Android App for the elderly

Geoffrey Manuel dos Santos Marcelino

Dissertation submitted to obtain the degree in
Master in Electrical and Computer Engineering

Supervisor: Professor João Paulo Baptista de Carvalho

Examination Committee
Chairperson: Prof. Nuno Cavaco Gomes Horta
Supervisor: Professor João Paulo Baptista de Carvalho
Member of the Committee: Prof. João Nuno de Oliveira e Silva

October 2014
“If you wish to make an apple pie from scratch, you must first invent the universe”

Carl Sagan
Acknowledgments

First of all, I would like to thank my family and my friends.

A special thank you to Professor João Paulo Carvalho, who attended my desire to work in this field, and made sure that I had access to everything I needed throughout this project. I wish to express my sincere gratitude to Pedro Fialho and Alberto Abad who helped me with the critical features of the project.

Finally I also want to thank all the people that directly or not helped me through this phase of my life.
Abstract

Human memory plays a crucial role in our lives, it is responsible for storing all data that enable us to perform from the simplest to the hardest task. The memory can be divided into two types: Short-term memory (STM), also known as working memory, and long term memory (LTM). The major distinction between the two is that STM enables us to, for example, remember a simple shop list; whereas LTM is where it is possible to store a language or more. With aging, adults tend to lose STM capacity.

It has been proved that STM memory as part of our brain can be trained and improved like a muscle, through memory stimulation, like remembering events, names, etc. This stimulation can be achieved via the human sensory system, mostly vision or audition. Knowing this, there are multiple solutions to help people train their memory, but it is mostly only through visual stimuli, almost always lacking auditory and voice interaction.

This project takes these into account trying to offer in a game a wide range of stimuli where the voice will function as an integral part of the player's interaction. A whole experience to make people remember visual and audible contents and interact in the game orally.

Keywords

Serious game
Mobile game development
Web-service
Automatic Speech Recognition
Text-to-speech
Resumo

A memória é parte crucial na qualidade de vida humana, responsável por armazenar dados que possibilitam a realização desde pequenas tarefas até às mais complexas.

A memória é composta por dois principais tipos: memória de curto prazo e memória de longo prazo. Como o nome sugere a memória de curto prazo dura poucos segundos ou minutos e permite memorizar pequenos eventos tal como o que comemos ao pequeno almoço ou uma lista de compras, já a memória de longa duração permite por exemplo memorizar uma língua tal como o português. Com a idade, a memória de curta duração tende a perder as suas capacidades.

Diversos estudos tem comprovado que a memória de curta duração pode ser treinada através de exercícios de memorização tal como recordar, eventos, nomes, etc. Esta estimulação da memória é feita através dos sentidos, tal como a visão e a audição. Com base neste conhecimento existem inúmeras soluções que permitem estimular a memória contudo a grande maioria tende a focar-se apenas na estimulação através da visão deixando geralmente de parte a audição e interacção vocal por parte do jogador.

Este projecto propõe realizar um jogo de memória que através de estimulos auditivos e visuais e utilizando a voz como principal forma de interacção oferece uma nova experiência para um jogo que estimule a memória.

Palavras-chave

Serious game
Mobile game development
Web-service
Automatic Speech Recognition
Text-to-speech
# Contents

1Introduction..................................................................................................................1
  1.1Context and Motivation..........................................................................................1
  1.2Objectives.............................................................................................................2
  1.3Purposed project....................................................................................................2
  1.4Original Contributions.........................................................................................4
  1.5Thesis Outline......................................................................................................5

2Mobile Game Development..........................................................................................7
  2.1Computing Platforms............................................................................................7
    2.1.1OpenGL ES or Framework..............................................................................9
    2.1.22D Game Engines Approach......................................................................11
  2.2Development Tools..............................................................................................11
    2.2.1SDK.............................................................................................................12
    2.2.2Programming Language...............................................................................12
    2.2.3Game Engine...............................................................................................13
    2.2.4IDE.............................................................................................................15
    2.2.5DRCS..........................................................................................................16
    2.2.6Testing Devices............................................................................................17

3State of the art.............................................................................................................19

4Web-Service development...........................................................................................21
  4.1Development Tools..............................................................................................21
    4.1.1Web application Framework.........................................................................21
    4.1.2Programming Language...............................................................................22
    4.1.3Database......................................................................................................23
    4.1.4IDE.............................................................................................................23
    4.1.5DRCS..........................................................................................................24
  4.2Third party servers...............................................................................................24
    4.2.1Authorization provider..................................................................................24
    4.2.2ASR and TTS...............................................................................................25

5Design solution............................................................................................................27
  5.1Application architecture.......................................................................................27
  5.2Web-server...........................................................................................................28
    5.2.1Web-Server Architecture.............................................................................28
5.2.2 Database design ................................................................. 31
5.2.3 Authentication ................................................................. 33
5.2.4 API Developed ................................................................. 36
5.2.5 TTS ........................................................................... 40
5.2.6 ASR ............................................................................. 41
5.2.7 Performance specifications ............................................. 42

5.3 Mobile Application .......................................................... 43
  5.3.1 Modules configuration .................................................. 43
  5.3.2 Managers ................................................................... 44
  5.3.3 Audio ......................................................................... 47
  5.3.4 Factory ....................................................................... 48

6 Gamification ........................................................................ 51
  6.1 Game Logic and Rules ...................................................... 51
  6.2 Levels ............................................................................ 53
  6.3 Scoring ........................................................................... 55
  6.4 Scenes and layers ............................................................. 56
    6.4.1 Splash ...................................................................... 56
    6.4.2 Language menu ......................................................... 57
    6.4.3 Main Menu ................................................................ 58
    6.4.4 Loading .................................................................... 59
    6.4.5 Game Scene ............................................................. 60

7 Survey Results ..................................................................... 65
  7.1 Survey design ................................................................. 65
  7.2 Results and analysis ....................................................... 66

8 Conclusions and Future Works ........................................... 71
List of Figures

Figure 1.1-Relation between STM and LTM................................................................. 1
Figure 1.2-Order placement.......................................................................................... 3
Figure 1.3-Returned client speaking........................................................................... 3
Figure 1.4-"Call client" game type............................................................................... 4
Figure 1.5- Game “Memory Trainer”[5]........................................................................ 5
Figure 1.6- Game "Animals Memory Game “[6]......................................................... 5
Figure 2.1-Global computing platform market share................................................. 8
Figure 2.2-OpenGL ES interface.................................................................................. 9
Figure 2.3- DIY vs Framework iron triangle.............................................................. 10
Figure 2.4-The Android SDK Manager........................................................................ 12
Figure 2.5- Java compile process in the left, Android in the right handside............... 13
Figure 2.6- AndEngine lifecycle.................................................................................. 15
Figure 2.7-Android-Studio, currently in version 0.8.4............................................... 16
Figure 2.8-Git Data Transport Commands.................................................................... 17
Figure 3.1-HearBuilder Auditory Memory................................................................. 19
Figure 3.2-Mimemo by binariq................................................................................... 19
Figure 4.1-Play Framework’s MCV architecture implementation............................ 22
Figure 4.2-IntelliJ IDEA, currently in version 13, with Play 2 plug-in......................... 23
Figure 4.3-Database tools from IntelliJ IDEA............................................................ 24
Figure 5.1-Main architecture and principal components............................................ 27
Figure 5.2-Data flow in server architecture............................................................... 29
Figure 5.3-Talking Memory database schema............................................................. 32
Figure 5.4: Authentication flow with all components.................................................. 34
Figure 5.5-Consent screen.......................................................................................... 35
Figure 5.6-Communication between Talking Memory server and TTS...................... 41
Figure 5.7-Communication between Talking Memory Server and ASR.................... 42
Figure 5.8-Scene and resources managers interaction................................................ 45
List of Tables

Table 1-Smartphones specifications.................................................................................................................. 18
Table 2- Scoring system........................................................................................................................................ 56
List of Charts

Chart 7.1-Most used language during game play..........................................................66
Chart 7.2-Statements about game challenge.................................................................67
Chart 7.3-ASR hits.......................................................................................................68
Chart 7.4-Game type preference..................................................................................69
Chart 7.5-Net Promoter Score.....................................................................................70
List of Acronyms

STM - Short term Memory
LTM - Long Term Memory
TTS - Text-to-speech
ASR - Automatic Speech Recognizer
OS - Operating System
RIM - Research In Motion
BSD - Barkeley Software Distribution
API - Application Programming Interface
DIY - Do It Yourself
UX - User Experience
SDK - Software Development Kit
GPU - Graphics Processing Unit
IDE - Integrated Development Environment
ADT - Android Developer Tools
DRCS - Distribution Revision Control System
CPU - Central Processing Unit
RAM - Random-Access Memory
SBT - Scala Built Tool
HTTP - Hyper Text Transfer Protocol
JDBC - Java Database Connectivity
XML - Extensible Markup Language
HTML - Hyper-Text Markup Language
MVC - Model-View-Controller
JVM - Java Virtual Machine
SQL - Structured Query Language
ORDBMS - Object-Relational Database Management System
CRUD - Create-Read-Update-Delete
URL - Uniform Resource Locator

JSON - JavaScript Object Notation

ANORM - Anorm Is Not An Object Relational Mapper

ID - Identifier

IO - Input-Output

HZ - Hertz

PCM - Pulse-Code Modulation

NPC - Non-player character

HUD - Heads-Up Display

NPS - Net Promoter Score
Chapter 1

1 Introduction

This chapter gives an overview on the main goals of this thesis. It explains the objectives and motivations that guided the development of this project with the structure of this work presented at the end of it.

1.1 Context and Motivation

In the adult stage, cognitive capacities tend to decrease through the years, especially short-term memory, incurring memory loss and sometimes leading to Alzheimer's disease.

The memory can be divided in two major groups, short-term memory (STM) and long-term memory (LTM). The first one has a duration of approximately 1 min, while the long-term one has a life-time duration. A stimuli is perceived by the human sensory system, then stored primarily on the STM before going to the LTM, this is achieved through a mechanisms called consolidation which is basically the process of stabilizing a memory after the initial acquisition. Repetition is an example of how information can be transfered to LTM[1]. STM is a crucial part of the whole memory and can be trained by practicing the recalling of events. The Figure 1.1 below shows what was described earlier.

Figure 1.1-Relation between STM and LTM

1http://www.kcse-online.info/schools/science/page286.htm
Brain stimulation of any kind is essential to prevent and minimize this side effect of aging. Memory stimulation is perceived in his grand majority by visual and auditory inputs and can be achieved through exercises such as learning new things, gamification of tasks that involve remembering, and so on[2]. The last suggestion is well known and usually the most common, accessible and fun, with tile-based games, object recognition, number recalling and much more.

With the “boom” in the new generation of smartphones and networking capabilities, new horizons are revealed. The tremendous variety of apps and mobile games is proof that exploring the smartphone’s hardware capabilities with a server side support can provide all kinds of mobile experiences. In particular, there is a great offer of memory games for these devices. However, the majority of these games only offer visual stimuli excluding all kinds of auditory integration or more precisely voice recalling. Although this is an indoubtable source of information, few games explore this concept, even the desktop or browser platforms.

Having has main goal the improvement of memory through visual but foremost auditory stimulation, and using the mobile devices as an extra motivation to enjoy this game in a practical and accessible way, this idea appeared as a proof of concept for a new type of memory games.

1.2 Objectives

This thesis proposes a mobile game for the Android platform[3], more precisely a memory game that defies the memorization and attention span of the user. Increased levels of difficulty challenge the player. A leaderboard helps him keep track of his improvements and compete in an asynchronous way with other players that share the same device.

As already said, the main objective of this game is to stimulate the user's STM memory by using visual and auditory stimuli. To accomplish this, the audio interaction between player and game has to be the main focus. This trait is implemented using Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) features. As the name suggests ASR is the translation of spoken words into text. The TTS converts text language into speech. Both have to be installed on a remote server and accessed through a network connection due to the overhead processing of these tasks and the lack of a mobile release.

1.3 Purposed project

The idea is to simulate the tasks and memory challenges of a bartender, which envolves memorization of faces, names and the client's orders. The goal was to gamify this environment.

In detail, the purposed game simulates a bar near the beach, where the user plays the role of a bartender. Here, several clients come, one at a time and place an order Figure 1.2.
This order placement is made by speech, and to do so the talk-to-speech feature is used. After this, the clients move on to make room for other clients. More difficult is the level more clients come to place an order.

The orders are usual products that are sold on in bar, such as drinks and snacks. After all the orders have been made, there are two possibilities to deliver the products according to the game type. In one of them the clients return to the scene, in a non particular sequence as shown in Figure 1.3, and the user must recall the name or the order orally. For this to be possible the automatic speech recognizer delivered by the implemented server is used. In the second one the orders appear on the balcony, textually, and it is up to the player to call the corresponding client to perform the delivery see Figure 1.4. Again this is only possible due to the ASR.

For each delivery, a score is calculated according to its accuracy. Wrong answers penalize the score. The number of clients and products per order starts increasing after each batch of clients as already mentioned. A batch represents a level. The audio input is controlled by a push-to-talk button which has three states.

Figure 1.2-Order placement

Figure 1.3-Returned client speaking
To conclude, the main goal is to create a game in which the non-player characters place an order with the player. To fulfill the objective of the game, the latter must remember and repeat orally what has been said according to the level rule.

![Game Screenshot](image)

Figure 1.4-“Call client” game type

To build this game with the orally and auditory features this resources must be accessed by a web server. This web server is a crucial part for the development of this project as it is responsible to deliver this services and that why it was implemented along side with the mobile game.

1.4 Original Contributions

There are divergent studies in the scientific community about which are the best approaches to effectively exercise the memory but in general using audio and visual stimula together tends to be more effective[4]. This thesis is trying to give its own contribution to those approaches, using an hybrid one, where visual and auditory stimula work together. This is achieved using visual elements, and audio interaction using TTS and ASR. These last ones are rarely used because of their complexity. As an example the top memory games available in the “Google Play Store” are:
The game presented on the Figure 1.5 has multiple modes, but the majority focuses on remembering sequences of figures and recall them. The one presented on the right hand-side is the most common game type, consisting in matching the tiles with the same figures remembering their positions.

More about the state of the art of the auditory memory game is explained in the chapter 3, State of the art.

1.5 Thesis Outline

This document is composed of eight chapters, explaining the pretended and achieved goals of this thesis and also the possible future development of this project.

The first chapter, the current one, introduces the main objectives, motivations and contributions of this project.

Chapter two will introduce the concepts used in the game development of this work and also describe the tools used. It also includes a brief comparison between similar solutions in order to justify why they fit this work.

The third chapter has the same goal as the second one but applied to the server side tools used, including the database and the third party servers.

The next chapter, four, describes all major steps and designed solutions implemented in both mobile and server-side. The ASR and TTS are also explained in this part.
In the fifth chapter, the focus will go toward the game logic, including a diagram with the game flow, the scoring, the scenes created, and the mathematical model used to generate the levels of difficulty.

The chapter number six will show statistically the level of satisfaction of people who played the game.

The overall conclusions of this thesis are presented in the last chapter, the seventh, where some features that could be developed in the future are also discussed.
Chapter 2

2 Mobile Game Development

Mobile game development is a category of mobile application development that, as the name indicates, focuses on creating games targeting mobile devices. What differentiates a game from a common App is the manner in which they interact with the user. Normally an App offers a service related with general productivity, problem solving or display of useful information to the user. A game challenges the player to fulfill objectives according to the game's rules, rewarding him according to the achievement of the objectives.

The game development, especially for mobile devices, is a challenge because of the resources consumptions of this type of software, which is needed to perform heavy animations, intense user interaction and background calculations. Fortunately, nowadays there are plenty of frameworks and game engines that simplify and optimize this process. In this chapter some of them including some advantages and drawbacks will be discussed.

Before embracing the game development itself, this chapter introduces the major computing platforms and the reasons for going mobile.

2.1 Computing Platforms

The major computing platform can be divided in five categories as shown on Figure 2.1. Those are: Android, Apple’s IOS, Microsoft's Windows, including the desktop and mobile distributions, RIM's Blackberry OS and finally the “Other” for the remaining OSs, as Linux, BSD, and so on.

Before going further into the OS comparison itself, one thing one has to observe is that mobile platforms are destroying the personal computer domination. In only four years the numbers have changed a lot: Android making mobile OSs detains now more than 70% of the market share, a quote belonging previously to Microsoft's Windows.

Diving into the mobile OS platforms, the analysis is quite simple, even though Apple’s IOS is growing, it remains minor compared to Android OS’s rise. The understanding of this phenomena isn’t deterministic but is somehow related to hardware diversity. Making manufacturers compete against each other to offer the best experience to Android users.

2 http://www.businessinsider.com/windows-monopoly-is-getting-destroyed-2013-7
The open source principle of Android gives freedom to manufacturers and developers to do what they want. This is proportional to the developer's community. Within a larger community, more developers tend to share their work and experience, build libs and frameworks that will help newcomers to easily break into the development process and increase productivity. Another major reason frequently pointed at, is the amount of support that Google is giving to Android. A range of free applications, a huge Apps market and all the events and meetups that they organize show that the project is strong, encouraging manufacturers, developers and consumers to use this platform.

Last but not least, and sometimes pointed as the main reason for Android's ascension is the cost! Due to its open source principle, it is cheap to develop and a huge range of prices makes it relatively cheap to buy a smartphone that runs Android. Actually, no other platform for mobile devices offers the same conditions as Android and probably justifies why it rules today's computing platform market[12].

For this project, according to a previous market analysis and a personal curiosity to work on this field, the choice was the mobile platform. With the power that mobile devices have nowadays, there's no reason, whenever possible, not to go mobile.

What was mentioned previously made Android platform the right choice, especially thanks to the community that offers plenty of support and very useful libs. There are also solutions to develop cross-platform games, that will be discussed in the next sub-chapters.
2.1.1 OpenGL ES or Framework

Modern mobile OS use the OpenGL ES standard it consists of a cross-platform API for 2D and 3D graphics on embedded systems like consoles, phones, appliances and vehicles. It is a low-level interface between software and graphics acceleration. Its architecture is shown below:\(^3\):

The application sends textures and rendering commands to the OpenGL ES Client. This client communicates to the OpenGL ES Server that transforms this data into a format, that the GPU can compute.

The list of mobile OS that implement this standard is large and contains almost all systems, according to the OpenGL's official website[13], some of them are:

- Android
- Linux
- Windows Mobile
- Apple
- Symbian
- brew
- ...

On top of OpenGL ES API there are some frameworks created to abstract the user from low level details of the API. These help avoiding that the user rewrites boiler-plate code and “reinvents the wheel”, there are even frameworks that can easily deploy into multi-platforms, thanks to the widely implemented OpenGL standards.[14]

Opting to use a framework instead of working directly with OpenGL ES has pros and cons, for instance the type described by the following iron triangle:

---

When using directly OpenGL ES one has to control every fine detail of the API, this leads to large time consumption, but if done properly, it can bring high performance and, since one is using directly the API there's no limit in terms of functionality.

Using a framework (usually referred as a game engine) that has some features out of the box, as sprites, scenes, entities, business threads and other utilities, that avoid the direct contact with OpenGL ES, and rewrite widely implemented classes, as geometric forms for example, can save a lot of time. The usual drawback is the scope that tends to be more limited. The programmer will tend to fit in the framework following its lifecycle, although it is usually possible to write raw OpenGL ES code when using a game engine.

It is the programmer who chooses which approach will fit the project specifications, taking into account the drawing of Figure 2.3. This figure shows one of the trade-offs to take into account when planning an engineering project. One can only achieve two of the three corners efficiently.

In this particular case, using a framework was the reasonable choice. The number of scenes and animations could be perfectly executed without deep knowledge of OpenGL ES, achieving the desirable results.

---

4 http://agilebench.com/blog/tag/iron-triangle
2.1.2 2D Game Engines Approach

After choosing to use one framework for this project, other decisions had to be made. Since there are dozens of frameworks for different purposes, one must choose the correct tool to fulfill the specifications.

Typically, in the mobile game development world there are two main approaches: cross platform or native development. Since almost all the platforms support OpenGL ES, the development is done with this API in mind, and then compiled into each available platform. The major drawback is that, depending on the framework, the user sometimes needs to use a new programming language, which differs from the native one.

Unlike the conventional mobile application development where the cross-platform solutions tend to decrease the overall performance and UX, in the game development the hybrid approach is as good as a native approach, and makes it the best option in the majority of the use cases. In spite of those advantages, to prepare a framework which will deploy to multi-platform adds some initial overhead. This would not bring particular advantages to this project, but for a commercial product trying to reach the maximum number of devices, cross-platform is the way to go.

For the cross-platform approach there are many 2D game engines options:

- LibGDX – a free and open-source framework that uses Java and deploys to the major OSs including desktop[15];
- Marmalade – payable framework with the code written in Lua[16];
- Unity3D – has a powerful GUI, with C# as main language and is payable[17];
- Cocos2D-x – is open-source and uses Lua as coding language[18].
- ...

Aware of the previous point and knowing the majority of cross-platform engines on the market, the Android oriented game engine AndEngine[19] was chosen for the development of this game just to enforce the coupling with the Android native environment. In the following chapters, more will be explained.

2.2 Development Tools

This section describes and explains all the approaches taken and development tools used in the mobile game development.
2.2.1 SDK

Having chosen the mobile platform and framework, the SDK had to be logically the Android SDK[20]. An SDK is a software development kit, in other words, it is a set of tools aiming at the development of applications for a certain software package, in this case the Android OS. This SDK contains a debugger, multiple libraries, a device emulator, documentation, sample codes and various tools like the Android SDK Manager (Figure 2.4) that allows the programmer to update the API, download libs, sample codes and so on. The current API level is 19.

![The Android SDK Manager](http://developer.android.com/tools/help/sdk-manager.html)

Figure 2.4-The Android SDK Manager

2.2.2 Programming Language

The official language for Android is Java[21], created in 1995, currently in version 1.8 and it is a general-purpose language that strongly embraces the object oriented and imperative paradigm. Currently it is considered one of the strongest languages due to its "endless" libs and frameworks within one of the biggest active developer communities.

The Java compiled code, is called Java Byte Code. It runs on a Java Virtual Machine which executes the instructions. Android works with the same principle but in a slightly different way. In the compilation process, an extra compiler called DX compiler assembles the previously compiled files with the third party libraries in order to produce a dex file which will run on the Dalvik Virtual Machine. The differences can be seen on Figure 2.5.

2.2.3 Game Engine

As mentioned earlier, the game engine used is the AndEngine, an open-source project, created by Nicholas Gramlich.

This engine works directly with the OpenGL ES API by creating an abstract layer to render scenes, draw sprites, deal with sound and so on. This way the programmer puts more

effort into the game logic and less into the GPU details. The important concepts of this engine are the following:

- **Engine** – component responsible for scene and camera management, game loop and all calculations to draw entities on the screen.
- **Camera** – as the name suggests it represents the view, showing a part of the scene. It can zoom, pan, follow an entity and so on.
- **Scenes** – these are the backgrounds where all the action happens, all entities and texts are attached to her. A game can have multiple scenes.
- **Layers** – a scene can have multiple layers, for example, one for the background another for the game action and one for additional informations displayed in a strategic fashion. With layers the programmer can group entities or text in a practical way.
- **Sprites and Entities** – Entity is the basic class in AndEngine: everything that can be drawn to the screen is a Entity. Sprite is a sub-type of entity representing a simple object that can be drawn on the screen, like characters, NPCs, screen buttons and so on. It is probably the most frequently used sub-type of Entity.
- **Textures and Texture Regions** – from the OpenGL definition, a texture is a 2D graphic used to give shape to an object. A sprite is just a box, it needs a texture to give him a pleasant appearance. A texture region is an area of the memory reserved to hold a texture.

In order to use AndEngine one should sub-class the “BaseGameActivity” and override the following methods after the launch of the Activity to create the foundations of the game, according with its lifecycle that is shown on Figure 2.6.

- **onCreateEngineOptions** – instantiates the camera and the EngineOptions object in order to define the display’s properties of the game (called in the onCreate of the activity).
- **OnCreateResources** – method to create and set up any resource for the game, as sounds, textures and fonts.
- **OnCreateScene** – instantiates the scene object.
- **OnPopulateScene** – used to attach children entities to the scene.

Alongside with the creation of the game, the lifecycle shows what happens when the game is paused and what goes on if the user choose to resume or terminate it.

### 2.2.4 IDE

The Integrated Development Environment used to create the game application was the Android-Studio\(^8\)[23]. Based on the IntelliJ IDEA IDE, it provides newer features and improvements over the Eclipse ADT[24] which is an largely used open source IDE equiped with the plugin to develop Android applications. The main features are: its integration with Gradle, a build system with repository dependency; an overall better Android oriented code, refactoring and completation, and so on... Although it is in a Beta version, it is a powerfull IDE with full support from Google's Android Team.

---

2.2.5 DRCS

Source code is everything in a software project, controlling the current version is fulcral to roleback critical errors, it saves the code within a remote server and keeps tracks of the development progress. A DRCS normally implements the client-server architecture, the remote server saves the whole code developed allowing multiple developers to work on the same project at the same time. DRCS stands for Distribution Revision Control System and in this project, Git[25] was the DRCS used. In this case, instead of implementing a dedicated server, a third party service was used in order to save the code on a remote server. This third party server was supplied by the web-based hosting service BitBucket[26], including free repositories for small developer teams.

The reason to use a DRCS on a single developer project is mainly to ensure version control and to easily deploy the code to a remote location to reduce the risks of losing the source code.

The git flow with the basic commands are as follows⁹:

⁹ http://blog.osteele.com/posts/2008/05/my-git-workflow
The workspace is where the ongoing work is. When a developer wants to send the current version to the server first he must add the files to the index to be committed. When the commit with the modified files are done, they can be pushed to the remote repository.

The other commands are used in a multi-developer environment where one must pull the others work to have the last version of the project.

### 2.2.6 Testing Devices

In order to test the progress of the development, a high and a mid-end devices were used. One of the biggest challenges of developing for the Android platform is its fragmentation, making it difficult for the application to run in the same way on multiple devices.

The emulator is a good tool to help give access to multiple devices, but due to the fact that it is an emulator, it never gives the same experience as the one of deploying to a real device. The devices used in the development process were the Oppo Find 5 and the TCT Vodafone 875, running Android 4.4.4 and 4.1.2 respectively.
The next table shows the technical specifications of each device:

<table>
<thead>
<tr>
<th>Specs\Devices</th>
<th>TCT Vodafone 875</th>
<th>Oppo Find 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Android 4.1.2</td>
<td>Android 4.4.4</td>
</tr>
<tr>
<td>CPU</td>
<td>1GHz Cortex-A9</td>
<td>Quad-core 1.5GHz Krait</td>
</tr>
<tr>
<td>GPU</td>
<td>PowerVR SGX531</td>
<td>Adreno 320</td>
</tr>
<tr>
<td>Display</td>
<td>320x480, 3.5&quot;</td>
<td>1080x1920, 5&quot;</td>
</tr>
<tr>
<td>RAM</td>
<td>512Mb</td>
<td>2Gb</td>
</tr>
</tbody>
</table>

Table 1-Smartphones specifications
Chapter 3

3 State of the art

There are numerous memory games available on the mobile apps market. However the majority of these games has none or poor auditory elements. For instance the top downloaded apps on the market covering memory training are the ones presented previously in chapter 1.4, these games are based on image recalling to test the user's memory and attention spawn.

A deeper research in auditory games for this particular platform, introduces some interesting ideas. The app, “HearBuilder Auditory Memory” developed by “Super Duper Publications”[27] was the most pertinent one because it interacts with the player through TTS. The player must then associate what was said with the corresponding objective, usually selecting some item or ordering some sequence according to the voice's command. There are other games with a similar model, for example:

- “Auditory Sequencing LS!” by Lickity-Split! Learning[28]
- “Memory” by Alma[29]
- “Mimemo” by binariq[30]
The Figure 3.1 is a game scene showing the type of challenges purposed by the “HearBuilder Auditory Memory”. One must press the “Play code” button then click on the item corresponding to the one mentioned in the speech. The scene on Figure 3.2 shows another type of auditory game. In this one the player must select two tiles where the image and the sound match.

In spite of the auditory interaction, none of these games uses the voice in order to achieve the recalling process.

After some research in other computing platforms, the result was similar. The voice recognition applications were used on other contexts, such as in the English learning process or voice commands. For the desktop/browser platform there were some applications with auditory interaction such as:

• “Voice Memory Game” by NextSpeak[31]. A desktop application to learn Mandarin by listening and memorizing vocabulary.
• “Audio Concentration Games” by Many Things[32]. Browser game based on flipped cards. One must recall the sound made by each card to perform the match.
• “Audio Word Match” by Spelling City[33]. Same game model as “Audio Concentration Games”.

Other enterprises have developed platforms with a subscription service that offers different types of brain games. Examples are BrainHQ[34] and Lumosity[35]. These two offer game exercises divided in:

• Memory – Such as remembering patterns and locations; associating names with faces; recalling sequences of objects and movements.
• Attention – Like ignoring distractions; quickly picking out patterns; responding to key information within a large area.
• Brain Speed – Decision-making in time sensitive situation; quickly recalling recent information.
• Intelligence – Using logical reasoning or planning efficient routes.
• Navigation – Reorienting yourself as perspective changes.

Each type of exercise has a list of games.

The majority of these games has only visual stimuli. Only a few games explore auditory features, and none has voice interaction with speech recognition.

In the development of this document no memory or brain game with voice recognition for the recalling process was found.
Chapter 4

4 Web-Service development

Nowadays smartphones have great specifications allowing the user to accomplish a great number of tasks without third-party resources. But sometimes it is not enough to process heavy computations or such resources are not available for smartphones such as ASR or some types of TTS. For this a web service was necessary in order to offer these features. The price to pay is in the networking usage.

This chapter will introduce the main technologies and tools used to develop the server side. The next chapter will focus on the concrete implementation and the API developed used to offer ASR and TTS capabilities to the game.

4.1 Development Tools

4.1.1 Web application Framework

By definition, a web application framework is a software framework designed to support the development of web applications. These frameworks reduce the development overhead providing, for example, libraries for database access and session management.

For this application the choice was Play! Framework[36]. What motivated this choice was the modernity of it, in its terms of technologies and practices implemented but also because the code can be written in a functional language, the Scala language[37] which offer greate scalability by enforcing proper code rules. Play is a framework written in Scala and Java. It follows the Model-View-Controller pattern[38], that dives each layer by its function: database manipulation, presentation layer and request handling respectively. This to ensure the scalability of the development and a RESTful architecture[39], which are a set of constraints applied to the API once again to have a structured application. It has a build system called SBT, is stateless too which means it did not save the state of the client, making it a robust solution. It is also non-blocking thanks to its foundation: the Http web server Jboss Netty. Since this is a framework based in a compiled language it has an overall better performance compared to web frameworks written in interpreted languages. The framework has also a plug-in to manage JDBC connections and a variety of modules to avoid having to re-invent the wheel. The next figure shows the Play Framework's architecture\(^\text{10}\) with the visual representation of some terms described previously. The router receives an Http Request, it envoques the respectively controller which manipulates data through the model layer. After this the controller render a tamplate (View) which is send to the client.

\(^{10}\) http://www.playframework.com/documentation/1.0/main
4.1.2 Programming Language

As already mentioned, development in Play can be done with Java or Scala, both JVM languages, in other words, a programming language that is compiled to run in the JVM.

Scala was the used language, especially because of its functional paradigm. This paradigm enforces the development of immutable code and focus on function evaluation instead of state and mutable data. Thus it turns it into a thread safe and easily parallelizable. On top of this, the desire to learn more about the functional paradigm was the major reason to choose Scala. There is more about Scala and the functional paradigm, but this is out of this document's scope.
4.1.3 Database

In order to keep data persistent, such as the user’s data storage for uploaded audio files and recognized strings from the ASR, the use of a database is the most robust way to do it. The database used was PostgreSQL[40]. It is an Object-Relational Database Management System (ORDBMS). This manager implements the majority of the SQL:2011 standards and is ACID-compliant and transactional in order to maintain data consistency. The concurrency is achieved through a system known as multiversion concurrency control which gives each transaction a “snapshot” of the database.

The management of the DB was made in the IDE itself, which has powerful database tools, this will be explained next.

4.1.4 IDE

![IntelliJ IDEA with Play 2 plug-in](http://plugins.jetbrains.com/plugin/7080)

Figure 4.2 - IntelliJ IDEA, currently in version 13, with Play 2 plug-in

Similarly to the mobile side development, to increase productivity, an IDE that offers a set of tools was used, in order to help in the development process. This IDE is the IntelliJ IDEA\(^\text{11}\), this IDE has a plug-in for Play Framework, making it an oriented IDE to develop Play applications, recognizing the project structure and dependency modules.

This IDE also offers powerful tools to DB management, providing an SQL console and table visualization making it simpler to visualize relations and data. Figure 4.3\(^\text{12}\)

\(^{11}\) [http://plugins.jetbrains.com/plugin/7080](http://plugins.jetbrains.com/plugin/7080)

\(^{12}\) [http://www.jetbrains.com/idea/features/sql_java.html](http://www.jetbrains.com/idea/features/sql_java.html)
In the figure above there is an example of the table visualization tool. It is more intuitive to figure out the model and its relations.

4.1.5 DRCS

The DRCS used in the server side was the same method and protocol used in the client development, to see more go to page 16, 2.2.5.

4.2 Third party servers

In the development of the server, third party servers offer some features. The main features obtained through third party services are presented here.

4.2.1 Authorization provider

In order to increase server security, requests should be secured. Only registered users should be allowed to make successful requests. The authentication is made through a provider that implements the Oauth standards[41]. Oauth specifies a way for resource owners
to provide access to third-party of their server resources without sharing credentials. It is a commonly used to log into third party servers using one of the providers without worrying about their credentials. In this case the chosen provider was Google.

To use Google as the provider for authentication, the user has to request a valid token to consume. This request is made by the client, who will send it to the server. In its turn it will send the token to Google's servers to check its validity. In case of success Google sends back to the server the user's information stored in Google's databases. This authorization is then used by the web-service to create an account and start a session for the user.

4.2.2 ASR and TTS

The other third party server is located at Inesc-ID and those are responsible for the use of TTS\cite{42} and ASR\cite{43} services. This server has a proper API explained later in the concrete implementation chapter. This server was used in other projects with similar features but it is used in a controlled environment. In this case when used in a mobile application extra-care must be taken. That is why the need to control who accesses this service. Which is the main reason for implementing authentication.
Chapter 5

5 Design solution

In this chapter all the implemented methodologies for each part of the application, such as server, mobile and architectures used are explained. In addition, all the database designs, authentication flow, the use of TTS and ASR and also the game architecture are described in detail.

5.1 Application architecture

Here, the main architecture of the application is explained and an overview of the flow of communication between device, server, database and third party servers is presented. The following figure shows how the different components are connected.

![Main architecture diagram](image)

Figure 5.1-Main architecture and principal components

The figure above shows three distinct groups: the client, called “Player” where the game logic happens, the “Oauth Provider”, responsible for the authentication process between client and provider and finally the server side, where all components are hosted in Inesc-ID infrastructures. This part is divided between the main server, the “Talking Memory Server” who controls the sessions and the data flow; the “ASR / TTS” service and finally the “Talking Memory Database” where the data are stored.

Starting with the client, an instance of the Talking Memory game is running on the Android device. In order to play the game one must login first. The login flow implies that the client
asks the OAuth provider for a token. After the client has received this token he sends it to the “Talking Memory Server” who itself will retrieve the client data from the Oauth Provider through an Http request. If he successfully retrieves the user information, then a session is created and a new entry for the user is inserted into the database, if none exists already, or an existing one is refreshed. This session will allow the client game instance to remotely access the ASR service uploading audio data and use the TTS service sending a Http request with a JSON[44] containing the string to be synthesized; the server then sends the audio file into a stream to be reproduced on the player's device.

The database will be used to save the users information and all the uploaded sounds and respectively recognized strings that come from the ASR. Since the TTS will only be used for pre-constructed strings randomly generated by the game instance, no data concerning the TTS will be saved in the database because this is of no particular interest for future analysis.

5.2 Web-server

The web-server supplies the core functionality of the game, with TTS and ASR features. As already mentioned these features are only available to registered users with valid sessions. This section explains in depth how the server behaves and the important steps it takes to process different requests.

5.2.1 Web-Server Architecture

The Figure 5.2 illustrates the entire architecture process and flow of logic and data, the layers are separated to give visibility to the MVC architecture. Since it is a web-server that offers a service, there is no need for the View layer. The other slight modification of the pattern talked about in the last chapter, is the introduction of the Service layer, also called Business layer which is the developer's choice to write a cleaner code. This layer handles the business logic of the web-server and communicates with the models for CRUD operations.

Deconstructing the implementation, follow the previous figure to understand the whole process.

The client makes an Http request invoking an URL that matches the server’s hostname plus an application known route, for example: http://talkingmemory.l2f.inesc-id.pt/audio/text in which:

- Hostname stands for: http://talkingmemory.l2f.inesc-id.pt
- Route stands for: /audio/text
The embedded Http server receives the request, then the routing system tries to match the URL to a controller action. In the figure the arrow “Controllers” shows the controllers layers that deal with valid requests. Typically one controller can have multiple actions and one action corresponds to a route (and a request). On the other hand, when evoking an URL with an undefined route, since the Http server can match any route, it returns a Http response with the “404” status code meaning “Not found” (all Http status code can be found here[45]). Reviewing the Http, the request methods[46] define the desired operation to be accomplished when executed. All these details along with the routes will be exposed when the complete server API will be explained.

Meanwhile the previously URL being a POST request, means that the request has a body message containing data to be used on the server side. This message can be of different types, and is defined by the content-type, the header field used to define the type of the body's message. A complete list of content-types can be found here[47]. In this case the content-type is the application/json, which means that the request body contains a JSON

Figure 5.2-Data flow in server architecture
data, a common data key-value pair exchange type due to its human readability. Once again, this will be explained in the server’s API.

After the action has been called, an important module called “Secure Social” intervenes. This module is part of the authentication flow and it intercepts all valid requests. This third party module is used to control the access to the controller's action verifying if the “person” making the request is a registered user with a valid session stored in Play's cache. If user is valid, then this module allows the execution of the code and delegates to the “Persistent User” service who actually communicates with the “User Identity” model that will fetch the corresponding “User Identity” from the database. If a unknown user tries to make a request, the server returns a 401 status code (Unauthorized).

Logically the only action that accepts unauthorized users is the login route. This one will receive in a POST, a JSON containing an access token. This token is exchanged in the “OAuth provider” by the corresponding profile information. In case of an invalid token, the provider returns an error and so the Talking Memory server reports to the user as an 401. Usually “invalid tokens” are related to timeout problems forcing the user to ask for another one.

Valid tokens will successfully return the profile of the user stored in the provider's datacenter. This profile is identified in “Secure Social” by “IdentityId” which is a class that contains two attributes: an id and the OAuth provider (ex.: Google, Facebook, Twitter, etc). The Auth controller will try to find an existing “User Identity” in the database, through the “Persistent User” service. In case of an empty result, a new one must be created. A “Game User” and a “Persistent OAuth2 Info” are also created and saved in the database. After this, the final step for authentication is to generate a session id for this user. This step is performed in the “Auth” service layer. The session is created by the “Authenticator” object that will be saved in Play's cache with a pre-defined time validity set in the secure social config file. After this time limit has passed, a new session is required in order to access the server's services. This session id is then sent through a cookie in the Http response header to the user's request. This cookie is what ensures the session of a user, when deleted the server can't back track the users identity and a new login is required.

The login done and a valid session created, the actions in “Audio” controller can be acceded. Apart from increasing server side security, making the player login is useful to access profile information to make it easy to create leaderboard and to mine data from the database. Continuing with the server's flow once an action has been invoked and the request parameters are valid, the “Audio” service will be responsible for sending and retrieving data both from TTS and ASR. The way these requests are done is explained in TTS and ASR sub-chapters respectively.
The last two remaining blocks from the diagram are the “Audio info” and the “Words” both in the model's layer. “Audio Info” stores in the database, metadata of the uploaded audio file from the user after being recognized by the ASR. This metadata is the path leading to the audio file and the user id. The “Words” model has a relation with the “Audio Info", and it stores the string recognized by the ASR for the uploaded audio file.

5.2.2 Database design

In order to persist data in an application of this type, a database is the right choice for its scalability and maintenance. Plus, Play offers a plug-in for Java Database Connectivity (JDBC) and a middle layer for CRUD operations making it simpler and faster to integrate a database in the application.

To connect the application with a database one needs to configure a connection pool in the “conf/application.conf” file of the project. In this case the connection pool has the following aspect:

```
db.default.driver=org.postgresql.Driver
db.default.url="jdbc:postgresql://localhost/talkingMemoryDB"
db.default.user=tmem
db.default.password=""
```

It is also possible to connect as many database as necessary. The middle layer is called ANORM (Anorm is not an Object Relational Mapper) and comes in the Play Framework. In summa, as the name claims, it does not create a DSL to deal with the tables, it is just an API to simplify the JDBC API itself. One writes SQL with the ANORM methods to query the database.

Starting with the tables involved in the login process: the “userIdentity” contains the information retrieved from the Oauth provider like the columns “firstName”, “lastName”, “fullName”, “email” and “avatarUrl”. This has a compound primary key, formed by the “userId” and the “providerId”. Those two attributes were already mentioned as the “IdentityId”, they are the id of the user in the provider. This table has two foreign keys, the “oAuth2InfoId” and the “gameUserId” as one can observe on Figure 5.3.

The “authenticationMethod” is just metadata of the protocol used to authenticate: “Oauth1", “Oauth2", “OpenId” or “UserPassword”. This column could have been deleted, because the only protocol used was the “Oauth2”. Continuing on the subject of the authentication protocol, the table “Oauth2Info” is prepared for different types of tokens. In this particular case the validity of the token is only necessary for the sign in. After that it is the session timer that counts. In this table the only column of interest is the “accessToken” which stores the received token and is updated every time the existed user logins again. This table
has a one-to-one relation with the “userIdentity” table ensured by the “unique” method defined in the schema creation that follows the SQL language.

The “gameUser” table represents a player. Each “gameUser” has a “userIdentity” (the foreign key and also a unique identifier ensuring the one-to-one relation similar to the “oauth2Info” relation). The only column besides the primary key is the “fullName”, only used to send the client the player’s name to be used in the “welcome” and on the “leaderboards”.

Each “gameUser” can upload multi audio files, for this a one-to-many relation is needed. The “many” in this case is the table “audioInfo” as already mentioned. It stores metadata of about the audio file like the name. One audio file (one row in the “audioInfo”) has a recognized string stored in the “words” table. Once again a one-to-one relation.
Sometimes the ASR can't recognize the speech from the audio file, so it returns a keyword informing of this fact. In any case the audio file is stored in the database as a way to know which audio files can't be correctly recognized.

This explains the main tables and columns. They were created with a .sql file. This file is a script that tracks the database evolutions. The framework interprets this script and applies it to create or update the database. For example the part of the script used to create the “audioInfo” table was (SQL code):

```
CREATE TABLE audioInfo (  
audioInfoId SERIAL Not null,  
fileName       varchar(255) not null,  
userId         INT not null,  
PRIMARY KEY(audioInfoId),  
FOREIGN KEY(userId) REFERENCES gameUser(gameUserId)  
);
```

This script is parsed following these keywords: (more details are given in the documentation of the framework[49]).

- “Ups” - part that describes the required transformations
- “Downs” - part that describes how to revert them.

### 5.2.3 Authentication

The authentication process was already introduced in previous chapters, and also the database that supports it. Here will be explained in detail the steps needed to authenticate a user.

Before requesting a token, the provider must know about the identity of the application requesting the user's data. In this particular case, the provider is Google, and the application is developed for the Android platform. As well known, and although Android is Open Source, it is highly supported by Google and normally an Android smartphone has a Google account associated to it. Existing dozens of services available for this platform, it makes it simple, and is the reason why Google was the chosen Oauth provider.

The authentication flow for the first time user is the following:
Authentication flow

Figure 5.4: Authentication flow with all components
In the development process, in order to use Google as the provider one must set up the Google Play Service SDK in the Android project adding this library as a dependency of the project and defining its version in the manifest file. After this the library is available and the App can now use the “getToken()” method from the GoogleAuthUtil class delivered by the recently installed SDK to make requests to the Google Play Services API.

Following the user perspective and understanding the code under the hood at the same time.

1. After the user clicks on the “Login” button, the “newChooseAccountIntent()” method from the “AccountPicker” class displays all the Google accounts on the smartphone (also allowing to add an account if necessary). When an account is chosen, the “getToken()” method will receive as a parameter the Google account and the scope to request a token for it. The scope is the type of information required of the application.

2. In the first attempt to log in, the “getToken()” will throw a “UserRecoverableAuthException” because the provider needs the user's consent to deliver his profile information to the App. The consent screen tells the user what type of information the App is trying to access, as shown on Figure 5.5.

![Figure 5.5](image)

3. When clicking on the “OK” button of the consent screen, the Google Play Service API saves this choice in its servers (to revert this, one must revoke the access from the security definitions of one's account).

4. With the user's consent, the Oauth provider sends a token to the client.
5. The client makes an HTTP request to the Talking Memory server sending the token in its body as a JSON.

6. The Talking Memory server will send a request to the Google Play Services API together with the token in order to retrieve the profile information related to the same.

7. If the token is valid, a JSON with the information defined in the scope is returned to the server.

8. With the retrieved profile information a “userIdentity” and a “gameUser” are saved in the database.

9. A session is created and returned to the client’s request through a cookie. The next requests are now valid thanks to the session id in the cookie.

### 5.2.4 API Developed

The API is the interface developed to communicate with the server. It was projected to match the RESTful API specifications. Here is explained the routes and parameters needed to correctly operate with the server and also the HTTP responses for all the cases.

Basically there are three routes, one for the authentication process, one to send texts to be synthesized and one to send audio to be recognized by the ASR.

- **Authentication**
  - Authenticate a user in the server

- **Parameters**
  - The name of the OAuth provider (ex.: facebook, google, etc)

- **Request**
  - Method: POST
  - Content-Type: application/json
  - URL: [http://talkingmemory.l2f.inesc-id.pt/authenticateMobile/google](http://talkingmemory.l2f.inesc-id.pt/authenticateMobile/google) (if “Google” was the chosen provider)
  - Body:
    
    ```
    {
        "accessToken": "ya29.MADfe1J2m82rEwAAAB1J14iQxRJfuFUCJSe2"
    }
    ```

- **Response**
○ Status code: 200 (OK)
  ▪ Description: Everything works as pretended
  ▪ Content-Type: application/json
  ▪ Body:
    
    ```json
    {
      "sessionId" : "234sdfsd238923jdojiwejr9834392pidomi8r32d2ewcf",
      "fullName" : "Paul Simons"
    }
    ```

○ Status code: 202 (Accepted)
  ▪ Description: User has a valid session and tries to login again
  ▪ Content-Type: application/json
  ▪ Body:
    
    ```json
    {
      "sessionId" : "234sdfsd238923jdojiwejr9834392pidomi8r32d2ewcf",
      "fullName" : "Paul Simons"
    }
    ```

○ Status code: 400 (Bad Request)
  ▪ Description: Invalid request due to syntax error in the body
  ▪ Content-Type: text/html

○ Status code: 404 (Not Found)
  ▪ Description: Didn't find any user for the token
  ▪ Content-Type: text/plain
  ▪ Body:
    
    ```text
    { 
      Authentication error
    }
    ```

○ Status code: 500 (Internal Server Error)
  ▪ Description: Some unexpected error
• Content-Type: text/html

• **Text-to-speech**
  ◦ Synthesize a string

• **Request**
  ◦ Method: POST
  ◦ Content-Type: application/json
  ◦ Body
    
    ```json
    {
        "text" : "Example of text to synthesize",
        "lang" : "en",
        "gender" : "female"
    }
    ```

    - The above example is for the English language. To use the Portuguese language set “lang” to “pt”. As for gender, for male set “gender” to “male”, for female set it to “female”.

• **Responses**
  ◦ Status code: 200 (OK)
    - Description: The bytes of the audio synthesized in the mp3 format.
    - Content-Type: application/octet-stream
    - Transfer-Encoding: chunked
  ◦ Status code: 400 (Bad Request)
    - Description: Invalid request due to syntax error in the request body.
    - Content-Type: text/html
  ◦ Status code: 401 (Unauthorized)
    - Description: If an unauthenticated user is trying to make a request.
    - Content-Type: text/plain
    - Body:
{  
  Credentials required  
}

- Status code: 500 (Internal Server Error)
  - Description: Some unexpected error or an unexpected parameter
  - Content-Type: text/html

- **Automatic speech recognition**
  - Recognize a file containing audio in raw format.

- **Request**
  - Method: POST
  - Content-Type: multipart/form-data
  - Form-data
    - File → key: “audioFile” | value: (audio file in raw format)
    - Params → {“lang”: “en”, “num_items”: “2”}
      - Other option for key value “lang” is “pt” if the audio file has portuguese spoken words, the example above is for english spoken words. The key “num_items” correspond with the number of elements to be recognized by the ASR.

- **Responses**
  - Status code: 200 (OK)
    - Description:
    - Content-Type: application/json
    - Body:
      
      ```
      {
      “audioWords” : “Example of recognized speech”
      }
      ```
  - Status code: 400 (Bad Request)
    - Description: Invalid request due to missing key
• Content-Type: text/plain
• Body
{
    Missing data 'Lang'
}
• If the key for the audio file is missing then the message is “Missing 'Audio File'” is presented.
  ◦ Status code: 401 (Unauthorized)
    • Description: If an unauthenticated user is trying to make a request.
    • Content-Type: text/plain
    • Body:
      {
        Credentials required
      }
  ◦ Status code: 500 (Internal Server Error)
    • Description: Some unexpected error or an unexpected value parameter
    • Content-Type: text/html

5.2.5 TTS

Once the text is successfully sent and received by the Talking Memory Server, this must be sent to the module responsible to perform this task. For each request, the server will communicate with the TTS.

Before going on in more details, notice that the previous figure represents a shortcut for how the data flows in the server. The main focus is on the service layer and how it deals with the TTS API communication.

Point number 1 is described on the server's API. With the received parameters(1) some additional configurations are needed to synthesize the text. This configurations are asked by the TTS API. Namely: the type of task, the encoding of the audio file, and the voice type desired. The task is “echo”, the encoding used was mp3, a compressed lossy data audio coding format to reduce the networking and finally the desired voice.
The TTS offers different voices of the two genders, in this case the choice for Portuguese and male was: “Viriato”, for female: “Violeta”, for English and male: “bdl_hts2010” and female: “slt_hts2010”. These parameters are then sent(2) to the TTS. When finished the module sends the URL to download the synthesized file in the mp3 format(3). The server downloads the audio file thanks to the previously received URL(4). This download(5) is then received as a stream of bytes that are sent to the client as soon as it is received by the server(6).

From a back-end development point of view, the Talking Memory Server plays the role of a client and this will bring some complications if not done in an asynchronous way. This because the server can be blocked waiting for an undertermined response. In the section 5.2.7 this will be revisited.

5.2.6 ASR

The ASR has a similar architecture to the TTS. After a valid raw sound is uploaded, it is sent to the ASR according to the recognizer's specifications together with a grammar containing possible words to be recognized.

Notice that the following schema doesn't represent the complete server's architecture. It only focuses on the ASR service and the communication with the server, similarly to how the TTS schema is presented. For the total architecture return to the beginning of this chapter where all layers and data flows are carefully shown.
The above figure shows the architecture for the ASR, after the upload of the audio following the specifications of the API (1), the file is sent to the speech recognizer. The sound is sent in a string composed by the bytes in a decimal form, such as `[20, -12, 2, -24, 0, 1, 34, -5]` alongside with a string containing the grammar. This grammar is stored in a XML on the server, containing all relevant strings for the game. There are two grammars, one for the English words and other for the Portuguese ones. The grammar will increase the recognizer's accuracy through keyword spotting, this technic looks for words on the speech that are contained on the grammar dramatically reducing the mismatched words.

The sound, as already mentioned, is in raw format meaning there are no header containing metadata about the file and it is part of the ASR API. In the mobile development chapter the details of how this audio file is recorded will be revealed. When the recognizer finishes its work a string with the recognized words is returned. The “Audio” service sends the data to the model to be stored in the database and finally the string is sent to the client.

5.2.7 Performance specifications

So far this chapter has only approached the server side in a functional way, without stating problems such as concurrency or scalability. There are potentially blocking operations such as I/O and remote services requests. A server must be tuned in order to handle a maximum of requests without trouble.
Under the hood the Http server used is non-blocking, which means it will never block while waiting for some computation, service or I/O operations unfinished, although it is possible to block the code. To deal with those situations the server was written using primitives to build non-blocking code, the only exception were CRUD operations. Apart from the database connections, all requests to the ASR and TTS are made in a asynchronous way. When the result is available the portion of the code responsible for dealing with the result is invoked at this particular time. Keeping this in mind, the size of the pool-thread was one thread per core.

5.3 Mobile Application

In this section the developed game is described. Starting by explaining the configuration of the modules used, then how the managers bring the scenes and the game logic to life.

In the Android context, when launching an application the system creates a new Linux process with a single thread, the main thread. In this case the main thread is created but when the game engine is configured, in run time, it creates its own thread, the update thread, responsible for processing each frame. In the next subchapter the main modules are presented.

5.3.1 Modules configuration

5.3.1.1 Game Engine

In the game development there is an important concept which is the notion of camera. The camera represents the part of the scene that is displayed to the user. In this case the game world being static, the camera has to be static too, and configured to match the background of the scenes. These were set to have a ratio of 16:9 and a ratio resolution policy, to fit in multiple screen sizes without distorting the graphics. The solution for screen with different ratios is to fill the gaps with black stripes.

The model followed to design the application was the “single-activity multi-scene” model to reduce the overhead of the activity lifecycle. At a later stage the scene manager will be introduced to explain how the diverse scenes were presented and managed without the use of activities.

When the game starts the system launches the main activity, where the game engine was created. The setup of the game engine goes through the following steps which are its foundation:

1. Create game engine options. In this step the camera and some options of the display properties are created, such as locking in portrait mode or never turning off the screen.
2. Create resources. The game needs resources to have something to show. Where usually the resources are created, in this particular case the resource manager was created. Later this manager will be explained.

3. Create scene. The game engine has the notion of scene where the game action happens. The resources need to live inside the scene. When initializing the engine the first scene must be set. In this case the splash screen.

4. Populate scene. Usually this is used to attach entities to the scene. In this case, it was used to fire up the first scene that needed player interaction.

5.3.1.2 Networking

This game has a high network activity. In order to reduce the development time on such critical module, a third party library (ION[51]) was used to help the API networking development.

The focus was on having an internal storage to save the cookie that contains the session. Every time the application with a logged user makes a request the cookie is set. The other critical point is the non-blocking. Once again the application can't be stuck in a pending request, so all the networking is non-blocking, using callbacks to handle the responses at the proper time.

5.3.2 Managers

The managers play the fundamental role in terms of how the application works. Following the singleton pattern, only one instance of the object is created (except for the session and the score manager, which are stateless). The way these managers interact between themselves is as follows. The figure shows how the scene and resource manager work in order to swap the current scene.

The “Game Manager” is only used in the “Game Scene” because this scene is responsible for showing the elements in the game action. In other words, the game scene deals with the visual part, showing the NPC, the buttons, the strings and all the visual content. Whereas the “Game Manager” deals with the current level, the score, whose NPC must be rendered and so on. These two components are tightly related.
5.3.2.1 Scene

This manager, as the name suggests, manages the scene transaction, and controls the resource manager according to which scene will be presented.

When a scene ends its job, it notifies the “Scene Manager” with the next scene that has to be rendered. This manager has full knowledge of which scene precede which. So when this manager receives the task to switch to, for example “Game Scene”, first it asks the “Resource Manager” to load the assets of the “Game Scene” into the memory, and to unload the resources used in the “Menu Scene” (the scene that preceded the “Game Scene”).

With the resources ready the manager can create a new instance of the scene. In the creation of the scene, it accesses the “Resource Manager” to get the graphics loaded to use them to create the visual content. When an instance of the scene is created, it is passed to the engine as the current one to be rendered in the next frame. Meanwhile the older scene can be disposed of, liberating unused memory.

5.3.2.2 Resource

When this manager needs to load the resources of a given scene, it does create a texture atlas big enough to contain all texture regions for this scene(Figure 5.9). Then for each of the graphics that needs to be represented in the scene a texture region has to be created. After
this each texture region is attached to the texture atlas then finally loaded into the GPU. Now the texture can be used at any time during the game.

![Texture Atlas with three texture regions](image)

**Figure 5.9-Texture Atlas with three texture regions**

**5.3.2.3 Game**

The game manager follows the singleton pattern, having only one instance in the whole application's life span.

It was built to provide an abstraction layer for the scene regarding the level construction in terms of difficulty and game progression. It offers an API to the “Game Scene”. This way the “Game Scene” only focuses on rendering what the “Game Manager” asks, so these two elements are related and will be explained in the gamification process in which the game logic is also explained.

**5.3.2.4 Session Manager**

The session manager keeps track of the validity of the session. The session manager can be in one of three states: “Logged in”, “Expired” or “Logged out”. When the user has a valid session he is in the “Logged in” state. When the user sends a request to the server during game play and the session has already expired, it will be set as “Expired”. When the user is in the main menu and the session has “Expired”, it will automatically be logged out and all data concerning this session will be erased.
The persistent storage system used was the “Shared Preferences” of the Android SDK. The data was stored in the internal storage of the application so as to become inaccessible to the other elements in the system.

5.3.2.5 Score Manager

The score manager works the same way as the “Session Manager”, using the “Shared Preferences” of the system. It saves the five best scores.

The following figure shows the relation between these managers with the two scenes that use them and the logic that flows between them.

![Figure 5.10-Relation between score, session and game manager with the game and menu scenes](image)

5.3.3 Audio

The audio plays an important role in the game play. Using the microphone as source of input and the speakers as the way to reproduce sound. The way both of this features were used are explained here.

5.3.3.1 Recorder

The recording process required fine tune of the recorder because it had to be sent to the ASR. Thus, a set of requirements had to be fulfilled to maximize the recognition process. So the audio was recorded with the following parameters:

- Sampling Rate: 16000 Hz
- Channel: Mono
5.3.3.2 Player

The player used was a media player offered by the SDK. The file was received through the network then saved in a temporary file. This audio file was in .mp3 format.

5.3.4 Factory

Each level requires a variable number of NPCs and each NPC needs a set number of orders according to the level of difficulty design, which will be explained in the next chapter in the "Level" sub-chapter (6.2). This level design only depends on the current level number to retrieve any desired level, since it's generated according to the designed equations. These equations return the number of NPCs and orders.

The main factory is composed of multiple factories for the following components: Level, NPC, Order and speech. Each one will be explained next:

• Level factory – the level factory is called everytime a new level must be created. This factory interacts with all the other factories, assembling all the pieces to create the level and deliver it to the “Game Manager”. The only information it needs to obtain is the level number. With the level number the level factory retrieves the number of NPCs and orders according to the level difficulty equations. The ones that determine how the game difficulty progresses. The game type is first chosen randomly, after that it will request each NPC from the NPC factory and ensure that all NPCs are different. It requests from the order factory all the orders for each NPC and finally from the speech factory to create the strings that will be used by the TTS to interact with the user. The architecture of this factory is basically composed of a for loop that will iterate until all numbers of NPCs are consumed.

• NPC factory – to create an NPC, the gender is at first randomly chosen. After that and according to the gender a set of faces and names is picked. This to ensure that a female face won't appear with a male name. Who deals with the non duplication of NPCs is the proper class NPC, implementing methods to compare two elements and to verify whether they are equals.

• Order factory – the order factory is straightforward, picking up randomly a number of items equal to the number generated by the equation.

• Speech factory – this factory receives the NPC’s name and orders and build random strings according to the type level. The phrases are divided in: a salutation like “Hello”, a presentation like “my name is Peter”, the order for instance “I'd like to order a pizza and a beer” and a farewell like “see you soon”. There are several phrases for each part. They were randomly chosen to create different strings and thus make the
interaction more dynamic. From a technical point-of-view, the each divided text is implemented using a enumerated type. Where each type is the divided phrase: salutation, presentation, order and a farewell.
Chapter 6

6 Gamification

This chapter introduces the game design. A game description, the level difficulty metrics, the screens created and the way they interact with each other. First the game logic is explained for the three game types. After the plots of the level's difficulty are presented, the scoring model and finally the scenes created are also presented.

6.1 Game Logic and Rules

In the Introductory chapter the game rules were explained, in this sub-chapter more details about it are going to be explained.

The game logic is divided in two different levels created to dinamize the game experience. They are “Match item” and “Call NPC”.

The game's flow is divided in two distinct parts. In each level, a batch of NPCs comes first to present themselves and to ask a set of orders.

The number of NPC as well as their characteristics like name, face and the number of orders are randomly generated according to the level's difficulty. The second part consists in remembering information depending on the level's type. The second part is when the player must interact.

The differences between each level type is presented next.

- **Match item**: in this level the player must remember the name and the orders of the NPC. When the NPC returns, it will present himself, then the player must repeat correctly and in a non particular way the correct items.

- **Call Npc**: in this level type, the NPC comes to present himself and to ask for some products like before, but now when the order is ready it appears on the balcony and the player must call the NPC that gave the order.

Figure 6.1 shows the logic used in the game scene. All the flags used in the decisions are controlled by the “Game Manager” as well as the getters: “Get current order”, “Get current Npc”, “Fetch next Npc” and “Generate new level”.

52
6.2 Levels

As already mentioned, the difficulty of the game is in terms of the number of NPCs and in the number of orders by NPC. The objective was to have a game with maximum progression limited only by a maximum combination of NPCs and names. Each level can't have two NPCs with the same name and face. So the maximum number of NPCs in level is: sum of names X sum of faces.

The levels are generated by a class factory designed for this purpose and divided in multiple sub-factories. It follows two linear functions more precisely two slope functions. One to generate the number of NPCs per level and the other to obtain the number of orders per NPC per level.

- NPC per level function: (where x is the level number)

\[ f(x) = \frac{1}{8} \cdot x + \frac{2}{3} \]  \hspace{1cm} (6.1)

With this equation everytime a level is generated by order of the “Game Manager” the number of NPC is calculated.
According to the figure above and since the equation has a continuous domain, everytime the returned $y$ value is a non natural value, a round to int has to be made.

The slope of $2/3$ enables the game's difficulty to progressively get harder in a linear fashion without causing the player an abrupt stress.

The second function that generates the number of orders per NPC in each level is the following:

- The function of the number of orders per NPC per level: (where $x$ is the level number)

\[
 f(x) = \frac{1}{6} \cdot x + \frac{2}{3} \tag{6.2}
\]
Like the Figure 6.3, the plot above is a slope function with just a slight difference in the slope factor: being its higher value.

6.3 Scoring

The scoring system makes the count of the player's results and enables to track the progression towards memory stimulation.

The main focus when designing the scoring system was not only to reward correct answers but also to penalize wrong ones. In this context a wrong answer varies according to the game type. So for the three game types the scoring system is the following:
Looking at the table above, the summing principle is to multiply each correct item by two. The opposite occurs when a wrong item or name is recalled, but the penalization is of minus one times the number of items.

Since it will be a major challenge out of the scope of this project to be able to recognize a name from a pronoun for example, every word that does not match any known word by the game logic will be discarded and the level progression will continue.

A detail about how the way scoring was implemented under the hood. Every item and name in the game is of an enumerated type, an ordered type that maps each value to an unique int value, its identifier. It is particularly efficient when iterating and comparing values.

### 6.4 Scenes and layers

The concept of scene and layer is already understood at this stage. It is where the action happens. The layer is a fraction of the scene, it is a lightweight way to interact with the user for particular purposes.

There are various scenes and layers in the whole game lifecycle, more precisely five scenes (not including the base scenes used in the development process) and six layers. Two layers are on the menu scene and four in the game scene.

#### 6.4.1 Splash

The first scene is the “Splash scene” with the purpose of initializing the engine, loading the first resources and also a brief way to start the application, presenting some developer/institution information.
6.4.2 Language menu

After 1500 milliseconds the splash screen disappears and the next scene appears, the “Language Menu Scene”. This one is responsible for presenting the user with the language choice. This way the game will load all resources for the selected language and prepare the TTS to synthesize strings and also recognize the spoken words in the selected language. This menu is the following:
6.4.3 Main Menu

After the selection of the language, the “Main Menu scene” is then loaded. Here is how the menu looks like before exploring each option:

![Main menu scene](image)

Figure 6.7 - Main menu scene

This scene contains two layers, one for the “Scores” and one for the “Tutorial” as shown next.

![Score layer](image)

Figure 6.9 - Score layer

![Tutorial layer](image)

Figure 6.8 - Tutorial layer
After observing Figure 6.7 one can see that the “Main menu scene” contains four options: “Play”, “Scores”, “Tutorial” and “Login”. Two of them were already presented. The “Login” option initializes the authentication process, by starting with the selection of the account as explained in the authentication section of this document.

![Select account](image)

Figure 6.10-Select account

### 6.4.4 Loading

After hitting the “Play” button the loading screen is displayed:

![Loading screen](image)

Figure 6.11-Loading screen
The last option of the menu scene was “Play”. This option starts a new game for the logged user. Since this screen needs several resources, a “Loading screen” appears to the user to justify the time delay. The loading screen is simple and merely functional as one can see on Figure 6.11.

6.4.5 Game Scene

The last scene is the “Game Scene” where the game action takes place. This scene is composed of multiple layers, including a HUD, has different layers containing information about scoring or demanding approval to continue the game. A typical fase of the game is the following one:

![Figure 6.12-Game scene](image)

The figure above shows the major components that will be described next.

On the top left side of the scene is the HUD, giving constant information on the current level and score. At the center is an NPC, the characters that interact with the player. Just below the NPC, on the balcony, two types of strings are displayed, one is an information text such the one presented on the figure, the other information is exclusive to the “Call NPC” and is composed of the ready to be delivered menu.
The last element is the button at the bottom right side of the screen. This button has three states, the grey circle indicates that the button is disabled and won't respond to any click, at the appropriate time this button will change into a red circle (the media symbol for recording), meaning that the button is enabled and ready to record the player's audio commands.

![Figure 6.13-Record button ready](image)

After the player clicks on the button the recording starts, the button being now a red square, the media symbol for stop. After the click on the stop button, the audio starts being processed, asking the user to confirm the spoken words as shown on Figure 6.15.
Figure 6.14 shows the repeat option after the NPC has come for the first time giving the order. When he return this option is not valid.
The figure above is a report offered to the player after each delivery, informing of what was recognized, which are the correct answers, informing on what was wrong and also giving the partial score earned on this delivery. After clicking on the OK button, the game continues to the next NPC/order or to the next level.

Before a new level, the game type and the level number are presented so that the user knows to what kind of details he must pay attention more. The typical announcement of a new level is as follows:

![Figure 6.17-Starting a new level](image-url)
Chapter 7

7 Survey Results

Since the aim of this game was to improve memory through playing, the ideal tests would be those that could effectively confirm it, by using a group of volunteers. This type of tests would be more costly in terms of time and resources.

To overcome this situation and to present some interesting data about the impact of the game on people, a different approach was taken. This approach was to elaborate a survey able to check the users satisfaction and general appreciation altogether with some technical statistics such as how effectively the ASR worked.

7.1 Survey design

The survey is composed of five sections. They are the following:

1. “In which language did you play”?
   
   Here the user can choose between “Portuguese” or “English” since they are the two available languages for the game.

2. “Evaluate the following statements”
   
   1. “The game is too difficult”
   2. “It is difficult to understand the goal”
   3. “The level progression is too fast”

   For each of the previous statements the interviewee must choose between a range of options from “Strongly disagree” to “Strongly agree”. The other options in between options are “Disagree”, “Neither disagree nor agree” and “Agree”. Through the answers, a general opinion about the gamification design can be made. Future releases can take these into account so as to fit better the user’s expectations.

3. “How many times did you need to repeat yourself until the speech was correctly recognized?”

   The choices vary between 1 and 4+. This gives an overall estimation of how effectively the ASR worked from a interviewee point of view.

4. “Order the game type by preference”
   
   1. “Call the client”
2. “Match order”

This question helps understand which game type was preferred by the users.

5. “How likely is it that you would recommend this game to a friend or colleague?”

Finally the last question is the Net Promoter Score[52], NPS. This reports a general appreciation of the game, by directly asking the user, in a scale of 1 to 10 how likely he is to recommends this app to a friend or colleague.

This classifies the interviewees in three distinct classes: “the detractors”, people who gave a score between 0 to 6, “the passives”, for scores of 7 and 8, and “the promoters” for top scores of 9 and 10. Although this question is vague, it is a strong indicator of how the person enjoyed the final product.

From a technical point of view, the survey was built using an online tool, called “Survey Monkey”[53]. This website offers a free platform to build your own survey and also provides an easy way of distributing it, through a hyperlink.

### 7.2 Results and analysis

The survey was randomly distributed to adults, of both genders living in Portugal and studying or working in IST.

- Sample size n = 22

The first question has the following results:

![Chart 7.1-Most used language during game play](image)

Since the majority of the interviewees were from Portugal, it is natural that they played it in their mother language, and won with a percentage of 77%, representing 17 persons. The statistical analysis for this chart is (assuming the value 1 for “Portuguese” and 2 for “English”):
average rating = 1 \cdot 0.77 + 2 \cdot 0.23 \approx 1.23 \quad (7.1)

\text{median} = 1 \quad (7.2)

\text{mode} = 1 \quad (7.3)

Question number two, results:

In the chart above, it is visible that most people disagreed with the statements, the second most voted option was neither disagree nor agree with the statements. Assuming the value 1 for “Strongly Agree”, 2 for “Agree”, 3 for “Neither”, 4 for “Disagree” and 5 for “Strongly Disagree”, the statistical analysis is the following for each statement, starting with: “The game is too difficult”.

average rating = 3 \cdot 0.45 + 4 \cdot 0.41 + 5 \cdot 0.14 \approx 3.69 \quad (7.4)

From the chart, the next two values are extracted:

\text{median} = 4 \quad (7.5)

\text{mode} = 3 \quad (7.6)
For the second statement, “It is difficult to understand the goal”, the results are the following:

\[
\text{average rating} = 2 \cdot 0.14 + 3 \cdot 0.22 + 4 \cdot 0.59 + 5 \cdot 0.05 \approx 3.55
\]  
(7.7)

From the chart, the next two values are extracted:

\[
\text{median} = 4
\]  
(7.8)

\[
\text{mode} = 4
\]  
(7.9)

The last statement, “The level progression is too fast”:

\[
\text{average rating} = 2 \cdot 0.18 + 3 \cdot 0.23 + 4 \cdot 0.5 + 5 \cdot 0.09 \approx 3.5
\]  
(7.10)

From the chart, the next two values are extracted:

\[
\text{median} = 4
\]  
(7.11)

\[
\text{mode} = 4
\]  
(7.12)

After the statistical analysis, “disagreement” was clearly the dominating choice. Being always the mean and the mode, and only once not being the median.

The next chart reveals, the hit rate of the ASR, from a user perception:

Chart 7.3-ASR hits

Statistics:

\[
\text{mean} = 1 \cdot 0.59 + 2 \cdot 0.36 + 3 \cdot 0.05 \approx 1.46
\]  
(7.13)
From the chart, the next two values are extracted:

\[
\text{median} = 1 \quad (7.14)
\]

\[
\text{mode} = 1 \quad (7.15)
\]

From the Chart 7.3 and after analysis, one can observe that the ASR has recognized most of the items in just one go, according to the users experience. The errors can be related to multiple words having to be recognized, microphone issues or strange accents.

The chart for game type preference:

![Chart 7.4 - Game type preference](chart.png)

Statistics assuming 1 for “Match order” and 2 for “Call the client” for favorite game type (1st):

\[
\text{average rating} = 1 \cdot 0.68 + 2 \cdot 0.32 \approx 1.32 \quad (7.16)
\]

From the chart, the next two values are extracted:

\[
\text{median} = 1 \quad (7.17)
\]

\[
\text{mode} = 1 \quad (7.18)
\]

People tend to enjoy more the “Match order” game type. Almost 70% of the interviewees have chosen this game as their preferred one. The mode and median confirm that.

The last chart is for the Net Promoter Score, which is an overall appreciation of the game by asking how likely it is that you would recommend this app to a friend or colleague. The result was the following:
Statistical analysis, assuming the value 1 for “Detractor”, 2 for “Passive” and 3 for “Promoter”:

\[ \text{average rating} = 1 \cdot 0.14 + 2 \cdot 0.50 + 3 \cdot 0.36 \approx 2.22 \]  \hspace{1cm} (7.19)

From the chart, the next two values are extracted:

\[ \text{median} = 2 \]  \hspace{1cm} (7.20)

\[ \text{mode} = 2 \]  \hspace{1cm} (7.21)

The NPS of the game is the difference between the percentage of “Promoters” and “Detractors”. So looking at Chart 7.5, the NPS is 22%. Another observation is that, around 50% of the people, gave a score between 7 and 8 which represents the group “Passives”.

![Chart 7.5 - Net Promoter Score](chart.png)
Chapter 8

8 Conclusions and Future Works

This document described the whole development process of a proof of concept of the mobile game called “Talking Memory”, both client and server components. A project whose main objectives were to create a game with visual and audio stimula defying the player's memory and attention span.

Throughout this dissertation, all parts were explained. First by introducing all used components and justifying the choices of tools such as programming language and frameworks. The server part has shown the designed API that delivers the TTS and ASR services to the mobile application, the database and the authentication process. These were explained as well, referring the architectures and data flow in the server.

In this mobile game high level methodologies are described, such as factory and singleton patterns.

In the final part, the process of gamification like the game logic, the equations to create difficulty, the scoring and lastly the scenes designed were put into words.

The results obtained can in a way depend both on the environment where the player finds himself and his voice. Some voices are better recognizable than others. An effort was made towards minimizing this impact by, for example, allowing the repetition of the audio part and a confirmation button for the words recognized. This way the user can always agree with the recognized words.

For a future work a suggestion system where the recognized words would be compared to the game strings and demand the user's approval could improve the game experience. More game types, with different scenarios and NPCs could also turn the game more attractive. A nice way to comprove the effectiveness of this approach would be to perform tests on real people for a period of time, sadly it would envolve larger amount of time and resources and so should be done at a later stage.

No matter what, this game will have worked as a proof of concept, creating a memory game for the mobile world, something seldom achieved, proving that this type of game can have better results as a memory stimulation than an ordinary game.

Plus in my point of view, it has been a great experience, allowing me to have my first contact with the game development, the web services, the database management and functional programming.
Rererences


