Title: Tic-Tac-Toe Solver

Name: Raj Ojha

Roll no: 45

University Roll no:202401100300192

Library I’d: 2428CSEAI1369

Introduction

Tic-Tac-Toe Game with AI and Difficulty Levels

This is a classic game of Tic-Tac-Toe where you can challenge either your strategic thinking or your skills against an AI opponent. You can choose between two difficulty levels for the AI, providing a dynamic and engaging experience based on your preferences.

Key Features:

1. Two Players: You play as a human (symbol "O") and the AI plays as "X".
2. Difficulty Levels: Choose between:
    - Easy: The AI makes random moves, offering a more casual experience.
    - Hard: The AI uses a Minimax algorithm, making the best possible moves and providing a challenging opponent.
3. Game Flow:
    - Players take turns to fill the grid.
    - The game ends when either player wins or when the board is full with no winner, resulting in a draw.
4. User Input: Players input moves using numbers 1 to 9, corresponding to positions on the Tic-Tac-Toe grid.

Game Objective:

The goal is to get three of your marks (either "X" or "O") in a row, either horizontally, vertically, or diagonally, before your opponent does.

Methodology and algorithm

Methodology and Algorithms Used in the Tic-Tac-Toe Game

This Tic-Tac-Toe game incorporates two primary algorithms to control the AI's behavior and ensure a dynamic and engaging experience for the player. These algorithms, combined with well-defined game logic, enable both a basic and an advanced gameplay experience.

---

1. Random Move Algorithm (Easy Difficulty)

Objective: In the easy difficulty setting, the AI makes random moves to provide a less challenging opponent.

How It Works:

- Identify Empty Cells: The algorithm scans the 3x3 game board and collects the coordinates of all empty cells (those that contain the EMPTY symbol).
- Select Random Cell: After identifying the empty cells, the algorithm randomly selects one of them.
- Place AI Symbol: The AI places its symbol (X) in the randomly chosen empty cell.

This approach is simple and does not involve strategic thinking, making it suitable for a more casual gameplay experience.

---

2. Minimax Algorithm (Hard Difficulty)

Objective: The Minimax algorithm is used in the hard difficulty setting to simulate a perfect AI opponent that selects the optimal move by evaluating all possible future game states.

How It Works:

- Maximizing and Minimizing Players:
    - The maximizing player (AI) attempts to maximize its score, aiming for a win.
    - The minimizing player (human) attempts to minimize the AI's score by avoiding a loss.
- Recursion: The algorithm uses recursion to simulate all possible future game states. For each move, the algorithm recursively evaluates the outcomes, considering the moves of both players until a terminal state (win, loss, or draw) is reached.
- Terminal States: If the game reaches a terminal state (win, loss, or draw), the algorithm assigns a score:
    - A win for the AI results in a score of 1.
    - A loss for the AI (human win) results in a score of -1.
    - A draw results in a score of 0.
- Score Propagation: As the recursion backtracks, the algorithm chooses the best possible move for each player:
    - For the maximizing player (AI), the algorithm selects the move with the highest score.
    - For the minimizing player (human), the algorithm selects the move with the lowest score.
- Optimal Move Selection: The AI ultimately selects the move that guarantees the best possible outcome (maximizing its chances of winning or minimizing the chances of losing).

This algorithm ensures that the AI plays optimally, providing a challenging opponent for the player.

Game Methodology

The game integrates these two algorithms into a seamless experience for the player, following a well-structured flow:

1. Game Board Representation

The game board is a 3x3 grid, represented as a 2D list in Python. Each cell can contain:

- AI (symbol "X") for AI moves.
- HUMAN (symbol "O") for player moves.
- EMPTY for unoccupied cells.

2. Player and AI Turns

The game alternates turns between the player and the AI:

- The player always plays first and can choose an empty cell to place their symbol (O).
- The AI plays second and selects its move based on the chosen difficulty:
    - In easy mode, the AI makes a random move.
    - In hard mode, the AI uses the Minimax algorithm to select the optimal move.

3. Difficulty Levels

- Easy Difficulty: The AI chooses a random empty cell from the board. This provides a casual, non-strategic experience for the player.
- Hard Difficulty: The AI selects the optimal move by evaluating all possible future game states using the Minimax algorithm. This provides a challenging opponent who plays optimally.

4. Win Conditions and Draw Check

The game checks for a winner or draw after every move:

- Win Conditions: The game checks if either the player or the AI has completed a winning combination, such as three marks in a row, column, or diagonal.
- Draw Condition: If the board is full and no winner has been determined, the game ends in a draw.

5. User Input

The player is prompted to enter their move, which corresponds to a number between 1 and 9. Each number maps to a specific position on the board. The input is validated to ensure it corresponds to an empty cell.

6. Game Flow Control

The game operates in a loop:

- The player inputs their move.
- The board is updated, and the game checks for a winner or draw.
- The AI then makes its move, either randomly (easy mode) or optimally (hard mode).
- After each move, the board is printed, and the game continues until there is a winner or a draw.

7. Turn-Based Gameplay

- The player's turn: The player selects an empty cell and places their symbol on the board.
- The AI's turn: After the player's move, the AI takes its turn based on the selected difficulty.
- This alternating process continues until the game ends.

Conclusion

By combining the Random Move algorithm (for easy difficulty) and the Minimax algorithm (for hard difficulty), the Tic-Tac-Toe game provides players with two distinct levels of challenge. The Minimax algorithm ensures that the AI plays optimally in hard mode, while the random move algorithm in easy mode allows players to practice and enjoy the game without the pressure of a strategic opponent. Together, these algorithms form the core gameplay mechanics, creating an engaging experience for both beginners and advanced players.

Code Typed

```
import random

import math

# Constants for the players

AI = "X"  # AI is the maximizing player

HUMAN = "O"  # Human is the minimizing player

EMPTY = " "  # Empty cell in the board

# Function to print the Tic-Tac-Toe board

def print_board(board):

   for i, row in enumerate(board):

      print(" | ".join(row))  # Print the row with cells separated by " | "

      if i != 2:

         print("-" * 8)  # Print separator only after the first and second
rows

   print("\n")
```

```
# Check if a player has won
def check_winner(board):
    win_combinations = [
        [(0, 0), (0, 1), (0, 2)],  # Row 1
        [(1, 0), (1, 1), (1, 2)],  # Row 2
        [(2, 0), (2, 1), (2, 2)],  # Row 3
        [(0, 0), (1, 0), (2, 0)],  # Column 1
        [(0, 1), (1, 1), (2, 1)],  # Column 2
        [(0, 2), (1, 2), (2, 2)],  # Column 3
        [(0, 0), (1, 1), (2, 2)],  # Diagonal 1
        [(0, 2), (1, 1), (2, 0)]   # Diagonal 2
    ]
    for combination in win_combinations:
        values = [board[x][y] for x, y in combination]
        if values == [AI, AI, AI]:
            return AI
        if values == [HUMAN, HUMAN, HUMAN]:
            return HUMAN
    return None
# Check if the game is a draw
def is_draw(board):
    for row in board:
        if EMPTY in row:
            return False
```

```
    return True


# MiniMax Algorithm to find the best move (for Hard difficulty)
def minimax(board, is_maximizing):
    winner = check_winner(board)
    if winner == AI: return 1  # AI wins, return 1
    if winner == HUMAN: return -1  # Human wins, return -1
    if is_draw(board): return 0  # Draw, return 0
    if is_maximizing:  # AI's turn (maximize score)
        best_score = -math.inf  # Start with a very low score
        for i in range(3):
            for j in range(3):
                if board[i][j] == EMPTY:  # If the cell is empty
                    board[i][j] = AI  # Try placing AI's symbol
                    score = minimax(board, False)  # Recursively evaluate the
move
                    board[i][j] = EMPTY  # Undo the move
                    best_score = max(best_score, score)  # Maximize score for AI
        return best_score
    else:  # Human's turn (minimize score)
        best_score = math.inf  # Start with a very high score
        for i in range(3):
            for j in range(3):
                if board[i][j] == EMPTY:  # If the cell is empty
```

```
                board[i][j] = HUMAN  # Try placing Human's symbol
                score = minimax(board, True)  # Recursively evaluate the
move
                board[i][j] = EMPTY  # Undo the move
                best_score = min(best_score, score)  # Minimize score for
Human
    return best_score
# Find the best move for AI (for Hard difficulty)
def find_best_move(board, difficulty):
  if difficulty == 'easy':
    # AI makes a random move (easy difficulty)
    empty_cells = [(i, j) for i in range(3) for j in range(3) if board[i][j] ==
EMPTY]
    return random.choice(empty_cells)  # Choose a random empty
cell
  else:
    # AI uses minimax for Hard difficulty
    best_score = -math.inf
    best_move = None
    for i in range(3):
      for j in range(3):
        if board[i][j] == EMPTY:
          board[i][j] = AI
          score = minimax(board, False)
          board[i][j] = EMPTY
```

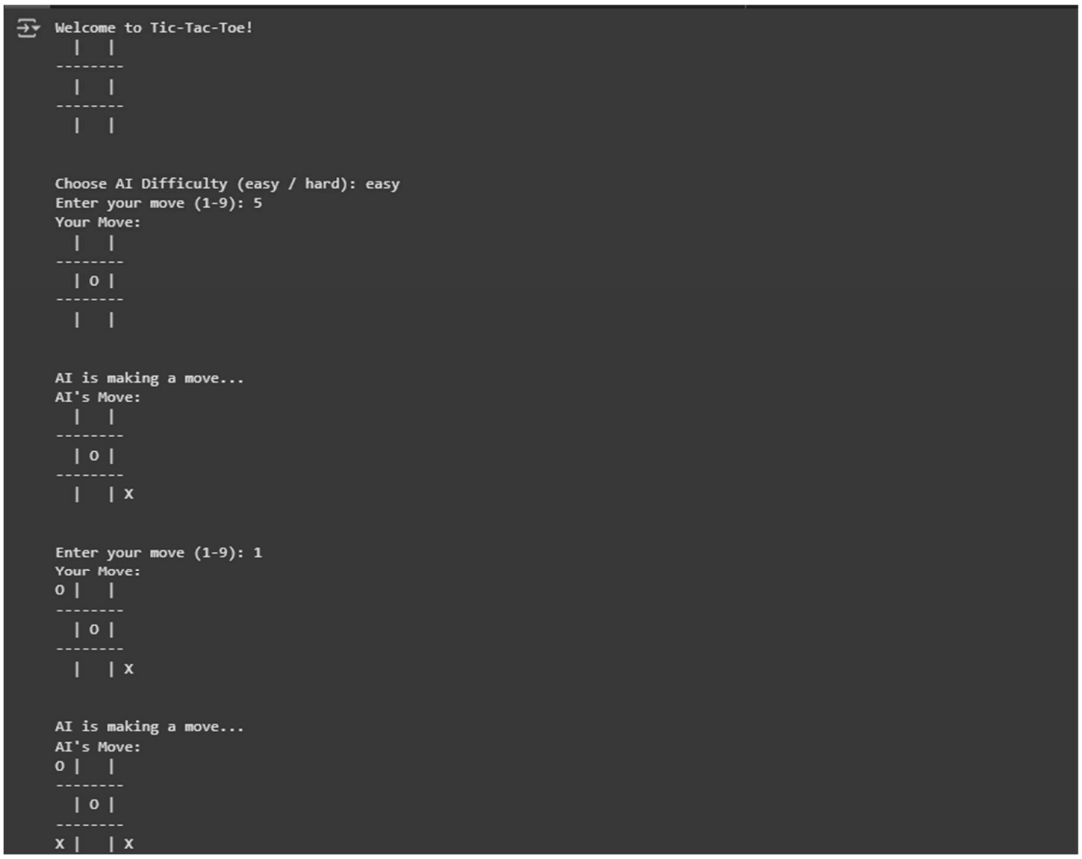
```
            if score > best_score:
                best_score = score
                best_move = (i, j)
    return best_move
# Function for the user to input their move
def user_move(board):
  while True:
    try:
      move = int(input("Enter your move (1-9): ")) - 1  # User inputs a number from 1-9
      row, col = divmod(move, 3)  # Convert the input to board indices
      if board[row][col] == EMPTY:  # Check if the cell is empty
        board[row][col] = HUMAN  # Place the Human's symbol
        break
      else:
        print("Cell already occupied. Try again.")
    except (ValueError, IndexError):
      print("Invalid move! Please enter a number from 1-9 corresponding to an empty cell.")
# Main function to run the game
def play_game():
  board = [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]]
  # Print the current board
  print("Welcome to Tic-Tac-Toe!")
```

```
    print_board(board)
    # Ask for the difficulty level
    difficulty = input("Choose AI Difficulty (easy / hard): ").lower()
    while difficulty not in ['easy', 'hard']:
        difficulty = input("Invalid choice. Please choose 'easy' or 'hard': ").lower()
    # Main game loop
    while True:
        # Player (Human) move
        user_move(board)
        print("Your Move:")
        print_board(board)
        if check_winner(board):
            print("Congratulations, you win!")
            break
        if is_draw(board):
            print("It's a draw!")
            break
        # AI move (Optimal or Random based on difficulty)
        print("AI is making a move...")
        best_move = find_best_move(board, difficulty)
        if best_move:
            board[best_move[0]][best_move[1]] = AI  # AI makes the move
            print("AI's Move:")
```

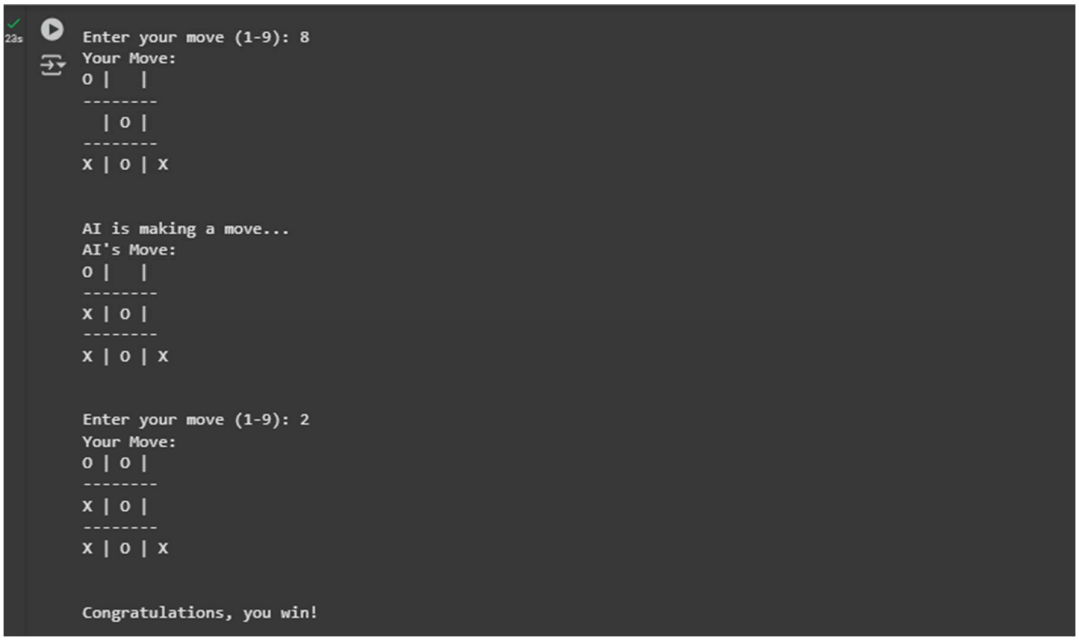
```
            print_board(board)
        else:
            print("Game Over! No moves left.")
            break
        if check_winner(board):
            print("AI wins!")
            break
        if is_draw(board):
            print("It's a draw!")
            break
# Start the game
play_game()
```

Output

easy

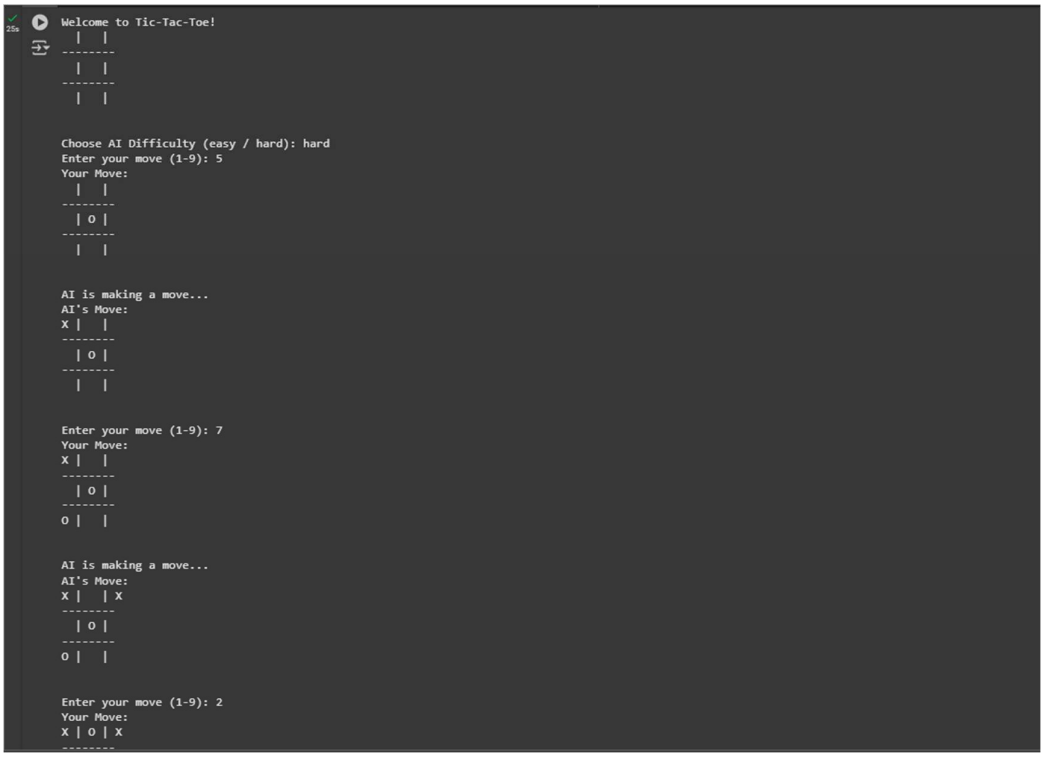


```
Welcome to Tic-Tac-Toe!
  |   |
--------
  |   |
--------
  |   |


Choose AI Difficulty (easy / hard): easy
Enter your move (1-9): 5
Your Move:
  |   |
--------
  | O |
--------
  |   |


AI is making a move...
AI's Move:
  |   |
--------
  | O |
--------
  |   | X


Enter your move (1-9): 1
Your Move:
O |   |
--------
  | O |
--------
  |   | X


AI is making a move...
AI's Move:
O |   |
--------
  | O |
--------
X |   | X
```

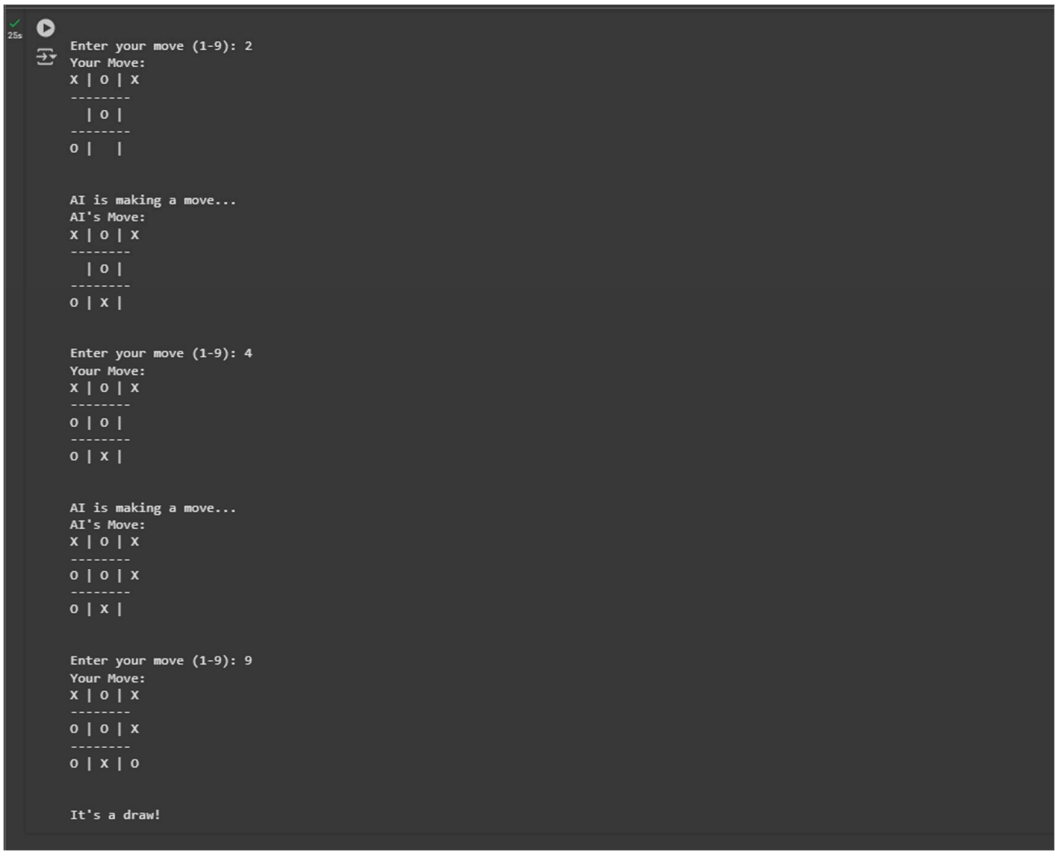


```
Enter your move (1-9): 8
Your Move:
O |   |
--------
  | O |
--------
X | O | X


AI is making a move...
AI's Move:
O |   |
--------
X | O |
--------
X | O | X


Enter your move (1-9): 2
Your Move:
O | O |
--------
X | O |
--------
X | O | X


Congratulations, you win!
```

Hard



```
Welcome to Tic-Tac-Toe!
  |   |
--------
  |   |
--------
  |   |


Choose AI Difficulty (easy / hard): hard
Enter your move (1-9): 5
Your Move:
  |   |
--------
  | O |
--------
  |   |


AI is making a move...
AI's Move:
X |   |
--------
  | O |
--------
  |   |


Enter your move (1-9): 7
Your Move:
X |   |
--------
  | O |
--------
O |   |


AI is making a move...
AI's Move:
X |   | X
--------
  | O |
--------
O |   |


Enter your move (1-9): 2
Your Move:
X | O | X
--------
```



```
Enter your move (1-9): 2
Your Move:
X | O | X
--------
  | O |
--------
O |   |


AI is making a move...
AI's Move:
X | O | X
--------
  | O |
--------
O | X |


Enter your move (1-9): 4
Your Move:
X | O | X
--------
O | O |
--------
O | X |


AI is making a move...
AI's Move:
X | O | X
--------
O | O | X
--------
O | X |


Enter your move (1-9): 9
Your Move:
X | O | X
--------
O | O | X
--------
O | X | O


It's a draw!
```