Claude AI

Game Architecture

The Advanced Hangman Game is built using an object-oriented approach, encapsulating game logic within the Hangman Game class. This design promotes:

Modularity

Easier code maintenance

Flexibility for future enhancements

Key Components

1- Game Configuration

Word Categories: A dictionary storing themed word lists

Allows players to choose from categories like:

Programming

Animals

Countries

2-Difficulty Levels

Dynamic attempt allocation based on chosen difficulty:

Easy: 8 attempts

Medium: 6 attempts (default)

Hard: 4 attempts

3-Scoring Mechanism

Score calculated based on word length

High score persistence using JSON file storage

Tracks and saves player's best performance

Advanced Features

1-Input Validation

```python
def _get_player_guess(self, guessed_letters):
    # Ensures valid, unique letter inputs
    while True:
        guess = input("\nGuess a letter: ").lower()

        # Multiple validation checks:
        # - Single character
        # - Alphabetic input
        # - Unique guess
```

2-Dynamic Word Selection

```python
def choose_word_category(self):
    # Interactive category selection
    # Allows player to choose word theme
    # Randomly selects a word from chosen category
```

3-Game State Visualization

```python
def _display_game_state(self, word, guessed_letters, attempts_left):
    # Renders:
    # - Hangman ASCII art
    # - Current word progress
    # - Remaining attempts
    # - High score
```

Python Concepts Demonstrated

1. Object-Oriented Programming

Class-based design

Method encapsulation

Instance and class-level attributes

2. Error Handling

Try-except blocks for file operations

Input validation loops

3. File I/O

JSON file handling for score persistence

Dynamic file reading/writing

4. Advanced Data Structures

Sets for tracking guessed letters

Dictionaries for word categories and configuration

5. Control Flow

Nested loops for game logic

Conditional branching

State management

Technical Enhancements

Cross-platform screen clearing

Sleep-based game pacing

Interactive difficulty selection

Themed word categories

Potential Future Improvements

Network leaderboard

More diverse word categories

Multiplayer mode

Hint system

Learning Opportunities

This implementation serves as an educational project demonstrating:

Practical Python programming

Game design principles

User interaction techniques

Code organization strategies