**Trexquant Hangman Challenge**

***Algorithm Explanation***

***By: - Isha Choudhary |*** [***shahooda637@gmail.com***](mailto:shahooda637@gmail.com)

This document explains the approach followed by me to play the Hangman Word guessing game in python. The explanation would be confined to the ‘guess()’ method of the complete code only. Each thought for formulating the algorithm is drafted in the points to maintain the good flow and understanding of my way of thinking in deriving this approach.

1. Some of the things which are known to us before formulating the algorithm are,
2. The dictionary contains 25000 words, and the game will be throwing us the words from these 25000 words only.
3. We have only 26 letters which combine in different combinations to create these 25000 words. We need to make guesses from these 26 letters only.
4. There are 5 vowels (a, e, i, o, u) and 21 consonants in the set of alphabets.
5. After doing some data analysis on the dictionary, I found that out of 25000 words, 99.45% of words contain vowels, and only 0.55% words are spelled without using any of the vowels.

A black text on a white background

Description automatically generated

1. The most common vowel used in these spellings is ‘e’ followed by ‘a’ , ‘i’, ‘o’, and ‘u’ . The visualization for the same is as follows:

A graph of blue rectangular bars

Description automatically generated

1. After plotting the graph for calculating the percentage of a certain word in the dictionary consisting of vowels, we get a graph as follows. The highest bar around 0.5 and concentration of ration in the range 0.3 to 0.55 shows that guessing based on the vowels would be a bad idea if the ration is greater than 0.55.

The reason for this is that guessing using vowels in such words would not narrow down the possible correct words and can lead us toward complexity in guessing the next correct letter to win the Hangman game, as we have limited tries to guess the spelling right.

1. Next observation of the words in the dictionary is that the length of the longest word in the dictionary is equal to 29, while the smallest word in the dictionary contains 1 letter.

***Now, coming to the main algorithm to guess the letters. The ‘guess()’ method.***

1. The first step is to clean the word and remove the spaces if exists. As, we only need letters in the spelling.
2. The next step to decide which letter to guess, using the dictionary we have is to filter all the words from the dictionary with similar length. To get the length of the word we need to guess, we must get the length of the word.

For this, I used the **len()**  function.

However, this is not the starting point of the algorithm, but just the first thought on how to start the guessing procedure. The new dictionary is created by appending the suitable words from the main dictionary into an empty dictionary, named as **‘new\_dictionary’** in my code.

The “for loop” is created to iterate through each word in the dictionary and append the words with similar length to the word we need to guess into the new dictionary. This helps us narrow down the database for us to increase the efficiency of our search.

1. Next is to count the frequency of occurrences of letters in the new filtered dictionary. The first guess could be the letter occurring the maximum number of times in all the words of similar length. The function **letter\_count()**  does the task.
2. The crux at this point was the presence of vowels and consonants. To check the ration of consonants present in the words, I used the function **vowel\_ratio()**. This function takes the word, counts the vowels in the word and divides the total number of vowels in the word to the length of the word. If this is greater than 0.55, guessing with the vowel is a bad idea.

This idea of guessing the letter is the first way my algorithm guesses the letter.

At the beginning of the algorithm, we have the filtered dictionary, and the letters with the number of times they appear in the spellings of all these words. The algorithm iterates through each letter, (in descending order of their appearance in spellings), check for the vowel ratio, if it is greater than 0.55, the vowel is ignored and the loop continues, otherwise the guess is made with that letter and loop breaks.

1. If this way of guessing the letter fails, the algorithm moves to the next method, which creates a filtered dictionary by using the clean\_word, i.e., the word we need to guess, and the **n\_word\_dictionary**.

Here, the **n\_word\_dictionary** is a dictionary where each key represents a word length, and the associated value is a list of all possible substrings of that length extracted from the words in our actual list of 25000 words.

Using this dictionary, we find the words matching the word we need to guess and add all the matches into a new list called **new\_dictionary**.

Then we count the letters frequency and make the guess with most occurring letter among these words. Here also we are checking for vowel ratio to be > 0.55 to ensure that we are narrowing down the list of probable answers significantly with each try and win the game.

1. The next method to make the guess takes the ‘1/2’ of the length of the word to be guessed as an integer, and then gets the substring of words from the clean\_word which is of half of the length of the clean\_word.

All the words of length ½ of the given word are filtered into a new dictionary and a guess is made with the letter appearing the most in these words.

The same method is again used but this time for length 1/3 of the given word.

1. At the end, we have a default fall back method if none of the above method works well for us. This one refers to the main dictionary to find similar words and get the letter appearing most. We use this letter as our next guess.

**All in all, the algorithm is based on match and rematch techniques where, we try to narrow down the list of words which might be the answer and make the guess with the letter occurring the most of the times in these words.**

**The filtering is done on the basis of the length of the word, and the placement of letters guessed correctly if any.**