Background

In classic Hangman, one person thinks of a word that she wants the player to guess, draws the appropriate number of spaces, one for each letter, and allows the player to begin guessing. For each incorrect letter guessed, a body part is drawn on a hangman stand. If the player completes the word before the stick figure is complete she wins, otherwise, she loses.

This project is a modified version of this well-know game. At the beginning of the game, the player is asked to input a word, which will be referred to as the “search word” moving forward. Next, all of the words that have the search word in their definitions will be found, and one of these words will be randomly selected to be the “game word”. If it so happens that the entered search word is not found in any of the definitions of the words in the dictionary, a completely random word is selected from the dictionary, to serve as the game word.

Not only do these modifications make for a more complex and difficult game, but they also eliminate the need for a second player. In setting up random selection of the game word, according to a given criterion, I have made it possible to play without a friend.

Dictionary Database

In order to realize my vision for this game, the first thing I needed was a searchable dictionary. I found a MySQL English Dictionary Database, but it was incompatible with Sqlite 3. So, I downloaded the .txt version, and created the database myself using a C++ script that I wrote (for .txt file formatting), and Termsql for SQLite. TermSQL, a project by Tobimensch available on GitHub is a SQLite 3 tool for conversion of plain text to tables in a SQL database. This tool was a near-perfect way to create the database I needed. I say near-perfect, because Termsql allows for no more than one delimiter to be specified to separate data into different fields. However, the database text file was formatted as: Word (part of speech) Definition. Hence, my plan to use ‘(‘, ‘)’, and ‘.’ as delimiters would no longer work.

In order to deal with this limitation, I decided to write a C++ script to re-write the entire dictionary text file to have a ‘+’ between each area. Each entry would then be: Word+(part of speech)+Definition.+

Several hours later, my script was almost working, albeit slowly (the input file is 14.7 MB after all), but the ‘+’ at the end of each entry was missing and I couldn’t find a way to fix it. Next thing I knew, I was re-writing my whole script. After all that work, it hit me that my approach was wrong. I decided to add the ‘+’ signs in the right places in the input file, instead of creating a new file. This allowed me to avoid end of line issues I was having, though I’m not clear exactly what was different. I most likely ended up fixing an issue that I had with my script previously, upon re-writing it. Another several hours later, my script was working as I had wanted it to work. I was now able to use Termsql to create my SQLite 3-compatible database.

Graphical User Interface – Tkinter in Python

I now moved on to creating my graphical user interface using the Tkinter module in Python. I ended up creating many widgets - Windows, Frames, Buttons, Radiobuttons, Labels (text and image) and

Canvases, as well as multiple different kinds of Canvas objects. I selected the type of GUI object to use deliberately, in order to fit my desired use, while minimizing syntactical complexity. That being said, at times, I elected to take the more complicated route towards implementing a feature, in order for it to be more aesthetically pleasing (e.g. used Canvas instead of Label to eliminate text background in welcome screen).

Obstacles Faced and Overcome

Along the way, I encountered a multitude of issues that I had to fix in order to produce a fully and correctly working game. First, I needed to prevent the player from submitting a blank search word, as allowing for this to be done resulted in the game crashing. I fixed this problem by showing a warning when the player tried to submit a blank search criterion. All of this occurred before I implemented the completely random word selection feature, which returned a random game word if the search word didn't match any of the words in the definitions in the dictionary. Had this latter behavior already been available, the crashing would have likely been avoiding. However, nonetheless, disallowing submission of a blank search criterion was one of the few input restrictions, was one of the few user entry restrictions that I DID want to implement, right from the get go.

The other entry limitation that I wanted to put in place, and did, was preventing the player from submitting more than one character as a letter guess. This was important so that the player wouldn't be able to cheat by guessing two letters at the cost of one attempt. I elected to allow for submission of blank letter guesses, special character guesses, and repeated correct and incorrect guesses, in order to make my Modified Hangman game more challenging. If the player wanted to do well, she would have to pay careful attention to the guesses she submits. My game makes this easier than it may sound at first, as incorrect guesses are displayed on the game screen, for the player's reference.

In order to implement the one-character guess entry restriction, I created a special entry box object based on a ValidatingEntry class, which serves as the foundation for several entry box restrictions, and one of its sub-classes, MaxLengthEntry, which specifically allows for the maximum allowed entry length to be specified upon creation of the Entry Box.

Aside from the aforementioned difficulties that I faced along the way, most of the remaining complications were a result of my limited knowledge of certain intricacies of Python 2.7. As I gained experience, I became less and less bothered by these "issues", but they gave me a fair amount of trouble overall. String and int objects are immutable in Python, so I had to use lists, most of which contained only one or two items, in order to gain the ability to update certain values. For GUI elements, on the other hand, I was able to use "var" objects (e.g. StringVar()), which are mutable. While I now somewhat understand the reasoning behind making objects of basic types immutable, avoiding the resulting issues was more troublesome than I would have liked.

An interpretation of how Python is designed is that rather than creating variables with or without names (as in C/C++), you assign names to values. This distinction makes it clear why all objects are passed by reference in Python. When you pass by reference, the address to the object is passed in and out of functions, rather than the value or values contained in the objects. If you're a C/C++ programmer, the option to pass objects by reference is a godsend. However, because Python doesn't have pass by value, pass by reference can’t quite be taken advantage of in the same ways. For example, other ways have to be found to return multiple values from a function (e.g. returning a list, returning a dictionary, etc.). Because storing multiple values in a list and returning that list is quite simple, this difference in style was not more than a minor annoyance.

Goals – Old and New

My initial goals when I started coding my game were to:

- Set-up an intuitive graphical user interface (GUI) that allows the user to play the game
- Interact with an English dictionary database to feed words and definitions for the game. Also to be used to search for eligible words given the user input
- Ensure that the game does justice to the original *Hangman* by containing all the components of the original game (with the additions that were mentioned)

- Set-up an intuitive graphical user interface (GUI) that allows the user to play the game
- Interact with an English dictionary database to feed words and definitions for the game. Also to be used to search for eligible words given the user input
- Ensure that the game does justice to the original *Hangman* by containing all the components of the original game (with the additions that were mentioned)

However, as I went along, I realized that certain other features were a necessity, and still others would be interesting to have. As such, my list of stretch goals grew from a couple of items to a list about as long as that above. Giving in to a certain level of ambition, I ended up adding:

- Other GUI windows
- Background images to certain windows
- A simple logo that I made
- Game over behavior
- Game won behavior
- The ability to play again without having to close the program and open it again
- An option to not use the clue
- Another harder difficulty
- Etc.

One final note that I would like to make is that Toplevel was likely the most useful capability of tkinter that I discovered while working on my project. Being able to create different windows for different parts of my game, such as one for the main game screen and one for the welcome screen, without having to jump through a plethora of syntactical hoops, allowed me to avoid having to delete widgets and replace widgets. It is much more convenient to allowed to just hide widgets and make them reappear, and thanks to Toplevel, that was achievable.

I would like to extend a big thank you to:

- x16bkkamz6rkb78 on Sourceforge for his MySQL English Dictionary files
  - https://sourceforge.net/projects/mysqlenglishdictionary/
- Tobimensch for Termsql –
  - https://github.com/tobimensch/termsql
- Fredrik Lundh for the ValidatingEntry and MaxLengthEntry Classes, as well as his other pages available for reference –
  - http://effbot.org/zone/tkinter-entry-validate.htm
- Espeon200 for Hangman Images –
  - http://media.photobucket.com/user/Espeon200/media/hangman/stage1.png.html?filters[term]=hangman&filters[primary]=images&filters[secondary]=videos&sort=1&o=11
- PlanWallpaper for Welcome screen Background image –
  - http://www.planwallpaper.com/static/images/518079-background-hd.jpg
- Rachel Gallen on StackOverflow, whose answer to a question helped me figure out how to automatically center my Tkinter windows
  - http://stackoverflow.com/questions/14910858/how-to-specify-where-a-tkinter-window-opens
- The StackOverflow community for asking and answering so many questions on a publicly-accessible website