Project Report: Tic-Tac-Toe Game

Date: 24.11.2024

Overview

The Tic-Tac-Toe game is a popular two-player strategy game where players take turns marking spaces in a 3x3 grid. The goal is to align three marks (either "X" or "O") in a row—horizontally, vertically, or diagonally. This project implements the Tic-Tac-Toe game in Python, allowing a human player to compete against an AI opponent. The AI opponent can play at varying levels of difficulty, ranging from random moves (Easy) to strategic gameplay using the Minimax algorithm (Hard).

Goals

- **Develop a fully functional command-line Tic-Tac-Toe game.**
- **Ensure clear and intuitive user interaction, with detailed prompts.**
- **Implement an AI opponent with multiple difficulty levels: Easy, Medium, and Hard.**
- **Structure the code to be modular, maintainable, and easy to modify for future enhancements.**

Specifications

- **The game board is represented as a 3x3 grid.**
- **The human player is assigned the symbol "X," and the AI opponent uses "O."**
- **The game tracks wins, losses, and ties, providing real-time feedback to the player.**
- **At the end of each game, players have the option to restart the game with a new difficulty setting.**

Implementation Details

1. Game Initialization

The game board is initialized as a 3x3 matrix with each cell set to `-1`, indicating that the cell is empty. The game is designed to allow players to make moves by entering numbers between 1 and 9, corresponding to grid positions.

python

Code

```
board = [[-1 for _ in range(3)] for _ in range(3)]
```

2. Displaying the Game Board

The `print_board(board)` function outputs the current state of the game board. It maps the `-1` value to a blank space, `1` to 'X' (player), and `0` to 'O' (AI).

python

Code

```
def print_board(board):
    symbols = {1: 'X', 0: 'O', -1: ' '}
    for row in board:
        print(" | ".join(symbols[cell] for cell in row))
        print("-" * 9)
```

3. Checking for a Winner

The `check_win(board)` function evaluates the board for a win. It checks for all possible win conditions (rows, columns, diagonals) by verifying if all three positions in any line have the same player's symbol.

python

Code

```
def check_win(board):
    win_conditions = [
        [(0, 0), (0, 1), (0, 2)],  # rows
        [(1, 0), (1, 1), (1, 2)],
        [(2, 0), (2, 1), (2, 2)],
        [(0, 0), (1, 0), (2, 0)],  # columns
        [(0, 1), (1, 1), (2, 1)],
        [(0, 2), (1, 2), (2, 2)],
        [(0, 0), (1, 1), (2, 2)],  # diagonals
        [(0, 2), (1, 1), (2, 0)]
    ]
    for condition in win_conditions:
        values = [board[x][y] for x, y in condition]
        if values[0] != -1 and all(v == values[0] for v in values):
            return True
    return False
```

4. Handling Tie Conditions

The `check_tie(board)` function checks if the game board is full and no player has won. If all cells are filled and no winning condition is met, the game results in a tie.

python

Code

```
def check_tie(board):
    return all(cell != -1 for row in board for cell in row)
```

5. Player's Turn

The function `get_player_input()` ensures that the human player selects a valid move by entering a number between 1 and 9. It then translates this input into corresponding board coordinates.

python

Code

```
def get_player_input():
    while True:
        try:
            move = int(input("Your turn (X), choose a cell (1-9): "))
            if move in range(1, 10):
                return divmod(move - 1, 3)
            else:
                print("Move must be between 1 and 9.")
        except ValueError:
            print("Invalid input. Please enter a number between 1 and
9.")
```

6. Computer's Turn (AI)

The `computer_turn(board, difficulty)` function controls the AI's move. Depending on the chosen difficulty level, the AI can make random moves (Easy), block the player or choose random moves (Medium), or use the Minimax algorithm for optimal gameplay (Hard).

python

Code

```
def computer_turn(board, difficulty):
    print("Computer's turn (O). Thinking...")
    if difficulty == "Easy":
        x, y = choice(get_empty_cells(board))
    elif difficulty == "Medium":
        move = medium_strategy(board)
    elif difficulty == "Hard":
        move = minimax(board, COMPUTER)
    board[x][y] = COMPUTER
    print_board(board)
```

7. Minimax Algorithm

The `minimax(board, player)` function implements the Minimax algorithm to determine the best move for the AI. It recursively evaluates future board states and chooses the move that maximizes the AI's chances of winning or minimizes its chances of losing.

python

Code

```
def minimax(board, player):
    if check_win(board):
        return [None, None, 1 if player == COMPUTER else -1]
    elif check_tie(board):
        return [None, None, 0]

    best = [None, None, -float('inf')] if player == COMPUTER else
[None, None, float('inf')]
    for x, row in enumerate(board):
        for y, cell in enumerate(row):
            if cell == -1:
                board[x][y] = player
                score = minimax(board, -player)
                board[x][y] = -1
                score[0], score[1] = x, y

                if player == COMPUTER and score[2] > best[2]:
                    best = score
                elif player == PLAYER and score[2] < best[2]:
                    best = score
    return best
```

8. Starting the Game

The `start_game()` function initializes the game, manages the gameplay loop, and allows the player to replay the game after each match.

python

Code

```
def start_game():
    board = [[-1 for _ in range(3)] for _ in range(3)]
    difficulty = choose_difficulty()
    print("Welcome to Tic-Tac-Toe!")
    print_board(board)
    while not check_win(board) and not check_tie(board):
        player_turn(board)
        print_board(board)
        if check_win(board):
            print("Congratulations, you won!")
            break
        if check_tie(board):
            print("It's a tie!")
            break
        computer_turn(board, difficulty)
        if check_win(board):
            print("The computer wins!")
```

```
            break

    if input("Play again? (y/n): ").lower() == 'y':

        start_game()
```

Milestones

1. **Game Initialization: Set up the board and initialize necessary game variables.**
2. **Player and AI Turns: Implemented alternating turns between the player and the AI.**
3. **Game Logic: Developed logic for detecting win and tie conditions.**
4. **AI Implementation: Incorporated varying levels of difficulty, including the Minimax algorithm for strategic AI behavior.**
5. **Replay Functionality: Added an option for players to restart the game after a match ends.**

Conclusion

This project successfully implements a classic Tic-Tac-Toe game in Python. The AI opponent provides varying levels of difficulty, ensuring a challenging experience for players. The use of the Minimax algorithm enhances the strategic depth of the AI, and the modular design of the code allows for future expansion and improvements. This project demonstrates strong problem-solving skills and effective use of algorithms in game development.