Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques

Universitat de Barcelona

ANÀLISI, DISSENY I IMPLEMENTACIÓ DEL CONTROL D'UN PERSONATGE EN UNA DEMO EN UNITY3D

Álvaro Vidal Martínez

Director: Oriol Pujol

Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 16 de Juny de 2013

Agradecimientos

Ante todo quiero agradecer a todo un conjunto de personas por la ayuda y constante ánimo que me han proporcionado.

- A mi tutor y profesor Oriol Pujol por haber dirigido mi proyecto, responder todas mis dudas y darme todo el soporte que he necesitado.
- A mis amigos con los que he compartido la mayoría de horas del día a día en la universidad.
- A mi familia que siempre ha estado apoyándome y dándome ánimos.
- Y en especial a mi hermano, que siempre ha sido mi punto de apoyo.

Resumen

Durante toda mi vida he sido un gran aficionado a jugar a videojuegos. Poco a poco mi interés por la creación de estos fue aumentando mientras jugando me preguntaba cómo podía alguien crear tales obras de arte. Poder entrar en un mundo virtual hecho por ordenador con tantas similitudes con el mundo real me fascinó hasta tal punto que decidí indagar sobre el desarrollo de los videojuegos y la creación de éstos.

Finalmente me decidí por crear un juego de acción en tercera persona para ordenador. Es el tipo de videojuegos con los que estoy más familiarizado. El juego en el cual pensé se maneja a una guerrera que ha de derrotar al jefe final del nivel mediante la combinación de ataques físicos.

En este proyecto me centraré en el diseño y desarrollo completo de varios objetos del juego realizando todas las tareas de modelación y animación necesarias para la puesta a punto de éstos. También implementaré toda la programación necesaria para poder interactuar con todos los objetos creados anteriormente.

En este documento se encuentra toda la información necesaria de todo el proceso de desarrollo de este juego tanto a nivel de diseño gráfico de los objetos como a nivel de programación de estos.

Resum

Durant tota la meva vida he estat un gran aficionat a jugar a videojocs. Poc a poc el meu interès per la creació d'aquests va anar augmentant mentre jugant em preguntava com podia algú crear aquestes obres d'art. Poder entrar en un món virtual fet per ordinador amb tantes similituds amb el món real em va fascinar fins al punt que vaig decidir indagar sobre el desenvolupament dels videojocs i la creació d'aquests.

Finalment em vaig decidir per crear un joc d'acció en tercera persona per ordinador. És el tipus de videojocs amb els que estic més familiaritzat. El joc en el qual vaig pensar es maneja a una guerrera que ha de derrotar el enemic final del nivell mitjançant la combinació d'atacs físics.

En aquest projecte em centraré en el disseny i desenvolupament complet de diversos objectes del joc realitzant totes les tasques de modelació i animació necessàries per a la posada a punt d'aquests. També implementaré tota la programació necessària per poder interactuar amb tots els objectes creats anteriorment.

En aquest document es troba tota la informació necessària de tot el procés de desenvolupament d'aquest joc tant a nivell de disseny gràfic dels objectes com a nivell de programació d'aquests.

Abstract

Throughout all my life I've been a big fan of playing games. Gradually my interest in the creation of these ones was increasing while playing I wondered how anyone could create such works of art. Be possible to enter a virtual world made by computer with many similarities to the real world fascinated me so much that I decided to investigate the development of video games and their creation.

Finally I decided to create a third person action game for computer. It's the kind of game with which I am most familiar. A game in which a warrior girl has to be handled in order to defeat the final boss of the level by the combination of physical attacks.

In this project I will focus on the design and full development of various objects in the game doing all the modeling and animation tasks necessary for the development of these. I also will implement all programming needed to interact with all the objects created previously.

In this document you will find all the necessary information about the development process of this game both graphic design objects and the programming of these objects.

Índice de contenido

1 - OBJETIVOS:

1.1 – Introducción

Con este proyecto se ha querido realizar una introducción al el mundo de la creación de los videojuegos creando un "juego de acción en tercera persona" donde una guerrera deberá neutralizar al jefe final de la pantalla mediante ataques físicos para poder acceder al tesoro custodiado por éste.

En el proyecto se utilizarán las tecnologías más punteras y más utilizadas en la actualidad. Se requerirá de un motor gráfico (*engine**) y de un programa para el modelaje y animación.

Respecto al engine se utilizará uno de los más jóvenes, más potentes y de carácter gratuito llamado Unity3D.

En cuanto al programa dedicado al modelaje y animación de los elementos del juego se utilizará el Blender, una herramienta también de carácter gratuito y con un gran potencial.

1.2 – Objetivos generales:

Este proyecto completo es el conjunto de dos partes, cada parte realizada por una persona distinta y donde el lenguaje de programación que se ha establecido para este proyecto ha sido C# (*C Sharp**) pese a que también puede programarse en *JavaScript**.

El juego dispondrá de un personaje principal y una cámara controladas por el jugador mediante teclado y ratón, también dispondrá un jefe final el cual utilizará una inteligencia artificial (realizada por el otro miembro del grupo del proyecto). Respecto al escenario de juego, se dispondrá de un mapa por el cual el jugador podrá moverse y una pequeña iglesia donde se realizará el combate final. El jugador también podrá acceder a un menú principal del juego con diversas opciones y un menú de pausa *in-game**.

*Consulte el glosario de vocabulario situado en la parte final del documento

Esta documentación estará realizada mediante el uso de gran cantidad de imágenes que ayuden al entendimiento de las tecnologías usadas Blender y Unity3D. Se ha realizado de esta manera ya que también se tiene el objetivo de proporcionar a nuevos alumnos una guía que les ayude a utilizar estas tecnologías sin los grandes problemas que aparecen en el inicio de su uso. Se darán todas las pistas y pasos necesarios para poder aprovechar el uso de estas tecnologías.

1.3 – Objetivos personales:

A continuación se detallan los objetivos que se quieren conseguir al finalizar éste proyecto.

- Realización de una micro guía sobre Blender y Unity3D destinada a los alumnos de la asignatura de Ingeniería del Software que utilizarán estas tecnologías el año que viene. Es por ese motivo que ésta memoria podrá ser más densa de lo habitual ya que contendrá un gran número de explicaciones e imágenes que detallarán el proceso paso a paso de realización de los elementos que van a ser creados.

- Mediante la utilización del programa de modelaje y animación Blender se realizarán las siguientes tareas:
    - Creación de un escenario de combate donde se pueda realizar el combate final entre el jugador y el enemigo.
    - Creación de un personaje principal jugable por el usuario y el conjunto de animaciones de movimiento y ataque de éste.

- Mediante la utilización del motor gráfico Unity3D y su editor de modelaje, se realizarán las siguientes tareas:
    - Creación de un escenario general por el cual pueda moverse el jugador.

*Consulte el glosario de vocabulario situado en la parte final del documento

- Creación de un menú principal y la programación del *script* encargado de dar comportamiento a cada uno de los elementos de éste.
- Programación de los scripts que den comportamiento a la cámara del personaje principal.
- Programación de los scripts que den comportamiento al personaje principal en función de las entradas por teclado generadas por el usuario.
- Programación de todos los scripts adicionales necesarios que den comportamiento a los distintos elementos contenidos en el juego.

Nota: Todos los scripts programados serán realizados con el lenguaje de programación C# (C Sharp).

2 – INTRODUCCIÓN

2.1 – Introducción a los videojuegos

Ya hace años que existen los videojuegos, estos juegos han sido y siempre serán la una de las principales actividades de ocio de mucha gente. Existen juegos de muchos estilos diferentes, cada uno adaptado a una persona y es un ocio que puede ser disfrutado tanto en personas jóvenes como en personas mayores. En los últimos diez años éstos han ido evolucionando muy rápidamente gracias al acceso a un hardware muy potente capaz de procesar grandes volúmenes de datos. Esto también ha permitido mejorar los videojuegos en todos sus aspectos, tanto a nivel visual (siendo capaces de lucir unos gráficos sublimes), como a nivel de inteligencia artificial (capaz de recrear el comportamiento y la actuación de personas o el entorno).

Cada vez han ido apareciendo más desarrolladoras de videojuegos viendo el potencial que se escondía detrás de éstos. A día de hoy, ésta industria mueve muchísimos millones de dólares, más incluso que la propia industria cinematográfica.

Por todo lo anterior citado y muchas razones más se puede confirmar que sin duda alguna los videojuegos son un arte, el séptimo arte.

2.2 – Tecnologías utilizadas

En el momento de escoger las tecnologías que se iban a utilizar en el proyecto, se barajaron varias posibilidades para finalmente decantarse por Unity3D y Blender como engine y programa de modelaje y animación 3D respectivamente. A continuación se analizarán cada una de estas dos tecnologías y se explicarán los motivos por los cuales fueron escogidas.

-2.2.1: Motor gráfico: UNITY 3D

Existe gran diversidad de engines utilizados para la creación de videojuegos. Primero están los engines privados creados por las propias empresas de videojuegos al que suelen tener acceso únicamente los miembros de la propia desarrolladora, tales como GoldSrc (creadores de Counter Strike), PhyreEngine (de la empresa Sony), JadeEngine (de la desarrolladora Ubisoft) entre muchos otros.

Por el contrario, existen los públicos tales como el Unity3D, Unreal Engine3, CryEngine 3 o Havock entre otros. Estos últimos engines son compatibles a nivel multiplataforma, son muy potentes y tienen una gran comunidad de desarrolladores debido a que son gratuitos siempre y cuando se utilizan con fines estudiantiles, de lo contrario se deberá que adquirir una licencia.

Como puede verse hay una gran cantidad de engines, pero para el proyecto que se está realizando, va a utilizarse uno de los de carácter gratuito ya que son los únicos a los que se puede tener acceso.

Después de una gran indagación acerca del engine que se querría usar, se decidió escoger el Unity3D a pesar de que CryEngine3 o UnrealEngine3 en especial disponen de una carrera videojuegil mucho mayor de la que ha tenido Unity3D. Pero a pesar poco tiempo que lleva Unity3D en el mercado, debido a la gran aceptación y la enorme comunidad de desarrollo que ha obtenido se he decidido finalmente utilizar este engine para el juego.

Unity3D ha sido usado en varios juegos famosos, tales como Dead Trigger 1 & 2, Bad Piggies, Temple Run 2, Three Kingdoms Online entre muchos otros.

Cabe decir que este engine está internamente programado, como en la mayoría de casos, en C++, un lenguaje que permite un gran control sobre los recursos utilizados por sistema y que en el mundo de los videojuegos es un punto a tener en cuenta debido a la gran cantidad de datos por segundo que se están procesando durante la ejecución de éste.

Este motor permite ser programado en C# (C Sharp) o JavaScript, a diferencia de sus principales competidores (CryEngine3 o UnrealEngine3) que utilizan C++. C# es un

lenguaje que es muy parecido en todos los aspectos al lenguaje de programación Java pero que mantiene unas ciertas similitudes al tan antiguo lenguaje de programación C. Se ha escogido C# porque es un lenguaje más rápido que JavaScript y en los juegos es muy importante utilizar lenguajes rápidos puesto que la cantidad de datos que se están procesando cada segundo es mucho mayor que en la mayoría de programas usados habitualmente.

Unity3D se empezó a comercializar en el año 2009 en su primera versión gratuita y desde ese entonces ha participado en el desarrollo de videojuegos en Windows, Linux, OS X y varias de las plataformas de consolas de sobremesa.

Además Unity3D tiene una gran fama en ser muy amigable a la hora de interactuar con las importaciones de modelos y animaciones. Dispone de gran cantidad de formatos de importación y por ello cubre prácticamente todos los formatos de importación realizadas por cualquier programa de modelaje. Permite importar prácticamente cualquier tipo de objeto sin dar ningún problema.

Es por ello que por todas estas razones explicadas anteriormente se ha decidido utilizar Unity3D, un engine muy potente, con un lenguaje de programación amigable y una compatibilidad con programas de modelaje externos sumamente espectacular.

-2.2.2: Programa de modelaje y animación:

En cuanto al programa de modelaje y desarrollo del los objetos contenidos en el juego cabe decir que hay también varios programas de modelaje, pero hay dos que siempre tienen disputas entre ellos: Blender y 3DStudioMax. Aunque la mayor diferencia de éstos recae en el precio ya que el primero es completamente gratuito y el segundo es de pago.

Para este proyecto se decidió realizar el modelaje y animación de los objetos del juego con Blender debido a su gratuidad y también quería conocerse el verdadero potencial de éste y que lo diferenciaba del tan ansiado 3DStudioMax, un motor que sobrepasa en gran medida

a Blender en fama y es sobre todo muy utilizado en los comienzos de cualquier diseñador gráfico de juegos.

Blender es un programa que permite modelar y a continuación animar el objeto en cuestión y también permite la composición y *renderización** de escenas 3D de gran complejidad. Tiene dos grandes ventajas y una gran desventaja. Respecto a las ventajas, la primera ya anteriormente citada, que es software libre (y por ello posee una gran comunidad de diseñadores que comparten y colaboran) y la segunda que permite programar en el lenguaje de programación *python** con una amplia gama de script en constante desarrollo. En cuanto a la gran desventaja es que respecto a sus competidores, 3DStudioMax, Maya...tiene una mayor complejidad de uso ya que no es tan intuitivo como estos dos últimos y requiere un aprendizaje mayor.

Blender posee un engine propio que permite ejecutar videojuegos pero que no va a ser utilizado ya se dispone de Unity3D, un engine mucho más potente.

Por todas las razones citadas anteriormente, por el gran apoyo al software libre ante todo y por la gran comunidad de diseñadores de la que dispone, se ha decidido utilizar el Blender.

*Consulte el glosario de vocabulario situado en la parte final del documento

3 – ANÁLISIS

3.1 – Escenario principal:

Éste escenario es un escenario creado por el propio engine Unity3D en el cual van a ser colocados todos los elementos del juego y por el cual el personaje podrá moverse libremente. Inicialmente se optó por un diseño de un mapa abierto donde se distinguían varias zonas. En primer lugar se pensó en que el personaje principal aparecería en la parte más al sur del mapa y debería cruzar éste enfrentándose a los distintos enemigos que yacían en el camino. Se pensó también en hacer poblados donde interactuar con gente y también la introducción de alguna mazmorra que permitiera dar más jugabilidad al juego.

En la imagen 1 puede verse el prototipo de escenario pensado. La zona sur donde el personaje aparece, un conjunto de poblados de orcos y esqueletos al este y al oeste respectivamente y finalmente en el norte el castillo donde yacerá el enemigo final.

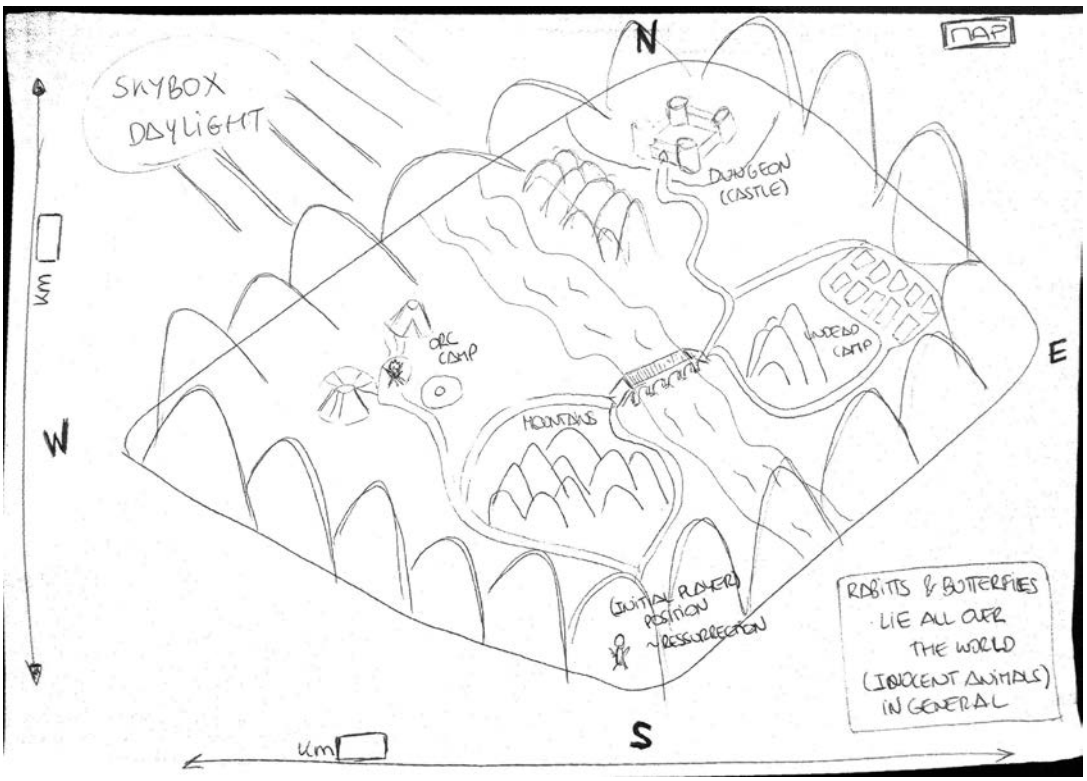


Imagen 1: Prototipo de escenario pensado

3.2 – Menú principal:

Desde un primer momento se pensó en que este menú debería de disponer 4 opciones seleccionables. En primer lugar el "Inicio de partida" que daría lugar a una partida nueva. En segundo lugar un "Continuar partida" que nos permitiría seguir con el juego desde un punto guardado anteriormente. En tercer lugar "Opciones", que permitiría modificar las opciones de vídeo y sonido deseadas. Y finalmente en cuarto lugar una opción de "Salir" que permitiría salir de la aplicación.

3.3 – Menú de pausa:

Éste menú será accesible mientras el jugador está en una partida. El jugador dispondrá de la opción de abrir el menú apretando la tecla 

del teclado y esto detendrá el juego y dará al jugador algunas opciones de elección tales como modificar gráficos actuales del juego, cambio del volumen del sonido, guardar partida, cargar partida o salir del juego.

3.4 – Escenario principal de combate (Castillo):

En este escenario se desarrollará todo el combate entre el personaje y el jefe final del juego. Estará situado al lado opuesto del punto de origen del personaje, al norte del mapa. Dentro de éste yacerá el jefe final con el cual el personaje principal se deberá enfrentar para poder optar a pasarse el juego. El castillo estará formado por una iglesia dentro de la cual estará ubicado el jefe final y ésta estará envuelta de unas murallas que protegerán el castillo de los invasores externos. Puede verse en la imagen 2 el prototipo de diseño de media iglesia (debido a que la otra mitad es simétrica) y las medidas de ésta.

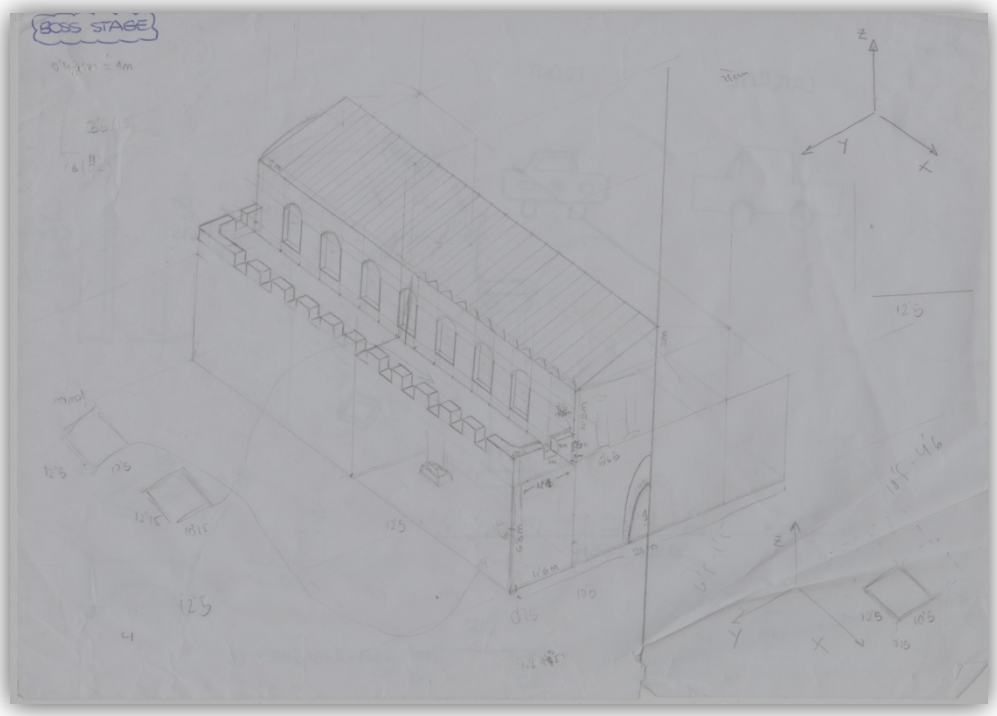


Imagen 2: Prototipo de escenario de combate pensado (Iglesia)

3.5 – Personaje principal

El personaje principal va a ser una chica guerrera la cual dispondrá de ataques físicos mediante el uso de una espada y también de un escudo que permitirá la defensa en contra de ataques enemigos. Estará ambientada en un mundo de fantasía medieval, y por ello vestirá un traje acorde con esta época. Éste personaje dispondrá también de ataques que consumirán magia, además de distintas habilidades que podrán ser mejoradas y subidas de nivel en función del uso que el jugador le dé a estos. Mientras se esté realizando una partida se visualizará la barra de vida, la barra de maná y un marcador con el nivel actual del jugador. El personaje será único y no se optará por la personalización de las características del mismo pero sí que dispondrá de la capacidad del cambio de armadura ya que al eliminar enemigos, éstos pueden dejar caer ciertos objetos, algunos de los cuales podrían ser armaduras. En la imagen 3 puede visualizarse un prototipo de una guerrera parecida al pensado extraída del juego Skyrim.

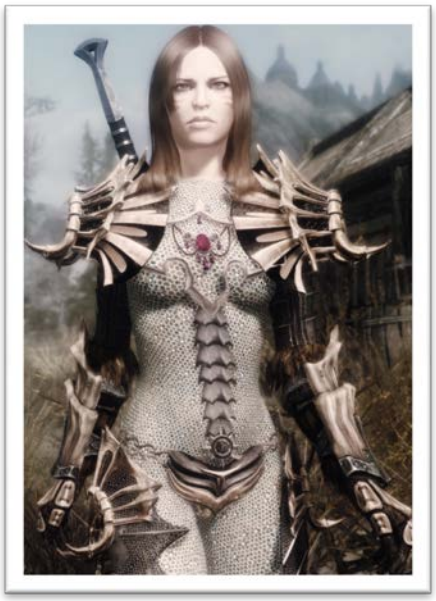


Imagen 3: Prototipo de personaje principal pensado. Imagen extraída del juego Skyrim

Los ataques del personaje principal estarán definidos por un conjunto de combinaciones. Utilizará la espada con una mano para realizar los ataques con un nivel de daño normal y las dos manos para realizar un ataque final de combo que permitirá un ataque con un nivel de daño alto. En la imagen 4 puede verse la lista de combinaciones de ataques pensada con su respectiva combinación de pulsaciones del mouse.

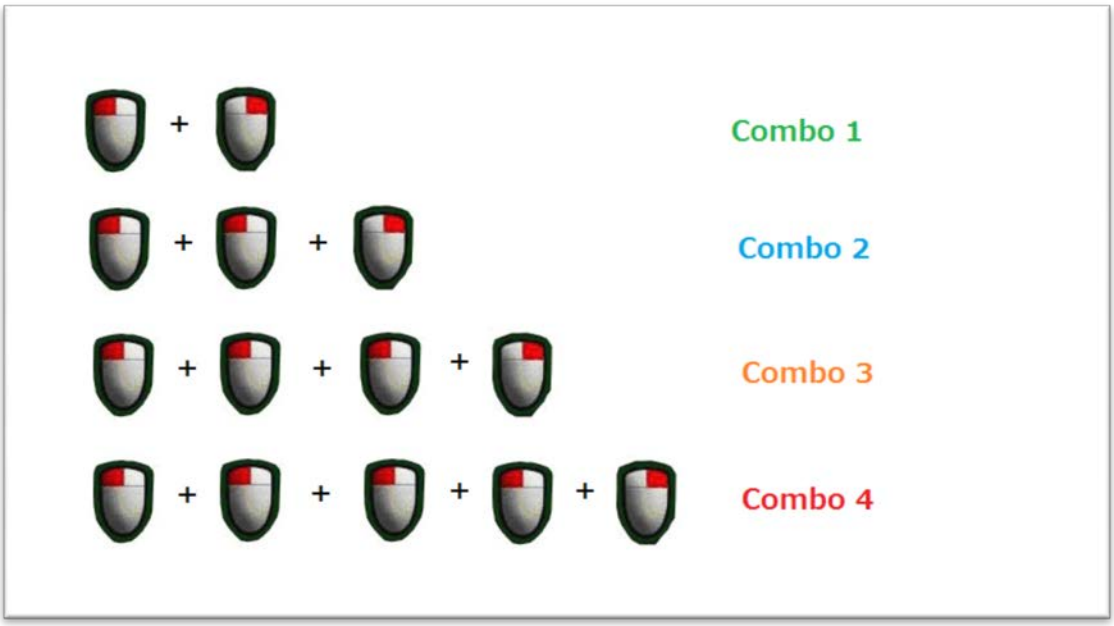


Imagen 4: Prototipo de la lista de combinaciones de ataques del personaje

El personaje principal dispondrá también de varios movimientos básicos tales como caminar, correr, saltar, moverse lateralmente o nadar.

En la imagen 5 puede verse un boceto con parte de los distintos movimientos y ataques de los que dispondrá el personaje principal.

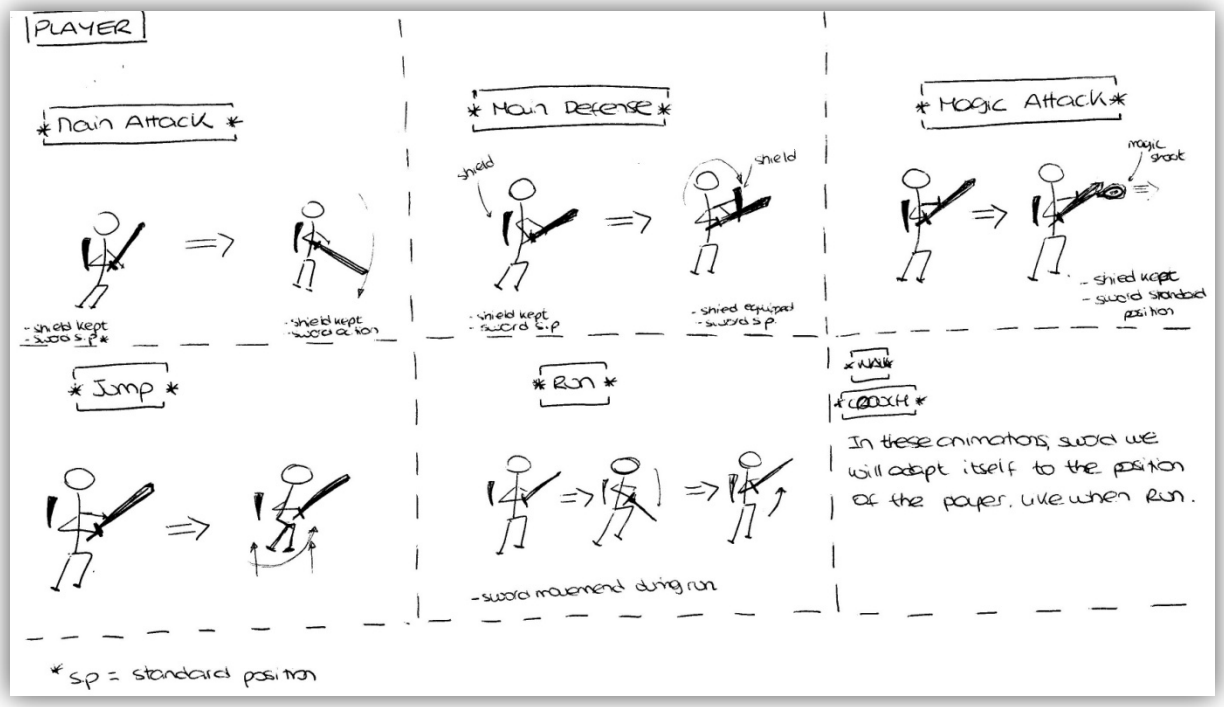


Imagen 5: Prototipo de las animaciones movimientos y de ataques del personaje

3.6 – Cámara personaje principal

La cámara que va a ser utilizada con el personaje principal va a ser una cámara en tercera persona. Ésta en su modo por defecto será manual, controlada por el jugador pero dispondrá también de una cámara automática que será controlada por el propio engine si así lo desea el jugador. La cámara en la que se ha pensado está presente en muchos juegos del estilo de juego al que pertenece éste, es decir un "*action third person role playgame"* o más comúnmente llamado *ActionRPG**. La cámara permanece a una distancia

*Consulte el glosario de vocabulario situado en la parte final del documento

del personaje y con una altura superior a éste para poder visualizarlo a él y a su entorno más cercano. En la imagen 6 puede visualizarse el prototipo deseado de cámara. Posicionada detrás del personaje y a una cierta distancia.

Imagen 6: Prototipo de cámara del personaje. Imagen extraída del juego Skyrim

3.7– Enemigos

Los enemigos estarán repartidos por el mapa, pudiéndose encontrar éstos por el camino hacia un nuevo campamento o en los propios campamentos. Los principales enemigos de nivel inferior serán orcos y esqueletos. Éstos dispondrán también de un nivel que irá variando en función de la zona y del nivel del personaje y también dispondrán de un arma con la que realizarán los ataques primarios. No dispondrán ningún tipo de ataque a distancia. Además no llevaran consigo ningún tipo de defensa ya que no se contemplará esa acción para éstos. Puede verse la lista de acciones a continuación. En la imagen 7 puede verse los distintos ataques que poseerán los orcos y esqueletos.

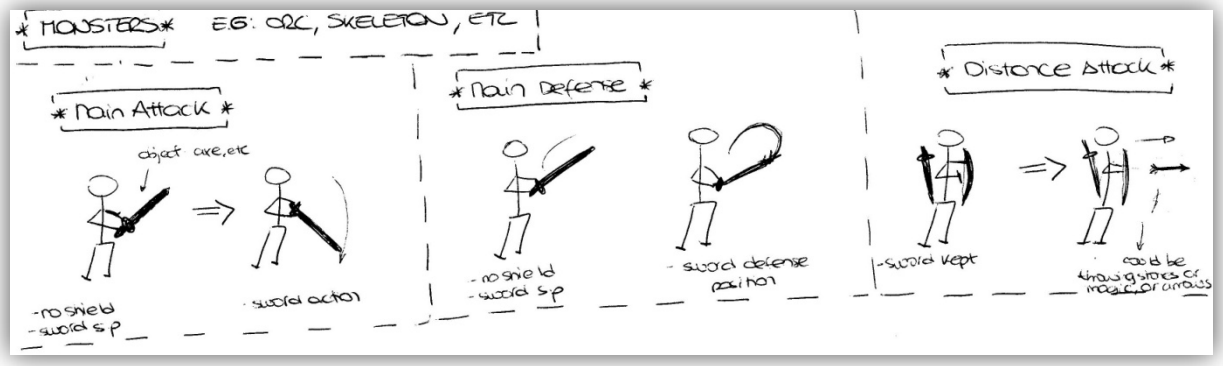


Imagen 7: Prototipo de las animaciones movimientos y de ataques de los enemigos generales

Además, dispondrán de una recompensa que, una vez el jugador elimine al monstruo, podrá recogerse. El arma que los enemigos lleven consigo formará parte de la recompensa que dejarán caer, aparte de dinero y/o la armadura. La experiencia que otorgarán al jugador al ser eliminados variará en función del nivel de estos, así como la cantidad de vida y el índice de dureza frente a los ataques del jugador. Puede verse en la imagen 8 y la imagen 9 un prototipo del orco y un esqueleto respectivamente, buscados para este juego. Estas imágenes han sido extraídas del juego Skyrim.

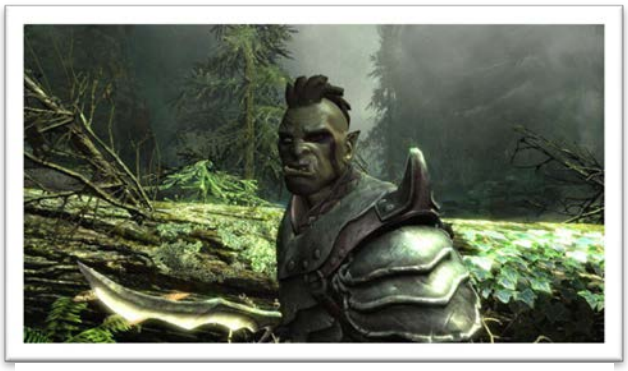


Imagen 8: Prototipo de orco. Imagen obtenida del juego Skyrim

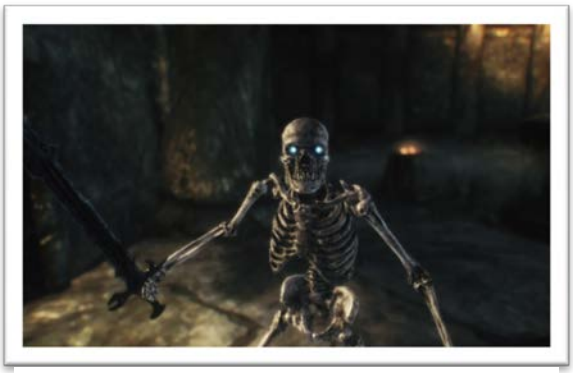


Imagen 9: Prototipo de esqueleto. Imagen obtenida del juego Skyrim

3.8 – Jefe final (*Boss**)

El boss será el enemigo que permitirá al jugador, una vez haya sido derrotado, pasarse el juego. Éste dispondrá de ataques físicos con un bajo nivel de daño con los que atacará al jugador siempre y cuando éste esté dentro del radio de afectación de éstos. También dispondrá de tres ataques fuertes, siendo uno de ellos mortal para el jugador. El primero de ellos será una llama lanzada desde el propio boss hacia el personaje principal. El segundo ataque será un campo de lava que se originará en el suelo, en la posición que esté el personaje principal y con un área suficientemente grande como para que cuando el boss vaya a lanzar el ataque el jugador deba apartarse de la zona corriendo. Finalmente, como tercer y último ataque dispondrá de un pequeño meteorito mortal que lanzará al jugador. En la imagen 10 pueden verse los los distintos ataques y ataques especiales del enemigo final.

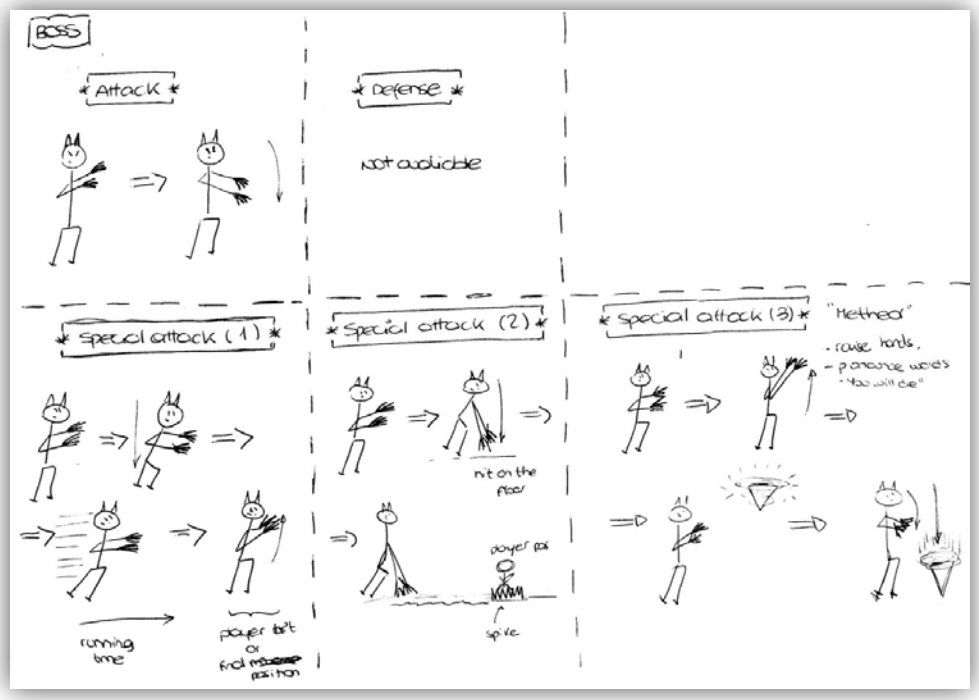


Imagen 10: Prototipo de las animaciones movimientos y de ataques del jefe final

*Consulte el glosario de vocabulario situado en la parte final del documento

Respecto a las características físicas del boss, éste será de un tamaño de aproximadamente dos veces y medio el personaje principal y será corpulento. Con piernas de humano y cara de demonio con los brazos llenos de llamas que le otorgarán poder para efectuar los distintos ataques. En la imagen 11 puede verse el prototipo de boss en el que se ha pensado. Ésta imagen pertenece a un enemigo final del juego Devil May Cry 4.

Imagen 11: Prototipo de cámara del jefe final. Imagen extraída del juego Devil May Cry 4

3.9 – Desarrollo del juego (*Gameplay**)

El jugador aparecerá en la parte inferior del mapa (parte sur), teniendo que cruzar éste hasta la otra punta donde se encontrará el castillo (parte norte), que contendrá la iglesia donde yacerá el boss. Una vez el jugador haya llegado a la zona y siempre y cuando el nivel alcanzado por el personaje, matando los enemigos encontrados durante el camino, tenga un valor mínimo de 2, podrá acceder al castillo. De lo contrario una barrera de energía le impedirá entrar en el recinto. Una vez allí deberá acceder a la iglesia y enfrentarse al boss. Si consigue matar a éste, un acceso a una sala contigua con una entrada cubierta de fuego

*Consulte el glosario de vocabulario situado en la parte final del documento

se desvanecerá y el jugador podrá entrar a recoger el tesoro que yace allí dentro habiendo completado el juego.

De lo contrario, si el jugador durante cualquier momento del juego muere, aparecerá en la zona de regeneración que está situada exactamente en el mismo punto del mapa donde éste aparece por primera vez, con la experiencia y habilidades del personaje intactas y totalmente recuperado de vida. También todos los enemigos volverán a aparecer y el boss recuperará su vida por completo.

4 – DISEÑO E IMPLEMENTACIÓN:

A lo largo de éste capítulo podrá verse el desarrollo de todos los elementos descritos anteriormente en el apartado de análisis.

Para la iglesia y el personaje principal ha sido utilizado Blender, mientras que para el escenario principal y el menú principal del juego se ha utilizando el propio editor que posee Unity3D.

En primer lugar se realizará una breve introducción a éstas dos tecnologías.

4.1 – Introducción a las tecnologías usadas

-4.1.1 – Blender

¿Qué hay que saber?

Blender es un programa de modelaje y animación muy completo, ya que puede hacerse prácticamente todo lo que se puede realizar con los demás programas similares. La dificultad de Blender es que visualmente es difícil de entender y realizar muchas de las tareas que resultarían bastante accesibles en otros programas, en Blender no lo son. Esto es debido a que Blender es un programa que utiliza en su mayoría los *shortcuts**. Tiene una gran combinación de shortcuts que al principio dificultan en gran medida la utilización de éste. Poco a poco y cuando se va cogiendo práctica la utilización de éstos se hace llevadera y la realización de según qué tareas puede volverse más rápida incluso que en otros programas. Para obtener una completa guía con todos los shortcuts de Blender pulsar aquí.

Nota: Es muy importante que se guarde el trabajo que se esté realizando frequentemente debido a que al cerrar la aplicación, Blender nunca pregunta al usuario si desea guardar el trabajo que se está realizando.

*Consulte el glosario de vocabulario situado en la parte final del documento

Interfaz

Blender trabaja con una interfaz que puede ser modificable al 100% a gusto del usuario. De hecho, no van a utilizarse las mismas partes de la interfície para modelaje que para animación, es por ello que se suele usar un tipo de configuración determinada para cada cometido a realizar. En la imagen 12 puede verse la interfície de Blender en su estado inicial (al abrir el Blender)

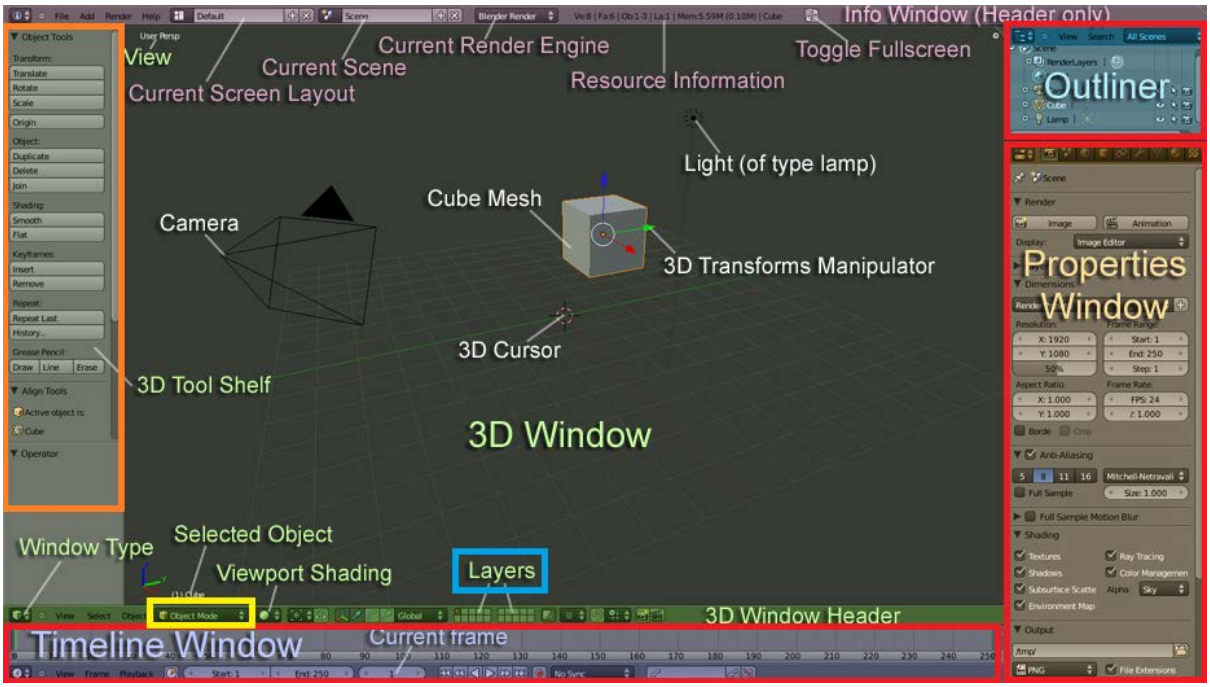


Imagen 12: Interfície por defecto de Blender en su versión 2.6. Imagen obtenida de wiki.blender.org y modificada posteriormente

Como puede verse en la imagen 12 en el recuadro , el "Outliner Window", el "Propierties Window" y el "Timeline Window" son tres de varias ventanas con las que se va a tener que interactuar dependiendo de la tarea a realizar. Para cambiar el tipo de ventana únicamente se debe pulsar sobre el icono de cualquier tipo de ventana y seleccionar la deseada. En la imagen 13 pueden verse el conjunto completo de ventanas disponibles en Blender.

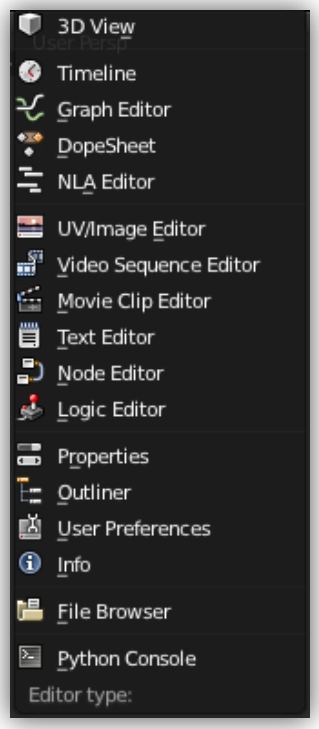


Imagen 13: Tipos de ventana existentes en Blender

A continuación se explicarán de forma genérica los tipos de ventana usados en este proyecto:

- 3D View - Vista del objeto en el estado actual.
- Timeline - Control del tiempo de la ejecución de las animaciones.
- DopeSheet - Ventana necesaria para el control y creación los *keyframes** (se verá cuando se expliquen las animaciones, apartado 4.2.2.5) de las animaciones.
- Graph Editor - Manejo de los keyframes de las animaciones.
- NLA Editor - Control de la secuencia de animaciones no lineales.
- UV/Image Editor - Editor de imagen y texturizado del objeto.
- Properties - Propiedades del objeto seleccionado. Aquí estarán prácticamente todas las opciones de modificación del objeto en sí.

*Consulte el glosario de vocabulario situado en la parte final del documento

- 

- Lista con todos los objetos de la escena, así como las cámaras y luces creadas.
- 

- Diversas opciones para personalizar Blender. Se dispone de todos los *add-on** para descargar.

Cabe decir que hay más tipos de ventana vistos en la imagen 13 que no han sido explicados debido a que no se les ha dado ningún tipo de uso.

Siguiendo con el contenido de la imagen 12, cabe destacar el recuadro  , existen dos de estos recuadros que aparecen a la izquierda y a la derecha del objeto dentro de la ventana 

. Para activar/desactivar estos recuadros hay que utilizar las teclas 

y 

.

Respecto al recuadro  muestra las *layers**. Las layers son distintas capas donde se trabaja sobre el objeto. Es muy importante aprender a utilizar éstas. Si se trata de objetos muy grandes deberán ser divididos en diferentes layers donde se podrá trabajar en cada parte de éste de una manera más precisa y mejor.

Finalmente está el recuadro . En este recuadro se muestra el modo en el que está el objeto. Existen tres modos entre otros en los cuales se trabaja con el objeto. Los más importantes son los siguientes:

- 

- Todo el objeto es seleccionado y todas las operaciones realizadas en este modo afectan a todo el objeto.
- 

- Una parte del objeto es seleccionada y todas las operaciones realizadas afectan a esa zona seleccionada.
- 

- Modo utilizado para la animación del objeto (se explicará éste punto detalladamente en la animación del personaje principal).

*Consulte el glosario de vocabulario situado en la parte final del documento

Utilización de vistas

Éste es un punto muy importante a tener en cuenta y al que se recomienda acostumbrarse desde el inicio de uso del programa. Blender posee un conjunto de vistas para visualizar los objetos desde distintos puntos de vista. Manteniendo el botón central del mouse puede rotarse la cámara en función del cursor 3D, pero se deberá habituarse a utilizar el conjunto de vistas de Blender mediante el *numpad**. Para poder ver el conjunto de vistas con las que trabaja Blender ha de irse a 

y aparecerá la siguiente ventana con todas las vistas disponibles. En la imagen 14 se muestran todas las vistas disponibles sobre el objeto 3D.

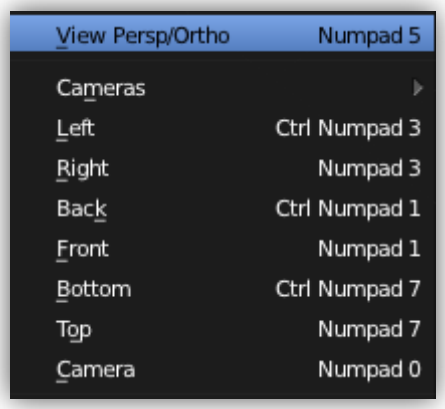


Imagen 14: Vistas disponibles sobre el objeto 3D

Nota importante: Hay una de ellas que está en color azul, ésta es el tipo de vista general que se quiere, si perspectiva o ortogonal. Hay que tener muy en cuenta que la vista con la que se permite un modelaje más cómodo es la ortogonal.

Nota importante 2: Absolutamente todas las vistas no girarán en torno al objeto, sino en torno al 3Dcursor  . Es por ello que deberá de posicionar éste en el lugar que se desee e ir actualizando su posición según convenga en cada momento.

*Consulte el glosario de vocabulario situado en la parte final del documento

¿Cómo realizar el modelaje y las animaciones?

En esta memoria, para cada elemento creado con el editor de Blender en este proyecto se realiza una explicación detallada de cómo utilizar cada parte de la interfície y cómo modelar, texturizar y posteriormente animar los objetos. A medida que vayan apareciendo problemas de modelaje a solucionar, se explicarán y dará una solución a ellos. Véase en los próximos apartados de éste tema.

-4.1.2– Unity3D

¿Qué hay que saber?

Unity3D es un engine bastante amigable y comprensible. No es difícil acostumbrarse a su uso pero se deberán tener en cuenta ciertas cosas. Unity3D funciona con objetos a los que se les llama *GameObjects**. Éste engine funciona mediante *scripting** sobre los GameObjects utilizados en él. Para consultar la *API** de Unity3D pulsar aquí.

Unity3D utiliza *assets**. Estos son el grupo de objetos con los que se trabaja en este engine. Unity3D posee diversos assets que ofrece de manera gratuita y otros que son de pago. Los de carácter gratuito son llamados “Standard Assets” y pueden ser adquiridos pulsando en *Assets -> Import Package*. Cualquier objeto que se importe o se quiera utilizar se deberá guardar dentro de alguna carpeta en “Assets”.

Interfaz

La interfaz de Unity3D es bastante intuitiva. En la imagen 15 puede verse la distribución de las partes que forman la interfície. A continuación se explicará cada una de ellas.

*Consulte el glosario de vocabulario situado en la parte final del documento

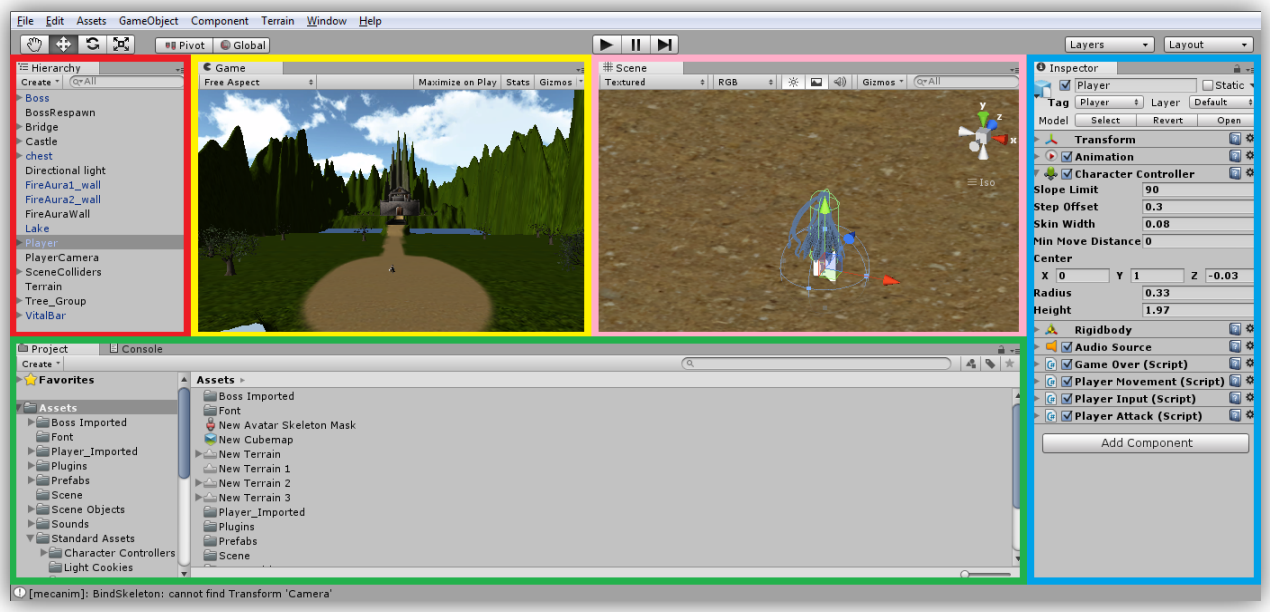


Imagen 15: Interfície de Unity3D con el juego realizado en este proyecto

Hiterachy: Panel en el que están situados todos los GameObjects utilizados en el juego. Los GameObjects que están marcados a su izquierda con una ▶ indican que el objeto está formado por diversas partes.

Game: Panel en el cual será ejecutado el juego al utilizar los botones ▶ ⏸ ⏭ . Se muestra la vista del scene desde la cámara introducida en el juego. Si no se ha colocado ninguna cámara en el juego la pantalla permanecerá en gris.

Scene: Panel en el que se muestran la disposición de los GameObjects del juego así como el escenario creado. Será en este panel donde se realizarán todas las acciones sobre los GameObjects tales como desplazar, rotar, etc.

Inspector: Panel en el que se mostrarán todos los componentes que forman parte del GameObject seleccionado. (Los componentes se explican en el apartado 4.4.2).

En este panel existen dos apartados:

- **Project:** Está situada la carpeta de Assets que contendrá todos los GameObjects guardados (ya sean usados o no) en nuestro proyecto.
- **Console:** Panel que mostrará la posible información solicitada y errores al ejecutar el juego.

Funcionamiento general

Unity3D es un engine que funciona con scenes. Cada scene corresponde a un nivel. Un juego suele dividirse en diversas scene que permitirán una carga menor al ordenador que si se utilizara una única scene. No obstante también existen las llamadas sub-scene que permiten interactuar con varias scene a la vez, cuando éstas han de compartir ciertos recursos. En la imagen 16 puede verse una representación de varias scene y una sub-scene.

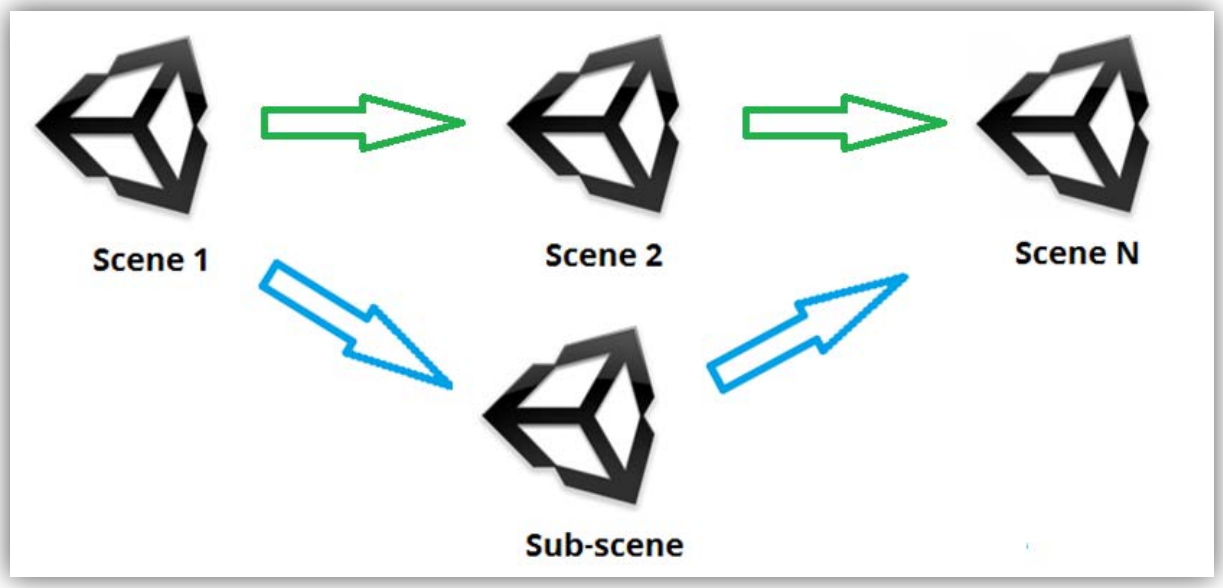


Imagen 16: Representación del funcionamiento de las scene en Unity3D

Se suelen utilizar números para definir las scene. En el caso de este proyecto se utilizan dos scene, una para el menú principal y otra para el propio juego.

¿Cómo utilizar el engine?

En los apartados 4.4 y 4.5 de esta memoria se explica detalladamente de qué manera utilizar los objetos creados y utilizados en el juego.

4.2 – Blender – Modelaje, texturización y animación

En esta sección se explicará el método de creación de la iglesia y el personaje principal que fueron los dos objetos que creados en su totalidad utilizando el editor de Blender.

-4.2.1: Escenario de combate (Iglesia)

-4.2.1.1: Creación del cuerpo (*Mesh**)

Introducción al problema

En la imagen 17 puede verse el resultado final al finalizar todos los pasos de creación, texturización y añadido de elementos internos y externos de la iglesia.

Imagen 17: Iglesia en su estado final

*Consulte el glosario de vocabulario situado en la parte final del documento

Toda la iglesia se ha realizado mediante la combinación de polígonos básicos y de modificadores (modifiers) propios del Blender. Estos modifiers son una herramienta imprescindible (aplicada a los objetos) para la creación y modelaje de las distintas partes de éstos y más adelante podrá verse el motivo. Antes de comenzar con la explicación del modelaje de la iglesia se explicarán los modificadores que van a ser utilizados.

Cabe decir que para poder mantener la escala adecuada de la iglesia, se tuvo que diseñar ésta con todas las proporciones correctas con respecto al personaje principal y al jefe final del juego (tal y como puede verse en el punto 3.4)

Debido a que el escenario era simétrico o prácticamente simétrico, se decidió optar por crear ¼ parte de este.

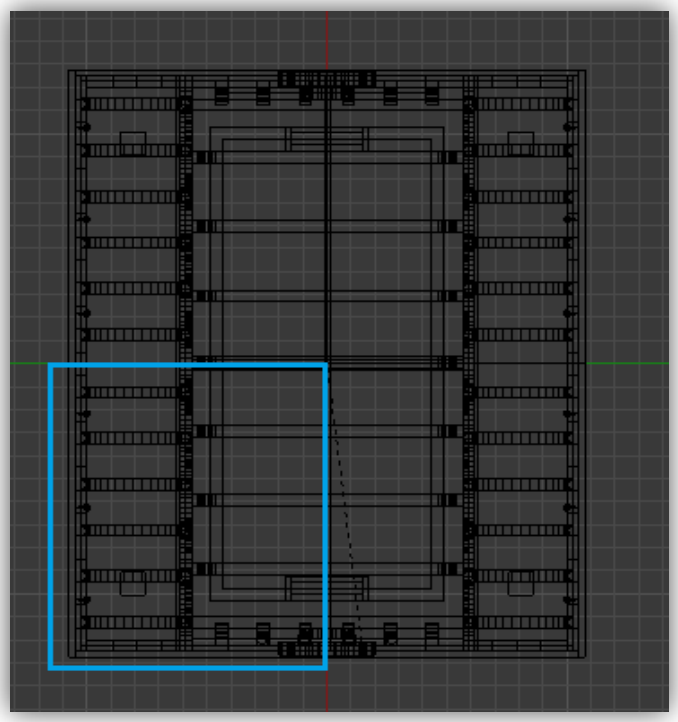


Imagen 18: Iglesia en su estado final

El recuadro azul indica la parte del escenario que se realizó, mientras que el resto, exceptuando pequeños detalles fueron clonados (más adelante podrá verse cómo)

En la imagen 19, situadas en la parte inferior puede observarse el estado final en la que se encontraba la iglesia antes de la aplicación del "mirror modifier".

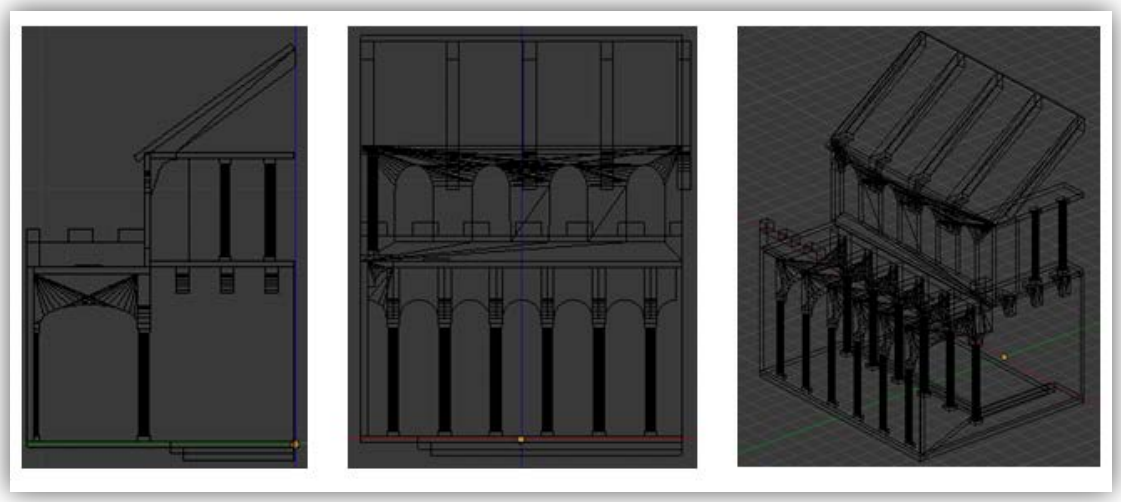


Imagen 19: En las imágenes de arriba puede verse la representación de ¼ de la iglesia. Vista frontal (izquierda), vista lateral (centro) y vista aérea (derecha)

MODIFICADORES UTILIZADOS (Modifiers):

Un modifier es un elemento que es aplicado a un objeto y que tienen un comportamiento no destructivo sobre éste. Permite realizar muchos efectos que, de hacerlo sin éstos modifiers sería una tarea muy tediosa. Un objeto no tiene límite en cuanto a los modifiers que se le pueden aplicar. En la imagen 20 puede visualizarse la lista completa de modifiers aunque en este proyecto únicamente serán usados los más comunes.

Imagen 20: Lista de modificadores de Blender

- Mirror Modifier

Éste modificador permite de manera automática duplicar un mesh en sus ejes locales X,Y y Z siempre en función del punto central del mesh. También permite duplicarse en función del punto central de otros mesh. En la imagen 21 puede verse el resultado de aplicar el mirror modifier a un objeto en su eje X.

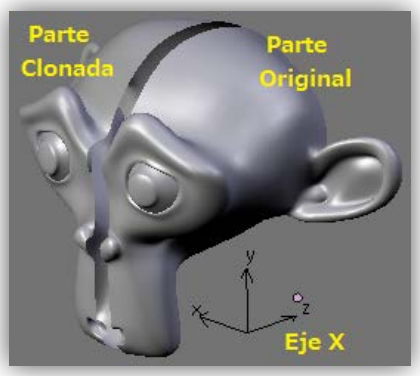


Imagen 21: Resultado de la aplicación del mirror modifier

Dispone también de dos opciones muy interesantes, la primera es el clipping, una característica de este modifier que previene que los vértices no se crucen entre ellos. En la imagen 22 puede visualizarse una demostración del efecto clipping entre dos cuadrados.

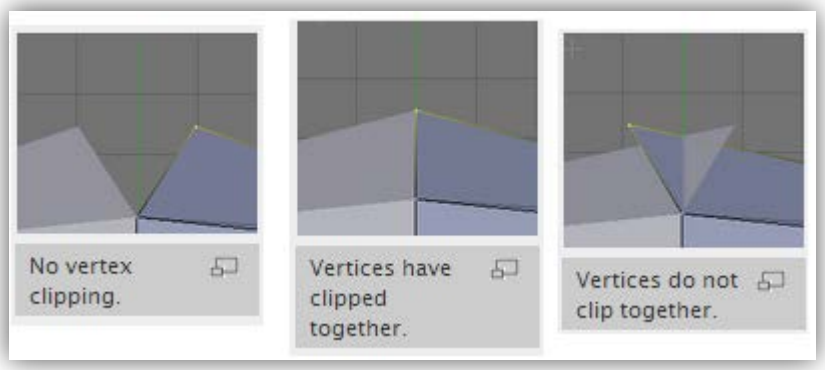


Imagen 22: Efecto de clipping sobre dos cuadrados

Y la segunda es merge, una característica que permite fijar la máxima distancia entre los vértices del objeto original y el de la copia.

- Boolean Modifier

Para la realización de éste contorno en forma de arco se utilizó un modifier también muy utilizado llamado boolean modifier. Este modificador nos permite, teniendo dos figuras, realizar las operaciones de unión, intersect y/o difference. En la imagen 23 puede visualizarse el panel de aplicación del boolean modifier para la combinación de un cubo y una esfera.

Imagen 23: Panel de aplicación del boolean modifier

Se dispondrá entonces de tres operaciones:

1- **Difference** – El mesh objetivo se sustrae del mesh que se modifica.
2- **Union** - El mesh objetivo se suma al mesh que se modifica.
3- **Intersection** – El mesh objetivo se sustrae del mesh que se modifica, quedando la parte común de ambos mesh.

En la imagen 24 puede verse la aplicación, entre un cubo y una esfera, de la operación de difference (izquierda), union (centro) e intersection (derecha).

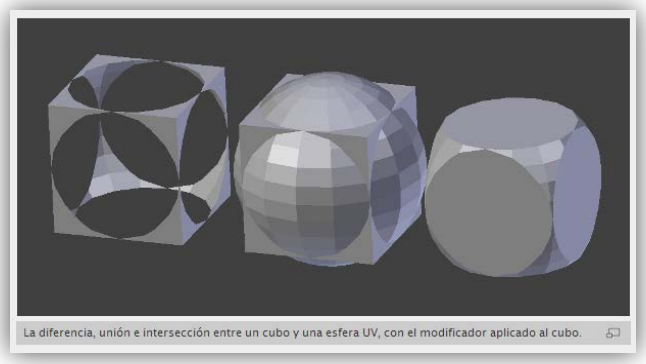


Imagen 24: Efectos de las distintas operaciones del boolean modifier

- Array Modifier

Éste modifier permite copiar un objeto y situarlo al lado de la base con un *offset** específico. Esto permite realizar grandes escenas que contengan varios objetos idénticos y/o que mantengan una simetría perfecta dentro de la escena. En la imagen 24 puede visualizarse el panel de aplicación del array modifier para la copia de algún objeto.

Imagen 24: Panel de aplicación del array modifier

Se dispondrá entonces de tres operaciones "Fit Type":

1- **Fit Curve** - Se habrá definido una curva previamente y éste modificador generará copias del objeto hasta llenar toda la curva con ellas.
2- **Fit Lenght** - Éste generará copias del objeto hasta llenar toda la longitud definida.
3- **Fixed Count** - Generará el número la cantidad de copias del objeto que se haya definido en el contador (count).

MODELADO ¼ DE LA IGLESIA:

Para poder explicar todo el proceso de modelaje de la iglesia, se dividirá en las tres zonas que pueden verse en la imagen 25: Zona 1 (primer piso), Zona 2 (segundo piso) y Zona 3 (techo).

*Consulte el glosario de vocabulario situado en la parte final del documento

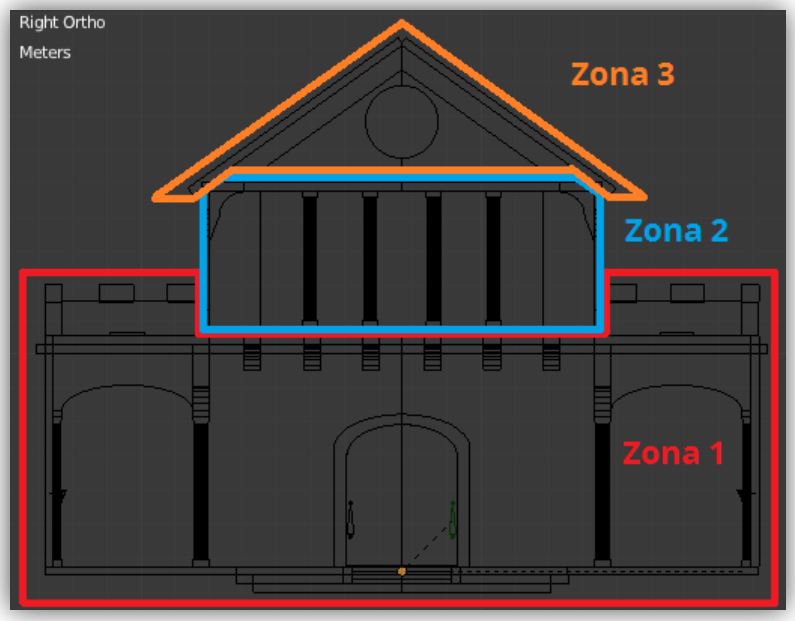


Imagen 25: División del modelaje de la iglesia en tres zonas

ZONA 1

El primer elemento que fue creado fue el suelo que soportaría la estructura exterior de la iglesia.

Para la creación del suelo simplemente se partió de un cuadrado 1 y se le añadió una base inferior 2 , y a esta una base inferior 3 tal y como puede verse en la imagen 26.

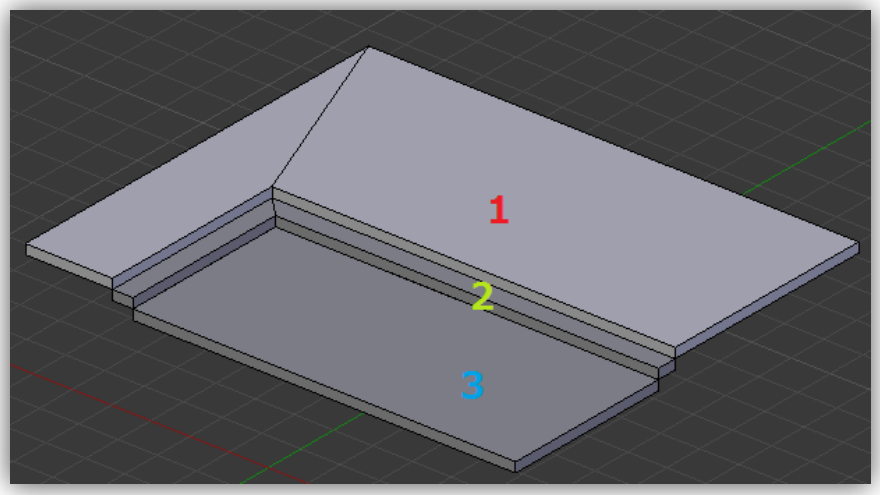


Imagen 26: División del suelo en tres partes

Una vez finalizada la base de apoyo de la estructura de la iglesia se empezó con las paredes de ésta. Las paredes son simples rectángulos con una anchura delgada. Una vez que éstas fueron posicionadas se creó el suelo del segundo piso que soporta las almenas. Para la realización de estas se crearon dos bases 1 y 2 que soportarían las almenas. En cuanto a éstas, se creó una y se le aplicó un array modifier para realizar la copia de éstas. Se tuvo que aplicar el array modifier dos veces, una para cada base debido a que las direcciones de copia del array son distintas. En la imagen 27 se muestra la creación de las almenas de la iglesia.

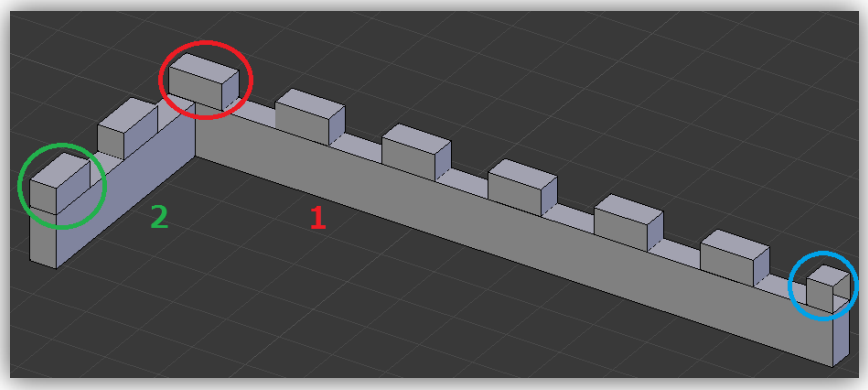


Imagen 27: Creación de las almenas de la iglesia

En la imagen 27, como puede verse en la redonda hay media almena en lugar de una completa. Esto es debido a que justo allí finaliza el cuarto de la pieza de la iglesia, es decir, faltaría la otra mitad de la muralla con las almenas y la redonda sería el punto de unión. Esto se conseguirá aplicando un mirror modifier. También ocurrirá lo mismo con ésta otra parte, la redonda que puede verse en la imagen 28. En esta imagen se muestran los puntos de unión con los que hay que vigilar antes de aplicar el mirror modifier, por lo tanto se deberá pensar previamente cómo se organizarán las uniones para que quede completamente simétrico.

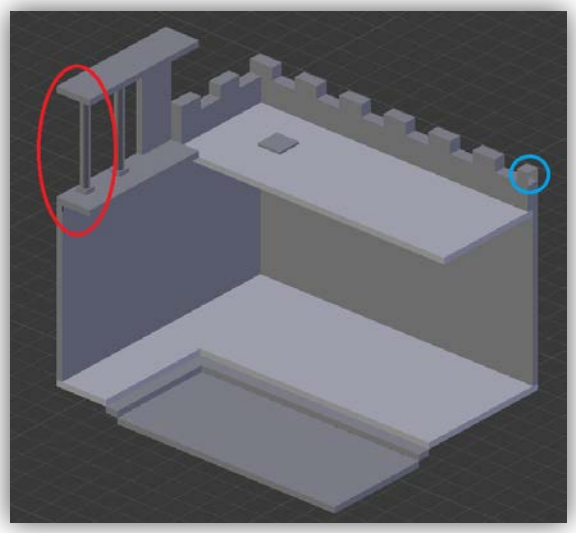


Imagen 28: Puntos de unión entre los cuartos de la iglesia

ZONA 2

En esta zona se contempla el segundo piso y los elementos internos del primer piso, soportes, vigas, columnas y arcos de soporte del segundo piso.

En primer lugar fue creada la pared del segundo piso que contiene las ventanas con la parte superior de éstas en forma de arco (señalada con un recuadro en rojo) tal y como puede verse en la imagen 29.

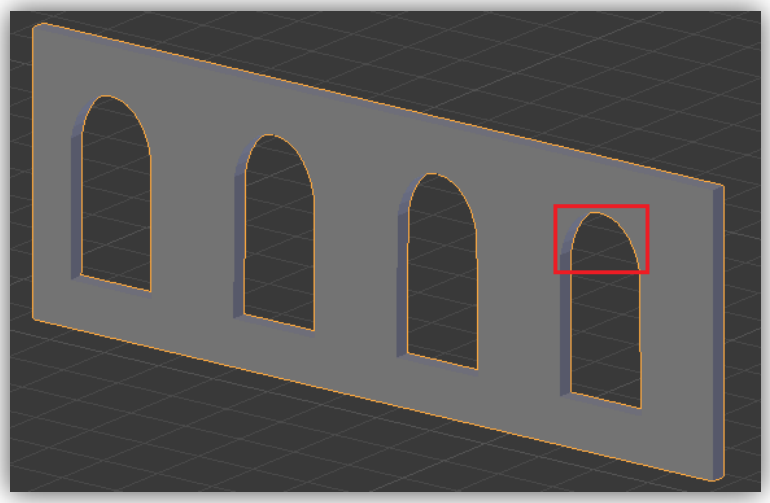


Imagen 29: Pared del segundo piso con la parte superior de las ventanas en forma de arco

Para la ventana que se quiere realizar, se utilizará un rectángulo al que se le llamará pared y un cuadrado al que se le habrá añadido medio cilindro encima al que se le llamará "polígono ventana".

Para la realización del "polígono ventana" se seguirán los siguientes pasos:

1- Crear un cuadrado
2- Crear un cilindro y eliminar la mitad de este
3- Unir ambos objetos con la forma deseada (altura)

Para poder realizar la ventana, se aplicará el método de difference del boolean modifier aplicado a la pared principal respecto al objeto polígono ventana. De esta manera podrá obtenerse la forma de éste en la pared. Puede verse la pared y la realización de la ventana por separado en la imagen 30.

Teniendo los dos objetos ya preparados para aplicar el boolean modifier.

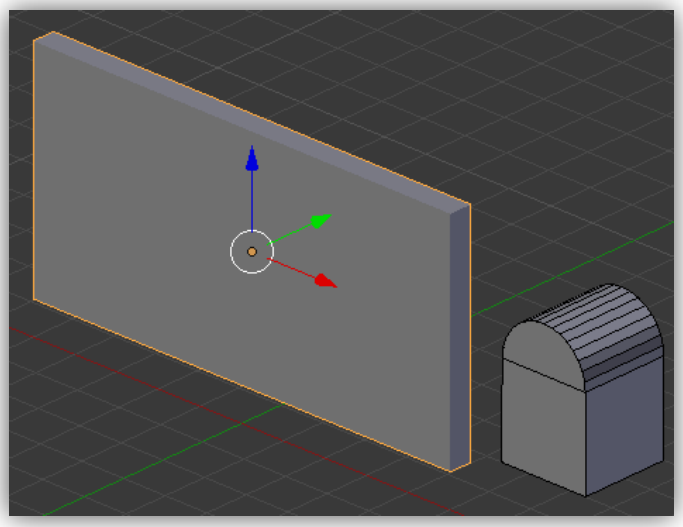


Imagen 30: Pared y ventana creadas por separado

Seguidamente, se coloca el "polígono ventana" encima de la pared en la zona deseada.

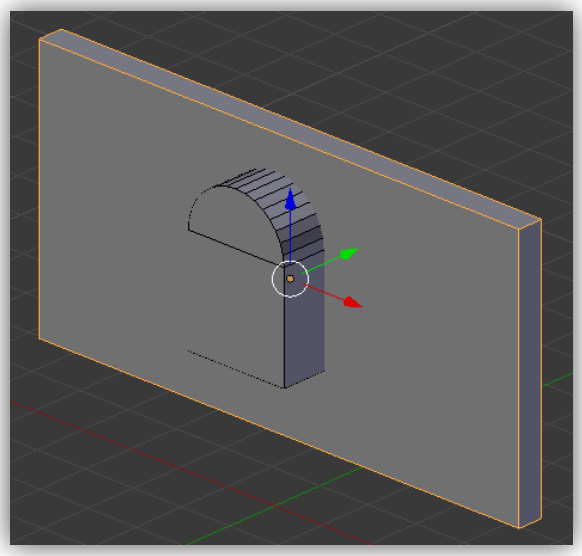


Imagen 31: Ventana situada en la posición deseada antes de aplicar el boolean modifier

A continuación se le añade el boolean modifier sobre la pared para que éste actúe respecto a la figura que se le indique, en este caso sobre el polígono ventana (imagen 31).

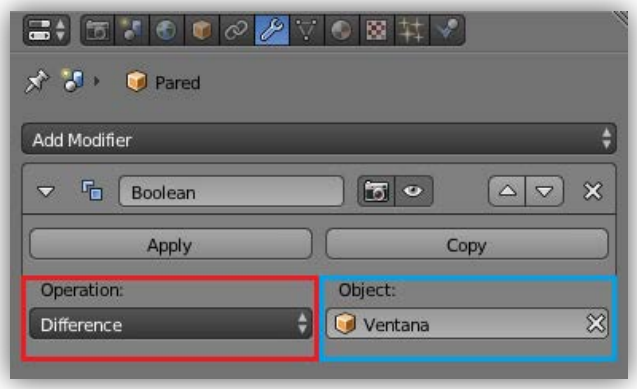


Imagen 32: Panel de aplicación del boolean modifier con los objetos a los que se aplicará

Como puede verse en la imagen 32, se selecciona la operación difference , para substraer toda la parte del objeto "Ventana" que está en contacto con el objeto "Pared". Una vez hecho esto se conseguirá lo esperado. Véase en la imagen 33 el resultado final de aplicar el boolean modifier entre la pared y la ventana.

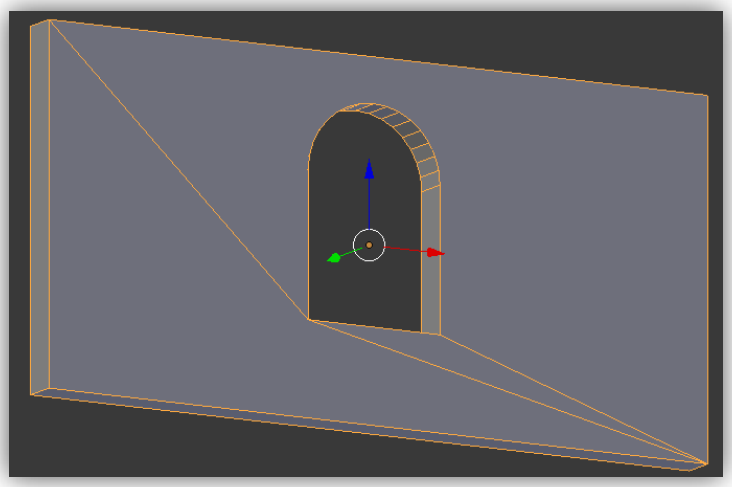


Imagen 33: Pared finalizada con la ventana hecha

Con ésta misma técnica de boolean modifier se realizaron muchas de las formas que forman parte del escenario, tales como los pilares de soporte de las vigas del techo y las columnas delanteras del primer piso, todos los arcos del primer piso que soportan el segundo piso y las puertas para poder entrar la iglesia tal y como se muestra en la imagen 34.

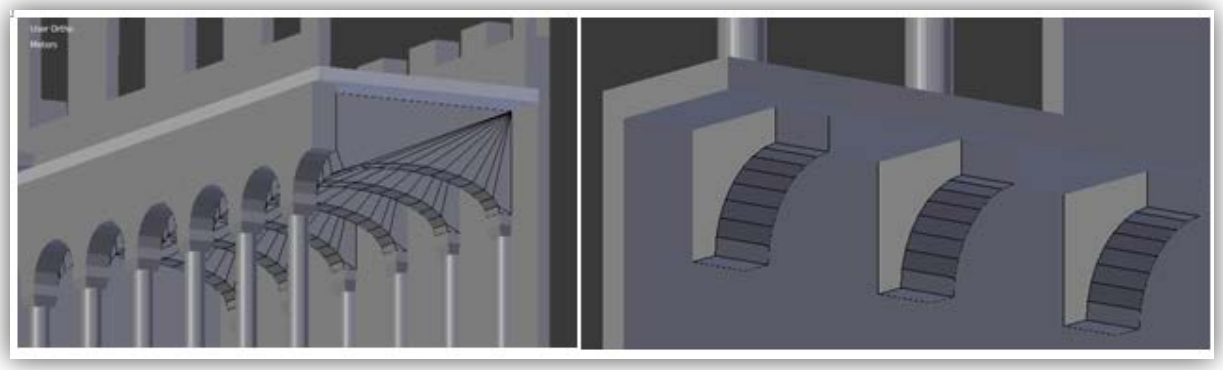


Imagen 34: Vigas y arcos del segundo piso realizados con la utilización del boolean modifier

A continuación, una vez acabado el segundo piso, se siguió con los elementos interiores del primer piso, que serían todas las columnas, los soportes en forma de arco del segundo piso, los soportes de las vigas y el suelo de éste.

Para esta parte se utilizó mucho el array modifier permitiendo realizar una simetría de elementos perfecta.

En el caso de las vigas (véase la imagen 35), únicamente se debió de crear una utilizando un cuadrado y un cuarto de cilindro y aplicando la difference del boolean modifier entre ambos tal y como se hizo con la ventana del segundo piso.

Imagen 35: Viga de soporte

Una vez creada, se coloca en la zona deseada y se le añade el array modifier al objeto en cuestión. Y tal y como puede verse en la redonda roja, se señala en el contador count la cantidad de veces que se quiere copiar el objeto. En la imagen 36 puede verse la distribución del panel de aplicación del array modifier.

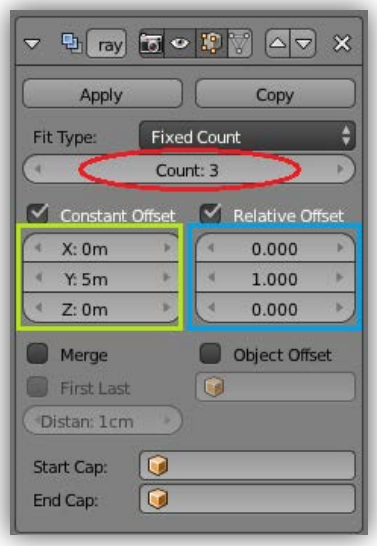


Imagen 36: Panel de aplicación del array modifier

El rectángulo verde le permite asignar los valores de separación en el eje que se desee. En éste caso el eje Y será en el cual se copiarán, y por ello se le da una separación de 5m entre objeto y objeto.

En el rectángulo azul se indica en qué eje se quiere copiar el objeto, en nuestro caso el eje Y. Una vez hecho esto se obtiene el conjunto de vigas clonado tal y como se puede visualizar en la imagen 37.

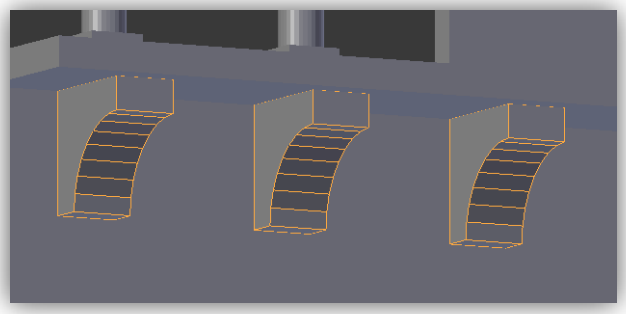


Imagen 37: Resultado final de las vigas clonadas

Ahora toca modelar una de las partes más largas de la iglesia, las columnas y arcos del primer piso. En la imagen 38 puede verse la división del proceso de creación de estas partes. A continuación se da una explicación detallada de cada parte.

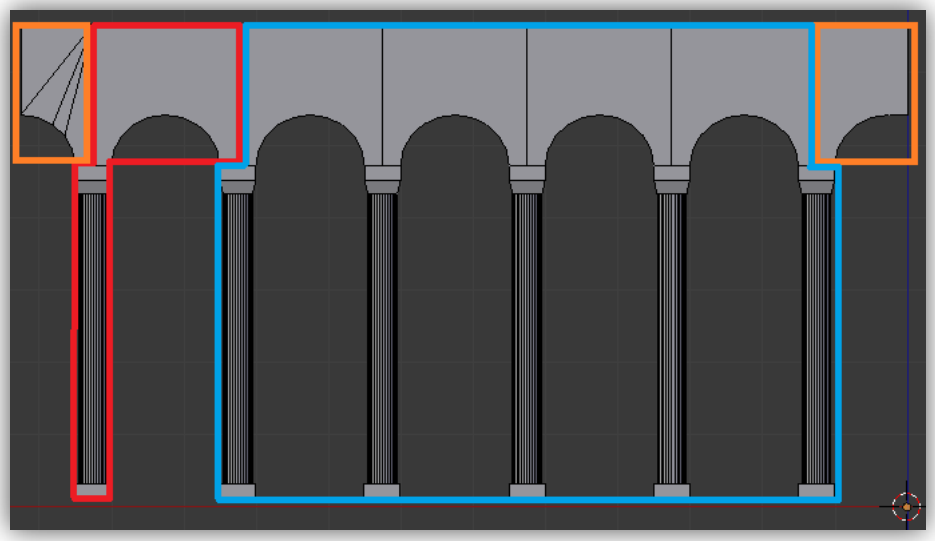


Imagen 38: División del proceso de creación de la Estructura 1

Resumen (Estructura 1- columnas & arcos)

Trozo original realizado. Columna formada por un cilindro alargado con una base de soporte cuadrada. La base superior ha sido realizada mediante la combinación de dos cuadrados, uno de ellos modificado para darle esta forma de medio cono. Finalmente al arco horizontal que une las columnas ha sido realizado aplicando la difference del boolean modifier (de la misma forma que se realizó la ventana del segundo piso).

Al tener ya la columna con el arco, se aplicó un array modifier pudiendo así copiar el objeto cuatro veces más.

Arco que colisiona con la pared de la iglesia (izquierda) y con la otra mitad de la iglesia cuando se le aplique el mirror modifier (derecha). Estas partes no añadidas hasta que no haya sido finalizado y .

Ahora toca acabar la unión de la estructura realizada anteriormente con la pared. Para ello se decidió hacerlo de la siguiente forma.

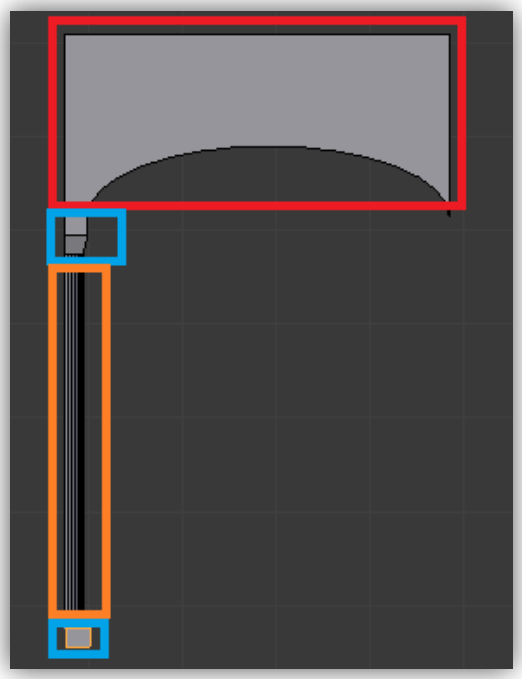


Imagen 39: División del proceso de creación de la Estructura 2

Resumen (Estructura 2- conexión columnas & arcos con pared)

Arco que soporta el segundo piso realizado a partir de un medio cilindro aplanado y un rectángulo al que se le ha aplicado la difference de un boolean modifier.

Columna realizada a partir de medio cilindro alargado

Base inferior realizada a partir de medio cuadrado y base superior realizada a partir de media combinación de dos cuadrados.

Una vez finalizada, a toda esta estructura se le añadirá un array modifier para obtener las distintas copias necesarias.

Tal y como puede verse en la imagen 40, al combinarse las dos estructuras anteriormente descritas generarán una nueva estructura encargada de soportar todo el peso del segundo piso.

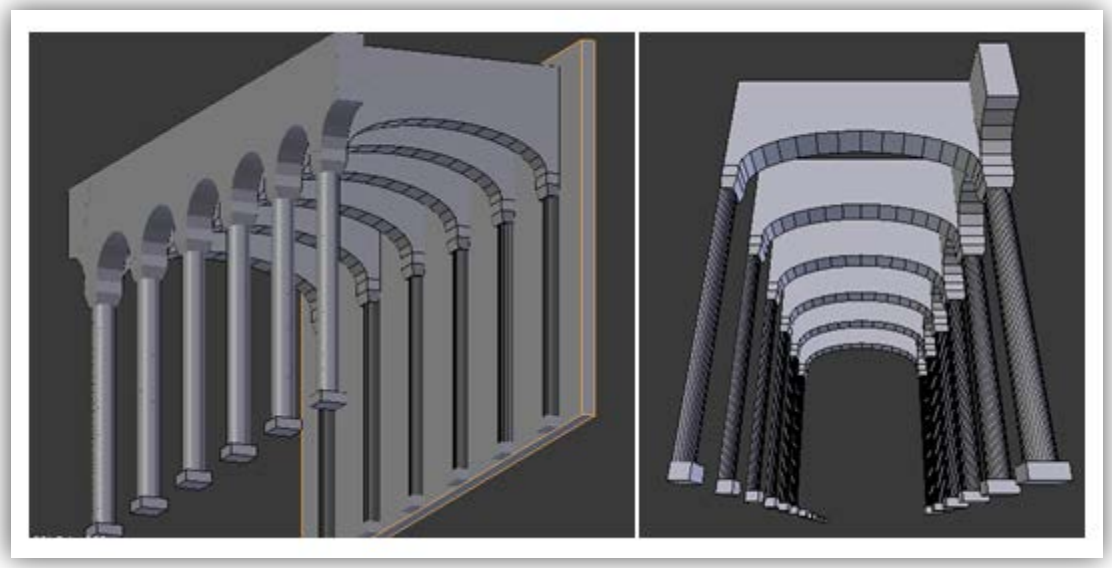


Imagen 40: Resultado final y combinación de la estructura 1 & la estructura 2

ZONA 3

Ésta zona contiene el techo y la parte frontal de la iglesia. Para el modelaje del techo se siguió utilizando las técnicas descritas anteriormente, ya que está formado por vigas, soportes y una plancha en forma de rectángulo. Véase la imagen 41 que contiene un lado (¼) del techo de la iglesia con sus respectivos soportes.

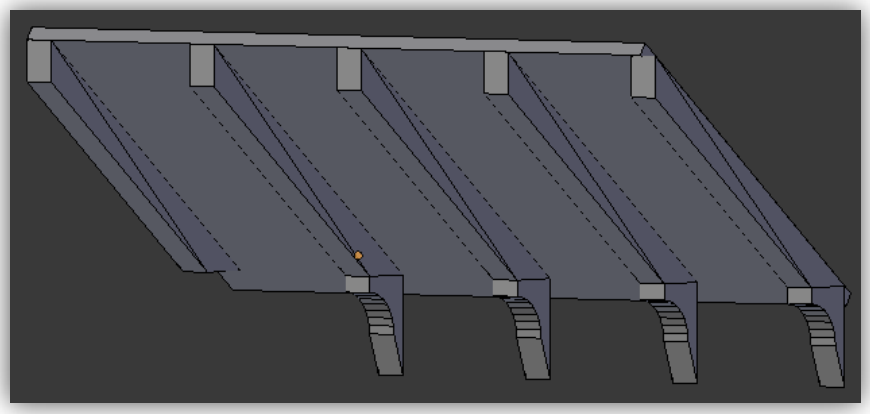


Imagen 41: ¼ del techo de la iglesia finalizado

Por último únicamente queda la parte frontal de la iglesia, véase en la imagen 42. Ésta está formada por un triangulo al cual se le ha aplicado un boolean modifier con la opción difference entre éste y un cilindro formando el circulo del medio. Esta pieza estará en el segundo piso soportando el techo de la iglesia.

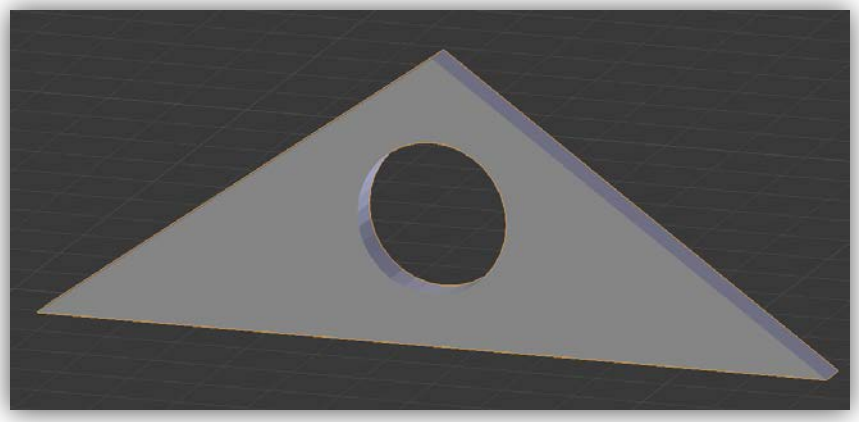


Imagen 42: Parte frontal de la iglesia

FINALIZACIÓN DE LA IGLESIA:

Y una vez hemos terminado definitivamente ¼ de escenario, toca realizar las copias con efecto espejo en sus ejes X e Y.

Empezando por el eje Y, se aplicará a cada una de las piezas que forman parte de este ¼ de iglesia un mirror modifier (en lugar de ir pieza por pieza también puede previamente realizarse una unión de todas las piezas haciendo del conjunto de éstas un único objeto al cual se le aplica el mirror modifier, pero esto podría dar problemas si luego se quiere realizar algún pequeño cambio visual en el escenario ya que no se podrá acceder a ninguna pieza de forma independiente). En la imagen 43 puede verse la representación de la aplicación del mirror modifier en el eje Y.

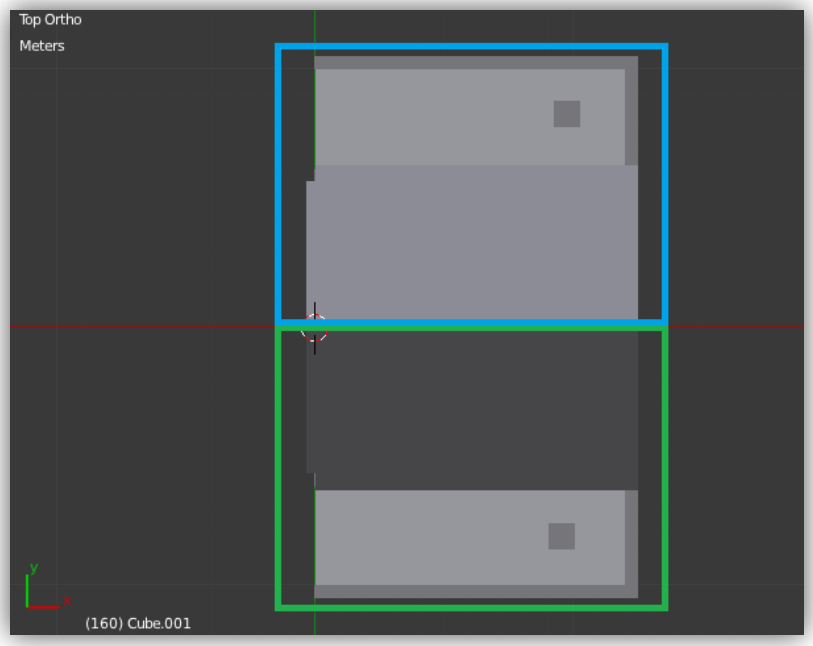


Imagen 43: Aplicación del mirror modifier en el eje Y al ¼ de la iglesia

El rectángulo en verde señala la parte original que se tenía previamente mientras que el rectángulo azul señala la parte duplicada que como puede verse es completamente simétrica. En la imagen 44 se muestra el panel del mirror modifier que ha sido aplicado. En este se señalan todas las características para realizar tal efecto, tiene le siguiente aspecto.

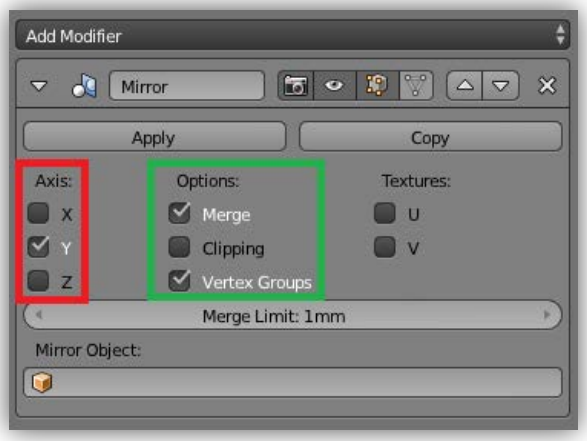


Imagen 44: Panel de aplicación del mirror modifier

En el recuadro en rojo se marca el eje sobre el cual se quiere la afectación del modificador. En este caso este se realizará en el eje Y, aunque podría haberse empezado por el eje X ya que no importa el orden. En el recuadro verde cabe destacar que hay la opción de clipping que por defecto estará desmarcada, y deberemos marcarla si se quiere que haya una perfecta unión de los vértices entre el objeto original y objeto duplicado. En la imagen 45 puede visualizarse el resultado de la aplicación del mirror modifier en el eje Y de la iglesia.

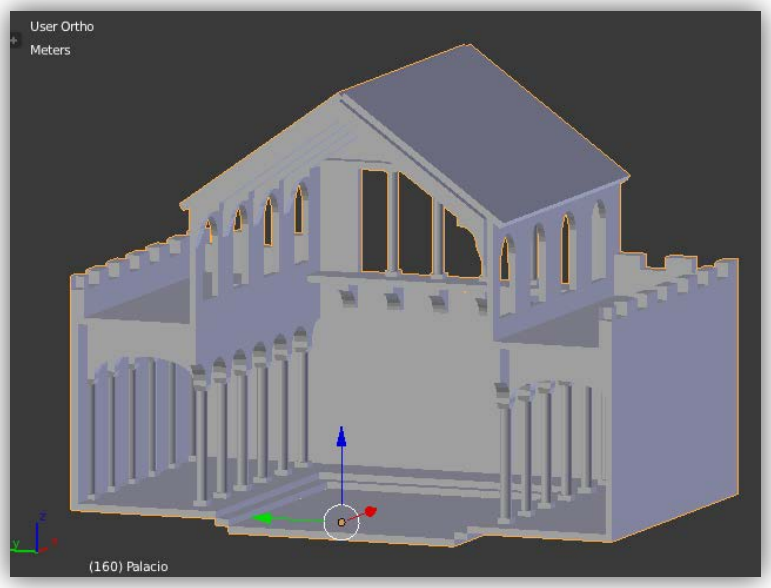


Imagen 45: Iglesia con el mirror modifier aplicado en su eje Y

El siguiente paso será el aplicar otra vez un mirror modifier de la misma manera que se le ha aplicado a Y pero cambiando al eje X y añadiéndole el modifier a los mismos objetos en los que se les añadió anteriormente. Como ya se ha dicho antes, un objeto puede llevar varios modifiers, así que no hay ningún tipo de problema. Pero hay que tener cuidado con el orden en el que se introducen éstos ya que los que se apliquen primero tendrán efecto directo sobre los modifiers que se apliquen a continuación y así sucesivamente. Esto permitirá que al aplicar el mirror modifier en el eje X éste será afectado por el previo mirror modifier en el eje Y e causará la duplicación del escenario en ambas partes. En la imagen 46 puede visualizarse el resultado final de aplicar un mirror modifier en el eje Y a la mitad de la iglesia obtenida en la imagen 45.

Imagen 46: Iglesia completada con la aplicación de los mirror modifier en su eje X e Y

Nota: En esta imagen se le ha quitado la pared trasera duplicada para poder ver el interior del escenario.

Como añadido final, se le añaden las puertas delantera y trasera y se deja preparado para su posterior texturización. En la imagen 47 se muestra la iglesia modelada acabada en su totalidad una vez se han realizado todas las acciones anteriormente descritas.

Imagen 47: Mesh de la iglesia completado en su totalidad

- 4.2.1.2: Aplicación de las texturas (Texturización)

La texturización de objetos es uno de los procesos más significativos para la visualización del objeto. Es un proceso sencillo pero en el cual hay que seguir un cierto orden debido a que el no hacerlo conllevará a un resultado distinto al que se esperaba. Los pasos correctos que tienen que seguirse están citados a continuación:

- **Selección del objeto a texturizar**
- **Creación de un material**
- **Asignación de propiedades específicas al material para este objeto**
- **Asignación de una textura al material**
- **UV Mapping del objeto**
- **Aplicación de la textura creada al UV Mapping del objeto**

A continuación se muestra el proceso completo para poder texturizar una parte de la iglesia. Cabe decir que para el resto de las partes de la iglesia se seguirá en general el mismo procedimiento.

- Selección del objeto a texturizar

En la imagen 48 se muestra la parte frontal de la iglesia, parte que será utilizada como demostración de los pasos a seguir para la texturización de la parte.

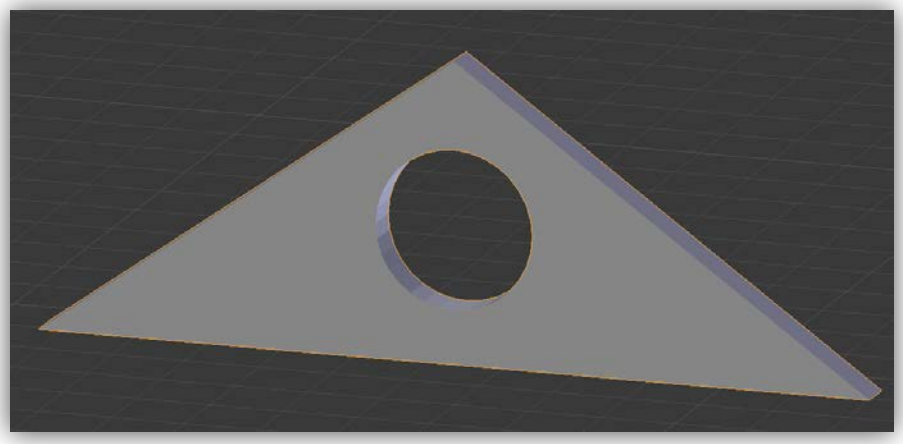


Imagen 48: Parte frontal de la iglesia sin texturizar

- Creación de un material

Es necesario que, antes de aplicar la textura se le dé al objeto un material, que permitirá escoger un color para éste y darle unas propiedades específicas, tales como brillo, reflejo, etc. En el caso dado, serán todos neutros a efectos de propiedades específicas y el material será color blanco (aunque el color podría ser el que se quisiera debido a que al insertar una textura, ésta se sitúa por encima del material y el color de origen de éste desaparece).

El primer paso será entonces la aplicación de un material, sin importar el color de éste. Para ello se deberá seleccionar el objeto, seleccionando el siguiente símbolo  y seguidamente  para añadir un nuevo material dándole el nombre deseado. En la imagen 49 puede visualizarse el panel que permite la creación de dicho material.

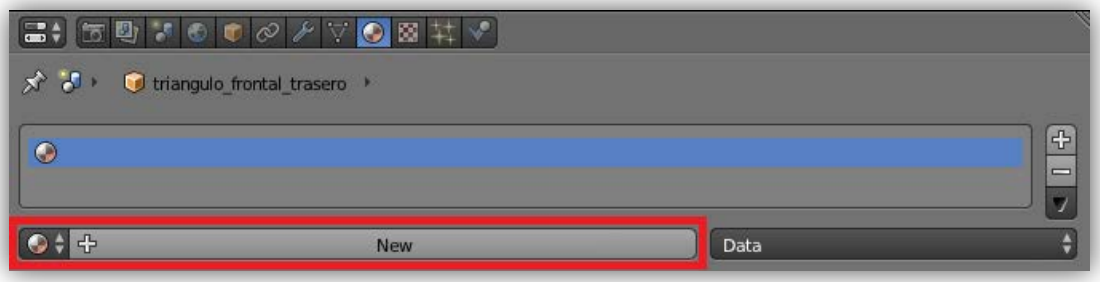


Imagen 49: Creación de un nuevo material

En el caso que se quiera utilizar un material anteriormente creado, véase en la imagen 50 (situación que se repetirá muchas veces debido a que muchos de los objetos compartirán texturas), se deberá escoger éste de la lista donde se disponen de todos los materiales creados anteriormente.

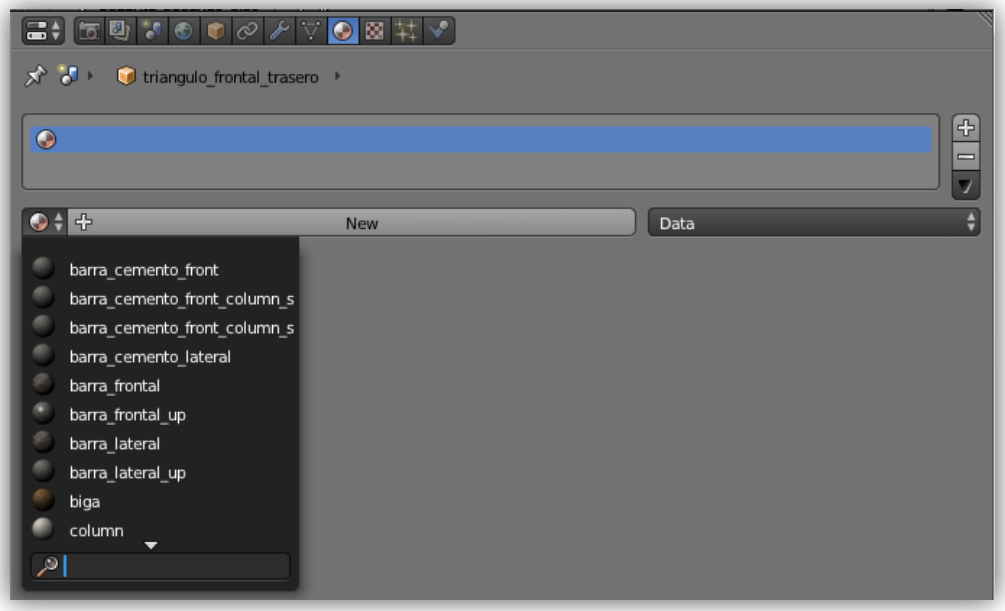


Imagen 50: Panel de elección de materiales anteriormente creados

- Asignación de propiedades específicas al material para este objeto

Los materiales tienen un conjunto de propiedades que podrán ser modificadas. Véase en la imagen 51 donde se muestra el panel con algunas de las propiedades de éste.

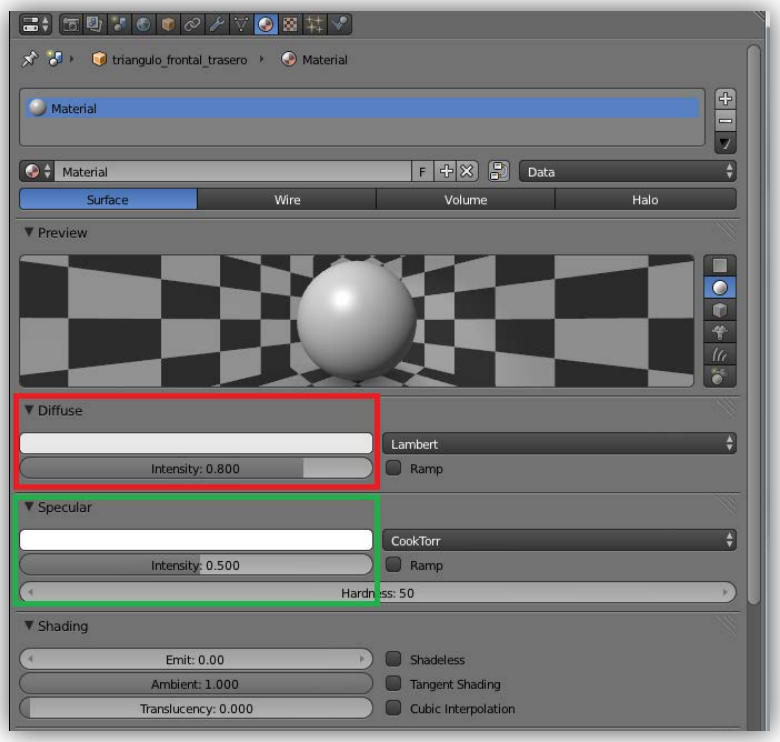


Imagen 51: Algunas de las propiedades específicas de los materiales

Las propiedades principales que deberán ser modificadas serán la Reflection Diffuse y la Reflection Specular. En este caso se han situado sus intensidades en 0.5 para la Diffuse (recuadro ▭) y 0.0 para la Specular (recuadro ▭). Para un efecto deseado se deberá variar estos valores hasta dar con el efecto de reflejos que se quiere conseguir.

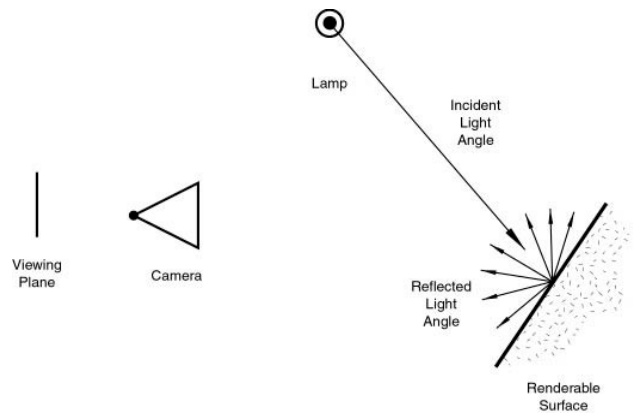


Imagen 52: Efecto del Diffuse Reflection

Diffuse Reflection

Esta reflexión determina básicamente el color general de un material cuando la luz refleja en contra de él. En la imagen 52 puede verse el efecto explicado anteriormente

Specular Reflection

Esta luz determina el reflejo directo del haz de luz que se refleja en la figura. Este tipo de reflexión permite que la superficie parezca pulida. En la imagen 53 puede verse el efecto explicado anteriormente

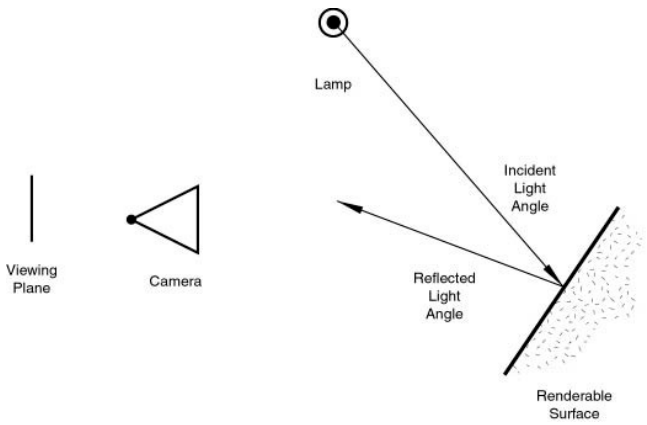


Imagen 53: Efecto del Specular Reflection

Las propiedades de Specular y Diffuse son las únicas dos propiedades que suelen ser modificadas a pesar de que existen algunas más tales como “Transparency”,” Shading” y “more Options”.

- Asignación de una textura al material

Seleccionando el siguiente símbolo  y posteriormente  para añadir una nueva textura al material. Véase en la imagen 54 el panel que permite realizar la acción anterior descrita.

Nota: El nombre del material ha sido modificado de “Material” a “Pared.001”

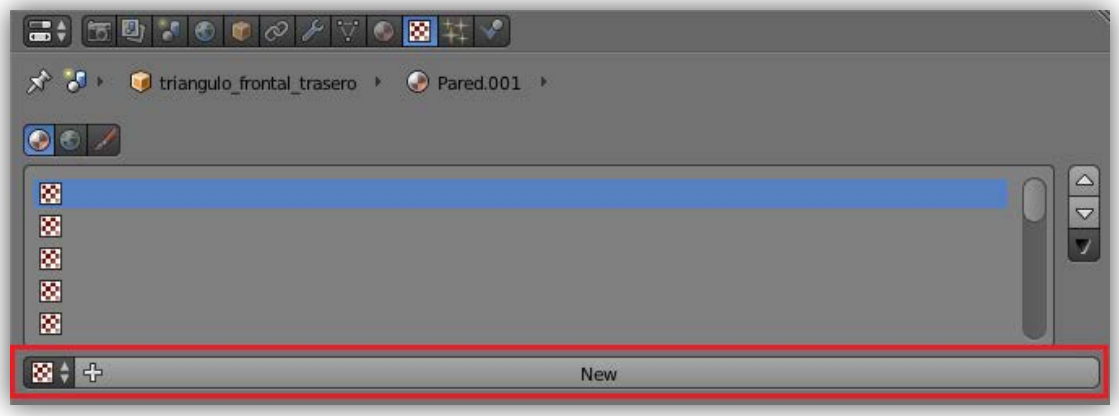


Imagen 54: Asignación de una nueva textura al material

Aparecerá la imagen 55 en la que se muestra un panel donde se nos dará a escoger entre varias posibilidades. Pero la que interesa en este caso es la de insertar una textura propia que previamente se habrá buscado. Por ello se seleccionará la opción de "Image or Movie".

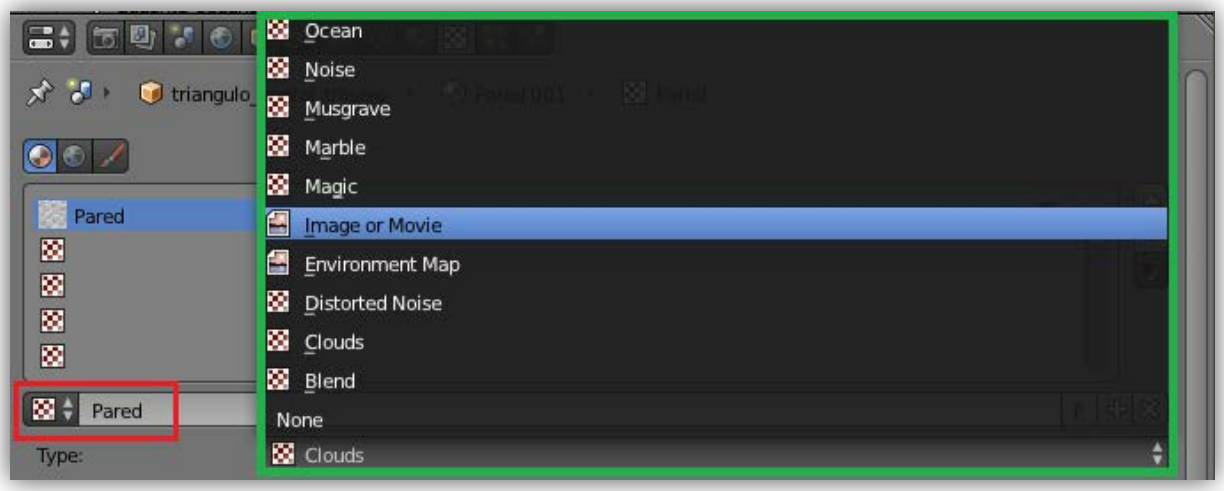


Imagen 55: Selección de una textura de tipo "Image or Movie"

Ahora tocará cargar la imagen que se desee aplicar al material. Véase en la imagen 56 el icono que permite abrir dicha imagen.

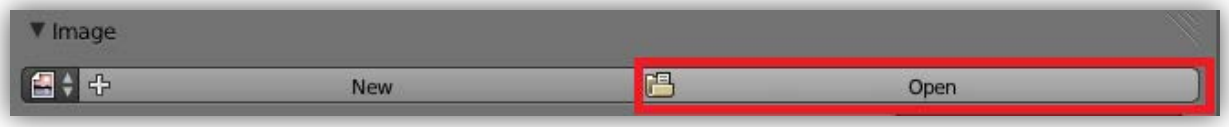


Imagen 56: Botón de carga de una textura propia

Una vez acabado este proceso ya se tendrá listo el material y la textura para añadirla al objeto en cuestión. Tal y como puede verse en la imagen 57 se dispondrá entonces del siguiente estado, donde el recuadro ▭ mostrará el material con la textura finalmente aplicada si todo ha ido correctamente y por el contrario se mostrará el material con el color previamente dado si no se ha podido aplicar la textura.

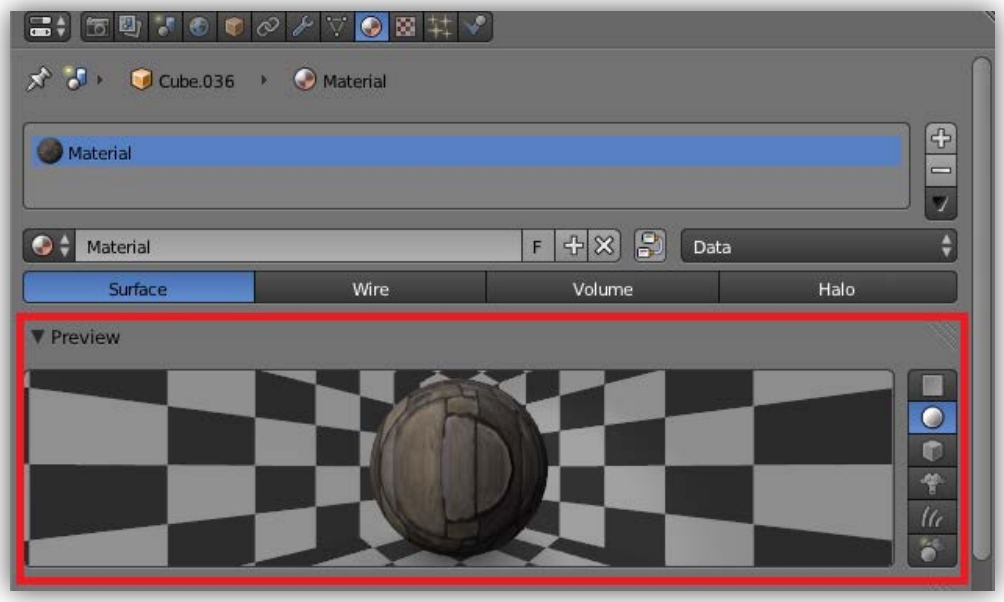


Imagen 57: Material con la textura aplicada correctamente

- UV Mapping del objeto

¿Qué es y cómo funciona?

El UV Mapping es el proceso para poder *mapear** de una textura 2D en un objeto 3D. Describe qué parte de una textura es aplicada a cada polígono que forma el objeto. A cada vértice de cada polígono se le es asignado unas coordenadas 2D (U y V). La idea viene dada de la descomposición del objeto 3D a un plano 2D. En la imagen 58 puede verse la acción de despliegue a UV Mapping de las caras de un cubo mientras que en la imagen 59 se representa el despliegue a UV Mapping de la bola del mundo, la texturización de éste y la posterior aplicación del de la textura mapeada sobre la bola del mundo.

*Consulte el glosario de vocabulario situado en la parte final del documento

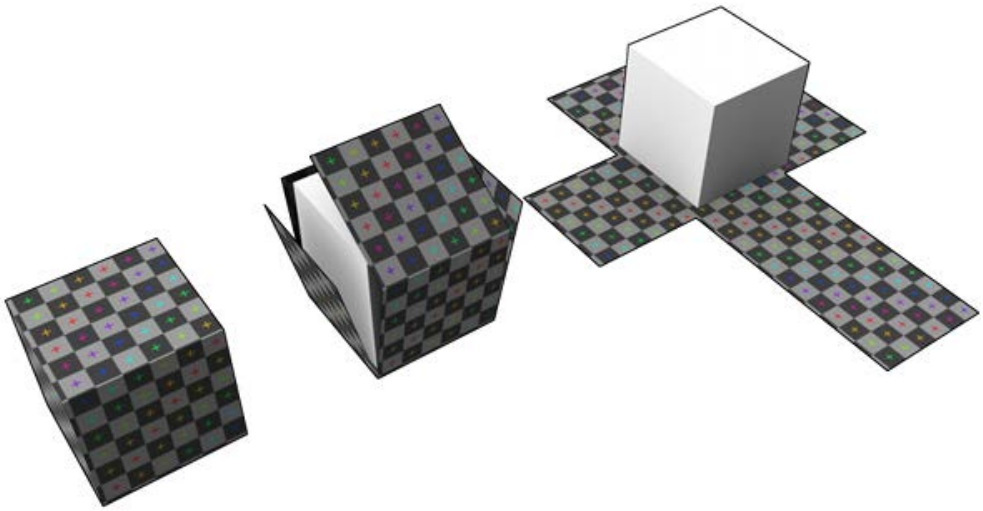


Imagen 58: Despliegue de las caras de un cubo, UV Mapping

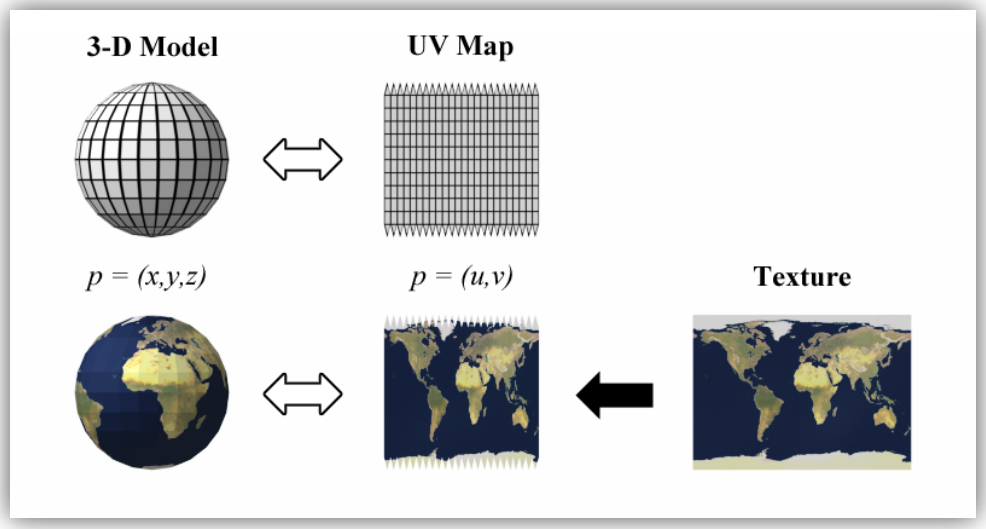


Imagen 59: UV Mapping aplicado a una bola del mundo. Imagen extraída de www.upload.wikimedia.org

Puede verse en la imagen 60 como un punto P (vértice de un polígono) del objeto 3D es mapeado en una imagen 2D con sus respectivas coordenadas 2D. Ésta conversión se realizará para absolutamente todos los puntos del objeto 3D al que se le aplicará el UV Mapping.

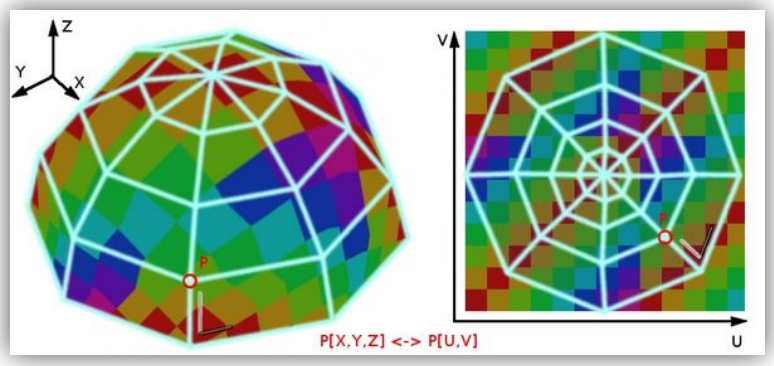


Imagen 60: Representación de un punto P en el objeto 3D y el punto P sobre la imagen de éste en 2D. Imagen obtenida de wiki.blender.org

- Aplicación del UV Mapping sobre un objeto

Teniendo ya la textura preparada tocará aplicarla al objeto y para ello se tendrán que realizar unos pequeños pasos.

En primer lugar, tal y como se aprecia en la imagen 61, se deberá seleccionar el objeto y realizar el "Unwrap" (pulsando la tecla U). Acción que desenvolverá la figura 3D en un plano 2D (UV map) y permitiendo texturizar ésta.

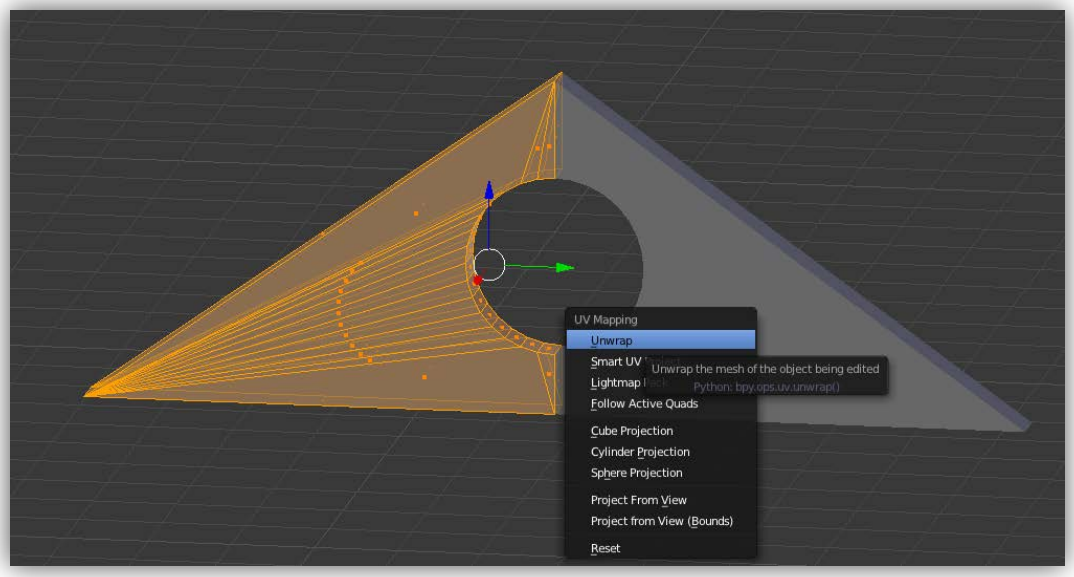


Imagen 61: Selección de la opción de Unwrap de la figura.

Nota: Únicamente se realiza el "Unwrap" sobre media figura debido a que la otra media parte es un mirror realizado previamente con un mirror modifier, por lo que cualquier modificación sobre de la parte original también afectará a su mirror.

Se mostrará por tanto la imagen siguiente donde la figura habrá sido desenvuelta en un UV Map de manera automática. A continuación, tal y como se muestra en la imagen 62, toca seleccionar la textura que se quiere aplicar sobre la imagen tal y como se puede apreciar en el recuadro .

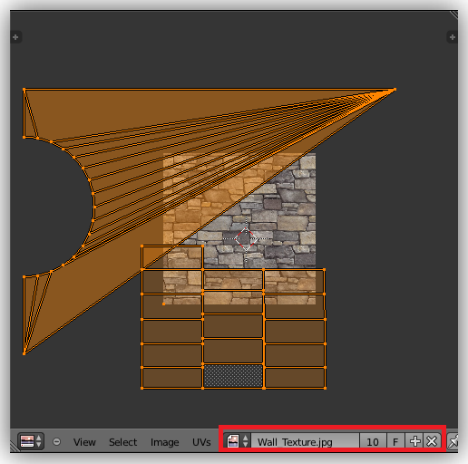


Imagen 62: Selección de la textura a aplicar sobre la imagen 2D del objeto

Y finalmente, véase en la imagen 63 el objeto con la textura aplicada siguiendo todo el proceso explicado anteriormente.

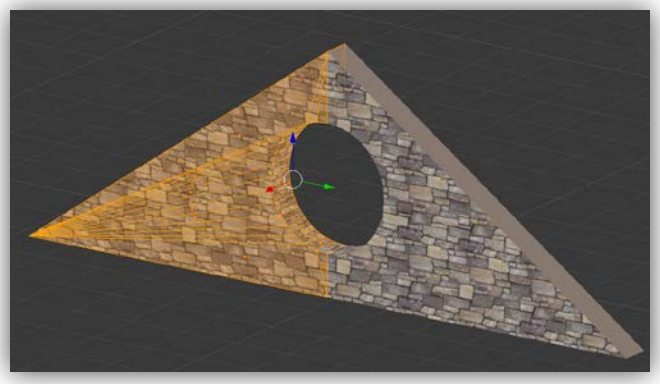


Imagen 63: Parte frontal de la iglesia texturizada

Opciones adicionales para el UV Mapping

Podría haberse seleccionado la opción "Smart UV Project". Ésta opción examina la forma del objeto en cuestión, las caras seleccionadas, las relaciones de éstas con las demás y crea un UV Map en función de los datos proporcionados. En la imagen 64 puede verse el panel de aplicación de la opción "Smart UV Project", el cual contiene algunas opciones que pueden ser modificables.

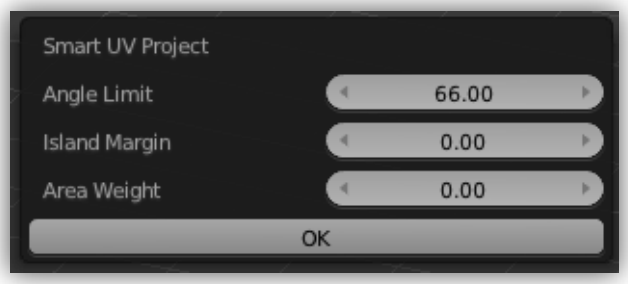
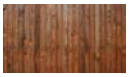


Imagen 64: Panel aplicación del Smart UV Project sobre un objeto

El proceso de texturización es un proceso idéntico para todo el resto de objetos que forman la iglesia. Es por ello que se deberá seguir los pasos descritos anteriormente cada vez que se desee aplicar una textura a un objeto, con a la diferencia de que si utilizan una misma textura se podrá coger ya el material creado con esa textura previamente.

En la siguiente tabla se muestra las distintas texturas utilizadas en la iglesia:

Textura	Uso
	Paredes interiores y exteriores, partes laterales, frontales y traseras de las almenas
	Cuerpo de las columnas
	Parte superior de las almenas, soportes generales de las columnas, arcos interiores de la iglesia , lateral escaleras interiores y pared de soporte del segundo piso

	Soporte de las vigas del techo
	Suelo interior de la iglesia
	Pequeñas puertas de madera situadas en el segundo piso
	Marco exterior e interior de la puerta, peldaños de las escaleras del interior de la iglesia, soportes superiores e inferiores de las columnas del interior de la iglesia
	Techo de la iglesia
	Soportes superiores e inferiores de las columnas del exterior de la iglesia
	Suelo de la segunda planta
	Soportes interiores de la iglesia
	Puerta principal de la iglesia

- 4.2.1.3: Añadido de elementos internos (antorchas)

En el interior de la iglesia se colocaron antorchas, únicamente se colocó el cuerpo de ésta porque las llamas serían realizadas mediante la técnica del particle system, un componente de Unity3D que se explicará más adelante. Se colocaron varias antorchas dentro de la iglesia con la intención de que iluminen el interior de ésta. Se utilizó el siguiente modelo de antorcha, descargado de la página http://www.blender-models.com .En la imagen 65 puede verse el modelo utilizado para las antorchas (izquierda) y el efecto de la aplicación del particle system sobre el cuerpo de la antorcha (derecha).

Imagen 65: Cuerpo de la antorcha (izquierda) y efecto del particle system (derecha)

En la imagen 66 se muestra la distribución realizada de las antorchas dentro de la iglesia. Hay distribuidas un total de doce antorchas, seis en cada lado de las paredes de la iglesia. Pueden verse identificadas en la imagen con el color amarillo anaranjado.

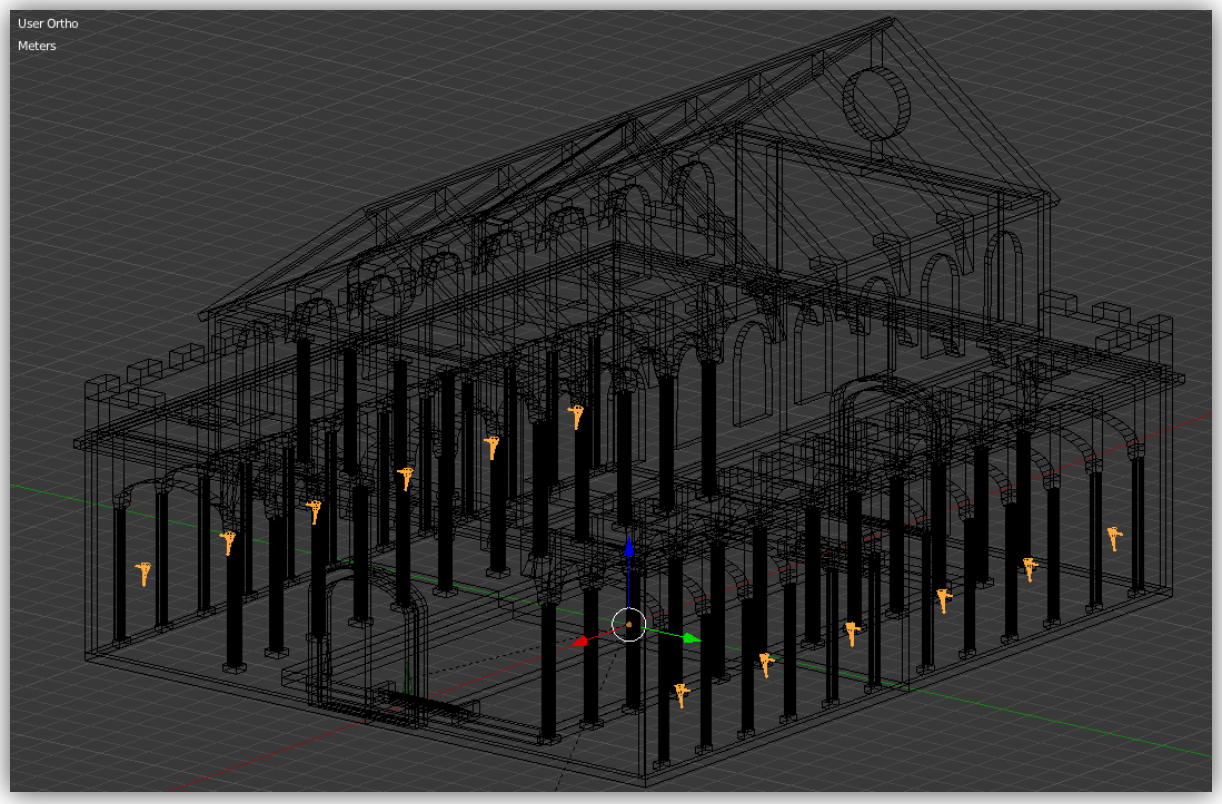


Imagen 66: Antorchas distribuidas por el interior de la iglesia

- 4.2.1.4: Entrada y salida de la iglesia

La iglesia dispone de una entrada (parte delantera) y de una salida (parte trasera). La entrada se ha realizado a partir de una puerta de madera, está dividida en dos partes ya que posee una animación de apertura de puerta cuando el personaje principal se acerca a ésta y una animación de cierre cuando el personaje principal se aleja.

La puerta trasera en cambio, dispone de una barrera de fuego que estará activa y no podrá ser atravesada mientras el jefe final permanezca con vida. Éste fuego también se ha realizado con la técnica de particle system que se explica más adelante. En la imagen 67 puede verse una imagen de cada una de las puertas de la iglesia, a la izquierda la puerta delantera y a la derecha la puerta trasera.

Imagen 67: Puerta delantera de la iglesia (izquierda) y puerta trasera (derecha)

- 4.2.1.5: Exportación a Unity3D

Imagen 68: Panel de exportación de objetos a formato .fbx

La exportación a Unity3D se realiza mediante el formato *.fbx*, un formato propio del programa de modelaje *Autodesk* que da muy buenos resultados en este *engine*.

Se deberá entonces seleccionar todo el objeto a exportar 

, a continuación *File-> Export -> Autodesk FBX*. Seguidamente, como puede verse en la imagen 68 que contiene el panel de exportación a .fbx que se abrirá al realizar la acción anterior, se comprueba el apartado de "Export FBX", es muy importante comprobar que se tengan los siguientes campos activos: *Selected Objects, ApplyModifiers, IncludeAnimation* y *All Actions*. Las demás opciones vienen predefinidas por Blender y no es necesario realizar ninguna acción sobre ellas.

- 4.2.2: Personaje principal

- 4.2.2.1: Creación del mesh & 4.2.2.2: Texturización

Imagen 69: Modelo utilizado en el juego. Imagen obtenida de www.smopic.com

El personaje principal que se decidió utilizar puede verse en la imagen 69. Fue un personaje descargado de la página: www.smopic.com . El hecho de utilizar un personaje que ya había sido modelado y texturizado presentó una serie de problemas con los cuales hubo que tratar a la hora de querer utilizarlo en el juego.

Éste modelo originalmente contenía cerca de 200.000 polígonos, una cifra que supera con creces a las cifras de polígonos utilizadas en la mayoría de juegos. Tener tantos polígonos implica un mayor consumo de recursos y no tiene por qué ser más vistoso a nuestros ojos en el momento de jugar con el personaje. De hecho muchos juegos de hoy en día están utilizando modelos que están en torno a los 20000 – 40000 polígonos y gracias al efecto de la posterior texturización pueden conseguirse efectos visuales del personaje y de los objetos impresionantes y muy realistas.

Además la utilización de modelos con un nivel bajo de polígonos permite al engine una menor cantidad de procesamiento de datos y ello conlleva a un aumento del rendimiento del juego. Es decir, el tener que gestionar una gran cantidad de polígonos únicamente nos causará una bajada de imágenes por segundo (fps) a la hora de ejecutar el juego.

Además Unity3D tiene un límite de 65.000 polígonos para un único mesh, y esto nos obliga a separar el objeto en dos sub-objetos. Esto genera aún más perdida de rendimiento.

Y debido a que el personaje principal contenía prácticamente diez veces más polígonos de lo habitual se tuvo que buscar alguna manera de reducir éstos sin que afectase en gran medida al modelado y sin que afectase a la posterior animación.

Se optó por utilizar un modifier. Anteriormente ya se habló de modifiers para la creación de la iglesia en el apartado 4.2.1.1. A continuación se introduce uno nuevo que será utilizado para la reducción de los polígonos de mesh. Éste modifier es llamado Decimate.

- Decimate Modifier

Éste modificador permite reducir los vértices o caras de un mesh con un cambio mínimo en la forma de éste. Éste modificador proporciona una forma muy rápida de reducción de los polígonos que forman un mesh. En la imagen 70 puede verse el efecto de la aplicación del decímate modifier sobre un cilindro. A la izquierda se muestra el cilindro original y a la derecha el estado final después de aplicarle el modificador.

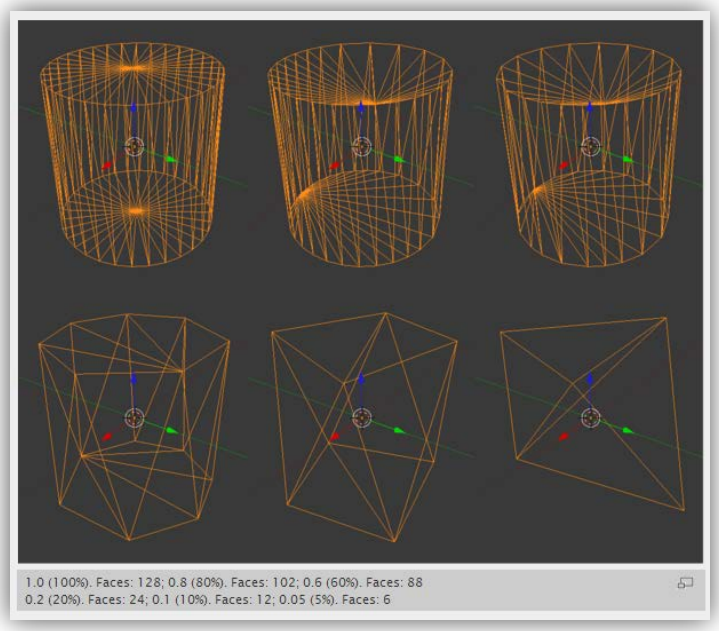


Imagen 70: Reducción de polígonos de un cilindro utilizando el Decimate modifier

En la imagen 71 se muestra el panel de aplicación del Decimate modifier con todas sus opciones.

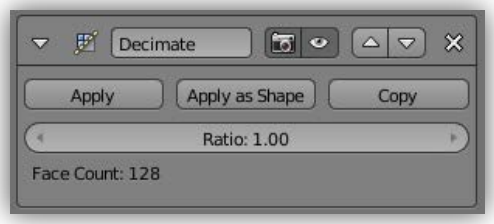


Imagen 71: Panel de aplicación del Decimate modifier

1- **Ratio:** Ratio de caras que se quiere mantener después de haber aplicado el modifier. Oscila entre 0 y 1 (0 – no queda ninguna cara, 1 – total de caras del actual mesh)
2- **Face Count:** Cantidad de caras actual del mesh.

- Reduciendo los polígonos del personaje principal

Se le añade el Decimate modifier y va reduciéndose el ratio hasta conseguir el resultado esperado.

En la siguiente tabla se muestran los resultados aplicación de distintos ratios al modificador y al reducción de polígonos.

Ratio	Face count (polígonos)
1 (original)	194.744
0.9	181.885
0.8	168.358
0.7	155.295
0.6	142.136
0.5	128.909
0.4	110.760
0.3	89.969
0.2	64.722
0.1	35.255

Finalmente se decidió utilizar la opción del personaje reducido hasta el ratio de 0.3 puesto que era la opción que mantenía la forma del personaje sin grandes cambios visuales y también permitía seguir sin problemas la aplicación de un esqueleto al personaje para su posterior animación.

También si se hubiera podido se hubiera escogido la opción del ratio 0.2 y 0.1, pero debido a que ya se notaba la reducción de polígonos que causaban que el mesh tuviera una forma un tanto cuadriculada y además al intentar continuar con su posterior aplicación del esqueleto daba el siguiente error: "Bone Heat Weighting: failed to find solutions for one or more bones". Este error se daba cuando había algún vértice estaba suelto, es decir que no formaba parte de ningún polígono o cuando quedaba alguna parte del objeto separada del propio objeto.

Es por ello que finalmente se decidió utilizar el mesh con el ratio de 0.3 con cerca de 90.000 polígonos. Pero aún así seguía teniendo una gran cantidad de polígonos y decidió, por el bien de la posterior utilización del personaje en el engine, reducir estos aún más pero no había otra forma de hacerlo sino que eliminando elementos innecesarios del personaje que contuviera gran cantidad de polígonos. Es por ello que se eliminó el lazo trasero que, al contener contornos redondeados generaba una gran cantidad de polígonos. También se decidió eliminar algunos detalles de los guantes y el reborde de la zona del vestido de la parte del pecho que contenía gran cantidad de contornos redondeados al igual que la parte superior de las botas y el lazo de la cabeza.

En la imagen 72 puede verse el personaje principal desde diversas perspectivas en Blender. El personaje se le ha aplicado el Decimate modifier pero no se le han quitado los complementos.

Imagen 72: Personaje principal con los polígonos reducidos y con todos sus complementos

En la imagen 73 puede verse el personaje principal desde diversas perspectivas en Blender. El personaje se le ha aplicado el Decimate modifier y se le han quitado los complementos citados anteriormente.

Imagen 73: Personaje principal con los polígonos reducidos y con los complementos quitados

Con todo se consiguió un decremento considerable de polígonos, cerca de 20.000 menos situando la figura con un total de 61.741 polígonos, menos de los 65.000 que el Unity3D posee como límite para un único mesh. En la imagen 74 se muestra el estado final del personaje en Blender que será utilizado.

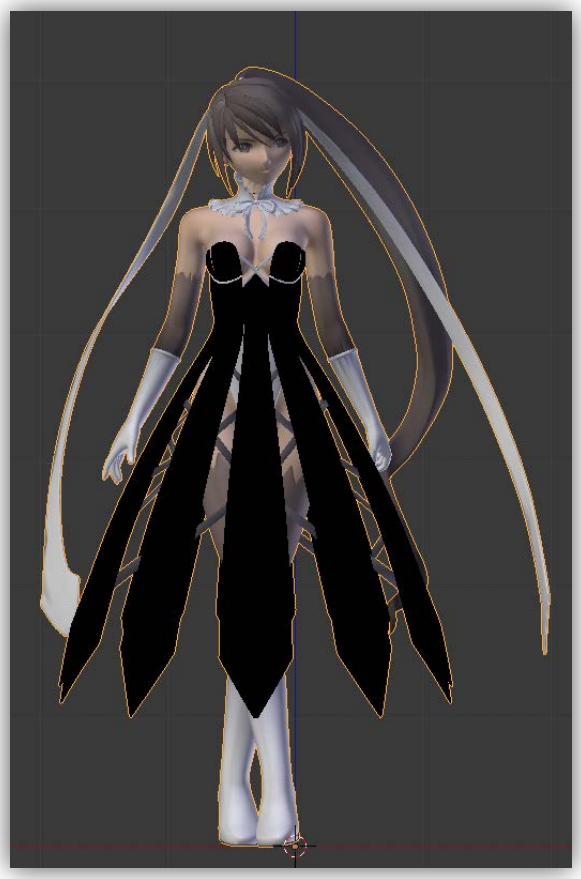


Imagen 74: Estado final del personaje principal antes de ser animado

- 4.2.2.3: Creación y aplicación del esqueleto (Rigging)

-Creación del esqueleto (*armature*)

Todos y cada uno de los objetos a los que posteriormente se les dé animaciones, se les deberá aplicar un esqueleto (armature).

El armature tiene la finalidad de establecer todas las partes del personaje que dispondrán de movimiento y funciona como un esqueleto de una persona en la vida real. Allí donde haya una articulación o un hueso, ésta dispondrá de una rotación que nos permitirá mover las partes del objeto. Un armature está formado por muchos huesos (bones). En la imagen 75, en la parte izquierda aparece el armature tal y como aparece al utilizar el add-on descargado previamente y en la parte derecha aparece el armature modificado y adaptado al cuerpo del personaje principal.

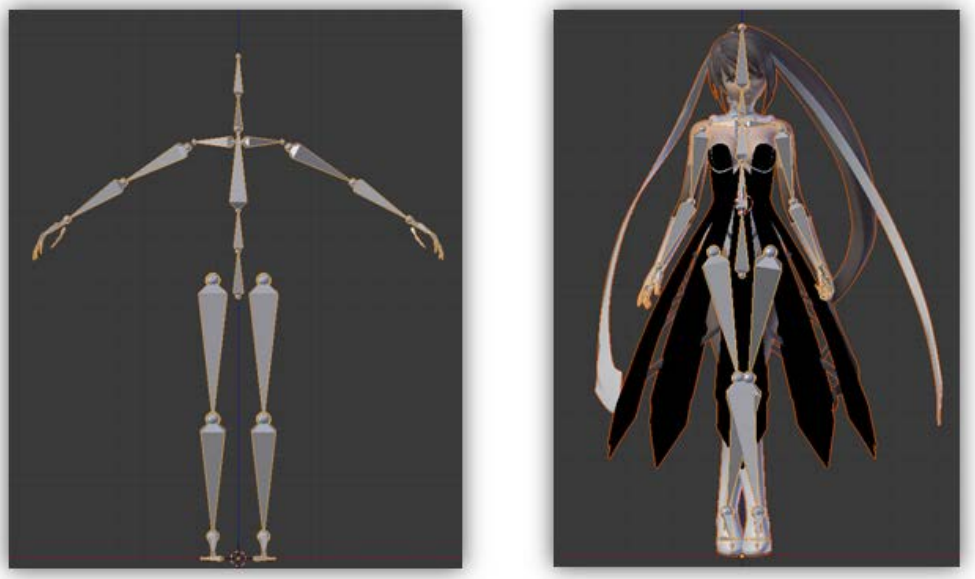


Imagen 75: Armature del add-on descargado (izquierda) y armature adaptado al cuerpo del personaje principal (derecha)

Para el armature que se le ha colocado al personaje principal se ha utilizado un armature predefinido que viene en un add-on de Blender y que ha sido posteriormente modificado para encajar perfectamente con el mesh del personaje. Puede verse a continuación.

-Generación la estructura *rigify*

Al seleccionar el armature creado, se mostrará la ventana de propiedades de éste con distintas opciones de las cuales escogemos el armature (tal y como puede verse en la imagen 76).

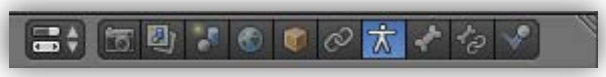


Imagen 76: Selección del armature en la ventana de propiedades

Se navega hasta la opción de "Rigify Buttons" para poder generar el rigify (rig) y se presiona sobre "Generate" tal y como se muestra en la imagen 77.

Imagen 77: Botón de generación del rigify

En este momento se obtendrá el rigify del personaje principal. Véase en la imagen 78 el rigify obtenido del armature del personaje. A la izquierda se muestra el rigify seleccionado con el personaje en modo WireFrame (donde únicamente se muestran los polígonos de éste) mientras que a la derecha puede visualizarse el rigify seleccionado con el personaje principal en su estado natural.

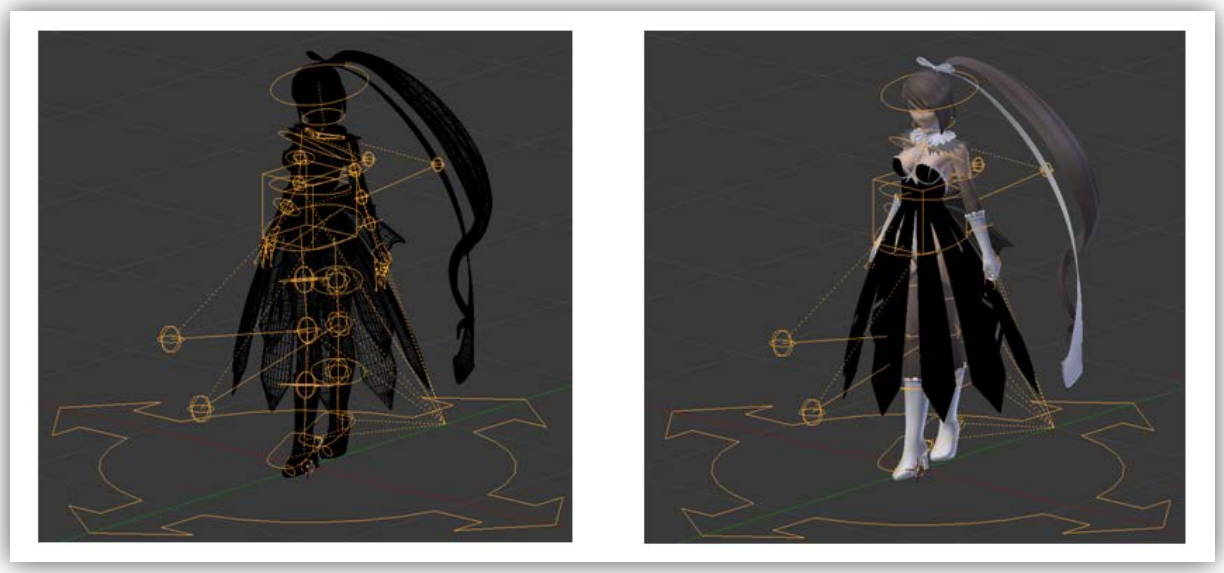


Imagen 78: Rigify obtenido del armature del personaje. Personaje en modo WireFrame (izquierda) y personaje en su estado natural (derecha)

-Unión del esqueleto con el mesh (*skinning*)

El siguiente paso se denomina skinning, un método de unión que asocia el esqueleto con las distintas partes del objeto. Para poder realizar éste paso se ha de seleccionar el mesh, a continuación el esqueleto y hay que presionar *Cntrl* + P -> *Automatic weight painting**.

Una vez realizada esta acción Blender automáticamente irá asociando a cada hueso con la parte que le corresponde del mesh (se asociará a cada hueso un conjunto de vértices del mesh que se moverán al efectuar un movimiento con este hueso).

Una alternativa al automatic weight painting sería el realizar manualmente el weight painting completo de cada hueso con cada vértice del mesh, una tarea muy tediosa y que consumiría muchísimo tiempo.

- 4.2.2.4 – Weight painting (pintado del cuerpo)

Primeros pasos

Al haber aplicado el skinning, Blender habrá realizado un pintado del cuerpo automático (automatic weight painting) que posiblemente, debido a la complejidad del mesh no habrá tenido un buen resultado en todas las zonas pintadas. Es por ello que ahora se deberá comprobar cada uno de los pesos que se le ha asociado a cada zona y modificar ésta si fuera necesario. Éste es uno de los pasos en los cuales se invertirá más tiempo. Un incorrecto weight painting ocasionará problemas a la hora de animar el personaje. Es por ello que en este paso conviene dedicarle su debido tiempo.

En la imagen 79 se mueve el personaje principal y puede verse que hubo errores con el automatic weight painting que deberán ser solucionados. Al mover el personaje para una dirección puede verse como muchos vértices que forman éste no efectúan ningún tipo de movimiento, es decir que no están influenciados por ningún hueso y por esa razón se quedan en el mismo lugar de siempre.

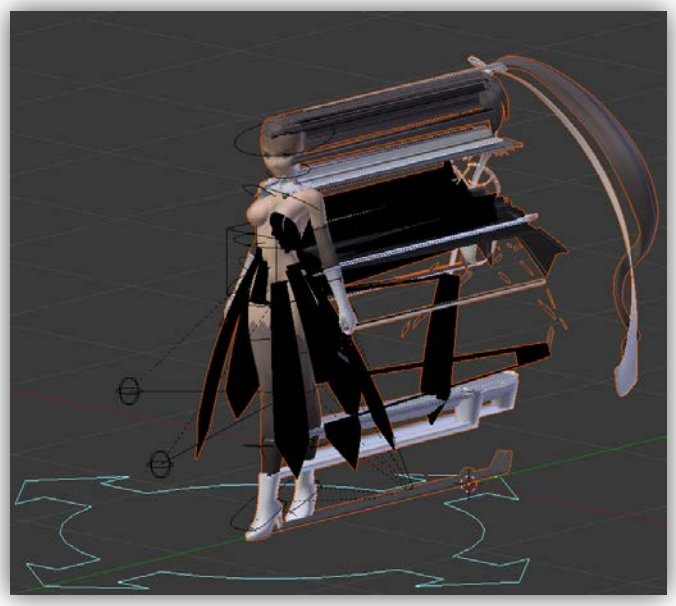


Imagen 79: Distribución incorrecta del weight painting en el personaje principal

A continuación veremos cómo realizar un ajuste de pesos, para ello se debe recorrer cada hueso del mesh y comprobar que el nivel de influencia del hueso sobre los vértices de los polígonos del personaje principal es también la correcta. Para ello, tal y como se muestra en la figura 80, se debe seleccionar el objeto y a continuación seleccionar el modo Weight Paint.

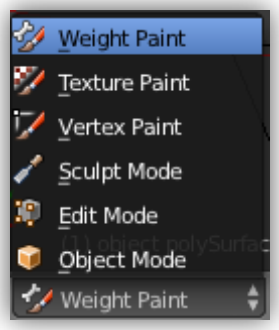


Imagen 80: Modo Weight Paint del objeto

Será en este momento que el mesh del cuerpo cambiará de color a un tono azulado y la aparición de un panel en la parte izquierda de la pantalla, síntoma que se está en el modo

Weight Paint. El panel que aparecerá puede verse en la imagen 81. Éste panel dispondrá de los modos de pintado que se deberá escoger.

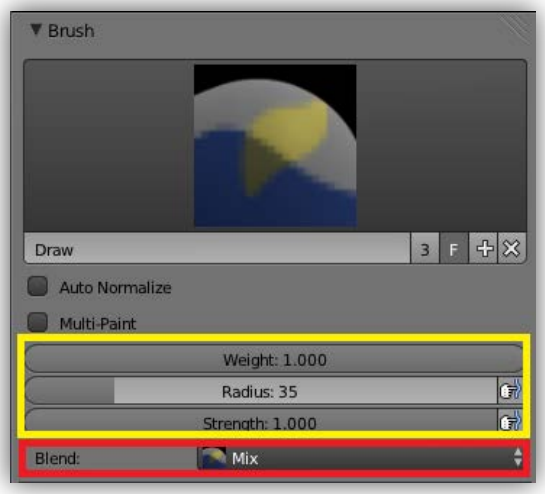


Imagen 81: Panel de aplicación del modo Weight Paint

Tal y como aparece en la imagen 81 en el recuadro permite escoger las opciones de personalización del pincel que se utilizará para realizar el pintado, mientras que el recuadro permite escoger las diversas opciones de pintado. En la imagen 82 se muestran todas estas opciones de pintado de las cuales únicamente suelen utilizarse dos: "Add" y "Substract".

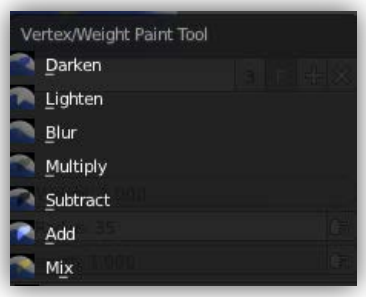


Imagen 82: Opciones de pintado disponibles

- **Add:** El nuevo color es añadido al más antiguo.
- **Substract:** El nuevo color es substraído del existente.

En la imagen 83 se muestra la tabla de influencias de los colores de los huesos sobre el mesh. El color azul implica una influencia del 0% en la zona del mesh sobre el hueso señalado en azul clarito (véase la imagen 84), mientras que por el contrario el color rojo implica una influencia del 100%.

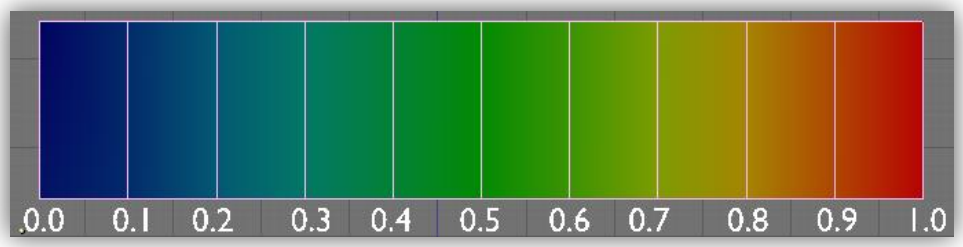


Imagen 83: Distribución de colores en función del nivel de influencia de estos

Weight painting

Ahora toca seleccionar los distintos huesos a comprobar, se empezará por la cabeza. En la imagen 84 se muestra la cabeza del personaje con el hueso de ésta seleccionado (azul clarito). A la izquierda está la cabeza originalmente pintada por el programa al realizar el automatic weight painting, mientras que a la derecha podemos observar la cabeza con la influencia de pesos que han sido modificados.

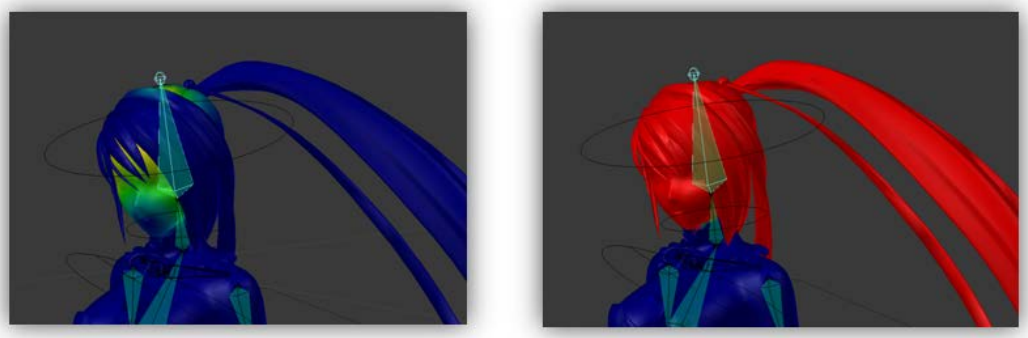


Imagen 84: Cabeza del personaje principal. A la izquierda el resultado del automatic weight painting y a la derecha la modificación realizada.

A continuación, para las imágenes 85, 86, 87, 88, 89 y 90 se irán mostrando las distribuciones de pesos que tuvieron que ser modificadas. A la izquierda podrá verse la distribución obtenida por el automatic weight painting, mientras que a la derecha podrá verse la modificación realizada.

Pecho

La imagen 85 representa la influencia del hueso del pecho del personaje principal. Con el automatic weight painting, al personaje le fue asignado una influencia al pecho muy escasa y únicamente a una parte de éste. Se tuvo que dar una influencia mucho mayor a los pechos, a la parte superior del corsé y al lazo que rodea el cuello del personaje.

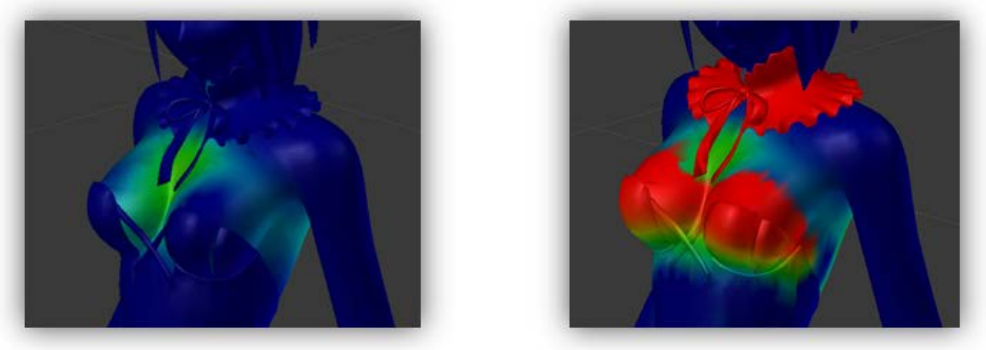


Imagen 85: Distribución de la parte del pecho

Abdomen

La imagen 86 representa la influencia del hueso de la cintura del personaje principal. Con el automatic weight painting, al personaje le fue asignado una influencia a la cintura del personaje correcta pero no asignó ningún tipo de influencia sobre el traje de esta. Se tuvo que asignar entonces una pequeña influencia sobre esta.

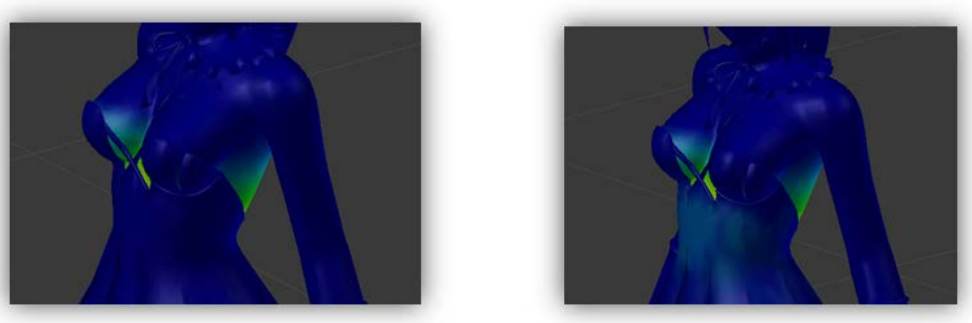


Imagen 86: Distribución de la parte del abdomen

Caderas

La imagen 87 representa la influencia del hueso de las caderas del personaje. Para éste hueso se tuvo que dar mucha más influencia sobre las caderas del personaje de la que había sido asignada previamente por el automatic weight painting.

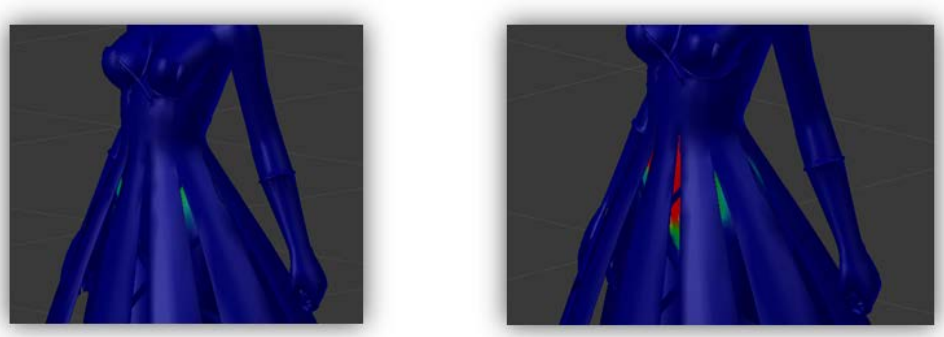


Imagen 87: Distribución de las caderas

Botas

La imagen 88 representa la influencia del hueso de la espinilla del personaje. Para éste hueso se decidió dar una influencia con un rango de afectación mucho mayor, llegando a afectar a toda la bota entera.

Imagen 88: Distribución de las botas

Pies

La imagen 89 representa la influencia del hueso del tobillo del personaje. Para éste hueso se decidió también dar una influencia sobre éste con un rango de afectación mucho mayor, cubriendo así toda la parte del pie.

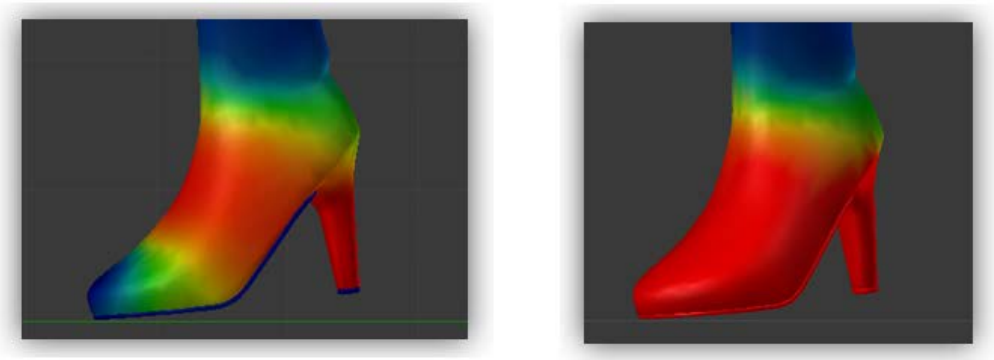


Imagen 89: Distribución de los pies

Hombros

La imagen 90 representa la influencia del hueso del húmero del personaje. Para éste hueso se decidió variar de manera sustancial la influencia que había sido asignada previamente por el automatic weight painting. Se decidió quitar toda la afectación mínima de alrededor del hombro debido a que al mover éste el personaje principal sufría unos estiramientos de piel inusuales.

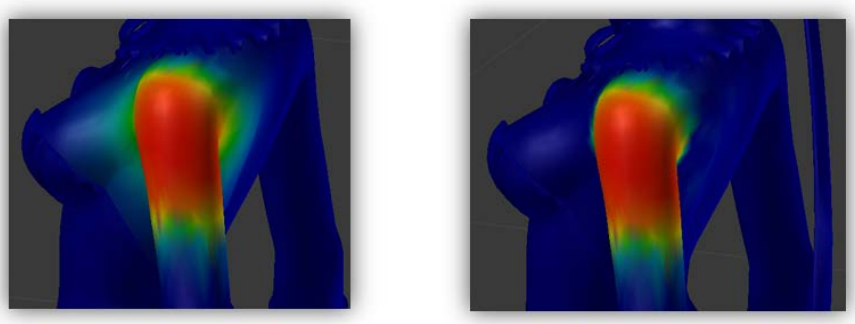


Imagen 90: Distribución de los hombros

A continuación, en la imagen 91 se exponen varias imágenes de las partes que no tuvieron que ser modificadas debido a que el peso dado por el automatic weight painting sobre cada uno de los otros huesos se consideró ya el correcto.

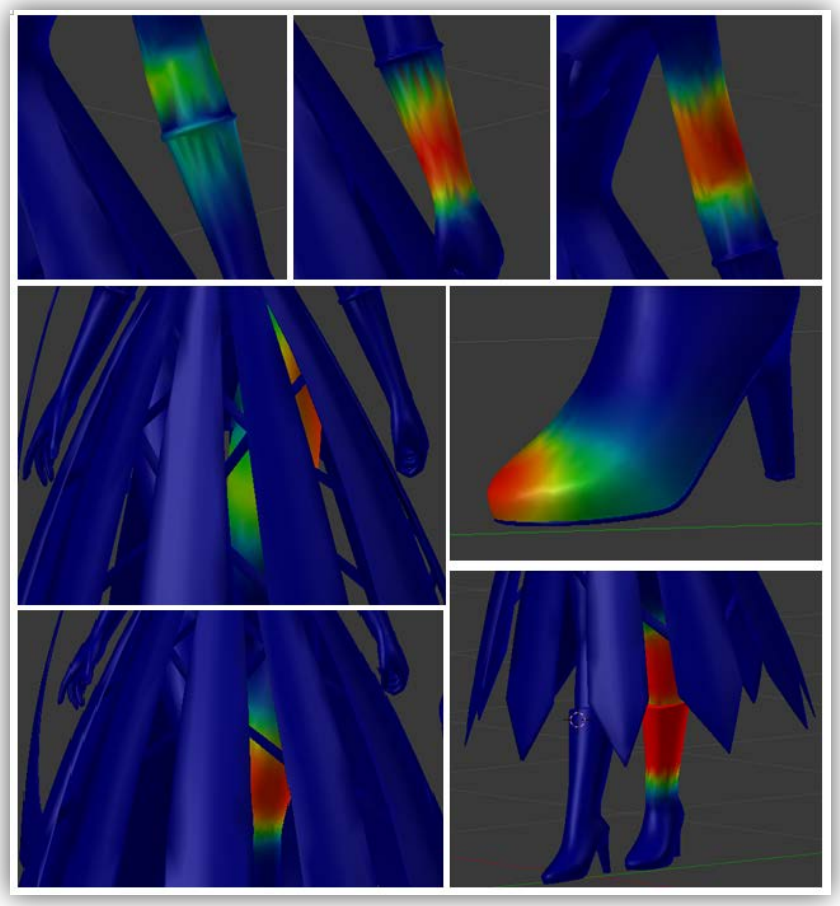


Imagen 91: Distribución de pesos del resto de las partes del personaje principal que no tuvieron que ser modificados

- 4.2.2.5 – Creación de las animaciones

Absolutamente todas las animaciones que serán creadas, se realizarán con el programa Blender. El personaje principal dispondrá de las acciones y movimientos siguientes:

Movimientos básicos

- **Parado (Idle):** Acción ejecutada cuando el personaje principal se encuentra parado (sin ninguna entrada por teclado del usuario) siempre y cuando no esté en contacto con el agua del escenario principal.
- **Correr:** Acción ejecutada cuando el personaje avanza o retrocede siempre y cuando el modo de correr esté activado.
- **Caminar:** Acción ejecutada cuando el personaje avanza o retrocede siempre y cuando el modo de correr esté desactivado.
- **Movimiento lateral:** Acción ejecutada cuando el personaje se desplaza de manera horizontal.
- **Saltar:** Acción ejecutada cuando el personaje realiza un salto.
- **Nadar:** Acción ejecutada cuando el personaje entra en contacto con el agua y se mueve por ella.
- **Caer:** Acción ejecutada cuando el personaje no toca el suelo durante un periodo de tiempo determinado.
- **Morir:** Acción ejecutada cuando la vida del personaje llega a cero.

Ataques

- **Primer ataque físico:** Primer ataque del combo que será ejecutado.
- **Segundo ataque físico:** Segundo ataque del combo que será ejecutado una vez se haya ejecutado el primer ataque físico.
- **Tercer ataque físico:** Tercer ataque del combo que será ejecutado una vez se haya ejecutado el segundo ataque físico.

- **Cuarto ataque físico:** Cuarto ataque físico que será ejecutado una vez se haya ejecutado el tercer ataque físico.
- **Quito ataque físico:** Quito y último ataque físico que marcará el final del combo.

Pasos previos a la animación

Para la animación del personaje se deberá efectuar movimientos sobre su rig que previamente fue insertado en el apartado de rigify (apartado 4.2.2.3). Es por ello que hay que seleccionar ésta y cambiar al Pose Mode (modo de posición). La imagen 92 muestra el rig del personaje seleccionado en el Object Mode y la intención de cambiar el modo de éste a Pose Mode.

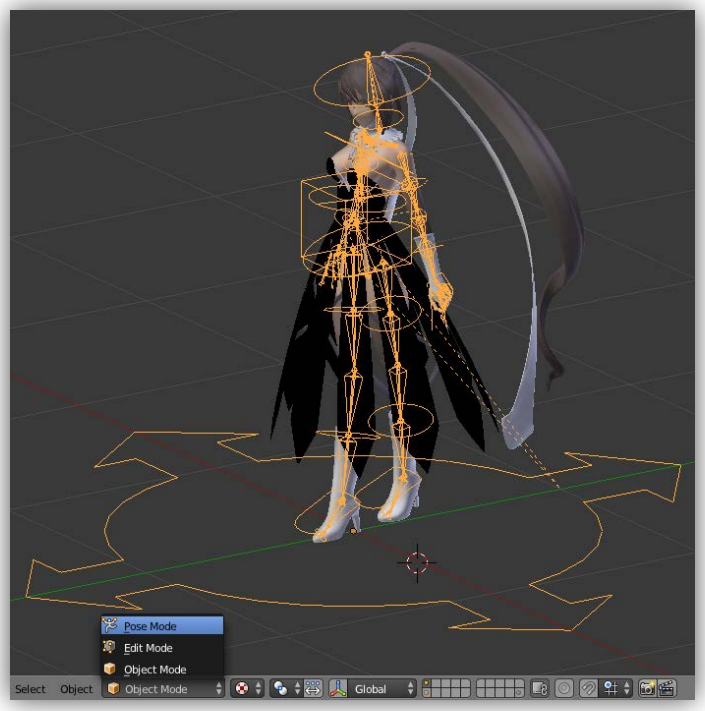


Imagen 92: Cambio al Pose Mode del personaje

Al seleccionar éste modo, el rig adquirirá un color azulado, indicando que está preparado para mover el mesh al mover cualquier parte del rig tal y como puede verse en la imagen 93.

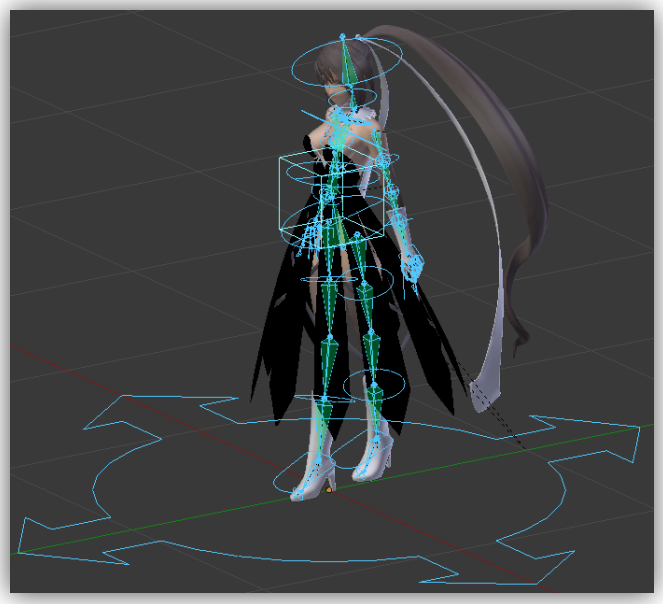


Imagen 93: Pose Mode del personaje activado

En estos momentos y con el personaje preparado para ser animado, se recomienda disponer el Blender con una distribución de la interfaz mostrada en la imagen 94, en la cual ésta interfície que permite el acceso a todos los elementos necesarios para la animación.

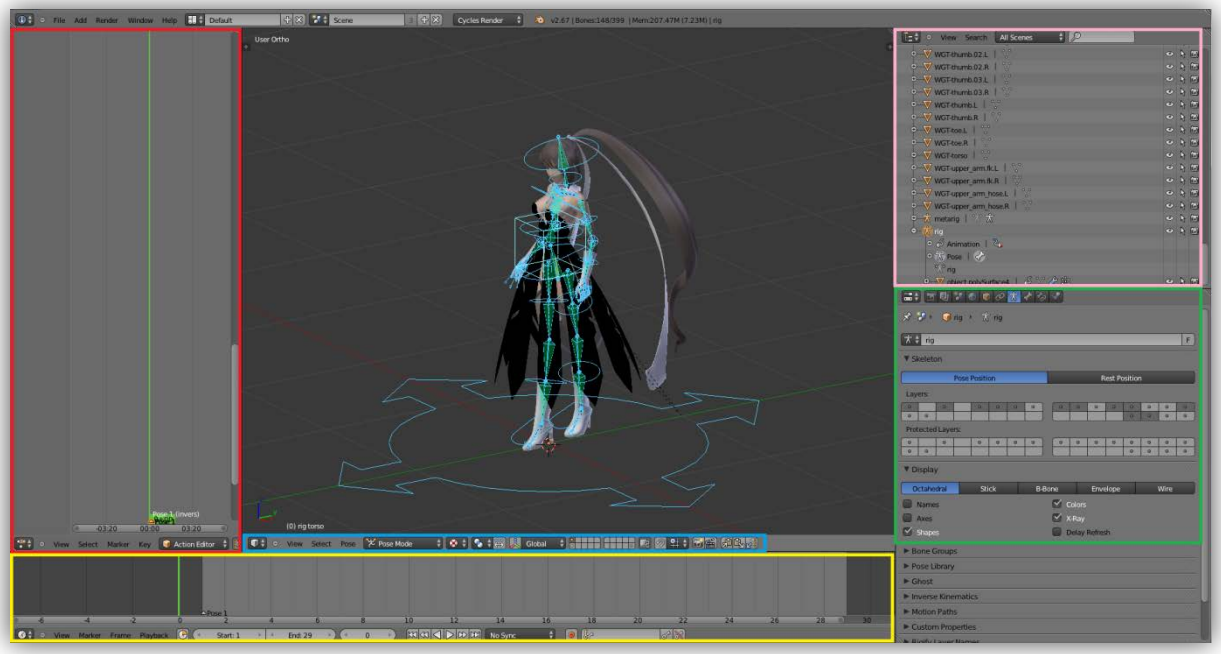


Imagen 94: Distribución de la interfície recomendada para la animación del objeto

Explicación de cada apartado:

DopeSheet: En este recuadro se señalan las posiciones de cada elemento que forma parte del rig en un instante de tiempo determinado (el conjunto de todas estas posiciones es llamada keyframe).

TimeLine: En este recuadro se muestra un rango de tiempo. Será en esta zona donde se definirán los distintos keyframes. También permitirá reproducir la animación.

OutLiner: Permite visualizar el rig que contiene el mesh entre otros.

Propierties: Muy importante el apartado Skeleton para controlar las *layers* (capas) que permitirán ocultar o mostrar las diferentes partes del rig. También contendrá el apartado Display que permitirá mostrar el nombre de los huesos, ejes, formas, colores, x-ray (para poder visualizar el rig completo) entre otros.

3D View: Desde aquí lo que más interesa para el apartado de animación serán estos iconos  . El primero de ellos copia todas los elementos que forman parte del rig seleccionadas (actúa como un portapapeles), el segundo pega el contenido de los elementos copiados en el primero y el tercero también pega el contenido de los elementos copiados en el primero pero de forma inversa (útil para muchas animaciones) .

A continuación se realizará la animación de correr paso a paso para que pueda verse el proceso de creación de ésta. Las demás animaciones se realizarían del mismo modo.

Proceso para crear una animación (Correr)

Hay que situarse en el **DopeSheet** y localizar el icono 

que permitirá introducir el nombre a una nueva animación. 

Nota: Es muy importante que el icono 

se marque para cada animación debido a que guarda el contenido de la ésta en un bloque de datos. De no hacerlo, al salir del Blender se perderán todas las animaciones en las no se haya activado la opción.

Ahora ya pueden empezarse a guardar los keyframes necesarios para la realización de la animación. Se ha decidido dividir la animación de correr en 5 pasos, es decir 5 keyframes que se deberán realizar con las distintas posiciones de correr del personaje. Entre un keyframe y otro, será el Blender el encargado de realizar el movimiento del rig de manera automática, es decir, no es necesario en cada instante de tiempo definir un keyframe a no ser que se prefiera modificar la trayectoria del rig creada por Blender ya que no ha sido la esperada. Esto podrá verse detalladamente más adelante.

Creación de los distintos keyframes

Comenzando con el **primer keyframe** hay que situarse en el **TimeLine** y marcar el instante de inicio de la animación, suele utilizarse el frame número 1 tal y como puede verse en la imagen 95, la barra del TimeLine.

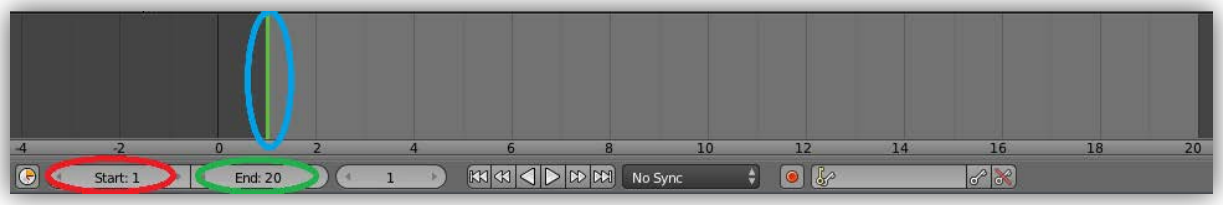


Imagen 95: Ventana del TimeLine de la animación

El  y  marcan el inicio y el fin de la animación respectivamente.

Ahora es el momento de definir la nueva posición del rig, y para ello se utilizará la ayuda de las vistas del Blender (apartado 4.1.1). Véase la imagen 96, el personaje principal al que se le ha definido una posición del ciclo de correr determinada que será asignada al keyframe 1. Se muestra éste desde dos vistas distintas para poder apreciar mejor la posición del personaje.

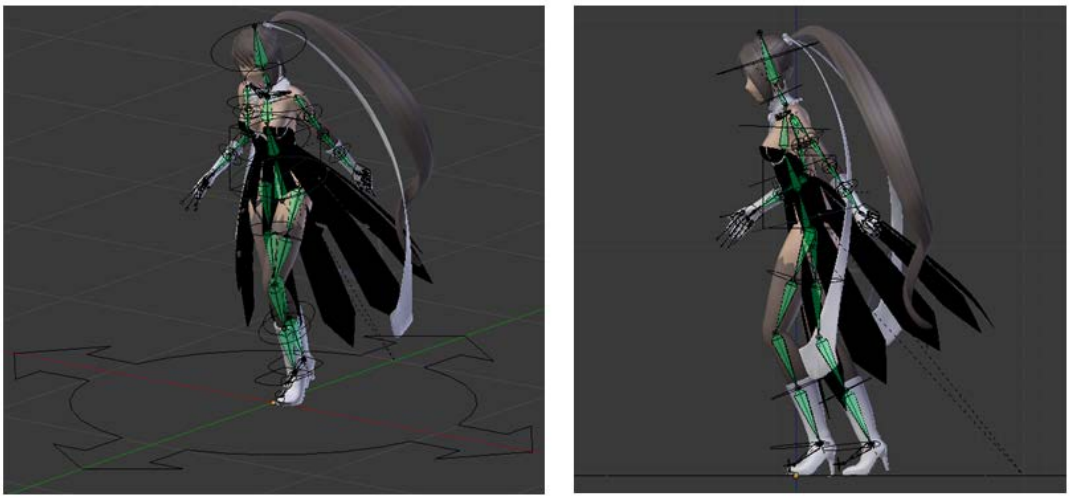


Imagen 96: Distintas vistas de una posición de correr del keyframe 1 del personaje principal

Una vez haya sido definida la nueva posición es **muy importante** guardar ésta ya que de no hacerlo se perderá. Para guardar la posición del rig y crear el primer keyframe se deben seleccionar todas las partes del rig que hayan sido modificadas (o simplemente seleccionar todo el rig presionando la tecla 

) .Seguidamente se presiona la tecla 

y aparecerá el menú mostrado en la imagen 97 con todas las opciones disponibles, donde se deberá escoger la opción deseada, dependiendo de lo que se quiera guardar sobre la nueva posición el rig, aunque lo más cómodo es simplemente guardar todo, la Location, Rotation y Scaling con la opción LocRotScale.

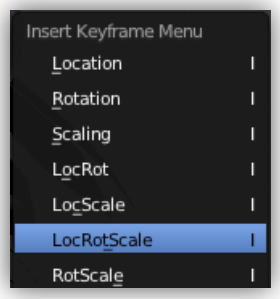


Imagen 97: Panel de opciones de guardado de las partes seleccionadas del objeto de la nueva posición

Nota: También se dispone de otra opción más comoda. En el **TimeLine** puede verse el siguiente icono . Si es pulsado, todos los movimientos del rig que se realicen mientras esté activado quedarán grabados, quitando la necesidad de tener que guardar la posición en cada keyframe del estado del rig.

En este momento aparecerá el nuevo keyframe. Tal y como se aprecia en la imagen 98, en el **DopeSheet** aparecerán todos los elementos del rig que hayan sido modificados con un punto en amarillo y en el **TimeLine** aparecerá una línea de color amarillo sobre el frame número 1 (frame sobre el que estábamos realizando la operación).

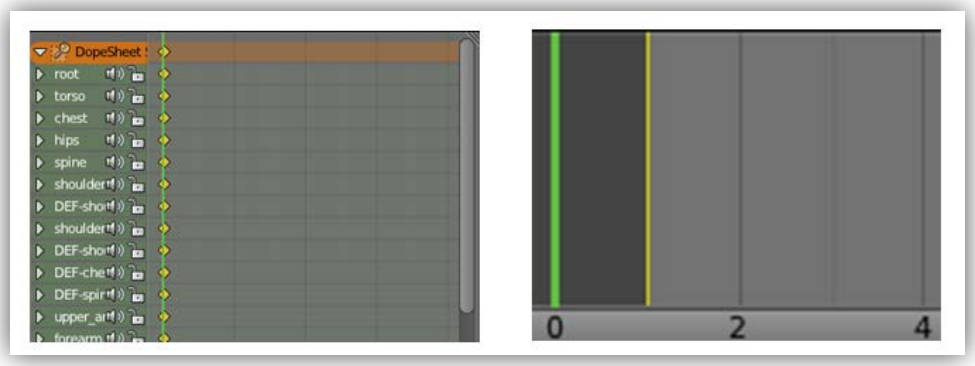


Imagen 98: A la izquierda el DopeSheet con los elementos del rig que hayan sido modificados y al a derecha la creación de un nuevo keyframe en el frame número 1

Antes de continuar y por comodidad se muestra como añadir un marker (marcador) de posición para poder identificar a ésta. Para ello se deberá seleccionar el keyframe deseado en este caso el keyframe del frame número 1 y realizar la siguiente acción de Add Marker tal y como puede verse en la imagen 99.

Imagen 99: Añadido de un marcador sobre el frame seleccionado

A continuación renombrar el nombre dado por defecto por Blender tal y como se puede apreciar en la imagen 100, donde se muestra a la izquierda el nombre del marcador dado por Blender (F_01) y a la derecha el nombre del marcador una vez cambiado (Pose1).

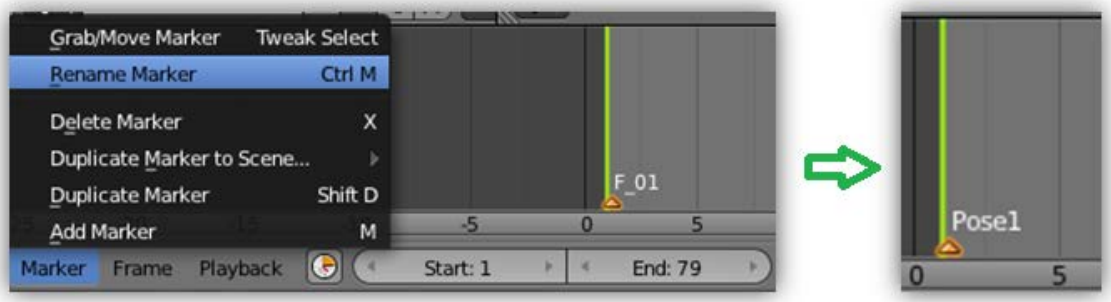


Imagen 100: Cambio del nombre del marcador del frame número 1

Siguiendo con el **segundo keyframe** se marcará la segunda posición del rig.

Nota: Hay que tener muy en cuenta que para realizar una correcta animación se ha de mantener la proporción de tiempo igual a la que se mantiene en la acción de correr en la vida real.

Éste frame se marcará en el frame 21, siguiendo la misma metodología que se ha seguido con el keyframe 1, una vez fijada la nueva posición, se realiza el guardado de ésta. Puede verse el resultado del keyframe 2 en la imagen 101.

Imagen 101: Distintas vistas de una posición de correr del keyframe 2 del personaje principal

Siguiendo con el **tercer keyframe** se marcará la tercera posición del rig. Ésta se marcará en el frame 41 y será el punto medio de la animación. Puede verse el resultado del keyframe 3 en la imagen 102.

Imagen 102: Distintas vistas de una posición de correr del keyframe 3 del personaje principal

Siguiendo con el **cuarto keyframe** se marcará la cuarta posición del rig en el frame 61. Éste frame será idéntico al segundo frame (frame 21) pero en su forma inversa. Puede verse el resultado del keyframe 4 en la imagen 103.

Imagen 103: Distintas vistas de una posición de correr del keyframe 4 del personaje principal

Y para acabar, el **quinto keyframe** se marcará la quinta posición del rig en el frame 81. Éste frame será idéntico al primer frame (frame 1).

En esta animación se ha dejado 20 frames de diferencia entre cada keyframe y se ha mantenido una proporción correcta para que la animación tuviera un aspecto lo más realista posible. La imagen 104 muestra la distribución de los cinco keyframes creados (palitos amarillos verticales) en la vista del TimeLine. A continuación se explicará cada uno de los recuadros seleccionados en la imagen.

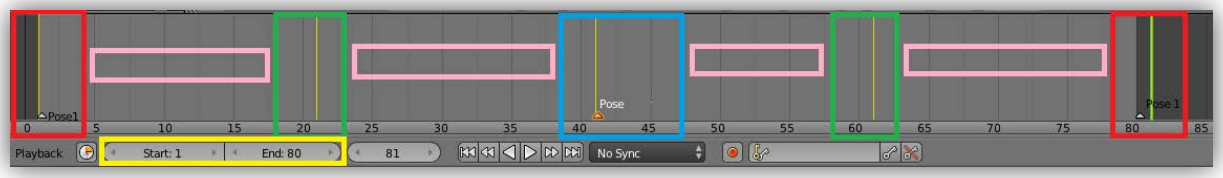


Imagen 104: Distribución de los cinco keyframes creados de la animación

 Animación utilizada en el inicio de la animación y el final de ésta.

 Animación intermedia de toda la acción.

 Éstas dos son animaciones inversas entre ellas.

 Durante todos estos frames, Blender gestionará para cada frame la posición del rig en función del keyframe previo y el siguiente keyframe definido.

(Por ejemplo, para gestionar el primer , partirá del keyframe contenido en el frame 1 hasta el keyframe contenido en el frame 21, comparará la diferencia de posición de los elementos del rig fijada en cada uno de los dos keyframes y trazará una trayectoria para cada uno de los elementos de éste).

 El tiempo Start define el inicio de la animación mientras que le tiempo End define el fin de la animación.

Como puede verse la animación empieza en el frame 1 y finaliza en el frame 80 a pesar de que el último keyframe (quinto keyframe) ha sido definido en el frame 81. Esto es debido a que el frame 1 corresponde al frame 81, y si se ejecutara la animación hasta éste último frame se podría visualizar una pequeña parada al final de ésta. Pero el frame 81 debe estar fijado ya que Blender, para poder gestionar todos los frames del último ha de conocer la posición del rig final  .

- 4.2.2.6 – Exportación a Unity3D

La exportación a Unity3D se realiza de la misma forma con la que se realizó anteriormente la iglesia (véase la imagen 68). El formato de exportación será .fbx y para poder realizar la exportación se deberá hacer lo siguiente: seleccionar todo el objeto a exportar  , a continuación *File -> Export -> Autodesk FBX*. Seguidamente se comprueba el apartado de "Export FBX" y se comprueba que todas las opciones de a continuación estén marcadas:

Selected Objects, Apply Modifiers, Include Animation, Include Default Take y *All Actions*. Las demás opciones vienen predefinidas por Blender y no es necesario realizar ninguna acción sobre ellas.

4.3 - Unity3D: Modelaje y Texturización

En esta sección se explicará el método de creación del escenario principal y el menú principal que fueron los dos objetos que creados en su totalidad utilizando el editor de Unity3D.

-4.3.1: Escenario principal (Mapa)

-4.3.1.1: Creación del cuerpo (Mesh)

En la creación del mapa se parte de un terreno principal llamado Terrain que se deberá añadir a partir de *File -> Terrain -> Create Terrain.* En la imagen 105 se muestra un Terrain en su estado inicial, antes de modificar ningún aspecto de éste.

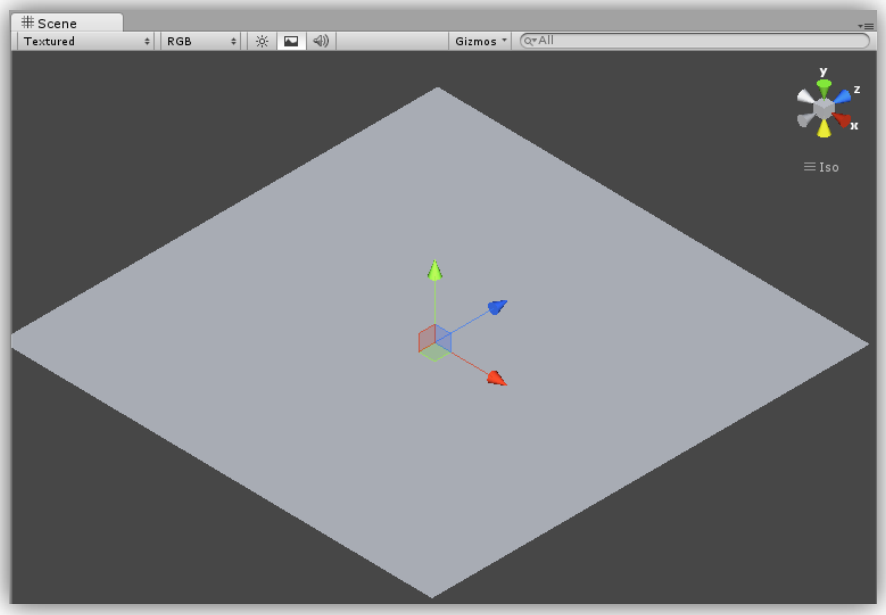


Imagen 105: Terrain en su estado inicial

Ésta será la base que será modificada utilizando las herramientas que posee Unity3D para la creación de terrenos puede verse a continuación. En la imagen 106 se muestran en el recuadro  las diversas opciones de modificación del Terrain.

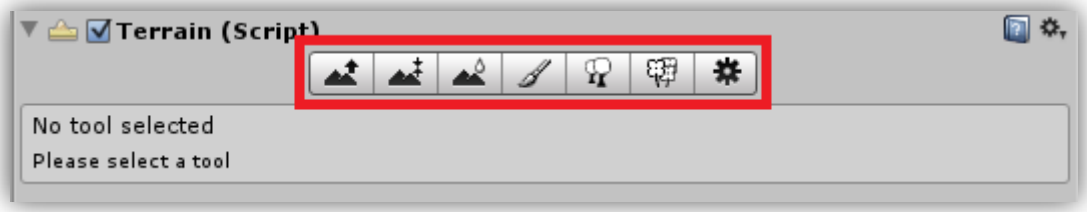


Imagen 106: Opciones de modificación del Terrain

A continuación se explicarán las diversas opciones vistas en el recuadro anterior.

 - Permite elevar el terreno.

 - Permite elevar o disminuir la altura del terreno a una altura dada en concreto.

 - Permite dar suavizar la zona a la que se le aplica.

 - Permite añadir texturas.

 - Permite añadir árboles de los cuales Unity3D ya dispone como prefabricados.

 - Permite añadir flores de las cuales Unity3D ya dispone como prefabricadas.

 - Permite modificar algunas otras opciones.

Utilizando todas las herramientas anteriores se construyen las montañas y pendientes de la zona. En imagen 107 puede verse la construcción de unas montañas sobre el Terrain utilizando el editor de Unity3D.

Nota: No pueden realizarse cuevas debido a que el editor de Unity3D aún no tiene implementada esta opción. Para poder realizar cuevas debe hacerse con un programa externo e importarlo a Unity3D.

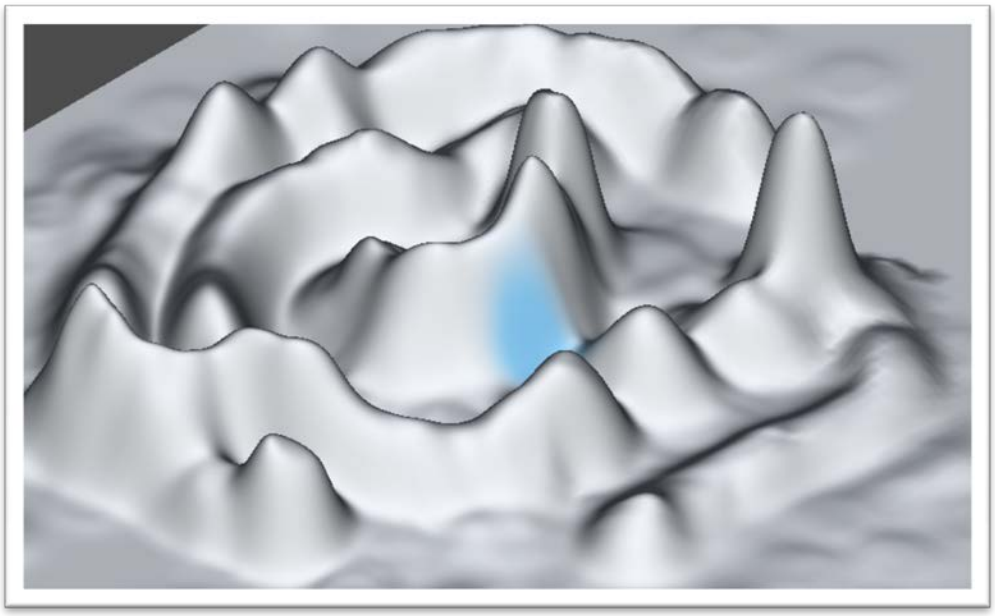


Imagen 107: Construcción de montañas sobre un Terrain

-4.3.1.2: Aplicación de las texturas (Texturización)

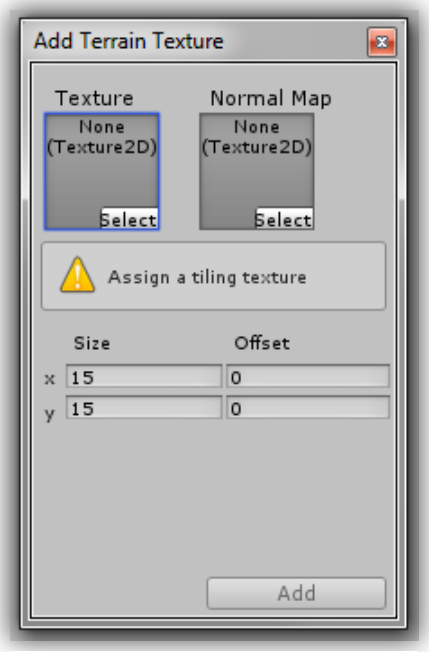


Imagen 108: Menú de elección del tipo de textura de Unity3D

Para la aplicación de texturas se marcará  y  para que salga la siguiente ventana. Ésta permitirá escoger Texture o Normal Map. Ésta es una característica que fue añadida en la versión 4 de Unity3D. Si Normal Map es utilizado, permite añadir al la zona deseada un material con unas propiedades específicas que darán un efecto de profundidad u otro a la textura. En el caso realizado, se ha utilizado Texture. Para poder aplicar la textura únicamente se tendrá que seleccionar la deseada y pintar la zona que se quiera. En éste caso se ha utilizado una textura de arena y otra de nieve dando el resultado siguiente.

En la imagen 108 puede verse el menú de la elección del tipo de textura a añadir. A continuación en la imagen 109 pueden visualizarse las montañas construidas anteriormente a las que se les ha añadido la textura de arena y de nieve.

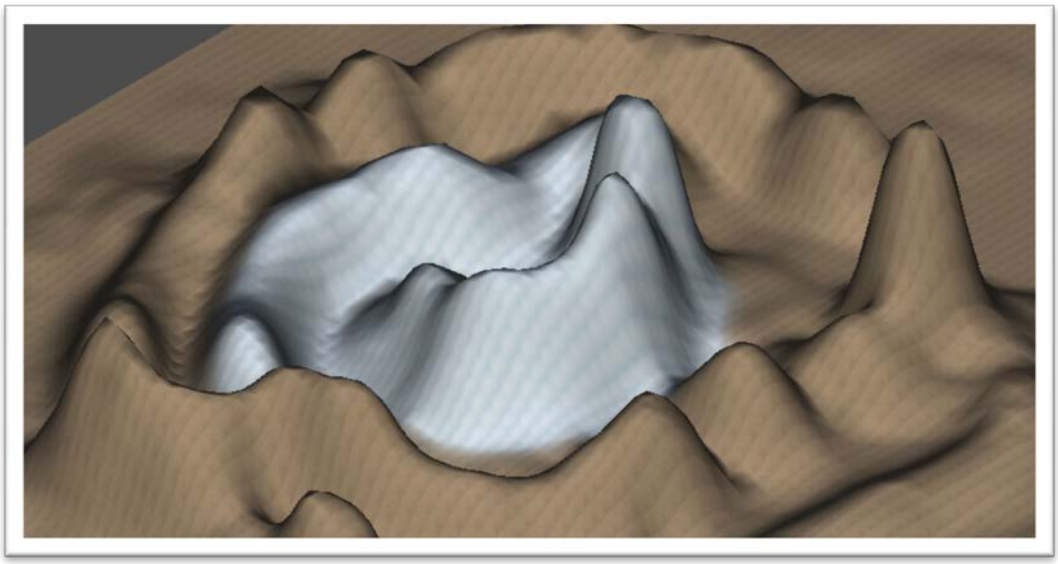


Imagen 109: Montañas creadas con textura de arena y nieve

Siguiendo la metodología de creación del Terrain con el editor de Unity3D, finalmente se obtuvo el escenario final que puede verse en la imagen 110, el cual está dividido por dos zonas separadas por un puente que está encima de un lago.

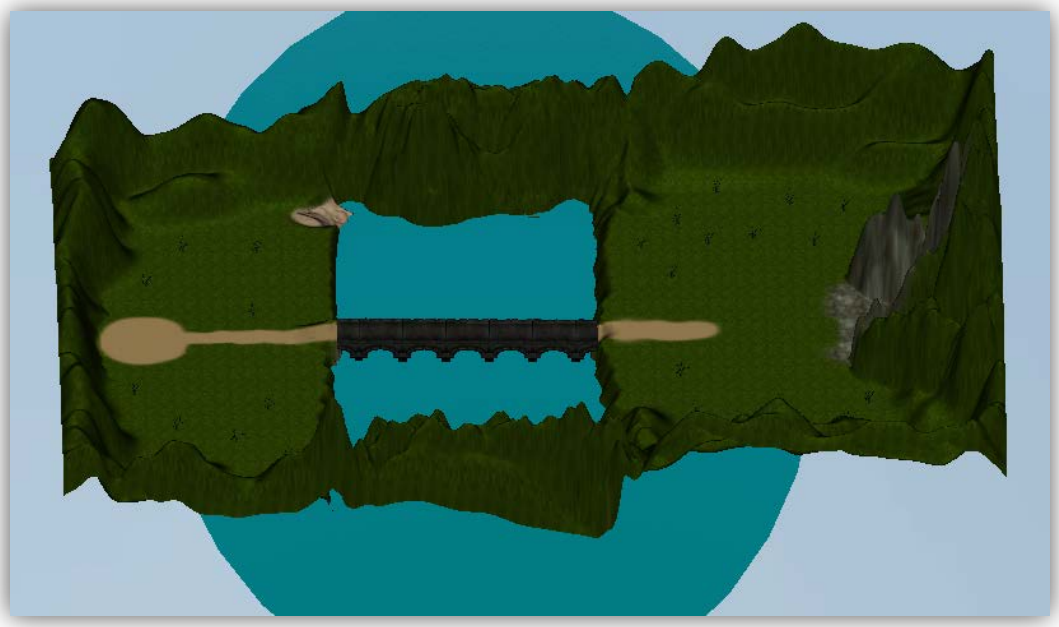


Imagen 110: Terrain modificado a su estado final del juego

-Creación del Skybox

Funcionamiento

Los skybox son un método de creación de fondos del escenario que tienen el efecto de dar más profundidad al escenario, pareciendo éste mucho más grande de lo que originalmente es. Esto se realiza mediante la incrustación de imágenes en el *background** dentro de un cubo.

Cuando se aplica el skybox, el escenario es encerrado en un cubo en el cual se proyectan en cada una de sus caras la parte del skybox que le corresponde. Véase la imagen 111 de a continuación donde se aprecia el despliegue de un cubo con sus caras recubiertas de imágenes que recrean un skybox.

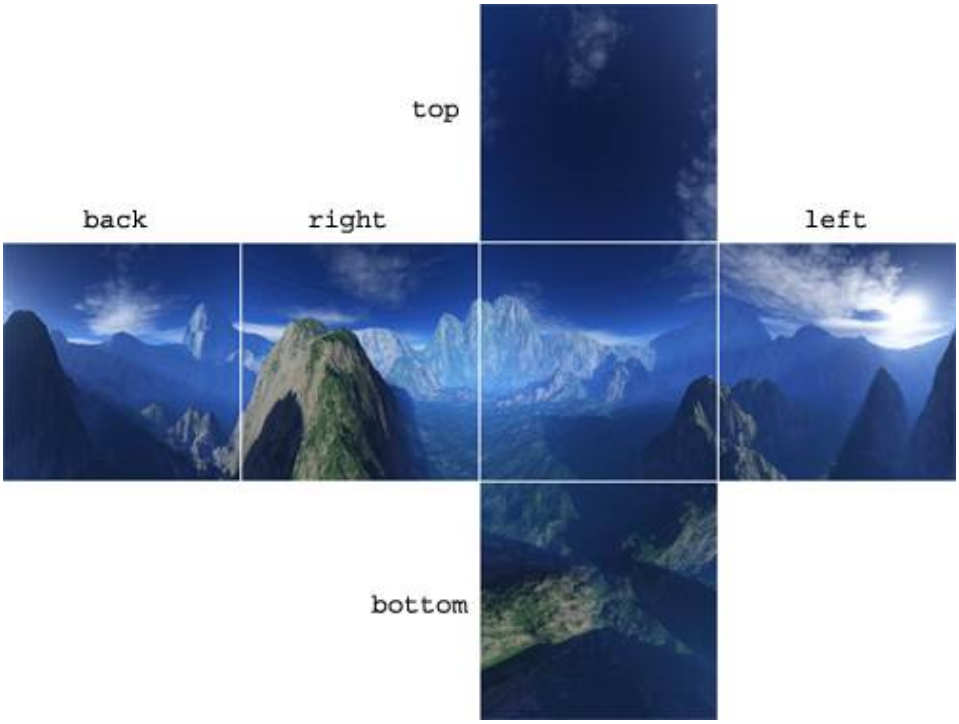


Imagen 111: Skybox desplegado. Imagen obtenida de www.gamedev.ru

*Consulte el glosario de vocabulario situado en la parte final del documento

Aplicación

Para este proyecto se utilizó un skybox que permitía dar un cielo realista (disponible para descarga en los assets de los que dispone Unity3D). Véase en la imagen 112 siguiente el efecto del fondo del escenario sin ningún tipo de skybox.

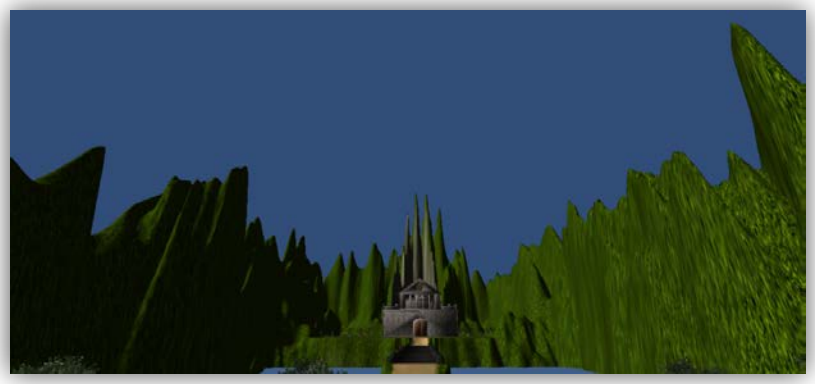


Imagen 112: Escenario sin utilizar skybox

En la imagen 113 de a continuación puede verse el resultado del escenario después de haberle sido aplicado un skybox.

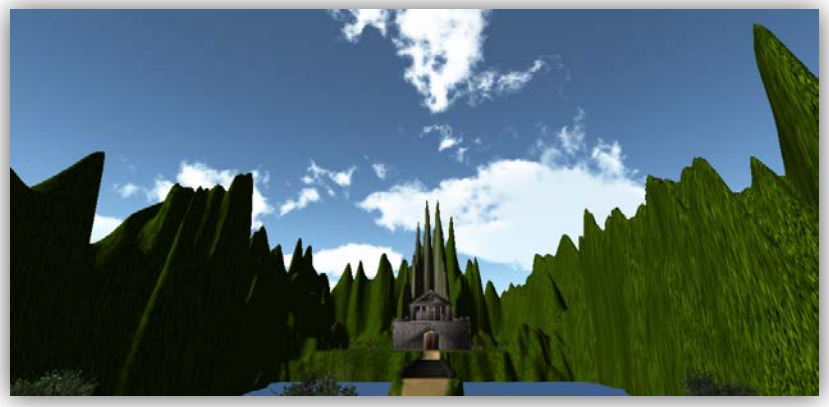


Imagen 113: Escenario utilizando skybox

Para la aplicación de los skybox ha de seguirse dos sencillos pasos:

- Ir a *Edit -> Render Settings*

- Aparecerá la pantalla que puede verse en la imagen 114 debajo de 

que nos permitirá seleccionar el skybox sobre la scene actual.

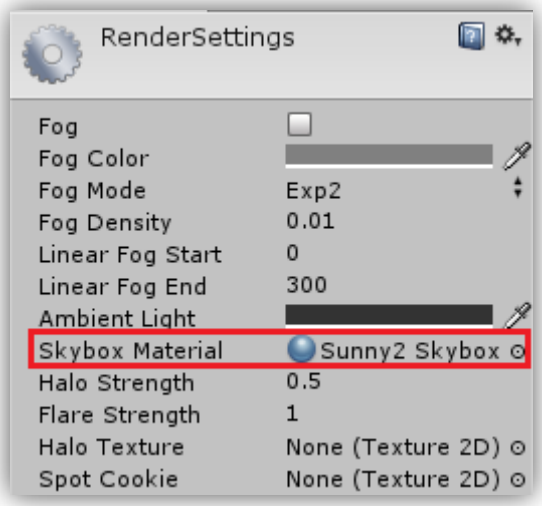


Imagen 114: Muestra de cómo introducir un skybox

En el recuadro  se deberá seleccionar el skybox deseado. También existen otras diversas opciones para modificar, pero eso ya será cuestión de ir probando hasta encontrar la deseada.

-4.3.2: Menú principal

Éste menú se creó con el editor de Unity3D al igual que el escenario principal. Existen dos tipos de menús, la versión en 2D o la versión en 3D. Finalmente se decidió utilizar la versión en 3D. Para la realización del menú principal se partió del escenario principal en el cual se le colocó un menú en el lugar de aparición del personaje principal. La primera tarea a realizar fue el background del menú principal. Éste está formado por un simple cuadrado modificado hasta obtener un rectángulo al que se le añadió una textura de piedra. En la imagen 115 puede verse el muro de piedra introducido.

Imagen 115: Bloque de piedra usado como background del menú principal

Posteriormente fueron creadas las letras del título: "Eternal Destiny" y los dos botones: "Play Game" y "Quit". Para ello se utilizó el GameObject llamado *3D Text*. Para añadir éste objeto se debe ir a *GameObject -> Create Other -> 3D Text.* Se colocaron tres de ellos con sus respectivos nombres. Finalmente se añadió el personaje principal a un lado. Puede visualizarse en la imagen 116 el estado final del menú principal con todos los elementos explicados anteriormente.

Imagen 116: Estado final del menú principal

4.4 – Funcionamiento de Unity3D

En esta sección se explicará el método de importación de los objetos creados con el Blender al Unity3D. También detallará la preparación de los objetos para ser usados en el engine y todos los componentes que forman parte de ellos.

-4.4.1: Importación de objetos desde Unity3D

Importación del mesh

Nota: Se recomienda leer el apartado 4.1.2 antes de proseguir con este apartado.

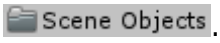
Para la importación de los objetos a Unity3D se ha optado por utilizar el formato .fbx a pesar de que puede leer también los formatos .dae, .3DS, .dxf y .*obj*.

Para la correcta importación del objeto se deberán tener dos elementos fundamentales: el objeto.fbx y una carpeta con las texturas de éste objeto.

Nota: Es muy importante mantener siempre el objeto.fbx y las texturas de éste en una misma carpeta con las texturas en su propia carpeta ya que de lo contrario podrá ocasionar muchos problemas en el momento de buscar y asignar éstas desde Unity3D.

Tal y como se muestra en la imagen 117, hay que situarse en la pestaña de 

y ha de crearse una nueva carpeta en el apartado de 

. Se renombra ésta al nombre que se desee, en este caso 

. Es en esa carpeta donde se deberá guardar el archivo objeto.fbx y las texturas. Para añadir estos se deberá localizar el lugar donde se instaló Unity3D, localizar la carpeta recién creada y copiar allí los archivos.

Nota: También existe otra alternativa más cómoda y es la de realizar el *Drag&Drop** de los archivos sobre la carpeta directamente en Unity3D.

*Consulte el glosario de vocabulario situado en la parte final del documento

En éste caso se realizará la importación del personaje principal del juego.

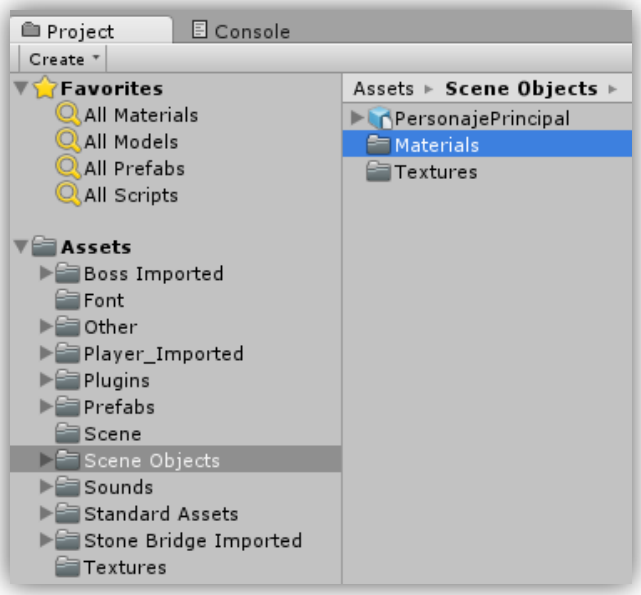


Imagen 117: Importación del personaje principal y sus texturas a Unity3D

Al haberse realizado la anterior acción, Unity3D comenzará a importar los objetos para su posterior uso y creará 3 archivos:

- PersonajePrincipal: Objeto del personaje principal que se utilizará en Unity3D.
- Materials: carpeta que contendrá los materiales importados del objeto.
- Textures: carpeta de texturas correspondientes al objeto importado.

Unity3D no asignará las texturas a los materiales de forma automática, es por ello que se deberá hacer manualmente. Para hacerlo hay que abrir la carpeta Materials y navegar por cada uno de ellos. La imagen 118 contiene todos los materiales que ha importado Unity3D del personaje principal.

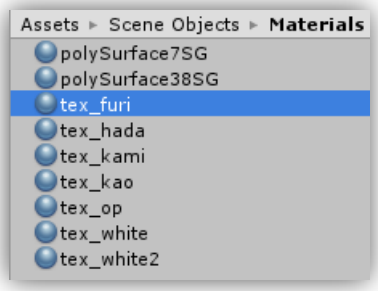


Imagen 118: Materiales del personaje principal importados a Unity3D

Una vez seleccionado un material de la lista hay que situarse en la pestaña Inspector (véase la imagen 119) donde podrá verse la información del material y la textura asignada (que como ya se ha citado anteriormente, por defecto, no dispone de ninguna, recuadro). El recuadro indica el color principal, que viene por defecto en color negro. Éste también se deberá de cambiar en función del color que se busque con la textura. Finalmente el recuadro señala el tipo de efecto del que dispone el material, por defecto viene la opción marcada en Diffuse, que es el que se suele utilizar normalmente.

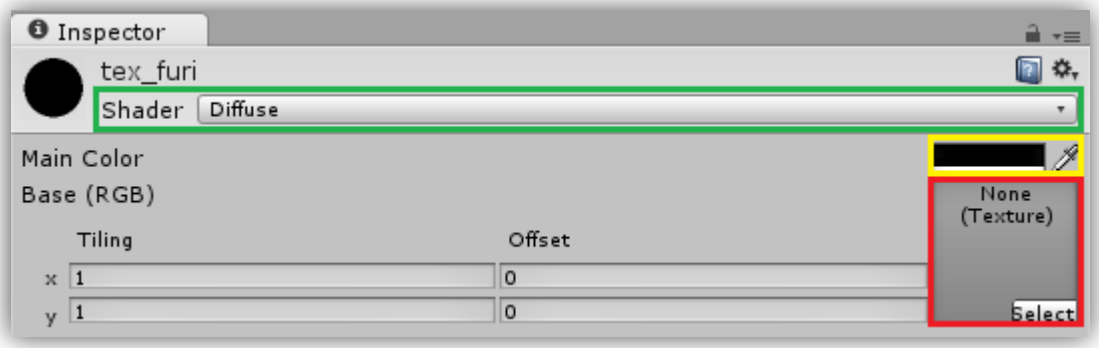


Imagen 119: Características por defecto de un material importado a Unity3D

Como se muestra en la imagen 120, el material en su estado por defecto mostrará el siguiente Preview debido a que su color por defecto es negro y además no dispone de ninguna textura.

Imagen 120: Preview del material sin textura y sin color seleccionado

En la imagen 121 se muestra el mismo material una vez le han si do cargados la textura y dado el color necesario, en éste caso color blanco.

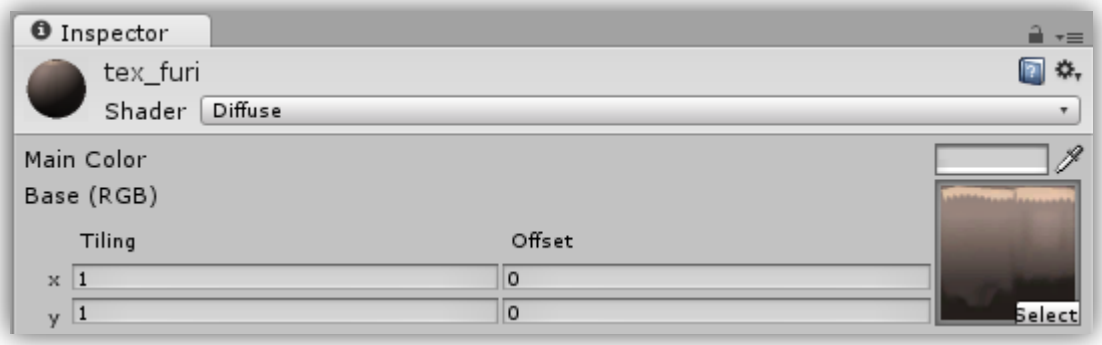


Imagen 121: Características modificadas de un material importado a Unity3D

Como se muestra en la imagen 122, el material una vez hay sido modificado mostrará el siguiente Preview debido a que su color es el blanco y dispone de una textura dada.

Imagen 122: Preview del material con color blanco y textura añadidos

Una vez realizada ésta acción para cada uno de los materiales se habrá logrado texturizar dentro de Unity3D el objeto en cuestión. Puede verse el resultado en la imagen 123, donde se muestra, en la imagen de la izquierda el personaje recién exportado al engine con todos los materiales por defecto (color negro y sin texturas) y en la imagen de la derecha el personaje con todos sus materiales modificados (con su respectivo color y texturas aplicadas).Para poder visualizar este resultado se debe ir al Preview del PersonajePrincipal.

Imagen 123: A la izquierda el personaje principal con todos sus materiales por defecto y a la derecha éste con todos los materiales modificados

Importación de las animaciones (si el objeto dispone de ellas)

Podría ser que haya objetos que hayan sido importados a Unity3D y que dispongan de animaciones previamente realizadas con el programa de modelaje y animación utilizado, en ese caso se deberá efectuar algunos pasos añadidos para poder utilizar las animaciones de éste.

Éste es el caso del personaje principal importado en el apartado anterior que dispone de varias animaciones. Se deberá seleccionar el PersonajePrincipal y situarse en la pestaña

Inspector donde aparecerá la información que puede verse en la imagen 124. En esta imagen se muestran las características modificables del modelo importado, el rig de y las animaciones de éste. A continuación de procede a la explicación en detalle de tales características.

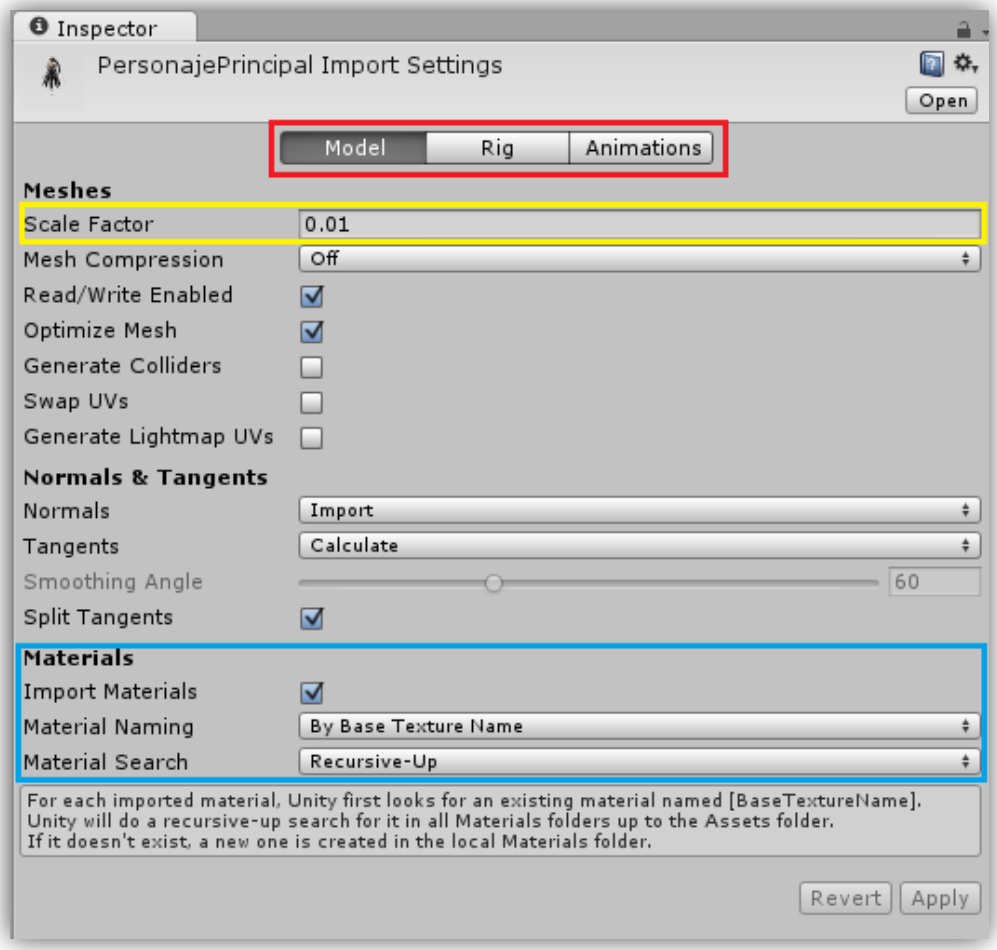


Imagen 124: Panel de ajustes de las características modificables del modelo

En el recuadro se muestran los tres pasos que deberemos modificar para el correcto funcionamiento de las animaciones del personaje principal en Unity3D. Son los siguientes:

- Model : En el recuadro aparece "scale factor" cuyo valor deberá ser modificado a 1 en lugar del 0.01 (el motivo de que el valor por defecto sea 0.01 es que, para todos los archivos .fbx importados, Unity3D les asigna erróneamente un tamaño

100 veces menor). Seguidamente habrá que comprobar el recuadro , donde se dice al Unity3D de qué manera ha de tratar con los materiales del personaje principal. A pesar de que existen varias opciones de tratado de los materiales, la opción más común es la que viene por defecto y se puede ver en la imagen anterior.

- Rig: Al pasar a esta pestaña aparecerá el contenido de la imagen 125 donde permite escoger el tipo de importación de las animaciones del rig del modelo.

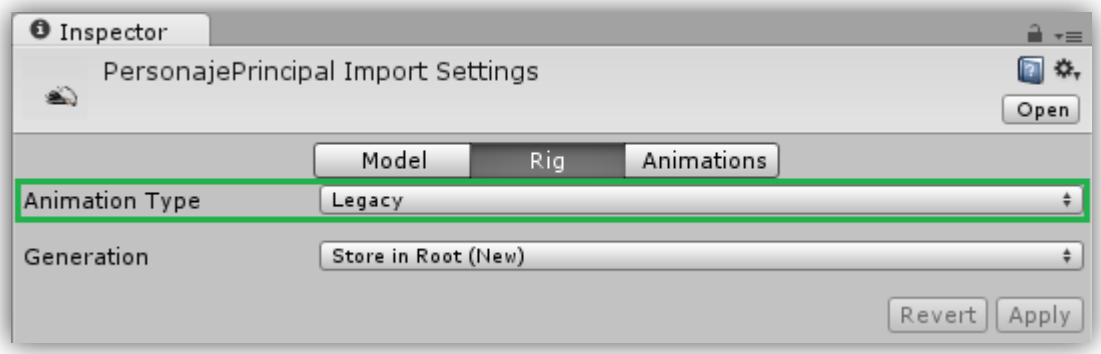


Imagen 125: Panel de ajustes del tipo de importación de las animaciones del rig

En este recuadro será necesario que el "Animation Type" sea cambiado a modo "Legacy". De no hacerlo no se podrá utilizar absolutamente ninguna animación del personaje.

- Animations: En esta pestaña, tal como se ve en la imagen 126, se mostrarán todas las animaciones de las que dispone el personaje y podrá ser modificado el tiempo de inicio y final de cada una de éstas , aunque si el tiempo se ha gestionado correctamente en Blender, no será necesario realizar ninguna modificación alguna.

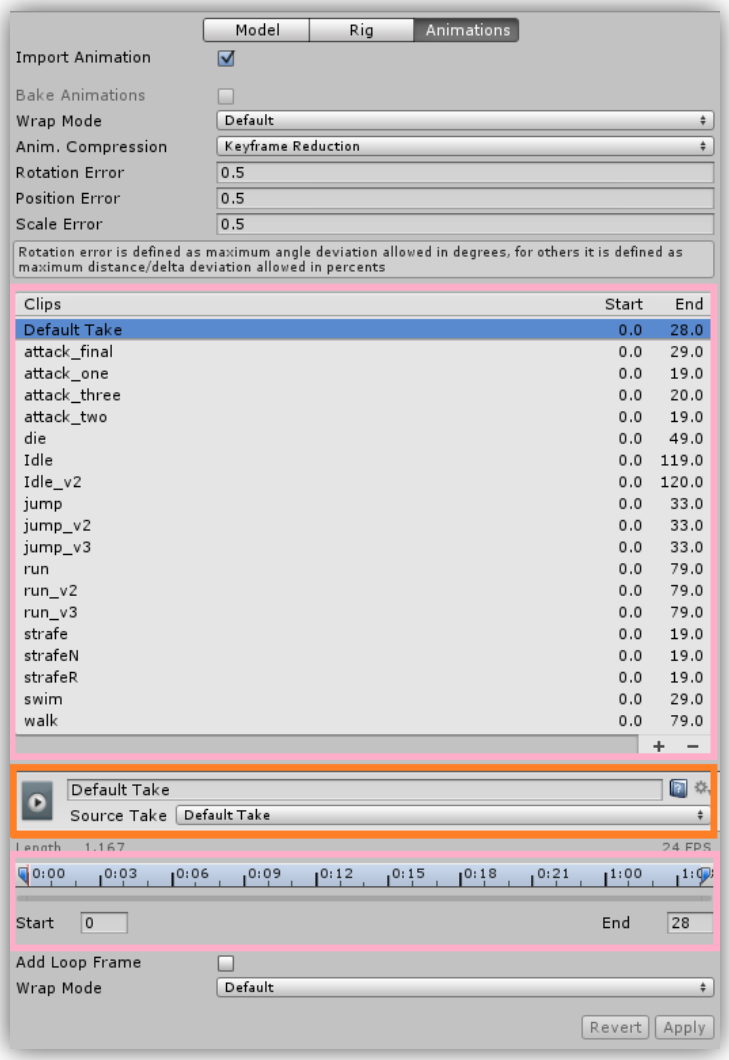


Imagen 126: Panel de ajustes de todas las animaciones importadas del modelo

En el recuadro  se muestra la animación que el Unity3D ejecutará por defecto al colocar el personaje en la scene y ejecutar la acción de animación, puede ponerse la que se desee pero ésta opción también puede modificarse vía código en los scripts (esto se podrá ver en el apartado 4.4.2.2).

Básicamente aquí se tendrá que comprobar que las animaciones funcionan de manera correcta utilizando el Preview que puede verse en la imagen 127 donde podrán visualizarse todas las animaciones del modelo importado.

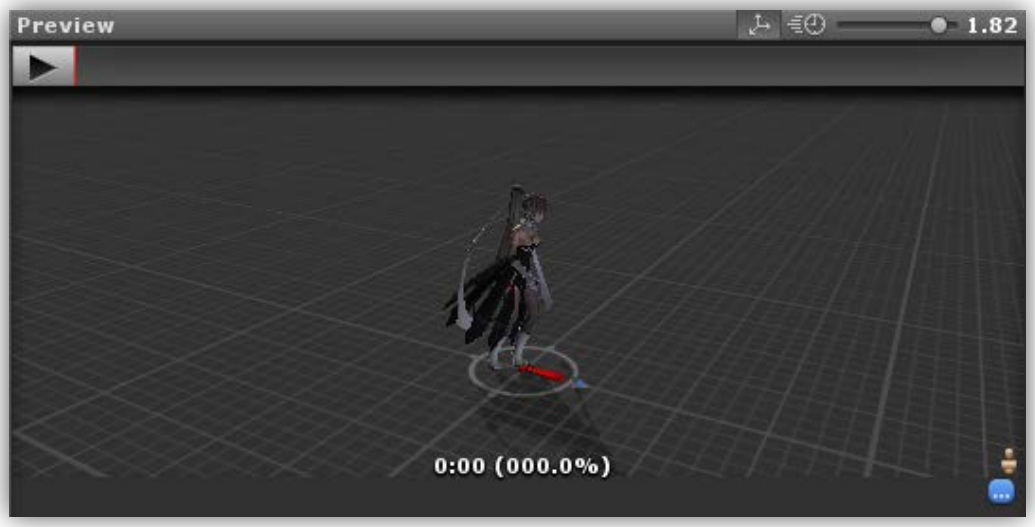


Imagen 127: Panel del Preview que ofrece la visualización de las animaciones del modelo importado

-4.4.2: Preparación de los objetos para su uso en Unity3D

Una vez se tiene importado el objeto a Unity3D correctamente, se debe realizar drag&drop del objeto, en este caso el PersonajePrincipal sobre el lugar que se quiera en la scene. En la imagen 128 se muestra, en la imagen de la izquierda la pestaña "Game" donde podrán ser visualizados todos los objetos introducidos en la scene y también el juego al ser ejecutado. Por otra parte en la imagen de la derecha está la pestaña "Scene" donde se realizará el drag&drop de todos los objetos utilizados en el engine.

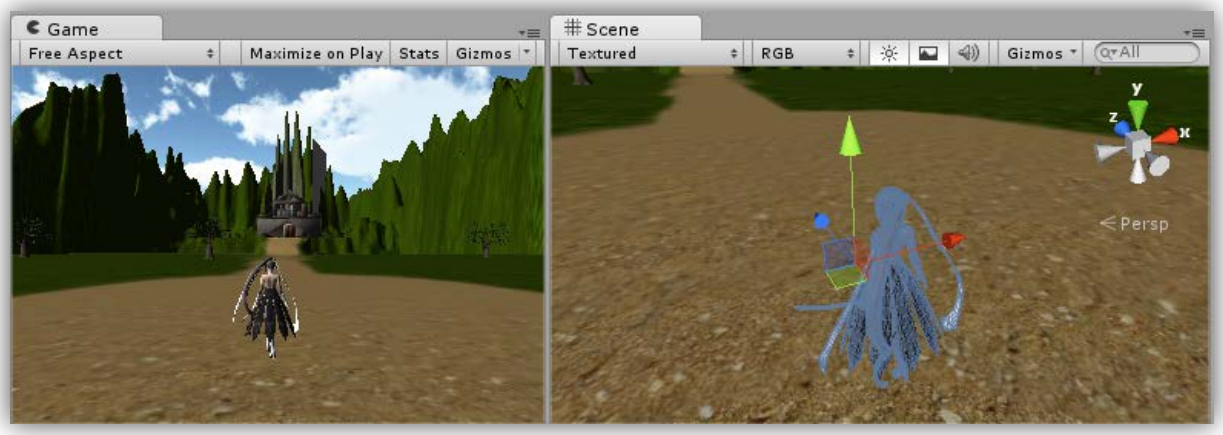


Imagen 128: A la izquierda puede verse el menú de juego y a la derecha la scene donde se arrastrarán mediante el drag&drop todos los objetos que se quieran utilizar

A continuación se analizarán cada uno de los componentes necesarios para el uso de objetos en Unity3D.

Cada componente añadido al objeto puede ser accedido vía la interfície de Unity3D tal y como se verá en las imágenes de a continuación o también pueden ser accedidos a través de los scripts, es decir, cualquier modificación que se ejecute en el componente visual introducido al objeto será modificado en el código de éste del script y viceversa.

Cabe decir que la inclusión de componentes al objeto es de vital importancia para poder utilizar éste dentro del juego. El scripting en los objetos de unity3D funciona sobre éstos componentes, si un objeto no dispone de algún componente no se podrá acceder a éste hasta que no le sea añadido.

Por ejemplo, siempre que se quiera mover un objeto se requerirá del componente Transform, siempre que se quiera ejecutar una animación de un objeto se requerirá un componente Animation, etc.

- 4.4.2.1: Componentes de Unity3D por defecto

Los componentes se encuentran justo debajo de la pestaña 

al seleccionar el objeto. Estos componentes vendrán por defecto con cada objeto introducido en la scene.

GAMEOBJECT, NAME & *TAG** 

- **Definición**

En este componente será adquirido por cada objeto que esté en la scene. Se definirá al objeto con un nombre y un tag y éstos podrán ser modificados. En este caso señarlaremos al objeto personaje principal con el nombre PersonajePrincipal y con el tag de “Player”.

*Consulte el glosario de vocabulario situado en la parte final del documento

Pueden crearse tantos tags como se desee. En la imagen 129 se muestran el componente con varias características del personaje principal.

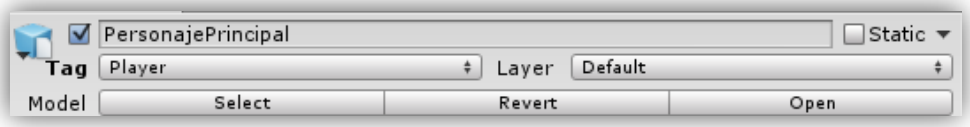


Imagen 129: Componente GameObject del personaje principal

- **Uso (Código)**

El uso de éste componente vía código será muy importante para poder encontrar a otros objetos dentro del escenario e interactuar con ellos. En el código que puede verse en la imagen 130, para poder buscar a un objeto se deberá crear un objeto de tipo "GameObject" al que se le llamará "objeto buscado". Seguidamente buscaremos el objeto jefe final que posee como tag "EnemigoFinal".

```
private GameObject objetoBuscado;
objetoBuscado = GameObject.FindGameObjectWithTag("EnemigoFinal");
```

Imagen 130: Código de instanciación de un GameObject del juego en C#

API de Unity3D de este componente pulsando aquí: GameObject

TRANSFORM

- **Definición**

Control de la posición, rotación y escala del objeto en la scene de cada uno de sus ejes. La transformada de un objeto será entonces una variable que contendrá la posición, rotación y escala actual del objeto en la scene. Pueden verse en la imagen 131 todos los valores de posición, rotación y escala del personaje principal en el juego.

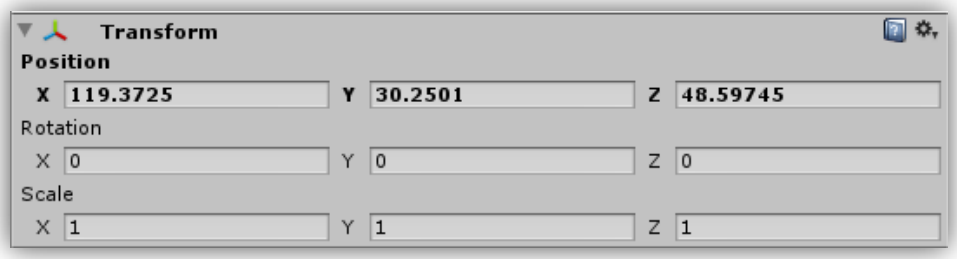


Imagen 131: Componente Transform del personaje principal

- **Uso (Código)**

En los scripts, ésta variable será esencial para poder realizar todas las acciones de movimiento durante el juego. En su versión de código se realizaría de la siguiente manera: Se declararía la variable de tipo 'Transform' a la que se le llama 'playerTransform':

```
private Transform playerTransform;
```

Seguidamente se le asignaría a 'playerTransform' la posición del objeto en el momento de iniciar el juego.

```
playerTransform = transform;
```

A partir de éste momento se podrá dar un comportamiento de movimiento al objeto pudiendo mover éste por el escenario o realizar una rotación de éste. Por ejemplo:

```
playerTransform.Rotate(x,y,z);
```

En la imagen de arriba se puede ver como se le puede dar una rotación al objeto siendo 'x' grados en el eje X, 'y' grados en el eje Y y 'z' grados en el eje Z.

API de Unity3D de este componente pulsando aquí: Transform

ANIMATION 

o Definición

Control de las animaciones, en el caso que disponga de ellas. Éste componente únicamente se añadirá al objeto añadido por defecto siempre y cuando éste disponga de animaciones. En la imagen 132 puede visualizarse el componente de Animation del personaje principal con las características por defecto.

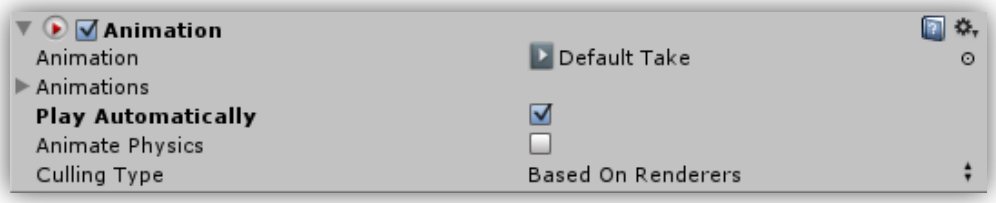


Imagen 132: Componente Animation del personaje principal

o Uso (Código)

En los scripts, una animación únicamente podrá ser llamada por el objeto siempre y cuando éste contenga el componente de Animation. De ser así, simplemente se deberá de utilizar el código siguiente que permite reproducir una animación en concreto utilizando el "Play", en este caso la animación llamada "Correr".

```
animation.Play("Correr");
```

En el caso de querer parar una animación se llamaría al "Stop". Ésta acción parará la ejecución de la animación que se esté ejecutando.

```
animation.Stop();
```

API de Unity3D de este componente pulsando aquí: Animation

- 4.4.2.2: Componentes de Unity3D añadidos

Estos componentes no vienen por defecto con cada objeto introducido en la scene de Unity3D, sino que deberán ser añadidos posteriormente. Se realizará la explicación y el uso de los más utilizados a pesar de que existe un número de ellos mayor.

RIGIDBODY & COLLIDERS

- **Definición Rigidbody**

El rigidbody al ser añadido a un objeto le otorga las físicas propias de los objetos reales. Es controlado por el control de físicas de Unity3D. Al realizar ésa acción el objeto está afectado por la gravedad. También se le pueden aplicar fuerzas exteriores sobre éste mediante *scripting**.

- **Definición Collider**

El collider al ser añadido a un objeto le otorga la capacidad de colisionar con los elementos de su entorno. También permite la activación de triggers en el juego (se verán a continuación). Todos los objetos a los que se le quiera dotar de capacidad de colisión deberán llevar uno. A continuación puede verse en la imagen 133, a la izquierda un mesh con forma esférica, y a su derecha el collider de éste. Es muy importante que el collider tenga la forma más parecida posible a la figura a la que se le aplicará (a no ser que se utilice para los triggers).

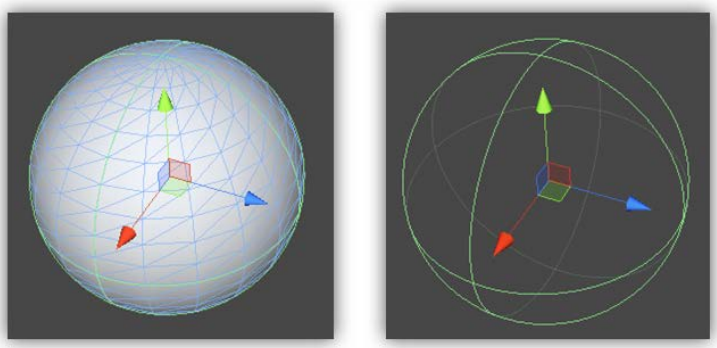


Imagen 133: Mesh con forma esférica (izquierda) y collider de éste (derecha)

Existen distintas formas de colliders. En la imagen siguiente 134 pueden verse los distintos colliders de los que dispone Unity3D.

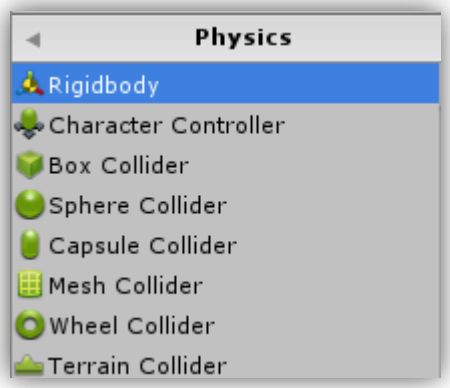


Imagen 134: Tipos de colliders disponibles en Unity3D

- **RigidBody VS Collider:**

El uso de rigidbody y collider en los objetos es un tema que suele causar diversas confusiones.

El rigidbody, como ya se ha dicho anteriormente otorga gravedad al objeto al que es aplicado y éste es controlado por el control de físicas de Unity3D. El rigidbody pues, se deberá aplicar entonces a todos aquellos objetos a los que se les quiera disponer de movimiento con una afectación por la fuerza de la gravedad i/o fuerzas externas vía scripting.

Por otro lado el collider no otorga al objeto estas capacidades de afectación de las distintas fuerzas sino que únicamente le otorga la capacidad de colisión con los objetos del entorno.

Actualmente, existe una gran contradicción con todo esto ya que al añadir un collider a un objeto, vía scripting se le puede añadir también gravedad o dotar de fuerzas externas a éste y de ésta manera dispondrá de un comportamiento tal y como si de un rigidbody se tratase.

Es por esa razón que prácticamente siempre se utilizan colliders en lugar de rigidbody, pero el uso adecuado de éstos sería el siguiente:

- Los **rigidbody**, deberían ser añadidos únicamente a los objetos a los que se les quiera aplicar la fuerza de la gravedad i/o fuerzas externas.
- Los **collider**, deberían ser añadidos únicamente a los objetos estáticos del escenario y que no van a sufrir ningún tipo de afectación por ningún tipo de fuerza.
- Los **rigidBody y colliders**, estos dos juntos, deberán ser añadidos a todos aquellos objetos a los que se les quiera aplicar la fuerza de la gravedad i/o fuerzas exteriores y también quieran ser dotados de capacidad de colisión con otros objetos.

Seguidamente, en la imagen 135 se muestra un ejemplo de la afectación de estos componentes de Unity3D sobre un cubo. A la izquierda se encuentra un cubo y a su derecha los componentes de éste.

Se le añade únicamente un collider de tipo box collider al cubo y se ejecuta el juego para ver qué sucede.

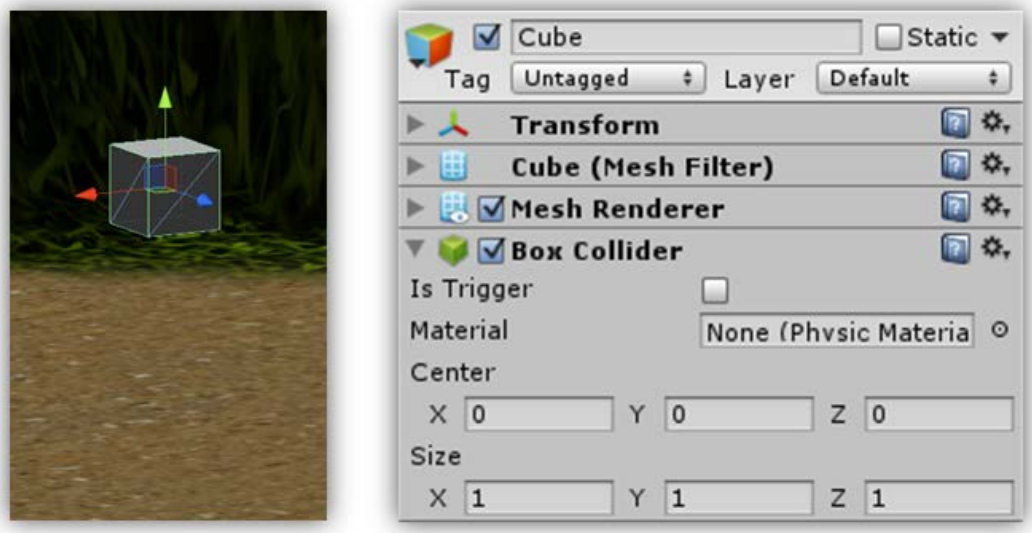


Imagen 135: Resultado de la aplicación de solamente un collider al cubo

Com puede verse en la imagen de arriba, al ejecutar el juego, el cubo se queda completamente quieto ya que no se le aplica ningún tipo de fuerza, pero mantiene el collider activo. De esta manera cualquier objeto puede colisionar con él durante el juego.

A continuación se le añade un rigidbody, se le quita el collider al cubo y se ejecuta el juego para ver qué sucede.

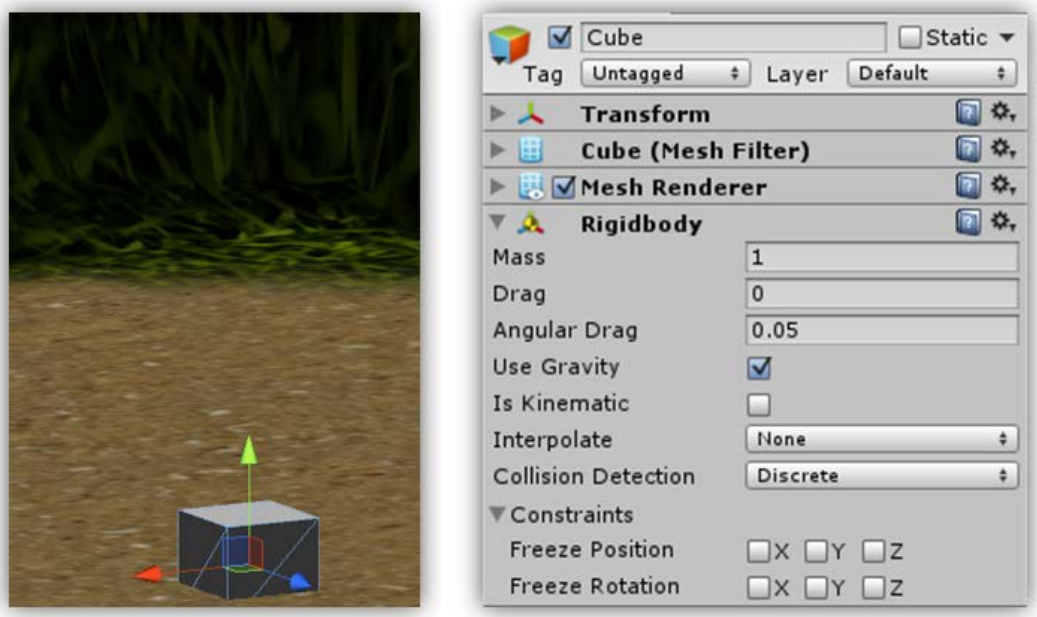


Imagen 136: Resultado de la aplicación de solamente un rigidbody al cubo

Como puede verse en la imagen 136, al ejecutar el juego, el cubo es afectado por la fuerza de la gravedad y comienza a descender hasta que se encuentra con el suelo, lo traspasa y sigue descendiendo debido a que, a pesar de que el suelo dispone de un collider propio, el cubo no.

En la imagen 137 puede verse la combinación del rigidBody con el collider y como puede verse en la imagen, al ejecutarse el juego, el cubo es afectado por la fuerza de la gravedad y comienza a descender hasta que se encuentra con el suelo y colisiona con éste. Esto es debido a que ambos objetos disponen de un collider que genera una colisión entre ellos.

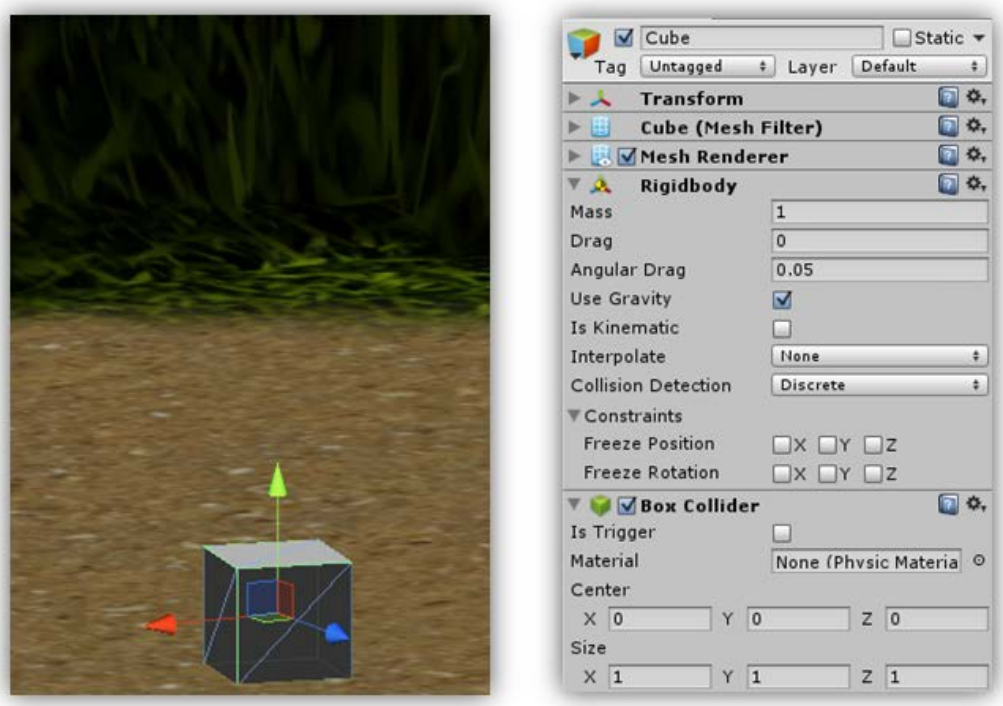


Imagen 137: Resultado de la aplicación de un rigidbody y un collider juntos en un cubo

- **Triggers:**

Los triggers son propiedades que únicamente son accesibles a través de los colliders. Todos los colliders pueden ser triggers siempre y cuando se active la opción correspondiente. Puede verse el icono de activación del trigger del collider en la imagen 138.

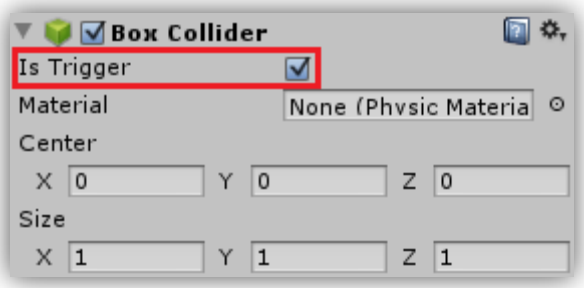


Imagen 138: Box collider con las propiedades de los triggers activadas

Los triggers son ignorados por el motor de físicas de Unity3D y son activados cuando otro collider de otro objeto colisiona contra ellos.

Importante: Todos los triggers tienen definido su comportamiento en el script asociado al objeto que dispone del collider (con el trigger activado).

En la imagen 139 de la izquierda, a la puerta se le ha añadido un sphere collider, y se ha activado la opción de trigger de éste.

Imagen 139: Efecto de la activación del trigger del sphere collider de la puerta principal de la iglesia

Como puede verse en la imagen del a derecha, en el momento en que un objeto con un collider (personaje principal) colisiona con el trigger (sphere collider), éste puede ejecutarse de tres maneras distintas:

- **OnTriggerEnter():** esta función es llamada cuando un objeto con un collider entra dentro del trigger.
- **OnTriggerExit():**esta función es llamada cuando un objeto con un collider para de tocar el trigger.
- **OnTriggerStay():** esta función es llamada cada frame siempre y cuando el objeto con un collider esté dentro del trigger.

El código para el uso de triggers es el siguiente (ejemplo de trigger contenido en el objeto personaje principal):

```
public void OnTriggerEnter(Collider other){
    if(other.CompareTag("Water")){
        Debug.Log ("Player has entered into the water!");
    }
}
```

Imagen 140: Código de activación del trigger del player principal cuando entra en contacto con el agua del lago

Explicación del código de la imagen 140: Si el collider del objeto personaje principal, entra en contacto con un collider llamado "Water" (que pertenece al lago), se mostará por pantalla el mensaje "Player has entered into the water!"

API de Unity3D de este componente pulsando aquí: Rigidbody, Collider

CHARACTER CONTROLLER 

o Definición

Este componente se suele utilizar para el control del jugador, en este proyecto se utilizará para el control del personaje principal. Actúa como un collider general que se aplica a humanoides. Además de proporcionar las características propias de un collider dispone de funciones de movimiento que permiten al objeto que lo lleve moverse por el escenario, subir pendientes y también permite subir escaleras. Éste componente no reacciona a ningún tipo de fuerza a no ser que se realice vía scripting.

Al añadir el componente al personaje principal se dispondrá de la configuración dispuesta en la imagen 141.

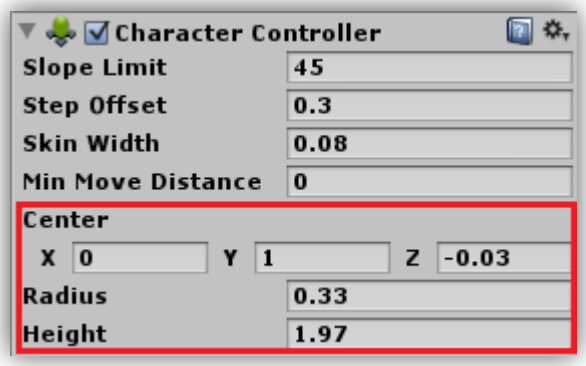


Imagen 141: Propiedades del Character Controller del personaje principal

Tal y como puede verse en la imagen 141, en el recuadro  estarán todas las variables que definirán el tamaño y la posición del character controller sobre el personaje. En la imagen 142, a la izquierda character controller aislado y a la derecha éste aplicado al personaje principal.

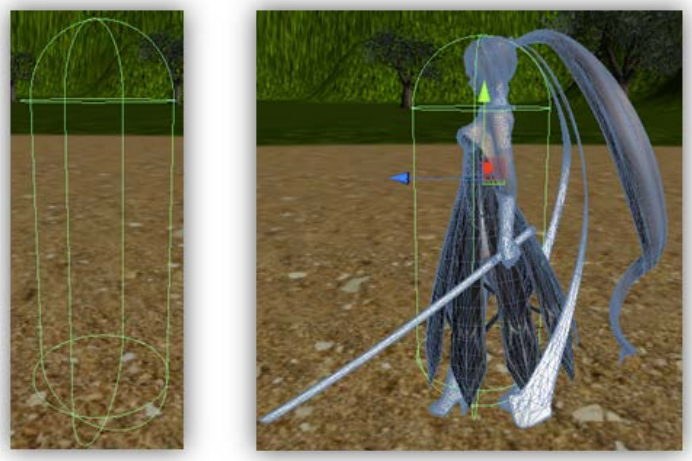


Imagen 142: Character Controller del personaje principal

API de Unity3D de este componente pulsando aquí: CharacterController

SCRIPTS 

o Definición

Los scripts que hayan sido añadidos a un objeto darán un comportamiento específico a éste a la hora de ejecutar el juego. Éstos pueden ser escritos en JavaScript o C#. En este proyecto todos han sido escritos en C# por lo que se explicarán las características de éstos en particular. En la imagen 143 puede verse un ejemplo de los scripts que posee el personaje principal.

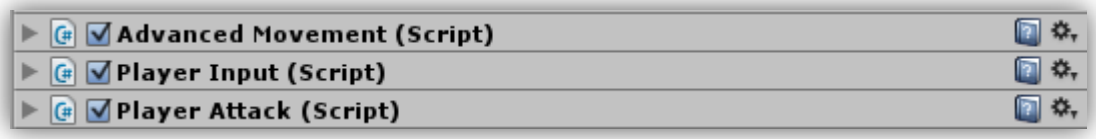


Imagen 143: Conjunto de scripts del personaje principal

o Uso (Código)

-Conceptos generales

Todos los scripts codificados son clases y mantienen las propiedades como tal. Todos los scripts en C# deben extender de la clase *MonoBehaviour** y así disponer de ciertas funciones heredadas de ésta mientras que los scripts en JavaScript no deben hacerlo ya que automáticamente derivan de éste y heredan sus métodos automáticamente. En la imagen 144 se muestra la definición de un nuevo script escrito en C# llamado 'PlayerInput' recién creado.

*Consulte el glosario de vocabulario situado en la parte final del documento

```
public class PlayerInput : MonoBehaviour
{
    // Use this for initialization
    void Start (){

    }

    // Update is called once per frame
    void Update ()
    {

    }
}
```

Imagen 144: Definición de la clase de PlayerInput

Como puede verse en la cabecera, en el recuadro ▭ se muestra la siguiente definición: "clase Nombre: deQuienExtiende". Al extender de la clase MonoBehaviour dispondrá de varias funciones heredadas. Dos de ellas definidas en el recuadro ▭ que vienen definidas por defecto al crear un nuevo script en C#.

- **Start():** Ésta función se llama antes que la primera vez que la función Update() es llamada. Por lo tanto únicamente será llamada una vez.
- **Update():** Ésta función es llamada en cada frame. Es decir, cuanto más potente sea el hardware del ordenador más veces por segundo se llamará. Se utiliza para definir el comportamiento in-game de un objeto.

El funcionamiento de los scripts en Unity3D es la siguiente:

Cuando el juego es ejecutado, todos los scripts ejecutarán en primer lugar la función de Start(), función en la cual se definirán los estados iniciales del objeto que contiene el script. A continuación todos ejecutarán la función de Update(), tantas veces como a fps se ejecute el juego, realizando los comportamientos programados en ésta función.

Existen también otras dos funciones más que complementan a éstas dos anteriores. Son las siguientes:

- **Awake():** Ésta función es llamada en el momento que se crea el objeto. Se utiliza normalmente para inicializar cualquier tipo de variable. Ésta función únicamente se llamará una única vez, antes que la función Start().
- **LateUpdate():** Ésta función es llamada cada frame. Es llamada después de llamar a la función Update().

Será pues en la función Update() y LateUpdate() donde habitualmente se situará prácticamente la mayoría del código.

Como se muestra en la imagen 146 se dispondrá entonces del siguiente orden de llamada de funciones.

Imagen 146: Orden de uso de las principales funciones de los scripts

Acceso a los componentes vía scripting

Éste será uno de los puntos más importantes. Todos y cada uno de los componentes de un objeto podrán ser accedidos vía scripting. Al acceder a un componente de un objeto, puede que se desee acceder a uno de sí mismo o a un componente de un objeto externo. Básicamente se utilizará el siguiente código mostrado en la imagen 147 para obtener el componente deseado.

```
componente = GetComponent<ComponenteBuscado>();
```

Imagen 147: Obtención general de un componente de un objeto

En el recuadro  se introduce el tipo de componente que se desea obtener.

- Si se quiere acceder a un componente del propio objeto:

```
private CharacterController controlador;
controlador = GetComponent<CharacterController>();
```

Imagen 148: Obtención del componente Character Controller del propio objeto que llama la a función

En la imagen 148 puede verse el trozo de código que permite obtener el componente Character Controller del propio objeto que contenga el script.

- Si se quiere acceder a un componente de un objeto externo:

```
camaraJugador = GameObject.FindGameObjectWithTag("MainCamera");
camaraJugador.GetComponent<Transform>();
```

Imagen 149: Obtención del componente Transform del objeto externo con tag MainCamera

En la imagen 149 puede verse el trozo de código que permite obtener el componente Transform de la cámara (con el tag de MainCamera)

AUDIO SOURCE 

Definición

Éste componente permite la reproducción de un AudioClip (fragmento de audio). El hecho de añadir un AudioSource a un objeto, hace de éste que sea un emisor de sonidos. En el caso del personaje principal, le será añadido este componente debido a que cuando realice cada una de sus acciones emitirá un sonido (por ejemplo, cuando ataque se reproducirá un AudioClip con un grito). A continuación se muestra en la imagen 150 el personaje principal con el icono de un altavoz, indicando que un AudioSource ha sido añadido. También se muestra la derecha el componente de AudioSource con todas sus características.

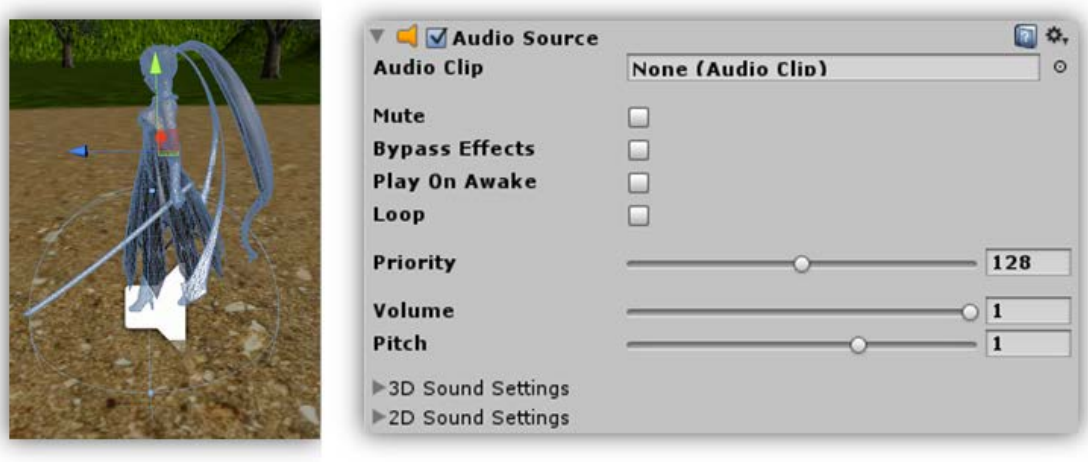


Imagen 150: Personaje principal con un Audio Source añadido (izquierda) y panel de ajuste de las características de éste (derecha)

Los sonidos pueden ser ejecutados en un modo 2D o un modo 3D. Hay que tener muy en cuenta ésta característica debido a que de ser 2D serán escuchados de igual manera independientemente de donde esté y qué posición tenga el emisor del sonido. Mientras que en un sonido 3D se tendrá en cuenta la posición y el lugar del objeto emisor. Para activar el sonido 3D véase el recuadro ▭ de la imagen 151 del componente AudioClip de a continuación.

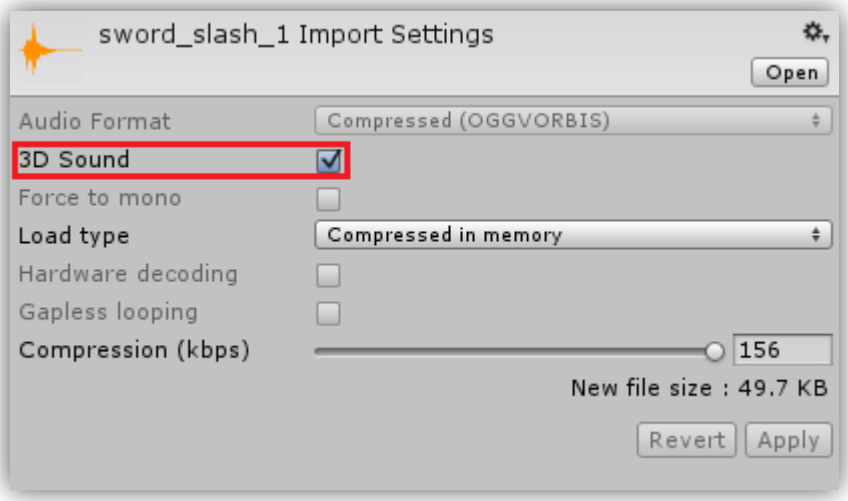


Imagen 151: Panel de ajuste de las características de un AudioClip

o Uso (Código)

Para poder ejecutar un sonido en el juego, existen dos maneras de hacerlo: ejecutar el sonido una única vez o ejecutar el sonido de manera continuada llamado loop.

Para poder ejecutar el sonido simplemente se deberá llamar al componente de AudioSource (imagen 152) del objeto, ejecutar el AudioClip que tenga establecido (recuadro ▭) y marcar si se desea ejecutar el sonido de manera continuada (recuadro ▭).

Imagen 152: Componente AudioSource del personaje principal

En la imagen 153 se muestra como ejecutar o parar un sonido del juego del objeto que ejecuta el código.

```
public AudioSource audio;
audio.Play();
audio.Stop()
```

Imagen 153: Código encargado de ejecutar y parar un AudioClip

Nota: Un objeto puede tener varios AudioSource, esto dependerá de si se quieren reproducir distintos tipos de sonidos al mismo tiempo.

API de Unity3D de este componente pulsando aquí: AudioSource

PARTICLE SYSTEM

Básicamente el particle system, es un sistema de partículas donde imágenes 2D son renderizadas en un espacio 3D. Para la posible creación de éste son necesarios tres elementos: Particle Emmiter, Particle Animator y Particle Renderer. Habrá que modificar las opciones de estos tres elementos para poder crear el efecto deseado.

Nota: Desde la versión 4 de Unity3D, se utiliza un nuevo sistema de partículas llamado Shuriken. Éste contiene muchas más opciones de personalización de partículas, no requiere de tantos elementos para su creación y se obtiene un mejor resultado final.

En la imagen 154 se visualiza el efecto del particle system llamado Legacy particle system a la izquierda, mientras que al a derecha puede verse el nuevo particle system Shuriken.

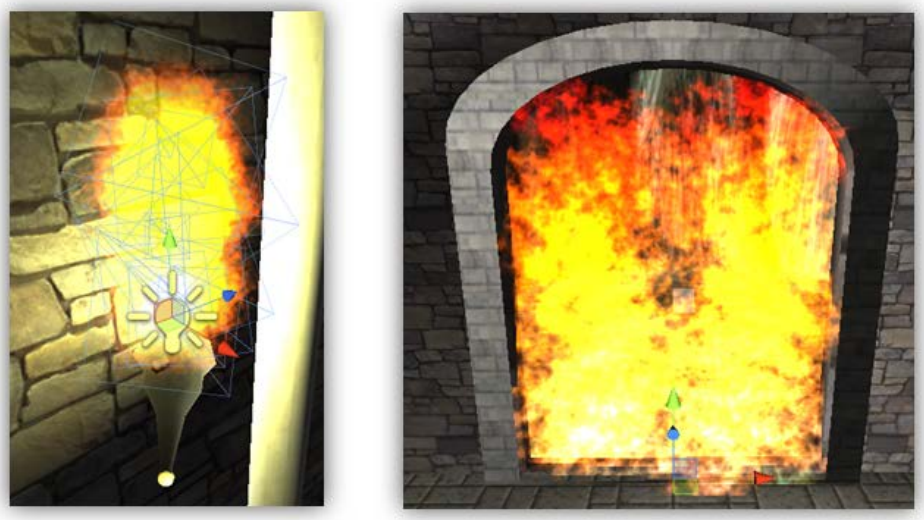


Imagen 154: Efecto del particle system Legacy en una antorcha de la iglesia (izquierda) y el particle system Shuriken en la puerta trasera de la iglesia (derecha)

API de Unity3D de este componente pulsando aquí: ParticleSystem

MESH RENDERER

Éste componente permite ocultar el mesh de cualquier objeto al que le se le sea aplicado. Simplemente se tiene que desmarcar la casilla de Mesh Renderer para no mostrar el mesh. Puede verse en la imagen 155 el panel con las características del componente Mesh Renderer.

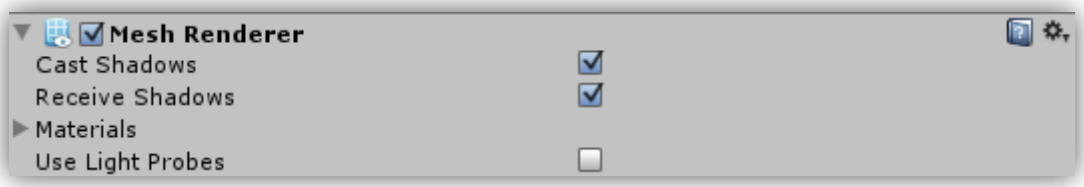


Imagen 155: Panel de ajuste de las características del Mesh Renderer

Es un componente muy útil ya que únicamente se está escondiendo el mesh mientras que el collider del objeto (si lleva) sigue activo. Esto suele utilizarse para definir límites de escenarios o realizar cuerpos invisibles. Puede verse en la imagen 156 el efecto de este. A la izquierda se muestra una pared con el mesh renderer activado mientras que en la derecha se muestra el mesh renderer del mismo objeto desactivado, dejando únicamente el collider de éste.

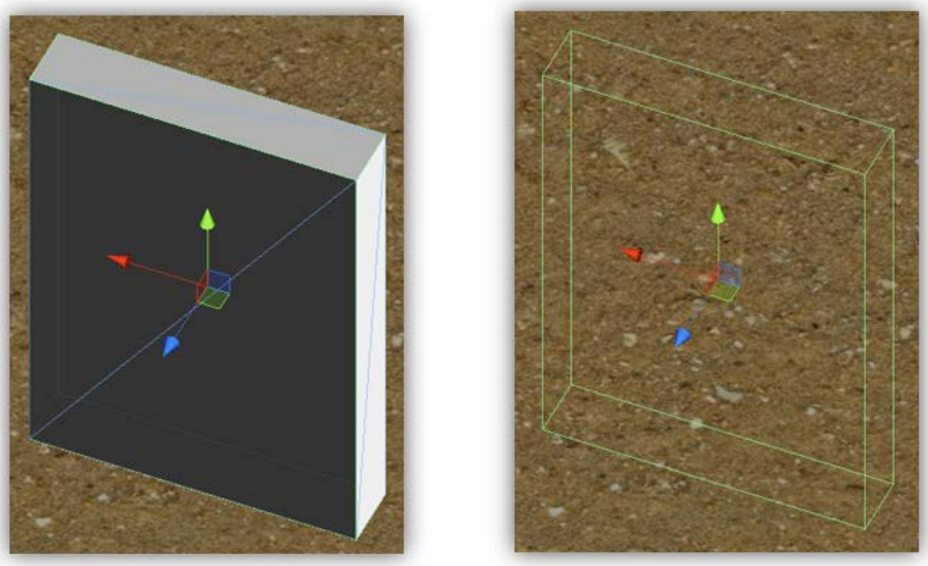


Imagen 156: Efecto del uso del Mesh Renderer sobre un objeto

4.5 – Funcionamiento de los objetos en Unity3D

En esta sección se detalla de qué manera funcionan los objetos importados y/o creados en este proyecto y cómo se comporta cada uno de ellos durante la ejecución del juego.

-4.5.1: Escenario Principal

El escenario principal se compone de un terreno rectangular con un lago en la zona media con un puente y una arboleda.

Terreno

El propio terreno dispone de un Terrain Collider (collider con forma de Terrain) mediante el cual el propio mesh del terreno actúa como un collider. También dispone de diversos colliders por todos los lados de éste para evitar que el personaje principal pueda acceder a zonas prohibidas o simplemente salirse del escenario escalando las montañas.

Para poder conseguir tal efecto se tuvo que añadir a la escena cuadrados, modificar las características de éstos de tal manera que fueran lo más finos posibles y se ajustaran a las medidas deseadas. Seguidamente se les quitaba el mesh, haciéndoles invisibles pero se les dejaba el box collider. De esta manera, cuando el personaje principal se mueve hacia uno de estos cuadrados e intenta atravesarlo, los colliders de ambos colisionarán evitando que el personaje principal avance. Cabe decir que todos los colliders introducidos fueron juntados en un único objeto llamado 'Scene Colliders' que puede verse en la imagen 157

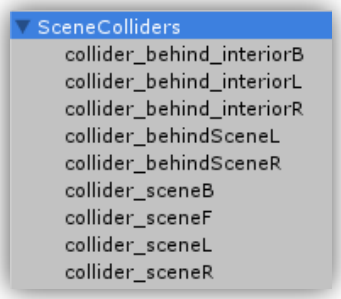


Imagen 157: Conjunto de colliders de paredes con el mesh renderer desactivado que definen los límites del escenario principal

Para poder ocultar el mesh de un objeto se debe ir a la pestaña Inspector, buscar el componente Mesh Renderer y desactivar la casilla. Puede verse el resultado en la imagen 158. A la izquierda puede verse la creación de las paredes (con el mesh renderer activado) con colliders colocadas por todo el escenario principal. A la derecha puede verse la misma distribución de paredes con sus respectivos colliders pero con el mesh renderer de cada pared desactivado.

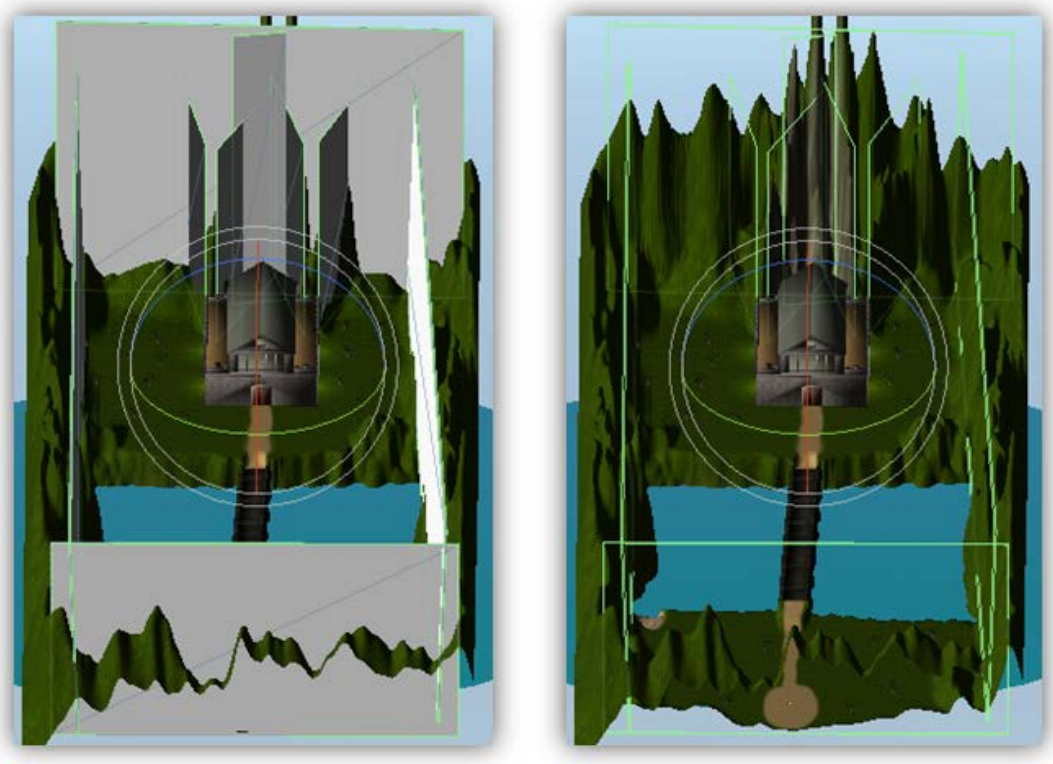


Imagen 158: Limitación del escenario utilizando colliders

Lago

Como puede verse en la imagen 159, en el lago también se utilizó un box collider para detectar la entrada del personaje principal en el agua. Éste collider dispone de un grosor mínimo. Esto se hizo de esta manera para que cuando personaje principal entrase en contacto con el agua del lago se pusiera a nadar.

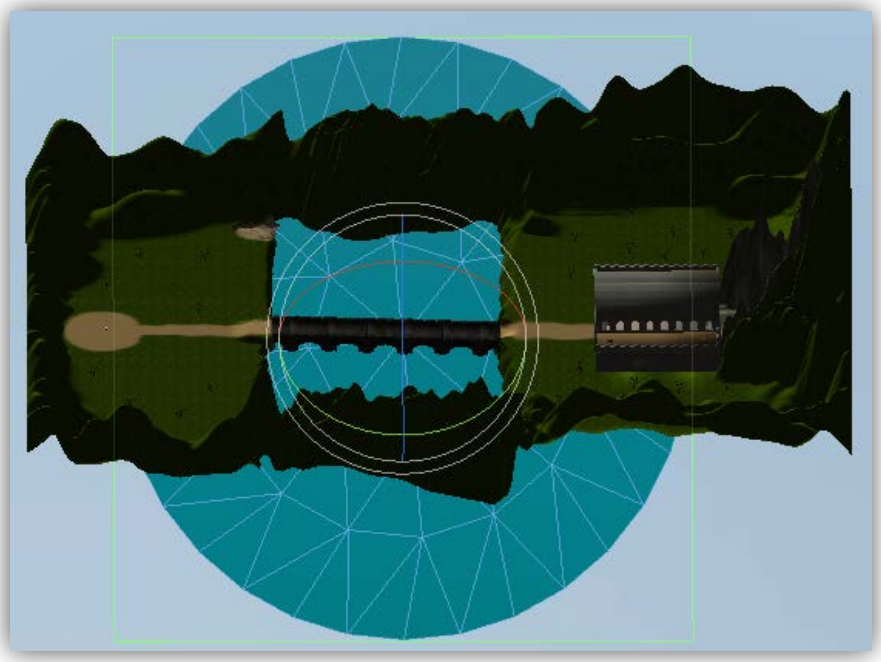


Imagen 159: Box collider del lago utilizado para detectar la entrada del personaje principal en él

Puente

El puente dispone de un mesh collider que permite que el personaje principal tenga contacto con éste al cruzarlo.

Arboleda

A todos los árboles que fueron introducidos en la escena, se les añadió un mesh collider permitiendo el contacto del personaje principal con éstos y posteriormente se juntaron todos en un único objeto llamado 'Tree_Group'.

-4.5.2: Menú Principal

Disponiendo del menú de la imagen 160, se le da un comportamiento para que el usuario pueda interactuar con los elementos de éste.

Imagen 160: Menú principal del juego

Como se dijo anteriormente, "Play Game" y "Quit" son objetos de tipo 3D Text. A ambos se les ha añadido un box collider y se les ha asignado un script común. Éste script diferencia un 3D Text del otro debido a que se ha definido una variable en el script llamada "is Quit Button" que identifica a cada uno de ellos. En la imagen 161 el script de control del comportamiento del menú principal.

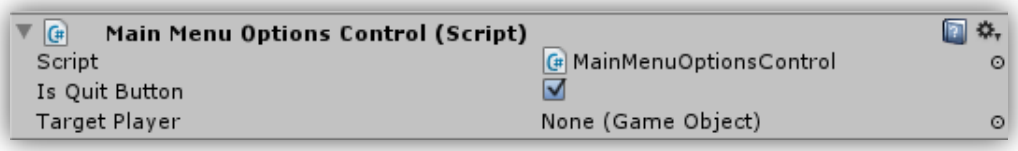


Imagen 161: Script de control del comportamiento del menú principal

En el botón "Quit" estará marcada la opción mientras que para el botón "Play Game" no lo estará.

El script utilizará tres funciones principales:

- **OnMouseEnter()*:*** Ésta función está atenta a cualquier entrada del mouse sobre la zona donde reside el collider de un elemento de la *GUI**, en este caso el 3D Text. Al acceder el mouse sobre el collider de "Play Game" o "Quit", ésta función será activada cambiando el color del texto a rojo quedando en rojo tal y como puede verse en la imagen 162.

Imagen 162: Selección de la opción deseada del menú

- **OnMouseExit():** Ésta función detecta cualquier salida del mouse sobre la zona donde reside el collider de un elemento de la GUI determinado, en este caso el 3D Text. Al ejecutarse esta función se volverá a cambiar el color del texto a color blanco.

- **OnMouseUpAsButton():** Ésta función se activa en el momento en que el usuario pulsa el mouse y lo suelta sobre un elemento de la GUI determinado. Al pulsar con el ratón sobre "Play Game" el Unity3D cargará la siguiente scene que contiene todo el juego, mientras que si se pulsa con el ratón en "Quit" la aplicación será cerrada.

-4.5.3: Escenario de combate (Iglesia)

La iglesia dispondrá de un script encargado de la apertura y el cierre de las puertas de la entrada delantera. Éste script será ejecutado cuando se detecte la cercanía a la puerta del

personaje principal. La iglesia también dispondrá de un componente AudioSource que contendrá el sonido de apertura y cierre de las puertas.

Imagen 163: Mesh collider de la puerta que impide el paso de objetos a través de ella

Como puede verse en la imagen 163, a la puerta se le ha aplicado un mesh collider que impedirá el acceso de objetos a través de ella.

Para poder realizar la apertura y el cierre de las puertas se ha colocado un sphere collider en la zona en la que reside la puerta de entrada a la iglesia. Puede verse la colocación del sphere collider en la imagen 164.

Imagen 164: Sphere collider de la puerta encargado de detectar la entrada del personaje principal en la iglesia. Se activa cuando éste entra dentro del propio sphere collider

Cada vez que el character collider del personaje principal entre en contacto con el sphere collider de la puerta saltará el trigger y ocurrirá lo siguiente:

- El trigger llamará al código para ejecutar la animación de apertura o cierre de las puertas.
- El trigger llamará al código para ejecutar el sonido de animación de apertura o cierre de las puertas.

-4.5.4: Personaje principal

El movimiento del personaje será controlado por el teclado. Éste contendrá como collider y controlador un character controller. También contendrá un AudioSource que será el encargado de gestionar todos los sonidos de las animaciones del personaje que se esté ejecutando. En cuanto a los script de éste, dispondrá de cuatro scripts principales llamados PlayerInput, PlayerMovement, PlayerAttack y PlayerSettings. A continuación se explica la función de cada uno de ellos.

Interacción de los scripts anteriores

En la imagen 165 se muestra el diagrama de interacción que poseen los scripts del personaje principal. Así como una pequeña definición del comportamiento de éstos. El script PlayerInput capturará todas las entradas de teclado generadas por el usuario que serán enviarán éstos al script PlayerMovement que generará la acción que corresponda y se comunicará con el script PlayerAttack para realizar cualquier ataque. PlayerSettings es un script del que PlayerMovmement heredará. Estarán definidas las características del personaje principal. A continuación se explicarán en detalle el funcionamiento de cada uno de estos scripts.

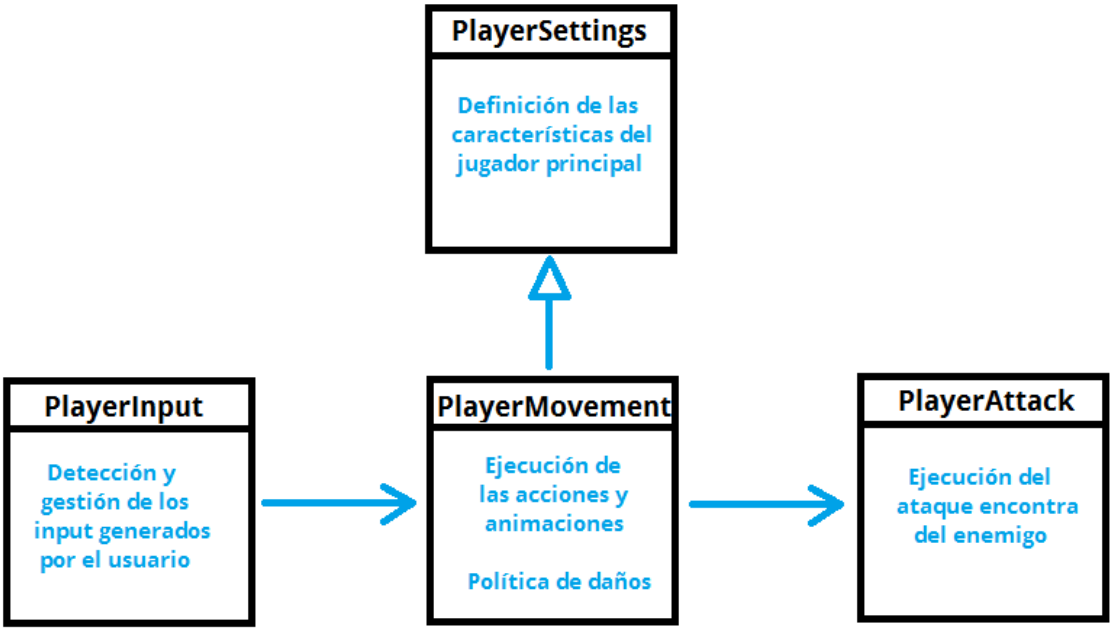


Imagen 165: Relación de los scripts del personaje principal

Script – PlayerInput

o Definición

Éste script tiene la única función de capturar todas las entradas de teclado realizadas por el usuario. En el momento en que se efectúa una entrada, se identifica qué tecla se ha pulsado y si está dentro del conjunto de teclas definidas en el Input Manager, se enviará un mensaje al script PlayerMovement indicando qué tecla ha sido pulsada.

o Funcionamiento del código

Para poder realizar la detección de entradas por teclado existen varias funciones. Puede ser que el Unity3D detecte una entrada por teclado cuando el botón sea pulsado o cuando sea pulsado y deje de estarlo. Para este script se ha utilizado el primero de ellos que puede verse en la imagen 166.

```
(Input.GetButton("Jump"))
```

Imagen 166: Detección de la tecla virtual definida Jump para saltar

Ha de definirse el botón virtual definido en el *Project Settings -> Input Manager*. Una vez se ha detectado la entrada se utiliza una función que permite la comunicación con todos los demás scripts de tipo MonoBehaviour que posea el mismo objeto, es decir, que se enviará a los cuatro scripts anteriormente citados ya que todos ellos lo son. La función encargada de realizar esa tarea se puede ver en la imagen 167.

```
SendMessage("JumpUp");
```

Imagen 167: Función encargada del envío de la acción de saltar del personaje al PlayerMovement

Nombre de una función que está codificada en el script PlayerMovement. De esta manera al utilizar esta función se llamará a una función de otro script llamada JumpUp.

A continuación se dispone de la imagen 168 donde puede visualizarse la el proceso de detección de una entrada de manera completa por teclado.

```
if(Input.GetButton("Jump")){
    SendMessage("JumpUp");
}
```

Imagen 168: Función completa encargada de detectar una entrada por teclado y mandar la información a al script encargado de accionar al personaje principal del juego

La manera en que se podría leer este código sería la siguiente:

"Si (el usuario ha pulsado la tecla saltar) Comunica que quieres saltar a todos los demás scripts del objeto".

Para todas y cada una de las entradas por teclado que se vayan a realizar se utiliza el mismo tipo de código, visto en la imagen anterior.

Script – PlayerMovement

o Definición

Éste script contiene la función de procesar absolutamente todas las acciones de movimiento recibidas desde el script PlayerInput y las animación del personaje. Extiende del script PlayerSettings.

o Funcionamiento del código

Máquina de estados

Este script funciona mediante una *máquina de estados**. Al iniciarse el script éste llama a la función Awake(), donde como ya se ha visto anteriormente es usada para inicializar ciertas variables que utilizará el propio personaje principal tales como su componente Character Controller o su AudioSource entre otras. Seguidamente llama a la función Start() donde un *bucle** infinito que contiene la una máquina de estados. Esta máquina de estados puede verse en la imagen 169.

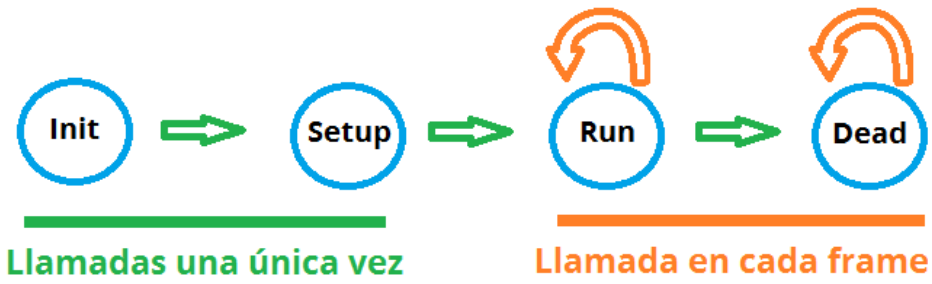


Imagen 169: Máquina de estados del script PlayerMovement

*Consulte el glosario de vocabulario situado en la parte final del documento

A continuación se explicará cada estado de la máquina de estados:

: Estado inicial de la máquina de estados el cual únicamente ejecutará una vez. En este estado se realiza una comprobación de si el personaje principal tiene asociados a éste un componente Character Controller y un componente Animation, el primero de ellos es requerido en el script para proseguir. Si no se dispone del componente Character Controller el script lo añadirá automáticamente, en cambio si no se dispone del componente Animation el script proseguirá y no ejecutará las animaciones del personaje cuando éste ejecute alguna acción. Una vez finalice este estado pasará al estado "Setup".

: Este estado también será ejecutado una única vez. En este estado se inicializa el vector de movimiento con el que se moverá el personaje principal y se preparará la iniciación de la ejecución de la animación del personaje llamada "Idle". También se declararán los valores iniciales de todas las variables usadas por el script. Una vez finalice este estado pasará al estado "Run".

: En este estado es donde el script contiene la mayor cantidad de las funciones utilizadas por éste. El script permanecerá en este estado que será llamado cada frame hasta que el personaje sea derrotado y será entonces cuando pase al estado "Dead". Es en este estado en el cual se mueve el personaje, efectúan los ataques, se calcula el daño del combo realizado mediante la política de daños y se ejecutan las distintas animaciones en función de las acciones recibidas desde el script PlayerInput.

: Este estado del script será llamado en cada frame una vez el personaje principal haya sido derrotado y hasta que el usuario vuelva al menú principal del juego. Una vez entre en este estado se ejecutará la animación de morir del personaje principal y no podrá efectuar ningún tipo de movimiento ni acción.

Combo de ataque y política de daños

Los ataques del personaje principal funcionan de la siguiente manera:

- El PlayerInput script detecta que ha habido una acción de ataque ya que el usuario ha pulsado uno de los dos botones del mouse.
- Hay un contador que cuenta la cantidad de ataques normales seguido que se han realizado (ataques con el botón izquierdo del ratón). Se ejecutará entonces la animación a la que corresponda el contador hasta un máximo de 3 ataques normales seguidos. Si se ejecuta un ataque fuerte (ataque con el botón derecho del mouse) se reiniciará el contador de ataques normales a 0 volviendo a empezar con el combo. También se reiniciará el contador si se realiza un cuarto ataque normal seguido, entonces éste cuarto ataque pasará a ser el primer ataque del nuevo combo.
- Los ataques normales basan su daño en función de la cantidad de ataques normales realizados. Si únicamente se ha realizado un ataque normal el daño será el más bajo posible, mientras que si se ha realizado tres ataques normales seguidos dentro de un período determinado de tiempo, el daño de éste tercer ataque será el más alto. Si se realiza un ataque fuerte, en función de la cantidad de ataques normales que se haya realizado previamente se causará un daño mayor o menor.
- Una vez realizado el ataque se calculará el daño generado por éste y se le enviará al enemigo.

Cosas a tener en cuenta sobre los ataques del personaje principal:

- Los ataques que se realicen no enviarán daño al enemigo a no ser que éste esté dentro del radio y del rango de visión del personaje principal. Puede verse el rango de visión en la imagen 170, lugar donde se producirá una afectación del ataque al enemigo.

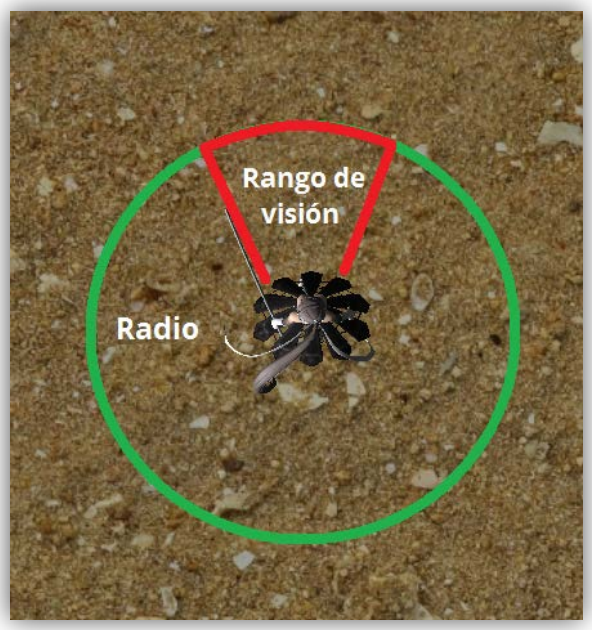


Imagen 170: Radio de ataque y rango de visión del personaje principal

- Existe un contador de tiempo que se activa una vez se ha finalizado un ataque, si el usuario no pulsa otra vez algún botón de ataque dentro de ese período de tiempo, el combo de ataque se reiniciará.
- Si el usuario realiza un ataque mientras el personaje ya está atacando, no se realizará caso alguno a esta acción. El usuario deberá pulsar el botón de ataque una vez la animación de ataque anterior haya sido finalizada.

Como se ha dicho anteriormente el personaje principal dispone de una combinación de ataques de cuatro movimientos distintos (tres ataques normales y uno fuerte). Es por ello que las distintas combinaciones del combo generarán distinta cantidad de daño. El personaje principal dispondrá entonces de los combos que pueden verse en la imagen 171.

Imagen 171: Combinación completa de ataques disponibles para el personaje principal

Script – PlayerAttack

o Definición

Éste script da soporte al script de PlayerMovement ya que es el encargado de ejecutar ciertas funciones que son llamadas por este.

o Funcionamiento del código

Éste script dispone de una función en encargada de comprobar la distancia que hay entre el personaje principal y el enemigo y si éste está dentro del rango de visión del personaje principal. También es el encargado de enviar el daño producido por los ataques del personaje principal al enemigo.

Script – PlayerSettings

- **Definición**

En este script se definen las características del personaje principal y es el encargado de gestionar la barra de vida de éste.

- **Funcionamiento del código**

Este script contiene todas las funciones que permiten definir y obtener los valores de ataque, vida actual, vida máxima, defensa y velocidad de movimiento del personaje principal. También es el encargado de gestionar la barra de vida y actualizarla en el momento de recibir daño o de curarse. El último cometido de este script es la gestión de la creación, activación y desactivación de las auras de fuego que aparecen alrededor del personaje principal cuando este es quemado por el fuego enemigo.

-4.5.5: Cámara del personaje principal

La cámara es controlada por el mouse y girará en torno al personaje mirando siempre a éste (manteniéndose a una distancia con el personaje siempre constante). Se moverá por los ejes X e Y.

Ésta cámara dispone de dos modos que podrán activarse in-game por el jugador apretando la tecla 

. Estos modos son:

- **Manual**: El jugador moverá en todo momento la cámara, ésta estará libre y podrá visualizarse el jugador en sus 360º. En la imagen 172 pueden verse dos vistas de cómo la cámara manual está situada por delante del personaje.

Imagen 172: Cámara manual del personaje principal

- **Automática**: La cámara girará automáticamente en función de la orientación del jugador situándose siempre a las espaldas de éste. En la imagen 173 pueden verse dos vistas de cómo la cámara automática está situada detrás del personaje.

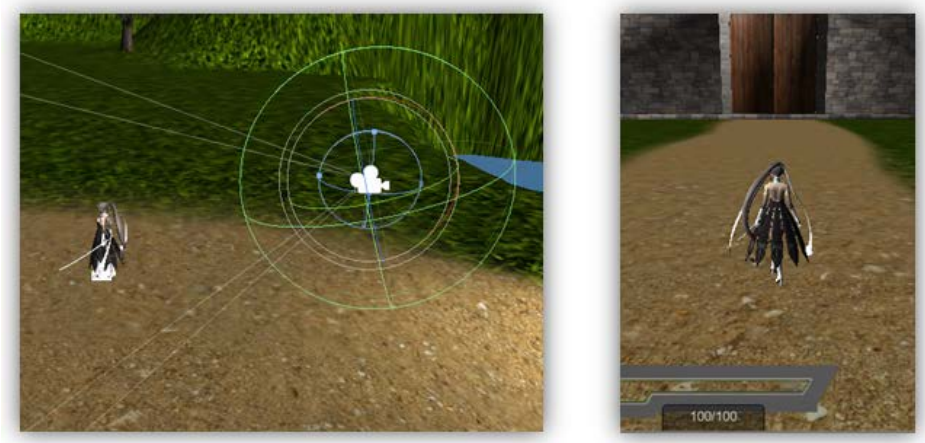


Imagen 173: Cámara automática del personaje principal

Funcionamiento de la cámara

La cámara dispone de un script que para el modo manual de la cámara, funciona de la siguiente manera:

- En cada fps se realiza una captura del valor de movimiento del mouse de los valores actuales de los ejes X e Y.
- Se busca la rotación que se ha generado mediante los valores de los ejes X e Y.
- Se calcula la nueva posición de la cámara. Para ello se multiplica la rotación obtenida por un vector que modifica la altura de la cámara y le da una distancia negativa. A todo esto se le suma la posición del personaje y de esta manera la cámara se sitúa a la altura y la distancia anteriormente citadas respecto al personaje.
- Finalmente se le aplica la posición y la rotación para la correcta visualización del personaje.

Puede verse una imagen representativa a continuación. La imagen 174 representa el estado inicial de la cámara manual mientras que la imagen 175 representa el estado final de la cámara manual al haber realizado un movimiento de rotación respecto al personaje.

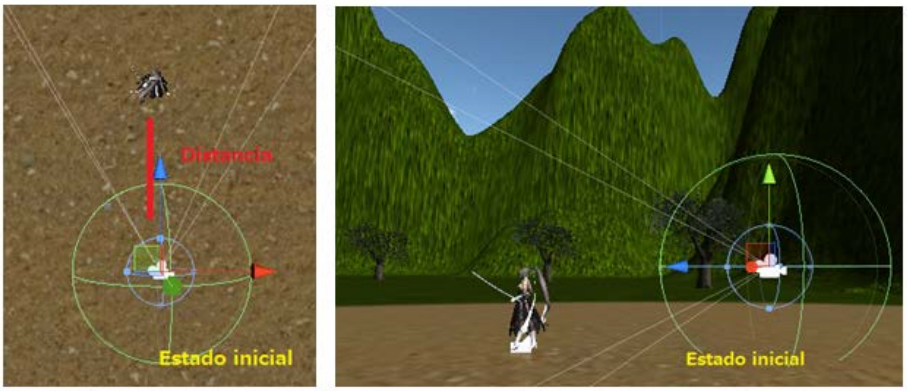


Imagen 174: Estado inicial de la cámara manual

Imagen 175: Estado final de la cámara manual después de realizar un movimiento de ésta

Para la cámara automática se utiliza un script que posee Unity3D en JavaScript llamado "SmoothFollow" que ha sido transformado a C# para su uso y que a diferencia de la cámara manual, la automática busca en todo momento la rotación que ha realizado el personaje principal y se la asigna a ella misma como rotación objetivo a conseguir, manteniéndose siempre detrás del personaje principal.

Nota importante:

El script de la cámara utiliza la función del Update() y LateUpdate(), es decir, como ya se vio anteriormente, éstas dos funciones son llamadas en cada frame y Update() es llamada antes que LateUpdate().

El motivo de utilizar la función LateUpdate() en el script de la cámara es que ésta estará siguiendo a objetos que previamente se habrán movido dentro de la función Update(). Normalmente todos los objetos se mueven en esta función. Es por eso que la cámara deberá primero conocer la posición final del objeto al que sigue y a continuación moverse hacia él.

4.6 – Creación del ejecutable del juego

Finalmente, una vez que todos los componentes sean creados, puestos en escena y programados, queda la última cosa, crear el ejecutable para el juego que permitirá la ejecución en las distintas plataformas. Para ello se deberá ir a *File -> Build Settings*.

Aparecerá entonces la pantalla que puede verse en la imagen 176, una ventana que permite la compilación del código en distintas plataformas y distintas arquitecturas de ordenadores.

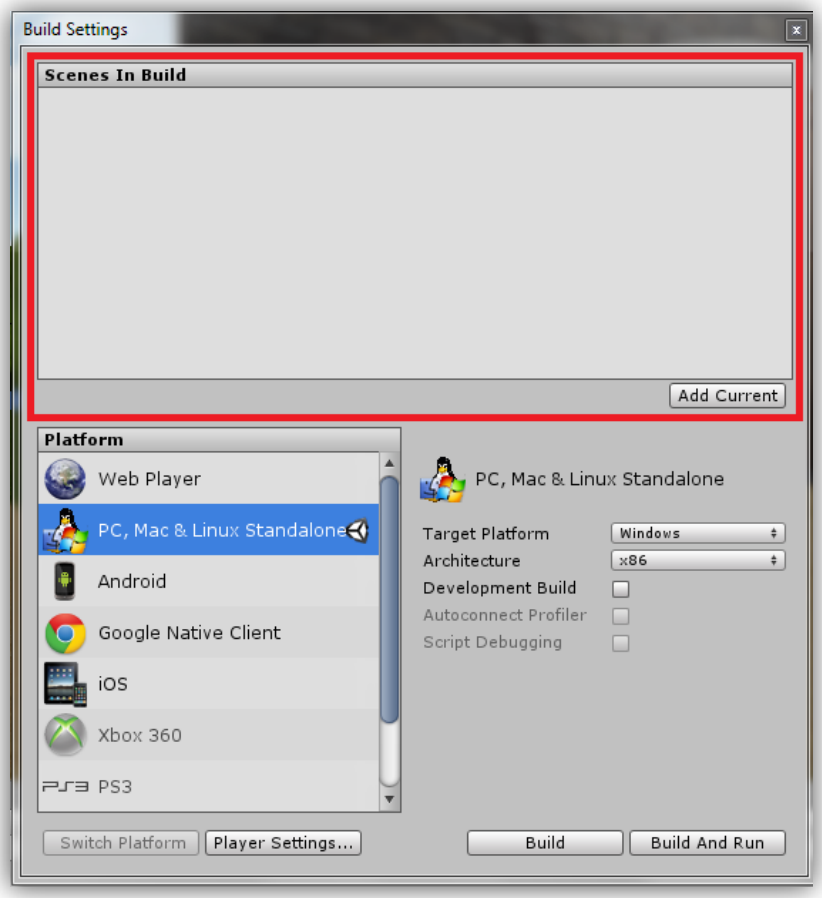


Imagen 176: Estado inicial de la cámara manual

En el recuadro  se realizar un drag&drop de las scene que se desee añadir al juego. En este proyecto se introducirán dos, el menú principal "Main Menu" y el juego "Eternal_Destiny". Puede verse las dos escenas contenidas en el juego en la imagen 177.

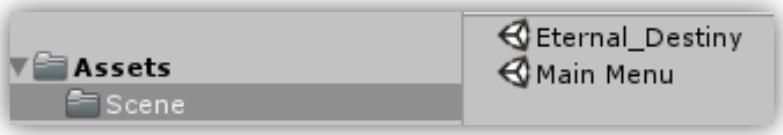


Imagen 177: Scene que van a ser añadidas para la compilación y creación del ejecutable del juego

Al realizar el drag&drop serán añadidas con un número que las identifica. Puede verse el número en la redonda en la imagen 178. El número les será añadido en función del orden de añadido.

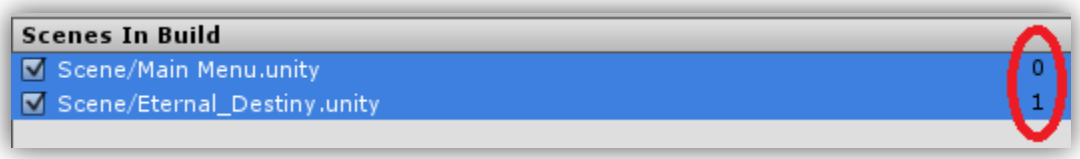


Imagen 178: Scene añadidas a la lista e identificadas listas para ser compiladas

La importancia de estos números recae en que serán el identificador utilizado para cargar la scene desde el código del script. En este proyecto en la scene "Main Menu", al seleccionar la opción de "Start Game" se ejecutará sentencia de código que puede visualizarse en la imagen 179.

```
Application.LoadLevel(1);
```

Imagen 179: Código que permite la carga de una scene determinada por su identificador de esta

Se realizará la carga de la scene con identificador número 1 que pertenece a la scene "Eternal_Destiny", el juego.

Para acabar se deberá pulsar la opción "Build" y se dispondrá del ejecutable del juego.

5 - TESTING:

En esta parte se analizará el estado final del los distintos elementos creados en el juego así como sus posibles mejoras. También, se añadirán videos en los puntos necesarios.

5.1 – Escenario Principal

El escenario final fue reducido respecto al que se originalmente se quiso realizar. En la imagen 180 puede verse la distribución que se realizó finalmente. En la imagen de a continuación puede verse en la redonda el área donde aparecerá el personaje cada vez que inicie el juego. En el recuadro puede verse la iglesia, la zona de combate entre el jefe final y el personaje principal. Finalmente en el recuadro contiene el tesoro accesible una vez se haya eliminado al jefe final.

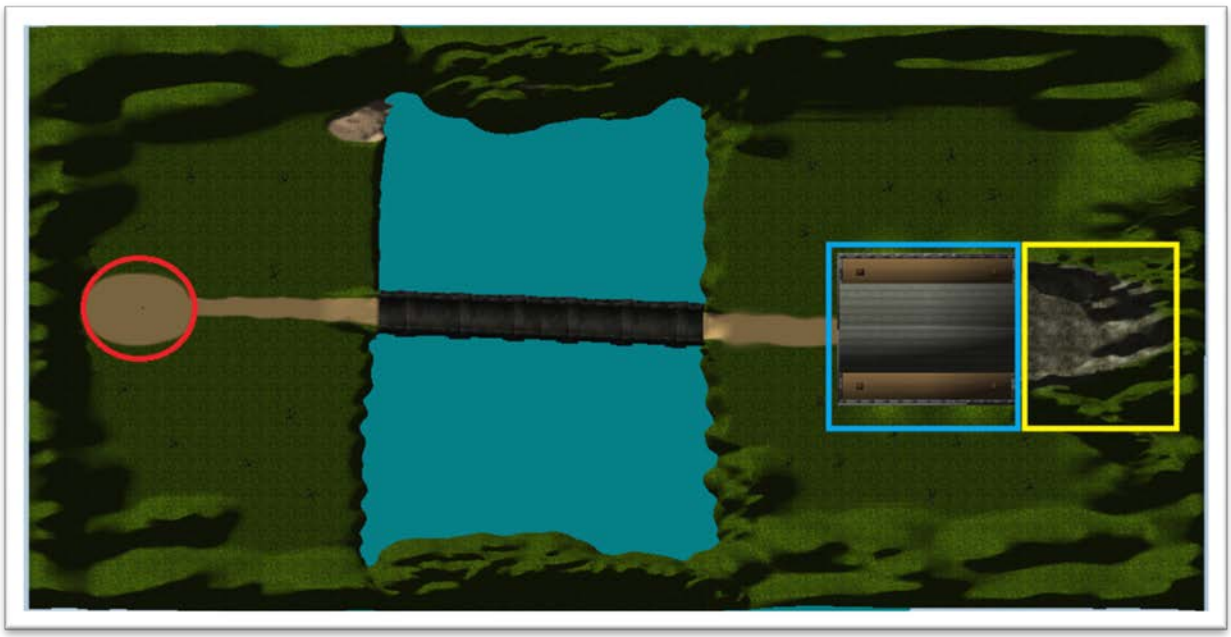


Imagen 180: Escenario principal con la distribución del personaje principal, iglesia y tesoro

En la imagen 181 puede verse el escenario desde otra perspectiva donde se aprecia la profundidad de éste.

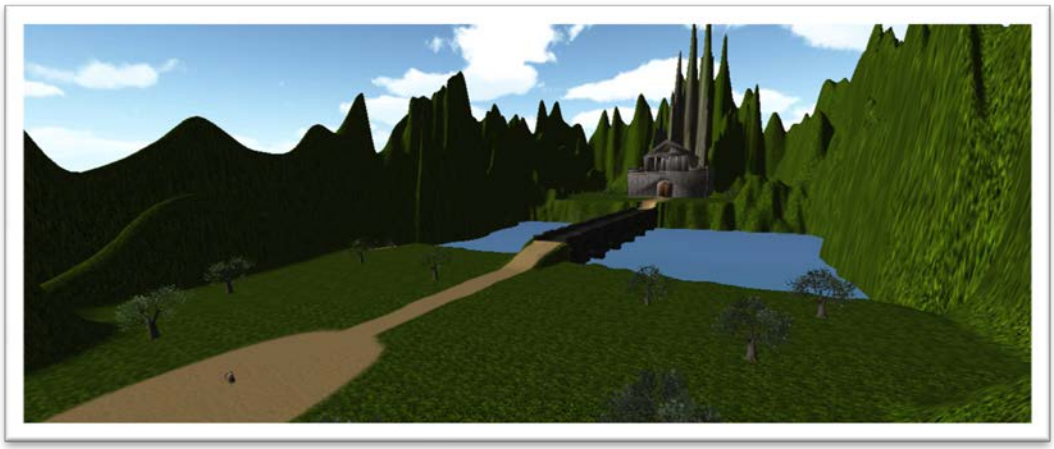


Imagen 181: Escenario principal con la distribución del personaje principal, iglesia y tesoro

5.2 – Menú principal

Finalmente, al menú principal se le introdujeron dos de las cuatro opciones que inicialmente se pensó añadir. Se puso la opción de "Play Game" que permite el inicio de una partida nueva y la opción de "Quit" que permite salir de la aplicación. Las otras dos opciones que se querían introducir (véase en el apartado 3: Análisis) eran: "Opciones" y "Continuar Partida". La primera es ahora una opción gestionada por el menú de pausa y es accesible a esta durante el juego. Respecto a la segunda no se ha podido añadir debido a que el juego no dispone de ningún tipo de estado de guardado. Véase en la imagen 182 el estado final del menú principal con las opciones explicadas anteriormente y el personaje principal que ejecuta una animación cuando se selecciona la opción de "Play Game".

Imagen 182: Menú principal finalizado

5.3 – Escenario de combate (Iglesia)

Se puede decir que en cuanto a la iglesia, ha habido un gran acierto con respecto a la idea principal que se tenía de ésta y las proporciones de todos los elementos de la iglesia. Se ha podido conseguir lo esperado tanto a nivel exterior como a nivel interior. Véase la imagen 183 donde se puede apreciar el estado final de la iglesia dentro del programa de modelación Blender.

Imagen 183: Iglesia finalizada con el programa Blender

En la imagen 184 puede apreciarse la iglesia en su estado final una vez importada al Unity3D. Como puede verse la importación al engine se realizó de manera totalmente satisfactoria.

Imagen 184: Iglesia finalizada e importada a Unity3D

5.4 – Personaje principal

- 5.4.1: Estado final

Finalmente la utilización del personaje principal en el juego se ha podido realizar sin ningún tipo de problema. Véase en la imagen 185 el personaje principal en su estado final en Blender (izquierda). Y el personaje en su estado final al ser importado a Unity3D (derecha).

Imagen 185: Iglesia finalizada e importada a Unity3D

- 5.4.2: Movimientos básicos

Demostración de los movimientos básicos del personaje principal del juego.

Parado (Idle)

Acción ejecutada cuando el personaje principal se encuentra parado (sin ninguna entrada por teclado del usuario) siempre y cuando no esté en contacto con el agua del escenario principal. En el video 1 puede verse la ejecución dicha acción en Unity3D, versión in-game.

Video 1: Animación de idle del personaje ejecutada en el engine Unity3D

Correr (Run)

Acción ejecutada cuando el personaje avanza o retrocede siempre y cuando el modo de correr esté activado. En el video 2 puede verse la ejecución de dicha acción en Unity3D, versión in-game.

Video 2: Animación de correr del personaje ejecutada en el engine Unity3D

Caminar (Walk)

Acción ejecutada cuando el personaje avanza o retrocede siempre y cuando el modo de correr esté desactivado. En el video 3 puede verse la ejecución de dicha acción en Unity3D, versión in-game.

Video 3: Animación de caminar del personaje ejecutada en el engine Unity3D

Saltar (Jump)

Acción ejecutada cuando el personaje realiza un salto. En el video 4 puede verse la ejecución de dicha acción en Unity3D, versión in-game.

Video 4: Animación de salto del personaje ejecutada en el engine Unity3D

Nadar (Swim)

Acción ejecutada cuando el personaje entra en contacto con el agua y se mueve por ella. En el video 5 puede verse la ejecución de dicha acción en Unity3D, versión in-game.

Video 5: Animación de nado del personaje ejecutada en el engine Unity3D

Caer (Fall)

Acción ejecutada cuando el personaje no está en contacto con el suelo durante un periodo de tiempo determinado. En el video 6 puede verse la ejecución de dicha acción en Unity3D, versión in-game.

Video 6: Animación de caída del personaje ejecutada en el engine Unity3D

Morir (Death)

Acción ejecutada cuando la vida del personaje llega a cero. En el video 7 puede verse la ejecución de dicha acción en Unity3D, versión in-game.

Video 7: Animación de muerte del personaje ejecutada en el engine Unity3D

- 5.4.3: Ataques y combos

Demostración de los ataques y el combo del personaje principal del juego.

Primer ataque físico (1st Attack)

Primer ataque del combo que será ejecutado. En el video 8 puede verse la ejecución de dicha animación del primer ataque físico en Unity3D.

Video 8: Animación del primer ataque físico del personaje ejecutado en el engine Unity3D

Segundo ataque físico (2nd Attack)

Segundo ataque del combo que será ejecutado una vez se haya ejecutado el primer ataque físico. En el video 9 puede verse la ejecución de la animación de dicho ataque en Unity3D.

Video 9: Animación del segundo ataque físico del personaje ejecutado en el engine Unity3D

Tercer ataque físico (3rd Attack)

Tercer ataque del combo que será ejecutado una vez se haya ejecutado el segundo ataque físico. En el video 10 puede verse la ejecución de la animación de dicho ataque en Unity3D.

Video 10: Animación del tercer ataque físico del personaje ejecutado en el engine Unity3D

Cuarto ataque físico (4th Attack)

Cuarto y último ataque físico que marcará el final del combo. En el video 12 puede verse la ejecución de la animación del cuarto y último ataque físico en Unity3D.

Video 12: Animación del cuarto ataque físico del personaje ejecutado en el engine Unity3D

Combo (Combo Attack)

La combinación de todos estos ataques de forma continuada genera un combo. Finalmente se decidió hacer el combo con cuatro ataques en lugar de los cinco que se tenía pensado. En el video siguiente se muestra el combo realizado por el personaje principal. En el video 13 puede verse la ejecución de la animación del quinto ataque físico en Unity3D.

Video 13: Animación del combo completo en el engine Unity3D

- 5.4.4: Posibles mejoras

En cuanto a los movimientos básicos del personaje principal, se podría haber mejorado todo el movimiento del traje y el pelo de éste, entonces el movimiento parecería mucho más real. La realización de esta tarea requería de una extensión del esqueleto del personaje también hacia el pelo y el traje de éste y su posterior pintado de cuerpo para su uso.

Con respecto a los ataques, se le podría haber dado unos movimientos con un rango de acción más pronunciado para que fueran visualmente más atractivos.

5.5 – Cámara del personaje principal

- 5.5.1: Cámara manual

Con éste tipo de cámara se ha permitido visualizar al personaje principal en sus 360 grados de manera satisfactoria. En el video 14 se muestra el resultado del funcionamiento de la cámara manual.

Video 14: Funcionamiento de la cámara muanual

- 5.5.2: Cámara automática

Con éste tipo de cámara ha permitido seguir el movimiento del personaje principal sin ninguna interacción del usuario con la cámara de manera satisfactoria. En el video 15 se muestra el resultado del funcionamiento de la cámara automática.

Video 15: Funcionamiento de la cámara automática

- 5.5.3: Posibles mejoras

Tanto para la cámara en su versión manual como para su versión automática, la única característica añadida que se le podría haber añadido es la de poder colisionar contra los objetos del entorno.

5.6 – Juego

- 5.6.1: Inicio del juego y primeros pasos

Una vez ejecutemos el juego, aparecerá el icono del motor de éste, Unity3D, y seguidamente el menú principal del juego mostrando el título de éste en la parte superior *Eternal Destiny*. Se podrá escoger entre las opciones de *Start Game* o *Quit Game.*

- **Start Game:** Nos permite empezar un juego

- **Quit Game:** Nos permite cerrar la aplicación

Una vez iniciemos aparecemos en el escenario principal. En el video 16 se muestra el inicio del juego y los primeros pasos recorriendo el mapa.

Video 16: Inicio del juego y primeros pasos por el mapa

- 5.6.2: Juego fallido

En el caso que el monstruo jefe final nos derrote, habrá una fallida del juego y deberemos volver a empezar el juego. En el siguiente video 17 se muestra la derrota y muerte del personaje principal del juego.

Video 17: Derrota y muerte del personaje principal del juego

- 5.6.3: Juego pasado

En el caso que venzamos al monstruo jefe final, se abrirá la puerta protegida por la pared de fuego y nos dará acceso al tesoro final que nos dará la victoria. En el video 18 se muestra la derrota del jefe final del juego y la obtención del tesoro.

Video 18: Derrota y muerte del personaje principal del juego

6 – CONCLUSIONES

Finalmente puede decirse que se han podido cumplir gran parte de los objetivos que se marcaron en un inicio. El proyecto inicialmente pensado era bastante grande para el tiempo del que se disponía, así que conforme se iba aprendiendo a utilizar las nuevas tecnologías aplicadas, se optó por crear una parte de éste e intentar llegar con esta parte al máximo exponente de jugabilidad.

En cuanto al uso de las nuevas tecnologías utilizadas, comenzando al programa Blender, encargado de todo el modelaje y animación se puede decir que se ha conseguido adquirir unos conocimientos de uso avanzados siendo capaz de modelar y animar al completo distintos objetos aprovechando todas las posibilidades que ofrece el programa.

Asimismo, en cuanto al engine Unity3D encargado de procesar todo el juego puede decirse que también se ha logrado adquirir unos conocimientos avanzados del uso de éste, siendo capaz de aprovechar gran parte de las posibilidades que éste ofrece.

El juego final ha resultado ser un juego completo con un inicio y un final bien definidos y con una buena jugabilidad.

7 – BIBLIOGRAFÍA

- http://www.blender.org/
- http://unity3d.com/learn
- http://docs.unity3d.com/Documentation/ScriptReference/
- http://www.katsbits.com
- http://cgcookie.com/unity/temp/
- http://forum.unity3d.com/forum.php
- http://www.unity3dstudent.com/
- http://www.3dbuzz.com/
- http://www.gamedev.net/page/index.html
- http://www.blender.org/

8 – MANUAL DE USUARIO

A continuación se mostrará el manual del usuario donde se detallará el proceso a seguir para poder jugar al juego

Controles del personaje:

En la imagen 186 se muestra un teclado con el conjunto de las teclas marcadas que serán utilizadas para el juego.

Imagen 186: Conjunto de teclas usadas en el juego

Movimiento del personaje

Cambio de modo correr – caminar y viceversa

Saltar

Controles de la cámara:

En la imagen 187 se muestran los botones de un ratón que ejecutarán los distintos ataques del personaje principal.

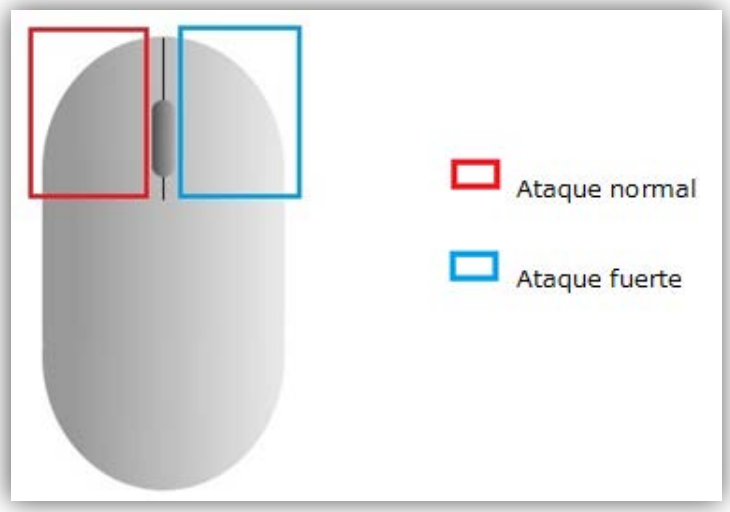


Imagen 187: Botones del ratón utilizados para realizar el ataque del personaje principal

Ataques combo:

Lista de combinaciones de ataque disponibles.

- Combo 1: ▭, ▭
- Combo 2: ▭, ▭, ▭
- Combo 3: ▭, ▭, ▭, ▭

Ejecución del juego:

Una vez se haya ejecutado el ejecutable del juego, aparecerá una ventana en la cual se podrán modificar características de vídeo a la que se ejecutará y los controles del personaje.

-Configuración

Véase la imagen 188 donde puede visualizare la pantalla comentada anteriormente.

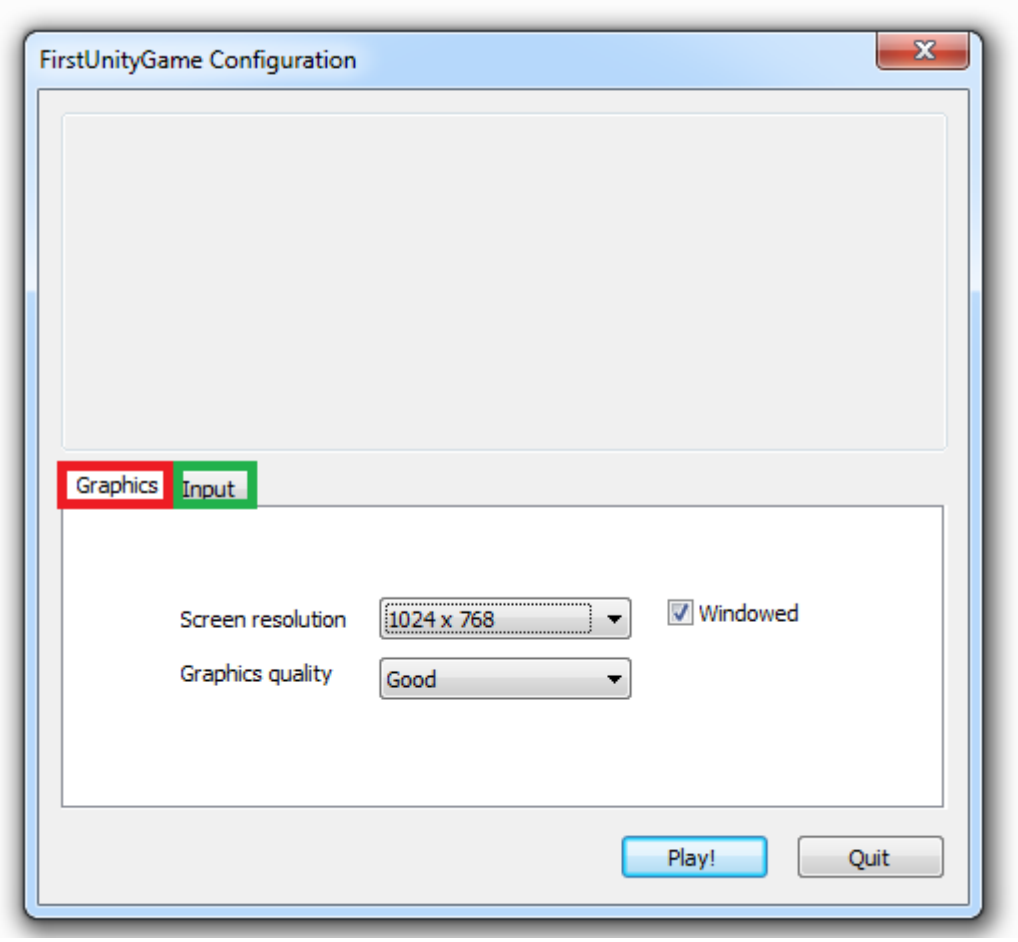
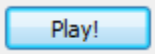


Imagen 188: Botones del ratón utilizados para realizar el ataque del personaje principal

En el recuadro  se elegirá la resolución a la que se desea ejecutar el juego, la calidad de gráficos deseada y si se quiere ejecutar en modo ventana o de lo contrario en pantalla completa.

Por otro lado, en el recuadro  estarán definidos todos los controles predeterminados del juego los cuales podrán ser modificados al gusto del usuario.

Al seleccionar la opción 

se ejecutará el juego con las opciones escogidas anteriormente.

-Menú

La ventana mostrada en la imagen 189 se dispondrá de dos opciones que permitirán realizar lo siguiente:

- **Play Game** – Iniciar una partida nueva.
- **Quit** – Salir de la aplicación.

Imagen 189: Opciones del menú principal

-Menú de pausa

Una vez dentro de la partida, pulsando la tecla 

aparecerá el menú de pausa. En la imagen 190 puede visualizarse el menú de pausa.

Imagen 190: Menú de pausa del juego

Dispone de las siguientes opciones:

- **Continue** – Permite continuar el juego.
- **Options** – Permite cambiar las opciones de audio y gráficas del juego.
- **Cheats** – Permite ver las teclas que activan los trucos del juego.
- **Salir** – Permite salir del juego.

-Juego

En la imagen 191 de a continuación se muestra una imagen in-game del juego mientras el personaje principal realiza un combate con el jefe final del juego. El recuadro ▭ indica el

nivel de vida del personaje principal mientras que el recuadro indica el nivel de vida del jefe final del juego.

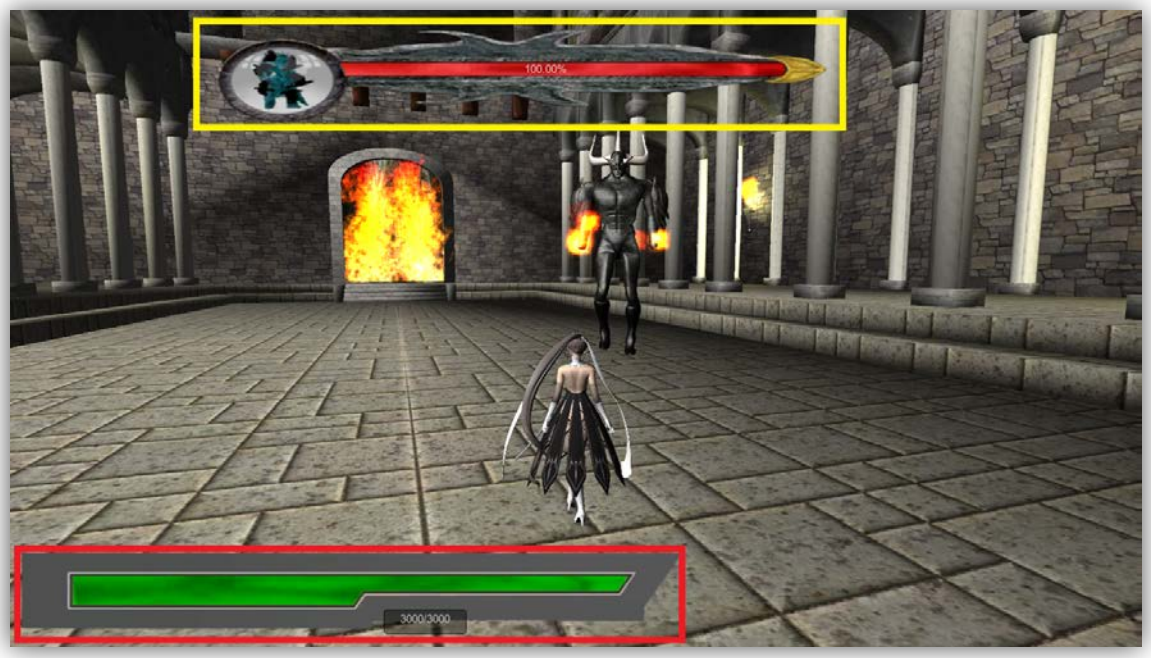


Imagen 191: Imagen del juego in-game con las vidas del personaje principal y el jefe final

9 – GLOSARIO

- **Script:** Archivo de texto que contiene código de programación que da un comportamiento determinado al objeto al que se aplique.
- **Engine:** Motor gráfico encargado de procesar el juego.
- **In-game :** Término que hace referencia a el momento en que se está jugando al juego.
- **C Sharp:** Lenguaje de programación
- **JavaScript:** Lenguaje de programación.
- **Renderización:** Generación de una imagen o vídeo mediante el cálculo de la iluminación indirecta partiendo de un modelo 3D.
- **Python:** Lenguaje de programación.
- **ActionRPG:** Juego de combate a tiempo real que combina elementos de rol.
- **Boss:** Término utilizado en videojuegos que identifica al enemigo final del nivel.
- **Gameplay:** Jugabilidad del juego.
- **Shortcuts:** Tecla o secuencia de teclas que efectúan una acción del programa definida previamente.
- **Keyframes:** Conjunto de posiciones de las partes del esqueleto en un determinado instante de tiempo que define una posición del objeto. Usado para animar.
- **Add-on:** pieza de software descargado e instalado que potencia y enriquece el uso de la aplicación a la que va destinado.
- Layers: Conjunto de capas del programa Blender que permiten una división del modelaje de un objeto en varias partes.
- **Num pad:** Panel numérico del teclado.
- GameObjects: Término usado para llamar a todos los objetos utilizados en Unity3D.
- Scripting: Acción de programar scripts.
- **API:** Interfaz de programación de aplicaciones. Es el conjunto de funciones y procedimientos que ofrece una biblioteca para la programación en un cierto lenguaje.
- **Assets:** Objeto o script realizado por un tercero que puede ser añadido a Unity3D.
- **Mesh:** Cuerpo del objeto.

- **Offset :** valor de separación .
- **Mapear:** Método para añadir texturas a un objeto 3D mediante la aplicación de texturas 2D.
- **Background:** fondo de la pantalla.
- **Drag&Drop:** Acción de mover (arrastrar) objetos con el ratón objetos de una ventana a otra del ordenador.
- **Tag:** Identificador de un elemento o un conjunto de ellos dentro del juego. Varios elementos pueden compartir un mismo tag.
- **Monobehaviour:** Clase madre de la que derivan todos los scripts de Unity3D. Heredan todas las funciones que tiene definidas para su posterior uso.
- **Máquina de estados:** Modelo de comportamiento de un sistema, que define un comportamiento de éste.
- **Bucle:** Sentencia de programación que se realiza varias veces sobre un trozo de código aislado del código.