UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Vicus Ragnarök: Preproducción y producción de un entorno 3D para videojuego.

Trabajo Fin de Grado

Grado en Comunicación Audiovisual

AUTOR/A: Valero Domingo, Alejandro

Tutor/a: Pérez Esteban, José Antonio

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandía

Vicus Ragnarök: Preproducción y producción de un entorno 3D para videojuego.

Trabajo Fin de Grado

Grado en Comunicación Audiovisual

AUTOR/A: Valero Domingo, Alejandro

Tutor/a: Pérez Esteban, José Antonio

CURSO ACADÉMICO: 23-24

Resumen

El Trabajo de Fin de Grado se centra en la creación del entorno 3D para un videojuego que transporta al jugador a la época vikinga en la costa islandesa. Los escenarios naturales incluirán todo tipo de modelos para dotar de mayor credibilidad a los espacios, incluyendo los interiores de edificaciones. En este juego, el protagonista se despierta solo en un barco mientras observa cómo sus compañeros se alejan hacia la guerra en el horizonte. Su misión es descifrar los enigmas que le permitirán unirse a la batalla antes de que el temido Ragnarök llegue. Para lograr una experiencia visualmente impactante sin sacrificar el rendimiento del juego, se economizará al máximo el número de polígonos manteniendo una estética realista en los modelos y materiales. Además, se contará con la colaboración de Juan Manuel Lima para la creación de una banda sonora que complemente y enriquezca la inmersión del jugador en este mundo vikingo. Dado que esta parte de música original y efectos de sonido no está directamente en nuestras manos, solo podrá incluirse si a fecha de presentación del presente TFG, estuviera lista. Así mismo, si bien no se contempla la posibilidad de completar todas las funciones de jugabilidad, sí que se implementarán los modelos en el motor de juego elegido con la posibilidad de moverse en tiempo real por los espacios creados.

Palabras clave: videojuego; gráfica 3D; optimización; Unreal.

Abstract

The Final Degree Project focuses on the creation of the 3D environment for a video game that transports the player to the Viking era on the Icelandic coast. The natural landscapes will include all kinds of models to provide greater credibility to the spaces, including the interiors of buildings. In this game, the protagonist wakes up alone on a boat while watching his companions sail towards war on the horizon. His mission is to solve the enigmas that will allow him to join the battle before the feared Ragnarök arrives. To achieve a visually impactful experience without sacrificing the game's performance, the number of polygons will be economized as much as possible while maintaining a realistic aesthetic in the models and materials. Additionally, Juan Manuel Lima will collaborate to create a soundtrack that complements and enriches the player's immersion in this Viking world. Since this part of original music and sound effects is not directly in our hands, it can only be included if it is ready on the date of presentation of this TFG. In addition, although it is not planned to complete all gameplay functions, the models will be implemented in the chosen game engine with the possibility of moving in real time through the created spaces.

Keywords: video game; 3D graphics; optimization; Unreal.

Índice

Capítulo 1. Introducción.

"Vicus Ragnarök" es un proyecto de investigación que se centra en la creación de un entorno 3D de un videojuego ambientado en la época vikinga, concretamente en la costa islandesa.

Mi interés por los videojuegos y el CGI (Computer Genereted Imagery) ha sido constante desde pequeño. Este trabajo no solo representa una oportunidad para poder profundizar en el conocimiento técnico de la creación de entornos virtuales, sino también para explorar una época histórica que también me fascina desde hace tiempo.

La principal motivación para hacer este proyecto viene de mejorar la experiencia que tiene el jugador creando un equilibrio entre unos gráficos realistas y un rendimiento adecuado. Consideramos que la necesidad de optimizar estos recursos gráficos es un área de interés dada las crecientes demandas de detalle y rendimiento de los usuarios de videojuegos.

El objetivo general del trabajo es diseñar y optimizar los elementos 3D que conforman el nivel del videojuego, siempre garantizando un rendimiento adecuado.

Es importante destacar desde el principio que se parte de unos diseños previos (en papel) desarrollados por el equipo artístico que colabora en el proyecto. Estos diseños podrían ser motivo de un trabajo paralelo al que se plantea ahora, para definir la estética y funcionalidades del juego (en base a un estudio de mercado), acotar referentes -en función de esos planteamientos- y para, finalmente, mostrar el proceso razonado completo desde sus propuestas iniciales hasta las finales en las que nos basaremos.

Centrándonos en nuestro proyecto y nuestros objetivos, se han definido varias etapas a seguir. Lo primero, la preproducción, que incluye la planificación y la estrategia que usaremos para modelar, texturizar y optimizar. Durante esta etapa, se aclaran los requisitos técnicos y artísticos del proyecto, asegurando una base para las siguientes fases.

Posteriormente, en la producción, se lleva a cabo el modelaje, la integración de los modelos en el motor de juego que estamos usando y la aplicación de texturas. Se pone especial interés en darle realismo al paisaje natural e implementar los objetos en los interiores de las estructuras que lo permiten, buscando siempre una estética realista.

Después, una vez hecho esto, se procede a la optimización de los modelos y el entorno. Esto incluye reducir el número de polígonos si es necesario y la implementación de diversas técnicas avanzas de optimización que nos permite usar Unreal Engine. A posteriori, se realizan pruebas que evalúan el rendimiento del juego en diferentes ordenadores, identificando posibles fallos de

optimización. Es una etapa crucial para garantizar que todo funciona de una manera correcta en todo tipo de sistemas.

Finalmente, una de las etapas más desafiantes, fue implementar los blueprints[1] para las interacciones con ciertos modelos y la recreación del fuego. Este último, es un reto técnico significativo debido a la necesidad de simularlo de una manera convincente.

El proyecto responde a las demandas crecientes en la industria de los videojuegos por entornos cada vez más realistas e inmersivos que a su vez consumen cada vez más recursos. Además, representa un buen inicio de porfolio para mí, permitiéndome aplicar conocimientos teóricos en un proyecto práctico y bastante desafiante.

[1] **Blueprint:** sistema de scripting visual dentro de UE5 que permite a los desarrolladores crear y modificar lógica de juego sin necesidad de escribir código.

Capítulo 2. Objetivo de este documento.

Objetivo general:

El objetivo principal es crear los elementos 3D que forman parte del videojuego Vicus Ragnarök para posteriormente implementarlos en un motor gráfico que permita su jugabilidad.

Objetivos específicos:

1. Optimizar los elementos 3D que forman parte del videojuego

2. Reducir el peso del archivo.

3. Respetar en gran medida el detalle de los modelos: A pesar de las optimizaciones, se pondrá un fuerte énfasis en mantener el detalle y la fidelidad de los modelos 3D.

4. Aumentar la compatibilidad: Se aplicarán técnicas de optimización específicas para asegurar que el videojuego pueda ser ejecutado de manera adecuada en una amplia gama de ordenadores.

Capítulo 3. Metodología.

La preproducción y producción de un entorno 3D para un videojuego se desarrolla mediante una serie de fases que abordan diferentes aspectos del proceso de creación de un videojuego. Para el modelado, se ha usado el programa de software libre llamado Blender. Los modelos tridimensionales, como ya se ha comentado en la introducción, se crean a partir de bocetos finales proporcionados por el equipo de diseño, que se encarga de la fase de concept art. Esta transición de modelos bidimensionales a tridimensionales requiere especial atención, ya que es importante mantener un número de polígonos adecuado sin perder detalles definidos por el equipo de diseño.

Seguidamente, se procede al mapeado UV y su posterior texturización. Este se realizó utilizando el mismo software que en el modelado. Además, con el objetivo de texturizar eficientemente el terreno creado donde colocamos los modelos 3D se desarrolla un sistema automatizado en Unreal Engine 5 mediante un *blueprint*. Este *blueprint* aplica texturas de manera inteligente según la altitud y pendiente del terreno, optimizando así los recursos.

Otras técnicas de optimización usadas durante el trabajo fueron la implementación de *LOD's* y de algunas herramientas que permiten realizar ajustes en la topología de los modelos, como *Simplify*, asegurando que el juego pueda ejecutarse adecuadamente en diferentes dispositivos.

Con respecto a la jugabilidad, aunque algunas funciones no están presentes, destaca la implementación de mecánicas como abrir puertas para poder acceder a ciertos espacios interiores y la simulación del fuego tanto para velas como para fogatas, haciendo uso de partículas y *shaders* de Unreal. Para estas implementaciones, se hace nuevamente uso de *blueprints*, una herramienta de programación por nodos que permite crear lógica de juego sin escribir código.

Mediante estos mismos *blueprints* se implementa un sistema de inspección de objetos, enfocado principalmente en inspeccionar las runas[2] que el jugador puede encontrar por el mapa. Además, se asigna una tecla que inicie la interacción con el objeto.

Por último, se realizan numerosas pruebas a través del editor de modo de perfilado que nos proporciona Unreal para evaluar el rendimiento del juego en función del tipo de ordenador usado y así poder detectar fallos en la optimización de este.

[2] **Runas:** artefactos esenciales que el jugador debe recolectar a lo largo del juego para realizar un ritual final que permite concluir la historia.

En conclusión, la metodología aplicada abarca desde la planificación inicial y el modelado y texturización en Blender, hasta la integración, optimización y evaluación del entorno en Unreal Engine, permitiendo un desarrollo eficiente.

Capítulo 4. Desarrollo y resultados del trabajo.

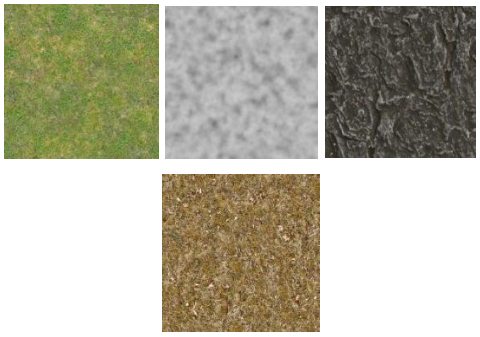
4.1 Preproducción.

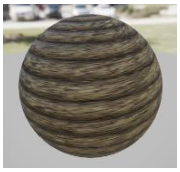
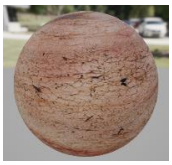
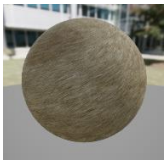
Los modelos 3D destinados a ser incluidos en un videojuego, difieren mucho de los modelos orientados al cine. En la industria cinematográfica, la renderización de un fotograma puede tardar varios minutos hasta horas o días debido a la gran cantidad de polígonos y la resolución de las texturas. Esto contrasta al hablar de videojuegos, en los que el renderizado debe realizarse en tiempo real, lo, que deja al ordenador únicamente una fracción de segundo para procesar cada fotograma (García, 2016). Esto implica la necesidad de planificación por parte del desarrollador para obtener la mayor eficiencia posible en su juego, asegurando que tanto los modelos como las texturas sean lo más detalladas posible sin perjudicar demasiado al rendimiento del ordenador.

4.1.1 Planificación de texturización y optimización.

Durante la fase de planificación de texturización y optimización, se establecen las resoluciones adecuadas para cada textura, además, una técnica crucial empleada en este proceso de texturización es el *tiling*, que proviene del término en inglés *tile* que significa ladrillo o baldosa. El *tiling* implica la repetición de una textura de manera que se genere un patrón continuo y sin aparentes interrupciones. Esta técnica resulta especialmente beneficiosa para la cobertura de superficies extensas como pueden ser la explanada y la playa en nuestro videojuego (Texturing For Games - Maintain A High Level Of Detail Without Extra Geometry, 2017).

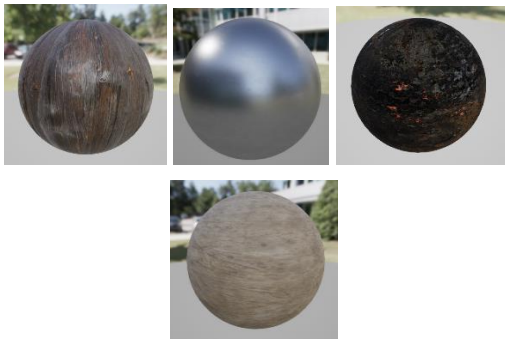
La siguiente tabla muestra las diferentes texturas que se usan en el proyecto, dependiendo del modelo e indicando su resolución:

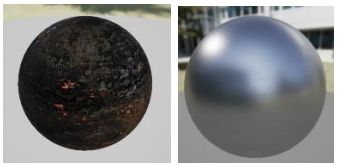
Modelo	Tipo	Materiales	Resolución
Explanada	Entorno		2048x2048
Playa	Entorno		2048x2048

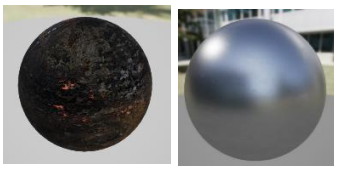
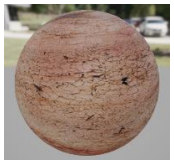
			
Muelle	Estructura	   	1024x1024
Torre	Estructura	   	1024x1024
Casa principal	Estructura	   	1024x1024
Casa secundaria	Estructura	  	1024x1024

			
Tienda	Estructura		1024x1024
Herrería	Estructura		1024x1024
Casa 1	Estructura		1024x1024
Casa 2	Estructura		1024x1024

Casa 3	Estructura		1024x1024
Runa	Objeto		1024x1024
Barco	Objeto		1024x1024
Barco hundido	Objeto		1024x1024
Jarra	Objeto		1024x1024
Carretilla	Objeto		1024x1024

Barril	Objeto		1024x1024
Pica	Objeto		1024x1024
Silla	Objeto		1024x1024
Hacha	Objeto		1024x1024
Arco	Objeto		1024x1024
Flechas	Objeto		1024x1024

Funda flechas	Objeto		1024x1024
Escudo v1	Objeto		1024x1024
Escudo v2	Objeto		1024x1024
Escudo v…	Objeto		1024x1024
Casco 1	Objeto		1024x1024
Casco 2	Objeto		1024x1024

Soplador	Objeto		1024x1024
Candelabro	Objeto		1024x1024
Mesa 1	Objeto		1024x1024
Vela	Objeto		1024x1024
Caja	Objeto		1024x1024
Cráneo ciervo	Objeto		1024x1024

4.2 Producción.

En este apartado se detallan los procesos necesarios para crear un diseño en 3D, comenzando por su modelado y escultura, todo mientras se verifica que la cantidad de polígonos usados sea adecuada para el detalle del modelo. Posteriormente, se abordan tareas como el mapeado de *UV's* y la texturización; a continuación, se emplean algunas funciones de Unreal Engine que nos permiten una mayor optimización. A lo largo de la explicación de estos procesos, se destacan los conocimientos técnicos necesarios para que el trabajo no solo se vea visualmente atractivo, sino también funcional.

Todas las figuras ilustrativas que aparecen en el presente documento han sido de realización propia, así que evitaremos repetir ese dato en cada una de las mismas.

4.2.1 Modelaje.

Para el modelado poligonal de objetos o assets, se usa Blender, un software de código abierto dedicado a la animación, modelado, simulación y renderizado en 3D. Por otro lado, el modelaje referente al entorno (como sinónimo de terreno-paisaje) donde se implementan los citados objetos es Unreal Engine 5, un motor de juego, también de código abierto, creado por Epic Games.

4.2.1.1 Modelado del entorno.

El modelado del entorno define los tamaños de los objetos creados posteriormente, por lo que es lo primero que se aborda. Para esta tarea se hizo uso de la herramienta *Landscapes* de Unreal. Esta función permite crear y modificar grandes y detallados entornos mediante un conjunto de herramientas de edición. De las múltiples opciones se han ido seleccionando las que más se ajustaban a las características del equipo de diseño. En este caso, el tamaño del terreno es de 63x63 quads y la resolución de 505x505 vértices. Es importante ajustar bien estos valores dependiendo de las necesidades con el objetivo de optimizar al máximo el rendimiento de la CPU (Landscape Technical Guide, s. f.).

El escenario se divide en dos niveles: la playa y el prado. Esta separación de niveles facilita la narrativa al distinguir dos partes de la historia, así mismo, la disposición del terreno está diseñada estratégicamente para ofrecer una guía clara al jugador de qué camino ha de tomar. Esto asegura que pueda explorar el entorno sin sentirse limitado a un camino predefinido, pero influyendo de manera inconsciente en la navegación del jugador.

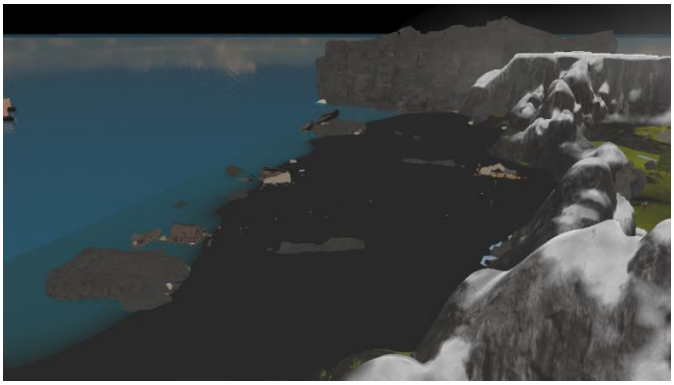


Figura 1. Playa unlit.

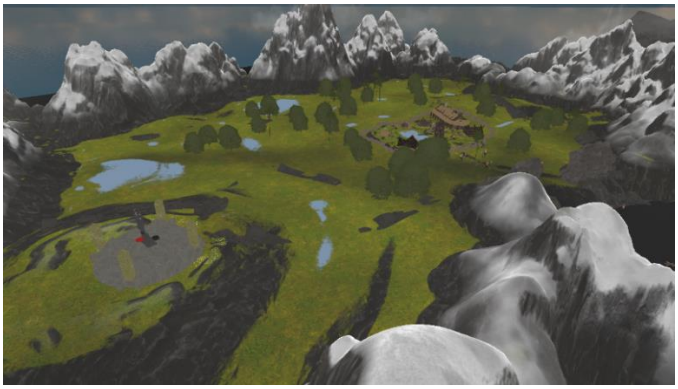


Figura 2. Prado unlit.

4.2.1.2 Modelado de estructuras-edificios.

Destacan seis estructuras que tienen mayor importancia para el jugador en el desarrollo del juego. Estas son: el muelle, la torre de vigilancia, la casa principal, la casa secundaria, la tienda y la herrería, todas ellas son accesibles, por lo que deben poder verse adecuadamente por el interior.

- El muelle es la primera estructura que el jugador encuentra al ejecutar el videojuego. Su proceso de modelado comienza con las columnas y los pilares, creando una forma básica para empezar a trabajar y permitir la colocación de las futuras paredes.

  Una posibilidad es simplemente implementar un plano con una textura de madera, lo que aporta una mayor optimización. No obstante, en este caso se decide construir mediante estructuras más pequeñas que actúan como listones modulares. Se justifica debido a que Unreal Engine ofrece herramientas de iluminación muy potentes, las cuales permiten que la luz ambiental atraviese las aberturas dejadas entre los listones, algo buscado artísticamente. Otra razón para esto es que un plano no logra una sensación de profundidad tan realista como puede alcanzar el crear un modelo de listón específico.

Esta misma justificación sirve para el caso del suelo, añadiendo que a través de este podremos ver el agua del mar.

Al contrario, el techo sí surge de un único plano al que se le aumentan los polígonos para crear la forma de las tejas y posteriormente se extruyen para darles profundidad.

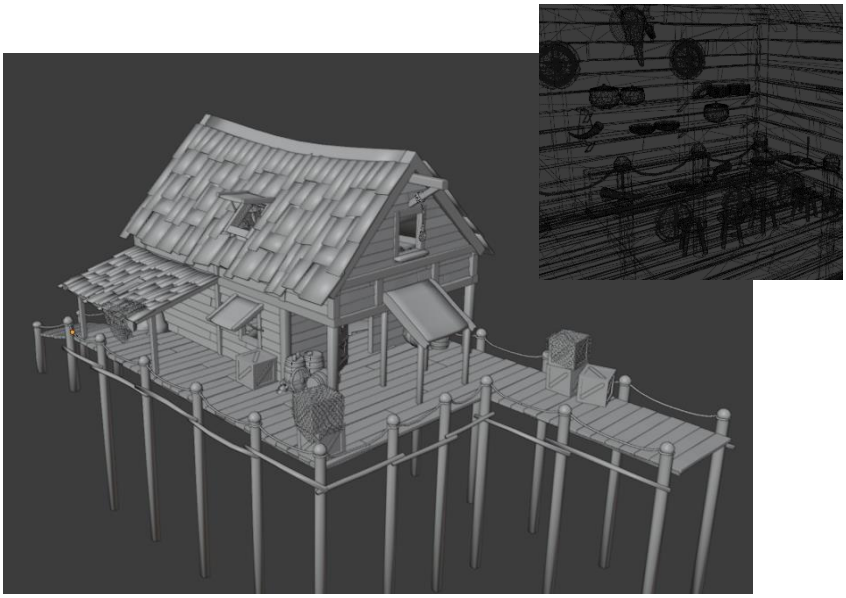


Figura 3. Nivel exterior y wireframe.

El interior presenta una distribución en dos niveles. El nivel inferior donde encontramos la taberna y uno superior al que se accede mediante las escaleras. En los dos se encuentran diferentes modelos modulares que también podemos hallar en otros puntos del mapa.

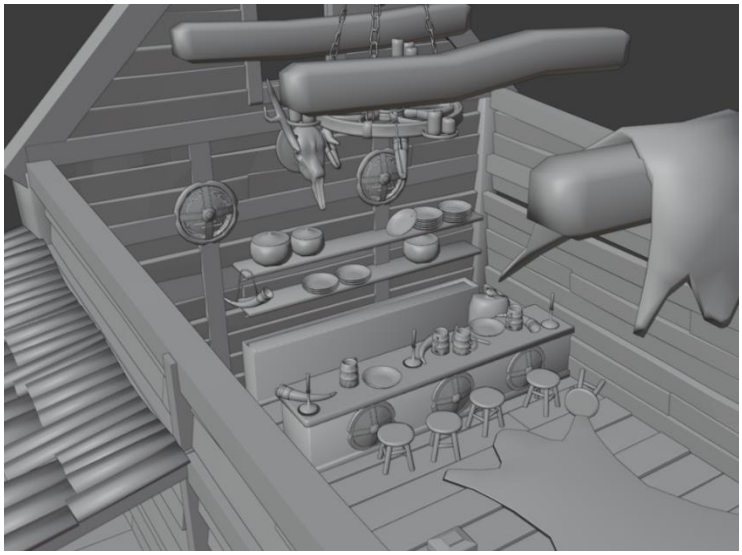


Figura 4. Nivel interior.

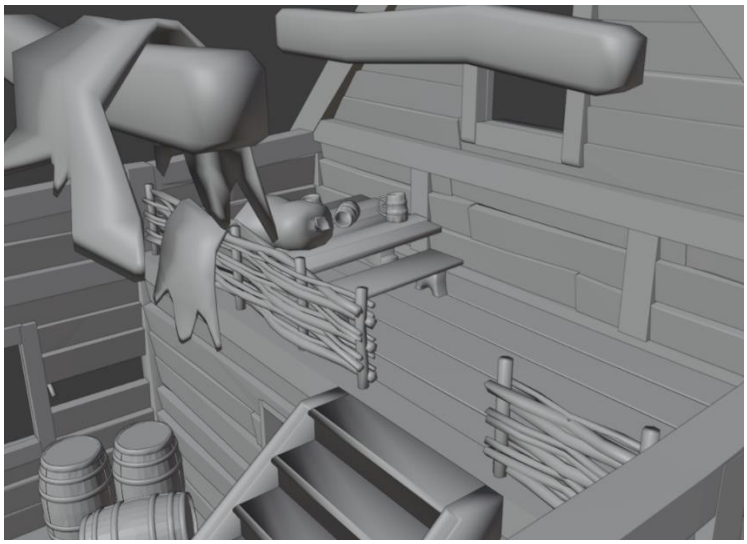


Figura 5. Nivel interior 2.

- El modelaje de la torre de vigilancia comienza por las patas, formadas por cubos escalados en su eje Z con algunas divisiones entre medias para poder darle esa forma curvada. Una herramienta muy útil que posee Blender para realizar un trabajo más eficiente en nuestro proceso de trabajo es la función de Mirror, capaz de generar una versión invertida del modelo. Su uso consiste en modelar la mitad de un objeto para después usar la transformación y así crear una versión invertida que completa el modelo.

  Los planos forman las diferentes paredes de la caseta y el techo, todos ellos extruidos mediante otra herramienta del programa llamada Solidify.

  También se añade una escalera formada por cilindros a un lateral y así poder subir y ver el interior de la estructura.

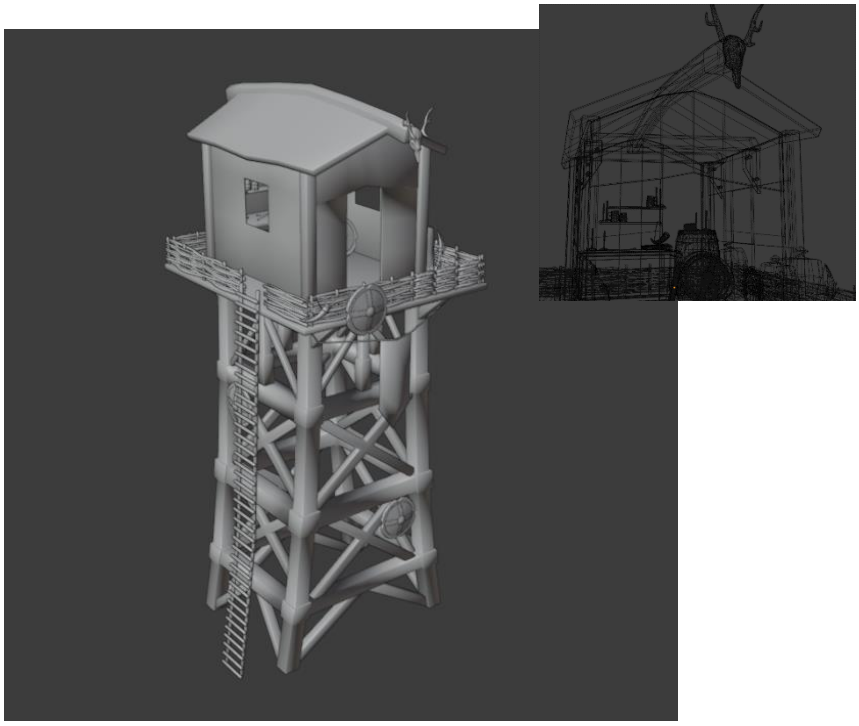


Figura 6. Torre vigilancia exterior y wireframe.

Dentro, hay una única habitación con algunos objetos a la que se añadieron vigas para aportar realismo a la estructura también desde una perspectiva interior.

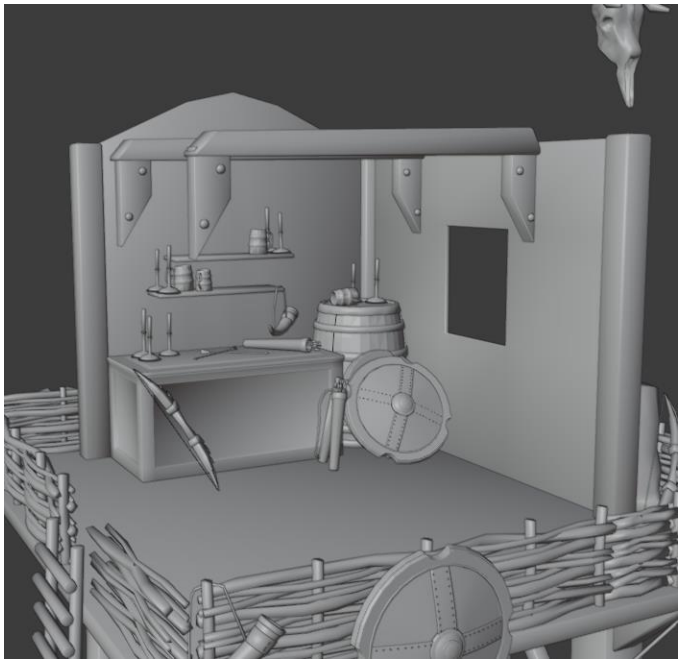


Figura 7. Torre vigilancia interior.

- La estructura de casa principal es la más grande de todas, se reutilizan tanto los pilares como las vigas de las anteriores para ser más eficientes, además de los modelos de listones creados para el muelle que se colocan formando las paredes.
  El techo superior al igual que el que rodea la casa, que en el juego se ven como paja, se forman por medio de un plano subdividido en polígonos el cual, gracias a sus aristas, elevamos o descendemos a fin de lograr un aspecto rugoso. En la parte superior de la fachada frontal se ubica una figura plana que forma dos cabezas de ciervos mirándose entre sí, la cual se crea a raíz de una herramienta llamada *Single Vert* que proporciona la posibilidad de añadir vértices sueltos. Con ella dibujamos el contorno de la mitad de la figura para posteriormente añadirle profundidad y emplear la función *Mirror* comentada anteriormente, creando así la segunda.

Figura 8. Casa grande exterior y wireframe.

Se trata de la mayor casa y por lo tanto una de las más complejas de elaborar en cuanto a cumplimentarla adecuadamente, especialmente la estructura interior, que se compone de tres espacios. El primero que descubre el jugador es el comedor del piso inferior, donde encontramos unas escaleras que nos permiten acceder a un segundo piso erigido mediante los mismos listones que forman las paredes. Por otro lado, un gran horno se alza en la casa dividiendo el primer espacio del tercero, donde vemos un altillo en el que se encontraría otra de las runas a buscar.

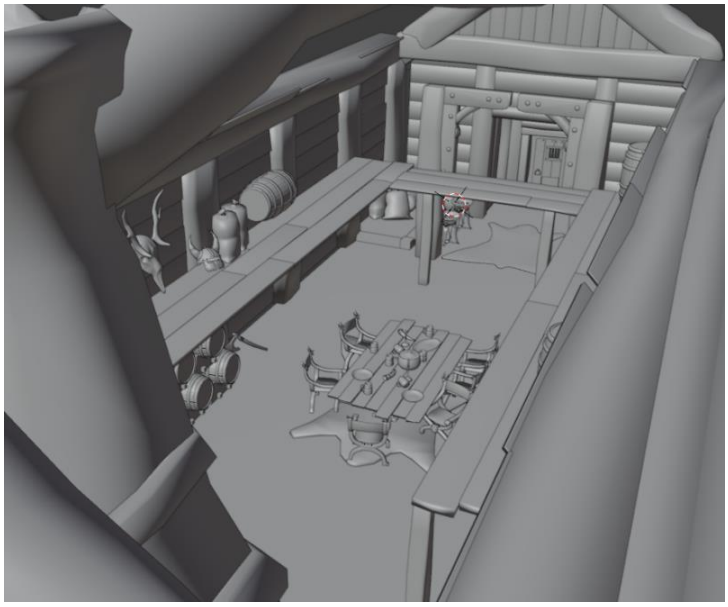


Figura 9. Casa grande interior.

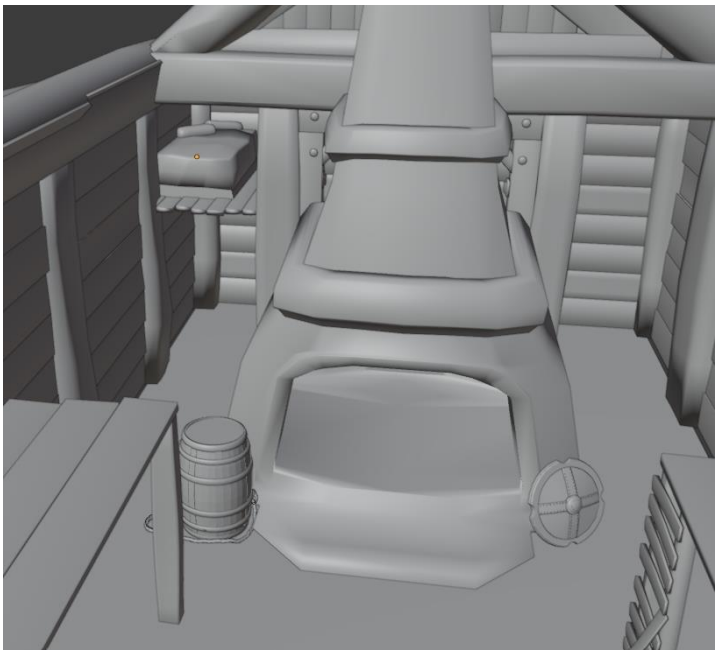


Figura 10. Casa grande interior 2.

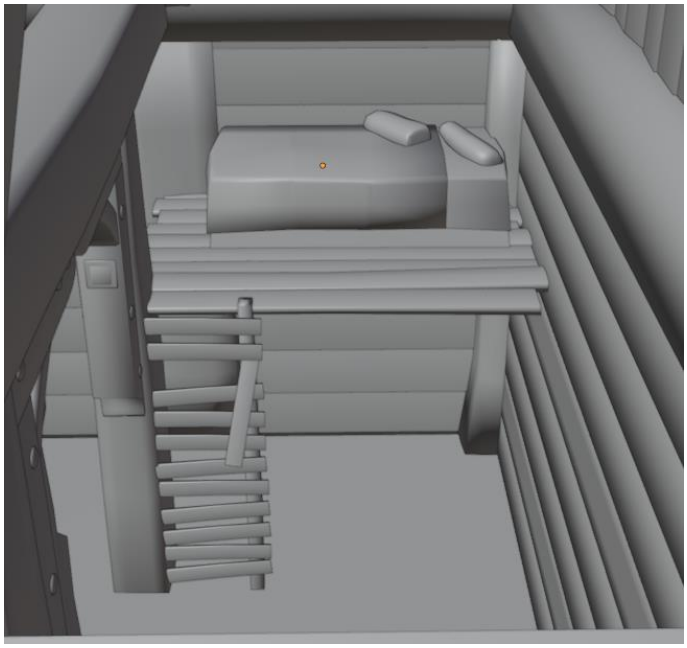


Figura 11. Casa grande interior 3.

- La casa secundaria es una de las estructuras más sencillas de levantar y como ya es habitual, nos valemos de los listones de los otros modelos para el presente.
  Destaca la cabeza de ciervo que la preside, cuyo cráneo está compuesto mediante la función de escultura del programa, cuyas herramientas nos permiten crear un modelo de una manera más libre y artística sin necesidad de manipular individualmente los vértices, aristas o caras de los diversos polígonos.

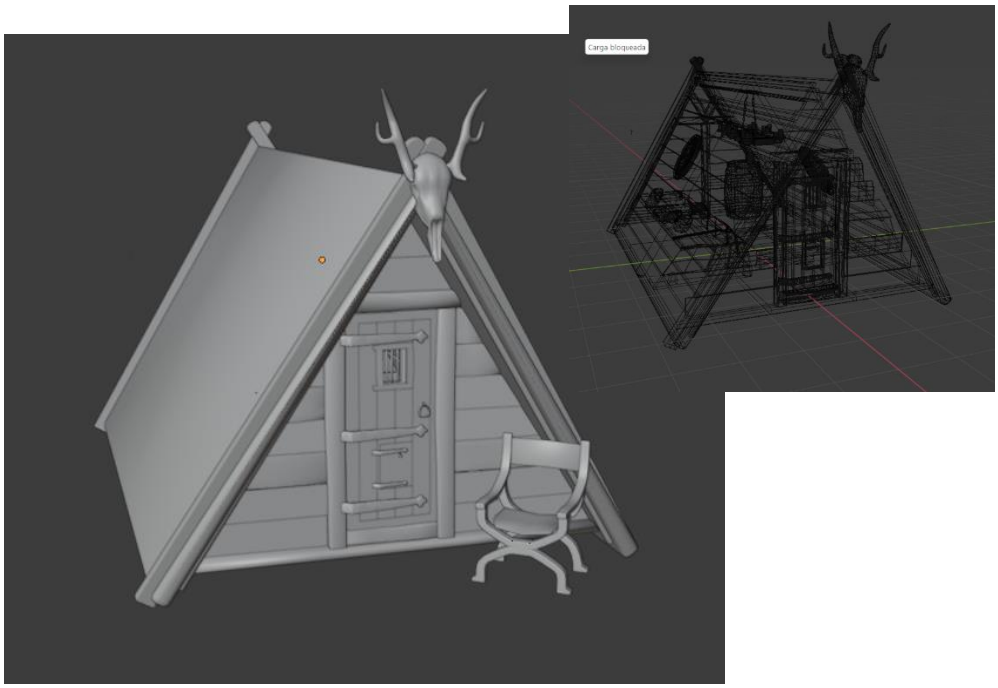


Figura 12. Casa secundaria exterior y wireframe.

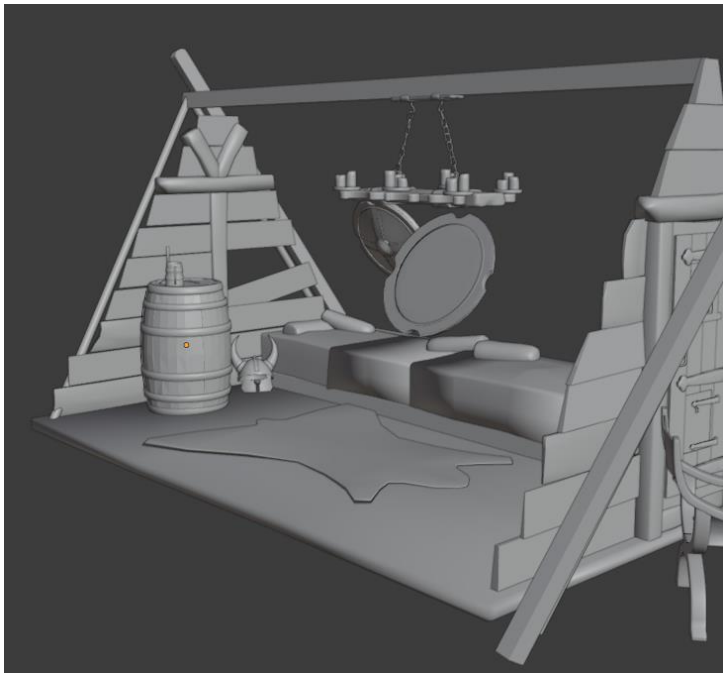


Figura 13. Casa secundaria interior.

- Respecto a la tienda, supone un reto especialmente por darle realismo a la tela que la cubre sin acabar con un número de polígonos desproporcionado. Esta se forma a partir de un plano subdividido, que se extruye desde unas determinadas aristas para generar la caída del tejido por los laterales. Seguidamente usamos *Solidify* para aportar un poco de profundidad a la tela. Por último, con objeto de darle suavidad a las irregularidades del modelo se aplica una función de Blender llamada *Smooth*, cuya tarea es suavizar las variaciones abruptas entre las caras adyacentes.

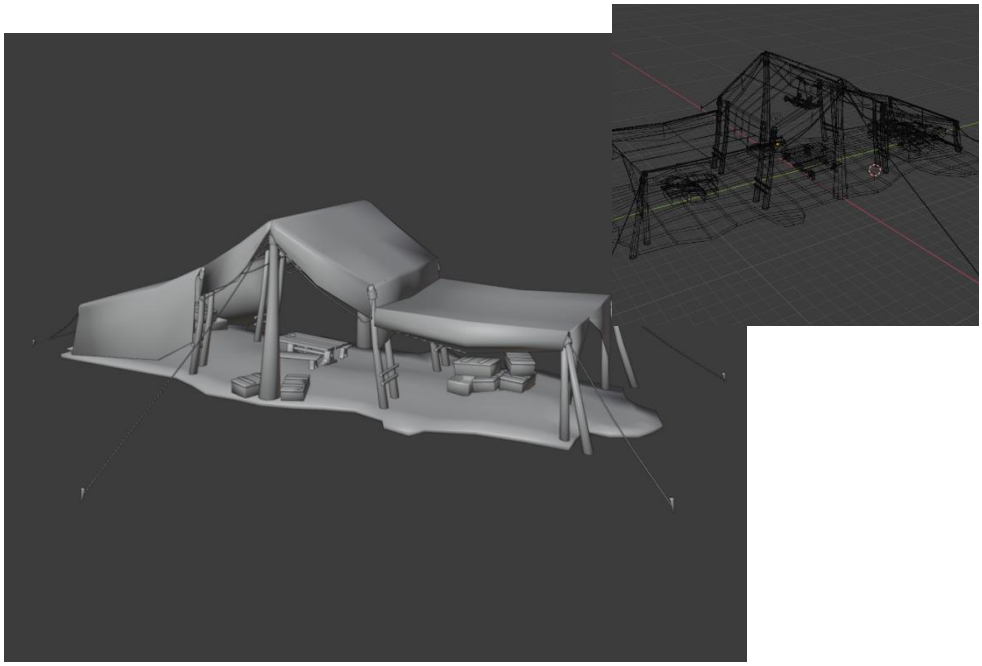


Figura 14. Tienda y wireframe.

4.2.1.3 Modelado de objetos.

Se crean un total de 26 objetos que se reutilizan en varias partes del entorno, actuando como piezas modulares.

Esta reutilización permite no solo mantener una consistencia visual, sino también optimizar el rendimiento del juego, ya que se reduce la necesidad de cargar diferentes modelos únicos.

Los objetos modelados incluyen elementos como herramientas, armas, decoraciones o barcos. Estos son algunos ejemplos:

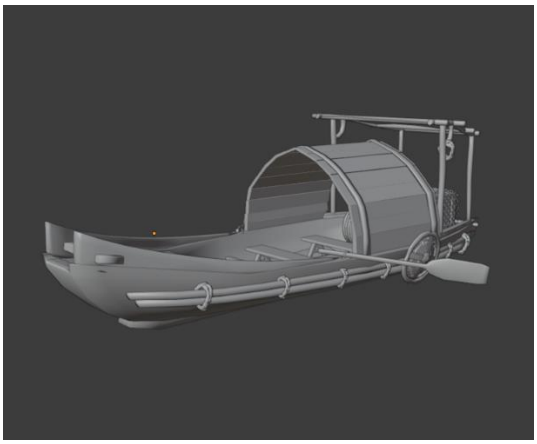


Figura 15. Barco. Figura

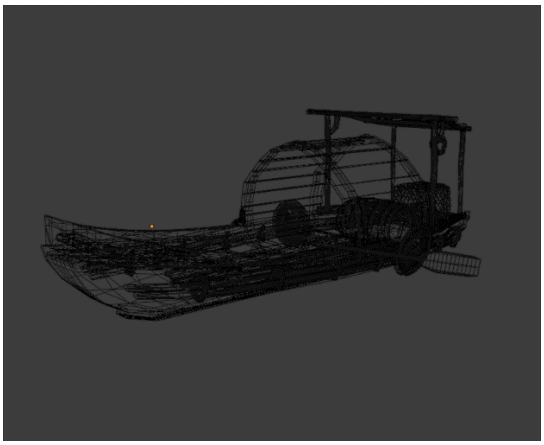


Figura 16. Barco Wireframe.

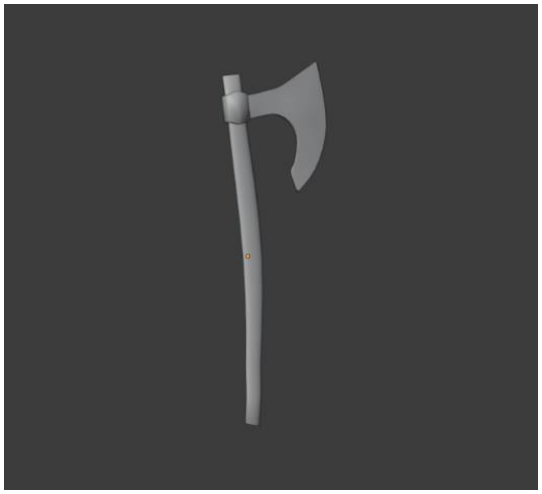


Figura 17. Hacha.

Figura 17. Hacha Wireframe.

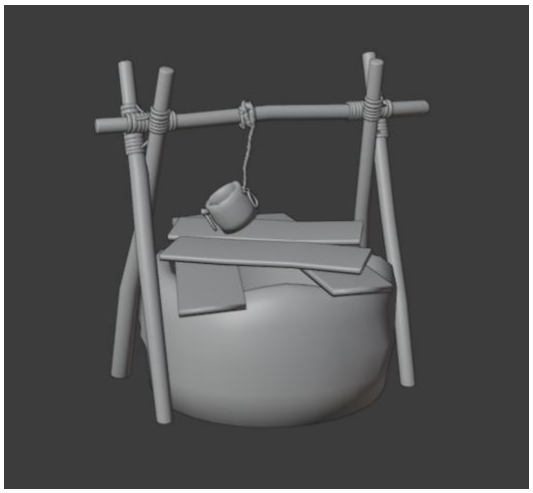


Figura 18. Pozo.

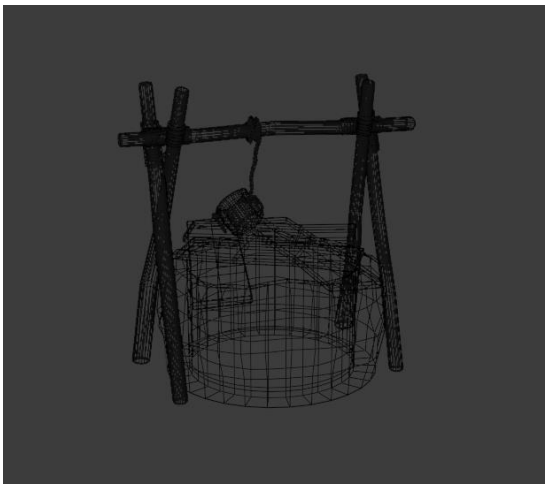


Figura 18. Pozo Wireframe.

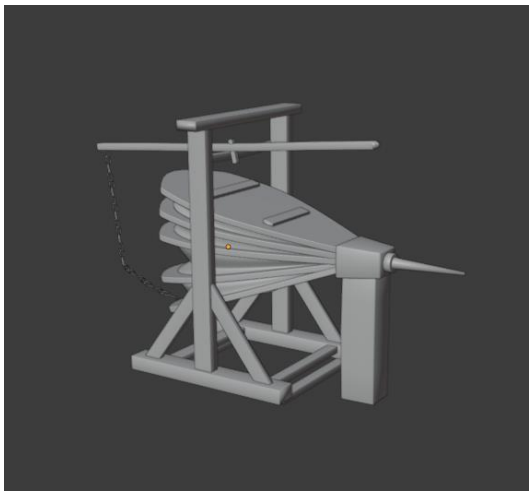


Figura 19. Soplador.

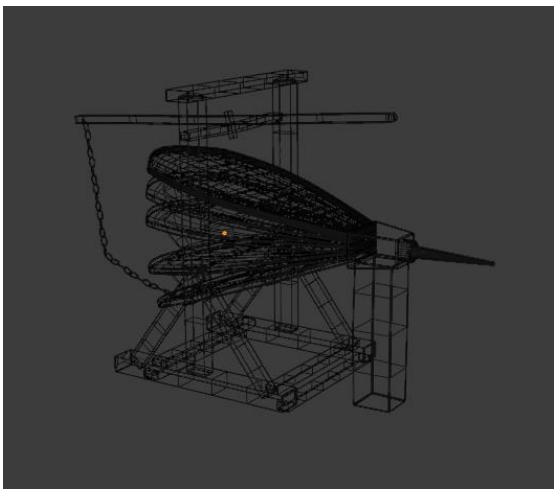


Figura 19. Soplador Wireframe.

Figura 20. Bandera.

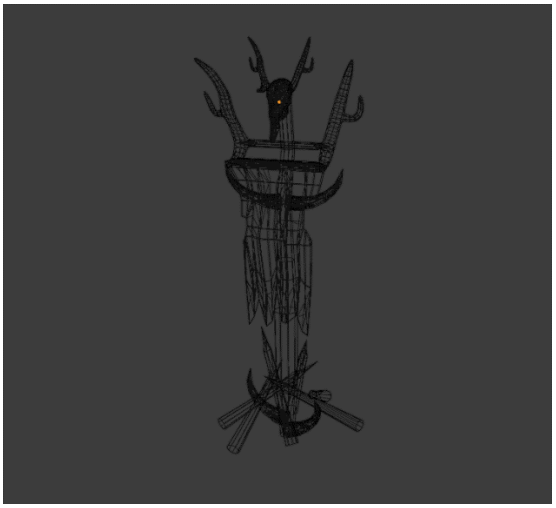


Figura 20. Bandera Wireframe.

En cuanto a las runas, se modelan con la función de esculpir, ya nombrada previamente, en concreto con ayuda de la herramienta *Draw* para darle un aspecto lo más similar a una piedra rugosa. Otro punto es la inscripción, tallada a través de la herramienta *Crease*, que junto a un hueco en el interior y añadiendo un plano que posteriormente aportará luminosidad mediante un material, se consigue que la grieta brille.

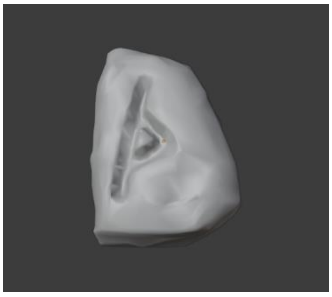


Figura 21. Runa.

4.2.1.4 Texturización del entorno.

Mediante un *blueprint* generado en Unreal Engine, se ha desarrollado un sistema de material automatizado que texturiza el entorno de manera eficiente. Este sistema aplica las texturas según la altitud y la pendiente del terreno, lo que conlleva un ahorro significativo de tiempo y recursos.

El proceso inicial consiste en la creación de funciones de material que se encargan de aspectos como la opacidad, el desvanecimiento de la profundidad, la combinación de texturas...

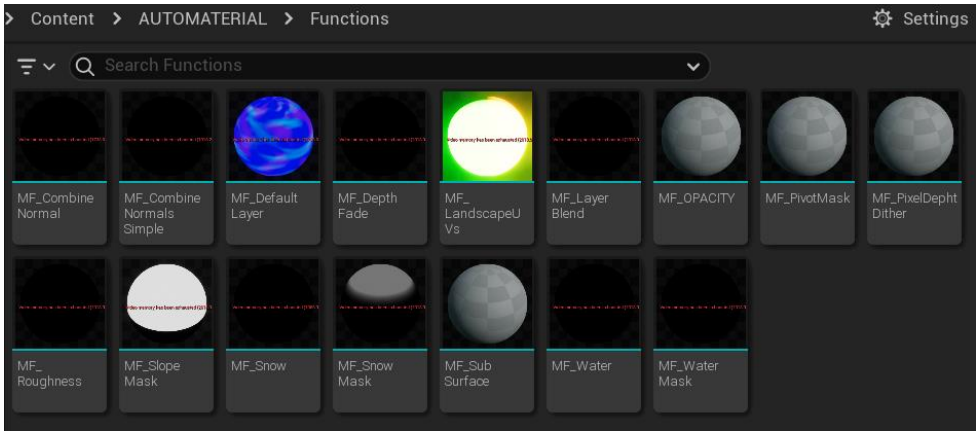


Figura 22. Carpeta automaterial.

Ya definidas estas funciones, se procede a la construcción del material automatizado en sí mismo. Este proceso implica la interconexión de las funciones previamente creadas y la configuración de ciertos parámetros como el *Slope Angle*, el ángulo de la pendiente que determina los cambios de textura, o el *Height Blend*, que mezcla las texturas según la altitud. También, la variación de texturas a gran escala depende del *Macro Variation*, y, así mismo a pequeña escala, del *Micro Variation*. Además del *Tiling Size*, que determina el tamaño del mosaico de la textura.

.

Una vez completadas, se procede a su ajuste para lograr el aspecto deseado. Este ajuste implica la modificación de los parámetros de las funciones de material, permitiendo la personalización del entorno en función de nuestras necesidades. (CG Dealers, 2023)

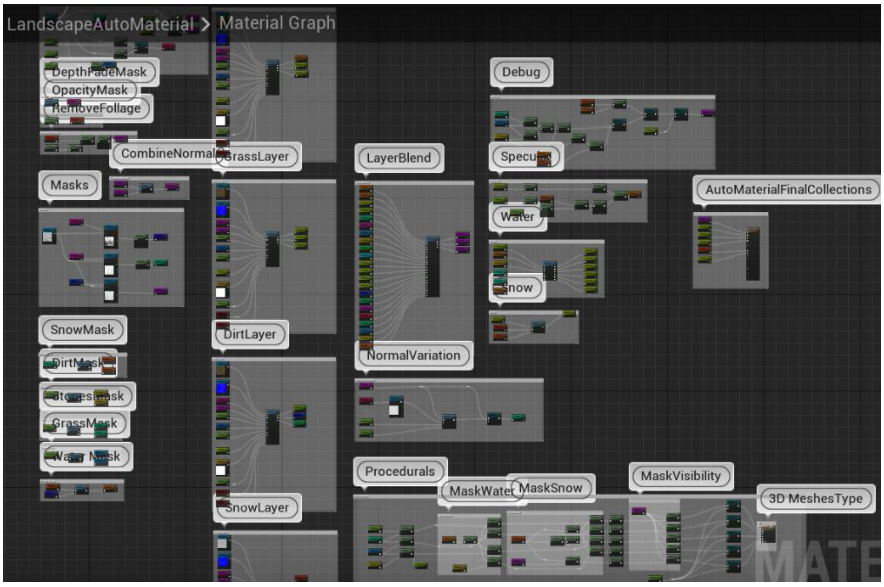


Figura 23. Blueprint automaterial.

4.2.1.5 Creación de mapas UV para los modelos.

Dado que las texturas que se usan son imágenes 2D, es necesario transformar nuestros modelos 3D a 2D para poder ubicarlas adecuadamente, esto se consigue mediante el uso de mapas UV (Mateus & Giraldo, 2012). Aunque muchos programas ofrecen soluciones automáticas para su creación, a veces es necesario un ajuste manual para obtener mejores resultados. Este proceso manual es igual para todos los objetos, pero variará en dificultad dependiendo del modelo. Consideremos la caja de madera, que es un ejemplo sencillo. Primero, identificamos que el objeto se compone de únicamente una forma básica, un cubo. En otros casos podrían ser tanto esferas como cilindros.

Es fundamental determinar las aristas para los cortes, eligiendo áreas que no sean visibles o pasen desapercibidas para evitar que se noten. En la figura 24 vemos las aristas marcadas con color rojo. Se realizan los cortes necesarios y luego, se seleccionan las caras que se van a unir y se aplica el método de proyección más adecuado según su forma.

Las piezas se disponen de manera óptima para maximizar el área destinada a las texturas. Es importante evitar escalar las partes de los mapas de manera desproporcionada para prevenir problemas de renderizado y optimizar el uso del espacio para resaltar detalles y evitar repeticiones. Se aplican principalmente tanto proyecciones de cubo y cilíndricas como automáticas, aunque estas últimas dividen el objeto en muchas piezas lo que puede llegar a causar problemas.

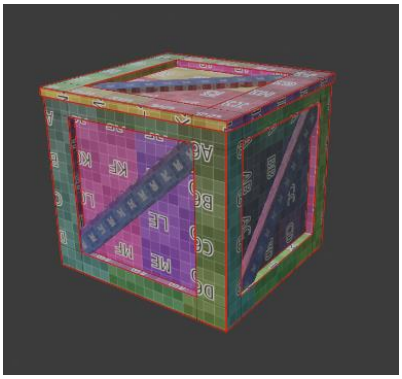
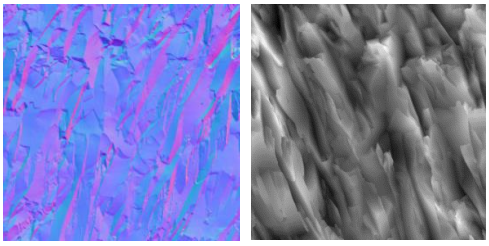


Figura 24. Aristas mapa UV.

Además, se pueden emplear mapas de normales y de especularidad mejorando el acabado final de los modelos (García, 2016).

Los mapas de normales usan colores para representar la orientación de cada punto en la superficie, mejorando la apariencia visual sin cambiar la geometría; en otras palabras: engañan al motor de render que muestra unos resultados similares a modelados con mucho más poligonaje pero sin alargar los cálculos.



Figuras 25 y 26. Normal y especularidad.

Los mapas de especularidad, mediante una escala de grises, indican las áreas de reflexión de luz en el objeto, adaptándolo a diferentes materiales y mejorando el realismo de este.

4.2.2 Integración de modelos en el entorno y aplicación de texturas.

Una vez completada las fases de modelado y despliegue de UV's, el siguiente paso es importar estos elementos a Unreal Engine. Cabe aclarar que la integración de los modelos 3D se lleva a cabo de manera constante durante toda la producción, no únicamente al final, lo que permite hacer cambios a lo largo del proceso. Para ello, se utiliza el formato FBX, elegido principalmente por su compatibilidad con el motor gráfico y su capacidad para retener información esencial del modelo, como el mapeado de UV's. En la ventana que se abre al elegir la opción de FBX seleccionamos *selected objects* lo que nos permite exportar exclusivamente las partes de la escena que tengamos seleccionadas.

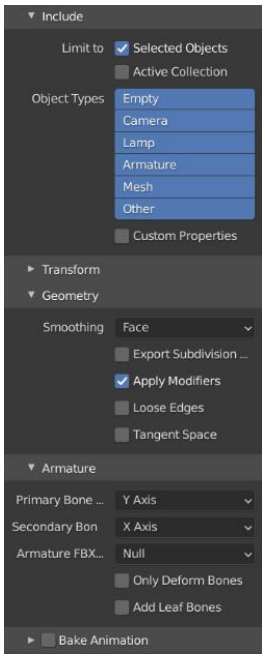


Figura 27. FBX.

Por otro lado, deseleccionamos la casilla de *add leaf bones* evitando incluir huesos a la estructura que no son necesarios para el trabajo que se realiza y, del mismo modo, quitamos la función de *bake animation*, que no es útil en este caso y así eludimos esos pesados archivos que se generan a raíz de ella. Por último, modificamos el desplegable de *smoothing* a *face*, asegurando que el suavizado de la superficie se aplica según las caras individuales del modelo.

Se verifica que no haya errores y, la siguiente etapa consiste en la aplicación de los materiales seleccionados provenientes de la función Quixel Megascans de Unreal. Es importante señalar que todas las dimensiones de las texturas descargadas son de 1024x1024, una resolución de calidad para las necesidades visuales del proyecto. Exceptuando las aplicadas a entornos, que sus resoluciones son -necesariamente- de 2048x2048 debido al gran tamaño del modelo.

4.2.3 Animaciones

Para añadir un punto extra de realismo al juego, es recomendable implementar efectos visuales como el fuego mediante el sistema de partículas Niagara que implementa Unreal Engine. Este sirve para crear efectos dinámicos y detallados, permitiendo a los desarrolladores generar y controlar sistemas de partículas complejos que le darán un aspecto más natural a nuestro juego.

El primer paso es ubicar los emisores necesarios para realizar esta tarea, suelen localizarse dentro de la carpeta predefinida de contenido inicial que genera Unreal al comenzar un proyecto. Una vez localizados se empieza a configurar cada uno de los emisores que sean necesarios como, por ejemplo, la tasa de generación de partículas, que determina con qué frecuencia se crean estas nuevas partículas de fuego, influyendo en su densidad e intensidad visual del fuego.

Otro parámetro crucial es el tamaño inicial, que permite crear llamas de diferentes tamaños y apariencias. En nuestro caso, buscamos generar tres tipos de fuego distintos: uno para un horno gigante, otro para implementar en velas y otro para lámparas. Cada uno de estos requerirá un tamaño específico según el lugar donde se vaya a colocar.

La velocidad de las partículas también juega un papel realmente importante ya que determina la velocidad a la que se mueven dentro de todo el sistema. Con este parámetro se pretende formar un fuego dinámico y natural, con un patrón de movimiento lo más realista posible. (SARKAMARI, 2023)

En cuanto a la optimización del fuego, es recomendable usar la *GPU* en lugar de la *CPU* en estos casos para mejorar el rendimiento general y no provocar bajones de *FPS*[3].

[3] **FPS:** Medida de cuántas imágenes completas se renderizan por segundo.

Figura 28. Simulación fuego.

Por otro lado, las estructuras más importantes tienen puertas con el objetivo de separar los espacios y que el jugador sienta algo más de interacción con el entorno. Para ello, se desarrolla un sistema que les permita abrirlas usando la habitual tecla “E”. Esta implementación comienza configurando un *blueprint* con función de actor que se equipa con un detector de colisiones para localizar al jugador y saber si se encuentra dentro de la zona de interacción. Al acercarse lo suficiente, el juego muestra un mensaje en pantalla indicando la tecla a presionar, y al hacerlo, se activa un evento que reproduce una animación de la puerta simulando su apertura o clausura.

Dentro del *blueprint*, se configuró el movimiento de la puerta utilizando el nodo Timeline, el cual define tanto la duración de la animación como los valores de rotación necesarios para la apertura de la puerta. Por otra parte, para alternar entre la apertura y el cierre, se empleó el nodo *Flipflop* (Kayra North, 2022).

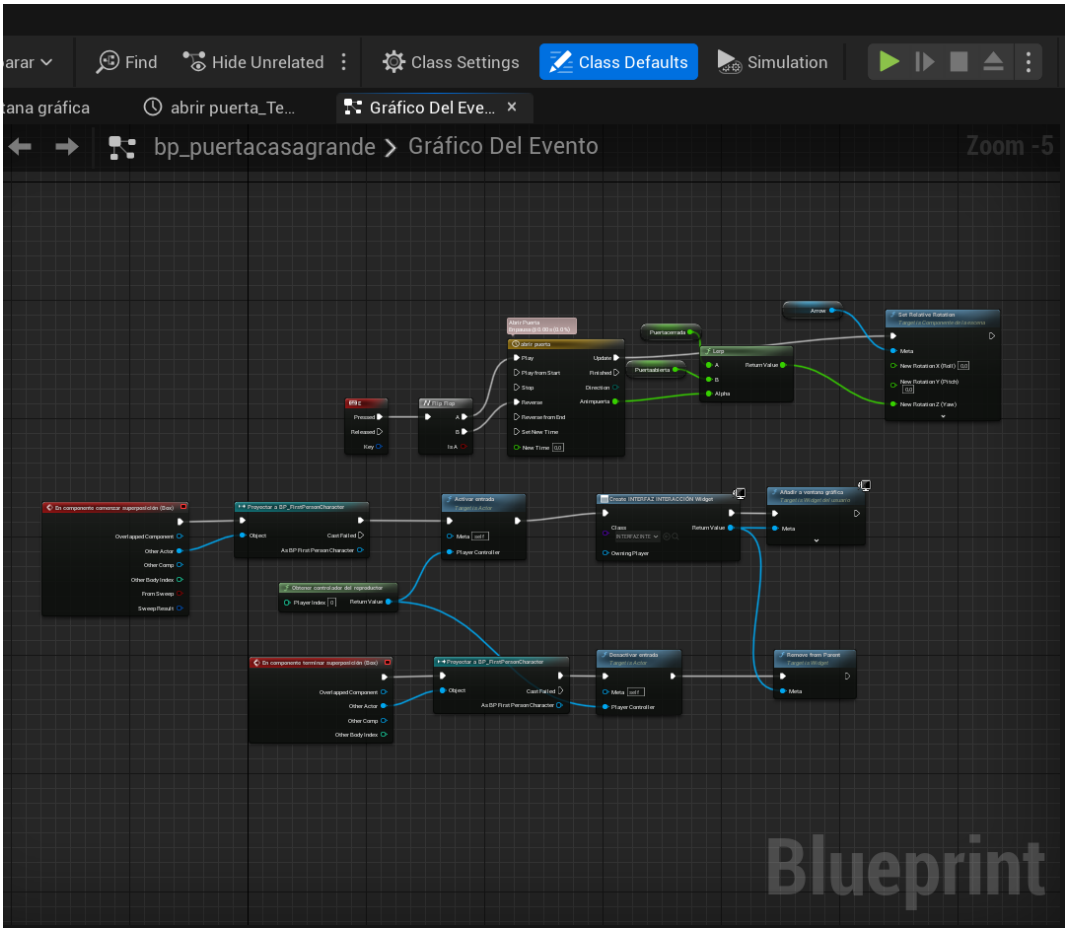


Figura 29. Blueprint puerta.

El último proceso que se implementó fue específicamente para la recolección de runas en el juego, permitiendo inspeccionar estos elementos. La metodología empleada asegura que el sistema sea adaptable a distintos objetos.

En primer lugar, se realizó una duplicación del *blueprint* del personaje, eliminando los brazos y ajustando tanto la cápsula como la cámara para minimizar las colisiones. Luego, se creó una interfaz llamada *InspectInterface*, que incluye las funciones *LookAt* e *InteractWith*, la primera se utiliza para enfocar la cámara en el objeto seleccionado, mientras que la siguiente permite la interacción directa con el objeto. Seguidamente, se desarrolló un *blueprint* de tipo Actor, configurando un *Static Mesh* como raíz y asignando la *InspectInterface* para uniformizar las interacciones.

Dentro del *blueprint* del personaje, se implementó la función *CheckLookAt*, que utiliza *LineTraceByChannel* desde la cámara para detectar objetos, se trata de un método utilizado en *Unreal Engine* para proyectar un rayo desde un punto de inicio (en este caso, la ubicación de la cámara) hacia una dirección específica y detectar colisiones con objetos que se encuentren en el

juego. Esta función se encarga de realizar un barrido visual que determina qué objeto se encuentra en la línea de visión del jugador, almacenando la referencia de dicho objeto en una variable denominada *LookAtActor*. Para la interacción, se configuraron los inputs *Inspect* y *StopInspect*, asignando la tecla 'Q' tanto para iniciar como para detener la inspección del objeto, y configurando los eventos de input del personaje para invocar las funciones respectivas.
En el *blueprint Blueprint_Inspect_Parent*, se crearon las funciones *StartInspect* y *StopInspect*, las cuales gestionan el inicio y fin del proceso de inspección. Estas funciones desactivan las físicas y colisiones del objeto, además de ajustar su posición frente a la cámara. (Nirnaeth, 2023)

4.2.4 Optimización de modelos y entorno en Unreal Engine 5.

4.2.4.1 Simplify

Utilizando la herramienta *Simplify* de Unreal Engine, se puede disminuir la cantidad de polígonos de un modelo sin perder calidad visual. Esta herramienta emplea algoritmos avanzados para reducir vértices y polígonos, conservando la forma y los detalles esenciales del modelo. Por ejemplo, en el caso de la calavera de ciervo, se logró disminuir los polígonos de X a Y. Esta reducción mejora el rendimiento del juego sin alterar la apariencia original del modelo. *Simplify* también detecta y elimina polígonos que no añaden detalle significativo, lo cual es útil en objetos con superficies planas o repetitivas. Además de reducir la complejidad geométrica, la herramienta optimiza las texturas, contribuyendo a una mejor gestión de la memoria.

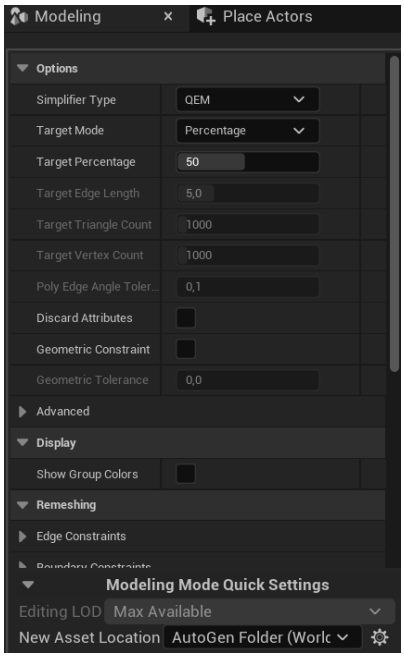


Figura 30. Simplify opciones.

Otro ejemplo es el medallón de la casa grande, donde se logró una reducción de polígonos del 70%, manteniendo el detalle y mejorando el rendimiento del juego.

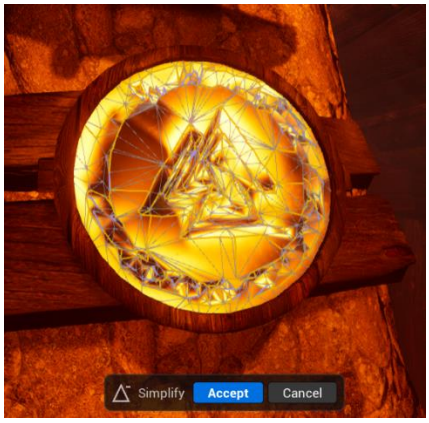


Figura 31. Simplify medallón.

4.2.4.2 LOD's. (Level of Detail)

Los *LOD's* o niveles de detalle son versiones simplificadas de los modelos 3D que permite crear Unreal Engine con el objetivo de mejorar el rendimiento. Estas versiones básicamente reducen la cantidad de detalles en los objetos que están a una distancia determinada del jugador, lo que ayuda a que el juego funcione eficientemente sin comprometer la calidad (García, 2016).

Unreal te permite crear estos *LOD's* de diferentes maneras. Por una parte, puedes crearlos automáticamente, pero el programa también permite que el usuario cree sus propias versiones personalizadas, lo que nos proporciona un mayor control.

4.2.4.3 Draw Calls.

Son un aspecto crucial en el rendimiento de un videojuego. Cada *draw call* es una solicitud que nuestro procesador gráfico recibe para poder dibujarlo en nuestras pantallas. Cuantos más haya, más trabajo tiene que hacer, lo que puede afectar negativamente al rendimiento, sobre todo en ordenadores que no sean especialmente potentes.

La técnica más común, más sencilla y de la que hacemos uso, consiste en combinar los modelos que sean similares en uno solo mediante una herramienta que nos ofrece Unreal, la fusión de actores. Esto significa que nuestra *GPU* procesará a la vez múltiples objetos, en lugar de hacerlo por separado, lo que mejorará la eficiencia de nuestro juego.

4.2.4.4 Cull distance

Establece la distancia a la cual una fuente de luz ya no es representada por el motor gráfico. Este parámetro regula el punto en el cual una luz comienza a disminuir su intensidad hasta extinguirse por completo, con el fin de mitigar el impacto de procesado en el rendimiento del juego.

Su relevancia se ve especialmente en escenarios como el del proyecto, complejos con múltiples fuentes de luz y con interiores, lo que posibilita al motor limitar el cálculo y la representación de

luces que no se encuentran dentro del campo visual del jugador. Al ajustar cuidadosamente el *cull distance* de las luces, se consigue mantener un equilibrio entre el rendimiento del juego y la calidad visual deseada, aspecto fundamental de nuestro proyecto.

4.2.5 Pruebas y evaluación.

En este apartado se busca evaluar el rendimiento del videojuego desarrollado. Esto se consigue llevando a cabo capturas de datos mediante el editor de modo de perfilado que ofrece Unreal Engine. Además, debe realizarse desde diferentes equipos informáticos para así obtener unos resultados más representativos. A continuación, se presenta la información recabada, mostrada en una tabla:

	Ordenador 1	**Ordenador 2**	**Ordenador 3**
FPS	40-45	50-60	60-90
Uso CPU	80-90%	60-75%	35-50%
Uso GPU	90-100%	70-85%	50-70%
Temperatura CPU	75-85ºC	60-70ºC	50-60ºC
Temperatura GPU	80-90ºC	65-75ºC	55-65ºC

A continuación, se reflejan los componentes que forman cada uno de los ordenadores:

- Ordenador 1 (Bajo rendimiento):
    - CPU: Intel Core i5-1035G1
    - GPU: NVIDIA MX 330
    - RAM: 8 GB DDR4
    - Almacenamiento: SSD 256 GB

- Ordenador 2 (Medio rendimiento):
    - CPU: AMD Ryzen 5 3600
    - GPU: NVIDIA GTX 1660 Super
    - RAM: 16 GB DDR4
    - Almacenamiento: SSD 512 GB

- Ordenador 3 (Alto rendimiento):
    - CPU: Intel Core i9-12900K
    - GPU: NVIDIA RTX 3080
    - RAM: 32 GB DDR5
    - Almacenamiento: SSD NVMe 1 TB

4.2.6 Resultados.

A continuación, se muestran los resultados finales del proyecto. Al acabar las fotografías también se facilitará un enlace con un vídeo en el que se muestra más detalladamente -y en movimiento- el juego.

Figura 32. Tienda renderizada.

Figura 33. Muelle exterior renderizado.

Figura 34. Muelle interior renderizado.

Figura 35. Torre exterior renderizada.

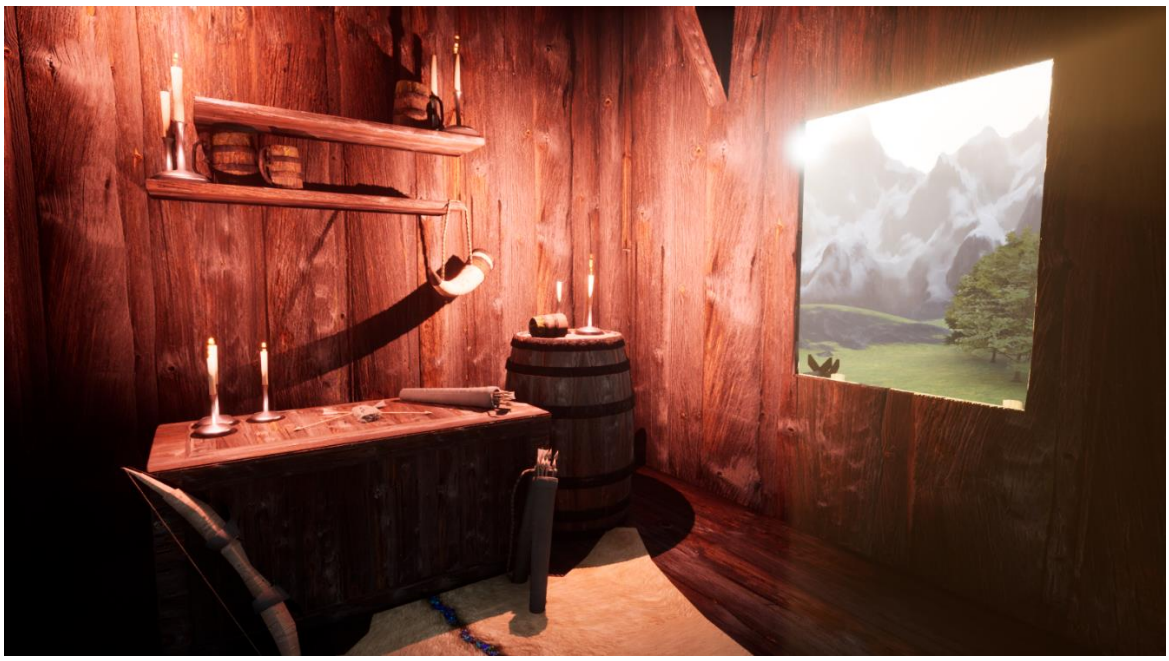


Figura 36. Torre interior renderizada.

Figura 37. Casa grande exterior renderizada.

´

Figura 38. Casa grande interior renderizada.

Figura 39. Casa pequeña exterior renderizada.

Figura 40. Casa pequeña interior renderizada.


Figuras 41 y 42. Casas secundarias renderizadas.

Figura 43. Carretilla renderizada.

Figura 44. Barril renderizado.

Figura 45. Horno renderizado.

Figura 46. Bandera renderizada.

Figura 47. Lámpara pie renderizada.

Figura 48. Vela renderizada.

Figuras 49. Entorno playa renderizado.

Figuras 50. Entorno prado renderizado.

A continuación, se facilita el enlace a un vídeo donde se ve el videojuego y sus escenarios con movimiento, tal como lo experimentará el jugador. Alternando con planos fijos para mostrar detalles de todo el trabajo realizado sin alargar la duración de este con los recorridos completos necesarios para cambiar de una ubicación a otra. Finalmente se ha podido incorporar la primera banda sonora realizada de prueba que, aunque todavía está en desarrollo, ya permite ver sus aportaciones principales.

[VIDEO ENLACE: *https://youtu.be/U1JW6Xfse1E****]***

Capítulo 5. Conclusiones y propuesta de trabajo futuro.

Los objetivos establecidos se centraban en la creación y optimización de los modelos 3D para el videojuego “Vicus Ragnarök”, así como la implementación de estos en un motor gráfico que permitiese la jugabilidad.

En cuanto a los objetivos específicos, se lograron de diferentes formas. Se llevaron a cabo algunas técnicas de optimización en los modelos haciendo uso de herramientas que incluyen los programas de Blender y Unreal como, por ejemplo, la implementación de *LOD's* o el uso de *Simplify* para la reducción de polígonos. Además, durante todo el proceso del modelaje ya se tenía en cuenta la cantidad de polígonos de los que se hace uso, priorizando así la optimización y evitando usar alguna de las herramientas ya mencionadas que pueden dar resultados más imperfectos. En segundo lugar, mediante la compresión de texturas y el uso de resoluciones adecuadas, se logró una reducción significativa del tamaño de los archivos del juego. Esto no solo facilita su posible posterior distribución, sino que también mejora los tiempos de carga. A pesar de las optimizaciones, también se buscaba en todo momento mantener el detalle y la fidelidad de los diseños proporcionados por el equipo que realizó el concept art. Se hicieron uso de técnicas de texturización como el *normal map* para garantizar el realismo sin sobrecargar el juego.

En cuanto a los resultados obtenidos, las pruebas realizadas al final nos indican que el rendimiento del juego en diferentes gamas de ordenadores es correcto, aunque con variaciones obvias en función del equipo usado. En ordenadores de gama alta, el juego mantuvo una tasa de *frames* elevada, entre 60 y 90, con un uso normal de la *CPU* y *GPU* y temperaturas adecuadas. En equipos de gama media, la tasa de *frames* era de entre 50 y 60, aunque con un uso más intensivo de los recursos del sistema. Por otra parte, en equipos de gama baja, el rendimiento se ve afectado, la tasa de *frames* es algo reducida y los recursos del dispositivo trabajan al máximo.

El método empleado posee algunas fortalezas como, por ejemplo, el uso de los *blueprints* en Unreal, facilitando la programación de estos sistemas sin necesidad de escribir código. Además, al ser modulares algunos objetos creados, se podrían crear otros niveles del mismo juego reutilizándolos. No obstante, una debilidad sería la limitación del acceso al juego para usuarios que poseen un ordenador de gama baja, pudiendo darse situaciones de menor rendimiento.

Aunque se podrían haber realizados pruebas en mayor número de diferentes dispositivos, consideramos suficientes las realizadas dado el rango relativamente amplio de la potencia de los que se han podido probar. Por necesidades de acotación del presente TFG, se ha limitado el uso de determinadas herramientas específicas a las que ofrece Unreal. Pudiéndose considerar esta circunstancia como una limitación, queda abierta la posibilidad futura de usar otras similares, aunque no parece previsible, dada su trayectoria, que Unreal deje de ser de código abierto.

En definitiva, consideramos que el proyecto ha servido como una experiencia valiosa para su autor y su futuro en el desarrollo de videojuegos, ampliando el conocimiento sobre los programas usados y afianzando una base sólida sobre la que mejorar. Si bien estamos satisfechos tanto de los aprendizajes en todo el proceso como de los resultados obtenidos, somos conscientes del siempre existente margen de mejora, considerándolo como un atractivo reto de futuro.

Capítulo 6. Bibliografía.

CG Dealers. (2023, 19 febrero). Create massive World Landscape Auto Material - Unreal Engine 5 complete tutorial [Vídeo]. YouTube. https://www.youtube.com/watch?v=rw8qDmFGsRo

García, S. M. (2016). Creación de personajes 3D para videojuegos. https://upcommons.upc.edu/handle/2117/107891

Guardia, P. F. (2016). Creación de entornos virtuales utilizando Unreal Engine 4. https://upcommons.upc.edu/handle/2117/108694

Kayra North. (2022, 4 julio). INTERACTUAR con Objetos (+ Animar Puerta) | Unreal Engine [Vídeo]. YouTube. https://www.youtube.com/watch?v=BDrzXoDQWac

Landscape Technical Guide. (s. f.). https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Landscape/TechnicalGuide/

Mateus, S. P., & Giraldo, J. E. (2012). Diseño de un Modelo 3D del Politécnico Colombiano Jaime Isaza Cadavid con Realidad Virtual. InformacióN TecnolóGica, 23(3), 95-102. https://doi.org/10.4067/s0718-07642012000300012

Nirnaeth. (2023, 18 octubre). Cómo INSPECCIONAR objetos en Unreal Engine 5 [Vídeo]. YouTube. https://www.youtube.com/watch?v=FnY3tthkQNY

Nirnaeth. (2023, 18 octubre). Cómo INSPECCIONAR objetos en Unreal Engine 5 [Vídeo]. YouTube. https://www.youtube.com/watch?v=FnY3tthkQNY

SARKAMARI. (2023, 19 agosto). #UE5 Series: Creating Fire with UNREAL Engine [Vídeo]. YouTube. https://www.youtube.com/watch?v=UQoFWJ5dwVM

Texturing for Games - Maintain a High Level of Detail without Extra Geometry. (2017, 1 junio). Pluralsight. https://www.pluralsight.com/blog/film-games/texturing-games-maintain-high-level-detail-without-extra-geometry

Manuales de referencia de los softwares utilizados:

- *Blender 4.1: https://docs.blender.org/manual/es/latest/*
- *Unreal Engine 5/4:* ***https://dev.epicgames.com/documentation/es-es/unreal-engine/unreal-engine-5-4-documentation***