Coding Standards

Naming Conventions:

- Class names should follow the CapsWords convention (ex. 'Hangman')
- Use names that accurately explain the purpose of both functions and variables
- Avoid excessively long names (>24 characters)
- Variable names should be lowercase, with underscores separating words (ex. word_to_guess)
- Constants should be in uppercase with underscores separating words (ex. 'WORDS')

Style and Spacing:

- Use a two-space tab standard indentation
- Use spacing between functions and lines of code when necessary to improve readability, but avoid excess spacing
- Use comments to explain the purpose of functions and other lines of complex code when necessary
    - Inline comments:
        - def get_word(): # function to get random word
    - Other comments:
        - def get_word():
function to get random word

Function Standards:

- Functions should have clear and singular purposes
- Avoid overly complex logic within the functions – make helper functions if needed to make it easier for others to follow along
- Ensure consistent error handling, such as input validation

Use of Global Variables:

- Minimize the use of global variables – only use them if/when necessary

Making Pull Requests:

- Include clear and concise descriptions of the changes made in the pull request
- Reference relevant issues or give context for the changes made
- Ensure that all tests are passed, and no new issues are introduced

Reviewing Pull Requests:

- Provide constructive feedback with an emphasis on clarity, readability, and adherence to coding standards
- Confirm that the changes made pass all tests and no new bugs are introduced
- Check for potential improvements or optimizations that could be made
- Ensure the code aligns with the project's overall structure and goals

Testing Requirements:

- Include comprehensive test coverage for new features and modifications
- Ensure tests cover all scenarios, such as valid inputs and edge cases

- Document testing procedures and requirements for other contributors
- Run tests regularly to catch errors early on

Tests:

1. Custom Word Support:

- Test 1: Enter a custom word and verify that the game starts with the provided word.
  - Steps: Choose the custom word mode, enter a word, and start the game.
  - Expected Result: The game should start with the entered custom word.
- Test 2: Enter multiple custom words and verify randomness.
  - Steps: Choose custom word mode, enter different words multiple times, and start the game.
  - Expected Result: Each time, the game should start with one of the entered custom words randomly.

2. Difficulty Levels:

- Test 1: Select different difficulty levels and verify the number of guesses.
  - Steps: Choose different difficulty levels and start the game.
  - Expected Result: The number of guesses allowed should vary according to the selected difficulty level.
- Test 2: Check if the difficulty level affects word selection.
  - Steps: Choose different difficulty levels multiple times and observe word complexity.
  - Expected Result: Higher difficulty levels should result in more complex words.

3. Theme Customization:

- Test 1: Change the theme and verify the visual changes.
  - Steps: Change the theme setting and start the game.
  - Expected Result: Visual elements (if any) should reflect the chosen theme.
- Test 2: Verify that theme customization doesn't affect gameplay.
  - Steps: Play the game with different themes and observe gameplay.
  - Expected Result: Gameplay should remain consistent across different themes.

4. Multiplayer Support:

- Test 1: Start a multiplayer game and verify connectivity.
  - Steps: Start a multiplayer game on two separate instances/devices.
  - Expected Result: Both instances should be able to communicate and play the game.
- Test 2: Check if multiplayer functionality handles simultaneous actions.
  - Steps: Both players make guesses simultaneously and observe game behavior.
  - Expected Result: Game should handle simultaneous actions gracefully without crashing.

5. GUI Implementation

- Test 1: Verify GUI elements and interaction.
  - Steps: Interact with GUI elements (buttons, input fields) and start the game.
  - Expected Result: GUI elements should respond to user interactions, and game functionality should be accessible.
- Test 2: Ensure consistency between GUI and console versions.
  - Steps: Compare gameplay and features between GUI and console versions.
  - Expected Result: Both versions should offer similar gameplay experience and features.

For each test, contributors should document the steps taken, the observed behavior, and whether the observed behavior matches the expected results. Additionally, they should ensure that test coverage is comprehensive, covering all possible scenarios and edge cases for each feature.