Hangman

You have to implement a console-based variation of the classical Hangman game. The computer will select a **Sentence** that the user can attempt to guess, letter by letter. Each time the user guesses a correct letter, the computer will fill it in the sentence at the correct positions. In case the letter does not appear, the computer will fill in a new letter in the word "hangman", starting from the empty string. The game ends when the user has guessed the sentence (user wins) or when the computer fills in the "hangman" word (user loses). Program functionality is broken down as follows:

1. Add a sentence. While not in a game, the user can add a sentence **[1p]**. Each sentence must consist of at least 1 word. Every word in the sentence must have at least 3 letters. There can be no duplicate sentences. **[1p].**

2. Start the game. When the user starts a game, the computer selects one of the available sentences and displays it on screen, hangman-style. This means that the computer reveals the first and last letter of every word, as well as all the apparitions of these letters within the words **[2p]**. The sentence selection is random **[1p]**.
   e.g. for the sentence "**anna has apples**", the computer reveals "**a _ _ a has a _ _ _ _ s**".

3. Play the game. The game consists of several rounds. In each round, the user proposes a letter. If the sentence contains the letter, the computer reveals where these letters appear within the sentence. If the sentence does not contain the letter, or the user previously proposed the letter the computer will add a new letter to the word "hangman", which is displayed to the user **[3p].**

4. Game over. The game is over when the sentence is correctly filled in (user wins), or when the computer fills in the word "hangman" (user loses). **[1p]**

Non-functional requirements:

- Implement a layered architecture solution with Repository, Controller and UI.
- Functionalities without tests or specification in the Repository or Controller layers will be graded at 50% value.

Observations!

- Sentences are loaded from/saved to a text-file that must initially hold at least 5 entries.

sentences.txt example:

```
anna has apples
patricia has pears
cars are fast
planes are quick
the quick brown fox jumps over the lazy dog
```

Gameplay example:

```
Output: "a _ _ a has a _ _ _ _ s" - ""
User guess: "m", output changes to: "a _ _ a has a _ _ _ _ s" - "h"
User guess: "n", output changes to: "anna has a _ _ _ _ s" - "h"
User guess: "m", output changes to: "anna has a _ _ _ _ s" - "ha"
User guess: "e", output changes to: "anna has a _ _ _ es" - "ha"
User guess: "x", output changes to: "anna has a _ _ _ es" - "han"
User guess: "t", output changes to: "anna has a _ _ _ es" - "hang"
User guess: "p", output changes to: "anna has app _ es" - "hang"
User guess: "l", output changes to: "anna has apples" - "YOU WON!"
```