Tic-Tac-Toe Game with AI Using Minimax Algorithm

Title Page

Project: Tic-Tac-Toe Solver

Name: Daksh Singh

Roll No.: 202401100400073

Course: AI for Engineers

Institution: KIET Group of Institutions

Date: March 10, 2025

Introduction:

Tic-Tac-Toe is a common two-player board game in which three marks ('X' or 'O') are to be obtained in a row, column, or diagonal on a grid of 3x3. Traditionally, the game is played between two players, but here we are simulating one player using AI. The AI will play 'O' and the other player will play 'X'. This is a problem to create an AI based on a famous algorithm named Minimax that determines the optimum moves for a game of Tic-Tac-Toe. The AI will try to maximize its winning chances and reduce the winning chances of the human player.

Visual Representation of Tic-Tac-Toe Board:

```
markdown
Copy
   |   |
-----------
   |   |
-----------
   |   |
```

The below picture shows an initial empty 3x3 grid, where the players will make their moves marking ('X' for human and 'O' for AI).

Methodology:

To solve this problem, the following methodology was adopted:

1. Game Representation: The Tic-Tac-Toe game board is represented in Python using a 2D list (a 3x3 matrix). Any cell may contain either 'X', 'O', or an empty space (" ").

2. The Minimax algorithm is a recursive decision-making process that is used to compute the best move for the AI. It goes through all possible moves and selects the one that will maximize the probability of the AI winning. The algorithm alternates between maximizing and minimizing, based on whether it is the turn of the AI or the human player.

3. It attempts all potential board positions by making each move and forecasting the next move and so on until it reaches a terminal position (win, lose, or draw).

4. Game Flow: The human player makes their move by specifying the row and column.

5. The AI makes its move by evaluating all possible outcomes using the Minimax algorithm.

6. The board gets updated and the winner is established after each move. The game continues until a player is declared the winner or all the cells are filled (resulting in a draw).

7. Winner Checking: The game checks for the winner after each move by looking at the rows, columns, and diagonals. When a player achieves three marks in a row, column, or diagonal, then they are the winner.

8. **Game Termination:** The game is finished when there is a winner or the board is completed but no one has won, and that's a draw.

Code:

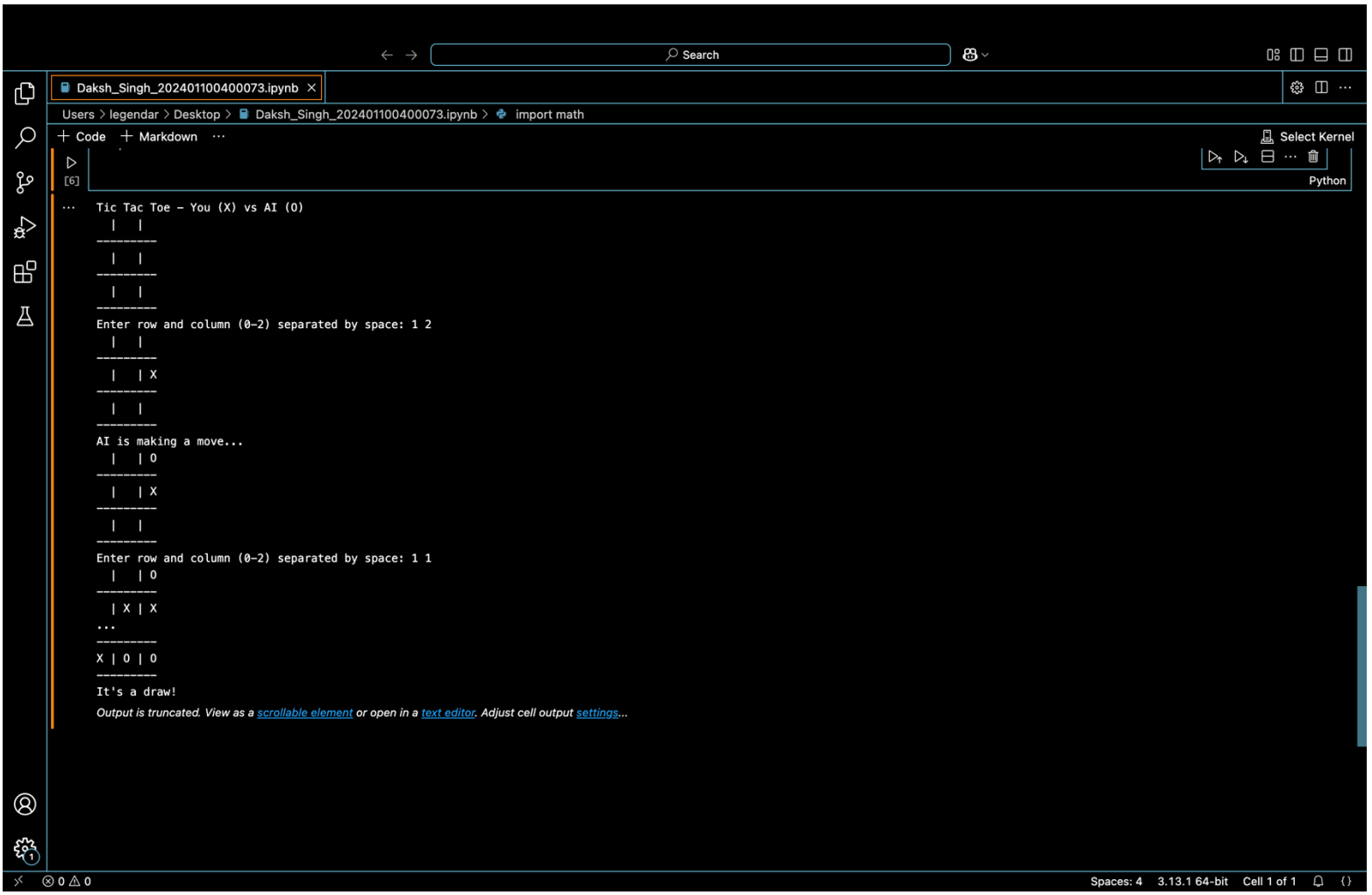
```
import math

# It is the Function for printing the Tic Tac Toe board.
def print_board(board):
    for row in board:
        print(" | ".join(row))  # I used '|' for separator.
        print("-" * 9)  # For print the separator between rows.

# Function to check if there is a winner or if the game is a draw
def check_winner(board):
    # It checks the rows for a winner
    for row in range(3):
        if board[row][0] == board[row][1] == board[row][2] != " ":
            return board[row][0]

    # It checks the columns for a winner
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != " ":
            return board[0][col]

    # It will Check the diagonal for a winner
    if board[0][0] == board[1][1] == board[2][2] != " ":
        return board[1][1]
    if board[0][2] == board[1][1] == board[2][0] != " ":
        return board[1][1]

    # Check if the board is full (Draw case)
    if all(board[r][c] != " " for r in range(3) for c in range(3)):
        return "Draw"

    return None  # No winner yet

# Minimax algorithm to find the best move for AI
def minimax(board, depth, is_maximizing):
    winner = check_winner(board)
    if winner == "X":
        return -10 + depth  # Human wins
    elif winner == "O":
        return 10 - depth  # AI wins
    elif winner == "Draw":
        return 0  # Game is a draw

    if is_maximizing:  # AI's turn (maximize score)
        best_score = -math.inf
        for r in range(3):
            for c in range(3):
```

```
                if board[r][c] == " ":  # Check empty cells
                    board[r][c] = "O"
                    score = minimax(board, depth + 1, False)
                    board[r][c] = " "  # Undo move
                    best_score = max(best_score, score)
        return best_score
    else:  # Human's turn (minimize AI's score)
        best_score = math.inf
        for r in range(3):
            for c in range(3):
                if board[r][c] == " ":
                    board[r][c] = "X"
                    score = minimax(board, depth + 1, True)
                    board[r][c] = " "  # Undo move
                    best_score = min(best_score, score)
        return best_score

# Function to find the best possible move for AI
def best_move(board):
    best_score = -math.inf
    move = (-1, -1)
    for r in range(3):
        for c in range(3):
            if board[r][c] == " ":
                board[r][c] = "O"
                score = minimax(board, 0, False)
                board[r][c] = " "  # Undo move
                if score > best_score:
                    best_score = score
                    move = (r, c)
    return move

# Main Function to run the Tic Tac Toe game
def tic_tac_toe():
    board = [[" " for _ in range(3)] for _ in range(3)]# Start with an empty board.
    print("Tic Tac Toe - You (X) vs AI (O)")
    print_board(board)

    for turn in range(9):  # Maximum of 9 moves in a 3x3 grid
        if turn % 2 == 0:  # Human player's turn (X)
            while True:
                try:
                    row, col = map(int, input("Enter row and column (0-2) separated by
space: ").split())
                    if board[row][col] == " ":  # Check if cell is empty
                        board[row][col] = "X"
                        break
                    else:
                        print("Cell already taken! Choose again.")
                except (ValueError, IndexError):
                    print("Invalid input! Enter numbers between 0-2.")
        else:  # AI's turn (O)
            print("AI is making a move...")
```

```
            row, col = best_move(board)
            board[row][col] = "O"

        print_board(board)  # Display the updated board.
        winner = check_winner(board)
        if winner:  # If the game ends (someone wins or it's a draw).

            if winner == "Draw":
                print("It's a draw!")
            else:
                print(f"{winner} wins!")
            return

# Run the game if this script is executed directly.

if __name__ == "__main__":
    tic_tac_toe()
    # Start the Tic Tac Toe game.
```

Output/Result:

To showcase the output, here is a **screenshot of the game in progress**:



In the game output, you will see:

- A Tic Tac Toe solver demonstrates key game theory principles.
- The Minimax algorithm ensures the AI is unbeatable, leading to a win or draw if the opponent plays optimally.
- Future enhancements could extend these methods to more complex games like Connect Four or Chess.

References/Credits:

- "Artificial Intelligence: A Modern Approach" by Stuart Russell and Peter Norvig.
- Online resources and research papers on game-solving algorithms.
