Gordonito: The Cooking Assistant

Group 32: Isabella Venancia Gardner(i.v.gardner@student.vu.nl, 2707838), Paula Kordes (p.r.kordes@student.vu.nl, 2718839), Caspar Grevelhörster (c.m.s.grevelhorster@student.vu.nl, 2707848), Zoë Azra Blei (z.a.blei@student.vu.nl, 2695271), Antoni Stroiński (a.j.stroinski@student.vu.nl, 2673064), and Sid Singh (s.r.singh@student.vu.nl, 2708156)

Vrije Universiteit Amsterdam

1 Introduction

This project encompasses the creation of a conversational agent tasked with being a cooking assistant. The assistant's name is "Gordonito", named and modeled after the famous celebrity chef, Gordon Ramsey. The aforementioned cooking assistant was designed to help select and guide a user through a variety of recipes. Explicitly, Gordinito is an agent with the ability to conduct extensive recipe instructions through a conversation and aims to prevent communication deficiencies between itself and its user. Conversational agents are becoming more ingrained into society and people's daily lives. They provide a wide expanse of benefits ranging from convenience and time efficiency to disability support and access. Cooking is a necessity of living for most people and the task can be considerably less daunting with the help of an assistant. In this case, the cooking assistant operates in a closed-domain and task-based capacity. It must all be able to maintain a conversation of extended length. Specifically, this was perpetuated through the utilization of an instructions. Furthermore, Gordonito's system provides an audio and visual interface. This is comprised of various food-related images, and instructions. Additionally, Gordonito incorporates handcrafted dialogue management and knowledge-based on various inputted cooking recipes.

The cooking assistant exists in a multi-agent system. Specifically, the agent-collaboration is comprised of three agents, the Dialogflow agent, the Cognitive agent, and the Human agent. The Dialogflow agent is used to classify texts as intents and to recognize entities the user provides. This intent and entity processing enables information extraction for later utilization. The cognitive agent is coded using Prolog and the GOAL programming languages and is responsible for the generation of agent responses. Agent responses are produced through the use of patterns that were individually designed to mimic natural conversation patterns. Lastly, the human agent or user interacts with the other two agents and ideally conducts the actions instructed to prepare a recipe. As much as the recipe itself, there are also a variety of components necessary to aid in the success of a cooking assistant. A cooking assistant requires knowledge on its domain, conversational capabilities, and user-centered engagement processes.

There are a plethora of tools that facilitate the creation and operation of this project's cooking assistant. The languages the programs were written in were GOAL and Prolog and this is contained in the Eclipse IDE. Moreover, the conversational agent employs The Social Interaction Cloud (SIC) framework. This provides voice interaction through bot software and a visual interface with browser software. Lastly, the cooking assistant was designed and run with Dialogflow and the GOAL SUPPLE Dialog Engine. The latter is a handcrafted tool built on the foundations of conversational patterns and information state updates. This is what the project's cognitive agent is comprised of. Google Dialogflow handles automatic speed recognition, speech synthesis, and information classification.

This project was divided into three sections Recipe Selection, Recipe Instruction, and Visual Support. Recipe Selection enables the bots greeting, and allows for the selection of recipes by a variety of features like name, country of origin, speed of recipe, and ingredients. Additionally, an ingredients check and utensil check was implemented to permit the user to certify their possession of the necessary materials to complete the selected recipe. Visual support integrates various displays for the recipe

name, buttons, feature representation, ingredient, and utensil representation, and additional instruction support displays. Finally, Recipe Instruction enables the proper provision of recipe instructions, clarifications, endings, conversational closings, conversation reparations, capability checks, appraisal enablement, and switching between recipes.

This report is comprised of an overview and description of the SUPPLE Dialogflow engine, the design choices and rationale implemented in the creation of Gordinito, and an evaluation of the conversational agent provided.

2 SUPPLE Dialogflow Engine

The problem of dialogue generation for conversational agents has been a topic of discussion for years. Two diverging interface approaches are automatic training and handcrafted interfaces. Trained interfaces learn to handle dialogues through exhaustive analysis of past conversations. While this minimizes human labor involved, problems arise regarding the naturalness of the resulting agent. Specifically, the miscellaneousness of training data can lead to recognizably inhuman conversation management; combining experience from conversations held by varying persons or agents becomes manifest in the frequent switching of character traits throughout longer dialogues. Handcrafted dialog management on the other hand mitigates this drawback; a thorough design process that primes the interaction manually can self-evidently feel as natural as conversing with a human. The main downside of this application, however, is the limited operational area of resulting agents; while a vast spectrum of conversations can be learned and understood through training, manual preparation requires a formidable expenditure of work. Thence, handcrafted dialogues necessitate a narrow application. For the deployment of an assistant in the realm of cooking, the disadvantage of crafted design, namely the limited scalability, is somewhat irrelevant due to the inherent narrowness of the topic. Since the content can be condensed to a simple dialogue, the latter be carefully prepared.

At the core of conversational agents lies the question of dialogue management, to wit, choosing the system used for keeping track of the past, present, and future of the current interaction. Dialogue management is commonly handled (see Google Assistant, Amazon Alexa, Apple's Siri) on the basis of information-retrieval approaches i.e. Intent-Entity-Context-Response (IECR) paradigms [1]. Nonetheless, this approach is limited in longer coherence, tracking of sequence expansion as well as scaling. A solution to these shortcomings is the Sequence-Update pattern-based Processing with Logical Expansions (SUPPLE). Common to both IECR and SUPPLE is the utilization of a pattern language that breaks down a conversation into sub-conversations, so-called patterns. Together, these patterns form a versatile structure that can be easily followed while remaining adjustable to the user's needs; they can be thought of as interchangeable building blocks, each handling a different part of the interaction. Since this approach breaks down the conversation into partitions, expanding parts of the conversation followed by returning to the central theme is straightforward. This logical expansion process is the principal concept of SUPPLE, the implemented engine structure (see fig. 1) of the cooking assistant discussed in this paper. Given a knowledge base storing the required patterns [patterns.pl] and a limited set of rules, the SUPPLE dialogue engine performs dialogue state tracking (DST) and dialogue act selection (DAS): Whenever a user or the agent say something, these two phases are working on handling this input. First, the engine receives a perceived intent with an optionally attached entity from the natural language understanding engine. Next, the DST component recognizes in what way this perceived entity is in line with the currently followed pattern. It does so on the basis of the following rules:

Upd-1: Is what was said in line with the current pattern?
→ Yes: follow the pattern ("contribution")
Upd-2: Is the intent start of another pattern p_a and does not fit the current?
→ Yes: expand current pattern with p_a if possible ("sequence expansion")
Upd-3: Does the intent not fit neither the current nor expandable patterns?
→ Yes: start repair pattern ("out-of-context repair")

Once the state has been tracked and it is decided which pattern is to be followed, the DAS component *dialog_generation.mas2g* recognizes which act needs to be performed next. This happens based on the following rules:

Gen-1: Is information missing to complete the step?
→ Yes: ask for that information ("slot-filling")
Gen-2: All information is given and the next step in the pattern is available?
→ Yes: act out the next step ("pattern-based selection")
Gen-3: The current pattern has been finished?
→ Yes: act out the first step in the pattern that is next on the agenda ("agenda based selection")

To understand the process from a practical standpoint, an exemplary program iteration will be explained step-by-step. In the following situation, the agent is already initiated; the user is greeted, a recipe is chosen, utensils, as well as ingredients, have been checked and the next step of the chosen recipe is being displayed. The agent expects input from the human user. The phrase *"I would like to make another recipe"* is passed to the interface by text or voice input. Firstly, this input phase is processed by the natural language processing tool (Google Dialogflow) and classified through natural language classification. Given certain predefined training phrases, Dialogflow classifies this input with as *intent = "requestDifferentRecipe"* and recognizes that no entities were mentioned. This information is processed by the SIC environment which transforms the information into a percept that is passed on to the GOAL agent as *percept(intent(requestDifferentRecipe, [], _, "I would like to make another recipe", _)).* This percept is received by the DST component (dialog_update.mas2g). As previously mentioned, in this part, the session is updated according to the ongoing conversation and a set of decision rules (in this case contribution):

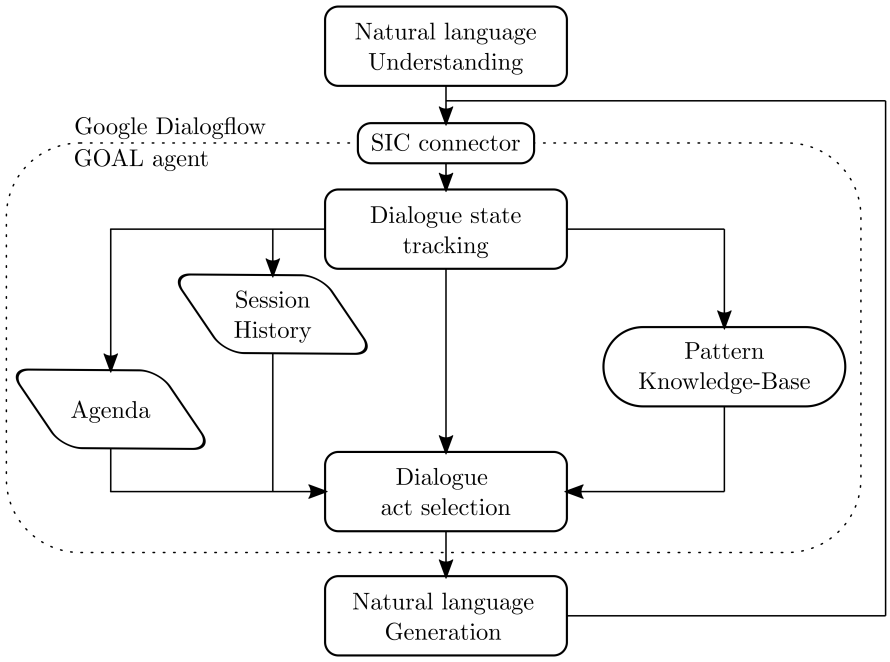
```
session([[a30recipe, [agent, recipeConfirm, []],[a30recipeStep,
[agent, aGordonQuoteinsult, []],[agent, recipeStep, []],[user,
requestDifferentRecipe, []]],[a50utensilsCheck, ...]])
```

This updated session is passed on to the *dialog_generation.mas2g*-module which interprets the new session information. Since there is no slot filling required, nor are there new agenda instructions, and the current session does not start with an empty list [], another case (Gen-2) is considered: the next agent action in the *a30recipeStep* pattern, namely the intent of *chooseOtherRecipeAcknowledgement*. The latter is passed back to the *dialog_update.mas2g*-module which updates the session knowledge again:

```
session([[a30recipe,[agent,recipeConfirm,[]], [a30recipeStep,
[agent,aGordonQuoteinsult,[]],[agent,recipeStep,[]], [user,
requestDifferentRecipe,[]],[agent,
chooseOtherRecipeAcknowledgement, []]], [a50utensilsCheck, ...]])
```

For the last time, the session is forwarded to the DAS (*dialog_generation.mas2g*) module which queries the respective predefined text for the intent in the knowledge base (specifically *dialogue.pl*). The text is sent through the SIC connector to the natural language generation of Google Dialogflow. After finishing the voice generation, the agent is ready for new input.

Fig. 1. Structure of the SUPPLE Dialogflow engine

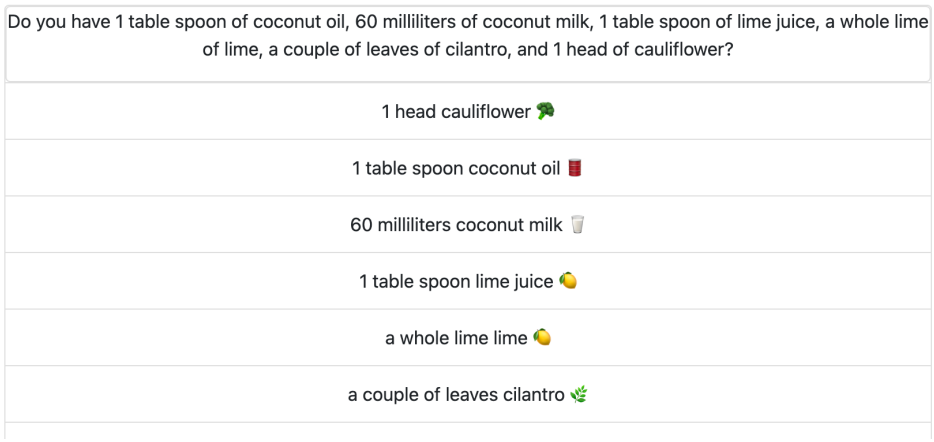


3 Design Choices and Rationale

First, it was collectively decided upon the personality the bot should display. Here, it was chosen that the bot should resemble Gordon Ramsay, a prominent British chef known for his bluntness and characteristic outbursts, hence, the name Gordonito. To achieve this, the bot was modeled to have a British accent, and some quotes of his were added to the bots' vocabulary. This also means that explicit language is used throughout the bot. Due to tool constraints, it was in this case not possible for the bot to speak in a male voice, although this would have been preferable. For recipe selection, the dialogue flow was designed to start with a greeting (pattern `c10greeting`) that the user should answer. Next, the bot would bluntly ask the user to choose a recipe using the `a12chooseRecipe` pattern. While in the first instance, the recipe feature selection was modeled to only be able to handle a single feature that was uttered by the user, it was managed to create a more general rule for the pattern, enabling multi-feature selection. The features by which the recipe database could be queried were chosen to be ingredients, country of origin, and quickness. Quickness hereby means that a recipe has ten or fewer steps. If no recipes match the query, the user has the option to request a `'restart'` and begin the selection progress again. When a user has selected the features he or she desires, a list of all recipes that fit that description will be displayed. It was decided that the bot only reads the contents of the list if the list is short (< 200 characters) because reading out a long list makes the bot seem very unnatural. For long lists, the bot simply asks the user to check the contents of the list. The user can then choose a recipe. Alternatively, if the user already knows what exact recipe they want to make, they can just tell the bot immediately when asked to select a recipe. Once a recipe is selected, the bot will register it via a choice receipt. Next, the `a50ingredientsCheck`, and subsequently the `a50utensilsCheck` patterns will be activated. Respectively, the user will be asked whether they have all ingredients/utensils out of a displayed list. This makes the user have a clear overview of everything

that is needed to cook the selected recipe, opposed to the ingredients and utensils being displayed in a text where the user has to extract the information first. To make the ingredient list visually appealing, it was decided that the ingredients displayed in the list should be accompanied by a fitting emoji (Figure 2). The same approach was originally envisioned for the utensils list; however, there are currently not enough utensil emojis in the *UTF-8* character set to allow for this to be implemented.

Fig. 2. Example of ingredient list



If a user does not have one of the items necessary for the recipe, the bot will redirect them to choose a different one. If they have everything, the bot will continue to `a30recipe` and begin to assist the cooking process. Hereby, every step in the recipe is explained to the user using a combination of speech and a picture that displays the current step of the recipe (Figure 3). The pictures allow the user to see if they are on the right track by comparing their progress with the displayed picture at each step.

Fig. 3. Example of instruction in recipe

Design choices regarding the recipe instructions included giving the user clarification of a step whenever it was asked for. For some recipe steps there were no clarifications provided, therefore, an alternative pattern was created, repeating the original instruction. This was done to show that the agent understood the user correctly, and did acknowledge the need for clarification. Furthermore, due to the functionality of the provided code, sub-patterns were not exited automatically. When creating patterns that expanded the conversation during the recipe instruction steps (i.e. `c50stepBack`), it would have been unclear to the user how to proceed with the recipe. To make the dialogue more intuitive for the user, the new implementation provides an explicit instruction on how to reconvene with cooking. Regarding entities, some of the utensil names were adjusted both in Dialogflow as well as in Gordonito's code to make them sound more appealing. When asking if a knife must be used for a certain recipe, the agent replies with "Of course, we need a massive Excalibur." Lastly, for making the interaction with assistant more convenient, both a talk button as well as a text input field were implemented. This gives the user the choice of using either input method.

4 Evaluation

4.1 Testing

For testing the capabilities of Gordonito, two people that were not familiar with the project nor with the cooking assistant were used. Before starting the test, the test participants were informed of possible text they could either say or write depending on what they preferred and at what point in the dialogue they could say this. While testing the assistant, no further information or help was provided by the project members. Additionally, as a third test participant, one of the project members tested the cooking assistant. User 1 and user 2 in Table 1, presented below, are the two test participants that did not have previous knowledge of the assistant. User 3 was one of the team members.

In the column "Capabilities," the instructions the test participants were informed to test are given. In the rest of the columns, a green check is placed for when a capability fully worked, and a red cross for when the user was confused with how to operate or the capability simply did not work at all. As seen in the table, the results of user 1 include one check for capability 9. When requesting another recipe, the user was confused with how to do this and what to ask for specifically. Eventually, the user managed to switch to a different recipe, after saying a sentence that was recognized by the agent. The confusion of the user could have been prevented by adding more differently structured sentences to Dialogflow. This way, the agent would have had a richer vocabulary to work with, and, thus, it would have made it easier for the user to operate the agent. The second user had difficulties with capabilities 11 and 14. The issue user 2 had with capability 11 was that the user wanted clarification on how much "olive oil" he needed to use, however, "olive oil" was not recognized by the agent. This was due to the fact that the "ingredients" entity lacked ingredient synonyms for "extra virgin olive oil." The issue with capability 14 was that the agent simply did not go a step back after user 2 confirmed that he wanted to do so. After testing the cooking assistant, user 2 suggested that he would have preferred more information on what to say to the assistant to go to the next step in whichever part of the recipe he was in. There happened to be several instructions that did not clarify that the user had to say 'next' in order to proceed with the recipe. Lastly, for user 3, the project member, all capabilities worked. Taking into account that user 3 already had previous knowledge while testing, and, additionally, took part in coding the cooking assistant, it was expected that operating the agent would cause no confusion nor difficulties for user 3.

Overall, the agent performed well, considering the fact that with every test participant there were no more than two failures. Additionally, it is significant to note that the occurring issues were regarding the lack of vocabulary of the agent.

4.2 Improvements

Gordonito has a lot of room for potential improvements. Since the machine is interacting with an unpredictable human there is always a place for adding more patterns to make the bot even more Gordon-like. To make the interaction more smooth a few features have been added to the bot.

The first and the most noticeable improvement was the addition of pictures to each step of the recipe (for the three chosen recipes). This was achieved by creating a knowledge base of

```
picture('recipe name', step, 'picture url').
```

For each step of the recipe, the belief base was updated and the new URL inserted. The HTML pages have been edited in such a way that the picture source has become a variable that was filled in from the belief base.
The second improvement was to ensure correct usage of the restart pattern. Additional belief states had to be reinitiated to make sure that the Gordonito bot works correctly. The next improvement was the most human-like, namely a give-up pattern allowing the user to quit the recipe instructions at any time. Gordonito will then insult the user and suggest ordering food from a supermarket or a restaurant. To make it more interactive, an additional function has been implemented, making sure that the link is going to be properly rendered and not read out loud by the bot. To achieve the robustness of Gordonito, a 200 characters limit has been implemented. This was necessary because a message that exceeds this amount of characters crashes Google Dialogflow, resulting in the bot not responding. Next to this limitation, the automatic exit of finished sub-patterns was also not feasible to implement since the provided connector did not allow for this change. The name of the bot comes from the famous Gordon Ramsey thus every text has been replaced by Gordon-like language. Moreover, at every 3rd step, the user is insulted by Gordon himself. To make it even more realistic the voice has been changed to the British one. To make the user experience more visually pleasing, emojis have been used to reflect the ingredients. To the recipes.pl file the predicate

```
\centering
emoji("ingredient", "emoji code").
```

```
\centering
createEmojis(Amount, Ingredient,Emoji):-
            emoji(Ingredient, ID),
            atom_concat(Ingredient, ID, Tmp),
            atom_concat(Amount, " ", AmountSpace),
            atom_concat(AmountSpace,Tmp,Emoji).
        createEmojisList([(Amount,Ingredient)],Results):-
            createEmojis(Amount, Ingredient,Emoji),
            append([],[Emoji],Results).
        createEmojisList([(Amount,Ingredient)|Tail], Results):-
            createEmojis(Amount,Ingredient, Emoji),
            createEmojisList(Tail, NewResults),
            append([Emoji], NewResults, Results).
```

The last improvement was an additional pattern allowing the user to take a step back in the recipe.

Gordonito is a useful assistant in the kitchen. This especially holds for the three recipes for which more specific instructions are available, and where the steps are underlined with pictures. The user can also ask for a more elaborate explanation of the steps. But also for the other recipes, Gordonito can be of help. One of Gordonito's strong suits is the dynamic recipe selection process, whereas this exceeds the level to which a human user would be able to filter the high amount of recipes for those that exactly match their specifications. Additionally, the user has several opportunities to retract to a previous situation or change the recipe mid-instruction. However, Gordonito also has some shortcomings. For one, Gordonito faces the typical obstacles innate to hard-coded bots, which is that some situations

may not have been accounted for in the code. Therefore, there might arise situations in which the bot does not understand the users' requests, no matter whether the user rephrases or not. To improve the bot, it would also be helpful to enable the user to type in the text field without having to wait until the bot finishes what it is saying. The solution to that problem is unclear as of now. Another improvement may be to implement dietary features, such as whether a recipe is dairy-free, vegan, or vegetarian so that the recipes can be tailored to the needs of the user even more narrowly. These changes could be implemented in the same way in which the countries and the rules they are queried by, are specified for each recipe. Additionally, the bot could ask for dietary restrictions at the beginning, which could be registered and used to filter out recipes that have ingredients the user is allergic to. For this, a dietary restrictions inquiry pattern would have to be created in a similar fashion to the feature inquiry, whereas instead of features possible dietary restrictions would be specified in Dialogflow. Then, these restrictions would have to be added to the feature selection rules as for example that the recipes cannot have peanuts as an ingredient. Also, different ingredients could be categorized under one entity in Dialogflow, in order to enable the user to search for recipes that have a specific food group (such as having an entity 'meat', under which all types of meats are listed). Most importantly though, user-to-bot communication during the recipe instructions should be improved. This could be done by adding more ways in which a user can ask for help, adding more instructions for the recipes, as well as implementing functions to explain specific techniques that occur frequently in the recipes, such as for example how to correctly peel potatoes or boil rice. Lastly, a children-appropriate mode could be implemented by copying the files and changing the language to be less explicit. The user could be asked which mode they prefer at the beginning of the program, whereby the files used will be dependent on the answer.

Table 1. Testing results on different test participants

	Capability	**User 1**	**User 2**	**User 3**
1	say 'hi' in different ways	✓	✓	✓
2	ask 'what can you do' at any point.	✓	✓	✓
3	say recipe you want to cook using recipe features	✓	✓	✓
4	say 'thank you' at any point in the dialogue	✓	✓	✓
5	say 'yes' in different ways	✓	✓	✓
6	say 'no' in different ways	✓	✓	✓
7	say 'i give up' at any point	✓	✓	✓
8	say 'done' or 'next' in different ways	✓	✓	✓
9	request another recipe	X	✓	✓
10	ask for clarification at any step	✓	✓	✓
11	ask how much of an ingredient is needed	✓	X	✓
12	ask if a specific utensil is needed	✓	✓	✓
13	ask to start over / cook other recipe	✓	✓	✓
14	ask to go a step back at any point	✓	X	✓
15	say 'bye' in different ways	✓	✓	✓

5 Conclusion

In summation, this project culminates into a natural conversational agent that can guide a user through the preparation of a recipe. It utilizes knowledge provided in recipes.pl about the recipe's ingredients, utensils, steps, speed, and country of origin. There were various design choices and rationale employed in order to try and emulate inter-person conversations and be as helpful as possible. For example, the user has the option to type or talk to Gordinito making it deaf- and blind-accessible. This project

progressed in multiple parallel threads that centered upon recipe selection, recipe selection, and visual support. The process utilized different platforms, languages, and tools like DialogFlow, the SUPPLE Dialogflow engine, GOAL, and Prolog. Within DialogFlow intents and entities were established to analyze and input the user's requests and phrases. Prolog supplies the knowledge necessary for the agent to conduct itself. GOAL maintains the belief, memory, and rule system. Furthermore, the SUPPLE Dialogflow engine makes it possible to natural language process and converse in this context. The process began with the selection of three recipes to fledge out. While instruction can be provided for 59 recipes the complete visual and instructional support is optimized for three: hot chocolate, winter greens salad, and cauliflower rice. Then the bot's greeting and farewell were implemented. Initially in the program, a recipe can be selected by name or a combination of features. Subsequently, Gordinito ensures the user has the necessary materials to complete the recipe. Next, the user is led through the recipes instructions. Finally, the bot asks the user if they require anything further and bids the user a farewell. The project's conclusion lead to the testing and evaluation of the cooking assistant. The bot was tested by selecting and cooking various recipes using elaborate feature selections and utilizing clarification functions. This assisted in the formation of the bot's evaluation. Some limitations the bot could be seen to possess mostly are centered upon possible weaknesses it has in conversational skills. Its conversation patterns are hardcoded and thus comprehension could be lacking and the natural fluency of human-to-human interactions. Gordinito does however have a relatively dynamic recipe selection process through features. It provides opportunities to restart and acquire further clarifications and assistance. Moreover, further improvements could always be implemented. The benefit of using a robotic cooking assistant is the ability to filter and find recipes based on a wide variety of specifications. Thus, further features could be for example dietary restrictions. Hence, in conclusion, a conversational agent can become a significantly useful tool to assist in the cooking of recipes.

References

[1] Robert J. Moore and Raphael Arar. *Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework*. New York, NY, USA: Association for Computing Machinery, 2019. ISBN: 9781450363013.

A Appendix

A.1 Weekly Report 1

The first thing we did was to have a call to look at the instructions together. Then we already decided to form the groups of two. Isabella and Paula do the recipe selection part, Zoë and Caspar work on the recipe instructions and Antoni and Sid create the visual support. We installed the parts necessary for running the environment. Then, every group worked on their own to complete their task for week 1. We decided to go for the recipes for zucchini noodles, the winter salad and cauliflower rice.

Recipe Selection. This week we gave the agent the ability to use greetings and the ability to self identify. This was done by adding intents to the Dialogflow agent and greeting patterns to the pattern.pl file. We created rules as to when to use these greetings in the dialogue flow generation file. Secondly, we enabled recipe selection. This was done by making a recipe entity in Dialogflow and adding the intent recipe request. In patterns.pl we added the a50 pattern to help the bot process and identify recipe names. Some issues we encountered were finding out where to place what code in the beginning. Also, we had to get into pulling and pushing code on GitHub. For the features for selection we decided on optimizing for name, ingredients, and country of origin

Recipe Instruction. During this week, we also worked on recipe instructions. We enabled the ability to greet the user at the end of the recipe instructions, as well as ask the user about further questions. For this, new intents needed to be created in Dialogflow and the patterns.pl file needed to be updated. We did both and everything worked except that upon restart, the agent would only repeat the last instruction instead of the whole recipe. This was an issue in the restart function contained in the dialog_generation.mod2g file: when restarting, the step count would remain at -1 which made the agent repeat the last instruction instead of starting from the beginning. After adding deletion and insertion functions, the agent restarted at the beginning of the recipe. We also enabled the agent to say bye or acknowledge the user by adding different hard-coded text snippets to the text.pl file. Generally, this week's instructions were easy to understand and the implementation was doable

Visual Support. During this week we have implemented a simple starting page that displays three pictures of the recipes we have chosen. The user has an idea of how the dishes will look at the end. We also gave a simple function that allows us to render the page based on a recipe chosen by the user. It still has to be connected to the dialog flow.

Action Steps for Week 2. Follow the instructions for week 2.

A.2 Weekly Report 2

Recipe Selection. For recipe selection this week we enabled the selection of a recipe by one feature. We chose the country of origin. We did this by adding an intent to Dialogflow, adding some information on the country of origin to our three recipes, and writing a rule to find all recipes with that country of origin. Next, we enabled an ingredients check. Before starting recipe instruction the bot helps the user check to see if they have all the ingredients. We added entity ingredients to Dialog Flow. We added a rule so the agent can terminate if the user did not have an ingredient necessary for the chosen recipe. We also added intent to our Dialogflow bot for confirmation and ingredients check. We had some issues figuring out how to make it to where the recipe selection process by features replaces the bot just asking the user for a recipe and deciding what order they should be in. Lastly, we enabled a utensils check to make sure the user has all the utensils necessary. We added a utensils entity to the DialogFlow bot and followed a similar procedure to ingredients check.

Recipe Instruction. For recipe instruction this week we started off well. However, after we thought we managed to make everything work, and finished the tasks for the week we found out that there was a problem regarding the *stepCounter*. When changing from the initial pasta recipe to a new one (which was one of the tasks) the *TotalSteps* was fixed on 3, and thus, only 2 steps of any recipe would show before arriving at the final step. This was not correct since each of our recipes contained at least 6 steps. We fixed this problem by coding our own way to retrieve the total steps of any recipe that the user desires to cook. While coding this we figured out that the *currentRecipe* function is case sensitive and would simply not match with our 'zucchini noodles' recipe, because it would only match with lowercase letters, and the zucchini noodles started with an uppercase Z. First, we fixed this problem by simply changing the recipe name in Dialogflow to start with a lowercase letter, after which, we decided to just choose a different recipe that by default starts with a lowercase letter; hot chocolate

Visual Support. This week we started by getting the pictures for the recipes to be rendered in the correct dimensions, opposed to them being stretched, by adding the correct css-lines. Now when the user chooses a recipe the correct picture in the correct format will be rendered while the ingredients and instructions are given by the bot. We also used this week to make the user interface more visually appealing. We tried doing so by making the background black and font white; however, we quickly realized that this is not possible without major changes. We then settled on keeping the background white and adding a banner in the header to make the page more appealing

Action Steps for Week 3. In the next week we want the bot to be able to handle the users questions, for example asking for help. Furthermore, the agent should be able to select a recipe through multiple features. We will also work on the displays during the recipe selection by multiple features.

A.3 Weekly Report 3

Recipe Selection. This week, we started by thinking about how we could filter for each feature. Since we already created the inquiry for the country of origin last time, we did not need to implement it anymore. Also choosing a recipe by its name was straight forward. However, we had issues getting the ingredients inquiry running. This was partly due to the fact that we only discovered that most *ingredients()* specifications only had 3 instead of 4 arguments after we had already created a working rule for 4 arguments. In the process of designing a rule for 3 arguments, we somehow lost some progress that we made and we currently do not have the recipe inquiry working. Also the *a70featureSelection* pattern that we wanted to implement does not work yet, and we also are debating whether we should implement it or just give the option of choosing by feature in a general *recipeInquiry* pattern. We also have yet to implement error mitigation rules, for when there are no recipes for the users query. We added all intents and entities for this week.

Recipe Instruction. For recipe instruction this week we enabled the user to request a clarification at any point during the recipe steps. We also realized that in the recipes.pl file, not every step of our recipes had a clarification predicate. We changed that by adding them since we don't want the agent to ignore a clarification. Next to that step clarification, we added an ingredients clarification function. This will respond to the user by telling them that a certain quantity of an ingredient will be needed for the recipe. Additionally, the user will now also be able to ask whether a certain tool is required to cook the recipe or not. Here, we wanted to be original and added "special names" to DialogFlow. For example when asking for a "knife", DialogFlow will interpret this as "Excalibur" and return that the latter is or is not needed. Further, we enabled the collection of missing information. DialogFlow might return that a user has not specified which recipe was selected or that no tool was mentioned. Through slot filling, the agent is now able to ask the user for missing information in this question.

Lastly, we made the agent understand “thank you” or other appreciation clues and reply to them before proceeding with the rest of the instructions.

Visual Support. We started this week by making the pages more visually appealing. In order to do so, we added a function that adds one or two emojis per ingredient. This list is then used in the ingredients page and shows the user the ingredient as text and as an emoji. Furthermore, we worked on the pages for the recipe selection through multiple features for example country.

Action Steps for Week 4. Next week we will think of two ways of improving the bot. We will also try to bring the three parts of the project together and have a fully functioning bot.