PROYECTO FIN DE CARRERA

Título:	Desarrollo de un Videojuego Móvil Multiplataforma de Educación Infantil Musical utilizando el Entorno de desarrollo Unity3d
Título (inglés):	Development of a Multi-Platform Mobile Musical Training Software for Children using the framework Unity3D Engine
Autor:	Unai Arríen Oroz
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:

Vocal:

Secretario:

Suplente:

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

PROYECTO FIN DE CARRERA

DEVELOPMENT OF A MULTI-PLATFORM MOBILE MUSICAL TRAINING SOFTWARE FOR CHILDREN USING THE FRAMEWORK UNITY3D ENGINE

Unai Arríen Oroz

Marzo de 2017

Resumen

Esta memoria es el resultado de un proyecto cuyo objetivo es el de desarrollar un videojuego móvil multiplataforma de educación infantil musical utilizando el entorno de desarrollo Unity3D.

Este desarrollo comprende los siguientes procesos: análisis de los requisitos, diseño de la arquitectura, implementación del videojuego, testeo del mismo y despliegue para las plataformas iOs y Android.

Así mismo, se utilizan varios plugins obtenidos a través de la Asset Store de Unity3D, que nos han permitido mejorar el rendimiento del videojuego y facilitar su desarrollo.

Además de los plugins mencionados, se integra Flurry Analytics como sistema de obtención de métricas y errores de uso de la aplicación para facilitar su posterior mantenimiento.

Por último, se presentan las conclusiones extraídas del trabajo, así como los objetivos que se han alcanzado tras completar el desarollo.

Palabras clave: Desarrollo multiplataforma, Desarrollo móvil, Unity3D, Asset Store, Android, iOs, Flurry, 2D Toolkit, HOTtween, Videojuego, Desarrollo software, UML

Abstract

This thesis is the result of a project whose objective is to develop a multi-platform mobile musical training software for children using the framework Unity3D engine.

This multi-platform development covers several processes: requirement analysis, architecture design, videogame implementation, testing this implementation and iOs and Android deployments.

Moreover, we use plugins obtained through Unity3D Asset Store, which allow us to improve the videogame performance and make its development easier.

Furthermore, we have integrated Flurry Analytics as a metrics an error tool management in order to facilitate future application maintenance.

Finally, we gather the extracted conclusions and check if we achieve the goals defined in the analysis requirement stage.

Keywords: Multi-platform development, Mobile development, Unity3D, Asset Store, Android, iOs, Flurry, 2D Toolkit, HOTtween, Videogame, Software development, UML

Agradecimientos

Después de estar casi tres años para escribir este proyecto la lista de agradecimientos podría ser grandísima, pero voy a agradecérselo tan sólo a la gente que quiero y que ha sido importante para mí.

Muchas gracias Sara por haberme aguantado durante estos años tan bonitos a tu lado. A mis padres y a mi abuela por darlo todo por mi hermano y por mí. A Aitor por haber tenido la oportunidad de ver mi bonito carácter todos estos años y por pasárnoslo de lujo con él cuando está majete.

Muchas gracias a Mariano por ser un master de Unity y tener las ganas y la paciencia de enseñar todo lo que sabe en cualquier momento.

Muchas gracias a Luis Fernando D'Haro por adentrarnos en el desarrollo de aplicaciones y, en especial, a Carlos Ángel por convertirme en un programador pragmático y hacer que me encante mi trabajo.

Muchas gracias a las grandes personas que conocí en la Escuela y que hicieron que esos años fueran geniales.

Por último muchas gracias a todos mis amigos de Stratio por decirme todos los días que escribiera este proyecto y también por hacer que me encante trabajar con ellos.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

This chapters provides an introduction to the problem which will be approached in this project. It provides an overview of the multi-platform application development technologies. Furthermore, a deeper description of the project and its environment is also given.

1.1 Context

In the last years, mobile devices ecosystem have experimented a huge transformation. These devices have evolved improving their characteristics and changing the way we use them.

Although some of these new features, like an improved performance, have made mobile application development easier, the device market suffers a huge fragmentation which causes lots of problems when we have to deal with multi-platform mobile application developments. This phenomenon, fragmentation, occurs when some mobile users are running different operating systems or different versions of the same operating system (software fragmentation) or when some mobile users are using older devices with less powerful characteristics (hardware fragmentation).

On one hand, the coexistence of different mobile OS such as Android, iOs or Windows Phone and on the other hand the wide range of screen sizes and resolutions make multi-platform mobile application development very complicated. This problem is presented greatly within consulting companies, whose clients requires multi-platform and multi-devices application developments.

When developing native applications, developers implement an application for one specific target platform using its software development kit (SDK) and frameworks. The app is tied to that specific environment. For example, applications for Android are typically programmed in Java, access the platform functionality through Android's frameworks, and render its user interface by employing platform-provided elements. In contrast, applications for iOS use the program- ming language Objective-C and Apple's frameworks.

In case multiple platforms are to be supported by native applications, they have to be developed separately for each platform. This approach is the opposite of the cross-platform idea. [1]

In order to find a solution, reducing development effort and costs, we need to use multi-platform development tools that allow developers to eliminate the immense effort required to build one mobile applications for each mobile SO we need.

Cross-platform development approaches emerged to address this challenge by allowing developers to implement their apps in one step for a range of platforms, avoiding repetition and increasing productivity. [1]

Usually, targeting more than one platform normally requires developing a corresponding application for each mobile operating system. However, this approach means that the

development time and hence the cost of the product will increase. A cross-platform approach, on the other hand, helps to solve this problem by developing a single code base that supports multiple platforms. Another benefit of cross-platform development, is that they allow changes to be made faster to portable mobile applications, as only one single code base needs to be modified. [2]

Within the game application development context for multi-platform devices, there are some tools such as Unity3D, Unreal Engine, Marmalade, Autodesk or Corona SDK.

Between all these development tools, Unity3D stands out due its soft learning curve, the possibility of development for multiple platforms in the same project and its huge community.

Moreover, the functions that Unity3D supports autonomously are very abundant. In fact, all game developments are possible such as shader, physics engine, network, terrain manipulation, audio, video, and animation, and it considered so that the revision is possible to the taste of user according to the need.

Unity3D that produces based on Java script and C# can apply and manage after producing the desired functions with script, not producing all of the programing at once. GUI composed on screen helps the first-time developer to approach easily, and the script and program that programer made with simple mouse drag. [3]

Furthermore, Unity3D Asset Store, which is driven by Unity3D community, includes community plugins which give us an added value.

The proposal of this project is to exemplify the development of a multi-platform mobile musical training software for children using the framework Unity3D Engine.

1.2 Master thesis description

The aim of this master thesis is the development of a multi-platform mobile musical training software for children using the framework Unity3D Engine.

This development has the purpose of training children musical skills. Through physical instrument miniatures which represent the five instrument families (percussion, keyboards, strings, woodwind, brass) the gamer will be able to interact with the application software in three different forms.

Firstly, the gamer will have the possibility to play one instrument of the selected instrument family. Secondly, the gamer will be able to interact with a whole orchestra in order to

enable or disable the instruments which will be playing a classical piece. Finally, the gamer will have the possibility to read the history and characteristics of each instrument family.

The multi-platform game development includes all the following development stages:

- *Requirement analysis*, determining the needs or conditions to meet for the game application.
- *Architecture design*, defining a structured solution that meets all of the application requirements.
- *Physical instrument pieces design*, creating the physical pieces that will be used to interact with the application.
- *Software implementation*, building the multi-platform game using Unity3D engine.
- *Software test*, testing the application to check if it accomplish the acceptance criteria.
- *Application deployment*, deploying the application to the needed application markets.

Within application architecture we can distinguish the following modules:

Unity3D engine is a cross-platform game engine developed by Unity Technologies and used to develop video games for PC, consoles, mobile devices and websites. Unity is notable for its ability to target games to multiple platforms. Within a project, developers have control over delivery to mobile devices, web browsers, desktops, and consoles. Supported platforms include Android, Apple TV, BlackBerry 10, iOS, Linux, Nintendo 3DS line, macOS, PlayStation 4, PlayStation Vita, Unity Web Player (including Facebook), Wii, Wii U, Windows Phone 8, Windows, Xbox 360, and Xbox One. [4]

Unity Asset Store is where a growing library of free and commercial assets are placed. These assets are created both by Unity Technologies and also members of the community. A wide variety of assets is available, covering everything from textures, models and animations to whole project examples, tutorials and Editor extensions. These assets are accessed from a simple interface built into the Unity Editor and are downloaded and imported directly into your project.

Flurry Analytics enable users to analyze consumer behavior through data observations. The platform provides features for user segmentation, consumer funnels, and applications portfolio analysis. The platform's funnels measure customized consumer conversions and trending metrics, while the portfolio analytics feature allows companies

to manage entire portfolios of mobile applications with the ability to monitor data about overlap among applications as well as up-sell and cross-sell conversions. [5]

1.3 Master thesis goals

The main purpose of this master thesis is to build a multi-platform mobile musical training software for children using the framework Unity3D Engine.

This multi-platform software includes the software application and the physical miniatures that represent the musical instrument family. The gamer will be able to interact with the software application through these physical instrument pieces.

Regarding the software application, it will have three different game modes which will allow the gamer to play different musical melodies with different musical instruments. Also, the gamer will be able to conduct a whole orchestra and to discover information about all instrument families represented by the physical miniatures instruments.

The multi-platform software development process includes the **requirement analysis** and the **architecture design**. Due to the need of physical pieces to interact with the application the build process also include both **physical instrument pieces design** and **instrument recognition algorithm design**. After building the game application it has to go through previously designed **software tests** to check it fits the requirements obtained in the previous stages.

Finally, the application should be deployed to Android and iOs application markets to let users to download and use it, as long as the development team and the client would retrieve metrics and other information about the application use.

1.4 Structure of this Master Thesis

In this section we will provide a brief overview of all the chapters of this Master Thesis. It has been structured as follows:

Chapter 1 provides an introduction to the problem which will be approached in this project. It provides an overview of the benefits of Unity3D engine. Furthermore, a deeper description of the project and its environment is also given.

Chapter 2 contains an overview of the existing technologies on which the development of the project will rely.

Chapter 3 describes one of the most important stages in software development: the requirement analysis using different scenarios. For this, a detailed analysis of the possible use cases is made using the Unified Modeling Language (UML). This language allows us to specify, build and document a system using graphic language. The result of this evaluation will be a complete specification of the requirements, which will be matched by each module in the design stage. This helps us also to focus on key aspects and take apart other less important functionalities that could be implemented in future works.

Chapter 4 describes the architecture of the system, dividing it into two groups and differencing application software development and physical instrument pieces design.

Chapter 5 describes selected use cases. It is going to be explained the interaction with the whole game going through two of its game modes.

Chapter 6 sums up the findings and conclusions found throughout the document and gives a hint about the work done for this master thesis.

Finally, the appendix provide useful related information, especially covering the application screens.

CHAPTER 2

Enabling Technologies

This chapter introduces which technologies have made possible this project. First of all there must be an engine to build the game application, this is Unity game engine, explained in section 2.1. Secondly, Unity integrates third part software as plugins to facilitate development through its Asset Store, this components are detailed in section 2.2. Finally, there should be a tool to manage application metrics and information after its deployment, this is done by Flurry Analytics, explained in section 2.3.

2.1 Unity3D game engine

Today's game creators rely on game engines to develop the main pieces of software for their games. A game engine simplifies the task of the programmers by offering convenient abstractions for the hardware and operating systems on top of which the game runs. [6]

Gamers play on so many different types of devices which have lots of differences regarding their hardware and software resources. One of the main purposes of a game engine is to make this development easier avoiding building one application for each device we want to be compatible with.

Unity3D game engine is an integrated development tool for producing other interactive contents such as video game, architectural visualization, real-time 3D animation. Its editor runs on Window, Mac OS X, so it could make games as the platforms of Window, Mac, Wii, iPad, and iPhone. It could also produce web browser game that uses unity web player plug-in. This is a similar form of flash, and it is designed so that flash user could easily adapt even with cross domain security policy and scripting.

The functions that Unity3D supports autonomously are very abundant. In fact, all game developments are possible such as shader, physics engine, network, terrain manipulation, audio, video, and animation, and it considered so that the revision is possible to the taste of user according to the need. Unity3D that produces based on Java script and C# can apply and manage after producing the desired functions with script, not producing all of the programing at once. GUI composed on screen helps the first-time developer to approach easily, and the script and program that programmer made with simple mouse drag. [3]

Figure 2.1: Unity3D logo

Unity3D is a flexible and high-performance development platform used to make creative and intelligent interactive 3D and 2D experiences. The "author once, deploy everywhere" capability ensures developers can publish to all of the most popular platforms. Unity Technologies boasts a thriving community of over 2 million developers including large publishers, indie studios, students and hobbyists. To remain at the forefront of innovation, Unity Technologies tirelessly re-invests in its award-winning 3D development tools and its

democratization initiatives, such as the Asset Store digital content marketplace and Unity Games publishing and distribution division. [7]

We will detail Unity3D concepts and how those concepts will be integrated in the application development in the following subsections. Firstly we will detail Unity3D principal features in 2.1.1, secondly in 2.1.2 we are going to detail all the Unity3D principal concepts and finally in 2.1.3 we are going to explain Unity3D interface from where we create our game.

2.1.1 Unity3D features

The latest update, Unity 4.6, was released in August, 2012. It currently supports development for iOS, Android, Windows, OS X, Linux, web browsers, Flash, PlayStation 3, Xbox 360, and Wii U. [4] The game engine can be downloaded from their website in two different versions: Unity and Unity Pro. Between its features we can point highlight the following:

- **Rendering**, The graphics engine uses Direct3D (Windows), OpenGL (Mac, Windows, Linux), OpenGL ES (Android, iOS), and proprietary APIs (Wii). There is support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects. Unity supports art assets and file formats from 3ds Max, Maya, Softimage, Blender, Modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks and Allegorithmic Substance. These assets can be added to the game project, and managed through Unity's graphical user interface. [8] The ShaderLab language is used for shaders, supporting both declarative "programming" of the fixed-function pipeline and shader programs written in GLSL or Cg. A shader can include multiple variants and a declarative fallback specification, allowing Unity to detect the best variant for the current video card, and if none are compatible, fall back to an alternative shader that may sacrifice features for performance. [9]

- **Scripting**, The game engine's scripting is built on Mono, the open-source implementation of the .NET Framework. Programmers can use UnityScript (a custom language with ECMAScript-inspired syntax), C# or Boo (which has a Python-inspired syntax). [10] Starting with the 3.0 release, Unity ships with a customized version of MonoDevelop for debugging scripts. [11]

- **Asset Tracking**, Unity also includes the Unity Asset Server, a version control solution for the developer's game assets and scripts. It uses PostgreSQL as a backend, an audio system built on the FMOD library (with ability to playback Ogg Vorbis compressed

audio), video playback using the Theora codec, a terrain and vegetation engine (which supports tree billboarding, Occlusion Culling with Umbra), built-in lightmapping and global illumination with Beast, multiplayer networking using RakNet, and built-in pathfinding navigation meshes. [12]

- **Platforms**, Unity supports deployment to multiple platforms. Within a project, developers have control over delivery to mobile devices, web browsers, desktops, and consoles. [4] Unity also allows specification of texture compression and resolution settings for each platform the game supports. [4] Currently supported platforms include Windows, Linux, Mac, Android, iOS, Unity Web Player, Adobe Flash, PlayStation 3, Xbox 360, and Wii. Although not officially confirmed, Unity also supports the PlayStation Vita as can be seen on the game Escape Plan. Upcoming platforms include BlackBerry 10, Wii U, Windows 8, and Windows Phone 8.

- **Asset Store**, Launched in November 2010, the Unity Asset Store is a resource available within the Unity editor. The store consists of a collection of over 4,400 asset packages, including 3D models, textures and materials, particle systems, music and sound effects, tutorials and projects, scripting packages, editor extensions and online services. We will detail this feature in section 2.2. The store also contains many extensions, tools and asset packages such as the package 2D Toolkit, which provides an efficient & flexible 2D sprite, collider set-up, text, tilemap and UI system integrating seamlessly into the Unity environment.

- **Versions**, The first version of Unity was launched at Apple's Worldwide Developers Conference in 2005. It was built to function and build projects on Mac computers and garnered enough success to continue development of the engine and tools for other platforms.[2] Unity 3 was released in September 2010 and focused on introducing more of the tools that high-end studios have at their disposal. This allowed the company to capture the interest of bigger developers while providing independent and smaller teams with a game engine in one affordable package. The latest version of Unity, Unity 4.6, was released in August 2014, and includes features such as New UI System: Design UIs for your game or application using Unity's powerful new component based UI framework and visual tools and an extensible event messaging system.

2.1.2 Unity3D concepts

In these subsections, we are going to explain the Unity3D basic concepts, which are the following:

- **Assets**, Assets are the actual contents we use to shape our own Unity world. Usually, most of the Assets are created using external software and then imported into the project. On their own they are no more than media content files: 3D models, textures, audio files, scripts and so on. In order to exist in our product the Assets need to be included inside a Unity Scene.

- **Scenes**, A Scene is a self-contained 3D space where all the action happens. Every Unity Project needs to have at least one Scene. Without it, the project will be just a pile of stored Assets because they have no place to actually exist. Most project will have multiple Scenes. The most common use of Scenes in game development is to create different levels inside a game, however this will largely depend on the project specifications and design. Although it may have as many as necessary, it is important to state that only one Scene is active and running at a given time. So, Scenes are the 3D space where Assets can exist. But, to include the Assets on a Scene we need GameObjects.

- **Game Objects**, A GameObject is the base entity of our Scene. If we want anything to exist on our Scene it needs has to be a GameObject. But by itself is no more than that. It just represents something that exists inside our 3D world and that's it. If a GameObject wants to be something more it needs to have Components.

- **Components**, A Component is responsible for assigning roles, properties and/or behaviors to GameObjects. It is the smallest building block on our world and by far the most important one. For example, If we want a GameObject to represent a Light in our Scene we just attach it a Light Component. If it is a static rock in our environment we attach the necessary Components for it to be a rock (we need to display the rock shape and texture and maybe a collision box). If we need a more complex GameObject such as our Player Avatar, we would need to attach a component for its' shape, textures and all his behaviors. So as you can see, according to the Component type (and believe me they are many) we can attach assets to our GameObjects, define properties, assign behaviors scripts and so on.

- **Prefab**, A Prefab is a template for a GameObject. It allows us to store a GameObject with Components and Properties already set. We can even store a full hierarchy of GameObjects, all with their own Components and default values.

2.1.3 Unity3D interface

When we create a project, the first thing we see is Unity3D interface. This interface can be easily customized so that it would fits developer needs. In figure 2.2 we can see the interface that we have to used during the game development process.

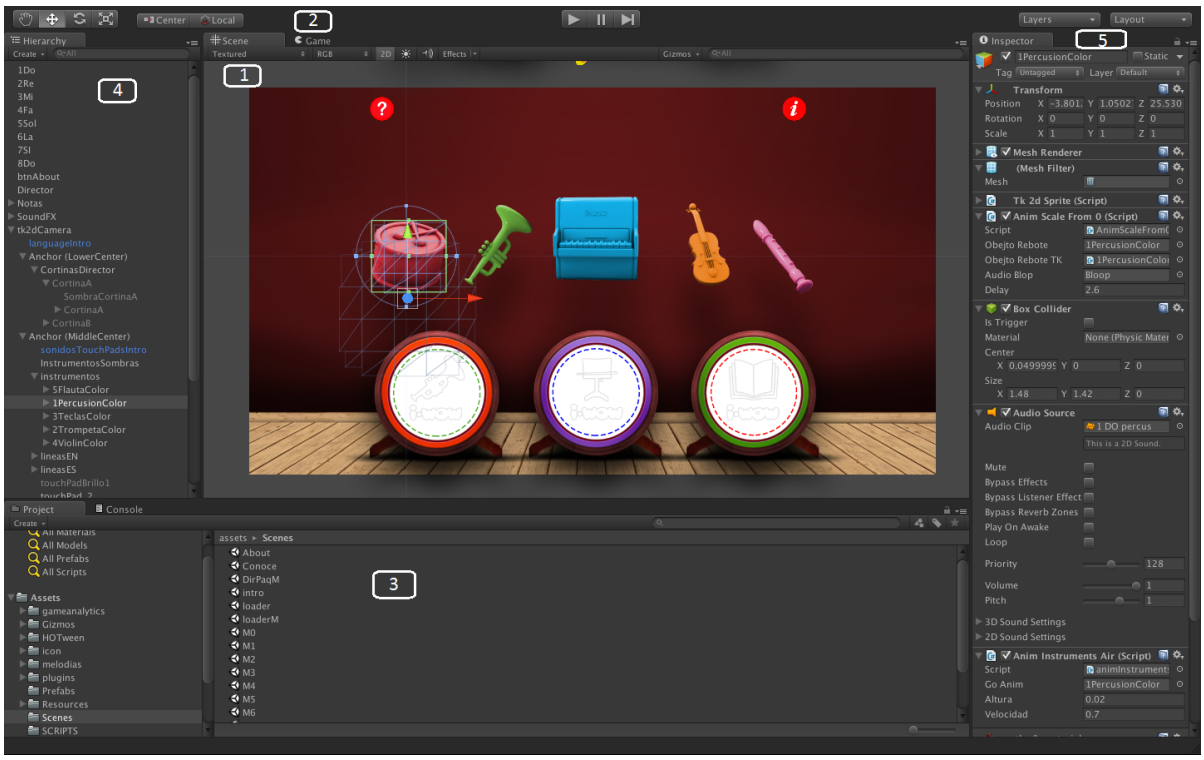

Figure 2.2: Unity3D interface

In figure 2.2 we can differentiate five principal areas:

- **Area 1 (Scene)**, this area contains the scene we are editing. This is the zone where we we will create and place the graphic components of the scene.
- **Area 2 (Game)**, in this window we will obtain the pre-visualization of the scene in execution. When we want to execute the scene, we have just to click that play button that is located in the top. We are also able to pause and a stop that execution with Pause and Stop buttons.
- **Area 3 (Project)**, it defines our project resources structure. It is divided into two parts. The left one shows all the folders that contains our project resources. In our case everything is structured around a folder called Assets. The right one shows the selected folder's content. We can also move archives to our project dragging them to this area.

- **Area 4 (Hierarchy)**, this area shows our scene hierarchy. Here every object included in the scene will appear.
- **Area 5 (Inspector)**, using this window we will be able to see and edit the selected object's properties.

2.2 Unity Asset Store

The Unity Asset Store is home to a growing library of free and commercial assets created both by Unity Technologies and also members of the community. A wide variety of assets is available, covering everything from textures, models and animations to whole project examples, tutorials and Editor extensions. The assets are accessed from a simple interface built into the Unity Editor and are downloaded and imported directly into your project. [13]

In the following subsections, we are going to detail the four plugins we are using in our development and why we have chosen them:

2.2.1 2D Toolkit

We decided to use this asset due to the game was designed to be built in a two dimensional space. Although Unity3D 4.3 version introduced A new native 2D toolset, 2D Toolkit provide developers additional features and benefits.

Figure 2.3: 2D Toolkit logo

2D Toolkit works in harmony with Unity's 2D features, complements and expands upon it in many ways. The package is actively developed and is designed to put more power in the hands of the 2D game developer, and blend seamlessly into Unity. [14]. These additional features are the following:

- *Painless handling of multiple device resolutions*, 2D Toolkit provides several tools, such as a special camera component (called tk2dCamera), and multi platform textures (e.g. for high density Retina devices) to almost completely eliminate the hassle around creating games for many devices. These tools help us to create pixel perfect scenes with automatic texture swapping for 1x, 2x and 4x resolutions. These changes are transparent to us, so your code stays clean as we don't have to reposition anything.

- *Advanced atlasing features*, 2D Toolkit's advanced atlasing (fully supported in both Unity Free and Pro) allows for some game changing optimizations. For example, large sprites and background images can be automatically diced up into smaller pieces. Duplicate dices and empty space is removed, potentially saving large amounts of atlas space.

- *Decreased build sizes with png textures*, 2D Toolkit allows us to significantly decrease the space needed to store your textures, by providing an easy one click configuration that tells Unity to store the images as PNG files.

- *Easy and flexible animation editor*, We may choose to create your animations using Unity's new 2D animation editor, or the animation editor provided by 2D Toolkit. We may even choose to mix and match, as each of these approaches provides certain advantages for various workflows and use cases.

- *Increase performance with sprite batching*, When creating backgrounds with many elements, 2D Toolkit lets we group these elements together so that they are drawn together, using less resources. This can significantly improve performance, especially on mobile devices.

2.2.2 HOTween

We decided to include HOTween asset because we need to build hundreds of animations within our application. And thanks to HOTween these animation management become easier.

Figure 2.4: HOTween logo

HOTween is a fast, type-safe object-oriented tween engine for Unity, compatible with all of Unity's scripting languages. [15]

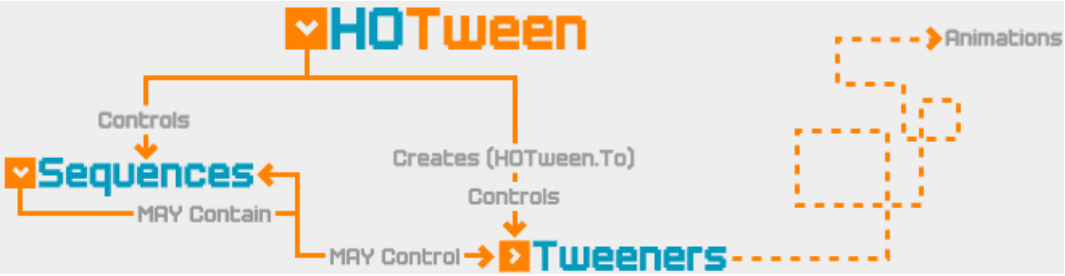

Figure 2.5: HOTWeen flow diagram

Also, HOTween works on Windows, Windows Phone 8, Windows 8 Store, Mac, iOS, Android.

2.2.3 iOs native plugin

iOs Native plugin, allow us to combine all native iOs features we need in one plugin, making it usage as easy as possible. Some of their principal features are the following: [16]

Figure 2.6: iOS native plugin logo

- In-App purchases support
- iCloud API management
- Game Center integration
- Video management
- iOS native events

2.2.4 Android plugin

Android Native plugin, provides the easy and flexible functionality of Android native functions, including in-app purchases, play service, advertising and native device API. Some of their principal features are the following: [17]

Figure 2.7: Android native plugin logo

- Play Service support
- Social networks integration
- Push Notifications
- Immersive Mode
- Android native events

2.3 Flurry analytics

Flurry analytics allow us to easily add analytics to our mobile game. Some of their principal features are the following: [5]

Figure 2.8: Flurry analytics plugin logo

- *Events*, Track in-app actions your users take and gain insight from how they are using your app. Understand and visualize usage trends, how users progress through the app and what events they are conducting with User Path analysis. Segment user actions by app version, usage, install date, age, gender, language, geography and acquisition channel.
- *Funnels*, 2Discover how your users progress through specific paths in your app. See where they are having issues and discover where those users who did not complete the process drop off. Leverage this insight to maximize the number of people who complete these paths.

- *Retention*, Measure user churn within your app. Understand the percentage of users that come back to your app to assess the vitality of your business. Layer on Segments to dive deep on specific user groups or acquisition channels.
- *Segments*, Analyze how different groups of app users vary in their usage and behavior. Build and layer segments across Usage, Retention, Funnels, and User Acquisition reporting to understand which set of users are most valuable to your business and what they are doing in your app.
- *Demographics*, Report out on users declared age and gender if you collect it from them. If not, utilize Flurry's machine learning and panel of 40 million devices to predict with accuracy you user's age and gender.
- *User Acquisition Analytics*, Monitor your user acquisition efforts and measure the impact of specific campaigns or channels on your user base, and therefore your business.

2.4 Summary

In this section, we have discussed the technologies related to the game we are developing.

Firstly, we took a look at Unity3D engine principal characteristics. We saw that Unity3D is a flexible and high-performance development platform used to make creative and intelligent interactive 3D and 2D experiences, which ensures developers to be able to publish to all of the most popular platforms with a single development process.

Later, we detailed Unity3D most remarkable features before explaining with more detail some important Unity3D concepts. Through these concepts we understand how Unity3D components work together. Moreover, we analyze Unity3D interface to know how the engine will be used to build the application.

Then, we explained how Unity3D Asset Store works and detailed the assets that we will be using during the game development.

Finally we took a look at Flurry Analytics, which covers the need of analyzing the application errors and metrics after its deployment.

CHAPTER 3

Requirement Analysis

This chapter describes one of the most important stages in software development: the requirement analysis using different scenarios. For this, a detailed analysis of the possible use cases is made using the Unified Modeling Language (UML). This language allows us to specify, build and document a system using graphic language.

3.1 Overview

The result of this chapter will be a complete specification of the requirements, which will be matched by each module in the design stage. This helps us also to focus on key aspects and take apart other less important functionalities that could be implemented in future works.

3.2 Use cases

These sections identify the use cases of the system. This helps us to obtain a complete specification of the uses of the system, and therefore define the complete list of requisites to match. First, we will present a list of the actors that are in the system and a UML diagram representing all the actors participating in the different use cases. This representation allows, apart from specifying the actors that interact in the system, the relationships between them.

These use cases will be described the next sections, including each one a table with their complete specification. Using these tables, we will be able to define the requirements to be established.

3.2.1 Actors dictionary

The list of primary and secondary actors is presented in table 3.1. These actors participate in the different use cases, which are presented later.

Actor identifier	Role	Description
ACT-1	Gamer	End user that plays the game using the physical instruments and the application
ACT-2	Instrument recognition algorithm	Algorithm that detects what physical figure has been placed on the application recognition zones
ACT-3	Client	Company that has outsourced the game development
ACT-4	Flurry	Technology that manages application metrics and errors

Table 3.1: Actors list

3.2.2 Game modes use case

This use case package collects the game play modes and their functionalities, as shown in 3.1.

The use cases presented in this section are as shown in the Figure 3.1:

- *play instrument* detailed in sub-section 3.2.2.1.
- *conduct orchestra* detailed in sub-section 3.2.2.2.
- *discover instrument* detailed in sub-section 3.2.2.3.
- *watch demo* detailed in sub-section 3.2.2.4.
- *select melody* detailed in sub-section 3.2.2.5.
- *select instrument* detailed in sub-section 3.2.2.6.

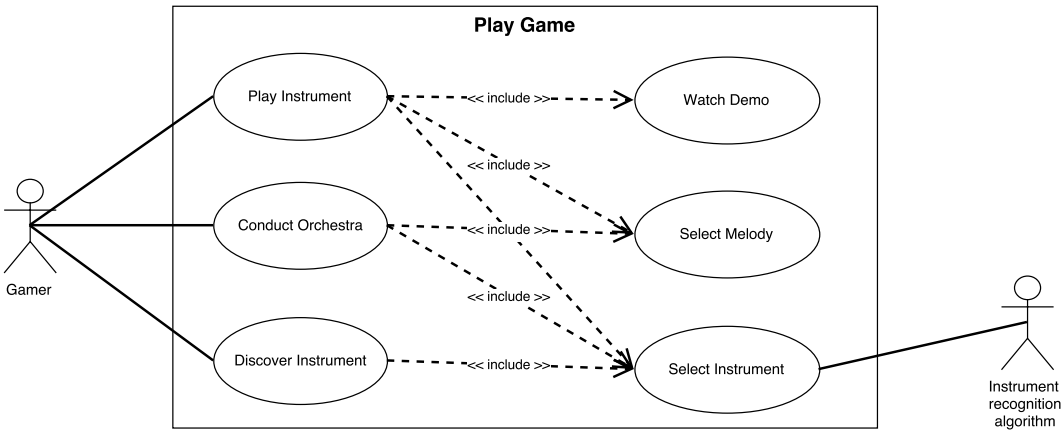

Figure 3.1: Game modes use case

3.2.2.1 Play instrument

Use Case Name	play instrument		
Use Case ID	UC1.1		
Primary Actor	Gamer		
Pre-Condition	The application is showing the home screen and the gamer has the instruments physical miniatures		
Post-Condition	Optionally, the gamer can watch a demo, change the instrument or change the melody		
Flow of Events		Actor Input	System Response
	1	The gamer puts an instrument miniature on home screen. The instrument is placed in the detection zone circle that represents the play instrument game mode	The application loads the play instrument mode game with the instrument that has been placed on the screen and the default melody.

3.2.2.2 Conduct orchestra

Use Case Name	conduct orchestra		
Use Case ID	UC1.2		
Primary Actor	Gamer		
Pre-Condition	The application is showing the home screen and the gamer has the instruments physical miniatures		
Post-Condition	Optionally, the gamer can play, stop or change the melody, enable or disable an instrument and change the instrument family		
Flow of Events		Actor Input	System Response
	1	The gamer puts an instrument miniature on home screen. The instrument is placed in the detection zone circle that represents the conduct orchestra game mode	The application loads the conduct orchestra game mode with all instruments enabled, the family instrument selector of the family instrument that has been placed on the screen enabled, and the default melody.

3.2.2.3 Discover instrument

Use Case Name	discover instrument		
Use Case ID	UC1.3		
Primary Actor	Gamer		
Pre-Condition	The application is showing the home screen and the gamer has the instruments physical miniatures		
Post-Condition	Optionally, the gamer can play the an instrument sound and change the instrument family		
Flow of Events		Actor Input	System Response
	1	The gamer puts an instrument miniature on home screen. The instrument is placed in the detection zone circle that represents the discover instrument game mode	The application loads the discover instrument game mode with the instruments of the family instrument that has been placed on the screen. The game mode shows instrument information and sounds

3.2.2.4 Watch demo

Use Case Name	watch demo		
Use Case ID	UC1.4		
Primary Actor	Gamer		
Pre-Condition	Play instrument game mode has been selected		
Post-Condition	-		
Flow of Events		Actor Input	System Response
	1	The gamer touches Demo button with a selected melody and instrument	The melody is played with the instrument selected and musical notes are highlighted

3.2.2.5 Select melody

Use Case Name	select melody		
Use Case ID	UC1.5		
Primary Actor	Gamer		
Pre-Condition	Play instrument game mode or conduct orchestra have been selected		
Post-Condition	-		
Flow of Events		Actor Input	System Response
	1	The gamer touch Melodies button	A list of the melodies are shown
	2	The gamer choses one of the available melodies	Selected melody is loaded into game mode

3.2.2.6 Select instrument

Use Case Name	select instrument		
Use Case ID	UC1.6		
Primary Actor	Gamer		
Secondary Actor	Instrument recognition algorithm		
Pre-Condition	Any game mode is selected and the gamer has the instruments physical miniatures		
Post-Condition	O-		
Flow of Events		Actor Input	System Response
	1	The Gamer place the instrument in the circle recognition zone	The Instrument recognition algorithm processes the instrument pad touches and determines which instrument have been placed
	2	The Instrument recognition algorithm detects a different instrument	Selected instrument is loaded into game mode

3.2.3 Game management use case

This use case package collects the main management use cases of the game, as shown in 3.2

- *metrics management* detailed in subsection 3.2.3.1
- *errors management* detailed in subsection 3.2.3.2

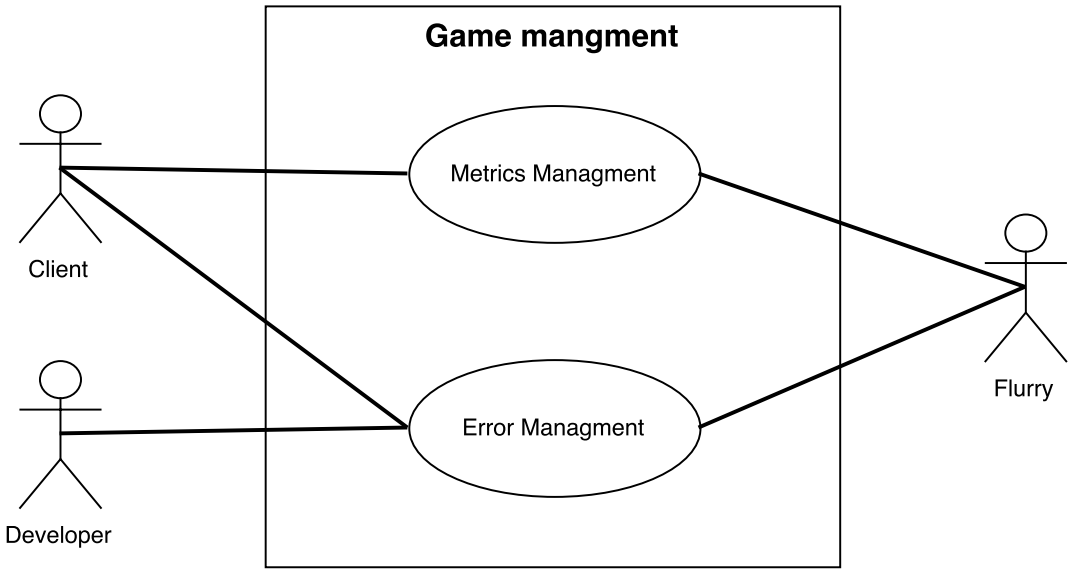

Figure 3.2: Game management use case

3.2.3.1 Metrics management

Use Case Name	Metrics management		
Use Case ID	UC2.1		
Primary Actor	Client		
Secondary Actor	Flurry		
Pre-Condition	Flurry client account has been created. Application is running		
Post-Condition	Metrics are added to Flurry servers		
Flow of Events		Actor Input	System Response
	1	The client access to their Flurry account	Flurry shows all metrics included with their SDK in the application

3.2.3.2 Errors management

Use Case Name	Metrics management		
Use Case ID	UC2.2		
Primary Actor	Client/Developer		
Secondary Actor	Flurry		
Pre-Condition	Flurry client and developer accounts has been created. Application is running and an error has been thrown		
Post-Condition	Errors are sent to Flurry servers		
Flow of Events		Actor Input	System Response
	1	The client and the developers accesses to their Flurry account	Flurry shows all errors included with their SDK in the application

3.2.4 Summary of requirements

After analyzing the previous use cases, some clear requirements seem to stand out. Many of the user cases requires similar features or certain characteristics in the architecture, pointing out the necessities or requirement of the system. In this section we will present those requirements, separated in two groups: functional and non-functional requirements.

3.2.4.1 Functional requirements

- **FR1:** *The Gamer* must be able to play three different game modes: *Play instrument*, *Conduct orchestra* and *Discover instrument*. We can observe this requirement in use case 3.2.2
- **FR2:** *The Gamer* must be able to watch a demo within *Play instrument* game mode. We can observe this requirement in use case 3.2.2
- **FR3:** *The Gamer* must be able to select melody within *Play instrument* and *Conduct orchestra* game modes. We can observe this requirement in use case 3.2.2

- **FR4:** *The Gamer* must be able to select instrument within *Play instrument*, *Conduct orchestra* and *Discover instrument* game modes. We can observe this requirement in use case 3.2.2
- **FR5:** *The instrument recognition algorithm* must detect the physical instrument miniature that has been placed on the game recognition zone. We can observe this requirement in use case 3.2.2
- **FR6:** *The client* must be able to access game metrics and errors. We can observe this requirement in use case 3.2.3
- **FR7:** *The developer* must be able to access game errors. We can observe this requirement in use case 3.2.3
- **FR8:** *Flurry* must be manage game metrics and errors. We can observe this requirement in use case 3.2.3

3.2.4.2 Non-functional requirements

- **NFR1:** Game application screens must be user-friendly. We can observe this requirement in use case 3.2.2
- **NFR2:** Melody and instrument selector modules must be reusable. We can observe this requirement in use case 3.2.2
- **NFR3:** Instrument recognition algorithm must be robust and must have a low response time. We can observe this requirement in use case 3.2.2
- **NFR4:** Game metrics and error management should be private and accessible for developers and the client. We can observe this requirement in use case 3.2.3

3.3 Summary

In this chapter, we detailed analysis of the possible use cases is made using the Unified Modeling Language (UML) for our application.

Firstly, we represented the actors dictionary, with both primary and secondary actors who will be participating in the different use cases.

Later, we described both Game modes and Game management use cases. With the use cases described we have introduced the basic functionalities that have been implemented

in this project. They will help us to understand how the different actors that can interact with our application. They can serve as a base for further development and different use cases that can come to mind.

Finally, we extracted both functional and non-functional requirements that were stood out after analyzing the use cases.

CHAPTER 4

Architecture

This chapter describes in depth how the system is structured in different modules and how the users interact with them and also how the modules interact with other modules by themselves.

4.1 Architecture Overview

In this section we will describe the **videogame architecture**, starting with its two main modules, the **physical instruments miniatures** and the **software application**. In Figure 4.1 we show the global game architecture identifying both main modules and their relation.

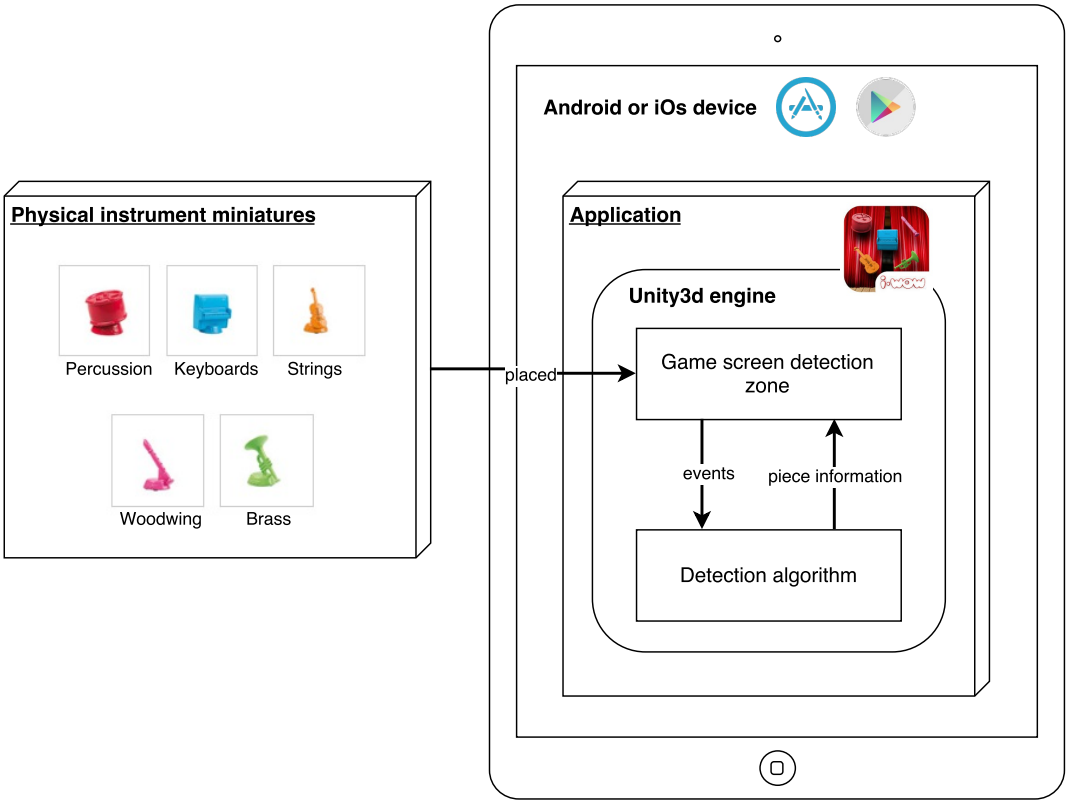

Figure 4.1: General Architecture

The modules are detailed below.

4.1.1 Physical instruments miniatures

There are five physical miniatures which represent each one of the musical instrument families used at the game: percussion, keyboards, strings, woodwind and brass. The gamer will use these pieces to interact with the app in order to change or activate the family represented by the piece. Also, pieces will be used to access to different game modes.

The interaction between the pieces and the app is haptic, that is, the gamer will pick a miniature and they will place it in the "instrument detection zone" represented by a shiny

circle in the app screen. The app "detection algorithm" will determine which piece has been placed and will make an event depending on the game mode and the app state. These events are, for example, changing instrument, activate instrument, etc.

Each piece base has three little pads which conform a triangle used to determine the instrument unequivocally.

Instrument miniatures design will be explain in more detail in section 4.2.

4.1.2 Application

The application has been developed using Unity, so we can identify the Unity components included on our app. These components are Unity Scenes, Unity Textures, Unity Assets, Unity Scripts and Unity Sounds. Integrating all of these components into our Unity projects we are able to build the game logic and its graphic interface.

Unity is notable for its ability to target games to multiple platforms. Within a project, we have control over delivery to different mobile devices, web browsers, desktops, and consoles. In our case, we need to develop both Android and iOs applications and Unity allow us to share the same code for them. Also, using Unity Android and iOs Plugins we are able to access to native Android and iOs SDKs, in case we need to access to some Android or iOs native components.

The application architecture will be explain in more detail in section 4.3.

4.2 Physical instruments miniatures

Besides the application software development, our videogame included five *physical instruments miniatures*. These pieces are used by the gamer to interact with the application.

The interaction is simple, the gamer place one of the physical pieces on one of the several recognition zones displayed within the game screens. Placing these pieces, the gamer is able to select or enable an instrument belonging to the family represented by the physical miniature. These pieces will be detailed in section 4.2.1.

The application has to determine what piece has the gamer placed on the screen. The responsible for that is the *detection algorithm* which will be explained in section 4.2.2.

4.2.1 Physical instrument pieces

We have five pieces, listed below:

- *Drum*, which represents the *percussion* instrument family, shown in Figure 4.2.
- *Piano*, which represents the *keyboards* instrument family, shown in Figure 4.3.
- *Violin*, which represents the *strings* instrument family, shown in Figure 4.4.
- *Flute*, which represents the *woodwind* instrument family, shown in Figure 4.5.
- *Trumpet*, which represents the *brass* instrument family, shown in Figure 4.6.

Figure 4.2: Drum piece (frontal and base)

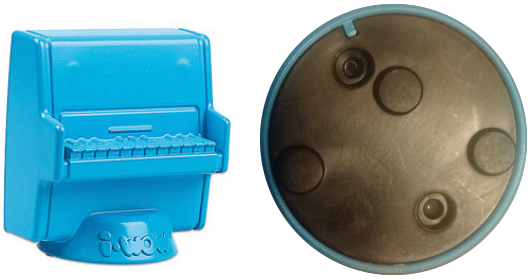

Figure 4.3: Piano piece (frontal and base)

Figure 4.4: Violin piece (frontal and base)

Figure 4.5: Flute piece (frontal and base)

Figure 4.6: Trumpet piece (frontal and base)

As we see in the figures above, each figure base has three little pads. These pads are used in order to determine which piece has the gamer placed on the screen. Each piece has

their pads placed in a certain position. A more detailed version of the piece bases are shown below:

Figure 4.7: Detail of the drum base

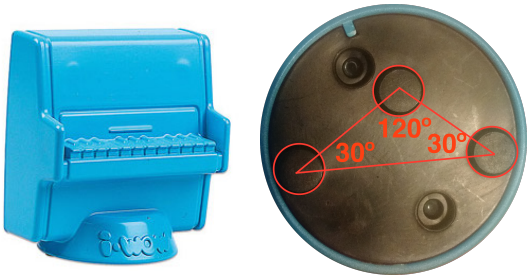

Figure 4.8: Detail of the piano base

Figure 4.9: Detail of the violin base

Figure 4.10: Detail of the flute base

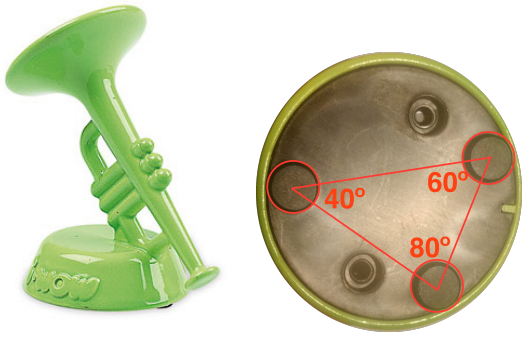

Figure 4.11: Detail of the trumpet base

As we can see in the above figures, the three pads located in the pieces bases create a triangle. Each piece creates a different triangle with the following angles:

- *Drum base*, whose triangle has three 60° angles, shown in Figure 4.7.
- *Piano base*, whose triangle has two 30° angles and one 120° angle and two 30° angles, shown in Figure 4.8.
- *Violin base*, whose triangle has two 45° angles and one 90° angle, shown in Figure 4.9.
- *Flute base*, whose triangle has one 30° angle and two 75° angles, shown in Figure 4.10.
- *Trumpet base*, whose triangle has one 40° angle, one 60° angle and one 80° angle, shown in Figure 4.11.

Using these angles information the recognition algorithm will be able to determine the placed piece.

4.2.2 Recognition algorithm

As we have detailed in the previous section, each piece is defined by the triangles on their base. As we know, each triangle is defined by their three angles.

In order to determine the placed piece, we have to retrieve the three instrument pads positions after the gamer places the piece on the recognition zone. This information will be the input of the algorithm that we have developed to determine the placed musical instrument.

We can see the recognition algorithm in Figure 4.12.

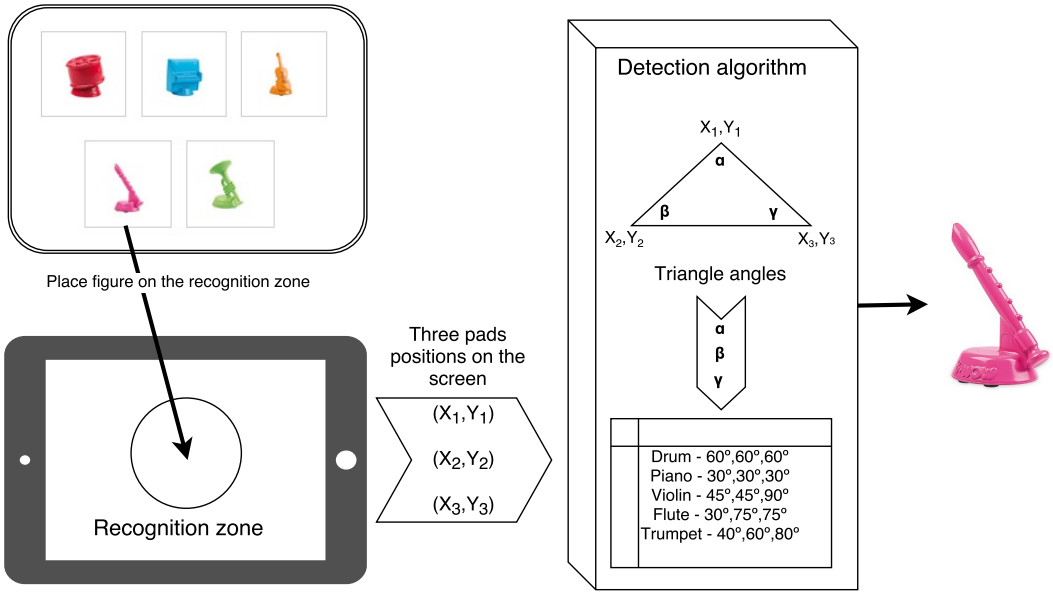

Figure 4.12: Detection algorithm diagram

If we look at 4.12, we can see how the piece if recognized by the application. This recognition could be divided within three steps:

1. The gamer has to place the piece on the recognition zone. When the three pads located in the piece base touch the screen, these pads coordinates $\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{pmatrix}$ are sent to the detection algorithm.

2. The detection algorithm calculate the angles of a triangle build using the three coordinates given by the piece base.

3. The detection algorithm compare the calculated triangle with the figure's defined triangles. After it, we retrieve the piece whose base triangle match the calculated one.

We can see the algorithm design below:

Data:
dictionaryAnglesInstruments is a {key,value} dictionary where the keys are the instruments and the values are their triangle base angles → $Dict\{piece, (\alpha_p, \beta_p, \gamma_p)\}$ where $\alpha_p, \beta_p, \gamma_p \in [0, \pi]$ and p is *key index*
instrumentCoordinates is a 3-tuple y-sorted coordinates made by the instrument base on the screen → $[A(x_1, y_1), B(x_2, y_2), C(x_3, y_3)]$ where $(x_i, y_i) \in \mathbb{R}_2$
Result:
Instrument recognized ∈ {Drum,Piano,Violin,Flute,Trumpet}

while *piece is placed on the recognition zone* **do**

```
// Obtain the triangle sides module
```
$a \leftarrow \sqrt{(x_3 - x_2)^2 + (x_3 - x_2)^2}$;
$b \leftarrow \sqrt{(x_3 - x_1)^2 + (x_3 - x_1)^2}$;
$c \leftarrow \sqrt{(x_2 - x_1)^2 + (x_2 - x_1)^2}$;
```
// Given the triangle sides we obtain the angles
```
$\alpha \leftarrow arcos(\frac{b^2+c^2-a^2}{2bc})$;
$\beta \leftarrow arcos(\frac{c^2+a^2-b^2}{2ca})$;
$\gamma \leftarrow arcos(\frac{a^2+b^2-c^2}{2ab})$;
```
// Given the triangle angles we subtract them from the
   dictionaryAnglesInstruments values
```
foreach *element* $(\alpha_p, \beta_p, \gamma_p)$ *in dictionaryAnglesInstruments* **do**
$difference_p \leftarrow |\alpha_p - \alpha| + |\beta_p - \beta| + |\gamma_p - \gamma|$;
```
// We return the instrument whose angle difference is the
   lowest
```
$instrument\_index \leftarrow index\ min(difference)$;
return $instrumentCoordinates[index]\ key$;

end

Algorithm 4.1: Instrument recognition algorithm

4.3 Application

The application is the biggest module of the game. It includes the whole software development as it is shown in Figure 4.13.

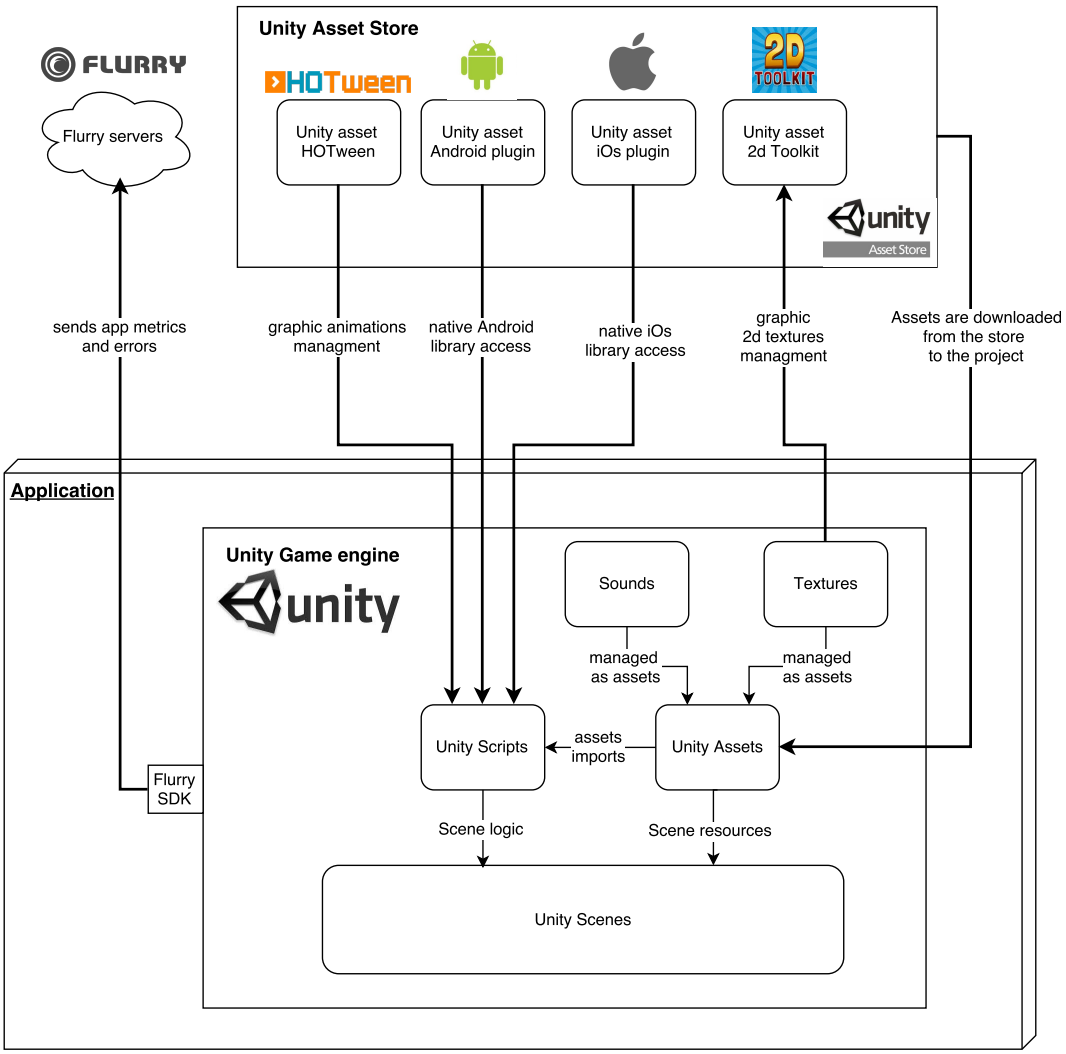

Figure 4.13: Application architecture diagram

If we look at the application architecture diagram we can see the Application module where Unity game engine will run. Unity has three main components, Unity Assets, Unity Scripts and Unity Scenes. These components and Unity interaction with Flurry are described below:

4.3.1 Assets

An asset is a representation of any item that can be used in your game or project. An asset may come from a file created outside of Unity, such as a 3D model, an audio file, an image, or any of the other types of file that Unity supports. There are also some asset types that can be created within Unity, such as an Animator Controller, an Audio Mixer or a Texture Managers. In other words, assets are any resource your game uses.

Thankfully, Unity's asset importing is robust and intelligent, it will accept all popular 3D file formats and also supports all common image file formats, including PNG, JPEG, TIFF and even layered PSD files directly from Photoshop. When it comes to audio, Unity supports WAV and AIF, ideal for sound effects, and MP3 and OGG for music.

In Figure 4.13 we can see that sounds and textures are included as assets to use them, but they are not the only assets we will use. We said that there are some assets types that have been created within Unity to make development easier. In our case we will use an Animation Controller called HOTween, a 2D Texture Manager called 2dToolkit and both Android and iOs plugins. All these assets are included in our Unity project downloading them from Unity Asset Store.

The Unity Asset Store is where a growing library of free and commercial assets are placed. These assets are created both by Unity Technologies and also members of the community. A wide variety of assets is available, covering everything from textures, models and animations to whole project examples, tutorials and Editor extensions. These assets are accessed from a simple interface built into the Unity Editor and are downloaded and imported directly into your project.

Assets will be used from the Scenes and/or the Scripts, which are detailed in section 4.3.3.

4.3.2 Scenes

Scenes contain the objects of your game. They can be used to create a main menu, individual levels, and anything else. Each unique Scene file as a unique level, where you will place your environments, obstacles, and decorations, essentially designing and building your game in pieces.

We can easily make an analogy between Scenes and screens in our app. Each screen is built from a Scene where all the Assets logic are managed by the Scripts.

Creating Scenes with Unity are possible thanks to their intuitive interface, where project assets can be drag to the interface Scene and Scripts can be attached to the assets to control them.

4.3.3 Scripts

Scripts, known in Unity as behaviors, let you take assets in your scene and make them interactive. Multiple scripts can be attached to a single object, allowing for easy code reuse. Unity supports three different programming languages; UnityScript, C#, and Boo. In our project we will use C#.

As we can see in Figure 4.13, Scripts will manage Unity Assets and will use external Assets from the Unity Asset Store as libraries to make the development easier. In our case, HOTween asset allow us to automate the animation of any numeric (and some non-numeric) property or field (numbers, vectors, transforms, and so on) in many different ways. 2dToolkit provide an efficient 2D sprite, collider set-up and text system which integrates seamlessly into the Unity environment. Android and iOs plugins allow us to access to native Android and iOs libraries.

As we said, scripts will be attached to the scene assets we need to provide them the functionality we need to.

4.3.4 Flurry

Flurry analytics allow us to easily add analytics to our mobile game applications. Integrating this technology within our application will allow us to retrieve game application metrics such as volume of users, time of use or crash reports.

As we can see in Figure 4.13, in order to integrate it into our development we have to add Flurry SDK into our Unity project. Before that we just have to create a project using Flurry web application. After all this process we will be able to use flurry library to manage all the components we want to. In our case we will retrieve the following metrics:

- *Active users*, the total number of unique users who accessed the application per day.
- *New users*, the total number of unique users who used the application for the first time per day.
- *Average session length*, average length of a user session per day.

- *Crash analytics*, information about the application crashes, exceptions and errors.

4.4 Application use workflow

The application has three game modes, for each one we will see the application use workflow to get a more precise idea of how the Gamer will interact with the application.

The three game modes were designed as a result of the Game modes use case defined in section 3.2.2:

- *Playing instrument game mode* detailed in sub-section 4.4.1.
- *Conducting orchestra game mode* detailed in sub-section 4.4.2.
- *Discovering instrument game mode* detailed in sub-section 4.4.3.

4.4.1 Playing instrument game mode

Playing instrument game mode workflow is represented in Figure 4.14

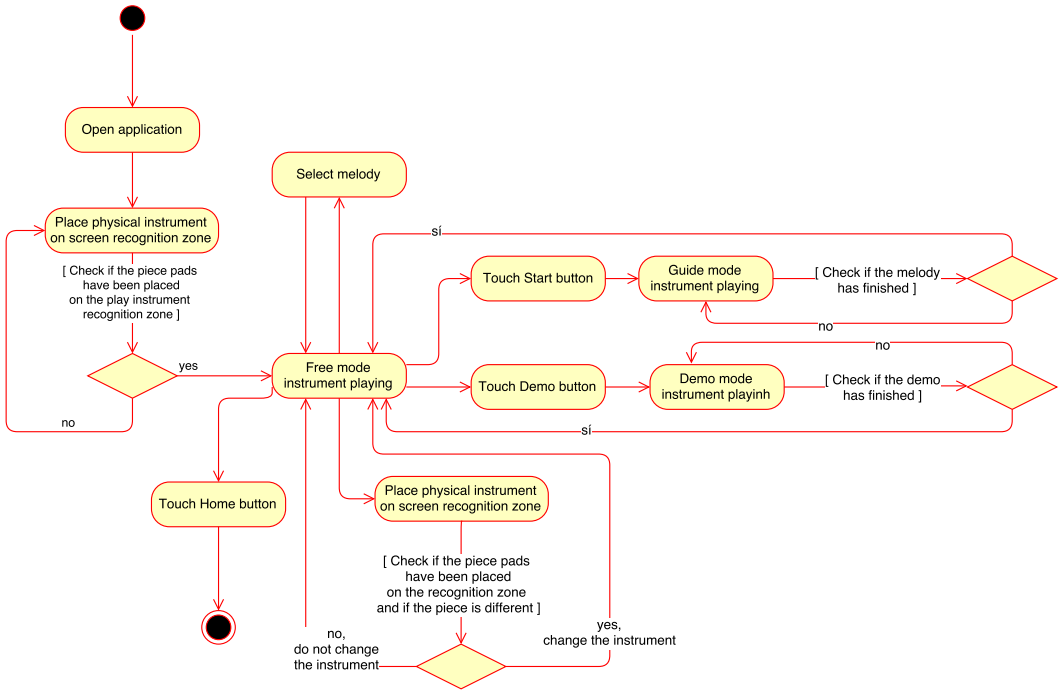

Figure 4.14: Playing instrument game mode

As we can see in Figure 4.14, to access *Playing instrument game mode*, the gamer has to place the physical instrument miniature in the Playing mode recognition zone after opening the application. If the instrument has been placed properly, the *Free mode instrument play* screen is opened.

Within this *Free mode instrument play* screen, the gamer can play freely with the instrument that has been placed to access to this game mode. Also, the gamer is able to change the instrument by placing another instrument miniature on the instrument recognition zone.

The gamer has the possibility to watch a demo. After touching the demo button, the selected melody is played with the selected instrument. Also, the gamer can change the melody using the melody selector menu, that is shown after touching the melody button.

Also, the gamer can play the selected melody with the selected instrument on a *guide mode instrument playing*. This guide mode is started after touching the start button. Within this guide playing, the notes are highlighted and the gamer has to play the highlighted note to compose the whole melody.

Finally, the gamer can go back to the home screen touching the home button.

4.4.2 Conducting orchestra game mode

Conducting orchestra game mode workflow is represented in Figure 4.15

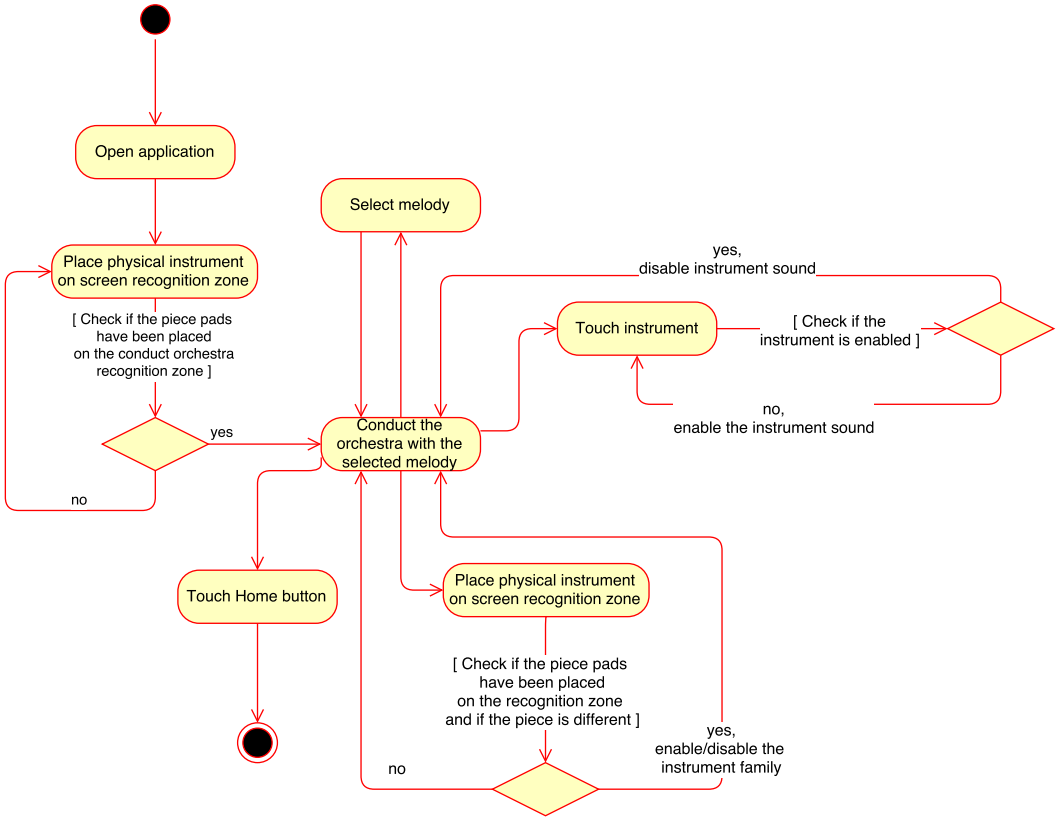

Figure 4.15: Conducting orchestra game mode

As we can see in Figure 4.15, to access *Discovering instrument*, the gamer has to place the physical instrument miniature in the Discovering mode recognition zone after opening the application. If the instrument has been placed properly, the *Discovering instrument* screen is opened.

Within this *Discovering instrument* screen, the gamer can read information and learn about some instruments of the family instrument that has been placed to access the game mode. Also, the instrument sound can be reproduce touching the instrument sound button.

Also, the gamer is able to change the instrument by placing another instrument miniature on the instrument recognition zone.

Finally, the gamer can go back to the home screen touching the home button.

4.4.3 Discovering instrument game mode

Discovering instrument game mode workflow is represented in Figure 4.16

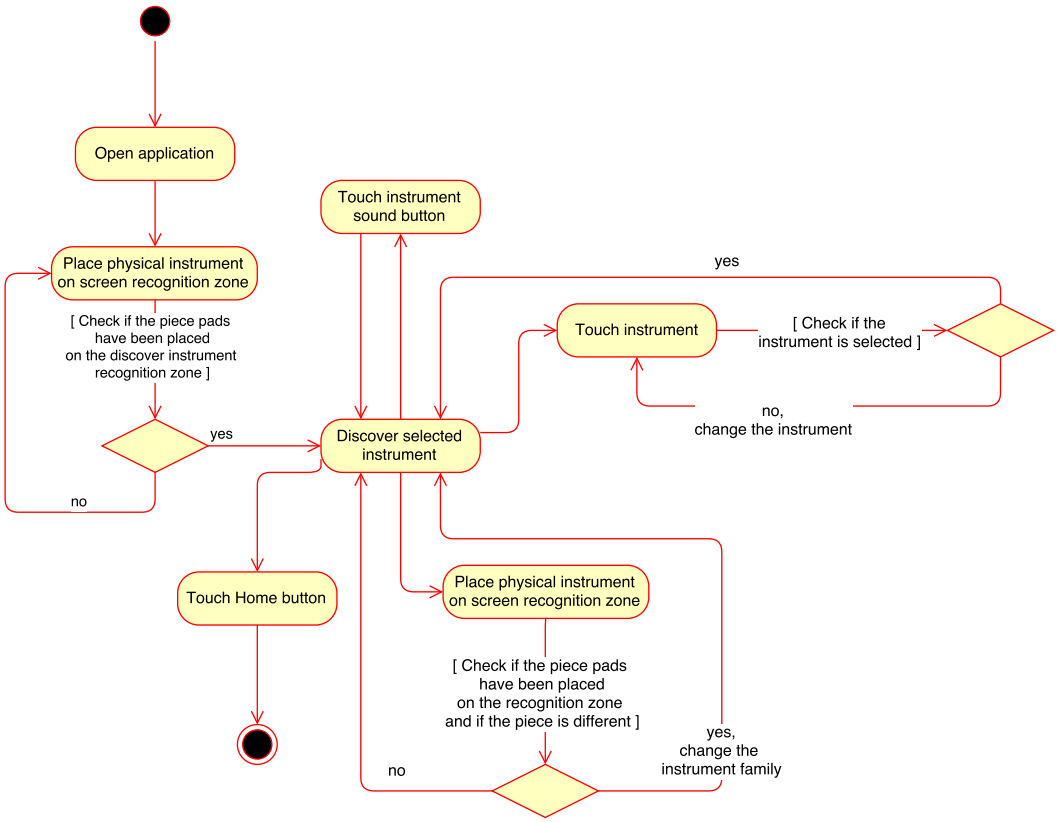

Figure 4.16: Discovering instrument game mode

As we can see in Figure 4.15, to access *Conducting the orchestra*, the gamer has to place the physical instrument miniature in the Conducting mode recognition zone after opening the application. If the instrument has been placed properly, the *Conducting orchestra* screen is opened.

Within this *Conducting orchestra* screen, the gamer can conduct an orchestra which is playing the selected melody. This melody can be changed by the gamer using the melody selector menu, that is shown after touching the melody button.

When this game mode is opened, the melody start to be played with all the instrument enabled. The gamer is able to enable or disable an instrument sound. Also, the gamer can enable or disable an entire family instrument placing one of the physical instrument miniatures in the instrument recognition zone.

4.5 Summary

In this chapter, we presented the proposed architecture for our application.

We started by taking a look at the architecture overview, where we differentiated two principal components, the physical instruments miniatures and the software application.

After introducing both principal components, we saw how these physical instruments miniatures were designed and how they are recognized by the software application using a recognition algorithm.

Later, we detailed the software application architecture, focusing on how Unity3D engine and its assets as long as other components work together.

Finally, we described the three application game modes workflow.

CHAPTER 5

Case study

In this chapter we are going to describe two selected use cases that represents how the gamer will interact with the application. Firstly, we will describe the gamer interaction with the playing instrument game mode. Secondly, we will study the conduct orchestra game mode

5.1 Introduction

In the following sections we will explain two of the principal application game modes:

- *Playing instrument game mode* detailed in section 5.2.
- *Conducting orchestra game mode* detailed in section 5.3.

In both use cases, two actors are involved, the gamer and the instrument recognition algorithm.

Actor identifier	**Role**	**Description**
ACT-1	Gamer	End user that plays the game using the physical instruments and the application
ACT-2	Instrument recognition algorithm	Algorithm that detects which physical figure has been placed on the application recognition zones

Table 5.1: Actors list

The *Gamer* is able to access to both game modes from the application home screen, using one of the physical instrument miniature. Within each game mode, the gamer is able to interact with every component in order to select other musical instrument, change melodies, play an instrument, watch a play instrument demo, choose what instruments must be playing, etc.

The *Instrument recognition algorithm* allows the application engine to detect which physical instrument miniature has been placed on the screen recognition zones. As a result, the gamer is able to interact with the game mode using the physical instrument miniatures.

5.2 Playing instrument game mode

In this section we are going to detail the whole application flow within the *Playing instrument game mode*. This game mode has been detailed in the application workflow, represented in section 4.4.1.

Firstly, the *Gamer* has to open the application that has been previously installed on their device from either *Android Play Store* or *iOs App Store*. Also, the gamer must has purchased the physical instrument miniatures. These miniatures are necessary to interact with the application so that we can access to different game modes and change instruments inside them.

After opening the application the home screen is loaded. We can see game home screen in Figure 5.1.

Figure 5.1: Application Home Screen

By interacting with this home screen we have the possibility to access to each of the three game modes availables. In case it is our first contact with the game and we do not know how to interact with it, we can get some help touching the help button, which is represented with a question mark at the top left corner of the home screen. This help screen is shown in Figure 5.2.

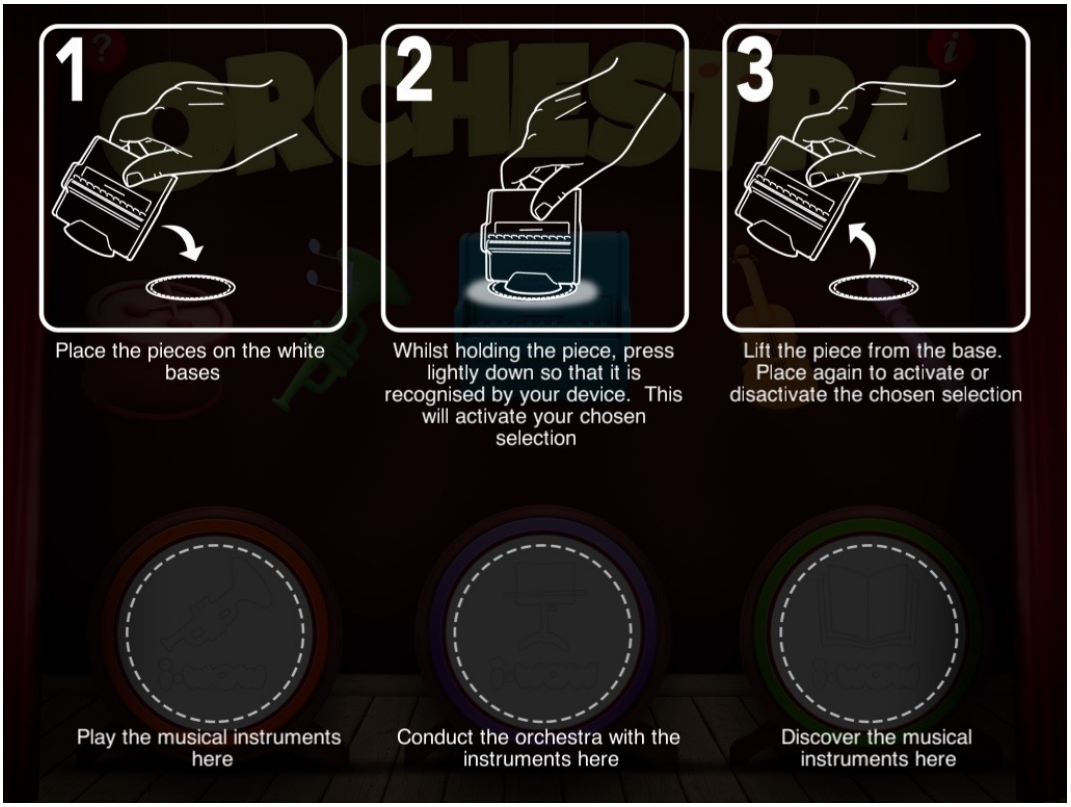

Figure 5.2: Help Home Screen

As we can see in Figure 5.2, in order to access to one of the three game modes, we should place one of the physical instrument miniature (from now on piece) on one of the instrument recognition zones (from now on white bases). After placing it, we just have to hold and press lightly down the piece so that the application can recognize it. This recognition is processed within the *Instrument recognition algorithm*. After it, we can lift the piece from the white base and place another one if we want to access to another game mode after getting back to the home screen.

In this use case, we want to access to the *Playing instrument game mode*, so we choose one of the pieces and place it on the left white base so that we can access to the *Playing instrument game mode*. In our case, we have chosen the percussion piece. After situating the percussion piece on the left white base, a new screen, shown in Figure 5.3, is opened.

Figure 5.3: Xylophone playing instrument game mode

In the *Playing instrument game mode* screen shown in Figure 5.3 we can see two buttons, the *Home* button and the *Help* button, at the top right of the screen. These buttons allow us to get to the home application screen or to show the help information for this game mode respectively.

In Figure 5.4 we can take a look at the help screen information.

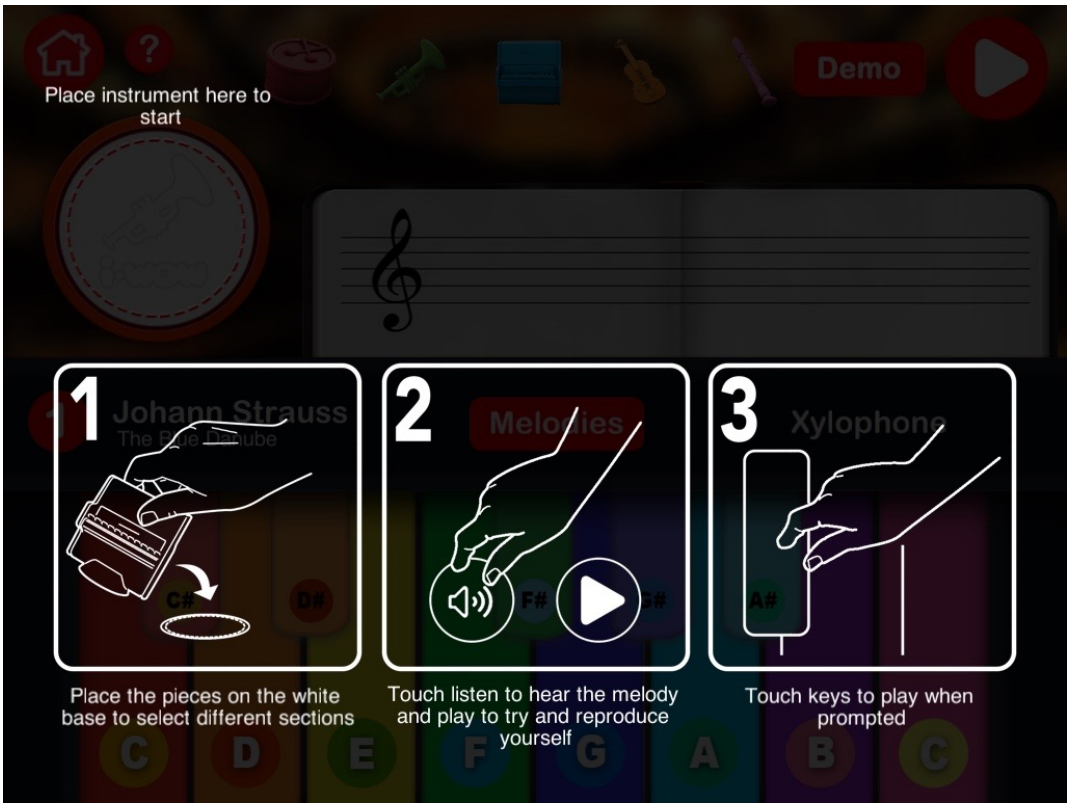

Figure 5.4: Help information playing instrument game mode

As we can read in the help information display, we can put another instrument on the white base, located in the left of the screen, in order to select another instrument to play with. The instruments available in this game mode are the following:

- *Xylophone*, which is selected after placing the *percussion* family piece.
- *Piano*, which is selected after placing the *keyboards* family piece.
- *Harp*, which is selected after placing the *strings* family piece.
- *Panpipes*, which is selected after placing the *woodwind* family piece.
- *Trombone*, which is selected after placing the *brass* family piece.

Besides choosing an instrument we can choose the melody we are going to play. We can choose it after touching the *Melodies* button placed in the center of the screen. All the melodies available in this game mode are shown in Figure 5.5.

Figure 5.5: Melodies selection instrument game mode

When we have decided which instrument we will play which melody with, we can touch the *Demo* button, which is positioned in the top right of the screen. By touching the *Demo* button the melody will be played automatically and the musical notes will be appearing at the sheet music placed in the center of the screen.

After watching the *Demo* we are ready to play the melody with the selected instrument. We can start playing the melody by touching the *Play* button located at the top right of the screen. After touching the *Play* button, we have to touch the instrument keys when prompted so that we can play the whole melody. In Figure 5.6 we can see that the key which have to be touch is the *E* key, the one that is prompted. Also, we can stop the *Play mode* touching the *Stop* button.

Figure 5.6: Playing instrument game mode

Before concluding this game mode use case, we can see the rest of instrument screens in the following figures:

- *Xylophone*, shown in Figure 5.3.
- *Piano*, shown in Figure 5.7.
- *Harp*, shown in Figure 5.8.
- *Panpipes*, shown in Figure 5.9.
- *Trombone*, shown in Figure 5.10.

Figure 5.7: Playing piano screen

Figure 5.8: Playing harp screen

Figure 5.9: Playing panpipes screen

Figure 5.10: Playing trombone screen

5.3 Conducting orchestra game mode

In this section we are going to detail the whole application flow within the *Conducting orchestra game mode*. This game mode has been detailed in the application workflow represented in section 4.4.2.

In order to access to this game mode, we have to proceed just as we have detailed in the previous game mode in section 5.2.

In this use case, we want to access to the *Conducting orchestra game mode*, so we choose one of the pieces and place it on the center white base so that we can access to the *Conducting orchestra game mode*. After situating one of the pieces on the left white base, a new screen, shown in Figure 5.11, is opened.

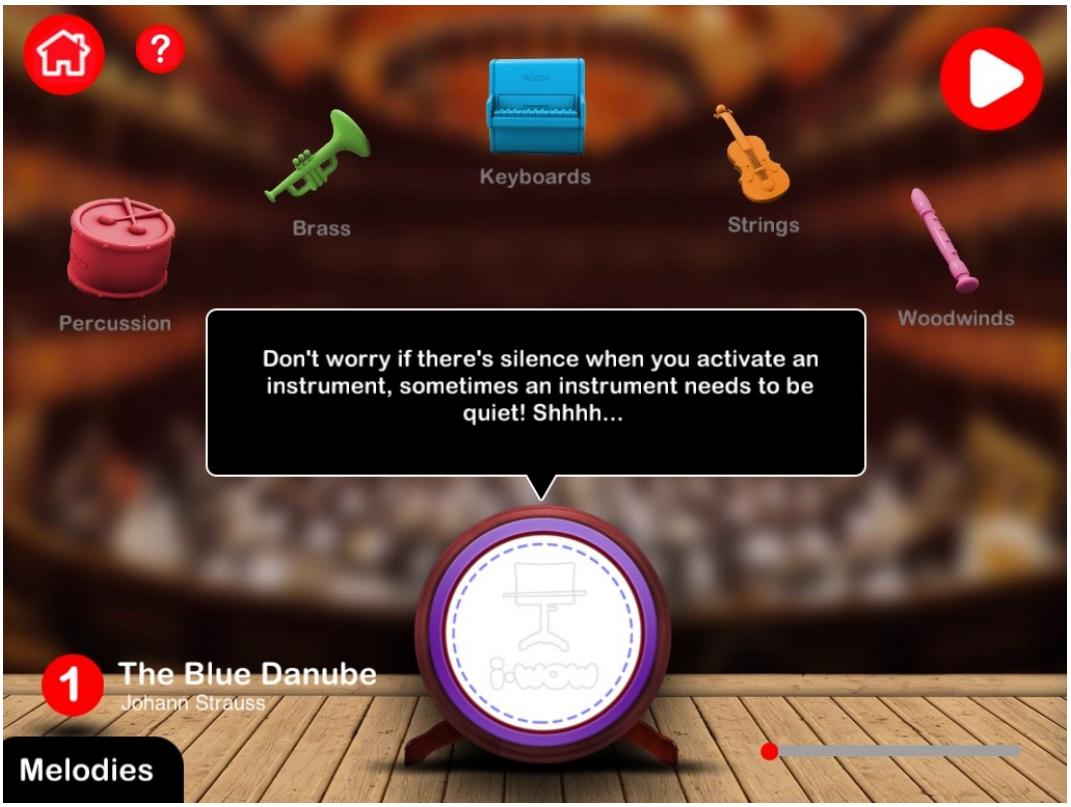

Figure 5.11: Conducting game mode access screen

In the *Conducting orchestra game mode* screen shown in Figure 5.11 we can see two buttons, the *Home* button and the *Help* button, at the top right of the screen. These buttons allow us to get to the home application screen or to show the help information for this game mode respectively.

In Figure 5.12 we can take a look at the help information.

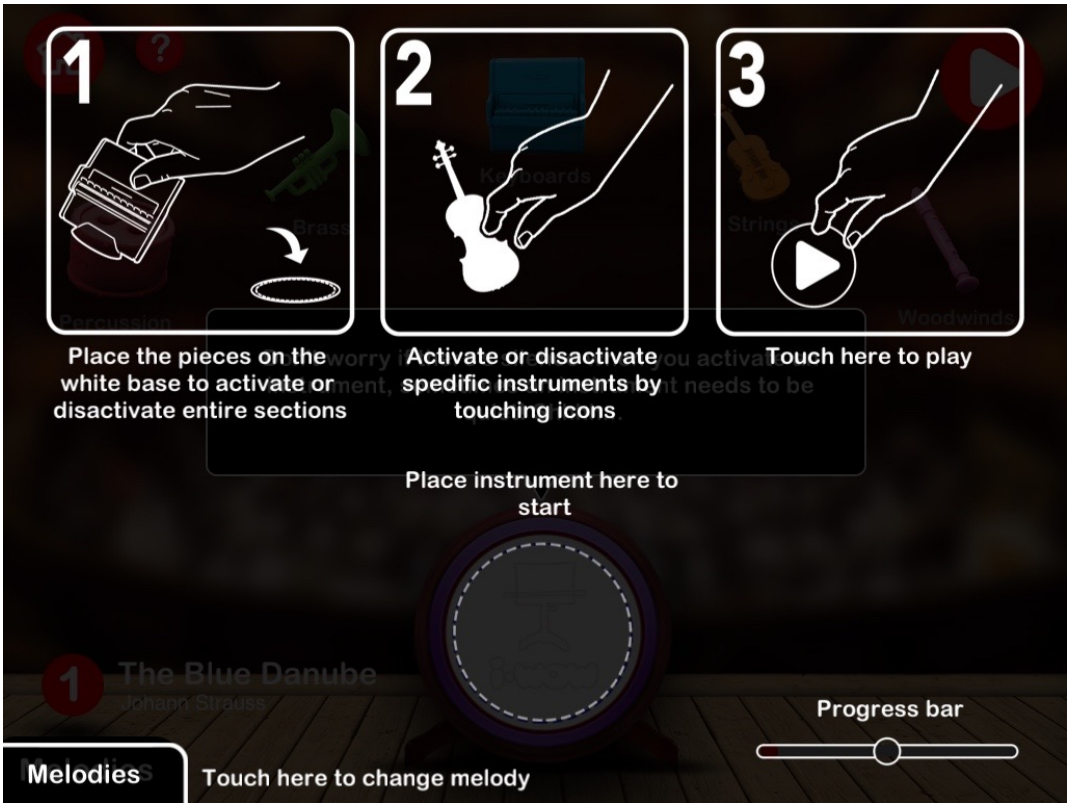

Figure 5.12: Help information conducting orchestra game mode

As we can read in the help information display, we should place the instrument miniatures on the white base, located in the center of the screen, in order to activate or deactivate an instrument family section. After activating one section, we will be able to activate or deactivate the instrument belonging to the related instrument section. After choosing the what instrument we want to be activated we can touch the *Play* button to start the selected melody.

When an instrument is activated, their sound will be played. So, by activating and deactivating we can conduct the musical instruments that are playing the selected melody.

Besides conducting the orchestra, we can choose the melody played by the orchestra touching the *Melodies* button placed in the bottom left of the screen. The melodies available in this game mode are shown in Figure 5.13.

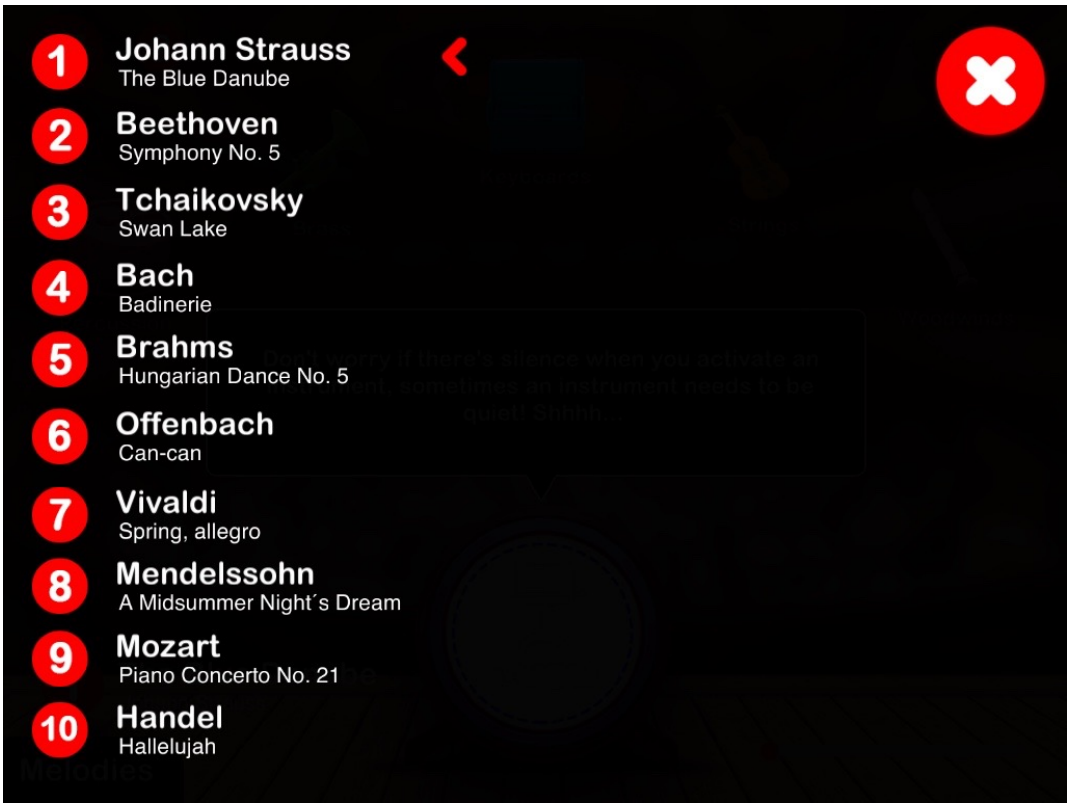

Figure 5.13: Melodies selection in the conducting orchestra game mode

In Figure 5.14 we can see the *Conducting orchestra game mode* screen where *The Blue Danube* melody has been selected and all the instrument have been activated.

As we can see, there is one section for each instrument miniature that we have. The relation between the pieces and the sections is described below:

- *Percussion section*, which is activated after placing the *percussion* family piece.
- *Brass section*, which is activated after placing the *brass* family piece.
- *Keyboards section*, which is activated after placing the *keyboards* family piece.
- *Strings section*, which is activated after placing the *strings* family piece.
- *Woodwings section*, which is activated after placing the *woodwind* family piece.

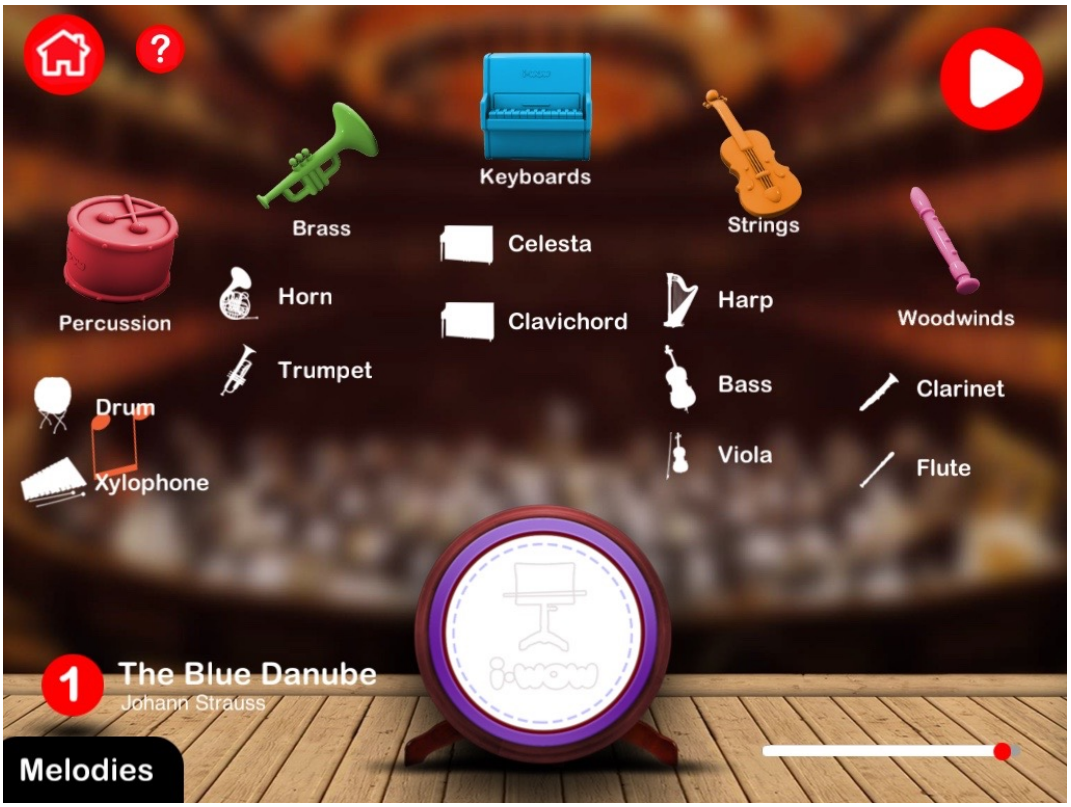

Figure 5.14: Conducting orchestra screen with all instruments activated

Moreover, the instruments that we can activate or deactivate within the above sections, are list below:

- *Percussion section*: Drum and Xylophone.
- *Brass section*: Horn and Trumpet.
- *Keyboards section*: Celesta and Clavicord.
- *Strings section*: Harp, Brass and Viola.
- *Woodwings section*: Clarinet and FLute.

To end this use case, in Figure 5.15 we can see the *Conducting orchestra game mode* screen where *The Blue Danube* melody is being played by the *Xylophone*, the *Celesta*, the *Clavichord*, the *Viola* and the *Harp*, which are the instruments that have been activated.

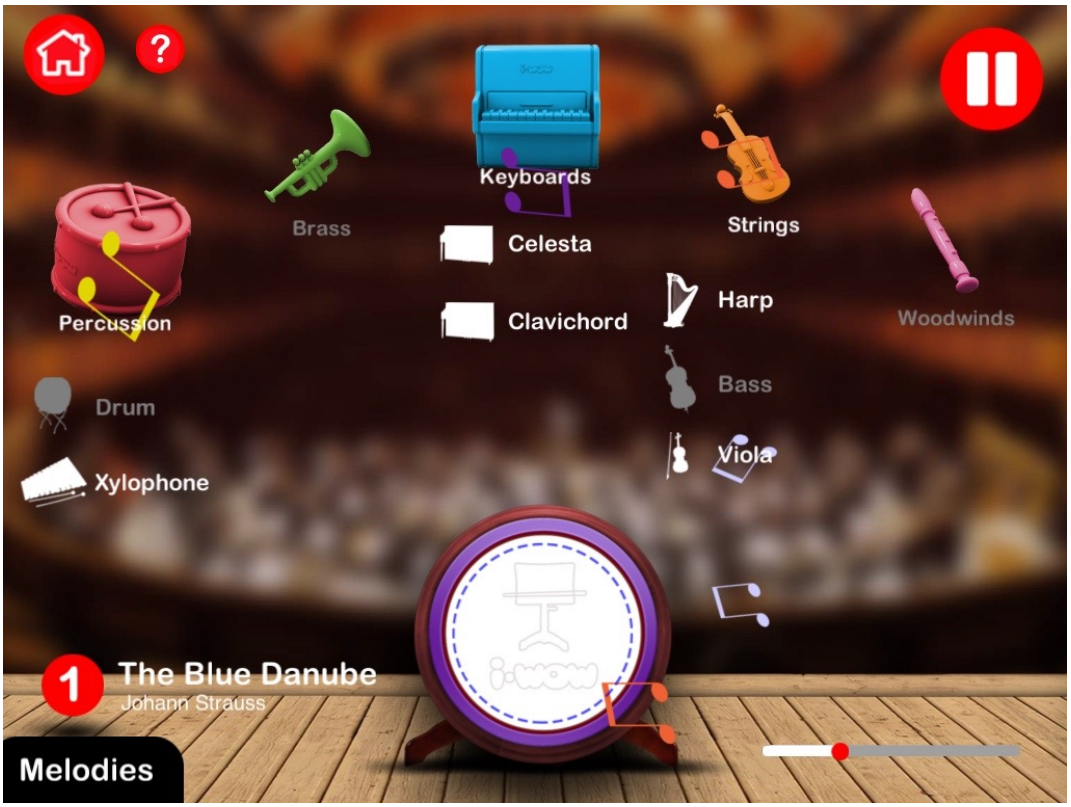

Figure 5.15: Conducting orchestra screen with some instruments activated

5.4 Summary

In this chapter, we detailed both *Playing instrument* and *Playing instrument* game modes.

Firstly, we defined the actors involved within both cases of study.

After that, we shown the game mode screen flow and hoe the gamer should interact with this game mode.

CHAPTER 6

Evaluation

In this chapter we will evaluate the game application through the information retrieved after its deployment

6.1 Overview

In the following sections we will observe the impact that our application has taken after its deployment. We will analyze both Android and iOs application that we have deployed to Google Play and Apple Store respectively.

The metrics used in this chapter have been recovered using *Game Analytics*. Although ideally we should have used Flurry to retrieved the desired metrics to write this chapter, the client restricted Flurry panel access two years after the application deployment.

Game Analytics is a free and powerful analytics tool for game developers, that helps us to understand player behavior and build better games. [18] It is natively included within Unity so we are able to access to lots of metrics out of the box.

6.2 Acquisition

In this section we are going to revise the user acquisition. The most important metrics for user acquisition are number and location of installations. These two metrics are by far the easiest way to tell if our application is something that people find valuable.

We have to notice that although our application is free to download, the pack with the physical instrument pieces should be bought in the client physical stores. This physical pack is shown in Figure 6.1

Figure 6.1: Physical instrument pieces box

This will impact in the application user acquisition due to the fact that our target users

will be the ones who have bought the physical game.

We will study the acquisition metrics in two time periods. Firstly, we will look at the information obtained during the first year since the application was first released, this covers the period between August 2014 and August 2015. Then we will observe the information during the three years that the application has been available on the market which covers the period between August 2014 and February 2017.

6.2.1 Number of users

As we said, one of the most important metrics for user acquisition is the number of installations or the number of users which have download our application in their device.

In Figure 6.2 we can see the new users engaged in the first year:

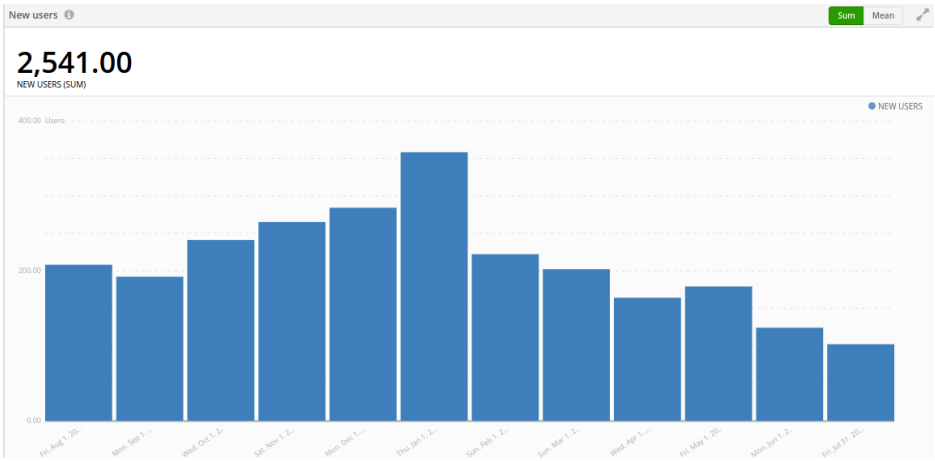

Figure 6.2: New users in the first year from release

As wee can see, we got 2541 users in the first year. Every month the users increased an average of 200 new people. Also, we can observe that there is an evident increase of new users in the months of December and January, which is consequence of the increase of the physical packs sells within Christmas days.

In Figure 6.3 we can see the new users obtained in the whole time our application have been on the market:

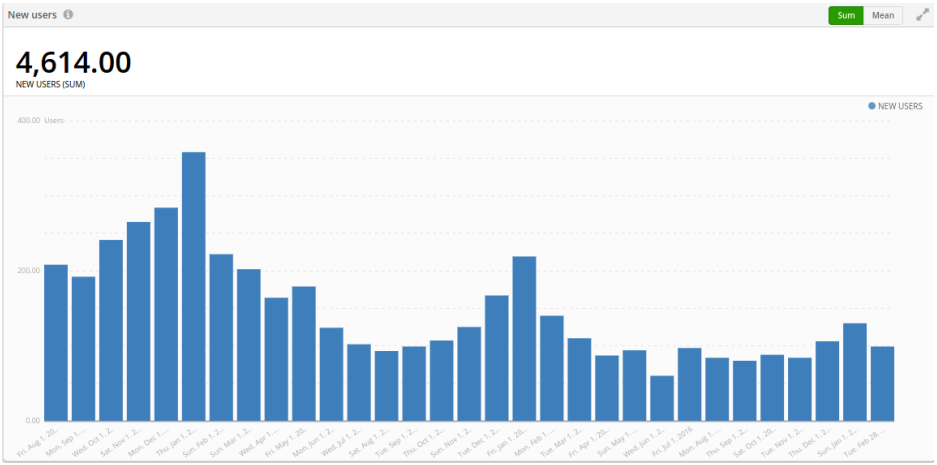

Figure 6.3: New users since application first release

As wee can see, we got 4641 users since our application first release. Also, we can observe that after first year new installations have been decreasing with the exception of the Christmas months described earlier. Even so, during 2016 our game has been constantly downloaded by an average of 100 new users.

6.2.2 Users location

Due to the physical instrument pack is sold in the countries list in the Table 6.1, is very useful to observe new user distribution across the world.

Argentina	Bulgaria	Colombia	Greece	Hungary
Israel	Latvia	Mexico	Poland	Qatar
Romania	Saudi Arabia	Switzerland	United Arab Emirates	Uruguay
Azerbaijan	China	France	Holland	
Italy	Lithuania	Peru	Portugal	
Russia	Spain	Turkey	United States	

Table 6.1: Countries were the physical instrument pack is sold

In Figure 6.4 we can see the world distribution map of users engaged in the first year:

Also, in Figure 6.5 we can observe the top four countries where the game have been downloaded within this period:

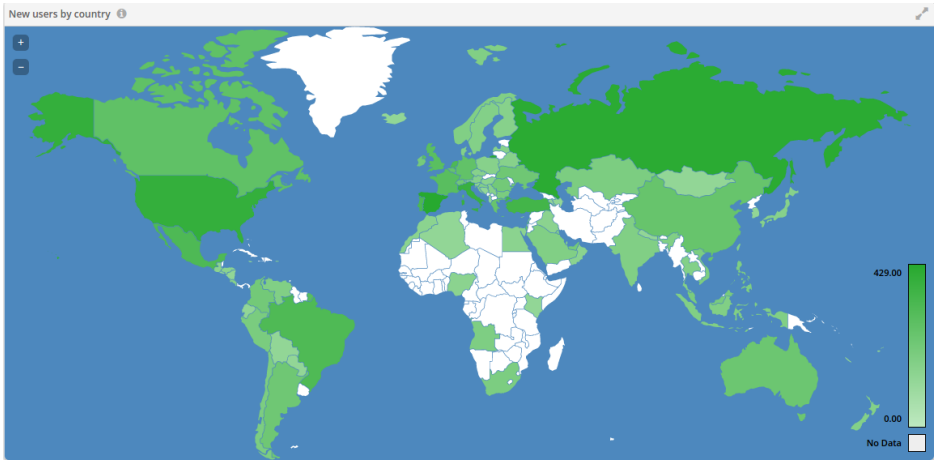

Figure 6.4: New users location in the first year from release

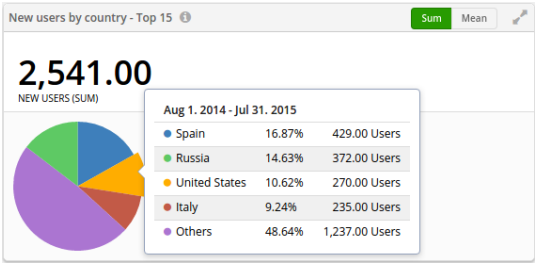

Figure 6.5: New users location in the first year from release top countries

As we can see, the application have been installed from most of the countries located in Europe and America, as long as many countries located in Asia. Also, Spain, Russia, USA and Italy are the countries where most of the users came from.

In Figure 6.6 we can see the world distribution map of users obtained in the whole time our application have been on the market:

Also, in Figure 6.7 we can observe the top four countries where the game have been downloaded within this period:

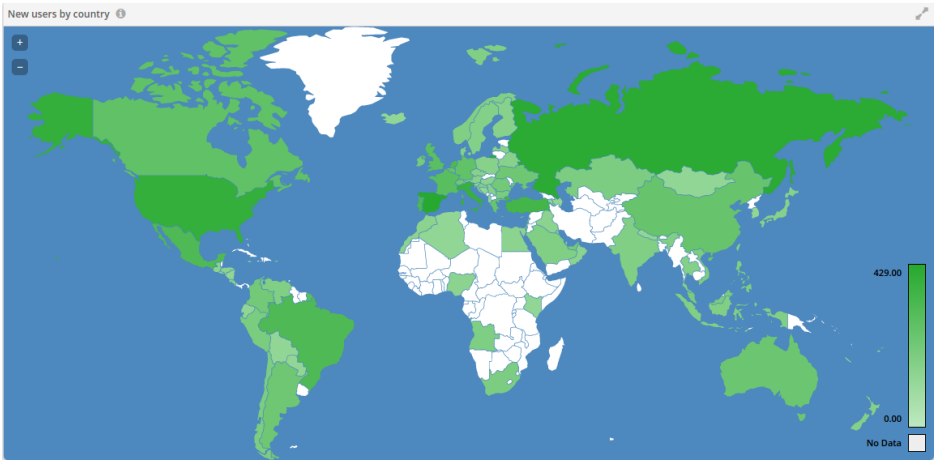

Figure 6.6: New users location since application first release

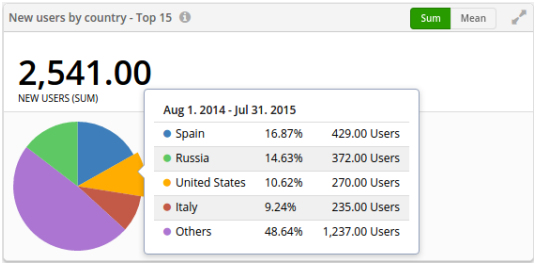

Figure 6.7: New users location since application first release top countries

As we can see, the application have been installed from a few more countries from the ones in the first year. Also, Spain, Russia, USA and Italy are the countries where most of the users came from.

6.3 Engagement

Mobile engagement is the act of engaging a user through available messaging channels inside and outside of an app. Because of that, engagement is such an important metric to analyze if the gamer has considered our application useful as their keep using it.

We will observe user engagement since our game is in the market and through two principal blocks of metrics, user retention and DAU, WAU and MAU metrics.

6.3.1 Retention

Retention is arguably the most important metric in a free-to-play game. Successful free-to-play games create long-term relationships with users. Users that enjoy the experience enough are willing to pay to for a competitive advantage. A game needs to have strong retention to have time to build this relationship.

To calculate retention, separate your users into cohorts based on the day they download your app. The day that the download occurs is Day 0. If a user opens your app the next day (Day 1), they are marked as retained. If they do not open the app, they are not retained. This calculation is performed for user cohort on each day after they download the app. Common days used for retention are 1, 3, 7 and 30. [19]

In Figure 6.8 we can see our game 90 day retention since its release:

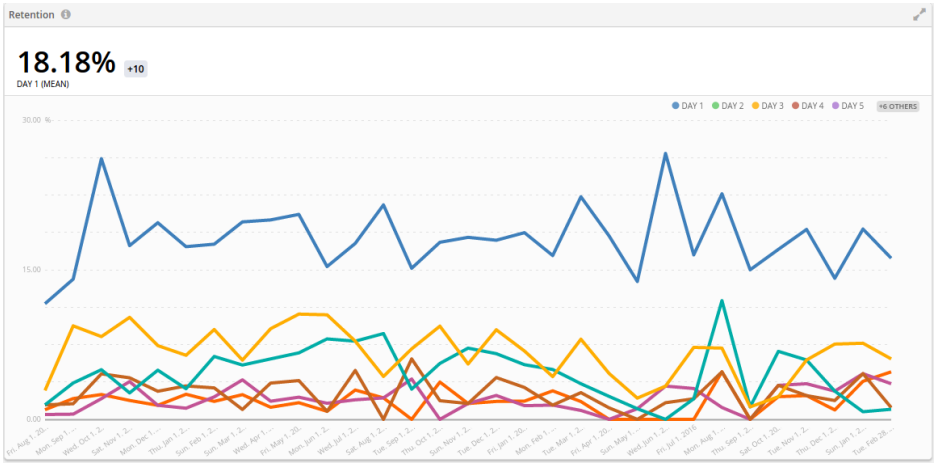

Figure 6.8: Application retention

As we can see, our game retention is 18.18%. Usually, game applications tend to have lower retention values, and according to Flurry studies, 8% retention rate at 30 days is average across the kids game iOs sub-category applications. [20] In our case, we have duplicate generally retention rates for our type of application.

6.3.2 DAU, WAU and MAU

Daily Active Users (DAUs) is the number of unique users that start at least one session in your app on any given day. By themselves, DAU and other high level metrics don't provide much insight into an app's performance. However, knowing these simple metrics is a useful starting point for an educated analytics discussion.

Weekly Active Users (WAUs) is the number of unique users that start at least one session in your app on any given week. Having both DAU and WAU, we can obtain the ratio of Daily Active Users to Weekly Active Users.

Monthly Active Users (MAUs) is the number of unique users that start at least one session in your app on any given month. Having both DAU and MAU, we can obtain the ratio of Daily Active Users to Monthly Active Users shows how well an app retains users and is often referred to as the stickiness of a game. This metric shows you how frequently users log in to your app.

Popular social networking apps like Facebook have reported DAU/MAU ratios as high as 50 percent. But most successful gaming apps have ratios closer to 20 percent. [19]

In Figure 6.9 we can see our DAU, WAU and MAU data:

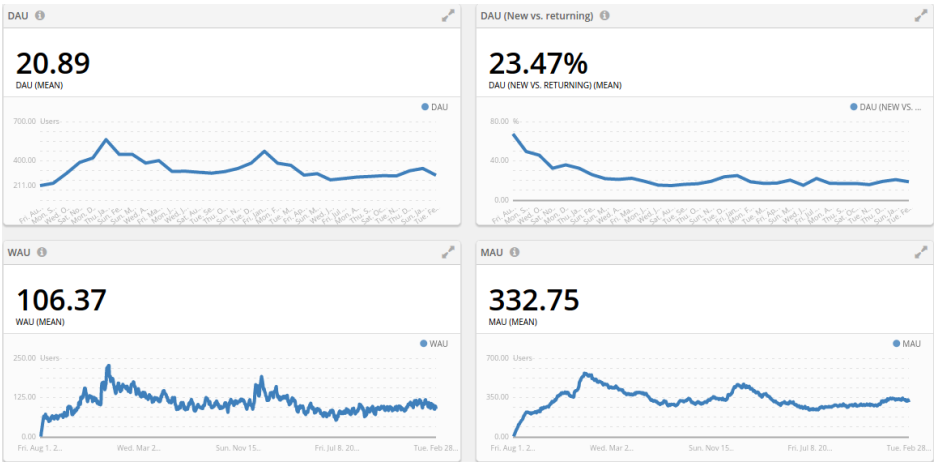

Figure 6.9: Application retention

As we can see, if we calculate the DAU/WAU and DAU/MAU ratios, we obtain 20% and 6%. This means that the average user logged in on roughly 6 percent of the days that month.

6.3.3 Quality

We will study our game quality in terms of the average numbers of screens the levels plays and in the average of frames per second their devices give.

In Figure 6.10 we can see the mean of levels that our gamers plays in every month since the application release.

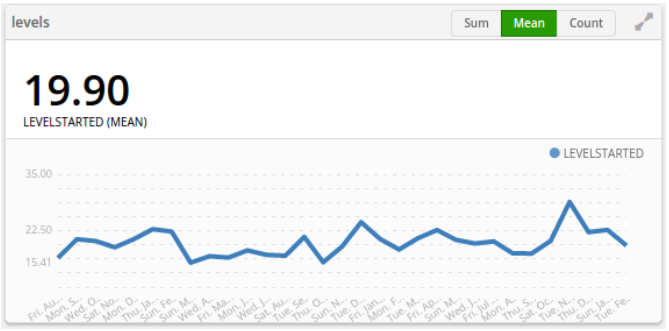

Figure 6.10: Application levels average

As we can see, gamers plays an average of 20 levels per month. In our case, each screen is a level, so the gamer plays thorough at least 20 different screens each month.

Finally, in Figure 6.11 we can observe the average frames per second shown by gamers devices.

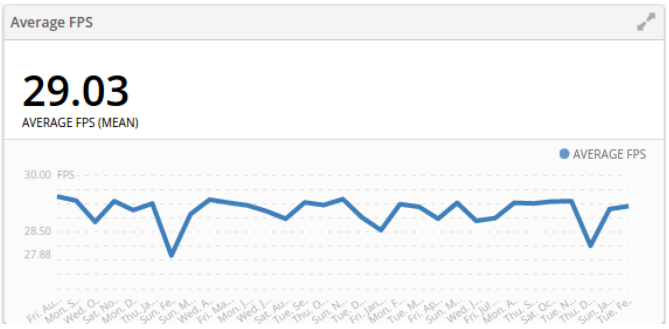

Figure 6.11: Application fps average

As we can see, gamers devices shown an average of 29 frames per second, which is the estimated value for a game which manages lots of audio and animation resources.

6.3.4 Summary

In this chapter we evaluated our application using the metrics available from Unity Game Analytics.

Firstly, we observe user acquisition in order to study the numbers of new users we obtained during the first year of the application in the market and since its release. Also, we took a look at the users location distribution.

Secondly, we detailed user engagement so that we can know if the users tend to use our game after its installation. We took a look at retention information since application release

as long as DAU, WAU and MAU metrics.

Finally, we observed the game quality detailed the numbers of levels the gamer played and the numbers of frames per second their devices shown since the application release.

CHAPTER 7

Conclusions

In this chapter we will describe the conclusions extracted from this master thesis and detail the achieved goals. Also, the thinkings about future work will be detailed.

7.1 Conclusions

By using Unity3D as the base to build the application we have been able to create a cross-platform mobile game without the effort requires to manage separate developments for each platform.

We have built an application from scratch following a typical mobile application development lifecycle defined the in the following four different phases: discovery, design, development, testing and deployment. [21]

Firstly, we covered the requirement analysis phase, determining the needs for our application, taking account of the possibly conflicting requirements of our client, analyzing, documenting, validating and managing software or system requirements. [22]

Secondly, we obtained the use cases, which covers all the functional requirements obtained in the first stage of our development.

Then, we design the game architecture, where we decided to use Unity3D as our game base engine. Moreover, four asset plugins placed in Unity3D (2D Toolkit, HOTween, iOs native and Android native) where included in the architecture in order to improve performance and make the development easier. Also we need to design physical pieces to let the gamer interact with the software application. This pieces have a unique base pads distribution to let our algorithm identify each piece. In order to detect what physical figure has been placed on the application recognition zones we build the instrument recognition algorithm.

Finally, we design three game modes to cover the use cases presented. In this document we have presented two of them.

Following this path we have been able to build a multi-platform mobile musical training software for children using the framework Unity3D engine.

This multi-platform game application have reached being a commercial product. Physical instrument pieces are sold in a renowned toy shop which have stores in more than 20 countries. Also, both iOs and Android applications have been are available in App Store and Google Play respectively.

7.2 Achieved goals

In this section we will detail our application's achieved goals checking if we have covered all the use cases presented in section 3.2.2

Play an instrument This goal has been achieved successfully. This use case is described in 3.2.2.1. The gamer is able to play five different instruments, one of each musical instrument family. In order to select the instrument to play with, the gamer is able to place the physical miniature, which represents the instrument family, on a recognition zone in the screen. Also, the gamer can watch a demo or change the melody to be played. We can see this achievement in figures 5.3, 5.7, 5.8, 5.9 and 5.10

Conduct the orchestra This goal has been achieved successfully. This use case is described in 3.2.2.2. The gamer is able to conduct the orchestra that is playing the selected melody. The gamer can conduct the melody by enabling or disabling the instruments that are playing this melody. In order to select the instrument family whose instruments will be able to be enabled or disabled, the gamer is able to place the physical miniature, which represents the instrument family, on a recognition zone in the screen. Also, the gamer can watch stop or change the melody to be conducted. We can see this achievement in figures 5.14 and 5.15.

Discover an instrument This goal has been achieved successfully. This use case is described in 3.2.2.3. The gamer is able to read information of instruments of the five different musical families. In order to select the instruments to discover, the gamer is able to place the physical miniature, which represents the instrument family, on a recognition zone in the screen. Also the gamer can reproduce the instrument selected sound. We can see this achievement in figures shown in section B.4

Watch a melody play demo This goal has been achieved successfully. This use case is described in 3.2.2.4. Gamer is able to watch a demo of all available melodies in the *Play instrument* game mode with each of the five instruments the gamer is able to play with.

Select a melody This goal has been achieved successfully. This use case is described in 3.2.2.5. Gamer is able to select a melody within both *Play instrument* and *Conduct orchestra* game modes. We can see this achievement in figures 5.5 and 5.13.

Select an instrument This goal has been achieved successfully. This use case is described in 3.2.2.6. As we have said in the previous achievements, the gamer is able to choose

an instrument within all game modes. In order to select the instrument, the gamer is able to place the physical miniature, which represents the instrument family, on a recognition zone in the screen.

Build the instrument recognition algorithm This goal has been achieved successfully. This actor is described in 3.1. The instrument recognition algorithm detects what physical figure has been placed on the application recognition zones. This algorithm is used within the three game modes included in our application when the user place the physical miniature, which represents the instrument family, on a recognition zone in the screen.

7.3 Future work

There are several lines than can be followed to continue and extend features of this work.

In the following points we present some improvements that we can add to our application to continue the development.

- Add new instruments to the *Play instrument game mode*. This new feature imply the development of new screens for each new instruments and the manage of new musical sounds.

- Add new instrument families to the game. This feature imply the design of new pieces as long as modifying the recognition algorithm to support them. Also, the three game modes should be adapted to represent the new instrument families.

- Update game project to Unity 5. This involves some work related with all Unity components. While many areas are upgraded automatically, there are certain parts of the project where we will need to manually adjust or refactor.

- Reduce artifacts size. While Android and iOs markets allow us to upload huge sized applications, this suppose that user should only install the application when they are connected through WiFi. This would suppose a barrier to attract new gamers, so artifact size should be reduced from 300 MB, its actual size.

- Add notifications management system in order to allow the client to interact with the gamer. This would let the client to promote their other applications and the new game features

- Reduce resources requirements. This would permit lower resources devices to run the application and decrease the battery use.
- Automate the deployment phase. This feature would allow us to decrease times when applying bug fixes to our application.

APPENDIX A

Game Play images

This appendix shows game play captures. It goes through the three different game modes and shows how the gamer interacts with them using the physical instrument pieces.

A.1 Playing game mode

Figure A.1: Entering playing game mode from home screen

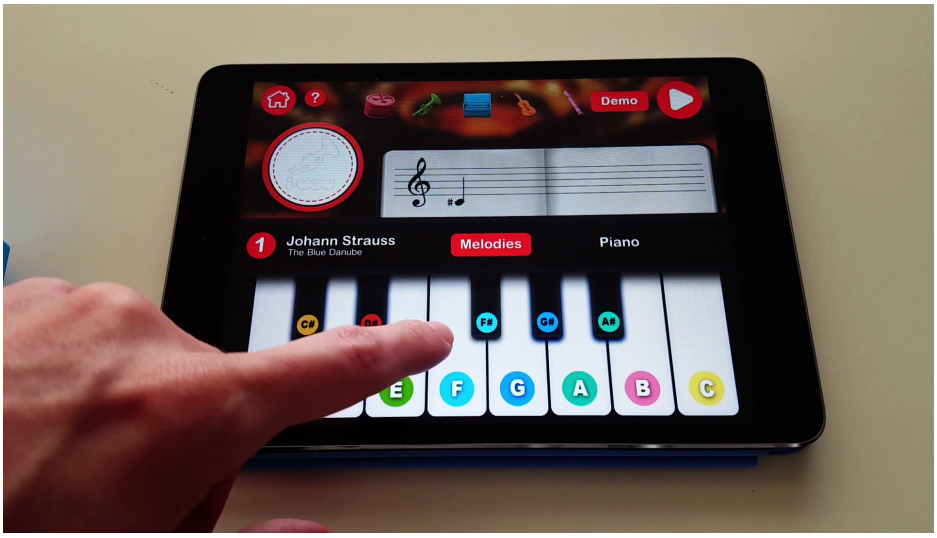

Figure A.2: Playing piano free mode

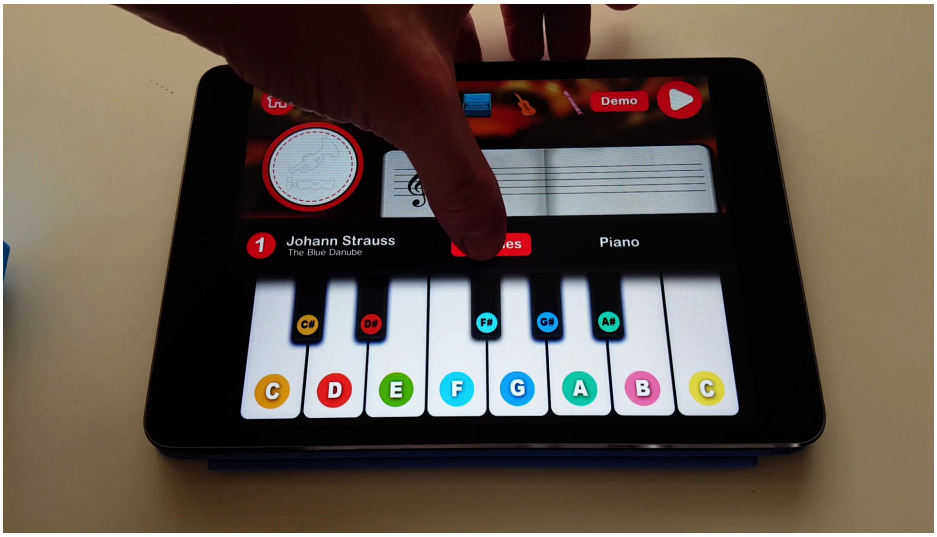

Figure A.3: Opening melodies menu in playing game mode

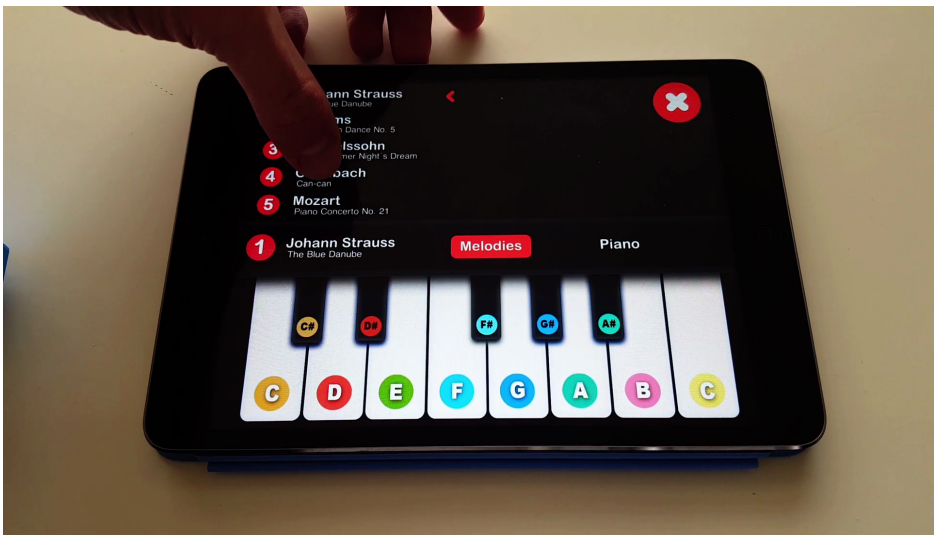

Figure A.4: Changing melody to be played in playing game mode

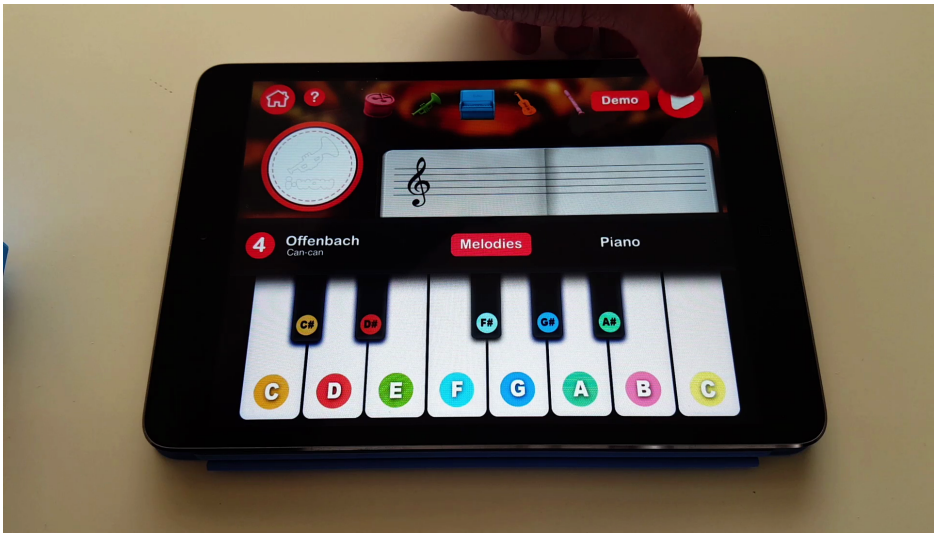

Figure A.5: Starting guided playing mode

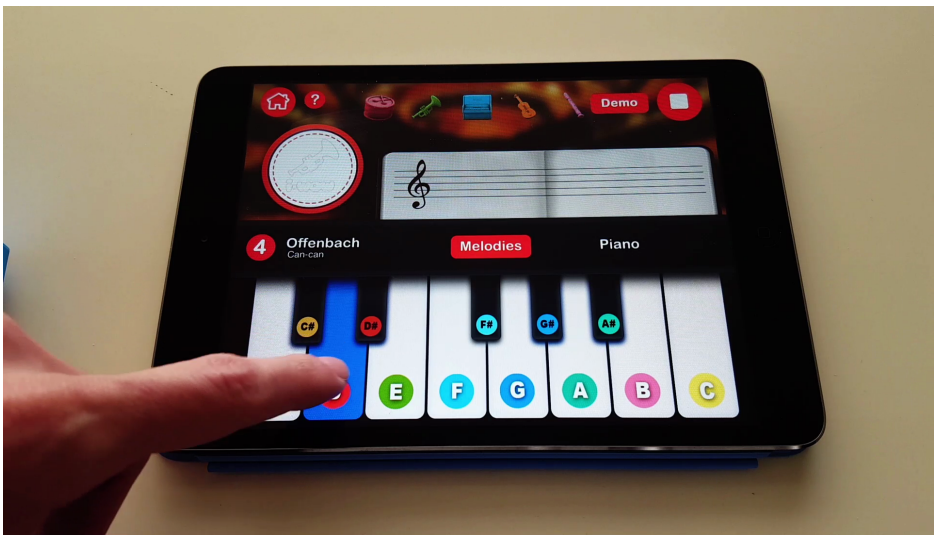

Figure A.6: Playing piano guided

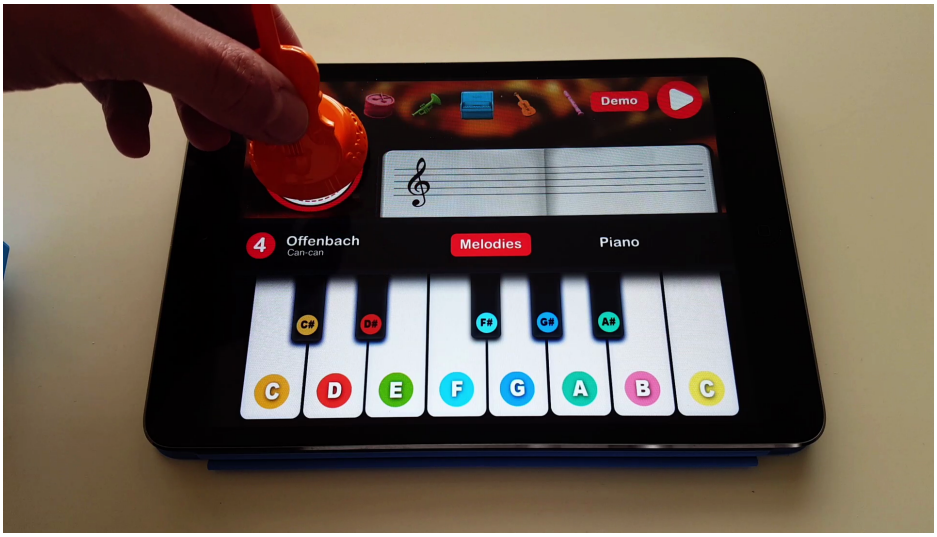

Figure A.7: Placing strings piece in playing game mode

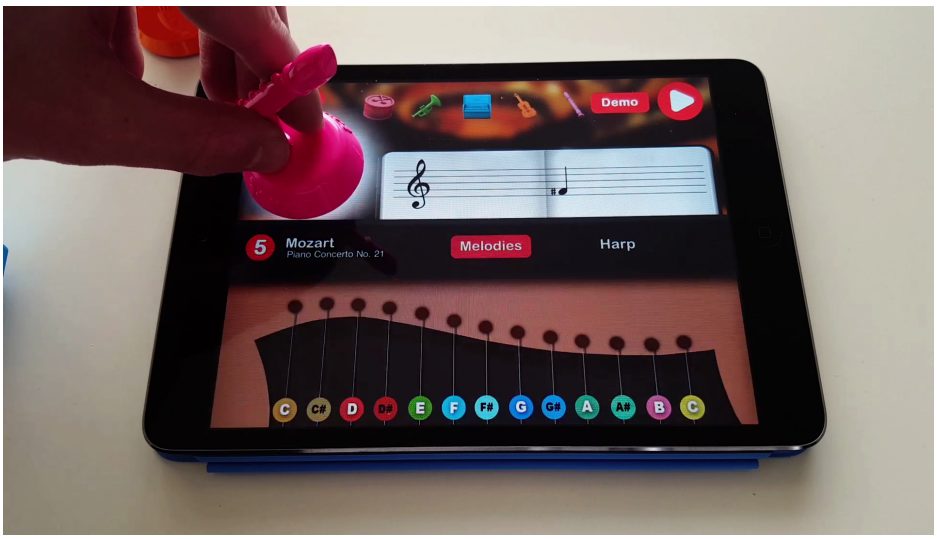

Figure A.8: Placing woodwind piece in playing game mode

Figure A.9: Placing brass piece in playing game mode

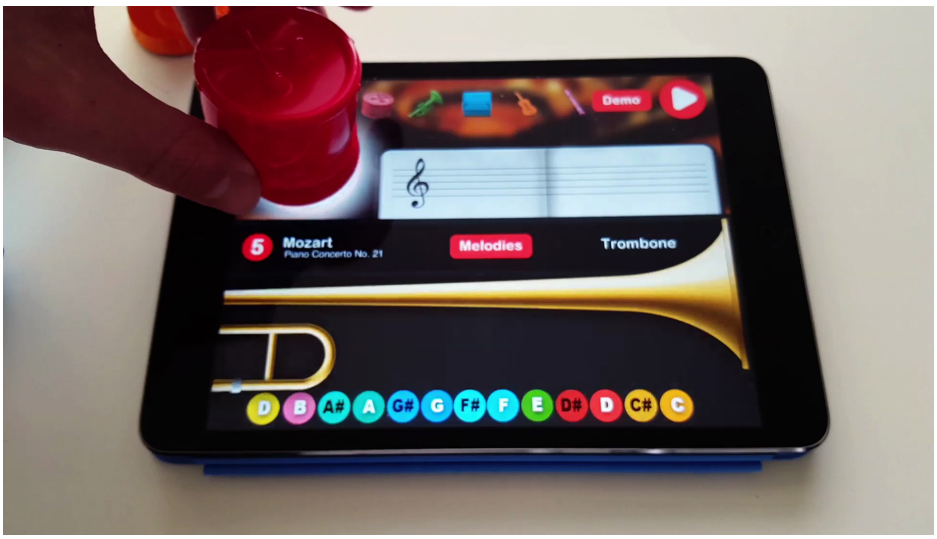

Figure A.10: Placing percussion piece in playing game mode

A.2 Conducting game mode

Figure A.11: Entering conducting game mode from home screen

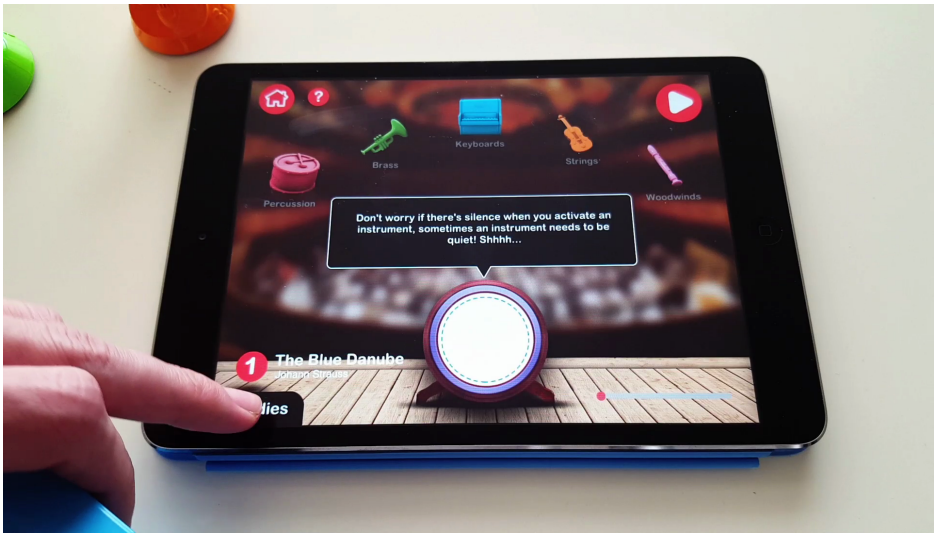

Figure A.12: Opening melodies menu in playing game mode

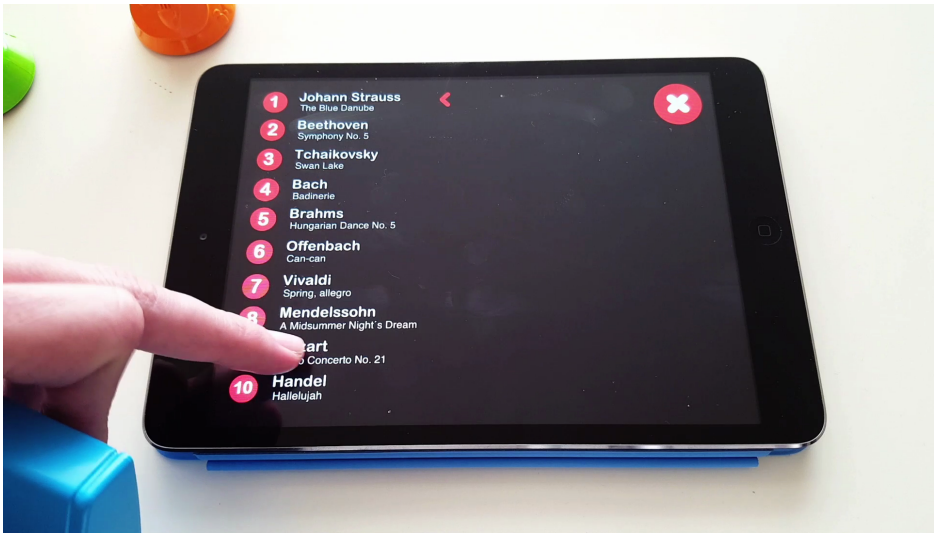

Figure A.13: Changing melody to be played in playing game mode

Figure A.14: Activating keyboards family

Figure A.15: Disabling instrument

Figure A.16: Enabling instrument

A.3 Discovering game mode

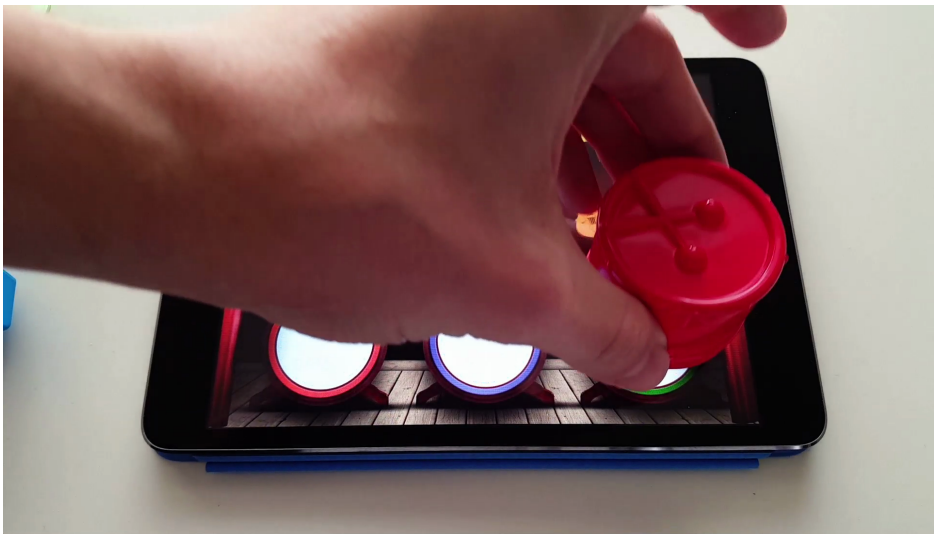

Figure A.17: Entering discovering game mode from home screen

Figure A.18: Playing instrument sound

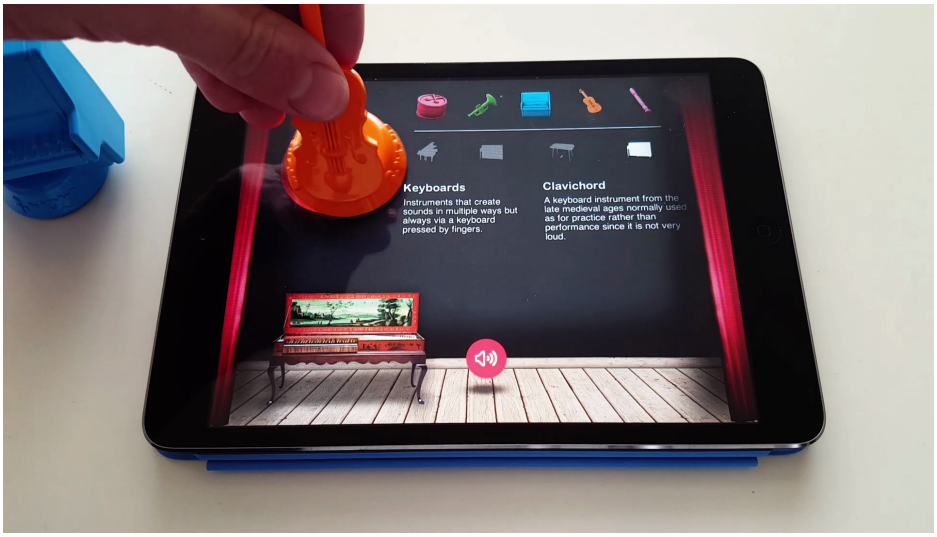

Figure A.19: Changing family in discovering game mode

Figure A.20: Changing instrument in discovering game mode

APPENDIX B

Application game screens

This appendix shows all the screens which conform the application. It goes through the three different game modes and shows all the screens contained within them.

B.1 Home

Figure B.1: Application Home Screen

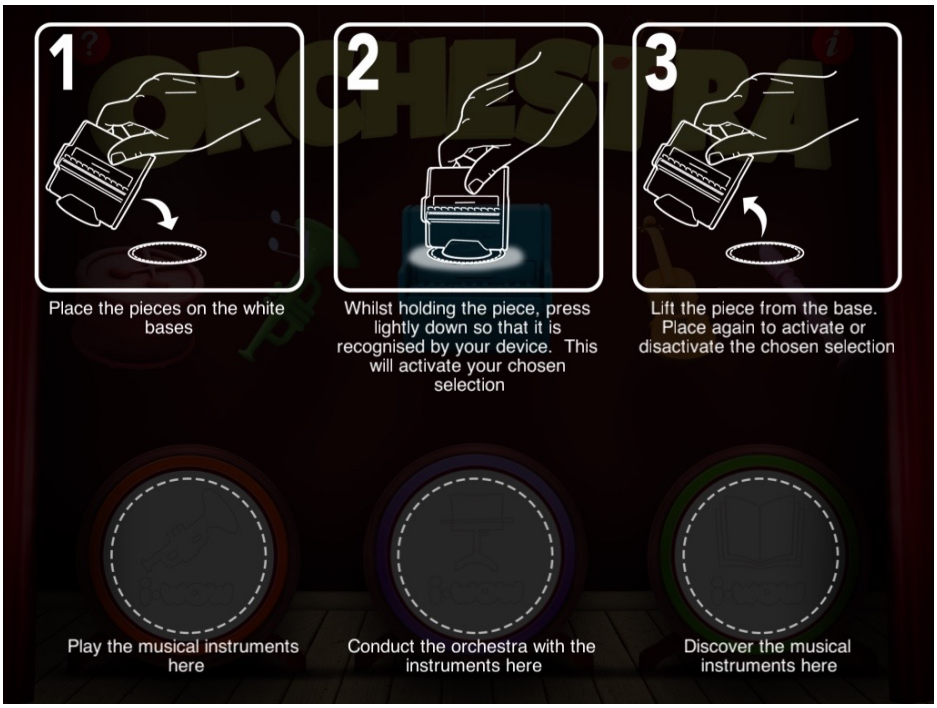

Figure B.2: Help Home Screen

B.2 Playing game mode

Figure B.3: Xylophone playing instrument game mode

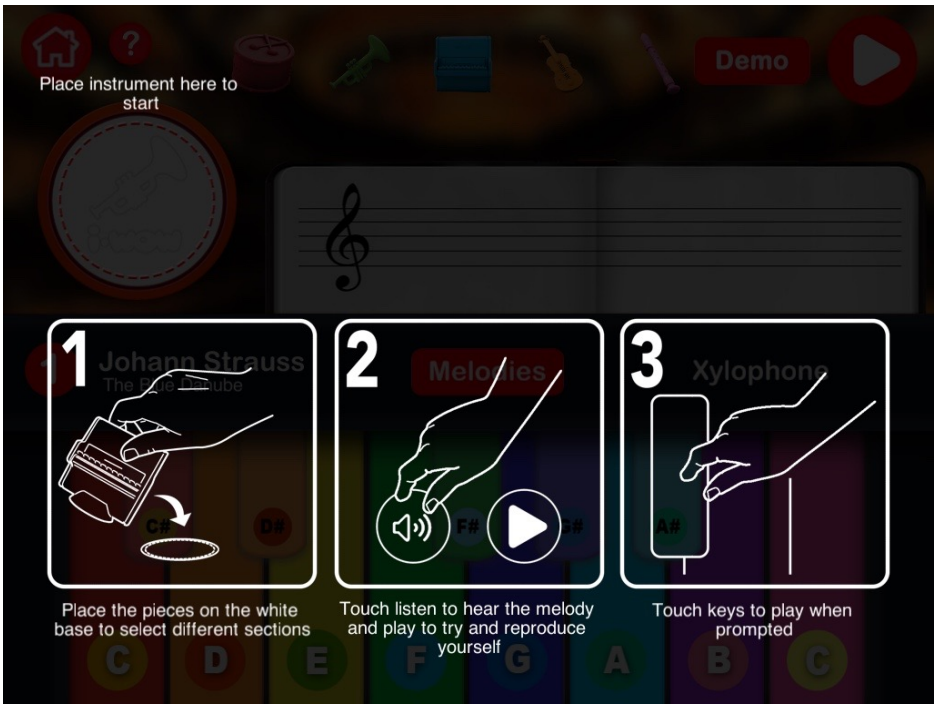

Figure B.4: Help information playing instrument game mode

Figure B.5: Playing panpipes screen

Figure B.6: Playing trombone screen

Figure B.7: Playing piano screen

Figure B.8: Playing harp screen

B.3 Conducting game mode

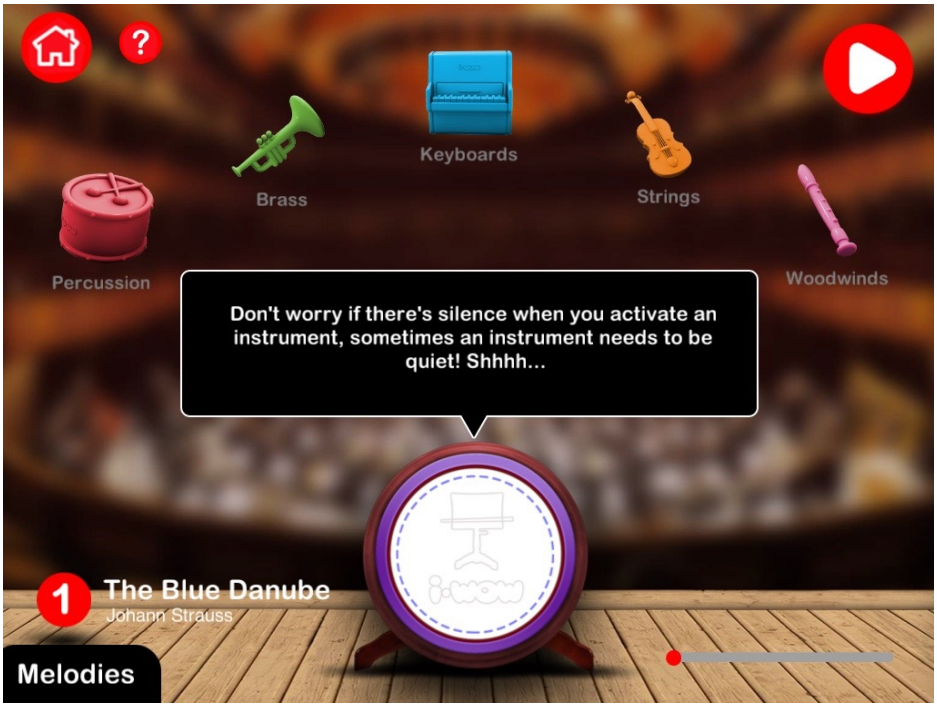

Figure B.9: Conducting game mode access screen

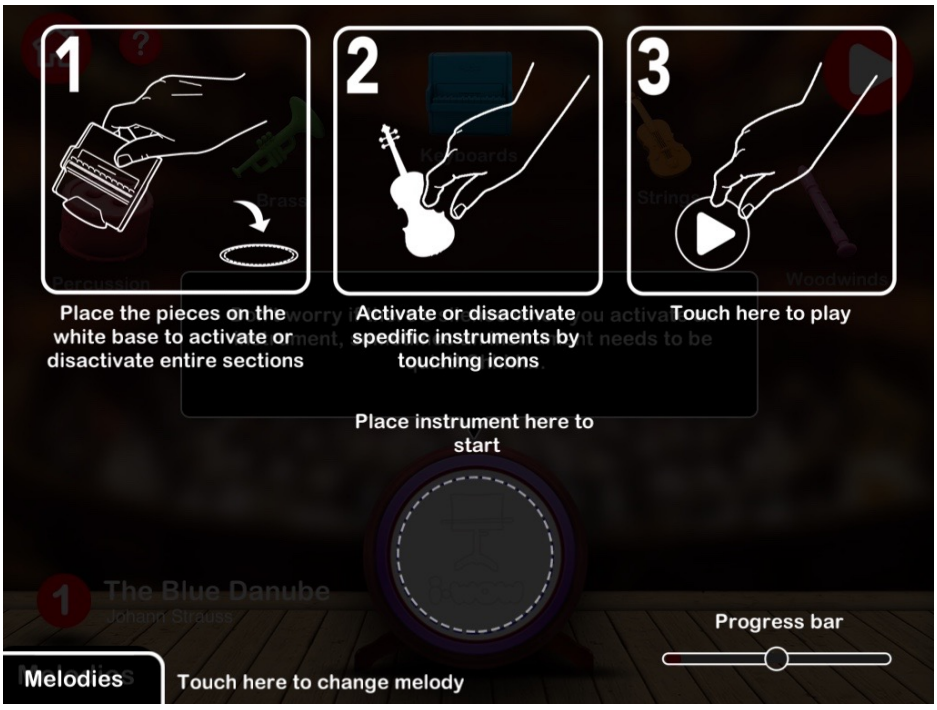

Figure B.10: Help information conducting orchestra game mode

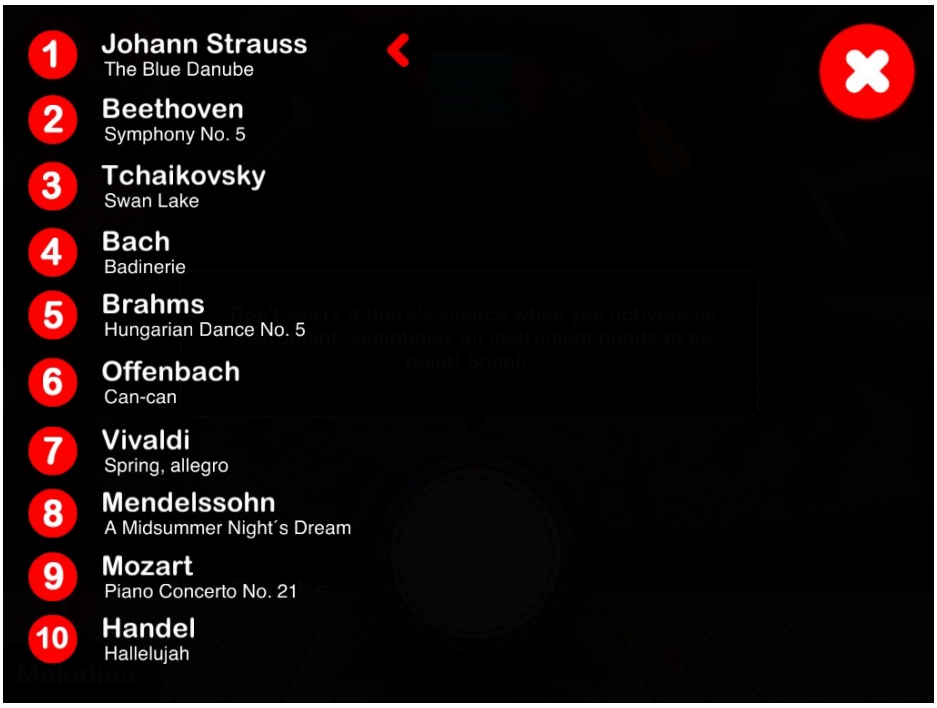

Figure B.11: Melodies selection in the conducting orchestra game mode

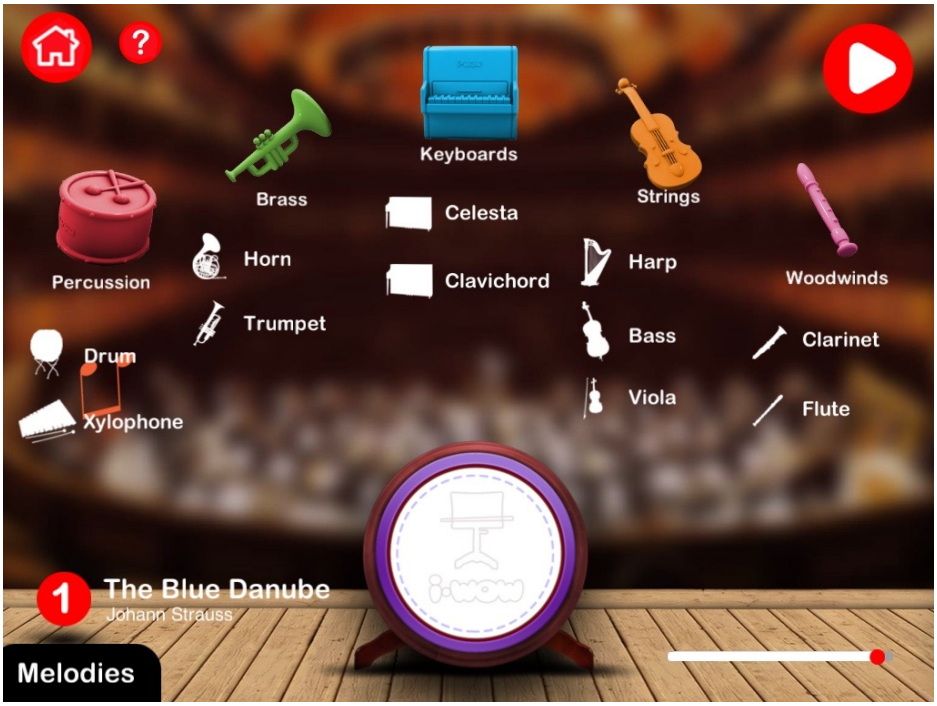

Figure B.12: Conducting orchestra screen with all instruments activated

B.4 Discovering game mode

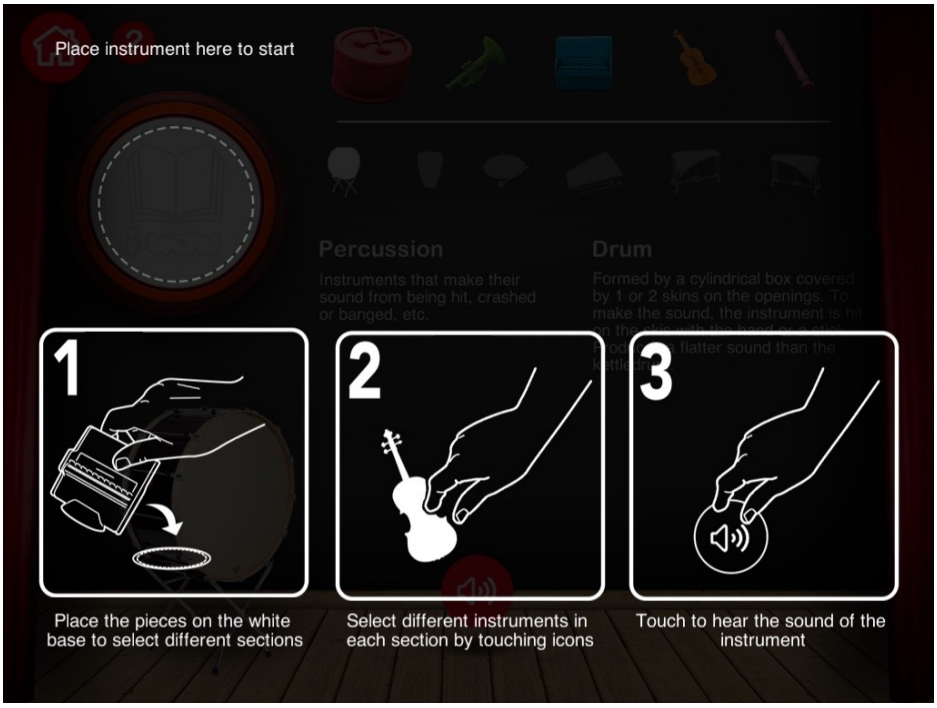

Figure B.13: Help Discovering Screen

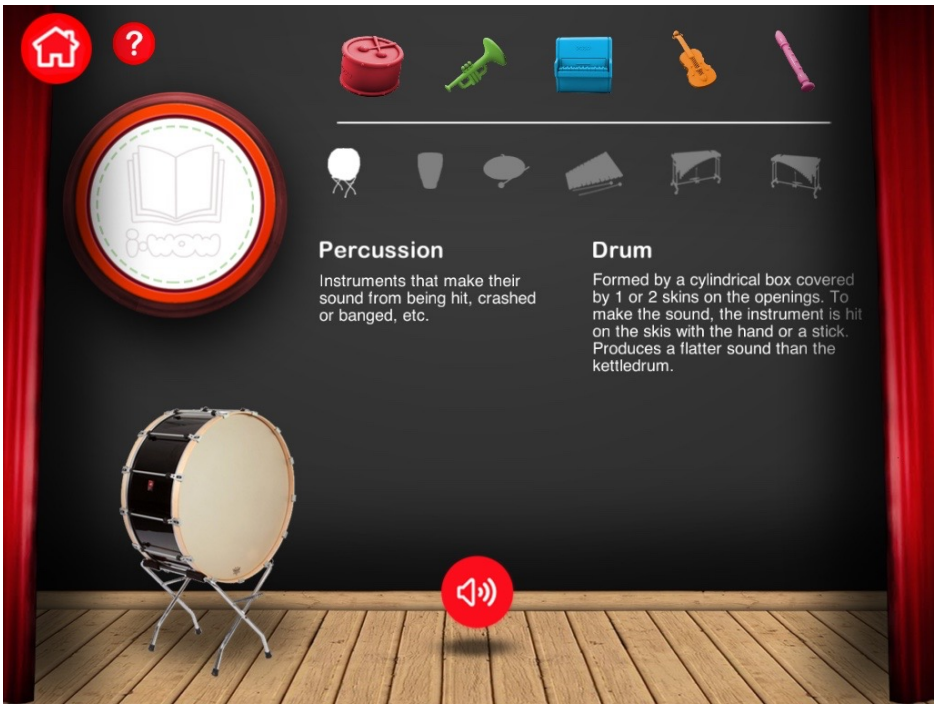

Figure B.14: Discovering drum instrument

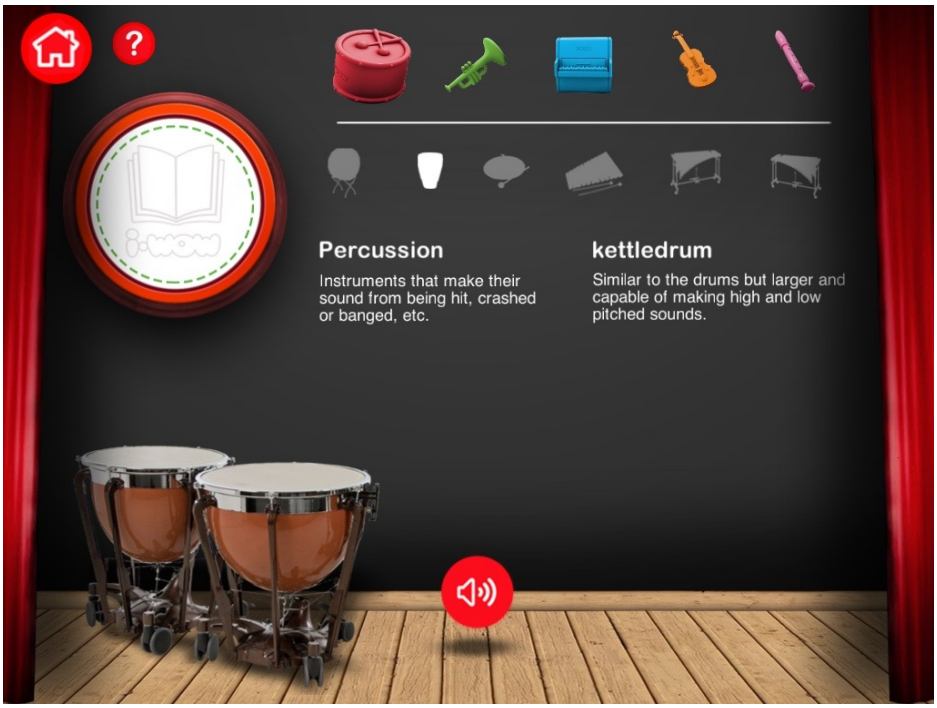

Figure B.15: Discovering kettle instrument

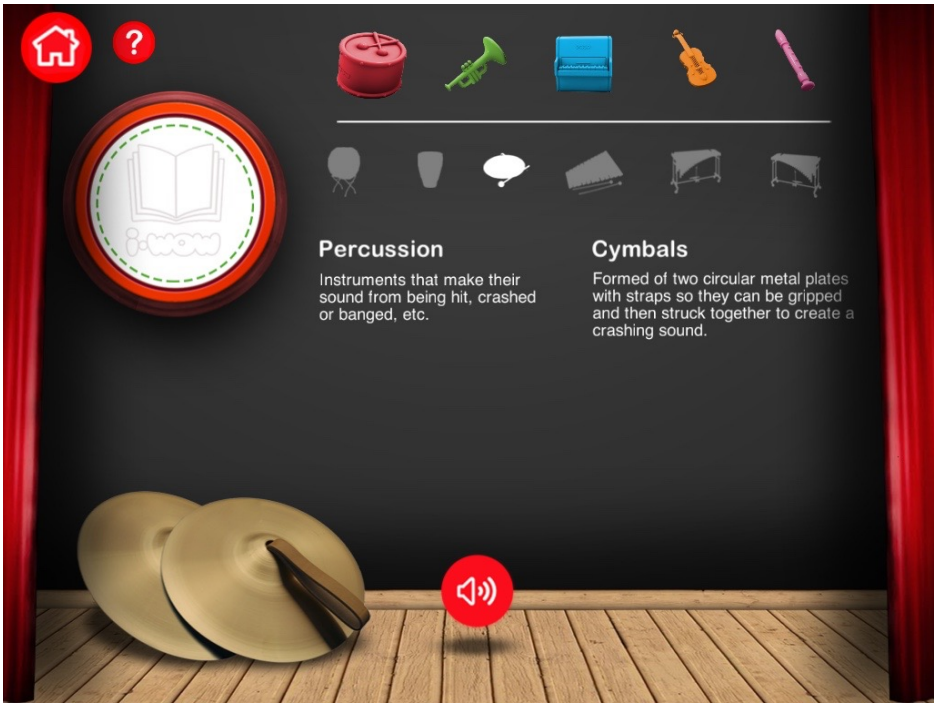

Figure B.16: Discovering cymbals instrument

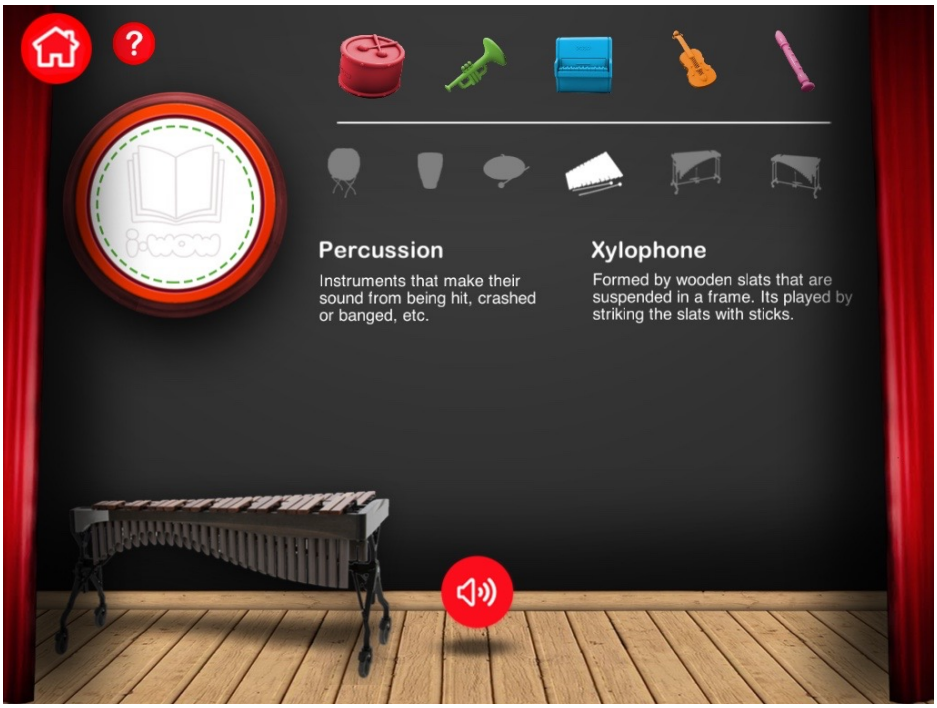

Figure B.17: Discovering xylophone instrument

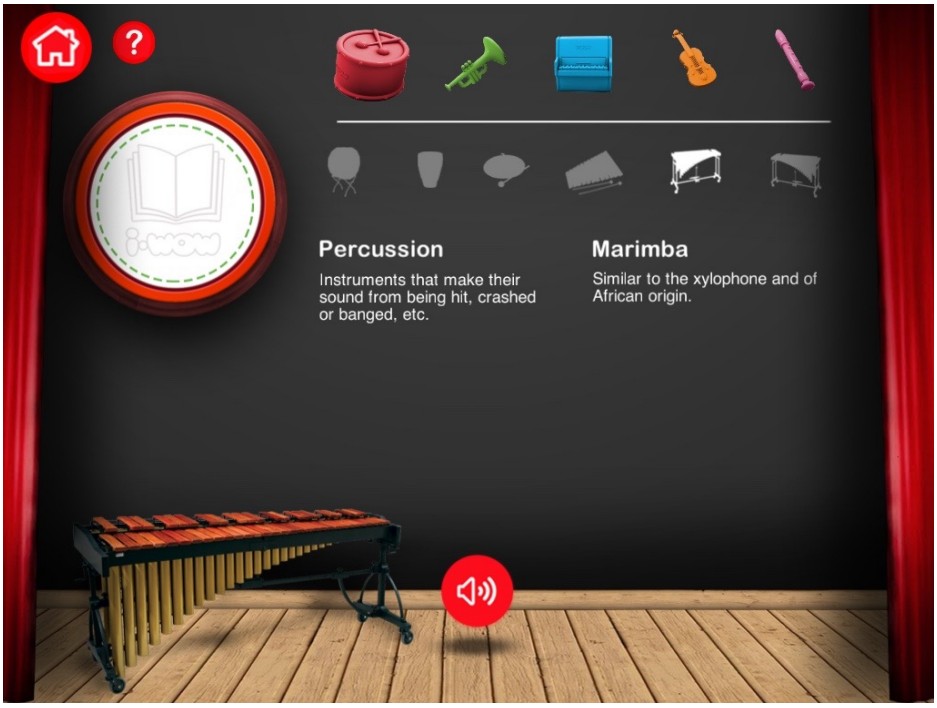

Figure B.18: Discovering marimba instrument

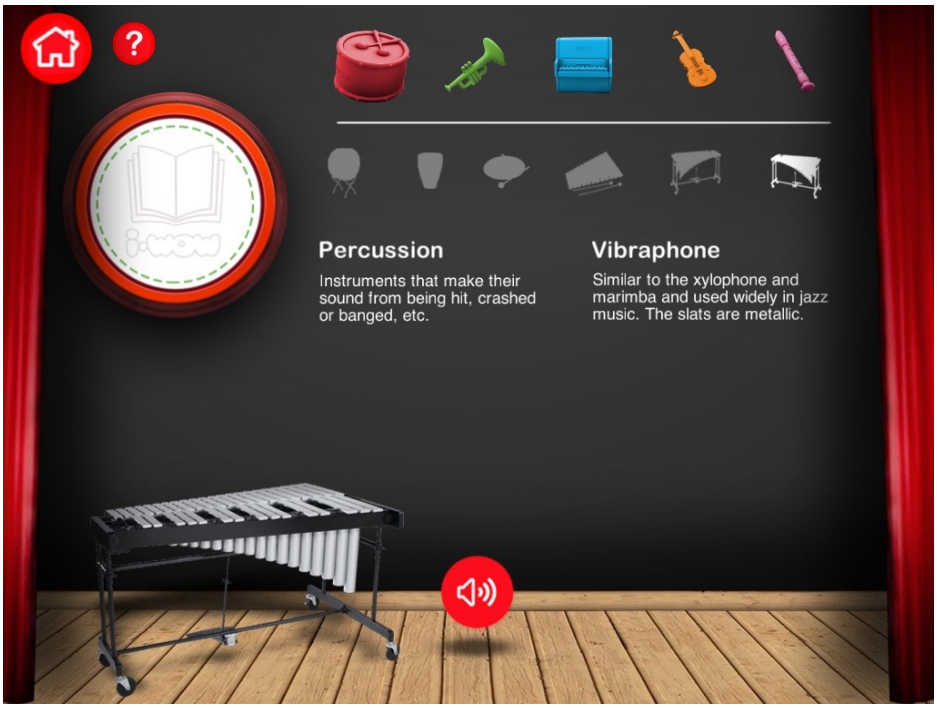

Figure B.19: Discovering vibraphone instrument

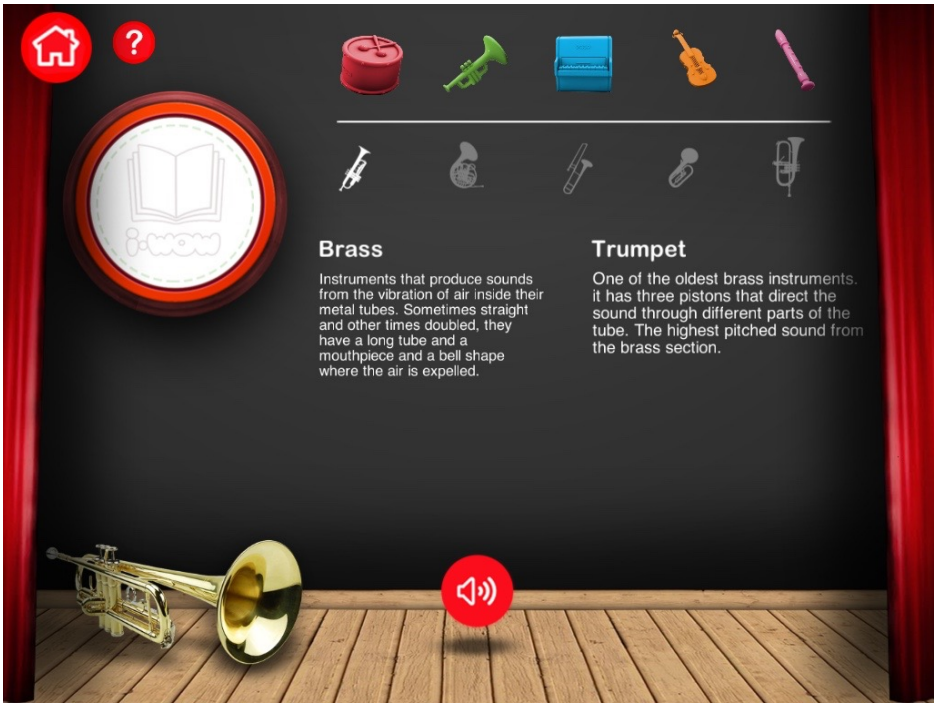

Figure B.20: Discovering trumpet instrument

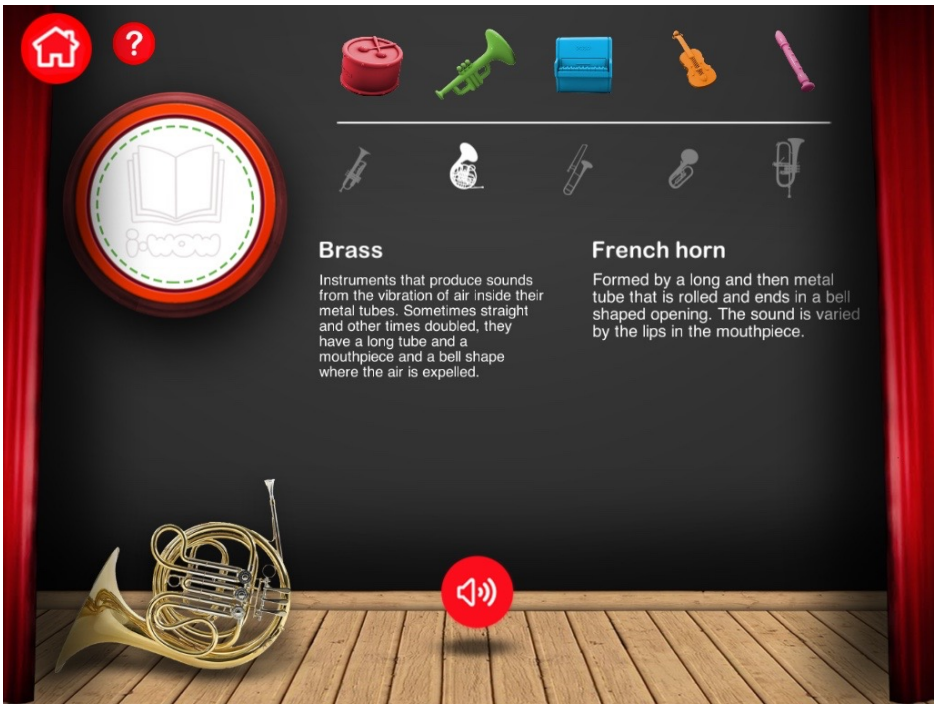

Figure B.21: Discovering French horn instrument

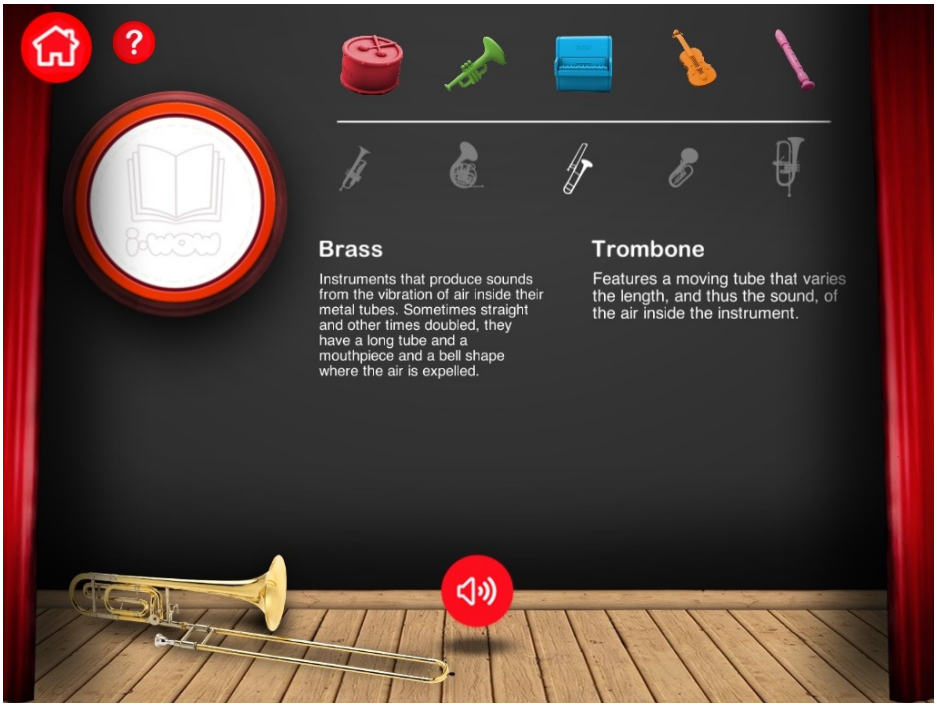

Figure B.22: Discovering trombone instrument

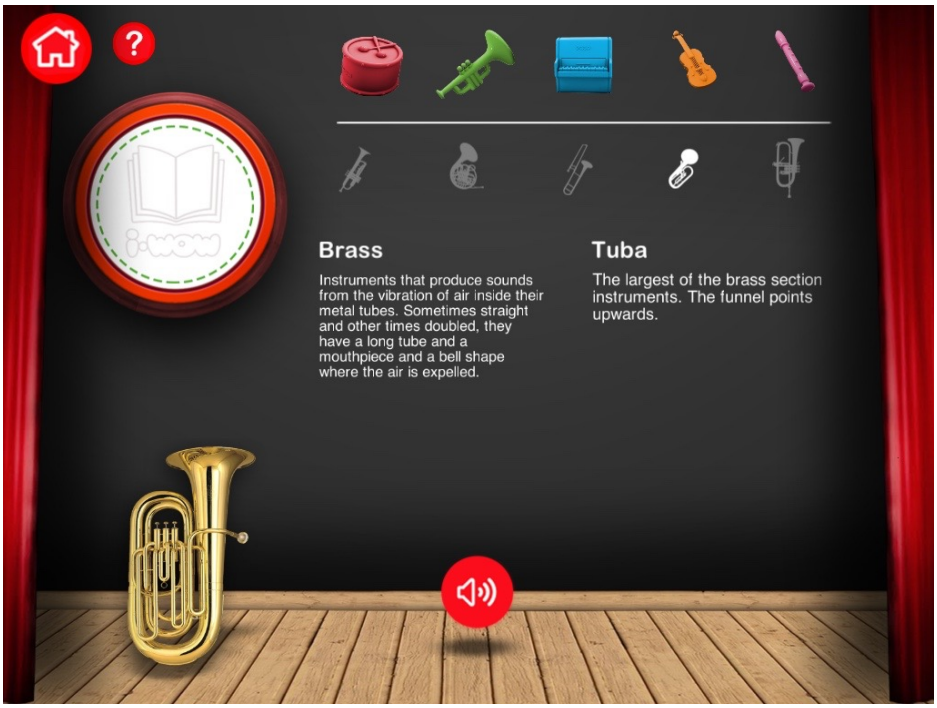

Figure B.23: Discovering tuba instrument

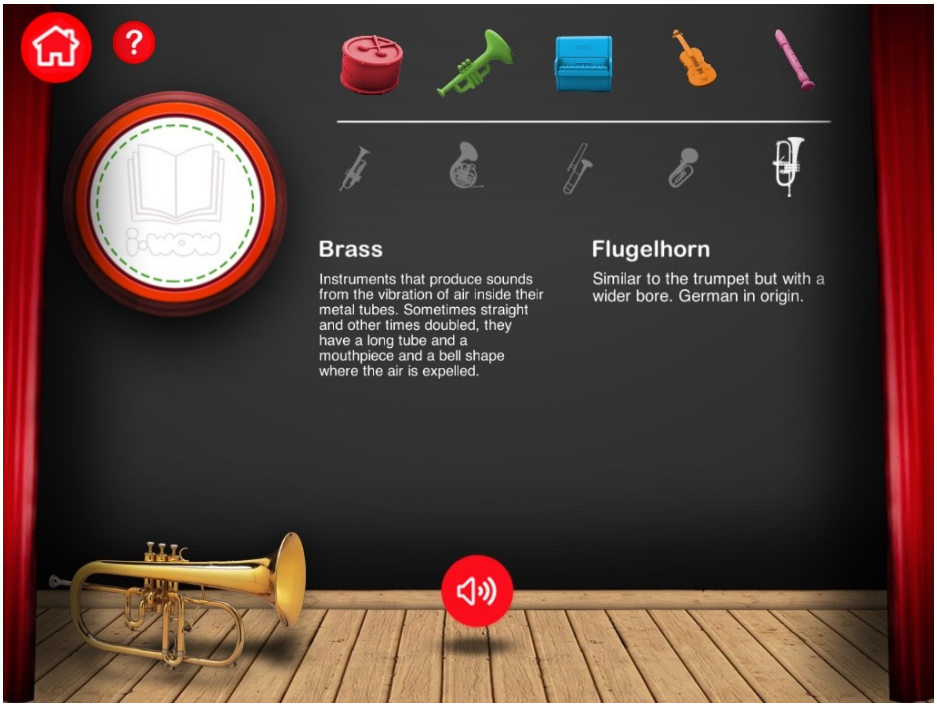

Figure B.24: Discovering flugelhorn instrument

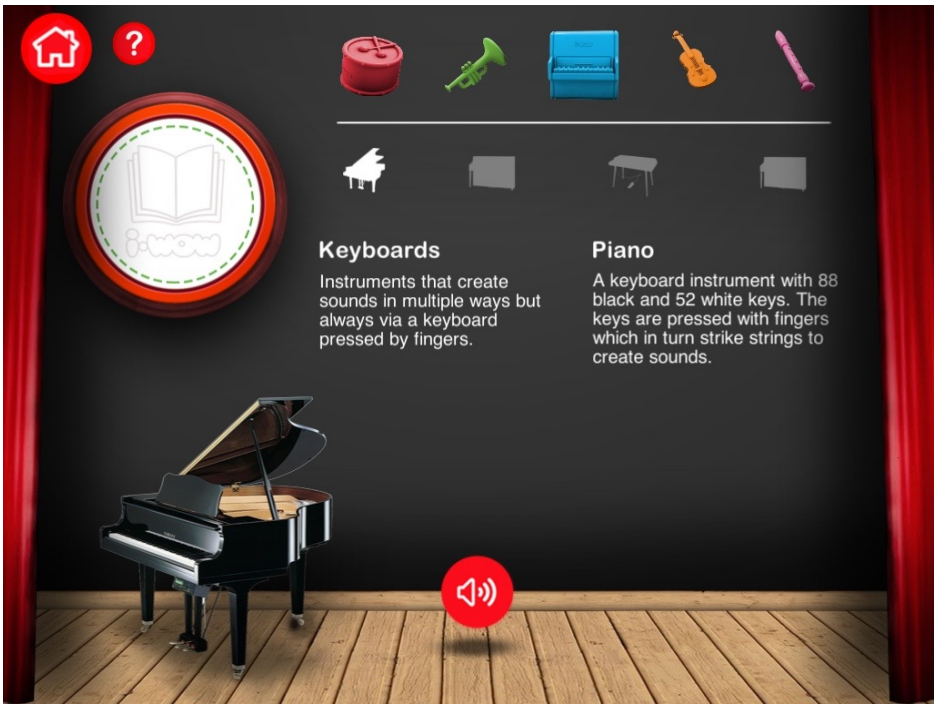

Figure B.25: Discovering piano instrument

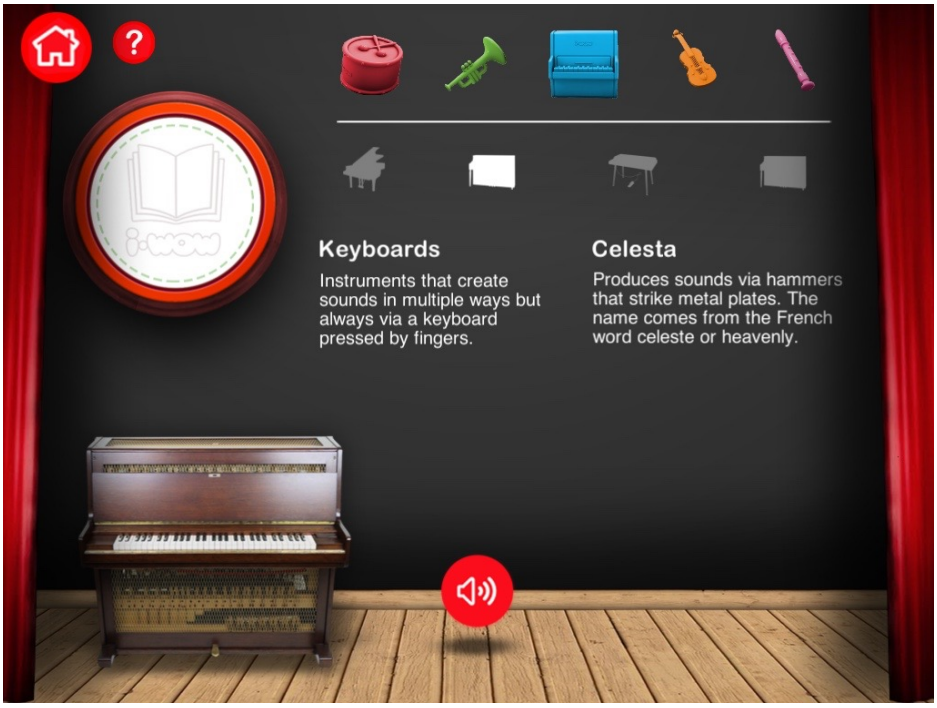

Figure B.26: Discovering celesta instrument

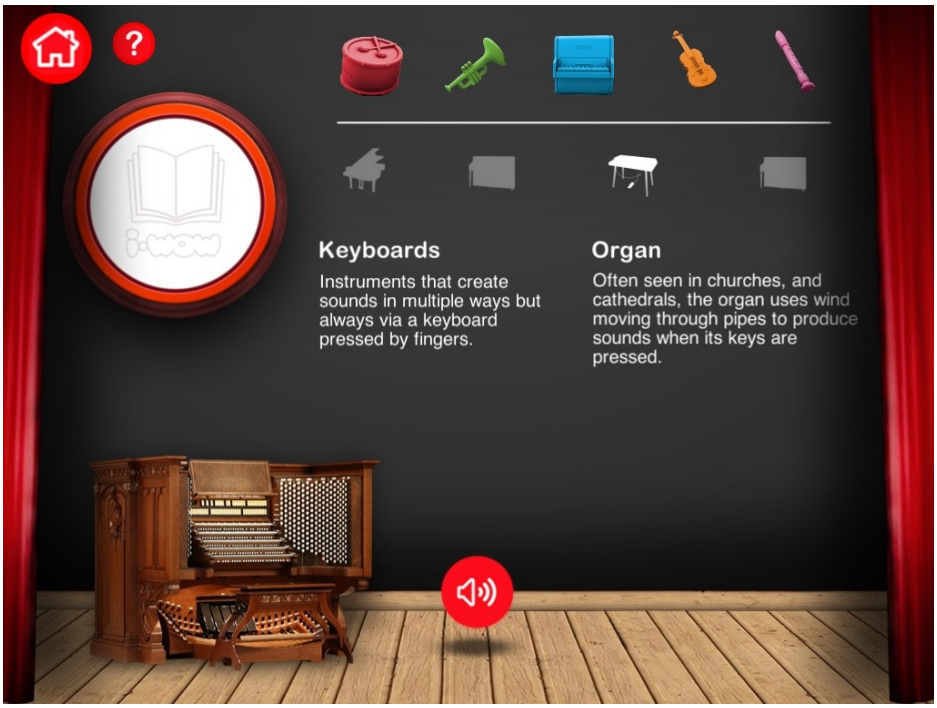

Figure B.27: Discovering organ instrument

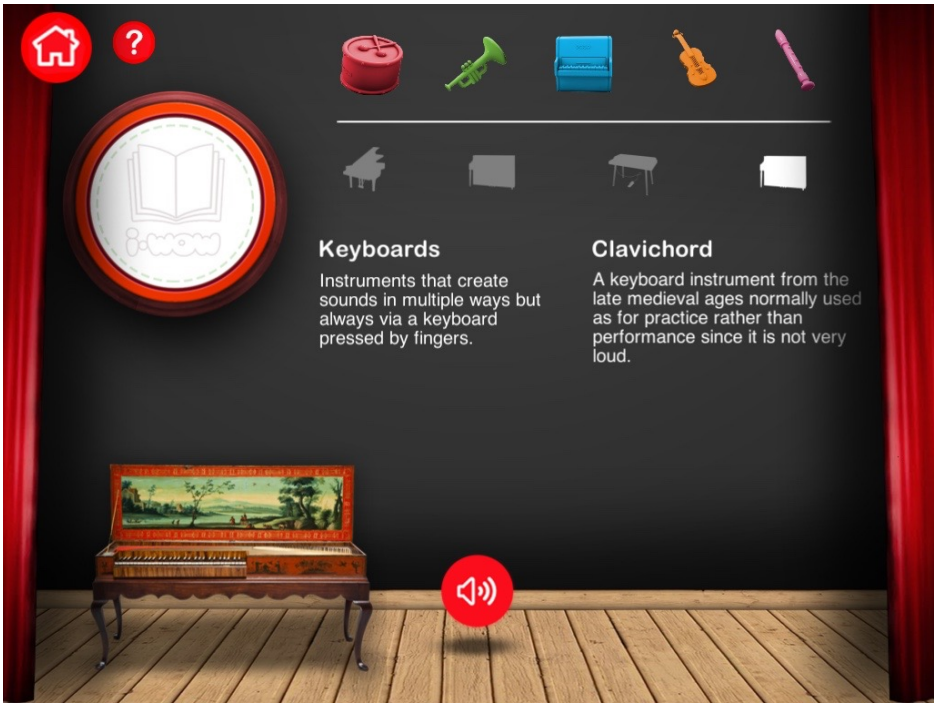

Figure B.28: Discovering clavichord instrument

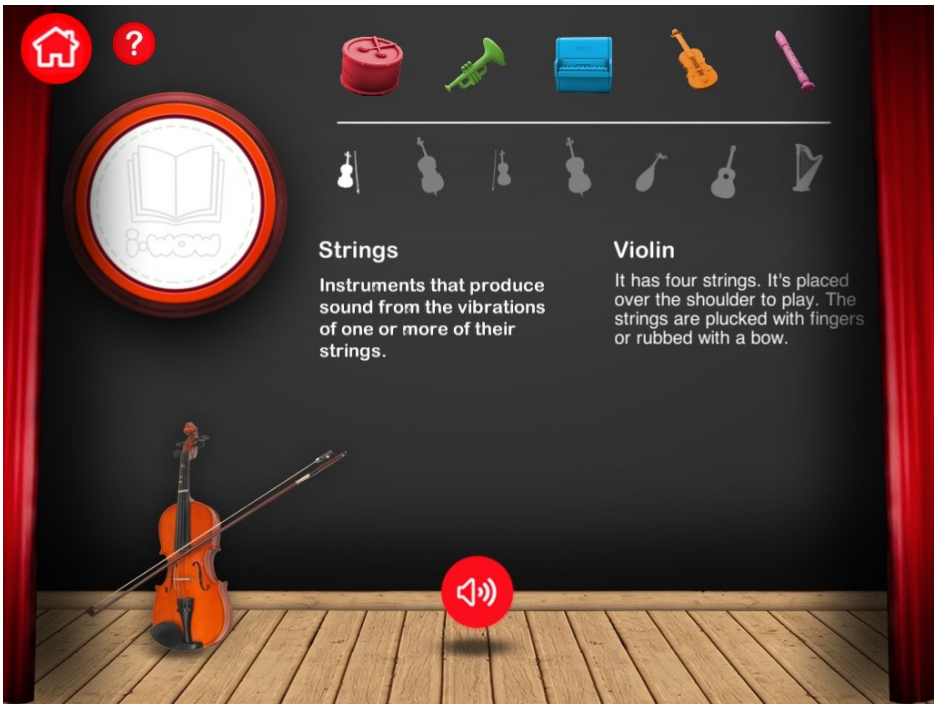

Figure B.29: Discovering violin instrument

Figure B.30: Discovering double bass instrument

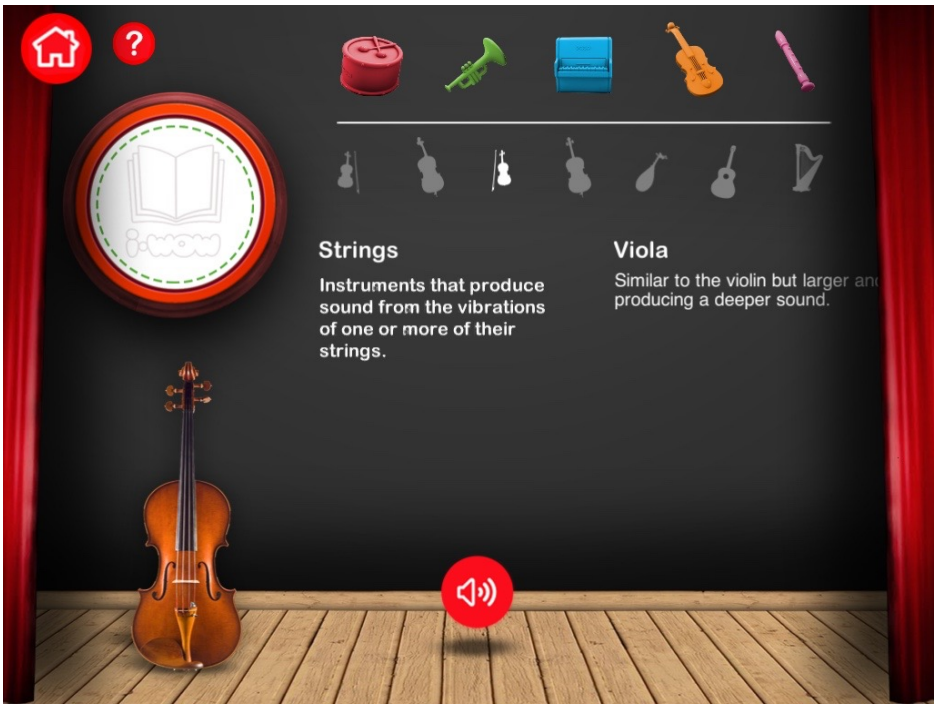

Figure B.31: Discovering viola instrument

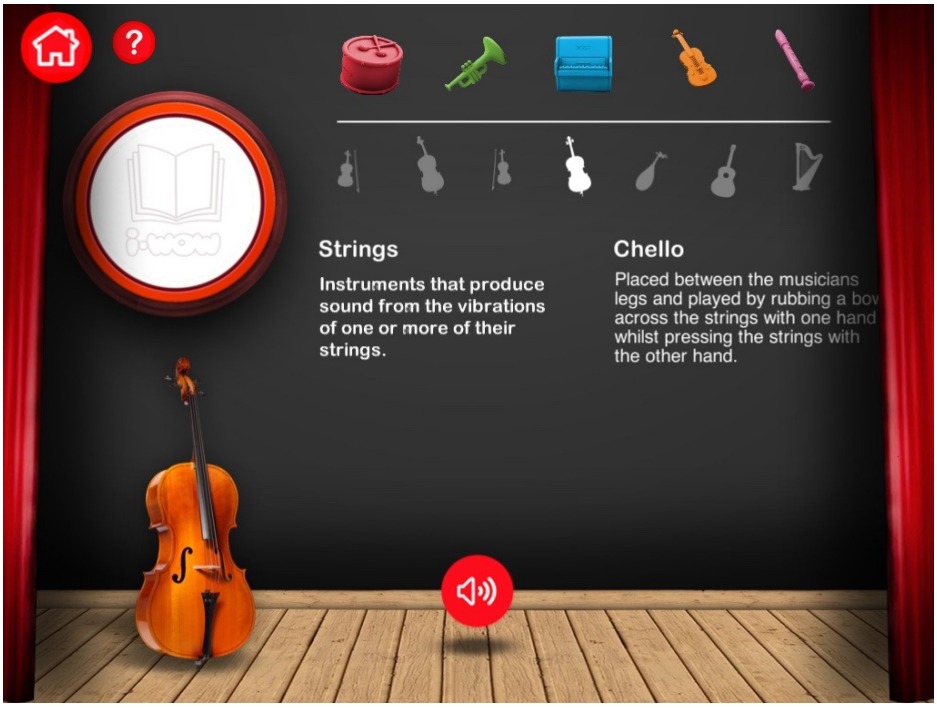

Figure B.32: Discovering chello instrument

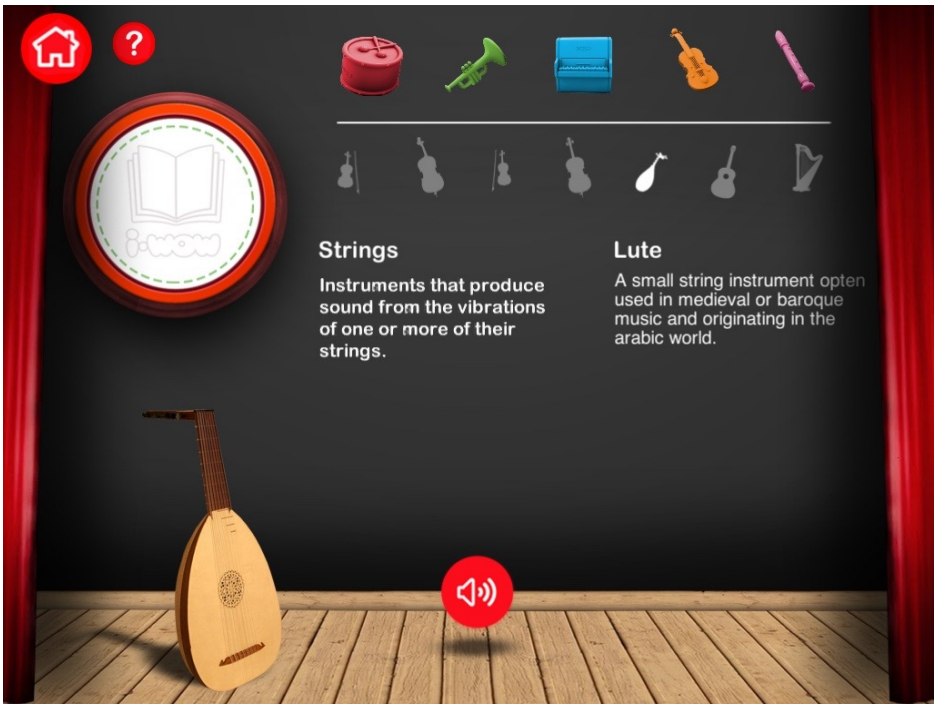

Figure B.33: Discovering lute instrument

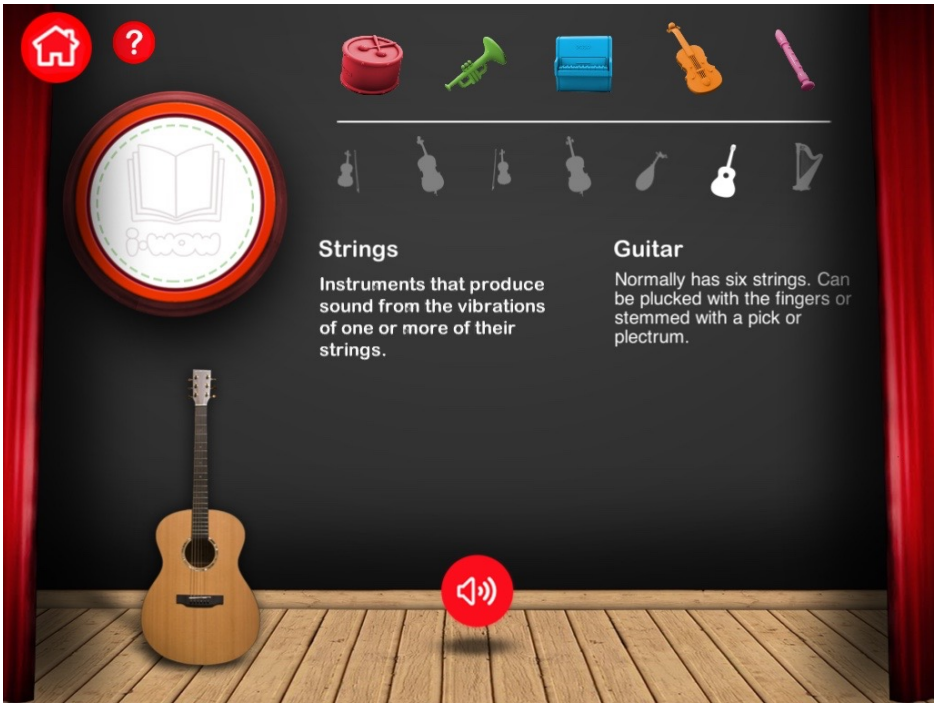

Figure B.34: Discovering guitar instrument

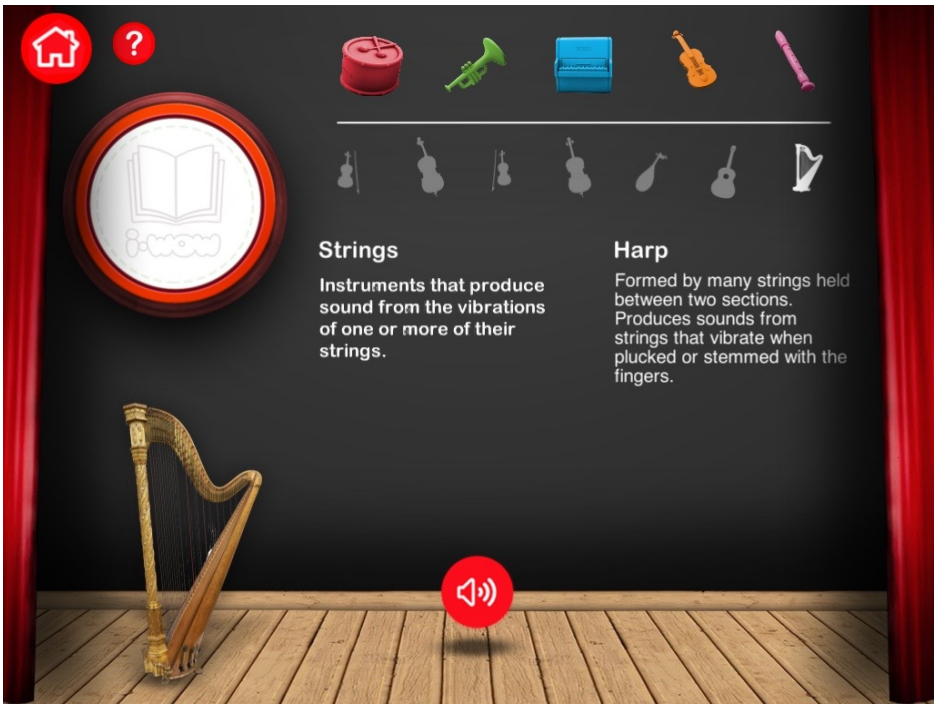

Figure B.35: Discovering harp instrument

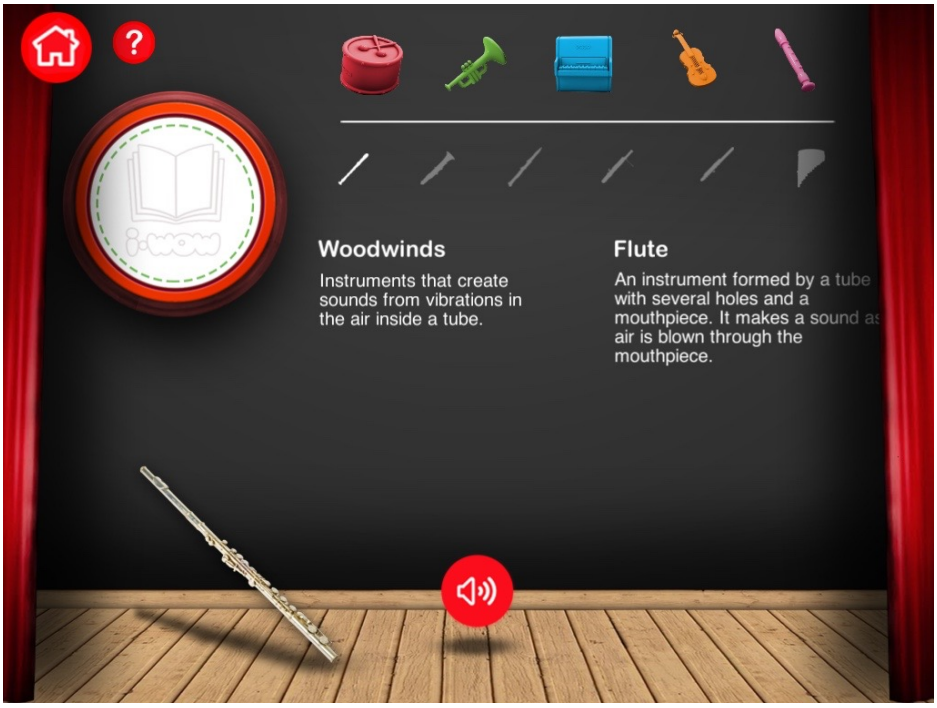

Figure B.36: Discovering flute instrument

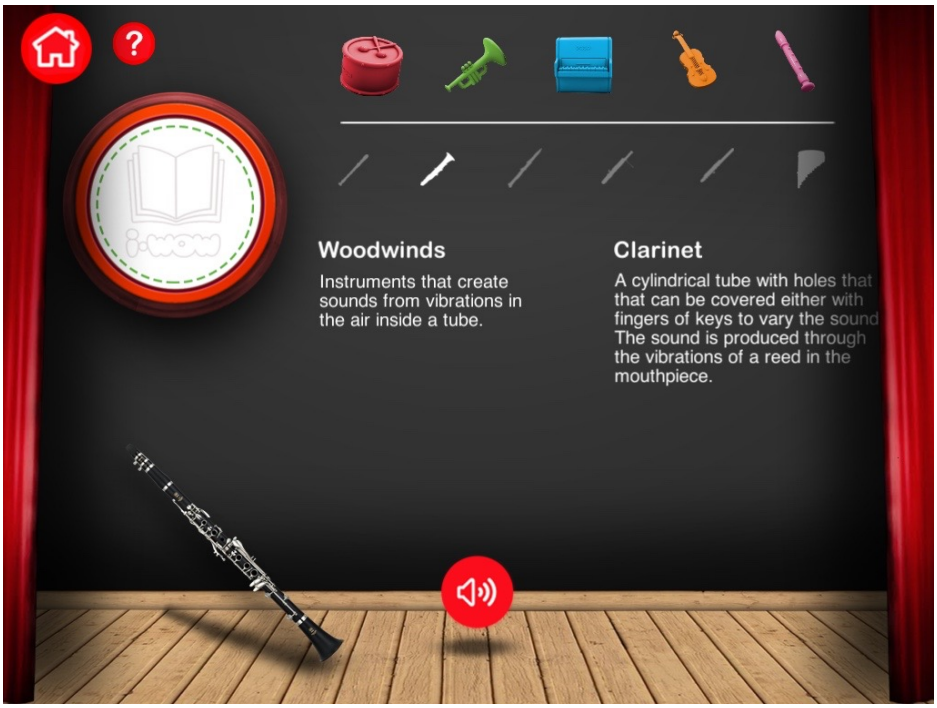

Figure B.37: Discovering clarinet instrument

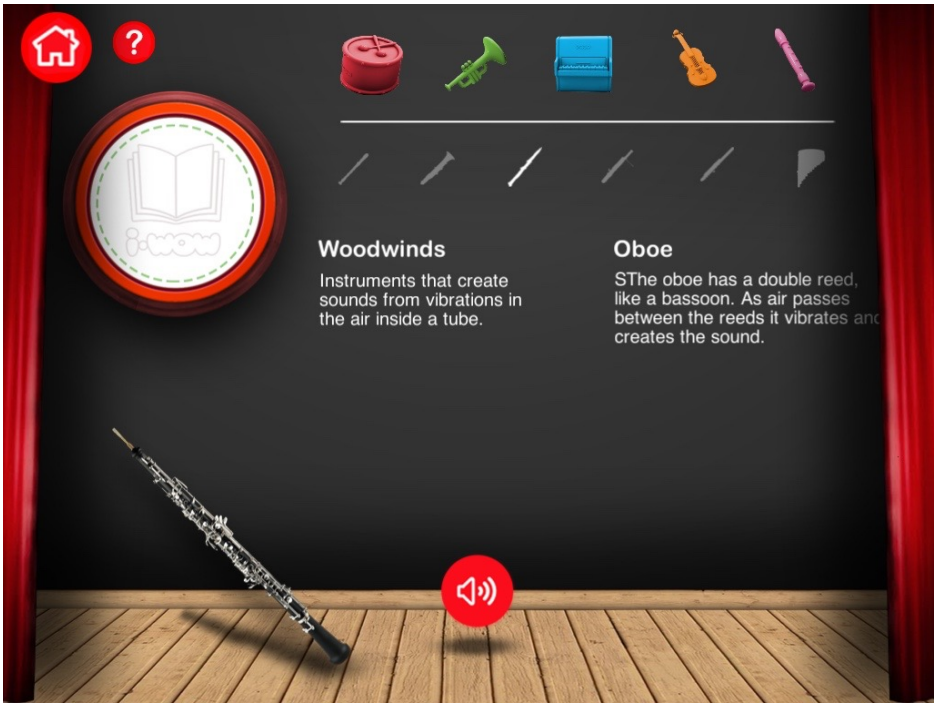

Figure B.38: Discovering oboe instrument

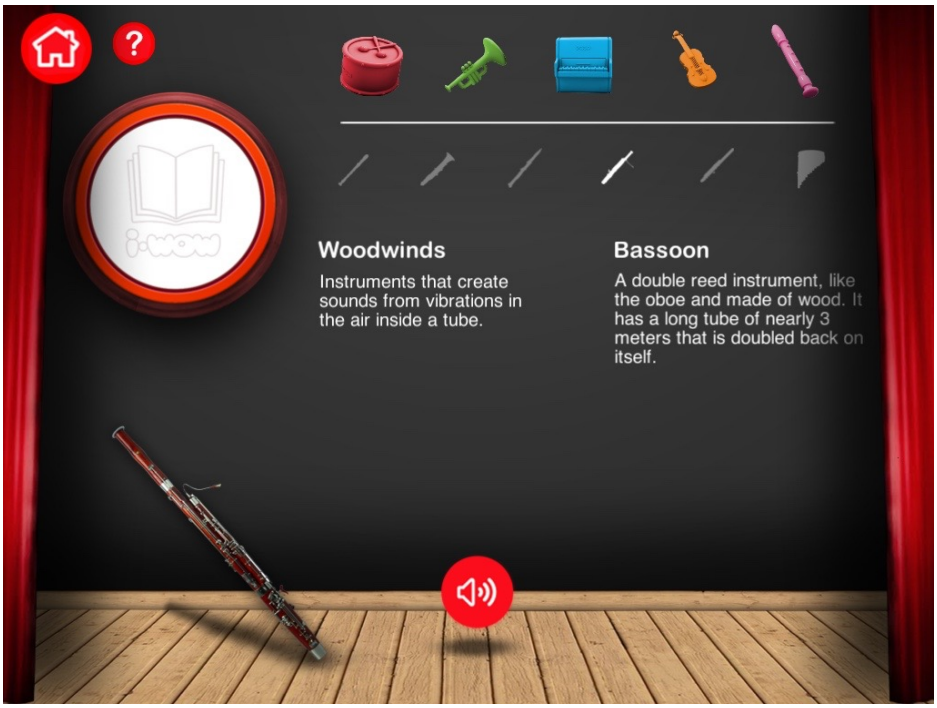

Figure B.39: Discovering bassoon instrument

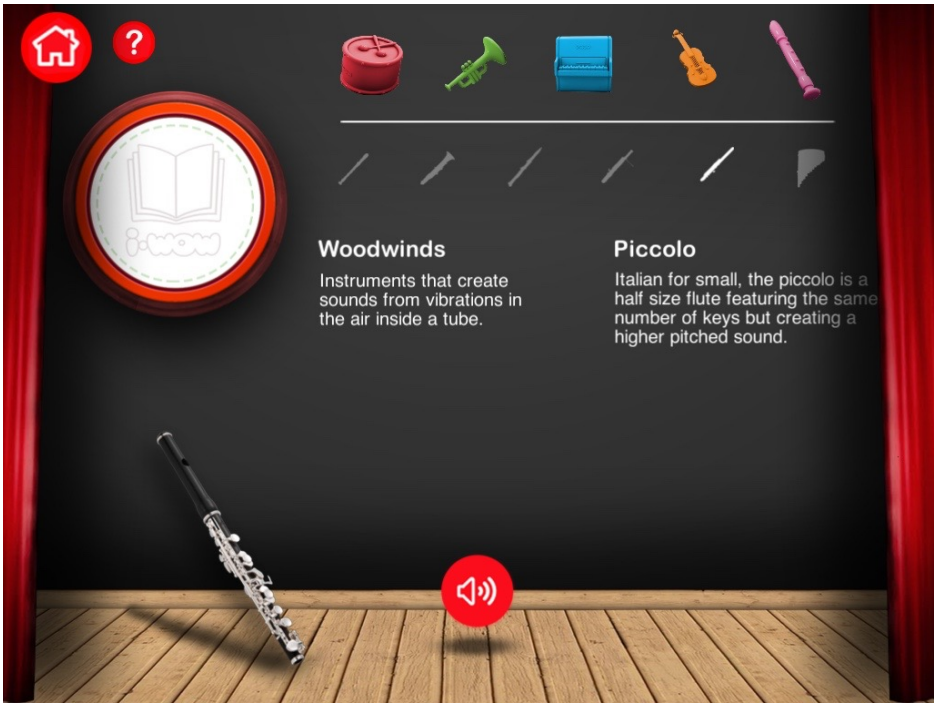

Figure B.40: Discovering piccolo instrument

Figure B.41: Discovering panpipes instrument

Bibliography

[1] T. A. M. Henning Heitkötter, Sebastian Hanschke, "Evaluating cross-platform development approaches for mobile applications," *Web Information Systems and Technologies*, pp. 120–138, April 2012.

[2] M. Yakubovich, *Evaluating the Potential of Developing Cross-Platform Mobile Applications*. PhD thesis, Chalmers university of Technology, 2013.

[3] A. Kim and J. Bae, "Development of mobile game using multiplatform (unity3d) game engine," *Social Networks*, vol. 5, pp. 29–36, March 2014.

[4] U. Technologies, "Unity - multiplatform." `https://unity3d.com/es/unity/multiplatform/`. Accessed October 14, 2016.

[5] Yahoo, "Flurry analytics / product features." `https://developer.yahoo.com/analytics/features.html`. Accessed October 14, 2016.

[6] F. Messaoudi, G. Simon, and A. Ksentini, "Dissecting games engines: The case of unity3d," in *2015 International Workshop on Network and Systems Support for Games (NetGames)*, pp. 1–6, Dec 2015.

[7] U. Technologies, "Unite 2015: 2015 unity awards winners revealed." `https://unity3d.com/es/company/public-relations/news/unite-2015-2015-unity-awards-winners-revealed`. Accessed October 14, 2016.

[8] U. Technologies, "How do i import objects from my 3d app?." `https://docs.unity3d.com/Manual/HOWTO-importObject.html`. Accessed October 14, 2016.

[9] U. Technologies, "Shaders." `https://docs.unity3d.com/Manual/Shaders.html`. Accessed October 14, 2016.

[10] U. Technologies, "Script reference." `https://docs.unity3d.com/ScriptReference/`. Accessed October 14, 2016.

[11] U. Technologies, "Monodevelop." `https://docs.unity3d.com/es/current/Manual/MonoDevelop.html`. Accessed October 14, 2016.

[12] U. Technologies, "Asset server (pro only)." `https://docs.unity3d.com/Manual/AssetServer.html`. Accessed October 14, 2016.

[13] U. Technologies, "Importing from the asset store." `https://docs.unity3d.com/Manual/AssetStore.html`. Accessed October 14, 2016.

[14] U. Software, "2d toolkit in unity 4.3." http://www.2dtoolkit.com/unity2d.html. Accessed October 14, 2016.

[15] D. Giardini, "Hotween a unity tween engine." http://hotween.demigiant.com/. Accessed October 14, 2016.

[16] U. Assets, "ios native documentation." https://unionassets.com/iosnative. Accessed October 14, 2016.

[17] U. Assets, "Android native documentation." https://unionassets.com/android-native-plugin. Accessed October 14, 2016.

[18] G. Analytics, "Analytics that empower you." http://www.gameanalytics.com/. Accessed February 14, 2017.

[19] G. Analytics, "15 metrics all game developers should know by heart." http://www.gameanalytics.com/blog/metrics-all-game-developers-should-know.html. Accessed February 14, 2017.

[20] Yahoo, "Enter the matrix: App retention and engagement." http://flurrymobile.tumblr.com/post/144245637325/appmatrix. Accessed February 14, 2017.

[21] I. Tejas Vithani, Member and A. Kumar, "Modeling the mobile application development lifecycle," in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2014 Vol I*, pp. 596–600, IMECS, 2014.

[22] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*. Wiley, 1998.