LAMK
Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

CHATBOT

A digital assistant with built-in AI.

LAHTI UNIVERSITY OF APPLIED SCIENCES
Information Technology
Spring 2019
Felipe Gadea Llopis

Abstract

Author(s)	Type of publication	Published
Gadea Llopis, Felipe	Bachelor's thesis	Spring 2019
	Number of pages	
	49	

Title of publication

CHATBOT
A digital assistant with built-in AI.

Name of Degree

Bachelor of Information Technologies.

Abstract

The objective of the thesis was to build a digital assistant with built-in AI. A study was carried out about which system can be used and how. Investigate about AI was carried out, as well as the possibilities to integrate it to the digital assistant.

The reason for developing a digital assistant was to provide an open source digital assistant with a custom experience to the users. This digital assistant is focused on the cities of Alcoi (Spain) and Lahti (Finland), and, by that, the user obtains a better use experience compared with the use experience provided by the traditional digital assistants in these concrete paces.

The key findings of this research were, that it is possible to build the desired digital assistant but, to obtain an advantage over the traditional digital assistants, a lot of resources and time are required. The conclusion was, that bringing to users a better treatment in specific places, Alcoi and Lahti in this case, comparing with the big businesses (Google, Apple, Microsoft and Alexa) is completely possible.

Keywords

AI, Artificial Intelligence, Chatbot, personal assistant, digital assistant, Unity, Wit.ai

CONTENTS

LIST OF ABBREVIATIONS AND TERMINOLOGY

Digital assistant: term for a small, mobile, handheld device that provides computing and information storage and retrieval capabilities for personal or business use, often for keeping schedules, calendars and address book information handy.

Chatbot: synonymous of “digital assistant”. In all the place where is written “chatbot” it means “digital assistant”. In this document, “digital assistant” and “chatbot” mean the same.

Script: program or sequence of instructions that is interpreted or carried out by another program rather than by the computer processor (as a compiled program is).

Method: piece of code associated with a class or object to perform a task (Part from the script).

Cloud Platform: A Cloud service is any service made available to users on demand via the Internet from a cloud computing provider's servers as opposed to being provided from a company's own on-premises servers. Cloud services are designed to provide easy, scalable access to applications, resources and services, and are fully managed by a Cloud services provider.

Unity: A cross-platform game engine.

Partial class: A partial class is a special feature of C#. It provides a special ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled.

Coroutine: It is a computer program component that generalize subroutines for non-preemptive multitasking, by allowing execution to be suspended and resumed.

C#: C# is an object-oriented programming language from Microsoft that aims to combine the computing power of C++ with the programming ease of Visual Basic.

JSON: JavaScript Object Notation. It is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).

Deserializing: Process of decoding a Json file. It refers to the process of transforming a representation of an object that was used for storage or transmission (Json file) to a representation of the object that is executable.

NLP System: It is a system that implements a Natural Language processing. A system is a set of things working together as parts of a mechanism or an interconnecting network. And a NLP is a subfield of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

Entity: Known as Named-entity recognition (NER), it is a subtask of information extraction that seeks to locate and classify named entity mentions in unstructured text into pre-defined categories such as the person names, organizations, locations, medical codes, ...

1 INTRODUCTION

Artificial intelligence has received much attention in recent years since it offers a huge range of possibilities. The artificial Intelligence can be used in almost everything. For example, face recognizing, pattern recognizing, deep fakes, scientific problems, medical statistics, etc.

One of artificial intelligence remarkable uses is in Natural Language Processing field. It includes, among others, voice recognition and language structure analysis.

In that field, the digital assistant applications appear. They help in commons daily tasks. These digital assistants usually work with voice commands and they can help in a variety of possibilities. For example, finding some place in the map, searching online, adding tasks to a schedule, calling someone, etc. Basically, they try to simplify daily tasks with basic voice commands saving time for the user.

Along this document, a digital assistant is developed. In the process, NLP system's technologies and the benefit-cost influence on the development are studied.

The main objective of this project is to program an application which offers a better service than other applications like Google Assistant or Siri. To achieve this, the point is to identify the deficiencies of the existing digital assistants and to implement them on this application.

The main limitation of the project is the accessibility of the relevant technologies. The systems required to power Artificial Intelligence applications are usually very expensive. Because of that, their use is generally sold as a service, and free versions have a server request limit that restricts their potential as mainstream applications.

Seeing how focusing the attention in concrete places can improve significantly the performance of this kind of applications instead of attending to the wider range of places is one of the cases of study on this document.

2 CURRENT TECHNOLOGIES

Nowadays, the technological market around Digital Assistants is fully led by four big business, Google, Apple, Microsoft and Amazon. Some other businesses are developing assistants as well but, their assistants are not as complex as the big companies. These other businesses are developing digital assistants which cannot play against the big businesses' because their performance is lower. The big ones are Google Assistant (Google), Siri (Apple), Cortana (Microsoft) and Alexa (Amazon).

These assistants are very good in their job, and they could be considered general-purpose assistants, but they have differences. Each one of them do something better than the others.

The most popular Digital Asistants are designed for general-purpose use. General-purpose use generates problems in the development because a huge range of possibilities must be taken in account. Because they must response correctly in a huge range of possibilities, some deficiencies are generated in short term view. This chatbot is focused on solving these deficiencies.

For example, you can have more detailed information about one place where the amount of population is not enough for generating interest in the big businesses, like a small city or a town. For them, the benefits of focussing on concrete places are not enough relative to their benefit-cost expectative.

Focussing on what each of them does better than their competitors, Google Assistant could be considered the most balance Digital Assistant between its competitors. It is really good in almost all the general categories used to evaluate them. (Gene Munster, 2018)

In Automatic Online Evaluation of Intelligent Assistants (2015), how to evaluate the quality of digital assistants and how the Digital assistants work in different scenarios are explained. The metrics Execute, Confirm, Question, Option, WebSearch, Error, NoAction, Comand, Yes/No, Answer and Select, are referenced in the previously cited document and, they can be use as "How many orders finishes with" for evaluating the Digital assistants.

With these metrics, which Digital Assistant is better than the others can be identified. The metrics are used in the testing part of the thesis to evaluate the chatbot. Finally, this graphic (Figure 1) shows almost currently which assistant is better:

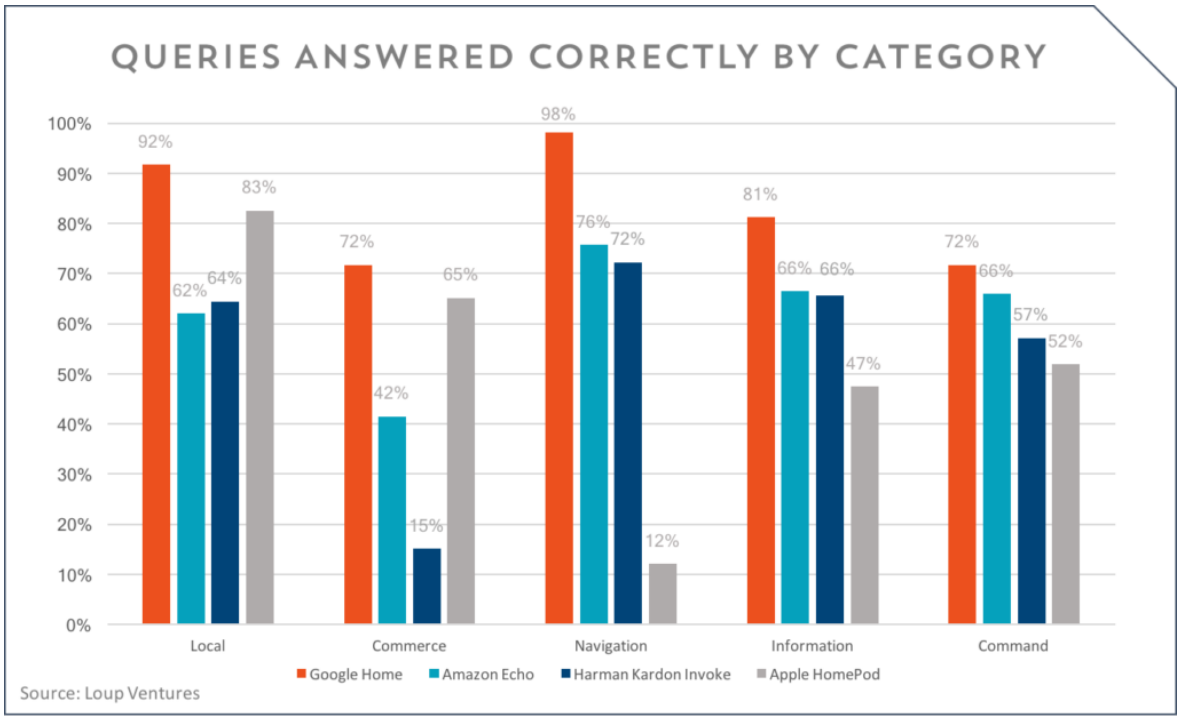


Figure 1 Current digital assistants comparison

3 RESEARCH

This chapter explains the research process made for developing a Digital Assistant. It contains an introduction followed by an explanation of the most common systems, basic concepts and the options that can be used for developing a Digital Assistant or Chatbot.

3.1 Introduction to NLP (Natural Language Processing) technology

This is a compact and understandable definition about what a Natural Language Processing is:

> *Natural Language Processing (NLP) is a sub-field of Artificial Intelligence that is focused on enabling computers to understand and process human languages, to get computers closer to a human-level understanding of language. (An easy introduction to Natural Language Processing, 2018.)*

The origin and history of the Natural Language Processing concept started on 1950 when Alan Turing published an article titled “Intelligence” where he wrote the now known as Turing test. The Turing test is a test of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human.

After that, in the 1960s SHRDLU and ELIZA were developed. SHRDLU was a natural language system that worked in a restricted “blocks worlds” (one of the most famous planning domains used in Artificial Intelligence), using a limited vocabulary, and ELIZA was a simulation of a Rogerian psychotherapist. Both of them were written by Joseph Weizenbaum and they are two basic programs that could be considered as the first chatbots.

The 1970s, many similar programs appear, like MARGIE (1975), SAM (1978), PAM (1978), TaleSpin (1976), QUALM (1977). In the 1980s, the machine learning algorithms that produces a revolution in natural language processing were introduced.

Before the machine learning algorithms, the neural processing systems were based on complex sets of hand-written rules, requiring a lot of lines of code and making it difficult to increase the knowledge of the chatbot because it had to be written by hand. Thanks to the introduction of machine learning and the Markov models, the chatbots started to focus on statistical models.

The statistical models made easy probabilistic decisions based on attaching real-valued weights to the features making up the input data. Thanks to this, the cache languages models (statistical language model) appeared. At the same time, many speech recognition

systems were generally more robust when the input had errors. That makes the development of the chatbots much easier.

Finally, in the 2010s, the representation learning, and deep neural network machine learning methods appeared. Then, those technoques became widespread in natural language processing. (Wikipedia, 2019c.)

NLP is a technique developed to make it easier for the computers to understand the natural languages and give them a more real feedback about what they do on the computer. These changed and added features to the traditional system of communication between human-machine. That provided the possibility of developing systems whose unique input is the voice.

With voice input, developers were able to create applications where a real conversation without problems was possible. NLP Systems allowed the voice recognition and, it includes different types of strategies or ways for get the correct result. These are:

- NLP based on Text, Voice and Audio.
- NLP based on computational models.
- NLP based on Text Analysis that leads to Discussion, Review, Opining, Contextual, Dictionary building, linguistic, semantics, ontological and many fields.
- Pattern Analysis and discovery of new ways such as writing styles.
- Machine Learning of languages.
- Machine Learning focused on prediction & classification of positive and negative views.
- First order logic.
- Automatic Report Generation from Data.

(Arora, P., 2018.)

The NLP based on Text, Voice and Audio is used in the Chatbot. The NLP System is crucial part of the chatbot because it is its brain.

3.2 The role of deep learning in NLP

At this chapter, the concepts of deep learning and artificial intelligence are explained. Also, the relation between deep learning and Natural Language Processing is explained.

Deep learning belongs to machine learning concept, subfield of Artificial Intelligence. It allowed to develop better AI systems which made it possible to define some responses of

the computer as intelligent. Following this, to get better knowledge about what deep learning is, the Artificial Intelligence is introduced first.

The Artificial Intelligence defines the intelligence demonstrated by machines and, sometimes, it is also called machine intelligence. Conceptually, the Artificial Intelligence is the intelligence exhibited by machines and software.

The history of the artificial intelligence started in 1943, when artificial neurons were formally designed by McCullouch and Pitts. The progress of artificial intelligence started to grow at 1970s. The Artificial Intelligence history can be divided in three parts starting at 1943.

Since 1943 until 1974, the first discovery and development of Artificial Intelligence was done. In this period, programming languages for artificial intelligence appeared like Lisp or Prolog. Also, foundations about the development of the artificial intelligence and, big promises about what the Artificial Intelligence can do, were created.

After 1974, the criticism of James Lighthill appeared and, by pressures from the US Congress to fund more productive projects, the U.S. and British governments cut the exploratory research in AI. Then, obtaining funding for AI projects was difficult almost until 1984. This period is known as “AI winter”.

Between 1985 and 1987 expert systems (AI program which simulates knowledge and analytical skills of humans) appeared in the AI market reaching over a billion dollars in these years. But in 1987, a second period without big progresses started by the Lisp Machine market collapse. This last period was informally named as “Second AI Winter”.

Finally, since 1990s until nowadays, the artificial intelligence began to be used for data mining, medical diagnosis, logistics and other areas. After this, the increase in computational power and the resolution of specific problems made the artificial intelligence again interesting for the market.

In 1997, the first computer chess-playing system beat a world chess champion. The deep learning methods started to dominate in 2011 and, in 2016 AlphaGo was the first computer Go-playing system to beat a professional Go player. (Wikipedia, 2019a.)

In summary, artificial intelligence is a word which describes other specifics and longer concepts, like deep-learning, neural networks and machine learning. So, when these technologies are referred, it is better to specify the right name and to avoid generalizations. Now, is the time of the deep-learning techniques explanation.

Deep learning is a machine learning subfield and, it is one of the methods used to train neural networks. Different methods for training neural networks exist and these are: supervised learning, unsupervised learning and reinforcement learning.

Supervised learning is a machine learning task consisting of a function that maps an input to an output based on input-output pair examples. It infers a function from the labeled training data.

Unsupervised learning is a type of machine learning that groups the unlabeled, unclassified and uncategorized data. It works by identifying similarities in the data using cluster analysis. After, it reacts based on the presence or absence of this similarities in each new piece of analyzed data.

Reinforcement learning differs from supervised learning in the labelled input/output pairs and in the sub-optimal actions. The sub-optimal actions don't need to be explicitly corrected and the labelled input/output pairs don't need to be presented. Then, the focus is finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). (Bhardwaj, A., 2018.)

Deep learning can be used in supervised and unsupervised learning. The NLP system's neural network used in the chatbot was trained using deep learning algorithm with supervised learning.

Commonly, deep learning is used in neural networks of multiple layers of nonlinear processing units for feature extraction and transformation. Then, each successive layer uses the output from the previous layer as input.

The deep learning's role inside Natural Language Processing is to simplify the learning by optimizing the time used for obtaining results. Thanks to deep learning, computational models composed of multiple processing layers which learn representations of data with multiple levels of abstraction can be used for teaching the NLP systems in a faster way. (Bhardwaj, A., 2018).

In NLP, deep learning solved problems of inefficiency caused by hand-crafted features. Hand-crafted features are time consuming and may need to be done again and again for each task or problem. Otherwise, deep learning learns information from data and representations across multiple levels, where the lower level corresponds to more general information.

Without deep learning, in an NLP system, the words are treated indecently but, with deep learning, a distributional representation that capture these similarities in a finite vector

space can be used. This brought the opportunity of do more complex reasoning and knowledge derivations in the future NLP systems.

Unsupervised learning and the possibility of learning multiple levels of representations were some of the most important advantages of deep learning. One of the advantages was the possibility of sharing the lower level of representation across tasks.

Finally, deep learning improves the capture of information sequences in a much better way. (Wikipedia, 2019b.). Deep learning helped to improve the NLP systems with a notable performance. Nowadays, Deep learning is still improving its performance.

3.3 Cloud services for NLP

In this chapter, first of all, what kind of services the market has for integrating an NLP system in an application are explained. After that reasoning, why a Cloud service was chosen and, which Cloud services are in the market are explained. Finally, the decision is explained.

Implementing an NLP system in an application has two possibilities: a local application or a Cloud service. In this project a Cloud service option was chosen because it allowed running the application correctly in all the platforms.

An NLP Cloud service has all the features of an NLP system needs. There are many of them on the internet. These Cloud services provide all the tools for managing the NLP system and the only work to be done is, teaching and training the neural network provided in the correct way.

These Cloud services provide an instance of their NLP system which can be modified depending of our needs. Then, a custom NLP system can be developed using the framework and tools provided.

The neural network must be trained for recognizing all the words in each one of the configurable languages. Then, based on the desired response, the neural network starts to change for fitting to the model desired by the user. Validating the correct samples in the neural network, a custom structure of response can be defined.

The decision, about which way to take for integrating NLP system in the application, depends on the developing necessities but, taking the Cloud way is more comfortable than implement it locally. The Cloud services are very advance systems and, implementing a similar system locally would be a titanic work rarely affordable by anyone.

3.3.1 IBM Watson

IBM Watson is the IBM's NLP Cloud Platform. This platform offers great flexibility to create a custom chatbot. It offers features like text to speech, visual recognition and Natural language Understanding (NLP variant focused on word classifying). Also, it offers other interesting features to develop a powerful chatbot.

In fact, Watson is the IBM's digital assistant. So IBM offers to the developers all the tools that they used in Watson. This gives developers a great experience with many options and customization possibilities.

IBM charges a monthly share for using their systems. The IBM Watson system is not open source, so one needs to pay to be able to use IBM Watson. IBM offers a free version that allows 10000 requests per month and after that, $0,0025 per request must be paid. Their other plans may cost less because they have a customized plan, but for knowing more about the plans, the developer must contact to IBM. (IBM Systems, 2019.)

3.3.2 Dialogflow

Dialogflow is the Google's AI system. Previously known as API.AI, Dialogflow is the NLP Cloud Platform based on the api.ai system launched by Speacktoit in 2014 for giving third-party support for voice interfaces in applications based on Android, IOS, HTML5 and Cordova. Google bought api.ai in 2016 and, after 2017, it is known as Dialogflow.

Google brings to developers a platform with an NLP system and features similar to IBM Watson. Among others, a text to speech service is offered and it can be used in chatbots.

Google offers streaming input of audio like IBM Watson and, all the features than an NLP System should have. Also, like the other services, they offer a range of Languages for speech recognition and the ability of multi-language recognizing at the same time.

As in the IBM case, Google also has a subscription model for using its systems. In this case, the system of IBM is more powerful than Dialogflow for developing.

Dialogflow is open source and it can be implemented on local environment but, it is not practical. To use the Cloud System is the best option if the application is distributed for the public use.

The monthly price can change because it depends of the use of the service. They offer a free plan, but only with 10000 request per month. It is not enough if the developer wants to build a real chatbot. To increase the request limit, the unlimited plan is needed and has a cost of $0,0065 per request with 15 seconds of audio. (Google, 2019.)

3.3.3 Wit.ai

Wit.ai is an open source platform which provides an NLP System. Facebook bought Wit.ai on 2015 but, nowadays, Wit.ai is open source and free to use. It supports a lot of languages, almost 50. Otherwise, it doesn't have features like Google Dialogflow or IBM Watson. For example, audio streaming, simultaneous translation or text to speech are missing.

Wit.ai has the basic features of an NLP System needs. They work very well and a very consistent chatbot can be built with them. With the correct planning, and the correct structure of the entities, a very competitive and practical chatbot can be made.

The best parts of Wit.ai are their open source philosophy and free cost. The applications developed with their services can be used for commercial use without paying any kind of share. (Wit.ai, Inc. 2019.)

3.3.4 Comparison of alternatives

Wit.ai was the chosen Cloud service for the chatbot. In terms of cost-benefit, Wit.ai was the best option because its missing features are not mandatory for the chatbot. Referring to the cost, Wit.ai offers a better service at the same price as the other platforms.

In the next table (Table 1), the features of the three Cloud services are explained. The practical part of the project explains more exhaustively why Wit.ai was chosen.

The table compares the most important features of an NLP System. The evaluated features are: speech-to-text, text-to-speech, streaming audio, supported languages, facility of use, price, is it open source, rest API, training modules, importing exporting models and limit of calls.

The comparison was made a priori with the information available on the internet. After, a small test on the platforms was done, and then, the points were given. All the features are evaluated from 0 to 10 and the maximum possible score is 110.

Wit.ai get the higher points with 90 and it was chosen. It must be noted that the most important feature was the price.

Table 1. Comparation between Cloud services

features \ cloud services	Wit.ai	Dialogflow	IBM Watson	Ideal score 0-10
speech-to-text	10 (yes)	10 (yes)	10 (yes)	10
text-to-speech	0 (no)	10 (yes)	10 (yes)	10
streaming audio	0 (no)	10 (yes)	10 (yes)	10
languages	10 (50 languages)	2 (15 languages)	1 (1 language)	10
facility	10 (Simple to use)	6 (A bit complex)	4 (So much tools)	10
price	10 (100% free)	2 (limited free plan)	2 (limited free plan)	10
open source	10 (100% opensource)	7 (opensource but must pay for use it)	0 (Not opensource)	10
api rest	10	10	10	10
training module	10	10	10	10
allow import/export model	10	10	10	10
Limit api calls	10 unlimited	5 10k free plan	5 10k free plan	10
Results	90	82	72	110

3.4 Strategies and methods in training a chatbot in Wit.ai

The strategies and logical structures used for the chatbot are explained on this chapter. The logical structure was planned for a correct interpretation of the sentences and, for a correct answer in the greater variety of scenarios.

First of all, to organizing the response, a logic representation was planned. Wit.ai works recognizing entities which match with the sentences sent by the user. These entities can be keywords or concepts and, the neural network is trained to recognize them.

Wit.ai can receive sentences by text or by audio clips. Normally, both ways work perfectly but, in some cases, it causes some errors. When the language is strange for Wit.ai, like German or Finnish, the audio transcription generates some errors.

The chatbot works with the following languages: Spanish, English and Catalan. With these three languages, Wit.ai works well. The Finnish was tested for implementing it on the chatbot but, Wit.ai has some problems with the transcriptions and with the use of Finnish. So, the Finnish language was delayed in the development and, it is one of the future improvements.

Wit.ai works with entities and, it has four types of entities: trait, keywords, free-text and keywords & free-text. These types of entities are used for obtaining different responses in the same situation depending of the context.

The Entity of type trait could be considered as the undefined entity. It is the entity responsible of discern about what the meaning of the sentence is. The trait entity can be used for defining if a sentence is affirmative or negative. Usually, this entity is used for interpreting things that do not appear explicitly in the sentence.

This kind of entity needs to be trained correctly, and for that, a set of samples is needed. With these samples, the solution must be known. The entity is trained with correct and incorrect responses.

When the system starts to recognize sentences outside the set of samples, it can be considered sufficiently trained. The entity can be trained more exhaustively keeping in mind the possibility of overfitting (when a neural network is overtrained and its results starts to be unsure).

One point of Wit.ai is that can be trained using the received data but, sometimes, it can be hard because the validation must be done by hand. Then, known this, this entity was used classifying the sentences in Order, Question or Y/N(Yes/No). The entity IN of the chatbot uses the trait way.

The keyword entity works searching matches between the defined values inside the entity and the text. After, it returns the obtained match. This kind of entity does not need a lot of training, but the NLP System must know which generated responses are correct. So, a lot of samples must be verified for making consistent the response.

If the logical structure marks some exceptions, like sentences which should not be recognized, these exceptions must be marked as wrong response in the validation. Only one entity with this strategy is used on the chatbot and it is ConcretePlaces. This entity is used as database of sites, concrete places and cities. It helps to understand and answer correctly the sentences.

The free-text entity uses a similar way than the keyword entity but it has some different and interesting features. This entity matches their values on the sentences, like the keyword entity does, and also it matches synonyms. In the entity, a synonym contains a group of words. So, it can be used to identify a range of words as one word, for example, to know if the sentence contains colors.

This entity can have one value named Colors and, in their synonyms, to contain all the colors than needs to be recognized. The same functionality can be done with keyword entity and trait entity but it is not affordable. In the other entities, the name of the entity would work as synonym and, their values would work as the words to recognize.

The keywords & free-text entity combines the keywords and free-text strategies, it means that it recognizes a match in the text, the defined values inside the entity and its synonyms. This type of entity is used in some entities of the chatbot. These are: *type*, *y/n*, *verbs* and *recommendations*. This kind of entity can have a value working as a concept and, a concrete value working as a synonym of a concept. Also, this type of entity brings the possibility of not to have several similar entities in the application.

To explain the logical structure of sentences, first, the purpose of the used entities must be explained. Then, the structure of Wit.ai's response is explained, and finally, a directed graph which explains how the chatbot internally works is presented. The entities made for the chatbot are:

- Type. It determines the type of the sentences: question or order. Their values are directions, actions and schedule.
- Y/N. Used for recognizing the words yes or no in the sentence. It is not contemplated in the schemes and, it is only used when a request needs some context. For example:
    - User: “I want bread”
    - App: “Do you want to go to the nearest bakery?”
    - User: “Yes, I want”
    - App: “Opening Google Maps…”
- Verbs. This entity works like a big database of possible verbs that can appear in the sentences. It helps to create a better response in the chatbot.
- ConcretePlaces. It works practically in the same way as Verbs but, with the names of concrete places, like for example “sMarket”, “Mercadona”, “bar Manolo”, etc.
- Recommendations. Entity used for generating a kind of recommendations about the recognized word, normally places. It is usually used for taking information about directions or opening hours about one place. It only works when the user did not define correctly the place on the request. For example:
    - User: “I want to go to the supermarket”
    - App: “The nearest supermarket is Mercadona, do you want to go there?”
    - User: “Yes”
    - App: “Opening Google Maps…”

- IN. Entity used for classifying sentence in “order”, “questions” or “y/n”.

The Figure 2 shows the structure of response planned for the chatbot. It presents, which entities must Wit.ai return when it recognizes a sentence (when the NLP System is trained).

There are 5 entities, *IN*, *Type*, *Recommendations*, *Verbs* and *ConcretePlaces*. The last two entities are used like databases, they contain a very long range of words. The structure below defines that each response must include mandatorily the entities IN, Type and Recommendations. After these three entities, it is possible to include the entities Verbs and ConcretePlaces which are not mandatory.

In addition, there is a special case when the response does not include the Recommendations entity. If the ConcretePlaces entity is in the response, the Recommendations entity can be replaced by ConcretePlaces and it is seen as a correct Wit.ai response. In the diagram (Figure 2), the point on the left side means “mandatory statement” and the interrogation symbol “optional statement”.

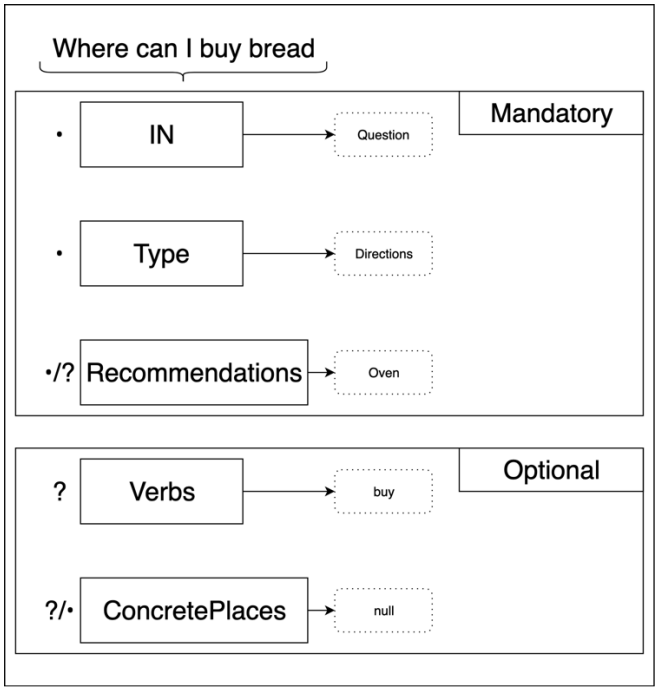


Figure 2. Abstraction of a Wit.ai response

In the Figure 2, one can see the response for the sentence “Where can I buy bread”. The response has three mandatory entities, IN with a value or “Question”, Type with a value of “Directions” and Recommendations with a value of “Oven”. Also, the Verbs entity with a value of “buy” is there. The ConcretePlaces entity is not received and is interpreted as a null value. All que responses from Wit.ai must have this structure.

In some cases, because Wit.ai works with probabilities, it can return twice the same entity in the response but with different statistical values. This happens because the entity matched more than one time in the sentences. So, in that case, the higher probability entity is taken.

Finally, the Figure 3 is the final state machine made for understanding how the chatbot works. Below, the graph's transitions are explained with the right and wrong paths. It helped in the programming part doing easier the implementation phase. Reco means Recommendations, cPlace means ConcretePlaces and the letters are for identifying the paths and transitions.

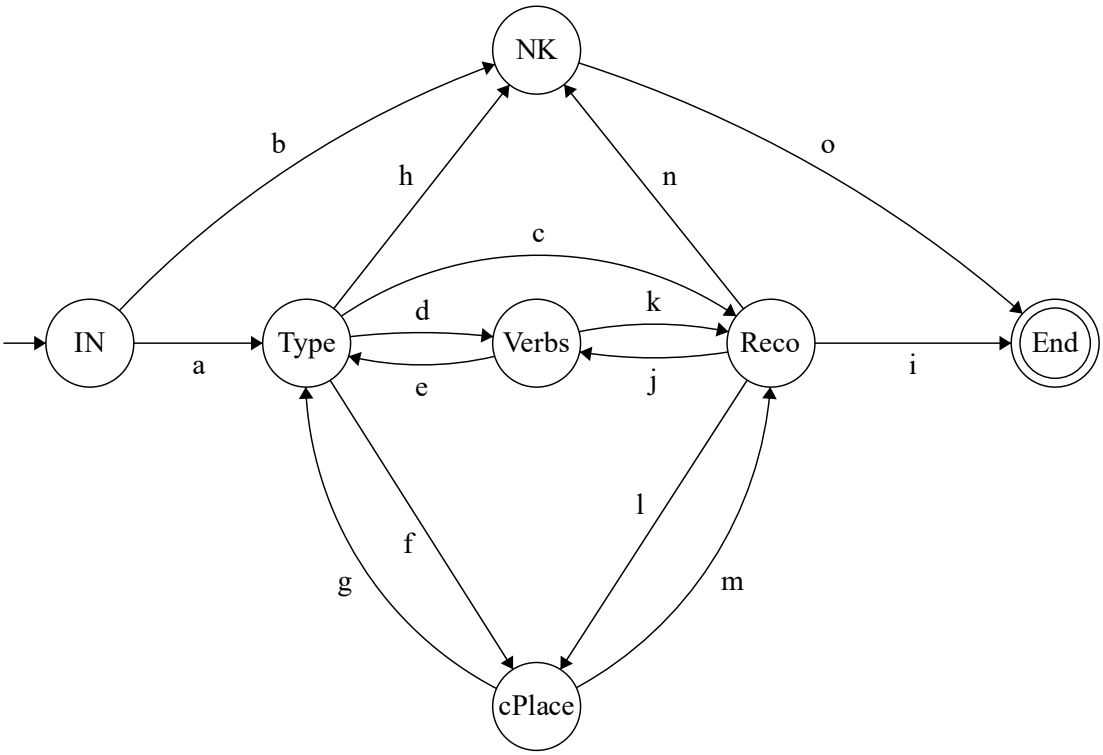


Figure 3. Chatbot's finite state machine

To get a right path, the response must have data from the entities IN, Type and Recommendations. If the response has the intents IN, Type and Recommendations, the path 1 is obtained. In the case where, the Recommendations entity is missing in the response but ConcretePlaces is in the response, the function of Recommendations is replaced by ConcretePlaces (Path 2).

In addition, the response can have the entities Verbs and ConcretePlaces, but the previous entities IN, Type and Recommendations must also appear (Path 3). The right paths are:

- a, c, i (1)
- a, c, l, m, i (2)
- a, c, j, k, l, m, i (3)

In the following cases a wrong path can be obtained. When the IN entity is not in the response means that some error occurred in the speech recognition or another kind of error. So, the path 4 is the followed, and an answer like this is generated: “I didn’t understand you, could you repeat your question?”.

If the IN entity is received but Type is missing, the entities Verbs and ConcretePlaces are checked if they are in the response. Then, the NK state generates the nearest answer to the question made by the user. In this case, the paths 5, 7, 8 and 9 are followed.

Finally, if the entities IN and Type are in the response but Recommendations is missing, ConcretePlaces replaces Recommendations if it is in the response. Then, the entities Verbs and ConcretePlaces are checked and the NK state generates the nearest response to the question made by the user. In this case the paths 6, 10, 11 and 12 are followed. The possible wrong paths are:

- b, o (4)
- a, h, o (5)
- a, c, n, o (6)
- a, d, e, h, o (7)
- a, f, g, h, o (8)
- a, d, e, f, g, h, o (9)
- a, c, j, k, n, o (10)
- a, c, j, k, l, m, n, o (11)
- a, c, l, m, n, o (12)

3.5 Analysis of possible queries in natural language in providing services of a city

There are some queries that a normal user can ask from the chatbot. they can be divided between the simple queries and the complex queries. The chatbot must be able to respond to each of these queries.

Some simple queries are: “What time is it?”, “Where is the Hospital?”, “Where is the super market?”, “What day is today?”, “I need bread”, “Call to emergencies”, “Where can I find plants?”, “Guide me to the nearest super market”, “Which weather will tomorrow do?”, “Will it rain today?”, “When the university is closed?”, etc.

Some complex queries: “Remind me tomorrow that I have piano class at 6”, “Call to xxxxxxx”, “What is the schedule of Prisma?”, “When is the birthday of David?”, “Which time slots does I have free tomorrow?”, “Put an alarm at 6 in the morning”, etc.

4 CASE: CHATBOT ASSISTANT

4.1 Implementation of the chatbot for the cities (Alcoi/Lahti)

In this chapter chatbot implementation using Wit.ai is explained. First, the implementation possibilities are explained. Then, which strategy was used, and which Cloud service was chosen are explained.

There are different ways to implement the chatbot. This chatbot was for public use, and, it must be possible to use it in the highest possible percentage of people. At least, the chatbot must be available on the most common mobile platforms, Android and IOS.

The chatbot was focused on the cities of Alcoi (Spain) and Lahti (Finland) to get reliability over its competitors, Google Assistant and Siri. By limiting the scope, it is possible to get more accurate results with specific data and more dedicated training. Then, to make sure that the chatbot works correctly, field work is needed. This field work includes, information research about places and their opening hours.

For the implementation of the chatbot, one framework has to be chosen. There are a lot of frameworks and, for this development the choice was between Flutter (Android/IOS), Android Studio (Android), XCode (IOS), Xamarin (Android/IOS) and Unity (Android/IOS/Web/Mac/Windows). Finally, Unity was the chosen because it allows to develop for Android, IOS, Web (WebGL), Windows and Mac at the same time.

These frameworks are subdivided in two categories, the native frameworks and the cross-platform frameworks. Among the native frameworks are Android Studio, for developing Android based apps, and XCode, for developing IOS and MacOS apps.

It could be nice to work with this kind of frameworks if the application was focused on one of them. However, if both platforms are needed, the work must be done twice. For this reason, neither of these two options were chosen.

Then, there are the cross-platform frameworks: Xamarin, Flutter and Unity. With this type of framework, an application can be developed on different platforms by writing the application’s code only once.

Among three options, Unity was chosen because it allows to develop with 3D (useful in the way of making this app different from the others). It also allows to develop for more than three platforms (Android, IOS, MacOS, WebGL, Windows, some video game consoles) with the same features as Xamarin and Flutter and, adding the 3D compatibility.

Xamarin and Flutter allow to develop the app in a faster way, but the fact that Unity have a lot of platforms for developing, tips the scale to Unity.

4.1.1 Strategy and Cloud service selection

Mainly, two things are needed to build this chatbot. The first is the platform or library that brings the NLP system for the application, and the second is the framework able to support the chosen NLP system.

The decision about the NLP system was to choose a cloud platform. The cost of the service motivated the decision. The chosen service was Wit.ai because it was completely free, and any kind of payment was not needed. This was an important feature because the chatbot is planned for public use in the App Store and Google Play Store.

In addition, Wit.ai was chosen because every tool needed for the chatbot was contained on it. The response structure can be built easily without problems, and their rest API allows to make a separate program for training the NLP System without updates in the application. Then, finally Wit.ai was chosen as the NLP system of the chatbot and Unity was the chosen framework for the development.

4.2 How Wit.ai works

This chapter gives an overview if Wit.ai. Then, how the Wit.ai's system was configurated is also explained.

First of all, for using Wit.ai, a Facebook or Github account is needed. The recommendable way is to login using a Github account because it is the most specific platform. After logged in, an app must be created.

The Wit.ai’s app is a Wit.ai’s logic abstraction used for identifying an instance of the NLP system. This instance only works with one language and its configuration is unique to the app.

Then, a name for identifying the app is needed. While configuring the application, you must decide if the application is open or private for external developers. The name of the app only has relevance inside Wit.ai and it does not affect to the integration. The open option means that everyone can use and modify the app. The private app is accessible only by the creator and authorized users (Figure 4).

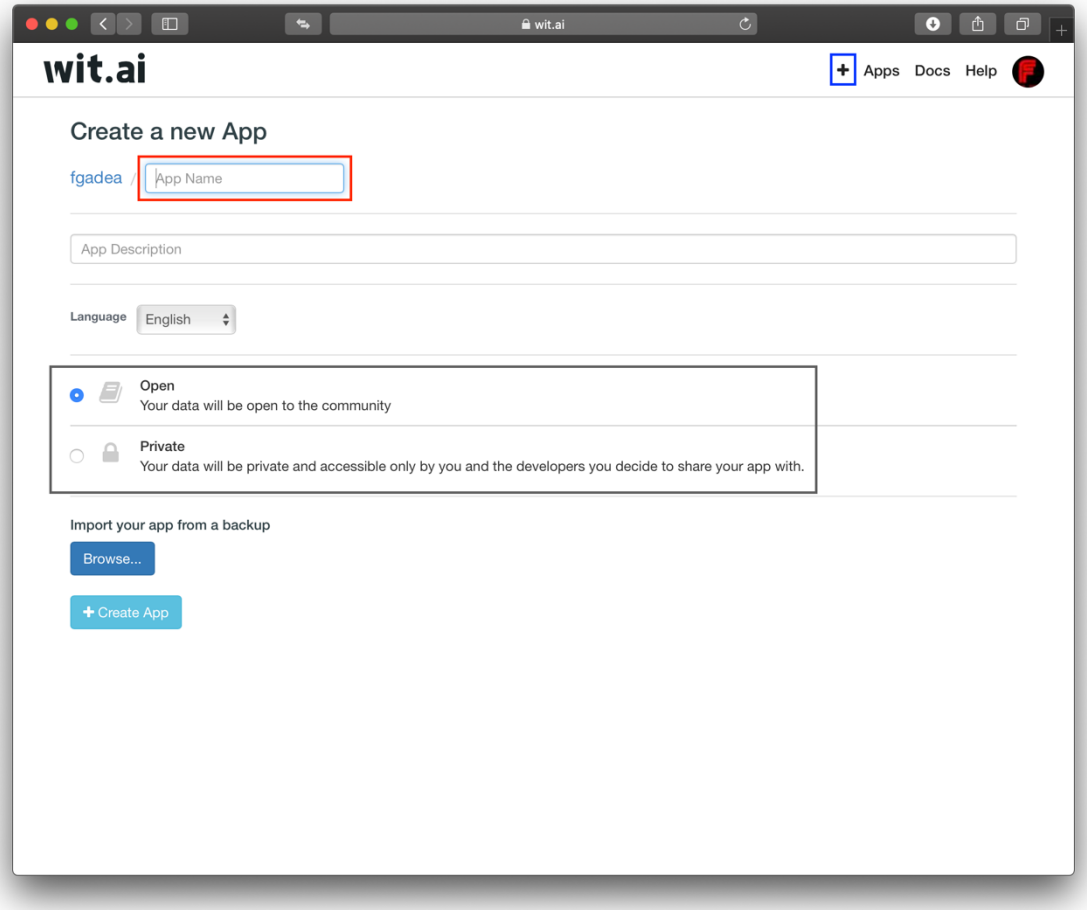


Figure 4. Wit.ai new app interface

Each app in Wit.ai is completely independent from the others because Wit.ai does not work in the same way with all the languages. However, Wit.ai allows to clone apps, a feature useful for working in the same way in all the languages, only doing small changes inside each app. Once the application has been created, it appears in the index of apps.

Next the app can be trained. For training the app, first of all the entities that the app recognize must be created. The entities in Wit.ai are easy to create, by clicking on “add entities”, writing the name of the entity and clicking enter.

When the entities have been created, the correct strategy for the entity must be chosen. The available entity strategies are trait, keyworks, free-text and keyworks&free-text. Once the strategy has been chosen, the values can be set through the entity manager view.

When all the entities are created, the process of training by samples can be done following the instructions of the interface. Putting the sentence, checking or modifying the suggested Wit.ai’s response, and validating (figure 5).

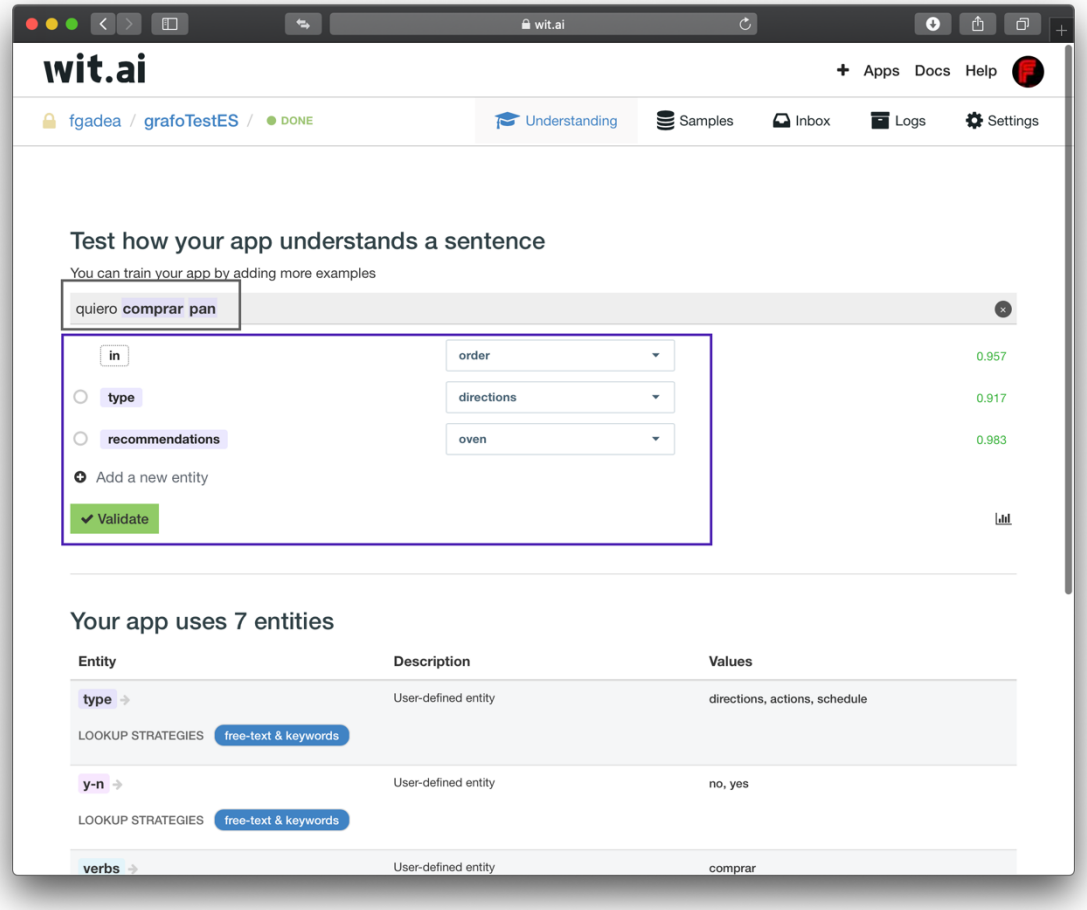


Figure 5. Wit.ai app settings interface

The process of setting a sentence and validate it must be done one by one because the interface only allows one sentence at a time. So, the process must be repeated a lot of times. As many times as sentences the training samples have.

To modify some parameters of an entity, only a click is needed over the entity (Figure 6) in the main view of the app. All the entities have the same UI and change their type or name is a very intuitive.

Inside of an entity, in the lowest part of the entity selected UI appears all the sentences related with the entity. These sentences can be deleted selectively, and also, it can be done by the Wit.ai rest API.

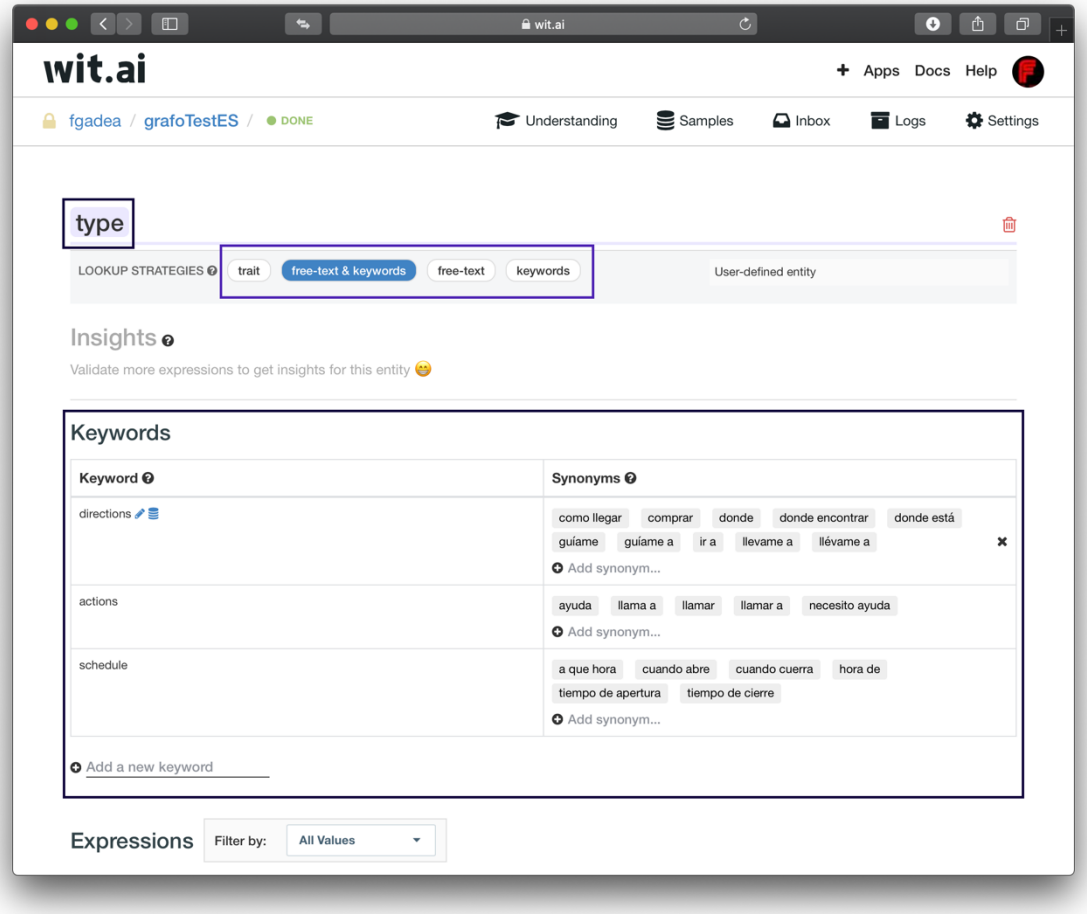


Figure 6. Entity UI

The interface has five buttons: Understanding, Samples, Inbox, logs and Settings. These five buttons form the main menu of the app, and they are used to manage the app.

Understanding button goes to the main view of the app. There, entities can be accessed, and examples can be added. Samples button goes to a view with a list of all the samples. In this list the responses to the sample can be seen and modified. Also, the sample can be deleted.

The Inbox button goes to a view where all the request done by users can be found. These sentences must be validated or discarded by the developer. Also, the audio requests can be listened to check the response.

The Logs button shows the NLP system processing output. Finally, the Settings button goes to the view where the name, permissions and other about the app can be modified.

It is not affordable to put samples by hand when the amount is big, for example 2000. In this case, Wit.ai has a rest API that allows to train the app outside the platform. This rest

API has request of type POST, GET, PUT and REMOVE. It can be used for training the app.

In order to implement this rest API, Postman can be used. It functions as a request manager. Any other program which implements the rest API can be used. In the project, a custom program was made because Postman did not allow to do repeated requests at time (for example: repeating 10 times the same request). Also, Postman difficulted concatenating different requests because it is a program focused on testing.

Because of these reasons, a custom program was made for training the chatbot and it was used at the end of the development. While in development, a short list of examples was used. This list compiled a basic and sufficiently reliable responses. The training program is explained in the chapter 4.5.

A complete set of useful requests is provided by the Wit.ai's rest API, nevertheless not all of them were used. The requests used in the development are explained in the next subchapter. It must be noted that all this information may not exactly be the same as the currently available in Wit.ai as it is subject to changes.

These were the basic issues for training the Wit.ai's app used in the Unity implementation. There were the most important aspects of Wit.ai for training the chatbot.

Finally, before passing to the next chapter, another feature of Wit.ai must be remarked, its learning curve. The Wit.ai's learning curve is not very steep which means that, in short time, anyone can manage and use Wit.ai without problems. This makes the development compared with other services.

4.2.1 Wit.ai rest API

In this subchapter, the requests of the Wit.ai's rest API used in the application (chapter 4.3) are explained. The requests used in the training program (chapter 4.5) are also explained. The number of available requests in the Wit.ai's rest API is 32, but in this case only 12 are used sorted into 5 categories.

Add, remove or get samples from an app POST/GET/DELETE

This group contains the three requests that work with the samples used for training the NLP System. The GET request is used for requesting a list of samples. The POST request is used for inserting a sample in the app. The DELETE request is used for removing a specific sample from the app.

With GET, a list of samples is obtained. The obtained list can be customized with parameters. There is a mandatory argument, *limit* (integer) which indicates how many samples are required. This *limit* can range between 1 and 10000 included.

There are other optional parameters. They allow GET to output a more specific list of samples. These optional arguments are: *offset* (integer) indicates the number of samples to skip from the start; *entity_ids* (array of strings) required names of entities that the samples must have; *entity_values* (array of strings), list of values of the specified entities that all returned samples must include, and it requires entity_ids; and *negative* (boolean), if true it will show the negative samples of the specified entity_id, and entity_values must not be included.

Example of a GET request is presented in the Figure 7. It gets a list of 10 samples which have the entity "entity" in their response and where the values of "entity" is "buy_car".

```
https://api.wit.ai/samples?entity_ids=intent&entity_values=buy_car,dri
ve&limit=10
```

Figure 7. Get request example

With POST, the sample sent is added to the list of samples in Wit.ai. There is a limit of 200 samples per minute that can be modified by contacting with Wit.ai. This request validates the sample automatically for training the app, and two mandatory arguments, *text* (string), and *entities* (array of entities object) are required.

Text field contains the sample sentence, and entities field contains a list of entity objects where the entities and values to recognize are specified. These two arguments are contained in a sample object, and this object is also contained within an array of sample objects which is sent using JSON structure seen in Figure 8.

```
[
   {
    "text": "I want to fly to sfo",
    "entities": [
      {
        "entity": "intent",
        "value": "flight_request"
      },
      {
        "entity": "wit$location",
        "start": 17,
        "end": 20,
        "value": "sfo"
      }
    ]
   },
]
```

Figure 8. JSON example of POST request

With DELETE, the samples sent in a list are removed. This request only has one mandatory argument, *text*, which contains the sentence. It can also be sent using the POST request, but it is only recognized in the text parameter (Figure 9).

```
[
  {
    "text": "I want to fly to sfo"
  },
  {
    "text": "I want a flight"
  }
]
```

Figure 9. List of "sample" objects with two objects

(Wit.ai, 2019.)

Add or remove expression of a value on an entity POST/DELETE

With the requests of this group, expressions of an specific value of an entity can be added or removed. It is not possible to modify the expressions of a value in Wit.ai. So first, the expression must be deleted, and after that, the new expression must be added.

With POST request, it is only possible to add an expression to a value. With the possibility of executing the request programmatically, it can be added, for example, 20 expressions with only one action.

In this request, an object of type expression must be sent as JSON object with the name of the expression no longer than 280 characters. This request is based on two parts. The link, where the entity and the value are specified (Figure 10), and the JSON file with the expression (Figure 11).

```
https://api.wit.ai/entities/favorite_city/values/Paris/expressions?v=2
0170307
```

Figure 10. POST url

```
{"expression":"Camembert city"}
```

Figure 11. JSON sample with expression

With the DELETE request, the expression to be deleted must be indicated on the url. If the expression matches, the expression is deleted. A JSON file is not needed on this request (Figure 12).

```
https://api.wit.ai/entities/favorite_city/values/Paris/expressions/Cam
embert%20city?v=20170307
```

Figure 12. Example of DELETE expression request

(Wit.ai, 2019.)

Add or remove a value of an entity POST/DELETE

These two requests are used for adding/removing a value from an entity. POST request adds values to an entity. However, it only works with entities of a keywork strategy. The Samples POST (previously explained) must be used for other type of entities.

The POST request works by specifying the entity on the url, and with a JSON file as a parameter. This JSON file contains three arguments of which the first is mandatory and the other two are optional.

The mandatory argument is *value* (string), the value to add to the entity. The other two optional arguments are *expressions* (array of strings) and *metadata* (string). The expressions field contains a list of expressions for the value, and the metadata field contains the metadata to attach to the value (Figure 13).

```
{
  "value":"London",
  "expressions":["London"],
  "metadata":"CITY_1234"
}
```

Figure 13. JSON example of POST value request

The DELETE request erases the value sent as parameter in the url (Figure 14). This request erases the value sent if it matches with some of the entity's values. It does not need any JSON.

```
https://api.wit.ai/entities/favorite_city/values/Paris?v=20170307
```

Figure 14. URL example of value DELETE request. entity "favourite_city", value "Paris".

(Wit.ai, 2019.)

Create, modify, delete and get entity from an app POST/GET/PUT/DELETE

These four requests work with the entity objects. With these requests it is possible to add/modify/delete entities in the app. Furthermore, with the GET request, obtaining a list of values with their expressions of an entity is also possible.

Only one entity can be added/deleted per request, but programmatically, more than one request can be done. Otherwise, updating multiple entities is possible with only one request. In the case of the GET request, obtaining a list of entities is possible, but with a maximum length of 1000 entities. However, if the entity is fetched with its values and expressions, then a maximum of 50 entities is required.

With the POST request, adding entities to the app is possible. This request works by sending a JSON file where the name of the entity, *id* (string), is mandatory, and a short description for the entity, *doc* (string), is optional (Figure 15). The lookup/strategy assigned to the entity is automatic and it can be changed in the Wit.ai web page.

With the GET request (Figure 16), a list that contains the values of the entity is obtained. The list has a maximum of 1000 values. Otherwise, if the list contains the expressions of each value, then 50 values is the maximum. This request is very useful for the PUT request explained next.

With the PUT request, the name of the entity followed for a list of the values is sent through a JSON file. On this request, all the parameters are optional. *Id* (string), name of entity; *doc* (string), description for the entity; *lookup* (array of strings), type of entity; and *values* (array of value objects), array with objects value that contains name and expression of the value (Figure 17).

Finally, the DELETE request deletes permanently the entity indicated in the url (Figure 18). If the name of the entity is not matched, an error is returned. It must be noted that the entity is deleted without confirmation, so, it is recommendable to backup the app before erasing anything.

```
{
  "doc":"A city that I like",
  "id":"favorite_city"
}
```

Figure 15. POST entity request example

```
https://api.wit.ai/entities/first_name?v=20170307
```

Figure 16. GET request example

```
{
  "doc":"These are cities worth going to",
  "lookups":["free-text", "keywords"],
  "values":[
     {"value":"Paris",
      "expressions":["Paris",
                     "City of Light",
                     "Capital of France"],
      "metadata":"{\"cityId\":342,\"countryId\":12}"
     },
     {"value":"Seoul",
      "expressions":["Seoul",
                     "서울",
                     "Kimchi paradise"],
      "metadata":"city_343"
     }
  ]
}
```

Figure 17. JSON example PUT entity request example.

```
https://api.wit.ai/entities/favorite_city?v=20170307
```

Figure 18. Sample of URL for DELETE entity request

(Wit.ai, 2019.)

Obtain a list of entities GET

This GET request returns a list of the entities. The list does not have any restriction of length and it comes inside a JSON file (Figure 19). This is useful for checking iif the creation of an entity was successful.

```
[
  "wit$amount_of_money",
  "wit$contact",
  "wit$datetime",
  "wit$on_off",
  "wit$phrase_to_translate",
  "wit$temperature"
]
```

Figure 19. GET entities request example.

4.3 Integration with Unity

When everything about Wit.ai is known, the chatbot can be built as an application. When building the application, the structure of the application must be defined, and how Wit.ai and the application are going to communicate. Then, how the chatbot interprets the Wit.ai's response must also be defined. This brings good and intelligent feedback to the user.

In this development the incremental model of developing software is followed. This model of development is contained inside the category of agile software development techniques.

As the incremental mode indicates, this chapter is subdivided in four parts: Analysis, Design, Code and Test. The model is followed as is said in the book “SWEEBOK”, certified by the IEEE Computer Society as a guide for Software Engineering.

4.3.1 Analysis

In this chapter the requirements to bild the application, which software to use and what platforms to use must be analyzed. This chapter has been already explained previously in the document.

For building the application, the framework used is Unity. Unity was the chosen because it brings the most compatibility between platforms.

Regarding the AI aspect, two options were possible: implementing the NLP System internally or using a Cloud System. The chosen option was the Cloud system.

Among the considered cloud systems, Wit.ai was chosen because, when comparing the features of the other three options, Wit.ai was the most reasonable option. Wit.ai is open source and is totally free to use in commercial environments. The others had to be paid in all the use cases.

For building the chatbot nothing more is needed. With the Unity framework and the Wit.ai cloud system, a powerful chatbot can be built.

Finally, must be noted that thanks to the framework used, the chatbot can work at least on Android, IOS and Web environments. In a future, the possibility of build for Windows and MacOS is taken into account. However, for the time being it will only be built for Android, IOS and Web environments.

4.3.2 Design

In this chapter, the design of the chatbot is explained. The chapter comprise the response's structure that Wit.ai must have, and the application's workflow. These two aspects have been also described previously in the chapter 3.4.

For Wit.ai, a short list of samples is used in the three languages that the chatbot supports (Spanish, Catalan, English). Then, the apps, one for each language, were trained using the pattern designed and explained previously (Figure 2). Each response must have the entities *IN*, *Type* and *Recommendations/ConcretePlace*.

Once the apps are trained, Wit.ai is ready to be integrated in the application. For integrating Wit.ai in Unity, the Wit.ai's rest API must be used. In the Wit.ai's rest API two requests exist, POST/speech and GET/message. With the POST/speech, a .wav file is sent as binary data and Wit.ai internally creates speech_to_text response message. With the GET/message, text is sent to Wit.ai and a response is generated.

On the other hand, for the software, the application was developed according to the diagram explained previously (Figure 3). The application was an implementation of the finite state machine designed to understand the Wit.ai response (Figure 3). Then, the application is subdivided on three parts: the user interface, the networking module (Wit.ai connection) and a module that processes the response of Wit.ai.

The workflow diagram of the application will be shown in the next Figure 20:

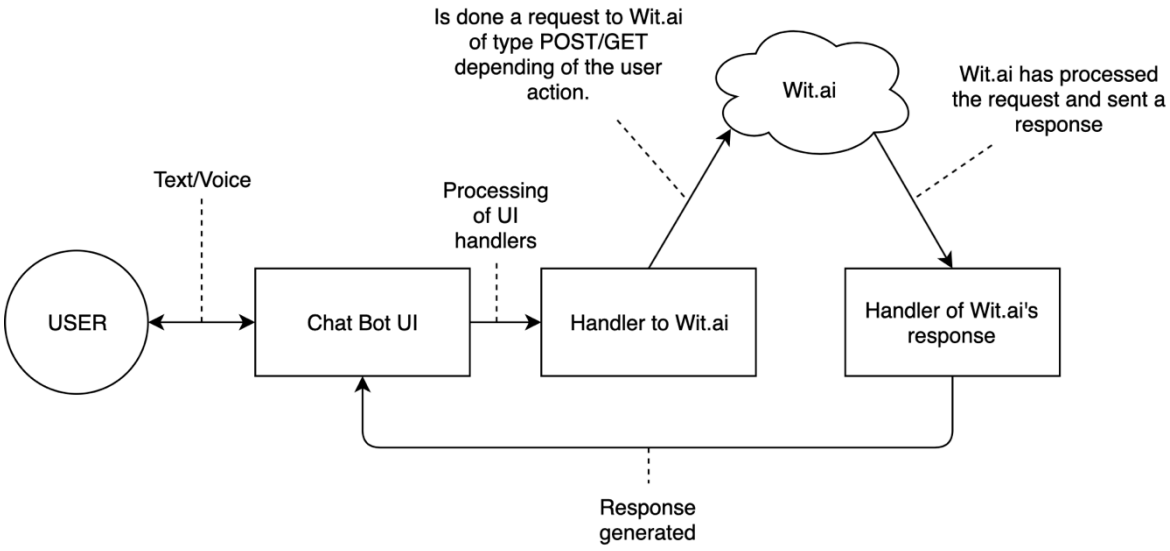


Figure 20. Workflow diagram of the chatbot.

4.3.3 Code

In this chapter, the third part of the process in the incremental model is explained, the code. This process can be subdivided in three main parts. One containing all the user interface functionalities. Other containing the communication between Wit.ai and the application. And the third part contains the process of understanding the Wit.ai response, followed by the generation of a correct answer to the user using databases and other tools.

At the same time, these three parts can be joined in two, the visual and the programming parts. The visual part contains the user interface, and the programming part contains the communication with Wit.ai and the answer generation. Then, this chapter explains these two groups, the UI design and implementation, and the Wit.ai implementation.

UI design and implementation

In this chapter, the design of the user interface and the implementation of its handlers are explained. In this point, also how the audio is recorded is explained as it also belongs to the UI part.

The user interface was designed based on user interaction. As the application is a digital assistant, there are two ways for it: using text or using voice. Also, the communication between the digital assistant and the user can be done in two ways, visual (text and images) and with voice.

In this case, for the communication between the user and the digital assistant both, voice commands and text commands, are used. These two options work independently but not simultaneously. The user may do a request by voice or a request by text but will not be able to do it at the same time.

For the communication between the digital assistant and the user, the voice and the visual response are used together bringing a great user experience. These options are used at the same time. For example, the digital assistant may respond to the question “Where am I?” showing the text, “You are in Lahti”, followed by an image of a map which indicates the real position with a link to Google maps. At the same time, through the speakers, the sentence “You are in Lahti” is played using a text_to_speech system.

Also, must be noted that the chatbot is working on three languages and that they cannot work together. This means that the user must specify which language they want to use, and the interface must contain a icon indicating which language is being used.

Then, visually, the user interface has a button for the voice input a textbox where the request is written, and a button for selecting the language. The sketch in Figure 21 was the base for the final implementation in Unity.

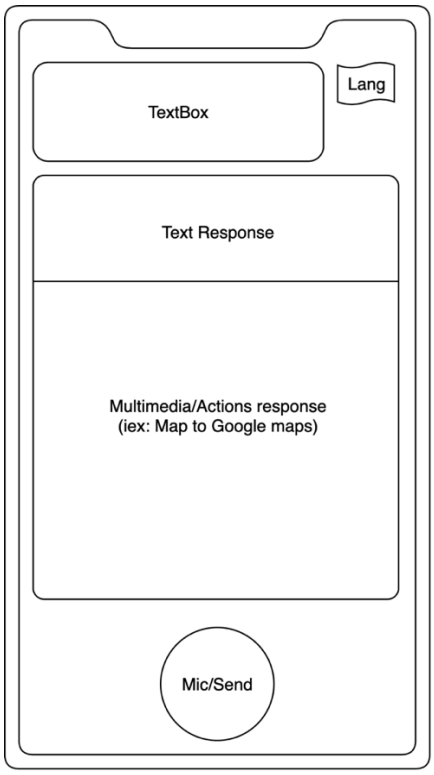


Figure 21. Sketch chatbot UI

The “microphone button” changes to a “send button” when a text is detected in the textbox, similar to other applications like Telegram or Whatsapp. The Language button changes depending of the chosen language, it has the appearance of the language’s flag. The response panel is divided in two parts when some multimedia source is needed in the response (iex: map, schedule). When the response is only text, it resizes itself.

In the Unity implementation, to make that the text look good, the plugin TextMeshPro is being used because the native text of Unity does not scale properly to high resolution screens. Another plugin, DOTween (http://dotween.demigiant.com), is used for making animations without using the Unity animator tool.

The interface of the application is designed in Unity and it is shown in the next Figure 22. In this project stage, the UI is not operational, the programming part is still missing.

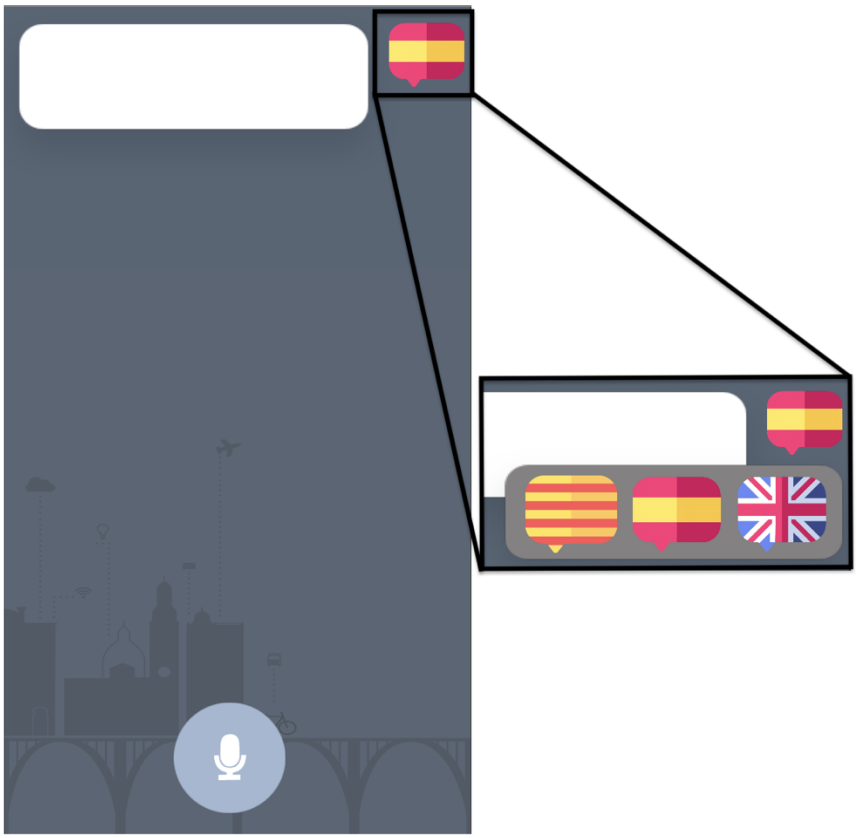


Figure 22. Chatbot UI in Unity.

Because the app is made in 2D initially, all the interface component is built over a canvas overlay. When the textbox is being used the view of the microphone button changes. The handlers of the main objects are explained in the next paragraphs. The textbox, the microphone/send button, the language button, and the response panel are explained.

The textbox works with an input field. Also, it contains a script named WriterController. The script WriterController contains two methods used as handlers, OnSelectClick() and OnDeselectClick(). OnSelectClick() method informs if it the textbox was clicked. Otherwise, the OnDeselectClick() method handles if the texbox has text, and it changes the functionality of the microphone button to send(Figure 23).

```
26      public void onSelectClick()
27      {
28          print("select");
29          sendButton.sprite = sendMessageImage;
30      }
31      public void onDeselectClick()
32      {
33          print("deselect");
34          if (gameObject.GetComponent<TMP_InputField>().text.Length == 0) {
35              sendButton.sprite = micImage;
36              micButton.setIsText(false);
37          }
38          else{
39              micButton.setIsText(true);
40          }
41      }
```

Figure 23. WriterController script.

The microphone button works in two different ways. It can work as a microphone button or as a send button.

When it works in microphone mode, it works like a push button. While the button is being hold down, an audio file starts to be recorded. When the button is released the recording stops and the audio is sent to Wit.ai through the script Master (the Master script will be explained in the Wit.ai implementation part). When it works in send mode, it works like a normal button and, when the button is released, the text is sent to Wit.ai through the Master script (Figure 24).

The working mode of the button is based on a Boolean variable. This variable is named isText and is initialized as False. IsText only changes if the textbox contains some text (39, Figure 23). All the basic functionality of the microphone button is contained in a script named OnClickMicrophone (Figure 24). The script is attached to the button and it controls the press and release events of the button (40-82, Figure 24).

```
40      private void MyOnClickDown()
41      {
42          if (imgPpla != null)
43          {
44              IsClicking = true;
45              if (!isText)//No text on textbox
46              {
47                  master.startStopRecord();//Starts recording
48                  audioSourceMic.Play();//Sound effect
49              }
50
51              //Visual feedback
52              imgPpla.rectTransform.DOSizeDelta(new Vector2(60, 60), 0.2f, true);
53              imgSombra.rectTransform.DOSizeDelta(new Vector2(68, 65), 0.2f, true);
54              imageMic.rectTransform.DOSizeDelta(new Vector2(160, 160), 0.2f, true);
55              rainbowScript.setRainbow(true);//Rainbow effect on
56          }
57          else print("Falta asignar la imagen");
58      }
59
60      private void MyOnClickUp()
61      {
62          if (imgPpla != null)
63          {
64              isClicking = false;
65              if (!isText)//No text on textbox
66              {
67                  master.startStopRecord();//Finish recording and is sent to Wit.ai
68                  audioSourceMic.Play();//Sound effect
69              }
70              else{
71                  master.sendText();
72                  audioSourceText.Play();//Sound effect
73              }
74
75              //Visual feedback
76              imgPpla.rectTransform.DOSizeDelta(new Vector2(50, 50), 0.2f, true);
77              imgSombra.rectTransform.DOSizeDelta(new Vector2(50, 50), 0.1f, true);
78              imageMic.rectTransform.DOSizeDelta(new Vector2(100, 100), 0.2f, true);
79              rainbowScript.setRainbow(false);//Rainbow effect off
80          }
81          else print("Falta asignar la imagen");
82      }
83
84      public void SetIsText(bool t){
85          isText = t;
86      }
87
```

Figure 24. OnMyClickMicrophone script.

The language button is managed by the scripts: OnClickLangButton (Figure 25) and ButtonManagement (Figure 26). OnClickLangButton manages the language panel menu and

the button views. ButtonManagement manages the language selection buttons, and it changes which app of Wit.ai should be working in the application.

```
7   public class ButtonManagment : MonoBehaviour
8   {
9       //Colors
10      private Color unselected = Color.white;
11      private Color selected = new Color(r:0.6084906f, g:0.7132255f, b:1.0f, a:1.0f);
12
13      //Buttons
14      public GameObject butonES;
15      public GameObject butonEN;
16      public GameObject butonVA;
17
18      public onClickLangButton myBtn;//Main button manager
19      public Wit3D myMaster;//Master script
20
21      //Spanish
22      public void OnClickES(){
23          myMaster.setVoiceLanguage(2);//Setting the langauge
24          myBtn.setImg(2);//Changing the main button image
25          myBtn.showPanel();//Hiding panel
26
27      }
28      //English
29      public void OnClickEN()...
35      //Catalan/Valencian
36      public void OnClickVA()...
45  }
46
```

Figure 25. ButtonManagement script.

```
6   public class OnClickLangButton : MonoBehaviour
7   {
8       public GameObject myPanel;//Panel object, languages menu
9       public Sprite[] imgs;//Images for the buttons
10      private Image myButtonImage;
11
12      private void Start()
13      {
14          myButtonImage = GetComponent<Image>();
15      }
16
17      public void ShowPanel()
18      {
19          //Hiding or showing the langauge menu
20          myPanel.SetActive(!myPanel.activeSelf);
21      }
22
23      public void SetImg(int i)
24      {
25          //Changing button image
26          switch(i){
27              case 1:
28                  myButtonImage.sprite = imgs[0];
29                  break;
30              case 2:
31                  myButtonImage.sprite = imgs[1];
32                  break;
33              case 3:
34                  myButtonImage.sprite = imgs[2];
35                  break;
36          }
37      }
38  }
39
```

Figure 26. OnClickLangButton script.

Finally, the response panel is managed by two scripts, ChatPanelController (Figure 27) and MyGoogleMapsPanelScript (Figure 28). Depending of the Wit.ai response, the script Master sends the hiding and showing commands to the panel's managers. The script ChatPanelController, manages the text part of the response (Figure 21) and, the script MyGoogleMapsPanelScript, manages the multimedia/action responses (Figure 21).

```
6   public class ChatPanelController : MonoBehaviour
7   {
8       private AudioSource audioSource;
9
10      //Literal cordinates refering to the center of the view
11      private float firstCenterX = -0.006332397f;
12      private float secondCenterX = -0.006332397f;
13      private float thirdCenterX = 0.2900009f;
14      private float fisrtInitPosX = 335f;
15      private float secondInitPosX = 335f;
16      private float thirdInitPosX = -322f;
17
18      public void ShowQuestion()
19      {
20          audioSource = GetComponent<AudioSource>();//Gets audio effect
21          DG.Tweening.DOTween.KillAll();//Cleaning stack of DOTween
22          //Showing animation
23          gameObject.GetComponent<RectTransform>()
24              .DOAnchorPos(new Vector2(firstCenterX, 200f), 2f, true);
25          audioSource.Play();
26      }
27
28      public void ShowResponse()...
33
34      public void ShowContent()...
39
40      public void HideQuestion()
41      {   //Hiding panel
42          gameObject.GetComponent<RectTransform>()
43              .DOAnchorPos(new Vector2(fisrtInitPosX, 200f), 2f, true)
44              .OnComplete(() => { gameObject.SetActive(false); });
45      }
46
47      public void HideResponse()...
54
55      public void HideContent()...
61  }
62
```

Figure 27. ChatPanelController script.

```
6   public class MyGoogleMapsPanelScript : MonoBehaviour
7   {
8       public GameObject[] categories;
9
10      //Showing panel
11      public void Show(){
12          gameObject.GetComponent<RectTransform>()
13              .DOAnchorPos(new Vector2(-5f, 24f), 2f, true);
14      }
15
16      //Hiding panel
17      public void Hide(){
18          gameObject.GetComponent<RectTransform>()
19              .DOAnchorPos(new Vector2(-326f, 24f), 2f, true)
20              .OnComplete(() => { gameObject.SetActive(false); });
21      }
22
23      //Set which multimedia panel will be shown
24      public void SetPanelType(int i)...
53  }
54
```

Figure 28. MyGoogleMapsPanelScript.

After the UI implementation, we start to implement the functionality. In the next section the interaction between Wit.ai and the application is explained.

Wit.ai implementation

In this chapter the integration with Wit.ai and Unity is explained. How the audio/text is sent to Wit.ai and, how the response is processed.

Unity works with C# and, it allows the creation of partial classes. The MasterScript was created, and it is split in three files containing the same class named MasterScript. The files are MasterScript, MasterScriptCoroutines and MasterScriptResponse.

MasterScript contains the necessary methods to manage the on click actions of the MicrophoneButton with the modes of voice and text (Figure 29, Appendix 1). MasterScriptCoroutine contains two coroutines, one to manage the text sending to Wit.ai, and another to manage the voice sending to Wit.ai (Figure 30).

```
59      void Start () ...
64      |
65      // Update used only for the voice request
66      void Update () ...
98
99      //On click MicrophoneButton mode Send
100     public void SendText() ...
104
105     //OnClickMicrophoneButton mode Voice
106     public void StartStopRecord()...
122
123     public void SetVoiceLanguage(int i)...
150
151     public void SetTextLanguage(int i)...
175 }
```

Figure 29. MasterScript methods.

```
9   public partial class MasterScript : MonoBehaviour
10  {
11      public IEnumerator SendTextCoroutine()...
83
84      public IEnumerator SendAudioCoroutine()...
145 }
146
```

Figure 30. MasterScriptCoroutine methods.

MasterScriptResponse contains all the methods needed for handling the Wi.ai response. The script contains two methods, ResponseHandler() and ExecuteAction() (Figure 31). ResponseHandler() calls the script which implements the finite state machine (Figure 3), WitAiGraph. WitAiGraph returns an object of type ResponseSolution (Figure 32) and this object is handled by ExecuteAction().

```
9   public partial class MasterScript : MonoBehaviour
10  {
11      public TextToSpeechScript textToSpeech;//textToSpeach library
12
13      private ResponseSolution solution;
14
15      void ResponseHandle(string jsonString)
16      {
17          WitAiGraph wGraph = new WitAiGraph();
18          solution = wGraph.GraphHandler(jsonString);
19          ExecuteAction(solution);
20      }//END OF HANDLE VOID
21
22      void ExecuteAction(ResponseSolution r){
23          pregunta.text = r.question;
24          //Switch between kind of actions
25          switch (r.actionType)...
42      }
43  }
```

Figure 31. MasterScriptResponse script.

```
using UnityEngine;
using System;

public class ResponseSolution : MonoBehaviour
{
    public int actionType { get; private set; }
    public String question { get; private set; }
    public String[] answer { get; private set; }

    public ResponseSolution(int actionType, String question, String[] answer)
    {
        this.actionType = actionType;
        this.question = question;
        this.answer = answer;
    }
}
```

Figure 32. ResponseSolution object.

ExecuteAction() implements the communication between the solution and the UI, it is the arrow “response generated” in the Figure 20. Three kind of actions can be seen in the UI: only text, only multimedia or text and multimedia.

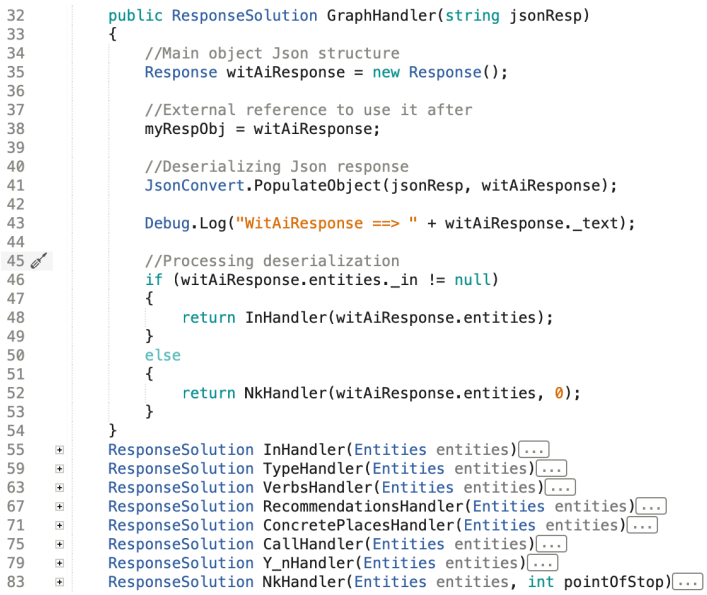
The most important script here is WitAiGraph (Appendix 3). This script is the most important of that part because it deserializes the response of Wit.ai and, it makes possible to generate a good response.

Wit.ai sends a response using JSON. To deserialize the JSON object is being used Json .NET for Unity (Json.net for Unity, 2016). This library allows to deserialize easily the response. With this library, the script which deserialize the Json object must contain one public class.

The Wit.ai’s json structure is built in an object containing three parameters. These parameters are: two strings, _text and msg_id, and one object, entities. The entities object is a list of entity objects. It contains as many entity objects as entities the sentence has.

The entity object has four parameters, which are: suggested, confidence, value, type. Then, in WitAiGraph must exist one public class for the main object, another for the entities object and one by each entity in the Wit.ai app. The complete WitAiGraph script can be seen in the Appendix 3.

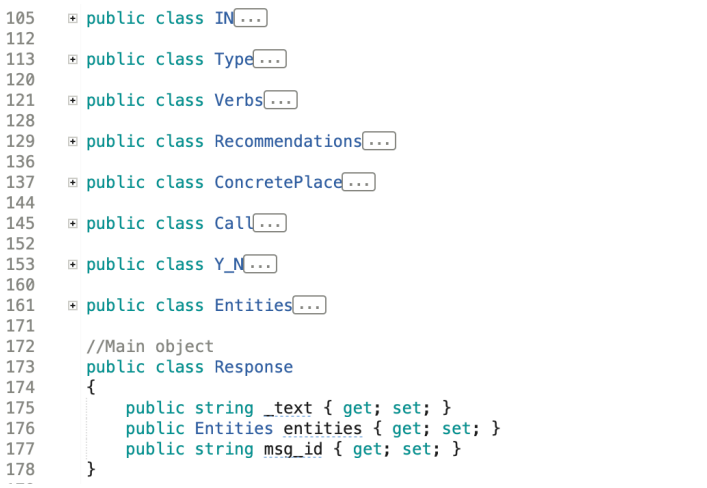
When the public classes are created, the Wit.ai response can be deserialized passing the main class of to the main Json’s object, and the output to the method PopulateObject. This method is contained inside JsonConvert (41, Figure 33).



```
32      public ResponseSolution GraphHandler(string jsonResp)
33      {
34          //Main object Json structure
35          Response witAiResponse = new Response();
36
37          //External reference to use it after
38          myRespObj = witAiResponse;
39
40          //Deserializing Json response
41          JsonConvert.PopulateObject(jsonResp, witAiResponse);
42
43          Debug.Log("WitAiResponse ==> " + witAiResponse._text);
44
45          //Processing deserialization
46          if (witAiResponse.entities._in != null)
47          {
48              return InHandler(witAiResponse.entities);
49          }
50          else
51          {
52              return NkHandler(witAiResponse.entities, 0);
53          }
54      }
55      ResponseSolution InHandler(Entities entities)...
59      ResponseSolution TypeHandler(Entities entities)...
63      ResponseSolution VerbsHandler(Entities entities)...
67      ResponseSolution RecommendationsHandler(Entities entities)...
71      ResponseSolution ConcretePlacesHandler(Entities entities)...
75      ResponseSolution CallHandler(Entities entities)...
79      ResponseSolution Y_nHandler(Entities entities)...
83      ResponseSolution NkHandler(Entities entities, int pointOfStop)...
```

Figure 33. Part of WitAiGraph script.

Then, as has been said before, WitAiGraph contains the main class WitAiGraph, and the deserialization classes. These are: 7 classes for the entities, one for the entities object, and one for the parent object (Figure 34).



```
105     public class IN...
112
113     public class Type...
120
121     public class Verbs...
128
129     public class Recommendations...
136
137     public class ConcretePlace...
144
145     public class Call...
152
153     public class Y_N...
160
161     public class Entities...
171
172     //Main object
173     public class Response
174     {
175         public string _text { get; set; }
176         public Entities entities { get; set; }
177         public string msg_id { get; set; }
178     }
```

Figure 34. Part of WitAiGraph script.

Finally, when the Wit.ai response has been deserialized, the response must be processed. This process implements the finite state machine (Figure 3). It has one method for each state. In each method the answer is generated and a transition is made.

The Figure35 is an example about how the state IN is handled in WitAiGraph. It must be noted that the values of the entity must be put literally. On entities like ConcretePlaces or Verbs, the data is taken from a real database. There are a foreach because the response can contain repeated entities.

```
55      ResponseSolution InHandler(Entities entities)
56      {
57          foreach(IN i in entities._in)
58          {
59              switch (i.value)
60              {
61                  case "nk":
62                      return NkHandler(entities, 0);
63                  case "y/n":
64                      if (entities.y_n != null) return Y_nHandler(entities);
65                      return NkHandler(entities, 0);
66                  case "question":
67                      if (entities.type != null) return TypeHandler(entities);
68                      return NkHandler(entities, 1);//Will check verbs and cPlaces
69                  case "order":
70                      if (entities.type != null) return TypeHandler(entities);
71                      return NkHandler(entities, 1);//Will check verbs and cPlaces
72                  default:
73                      return NkHandler(entities, 0);
74              }
75          }
76          return NkHandler(entities, 0);
77      }
```

Figure 35. Part of WitAiGraph script.

Finally, when everything is implemented is time to test. In the next chapter testing of the app is explained.

4.3.4 Test

In this chapter a short view across the versions of the application is explained. The first implementation was Alpha version. After this, the application was tested outside the development environment, and this generated more flaws in the UI design.

Alpha version

In this chapter the 1st version of the application is presented. The first UI was based on the Whatsapp interface (Figure 36). It had more brilliant colors, and response panels which show the response to the user.

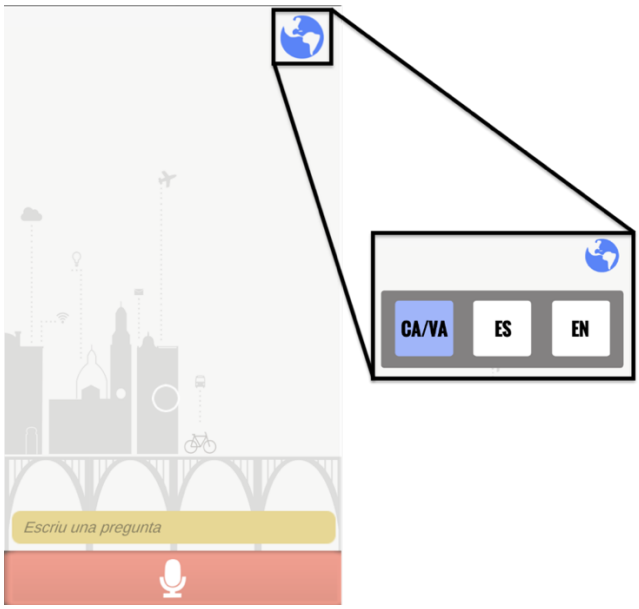


Figure 36. UI Alpha version.

In this version, another kind of finite state machine was implemented. The finite state machine was not optimal. It had some errors caused by its complexity. The finite state machine was very complex because the Wit.ai's apps had 19 entities.

In the Figure 37 the finite state machine used for the alpha version can be seen. The empty nodes in the finite state machine are swap nodes. This graph causes problems when the NLP system of Wit.ai was trained and implemented inside the application. The complexity of the graph was one of the problems in the NLP system training. The examples needed for training the NLP system are not affordable in a short term.

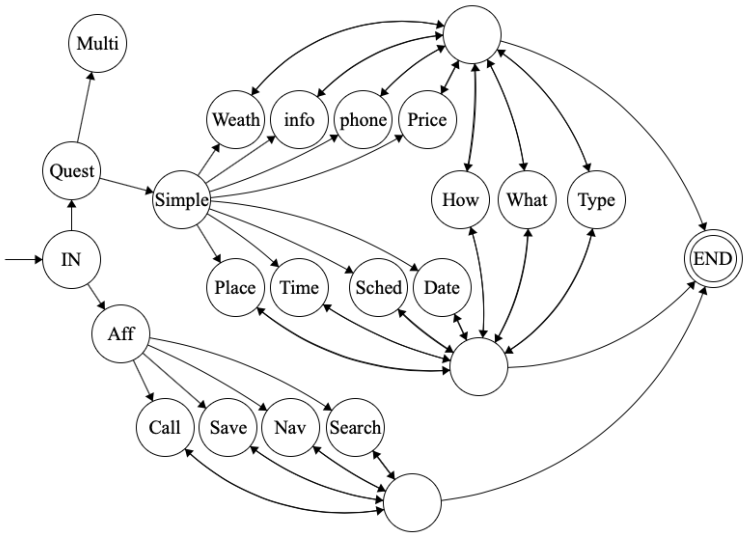


Figure 37 Finite state machine alpha version.

The underfitting was one of the reasons to change the graph. This graph in short term lacks training. The new graph solved this problem because the quantity of samples needed for training less than previously.

The implementation of the structure in the application was another problem. The complexity made the code longer.

Other findings in the testing were about the language and microphone buttons. The problem with the language button was that, it did not show which language was chosen.

About the microphone button were two issues, one about its interaction and another about its aspect. About the interaction, it was reported that sometimes, the touch interaction did not work correctly. It was caused by the Unity touch handler. About the aspect, it was reported that the button was too small for a normal interaction. The beta version solved all these problems.

Beta version

The beta version solves the problems found in the Alpha version and a new UI was made. It can be seen in the Figure 22. This version is a good candidate for the final version.

The new UI solved the problems with the microphone button, and it works now perfectly. Also, the new UI solved the language button problem, and now the languages button changes depending of the language.

The other problem seen in the alpha version was the graph. In the beta version the graph has been simplified and now it has only 7 states (Figure 22) against 18 states of the alpha version graph (Figure 37).

In the old graph, all the entities were trait type. In the new graph, only one entity has trait strategy and the others use different strategies. This makes easier the training of the NLP System and the implementation of the graph in the application. Also, thanks to the simplification of the graph, the complexity of processing and training has decreased exponentially.

Finally, when the beta version was made and it solved the alpha version problems, the test with external persons started. Based on these tests, it was decided whether the beta version was the final or not.

Final version

The final version only has changes the server part. The functionality of the beta version was good and, the programming part that implemented the finite state machine very good working very fine.

The beta version was given to 5 persons. These persons were testing the application 5 days, and each one of them made their own tests to the application. The reports were positive. The problems with the UI were solved and the response feedback was good.

With these reports, the beta version of the application was turned in the final version. The changes in this version are in the Wit.ai platform, and there were only some small problems with the response in concrete cases.

Improving

There are some things to improve in future versions of the applications. The language recognition sometimes faults and must be improved. The range of questions the app is able to response is limited and must be improved. The UI always can be improved and probably will be improved in future versions. And the range of actions must be improved, like for example: to call someone by phone, to save some event in the calendar, to response to sentences like “tell me a joke”, etc.

In a future, if the application receives some funds or similar, Wit.ai can be replaced by some other service which brings better tools, like streaming audio input or better time of response.

4.4 Auto-learning implementation

In this chapter implementation of the wit.ai rest API is explained. The program is used in the process of training and improving of the chatbot’s NLP system. It allows upgrades the chatbot without changing the front-end, only the back-end (Wit.ai).

This program implements all the rest API of Wit.ai. The program allows to train and improve the performance and behavior in the chatbot without use the Wit.ai’s UI. Thanks to that, the insertion of data in Wit.ai was easier than the normal way, by hand.

The program was built with javaFX because it is a fast environment for developing simple programs. The program has a simple interface (Figure 38), with two check list, one table and one button.

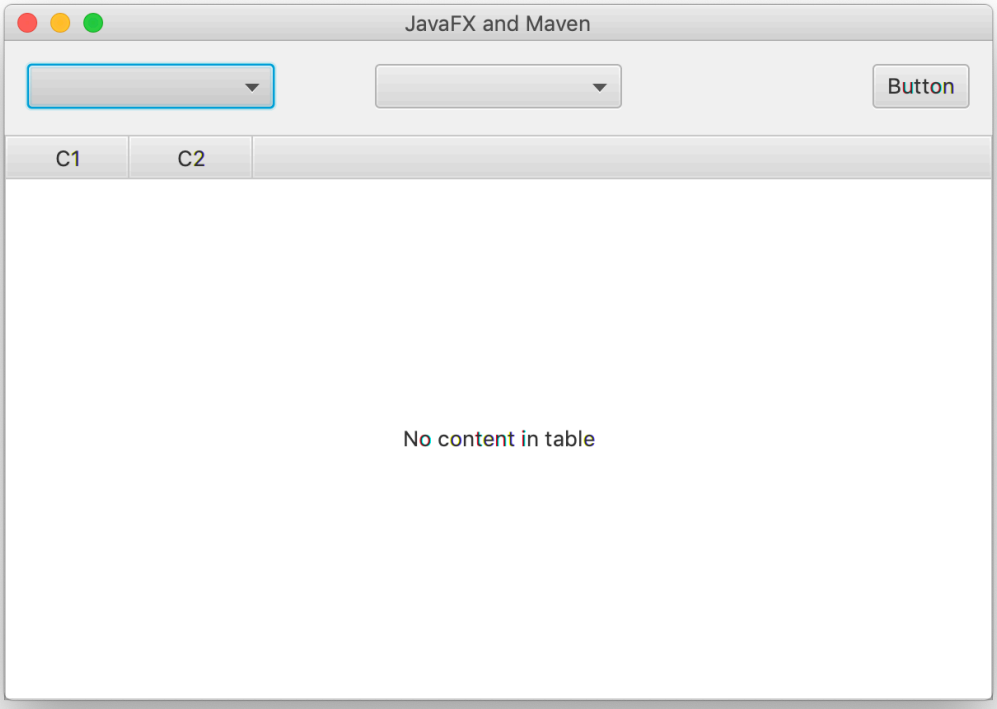


Figure 38 Training program UI

With the check list, user is able to execute one request at time choosing the entity with the options. The table takes data for the request and, if needed, it shows the response. The button only executes the selected request.

The addition of auto-learning makes the application faster and easier to maintain since only the back end needs to be upgraded. If the back-end changes in the future, the front-end can be upgraded.

4.5 Final version

As in the chapter 4.3.4 was said, the final version was born from the beta version. This version was tested and, after some changes in the NLP System, it was ready to be the final version.

In this version, the UI has been improved from the beta version (Figure 22), and it has some visual changes. The microphone button has been animated when the button is pressed. The animation involves a rainbow effect with the sound while the button is being press.

Also, the response panel has been animated. When the response panel appears, it appears sliding from the right part of the screen, if it contains only text, and from the left, if it contains multimedia information.

In the Wit.ai part, that version has some changes focused in the efficiency of the Wit.ai's response. Then, some adjustments have been done in Wit.ai using the autolearning program (chapter 4.4).

Finally, this version of the application is going to be released public and, it will be published in the Apple Store and in Google Play. The process of publishing will be done in less than a month after publishing this document.

4.6 Future improvements

While chapter 4.1 explained the general case of the chatbot, here the Alcoi implementation is discussed. In the city of Alcoi, thanks to "Universitat Politècnica de València - Campus d'Alcoi", the city council of Alcoi was met and, it was agreed that the city promotes the chatbot through big touch screens in the street and, through the application.

Then, thanks to the agreement with the Alcoi's city council, the application is going to be promoted through the city and, for sure, some changes in the UI will be done. The new interface will be adapted to a new way of interaction because it passes from a smartphone screen to a 100-inch screen.

This future project is going to be managed together with other business. Some of these businesses bring us a big-data environment for training the application. That new project is already started as bifurcation of the chatbot and, it is estimated to finish it before September of 2019.

5 CONCLUSIONS

Developing an application which offers a better service than other applications like Google Assistant or Siri was the target of the thesis. While developing the main objective, other goals appeared like how digital assistants work using an NLP system, the importance of UIs and how cost-benefit analysis affects the development process.

For working in the cities of Alcoi and Lahti a field study was made but, only the information of Alcoi has been implemented in the application so far because it was more consistent and precise than Lahti's.

Several UIs along the project were designed and those user experience cualuated. Some problems occurred in first versions like wrong feedback or wrong behavior. These and other errors were fixed in the following version.

The NLP system implementation was the crucial part of the project. Two versions were made where the first version followed a complex strategy, which caused undesired behavior. The second, more simplified version brought the designed behavior.

Finally, some aspects were affected by the cost-benefit analysis. Some of these aspects are the information about Lahti in the application and the NPL system implementation.

This project is affordable if it is focused in concrete places. If the focus is every place, it means a general purpose focus, and it is completely not affordable for only one person. The time of development was a bit longer than estimated, but it was due for the inexperience in this area of development.

Concluding, the main goal of the project was achieved, and the project was finished successfully. This project is still under development and will implement future improvements like Lahti's data and the features stated in the Alcoi's agreement.

REFERENCES

Arora, P. 2018. Different types of Natural Language Processing. Available at: https://www.quora.com

Bhardwaj, A., Di, W., & Wei, J., 2018. Deep Learning Essentials: Your Hands-On Guide to the Fundamentals of Deep Learning and Neural Network Modeling. Birmingham: Packt Publishing Ltd []. Available at: https://www.kdnuggets.com

Bourque, P., R, E., D, F., 2004. SWEBOK Guide V3.0. Available at: www.swebok.org

Google, 2019. Dialogflow web page. Available at: https://dialogflow.com

IBM systems, 2019. IBM web page. Available at: https://www.ibm.com/watson

JGraph Ltd.,2019. Draw.io [accessed 5 Apr 2019]. Available at: https://www.draw.io

Munster, G. 2018. We Ran HomePod Through the Smart Speaker Gauntlet. Available at: https://loupventures.com

Pages 506-516. Automatic Online Evaluation of Intelligent Assistants [referenced 5.18.2015]. Proceedings of the 24th International Conference on World Wide Web. Available at: https://dl.acm.org

ParentElement, LLC, 2016. Json .NET for Unity [accessed 20 Jan 2019]. Available at: https://www.newtonsoft.com

Seif, G. 2018. An easy introduction to Natural Language Processing. Available at: https://towardsdatascience.com

Wallace, E. 2010. Finite State Machine Designer. Available at: http://madebyevan.com

Wit.ai, Inc, 2019. Wit.ai web page. Available at: https://Wit.ai

Wikipedia, 2019a. Artificial intelligence. Available at: https://en.wikipedia.org

Wikipedia, 2019b. Deep Learning. Available at: https://en.wikipedia.org

Wikipedia, 2019c. Natural language processing. Available at: https://en.wikipedia.org

APPENDICES

Appendix 1 MasterScript

```
21      using UnityEngine;
22      using System.Collections;
23      using System;
24      using System.Collections.Generic;
25      using System.IO;
26      using UnityEngine.UI;
27      using TMPro;
28      using System.Text.RegularExpressions;
29      using UnityEngine.EventSystems;
30
31      public partial class MasterScript : MonoBehaviour
32      {
33
34          //Audio record for wit.ai
35          public AudioClip commandClip;
36
37          //Audio sample rate
38          int samplerate;
39
40          /*
41           * Is used in response
42           * 0 = En
43           * 1 = Es
44           * 2 = Ca
45           * deffault in english
46          */
47          int languageIndex = 0;
48
49          //public valiables
50          public GameObject loadingAnimation;
51          public GameObject answerPanel;
52          public GameObject myGoogleMapsPanelGO;
53
54          public TMP_InputField imputField;
55          public TextMeshProUGUI pregunta;
56
57          //private variables
58          private TextMeshProUGUI answerTextInPanel;
59
60          // API access parameters
61          string urlVoice = "https://api.wit.ai/speech?v=20190511";
62          string urlText = "https://api.wit.ai/message?v=20190511&q=";
63          //Default in english
64          public string token = "55TCHMXTMCCXQAB6GRUYNTPYERLY7ABN";
65
66          //Bolean variables
67          private bool isRecording = false;
68          private bool pressedButton = false;
69
70          void Start()
71          {
72              // set samplerate to 16000 for wit.ai
73              samplerate = 16000;
74              answerTextInPanel = answerPanel.GetComponentInChildren<TextMeshProUGUI>();
75          }
76
```

```
76
77          // Update used only for the voice request
78          void Update()
79          {
80              if (pressedButton == true)
81              {
82                  pressedButton = false;
83                  if (isRecording)
84                  {
85                      Debug.Log("Listening for command...");
86                      //Start recording (rewriting older recordings)
87                      commandClip = Microphone.Start(
88                          null,
89                          false,
90                          5,
91                          samplerate
92                      );
93                  }
94
95                  //When the user finish the voice querie
96                  if (!isRecording)
97                  {
98                      Debug.Log("Saving Voice Request...");
99                      // Save the audio file
100                     Microphone.End(null);
101                     if (SavWav.Save("sample", commandClip))
102                     {
103                         Debug.Log("Saving to wave file...");
104                     }
105                     else
106                     {
107                         Debug.Log("Saving FAILED!!!");
108                     }
109
110                     //Delete the audio
111                     commandClip = null;
112
113                     //Start to send to wit.ai
114                     StartCoroutine(SendAudioCoroutine());
115                 }
116             }
117         }
118
119         //On click MicrophoneButton mode Send
120         public void SendText()
121         {
122             StartCoroutine(SendTextCoroutine());
123         }
```

```
124
125         //OnClickMicrophoneButton mode Voice
126         public void StartStopRecord()
127         {
128             if (isRecording == true)
129             {
130                 pressedButton = true;
131                 isRecording = false;
132                 print("ButtonUp");
133             }
134             else if (isRecording == false)
135             {
136                 isRecording = true;
137                 pressedButton = true;
138                 print("ButtonDown");
139             }
140
141         }
142
143         public void SetVoiceLanguage(int i)
144         {
145             string theDate = System.DateTime.Now.ToString("yyyyMMdd");
146             Debug.Log("The date ==> " + theDate);
147             urlVoice = "https://api.wit.ai/speech?v=" + theDate;
148             switch (i)
149             {
150                 case 1:
151                     //Valencià
152                     token = "JK4HF63SQVGBZYPDYXSSYV6K6HEQ2HNX";
153                     languageIndex = 2;
154                     break;
155                 case 2:
156                     //Castellà
157                     token = "QMWMGXIAVTHRRCAH6MYYI2WUYYEMKVJV";
158                     languageIndex = 1;
159                     break;
160                 case 3:
161                     //Anglés
162                     token = "55TCHMXTMCCXQAB6GRUYNTPYERLY7ABN";
163                     languageIndex = 0;
164                     break;
165                 default:
166                     token = "55TCHMXTMCCXQAB6GRUYNTPYERLY7ABN";
167                     languageIndex = 0;
168                     break;
169             }
170         }
171     }
```