Project Report: Tic Tac Toe Game

Introduction

The **Tic Tac Toe Game with Player Score Management** is a Python-based project that integrates the timeless game of Tic Tac Toe with an efficient system for tracking and managing player scores. The application is designed using Python's **Object-Oriented Programming (OOP)** paradigm, highlighting the four pillars of OOP: **Encapsulation**, **Abstraction**, **Inheritance**, and **Polymorphism**.

This report provides a detailed overview of the project, its functionality, implementation, and the use of OOP concepts that make the system modular, scalable, and easy to maintain.

Objective

The objective of this project is to:

1. Provide a fun and interactive Tic Tac Toe gaming experience.
2. Enable users to compete in two-player or single-player modes.
3. Track, search, and manage player scores using persistent storage.

Features

Core Gameplay Features

- **Two-Player Mode**: Compete against a friend in a classic Tic Tac Toe match.
- **Single-Player Mode**: Play against an AI-powered computer opponent with basic strategy.
- **Winner Detection**: Check for winning patterns or draw conditions.

Player Score Management

- **Persistent Score Storage**: Save player scores in a CSV file (player_scores.csv).

- **Score Viewing**: View all recorded player scores.
- **Search Players**: Find individual player records by name.
- **Delete Scores**: Remove a player's score record.

Four Pillars of Python OOP in the Project

1. **Encapsulation**:
   - Classes like Board, Player, and PlayerScoreHandling encapsulate their data and methods, exposing only necessary functionalities.
   - Example: The Board class encapsulates the game grid and provides controlled access through methods like update and display.
2. **Abstraction**:
   - The project hides implementation details, allowing users to interact with high-level methods like play_game or get_move without worrying about underlying logic.
   - Example: Players use get_move to input their moves without needing to know how the game checks for valid moves.
3. **Inheritance**:
   - The Player class serves as the base class, with HumanPlayer and ComputerPlayer inheriting and extending its functionality.
   - Example: HumanPlayer overrides methods to get user input, while ComputerPlayer implements AI logic.
4. **Polymorphism**:
   - Polymorphism is utilized when get_move is called on either HumanPlayer or ComputerPlayer, allowing the same interface to behave differently.
   - Example: The game treats both players as Player objects, calling their respective get_move methods dynamically.

Implementation Details

Code Structure

The project is divided into modular components, making it easy to manage and extend:

1. **board.py**: Manages the Tic Tac Toe grid.
2. **player.py**: Defines the generic Player class.
3. **human_player.py**: Implements a human player's interaction.
4. **computer_player.py**: Implements the computer's AI logic.
5. **tic_tac_toe.py**: Handles game flow and logic.
6. **player_score_handling.py**: Manages player score data.
7. **main.py**: Integrates all components into a user-friendly menu system.

Key Classes and Methods

Board Class (board.py)

- **Encapsulation Example**:
  - self.cells: Maintains the state of the game grid.
  - update(), display(), is_full() methods provide controlled access to the board's state.

Player Class (player.py)

- **Inheritance Example**:
  - Acts as the base class for HumanPlayer and ComputerPlayer.
  - Contains common attributes like name and symbol.

HumanPlayer and ComputerPlayer Classes

- **Polymorphism Example**:
  - Both override the get_move method with distinct behaviors: user input for HumanPlayer and AI strategy for ComputerPlayer.

PlayerScoreHandling Class (player_score_handling.py)

- **Encapsulation Example**:

  - Manages CSV file operations (add_player_score, display_players, find_player, delete_player) while abstracting file handling details.

Technical Design

1. AI Strategy (ComputerPlayer)

The computer's moves are decided using:

1. Winning move detection.
2. Blocking opponent's winning move.
3. Preference for the center cell or corners.
4. Random move selection if no other strategy applies.

2. Persistent Data Handling

The player_scores.csv file ensures all scores are saved persistently, making the game state consistent across sessions.

How to Use the System

1. **Launching the Game**:
   - Run main.py to start the game and access the menu.
2. **Menu Options**:
   - **1**: Play against a friend.
   - **2**: Play against the computer.
   - **3**: View all recorded scores.
   - **4**: Search for a player's score by name.
   - **5**: Delete a player's score record.
   - **6**: Exit the application.
3. **Game Flow**:
   - Players alternate turns, inputting moves by selecting grid cells (1-9).

  - The game announces the winner or a draw when the board is full.

Conclusion

This project highlights Python's OOP capabilities through a feature-rich Tic Tac Toe game with score management. By adhering to the four OOP principles, the project ensures modularity, scalability, and ease of maintenance.

Future Enhancements

1. Add multiple difficulty levels for the AI.
2. Implement a graphical user interface (GUI) for better user experience.
3. Introduce online multiplayer functionality.

Appendix

Sample player_scores.csv File

Player 1	P1 Score	Player 2	P2 Score
Hamid	3	Hammad	2
Ahad	4	Computer	1

Example Menu Output

```
---: Welcome to Tic Tac Toe :---

1. Play with a friend

2. Play against the computer

3. Display player scores

4. Find Player

5. Delete player score

6. Exit

Enter your choice (1-6):
```

This structured design provides users with an enjoyable experience while adhering to robust coding practices.