Escola Tècnica Superior d'Enginyeria La Salle

Treball Final de Grau

Grau en Enginyeria Multimèdia

Development of The Video Game Beat Repeat

Alumne

Oriol Murcia Catalan

Professor Ponent

Joan Claudi Socoró Carrié

Jordi Roig Vilaseca

ACTA DE L'EXAMEN DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Oriol Murcia Catalan

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

Development of the Video Game Beat Repeat

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Acknowledgement

Thank you to Montse, Dani and Albert for supporting me on everything I do. Thank you to my partners Pablo, Biel, Simon and Gerard for pushing me and being the best classmates. Thank you to Joan, Biel, Duarte and Oriol for helping me with the project as playtesters, thank you to them, as well as Martin and Olivia, for making me the person who I am today. Thank you to Alexandra for the help and motivation she has brought to finish this project. And thank you to Joan Claudi and Jordi for their help on the theoretical and practical part of this project.

Table of Contents

Figure index

1. Introduction

The reason why I chose this project is because it has a perfect combination of subjects and ideas that I like. Video games are a big part of our lives today: not only as entertainment but also as a way of exploration of the known universe, living out our dreams and, of course, learning. They are also an excellent way to boost creativity. Thus, on one hand, I was motivated by the idea to create something that would help children learn more about music. Besides, having the chance to create my own videogame and being able to relate it with music, sound theory, design and programming is an amazing opportunity. Being able to apply the knowledge I have acquired during the degree course is what motivates me the most. The subjects dedicated to programming in different languages have allowed me to use Unity at a very easy level. Applying different mechanics and theory of game design, user interfaces or user experience I have studied during the last four years has helped me to review the concepts and have a better grasp of the basics. And finally, I found it really interesting and challenging to apply my knowledge of sound processing to a game, and I hope to be able to use it in my future professional life.

When I started this project, I wanted to create a 2D videogame that allowed the player to create their music at a very easy level. With four music robots (instead of five now), the player is able to drag and drop different melodies to each of them. Each robot had an instrument, drums, electric bass, electric guitar and electric piano (the singer was added after). To stand out with this project, I wanted to create something unique and make the player have a unique experience with the real-time reaction of different elements to the music. I also wanted to create the drawings and animations of the characters, but after thinking about it I concluded that it would have a better result if I hired someone. Other things like UI & UX design have been implemented, as well as the playtesting part. In order to create the Story Mode (something I did not have in mind at the beginning) I had to sacrifice part of the creation of the melodies and sounds of the robots, something I will do in the future.

1.1. Introduction to videogames

Videogames are programs that are meant to have users interacting with them for different purposes like entertainment, competition, learning, etc. From the first games in the 1950s, with the big hit Pong (Version of the game Pong, 2022) in 1972, to popular games like Super Mario Bros. (Arcade Archives VS. SUPER MARIO BROS., 2017), Pokémon (Official Pokémon Website, 2022) or more recent ones like Minecraft (Official Minecraft Page, 2022), The Last of Us (Buy TLOU, 2022) or Valorant (Official Valorant Web Page, 2022). A lot of different genres exist in videogames, like RPGs (Role-Playing), shooters, RTS (Real-time strategy), puzzlers, action, survivals, etc.

Videogames have been around for some time now, but their popularity has increased in the last years. The game industry is nowadays one of the most important and innovative ones. The rise of social media is probably one of the reasons for this rise, platforms like YouTube or Twitch are plenty of influencers playing videogames with millions of viewers. The market has grown not only because of the people who buy and play videogames, but also because of the ones who watch gameplays, interact with their favorite streamer or even the ones who support their favorite eSport team. *"YouTube videos are the most influential factor on game purchase decisions followed by user reviews from communities or digital storefronts."* (Dautovic, 2022)

The market is also rising with its new technologies like new last generation consoles, the number of new mobile videogames, Virtual Reality, Augmented Reality or eSports.

To develop a game different roles and departments are needed, some of them are:

- Design department: Creates the story, the levels and decides how the game is going to be played. Some of the roles are level designer, script writer, user experience designer or gameplay designer.
- Art department: Creates the art of the game like models, textures, characters or environments. Some of the roles are concept artist, 3D modelling artist or texture artist.
- Animation department: Creates the animations for 2D and 3D art. Some of the roles are animator or rigger.
- Technical art department: Creates tools and acts like a bridge for artists, programmers and engineers. Some of the roles are technical artist or graphics programmer.
- Programming department: Creates the mechanics of the game and make the game and engine work. Some of the roles are gameplay programmer, engine programmer, artificial intelligence programmer or network programmer.
- Audio department: Responsible for composing and creating music, audio effects and programming audio. Some of the roles are music composer or sound designer.
- Quality Assurance department: Make sure everything works properly and reports the glitches and bugs in the game. Some of the roles are tester and quality assurance engineer.
- Production department: Responsible for the timelines and organization of the team. Some of the roles are marketing executive or community manager.

1.2. Learning music within the world of video games

In the videogame industry there have been some really popular games related to music like Guitar Hero (Guitar Hero Information Web Page, 2022) series or Rock Band (Official Rock Band Web Page, 2022). These games are popular for the amount of songs that can be played, the specific way they are meant to be played (with the instrument-shaped external controllers) and the number of other nice and original characteristics these games have. But little attention has been paid to the enormous potential these games have for music education.

These games offer a great opportunity to learn music for novice students. Unlike the traditional method, that sometimes can be tough and boring, this alternative method can be motivating and funnier. Unlike the traditional method, this system is focused only on learning basic concepts like rhythm, coordination or time-reaction. Playing these games is not as easy as studying music if learning how to play the piano is what you are looking for, but it is definitely a way to introduce yourself to the music world and learn the basic aspects of it while enjoying and having fun.

“...rhythm action games provide players with a sense that they are performing music and enhance the musicality of one's experience through connecting gameplay to visuals, audio, controllers, and game structures, such as the rewarding of higher point values...” (Tobias, 2012)

The aspects mentioned above are usually learned in games with mechanics where the player has to play buttons at a specific time, but other music games with different mechanics also exist. An example would be Incredibox (So Far So Good, 2022), where the player is given different melodies and he is responsible for selecting which of them fit the best in different scenarios. This selection makes the player think and discuss with himself to pick which ones fit the best by his own judgment, a very similar idea to the creation of original music. It is a very passive learning process, but this method requires musical analysis and makes the player learn critical listening.

“Campbell (2004) describes engaged listening as "the active participation by a listener in some extent of music-making while the recorded (or live) music is sounding" (p. 91). She argues that for many students, "listening must be folded into a means of interactive engagement with the music”” (Tobias, 2012)

In conclusion, any game that makes the player have an active participation with the music will have beneficial effects on new music learners and even on mid-level and professionals to finish polishing little aspects and details.

1.3. State of the art

In this section a review of different videogames similar to the one that is going to be exposed is presented. Many characteristics and mechanics are explained considering that there are games from 1998 to 2020, for consoles like smartphones, PC, Xbox, PlayStation or even Arcades.

In the final part, a comparation of the key elements from every game is made with a table to make it easier to see advantages and disadvantages.

1.3.1. Incredibox

Incredibox is a music videogame developed by the French company So Far So Good (So Far So Good, 2022). This game allows you to drag and drop different icons to the stage. What makes this app different from the others is that every sound is previously recorded with the mouth, giving this feeling of beat box. Incredibox is a game with a lot of potential that has been very successful in recent years, has won many awards and has got millions of views on different social media like Tik Tok. Thanks to its attractiveness and ease, the app has caught the attention of many people, with or without musical knowledge.

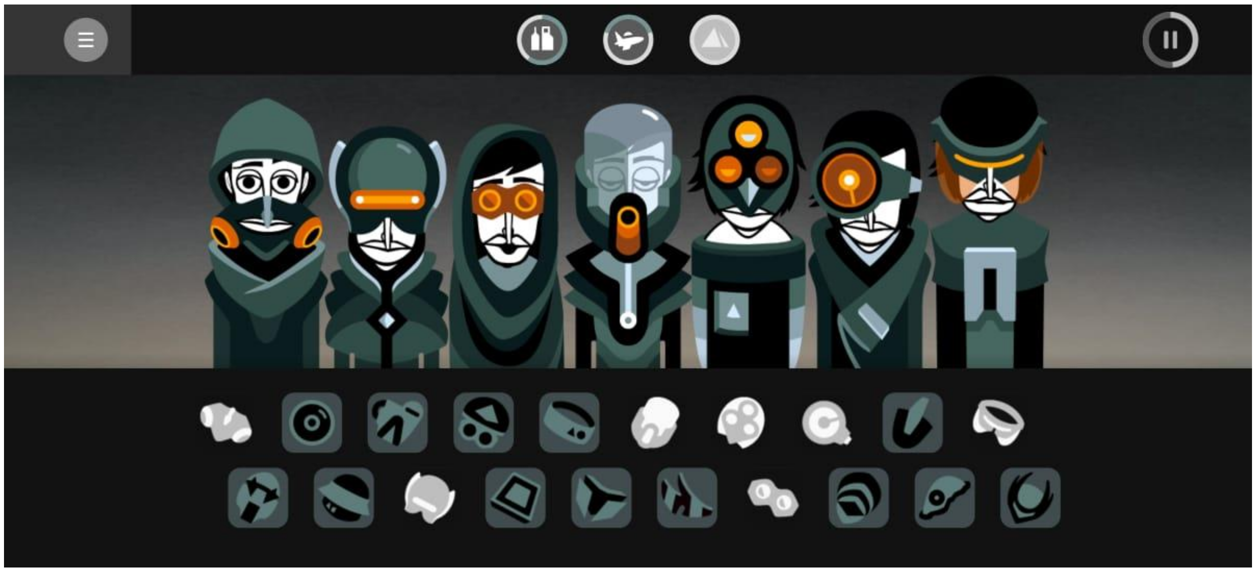


Fig 1 Incredibox Game Screenshot

Incredibox is the app where I got the most inspiration from to get my idea for the final degree project. I really like how intuitive the app is, where you drag each melody to each one of the beat boxers to play each one of the melodies. As it has been previously mentioned, this app is one of the best in the music area, but in my opinion, it is not fully exploited and has some possible improvements. For instance, the background feels a little empty. Also, having 7 characters where to drop the melodies can give the user the feeling of having too much without having the control of everything. Playing this game made me realize I have a lot of options, and I have a lot of fun with every different one, but at one point it feels like chaos and feels like you don't know what you are doing. In conclusion, playing Incredibox is a lot of fun, but after playing for some time you don't feel like learning skills or having the feeling of improvement.

1.3.2. Guitar Hero III: Legends of Rock

Guitar Hero is a music video game franchise published by RedOctane in conjunction with Activison (Guitar Hero Information Web Page, 2022). This video game is well-known for the way it is played, where you need an external guitar-shaped controller to play the music. In this game the player has 5 empty circles at the bottom part of the screen and must manage to press the button of the guitar at the exact moment when one of the filled circles that are going down comes to pass over the empty circles; the higher the accuracy, the more points the player will get, and the same for the number of combos.

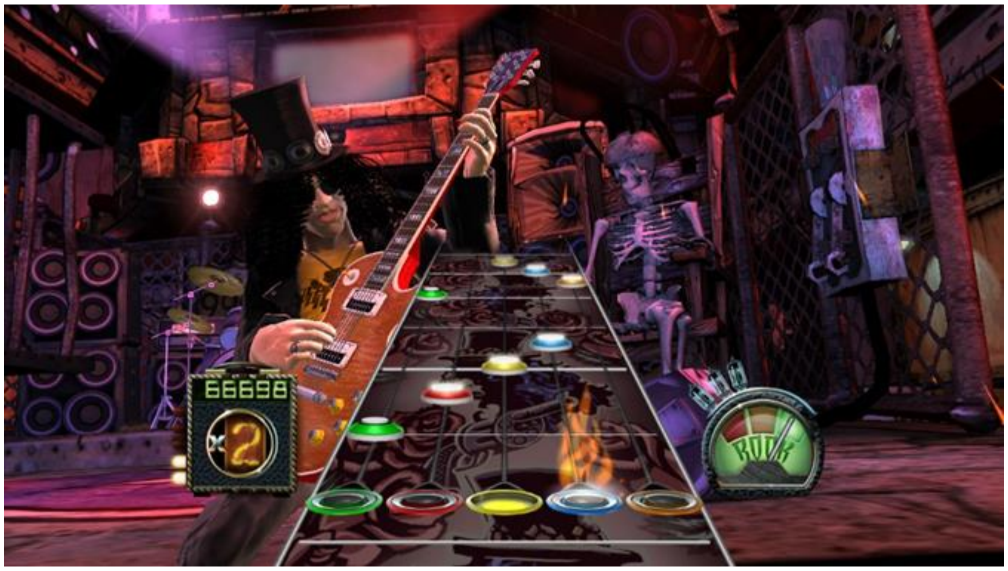


Fig 2 Guitar Hero III: Legends of Rock game screenshot extracted from [0]

Fig 3 Guitar Hero controller extracted from [1]

In Guitar Hero you unlock different songs to play depending on the score you get, which forces the player to learn and improve his skills if he wants to advance in the videogame. Also, the game has four levels of difficulty: Easy, Medium, Hard and Expert; so, people who have little or no experience can play and the ones who have a lot do not get bored. *Guitar Hero III: Legends of Rock* also has the Battle Mode, where two players play the same song and the one who scores more points wins. Another extra is the option of buying different skins for your in-game guitar, which is another reason to play and unlock your favorite guitars. One difference from Incredibox is that this game uses existing songs instead of original ones.

Guitar Hero can attract people who like rock, even if they do not know how to play a guitar in real life. A well-played song in Guitar Hero creates a feeling of immersion and satisfaction. With the visual effects and sound effects guitar hero has a polished finish that helps the player enjoy every single song.

1.3.3. Rock Band

Rock band is a music game developed by Harmonix Music Systems and MTV Games, and distributed by EA (Official Rock Band Web Page, 2022). In general, Rock Band is based on the same principles as Guitar Hero but adds more than one instrument, apart from the guitar it has a bass, drums and a microphone. This hooks the player on the game for more hours allowing him to explore the same levels using different instruments, which could be useful for my project as well.

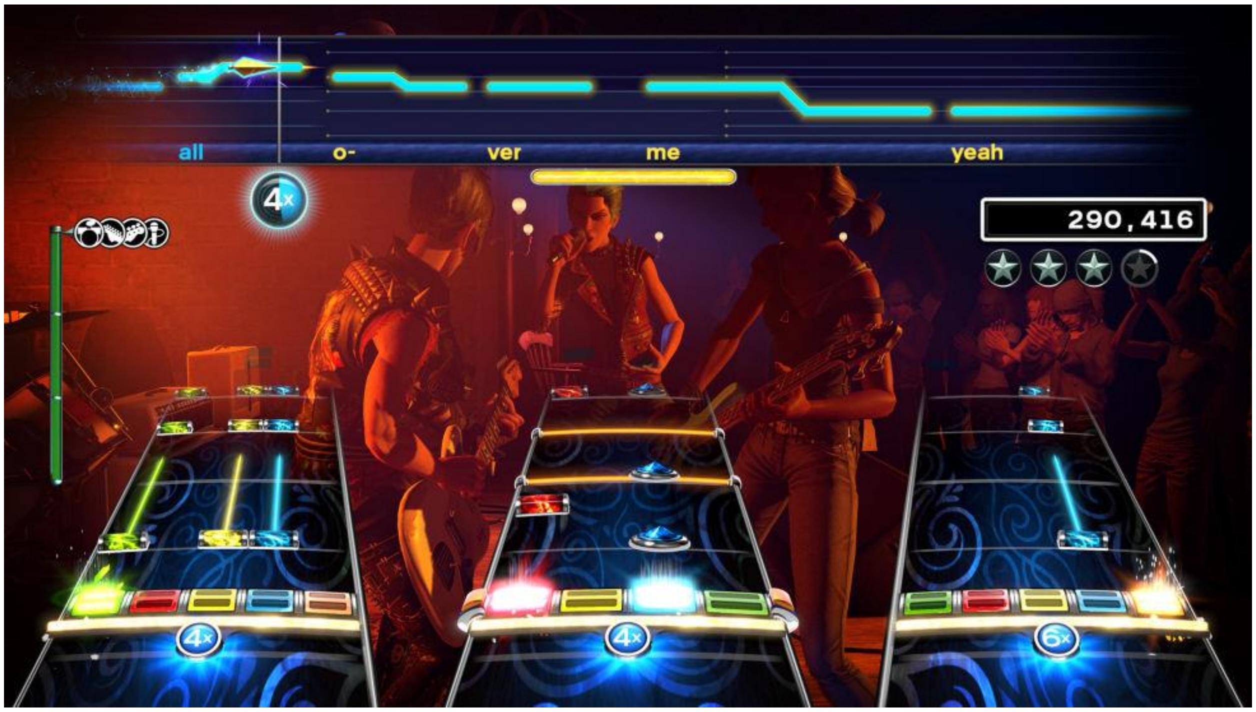


Fig 4 Rock Band game screenshot extracted from [2]

1.3.4. Dance Dance Revolution (DDR)

Dance Dance Revolution is a series of video games produced by the Japanese company Konami (DDR Official Site, 2022). This video game has a very particular way of playing it, using a dance platform and your feet. The concept is the same as Guitar Hero but dancing instead of playing a guitar. A feature of this game is the timing the player presses the notes. Depending on how much time before or after the player coincides with the notes, a message "Marvelous", "Perfect", "Great", "Good", "Almost" or "Miss" appears. The marks "Great" and higher increase your health bar, whereas the "Almost" and "Miss" decrease it. The players lose when they run out of life and win when the song finishes without losing all the health bar. DDR uses a list of video game and J-Pop (Japanese Pop) genre songs.

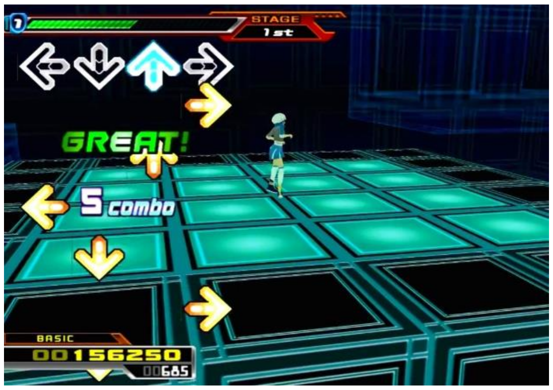


Fig 6 DDR game screenshot extracted from [3]

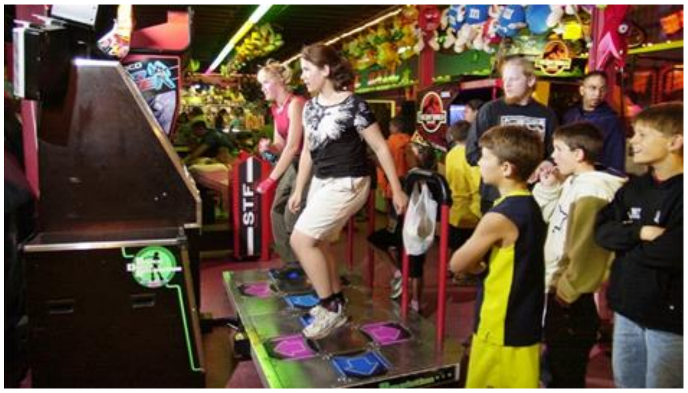


Fig 5 people playing DDR on arcade console extracted from [4]

1.3.5. Friday Night Funkin’

Friday Night Funkin’ is a music-rhythm game developed by Cameron Taylor (ninjamuffin99, 2022). In this game you have the main character defeat different enemies in singing battles. These battles consist in clicking at the exact time the keyboard keys to move the bottom bar as much to the left as possible. The game contains two different modes: the story mode in which songs are played linearly and the story of the main character is told between battles, and a "free play" mode which allows the players to play the track they want.

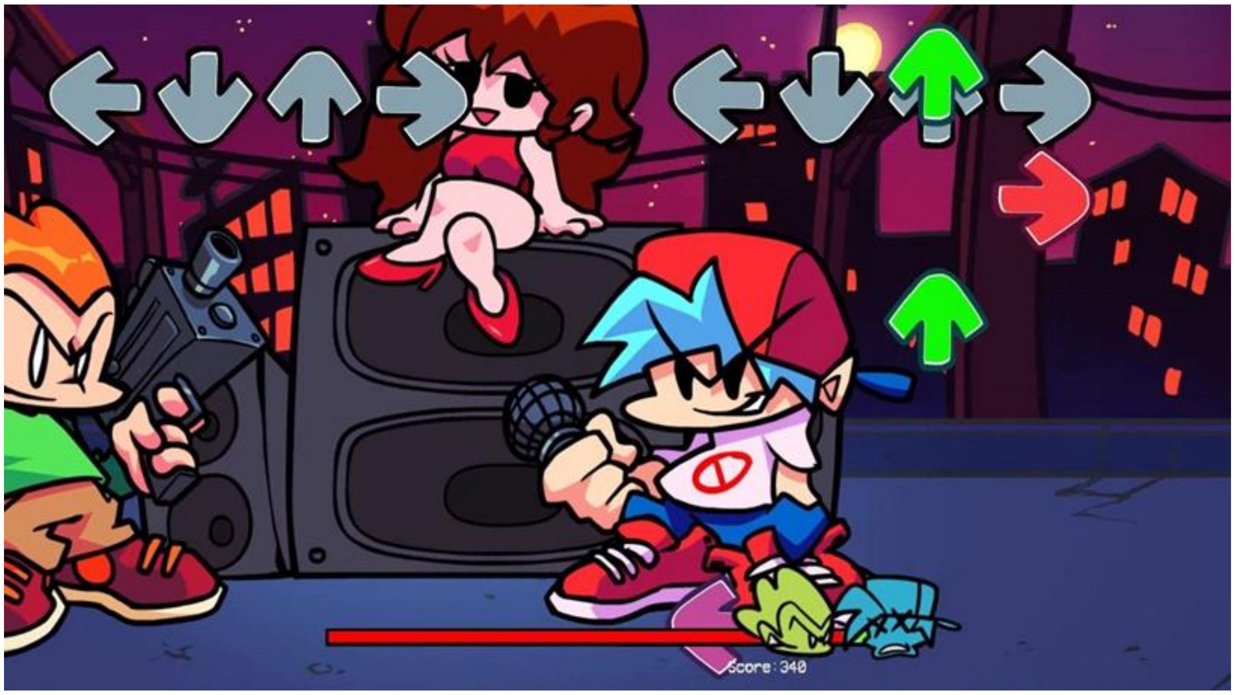


Fig 7 Friday Night Funkin' game screenshot

1.3.6. Comparison between music videogames

In conclusion, all these games know how to adapt to the console they are made for and in every case they have their unique trait that makes them differentiate from every other game.

In the different music videogames reviewed in the previous sections are compared in terms of the level the player must have in order to play them, the different platforms of these games and also such additional features that motivate the player as unlockable levels, as a background story, items to buy and player ranking.

Game	Incredibox	Guitar Hero	Rock Band	DDR	FNF
Elements to borrow	Original Music Drag and drop system Free Mode Characters	Controlled Mode (falling circles) Combos	Different instruments	"Great", etc. messages in the Controlled Mode	Uniqueness of aesthetics and characters The story part, not a complete story, but short stories or dialogs for the characters to give the player more information about them
Play Style	Free Mode	Gameplay	Gameplay	Gameplay	Gameplay
Level	Easy	From easy to expert	From easy to expert	From easy to expert	Normal
Platforms	Smartphone and PC	PlayStation 2, PlayStation 3, Xbox 360 and Wii	PlayStation 4, Xbox One, PlayStation 3, Wii, Xbox 360, PlayStation 2, Nintendo DS, iOS	Arcade, Nintendo's, Smartphones, PlayStation's, Xbox's, etc.	PC
Items to buy	No	Yes	No	No	No
Story or unlockable levels	No	Yes	Yes	Yes	Yes
Player Ranking	Yes	Yes	Yes	Yes	No

Table 1 Comparison table between the games mentioned in the State of the Art

It is also worth mentioning that some of this type could be developed with more interesting backgrounds, adding elements that react to the music, making the player more interested not only in the listening part but also visually. This is something I would like to work on for the final release of the game in the future.

1.4. Objectives

This project consists in the development of a 2D mobile game made with Unity. This application, named Beat Repeat, is a music videogame with electronic music as its main concept. The objective is to create your own music at a very basic level with the help of 5 robot characters. Each character has its own instrument, there are the drums, the electric bass, the electric guitar, electric piano and the robotic singer. Each robot has 5 different melodies to choose from, and each melody combines perfectly with each other melody from the other robots. Also, these robots have the ability to change the aspect of their instrument, changing its sound too. Another characteristic of the game is the possibility of applying different effects like reverb or filters. The idea is to entertain users and make them have fun creating and trying new and original combinations with the music, trying to find which combinations they like the most. This part is also beneficial for children and teenagers to be able to learn and practice different music skills like critical listening or rhythm.

Another important point this game has is the Story Mode, where the player can go through different levels pressing and using different buttons and melodies when told, like a guitar hero game but with mechanics adapted to the game. These levels will tell a story, giving more background to each of the robots. This part can be a little difficult on the top levels, so it is perfect for more hardcore players.

This game is free to play but has different packs and the player can choose whether to pay for them or not. These packs are inspired by different thematic and contain different melodies, sounds and levels.

Since the beginning of this project, there were different main objectives I wanted to accomplish with it:

- Creating a music videogame.
- That this videogame not only entertains but also teaches music.
- Use Unity as game engine.
- Create my own designs for the game.
- Produce the music of the game by myself.
- Have a unique characteristic that makes it differentiate from other games, for example, that the scenery reacts to the music in real-time.

The main objective is creating a videogame app that uses music as the main mechanic. The idea with this app is it entertains players while they create their music and play at the same time, but also, an important part of the app is focused on teaching music skills on people, mostly children, who are starting in the music world. So, with these two main ideas, the target of the game goes from 7 to 25 years old. Within this objective, different subobjectives appear. One of them is learning and developing my skills on game engines, in this case Unity. As I want to dedicate myself professionally to developing videogames, understanding and developing games with game engines is essential. Another objective is that, as I want to create my own designs for the game, I need to improve my artistic skills working hard and practicing a lot. Also, as one of my main hobbies since I was 14 has been producing music, I wanted to have the opportunity of adding my own music in my own game, and since this game is focused on creating music, I can create as music as I want for the development of the game. Finally, one final objective is creating elements that react to the music of the game, for this part it is important to use audio processing theory learnt through the degree course and learn and apply new concepts of it.

I am really motivated and I am sure all these objectives, as well as other possible ones that appear during the process of creation of the project, will come out with a great result.

1.5. Document contents

The chapters of this memory are distributed as, first, in chapter 2, an explanation of different concepts of the project is made to better understand every part of this document. This explanation is about different topics like, the game engine, Unity, used to develop this game, various theory concepts about signal and audio processing, explanation about different ideas of game design, user interface and user experience in videogames and an explanation of the software FL Studio, used in this project to create the music of the game. Then, in chapter 3, a deep insight about the game created for this project is done, their options, functionalities and its possibilities. After understanding the game and all the main concepts of the project, in chapter 4 it is explained how has been developed the mechanics of the game, the creation of the designs and art, the methodology followed to produce the music, and the creation of the shaders and animations inside Unity. After finishing with this chapter, in chapter 5, it is analyzed the game experience and playtestings of various players, with the extracted results, possible improvements for the game are mentioned. To finish, an economic study is made in chapter 6, as well as the conclusions of the project and the future plans for the game in chapter 7. It is also important to mention the Bibliography, where you can find all the documentation used to check and compare information used in this document, as well as the Index I, which contains all the citations of the document, and Index II which contains all the sources of the images used for the realization of this project. The use of these indexes is to mention and recognize the work of others that helped me in some parts of this project.

2. Theoretical and practical foundations

In this section a review of the main theoretical and practical foundations of every subject and software used in this project is presented. Every aspect and concept that will be referenced in the next sections is explained here.

Specifically in this section an introduction about the design of videogames using Unity, the game engine used to create this project, will be explained make understand all concepts before talking about how the game has been developed. Also, there is a need to deepen in concepts related with signal processing, and more specifically with audio signal processing which is the topic that will be discussed in order to allow the user of the videogame to interact with music effects like the reaction of the music at the background.

2.1. Unity

2.1.1. What's Unity?

Unity is a 3D/2D game engine and powerful cross-platform IDE (integrated development environment) for developers. It came out in 2005 and it is one of the most famous free-to-use game engines, meaning everyone can download it, use it, and publish games made on it for free, considering that Unity will take a percentage of your revenue. Examples of popular video games developed with Unity are Cuphead (Buy Cuphead on Steam, 2022), Fall Guys (Official Fall Guys Web Page, 2022), Beat Saber (Beat Saber Official Web Page, 2022) or Rust (Rust Official Web Page, 2022).Unity provides the users with many complex and key features that allow them to create videogames at every level. Developers do not need to start from zero creating new tools or new physics systems to begin with their project. Unity is also easy to use, and you can find a lot of information, videos and related questions about it on the internet, which makes life much easier for beginner developers.

Unity has a way of working I have never seen before on other programs. In Unity, the developer not only has to write the code, but he also needs to interact with the editor screen to make sure everything is correctly connected. This is called IDE, integrated development environment, and refers to an interface that grants access to all the tools in one same screen. For example, the user can add scripts to GameObjects with a simple drag and drop system, or even drop different GameObjects to public variables. This editor screen also offers the ability to navigate through the different folders of the project or even create animations with a simple timeline tool, among other possibilities. However, the developer must switch to an alternative editor, to program the code in C# language.

Fig 8 Unity logotype extracted from [17]

2.1.2. Editor in Unity

In this section an explanation of the functioning for different mechanics, characteristics and items is explained. All aspects exposed in this section will be used for the creation of the videogame and will be talked about in the next sections.

2.1.2.1. Main tabs

In this section the main editor windows used in this Unity project are exposed.

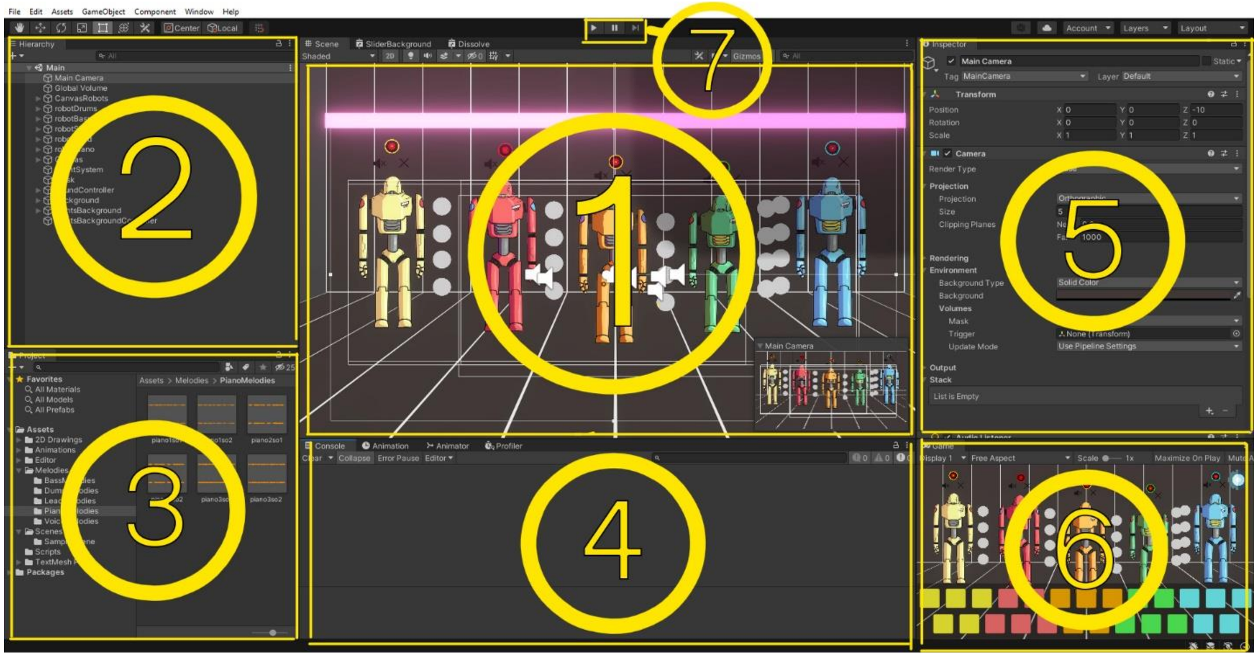


Fig 9 Unity IDE Interface

1: The scene view is the way the user interacts with the game world viewing, selecting and modifying the different characters, cameras, lights and all other types of GameObjects. Some of the things you can do with the different GameObjects are transformations involving moving, sizing or rotating. Most games have more than one scene and here we can see the selected one.

2: The hierarchy tab shows every GameObject in the scene we are currently working on. Same as scene view but as a list of objects you can select without being able to modify them. Parenting different GameObjects is a characteristic we can work with in this window. Parenting is a method Unity uses to group different GameObjects. With this method GameObjects have the possibility to have other GameObjects as children, siblings or parents, making it easier for the developer to work on the project with different functions. In this tab we can also sort and search GameObjects in different ways such as alphabetically or depending on the type of the GameObject.

3: In the project window the user is able to navigate through the different folders of the project, an easy way to find and drag and drop different objects to the game.

4: The console view displays all types of errors, warnings or messages your game has. This will help the user find what problems his game has. To print your own messages in the console, use the Debug class through the different parts of your code.

5: The inspector window allows the user to view and edit all the properties and components every GameObject, asset, material, etc., has. A commonly used function is dragging and dropping different GameObjects to the public variables of a script the selected object has. In this tab the user can also add all kinds of components to the object selected.

6: In the Game view you will be playing with the game. It renders the camera/s on your application and represents the final product of your game. This window is directly related to the number 7.

7: These 3 buttons (play, pause and step) allow the user to play/stop the game, pause it or play one frame of the game when paused.

*All these windows are part of the Editor UI, which can be customized as the user prefers. They can be moved to whichever part of the screen and more tabs with other functionalities can also be added.

2.1.2.2. Other tabs

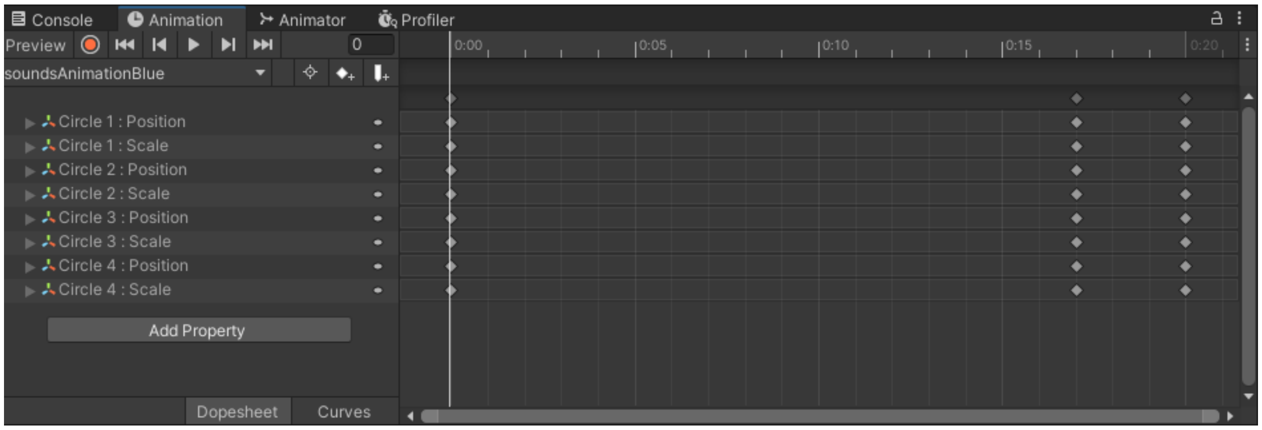


Fig 10 Animation tab in Unity

The animation tab (see Fig 10) is used to preview and modify animations for different GameObjects. This tab displays a timeline of the animation where the user can add keyframes to different parameters of the object to make them change through the time.

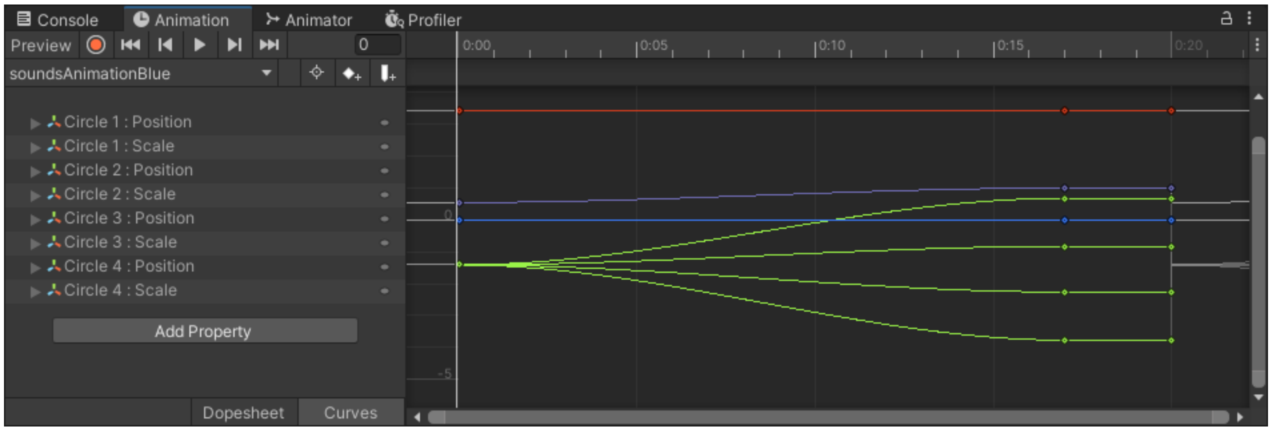


Fig 11 Curves panel of the animation tab

In the animation panel, the Curves window, where you can change velocities of the different parameters to make, for example, smoother transitions, is also very useful.

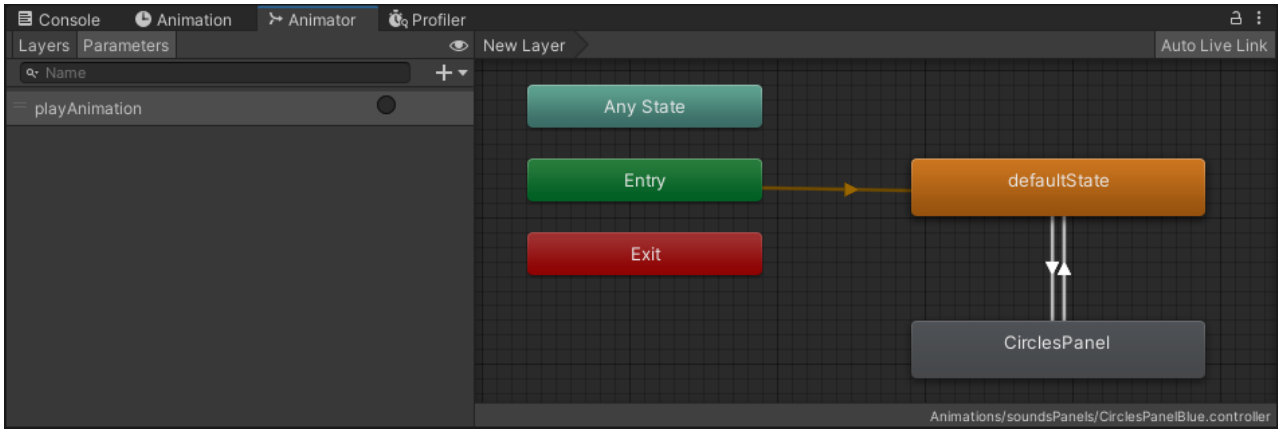
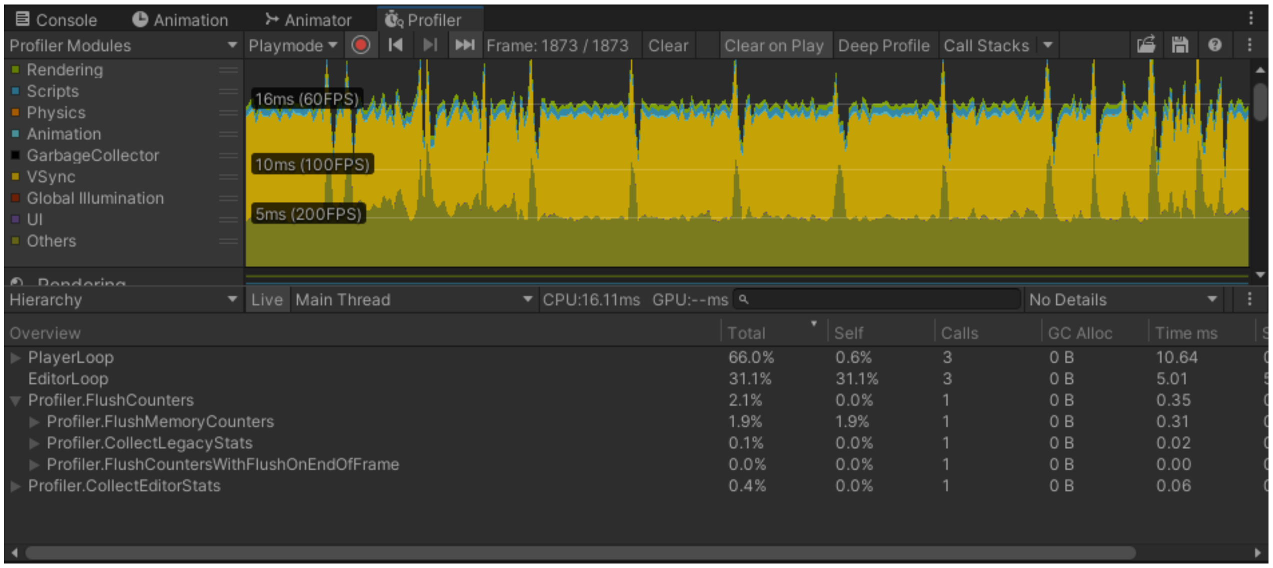


Fig 12 Animator tab in Unity

The clips the user creates in the Animation tab (see Fig 12) can be combined and mixed using the Animator. The Animator window allows the user to control which animation is played using different conditions and parameters he specifies. To sum up, in this panel the user can create different states, with different directions and transitions that depend on different parameters to create the path he prefers to control any kind of animation.



Frame: 1873 / 1873

CPU:16.11ms GPU:--ms

Overview	Total	Self	Calls	GC Alloc	Time ms
PlayerLoop	66.0%	0.6%	3	0 B	10.64
EditorLoop	31.1%	31.1%	3	0 B	5.01
Profiler.FlushCounters	2.1%	0.0%	1	0 B	0.35
Profiler.FlushMemoryCounters	1.9%	1.9%	1	0 B	0.31
Profiler.CollectLegacyStats	0.1%	0.0%	1	0 B	0.02
Profiler.FlushCountersWithFlushOnEndOfFrame	0.0%	0.0%	1	0 B	0.00
Profiler.CollectEditorStats	0.4%	0.0%	1	0 B	0.06

Fig 13 Profiler tab in Unity

The Profiler tab (see Fig 13) is another very used tab that shows every aspect of the game's performance. This allows the user to know which parts of the game might be optimized and to confirm which other aspects of the game are working as desired.

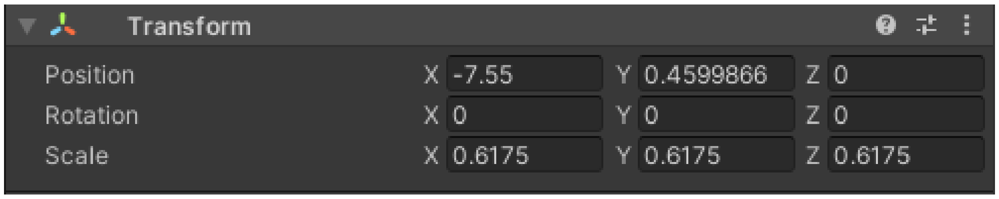
2.1.3. Components in Unity

In this section an introduction to the components of Unity is made, as well as an explanation of some of the most important and used components for this project.

The components in Unity are pieces that are attached to GameObjects to make them have a certain functionality or behavior. These components are made of distinct properties which, some of them, can be edited by the user. All GameObjects can have as many components as the user wants, in order to create complex behaviors for specific objects, a complex system of components is required. That is one of the main functionalities of Unity developers for example, creating new and complex components to create game mechanics, implement functionalities or bring to life objects from Unity.

Some of the most important components are:

- Transform: Each GameObject in Unity has one Transform component. This component is used to modify the position, scale or rotation in the x, y and z axis of the GameObject.

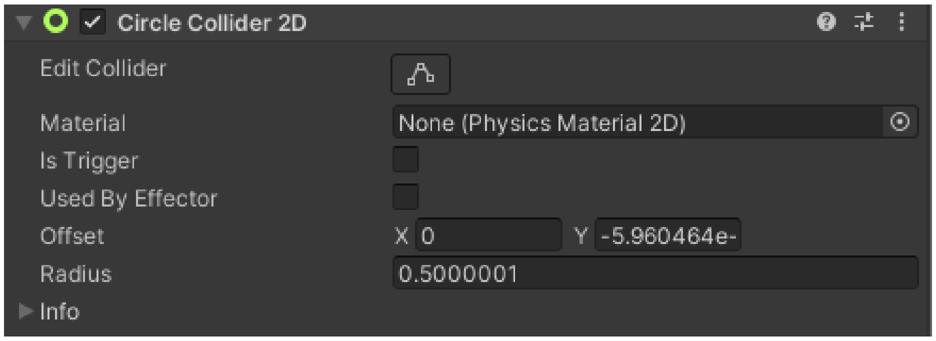


Transform

	X	Y	Z
Position	-7.55	0.4599866	0
Rotation	0	0	0
Scale	0.6175	0.6175	0.6175

Fig 14 Transform component in Unity

- Colliders: In Unity, the colliders components are responsible of handling collision between GameObjects that have a collider on them. A collider is modifiable area (2D) or volume (3D) that is invisible.

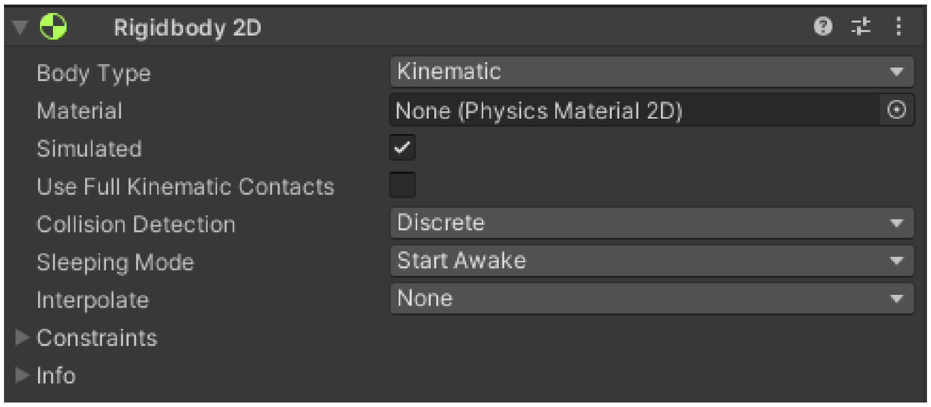


Circle Collider 2D

Edit Collider	
Material	None (Physics Material 2D)
Is Trigger	☐
Used By Effector	☐
Offset	X 0 Y -5.960464e-
Radius	0.5000001
▶ Info	

Fig 15 Circle Collider 2D component in Unity

- Rigidbody: Rigidbody components allow the user to simulate physics on a GameObject. This component has different properties that facilitate the real simulation of physics inside Unity.

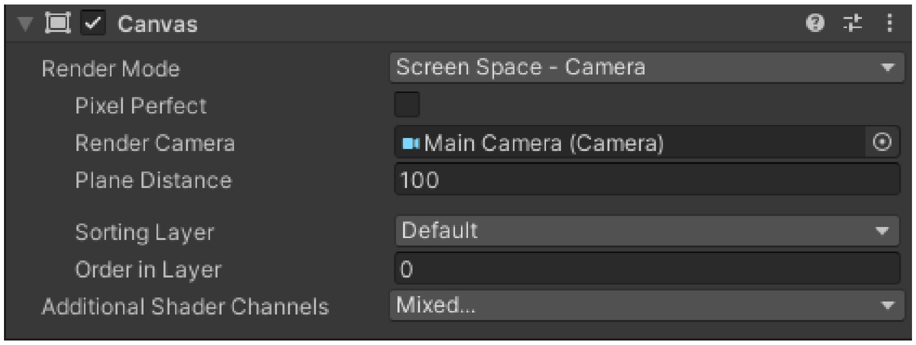


Rigidbody 2D

Body Type	Kinematic
Material	None (Physics Material 2D)
Simulated	✓
Use Full Kinematic Contacts	☐
Collision Detection	Discrete
Sleeping Mode	Start Awake
Interpolate	None
▶ Constraints	
▶ Info	

Fig 16 Rigidbody 2D component in Unity

- Canvas: The GameObject with a Canvas component is where usually all UI GameObjects are inside of. This component allows the user to sort the UI components easier, as well as changing different settings as the space where all these components will be projected.

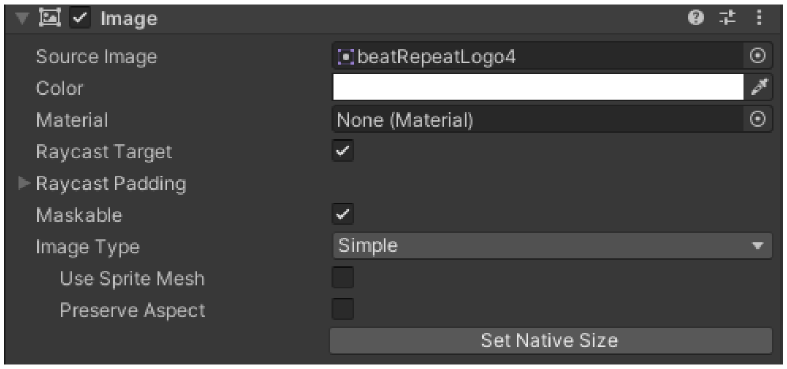


Canvas

Render Mode	Screen Space - Camera
Pixel Perfect	☐
Render Camera	Main Camera (Camera)
Plane Distance	100
Sorting Layer	Default
Order in Layer	0
Additional Shader Channels	Mixed...

Fig 17 Canvas component in Unity

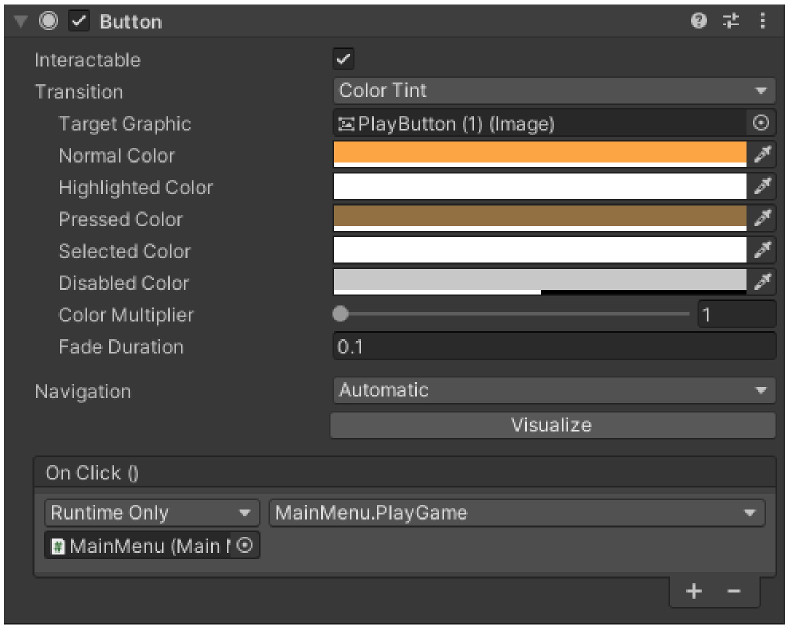
- UI components: other interesting components used in this project are the UI components. These components are made to create UI objects in a very easy way, some examples would be buttons, images or text.



Image

Source Image: beatRepeatLogo4
Color
Material: None (Material)
Raycast Target ✓
Raycast Padding
Maskable ✓
Image Type: Simple
Use Sprite Mesh
Preserve Aspect
Set Native Size

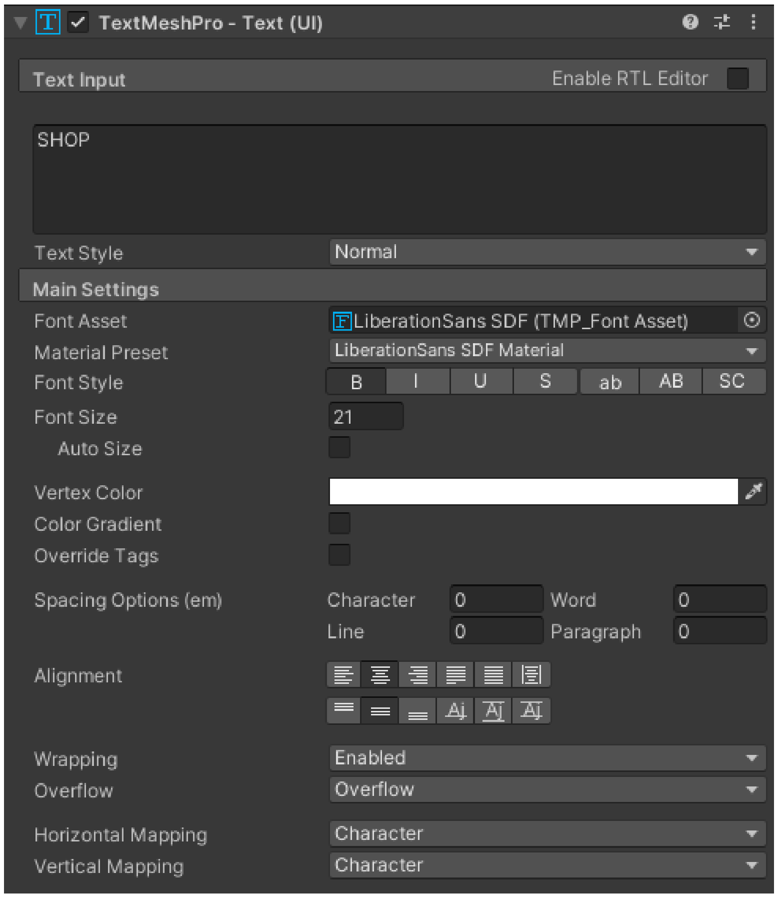
Fig 18 Image component in Unity



Button

Interactable ✓
Transition: Color Tint
Target Graphic: PlayButton (1) (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier: 1
Fade Duration: 0.1
Navigation: Automatic
Visualize
On Click ()
Runtime Only — MainMenu.PlayGame
MainMenu (Main I
+ −

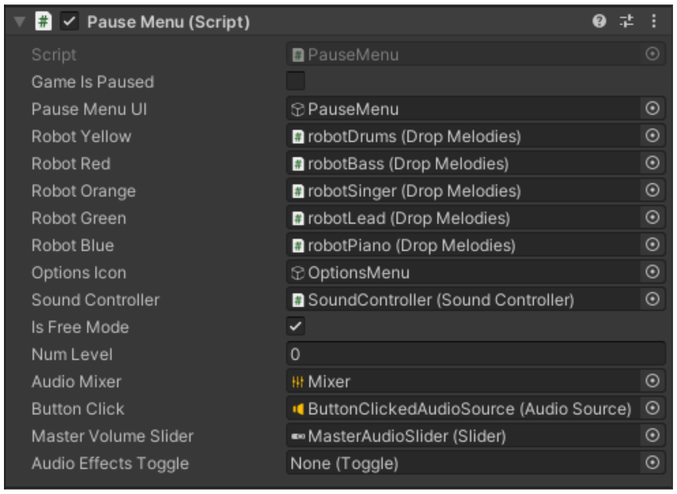
Fig 19 Button component in Unity


TextMeshPro - Text (UI)

Text Input	Enable RTL Editor
SHOP	
Text Style	Normal
Main Settings	
Font Asset	LiberationSans SDF (TMP_Font Asset)
Material Preset	LiberationSans SDF Material
Font Style	B I U S ab AB SC
Font Size	21
Auto Size	
Vertex Color	
Color Gradient	
Override Tags	
Spacing Options (em)	Character 0 Word 0
	Line 0 Paragraph 0
Alignment	
Wrapping	Enabled
Overflow	Overflow
Horizontal Mapping	Character
Vertical Mapping	Character

Fig 20 TextMeshPro component in Unity

- Scripts: Scripts are probably the most important components in Unity. By attaching scripts to GameObjects, Unity allows the user to create their own components.


Pause Menu (Script)

Script	PauseMenu
Game Is Paused	
Pause Menu UI	PauseMenu
Robot Yellow	robotDrums (Drop Melodies)
Robot Red	robotBass (Drop Melodies)
Robot Orange	robotSinger (Drop Melodies)
Robot Green	robotLead (Drop Melodies)
Robot Blue	robotPiano (Drop Melodies)
Options Icon	OptionsMenu
Sound Controller	SoundController (Sound Controller)
Is Free Mode	✓
Num Level	0
Audio Mixer	Mixer
Button Click	ButtonClickedAudioSource (Audio Source)
Master Volume Slider	MasterAudioSlider (Slider)
Audio Effects Toggle	None (Toggle)

Fig 21 Example of Script component in Unity

As shown, there is a big list of default components Unity provides users that makes programming too much easier, even so, scripts is what usually takes most of the work when developing a videogame.

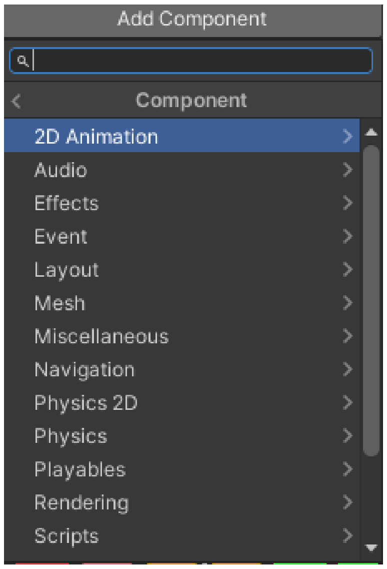


Fig 22 List of types of components inside Unity

2.1.4. Shaders in Unity

In this section a short explanation on what are shaders and how to use them in Unity is made.

2.1.4.1. What are shaders

Shaders are programs that run on GPUs that contain mathematic calculations and algorithms to calculate the color of each pixel rendered in a screen.

There are two basic types of shaders:

- The Vertex shader: Places the vertices on the screen.
- The Fragment shader: Calculates the color of the pixels on the screen.

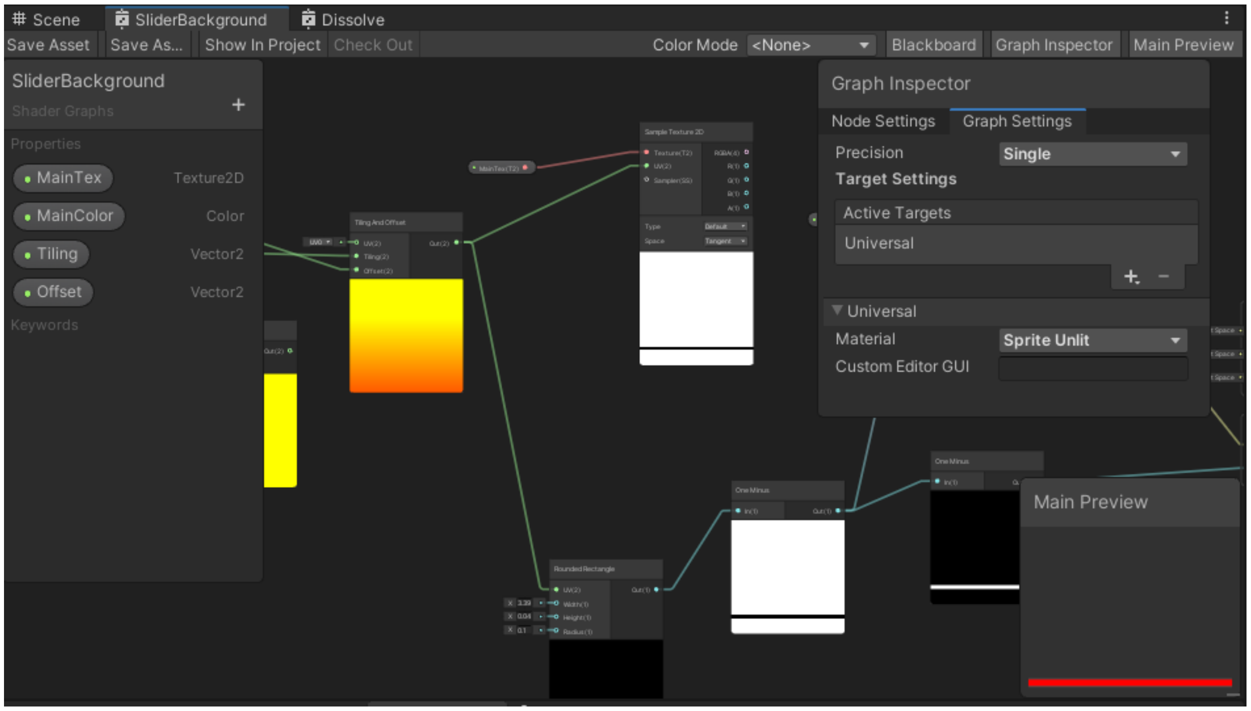
But Unity has another type of shader, the Surface shader. With Surface shaders it is like having a Fragment and Vertex shader, but you don’t need to do lighting calculations.

2.1.4.2. Shader Graph

Shader Graph is a tool that allows users to create their own shaders visually instead of writing their code. With this tool you can create and connect different nodes that adjust your shader the way you are looking for.

For this, we need a Render Pipeline. There are different Render Pipelines you can download with your Package Manager in Unity; in my case I am using the Universal Render Pipeline.

Once we have our Render Pipeline configured, we just have to create a new material and open it with the Shader Graph.



Color Mode <None>

MainTex	Texture2D
MainColor	Color
Tiling	Vector2
Offset	Vector2

Precision Single
Target Settings
Active Targets
Universal
Universal
Material Sprite Unlit
Custom Editor GUI

Fig 23 Shader graph in Unity

A lot of different nodes can be created and connected in many different ways. Any kind of texture you can imagine can be created with this method. From a simple flat color to a high-realistic real-time reacting water texture. Unity offers all kinds of nodes like noise, mathematic functions, masks, adjustment layers, etc.

On the left side of the screen (see Fig 23), we can see a panel with our public properties. These properties can be connected to different nodes and can be directly modified from the Editor or even via script.

2.1.5. Audio in Unity

In this section different concepts and methods to use audio in Unity is made.

2.1.5.1. Introduction to sound in Unity

All games need audio in it, it can be background music or sound effects, but we can assure that a game is incomplete with no sound. That is why Unity has two main audio components, the Audio Source and the Audio Listener.

Audio Sources are objects that emit sounds, and these sounds will be heard by the Audio Listener. Audio Sources can play sounds in 2D, for example for background music, and sounds in 3D, for 3D set objects. So, the difference between them is that the Audio Source will emit the sound, and the Audio Listener will listen to those sounds to then send it to the soundcard of the computer.

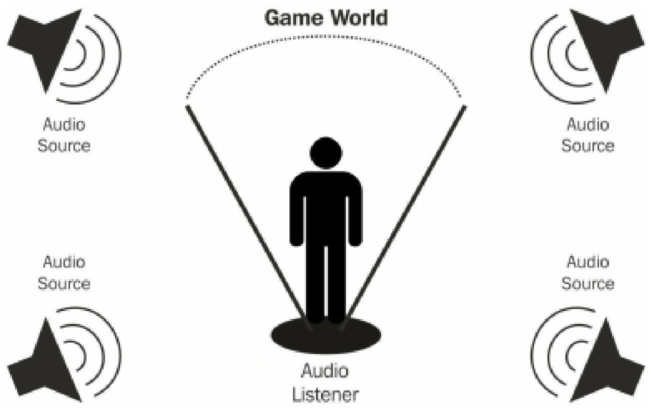


Fig 24 Schema to easily explain how audio sources and audio listeners work on Unity

Typically, the Audio Listener is attached to the main camera because it is where the player is watching from.

Having a lot of sounds and Audio Sources in one scene can sometimes be tough and complex, that is why we use the Audio Mixer.

2.1.5.2. Audio Mixer

The Audio Mixer is a tool (see Fig 25) that allows the user to organize and mix various Audio Sources.

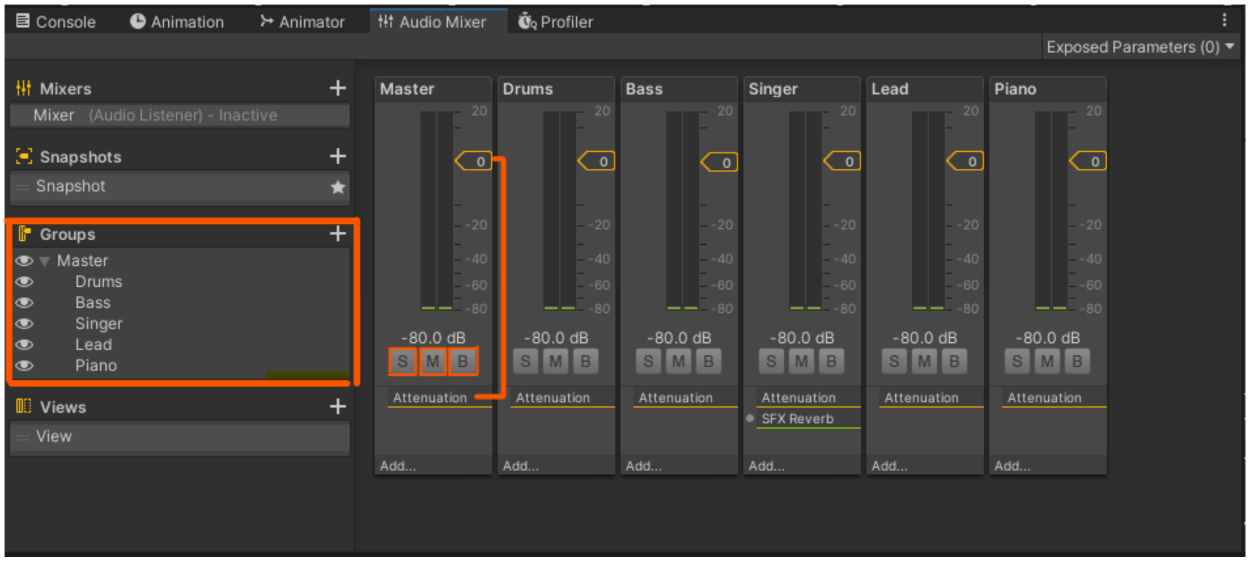


Fig 25 Audio Mixer in Unity

This window shows different channels, and each of these channels corresponds to one Audio Source. As you can see in the Groups section on the left side of the screen, all these channels are inside one Master channel, that means that all these channels go through the Master before going to the Audio Listener.

The “S” button refers to a Solo function, that means that all other channels than that will be silenced. The “M” button refers to a Mute function, muting the selected channel. And the “B” button refers to a Bypass function, which will bypass all the audio effects in that channel. Finally, as a default audio effect we have the Attenuation, which allows us to turn up or down the volume of that channel.

More than one Mixer can be created, that means that different groups of Audio Sources can be treated differently. Another possibility Unity offers is linking one Master of one Mixer to one channel of another Mixer, that’s useful for example when we want to mix music separately from sound effects.

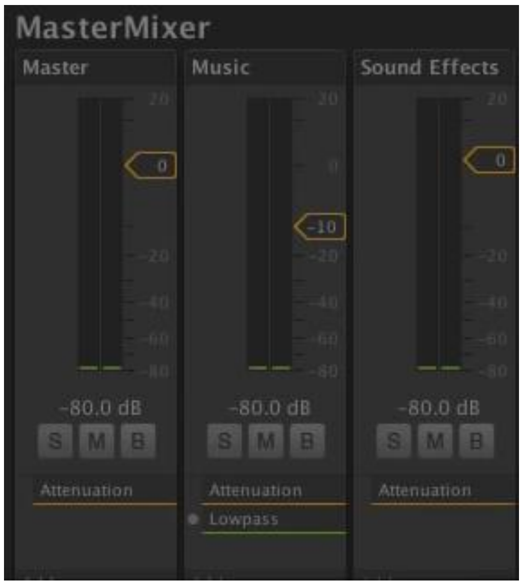


Fig 26 Example of group of channels in Unity

Another interesting implementation this tool offers is the possibility of adding different audio effects to the channels, like reverb, filters or compression.

One important parameter in this section of audio effects is the duck volume effect. The duck volume effect, also known as side chain compression, is an effect that varies the volume of one channel depending on the current volume of another channel. The way it works is simple, you just have to link the Duck Volume effect to the Send parameter of another channel.

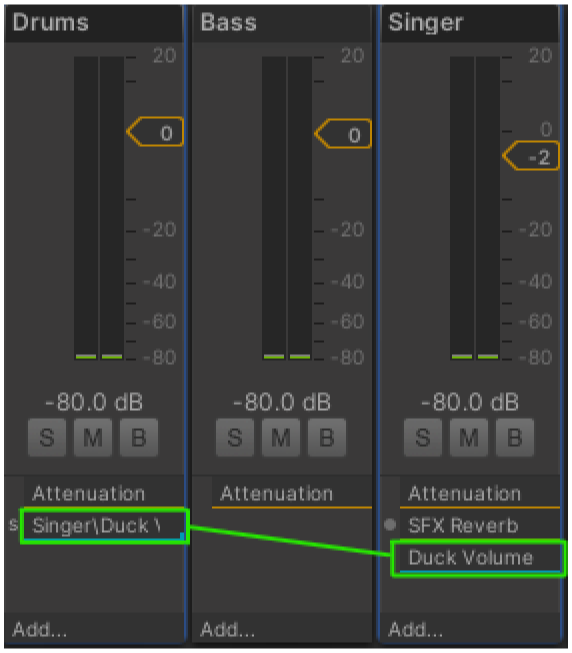
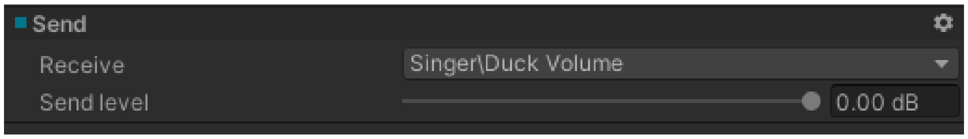


Fig 27 Example of the Duck Volume effect linked to another channel in Unity



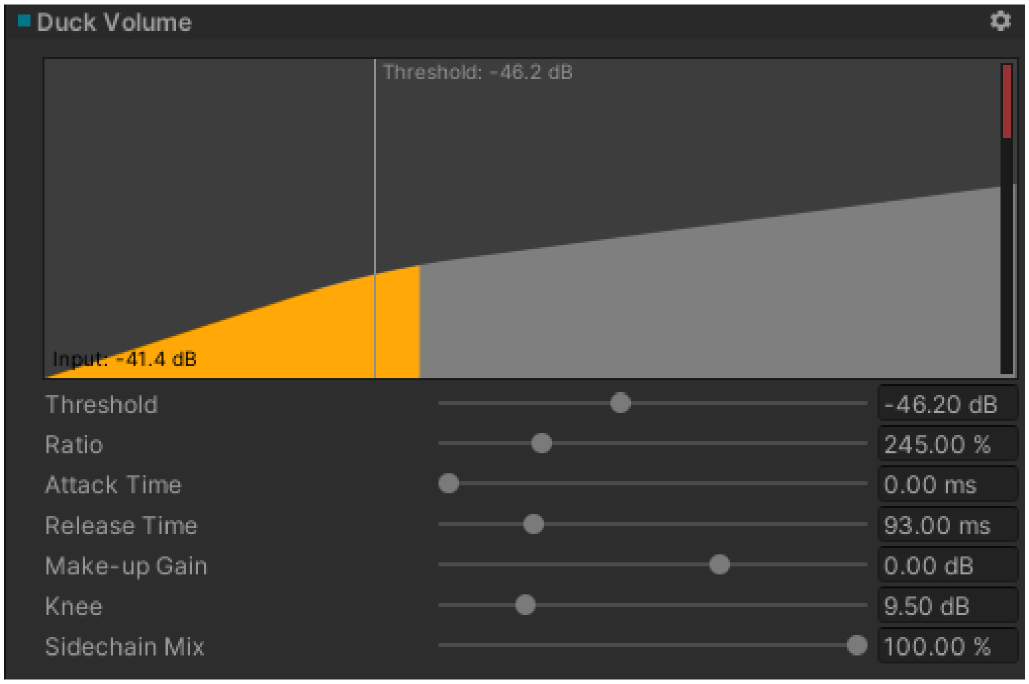
Send

Receive Singer\Duck Volume

Send level 0.00 dB

Fig 28 Example of the Duck Volume effect Receive parameter in Unity

With these linked, now you just have to adjust the parameters of the Duck effect.



Duck Volume

Threshold: -46.2 dB

Input: -41.4 dB

Threshold	-46.20 dB
Ratio	245.00 %
Attack Time	0.00 ms
Release Time	93.00 ms
Make-up Gain	0.00 dB
Knee	9.50 dB
Sidechain Mix	100.00 %

Fig 29 Duck Volume Effect parameters in Unity

The parameters that the duck volume effect offers are the ones that are usually used on professional plugins used for sidechain (see Fig 29) which allow to change the dynamics of the controlled source when there are variations on the linked source level (e.g., when the Drums volume rise the Singer volume down):

- Threshold: Volume at which the gain reduction starts to work with the entered signal.
- Ratio: It determines how much gain reduction is going to be applied once the threshold is triggered.
- Attack Time: It determines how quick the start of the gain reduction is, once the threshold is triggered.
- Release Time: It determines how quick the end of the gain reduction is, once the signal is finished.
- Make-up Gain: It allows to make back up again some of the volume it is reduced by the compression.
- Knee: It allows to make harder or smoother compressions. If the knee is 0db, a harder compression will be applied, meaning a flat line to the compression will be used. If the knee value is increased, a softer compression will be applied, meaning the line now will be a curve so a small compression will be applied even before the signal gets to the threshold.
- Sidechain Mix: It allows to regulate the amount of variation of the ducked signal volume in terms of the variations of the original signal.

Other interesting effects are the Lowpass and Highpass filters (see Fig 30) These filters are used to cut down frequencies above or below the selected frequency. In the case of the Highpass filter, only the frequencies above the one selected will continue to sound, and on the Lowpass filter only the ones below will still sound instead.

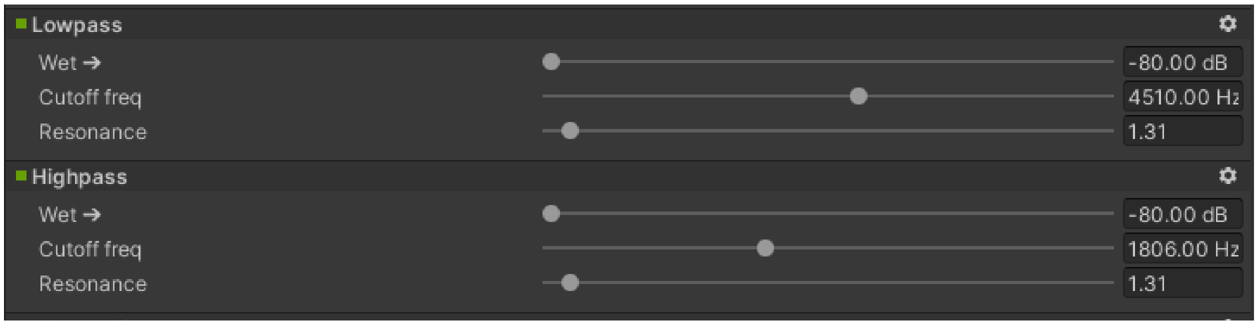
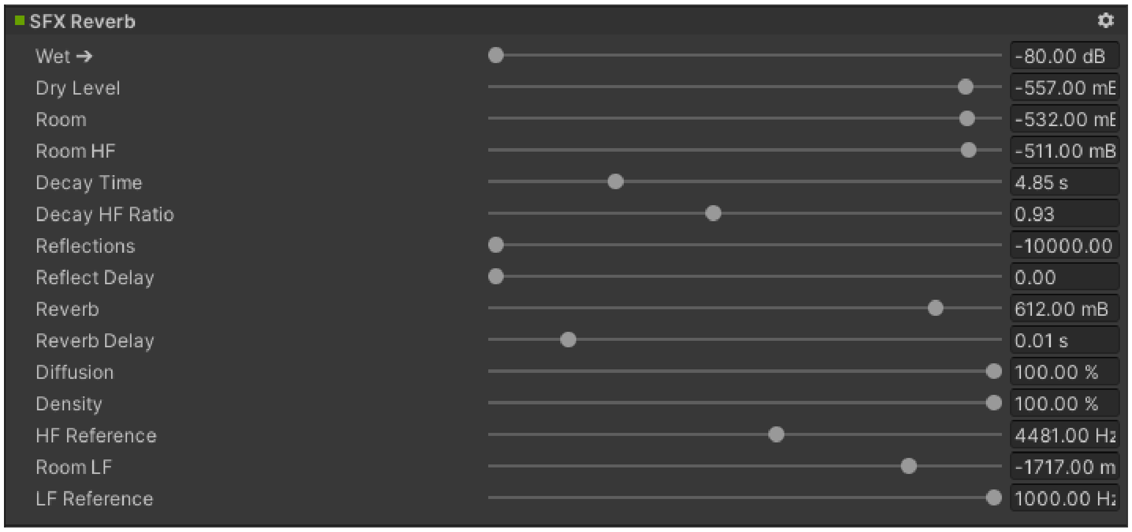

Lowpass
Wet → -80.00 dB
Cutoff freq 4510.00 Hz
Resonance 1.31
Highpass
Wet → -80.00 dB
Cutoff freq 1806.00 Hz
Resonance 1.31

Fig 30 Lowpass and Highpass filters in Unity

The resonance parameter will determine how much gain the cutoff frequency will have.

Another important effect is the Reverb (see Fig 31). Reverb takes the original signal and prolongates it to make it longer and simulate the sound produced in a certain space or room. Reverb effect simulates the reflections of the propagation acoustic rays along the simulated space. An example of a space with a lot of reverb would be a church.



SFX Reverb

Wet →	-80.00 dB
Dry Level	-557.00 mE
Room	-532.00 mE
Room HF	-511.00 mB
Decay Time	4.85 s
Decay HF Ratio	0.93
Reflections	-10000.00
Reflect Delay	0.00
Reverb	612.00 mB
Reverb Delay	0.01 s
Diffusion	100.00 %
Density	100.00 %
HF Reference	4481.00 H
Room LF	-1717.00 m
LF Reference	1000.00 H

Fig 31 Reverb effect in Unity

Some of the most important parameters of the reverb effect are:

- Dry Level: Volume of the original signal. If the Dry Level is at its lowest, only the reverb will be played.
- Room: Simulates the size of the room of the reverb.
- Decay Time: How long the reverb tail is, so how much time will the reverb lasts.
- Reverb: Volume level of the reverb.
- Reverb Delay: How much time will take for the reverb effect to start playing.
- Diffusion: It is used to modify the initial buildup of the echo density.
- Density: It controls the thickness of the reverb, that means that the more density, the greater number of reflections the reverb will have.

Pitch is a perceptual property of harmonic sounds like guitar, piano, or speech, that controls how “high” or “low” the sound note is played. It has a non-linear relation to the note the sound is played, depending on the logarithm of its frequency. The Pitch Shifter in Unity (see Fig 32) has different parameters that allow us to configure a pitch-change effect in real time without changing the duration of the sound, which can be implemented with the phase vocoder, which uses the Fast Fourier Transform as mathematical tool.

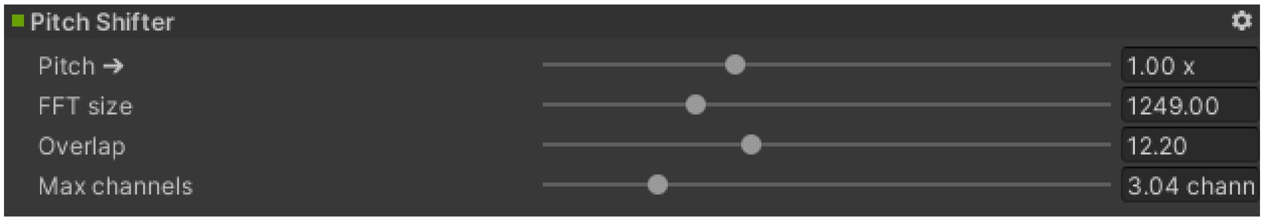

Pitch Shifter
Pitch → 1.00 x
FFT size 1249.00
Overlap 12.20
Max channels 3.04 chann

Fig 32 Pitch Shifter effect in Unity

The Pitch Shifter effect has the following control parameters:

- FFT size: It determines the size of the Fast Fourier Transform of the input signal buffer (see *2.2.4. Buffer*). It is the total number of samples of the FFT.
- Overlap: It determines how much overlap the windows used in the FFT have (see *2.2.4. Buffer*).
- Max Channels: Number of maximum channels of the FFT.

2.2. Audio Processing

In this section different theoretical aspects of real-time sound processing are going to be explained, due to the performed music application makes use of some music and audio effects that, its concepts, need to be previously explained to fully understand the use of them.

2.2.1. The Fourier Transform

The Fourier Transform is a mathematical transformation tool that allows the representation of time domain waveforms into the frequency domain.

The Fourier Transform is defined as:

$$X(f) = \frac{1}{2\pi} \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t}\, dt$$

Equation 1

Where *x(t)* is the time domain function, *t* is time and *f* frequency.

However, this function cannot work with discrete signals due to the usage of the integral. To solve this issue, we can use the DFT, Discrete Fourier Transform, which allows to decompose a finite signal frame of length N into a sum of N stationary exponentials:

$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}$$

Equation 2

Where N is the total number of samples of the DFT.

A window function w[n] is usually used as a signal delimiting the observation at a finite time of the discrete signal:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} w(n)x(n)e^{-j\frac{2\pi}{N}kn}$$

Equation 3

Even so, the transformation that Unity uses to get the spectrum data of the audio signal is the FFT, Fast Fourier Transform, which does the same as the DFT but so much faster.

2.2.2. Windowing

Windows are mathematic functions used in the signal processing field to avoid discontinuities on the analyzed blocks. When applying Fourier or analyzing spectral data, we use windows to locate the analyzed signal on time domain, allowing to observe the variations of the spectrum over the time. Also, taking the entire signal is a very expensive computational process, that's why we use windows, to select just a portion of that signal and allow real-time processing of signals. We could just do that with a rectangular window, which selects a part of the signal just as it originally is, however, when analyzing the signal with a rectangular window we will obtain a signal with some lobes around the signal frequency. This is called spectral leakage, which means that the signal energy from a frequency component has leaked to another frequency component which originally did not exist. To reduce spectral leakage we use smoother windows.

To do the process of windowing a signal we will multiply the signal by the window function, for example, in the DFT:

$$S(k) = DFT_N\{w(n) \cdot x(n)\}$$

Equation 4

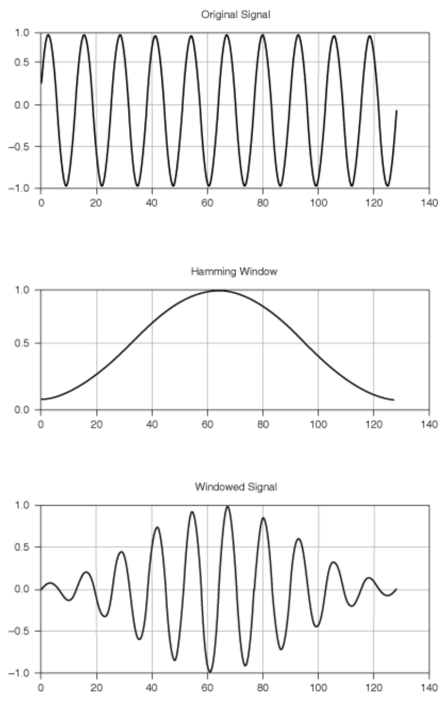


Fig 33 Multiplication of a sinusoidal signal with a Hamming window extracted from [9]

Which means the following property on the frequency domain:

$$w(n) \cdot x(n) \leftrightarrow \frac{1}{2\pi}\left[X(e^{jw}) * W(e^{jw})\right]$$

Equation 5

The most used windows are:

- Rectangular window:

$$w(n) = \left\{ \begin{smallmatrix} 1 & 0<n<N \\ 0 & other \end{smallmatrix} \right\}$$

Equation 6

- Hamming window:

$$w(n) = 0.54 + 0.46 \cdot \cos\left(\frac{2\pi n}{N}\right)$$

Equation 7

- Hanning window:

$$w(n) = 0.5 \cdot \left(1 + \cos\left(\frac{2\pi n}{N}\right)\right)$$

Equation 8

- Blackman window:

$$w(n) = 0.42 - 0.5 \cdot \cos\left(\frac{2\pi n}{N}\right) + 0.08 \cdot \cos\left(\frac{4\pi n}{N}\right)$$

Equation 9

- Triangular window:

$$w(n) = \begin{cases} \dfrac{2n}{N+1}, & 0 < n \leq \dfrac{N+1}{2} \\ 2 - \dfrac{2n}{N+1}, & \dfrac{N+1}{2} + 1 \leq n \leq N \end{cases}$$

Equation 10

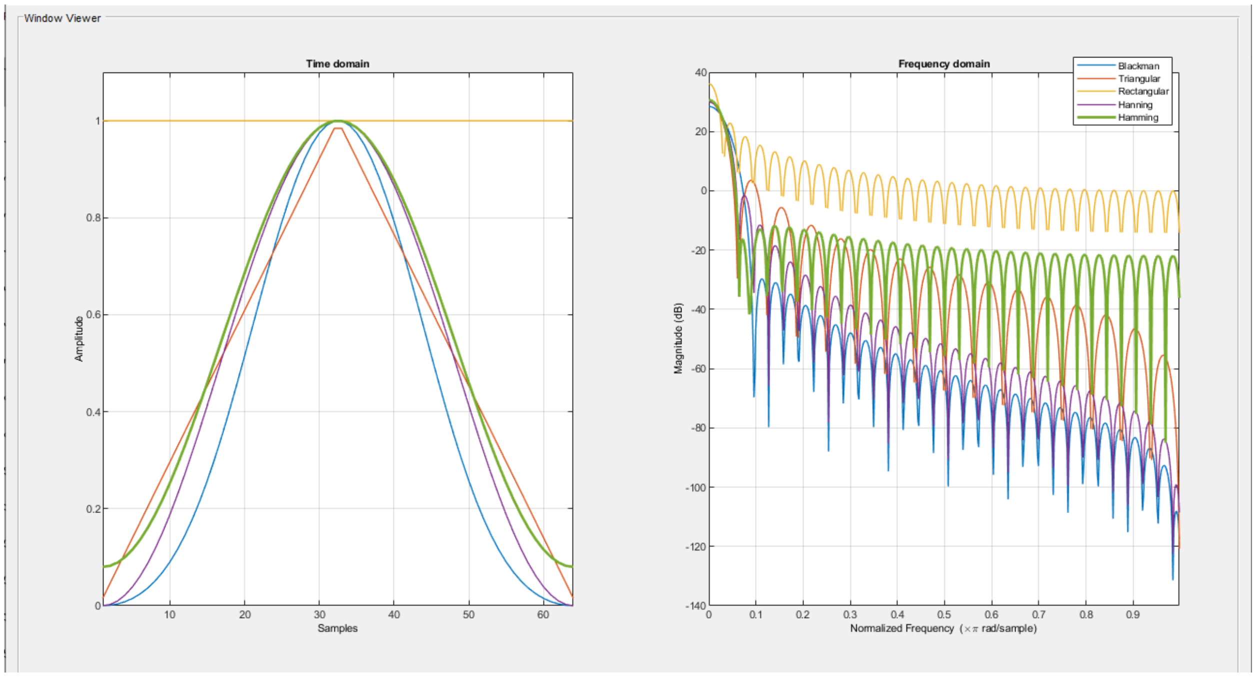


Fig 34 Window comparison in time/frequency domain (generated with windowDesigner App in Matlab)

2.2.3. Phase-Vocoder

The standard pitch-scaling technique based on the phase-vocoder consists in combining the pitch modification with the time modification. First, the phase-vocoder is used to multiply by a factor of β the timescale (being β the factor of pitch desired to modify). Then, this resulting signal is resampled, multiplying the original sampling period ΔT by β. This final signal has the same duration as the original but with a pitch modification of β. Even so, this method has a lot of computational costs, and it is impossible to calculate in real-time.

On the other hand, there is another method for changing the pitch with a phase vocoder in real time, a peak-based phase-vocoder that changes the pitch while the time stays the same, which is a possible method Unity can use due to Unity calculates the pitch change in real time while does not modify the time of the signal. This method consists, first of all, in detecting the peaks in the STFT. To declare the peaks, only the bigger than their neighbors are selected.

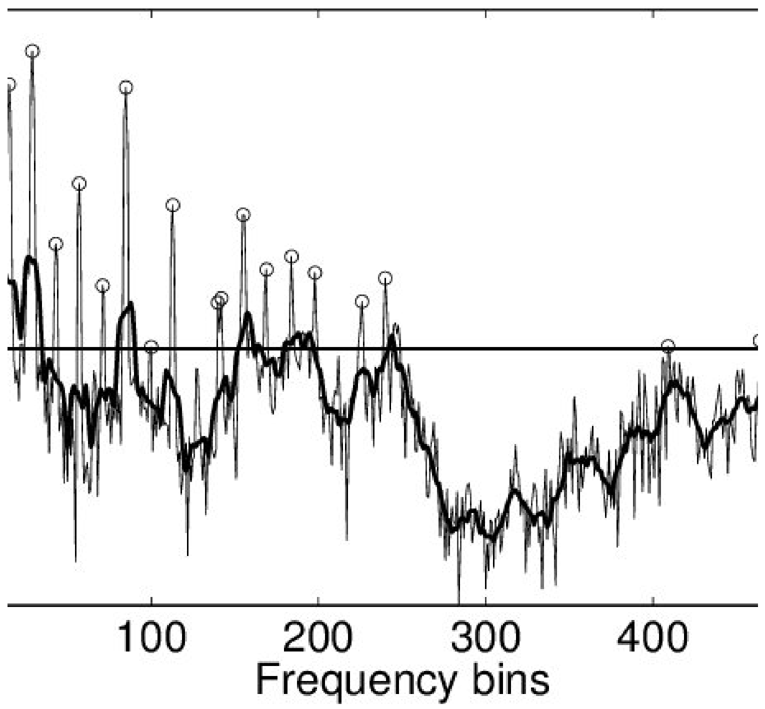


Fig 35 Peak detection in frequency domain extracted from [10]

When the peaks are picked, it is needed to calculate and shift each region around them in order to get the frequency modification we are looking for.

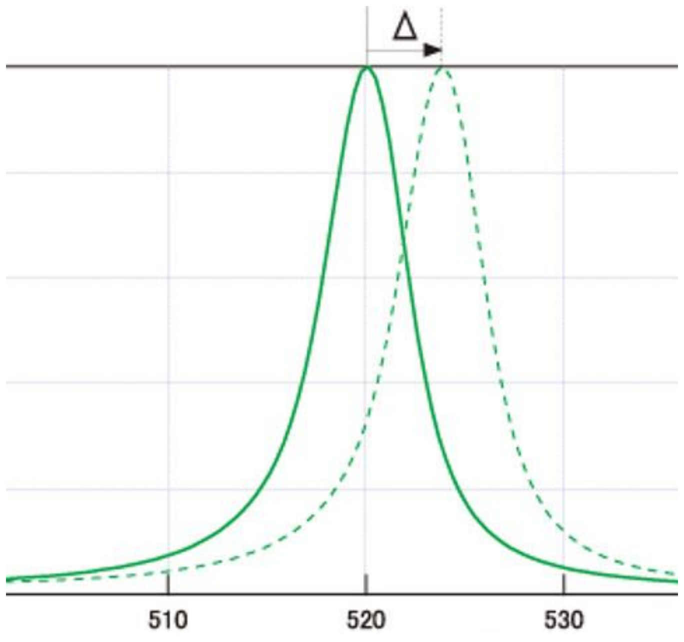


Fig 36 Region shifting extracted from [11]

Once the peaks are shifted, there is only one step left, phase adjustment. The phases of the peaks may be modified to be adjusted to their frequency modification.

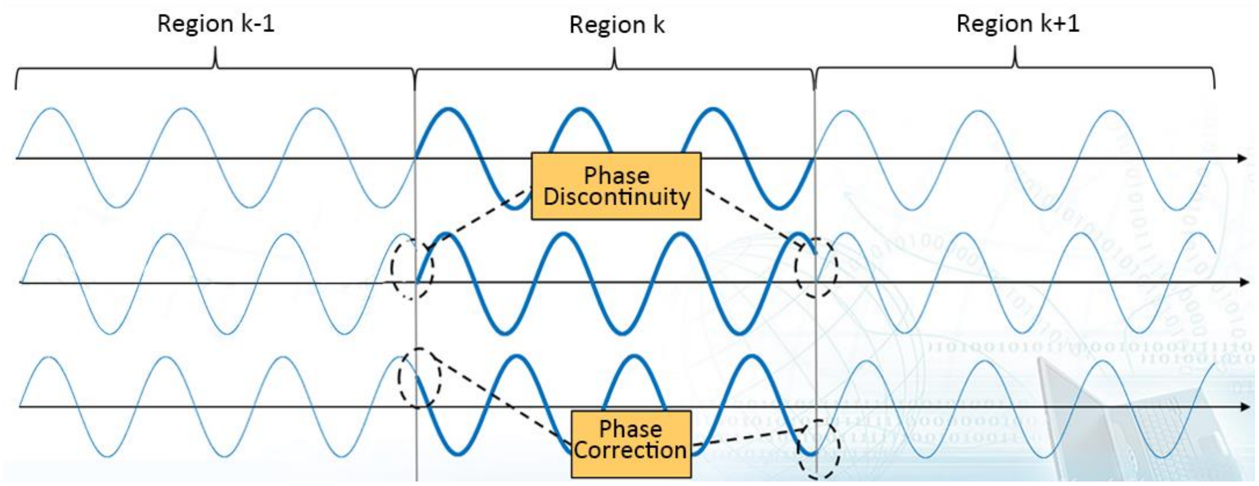


Fig 37 Phase adjustment extracted from [12]

2.2.4. Buffer

In signal processing, the buffer is responsible of redistributing the input samples from a signal to a new larger or smaller size. Processing a larger signal with buffers will set up a slower frame rate to the new signal, as well as making it smaller will set up a faster frame rate. To buffer a signal, we use overlapping, which means that in order to make the signal smaller or smaller, we will use the repetition of a specific number of samples.

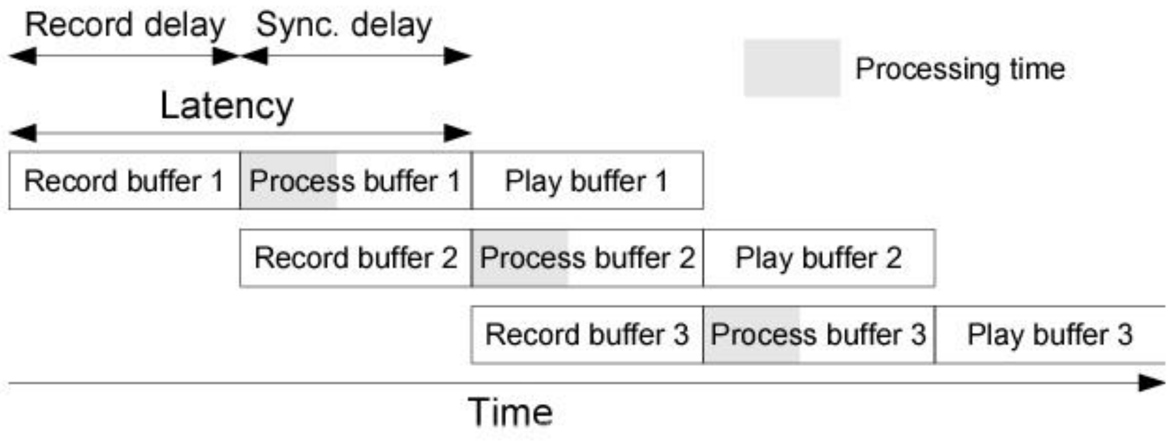


Fig 38 Buffer Process example extracted from [13]

2.2.5. Mels

The mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another. As the frequency spectrum works with a logarithmic scale, we cannot find equal distances to pitch human perception, that is why we change to the mel scale. This scale method can be interesting to use for grouping frequency bands that contain information that perceptually can be equally important.

The function to convert from *f* hertz into *m* mels is:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

Equation 11

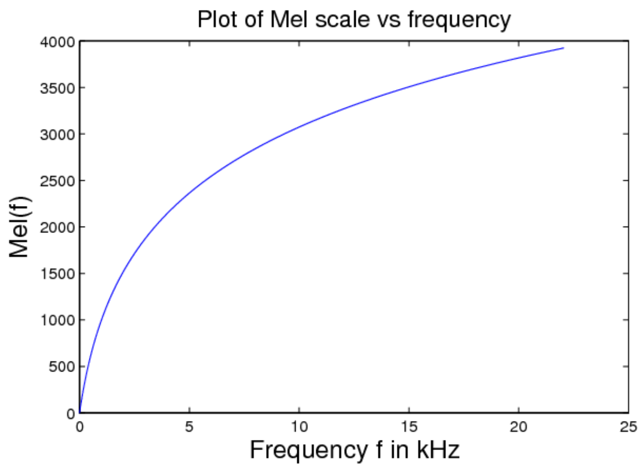


Fig 39 Mel and Frequency relation extracted from [14]

2.3. Game Design, User Interface & User Experience

In this section it is going to be explained some important aspects that are necessary to have in mind before and when creating a game.

2.3.1. Game Design

Game design is the essence of game development. Game design is the creation of stories, characters, levels, mechanics, rules and objectives a game has. A game designer needs to have a big spectrum of ideas and knowledge about videogames, not only about storytelling, art and other creative ideas, but also about programming and more technical aspects.

One important document that helps game designers keep track of what they have in mind, forces them to think and make the game easier to understand and follow for everyone in the team is the GDD (Game Design Document).

The GDD is the document that collects all the information about a videogame and how it has been created and developed. In this document it will appear important aspects as the story, mechanics, levels, videogame progress, interfaces, characters and objects, music and sounds and production details.

2.3.2. User Interface & User Experience

UX stands for User Experience. It refers to all aspects of how end-users interact with a product. In the case of game development, UX designers focus on how to design a game that will give the best possible experience to the player. The UX designer needs to make sure that the target of the game feels as desired, that their way to the objective is fluent and that their goal can be easily achieved. Or not that easily, some videogames are characterized by their difficult levels and the hard work that is required to get to the end of the game. All these aspects need to be analyzed by the UX designer and he is the one who will make important decisions and changes to the game to adapt it to a better experience.

The User Interface of a website, application or video game is created with the purpose of getting the most visually and technically attractive design. That comes from different processes:

- Know and understand what our target is, emphasize and know what their necessities are.
- After getting all our ideas together and knowing what our objectives are, it is time to start with the concepts and designs.
- The first designs are the wireframes. Wireframes are schematic designs that are created just with the structure and concepts of where every component will be placed. Wireframes are useful for communication between programmers and designers to start building the structure of the software.

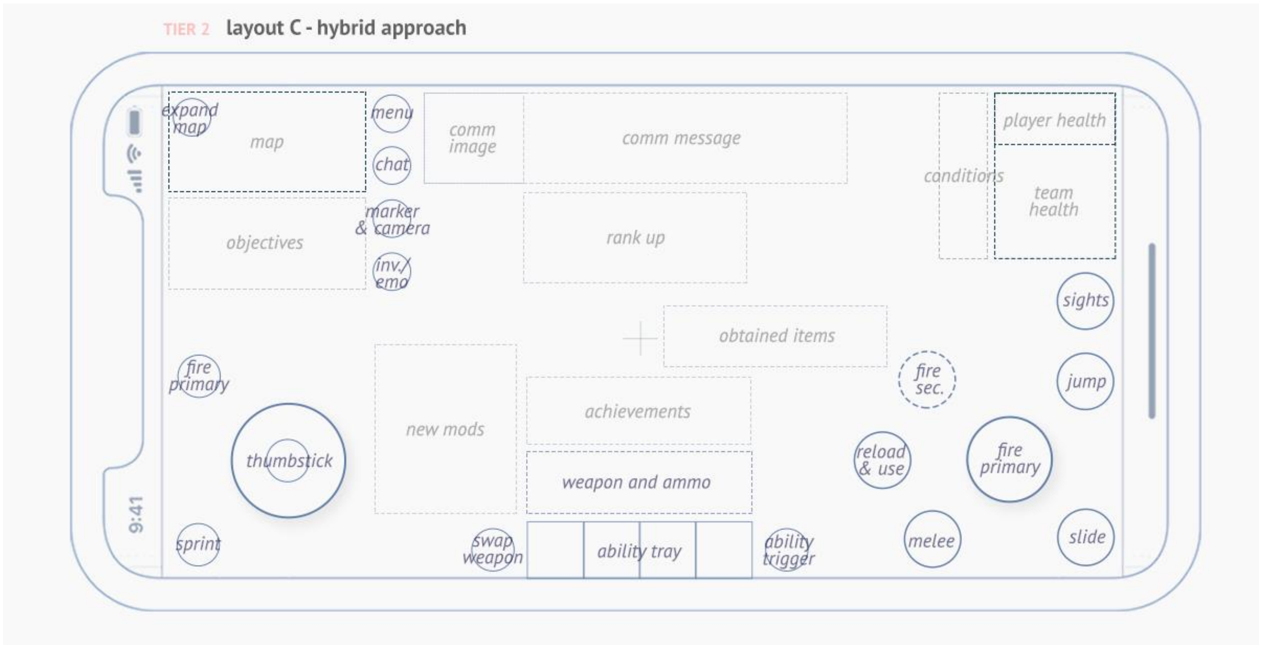


Fig 40 Example of User Interface on smartphone extracted from [15]

- Once the wireframes are finished, the designers and artists must start creating all the colors, textures and images to fulfill the structure of the wireframe with a finished look.

- Finally, the development team starts programming the interface linking and making every button and element work. With this step completed, it is time to put this part together with the back end.

Some other interesting points user interface and user experience designers should consider when creating a game are:

- Before starting to create the concepts, the designer should do some research and look for similar ideas and concepts of the competitors. Knowing what your target likes the most is crucial to create an interface the user will enjoy.
- Not overloading the user with useless information, having unnecessary images or buttons on the screen can make the user lose interest and even uninstall the app. Also, designing games for smartphones requires optimizing more the space of every element on the screen due to the small size of it.
- Good feedback is necessary for the player to know if he is doing everything as it should be. When a button is clicked, or every other action is realized, the user needs visual or auditive feedback to know the app is working correctly.
- Correct buttons sizes are essential to have a good interface in the game. If the button is too small, the player will struggle trying to press it, and if the button is too big, it will take up too much space on the screen and it will not have enough space for other things or even will feel weird for the player. Also, different predefined sizes of typography need to be specified for different types of texts, like titles or paragraphs, if not, the player can feel something weird too.
- Testing your designs is as important as testing the logic of the game. The design does not look the same on the computer as it does on smartphones, that is why it is important to test them and see if they are good and fit with the expectations. It is also important that someone who has not been working on the design tests it because he will always offer a different point of view on different aspects.
- Different classes can be used to explain the interface of a videogame. These classes depend on two simple questions:
  - Is it in the game story?
  - Is it in the game space?

  When these two questions are answered, four classes appear: Non-Diegetic, Diegetic, Spatial or Meta (see Fig 41).

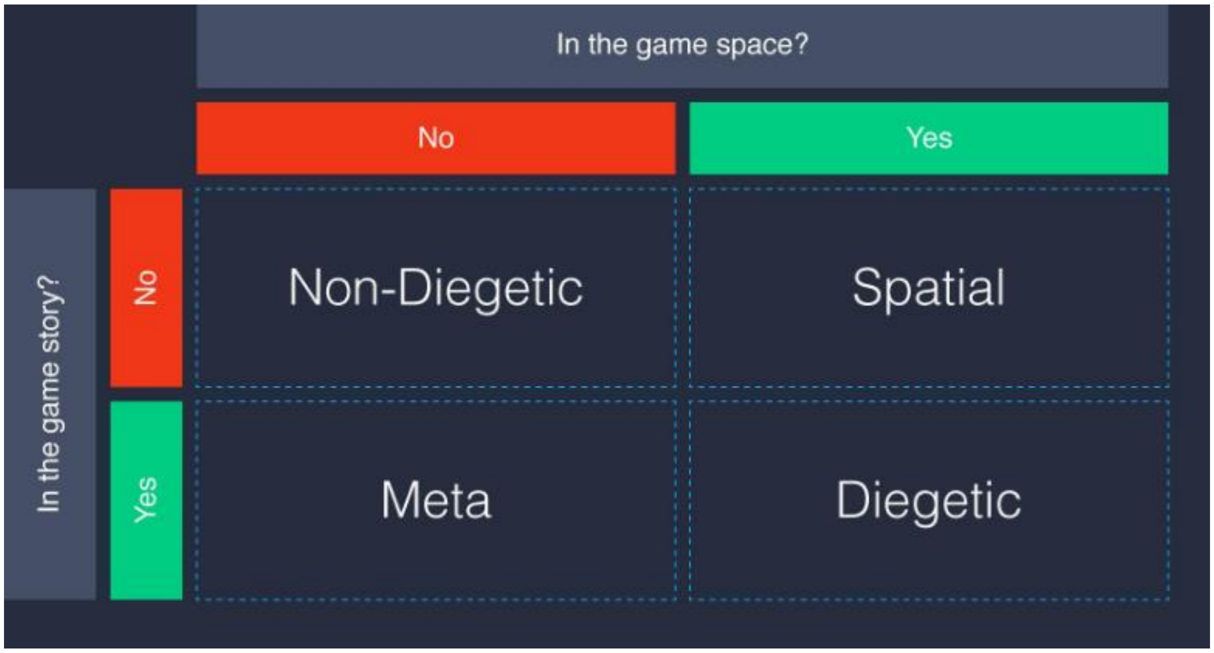


Fig 41 Types of User Interface in a game extracted from [5]

- Non-Diegetic

When the elements are not in the game space and neither in the game story, we are talking about a Non-Diegetic UI component. These elements are usually used for letting the player to know different parameters that will help him keep track of the game. Some examples could be points, life, ammo, time, etc. (see Fig 42).

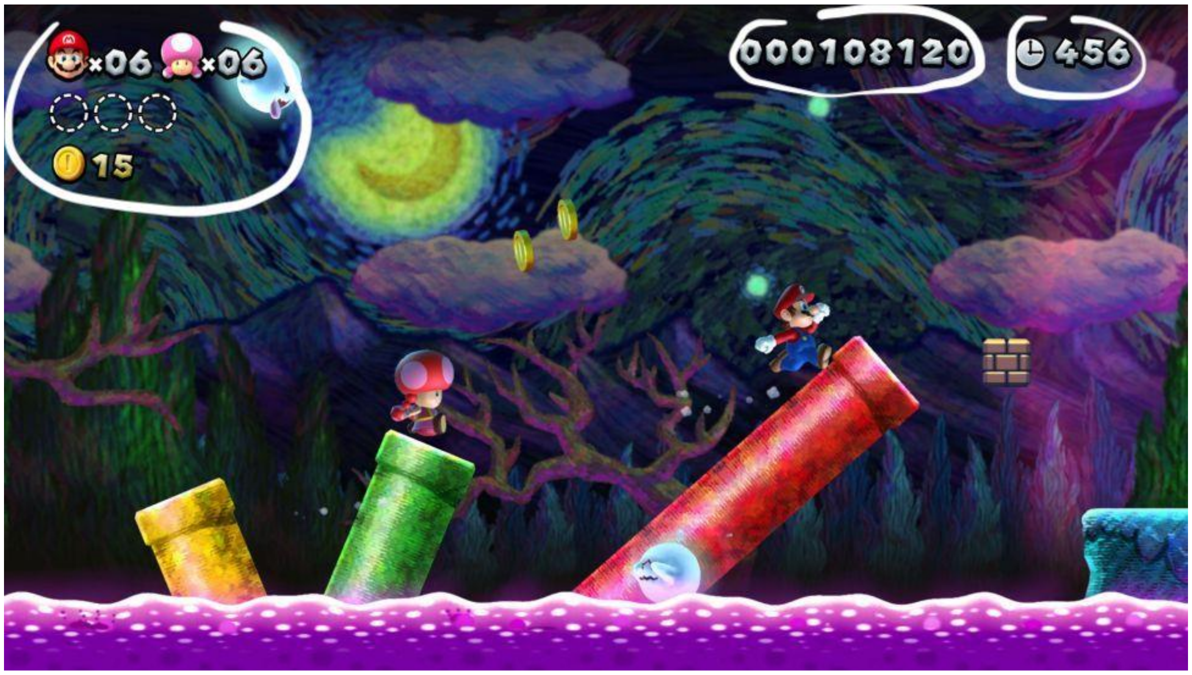


Fig 42 Example of Non-Diegetic Interface in Super Marios Bros. U extracted from [6]

- Diegetic

When the elements are in the game space and in the game story, a Diegetic UI component is referred. These elements can help the player to understand specific parameters like the health of a character or a map position. These are implemented inside the game instead of the HUD (see Fig 43), for example in the main character or other objects, and it just makes sense in the context and story of the game.

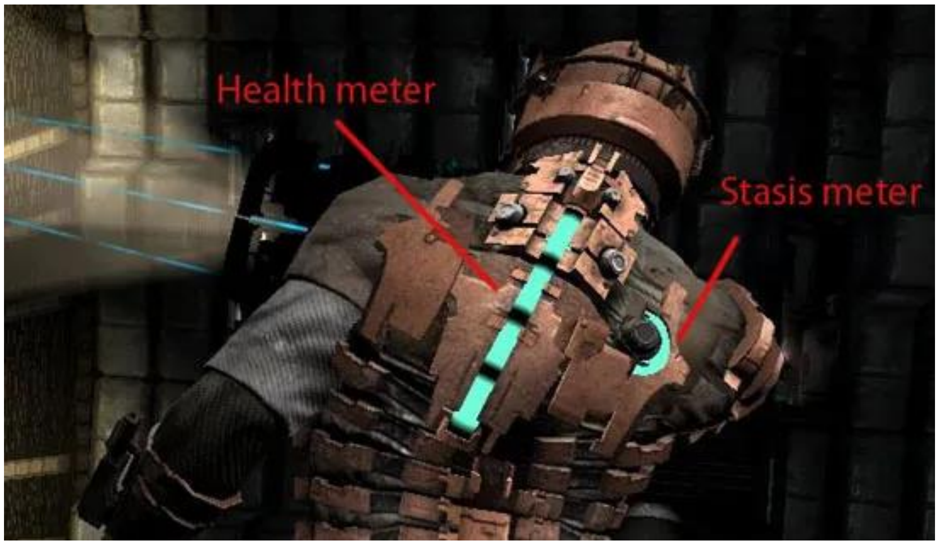


Fig 43 Example of Diegetic Interface in Dead Space extracted from [7]

- Spatial

Spatial UI components are the ones that appear inside the game space but only make sense for the player, not for the virtual characters or the story. These elements are usually to help players with buttons, directions or marks in general (see Fig 44).

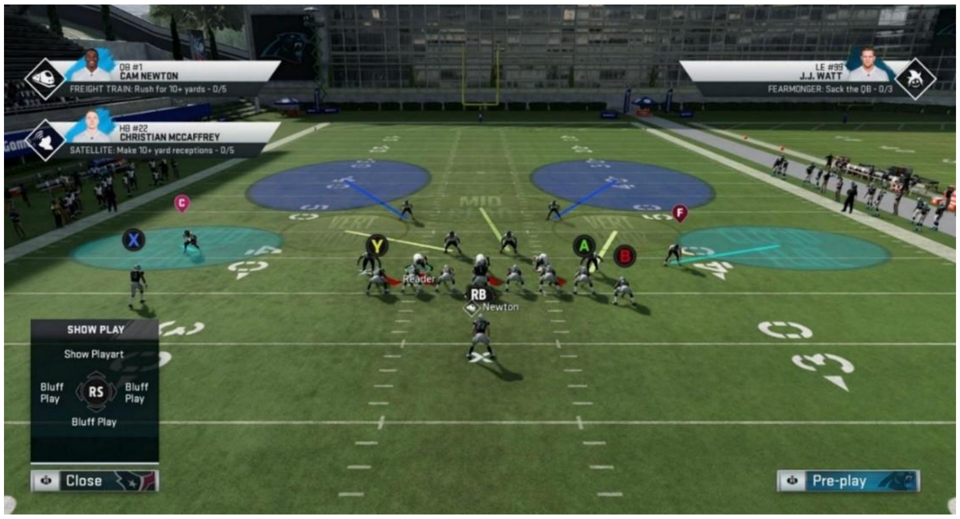


Fig 44 Example of Spatial Interface in The American football franchise Madden extracted from [5]

- Meta

Meta UI components form part of the story of the game but do not appear inside the game space. Traditionally, these components have been used to show damage to a player's character, but they can also be used to show weather changes, shaking cameras or speed effects for example (see Fig 45).

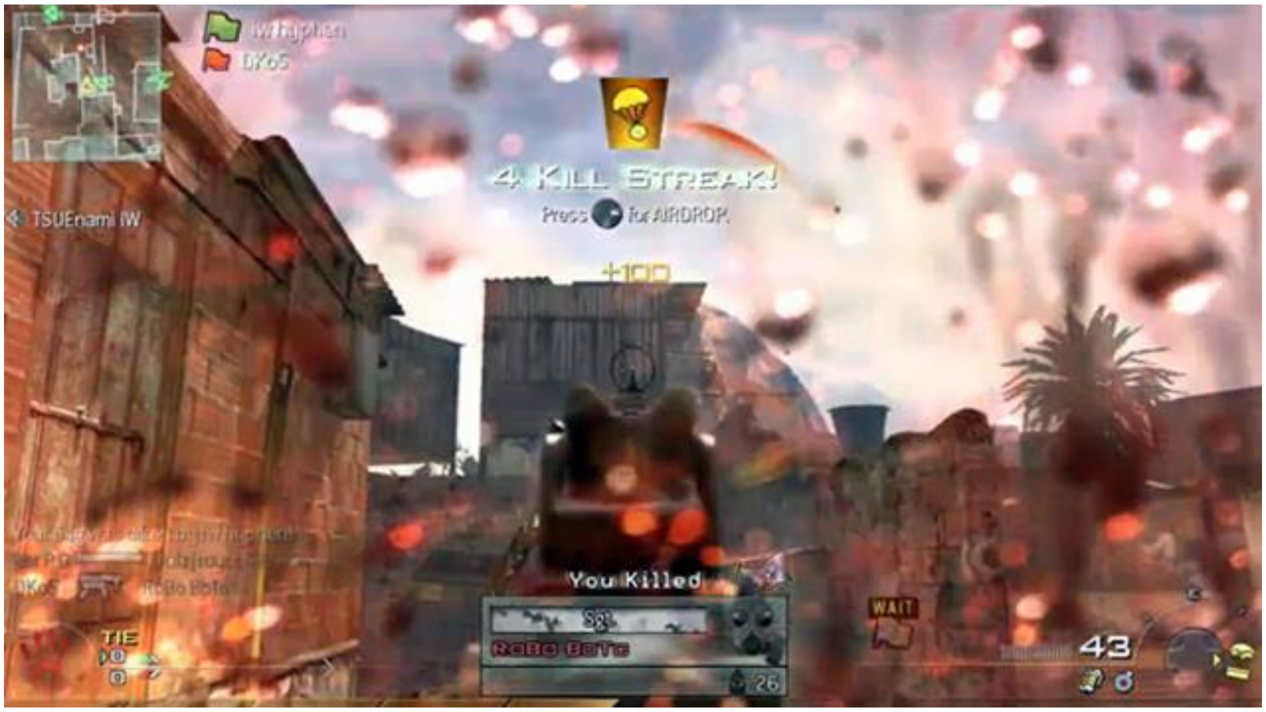


Fig 45 Example of Meta Interface in Call of Duty extracted from [8]

2.3.3. Playtesting & QA

In this section we are going to describe what playtesting & QA means and their differences.

Playtesting and QA are very similar concepts but play different key roles during game development.

QA (Quality assurance) testers are the ones in charge of going through the whole game to make sure the software works correctly. It is a more technical role and is more focused on finding bugs, testing mechanics, gameplay and game balance to make sure nothing is too overpowered or has a lack of utility for example.

Playtesting, instead, focuses on having people to play your game and seeing how they react to it. In this part the game developer needs to make sure that every aspect and mechanic of the game is working as they wished, considering the playtester has never seen the game.

For the part of playtesting, it is important to consider different aspects:

- The playtester has to play the game without having played it or seen it before. The first reaction of the player to the game is needed in this part.
- A test plan is needed, different objectives and tasks must be done. For every task the game developer must collect different data like the success ratio or the time.
- The playtester is not able to ask for help or ask questions referring to the gameplay. If the playtester does not know how to continue, for example, the game developer has to write it down to improve the game in the future.
- The game developer must ask questions while the playtester is playing the game and after he is finished. Some possible In-Game questions are:
  - Why did you make that?
  - Did that make you feel confused?
  - What did you think that would do?
- Some possible Postgame questions are:
  - Did you get stuck in any part?
  - Has something frustrated you?
  - Tell some aspects that felt satisfactory.
  - How did you feel about the controls?
  - Was there some mechanic that was difficult to understand?
  - What do you think that can be improved?
  - What elements of the game attracted your attention?
  - Was something missing in the game?

2.4. Fruity Loops Studio

In this section different basic concepts of the software FL Studio are going to be explained to be able to understand how the process of music production for the project has been done with this tool.

2.4.1. What's Fruity Loops Studio?

Fruity Loops Studio (see Fig 46) is a digital audio workstation (DAW) with different versions and editions. FL Studio is a software that, with previous payment, everyone, from an apprentice to an expert, can enjoy and create with. All the process of creating a song can be done with FL Studio, creating ideas, recording instruments, composing melodies and structures, applying effects, mixing, mastering, etc. These types of tools, apart from music, are also useful for creating sound effects for movies or videogames.

Fig 46 Fruity Loops Studio Logotype extracted from [16]

2.4.2. Fl Studio Interface

The FL Studio user Interface is depicted in Fig 47.

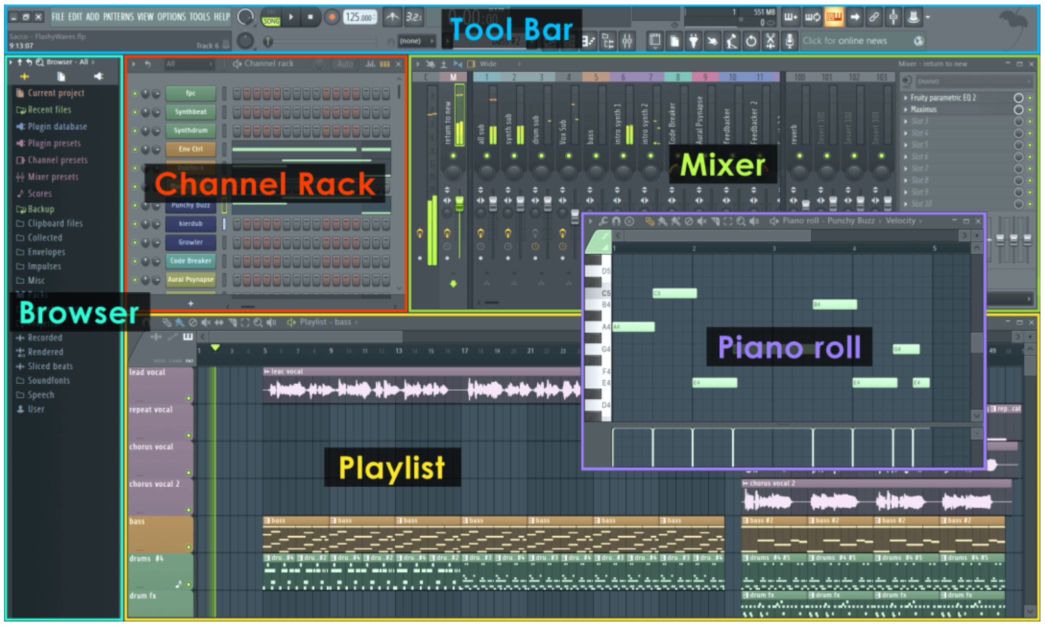


Fig 47 Fruity Loops Studio Interface extracted from [18]

The main elements found in the FL Studio user interface are the following:

- Playlist: Is where the final song is played. All the elements that form the song are put there in the order the user wants to create and distribute the song however he wants.
- Channel Rack: This element allows the user to create different patterns that then he will be able to put on the Playlist section. In this section we can see all the sounds the project has. One same pattern can have different melodies and different rhythms on it with the sounds the user wants.
- Piano roll: The piano roll shows a complete piano with all the notes and a lot of scales, allowing the user to create melodies for different patterns on the Channel Rack.
- Mixer: The mixer is where all the sounds are usually sent to make it easier for the user to treat separately each sound. In this section, all channels can have up to 10 plugins. These plugins will usually apply effects such as compression, distortion, filters, delay, reverbs, etc. The final result of the song plays what is played on the Master channel. Each channel is sent to the Master by default but can also be sent to other channels.
- Browser: This section shows a list of files and folders. These files are usually used to easily navigate and listen to different packs of sounds or plugin presets the user has on his computer. When the user wants to add one sound to the project, an easy drag and drop system allows to import it to the project.
- Tool Bar: In the tool bar all kinds of options can be found. Not only options referring to the project, like basic elements such as the play button, recording button or the bpm button, but also different settings for the software.

3. The Game

In this section an explanation of every part of the game is going to be presented, talking about the main mechanics, how the game works and which decisions were taken for every idea.

This game is about creating music, the user is given different sampled melodies for different instruments, and he is the responsible for using them with different combinations to create his own music.

3.1. Free Mode

On this game mode, the player is free to play the music however he wants. With 5 different robots, each one with a different instrument (electronic drums, electronic bass, electronic guitar, electronic piano and robotic singer), he can choose between 5 melodies to play with each robot.

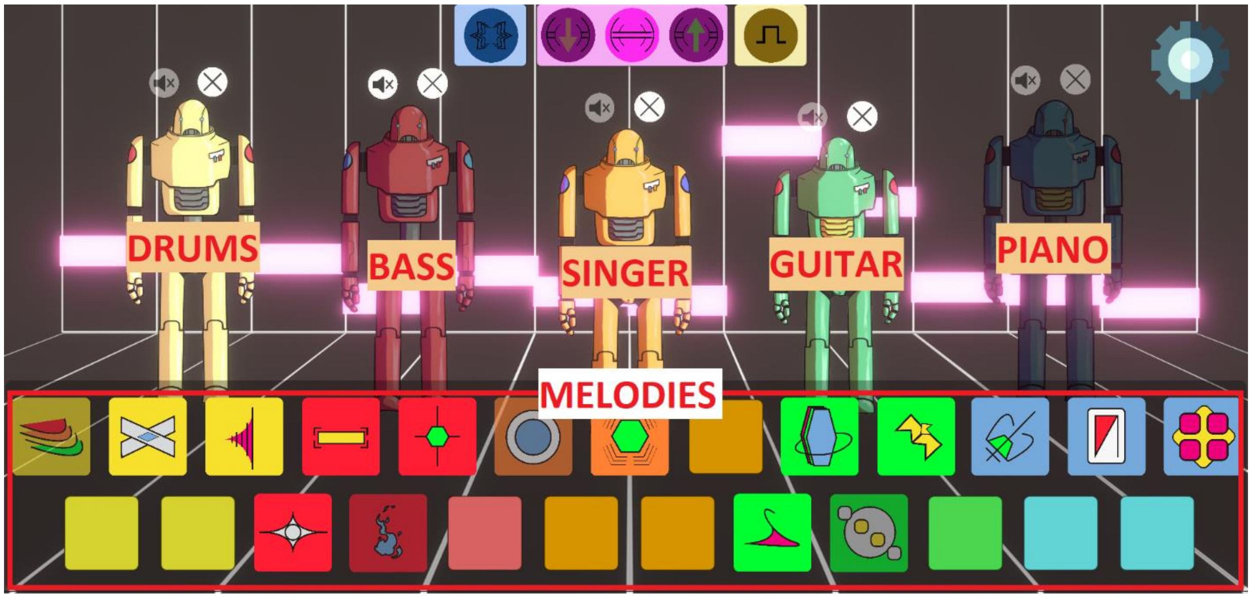


Fig 48 Beat Repeat Free mode Interface

As shown, the melody that is currently playing has its opacity downed so the player knows which one is currently playing. (Melody icons with no drawing need the melody and the drawing to be finished, as well as the design of the robots)

Then there are also the cross buttons and the silence buttons. Both, one for each robot, meaning the player can silence and quit their melodies separately. Notice how the cross button is highlighted when the robot has a melody playing, meaning that the user is able to press the button, as well as the silence button is highlighted when the melody is silenced.

Also, another important functionality on this mode is that the player has the possibility to choose between 4 different electronic sounds for each of the instruments (the singer not included), giving the player the opportunity to play and create songs with thousands of different combinations with these sounds and melodies. To do so, the player has to press one of the robots to see which other sounds the robot has:

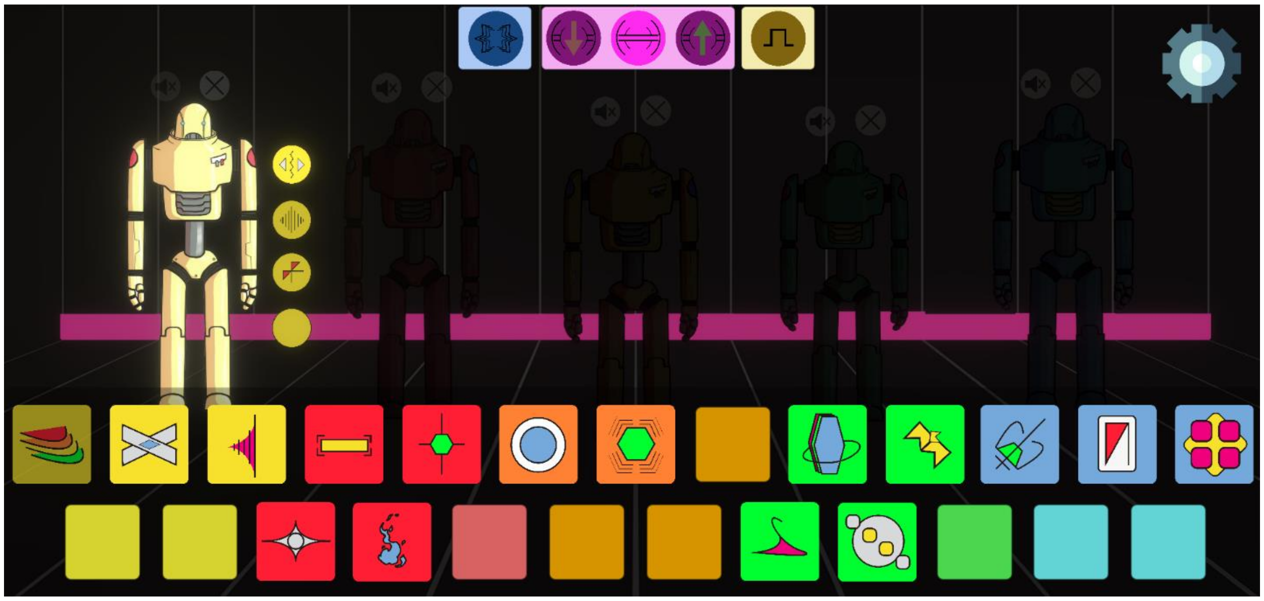


Fig 49 Beat Repeat example of sound selection for the drums instrument

As shown in Fig 49 the selected sound is highlighted.

In total, each robot has:

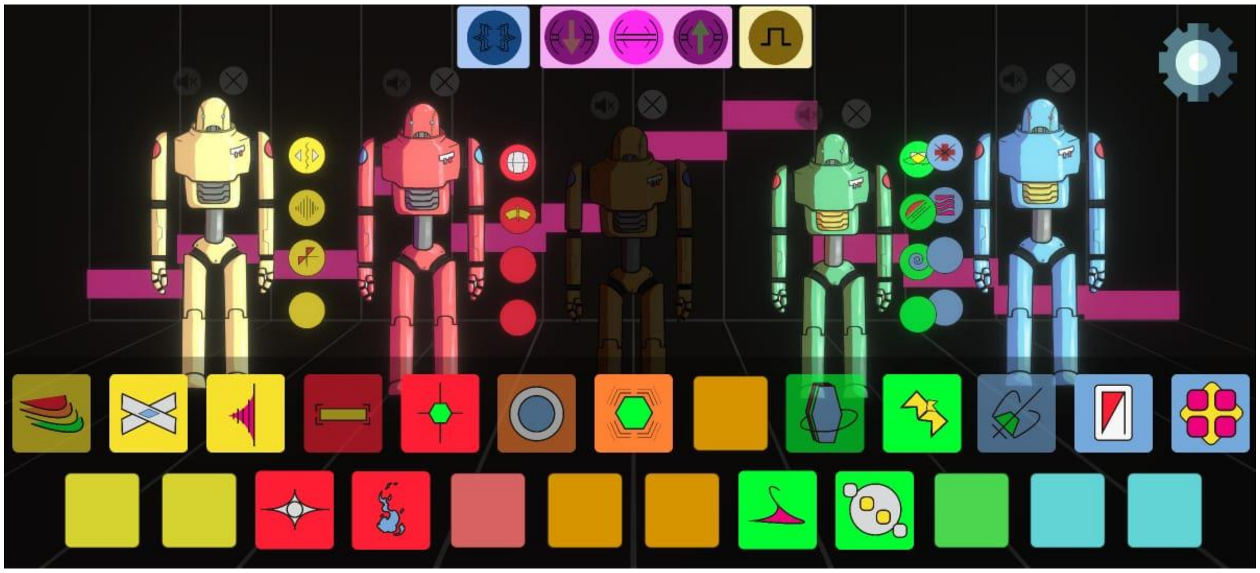


Fig 50 View of all the sound icons each instrument has in Beat Repeat

The singer does not have more sounds for the moment because I wanted to give him more leadership and because he does not have something like an instrument that can be changed, but adding more sounds to him is a possibility.

It is important to mention the 3 audio-effect buttons the player can use to:

1.- Apply a default reverb to the overall mix. This reverb is a long reverb with the Dry knob very low to give the effect of being in a kind of big empty room. This button is the blue one on the left.

2.- Apply a default bandpass filter to the overall mix. This filter is usually used to give the feeling of being on a phone call. This button is the yellow one on the right.

3.- Apply a pitch change to the overall mix. In this case the user is able to pitch up the song on a predetermined pitch, or also pitch it down to a predetermined pitch too. The center button is clicked by default because is where the pitch is not being modified. When one pitch button is clicked, the other two are deselected. These buttons are the pink ones and are in the middle of the effects buttons.

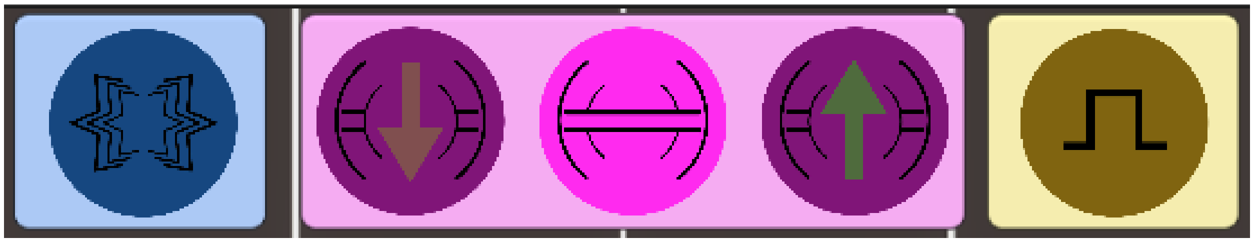


Fig 51 Sound Effects panel in Beat Repeat

These 3 different buttons are on the game to give the player a simple and funny way to create different parts of a song and give tools to easily fulfill the desire of creativity.

3.2. Story Mode

On the Story Mode the game offers different levels the player must complete one by one if he wants to go through all of them. The idea of this mode is telling a small story in each of the levels to let the player know better about the robot's life.

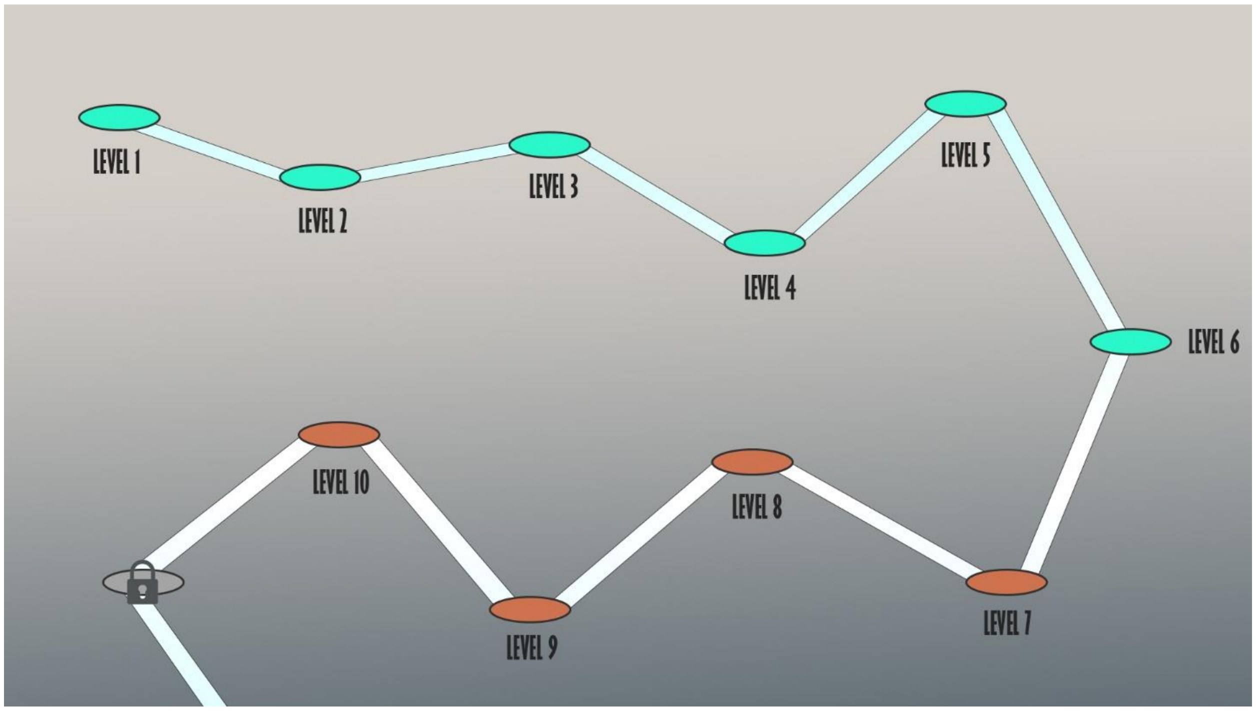


Fig 52 Concept idea for the Levels screen of the Story Mod in Beat Repeat

The green circles would be the levels the player has already passed and the red levels the ones he has not. In the future version of the game, every level will have a minimum score the player will need to have in order to complete the level and pass to the next one.

In this Mode, every level will have a structure of a song, and the player will have to copy and play it by following the instructions given.

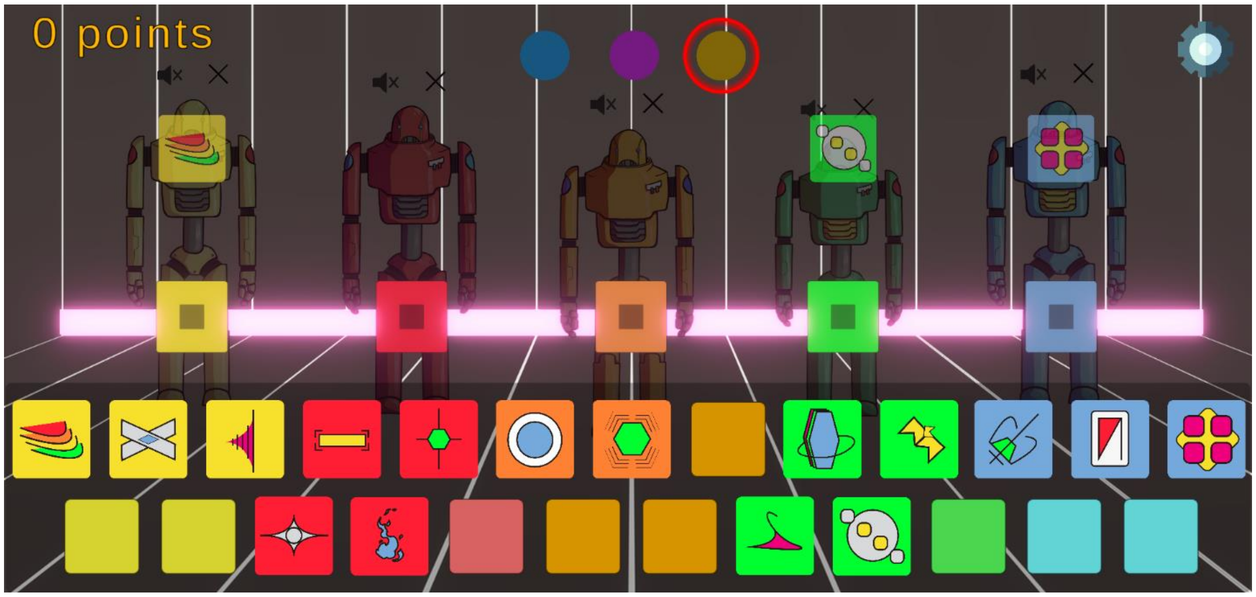
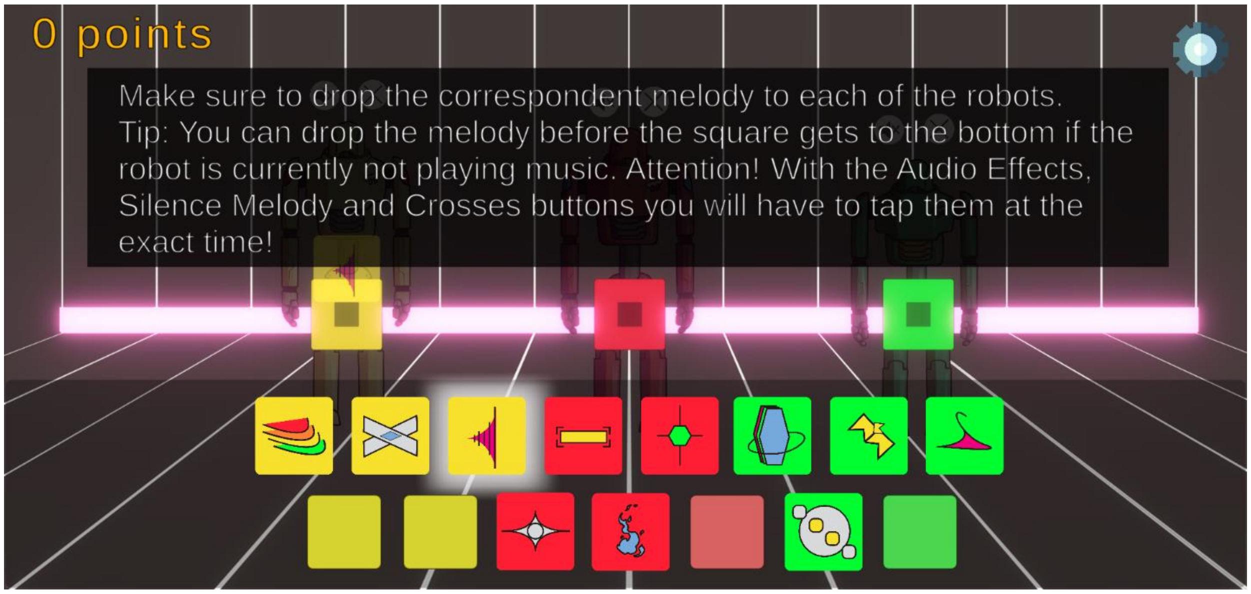


Fig 53 Example of icons falling down on the levels of the Story Mode in Beat Repeat

Different icons fall from above the screen to show which melody has to be dropped to the different robots. When the icon goes through the different bottom squares of each robot, the melody will have to be dropped. It happens the same with the buttons that need to be clicked, like the Quit melody, Silence Melody or Audio Effects buttons, but instead of icons falling down, a circle appears when that button needs to be pressed.

When the player starts the game a help text appears on the screen:



Make sure to drop the correspondent melody to each of the robots. Tip: You can drop the melody before the square gets to the bottom if the robot is currently not playing music. Attention! With the Audio Effects, Silence Melody and Crosses buttons you will have to tap them at the exact time!

Fig 54 View of the tutorial for the first level in the Story Mode in Beat Repeat

At this moment, the game stops, giving the player time to read the tip shown. The tip will disappear when the player drops the correspondent melodies.

Different messages appear on the screen depending on how good the player plays the music required:

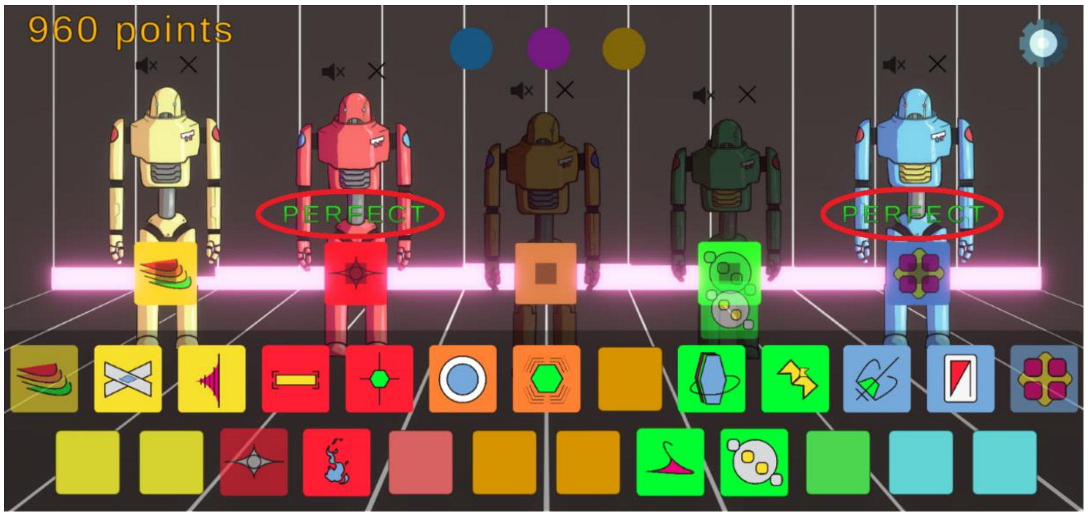


Fig 55 Example of a "Perfect" messages in the Story Mode of Beat Repeat

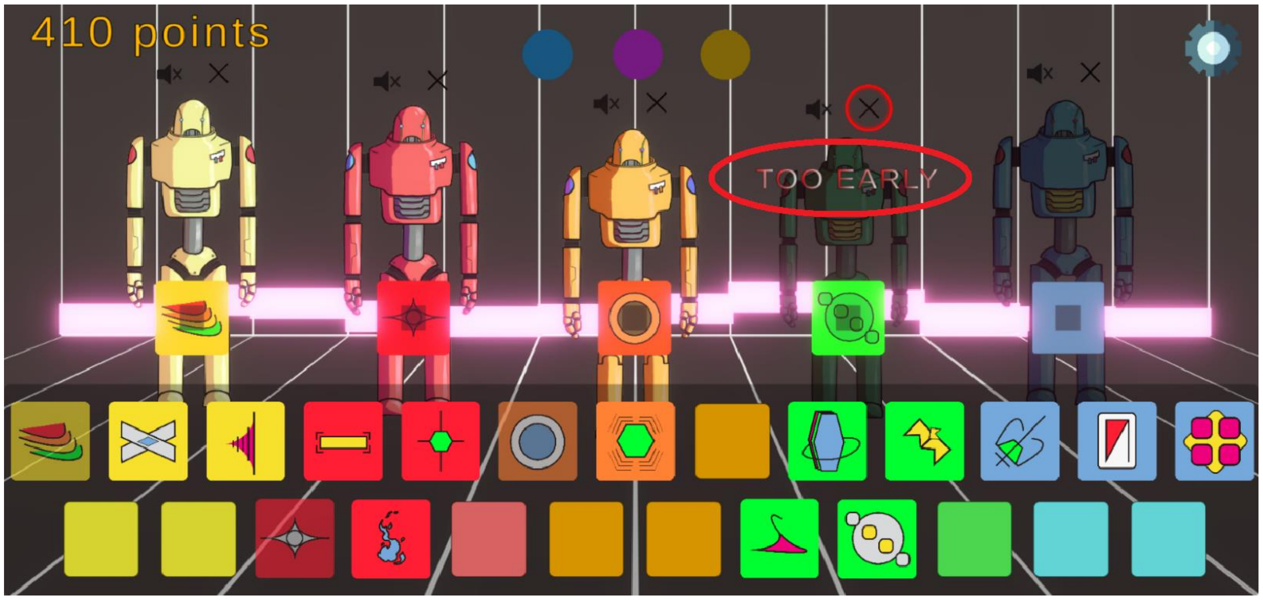


Fig 56 Example of a "Too Early" messages in the Story Mode of Beat Repeat

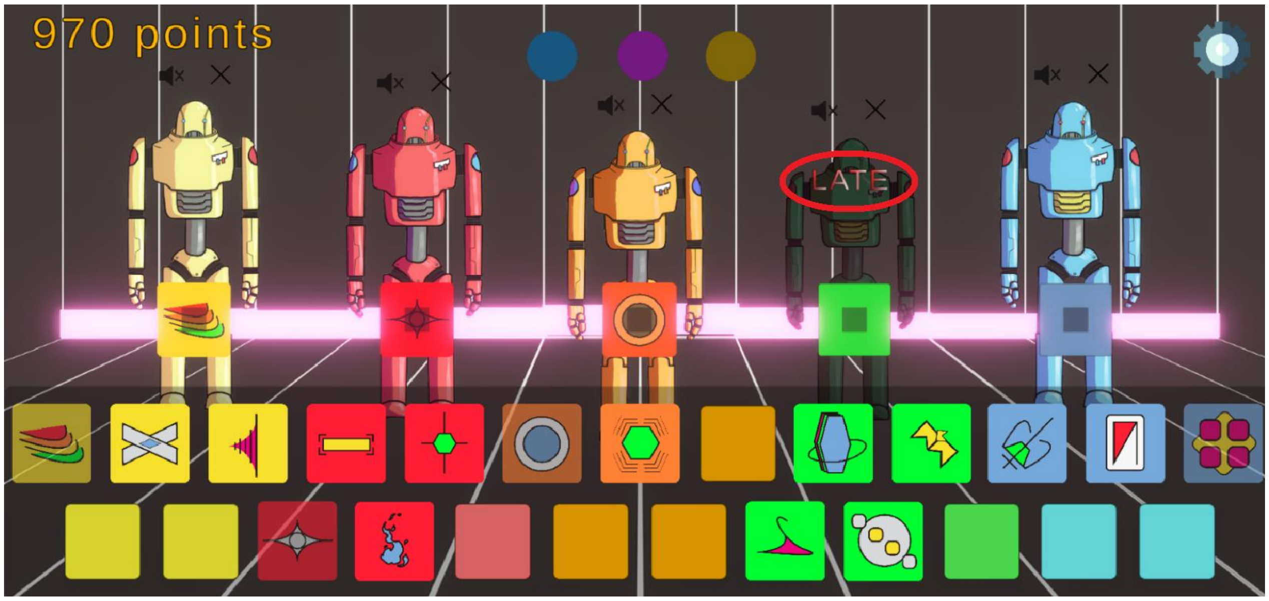


Fig 57 Example of a "Late" messages in the Story Mode of Beat Repeat

Also, more or less points are added to the score depending on how good the player goes through the level. If the player gets something wrong, some points are subtracted. The number of points is shown at the top left of the screen.

Some features, apart from the help messages, that are in this mode to help the player are, when a robot needs to be playing a melody, its icon will stay in the reference square, helping the user if he gets lost when playing:

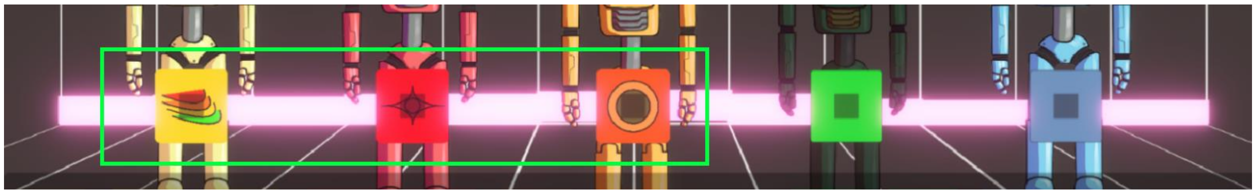


Fig 58 Example of the icons in the reference square in the Story Mode of Beat Repeat

Also, when a melody is needed to play, not only will the icon of the melody fall down, but the melody on the melody panel will be highlighted to make it easier for the player to see which one is required.

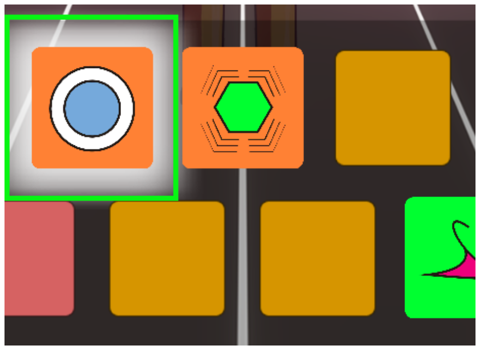


Fig 59 Example of an icon highlighted in the Story Mode of Beat Repeat

In this version of the game, there are three levels:

- Level 1: It is a short level to let the player know the basic mechanics. Here, a short tutorial is introduced where the player learns when he has to drop the melodies and when to press the silence and cross buttons (see Fig 60). In this level, there are only three of the five robots, and non-audio effects are used to make it easier for the player to learn the functionalities.

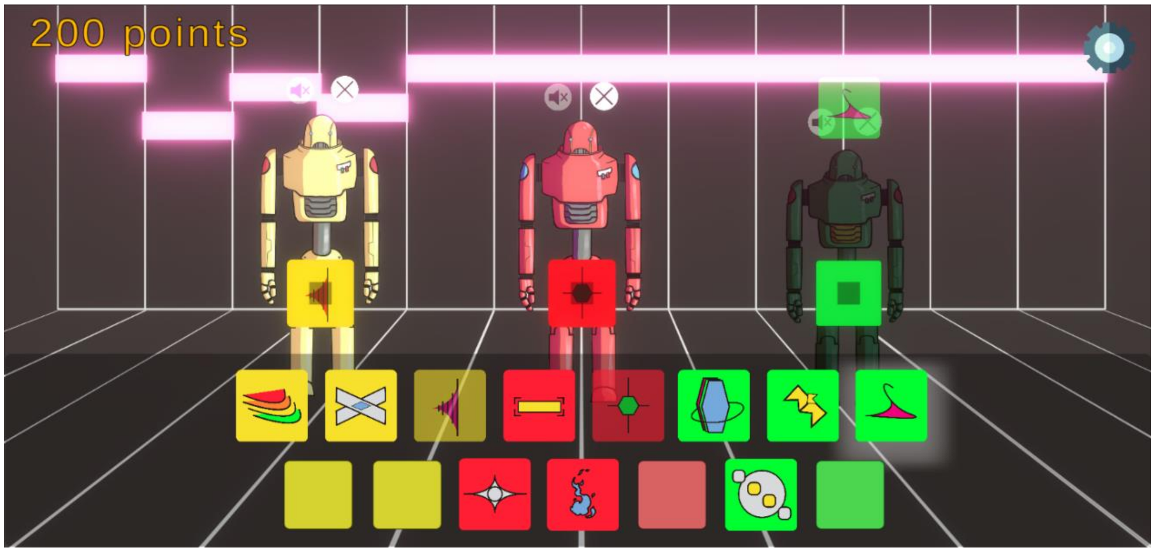


Fig 60 Level 1 of the Story Mode of Beat Repeat

- Level 2: In this level there is introduced one new robot and three new mechanics, the first mechanic is the audio effects, which work the same as the cross and silence buttons, where the player has to press the button when the circle closes. The second one is pressing two buttons at the same time; due to we are playing on smartphones the player needs to understand and learn that he can interact with the game using more than one finger. And finally, there is a new mechanic where the player has to drop a melody when there is already one melody playing on that robot, in this case, the player has to drop the melody just at the same time the icon goes through the reference square, if he does it before, a message saying “Too early” will appear, and if he does it after a message of “Late” will appear. There is no need for a tutorial to learn these new mechanics, all the game needs is the repetition of these to make the player understand why he is failing; that is why these mechanics are repeated more than one time through this level.

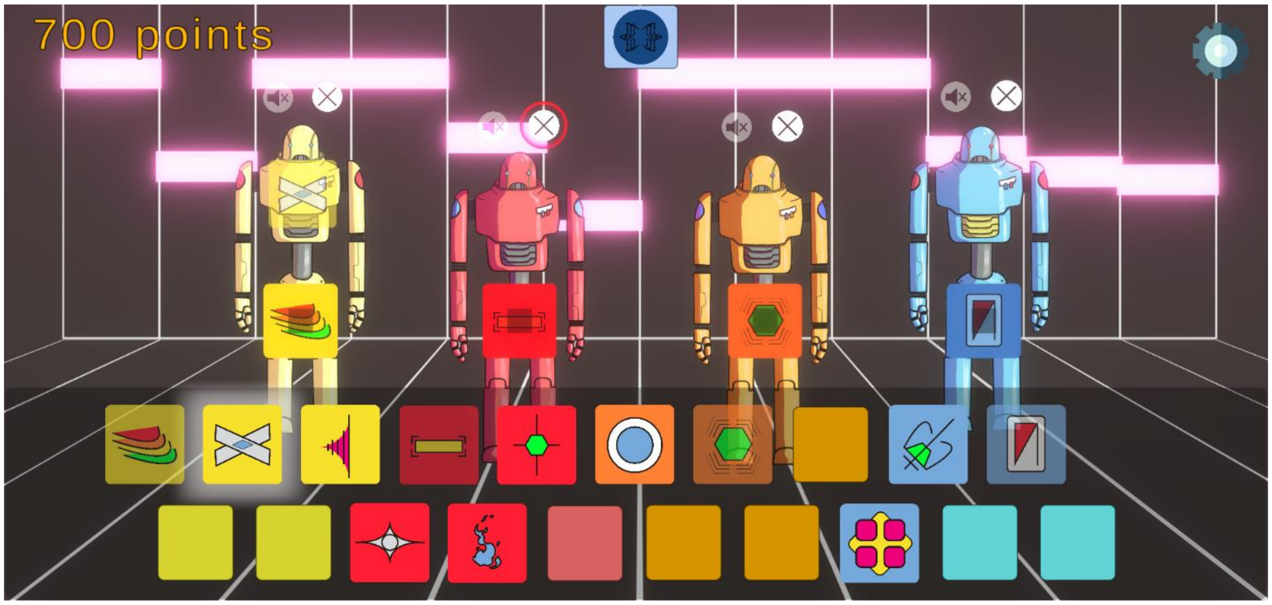


Fig 61 Level 2 of the Story Mode of Beat Repeat

- Level 6: I decided to make 2 intro/tutorial levels to make the player learn and understand the mechanics of the game, but I also wanted to know how difficult a level could be. That is why, for this prototype, I decided to create level 6, where the player needs to apply all the mechanics of the game to get a good score point.

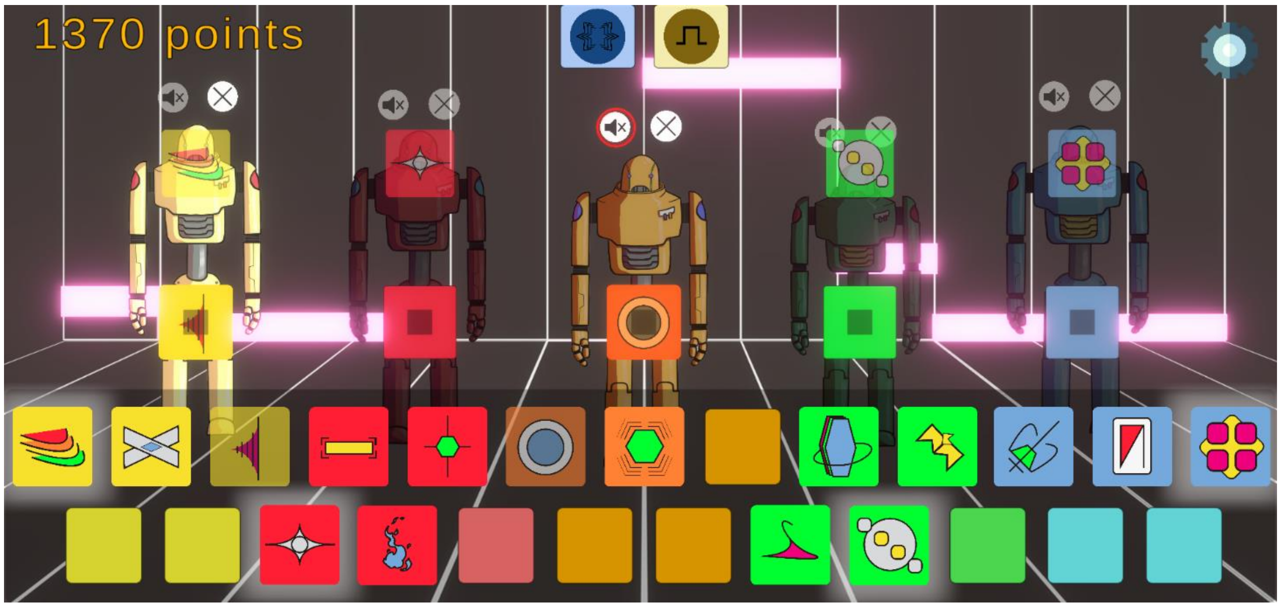


Fig 62 Level 6 of the Story Mode of Beat Repeat

The points system in this mode is the following one:

- Perfect: +100 points.
- Too Early: +20 points.
- Too Late: +10 points.
- If the desired melody is not playing when the music loop starts: -100 points.
- If the cross, silence or audio effects buttons are pressed when it is not required: -50 points.

I personally think that I did a great job with this game mode because, even if it is based on other games like Guitar Hero, the system of the buttons and the idea of teaching how to create a song instead of just pressing the buttons to follow the rhythm of the music, is totally original and I do not know any existing game like this.

3.3. Options

In the options screen different parameters appear to let the player change them and choose which fits better for him while playing the game.

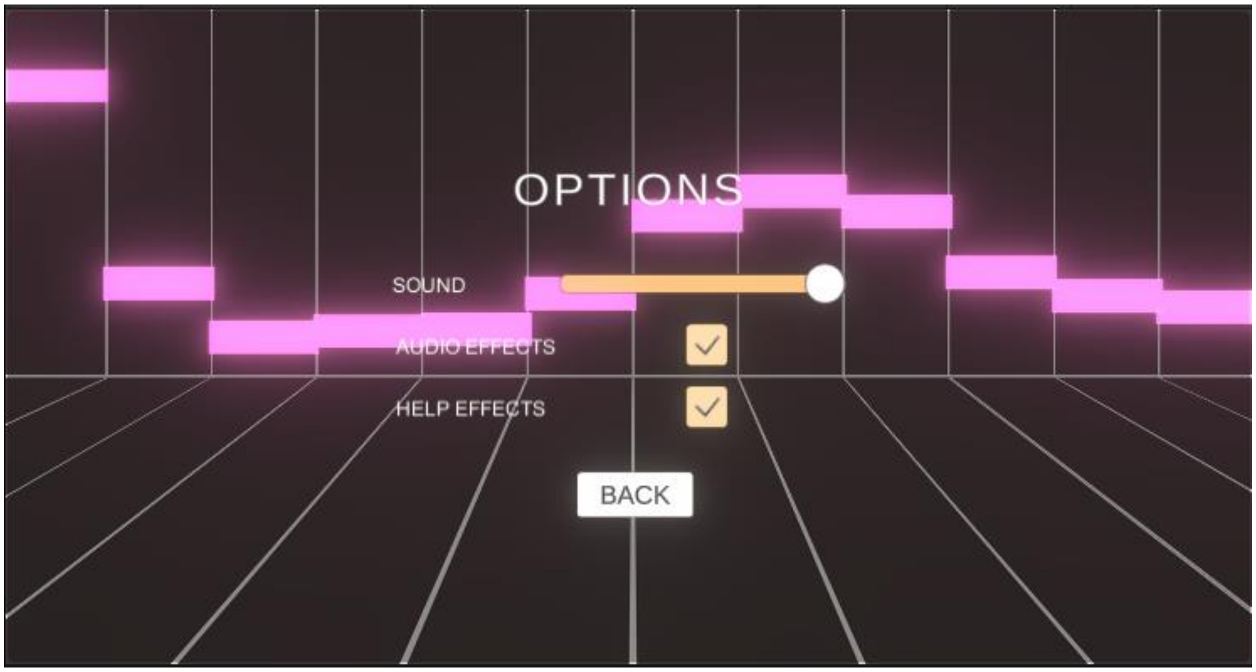
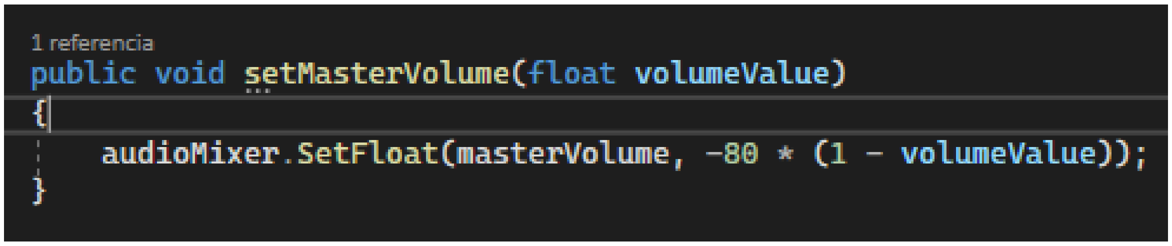


Fig 63 Options screen in Beat Repeat

First, the Options menu shows an option every game need, changing the level of the sound. The player may need to adjust the level of the game for any possible reason. To calculate this, as the slider has values that go from 0 to 1, we need to make it from –80 to 0, as the values of the mixer have this range. To make this happen, we change the value of the Master in the mixer by calculating Fader = -80 * (1 – slider).



```
1 referencia
public void setMasterVolume(float volumeValue)
{
    audioMixer.SetFloat(masterVolume, -80 * (1 - volumeValue));
}
```

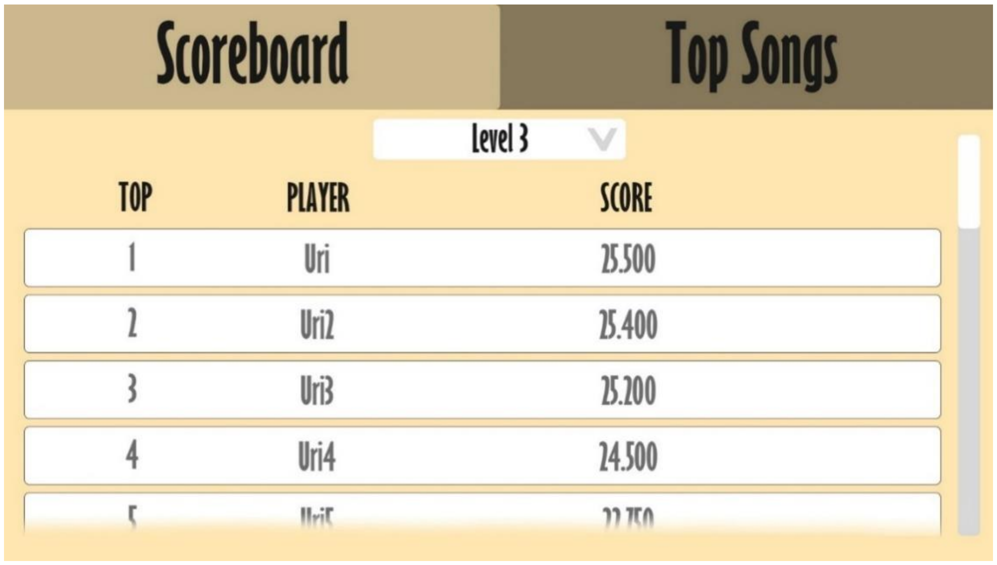
Fig 64 Code to calculate the Output Volume depending on the slider of the Options menu in Beat Repeat

Another option this menu has is the possibility of disabling sound effects, like the sound it makes when the player presses a button on a menu, or other that could be added in the future.

And finally, this menu offers a button that gives the player the opportunity to choose if he wants different helps on the story mode levels or not. This helps refer to the ones mentioned before which will highlight the buttons of the melodies that need to be played (see Fig 59).

3.4. Scoreboard and Top Songs

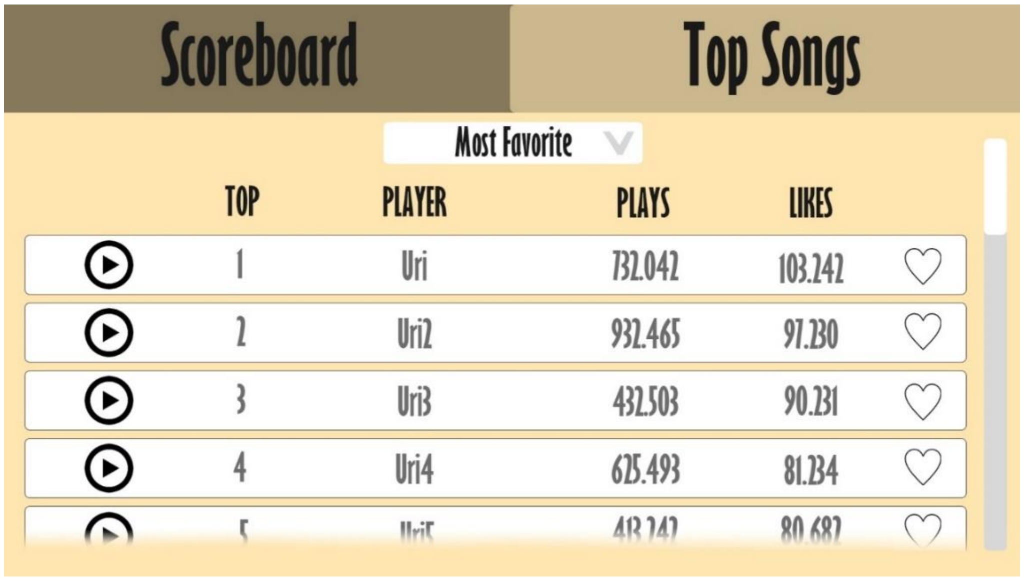
In the Scoreboard menu the player will be able to see how many points compared to other players around the world he has on the different levels of the Story Mode. This is a very important point for the more hardcore players to let them see how many points they have compared to the other ones and make them want to overcome their own record again and again.



TOP	PLAYER	SCORE
1	Uri	25.500
2	Uri2	25.400
3	Uri3	25.200
4	Uri4	24.500
5	Uri5	22.750

Fig 65 Concept Idea of the Scoreboard screen in Beat Repeat

Another important feature this menu would offer in the future is the possibility of listening to the songs the community has created. A button for recording and uploading would be on the game, letting to share with other people your own creations and listening to the ones from the others. In this section different filters like *Most Favorite, Most Played or Most Recent* would appear.



TOP	PLAYER	PLAYS	LIKES
1	Uri	732.042	103.242
2	Uri2	932.465	97.230
3	Uri3	432.503	90.231
4	Uri4	625.493	81.234
5	Uri5	413.142	80.682

Fig 66 Concept Idea of the Top Songs screen in Beat Repeat

3.5. Shop and Monetization of the game

In this section it is going to be explained what the shop menu will offer and how the game would earn money in a possible future where the game would be published.

First, it is important to mention that this game would be free to play, so everyone could download it for free and try it totally free. Even so, different things would form part of the shop list.

The idea is to sell different packs with different settings for each one. Each pack would be inspired on different thematic so each one would have different melodies, sounds, robot skins and sceneries according to a specific theme. Some examples could be a Futuristic Pack, Country Pack or even a typical Beach Inspired Pack.

Each pack, apart from adding different melodies, sounds, etc., would also add new stories and new levels for the Story Mode, giving the player not only the possibility of creating new songs, but also more game-time to spend overcoming the new levels and trying new records. Each pack would cost something around 1’99€ and 4’99€.

Another possibility the shop could offer would be to sell the different levels of the story mode, so the player would not need to go through every level to unlock all of them.

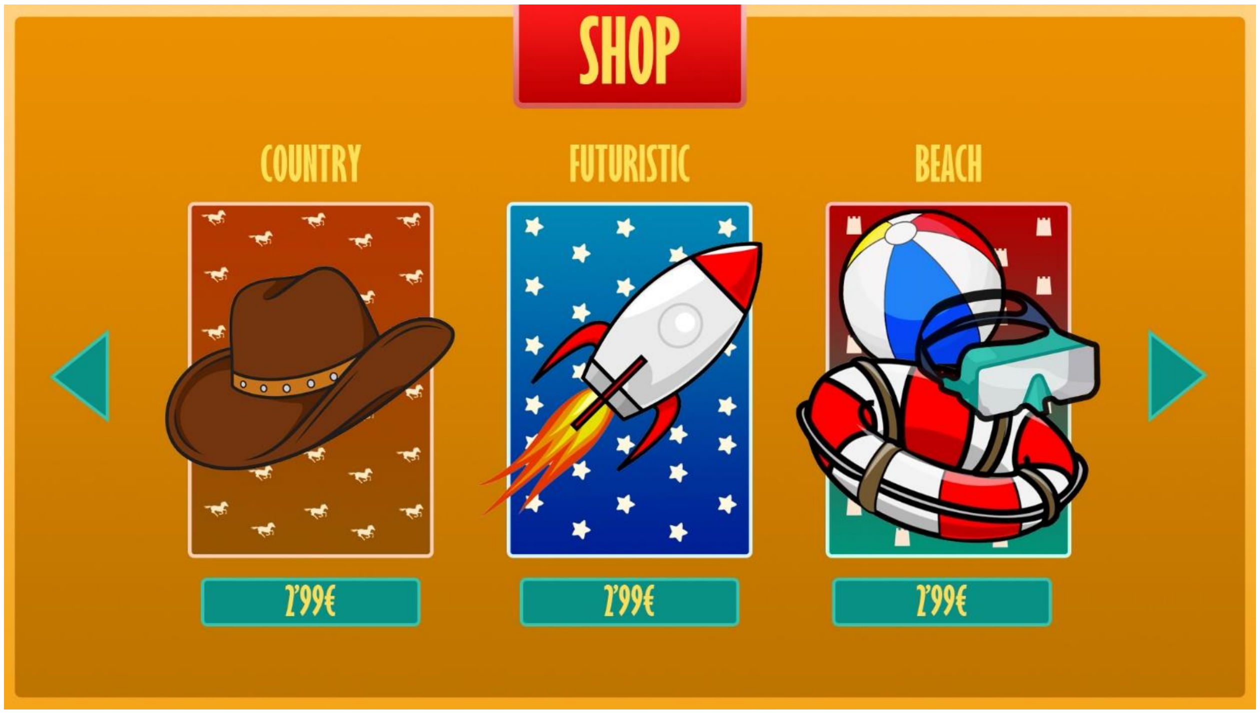


Fig 67 Concept Design of the Shop screen in Beat Repeat

- Country: A story about cowboys and electronic music with country details.
- Futuristic: A story about space travel and electronic music with futuristic synthesizers.
- Beach: A story about the robots going to the beach and electronic music with Latin music details.

4. Development of the App

The most important scripts are explained in this section in order to be able to understand the main mechanics of the game, as well as an explanation of the design of the user interface, shaders and music production for the game. Also, in this chapter it is illustrated what is my idea for the character design of the game and what animations have been created for this prototype.

4.1. Programming on Unity

In this section different scripts and programming ideas applied to this project are explained. In order to understand how the structure of the game is and how every part in the game is related to the others, here is a diagram that entails it:

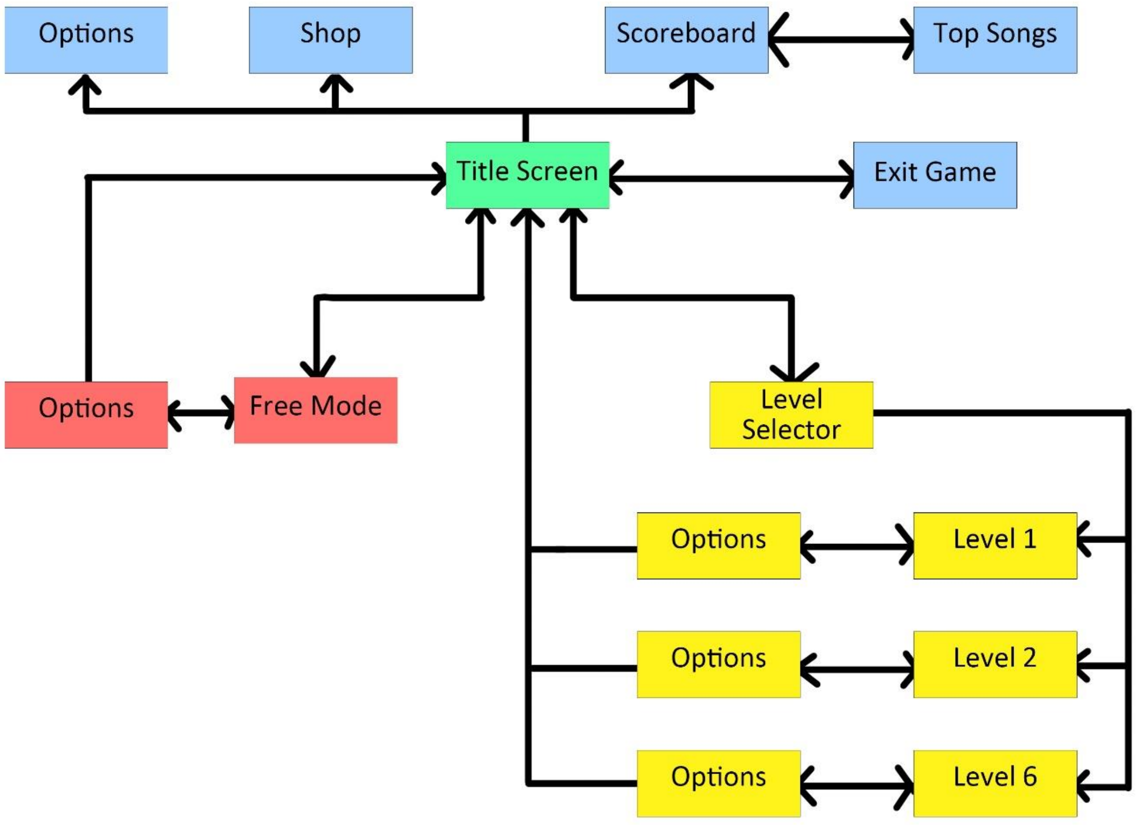


Fig 68 Flux Diagram of the sections of the game

4.1.1. Basic functionality

Referring to the IDE system mentioned before on the *2.1. What's Unity*, where the user has to drag and drop different scripts or game objects to different variables, we could even say there is not much difference between this and my videogame, where the player has to drag the different melodies and drop them to the robots to make them come to life playing music.

The main system of the game remains on the script called “DropMelodies”, where the update function is waiting for the player to drop an icon to one of the robots. When one melody is dropped to the robot, the “wantToPlay” Boolean variable will activate and, if there is no other instruments currently playing, it will start to play; if there is already music playing, the robot will have to wait until the next start of the music loop to start playing.

The loop function is a process called on the “SoundController” script where every eight seconds a bool will be activated to allow the “wantToPlay” melodies to start playing. This function starts counting when the player drops a melody for the first time to one of the robots and gets reset when every melody is removed. To have control of all the melodies, there is only one “SoundController” script with the reference of all the robots, unlike the “DropMelodies” script, which every robot has one.

Going back to the “DropMelodies” script, it is also the one in charge of detecting when the user goes over the robots with a melody, it is the one to detect when the user presses the robots to change the sound of the melody, and the one to control the buttons that allow the player to silence or quit the melodies.

Continuing with important scripts, we also have the “DragDrop” script, which allows to transform the position of the melody icons and make it follow the mouse. When I started with this script, the GameObjects followed the mouse but there was a little offset, so it did not follow the mouse perfectly. The reason for this was the difference between the mouse movement and the canvas scale, due to the varied sizes of the canvas depending on the size of the screen, made the drag system not work properly on different appliances. To correct this, we just have to divide the anchored position for the scale factor of the canvas.

To have a better understanding on the logic of the game, a representation in a flux diagram has been made (see Fig 69).

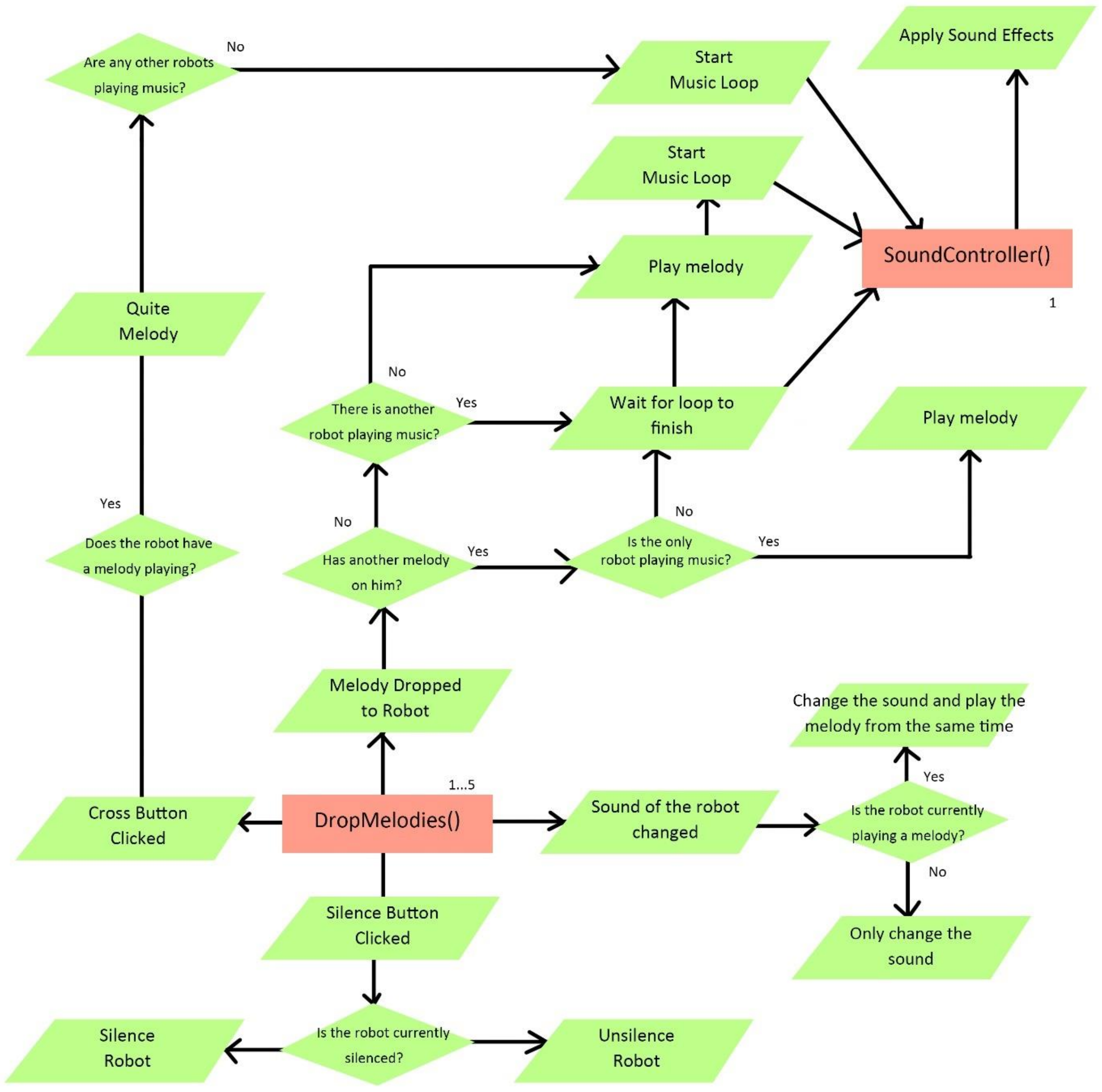
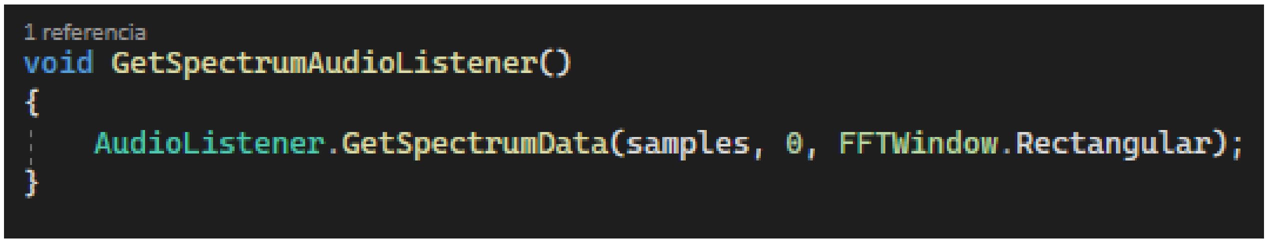

Are any other robots playing music?
No
Start Music Loop
Apply Sound Effects
Start Music Loop
Play melody
SoundController()
1
Quite Melody
No
There is another robot playing music?
Yes
Wait for loop to finish
Play melody
Yes
Does the robot have a melody playing?
No
Has another melody on him?
Yes
No
Is the only robot playing music?
Yes
Melody Dropped to Robot
Change the sound and play the melody from the same time
1...5
Yes
Cross Button Clicked
DropMelodies()
Sound of the robot changed
Is the robot currently playing a melody?
No
Silence Button Clicked
Only change the sound
Silence Robot
Is the robot currently silenced?
Unsilence Robot

Fig 69 Flux Diagram of the main logic of the game

4.1.2. Audio-visualization to the screen in real time

Making different items react to the music on the screen in real time while playing the game was one of the main objectives when starting the project. This functionality remains principally on one Unity function, the AudioListener.GetSpectrumData().



```
1 referencia
void GetSpectrumAudioListener()
{
    AudioListener.GetSpectrumData(samples, 0, FFTWindow.Rectangular);
}
```

Fig 70 Code Function GetSpectrumData() in Beat Repeat

The first parameter of this function needs to be an empty float array which its length will determine the number of samples our data will have. This data depends on the sample rate of the audio sources, in our case we have a sample rate of 44100Hz, so Unity will take only half of the 44100 samples, since Unity and some other FFT-based algorithms, only use half of the frequencies because the second half is a mirror of the first one. Then, in this case there are 22050 samples selected into 512 (length of the array selected).

The number of audio samples taken in the time domain to perform the FFT is double than the number of frequency bins returned by the function GetSpectrumData(). Meaning that if I use an array of 512 floats, the size of the FFT is 1024. That means that the size of the buffer is 1024samples/44120Hz = 23’22ms, this is a good small length so there is not much latency. Unity does not explain this system of audio processing in any of its documentation, but as Unity is processing this information in real time, it probably is like that. To test it, I am going to try different numbers of frequency bins returned by the GetSpectrumData() function, and if the reaction of the spectrum gets slower when the number of bins increases, it means this is the method Unity uses.

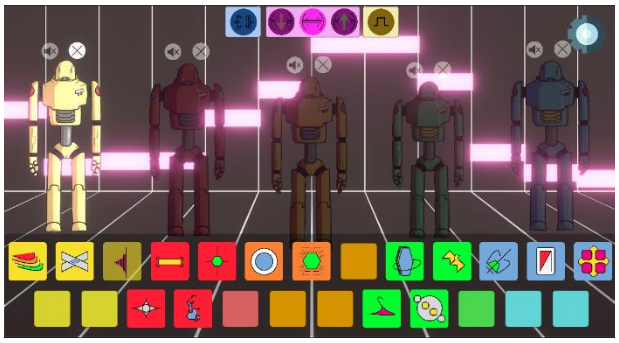


Fig 72 Game using 8192 samples for GetSpectrumData()

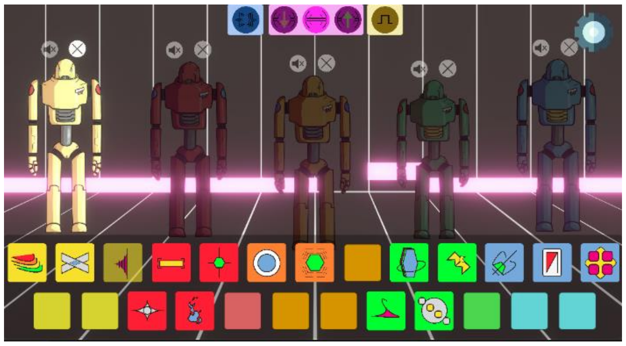


Fig 71 Game using 512 samples for GetSpectrumData()

As we can see in the comparison of these two images taken at the exact same moment, different number of frequency bins have been applied, the one on the left (with 8192 frequency bins) is slower than the image on the right (with 512 frequency bins). This is because the size of the buffer is increased if the FFT is bigger. In this case the size of the buffer is 8192samples/44100Hz =

185,58ms, so the reaction of the spectrum in the image on the left is 185’58ms - 23’22ms = 162.36ms slower than the right one.

The second parameter (see Fig 70) is the channel the user wants to sample from, in this case it is the channel 0 because it is sufficient with the audio in mono.

And the last parameter (see Fig 70) is what type of window do we want to use for the FFT when sampling. For this last parameter we can select a default window Unity offers like a rectangle, triangle, Hamming, Hanning or Blackman. This is a very simple way to get the frequency spectrum of our audio, but it has one perk, when using a default window by Unity, the window is being calculated on every iteration, so using a complex window can slow down the process. It seems like Unity is calculating the window at each iteration because in its documentation it says that using a complex window can reduce the speed, which does not make sense because a window can be calculated once and be used as many times as desired. So, I decided to check if a previously created window function could be used instead of the default one.

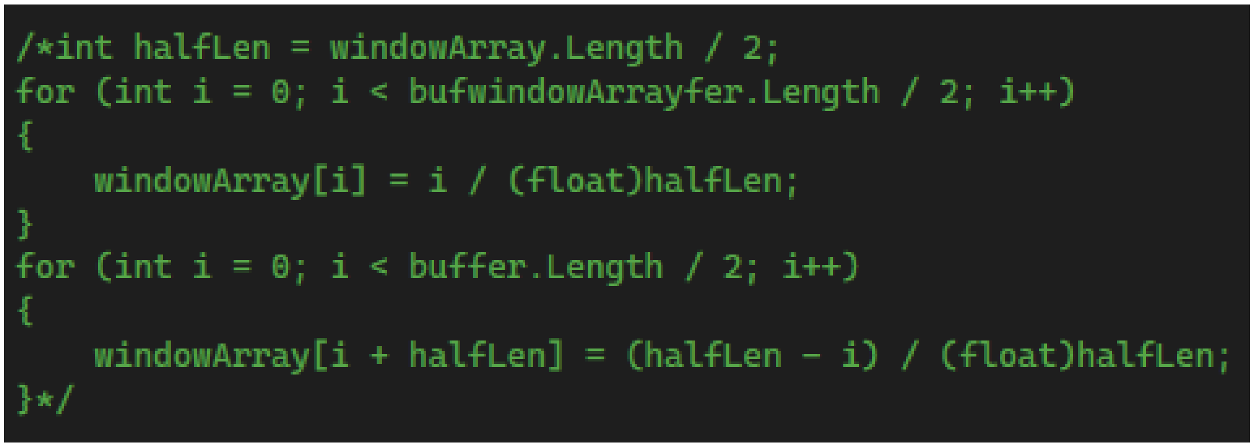

```
/*int halfLen = windowArray.Length / 2;
for (int i = 0; i < bufwindowArrayfer.Length / 2; i++)
{
    windowArray[i] = i / (float)halfLen;
}
for (int i = 0; i < buffer.Length / 2; i++)
{
    windowArray[i + halfLen] = (halfLen - i) / (float)halfLen;
}*/
```

Fig 73 Code example to calculate a Triangle Window

If we could use a personalized FFTWindow to consume less resources, we could calculate some window, in this case a Triangle one, by our own and use it on the GetSpectrumData() function, however, we cannot use a float array instead of the FFTWindow function.

So, to have good spectrum data and not consume so many resources the best option is using a Triangle Window, due to the results with a Rectangular are not very satisfactory.

Now that all the spectrum data is in 512 samples, it is time to put those frequencies into the number of bands required. In this case, the game has 12 sliders on the background which we want to react to the music. Each band represents a portion of frequencies from lows to highs from left to right. As the frequency domain has a logarithmic scale, it would be difficult to know how many samples should be in each band to make the sensation that each band has the same number of frequencies. For that, the mel scale needs to be used.

We take the maximum frequency of our audio, which is half of the sampling rate, in this case 22050Hz (44100/2), then, with this frequency we calculate the maximum mels with the corresponding formula:

$$m_{max} = 2595 \log_{10}\left(1 + \frac{22050Hz}{700}\right) \; = \; 3923'34\, mels$$

Equation 12

Now we divide the result into 12 bands to know how many mels correspond to each band:

$$\frac{3923'37mels}{12\, bands} \; = \; 326'94\frac{mels}{band}$$

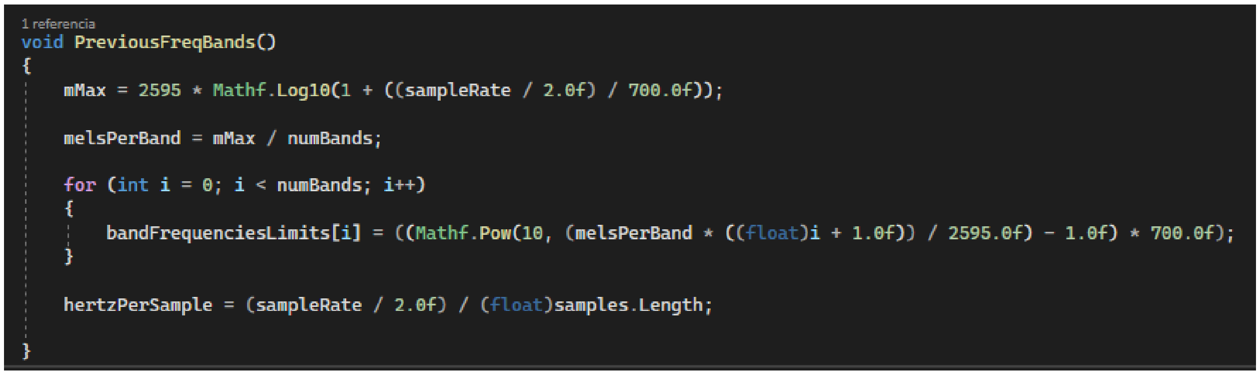
Equation 13

Now, for each band in mels, we calculate its correspondent frequency in Hertz with the same formula, and we have:

- F1 = 235’59Hz
- F2 = 550’49Hz
- F3 = 971’36Hz
- F4 = 1533’88Hz
- F5 = 2285’72Hz
- F6 = 3290’61Hz
- F7 = 4633’72Hz
- F8 = 6428’86Hz
- F9 = 8828’19Hz
- F10 = 12035’06Hz
- F11 = 16321’24Hz
- F12 = 22050Hz

An important detail to mention from this part is that I have selected a sample rate of 22050, when in reality, the minimum audible frequency is 50-60Hz, however, this is not a critical problem because in this case we do not need a real representation of the frequency spectrum, that is why the frequencies of the first band go from 0hz to 235’59Hz.

Before continuing with the calculations, we need to know how many Hertzs correspond to each 512 samples of the spectrum. To do that we divide 22050Hz/512samples = 43Hz/sample. To optimize the code and be able to change any of these parameters if it is needed, I wrote these calculations inside the script.



```
1 referencia
void PreviousFreqBands()
{
    mMax = 2595 * Mathf.Log10(1 + ((sampleRate / 2.0f) / 700.0f));

    melsPerBand = mMax / numBands;

    for (int i = 0; i < numBands; i++)
    {
        bandFrequenciesLimits[i] = ((Mathf.Pow(10, (melsPerBand * ((float)i + 1.0f)) / 2595.0f) - 1.0f) * 700.0f);
    }

    hertzPerSample = (sampleRate / 2.0f) / (float)samples.Length;
}
```

Fig 74 Code used to calculate the frequency bands in Beat Repeat

Knowing this, we just have to follow the next iteration, where *i* is the number of the band and *x_i* is the number of bins of the FFT from the band *i*:

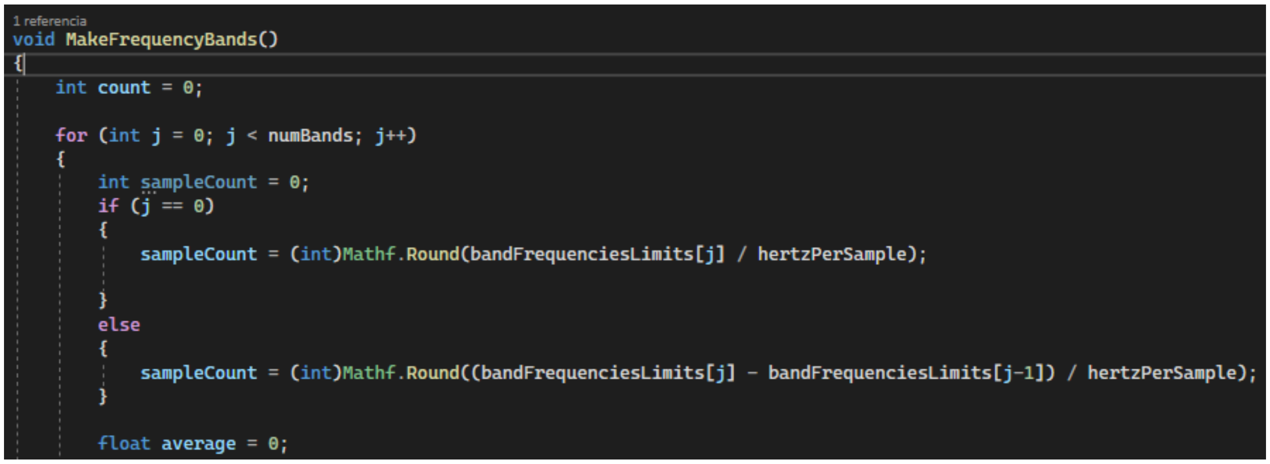
$$x\_i = \frac{(f_i)}{43\,\frac{Hz}{sample}} \quad for\ i = 1$$

$$x\_i = \frac{(f_i - f_{i-1})}{43\,\frac{Hz}{sample}} \quad for\ i \geq 2$$

Equation 14

With these calculations we got the corresponding number of samples to the following bands:

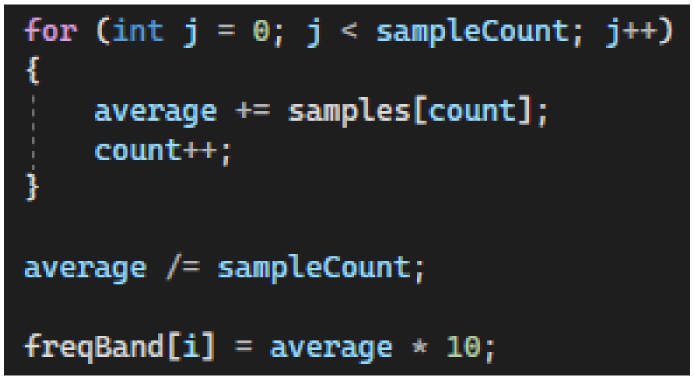
- Band 1 ↔ 5 samples
- Band 2 ↔ 7 samples
- Band 3 ↔ 10 samples
- Band 4 ↔ 13 samples
- Band 5 ↔ 18 samples
- Band 6 ↔ 23 samples
- Band 7 ↔ 31 samples
- Band 8 ↔ 42 samples
- Band 9 ↔ 56 samples
- Band 10 ↔ 74 samples
- Band 11 ↔ 100 samples
- Band 12 ↔ 133 samples



```
1 referencia
void MakeFrequencyBands()
{
    int count = 0;

    for (int j = 0; j < numBands; j++)
    {
        int sampleCount = 0;
        if (j == 0)
        {
            sampleCount = (int)Mathf.Round(bandFrequenciesLimits[j] / hertzPerSample);

        }
        else
        {
            sampleCount = (int)Mathf.Round((bandFrequenciesLimits[j] - bandFrequenciesLimits[j-1]) / hertzPerSample);
        }

        float average = 0;
```

Fig 75 Code used to distribute the frequency bins into each of the frequency bands in Beat Repeat

Once these samples are assigned to each of the frequency bands, it is time to calculate the average of the samples for each frequency band:



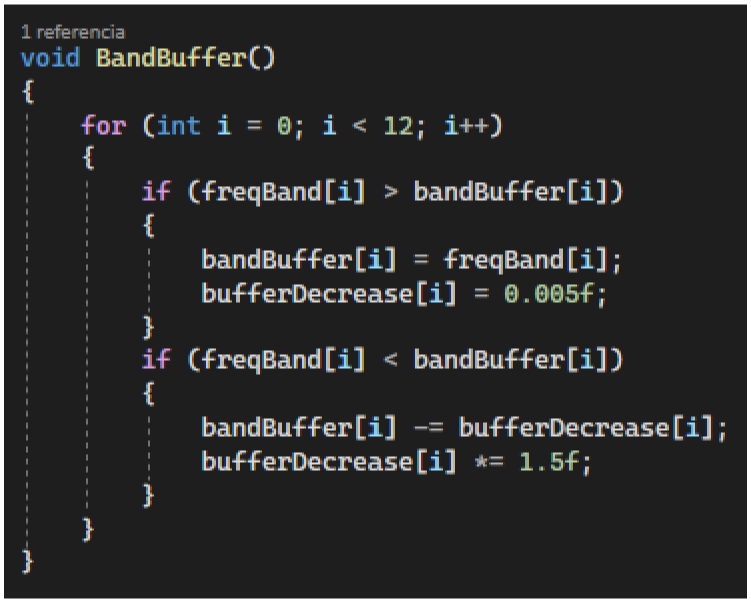
```
for (int j = 0; j < sampleCount; j++)
{
    average += samples[count];
    count++;
}

average /= sampleCount;

freqBand[i] = average * 10;
```

Fig 76 Code used to calculate the average amplitude of each band in Beat Repeat

*The average is multiplied by 10 to not be very small.

Right now, a perfect representation of the 512 samples divided into 12 has been created, however, we want the game to have a cool representation and not a realistic one, so some details have been changed on the steps mentioned before to have a nice and funnier representation of the sound.

For now, what we have is real time simulation of the audio frequencies joined in 12 bands, even so, if we use these bands for the sliders on the background, it can be a little annoying due to the fast changes the amplitudes of the different frequencies can have. To solve this, it is used a buffer that makes the movement of the sliders smoother. To do that, 12 new bands have been created, these bands take the same numbers as the original bands when going up, but have a specific decrease when going down.

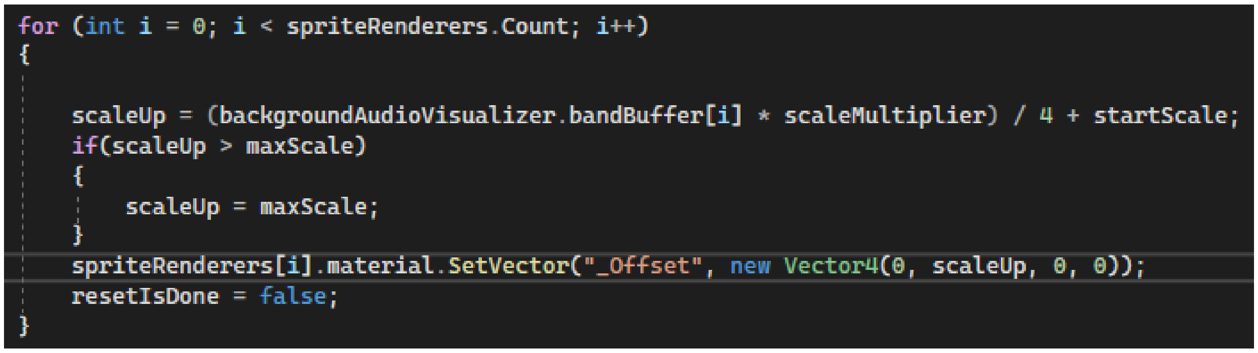


```
1 referencia
void BandBuffer()
{
    for (int i = 0; i < 12; i++)
    {
        if (freqBand[i] > bandBuffer[i])
        {
            bandBuffer[i] = freqBand[i];
            bufferDecrease[i] = 0.005f;
        }
        if (freqBand[i] < bandBuffer[i])
        {
            bandBuffer[i] -= bufferDecrease[i];
            bufferDecrease[i] *= 1.5f;
        }
    }
}
```

Fig 77 Code used to calculate smoother reactions to the music for each band in Beat Repeat

As we can see, when the original band is bigger than the new one, the new one is equalized to the original one, but when the new one is the bigger one, meaning that the original one is decreasing, the new one will be reduced only by 0.005. When this happens, the 0.005 variable will be multiplied by 1.5 to make the slider fall down faster on each iteration.

Once the 12 frequency bands have their amplitude average with the buffer on it, it is just needed to send them to the script responsible to convert that number into the sliders' amplitude on the background.



```
for (int i = 0; i < spriteRenderers.Count; i++)
{

    scaleUp = (backgroundAudioVisualizer.bandBuffer[i] * scaleMultiplier) / 4 + startScale;
    if(scaleUp > maxScale)
    {
        scaleUp = maxScale;
    }
    spriteRenderers[i].material.SetVector("_Offset", new Vector4(0, scaleUp, 0, 0));
    resetIsDone = false;
}
```

Fig 78 Code used to modify the sliders with each band in Beat Repeat

4.1.3. Story Mode

In this section we are going to explain the process of creating the Story Mode the game has apart from the free mode.

The way this mode works is simple and easy to program but requires precision and some patience. In the scene we can find different game objects that will be falling down, each of these are timed to apply different functions when arriving at the bottom:

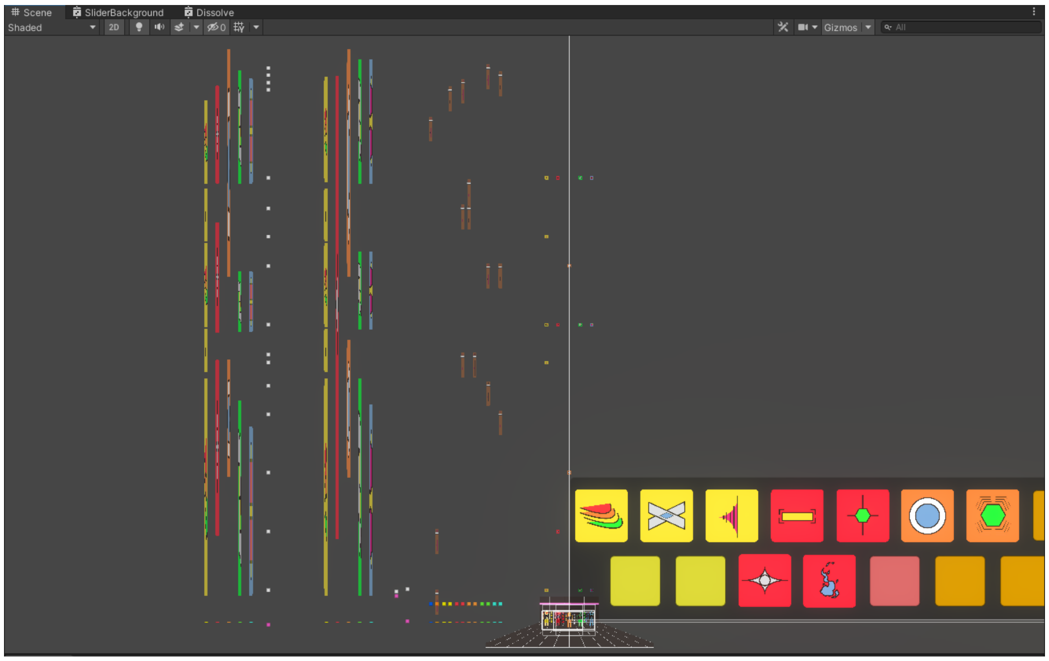


Fig 79 Distribution for Level 6 in Beat Repeat

First, we have the main melodies:

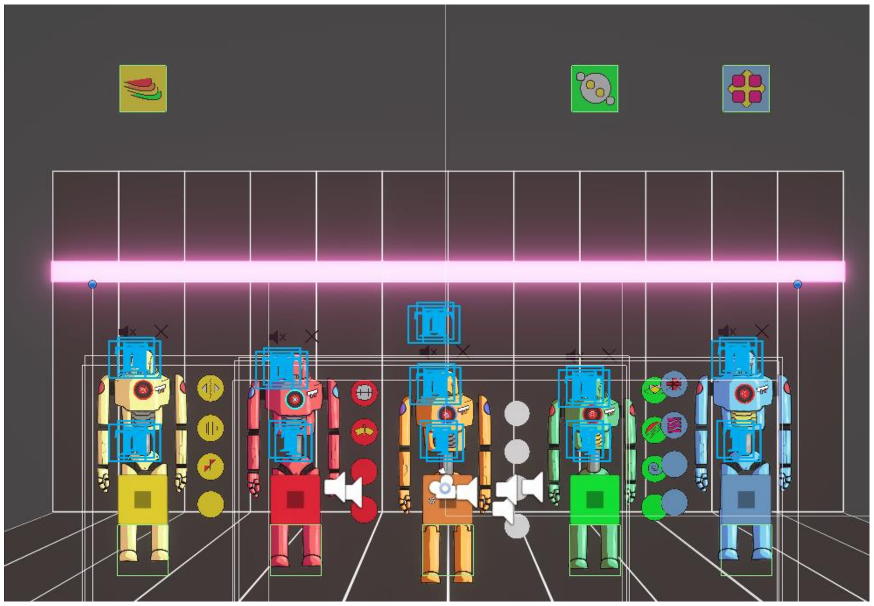


Fig 80 Showcase of all the elements in the scene of the Game Mode

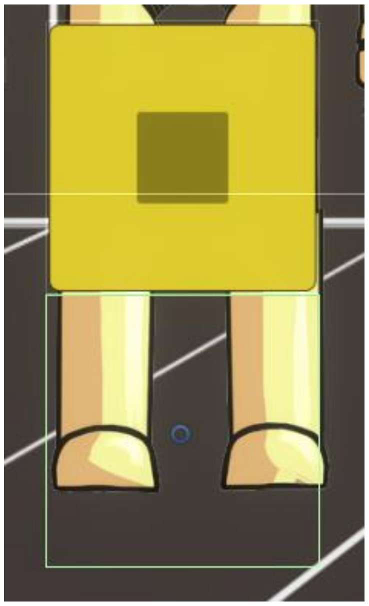


Fig 81 Example of the invisible square in the Game Mode

When the icon gets to the invisible square (see Fig 81) at the bottom of the main square, we check if the melody that should be playing has been dropped to the correspondent robot. We also check if the required melody has not been dropped on time; so, if the melody is dropped while the falling icon still is touching the invisible square, we will make appear a “too late” message to the screen and will add less points to the score.

Then, we have got the references for the mute, cross and audio effects buttons:

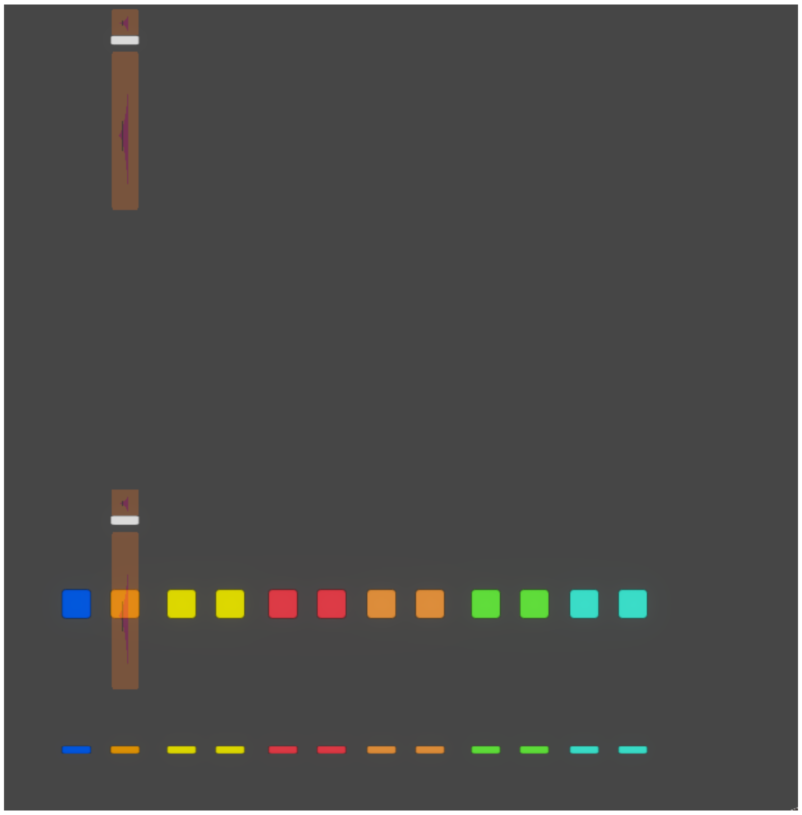


Fig 82 Example of the reference squares for the mute, cross and audio effects buttons in Game Mode

Every robot has two squares, one for the mute button and one for the cross button, then we also have one square for the filter button and one for the reverb button. The way this works is, when the white square of one of the falling objects gets to the correspondent big colored square, the animation of the cercle closing will start, then, depending on when the player presses the button, a message of Too Early (if pressed when the falling square below the white one is touching with the colored small square), Perfect (if pressed when the white square is touching the small colored square) or Late (if pressed when the falling square above the white one is touching with the colored one) will appear.

There are also some squares that trigger the help messages:

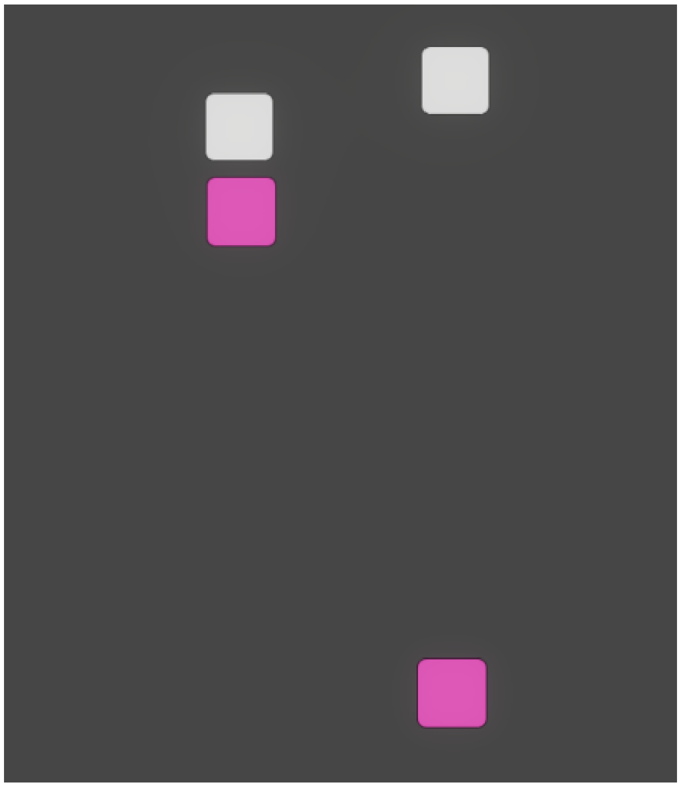


Fig 83 Example of the reference squares used for the help commands in Game Mode

As well as one square for each robot that checks if the melody playing is the correct one on every loop. To do that, the little squares on the bottom check if the square colliding has the same name as the one that is currently playing the melody:

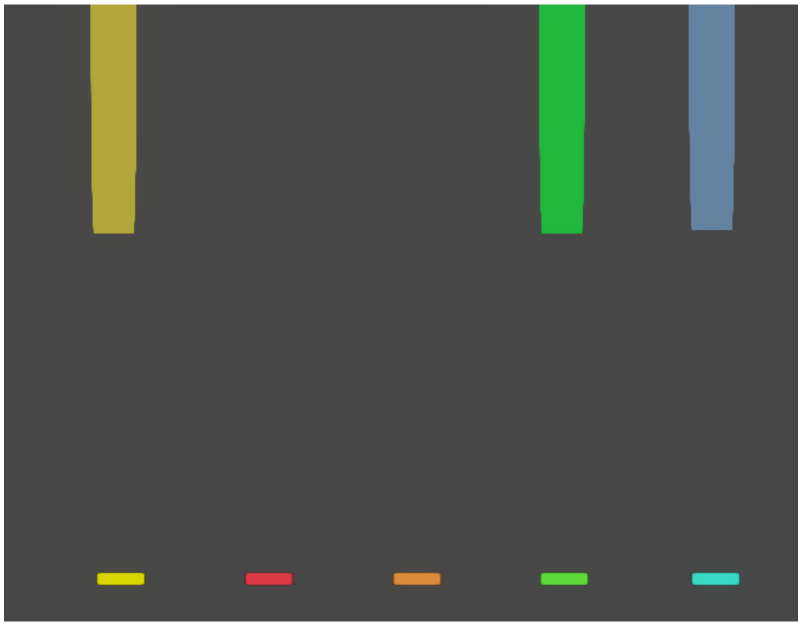


Fig 84 Example of the reference squares that check if the melody that is currently playing is the one that should be playing

And finally, we have one square for each robot that updates the image on the main square depending on the melody that has to be playing (see Fig 85).

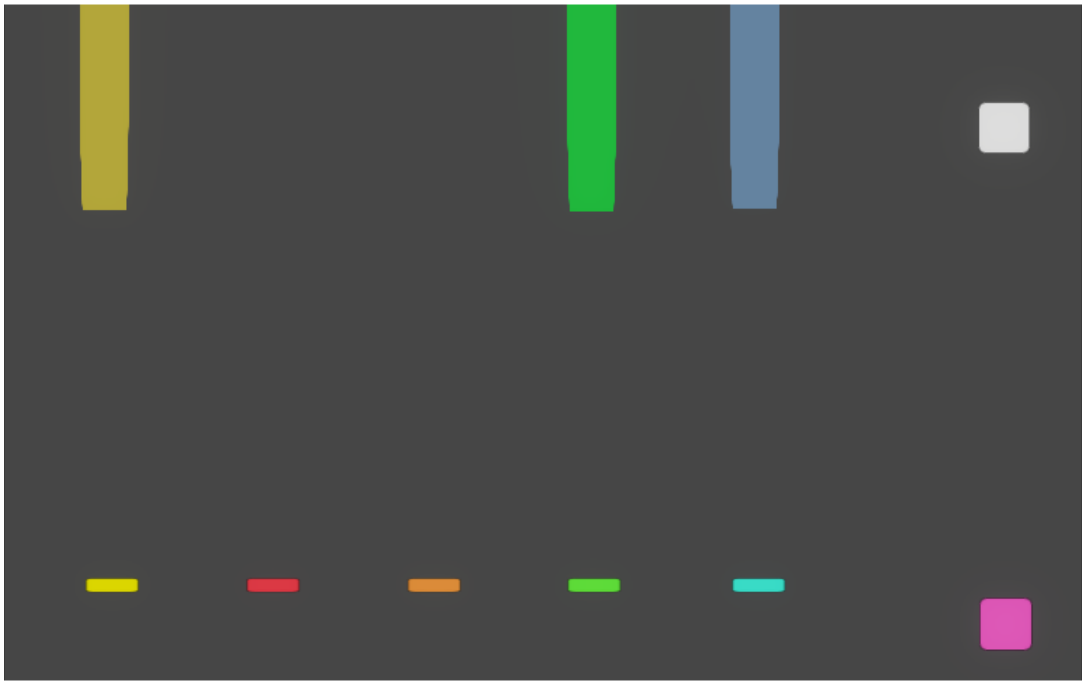


Fig 85 Example of the reference squares used to change the images that tell which melody has to be playing

The trigger on the right is to update the images when it is necessary, taking as reference the image colliding with the small squares on the bottom.

4.2. User Interface

In this section I will expose all the process that I followed to create the interface in my videogame.

The first idea I came up with was the idea of drag and dropping different melodies to each of the robots, because of that I needed some space on the screen for the different melody icons. The best place to set the icons was the bottom of the screen so we are able to see the top part of the robots and still have some space on top of the screen.

Due to we are creating a videogame for smartphones and people from 8 to 25 years old we need a very simple, clear and colorful interface. That is why I started with the idea of having different colors for each of the robots that could be related easily with the different melody buttons.

The next step is to distribute where the *Quit Melody*, *Silence Melody* and *changing between sounds* buttons would be. For that, as the Quit Melody and Silence Melody are needed for every robot and must be very accessible, the best place is above every robot. Now, I do not have so much space left on the screen, and I do not want to over saturate the player with too much information and a lot of buttons on the screen, so I came up with the idea of showing the different sounds buttons when pressing the corresponding robot. That is why when we press one robot, a small animation appears with 4 different new buttons, one for each sound; and a black panel with little opacity appears in the background to separate all the other things from the sound buttons.

And finally, we need to set the *Audio Effects* buttons, at the beginning I thought it would be interesting leaving some space at the laterals and put the buttons there, but for the final result, I want the robots to be the biggest possible, so it was not a possible option I did not want them to overlap. Then I thought I would add them on every robot, with the same mechanic as the *Changing Sounds* buttons, but I thought that could make it more complicated having that many buttons when pressing one robot, so to simplify I decided to globalize the effects and when they were pressed it would be applied to every robot, so what I did was to bring the robots down a little and leave some space at the top of the screen. As the melody buttons panel is a little transparent, we can still see a big part of the robots' legs.

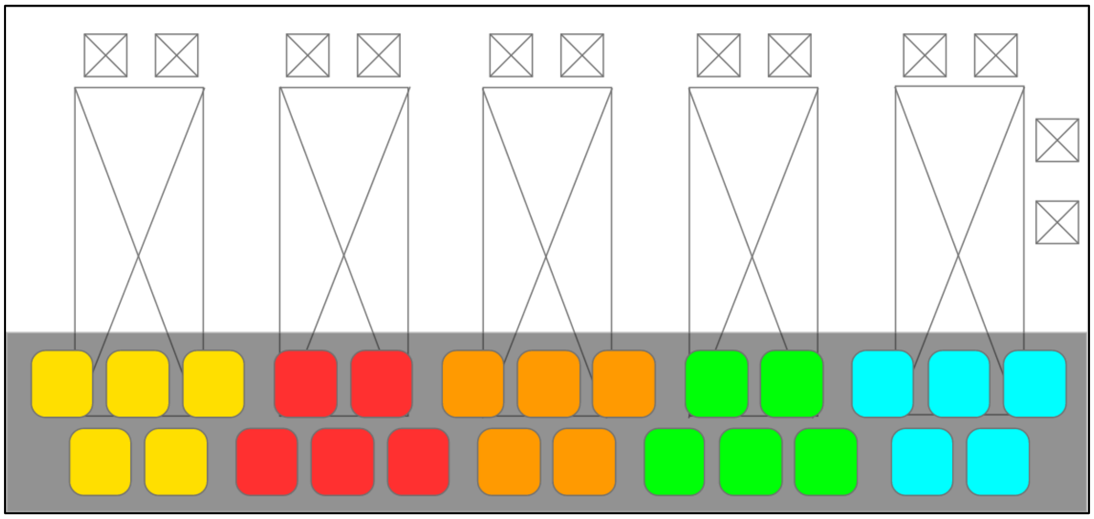


Fig 86 Wireframe of the first idea for the game interface

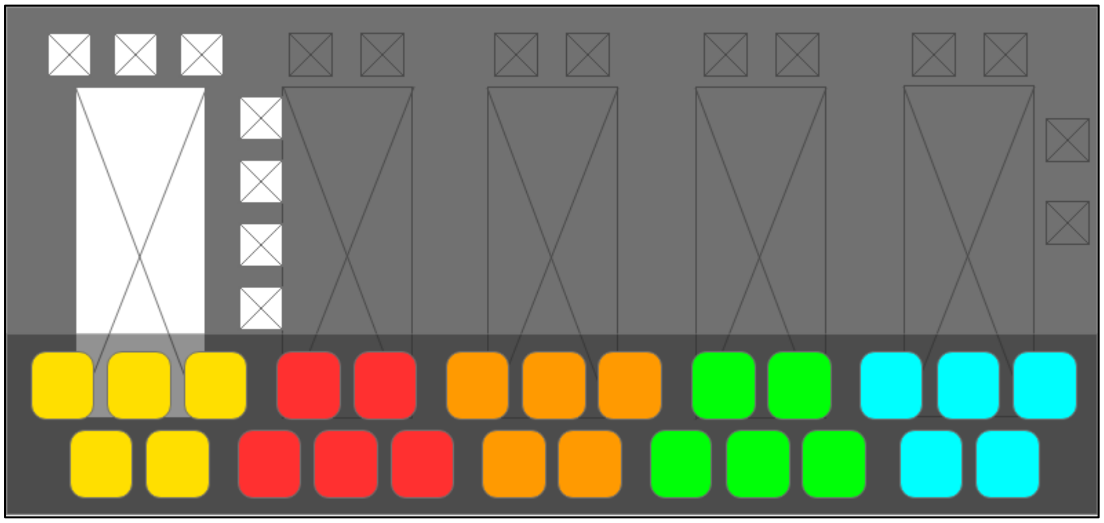


Fig 87 Wireframe of the second idea for the game interface

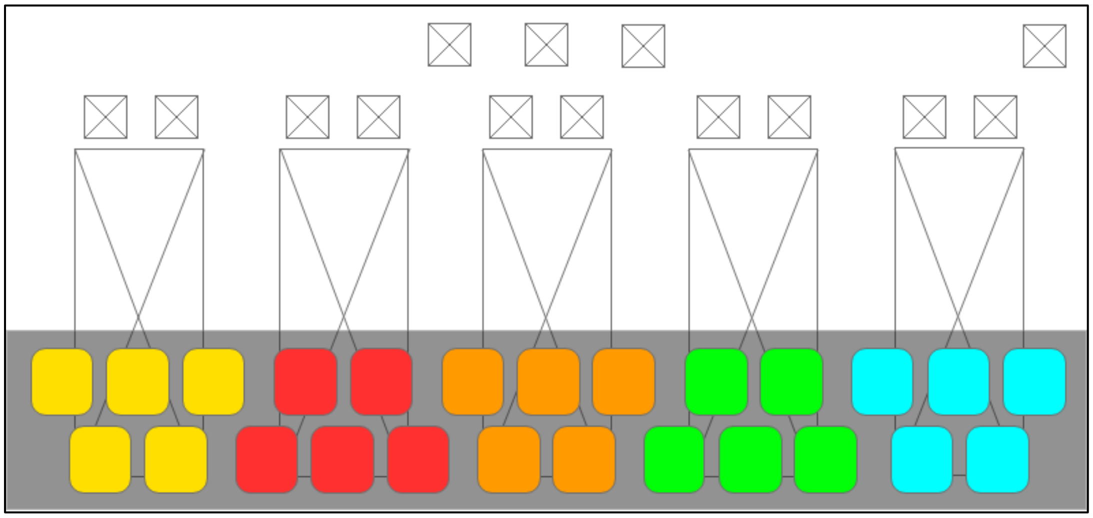


Fig 88 Final wireframe for the game interface

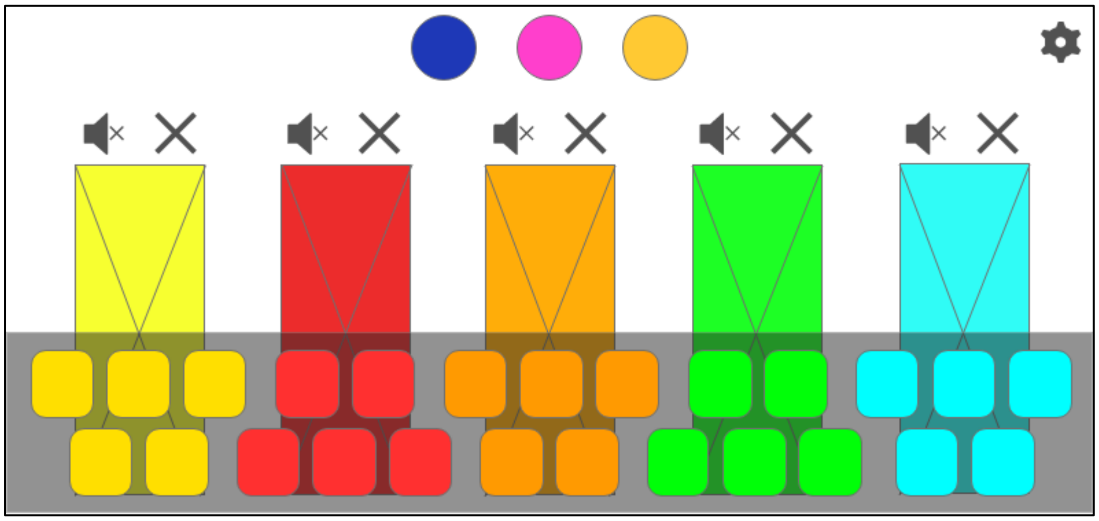


Fig 89 Mockup for the game interface

Now, all buttons need something that make them unique, not only for the Story Mode, to be able to recognize which buttons is required in every moment, but also for the Free Mode, where memorizing and being able to remember which buttons correspond to each melody, sound or sound effect is fundamental to be capable of playing with the combination the player likes the most.

So, I created simple designs to ease the recognition and distinction between them, trying to draw a different icon depending on what the melody made me feel.

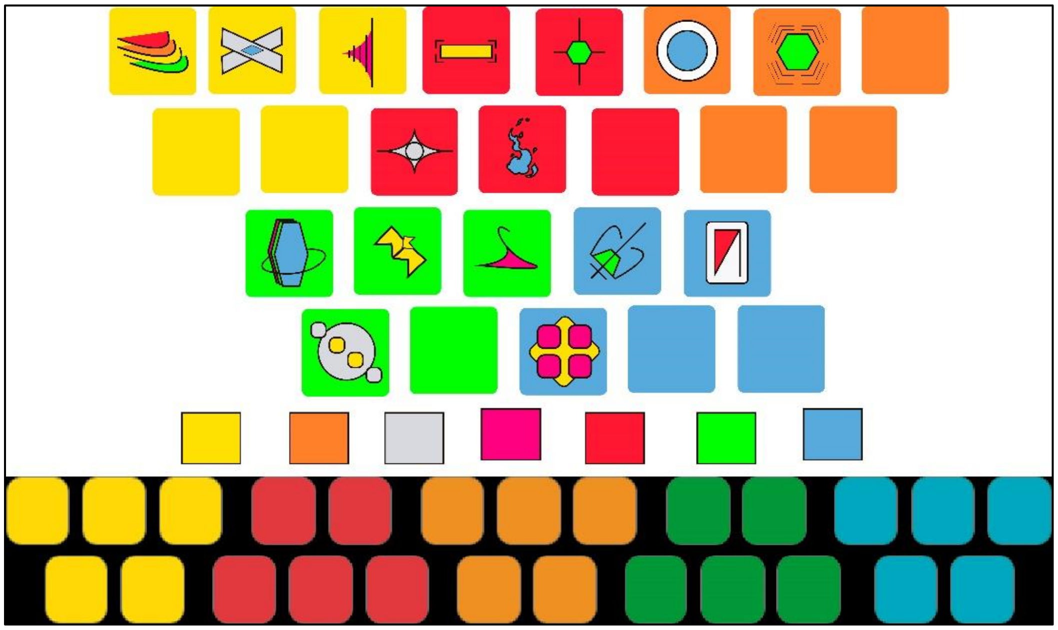


Fig 90 Button's structure and designs for Beat Repeat

We also have the main menu, where it is important to have the title of the game on it and access to all the sections of the game such as the Free Mode (Play), the Story Mode, the Shop, the Options Menu, and the Trophy section; as well as an Exit button in case the player wants to leave the game. At first, I thought of adding a text button to the settings menu, but as we are developing a mobile game, everything that can be displayed with a simple design will make it easier to understand and will leave more space on the screen.

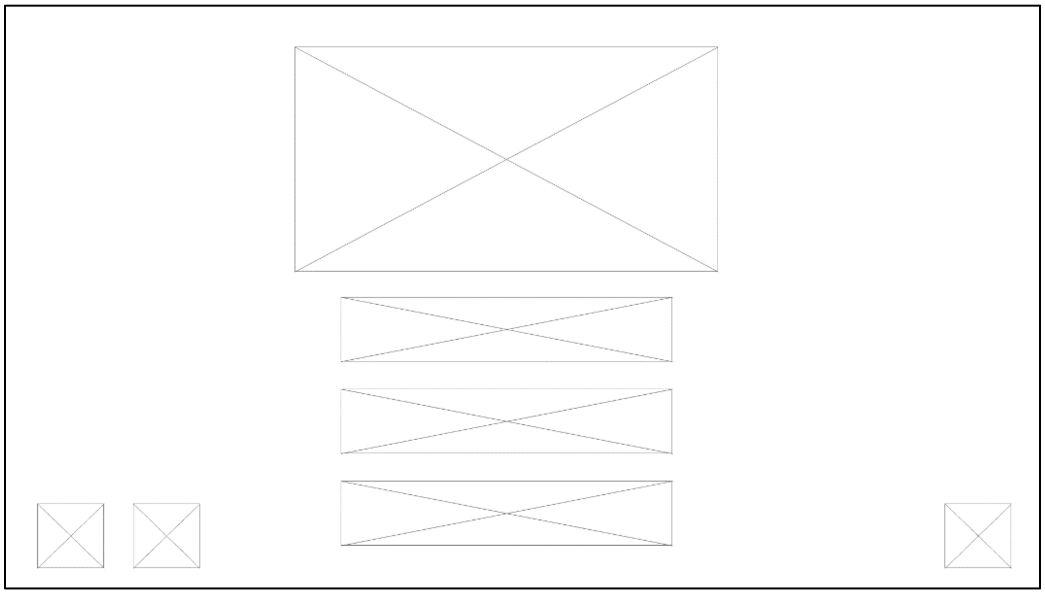


Fig 91 Title screen Wireframe in Beat Repeat

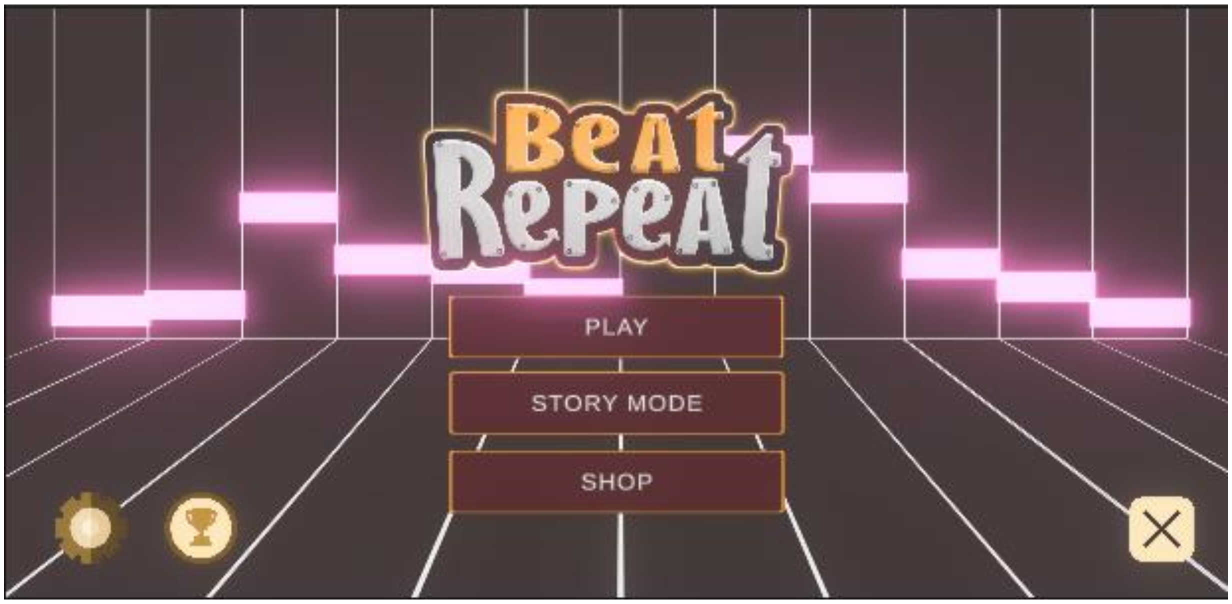


Fig 92 Title screen in Beat Repeat

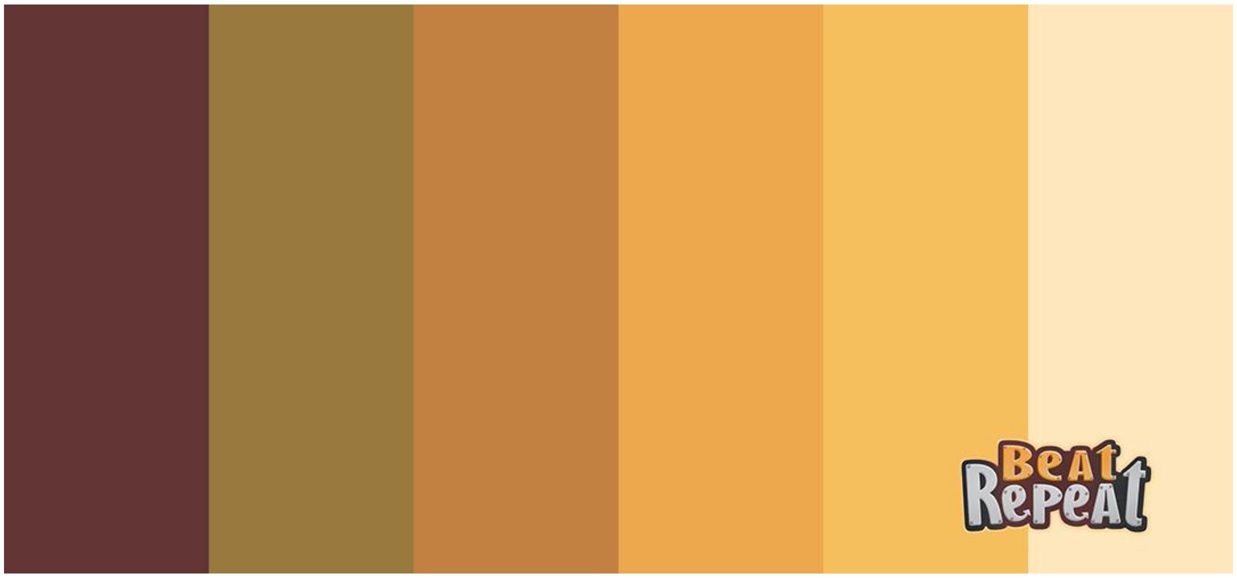


Fig 93 Color palette for Beat Repeat Interfaces

I wanted the orange to be the main color to give more importance and character to the singer robot, from there I used brown and pastel colors to fill the game with great tonality.

The logo of the game needs to be simple and memorable, generate impact and not easy to forget. Needs to be readable, even with a very small size and distinguishable, with black and white colors and in positive and negative.

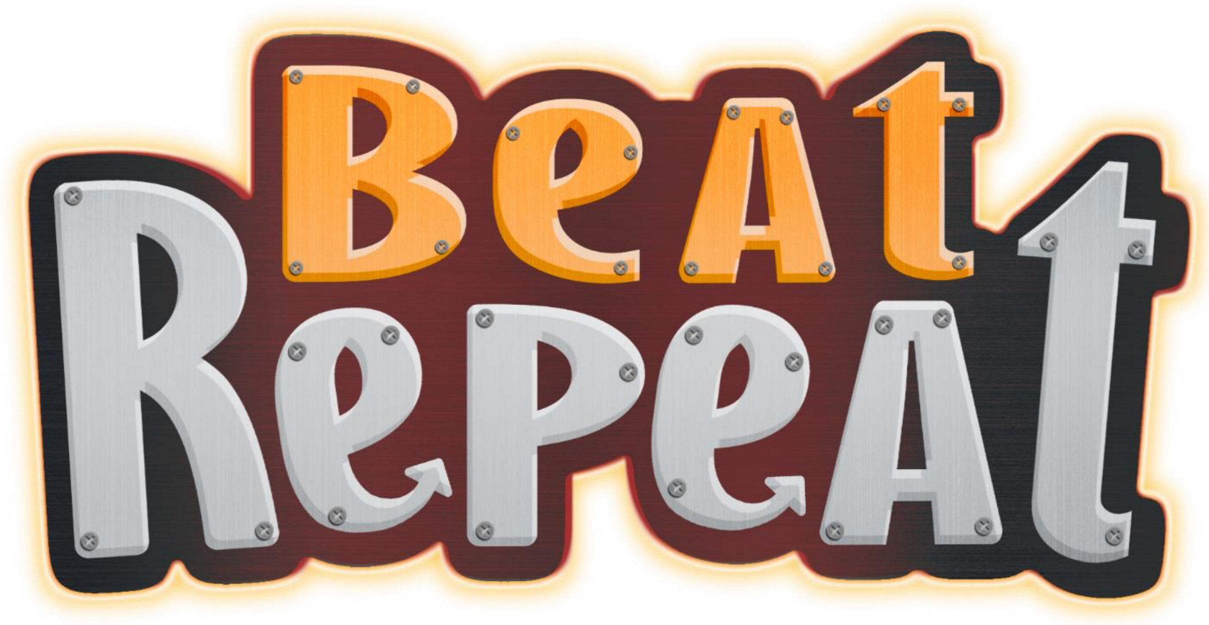


Fig 94 Beat Repeat Logotype

Fig 96 Beat Repeat Logotype in black and white

Fig 95 Beat Repeat Logotype with two tonalities

I wanted the logo to be cartoonish and with some robotic aspects, that is why I added a steel texture to the letters and some screws to it. To make it more distinguishable I made the word "Repeat" bigger making the "R" and "t" even bigger to surround the "Beat" word. Also, I combined some lowercase and uppercase letters and made the borders smoother. Another important trait of the letters is that I modified the letters "e" on the word repeat to make them have an arrow on them, recreating like a repeat logo.

An important design of the game is the shop menu. I wanted the shop to have a great design, even if it was only a concept idea, but as the game is free to play, the shop needs to be colorful and make you want to buy the different packs by just looking at them.

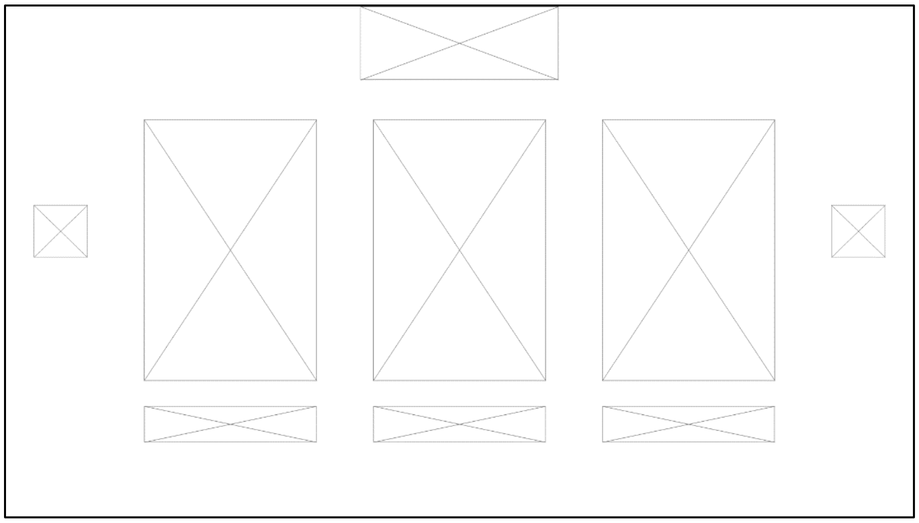


Fig 97 Shop screen Wireframe in Beat Repeat

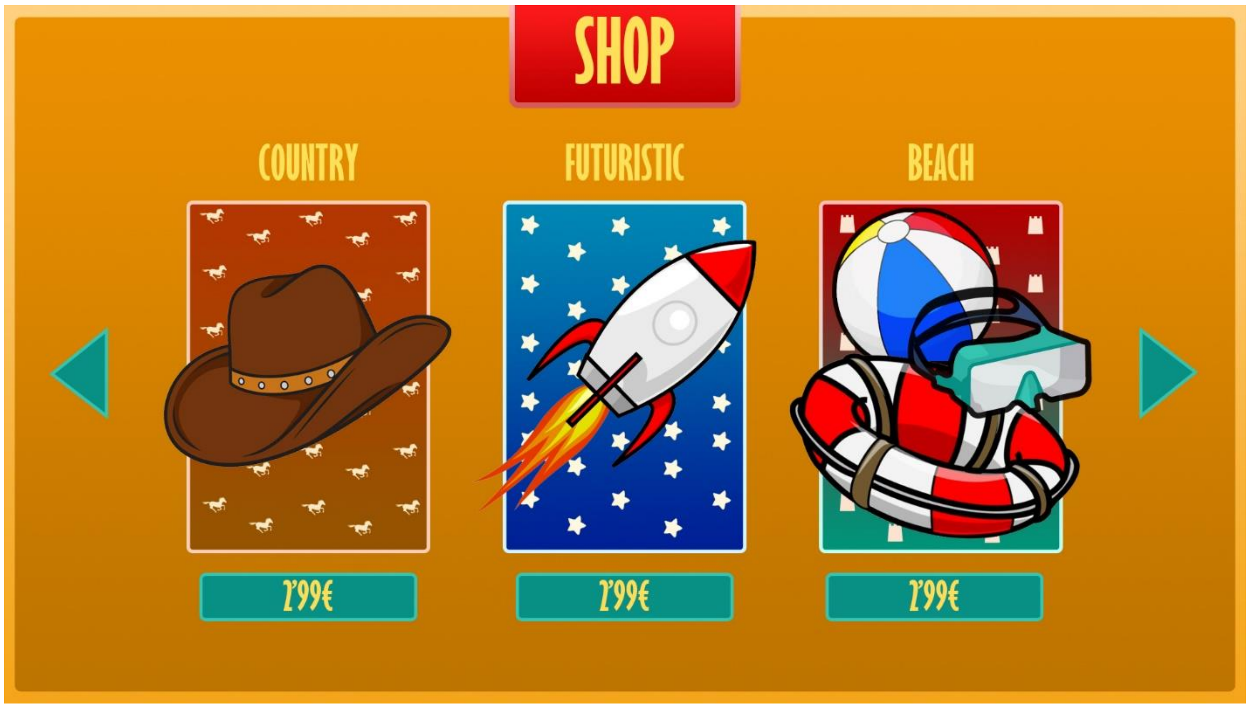


Fig 98 Shop screen in Beat Repeat, images extracted from [20]

With a main drawing on the card giving a clear idea of what the pack is about:

The idea is when the player presses the cards, the card selected will have an animation of the card turning and a description and a video-demo will be shown to let the player know all the characteristics of the pack. And when the player presses the price button, another tab will appear to let the player proceed with the payment of the pack selected.

The designs of the Scoreboard (see Fig 65), Top Songs (see Fig 66), Level Selector(see Fig 52) are simple designs, concept ideas with only their functionalities, like if they were just a colored wireframe:

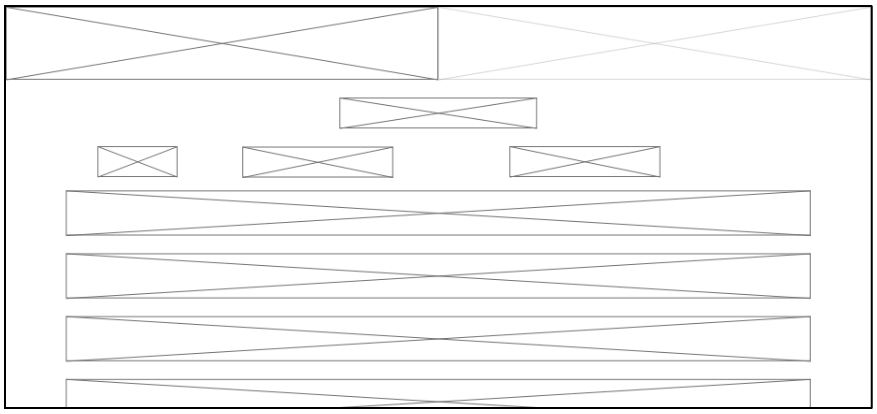


Fig 99 Scoreboard screen Wireframe in Beat Repeat

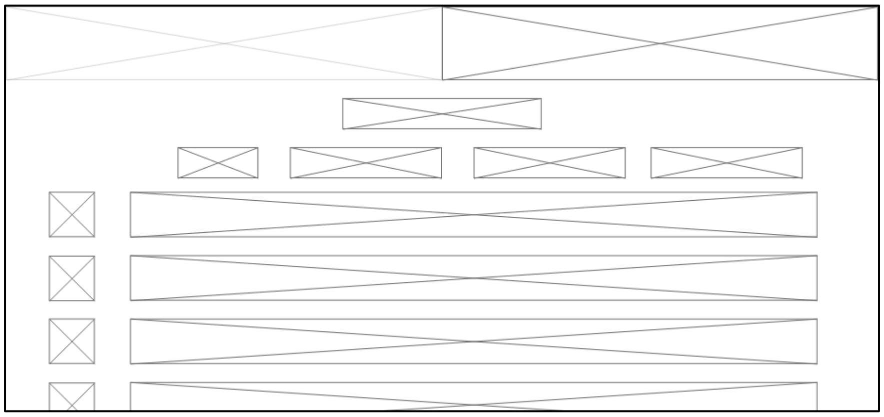


Fig 100 Top Songs screen Wireframe in Beat Repeat

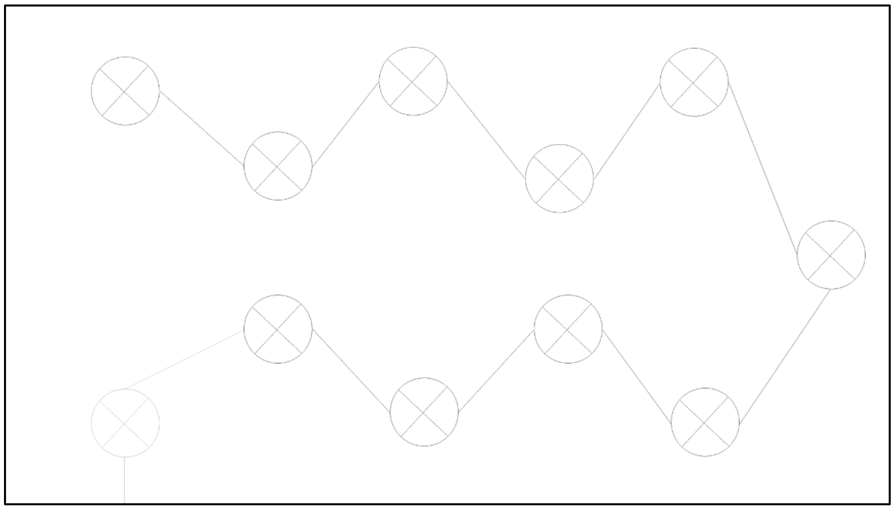
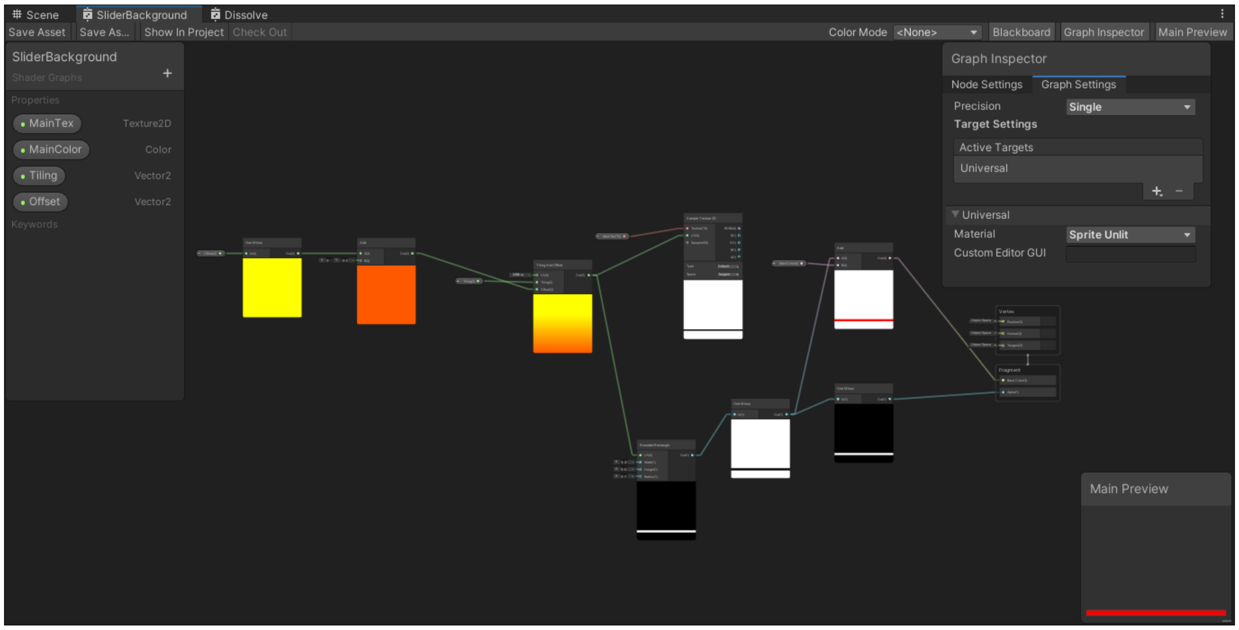


Fig 101 Level Selector of the Story Mode screen Wireframe in Beat Repeat

4.3. Shader's creation

In this section I am going to explain how I created the different shaders of the game.

The first shader we have is the one used for the lights in the background reacting to the music:



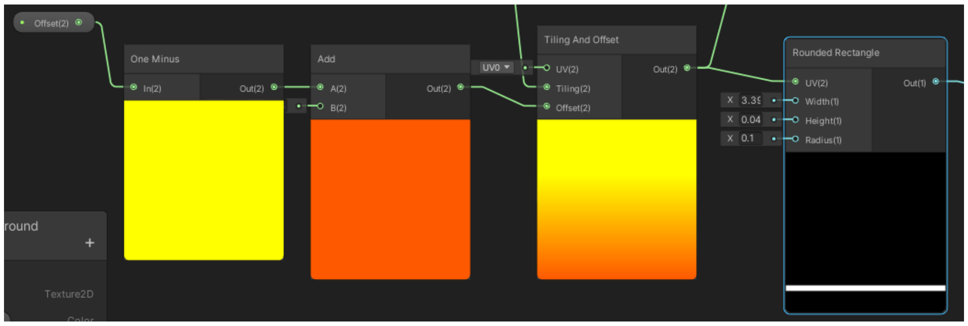
SliderBackground
Shader Graphs

Properties	
MainTex	Texture2D
MainColor	Color
Tiling	Vector2
Offset	Vector2

Graph Inspector
Node Settings Graph Settings
Precision: Single
Target Settings
Active Targets: Universal
Universal
Material: Sprite Unlit
Custom Editor GUI

Fig 102 General view of the Slider shader in shader graph for Beat Repeat

The main idea in this shader is having a rectangular shape that we can move up and down. That is why we need to take a Rounded Rectangle and make its height very small, then use the Node of Tiling and Offset using the UVs of our material and finally we will apply an offset to these UVs. For that, we create a variable called Offset that we can control from our code.



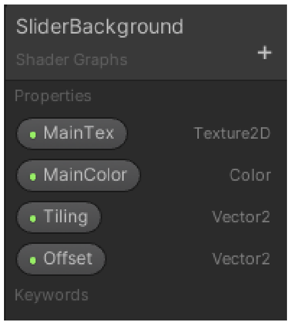


Fig 104 First part of the Slider shader in shader graph for Beat Repeat

Fig 103 Properties of the Slider shader in shader graph for Beat Repeat

Now, with this moving up and down rectangle we need the black and white node to apply to it the alpha for the limits of the rectangle on the texture, and then use another node to invert the color of this rectangle and be able to change the color of the rectangle.

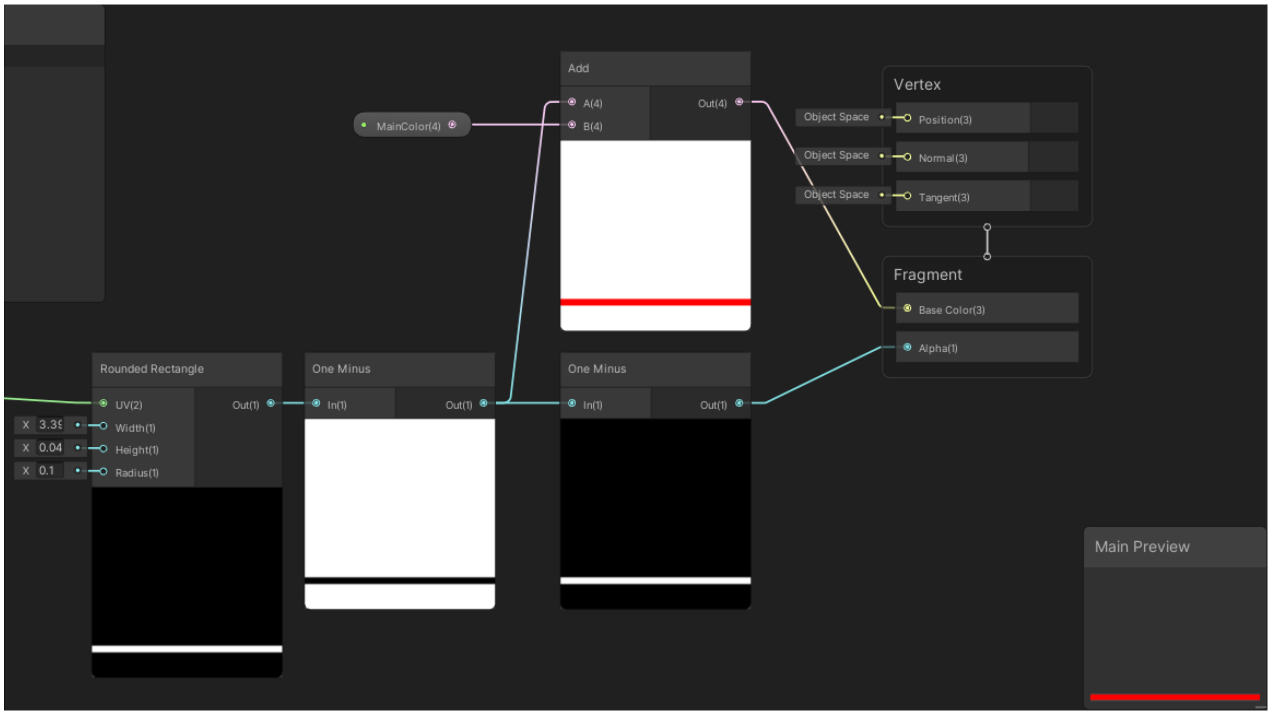


Fig 105 Second part of the Slider shader in shader graph for Beat Repeat

The only step left is applying the bloom effect. For that we need to go to the scene and create a Global Volume:

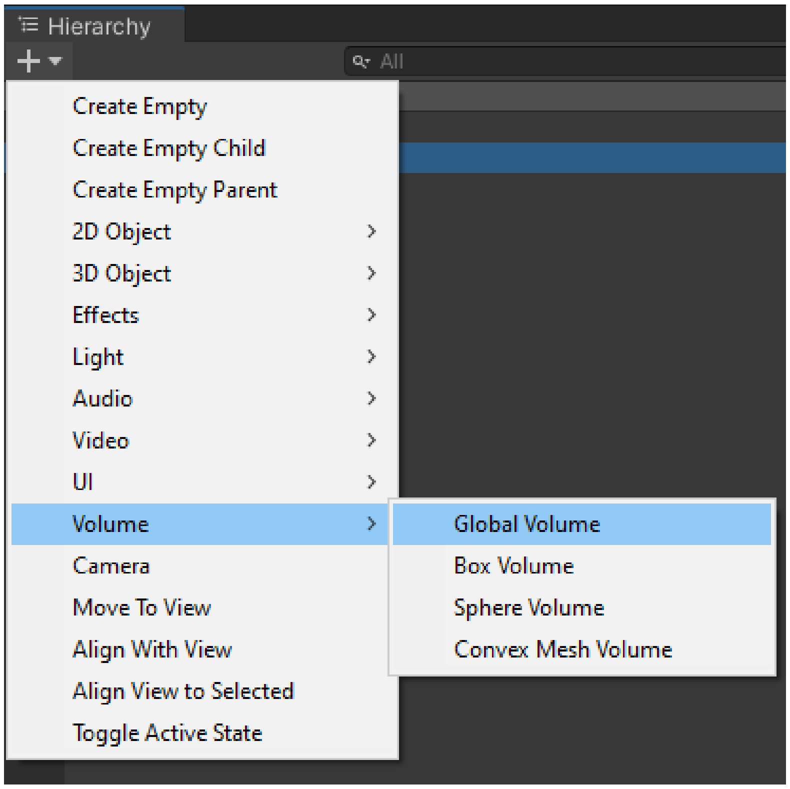
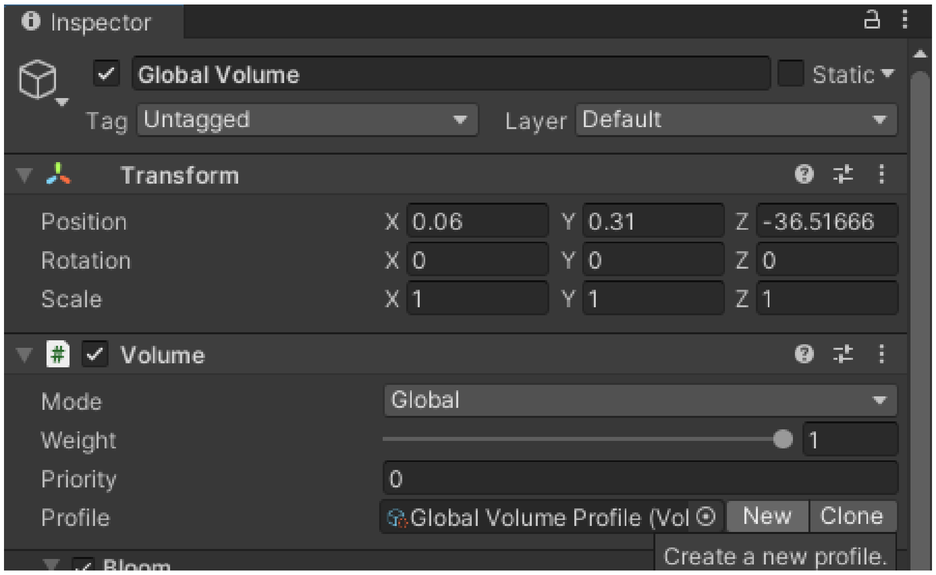


Fig 106 Creation of the Global Volume object in Unity

We create a new profile and add a new Override: Bloom.

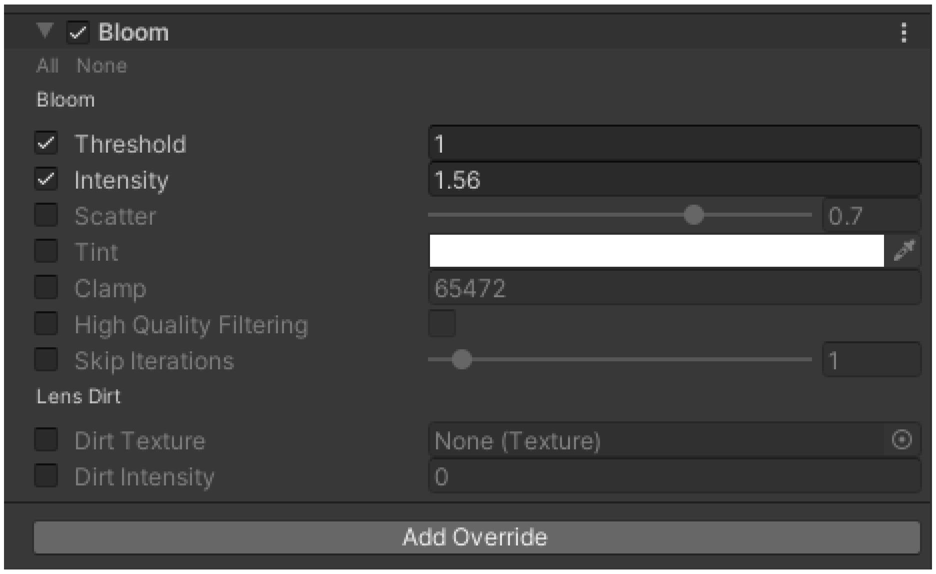


Global Volume
Tag Untagged
Layer Default

Position X 0.06 Y 0.31 Z -36.51666
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Mode Global
Weight 1
Priority 0
Profile Global Volume Profile (Vol

Fig 107 Properties of the Global Volume object in Unity



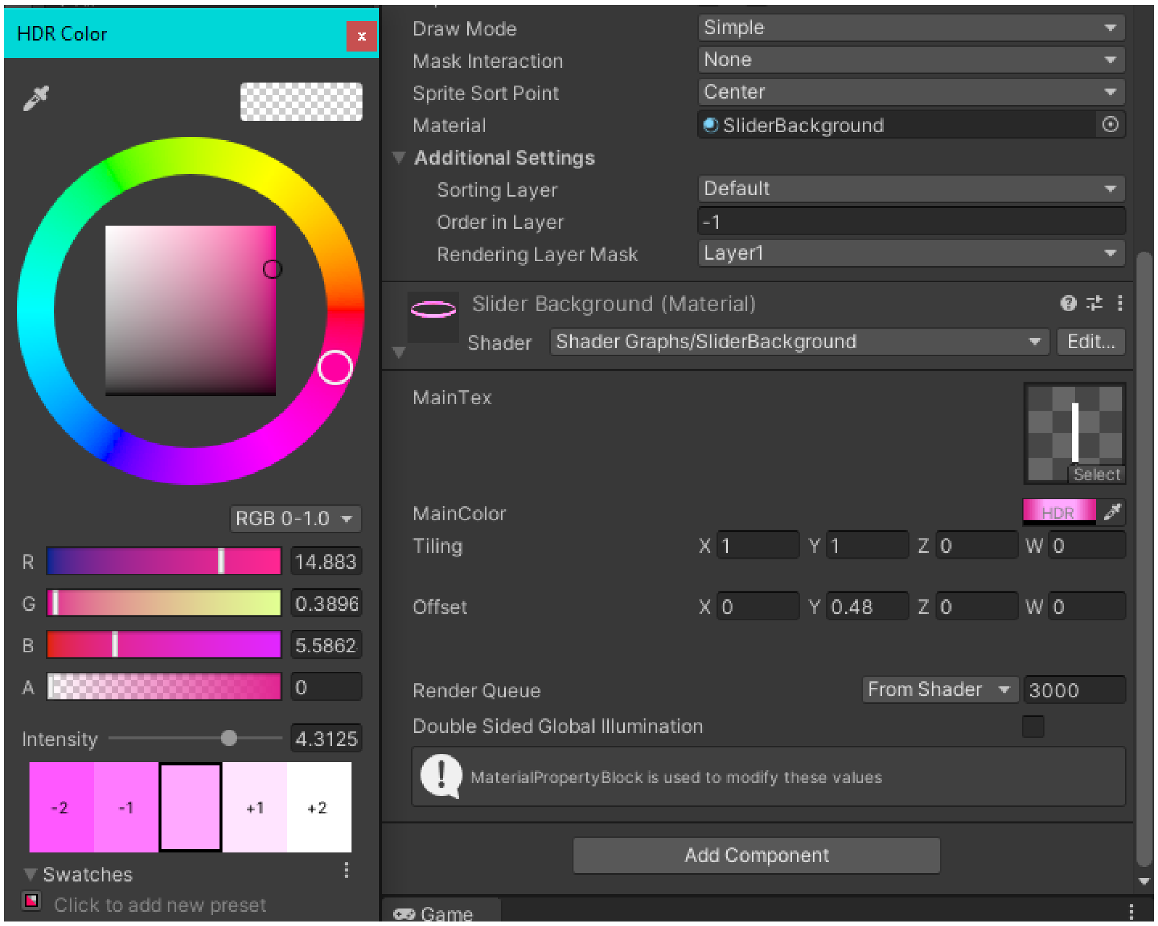
Bloom
✓ Threshold 1
✓ Intensity 1.56
Scatter 0.7
Tint
Clamp 65472
High Quality Filtering
Skip Iterations 1

Lens Dirt
Dirt Texture None (Texture)
Dirt Intensity 0

Fig 108 Bloom effect in Unity

Now, we can try with different numbers on the intensity to add more or less bloom, and on the threshold to make bloom only the objects with an intensity bigger than that number.

Before finishing, we need to add intensity to the material, for that we have to select the material and add intensity to the color:



R 14.883
G 0.3896
B 5.5862
A 0
Intensity 4.3125
-2 -1 +1 +2

Draw Mode: Simple
Mask Interaction: None
Sprite Sort Point: Center
Material: SliderBackground
Additional Settings
Sorting Layer: Default
Order in Layer: -1
Rendering Layer Mask: Layer1
Slider Background (Material)
Shader: Shader Graphs/SliderBackground
MainTex
MainColor: HDR
Tiling: X 1 Y 1 Z 0 W 0
Offset: X 0 Y 0.48 Z 0 W 0
Render Queue: From Shader 3000
Double Sided Global Illumination
MaterialPropertyBlock is used to modify these values

Fig 109 Color selection with intensity in Unity

For the moment this is the only shader in the game, but more shaders are going to be added in the future, like one where the robot has a shining effect when a melody is successfully added to him.

4.4. Music Production

In this section an explanation of different processes I have made to produce the music of the game is explained.

4.4.1. Inspiration

The idea of this project was to create a world where 5 robots could play their electronic music. As the idea of the whole game is having melodies playing repeatedly every 8 seconds, music that repeats but does not get boring was necessary. With these ideas, an amazing music band comes to mind, Daft Punk.

Daft Punk were a French electronic music duo. With six Grammy awards, Daft Punk were one of the most influential groups in the electronic music scene. Some of their most popular singles were “One More Time”, “Harder, Better, Faster, Stronger”, “Get Lucky” or “Starboy”.

The music style of a lot of their songs remains on repetitive loops, sometimes with sampled sounds, with the electronic music style as the main genre but with important doses of the dance and disco styles. The robotic vocals are also a very used resource and all this combined with melodic and uplifting drops, they get to be one of the best duos in the history of electronic music.

4.4.2. Organization and Process of creation

The main idea and characteristic this project has is the creation of melodies that fit each other without depending on the combination the player chooses. With this idea in mind, I had to create all the melodies and sounds, and whatever the combination of these were, it had to sound good. To verify that, I created an FL Studio project where all the patterns and melodies were one above the other, so I could select the combinations I wanted to listen to in a very easy and fast way.

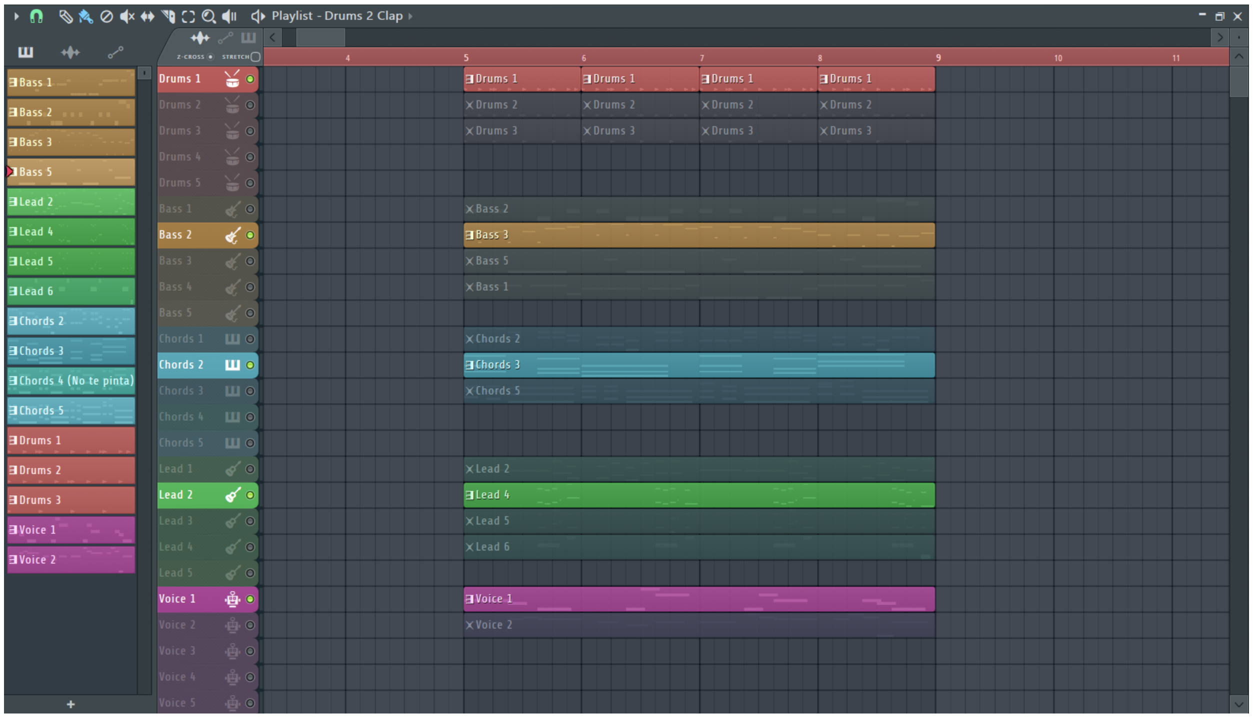


Fig 110 FL Studio Project with all the melodies for each instrument

4.4.3. Level Design

To create the idea and structure of the levels from the Story Mode I used FL Studio. The first level I created was Level 6. As this game mode is an original idea and there is nothing like this in the market, I could not use any reference to know how good or difficult a level was, so instead, I tried to imagine how a level with all the mechanics of the game would be. To do that, I created a level with all the instruments, with reverb and filter effects, with breaks where the melodies need to be quit, silenced and replaced for others. When I had all of that, I made sure that every part of the song was actually possible to play, and with some changes, I had the first level finished. The difficulty of the level starts low and grows as the player goes through the level, meaning the level has a difficulty progression and will be hard to get to the final with good punctuation.

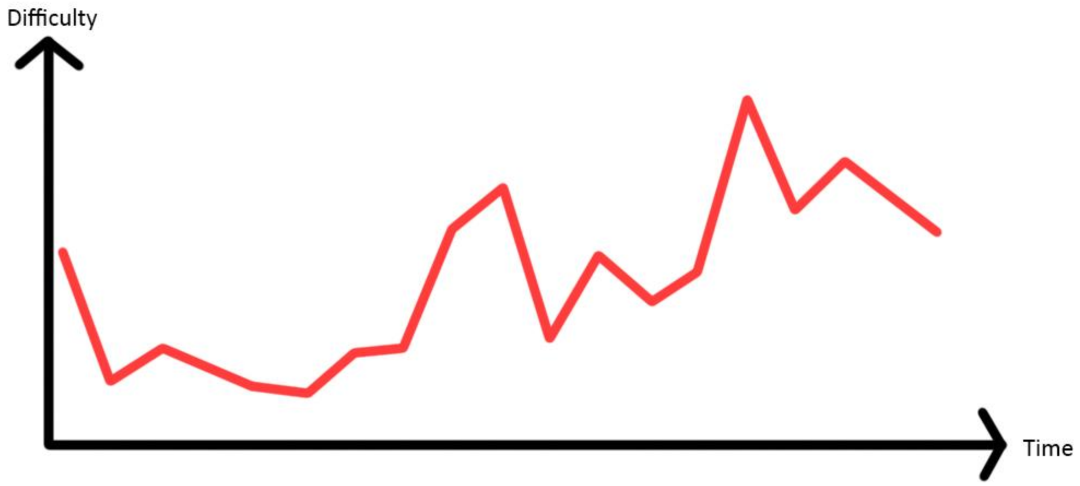


Fig 111 Difficulty progression in Level 6

This level was created in the same project as the project where all the melodies and sounds were created. To create the structure of this level I was also inspired by the original song Music Sounds Better With You by Stardust, which has a really similar style to Daft Punk and is the perfect example of a simple song without getting it boring.

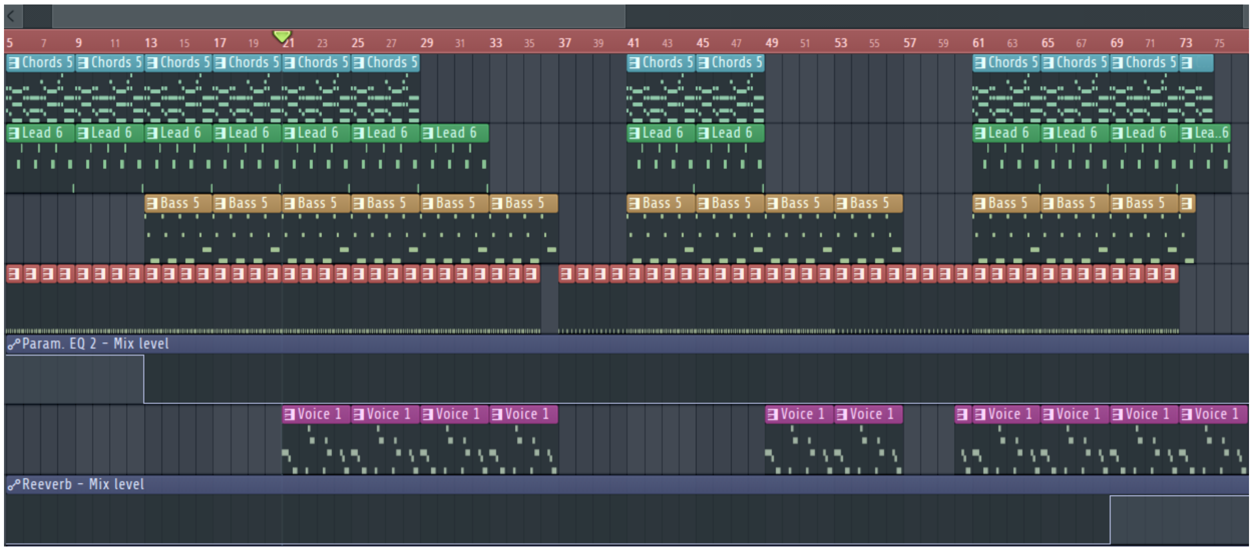


Fig 112 Level 6 structure in FL Studio for the Story Mode in Beat Repeat

After creating level 6, I had to create a level where the player was introduced to the mechanics of the game mode, so it had to be very easy. To do that I just simply used three instruments and the player will have to be putting and quitting melodies on them.

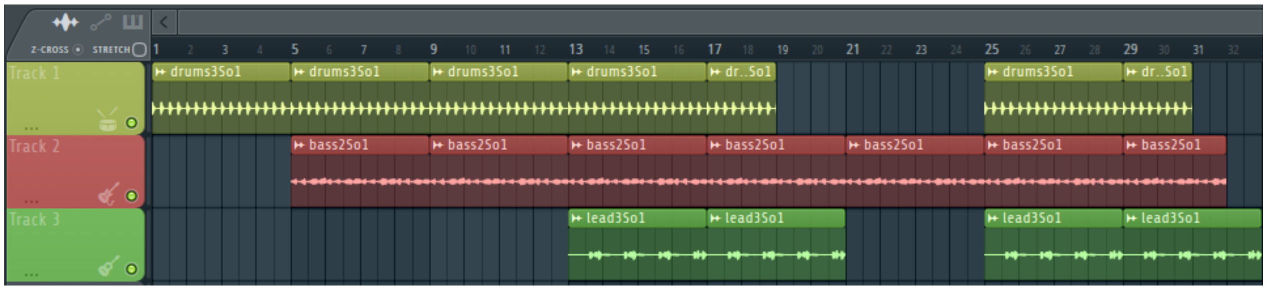


Fig 113 Level 1 structure in FL Studio for the Story Mode in Beat Repeat

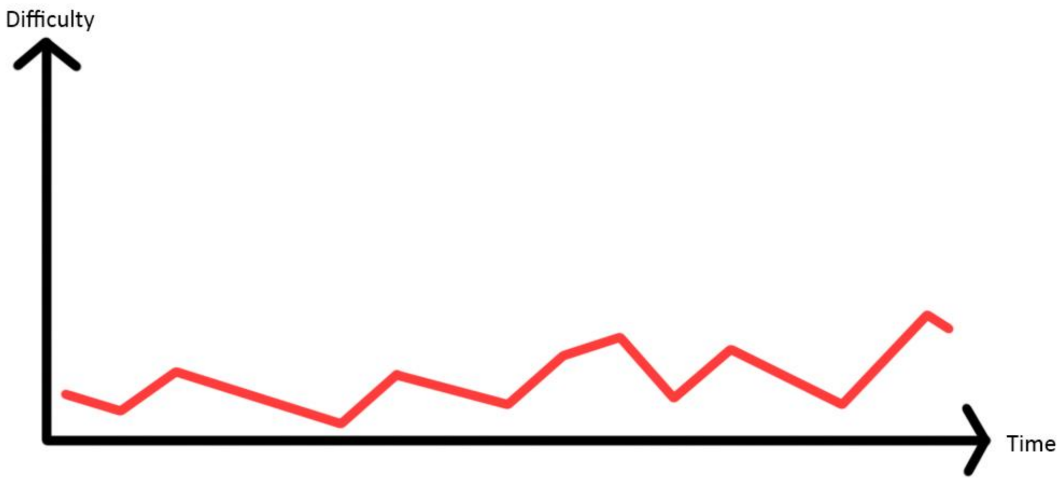


Fig 114 Difficulty progression in Level 1

And finally, the last level created for this prototype was Level 2, where the player now has 4 instruments to play with and new mechanics mentioned in the section *3.2. Story Mode*. When creating this level, I had in mind it had to be a little more difficult than 1 and had to introduce the rest of mechanics.

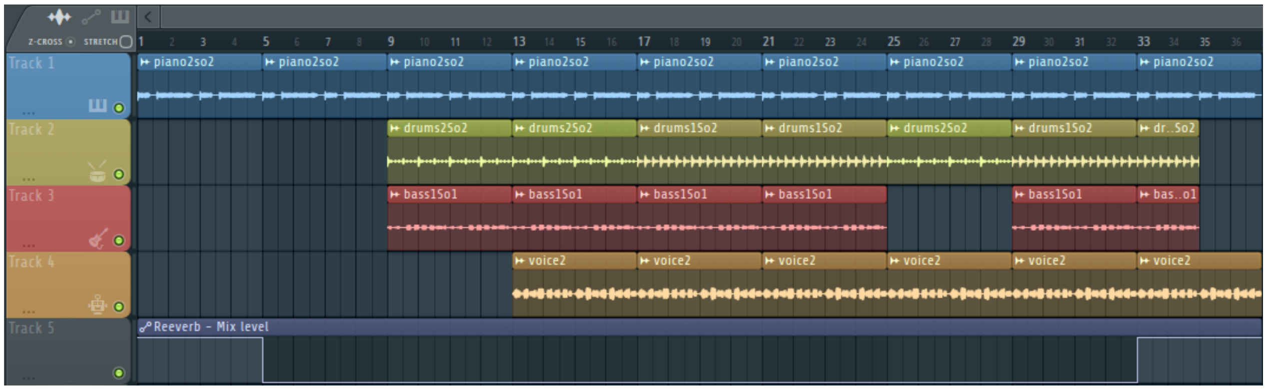


Fig 115 Level 2 structure in FL Studio for the Story Mode in Beat Repeat

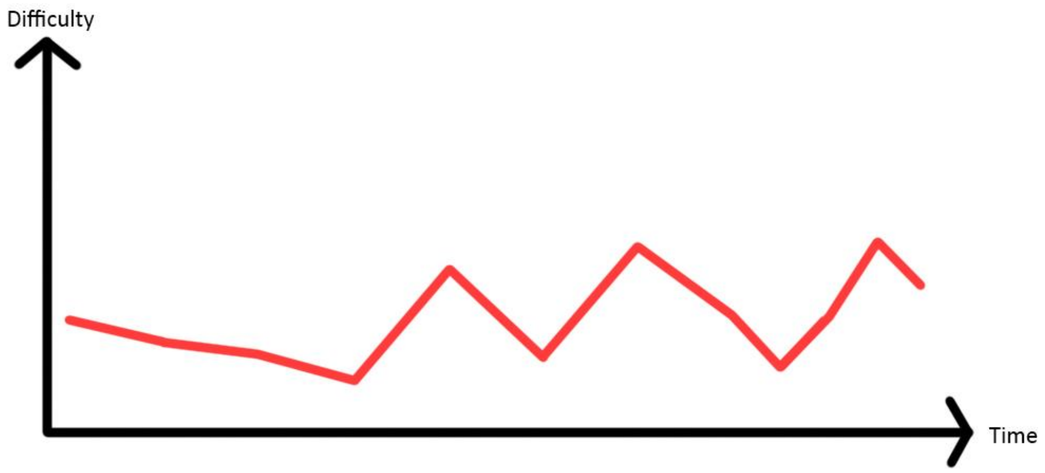


Fig 116 Difficulty progression in Level 2

4.4.4. Sound Design & Mixing

Before starting with this part of the project, I stopped to think about what aspects of my game I had to consider related to sound. Considering that this game is going to be for smartphones, probably most of the users that play the game are going to use the speaker of their phone to listen to the music. As the main function of this game is playing with music, I needed to make sure that every sound could be perfectly heard in any case. Although playing with headphones would be the recommended option, it is probable that most of the users do not use them for commodity purposes; for that reason, when I designed all the sounds for the game, I had to make sure that every sound had high frequencies on it, in fact, I had to pay special attention to basses and kick drums. These instruments are usually used to fill low frequencies, although they can also have high frequencies if the producer wants them. As I produced all these sounds with headphones and speakers with a low frequency range, in order to be able to listen to what the usual player would be hearing, I applied a filter in the master (which I can enable and disable whenever I want) to cut the low frequencies, allowing me to only listen to the high frequencies.

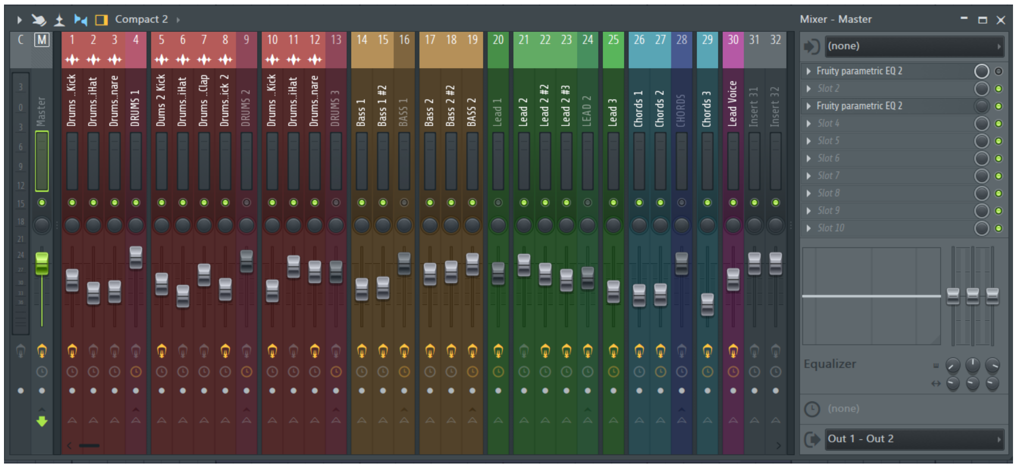


Fig 117 Mixer of the FL Studio project for Beat Repeat

Another important aspect is to give the necessary space in the mix for every sound, because in the game is easy to understand the melodies when there is only one or two instruments playing, but I had to make sure that when the five instruments were playing, each of them could be recognized separately by the user; which is another difficult task considering that most of the players will be using a phone speaker, which neither usually have low frequencies nor stereo system on it. To solve this problem, I had to do different processes to make sure the sound, which is one of the most important aspects of the game, was perfect.

- First, I needed every sound to be loud, the louder the sounds were, the easier they were to be heard on a phone speaker.
- Even if the frequency range is smaller than with a normal speaker, I had to make sure that every instrument had their space in the frequency mix, starting from lows to highs, the bass is the first one, the piano the second one, the singer the third one and the lead the fourth one. The drums are the only instrument I did not pay attention to in the frequency spectrum, due to the audio effect explained at the *2.1.4.2. Audio Mixer*, the Duck effect in Unity allows me to make all the volumes of the other sounds go lower when the kick plays, what is usually known as sidechain.
- To have a good mix, another tip I used was to listen to the instruments at a very low volume, this technique allows me to make sure that every instrument can be heard even if the music is super low.
- As mentioned, phone speakers do not usually have stereo systems, so another aspect I had to pay attention to, was checking the mix in mono.

With all these steps, now I just had to try and listen repeatedly to continue correcting little details.

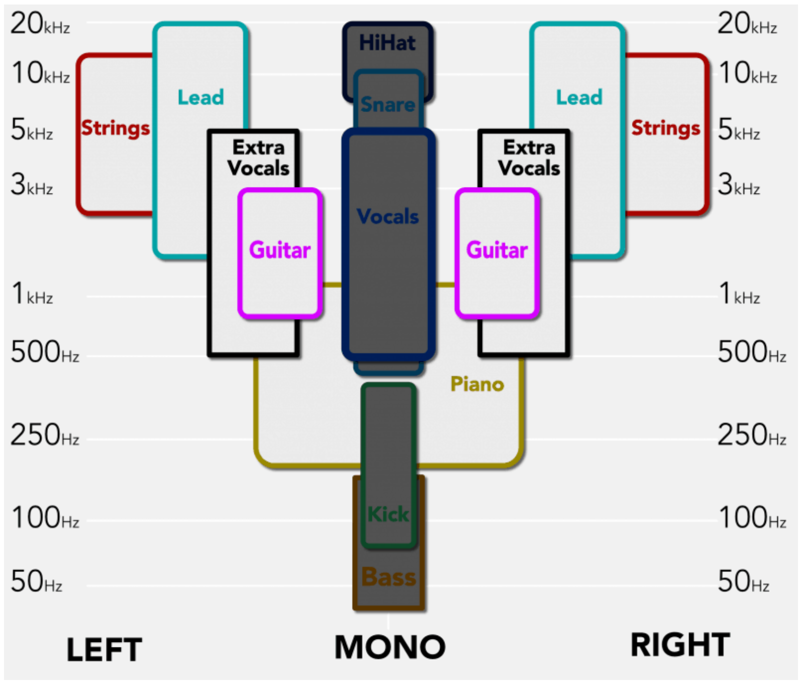


Fig 118 Mixing space reference for electronic music songs extracted from [19]

4.5. Character Design

At first, I wanted to create the characters of the game by myself, as at time to time I like to draw, and in some projects I have designed different characters, I thought it could be a great idea to draw my own character for my own game; but after doing some research I knew I had to hire a character designer if I wanted to get professional results and have an opportunity of success with the game in the market. Although it has not been possible to find someone to design my characters during the development of this prototype, I have written a description of what I am looking for:

“I'm working on a mobile game and I'm looking for a 2D artist for the character concepts (5 cartoon robots) who can animate 2D or do the key-poses. Each character would have an instrument (drums, electric bass, electric guitar, piano, and singer).

2D cartoon, eighties-style humanoid robots with daft punk as inspiration. With a defined style and each one with a predominant color: the yellow one is the drummer, the red one has the bass, the orange one is the singer, the green one is the guitarist and the blue one the pianist. The instruments will have electronic sounds, so none of these will have a classic look, they will be vintage/futuristic. Each robot will be different, with an aspect that makes it unique and distinguishable, with some human details. Mood board with the idea:”

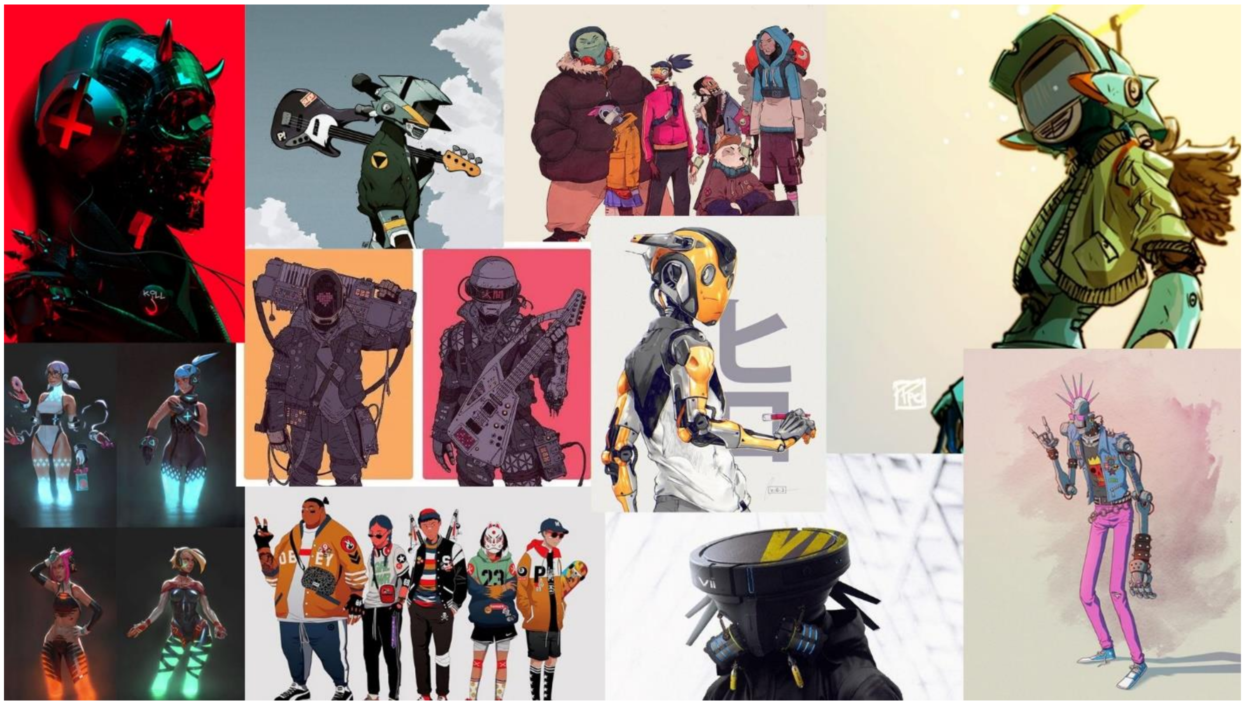


Fig 119 Moodboard of the character design for Beat Repeat, images extracted from [21]

4.6. Animations

The first animation I created was for the panel of sounds. The easy way was making the panel appear and disappear, but I thought that if I animated it, the game would have a more finished look and would create a better experience for the player.

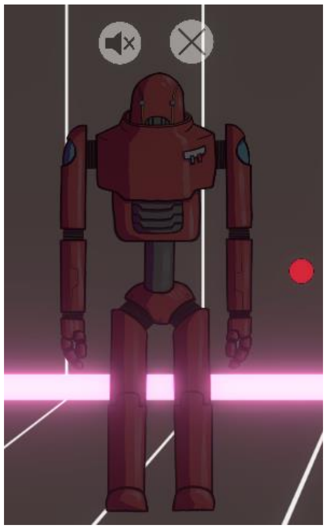


Fig 123 First part of the Sound selection animation

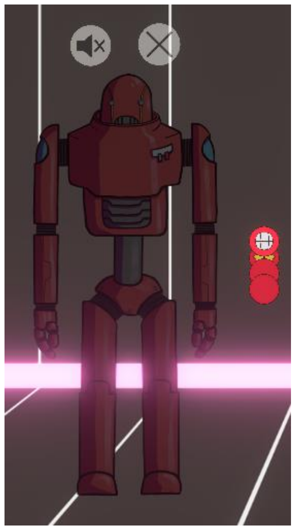


Fig 122 Second part of the Sound selection animation

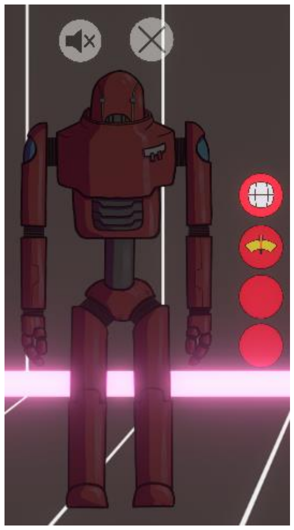


Fig 121 Third part of the Sound selection animation

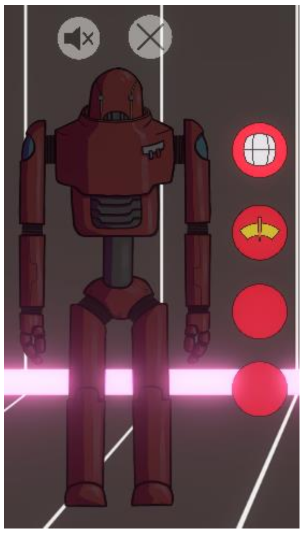


Fig 120 Fourth part of the Sound selection animation

In this animation the scale and the size of the 4 circles are being modified.

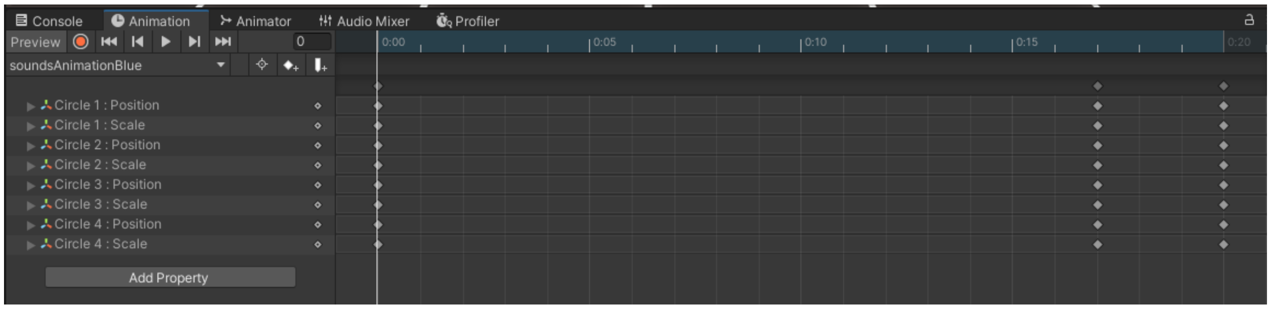


Fig 124 Animation tab for the Sound selection animation

When creating the Story Mode, I needed some kind of animation to let the player know when he must press the cross, quit and audio effect buttons, so I thought the best way to show that was by animating a cercle closing.

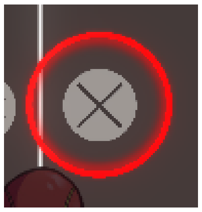


Fig 127 First part of the circle animation

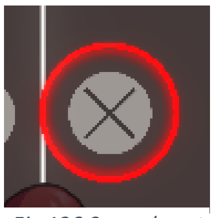


Fig 126 Second part of the circle animation

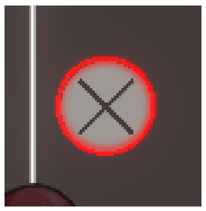
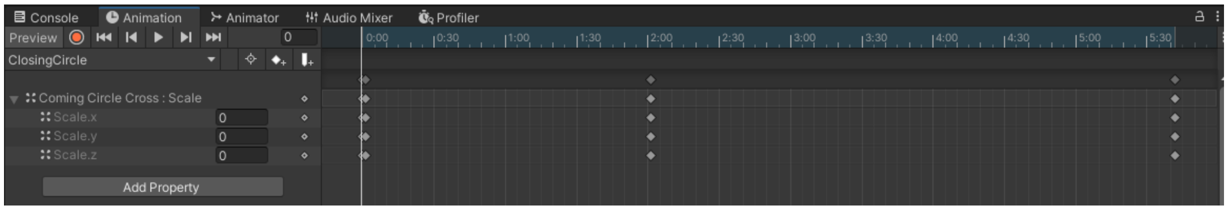


Fig 125 Third part of the circle animation

In this animation there is only modified the size of the image. Also, it requires more time than the previous one because we need to make the player see with antecedence that he will have to press the button in a few seconds.



Coming Circle Cross : Scale
Scale.x 0
Scale.y 0
Scale.z 0

Fig 128 Animation tab for the circle animation

And finally, one last animation I did after doing the playtesting is the following one. This animation is made to substitute the tutorial text on the first level, as explained in the *5.1.2. Results*.

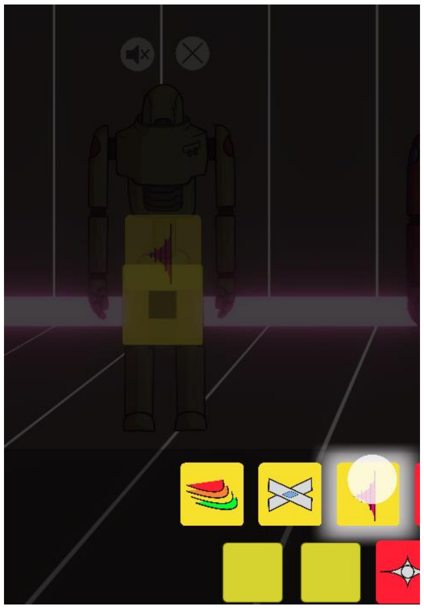


Fig 130 First part of the tutorial animation

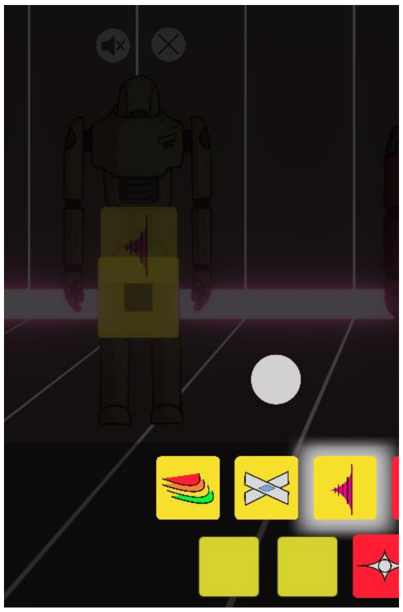


Fig 131 Second part of the tutorial animation

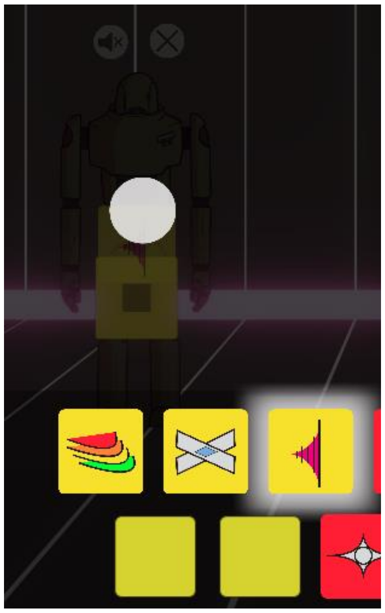
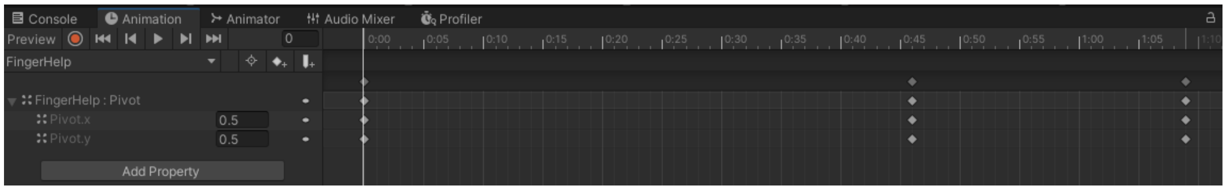


Fig 129 Third part of the tutorial animation

This animation just modifies the position of the white circle.



FingerHelp : Pivot
Pivot.x 0.5
Pivot.y 0.5

Fig 132 Animation tab for the tutorial animation

5. Game design analysis

In this section it is going to be analyzed the resulting prototype by playing the game and selecting the improvements that need to be added.

5.1. Playtesting

For this part, playtesting on different people has been done, with a concrete target of players, different aspects of the game have been tested to clarify if them are correct or if there is something that needs to be changed.

5.1.1. Process

Before starting with each playtester, different tasks have been asked to do to see if every aspect of the game is accessible and easy to understand.

The tasks are:

- Go to the shop and buy the Country pack.
- Go to the settings screen and unable the Audio Effects.
- Look for the Top Songs and listen to the top 1 song.
- Play the free mode.
- Look for the level 1 of the Story Mode and play it.

After finishing these tasks, each player is told to play and try the Free Mode. When every mechanic of the Free Mode has been discovered and tried on his own, now he is told to try the three different levels the game has, level 1, level 2 and level 6.

5.1.2. Results

As we can see in Table 2, all the tasks have been well understood and the players can easily find every section of the game.

	Joan M.	**Biel V.**	**Duarte D.**	**Oriol R.**	**Average**
Task 1	3.5s	4.3s	2.3s	4.3s	3.6s
Task 2	1.7s	2.6s	2.1s	2.5s	2.2s
Task 3	3.9s	2.9s	4.3s	3.0s	3.5s
Task 4	0.7s	1.1s	1.0s	0.9s	0.9s
Task 5	3.1s	2.4s	1.7s	1.3s	2.1s

Table 2 Comparison and average of the time taken to complete the tasks

Joan Mesalles Campos is the first playtester. He is 21 years old, from Andorra and living in Barcelona. He loves playing video games and developing them. He is studying for a degree in video game development at Enti School (UB). He has developed great games like Through Kippy's Eye, which has won different awards. Joan also is a music producer, known as Nexum, with thousands of plays on Spotify.

After completing the tasks and playing the Free Mode and the three levels of the story mode, these are the issues he has gone through the game:

- He is lost through the level of the Story Mode; he thinks the level does not have a finish point. He says a progress bar could be added to the top of the screen to know how much is left.
- When he fails in the Story Mode, he feels like he is going to lose the game but never loses. He considers a health bar could be added and when the player loses 3 hearts, he loses the game.
- He does not know when to click the effects, crosses and silence buttons in the Story Mode. The animation does not specify in any certain way when to click. He adds that a reference circle could be added to the background, so when the red circle goes through the background circle, the button should be clicked.
- He considers that there is too much text on the tutorial of the Story Mode, he thinks that some animation or other type of feedback could be added instead of the text, because he says that very few players would read that amount of text.

Biel Vicente Garcia is the second playtester. He is 21 years old, from Andorra and living in Barcelona. He loves playing videogames with his friends. He studied different courses related to audio and video technician, right now is studying Audiovisual Communication at Universitat Pompeu Fabra and at the same time he is a video editor for the ACB and an audio and light technician for live events. Biel also loves movies, TV series and TV shows; he listens to a big variety of music and has a strong knowledge of technical aspects related to audio and video.

After completing the tasks and playing the Free Mode and the three levels of the story mode, these are the issues he has gone through the game:

- In the Free Mode, he clicks repeatedly the high-lighted pitch button. He tries to do different things with it but without success. After some seconds he realizes he must click the other buttons to change something. He says it is a little bit confusing because he thought that the other buttons were locked, and he could only use the middle one because it was the only one highlighted. He does not know how he would solve the problem.
- He has the same issue of not knowing when to click the effects, crosses and silence buttons in the Story Mode. In this case Biel does not give any suggestions to solve this problem.
- When playing the Story Mode, he thinks the melodies have to be dropped just when the icon gets to the bottom, he did not realize he can drop them before. To solve this issue, instead of having the tutorial just when the melody gets to the bottom, the tutorial should appear just when the icon appears on screen so the player understands he can drop it before.

- He has trouble clicking the silence, cross and change-sound buttons. He considers making the buttons bigger would solve the issue.
- He asks if the player can lose the game in the Story Mode. He thinks that adding health to the player would be a good idea.
- He considers it a good idea to add different styles of music to the game.
- He says that the icon drawings make sense, feeling they are good related to each melody.
- Biel suggested a nice idea when asked for things he would improve. He said that, when the game has finished characters, he would change the appearance of the instruments when another sound is selected.
- One of the aspects that Biel loved the most was the music of the game.

Duarte Domingos Vieira is the third playtester. He is 22 years old, from Andorra and lives there. He plays a variety of games and loves gaming. Duarte studied a bachelor's in computer science at the University of Andorra and right now is working as junior developer in BDR Informàtica. He considers himself a "tryhard" player, meaning that he is always trying to optimize every aspect of a game and tries to go the hardest on competitive games and difficult levels. Duarte loves electronic music.

After completing the tasks and playing the Free Mode and the three levels of the story mode, these are the issues he has gone through the game:

- He has some trouble clicking the buttons of the cross and silence, he says it would be better if they were bigger.
- He has the same issue of not knowing when to click the effects, crosses and silence buttons in the Story Mode. Duarte also does not know how to solve this issue.
- Duarte did not realize he could change the sound of every instrument in the Free Mode. Once told, he knew perfectly how to work with it.
- When Duarte fails in some parts of the Story Mode levels, he feels a little bit lost because he does not know what he missed. He considers that giving more feedback when something is going wrong would solve it.
- Duarte did not see and missed some animations of the crosses and silence buttons. Making thicker the red cercle could help.
- When asked to say something more he would improve he says that adding some other mechanic like sliding or something like that would be great.
- He remarks he loves the Story Mode; he finds it original and says he would love to play it in the future.
- One of the aspects that he enjoyed the most was the music of the game.

Oriol Ruiz Sanchez is the fourth playtester. He is 22 years old, from Andorra and living in Vancouver, Canada. Oriol loves playing videogames and is very attentive to details. He studied

Animation and Special effects in La Salle Barcelona and right now is working for Icon on the Monsters at Work TV series. Oriol is a very creative person and always loves videogames with good art on them.

After completing the tasks and playing the Free Mode and the three levels of the story mode, these are the issues he has gone through the game:

- His first reaction to the effect buttons is clicking the central highlighted button, without success he realizes he has to press the other buttons.
- He has some trouble clicking the buttons of the cross and silence, and also thinks it would be better if they were bigger.
- When changing the sounds of some instruments he is a little bit lost. In some cases, he does not realize the change of the sound of the instrument. It is true that for this demonstration, some of the sounds are quite similar to others, so to improve it there should be more characteristic sounds for each of the instruments. Another improvement to this problem is the idea Biel gave, where each sound could change the aspect of the instrument, giving more feedback to the player when changing the sound and maybe making it easier to understand that the sound of the instrument is actually changing.
- In the Story Mode, he understood he could add the melody before the icon got to the bottom but did not understand when to drop it when there was another melody currently playing on the same instrument. Maybe a separated tutorial to this mechanic could be added.
- He has the same issue of not knowing when to click the effects, crosses and silence buttons in the Story Mode. Oriol says that changing the color of the circle that is closing to red, yellow and green like a traffic light could solve the problem.
- In the Story Mode he has some problems when he has to click the cross and silence buttons, he says that his hand occupies the entire screen and cannot see what is happening. He considers that moving those icons to the bottom of the robots would solve the issue.
- Oriol suggests adding more visual feedback when the player makes Perfects in the Story Mode. He also suggests having more and more feedback when the player does long combos.
- One of the aspects that Oriol most liked was the music of the game.

5.1.3. Selected Improvements

A progress bar to the levels of the Story Mode is going to be added. This progress bar is going be displayed at the top screen of every level and is going to tell how much time is left to finish the current level.

A new system with Life points is going to be implemented in the Story Mode levels. On each level, the player will be gifted three heart points and when he makes mistakes, depending on the dimension of the mistake, more or less hearts are going to be subtracted. If the player loses all the hearts, he will lose the game and will need to start from the beginning again.

As the tutorial has too much text, instead of text, an animation that tells the player to drag and drop the necessary melody will appear. This mechanic has already been implemented in this version of the game.

To improve the timing of the circle's animation, a circle in the background will be added to tell he needs to press the button when the red circle goes over the background circle. To help better with this problem, more visual feedback is going to be added to the game.

To tell the player he can drop melodies on the Story Mode before the icon gets to the bottom, the first tutorial is going to be shown just when the melody icon appears on the top of the screen instead of appearing when the melody icon is at the bottom, showing the player he can add the melodies before the loops start.

The cross and silence buttons need to be bigger to make it easier for the player to press them.

More feedback on the screen is going to be implemented to tell the player when some melodies or effects are not correctly used.

An information button is going to be added on the interface of the game to allow the player to learn all the diffeent mechanics in case he wants to.

As sometimes the circle animation is not seen for the players in the Story Mode, a thicker circle is going to be added.

On the level 2 of the Story Mode, some players asked for another tutorial when they have to drop a melody when there is already one, instead, this mechanic is going to be added more times on the level instead of just one. This will make the player understand the mechanic with the repetition of it.

A great idea suggested by one of the play testers is to change the skin of the instrument when the sound of it is changed. This will make the player understand better the mechanic of changing the sound.

As one of the play testers said, it can be difficult sometimes to see what is going on the screen when you have to press the cross and silence buttons due to the visual cancelation with the user hands, to solve this, these buttons are going to be placed on the bottom of the screen, between the robots and the panel of melodies.

6. Economic study

This section shows a brief analysis of the development cost of the project. The hours of reading, researching information, programming and practical development, hours of tutoring, final studies, tests and results, as well as the documentation are considered. First, an estimation of the hours dedicated to each task will be made, and then an approximation of the total cost of the project.

The hours dedicated to each of the tasks are the following ones:

- Reading, researching and learning, as well as watching tutorial videos have taken an important portion of the project, mostly at the beginning of the project to learn about Unity and new tools, as well as at the final part to finish writing the memory and contrast all the information with other resources. 50 hours have been spent for this part.
- Programming and practical development is where most of the hours of the project have been focused on, with the programming part, creating the logic of the game, creating mechanics by myself, fixing bugs and glitches as well as optimizing the code, together with all the aspects modified inside the IDE of Unity as for example the scenes, animations, shaders or different parameters. 210 hours have been spent to create the current prototype of the game.
- Art and designs of the game have been very important to acquire the great look the game has considering that the interface, designs of the buttons, wireframes and all the art ideas form part of this. 20 hours have been spent on this section.
- Music production is an essential part of this project, and without considering the time thinking about the music style and listening to references. 15 hours have been spent only producing the music and sounds.
- In order to do the tests realized on this project and analyze the results, 15 hours haven been spent on this part.
- Hours of tutoring, considering the meetings, preparing and reading mails, tutor search and tutor correction and mentoring. 45 hours have been spent from me and the tutors.
- Documenting the project has been a long and comforting process, writing down all the things learnt and applied during this project has made me see all the hard work I have done. 115 hours have been spent to create the memory of the project.

Task	**Time**
Researching	50 hours
Development of the app	210 hours
Creation of the art	20 hours
Music production	15 hours
Playtesting	15 hours
Mentoring	45 hours
Documentation	115 hours

Table 3 Hours dedicated to each task of the project

The sum of all these hours gives a total of 470 hours. Considering that the median salary range of a game developer that just started to work in the game industry is 1220 (7’03 EUR/hour) EUR per month in Spain (Game Developer Average Salary in Spain 2022, 2022), the cost of this project would be 3304’10 EUR.

If we look forward, adding the time costs of the future work mentioned on the *7.2 Future work*, an approximation of the hours spent to finish the game would be about 300. Considering this and the possible artist that is going to be hired to create the characters, animations and art of the game, which can cost, in total, about 1500 EUR, a total sum of 3304’10 EUR + 1500 EUR + (300 hours * 7’03 EUR/hour) = 6913’10 EUR is given.

7. Conclusions and future work

7.1 Conclusions

A long-term objective of this project is to create a video game that is fun to play, easy to use, helpful for teachers of music, motivating for children and commercially sound. For this purpose, I set out to review the existing music video games that have various features similar to my target game and went through the theoretical basis used for its creation. Having reviewed the state of the art and the theory to be applied, I started working on the practical part of it and designed a demo version that could be presented to a potential developer (a company producing video games) in order to illustrate my main ideas. On the programming level I have complied with most of the tasks I initially proposed to perform. Designing and programming additional mechanics, as the gameplay of the story mode, helped me improve professionally and come up with new, sometimes unexpected, ideas. Though the character designs and animations are still to be developed with the help of professionals, I am happy with the final UI & UX design, as well as the play testing part. I have also managed to use the knowledge acquired during the years of my study to apply concepts of audio processing to the game. Besides, I feel that my knowledge of music production has given me a chance to offer an original and personal viewpoint to the project.

I have gone through difficult problems during the development of this project, like the optimization of the game to have good performance on smartphones, the endless hours looking for bugs to correct every single detail, the development of code functions I have never done before or even the creation of new mechanics I have never seen on other games. All these processes have helped me to get a fantastic result for my own game and, if we compare it with other games like the ones mentioned in the *1.3. State of the Art*, I am really proud to say that this game is unique, not only for combining different aspects of those games, but also for creating something that does not exist in the actual game market. Even so, a lot of details can be polished, and a lot of new content needs to be added to the game, and maybe, even new mechanics can form part of the game, like the one commented on the 5*.1. Playtesting* of adding sliding mechanics, or one that has not been previously mentioned but it has been an idea during the development of the app, having midi patterns to let the player have more control over the robots in the game.

I consider that the playtesting process was one of the most important ones, it made me see how most of my ideas worked and how the player had a nice experience with the game, on the other hand, it cleared all the aspects that could be improved in order to get a great final product. One of the most satisfying comments the testers made was that they really enjoyed the music of the game, which is one of the aspects that I really focused on to be perfect.

All these points make me believe that a video game company could be interested in further developing my game, as it would allow a player to have a unique experience with the real-time reaction of different elements to the music. I also think that, properly developed, it could make an excellent tool for music teachers and a great game for children, especially when all the melodies and sounds of the robots have been created, as well as a really interesting game for players who would like to get high score points on the Story Mode.

7.2. Future work

The prototype of the game has been finished and I could not be happier with the final result, but now, it is time to keep working on the game and try to publish it. To do so, the first thing to do is start solving all the suggestions the play testers have told, keep pushing and keep testing all the versions. When these problems are solved, new melodies and sounds are going to be created, together with new levels for the Story Mode. While I keep programming and designing all this stuff, I will also be looking for an artist that creates my robots' design ideas and that can animate them. When the design of the characters is finished, I also want to add more elements that react to the music and a better-looking background. When all this stuff has been done, the demo of the game will be finished. Even so, the final game will not to be finished because, as the game is free to play, there needs to be some kind of income. To do so, I will need to create different packs which the player will need to pay if he wants to play them. For these packs, as all the logic and mechanics of the game will already be done, I will only need to create new melodies, new sounds, new stories, new levels and customized skins for the robots.

I know it is a big project, but I am really motivated and eager to work on it. I want to have a good project in my portfolio and maybe, if I have good luck, I can get to sell some copies.

8. Bibliography

Chung, SM., Wu, CT. (2017). Designing Music Games and Mobile Apps for Early Music Learning. In: Ma, M., Oikonomou, A. (eds) *Serious Games and Edutainment Applications* . Springer.

Comparison of game engines for serious games | Semantic Scholar. (n.d.). Semantic Scholar | AI-Powered Research Tool. Retrieved September 14, 2022, from https://www.semanticscholar.org/paper/Comparison-of-game-engines-for-serious-games-Pavkov-Frankovic/09ec5adb8c9a9b6e80c4eb768f84499a5961ff9e

Custom implementation of Unity FFT Window Function · GitHub. Gist; 262588213843476. Retrieved September 14, 2022, from https://gist.github.com/asus4/8de866791f19f2eb445149a3f0f45d41

FL Studio User Interface (GUI). (n.d.). The DAW Every Music Producer Loves | FL Studio. Retrieved September 14, 2022, from https://www.image-line.com/fl-studio-learning/fl-studio-online-manual/html/basics_interface.htm

How to choose FFT Window type - Unity Answers. (n.d.). The Best Place for Answers about Unity - Unity Answers. Retrieved September 14, 2022, from https://answers.unity.com/questions/1623827/how-to- choose-fft-window-type.html

Jesse. (2018, February 27). *Algorithmic Beat Mapping in Unity: Real-time Audio Analysis Using the Unity API | by Jesse | Giant Scam | Medium*. Medium; Giant Scam. https://medium.com/@jesse_87798/6e9595823ce4

Laroche, J. and Dolson, M. (Oct. 17-20, 1999). *New phase-vocoder techniques for pitch-shifting, harmonizing and other exotic effects*. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New Paltz, New York.

Peppler, Kylie, Michael Downton, Eric Lindsay, and Kenneth Hay. (2011). The Nirvana Effect: Tapping Video Games to Mediate Music Learning and Interest." *International Journal of Learning and Media*. *3.1*. 41-59.

Play, P. (2016, September 17). *Audio Visualization - Unity/C# Tutorial [Part 1 - FFT/Spectrum Theory]*. YouTube. https://www.youtube.com/watch?v=4Av788P9stk

Renzi, M., Vassos, S., Catarci, T., Kimani, S (January 15 - 19, 2015*). Touching Notes: A Gesture-Based Game for Teaching Music to Children*. TEI '15: Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction, Stanford, California, USA.

Stefyn, N. (2019, October 13). *What is Game Design and How to Become a Game Designer | CG Spectrum*. Online Digital Art & Animation School | CG Spectrum; CG Spectrum. Retrieved September 14, 2022, from https://www.cgspectrum.com/blog/what-is-game-design

Tobias, E. (2012). Let's play!: Learning music through video games and virtual worlds in *Oxford handbook of music education* (Vol. 2, pp.531-548). Oxford University Press.

Top Video Game Industry Statistics (2022) | Fortunly. (n.d.). Fortunly. Retrieved September 14, 2022, from https://fortunly.com/statistics/video-game-industry-statistics/#gref

What is Unity? Everything you need to know - Android Authority. (2021, March 20). Android Authority; https://www.facebook.com/androidauthority/. https://www.androidauthority.com/what-is-unity-1131558/

Annex I.

In this index, all the sources used for the citations of this document are exposed. These sources are made up of websites of videogames mentioned in this document, documentation used to talk about the learning of music skills with videogames, and websites used to compare salaries for game developers in Spain.

Arcade Archives VS. SUPER MARIO BROS. for Nintendo Switch - Nintendo Official Site. (n.d.). Nintendo Official Site: Consoles, Games, News, and More. Retrieved September 14, 2022, from https://www.nintendo.com/store/products/arcade-archives-vs-super-mario-bros-switch/

Contributors to Wikimedia projects. (2007, July 9). *Guitar Hero III: Legends of Rock - Wikipedia*. Wikipedia, the Free Encyclopedia; Wikimedia Foundation, Inc. https://en.wikipedia.org/wiki/Guitar_Hero_III:_Legends_of_Rock

Cuphead on Steam. (n.d.). Welcome to Steam. Retrieved September 14, 2022, from https://store.steampowered.com/app/268910/Cuphead/

DanceDanceRevolution A. (n.d.). DanceDanceRevolution A | E-Amusement. Retrieved September 14, 2022, from https://p.eagate.573.jp/game/ddr/ddra/p/index.html?REDIRECT=1&_ga=2.43629605.14573%205493.1663083332-1598175418.1663083332

Dautovic, G. (2022, March 8). *The Rise of the Virtual Empire: Video Game Industry Statistics*. Fortunly. Retrieved from https://fortunly.com/statistics/video-game-industry-statistics/#gref

Fall Guys | Play for Free Today. (n.d.). Fall Guys | Play for Free Today. Retrieved September 14, 2022, from https://www.fallguys.com/

Ler studio (n.d.). *Beat Saber - VR rhythm game*. Beat Saber - VR Rhythm Game. Retrieved September 14, 2022, from https://beatsaber.com/

ninjamuffin99. (n.d.). *GitHub - ninjamuffin99/Funkin: A rhythm game made with HaxeFlixel*. GitHub. Retrieved September 14, 2022, from https://github.com/ninjamuffin99/Funkin

Official minecraft page. (n.d.) Minecraft. Retrieved September 12, 2022, from https://www.minecraft.net/es-es

Pong Game. Pong Game. Retrieved September 14, 2022, from https://www.ponggame.org/

Rock Band Rivals | Harmonix Music Systems, Inc. (n.d.). Rock Band Rivals | Harmonix Music Systems, Inc. Retrieved September 14, 2022, from https://www.rockband4.com/

Rust — Explore, Build and Survive. (n.d.). Rust — Explore, Build and Survive. Retrieved September 14, 2022, from https://rust.facepunch.com/

So Far So Good. *Incredibox*. Incredibox. Retrieved September 14, 2022, from https://www.incredibox.com/

The Last of Us™ Part I. (2022, September 2). PlayStation; Sony Interactive Entertainment. https://www.playstation.com/en-us/games/the-last-of-us-part-i/

The Official Pokémon Website | Pokemon.com | Explore the World of Pokémon. (n.d.). Retrieved September 14, 2022, from https://www.pokemon.com/us/

VALORANT: Riot Games' competitive 5v5 character-based tactical shooter. (n.d.). VALORANT: Riot Games' Competitive 5v5 Character-Based Tactical Shooter. Retrieved September 14, 2022, from https://playvalorant.com/

Game Developer Average Salary in Spain 2022 – The Complete Guide. (n.d.). Salary Explorer | Salary and Cost of Living Comparison. Retrieved September 15, 2022, from http://www.salaryexplorer.com/salary-survey.php?loc=203&loctype=1&job=10578&jobtype=3

Annex II.

This index contains all the sources of the images used in this memory, as well as the ones used for artistic purposes of the project.

[0] Good, O. S. (2015, February 24). *Report: Activision bringing Guitar Hero back to consoles, maybe even this year - Polygon*. Polygon; Polygon. *Guitar Hero III: Legends of Rock game screenshot* [Fig.2]. https://cdn.vox-cdn.com/thumbor/7Qsaazn9JZAXAYIDia-N-QfiNVA=/100x0:1180x720/920x613/filters:focal(100x0:1180x720):format(webp)/cdn.vox-cdn.com/uploads/chorus_image/image/45761464/Guitar-GH3-hammeron.0.jpg

[1] Wikimedia commons (2017). Guitar hero controller [Fig.3]. https://upload.wikimedia.org/wikipedia/commons/8/80/Guitar_Hero_series_controllers.jpg

[2] Ars Technica (2015). Rock Band game screenshot [Fig. 4]. https://cdn.arstechnica.net/wp-content/uploads/2015/10/RockBand4_HUD_02-980x551.jpg

[3] Game Fabrique (n.d.) DDR game screenshot [Fig. 6]. https://gamefabrique.com/screenshots/ps2/dance-dance-revolution-extreme-02.jpg

[4] Variety (n.d). People playing DDR on arcade console [Fig. 5]. https://variety.com/wp-content/uploads/2018/10/dance-dance-revolution- game.jpg?w=681&h=383&crop=1

[5] Toptal designers (n.d) Types of User Interface in a game [Fig. 42]. https://bs-uploads.toptal.io/blackfish-uploads/uploaded_file/file/52908/image-1568632674228-453d06dbb39af9fe6c5ed9ed2777fad0.jpg

[6] Codechangers (2020). Example of Non-Diegetic Interface in Super Marios Bros [Fig. 43]. https://codechangers.com/NEW/static/img/blog/cc-super-mario-wii-u-min.jpg

[7] joansticar (2020). Example of Diegetic Interface in Dead Space [Fig. 44]. imgvideogames. https://i.pinimg.com/originals/fb/01/0f/fb010fc94e55683ac1c168a6f00793aa.jpg

[8] Tan Hao Ming, D. (2021). Example of Meta Interface in Call of Duty [Fig. 46]. Medium. https://miro.medium.com/max/636/1*GXyjSUij8jwqu4oN0Baiew.png

[9] Physik (n.d.). Multiplication of a sinusoidal signal with a Hamming window [Fig. 34]. https://www.physik.uzh.ch/local/teaching/SPI301/LV-2015-Help/common/GUID-12EE66D8-D6BA-4AAB-9373-1A0E5A6347A6-help-web.png

[10] Shi, Z. (2008). Peak detection in frequency domain [Fig. 36]. https://www.researchgate.net/profile/Zhenwei-Shi-3/publication/3458051/figure/fig1/AS:340442810994695@1458179299552/Peak-detection-algorithm-illustration-The-thin-curve-is-the-spectrum-the-bold-curve-is.png

[11] Nano photon (n.d). Region shifting [Fig. 37]. https://www.nanophoton.net/wp-content/uploads/2020/10/20_img03-300x231-1.jpg ´

[12] Socoró, J.C. and Alías, F. (n.d.).Phase adjustment [Fig. 38].

[13] Juillerat, N., Arisona, S., Schubiger-Banz, S. (2007). Process example [Fig. 39]. Semantic Scholar. https://www.semanticscholar.org/paper/Real-Time%2C-low-Latency-audio-Processing-in- Java-Juillerat-Arisona/3d9827309e9bf2b8d66e2f3d087c186d51de8169

[14] Ramaseshan, A. (n.d.) Mel and Frequency relation [Fig. 40]. Researchgate. https://www.researchgate.net/publication/259479391/figure/fig3/AS:667809986125842@153 6229716748/The-mel-scale-used-to-map-the-linear-frequency-scale-to-a-logarithmic-one.png

[15] The Wingless (2021). Example of User Interface on smartphone [Fig. 41]. https://thewingless.com/index.php/2021/11/12/how-to-find-your-first-video-game-ui-ux-design-job-opportunity/

[16] FL Studio (n.d). Fruity Loops Studio Logotype [Fig. 47]. https://bassgorilla.com/wp-content/uploads/2015/08/FL-Studio-logo-vector.png

[17] Wikimedia commons (n.d.). Unity logotype [Fig. 9]. https://upload.wikimedia.org/wikipedia/commons/1/19/Unity_Technologies_logo.svg

[18] Ingeniería musical (n.d). Fruity Loops Studio Interface [Fig. 48]. https://www.ingenieriamusical.net/wp-content/uploads/2018/03/interface-prin.jpg

[19] Musician on a mission (n.d). Mixing space reference for electronic music songs [Fig. 117].

https://musicianonamission.com/wp-content/uploads/2018/12/6.-Wide-elements-800x681.png

[20] Freepik; i.ytimg; pngkey and pngitem (n.d).Shop screen Wireframe [fig. 96]. https://img.freepik.com/premium-vector/snorkeling-mask-icon-scuba- diving-equipment-modern-design-underwater-glasses-with-rubber-holder-swimming-sea- ocean-pool-isolated-white-background-cartoon-vector-illustration_87771-14959.jpg?w=2000; https://i.ytimg.com/vi/ONjjWiVgxNM/maxresdefault.jpg; https://www.pngkey.com/png/detail/48-481318_28-collection-of-spaceship- drawing-png-cartoon-rocket.png and https://png.pngitem.com/pimgs/s/200-2007287_float-clipart-hd-png-download.png

[21] Murcia, O. (2022). Moodboard of the character design for Beat Repeat [Fig. 118]. Pinterest. https://www.pinterest.es/oriolmurciaaa/tfg/