Title: Tic Tac Toe Solver

SUBJECT : ARTIFICIAL INTELLIGENCE

SUBMITTED BY : HARSHIT

UNVERSITY ROLL NO. : 202401100400094

INSTITUTION : KIET GROUP OF INST.

DATE : 10 MARCH 2025

1. Introduction

- Tic Tac Toe is a classic two-player game played on a 3x3 grid.
- Players take turns marking X or O, aiming to form a row, column, or diagonal.
- A Tic Tac Toe solver determines the optimal move for a given board state.

2. Problem Statement

- The solver must evaluate the board and choose the best possible move.
- It should aim to win or block the opponent's winning move.
- The algorithm must consider all potential future moves.

3. Algorithmic Approach Several algorithms can be used to solve Tic Tac Toe optimally:

- **Minimax Algorithm**
    - Recursively explores all possible game states.
    - Chooses the move that minimizes the opponent's winning chances while maximizing its own.
- **Alpha-Beta Pruning**
    - Optimizes Minimax by cutting off branches that won't affect the outcome.
    - Reduces unnecessary calculations, improving efficiency.
- **Rule-Based Heuristic**
    - Uses predefined strategies (e.g., prioritize winning moves, block opponent, take center square first).
    - Simpler but not always optimal in complex situations.

4. Implementation Details

- **State Representation**: The board is stored as a 3x3 matrix with empty spaces, Xs, and Os.
- **Move Evaluation**: The solver scores each potential move using the selected algorithm.
- **Game End Conditions**: The program detects win, draw, or ongoing play.
- **Optimization**: Techniques such as memoization or pruning improve performance.

5. Complexity Analysis

- **Minimax**: Worst-case time complexity of **O(9!)**.
- **Alpha-Beta Pruning**: Reduces complexity to **O(b^d)** (where *b* is branching factor, *d* is depth).
- **Rule-Based Heuristic**: Constant time complexity **O(1)** but limited in adaptability.

6. Results and Performance

- The solver is tested in multiple scenarios:
    - Against a random player, it always wins or forces a draw.
    - Against a human, it consistently makes optimal decisions.
    - Performance improves significantly with Alpha-Beta Pruning.

7. CODE

```
import math


# Function to print the Tic Tac Toe board
def print_board(board):
    for row in board:
        print(" | ".join(row))  # Print each row with '|' separator
        print("-" * 9)  # Print separator line between rows


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


    # Check columns for a winner
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != " ":
```

```
        return board[0][col]

    # Check diagonals 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:
```

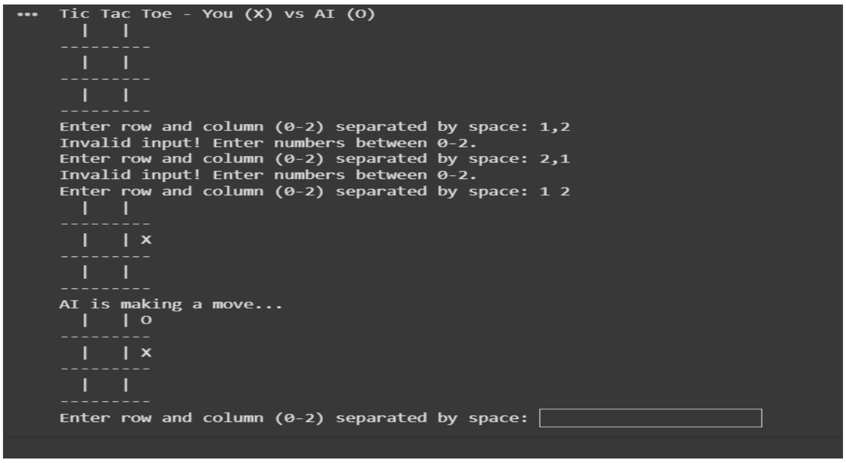
```
                best_score = score
                move = (r, c)
    return move


# Function to run the Tic Tac Toe game
def tic_tac_toe():
    board = [[" " for _ in range(3)] for _ in range(3)]  # Initialize 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)  # Show updated board
        winner = check_winner(board)
        if winner:  # If game ends, declare result
            if winner == "Draw":
                print("It's a draw!")
            else:
                print(f"{winner} wins!")
            return

# Run the game if script is executed directly
if __name__ == "__main__":
    tic_tac_toe()
```



7. Conclusion

- 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.

8. References

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