Programming Project / C++ - Hangman Game

Student 1: Fodor Robert Mihai

Student 2:

I. Task Description

Student 1 is responsible for core game mechanics and logic:

- Managing word list and word selection
- Tracking guessed letters and game state
- Managing game outcomes and scoring
- Handling file I/O for word bank and saved games

Student 2 is responsible for user interaction and game interface:

- Reading command-line arguments to control the game
- Displaying game progress (masked word, guessed letters)
- Implementing save/load game options
- Error handling for invalid inputs or commands

II. Data Structures Used by the Team

- `Word` (class): `std::string content` - Represents the word to be guessed
- `HangmanGame` (class):
  - Fields: `Word word`, `std::set<char> guessedLetters`, `int incorrectTries`, `int maxTries`
  - Methods: `bool guessLetter(char)`, `std::string getMaskedWord()`, `bool isGameOver()`, `bool isWin()`
- `GameState` (struct or class): Stores current state to be saved/loaded from file

Relationships:

- `HangmanGame` uses a `Word` (composition)
- `HangmanGame` and `GameState` have mutual interaction (association)

III. File Structure

- **words.txt** - Word bank for random selection:

<word1>\n<word2>\n<word3>\n

- **savegame.txt** - Saved game state:

```
<word>\n<incorrectTries>\n<guessedLetter1> <guessedLetter2> ...
```

IV. Commands Implemented by the Apps

Application 1 - Game Logic & Management:

```
./game start <difficulty>
./game guess <letter>
./game status
./game save <filename>
./game load <filename>
```

Application 2 - Word Bank & Tools:

```
./tools add_word <word>
./tools remove_word <word>
./tools list_words
```

Note: All input is passed via command-line arguments only (no use of std::cin). Output is printed to the console.

V. Summary

This Hangman game project follows a modular architecture using C++ OOP features such as composition and association.

It uses persistent file storage for game state and a shared word bank, and includes separate responsibilities for logic and user interaction.

All communication is done via command-line arguments, making it suitable for automation and testing.