

WORDZONE

Submitted in partial fulfilment for the award of the degree of

Bachelor of Technology in Information Technology

by

RITVEAK DUGAR

16BIT0407

Under the guidance of

Prof. E Sathiyamoorthy

**School of Information Technology & Engineering
VIT, Vellore.**



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

May, 2020

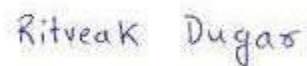
DECLARATION

I hereby declare that the thesis entitled “WORDZONE ” submitted by me, for the award of the degree of Bachelor of Technology in Information Technology to VIT is a record of bonafide work carried out by me under the supervision of Prof E Sathiyamoorthy.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 14 May 2020

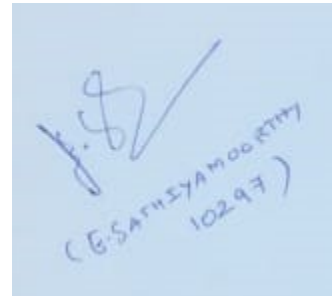
A handwritten signature in blue ink that reads "Ritvik Dugar". The signature is written in a cursive style with some capitalization.

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled “**WORDZONE**” submitted by **RITVEAK DUGAR (16BIT0407)** SCHOOL OF INFORMATION TECHNOLOGY & ENGINEERING VIT, for the award of the degree of Bachelor of Technology in **Information Technology** is a record of bonafide work carried out by him under my supervision during the period, 01. 12. 2019 to 30.04.2020, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The Project report fulfils the requirements and regulations of University and in my opinion meets the necessary standards for submission.



Place : Vellore
Date : 13 May 2020

Signature of the Guide

Internal Examiner

External Examiner

Head of the Department
SITE, VIT VELLORE

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to **Prof E Sathiyamoorthy**, School of Information Technology and Engineering, Vellore Institute of Technology, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavour.

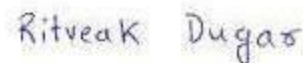
I would like to express my gratitude to **Dr.G.Vishwanathan, Mr. Sankar Viswanathan, Dr.Anand A. Samuel, Dr. S. Narayanan, and Dr BalakrushnaTripathy**, School of Information Technology and Engineering, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to **Dr. Jasmin Norman(Head of Department, SITE)**, all teaching staff and members working as limbs of our university for their not-self-centred enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore

Date:13 May 2020



RitveakDugar

ABSTRACT

This project is an attempt to bring in various word related paradigms in a single place, where users can learn, understand and find their words. The project has four main sections namely Find Zone, Understand Zone, Knowledge Zone and Learning Zone. Find Zone is the section where user can find words they want, for example, rhyming words with particular meaning or a word for their crossword with particular hints or words for their scrabble game. Understand Zone is the section where users can understand a word by looking up multiple meanings and examples of it. Knowledge Zone is the section where users can know more related words like synonyms, antonyms, hypernym, holonym and hyponym. Learning Zone is the section which utilises the searches done in the whole application to train a machine learning model. This model smartly suggests words for the user to learn.

The whole project is aimed to improve vocabulary of users by giving them a platform to find, understand, know and learn words based on their personalised usage.

CONTENTS

Acknowledgement	i
Abstract	ii
Contents	iii
List of Figures	vi
List of Tables	vii
List of Abbreviations	viii

1. INTRODUCTION

1.1 OBJECTIVE.....	1
1.2MOTIVATION.....	1
1.3BACKGROUND	2

2.PROJECT DESCRIPTION AND GOAL

2.1COMPONENTS	4
2.1.1FIND ZONE.....	4
2.1.2 UNDERSTANDZONE	4
2.1.3 KNOWLEDGEZONE	5
2.1.4 LEARNING ZONE	5
2.2GOAL	5

3. TECHNICAL SPECIFICATION

3.1 PROGRAMMING LANGUAGE.....	6
3.1.1PYTHON	6
3.1.2KIVY	7
3.2 TOOLS AND LIBRARIES.....	7
3.2.1NLTK	7
3.2.2WORDNET	8

3.2.3SPELL CHECKER	9
3.2.4 PRONOUNCING.....	10
3.2.5DATAMUSE AND OXFORD API	11
3.2.6WORDFREQ	13
3.2.7PICKLE	14
3.3 TRAINING MODEL.....	15
3.3.1LSTM	15
 4. DESIGN	
4.1 UML DESIGN.....	16
4.1.1FIND ZONE.....	16
4.1.2UNDERSTAND ZONE	17
4.1.3KNOWLEDGE ZONE.....	18
4.1.4 LEARNING ZONE.....	19
4.2 SYSTEM ARCHITECTURE.....	20
 5. SCHEDULE AND MILESTONES	
5.1 SCHEDULE.....	21
5.2 MILESTONE	22

6.PROJECT DEMONSTRATION

6.1 FIND ZONE	24
6.1.1RHYMING ZONE	25
6.1.2CROSSWORD HELPER	29
6.1.3SCRABBLE HELPER	33
6.2UNDERSTAND ZONE	36
6.3KNOWLEDGE ZONE	38
6.3.1SIMILAR MEANING WORDS	39
6.3.2OPPOSITE MEANING WORDS	41
6.3.3HYPERNYM / HYPONYMN / HOLONYM	43
6.4 LEARNING ZONE	46
6.4.1LOW LEVEL WORDS	47
6.4.2MEDIUM LEVEL WORDS	47
6.4.3HIGH LEVEL WORDS	47
6.4.4TRAINING AND PREDICTION	48

7. RESULT & DISCUSSION

7.1 RESULTS	51
7.2 FUTURE WORK	53

8.SUMMARY54

9.REFERENCES55

List of Figures

Figure No.	Title	Page No.
3.1	Spell Checker	9
3.2	Pronouncing Output	10
3.3	Datamuse API Output	11
3.4	Oxford Dictionary API Output	12
3.5	WordFreq code and output	13
4.1	Find Zone UML	16
4.2	Understand Zone UML	17
4.3	Knowledge Zone UML	18
4.4	Learning Zone UML	19
4.5	System Architecture	20
6.1	Home Screen	23
6.2	FindZone Screen	24
6.3	Rhymer Dictionary Screen	26
6.4	Rhymer Dictionary Option Screen	27
6.5	Rhymer Dictionary Input Screen	27
6.6	Rhymer Dictionary Output Screen	28
6.7	Rhymer Dictionary misspelled Output Screen	28
6.8	Crossword Helper Input Screen	30
6.9	Crossword Helper Output Screen	31
6.10	Crossword Helper Narrower Results Selection Screen	31
6.11	Crossword Helper Narrower Results Screen	32
6.12	Scrabble Helper Input Screen	34
6.13	Scrabble Helper Output Screen	35
6.14	Understand Zone Input Screen	37
6.15	Understand Zone Output Screen	37
6.16	Knowledge Zone Screen	38
6.17	Knowledge Zone Synonyms Input Screen	40
6.18	Knowledge Zone Synonyms Output Screen	40
6.19	Knowledge Zone Antonyms Input Screen	42
6.20	Knowledge Zone Synonyms Output Screen	42
6.21	Knowledge Zone Hypernymn input Screen	45
6.22	Knowledge Zone Hypernymn Output Screen	45
6.23	Learning Zone Screen	46
6.24	Learning Zone training data to learning section mapping	47
6.25	Learning Zone Pickle files	48
6.26	Learning Zone flow	48

List of Tables

Table No.	Title	Page No.
1.1	Literature Survey	2
5.1	December 2019 Task Schedule	21
5.2	January 2020 Task Schedule	21
5.3	February 2020 Task Schedule	21
5.4	March 2020 Task Schedule	22
5.5	April 2020 Task Schedule	22
5.6	Milestones	22
7.1	Low model Actual Vs Prediction test	51
7.2	Medium model Actual Vs Prediction test	51
7.3	High model Actual Vs Prediction test	52
7.4	RMS Error for all models	52

List of Abbreviations

Abbreviation	Full form
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
LSA	Latent Semantic Analysis
GUI	Graphical User Interface
LSTM	Long short-term memory
CMU	Carnegie Mellon University
API	Application Programming Interface
UML	Unified Modelling Language

Chapter 1

Introduction

1.1 OBJECTIVE

The objective of WORDZONE is to provide a single platform for users to do multiple word related searches and improve their vocabulary by learning words that are suggested to them based on their personalised usage.

Therefore giving a wide range of functionalities to the user to play around words and increase their knowledge as well.

1.2 MOTIVATION

Being a writer I often feel the need to improve my vocabulary and at times look for specific words as well. However increasing one's vocabulary is a tough task to do, because there is no magical/efficient place where you can get words on the basis of your own vocabulary.

While studying Machine Learning, an idea hit my mind: What if a model is trained on basis of my word searches and suggests me new words to learn on the basis of that!

Hence to accomplish it, WORDZONE has a lot of word search related sections, which enables user to not only find the words they are looking for but passively generate personalised set of word data as well.

6.1 BACKGROUND

A lot of projects have been made previously on NLP using the features, tools and techniques that are used in wordzone.

Below is a literature survey on few of those tools, techniques and properties.

Ref no.	Topic	Summary	Use/Meaning
1.	Web Corpus[1]	Traditionally written corpora are primarily recorded from print media, with advent of internet, web data can be used to train corpora as well.	Corpora are basically huge collections of words and their associated features like meanings, pronunciations. They are used for NLP based usages
2.	NLTK[2]	The need of an umbrella covering a lot of linguistics related paradigm gave birth to NLTK toolkit.	NLTK being a collection of modules which helps in computation of linguistics is extensively used in projects related to NLP.
3.	WordNet[3]	WordNet is a lexical database for the English language. It is basically a combination of dictionary and thesaurus which contains words, their definitions and many other related words.	Wordnet acts as database which is used for fetching properties of words and their related words.
4.	Semantic Analysis[4]	Latent Semantic Analysis is a good approach for finding accurate results. Another important use of LSA is to find out semantic similarities between different set of textual data.	Semantic Analysis helps in determining the meaning of sentences.
5.	Phonetics[5]	Phonology is used to recognize sound so as to find out the language, semantic and syntactic meaning. It also helps in pronouncing words and finding phonetically similar words.	Stores data related to pronunciation of words, hence used in projects related to rhyming words.

6.	Python GUI [6]	There are many GUI libraries for Python programs. The most famous ones are: Kivy, Tkinter, PyQt, PyGUI, Pyforms, PySide, Flexx, PySimpleGUI, IPyWidgets, Wax Python GUI, etc.	To give a Graphical User Interface to the python programs.
7.	Dictionary[7][8]	Dictionary requires corpus which contains words and its attributes like meaning, pronunciation, example etc. So that it can also be used for machine translation and language processing.	In addition to using dictionary for finding meanings, one can also use it to find rhyming words and pronunciations.
8.	LSTM[9]	LSTM is a Machine Learning algorithm used for efficiently predicting and training on time series data.	LSTM can be used for predicting the word based on the data generated by the user's usage.

Table 1.1 Literature Survey

Chapter 2

Project Description and Goal

WordZone is a place where user can Find, Understand, Know and Learn new words, to accomplish that, it is divided into different components. Each component has a specific role to play and together they serve for improving of user's vocabulary.

2.1 COMPONENTS

Wordzone has four main sections, which combined makes wordzone a one of its own platform:

1. Find Zone
2. Understand Zone
3. Knowledge Zone
4. Learning Zone

2.1.1 FIND ZONE

FIND ZONE allows the user to find words that they are looking for.

A user can look up for words which rhyme with a given word, further the result can be narrowed down to be of some specific meaning in the “Rhyming Words” section.

User can also look up for words with some meaning, which can be narrowed down to a list of words with fixed character at a position in the “Crossword Helper” section.

Words for Scrabble can be searched using “Scrabble Helper” where the word fragment, its position and length of the word can be found.

2.1.2 UNDERSTAND ZONE

UNDERSTAND ZONE allows user to find the meaning and examples of the word they enter. It also shows multiple meanings that a word might have and returns example for each case as well.

2.1.3 KNOWLEDGE ZONE

KNOWLEDGE ZONE allows user to know more words around the word they enter. A user can find Synonyms in the “Similar meaning” section, antonyms in “Opposite meaning” section and other related words in “Hypernymn/Hyponymn/Holonymn” section

2.1.4 LEARNING ZONE

LEARNING ZONE is the section that makes use of all the searches made by the user in the application to smartly suggest new words to learn. The searches and results help to train three different models, giving user option to learn a low-level, mid-level and high-level word. The model can be retrained manually, whenever substantially new data is being recorded. Hence making WORDZONE an evolving model which not only gives user a platform to find, understand and know words, but makes their learning personalised as well.

2.2 GOAL

The goal of WORDZONE is to provide a platform for users, which they can use not only for fun and curiosity but for improving their overall vocabulary as well. The learning should be at a personalised level so that everyone gets to learn at their own level and at their own pace.

Chapter 3

Technical Specification

3.1 PROGRAMMING LANGUAGES

Programming language is the base for any software development; they act as the field where the crop grows. Hence there is always a particular kind of field which is best for a given crop.

The following are the programming languages that are used in WORDZONE.

3.1.1 PYTHON

Python is a high level programming language which is always preferred over others when it comes to Machine Learning and Natural Language Processing, because of the wide variety of libraries that it supports.

Since WORDZONE uses NLP and Machine Learning, python was the go to language.

Python being easy and widely used has a lot of support online, hence development and troubleshooting is eased because of the rich documentation and forums online.

Many libraries like NLTK, pronouncing, spellchecker allowed implementation and raw data for processing in the software.

The function written in python then processes the data and returns result.

Python in WORDZONE is primarily used for the all round processing, and hence it can truly be called as the soul of wordzone.

This soul however needed a body as well, so to provide a Graphical User Interface, Kivy was selected.

3.1.2 KIVY

Kivy is an open source python framework for developing user interfaces. There were other options for GUI like Tkinter, Pyqt and many others. Kivy stood out in the line because the modern look and the fact that it is more customisable.

Kivy has its own syntax for defining screens, buttons and all other graphical components. Thereby aiding in development of rich but simplistic User Interface.

3.2 TOOLS AND LIBRARIES

There are few tools and libraries which helped in building WORDZONE apart from the basic programming languages. All these tools and libraries are freely available and have good documentation to understand and work with them. There are the common ones like Pandas, Numpy, Keras, Tensorflow, Pickle and other specific ones like NLTK, Pronouncing, Spell Checker, wordfreq; all these along with logic facilitate the working of wordzone.

3.2.1 NLTK

NLTK – Natural Language Toolkit is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language. This is one of the most usable and mother of all NLP libraries.

It consists of the most common algorithms such as tokenizing, part-of-speech tagging, stemming, sentiment analysis, topic segmentation, and named entity recognition. NLTK helps the computer to analysis, pre-process, and understand the written text.

Since wordzone only plays around words and not the syntactic meanings, wordnet is used at most. Wordnet is a lexical database which records semantic relation of words including synonyms, hyponyms, and meronyms. The synonyms are grouped also known as synsets with short definitions and usage examples. This comes in handy for making a rich group of related words.

3.2.2 WORDNET

Wordnet acts as a corpus, which is nothing but a collection of words. In the view of wordzone, wordnet acts as our database which has a lot of words along with some associated meaning and properties. The words are in turn linked to each other forming a network of words which helps to find related words to a given word.

The logic whatsoever required is applied on data collected from wordnet which can be made rich by including more and more related words.

For example in Rhyming words section, where the rhyming word is required to be related to something, a rich collection of words are made by using wordnet:

```
syns=wordnet.synsets(mword)
for syn in syns:
    rr+=syn.lemma_names()

#Since simple synonym set is not enough, lets add more related words
#finding all related words among which rhyming words is to be found

hr=[]
syns=wordnet.synsets(mword)
for syn in syns:
    sn=syn.hypernyms()#broader category:colour is a hypernym of red.
    An=syn.hyponyms() #narrower category - red : color
    dn=syn.member_holonyms()#Body is a holonym of arm, leg and heart
    for s in sn:
        hr+=s.lemma_names()
    for a in an:
        hr+=a.lemma_names()
    for d in dn:
        hr+=d.lemma_names()

#now even "loaf" gets included when "food" is given as input

#making the list richer by adding synonyms of the words which are in h
r.

Fn=[]
for h in hr:
    ss=wordnet.synsets(h)
    for s in ss:
        fn+=s.lemma_names()
```

Thereby making the collection rich and then applying logic.

3.2.3 SPELL CHECKER

Spell checker is a library in Python which suggest words near the wrongly spelled words. This helps in the result pages when there is no result due to wrong spelling.

Below is an example where the entered word was “Conundrom” instead of “Conundrum”:

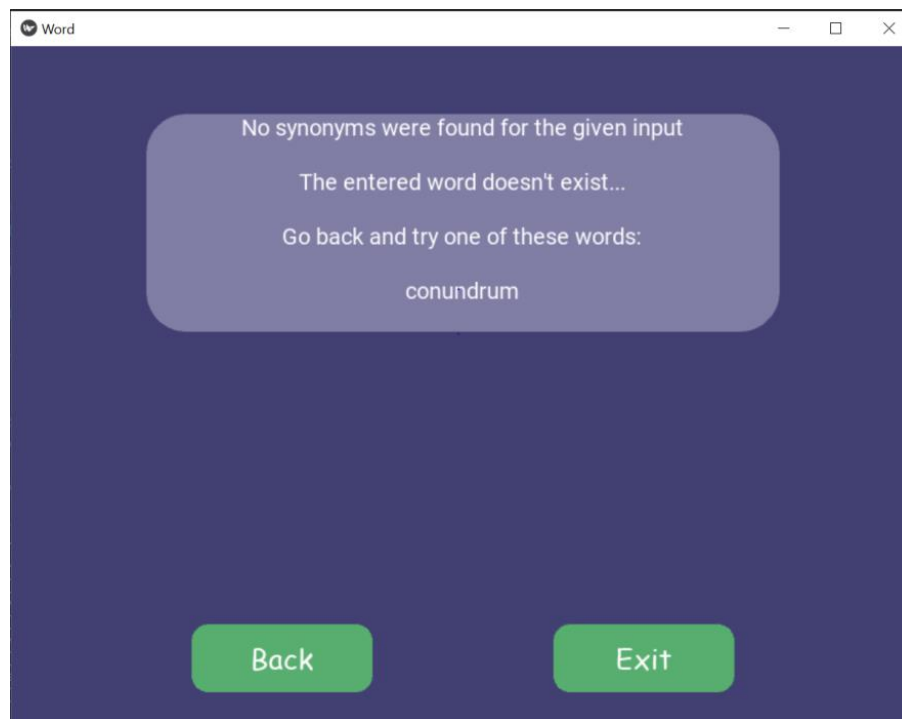


Fig3.1 Spell Checker

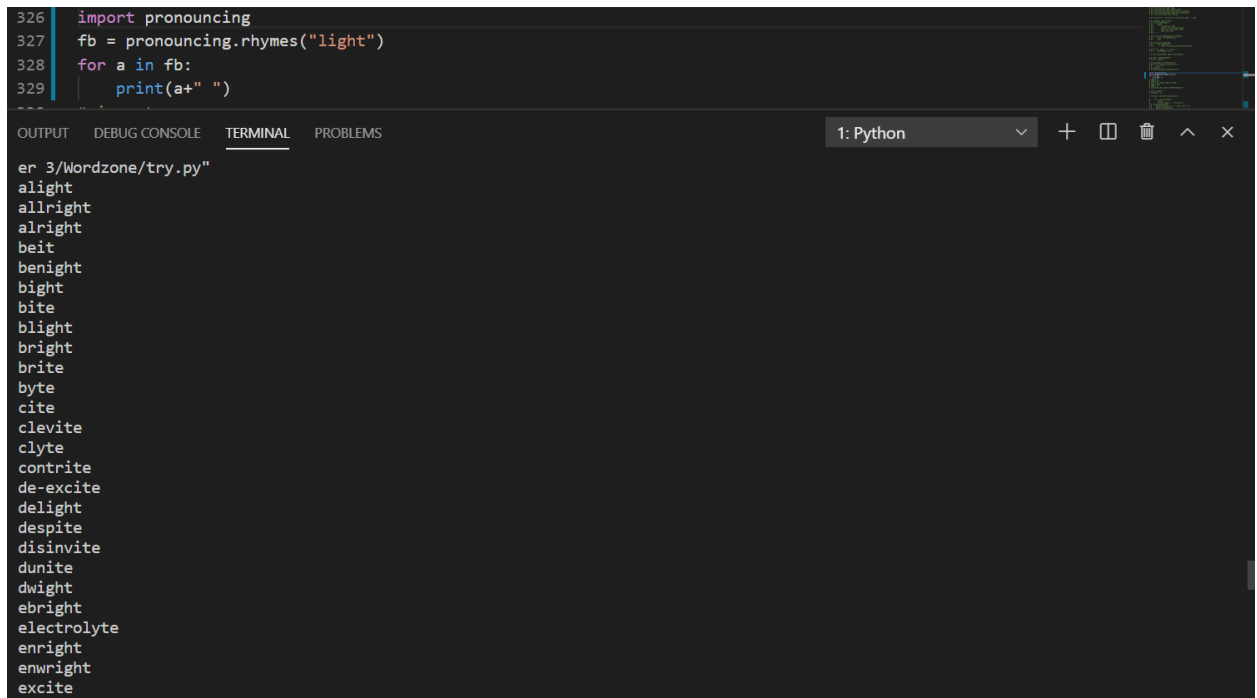
Code snippet:

```
if(len(synonyms)>0):
    ss=set(synonyms)
    new = ""
    for x in ss:
        new += x+"\n"
    return new
else:
    str="No synonyms were found for the given input\n\n"
    if(spell.correction(word)!=word):
        str+="The entered word doesn't exist...\n\n"
        if len(spell.candidates(word))>0 :
            str+="Go back and try one of these words:\n\n"
            for s in spell.candidates(word):
                str+=s+"\n"
    return str
```

3.2.4 PRONOUNCING

Pronouncing is a library in python which helps in finding similar sounding words, it makes use of CMU Pronouncing Dictionary which holds the phonetics of words.

For example, searching rhyming words to “Light” would return:

A screenshot of a Python IDE terminal window. The top part shows a code editor with the following Python code:


```
326 import pronouncing
327 fb = pronouncing.rhymes("light")
328 for a in fb:
329     print(a+" ")
```

The bottom part shows the terminal output, which lists various words that rhyme with "light":

```
er 3/Wordzone/try.py"
alight
allright
alright
beit
benight
bight
bite
blight
bright
brite
byte
cite
clevite
clyte
contrite
de-excite
delight
despite
disinvite
dunite
dwight
ebright
electrolyte
enright
enwright
excite
```

Fig3.2Pronouncing Output

In WordZone, it is used as a part of rhymer dictionary:

A screenshot of a Python code editor showing a function definition for a rhymer dictionary. The code is as follows:

```
def rhy(self,word):
    if(word==""):
        return "No word entered"
    fb = pronouncing.rhymes(word)
    if(len(fb)>0):
        ss=set(fb)
        new = ""
        for x in ss:
            new += x+"\n"
        #print(new)
        return new
```

3.2.5 DATAMUSE AND OXFORD API

The Datamuse API is a word-finding query engine which has a lot of methods.

NLTK is a more raw form of receiving data, but Datamuse is a smarter way.

One can use it to find words that match a given set of constraints.

For example searching for words related to a particular description let's say "Ringing in ears" would give:

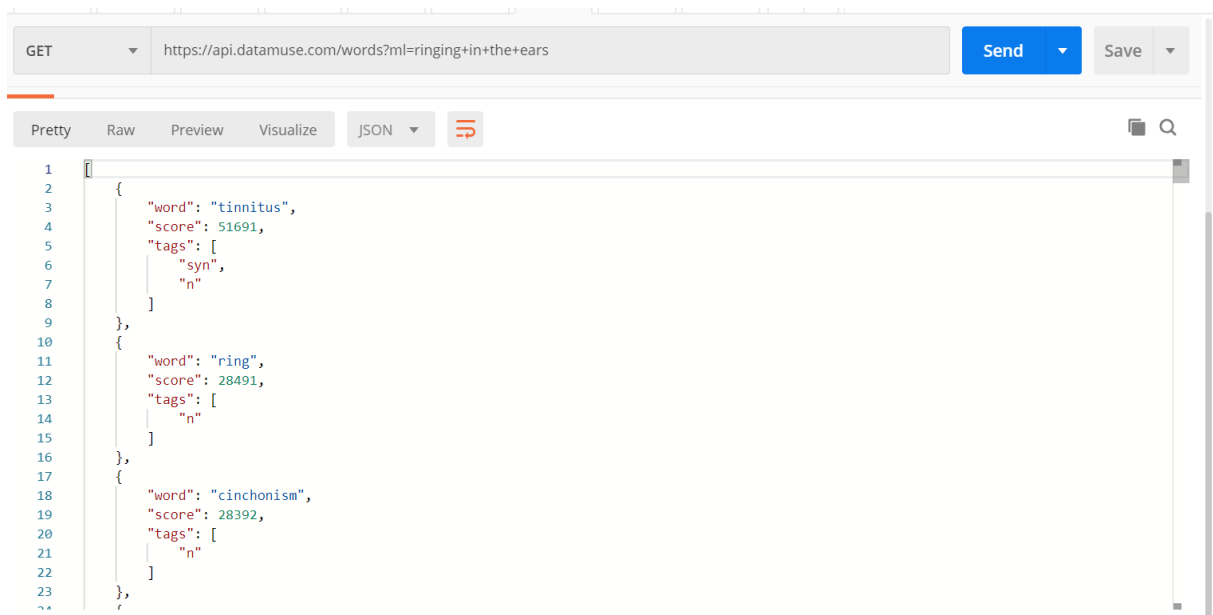


Fig3.3Datamuse API Output

This is used in Crossword Helper, where we need list of words when a user enters a hint.

Further operations are done on the list obtained by using this API.

Code Snippet:

```
fn=[]  
res= requests.get("https://api.datamuse.com/words?ml="+w)  
for a in res.json():  
    fn.append(a["word"])  
  
#The words related to description are stored in fn now.
```

The Oxford Dictionary API uses official oxford dictionary for returning word meanings.

For example, it returns all the possible meanings with examples:

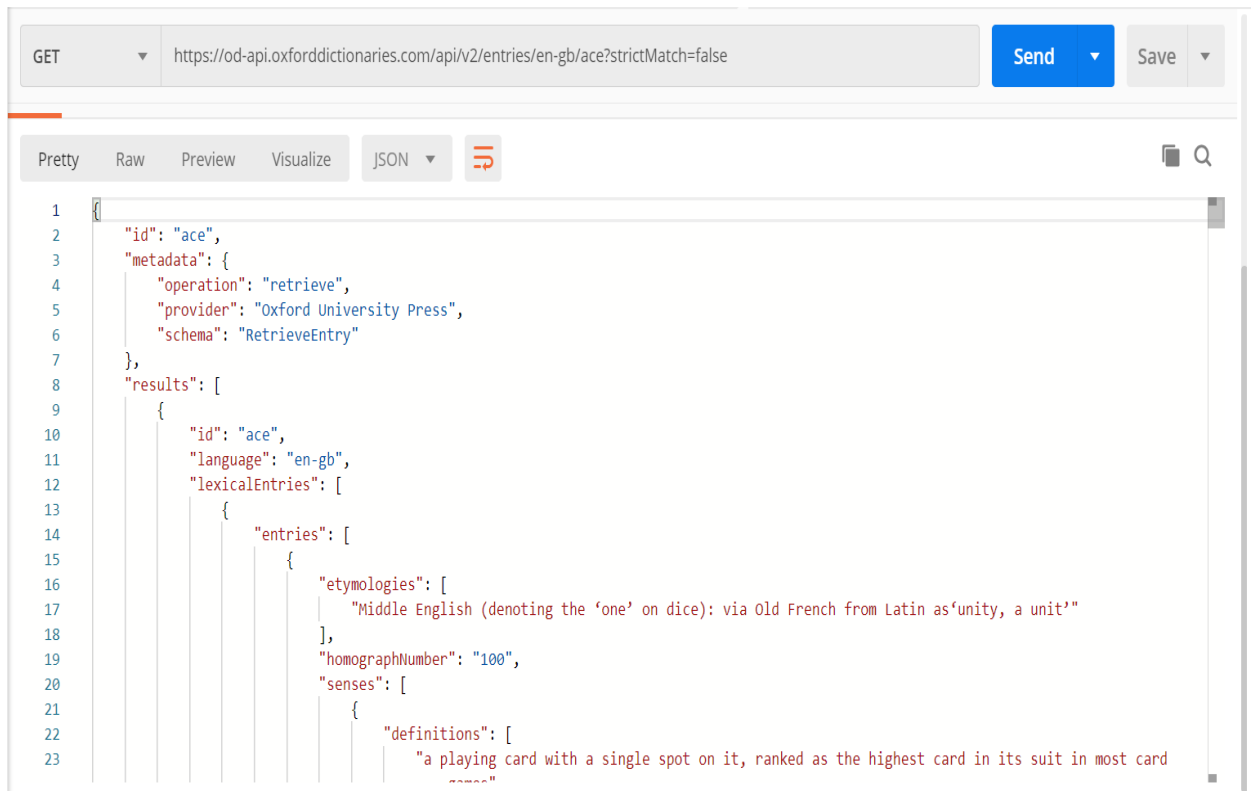


Fig3.4 Oxford Dictionary API Output

This is used in Understanding Zone where user enters a word and gets meanings of it with examples.

Code Snippet:

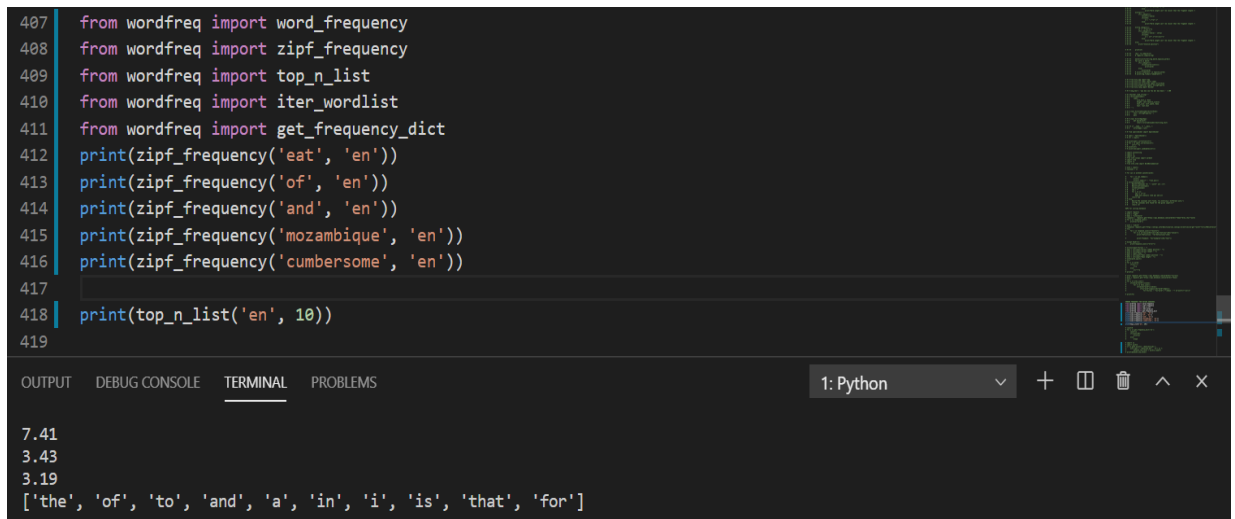
```
str=""
response= requests.get("https://od-api.oxforddictionaries.com/api/v2/entries/en-gb/"+word+"?strictMatch=false",headers={"Accept": "application/json","app_id": "c1498ba3","app_key": "ec959282e97d787344cbe7cfef13c965"})
try:
    for a in response.json()["results"]:
        for b in a["lexicalEntries"][0]["entries"][0]["senses"]:
            str+="Definition:\n "+b["definitions"][0]+" \n"
            str+="Example: \n"+b["examples"][0]["text"]+" \n \n"
```

3.2.6WORDFREQ

Wordfreq^[10] is a python library which gives out frequency of a word in English language. This frequency can be taken as a measure of how common or less common is a word in English language, thereby can be used to show easy (ie, very common) and medium(ie. Common) and hard (ie. Less common) words.

This feature is used in training and predicting words for Leaning Zone which uses the word entered by user in whole wordzone to train a model and then suggesting new words to learn based on the user's usage.

A demo of what this library returns is below:

A screenshot of a Python IDE with a dark theme. The editor shows a Python script using the wordfreq library. The script imports word_frequency, zipf_frequency, top_n_list, iter_wordlist, and get_frequency_dict from wordfreq. It then prints the zipf frequency for the words 'eat', 'of', 'and', 'mozambique', and 'cumbersome' in English. Finally, it prints the top 10 most common words in English. The terminal window at the bottom shows the output of these operations.

```
407 from wordfreq import word_frequency
408 from wordfreq import zipf_frequency
409 from wordfreq import top_n_list
410 from wordfreq import iter_wordlist
411 from wordfreq import get_frequency_dict
412 print(zipf_frequency('eat', 'en'))
413 print(zipf_frequency('of', 'en'))
414 print(zipf_frequency('and', 'en'))
415 print(zipf_frequency('mozambique', 'en'))
416 print(zipf_frequency('cumbersome', 'en'))
417
418 print(top_n_list('en', 10))
419
```

OUTPUT DEBUG CONSOLE TERMINAL PROBLEMS

7.41
3.43
3.19
['the', 'of', 'to', 'and', 'a', 'in', 'i', 'is', 'that', 'for']

Fig3.5WordFreq code and output

3.2.7 PICKLE

Pickle is a library in python which converts data into byte stream(called serializing) and converts byte stream into data (called de-serializing). Pickling is required to store python objects, list in our case.

The data entered by the user in wordzone is of great importance and can be used in suggesting new words to learn, hence pickling comes in handy for storing the data.

In Wordzone, there are three different .pkl files which contains low, medium and high level words based on their frequency in English language.

Code snippet:

```
if os.path.exists('./data/mid.pkl'):
    with open ('./data/mid.pkl', 'rb') as f:
        dataset_mid_level = pickle.load(f)

else:
    dataset_mid_level = []

if(zipf_frequency(word, 'en')>2.5 and zipf_frequency(word , 'en')<4.5):
    dataset_mid_level.append(zipf_frequency(word, 'en'))
with open ('./data/mid.pkl', 'wb') as f:
    pickle.dump(dataset_mid_level, f)
```

3.3 TRAINING MODEL

Where there is data, there must be a model that trains on it and gives features for predicting upcoming data.

Since WordZone works with the words and inputs given by the user, it receives a lot of data. This data can be used to give a personalised feel to the user in another zone called Learning Zone.

The Learning Zone basically shows new words on the basis of the user's past searches.

There are a lot of models that one can use, but the reason LSTM was chosen is that it is designed to store time series data.

Since user's vocabulary would be increasing with time, it would be reflected in their usage, which can be recorded and used to predict future values and hence help the user to increase their vocabulary to higher levels.

This requires Time Series Forecasting.

3.3.1 LSTM

The data generated in WordZone is univariate and over a period of time the user's vocabulary increases, hence it is a time series data as well.

LSTM is a Machine Learning algorithm which trains on time series data and it works on univariate data as well. Hence LSTM stood as a viable choice for predicting the word based on the univariate time series data generated by the user's usage.

Code Snippet:

```
train_generator = TimeseriesGenerator(frequency_train, frequency_train, length
=look_back, batch_size=20)

from keras.models import Sequential
from keras.layers import LSTM, Dense

model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(look_back, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Chapter 4

Design

Design is an integral part of development, it paves the pathway for the development. Below are the designs made for wordzone.

4.1 UML DESIGN

4.1.1 FIND ZONE:

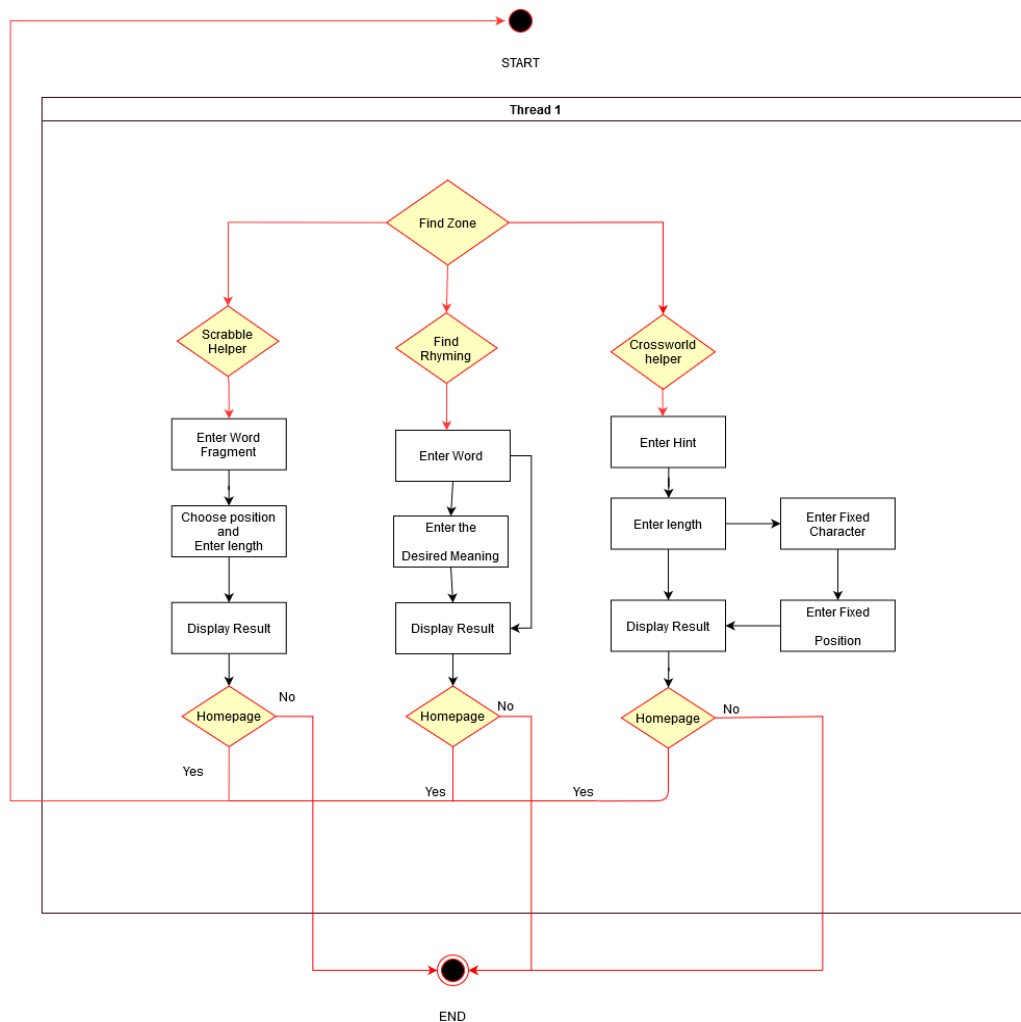


Fig 4.1 FindZone UML

4.1.2 UNDERSTAND ZONE:

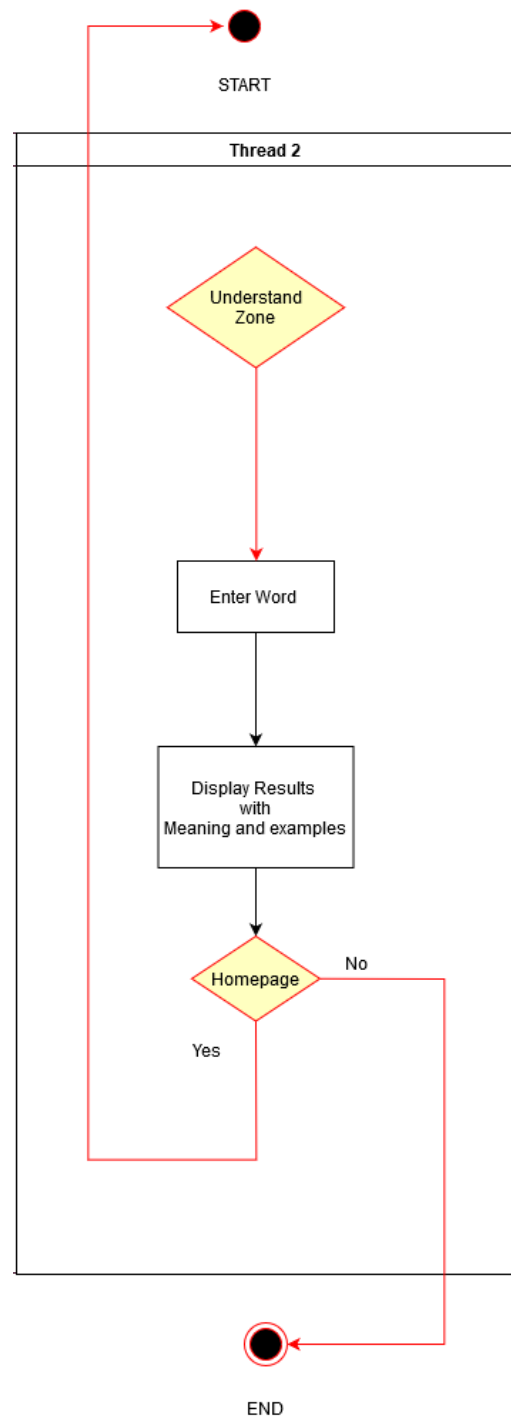


Fig 4.2 Understand Zone UML

4.1.3 KNOWLEDGE ZONE:

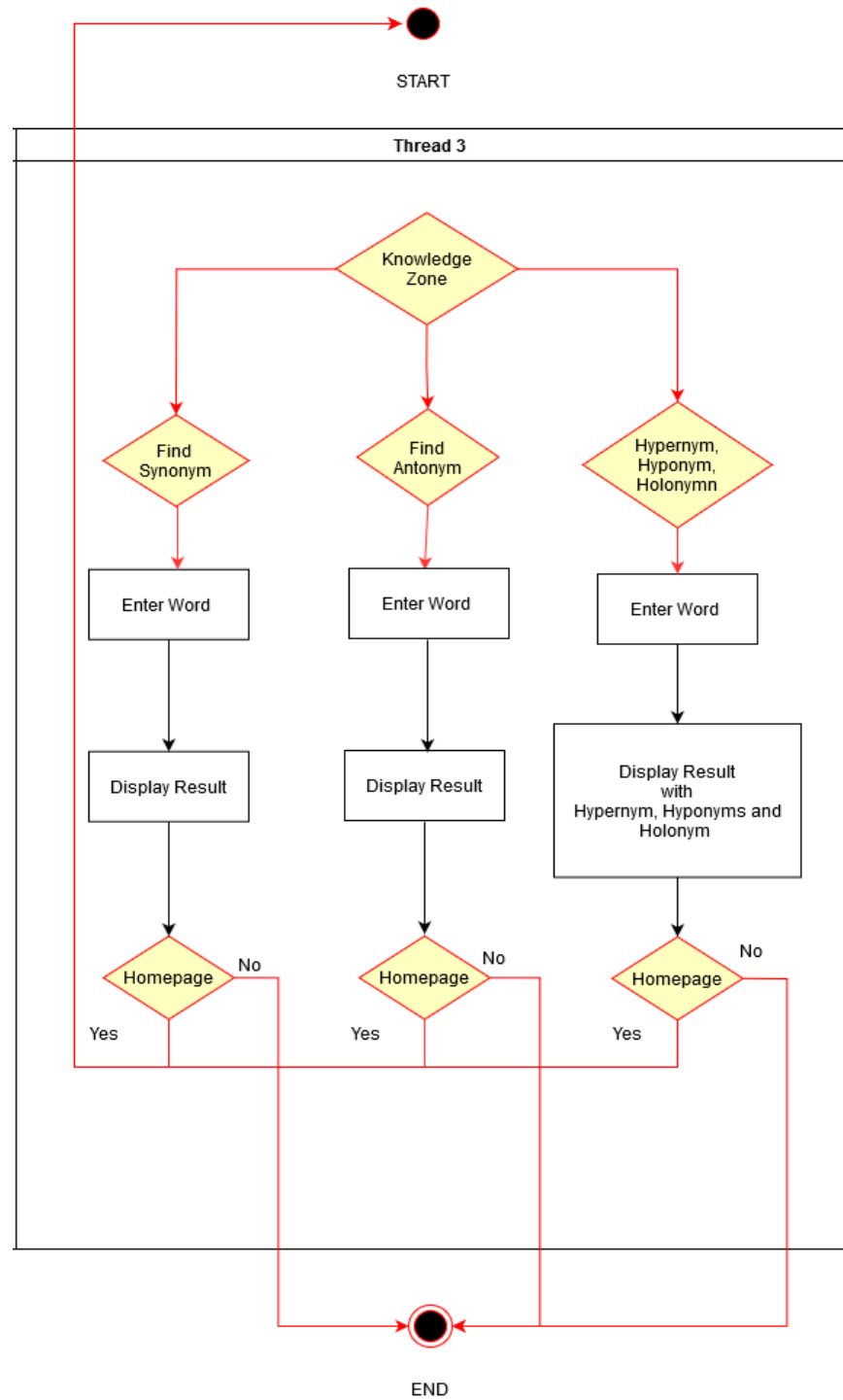


Fig 4.3 Knowledge Zone UML

4.1.4 LEARNING ZONE:

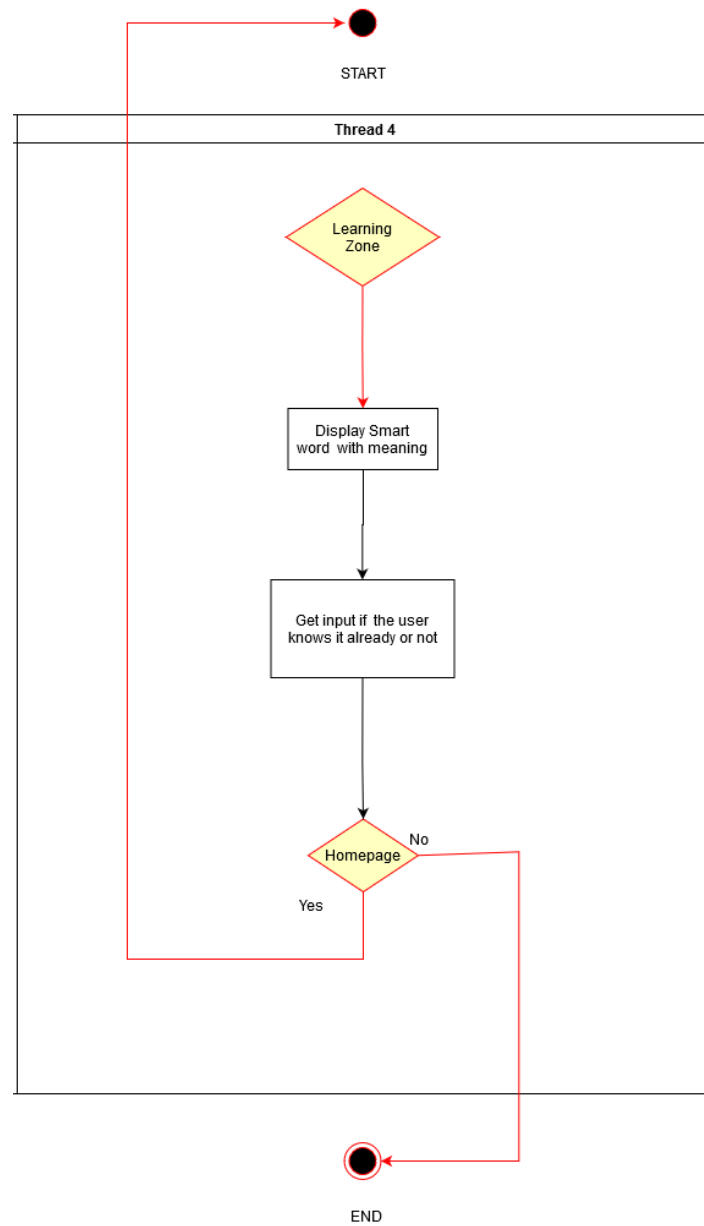


Fig 4.4 Learning Zone UML

6.1 SYSTEM ARCHITECTURE:

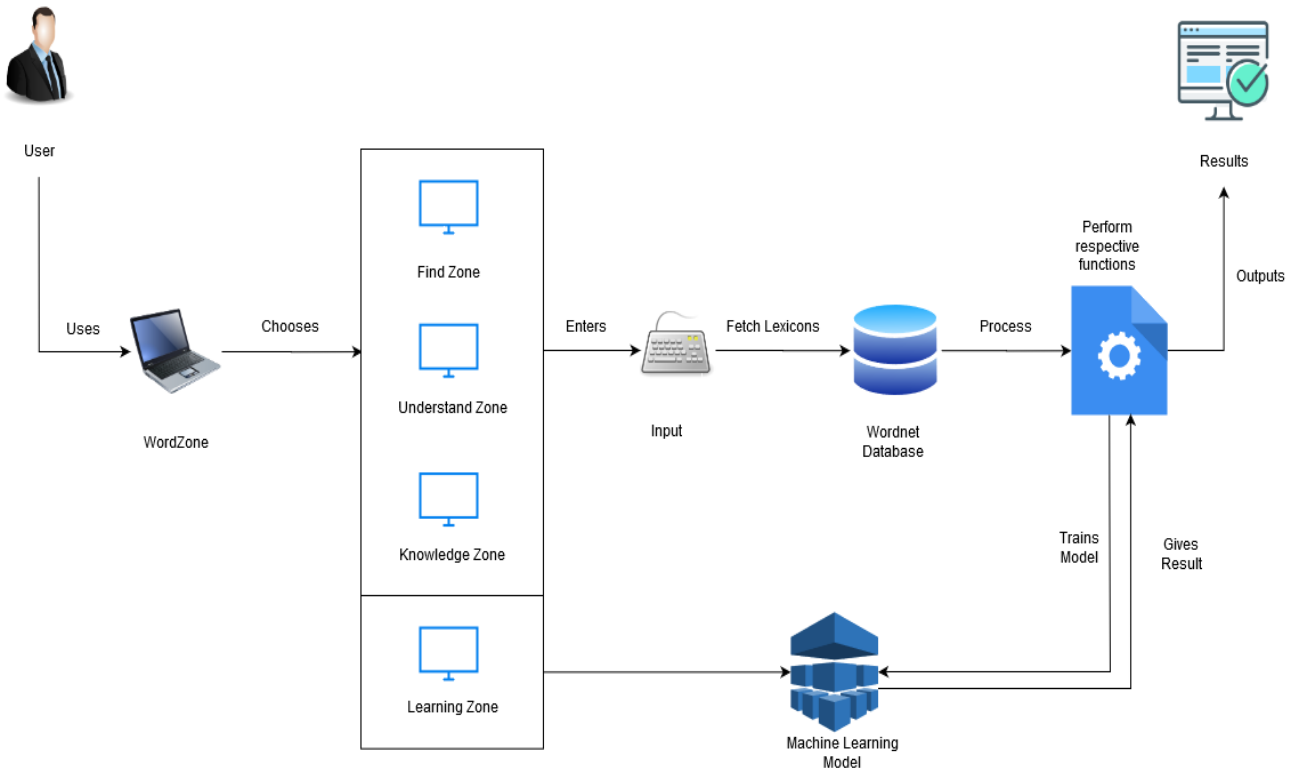


Fig 4.5System Architecture UML

Chapter 5

Schedule and Milestones

5.1 Schedule

Week 1	Defining Problem statement for the project.
Week 2	Literature Review of 5 papers
Week 3	Literature Review of next 5 papers
Week 4	Goals defining

Table5.1 December 2019 Task Schedule

Week 5	Detailed Workflows
Week 6	Learning Kivy
Week 7	Learning Kivy
Week 8	Implementing Rhymer Dictionary

Table5.2 January 2020 Task Schedule

Week 9	Implementing Crossword Helper
Week 10	Implementing Scrabble helper
Week 11	Completing Findzone GUI with Kivy
Week 12	Implementing and completing Understand Zone GUI

Table5.3 February 2020 Task Schedule

Week 13	Implementing Synonyms and Antonyms
Week 14	Implementing Hypernyms, Hyponyms and Holonyms
Week 15	Completing Knowledge Zone GUI
Week 16	Merging all Screens

Table5.4 March 2020 Task Schedule

Week 17	Research on data models
Week 18	Planning which data goes into which model
Week 19	Data collection and model building
Week 20	Learning Zone Implementation along with GUI

Table5.5 April 2020 Task Schedule

5.2 Milestones

Milestone Number	Week Number	Milestone
1	5	Project Design
2	11	Completing FindZone
3	12	Completing Understand Zone
4	15	Completing Knowledge Zone
5	20	Completing Learning Zone

Table5.6 Milestones

Chapter 6

Project Demonstration

WORDZONE has four main modules namely – FIND ZONE, UNDERSTAND ZONE, KNOWLEDGE ZONE and LEARNING ZONE. All these modules further have different sections. These sections together contribute to train a model which helps in personalised word suggesting model which helps the user to learn new words.

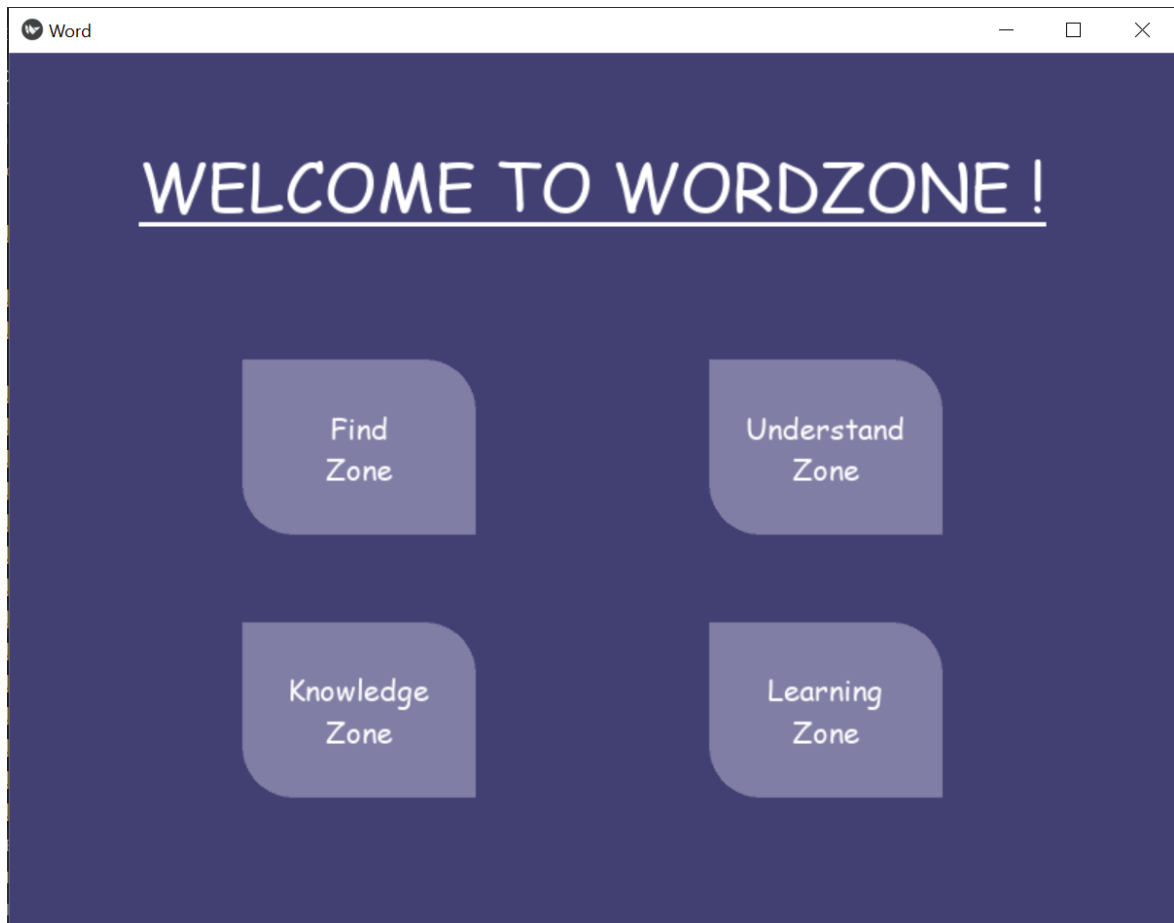


Fig 6.1 Home Screen

6.1 FIND ZONE

Find Zone comprises of sections which helps the user in finding words with constraints which belong to any of the three sections:

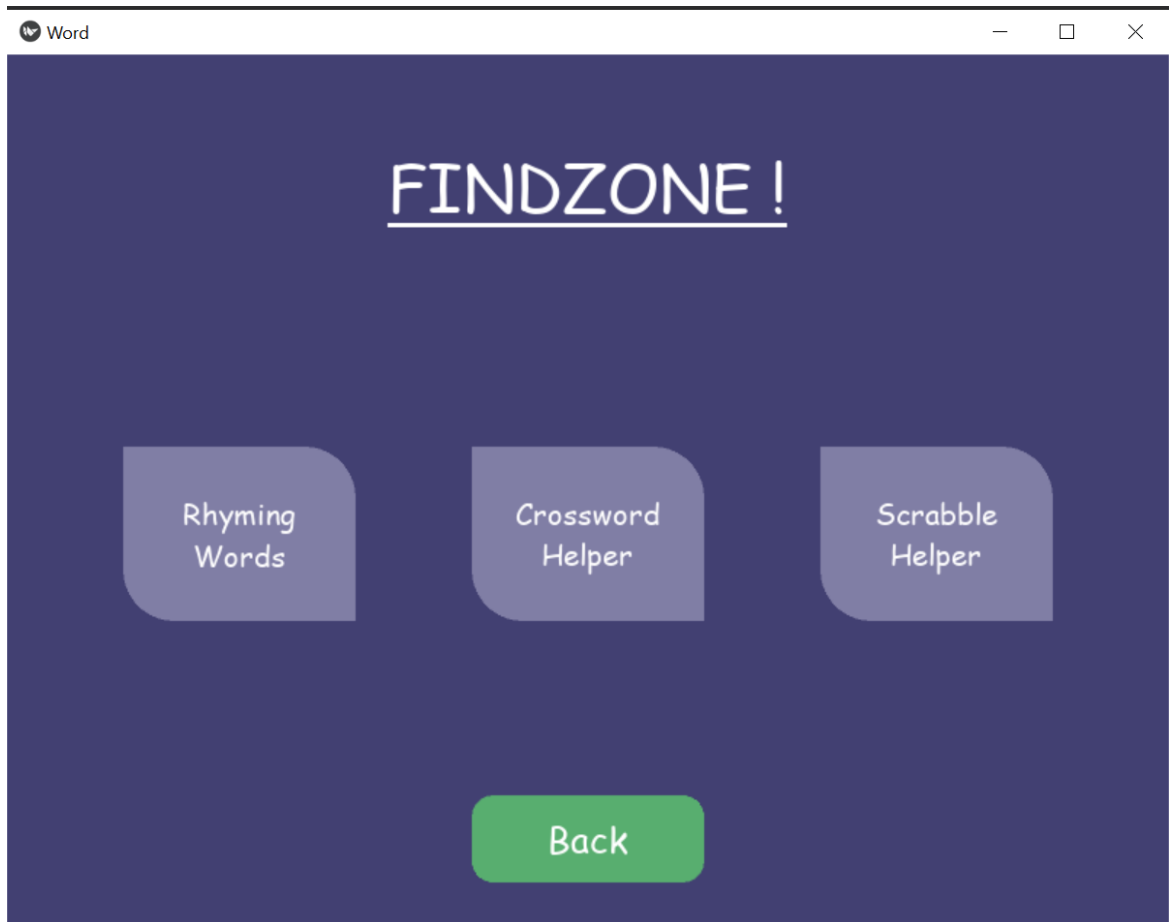


Fig 6.2FindZone Screen

6.1.1 RHYMING ZONE

Who doesn't like to put a bit of rhyme into their writings once in a while! This is the section where user can find words rhyming to a particular word, additionally; the user can input a word with which the rhyming word should be related to, thereby easing the way for poets who are looking for a rhyming word with a particular word but with a particular meaning.

Logic Code with explanation in comments:

```
def rhymerdic(self,iword,mword) :
    if(iword==""):
        return "No word entered"
    # a proper module separately made for finding rhyming words,
    based on cmudict.
    Fb = pronouncing.rhymes(iword)
    #Simple synonym set
    rr=[]
    syns=wordnet.synsets(mword)
    for syn in syns:
        rr+=syn.lemma_names()

    #Since simple synonym set is not enough, lets add more related words
    #finding all related words among which rhyming words is to be found

    hr=[]
    syns=wordnet.synsets(mword)
    for syn in syns:
        sn=syn.hypernyms()#broader category:colour is a hypernym of red.
        An=syn.hyponyms() #narrower category - red : color
        dn=syn.member_holonyms()#Body is a holonym of arm, leg and heart
        for s in sn:
            hr+=s.lemma_names()
        for a in an:
            hr+=a.lemma_names()
        for d in dn:
            hr+=d.lemma_names()

    #now even "loaf" gets included when "food" is given as input
    #making the list richer by adding synonyms of the words which are in h
    r.

    Fn=[]
    for h in hr:
        ss=wordnet.synsets(h)
        for s in ss:
            fn+=s.lemma_names()

    fn = list(dict.fromkeys(fn)) # removing duplicates

    #now selecting only the words that are common in both

    fo = list(set(fb)&set(fn))
    for chk in fb:
        for chk1 in fn:
```

```

my_regex = r"^.*" + re.escape(chk) + r"$"
found = (re.search(my_regex ,chk1, re.M|re.I))
if found:
    fo.append(found.group())
fo=list(set(fo))

new = ""
for x in fo:
    new += x+"\n"
if(new==""):
    return "No word found with specified meaning !"
return new

```

The code above is incorporated in a GUI by using Kivy. Below are the snippets with the workflow:

1. Taking Input:

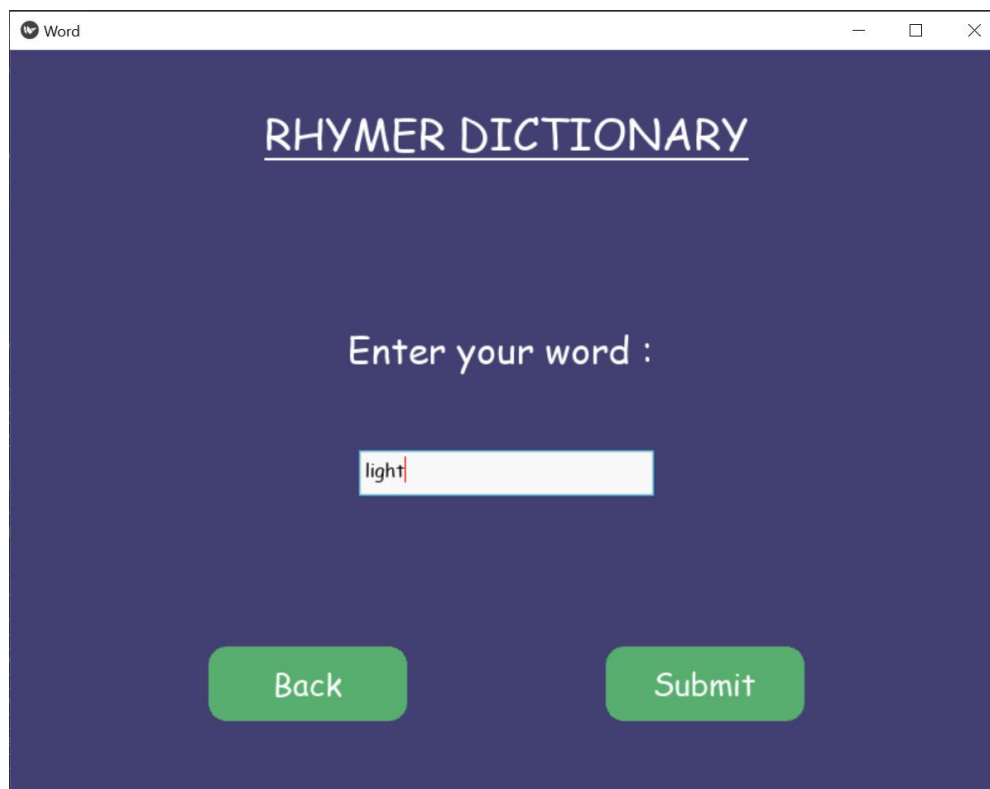


Fig6.3 Rhymer Dictionary Screen

2. Choosing if the word should be of particular meaning:

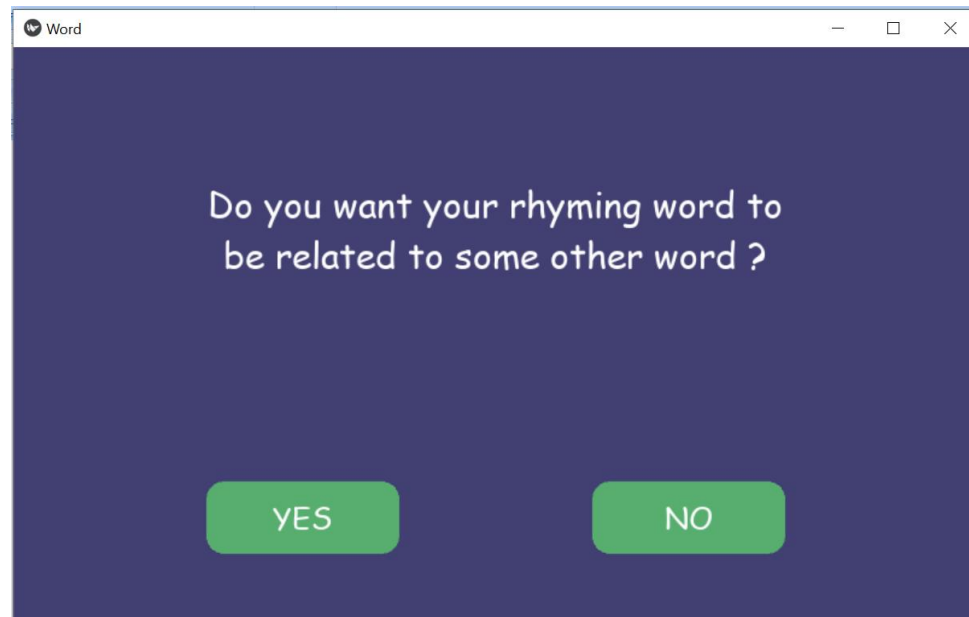


Fig 5.4 Rhymer Dictionary Option Screen

3. Entering the word, the results should be related to:

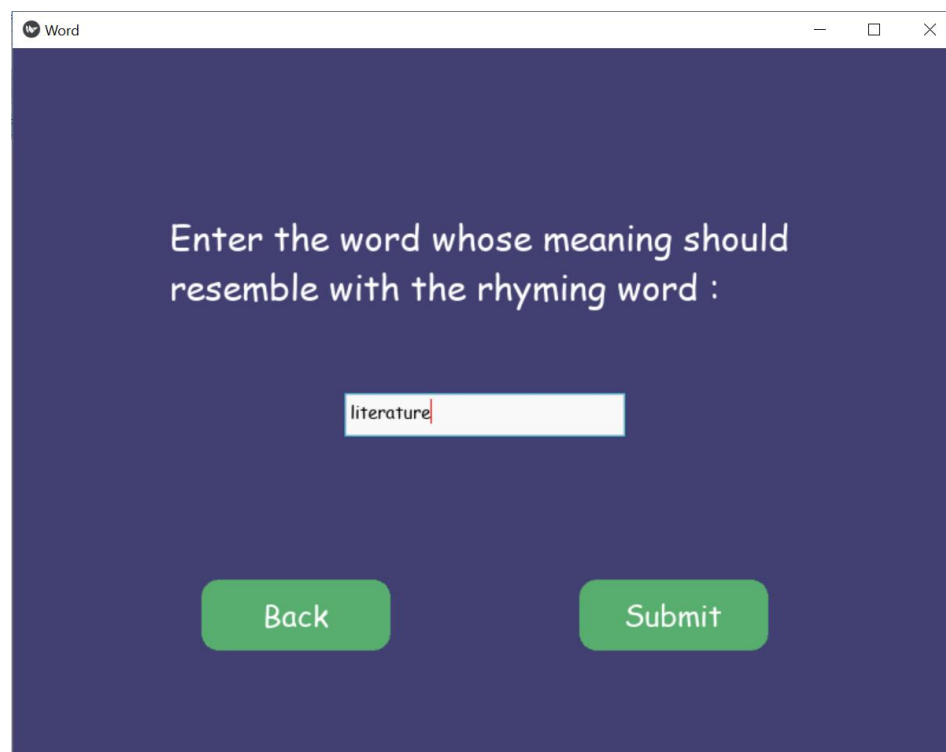


Fig 5.5 Rhymer Dictionary Input Screen

4. The results:

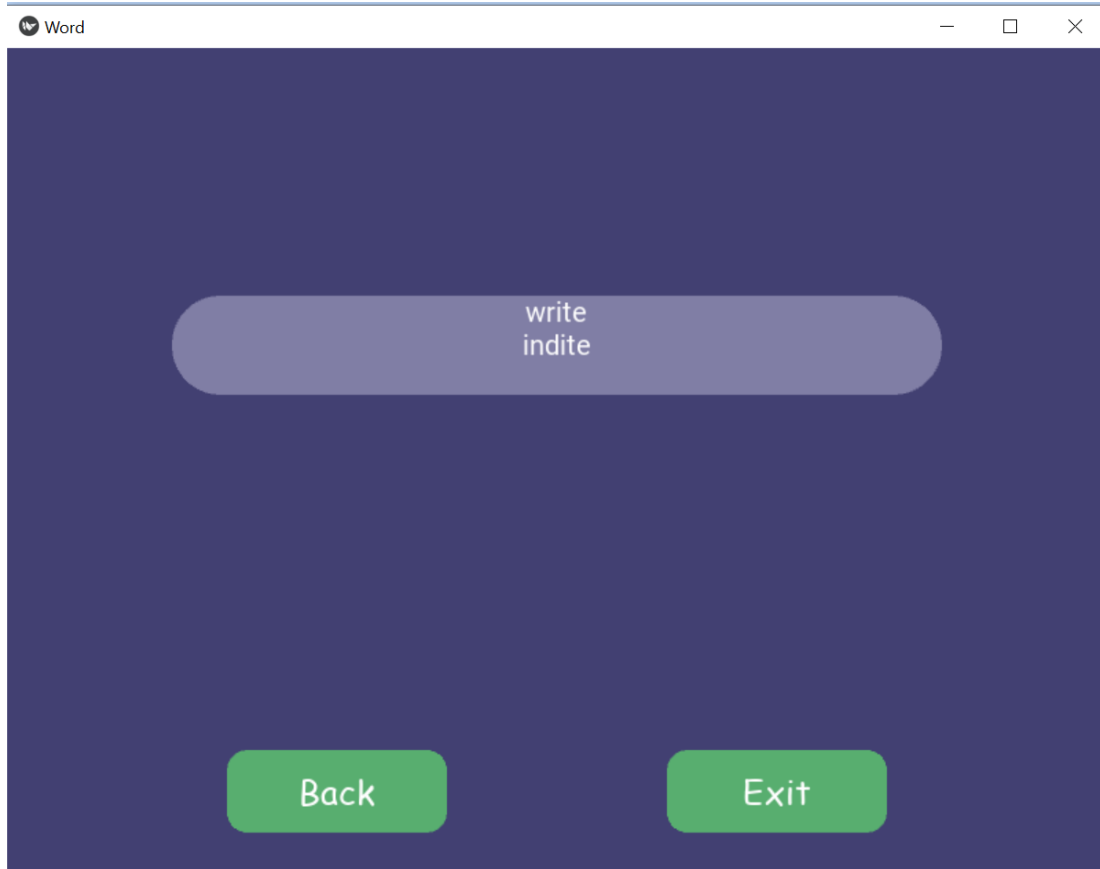


Fig 5.6 Rhymer Dictionary Output Screen

Hence the results comprise of the words that are related to literature and rhymes with Light.

If there are no words found, the application checks if there was spelling mistake and returns list of probable words.

For example if the meaning word is misspelled:

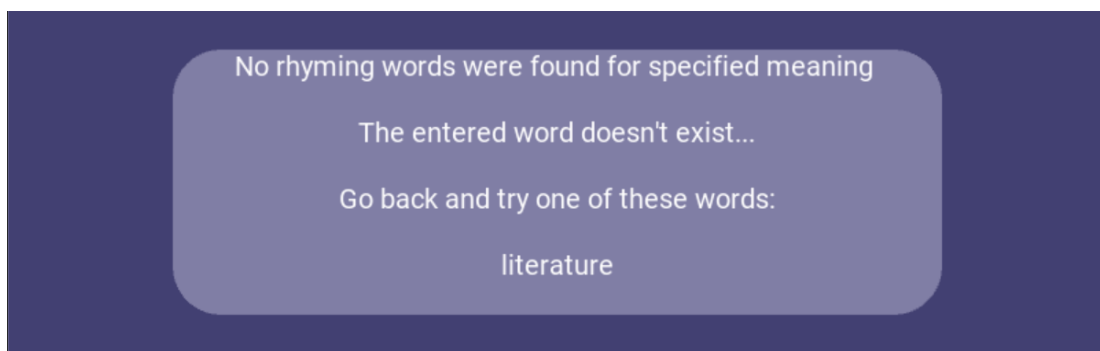


Fig 5.7 Rhymer Dictionary misspelled Output Screen

6.1.2 CROSSWORD HELPER

This section allows the user to find words for their crossword puzzle.

The user can enter the description they are looking for and they can define the length as well. The results are shown with an option to narrow them down by mentioning a fixed character and its position.

Logic Code with explanation in comments:

```
def cross(self, word, le):
    if(word==""):
        return "No word entered !"
    if(le.isdigit()):
        leng = int(le)
    else:
        return "Length wasn't entered !"
    rr=[]

    #Splitting the words and putting them in the API's format
    words=word.split()
    w=""
    for a in words:
        if(w==""):
            w+=a
        else:
            w+=" "+a

    fn=[]
    res= requests.get("https://api.datamuse.com/words?ml="+w)
    for a in res.json():
        fn.append(a["word"])

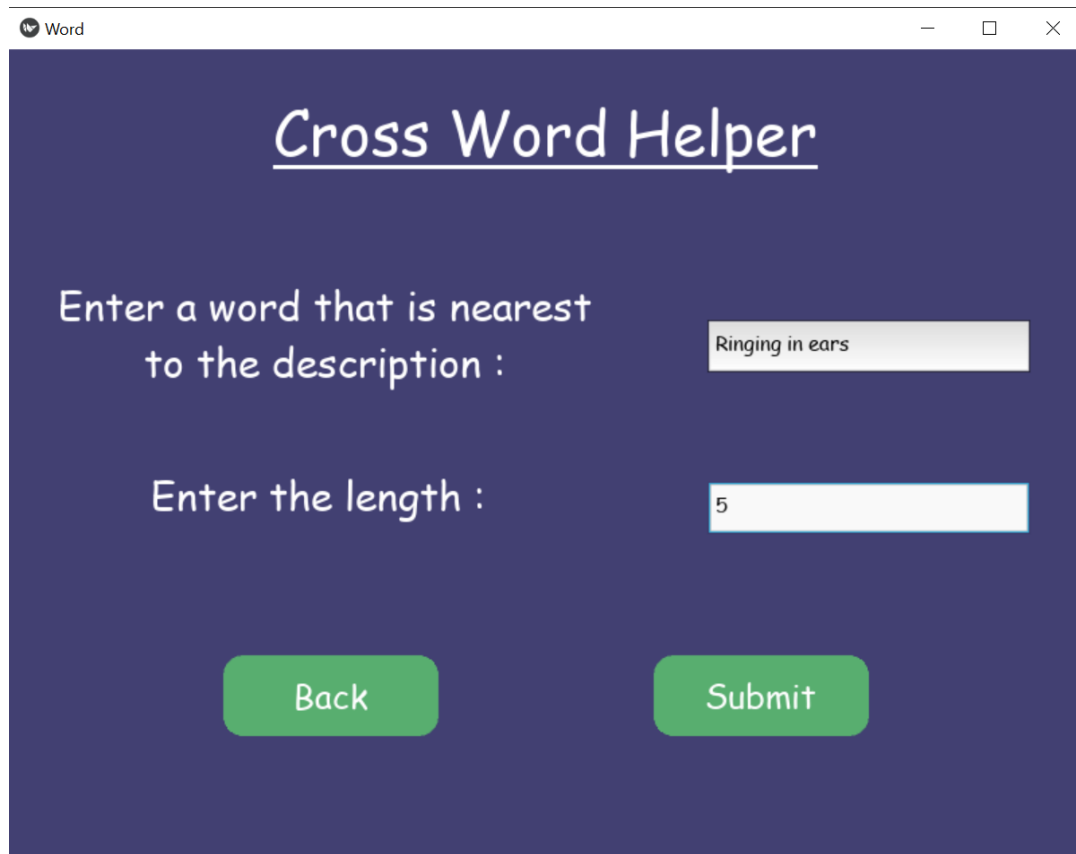
    #The words related to description are stored in fn now.
    #Finding words with defined length
    new = ""
    for l in fn:
        if len(l) == leng:
            new+=l+"\n"

    #returning error messages if no words are found.
    If(new==""):
        str="No words were found for the given inputs\n\n"
        if(spell.correction(word)!=word):
            str+="The entered word doesn't exist...\n\n"
            if len(spell.candidates(word))>0 :
                str+="Go back and try one of these words:\n\n"
                for s in spell.candidates(word):
                    str+=s+"\n"

        return str
    else:
        return new
```


The code above is incorporated in a GUI by using Kivy. Below are the snippets with the workflow:

1. Taking Input:



Word

Cross Word Helper

Enter a word that is nearest to the description :

Ringing in ears

Enter the length :

5

Back Submit

Fig 5.8 Crossword Helper Input Screen

2. Results :

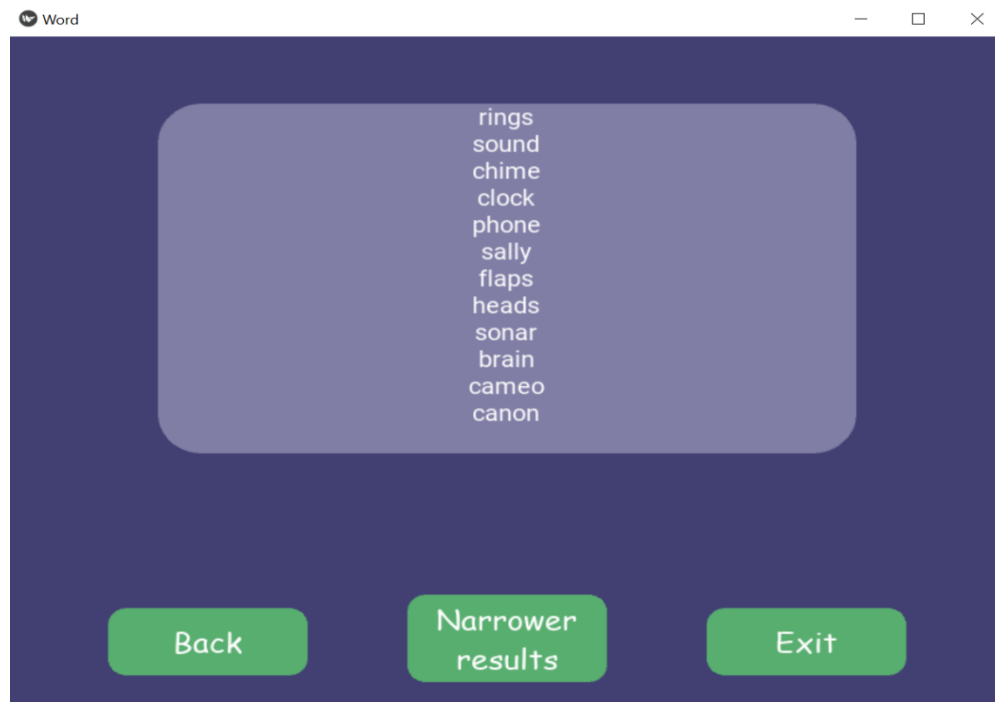


Fig 5.9 Crossword Helper Output Screen

3. Narrower Results Selection Screen:

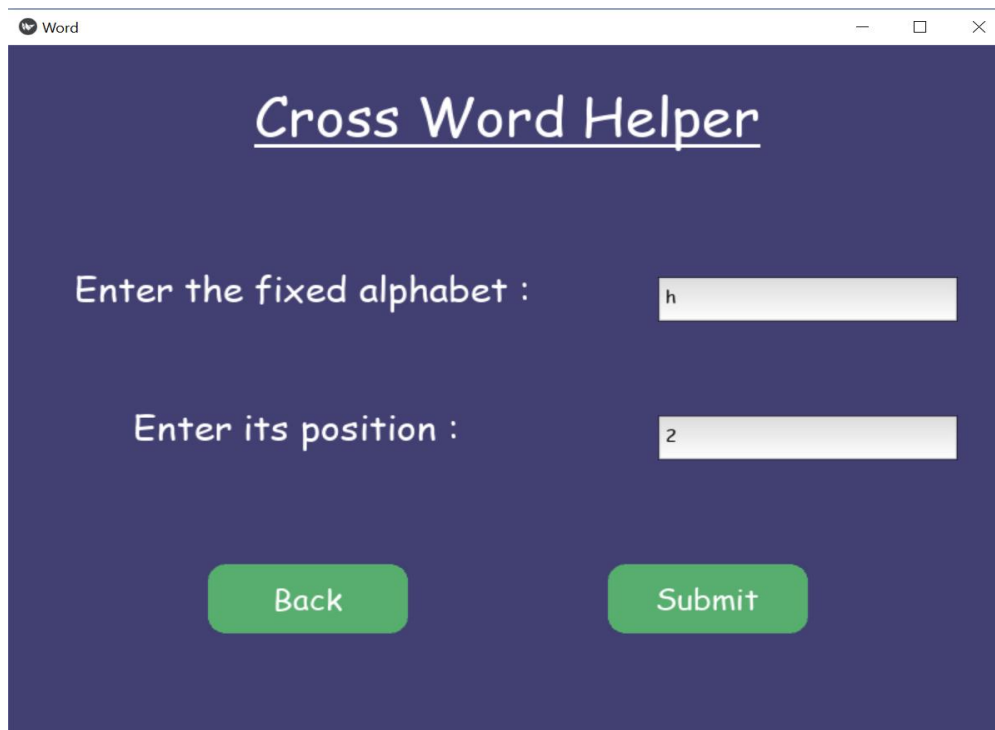


Fig 5.10 Crossword Helper Narrower Results Selection Screen

4. Narrow Results:

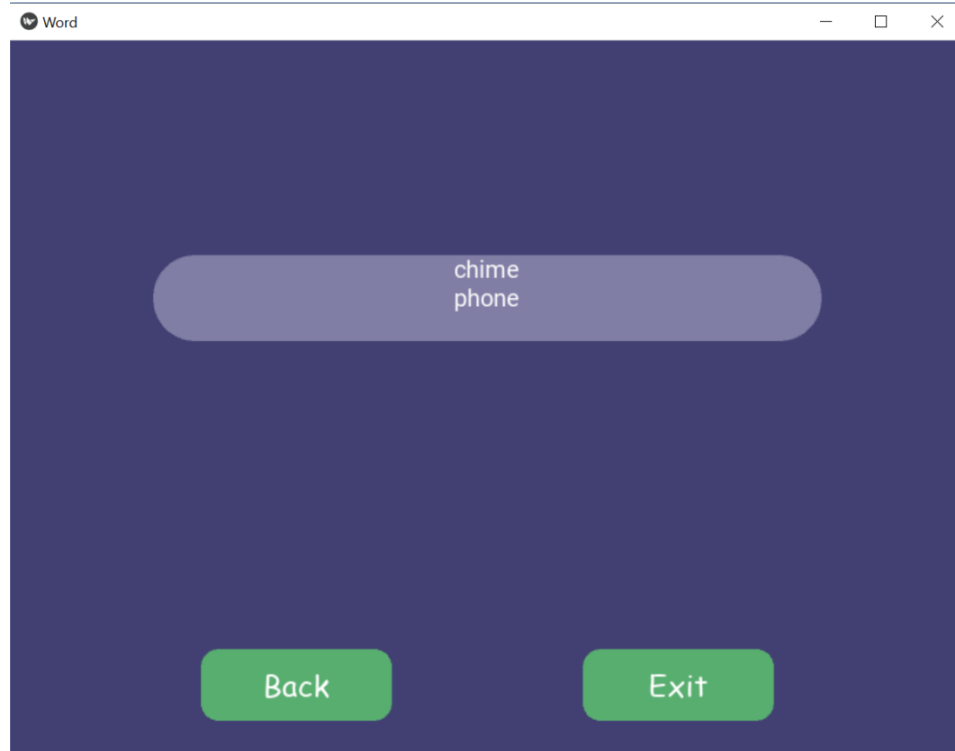


Fig 5.11 Crossword Helper Narrower Results Screen

Code for Narrow Results:

```
def narrow(self,s,ch,pos):
    if(s==""):
        return "No word entered"
    posi= int(pos)-1
    words=s.splitlines( )
    new = ""
    for w in words:
        # print(w[posi]+" ")
        if w[posi] == ch+"":
            new+=w+"\n"
    # print(new)
    if new == "":
        str="No words were found for the given inputs\n\n"
        if(spell.correction(s)!=s):
            str+="The entered word doesn't exist...\n\n"
            if len(spell.candidates(s))>0 :
                str+="Go back and try one of these words:\n\n"
                for s in spell.candidates(s):
                    str+=s+"\n"
        return str
    return new
```

6.1.3 SCRABBLE HELPER

This is the section where user can find words for their scrabble game.

The user can enter the word fragment they want the resultant words to have, its position and the length of the whole word.

Logic Code with explanation in comments:

```
def scrabble(self,w,p,l):
    if(w==""):
        return "No word entered"
    with open('words_alpha.txt') as word_file:
        33nglish_words = set(word_file.read().split())

    if(w==""):
        return "No Input"

    #Generating regex expression for all the cases:
    if(p=="^"):
        st=""^"+w
        if(l.isdigit()):
            le=int(l)-len(w)
            if(le>0):
                st = st+"{ "+str(le)+"}"
            else:
                return "Word Length can't be lesser than fragment length."

    Elif (p=="$"):
        st=w+"$"
        if(l.isdigit()):
            le=int(l)-len(w)
            if(le>0):
                st = "{ "+str(le)+"}" +st
            else:
                return "Word Length can't be lesser than fragment length."

    Elif(p==""):
        if(l.isdigit()):
            le=int(l)-len(w)
            if(le>0):
                st = ".*" +w+ ".*"
            else:
                return "Word Length can't be lesser than fragment length."

    Elif(p.isdigit()):
        pp = int(p)-1
        st = "{ "+str(pp)+"}" +w
        if(l.isdigit()):
            le=int(l)-len(w) - pp
            if(le>0):
                st = st+"{ "+str(le)+"}"

    #matching and finding the words that contain the specified fragment in
    the specified position
```

```

reg = re.compile(st)
match=list(filter(reg.match,34nenglish_words))
new = ""
for word in match:
    if(l.isdigit()):
        if(len(word)==int(l)):
            new+=word+"\n"
    else:
        new+=word+"\n"
if(new==""):
    return "No words were found, try entering different parameters."
Else:
    return new

```

The code above is incorporated in a GUI by using Kivy. Below are the snippets with the workflow:

1. Taking Input:



Word

Scrabble Helper

Enter the word fragment :

Enter its position (optional):

Enter the length (optional) :

Fig 5.12 Scrabble Helper Input Screen

2. Results:

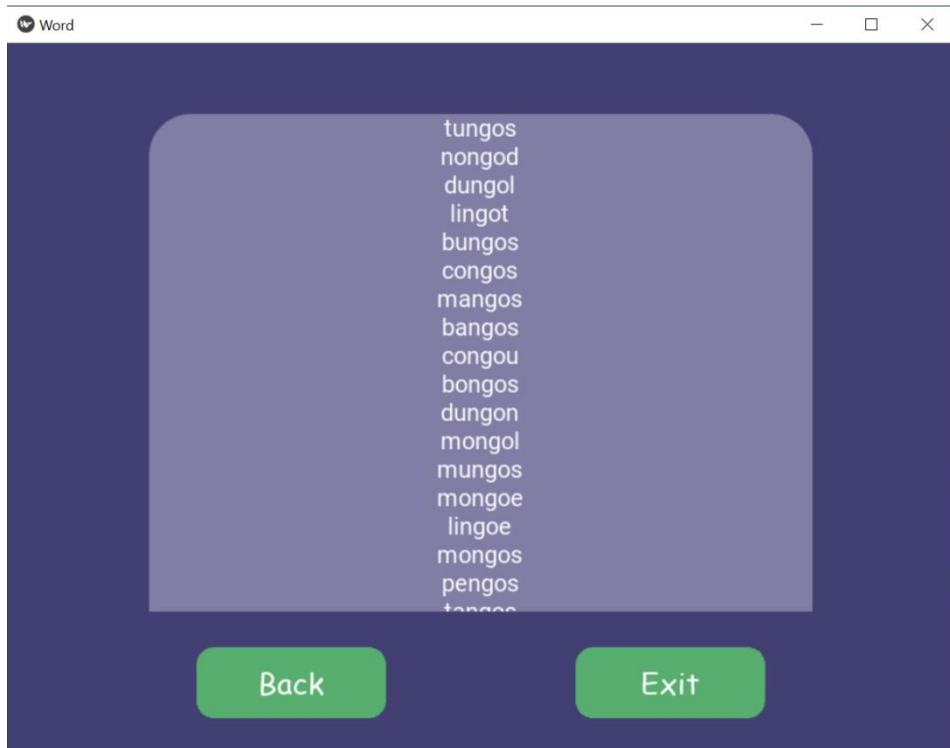


Fig 5.13 Scrabble Helper Output Screen

Hence the user gets the list of words with length as 6 and has “ngo” at third position, thereby aiding them to get variety of words for their scrabble game.

The application makes use of a file which contains around 370150 English words. It applies regular Expression to get the words with defined fragments at defined position of specifies length.

6.2 UNDERSTAND ZONE

Understand Zone is the section which allows user to understand a word that they enter.

The result is fetched from Oxford Dictionary's official API which is free for a limited number of hits.

Below is the code:

```
def mean(self,word):  
    if(word==""):  
        return "No word entered"  
    str=""  
    response= requests.get("https://od-  
api.oxforddictionaries.com/api/v2/entries/en-  
gb/"+word+"?strictMatch=false",headers={"Accept": "application/json","app_id":  
"c1498ba3","app_key": "ec959282e97d787344cbe7cfeb13c965"})  
    try:  
        for a in response.json()["results"]:  
            for b in a["lexicalEntries"][0]["entries"][0]["senses"]:  
                str+="Definition:\n "+b["definitions"][0]+" \n"  
                str+="Example: \n"+b["examples"][0]["text"]+" \n \n"  
    except KeyError:  
        str="No meanings were found for the given inputs\n\n"  
        if(spell.correction(word)!=word):  
            str+="The entered word doesn't exist...\n\n"  
            if len(spell.candidates(word))>0 :  
                str+="Go back and try one of these words:\n\n"  
                for s in spell.candidates(word):  
                    str+=s+" \n"  
    return str
```

The result comprises of multiple meanings along with supporting meanings.

If the user enters a wrongly spelled word, he is prompted with nearest correctly spelled words with the help of the spellcheck library.

Snippets and workflows:

1. Input:

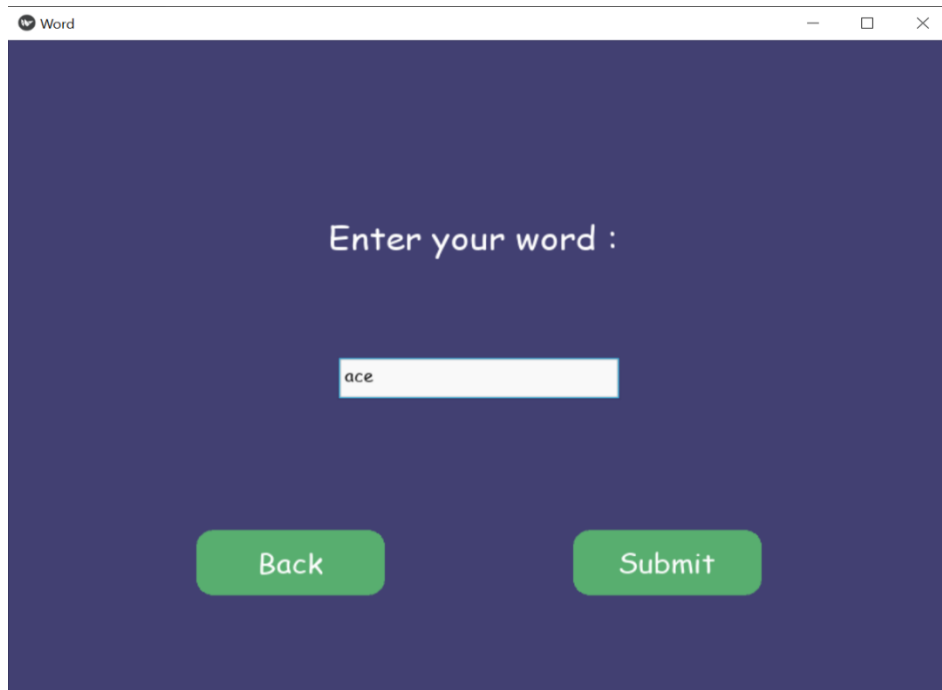
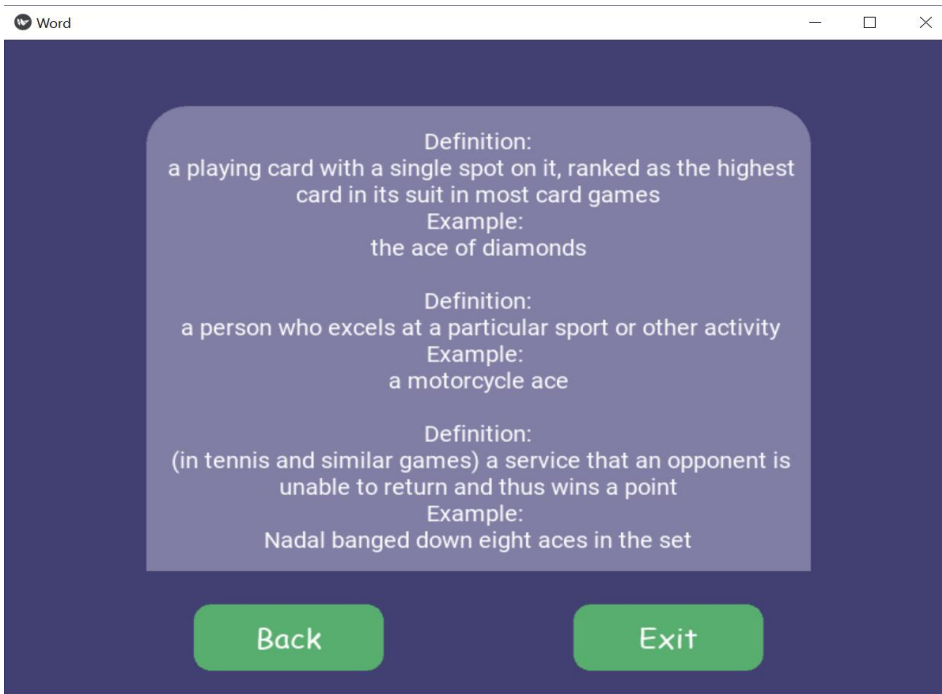


Fig 5.14 Understand Zone Input Screen

2. Output:



Definition:
a playing card with a single spot on it, ranked as the highest
card in its suit in most card games
Example:
the ace of diamonds

Definition:
a person who excels at a particular sport or other activity
Example:
a motorcycle ace

Definition:
(in tennis and similar games) a service that an opponent is
unable to return and thus wins a point
Example:
Nadal banged down eight aces in the set

Fig 5.15 Understand Zone Output Screen

6.3 KNOWLEDGE ZONE

Knowledge Zone is the section in which user can know more related words to a given word.

There are three sections inside Knowledge Zone where they can find synonyms, antonyms, hyponym, hypernym and holonyms.

All these sections make use of Wordnet which stores a lot of words related to a given word.

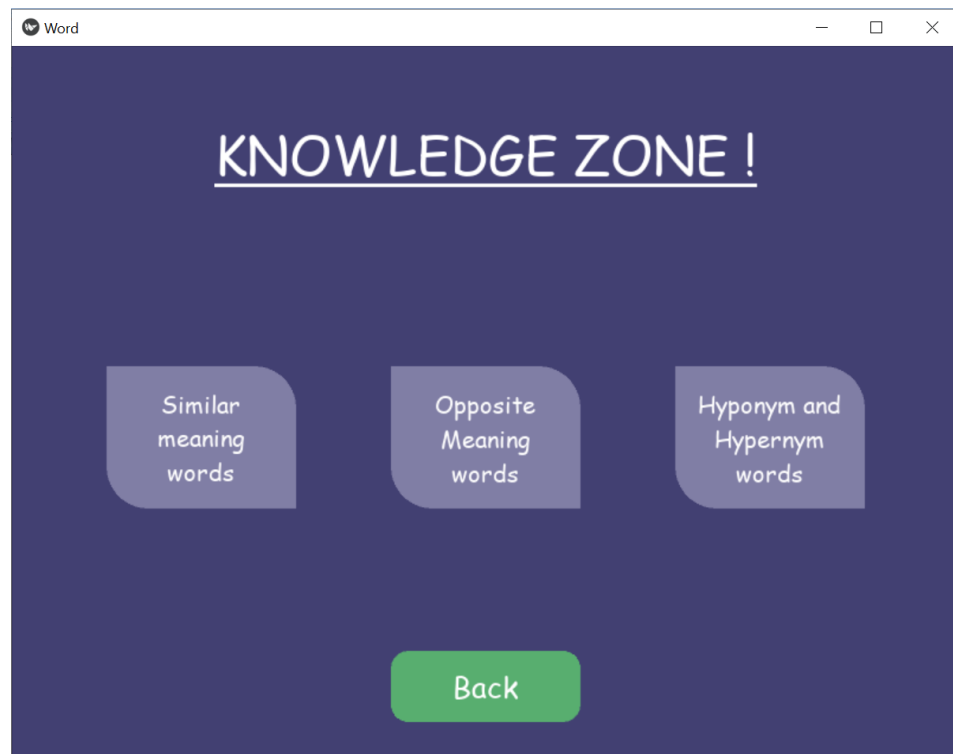


Fig 5.16 Knowledge Zone Screen

6.3.1 SIMILAR MEANING WORDS

User can find similar meaning words, i.e. synonyms of a word by entering a word in this section.

Code snippet:

```
def syn(self,word):
    synonyms = []
    if(word==""):
        return "No word entered"

    for syn in wordnet.synsets(word):
        for l in syn.lemmas():
            synonyms.append(l.name())

    if(len(synonyms)>0):
        ss=set(synonyms)
        new = ""
        for x in ss:
            new += x+"\n"
        return new

    else:
        str="No synonyms were found for the given input\n\n"
        if(spell.correction(word)!=word):
            str+="The entered word doesn't exist...\n\n"
            if len(spell.candidates(word))>0 :
                str+="Go back and try one of these words:\n\n"
                for s in spell.candidates(word):
                    str+=s+"\n"

    return str
```

If the user enters a wrongly spelled word, he is prompted with nearest correctly spelled words with the help of the “spellcheck” library.

Snapshots and workflow:

1. Input:

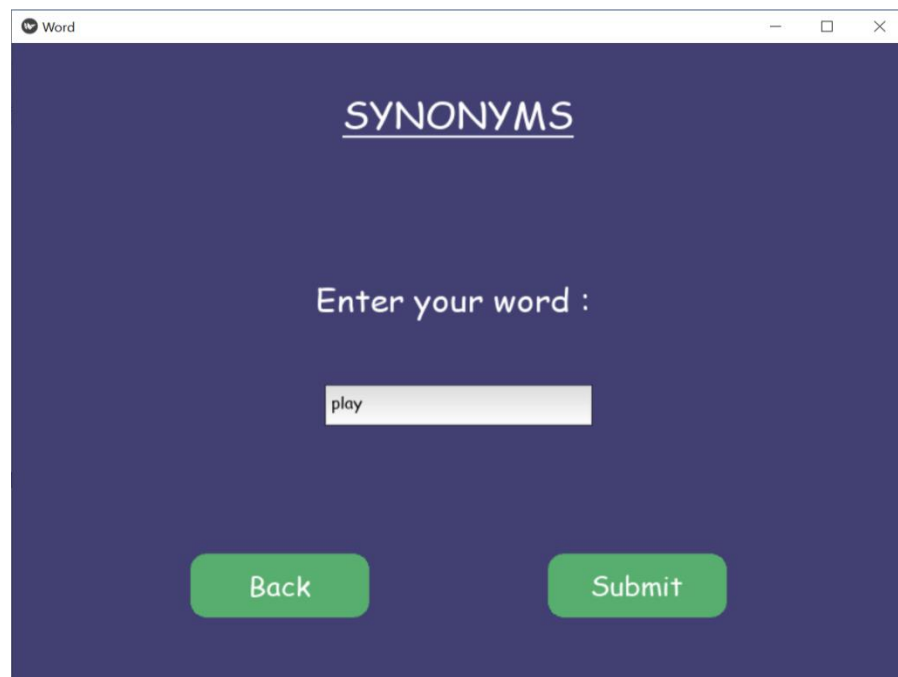


Fig 5.17 Knowledge Zone Synonyms Input Screen

2. Output:

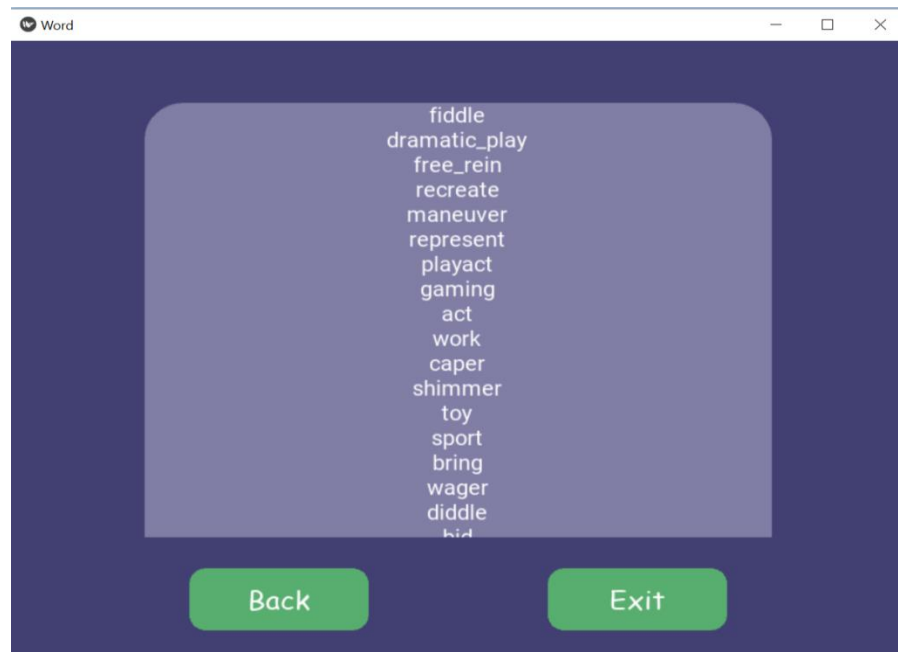


Fig 5.18 Knowledge Zone Synonyms Output Screen

6.3.2 OPPOSITE MEANING WORDS

In this section, user can find opposite meaning words, i.e. Antonyms by entering a word.

This also makes use of wordnet's synsets to find antonyms.

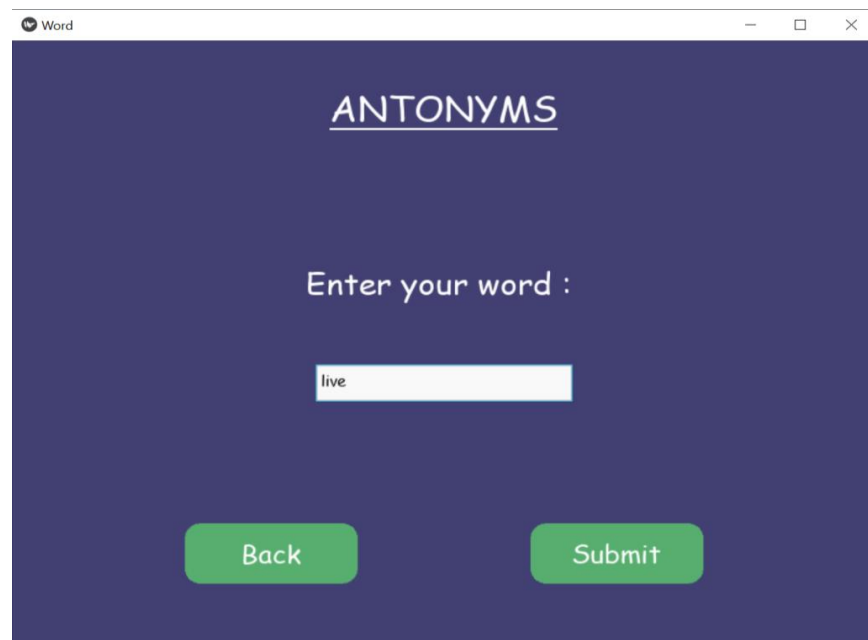
Code Snippet:

```
def ant(self,word):  
    antonyms = []  
    if(word==""):  
        return "No word entered"  
  
    for syn in wordnet.synsets(word):  
        for l in syn.lemmas():  
            if l.antonyms():  
                antonyms.append(l.antonyms()[0].name())  
  
    if(len(antonyms)>0):  
        ss=set(antonyms)  
        new = ""  
        for x in ss:  
            new += x+"\n"  
        return new  
  
    else:  
        str="No antonyms were found for the given input\n\n"  
        if(spell.correction(word)!=word):  
            str+="The entered word doesn't exist...\n\n"  
            if len(spell.candidates(word))>0 :  
                str+="Go back and try one of these words:\n\n"  
                for s in spell.candidates(word):  
                    str+=s+"\n"  
  
        return str
```

If the user enters a wrongly spelled word, he is prompted with nearest correctly spelled words with the help of the “spellcheck” library.

Snapshots and workflow:

1. Input:



Word

ANTONYMS

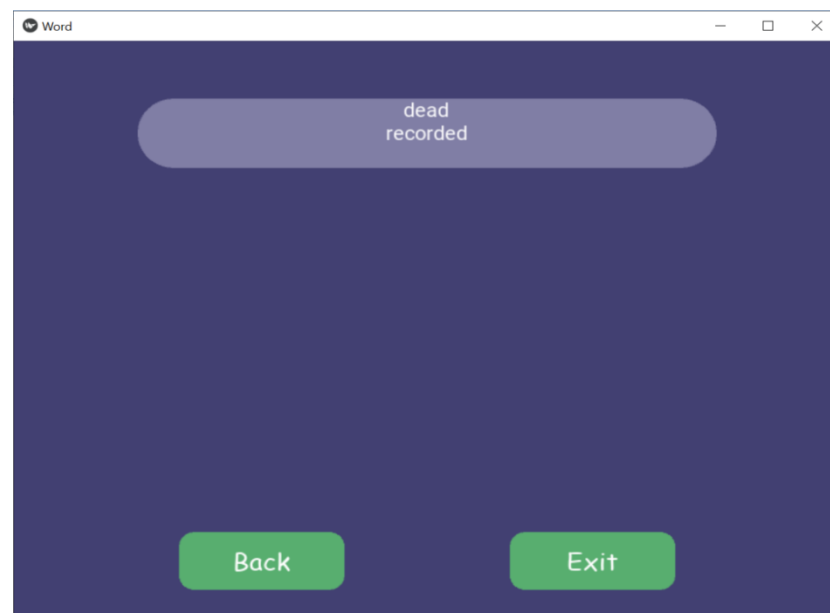
Enter your word :

live

Back Submit

Fig 5.19 Knowledge Zone Antonyms Input Screen

2. Output:



Word

dead
recorded

Back Exit

Fig 5.20 Knowledge Zone Antonyms Output Screen

6.3.3 HYPERNYM – HYPONYM – HOLONYM

This section allows user to explore hypernyms, hyponyms and holonyms of entered words.

Hypernym are the broader category term for a given word, for example ‘Red’, ‘Purple’, ‘Green’ all have a common hypernym as ‘Color’.

Hyponym are the narrow category term for a given word for example ‘Red’ is a hyponym of ‘Color’.

Holonym defines the relationship between a term denoting the whole and a term denoting a part of, or a member of, the whole. For example ‘Tree’ is a holonym of ‘Trunk’.

Code Snippet:

Hypernym:

```
def hyper(self,word):
    sn=""
    if(word==""):
        return "No word entered"
    syns=wordnet.synsets(word)
    for syn in syns:
        s=syn.hypernyms()
        for q in s:
            sn+=q.lemma_names()[0]+"\\n"
    if(sn!=""):
        return sn
    else:
        str="No words were found for the given input\\n\\n"
        if(spell.correction(word)!=word):
            str+="The entered word doesn't exist...\\n\\n"
            if len(spell.candidates(word))>0 :
                str+="Go back and try one of these words:\\n\\n"
                for s in spell.candidates(word):
                    str+=s+"\\n"
        return str
```

Hyponym:

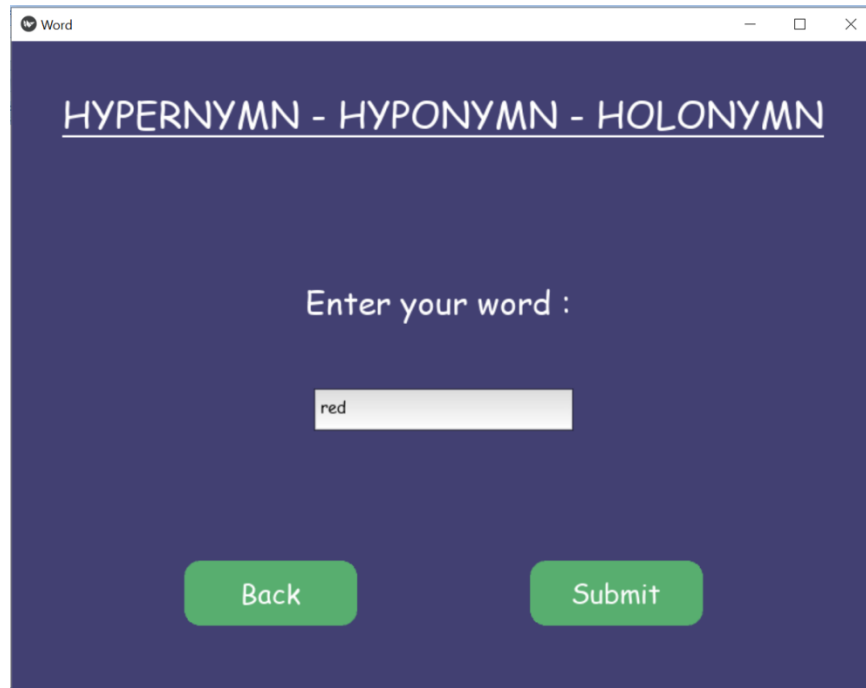
```
def hypo(self,word):
    sn=""
    if(word==""):
        return "No word entered"
    syns=wordnet.synsets(word)
    for syn in syns:
        s=syn.hyponyms()
        for q in s:
            sn+=q.lemma_names()[0]+"\\n"
    if(sn!=""):
        return sn
    else:
        str="No words were found for the given input\\n\\n"
        if(spell.correction(word)!=word):
            str+="The entered word doesn't exist...\\n\\n"
            if len(spell.candidates(word))>0 :
                str+="Go back and try one of these words:\\n\\n"
                for s in spell.candidates(word):
                    str+=s+"\\n"
        return str
```

Holonym:

```
def holo(self,word):
    sn=""
    if(word==""):
        return "No word entered"
    syns=wordnet.synsets(word)
    for syn in syns:
        s=syn.member_holonyms
        for q in s:
            sn+=q.lemma_names()[0]+"\\n"
    if(sn!=""):
        return sn
    else:
        str="No words were found for the given input\\n\\n"
        if(spell.correction(word)!=word):
            str+="The entered word doesn't exist...\\n\\n"
            if len(spell.candidates(word))>0 :
                str+="Go back and try one of these words:\\n\\n"
                for s in spell.candidates(word):
                    str+=s+"\\n"
        return str
```

Snapshots:

1. Input:



Word

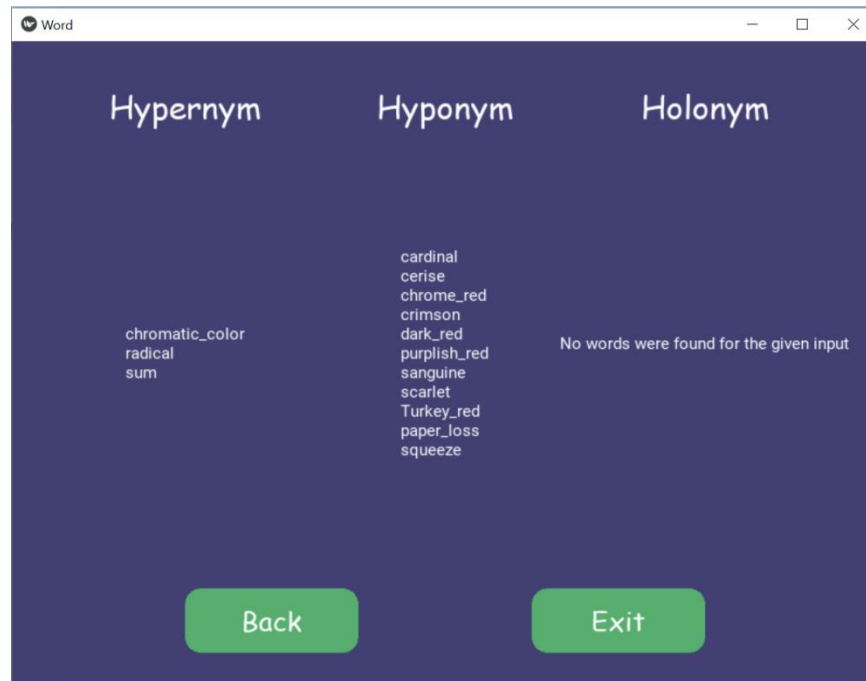
HYPERNYMN - HYPONYMN - HOLONYMN

Enter your word :

Back Submit

Fig 5.21 Knowledge Zone Hypernymn input Screen

2. Output:



Word

Hypernym	Hyponym	Holonym
chromatic_color radical sum	cardinal cerise chrome_red crimson dark_red purplish_red sanguine scarlet Turkey_red paper_loss squeeze	No words were found for the given input

Back Exit

Fig 5.22 Knowledge Zone Hypernymn Output Screen

6.4 LEARNING ZONE

Learning Zone is the section which has three sections which shows low, mid and high level words based on user's usage.

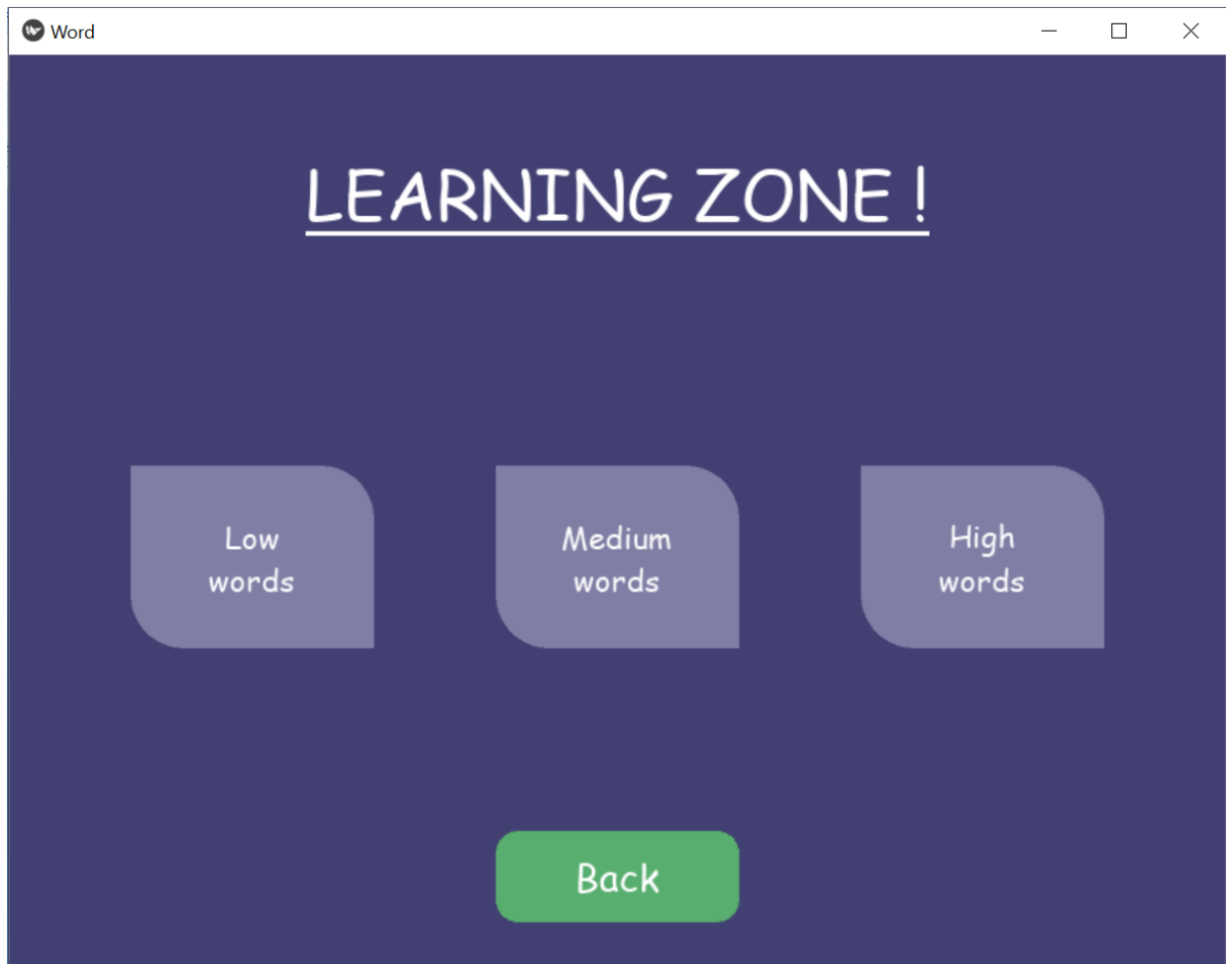


Fig 5.23 Learning Zone Screen

The levels are trained with respective user's usages:

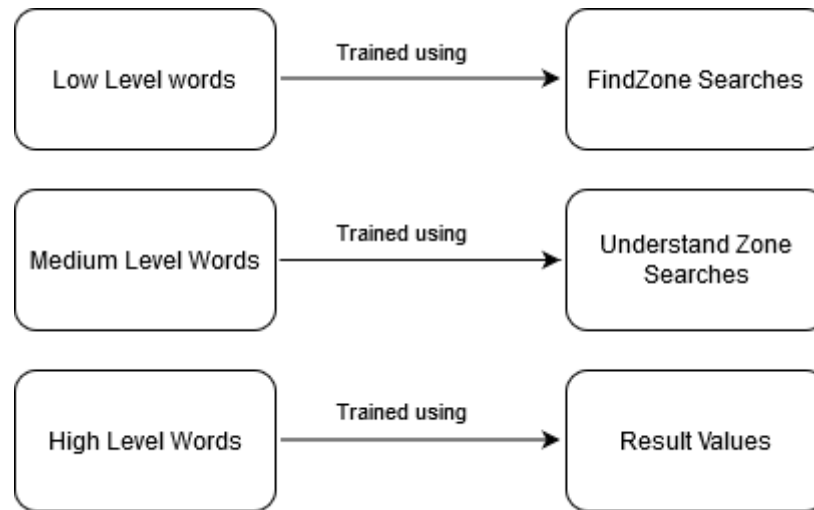


Fig 5.24 Learning Zone training data to learning section mapping

6.4.1 LOW LEVEL WORDS

Low level words are the easy words, Findzone searches are done using words which the user already knows, and hence these words can be used to show low level words.

6.4.2 MEDIUM LEVEL WORDS

Medium Level words are the words which the user genuinely wants to know, hence understand zone searches (where user enters the words that they want to understand) are used to train medium level model.

6.4.3 HIGH LEVEL WORDS

High level words are the words that are very less frequently used and are trained by the less frequently used words from results.

6.4.4 TRAINING and PREDICTION:

Training is done on the data stored in any of the three files:




 high.pkl	3/30/2020 6:50 PM	PKL File	1 KB
 low.pkl	4/20/2020 10:36 AM	PKL File	1 KB
 mid.pkl	4/14/2020 9:33 AM	PKL File	1 KB

Fig 5.25 Learning Zone Pickle files

These files contains the frequency of the words according to the mapping shown in previous section.

The toughness of a word is determined by how frequently it is being used in English language. It is determined using the library WordFreq. Hence frequencies of words from respective modules are stored in their respective pickle files which are then used to train respective models.

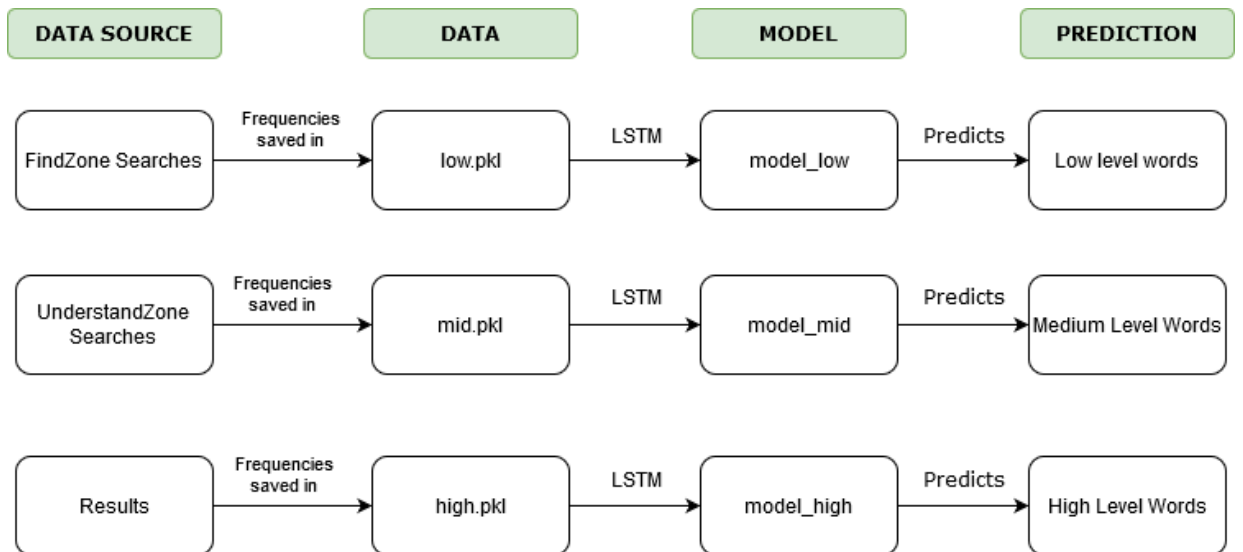


Fig 5.26 Learning Zone flow

Generic code for training:

```
def train(data_path):
    with open("./data/"+data_path+".pkl", 'rb') as f:
        data_list = pickle.load(f)

    df = pd.DataFrame(data_list, columns=['frequency'])

    frequency_train = df['frequency'].values.reshape((-1, 1))

    look_back = 5

    train_generator = TimeseriesGenerator(frequency_train, frequency_train, length=look_back, batch_size=20)

    from keras.models import Sequential
    from keras.layers import LSTM, Dense

    model = Sequential()
    model.add(
        LSTM(10,
            activation='relu',
            input_shape=(look_back,1))
    )
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')

    num_epochs = 25
    # train_generator = train_generator.reshape(-1, -1, 1)
    model.fit_generator(train_generator, epochs=num_epochs, verbose=1)

    with open('./models/model_' + data_path + ".pkl", 'wb') as f:
        pickle.dump(model, f)
```

The trained models can be retrained and are kept separately in folder.




 model_high.pkl	4/12/2020 10:04 PM	PKL File	13 KB
 model_low.pkl	4/14/2020 9:34 AM	PKL File	13 KB
 model_mid.pkl	4/14/2020 9:35 AM	PKL File	13 KB

Fig 5.27 Learning Zone model files

These models are then used to predict words using the predict function :

```
def predict(mtype, num_prediction=1):
    with open("./data/"+mtype+".pkl", 'rb') as f:
        data_list = pickle.load(f)
        model=None

    look_back=5

    with open('./models/model_' +mtype+".pkl", 'rb') as f:
        model=pickle.load(f)

    df = pd.DataFrame(data_list, columns=['frequency'])

    frequency = df['frequency'].values[-look_back:]

    prediction_list = frequency[-look_back:]

    for _ in range(num_prediction):
        x = prediction_list[-look_back:]
        x = x.reshape((1, look_back, 1))
        out = model.predict(x)[0][0]
        prediction_list = np.append(prediction_list, out)
    prediction_list = prediction_list[look_back-1:]

    return prediction_list
```

Chapter 7

Result and Discussion

7.1 RESULTS

WORDZONE can successfully return results for user's word related queries and makes use of the user's usage to help them learn at a personalised level.

Learning Zone has three models built using LSTM below are the RMS values found during testing:

Model for Low words	
Predicted	Actual
5.15	5.37
5.63	6.05
5.32	6.12

Table 7.1 Low Model Actual vs Prediction test

Model for Mid words	
Predicted	Actual
3.06	4.12
3.69	3.38
3.45	3.84

Table 7.2 Medium Model Actual vs Prediction test

Model for High words	
Predicted	Actual
2.6	3.12
1.433	3.47
2.66	3.74

Table 7.3 High Model Actual vs Prediction test

Hence Final RMS Error for the three models are:

Model	RMS Error
Low	0.5369047
Mid	0.6762149
High	1.3645634

Table 7.4 RMS Error for all models

7.2 FUTURE WORK

There is always room for improvement, hence to improve WordZone I brainstormed and asked few seniors, friends and colleagues. All these resulted in a set of points which can be implemented in future work.

Here are few of the brainstormed ideas:

- More details can be shown in the results of each section, like their part of speech and varied usage.
- WordZone should be modified in such a way that it auto trains after entry of substantial amount of new data.
- Research on whether old data should be deleted or kept for predicting words as per user's current vocabulary.
- Research on finding better feature than frequency of a word in English Language for getting toughness of a word from vocabulary point of view.
- Since Kivy can be used for Mobile App development also, making an app and publishing it is also a viable future work.

Chapter 8

Summary

Wordzone offers a lot of word related usages, a user can find words they are looking for, understand words, know more about a particular word and learn new words according to their vocabulary.

Features like Crossword helper, Scrabble helper and Rhymer dictionary in Find Zone can not only be used while playing games but also be used for fun and general curiosity. Understand Zone acts as a dictionary which also gives out examples for different definitions, thereby allowing user to understand the entered word completely.

Knowledge zone quenches user's curiosity by showing them more related words to the entered word by showing synonyms, antonyms, hypernym, hyponym and holonym.

Learning Zone which has three sections trained on the user's data searches suggests new words for learning at a personalised level for each user.

In conclusion, this capstone project can successfully help users with returning results to user's different word queries, can build model on the data generated by its usage and help users learn new words on basis of it.

Chapter 9

References

- [1] Liu Vinci & Curran James (2006), Web Text Corpus for Natural Language Processing.
- [2] Loper, Edward & Bird, Steven. (2002). NLTK: the Natural Language Toolkit. CoRR. cs.CL/0205028. 10.3115/1118108.1118117.
- [3] Miller, George & Beckwith, R. & Fellbaum, Christiane & Gross, Derek & Miller, Katherine. (1991). Introduction to WordNet: An On-line Lexical Database*. 3. 10.1093/ijl/3.4.235.
- [4] Rajani S, M. Hanumanthappa, 2016, "Techniques of Semantic Analysis for Natural Language Processing – A Detailed Survey"
- [5] Dr. M Hanumanthappa, Rashmi S, Jyothi N M, "Impact of Phonetics in Natural Language Processing: A Literature Survey", IJISSET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 3, May 2014.
- [6] Primoz Podrzaj . A brief demonstration of some Python GUI libraries Proceedings of The 8th International Conference on Informatics and Applications ICIA2019, Japan, 2019
- [7] Hassanin M. Al-Barhamtoshy, Fatimah M. Mujallid, "Building Mobile Dictionary" 2013
- [8] Byrd, Roy & Chodorow, Martin. (2002). Using An On-Line Dictionary To Find Rhyming Words And Pronunciations For Unknown Words. 10.3115/981210.981244.
- [9] Time Series forecasting - Siami Namini, Sima & Tavakoli, Neda & Siami Namin, Akbar. (2018). A Comparison of ARIMA and LSTM in Forecasting Time Series. 1394-1401. 10.1109/ICMLA.2018.00227.
- [10] Robyn Speer. Wordfreq is a Python library for looking up the frequencies of words in many languages, based on many sources of data.