UNIVERSITAT JAUME•I

Videogames Design and Development Degree

VJ1241: Technical Report of the Final Degree Project

Creation of a complete horror level
with its sound environment in Unreal Engine 4.

Author: Jaime Lara Mora.

Tutor: Miguel Chover Selles.

Summary

A key point to the success of a game is the immersion that creates on the player through its environment, this refers to the playable levels and the soundtrack of the game [1]. Focusing on this matter is something very valuable for game makers. This Final Degree Project is about creating such engaging and explorable horror environment, as a demonstration of what can be done in a game without actually creating the whole game, but a demo. On this document, the demo will be referred to as a game.

The main objective of this project is the creation of a complete and explorable 3D environment that creates tension and fear in whoever plays it. Specifically, it consists on a mountain which you are be able to explore and hear. It is set on night time, thus having the corresponding sounds of nocturnal animals and some added extras, that creates a tense atmosphere. This is entirely done on Unreal Engine 4

Key Words

Horror environment, Game engine, 3D modelling, Sound Design, Events programming.

Index

Figures Index

Tables Index

1. Introduction

1.1 Objectives

We can summarize the main purpose of this project in one sentence: Creating a complete and explorable 3D horror environment, including its soundtrack implemented in a game engine. In order to do this project, it is required to break down this idea into smaller objectives:

1. Creating a complete environment that generates fear and tension in the player through 3D modeling and texturing.
2. Record, edit and generate sounds for this project that will create the feelings mentioned above in the player.
3. Being able to implement everything in a game engine.
4. Professionally documenting all the work done, including a time schedule.
5. Put into practice all the competences acquired in the degree.

1.2. Visual References

The level will be inspired in a forest named "Os Grobos" located in Galicia. The idea is to create a forest that has many karst formations which will make the player feel secure and insecure at the same time. In these formations you can hide but there's also the possibility that something has been hiding in there before you. You will start at the top of the mountain and will have to go all the way down to the end, experiencing a spooky night time ambient and some events, like rocks moving, falling, and even a strange presence. Here's a picture of the mentioned forest:

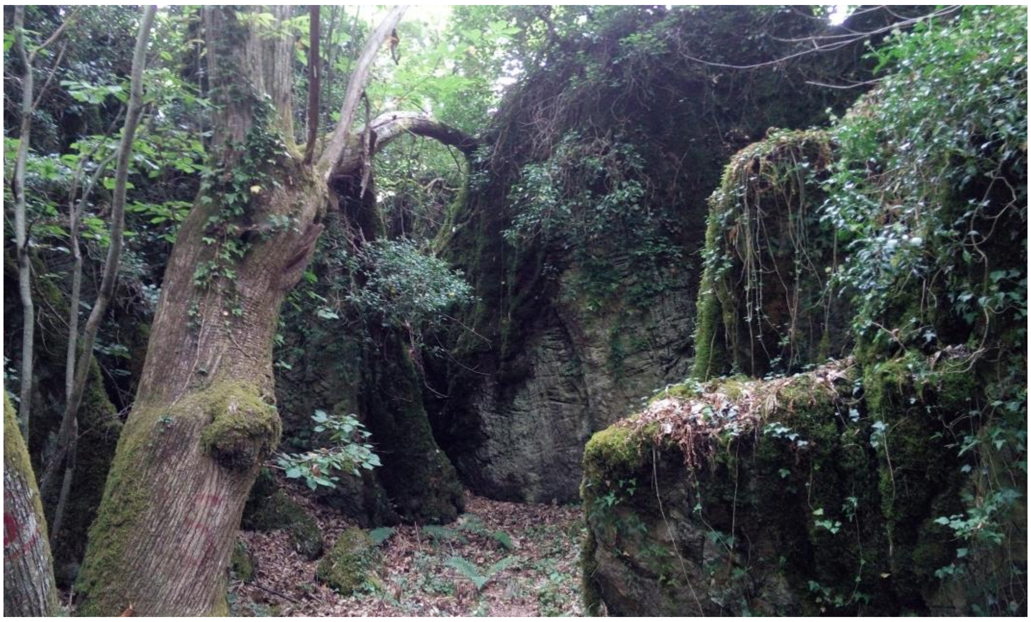


Figure 1. Photo of the "Os Grobos" forest.

1.3. Resources

For this project, a wide variety of software was used. Here they will be mentioned and briefly explained.

Blender, 3D Max and zBrush

These softwares were used for every 3D asset that needed to be created externally, including the terrain. These are currently widely used in the videogame industry.

Ableton Live Lite, Kontakt and Adobe Audition

For the sound aspect of this project Ableton Live Lite was used. It is a good musical editing software that suits the needs of this project. Adobe Audition was also used to edit some of the recordings that were made, and even some of the external sounds used were edited with it. Kontakt is a software to load virtual instruments and use them.

Unreal Engine 4

Unreal Engine 4 was used to merge everything and create the finished project. This engine was chosen because it’s a big reference inside the game industry that is worth learning. Also, it may suit better the needs of the project because of its lighting engine.

Google Apps

Nowadays, Google is present in almost everything we do. It will be used for every research needed. Moreover, its other applications are valuable tools to work with, like Google Docs for example. The decision of using Google apps instead of other software like Microsoft Word, is mainly because Google apps are free and constantly synced with every device one may own.

2. Planning

This whole section is taken from the Technical Proposal. Later on this document, the deviations of the original planning will be discussed. In this section, a list the tasks will be shown along with a table with the relations between them and the objectives.

2.1 Tasks

The Final Degree Project can be divided in the following tasks to have a better understanding of what is going to be done and how:

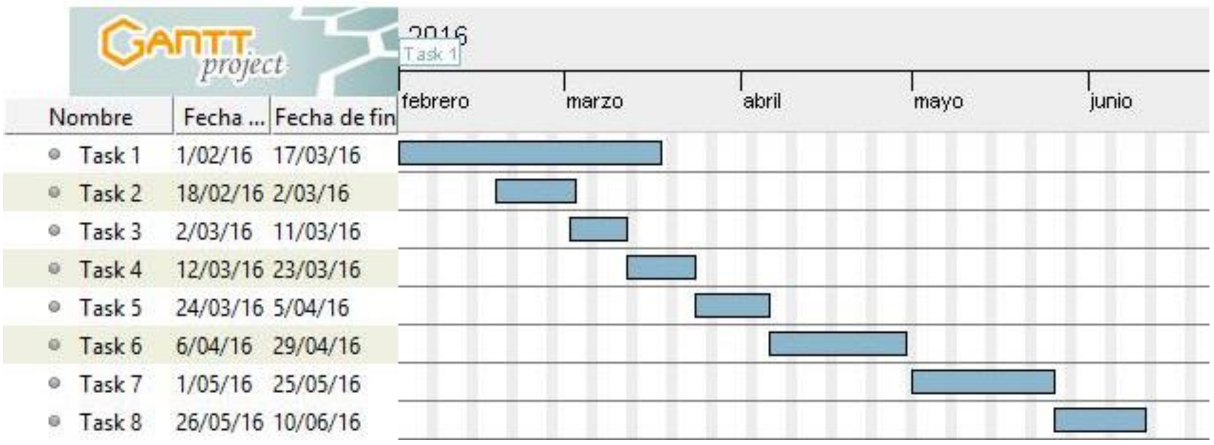
1. Doing a professional technical proposal where the project is explained.
2. Investigating horror games, their environments, music and sound effects.
3. Modelling the level.
4. Texturing the level.
5. Modelling and texturing the asset.
   5.1. Implementing them on the game engine.
6. Doing the soundtrack and designing the sound effects.
7. Programming all the interaction in the game engine along with the sounds.
8. Documenting the final results of the project and preparing a presentation for it.

Considering this tasks, the Table 1 below will show the relations between them, the time and the objectives of this project:

Tasks	Time	Objectives
1. Doing a professional technical proposal where the project is explained.	15h	4 and 5
2. Investigating horror games, their environments, music and sound effects.	15h	5
3. Modelling the level.	40h	1 and 5
4. Texturing the level.	30h	1 and 5
5. Modelling and texturing the asset and having them implemented on the game engine.	50h	1, 3 and 5
6. Doing the soundtrack and designing the sound effects.	40h	2 and 5
7. Programming all the interaction in the game engine along with the sounds.	40h	3 and 5
8. Documenting the final results of the project and preparing a presentation for it.	50h	4 and 5

Table 1. Relation between tasks, objectives and deadlines.

It is a total of 300 hours that will be used to complete this game. Below this you will see the dates that were on the original planning on Figure 2:



Nombre	Fecha ...	Fecha de fin
Task 1	1/02/16	17/03/16
Task 2	18/02/16	2/03/16
Task 3	2/03/16	11/03/16
Task 4	12/03/16	23/03/16
Task 5	24/03/16	5/04/16
Task 6	6/04/16	29/04/16
Task 7	1/05/16	25/05/16
Task 8	26/05/16	10/06/16

Figure 2. Gantt chart for the tasks.

2.2. Risks and Contingency Plans

This project, like almost every one, has its risks. So, in order to succeed, it's a good practice to sit down to think about about them and how to avoid them. Here's the result of that thinking process:

Risks	**Contingency Plans**
Unreal does not suit the needs of the project.	The project will be moved to Unity.
There's not enough time to model the assets.	There will be less assets with less detail.
Not enough time to finish the whole level.	A smaller portion of the level will be then presented.

Table 3. Risks and Contingency Plans

3. Art

This chapter will show the art of the project, everything that has been done and used for it, including photos. The work done will be separated in 2 categories: Assets and Ambience. For showing purposes, in some images the lighting of the game will be raised to a point where everything can be seen. On the original game, the light is at a minimum value to give the game a dark feeling.

The assets that were modeled on high poly had first to be processed to a lower poly version to optimize the performance of the game. Implementing some assets in the engine was a bidirectional task. One has to keep in mind how they will look in the engine while texturing them, so sometimes the model had to go back and forth from the 3D software to Unreal to make it look right.

Materials in Unreal are created by a series of connected nodes. These nodes are a form of visual scripting, as they contain HLSL code (High-Level Shader Language), so they can be as complex as the user wants. But to make it a little bit more difficult, complexity obviously affects performance as well.

3.1. Terrain

This was the first asset modeled. It was done using Blender and its sculpt mode. Although the texturing was done entirely on Unreal, this asset needed to go back and forth from Blender to Unreal to be able to extract the right normal maps in order to get a decent result on the engine. Also, the terrain on Unreal needed a bit of tweaking on its resolution and dimensions.

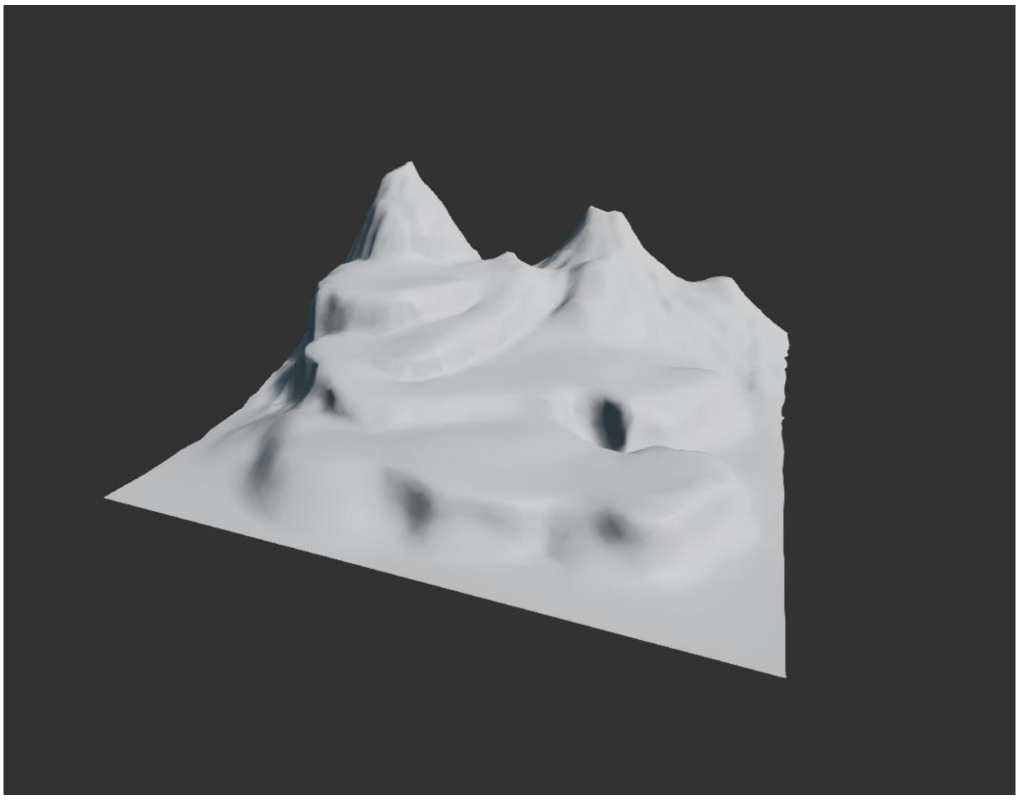


Figure 3. The model of the terrain inside the engine.

Texturing the level was a bit of a challenge. First finding a texture that suited the atmosphere of the project. Then making it work on Unreal Engine. The terrain had to have different textures, but Unreal only allows one material for it, so it's necessary to organize the material in layers. The terrain is composed by four layers, and every layer has its texture, some nodes to resize them, and other nodes to make variations on the surface, such as randomly rotating the texture along with lerp nodes to make it look good. To try what you do, you have to compile the material and wait for it. This takes a little bit of time, so it took longer than expected to make it right. This is what the material blueprint looks like:

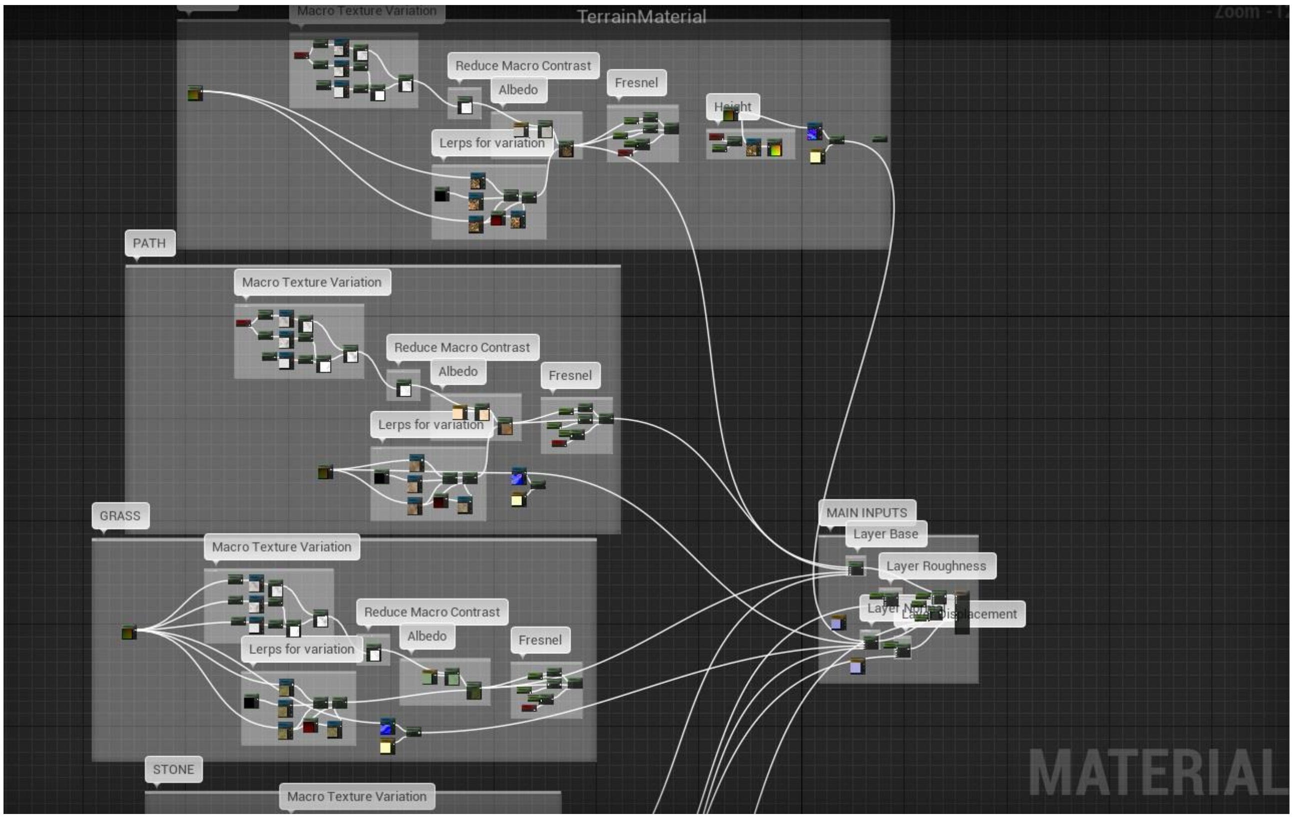


Figure 4. Material of the terrain.

This image barely shows the four layers used on this material. The first is “Leaves”, but the title is not visible, the second “Path”, the third “Grass” and the fourth “Stone”. Let's take a closer look at one of the Layers of the material to understand what's going on, we’ll focus on the “Path” layer:

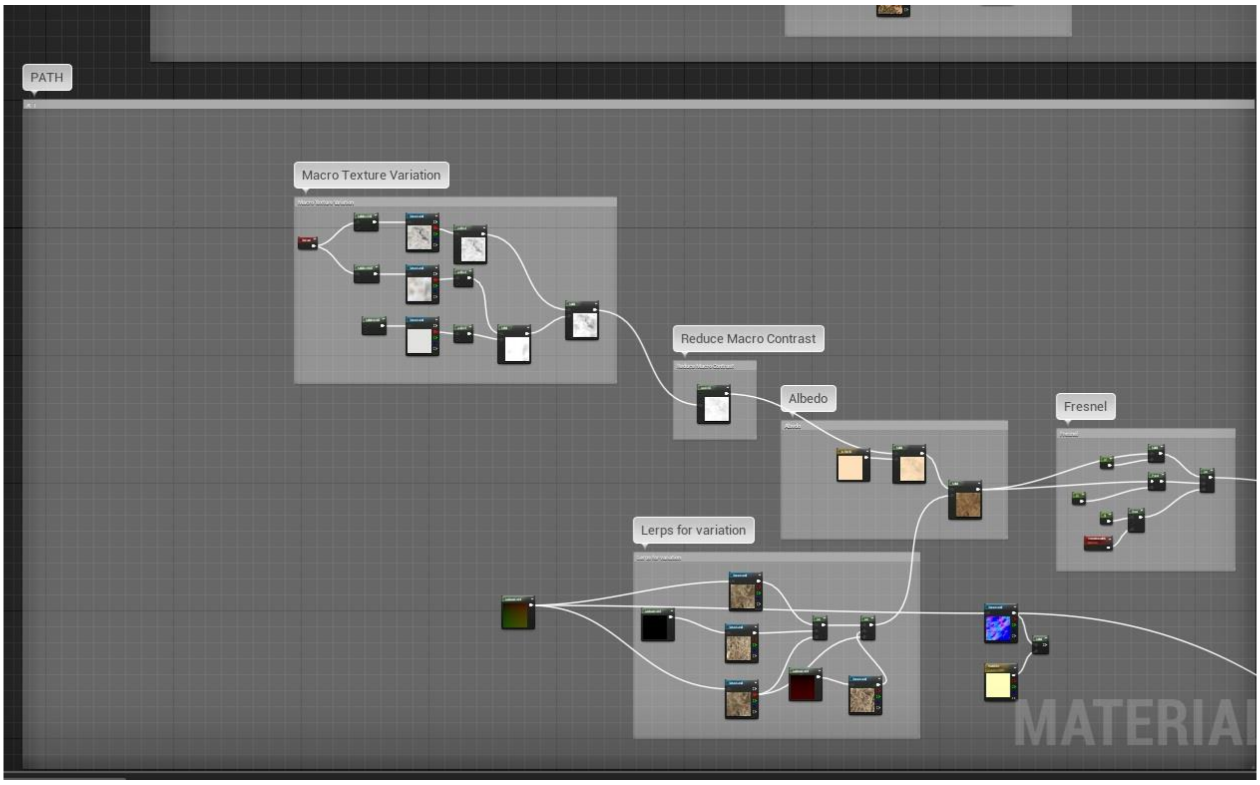


Figure 5. Group inside the material of the terrain.

As you can see, every part is grouped in boxes to make it easier to understand. From left to right, the first group is a variation applied to the texture. It consists in a "variation map" with its own scale, that when applied to the texture through a combination of add and multiply nodes, it slightly varies the color of the texture following a random pattern based on the map. It is then reduced multiplied by the color variation and applied to the main texture on the next two groups. On the bottom group, the main texture is rotated a few times, joined together by lerps and then applied to the main texture. This is a really important part of the material. Without this (Figure 6), the texture would look almost like tiles. Here's an external example of what is being mentioned:

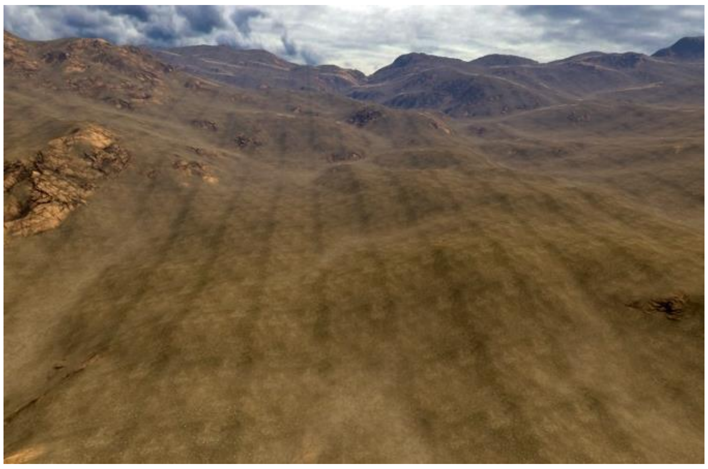


Figure 6. Example terrain without rotations.

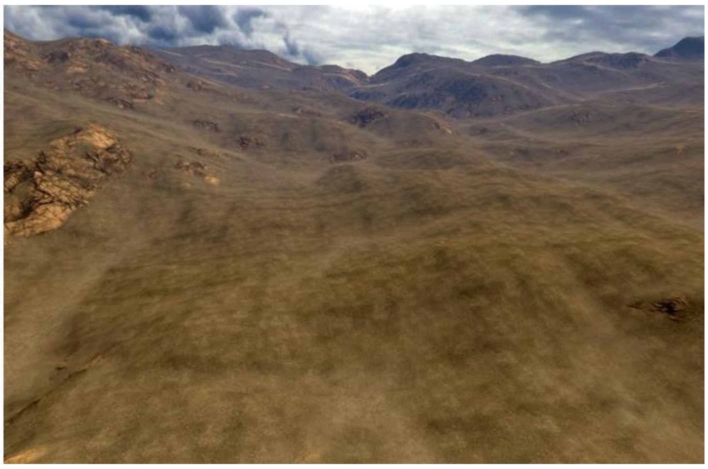


Figure 7. Same terrain with rotations

The last group on the right creates a Fresnel effect on the layer. This effect makes the material look better and a bit more realistic in the distance, is a good effect to apply to landscape materials. All this is joined by a Layer Blend node that joins the four layers to the material. So every layer gets its own texture, normal map, roughness, specular value, etc. This way you can manually paint on the terrain with any layer you want. Here you will be able to see an example, this is how the path texture and the leaves texture blend together. This was done using Unreal's paint and smooth brushes:

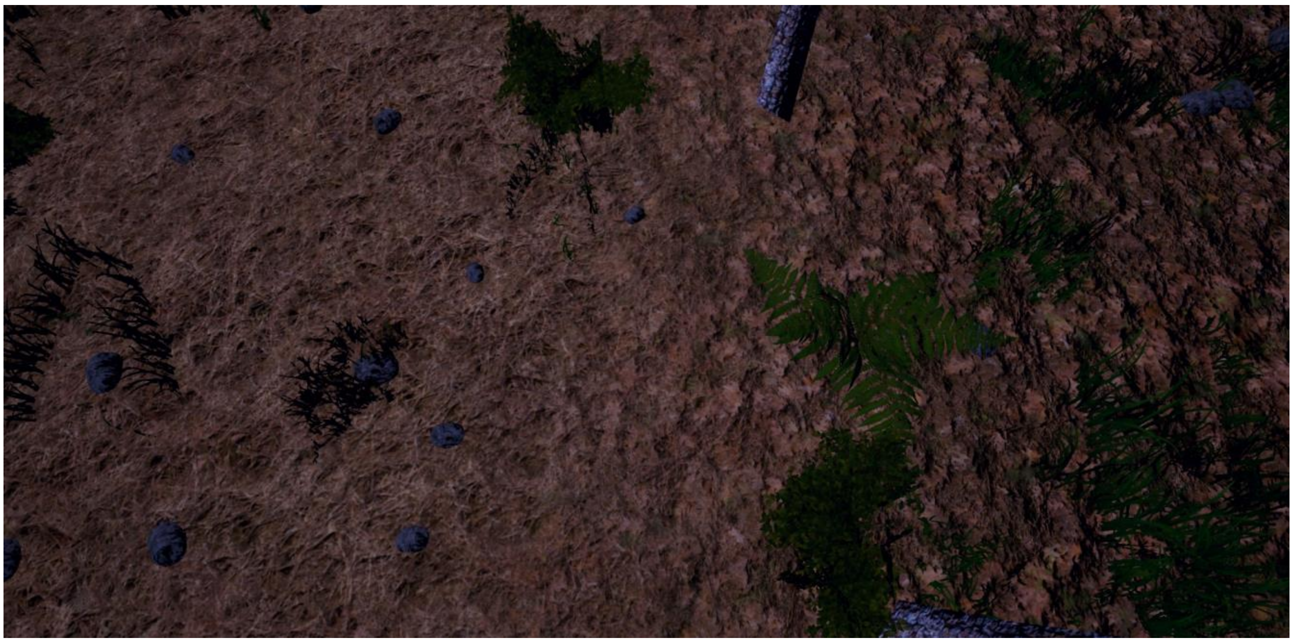


Figure 8. How Path texture (left) and Leaves texture (right) blend together on the terrain

3.2. Moon

The moon consists in two objects: A semi sphere and a plane. The semi sphere is textured with an actual image of the moon multiplied by a similar color, which is what gives it brightness. The halo works in a similar way. It's a 2D plane with a texture of a black and white radial gradient, and also is multiplied by a white color to add brightness. The only difference is that the plane is a masked material, this means that depending on its texture, some parts of the plane can be translucent, so the plane is seen as a circular halo and not as a plane. With the brightness multipliers we get the sensation that the moon is actually lighting the scene, but the real actor that lights the scene is invisible. Here's how the materials look:

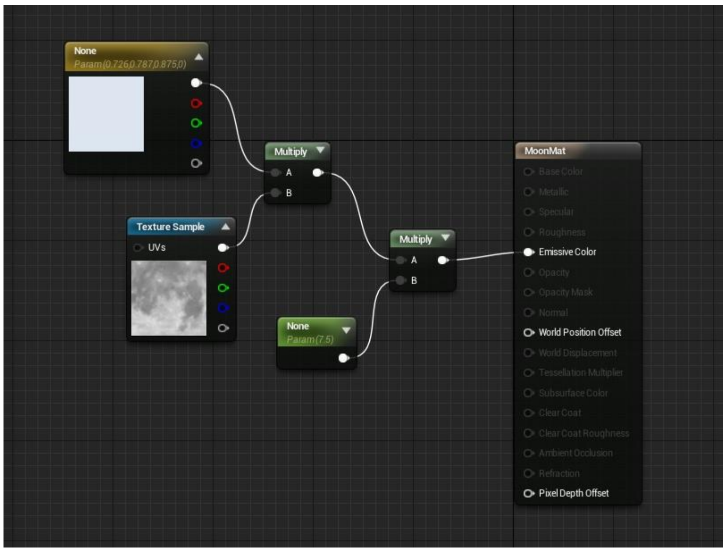


Figure 9. Moon material.

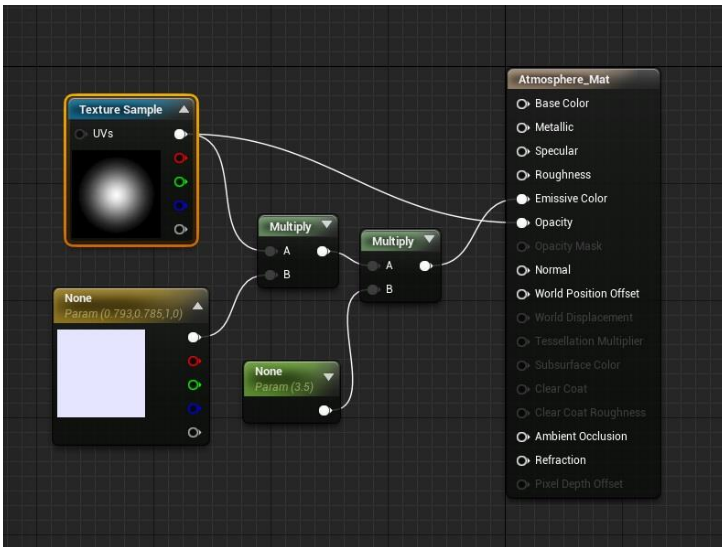


Figure 10. Halo material.

In order to have a glowing material, you have to use the emissive color connection instead of the base color. The base color is affected by the external lights, but the emissive color is not and has its own light. So when multiplied, it glows. To better understand the difference between emissive color and base color only, let's see two examples of the same moon with and without emissive color.

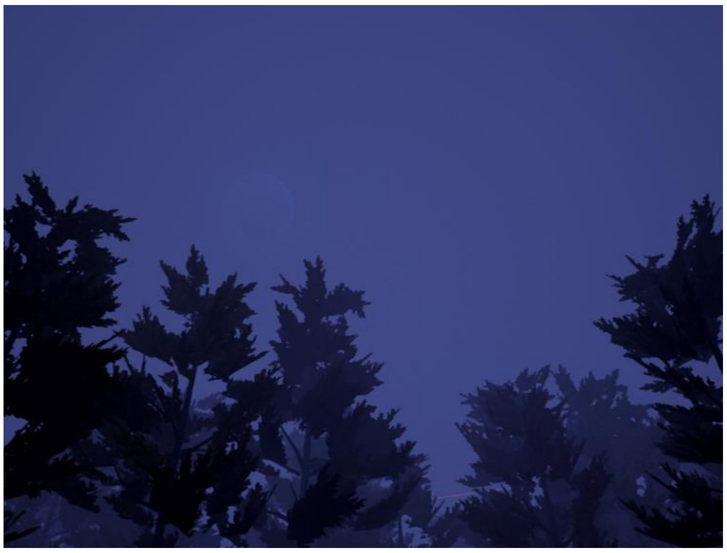


Figure 11. Moon with base color.

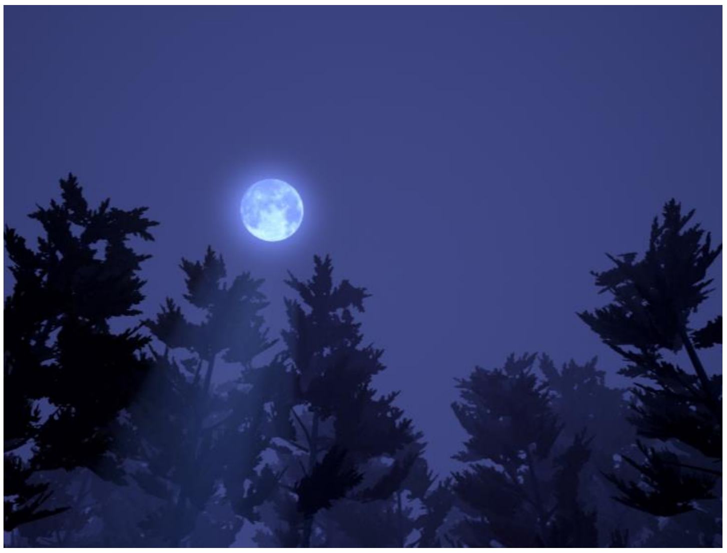


Figure 12. Moon with emissive color.

3.3. Owl

The project needed to have some decoration, to make it have a sort of “soul” or personality. The idea was to have some animal with glowing eyes opening and closing and owls were perfect for this matter. Owls are often surrounded by beliefs and superstitions and associated with Occult knowledge, shamanism, demons and other spiritual matters. Their sound is often used to create mystery at night on films and games, plus they're often portrayed with glowing eyes on some tales. This asset was inspired on an external design of a cartoon owl. It was then modeled on zBrush. The body was sculpted from a sphere using the grab brush and smooth. After that, using the mask brush, its body parts were painted on it, then extracted and modified also with the grab and smooth brushes. Every part was done using the same system. Then the feet and some details were added to it using the sculpt brush. In the next figure you will be able to see its making process:

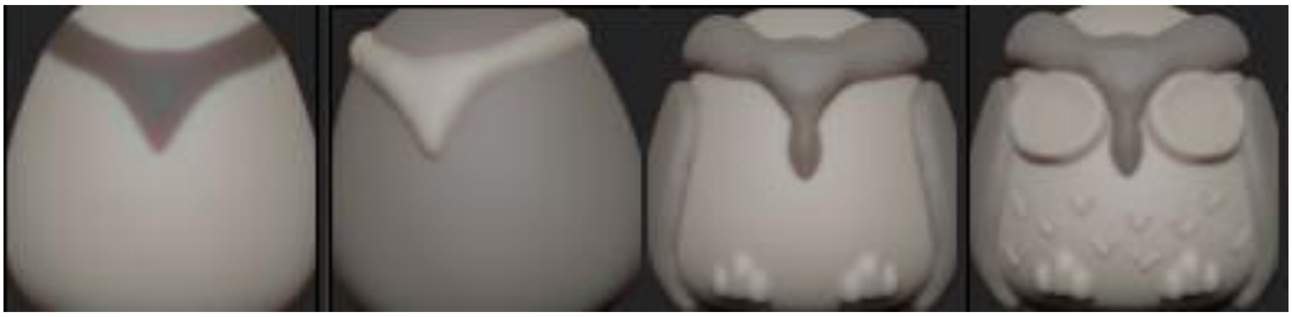


Figure 13. Making process of the owl.

It wasn't texturized because its purpose is to be totally black, so you can only see its silhouette and glowing eyes. Its eyes will be explained further more in the fourth section. Here you can see how his low poly version looks rendered on the engine:

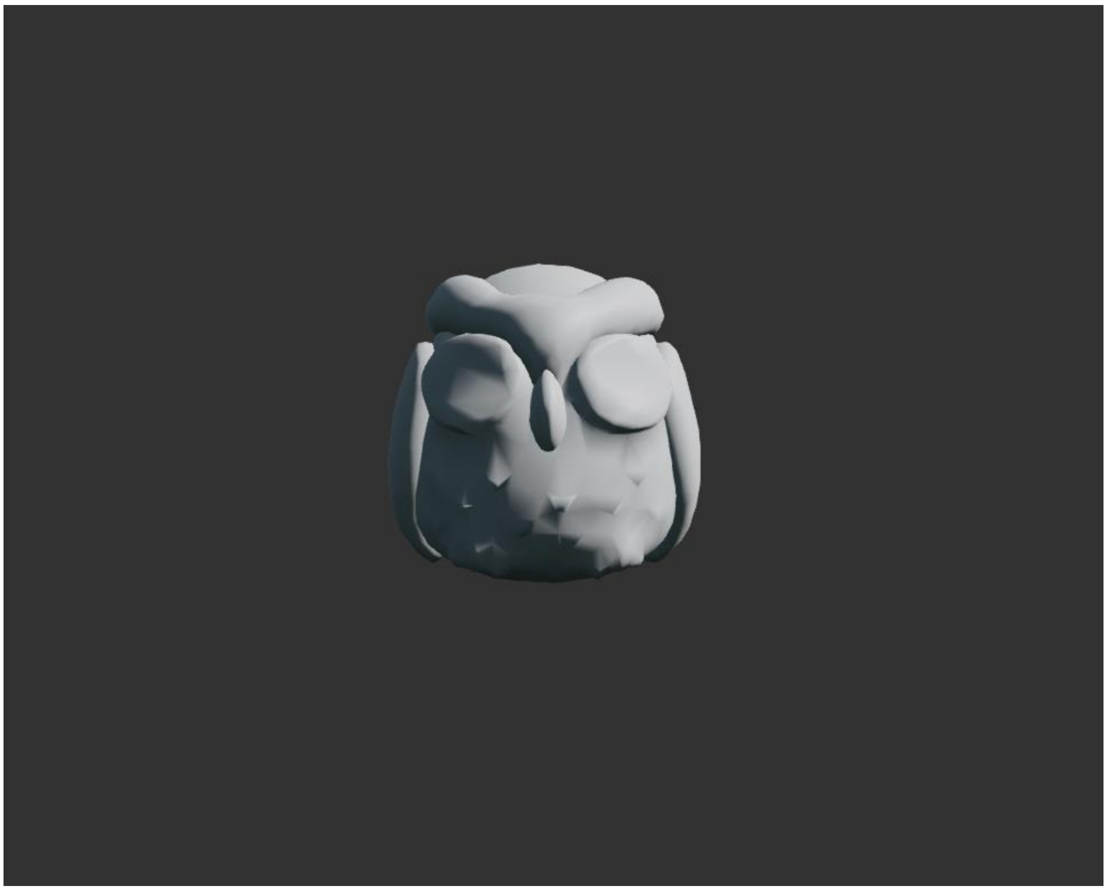


Figure 14. The owl on the engine.

3.4. Rocks

As the initial idea was to make a forest with karst formations, some rocks were modeled for that purpose. Three rocks were modeled to have some variations. They all started as simple cubes, got resized and subdivided enough to be able to model them on high poly, even though only their low poly version were going to be used on the engine. This way the resulting low poly object will have more details even without normal maps. A total of three rocks were made for this project. This is an image of the high poly version of one of the rocks:

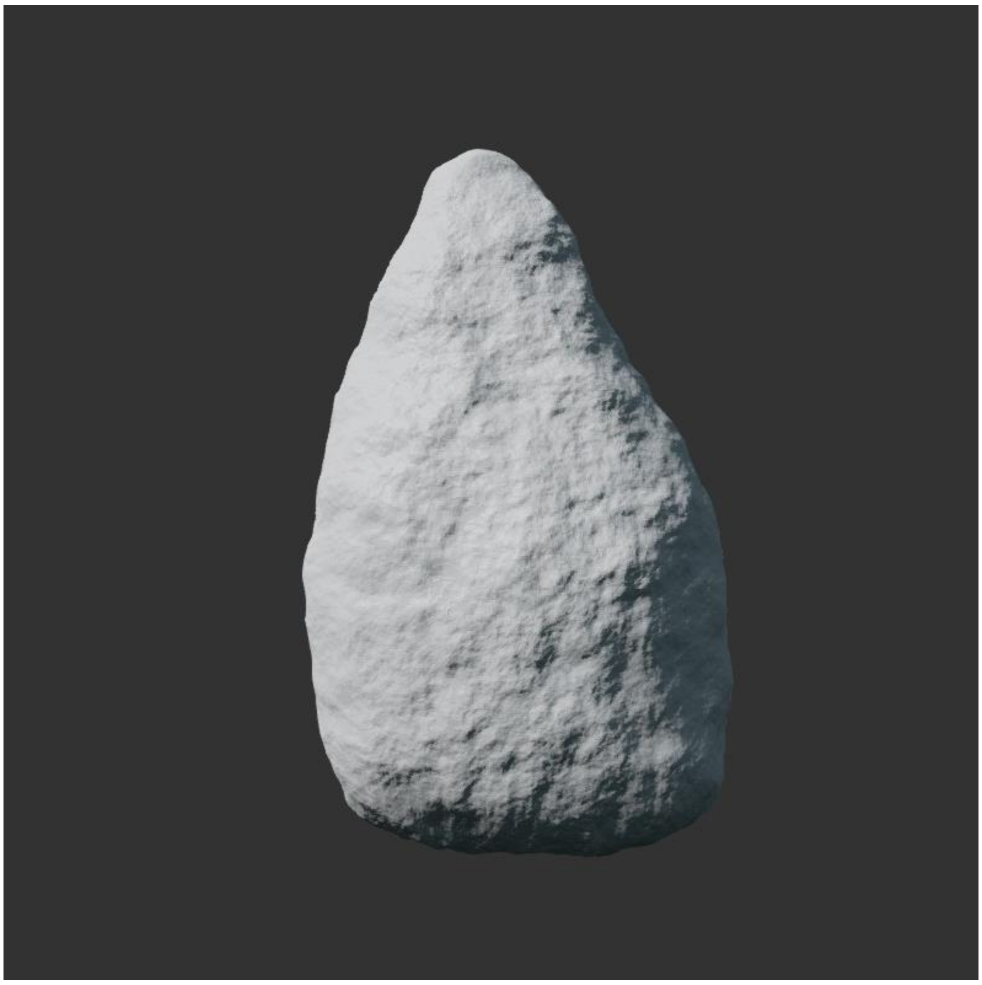


Figure 15. A high poly version of one of the rocks.

There were two options to texture it: painting it on blender or doing it entirely on the engine. The result of the first option wasn't exactly what the project needed. Because of that, the texturing was done inside Unreal. First, the model has to have a defined UV Map. To create a UV Map, you "unfold" the model into a 2D image, kind of like origami. For a better understanding of what this is, here's an example on the next figure:

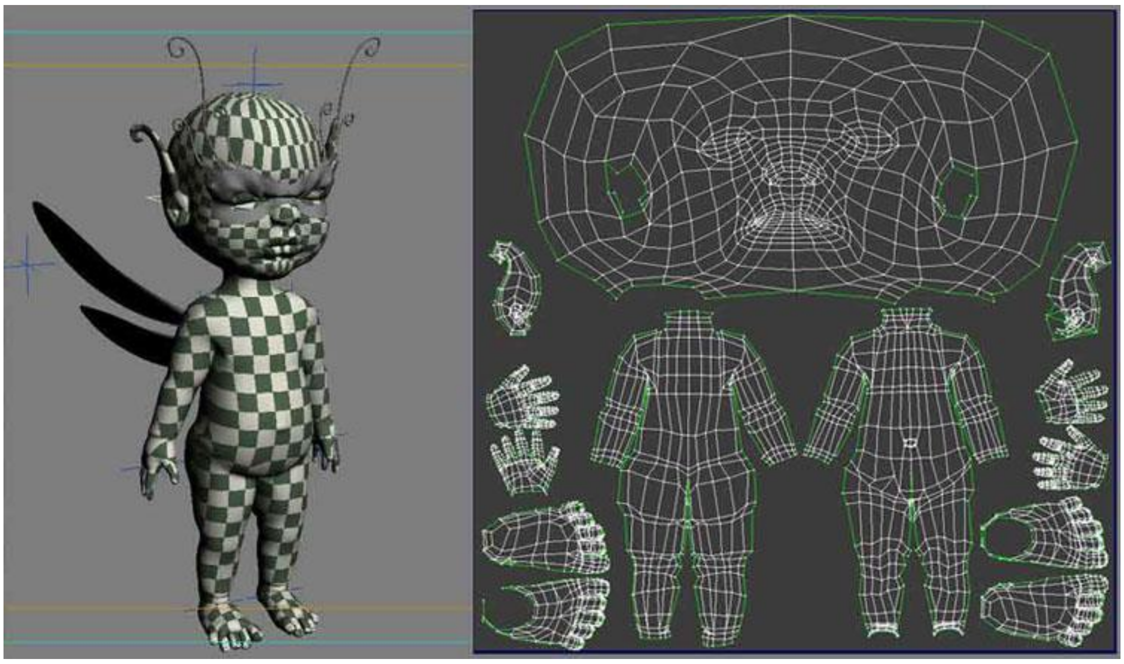


Figure 16. UV Maps of an example model.

Various techniques were used to get these maps, but the UV seams keep appearing on the model. The UV seams are the joints that join together every piece of the map on the model. It's really difficult to avoid having those seams on your model. After some research and tries, the best option was to do the unwrap with 3D Max, tweaking some parameters. This way, the rock could finally be moved inside the Engine. Once in Unreal, the project already had in use a good stone texture for the terrain, so the same was used for the rock, just needed a "Texture Coordinate" node (red node on the left of Figure X), which resizes the texture to the size you want. After that, a normal map is needed. Normal maps can easily be baked from the high poly model to the low poly one, but this didn't suit well the texture. The best option was to use the texture's normal map with the same Texture Coordinate node. Also, the normal map is multiplied by a specific blue color to soften the overall effect of the normal map. This is because if the normal map is too abrupt, it will create really dark shadows, so it needs to be softened this way to make it look good. Lastly, some parameters needed a bit of tweaking. Unreal uses PBR Materials, this materials approximate what light actually does instead of approximating what we think it should do. This results in more natural looking materials. Its properties are values that can be measured from real world Materials. These are: base color, roughness, metallic and specular [2].

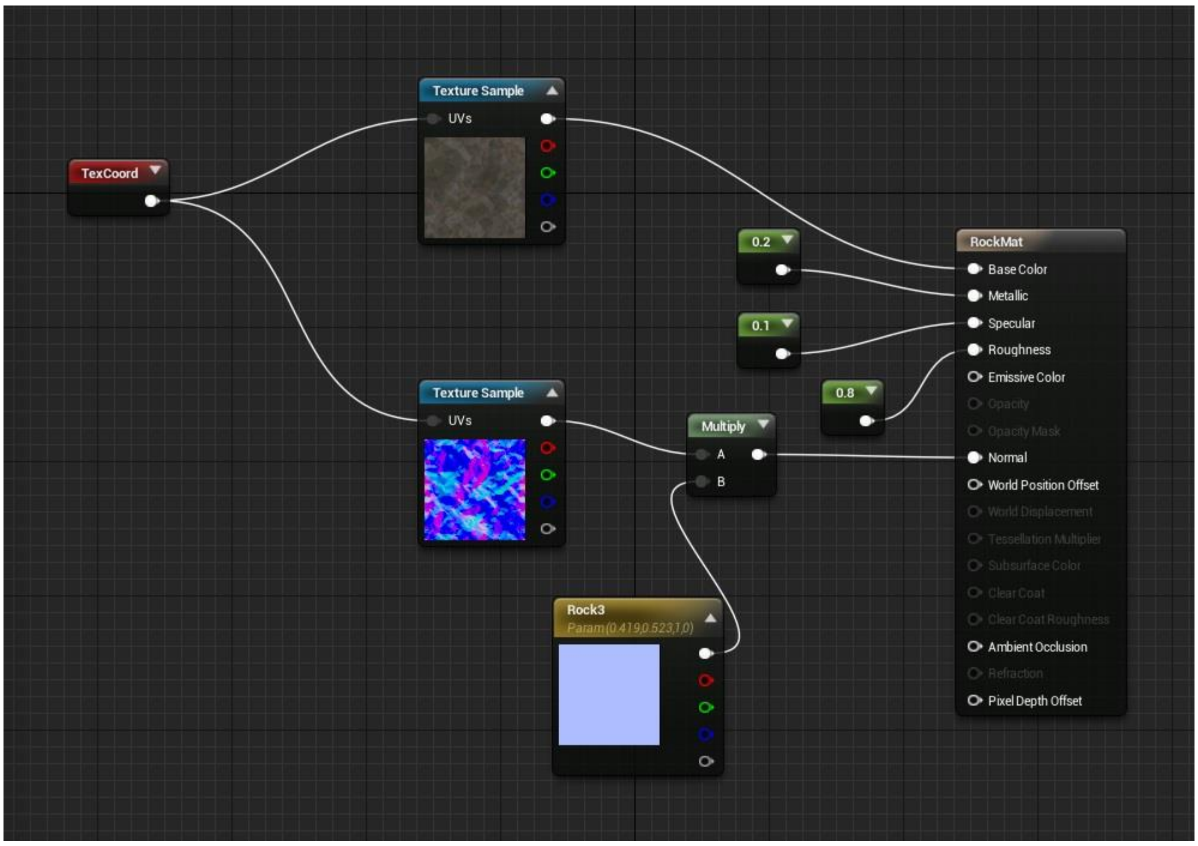


Figure 17. Rock material blueprint.

After some tweaking, the rock finally looks like this on the engine:

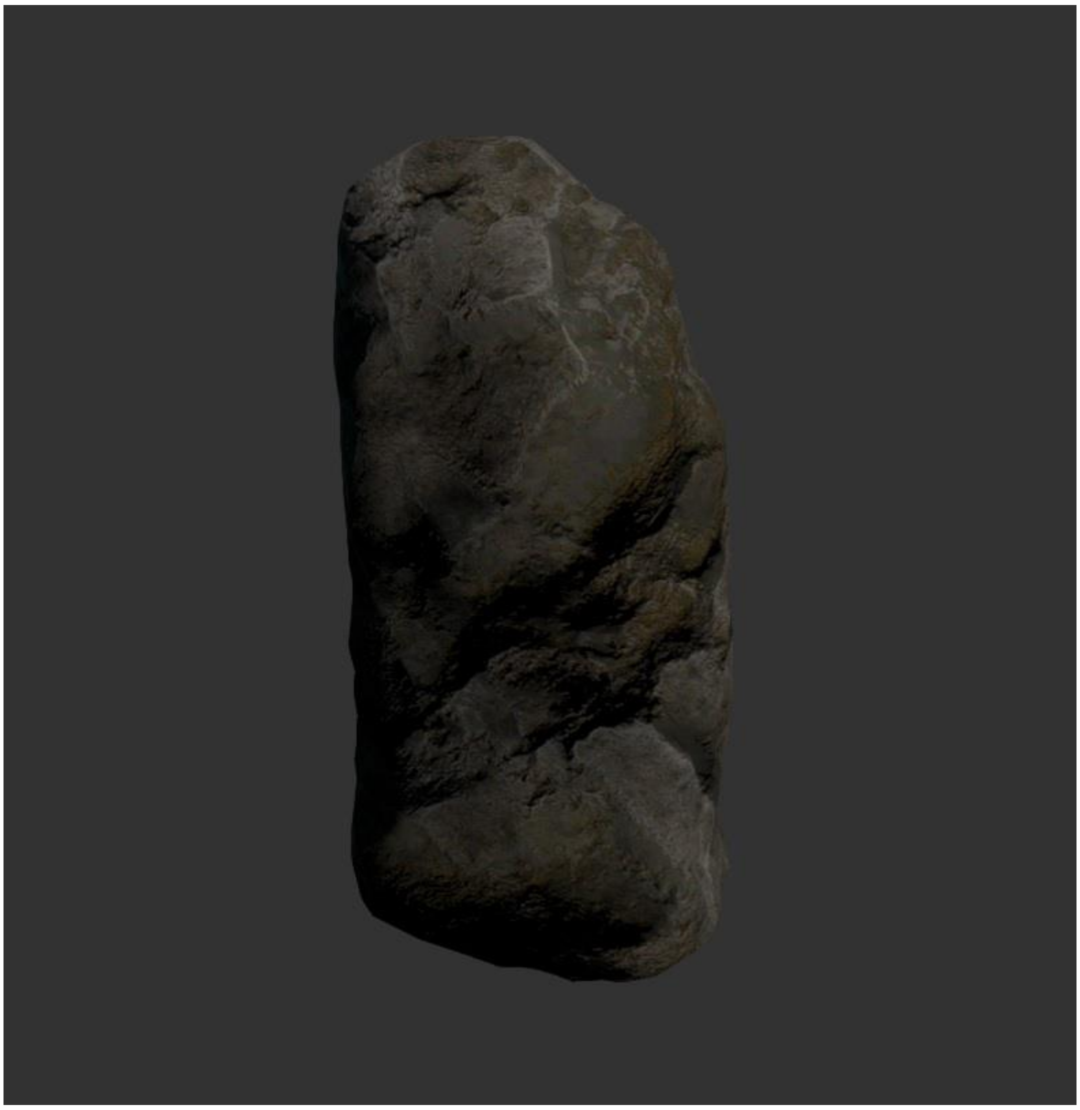


Figure 18. One of the rocks textured on the engine.

3.5 Water

The end of the game is placed in a lake. This lake is created by a cube that is below the terrain and has a water material.

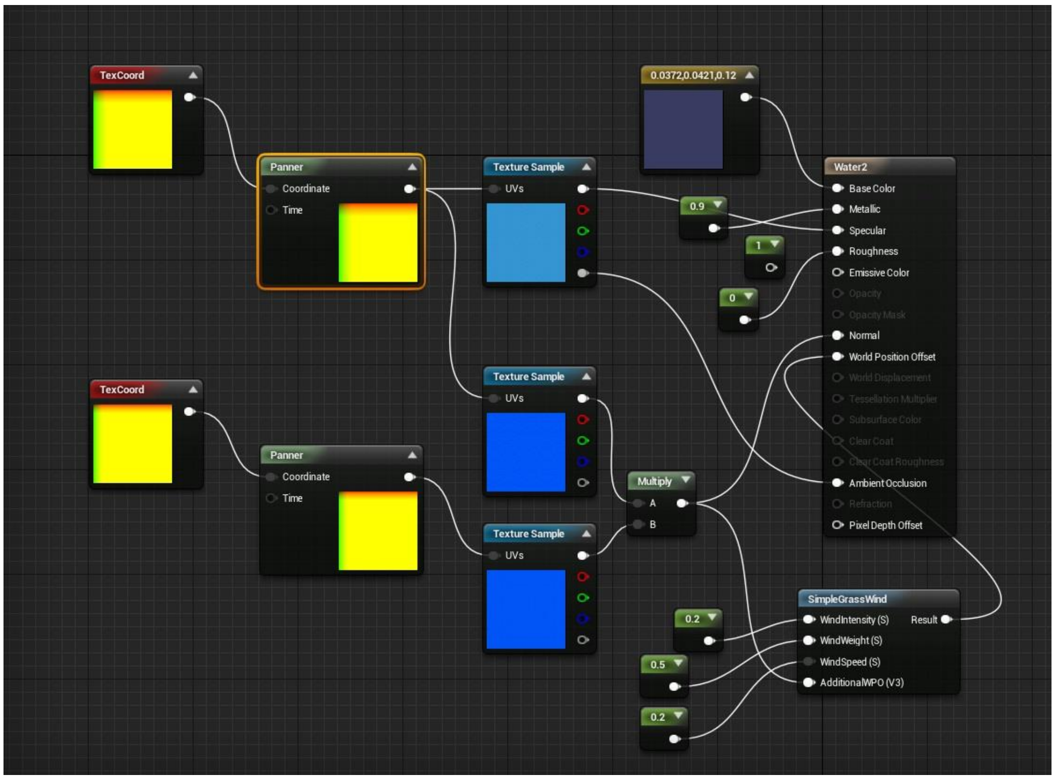


Figure 19. Water material.

To simulate the waves of the water, a "Panner" node was used. This node makes a texture move on the X and Y axis at the speed you want. To get a good effect, two normal maps are used for this. Both are the same, but each one has a different Panner node with a different movement. Then they are multiplied to have two normal maps moving. This combined with a dark blue base color and some tweaking of the PBR values, generates a good looking water effect. But that's not all, if you look closely, the water not only has waves but also goes up and down. This is generated by the "SimpleGrassWind" node. This node, as his name shows, was designed to be used to simulate wind on grass, and can also be used on trees for the same reason. It is used on this project for that purpose. But after realizing what this node really does, it was clear that it was a good idea to use it to simulate water movements. After some tweaking of the values, a satisfactory result was achieved.

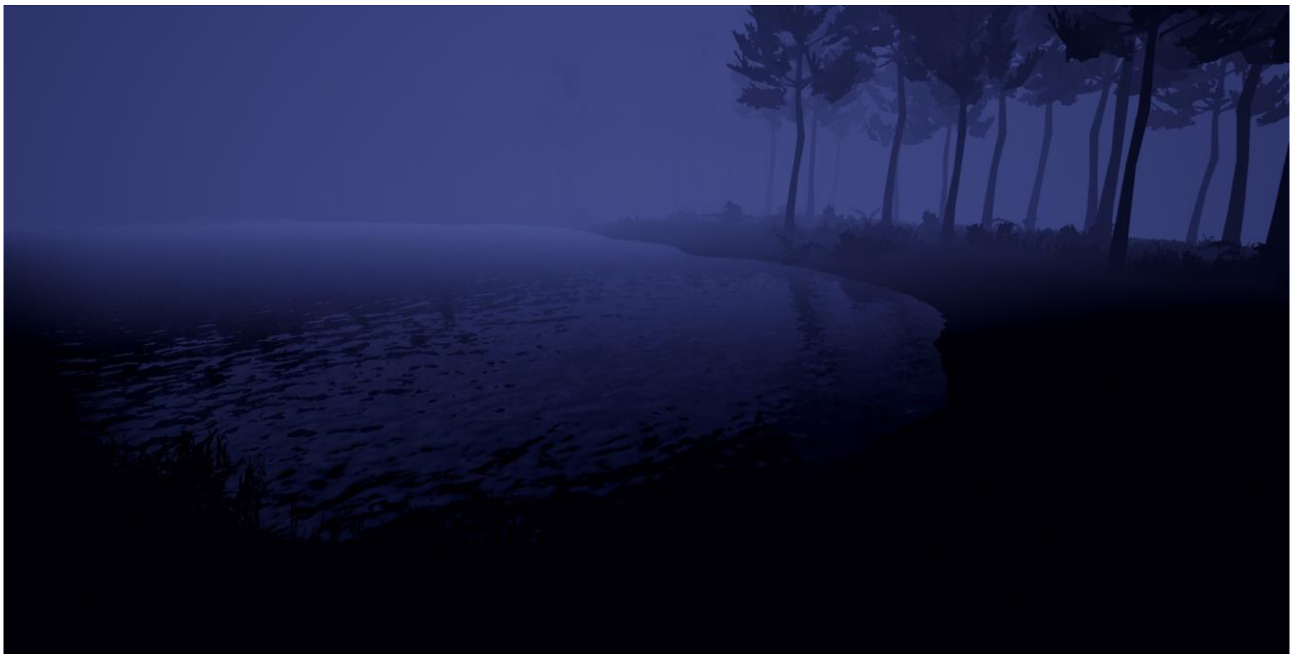


Figure 20. The result of the water.

3.6. Ambience

This part of the project is actually the key to its overall "creepy" and mystery feeling. This is what gives the world its personality. A good amount of thought and time has been used for this to suit perfectly what the project needed to create and transmit. Here you're going to be able to see the results of it.

3.6.1. Main Post Process Volume

Unreal Engine 4 has an actor called "PostProcess Volume". This works like a "filter" for the current camera, but is more complex than that. With this feature, you can change the brightness of the scene, contrast, saturation, colors, exposure to the light, etc. A default scene has all of these values at default, plus the default auto exposure is really aggressive for a night scene. With the Post Process Volume the auto exposure was change to be almost non-existant. The colors have been slightly altered towards blue tones to simulate a night scene, and the contrast and saturation values have been tweaked to improve the overall visuality. Here's the example of the scene with and without Post Process Volume.

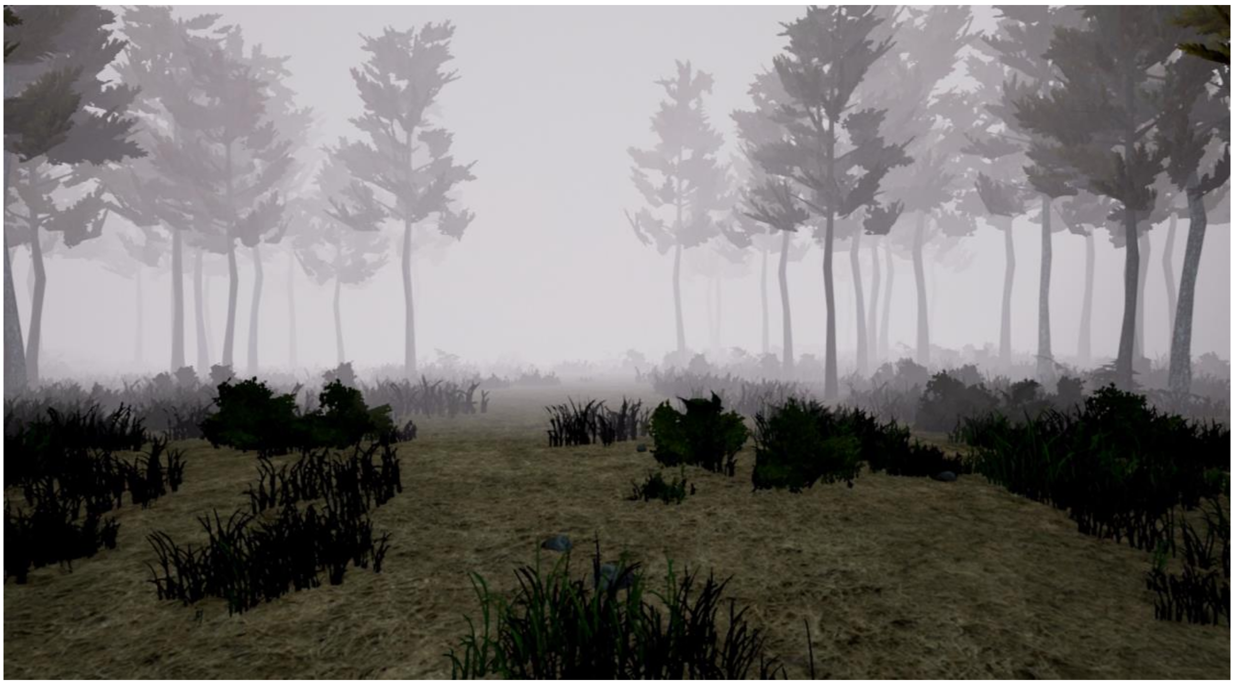


Figure 21. Scene without Post Process Volume.

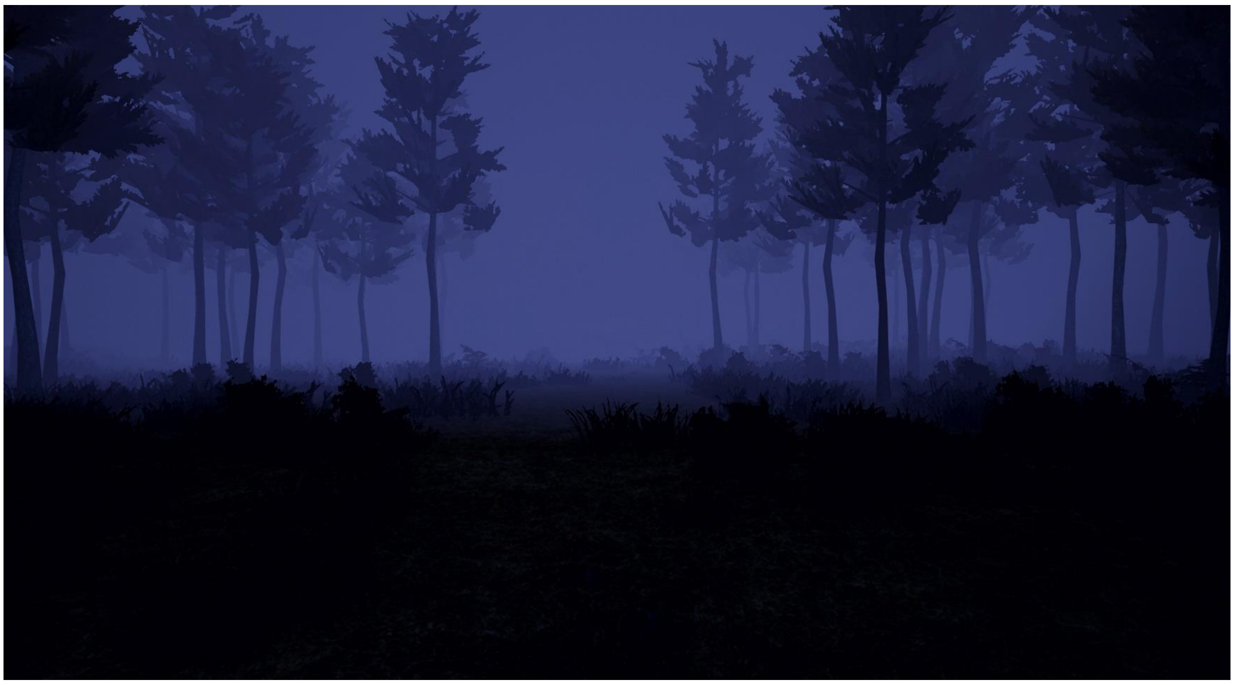


Figure 22. Scene with Post Process Volume.

3.6.2 Fog

The fog is a very important part of the game. Not only impairs the player's vision, which makes the player afraid of what he can't see, but also adds a huge impact to the overall feeling of the game. For this, Unreal Engine 4 has already an "actor" that simulates fog. This fog actor didn't quite suit the needs of the project, so instead of using the default actor, the fog was created in an alternative way. Unreal's Post Process Volumes are not only used only to tweak its parameters to change the global appearance of the world, but can also take some materials as "blendables" to process them. With this system, you can create your own material and use it as a custom Post Process Volume. This is how the fog was created:

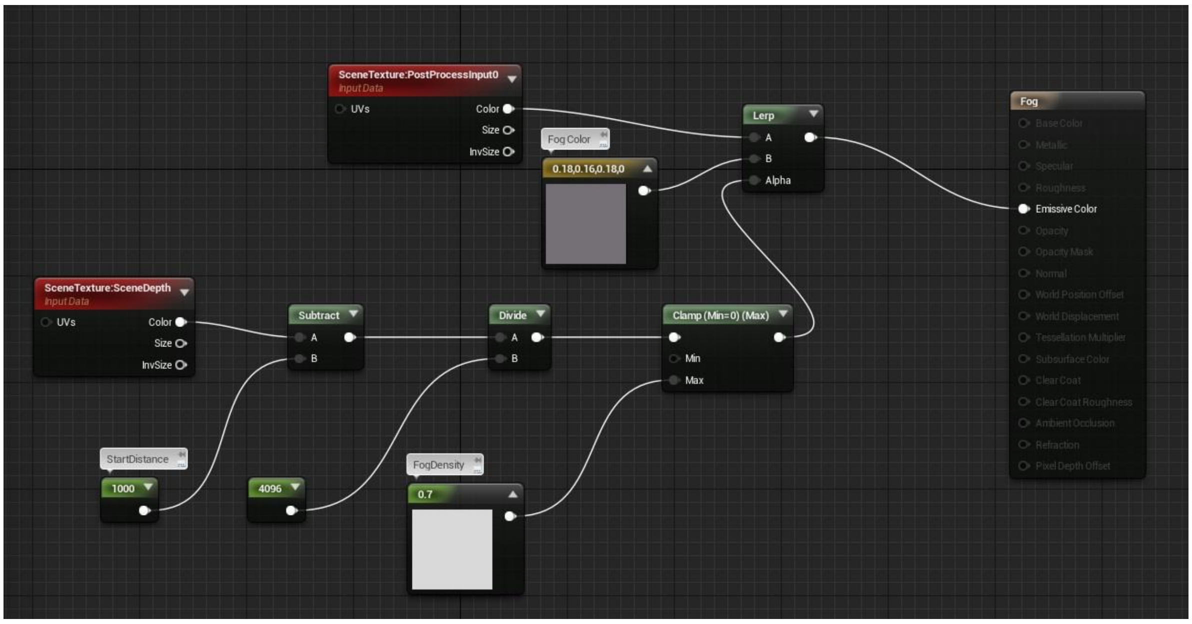


Figure 23. Fog material.

The bottom part calculates the overall density of the fog. It also determines where the fog should start. This means that wherever is the player standing, there will be a space of X units, in this case 1000, without fog. So, the player will see the fog in the distance, and not on its own feet. With the part above it you can change the color and make it look brighter or darker. Below this you will see an example of what the fog provides to the world and how it would look without it. Figure 25 is a screenshot with the lighting slightly raised, otherwise the image would be too dark.

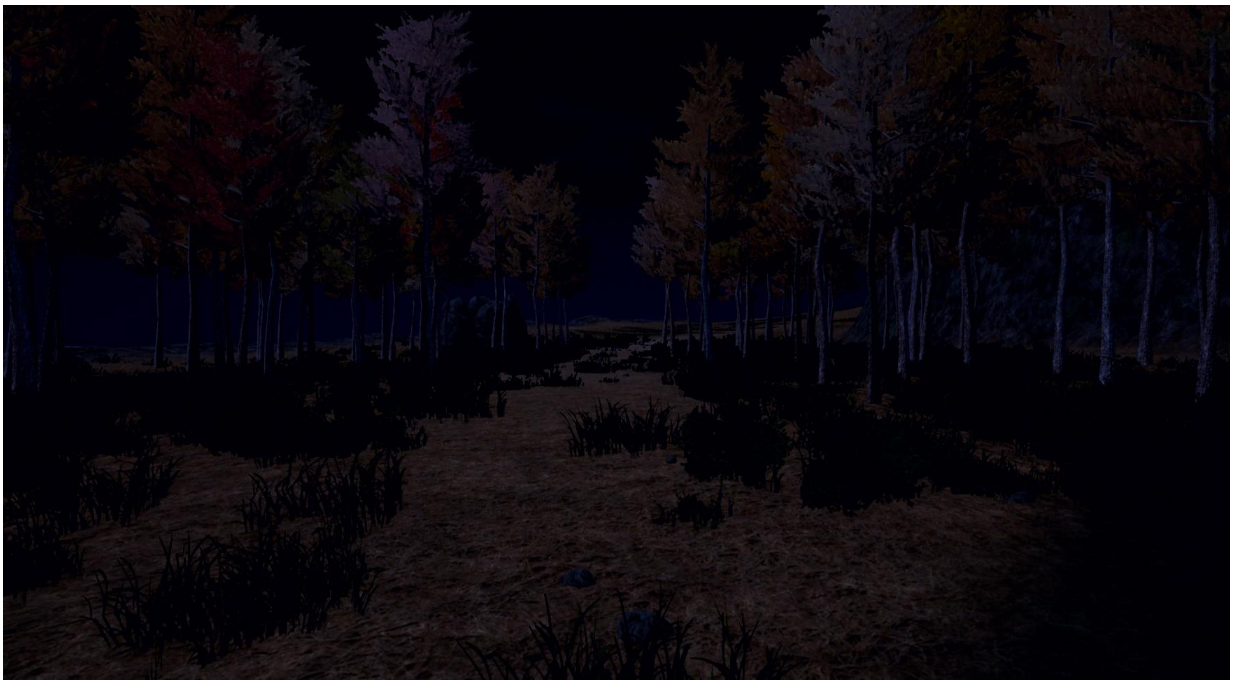


Figure 24. Scene without fog.

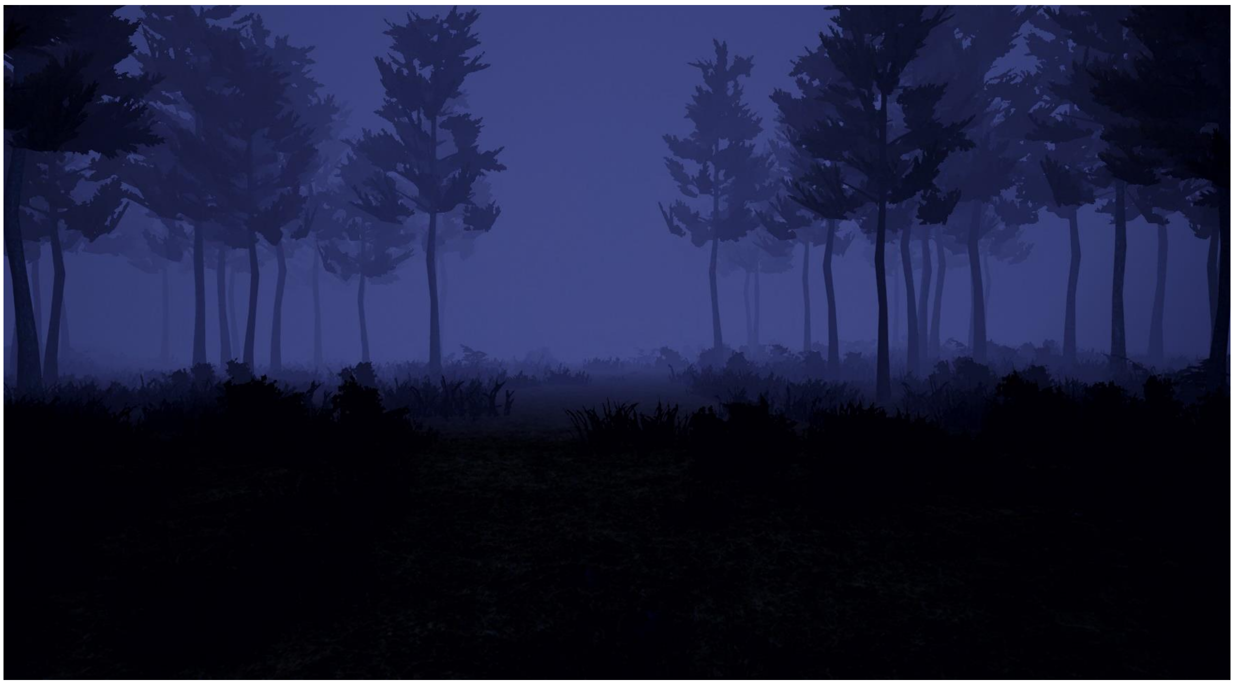


Figure 25. Scene with fog.

4. Functional and Technical Specifications

This section will be used to explain the behavior of all the events programmed on the game. Unreal uses a sort of visual scripting that is called “Blueprints”. On this section more things about it will be explained and images of them will be provided.

4.1. Sound triggers

There are various sound triggers on the game, so one of them will be explained as an example. These triggers consist in a box-shaped collider. When the player walks through it, the collider gets destroyed and the sound is played. The collider is always destroyed first to prevent the blueprint to be called multiple times. The behavior of these blueprints is pretty much straight forward.

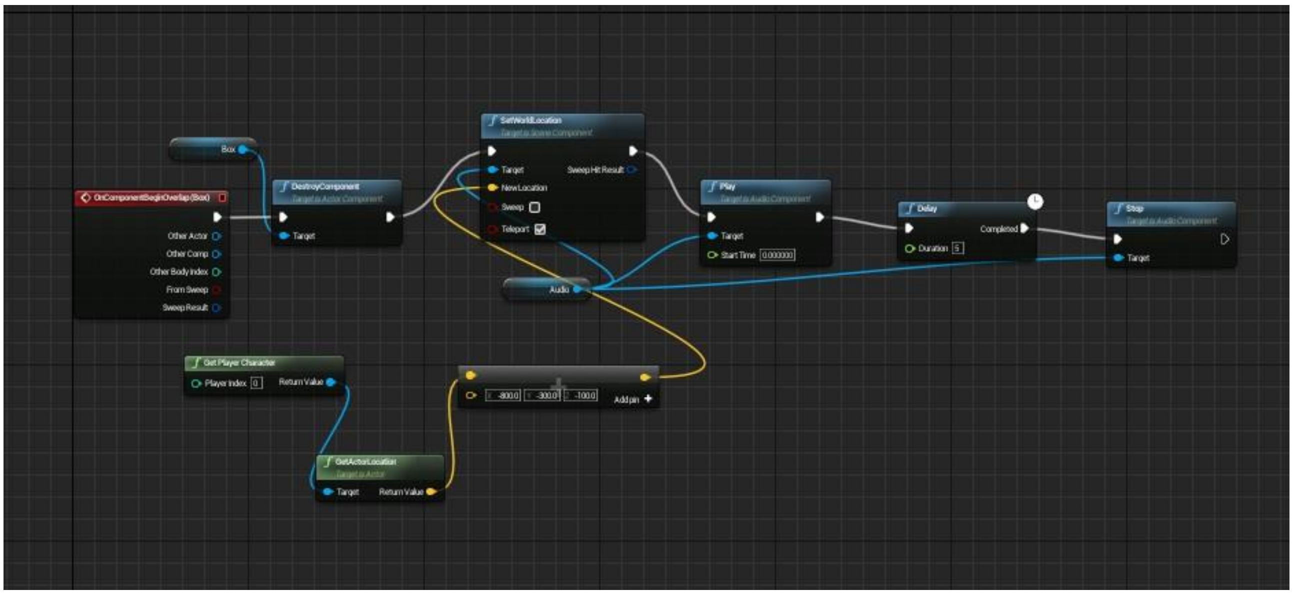


Figure 26. Triggered sound blueprint.

This blueprint triggers a breathing sound. The sounds will be explained later on this document. In this blueprint, when the players steps on the collider, the collider gets destroyed and the audio actor is spawned. The location for this actor is calculated getting the actual location of the player and adding it to a vector that makes the audio to spawn next to the player, at his right or at his left depending on what sound we're talking about. This kind of calculations are done in almost every blueprint. The collider is lengthened widthwise, so the player will always trigger the collider when he's reaching that point of the path, even if he's all the way to the left or to the right. After that, the sound is played. Then a "delay" node is placed to wait till the sound is finished, and then it stops. Without the "stop" node the sound could remain being played on loop.

4.2. Steps sound

This blueprint works similarly to the above one, but making the audio actor move. The sound used on this blueprint is a steps sound. This combined with movement makes the player think someone is following him. The audio actor is also located using the actual player location. Then, it's moved to a relative location from the location the audio actor was spawned. Also, the sound has some Attenuation settings that makes the player to hear it on stereo, this means the player will hear the steps at his right, and if he moves the camera the sound will move accordingly. Later on this document the Attenuation settings will be explained.

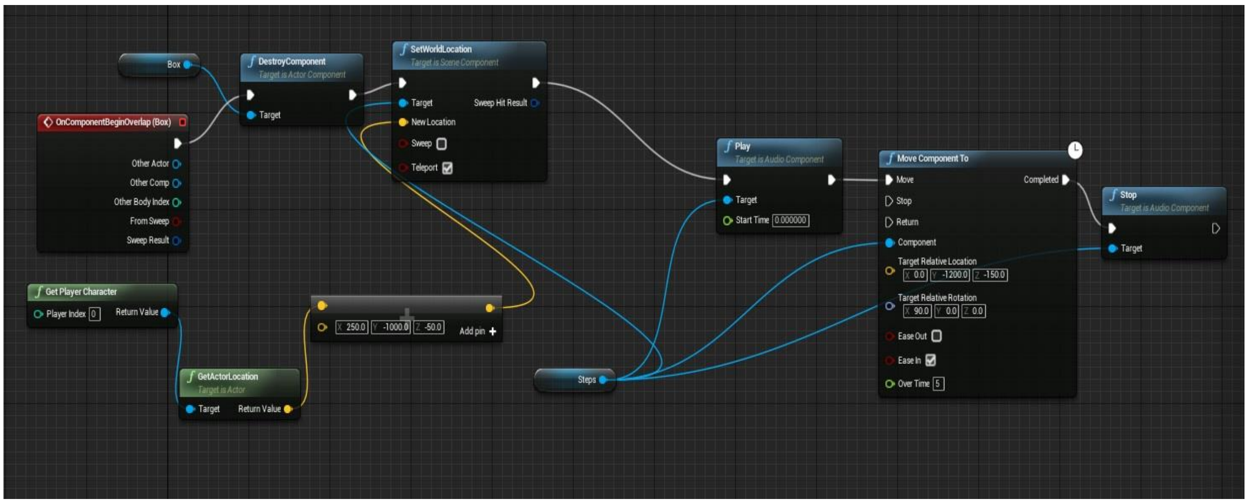


Figure 27. Steps sound blueprint.

4.3. Jump scare

It's time to talk about more elaborated blueprints. This blueprint makes a "monster" appear in front of the player along with two sounds, one sound is a "hit" and the other is one called "after scare". More on that sound later on this document. At the same time, the camera will shake.

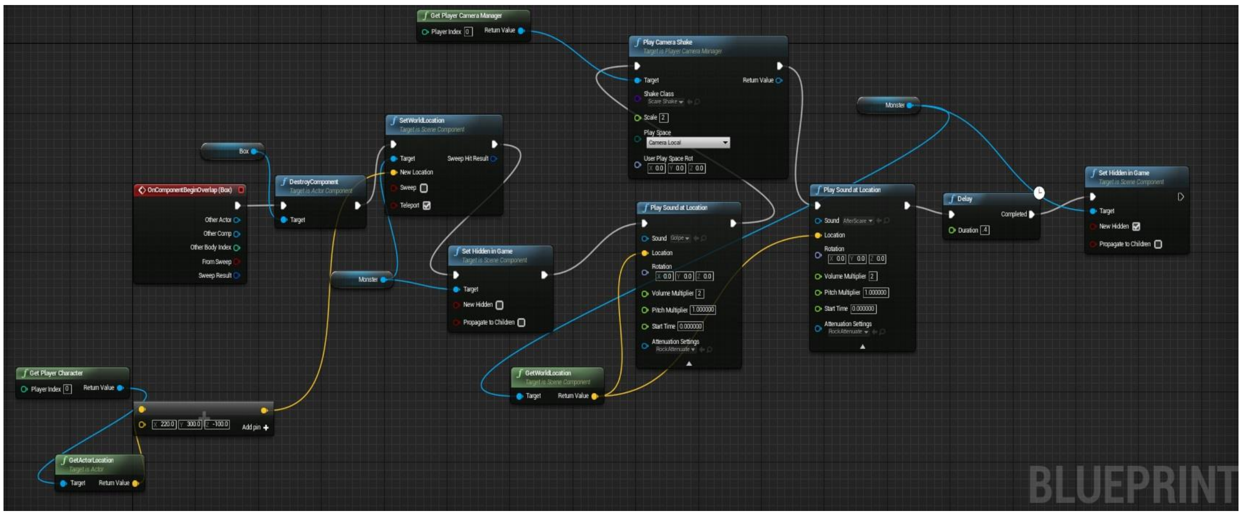


Figure 28. Jump Scare blueprint.

When the player collides with the box, it is destroyed as always first, then the monster is spawned in front of the player. This follows the same calculations as mentioned before, it takes the actual player location and adds a vector to make the actor always appear in front of the player, no matter where does the player collides with the trigger. Then, the monster actor is set to visible and then a hit sound is played. At the same time, the camera shakes using an Unreal function that uses a specific interface for it. Even though the "after scare" play sound node is after the function, it is played instantly at the same time. You can change the volume and pitch of the sounds you use via the blueprint. This is a very useful function that allows you to use the same sound with different volume and/or pitch for different situations. After that, the blueprint waits 0,4 seconds to hide again the monster actor. This took some tests with people to see what time provided the best result. Generally, the lesser time the monster is shown the better, it must be a quick appearance, but it shouldn't be too quick, so the player can recognize the silhouette as something unnatural and threatening. In this case the stop node is not needed, because the "Play Sound at Location" node only plays the sound once, so there's no chance for the sound to enter in a loop.

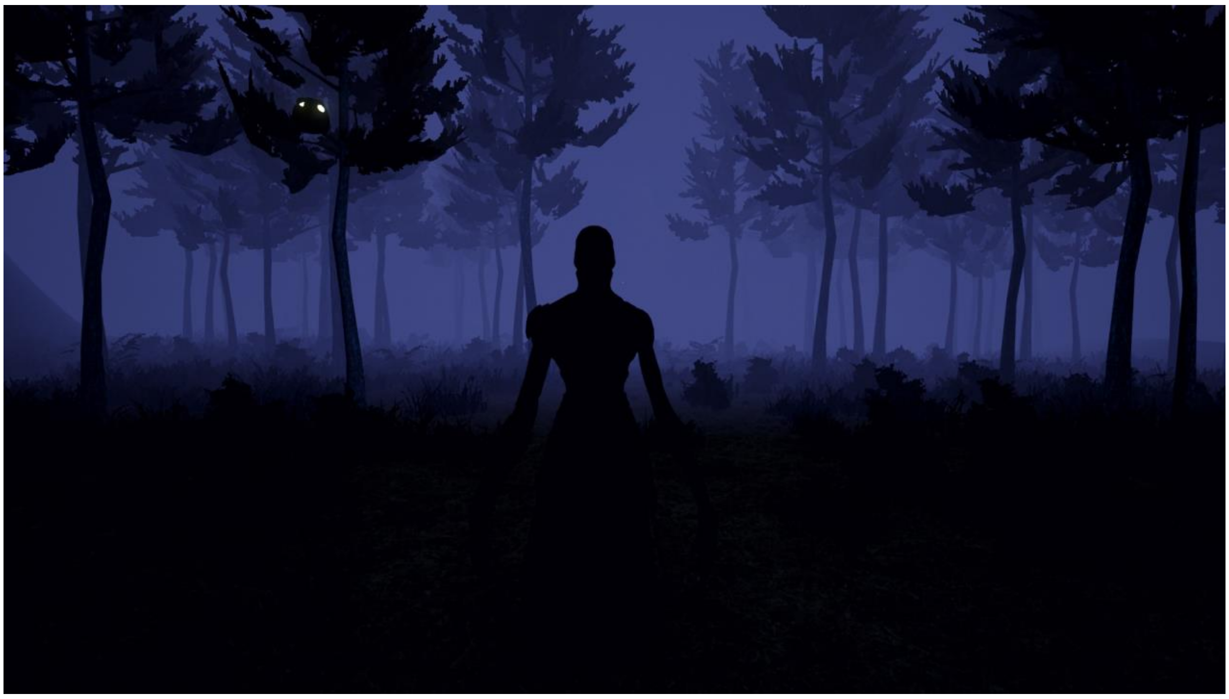


Figure 29. Jump Scare on the engine.

4.4. Rock Falling

When the player walks through the collider of this blueprint, a rock appears up above at his right and rolls out from the ravine to the floor, with its corresponding sound. This is the first event that the player encounters. This could be caused by a natural rockfall, or maybe someone or something can cause this. After experiencing this, the player is forced to start thinking whether he's alone at the forest or not.

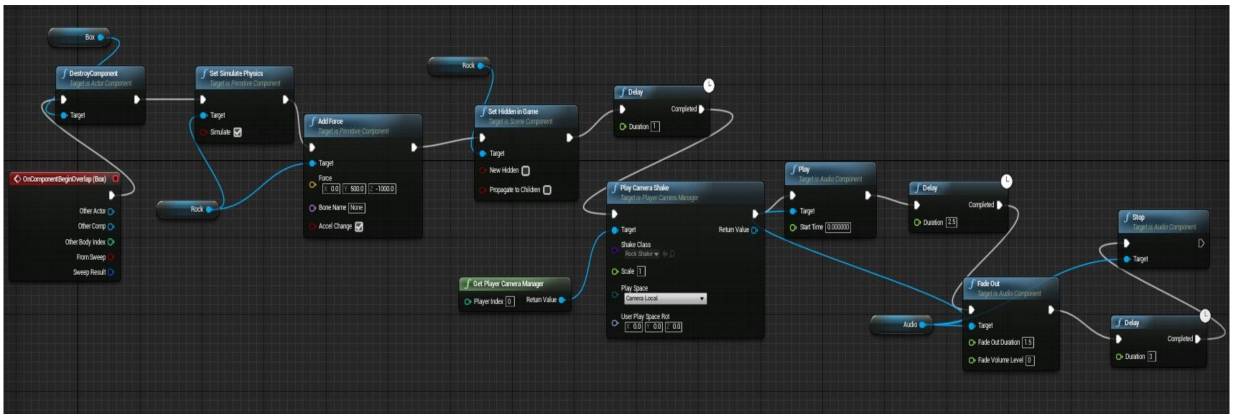


Figure 30. Rock Falling blueprint.

This blueprint consists on a rock already placed on the air from where it should fall. The rock is hidden and its physics are off. Once the player collides with the trigger, the rock begins to simulate physics and stops being hidden in the game. The node "Add Force" adds an external force to the rock to make it fall with strength, like if something pushed it. The blueprint waits till the rock starts hitting the terrain and starts the camera shake and the sound. Both will be better explained further on. When the rock stops moving, it has a "Fade Out" node for the sound to slowly fade out and then finally gets stopped.

4.5. Final Event

Here is where the game ends. At the beginning you're told to look for blue luminescent grass, so the player finds it surrounded by rocks. When he enters, all the rocks blow away and then the monster appears screaming

.

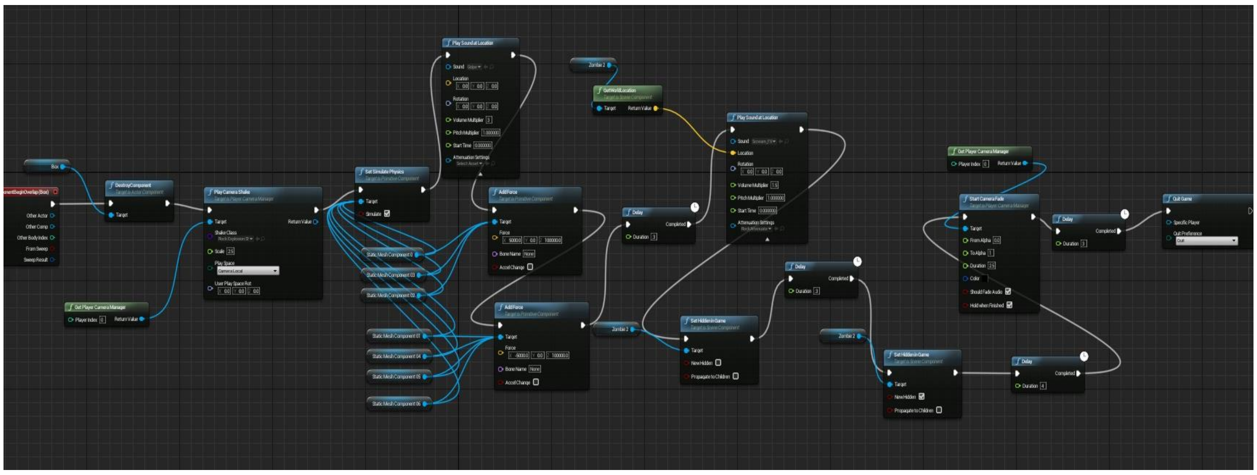


Figure 31. Final Event Blueprint.

All the rocks have their physics turned off. After the player walks through the collider, all their physics are turned on, and a node adds a vector force to them, also a camera shake is applied. This simulates a supernatural force being applied to the rocks from the center of them, right where the monster appears. Two different nodes with two different vectors were needed to make it right because every rock has a different position and rotation. 0,3 seconds after they go away, the monster appears and screams to the player, then it disappears. Then the blueprints waits a few seconds and starts a fade out. This fade out not only fades the alpha value of the camera, but also fades out the overall sound of the game, making it perfect for an end game fade out. After everything is faded out, the game ends.

Figure 32. The Final Event (the image is blurry because of the camera shake).

4.6 Owl Eyes

The purpose of this blueprint is to make the owl open and close his eyes every 5 seconds. At first this blueprint was supposed to be one of the easiest to do, but it turned out to be one of the hardest. The main reason behind this is because the blueprint needed to be always running, this means using the "Event Tick" node as the initiator, which is a node that is constantly being called for every tick of the game. The blueprint needed to make a rotation, then wait, and then make another rotation and so on. If you just use a "delay" node to make the blueprint wait for 5 seconds without moving is not going to work. The blueprint is a one way call only, this means the call only goes forward, never backwards nor on loops. With the delay node, every call makes a tiny movement and then waits 5 seconds to make the other movement. This results in 5 seconds of moving the eye normally and another 5 seconds of delayed movement calls stacked with the actual calls, so the eye starts running at a high speed and losing control. The initial approach was just wrong and needed to be entirely remade.

Let's start with the variables. We have a "Roll Float" with an initial value, which is the variable that will control the rotation. There's two more variables like it, one exactly the same, and the other inversed. This last two variables are used to reset the main "Roll Float" variable. The reset variables are needed because every tick, this value increments to move the eye. The variable "Time Float" represents every tick of the game, and it's multiplied to "Roll Float" to make the movement. "Timer" is the variable that manages the switches that the blueprint needs to do every 5 seconds. There are two booleans: To Close and To stop. One is to see if the eye needs to be closed or opened, and the other is to pause the blueprint for 5 seconds between movements. So the call works this way:

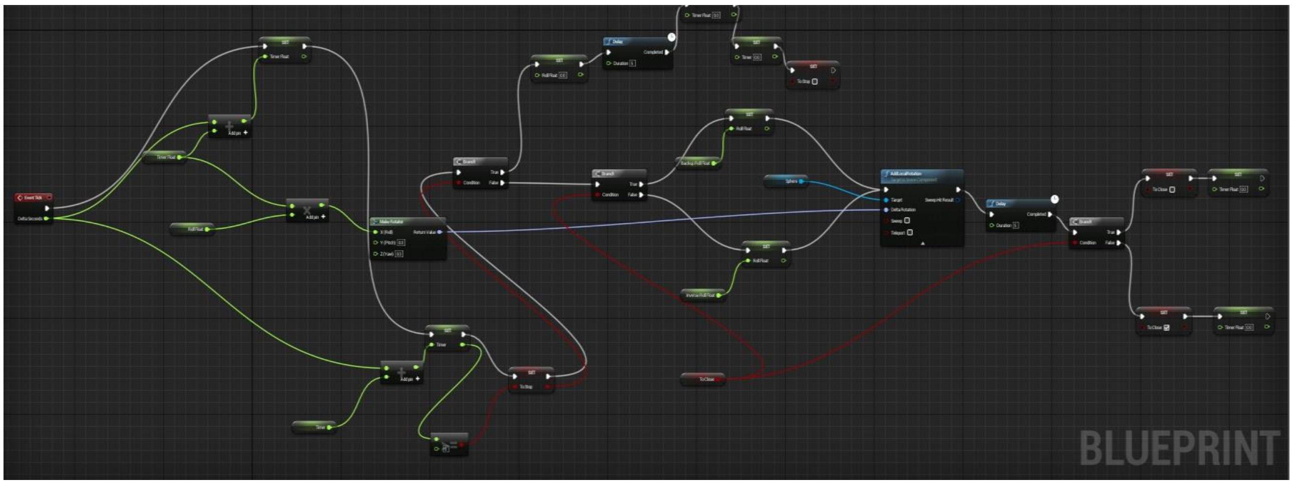


Figure 33. Owl Eye blueprint.

1. Every tick adds a value to the Timer Float, the Roll Float and the Timer. This is then stored into a vector for the rotation. Everytime the Timer variable gets to 5, the boolean To Stop is set true.
2. If To Stop is true, the Roll Float value gets reset to stop the movement, and after waiting the 5 seconds, the Timer Float and the Timer get also reset and To Stop becomes false.
3. If To Stop is false, then it checks if the eye has to open or close. If it has to be closed or opened, the correspondent Roll Float variable is set and it proceeds to rotate the eye. After 5 seconds, sets the condition to true or false, depending on which case is it in, and sets the Timer Float to 0.

This way, we can have an owl with opening and closing eyes every 5 seconds.

4.7. Player controller

The project has its own player controller blueprint. Not only the player can move, but can also crouch. This blueprint also controls two camera shakes and the steps sound. Here's how it looks like:

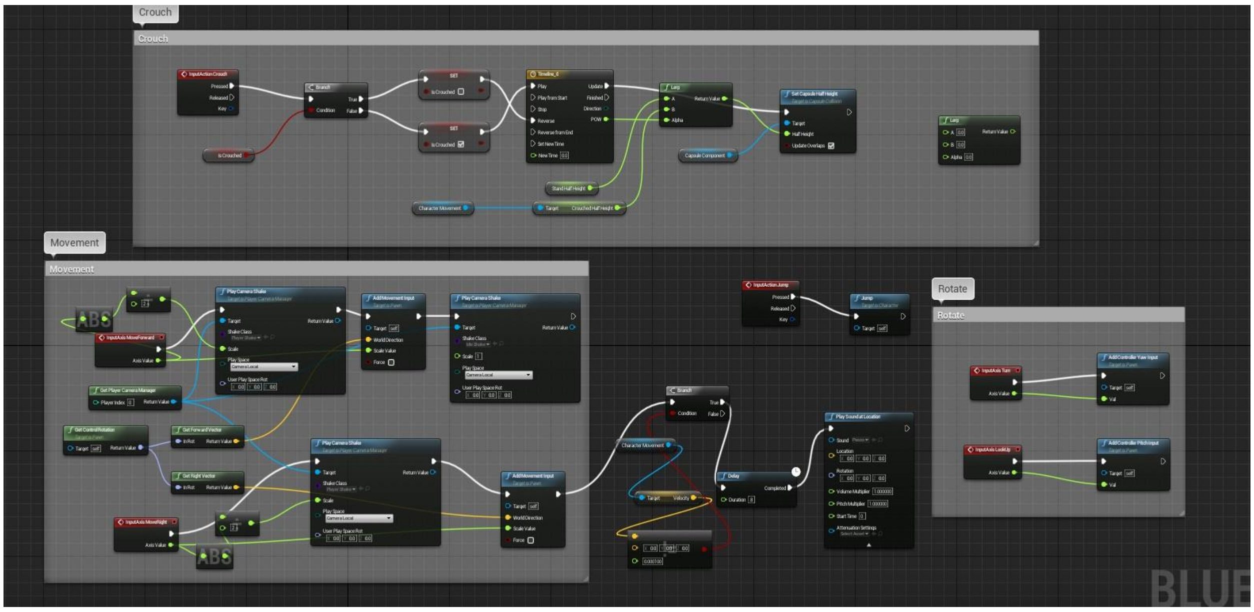


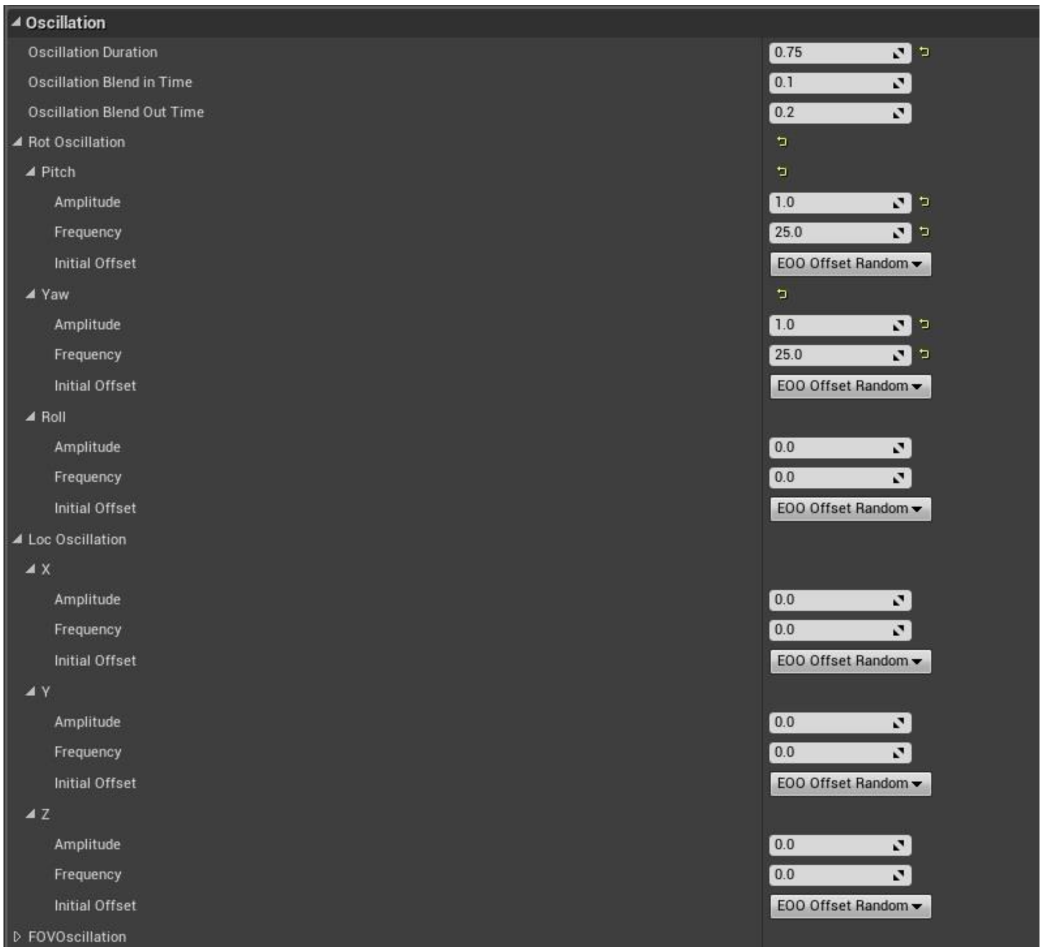
Figure 34. Player controller.

There are two nodes that are continuously waiting for movement inputs, one to go forward/backwards, and the other for right/left. When none of them are being called, the blueprint calls a camera shake for the idle state. This camera shakes simulates the character's breathing. When the player is moving, whether it's forward/backwards or right/left, another camera shake is used to simulate steps. The camera goes up and down as if the character was walking with its legs. Another node gets the velocity of the player, when it's above 0, it waits 0,8 seconds and then plays a step sound. This

synchronizes with the up and down movement on the camera, and also with the overall velocity of the character, so you can see and hear the character walking leg by leg.
For the crouching, a node is waiting for that specific input. Once the player hits the "Ctrl" button, there's a condition that checks if the character is already crouched or not. If not, the character goes to the crouch position. As it is only a camera, this means that the height of the camera gets lower. This is done by two variables that store the standing position and the crouching position, and a "Timeline" node which is the one that allows the position to be changed smoothly, instead of doing it abruptly. All of this is interpolated by a Lerp node and then passed to the node that actually changes the height of the character.
The movement of the camera and the jump function is pretty much straightforward from Unreal's nodes.

4.8 Camera Shake Class

These are a special kind of blueprint class that Unreal has to make a camera shake and use it inside of a blueprint. Tweaking the parameters, you can use it in your blueprints to make the camera shake however you want and whenever you need it. They look like this:



Oscillation	
Oscillation Duration	0.75
Oscillation Blend in Time	0.1
Oscillation Blend Out Time	0.2
Rot Oscillation	
Pitch	
Amplitude	1.0
Frequency	25.0
Initial Offset	EOO Offset Random
Yaw	
Amplitude	1.0
Frequency	25.0
Initial Offset	EOO Offset Random
Roll	
Amplitude	0.0
Frequency	0.0
Initial Offset	EOO Offset Random
Loc Oscillation	
X	
Amplitude	0.0
Frequency	0.0
Initial Offset	EOO Offset Random
Y	
Amplitude	0.0
Frequency	0.0
Initial Offset	EOO Offset Random
Z	
Amplitude	0.0
Frequency	0.0
Initial Offset	EOO Offset Random
FOVOscillation	

Figure 35. Camera Shake class.

You can chose from any angle to make the camera rotate. You can control its "Intensity", duration, etc. This has been really helpful to add value to the character controller and the events of the game. It's a very useful feature that every game engine should have.

4.9 Fade In

The matinée feature of Unreal is used to make cutscenes inside of the game. After getting the fade out done for the end of the game, it felt like the beginning needed also a fade of its own. After some research, it became clear that the best way to do it was with the matinée feature. You have to create a new group, and select the fade option. Then, you select how much time you want it to last, create some key points on the timeline and change their alpha value. After this, placing the matinée actor into the game and making it autoactivate at the beginning makes the game to have a fade in when it starts.

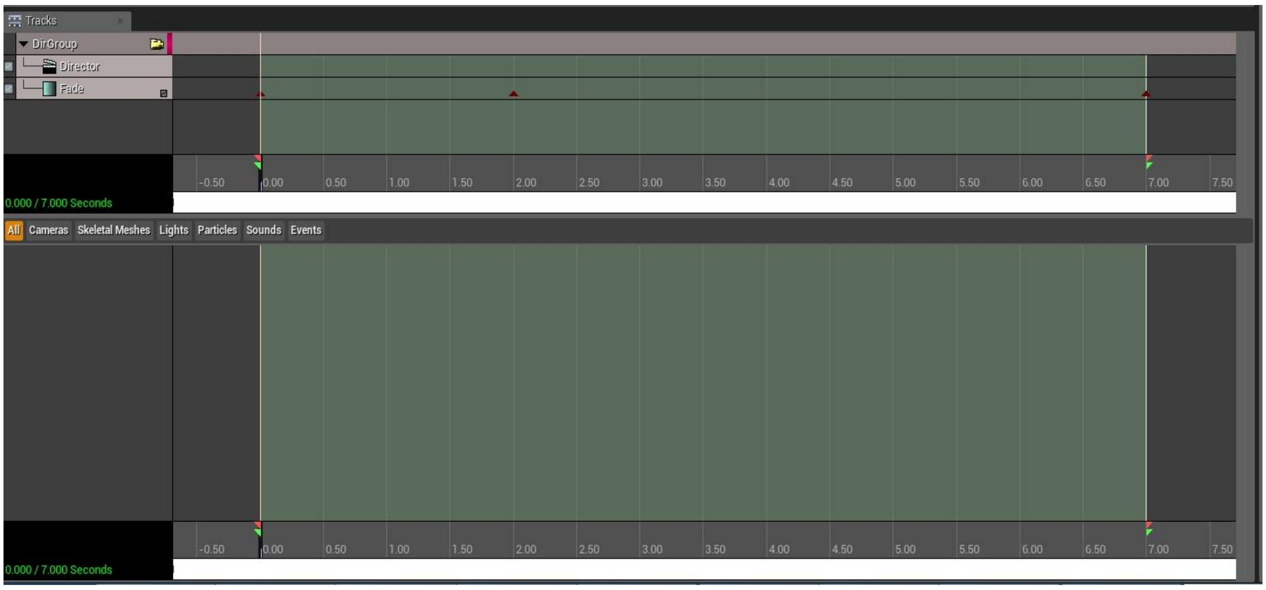


Figure 36. Matinee actor for the fade in.

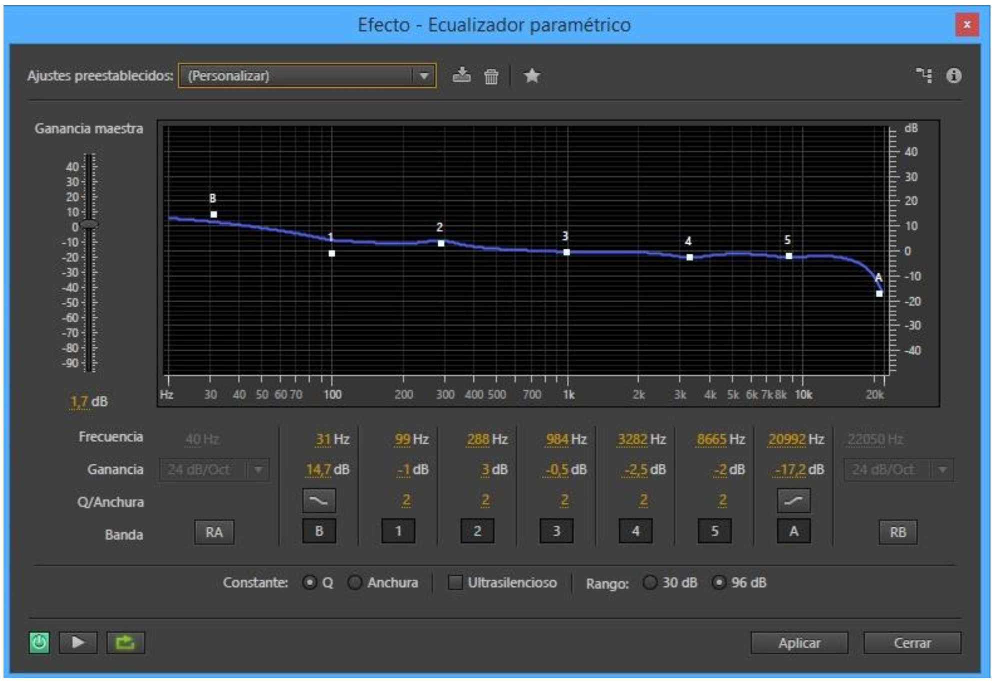
5. Sounds

The sounds that were made exclusively for this project will be explained. Some of the sounds recorded for this project, were recorded by a professional recorder, courtesy of the Jaume I University, and one of them was recorded with the microphone of a smartphone. In this google drive folder [2] you will be able to download and hear every sound that was made for the project, the other sounds that are not here are taken from an external website:

Figure 37. Professional recorder used for this project (Zoom H4n).

5.1. Background sound

After the research that has been done about horror games and their sound, when building the sonorous aspect of the game the first thing that comes in mind is that the game needs its own "background noise", just like every other game has. Standing in a forest at night hearing the wind will probably not scare anyone, but is the background noise that makes you think something is not right. That noise symbolizes a strange presence, maybe the blood pressure of the character's brain who's going insane, etc. After thinking about it and trying some things, a conclusion was made. The sound of a desktop is always on the background and nobody notices it, is the background noise of a room. A desktop was recorded and the result was not bad at all, but it needed to be darker and less "mechanical". For this, Adobe Audition and its parametric equalizer was used.



Ajustes preestablecidos: (Personalizar)

Ganancia maestra: 1,7 dB

Frecuencia	40 Hz	31 Hz	99 Hz	288 Hz	984 Hz	3282 Hz	8665 Hz	20992 Hz	22050 Hz
Ganancia	24 dB/Oct	14,7 dB	-1 dB	3 dB	-0,5 dB	-2,5 dB	-2 dB	-17,2 dB	24 dB/Oct
Q/Anchura			2	2	2	2	2		
Banda	RA	B	1	2	3	4	5	A	RB

Constante: [✓ Q] Anchura ☐ Ultrasilencioso Rango: 30 dB [✓ 96 dB]

Figure 38. Parametric Equalizer.

Features like this one of this program really helped the project to get a good sounding quality and reach what it needed to transmit. After being happy about the result, the noise was only some seconds long, but wasn't loopable. To do this, a few seconds of the end are needed to be cut and pasted at the beginning. Then, the old end and the old beginning must be joined together with a "crossfade". A crossfade is a function that fades out the first sound and fades in the second one.

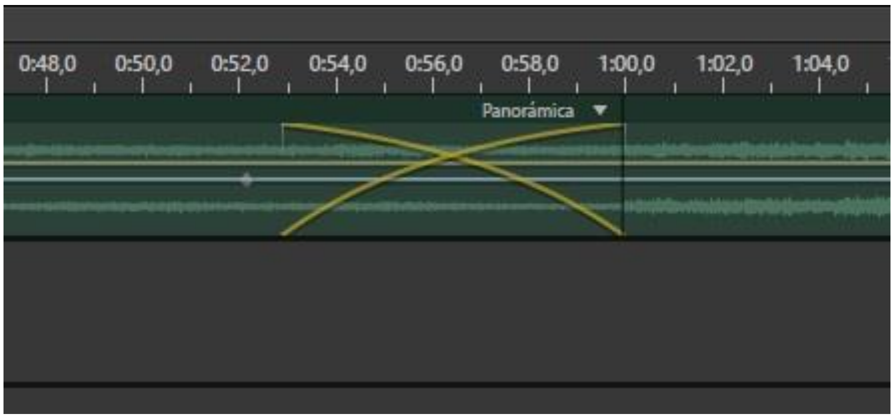
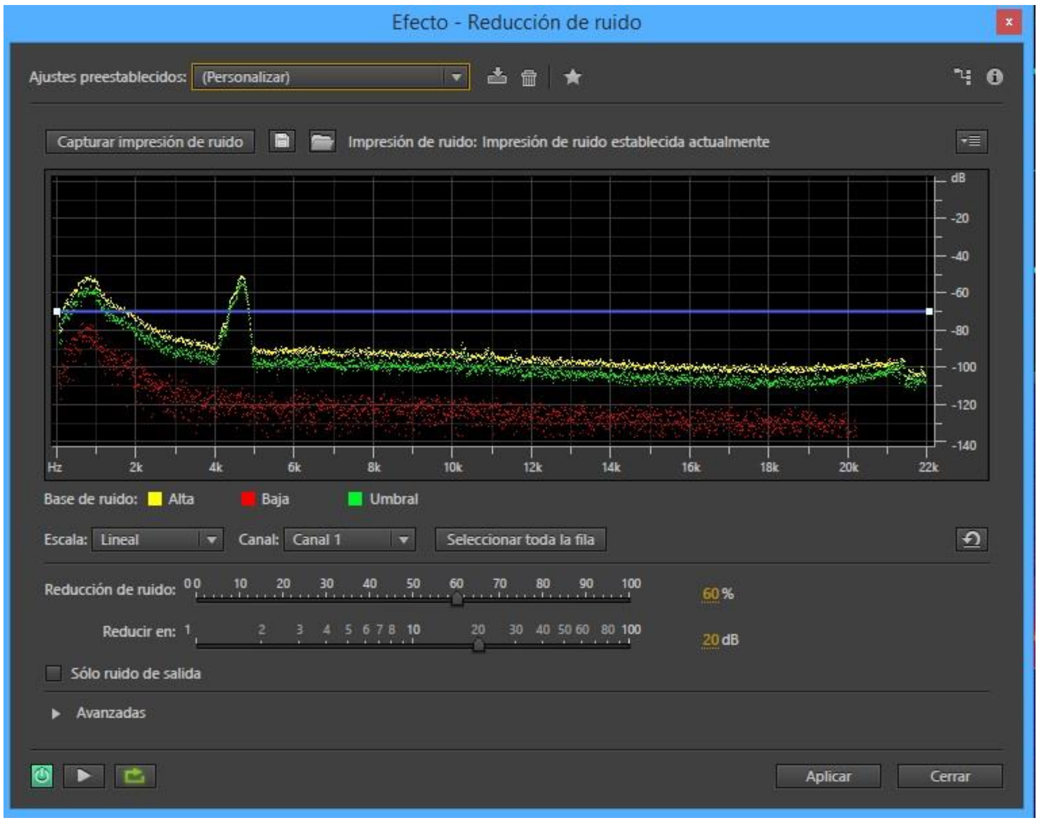


Figure 39. Crossfade.

This way, you can join together two sounds without hearing an erroneous "clipping" sound. With this technique you get a perfect loop in any engine. That's how the background noise was made for this project.

5.2. Steps

This sound can be called a “foley effect”. These kind of sound effects are made on studio, and are a simulation of how something might sound like. For example, instead of recording the sound of a fire, you can record an egg being fried and then edit it. This way you may get a “fire” sound effect that may be even better than the sound that an actual fire makes. So, back to the project, the step sounds were done by folding a newspaper and then putting it inside a plastic bag. To simulate steps, a hand was recorded slightly tapping the plastic bag with the newspaper inside it. After this was done, the recording was moved into Adobe Audition. Once in there all the extra noise that was on the recording was reduced thanks to a special function of this program. It consists in selecting a "silent" part of the recording, where there's only white noise, and then apply the reduction on the whole recording based on that noise fragment.



Ajustes preestablecidos: (Personalizar)

Impresión de ruido: Impresión de ruido establecida actualmente

Base de ruido: 🟨 Alta 🟥 Baja 🟩 Umbral

Escala: Lineal Canal: Canal 1

Reducción de ruido: 60 %

Reducir en: 20 dB

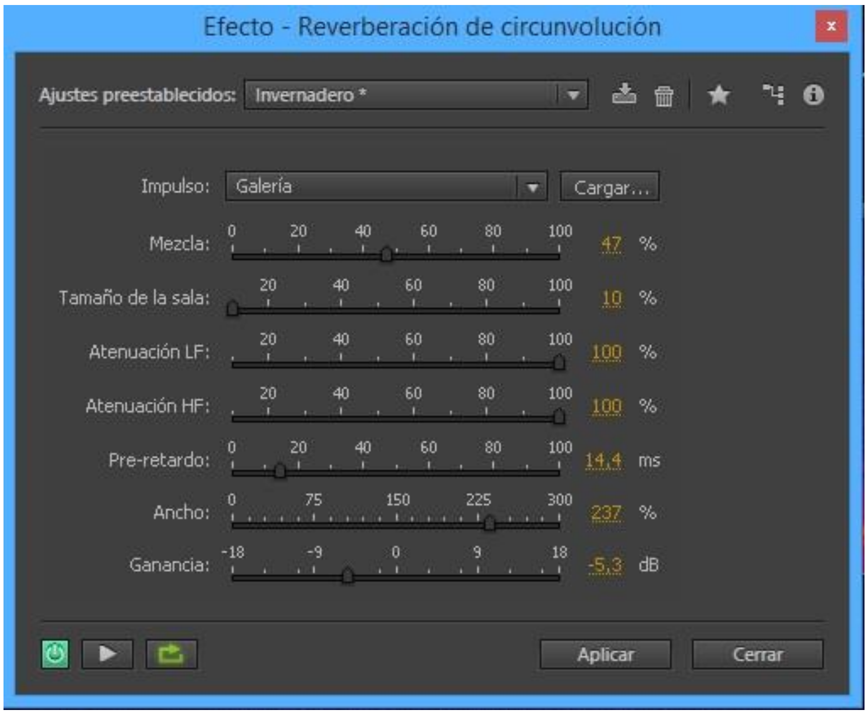
☐ Sólo ruido de salida

Figure 40. Audition's feature for noise reduction.

After this is done, the parametric equalizer is used again to get the better quality out of the recording.

5.3. Breathing

The project needed a ghostly breathing sound to add more sound triggers. It could have been taken from some free website but it would be good for the project if this could be originally recorded and edited without external help. Smartphones don't have the best microphones, but with Audition's noise reduction feature one can get a decent recording. After recording a real human voice breathing deeply and equalizing it, the sound needed an extra effect to suit the game correctly. Here enters in play the convolution reverb effects that Audition has. These effects are already set, but can also be tweaked. After some testing, a satisfactory sound of the recording was finally achieved with one of the effects and some tweaking.


Efecto - Reverberación de circunvolución

Ajustes preestablecidos: Invernadero *

Impulso: Galería

Mezcla:	47 %
Tamaño de la sala:	10 %
Atenuación LF:	100 %
Atenuación HF:	100 %
Pre-retardo:	14,4 ms
Ancho:	237 %
Ganancia:	-5,3 dB

Figure 41. Convolution reverb panel.

5.4. After scare and Final scare

There are some good horror sound effects already made and prepared to be freely used. These effects are made with synthesizers. Even though there are being used two external sounds for this project, it would be good for the project to create its own synthesizer sound effects. For this, Ableton Live was used along with Kontakt. Kontakt is a software sampler, used to load virtual instruments to be played. This software used with a music production software like Ableton, is used to create music and all kinds of sounds. After searching for some instruments and finally finding two good ones that suited what the project needed, some notes were played on a MIDI keyboard, mixing low pitched notes with high ones, thus creating a haunting sound effect for the project. The Final Scare is also designed to be played on an infinite loop, so the same technique used to loop the Background noise was used for this.

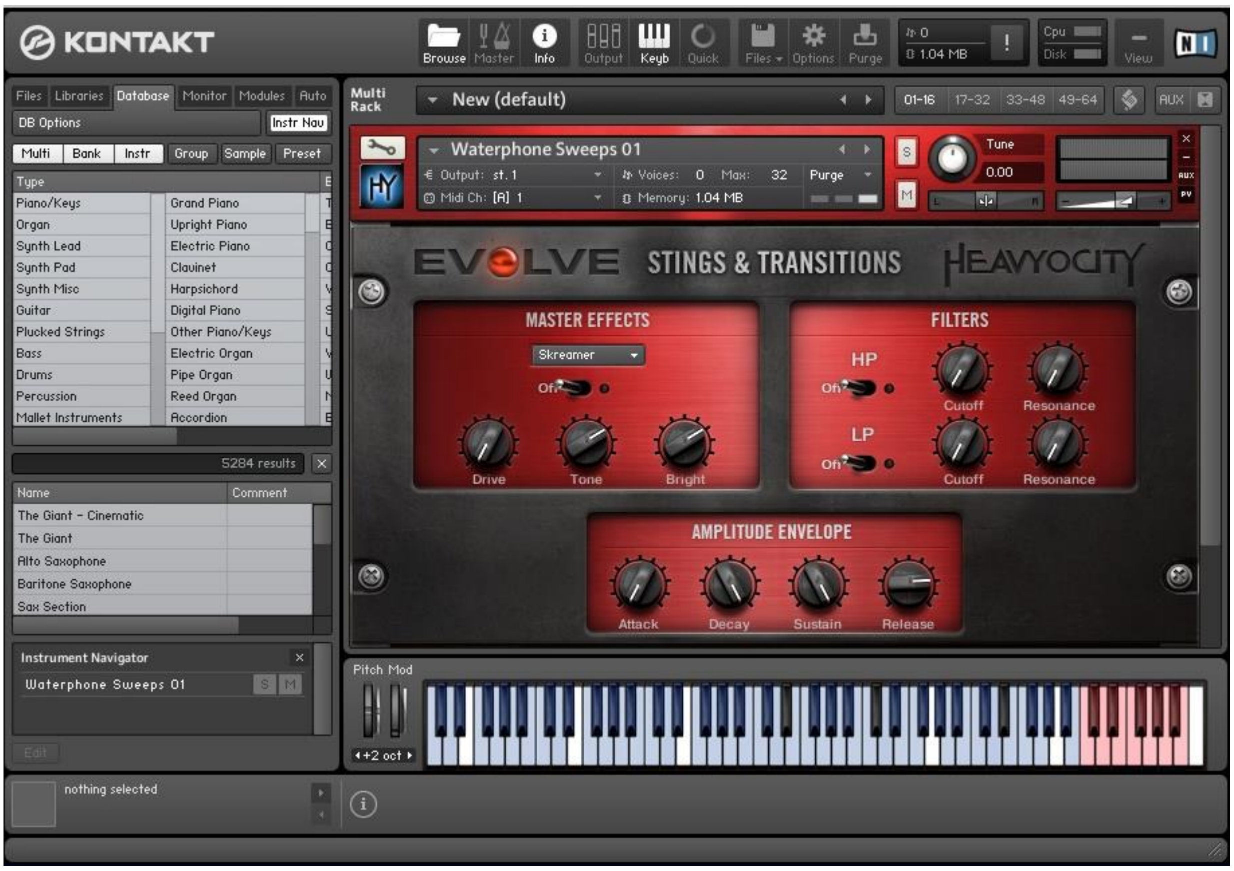


Figure 42. Kontakt interface with one of the instruments used loaded.

The Final Scare sound is placed on the end of the game as a sound actor. It also has its own attenuation effects. This is a function that Unreal uses to spatialize sounds. This is what makes a sound to be heard on 3D. In order for the sound to be heard only with one ear if it's totally on one side of the listener, the sound must be in Mono and not in Stereo. If the sound is in stereo, it will only fade with distance, but be heard by both ears no matter where the listener is. The main idea of this, is that when the player is reaching the end of the game, he starts to hear a sound that really haunts the player and makes him alert to what can happen.

6. Project Monitoring

Here will be explained how the project got developed step by step. From what was planned on the technical proposal to what has been improvised or changed. A lot of problems have emerged during the making process of this project, so some changes needed to be done in order to finish it. On this stage of the work is where one can really appreciate and learn what to plan a project is and how important it is.

1. **Doing a professional technical proposal where the project is explained.**

Here's where the project started. This has been a great help for speeding things up, but some mistakes were made on the planning that had to be changed.

2. **Investigating horror games, their environments, music and sound effects.**

This was an important part of the project. In order to succeed in the industry, you first have to see what are your competitors doing, what works and what doesn't. An analysis of what works and what not on horror games was done. It was really important to know where the fear and tension really comes from. This is how some good ideas for the project were born, like the continuous background noise. Every fear game or movie has it, but they're all different and have its own identity.

3. **Modelling the level.**

This task was mostly straightforward, although it gave some trouble. To import a terrain on Unreal, you can't just import the ".obj" file. It's better to use Unreal's terrain manager. So to import your modeled terrain into Unreal, you have to generate the height map and import it. This caused a little bit of travel to get right, and took some tweaking inside Unreal as well.

4. Texturing the level.

This part is one of the tasks that took longer than expected to accomplish. So much problems appeared and every change that was made took time to compile. A good amount of research was needed to learn how to make the textures look right.

5. Modelling and texturing the asset.

To avoid overextending the time needed for the project, a few external were used, such as the vegetation and the 3D characters. The assets that were made for the project are the rocks, the walls of the beginning of the game and an owl with bright eyes to bring a special ambient to the game. Texturing some of the assets was also a challenge. If this is not done carefully, it results in a lot of UV seams all over the asset, which makes it look wrong.

6. Doing the soundtrack and designing the sound effects.

This part went more or less as planned. It was a rewarding task, every hour spent on searching for external sounds, recording or editing was really noticeable on the final project.

7. Programming all the interaction in the game engine along with the sounds.

This is the part that took the longest time to do by far. There was an idea of how the events should be and feel in general, but their content was still open. As explained above, every event that has been programmed took time and effort, long researches on Unreal's documentation page and a lot of try and failure tests.

8. Documenting the final results of the project and preparing a presentation for it.

This part was really useful to stop and analyze all the work done and how much one can learn with it. Sadly, some hours had to be taken away from this part.

Tasks	Time	Objectives
1. Doing a professional technical proposal where the project is explained.	15h	4 and 5
2. Investigating horror games, their environments, music and sound effects.	10h	5
3. Modelling the level.	20h	1 and 5
4. Texturing the level.	45h	1 and 5
5. Modelling and texturing the asset and having them implemented on the game engine.	50h	1, 3 and 5
6. Doing the soundtrack and designing the sound effects.	40h	2 and 5
7. Programming all the interaction in the game engine along with the sounds.	85h	3 and 5
8. Documenting the final results of the project and preparing a presentation for it.	35h	4 and 5

Table 2. Final table of time.

The Table 2 shows exactly how the time had to be reassigned from some tasks to others. This was the result of unexpected problems or needs that came along the working process.

7. Conclusions

This project is the culmination of everything that we learnt over the course of this degree. Here was put on practice our knowledge of videogames making. In this particular case, this knowledge also was used as background to be able to learn new software, different from what we used on the degree, like Unreal Engine. For example, this engine uses visual scripting for programming, and even though we haven’t really touched that topic, knowing how to program was an essential part to be able to understand and use blueprints. This is where all goes down to, knowing first the basics, and then you can learn to use every software without too much complication. The documentation part was tedious but necessary to understand and learn how planning works, and to get to know one’s possibilities and limitations.

The project was successfully completed in time. The idea was to create a horror environment that created tension to the player and scare him, and so it did. Several tests were made with people inside and outside of the degree, and the results were promising. It is true that every person has a different tolerance to fear, but when the player focuses only on playing the game with headsets the tension is visible.

This game achieved its own personality and identity. In a future, without limiting the time and the human resources, this demo could easily be developed into a full game, with a story and more levels. On the bibliography you’ll be able to find a link for a playable demo.

8. Bibliography

[1] Gamasutra webpage:
http://www.gamasutra.com/view/news/200150/Why_ambient_sound_matters_to_your_game.php
[2] Unreal Engine documentation:
https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/PhysicallyBased/
[3] Folder with the sounds that were made for the project:
https://drive.google.com/open?id=0B1gJncMeKxfSYzZsckFOT0JQTVU (26/09/2016)

External Assets Source

https://www.freesound.org/
https://www.assetstore.unity3d.com/
https://forums.unrealengine.com/
http://tf3dm.com/

Playable demo

https://drive.google.com/file/d/0B1gJncMeKxfSWTJDcjd6dGNpM1U/view?usp=sharing
(26/09/2016)

9. Annex - External Assets used

As this project has a time limit of 300h, there was not a chance for a single person to be able to do all the assets needed for it, while also programming everything. Also, repeating tasks will devalue the project, so the best choice was to do some assets and then use some other external assets and focus on other tasks to have more variety in the work done. On the image below, you will see the 4 external assets that were used on this project:

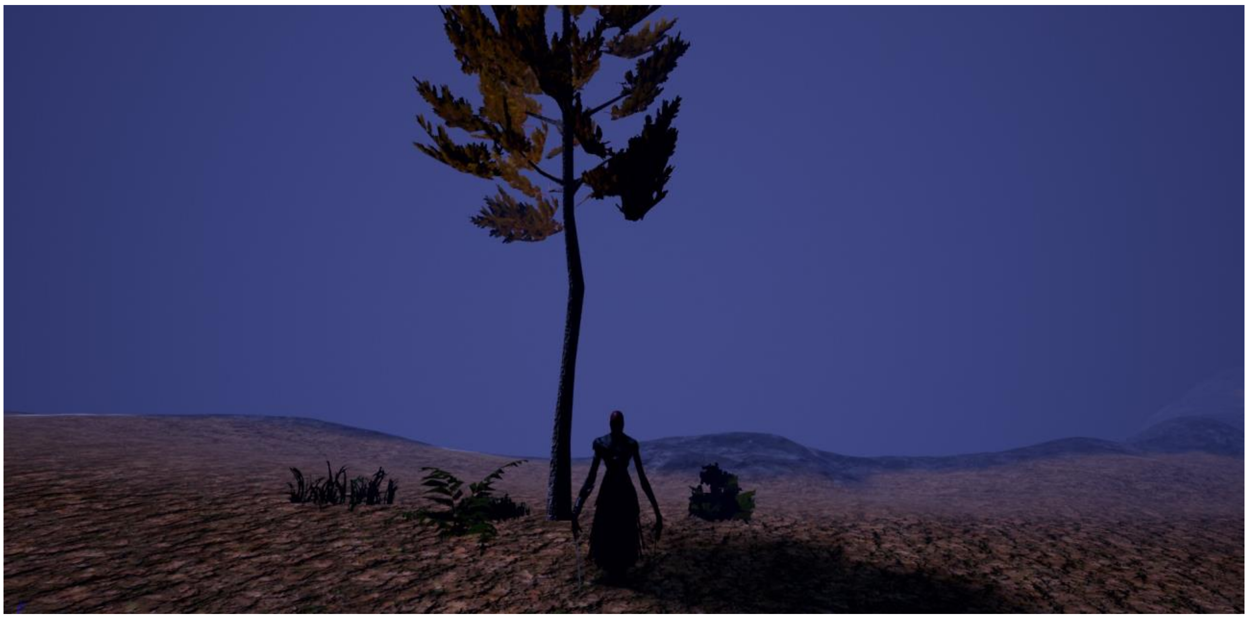


Figure 43. External assets used.

Also, as already mentioned before, some sounds have been taken from other websites mainly because most of them could not be recorded due to not having proper equipment nor human resources. These include:

- Owl sound.
- Cricket sound.
- Scream sound from the final event.
- A “hit” sound that sounds like a bass drop.

These were all free assets taken from some websites that you can check on the bibliography.