Semester Project

Emotion dialog management system through a Chatbot transaction

Author:
Oussama Abouzaid

Supervisor:
Dr. Pearl Pu Faltings
Yubo Xie

June 6, 2019

Contents

Abstract

When developing chatbots, the customer experience should always be the important thing. That is why it is important to ensure a seamless collaboration between people and technology. In this project, we introduce a transactional chatbot that helps a user order beer, while keeping the conversation as human as possible, by introducing emotions and small talk.

1 Introduction

A *chatbot* is an artificial intelligence software that can simulate a conversation (or a chat) with a user in natural language through messaging applications, websites, mobile apps. etc... It has nowadays a lot of application domains in daily life, and it helps the user accomplish specific tasks. For example, Apple's Siri and Amazon's Alexa are great examples of AI-based chatbots. They can answer questions, make recommendations, set an alarm or a reminder, and much more. But most of time, chatbots (including Siri and Alexa) don't detect emotions in the user's utterances.

In this project, we build a transactional chatbot that manages a dialog and supports emotions. The bot walks the user through a beer order, and uses slot-filling to extract all the required information, such as the number of beers, the type and the size. Eventually, the chatbot concludes the transaction when the user pays using a valid credit card, after it has been validated. Besides the transaction, the other focus of this project is showing the ability of the bot the handle some emotional utterances of the user, which can include sadness, anger, or even respond to small talk.

Firstly, present the components used to build the bot as well as its architecture. Secondly, we will explain the methodology behind building the emotional part of the bot. Thirdly, we will go through the external third-party APIs used by the bot to complete some of its tasks. And lastly, we will show a demonstration of a full transaction, include some emotional interactions between the bot and the user.

2 DialogFlow

The main part of the chatbot has been developed on **Dialogflow**, which is a Google-owned developer of human–computer interaction technologies based on natural language conversations. One of its most amazing features is its strong capacity to expand its learning and understanding of the user's intents, after a few training sentences only.
Let us explain more in detail the components and features used to build this chatbot.

2.1 Agent

An **agent** (chatbot) is responsible for classifying the user utterances into intents. These intents can be seen as states that are triggered in such a way to maximize the likelihood of the user matching his actual intention, based on previous training phrases fed to the agent.

2.2 Intents

The agent is trained to recognize many intents. From the transactional point of view, it is capable of welcoming back the user, asking for beer order (type and size), giving recommendations about beer, and finally and confirm the transaction by verifying the validity of the credit card entered, and hence completing the transaction. Besides this, the agent is able to tell jokes, comfort the user when they are sad or angry, talk about the weather, handle some small talk, and also crawl Wikipedia for additional information for extended knowledge. Intents of the agent are shown in Figure 1.

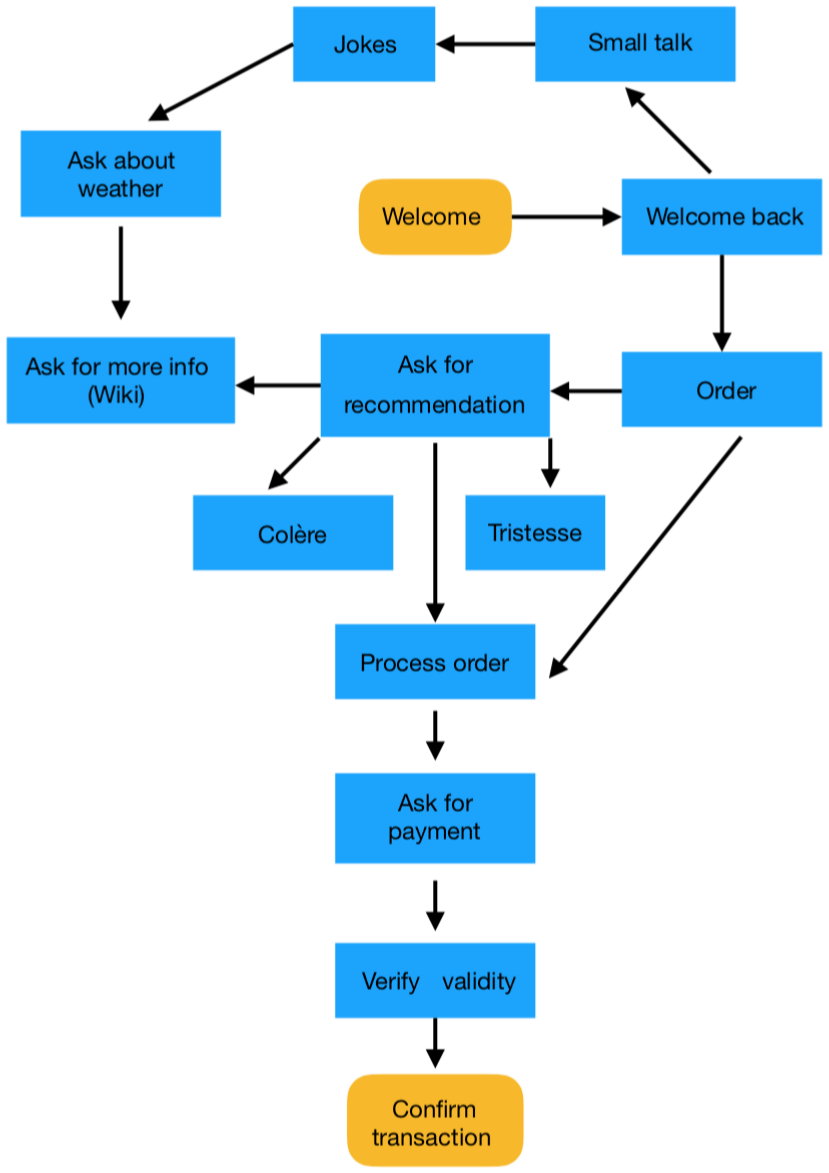
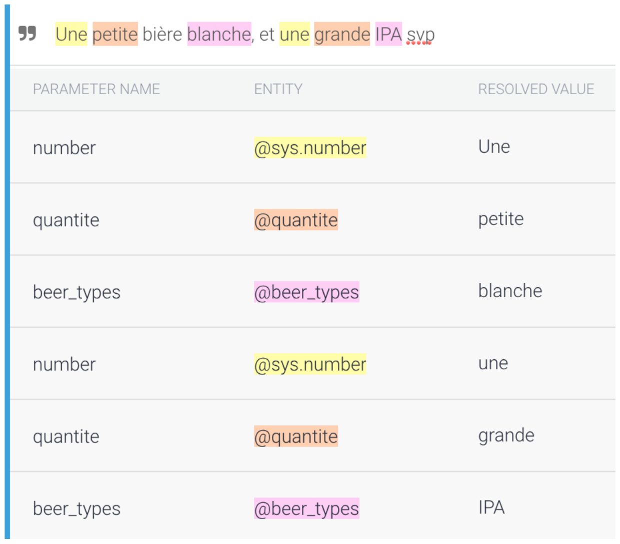


Figure 1: Diagram showing the agent's intents, with the possible transitions

2.3 Entities

While intents allow the agent to understand the motivation behind a particular user input, entities are used to extract specific pieces of information that the user mentions. This is known as **slot-filling.**

Example: In the *Order* intent, when the user is ordering a beer, 3 different entities are extracted: the number of beers, the types of beers (blond, white, IPA, etc.) and the quantity (large or small).



Une petite bière blanche, et une grande IPA svp

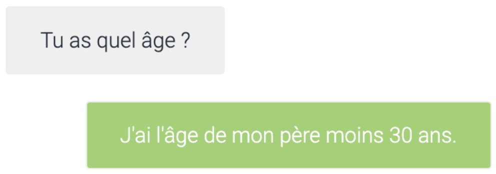
PARAMETER NAME	ENTITY	RESOLVED VALUE
number	@sys.number	Une
quantite	@quantite	petite
beer_types	@beer_types	blanche
number	@sys.number	une
quantite	@quantite	grande
beer_types	@beer_types	IPA

Figure 2: Entities extraction by the agent

2.4 Response

Each intent defines a response that is returned to the user. There are two primary ways a response is returns to the user:

- A pre-defined static response: set of pre-defined sentences for each intent, usually helpful to handle part of small talk or static information that does not include any emotions.
- A response generated from a webhook: when the response to a user utterance needs to be dynamic (e.g. asking about the weather), the user can write customized code called **fulfillment**, that sends HTTP

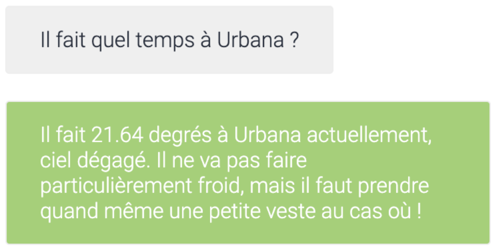


Tu as quel âge ?

J'ai l'âge de mon père moins 30 ans.

Figure 3: Example of a pre-defined static response

requests from Dialogflow external APIs, once the intent has been determined by the agent, and the entities extracted (if any). The response is typically in JSON format, which is manually parsed to return a proper response to the user. Figure 4 shows an example of a response from a webhook, and Figure5 shows the raw JSON response.



Il fait quel temps à Urbana ?

Il fait 21.64 degrés à Urbana actuellement, ciel dégagé. Il ne va pas faire particulièrement froid, mais il faut prendre quand même une petite veste au cas où !

Figure 4: Webhook response: Weather at Urbana, IL

```
 {'coord': {'lon': -88.21, 'lat': 40.11}, 'weather': [{'id': 800, 'main': 'Clear', 'description':
 'ciel dégagé', 'icon': '01d'}], 'base': 'stations', 'main': {'temp': 21.64, 'pressure': 1014, 'h
umidity': 56, 'temp_min': 20, 'temp_max': 23.33}, 'visibility': 16093, 'wind': {'speed': 5.7, 'de
g': 360}, 'clouds': {'all': 1}, 'dt': 1559493454, 'sys': {'type': 1, 'id': 3657, 'message': 0.006
1, 'country': 'US', 'sunrise': 1559471137, 'sunset': 1559524581}, 'timezone': -18000, 'id': 49145
70, 'name': 'Urbana', 'cod': 200}
```

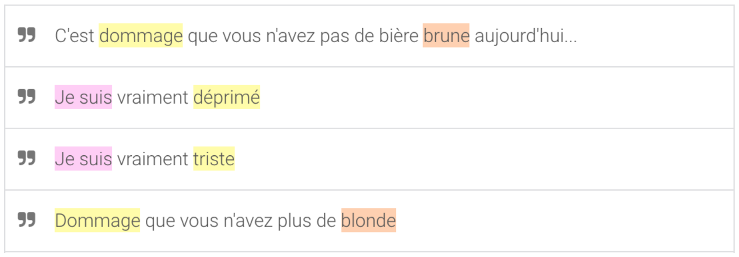
Figure 5: Raw JSON response from the webhook [1]

3 Emotion dialog

As stated earlier in the introduction, the main focus of building this chatbot is to go beyond achieving a single transaction with the user. That is, to further be able to recognize some emotions (and reply accordingly), and conduct small talk. In this section we will explain how the emotion part of the bot was designed, with some illustrative examples.

3.1 Training phrases

In order to train the bot to recognize emotions such as **sadness** and **anger**, some expressive training phrases have been fed to the bot. These phrases are chosen as the most common and used in daily life, in an informal way.



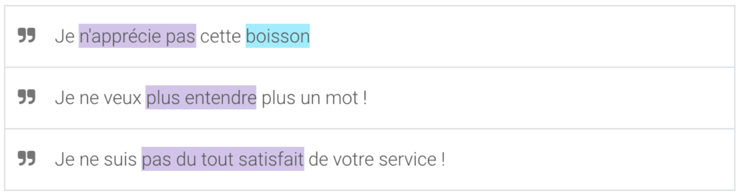
C'est dommage que vous n'avez pas de bière brune aujourd'hui...

Je suis vraiment déprimé

Je suis vraiment triste

Dommage que vous n'avez plus de blonde

Figure 6: Example of training phrases for sadness



Je n'apprécie pas cette boisson

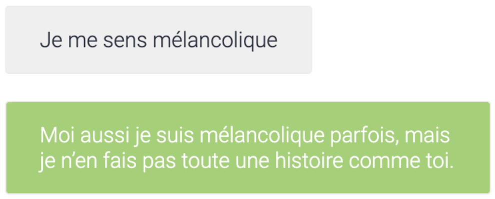
Je ne veux plus entendre plus un mot !

Je ne suis pas du tout satisfait de votre service !

Figure 7: Example of training phrases for anger

An important point to highlight here is that the words expressing the sadness (déprimé, triste, dommage) or anger (pas du tout satisfait, plus entendre) are defined in two separate entities called "tristesse" and "colère", respectively. They can be thought of as lexical fields of words. A powerful feature in Dialogflow allows for automatic expansion of this lexical fields, that is, the bot is able to classify words that are semantically close to the ones defined in the entities, even if it is not trained using such words.

In Figure 8, the word "mélancolique" appears nowhere in the training examples, however, the bot successfully classifies it as a sadness example, and responds accordingly (with some irony!).


Je me sens mélancolique

Moi aussi je suis mélancolique parfois, mais je n'en fais pas toute une histoire comme toi.

Figure 8: Example of reaction to sadness

3.2 Generating responses to emotions

For generating a response to a user's utterance that involves emotions, the bot tries as much as possible to be the least generic as possible by stating the emotion the user has just expressed, in the response. This is a tricky problem, since it is important to know whether the expression was expressed using a noun, verb, adverb, adjective, etc — which is known as part-of-speech tagging[2]. For this matter, we used **SpaCy**[**?**], which is an open-source software library for advanced Natural Language Processing. Part-of-speech tagging is one of the main features on SpaCy; an example of how it runs is depicted in Figure 9.

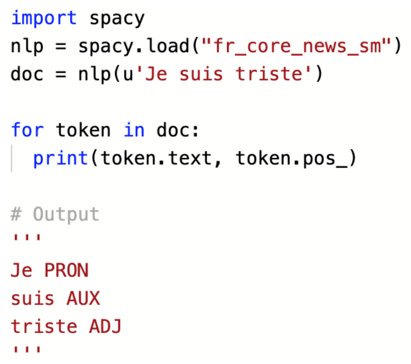
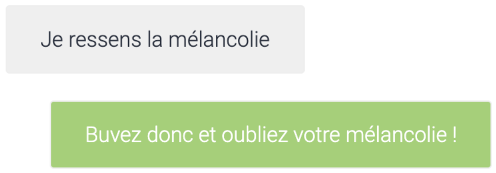

```
import spacy
nlp = spacy.load("fr_core_news_sm")
doc = nlp(u'Je suis triste')

for token in doc:
    print(token.text, token.pos_)

# Output
'''
Je PRON
suis AUX
triste ADJ
'''
```

Figure 9: Part-of-speech tagging

Now that we are able to tag the speech it becomes easier to reply in an appropriate way. Figure 10 shows an alternative way of expressing the sadness expressed in Figure 8, but using a noun (”mélancolie”) instead.

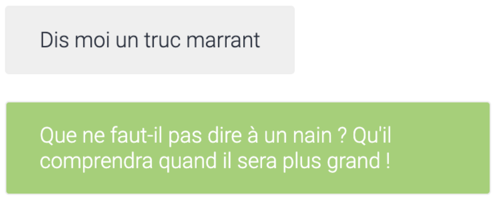

Je ressens la mélancolie

Buvez donc et oubliez votre mélancolie !

Figure 10: Alternative way of reaction to sadness

3.3 Jokes

One of the coolest features of the bot is telling jokes! A user can ask the bot either explicitly (e.g.: "Raconte-moi une blague") or implicitly (e.g.: "Fais moi rire") and the bot will randomly chose a joke from the ones predefined. A potential extensive work would be to link the "Joke" intent into an API that returns random jokes that defined in a large number, to make the bot less repetitive. Figure 11 is a good example of a user requesting a joke.


Dis moi un truc marrant

Que ne faut-il pas dire à un nain ? Qu'il comprendra quand il sera plus grand !

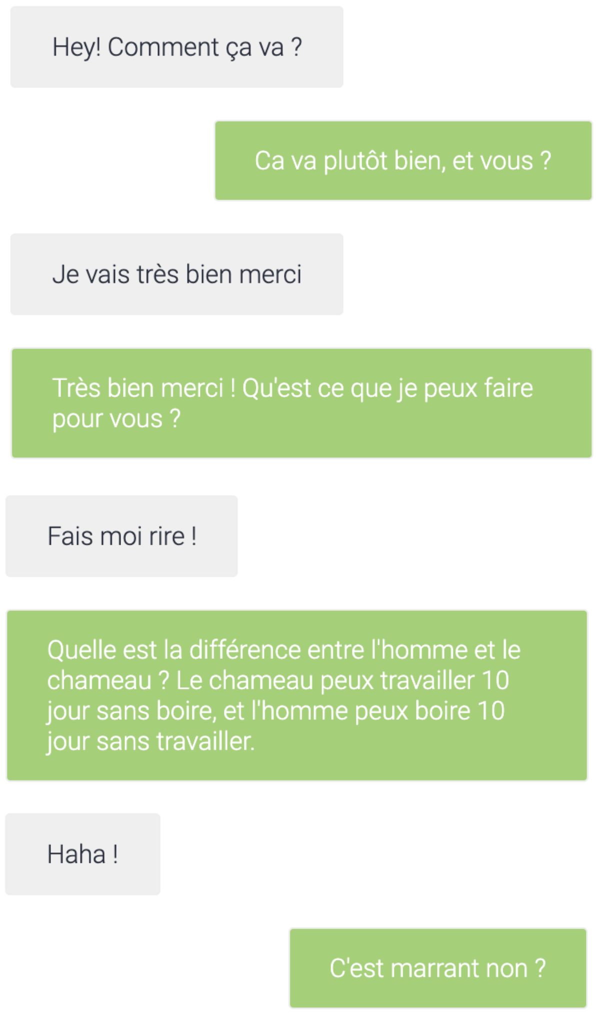
Figure 11: Bot reaction to a joke request

3.4 Small talk

Being able to conduct small talk is a very good feature that every chatbot should possess. It adds a "human-like" behavior that makes the conversation closer between the 2 parties, and almost forget that you are conversing with a machine. In order to cover up the "Small Talk" intent, various examples for daily life has been added to the training phrases. These include:

- Some information about the bot, so it knows who it is. (Name, age, etc.)
- Courtesy phrases, to respond the user (Hello, goodbye, thanks, etc.)
- Response to an ambiguous utterance, i.e. a one that could not be classified in any of the previously defined intents

Here is some small talk in a conversation with the bot.


Hey! Comment ça va ?

Ca va plutôt bien, et vous ?

Je vais très bien merci

Très bien merci ! Qu'est ce que je peux faire pour vous ?

Fais moi rire !

Quelle est la différence entre l'homme et le chameau ? Le chameau peux travailler 10 jour sans boire, et l'homme peux boire 10 jour sans travailler.

Haha !

C'est marrant non ?

Figure 12: Example of small talk in a conversation

4 Communication protocol with external APIs

When the a webhook is used to generate a response, the fulfillment code is called, and according to the respective intent, an HTTP request is sent to the correct external API. In a nutshell, the agent parses the user's utterance, matches it to the correct intent and call the fulfillment code that connects to the external API.

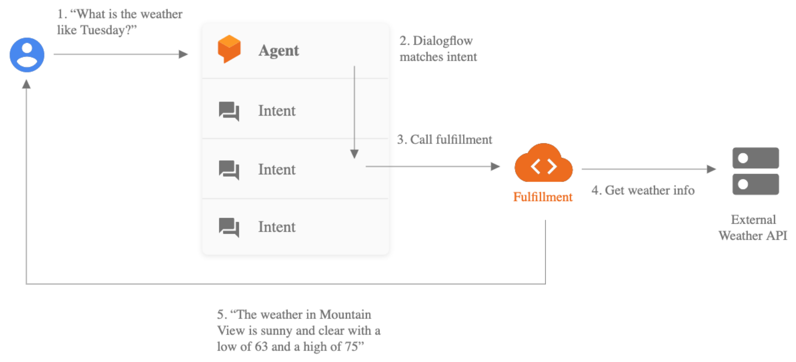

1. "What is the weather like Tuesday?"
Agent
Intent
Intent
Intent
2. Dialogflow matches intent
3. Call fulfillment
Fulfillment
4. Get weather info
External Weather API
5. "The weather in Mountain View is sunny and clear with a low of 63 and a high of 75"

Figure 13: Webhook: Communication protocol

Step (3) in Figure 13 is managed through **Flask**, which is a micro web framework written in Python. It is extremely flexible, simple and use and easily extendable. Figure 14 shows an example of how Flask chooses the correct API based the intent.

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/webhook', methods=['POST'])
def webhook():
    req = request.get_json()

    if req.get("queryResult").get("action") == "weather":
        ''' Query the weather API '''

    if req.get("queryResult").get("action") == "wiki":
        ''' Query the Wikipedia API '''

    if req.get("queryResult").get("action") == "tristesse":

        ''' Use trained french SpaCy models to return
        gramatically correct responses '''

if __name__ == "__main__":
    app.run(port=5000)
```

Figure 14: Flask abstract code for communication with external APIs

The external APIs that the chabot communicates with are **OpenWeath-**

erMap and **Wikipedia**.

4.1 OpenWeatherMap API

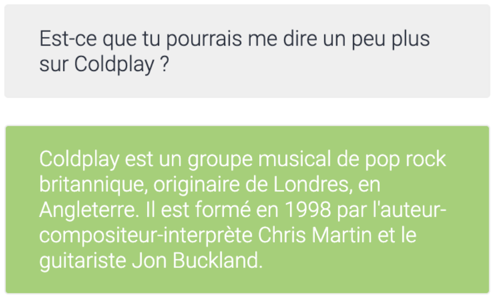
If a user queries the weather in a particular city in the world, the city name is extracted as an entity for slot-filling, and an HTTP request is sent to `https://openweathermap.org/` provided the user owns a valid API key.

```
url = "http://api.openweathermap.org/data/2.5/weather?q=" + city +
"&units=metric&lang=fr&APPID=" + api_key

# Query the API
data = json.loads(urlopen(url))

# Extract the temperature, and the weather condition
temp = data.get("main").get("temp")
condition = data.get("weather")[0].get("description")
```

Figure 15: OpenWeatherMap query

4.2 Wikipedia

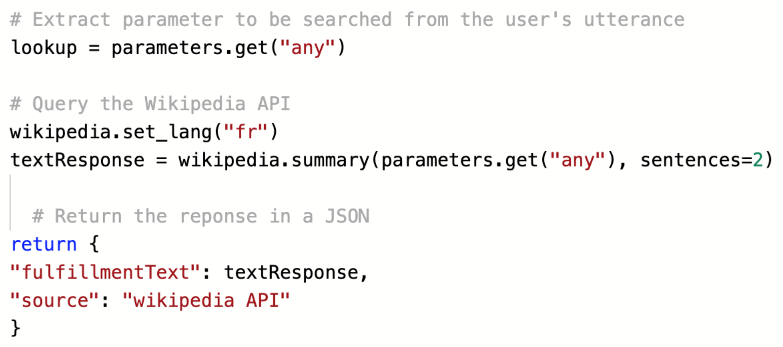
When the user's utterance is not matched to any of the explicitly defined intents, such as ordering, asking for recommendation, "colère" (anger) or "tristesse" (sadness), it queries the Wikipedia API with the most significant keyword in the utterance. The number of sentences is limited to 2, to keep the answers short realistic, as if the bot was conversing with the user.


Est-ce que tu pourrais me dire un peu plus sur Coldplay ?

Coldplay est un groupe musical de pop rock britannique, originaire de Londres, en Angleterre. Il est formé en 1998 par l'auteur-compositeur-interprète Chris Martin et le guitariste Jon Buckland.

Figure 16: Example of external API call

We can see that the most significant look-up keyword is "Coldplay", so this keyword is searched using the **Wikipedia** library, available on Python.

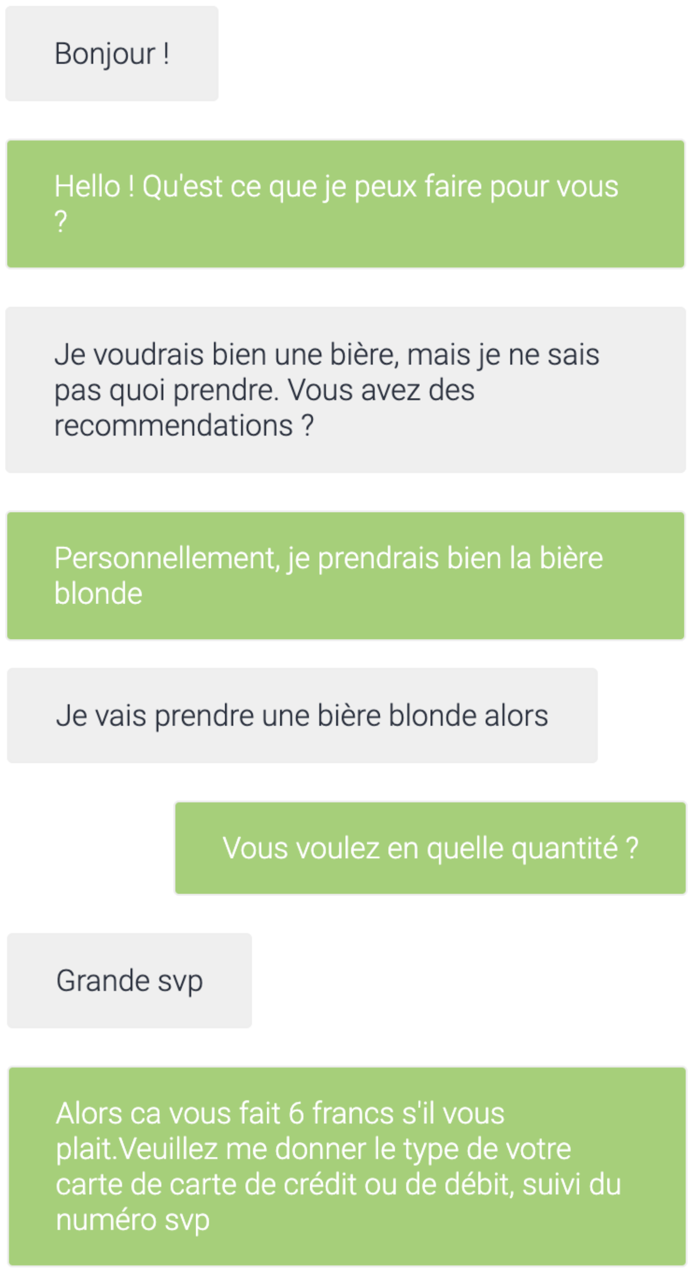


```
# Extract parameter to be searched from the user's utterance
lookup = parameters.get("any")

# Query the Wikipedia API
wikipedia.set_lang("fr")
textResponse = wikipedia.summary(parameters.get("any"), sentences=2)

  # Return the reponse in a JSON
return {
"fulfillmentText": textResponse,
"source": "wikipedia API"
}
```

Figure 17: Wikipedia query

5 Full transaction

Now that all the components and the capabilities of the chatbot are introduced, we present below an example of a full transaction, from greeting the chatbot to the beer ordering.



Bonjour !

Hello ! Qu'est ce que je peux faire pour vous ?

Je voudrais bien une bière, mais je ne sais pas quoi prendre. Vous avez des recommendations ?

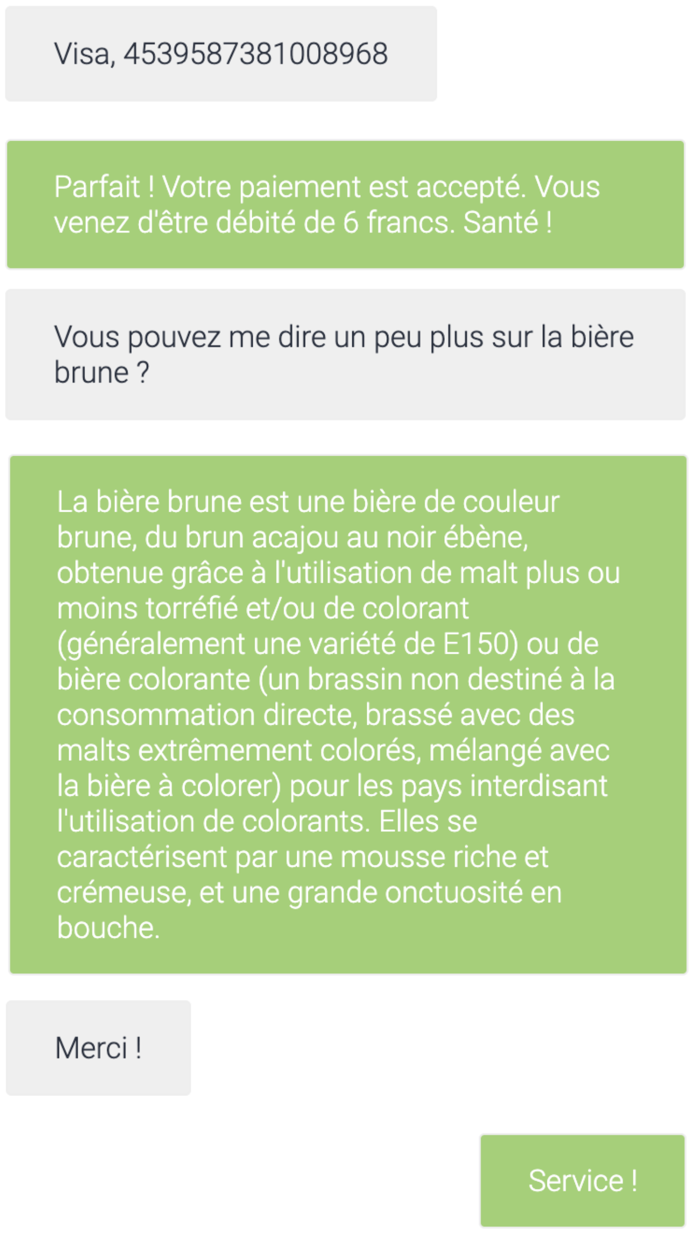
Personnellement, je prendrais bien la bière blonde

Je vais prendre une bière blonde alors

Vous voulez en quelle quantité ?

Grande svp

Alors ca vous fait 6 francs s'il vous plait.Veuillez me donner le type de votre carte de carte de crédit ou de débit, suivi du numéro svp



Visa, 4539587381008968

Parfait ! Votre paiement est accepté. Vous venez d'être débité de 6 francs. Santé !

Vous pouvez me dire un peu plus sur la bière brune ?

La bière brune est une bière de couleur brune, du brun acajou au noir ébène, obtenue grâce à l'utilisation de malt plus ou moins torréfié et/ou de colorant (généralement une variété de E150) ou de bière colorante (un brassin non destiné à la consommation directe, brassé avec des malts extrêmement colorés, mélangé avec la bière à colorer) pour les pays interdisant l'utilisation de colorants. Elles se caractérisent par une mousse riche et crémeuse, et une grande onctuosité en bouche.

Merci !

Service !

Figure 18: Full transaction example

6 Conclusion & Future Work

When designed well, chatbots can be extremely effective and helpful. Nonetheless, one of the most challenging difficulties is training the bot on emotions. A corpus containing emotional dialogues and reactions could be used in the future to strengthen the bot's ability to recognize more emotions. Consequently, more entities and intents could be defined, and a good NLP model could be implemented for response generation, and result in a more genuine human-computer interaction.

References

[1] OpenWeatherMap API Docs. `https://openweathermap.org/appid`

[2] Part-of-speech tagging, Spacy's website. `https://spacy.io/usage/linguistic-features#pos-tagging`

[3] Industrial-Strength Natural Language Processing, Spacy's GitHub repository. `https://github.com/explosion/spaCy`