Awesome-Bot, a Conversational Agent

University of Kent, Computer Science Department, Canterbury

François Caïcedo

Supervisors: Jason R. C. Nurse and Rahime Belen-Saglam

June 2020

Abstract

You probably interacted with a chatbot whether you knew it or not. For example, you are looking for a product on your computer and a window pops up on the screen asking if you need help. Or maybe you are about to go to a concert, and you use your smartphone to request a ride via chat? Or maybe you have used voice commands to order a coffee at your local bar and you have received a reply telling you when your order will be ready and how much it will cost. These are all examples of scenarios in which you might encounter a chatbot.

In 2020, 80% of companies will use chatbots for customer interactions. This increase in demand for chatbot is not only linked to the constant improvement of technology and artificial intelligence. The success of chatbots is mainly due to the fact that they meet consumer needs and customer service challenges at the same time.

This dissertation presents Awesome-Bot, a conversational agent able to speak about any subject it ever studied. The focus was to direct the research towards medicine. This dissertation details Awesome-Bot from the software architecture to the behaviour of the artificial intelligence.

The first chapter is an introduction to conversational agents or chatbots, a review of similar work is conducted in the second chapter, chapter three focuses on the design and implementation of Awesome-Bot, chapter four presents the behaviour of the agent in testing environment, and the conclusion serves as last chapter.

Acknowledgements

I would like to thank my supervisors, Jason R. C. Nurse and Rahime Belen-Saglam, for their guidance and kindness during the development of this dissertation.

I would like to thank my family for helping me financially study abroad in the University of Kent.

I would like to thank my friends who flew from France for my birthday.

I would also like to thank the friends I made at university to bear with me during those eight months.

Declaration

I hereby declare that this dissertation represents my own work which has been done after registration for the degree of Msc Cyber Security at Kent University in Canterbury, and has not been previously included in a thesis or dissertation submitted to this or any other institution for a degree, diploma or other qualifications.

Contents

1 Introduction 6
1.1 Background 6
1.2 Research Aims and Objectives 7
1.3 Scope and Limitations 8
1.4 Report Outline 8

2 Literature Review 9
2.1 A Review of MedKiosk: An Embodied Conversational Intelligence via Deep Learning[1] 9
2.2 Home Automation using IoT and a Chatbot using Natural Language Processing[2] 10
2.3 Implementation of a Chatbot System using AI and NLP[3] . . 11
2.4 SuperAgent: A Customer Service Chatbot for E-commerce Websites[4] 12

3 Design and Implementation 14
3.1 Design Methodology 15
3.2 Implementation 17

4 Testing and Evaluation 24
4.1 Testing environment 24
4.2 Database used 24
4.3 Agent Testing 25

5 Conclusion **28**

Chapter 1

Introduction

1.1 Background

A chatbot, also called conversational agent, is a program that chats with a user. Research on person-machine interface is influenced by the competition on Turing's test in 1950: giving the illusion that a program thinks a meaningful dialogue.

The Turing test is a proposed artificial intelligence test based on the ability of a machine to mimic human conversation. Described by Alan Turing in 1950 in his publication Computing Machinery and Intelligence, this test consists of putting a human in a blind verbal confrontation with a computer and another human. If the person initiating the conversations is not able to tell which of his or her interlocutors is a computer, the computer software can be considered to have passed the test.

A chatbot is generally composed of the same components a human would use to communicate: an input (eyes and/or ears of the bot), an algorithm (brain of the bot) and an output (mouth of the bot).

For the first step, a user formulates his request in natural language by writing through a command line interface or using his voice, in the case of voice the data has to be captured via a microphone and transformed to text using speech to text recognition (not covered in this thesis). The program then uses his “brain” to create a response, this part is composed of a database of words and sentences linked with multiple algorithms, to create his memories so it can search through his memories to respond in the best way possible. After finding the best response, the bot outputs directly on the screen the response, it can also output the response through speaker using text to speech transformation (not covered in this thesis).

On the one hand with all the information available on the internet, it is relatively easy to obtain information on a subject, but on the other hand this massive and scattered availability can make it difficult to obtain quality information, because anyone can write on the internet pretending to be a professional.

It is therefore becoming necessary to create platforms that bring together information on important topics in an organized way, while promoting the quality and integrity of the information.

Thanks to recent technological advances it is now possible to create conversational agents capable of mastering a subject and conversing about it with a human.

1.2 Research Aims and Objectives

This research aims to:

- Create an intelligent conversational agent that can engage with humans and provide them with accurate and detailed help.
- Consider that the agent must sound like a human to make its answers convincing and to make the user want to exchange more.
- Evaluate the suitability and the versatility of the agent.

Specifically, the objectives are to:

- Develop a conversational agent which implements an algorithm to learn from any given data and another one to find accurate data in its database to respond to the user.

1.3 Scope and Limitations

Although the possibility of topics can be infinite, we will direct the chatbot towards medicine. The goal is not to make it understand subjects that are too complex and multifaceted, but to bring together information on diseases and medications in an accessible way for everyone.

1.4 Report Outline

To achieve the research objectives, the thesis structure is as follows.

Chapter 2 gives an overview on some of existing literature about conversational agents, including medicine oriented agents.

Chapter 3 describes the design and implementation of Awesome-Bot

Chapter 4 provides the testing of the agent.

Chapter 5 conclusion.

Chapter 2

Literature Review

This chapter will be used to explore the literature on conversational agents, in order to understand existing projects and the techniques used.

2.1 A Review of MedKiosk: An Embodied Conversational Intelligence via Deep Learning[1]

This paper describes an interactive kiosk, defined as a self-service computer terminal integrated with specialized hardware and software designed to display information and perform a variety range of functions to a public exhibit. This research intends to escalate the productivity and efficiency in medical institutions by offering the capabilities to provide immediate reply as well as initiating a conversation, like conversing with the experienced customer service assistant. The proposed agent is build using natural language processing (NLP), machine learning and deep learning, as those technologies can ensure smarter decision making in providing accurate, reliable, and up-to-date information. This research aims to design and develop a framework for revolutionizing medical kiosk which incorporates an intelligent chatbot agent to increase the productivity and efficiency in hospital by providing the capabilities of answering routine and frequently asked questions besides permitting relevant searches as well as initiating a conversation. The medical kiosk would be placed in hospital to be used in real-time environment for data collections. This paper provides a clear overview of what could be a medicine oriented conversational agent, that could interact in medical centres to help users as a human could.

2.2 Home Automation using IoT and a Chatbot using Natural Language Processing[2]

This paper describes a web application and a chatbot working together to provide home automation, using Internet of things (IoT). The web application used to enable home automation has security features wich is an important concern when working with IoT. The architecture is composed of two parts, the first being the chatbot written in python and running on a Raspberry Pi. The chatbot uses Natural Language processing (NLP) to treat user input, then it communicates user's request to the web application that controls devices connected to the same LAN, and can modify states of lights, fans, door lock and more. It can even notify intruder alert using a PIR sensor on a Raspberry Pi, connected to a smtp server to send an E-mail or can send a SMS to home owner. The following techniques are applied to the text when user communicates with the chatbot:

- Tokenization: sentence split into words using Natural Language Toolkit (NLTK)

- Conversion into lowercase: the bot doesn't need to make a difference between uppercase and lowercase so this adds uniformity and make processing easier.

- Removal of stop-words: the list of words previously made is reviewed to remove useless words using a predefined list of stop-words, now it only remains meaning-full words.

- Defining keyword and action lists: the chatbot knows a list of keywords, it uses this list to understand what is the request of the user, one list for action (increase, decrease, switch, open, close, ...) and a list for the objects (light, fan, door, temperature, ...).

- Actuation: now the relevant action is taken, if keyword is light and action is decrease, the bot can send a request to the web application to control the desired object.

This paper is very use-full because it describes human to machine interaction using NLP and Python for simple conversation, a simple request for a simple response. The same process could be used for medical assistance, requesting information about a medicine or a condition, the bot could respond with desired data.

2.3 Implementation of a Chatbot System using AI and NLP[3]

This paper describes a conversational agent for college, the major features are: college admission related queries, viewing user profiles and retrieve attendance and grade/pointers, college students could get information about examinations to be held, college students could getch particulars about placement activities. The proposed system is split in five parts:

- Context identification: pre-processing is applied to the input text to standardize the input as per the system's requirement. Based on the keywords used in the text, appropriate context is recognized.
- Personal Query Response System: Upon receiving personal queries like CGPA, attendance, etc., the authenticity of the user is checked through user-id and password.The input text is processed to extract keywords, then the information required by the user is understood and the information is provided from the database.
- AIML Response System: If the user is trying to make a normal conversation with the bot, the input is mapped to an appropriate pattern in Artificial Intelligence Modeling Language (AIML) files. If the pattern is not available in AIML files, a random response is sent suggesting "Invalid Input".

- Query Analysis and Response System: An algorithm to check sentence similarity (NLP) is applied to the modified input to check its similarity with the questions of a predefined question-set, whose answers are available. If a sentence is retrieved with confidence¿ 0.5, the bot returns the answer of that question as the response.

- Context Reset: Once the user is satisfied with the response of bot and does not wish to chat further, he/she has the option to log out of the system or simply exit. Once the user exits the system, all input parameters are automatically reset.

This paper provides an implementation of a chatbot that can interact with college students and employees, the chatbot provides fast and efficient search for answers to the queries and gets the relevant links to their question.

2.4 SuperAgent: A Customer Service Chatbot for E-commerce Websites[4]

This paper describes a customer service chatbot that leverages large-scale and publicly available e-commerce data. SuperAgent is a web extension that crawls the html information from the webpage. When users visit the page, SuperAgent will be notified since the add-on extension is associated with each webpage. When the user queries the chatbot, different engines are processed in parallel. If one of the answers from the first three engines has high confidence, the chatbot returns with the answer as the response. Otherwise, the chit-chat engine will generate a reply from the predefined permitted response sets. The three engines are defined as such:

- Fact QA: The fact QA engine is designed for answering questions regarding the facts of the product, e.g Q: ”What is the CPU ?” A: ”The processor is 3GHz Intel Core i5”.

- FAQ Search for Customer QA Pairs: given a set of QA pairs and a customer’s question, the engine finds the most similar question and return the corresponding answer as the reply.

- Opinion-Oriented Text QA for Reviews: Customer reviews provide rich information for different aspects of the product from users' perspective. Review text is split into sentences and an opinion module is used to extract the ascpects and corresponding opinions.

The chit-chat engine is mainly designed to reply to greeting queries such as "hello" and "thank you", as well as queries that cannot be answered by the previous three engines, such as "you are so cute". Not only this paper provides a good implementation of a customer service chatbot for e-commerce websites, it provides a very good example of what the architecture of a chatbot should be, from the web crawler that retrieves data from websites to the question-answer engine and the chit-chat engine. The architecture of this chatbot fits very well with Awesome-Bot.

Chapter 3

Design and Implementation

Software design is the process of accomplishing goals using a set of primitive components and subjects to constraints. It is composed of all the activity involved in conceptualizing, framing, implementing, and modifying complex systems to follow requirements specification before programming the software itself.

Software implementation describes concretely how the program is made, the technologies used and the process of achieving the result. The creation of a working software is a combination of coding, verification and testing.

3.1 Design Methodology

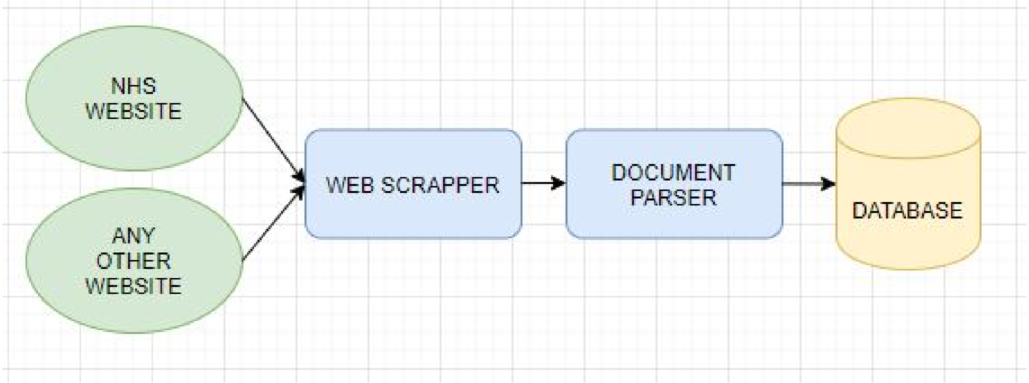


Figure 1: Data Extraction Program (to create and update the database)

The Data Extraction Program is a Python3 script that creates the database used by Awesome-Bot, it is composed of a web scraper, a document parser, and a database.

The web scraper retrieves data from a website, in the case of the NHS's website, it uses the main medicine page (or conditions) to loop through every medicine available, then it save the content with its title (e.g. Coronavirus).

The document parser parses every document saved by the web scraper; it is adapted for each desired website. Its goal is to create a clean corpus, composed of blocks of texts linked to titles. After this process, the corpuses are saved in the database, in the desired section.

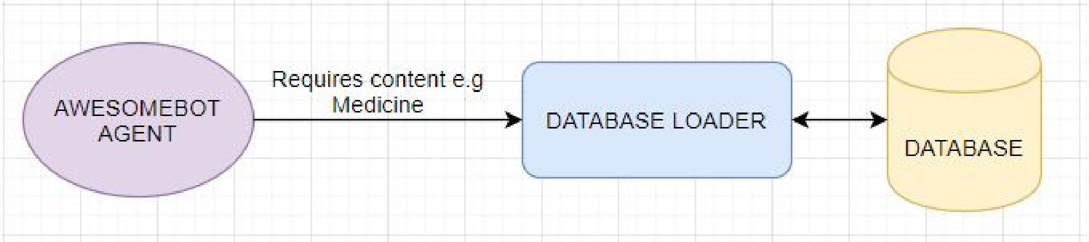


Figure 2: Initial Loading

Because Awesome-Bot could be used for subjects others than health, it has to be specified at the start of the program which subject it should load.

This can be done using the database loader: e.g. database.load("Medicines") and/or database.load("Conditions")

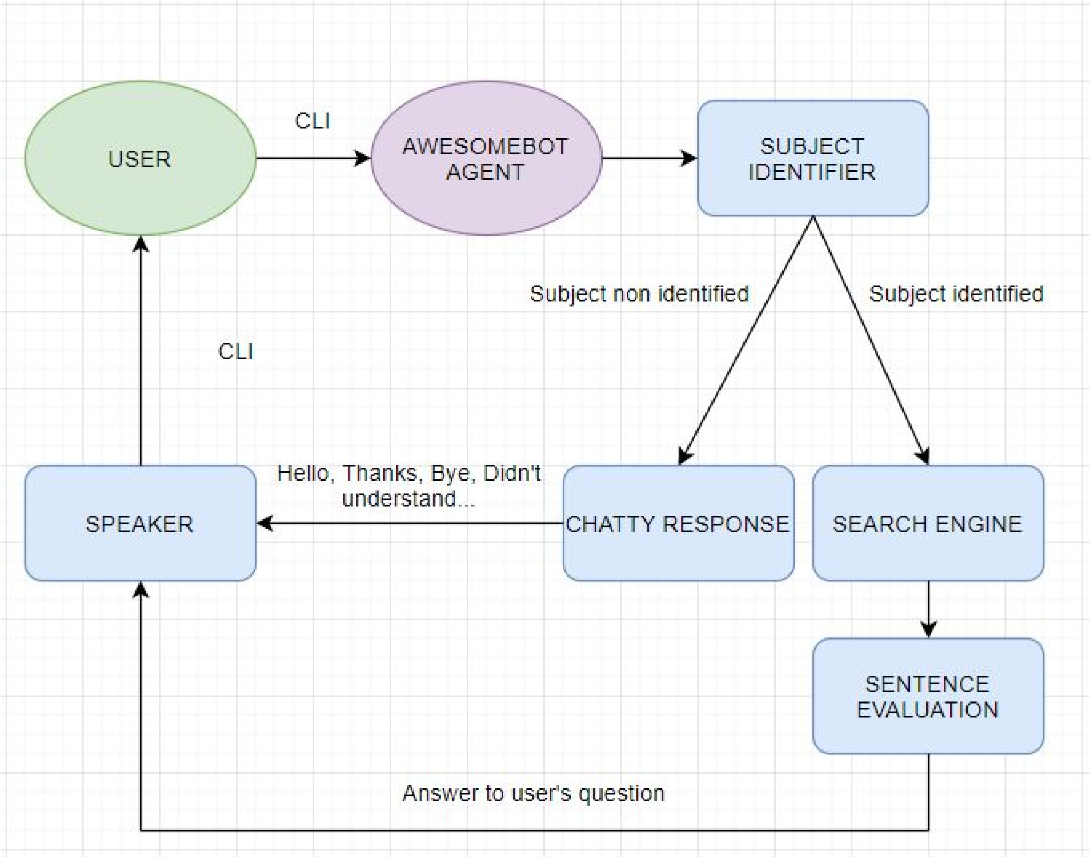


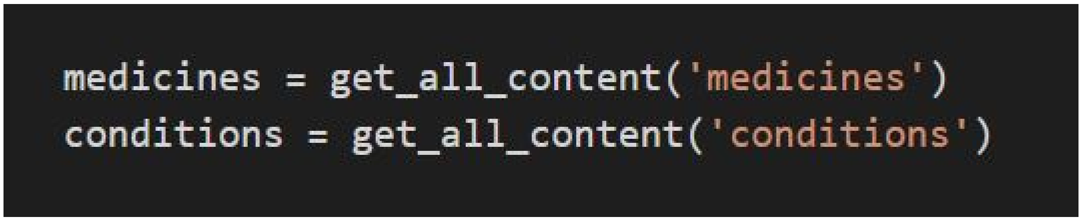
Figure 3: Awesomebot's Process Flow

Awesome-Bot is coded in Python3, its process flow is described as follows: The user and Awesome-Bot interact through a Command Line Interface (CLI), when the user sends a message to Awesome-Bot, the message is sanitized (transform to lowercase letters, delete useless punctuation, delete useless whitespaces) the Subject Identifier searches if the message contains a key word of a subject it has loaded previously, if yes it uses his Search Engine to find the subject that matches the most with user's key words, in order to send back the best response using the Sentence Evaluation module, if not it uses its Chatty Response module to keep the discussion going.

3.2 Implementation

As described in Figure 1, a program that parses NHS's website must be used to create the database of Awesome-Bot, it'll save data from every listed condition and medicine on the NHS's website.

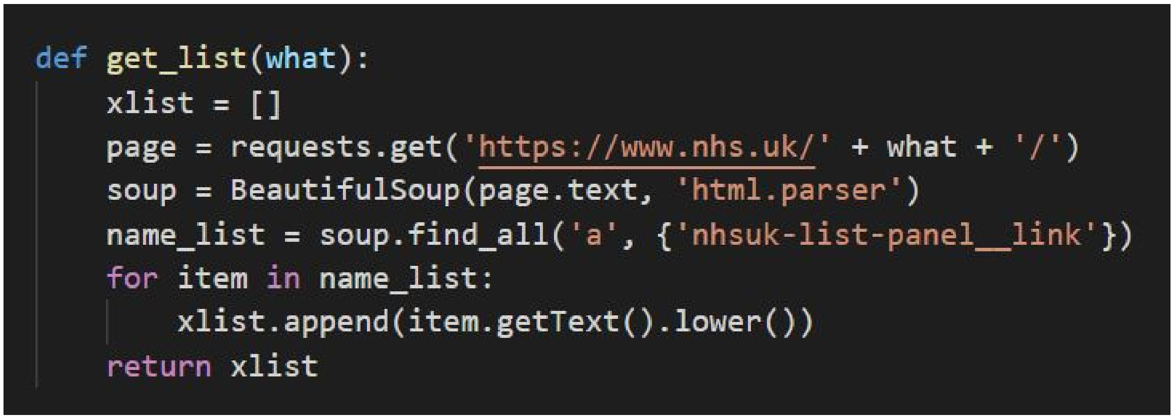
First we get the content of those pages with the function get_all_content(string) with the first parameter being a string that specifies what to extract, 'medicines' or 'conditions'.



```
medicines = get_all_content('medicines')
conditions = get_all_content('conditions')
```

Figure 4: get_all_content(string) function call

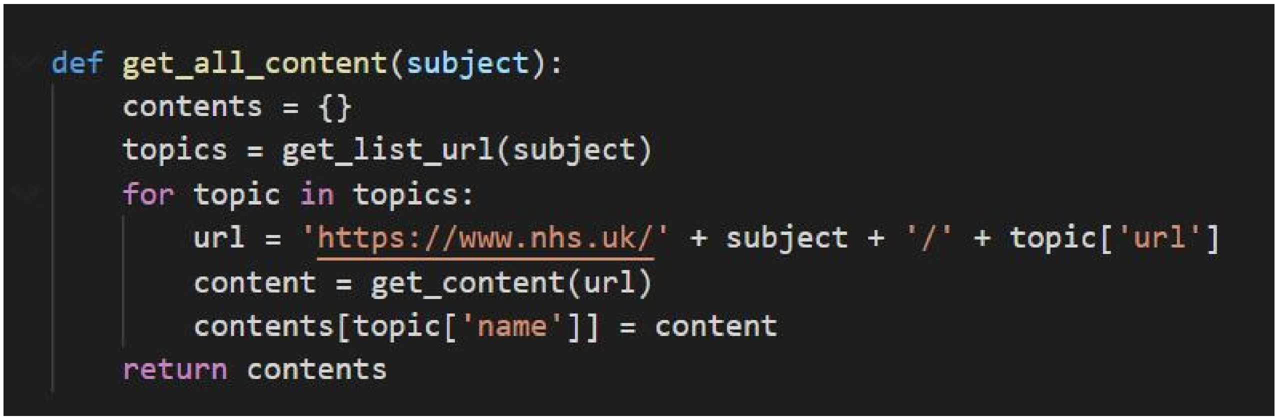
The get_all_content(string) function itself will retrieve a list of every topic with the function get_list(string) by specifying the subject i.e 'medicines'. Using the BeautifulSoup module in Python 3, we can retrieve every topic listed on the subject's page.



```python
def get_list(what):
    xlist = []
    page = requests.get('https://www.nhs.uk/' + what + '/')
    soup = BeautifulSoup(page.text, 'html.parser')
    name_list = soup.find_all('a', {'nhsuk-list-panel__link'})
    for item in name_list:
        xlist.append(item.getText().lower())
    return xlist
```

Figure 5: get_list(string) function

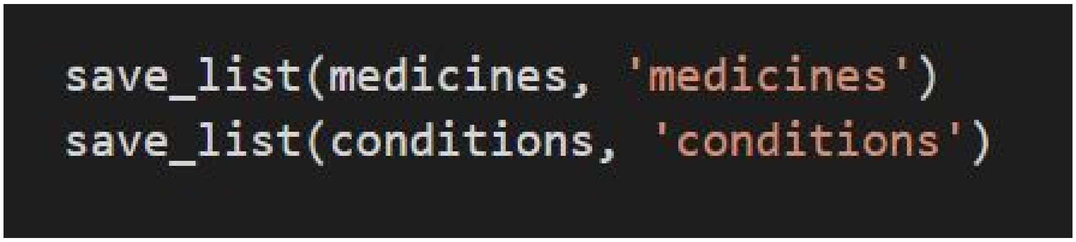
Then for each topic in the topic list, we can create the desired url to retrieve data. The subject is 'medicines' or 'conditions' and the topic is the medicine or condition itself, as described in figure 6.



```python
def get_all_content(subject):
    contents = {}
    topics = get_list_url(subject)
    for topic in topics:
        url = 'https://www.nhs.uk/' + subject + '/' + topic['url']
        content = get_content(url)
        contents[topic['name']] = content
    return contents
```

Figure 6: get_all_content(string) function

Using the created url for each topic, the function get_content(string) is used to retrieve the content of the page, giving the url as first parameter.

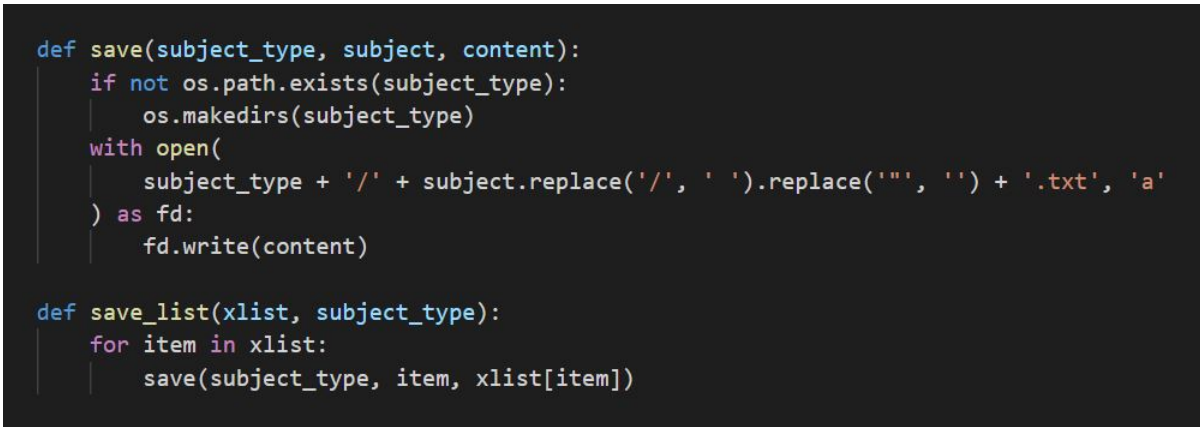


```
save_list(medicines, 'medicines')
save_list(conditions, 'conditions')
```

Figure 7: save_list(list, string) function call

‘

save_list(list, string) takes two parameters, the first one is the list of every medicine or condition, and the second one is the label of the list. This function is used to store all the data retrieved by the get_all_content(string) function.

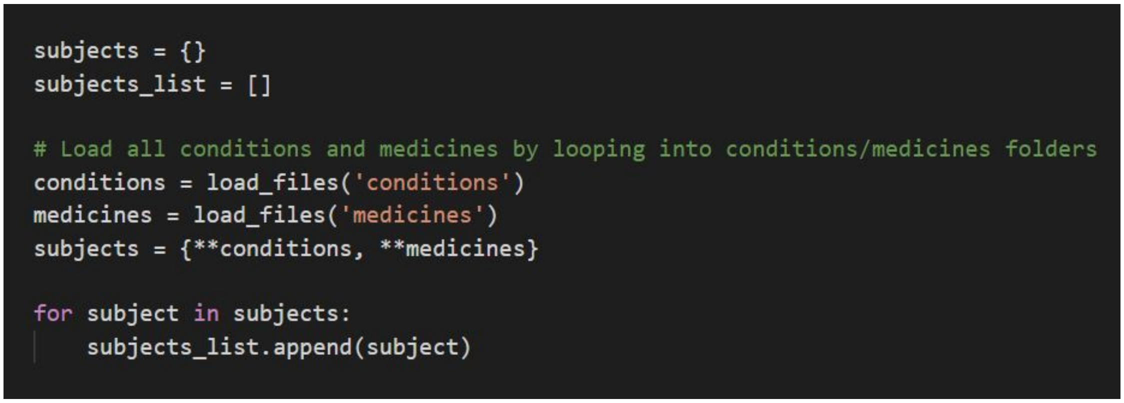


```
def save(subject_type, subject, content):
    if not os.path.exists(subject_type):
        os.makedirs(subject_type)
    with open(
        subject_type + '/' + subject.replace('/', ' ').replace('"', '') + '.txt', 'a'
    ) as fd:
        fd.write(content)

def save_list(xlist, subject_type):
    for item in xlist:
        save(subject_type, item, xlist[item])
```

Figure 8: save_list(list, string) function and save(string, string, string) function

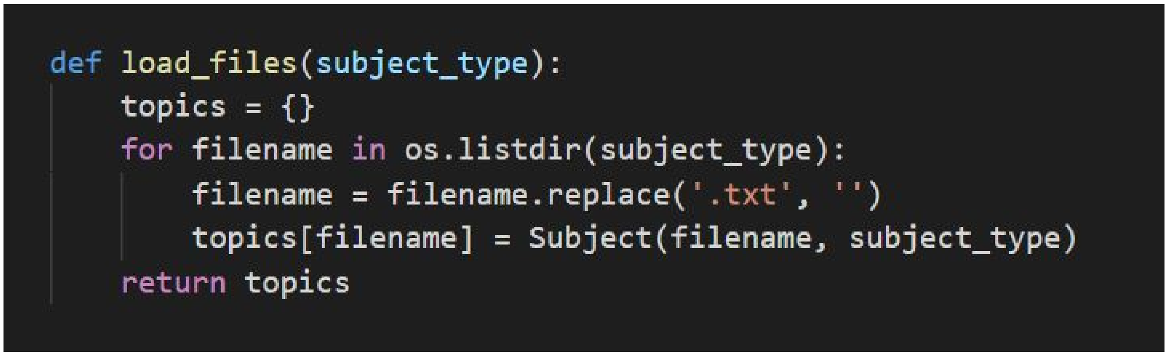
For each item in the list, save_list(list, string) will call save(string, string, string), the first parameter being the subject type, the second one the subject itself or the topic, and the last one is the content or the data about this topic. The save(string, string, string) function creates a folder for the subject type and writes a file named by the subject name, containing the data.

Figure 2 describes the initial loading of the data at Awesome-Bot's start.



```python
subjects = {}
subjects_list = []

# Load all conditions and medicines by looping into conditions/medicines folders
conditions = load_files('conditions')
medicines = load_files('medicines')
subjects = {**conditions, **medicines}

for subject in subjects:
    subjects_list.append(subject)
```

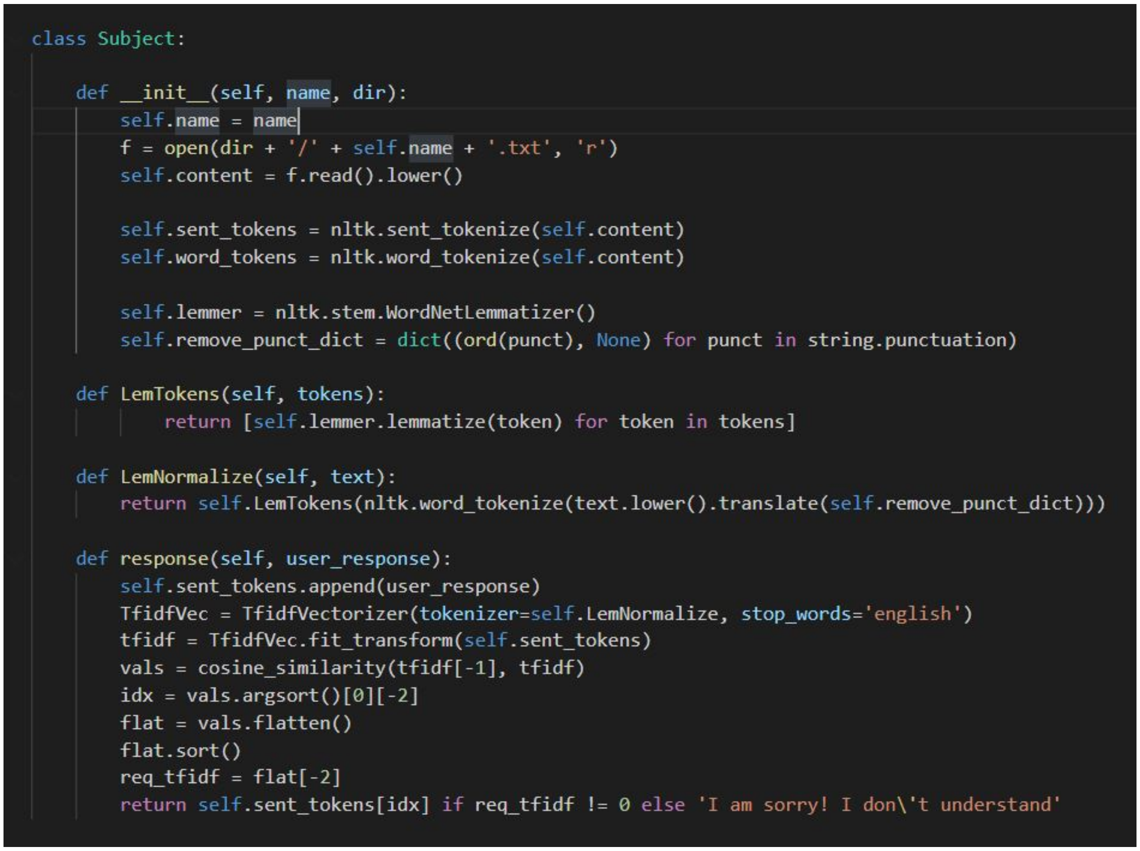
Figure 9: loading data of Awesome-Bot



```python
def load_files(subject_type):
    topics = {}
    for filename in os.listdir(subject_type):
        filename = filename.replace('.txt', '')
        topics[filename] = Subject(filename, subject_type)
    return topics
```

Figure 10: load_files(string) function

The load_files(string) function is used to load the database, it creates a list of all topics based on the files saved previously. This function introduces the Subject(string, string) class, this class is used to setup the data for every topic.

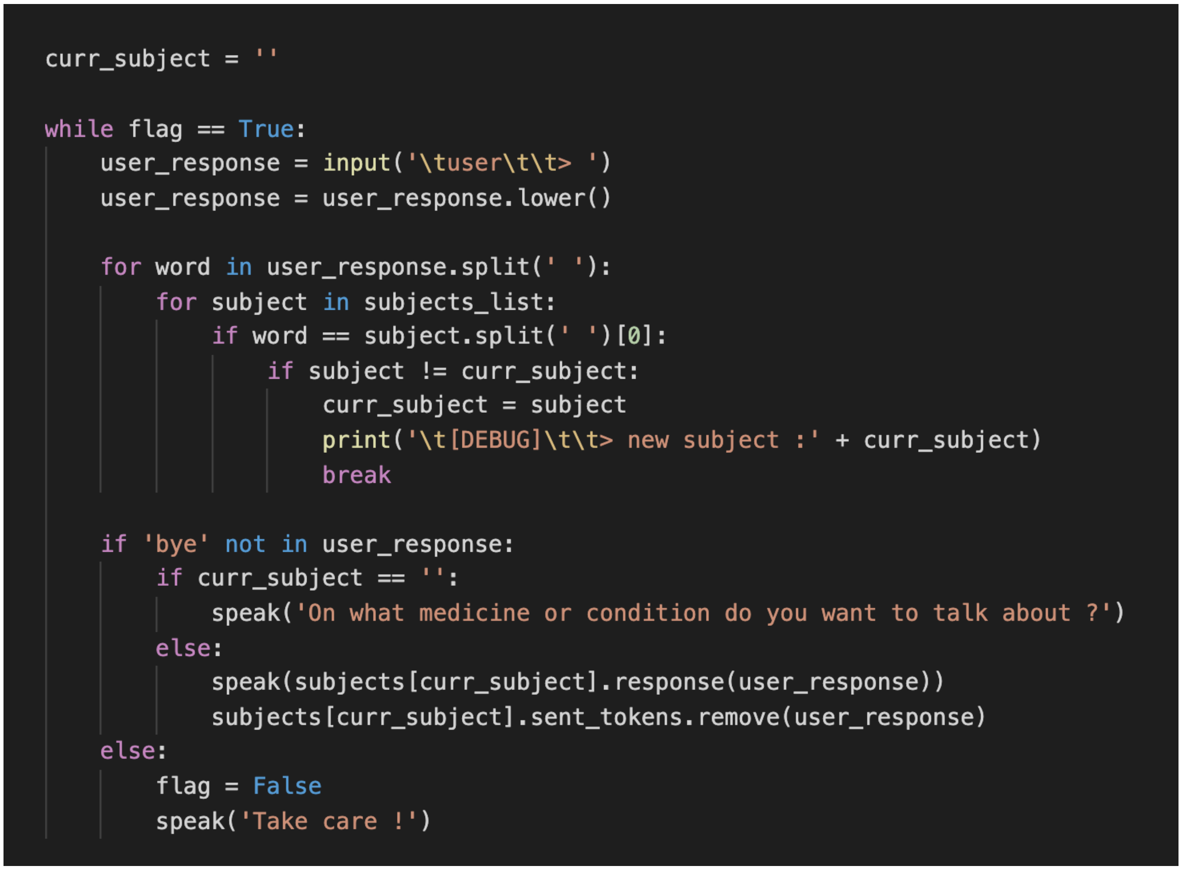


```python
class Subject:

    def __init__(self, name, dir):
        self.name = name
        f = open(dir + '/' + self.name + '.txt', 'r')
        self.content = f.read().lower()

        self.sent_tokens = nltk.sent_tokenize(self.content)
        self.word_tokens = nltk.word_tokenize(self.content)

        self.lemmer = nltk.stem.WordNetLemmatizer()
        self.remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)

    def LemTokens(self, tokens):
            return [self.lemmer.lemmatize(token) for token in tokens]

    def LemNormalize(self, text):
        return self.LemTokens(nltk.word_tokenize(text.lower().translate(self.remove_punct_dict)))

    def response(self, user_response):
        self.sent_tokens.append(user_response)
        TfidfVec = TfidfVectorizer(tokenizer=self.LemNormalize, stop_words='english')
        tfidf = TfidfVec.fit_transform(self.sent_tokens)
        vals = cosine_similarity(tfidf[-1], tfidf)
        idx = vals.argsort()[0][-2]
        flat = vals.flatten()
        flat.sort()
        req_tfidf = flat[-2]
        return self.sent_tokens[idx] if req_tfidf != 0 else 'I am sorry! I don\'t understand'
```

Figure 11: Subject(string, string) class

This class takes two parameters in its constructor, the first one is the name of the topic i.e ”Corona”, the second parameter is the folder where the data is. The content of the file is read and stored in the self.content variable. Using the module nltk (Natural Language Toolkit) in Python 3, tokens of the content are saved with the function sent_tokenize(string), and word tokens are saved with the function word_tokenize(string). Those variables are used to store data in our class. The response function is crucial, it creates a matrix with the data of the topic and uses cosine similarity to sort the most adequate response to the given user_response. The data matrix is created thanks to the feature_extraction of sklearn module, the cosine_similarity function is also from the sklearn module.

Awesome-Bot’s process flow is composed of three main parts, as described in Figure 3. The first part is the interface between Awesome-Bot and the user. Awesome-Bot uses a command line interface (CLI) even though vocal command could be implemented in the future. Awesome-Bot can be run in a terminal using Python 3: ”python3 awesome-bot.py”, once started, the user can communicate with the agent through CLI and pressing the enter command to send a sentence.



```
curr_subject = ''

while flag == True:
    user_response = input('\tuser\t\t> ')
    user_response = user_response.lower()

    for word in user_response.split(' '):
        for subject in subjects_list:
            if word == subject.split(' ')[0]:
                if subject != curr_subject:
                    curr_subject = subject
                    print('\t[DEBUG]\t\t> new subject :' + curr_subject)
                    break

    if 'bye' not in user_response:
        if curr_subject == '':
            speak('On what medicine or condition do you want to talk about ?')
        else:
            speak(subjects[curr_subject].response(user_response))
            subjects[curr_subject].sent_tokens.remove(user_response)
    else:
        flag = False
        speak('Take care !')
```

Figure 12: Awesome-Bot’s process flow function

Figure 12 is the main loop of Awesome-Bot, it first waits for an input from the user, then it uses the subject identifier to understand on what topic the user wants to talk. If a subject is detected, the agent will respond using the response function of the desired topic within the Subject class. If the user's input contains the word 'bye', the agent will close.

Chapter 4

Testing and Evaluation

4.1 Testing environment

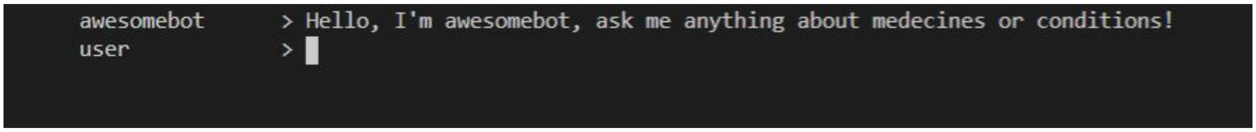
Python version is Python 3.8.2 for this testing part. There is no particular reason for the choice of this version it was simply the latest version at the start of the project.

All testing was executed on a Windows 10 Education version 1909 64-bits operating system, with a AMD Ryzen 7 3700x 8-Core 3.59 GHz processor, 16.0 Gb of DDR4 RAM 3200 MHz, no GPU was used, the storage (used to load the database at program's startup) is a Samsung SSD 970 EVO Plus of 500Gb.

4.2 Database used

The database used was created by the function "get_all_content" as shown in Figure 4, this function retrieves data from NHS's website. Both medicines and conditions were loaded for a sum of 1385 files (1174 conditions + 211 medicines). Conditions and medicines are loaded at startup of program using the function "load_files" as shown in Figure 9, by passing the name of the desired content as first parameter i.e. "medicines".

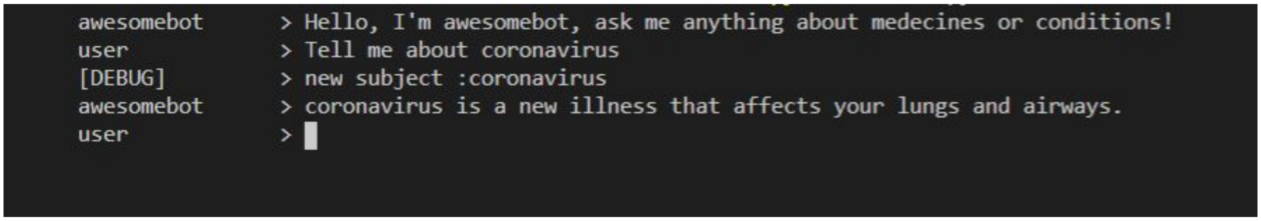
4.3 Agent Testing



```
awesomebot      > Hello, I'm awesomebot, ask me anything about medecines or conditions!
user            > █
```

Figure 13: "Hello, I'm awesomebot"

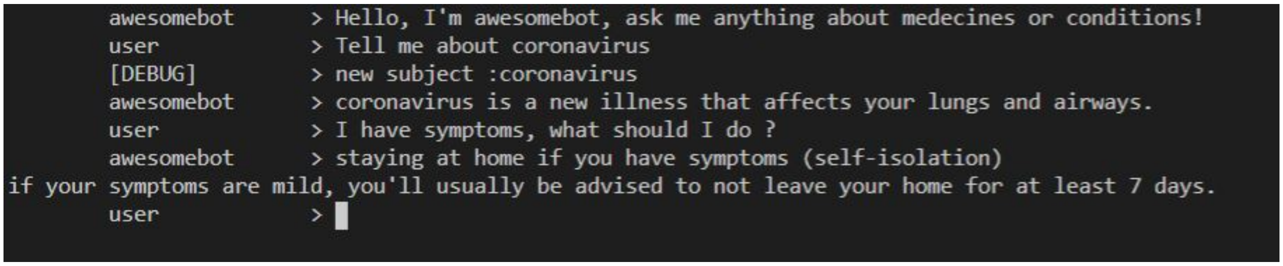
At every start the agent introduces itself and show what subject it can talk about.



```
awesomebot      > Hello, I'm awesomebot, ask me anything about medecines or conditions!
user            > Tell me about coronavirus
[DEBUG]         > new subject :coronavirus
awesomebot      > coronavirus is a new illness that affects your lungs and airways.
user            > █
```

Figure 14: "Tell me about coronavirus"

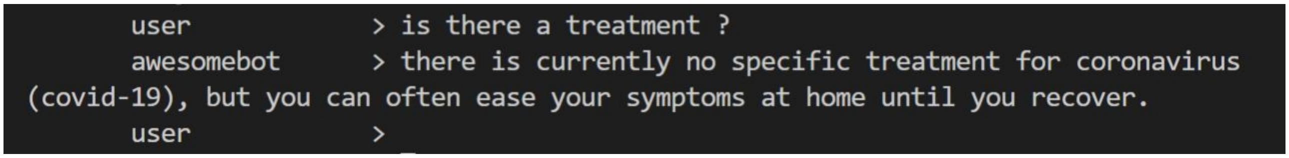
Here the user asks awesome-bot about the coronavirus, with the [DEBUG] label we can see the agent understood a new subject, here the coronavirus. Then the agent outputs a description of the subject.



```
        awesomebot      > Hello, I'm awesomebot, ask me anything about medecines or conditions!
        user            > Tell me about coronavirus
        [DEBUG]         > new subject :coronavirus
        awesomebot      > coronavirus is a new illness that affects your lungs and airways.
        user            > I have symptoms, what should I do ?
        awesomebot      > staying at home if you have symptoms (self-isolation)
if your symptoms are mild, you'll usually be advised to not leave your home for at least 7 days.
        user            > █
```

Figure 15: "I have symptoms..."

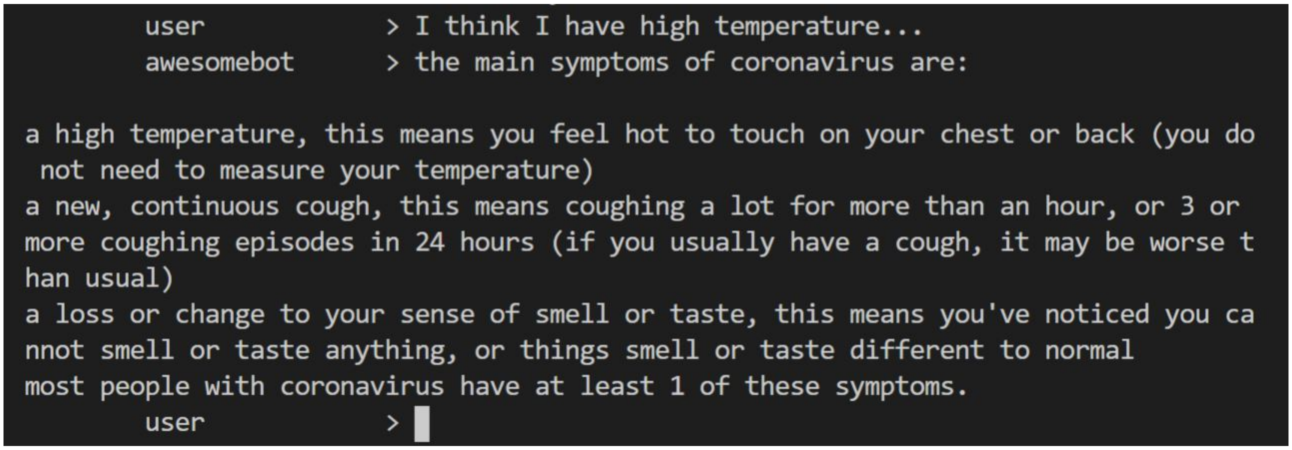
The user then tells the agent he might have symptoms, the agent responds with the most accurate block of text by looking in its database and comparing user's input with the corresponding data.



```
        user            > is there a treatment ?
        awesomebot      > there is currently no specific treatment for coronavirus
(covid-19), but you can often ease your symptoms at home until you recover.
        user            >
```

Figure 16: "is there a treatment ?"

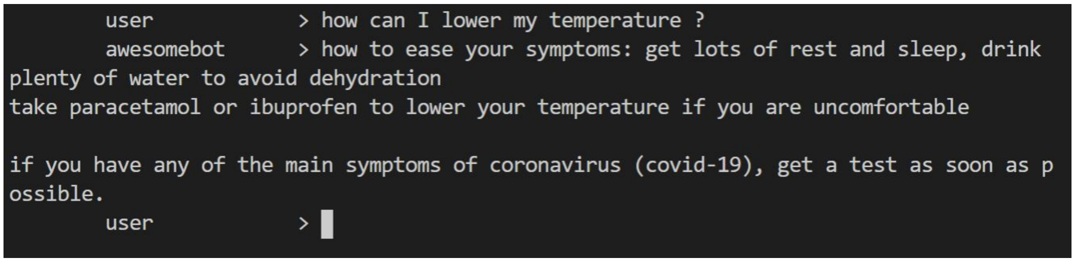
Following the precedent conversation, the user asks the agent if there is any treatment to the coronavirus, the agent searches in its database of coronavirus for a sentence matching with strongs words like 'treatment' and replies that there is currently no specific treatment.



```
        user            > I think I have high temperature...
        awesomebot      > the main symptoms of coronavirus are:

a high temperature, this means you feel hot to touch on your chest or back (you do
 not need to measure your temperature)
a new, continuous cough, this means coughing a lot for more than an hour, or 3 or
more coughing episodes in 24 hours (if you usually have a cough, it may be worse t
han usual)
a loss or change to your sense of smell or taste, this means you've noticed you ca
nnot smell or taste anything, or things smell or taste different to normal
most people with coronavirus have at least 1 of these symptoms.
        user            >
```

Figure 17: "I think I have high temperature..."

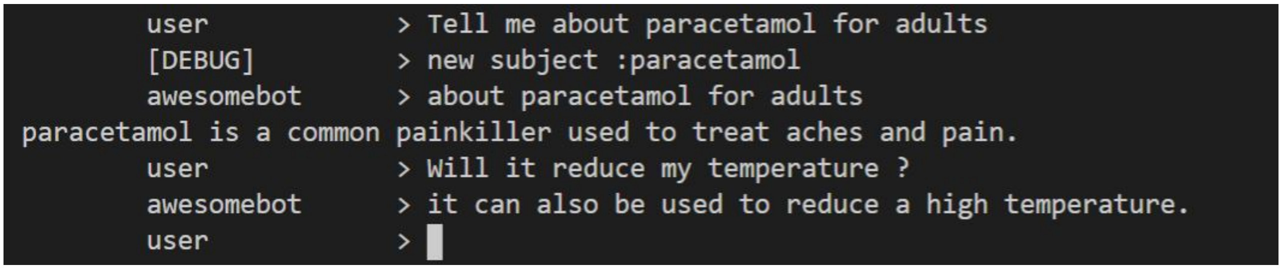
The user is worried that his high temperature might be related to a coronavirus symptoms so he tells the agent he has high temperature, the response of the bot could have been a good response to the precedent input because its response has 'symptoms' in it, but this response is best suited here because it is partially about the 'high temperature' keyword.



```
        user            > how can I lower my temperature ?
        awesomebot      > how to ease your symptoms: get lots of rest and sleep, drink
plenty of water to avoid dehydration
take paracetamol or ibuprofen to lower your temperature if you are uncomfortable

if you have any of the main symptoms of coronavirus (covid-19), get a test as soon as p
ossible.
        user            > █
```

Figure 18: "how can I lower my temperature ?"

Concerned about his high temperature, the user asks the agent how to lower it. A sentence about the ease of symptoms is found and the agent tells the user to take paracetamol or ibuprofen to lower his temperature, which a really good response because it is about 'ease of symptoms' and 'lower temperature'.



```
        user            > Tell me about paracetamol for adults
        [DEBUG]         > new subject :paracetamol
        awesomebot      > about paracetamol for adults
paracetamol is a common painkiller used to treat aches and pain.
        user            > Will it reduce my temperature ?
        awesomebot      > it can also be used to reduce a high temperature.
        user            > █
```

Figure 19: "Paracetamol"

After the agent spoke about a medicine that could ease symptoms of coronavirus, the user asks about this medicine: paracetamol. The agent recognize a new subject and tells the user a simple sentence that describes what is paracetamol. The user then asks if it will reduce his temperature to which the agent replies that it can also be used to reduce high temperature.

Here we have a good example of subject change, but also a very good example of sentence matching.

Chapter 5

Conclusion

This part of the essay highlights the main conclusions and assesses the validity of the work done.

Analysis of the results leads to the conclusion of these points:

- The chapter 4 proves that we have succeeded in creating a conversational agent that can engage with humans and provide them with accurate and detailed helps.
- The agent sounds like a human being, its responses were created by humans on the NHS's website, it makes the user want to exchange more with the agent, the flow of the conversation is fluent and smooth.
- The structure of the agent implements successfully an algorithm that makes it learn from any given data (i.e. medicine) and implements an algorithm that makes it able to find accurate data in its database to respond with the best matching response to the user.

We can conclude that all the objectives are completed, the agent learns from data, can match user's input to accurate block of text and sounds like a human.

Bibliography

[1] Pui Huang Leong, Ong Sing Goh, and Yogan Jaya Kumar. *MedKiosk: An Embodied Conversational Intelligence via Deep Learning*. Malaysia, 2017.

[2] Cyril Joe Baby, Faizan Ayyub Khan, Swathi J. N. *Home Automation using IoT and a Chatbot using Natural Language Processing*. VIT University, Vellore, India.

[3] Tarun Lalwani, Shashank Bhalotia, Ashish Pal, Shreya Bisen, Vasundhara Rathod. *Implementation of a Chatbot System using AI and NLP*. Shri Ramdeobaba College of Engineering and Management, India.

[4] Lei Cui, Shaohan Huang, Furu Wei, Chuanqi Tan, Chaoqun Duan, and Ming Zhou. *SuperAgent: A Customer Service Chatbot for E-commerce Websites*. Microsoft Research Asia