Thought Sprint

Python-based word Game

Table of Contents

1. Introduction

The word-based game called Thought Sprint is built using Python application that integrates multiple functionalities, including user management, gameplay, and feedback collection. The game is designed to challenge players to memorize and recall words from different categories within a time limit, with the goal of providing an interactive and engaging experience. The system also offers features for administrators to manage users, track performance, and gather user feedback.

This report provides a comprehensive breakdown of the game's architecture, functionality, Features and the key components of the code. The game's features, user interface, file handling operations, and additional functionalities are discussed in detail, with an emphasis on both the user and admin roles.

2. Overview of the Game Design and Flow

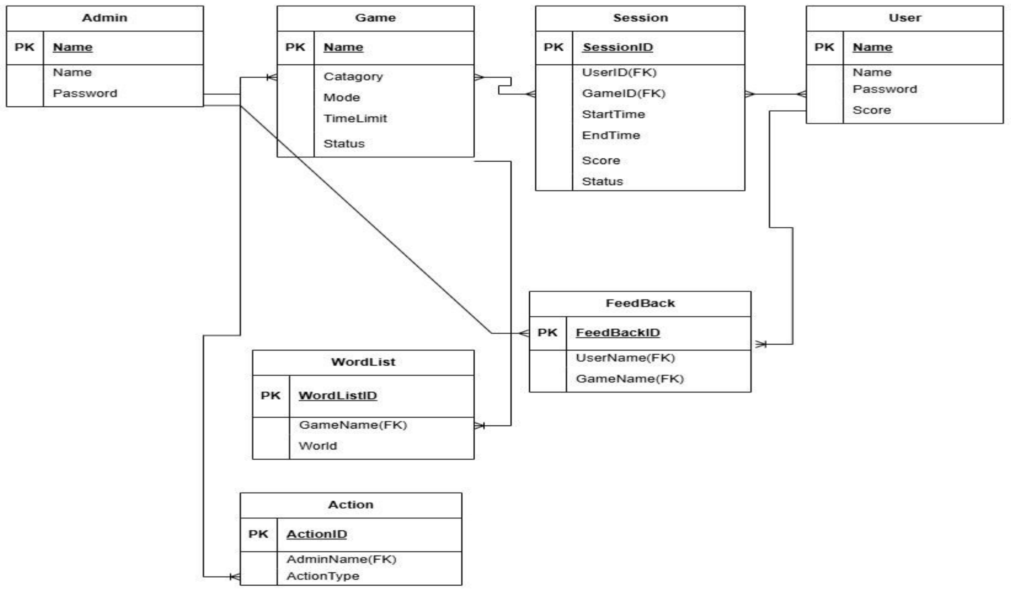
1. Game Design Overview

The game operates in a structured manner, with distinct functionalities tailored to two main roles: **admin** and **user**. These roles are governed by the following flow:

- **Admin Role**: The administrator is provided with the ability to manage registered users, view feedback, and handle data related to user performance. Admins have full access to the game's backend data and can make significant changes to the user base.
- **User Role**: Player's register, login, and participate in the Word Memorization Game. They are scored based on their ability to memorize and recall words within a specified time. Users can also send feedback, view their scores, and update their passwords.

Game Objective: The primary objective is for the player to memorize a set of words from predefined categories (Books, Movies, Famous People) and then attempt to recall those words by guessing them correctly within a limited time. The game allows for increasing difficulty based on time constraints and offers rewards for correct answers while penalizing incorrect ones.

2. Entity-Relationship (ER) Diagram and Data Flow



Relationships:

1. Admin-User Relationship (One-to-Many)
   - An Admin can manage multiple Users, but each User is managed by one Admin. The relationship is one-to-many (1:N) because an admin typically oversees multiple users, but each user has only one admin assigned to them.
2. User-Session-Game Relationship (Many-to-Many)
   - A User can participate in multiple Sessions of different Games, and a Game can have many Sessions from different users. This relationship is many-to-many (M:N), connected through the Session entity, which tracks the participation of each user in a particular game session.
3. User-Feedback-Game Relationship (One-to-Many)
   - A User can provide feedback for multiple Games they play, but each Feedback entry is associated with one User and one Game. This is a one-to-many (1:N) relationship, where a user can submit feedback for each game they play.
4. Game-WordList Relationship (One-to-Many)

    - Each Game has a corresponding WordList containing the words from predefined categories (e.g., Books, Movies, Famous People). The relationship is one-to-many (1: N), as each game can have multiple words, but each word list belongs to a single game.

5. Admin-Action Relationship (One-to-Many)
    - An Admin can perform many Actions, such as updating games, managing users, and reviewing feedback. This relationship is one-to-many (1:N), as one admin may perform multiple actions, but each action is associated with a single admin.

3. Game Mechanics

1. Word Categories:

The game revolves around three-word categories:

1. **Books**: Famous literary works, such as Moby Dick, 1984, The Great Gatsby, etc.
2. **Movies**: Popular films like Inception, Titanic, The Godfather, and Star Wars.
3. **Famous People**: Historical figures such as Einstein, Newton, Shakespeare, and Mahatma Gandhi.

These categories contain a predefined list of words, which are presented to players during the memorization phase.

2. Game Modes:

There are three distinct difficulty levels in the game:

- **Easy**: 60 seconds to guess the words.
- **Medium**: 45 seconds to guess the words.
- **Hard**: 30 seconds to guess the words.

The time limit is associated with each difficulty mode and impacts how long the player has to recall and guess the words. Players are given a brief period (10 seconds) to memorize the words from the selected categories before the guessing phase begins.

3. Gameplay Flow:

- **Memorization Phase**: Players are shown the words from the selected categories for 10 seconds, after which the words are hidden, and the player is asked to recall them.

- **Guessing Phase**: The player is prompted to guess words from one of the categories, and for every correct guess, they earn points and extra time. Incorrect guesses reduce the remaining time.

4. Scoring:

- **Correct Guess**: +5 points and +5 seconds added to the remaining time.
- **Incorrect Guess**: -5 seconds deducted from the remaining time.

End of Game: The game ends when the timer reaches zero or when the player has guessed all the words. The player's final score is calculated based on the number of correct answers.

4. Functionality & Features Breakdown:

Admin Functionalities & Features:

1. Admin Login

This feature allows admins to log in using a fixed username and password. When the admin enters the correct credentials, they gain access to the admin dashboard.

Code:

```
def admin_login(username, password):     if
username == "admin" and password == "1234":
        return "Login Successful"
else:         return "Invalid
Credentials"
```

- **How it works:** The function compares the entered username and password with predefined credentials ("admin" and "1234"). If they match, access is granted.

2. View All Users

Admins can view a list of all registered users, along with their scores and feedback.

Code:
```
def view_all_users(users):     for user in users:
print(f"Username: {user['username']}, Score: {user['score']}, Feedback:
{user['feedback']}")
```

- **How it works:** This function iterates over a list of users and prints out their username, score, and feedback.

3. Search for Users

Admins can search for a specific user by their username. If the user is found, their details are displayed.

Code:

```
def search_user(username, users):
for user in users:
        if user['username'] == username:             return f"User:
{user['username']}, Score: {user['score']}, Feedback:
{user['feedback']}"
return "User not found"
```

- **How it works:** This function searches the list of users for a matching username. If found, it returns the user's details; otherwise, it returns a "User not found" message.

4. View Feedback

Admins can view the feedback submitted by users.

Code: def

```
view_feedback(feedback_list):
for feedback in feedback_list:
print(feedback)
```

- **How it works:** The feedback is stored in a list and displayed by iterating over each feedback entry and printing it.

5. Delete Users

Admins can delete registered users from the system, which removes their account and associated data.

Code:

```
def delete_user(username, users):
for user in users:         if
user['username'] == username:
            users.remove(user)             return f"User
{username} deleted successfully"     return "User not
found"
```

- **How it works:** This function searches for the user by their username and removes them from the list of users if found. It returns a success message or an error message if the user is not found.

User Functionalities & Features:

6. User Registration

Users can create an account by selecting a unique username and password. The system checks if the username is already taken.

Code:

```
def user_registration(username, password,
users):     for user in users:         if
user['username'] == username:
            return "Username already taken"     users.append({'username': username,
'password': password, 'score': 0, 'feedback': ''})     return "Registration Successful"
```

- **How it works:** This function checks if the username already exists in the list of users. If not, it adds a new user to the list with an initial score of 0 and no feedback.

7. User Login

Users can log in using their credentials. If the login is successful, they gain access to the game. **Code:**

```
def user_login(username, password, users):     for user in users:         if
user['username'] == username and user['password'] == password:
            return "Login Successful"
    return "Invalid Username or Password"
```

- **How it works:** The function checks if the provided username and password match an existing user's credentials. If they match, login is successful; otherwise, it returns an error message.

8. Play the Game

Users can select a difficulty level and play the game. Their score is tracked based on their guesses.

Code:
```
def play_game(username, difficulty,
users):     score = 0
    # Game logic based on difficulty
    # Example: Increment score for correct answers
if difficulty == "easy":
        score += 10     elif
difficulty == "medium":
        score += 20     elif
difficulty == "hard":
        score += 30     for user in users:
if user['username'] == username:
            user['score'] += score
return score
```

- **How it works:** The game logic assigns a score based on the difficulty level chosen by the user. The score is added to the user's total score in the users list.

9. View Scores

Users can view their current and past scores. Scores are accumulated over multiple rounds.

Code:
```
def view_scores(username, users):
for user in users:         if
user['username'] == username:
return f"Score: {user['score']}"
return "User not found"
```

- **How it works:** This function retrieves and displays the current total score for a given user.

10. Send Feedback

Users can provide feedback on their game experience. The feedback is saved in their profile.

Code:
```
def send_feedback(username, feedback,
users):     for user in users:         if
user['username'] == username:
```

```
user['feedback'] = feedback            return
"Feedback Submitted"     return "User not
found"
```

- **How it works:** This function allows users to submit feedback, which is stored in the user's profile under the "feedback" field.

11. Update Password

Users can change their password by providing their current password and a new one. If the current password is correct, it updates the password.

Code:
```
def update_password(username, current_password, new_password,
users):     for user in users:         if user['username'] == username and
user['password'] == current_password:
            user['password'] = new_password
return "Password Updated"
   return "Invalid Current Password"
```

- **How it works:** The function checks if the entered current password matches the user's existing password. If correct, the password is updated; otherwise, it returns an error message.

These functionalities and features ensure that users can engage fully with the game, track their progress, and contribute to system improvements through feedback.

5. File Handling and Data Persistence

The game stores and retrieves data from several files to ensure persistence:

- **users.txt**: Stores the registered users' credentials (username and password).
- **leaderboard.txt**: Stores user scores across different gameplay sessions, allowing the leaderboard to reflect updated scores.
- **feedback.txt**: Stores the feedback submitted by users regarding their experience with the game.

These files are loaded and saved at appropriate points in the program, ensuring that user and game data are maintained across sessions. This design allows the game to function smoothly without losing data after the program terminates.

6. Code Structure and Design

1. Functions and Modularization:

The code is organized into distinct functions that handle different aspects of the game:

- **Game Logic**: Functions like play_game(), clear_terminal(), and view_leaderboard() handle the main gameplay loop, clearing the terminal, and displaying the leaderboard.
- **File Operations**: Functions like load_user_data() and save_user_data() manage reading and writing user information to files.
- **Admin and User Management**: Admin functions such as admin_login(), view_all_users(), and search_user() handle the management of users and their data.
- **Feedback Management**: Functions like send_feedback() and load_feedback() allow users to submit feedback and for the admin to view it.

This modular approach enhances the maintainability and readability of the code, making it easy to add or modify features in the future.

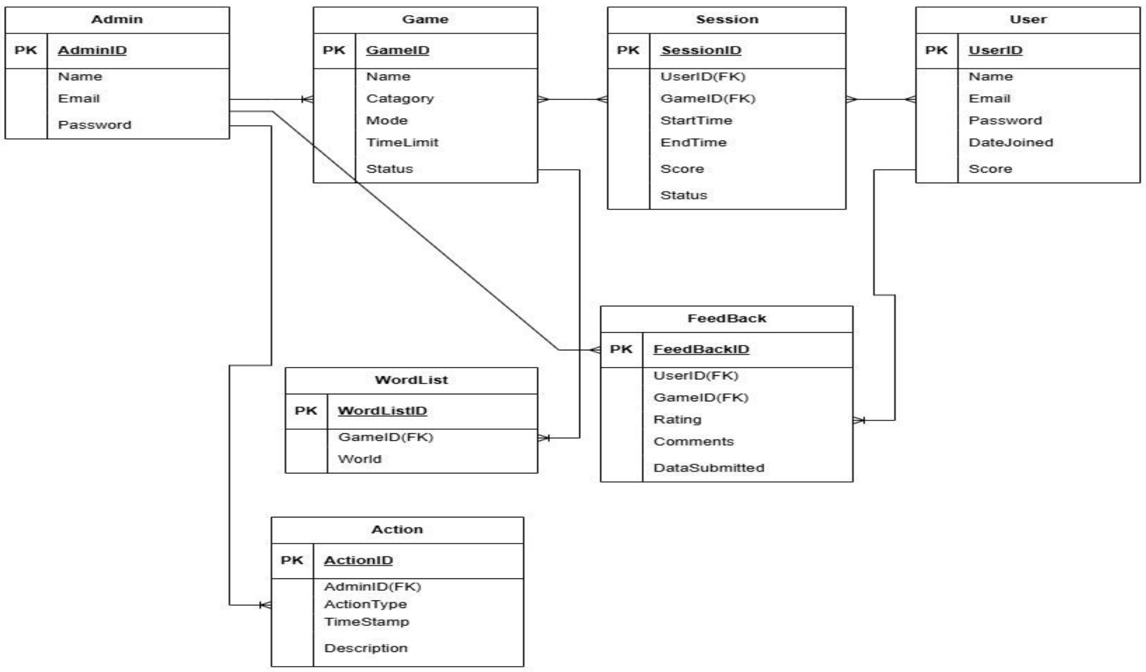
2. Error Handling and Validation:

The program checks for common errors, such as:

- Incorrect admin or user login credentials.
- Invalid mode selection during gameplay.
- Attempted actions on empty lists (e.g., no registered users or scores).

While the script handles some basic errors, further enhancements can be made by adding more robust validation for input and additional exception handling for file operations.

7. Potential Enhancements



1. Security Improvements:

- **Password Encryption**: Currently, passwords are stored in plain text, which is not secure. Using libraries such as bcrypt or hashlib to encrypt passwords would significantly improve security.

2. User Interface Improvements:

- The user interface is currently text-based and could be enhanced by implementing a graphical user interface (GUI) using a library like Tkinter or PyQt to make the game more visually appealing.

3. Database Integration:

- For better scalability and performance, integrating a database (e.g., SQLite, MySQL) to store user data, scores, and feedback would allow for more complex queries and faster data retrieval.

4. Enhanced Game Features:

- Adding more categories or randomizing word difficulty could increase replay ability.
- A progress system where users can level up or unlock achievements would further engage players.

8. Conclusion

This Python-based Throught Sprint Game successfully integrates gameplay with user and admin functionalities, offering an engaging experience for users while providing necessary administrative controls. The game's core features memorization, guessing, and scoring are simple yet effective, and the additional functionality for managing users, viewing feedback, and maintaining data persistently adds depth to the system.

While the code serves its purpose well, there is room for improvement, particularly in terms of security, scalability, and user interface design. With these enhancements, the game could evolve into a more robust, interactive, and secure platform for users to enjoy.