Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Bachelor Degree in Informatics Engineering

Creation of a 3D Game: Mario Rolling Sky

Project Report

Arnau Arqué and Daniel Esquina

Professor: Jesus Alonso Alonso

December 19, 2019

Contents

List of Figures

List of Tables

1 Introduction

The present project arises as an integral part of the Video Games course within the Computer Engineering degree program at the Faculty of Informatics in Barcelona (Polytechnic University of Catalonia). This initiative exemplifies the convergence of technological innovation and artistic creativity, reflecting the advancements in the video game industry and the capabilities of modern students.

In this document, we provide a detailed explanation of the conception and execution of an exciting 3D game, serving as a tribute to the well-known game Rolling Sky. This project fits seamlessly within the educational framework of this course, providing a platform to explore the intricate connections between programming, game design, and user experience.

We delve into the core features of our tribute game, drawing inspiration from the unforgettable characters and enemies of Nintendo's Mario saga. Through the use of Unity, we have created three unique levels of Rolling Sky that incorporate these iconic elements, while also allowing our creativity to reinvent gameplay mechanics and surprise players.

We thoroughly analyze the dynamics of gameplay, the challenges, and the unique twists we have incorporated to maintain excitement and player engagement at each level. Furthermore, we provide detailed instructions on how to fully enjoy this gaming experience and overcome the challenges we present.

A crucial section of this document outlines the detailed description of the development process. We explore the stages we have followed to bring our game to life, from initial conception to implementation and refinement. This will offer a comprehensive view of the dedication and teamwork we have invested in this project.

Finally, we will share the methodology we have adopted to tackle this challenge and the conclusions drawn from our experience. These reflections distill the lessons learned and achievements attained, as well as the future opportunities for improvement and the continuation of this exciting journey of game creation.

2 Reference game

The reference game is Rolling Sky, developed by Cheetah Mobile. The distribution is handled by Cheetah Mobile in collaboration with Rising Win Tech and TurboChili. It is available on the Android, iOS, and Switch platforms. It carries a PEGI rating of 3/4+ and boasts over a hundred million downloads, along with more than one million reviews, resulting in an average rating of 4.2/5. A summary of the outlined specifications can be found in Tables 2.1 and 2.2.

Table 2.1: Summary table of Rolling Sky game specifications.

Concept	Specification
Developer	Cheeta Mobile
Distributor	Cheetah Mobile, Rising Win Tech, and TurboChili
Platforms	Android, iOS, and Switch
PEGI	3/4+
Downloads	+100 million
Reviews	+1 million
Average Rating	4.2/5
Official Website	`https://cheetahgames.cmcm.com/RollingSky`
Trailer	`https://youtu.be/4LLTqh0HtzQ`
Gameplay	`https://youtu.be/fdgbYMX9wvA`
Wiki Entry	`https://rolling-sky.fandom.com/wiki/Rolling_Sky`

Source: Own elaboration.

Table 2.2: Summary table of Rolling Sky game on different platforms.

Concept	Google PlayStore	Apple Store	Nintendo e-Shop
Release Date	26/01/2016	20/01/2016	12/04/2019
Latest Version	3.3.8.1	2.3.7	–
Price (EUR)	Free	Free	9.99
Size (MB)	73	208.7	288

Source: Own elaboration.

3 Game Description

In this game, we aimed to pay tribute not only to the proposed Rolling Sky but also to the iconic Mario franchise. All game entities and level designs were crafted with the gameplay of titles like Mario Kart or New Super Mario Bros in mind. This endeavor allowed us to seamlessly blend two games of distinctly different styles into the outcome of our project.

Within this game, players have the opportunity to engage in three distinct levels. The thematic design of these levels draws inspiration from the classic structure found in New Super Mario Bros games—comprising grassy terrains, sky-themed sections, and culminating with a boss encounter. The game's levels progressively increase in difficulty. All three levels are aptly named after Mario Kart circuits, and their chosen themes closely resemble the energetic rhythm of Rolling Sky's music. Other functionalities, such as menus, have also been meticulously developed with a keen eye for the distinctive style of the Mario franchise. The main menu distinctly embodies the aesthetic of Mario Kart, while the death menu finds its foundation in Super Mario 64. The start and pause menus bear a closer resemblance to the original Rolling Sky game, though adapted to ensure visual coherence.

3.1 Objectives

The main objective of each level is to reach the end, where a flagpole, a hallmark of Mario games, is prominently positioned (Figure 3.1).

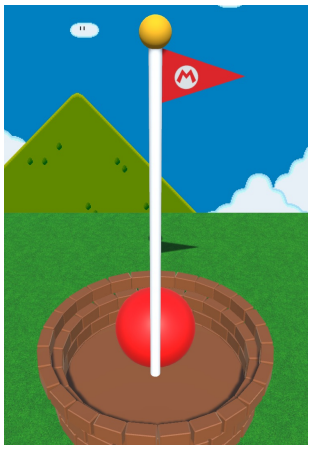
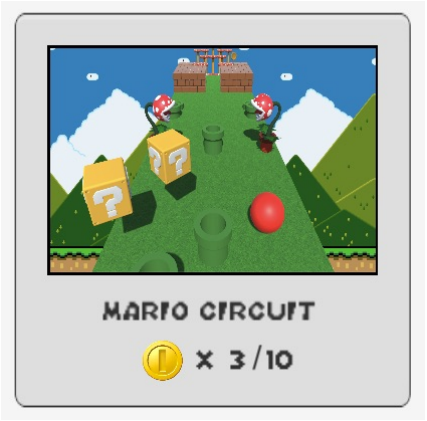


Figure 3.1: *Flagpole* in the game positioned at the end of level 1.

Just like the gems in the original Rolling Sky, strategically placed coins have been employed in our game. These coins serve to encourage exploration of non-traditional paths and provide an additional gameplay objective, thereby enhancing the replayability of levels. The maximum coin count achieved in a level is displayed both in the start menu (Figure 3.2a) of each level and on the main menu (Figure 3.2b). Conversely, the pause menu indicates the current coin count.

(a) Coins in the start menu.

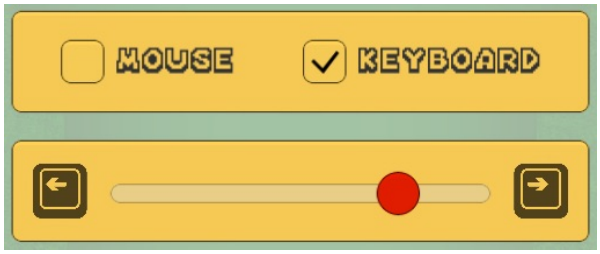


(b) Coins from level 1 on the main menu.

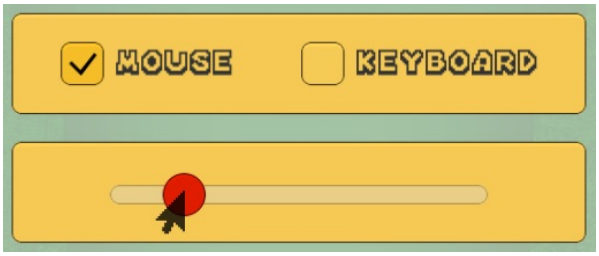
Figure 3.2: Coin count in menus.

3.2 Instructions

There are two ways to control the lateral movement of the ball: using the keyboard's left/right arrows or dragging the ball with the mouse. These options are selected at the beginning of each level, and their respective functionalities are demonstrated, as depicted in Figure 3.3.



(a) Keyboard control option.



(b) Mouse control option.

Figure 3.3: Control selection screen.

All menu transitions and interactions within the game are mouse-driven. These interactions encompass level selection, starting, pausing, and exiting the game. To facilitate testing and experimentation with the game's mechanics, shortcuts have been added that offer functionalities not intended for public gameplay. A summary of these commands is provided below:

`c` Disable ball collisions.

`m` Halt the movement of the ball, camera, and background.

`↑` Move the ball upward on the y-axis.

`↓` Lower the ball to its default y-axis height.

`1` Position the ball, camera, and background at 1/5 of the z-axis map.[a]

`2` Position the ball, camera, and background at 2/5 of the z-axis map.

`3` Position the ball, camera, and background at 3/5 of the z-axis map.

`4` Position the ball, camera, and background at 4/5 of the z-axis map.

`5` Position the ball, camera, and background at 5/5 of the z-axis map.

3.3 Game Flow

In Figure 3.4, we can observe the initial screen presented upon launching the application. This screen permits the selection of one of the three levels or exiting the game. Upon selecting a level, we are redirected to its corresponding level start menu, as illustrated in Figure 3.5.

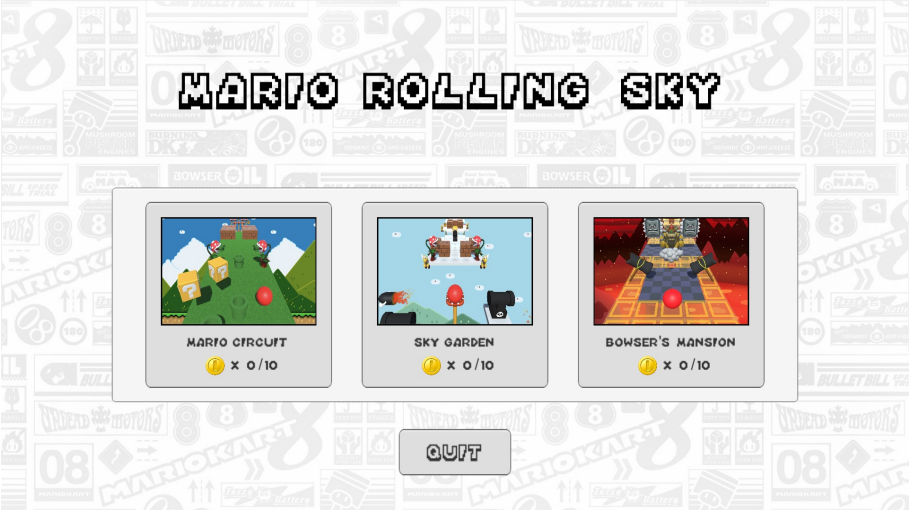


Figure 3.4: Main menu.

Figure 3.5: Level 1 start menu.

In the menu depicted in Figure 3.5, we find the name of the circuit being played and the maximum coins attained. In the upper left corner, a button in the shape of a house allows us to return to the main menu. The mouse and keyboard toggle enables the selection of control method for the level. Upon pressing the start button while on the first two levels, we encounter the countdown and begin gameplay, as seen in Figure 3.6.

While a level is in progress, a pause button can be found in the upper left corner, granting access to the game's pause menu, illustrated in Figure 3.7. Upon colliding with an entity or falling into the void, a transition occurs, leading us to the death menu, depicted in Figure 3.8. Upon completing a level, the game returns us to the main menu.

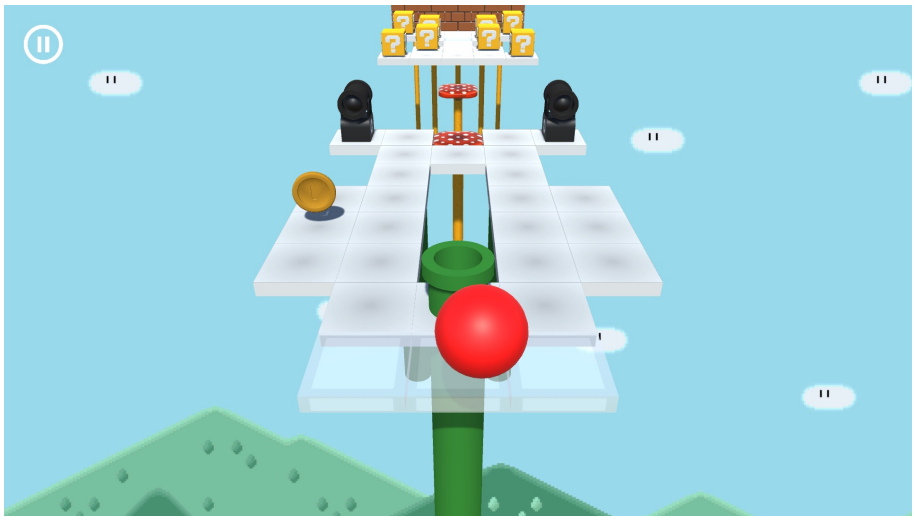


Figure 3.6: In-game on level 2.

Figure 3.7: Pause menu.

Upon death in any of the levels, we are directed to a death menu, similar to Figure 3.8. The button located in the upper left corner allows us to return to the main menu, while the restart button takes us to the level start menu, enabling the selection of controls and level retry.

3.4 Tiles

In this section, we will provide a detailed description of the different types of tiles we have used during the creation and design of the game.

3.4.1 Basic Tile

This is the default tile used throughout all levels. It is a basic, immobile tile that functions as ground in the game.

For Level 1, we used a texture with a basic green color. We combined it with a grass texture extracted from an original Mario game to create a blended effect (Figure 3.9a). For Level 2, which is an aerial level, we opted for a gradient texture of gray tones to simulate the colors of a cloud (Figure 3.9b).

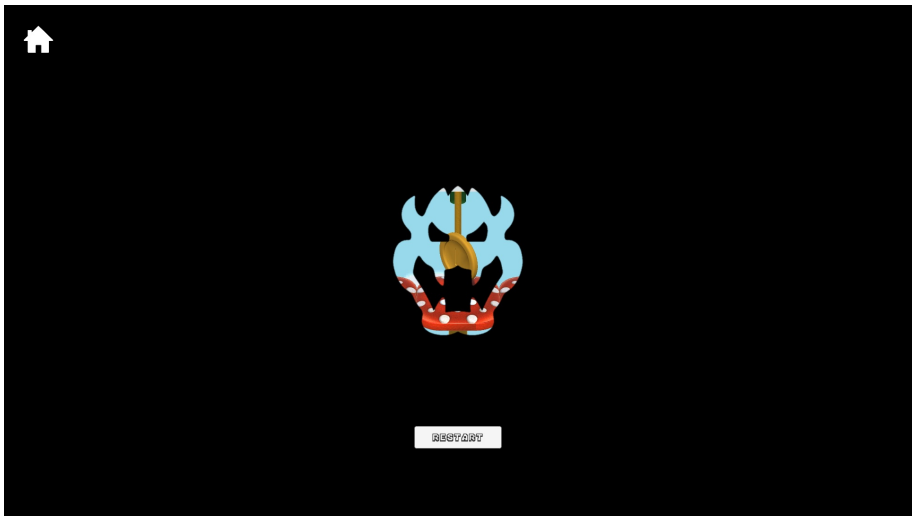
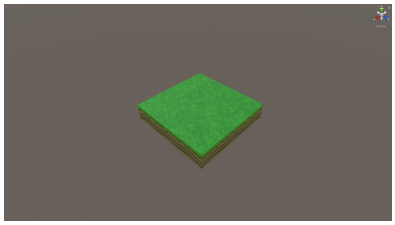
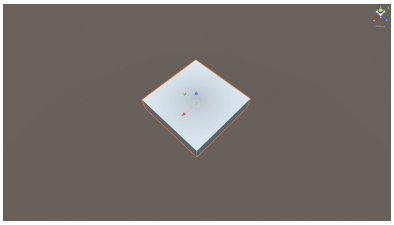


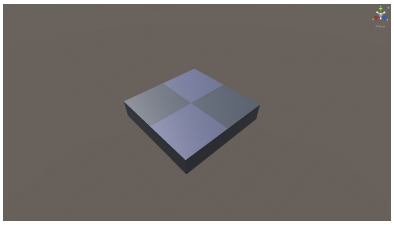
Figure 3.8: Death menu.



(a) Basic tile of Level 1.



(b) Basic tile of Level 2.



(c) Basic tile of Level 3.

Figure 3.9: Basic tiles.

For Level 3, we used a texture resembling a chessboard, with shades of dark and light purple. We drew inspiration from color palettes found in boss levels of Super Mario (Figure 3.9c).

3.4.2 Falling Basic Tile

This tile has the exact same textures as a Basic Tile. The difference lies in its functionality. These tiles do not appear at ground level. When the player approaches a Falling Basic Tile, it appears as a hole, creating the illusion of a pitfall. Suddenly, the tile falls rapidly from the sky, filling the path. Falling Basic Tiles are used in Levels 2 and 3.

3.4.3 Jumping Tile

The Jumping Tile propels the ball, causing it to execute a jump of 4 tiles in distance (relative to the Z-axis). Since no force is used in the game's mechanics, we simulate the jump using the following equation:

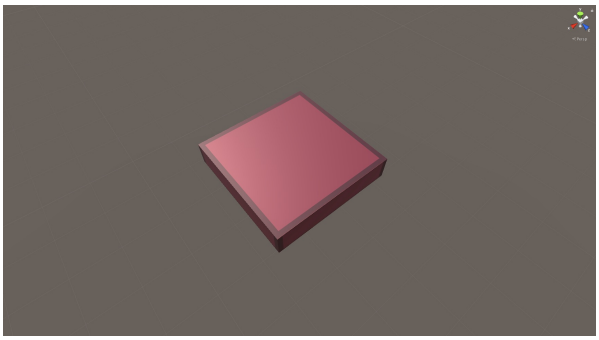
$$y = f(z) = -\frac{1}{2}(z-2)^2 + \frac{5}{2}$$

For Levels 1 and 2, we created a special jumping tile resembling the mushrooms that provide jumping abilities in Mario Kart (Figure 3.10a). For Level 3, we designed a jumping tile with a purple color and dark margins (Figure 3.10b).

3.4.4 Falling Tile

Falling Tiles are a type of tile with translucent textures. When the ball passes over them, they enter freefall. However, they do not cause the ball to fall; they create an interesting visual effect.

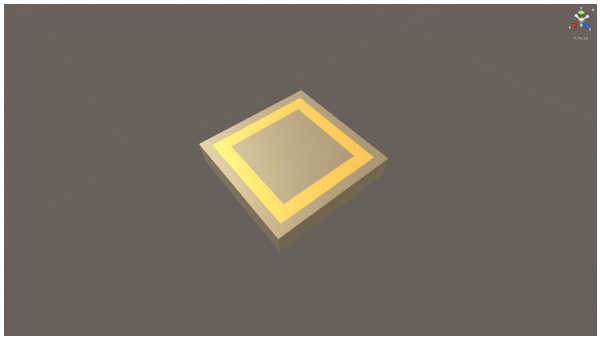
(a) Mushroom Jumping Tile.



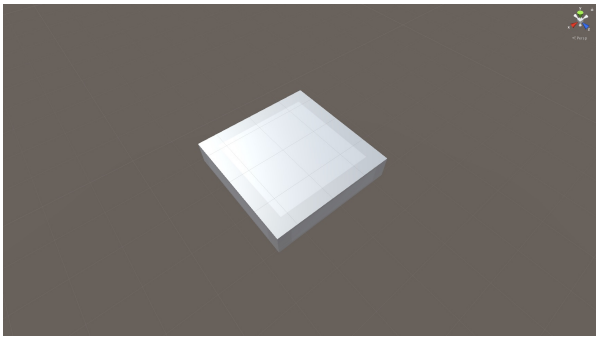
(b) Jumping Tile of Level 3.

Figure 3.10: Jumping tiles.

For Levels 1 and 3, we used falling tiles with translucent orange textures and darker orange margins (Figure 3.11a). For Level 2, we employed similar tiles but with white tones, maintaining the celestial color palette of the level (Figure 3.11b).



(a) Falling Tiles of Levels 1 and 3.



(b) Falling Tile of Level 2.

Figure 3.11: Falling tiles.

3.4.5 Forward Tile

Forward Tiles are tiles with a hole right in front of them. When the ball passes over them, they move one unit forward, preventing the player from falling.

All three levels share the same texture applied to these tiles. Additionally, when the ball is on these tiles, the arrow indicating the direction of movement changes color to white (Figure 3.12a).

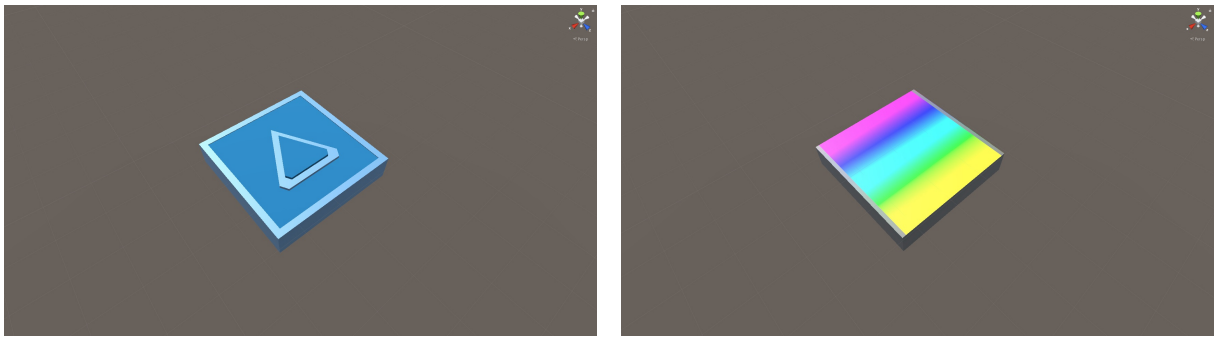
3.4.6 Boost Tile

Boost Tiles act as boosters for the ball, propelling it when it passes over them. They are used in Level 1 and feature a rainbow texture that moves over time (Figure 3.12b). This simulates the effect of boosters found in the Mario Kart game. Additionally, we added particles to simulate a propulsion effect on the camera.

3.5 Game Entities

As explained in previous sections, with this game, we wanted to pay homage not only to RollingSky but also to various games in the Mario universe, such as Super Mario Bros, Mario Kart, etc. Thus, most of the entities in the game are inspired by the iconic Mario series.

While we did find various 3D models on the internet that could have been useful, a significant portion of the collision objects were created by us using the Blender 3D editing software. This



(a) Forward Tile. (b) Boost Tile.

Figure 3.12: Forward and Boost tiles.

was a challenge, as none of our team members had prior experience with Blender, and we had to quickly learn to achieve satisfactory results.

In this section, we will explain the functionalities of different objects and the steps we followed to create and design them using Blender.

3.5.1 The First Object: Mystery Block

The Mystery Block was the first object we created using Blender. It had a simple shape, making it a suitable starting point to become acquainted with the editing software.

We began by creating a cube with a size of 1x1 and applied a bevel to the edges to achieve rounded corners (Figure 3.14a). Next, we applied a smooth shading transformation to obtain a rounded appearance (Figure 3.14b). With these steps, we had the base of the Mystery Block.

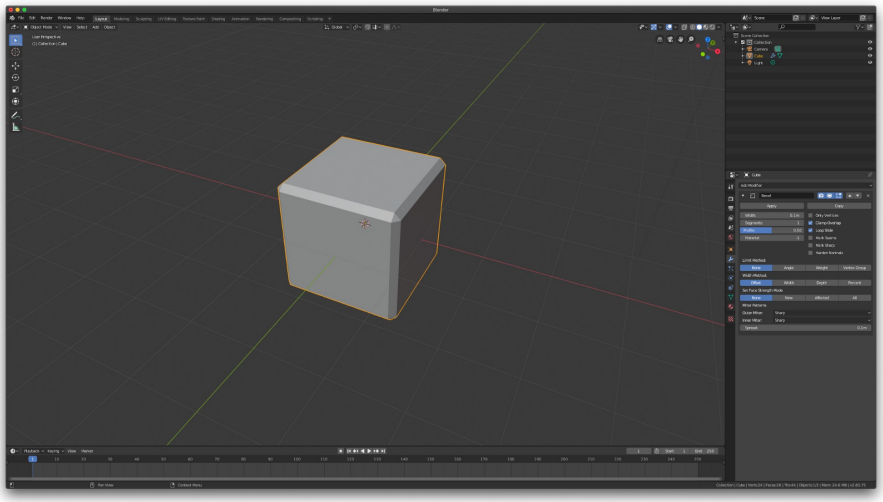
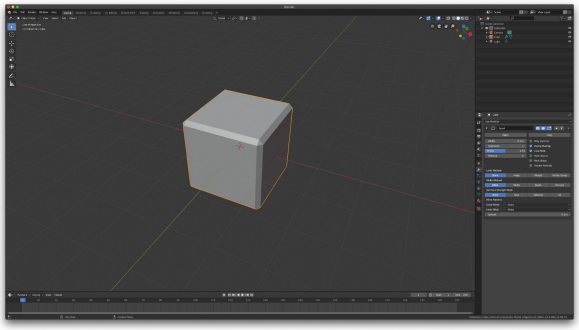


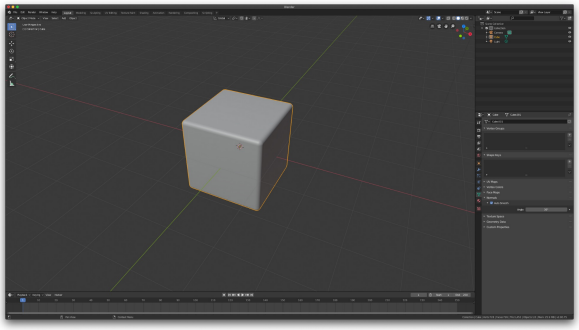
Figure 3.13: Cube with beveled edges.

We then took a template of the question mark icon to be displayed on each face (excluding the top and bottom) of the block. Starting with another cube, we applied various transformation operations to obtain the desired question mark shape (Figure 3.15a).

Finally, we duplicated the question mark symbol and attached it to each of the side faces of the cube we created earlier. We also created four holes at the corners of each lateral face and assigned the appropriate materials to achieve the desired appearance (Figure 3.15b).

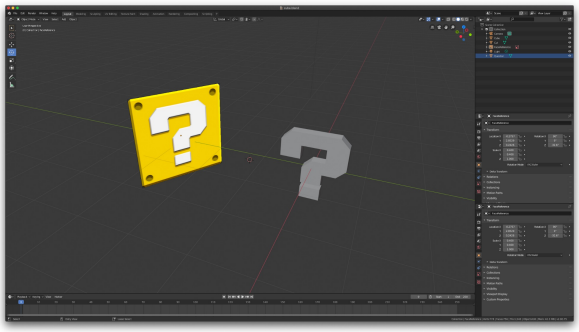


(a) Cube with beveled edges.

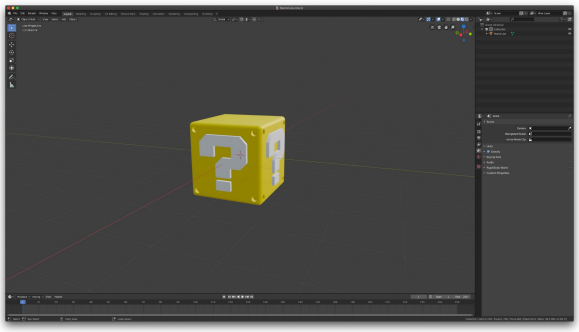


(b) Cube with smooth beveled edges.

Figure 3.14: Basic beveled cubes.



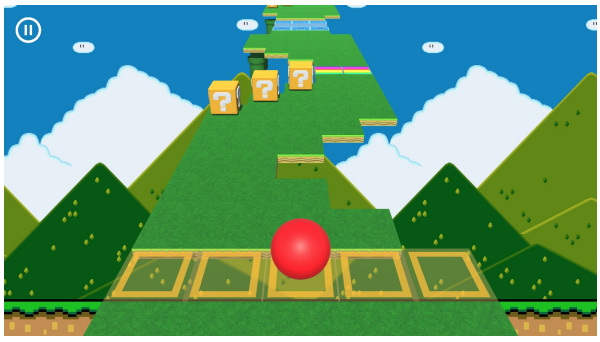
(a) Template and question mark object.



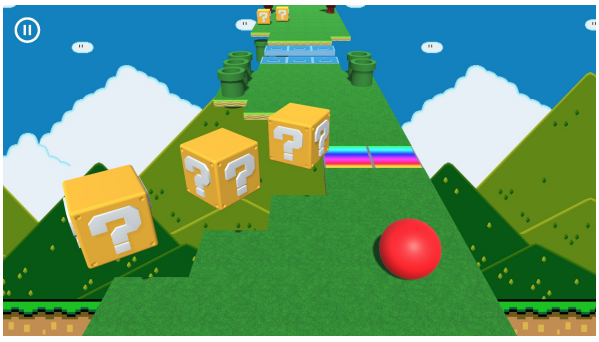
(b) Mystery block completed.

Figure 3.15: Developement of the mystery block.

The functionality we envisioned for this block was purely a distraction. We positioned the block on the ground of the map (Figure 3.16a), giving the player the impression of an obstacle. As the ball approached the object, it would elevate and rotate about its Y-axis, creating a passage for the player (Figure 3.16b).



(a) Block on the ground.

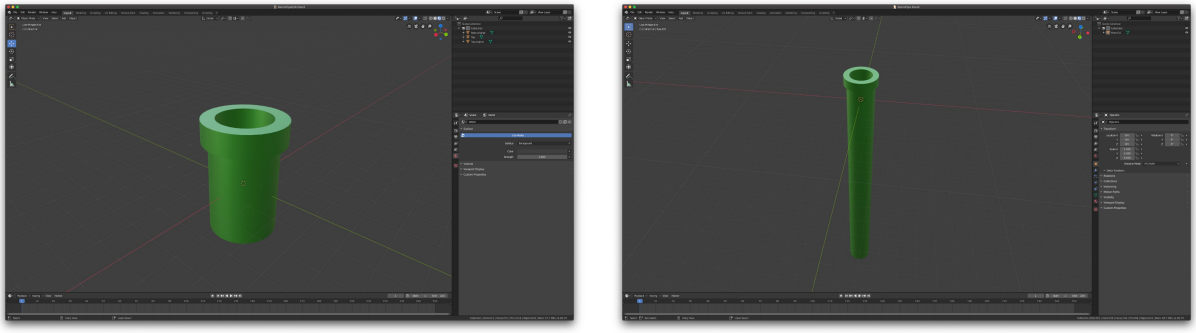


(b) Activation of the Mystery Block.

Figure 3.16: Mystery Block in-game.

3.5.2 Pipe

In our RollingSky-based game with a Mario theme, the iconic pipe could not be missing. Its construction was simple. We just needed to create two cylinders of different radii and stack them on top of each other to simulate the shape of a pipe, then color them green (Figure 3.17a). We created two versions: a short pipe and a longer one (Figure 3.17b) for the second level.



(a) Short pipe.

(b) Long pipe.

Figure 3.17: Pipe objects.

Its functionality is that of a collidable object that emerges from below the ground. At first, it seems that the path is clear, but when the ball approaches a certain distance, the pipe rises from underground, obstructing the way. If the ball collides with the pipe, the game ends. In Figure 3.18, you can see the pipes that have already emerged at the forefront of the image, while in the background, you can see the ones hidden underground.

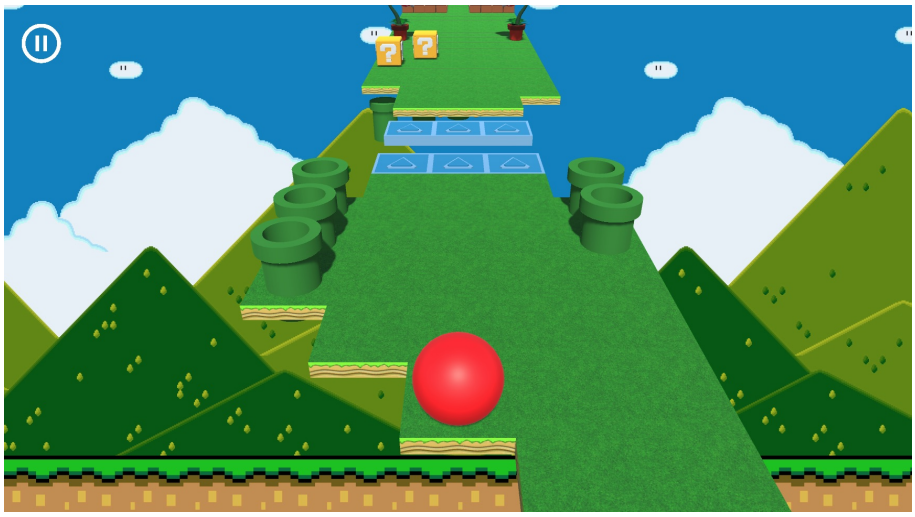


Figure 3.18: Raised pipes in the foreground and hidden pipes in the background.

3.5.3 Hammer

This is one of the few objects that we decided to obtain from the internet. The reason is mainly that it was very well made, and replicating it would have been much more complicated.

The object is a hammer that appears in Super Mario Bros games. It is an adaptation of the hammers thrown by Koopa Troopa enemies from a cloud. While we didn't create the object ourselves, we did edit its animation. This also served as our first introduction to Blender's animation editor (Figure 3.19). The animation involves dropping the hammer from a vertical position to the ground. Once it touches the ground, the hammer bounces slightly to create a more natural free-fall effect.

The functionality associated with the hammer, as implied in the previous paragraph, is to remain in a vertical position until the ball approaches. Then, the hammer is released in free-fall

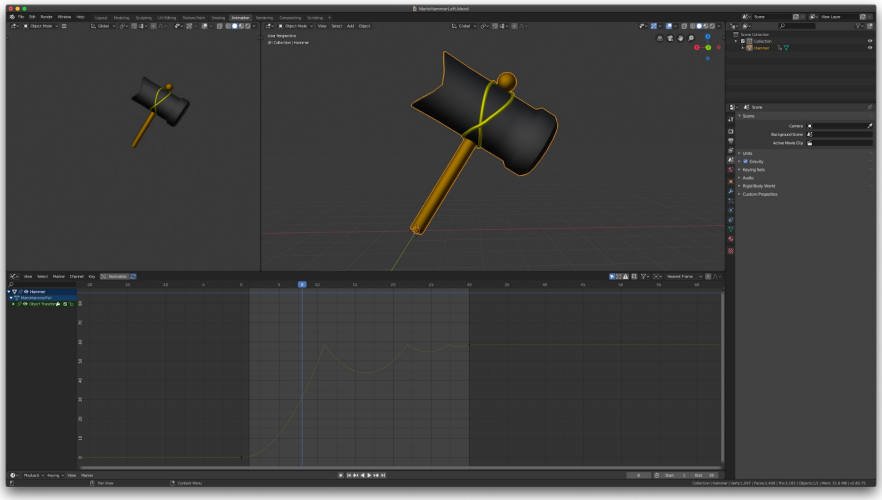


Figure 3.19: Bouncing animation of the hammer.

until it reaches the ground and bounces a little. This way, the player initially thinks that there is an available block that, once the hammer has fallen, becomes occupied by the object in a horizontal position. The collidable object occupies one tile when it is in a vertical position and, when it falls, occupies the previous tile and an additional one to the left (or right, depending on its position). If the ball collides with the hammer, the game ends.

3.5.4 Bullet-Bill and Bill-Blaster

The creation of this object was one of the first significant challenges we encountered during the project. Various Bullet Bill models were available on the internet, but they had very low quality. On the other hand, we couldn't find a Bill Blaster model at all.

This object combines the creation of objects from basic 3D geometric volumes with animation using Blender. Furthermore, once we imported them into Unity, we generated particle systems to simulate the fire that comes out of the Bullet Bill after it has been fired from the Bill Blaster (Figure 3.20).

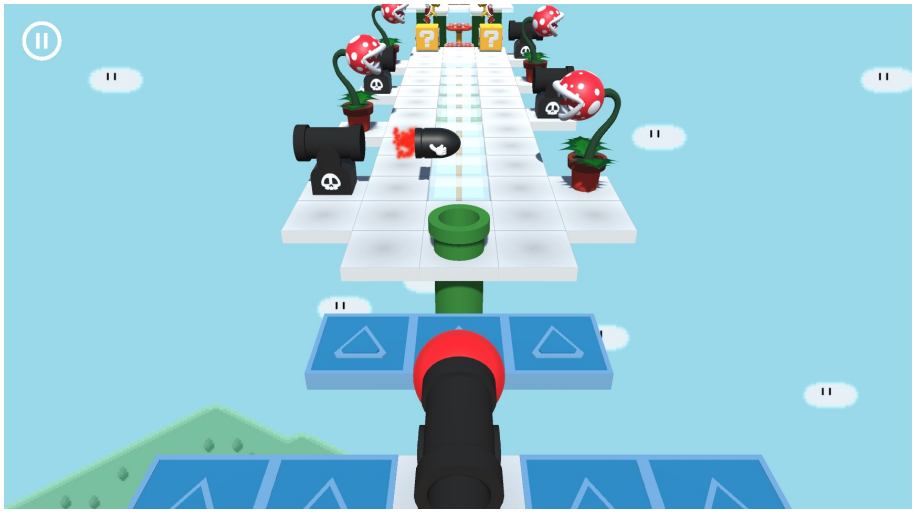
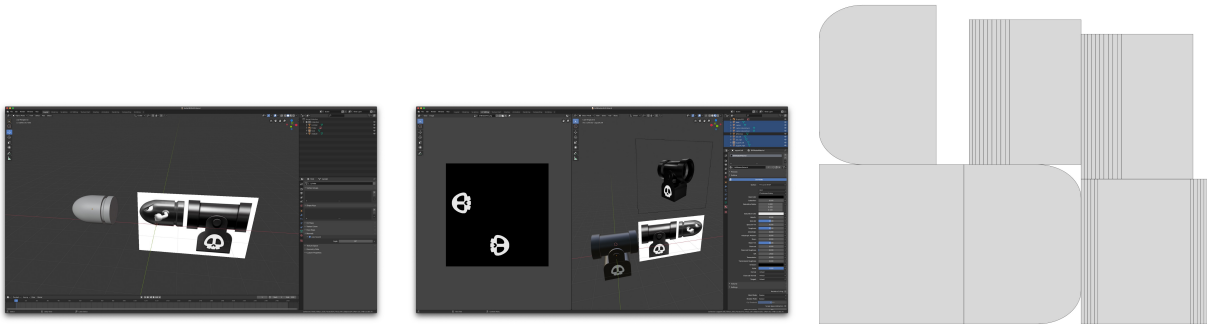


Figure 3.20: Particle system of the Bullet Bill.

As we mentioned earlier, we created both objects from basic 3D geometric volumes. We won't explain the process step by step because it would be too extensive, but we've included some images (Figure 3.21) that show the process. In addition to creating the objects, it was also the first time we created textures for them (for the Bill Blaster). To do this, we had to unwrap (Figure 3.21c) the volume and create the texture image (also shown in Figure 3.21b) separately using the Gimp program.

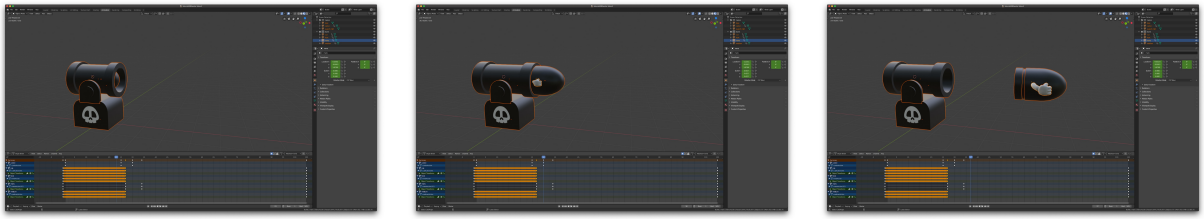


(a) Bullet Bill creation. (b) Bill Blaster creation. (c) Bill Blaster unwrap.

Figure 3.21: Bullet Bill and Bill Blaster creation in Blender.

The animation we created was much more complex than any we had done before. It involved the Bill Blaster contracting until it reached a point where it would expand again, simulating the firing of a bullet. When the Bill Blaster executed the firing animation, the Bullet Bill had to be launched from inside the cannon and activate its particle system. The bullet travels a certain distance until it comes to a stop (Figure 3.22).

The object's functionality is that, when the ball approaches, the Bill Blaster fires a Bullet Bill. If the Bullet Bill collides with the ball, the game ends. We decided to apply rotations to the object so that the bullet could move from left to right, right to left, and in the opposite direction of the ball's movement.



(a) Start of the animation. (b) Intermediate point. (c) Advanced animation.

Figure 3.22: Animation of the Bill Blaster shooting a Bullet Bill.

3.5.5 Piranha Plant

Another classic from the Mario universe is the Piranha Plant. We found this object on the internet, and it was so well-made that we couldn't resist using it. To leave our mark, we decided to animate it.

The piranha animation was our first introduction to Blender's bone (skeleton) utility. This feature allows you to create a skeletal structure in which each bone is associated with a set of triangles in the mesh that forms the object. Blender automatically associates the bones with

the different triangles and vertices of the mesh based on their proximity. However, after the automatic weight distribution, we had to adjust some weights to ensure that the bone movements did not affect unwanted parts of the object.

The piranha's skeleton is relatively simple. It consists of a spinal column and different bones to allow mouth movement (Figure 3.23). Learning to use Inverse Kinematics (IK) was considerably challenging. This feature generates a bone that, when movement and/or rotations are applied, propagates the natural movement of a bone structure through the connected bones. Despite the added complexity, applying IK greatly facilitated the subsequent animation of the object.

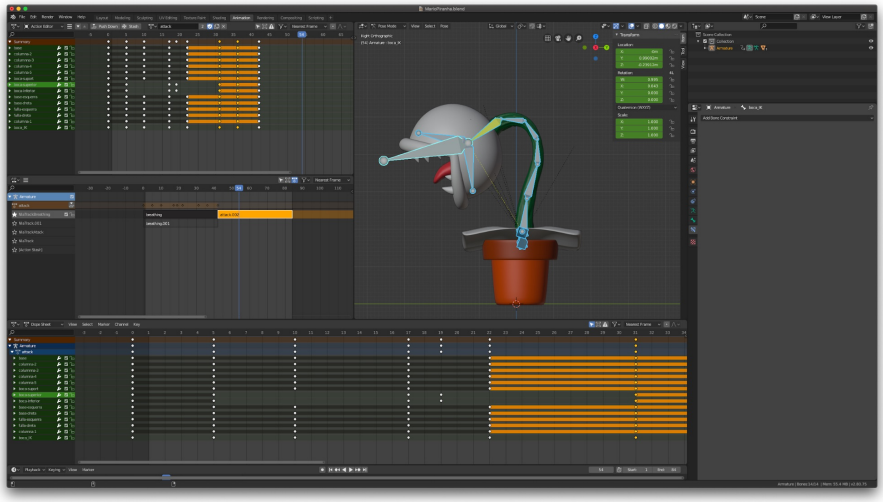


Figure 3.23: Piranha Plant animation.

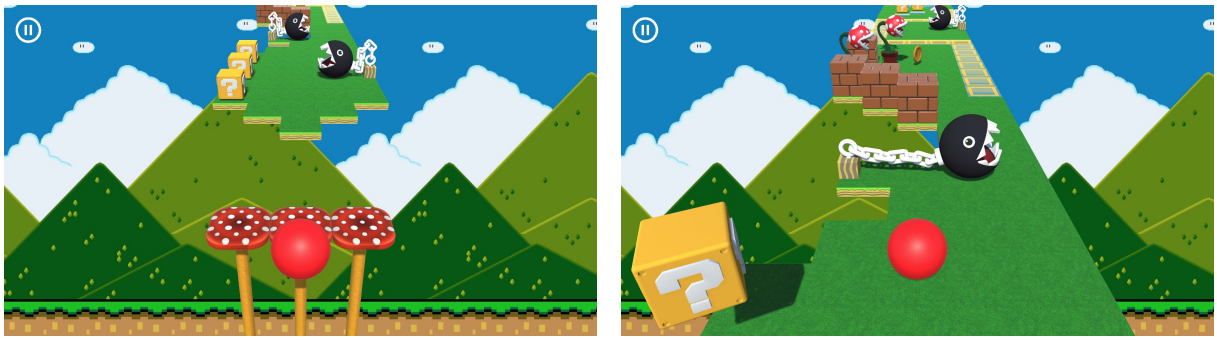
The animation of the piranha involved two parts: In the first part, the object is in constant movement, simulating breathing. The second part involves the plant leaning towards the ball with its mouth open, making a biting gesture, when the ball approaches. The functionality is as described for this second animation.

Initially, we decided that the plant would be an object that, if the ball approached sufficiently, would bite it, and if there was a collision, the game would end. However, after testing it in the game, we found that the animation was too fast, and the player didn't have enough reaction time to avoid the collision. Therefore, we decided that the attacking animation would serve as a distraction. Nevertheless, the ball can collide with the piranha's test, and if that happens, the game ends.

3.5.6 Chomp

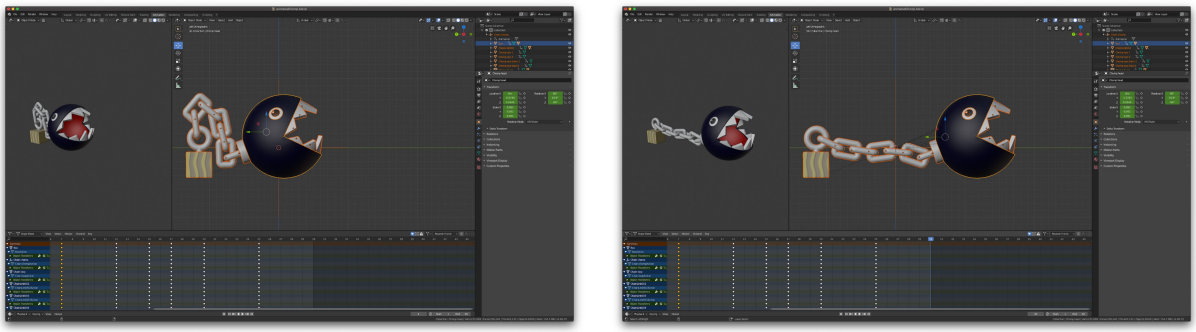
The iconic Chomp! How many times has it surprised us with a sudden bite while playing Mario Kart? Going from first to last place in a matter of seconds... We found a high-quality 3D model of the object and didn't waste the opportunity. The main challenge was animating the chains. The functionality we had in mind was the original one: the Chomp would be stationary (Figure 3.24a), and when the ball approaches, it would move quickly to bite it (Figure 3.24b). In case of collision, the game would end.

As we mentioned earlier, the difficulty in this case was animating the object (Figure 3.25) so that all the chains moved correctly and simulated a "pull-and-release" motion of the chains. Again, we considered using inverse kinematics, but for various reasons, we concluded that it would be more complicated than performing the animation chain by chain.



(a) Chomp stationary when the player is far away. (b) Chomp attacking as the player approaches.

Figure 3.24: Chomp states.



(a) Stationary Chomp animation. (b) Attacking Chomp animation.

Figure 3.25: Chomp animations in Blender.

3.5.7 Thwomp

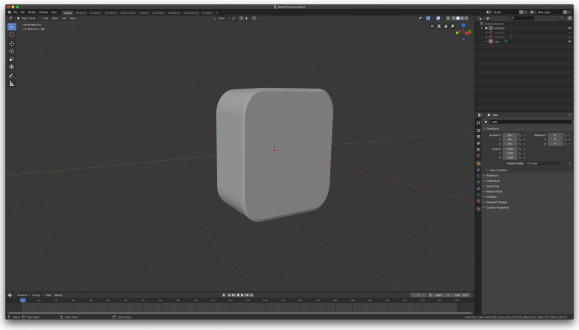
This entity was particularly difficult to create. In this case, not so much for the animation, but for the high number of parts that make it up and for texturing them. It was very challenging for us to find the appropriate materials, but the final result has been satisfying.

The Thwomp consists of a cube with beveled edges (without shade-smooth) that serves as the support for the other parts (Figure 3.26a). It also has a smaller second cube that protrudes from the front and back in the characteristic shape of the original Mario Thwomp (Figure 3.26b). Finally, it has two rows of spikes that surround the top, bottom, and lateral faces of the first cube (Figure 3.26c). To texture the object, we unwrapped the entire figure and applied strategic parts of an original Thwomp texture from a Mario game (Figure 3.26d).

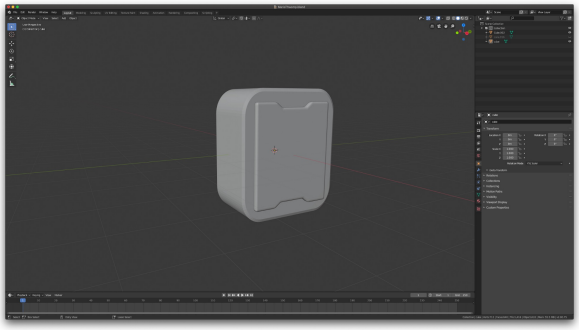
The functionality is the same as the original Mario game. The Thwomp is elevated above the Y-axis, and when the ball approaches, it is released in free fall until it touches the ground, obstructing the player's path. In case of collision, the game ends. We didn't need to animate the object because the movement could be easily generated through a script that modifies the Y-axis position of the entity, following the formula for free fall.

3.5.8 Koopa Troopa and Blue Shell

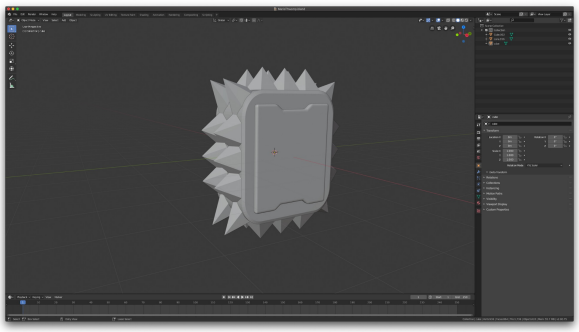
For the second level, which, as we explained before, is set in an aerial Mario level, we decided to have some entities or objects with wings that simulate flying around the map while also serving as collidable objects. A classic example that fits these characteristics is the Koopa Troopa figure and its shells. Specifically, that infamous blue shell that ruins all Mario Kart games... Both ob-



(a) Basic Thwomp block.



(b) Thwomp with beveled face.



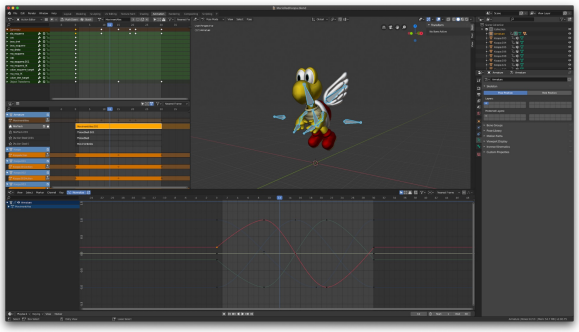
(c) Thwomp with spikes.

(d) Textured Thwomp.

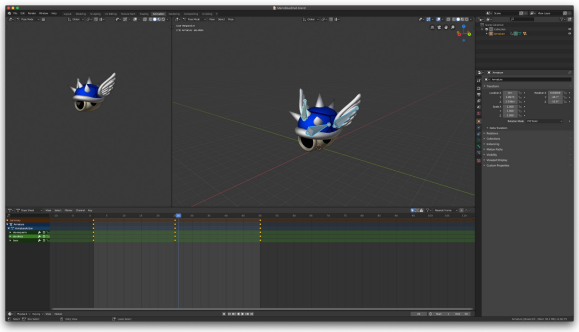
Figure 3.26: Creating the Thwomp in Blender.

jects were too complicated to create from scratch, so we found two convincing and high-quality 3D models on the internet.

However, we did handle the animation. We added a basic bone structure for the wings of both the Koopa and the shell (Figure 3.27) using Blender and animated them so that the entities would make looping jumps and move their wings. With this movement, it appears as if the objects are floating in the air of the level.



(a) Koopa Troopa.



(b) Blue shell.

Figure 3.27: Skeleton of Koopa Troopa and the blue shell.

The Koopa Troopa's functionality is to hover in the air and, when the ball makes a jump, it must avoid colliding with it. If there is a collision, the game ends. The blue shell is simply for decoration. There is one floating near the end of the second level (Figure 3.28).

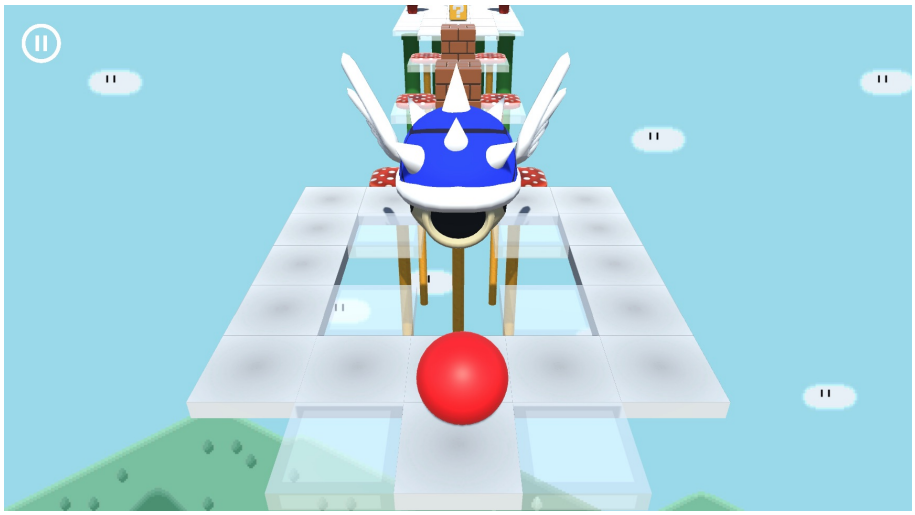
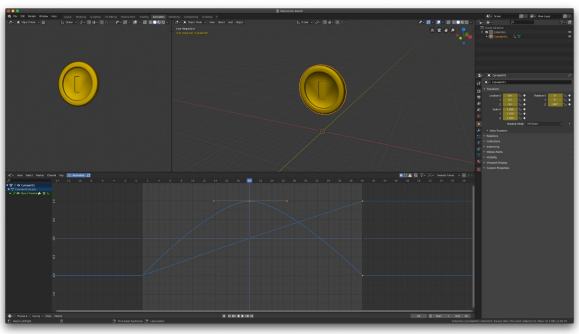


Figure 3.28: Blue shell in-game.

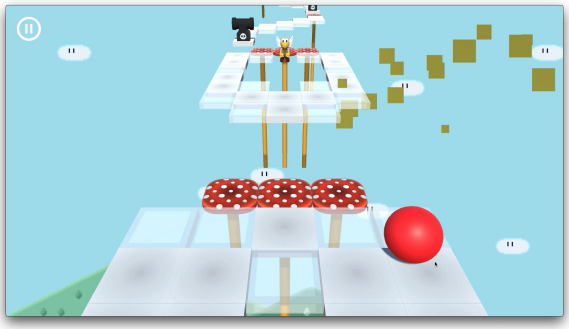
3.5.9 Coin

The coin was one of the simplest objects to collect and animate. We found an acceptable model online and edited its animation using Blender. This animation simply involved a constant rotation around the Y-axis and a vertical movement to simulate a continuous bounce. To ensure that the bouncing appeared convincing and didn't seem like the coin was just moving up and down, we combined Blender's DopeSheet tool (which facilitates frame editing) with the Curves tool, which allows you to modify the motion lines of your object's animation (Figure 3.29a).

This object serves as a scoring method. There are a total of 10 coins spread throughout the level. When the ball collides with a coin, it disappears, and its particle system is activated, which involves dispersing a set of golden particles (Figure 3.29b).



(a) Coin's motion curve.



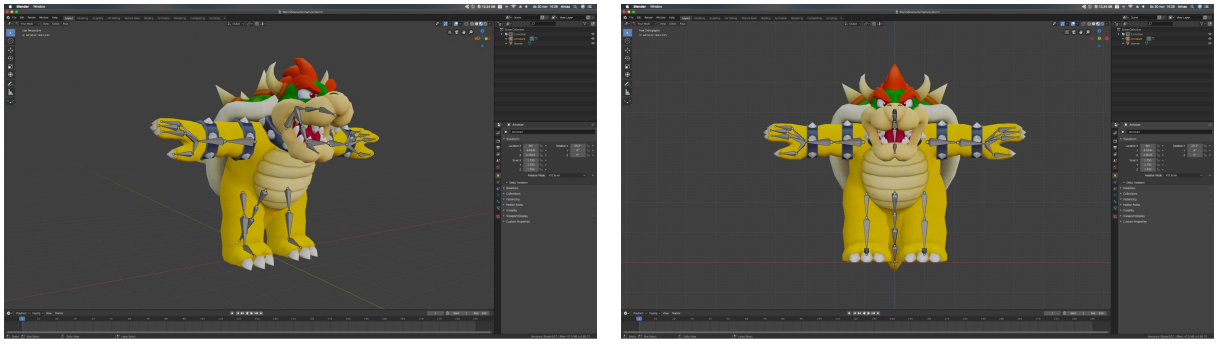
(b) Coin's particle system.

Figure 3.29: Coin's motion and particles.

3.5.10 The Great Challenge: Bowser

This was the pinnacle model of the entire project. Once we gained experience in editing with Blender thanks to the other entities, we decided to base the third level on a boss fight. And who's the baddest of all bad guys in the Mario world? Without a doubt, Bowser!

The first step for animating the model was to break it down into parts. Next, we added a basic bone structure (Figure 3.30) to make initial movements and test its functionality. Once we had the bones attached to the mesh, we had to significantly adjust the weights (Figure 3.31) each bone had associated with different triangles of the mesh. This was necessary because without proper weight adjustments, when we moved, for example, leg bones, it would also affect the movement of some triangles in the belly.



(a) Basic Bowser skeleton.

(b) Front view of the skeleton.

Figure 3.30: Bowser's skeleton.

Subsequently, we added a few more bones which, after attempting some movements, we realized we needed in addition to the ones we already had. Once we regulated their weights (Figure 3.31), we moved on to add different inverse kinematics (IK) to facilitate the animation process. This model had a much higher level of customizable movement compared to the other entities we had used. Therefore, we needed many more IK constraints than in the previous objects. We added an IK constraint for each of the fingers (of both hands) and another to control the movement of the arms (elbow joints). To ensure that the elbows pointed where we wanted, we generated free bones that served as "targets" for the IK (Figure 3.32).

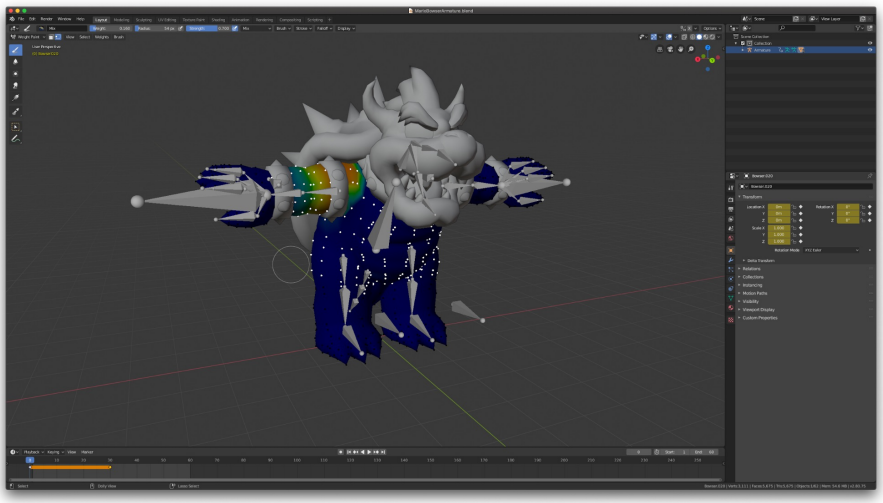


Figure 3.31: Skeleton weight calibration.

After adding the IK constraints for the arms, we also added them for the feet and legs (Figure 3.33). Again, to ensure that the knees always pointed where we wanted, we generated free bones that acted as "targets" for the leg joints.

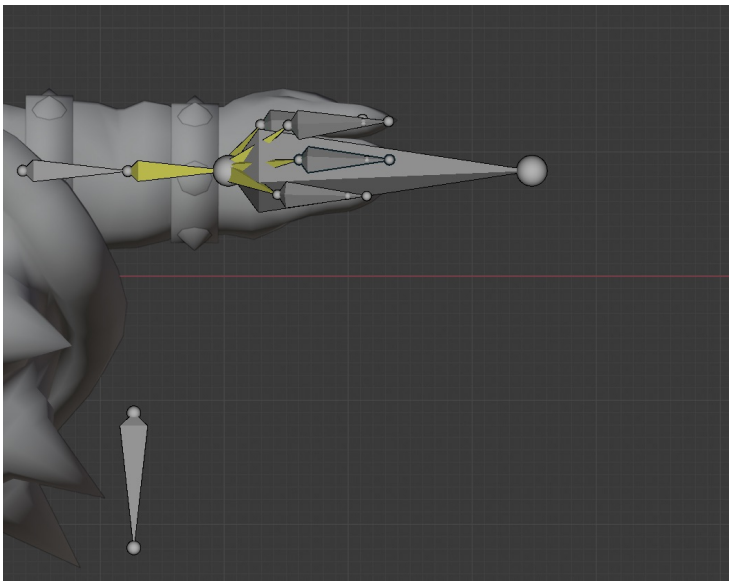


Figure 3.32: Top view of Bowser's arm inverse kinematics.

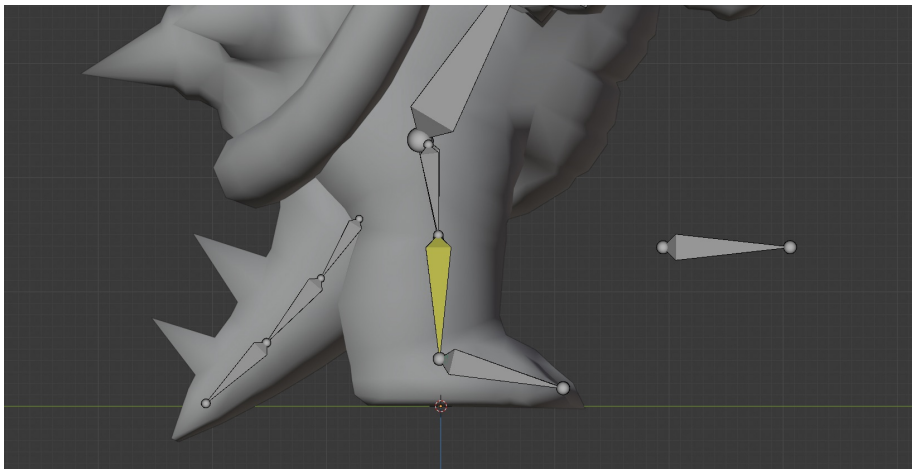


Figure 3.33: Side view of Bowser's leg inverse kinematics.

Once we had the complete armor with appropriate weights and necessary IK constraints for proper movement of all parts, we started with the animation. To prevent the boss figure from being monotonous, we planned out all the movements we wanted it to perform during the level using a flowchart (Figure 3.34) that illustrated Bowser's behavior throughout the level. The diagram is an informal approximation of what the boss had to do during Level 3. The resulting animations were as follows:

- Initial jump: The boss appears by jumping into the scene.
- Initial roar: After landing, Bowser lets out a terrifying roar from left to right (Figure 3.35a).
- Jump onto the cloud: Bowser jumps and lands on a cloud that appears out of nowhere. Throughout the level, he will stay on the cloud and in front of the ball (Figure 3.35b).
- Attack: At certain points in the level, the boss will attack the ball with a fire breath (particle system) (Figure 3.35c). If the ball touches the flame, the game ends.
- Impact: At certain points in the level, the ball can grab the Flower object, which will shoot spiked fragments towards the boss, causing damage (Figure 3.35d).

- Get up: If the boss still has health after an impact, he will get up with a jump.
- Breathing: Whenever Bowser is not performing any of the above animations, he will continue breathing cyclically, giving the impression that Bowser is constantly interacting.

Bowser has a total of 6 health points. Each time he takes a hit, you can see his health decrease in the life bar displayed at the top-right corner throughout the level (Figure 3.35d). If Bowser's health reaches 0, or if the level ends and the ball hasn't impacted any objects, he dies and performs a final impact animation, ending up in its final position.

3.5.11 The Final Object: The Flag

After generating all the objects, we realized that the ending of the levels was too simple. There was no celebratory moment, and it just returned the player to the start. So, we thought, why not create a final object that serves to culminate all the levels? And what could be better than the classic castle flag from Mario games?

We repurposed the bricks that we had used for the brick block, resized them, and arranged them in a circle to form the shape of a castle. Finally, we placed a cylinder in the middle of the structure with a sphere on top and the flag with the Mario "M." The functionality of this entity or object is that when the player completes the level, the ball makes a final jump and starts spinning around the flagpole before coming to a stop on the ground (Figure 3.1).

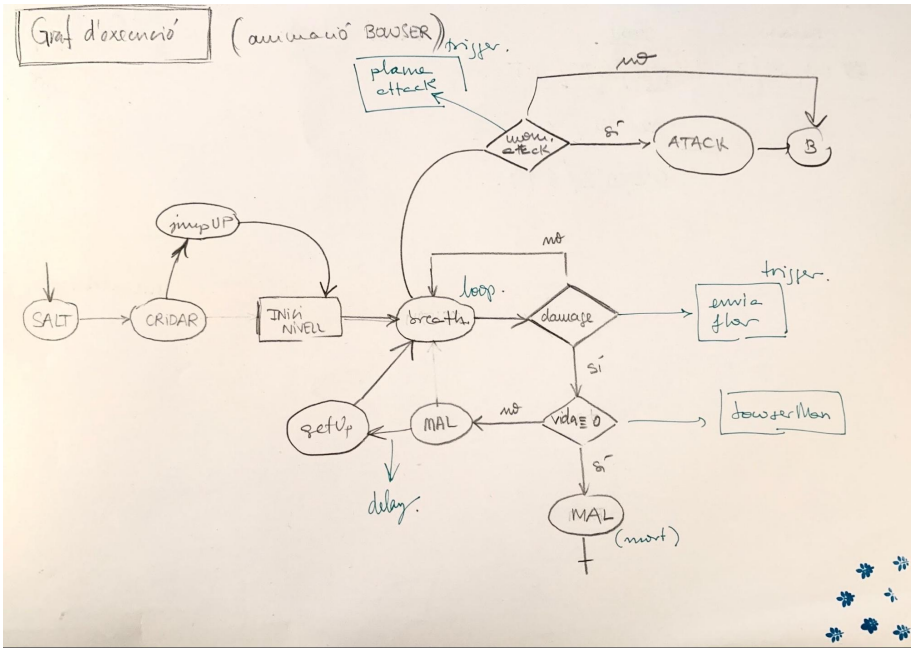


Figure 3.34: Flowchart of boss animation.

(a) Initial roar.

(b) Jump onto the cloud.

(c) Boss attack with flame.

(d) Player's attack impact.

Figure 3.35: Boss (Bowser) animations.

4 Methodology

In this section, we will describe the software used during the project development and the established planning. Additionally, we will outline the communication and version/task management methods and tools utilized.

4.1 Software

4.1.1 Unity

Unity has been the primary application we used throughout the project. Unity is a game engine and development platform geared towards game developers. It offers a wide range of associated features that enable a well-integrated programming phase. For instance, particle systems, UI editors, asset stores, compatibility with various external programs (including Blender), etc.

4.1.2 Blender

Another program extensively used during the game development was Blender version 2.8. Blender is a free, cross-platform 3D editing software. It includes tools for object modeling, animation, rendering, material editing, among others. We used it to create most of the entity models and animations found throughout the levels. Its compatibility with Unity greatly facilitated the process of exporting from Blender to Unity without the need for intermediate formats.

4.1.3 Github, Gitkraken, and Sourcetree

Github is a collaborative development platform used to host projects. Its version control system was invaluable in maintaining an up-to-date version of our game.

The two members of the team used different applications as tools for merging. Both Gitkraken and Sourcetree are tools that proved highly useful, significantly simplifying the day-to-day tasks of programmers, in this case, us.

4.2 Planning

4.2.1 Weekly Meetings

Coincidentally, the two members of the team do not reside in Barcelona, but we live two minutes apart. This made it easy for us to meet at one of our homes every week to discuss the next steps in the game's development and gradually build what has become our final version. In each meeting, at a minimum, the following points were addressed:

- Progress on the design of each level (1, 2, and 3).
- Progress on the design of menus (start, level start, etc.).
- Progress on creating different models using Blender.
- Progress on model animations.

4.2.2 Github

To maintain version control, we used a GitHub repository (Figure 4.1). We used Sourcetree and GitKraken tools (Figure 4.2) to handle it in a simpler and more efficient manner. The repository consists of a master branch where all commits are made. These commits always have a description indicating what change that modification adds.

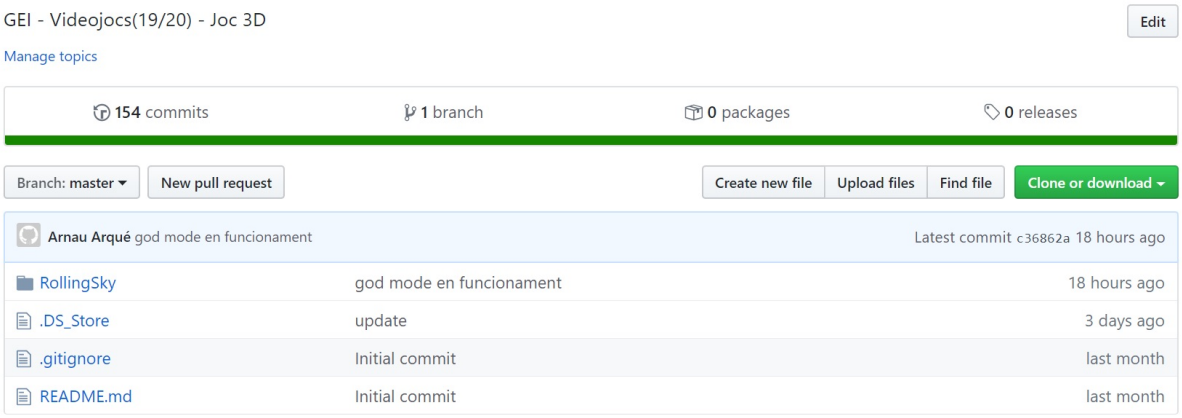


Figure 4.1: Screenshot of the GitHub repository.

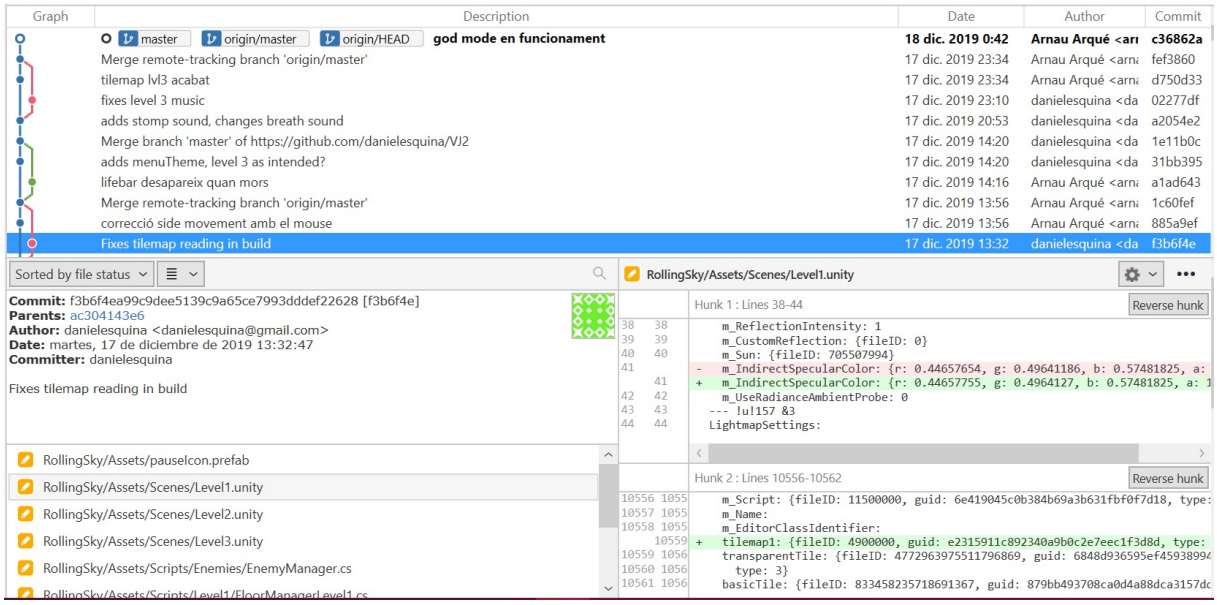


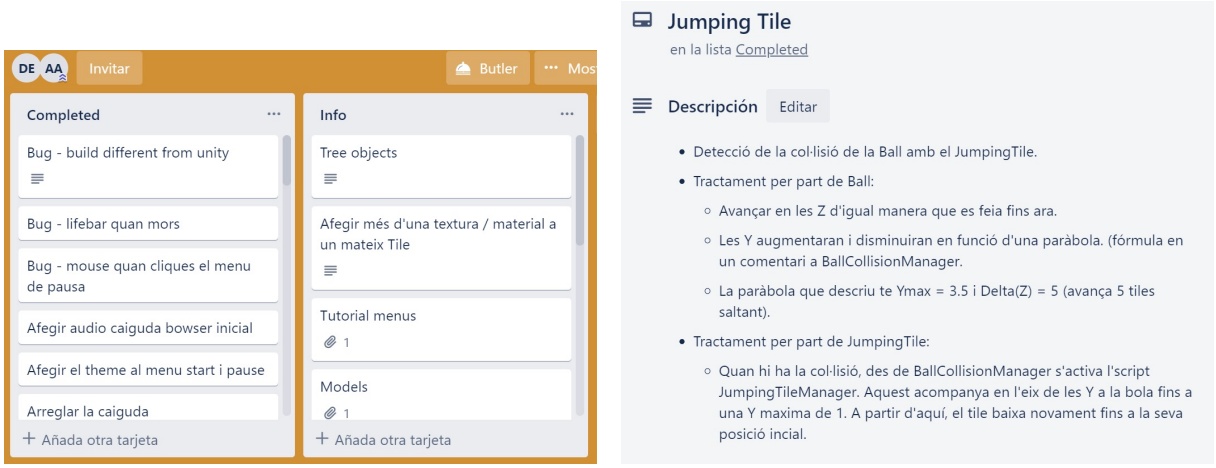
Figure 4.2: Screenshot of Sourcetree.

4.2.3 Trello

To organize the project and keep track of tasks, we used Trello. This tool allowed us to move tasks between different lists, add descriptions (Figure 4.3b), or relevant links. We used 5 lists (Planned, InProgess, Tested, Completed, Info) (Figure 4.3a) to classify tasks based on their status.

4.2.4 Discord

To ensure real-time communication among team members, we used Discord. This program enables participants to make calls, share screens, files, or jot down any details in a chat. This allowed us to engage in pair programming when required.



(a) Screenshot of *completed* and *info* lists.

(b) Screenshot of a task description.

Figure 4.3: Established tasks.

4.2.5 WhatsApp

We used WhatsApp for instant communication. This allowed us to address implementation doubts in many cases and plan the tasks to be performed and their division between team members in a timely manner.

5 Conclusions and Personal Assessment

Upon completing the project and finalizing the rest of the report, it is time to provide a personal assessment and make some comments regarding the work organization.

For us, this has been a completely new experience. Until now, we had never worked with Unity or programmed a game with these characteristics. Moreover, we had not worked with any software similar to Blender either. One of the team members had previously taken the Graphics course at FIB and had some experience with OpenGL and 3D models, but even so, this has been a very different experience from anything we had worked on before.

The idea of programming a 3D game excited us. Furthermore, Rolling Sky has been a game that both of us have played on our phones for many years. Nevertheless, it didn't take long for us to realize that time was running out. We believe that we "lost" too much time learning how to use Unity and its extensions. In a way, as we progressed in the game's programming, we also learned more about using the software (which is not a bad thing). At that point, we noticed mistakes we had made earlier, and due to how we had structured the game, it became challenging for us to make changes. To put it simply, we feel like we learned Unity through trial and error. And that, when you have a project with a limited amount of time, is a problem.

The other point of self-critique is more related to Blender than Unity. Wanting to not only learn how to use Unity but also dive into the world of Blender was a very risky move considering the time we had. Blender, as we mentioned earlier, is a comprehensive and highly customizable free 3D editing software. While it's perfect for generating almost anything you can think of, it also has a downside in that the program is as complex to use, learn, and understand as it is comprehensive.

Regarding the brainstorming process, we can only speak favorably. Initially, we wanted each level to have a different theme (the first one in the style of Mario, another set in a jungle...). Nevertheless, after seeing the potential of the Mario universe, we decided to base the entire game on that theme. It was fun to start thinking about the objects we would use, their functionalities, animations they would perform... It is true that we might have been more ambitious than necessary, but seeing the result, we are very content and satisfied with the final outcome, and how the objects and entities look and interact in each of the levels.

As can be seen throughout the report, we not only focused all our efforts on Unity. A significant portion of our work was also done with Blender, specifically the animation of the final boss, Bowser. It was a challenging task that took us weeks to complete. We started with the armor (bones), then moved on to the process of recalibrating the weights of each bone, and finally generated all the movements and animations. All we can say is that we are very happy with the final result and, although we know it is not perfect, for it being our first animation, it is quite convincing.

Having completed the project and looking at it from this perspective, we believe that regardless of the grade we ultimately receive, we have done good work and have been consistent in our efforts. We met weekly, progressed steadily, and even though these last few weeks have been intense (also due to accumulated work from other courses), we worked continuously on the project until we achieved a final result that we are content and satisfied with.

6 References

Note: The following references follow the following structure:

"Concept", Author (if available). `URL`. Description.

6.1 Websites

"Trello". `www.trello.com`. A project development board website.

"Google Drive". `https://drive.google.com/`. The main page of Google's storage system.

"Opacity of an object". `https://bit.ly/36K02mv`. A Blender Stack Exchange page explaining how to modify the opacity of an object.

"Bullet Bill", by Anthony Yanez. `https://bit.ly/2S6TaLM`. A Sketchfab page containing an example of a Bullet Bill model from Mario.

"Mario Hammer", by Antony Yanez. `https://bit.ly/2EvTGLo`. A Sketchfab page containing a Hammer model from Mario.

"Mario coin", by Antony Yanez. `https://bit.ly/2Qb8rZF`. A Sketchfab page containing a Mario coin model.

"Preserving aspect ratio when unwrapping". `https://bit.ly/2YZuSEW`. A Blender forum page explaining how to unwrap a model while preserving its aspect ratio.

"Graphing calculator". `https://www.desmos.com/calculator`. An online calculator for functions and graphs.

"NLA editor and actions". `https://bit.ly/2tqVdQt`. An RGB-LABS page explaining how to use Blender 2.8's NLA editor and actions. The page is in German but very useful.

6.2 Videos

"Best places to find 3D models", by Max Novak. `https://youtu.be/XGjBOLY8utg`. A YouTube video listing the best websites to find 3D models.

"Basic smoothing", by SadowickDevelopment. `https://youtu.be/HfDBVjLyQtc`. A YouTube video explaining the basics of smooth shading in Blender 2.8.

"Modeling Super Mario Bomber", by Redninja Multimedia Productions. `https://youtu.be/55wDgt2kuDk`. A YouTube video demonstrating basic modeling of a Super Mario Bomber.

"Alpha transparent materials", by KatsBits. https://youtu.be/z2cFVVotcs8. A video demonstrating how to create translucent materials with Blender 2.8.

"How to animate in Blender 2.8", by Thilakanathan Studios. https://youtu.be/Huuo7RKO1RE. A video explaining the basics of animation using Blender 2.8.

"Volumetric clouds", by PuckLovesGames. https://youtu.be/LLUUIAKFgWg. A video showing how to generate volumetric clouds in Unity.

"How to create fog", by Jimmy Vegas. https://youtu.be/UgJE3TgT3o8. A video explaining how to generate fog in Unity.

"Tips using bevel", by TopChannel1on1. https://youtu.be/2jTLbjbN1DY. A video demonstrating how to use the bevel tool in Blender 2.8.

"Armature tutorial", by Steven Scott. https://youtu.be/l3IEwxHRs3g. A video demonstrating how to create a basic armature using Blender 2.8.

"How to create and bake normal maps", by Darkfall. https://youtu.be/Hpdwo6F_WYQ. A video showing how to generate a normal map from a texture using Blender 2.8.

"How to bake normal maps", by Bestdani. https://youtu.be/h24akA8K-40. A concise video on generating a normal map with Blender 2.8.

"Procedural lava", by Ducky 3D. https://youtu.be/HOYYB50qbjk. A video explaining how to generate procedural lava using Blender 2.8.

"How to unwrap and bake textures", by One Wheel Studio. https://youtu.be/c2ut0Trcdi0. A video explaining how to unwrap and bake textures in Blender 2.8.

"How to animate scroll textures", by Jimmy Vegas. https://youtu.be/auVq3TSz20o. A video demonstrating how to generate scrolling textures.

"Bones and armature", by Grant Abbitt. https://youtu.be/IAiTYaiZmY0. A highly useful video on creating armatures, binding them to the mesh, and animating them using Blender 2.8.

"Basic IK and foot roll", by Grant Abbitt. https://youtu.be/gH5uATTTYB4. Another very useful video explaining the use of Blender 2.8's inverse kinematics tool.

"Rolling sky all levels", by Tugan Nguyen. https://youtu.be/w51dzhrQPXc. A video showcasing all the current levels of the original Rolling Sky game.