Development of a metroidvania centered around combat and movement

Álvaro López Ortiz

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

July 3, 2023

Supervised by: Zoe Valero Ramón

Acknowledgments

First of all, I would like to thank my Final Degree Work supervisor, Zoe Valero Ramón, for all the help and tips on how to approach my project.

I would also like to thank my mother and my father for supporting me and for doing everything in their hands to help me.

And thanks to Víctor for helping me find an appropriate name for the video game.

I also would like to thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this report.

Abstract

This document presents the Final Degree Work project of Álvaro López Ortiz in Video Games Design and Development.

The project consists of a video game about a city destroyed by evil corporations. The player will control Taffy, a part human part cyborg girl who will traverse this city, defeating enemies and final bosses in order to bring down all the evil corporations. This will be represented in a 2D environment with action mechanics and pixel art style.

Contents

Chapter 1

Introduction

Contents

This chapter contains a presentation of the project and a summary of what is intended to accomplish. Here there are the reasons why the project is shaped the way it is, all its first ideas and objectives. There is also a summary of the project, how it started and all the previous knowledge.

1.1 Work Motivation

The main motivation that made me develop this project was my love for the Metroidvania genre[1], one of my favourite videogames genres, and I wanted to create my take on them while learning how they work and what makes them stand out.

The Metroidvania genre consists of a genre of videogames that revolves around exploration, but platforming and action are also included. One of the main features of this genre is that as the player advances in the game, new abilities will be unlocked, and these new abilities will let the player visit new parts of the map that were inaccessible before. The name Metroidvania comes from the fusion of the names Metroid and Castlevania, two popular videogame sagas that are the main inspiration for these games

Another motivation I had was that I recently started doing pixel art, and I saw this as an opportunity to improve my art and develop pixel art skills.

Regarding the theme, I have always enjoyed the cyberpunk aesthetic and have wanted to include it in a project for a long time, so I thought this was the perfect opportunity.

1.2 Objectives

- Learn how Metroidvania games work and apply this knowledge to create one.
- Implement a diverse and responsive movement system.
- Create a detailed pixel art.
- Animate everything so it feels cohesive.
- Implement a fully functional attack system with multiple diverse attacks and operative combos.

1.3 Environment and Initial State

When I was thinking about the project, I started seeing some problems. While I still wanted to create a game with a Metroidvania structure, as I started to think more about it, I realized that these games are too big to be done by just one person with this small amount of time, so I decided that the best option would be to create a small Metroidvania, with fewer levels, and focus on specific things like movement and the environment and art.

The project will be fully done in Unity[2] since is the game engine that I know how to use the most, and it is also the most fitting for this project, code will be written in Visual Studio[3]. No free assets will be used for the artistic part of the project; all the programming and every part of the visual aspect of the project will be made entirely by the author of the project, however, different sound effects with free royalties will be used. All the art will be done using Graphicsgale[4] and Krita[5].

CHAPTER 2

PLANNING AND RESOURCES EVALUATION

Contents

This chapter mainly contains the technical part of the project, its original planning and the tools used for it. During development, the planning changed a lot, so here is also included the final schedule used for the project and how much time each took task.

2.1 Planning

This section explains every project objective and shows how much time is expected to use in each part. The following table consists of the initial planning for the project and the way I decided to distribute the work (table 2.1).

Implementation of a basic movement	2 hours
Basic attacks with first melee weapon	10 hours
Character design and first 3 melee weapons	5 hours
Implementation of various attacks for each weapon, basic combos	15 hours
Design of first level	7 hours
Movement upgrade adapted to the first level	10 hours
Basic enemies design and implementation	5 hours
Basic AI (artificial intelligence) implementation for the enemies	6 hours
First final boss	12 hours
First 3 ranged weapons implementation	12 hours

Adapt combos to make them work with ranged weapons	20 hours
Parries implementation	10 hours
Roller Skate implementation, movement upgrade	10 hours
Second final boss	15 hours
Add two more melee weapons	10 hours
Add two more ranged weapons	10 hours
Add two more levels	16 hours
Main hub implementation	10 hours
Polish combat and combo system	30 hours
Add last level	10 hours
Third final boss	17 hours
Add various NPCs (non-playable characters) and dialogue system	25 hours
Main screen	3 hours
Saving system	10 hours
Playtesting	5 hours
Final memory and presentation	10 hours

Table 2.1: Initial planning for the project

However, during the development of the project, this table experienced lots of changes, some things took longer than expected, and others took more priority than others and got shelved. Therefore, here is the table rescheduled with the final planning (table 2.2).

Design of the main character	2 hours
First animations	12 hours
Movement implementation	7 hours
Jump and physics	8 hours
Implementation of basic attacks	5 hours
Animations and state machines implementation	7 hours
Camera implementation, cinemachine and pixel perfect extension	5 hours
First enemy implementation, AI (artificial intelligence) and design	7 hours
Implementation of diverse attacks	15 hours
Diverse attacks animations	10 hours
Movement animations	12 hours
Level layout design	7 hours
First level design	7 hours
Ladders implementation	2 hour
First final boss	15 hours
Ranged weapon implementation	10 hours
Rollerblades implementation	5 hours
Rollerblades animation	6 hours

Trail implementation	3 hours
Multiple basic enemies addition	15 hours
Second final boss	12 hours
Combat feedback, camera shake, knockback and diverse colors on hit	8 hours
Third final boss	12 hours
Second and third level implementations	10 hours
Background implementations	8 hours
Attacks polishment	20 hours
Settings menu implementation	5 hours
Inventory implementation	12 hours
Addition of diverse items	8 hours
Level progression adaptation	8 hours
Unlockable abilities implementation	5 hours
Main menu implementation	4 hours
Game ending	5 hours
Playtesting	5 hours
Final memory and presentation	20 hours

Table 2.2: Final time distribution for the project

2.2 Resource Evaluation

To create this project, I used mainly the resources I had available. This consists of my personal computer and software familiar to me. I used this computer because it is my main work tool, and its components are good enough to use every desired tool. The elected software also consists of personal preference, all tools used are ones that I feel comfortable using, and I know how to use already, so I will not need to learn any new program to develop the project.

1. Main computer (i7 CPU, 16GB RAM, RTX 2060 GPU, Windows 10): 800 euros approximately
2. Unity 2D 2021.3.18f1: Free
3. Visual Studio 2019 16.11: Free
4. Krita 5.0: Free
5. GraphicsGale 2.09: Free

Chapter 3

System Analysis and Design

Contents

This chapter contains the requirement analysis of the project, the functional requirements, the non-functional requirements and some tables explaining these requirements. An explanation of how the visual interface works is also included.

3.1 Requirement Analysis

Before continuing with the analysis, I think it is convenient to talk about the project. The game will be an action and platforming game with a Metroidvania structure, and the game will consist of four levels, the first of them being a main hub with no enemies. The rest of the levels will have a final boss each; in order to progress through the game, the player will have to complete these levels and defeat every single boss. Even if there are multiple levels, the game will be linear, and the player must face the bosses in a specific order. Each boss will grant the player a new ability that will allow the player to go to the next level; when the third boss is defeated, the game will be completed.

3.1.1 Functional Requirements

First, this section contains a list of functional requirements [6] explaining how the game works.

1. R1. The player can move to the left and to the right
2. R2. The player can jump
3. R3. The player can attack
4. R4. The player can shoot
5. R5. The player can pause the game
6. R6. The player can run
7. R7. The player can use rollerblades
8. R8. The player can slide through pipes when using rollerblades
9. R9. The player can climb ladders
10. R10. The player can damage enemies
11. R11. The enemies can move alongside a determined path automatically
12. R12. The enemies can detect the player automatically
13. R13. The enemies can attack the player automatically
14. R14. The bosses can use multiple attacks automatically

3.1.2 Non-functional Requirements

This section contains a list of non-functional requirements[7] summarising the game's qualities.

1. R16. The game will be playable with a keyboard and with a controller
2. R17. The game will have pixel art style
3. R18. The game will have a cyberpunk theme
4. R19. The UI (User Interface) will be simple
5. R20. The animations will be fluid
6. R21. The controls will be easy to understand

3.2 System Design

To end with the requirements of the game, in this section, some tables are presented explaining how these requirements work and how they are carried through the game.

Requirement:	R1
Actor:	Player
Description:	The player moves to the left or to the right.
Preconditions:	1. The player is not attacking
Normal sequence:	1. The player presses A or D 2. The character moves
Alternative sequence:	None

Table 3.1: Case of use «CU01. Move»

Requirement:	R2
Actor:	Player
Description:	The player jumps.
Preconditions:	1. The player is on the ground 2. The player has the ability to double jump and has one jump left
Normal sequence:	1. The player presses the space button 2. The character jumps
Alternative sequence:	1. The player has no jumps left 2. The player does not jump

Table 3.2: Case of use «CU02. Jump»

Requirement:	R3
Actor:	Player
Description:	The player attacks.
Preconditions:	None
Normal sequence:	1. The player presses the attack button 2. The character attacks
Alternative sequence:	None

Table 3.3: Case of use «CU03. Attack»

Requirement:	R4
Actor:	Player
Description:	The player shoots in the direction of the mouse.
Preconditions:	None
Normal sequence:	1. The player moves the mouse in the direction desired 2. The player presses the shoot button 3. The character shoots
Alternative sequence:	None

Table 3.4: Case of use «CU04. Shoot»

Requirement:	R5
Actor:	Player
Description:	The player pauses the game and opens the options menu.
Preconditions:	1. The game has started
Normal sequence:	1. The player presses the pause button 2. The menu is opened
Alternative sequence:	None

Table 3.5: Case of use «CU05. Pause»

Requirement:	R6
Actor:	Player
Description:	The player runs to the left or to the right.
Preconditions:	1. The player is not attacking 2. The player is moving
Normal sequence:	1. The player presses the run button 2. The character runs
Alternative sequence:	1. The player presses the run button 2. The character is not walking 3. The character doesn't move

Table 3.6: Case of use «CU06. Run»

Requirement:	R7
Actor:	Player
Description:	The player moves quicker when using roller blades.
Preconditions:	1. The roller blades are equipped
Normal sequence:	1. The player moves when the roller blades are equipped 2. The character runs faster
Alternative sequence:	1. The roller blades aren't equipped so the character moves normally

Table 3.7: Case of use «CU07. Roller blades»

Requirement:	R8
Actor:	Player
Description:	The player slides automatically through a pipe.
Preconditions:	1. The player is wearing the roller blades
Normal sequence:	1. The player jumps on the pipe 2. The character slides through the pipe
Alternative sequence:	1. The player is not wearing the roller blades 2. The character falls through the pipe

Table 3.8: Case of use «CU08. Pipes»

Requirement:	R9
Actor:	Player
Description:	The player climbs a ladder upwards or downwards.
Preconditions:	1. The player is near a ladder
Normal sequence:	1. The player presses the up button 2. The character climbs the ladder
Alternative sequence:	1. The player is not near a ladder 2. Nothing happens

Table 3.9: Case of use «CU09. Climb ladder»

Requirement:	R10
Actor:	Player
Description:	The player attacks an enemy.
Preconditions:	1. The player is near an enemy
Normal sequence:	1. The player presses the attack button 2. The enemy's health decreases
Alternative sequence:	1. The player is not near an enemy 2. The character attacks normally

Table 3.10: Case of use «CU010. Damage enemies»

Requirement:	R11
Actor:	Enemy
Description:	The enemies move between two determined points.
Preconditions:	None
Normal sequence:	1. The enemy moves to the next determined point
Alternative sequence:	None

Table 3.11: Case of use «CU11. Enemies movement»

Requirement:	R12
Actor:	Enemy
Description:	The enemies detect the player when is in range.
Preconditions:	1. The player is in range
Normal sequence:	1. The player enters the enemy range 2. The enemy detects the player
Alternative sequence:	1. The player does not enter in range 2. The enemy keeps moving

Table 3.12: Case of use «CU12. Enemies detecting the player»

Requirement:	R13
Actor:	Enemy
Description:	The enemies attack the player.
Preconditions:	1. The player is in range 2. The player does not go outside the detection range before being attack
Normal sequence:	1. The player enters the enemy range 2. The enemy detects the player 3. The enemy attacks the player
Alternative sequence:	1. The player leaves before being attack 2. Nothing happens

Table 3.13: Case of use «CU13. Enemies attacking the player»

Requirement:	R14
Actor:	Boss
Description:	The enemies use multiple attacks when fighting the player, these attack are decided randomly.
Preconditions:	1. The boss is not attacking
Normal sequence:	1. The boss selects a random attack 2. The boss uses that attack
Alternative sequence:	None

Table 3.14: Case of use «CU14. Bosses using multiple attacks»

3.3 System Architecture

1. Operating system Windows 7 at least
2. CPU with x86 or x64 architecture
3. Graphics card with DX10 capabilities

3.4 Interface Design

The UI [8] is designed to be as clean and as simple as possible, giving only the necessary information to the player, this only being the character's current health, appearing at the top left of the screen. Some inspiration was taken from Hollow Knight (2017)[9] in order to create this UI.

During boss battles, at the bottom of the screen, the boss' health bar will also appear, but the rest of the enemies' health will not be visible, so the screen is not crammed with multiple bars.

Chapter 4

Work Development and Results

Contents

Here is the more technical part of the memory. In this chapter, every part of the work done is presented, as well as all the complications and the results obtained while doing so.

4.1 Work Development

This section will consist of a discussion of the work development in chronological order. Most of the things and the mechanics implemented in the project depend on each other, so explaining it this way makes the most sense.

4.1.1 Main character

The first thing implemented for the project was the main character design, a girl named Taffy with cyborg characteristics that wanders through the city at night. Taffy being the most important character of the game, it made sense for her to have a design that stood up, so the first prototypes were drawn on a piece of paper as sketches, and then her design was translated to a pixel art style using graphics gale. Some inspiration for the design was the movie Blade Runner 2049 (2017)[10], especially the characters K and Joi, these characters represent very well the cyberpunk aesthetic, so Taffy has a trench coat similar to K's and a face and hair inspired by Joi. Taffy has great strength thanks to her cyborg implants, which must be reflected in her design. To create every design of the project, two different colour palettes were used, one for the characters and another for the background. After that designing the main character in pixel art was quite easy (Figure 4.1).

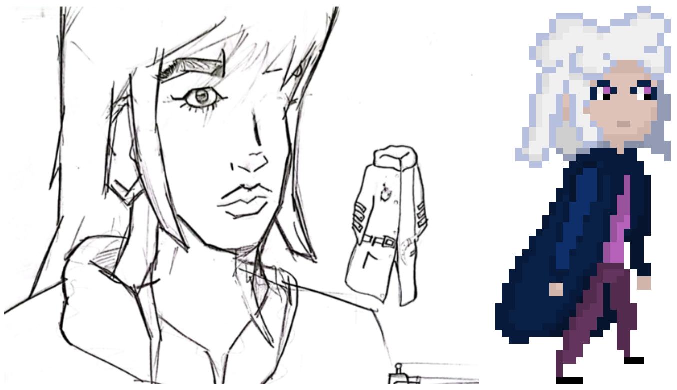


Figure 4.1: Taffy's design

In order to keep working on her design, the next thing implemented was some movement animations. This was the moment I realised the amount of time and effort that took animating in pixel art, designing the main character took way more time than I expected, I had to implement some techniques to make the process quicker, I completely swapped to Krita because I felt more comfortable using its animation tools and I reduced the character to simple shapes, each one corresponding to a different type of her body (Figure 4.2), using this shapes was easier to create movement, and when the animation was complete using these shapes I draw the character on top of them. With this technique, I made three animations for three different attacks.

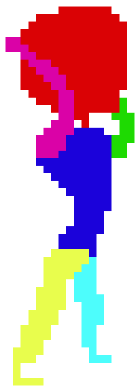


Figure 4.2: Reduction of Taffy's design to simple shapes

Next, I coded the movement script for the player, here, I had to make the first important decision, if I wanted to implement a physics-based system or not. A physics-based movement works so different forces are applied to the character, and these forces determine its movement. A movement system without physics consists of changing the transform component of the character, creating a variation of its coordinates and creating movement. The latter is way simpler to implement but lacks precision. I decided not to implement a physics system since it was simpler, and I felt more comfortable doing so, also before coding it, I thought that it would not be necessary to have a physics-based system and that it would not be worth the effort because the game would not need to be as precise and complex, but when I started to code the jump button I realized that it didn't felt good at all, and I was starting to have lots of complications, so I swapped to a physics-based system, this change of plans took a bit of time but not a lot.

Regarding the character movement, I also added a trail that follows Taffy (Figure 4.3). It does not always follow her, only when she reaches certain speed, for example, it activates while running, and while dashing. This effect does not give the player an increase in speed, however, the effect helps to feel the velocity rise, and also makes lots of sense in this cyberpunk setup.

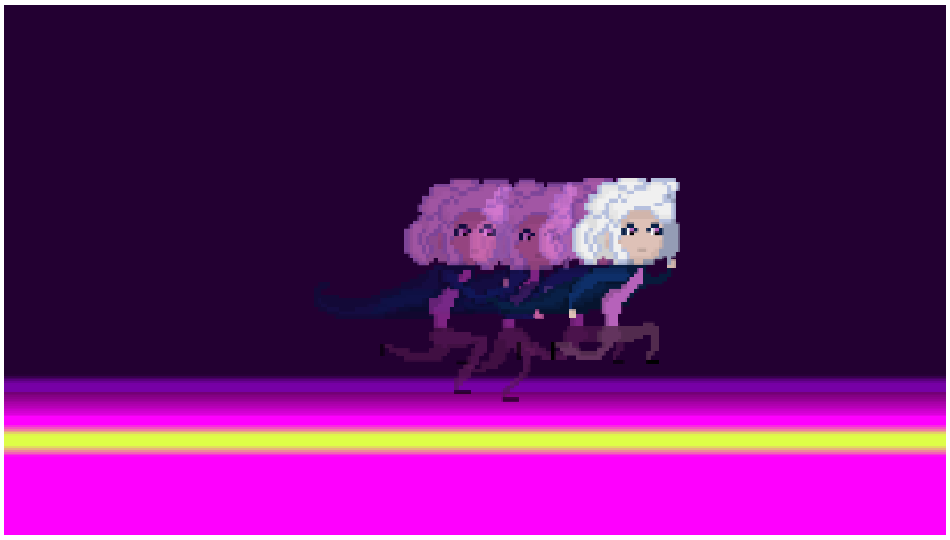
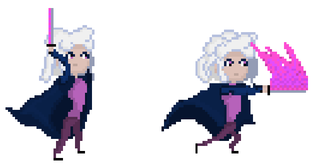
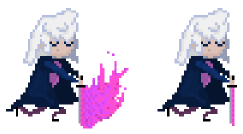


Figure 4.3: Trail

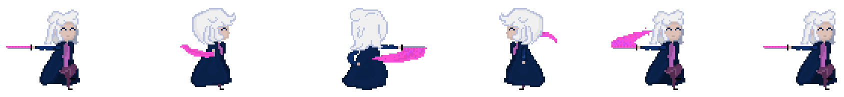
4.1.2 Combat and enemies

In order to finish with the player scripts, I coded the attacks, and this was quite an easy task, I used two empty objects to form a square, this square marks the area where the attacks will deal damage; this area appears when the attack button is clicked and disappears when is not being clicked. On the other hand, implementing the attack animation was more difficult since I wanted to implement a simple combo attack[4.4], I wanted when the attack button clicked multiple times different attacks would follow each other, and I had to investigate Unity's animation tools, I ended up using animation behaviours and state machines[11], this was quite intimidating because I had never used them, but it ended up being quite easy and intuitive, and I used them for every animation in the project[4.5].

(a)

(b)



(c)

Figure 4.4: Taffy's basic attacks

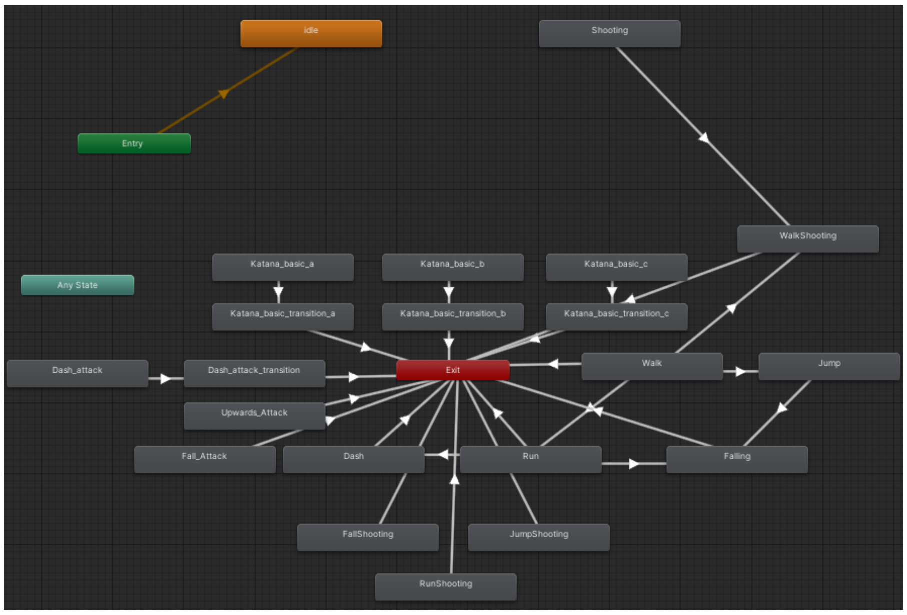


Figure 4.5: Taffy's state machine

I wanted to try to see if attacks worked right, so I implemented the game's first enemy with its design being a placeholder. I wanted to make a simple script that could work as a primitive script for every enemy of the game, so I implemented a patrolling system that consisted of two empty objects with the enemy moving between them. I also added an area in front of the enemy that detects if the player is inside. If it is, it attacks them. Finally, I added one last script to keep track of the health.

To finish with the character script, I designed three more attacks, a dash attack, a falling attack and an upwards attack. I also animated more actions, such as walking, running and jumping. Implementing all of this was simple and easy, but the design took a lot of time; animating in pixel art was way harder than I expected, especially if I wanted it to look more or less good.

At this moment, I had another critical problem, I wanted to implement forces to the player's attacks so that every time she attacked, she would also move forward a little bit, I also wanted to implement a knockback system. When I was implementing this, I had lots of problems with the physics system, and it would not know what was going wrong. It took me a lot of time, so I decided to book the issue.

4.1.3 First level

The design of the world was simple, I wanted to create a big city with a strong cyberpunk aesthetic for the main level. To do this, I looked for lots of references from different media, such as the city from Cyberpunk 2077 (2020)[12] and the movies Blade Runner (1982)[13] and Blade Runner 2049 (2017)[10], and I eventually came up with a design that I felt like it fitted. I wanted to give this scenery lots of detail, so it took longer than expected (Figure 4.6 and Figure 4.7).

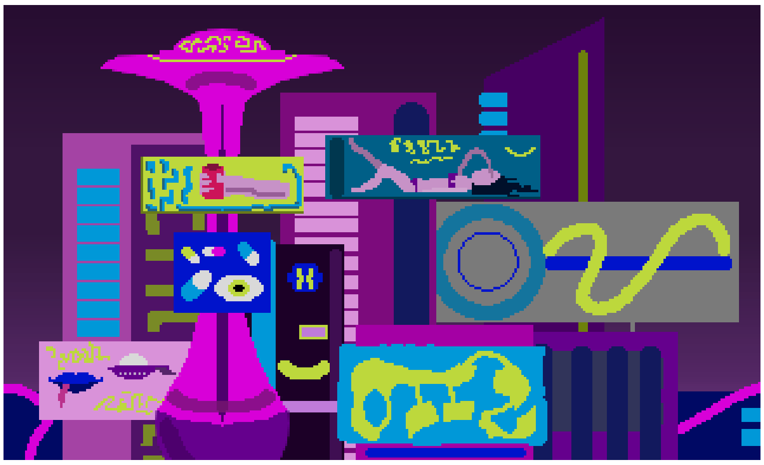


Figure 4.6: Background of the first level

I wanted to give the scene a more dynamic feel so I added a parallax effect to it, I divided the scene into multiple layers, each corresponding to a different building, and

I assigned different speeds to each building. To create more ambience, I added a small rain effect too. I also designed the structure of the level and began to design the game's progression. After the first boss is defeated, the player will unlock the ability to double jump, so when the game starts, the player can not go to the left since the platform is too high; the first boss must be defeated first. Then I designed the rest of the levels and more unlockables, I also added the possibility to equip rollerblades, which will be unlocked after defeating the second boss, and I coded a metal pipe where you can go only when the rollerblades are equipped.

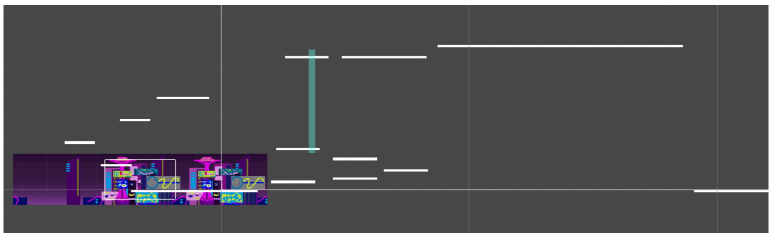


Figure 4.7: Layout of the first level

4.1.4 Bosses

When the main levels were designed, I started coding the first boss (Figure 4.8). I wanted to make a robot so the move set would be simple and easy to avoid, perfect for the first boss. I created three unique attacks for the boss, a punch, some stomps and a laser ray that aims at the player. Since I was already coding the laser ray, I also decided to give the player a ranged weapon and the ability to shoot it because the code was quite similar (Figure 4.9). Implementing this boss was quite easy, and it took the expected time. After finishing this boss, I designed two simple enemies, a robotic security guard and a robotic hedgehog, both of them patrol between two given points and attack the player when in range.

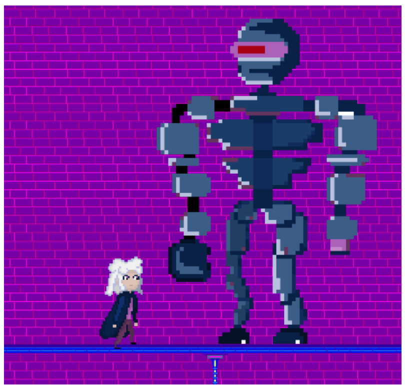


Figure 4.8: First boss of the game

When coding the punch, I realised that adding a knockback would be perfect, so I investigated the issue once again, and I eventually solved the problem, the physics of the movement needed to be disabled when applying this knockback, knowing it was simple to implement, but finding the problem took a lot of time.

Since the background of the first level took longer than expected, I decided to swap to use tiles for the rest of the levels, the result was satisfactory, and the time used was lessened.

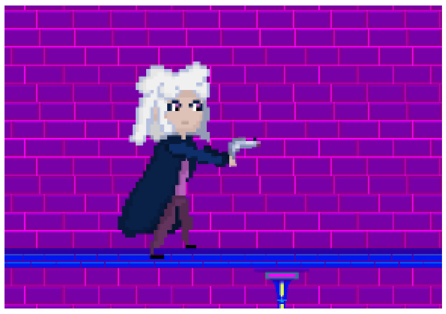


Figure 4.9: Player aiming the gun

I wanted to create a bigger challenge for the second boss, so I designed a cyborg swordsman with a katana and similar skills as the player. I gave him four different attacks, two slashes in different directions, a punch with great knockback and a dash attack. The script used for the cyborg inherits from the basic enemy script but changes it so that instead of moving through two different points, he always runs in the direction of the player. When in range, the boss makes one attack randomly. The implementation of this boss was simple, except for the dash attack, which was a bit problematic (Figure 4.10).

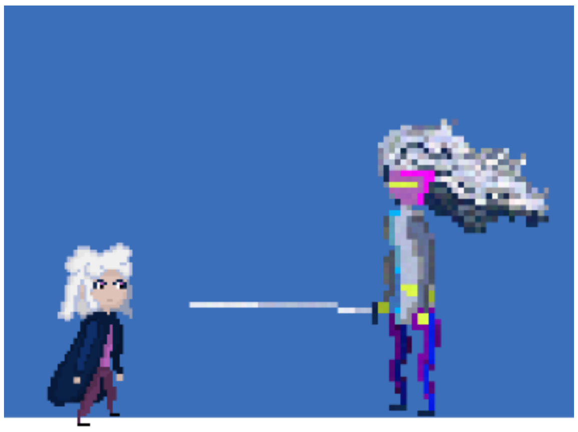


Figure 4.10: Second boss of the game

I also designed the background for this boss battle, I wanted to create a simple one, so I designed some tiles with a main purple colour and some complementary colours working as neon lights. While simple, this background kept the essence of the cyberpunk aesthetic.

Then, to finish with the bosses of the project, I designed the third and last final boss. For this boss, I wanted to make an eerie design. I liked the idea of floating eyes, so I designed the boss around it, I created the eyes and a hand, and I made a simple animation for both of them floating around, for the most part of the battle the boss is untargetable, the player can not hit its eyes, nor its hand, the only way to damage it is to wait for its attacks. The attack pattern is simple, the boss has two attacks, one where it puts its hand above the player, waits for a second and then hits the ground; the second attack is a horizontal punch. Both attacks have a cooldown. When the attack finishes, the hand remains still for a second before returning to its initial state, that is the moment when the player has the ability to damage the boss (Figure 4.11). After designing this boss, I thought that it would fit better as a second boss better than the one with a katana, and the background was more fitting as well, so I moved the boss there. In order to finish adapting the background, I added a new tilemap on top of the current tilemap, for this new tilemap I created a new tile set with four more tiles, these tiles consisted of a colour transition from purple to black, so the zone of the level where the boss stands is completely black. When the boss is defeated, this new tile set is hidden.

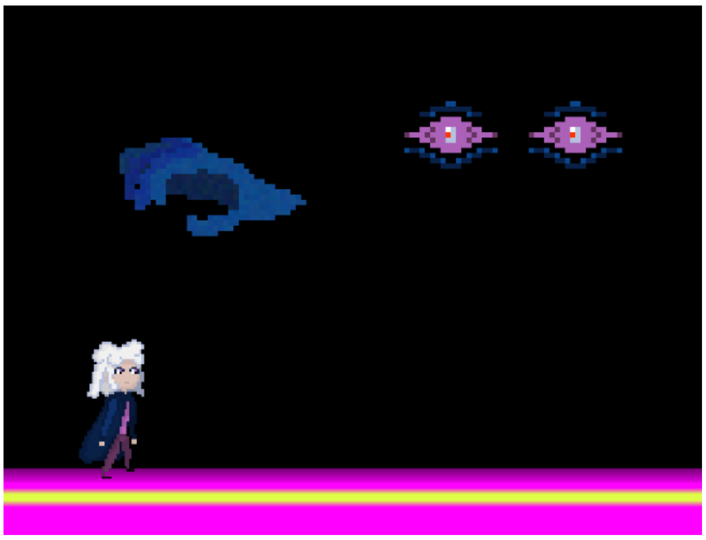


Figure 4.11: Third boss of the game

4.1.5 Combat fixes

After testing the battle against the bosses, I realised that there were some issues regarding the combat system, most hitboxes did not work well, and the shooting animations were not working either. First of all, I fixed the shooting animations; the main problem was that I designed the shooting mechanic so that when the player shot a new sprite consisting of two arms holding a gun appeared on top of the main sprite, this sprite didn't change so it looks like the character had four arms, I had to create variations of the sprites without arms so when the sprite holding the gun appeared it seemed normal. I created variations for the idle animation, the walking animation, the running animation, the jumping animation and the falling animation since these are the moments where the player is able to shoot. With the use of animation behaviours, these new sprites were easy to implement. Then, regarding the hitboxes, the only one that worked well was the main attack one. To fix this, I created new collision boxes. These collision boxes were inactive for the most part until the player did the desired attack, and then the corresponding collision box activated for a moment.

4.1.6 Inventory

I was not sure if an inventory system was necessary for the game, it would have been mandatory if I had added multiple weapons, but since only one was usable, I was starting to doubt. However, I did some research for the implementation of inventory systems, and it was quite simple. Taking this into account and the fact that I needed a way to equip and unequip the rollerblades, I decided to implement an inventory. The implemented inventory was simple since it is not a fundamental game part. I decided not to give it lots of layers and to give it only essential functions, when the inventory is opened, a list of items appears, and the player can click on the items to use them. Having now an inventory, I realised that I could also add simple items that could give more variety to the game and could serve as rewards for exploration, such as potions that will heal the player a bit and upgrade tools that will grant the player more damage (Figure 4.12). When finishing the inventory I also added a settings menu.

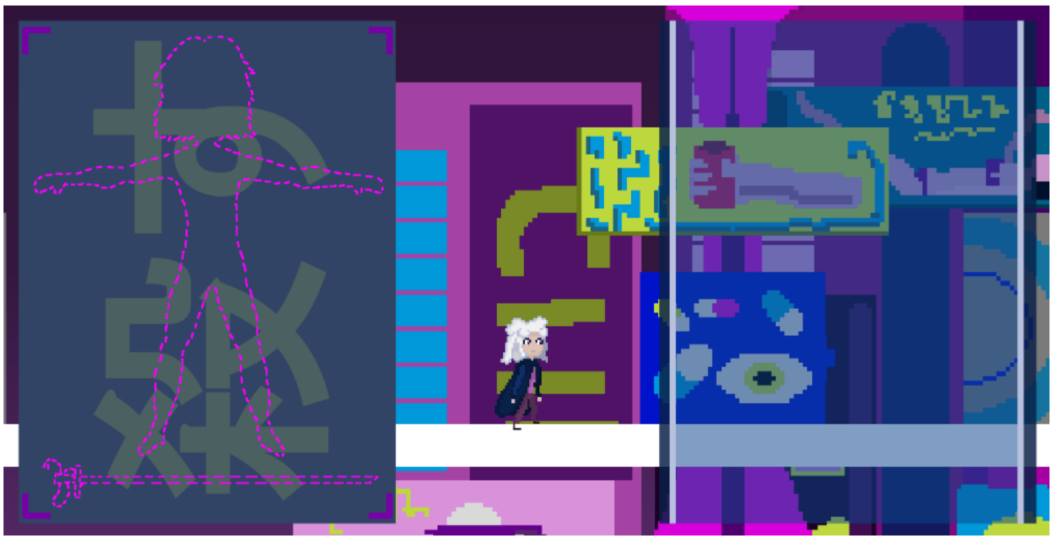


Figure 4.12: *Inventory*

4.2 Results

During the development of the project, some changes needed to be made due to time restrictions, mainly regarding the animations. Animating in pixel art took way more time than expected, and if I wanted to get satisfactory results, I had to expend more time in this area than I could afford, so in order to create more mechanics and develop them correctly, I decided to make simpler animations.

Some other changes needed to be made, I also discarded the idea of having multiple weapons since using just one would be simpler and easier to understand, and I could spend more time improving the main weapon. Also, balancing the game around multiple weapons would take an unnecessary amount of time. After making this decision, I was really satisfied with the results of the main weapons, the melee one and the ranged one, so I think that this change was necessary to make a better game.

Regarding the project's goals, reducing my workload made it easier to accomplish them. The combat system was fully implemented, it has diverse attacks that work well with each other and a combo system that works so that when the player hits the attack button multiple times, the character effectuates diverse attacks, a fully functional range weapon is also implemented. The implementation of the movement system was also satisfactory; the character can walk and run, but she can also dash, giving an extra layer that makes the movement more fun. The running attack also makes the movement quite interesting because it is a form of movement that also deals damage to the enemies, but it has a counterpart that, when performing it, the direction can not be changed. Overall having to spend less time animating more weapons, I could perfect the attacks and the player's movement.

Chapter 5

Conclusions and Future Work

Contents

This chapter contains the conclusions of the work, as well as its future extensions. I also stated how I feel about the final work and how it could be improved.

5.1 Conclusions

The development of the project was quite a different experience since most of the projects of the degree are in groups, this is the first video game that I am doing all on my own. During these months, I experienced many things regarding video game development that I had never experienced before. One of the most important things that I learn is to reduce expectations. For the most part, everything takes longer than expected, so when planning the project is important to take this into account. One of the problems I had while developing the game was that I wanted to implement way more things than I could handle, so I had to decide which mechanics should be in the game and which ones should be shelved.

I also learnt about video game art, even I already knew about its importance I still underappreciated a lot. The artistic part of the project was the portion that took me the most time by far, I wanted to cover lots of artistic things but I had to reduce my workload because I just could not keep with it. And if we talk about pixel art I think that these characteristics intensify, having a small amount of pixels makes the mistake stand out more, so everything needs to be really precise, more so if every sprite needs to be animated too, because that precision needs to be reflected in the animation as well.

Even if I think the art was harder than I expected, I am very proud of the results. I think that the game looks well and that I did a good work with the animations, I am especially proud of the results of the background of the first level and the UI of the inventory.

5.2 Future work

While I am proud of the amount of content the game has, I think it could be expanded in many ways that would make a better experience. First of all, the weapon variety, adding more weapons to the game, was an idea that I had wanted to implement since the beginning, but it needed to be discarded. However, I think it would add variety and make the game more interesting, besides the game is already programmed with this in mind, so adding it will not be complicated.

I also think that it would be beneficial to the game to make the map larger, adding more levels and final bosses, making it a more round experience, I think this type of game benefits a lot from its length and from its map, making it longer would make the player be more familiar with the game and with the map, which really enhances the experience.

Adding more enemies to the game could also be interesting, the combat is inspired by hack and slash games, so fighting multiple enemies at once would add a new layer to the combat and it would make everything feel more complete and cohesive.

Finally, even if I think that the game looks good, I think working more on the art could make a big improvement. The presentation is really important, and Metroidvania games benefit a lot from it, I think that the pixel art itself is fine and works well enough, but working more on the animations would improve the looks of the game a lot.

Bibliography

[1] Wikipedia. Metroidvania. https://en.wikipedia.org/wiki/Metroidvania.

[2] Unity Technologies. Unity documentation. https://docs.unity.com.

[3] Microsoft. Visual studio documentation. https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022.

[4] Humanbalance Ltd. Graphicsgale. https://graphicsgale.com/us/.

[5] The Krita Team. Krita documentation. https://docs.krita.org/en/index.html.

[6] Wikipedia. Functional requirements. https://en.wikipedia.org/wiki/Functional_requirement.

[7] Wikipedia. Non-functional requirements. https://en.wikipedia.org/wiki/Non-functional_requirement.

[8] Wikipedia. Ui. https://en.wikipedia.org/wiki/User_interface.

[9] Team Cherry. Hollow knight. https://www.hollowknight.com, 2017.

[10] Denis Villeneuve. Blade runner 2049. https://en.wikipedia.org/wiki/Blade_Runner_2049, 2017.

[11] Unity. State machines. https://docs.unity3d.com/Manual/StateMachineBasics.html.

[12] CD Projekt RED. Cyberpunk 2077. https://www.cyberpunk.net/, 2020.

[13] Ridley Scott. Blade runner. https://es.wikipedia.org/wiki/Blade_Runner, 1982.

[14] Overleaf. Overleaf documentation. https://www.overleaf.com/learn.

[15] Vincent Knight. Writing with latex. https://vknight.org/tex/.

[16] The Game Kitchen. Blasphemous. https://thegamekitchen.com/blasphemous/, 2019.

[17] Halberd Studios. 9 years of shadows. https://www.halberdstudios.com/9-years-of-shadows, 2023.

[18] Platinum Games. Bayonetta. https://es.wikipedia.org/wiki/Bayonetta, 2009.

[19] Askiisoft. Katana zero. https://www.katanazero.com/age-gate, 2019.

[20] Studio Pixel Punk. Unsighted. https://www.humblegames.com/games/unsighted/, 2021.

[21] Brandon James Greer. Bjgpixel. https://www.youtube.com/@BJGpixel.

[22] Brakeys. Brakeys. https://www.youtube.com/@Brackeys.

[23] Stack Exchange. Stack overflow. https://stackoverflow.com.

[24] Gfx Sounds. Gfx sounds. https://gfxsounds.com.

[25] Pixabay. Pixabay. https://pixabay.com/.

[26] GFX sounds. Retro 8-bit damage sound effect. https://gfxsounds.com/sound-effect/retro-8-bit-damage/.

[27] GFX sounds. Arcade 8-bit shot sound effect. https://gfxsounds.com/sound-effect/arcade-8-bit-shot/.

[28] Lesiakower. Impact sound effect 8-bit retro. https://pixabay.com/es/sound-effects/impact-sound-effect-8-bit-retro-151796/.

[29] Pixabay. Kick-hard (8-bit). https://pixabay.com/es/sound-effects/kick-hard-8-bit-103746/.

[30] Pixabay. Fire. https://pixabay.com/es/sound-effects/fire-88783/.

[31] Pixabay. 8-bit explosion. https://pixabay.com/es/sound-effects/8-bit-explosion1wav-14656/.

[32] Pixabay. Hurt-c-08. https://pixabay.com/es/sound-effects/hurt-c-08-102842/.

[33] Pixabay. Retro video game death. https://pixabay.com/es/sound-effects/retro-video-game-death-95730/.

[34] Pixabay. Game fail sounds. https://pixabay.com/es/sound-effects/game-fail-sounds-104952/.

[35] Pixabay. Sfx jump 07. https://pixabay.com/es/sound-effects/sfx-jump-07-80241/.

Appendix A

Game Design Document

A.1 Presentation

Name: Infinity Holopunk
Platform: Personal Computer
Genre: Metroidvania

A.2 Game Summary

The game developed for the final project will be a Metroidvania game focused in movement and combat, it will be set in a dystopian future in which evil corporations dominate the world, and the goal of the main character is to stop them. The main character will traverse through various locations, defeating enemies, obtaining objects and unlocking new actions to accomplish her goal.

The main focus of the game will not be the plot but the mechanics, there will be a diverse range of movements available to the player, multiple enemies and various final bosses, each of them being a challenge to the player.

A.3 Game States

At the beginning of the game, the only thing the player can do is to move the character. In order to keep progressing, the player will have to kill enemies. There are to types of enemies, normal enemies and final bosses, if the player defeats a normal enemy nothing special will happen, however, if the enemy defeated is a final boss the player will gain a new ability that will grant the player the possibility to go to the next level. In the next

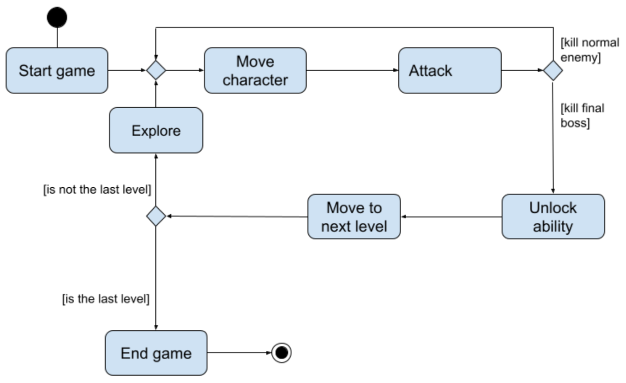


Figure A.1: Block diagram of the game states

level, the player will be able to explore, keep moving the player and kill more enemies. If the defeated boss is the final boss of the last level the game will be over (Figure A.1).

A.4 Demographics

A.4.1 Target

Being an action video game, it will be somewhat violent, with blood and some references not suitable for children. It will also have adult themes regarding the cyberpunk genre, so the game will be mostly aimed at young adults. Being a complex and hard video game it will also be aimed at a more hardcore audience.

A.4.2 Competitive analysis

Combining the Metroidvania genre with the hack and slash genre is a new concept that has not been explored before, however we can find similar concepts if we explore the genres individually.

Currently Metroidvania games are experiencing a rise in popularity, lots of them are being developed and released and it is a fairly popular genre, so the competition is quite high. Some of the direct competitors, that are also inspiration, are Hollow Knight (Team Cherry, 2017)[9], Blasphemous (The Game Kitchen, 2019)[16] or 9 Years of Shadows (Halberd Studios, 2023)[17].

On the other hand, hack and slash games are not as popular nowadays, but most of the competition comes from the AAA industry, with franchises such as Devil May Cry (Capcom) or Bayonetta (Platinum Games).

A.5 Narrative

A.5.1 Characters

The main character of the game will be Taffy, a girl with cyborg implants that grant her great strength who wants to destroy all evil corporations just because she enjoys it. Moreover, she does not speak, so the player can relate more to her.

The other characters of the game are the enemies, they also do not talk, but their design reflects their personality.

A.5.2 World

The video game will be set in a dystopian cyberpunk world, plagued by evil corporations, poverty and general despair. However, while everything is falling apart, it also has a futuristic aesthetic so everything looks beautiful and functional.

A.6 Rules, mechanics and game balance

A.6.1 Rules

The game will be a compound of different mechanics, but the main ones will be every mechanic regarding the attacks. In order to progress through the game, the player will have to travel through multiple interconnected levels that will contain a final boss. Each final boss will grant the player an upgrade or a new mechanic that will help to progress through the rest of the game.

A.6.2 Mechanics

The main goal of the project is to explore mechanics related to movement and combat, so the most important mechanics will be focused in these two fields.

Movement

The first form of movement available is running, it helps the player move faster, while running Taffy can also perform a running attack, which will deal damage but can not be redirected. Some of the other unlockable movement mechanics will be rollerblades, that will allow the player to move faster and to grind on metal pipes, or double jump that will help the player reach new heights. Eventually a dash button which also dodges enemies attacks will be unlocked.

Combat

The combat of the game will be centered around a combo system, depending on different combination of buttons the resulting attack will be different, for example, if the player stands still and hits the attack button the character will do an attack, but if the player hits the attack button while running the character will do a different attack. The player can also attack with a ranged weapon that will have just a simple attack, but it can be interconnected with the melee weapon.

A.6.3 Game balance

The difficulty of the game will be progressive, each stage will be harder. The first enemies and boss will be easier than the last one, which will be a hard challenge for the player.

A.7 Gameplay

A.7.1 Controls

The game will be playable with controller and with keyboard and mouse. Each button will be linked to just one action so it's always clear what button to touch.

The main controls will be simple and intuitive, there will be a button to run, one to use the melee weapon, another one to use the ranged weapon and one to jump. There will also be a button linked to a quick weapon change, so it's easier to do combos with different weapons.

A.7.2 Interface

The HUD will always be displayed and will be as simple as possible. In the top left corner will be displayed the player's health and in the bottom right corner will be displayed the selected weapons.

However, most of the important information will be displayed in the menu, here will be shown all the unlocked items and the player upgrades. There will also be another menu to alter the configuration of the game, like the resolution or the music volume.

A.7.3 Unlockables

Like most Metroidvanias, the game will be heavily based on unlocking new abilities and obtaining new weapons, in the beginning, the player will only have a simple weapon and the jumping and running abilities. Scattered through the map there will be lots of new weapons and objects, some of them hidden, that will add variety to the gameplay experience, all of these objects will be optional, and it will be possible to complete the game without collecting any of them, but they are a way to reward exploration.

However, new abilities like the double jump, will be unlocked each time a final boss is defeated. These abilities are required to finish the game and will help the player to traverse through the map and to reach new levels.

A.8 Graphics

Visually, everything in the game will be made in a pixel art style, it will have simple but appealing graphics. Every animation will be made with a sprite sheet, this will help to make a better looking movement but it will also remark on every action and it will be clearer what is being done.

Aesthetically it will have lots of cyberpunk inspiration, with lots of neon lights and bright colours, but also dark and scary spaces. Each level of the game will be placed in distinctive locations, such as a corporate building or a nightclub.

The color palettes will be the following[A.2]:

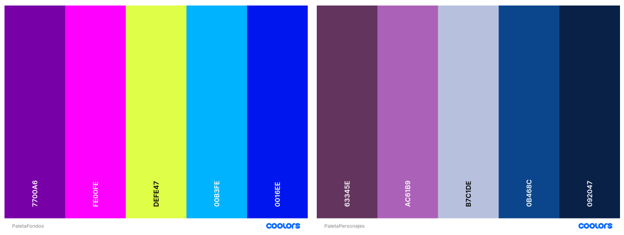


Figure A.2: Color palettes used for backgrounds and for characters

A.9 Tools

1. Unity Main tool used for developing the game, it will be used for building scenes, adding characters and to put everything together.
2. Graphicsgale Tool used mainly for pixel art, it will be used for drawing in a pixel art style.
3. Krita Another tool for drawing, it will be used for altering slightly the pixel art, making a definitive version and to create animations.
4. Visual Studio Tool used for coding.

A.10 Inspiration

Some of the inspirations for the game will be the following

1. Hollow Knight (Team Cherry, 2017)[9]
2. Blade Runner (Ridley Scott, 1982)[13]
3. Katana Zero (Askiisoft, 2019)[19]
4. Unsighted (Studio Pixel Punk, 2021)[20]

Appendix B

Others

B.1 Disctionary

1. Metroidvania. Video game genre that revolves around combat and exploration.
2. Cyberpunk. Science fiction subgenre set in a distopian future.
3. NPC. Non Playable Character, character that appears in the game which can not be controlled bt the player.
4. UI. User Interface, part of the game that gives information to the player.
5. Combo. Combination of multiple attaack that intertwine between themselves.
6. Parry. Deviation of a coming attack.
7. Hub. Central level of a game.
8. Cooldown. Time that a weapon needs to be used again.

B.2 References

For the development of the project I used multiple references and inspirations, my main one, regarding gameplay, was Hollow Knight[9], but for the aesthetic and for the feel of it I got lots of inspiration from the movie Blade Runner 2049[10].

For the moment of development and creation I also did online research, it was my first time doing pixel art, so I had to do lots of investigation to find techniques, learn the best way to do it and learn tips to make it look better, the place that brought me the most information about pixel art was the youtube channel BJGpixel[21]. Regarding of the coding aspect of the project I also looked for references online, the youtube channel Brakeys[22] helped me a lot to understand concepts like how movement and physics work, the other website I frequented the mst was the blog Stack Overflow [23] that helped me to find problems with my code and to understand why some things didn't work.

It was also my first time using LaTeX, so I had to do lots of research to understand how it works, my main references where the Overleaf documentation[14] and vkinght's blog Writing With Latex[15] which taught me how to use useful commands and helped me understand various concepts.

I also got all the sound effect for the game from diverse sound libraries online, the main sites that I used were Gfx Sounds[24] and Pixabay[25], I used this sites to get multiple hit and impact sound effects[26][31][29][28], some shot sound effects [27][30], some death sound effects[34][33] and a jump sound effect[35].

Appendix C

Source code

One big part of the project was writing script and developing code, here are listed some of the scripts that were created for the project.

PlayerAttack script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerAttack : MonoBehaviour
{

    public Transform AttackPosition1;
    public Transform AttackPosition2;
    public LayerMask Target;

    public BoxCollider2D UpwardsAttack;
    public BoxCollider2D FallAttack;
    public BoxCollider2D DashAttack;

    public int Damage;
    public Animator animator;
    Rigidbody2D rb;
    public float m_Thrust = 20f;

    public static PlayerAttack instance;

    public bool isAttacking = false;
    public bool isFalling = false;

    public float direction;
    [SerializeField] private float ChargeTime = 0;
    private float Force;
    public string AttackType = "";
    public PlayerController playerController;

    private void Awake()
    {
        instance = this;
    }

    void Start()
    {
        animator = GetComponent<Animator>();
        rb = GetComponentInParent<Rigidbody2D>();
        UpwardsAttack.enabled = false;
        FallAttack.enabled = false;
        DashAttack.enabled = false;
    }
    void Update()
    {
        PerformAttack();
        if (rb.velocity.y == 0)
        {
            isFalling = false;
            FallAttack.enabled = false;
        }
    }
    public void PerformAttack()
    {

        if (rb.velocity.y != 0)
        {
            if (Input.GetButtonDown("Fire1"))
            {
                isAttacking = true;
                AttackType = "falling";
                Attack(AttackType);
            }

        }
        else if (Input.GetButton("Fire1"))
        {
            ChargeTime += Time.deltaTime;
        }
        else  if (Input.GetButtonUp("Fire1") && !isAttacking)
        {
            isAttacking = true;
            if (AttackPosition1.position.x > AttackPosition2.position.x) direction = -.1f;
            else direction = .1f;
            if (playerController.isRunning) AttackType = "running";
            else if (Input.GetButton("Vertical") && Input.GetAxisRaw("Vertical") > 0)
            AttackType = "upwards";
            else if (ChargeTime >= 1) AttackType = "charged";
            else AttackType = "default";
            Attack(AttackType);
            ChargeTime = 0;
        }
    }

    private void Attack(string type)
    {
        Collider2D[] Enemies;
        playerController.canWalk = false;

        switch (type)
        {
            case "falling":
                FallAttack.enabled = true;
                isFalling = true;
                playerController.CurrentSpeed = 0;
                rb.gravityScale = 10;
                rb.velocity = new Vector2(0, -(rb.velocity.y + ((30 + (0.5f * Time.fixedDeltaTime
                * -rb.gravityScale)) / rb.mass)));
                break;
            case "running":
                DashAttack.enabled = true;
                Force = 5f;
                StartCoroutine(playerController.Dash(Force, .4f, true));
                Enemies = Physics2D.OverlapAreaAll(AttackPosition1.position,
                AttackPosition2.position, Target);
                for (int i = 0; i < Enemies.Length; i++)
                {
                    Enemies[i].GetComponent<Enemy>().RecieveDamage(Damage, gameObject);//,
                    direction);
                }
                break;
            case "upwards":
                UpwardsAttack.enabled = true;
                Debug.Log("upwards");
                rb.velocity = new Vector2(rb.velocity.x, rb.velocity.y + ((20 +
                (.5f * Time.fixedDeltaTime * -6)) / rb.mass));
                break;
            default:
                Debug.Log("Ataque␣normal");
                if (PlayerController.instance.IsGrounded)
                {
                    Force = .5f;
                    StartCoroutine(playerController.Dash(Force, .2f, false));
                    Enemies = Physics2D.OverlapAreaAll(AttackPosition1.position,
                    AttackPosition2.position, Target);
                    for (int i = 0; i < Enemies.Length; i++)
                    {
                        Enemies[i].GetComponent<Enemy>().RecieveDamage(Damage, gameObject);//,
                        direction);
                    }
                }
                break;
        }
    }
}
```

Shoot script

```
      using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Shoot : MonoBehaviour
{
    private Camera mainCamera;
    private Vector3 aimPos;

    public GameObject bullet;
    public GameObject arms;
    public Transform bulletTransformRight;
    public Transform bulletTransformLeft;
    public bool canFire;
    private float timer;
    public float bulletTime;

    public bool isShooting;
    public bool isFacingRight = true;

    public Animator animator;
    public static Shoot instance;

    private void Awake()
    {
        instance = this;
    }

    void Start()
    {
        mainCamera = GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Camera>();
    }

    void Update()
    {
        aimPos = mainCamera.ScreenToWorldPoint(Input.mousePosition);

        Vector3 rotation = aimPos - transform.position;

        float rotationZ = Mathf.Atan2(rotation.y, rotation.x) * Mathf.Rad2Deg;

        transform.rotation = Quaternion.Euler(0, 0, rotationZ);

        if (!canFire)
        {
            timer += Time.deltaTime;
            if(timer > bulletTime)
            {
                canFire = true;
                timer = 0;
            }
        }
        if (Input.GetButton("Fire2") && canFire)
        {
            isShooting = true;
            canFire = false;
            if(isFacingRight) Instantiate(bullet, bulletTransformRight.position,
            Quaternion.identity);
            else Instantiate(bullet, bulletTransformLeft.position, Quaternion.identity);
        }
        if (isShooting)
        {
            arms.SetActive(true);
        }
        else arms.SetActive(false);

        if (rotationZ > -90 && rotationZ < 90)
        {
            isFacingRight = true;
            arms.transform.localScale = new Vector2(Mathf.Abs(arms.transform.localScale.x),
            Mathf.Abs(arms.transform.localScale.y));
        }
        else
        {
            isFacingRight = false;
            arms.transform.localScale = new Vector2(-Mathf.Abs(arms.transform.localScale.x),
            -Mathf.Abs(arms.transform.localScale.y));
        }
    }
}
```

Bullet script

```
      using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class bullet : MonoBehaviour
{
    private Vector3 mouse;
    private Camera mainCamera;
    private Rigidbody2D rb;
    public float force;

    public Enemy enemy;

    public float TrailDelay;
    private float TrailDelaySeconds;
    public GameObject Ghost;
    public bool MakeTrail = true;

    void Start()
    {
        TrailDelaySeconds = TrailDelay;

        mainCamera = GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Camera>();
        rb = GetComponent<Rigidbody2D>();
        mouse = mainCamera.ScreenToWorldPoint(Input.mousePosition);
        Vector3 direction = mouse - transform.position;
        rb.velocity = new Vector2(direction.x, direction.y).normalized * force;
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        int enemyLayer = LayerMask.NameToLayer("Enemy");
        if ((collision.gameObject != null) && (collision.gameObject.layer == enemyLayer))
        {
            enemy = collision.GetComponent<Enemy>();
            enemy.RecieveDamage(1, gameObject);
        }
        Destroy(gameObject);
    }
}
```

Knockback script

```
      using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

public class KnockbackFeedback : MonoBehaviour
{
    [SerializeField]  Rigidbody2D m_Rigidbody;

    [SerializeField] private float knockbackStr = 16f;
    [SerializeField] private float knockbackDelay = .15f;

    public UnityEvent OnBegin;
    public UnityEvent OnEnd;

    public void knockbackFeedback(GameObject sender)
    {
        StopAllCoroutines();
        OnBegin?.Invoke();
        Vector2 direction = (transform.position - sender.transform.position).normalized;
        m_Rigidbody.AddForce(direction * knockbackStr, ForceMode2D.Impulse);
        StartCoroutine(reset());
    }

    private IEnumerator reset()
    {
        yield return new WaitForSeconds(knockbackDelay);
        m_Rigidbody.velocity = Vector3.zero;
        OnEnd?.Invoke();
    }
}
```

Enemy Basic script

```
      using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

public class Enemy : MonoBehaviour
{
    Rigidbody2D m_Rigidbody;
    public int maxHealth;
    public int health;
    public float m_Thrust = 20f;

    public HealthBar healthBar;
    public Collider2D[] hitboxes;

    public BoxCollider2D _collider;
    public LayerMask PlayerLayer;
    public float cooldown;
    public float range;
    public float ColliderDistance;
    public int damage;
    public float CooldownTimer = Mathf.Infinity;

    public bool damagable = true;
    public Health PlayerHealth;
    public KnockbackFeedback knockback;
    public Animator animator;


    void Start()
    {

        m_Rigidbody = GetComponent<Rigidbody2D>();
        healthBar.SetHealth(maxHealth);
    }

    private void OnDisable()
    {
        //Parar animacion
    }

    void Update()
    {
        CooldownTimer += Time.deltaTime;
    }

    public void RecieveDamage(int damage, GameObject attacker)//, float knockback)
    {

        if (damagable)
        {
            CooldownTimer = 0;

            StartCoroutine(damageColor());

            health -= damage;
            if (health <= 0)
            {
                health = 0;
                Destroy(gameObject);
            }
            healthBar.Health(health);
            Shake.Instance.ShakeCamera(10f, .1f);
            knockback.knockbackFeedback(attacker);
        }

    }

    public virtual bool PlayerDetected()
    {
        RaycastHit2D hit = Physics2D.BoxCast(_collider.bounds.center + transform.right *
        range * transform.localScale.x * ColliderDistance,
            new Vector3(_collider.bounds.size.x * range, _collider.bounds.size.y,
            _collider.bounds.size.z), 0, Vector2.left, 0, PlayerLayer);

        if (hit.collider != null)
        {
            PlayerHealth = hit.transform.GetComponent<Health>();
        }

        return hit.collider != null;
    }
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireCube(_collider.bounds.center + transform.right * range *
        transform.localScale.x * ColliderDistance,
            new Vector3(_collider.bounds.size.x * range, _collider.bounds.size.y,
            _collider.bounds.size.z));
    }

    private void DamagePlayer()
    {
        if (PlayerDetected())
        {
            PlayerHealth.RecieveDamage(damage, gameObject);
        }
    }

    IEnumerator damageColor()
    {
        SpriteRenderer sprite = GetComponent<SpriteRenderer>();
        sprite.color = Color.red;
        yield return new WaitForSeconds(0.3f);
        sprite.color = Color.white;
    }
}
```

First Boss script

```
      using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BossBehaviour : Enemy
{
    [SerializeField] private float timer = 0f;
    [SerializeField] private float randTime;
    [SerializeField] private float randAttack = 0;
    public bool isPunchAttacking = false;
    public bool isLaserAttacking = false;
    public bool isSmashAttacking = false;
    private bool isCharging = true;
    public bool isAttacking = false;


    public GameObject arms;
    public GameObject player;
    public GameObject rotationPoint;

    public GameObject SmashAttackRightHitBox;
    public GameObject SmashAttackLeftHitBox;
    public GameObject LaserHitBox;
    public GameObject PunchAttackHitBox;

    public Transform LaserPoint;
    private Vector3 PlayerPosition;
    public LineRenderer Laser;

    private float rotationZ;

    public Animator animator;
    public static BossBehaviour instance;

    private void Awake()
    {
        instance = this;
    }

    void Start()
    {
        randTime = Random.Range(2f, 5f);
    }

    void Update()
    {
        if(isCharging) timer += Time.deltaTime;

        if(timer >= randTime)
        {
            randAttack = Random.Range(1, 4);
            isAttacking = true;
            switch (randAttack)
            {
                case 1:
                    PunchAttack();
                    break;
                case 2:
                    SmashAttack();
                    break;
                default:
                    isCharging = false;
                    timer = 0;
                    StartCoroutine(LaserAttack());
                    break;
            }
        }

        Vector3 rotation = player.transform.position - rotationPoint.transform.position;

        rotationZ = Mathf.Atan2(rotation.y, rotation.x) * Mathf.Rad2Deg;

        if (rotationZ < -90 || rotationZ > 90)
        {
            if(!isAttacking) transform.localScale = new Vector3(-9, 9, 1);
            arms.transform.localScale = new Vector3(-1, -1, 1);
        }
        else
        {
            if (!isAttacking) transform.localScale = new Vector3(9, 9, 1);
            arms.transform.localScale = new Vector3(1, 1, 1);
        }

        if (isLaserAttacking) arms.SetActive(true);
        else arms.SetActive(false);
    }

    void PunchAttack()
    {
        isAttacking = true;
        PunchAttackHitBox.SetActive(true);
        isPunchAttacking = true;
        isLaserAttacking = false;
        timer = 0;
        randTime = Random.Range(2f, 5f);
    }

    IEnumerator LaserAttack()
    {
        isLaserAttacking = true;
        isAttacking = true;
        rotationPoint.transform.rotation = Quaternion.Euler(0, 0, rotationZ);
        PlayerPosition = player.transform.position;
        yield return new WaitForSeconds(1f);

        Laser.SetPosition(0, LaserPoint.position);
        Laser.SetPosition(1, PlayerPosition);

        Laser.enabled = true;
        yield return new WaitForSeconds(0.75f);
        Laser.enabled = false;

        randTime = Random.Range(2f, 5f);
        isCharging = true;

        yield return new WaitForSeconds(randTime);
        isAttacking = false;
    }

    void SmashAttack()
    {
        isAttacking = true;
        SmashAttackRightHitBox.SetActive(true);
        SmashAttackLeftHitBox.SetActive(true);
        isSmashAttacking = true;
        isLaserAttacking = false;
        timer = 0;
        randTime = Random.Range(2f, 5f);
    }
}
```

First Boss State Machine script

```
      using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BossIdleBehaviour : StateMachineBehaviour
{
    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
    //override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    //{
    //
    //}

    // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        if (BossBehaviour.instance.isPunchAttacking)
        {
            BossBehaviour.instance.animator.Play("Attack1");
        }
        else if (BossBehaviour.instance.isSmashAttacking)
        {
            BossBehaviour.instance.animator.Play("SmashAttack");
        }
        else if (BossBehaviour.instance.isLaserAttacking)
        {
            BossBehaviour.instance.animator.Play("LaserAttack");
        }
    }

    // OnStateExit is called when a transition ends and the state machine finishes evaluating this state
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        BossBehaviour.instance.isPunchAttacking = false;
        BossBehaviour.instance.isSmashAttacking = false;
        //BossBehaviour.instance.isAttacking = false;
    }

    // OnStateMove is called right after Animator.OnAnimatorMove()
    //override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    //{
    //    // Implement code that processes and affects root motion
    //}

    // OnStateIK is called right after Animator.OnAnimatorIK()
    //override public void OnStateIK(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    //{
    //    // Implement code that sets up animation IK (inverse kinematics)
    //}
}
```

Health script

```
      using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Health : MonoBehaviour
{
    public int maxHealth;
    public int health;

    public KnockbackFeedback knockback;

    private void Start()
    {
        health = maxHealth;
    }
    public virtual void RecieveDamage(int damage, GameObject attacker)
    {
        health -= damage;
        StartCoroutine(damageColor());
        knockback.knockbackFeedback(attacker);
    }

    IEnumerator damageColor()
    {
        SpriteRenderer sprite = GetComponent<SpriteRenderer>();
        sprite.color = Color.red;
        yield return new WaitForSeconds(0.3f);
        sprite.color = Color.white;
    }
}
```