UNIVERSITY OF CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

COMPUTER ENGINEERING

Specific Technology of
Information and Communication Technologies

FINAL DEGREE PROJECT

Allegra: Gamification-based Tool to Practice Melodic Dictation

Adrián Ollero Jiménez

July, 2019

UNIVERSITY OF CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Technologies and Information Systems Department

Specific Technology of
Information and Communication Technologies

FINAL DEGREE PROJECT

Allegra: Gamification-based Tool to Practice Melodic Dictation

Author: Adrián Ollero Jiménez

Supervisor: Diego Molero Marín

Supervisor: David Vallejo Fernández

July, 2019

TFG Adrián Ollero - Allegra
e-mail: adrian.ollero@alu.uclm.es

Tribunal:

Presidente: _____

Vocal: _____

Secretario: _____

Fecha de defensa: _____

Calificación: _____

Presidente	Vocal	Secretario
Fdo.:	Fdo.:	Fdo.:

A mis padres y
y mi hermano.

Abstract

Music students have to master numerous disciplines in order to become a professional musician. One of them is the perception of the world of sound, which has to become more and more perfect. Conservatories and music schools have adopted the practice of melodic dictation as the principal means of achieving this perfection. This practice consists of transcribing a musical piece normally played by a piano. Despite its popularity, it is a very difficult practice to master as it requires many hours of autonomous work outside the classroom for which there are not many mechanisms.

On the other hand, new technologies are increasingly present in our lives. In recent years, mobile devices are gaining more weight thanks to their power and versatility. In fact, music has greatly benefited from this technology thanks to the audio systems they incorporate and the many music services available. These technologies have not only revolutionized how music is enjoyed, but also how it is learned. There are thousands of systems for teaching instruments or music theory. To reinforce the effectiveness of educational systems, gamification techniques may be used, so that learning takes place in environments that engage the student by containing game elements.

Starting from the need for mechanisms for practicing melodic dictation outside the classroom, and taking advantage of the power and versatility of mobile devices and gamification techniques, it is proposed to develop a tool that, through the use of these techniques, helps music students practice dictation anywhere and in an environment that makes them feel motivated, in order to achieve a mastery of this difficult discipline.

Resumen

Durante la vida de un estudiante de música, este se tiene que manejar a numerosas disciplinas para poder llegar a ser un músico profesional. Una de ellas es la percepción del mundo sonoro, la cual tiene que ser cada vez más perfecta. Los conservatorios y escuelas de música han adoptado la práctica del dictado musical como principal medio para conseguir dicha perfección. Esta práctica consiste en transcribir una pieza musical ejecutada normalmente por un piano. A pesar de su popularidad, es una práctica muy difícil de dominar ya que requiere muchas horas de trabajo autónomo fuera del aula para lo que no hay muchos medios.

Por otro lado, las nuevas tecnologías están cada vez más presentes en nuestras vidas. En los últimos años, los dispositivos móviles son los que están ganando mayor peso gracias a su potencia y versatilidad. De hecho, la música ha sacado mucho partido de esta tecnología gracias a los sistemas de audio que incorporan y a los numerosos servicios de música disponibles. Estas tecnologías no solo han revolucionado el cómo se disfruta de la música, sino también cómo se aprende. Hay miles de sistemas destinados a la enseñanza de instrumento o teoría musical. Para reforzar la efectividad de los sistemas educativos, se pueden emplear técnicas de gamificación, de forma que se aprende en entornos que cautivan al alumno al contener elementos propios de los juegos.

Partiendo de la necesidad de mecanismos de práctica del dictado musical fuera del aula, y sacando provecho de la potencia y versatilidad de los dispositivos móviles y de las técnicas de gamificación, se propone desarrollar una herramienta que, mediante el empleo de estas técnicas, ayude a los estudiantes de música a practicar dictado en cualquier lugar y en un entorno que fomente su motivación, para así lograr un dominio de esta difícil disciplina.

AGRADECIMIENTOS

Este Trabajo de Fin de Grado nació como una idea pra ayudar a los alumnos de música pero ha terminado siendo un proyecto que me ha cautivado y ha revivido en mí la pasión por la música, y aunque yo he sido su ejecutor, hay muchas personas que, directa o indirectamente, han contribuido a hacer esto posible.

Me gustaría comenzar dando las gracias a mi compañero y amigo Diego por la confianza puesta en mí para que llevase a cabo este proyecto. También a mi director y amigo David, por su paciencia y sus sabios consejos.

Por otro lado me gustaría dedicar unas palabras a mis compañeros y amigos de Furious Koalas Interactive por confiar en mí y por la oportunidad que me han brindado para formar parte de su equipo. A Santiago por todo lo que me enseña y por la paciencia que tiene conmigo y a Carlos por ser un mentor para mí e inspirarme y apoyarme en todo.

A todos aquellos que han estado conmigo estos cuatro años, pero en especial a mis amigos Álvaro y Enrique, por todas las noches en vela y los momentos juntos. Sin vosotros no habría sido lo mismo.

Por último a mis padres y mi hermano. Sin vuestra confianza, ayuda, apoyo y sacrificio no sería quien soy y no habría podido llegar a donde estoy.

En definitiva, gracias a todo aquel que ha creído en mí.

Adrián Ollero Jiménez

CONTENTS

List of Figures xix

List of Tables xxi

Listings xxiii

1. Introduction 1

2. Objectives 7

2.1. General objective . . . 7

2.2. Specific Objectives . . . 7

3. State of the art 11

3.1. Musical education . . . 11

3.1.1. Basic concepts . . . 11

3.1.2. Melodic dictation as a competence . . . 14

3.1.3. Piano as learning tool . . . 14

3.1.4. Learning methodologies . . . 15

3.2. Technological tools in music learning . . . 16

3.2.1. Overview . . . 16

3.2.2. Digital systems to practice melodic dictation . . . 16

3.2.3. Mobile devices apps . . . 17

3.3. Gamification . . . 19

3.3.1. Introduction . . . 19

3.3.2. Application domains . . . 19

3.3.3. Gamification techniques used in musical apps . . . 20

3.4. Development process for mobile devices . . . 22

3.4.1. General background . . . 22

3.4.2. Monetization models . . . 24

3.4.3. Development tools . . . 25

4. Methodology **29**
4.1. Work methodology . . . 29
4.2. Resources . . . 30
4.2.1. Hardware resources . . . 30
4.2.2. Software resources . . . 31

5. Architecture **33**
5.1. Overview . . . 33
5.2. Gamification and GUI layer . . . 35
5.2.1. Exercise module . . . 35
5.2.2. Levels . . . 39
5.2.3. Statistics . . . 40
5.2.4. Challenges . . . 41
5.2.5. Song level and exercise note . . . 41
5.3. Communication layer . . . 42
5.3.1. Receptors . . . 42
5.3.2. Communication module . . . 47
5.4. Persistence layer . . . 48
5.4.1. Game instance . . . 48
5.4.2. XML reader . . . 50
5.4.3. Firebase realtime database . . . 50
5.5. Design patterns . . . 52
5.5.1. Singleton pattern . . . 52
5.5.2. Command pattern . . . 52
5.5.3. Observer pattern . . . 53

6. Results **55**
6.1. *Allegra* - Melodic dictation app . . . 55
6.1.1. Lessons practice . . . 56
6.1.2. Popular songs . . . 57
6.1.3. Statistics and custom dictations . . . 58
6.1.4. Challenges . . . 59
6.1.5. Exercise . . . 59
6.2. Landing page . . . 61
6.3. Work distribution . . . 62
6.4. Project statistics . . . 65
6.5. Development cost . . . 65

7. Conclusions **67**

7.1. Achieved objectives 67

7.2. Future work 68

7.3. Personal opinion 70

A. Exercises Solutions **73**

A.1. Lessons 73

A.2. Popular song 74

B. JSON tree in Firebase realtime database **75**

Bibliography **77**

LIST OF FIGURES

3.1. Notes representation and duration 12
3.2. Staff and signatures 13
3.3. Piano keys layout of two octaves 14
3.4. *teoría* learning web page. 17
3.5. *SonicFit* melodic dictation exercise 18
3.6. Simply Piano App 20
3.7. Dream Piano App 21
3.8. Jungle Music app screenshots 22
3.9. Number of apps available in leading app stores - 1st quarter 2019 (from www.statista.com) 23
3.10. Operating Systems used on mobile devices - May 2019 (from www.statista.com) 23
3.11. Android Studio IDE for app development 26
3.12. Unity Editor customized windows layout 27

4.1. Iterative and incremental methodology scheme 30

5.1. Allegra's architecture - General scheme 34
5.2. Exercise detailed flow scheme 36
5.3. A unique level selection screen for everything 39
5.4. Simplified Statistics flow scheme 40
5.5. Challenges simplified scheme 41
5.6. Domain entities and how they are related 42
5.7. Receivers communication with exercise 44
5.8. Simplified conceptual scheme of Microphone Detection Module 47
5.9. Example of some modules interacting with the Game Instance 49
5.10. SongLevel specification and XML relation with other classes 51

6.1. Logo designed for Allegra 55
6.2. Allegra's menu layout 56
6.3. Lesson selection screen 56

6.4. Popular songs purchase screen 57
6.5. Purchase confirmation dialog 57
6.6. Allegra's statistics system 58
6.7. Challenges button in main menu changes color according to unlocked challenges. 59
6.8. Statistics detailed dialog when a key is pressed 59
6.9. Exercise screen used to perform the lessons and songs 60
6.10. Implemented inputs methods 60
6.11. Final screen when an exercise is being achieved 61
6.12. Musical Dictation (Blue) vs Ear Trainer (Red) - Google Trends 61
6.13. Preview of Allegra's Landing Page 62
6.14. GitHub repository statistics charts 65

LIST OF TABLES

5.1. Relationship between the obtained stars and the number of mistakes . . . 39

6.1. Relationship between correct note rate and key color 58
6.2. Total cost calculation (taxes excluded) 66

LISTINGS

5.1. Pitch management in Notes Player 38
5.2. Code from Receivable 43
5.3. Core code Vpiano 45
5.4. Core of Mic 46
5.5. Core of Communication Module 48
5.6. Game Instance 49
5.7. Example of level specification 50
5.8. Singleton implementation 53

CHAPTER 1
INTRODUCTION

Music is part of humanity. It has been since the beginning of time to the present day, where it can be found in almost every imaginable field. Everyone knows music. Music can be found on the way to work, at home to relax, at mass events and in countless other situations. However, one of the areas in which it has had the most impact throughout history has been on education.

Going back to ancient civilizations, the Greeks considered artistic education, where music was situated, as one of the fundamental pillars, together with physical education, of the basic knowledges that every human being should know. With the passage of time, in the Middle Ages, other branches of great importance arose, such as Astronomy and Mathematics, but at the same level music was maintained, although each time with a more theoretical approach. This trend continued during the later periods until the Enlightenment. It was at this time that the first rationalist movements appeared, so that the new educational systems began to be based on rationalism and the scientific method [17]. This is how the bases of the current educational systems emerged, in which music goes practically unnoticed, and there are few students who maintain and show interest in increasing and reinforcing their knowledge about musical theory and practice.

Nowadays, anyone who wants to go deeper into music must go to specialized centers such as conservatories or music academies. There, both theoretical and practical training are taught and they can become professional musicians. This process is not simple as it involves multiple disciplines to master, and some with a high level of complexity. One of them is sound perception, which, for both students and professional musicians, has to be increasingly perfect. One of the pedagogical means adopted by most music training centres is *Melodic Dictation*. This practice consists in correctly writing a musical fragment played by an instrument, usually a piano.

The teachers of musical language are usually responsible for introducing the student to the practice of *melodic dictation*. Normally, the methodology used consists in a melodic piece played by the teacher in class in order to work on it. However, the student has to spend many hours outside the classroom practicing and solving exercises to achieve the necessary level of skill, this being one of the biggest challenges on this practice. A few years

ago, practicing at home was not only a difficult task, but sometimes practically impossible due to lack of resources. The only way to do this was to have a recording device to take to class and record fragments of musical pieces played by the teacher as proposed exercises in order to have material with which to work outside school hours, although in a limited way. It is for this reason that the road to travel to achieve a mastery of sound perception is long and complex, and requires a lot of autonomous work solving exercises, increasingly repetitive.

On the other hand, new technologies are increasingly present in everyday life. This is due to their potential, ease of use and accessibility. With a device, no matter how small, connected to the Internet, it is possible to have access to any type of information. This makes it possible to expand the frontiers of any educational model, including musical learning. Information Technologies have allowed students and teachers to share spaces in which to host agendas, practices and exercises. By making use of these technologies, knowledge can be acquired in any conceivable field. In fact, anyone with a little interest can acquire advanced knowledge about scales, chords, intervals and even melodic dictation. In this way, a student who needs resources and material to practice dictation only has to use a device with Internet to access thousands of exercises with which to practice. However, in spite of this, there is still a problem whereby students continue to find it difficult and complex to master this discipline: the number of hours of self-study that many abandon their musical studies, mainly for lack of motivation [2].

The market for mobile devices has gained a lot of importance in recent years [20]. Already they can be found mobile devices with a similar or even higher power than some desktop devices. Not to mention their versatility. Thanks to the amount of peripherals they incorporate, such as cameras, touch screens, etc., a mobile device can be converted into, for example, a photographic studio or a personal cinema. However, this would not be possible if there were no applications that knew how to take advantage of the power offered by these devices.

An application is nothing more than a computer program designed as a tool to allow the user to perform a specific task. There are applications of all kinds: to listen to music, to consult social networks, to send instant messages, to watch series and movies, to play games of thousands of different themes... There are even applications aimed entirely at education.

There are more and more applications aimed at educational topics. In fact, thanks to the ease of carrying one of these devices around all day, it is the ideal platform for this type of system [13]. Why? Because almost anyone has these devices at hand at any time, and uses them as a distraction in public transport, at home, in the office... and these are ideal times for those users who are concerned about a topic in which they want to train and can spend some time taking advantage of any moment to do so.

Thanks to technologies such as touch screens that allow you to touch elements simulating real objects, or built-in studio-quality audio systems, mobile devices are perfect for learning and practicing music [13]. There are many applications for this purpose in the app stores. There are some whose objective is focused on the teaching of an instrument, such as the piano or guitar, others to learn theory and even some focused on the practice of melodic dictation. Although these apps play with the benefit of taking advantage of the power offered by mobile technology, they run the risk of failing in their mission. All these applications must avoid falling into the problems of many traditional methodologies, turning out to be of little interest to the students and failing in the transmission of knowledge. To avoid this, some techniques can be applied to ensure an enjoyable learning process, resulting in a fun and challenging experience in which learning is not boring anymore.

These techniques are called *gamification techniques*. According to the expert Yu-Kai Chou, gamification is understood as "the craft of deriving fun and engaging elements found typically in games and thoughtfully applying them to real-world or productive activities" [4]. The aim of gamification is to use elements that are used in the context of games to engage and motivate the user. These techniques are increasingly used in business to increase employee productivity or to improve the customer experience. In recent years, gamification has also been introduced in mobile applications. In this context, its goal is to get the user to use an application as long as possible. A typical example could be rewarding the user for successfully performing tasks within the app, measuring progress, or even showing a ranking of the best users so that it can be compared and the motivation to continue progressing to reach the best positions arises.

However, for these techniques to be effective it is not enough just to introduce them into the design, but they must be sustained on a solid foundation of a mechanics to exploit. This means, that it is of great importance that gamification techniques have to be a support, and never the basic pillar of a system. They have to help the user to have positive sensations and enjoy the time spent using the application.

In fact, thanks to these techniques, so-called *serious games* emerge. These games, also known as "*formative games*" or "*educational games*", are video games designed for a different purpose than simply for fun. Today they are very useful tools in many fields [10]. There are games aimed at a wide variety of audiences. They are even important tools in the training of professionals, in education, health, military operations, etc. These games, by means of gamification techniques and a user-friendly appearance and mechanics, allow the user to acquire knowledge by means of entertainment.

Back to the problem previously described with respect to the practice of melodic dictation, it has been discussed how difficult it is for students to practice it and how much autonomous work is involved in mastering this discipline. However, now, thanks to new technologies, mobile devices and the good use of gamification techniques adapted to this context, new

fronts are opening up to make the hard process of perfecting sound perception much more entertaining and challenging and even enjoyable for the user. Thanks to this motivation, it was born the idea of designing and developing *Allegra*, a system with these characteristics, which combines the traditional methodologies of learning and improving musical dictation, such as listening to a melody and transcribing it, with gamification techniques and the power and versatility of mobile devices, with the aim of making this practice a pleasant process.

The system object of this development must fit and serve as a support to the current techniques of musical teaching, always acting as a complement. In this way, *Allegra* follows the traditional methodology of melodic dictation, proposing exercises that reproduce melodies, organized progressively, and providing mechanisms for the user to introduce the answer. These exercises have the task of returning feedback to the users, allowing them to know at any time how well they are doing. On the other hand, with the aim of limiting the effort of the user in the tasks of correction and evaluation of the exercises, the system has the capacity to correct the proposed dictations automatically in real time, avoiding one of the toughest tasks for the students.

Moreover, the use of gamification techniques intends to achieve a game approach and an environment where repetitive practices are a fun task. Among the techniques used are the rewards obtained depending on the exercises performed. With these rewards it offers the possibility of purchasing exercises based on popular songs, thus promoting the user's motivation when working with melodies that remind him or her a different environment from the academic. There is also a system of challenges, with generous rewards but which can only be taken every certain period of time, motivating the user to use the application again and again.

Additionally, it includes the functionality of a statistics system that allows, at a glance, to know the strengths and weaknesses of every user, offering them customized exercises to deal with such weaknesses.

Finally, and in order to make the application as accessible as possible, the development has been oriented to mobile devices, so that, apart from taking advantage of all the benefits previously discussed, it may be available in the app stores and can be used at any time and place.

In this document, the entire development process is detailed, as well as the results obtained and the previous studies that have been carried out. The structure that has been followed is briefly summarized next:

- *Chapter 2: Objective*
  In this chapter, a detailed description of the general objective that this project aims to achieve is given first, and then the specific objectives that will make it possible to complete the first one are broken down and explained in detail.

- *Chapter 3: State Of Art*
  Every project has a background on which it is based or inspired to achieve its objective. In this case, this chapter is reserved to discuss some musical concepts of considerable relevance as well as previous systems that have served as a study prior to the development of this system.

- *Chapter 4: Methodology*
  This chapter deals with the methodology used during the completion of the project as well as a compilation of the resources used to carry it out.

- *Chapter 5: Architecture*
  The architecture chapter focuses on a technical description of the system, detailing the problems that have arisen during development, which are the solutions adopted and what advantages they offer.

- *Chapter 6: Results*
  This chapter will review the result obtained at the end of the development, how it has been reached and an estimation of the cost project.

- *Chapter 7: Conclusion*
  This last chapter concludes with an assessment of the initial objectives and whether they have been completed. It also briefly discusses possible future work. It ends with a personal conclusion from the author of this document and system developer.

CHAPTER 2

OBJECTIVES

In this chapter it is intended to carry out a detailed description of the objective that this project aims to achieve, summarizing, firstly, the general objective, and then describing in detail its specific objectives.

2.1. GENERAL OBJECTIVE

The general objective of this project is *the design and development of a tool that facilitates the learning of melodic dictation through gamification techniques, which will be deployed on mobile devices*. In this way, it is intended to use a market highly exploited nowadays, as are mobile devices, so that learning and the hours it requires are more bearable and enjoyable by framing the application within a playful context thanks to gamification techniques, but preserving the bases of a good methodology of musical teaching.

2.2. SPECIFIC OBJECTIVES

Based on the general objective defined above, a number of specific objectives can be identified which, once all of them have been achieved, will make it possible to complete this main objective. The specific objectives of this project are as follows:

1. **Automatic dictation evaluation.**
   When students use a tool that will help them practice and improve an area such as melodic dictation, they expect to receive feedback on the actions they take at each moment, to know if they are correct or incorrect, and information that reports on their progress throughout the time of use. For this reason, a mechanism must be included so that each of the exercises returns feedback during and at the end of the exercise, and all this information must be registered in order to create a report that the user can check at any time on their progress and thus be able to put more emphasis on those concepts with more negative statistics.

2. **Progressive learning**.
   The learning process, regardless of the field, is slow and can be excessively complex if it is not adequately adapted to the learner. Special attention should therefore be paid to ensuring that this process is progressive in such a way that a level of complexity cannot be reached until it has been ensured that the previous knowledges are fully understood. In addition, since users can have very different levels when starting to use the application, it must be able to start from the most basics for those who are less familiar with melodic dictation, but at the same time allowing those who are already hardened in the subject to progress quickly and reach their real level in a short period of time.

3. **Student motivation by means of gamification techniques**.
   Thanks to the use of gamification techniques it is intended to provide the application whit mechanisms that motivate and engage the user to use it as long as possible promoting a faster growth in melodic dictation. In this way, a system of rewards will be introduced that will allow the user to increase their level of experience or acquire popular songs that are fun to work with. It will also include a system of weekly and daily challenges, which when completed will reward the user generously, encouraging him to use the application everyday.

4. **Adaptability to enable the use of the system anywhere, anytime**.
   One of the main features of mobile devices is that they can be taken and used by the user anywhere and at any time. Then, it must be borne in mind that the connection to the network may vary or even be null in certain places, affecting the user experience if it is of critical importance for the performance of the system. Therefore, the application has to be designed so that it works transparently to the network connection, although it will use it to store certain information and avoid the use of local storage prone to loss.

5. **Good user experience**.
   It is important to ensure that the user's experience while using the application is both pleasant and not frustrating, so that they do not feel the need to drop or stop using the application. In order to do this, a friendly, visually attractive interface must be designed and its use must be simple, comfortable and intuitive so that the user feels that the application adapts to him. In addition, since each user may be familiar with the use of a different musical instrument, the application must be able to work transparently to the medium used to interact with it.

6. **Scalability to integrate new dictations and levels**.
   In a level-by-level application, attention should be paid to making the inclusion of new ones easy, quick and simple. Also, since each of these levels can be played in

many different ways, new game modes must also be implemented very quickly and without affecting the elements of the application already implemented. To achieve that, both the design and the implementation of the system must be focused on making it scalable both to add new levels and songs and to be able to implement new game modes.

CHAPTER 3

STATE OF THE ART

This chapter will deal with the foundations on which this project is based. First, some basic musical concepts will be introduced, which will later be decisive in the development. Among them, there will be a more theoretical content to later define and understand what melodic dictation is and how it is practiced, as well as its importance among professional musicians. Next, there will be an overview of how technology has been used in recent years as a complement in music classrooms and schools, as well as tools designed specifically for melodic dictation for both desktop and mobile devices. In addition, there will be a brief review of what it is, how it is used and what advantages gamification techniques have, especially in educational environments, with some real and present examples. To conclude this chapter, it will be reviewed the most important tools currently for development in mobile devices, the importance of this market, how to make a profit with it and why Unity has been the tool chosen to carry out this development.

3.1. MUSICAL EDUCATION

Music accompanies us in our daily lives. We can find it at work, in a party, studying or while taking a moment of relax. It can be enjoyed at any time. During our life, we listen to many different songs and musical pieces. However, music is something very complex. It requires years of study to fully understand and interpret it.

3.1.1. Basic concepts

The musical note

A *beat* is a pulse of time. Everything around us has a rhythm to it, from a ticking clock to car engines, including birds and fishes. These *rhythms* are formed by regular or irregular beat patterns.

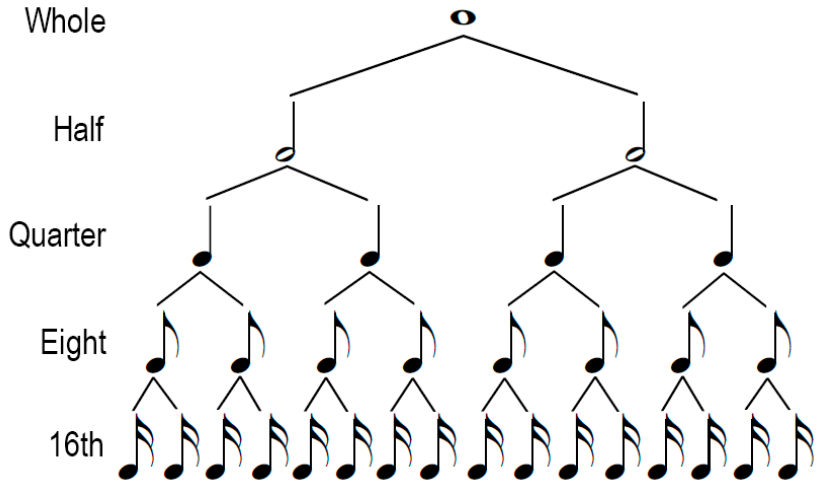


Figure 3.1: Notes representation and duration

A *note* is basically the representation used in music notation to tell the performer how long and how often to play a certain musical pitch within the beat [14].

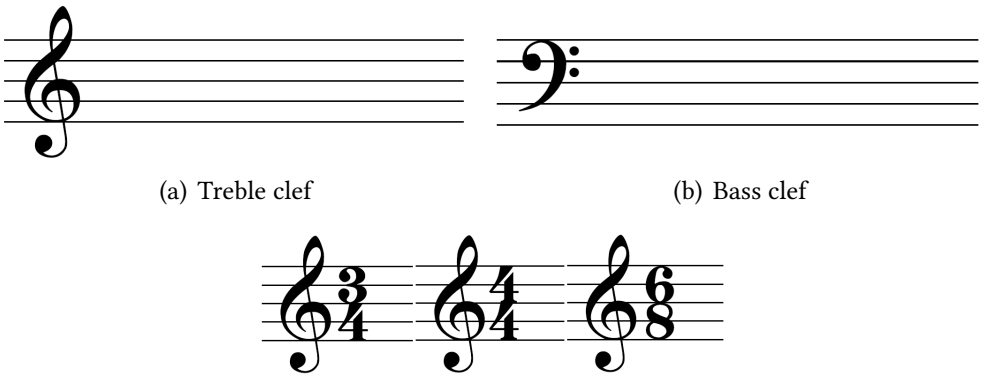
The *pitch* specifies the frequency of a note and is represented locating the note on a line or space on the staff, as explained in 3.1.1. The *time value* is the duration of a certain sound and determines the rhythm and melody of the resulting piece of music. Every note is represented by three components: the *note head*, the *stem* and the *flag*. According to the duration of the note, it has a different combination of the mentioned components. For example, a whole note is represented by and hollow head, a quarter note by a filled head and a stem, etc. (see Figure 3.1).

The Staff and it signatures

The *staff* is where music notes are written [14]. A staff is made up by 5 parallel horizontal lines and 4 spaces, so notes are written in either a line or a space of the staff. Each of these lines or spaces hold a different pitched note and to determine which particular note are meant by each line or space a *clef* is written at the beginning of the staff.

Even though there are some others, the two clefs used in most books and taught on music schools are the *treble clef* (see Figure 3.2(a)) and the *bass clef* (see Figure 3.2(b)). The *treble clef* is used for higher notes and the bass clef for lower notes.

Rhythm can also be represented on a staff. In printed music the *time signature* is represented as a pair of numbers, one over the other, and they are written right after the clef. This time signature informs of two things: the top number tells the number of beats in each measure, while the bottom number specifies which note value equals one beat.



(a) Treble clef (b) Bass clef

(c) Most popular time signatures

Figure 3.2: Staff and signatures

Tones and semitones

Each note on a music scale represents, from a physical point of view, a wave frequency, which is closely related to the pitch of that note but not the same. *Frequency* is an objective, continuous, scientific attribute that can be measured, however, *pitch* of a note is the subjective perception of a sound wave that can not be directly measured [7].

The frequency spectrum perceived by the human ear goes from 20Hz to 20.000Hz. The discretization of this continuous range results on 1024 notes (2^{10}). The separation between two adjacent notes is called *semitone* and he separation between two notes separated by another note is called *tone* (two semitones) [7].

The frequency associated to each note can be obtained, from a reference frequency n_r (usually the reference frequency is 440Hz, note A), applying the equation 3.1. This equation calculates the frequency of the target note s, taking the reference frequency n_r, the relation between the same note in two different octaves, which is 2, the number of notes on a octave 12, and the distance s in semitones between the reference note and the target [5].

$$n = n_r * 2^{s/12} \tag{3.1}$$

As can be seen the notes are actually a name or label given to a specific wave frequency value. For example, instead of calling a note 440Hz, which is the frequency it represents, it is called A (or La), or instead of 392Hz it is G (or Sol), and so on.

The twelve-note chromatic scale is the most common way of organizing the twelve notes composing an *octave*. From these twelve, only 7 has its own name, there are five which name is derived *C, C#, D, D#, E, F, F#, G, G#, A, A#, B* in the English notation system and *Do, Do#, Re, Re#, Mi, Fa, Fa#, Sol, Sol#, La, La#, Si* in the Latin notation system.

Derived notes are those in which a semitone varies with respect to the base note. The base note is known as the natural note and represents the note itself. A sharp note (#) is a note one semitone higher than its natural note. A flat note (b) is a note one semitone lower than the natural note. This implies that a derived note has two names, depending on whether it is named as relative to its major note or its minor note. For example a *C#* and a *Db* are the same note but with a different name as the natural reference note changes. As a general rule, a homogeneous representation is used to represent derived notes, i.e. only sharps or flats are used.

3.1.2. Melodic dictation as a competence

During the period of time covered by their musical studies an through his future professional life, the music students must improve their "tuned ear", becoming more and more perfect . One of the pedagogical means introduces into teaching in Conservatories and Music Schools is "*Melodic Dictation*". This practice consists on listening and writing correctly on a staff a music fragment played on an instrument, usually a piano. It is usually introduced in the early years until reaching the domain of *melodic dictation* of up to three or four harmonic voices [24].

3.1.3. Piano as learning tool

The piano is the most popular instrument used by music teachers because it is very easy to understand notes, scales, intervals, etc thanks to the keys layout. IT also has many advantages over others that make this an ideal tool for learning, such as the ability to play several notes at once, or that each key is perfectly tuned.

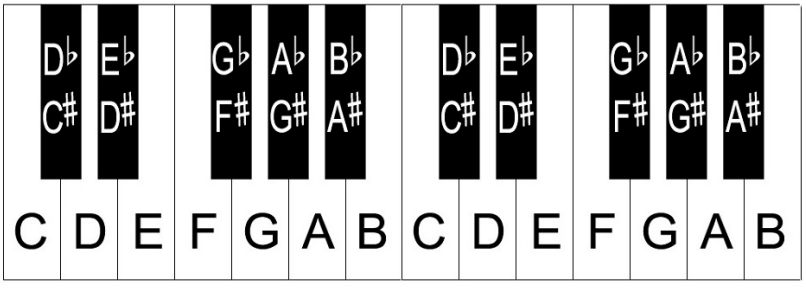


Figure 3.3: Piano keys layout of two octaves

A *piano*[1] is a musical instrument played by keys. Each of those keys is connected to a hammer, which hits a string when the key is pressed, producing a sound with a specific

[1] https://es.yamaha.com/es/products/contents/musical_instrument_guide/piano/index.html

frequency, a note[2]. There are 88 keys in a standard[3] key board, 36 black keys and 52 white keys, which are a total of 7 octaves. The white keys correspond to the natural notes (*C, D, E...*) and the black ones corresponds to the flat and sharp notes (*C#*, *D#...*) as is shown in figure 3.3.

Most music schools use a piano, played by the teacher, to practice *melodic dictation*.

3.1.4. Learning methodologies

Musical language teachers have always considered the practice of *melodic dictation* a good formative method, including it in all music courses. However, teachers usually face two problem when carrying out this practice with students: lack of methodologies at the time of teaching *melodic dictation* and the impossibility for the student to practice it out of the classroom [24]. In fact, there were no mechanisms that allowed the student to practice melodic dictation without attending class, having a reinforcement teacher or spending long hours in front of a piano until every note was assimilated. Then, teachers should plan progressive lessons to assure an improvement in the student skills to solve the first problem found. For the second problem, years ago, teachers encourage students to bring a tape recorder, so that a piece was played and recorded at the end of the class so students could work on it at home.

Despite these efforts, *melodic dictation* is an activity that curiously, it is accompanied by negative connotations for students, and as a consequence of this, the displeasure of his realization in the lessons.

A study carried out on students of many conservatories from Madrid obtained a rather unfavourable result regarding the opinion of this practice. It consisted on survey formed by open and close questions about *melodic dictation*, filled by the students. The result show that the students perceive the importance that is given to the *melodic dictation* practice in the lessons, but they also recognize experimenting insecurity, anxiety and other unpleasant sensations when they perform this activity [2], and argues that an ICT support can help make this practice more bearable and enriching for students.

[2] https://es.yamaha.com/es/products/contents/musical_instrument_guide/piano/mechanism/mechanism003.html

[3] https://es.yamaha.com/es/products/contents/musical_instrument_guide/piano/trivia/trivia007.html

3.2. TECHNOLOGICAL TOOLS IN MUSIC LEARNING

3.2.1. Overview

Information and Communication Technologies are becoming increasingly important in all areas of everyday life. In education, various technological methodologies are implemented every day as a valuable support and complement to the teaching activity [11]. Of course, musical learning is also included.

Students at present enjoy the musical learning process more when it is supported by Information Technologies tools. According to these students, the most commonly used tools in the classrooms are CD and DVD players and projectors for presentations of didactic content and documentaries, and computers for web searches for reinforcement content, as well as music instruction tools [13].

An example of computed-based system for musical learning in is *LenMus*[4] a free software that allows and facilitates the practice of theory, musical language and auditory development by customizable and leveled exercises. It also provides a development tool called *Lomse*[5] designed to provide software developers with a library to add capabilities to any program for rendering, editing and playing back music scores.

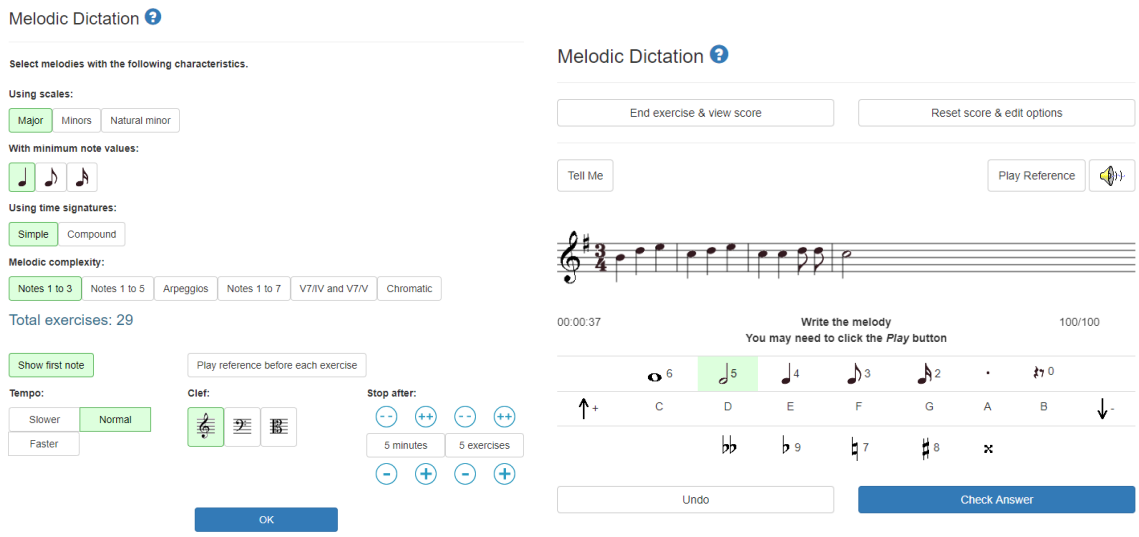
3.2.2. Digital systems to practice melodic dictation

A system to practice *Melodic Dictation* is an application that allows the development of the ear using organized exercises of recognition of scales, intervals, etc [2] by means of ICT tools.

teoría[6] is a good example. It is a music theory web page for music learning, that provides melodic dictation exercises. Before stating a dictation, the system allows you to provide some rules about melody complexity, clef and tempo, among other setting (see figure 3.4(a)), and then, a set dictation exercises begin (figure 3.4(b)). As the exercises can be fully customized is the perfect tool for beginners.

Other example is *Toned Ear: Ear Training*[7] a web page full of many different exercises about theory, scales, intervals, etc, but with a specific module for practicing Melodic Dictation[8]. The customization parameters are very basic and the difficulty of the exercises are quite high.

[4] www.lenmus.org
[5] https://github.com/lenmus/lomse
[6] http://www.teoria.com/index.php
[7] https://tonedear.com/
[8] https://tonedear.com/ear-training/melodic-dictation-practice



(a) Previous settings (b) Exercise

Figure 3.4: *teoría* learning web page.

One more example is *SonicFit*[9]. *SonicFit* is a free web platform developed thanks to the work of students. It has lessons in music theory and ear training organized in a comprehensive curriculum. After learning about each topic, students take quizzes to check for understanding, and then work exercises to develop fluency. The melodic dictation exercises, shown in figure 3.5(a) progressively increase the complexity and work in a very intuitive way, being a good option for beginners. On the other hand, these exercises have the disadvantage that the system does not provide a correction in real time, but offers some templates of answers and at the end of an exercise reveals the solution for the user to self-correct.

Finally, the streaming platforms will be treated as a tool widely used by music teachers to upload exercises and dictations so that students can use them to practice at home. Thanks to platforms such as *YouTube*[10] or StreamCloud[11], in which uploading content is free, it is easy for music language teachers, responsible for teaching dictation, can host their exercises, previously recorded and treated during the class, and so students can access them freely, free and at any time.

3.2.3. Mobile devices apps

The App Stores are full of all kinds of applications, and of course, melodic dictation is one of the subjects for which hundreds of tools have been developed.

An example is *Oído Perfecto*[12], a free Spanish application focused on different topics of musical learning such as tempo, intervals, chords, etc. The ”absolute note detection”

[9] http://sonicfit.com/
[10] https://www.youtube.com/
[11] https://soundcloud.com/search/sets?q=melodic%20dictation
[12] https://play.google.com/store/apps/details?id=com.evilduck.musiciankit

(a) Exercise page (b) Answering template

Figure 3.5: *SonicFit* melodic dictation exercise

exercises, the basics for melodic dictation, are based on playing a sound and then the user has to choose the key of the note on the virtual keyboard, depending on what he has heard. There are many other types of exercises adapted to each of the topics that can me studied within the app. As it deals with very simple subjects at the beginning and can reach other very complex ones, it is the perfect application for beginners and experts.

Toned Ear: Ear Training[13], tool mentioned in 3.2.2, evolved over time and now it is also available as an app, It offers the exact same features and the exact same look. It is the same powerful tool but as an app for a mobile device. Its cost is 4.89€.

MeloDic (ear trainer)[14] is an app specially designed to practice melodic dictation. The exercises consist on a familiar melody that is played while most note are drown in the screen staff. However some of them are missing, so is the user who, by means of a virtual keyboard, must play the note that he has just heard. This app costs 1.00€, but there is a Lite version limited in features, aimed to test it.

Although the app stores have numerous applications for learning and practicing melodic dictation, all are based on the principle of reproducing note or melody for the user to identify it, in a simple and basic way, since all have a similar appearance and functionality.

[13] https://play.google.com/store/apps/details?id=com.tonedear.tonedear
[14] https://play.google.com/store/apps/details?id=darktools.melodyLibrary

3.3. GAMIFICATION

3.3.1. Introduction

Gamification can be defined, as proposed by Yu-Kay Chou [4], as "the craft of deriving fun and engaging elements found typically in games and thoughtfully applying them to real-world or productive activities". It is proposed to focus the design of processes or activities on the motivation of the people who are going to carry them out instead of the pure functionality of the system. This is known as *human-focused design*.

Other definition for *gamification* is "the use of game design elements in non-game contexts" [8]. This definition relates two very different concepts, therefore, consider them separately. The term *game design elements* refers to the those design aspects than are commonly found in games and help to engage people. On the other hand, the term *non-game context* refers to anything outside a game, from simple everyday tasks to complex business processes.

3.3.2. Application domains

The concept of making things game-like is something that has been accomplished throughout the history. Humans tried to get motivated in daily tasks, such as hunting or gathering, by any kind of small competitions, where small groups of people compete against each other.

Nowadays, this techniques have evolved until the point of being used in lots of fields. For example, is quite common that big restaurant chains rate clients according to their amount of orders, giving more benefits (like special offers or trying a new product before it is on sale) to those who are more frequent, such as McDonald's[15] or Fosters Hollywood[16]. In fact, gamification principles are not only applied to customer engagement but also to improve employees productivity, such as Freshdesk[17], a helpdesk software program for customer support centers, that rewards employees with some kind of trophies when achieving a challenge, maintaining the employee motivated and the client satisfied as the service received is better.

Of course this are just two examples applying gamification to business processes, but gamification can be applied to daily tasks, education, etc.

In order to ensure the success of gamification it is quite important to apply correctly the gamification principles and not only the use of the most basic techniques. Most people that

[15] https://www.mcdonalds.es/
[16] https://fostershollywood.es/fosterianos
[17] https://freshdesk.com/es/

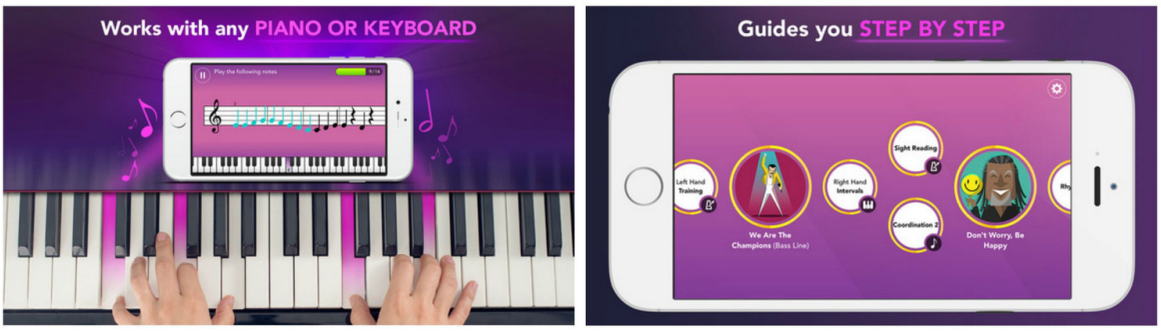


Figure 3.6: Simply Piano App

work in gamification think that adding scores, budgets, rewards, etc. to a process or product make it funnier and more engaging. This is an error. The important thing to be successful applying gamification techniques is to analyze the process or product to understand how we want to make the user feel, motivating him in the most tedious tasks, ensuring that these are carried out while the user enjoys the process.

3.3.3. Gamification techniques used in musical apps

Taking a quick look at the app stores, a variety of musical applications can be found. These range from learning applications, for example of a musical instrument, to playful ones. However, in the most successful ones, we can see a variety of elements derived from gamification.

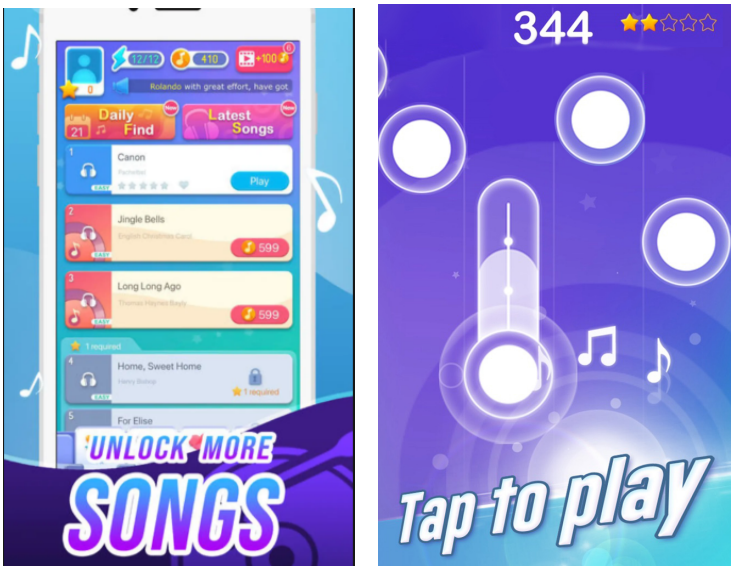
Simply Piano

Simply Piano[18] is one of the most popular applications to learn music. It is aimed to allow people to learn to play the piano at home. It is organized in courses, each one focused on a different musical aspect. As the lessons are completed, the knowledge is reinforced with popular songs adapted to the current user level (Figure 3.6). This songs help to maintain the user engaged as he can check that what has been learned can be applied to real songs, providing a the sense of quick progress, motivating him to keep using the app.

Dream Piano

Dream Piano[19] is actually a game, but it implements gamification techniques that could be used on any music learning app. The game consists on tap over circles that fall through the screen and play a melody. There is a set of songs that are unlocked according to the progress over previous songs, the more you play, more song get unlocked, and there is a set

[18] https://www.joytunes.com/
[19] https://play.google.com/store/apps/details?id=com.eyu.piano



(a) Main menu (b) Playing Level

Figure 3.7: Dream Piano App

of songs, which are famous current songs, that are unlocked only with in-game currency, that is earned when levels are achieved.

It offers daily challenges based on user's music preferences, in which currency and stars can be earned. There is also an online mode, where a user competes against three others in real time, winning some rewards depending on the position obtained.

In this game everything is focused on earning in-game currency to unlock more and more songs, being this songs those preferred by the user so he enjoys and has fun while playing and using the app.

Jungle Music

Among the music learning applications, *Jungle Music*[20] is one of the best known aimed at children. Its aim is to make known the musical language and especially the reading of the staff from very early ages since it has managed to turn this task into an attractive and entertaining game.

The application is divided into levels organized by islands, according to the clef, and each of these levels is responsible for working a specific set of notes related to that clef. The exercise or level plays notes one by one and shows them on the staff so that the user must identify them. Thanks to its graphic user interface adapted to a child audience, it manages to turn an apparently boring and repetitive task into something fun for children. In addition it offers the possibility of generating customized exercises, according to the necessities of the user, and gives to choose between a set of instruments to work the exercises with it.

[20] https://play.google.com/store/apps/details?id=air.junglemusicfree&hl=es

(a) Level selection level (b) Playing Level

Figure 3.8: Jungle Music app screenshots

A rewards system is also implemented so any time the user achieves a level it gets stars, representing how good it was performed, and notes that work as currency to buy new resources.

It is a clear example of how a good use of gamification techniques can turn unattractive tasks into educational games, in which while motivating and entertaining the user some musical knowledge is boosted.

3.4. DEVELOPMENT PROCESS FOR MOBILE DEVICES

3.4.1. General background

Mobile applications market has experienced a huge and fast expansion over the last 10 years as mobile platforms keep evolving and the increasing needs of users in a wide variety of applications [16]. The development process for these applications comes with unique characteristics well differentiated from others. These are: high level of competitiveness, short time of delivery, mobility, portability, specific and constantly changing capabilities, different and incompatible systems, among others [1].

It is quite important to take into account that the most popular market places are overcrowded. Developers upload thousands of applications every day to these markets. The most popular ones are *Google Play Store*[21] and *Apple App Store*[22] as shown in figure 3.9. This implies that most mobile apps are developed for *Android* and *iOS*, being, as well, the two most popular operating systems for mobile devices. In May 2019[23], Android leads the market, 75, 27%, followed by iOS, 22, 74%, and then some other operating systems (see figure 3.10).

[21] https://play.google.com/store/apps
[22] https://www.apple.com/es/ios/app-store/
[23] http://gs.statcounter.com/os-market-share/mobile/worldwide

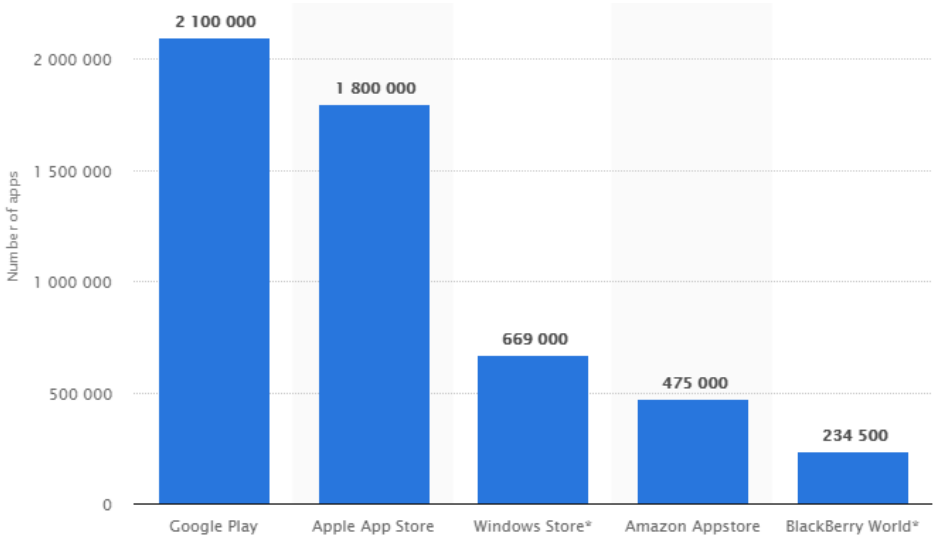


Figure 3.9: Number of apps available in leading app stores - 1st quarter 2019 (from www.statista.com)

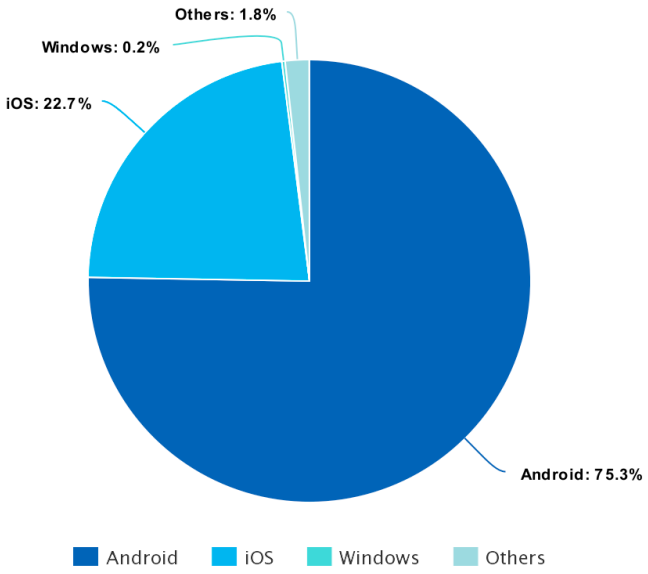


Figure 3.10: Operating Systems used on mobile devices - May 2019 (from www.statista.com)

3.4.2. Monetization models

Monetization means converting an object or asset into legal tender or money in any form of operating business [18]. An *App Monetization Model* can be defined as a framework showing how a mobile app can earn revenue generating profits over the production costs. There are many different app monetization models that adapt to different types of businesses and ways of making money from an *app*[24].

In-app Advertising

In-app advertising refers to apps on which advertisements periodically pop up [20]. These apps are generally free for users to download and use them and profits are earned based on advertisement impressions. Gaming apps also offers the user the possibility of see and advertisement in exchange for in-game rewards. *Flappy Birds*[25] is an example of a free gaming app that was downloaded over 50 million times, earning over $50,000 per day only through in-app advertisement model [22].

Freemium apps

The term *"freemium"*, comprising the words 'free' and 'premium', describes a monetization model that earns the benefits from *in-app purchases* [20]. Users make these in-app purchases to buy extra resources or extra functions, even to upgrade to full content or to remove advertisements that pop up on the free version of the application. A successful example currently is Super Cell's *Clash Royale*[26], a free to use game for mobile devices that has generated billions of dollars only through in-game purchases to get game virtual currency [9].

Paid apps

This is the most basic monetization model used for mobiles apps. *Paid apps* basically consists on purchasing the app itself. Developers upload the application to an app store, such as Google Play Store[27] or Apple App Store[28] where users can download it for a stated price. This apps generally do not offer in-app purchases nor pop up advertisements. An example of app that uses this monetization model is Rockstar's *Grand Thef Auto: San Andreas*[29], on sale by $6.99.

[24] "App" stands for Mobile Application
[25] https://flappybird.io/
[26] https://clashroyale.com/es/
[27] https://play.google.com/store/apps
[28] https://www.apple.com/es/ios/app-store/
[29] https://play.google.com/store/apps/details?id=com.rockstargames.gtasa&hl=es_419

Paidmium

Paidmium model involves both paid downloads and in-app purchases for additional revenue. In 2013, around the 61% of paidmium apps revenue came from direct downloads, while the remaining 39% were from in-app purchases [20]. This model is not so popular nowadays but there are some apps that still using it. A very popular app that leads the stores trends was *Minecraft Pocket Edition*. It was sold by $6.99 in Google Play Store[30] and $7.99 in Apple App Store, but also offers special in-app sales, getting a total revenue of $1M only in Christmas 2013.

Subscription

Software-as-a-Service is a subscription based monetization model in which companies lure users into long-term payment approach [18]. It can be *pay as you go* subscription, where the user are charged based on the usage rate, or *tiered pricing* where the services offered by the application are limited according to the user usage, so they can choose from a set of different plans with different prices. *Tiered pricing* is the most popular approach for subscription apps. It can be found, for example, in Netflix[31] or Spotify[32].

3.4.3. Development tools

As mentioned above, the market for mobile applications has been growing enormously in recent years. This means that new frameworks and development tools are constantly emerging.

Android Studio

Android Studio[33] is the official integrated development environment for the Android platform based on IntelliJ IDEA. Android Studio offers a powerful text editor and many different development tools useful during development, such as realtime profilers, APK analyzer, etc. all included on a unique Integrated Development Environment (IDE) (figure 3.11).

The development in Android Studio is based on *Activities*, working as windows in a desktop system. Each activity has it own layout, to determine how elements are organized on the screen. This elements can be *containers* to host more elements, or *widgets* which are images, text, scroll list, etc.

[30] https://play.google.com/store/apps/details?id=com.mojang.minecraftpe
[31] https://www.netflix.com/es/
[32] https://www.spotify.com/es/
[33] Android Studio Installation Page: https://developer.android.com/studio

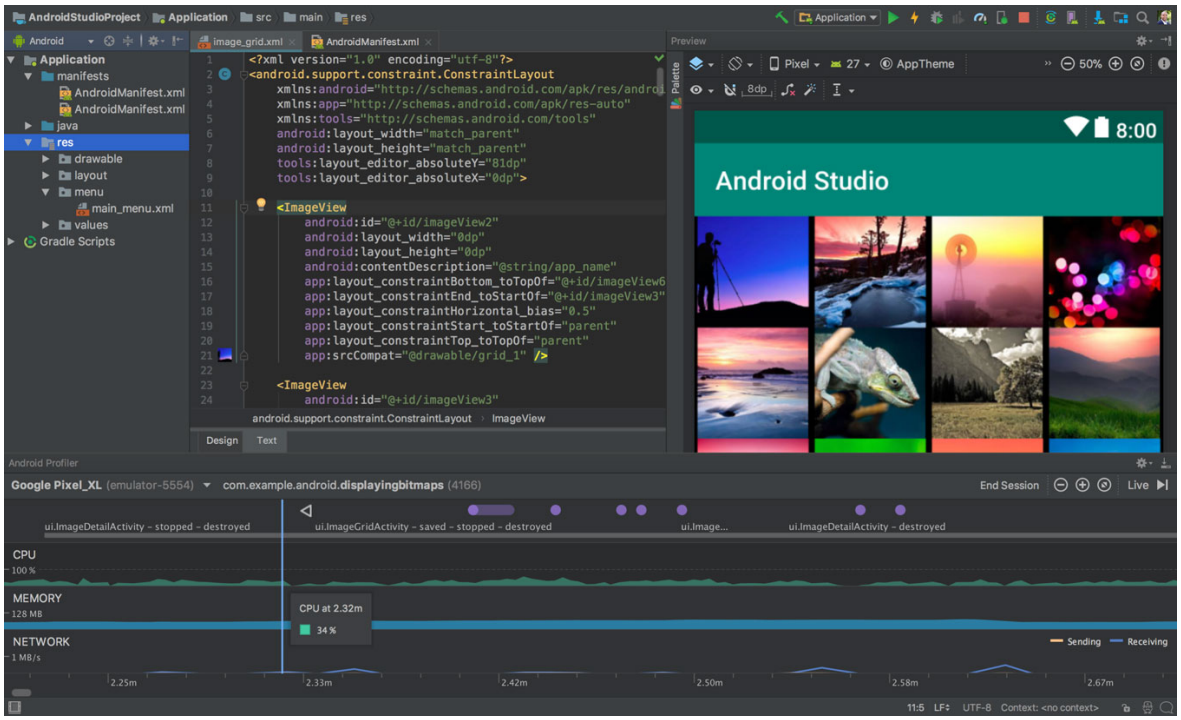


Figure 3.11: Android Studio IDE for app development

Android Studio is a very powerful tool that allow to create from basic apps for beginners, to really complex and professional systems.

Xamarin

Xamarin[34] is a tool that allows developers to create apps for iOS, Andorid and Windows Phone using C#. Basically, this tool translate the code written in C# so it can be executed in mobile devices.

To develope an app for iOS it must be written using Objetive-C, and for Android in Java, so Xamarin unify all this diferences in an IDE, called *Xamarin Studio*. It contains an editor to write code in C#, and then it can be automatically translated and deployed to any mobile device.

Although *Xamarin* is a very powerful tool, it was not free to use originally. However, as Microsoft bought the company in 2016, it can be downloaded to be used directly in Microsoft Visual Studio[35] while keeping all its original functionalities. It is one of the most used cross-platform development tools on the market.

[34] https://docs.microsoft.com/es-es/xamarin/
[35] https://visualstudio.microsoft.com/es/

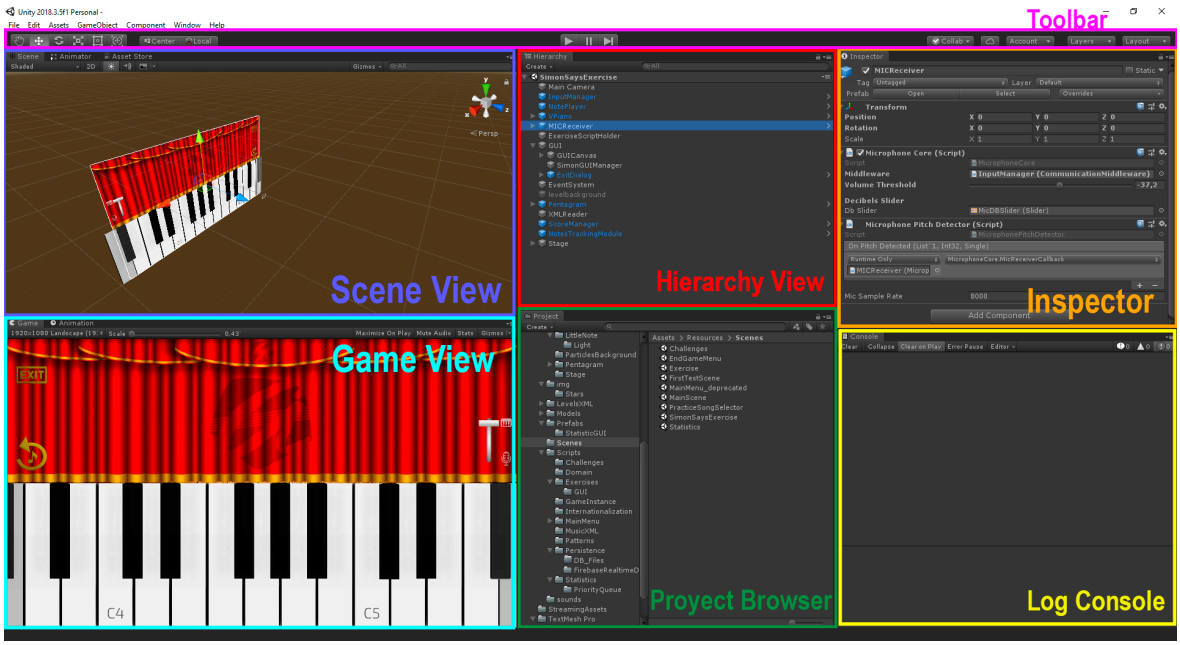


Figure 3.12: Unity Editor customized windows layout

Unity

Unity[36] is a cross-platform game engine developed by Unity Technologies. It was firstly created as an Mac OS exclusive game engine but it evolves until supporting more than 25 target platforms in 2019. It support development in 2D, 3D, Virtual reality (VR) and Augmented reality (AR) environments.

The development at Unity is based on two main elements: *Game Objects* and *Components*. A *Game Object* is the fundamental object in Unity that acts as a container for Components [21]. It represents characters, props and scenery. The *Components* are the functional pieces of every Game Object [21]. Every Game Object must contain a *Transform Component* in order to be part of a Scene. Then, many other components can be add to that Game Object to get different behaviours. Some Components examples are: Rigidbody (for physics), Audio Source (to play sounds), Canvas (where UI object are rendered), Box Collider, developer's custom scripts, etc.

The *Unity Editor* is the tool used to develop new games with Unity Engine. It is divided into different windows (see Figure 3.12). The main windows are [21]:

- *Project Window:* this view allows to manage the assets that belong to the project. An asset is any media or data that can be used in the game.
- *Scene View:* this is the interactive view used to create the world selecting and locating the characters, props and all other types of Game Objects.
- *Hierarchy Window:* this window contains a list of every Game Object in the Scene organized in a hierarchical structure. This structure basically is that every child of an

[36] https://unity.com/es

object inherits the transform component from the father.

- *Inspector:* this is the window that displays detailed information about a selected Game Object. It also allows to add or remove new components and change it properties.
- *Toolbar:* here is where the development controls are located. From here the developing game can be executed and stopped among other functionalities.
- *Game View:* this view is rendered from the camera of the game. It represents the actual game.

Unity comes with lots of different components to set different behaviours to Game Objects. However, although these components can be customized by modifying their properties in the inspector, they usually are not enough to get the desired behaviour. Here is where scripting plays a critical role. Scripts are treated in Unity as object components, so that, a script is attached to a Game Object, conferring the specified behavior to that object. These can be written in C# or JavaScript. Every script written for Unity development must inherit from *MonoBehaviuor*. This class defines the basic operation of a game object to exist in the world.

Even Unity is focused on game development, it can also be used to create non-game apps, however, this non-game apps include Unity's core features, which may not be worthy for some applications. This type of development is usually used for apps that include architectural walkthroughs, instructional interactive demonstrations, training simulations and product visualization.

CHAPTER 4

METHODOLOGY

Every engineering project needs a methodology to be able to approach, in a complete and efficient way, the objectives that were proposed at the beginning of the project. Therefore, the following chapter will detail the methodology used this project development.

4.1. WORK METHODOLOGY

Allegra is a musical learning application, so it requires advice from a professional of the sector. Meetings with this person, as well as with the project tutor, have been held on a weekly frequency. In these meetings, the requirements that the application had to meet were defined, which, due to the nature of the application, changed week after week.

When a software development is faced with rapidly changing requirements, specially when it is done by one one person, it is convenient to be prepared to prevent these changes from having an impact on this process. A widely used approach in professional development are the agile methodologies. The agile methodologies are oriented to small projects and provide a high simplification but maintaining the essential practices to ensure product quality.

Iterative and incremental development is a type of agile methodology that is based on a series of distributed iterations during the time the development lasts. In figure 4.1 this methodology phases are shown. First an initial planning is agreed with the client, in order to define the scope of each of the iterations and then start with the iterations. Each iteration has its phases of analysis, design, implementation and testing until the evaluation of the result obtained. Each of these iterations usually lasts a short period of time and results in a piece of software that can be seen as a mini-project, and that adds functionality and complexity to the project, until it constitutes the final result of the development. When one of these iterations is completed, the results obtained must be analyzed in such a way as to guarantee that the iteration has fulfilled the established objectives and thus be able to proceed to the planning of the next one.

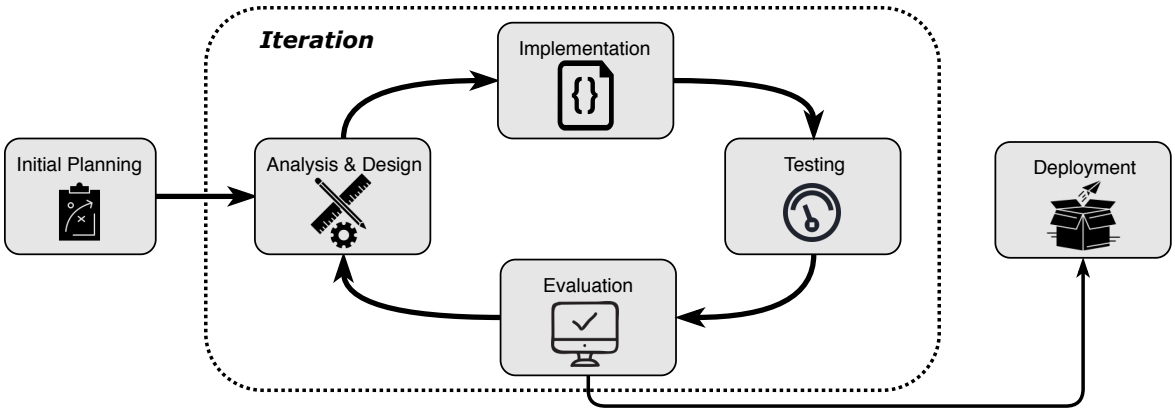


Figure 4.1: Iterative and incremental methodology scheme

There are numerous advantages to using this methodology and it has consequently been employed in the project described in this document. An iterative development allows the whole process to be reviewed periodically, improving and correcting errors, thus giving the system a correct performance. On the other hand, as it is an incremental methodology and the work is divided into milestones that add functionality as the development progresses, these functionalities can be separated into different modules, each one totally independent from the others, providing the system with a scalable architecture.

4.2. RESOURCES

In order to be able to face a development of this style certain resources are required, both hardware and software, which will be discussed in the following section in a very brief and concise way.

4.2.1. Hardware resources

As the development is an application aimed to mobile devices, there are only two important hardware resources:

- *MSI Ge63 7RD Raider Computer*[1]: It is one of the most powerful gaming laptops on the market. It is ideal for this type of development as it has 4GB of dedicated graphics card Nvidia, which allows optimal fluidity when rendering and working with graphic tools.
- *BQ Aquaris X5 Plus*: This mid-range smartphone has been used for the testing phase. It has been chosen not to use one with excessive power to ensure that the application can be run on most devices on the market.

[1] https://es.msi.com/Laptop/GE63-7RD-Raider.html

4.2.2. Software resources

Operating System

- *Windows 10 Pro*. This is the OS used for the whole development process since all the development tools were compatible with it and also the Android package and deployment.
- *Mac OS Mojave*. This OS will be eventually used to package and deploy the app in iOS devices as it can only be done using Apple devices.
- *Android*: this is the target operating system of the application in which tests have been made at the end of each iteration to ensure its operation in a real environment.

Development tools

- *Unity*[2]. This game engine, thanks to its powerful support for the development of multimedia applications, has been the ideal technology to carry out this development. In addition, thanks to all the community that it has, there are many tools and forums that are of great help during the development process.
- *Visual Studio 2019*[3]. This is the editor used as it is a very complete IDE and recommended by Unity which makes debugging and profiling simple and powerful processes.
- *Git Kraken*[4]. This is a tool used for version control management. This git client takes advantage of all the functionalities offered by Git technology thanks to its powerful graphical interface that facilitates complex operations in just a few clicks.
- *Xcode*. This is an integrated development environment for macOS that contains a set of tools created by Apple for the development of software for macOS, iOS, watchOS, and tvOS. It has been used for deployment on iOS devices.
- *Adobe Illustrator*[5]. It is a vector image design tool widely used in industry. It has been very useful when designing and creating 2D visual assets.
- *Piskel*[6]. It is a desktop or online application specially designed to generate a sprite sheet with animations quickly and easily. Useful to create 2D animated assets.

Libraries and SDKs

- *RAPT Pitch Detector*[7]. This package from the Unity asset store offers the possibility to recognize the pitch of a sound recorded by the microphone using the RAPT (Robust

[2] https://unity.com/es
[3] https://visualstudio.microsoft.com/es/vs/
[4] https://www.gitkraken.com/
[5] https://www.adobe.com/es/products/illustrator.html
[6] https://www.piskelapp.com/
[7] https://assetstore.unity.com/packages/tools/audio/human-voice-pitch-detector-109019

Algorithm for Pitch Tracking) algorithm.

- *Firebase SDK*[8]. This is a platform powered by Google that provides, among other tools, a database in JSON format very useful for storing the status of the player. The Unity SDK is already provided by the platform.
- *Android SDK*. Android is the most popular OS for mobile devices. In order to build and deploy an app targeting this OS, the system must made use of some specific libraries, all of them can be found in the official Android SDK.
- *Obri Landing Page Template*[9]. In order to facilitate the development of the visual part of the website and thus put all efforts into the positioning part, it has been decided to acquire a template called Obri, specially designed for mobile apps.

Programming languages

- *C#*: it is a object oriented language developed by Microsoft. It is the scripting language supported by Unity.
- *MusicXML*: this open music notation format based on XML was designed for the exchange of scores, particularly between different score editors, and it is used to store the exercises scores.
- *JSON*: is a simple text format for data exchange and it is used by Firabase Realtime Database to store the player information.
- *HTML/CSS*: HTML is a markup language that is used for the development of Internet pages and CSS is a graphic design language for defining and creating the presentation of a structured document written in a markup language such as HTML. Has been used in the landing web page.

Documentation

- *LaTeX*: is a text composition system, oriented to the creation of written documents that present a high typographic quality and especially for scientific texts. The tool used to produce this document has been *Overleaf*[10], an online platform that simplifies the work as it integrates all the necessary libraries and spelling correction.
- *Draw.io*[11]: is an online tool for making diagrams and schemes that also allows for cloud storage and collaborative work. It has been widely used to layout all the diagrams in this document.
- *Photoshop*[12]: it is a image editing tool that has been used to edit, format and highlight the images in this document.

[8] https://firebase.google.com/
[9] https://themeforest.net/item/obri-app-landing-page-html-template/23382357?s_rank=2
[10] www.overleaf.com/
[11] www.draw.io
[12] www.adobe.com/Photoshop

CHAPTER 5

ARCHITECTURE

This chapter discusses, from a technical point of view, the architecture proposed to deploy *Allegra*. In the first place, there will be a general discussion of the distribution in layers and the scalable approach that has been followed, to then go deeper into each of these layers analyzing the problems that each one of them addresses, the solution that has been chosen and the advantages that this entails.

5.1. OVERVIEW

In order to obtain a scalable system, easy to understand and in which to modify and add functionality without affecting the rest of the elements already implemented, a layered architecture has been chosen.

In a system like this, in which the requirements are changing and the functionalities grow with each iteration, due to the methodology followed in the development, it is necessary to take a scalable design. A scalable system should be understood as a system ability to grow, add functionality and adapt to changes without affecting the rest of the system. In this way, as the development progresses, new functionalities and features can be incorporated or modified without this having an impact on the rest of the implementation.

The system is divided into 3 clearly differentiated layers, each with its own objectives and functionalities (see Figure 5.1). The layer with the greatest responsibility is the *Gamification and GUI layer*, since it is the one that will be in charge of managing the behaviour of the exercises and how they are represented, as well as the different menus that the user can navigate through. The *Communication layer* is the one that allows the user to communicate with the application by means of the so-called receptors. Finally, the *Persistence layer* is the one that will allow storing and processing user information and levels. Each one of these layers, is formed in turn of multiple modules, where each one of them encapsulates a different functionality that adds value to the layer to which it belongs. The communication between the different layers is done through a system of events, since the layers do not

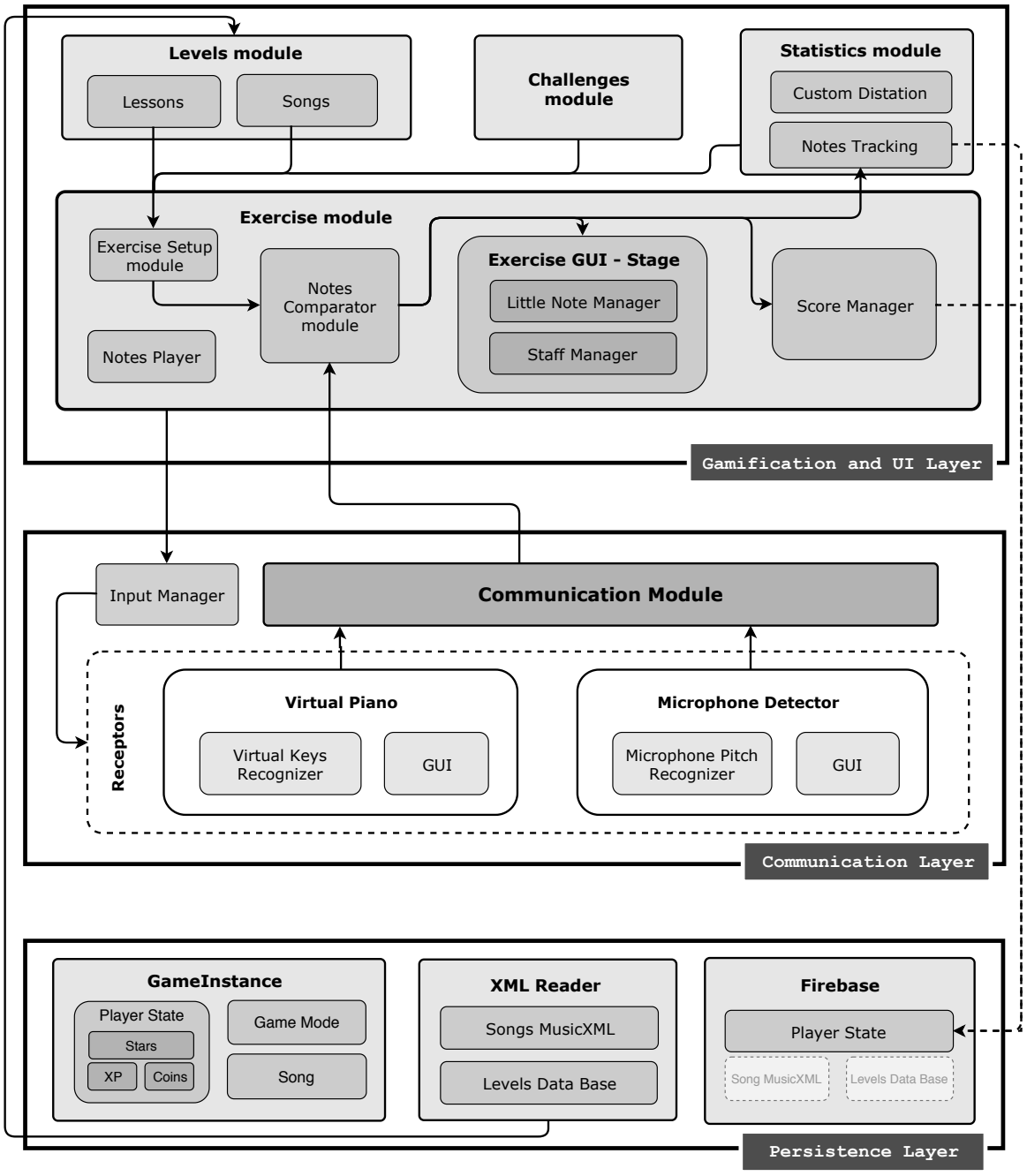


Figure 5.1: Allegra's architecture - General scheme

know the way to work of the others, which allows to isolate even more of the impact in the modifications of some of them.

In the following sections we proceed to detail the operation and objective of each of the layers and modules.

5.2. GAMIFICATION AND GUI LAYER

This is the layer with the greatest number of responsibilities of the whole system. It will take care of all the logic of the application, from the behavior of the levels, to the calculation of statistics and visual representation of the menus. It also manages gamification elements such as challenges.

5.2.1. Exercise module

In an application whose objective is the practice and improvement of melodic dictation, it is critical to have some module in charge of managing the different exercises. This management refers to defining the rules of the game, its graphic representation, as well as comparing the notes produced by the user and returning feedback in response. This will be the foundation of the whole system.

An *Exercise module* is defined by 5 main elements: the exercise setup module, the notes comparator, the notes player, a graphical interface of the exercise state and the score managers. These elements are shown in figure 5.2. The first four must be defined in order to implement new exercises. The last module is the score manager, which is optional, because if there is no score manager, the exercise will work correctly except that the results will not be stored persistently, as is explained in the *Score Manager* section below.

This approach is based on the definition of an abstract class, called *Exercise* that defines the basic elements of this module. Thus, when the developer wants to create a new type of exercise, he must create a class that inherits from *Exercise* class. In this new class the behaviour of the different modules is coded, defining here the game rules.

Moreover, an *Exercise* invokes three different events (see figure 5.2). These events are of utmost importance, as it is the means by which an exercise communicates with the rest of the modules. There events are the following:

- *NoteChecked Event*: it will be called when the exercise checks if a note is correct or not, and returns some feedback data about the result (whether it is correct or not, the note provided by the user, the feedback time, etc). This can be used by the GUI and the communication layer to return visual feedback to the user.

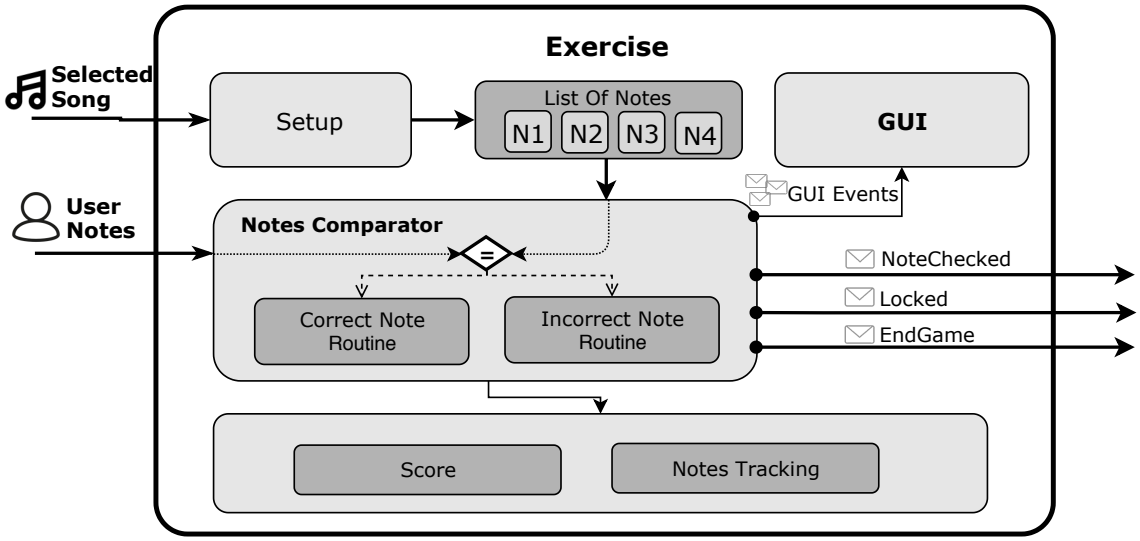


Figure 5.2: Exercise detailed flow scheme

- *Locked Event*: this event is invoked when the exercise wants to block or unblock the sending of more notes. This is useful, for example, if the exercise is giving some feedback, or playing the notes, it makes no sense for the user to be able to enter notes.
- *EndGame Event*: this event can only be invoked once, since its function is to notify when a exercise has reached its end, performing then the corresponding operations.

Exercise Setup

At the beginning of an exercise, this module gets the information from the selected song in the *GameInstance* (see section 5.4.1). The notes of this song must be loaded into the list of notes of the exercise. This list should not be modified during game play as it contains the solution of the exercise. It works as a read-only buffer, used to load the notes in order, to be asked to the user, working as an input to the *Comparator* module, detailed below. Once this loading process has been completed, this module notifies the exercise to load the first set of notes and the game starts.

Comparator

This is the module where the rules of the game are defined. The *Comparator* receives two different inputs. A set of notes that are asked to the user and a set of notes provided by the user. This leads to two different situation:

- Both notes sets are *equals*, which means that the user gave the correct answer.
- Both notes sets are *different*, which corresponds to the fact that the user got it wrong when introducing the notes.

Once this comparison is resolved, the *Comparator* module executes different pieces of code according to the comparison result. These are the code fragments in which the rules and procedures of the exercise must be defined, such as, for example, restart everything when it turns out to be incorrect or maybe just to give a hint, or instead, if it is correct, the next note or the whole set can be asked again, etc. This can be done to the will of the developer.

Finally, after comparing the notes and perform the corresponding actions, the *Comparator* always fires the event called *NoteCheckEvent*.

Exercise GUI

The exercise, so far, has simply been dealt with its part of logic and algorithmic, however, to ensure a user-friendly use of the application, a complete and attractive graphical representation must be taken into account.

In other words, every exercise must know who is the entity in charge of managing its graphic part. As this can have a certain complexity, it has been separated into a different script apart from the logic. In this way, the exercise simply notifies the state of the game, so the graphical interface can perform the necessary operations to match the possible changes.

The communication mechanism used is based on specific events so that the exercise can notify its graphical interface. These events depends on the implementation of the exercise. There must be an event for each type of notification. For example, in the case of study currently implemented in *Allegra*, there is a notification when a note is played by the system, when the melody is played as well, and when a note is checked. There may be repeated events between those of the graphical interface and those of the exercise, but this approach has been chosen because it is possible to differentiate between communication with the graphical interface and communication with other modules and entities of the system.

Notes Player

In a music application it is vital to be able to play notes independently and with full control.

A possible solution would be choosing to have a sound file for each of the musical notes to be used, and play it when necessary. This, however, is not scalable and can considerably increase the size of the final packaging of the application.

Another possible solution would be to have a script in charge of, from an audio file of a specific note, calculating how much to modify the tone of that note in order to reproduce the one you are interested in. This is what *Notes Player* module does. This approach also solves the inconveniences mentioned before.

Listing 5.1: Pitch calculation in Notes Player

```
public class NotesPlayer : MonoBehaviour
{
  [...]
  public void Play(int NoteMidi)
  {
    float offset = (float)(NoteMidi - ↩
      ↪ ReferenceNoteMidiValue);
    GetComponent<AudioSource>().pitch = Mathf.Pow(2.0f, ↩
      ↪ offset / 12.0f);
    GetComponent<AudioSource>().Play();
  }
  [..]
}
```

The *Notes Player* parts from the note *A4*, the standard reference note (see 3.1.1). When an exercise, or another module, needs to play a note, it calls to a function in the *Notes Player* which receives as argument the midi representation of the note. With this value, which is an integer, the pitch offset, respecting the A4 note, is calculated as shown in listing 5.1 using the formula 5.1.

$$n = n_r * 2^{s/12} \tag{5.1}$$

Score Manager

In a game, it is important to know at all times the score of the player, especially at the end of a level, so that the player knows how well he has done in the execution of the exercise. If this information also has an impact on the overall state of the player, such as points, experience, unlocked elements, etc., something common in this type of application, it should be stored in the persistence file or database, or modify and update the information already stored.

The *Score Manager* is the module to which this responsibility has been assigned. Its mission is, from certain level parameters, in this case the number of correct and incorrect notes, to calculate the final score of the player. This information consists of 3 parameters:

- *Stars*: It can be gotten a maximum of three, and the number of stars obtained is according the number of mistakes, as reflected in table 5.1.
- *Coins*: For each star achieved in the level a fixed amount of coins is obtained, so the more stars achieved the greater the number of coins.
- *XP points*: which are equivalent to the experience gained by the player. The total experience earned is calculated based on the number of notes played by the user, regardless of whether they are correct or not, as a mistake also brings experience to the player.

Table 5.1: Relationship between the obtained stars and the number of mistakes

Stars	Number of Mistakes
3	0
2	1 - 2
1	3+

Besides, as the application implements an statistics module, it is interesting to store the notes played by the user and whether they are correct or not (see section 5.2.3).

Once the exercise is finished, this information, apart from being shown to the user, is sent to the module that will be in charge of storing it in a persistent way, in this case this module will be the *Firebase Realtime Database* (see section 5.4.3).

5.2.2. Levels

In any mobile application similar to *Allegra*, over time it is possible to introduce different game modes or to have different databases from which to read songs. This leads to several level selection screens. For example, in the case of Allegra, one for lessons and one for popular songs. In fact, one way to deal with this problem is to actually create a different screen for each game mode or database that requires one. However, this approach is not scalable and is not recommended.

In *Allegra*, a specific module has been created to carry out these level selection operations, with different game modes and databases, in a scalable way. Starting from a single selection screen, and a template representing a level on that screen, you can load everything. This is because this module receives at the beginning the reference to the database from which to read the songs and the corresponding game mode. In this way, the desired songs will be loaded and the levels, starting from a template, the parameters can be loaded dynamically.

Each level is represented as a big button which contains a *Song Level* object (see section 5.2.5) and displays all the information related to that object. This information is the name, the difficulty and the required amount of resources. This resources can actually change from a game mode to another as, for example, in lessons XP points are required, while in popular songs, it is coins.

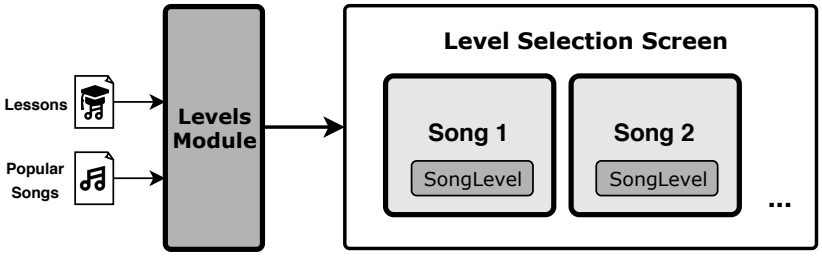


Figure 5.3: A unique level selection screen for everything

5.2.3. Statistics

When we talk about an application, especially if its objective is educational, it is of special interest to collect certain information during the use of it, in order to produce a report on the progress and improvement of the player. This means, a statistics system can be useful to improve the efficiency of the application.

The *Statistics Module* is responsible for obtaining the necessary information and producing a simplified report, within the application itself, so that the user can check it at any time. In addition, this information can be used for other purposes, such as creating custom exercises for each user.

Notes Tracking

A sub-module called *Notes Tracking* is responsible for collecting information from the notes during the exercise. Each time the comparator resolves whether the note entered by the user is correct, this module collects that verdict and stores it, for later, once the exercise has finished, it is stored in the database. In this way, when the user accesses the statistics report view, simply with a quick query to the database, the information can be displayed.

Custom Exercises

Another function of the statistics module is the creation of exercises from the data collected, thanks to the sub-module called *Custom Exercise*. This, based on the statistics report, according to certain parameters given by the user, such as, for example, the number of notes that make up the exercise, will create a completely new exercise.

As the exercise entry is a list of notes from a *SongLevel* object (5.2.5), thanks to the scalability of the system, instead of reading it from a file, this module will generate this list, host it in the *Game Instance* (see 5.4.1) and start an exercise, which will load these notes.

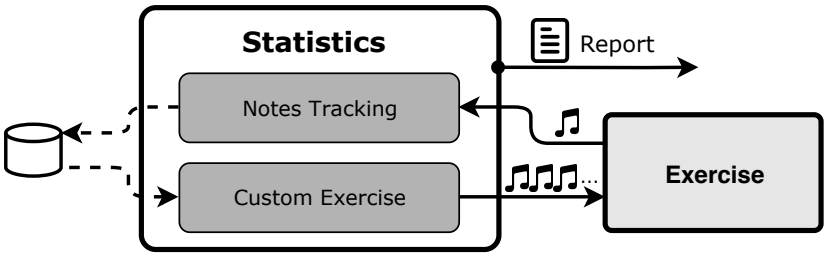


Figure 5.4: Simplified Statistics flow scheme

5.2.4. Challenges

Challenges are an important feature in applications that use gamification techniques to engage users. To give Allegra this feature, a specific module called *Challenges* has been developed.

A challenge, as defined by this module, is an exercise that have a specific reward and that can be played from time to time. Each challenge has a time stamp that specifies when it was last played, so that comparing it with the current time you can know whether the challenge is unlocked or not. If it is unlocked, it can be played by the user. This launches an exercise and overwrites the value of the time stamp in the database, so that the user has to wait the indicated time to play again. However if it is locked it cannot be played and a counter will appear showing the remaining time for unlocking.

Currently there are two types of challenges implemented: daily and weekly, so that they can be played every 24 hours and 7 days, respectively.

The Challenges module has an associated database with songs, among which are taken for each of the challenges. Thus, when the user begins to play a challenge, this module generates a *Song Level* (5.2.5) and launches the corresponding exercise that will take this object as a starting point.

Figure 5.5: Challenges simplified scheme

5.2.5. Song level and exercise note

Allegra is an application divided into well-differentiated modules, each with its own independent functionality. However, all of them deal with the same concepts: songs, levels, notes, etc. In order to avoid that each one of these modules uses these data in its own way and making impossible a communication between them, two objects have been defined that will set the foundations of all the architecture: the *Song Level* and the *Exercise Note*. In this way, every module knows and can make use of these objects, making it very easy to work with them with different modules at the same time.

- A *Song Level* is an object that represents a song inside the system. It contains all the relevant information about a song: the title, path, rewards, the stars obtained and a list

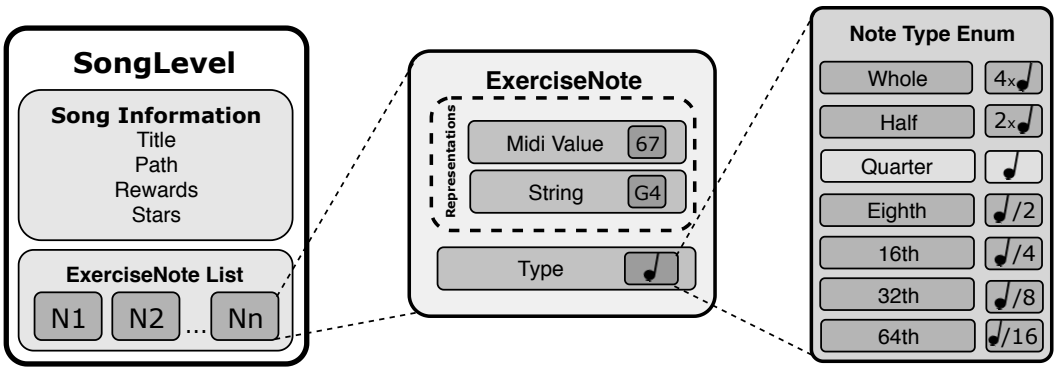


Figure 5.6: Domain entities and how they are related

of *Exercise Note* (see figure 5.6). This object can be created by any module manually or read by a persistence file, and it is mainly used as input for an exercise.

- The *Exercise Note* is a universal container that stores all the information about a music note inside the system. A note can have many representations in the application, so this object encapsulate all of them (see figure 5.6). These representations are the midi value, which is an integer assigned uniquely to each note, and the string representation, which is the classic way of music notation (C4, D5...). This object also stores the type of note, which specified its duration, as explained in 3.1.1, taking into account the *Quarter Note Reference Time* stored in the *Game Instance* (see section 5.4.1).

The *Note Type* enumeration allows to represent the type of the note. The system supports 7 different types of notes, from the *whole* note (four time the quarter note) to the *64th* note, which is the value of the quarter note time divided by 16, as is shown in figure 5.6.

5.3. COMMUNICATION LAYER

The communication layer is in charge of collecting the information produced by the user, thus allowing the interaction between the system and the user. It basically consists of two important parts: receptors and the communication module.

5.3.1. Receptors

An application related to music, in which communication is required between the user and the application, mechanisms are needed to enable this action. These can be emulations of instruments within the app to simulate a real one, or mechanisms that collect the information from a real world instrument and convert it so that the information is understood by the system. This need resides in the fact that, since the whole application is in a musical context,

the input mechanisms must follow this approach. To this end, the use of so-called *receptors* is proposed.

A *receptor* is the mechanism in charge of collecting the notes provided by the user, in other words, it is the means by which the user can interact with the application during an exercise. *Allegra* supports many different types of receptors. Every receptor implemented on the app must inherit from the abstract class called *Receivable*. This class is in charge of the most general functionalities of a receptor. This class also inherits from *Monobehaviour* as in Unity every game object must inherit from that class (see section 3.12).

Listing 5.2: Code from Receivable abstract class

```
abstract public class Receivable : MonoBehaviour
{
    // Reference to communication module
    public CommunicationModule module;
    // If true then the receiver is active
    protected bool isActive = false;
    // If true then the receiver cannot be used by the user
    protected bool isLocked = false;

    // Called when the receiver method is activated
    public abstract void OnActivated();
    // Called when the receiver method is disactivated
    public abstract void OnDeactivated();
    // Event Handler when a note is received
    public abstract void OnNoteChecked(int midi, string
        ↪ NoteString, bool isCorrect, float FeedbackTime);
    // Event Handler when the receiver method is locked or unlocked
    public virtual void OnLocked(bool locked, float Delay)
    {
        isLocked = locked;
    }
}
```

As is shown in the code on listing 5.2, a receiver must know whether it is active or locked and who is the module to which the information is send to (explained in 5.3.2).

The variable *isActive* is used to determine whether a receiver is being used by the user or not. Of all the receivers only one of them can have this variable to true, as only one receiver can be active at a time. The *input manager*, is the module responsible of switching between receivers. When a receiver is activated, this module subscribe the event handlers *OnNoteChecked* and *OnLocked* to the events with the same name on the exercise (see 5.2.1) and the function *OnActivated* is called to perform the corresponding actions. However, if the receiver is deactivated the function *OnDeactivated* is executed and the event handlers are unsubscribe from the exercise, so that, only the handlers of the active receiver are called.

While the receiver is active, it can be used by the user except in some specific moments, such as, while a melody is being played. For those cases, there is a variable called *isLocked*

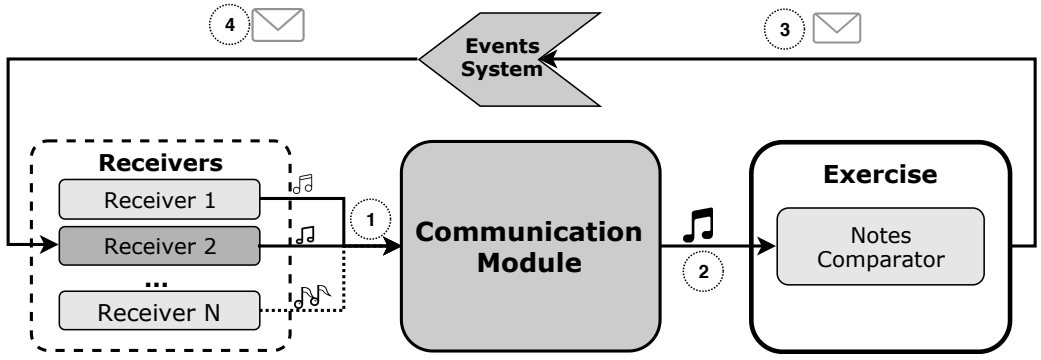


Figure 5.7: Receivers communication with exercise

used to check whether the receiver must send information or not, depending on the game state. To change it value, an exercise (see section 5.2.1) must invoke the *OnLocked* event, passing the locking information. This information is a boolean variable (true for locking and false for unlocking) and a delay to perform the action.

The *OnNoteChecked* function will be called after the exercise check the note provided by the user using this receiver. It will provide the midi and string representations of the note, a boolean variable saying if the note is correct or not and the time that the exercise will be giving feedback. With this information, the receiver is also capable of create and give its own feedback to the user.

As mentioned above, a receptor is the mean by which the user introduces notes to the application, so an important task is sending notes to the communication module. For that, it is as simple as calling the specific function for each receiver in the communication module.

The diagram in Figure 5.7 illustrates the process of communication between receptors and the exercise. In this way, a set of receptors, of which only one will be active for use, sends the note information (1) to the communication module. This module processes the note and forwards it to the exercise comparator. Here it is evaluated and compared according to the rules established for later use of the event system (3) that sends the information from the comparison result to the active receiver (4).

To summarize, there is a *Receivable* class that defines the basic behavior of every receptor, and a set of event handlers that allow the communication of other modules with the receptors. This approach has the great advantage of making this part of the system highly scalable. When the developer implements a new receptor module by introducing an input method, he simply has to inherit from that class, register the input method in the *Input Manager*, and place all the necessary logic in the resulting class, completely transparent to the rest of the system.

Currently, in *Allegra*, there are 2 different receptors implemented: a Virtual Piano and a Microphone Pitch Detector.

Virtual Piano

The virtual piano is one of the receptors implemented in *Allegra*. It consists in a set of virtual keys displayed on the button part of the screen during game play. Each key has associated a note so, when the key is pressed, that note is sent to the *PlayPressedNote* function in the core script of this module as shown in listing 5.3. This is the function in charge of sending the note to the communication module (see section5.3.2) and also uses the *NotePlayer* (see section 5.2.1) module to reproduce the note simulating a real piano.

Once the exercise has checked whether the note is correct or incorrect, the event handler *OnNoteChecked* is invoked, so the receiver can give some feedback to the user. In this case, the virtual piano colors the pressed key in green if the note is correct, or in red if it is incorrect.

Listing 5.3: Core code from VPianoCore Script

```
public class VPianoCore : Receivable
{
    [...]
    // Called by the pressed key
    public void PlayPressedNote(string PressedNote)
    {
        if (!isLocked && !isKeyPressed)
        {
            notePlayer.Play(PressedNote);
            module.SendVPiano(PressedNote);
        }
    }
    // Called when the note is checked by NoteCheckEvent
    public override void OnNoteChecked(int midi, string
        ↪ NoteString, bool isCorrect, float FeedbackTime)
    {
        [...]
        if (key != null)
            ColorKey(key, isCorrect, FeedbackTime);
    }
}
```

Microphone Pitch Detector

A receiver method implemented in *Allegra* is a Microphone Pitch Detector, based on the RAPT[1] Pitch Detector algorithm. This algorithm takes as reference a *fundamental frequency* that is the quantity that is being estimated by all virtual pitch trackers and is an inherent property of periodic signals and tens to correlate well with perceived pitch. This means, this *fundamental frequency* is compared with the frequency obtained from the input signal using a cross-correlation function. It is a complex but fast process detailed in [19].

[1] A Robust Algorithm for Pitch Tracking

Listing 5.4: Core code from MicrophoneCore Script

```
public class MicrophoneCore : Receivable
{
    // Volumen Threshold
    [Range(-100.0f, 12.0f)]
    public float VolumeThreshold = -40.0f;
    // Detector reference
    private MicrophonePitchDetector detector;
    //Stores midi data from mic until it is sent
    private List<int> MidiBuffer;
    [...]

    // Called by the detector
    public void MicReceiverCallback(List<float> ↩
        ↪ pitchList, int samples, float db)
    {
        [...]
        // When volume is higher than threshold it start recording
        if (pitchList.Count > 0 && db > VolumeThreshold ↩
            ↪ && pitchList[0] > 0)
            MidiBuffer.AddRange(pitchList);
        // Data in the buffer is sent as the volume is too low to get more data
        if(MidiBuffer.Count != 0 && db < VolumeThreshold)
            SendDataToCommunicationModule();
    }

    // Called when data must be sent to the module
    void SendDataToCommunicationModule()
    {
        List<int> frequentValue = ↩
            ↪ GetDominantValues(MidiBuffer);
        for(int i = 0; i < frequentValue.Count; i++)
        {
            if (frequentValue[i] >= minValue && ↩
                ↪ frequentValue[i]<= maxValue)
            {
                module.SendMic(frequentValue[i]);
                break;
            }
        }
    }
    [...]
}
```

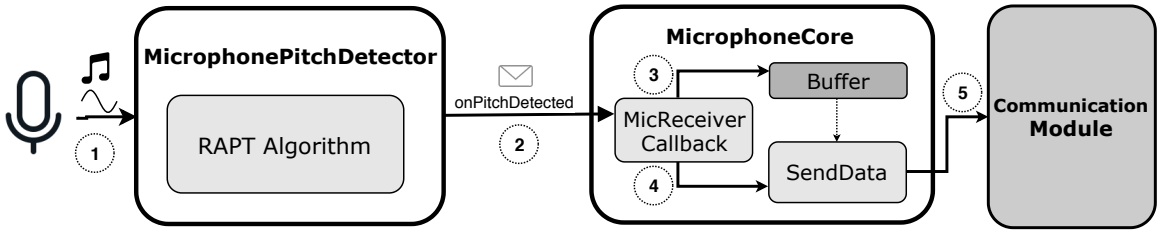


Figure 5.8: Simplified conceptual scheme of Microphone Detection Module

In listing 5.4 is shown the core code of this module. The attribute called *detector* holds a reference to the *MicrohonePitchDetector* instance, which is the script that actually implements the RAPT algorithm.

This module, as shown in Figure 5.8, when the microphone detects any sound (1), forwards the collected data to *MicReceiverCallback* function (2), that is a handler previously subscribe to the *onPitchDetected* event invoked by the detector. This function stores the pitch data in a buffer called *MidiBuffer* (3) while the volume of the input is higher than the *VolumenThreshold*. This threshold is the minimum value from which the microphone starts to record. If the volume of the sound detected is lower than this value, the data is ignored. This is also used to detect when a note is finished to be played, as when the data is being stored in the buffer, if the volumen decreases from this threshold, then it is interpreted as the note is being finished to be played (4), forwarding then the data to the communication module (5). This last task is performed in *SendDataToCommunicationModule*, where the dominant pitch is found and forwarded to the communication module.

5.3.2. Communication module

The *Communication Module* is the intermediate module between the receivers on the Communication layer and the Gamification and UI layer. It purpose is to receive the information provided by the active receiver and forward it to the exercise *comparator* (see 5.2.1). Thanks to this approach, greater isolation is achieved between the different receptors and the exercise module. The receptors only have to know whi is the communication module, ignoring the game mode of the exercise, and at the same time, the exercise works the perfectly independently of the receptor that is being used in each moment, being the communication module the intermediary that facilitates the communication process.

In listing 5.5 is the core code for this Communication Module. It inherits from *MonoBehaviour* as it is a script attached to a Unity game object (see section 3.12). It must know the *Exercise* that is being played, and the range of effective notes. The function *SendToComparator* is intended be called by receivers to send the note information to the exercise if it is in the effective range. It is the core function of this module. It receives as parameter the note midi representation, if the note representation is not midi, it must be translated to it.

Listing 5.5: Core code of the Communication Module

```
public class CommunicationModule : MonoBehaviour
{
    // Reference to Exercise
    public Exercise exercise;
    // Notes interval where notes are parsed
    public string MinNote = "A3";
    public string MaxNote = "F5";
    [...]
    void SendToComparator(int note)
    {
        if(note >= MinMidi && note <= MaxMidi)
            exercise.Comparator(note);
    }
}
```

5.4. PERSISTENCE LAYER

In an application that deals with player progress, status, and levels, each with unique information, it is of great importance to keep these data in some persistent storage permanently to avoid losing them. For this, this section will see the different technologies and media that have been selected for this task. In addition, it will explain the use of having a container within the application itself to share information between the different modules.

5.4.1. Game instance

In a mobile application, it is common for the user to navigate between different windows and these windows may have to share certain information. An example of this is that the level selector selects a song from a list and the exercise must read it. For this type of communication, it is of special interest to have a persistent container during the whole execution, that means, that it is not destroyed when changing the screen, and where these information can be stored.

Game Instance is the name given to this module because it works as a container to share information persistently during game play. This object is instantiated at the beginning of the main map of the app but it must remain existing during the whole execution, so Unity deals with that as follows.

In Unity, objects exist in the world. The world is actually a Scene, so when the game changes from a Scene to another all the object in the old scene are deleted and all of the new scene are instantiated. In case the developer would like to maintain alive an object across different scenes, he must set the object to not destroy when a new scene loads. Unity offers a function called *DontDestroyOnLoad()*[2] which can be set to true, so that object will not be

[2]https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html

destroy. However, when the game changes from a scene to another and then came back to the main scene the *GameInstance* object will be created again, resulting two instances of the same object. This does not make sense in this context, because it is not necessary to have two persistent containers with different data across the game as it would lead to data inconsistency. To avoid this situation the *singleton* pattern has been used. The *singleton* pattern is explained in detail in section 5.5.

Listing 5.6: Most relevant code from GameInstance script

```
public class GameInstance : MonoBehaviour
{
    // Selected Song
    public SongLevel SelectedSong;
    // Current player state
    protected PlayerState playerState;
    // Selected game mode
    public GameMode gameMode;
    // Quarter note time duration
    public float QuarterNoteTime = 1.0f;
    [...]
}
```

At the beginning of the code in listing 5.6 there are four attributes. These are the data that can be stored in this container.

- *Selected Song:* this is a reference to a *SongLevel* object. It is set when a song is selected to be played, independently from the game mode. It contains all the information relative to that level (notes set, rewards, name of the song, etc).
- *Player State:* this object is load at the beginning of the execution. The information is retrieved from the cloud using *Firebase* module and any time it is modified by the app, it is synced with the cloud (see 5.4.3).
- *Game Mode:* a value from a enumerate where all the game modes are specified. At the moment there are two game modes: the practice mode and the popular song mode.
- *Quarter Note Time:* the reference of how long a quarter note is being played. Its value can be changed in a song staff if this duration is specified.

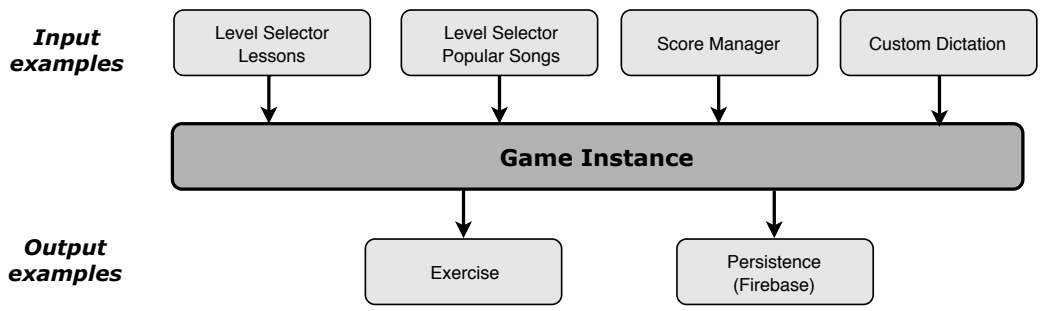


Figure 5.9: Example of some modules interacting with the Game Instance

5.4.2. XML reader

MusicXML is an XML-based file format used to represent Western music notation. As it is an XML file with specific labels (specified by the W3C[3]), it is read as a regular XML, so that it is needed some means or library to be able to perform this operation. *XMLFileReader* is the module implemented in *Allegra* and it is charge of the execution of this task, making use of the Microsoft library *XDocument*[4].

The *XMLFileReader* script consists of two parts. The first one is in charge of reading the notes in the *MusicXML* file. This part retrieved from the file the pitch and the duration of each of the notes, and creates a list of *ExerciseNote*. *ExerciseNote* is a class that holds all the information of each note, so any module of the app knows how to deal with this data, avoiding the access the file several extra times.

Listing 5.7: Example of a level specification in Lessons.xml file

```
<levels>
    <level>
        <title>Sol Mayor I</title>
        <path>04_-_SolMayorI</path>
        <difficulty>1</difficulty>
        <requiredXP>1750</requiredXP>
        <coinsForeachStar>20</coinsForeachStar>
        <xpForPlayedNote>400</xpForPlayedNote>
    </level>
</levels>
```

The second part of the script is in charge of reading from a list of songs or levels, obtaining the general information of each one. In listing 5.7 an example of this song specification is shown. Then this data is parsed to an object of type *SongLevel*, so that, any other module deals with an object that encapsulates all this information, including the *ExerciseNote* list, instead of dealing with the raw data from the XML file.

This script is invoked at the beginning of the execution to read the *SongLevel* list, including all the information except the notes. This operation takes as an input a file, that can be changed in execution time, allowing to have several levels database. This is useful, for example, to differentiate from lessons and popular songs. The notes are read at the begining of the exercise based in the selected song level.

5.4.3. Firebase realtime database

As the player moves up and down the level, his progress improves and has to be persistently stored to ensure that progress is not lost. There are lots of technologies developed for

[3] https://www.w3.org/2017/12/musicxml31/

[4] https://docs.microsoft.com/es-es/dotnet/api/system.xml.linq.xdocument?view=netframework-4.8

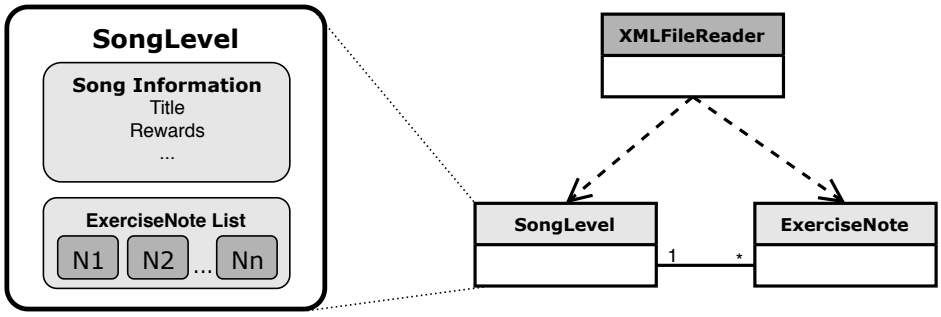


Figure 5.10: SongLevel specification and XML relation with other classes

this purpose, however, as the target platform is a mobile device a cloud storage can be compromised if the device lost connection, something really common for these devices. Other solution is a local storage approach, which does not depends on network connection, but in case the app is uninstalled or the user changes his device, the progress would get lost. Then, the perfect approach would be a mixed technology, with local storage for non-connected devices and cloud storage to assure that the progress is not lost. An example of these kind of technologies is *Firebase Realtime Database*.

Firebase[5] is a mobile and web development platform created in 2011 and adquired by Google in 2014. Currently it offers more than 15 different products aimed to ease the development for the cited platforms. One of those platform is *Firebase Realtime Database*[6], a NoSQL database hosted in the cloud. The data is stored in JSON format and synchronized in real time with each connected client, and remain available when the app has no connection [12].

These characteristics were determinant to take this technology as the one chosen to take care of the persistence of the state of the different users, in addition to the fact that there is a library adapted and ready to use for Unity.

The *Firebase Realtime Database* module is queried at the beginning of the execution of the application, obtaining a copy of the JSON tree stored on the cloud that is parsed in a container which is the *PlayerState* class. After reading the information, the *Firebase* module creates an instance of *PlayerState* with the current information. When this state is updated, for example when a reward is obtained, it is automatically synced with the cloud. If the device is not connected to the network, then the updates are stored locally in a cache file waiting to be connected again.

Every operation performed using the *Firebase* library is an Asynchronous operation, this means, is not executed in the main execution thread, but in another specially created for that operation. This can leads to concurrency problems so there must be a mechanism to control this. This mechanism is an events system. The system will not perform the operations

[5] https://firebase.google.com/
[6] https://firebase.google.com/products/realtime-database/

related to the database data until the event in invoked, so this assures that the execution order is the correct one.

5.5. DESIGN PATTERNS

A design pattern is a solution to a design problem. It is defined as a set of techniques for solving common problems in software development [6]. Thanks to a good selection of design patterns a simpler and more scalable architecture can be obtained. Therefore, in order to face certain difficulties, 3 patterns have been used in the development of *Allegra* .

5.5.1. Singleton pattern

Singleton is a pattern that represents a solution to two problems: maintaining only one instance of a class and having a global access point to that instance. However, many authors consider it an anti-pattern, as they advise the use of other alternatives when there is only one of the two problems [23].

It is for this reason that Allegra's architecture has not used a pure implementation of this pattern, but a variation that allows to solve the first problem: maintain a single instance of a class, in this case the *Game Instance*, previously defined in this section. In the implementation used, a static variable containing the instance of the class is maintained, but unlike the implementation recommended by the pattern, this variable is private and simply checks whether there is already an instance of that class or not. The reason for this approach is due to Unity's workflow.

Unity tries to create an instance of the Game Instance class every time the main menu is accessed, but it first checks if this instance already exists, in which case it removes the new one, as can be seen in the listing 5.8. This code shows how the *GameInstance* class contains a static reference to an instance of this class, so, when a new one is being created, its reference is stored in the variable called *_instance*, if there is already an instantiated object, it is destroyed. This is performed in the *Awake()* function, which is executed before the first frame of the scene.

5.5.2. Command pattern

The Command pattern can be understood as a substitute for a callback function. It consists of encapsulating a function within an object [6]. An example of implementation would be to generate a Command class with an abstract Execute function, which will later be used to encapsulate different functions, which will be launched when the function is executed[21].

Listing 5.8: Singleton implementation variation in Game Instance class

```
public class GameInstance : MonoBehaviour
{
    [...]
    // Singleton
    private static GameInstance _instance;

    void Awake()
    {
        if(_instance != null && _instance != this)
        {
            Destroy(this.gameObject);
        }
        else
        {
            _instance = this;
            [...]
            // Remains the object alive in all scenes
            DontDestroyOnLoad(this.gameObject);
        }
    }
}
```

The Unity event system is based on pattern. This system of events encapsulates within the *UnityEvent* object a set of references to functions that will be invoked when the *Invoke()* function is executed.

5.5.3. Observer pattern

The *Observer* pattern is a design pattern widely used in the software development industry that allows an entity to execute a piece of code to announce or notify events without worrying about who will receive the notification [23].

A classic implementation of this pattern specifies that two classes must be created, an *Observer* in charge of receiving the events, from which all those who want to receive those events must inherit, and another *Subject* that will be the base class in charge of sending the events. However, in Allegra, this pattern has been made use of through the system of events and delegates that Unity's technology offers the developer.

The exercise module sends the *CheckedNote* and *Locked* events in a transparent way to the entities who are managing them, being able to be any entity subscribed to these events. In the same way, the specific implementation of the exercise logic sends notifications to a GUI module through a set of delegates, specific to each game mode. Thanks to this approach it is possible to achieve a behavior as specified by the observer pattern through a simpler implementation and using the tools offered by the Unity technology.

CHAPTER 6

RESULTS

This chapter will address the final result of this project. First, it will discuss the prototype of the developed artifact, *Allegra* , and the levels and functionalities that have been implemented. It will also show the landing page of the application. Finally, a brief summary of the project cost will be made. Moreover, the whole project, including the source code and an Android *apk* file for testing, is available in the following link:

https://github.com/adrian-ollero/Allegra-MelodicDictation

6.1. *ALLEGRA* - MELODIC DICTATION APP

Allegra is the result of the development process of this project. A complete app prototype has been developed, from a scalable architecture to a user-friendly graphical user interface, including a unique identity with a logo (shown in figure 6.1) and a landing web page.

Figure 6.1: Logo designed for Allegra

The application has been developed using the architecture described in chapter 5. It consists in a main screen from which the different menus can be accessed (figure 6.2).

From this menu, the user can navigate to practice a lesson (1), buy and enjoy a popular song (2), check the statistics and create a new custom dictation (3), play a daily or weekly challenge (4) or simply check the player state status (5) which is the *XP* points and the *MusiCoins* that the user currently owns.

Figure 6.2: Allegra's menu layout

6.1.1. Lessons practice

The practice screen is the the menu where all the guided lesson dictations can be found, as shown in figure 6.3. This screen shows a set of different lessons (1), each one represented as a big gray button, where its information, such as the tittle (4), is displayed. Lessons are organized by the difficulty and experience required to unlock them. There are 4 levels of difficulty: easy, normal, hard and insane, and it is displayed in (5). Each level also has a minimum number of experience points from which it is considered that the user has enough ability to play that level (6), so only if the user reaches this number the lesson would be unlocked. An unlocked level is represented as a light-colored button (2), while a dark one with a locker over it represents a locked lesson (3). Also, as it will be seen in all menus, there is a back button (7), to return to the main menu.

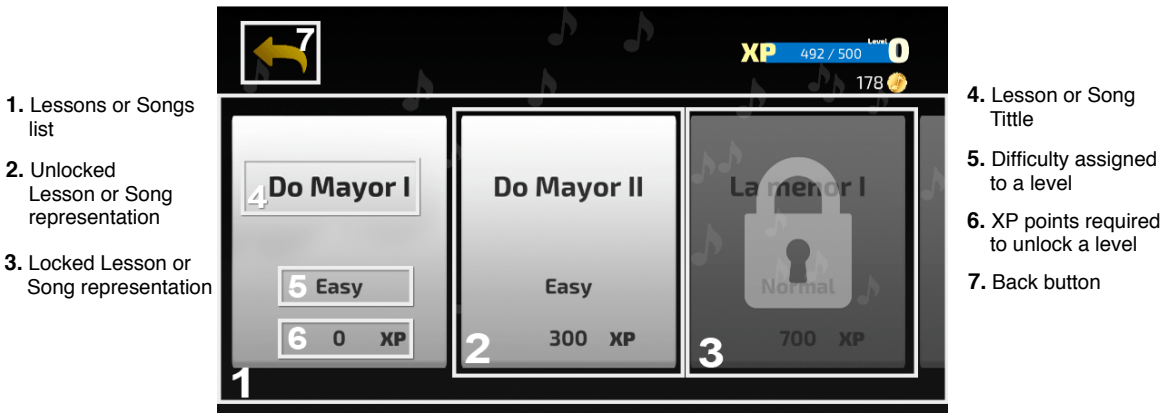


Figure 6.3: Lesson selection screen

The user is free to select any of the unlocked lessons, as many times as he wants, allowing the repetition of a level to ensure that the knowledge it tries to transmit, is acquired by the user, as well as the rewards, that are earned again. Once a lesson is selected, an exercise begins and the exercise screen is displayed.

6.1.2. Popular songs

The popular songs screen is a selection menu, similar to the lessons one, but instead of lessons what can be selected are well-known songs that the user can unlock in exchange for *MusiCoins*, the in-game currency. This way you don't need *XP* points to be able to unlock each level, but those for which the user has enough coins to be able to buy them will be unlocked. As can be seen in the figure 6.4, each of the songs has a certain price (1) and when the user purchases one of them, that amount of coins is subtracted from the total coins of the player.

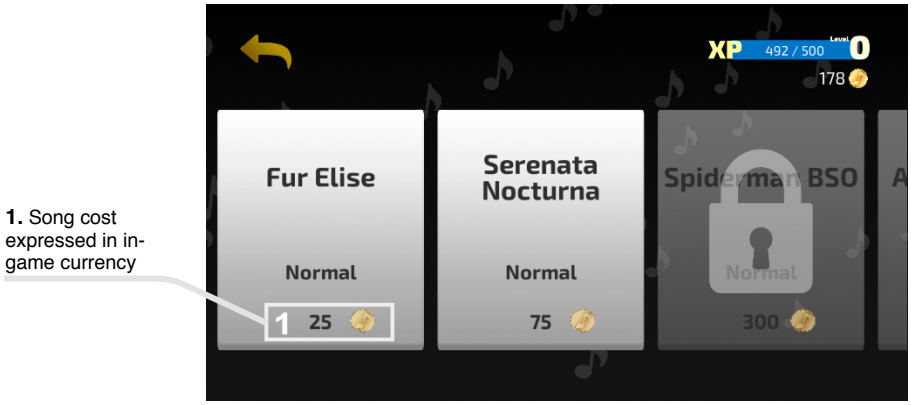


Figure 6.4: Popular songs purchase screen

As it is a direct modification of the user's resources, before carrying out the purchase action, the user is asked to confirm the transaction by means of the dialog shown in the figure 6.5. This dialog is displayed showing the information of the selected song, such as name (2) and the price (3). Here the user can choose between two options: return and not make the purchase of that song with the NO button (4) or accept the purchase and spend *MusiCoins* to play the song with the YES button (5).

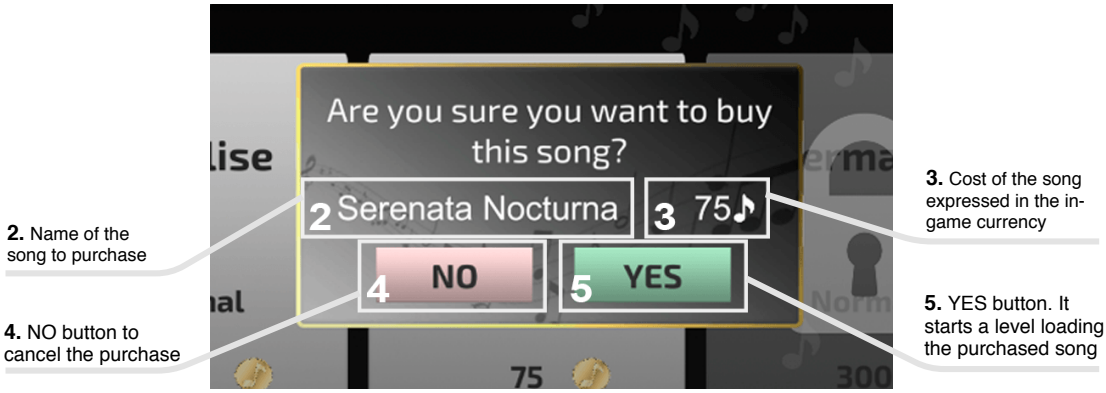


Figure 6.5: Purchase confirmation dialog

Thanks to these popular songs is intended to motivate the user to play and progress to win coins and unlock fun songs. However, these songs only give *XP* points as a reward, as it would not make sense to reward such practices with coins.

6.1.3. Statistics and custom dictations

When the user has spent some time playing and solving the exercises proposed by the application, it is of special interest to know certain data about how efficient is being the progress, the strong points of the user, as well as in which concepts he needs to improve. In order to do this, in the statistics screen your can be helpful.

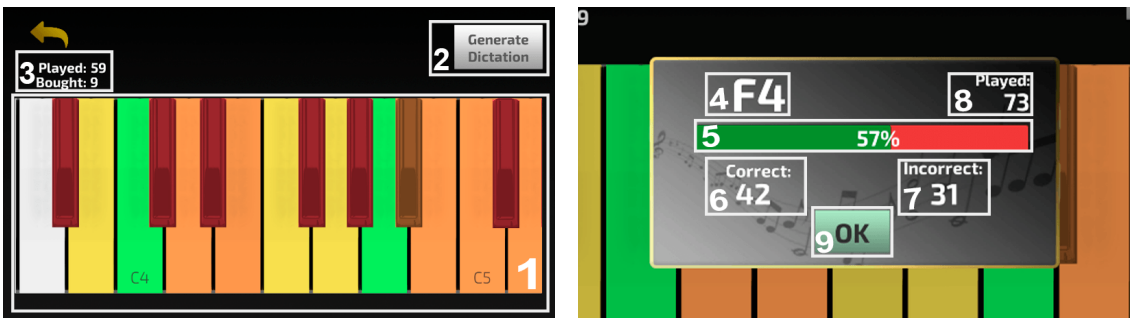
When the user accesses this screen, a piano is shown, as we can see in figure 6.6(a), whose key layout is equivalent to the range of notes that can be worked with *Allegra* . The notes will be coloured according to the user's success rate on that note according to table 6.1. This screen consists on a scroll view (1) to navigate throughout the whole key set, a "Generate Dictation" button (2) and other information that can be displayed as well (3).

The "Generate Dictation" button basically starts an exercise where the notes are selected based on the statistics. The user can specify the number of notes that the exercise is formed by and even customize it completely.

Table 6.1: Relationship between correct note rate and key color

Color	Correct Rate
	0 - 25 %
	25 - 50%
	50 - 75%
	75 - 100%

The keys of the piano are actually buttons, which can be pressed to see more detailed information. An example is shown in figure 6.6(b). This information contains the name of the pressed note (4), a bar showing the percentage success (5) where the green part represents the number of correct hits (6) and the red part the number of incorrect times (7). Also an addition of these two values, which is the total times this note has been played during the exercises, is shown in (8). Finally, to close this dialog the OK button (9) can be used.



(a) Statistics Menu (b) Detailed statistics dialog

Figure 6.6: Allegra's statistics system

6.1.4. Challenges

Challenges are one of the most interesting features of the whole application. They allow you to play levels every 24 hours or 7 days obtaining greater rewards than the regular exercises. To access this screen, in the lower right corner of the main menu it can be seen the "*Challenges button*". This button can take two colors: gray if there is no challenge that can be played, or red with a golden exclamation that indicates that there is at least one challenge unlocked. This is illustrated in figure 6.7.

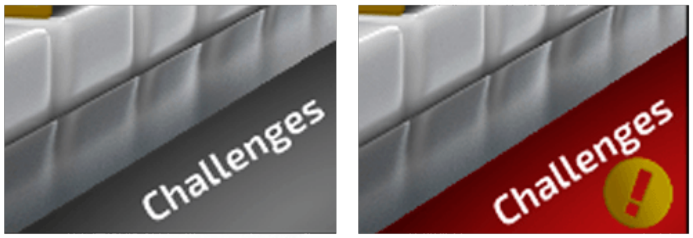


Figure 6.7: Challenges button in main menu changes color according to unlocked challenges.

The challenges screen consists on two main elements (see figure 6.8): the daily challenge button (1) and the weekly challenge button (2). Each of the buttons will launch the corresponding exercise. The rewards if this exercise is different for each one, as can be seen in (3). If a challenge is unlocked and can be played, the button will be enabled, however if it is not, a timer is displayed (4) over the button showing the remaining time until this challenge can be played again.

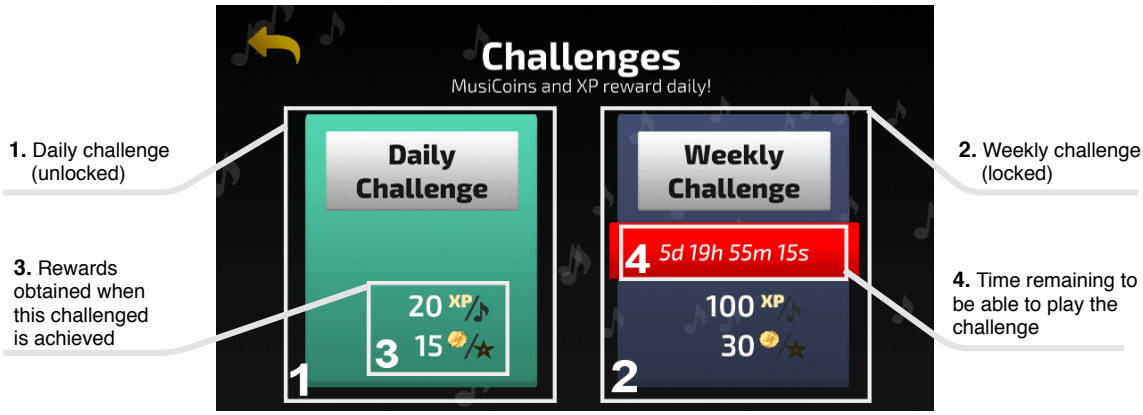


Figure 6.8: Statistics detailed dialog when a key is pressed

6.1.5. Exercise

The exercise screen is where the game really takes place. In *Allegra* the game mode that has been implemented complies with the rules of the traditional Simon Says, that is, the user is asked to repeat a sequence of notes, which increases in number of notes, so that when

it is complete is considered to have passed the level. However, if the note is incorrect, the level is restarted and the initial sequence has to be started again. This approach allows, in addition to working on the identification of notes through auditive perception, to improve musical memory, so that it is easier for you to identify notes within a context.

The exercise, shown in figure 6.9, basically consists on rounds, each one asking a *round note*. This note is represented as a white sleepy music note (1). The user is asked to introduce the lighted up note (2) and any time a *round note* is hit, it is marked as correct, what is represented as a golden happy note (3), and a new round starts. This new round will play the asked note sequence adding a new note, until all the *remaining notes* are completed (4). The background staff (5) is a helpful tool specially for beginners. It represents, in a G-clef, the note played by the user, so he obtained some visual feedback in a representation he may find easier that hearing.

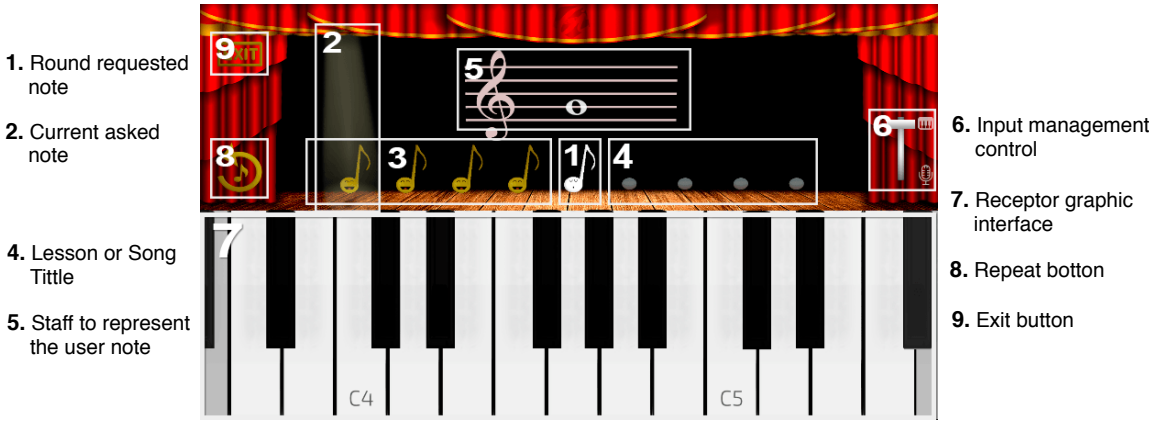
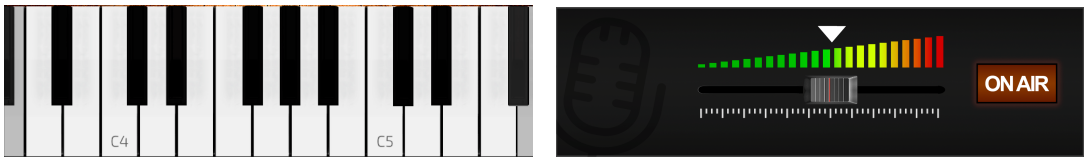


Figure 6.9: Exercise screen used to perform the lessons and songs

The input method adapts to the user and can be changed at any time. Using the input controller (6) the user can switch between the two implemented methods: the Virtual Piano (figure 6.10(a)) and the Microphone Detector (figure 6.10(b)). The graphic interface of these methods is displayed in the bottom part of the screen (7).

To improve the user experience, some controls have been added to the left part of the screen. The repetition button (8) is used to repeat the asked sequence and can be used at any time. The *Exit* button (9) allows to quit the exercise, but if this operation is performed all progress and rewards are lost.



(a) Virtual Piano GUI (b) Microphone Detector GUI

Figure 6.10: Implemented inputs methods

Finally, when an exercise is completed, the whole melody is played and a new screen is loaded. This is the *End Game* screen (figure 6.11) and it is used to display the earned *MusiCoins* (1) and *XP points*(2), and the stars obtained (3). The home button (4) is used to return to the main menu.

Figure 6.11: Final screen when an exercise is being achieved

6.2. LANDING PAGE

The market for mobile applications, as mentioned above, is very saturated as thousands of applications are uploaded every day, so highlighting *Allegra* is a difficult task. However, there are certain techniques and mechanisms that can help to position and promote the application.

In order to publicize *Allegra* , a landing page presenting the app has been developed. This web page, that can be accessed at www.allegramelodicdictation.com, contains keywords that would help in SEO positioning. To obtain which are the most relevant keywords related to this app a tool called *Google Trends*[1] has been used (see Figure 6.12). This tool provides a list of the most common words searched on Google by date, geographical location, etc. and the statistics of some custom words provided by an user. In this case, as the two terms that fit best with the description of the app they both were compared and the most

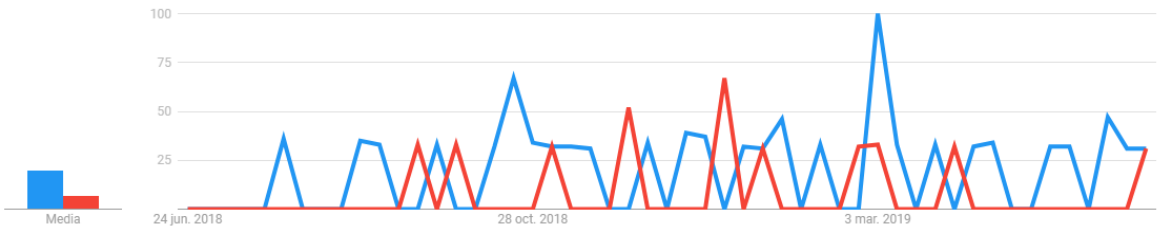


Figure 6.12: Musical Dictation (Blue) vs Ear Trainer (Red) - Google Trends

[1] https://trends.google.es/trends/

Allegra's website has been designed with special attention to these keywords and also includes monitoring tools, such as Google Analytics[2], to improve the quality and information of it.

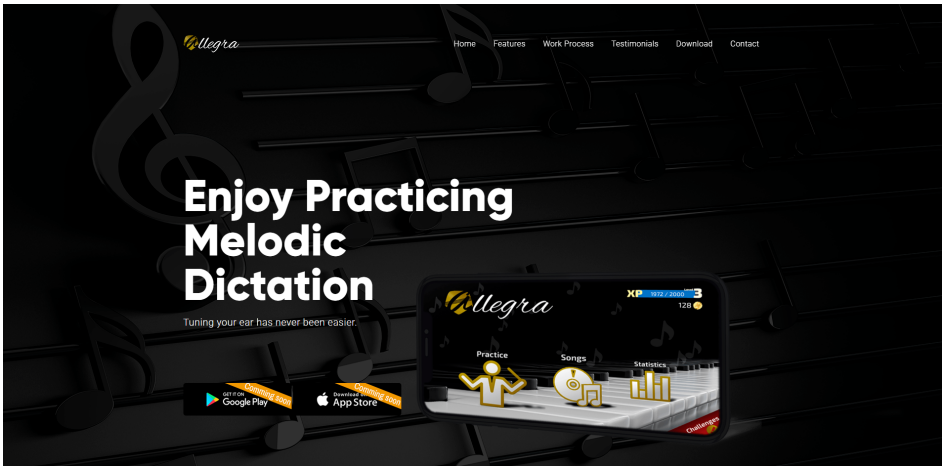


Figure 6.13: Preview of Allegra's Landing Page

6.3. WORK DISTRIBUTION

Starting from the methodology explained in chapter 4, this section will detail the iterations that have shaped this project up to the current result, as well as the tasks that each of them has accomplished. At the end of each one of them, a meeting was held with the tutor and the musician advisor, in order to validate the new functionalities and define the scope and requirements of the next one.

Iteration 1: Initial Plan			
From:	21st January	**To:**	9th February
Starting from the idea of facilitating the practice of melodic dictation, a study is begun on the feasibility of the development and study of similar previous tools. Once the idea is considered viable, tests are carried out with different development tools to find the most suitable one. Among these tools, Unreal Engine and Unity were tested, and the last one was the selected one to develop the system as is more adequate for multimedia mobile apps.			

[2]https://analytics.google.com/analytics/

Iteration 2: Concept test			
From:	10th February	**To:**	17th February
Once the tools and libraries were identified, during the week that lasted this iteration a prototype was developed that reproduced a random note and identified the notes detected through the microphone, drawing them on the screen, and giving feedback on the coincidence of both, in order to validate that the operation of the libraries was correct and the desired. This prototype was also tested on mobile devices to speed up subsequent testing phases on real devices.			

Iteration 3: Design and communication implementation			
From:	18th February	**To:**	5th March
During this iteration, the general design of the application, the distribution in layers and of the responsibilities were carried out. With this in mind, the two communication modules began to be implemented, one built-in the application itself, such as the *Virtual Piano*, and the other using the microphone thanks to the previously tested libraries, and the *Communication Module* that will be in charge of parsing the information generated by the receptors and sending it to the logic module. In this time, an event system is developed that allows communication between the different layers As it has several communication elements, the *Input Manager* module is added in order to be able to switch between them at any time.			

Iteration 4: Exercise implementation			
From:	6th March	**To:**	20th March
With the communication layer working, the logic of a game mode could be implemented. For this purpose, the *Exercise* class was created, which defined the bases that each exercise had to comply with, in order to subsequently proceed with the implementation of the exercise in question. The flow that was decided to follow was that of the well-known game *Simon Says*, whose behavior was replicated and adapted to the context of the exercise. Also, in this iteration, the gamification techniques to be used were defined in a general way, such as the use of experience points, coins to buy songs, scores, etc.			

Iteration 5: Levels scores and gammification techniques			
From:	21st March	**To:**	12th April
With the exercise correctly implemented, levels could be introduced. A brief study was made on which was going to be the format to store the levels, resulting *MusicXML* as the ideal one for this system. Several exercises were defined and exported to this format. In order to be interpreted by the application, the *XMLFileReader* module was implemented, which parsed the information contained in the score files to notes for the exercises. In this iteration, the gamification techniques defined were also implemented. The exercises calculated scores, gave rewards that could be spent for new songs and levels were unlocked according to progress. These techniques could be introduced relatively quickly thanks to the architecture in place to date.			

Iteration 6: Statistics and Challenges			
From:	13th April	**To:**	1st May
From the information produced during the exercises and with the scores calculated at the end of the exercises, a system of statistics was created that the user could consult at any time. Special care was taken to ensure that the report was clear and intuitive to view. A personalized dictation module was also implemented thanks to the processing of these data. In addition, as a mechanism to motivate and engage the user, a module is implemented to manage challenges and their rewards.			

Iteration 7: Graphics improvements			
From:	2nd May	**To:**	22nd May
Up to this point in the development the graphic aspect of the application was quite poor, so this iteration was dedicated to do a graphic review and completely modify the aesthetics of the graphical interface. These modifications include animations of some elements, sound effects on buttons, aesthetic improvement in the menus, among others.			

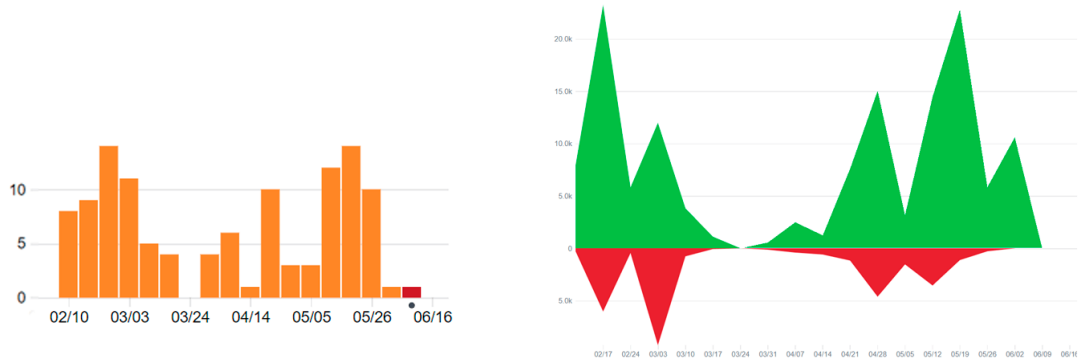
Iteration 8: Firebase			
From:	23rd May	**To:**	30th May
This iteration was intended to improve the persistence layer. Thanks to *Firebase* technology, a persistence system with storage in the new and real-time synchronization could be introduced into the application easily thanks to its high compatibility with Unity. During this iteration, in addition, numerous tests were carried out to ensure the proper functioning of the system.			

Iteration 9: Landing Page and documentation			
From:	31st May	**To:**	21st June
Once the system is in a stable version, a landing page tries to make it known. During this last iteration, this page is prepared and uploaded to the network under the domain *allegramelodicdictation.com*, selected after a brief keyword study. At the same time, the whole document about the project is made.			

6.4. PROJECT STATISTICS

All the work detailed in the previous section has been registered in the *GitHub* repository[3] of this project, so that the data collected in it will be shown next.

The project, at the time of writing of this document, has a total of 128 commits distributed from February 10 to June 20, as shown in Figure 6.14(a) where the 76.0% of the code is written in *C#* and the rest 24.0% is written in many other languages such as the Unity notation used to defined scenes and prefabs, *XML, JSON, ShadelLab*, etc. The only contributor has been the author of this document with a total of 9508 resulting lines, distributed in added and deleted as shown in Figure 6.14(b)).



(a) Commmit per week (b) Lines additions and deletions

Figure 6.14: GitHub repository statistics charts

6.5. DEVELOPMENT COST

Allegra is a project that has been developed between January 21st and June 21st. This period of 4 months has meant a total of 480 hours assuming 4 weeks per month and 5 days a week to half a day, 6 hours. On average, a software developer costs about 30 €/h

[3] https://github.com/adrian-ollero/Allegra-MelodicDictation

according to a popular web page called *Infojobs*[4], which means a total of 14.400 € for the entire development and assuming a single developer, as has been the case of this application.

Other costs to take into account are software pieces used, whose license had a cost. These are Pitch Recognizer, purchased at the Unity Asset Store (18€) and Obri Landing Page Template (9€), to facilitate the development and layout of the website. The website also requires a hosting and a unique domain. For the domain a .com has been chosen, which has cost 9€ and for the hosting the OVH[5] platform has been used, for a total of 10€.

For the development of the application, the following tools have been used: a *MSI GE63 7RD Raider* computer, sold in Amazon for 1299€, a Android smartphone *BQ Aquaris X5 Plus* for testing, currently sold for 219.90€, and the free license of Unity.

Taking into account only the detailed expenses, the project cost results in a total of 15.920,89€. In table 6.2 a summary is shown.

Table 6.2: Total cost calculation (taxes excluded)

Resources	Units	Cost/u	Cost
Pitch Recognizer Package	1u	18,00€	18,00€
Obri Template	1u	9,00€	9,00€
Domain + Hosting	1u	4,99€	4,99€
MSI Ge63 7RD Raider Computer	1u	1.299,00€	1.299,00€
BQ Aquaris X5 Plus	1u	219,90€	219,90€
Developer Salary	480 hours	30,00€/h	14.400,00€
		Total:	15.950,89€

[4]https://www.infojobs.net/
[5]https://www.ovh.es/hosting/

CHAPTER 7

CONCLUSIONS

This chapter will include the conclusions obtained throughout the design and development of this project. Firstly, the degree of satisfaction of each of the specific objectives defined at the beginning will be assessed in order to determine whether the main objective has been achieved. Subsequently, possible future lines of work will be briefly discussed. Finally, the author will conclude with his personal conclusion.

7.1. ACHIEVED OBJECTIVES

This section will detail what solutions have been adopted to achieve each of the objectives defined in the section 2.

1. *Automatic dictation evaluation.* Thanks to the *Comparator* module, implemented inside de *Exercise*, which is capable of resolving whether a note is correct or incorrect, an automatic correction of dictations is achieved, as established in this objective. In addition to this, this action has been enriched with visual feedback in order to inform the user in a simple way the result of the correction.

2. *Progressive learning.* In order to achieve this objective, the application has been divided into levels. The levels have been selected from a book called "*Dictados Progresivos*" by Antonio Zamorano (see [24]). This book contains material for practicing dictation and is widely used and recommended by music schools thanks to its methodology. The methodology used has been developed by its author after many years of experience as a teacher in charge of the discipline of Melodic Dictation. Additionally, to ensure that the user does not access a level without having perfectly understood the previous ones, a system of experience points (*XP*) has been used. In order to access a certain level, a specific amount of these points is required, which are earned by making the levels already unlocked. In this way, the user is forced to play one level several times before being able to advance to the next one.

3. *Student motivation by means of gamification techniques.* This objective is considered achieved thanks to all the gamification techniques used in the final result of the application. Rewards are the main element that contributes to this motivation. The more the application is used, the greater the rewards and benefits earned by the user. Thanks to these rewards, familiar songs can be purchased that can increase the interest of the user, as they convey a strong sense of fun. In addition, to encourage continued use of the application, the challenges offer generous rewards, encouraging the user's return everyday.

4. *Adaptability to enable the use of the system anywhere, anytime.* To ensure this adaptability, the application has been targeted entirely to mobile devices as everyone can carry with them at any time. In addition, the application offers several communication systems that the user can use depending on where they are.

5. *Good user experience.* An attractive, simple and intuitive graphical user interface has allowed to make the user experience pleasant. By means of simple menus, visual feedback and a constant feeling of gameplay, they allow the user to generate positive sensations during the time of application usage. In addition, since there are many student profiles that may be interested, depending on their experience or the instrument they master, in the use of this tool, several communication mechanisms have been implemented. In this way, one of them allows the student to use his favorite instrument to perform the exercises, while another simulates a piano that allows to use the application even not knowing how to play any instrument.

6. *Scalability to integrate new dictations and levels.* This objective can be considered completed thanks to the design adopted for the architecture. On the one hand, starting from the basic concept of Exercise, a multitude of game modes can be derived only by defining their behaviour, as all the necessary communications and dependencies are already defined previously. On the other hand, since the different levels and the notes that each one contains are defined in XML files that the application loads at the beginning of the execution, simply adding in these files the specification of new content, this would be automatically loaded in the application.

7.2. FUTURE WORK

This section will describe the lines of work to be followed in the future as improvements or additions to this project.

- *Publish the application on the market.* Taking a stable version of the application resulting from this project, it would be very interesting to publish it in the most

important application markets, such as Google Play Store and Apple App Store. In order to do so, it would be necessary to package both Android and iOS platforms and upload each of them to their respective platforms. For this to be possible, the corresponding fees charged by each of the platforms must be paid and the validation process must be awaited. Once completed, the application would be public and accessible by any interested user.

- *Add new levels and songs.* With the aim of offering a better quality in the educational aspect, it is proposed to increase the quantity of exercises and to categorize them according to the difficulty, so that they are better adapted to the level of each user. The catalogue of popular songs would also be extended, adapting to the most popular songs of each moment.

- *Database of levels and songs in the cloud.* In order to allow greater flexibility when including new levels and songs, it is proposed to change the location of the files that store the specifications of the levels and songs, currently stored locally, to a database hosted in the cloud. For this purpose, Firebase technology can be used, already employed to store the player's state, which offers the possibility of integrating a storage service in the cloud synchronized in real time with every device. In this way, simply adding new content here would automatically appear in the applications of all users.

- *Usability testing with real students.* In order to verify the real effectiveness of the application developed, it is proposed to carry out tests with potential users. These tests would consist of a pre-test to know the profile of the student and his experience with the practice of melodic dictation. The user would then test the application for a few minutes, carrying out a few exercises and experimenting with the different functionalities offered. Finally, a final test would be carried out, this time on the aspects worked on in the application and their opinion about it.

- *Tutorial for beginners.* In order to help those who do not have previous knowledge of music to become familiar with it, it is proposed to elaborate a step-by-step tutorial adapted to the students' unknowledge of the musical concepts involved, and a set of very basic and guided levels to introduce this complex competition in a very progressive way.

- *Multi-language support.* In order to make the application available to a wider and more international audience, several languages are proposed to be supported. To achieve this, it is possible to use a file that stores all the text strings of the application and loads only those that correspond to the language specified by the user.

- *Teacher Monitoring Tool.* It is proposed to develop a system that allows teachers to control the levels and monitor the activity of each of their students who use the application. In this way, the selection of levels would be more efficient and the effectiveness of the use of this tool would increase. To achieve this, the use of Firebase technology and its Realtime Database can be used. In this database are already stored the statistics of each user, and could also store the specifications of the levels. In this way, by means of a tool, for example a web platform, the teacher can have access to the information related to the student, viewing and assigning his practices and dictations.

7.3. PERSONAL OPINION

This is the project that puts an end to my university studies. After 4 years of a lot of effort and work, I conclude a period in which I have learned about my passion and I have been able to grow and train myself to be a good computer engineer, and this work aims to reflect, in a single project, all the skills and competences acquired.

Allegra 's design and development process has allowed me to experience first-hand what it means to face a development of this magnitude, with all the difficulties that this brings. First of all, designing a complete system paying attention to requirements such as scalability and usability, has been a complex task for me, but thanks to what has been studied about it. I think it has been quite achieved. In addition, the use of an agile methodology has been a great success and very helpful, as it has allowed adapting and reacting quickly to unexpected events and changes.

Moreover, Unity, the tool used to accomplish the development, was a technology that before this project had barely used and always from the graphic side. Thanks to this development I have been able to deepen in the power that it gives us and to know how to take advantage of it to obtain solid systems.

Finally, I would like to emphasize that, thanks to this project, I have been able to verify the scope of Computer Engineering, and how it can complement other areas. From my point of view, applications such as *Allegra* , which can increase the interest of children and adults for, in this case, melodic dictation, are necessary for a society so adapted to new technologies, to appreciate and take advantage of traditional practices and disciplines. For this reason, after the dedication and effort that this work has meant, I would be very excited to see this system being used by students, thus contributing to their training as professional musicians.

APPENDICES

APPENDIX A

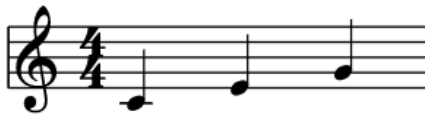
EXERCISES SOLUTIONS

This appendix aims to show the solutions of all the exercises and songs included in *Allegra* in order to show the progression they follow.

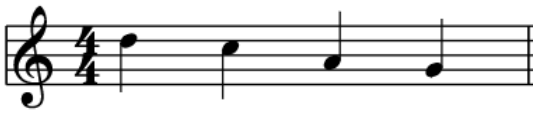
A.1. LESSONS

These are the exercises that are intended to introduce different musical concepts to the student.

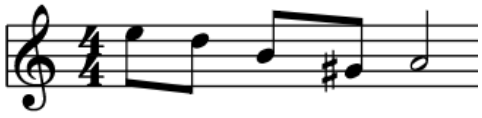
Do Mayor I:



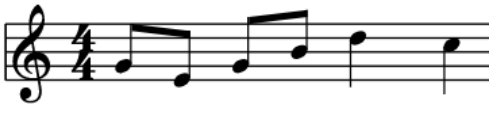
Do Mayor II:



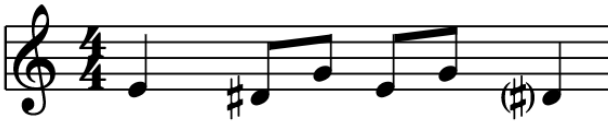
La Menor I:



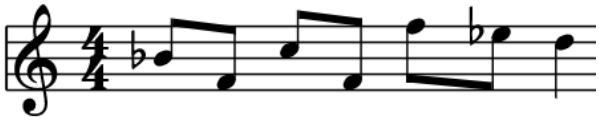
Sol Mayor I:



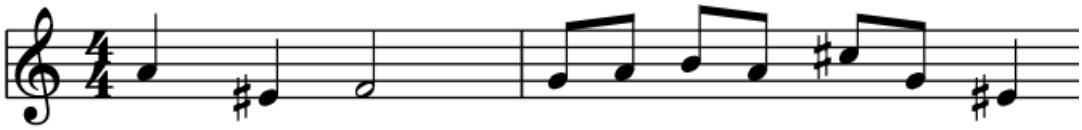
Mi Menor I:



Si b Mayor:

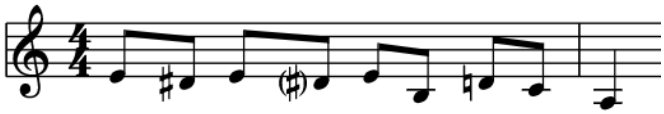


Fa # Mayor:

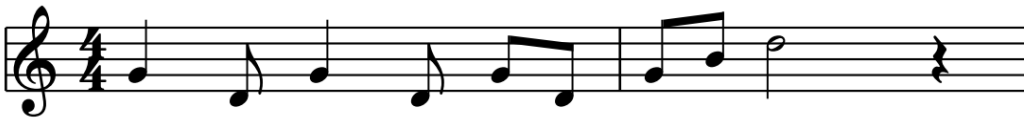


A.2. POPULAR SONG

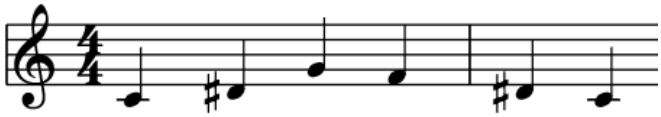
Fur Elise by Beethoven:



Serenata Nocturna by Mozart:



Spiderman BSO by Danny Elfman:



Avengers BSO by Alan Silvestri:

APPENDIX B

JSON TREE IN FIREBASE REALTIME DATABASE

This is the JSON tree used in the Firebase Realtime Database with data examples. Each user that uses the application will be assigned an ID, in this case *«player_test»*.


```
{
  "player_test" : {
    "challenges" : {
      "daily" : 1559738073,
      "weekly" : 1560736986
    },
    "state" : {
      "coins" : 178,
      "xp" : 1792
    },
    "statistics" : {
      "songs" : {
        "bought" : 10,
        "played" : 62
      },
      "notes" : {
        "A3" : {
          "correct" : 0,
          "incorrect" : 3
        },
        "A4" : {
          "correct" : 151,
          "incorrect" : 29
        },
        "B3" : {
          "correct" : 22,
          "incorrect" : 14
        },
```

```
        "B4" : {
          "correct" : 43,
          "incorrect" : 58
        },
        "C4" : {
          "correct" : 104,
          "incorrect" : 9
        },
        "C5" : {
          "correct" : 22,
          "incorrect" : 26
        },
        "D4" : {
          "correct" : 19,
          "incorrect" : 35
        },
        "D5" : {
          "correct" : 13,
          "incorrect" : 27
        },
        "E4" : {
          "correct" : 35,
          "incorrect" : 30
        },
        "E5" : {
          "correct" : 3,
          "incorrect" : 25
        },
        "F4" : {
          "correct" : 46,
          "incorrect" : 32
        },
        "F5" : {
          "correct" : 1,
          "incorrect" : 25
        },
        "G4" : {
          "correct" : 54,
          "incorrect" : 33
        },
        [...]
      }
    }
  }
}
```

BIBLIOGRAPHY

[1] Pekka Abrahamsson et al. "Mobile-D: an agile approach for mobile application development". In: *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. ACM. 2004, pp. 174–175.

[2] Mónica Balo González, Pilar Lago Castro, and Luis Ponce de León Barranco. "Los alumnos ante el dictado musical: las TIC como aliadas para mejorar las experiencias". In: *Didáctica, innovación y multimedia* 28 (2014), pp. 01–14.

[3] José H Canós and Mª Carmen Penadés Patricio Letelier. *Metodologías ágiles en el desarrollo de software*. 2012.

[4] Yu-Kai Chou. *Actionable Gamification: Beyond Points, Badges and Leaderboards*. Octalysis Media, 2014. ISBN: 978-1511744041.

[5] Patricio De La Cuadra, Aaron S Master, and Craig Sapp. "Efficient Pitch Detection Techniques for Interactive Music." In: *International Computer Music Association Journal (ICMC)*. 2001.

[6] E. Gamma et al. *Patrones de diseño: elementos de software orientado a objetos reutilizable*. Addison-Wesley professional computing series. Pearson Educación, 2002. ISBN: 978-8478290598.

[7] Robert Gauldin. *Harmonic practice in tonal music*. WW Norton, 1997. ISBN: 978-0393976663.

[8] Fabian Groh. "Gamification: State of the art definition and utilization". In: *Institute of Media Informatics Ulm University* 39 (2012), p. 31.

[9] Iván Linares. "Clash Royale es una mina de oro, Supercell gana miles de millones". In: *El Androide Libre* (Feb. 2017).

[10] David R Michael and Sandra L Chen. *Serious games: Games that educate, train, and inform*. Muska & Lipman, Premier-Trade, 2005. ISBN: 978-1592006229.

[11] Tomás Thayer Morel. "Música y tecnología: taller para la integración de las TIC en el aula de educación musical". In: *Contextos: Estudios de Humanidades y Ciencias Sociales* 27 (2016), pp. 109–124.

[12] Laurence Moroney. "The Firebase realtime database". In: *The Definitive Guide to Firebase*. Springer, 2017. ISBN: 978-1484229422.

[13] Galera Núñez, Mª del Mar, and Rosario Gutiérrez Cordero. *La tecnología musical como herramienta didáctica*.

[14] Michael Pilhofer and Holly Day. *Music theory for dummies*. John Wiley & Sons, 2015. ISBN: 978-0764578380.

[15] Alejandro Ramirez. "Introducción a los Patrones de Diseño". In: *Creative Commons. Agosto* (2004).

[16] K. Restivo. "Worldwide Quarterly Mobile Phone Tracker". In: *International Data Corporation* ().

[17] Mar Rodrigo Fernandez. *El potencial educativo de la música*. 2015.

[18] Paolo Roma and Daniele Ragaglia. "Revenue models, in-app purchase, and the app performance: Evidence from Apple's App Store and Google Play". In: *Electronic Commerce Research and Applications* 17 (2016), pp. 173–190.

[19] David Talkin. "A robust algorithm for pitch tracking (RAPT)". In: *Speech coding and synthesis* 495 (1995), p. 518.

[20] Ailie KY Tang. "Mobile app monetization: App business models in the digital era". In: *International Journal of Innovation, Management and Technology* 7.5 (2016), p. 224.

[21] Unity Technologies. *Unity User Manual (2019.1)*. URL: https://docs.unity3d.com/Manual/index.html.

[22] Daniel Terdiman. "No, Flappy Bird developer didn't give up on $50,000 a day". In: *CNET* (Feb. 2014).

[23] D. Vallejo Fernández and S. Sánchez Sobrino. *Curso de Experto en Desarrollo de Videojuegos: Módulo 1*. Universidad de Castilla-La Mancha, 2019. ISBN: 978-1517309558.

[24] A Zamorano. *Dictados progresivos*. 1997. ISBN: 978-8488038951.

[25] B.C. Zapata. *Android studio application development*. Packt Publishing Ltd, 2013. ISBN: 978-1783285273.