Problem statement: Tic-Tac-Toe solver

Name: Anirudh Kumar

Roll Number: 202401100300045

Branch: CSE AI-A

Class roll no: 45

Introduction

Tic-Tac-Toe is a widely known two-player game played on a 3x3 grid, where players alternate turns placing their marks (X or O) in an attempt to form a winning row, column, or diagonal.

This project implements an AI-powered Tic-Tac-Toe Solver using the Minimax Algorithm with Alpha-Beta Pruning . The AI plays optimally, ensuring it either wins the game or forces a draw.

Methodology

1. Game Representation:

- The Tic-Tac-Toe board is modeled as a 3x3 matrix .
- Players make moves by placing 'X' or 'O' on the board.
- The game terminates when a player wins or when all cells are filled (tie).

2. Minimax Algorithm:

- The AI evaluates all possible moves recursively to determine the best possible decision.
- It assigns scores based on outcomes:
  - +1 for an AI (X) win
  - -1 for an opponent (O) win
  - 0 for a draw

3. Alpha-Beta Pruning:

- The Alpha-Beta Pruning technique is incorporated to enhance the efficiency of the Minimax Algorithm .
- This optimization helps in reducing unnecessary computations by eliminating branches that don’t need to be explored.

4. Implementation Details:

- Developed in Python .

- Executes in a loop where the AI and human player take turns.

- Utilizes recursion to compute the best move using Minimax.

Code

python

```
def print_board(board):
  """Prints the current state of the Tic-Tac-Toe board."""
  for row in board:
    print(" | ".join(row))
    print("-"   9)


def check_winner(board):
  """Checks if there's a winner or a tie."""
    Check rows
  for row in board:
    if row[0] == row[1] == row[2] != ' ':
      return row[0]

    Check columns
  for col in range(3):
    if board[0][col] == board[1][col] == board[2][col] != ' ':
      return board[0][col]

    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]

  Check for tie
 for row in board:
  if ' ' in row:
   return None   No winner yet
 return 'Tie'

def minimax(board, depth, is_maximizing):
 """Minimax algorithm with alpha-beta pruning."""
 winner = check_winner(board)
 if winner:
  if winner == 'X':
   return 1
  elif winner == 'O':
   return -1
  else:
   return 0

 if is_maximizing:
  best_score = -float('inf')
  for i in range(3):
   for j in range(3):
    if board[i][j] == ' ':
     board[i][j] = 'X'
     score = minimax(board, depth + 1, False)
```

```
          board[i][j] = ' '    Backtrack
          best_score = max(score, best_score)
      return best_score
    else:
      best_score = float('inf')
      for i in range(3):
        for j in range(3):
          if board[i][j] == ' ':
            board[i][j] = 'O'
            score = minimax(board, depth + 1, True)
            board[i][j] = ' '    Backtrack
            best_score = min(score, best_score)
      return best_score



def find_best_move(board):
    best_score = -float('inf')
    best_move = (-1, -1)

    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = 'X'
                score = minimax(board, 0, False)
                board[i][j] = ' '    Backtrack
                if score > best_score:
```

```
                best_score = score
                best_move = (i, j)
    return best_move



  Example usage:
board = [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]
print_board(board)

while True:
    move = find_best_move(board)
    board[move[0]][move[1]] = 'X'
    print_board(board)
    winner = check_winner(board)

    if winner:
        print(f"{winner} wins!")
        break

      Placeholder for user input (replace with actual user input)
    row, col = map(int, input("Enter row and column (0-2, space-separated):
").split())
    board[row][col] = 'O'

    print_board(board)
    winner = check_winner(board)
    if winner:
```

```
print(f"{winner} wins!")

break
```

Output/Result

1. AI makes an optimal move:

   - The AI determines the best possible move using Minimax Algorithm .

   - AI places 'X' in an advantageous position.

2. Game Progression:

   - The AI and the human player take turns making moves.

   - The board updates after each move.

3. Final Outcome:

   - The game concludes when a player wins or the board is filled.

   - The AI ensures a win or forces a tie.

References/Credits

- Minimax Algorithm : Fundamental AI decision-making technique.
- Alpha-Beta Pruning : Optimization method to enhance efficiency.
- Python Implementation: Developed and executed in Google Colab .
