COSC 310

Individual Project - Report

Sarvagya Pandey

Note: To see what's new for 'Individual Project', jump to page 27.

Introduction

The goal of this project is to create an interactive conversational agent, or a 'chat bot' using an object-oriented programming language. The chat bot needs to be assigned a role and is expected to respond to and hold a conversation with the user for thirty or more turns of dialogue. To create the chat bot, our team decided to make use of the Python programming language along with the help of some Natural Language Processing (NLP) frameworks and toolkits. We decided to assign the bot the role of Justin Trudeau, the current Prime Minister of Canada, with whom the user can interact with and ask questions about his personal and political interests in the past, present and the future, along with a few general questions and greetings. The GitHub Repository for this project can be found here: https://github.com/COSC310Group23/COSC310-Interactive-Conversational-Agent

Choice of Software Development Life Cycle (SDLC)

After some discussion, our team decided on using the **Waterfall Model** for developing the Interactive Chat Bot. Our choice was based on a few key points: Small Team size, small project size, a clear set of unchanging requirements, and familiarity with the software used for making the project. Along with these key points, we felt that organizing the workflow with the help of distinctive phases will help some of our members become used to developing software in a team environment, as opposed to having to go back on forth between development phases in other SDLCs such as the Agile model.

Phases of Development

Requirements Analysis

- **Software**: Interactive Chat Bot that can respond to user input with appropriate messages.
    - Must be created using an object-oriented programming language.

- **Role Assignment**: Chat Bot must be assigned the role of a specific person and must answer questions within the given context.

- **User Input**: User must be able to converse with the Chat Bot by entering general greetings or questions specific to the bot's assigned role.
    - User must be able to quit the software by entering a command, which must be provided to the user at the beginning of the conversation.

- **Goal**: User must be able to hold a complete and coherent conversation with the Chat Bot for a minimum of 30 turns of dialogue.

System Design

- **Programming Language**: Python will be used to develop the Interactive Chat Bot.
    - Frameworks and Toolkits such as Natural Language Toolkit (NLTK) and TensorFlow will be used to process user input and 'train' the bot's responses.

- **Role**: Chat Bot will be assigned the role of 'Justin Trudeau', Prime Mister of Canada.

- **Dialogue**: Possible variations of all possible questions and the Bot's responses will be stored in a text file.

- **I/O**: User will type their inputs into the console. The Chat Bot's response will also be printed out to the console.
    - User will be able to end the conversation by entering a command such as 'quit' into the console.

Implementation

- Create text file that contains 'patterns' that the Bot should understand and 'responses' that the Bot should output based on the most appropriate pattern.

- Implement a class that uses the NLP algorithms to get a probability for a matching tag based on a user's input

- Implement a method to take in user input, check which pattern it has the highest similarity to from all the pre-written dialogue.

- Output a random response from the list of responses pertaining to the pattern on the console.

- Create a README file containing a brief explanation of the project and the libraries used.

- Document the code by adding comments explaining the features of various classes/methods, etc.

Testing

- Test for bot's responses by entering existing questions or 'patterns' from the text file containing all the dialogue.
  - Check if any of the existing patterns fails to generate an appropriate response.

- Test for an appropriate response when an unrecognizable pattern is entered by user.

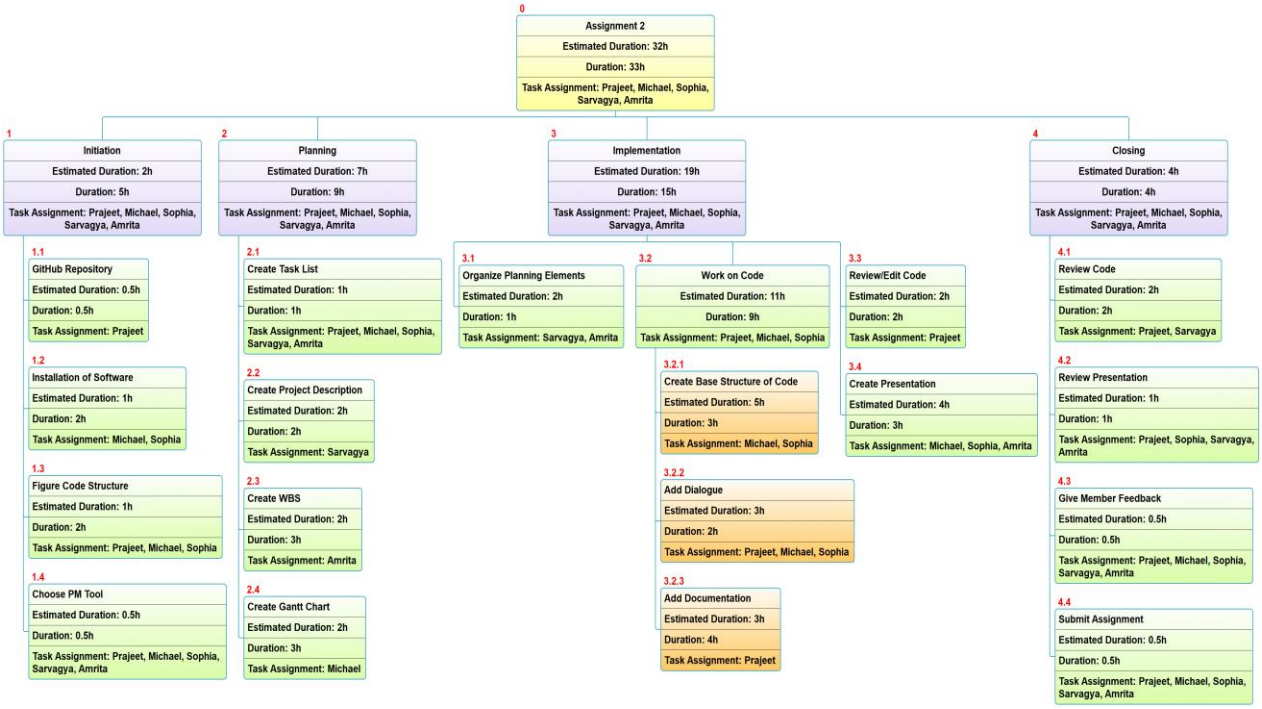
- Test 'quit' command that ends conversation on input.

- Review Code and Documentation.
  - Check for possible grammar/language errors in the dialogue file.
  - Check for redundancies/logic errors in code.

- Check for exceptions and crashes on abnormal input.

Project Breakdown

Work Breakdown Structure



	WBS	Name	Task Assignment	Estimated Task Duration (h)	Actual Task Duration (h)	Start	Finish
0	**0**	**Assignment 2**	**Prajeet, Michael, Sophia, Sarvagya, Amrita**		**59.5h**	**2021-02-22**	**2021-03-04**
1	**1**	**Initiation**	**Prajeet, Michael, Sophia, Sarvagya, Amrita**		**5h**	**2021-02-22**	**2021-02-22**
2	1.1	GitHub Repository	Prajeet	0.5	0.5h	2021-02-22	2021-02-22
3	1.2	Installation of Software	Michael, Sophia	1	2h	2021-02-22	2021-02-22
4	1.3	Figure Code Structure	Prajeet, Michael, Sophia	1	2h	2021-02-22	2021-02-22
5	1.4	Choose PM Tool	Prajeet, Michael, Sophia, Sarvagya, Amrita	0.5	0.5h	2021-02-22	2021-02-22
6	**2**	**Planning**	**Prajeet, Michael, Sophia, Sarvagya, Amrita**		**9h**	**2021-02-23**	**2021-02-24**
7	2.1	Create Task List	Prajeet, Michael, Sophia, Sarvagya, Amrita	1	1h	2021-02-23	2021-02-23
8	2.2	Create Project Description	Sarvagya	3	2h	2021-02-24	2021-02-24
9	2.3	Create WBS	Amrita	2	3h	2021-02-24	2021-02-24
10	2.4	Create Gantt Chart	Michael	2	3h	2021-02-24	2021-02-24
11	**3**	**Implementation**	**Prajeet, Michael, Sophia, Sarvagya, Amrita**		**22h**	**2021-02-26**	**2021-03-04**
12	3.1	Organize Planning Elements	Sarvagya, Amrita	2	2h	2021-02-26	2021-02-26
13	**3.2**	**Work on Code**	**Prajeet, Michael, Sophia**		**20h**	**2021-02-26**	**2021-02-27**
14	3.2.1	Create Base Structure of Code	Michael, Sophia	4	5h	2021-02-26	2021-02-26
15	3.2.2	Add Dialogue	Prajeet, Michael, Sophia	3	4h	2021-02-27	2021-02-27
16	3.2.3	Add Documentation	Prajeet	3	4h	2021-02-26	2021-02-26
17	3.3	Review/Edit Code	Prajeet	2	3h	2021-03-01	2021-03-01
18	3.4	Create Presentation	Michael	4	4h	2021-03-04	2021-03-04
19	**4**	**Closing**	**Prajeet, Michael, Sophia, Sarvagya, Amrita**		**3.5h**	**2021-03-02**	**2021-03-04**
20	4.1	Review Code	Prajeet	2	2h	2021-03-02	2021-03-03
21	4.2	Review Presentation	Prajeet, Sophia, Sarvagya, Amrita	1	1h	2021-03-03	2021-03-04
22	4.3	Give Member Feedback	Prajeet, Michael, Sophia, Sarvagya, Amrita	0.5	0.5h	2021-03-03	2021-03-04

Gantt Chart

TASK NUMBER	WBS NUMBER	TASK TITLE	TASK PARTICIPANTS	START DATE	DUE DATE	DURATION	PREDECESSORS	WEEK 1 (Feb 22 - Mar 28)							WEEK 2 (Mar 1 - Mar 7)						
								M	T	W	TH	F	SA	SU	M	T	W	TH	F	SA	SU
1	1	Initiation				1 day															
2	1.1	GitHub Repository	Prajeet	2/22/21	2/23/21	1 day	-														
3	1.2	Installation of software	Michael, Sophia	2/22/21	2/23/21	1 day	-														
4	1.3	Figure Code Structure	Prajeet, Michael, Sophia	2/22/21	2/23/21	1 day	-														
5	1.4	Choose PM Tool	Prajeet, Michael, Sophia, Sarvagya, Amrita	2/22/21	2/23/21	1 day	-														
6	2	Planning				3 days															
7	2.1	Create List of Tasks	Prajeet, Michael, Sophia, Sarvagya, Amrita	2/23/21	2/24/21	1 day	2;3;4;5														
8	2.2	Create Project Description	Sarvagya	2/24/21	2/26/21	2 days	7														
9	2.3	Create WBS	Amrita	2/24/21	2/26/21	2 days	7														
10	2.4	Create Gantt Chart	Michael	2/24/21	2/26/21	2 days	7														
11	3	Implementation				6 days															
12	3.1	Organize Planning Elements	Sarvagya, Amrita	2/26/21	2/27/21	1 day	8;9;10														
13	3.2	Work on Code	Prajeet, Michael, Sophia	2/26/21	2/28/21	2 days	8;9;10														
14	3.2.1	Create Base Structure of Code	Michael, Sophia	2/26/21	2/27/21	1 day	8;9;10														
15	3.2.2	Add Dialogue	Prajeet, Michael, Sophia	2/27/21	2/28/21	1 day	14														
16	3.2.3	Add Documentation	Prajeet	2/26/21	2/28/21	1 day	14														
17	3.3	Review/Edit Code	Prajeet	2/28/21	3/2/21	2 days	15;16														
18	3.4	Create Presentation	Michael, Sophia, Amrita	3/2/21	3/4/21	2 days	17														
19	4	Closing				2 days															
20	4.1	Review Code	Prajeet, Sarvagya	3/4/21	3/5/21	1 day	18														
21	4.2	Review Presentation	Prajeet, Sophia, Sarvagya, Amrita	3/4/21	3/5/21	1 day	18														
22	4.3	Give Member Feedback	Prajeet, Sophia, Michael, Amrita, Sarvagya	3/4/21	3/5/21	1 day	18														
23	4.4	Submit Assignment	Prajeet, Sophia, Michael, Amrita, Sarvagya	3/5/21	3/6/21	1 day	20;21;22														

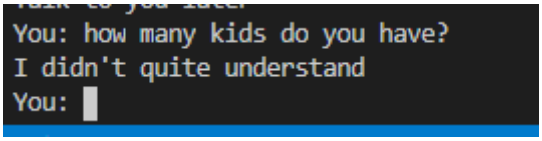
Software Limitations

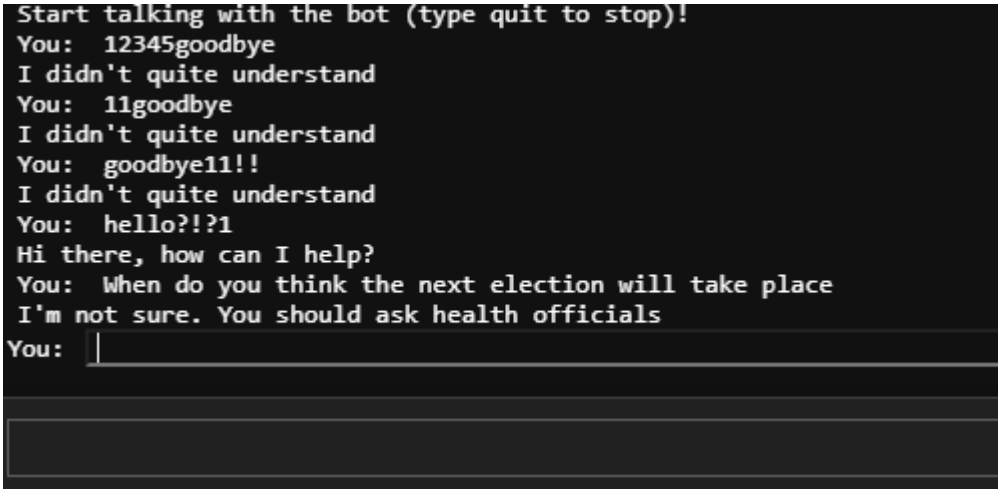
- Unable to handle some synonyms.
- Unable to handle input not included in the pre-written dialogue file.
- Unable to handle complex phrases such as idioms, similes, etc.
- Unable to handle major incorrect spellings.

Sample Output

Working as Intended

Negative/Abnormal Behaviour





Assignment 3

We have implemented some new features to further enhance the user's experience and to improve the flow of conversation with the chat bot. A Graphical User Interface (GUI) has been used in place of the terminal for input/output, so that the user has a more pleasant experience when talking with the bot. The bot also utilizes Stanford's CoreNLP Toolkit to make use of features such as Sentiment Analysis, which allows the bot to provide appropriate responses to statements that are not present in the question bank, and Part-of-speech(POS) tagging which helps the bot recognize certain key words as parts of speech(useful for performing other functions). Synonym recognition was also implemented with the use of NLTK's WordNet. This feature helps the bot understand a wider range of vocabulary and sentences while providing the correct response.

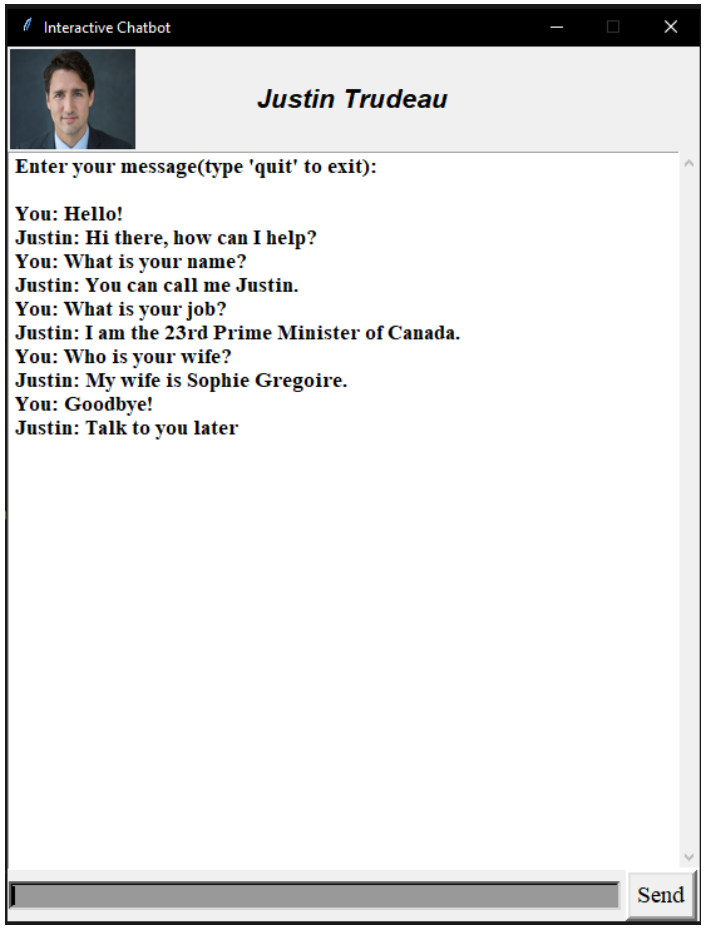
New Files

- **stanfordload.py -** Establishes connection to Stanford's CoreNLP Server
- **gui.py** - Comprises of functions that make the GUI for the chat bot
- **sentiment.py** - Contains functions that conduct sentiment analysis on the user's input statement using Stanford's CoreNLP Toolkit
- **syn_recognition.py -** Contains functions that tag parts-of-speech in user's input, and creates different sentences with different combinations of synonyms
- **test_chatbot.py -** Contains unit tests for major functions and classes to validate them

New features

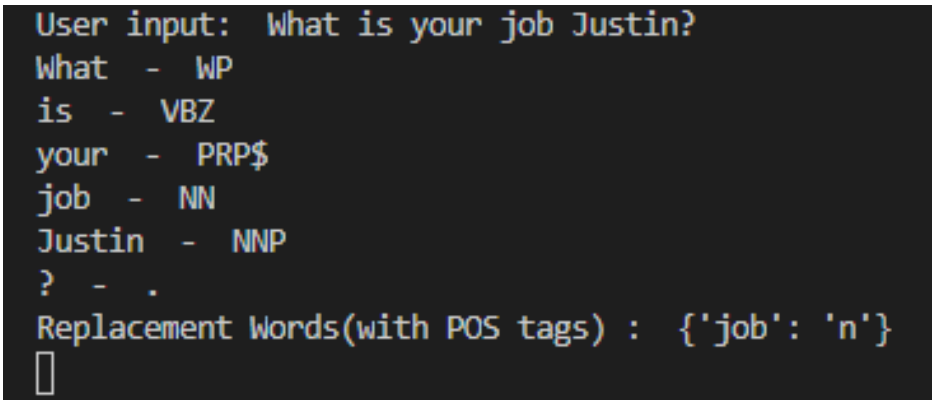
Graphical User Interface (GUI)

A simple GUI was implemented to provide a much more appealing way of using the chat bot. This GUI was made using Tkinter and image functionalities from Pillow were utilized. It consists of a medium-sized window with a text box(to display the conversation), an entry box(for the user to type their message in), a 'Send' button which can be pressed to send the typed message to the bot(They can also choose to press 'Enter'), and various other smaller features such as a label containing the image and name of the person who the bot is based on, and a scrollbar for the user to access/read older sections of the conversation. Here is an image showcasing the GUI with a few turns of conversation:



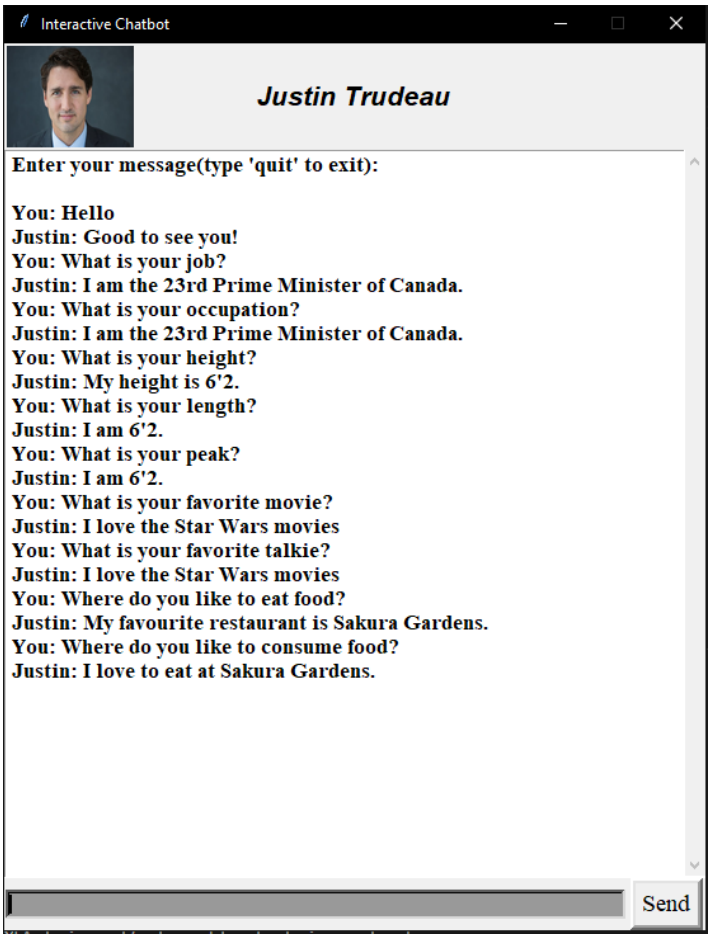
Part-of-Speech (POS) Tagging

Stanford's CoreNLP Toolkit is used to tokenize and categorize each word in the user's input sentence to specific parts of speech. If the word's POS matches with the one specified, it is placed into a dictionary of words that would be later used for another feature. The values assigned to each word is their corresponding POS, which is converted from Stanford's CoreNLP format to NLTK.WordNet's format. Here is an image displaying sample output for a possible sentence that the user could input, and its breakdown into different parts of speech, along with the dictionary of key words(Note: The following output is not visible in the final version of the application. This image is for demonstration purposes only):



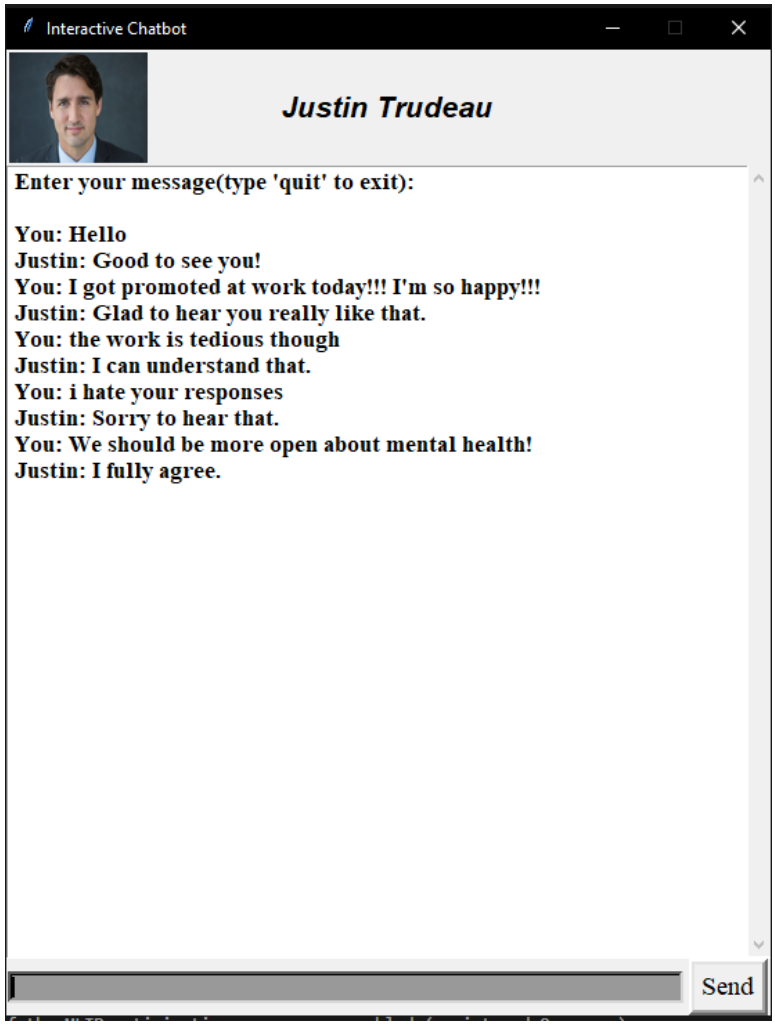
Synonym Recognition

A lexical database in the form of NLTK's WordNet is used to find synonyms of key words from the user's input sentence (see above feature on how to get these 'key words'). The original word is then replaced by each of these synonyms to form a new sentence, each of which is stored in a list. This list is returned to chatbot.py where each sentence is incrementally sent to the bot until it recognizes one and forms an appropriate response. If none of the sentences manage to get recognized, an appropriate message from the bot is shown to the user instead (Note: The first sentence in the list is always the user's original input). The following image showcases the bot's ability to recognize different words as synonyms of each other and provide the same type of response to all sentences:



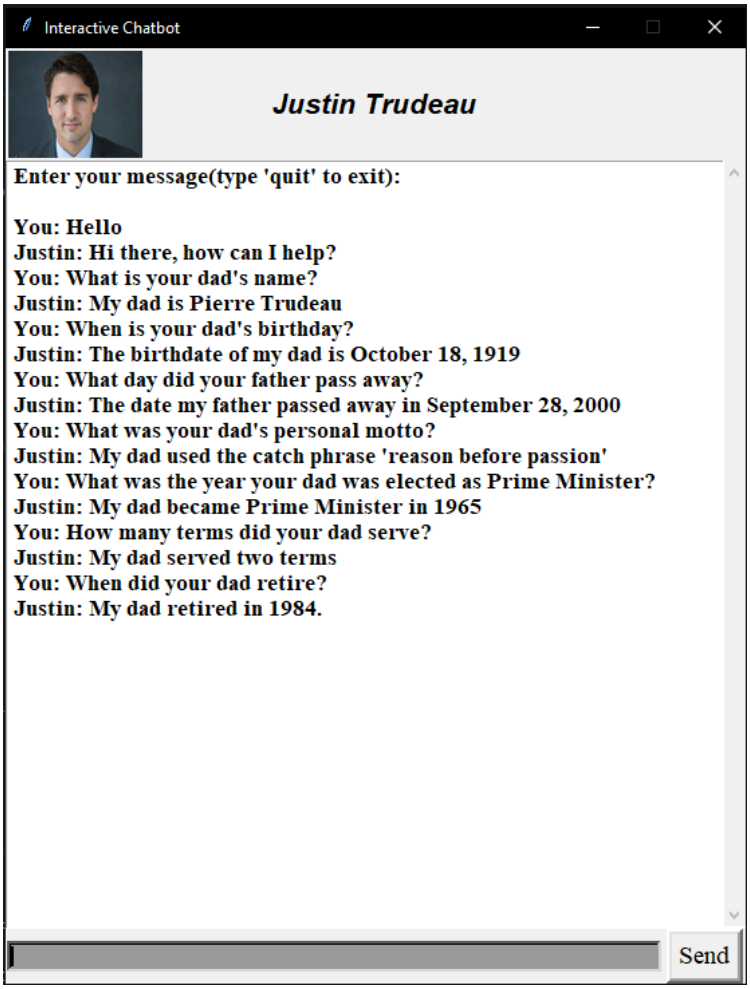
Sentiment Analysis

The CoreNLP Toolkit is also used to perform Sentiment Analysis; it detects words in the sentence that have different levels of sentiment and provides an integer value from 0-3 based on how positive or negative the sentiment is. The bot then responds appropriately based on the sentiment number generated from the sentence. Here is a sample image of the bot responding to different levels of sentiment with the user's input sentences:



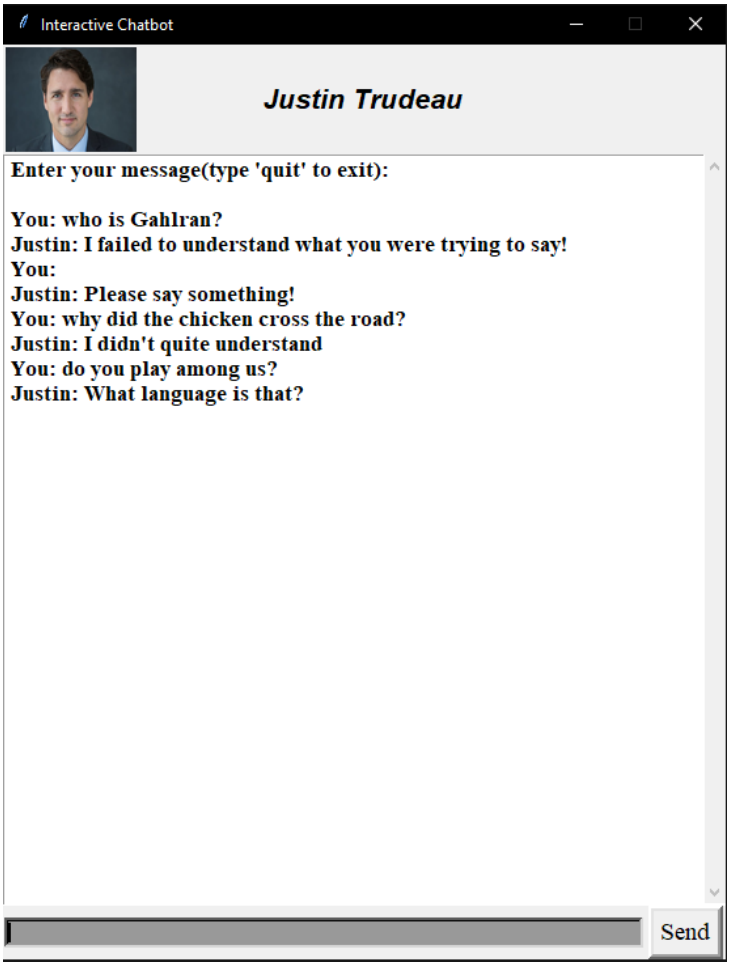
Extra Topic of Conversation

The bot now supports conversation about his father, Pierre Trudeau and his relationship with him. The following image demonstrates some new possible combinations of dialogue with the bot:



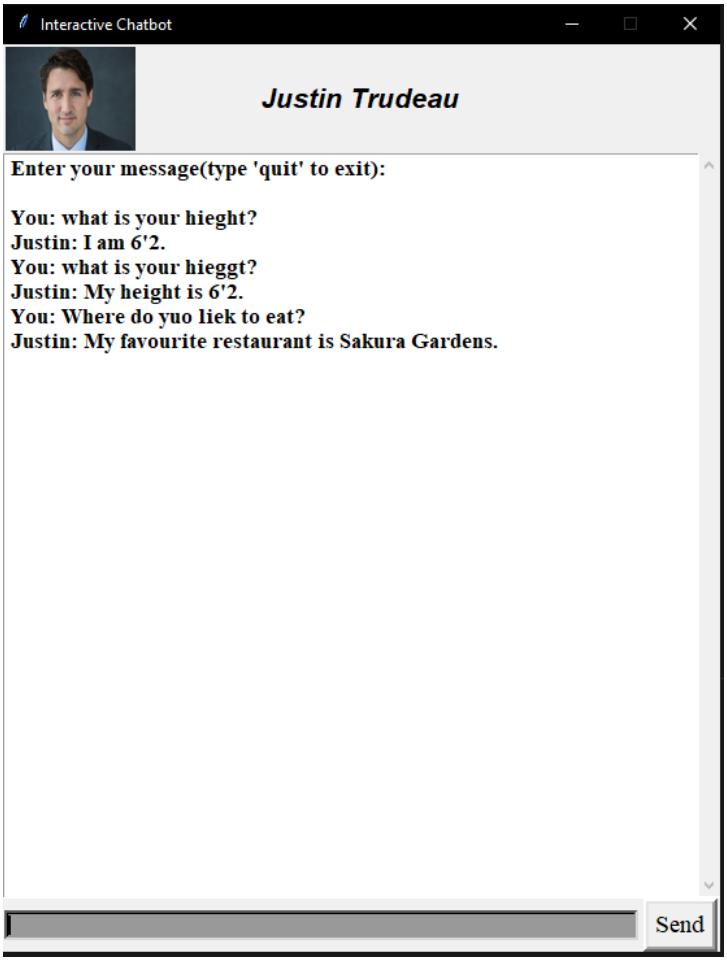
Varied 'Fail' Responses

When the bot fails to recognize a sentence input by the user, it now has the possibility to reply with 5 different responses instead of just one. The image below showcases this new feature:



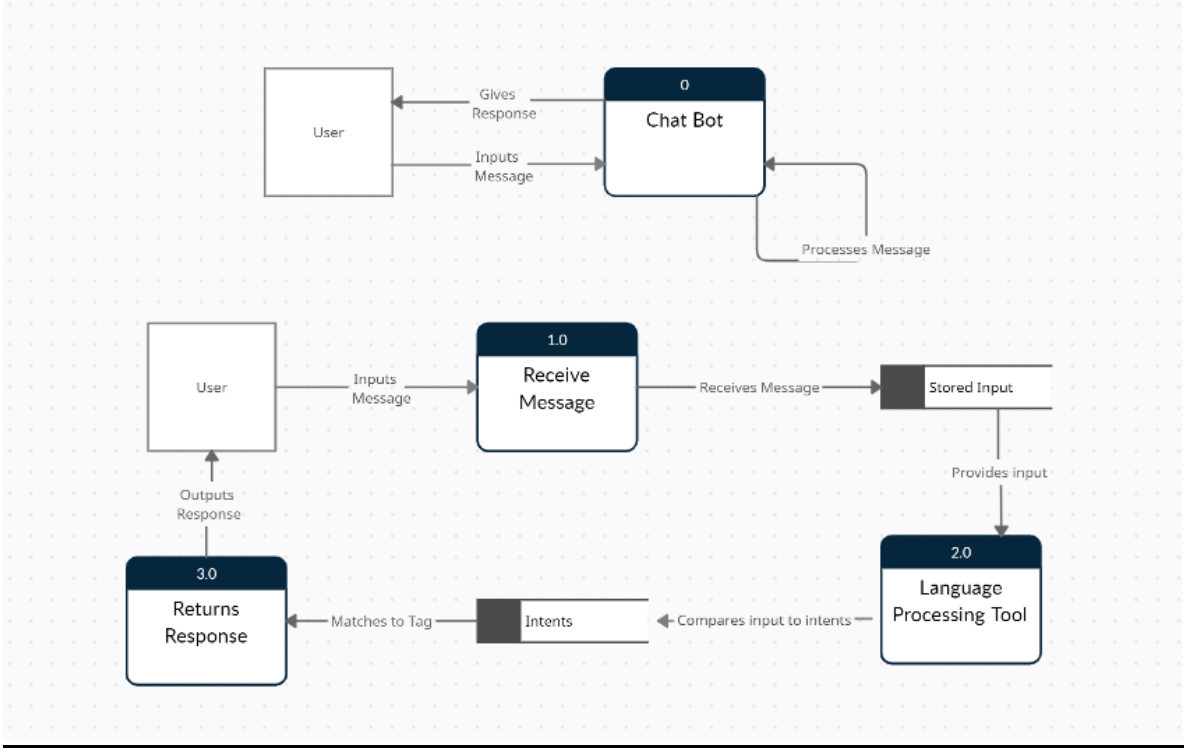
Spelling Error Detection

The Lancaster Stemmer Algorithm tokenizes the words in the user's input sentence, stems them to provide the root words, removes 'stop words' and as a result, can handle minor spelling errors. The following image shows the bot's ability to respond appropriately even if the input sentence has minor errors in spelling:

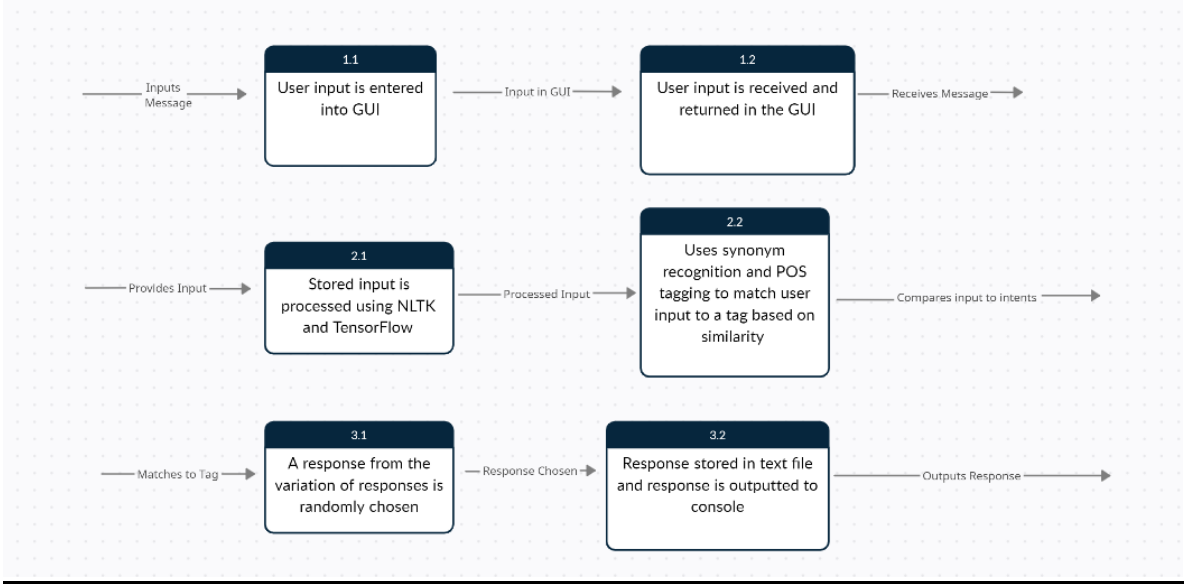


Data Flow Diagrams

Level 0



Level 1


Inputs Message
1.1
User input is entered into GUI
Input in GUI
1.2
User input is received and returned in the GUI
Receives Message
Provides Input
2.1
Stored input is processed using NLTK and TensorFlow
Processed Input
2.2
Uses synonym recognition and POS tagging to match user input to a tag based on similarity
Compares input to intents
Matches to Tag
3.1
A response from the variation of responses is randomly chosen
Response Chosen
3.2
Response stored in text file and response is outputted to console
Outputs Response

Unit-Testing

The file test chatbot.py is used to run unit testing on the bot. The file can be run directly after the Stanford NLP server is started. There is a set up process that will load in the intents.json file to grab some expected responses that will be tested. There is a total of seven tests that are run, covering major parts of the bot's functionality.

```
def setUp(self):
    self.l = load("intents.json")
    self.data = self.l.getData()
    for t in self.data["intents"]:
        if t['tag'] == "greeting":
            self.responses = t['responses']
        if t["tag"] =="music":
            self.responsesmusic = t['responses']
```

Test Basic Chat Functionality

The test chat function tests basic chat inputs and responses. It will assert true if an input matches with an expected response. For this function we've decided to check responses for greetings. As the bot is programmed to throw out a random relative response from the matching tag, we must check if the response we receive is inside the list of responses. If it is, the function will assert it as true.

```
def test_chat(self):
    self.assertTrue(chatbot.chat("How are you") in self.responses)
    self.assertTrue(chatbot.chat("Is anyone there?") in self.responses)
    self.assertTrue(chatbot.chat("hello") in self.responses)
    self.assertTrue(chatbot.chat("good day") in self.responses)
    self.assertTrue(chatbot.chat("whats up?") in self.responses)
```

Testing Tags

Each response that we receive is related to a tag. This tag is used to output the response, so we can check if the input that we give matches the expected tag. Using assert equals we check if 'chatbot.tag' is equal to the relevant tag.

```
def test_tag(self):
    chatbot.chat("How are you")
    self.assertEqual(chatbot.tag,"greeting")
    chatbot.chat("see you later")
    self.assertEqual(chatbot.tag,"goodbye")
    chatbot.chat("how old")
    self.assertEqual(chatbot.tag,"age")
```

Testing Sentiment Analysis

Sentiment analysis works by returning a statement relevant to an input. This is tested by using assert equals as the responses are all hard coded, and work based on a scale from 0-3.

```
def test_sentiment(self):
    self.assertEqual(chatbot.chat("I love your work"),"Glad to hear you really like that.")
    self.assertEqual(chatbot.chat("I hate you"),"I can understand that.")
    self.assertEqual(chatbot.chat("I like this conversation"),"I fully agree.")
```

Testing Capitalization

For this test, we input the same statement with different capitalization. We then check if the response we receive is in the expected responses list.

```
def test_capitalization(self):
    self.assertTrue(chatbot.chat("hOW aRE yOU") in self.responses)
    self.assertTrue(chatbot.chat("HOW ARE YOU") in self.responses)
    self.assertTrue(chatbot.chat("how are you") in self.responses)
```

Testing Punctuation

Like the previous test, we input the same statement, with different punctuation and check if the expected response is in the responses list.

```
def test_punctuation(self):
    self.assertTrue(chatbot.chat("how are you.") in self.responses)
    self.assertTrue(chatbot.chat("how are you!") in self.responses)
    self.assertTrue(chatbot.chat("how are you?") in self.responses)
```

Testing Synonyms

For this test, we must put in similar sentences, with only a few words swapped around that are not in our expected inputs list. Then we can check if the response we receive is still in the responses list, telling us that replacing with synonyms worked.

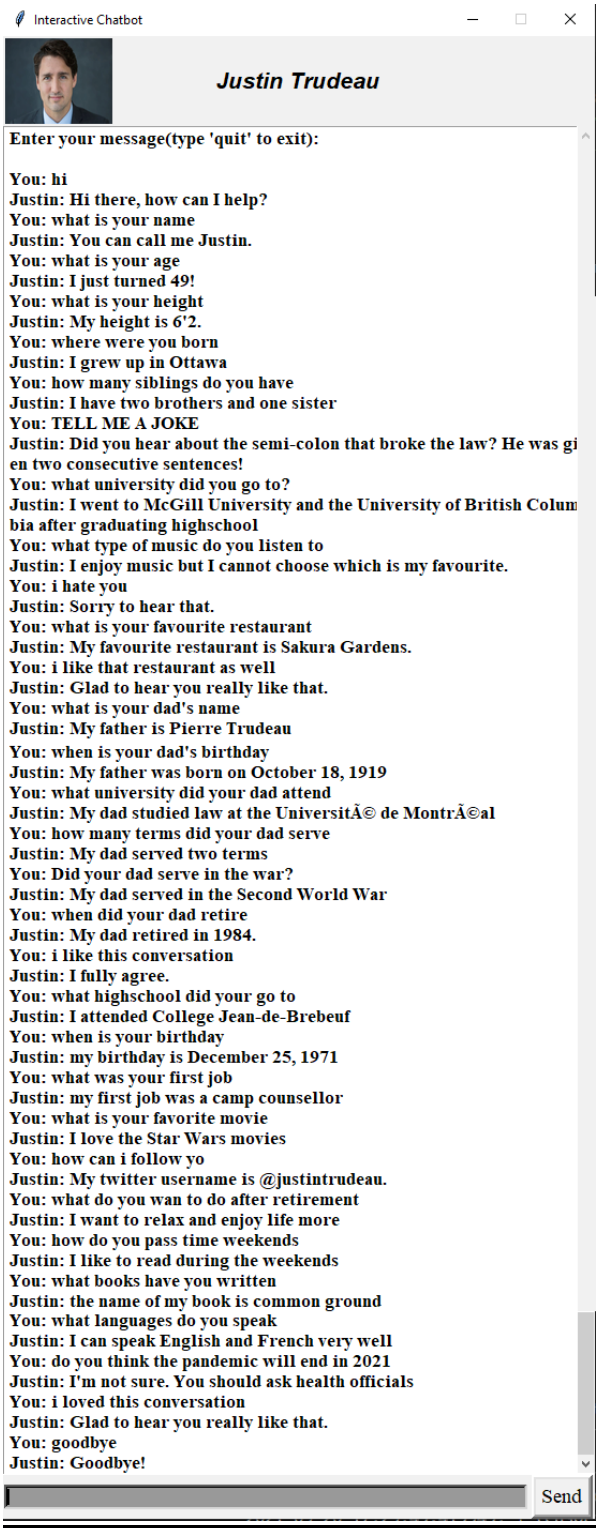
```
def test_synonyms(self):
    self.assertTrue(chatbot.chat("what type of music do you listen to") in self.responsesmusic)
    self.assertTrue(chatbot.chat("what tunes do you listen to") in self.responsesmusic)
```

Testing Out of Scope Responses

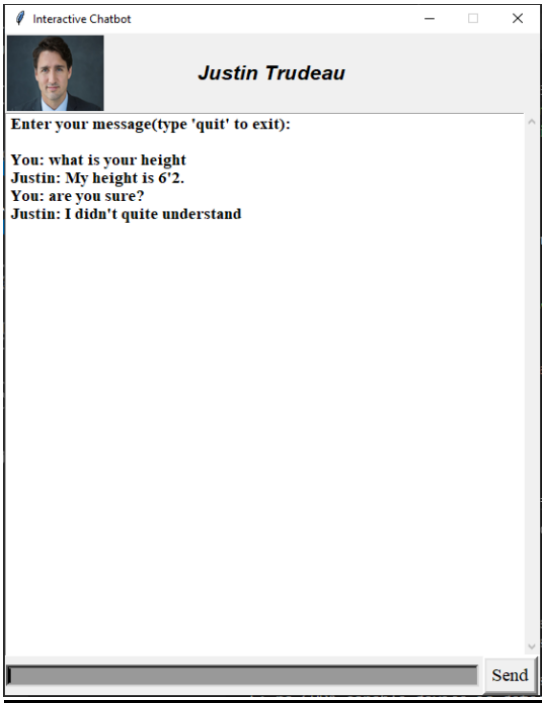
This test will check that the bot gives an appropriate response if it can't understand what the input is. It asserts true if the expected response is in 'chatbot.others' list.

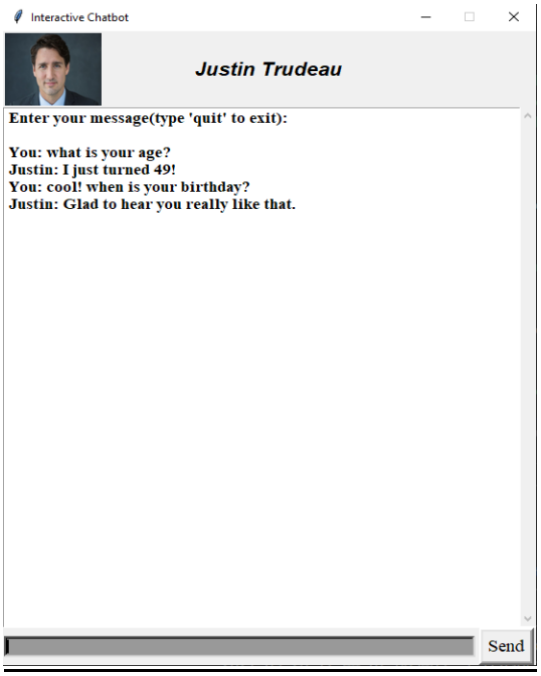
```
def test_failResponses(self):
    self.assertTrue(chatbot.chat("Who is Gahlran?") in chatbot.others)
    self.assertTrue(chatbot.chat("do you play among us?") in chatbot.others)
```

Sample Output (Normal Behaviour)



Sample Output (Abnormal/Unexpected Behaviour)





Limitations of Software

- Unable to recognize multi-sentence inputs.
- Unable to handle major spelling errors.
- Unable to understand vague questions.

Extractable Features (API)

- Getting a list of sentences with similar meanings (using synonyms) from 'syn_recognition.py'
- Getting a list of all the Nouns in each sentence from 'syn_recognition.py'
- Checking whether a sentence is a question or a sentiment using 'sentiment.py'
- Generating a sentiment score for a given sentence from 'sentiment.py'
- Generate an array of words in their root form from a given sentence using 'load.py'

Individual Project

For the Individual project, implementations of 2 different APIs were added as features for the Chatbot. With the help of the 'Tweepy' library, the Twitter API was implemented to allow users to check out Justin Trudeau's recent activity on Twitter, and to also find out recent tweets concerning him. This implementation can get Trudeau's latest tweet, the number of likes and retweets it has, and the bot's reaction based on them. It can also pull the latest tweet regarding Trudeau with the help of Twitter's keyword search feature, which the bot then performs sentiment analysis on and in addition to the tweet, also provides a basic reaction to the tweet's contents.

The Wikipedia API was also implemented for users to be able to learn more about Trudeau's political references and important political definitions, as well as to allow the user to ask the bot for any definition that they wish to learn about. The bot provides an appropriate error message if the definition page does not exist or if the user provides invalid input when asking for the definition. This implementation uses the 'Wikipedia' library for Python.

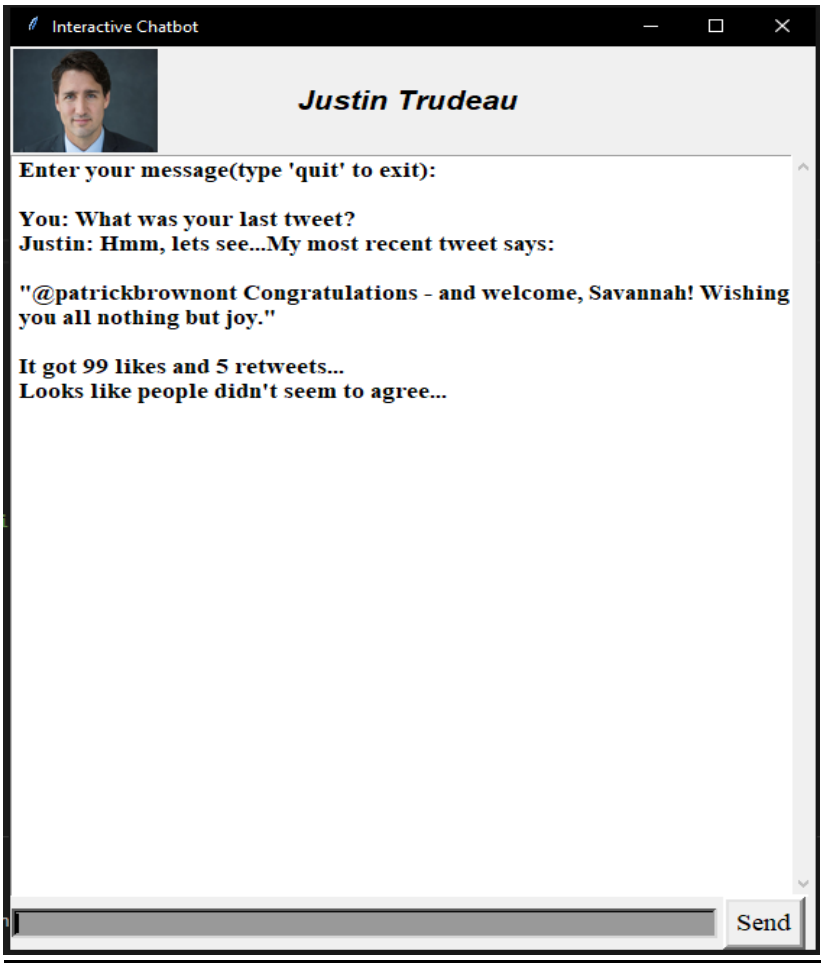
New Files

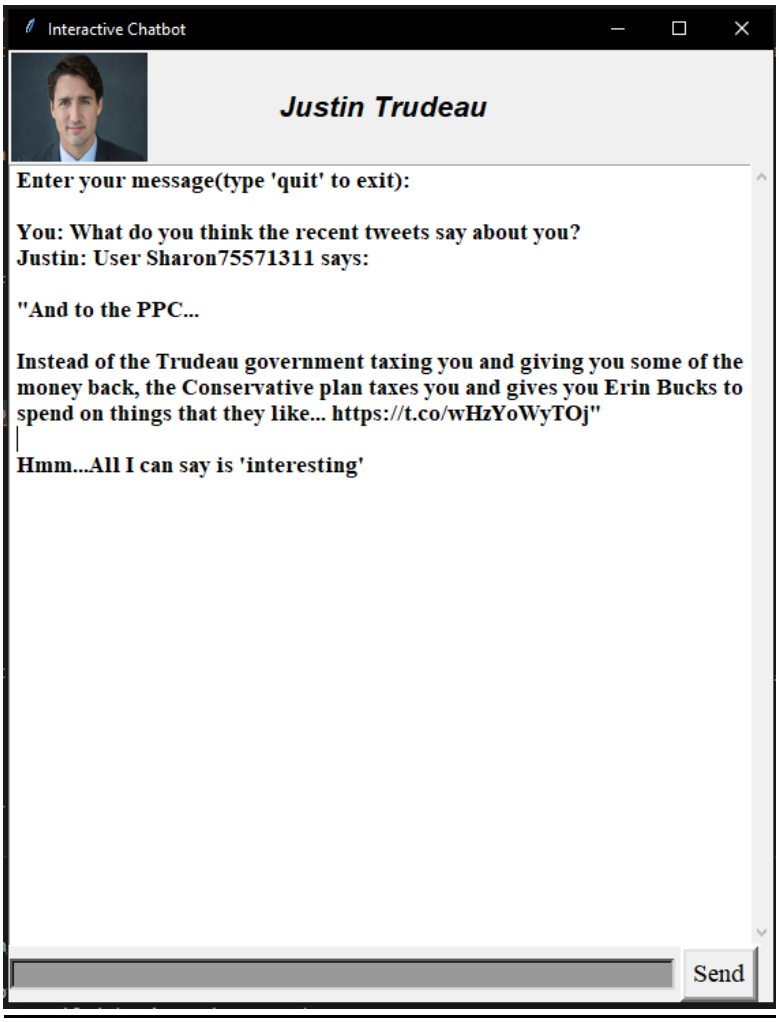
- **get_tweets.py** - Contains functions that use the twitter API to get recent tweets from or about Justin Trudeau.
- **wikiAPI.py** - Contains functions that return useful political definitions or user-requested definitions from Wikipedia.

New Features

Recent Tweets (Twitter API)

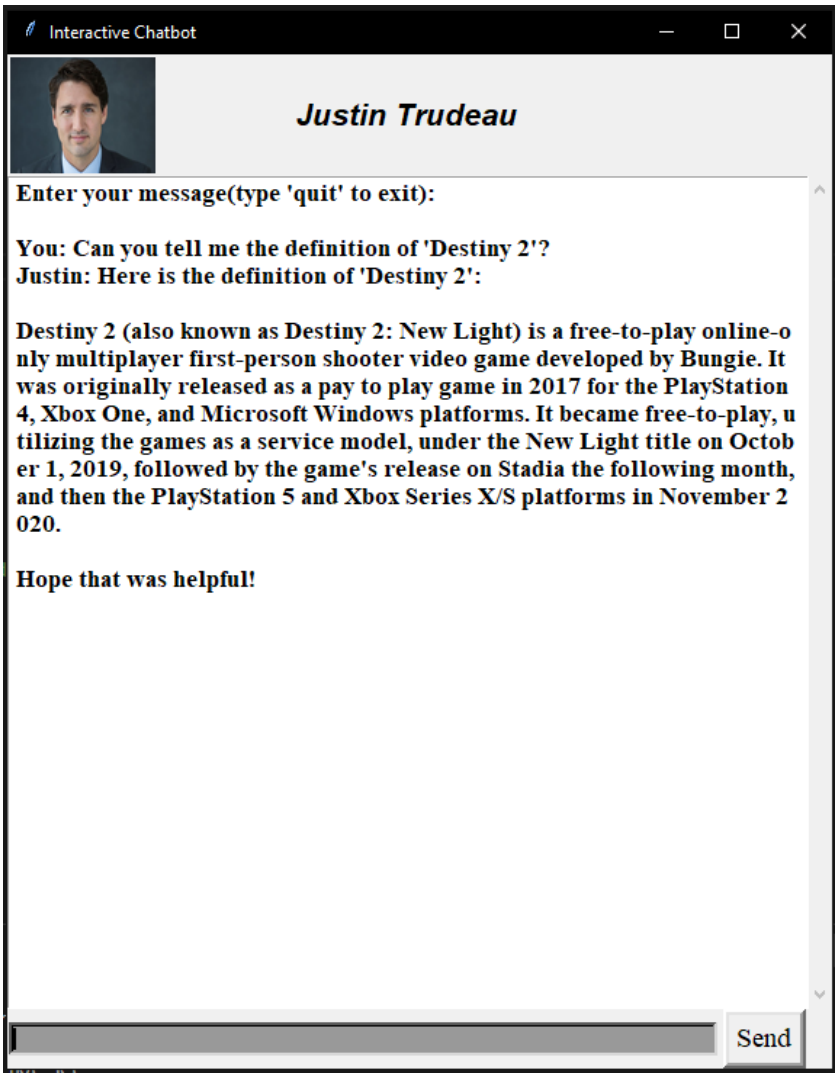
With the use of Twitter API and the Tweepy library for Python, the user can request the bot to show Justin Trudeau's latest tweets, the number of likes and retweets it has, and whether the bot thinks it performed well or not (based on the number of likes and retweets). The user can also request the bot to show the most recent tweet about Justin Trudeau, list the user who posted the tweet, perform basic 'Sentiment Analysis' on the tweet by using a previously implemented feature(see 6) to get the bot's reaction to the tweet. Here ae a few images that showcase these 2 features involving the Twitter API:




Interactive Chatbot
Justin Trudeau
Enter your message(type 'quit' to exit):
You: What do you think the recent tweets say about you?
Justin: User Sharon75571311 says:
"And to the PPC...
Instead of the Trudeau government taxing you and giving you some of the money back, the Conservative plan taxes you and gives you Erin Bucks to spend on things that they like... https://t.co/wHzYoWyTOj"
Hmm...All I can say is 'interesting'
Send

Definitions (Wikipedia API)

With the use of the 'Wikipedia' library for Python and the Wikipedia API itself, the user can either request the bot for an important political definition (chosen randomly from a pre-defined list of phrases), or they can ask the bot for the definition of any word or phrase they would like. For the second feature, if the definition's wiki page does not exist, the bot replies with a suitable error message. Here are some example images for these features:


Interactive Chatbot
Justin Trudeau
Enter your message(type 'quit' to exit):
You: Can you tell me the definition of 'Destiny 2'?
Justin: Here is the definition of 'Destiny 2':
Destiny 2 (also known as Destiny 2: New Light) is a free-to-play online-only multiplayer first-person shooter video game developed by Bungie. It was originally released as a pay to play game in 2017 for the PlayStation 4, Xbox One, and Microsoft Windows platforms. It became free-to-play, utilizing the games as a service model, under the New Light title on October 1, 2019, followed by the game's release on Stadia the following month, and then the PlayStation 5 and Xbox Series X/S platforms in November 2020.
Hope that was helpful!
Send