My Modified Hangman is set up rather clearly. However, I highly recommend reading the comments in the code prior to modifying my game, as there are certain functions that are called in several places in the program, and editing them without understanding what they do and why may result in runtime errors.

My program can be looked at in three sections: interaction with, and extraction of information from, the English dictionary database (dictionary.db), functions and single lines of code that set-up the GUI (graphical user interface), and functions and single lines of code that allow for user interaction with the GUI.

The data in dictionary.db was obtained on sourceforge (see Project_Report.pdf), and organized to fit the uses of this game. It contains one table, called eng_dict, which, in turn, contains three columns: word, pos, and def. Pos is part of speech, and def is the definition of the word in that row. Words may have multiple definitions, all of which are included in the same cell in the table. While part of speech is not used in Modified Hangman in its current form, an interesting expansion of the game would be to allow the player to restrict eligible game words by part of speech.

In Modified Hangman, interaction with the database occurs early in the game. After the player inputs a search word, if that word is found in any of the definitions in the dictionary database, the words corresponding to those definitions form a set of eligible game words, of which one is randomly selected. If no match is found, a word to be guessed by the player is randomly selected (see Tutorial.pdf for how to play, as well as the prompts that appear as you play). If the selected word doesn't contain letters, a new word is selected.

Now that the game word is selected, interaction with the database ends, and the build-up of the GUI begins. The length of the game word is obtained in order to set up the correct number of empty spaces (represented as underscores), one for each letter, to be replaced with correctly-guessed letters.

When the player submits a letter guess, the game word is checked, character by character, for matches. Every time there is a match, the position of the match in the game word in added to a list of indices. This list is where the program gets its information about which letters to reveal when the edit_letter_space function is called. For incorrect guesses, a counter and an empty list are set up. Every time there is an incorrect guess, the counter goes up by one, and the incorrect guess is added to the list. These changes are also reflected in the GUI when the edit_letter_space function is called.

Depending on whether the player picks the easy difficulty (10 tries) or the hard difficulty (5 tries), he is allowed the corresponding number of incorrect guesses before a "Game Over!" message is displayed. It is of key importance to note here that special characters are allowed to be submitted as letter guesses. Additionally, the player is allowed to repeat incorrect guesses, and that counts as another mistake. Finally, blank guesses are also registered as incorrect. These behaviors are intentional, forcing the player to pay closer attention. A limitation that IS placed on letter guesses, is that no more than one character may be guessed at once. This is done in order to prevent cheating, as otherwise the player would be able to check multiple letters for matches at the cost of one try.

The limiting of letter guess entries to no more than one character is accomplished through a class and a sub-class. The ValidatingEntry class sets up the frameworks for a variety of entry box restrictions, and the MaxLengthEntry class is the specific implementation for limiting entry length. Each entry box to be limited is declared as a MaxLengthEntry object, with a maximum length given as one of the arguments (e.g. maxlength = 1).

All of the warning messages in Modified Hangman are presented to the user in new window, created using Toplevel, which allows for creation of additional windows besides the master (root) window. These messages include the reminder to enter a search word, the game won screen and the game over screen. The first screen that the player encounters upon running the game, which I have called the welcome screen, is also a Toplevel window, set up using both widgets and Canvas objects. This welcome screen can be displayed at any point by pressing the “How to Play” button on the bottom right in the main game screen, and it is the only time when the main window is minimized upon display of a different window. For the previous three Toplevel windows described, I have used the grab_set() and grab_release() methods, to restrict the potential for interaction to only the topmost window displayed (the main window remains in the background). When the message window on top of the main window is dismissed, the main window is re-enabled, as grab_release() is called on the now dismissed window, at this time.

The Hangman images in my game are each assigned to a PhotoImage object, and are then displayed as Canvas objects using the create_image() method on the correct image at the correct time. This is also tied to the incorrect letter count, and the image changes to the next step (a body part is added) each time an incorrect guess is registered. All of the images used in this game are in the .gif format, as this was one of only a few image formats that could be used without bringing other Python modules into the picture, which was something I aimed for. A drawback of converting the original .png images to the .gif format is that the supported range of colors is more limited. However, I was satisfied with how the images turned out, and I think that they stylistically fit this game, considering that it is based on an old classic, Hangman.

Once the player wins or loses, there is an option to play again, without having to close and re-open the program. As there is no simple platform-independent way to make a Python program restart itself, I elected to manually reset all of the GUI elements in the game in my clear_setup function. Notice that in order to reset a list, you must use list_name[:] = [], not list_name[] = [], as the latter doesn’t reset any of the elements of the list. The former does.

If you have any more questions or concerns, please leave me a comment on my github repository at:

https://github.com/darknessomega/Hangman (though you must be there already if you’re reading this file.

Thank you for reading! Enjoy!