Bachelor’s Thesis

Embedded Systems

2019

Daniel Beato de la Torre

FUNCTIONALITY IMPROVEMENT OF AUDIO GAME ENGINE IMAGE AND PROOF OF CONCEPT OF MUSIC MAZE

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR’S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Embedded Software

June 2019 | 42 total pages, 7 appendix pages

Daniel Beato de la Torre

FUNCTIONALITY IMPROVEMENT OF AUDIO GAME ENGINE IMAGE AND PROOF OF CONCEPT OF MUSIC MAZE

myTrueSound is a Turku-based startup developing phone audio games for the blind and visually impaired. Additionally, the company is developing its own game development tool, based on game engine Unity.

The aim of the thesis was to solve two problems the company had: firstly, the need to keep developing company’s own game development tool, and secondly, the need to create a new appealing and entertaining game.

The game problem was solved with MusicMaze, an infinite runner procedurally generated through a random algorithm, with 3D immersive audio, and based on a fantasy animated world. The player must run and avoid all the traps, enemies, and monsters in the Maze by making swipes and taps on the screen.

The game development tool improvements carried out in the thesis are: procedural generation structure, random selection algorithm, feedback sound to the player action, easier game development, multiple correct gesture at the same time and fully accessible game.

All the needed game development tool infrastructure has been carried out and MusicMaze demo is ready to be tested in beta testing process.

KEYWORDS:

Game development, Unity, accessibility, procedural generation.

CONTENTS

LIST OF ABBREVIATIONS (OR) SYMBOLS 5

1 BACKGROUND AND INTRODUCTION 6

2 THE NEW GAME, MUSIC MAZE 8

2.1 Working process and methods 8

2.2 Game concept 12

2.3 Game Design Document 14

2.4 Audio Assets 15

2.5 Menus flowchart 16

2.6 Accessibility 19

3 IMAGE 20

3.1 IMAGE background 20

3.1.1 Introduction 20

3.1.2 IMAGE elements 20

3.1.3 Unity Animator Controller 22

3.2 FMOD integration. Previous IMAGE version 23

3.3 IMAGE improvements 24

3.3.1 MultipleChunk class 25

3.3.2 RandomChunkState and RandomChunkStateEditor classes 26

3.3.3 ChunkXOR class 27

3.3.4 ChunkXORState class 27

3.3.5 LoopAnimator and LoopAnimatorEditor classes 28

3.3.6 Sound feedback to gestures 30

3.3.7 Score mechanism 31

4 CONCLUSION 33

REFERENCES 35

APPENDICES

Appendix 1. MusicMaze original story 36

Appendix 2. MusicMaze split story 38

Appendix 3. GDD section 39

FIGURES

Figure 1. Example slide of game presentation. Level design idea. 9

Figure 2. Testing Android smartphone screenshot of Thesis.apk. 10

Figure 3. Game Design Document version control table. 12

Figure 4. MusicMaze audio asset table example. 16

Figure 5. Menu visibility setting in Unity interface. 16

Figure 6. MusicMaze menus hierarchy under Unity scene. 16

Figure 7. Different hover features in MusicMaze main menu buttons. 17

Figure 8. MusicMaze menus flowchart. 18

Figure 9. Diagram in animator example. 22

Figure 10. Behaviour example of a state with a game chunk. 23

Figure 11. Animator transitions Done, Time Out and Incorrect. 23

Figure 12. Audio action object inspector in Unity interface. 24

Figure 13. Screenshot of procedural algorithm method getRandomChunk(). 25

Figure 14. Process of MultipleChunk creation menu. 26

Figure 15. Diagram state inspector with MultipleGameChunk as a behaviour. 26

Figure 16. XOR truth table (Csetutor 2019). 27

Figure 17. OnStateEnter function. ChunkXOR gestures initialization. 28

Figure 18. LoopAnimator class screenshot. 29

Figure 19. Example of game demo diagram in Unity animator. 29

Figure 20. Screenshot of gesture swipe cast attempt. 30

Figure 21. Game state diagram with sound feedback in Unity animator. 31

Figure 22. Jump feedback state inspector in Unity interface. 32

LIST OF ABBREVIATIONS (OR) SYMBOLS

B&VI	Blind and Visually Impaired
CEO	Chief Executive Officer
CPU	Central Processing Unit
GDD	Game Design Document
IMAGE	Immersive Audio Game Engine
UAP	User Interface Accessibility Plugin
UI	User Interface

1 BACKGROUND AND INTRODUCTION

There are worldwide over 250 million people affected by a severe degree on visual impairment. Blind and visually impaired people (B&VI) encounter limited access to highly visualized games and social media platforms, even though the B&VI community is well versed in the use of technology. Accessible technology helps to make their life slightly easier. Related to mobile technology and gaming, it is estimated that 20 million B&VI people own and use daily smart devices, 10 million play games, and 2 million are active players. Unfortunately, very few games are accessible (i.e., can be used) by blind people, and, therefore, the demand is much higher than the offer. In proportion to population, there are 46 times less games available or accessible for the blind people than for people who can see. Existing games are often rated as boring, repetitive, and of poor quality. (myTrueSound Oy 2018).

myTrueSound is a Turku based start-up developing a new generation of audio games specially design for the B&VI. Audio games are defined as digital games where the main playing mechanics are based in audio. In order to succeed with its business goals and growth strategy, the company is developing its own game development tool, IMAGE, and at the same time it creates new games to open the market and grow its brand among the community.

The author of this Thesis had been working for myTrueSound from 11.12.2018 until 03.06.2019. The work, all in all, aimed to solve two of the greater challenges the company was facing at the time the collaboration started; 1 – the need to keep developing company's own game development tool IMAGE, and 2 – the need to develop a new appealing and entertaining game able to reach high selling digits.

This Thesis presents and details the methods followed to solve both problems. First, it describes all the features, bug-fixing actions, and new functionalities which were implemented by the author in order to improve the above mentioned development tool. There are not any other similar studies about this game development tool since it is an individual creation started in 2018. However, it is relatively easier to find previous studies about game development on the Internet or Theseus. Theseus is the database of theses and publications from Finnish Universities of Applied Sciences, where it is possible to read and utilize entire thesis and publications for a research or a development work. (Theseus 2019)

Throughout the thesis, the user or player, and the author or developer of this thesis will be referred as a man. Therefore, all pronouns will be masculine.

2 THE NEW GAME, MUSIC MAZE

This chapter describes the new developed game, but it also includes a description of the process and the methods followed along the creation process.

2.1 Working process and methods

The author (in this paper also referred as 'the developer') was the person behind the new game idea, and the main responsible person for the game concept, its design, and the implementation of the first demo or prototype. During the whole creative process, the developer had to select the most appropriate tools and methods. Nevertheless, the work was supervised by the commissioner, typically along the meetings which were performed every two weeks. The meetings were typically held with the commissioner and the sound designer. Later on, the lead programmer of the company joined the meetings as well the new art designer.

Taking all that earlier information into account, it can be fathomed that the thesis has followed an agile methodology based on iterative and incremental approach. In every meeting, the group discusses the result of developer's previous sprint. It is important to understand that although everybody could give his opinion in the company about a finished sprint, it was actually the CEO or the developer who would decide what was going to be next sprint about. Besides, it was mainly the developer who, knowing the requested task, thought of how to really implement, develop, or create it.

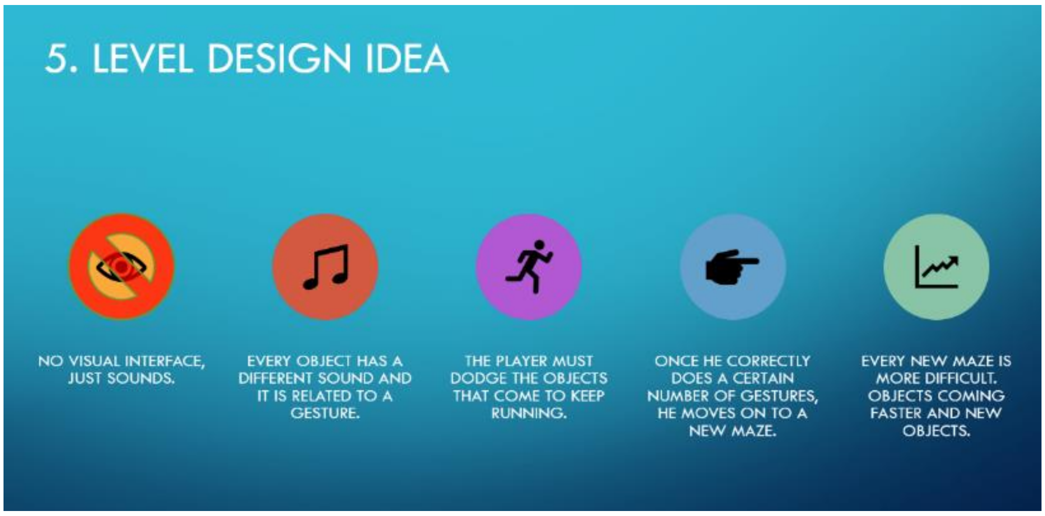
The working process is summarized as follows:

- Presentation of game concept
- Formulation of the game design document
- Implementation of new functionalities into the game development tool
- Implementation of first demo

All in all, the working period was divided in 16 sprints, each lasting two weeks

Sprint 1 – Game concept

The work started with the signing of the Thesis commission agreement. The work focused on the design of a good sellable game idea. The developer, based on his playing background and interests, decided that the new game should be an arcade game of the infinite runner type. The main parameters were set. The game idea was presented to the commissioner and to the new audio designer. This presentation, see Figure 1, was important because it was needed to convince the new audio designer to participate in the implementation of the game.



5. LEVEL DESIGN IDEA

NO VISUAL INTERFACE, JUST SOUNDS.

EVERY OBJECT HAS A DIFFERENT SOUND AND IT IS RELATED TO A GESTURE.

THE PLAYER MUST DODGE THE OBJECTS THAT COME TO KEEP RUNNING.

ONCE HE CORRECTLY DOES A CERTAIN NUMBER OF GESTURES, HE MOVES ON TO A NEW MAZE.

EVERY NEW MAZE IS MORE DIFFICULT. OBJECTS COMING FASTER AND NEW OBJECTS.

Figure 1. Example slide of game presentation. Level design idea.

The presentation was a success and the audio designer, Janne Kantola, joined the project.

Sprint 2 – Audio assets

Audio designer needed a reference to know what he was going to work with. That was developer’s task during the second sprint.

Sprint 3 – Story

The developer wrote a background story for the game where the hero, monsters and different characters were introduced. To read the whole story, check Appendix 1.

Sprint 4 - Tutorial

The developer wrote an interactive tutorial where the user learns by playing. It contained retries in all the steps with extra indications to keep the player always motivated.

Sprint 5. MultipleChunk feature developed.

Sprint 6. RandomChunkState + Editor feature developed.

Sprint 7 – First testing and demo

When the audio designer had some special effects, the developer did a small demo with them and few gestures such as swipe up or down. Gestures are the actions the player must make to play the game. In order to show it better to the team, the developer compiled the game for Android.

To build the app, the developer used Java Development Kit version 11.0.2. In Figure 2, it is also the first game of the company, GoldGun.

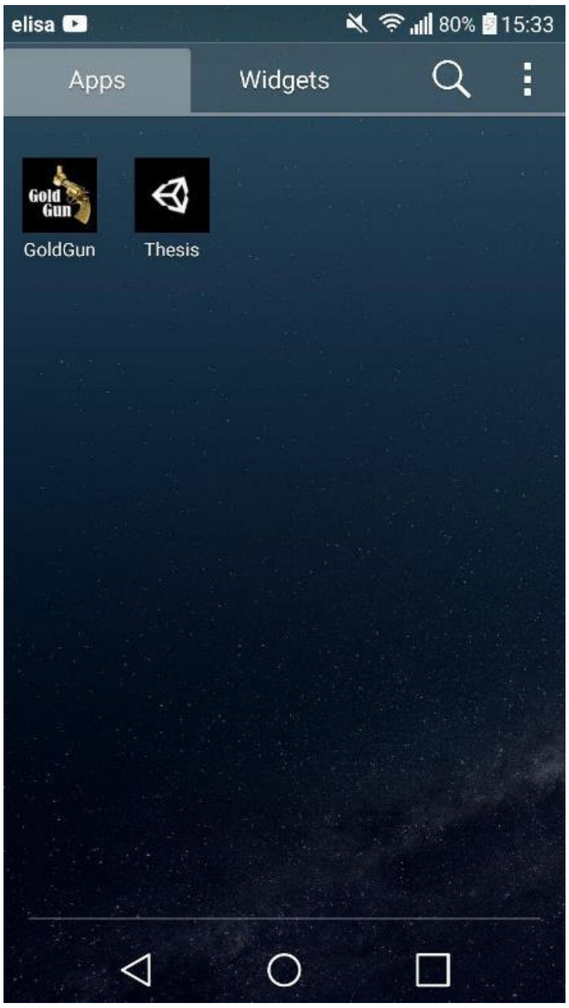


Figure 2. Testing Android smartphone screenshot of Thesis.apk.

Sprint 8. Audio assets feedback

After testing the game demo, the developer wrote a detailed feedback report on all the recorded audio in order to help the sound designer. In addition, the CEO purposed to start modeling the final GDD.

Sprint 9. GDD

The developer focused only in the GDD to show a finished version on next meeting.

Sprint 10. GDD correction

The GDD was presented to the team and feedback was gathered. The possibility of adding visuals to the audio game in order to reach more public was discussed. An art section was added for the incoming designed art.

Sprint 11. ChunkNOR + ChunkNOTState feature developed.

Sprint 12. LoopAnimator + Editor feature developed.

Sprint 13. Sound feedback to gestures feature developed.

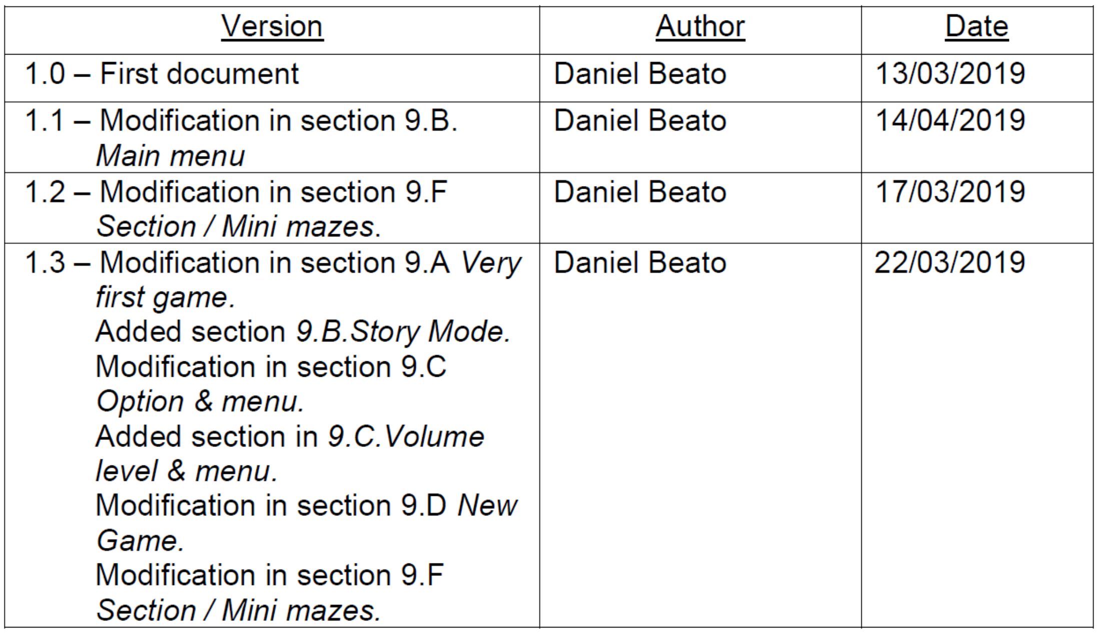
Sprint 14. Story split

In the meeting the group discussed if it was a good idea to have one minute of recorded story before playing, without boring the player. The conclusion was to split the story in seven smaller pieces and to sum it up. Furthermore, the tutorial would be mixed with the story in only one document where the game narrator would tell a small piece of it in every beginning of the maze. For more information, read Appendix 2.

Sprint 15. Score calculation feature developed.

Sprint 16. Menus demo developed.

Every sprint was subject to changes. That is why the main documents such as the story or the GDD have been following a version control table with the version number and changes, the author of those changes, and the date (Figure 3).



Version	Author	Date
1.0 – First document	Daniel Beato	13/03/2019
1.1 – Modification in section 9.B. *Main menu*	Daniel Beato	14/04/2019
1.2 – Modification in section 9.F *Section / Mini mazes.*	Daniel Beato	17/03/2019
1.3 – Modification in section 9.A *Very first game.* Added section *9.B.Story Mode.* Modification in section 9.C *Option & menu.* Added section in *9.C.Volume level & menu.* Modification in section 9.D *New Game.* Modification in section 9.F *Section / Mini mazes.*	Daniel Beato	22/03/2019

Figure 3. Game Design Document version control table.

The GDD had to be modified four times due to the team feedback from the meetings. Those updates were not only changes, but new contents or sections added to the document.

2.2 Game concept

MusicMaze is based on the essence of an infinite runner where the scope is to survive as long as possible avoiding obstacles, dangers, and enemies with different gestures on the screen and in that way achieve a final score. The player will always want to beat his own best score.

The game is presented in the fantasy world of Liande. In that world, there is a small town populated with native creatures, called Chips. One of them is the player's guide and he orientates the player during the whole game, either by telling him some story pieces or by helping him in the tutorial. The player must rescue all the Chip tribe from Liande's maze, where they are lost. For more information or background about the story, read Appendix 1. The maze is located in the jungle and the player must face maze traps, which come from the sides, Nagys, enemies who try to kill the player, and Cugas, monsters that live in the maze.

The game is divided in two parts: the story mode and the final MusicMaze. The story mode is played along seven different levels called mazes. At the beginning of every maze, the Chip tells the player a piece of the story, to let him know the background of the game and also the needed controls to succeed in that maze. The final maze is locked by default. To unlocked, the player must reach the last maze, the seventh maze. The final MusicMaze mode is the mixing of all the maze traps, enemies and monster of the game, played all in one endless level. That mode is the real infinite runner when the player must achieve his highest score.

Regarding the game mechanics, all the gestures are made through the touch screen. They are:

- Swipe up to jump.
- Swipe down to slide down.
- Swipe left and right to turn left and right.
- Double tap to kill enemies.

Tilting was discarded due to negative experience with previous company games.

All the company games use immersive audio, so it is applied on MusicMaze as well. Its aim is to be felt and listened to as realistic sound in a normal room. In the game, obstacles and enemies go toward the player, and maze traps come from the walls. The player hears those sounds coming from far away and getting closer like if they would be in a 3D space and he would be in the middle of it. Those audio assets are managed with two Unity vector3s, the first one, to set the starting point of the sound and the second one for the end point. With the 3D immersive sound, the player can guess where the sound come from and to know how to avoid it by doing the opposite direction gesture. That is also what makes MusicMaze fully accessible to play. Some sounds are easier than others since they are known creatures, such as a dragon or a rhino, and the player already knows which is their environment, air, and ground respectively.

Music must be enjoyable and entertaining since the player is supposed to keep playing for long time periods. There is a music track for each l of the seven levels the story mode is split in. The main feature is not that must be cheerful, but that is synchronized with the player action. That means that every time the player makes an action on the screen, that gesture triggers the activation of a new layer on the soundtrack with different instruments or rhythm, giving back to the player the feeling of having an influence over the music. The music was composed by Janne Kantola, the sound designer.

As already mentioned, MusicMaze is an infinite runner game. The way to make the game endless is through procedural generation. In computing, procedural generation is a method of creating data algorithmically as opposed to manually (Wikipedia 2019). There are many possible algorithms to decide how to create the data. In this project, it was generated through a random algorithm.

MusicMaze can be categorized as a roguelike game. Some of its main characteristics match with the definition. Roguelike is a subgenre of role-playing video game characterized by a dungeon crawl through procedurally generated levels, turn-based gameplay and tile-based graphics. (Pena 2019) Most roguelikes are based on a high fantasy narrative (Blog.roguetemple 2019).

More details about characters, level or environment design and gameplay are shown in Appendix 3.

2.3 Game Design Document

The GDD is the document that includes all the information relevant to the game. That means that it features all the aspects and details needed to explain and describe the game. It must contain the enough information so that each member of the team can know at every moment what to do and why. As it is written on Wikipedia: "*Although considered a requirement by many companies, a GDD has no set industry standard form. For example, developers may choose to keep the document as a word processed document, or as an online collaboration tool*." (Wikipedia 2019). Many hours of researching were needed to finally write the structure of the GDD. In this case, the use of an online collaboration tool was discarded since only the author of the thesis was going to write it.

The contents of the GDD are:

1. Overview (which contains license, document scope, mindset and audience).
2. Target system
3. Development system
4. Scope of the game
5. Story

6. Tutorial
7. Characters
8. Level / environment design
9. Gameplay (where it is explained the menus, gaming experience, traps, obstacles, levels and scoring)
10. Game controls
11. Sound
12. Music
13. Art
14. Accessibility
15. Monetization model
16. References

This table of contents was not only based on external sources, but on the author’s point of view since the GDD is adapted for MusicMaze requirements. Only few GDD sections were added in Appendix 3 of this document as the whole GDD has 42 pages.

2.4 Audio Assets

When the game idea was approved by the company CEO, and the sound designer decided to join the team, the latter one needed a list of Audio Assets to know how much work he would have to do.

The used template is the same myTrueSound Ltd has used in previous games, but adapted for MusicMaze (Figure 4). As the sound designer needed to check the specifications and work remotely, Google Sheets was chosen as the most proper tool to work with. It includes version control as well.

The template contains the following fields:

- Packet: related to its gesture or character in the game.
- Sound: a name which describes briefly the sound.
- Description: detailed description of the sound.
- Variations needed: number of different variations for a sound if needed.
- Loop: bool, if the sound must loop or not. Empty if no.
- File name: technical name for the recorded sound.

- Sentence: in case the sound to be recorded is a sentence.
- Notes

Packet	Sound	Description	Variations needed	Loop	File Name
Steps	Running Steps	Running steps through a stone floor	5-10	Yes	*Steps_StoneSteps*
Maze	Maze Door	Sound of a slow opening huge heavy door made of stone.			*Maze_MazeDoor*
~~*Effects*~~	~~Effort Effect~~	~~Body noise that indecates the player is tired of running.~~			~~*Effect_Effort*~~
Effects	Breath effect	Fast breath from the player that indecates the player is tired.			*Effect_Breath*

Figure 4. MusicMaze audio asset table example.

2.5 Menus flowchart

Menus were not included in the first thesis planning, but the developer decided to design them when the game was going to include a visual interface. In Unity all the menus are managed by a Canvas. Their visibility is set with a boolean displayed in a checkbox. In runtime, the user can see it if the checkbox is true and cannot if it is false (Figure 5).

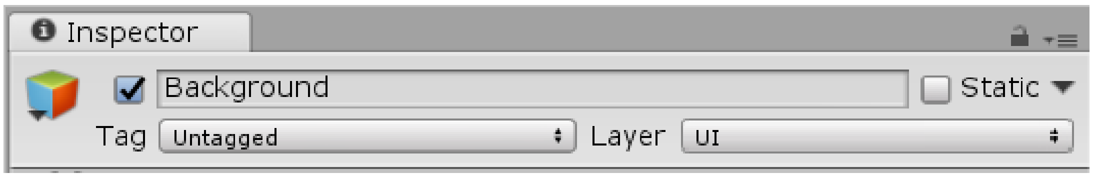


Figure 5. Menu visibility setting in Unity interface.

In the scene called MenuTry2, menus are displayed in Unity as next:

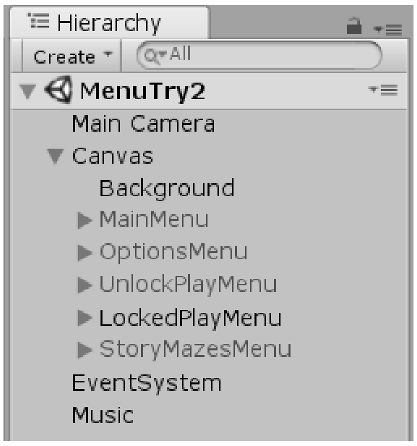
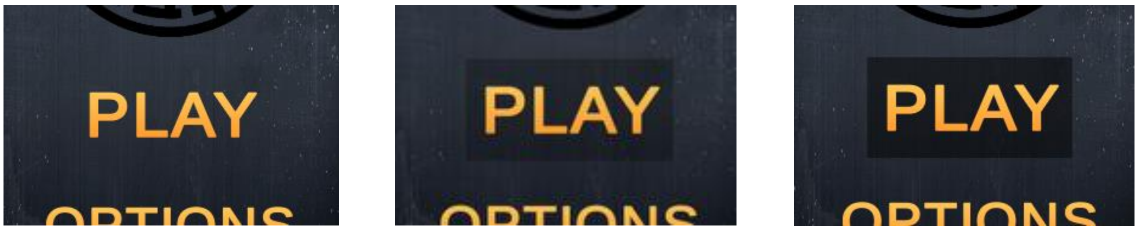


Figure 6. MusicMaze menus hierarchy under Unity scene.

There are 5 menus:

- Main menu: It is made up of the MusicMaze icon and 3 buttons: the Play button, which goes to the unlocked or locked play menu, the Option button, which goes to options menu and the Quit button, which exits the game.
- Options menu: It is made up of sounds volume slider, a music volume slider and a Back button that goes back to the main menu.
- Unlocked play menu: It is made up of three buttons: the Play Story Mode button, which goes to story mazes menu, the Final MusicMaze button, which enters the final game and the Back button, which goes back to the main menu.
- Locked play menu: It has the same composition as the unlocked play menu, but the Final MusicMaze button is blocked and cannot be used. It has a lock picture on.
- Story mazes menu: It is made up of eight buttons: seven numerated buttons for each maze and a Back button to go back to unlocked or locked play menu.

All the buttons have what in CSS is called a hover selector. It is used to select elements when the user mouse over them. (w3schools 2019)



No action. Placing the finger. Tapping the button.

Figure 7. Different hover features in MusicMaze main menu buttons.

In this project hover selector is done to give some feedback to the user when he touches that button without tapping, only passing his finger over that part of the screen. If the user would get to tap on that button after placing his finger on it, it would change its colour to one even darker (Figure 7).

Next, a workflow of all the game menus is displayed:

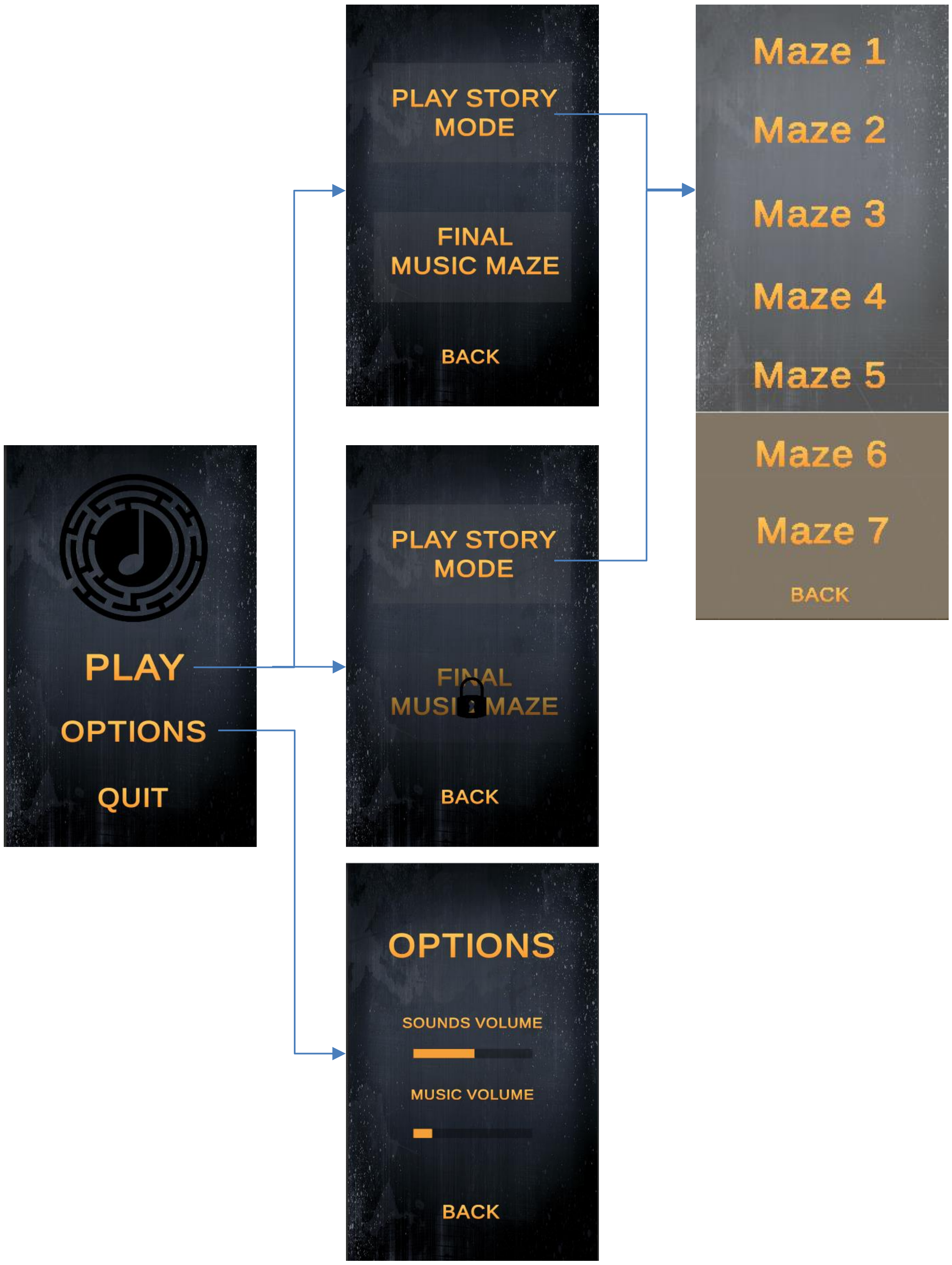


Figure 8. MusicMaze menus flowchart.

2.6 Accessibility

The original audience for MusicMaze was B&VI people. During the development, myTrueSound decided to broaden the target group and reach more people, adding visuals to the game, but it should remain accessible, so everyone could play without keeping eye contact with the phone screen.

Accessibility was created through a UI Accessibility Plugin or UAP. The plugin works by attaching an accessibility component to all relevant UI elements in the Unity scene. These components allow the Accessibility Manager to keep track of every component active on the screen. The plugin can then handle navigation between the corresponding UI elements using Swipes. It will read out names, types and hints and forward interaction events to the UI elements. (UAP Basics 2019).

The plugin supports Labels, Buttons, Sliders, Toggles, Dropdowns and Text Edit Fields. Each accessible UI element has a name, a value, a type and a hint. These are all read out by the accessibility plugin when the element is selected. (UAP Components 2019)

The components that form MusicMaze interface are the buttons in all menus selection, and the sliders in volume settings. The buttons are activated with a double tap, after they have been selected. A single tap on a button, toggle or any other UI element will only select it, but not trigger it. Regarding the sliders, when one is selected, users can double tap it to change the value. Swiping up and down will change the value up or down. (UAP Navigation 2019).

UAP is on sale in Unity Asset Store. It was chosen for being the one that best suited the MusicMaze needs. The game will be released for iOS and Android. Both platforms are fully supported in UAP, and both with VoiceOver (iOS) and TalkBack (Android), screen reader apps included. Besides, if UAP detects an installed screen reader app, the plugin will activate itself. (UAP Platforms 2019)

3 IMAGE

The aim of this section is to show the difference between the original IMAGE version and the developer version after his thesis work. To do that, first, it is required some basic knowledge to understand how IMAGE works.

3.1 IMAGE background

3.1.1 Introduction

IMAGE is an audio game development tool based on game engine Unity to create specialized mobile game for the blind and visually impaired people. In IMAGE, the structure is based on two objects. One is the audio action and the other is the gesture:

- The audio action corresponds with a sound in the real world.
- The gesture matches with a player action on the screen.

Then, those objects are combined in different ways, to create different, of what is called, “game objects”. They are called like that because those are the objects used in the animator as the end parts of the game, but they are not inserted on it straightaway. Every game object type demands its state class in order to be inserted in the animator.

3.1.2 IMAGE elements

Next, a brief explanation of the main elements of the tool to better understand its functioning:

Audio action

It represents a game sound. It has several attributes:

- Delay: float type. It determines in seconds how long the sound must wait to be played.
- Is Background: boolean type. It determines whether the sound is a background sound such as ambiance or music.

- Fmod event: it is the fmod event to be played after the delay.
- Start position: Unity vector3 type. It is the start place in the space where the audio will begin playing.
- End position: Unity vector3 type. It is the end place in the space where the audio will finish playing.

Gesture

Gestures represent the actions of the player. They have, among others, a timeout attribute, to determine when to stop waiting for a player action on the screen. There are two types:

- Gesture Swipe: It waits a swipe on the screen. The direction can be determined with an attribute (UP, DOWN, LEFT, RIGHT).
- Gesture Tap: It waits a single tap on the screen.

GameChunk and GameChunkState

The game chunk is what it was called “game object”. Indeed, this data structure contains two elements:

- A list of audio action objects.
- A gesture.

It is the most used object in the game as the mechanics consist in hearing an obstacle coming and make a gesture to avoid it.

The GameChunkState class defines how the game chunk is actually used in the game through Unity animator. According with the game chunk state, the animator will play all the audio actions located in game chunk list and, then, when the last one ends, the player action will be accepted. Regarding the audio action list, it plays all the audio actions at the same time, but each one is supposed to have a different delay, set by the developer, in order to hear them in order and separately. Getting together several audio actions is not usually used for the main game, in the playing process at least, but it can be in other situations, the tutorial for example, where the script is recorded in small separate pieces, and it must play some in a row before a gesture.

ChunkStateAsync

It is similar to game chunk state, but with a peculiarity. In the previous state, the player must wait until all the audio actions are played to make the gesture. In the chunk state async, the player can make the gesture as soon as he hears the sounds.

3.1.3 Unity Animator Controller

According to Unity documentation, *“an Animator Controller allows you to arrange and maintain a set of Animation Clips and associated Animation Transitions for a character or object. In most cases it is normal to have multiple animations and switch between them when certain game conditions occur.”* (Unity 2019). Thus, its functioning is comparable to a state machine diagram.

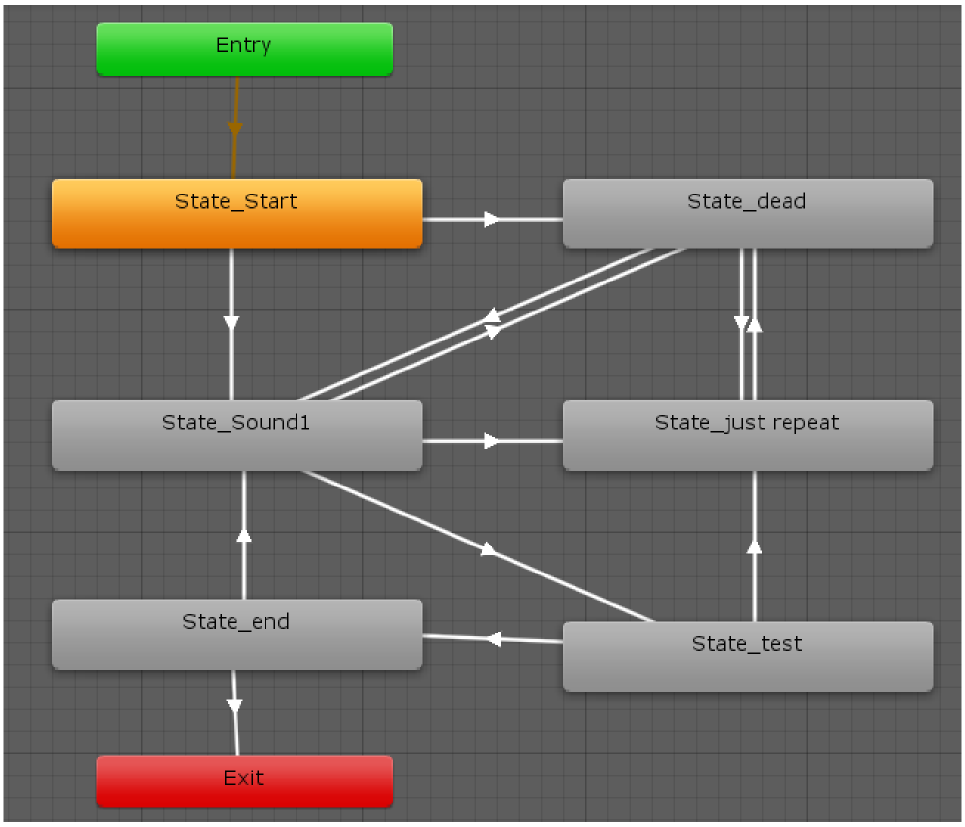


Figure 9. Diagram in animator example.

The game bases on states, that is, each GameChunk matches with a state through a state class which regulates how the game object behave in the game. Each state can have more than one state class that will execute diverse behaviours.

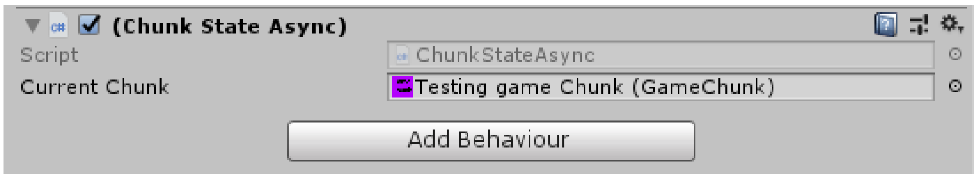


Figure 10. Behaviour example of a state with a game chunk.

As in every state machine, there are transitions. According to Unity documentation, *"Animation transitions allow the state machine to switch or blend from one animation state to another. Transitions define not only how long the blend between states should take, but also under what conditions they should activate."* (Unity 2019). They are represented as arrows in a diagram and they are triggered depending on the state class. There are three programmed transitions in IMAGE: Done, Time Out and Incorrect. The first one, if the gesture is correct, the second one if the current state runs out of time and the last one if the gesture is incorrect. Every state can have more than one transition.

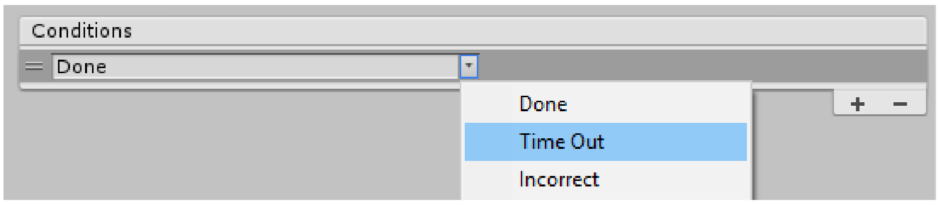


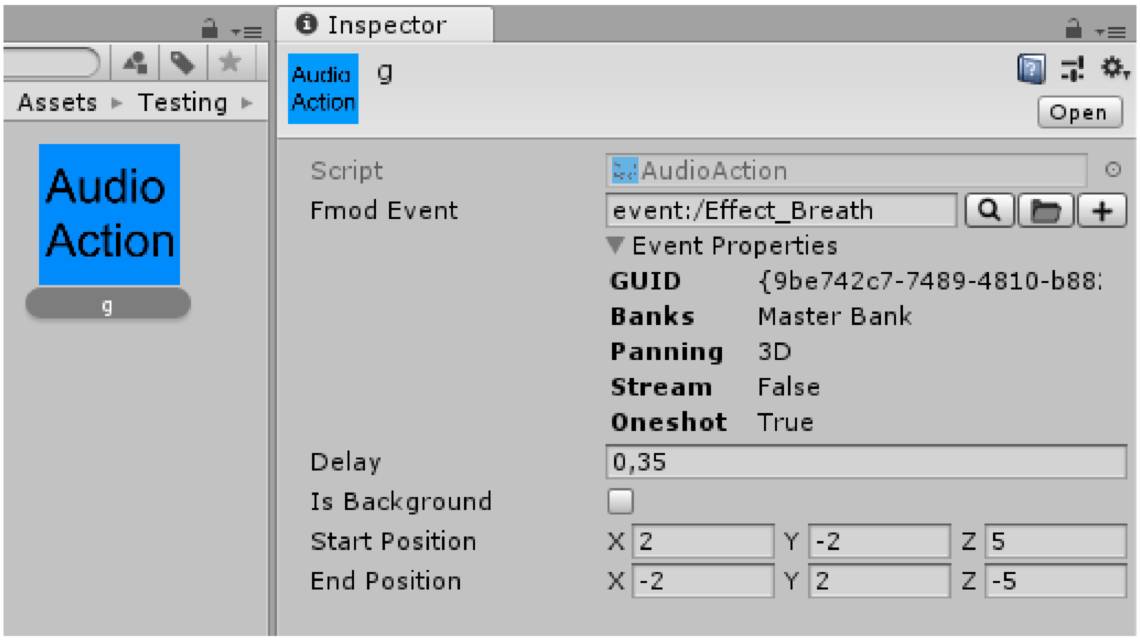
Figure 11. Animator transitions Done, Time Out and Incorrect.

3.2 FMOD integration. Previous IMAGE version

Before the company decided to integrate FMOD, IMAGE was a useable tool. The developer worked first in that version, solving a bug that made ChunkStateAsync totally useless, and programming the incorrect transition for the swipe gesture. Nevertheless, it had some lacks in the audio field, since all the sound should be edited separately, and it could not be changed in execution time. That is why, myTrueSound decided to update IMAGE tool to make it compatible with FMOD.

FMOD Studio is an audio content creation tool for games, with a focus on a Pro Audio approach. It has an interface that will be familiar to those using professional Digital Audio Workstations and is loaded with powerful features. (Fmod 2019).

Audio files are imported to FMOD as normal assets, preferably in wav format. Then FMOD converts them into events and assigns them a bank. Banks are collections of the metadata, assets, events and other content used in a project, to be built and used in the game. (Fmod 2019). Then, after building for all platforms, and if everything is well configured and linked in Unity, FMOD events are accessible from IMAGE.



Script AudioAction
Fmod Event event:/Effect_Breath
Event Properties
GUID {9be742c7-7489-4810-b88:
Banks Master Bank
Panning 3D
Stream False
Oneshot True
Delay 0,35
Is Background
Start Position X 2 Y -2 Z 5
End Position X -2 Y 2 Z -5

Figure 12. Audio action object inspector in Unity interface.

As it is explained above, FMOD event is an attribute of the audio action object. That is how sound interacts with IMAGE. Loupe icon is used to consult the available fmod events list.

3.3 IMAGE improvements

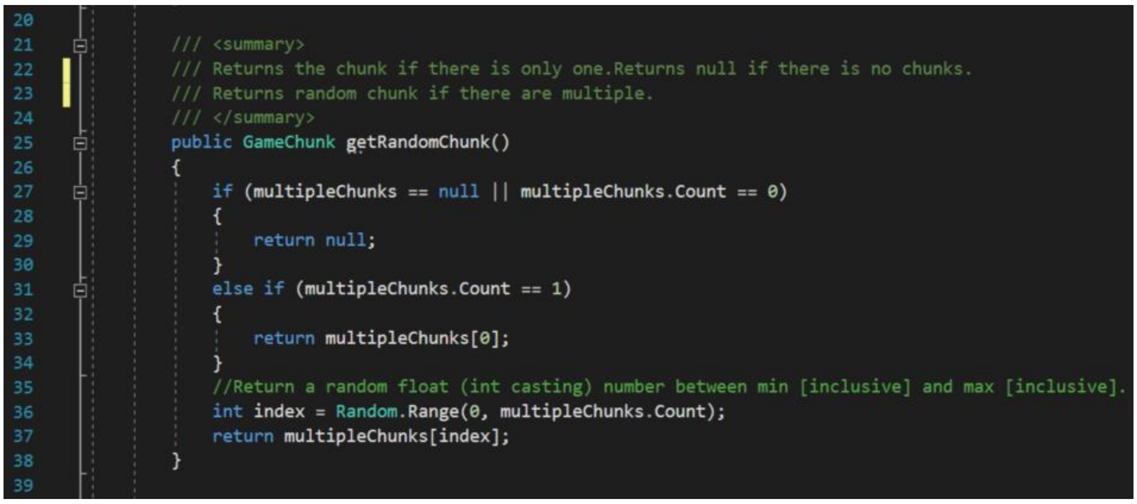
IMAGE has been properly explained, how it works, and which are the features implemented already. In this section, the author of the thesis will explain what he has done and why.

3.3.1 MultipleChunk class

The game MusicMaze was thought to be an infinite runner. Every infinite mode must be generated procedurally, that is, generated in execution time through an algorithm. To do that, the developer came with the MultipleChunk idea.

In MusicMaze, traps, enemies or monsters come towards the player, to try to kill him and he must avoid them to survive while running, if he fails, he dies. Those dangers are the GameChunks, which have the peculiar danger sound and the suitable gesture to dodge it. Those game chunks are what really build the game in runtime.

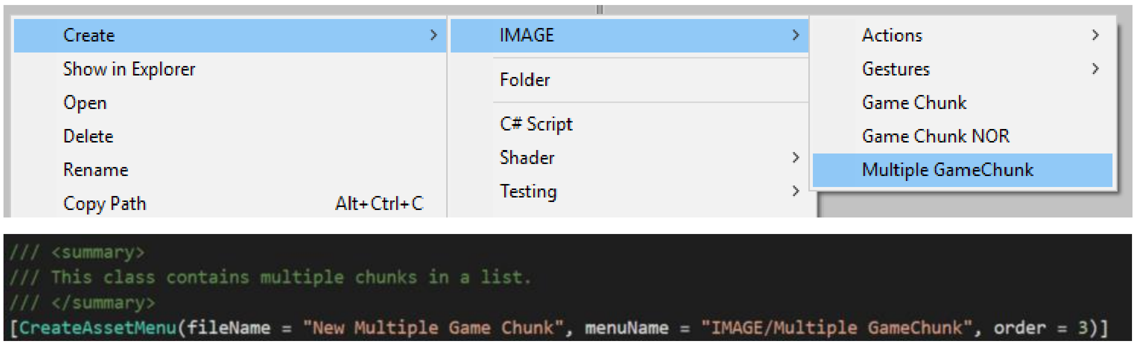
Multiple chunk object is basically a list of game chunks. That list is the structure used in the algorithm, and the algorithm method is called `getRandomChunk`. It returns a game chunk out of the multiple chunk list randomly, based on its size. It returns a null if the list is empty, and the game chunk if the list only has one. The algorithm is pretty simple and does not require a big amount of CPU time.



```
        /// <summary>
        /// Returns the chunk if there is only one.Returns null if there is no chunks.
        /// Returns random chunk if there are multiple.
        /// </summary>
        public GameChunk getRandomChunk()
        {
            if (multipleChunks == null || multipleChunks.Count == 0)
            {
                return null;
            }
            else if (multipleChunks.Count == 1)
            {
                return multipleChunks[0];
            }
            //Return a random float (int casting) number between min [inclusive] and max [inclusive].
            int index = Random.Range(0, multipleChunks.Count);
            return multipleChunks[index];
        }
```

Figure 13. Screenshot of procedural algorithm method getRandomChunk().

Multiple chunks can be created from Unity interface through IMAGE create menu, making a right click in the project folder space. That is programmed in the same `MultipleChunk.cs` class. After creating the object, it will have the “New Multiple Game Chunk” default name.



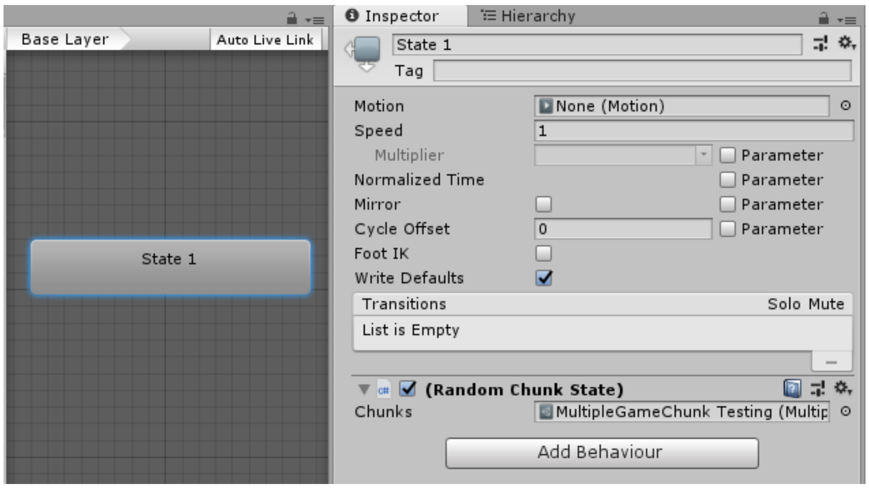
```
/// <summary>
/// This class contains multiple chunks in a list.
/// </summary>
[CreateAssetMenu(fileName = "New Multiple Game Chunk", menuName = "IMAGE/Multiple GameChunk", order = 3)]
```

Figure 14. Process of MultipleChunk creation menu.

3.3.2 RandomChunkState and RandomChunkStateEditor classes

As it has been explained before, every structure or game object needs a state to be able to interact and be executed in Unity animator and then, in the game. Thus, once the multiple chunk game structure is created, it is needed a state for it, it is called RandomChunkState.

RandomChunkState functioning is simple, the class checks if there is any chunk selected in the state behaviour. If not, IMAGE makes a positive trigger transition, that is, Done transition. If so, it calls the method `getRandomChunk` to choose a game chunk randomly and passes it to the animator. It checks also if the passed game chunk is null.



State 1
Tag
Motion: None (Motion)
Speed: 1
Multiplier: ☐ Parameter
Normalized Time: ☐ Parameter
Mirror: ☐ ☐ Parameter
Cycle Offset: 0 ☐ Parameter
Foot IK: ☐
Write Defaults: ✓
Transitions Solo Mute
List is Empty
✓ (Random Chunk State)
Chunks: MultipleGameChunk Testing (Multip

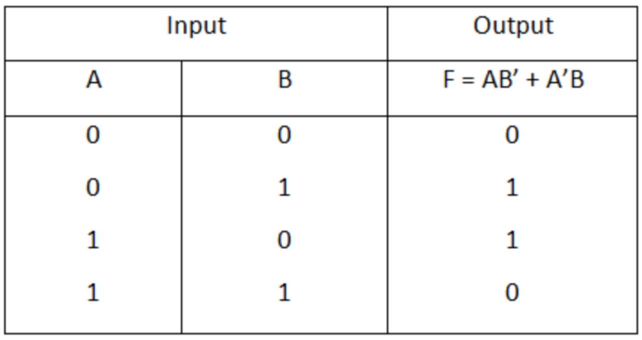
Figure 15. Diagram state inspector with MultipleGameChunk as a behaviour.

This class inherits from ChunkState in order to reuse all the audio player, gesture waiter and gesture checker methods. The editor of this class is done so the programmer will only see the current multiple game chunk in the Inspector of the diagram state.

3.3.3 ChunkXOR class

With the GameChunk a programmer can build a game manually, where the player would hear a sound and right after it, he would make the correct gesture. With the MultipleChunk, a programmer can build a procedural random game which is created in runtime, but with the same mechanics: a sound and only one correct gesture.

That is why the developer decided to create the ChunkXOR. The feature of this game chunk is that it has two correct gestures instead of only one, so then now on, there can be enemies coming that can be avoided making two different gestures.



Input		Output
A	B	$F = AB' + A'B$
0	0	0
0	1	1
1	0	1
1	1	0

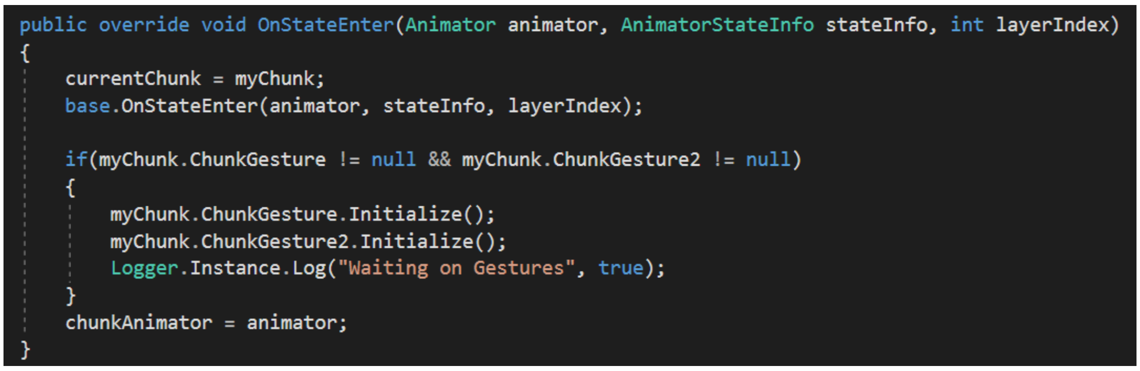
Figure 16. XOR truth table (Csetutor 2019).

ChunkXOR structure has his name because of XOR (exclusive disjunction) logical gate. That means, the game chunk accepts either gestures A or gesture B, but not both at the same time, only one. It can be better understood with its truth table (Figure 16).

3.3.4 ChunkXORState class

It inherits from ChunkStateAsync class, which inherits from ChunkState, so with a chunk XOR, the player is able to make the gesture right after the sound starts playing. Notwithstanding, since it has two gestures, most of the methods have been overridden.

In the method OnStateEnter, which is executed once at the beginning of the state, both gestures must be initialized.



```
public override void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{
    currentChunk = myChunk;
    base.OnStateEnter(animator, stateInfo, layerIndex);

    if(myChunk.ChunkGesture != null && myChunk.ChunkGesture2 != null)
    {
        myChunk.ChunkGesture.Initialize();
        myChunk.ChunkGesture2.Initialize();
        Logger.Instance.Log("Waiting on Gestures", true);
    }
    chunkAnimator = animator;
}
```

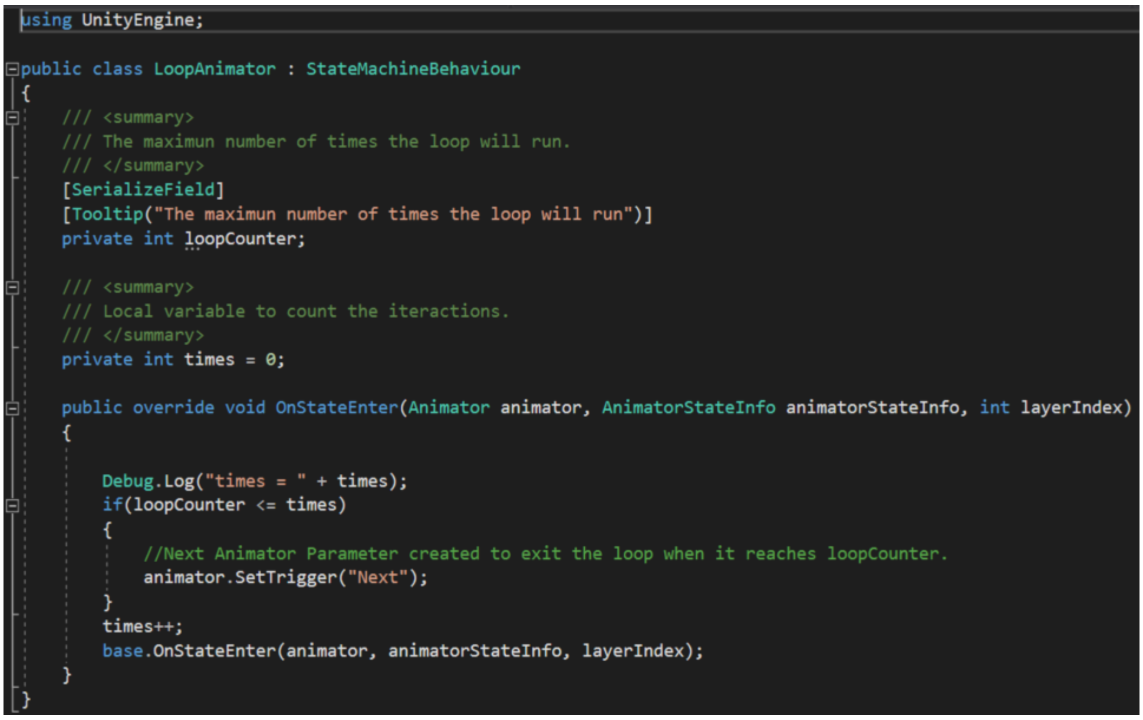
Figure 17. OnStateEnter function. ChunkXOR gestures initialization.

Then, when the audio is played and the player makes a gesture, IMAGE compares it with both gestures to know if the player is right, but before it checks that generated gestures are not null.

3.3.5 LoopAnimator and LoopAnimatorEditor classes

With game chunk and chunk XOR structures, it is possible to build a game with different gestures and sounds. With multiple chunk and its state, that game can be randomized and procedurally generated. The problem is the developer must program a state and several transitions in the animator diagram as many times as enemies in the game he wants, if he wants a limited amount. All that is useless, since it is a repetitive task with any change. It was needed some tool that could repeat a certain state as many times as desired. That tool is the LoopAnimator.

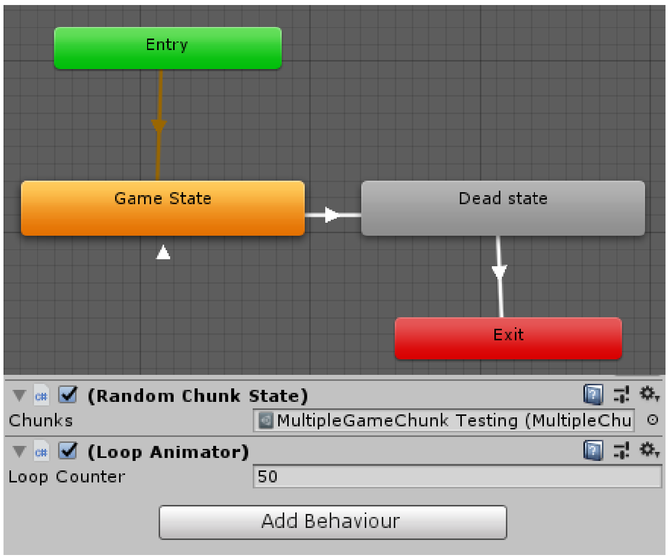
LoopAnimator class creates a behaviour that can be added to any state. The programmer introduces the number of times he wants to replay a certain state, and in every iteration, the class checks if a counter is equal to the number the player introduced. If so, the class triggers a transition ‘Next’ to go to the programmed state through the transition labelled like that.



```
using UnityEngine;

public class LoopAnimator : StateMachineBehaviour
{
    /// <summary>
    /// The maximun number of times the loop will run.
    /// </summary>
    [SerializeField]
    [Tooltip("The maximun number of times the loop will run")]
    private int loopCounter;

    /// <summary>
    /// Local variable to count the iteractions.
    /// </summary>
    private int times = 0;

    public override void OnStateEnter(Animator animator, AnimatorStateInfo animatorStateInfo, int layerIndex)
    {

        Debug.Log("times = " + times);
        if(loopCounter <= times)
        {
            //Next Animator Parameter created to exit the loop when it reaches loopCounter.
            animator.SetTrigger("Next");
        }
        times++;
        base.OnStateEnter(animator, animatorStateInfo, layerIndex);
    }
}
```

Figure 18. LoopAnimator class screenshot.

LoopAnimatorEditor class is made only to display the attribute `loopCounter` in Unity and hide the counter `times`.



(Random Chunk State)
Chunks: MultipleGameChunk Testing (MultipleChu
(Loop Animator)
Loop Counter: 50

Figure 19. Example of game demo diagram in Unity animator.

Thanks to loop animator feature it is much easier to create a game in the animator. With a multiple chunk with some game chunks or chunks XOR inside it, and the loop animator, in just a state it is possible to build a large procedural demo.

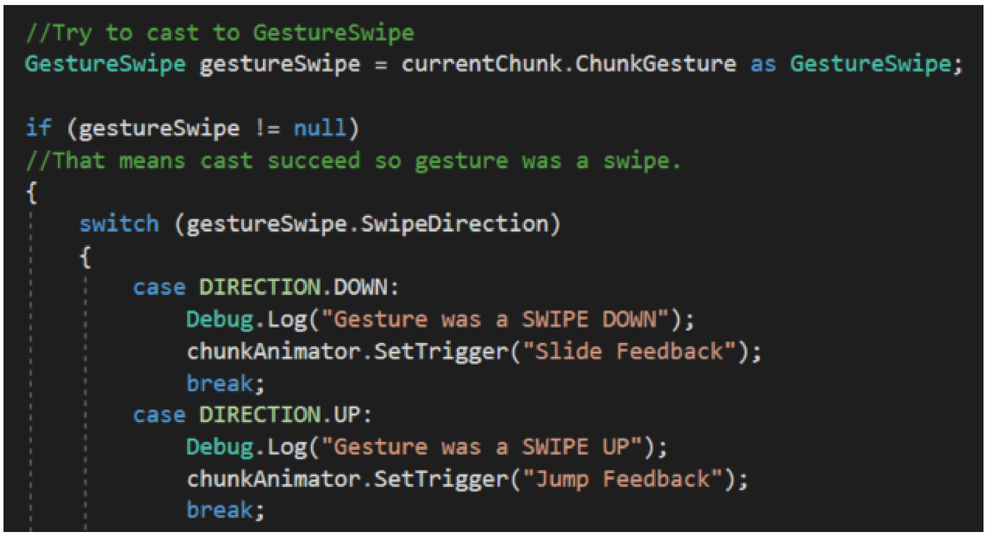
3.3.6 Sound feedback to gestures

At that point, the game was already functional, but not enough playable. It could be highly improved.

The player must make gestures to avoid the obstacles or enemies. Those gestures represent movements in the game, and those movements are jumping, sliding, turning left or right and killing enemies. When the player makes a gesture, he does not receive any feedback from it to know if it went well. Of course, if he dies, he will hear a crashing sound, but he also needs some visual or sound confirmation to know that the gesture was correct.

The idea was that every time the player made a correct gesture, he heard a feedback sound right after. If he jumped to avoid an obstacle, he would hear a sound of the player landing on the ground, for example. The same for the rest of actions.

There was a concept problem at the beginning: if the game is generated procedurally and game chunks are played randomly, it is impossible to know which kind of gesture will be asked next. So, what was done to implement this feature was to check if the player had made a gesture, and if it was correct, then, try to cast it with the different kinds of gestures, which are, swipe up, down, left, right or tap and the screen. If the cast succeeded, then, IMAGE would trigger the animator transition of that guessed gesture. In the animator diagram, there would be a state for each transition, and as it is been explained, a transition for each gesture.



```
//Try to cast to GestureSwipe
GestureSwipe gestureSwipe = currentChunk.ChunkGesture as GestureSwipe;

if (gestureSwipe != null)
//That means cast succeed so gesture was a swipe.
{
    switch (gestureSwipe.SwipeDirection)
    {
        case DIRECTION.DOWN:
            Debug.Log("Gesture was a SWIPE DOWN");
            chunkAnimator.SetTrigger("Slide Feedback");
            break;
        case DIRECTION.UP:
            Debug.Log("Gesture was a SWIPE UP");
            chunkAnimator.SetTrigger("Jump Feedback");
            break;
```

Figure 20. Screenshot of gesture swipe cast attempt.

Since the gesture result, correct or incorrect, and the gesture type should be reviewed at the same time, the sound feedback to gestures feature was implemented in ChunkStateAsync class. It could be added to ChunkState class as well, but since the game will be based on async game chunks, it is not needed.

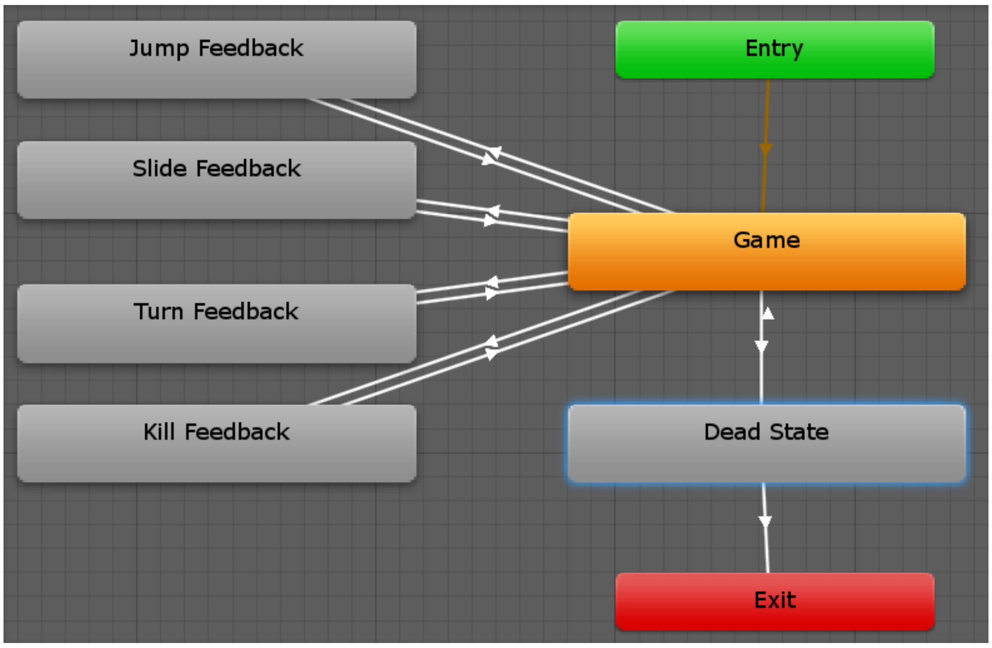


Figure 21. Game state diagram with sound feedback in Unity animator.

Figure 21 is an example of how a game is developed with the sound feedback feature. The state Game is the main one where the multiple chunk runs. Depending on its output, the flow goes to the state which gesture was correctly done by the player. Then, in that state there is a game chunk with the sound of the character doing the action and no gesture asked. After playing that sound, the flow comes back to the Game state to keep playing until the player dies, either by making an incorrect gesture or by reaching the timeout.

3.3.7 Score mechanism

MusicMaze is an arcade infinite runner. That means the player will always have to beat his best score. Thus, scoring is a crucial factor in the game.

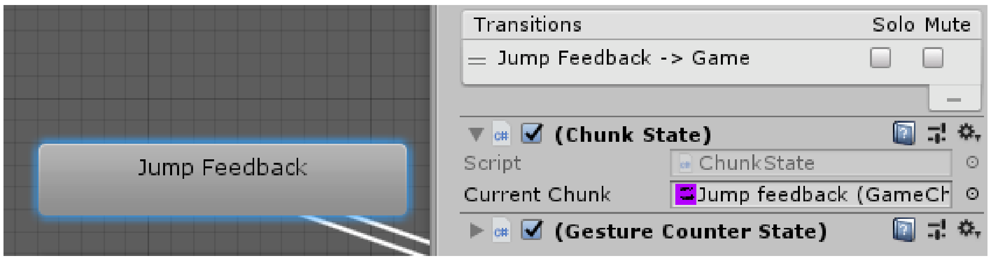
The score calculation is based on the number of correct gestures in the game. Besides, there are a special enemy which gives the player extra points. So, not all the gestures give the same score. That means there must be an independent counter for each variable.

Some studies show that players usually prefer a high score starting at about 10000 points. (Tvtropes 2019). Moreover, in order to keep them motivated, it must be very easy to get those 10000 points, pretty difficult to get one million points and almost impossible to reach 10 million points. That is how next equation was though:

$$score = gestureCounter * 6153 + extraPointsCounter * 10000$$

Equation 1. MusicMaze score system.

That is how score is calculated once the player dies in the game, with a gesture and an extra point counter. In IMAGE, those counters are programmed in separated classes: GestureCounterState and KilledCugasCounterState, this last one has its name due to game character and story. They inherit from StateMachineBehaviour, so they are state classes used as behaviours in the animator. That means they are added to a state and they run every time its state runs. To difference between different gestures, the behaviours are placed in the feedback sound states.



Transitions Solo Mute
Jump Feedback -> Game
(Chunk State)
Script ChunkState
Current Chunk Jump feedback (GameCh
(Gesture Counter State)

Figure 22. Jump feedback state inspector in Unity interface.

In figure 22, jump feedback state has the game chunk of the jump feedback sound and the gesture counter state. This last will trigger the counter increment every time the player makes a correct jump action.

4 CONCLUSION

This thesis aimed to improve the game development tool IMAGE of myTrueSound company, and to create a new sellable phone game for the B&VI which would fit in those new features. The improvements were up to the developer since the game concept was as well.

After the thesis work, the system, IMAGE, is able to create games with procedural generation structure thanks to the MultipleChunk and RandomChunkState classes. Game content is generated in runtime based on random selection algorithm. The designed structure is the multiple chunk which contains a limited amount of game chunks.

The system is able to recognize more than one correct gesture per action in Unity animator. That is achieved through the ChunkXOR and ChunkXORState classes which enable the chunk XOR structure in IMAGE.

The system is able to give sound feedback to the user while playing. That is achieved through the object try cast made in ChunkState class. This feature allows IMAGE to give a sound feedback depending on the player gesture right after he makes an action in the game.

The game development through IMAGE is easier. LoopAnimator class allows the programmer to create an action that can be repeated a determined amount of times. It works with all the structures, so the same sound and gesture can be replayed through a game chunk, or the procedural generation through the random state and the multiple chunk can be done only a certain number of times.

The game concept was a success with MusicMaze, an infinite runner procedurally generated with 3D immersive audio, based on a fantasy animated world. The game idea convinced the company and the team. The Game Design Document (GDD) was written and a game demo was made. The game is still missing the beta testing process; thus, it is not ready to be released.

Unfortunately, time was limited and the music - player action synchronization was not achieved.

Further development is required, and it is summarized in two main points. The first future development is to continue, with the graphical implementation. With a UI, MusicMaze may reach more potential customers. The second future development is to implement the music and action synchronization. Even though it is a difficult task, it would give an attractive add-on to the game. Other ideas to make the game grow are: to introduce new characters, to add in-app purchases, or to connect the game with social media and Google Play.

Summing up, the real challenges of this thesis have been:

1. Taking on a project with no background or previous knowledge about Unity, with bare help from the company, where the main procedure was to learn by doing, and spending tons of time on researching IMAGE.
2. Knowing which features were already implemented on IMAGE and thinking of new features that could be used to create a new game based on them.

In conclusion, although perhaps some solution or decision may not be the most accurate, the project has progressed, IMAGE features have been programmed and the MusicMaze demo is completed.

REFERENCES

Csetutor (2019). XOR truth table. https://www.csetutor.com/logic-gates-and-truth-table/ Retrieved 08-06-2019.

Fmod (2019). Fmod Studio. https://www.fmod.com/resources/documentation-api?version=1.10&page=content/generated/common/introduction_web.html Retrieved 08-06-2019.

Fmod (2019). Fmod Banks. https://www.fmod.com/resources/documentation-studio?version=1.10&page=getting-events-into-your-game.html Retrieved 08-06-2019.

myTrueSound Ltd (2018). Interactive Entertainment Experiences with Immersive Audio for the Blind & Visually Impaired (Market Size and Opportunity). myTrueSound PitchDeck Investors Autumn 2018.

Pena, M (2019). Roguelike subgenre. https://blogderoguelikeparatutfg.blogspot.com/ Retrieved 08-06-2019.

RogueTemple (2019). Roguelike. https://blog.roguetemple.com/what-is-a-traditional-roguelike/ Retrieved 08-06-2019.

Theseus (2019). The theses and publications of the Universities of Applied Sciences on the Internet. https://www.theseus.fi/ Retrieved 12-06-2019.

Tvtropes (2019). Game scoring. https://tvtropes.org/pmwiki/pmwiki.php/Main/ScoringPoints Retrieved: 08-06-2019.

UI Accessibility Plugin (2019). Accessibility components. http://www.metalpopgames.com/assetstore/accessibility/doc/MakeYourUIAccessible.html#AccessibilityComponents Retrieved: 08-06-2019.

UI Accessibility Plugin (2019). Plugin Basics. http://www.metalpopgames.com/assetstore/accessibility/doc/MakeYourUIAccessible.html#Introduction Retrieved: 08-06-2019.

UI Accessibility Plugin (2019). Plugin Navigation. http://www.metalpopgames.com/assetstore/accessibility/doc/Navigation.html Retrieved: 08-06-2019.

UI Accessibility Plugin (2019). Supported Platforms. http://www.metalpopgames.com/assetstore/accessibility/doc/Platforms.html#SPlatforms Retrieved: 08-06-2019.

Unity (2019). Animator controller documentation. https://docs.unity3d.com/Manual/class-AnimatorController.html Retrieved: 08-06-2019.

Unity (2019). Animator transitions. https://docs.unity3d.com/Manual/class-Transition.html Retrieved: 08-06-2019.

W3schools (2019). Hover selector. https://www.w3schools.com/cssref/sel_hover.asp Retrieved 08-06-2019.

Wikipedia (2019). GDD structure. https://en.wikipedia.org/wiki/Game_design_document Retrieved: 08-06-2019.

Wikipedia (2019). Procedural generation. https://en.wikipedia.org/wiki/Procedural_generation Retrieved: 08-06-2019.

Appendix 1. MusicMaze original story

In this subsection, the original story of Liande and the Chips tribe as it was written by the developer:

(Epic music and a shining decreasing sound)
(Few seconds wait)
(Epic, slow music starts)

CHIP

Hey! Welcome to my world, welcome to LIANDE. We were waiting for you! I'm Chip! It's so great to meet you finally. (a beat) Don't worry, everything is quiet now, they have gone. (A beat).

You see, we are the Chip tribe, we have been living in this world for generations and generations. We have always been a peaceful people and we have been able to enjoy life together as brothers. At least until recently.

(A beat. Sad and slower voice)

I just remember that it all started on a cold, dark night. The tribe was dancing and singing around the fire when a portal from another world suddenly opened. Before we could react, strange creatures invaded our place and captured some of us.

We call them "Nagys" and they had come to hunt us. They did not attack us with stones and sticks, they brought with them a handful of metallic machines with which they destroyed our homes.

They were not tall or strong, but they were smart. They handle iron and other materials and I do not even want to think what they can do with the fire. They are evil beings who only seek destruction and chaos as a way of life.

Our people divided, those who managed to escape, fled in terror towards the North and entered the Maze with the hope of not being found there. But now they are alone, lost and scared.

The sun has not come up in Liande since then. I fear we will not see the light of day again until the whole tribe are together again.

The maze is not a safe place, thousands of monsters lurk inside, we call them "Cuga", the Maze Monsters. Our ancestors built that maze thousands of years ago, to protect our people from Cugas. They placed an innumerable number of all kinds of traps to kill them. Paradoxically, now those traps could kill any Chip or even you.

There’s still more. The “Nagys” that came out of the portal went after the Chips and entered the Maze as well. They have been looking for my friends for days and do not mind burning forests or knocking down trees to find them. They have enough weapons and machines to destroy the entire planet.

Without light, we only have the sound.

Help us to recover the peace. Find my friends. I will be your guide!

(Music increases)

Appendix 2. MusicMaze split story

In this subsection, the story of Liande and the Chips tribe split in different mazes to connect it with the game tutorial. Only two first mazes of 7 shown:

MAZE 1.- 1 minute. Only Left – Right to find chips and swipe up to jump the river.

⏱ 45 seconds. – 164 words.

Hey! Welcome to my world LIANDE. We were waiting for you! All started on a cold and dark night. We were dancing and singing around the fire, when that portal from another world suddenly opened.

Before we could react, strange creatures invaded our place and captured some of us. Those who managed to escape, fled in terror towards the old Maze. But now they are alone, lost and scared. We need you to find them and bring them back.

My chip friends are normally hiding behind every corner, and they make sounds so you can easily recognize them. Turn to the left or to the right when you listen them (do that by swiping from the center of the screen to the desired direction), so you avoid also crashing against the walls. Here, close to the entrance you only need to worry about the rivers. Try not to fall in them by jumping (do that by swiping up). Be careful, and please, don’t stop

(Maze starts, with door opening)

Music Maze Story

MAZE 2.- 2 minutes. Previous movements, plus Cugas: double tap and two traps.

⏱ 45 seconds – 146 words.

The maze is not a safe place, nooo, oh no, not really. Thousands of monsters lurk inside. We call them "Cuga". They make sounds like (sound) or similar.

When they are on our way, you could jump over to avoid them (do that by swiping up), but I prefer that you kill them instead, so we don’t find them ever, ever, ever again. They are so disgusting. So, if you kill them (double tap on the screen), I will give you extra points, ok?

Our ancestors built this maze thousands of years ago, to protect our people from Cugas. And they filled it up with an innumerable number of all of traps to kill

them. Paradoxically, now those traps could kill any Chip or even you.

For now, remember to jump over the stone ball (sound) and to slide down to avoid the arrows (sound).

Let’s go!

Appendix 3. GDD section

In this subsection, the GDD sections: characters, level environment and new game gameplay. It is shown to provide a better background of MusicMaze.

Characters

Next, a list of characters with their motivation, description, history, etc:

- Runner/player: All it is known is that he came to Liande in the beginning of the game, when Chip starts to tell the story. His scope will be to recover as many Chips as possible and run to get out of the maze, which is impossible because it never ends (infinite runner). He is a human. He will enter the maze and start running until he dies. He does not have a name because the player has to identify himself with him.
- Chip Pet: His name is Chip and he comes from Chip's tribe. All of them have the same name, Chip. We will talk about our friend and instructor as Chip and about the tribe and Chip's friends as Chips. He is the survivor of the big Nagy's attack. He saw the Nagys had heavy machines and how they built them, so he can provide you information about them. As part of the Chips tribe, he knows his planet and he can give the player orders, indications or instructions of how to survive inside the maze or avoid the death. His scope is clear, help the player to survive and save all his Chips friends lost in the maze.
- Chips tribe: As it is said in the story, the tribe have been living in Liande since time began. They are a pacific and quiet community. They are small and lovely (more physical details in 13.- Art), but they are not warriors, so when 'Nagys' came, they try to escape and ran away to the Maze. Now they are hidden there, frightened for their destiny. The player will always find one in every corner or turn.
- Nagy: One Nagy or several Nagys. They are the creatures who came from the portal. They are naturally evil, with the only intention of destroy. Scientifically much more advanced than Chips. They can be described as aliens for Chips. They are enough smart and advanced that they design and build mechanical machines to fight and kill. Their scope is to kill the Chips, and since the player is the one who is trying to save them, to kill the player too. To achieve their goal, they will place traps inside the Maze, and they will build more mechanical machines with different

features and skills. They are small as the Chips, a bit taller maybe. That explains how the player can jump over their metal machines.

- Cuga: One Cuga or several Cugas. They can also be referred as ‘Maze Monsters’. These creatures are real monsters who were wandering in the Maze before the portal opened, since time began. They have animal instinct, they are monsters, they only want to kill everything they see to feed themselves. They will attack the player if they see them. There will not be any conflict between Nagys and Cugas. It is very important to see the difference between these two to understand the story.

Level/environment design

Liande is a small planet whose population have always been Chips. Since Chips tribe is a developing community who can handle fire but not much more, the environment of the planet is still almost untouched. The main ecosystem is jungle, full of nature, trees, plants. It does not mean in the future we cannot create more mazes with different environments, such as deserts, quicksand, icebergs, see, rivers, etc.

The Maze was built by Chips’ ancestors in order to get Cugas away the people. During the Maze building, Chips’ ancestors placed lots of traps to kill Cugas when they passed by. These traps are arrows, lances, poisonous darts, etc (more details in section 9.3). Now, Chips have returned that Maze and the player must enter to rescue them, all those traps are a threat and a risk for him as well.

The Maze is done with big old rocks. Nature has got inside it and it is easy to see bindweeds, moss, lichens, etc. The wind has eroded the stones and the floor so there are some sand and soil on the ground, things the player will notice when he listens to slide sound. It has high walls and some parts or stones have fallen down with time. It can be seen as an old Aztec building hidden in the middle of the jungle.

Walls and floor are made of stone, there is no roof. Corridors are wide; however, the player can only run in one direction with no chance of moving laterally.

Gameplay. New game

When the game starts, the first thing the player listens to is the huge main Maze door opening with a squeal. After that the player starts to run and he can hear the fast steps against the stone floor.

He is in the first maze. The player will have to go through different mazes until he dies. A maze is just a combination of gestures that all together will fit with a soundtrack (more information about music in section 12.- Music).

To progress in the game, the player will have to survive avoiding traps, monsters and enemies. Next, all the obstacles the player can face and how to survive them:

Maze Traps

Chips' ancestors placed lots of traps to kill Cugas, but now they can kill the player because they get activated with the movement if someone pass by. There is not any case when any trap can kill or concern Nagys.

- Stone Ball: A stone ball coming against the player, he must jump to avoid it.
- Flying axe/sword: Nagys throw axes, swords or knives. The player must slide down to avoid them.
- River: There is a river crossing the Maze. The player must jump over it to continue playing.
- Arrows: Arrow are thrown from the walls. Player must slide down to avoid them.
- Poisonous darts: Poisonous darts are thrown from the walls. Player must slide down to avoid them
- Lances: Lances are thrown from the walls. Player must slide down to avoid them. Possibility of mix them with arrows.
- Stone collapse: Some stones from the wall fall and the player must jump to avoid the obstacle.
- Hole in the ground: There is a hole in the ground cover with leaves and sticks and with sharp lances inside. The player must jump to avoid it.
- Extras: wild animals such as scorpions or rattlesnakes.

Nagys machines and traps

Nagys can handle metals easily and they know how to use fire, technology and engines. They have placed some traps in the Maze as well, but these are more modern:

- Falling tree: A tree just cut down by a Nagy, falling on the ground. The player must jump to avoid it.
- Explosion: An explosion on the ground activated remotely by Nagys. The player must jump to avoid it.
- Cannonball: Nagys have thrown a cannonball, the player must slide down to avoid it.
- Fire: Nagys have started a fire, the player must jump to avoid it.
- Buzzsaw Blade: Nagys throw buzzsaw blades against the player. He must slide down to avoid them.
- Metal nails rain: Nagys throw lots of metal nails against the player. He must slide down to survive.
- Metallic dragon: Nagys have built a flying fire spitting dragon. The Nagy controlling the dragon tries to kill the player flying close to him or spitting fire, the player must slide down to avoid the danger.
- Metallic rhino: Nagys have built a metallic rhino. The Nagy controlling it tries to kill the player running over him. The player must jump over it to avoid it.

Cugas / Maze monsters

Cugas will try to kill the player. To avoid that, he must kill the Cuga first or avoid it by jumping over. Killing the Cuga will give the player more points.

Turn left/right

There was a problem with the turning since all the obstacles are noticed by the player with a sound. How could be possible to find a sound for a turning? The answer are Chips. Chips are in every corner to tell the player when to turn. Chips will make a sound, like a little not understandable scream or call. With Unity 3D audio, the player will feel how that sound comes from left or right (while playing with headphones). He will know where to turn.