
# Run the game if the script is executed
if __name__ == "__main__":
    tic_tac_toe()
```

Output



Copy of PreetiTiwari_202401100400147.ipynb

```
  |   |
---------
  |   |
---------
  |   |
---------
Player X's turn.
Enter row and column (0-2) separated by space: 1 2
  |   |
---------
  |   | X
---------
  |   |
---------
Computer O's turn.
Computer chose: 0 2
  |   | O
---------
  |   | X
---------
  |   |
---------
Player X's turn.
Enter row and column (0-2) separated by space: 1 1
  |   | O
---------
  | X | X
---------
  |   |
---------
Computer O's turn.
Computer chose: 1 0
  |   | O
---------
O | X | X
```



Copy of PreetiTiwari_202401100400147.ipynb

```
Computer chose: 2 1
  | X | O
---------
O | X | X
---------
  | O |
---------
Player X's turn.
Enter row and column (0-2) separated by space: 2 2
  | X | O
---------
O | X | X
---------
  | O | X
---------
Computer O's turn.
Computer chose: 0 0
O | X | O
---------
O | X | X
---------
  | O | X
---------
Player X's turn.
Enter row and column (0-2) separated by space: 2 0
O | X | O
---------
O | X | X
---------
X | O | X
---------
It's a draw!
```

References/Credits:-

- ChatGPT
- Google Colab and Google