**Manual of code**

**Lab 01 Mini projects**

**1. The Hangman Art**

The game uses a list called hangman to store ASCII art representations of the hangman figure at different stages. Each stage corresponds to the number of incorrect guesses made by the player. For example:

* Stage 0: No incorrect guesses (just the gallows).
* Stage 1: One incorrect guess (head is drawn).
* Stage 2: Two incorrect guesses (body is drawn).
* And so on, until the final stage where the hangman is fully drawn.

This visual feedback makes the game more engaging and helps the player track their progress.

**2. The Hangman Class**

The core logic of the game is encapsulated in the Hangman class. Here's what each part of the class does:

**a. Initialization (\_\_init\_\_ method):**

* **self.word:** The word to be guessed, converted to lowercase for case-insensitive comparison.
* **self.guessed\_letters:** A list to store all the letters the player has guessed so far.
* **self.incorrect\_guesses:** A counter to track the number of incorrect guesses.
* **self.max\_incorrect\_guesses:** The maximum number of incorrect guesses allowed, which is determined by the length of the hangman list minus one (since the list starts at stage 0).
* **self.display\_word:** A list that represents the current state of the word being guessed. Initially, it contains underscores (\_) for each letter in the word. As the player guesses correctly, the underscores are replaced with the correct letters.

**b. Displaying the Game State (display\_game\_state method):**

* This method prints:
  1. The current stage of the hangman figure (based on the number of incorrect guesses).
  2. The current state of the word being guessed (with guessed letters revealed and unguessed letters as underscores).
  3. The number of incorrect guesses remaining.
  4. The list of letters the player has guessed so far.

This provides the player with all the information they need to make their next guess.

**c. Making a Guess (make\_guess method):**

* This method processes the player's guess:
  1. Converts the guess to lowercase for case-insensitive comparison.
  2. Checks if the letter has already been guessed. If so, it informs the player and does not count the guess.
  3. If the letter is in the word, it updates the display\_word list to reveal all instances of the letter.
  4. If the letter is not in the word, it increments the incorrect\_guesses counter.
  5. Adds the guessed letter to the guessed\_letters list.
  6. Returns True if the guess was valid, or False if the letter was already guessed.

**d. Checking if the Word is Complete (word\_complete method):**

* This method checks if all letters in the word have been guessed by checking if there are no more underscores (\_) in the display\_word list.

**e. Checking if the Game is Over (game\_over method):**

* This method checks if the game is over by determining if:
  1. The player has made too many incorrect guesses (self.incorrect\_guesses >= self.max\_incorrect\_guesses).
  2. The player has successfully guessed the entire word (self.word\_complete()).

**3. The main Function**

The main function is the entry point of the game. Here's how it works:

1. **Word Selection:**
   * A list of words (word\_list) is defined. The game randomly selects one word from this list using random.choice().
2. **Game Initialization:**
   * An instance of the Hangman class is created with the selected word.
3. **Game Loop:**
   * The game runs in a loop until the game is over (either the player wins or loses).
   * In each iteration of the loop:
     + The current state of the game is displayed using display\_game\_state().
     + The player is prompted to guess a letter.
     + The input is validated to ensure it is a single alphabetic character.
     + The guess is processed using make\_guess().
4. **Game Over:**
   * After the loop ends, the game checks if the player won or lost:
     + If the word is complete, the player wins, and a congratulatory message is displayed.
     + If the player ran out of guesses, the game reveals the word and informs the player they lost.

**4. Example Gameplay**

Here’s an example of how the game might play out:

1. The game selects the word "python".
2. The player sees:

-----

| |

|

|

|

|

Current word: \_ \_ \_ \_ \_ \_

Incorrect guesses left: 6

Guessed letters:

1. The player guesses "p":

Correct guess! 'p' is in the word.

Current word: p \_ \_ \_ \_ \_

1. The player guesses "z":

Wrong guess! 'z' is not in the word.

The hangman figure updates to show the head.

1. The game continues until the player either guesses the word or runs out of guesses.