Cognitive Robotics Study Project WS2019/20

Playing the 20Q Game with Pepper

Authors:
Anne-Kathrin Patricia Windler
Eunyoung Hwang
Minhee Kim
Yeji Kim

Supervisors:
Prof. Dr. Kai-Uwe Kühnberger
Prof. Dr. Gordon Pipa

A report submitted at the Institute of Cognitive Science
Osnabrück University

March 31, 2020

Contents

1 Introduction **1**

2 Scope of the Project **4**

2.1 The Study Project 4

2.2 The 20 Question Game 5

2.2.1 The description of 20 Question Game 5

2.2.2 The application of the 20Q game 6

3 Conversation **6**

3.1 Introduction 6

3.1.1 Main Goal and Achievements of the Group 7

3.1.2 Previous Work 7

3.2 Watson Assistant 8

3.3 General Structure 10

3.4 Dialogues 11

3.5 Python 12

3.5.1 Python and the ”I spy” game 12

3.5.2 Python and the 20Q game 14

3.5.3 Python for dialogue testing 15

4 Speech Recognition **18**

4.1 Theory on speech recognition 18

4.2 Speech recognition in Python 19

5 Physical Functions of Pepper **21**

5.1 Introduction 21

5.2 Touch Screen 22

5.2.1 Tablet 22

5.2.2 Challenges of the Touch Screen Part 23

5.3 Motions 23

5.3.1 Choregraphe 24

5.3.2 Challenges of Using Choregraphe 26

5.4 Mobility 26

6 Conclusion 27

7 Outlook 28

A Team Members and Attributions 32

A.1 Study project members 32

A.2 Contributions of each member 32

1 Introduction

In this report we will describe our work on the study project ”Cognitive Robotics” in the summer semester 2019 and winter semester 2019/2020. The study project is part of the IKW and supervised by Prof. Kai-Uwe Kühnberger and Prof. Gordon Pipa.

The focus of the study project is the work with Softbank's Pepper robot. Pepper is, as described on the Softbank website, ”a robot designed to interact with humans”, wherefore the focus of the study project lies on developing social interactions that Pepper can have with humans.

Pepper is a 120 centimeter high humanoid robot that is able to move its upper body, arms, fingers and head, move through the room, grab objects, recognize speech and emotions and talk to humans. It has a tablet with an interactive touchscreen on its chest. It has 2 RGB cameras, one 3D sensor, two laser sensors, two infra-red sensors, as well as in-built human detection and object avoidance [1].

Pepper robots are used in various areas. In Japan, they are used for explaining and selling Nescafé products like coffee machines [2]. Another example is a Pepper robot called Josie at the Munich airport who welcomes passengers and can answer their questions about shops, restaurants and flight operations [3]. The robots are also used in education to help children developing social and emotional skills and offer educational programs for students with disabilities such as autism, emotional and behavioural disorders. Another application are retirement homes in which they entertain residents.

The institute of cognitive science has a Pepper robot called Lou since 2017 which led to the formation of this study project in the summer semester 2018. Lou had in the past been used for different purposes. For example, very early it was programmed to talk about neurobiology knowledge as a tutor. It has also been programmed to talk about cognitive science and offer some cognitive science related quiz questions, in order to inform people about cognitive science and the IKW when it is, for example, at a fair. In the previous study project, it was programmed to serve as a bartender's assistant at cocktail parties, to greet customers and take drink orders.

The goal of our study project is to program Lou to play a game in a humanoid and child-like way in order to entertain people and appear fun and cute. The motivation for this was that people felt uncomfortable talking to a robot, wherefore we wanted to make Lou appear cuter, more quirky and more child-like, in order to make it more fun to talk to Lou. Additionally, as stated above, Pepper robots are used to entertain and educate children. Therefore, we decided to implement children's games, namely the I spy game, and the 20Q game.

One large former project is the cocktail party project, which started in the winter semester 2018/2019. In that project, Pepper should work as a bartender's assistant which takes people's drink orders and tells the waiter who ordered which drink so that the bartender can serve the drink to the person. The task was based on the cocktail party challenge in 2018 [4].

The challenges that had to be solved in the task are:
1. Spatial navigation: The robot had to be able to move in a surrounding with people and approach people that wave at it.
2. Face recognition: The robot had to identify the person who talked to it and made an order.
5.2.2 Challenges of the Touch Screen Part As one of the crucial goals of using a tablet in this project is to make users interact with Lou, it would be better if we diversely used the table to reinforce visual effects. For instance, the tablet could show if the answer is correct or not when users touch the object they think of as the answer. Otherwise, if the tablet shows different images as the conversation proceeds, such as it shows another image as a response, users could easily feel like they truly communicate with Lou not only verbally but also visually. Furthermore, it is also nice if the game is implemented by using some techniques of Computer Vision such as Feature Extraction. Lou could detect features of the objects, such as colors or the position in the image, and use them to play the real guessing game. However, this might be too ambitious to finish in a semester with 4 people, wherefore we used entities of a dialog for Lou to detect the features. 5.3 Motions When it comes to social interaction, gesture is one of the most important features which helps humans illustrate what speech alone cannot provide, e.g. to convey referential, spatial or iconic information. This is not only the case for humans; humanoid robots that are intended to engage in a natural human-robot interaction also need to produce some gestures that go along with speech for comprehensible and intelligible behavior. According to research, humans evaluate more positively when non-verbal behaviors such as hand and arm gestures are accompanied with speech [20]. In order to maximize the effect of the interaction between users and Pepper by adding some gestures, we utilized Choregraphe. 5.3.1 Choregraphe Choregraphe [21] is a software by Softbank Robotics, the company that develops robots including Pepper. This multi-platform desktop application allows developers to easily create human-being-like motions and behaviors for the robot, such as dancing, waving, or even saying something by using the NAOqi. Choregraphe is indeed very accessible as the user does not have to code the behaviors. In the Box Libraries Panel, we could see possible behaviors and use them by simply dragging and dropping onto the Flow Diagram Panel. Also, we could see how it looks like on the screen by simulating a virtual robot in the Robot View. [22] It is also possible to add behaviors written in Python or C++, allowing for more detailed and quickly executed behaviors. We added various gestures between the conversation according to what Lou says in order to make it look natural. First of all, before Lou shows a picture, it starts with a greeting. This can be simply done by using the Say box from the Box Libraries Panel. 22 Figure 16: Box Libraries Panel, Flow Diagram Panel, and Robot View on Choregraphe Figure 17: Example Script of a Box 23 Such a short script can be edited on Choregraphe itself without a Python script, as you can see in the figure 17. Here, we modify the parameter ”Text” to make Lou say other sentences that we want. Secondly, Lou moves its arm when it shows the picture of the objects. To do this, we chose Hands Movement from the Box Libraries. By setting the parameters, Lou can move either the left or the right hand or both as well. There are other possible behaviors such as walking, sitting down and standing up, or even applied posture. Thirdly, we wanted to show the picture of the objects on the tablet as explained earlier. We chose the Python Script box on the

libraries and modify it as we wanted above. Finally, all the boxes are simply connected sequentially so as to define the sequence of action to perform. Figure 18: Connecting multiple behaviors sequentially 5.3.2 Challenges of Using Choregraphe Adding motions by using Choregraphe was one of the last parts of our project but crucial to make it high quality. How much the motions of Lou would affect users to feel friend- liness was one possible question of the usability test that we had planned. However, unfortunately, as far as the IKW building closed due to the pandemic, we were not able to work with Lou in person so we could not finalize the motion part and conduct the usability test. As there are various functions on Choregraphe itself including Autonomous Abilities where Pepper can decide proper actions autonomously [23], we recommend those who will work on the following project to use them proactively. Pepper making its own action detecting the surrounding environment and reinforcing its action accordingly will be fitting for the original purpose of this project. It is also true that using only Choregraphe might come at the expense of making the process flexible and the behaviors are slower at execution time. Coding a Python script would be helpful to solve the problems. 24

3. Speech recognition: The robot had to understand what the person says despite of the noisy environment.
4. Conversation & Contextual Memory: one has to build a realistic dialog system with Watson Assistant.

Each challenge was faced by an individual group. The result of the group was a working prototype with some simplifications of the original cocktail party challenge. For example, the robot did not move through the room and approach the guest, but the guests and the bartender had to walk to the robot. They could order drinks by speech or by choosing it on the tablet. The robot was able to save and recognize the bartender and the different guests and give a description of the guest to the bartender. The bartender had the option to tell Pepper that the ordered drink is not available, whereupon the robot would suggest other drinks to the guest.

As we wanted to focus on a particular part of Lou's functions and improve it in as much detail as we could, and as our study project group was smaller than the group before us, we chose a smaller task to solve. The I spy game still had a number of different challenging sub-tasks like object recognition, conversation and improvement of the speech recognition, while the 20Q game was a task with a smaller scope and focus due to the small size of our group. In the I spy game project in the summer semester 2019, our group already chose to focus on the dialog system. At this point, we learned how to use Watson Assistant to build a dialog system and how to integrate it with the other components of Pepper. The microphone group dealt with the challenge of using external microphones to improve the speech recognition while avoiding that Lou talked to itself (Lou's speech output was also detected as input), to filter noise and to localize sound sources. The object recognition group, on the other hand, faced the challenge of working with camera images of low resolution, the challenge of finding a suitable object recognition algorithm, and training it on our set of objects. In the winter semester 2019/20 we chose a new task, namely the 20Q game. We made the decision because firstly, we wanted to focus on the improvement of the dialogue options in more detail, in order to make the dialogue appear more natural, and, secondly, we were only four people left after the summer term 2019, wherefore it would not be feasible to work on a large variety of tasks like the large groups before us.

2 Scope of the Project

2.1 The Study Project

This study project started in the winter semester 2016 and Pepper did not arrive until later. This study project course offers students of the Cognitive Science Master Program the opportunity to work with Lou which is a humanoid Pepper robot, created by Softbank Robotics. Students in the study project work on the project for two semester and have opportunities to discover various different areas among human-robot interaction, conversation system, web development, and the creation of a knowledge base.

The goal of this project is to explore the different functions that the humanoid robot offers, improve its functions and to equip Lou with specific abilities to perform as a social robot, as the Pepper robot is used for welcoming customers, helping them, entertaining and teaching people and also available for private households and academic use.

Since Lou arrived, previous members of this project explored the ways in which it can be used for teaching and learning, using it as a neurobiology tutor. Later, they focused on improving several specific functions that Lou possibly could offer, such as conversation and contextual memory, spatial navigation, and facial recognition parts, to develop Lou's entertainment capabilities as well as performance as a waiter at a party.

As previous project work emphasized the abilities as a waiter at a party/ in a pub rather than entertainment abilities, we, in our project starting from summer semester 2019, wanted to continue to step the entertainment ability up and aimed to create the "I (eye) spy" game. As the purpose of the "I spy" game makes sense when Lou itself can recognize objects physically presented in front of Lou and a player, the object recognition part is essential for the game. Therefore, our first goal was more focused on making Lou recognize objects and lead the game as a controller in a human-like fashion.

However, there was a limitation with the object recognition that Lou only seems to be able to recognize objects by itself and perform the game, but in fact, Lou is not able to recognize objects physically shown with 'its own eyes'. Such being the case that previous work left limitations which is contrary to the purpose of the original "I spy" game as well as its literal meaning of the game name, it is too much for us, a smaller group than before, to solve those existing limitation of the object recognition, carry out all the works and consequently improve the game.

Thus, during the second semester, we intended to keep our existing goal to improve entertainment ability and focus on programming a new game and place emphasis on completion of the game rather than leaving limitations and constraints at the same time.

The game called “20 Question Game” is not only rather simple compared to the “I spy” game, but also can be played in a more human-like way, emphasizing the conversation part.

We split the work among several subgroups. The group “Smalltalk” created dialogues for each event. The group “Speech Recognition” focused on installing microphones and improving speech recognition in a noisy environment. The group “Robot’s Physical Part” focused on Lou’s abilities of movement, navigation, and its tablet. The details of the work of each group are explained in the following sections.

2.2 The 20 Question Game

2.2.1 The description of 20 Question Game

The 20 Question Game is a guessing game in which one person thinks of a specific word and the other person asks up to 20 questions to guess the words the first person has their mind. The first person (person A) can only respond with either “yes” or “no” to all the questions. Once person A comes up with a specific word, they tell the other person (person B) the upper category that the word belongs to. Person B makes a question to guess what word person A keeps in their mind. This is how the game continues to narrow down to the specific word.

Consequently, person B needs to ask questions about subcategories and attributes that fit to the upper category. If person B asks for a thing that does not fit to categories earlier suggested, the question is made invalid and not counted as a question. If this occurs two times in a row, person A wins. On the other hands, when person B guesses the word correctly within 20 questions, they win.

For example, person A comes up with ‘plate’ and give a clue of the upper category as ‘a thing in the kitchen’. Once the upper category is given, person B could ask “is it a tool that you use for cooking?” as a question about a subcategory belonging to the upper category, rather than ”Do you use that when you are in the shower?”. So their questions are meant to narrow down possible words to a specific one and get closer and closer to ‘plate’.

Since the response to all the questions should be either “yes” or “no”, the questions that could bring different responses depending on the situation, not ”yes” or ”no”, are made invalid. For example, when person B asks whether it is expensive or not, the question is not counted as a question due to the wide range of the price of plates and they have to make a valid question. For the other example, when the question is whether it is big or not, the question is also made valid as the question requires a subjective criterion for the size. Therefore, person B who guesses needs to make each question about an obvious and objective subcategory or feature. For example, when the correct answer is ‘dress’, it is more clear to ask whether it is women’s clothing or not rather than whether it is expensive or not.

2.2.2 The application of the 20Q game

In our game with Lou, we made a modification of the 20Q game where we only asked less than 10 questions for each object. We needed to collect objects that have respectively different and obvious forms rather than words that could be abstractive depending on what image a user could recall upon hearing the words. Moreover, as we set the subjects of our game as the age of children (5 - 11), we intended to collect the images of each object with definite and objective forms, categories, and chose their respective features to be uncontroversial so that the age group of children is familiar with them. The upper categories and objects are also set a limit on the number up to 2 to 4 so that Lou is able to handle the objects, their upper categories and their details.

The game starts with Lou presenting twenty-nine objects on its tablet. Each object is classified into one of two upper categories, e.g. living being or non-living things and also a couple of categories within each upper category. As a limited number of object images are presented in a picture during the game with Lou, we let the game start with Lou asking the first question about the upper category, instead of the user telling which upper category the object belongs to before the game starts. For example, once the user confirms to have chosen an object in the picture, Lou asks the user a question "is the object alive?" and sequentially narrows it down into smaller categories and finally to only one object, according to the answers given earlier by the user.

3 Conversation

Pepper as a social humanoid robot is an ideal model for educating people in this study project. Pepper is optimized for human interaction and is able to engage with people through conversation and its touch screen in the conversation and contextual memory part. As the purpose of this study project in general is to make human interaction with the Pepper robot natural and friendly, the purpose in this semester was to narrow down its function by making it possible to educate people, especially children, in an entertaining and more human-like way. According to the purpose, we focus on the game, called the '20Q', which could emphasize the conversation function of the Pepper robot and create appropriate dialogues for the '20Q' game, using Watson Assistant and Python.

3.1 Introduction

We work on the conversation part of Lou. For the function of conversation, we aimed to make Lou able to lead a conversation with a user and control the direction of the conversation so that the conversation flow can be kept during the 20Q game. For example, once Lou receives a command from the user to start the game, Lou asks approximately 5 to 10 questions to get closer to the correct answer, reacting according to the user's words. To make the conversation functions perform efficiently, we show a picture of all objects that can be chosen by the user on Lou's tablet and create dialogues for each object. Lou asks for a feature, recognizes the user's answer and asks for other features according to previous answers from the user. It is essential for us to consider as many different scenarios as we can so that Lou has no lack of capacity to handle unexpected events and answers, so that the unexpected events do not break away from the created dialogues of the '20Q' game and Lou is able to control the direction of conversation according to what the user says.

3.1.1 Main Goal and Achievements of the Group

The main goal of this study project is to make human-robot social interaction possible. Therefore, we focus on the function of conversation that Lou has, making it flow in a more natural way so that people can have the feeling of real-human-like interaction with Lou as well as making people feel comfortable with playing a game and having a conversation with Lou.

Our purpose is to make Lou handle the created diverse scenarios and take a lead in the conversation for the game. To be specific, once the user picks an object of all the object displayed on the tablet, Lou asks whether the object is alive or not. The user answers in the affirmative or the negative and Lou asks, according to the either affirmative or negative response, respectively 'whether the object is a plant or an animal' or 'whether the object is a thing edible, wearable, or drivable' as the right following question. This means each event is completely led to the following events by Lou's questions.

In more detail, our goal is to apply the '20Q' game while not losing the meaning of the game and to make Lou take the lead in playing the '20Q' game as a guesser and controller of the conversation direction. In order to achieve this, we use many different objects that have several features in common with each other, and show the objects on the tablet instead of recognizing the real objects.

3.1.2 Previous Work

The first 'I spy' game that we had focused on during the last summer semester in 2019 was created with six specific objects that have at least three well defined features. We had worked on creating a dialogue tree for all objects. For example, once the user picks an object, the user starts by saying e.g. *"I spy with my little eye something **blue**"*. Lou then narrows down the objects that are corresponding to the user's earlier clues and asks for the other clues about size and sequentially shape. When all clues are collected, Lou guesses the user's object. The game can start with Lou picking an object as well as the user picking an object. As only six objects that have specific attributes were used, a simpler conversation structure was created. Moreover, as each object differs from the other objects in at least one attribute, the game has a possibility to end quickly and easily.

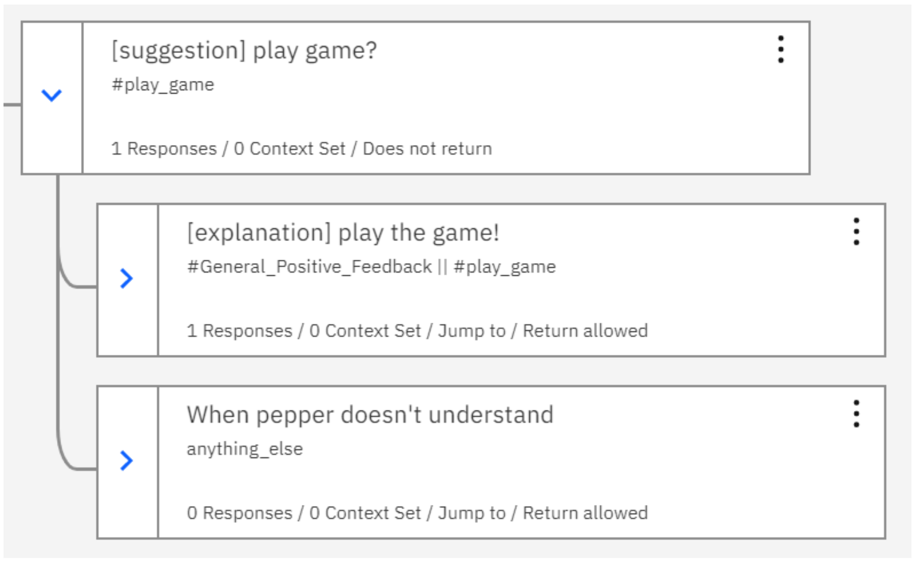
As the name of the ‘I spy with my little eye’ game implies, the function of Lou’s visual recognition is essential to make this game possible and enable Lou to participate in the game with people as a leading subject. Once the game starts, Lou should recognize all the objects displayed on the table with its ”eyes” no matter whose turn to guess it is. Although we had a group who tried to get it to work, it did not work until the end of the semester. Consequently, the limit of this game was the absence of the function of the visual recognition part of Lou, including a loss of meaning of the game, which means Lou was not able to recognize the objects presented on the table by itself.

Thus, we want to make a game that emphasizes its ability of the conversation so that it can take the lead in playing the game in a more natural, human-like way, ruling out the limitation of visual recognition. However, the goal of the ‘I spy’ game is not different to the goal of the 20Q game.

3.2 Watson Assistant

Description of Watson Assistant
Watson is an artificial intelligence system capable of answering questions in the form of natural language. Watson Assistant can be used to build customer’s own branched live chat bot into any application, device, or channel. Users can interact with the assistant through one or more integration points such as social media platforms or custom applications.

In our project, users interact with it through the Pepper robot. The assistant takes in user input and routes it to the dialog skill which interprets the user input further, then directs the flow of the conversation. The dialog gathers any information that it needs to respond or perform a transaction on the user’s behalf.


[suggestion] play game?
#play_game
1 Responses / 0 Context Set / Does not return

[explanation] play the game!
#General_Positive_Feedback || #play_game
1 Responses / 0 Context Set / Jump to / Return allowed

When pepper doesn't understand
anything_else
0 Responses / 0 Context Set / Jump to / Return allowed

Figure 1: A parent node and its child nodes

There are some components that you face when you build up your dialogue on Watson Assistant, one of which is called a node. As you can see in Figure 1, there are 3 nodes here. They are connected and composed in a hierarchical way. The first node is a parent node and the other nodes below are its child nodes. Of course, the child nodes can also produce their own child nodes. Those nodes are activated by intents and entities. Intents represent a collection of user's intentions, while entities contain a variety of values standing for each certain content in a conversation.

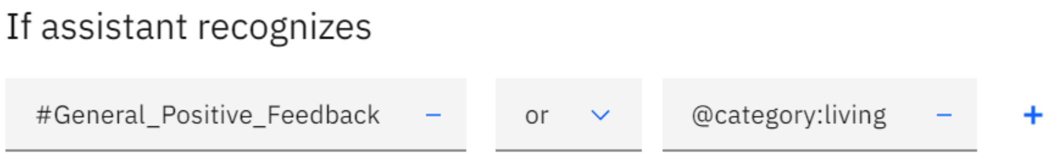


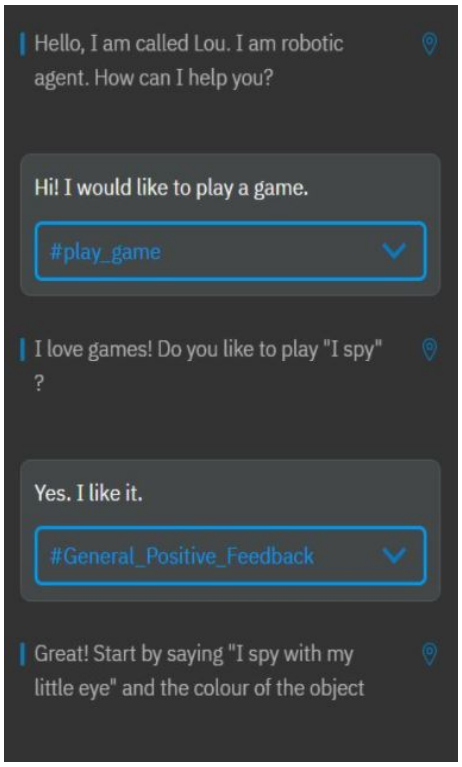
Figure 2: An intent and entity

Dialogues
A dialogue is structured in what is called a 'Decision Tree'. When the user says something, Watson will first identify the overall intent, and extract any entities mentioned. We use these intents and entities and other complex conditions to find the first logical matching dialogue node. If the user were to say something that Watson Assistant identified as the positive feedback intent or living category entity (Fig 2), it would match the condition and Watson Assistant would give the response associated with it.

Nodes
A node can have several types. First there is a basic FAQ dialogue. If a user asks, for example, what the agent's name is, the agent will give one response and return back to the start position of the dialogue tree and wait for the next response from the user. Next, there is the multiple conditioned response, which leads it to conditionally respond in a more sophisticated way. This type of response is most often used for responding to each entity value within an overall intent. The third type of node is called slots. You can use this to collect several pieces of information in any order, and Watson will only ask for the information it did not receive.

Intents
As seen in Figure 2, the one with # is called an *intent*, which is a collection of user's statements that can be said in order to achieve a goal. Intents are the major way for Watson to understand what the user is saying or asking for. For example, there is an intent named *#General_Positive_Response* which collects possible positive answers that are possibly said by users in many different ways. Although more expressions are always better, adding expressions that look almost identical to each other is never teaching Watson anything new. Rather, it would only take up your time.

We create suitable intents for our game, adding possible expressions and statements that contain user's intention. For example, we have the intent *#play_game*. *#play_game* is intended for our assistant to understand a variety of ways that user can express the specific goal, ‘game play’, with regard to this intent.


Hello, I am called Lou. I am robotic agent. How can I help you?

Hi! I would like to play a game.

#play_game

I love games! Do you like to play "I spy" ?

Yes. I like it.

#General_Positive_Feedback

Great! Start by saying "I spy with my little eye" and the colour of the object

Figure 3: Conversation test: intent recognized

Intents can activate nodes. Figure 3 is the result of testing our conversation. When a particular intention is expressed by a user, the blue tool displays that the specific intent was recognized. Consequently, our assistant can react with a suitable answer according to the recognized intent.

Entities
With entities, you can easily narrow down what exactly your user is referring to. While intents are the general way to ask pepper to do something, entities are usually the objects that they will want to do. Entities are also often used to quickly build a keyword base flow for simple input or even button click such as yes or no. You can also build pattern entities for things like area code (3 digit number) and zip code (5 digit number). Synonyms can also be added to each value, but you don’t need to add every single different form of the same word as Watson Assistant has a rough matching algorithm for its entities.

3.3 General Structure

As Lou is the agent leading the game, we want Lou to be able to ask questions that follow the flow of the conversation. Thus, we need to create dialogues fitting to each situation and including each object that the user could potentially choose during the game. For example, Lou explains how to play the '20Q' game when one wants to play the game. Lou serves as a guesser to lead the game into the appropriate direction. During the game, Lou asks a question about the upper category that the picked object could belong to, e.g. whether the object is alive or not. Once the user responds either "yes" or "no", Lou asks for the subcategory and more detailed attributes of the object, e.g. whether the object has legs, and, if the answer is yes, whether the object has two legs. Objects that do not fit to clues given before are excluded in this process.

3.4 Dialogues

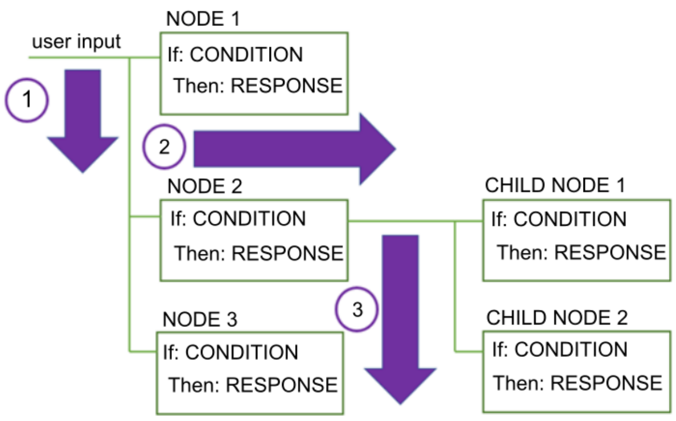


Figure 4: scheme of a dialogue (Miller & Dean, 2020)

The general scheme can be seen in figure 4. First, pepper asks if the user wants to play the game, then the user answers (user input). If the user's answer meets a particular condition (which is in this game, 'general positive response'), the game starts. Pepper then asks a range of questions ranging from broad categories to detailed features that an object could have. It memorizes the clues and continues with the following questions connected to those clues. The following questions get more specific and closer to the target object. This is how pepper narrows down what clues it should ask for reaching the last step. Each object has its own dialogue along the tree.
In this game, it starts with Pepper asking the most general question which is "Is it alive or not alive?". If the target object is, for example, a Coca-Cola bottle, the user should say no (general negative response). Then it asks whether it is edible or not. Then the user says it is edible, which is also processed as one of the general positive responses within this specific question. Now it proceeds under the subcategory named 'edible' where the user should decide how it is shaped (round/ long). The Coca-Cola bottle in the picture has a long shape. And when Pepper reaches this step, the remaining object in this specific sub-category is only the Coca-Cola bottle. In this way, Pepper guesses what the user picked.

3.5 Python

3.5.1 Python and the ”I spy” game

In this chapter, it will be explained how and why we combined the use of Watson Assistant and Python. In our first semester's task, the ”I spy” game, it was much easier not to only use Watson Assistant, but to complement it with a Python function to find a fitting object based on the three properties that are asked in the dialogue (color, size, shape). This is useful when we are in the subtree where the user chooses an object and Lou is guessing.

One challenge was to change the context variables in Watson Assistant through the Python program. It took a lot of trial and error until we were able to change the context variables reliably through the Python program. We need to import the client library *watson_developer_cloud.AssistantV1* (using Python 2.7) in order to access the dialogue that we created in Watson Assistant. For doing this, we create an instance of the Assistant and authenticate using the API-key [5], as shown in figure 5.

```
assistant = AssistantV1(
    version='2019-06-14',

    iam_apikey ='chSYJa3S5njj2lGGyqUovTsTEbYmbcMPDesroDyidCwV',
    url = 'https://gateway-lon.watsonplatform.net/assistant/api'
)
```

Figure 5: authentification with Watson Assistant

Figure 6 shows the function that sends a query to Watson Assistant containing the user input text and the context variables. You also have to specify the workspace_ID of the dialogue.

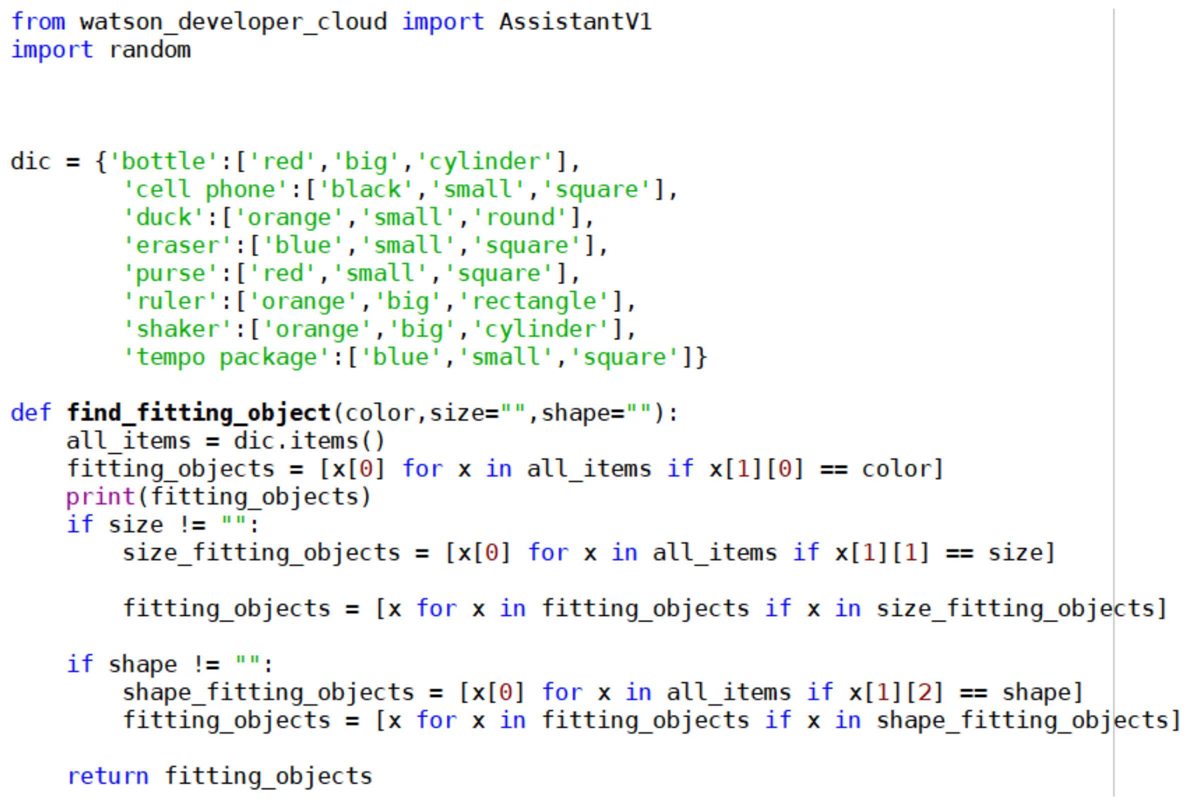
```
while(True):

    user_input = input()
    if user_input == "Quit" or user_input == "quit" :
        break
    response = assistant.message(
            workspace_id='019064cf-4468-449e-804d-8ba291707347',
            input={'text': user_input},
            context= context)
    old_context = context
    context = response.result['context']
```

Figure 6: accessing a response from Watson Assistant to a given user input

The response that we receive is in JSON format. We receive not only the answer text, but also the context variable assignments and a list of dialogue tree nodes that were visited, among other information. We have to save the context that was received in the last query and pass it when we run the next query, so that Watson Assistant knows the previous progression of the dialogue and where in the dialogue tree we are. Otherwise, it would not remember what had already been said and not understand e.g. when the user answers Lou's question, because it would "have forgotten" that it had asked a question. An advantage is that we can also make changes in the context variable assignments before passing the context to the next query.

The simplification through Python was the following: At the dialogue node where Lou should guess the object that the player had chosen, the *find_fitting_object* Python function is called. We use the color clue, and optionally the size and shape clue that the user has given, as its arguments. Those context variables are saved in Watson Assistant and can be extracted from the JSON response in Python. The *find_fitting_object* function returns a list containing all objects that fit into the description. We then change the context variable called *objects* to one of the objects from this list, such that Lou will guess this object. In order to do this, we call the query again using the old context but with the updated *objects* variable. In the Watson Assistant dialogue, it is defined that Lou will guess the object saved in the *objects* variable.



```python
from watson_developer_cloud import AssistantV1
import random



dic = {'bottle':['red','big','cylinder'],
        'cell phone':['black','small','square'],
        'duck':['orange','small','round'],
        'eraser':['blue','small','square'],
        'purse':['red','small','square'],
        'ruler':['orange','big','rectangle'],
        'shaker':['orange','big','cylinder'],
        'tempo package':['blue','small','square']}

def find_fitting_object(color,size="",shape=""):
    all_items = dic.items()
    fitting_objects = [x[0] for x in all_items if x[1][0] == color]
    print(fitting_objects)
    if size != "":
        size_fitting_objects = [x[0] for x in all_items if x[1][1] == size]

        fitting_objects = [x for x in fitting_objects if x in size_fitting_objects]

    if shape != "":
        shape_fitting_objects = [x[0] for x in all_items if x[1][2] == shape]
        fitting_objects = [x for x in fitting_objects if x in shape_fitting_objects]

    return fitting_objects
```

Figure 7: attribute dictionary and *find_fitting_object* function

The *find_fitting_object* function works as follows: It uses an underlying dictionary whose keys are the game objects and whose values are the lists of the three attributes that belong to each object (for example: 'ruler':['orange','big','rectangle']). It first extracts all objects that have the given color into a list. Subsequently, if the size clue is given, it extracts all objects of the given size into another list, and then selects all objects that are in both lists. It does the same for the shape clue, if it is given. Finally, it outputs the remaining list.

If the guessed object was right, the game is ended and Lou happily states that it has won the game. If the guessed object was wrong, it is added to a list called *already_asked_objects*. The next time Lou is guessing, it will not choose an object that is in the *already_asked_objects* list.

If no object can be found that fits into the description and has not already been asked, the *objects* variable is set to *None* and Lou will give up and respond that it does not know anything that fits into the description and will ask which object it was.

When we reach a node where the game has ended, we use Python to clear all context variables so that they are empty and do not interfere with a new round of the game. We also empty Python variables like the *already_asked_objects* list.

By using Python to find the objects that fit into the description, the dialogue tree was simplified a lot. If we would use only Watson Assistant, the dialogue tree would have to split into multiple branches at each question, resulting in 24 pathways (because we have 4 colors, 2 sizes and 3 shapes). This dialogue tree is shown in figure 8. By saving the answers and evaluating them at the guessing stage, the dialogue tree has only one straightforward pathway consisting of the three questions, as shown in figure 9.

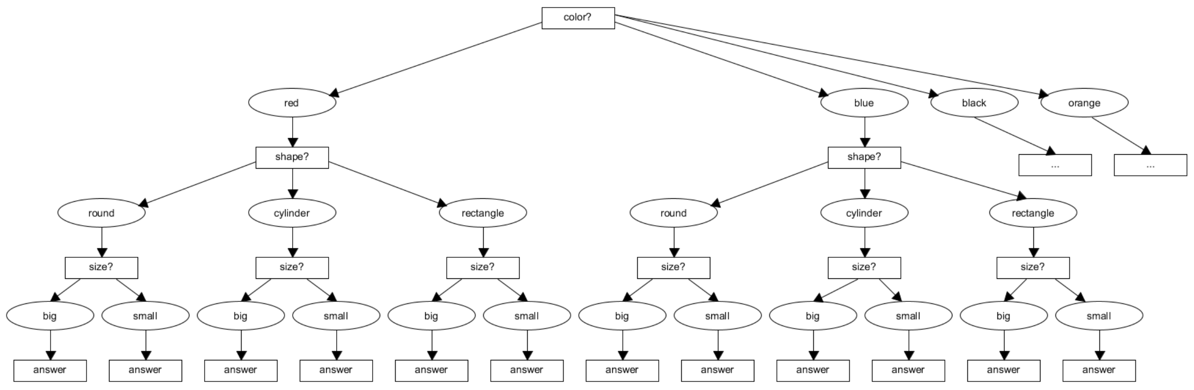


Figure 8: example of the pathways in the dialogue tree without the use of python

The use of Python was also necessary when the game was played the other way around, which means that the conversational agent chooses an object and the user guesses. The Python program is used to detect the node where a round of this version of the game is started, and it randomly chooses an object from the game object list. It then assigns the context variables *objects*, *color*, *size* and *shape* accordingly and calls the user query with the updated context again, so that Lou can start giving the color clue that has just been forwarded to the Watson Assistant dialogue. Whether the objects which the user guesses are right is then evaluated in Watson Assistant, as the right object is saved in the context variables.

3.5.2 Python and the 20Q game

As the 20Q project originated from the idea of the "I spy" game, which was then modified,

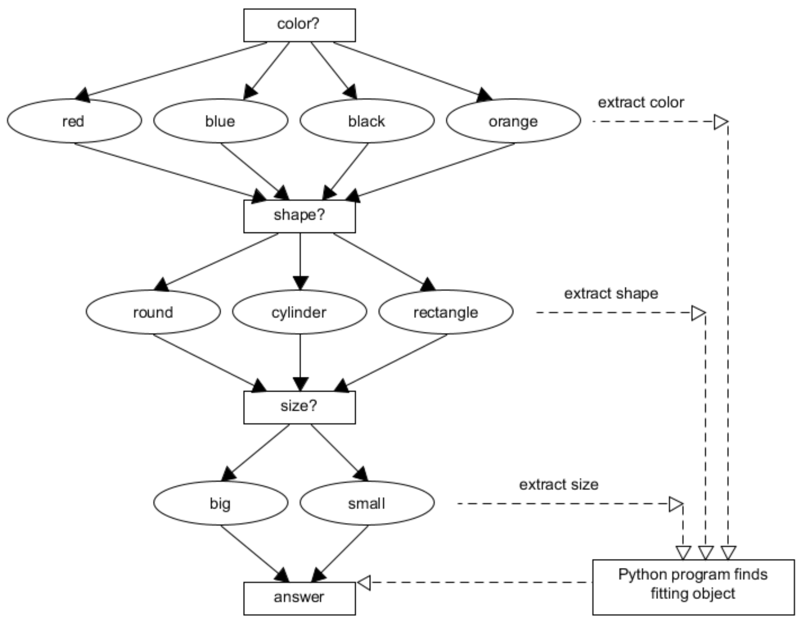


Figure 9: example of the pathways in the dialogue tree with the use of Python

we kept the structure, meaning the combination of Watson Assistant and Python when we started working on the 20Q project. We kept this structure until we relatively lately realized that the use of the Python program had become obsolete.

In our early versions, we extracted the context variables from Watson Assistant and used a dictionary-like function to determine which object was described. It is shown in figure 10.

At some point, we realized that this procedure of extracting the variables that were saved in Watson Assistant and using them together with a dictionary-like function did not yield any benefit, because when we travelled a certain path in the Watson Assistant dialogue tree and arrived at the node where Lou should decide which object was described, we could determine it without the combined variables, but from the path that was chosen. We first only had one node for guessing the object, and every path jumped to that node when all questions were asked, but we realized that we could also create one of these guessing nodes for each path and respectively each object that could be guessed.

The use of the dictionary-like function had also the major flaw that whenever the dialogue structure or the names of the entities in Watson Assistant were changed, the function also needed to be updated, because it could only extract the attributes correctly if they had specific names.

3.5.3 Python for dialogue testing

Another reason to use Python was to simplify the testing of the dialogue, especially when we wanted to test the deeper parts of the dialogue tree. At the beginning, we would test

```python
def find_fitting_object(atts):
    if atts == ['living','plant','green']:
        return "tree"
    if atts == ['not living','wear','yellow']:
        return "dress"
    if atts == ['not living','eat','bite','yellow']:
        return "bananas"
    if atts == ['living','plant','yellow']:
        return "bananas"
    if atts == ['not living','drive','four']:
        return "car"
    if atts == ['not living','wear','pink']:
        return "bag"
    if atts == ['not living','eat','bite','red']:
        return "apple"
    if atts == ['living','plant','red']:
        return "apple"
    if atts == ['living','animal','four','black']:
        return "cat"
    if atts == ['not living','wear','green']:
        return "socks"
    if atts == ['living','animal','four','white']:
        return "cat"
    if atts == ['not living','wear','blue']:
        return "jeans"
    if atts == ['living','animal','zero','green']:
        return "snake"
    if atts == ['living','plant','yellow']:
        return "sunflowers"
    if atts == ['not living','drive','two']:
        return "bicycle"

    if atts == ['not living','drink','brown']:
        return "cola bottle"

    if atts == ['not living','drink','black']:
        return "cola bottle"
```

Figure 10: *find_fitting_object* function in the early stage of the 20Q game

the dialogue only on the Watson Assistant web page, which offered an option to try out how the conversational agent answers to certain user queries. The problem was that, if we wanted, for example, to test a node that can only be reached after answering a number of other questions, we would have to type in all the answers that lead to this node each time we want to test it. The use of Python saves time because we can write a script that automatically answers the previous questions that lead to the node that should be tested.

```python
import Q20Chat as ngc

print(ngc.get_answer("Hello"))
print(ngc.get_answer("I want to play"))
print(ngc.get_answer("yes"))
print(ngc.get_answer("ok"))
print(ngc.get_answer("no"))
print(ngc.get_answer("no"))
print(ngc.get_answer("you drink it"))
print(ngc.get_answer("brown"))
print(ngc.get_answer("yes"))
print(ngc.get_answer("yes"))
print(ngc.get_answer("I am ready"))
print(ngc.get_answer("yes"))
print(ngc.get_answer("no"))
print(ngc.get_answer("green"))
```

Figure 11: example script for testing a dialogue

Figure 11 illustrates how this script looks like. All of Lou's outputs to the user's input phrases are printed. However, we cannot show the printed output because we do not have access to the service anymore. Therefore, we will recreate the dialogue from our memory. The dialogue in the example would look like this:

User: Hello
Lou: Hello, I am Lou. How can I help you?
User: I want to play
Lou: You like playing games? Me too! Do you want to play the 20Q game with me?
User: Yes
Lou: Great! You will start by choosing an object, and I will ask questions! Are you ready?
User: Ok
Lou: Is it alive?
User: No
Lou: It is not alive then... Is it something you can wear?
User: No
Lou: Then is it something that you can eat or drink?
User: You drink it
Lou: So it is a drink! Which color does it have?
User: brown
Lou: A brown drink. I see! Is it the cola bottle?
User: Yes
Lou: Yeah, I won this round! That was fun! Do you want to play it again?

User: Yes
Lou: Great, let's play another round! Tell me when you are ready.
User: I am ready
Lou: Ok, then my first question is: Is it alive?
User: Yes
Lou: I see! Is it an animal?
User: No
Lou: It is alive but not an animal. Then it must be a plant. What color does it have?
User: Green
[...]

If we, for example, want to test whether the transition from the node where Lou guesses the object correctly to the node where a new game is started works correctly, we use this script to answers all the questions that lead to Lou guessing the object (in this example cola bottle) correctly and then observe how Lou reacts if we vary our answers to Lou's question whether we want to play the game again. In order to do this, we would just change one line in the shown script in figure 11 - for example, replace the "yes" in the tenth call of the *get_answer* function by the sentence "I want to play it again" and observe whether Lou still responds as we expect. By using a Python script for this, we do not need to type in all of the answers each time that we want to test a certain node and therefore save a lot of time.

4 Speech Recognition

4.1 Theory on speech recognition

In order to play games in a humanoid and user-friendly way, the robot needs to convey the impression that it understands spoken language. Interpreting the audio signals that the microphone receives and translating them into a text sequence that is as close as possible to the spoken data is a difficult task that has been tackled by a number of approaches.

In the past, hidden Markov models [6] were often used for these speech recognition tasks. An example architecture is shown in figure 12. As shown in the figure, the acoustic signal had to be in the first step converted into a sequence of fixed size acoustic vectors through feature extraction. [7] The decoder then tries to find the most likely sequence of words that could have produced the observed sequence of acoustic vectors. It is comprised of an acoustic model which describes how words are formed out of basic speech units (phones). The phone model is trained on training data consisting of speech waveforms and their corresponding phones. The pronunciation dictionary defines how concatenations of the retrieved phones result correspond to words. The language model describes the probability of a word appearing in the context of its N predecessors. By putting the probabilities of the acoustic model, the pronunciation dictionary and the language model together, the decoder outputs the most likely sequence of words.

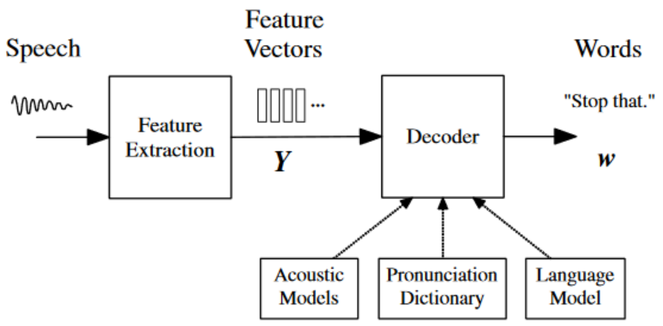


Figure 12: architecture of a HMM-based recogniser. Figure taken from [8]

In the last few years, hidden Markov models have been improved and outperformed by deep learning models such as recurrent neural networks (RNN). [7] One example is Google's *Convolutional, Long Short-Term Memory Fully Connected Deep Neural Network* (CLDNN) [9], which is a deep learning improvement of the traditional hidden Markov models with good performance.

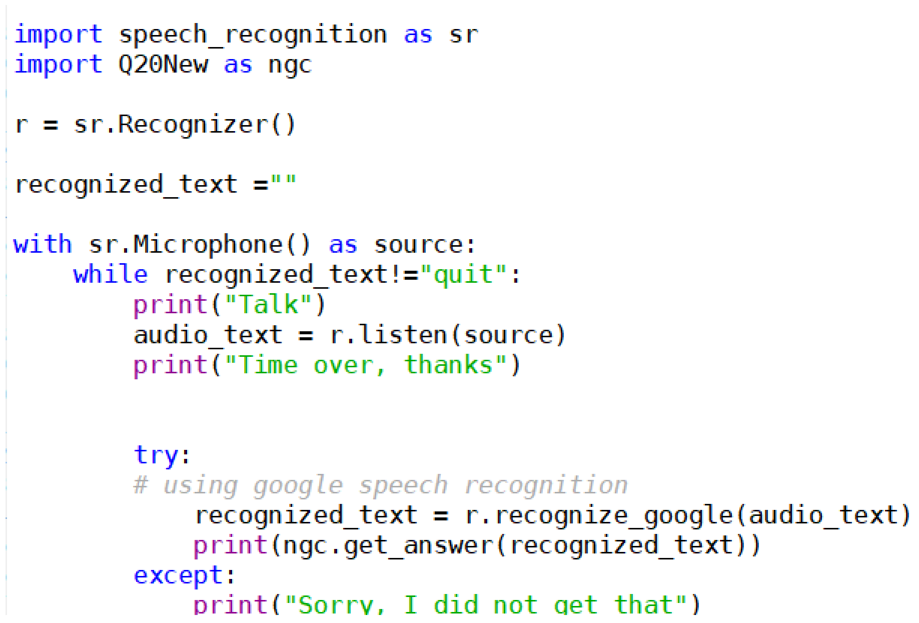
Another example is *Listen, Attend and Spell* (LAS) [10] which uses an RNN-based encoder and an attention decoder. Its innovation is that it does not consist of the acoustic, pronunciation and language models, but, in contrast, it is an end-to-end model.

4.2 Speech recognition in Python

In our first semester of our study project, we focused only on the dialogue part, where we worked only with written text. The speech recognition system was already set up by the people who worked on the study project before we started, and another group was responsible for speech recognition and microphones during our first semester. In our second semester, there were less people in the study project and the speech recognition became our responsibility. However, we first focused on the dialog with written text, before we directed our attention to the speech recognition part. Until the end of our second semester, the speech recognition provided by the groups before us which used Google's speech-to-text services [11] (Google Cloud Speech API) worked well, but it expired before we were finished with our project, so could not use it any longer. We then thought about alternatives and found a Python package called *SpeechRecognition* [12], which worked well when we ran it on our own computers. It enabled us to play the game with the conversational agent by speaking into the computer's microphone and receiving the conversational agent's answers as text output with a short delay.

The *SpeechRecognition* python package is easy to use and works out of the box with only a few lines of python code. It has a built-in function to access the computer's microphone, so it was easy to set up on our own computer, although it might be more difficult to set it up on the robot which we were not able to test yet, due to reasons we will explain in chapter 6. The package has a *listen* function which detects speech in the audio signal from the microphone and stops the recording when it detects silence.

The detected audio signal can then be processed by a speech-to-text service of the user's choice. The package offers integration for 9 speech recognition services, including Google Speech Recognition, Google Cloud Speech API [11], CMU Sphinx [13] and IBM Speech to Text [14], among others. We chose Google Speech Recognition, which is, in contrast to Google Cloud Speech API, free to use. The package offers functionalities to customize the speech recognition to one's need. Beside the use of different speech recognizers, one can, for example, change the language or dialect, or the sensitivity of the speech recognition which can be adapted for noisy or silent environments.



```
import speech_recognition as sr
import Q20New as ngc

r = sr.Recognizer()

recognized_text =""

with sr.Microphone() as source:
    while recognized_text!="quit":
        print("Talk")
        audio_text = r.listen(source)
        print("Time over, thanks")


        try:
        # using google speech recognition
            recognized_text = r.recognize_google(audio_text)
            print(ngc.get_answer(recognized_text))
        except:
            print("Sorry, I did not get that")
```

Figure 13: use of the *SpeechRecognition* package

An example of how we used the *SpeechRecognition* package is shown in figure 13. After the speech is converted to text, it is transmitted to the *get_answer* function of the 20Q package which calls the Watson Assistant API to receive the conversational agent's answer. It also formats the answer, which can consist of lists containing Unicode strings, into one standard string. After the answer is printed, the *listen* function is called again and will record the next speech input from the user which can be a response to the last answer that the conversational agent just gave.

5 Physical Functions of Pepper

Since the concept of the 20Q game itself is quite simple, the frameworks that have been described so far would be indeed enough. However, one of the main purposes of the Pepper robot is to make users get familiar with robots and to reinforce the communication with users. In order to supplement this purpose, we decided to use additionally various physical functions that Pepper has, including Touch Screen, Movement, and Mobility. A touch screen shows images to users and lets users touch the screen so that makes users feel that they are interacting with the robot. Pepper can also make some motions with an inbuilt software named Choregraphe which helps Pepper to act more like a human-being. Furthermore, the mobility of Pepper assists to obtain activity. In this part, we are going to briefly explain each feature and how we have used them in our project.

5.1 Introduction

Before we dive into it, let us shortly explain some basic tools from Softbank that we mainly used. Although the opinion that they have many limits has been predominant in the previous projects, using the tools allowed us to save time with a limited number of people. Softbank offers not only Choregraphe but also SDK and NAOqi APIs.

API stands for Application Programming Interface which is a set of functions and procedures that allow for the creation of applications that access data and features of other applications, services, or operating systems. [15] The NAOqi API that Softbank provides is currently available in more than 8 languages. There are some core modules to manage basic systems and all of them come with a list of default methods. For example, ALBehaviorManager is used to start and stop behaviors, ALMemory is used to get and insert data for every other module, and ALWorldRepresentation is used to store long-term data about detected objects in a spatially structured database. Not only that, there are modules for Motion, Audio, or Vision as well, each of them coming with various methods. You can have a look at the document provided by Softbank to utilize further functions. [16] Among those many functions, we mainly used ALTabletService to handle the tablet for loading images and ALMotionProxy to facilitate making the robot movement.

On the other hand, SDK which stands for Software Development Kit functions provide a set of tools, libraries, relevant documentation, code samples, processes, and guides that allow developers to create software applications on a specific platform. Understanding the difference between API and SDK is important. If an API is a set of building blocks that allow for the creation of something, an SDK is a full-fledged workshop, facilitating creation far outside the scope of what an API would allow. [17] In this project, we used the python SDK from Softbank to run the existing modules or create additional modules.

5.2 Touch Screen

Touching, one of the main five senses, is essential for humans. Humans usually feel interaction by touching other humans, animals, and even plants. One of the most important elements in touching is an engagement when you touch something different from the inanimate. Unfortunately, robots are made up of inanimate elements; plastic, irons, and so on. So users cannot get enough of an effect when they touch robots as opposed to when they touch humans, wherefore many robot scientists try to solve this problem [18]. In this project, we expected that the touch screen will be helpful to resolve this issue and for this reason we decided to use the tablet.

5.2.1 Tablet

The tablet is an inbuilt external device placed on Pepper's chest that enables users to communicate with Pepper by providing feedback by non-verbal means. It is an Android tablet where special apps could be developed to integrate with the robot. Moreover, the tablet and Pepper have an autonomous wireless connection. The tablet cannot be discussed without delving into its physical property of 10.1-inch displaying information.

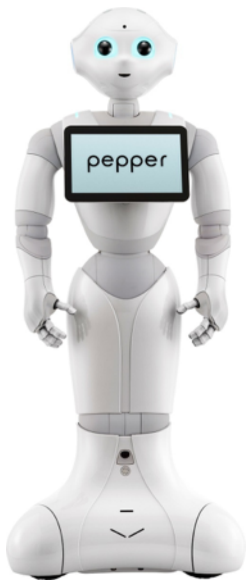
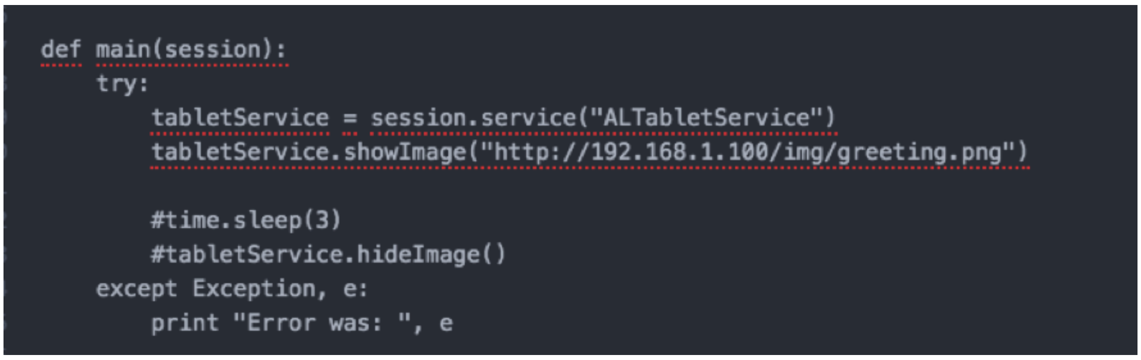


Figure 14: Tablet on the chest of Pepper

In the previous projects, the tablet has mainly functioned. For example, similar to Kiosk which is popularly used in many restaurants or cafes recently, the tablet was used to show the menu to people who come to the party and place orders. According to the users' decision, the following screen also would be different; for example, if the user chooses non-alcohol drinks, the tablet will show the option of various teas or coffee menus in order to get the user's specific decision.

As we designed the game in a way that users are given a set of objects, choose one object among them, and give Lou a clue in each round by describing the object, the tablet mainly has the purpose of showing an image of a set of objects to the users. We put various objects in one picture where each of them has more than one specific feature including colors, shapes, categories, and so on. By using the *show_image* function of ALTabletService, as you can see in the figure below, we specified the relative path of the image and preloaded the image on the tablet in order to make it display more quickly. This code needs as a parameter an URL or a path to the image to download it and returns true if it succeeds, and false if it can't ping the URL or the file is not an image. It is also possible to hide the image after a certain time with time.sleep(time) and tabletService.hideImage().



```
def main(session):
    try:
        tabletService = session.service("ALTabletService")
        tabletService.showImage("http://192.168.1.100/img/greeting.png")

        #time.sleep(3)
        #tabletService.hideImage()
    except Exception, e:
        print "Error was: ", e
```

Figure 15: Using the *show_image* function of ALTabletService

Pepper can not only load an image, but also play videos using ALTabletService::playVideo, display a web page using ALTabletService::showWebview, and react to touch events with ALTabletService::onTouchDown. [19]

5.2.2 Challenges of the Touch Screen Part

As one of the crucial goals of using a tablet in this project is to make users interact with Lou, it would be better if we diversely used the table to reinforce visual effects. For instance, the tablet could show if the answer is correct or not when users touch the object they think of as the answer. Otherwise, if the tablet shows different images as the conversation proceeds, such as it shows another image as a response, users could easily feel like they truly communicate with Lou not only verbally but also visually.

Furthermore, it is also nice if the game is implemented by using some techniques of Computer Vision such as Feature Extraction. Lou could detect features of the objects, such as colors or the position in the image, and use them to play the real guessing game. However, this might be too ambitious to finish in a semester with 4 people, wherefore we used entities of a dialog for Lou to detect the features.

5.3 Motions

When it comes to social interaction, gesture is one of the most important features which helps humans illustrate what speech alone cannot provide, e.g. to convey referential, spatial or iconic information. This is not only the case for humans; humanoid robots that are intended to engage in a natural human-robot interaction also need to produce some gestures that go along with speech for comprehensible and intelligible behavior. According to research, humans evaluate more positively when non-verbal behaviors such as hand and arm gestures are accompanied with speech [20]. In order to maximize the effect of the interaction between users and Pepper by adding some gestures, we utilized Choregraphe.

5.3.1 Choregraphe

Choregraphe [21] is a software by Softbank Robotics, the company that develops robots including Pepper. This multi-platform desktop application allows developers to easily create human-being-like motions and behaviors for the robot, such as dancing, waving, or even saying something by using the NAOqi. Choregraphe is indeed very accessible as the user does not have to code the behaviors. In the Box Libraries Panel, we could see possible behaviors and use them by simply dragging and dropping onto the Flow Diagram Panel. Also, we could see how it looks like on the screen by simulating a virtual robot in the Robot View. [22] It is also possible to add behaviors written in Python or C++, allowing for more detailed and quickly executed behaviors.

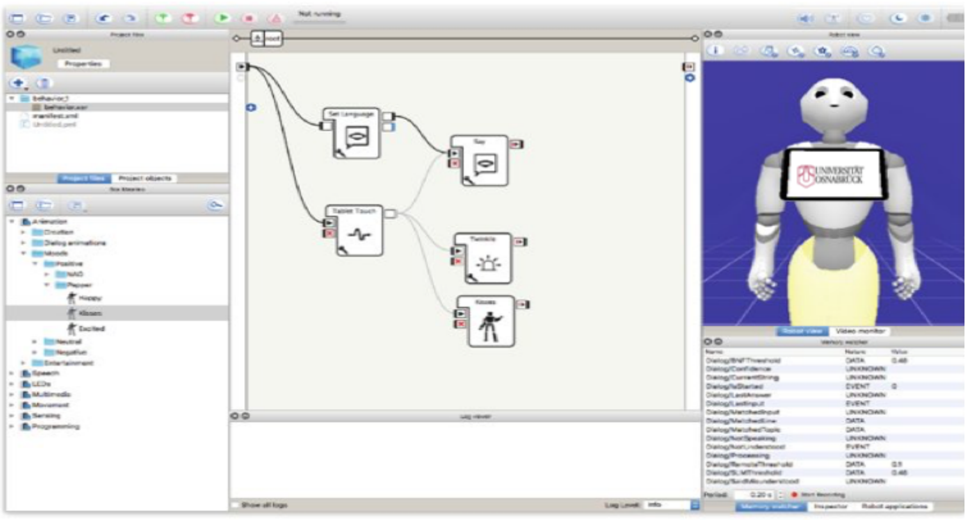
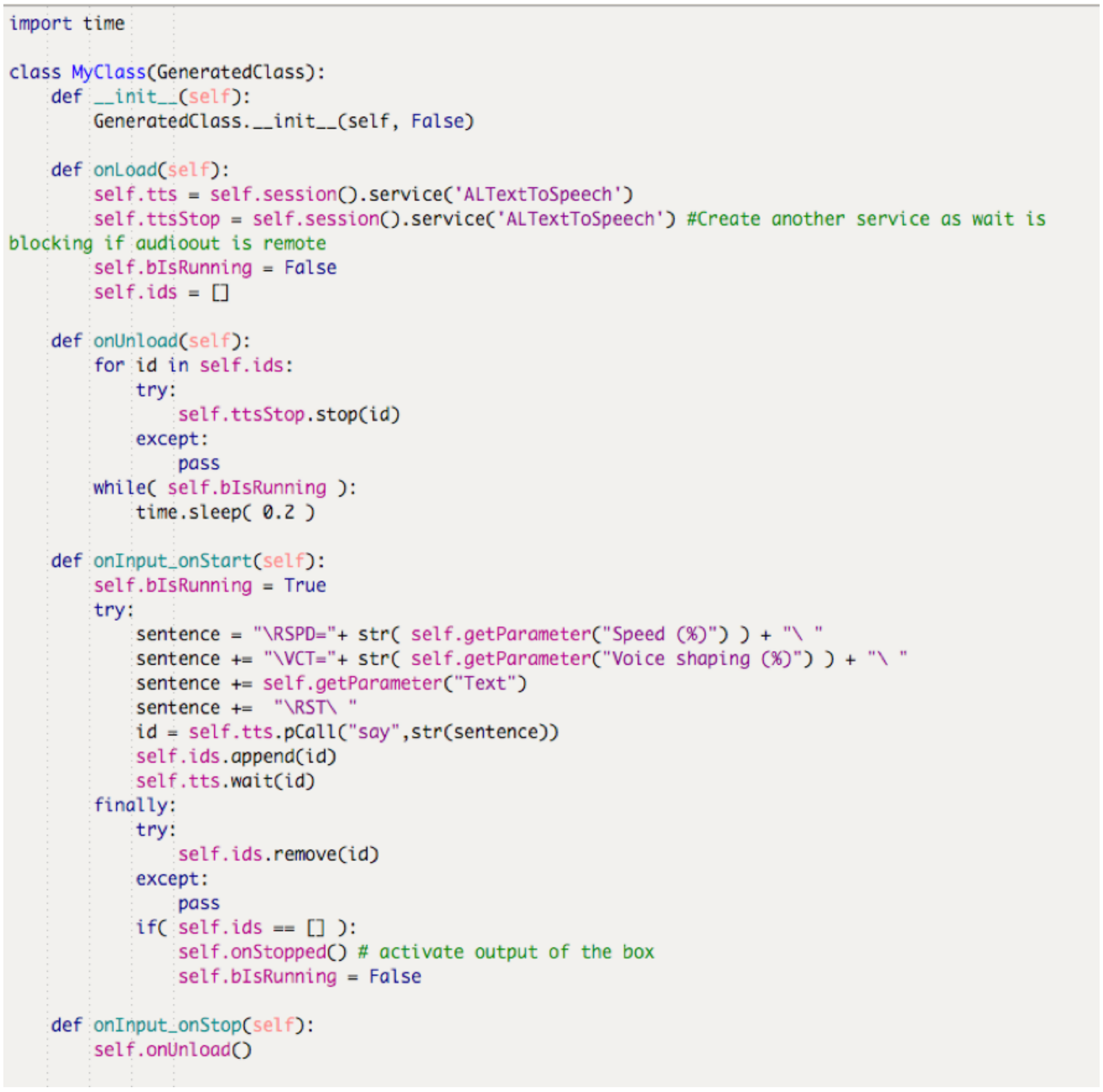


Figure 16: Box Libraries Panel, Flow Diagram Panel, and Robot View on Choregraphe

We added various gestures between the conversation according to what Lou says in order to make it look natural. First of all, before Lou shows a picture, it starts with a greeting. This can be simply done by using the Say box from the Box Libraries Panel. Such a short script can be edited on Choregraphe itself without a Python script, as you can see in the figure 17.

Here, we modify the parameter ”Text” to make Lou say other sentences that we want. Secondly, Lou moves its arm when it shows the picture of the objects. To do this, we chose Hands Movement from the Box Libraries. By setting the parameters, Lou can move



```
import time

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self, False)

    def onLoad(self):
        self.tts = self.session().service('ALTextToSpeech')
        self.ttsStop = self.session().service('ALTextToSpeech') #Create another service as wait is 
blocking if audioout is remote
        self.bIsRunning = False
        self.ids = []

    def onUnload(self):
        for id in self.ids:
            try:
                self.ttsStop.stop(id)
            except:
                pass
        while( self.bIsRunning ):
            time.sleep( 0.2 )

    def onInput_onStart(self):
        self.bIsRunning = True
        try:
            sentence = "\RSPD="+ str( self.getParameter("Speed (%)") ) + "\ "
            sentence += "\VCT="+ str( self.getParameter("Voice shaping (%)") ) + "\ "
            sentence += self.getParameter("Text")
            sentence +=  "\RST\ "
            id = self.tts.pCall("say",str(sentence))
            self.ids.append(id)
            self.tts.wait(id)
        finally:
            try:
                self.ids.remove(id)
            except:
                pass
            if( self.ids == [] ):
                self.onStopped() # activate output of the box
                self.bIsRunning = False

    def onInput_onStop(self):
        self.onUnload()
```

Figure 17: Example Script of a Box

either the left or the right hand or both as well. There are other possible behaviors such as walking, sitting down and standing up, or even applied posture. Thirdly, we wanted to show the picture of the objects on the tablet as explained earlier. We chose the Python Script box on the libraries and modify it as we wanted above. Finally, all the boxes are simply connected sequentially so as to define the sequence of action to perform.

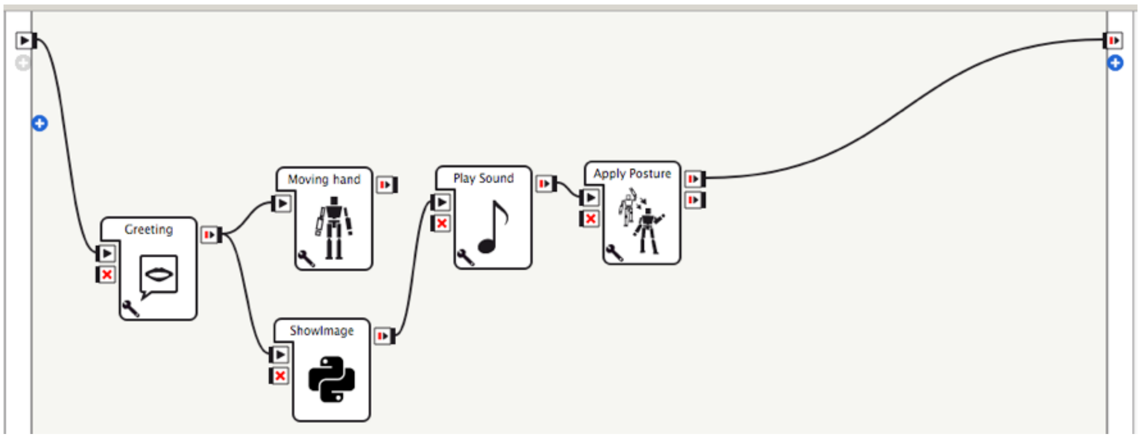


Figure 18: Connecting multiple behaviors sequentially

5.3.2 Challenges of Using Choregraphe

Adding motions by using Choregraphe was one of the last parts of our project but crucial to make it high quality. How much the motions of Lou would affect users to feel friendliness was one possible question of the usability test that we had planned. However, unfortunately, as far as the IKW building closed due to the pandemic, we were not able to work with Lou in person so we could not finalize the motion part and conduct the usability test.

As there are various functions on Choregraphe itself including Autonomous Abilities where Pepper can decide proper actions autonomously [23], we recommend those who will work on the following project to use them proactively. Pepper making its own action detecting the surrounding environment and reinforcing its action accordingly will be fitting for the original purpose of this project. It is also true that using only Choregraphe might come at the expense of making the process flexible and the behaviors are slower at execution time. Coding a Python script would be helpful to solve the problems.

5.4 Mobility

In addition to other functions and abilities, Pepper is equipped to perform the task of spatial navigation. Navigation is considered to be a useful ability for robots which can extend the range of tasks that the robot can perform. “Robot navigation” refers to the ability of the robot to move from the current position to the desired destination. More specific tasks and approaches of robot navigation include localization, motion, and path planning, as well as building a map and interpreting it. These kinds of features may be used to have the robot avoid dangerous environments or to make it perform interactive tasks.

Pepper has laser and sonar sensors as well as a depth camera to register its surroundings. It is thus possible to implement algorithms that allow it to make a map of a room and to navigate within it. However, Pepper robots only have limited mobility. Since their main function is interacting with humans, they are not built for complex navigating tasks like, for example, navigating obstacle courses or exploring complex environments.

Unfortunately, we did not use this function due to a lack of the number of members and we found that it was unnecessary for our 20Q game. If there will be a further project about this, we strongly recommend to use the Navigation function since there is a map from the last semester and there will be more possibilities when Pepper can move around the room.

6 Conclusion

When we started our project in the first semester, we designed the "I spy" game. The game starts with the sentence "I spy with my little eyes something" and color name which means the first clue would be limited to a color all the time. However, we changed our minds and decided to design the 20Q game which is more of an interactive chatting style game. Because we didn't have a team in charge of the vision part of the robot, we wanted to focus on and emphasize the conversation part. For this, the number of objects has increased from 10 to 30, the features for each object also have been added, and the first clue is not restricted to color.

We thought that this 20Q game would be suitable for the own purpose of this project, Cognitive Robotics, since we programmed the robot itself to guess an object that a user thinks by collecting and using some clues from the conversation. Not only that, but the user is also supposed to communicate with the robot and have a conversation with it which helps their socializing.

We have been facing a lot of limits to conduct this project. First of all, we have less than one-third of the number of team members in our second semester compared to the first semester, i.e. we were only four people. It hindered us to develop the game with various functions as the previous projects had done. Besides, when we finalized our game, none of us were able to be in Osnabrück due to the pandemic situation. Even though we were planning to make an educational video for explaining the game and how to use it, to gather a target group and let them play the game, and to conduct a usability test for gaining feedback; every plan had to be canceled accordingly. Furthermore, the IBM Watson Assistant which is the main workspace in our project was expired and we could not extend it beforehand.

Despite the limits, this project allowed us to learn many things. First of all, we could understand how a robot works. As it was the first time to conduct a project using a robot for all of us, we had to research not only about the robot itself but also many tools to work with the robot. Through some courses of Cognitive Science such as Human-Computer Interaction, we had learned the basics of robots theoretically, however, treating robots would be a totally different story. In order to utilize the robot effectively, we were supposed to use various tools including Naoqi API, Python SDK, IBM Watson Assistant, and ROS. By using these, we were able to learn how to make a robot human-like according to the purpose of use.

In addition, through this project, we could learn how to build up a chat-bot since the process of designing a dialogue is similar to developing a conversational agent. We reviewed and compared existing chatbot development tools and chose what was suitable for our project. As described in the Conversation part earlier, we learned the essential parts of building a dialogue: entity, intent, node, and so on. By combining them, we could set up a more realistic dialogue and prepare proper responses to unexpected input from the users.

This project also helped us to improve our programming skills, especially in Python. For example, we could understand the types of data such as a dictionary, list, and float when we considered in which type we would create values. Also, we learned the difference between a for loop and an if loop, the way of defining a function, and inserting external dataset or libraries into the code.

On top of that, finally, we tried to find the most appropriate and effective way of working to carry out a particular task as a group. In the first semester, when we were in a group of 14 people, we divided the group into three subgroups and had a weekly general meeting in order to catch up with the work of the other teams. Also, we created protocols in turns in every meeting so that if someone missed the meeting they could easily follow up. There were diverse workspaces that we have used: Github for sharing the code of each subgroup, Slack for uploading papers and notices, and Stud.IP for archiving protocols. In the second semester, when we were in a group of 4, we tried to gather all the people weekly as much as possible. Sometimes we came across personal issues or situational difficulties, but kept having virtual meetings via Skype and succeeded to finish our project.

7 Outlook

This program can be used for education and entertaining children. It will give children an exciting experience and interaction with the Pepper robot and make them more interested in robots in general. However, there were several limitations as mentioned above, so here are suggestions that can be made along with the limitations.

We were going to try to implement object detection into Pepper in the last semester, but there was an intrinsic restriction in Pepper implemented with object detection. So we decided to change our way from object detection to dialogue. The game is focused on the function of conversation, by which users can interact with pepper. But there is still room for improvement. Although we deleted this object detection function, the future study project could implement this function into Pepper so that user experience can be enhanced with extra visual effects.

Secondly, it is suggested using various methods of data analysis to develop the dialogue tree. For future study, some machine learning techniques such as decision tree or other classification methods could be used. This way more objects could be used to be guessed, and you will need less maintenance since you do not need to put each object and the corresponding connection from feature to feature. Executing a usability test and actually implementing speech recognition API could also help the next project to get much more intuition into how users with no prior knowledge feel about our program. So, for the next study, it is strongly suggested to implement the program with the actual Pepper robot and to conduct meaningful tests.

Despite the fact that Choregraphe is very user-friendly and can be used intuitively, it is rather inflexible. So if you want to make a human-like robot, you need to use a programming language so it can do a lot more various actions. We used Python to have Pepper display what we want to show and speak at the same time, and added some gestures that could attract children. However, there are much more that can be implemented by using code. So we hope that in the following project, more diverse movements and even neural networks could be used for human-like interactions.

In a real world scenario there would not only be one target voice, but rather a lot of background voices and noises are also recorded along with the target voice. In this situation the target voice must be 'attended' to, which is when attention networks come into play in speech recognition. Since Pepper always has trouble when there's noise in the voice recording, this attention based end-to-end speech recognition system would further facilitate interactions with users in the real world.

References

[1] A. Gardecki and M. Podpora, "Experience from the operation of the pepper humanoid robots," *Progress in Applied Electrical Engineering (PAEE)*, pp. 1–6, 2017.

[2] "Nestlé to use humanoid robot to sell nescafé in japan." `https://www.nestle.com/media/news/nestle-humanoid-robot-nescafe-japan`. Accessed: 2021-04-16.

[3] "Josie pepper - the humanoid robot with artificial intelligence." `https://www.munich-airport.com/hi-i-m-josie-pepper-3613413`. Accessed: 2021-04-16.

[4] M. Matamoros, C. Rascon, J. Hart, D. Holz, and L. van Beek, "Robocup@ home 2018: Rules and regulations." http://www.robocupathome.org/rules/2018_rulebook.pdf, 2018.

[5] "Ibm cloud api docs - watson assistant v1." `https://cloud.ibm.com/apidocs/assistant/assistant-v1?code=python`. Accessed: 2021-04-16.

[6] L. E. Baum and T. Petrie, “Statistical inference for probabilistic functions of finite state markov chains,” *The annals of mathematical statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[7] M. Alam, M. D. Samad, L. Vidyaratne, A. Glandon, and K. M. Iftekharuddin, “Survey on deep neural networks in speech and vision systems,” *Neurocomputing*, vol. 417, pp. 302–321, 2020.

[8] M. Gales and S. Young, “The application of hidden markov models in speech recognition,” 2008.

[9] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, “Convolutional, long short-term memory, fully connected deep neural networks,” in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 4580–4584, IEEE, 2015.

[10] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, “Listen, attend and spell,” *arXiv preprint arXiv:1508.01211*, 2015.

[11] “Google cloud speech-to-text.” `https://cloud.google.com/speech-to-text`. Accessed: 2021-04-16.

[12] A. Zhang, “Speech recognition (version 3.8) [software].” `https://github.com/Uberi/speech_recognition#readme`. Accessed: 2021-04-16.

[13] “Cmusphinx documentation.” `https://cmusphinx.github.io/wiki/`. Accessed: 2021-04-16.

[14] “Ibm cloud - watson speech to text.” `https://www.ibm.com/de-de/cloud/watson-speech-to-text`. Accessed: 2021-04-16.

[15] “[api] application program interface defiition.” `https://apifriends.com/api-management/what-is-an-api/`.

[16] “Naoqi apis.” `http://doc.aldebaran.com/2-5/naoqi/index.html`.

[17] “What is the difference between an api and sdk.” `https://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/`.

[18] R. Andreasson, B. Alenljung, E. Billing, and R. Lowe, “Affective touch in human–robot interaction: conveying emotion to the nao robot,” *International Journal of Social Robotics*, vol. 10, no. 4, pp. 473–491, 2018.

[19] “Altabletservice api.” `http://doc.aldebaran.com/2-5/naoqi/core/altabletservice-api.html`.

[20] M. Salem, K. Rohlfing, S. Kopp, and F. Joublin, “A friendly gesture: Investigating the effect of multimodal robot behavior in human-robot interaction,” in *2011 Ro-Man*, pp. 247–252, IEEE, 2011.

[21] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier, “Choregraphe: a graphical tool for humanoid robot programming,” in *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pp. 46–51, IEEE, 2009.

[22] “Box libraries panel.” `https://developer.softbankrobotics.com/pepper-naoqi-25/naoqi-developer-guide/choregraphe-suite/main-panels/box-libraries-panel#chore-box-libary-panel`.

[23] “Autonomous abilities.” `http://doc.aldebaran.com/2-4/ref/life/autonomous_abilities_management.html`.

A Team Members and Attributions

A.1 Study project members

The following list provides information on the team members participating in the Cognitive Robotics study project from April 2019 to March 2020:

1. Kim, Minhee
2. Kim, Yeji
3. Hwang, Eunyoung
4. Windler, Anne-Kathrin Patricia

A.2 Contributions of each member

Between the participants of the Study Project, the members contributing to this report are:

- for the study project (April 2019 - March 2020)
  - Hwang, Eunyoung
  - Kim, Minhee
  - Windler, Anne-Kathrin Patricia

- for the interdisciplinary course (October 2019 - March 2020)
  - Kim, Yeji

The contributions were made in the following way: In general, our team worked on most of the tasks together, especially in our first semester. In our second semester, while we also worked on many things together, the contributions can generally be described as follows: The idea of the Project Purpose and the Structure of the game came with Minhee and for the structure of the game we all worked togather. Minhee and Yeji worked on building the conversation in Watson Assistant, constructing the conversation structure and creating the dialog in detail. Minhee worked on the organization of finalizing the project and the final report. Patricia worked on the development of the speech recognition system. Eunyoung worked on the Choregraphe part. Patricia worked on the Python integration part and Patricia and Minhee kept working on smaller tasks concerning details of the Watson Assistant dialogue until the end.

Additionally, we helped each other to fix errors and gave advice when needed.

The following table shows the contributions of each team member to the written report:

Report section	Author(s)
1 Introduction	Patricia
2 Scope of the Project	Minhee
3 Conversation	
3.1 Introduction	Minhee
3.2 Watson Assistant	Minhee, Yeji
3.3 General Structure	Minhee
3.4 Dialogues	Minhee, Yeji
3.5 Python	Patricia
4 Speech Recognition	Patricia
5 Physical Functions of Pepper	Eunyoung
6 Conclusion	Eunyoung
7 Outlook	Yeji

Edition task	Contributor(s)
1 Organization	Minhee
2 Structure and Content	Minhee
3 Language	Patricia