Vellore Institute of Technology, Vellore

Winter Semester 2019-20

CSE2001: Data Structures and Algorithms

Title: Word-Play Dictionary

Instructor: Prof. Shalini L

Team – Members:

1. Manu Mrinal Singh, 18BCB0103
2. Nitin Ranjan, 18BCE0272
3. Glenn V. George, 19BCE2495

Abstract:

Education is the single most important sector in modern Economy both in terms of necessity and investment on a private scale. English, as a language is spoken by about 1.5 Billion people across the world (2018 stats) – virtually making it the Lingua Franca of a sixth of world's population. With the advent of virtual and digital dictionaries, people reduced their interest in knowing new words and rather relied on technology to do the same.

That is yet another reason why gamification becomes so important. Gamification allows and causes users to develop a healthy habit to learn new things. And so, as a project, we developed upon a popular word game called Hangman. It is a simple word-guessing game where the player knows the number of characters involved in a word that the second player provides and then has to randomly guess the word in a limited time-frame.

We also decided to work upon the hangman algorithm to find a more optimised programming and solution for the same. That resulted in some interesting set of observations. We found out a practical performance difference in different data-types and also found performance variation in different programming languages.

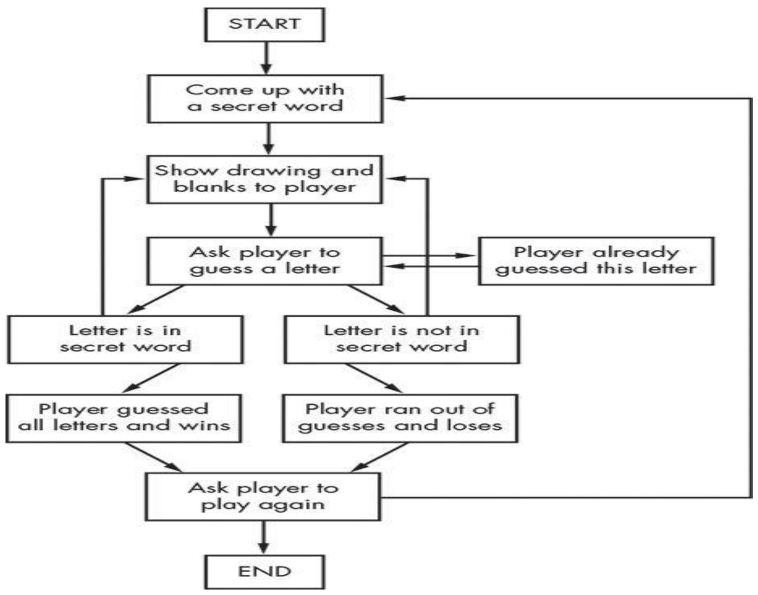
Keywords: Conditionals, python, lists, arrays, time complexity, gamification, C, programming, optimisation, dictionaries, data-set, alphabets, game-theory.

Introduction

The English Language has undergone fundamental changes in usage over the years. According to language experts, emoticons form the new age of language and communication. Oxford Dictionary awarded the 2015 word of the year title to the ‘tears of joy’ emoji. However, on a more formal scale, language has itself seen no change but the vocabulary of people, on average. The introduction of digital dictionaries in the late 2000s and later a reduction in internet access costs led to every person accessing the web. And while internet databases with sites like Google and Wikipedia have a way more diverse information set on every word, the overall result is that users have a reduced need to learn new words which does not serve them well in formal sectors or entrance examinations where it is expected of them to have a strong vocabulary.

So, one possible solution to the problem is gamifying the learning process. (*Gamification: the application of typical elements of game playing (e.g. point scoring, competition with others, rules of play) to other areas of activity.*)

Hangman is a popular word guessing game where the player attempts to build a missing word by guessing one letter at a time at the cost of saving oneself from the gallows. After a certain number of incorrect guesses, the game ends and the player loses and gets hanged. The game also ends if the player correctly identifies all the letters of the missing word. It’s a really fun game especially for one to improve his/her vocabulary. We replaced a player by the program that contains a database of all the words found in the dictionary in a text file. The game chooses a random word and the player has to guess the letters, the entire word or unscramble the jumbled words.


START
Come up with a secret word
Show drawing and blanks to player
Ask player to guess a letter
Player already guessed this letter
Letter is in secret word
Letter is not in secret word
Player guessed all letters and wins
Player ran out of guesses and loses
Ask player to play again
END

The game uses a generic tree data structure, to retrieve words from the text file, a list data structure is employed in python where a linear search is employed. We use linear search algorithm to compare the letters of the expected word to the letters of the word the player enters.

Also, with this we give the player the option of storing new words in a file. The player's wins and losses are also recorded by the program. The program records the words and characters that the user plays in a separate file. This helps to tabulate the most frequently used characters and words the user uses in the game.

What we further did was to test what would be the way in which words would randomly be constructed by the computer program and what is the average probability of the computer to predict the correct word in the game. And to further compare both the human guesses and words constructed by the computer.

Motivation:

In today’s world with ever growing use of abbreviations and broken English, most of us don’t know the proper spelling of many words, also the fact that the thought of using the internet to find the words that you don’t know and forgetting them later on brings the problem of not learning or improving your vocabulary. Also we can’ forget the fact it also quite boring and time consuming to actually search for a word though probably in the long run maybe useful but often ignore it and regret it later on.

Enter games. What most adults think that can worsen their child’s growth can now be thought of as a boon. There are many educational games out there and making many of the educational board games into video games or gamification of those topics is the right way to change and integrate into this new society of gamers. Here we decided to explore hangman and find a solution to find the most effective way of finding out the letters before our character gets hanged at the block. Also, we have a two-player game to unscramble a jumbled-up word (This is the general Hangman game).

Also, we wanted to compare the human guesses and computer guesses.

Related Works:

Theoretically speaking, game theory will be needed to optimise the strategy to ensure victory all the times. (Game Theory: Game theory is the study of mathematical models of strategic interaction among rational decision-makers)

However, here the purpose is to algorithmically enhance chances of victory. The game has been subject to extensive discussion in both discussion forums and research papers as it deals with the English language as a probability-based game and because the associated algorithms are also thus based on probability and mathematics.

For instance, it was found that while in English texts, the frequency of occurrence of first alphabet in a word are: ESIARN TOLCDU PMGHBY FVKWZX QJ

It is not the same for the frequency of occurrence in an English Dictionary, which is: ETAOIN SHRDLU CMFWYP VBGKQJ XZ

Research also shows that:

- There are only two words with one letter.
- There are no two letter words containing the letter C, Q, V or Z.
- From one to four letter words, the most popular letter is A. For five letter words it changes to S, then from six to twelve it is the letter E. From thirteen letters onwards, the most likely letter to be in a word is the letter I.
- The most challenging words are three letter words. And this result is purely based on probability. It might take about 10 guesses to get even a letter on the board.
- While it is easier to guess 2-character or a single character word.

Proposed Work:

A set of words was downloaded. There are around 400,000 words in total. Then a list was updated with all the words in python. A similar experiment was carried out in C where an array was updated. Here, a ternary tree structure was updated.

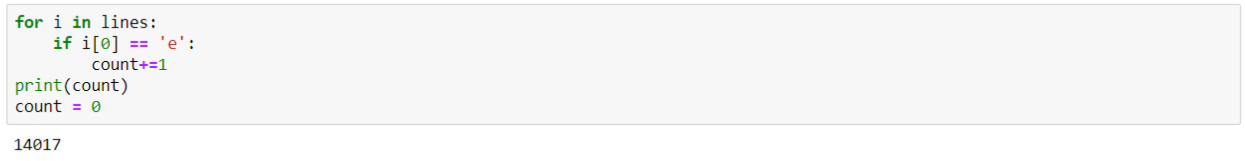
The program maintains a list of words that the user played and the guesses the user makes in a separate file. This can be useful to study user patterns and the most frequent characters and words the user uses. Now, the only optimal strategy is to guess the correct word in least number of attempts. This can be done by either studying the results of the game. Most-Frequently used alphabet that we have found was 'E'. This is the first possible way to optimise algorithm. To know what probability words, occur within a dictionary.

We let a group of users play the game and recorded their guesses as well the words they had to guess.

As per the observations, the player strategy involves:

1. Users mostly start with 'a'.
2. Second-most frequent choice is 'e'.
3. Least-used alphabet is 'q'.
4. Look for word pairs. e.g. TH, BR.
5. Double letters may not play well together as it seems.

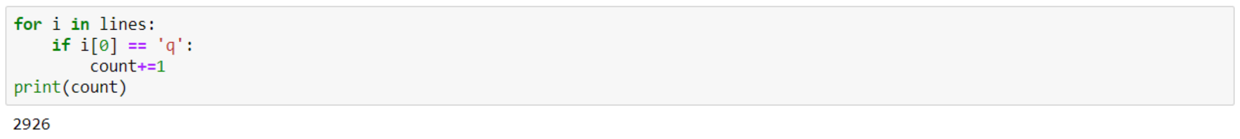
And the first three observations seem pretty good, since probability dictates that the chance of occurrence of a certain word will always be proportional to their occurrence in the dictionary itself.



```
for i in lines:
    if i[0] == 'e':
        count+=1
print(count)
count = 0
```

```
14017
```

'e' is the first character in 14,000 words



```
for i in lines:
    if i[0] == 'q':
        count+=1
print(count)
```

```
2926
```

‘q’ is first character in only about 3000 words.

Further, we also created a program where we let the computer guess the words the program randomly selected.

These were the observations:

Number of letters	**Number solved/possible**	**Win probability**	**First guess**
2	44 out of 96	0.458333	A
3	321 out of 972	0.330247	E
4	1793 out of 3073	0.583469	E

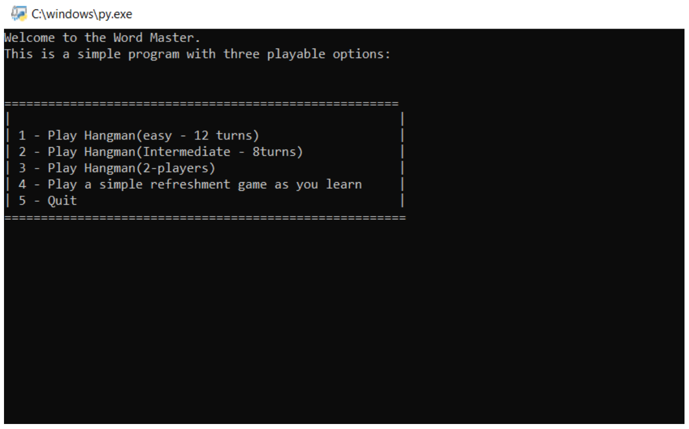
This shows that the computer itself works quite similar to that of a human being at least up to 4-character words.

The program code:

Main Menu:

```
import datetime

import os

file1 = "D:/Aki/Recorded_words.txt"

file2 = "D:/Aki/words.txt"

print("Welcome to the Word Master.\nThis is a simple program with three playable options:\n")

print()

print('========================================================= ')

print('|                                                       |')

print('| 1 - Play Hangman(easy - 12 turns)                     |')

print('| 2 - Play Hangman(Intermediate - 8turns)               |')

print('| 3 - Play Hangman(2-players)                           |')

print('| 4 - Play a simple refreshment game as you learn       |')

print('| 5 - Quit                                              |')

print('========================================================')

choice = int(input())

if choice==1:

    os.system('D:/Aki/Game2.py')

elif choice==2:

    os.system('D:/Aki/Hangman_ai.py')

elif choice==3:

    os.system('D:/Aki/regame.py')

elif choice==4:

    os.system('D:/Aki/Game3.py')

elif choice==5:
    print()

print("Do you have a word that you wish to enter in the list today?")

print("\nAnswer in yes or no")

choice = input()

if choice.lower() == "yes":

    print("Enter the word here:\t")

    word = input()

    with open(file1,'a') as writefile:

        writefile.write('Word %s recorded at %s.\n' %(word, datetime.datetime.now()))

    with open(file2,'a') as writefile2:

        writefile2.write('\n'+word+'\n')

    print("Thankyou. Have a nice day ahead.")

elif choice.lower() == 'no':

    print("Thankyou for playing...\n")

    print("Hope to meet you again...")
```

Output Screen:



```
C:\windows\py.exe
Welcome to the Word Master.
This is a simple program with three playable options:


======================================================
|                                                     |
| 1 - Play Hangman(easy - 12 turns)                   |
| 2 - Play Hangman(Intermediate - 8turns)             |
| 3 - Play Hangman(2-players)                         |
| 4 - Play a simple refreshment game as you learn     |
| 5 - Quit                                            |
=======================================================
```

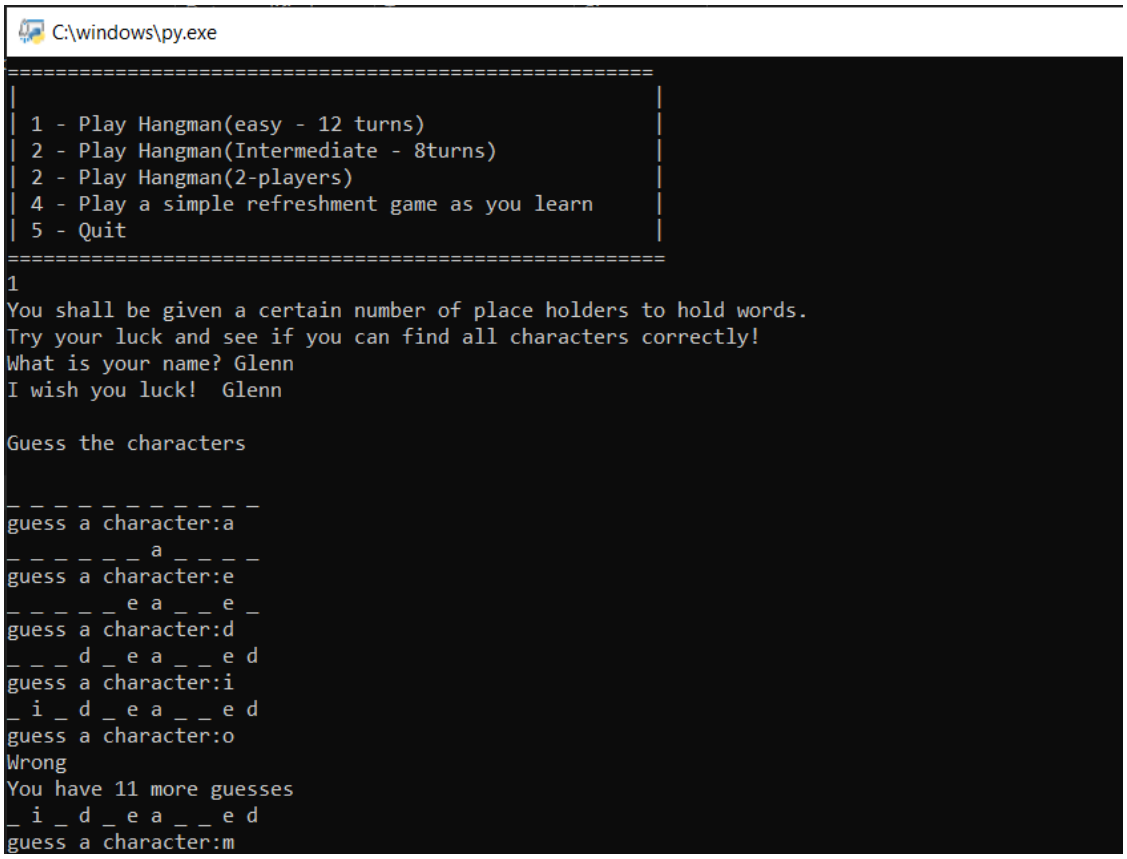
1.Hangman(easy) code:

```
import random

import datetime
```

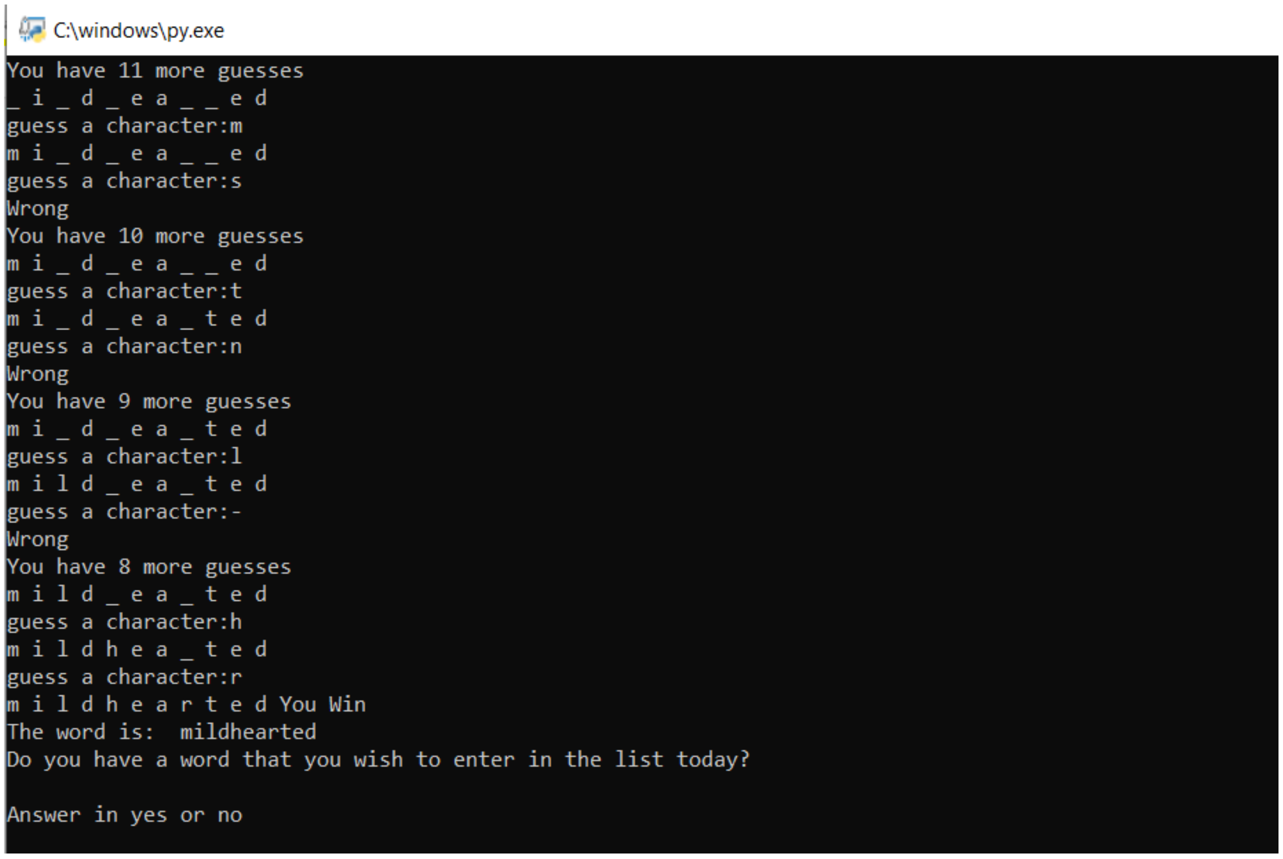
```
file1 = "D:/Aki/Game2_score.txt"

print("You shall be given a certain number of place holders to hold words.\nTry your luck and see if you can find all characters correctly!")

name = input("What is your name? ")

# Here the user is asked to enter the name first

print("I wish you luck! ", name)

words = [line.rstrip('\n').lower() for line in open('D:/Aki/words.txt')]

# Function will choose one random

# word from this list of words

word = random.choice(words)

print("\nGuess the characters\n")

guesses = ''

# any number of turns can be used here

turns = 12

while turns > 0:

    # counts the number of times a user fails

    failed = 0

    # all characters from the input

    # word taking one at a time.

    for char in word:

        # comparing that character with

        # the character in guesses

        if char in guesses:

            print(char,end=' ')

        else:

            print("_",end=' ')
            # for every failure 1 will be
            # incremented in failure
            failed += 1
    if failed == 0:
        # user will win the game if failure is 0
        # and 'You Win' will be given as output
        print("You Win")
        with open(file1,'a') as writefile:
            writefile.write('Player %s played %s and won at %s.\n' %(name,word, datetime.datetime.now()))
        # this print the correct word
        print("The word is: ", word)
        break
    # if user has input the wrong alphabet then
    # it will ask user to enter another alphabet
    guess = input("\nguess a character:")
    # every input character will be stored in guesses
    guesses += guess
    # check input with the character in word
    if guess not in word:
        turns -= 1
        # if the character doesn’t match the word
        # then “Wrong” will be given as output
        print("Wrong")
        # this will print the number of
        # turns left for the user
        print("You have", + turns, 'more guesses')
```

```
if turns == 0:
    print("You Loose\n")
    print("The word was: ",word)
    with open(file1,'a') as writefile:
        writefile.write('Player %s played %s and lost at %s.\n' %(name,word, datetime.datetime.now()))
```

Output Screen:



```
C:\windows\py.exe
======================================================
|                                                    |
| 1 - Play Hangman(easy - 12 turns)                  |
| 2 - Play Hangman(Intermediate - 8turns)            |
| 2 - Play Hangman(2-players)                        |
| 4 - Play a simple refreshment game as you learn    |
| 5 - Quit                                           |
=======================================================
1
You shall be given a certain number of place holders to hold words.
Try your luck and see if you can find all characters correctly!
What is your name? Glenn
I wish you luck!  Glenn

Guess the characters


_ _ _ _ _ _ _ _ _ _ _
guess a character:a
_ _ _ _ _ _ a _ _ _ _
guess a character:e
_ _ _ _ _ e a _ _ e _
guess a character:d
_ _ _ d _ e a _ _ e d
guess a character:i
_ i _ d _ e a _ _ e d
guess a character:o
Wrong
You have 11 more guesses
_ i _ d _ e a _ _ e d
guess a character:m
```



```
You have 11 more guesses
_ i _ d _ e a _ _ e d
guess a character:m
m i _ d _ e a _ _ e d
guess a character:s
Wrong
You have 10 more guesses
m i _ d _ e a _ _ e d
guess a character:t
m i _ d _ e a _ t e d
guess a character:n
Wrong
You have 9 more guesses
m i _ d _ e a _ t e d
guess a character:l
m i l d _ e a _ t e d
guess a character:-
Wrong
You have 8 more guesses
m i l d _ e a _ t e d
guess a character:h
m i l d h e a _ t e d
guess a character:r
m i l d h e a r t e d You Win
The word is:  mildhearted
Do you have a word that you wish to enter in the list today?

Answer in yes or no
```

Hangman(intermediate):

```
import datetime

import random

hangPatterns = ['''

  x-----x

  |    |

  |

  |

  |

  |

--x-----''', '''

  x-----x

  |    |
 |    O
 |
 |
 |
 |
--x-----"', "'
 x-----x
 |     |
 |    O
 |     |
 |
 |
 |
--x-----"', "'
 x-----x
 |     |
 |    O
 |     |
 |     |
 |
 |
--x-----"', "'
 x-----x
 |     |
 |    O
 |   \|
 |     |
 |
 |
--x-----''', '''
 x-----x
 |     |
 |     O
 |    \|/
 |     |
 |
 |
--x-----''', '''
 x-----x
 |     |
 |     O
 |    \|/
 |     |
 |    /
 |
--x-----''', '''
 x-----x
 |     |
 |     O
 |    \|/
 |     |
 |    / \\
  |

--x-----''']


def get_word():


    # Path to the text file

    with open('D:/Aki/words.txt', 'r') as f:


        # Reads each word after splitting

        words1 = f.read().splitlines()


    # Returns any random word

    return random.choice(words1)




incorrectLetters = ""        #keeps track of incorrect letters guessed

correctLetters = ""          #keeps track of correct letters guessed

play = True

guess = ""

correct = 0

index = 0

def printPattern(guesses):

    print (hangPatterns[guesses])

randomWord = get_word()

hiddenWord = list("")               #needs to be a list so the underscores in the middle of the word can be changed to
letters
```

```
i = 0

while i < len(randomWord):

    hiddenWord += "_"              #makes a list of underscores the size of the randomword

    i += 1



print("HANGMAN GAME")

print("------------")

print("You are a captive in the town of Vocabulary. Guess the correct word before you are hung.\n")



while play == True:



    printPattern(len(incorrectLetters))                #decides which pattern to print depending on how many incorrect
guesses youve made

    print ("Word: " + "".join(hiddenWord))               #how to print a list as a string

    print ("Incorrect letters: " + incorrectLetters)

    guess = input("Guess a letter (1 for guessing whole word): ")       #input for guess



    if guess == '1':

       guessWord = raw_input("Guess the word: ")                          #if user enters '1' they are able to guess the whole
word, if they get it wrong they lose the whole game

       if randomWord.find(guessWord) > -1:

           play = False

           print ("You have correctly guessed the word!")

       else:

           print ("You have incorrectly guessed the word, you lose...")

           print (hangPatterns[7])
```

```
            play = False

    elif len(guess) > 1:

        print ("Just guess a single character")        #error trap for more than one character

    elif correctLetters.find(guess) > -1:

        print ("You've already guessed " + guess)        #error trap for already guessed character

    else:

        if randomWord.find(guess) > -1:

            correctLetters += guess

            print ("Found " + guess)

            index = 0     #index needs to be reset each time

            while index < len(hiddenWord):          #This is for more than one occurence of the letter in the hidden
word

                index = randomWord.find(guess, index)

                if index == -1:

                    break

                hiddenWord[index] = guess

                correct += 1                  #increases correct letters guessed

                index += 1

        else:

            incorrectLetters += guess            #not including an error trap for already guessed incorrect letters, it
is the players' responsibility to not be an idiot. So he can choose 'g' twice if he wants and he will add to the hangman
a second time

            print ("Did not find " + guess)




    if len(incorrectLetters) == 7:        #lose check

        print (hangPatterns[7])
        print ("You lose...")

        play = False

    elif correct == len(randomWord):                         #win check

        print ("You win!")

        play = False


    if play == False:

        again = input("Would you like to play again? (y/n): ")          #option to play again or not

        if again.lower() == 'y':

            print ("RESETING GAME")


            randomWord = words[random.randint(0,len(words) - 1)]

            hiddenWord = list("")

            i = 0

            while i < len(randomWord):

                hiddenWord += "_"

                i += 1


            incorrectLetters = ""        #keeps track of incorrect letters guessed

            correctLetters = ""          #keeps track of correct letters guessed

            play = True

            guess = ""

            correct = 0
```

Output Screen:

Two Player:

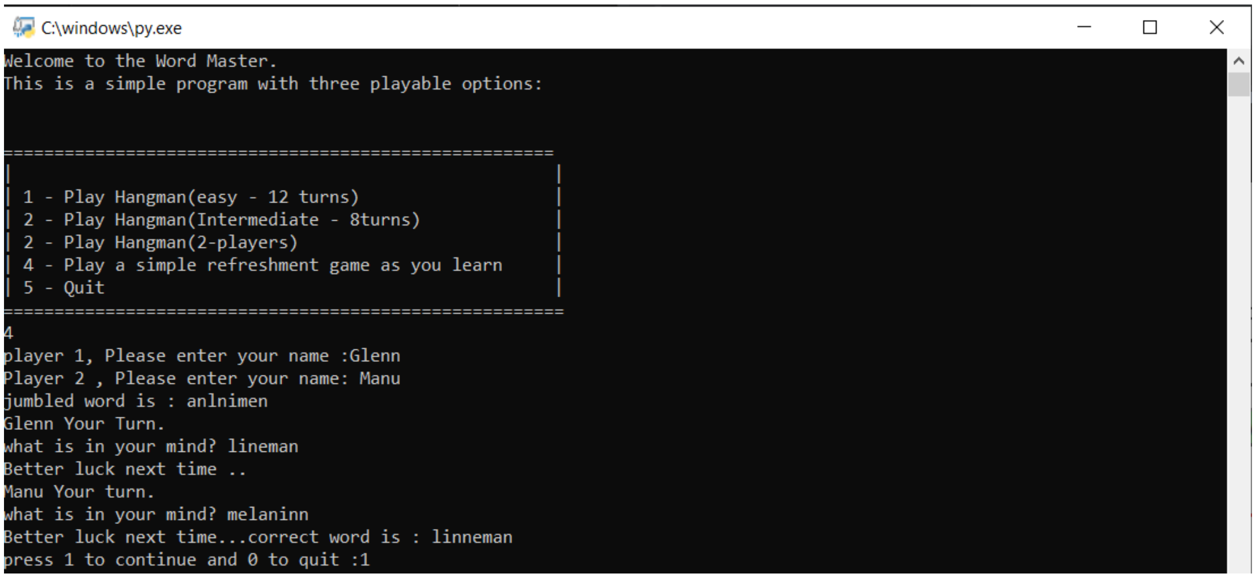
```
# -*- coding: utf-8 -*-

import datetime

import random

lines = [line.rstrip('\n').lower() for line in open('D:/Aki/words.txt')]

file1 = "D:/Aki/Game3.txt"

def choose():

    pick = random.choice(lines)

    return pick

def jumble(word):

    random_word = random.sample(word, len(word))

    # join() method join the elements

    # of the iterator(e.g. list) with particular character .

    jumbled = ''.join(random_word)

    return jumbled

# showing final score.

def thank(p1n, p2n, p1, p2):

    print(p1n, 'Your score is :', p1)

    print(p2n, 'Your score is :', p2)

    with open(file1,'a') as writefile:

        writefile.write('Scores %s and %s recorded by %s and %s at %s.\n' %(p1,p2,p1n, p2n,
datetime.datetime.now()))

    # check_win() function calling

    check_win(p1n, p2n, p1, p2)

    print('Thanks for playing...')
```

```
# declaring winner

def check_win(player1, player2, p1score, p2score):

    if p1score > p2score:

        print("winner is :", player1)

    elif p2score > p1score:

        print("winner is :", player2)

    else:

        print("Draw..Well Played guys..")

# playing the game.

def play():

    # enter player1 and player2 name

    p1name = input("player 1, Please enter your name :")

    p2name = input("Player 2 , Please enter your name: ")

    # variable for counting score.

    pp1 = 0

    pp2 = 0

    # variable for counting turn

    turn = 0

    # keep looping

    while True:

        # choose() function calling

        picked_word = choose()

        # jumble() fucntion calling

        qn = jumble(picked_word)

        print("jumbled word is :", qn)

        # checking turn is odd or even
```

```
if turn % 2 == 0:

    # if turn no. is even

    # player1 turn

    print(p1name, 'Your Turn.')

    ans = input("what is in your mind? ")

    # checking ans is equal to picked_word or not

    if ans == picked_word:

        # incremented by 1

        pp1 += 1

        print('Your score is :', pp1)

        turn += 1

    else:

        print("Better luck next time ..")



        # player 2 turn

        print(p2name, 'Your turn.')

        ans = input('what is in your mind? ')

        if ans == picked_word:

            pp2 += 1

            print("Your Score is :", pp2)

        else:

            print("Better luck next time...correct word is :", picked_word)

        c = int(input("press 1 to continue and 0 to quit :"))

        # checking the c is equal to 0 or not

        # if c is equal to 0 then break out

        # of the while loop o/w keep looping.
```

```
            if c == 0:

                # thank() function calling

                thank(p1name, p2name, pp1, pp2)

                break

    else:

        # if turn no. is odd

        # player2 turn

        print(p2name, 'Your turn.')

        ans = input('what is in your mind? ')


        if ans == picked_word:

            pp2 += 1

            print("Your Score is :", pp2)

            turn += 1


        else:

            print("Better luck next time.. :")

            print(p1name, 'Your turn.')

            ans = input('what is in your mind? ')

            if ans == picked_word:

                pp1 += 1

                print("Your Score is :", pp1)


            else:

                print("Better luck next time...correct word is :", picked_word)
                c = int(input("press 1 to continue and 0 to quit :"))


                if c == 0:

                    # thank() function calling

                    thank(p1name, p2name, pp1, pp2)

                    break

            c = int(input("press 1 to continue and 0 to quit :"))

            if c == 0:

                # thank() function calling

                thank(p1name, p2name, pp1, pp2)

                break

if __name__ == '__main__':

    # play() function calling

    play()
```

Output Screen:



C:\windows\py.exe

```
Welcome to the Word Master.
This is a simple program with three playable options:


=====================================================
|                                                    |
| 1 - Play Hangman(easy - 12 turns)                  |
| 2 - Play Hangman(Intermediate - 8turns)            |
| 2 - Play Hangman(2-players)                        |
| 4 - Play a simple refreshment game as you learn    |
| 5 - Quit                                           |
======================================================
4
player 1, Please enter your name :Glenn
Player 2 , Please enter your name: Manu
jumbled word is : anlnimen
Glenn Your Turn.
what is in your mind? lineman
Better luck next time ..
Manu Your turn.
what is in your mind? melaninn
Better luck next time...correct word is : linneman
press 1 to continue and 0 to quit :1
```

Conclusion:

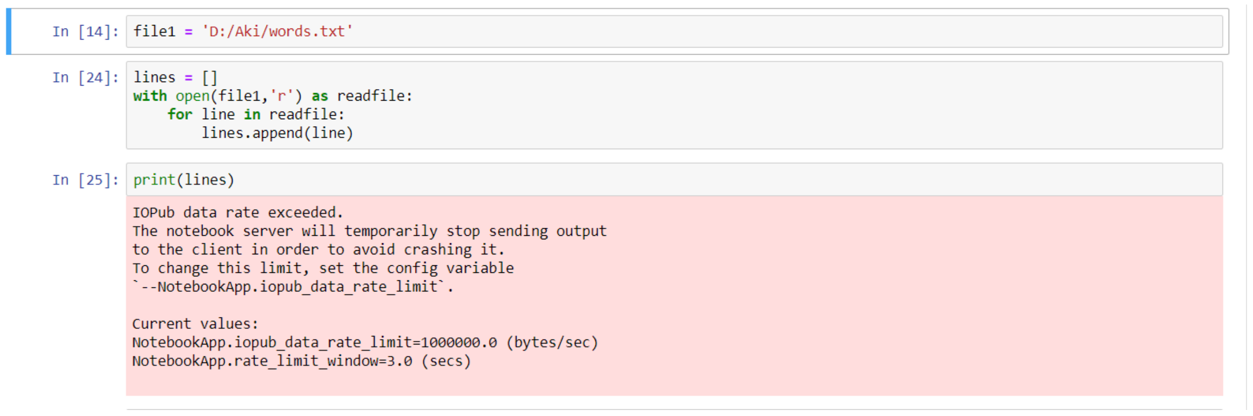
Further, using both python and C in the project gave us the following insight:

Using Python list data structure is a time-consuming process as lists allow only sequential access. And thus, we need to use linear search algorithm for comparison of words. And the same is needed for loading word database. The average time-complexity for this algorithm is O(n).

A possible solution can be using Ternary search trees. Here, observation suggests that the process is way faster with mathematics suggesting that the average time complexity was O(log(n)) and for worst case, it was O(n). But the problem here was that there will be multiple lines of code for fairly no output and thus a very long program.

Thus, we realise that using python, the largest drawback of our code is the great time consumption.

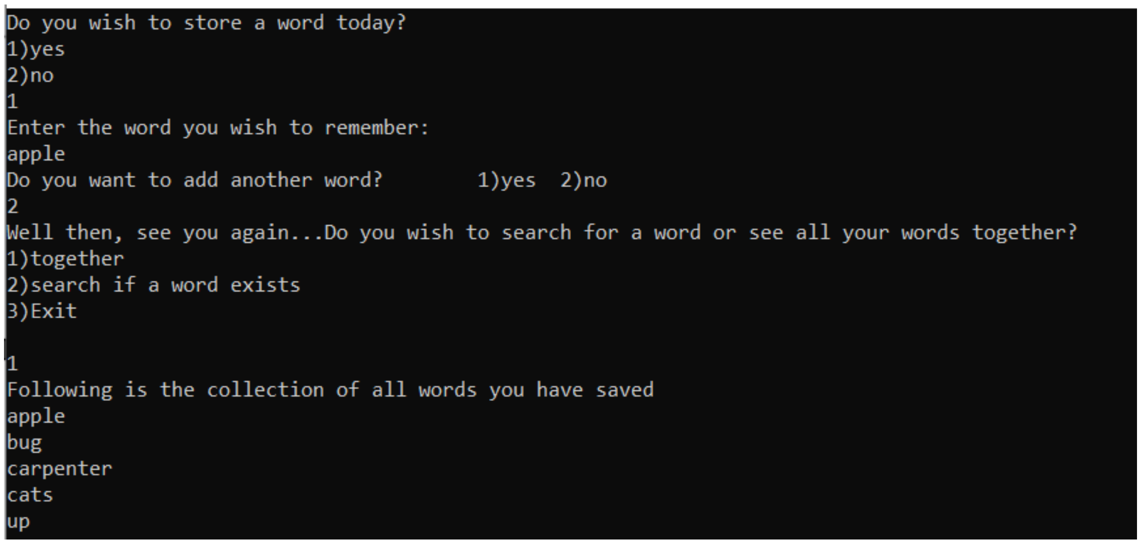
Comparing Python (Jupyter Notebook and Ternary Search in C)



```
In [14]: file1 = 'D:/Aki/words.txt'

In [24]: lines = []
         with open(file1,'r') as readfile:
             for line in readfile:
                 lines.append(line)

In [25]: print(lines)

         IOPub data rate exceeded.
         The notebook server will temporarily stop sending output
         to the client in order to avoid crashing it.
         To change this limit, set the config variable
         `--NotebookApp.iopub_data_rate_limit`.

         Current values:
         NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
         NotebookApp.rate_limit_window=3.0 (secs)
```

Python was unable to handle the large set of data.

Additionally, the time python lists took to load the words data was more than 30 seconds on a 64-bit architecture.



```
Do you wish to store a word today?
1)yes
2)no
1
Enter the word you wish to remember:
apple
Do you want to add another word?        1)yes  2)no
2
Well then, see you again...Do you wish to search for a word or see all your words together?
1)together
2)search if a word exists
3)Exit

1
Following is the collection of all words you have saved
apple
bug
carpenter
cats
up
```

The C program took way less time in loading and searching for the words.

Future Work:

Developing a computer application that has a choice of words similar to human beings. This is different from auto-predict keyboard which is more algorithm-based.

References:

[1] http://datagenetics.com/blog/april12012/index.html

[2] http://www.sharkfeeder.com/hangman/

[3]https://unbabel.com/blog/dictionary-digital-age/

[4] https://www.futurelearn.com/courses/english-academic-study/0/steps/41870

[5]https://www.washingtonpost.com/news/the-intersect/wp/2015/12/08/why-dictionaries-have-started-sanctioning-even-the-dumbest-internet-slang/

[6] https://thelearnersway.net/ideas/2014/12/7/dictionaries-vs-internet