Assumptions:

1. Almost every word contains a vowel. Since, number of vowels are less, so that helps to narrow down the possibilities.
2. All the alphabets are not equally likely. Words that occur more often in the dictionary should be given priority.
3. The above priority sequence is not unique. i.e. the priority sequence depends on the length of the word. So, the sequence is different for words of different word length.

Trade Offs:

1. Considering the size of data frequency distribution does not generalise well.
2. Not able to consider the bigrams due to lesser number of trials also leads to low accuracy.
3. Due to use of frequency distribution and given number of chances (6 in our case), rare words are not guessed. This compromises with the accuracy.

The entire code is written in python2.7+.

Requirement to the program

1. Numpy
2. NLTK

Run the Program

1. Keep the words.txt in the same folder as hangman.py.
2. Go to cmd line and run the file as follows

   python hangman. py <mode> <word>

   Mode :
   - mode = 1 -> use unigrams for guessing and print the state at every step.
   - mode = 2-> use unigrams and bigrams for guessing and print the state at every step.
   - mode = 3 -> run on entire words.txt (unigrams only) and print the accuracy.

   Word :
   - This is to used only when giving a single word otherwise scripts uses the word.txt to print the percentage match.