**Laguna College**

**San Pablo City**

**ITE Department**

**QuiliBot**

**Presented as partial fulfillment of the requirements for**

**ITE P402-1 Natural Language Processing using Python**

**January 18, 2023**

**Instructor: Gracelyn Ramos**

**Ayes, Giomel A.**

**Del Mundo, Ricsean Earl C.**

**Espinili, Jules Carlo Q.**

**Quiliope, Paul William V.**

**BSCS – 4th Year**

**Table of Contents**

|  |  |
| --- | --- |
| Title | Page Number # |
| Chapter 1 - Introduction | 3-8 |
| * Abstract | 3 |
| * Purpose | 4 |
| * Benefits and Impact | 5 |
| * Definitions | 6 |
| * Goals and Objectives | 7 |
| * Scope and Limitations | 8 |
| Chapter 2 – Review of Related Systems | 9-11 |
| * QuillBot.com | 9-10 |
| * Paraphrasing-Tool | 10 |
| * Spinbot.com | 10-11 |
| Chapter 3 – Life Cycle Method | 12-13 |
| * Software Development Life Cycle | 12-13 |
| Chapter 4 – Flow of Program | 14-19 |
| * Screen Layout | 14-18 |
| * Flow Chart | 19 |
| Chapter 5 – Results | 20-21 |
| * Conclusions | 20 |
| * Recommendations | 20-21 |
| * References | 21 |
| Appendices | 22-39 |
| * User Manual | 22 |
| * Source Code | 23-35 |
| * Curriculum Vitae | 36-39 |

**Chapter 1 – Introduction**

**Abstract**

The QuiliBot focuses on a number of components that will help individuals, especially students, write or verify their work by rephrasing, pointing out incorrectly spelled words and grammar, offering recommendations for corrections, and defining modified words. Additionally, it provides word synonyms. The total number of words is also counted. This application differs from other paraphrasing tools in that it does more than just paraphrase the text that is given. The QuiliBot aims to make it simpler to write accurate text. Using various words and phrases to convey the same ideas is known as paraphrasing. Short passages of text, such as phrases and sentences, are paraphrased. Short passages of text, such as phrases and sentences, are paraphrased. This program has a lot to give in addition to tools for paraphrase, so many individuals will benefit from it. The user can vary the use of direct quotations and include source material in their assignments by using this program or application. When paraphrasing, you swap out the author's words for new ones and synonyms. We incorporate the use of synonyms into the program for this reason.

**Purpose**

The user can paste the entire text into the tool and wait for the procedure to complete, making this program user-friendly. This program is crucial because it will enable students to comprehend the provided text or source, make their own corrections, and write it in their own terms. This training will be beneficial to job applicants as well as students. When a user is writing an email, proper grammar is equally crucial. Most employers today have a tendency to lose faith and confidence in an applicant's talents when they receive communication full of spelling and punctuation mistakes. You could come off as disorganized and sluggish if you use poor grammar. One of the key goals of this program is to ensure that you comprehend the source material well enough to write about it in your own words.

All users of this application can benefit from a variety of features. For instance, it reduces the likelihood of plagiarism. Search engines can punish you severely for plagiarism, and altering your material lowers the likelihood that it will be found to be plagiarized. More people are impressed by flawless grammar than by flowery words and an overused vocabulary. In actuality, they're more inclined to scan than read such stuff. The user may quickly begin their writing while utilizing a paraphrase tool. Because the user has a head start on what they are composing, this application saves time and can increase productivity. Last but not least, this application is a strong tool that enables users to quickly rewrite original content from the source material.

**Benefits and Impact**

Utilizing a paraphraser has benefits and a significant impact since it might lessen the likelihood of plagiarism, which is undesirable. We might be able to write faster with its assistance. A brand-new version of your work might be created by the computer in under a minute. It won't take long for these instruments to start working. The majority of these rewriting solutions also have a straightforward user interface, are compatible with a broad range of other web-based programs, and offer easy and user-friendly interfaces. Tools for paraphrasing improve the caliber of previously written information, but they may also aid in the creation of new content. The structure of the phrase makes it appear as though the reader is receiving low-quality information since it is not clear enough. Students may gain from using a paraphrase tool to correct grammatical errors in their writing. The phrases that the program generates are written accurately and are grammatically and semantically equivalent to the originals. When writing effectively, sentence structure must be maintained. Although it is preferred, it is not always possible to achieve in practice a link between the sentences that come before and after another in the text. By generating fresh sentence structures, the paraphraser will assist students in improving their academic writing skills.

**Definitions**

* Paraphrase - a rewording of something written or spoken by someone else.
* Paraphrasing tool - used to rewrite or rephrase a sentence without altering its meaning. This is accomplished by substituting any number of alternate versions for specific words, phrases, sentences, or even whole paragraphs to create a slightly different variant.
* Synonyms - word having the same or almost the same meaning as another word in the same language.
* Grammar - the whole system and structure of a language or of languages in general, usually taken as consisting of syntax and morphology (including inflections) and sometimes also phonology and semantics.
* Plagiarism - the practice of taking someone else's work or ideas and passing them off as one's own. Presenting someone else's work or ideas as your own, with or without their consent, by incorporating it into your work without full acknowledgement. All published and unpublished material, whether in manuscript, printed or electronic form, is covered under this definition.

**Goals and Objectives**

By paraphrasing the text, the QuiliBot attempts to be able to take the users' input and modify it into a different and better form. The software was written in the Python programming language, and it will take the text that the user has entered, analyze its format and content, and then paraphrase it, correct any errors, and change it from the text that the user has previously entered.

The goals of this project are the following:

* To avoid plagiarism by rephrasing the text that users provide.
* To review the supplied text's spelling and correct any mistakes
* To verify the entered text for grammatical mistakes
* To offer more other suggestions for synonyms in the paraphrased text.
* To make a user-accessible, free paraphraser available.

**Scope and Limitations**

The application is designed for users who are preparing documents or undertaking initiatives that call for writing papers or conducting research for presentations. These users, whose occupation is to provide reports and records for the research, might range from younger persons to senior people. Other targeted users may be those who work in communication or reporting, both of which need the user to produce documents with amended content that also adheres to proper spelling and grammatical rules. Students are the program's primary users since they frequently have tasks that call for producing documents or reports. This software can assist those users in preventing text plagiarism and producing academically sound papers.

The software merely does the following tasks: paraphrasing the text, examining the program for grammatical problems, verifying the spelling of the words, showing alternative word meanings, and tallying the total amount of words. Since these were the functions that were considered by the developers, other functions that are not stated are not included in the system. By entering their text in the input box and pressing the enter key, the user may have the computer produce the paraphrased text in the output box. Utilizing Visual Studio on the laptop, the software is written in the Python programming language. Before utilizing the software, you should download and install specific packages and models since they include the core functionality of the application. The program had to archive and pack the application; thus, it is not entirely installed in the system.

**Chapter 2 – Review of Related Systems**

1. **QuillBot.com**

There are seven different paraphrase styles available on Quillbot.com: formal, basic, creative, expand, and abbreviate. The usage of standard and fluency modes is totally free. Users must, however, pay for the premium service in order to use the formal, basic, creative, expand, and shorter modes. There are several QuillBot modes available. The default QuillBot.com mode is standard. It maintains meaning and aims to sound as natural as possible while balancing any changes it makes to the source text. The second mode on QuillBot.com is fluency. Fluency Mode employs a text-to-speech algorithm that produces language that is as grammatically and stylistically accurate and as naturally right in English as feasible, while also generating the fewest errors possible and altering the content without changing the meaning.

Formal mode changes the tone of your writing to make it sound more authoritative. attendees at formal events. For scholarly publications and corporate reports, it is very useful. The fourth mode is simple mode. The goal of simple mode is to provide you as much flexibility as possible while editing the text. The overall coherence and relevancy of the content may be impacted by this. The fifth option is the creative mode. While in creative mode, QuillBot's artificial intelligence (AI) is more perceptive and comprehends things like common sayings and phrases in the language. Expanding mode is the sixth and final mode. The expand mode is most appropriate for manuscripts with a large word count since it aims to prolong the text by adding as many words as feasible. The sixth selection on QuillBot.com is short mode. When you need to reduce the overall text length or word count, this method of brevity aims to condense your information without losing its meaning. The paraphraser application that is created by the developers contains the same features as QuillBot, such as word count, grammatical advice, suggestions for incorrect spelling, and a list of synonyms.

**2. Paraphrasing Tool.com –**

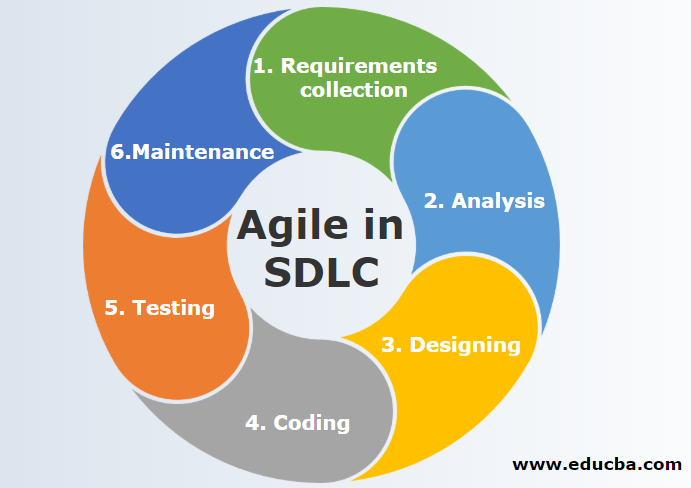
Paraphrasing-Tool.com use intelligent decision-making tools to identify the most appropriate way to rewrite or paraphrase the text being rewritten. The websites of Paraphrasing-Tool.com, in contrast to QuillBot.com, are straightforward and only provide one method of paraphrasing. It didn't have the qualities of quillbot.com. Simply said, this website provides a free paraphrase generator. The websites claim that paraphrasing-tool.com works like an automated thesaurus since it just uses synonyms and doesn't produce new text structures as a result of their use. In addition, it is stated previously in the first example, our paraphraser offers more features than the paraphrasing tool. The paraphrase tool is lacking in features, as opposed to the paraphraser application that is created by the developers, which is extendable and offers a variety of possibilities according on what you want to achieve.

**3. Spinbot.com**

The article spinner on Spinbot.com rewrites human-readable content into additional language that is simple to grasp. It's a machine that spins articles for free and may be used to create fresh material. It is a tool for rewriting articles that uses a marked-up version of the text to show which passages need to be modified or rearranged. Spinbot.com offers a service that is totally free. Spinbot.com generates several variations of the fundamental text while preserving its original meaning. On Spinbot.com, you can find text spinners, a paraphrasing tool, and a Translate and Spin feature. One of the most often utilized features is text spinners. Spinbot.com is a straightforward paranormal platform. In the same way as the previous two instances, using spinbot.com is as easy as using the paraphrasing tool. Unlike QuillBot and our paraphraser, which have several choices to pick from depending on your intended outcome, the Spinbot is as simple as the paraphrasing tool.

**Chapter 3 – Life Cycle Method**

**Software Development Life Cycle**



**Agile Method**

**Stage 1: Requirements Collection**

The purpose of this paper is to identify existing pain issues that software developers should work to solve. It might be a helpful tool for the team to find novel ways to alter and enhance their products.

During this phase, we learn more about the key characteristics of the program we have selected. To make the application even more beneficial for many users, we also offer more functions. We determine the strategies required to carry out our program.

**Stage 2: Analysis**

Team members collaborate to discuss and plan throughout this first development phase.

Our goal with this project is to provide users with a single tool that will enable them to generate texts and articles without worrying about grammatical usage. The prerequisites for our project are planned. We consider the potential problems and dangers. We search for this program's opportunities as well.

**Stage 3: Designing**

This phase focuses on product design.

At this step, we generate a few ideas before selecting the one we believe would work best for our program. This is the stage where the program's interface and functionalities are being developed by the program's developers. We examine the plan and evaluate its quality. The developers will create the program in the python language

**Stage 4: Coding**

The product is constructed in Stage 4 as production gets under way. To build the product as quickly and efficiently as possible, the programming code is created in accordance with the Designing Stage. The code is created by developers using a variety of tools and programming languages. These are chosen in accordance with the requirements of the software being created.

We now begin the programming. Python is the programming language we employ.

**Stage 5: Testing and Deployment**

In stage 5, the development team checks the program for flaws and mistakes. Does the software deliver the desired outcomes? Does it meet the basic goals and criteria defined in the SDLC? These are some illustrations of important inquiries that may be made throughout the testing stage.

Now that we have attempted to execute our program, we have looked for any potential issues and have corrected them ahead of time. The developers then deploy the program for testing by different users. The users that tested the program are the other group of developers in which they test the interface and the functions of the application.

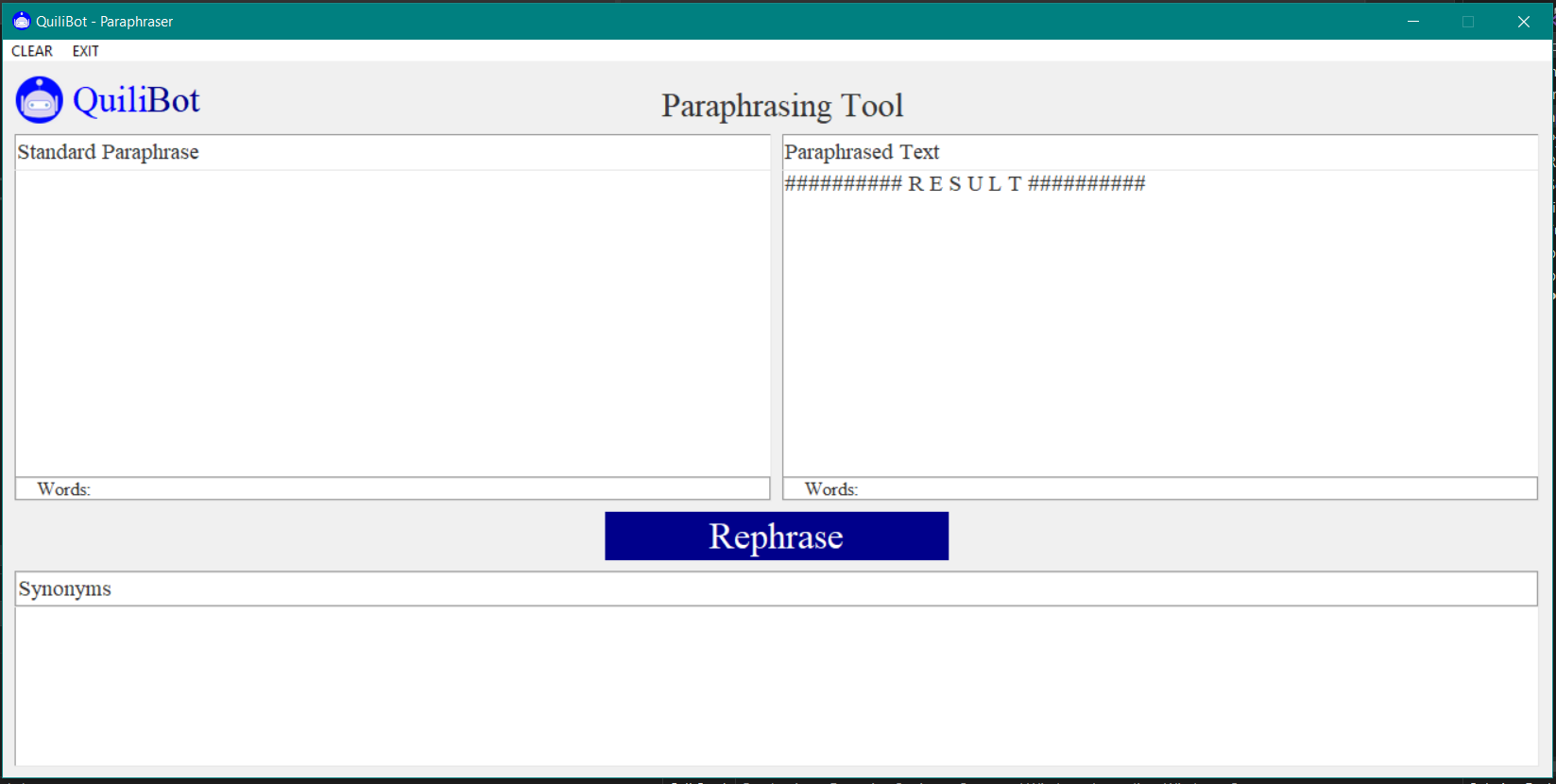
**Stage 6: Maintenance**

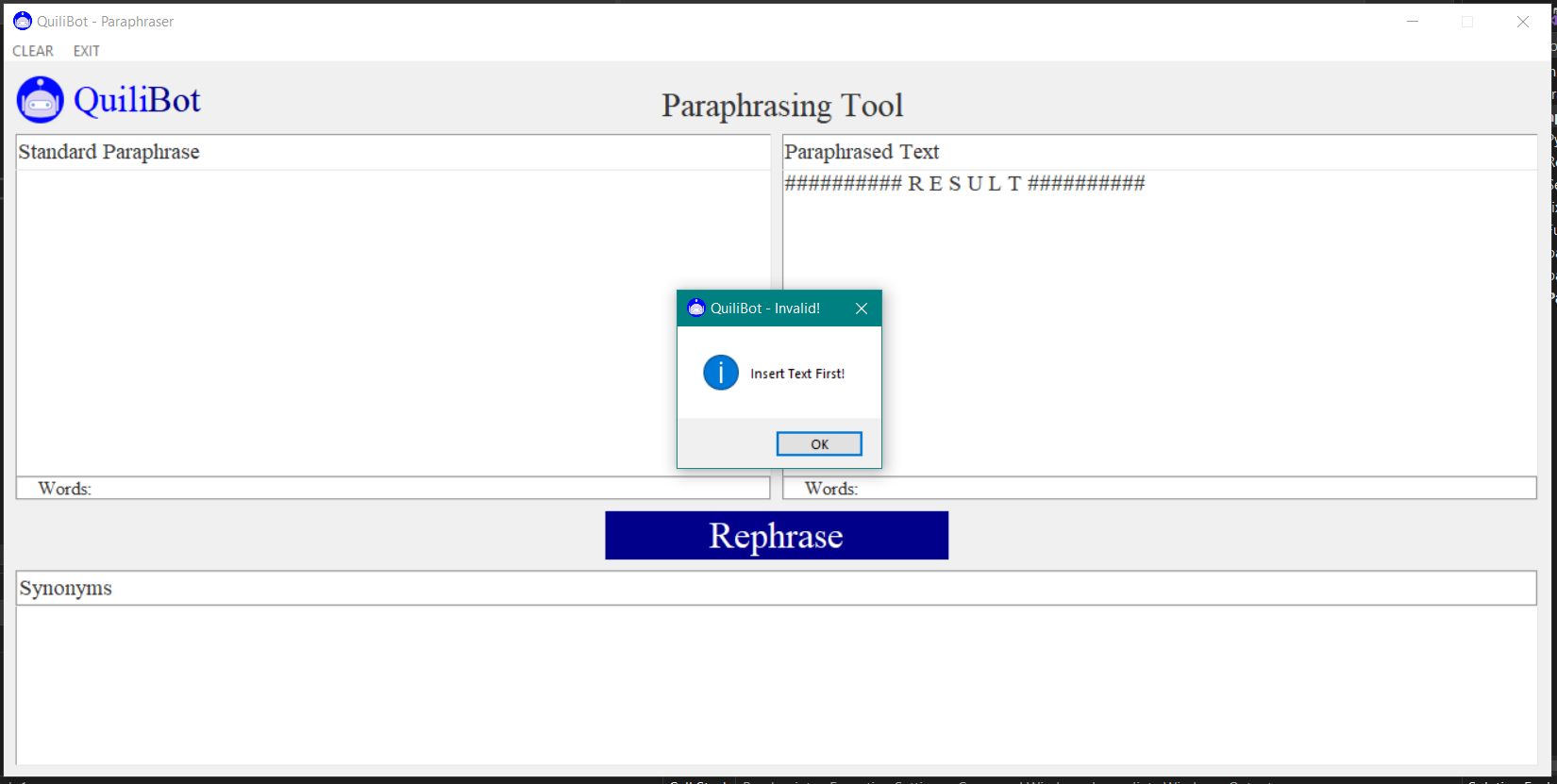
Each consumer uses software products differently since they have various demands, thus there can be certain problems that arise and need to be fixed. In this maintenance stage, these user issues are resolved.

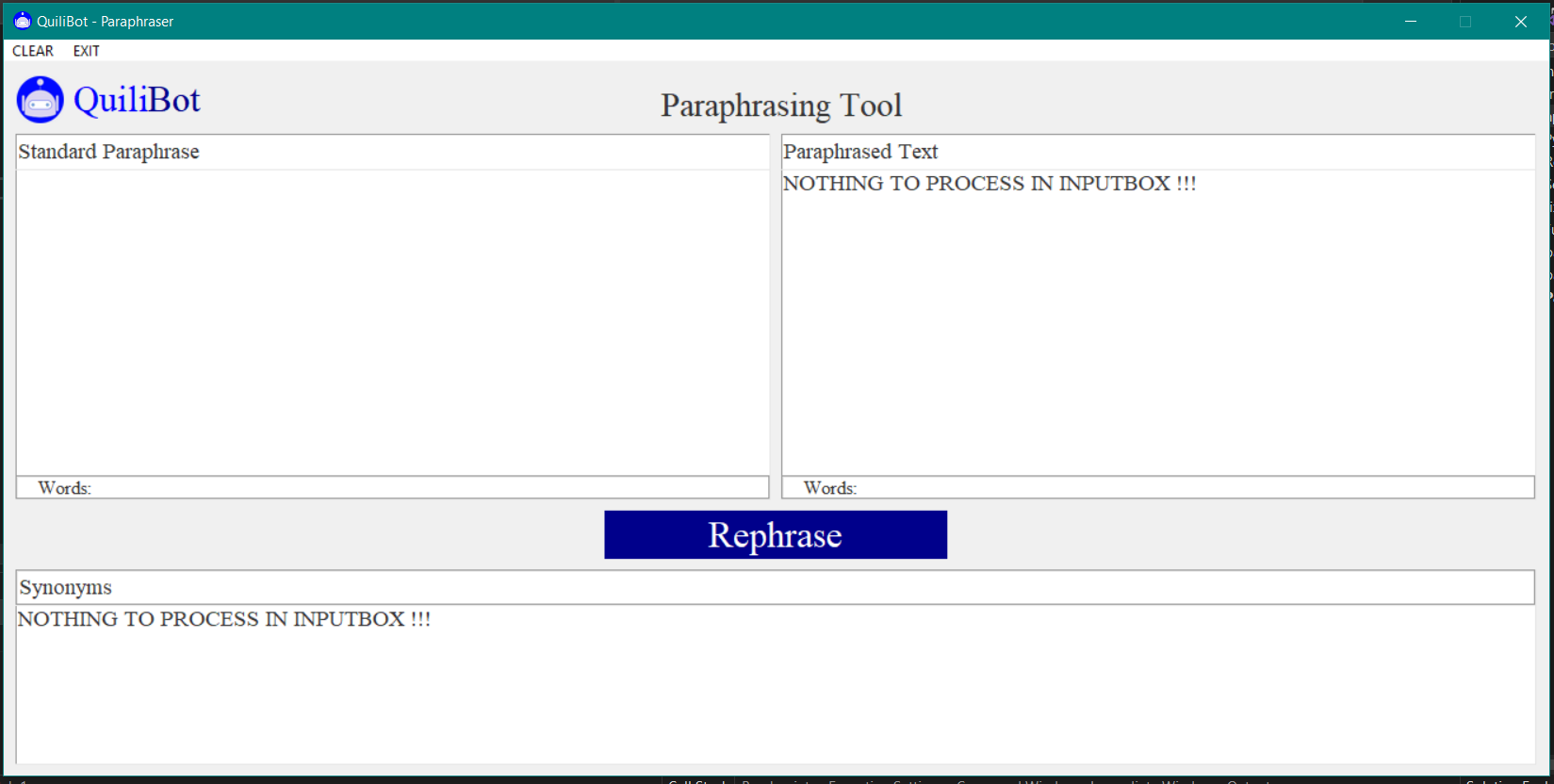
If the user or the program's user needs to add something or if they ran into any problems, they may do so at this last stage. In order for them to have the greatest experience, we correct it. This is the phase where in the errors and bugs that are pointed out and noticed by the users are fixed in order to improve its efficiency and usability of the program.

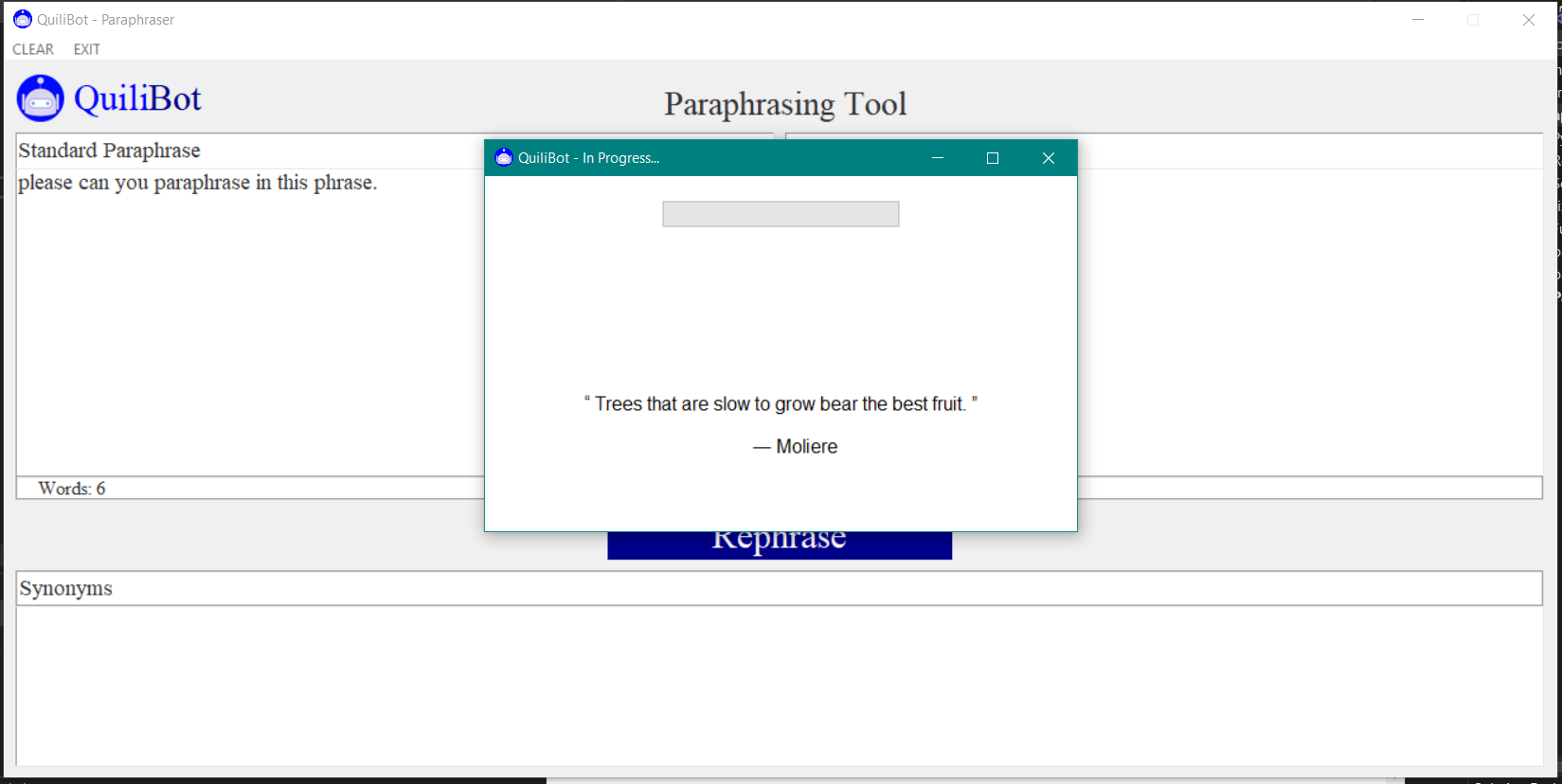
**Chapter 4 – Flow of Program**

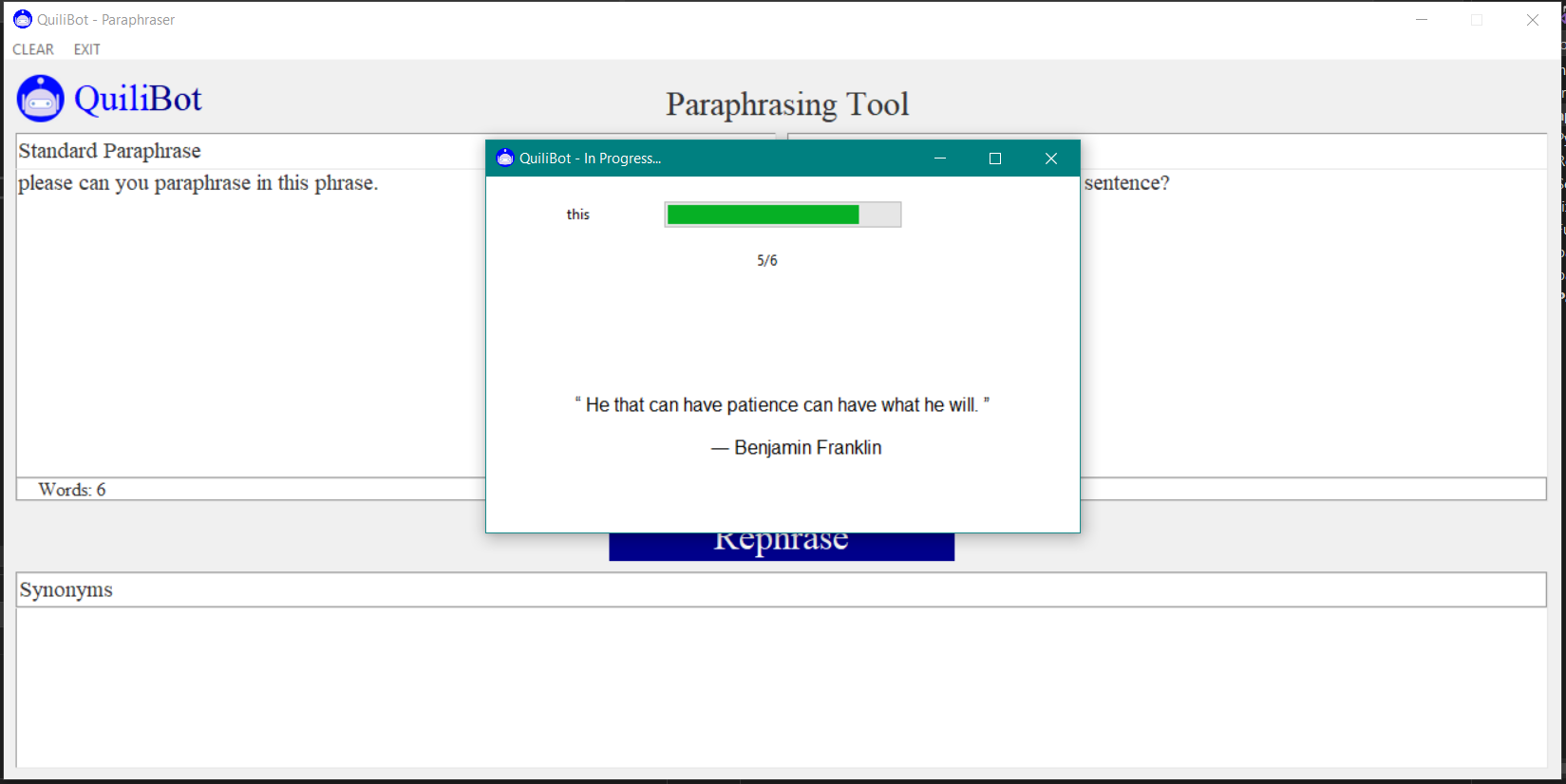
**Screen Layout**

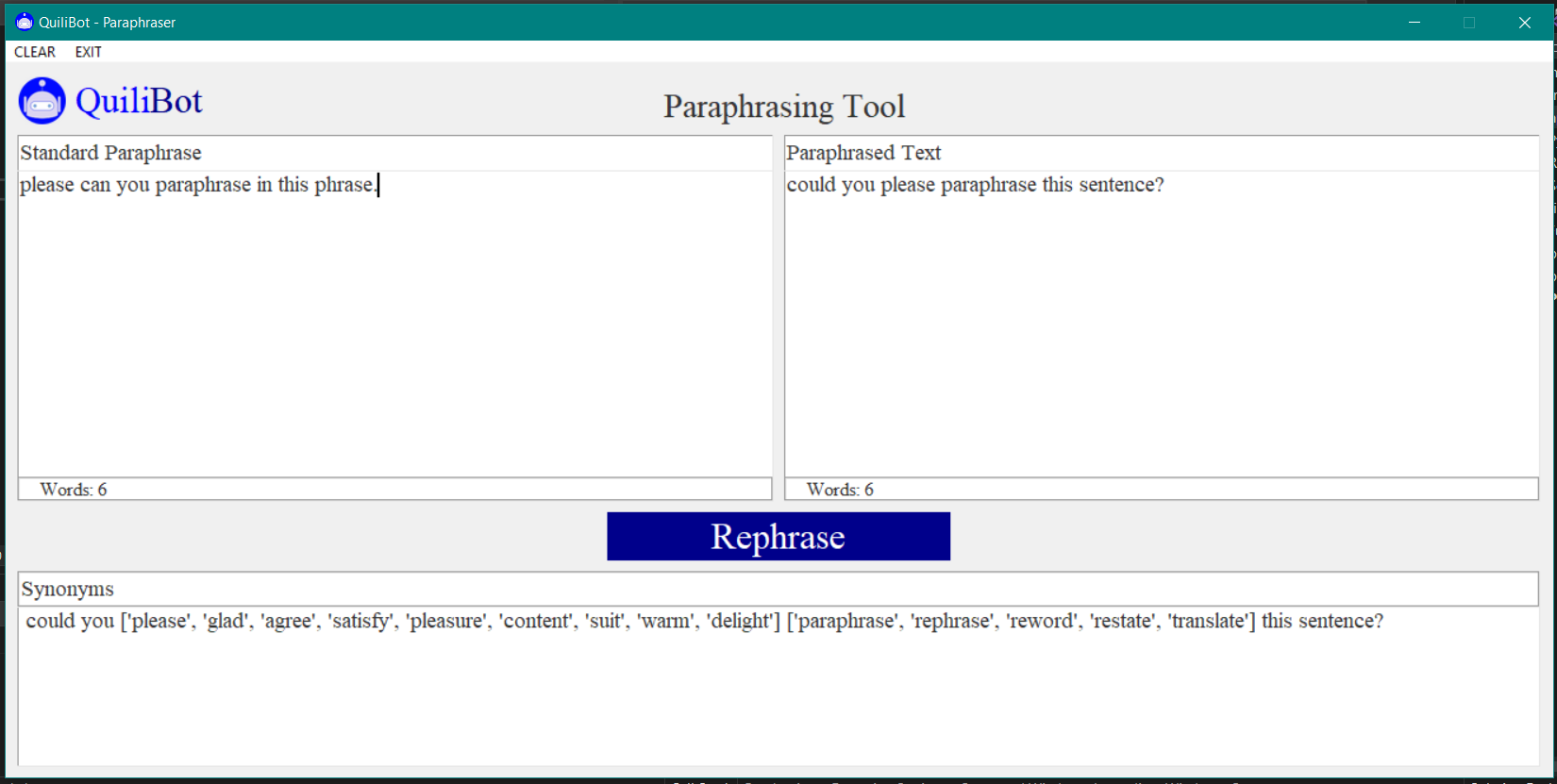


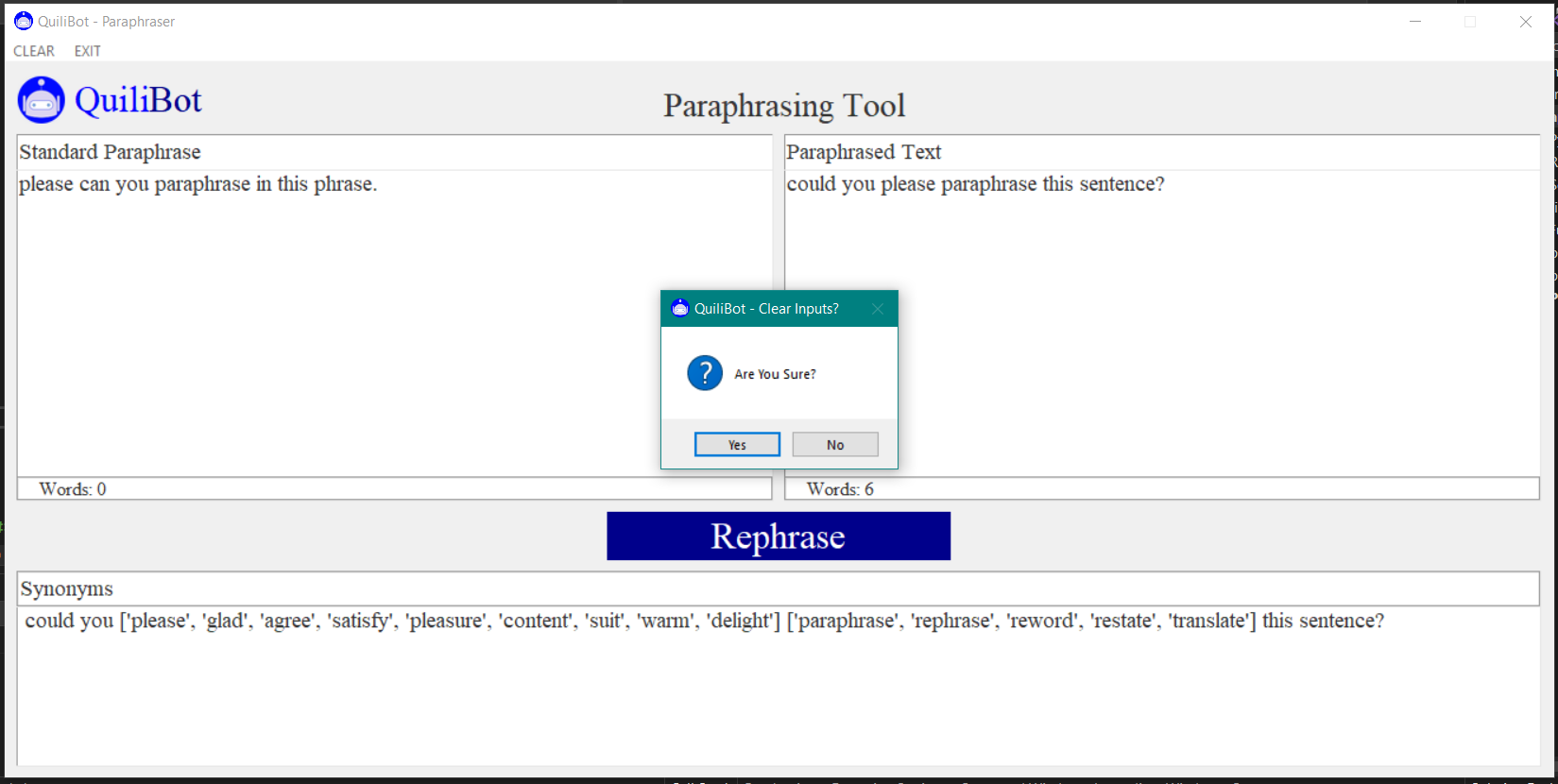


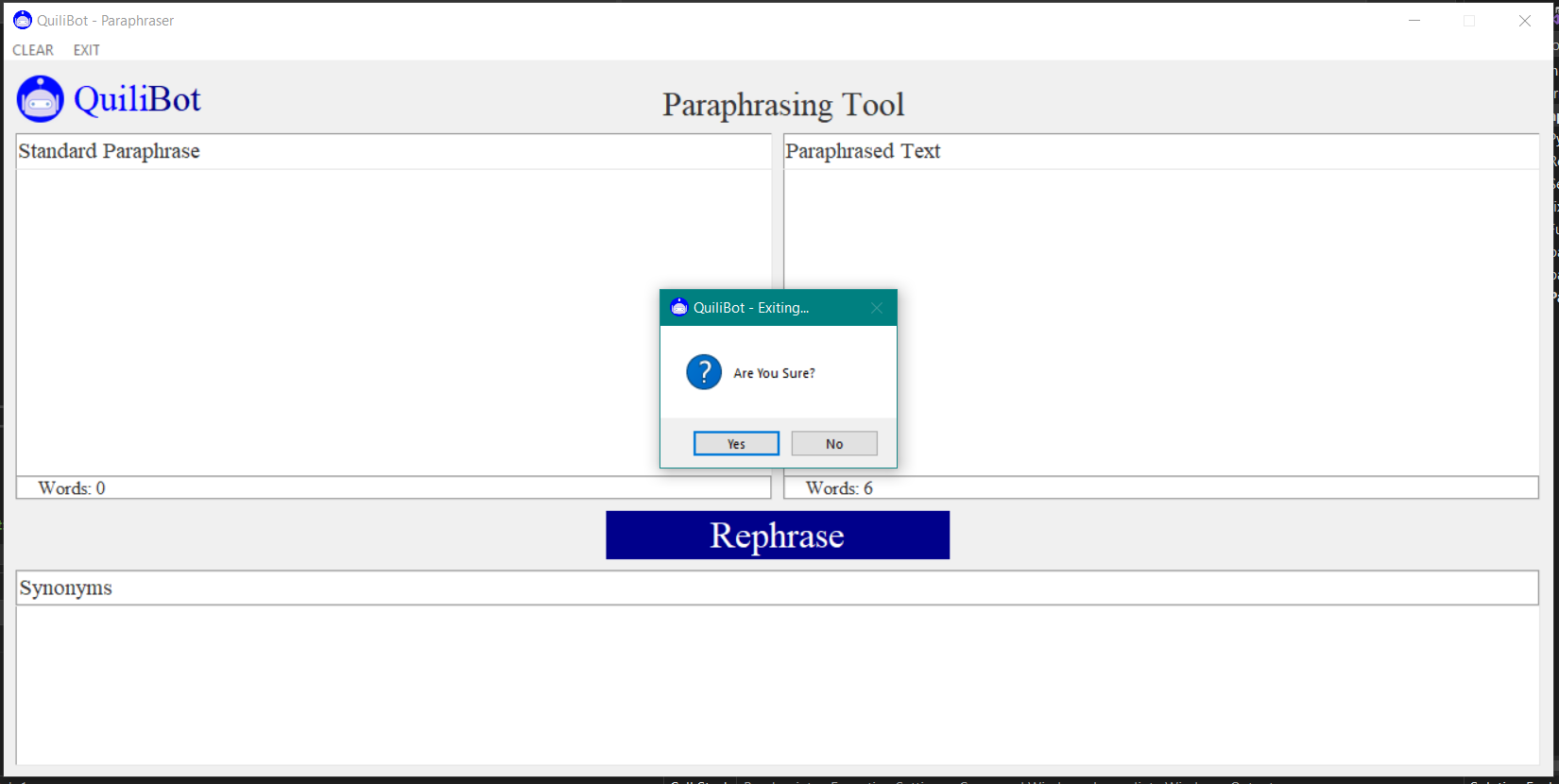






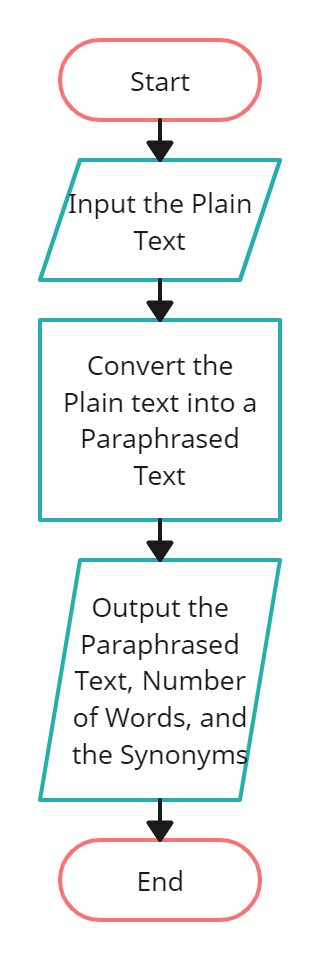








**Flow Chart**



**Chapter 5 – Results**

**Conclusions**

This QuiliBot program was developed with the intention of assisting individuals, particularly students, in changing the format of an essay or other piece of writing they have generated in order to avoid plagiarism and create a fresh, new version of the content. In order to assist users in meeting any word requirements, this program also shows the overall number of words in the text. In order for the user to decide if they wish to use the synonym of the altered word, the application also displays the synonyms of the changed terms. The QuiliBot application is beneficial and effective to use and implement in their job, based on the testing that is carried out by the program's testers. The program was successfully developed utilizing the Agile Methodology, and it generates a valuable and effective application that can be utilized by users who are employed by businesses that consistently produce documents in addition to students. The testing findings were successful because they met the application's aims and objectives, which included changing the text's form by paraphrasing it, counting the total number of words, and displaying the modified words' synonyms.

**Recommendations**

The testers of the system have suggested that the application's GUI be improved in order to make it more user-friendly and aesthetically acceptable to the application's users. This can result in an application that is both practical and user-friendly. The functionality of the program may be enhanced by including more paraphrasing choices, such as other ways to paraphrase the text. As a result, if the user is paraphrasing the text, they have more possibilities to pick from. Another feature that may be added to the program is the ability to verify the definition of the text when the cursor is over a certain text. Users may be better able to grasp the modified content as a result. Last but not least, another feature that is advised is changing the color of the modified text so that it is clear which portion of the text has been altered.

**References**

<https://quillbot.com/>

<https://paraphrasing-tool.com/>

<https://spinbot.com/>

<https://github.com/PrithivirajDamodaran/Gramformer>

<https://github.com/PrithivirajDamodaran/Parrot_Paraphraser/tree/main/parrot>

<https://visualtk.com>

<https://stackoverflow.com>

<https://www.tutorialspoint.com/index.htm>

**Appendices**

**User Manual**

When the user starts the application, the screen displays the main interface of the program and all of the functionalities that the application has.

- The first step in using the application is to make sure that you have copied the text that you want to paraphrase using the application.

- The next step is to paste your copied text in the input box given in the application.

- After pasting the text, the next step is to click the "Rephrase" button in order to rephrase the text and change it into a new one.

- The application then displays the rephrased text in the output box of the application. It also displays the total number of words in the input and output boxes, respectively. The program also displays the synonyms of the changed words so that the user can choose which other word to use from the synonyms.

- The user can clear the text in all of the text boxes by clicking the "Clear" button in the upper left corner of the application. This function clears and deletes all of the text that was used previously in order to paste the newly rephrased text.

**Source Code**

Classes and Modules

paraphrase\_filters.py

import torch

class Adequacy():

def \_\_init\_\_(self, model\_tag='prithivida/parrot\_adequacy\_model'):

from transformers import AutoModelForSequenceClassification, AutoTokenizer

self.adequacy\_model = AutoModelForSequenceClassification.from\_pretrained(model\_tag)

self.tokenizer = AutoTokenizer.from\_pretrained(model\_tag)

def filter(self, input\_phrase, para\_phrases, adequacy\_threshold, device="cpu"):

top\_adequacy\_phrases = []

for para\_phrase in para\_phrases:

x = self.tokenizer(input\_phrase, para\_phrase, return\_tensors='pt', max\_length=128, truncation=True)

x = x.to(device)

self.adequacy\_model = self.adequacy\_model.to(device)

logits = self.adequacy\_model(\*\*x).logits

probs = logits.softmax(dim=1)

prob\_label\_is\_true = probs[:,1]

adequacy\_score = prob\_label\_is\_true.item()

if adequacy\_score >= adequacy\_threshold:

top\_adequacy\_phrases.append(para\_phrase)

return top\_adequacy\_phrases

def score(self, input\_phrase, para\_phrases, adequacy\_threshold, device="cpu"):

adequacy\_scores = {}

for para\_phrase in para\_phrases:

x = self.tokenizer(input\_phrase, para\_phrase, return\_tensors='pt', max\_length=128, truncation=True)

x = x.to(device)

self.adequacy\_model = self.adequacy\_model.to(device)

logits = self.adequacy\_model(\*\*x).logits

probs = logits.softmax(dim=1)

prob\_label\_is\_true = probs[:,1]

adequacy\_score = prob\_label\_is\_true.item()

if adequacy\_score >= adequacy\_threshold:

adequacy\_scores[para\_phrase] = adequacy\_score

return adequacy\_scores

class Fluency():

def \_\_init\_\_(self, model\_tag='prithivida/parrot\_fluency\_model'):

from transformers import AutoModelForSequenceClassification, AutoTokenizer

self.fluency\_model = AutoModelForSequenceClassification.from\_pretrained(model\_tag, num\_labels=2)

self.fluency\_tokenizer = AutoTokenizer.from\_pretrained(model\_tag)

def filter(self, para\_phrases, fluency\_threshold, device="cpu"):

import numpy as np

from scipy.special import softmax

self.fluency\_model = self.fluency\_model.to(device)

top\_fluent\_phrases = []

for para\_phrase in para\_phrases:

input\_ids = self.fluency\_tokenizer("Sentence: " + para\_phrase, return\_tensors='pt', truncation=True)

input\_ids = input\_ids.to(device)

prediction = self.fluency\_model(\*\*input\_ids)

scores = prediction[0][0].detach().cpu().numpy()

scores = softmax(scores)

fluency\_score = scores[1] # LABEL\_0 = Bad Fluency, LABEL\_1 = Good Fluency

if fluency\_score >= fluency\_threshold:

top\_fluent\_phrases.append(para\_phrase)

return top\_fluent\_phrases

def score(self, para\_phrases, fluency\_threshold, device="cpu"):

import numpy as np

from scipy.special import softmax

self.fluency\_model = self.fluency\_model.to(device)

fluency\_scores = {}

for para\_phrase in para\_phrases:

input\_ids = self.fluency\_tokenizer("Sentence: " + para\_phrase, return\_tensors='pt', truncation=True)

input\_ids = input\_ids.to(device)

prediction = self.fluency\_model(\*\*input\_ids)

scores = prediction[0][0].detach().cpu().numpy()

scores = softmax(scores)

fluency\_score = scores[1] # LABEL\_0 = Bad Fluency, LABEL\_1 = Good Fluency

if fluency\_score >= fluency\_threshold:

fluency\_scores[para\_phrase] = fluency\_score

return fluency\_scores

class Diversity():

def \_\_init\_\_(self, model\_tag='paraphrase-distilroberta-base-v2'):

from sentence\_transformers import SentenceTransformer

self.diversity\_model = SentenceTransformer(model\_tag)

def rank(self, input\_phrase, para\_phrases, diversity\_ranker='levenshtein'):

if diversity\_ranker == "levenshtein":

return self.levenshtein\_ranker(input\_phrase, para\_phrases)

elif diversity\_ranker == "euclidean":

return self.euclidean\_ranker(input\_phrase, para\_phrases)

elif diversity\_ranker == "diff":

return self.diff\_ranker(input\_phrase, para\_phrases)

def euclidean\_ranker(self, input\_phrase, para\_phrases):

import pandas as pd

from sklearn\_pandas import DataFrameMapper

from sklearn.preprocessing import MinMaxScaler

from scipy import spatial

diversity\_scores = {}

outputs = []

input\_enc = self.diversity\_model.encode(input\_phrase.lower())

for para\_phrase in para\_phrases:

paraphrase\_enc = self.diversity\_model.encode(para\_phrase.lower())

euclidean\_distance = (spatial.distance.euclidean(input\_enc, paraphrase\_enc))

outputs.append((para\_phrase, euclidean\_distance))

df = pd.DataFrame(outputs, columns=['paraphrase', 'scores'])

fields = []

for col in df.columns:

if col == "scores":

tup = ([col], MinMaxScaler())

else:

tup = ([col], None)

fields.append(tup)

mapper = DataFrameMapper(fields, df\_out=True)

for index, row in mapper.fit\_transform(df.copy()).iterrows():

diversity\_scores[row['paraphrase']] = row['scores']

return diversity\_scores

def levenshtein\_ranker(self, input\_phrase, para\_phrases):

import Levenshtein

diversity\_scores = {}

for para\_phrase in para\_phrases:

distance = Levenshtein.distance(input\_phrase.lower(), para\_phrase)

diversity\_scores[para\_phrase] = distance

return diversity\_scores

def diff\_ranker(self, input\_phrase, para\_phrases):

import difflib

differ = difflib.Differ()

diversity\_scores ={}

for para\_phrase in para\_phrases:

diff = differ.compare(input\_phrase.split(), para\_phrase.split())

count = 0

for d in diff:

if "+" in d or "-" in d:

count += 1

diversity\_scores[para\_phrase] = count

return

paraphrase.py

class Paraphraser():

def \_\_init\_\_(self, model\_tag="prithivida/parrot\_paraphraser\_on\_T5", use\_gpu=False):

from transformers import AutoTokenizer

from transformers import AutoModelForSeq2SeqLM

import pandas as pd

from paraphrase\_filters import Adequacy

from paraphrase\_filters import Fluency

from paraphrase\_filters import Diversity

self.tokenizer = AutoTokenizer.from\_pretrained(model\_tag, use\_auth\_token=False)

self.model = AutoModelForSeq2SeqLM.from\_pretrained(model\_tag, use\_auth\_token=False)

self.adequacy\_score = Adequacy()

self.fluency\_score = Fluency()

self.diversity\_score= Diversity()

def rephrase(self, input\_phrase, use\_gpu=False, diversity\_ranker="levenshtein", do\_diverse=False, style=1, max\_length=32, adequacy\_threshold = 0.90, fluency\_threshold = 0.90):

if use\_gpu:

device= "cuda:0"

else:

device = "cpu"

self.model = self.model.to(device)

import re

save\_phrase = input\_phrase

if len(input\_phrase) >= max\_length:

max\_length += 32

input\_phrase = re.sub('[^a-zA-Z0-9 \?\'\-\/\:\.]', '', input\_phrase)

input\_phrase = "paraphrase: " + input\_phrase

input\_ids = self.tokenizer.encode(input\_phrase, return\_tensors='pt')

input\_ids = input\_ids.to(device)

max\_return\_phrases = 10

if do\_diverse:

for n in range(2, 9):

if max\_return\_phrases % n == 0:

break

#print("max\_return\_phrases - ", max\_return\_phrases , " and beam groups -", n)

preds = self.model.generate(

input\_ids,

do\_sample=False,

max\_length=max\_length,

num\_beams = max\_return\_phrases,

num\_beam\_groups = n,

diversity\_penalty = 2.0,

early\_stopping=True,

num\_return\_sequences=max\_return\_phrases)

else:

preds = self.model.generate(

input\_ids,

do\_sample=True,

max\_length=max\_length,

top\_k=50,

top\_p=0.95,

early\_stopping=True,

num\_return\_sequences=max\_return\_phrases)

paraphrases= set()

for pred in preds:

gen\_pp = self.tokenizer.decode(pred, skip\_special\_tokens=True).lower()

gen\_pp = re.sub('[^a-zA-Z0-9 \?\'\-]', '', gen\_pp)

paraphrases.add(gen\_pp)

adequacy\_filtered\_phrases = self.adequacy\_score.filter(input\_phrase, paraphrases, adequacy\_threshold, device )

if len(adequacy\_filtered\_phrases) > 0 :

fluency\_filtered\_phrases = self.fluency\_score.filter(adequacy\_filtered\_phrases, fluency\_threshold, device )

if len(fluency\_filtered\_phrases) > 0 :

diversity\_scored\_phrases = self.diversity\_score.rank(input\_phrase, fluency\_filtered\_phrases, diversity\_ranker)

para\_phrases = []

for para\_phrase, diversity\_score in diversity\_scored\_phrases.items():

para\_phrases.append(para\_phrase)

para\_phrases.sort(key=lambda x:x[1], reverse=True)

return para\_phrases[0]

else:

return [(save\_phrase,0)]

def augment(self, input\_phrase, use\_gpu=False, diversity\_ranker="levenshtein", do\_diverse=False, max\_return\_phrases = 32, max\_length=32, adequacy\_threshold = 0.90, fluency\_threshold = 0.90):

if use\_gpu:

device= "cuda:0"

else:

device = "cpu"

self.model = self.model.to(device)

import re

save\_phrase = input\_phrase

if len(input\_phrase) >= max\_length:

max\_length += 32

input\_phrase = re.sub('[^a-zA-Z0-9 \?\'\-\/\:\.]', '', input\_phrase)

input\_phrase = "paraphrase: " + input\_phrase

input\_ids = self.tokenizer.encode(input\_phrase, return\_tensors='pt')

input\_ids = input\_ids.to(device)

if do\_diverse:

for n in range(2, 9):

if max\_return\_phrases % n == 0:

break

#print("max\_return\_phrases - ", max\_return\_phrases , " and beam groups -", n)

preds = self.model.generate(

input\_ids,

do\_sample=False,

max\_length=max\_length,

num\_beams = max\_return\_phrases,

num\_beam\_groups = n,

diversity\_penalty = 2.0,

early\_stopping=True,

num\_return\_sequences=max\_return\_phrases)

else:

preds = self.model.generate(

input\_ids,

do\_sample=True,

max\_length=max\_length,

top\_k=50,

top\_p=0.95,

early\_stopping=True,

num\_return\_sequences=max\_return\_phrases)

paraphrases= set()

for pred in preds:

gen\_pp = self.tokenizer.decode(pred, skip\_special\_tokens=True).lower()

gen\_pp = re.sub('[^a-zA-Z0-9 \?\'\-]', '', gen\_pp)

paraphrases.add(gen\_pp)

adequacy\_filtered\_phrases = self.adequacy\_score.filter(input\_phrase, paraphrases, adequacy\_threshold, device )

if len(adequacy\_filtered\_phrases) > 0 :

fluency\_filtered\_phrases = self.fluency\_score.filter(adequacy\_filtered\_phrases, fluency\_threshold, device )

if len(fluency\_filtered\_phrases) > 0 :

diversity\_scored\_phrases = self.diversity\_score.rank(input\_phrase, fluency\_filtered\_phrases, diversity\_ranker)

para\_phrases = []

for para\_phrase, diversity\_score in diversity\_scored\_phrases.items():

para\_phrases.append(para\_phrase)

para\_phrases.sort(key=lambda x:x[1], reverse=True)

return para\_phrases

else:

return [(save\_phrase)]

fix\_grammar.py

class Fix\_Grammar:

def \_\_init\_\_(self, models=1, use\_gpu=False):

from transformers import AutoTokenizer

from transformers import AutoModelForSeq2SeqLM

#from lm\_scorer.models.auto import AutoLMScorer as LMScorer

import errant

import spacy

nlp = spacy.load('en\_core\_web\_sm')

self.annotator = errant.load('en', nlp)

if use\_gpu:

device= "cuda:0"

else:

device = "cpu"

batch\_size = 1

#self.scorer = LMScorer.from\_pretrained("gpt2", device=device, batch\_size=batch\_size)

self.device = device

correction\_model\_tag = "prithivida/grammar\_error\_correcter\_v1"

self.model\_loaded = False

if models == 1:

self.correction\_tokenizer = AutoTokenizer.from\_pretrained(correction\_model\_tag, use\_auth\_token=False)

self.correction\_model = AutoModelForSeq2SeqLM.from\_pretrained(correction\_model\_tag, use\_auth\_token=False)

self.correction\_model = self.correction\_model.to(device)

self.model\_loaded = True

elif models == 2:

# TODO

print("TO BE IMPLEMENTED!!!")

def correct(self, input\_sentence, max\_candidates=1):

if self.model\_loaded:

correction\_prefix = "gec: "

input\_sentence = correction\_prefix + input\_sentence

input\_ids = self.correction\_tokenizer.encode(input\_sentence, return\_tensors='pt')

input\_ids = input\_ids.to(self.device)

preds = self.correction\_model.generate(

input\_ids,

do\_sample=True,

max\_length=128,

# top\_k=50,

# top\_p=0.95,

num\_beams=7,

early\_stopping=True,

num\_return\_sequences=max\_candidates)

corrected = set()

for pred in preds:

corrected.add(self.correction\_tokenizer.decode(pred, skip\_special\_tokens=True).strip())

for c in corrected:

corrected\_output = c

#corrected = list(corrected)

#scores = self.scorer.sentence\_score(corrected, log=True)

#ranked\_corrected = [(c,s) for c, s in zip(corrected, scores)]

#ranked\_corrected.sort(key = lambda x:x[1], reverse=True)

return corrected\_output

else:

print("Model is not loaded")

return None

def highlight(self, orig, cor):

edits = self.\_get\_edits(orig, cor)

orig\_tokens = orig.split()

ignore\_indexes = []

for edit in edits:

edit\_type = edit[0]

edit\_str\_start = edit[1]

edit\_spos = edit[2]

edit\_epos = edit[3]

edit\_str\_end = edit[4]

# if no\_of\_tokens(edit\_str\_start) > 1 ==> excluding the first token, mark all other tokens for deletion

for i in range(edit\_spos+1, edit\_epos):

ignore\_indexes.append(i)

if edit\_str\_start == "":

if edit\_spos - 1 >= 0:

new\_edit\_str = orig\_tokens[edit\_spos - 1]

edit\_spos -= 1

else:

new\_edit\_str = orig\_tokens[edit\_spos + 1]

edit\_spos += 1

if edit\_type == "PUNCT":

st = "<a type='" + edit\_type + "' edit='" + \

edit\_str\_end + "'>" + new\_edit\_str + "</a>"

else:

st = "<a type='" + edit\_type + "' edit='" + new\_edit\_str + \

" " + edit\_str\_end + "'>" + new\_edit\_str + "</a>"

orig\_tokens[edit\_spos] = st

elif edit\_str\_end == "":

st = "<d type='" + edit\_type + "' edit=''>" + edit\_str\_start + "</d>"

orig\_tokens[edit\_spos] = st

else:

st = "<c type='" + edit\_type + "' edit='" + \

edit\_str\_end + "'>" + edit\_str\_start + "</c>"

orig\_tokens[edit\_spos] = st

for i in sorted(ignore\_indexes, reverse=True):

del(orig\_tokens[i])

return(" ".join(orig\_tokens))

def detect(self, input\_sentence):

# TO BE IMPLEMENTED

pass

def \_get\_edits(self, orig, cor):

orig = self.annotator.parse(orig)

cor = self.annotator.parse(cor)

alignment = self.annotator.align(orig, cor)

edits = self.annotator.merge(alignment)

if len(edits) == 0:

return []

edit\_annotations = []

for e in edits:

e = self.annotator.classify(e)

edit\_annotations.append((e.type[2:], e.o\_str, e.o\_start, e.o\_end, e.c\_str, e.c\_start, e.c\_end))

if len(edit\_annotations) > 0:

return edit\_annotations

else:

return []

def get\_edits(self, orig, cor):

return self.\_get\_edits(orig, cor)

Functions.py

from textblob import TextBlob

def fix\_spelling(text):

output = str(TextBlob(text).correct())

return output

Main Code

Paraphrase\_Generator.py

from cgitb import enable

from nltk.tokenize import word\_tokenize, sent\_tokenize, RegexpTokenizer

from nltk import pos\_tag

from nltk.corpus import wordnet

from bs4 import BeautifulSoup as soup

from urllib.request import urlopen as uReq

from urllib.error import HTTPError

from random\_word import RandomWords

import nltk

from threading import Thread

from pattern.en import superlative,pluralize,comparative,superlative

import spacy

from tkinter import filedialog

import csv

import ast

import requests

import os

import random

from pyinflect import \*

from tkinter import \*

from tkinter import messagebox

from tkinter.ttk import Progressbar

import tkinter.font as tkFont

import tkinter as tk

import tkinter.ttk as ttk

import random

import Functions as fn

from fix\_grammar import Fix\_Grammar

from paraphrase import Paraphraser

import torch

import warnings

import re

quotes\_list=['“ Trees that are slow to grow bear the best fruit. ”\n\n ― Moliere','“ He that can have patience can have what he will. ”\n \n ― Benjamin Franklin','“ Patience is a conquering virtue. ” \n\n ― Geoffrey Chaucer','“ Patience is bitter, but its fruit is sweet. ”\n\n ― Aristotle','“ The strongest of all warriors are these two — Time and Patience. ” \n\n ― Leo Tolstoy, War and Peace','“ Rivers know this: there is no hurry. We shall get there some day. ”\n\n ― A.A. Milne, Winnie-the-Pooh']

#root=Tk() #main GUI window

##model which contains word vectors (used in program to compare similarities)

#root.wm\_title('Paraphrase Generator')

#spacy.cli.download("en\_core\_web\_lg")

nlp = spacy.load('en\_core\_web\_lg')

#file containing input

class App:

def \_\_init\_\_(self, root):

#setting title

root.title("QuiliBot - Paraphraser")

#icon

root.iconbitmap('.img\\icons8-bot-80.ico')

#setting window size

width=1312

height=625

screenwidth = root.winfo\_screenwidth()

screenheight = root.winfo\_screenheight()

alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)

root.geometry(alignstr)

root.resizable(width=False, height=False)

def set\_seed(seed):

torch.manual\_seed(seed)

if torch.cuda.is\_available():

torch.cuda.manual\_seed\_all(seed)

def progressbar(i):

label['text']=str(i+1)+'/'+str(total\_length)

bar['value']=(100/total\_length)\*(i+1)

status.config(text=words[i])

quote['text']=quotes\_list[random.randint(0,len(quotes\_list)-1)]

if i+1==total\_length:

top.destroy()

#1st function to find similarities

def synonyms2(term):

synonyms\_ = []

for syn in wordnet.synsets(term):

for l in syn.lemmas():

synonyms\_.append(l.name())

return synonyms\_

def synonyms(string):

try:

if string!=u'"' and string!=u'\"' and string!=u'\'' and string!=u"'":

stripped\_string = string.strip()

fixed\_string = stripped\_string.replace(" ", "\_")

# Set the url using the amended string

my\_url = f'https://thesaurus.plus/thesaurus/{fixed\_string}'

# Open and read the HTMLz

uClient = uReq(my\_url)

page\_html = uClient.read()

uClient.close()

# Parse the html into text

page\_soup = soup(page\_html, "html.parser")

word\_boxes = page\_soup.find("ul", {"class": "list paper"})

results = word\_boxes.find\_all("div", "list\_item")

sim\_words\_list=[]

# Iterate over results and print

for result in results:

sim\_words\_list.append(result.text)

# Remove whitespace before and after word and use underscore between words

except Exception as e:

sim\_words\_list=[]

if "\_" in fixed\_string:

print(e)

else:

print(e)

print('yeahh')

return sim\_words\_list

def syn(word):

urls=["https://dictionaryapi.com/api/v3/references/thesaurus/json/{}?key=a24dfaa9-ffe0-4a9a-8906-c8ff3e8dd406".format(word),"https://dictionaryapi.com/api/v3/references/thesaurus/json/{}?key=92432adb-24d2-46a1-a979-f1b701155bf2".format(word)]

return\_list=[]

try:

url=urls[random.randint(0,1)]

r = requests.get(url)

if r.json()!=[] or r.json()!='':

return\_list=r.json()[0]['meta']['syns'][0]

else:

raise Exception('error')

print('1')

except:

try:

url=urls[random.randint(0,1)]

r = requests.get(url)

if r.json()!=[] or r.json()!='':

return\_list=r.json()[0]['meta']['syns'][0]

else:

return\_list=r.json()[0]['meta']['syns'][0]

raise Exception('error')

print('2')

except:

try:

url = "https://dictionaryapi.com/api/v3/references/ithesaurus/json/{}?key=61d0a7b2-125f-4b16-b0d8-1f3593627bf9".format(word)

r = requests.get(url)

if r.json()!=[] or r.json()!='':

return\_list=r.json()[0]['meta']['syns'][0]

else:

return\_list=r.json()[0]['meta']['syns'][0]

raise Exception('error')

print('3')

except:

try:

url = "https://dictionaryapi.com/api/v3/references/ithesaurus/json/{}?key=ea94eb40-67b0-4a3d-ba53-5a3da9079906".format(word)

r = requests.get(url)

if r.json()!=[] or r.json()!='':

return\_list=r.json()[0]['meta']['syns'][0]

print('4')

except:

if word not in ['.',',','(' ,')','',' (','( ',' )',') ',' .','. ','!','doesn','t','don','\'','i','l',' t','t ','\'t', "'",'wasn','didn','couldn','wouldn','weren',';',':','|','I','l','L','s','ain','shouldn','&','?',"\'","'",'"','\"']:

if word!='\"' and word!='"' and word!='\'' and word!="'":

return\_list=[]

else:

return\_list=[]

if return\_list!=[]:

with open('.img\\words.csv','a',newline='') as csvfile:

wr=csv.writer(csvfile)

global d

d={'id':word,'value':return\_list}

for i in range(int(len(d)/2)):

k=tuple(d.values())

w=k[0]

v=k[1]

wr.writerow([w,v])

return return\_list

def fun(word):

csv\_\_file= open('.img\\words.csv','r')

reader=csv.reader(csv\_\_file)

k=True

if reader!=[]:

for row in reader:

if row!=[]:

if row[0].lower()==word.lower():

k=False

return (ast.literal\_eval(row[1]))

else:

k=True

if k==True:

return (syn(word))

def start(position=None,\*args):

global output,output2,total\_length,words

gf = Fix\_Grammar(models = 1, use\_gpu=False) # 1=corrector, 2=detector

fixed\_spell = fn.fix\_spelling(input\_box.get('1.0','end-1c')) #fixes spelling

set\_seed(1212) #fixes grammar

fixed\_grammar = gf.correct(fixed\_spell, max\_candidates=1)

warnings.filterwarnings("ignore")

set\_seed(1234)

paraphrase = Paraphraser(model\_tag="prithivida/parrot\_paraphraser\_on\_T5")

sentence\_tokenize = sent\_tokenize(fixed\_grammar)

for sentence in sentence\_tokenize:

para\_phrases = paraphrase.rephrase(input\_phrase=sentence, use\_gpu=False)

for para\_phrase in para\_phrases:

result\_box1.insert(END,para\_phrase)

tokenizer=RegexpTokenizer(r'\s+', gaps=True)

words=tokenizer.tokenize(result\_box1.get('1.0','end-1c'))

total\_length=len(words)

taggs=pos\_tag(words)

output=''

output2=''

real\_words=[]

for i in range(len(words)):

a=taggs[i][1]

print('word: ',words[i] ,'tag: ',a)

r=[]

# not to fin synonym for the name of a person,a decorator etc.

if ( taggs[i][1]!='DP' and taggs[i][1]!='CD' and taggs[i][1]!='TO' and taggs[i][1]!='PRP$' and taggs[i][1]!='IN' and taggs[i][1]!='PRP' and taggs[i][1]!='DT' and taggs[i][1]!='WRB' and taggs[i][1]!='WR') and( words[i] not in ['.',',','(' ,')','',' (','( ',' )',') ',' .','. ','!','doesn','t','don','\'','i','l',' t','t ','\'t', "'",'wasn','didn','couldn','wouldn','weren','I','L','1','|',';',':','s',' s','ain','ll','-','\_\_']) and words[i].lower() not in ['time','second','seconds','month','months','year','years','minute','minutes','indian','countries','let']:

if words[i]!='.' and words[i]!=',' and words[i]!="'" and words[i]!="\"" and words[i]!="\"" and words[i]!='"' and words[i]!=' "' and words[i]!='" ' and words[i]!='?':

r=fun(words[i])

progressbar(i)

if r!=[]:

print('list of words: ',r)

real\_words=[]

# to make the similar words more similar by changing their tense etc.

for j in r:

tag=pos\_tag([j])[0][1]

if tag==a or a=='JJ':

# if already similar

real\_words.append(j)

elif tag!=a: #check part-of speech tags and change accordingly

if a=='NNPS':

token=nlp(j)

w=tokens[0].\_.inflect('NNPS', form\_num=0)

if w!=None and pos\_tag([w])==a:

real\_words.append(tokens[0].\_.inflect('NNPS',inflect\_oov=True, form\_num=0))

elif a=='NNS':

token=nlp(j)

w=token[0].\_.inflect('NNS',inflect\_oov=True, form\_num=0)

if w!=None and pos\_tag([w])==a:

real\_words.append(token[0].\_.inflect('NNS',inflect\_oov=True,form\_num=0))

elif a=='NNP':

token=nlp(j)

w=token[0].\_.inflect('NNP',inflect\_oov=True, form\_num=0)

if w!=None and pos\_tag([w])==a:

real\_words.append(token[0].\_.inflect('NNP',inflect\_oov=True,from\_num=0))

elif a=='NN':

token=nlp(j)

w=token[0].\_.inflect('NN',form\_num=0)

real\_words.append(w)

elif a=='RB':

token=nlp(j)

w=token[0].\_.inflect('RB',inflect\_oov=True,form\_num=0)

real\_words.append(token[0].\_.inflect("RB",inflect\_oov=True,form\_num=0))

elif a=='RBR':

token=nlp(j)

w=token[0].\_.inflect('RBR', inflect\_oov=True,form\_num=0)

if w!=None and pos\_tag([w])==a:

real\_words.append(token[0].\_.inflect('RBR',inflect\_oov=True,form\_num=0))

elif a=='RBS':

token=nlp(j)

w=token[0].\_.inflect('RBS',inflect\_oov=True, form\_num=0)

if w!=None and pos\_tag([w])==a:

real\_words.append(token[0].\_.inflect('RBS',inflect\_oov=True,form\_num=0))

elif a=='VB' :

tokens = nlp(j)

w=tokens[0].\_.inflect('VB',inflect\_oov=True, form\_num=0)

if w!=None :

real\_words.append(tokens[0].\_.inflect('VB',inflect\_oov=True, form\_num=0))

elif a=='VBD' :

tokens = nlp(j)

w=tokens[0].\_.inflect('VBD', form\_num=1)

if w!=None :

real\_words.append(tokens[0].\_.inflect('VBD', inflect\_oov=True,form\_num=0))

elif a=='VBG':

tokens=nlp(j)

w=tokens[0].\_.inflect('VBG',inflect\_oov=True, form\_num=0)

if w!=None and pos\_tag([w])==a:

real\_words.append(tokens[0].\_.inflect('VBG',inflect\_oov=True, form\_num=0))

elif a=='VBN ':

tokens=nlp(j)

w=tokens[0].\_.inflect('VBN',inflect\_oov=True, form\_num=0)

if w!=None and pos\_tag([w])==a:

real\_words.append(tokens[0].\_.inflect('VBN', inflect\_oov=True,form\_num=0))

elif a=='VBP':

tokens=nlp(j)

w=tokens[0].\_.inflect('VBP',inflect\_oov=True, form\_num=0)

if w!=None and pos\_tag([w])==a:

real\_words.append(tokens[0].\_.inflect('VBP',inflect\_oov=True, form\_num=0))

elif a=='VBZ':

tokens=nlp(j)

w=tokens[0].\_.inflect('VBZ',inflect\_oov=True, form\_num=0)

if w!=None and pos\_tag([w])==a:

real\_words.append(tokens[0].\_.inflect('VBZ',inflect\_oov=True, form\_num=0))

elif a=='JJR' :

real\_words.append(comparative(j))

elif a=='JJS' :

real\_words.append(superlative(j))

print('real words: ',real\_words)

if real\_words==[] or r==[]: #if no similar word is found

output=output+' '+words[i]

output2=output2+' '+words[i]

else :

output\_words=[]

max\_sim=[]

token1=nlp(words[i])

for h in real\_words:

if h!=None and h!='' :

token2=nlp(h)

f1=h.replace(' ','')

f=f1.replace('\_',' ')

sim=token1.similarity(token2)

if h not in output\_words and (f not in output\_words ) and words[i].lower()!=h.lower() and words[i].lower()!=f.lower() : #adding appropriate word

output\_words.append(f)

max\_sim.append(sim)

else:

sim=0

final\_listwords=[]

for jj in max\_sim:

final\_listwords.append(output\_words[max\_sim.index(max(max\_sim))])

max\_sim[max\_sim.index(max(max\_sim))]=-1

if output\_words==[]:

output=output+' '+words[i]

output2=output2+' '+words[i]

elif position==None or type(position)!=int:

print('final\_listwords: ',final\_listwords)

if len(output\_words)>3:

output=output+' '+final\_listwords[0]

else:

output=output+' '+final\_listwords[random.randint(0,len(final\_listwords)-1)]

elif position!=None and (type(position)==int or type(position)==str):

if type(position)==str: #choose a specific position of word from list of similar words

if len(output\_words)>int(position) :

output=output+' '+final\_listwords[int(position)]

elif type(position)==int:

if len(output\_words)>(position):

output=output+' '+final\_listwords[position]

else:

output=output+' '+final\_listwords[len(output\_words)-1]

if len(output\_words)>7:

output2=output2+' '+str([words[i]]+final\_listwords[0:8])

# if number of similar words is less than 4 than choose randomly

elif len(output\_words)<=7:

output2=output2+' '+str([words[i]]+final\_listwords[0:])

word\_count2.config(text=f" Words: {total\_length}")

result\_box2.insert(END,output2)

def go\_():

if input\_box.get('1.0','end-1c')=='' or input\_box.get('1.0','end-1c')==' ':

messagebox.showinfo('QuiliBot - Invalid!','Insert Text First!')

result\_box1.delete('1.0',END)

result\_box2.delete('1.0',END)

result\_box2.insert(END,'NOTHING TO PROCESS IN INPUTBOX !!!')

result\_box1.insert(END,'NOTHING TO PROCESS IN INPUTBOX !!!')

else:

global bar,label,top,quote,status

top=Toplevel(root,bg='white')

top.title('QuiliBot - In Progress...')

top.geometry('500x300')

top.iconbitmap('.img\\icons8-bot-80.ico')

style = ttk.Style()

style.theme\_use('vista')

style.configure("black.Horizontal.TProgressbar", background='blue',bg="white",fg="black")

bar = Progressbar(top, length=200, style='black.Horizontal.TProgressbar')

bar['value'] = 0

bar.place(relx=0.3,rely=0.07)

status=Label(top,bg="white",fg="black")

status.place(relx=0.13,rely=0.07)

label=Label(top,bg="white",fg="black")

label.place(relx=0.45,rely=0.2)

quote=Label(top,font=('Arial',12),text=quotes\_list[random.randint(0,len(quotes\_list)-1)],bg="white",fg="black",justify=CENTER,anchor=CENTER)

quote.place(relwidth=1,relx=0,rely=0.4,relheight=0.6)

result\_box2.delete('1.0',END)

result\_box1.delete('1.0',END)

if pos.get()==-1:

Thread(target=start).start()

else:

Thread(target=start,args=(pos.get(),)).start()

pos.set(-1)

def exit\_(): #exit

ms = messagebox.askquestion('QuiliBot - Exiting...', 'Are You Sure?')

if ms == 'yes':

root.destroy()

def clear\_(): #to clear screen

ms = messagebox.askquestion('QuiliBot - Clear Inputs?', 'Are You Sure?')

if ms == 'yes':

input\_box.delete('1.0',END)

result\_box1.delete('1.0',END)

result\_box2.delete('1.0',END)

########################################################################### GUI

GLabel\_246=tk.Label(root)

ft = tkFont.Font(family='Times',size=23)

GLabel\_246["font"] = ft

GLabel\_246["fg"] = "#00008B"

GLabel\_246["justify"] = "center"

GLabel\_246["text"] = "Bot"

GLabel\_246.place(x=103,y=10,width=84,height=41)

GLabel\_992=tk.Label(root)

ft = tkFont.Font(family='Times',size=23)

GLabel\_992["font"] = ft

GLabel\_992["fg"] = "#0000ff"

GLabel\_992["justify"] = "center"

GLabel\_992["text"] = "Quili"

GLabel\_992.place(x=60,y=10,width=62,height=41)

GLabel\_325=tk.Label(root)

ft = tkFont.Font(family='Times',size=22)

GLabel\_325["font"] = ft

GLabel\_325["fg"] = "#333333"

GLabel\_325["justify"] = "center"

GLabel\_325["text"] = "Paraphrasing Tool"

GLabel\_325.place(x=550,y=20,width=221,height=31)

def keyPress(e): #for word count

words = input\_box.get(1.0, END)

wordcount = len( re .findall( '\w+', words ) )

word\_count1.config(text = f' Words: {wordcount}')

input\_box=tk.Text(root, wrap=WORD)

input\_box["bg"] = "#ffffff"

input\_box["borderwidth"] = "1px"

ft = tkFont.Font(family='Times',size=14)

input\_box["font"] = ft

input\_box["fg"] = "#333333"

input\_box.bind("<Key>", keyPress)

input\_box.place(x=10,y=90,width=641,height=261)

result\_box1=tk.Text(root, wrap=WORD)

result\_box1["bg"] = "#ffffff"

result\_box1["borderwidth"] = "1px"

ft = tkFont.Font(family='Times',size=14)

result\_box1["font"] = ft

result\_box1["fg"] = "#333333"

result\_box1.insert(END,'#'\*10+' R E S U L T '+'#'\*10)

result\_box1.place(x=660,y=90,width=641,height=261)

result\_box2=tk.Text(root, wrap=WORD)

result\_box2["bg"] = "#ffffff"

result\_box2["borderwidth"] = "1px"

ft = tkFont.Font(family='Times',size=14)

result\_box2["font"] = ft

result\_box2["fg"] = "#333333"

result\_box2.place(x=10,y=460,width=1291,height=136)

button\_start=Button(root)

button\_start["bg"] = "#00008B"

button\_start["borderwidth"] = "0px"

ft = tkFont.Font(family='Times',size=23)

button\_start["font"] = ft

button\_start["fg"] = "#ffffff"

button\_start["justify"] = "center"

button\_start["text"] = "Rephrase"

button\_start.place(x=510,y=380,width=291,height=41)

button\_start["command"] = go\_

GLabel\_365=tk.Label(root, borderwidth=2, relief="groove", anchor="w")

GLabel\_365["bg"] = "#ffffff"

GLabel\_365["borderwidth"] = "1px"

ft = tkFont.Font(family='Times',size=14)

GLabel\_365["font"] = ft

GLabel\_365["fg"] = "#333333"

GLabel\_365["justify"] = "left"

GLabel\_365["text"] = "Paraphrased Text"

GLabel\_365.place(x=660,y=60,width=641,height=31)

GLabel\_608=tk.Label(root, borderwidth=2, relief="groove", anchor="w")

GLabel\_608["bg"] = "#ffffff"

GLabel\_608["borderwidth"] = "1px"

ft = tkFont.Font(family='Times',size=14)

GLabel\_608["font"] = ft

GLabel\_608["fg"] = "#333333"

GLabel\_608["justify"] = "left"

GLabel\_608["text"] = "Standard Paraphrase"

GLabel\_608.place(x=10,y=60,width=641,height=31)

GLabel\_427=tk.Label(root, borderwidth=2, relief="groove", anchor="w")

GLabel\_427["bg"] = "#ffffff"

ft = tkFont.Font(family='Times',size=14)

GLabel\_427["font"] = ft

GLabel\_427["fg"] = "#333333"

GLabel\_427["justify"] = "left"

GLabel\_427["text"] = "Synonyms"

GLabel\_427.place(x=10,y=430,width=1291,height=31)

word\_count1=tk.Label(root, borderwidth=2, relief="groove", anchor="w")

word\_count1["bg"] = "#ffffff"

ft = tkFont.Font(family='Times',size=12)

word\_count1["font"] = ft

word\_count1["fg"] = "#333333"

word\_count1["justify"] = "left"

word\_count1["text"] = " Words:"

word\_count1.place(x=10,y=350,width=641,height=21)

word\_count2=tk.Label(root, borderwidth=2, relief="groove", anchor="w")

word\_count2["bg"] = "#ffffff"

ft = tkFont.Font(family='Times',size=12)

word\_count2["font"] = ft

word\_count2["fg"] = "#333333"

word\_count2["justify"] = "left"

word\_count2["text"] = " Words:"

word\_count2.place(x=660,y=350,width=641,height=21)

menu1 = Menu(root, bg='red', fg='blue', activeforeground='#209EFF', font=('Areial', 10))

sub1 = Menu(menu1, tearoff=0, activebackground='white', activeforeground='#209EFF', bg='white', )

menu1.add\_command(label='CLEAR ', command=clear\_,)

menu1.add\_command(label='EXIT', command=exit\_, )

root.config(menu=menu1)

pos=IntVar()

pos.set(-1)

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

img=PhotoImage(file=".img\\rsz\_icons8-bot-80.png")

GLabel\_671=tk.Label(image=img)

GLabel\_671.pack()

GLabel\_671.place(x=10,y=10,width=41,height=41)

app = App(root)

root.mainloop()

**Curriculum Vitae**

**Giomel A. Ayes   
Barangay. San Cristobal San Pablo City   
Mobile: 09560757348**

**Email:** [**gioayes@yahoo.com**](mailto:gioayes@yahoo.com)

**PERSONAL INFORMATION:**

**Birthdate: August 11, 2001  
Birthplace: San Pablo City  
Marital Status: Single  
Sex: Male  
Religion: Catholic**  
**CAREER OBJECTIVES:**

A highly organized and hard-working individual looking for a responsible position to gain experience.

**EDUCATIONAL BACKGROUND:**

**2006 -2012**   **San Cristobal Elementary School  
 Barangay. San Cristobal San Pablo City**

**2012 - 2019 LAGUNA COLLEGE**

**Paseo de Escudero Street, San Pablo City**

**2019 - 2023 LAGUNA COLLEGE**

**Paseo de Escudero Street, San Pablo City**

**Bachelor of Science in Computer Science**

**SKILLS:**

**- Active Listening**

**- Collaboration**

**- Adaptability**

**- Decision Making**

**- Customer Service Skills**

****

**JULES CARLO Q. ESPINILI**

Brgy. San Benito, Alaminos

Laguna, 4001, Philippines

Mobile: 09173172771

Email: julessp23@gmail.com

**CAREER OBJECTIVES**

To succeed in an environment of growth and excellence and earn a job which provides me a job satisfaction and self-development and help me achieve personal as well as organization goals.

**EDUCATIONAL BACKGROUND**

**2019 - Present** LAGUNA COLLEGE

Paseo de Escudero Street, San Pablo City

**Bachelor of Science in Computer Science**

**2012 - 2019** LAGUNA COLLEGE

Paseo de Escudero Street, San Pablo City

**2006 -2012** SAINT PAUL LEARNING SCHOOL

Alaminos Heights, Alaminos, Laguna

**SKILLS**

* Ability to work in harmony with co-student
* Flexibility
* Eager and willing to add to their knowledge base and skills
* Loyalty
* Good communication skills

**CHARACTER REFERENCE**

Ms. Marriel Bondad - Baet  
IT Instructor, Laguna College, San Pablo City  
0945-832-1642

****