SAPIENZA
UNIVERSITÀ DI ROMA

**Artificial Intelligence**

ACADEMIC YEAR 2020/2021

# Spoken Human-Robot Interaction

Rossella Bonuomo, 1923211

Emanuele Giacomini, 1743995

Veronica Vulcano, 1760405

*Master's Degree in Artificial Intelligence and Robotics*

## Contents

## 1 Introduction

The goal of this project is to build a spoken dialogue system in order to let the user talk with the agent about the COVID-19 emergency. The user can ask information about the number of infected, cured and dead patients in each Italian region; moreover, the user can ask for statistical information such as the contagion rate, the death rate or the cure rate given the fact that the agent knows the population of each region. In addition, the agent is learned about how to behave when:

- The user does not feel well; by considering the user's symptoms, the agent suggests him what he should do.
- The user has met an infected person.
- The user is infected.

We remind that the data stored in the knowledge base of the agent are acquired until 31$^{st}$ December 2020.

The project is composed of three main modules:

- Listener
- Speaker
- Agent

## 2 Listener and speaker

This is the part of the project used to provide an interface between the system and the user. We have developed two modules to let the user speak with the agent, and to let the user listen what the agent has to say about a specific topic.

### 2.1 Listener

The listener module uses the speech recognition library to take an audio input from the user microphone and to analyze it. In fact, the voice command is stored in a string to be used from the other modules. The speech recognition that we have used is the one from Google.

```python
import speech_recognition as sr

class Listener():

    def __init__(self):
        return

    def listen(self):
        r=sr.Recognizer()

        with sr.Microphone() as source:
            r.adjust_for_ambient_noise(source)
            print("Bot: Sto ascoltando. . .")
            audio=r.listen(source)

            sentence=""
            taken=True
            try:
                sentence=r.recognize_google(audio, language='it-IT')
                print("User: ", sentence)
            except sr.UnknownValueError:                            # speech is unintelligible
                taken=False


        return sentence,taken
```

Figure 1: The Listener class

To incentive the user to speak we print the sentence "Sto ascoltando..."; when this sentence appears, the user is aware that he can start talking with the agent. Moreover, to be sure that the signal is acquired properly, we print the understood sentence.

## 2.2 Speaker

The speaker module enables the agent to say something. In order to implement it, we have imported the library *pyttsx3*, which is a text-to-speech conversion library from Python.

```python
import pyttsx3


class Speaker:
    def __init__(self):
        self.engine = pyttsx3.init()
        self.engine.setProperty('volume', 5.0)

        self.engine.setProperty('rate', 115)

        voices = self.engine.getProperty('voices')
        self.engine.setProperty('voice', voices[36].id)

    def speak(self, sentence):
        self.engine.say(sentence)
        self.engine.runAndWait()
```

Figure 2: The Speaker class

We have set the volume level, the speaking rate and some properties of the voice. Then, we have defined the *speak* function which is called in main.py in order to let the agent talk; in fact, this function takes as input the string that the agent has to say.

# 3 Agent

The "Agent" class requires as parameters a reference to a Speaker through which a voice response will be produced, and a path to a knowledge base file encoded as JSON.

## 3.1 Implementation

During the initialization phase, the *handle_fn* map is defined, which associates all possible interaction keys to the respective callback functions. Each callback function receives as arguments the reference to a speaker module, the interaction key, possible features extracted from the input sentence, and finally, the reference to the KB currently in use. Within the function, the decision process that generates the answer and changes inside the KB occurs. To find the interaction key and any key information within the input, the *Agent.process* function is used, which performs a pattern matching operation among all the patterns provided by the "interactions" module. When the match is found, both the key and the groups matched by the algorithm are returned. The first one obviously represents the interaction key, while the second one represents the group of key information used as arguments for callbacks.

## 3.2 Interactions

In interactions.py, we have US_AG_INTERACTION_MAP and US_AG_RESPONSE_MAP. The former contains several indexed questions that the user can ask to the agent. The latter contains the responses associated to those questions.

# 4 Knowledge base

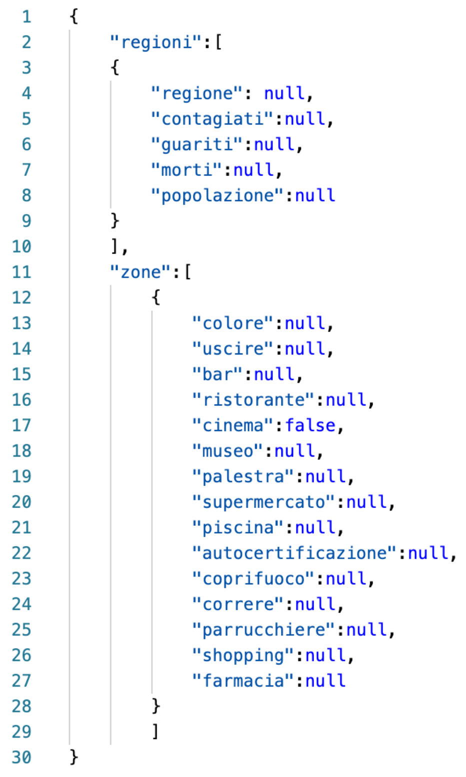
The main part of the project is the definition of the knowledge base. The knowledge base itself, is modeled as a JSON file because this allows us to read and write data in a fast way.

In order to handle the KB and to access to the structures that compose it, we create a knowledge base class called KB. This class has some functions that we can use to work on the specific data that we are looking for. Let's see in detail how we have modeled both the KB class and the JSON file.

## 4.1 JSON file

As we have already told, the knowledge base is a JSON file and it is organized in two main parts:

- In ”regioni” we have all the Italian regions. For each of them, we have the voice ”regione” to denote the name, then ”contagiati”, ”guariti”, ”morti”, for the number of infected, cured, dead people respectively and ”popolazione” for the population. This is the part of the knowledge base built in a procedural way; in fact, this is the knowledge that we have given to the agent and that we are not going to modify.
- In ”zone” we have the three kind of areas in which Italy has been divided during the pandemic situation. The voice ”colore” denotes the color of the area (red, orange or yellow), then we have other 14 voices like ”uscire”, ”ristorante”, and so on which can be either true or false according to whether that thing is allowed in the particular area. At the beginning, the agent only knows the colors of the areas, in fact this is the part of the knowledge that will be updated according to the information provided by the user.



```
{
    "regioni":[
    {
        "regione": null,
        "contagiati":null,
        "guariti":null,
        "morti":null,
        "popolazione":null
    }
    ],
    "zone":[
        {
            "colore":null,
            "uscire":null,
            "bar":null,
            "ristorante":null,
            "cinema":false,
            "museo":null,
            "palestra":null,
            "supermercato":null,
            "piscina":null,
            "autocertificazione":null,
            "coprifuoco":null,
            "correre":null,
            "parrucchiere":null,
            "shopping":null,
            "farmacia":null
        }
        ]
}
```

Figure 3: The KB template

## 4.2 KB class

The KB class is used to deal with the JSON file and with the data written in it in a very simple way. The functions that we have defined to access to the KB and to modify it are the following:

- *get_kb*: it just returns the knowledge base.
- *load*: it allows to load the knowledge base once we have specified the path.
- *save*: we use this function to save the knowledge base. When the agent is told new knowledge that it has to learn, we are going to save the expanded knowledge base writing explicitly in the path the date of the modification.

- *get_region*: this function returns the specific region that we are looking for. It will be used to extract the field of interest of each region in the KB.
- *get_dato*: it is used to extract the data that we want to work on in the knowledge base.
- *set_dato*: it is used to add information in the KB. It takes as input the color of the zone (e.g. "rosso", "giallo", "arancione"), the specific thing that we want to modify (e.g. "bar", "museo", etc) and the word "posso", "non posso". If the word is "posso", it means that the specific thing is allowed in the zone, therefore, we set it as true. Otherwise, "non posso" means that we cannot do that specific thing, so we set it as false. This is a very simple way to understand what a person can or cannot do in the specific zone.
- *get_zone*: it takes as input the zone color that we want to analyze and returns it.

## 5 Conclusions

The goal of the project is to understand how a spoken dialog system works. Even if the project is very simple, we have understood the basic structures that are needed in order to implement these kind of frameworks.

We have decided to focus on the pandemic situation that is affecting the whole world this year, but we have decided to deal only with statistic data concerning Italy. Moreover, the agent is able to advice people about how to behave when they are infected, or when they are worried about being infected.

It is possible to make some improvements by considering also statistical data of other countries. In this way, this simple dialog system can be really useful in a very hard situation like the one we all are facing.