Universidade da Beira Interior
Departamento de Informática

Nr. 213 - 2021: *2D Platform Game*

Elaborado por:

João Miguel Silva

Orientador:

Doutor Professor Frutuoso Silva

July 8, 2021

Acknowledgement

First I want to thank my grandparents for giving me the opportunity to enroll in a University level of education.They are the sole reason for me to be able to do this, and achieve a higher level of education.
Secondly I want to thank every person I now call a friend to putting up with me and helping me whenever I went to them asking for help.
I want to also thank every faculty member that I encountered along my academic path, for providing me with a rich and knowledgeable education.
I obviously wanna give a special thanks to Professor Doctor Frutuoso Silva,for the amazing coordination of this project throughout the semester and for always being ready to give me the help I needed, despite being in a pandemic environment.
Last but not least I also want to thank myself, for deciding to go through the decision of getting better as a person on a intellectual level but also as a member of society, for this whole experience is one of a kinda.

Contents

List of Figures

List of Tables

List of Code Snippets

Acronyms

UBI	Universidade da Beira Interior
NPC	Non-Player Character
NPCs	Non-Player Characters
CSS	Counter-Strike Source
UI	User Interface
OS	Operating System
SLOBS	Streamlabs OBS
IDE	Integrated Development Environment
HUD	Heads-Up Display
BG	Background
JSON	JavaScript Object Notation
AI	Artificial Intelligence

Chapter

1

Introduction

1.1 Ambit

This report was made within the goal of the Project unit in the course of *Computer Science and Engineering* of Universidade da Beira Interior (UBI).

1.2 Motivation

The main reason behind the choice of this project [4] is the genuine interest in the area of video games. Since I can remember I had always the thought in the back of my mind on "How is this made". Another aspect to take into account is the fact I had never indulged in the actual making of a video game in spite of my interest, so this will also serve as an introduction to what makes a video game come to existence and all the inner working of game development.

1.3 Objectives

The objective of this project is to develop a 2D platform game. The focus of the player is to reach the end of each level as fast as possible while avoiding Non-Player Characters (NPCs) that have intelligent behaviour.

1.4 Document Organization

The document has the following structure:

1. First chapter – **Introduction** – presents the project, the motivation for its choice, the context for it, its objectives and the respective organization of the document.

2. Second chapter – **Inspiration and Research** – describes the ideas behind the game and its implementations, comparing it to similar ones.

3. Third chapter – **Software Engineering** – presents the non functional requirements of the game, and a fluxogram of each level.

4. Fourth chapter – **Technologies Used** – establishes the development environment, enumerates the various technologies used and the function of each one.

5. Fifth chapter – **Development and Implementation** – describes each development step in a more detailed manner.

6. Sixth chapter – **Surveying** – describes how the surveying was done and what was done afterwards.

7. Seventh chapter – **Conclusions and Future Work** – concludes the report, comparing the objectives accomplished with the ones expected and enumerates future implementations to the game.

Chapter

2

Inspiration and Research

2.1 Introduction

In this chapter are mentioned the inspirations on which the game was based as well as comparing them.

2.2 Inspiration

The first aspect of the game which is the name itself, is based on one of the characters from the Looney Tunes world, that is called Roadrunner [5] and makes a sound like *Beep-Beep*, but only after I came up with the name *Meep-Meep* I discovered I was wrong , but since it is all about fun I left the name of the game as *MeepMeep*. Big part of this game is based on simple and already mastered ways of building a *2D Platformer*, such as intuitive controls, quick levels and a increasing skill curve the further a player gets into the game itself. So for the foundation of the game I based myself on childhood games I used to play as a kid, such as **Super Mario** and **Sonic** figure 2.1, which can be compared to this game's design figure 2.2. Other big part of my inspiration came from the video game that I most play nowadays **Counter-Strike Source (CSS)** [6], more specifically a game mode within the game, called *Surf* [7].

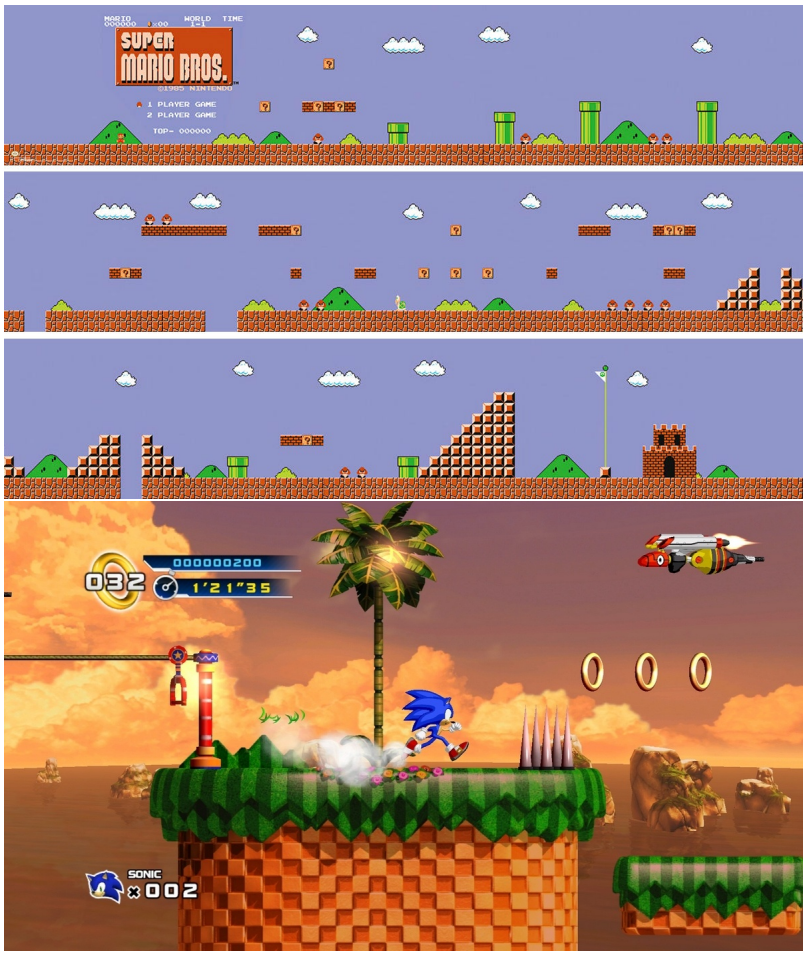


Figure 2.1: Super Mario and Sonic Design examples obtained from [1][2]

Figure 2.2: MeepMeep design example

2.3 Related Concepts

Some concepts included in this game were based on classic 2D Platformers, such as intuitive controls and level-by-level typical game paths.

Other concepts came from the already referred CSS game mode called *Surf*. Such as:

- Time system
- Checkpoint system
- High Score system
- Replay system

2.3.1 Timer / Checkpoints

The player has a timer on each level with the intent of being as fast as possible on completing the level. The figure 2.3 shows how it is represented current time in *MeepMeep* figure 2.3a and figure CSS 2.3b. The figure 2.4 shows how the checkpoints are represented in *MeepMeep* (figure 2.4a) and CSS (figure 2.4b).

(a) MeepMeep Timer (b) Surf Timer

Figure 2.3: Comparison between MeepMeep and Surf - *Timer*

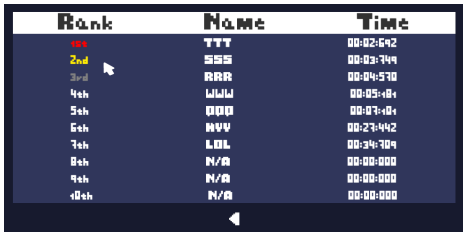
2.3.2 High Scores

The high score system is also similar to Surf concept. Where on each checkpoint the players passes it compares the time with the best time available on each level, presenting with the correct values. The figure 2.5 shows how the ***highs cores*** are shown in *MeepMeep* (figure 2.5a) and CSS (figure 2.5b).

(a) MeepMeep Checkpoint

(b) Surf Checkpoint

Figure 2.4: Comparison between MeepMeep and Surf - *Checkpoints*



(a) MeepMeep High Scores

(b) Surf High Scores

Figure 2.5: Comparison between MeepMeep and Surf - *High Scores*

2.3.3 Replay

The replay system is basically a re-run of the best time on each level, which is used to then, (if the player wants to see how to be faster or compare himself to the best time) replay the best player's time of said level.

2.4 Conclusions

With the indication of the game inspirations and similar concepts, the next chapter will be dedicated to the specification of Software Engineering concepts.

Chapter

3

Software Engineering

3.1 Introduction

In this chapter are settled the non functional requirements of the game and shown a diagram that represents the cycle of each level.

3.2 Non Functional Requirements

1. ***Singleplayer*** **–** It is expected to be a single player game.
2. ***Offline*** **–** The game should not have any online functionality.
3. ***Windows*** **–** The game shall be operational in *Windows*.
4. ***Unity*** **–** The development shall be done in *Unity*.
5. ***C#*** **–** Coding language shall be C#

3.3 Level Cycle Fluxogram

In Unity the behaviour of every single object is controlled by ***Scripts*** or ***Components***, so it is somewhat hard to pinpoint each case, that can and may happen within the game. Unity has a really complete flowchart**[8]** with every single function that gets called during a scene and their respective order. So the following fluxogram 3.1 intends to generalise and simplify all the processes that are embedded in Unity itself and taking in consideration this particular game's scripts and components on each object, with the intent of making it more intuitive to the reader.

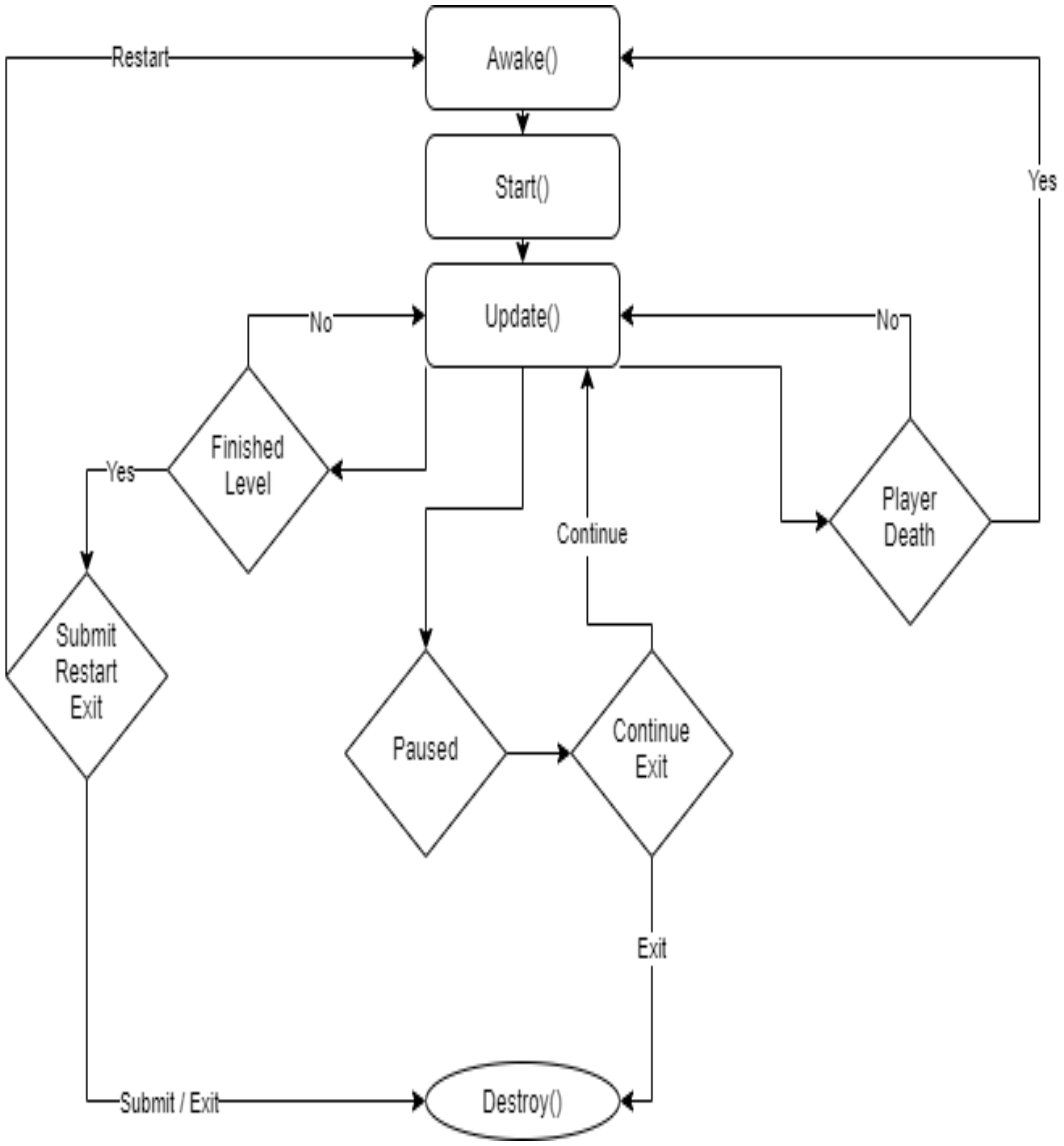


Figure 3.1: Level Cycle Fluxogram

The job of each function is the following:

1. ***Awake()*** – This function gets called when a scene starts (once for each object in the scene). It is where all the objects are initialized.

2. ***Start()*** – This function is called before the first frame update and takes place after every object as been correctly initialized.

3. ***Update()*** – This function gets on every frame.

4. ***Destroy()*** – Occurs when a scene ends.

Now for the possible cases that may happen on each level, this is how it may go:

1. ***Finished Level*** – If the player reach the end of a level it goes through this case and gets redirected to the next one, which is *Submit* / *Restart* / *Exit*.

2. ***Submit / Restart / Exit*** – In this case the player gets prompted with User Interface (UI) depending if it the player got a good time or not on the level, which gives the player the 3 options in this case.

   In the outcome of being a TOP10 time the player maybe choose between Submit and Exit.
   Submit will lead to the end of the scene and so will ***Exit***.

   In the outcome of not being a TOP10 the player may choose between Restart and Exit.
   Restart will lead to Awake() function of the scene and ***Exit*** will lead to the end of the scene.

3. ***Paused*** – This function gets on every frame. This state is achieved everytime the player pauses the game. It has two possible outcomes, Continue and Exit. *Continue* will lead to the continuation of the scene and *Exit* will lead to the end of the scene.

4. ***Player Death*** – Occurs when a the player dies and lead to the Awake() function of the scene.

3.4 Conclusions

Concluded this chapter, the next chapter shall be dedicated to the Technology used and their respective assigned job.

Chapter

4

Used Technologies

4.1 Introduction

In this chapter will be mentioned the environment in which the game was developed as well as what technologies were used.

4.2 Development Environment

This project was developed in a Desktop with the following components:

1. ***Processor*** – AMD Ryzen 5 3600X 6-Core Processor 4.25 GHz
2. ***Graphics Card*** – Nvidia GeForce GTX 1060 6GB
3. ***RAM*** – 16 GB
4. ***Operating System (OS)*** – Windows 10 (64-bit operating system, x64-based processor)

4.3 Used Technologies

1. ***Unity*** – The game was developed using Unity's Engine [9] on version 2020.3.4f1. It was used due to its ease of bringing the developers an easy and intuitive way of creating games and its immense online support but above all , it is free.
2. ***Visual Studio Code*** [10] – This was the chosen Integrated Development Environment (IDE) [11] of choice since it is my most used one so it will provide a more controlled and safe environment to my own self.

3. ***C#*** [12] – Since the game was developed using ***Unity Engine*** the used language was C#, due to its necessary need of use when using said engine.
4. ***Paint*** [13] – Used to change the color of certain assets used.
5. ***Streamlabs OBS (SLOBS)*** [14] – Used on the recording of gameplay for the creation of a trailer.
6. ***LateX*** [15] – This report was done using LateX, as the standard of UBI states.
7. ***Overleaf*** [16] – Online application used to compile and managed ***.tex*** files.

4.4 Conclusions

With the enumerated technologies it was possible to develop both game and report. In the next chapter will be presented every functionality implemented as well as its details.

Chapter

5

Development and Implementations

5.1 Introduction

In this chapter will described the different development stages and the functionalities used within the game.

5.2 Development Stages

1. ***Research*** – Research was done on the required game genres ***2D Platformer*** [17] and ***Speedrun*** [18].

2. ***Tutorial*** – Done parts of a tutorial [19] to understand the use of the basic components of ***Unity's Engine*** [9].

3. ***Planning*** – Deciding on what the core gameplay of the game would involve.

4. ***Assets*** – Research on available and free assets [20] to use.

5. ***Implementation*** – Implementation of the previously planned functionalities.

6. ***Debugging*** – Testing of all implemented functionalities.

7. ***Surveying*** – A Google Form was used to gather information and opinions on the multiple implemented functionalities.

8. ***Evaluation*** – Assessment of the current state of the game.

9. ***Trailer*** – Creation of a small and short trailer that quickly showcase what the game is about.

10. ***Report*** – Written a report using the standard templates of UBI.

5.3 Implementation

5.3.1 Unity Editor

As previously stated the making of this game was made possible with ***Unity's Engine***. This engine allows the developer to abstract itself from complicated physics and mathematical issues.
The figure 5.1 show the basic elements present in the editor.

1. ***Project Window*** – Lists every game directories and files.

2. ***Hierarchy Window*** – Displays all the objects present in the Game Window.

3. ***Inspector Window*** – Lists every object's components.

4. ***Scene Window*** – Window where the objects are manipulated to the developers desire. It is where the actual game takes form in the visually manner.

5. ***Toolbar*** – This toolbar (above the Hierarchy Window) gives quick access to multiple object editing tools.

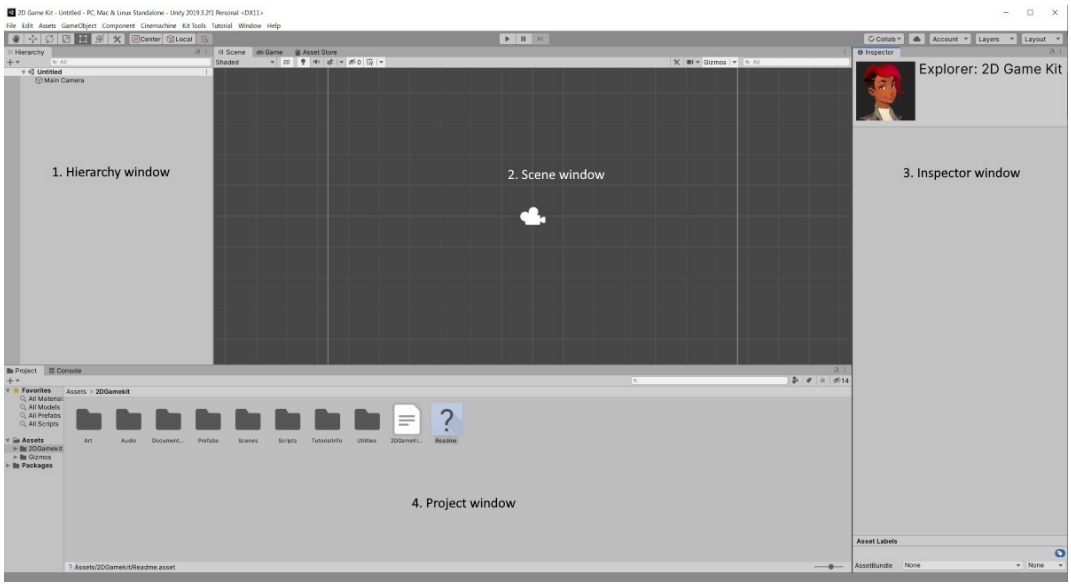


Figure 5.1: Unity Editor – obtained from [3]

5.3.2 Components

The main thing about Unity's Editor is that provides a simple way of implementing hard concepts with a few clicks. This is done via ***Components***, which in a very quick summarized way is nothing but a attribute given to an object, that can have multiple behaviours. The following enumeration states which components were used on this game.

1. ***Transform*** [21] – Every objects has this component. Controls the coordinates, scale and size of the object
2. ***Camera*** [22] – Defines what and how to show the objects in the scene.
3. ***Sprite Renderer*** [23] – Image that represents the object (what the player sees).
4. ***TileMap*** [24] – Allows level design via ***Tiles*** [25].
5. ***RigidBody2D*** [26] – Make use of the Physics Engine [27], which basically meeans that the sprite that represents the objects gets put through the Physics Engine.
6. ***Collider2D*** [28] – Allow the Physics Engine to identify what shall and shall not be taken in consideration on performing calculation within the Physics Engine itself.

7. ***Animator*** [29] – Has a mechanism that allows the control of the animation system.
8. ***Event System*** [30] – Controls events based on every input on the scene.
9. ***Canvas*** [31] – Represents the abstract space where the UI gets represented.
10. ***Button*** [32] – Used to trigger an event on the ***Event System***.
11. ***Text*** [33] – Used to display font on the screen.
12. ***Audio Listener*** [34] – Used to received any input from any ***Audio Source***.
13. ***Audio Source*** [35] – Representation of audio sources in the 3D world.
14. ***C# Scripts*** [36] – This component is a special one, which allows the developer to control every other component on an object via code.

Besides this components, there were also used functionalities such as ***Tags*** [37] , ***Layers*** [38] (which allow a more precise control over the objects within the scene and how they are shown to the player) and ***Prefabs*** [39] which allow the developer to create a object at any given point in the game via Script. For example, instead of dragging and dropping 50 objects that represent a box to the Hierarchy Window, it is possible to Instantiate [40] the same object how many times it is needed using code.

5.3.3 Core Gameplay

The main goal boils down to performing quick and precise movements and reach the end of a level the fastest while avoiding enemies and/or possible deathly objects. Something that is also worth explaining is the fact that the principle behind every functionality created and used in this game was made so it has personalising capabilities. This means that each level can have different implementations from the player movement to the way the NPCs function. A quick example , it is possible to set how many jumps a player can perform (by default it is 2).

5.3.4 Managers

The following 4 subsections speak of a special kinda of object named ***Singleton*** [41]. This mechanism allows only one instance of said object in the scene, which contains multiple variables and methods that control how the game

works and flows.
Afterwards a ***Static Class*** [42] that is called ***Save Manager*** will also be explained.

5.3.4.1 Game Manager

This script does the following [5.1][5.2][5.3]:

- Contains the 3 classes of the variables that construct the data in the game. Which are the following:

```
public class DB_TABLES
{
    public List<HighscoreTable> list = new List<
        HighscoreTable>();
    public string saveName;
}
```

Code Snippet 5.1: DB_TABLES class

```
public class HighscoreTable
{
    public List<HighscoreEntry> list = new List<
        HighscoreEntry>();
}
```

Code Snippet 5.2: HighScoreTable class

```
public class HighscoreEntry
{
    public string name;
    public float time;
    public float CP_1;
    public float CP_2;
    public float CP_3;
}
```

Code Snippet 5.3: HighScoreEntry class

The structure of these shown classes are represented in figure 5.2

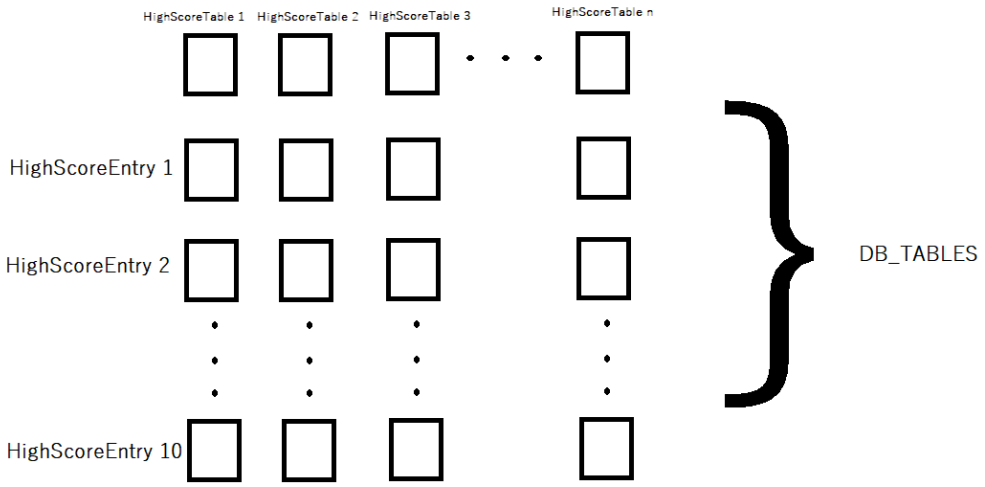


Figure 5.2: Rudimentary example of data structure in the game

- Create, Modify and Delete the list that contains all of data of each level on the game.
- Load and Save the variables that contain all the game's data.
- Load and Save PlayerPrefs [43].
- Persists data across the scenes.

5.3.4.2 Sound Manager

The way I designed the sound manager is based on the foresight of future implementations, such has expanding the audio system within the game. The audio manager is responsible for playing and stopping sounds inside scenes. The structure of this manager is based on having multiple layers of audio throughout the game development and gameplay. To better explain this, the following images (figures 5.3 and 5.4) show how it is structured and what it contains.

The way it works is quite intuitive. The game can have as many containers as one wishes, in this particular case I went with 2 containers named ***SFX*** and ***Music***. Then in each container there can be any number of Audio Clips that has the following structure:

```
public class SoundClip
{
    public AudioClip clip;
```

```
    public string name;
    [Range(0,1)]
    public float volume;
    [Range(.1f,3f)]
    public float pitch;
    public bool loop;

    [HideInInspector]
    public AudioSource source;
}
```

Code Snippet 5.4: SoundClip class

I can also set some different attributes to those Audio Clips, such as *"name"*, *"volume"*, *"pitch"*, *"loop"*, that do exactly what the name means. When the game gets initialized the Sound Manager gets initialized as well, which is responsible to create those added Audio Clips as an Audio Source Component on the respective Container. One example is figure 5.5.

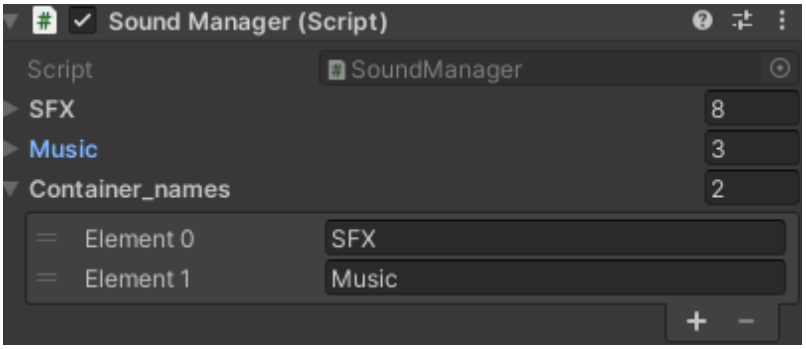


Figure 5.3: MeepMeep's Containers used

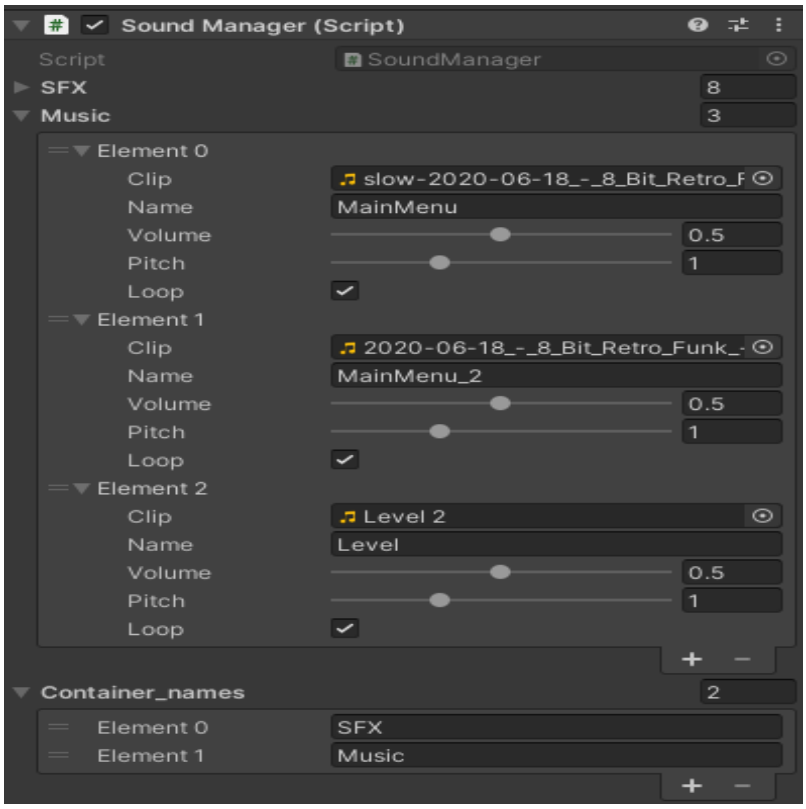


Figure 5.4: MeepMeep's Audio Clips

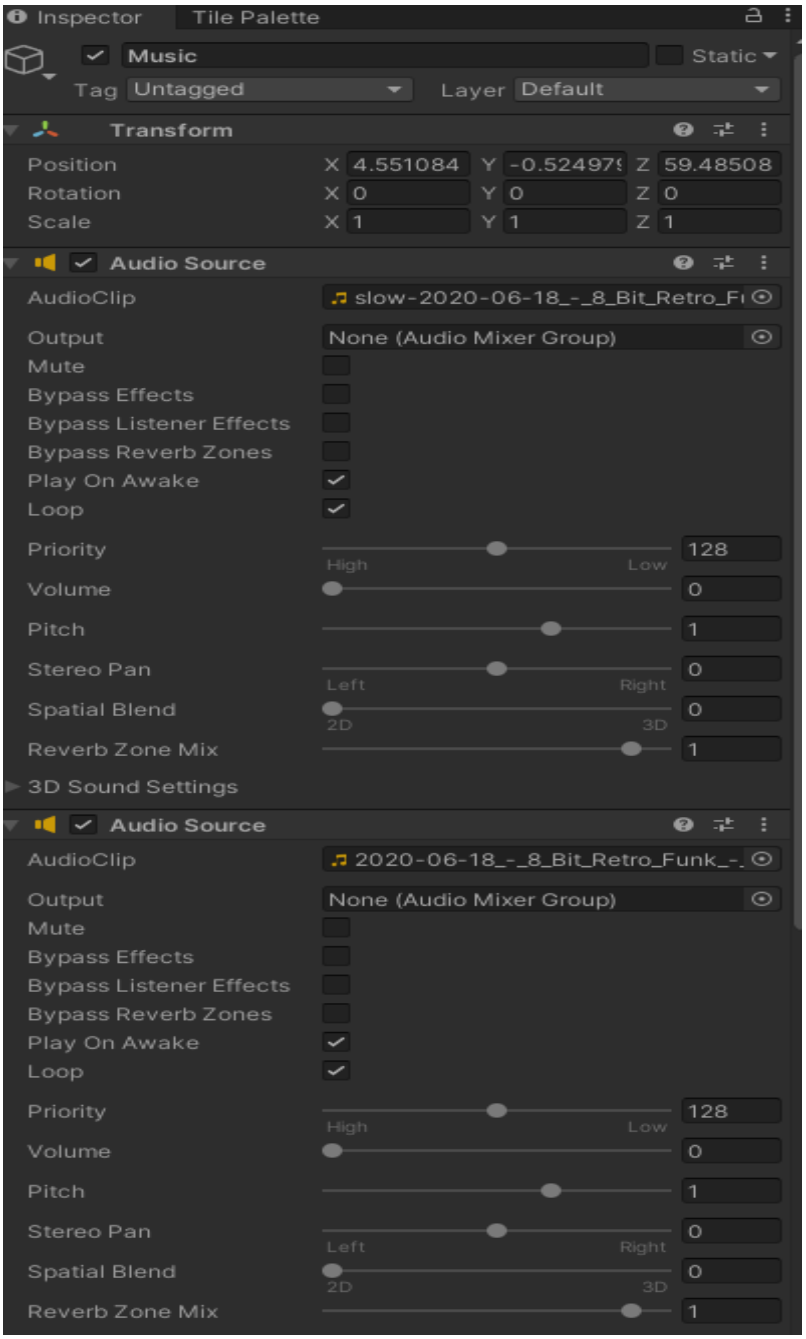


Figure 5.5: MeepMeep's Music Container

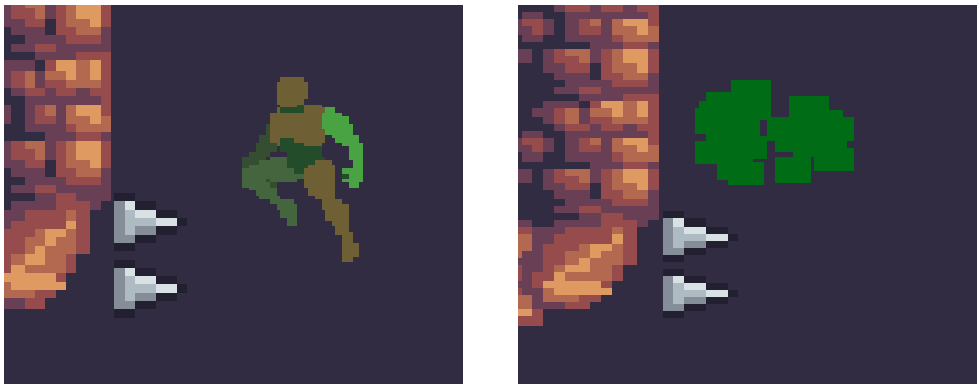
5.3.4.3 Click Manager

This script is really straight forward on what it is responsible for. It is responsible for every Action performed or to be performed when using the UI that is

presented to the player.

5.3.4.4 Particle Manager

This script is really straight forward on what it is responsible for. It is responsible for every Instantiate a *Particle System* [44] when the player dies as seen in figure 5.6a and figure 5.6b.



(a) Before Death (b) After Death

Figure 5.6: Particle System Activation

5.3.4.5 Save Manager

This class has static variables and static methods, that serve as an API [45]. This class contains:

1. Variables that contain the values that lead to the directories where the game's data will be stored at.

2. Methods that allow to Load, Save, Delete and Move the game's data.

This class is also responsible to take the screenshots that will later compose the ***Replay*** [5.3.12] used in the ***Replay System***.

The file extension chosen to store the game's data was JavaScript Object Notation (JSON) [46], for its simple **ease of use**, **online support** and the fact this game is ***offline*** my biggest concerns were not data *security* or *integrity*

5.3.5 Level Design

5.3.5.1 Levels

There are 3 different levels in this game. The goal was to make it gradually increase in difficulty throughout the game, where Level 1 (figure 5.7) would be more of an introductory level and Level 2 (figure 5.8) and Level 3 (figure 5.9) would have its own tweaks.

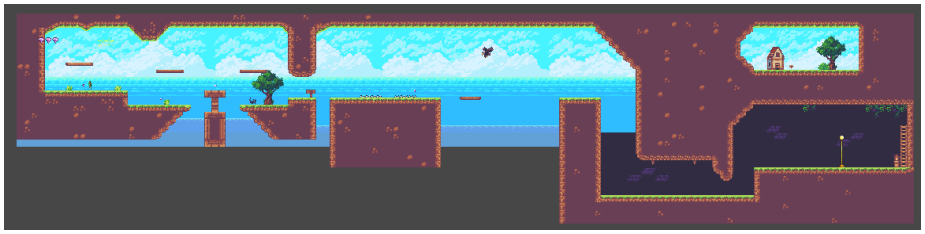


Figure 5.7: Level 1 Design

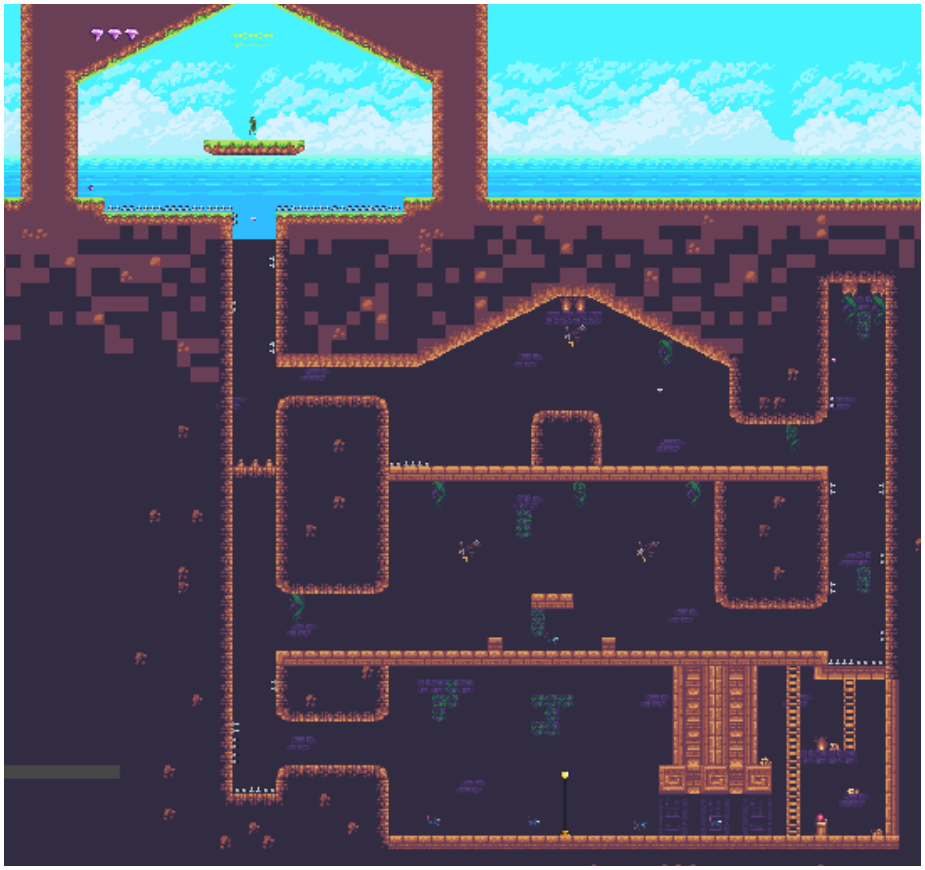


Figure 5.8: Level 2 Design

Figure 5.9: Level 3 Design

5.3.5.2 Tilemaps

Each level is composed by different Tilemaps and given a specific Layer. The tilemaps are the following:

1. ***Floor*** – This tilemap represents the floors of each levels, it contains a collider with will be used on the Physics Engine and interact with all the other objects that also contain a collider. For better performance this object [47] also contains a component called ***Composite Collider***. [48], that aims to transform multiple polygons [49] to a sigular one.

2. ***Wall*** – This tilemap is the same as the tilemap that represents the floor, just applied to a different layer.

3. ***Ceiling*** – This tilemap is the same as the tilemap that represents the floor, just applied to a different layer.

4. ***Platforms*** – This tilemap is the same as the tilemap that represents the floor, just applied to a different layer.

5. ***Water*** – This tilemap is the same as the tilemap that represents the floor, just applied to a different layer. This tilemap will also trigger the players dead, upon collision.

6. ***Props*** – This tilemap has no colliders it serves as a simple decorative layer.

7. ***Background (BG)*** – This tilemap has no colliders it serves as a simple BG layer.

5.3.6 Camera

The camera used is an component [50] that Unity Editor has called ***Cinemachine*** [51] which allows for a really intricate and elaborate camera work with simple UI. In this case I only used it for simple fact it has a property that permits the camera to follow the player throughout the level.

5.3.7 Main Character

The behaviour of the main character is spread along 3 different scripts. They are the following:

- ***MeepController***
- ***PlayerMovement***
- ***DashCounter***

Another component used is the ***Animator*** which allows to (in my case) implement Assets from the Unity Store [52] pre-built animations when using an Animator Controller [53] and a State Machines. A ***State Machine*** is nothing more than a set of State Machine Behaviours [54]. What the State Machine allows the developer to do is create a set of states from which the players transitions from or to, depending on the value of said triggers responsible for said transition (which are controlled via code as seen in figure 5.11, which in this particular case it is setting the variable responsible for transition to the state of jump to true). The figure 5.10 shows the State Machine used in this game, for the player's animations.

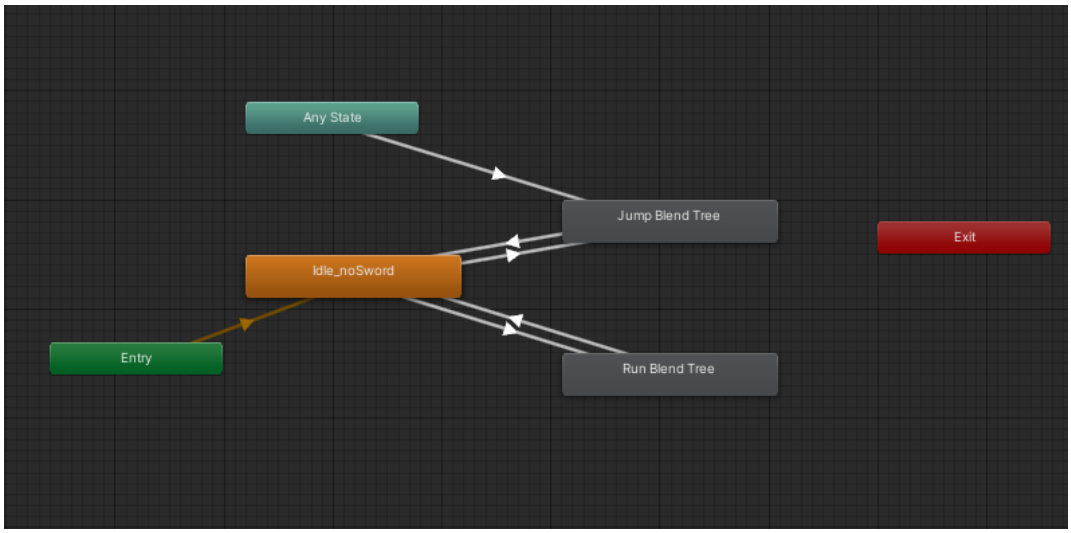


Figure 5.10: Player's Animation's State Machine

```
player_animator.SetBool("isJumping", true);
```

Figure 5.11: Jump Animation Trigger

5.3.7.1 MeepController

This script acts as the brains of the player. It is responsible for:

- Assigning every component to variables so it is possible to control the player via code.
- Detecting players death.
- Triggering Pause/Unpause and Finish Menu.
- Saving 3 floats that represent the current players CheckPoint time in the level.
- Stoping the timer of the player finishes the level.
- Play the assigned Music.
- Set the values on the variables that triggers the different states on the animator tree of the player.

5.3.7.2 PlayerMovement

This script is responsible for receiving the input from the player and process it accordingly.

The Controls of the player are the following:

1. ***A/D*** – Left and Right Movement.

2. ***S*** – Quick Fall.

3. ***Spacebar*** – Jump / Wall Jump.

4. ***LeftShift*** – Dash.

5. ***R*** – Restart Level.

6. ***P*** – Pause Level.

Before enumerating what are the functionalities of this script I wanna mention a few particularities regarding both ***Wall Jumping*** and ***Dashing***.

The ***Wall Jumping*** movement is the same as if the player was on the ground, what this means its that the player will easily treat the Wall Jump and the "Ground Jump" as the same, the outcome when jumping from the ground or the wall will be the same. There is also Wall Sliding present in the game, which is done by pressing the key the corresponds to the direction where the wall is. In a simple example, if the player has a wall to his left, he must press the ***A*** key in order to slide along the wall. Another coding factor taking in consideration is that the player can only slide if it is also not on the ground, which is controlled with a variable, on whether it is or not true.

Regarding ***Dashing*** , a player can only dash if it is moving along the x-axis (Left or Right) and if they have available dashes on the UI that represents the current value of the dashes as seen in figure 5.13. The opposite occurs when there are no Dashes available as seen in figure 5.12.

Figure 5.12: Has no Dashes Available (Top Left Corner)

Figure 5.13: Has 1 Dash Available (Top Left Corner)

The ***Player Movement*** script is responsible for:

1. Checking whether the player is Grounded or not.
2. Have direction of the player's movement on both axis.
3. Set the value of multiple variables such as:
    - Movement Speed
    - Jumping Force
    - Falling Force
    - Dash Force
    - Wall Slide Speed

4. Play the sounds when the player dies, jumps, dashes and quickfall's.
5. Calculations on whether the player can Jump, Dash, WallJump, WallSlide and QuickFall.
6. Start the timer when the player starts to move for the first time when the level begins.
7. Flip the sprite depending the movement direction of the player.
8. Set the values on the variables that triggers the different states on the animator tree of the player.

5.3.7.3 DashCounter

This script is responsible for:

1. Having the value of the number of dashes a player when they are playing.
2. Setting the UI that represents the value of the current dashes a player has (figure 5.14).

Figure 5.14: Dash

5.3.7.4 Player Collisions

All the player Collisions are done abstractly with the use of the ***RigiBody2D***, ***Collider2D*** with Unity's Physics Engine.

5.3.8 Enemies

This sub chapter will indulge on how all the NPCs work. I also want to make it known that the enemies were implemented and created with the intuit of being possible to make iterations of them where there can be different behavioral NPCs withing the same category.

5.3.8.1 Eagle

Figure 5.15: Eagle

This Non-Player Character (NPC) (figure 5.15) has 2 principal behaviours that can be tweaked and/or activated:

1. ***Shoot Bullets***

2. ***Follow The Player***

A simple and forward way of explaining how this NPC behaves is the following: it has an AoE [55] detection mechanism that acts along two Rays [56], as seen in figure 5.16. If the player is withing the circular AoE detection range **and** at least one of the Rays is reaching the player the NPC can ***Shoot*** and ***Move*** (or not) towards the player's position.

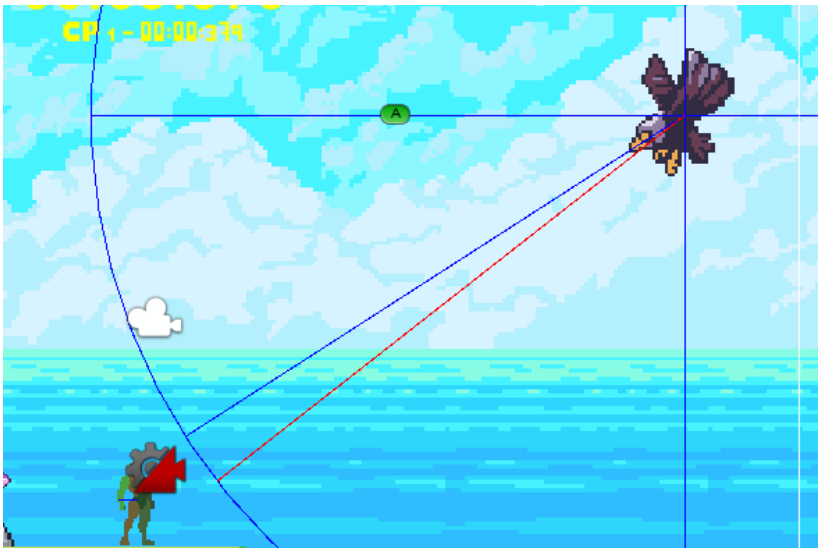


Figure 5.16: Eagle Detection

5.3.8.2 Rat

Figure 5.17: Rat

This NPC (figure 5.17) has 2 principal behaviours that can be tweaked and/or activated:

1. ***Patrol***
2. ***Leap Towards the Player***

The way this behaviours are triggered are similar to the previous NPC. It has an AoE where if the player is withing said range **and** the rat is facing the

player it performs a ***Leap Attack*** towards the player. The other behaviour (***Patrol***) , is created using "Physcis2D.OverlapingCircle" [57], 2 circles in this case (1 shown as blue and 1 shown as red in figure 5.18) which are similar to a RayCast, the difference is that it provides a trigger-like mechanism where if any collider is within said circle are (like the ones shown in figure 5.18) a trigger is thrown.

In this case the blue circle is responsible to detect if the NPC has reached the limit of a ground layer in which case it flips its direction and starts patrolling the other way. The red circle is responsible to detect if it collided with any tile in the "Wall" or "Ground" Layer, in which case also performs a flip of direction.

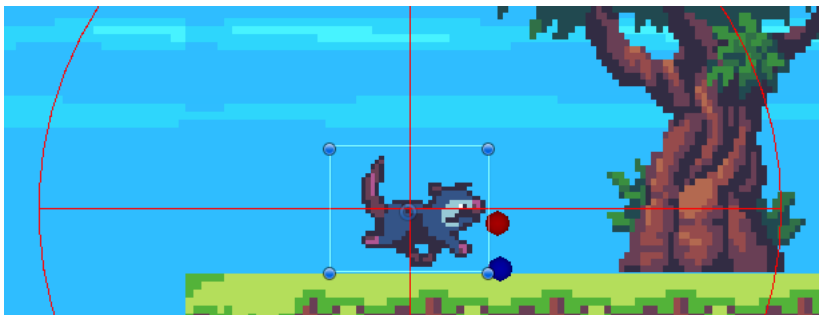


Figure 5.18: Rat Detection

5.3.8.3 Spikes

Figure 5.19: Spikes

This Entity (figure 5.19) is very simple. It has a *Collider* component that if collides with the players it kills him.

5.3.8.4 Platforms

There are 2 types of platforms:

1. Moving Platforms

2. Static Platforms (via tilemap)

The ***Moving Platforms*** have two main components:

1. ***Collider*** – integrates this object within the calculations performed by the Unity Physics Engine.

2. ***Script*** – controls the behaviour of the platform.

The state of moving platform is based on a simple iteration along an list of coordinates. Where the platform moves from one position to another depending the order of said positions on the list as seen in figure 5.20, in this example there are 2 positions.
The ***Static Platforms*** can be achieved with different mechanisms :

1. ***Tilemap*** – this way the platform in figure 5.21 is created withing the tilemap that corresponds to the layer of ***"Platform"***.

2. ***Script*** – via the previously mentioned type of platformed (***Moving Platform***), but in this case in the script inside the Moving Platform I can set the velocity to 0, making it ***Static***.

Figure 5.20: Moving Platform

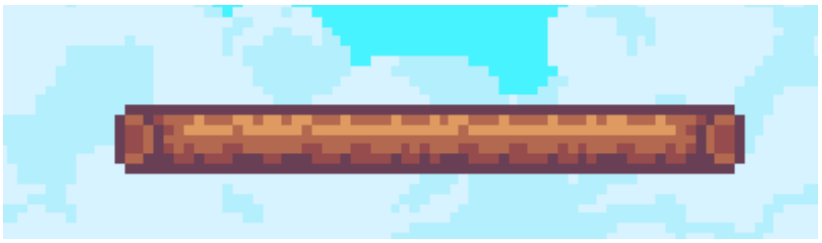


Figure 5.21: Static Platform

5.3.9 User Interface

The next sub-sections will be related to the UI, how it looks and how it works.

5.3.9.1 Main Menu

The Scene in figure 5.22 contains all the UI elements (***Buttons***) that allows the player to access the rest of the game's content.

Figure 5.22: Main Menu

5.3.9.2 Heads-Up Display

The Heads-Up Display (HUD) is made of a ***Canvas*** where there the elements such as the ***Dash Counter*** (top left corner) and ***Timer*** (top center) are all controlled via ***Scripts***, where (to be simply put) the UI just display the value of the values referenced to them.

Figure 5.23: HUD

5.3.9.3 Checkpoints

The checkpoint system is composed by ***3*** objects, each one of them correspond to a different checkpoint number, in this case the system in use has ***3 checkpoints***.
A checkpoint (figure 5.24) is nothing but an object that contains a ***Collider*** and a ***Script*** that contains a reference to the UI responsible for display a certain checkpoint's time in the HUD and the number of said checkpoint.

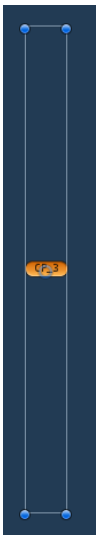


Figure 5.24: Checkpoint

5.3.9.4 Finish Pole

This object (figure 5.25) is responsible for triggering the correct UI to the player, when the player finishes a level, depending on whether or not the player's time was in the ***TOP 10***.

The ***Top 10*** UI in figure 5.26 asks the player for a string, that represents the player, which will be added to the ***High Scores*** list of the level. It allows the player to ***Exit*** to the ***MainMenu*** and shows the player what time they got.

The ***Not Top 10*** UI in figure 5.27 simply shows the player the time they got and allow the player to either ***Restart*** the level or exit to the ***MainMenu***.

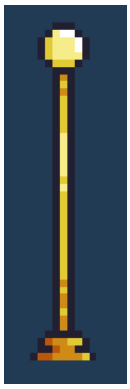


Figure 5.25: Finish Pole

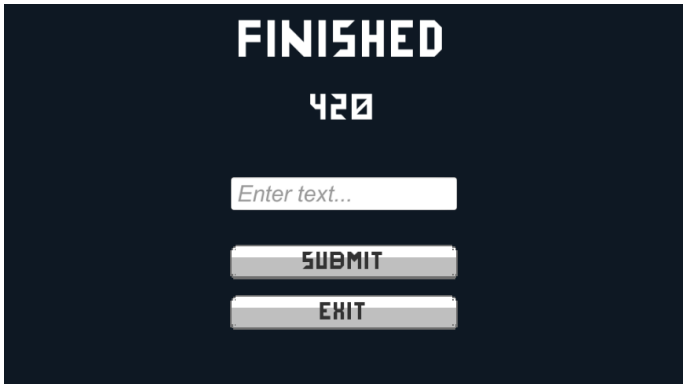


Figure 5.26: Top 10

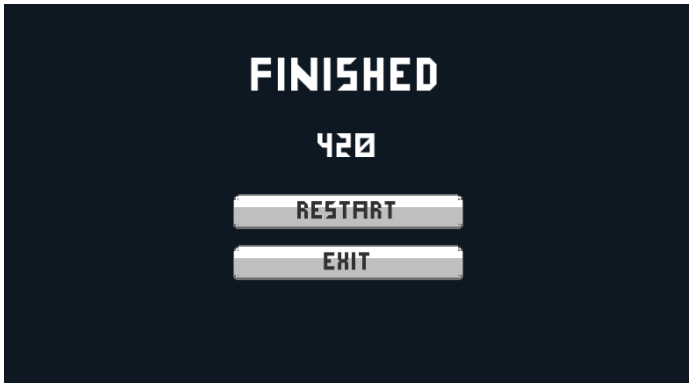


Figure 5.27: Not Top 10

5.3.9.5 Pause Menu

This UI is presented to the user every time they paused the game, in which allows the player to either ***Continue*** playing the level or exit to the ***MainMenu***.

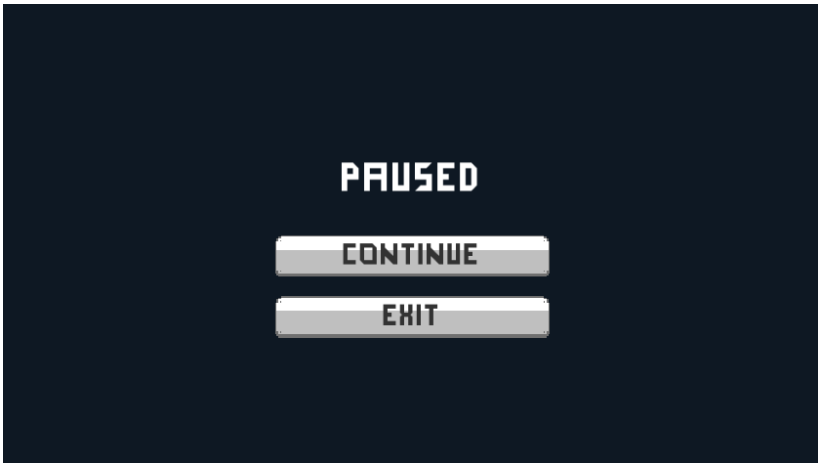


Figure 5.28: Paused Menu

5.3.10 Selector Menu

This scene in figure 5.29 is shown every time the player either accesses the following ***Buttons*** when in the ***MainMenu*** :

1. ***Play***

2. ***High Scores***

3. ***Replay***

Figure 5.29: Selector Menu

5.3.11 High Scores

This scene in figure 5.30 shows the player what are the ***Top 10*** times of the selected level.
The way it works is based on the functionality that Unity has called ***Prefab***, where for each time on the ***High Score Table*** list of the selected level, a ***Prefab*** (that represents a single row) gets , and then given the values of the correspondent ***Rank***, ***Player Name*** and ***Time***.

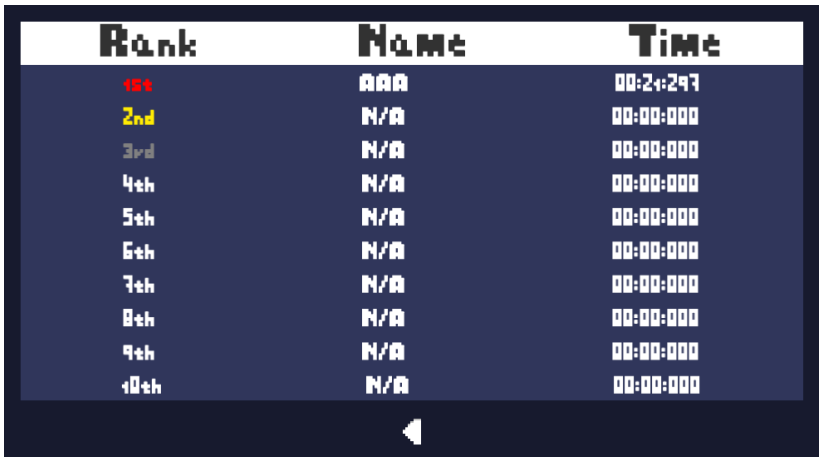


Figure 5.30: High Score Table

5.3.12 Replays

For commodity in further explanations the word ***Replay*** is referring to the group of screenshots taken on each player's attempt.

5.3.12.1 Making of Replay

The replay system is based on the functionality that Unity Engine has of being able to capture screenshots during the game. Each screenshot is taken on each frame and it is given a frame number which is unique and gets saved in the respective order in a ***Temp*** folder.

Each level has its own folder that contains the ***Replay*** of the best time of the respective level.

When the player finishes a level the time gets compared to the ***1st*** time in the respective level. If the time achieved is faster then the already ***1st*** time in the respective level, the ***Replay*** gets replaced by the newer one which is faster. Otherwise (when it is not faster) it is simply performed a clean up of the ***Temp*** folder that contains the most recent replay.

5.3.12.2 Showing Replay

The way a ***Replay*** is played again, is displaying every screenshot taken each frame at the same rate that it was taken, Unity's Engine allows this via a function called ***FixedUpdate()*** [58] and using the same ***DeltaTime*** [59] between frames.

The replays are accessible via the ***MainMenu*** button ***"Replays"***, which is than presented by a replay selector as previously mentioned like figure 5.29.

If there is an existing ***Replay*** of the selected level it gets played and when it finishes it displays an UI as seen in figure 5.31 that allows the player to either watch the replay again or exit onto the ***MainMenu***.

If there is not ***Replay*** available the user gets prompted with UI as seen in figure 5.32 that indicates there are no available replays and redirects the player to the replay selector as seen in figure 5.29.

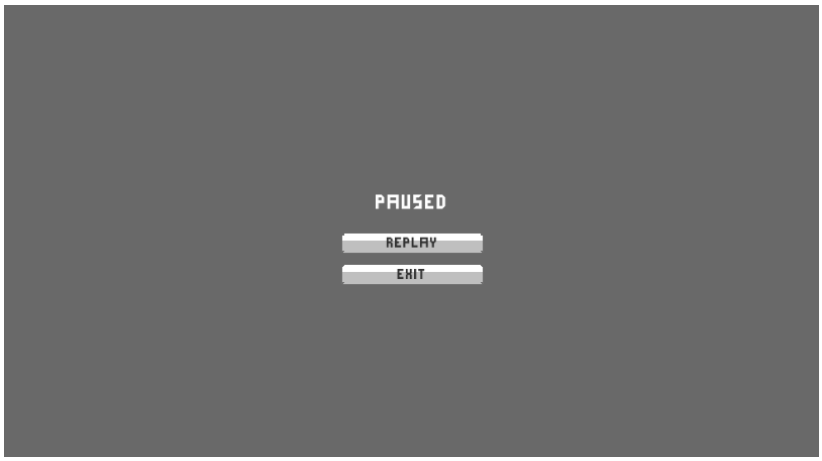


Figure 5.31: UI Finished Replay

Figure 5.32: No Replay

5.3.13 Instructions

This is a simple UI that show the player what are the controls of the game as seen in figure 5.33.

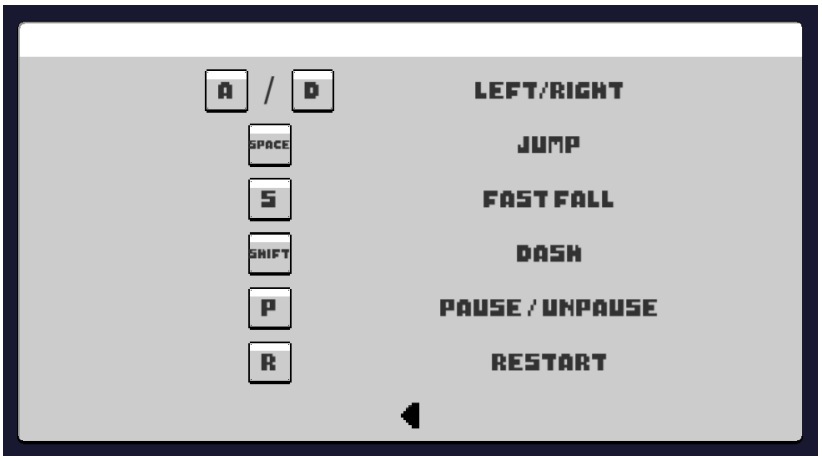


Figure 5.33: Instructions

5.3.14 Options

This Scene has UI that allow the user to :

1. Control Music Volume
2. Control SFX Volume
3. Control Quality
4. Set Fullscreen
5. Set Resolution
6. Enable Replay System

Each option is then saved using ***Player Prefs***, which get saved when altered and loaded at the beginning of each ***Session***.

A ***Session*** is what defines the continuous playtime of a player which starts when the player opens up the game and end when the player closes the game.

Figure 5.34: Options

5.4 Licensing

Every ***Asset*** used in the making of this game was obtained from the ***Unity Asset Store*** with the duly License that grants the free use of said assets.

The only resource used external to the ***Unity Asset Store*** was the sound file used in the making of the ***Trailer***, which requires me to explicitly refer the License [60] and state that there were not made any changes to the default file.

5.5 Conclusions

The next chapter will show the results obtained from ***Surveying***.

Chapter

6

Surveying

6.1 Introduction

This chapter shows how the ***Surveying*** was done, the results obtained and what was done afterwards.

6.2 Data

This ***Surveying*** was prepared through ***Google Forms***, regarding the design, intuitiveness and functionalities that the game presents.The *form* was sent to people around the age of 20-24 as well as people that are considered avid *gamers* and have a bunch of experience playing video games, with a total of ***4 responses***. Each question had a set of possible answers that give concise input towards the game. The table [6.1] shows the given questions.

Questions
Did you enjoy playing it?
Were the Controls intuitive?
Were the enemies hard to deal with?
Was the graphical art used good?
Since there is a High Score system, do you feel it makes you be more competitive or would you still just play the game casually?
Since there is Replay system (which records the best time of each level), is the lag it causes worth the trouble of having it on? (THIS FEATURE CAN BE TURNED OFF OR ON)

Table 6.1: *Google form* questions

6.2.1 Results

6.2.1.1 Question 1

On this question the unanimous answer was "***Yes,ofc***" with the answer "***Nope***" not receiving any votes as seen in figure 6.1.

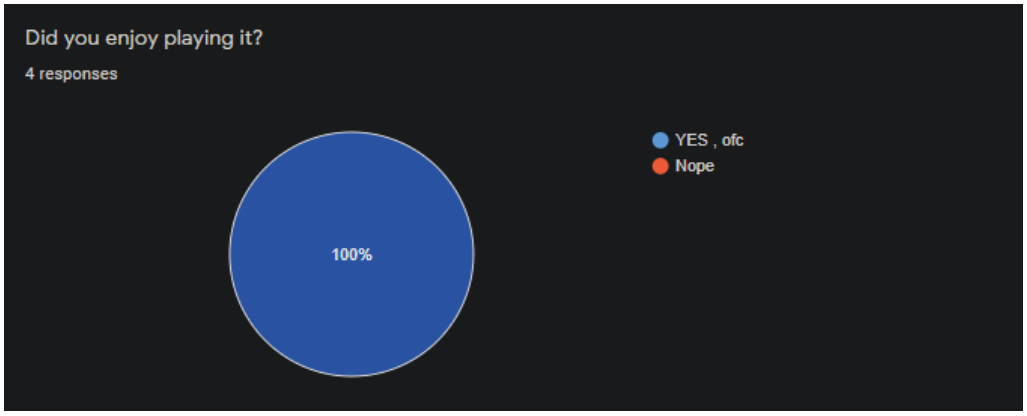


Figure 6.1: Question 1

6.2.1.2 Question 2

On this question the unanimous answer was "***Yes***" with the answer "***No***" not receiving any votes as seen in figure 6.2.

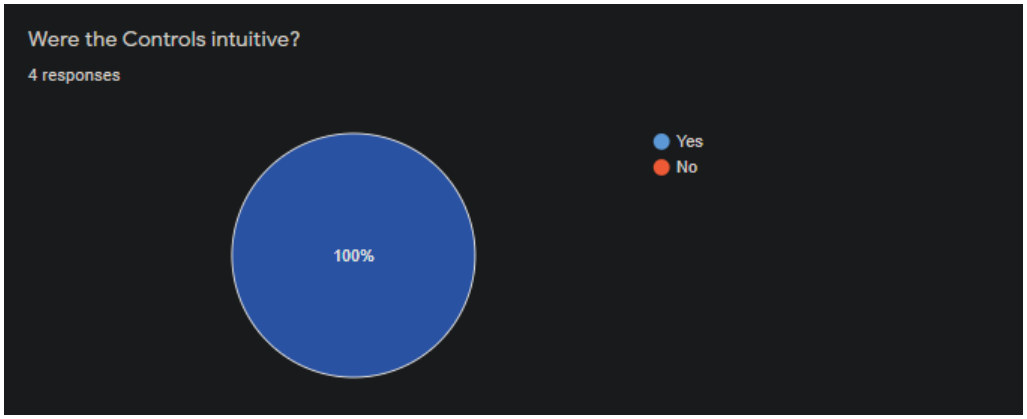


Figure 6.2: Question 2

6.2.1.3 Question 3

On this question both answers "***Not really***" and "***At first, then it got easier***" received 50% of the votes each with the answer "***Yes***" not receiving any votes

as seen in figure 6.3.

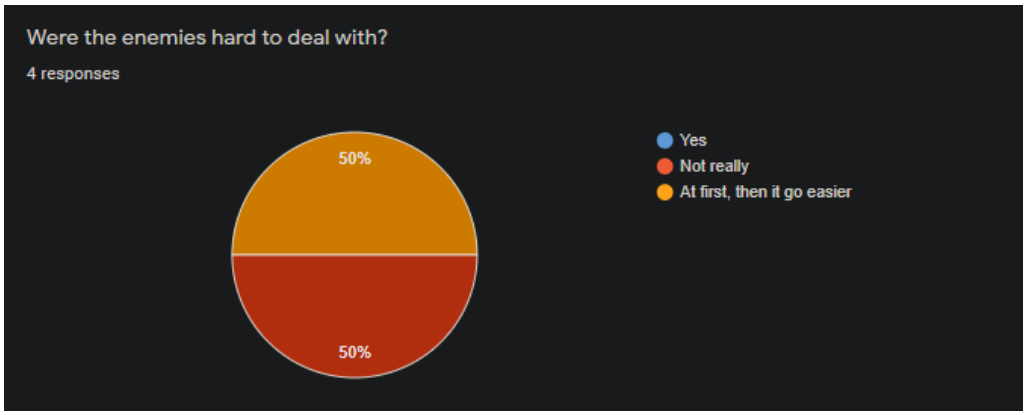


Figure 6.3: Question 3

6.2.1.4 Question 4

On this question the unanimous answer was "***Yes***" with bith answers "***No, not my style***" and "***Not my thing but i liked it***" not receiving any votes as seen in figure 6.4.

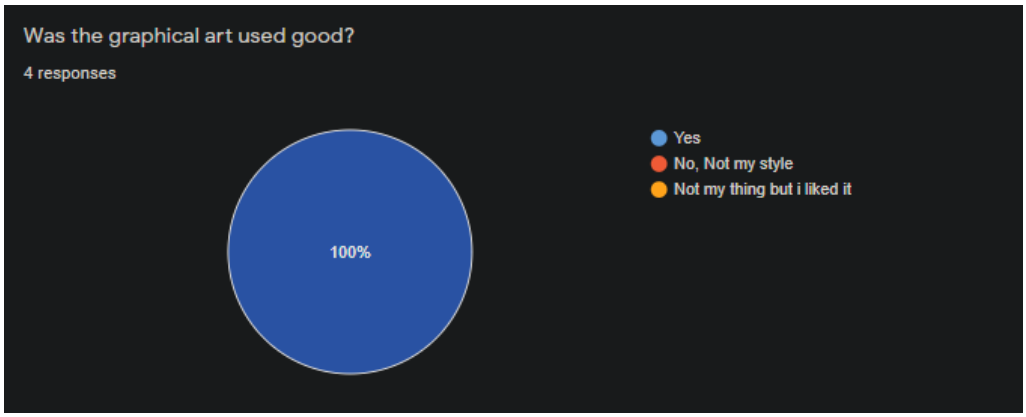


Figure 6.4: Question 4

6.2.1.5 Question 5

On this question the answer "***Makes me more competitive***" got 75% of the votes and "***I am not that competitive***" got 25% as seen in figure 6.5.

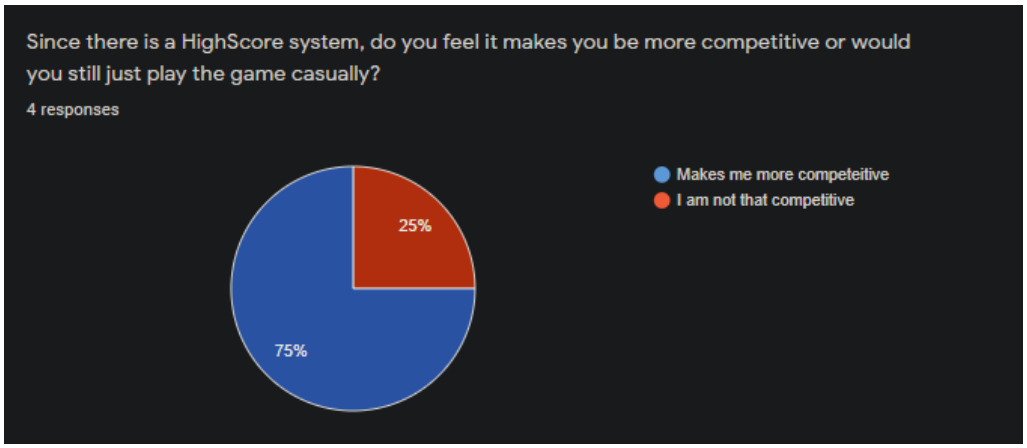


Figure 6.5: Question 5

6.2.1.6 Question 6

On this question both answers "***I dont play with replay ON***" and "***It is kinda bad but still playable***" received 50% of the votes, with the answer "***Its unplayable***" not receiving any of the votes as seen in figure 6.6.

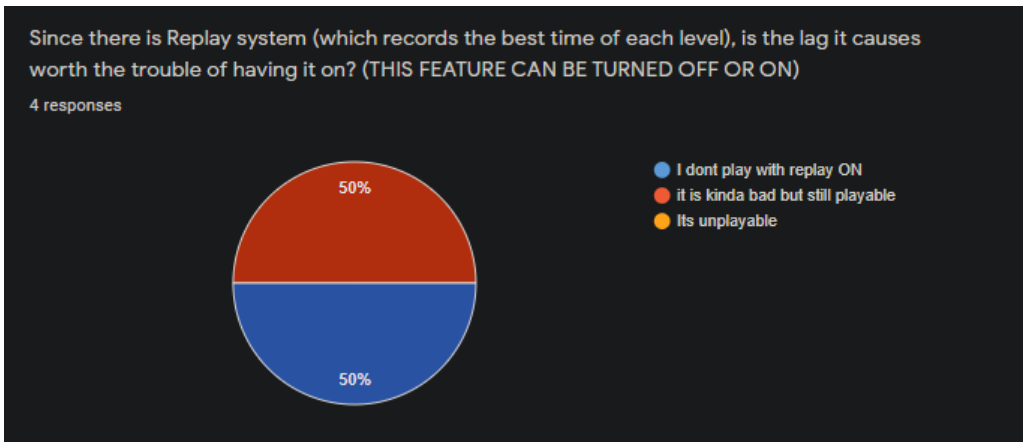


Figure 6.6: Question 6

6.3 Afterwork

Based on the results of the *form* the only alteration made was taking in account the difficulty of each level and making them more accessible to the player.

6.4 Credits

There was an optional *"question"* as well, where the people who answered the *form* were given the option to leave their name or nickname which would be credited in this report.

Major thanks to the following people for taking their time to answer the *form*:

1. ElDavido
2. Ricardo
3. Nicko
4. Afonso

6.5 Conclusions

The next chapter will compare the expected objectives to the executed ones and further discuss the future work.

Chapter

7

Conclusions and Future Work

7.1 Objectives Met

As initially proposed was made a "***2D Platformer***", where the main goal is to reach the end as fast as possible while avoiding ***Obstacles*** and ***Enemies***.
The input controls support only ***keyboard***. Multiple movement mechanics were implemented (***Jump***, ***Dashes***, ***Quick Fall***, ***Wall Jump*** and ***Wall Slide***).
It was created a ***Main Menu***, ***3 Levels*** that also contain 3 different UI menus (***Pause Menu***, ***Top10 Menu*** and ***Not Top10 Menu***). ***One scene*** that displays the ***High Scores*** of a selected level. ***One scene*** that shows the ***Replay*** of a selected level. ***One Options*** menu that allow the player to control various aspects of the game. A scene that allows a ***selection*** of either the Levels to ***play***, High Score Levels ***Consultation*** and Level ***Replay*** Selection.
Two types of rudimentary enemies with Artificial Intelligence (AI), and obstacles (both ***static*** and ***dynamic***) were also created.
Three additional features developed were a ***Timer / Checkpoint system*** , a ***High Score System*** and a ***Replay System***.
Every ***Player Mechanic***, ***Obstacle*** and ***Enemy*** were created with the intent of ***tweaking*** while creating additional levels, leading to an endless realm of possibilities when designing the levels.
It is possible to conclude every objective was met and even more additional features were created in the making of this game.

7.2 Future Work

As a future endeavour the main objective is to ***re-evaluate*** the ***Replay System***, due to its toll on performance in this version of the game.
Another big improvement would also be the creation of ***more NPCs*** and ***Obstacles*** with either more complex or simple, yet fun and harder behaviours, that would take advantage of the movement mechanics developed.
A ***graphical and sound overall*** could also be possibly pondered, that would have a more custom and personal take to it, which would allow me to not rely on the *Unity Asset Store* as a resource.
The implementation of ***Local-Multiplayer*** and ***Online Leaderboards*** would also bring useful gaming components to the game and the competitive aspect of it.
The ***Local-Multiplayer*** would work as a split-screen, where 2 players would compete on the same level against each other.
The ***Online Leaderboards*** would simply be an online version of the already created local High Score tables for each level, this leaderboard could possibly have the replay of the best time of each level, that could then be watched online as well.
Last but not least a major step in the game's development would be a ***public release***.

Bibliography

[1] *Mario.* https://i.kinja-img.com/gawker-media/image/upload/t_original/rp9crpla17856bynctsm.jpg Last access on June 6 2021.

[2] *Sonic.* http://2.bp.blogspot.com/-PVySb6zVo84/TrgwlI1rBwI/AAAAAAAAIxA/HI2sK3LIrX8/s1600/972790_20101011_screen001.jpg Last access on June 6 2021.

[3] *editorimage.* http://www.vmg206.com/images/unity-editor-3.jpg Last access on June 6 2021.

[4] *Project.* https://www.di.ubi.pt/~ngpombo/units/docs/project/proposals/P213.pdf Last access on June 6 2021.

[5] *RoadRunner.* https://looneytunes.fandom.com/wiki/Road_Runner Last access on June 6 2021.

[6] *CSS.* https://en.wikipedia.org/wiki/Counter-Strike:_Source Last access on June 6 2021.

[7] *Surf.* https://www.youtube.com/watch?v=kuUOrCxOiQ4 Last access on June 6 2021.

[8] *Unity's Flow Chart.* https://docs.unity3d.com/Manual/ExecutionOrder.html Last access on June 6 2021.

[9] *UnityEngine.* https://unity.com Last access on June 6 2021.

[10] *VSC.* https://code.visualstudio.com Last access on June 6 2021.

[11] *IDE.* https://www.codecademy.com/articles/what-is-an-ide Last access on June 6 2021.

[12] *Csharp.* https://en.wikipedia.org/wiki/C_Sharp_(programming_language) Last access on June 6 2021.

[13] *Paint.* https://support.microsoft.com/en-us/windows/get-microsoft-paint-a6b9578c-ed1c-5b09-0699-4ed8115f9aa9 Last access on June 6 2021.

[14] *SLOBS.* https://streamlabs.com Last access on June 6 2021.

[15] *LateX.* https://www.latex-project.org Last access on June 6 2021.

[16] *Overleaf.* https://www.overleaf.com Last access on June 6 2021.

[17] *platformer.* https://en.wikipedia.org/wiki/Platform_game Last access on June 6 2021.

[18] *speedrun.* https://en.wikipedia.org/wiki/Speedrun Last access on June 6 2021.

[19] *tutorial.* https://learn.unity.com/project/ruby-s-2d-rpg Last access on June 6 2021.

[20] *assets.* https://duckduckgo.com/?q=unity+assets&t=vivaldi&ia=web Last access on June 6 2021.

[21] *transform.* https://docs.unity3d.com/ScriptReference/Transform.html Last access on June 6 2021.

[22] *camera.* https://docs.unity3d.com/Manual/class-Camera.html Last access on June 6 2021.

[23] *Sprite.* https://docs.unity3d.com/ScriptReference/SpriteRenderer.html Last access on June 6 2021.

[24] *tilemap.* https://docs.unity3d.com/Manual/class-Tilemap.html Last access on June 6 2021.

[25] *tiles.* https://docs.unity3d.com/Manual/Tilemap-CreatingTiles.html Last access on June 6 2021.

[26] *rigidbody.* https://docs.unity3d.com/ScriptReference/Rigidbody2D.html Last access on June 6 2021.

[27] *upe.* https://docs.unity3d.com/Manual/PhysicsSection.html Last access on June 6 2021.

[28] *collider.* https://docs.unity3d.com/ScriptReference/Collider2D.html Last access on June 6 2021.

[29] *animator.* https://docs.unity3d.com/ScriptReference/Animator.html Last access on June 6 2021.

[30] *eventsystem.* https://docs.unity3d.com/2018.4/Documentation/Manual/EventSystem.html Last access on June 6 2021.

[31] *canvas.* https://docs.unity3d.com/560/Documentation/Manual/class-Canvas.html Last access on June 6 2021.

[32] *button.* https://docs.unity3d.com/2018.3/Documentation/ScriptReference/UI.Button.html Last access on June 6 2021.

[33] *text.* https://docs.unity3d.com/2018.4/Documentation/ScriptReference/UI.Text.html Last access on June 6 2021.

[34] *listener.* https://docs.unity3d.com/Manual/class-AudioListener.html Last access on June 6 2021.

[35] *source.* https://docs.unity3d.com/ScriptReference/AudioSource.html Last access on June 6 2021.

[36] *script.* https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html Last access on June 6 2021.

[37] *tag.* https://docs.unity3d.com/Manual/Tags.html Last access on June 6 2021.

[38] *layers.* https://docs.unity3d.com/Manual/Layers.html Last access on June 6 2021.

[39] *prefab.* https://docs.unity3d.com/Manual/Prefabs.html Last access on June 6 2021.

[40] *instantiate.* https://docs.unity3d.com/ScriptReference/Object.Instantiate.html Last access on June 6 2021.

[41] *singleton.* https://csharpindepth.com/Articles/Singleton Last access on June 6 2021.

[42] *staticclass.* https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members Last access on June 6 2021.

[43] *playerprefs.* https://docs.unity3d.com/ScriptReference/PlayerPrefs.html Last access on June 6 2021.

[44] *particles.* https://docs.unity3d.com/Manual/ParticleSystems.html Last access on June 6 2021.

[45] *API.* https://www.ibm.com/cloud/learn/api Last access on June 6 2021.

[46] *JSON.* https://www.w3schools.com/whatis/whatis_json.asp Last access on June 6 2021.

[47] *obj.* https://docs.unity3d.com/ScriptReference/Object.html Last access on June 6 2021.

[48] *compcoll.* https://docs.unity3d.com/Manual/class-CompositeCollider2D.html Last access on June 6 2021.

[49] *poly.* https://en.wikipedia.org/wiki/Polygon Last access on June 6 2021.

[50] *components.* https://docs.unity3d.com/Manual/Components.html Last access on June 6 2021.

[51] *cinemachine.* https://docs.unity3d.com/Packages/com.unity.cinemachine@2.8/manual/index.html Last access on June 6 2021.

[52] *assetstore.* https://en.wikipedia.org/wiki/Polygon Last access on June 6 2021.

[53] *animcontroll.* https://docs.unity3d.com/Manual/class-AnimatorController.html Last access on June 6 2021.

[54] *smb.* https://docs.unity3d.com/Manual/StateMachineTransitions.html Last access on June 6 2021.

[55] *aoe.* https://www.esports.net/wiki/guides/aoe-meaning/ Last access on June 6 2021.

[56] *raycast.* https://docs.unity3d.com/ScriptReference/Physics2D.Raycast.html Last access on June 6 2021.

[57] *overlapcircle.* https://docs.unity3d.com/ScriptReference/Physics2D.OverlapCircle.html Last access on June 6 2021.

[58] *fixedupdate.* https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html Last access on June 6 2021.

[59] *deltatime.* https://docs.unity3d.com/ScriptReference/Time-fixedDeltaTime.html Last access on June 6 2021.

[60] *lic.* https://creativecommons.org/licenses/by-sa/3.0/ Last access on June 6 2021.