C++ Programming Project – Hangman Game

Student 1: Urs Nicolas Robert
Student 2: Rares Taut

1. Project Overview

This project implements a text-based Hangman game in C++, split between two separate applications that communicate via shared files. The design emphasizes modular class-based architecture, cross-platform compatibility, and a command-line interface.

2. Team Responsibilities

Student 1: Urs Nicolas Robert

- Developed core game engine functionality
- Handled word management: loading, adding, and displaying words
- Managed game state: guesses, attempts, and word masking
- Implemented command-line argument processing

Student 2: Rares Taut

- Designed user interface and game flow
- Implemented leaderboard functionality
- Developed game history tracking system
- Enhanced terminal display with color formatting

3. Key Data Structures

- **GameWord**
  - `std::string word` – The actual word to guess
  - `std::string maskedWord` – The masked version (e.g., _a_e)
  - Methods for updating and checking letter guesses
- **Player**

  - `std::string name` – Player's name
  - `int score` – Player's score
  - Methods to update and retrieve player stats

- **GameHistory**
  - `std::vector<std::string> history` – List of recorded game sessions
  - Methods to append and retrieve game history entries

- **Leaderboard**
  - `std::vector<Player>` – List of players with scores
  - Methods to update and sort the leaderboard

- **WordManager**
  - `std::vector<std::string> words` – Collection of valid words
  - Methods for loading, saving, and managing words

- **HangmanGame**
  - Composes instances of all other core classes
  - Controls the main game loop and logic

4. File Structure

`words.txt`

Stores the pool of available words:

```
apple
banana
computer
programming
algorithm
...
```

`leaderboard.txt`

Stores player scores in descending order:

```
Player1 150
Player2 125
Player3 110
...
```

`history.txt`

Logs each game session:

```
Player: Player1 | Word: banana | Score: 50 | Result: Win | Date: Mon May 20
Player: Player2 | Word: computer | Score: 0 | Result: Loss | Date: Mon May
...
```

5. Command-Line Interface

The application can be launched with various command-line arguments for non-interactive use:

- `view_words` – Display all available words
- `add_word <word>` – Add a new word to the list
- `view_leaderboard` – Display the current leaderboard
- `view_history` – Display game history log

If no arguments are provided, the program enters interactive mode, displaying a menu to the user.

6. Implementation Highlights

- Designed using modular C++ classes with strong separation of concerns
- Employs composition (e.g., `HangmanGame` composes `Player`, `GameWord`, etc.)
- Utilizes file I/O for persistent state management
- Color-coded terminal output improves the user experience
- Cross-platform compatibility (tested on both Windows and Unix-like systems)
- Command-line support allows flexible operation and scripting