Prototipo de realidad virtual para el diseño de edificios simples con Unreal Engine

Trabajo Final de Grado

Autor: Sergio García Vera
Tutor: Agustín Rafael Trujillo Pino

Grado en Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

21 de septiembre de 2019

Resumen

Este TFG pretende explorar la aplicación de la realidad virtual en el proceso de diseño de edificios, realizando una aplicación que permita a profesionales y al público general diseñar edificios completamente desde un entorno en realidad virtual.

Para ello se utilizará el motor Unreal Engine 4 y el sistema de realidad virtual Oculus Rift. Los usuarios podrán diseñar la estructura completa del edificio, y, una vez estén satisfechos con el diseño estructural podrán comenzar a decorar las diferentes habitaciones del edificio con muebles, figuras y todo tipo de objetos.

Además, los usuarios podrán realizar bocetos en 3D que se integran con el diseño, ya sea para apuntar notas o dar rienda suelta a su imaginación y crear diseños propios.

Abstract

This Final Degree Project aims to explore the application of virtual reality in the building design process by making an application that allows professionals and the general public to design buildings completely from a virtual reality environment.

For this, the Unreal Engine 4 game engine and the Oculus Rift virtual reality system will be used. Users will be able to design the complete structure of the building, and once they are satisfied with the structural design, they will be able to start decorating the different rooms of the building with furniture, figures and all kind of objects.

In addition, users can make 3D sketches that integrates with the design, either to write notes or to unleash their imagination and create their own designs.

Índice general

Bibliografía

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Durante los últimos años ha resurgido una tecnología que hasta hace poco se daba por olvidada y que ha revolucionado todos los sectores a los que se ha aplicado, desde el sector educativo hasta el sector médico, ofreciendo una forma más natural de interactuar con el entorno virtual, y abriendo una puerta al consumo de todo tipo de contenido inmersivo. Ésta tecnología es la realidad virtual.

Además, gracias a los avances de la informática de consumo, cualquiera que posea un ordenador medianamente potente puede disfrutar de esta tecnología que hasta hace poco tiempo parecía ciencia ficción. Y si además le añadimos la amplia oferta de motores gráficos disponibles para crear cualquier tipo de aplicación inmersiva, la dificultad y el coste de desarrollar aplicaciones en realidad virtual se reducen considerablemente.

Pero como ocurre durante los primeros años de vida de todas las tecnologías emergentes, no hay aún un estándar definido de qué funciona y qué no funciona a la hora de crear aplicaciones de realidad virtual. Por lo que los desarrolladores tienen la complicada tarea de abrir el camino a los futuros desarrolladores, arriesgándose con diseños que pueden ser un éxito o fallar, y atrayendo a nuevos usuarios con la mayor variedad posible de aplicaciones.

Esa necesidad de exploración de un nuevo medio es la base de este proyecto, que pretende aplicar las características de una aplicación en realidad virtual al proceso de diseño de edificios. Aprovechando para ello todas las ventajas de esta tecnología, como pueden ser la interacción natural o la representación a escala real de los edificios diseñados.

1.1. Motivación inicial

La principal motivación a la hora de realizar este TFG es explorar las distintas posibilidades que ofrece la realidad virtual a la hora de interactuar con los entornos digitales, desde la interacción con menús hasta la selección de objetos en el mundo virtual. Y crear un prototipo de aplicación que saque el máximo partido de las características que ofrece la realidad virtual, a la vez sea funcional y útil.

Por otro lado, también se pretende conocer el funcionamiento del motor Unreal Engine 4. Uno de los motores más galardonados en el mundo de los videojuegos, y

que además es totalmente gratuito. Además, este motor utiliza el lenguaje C++, un lenguaje que no se ha utilizado mucho a lo largo de la carrera por lo que es un reto interesante trabajar con él.

Finalmente, se pretende hacer una aplicación funcional y que tenga una aplicación real en el sector al que está dirigida. Y, tras valorar varias posibilidades, se ha decido apostar por una aplicación de diseño de edificios. Por un lado, porque este tipo de aplicaciones pueden verse muy favorecidas en realidad virtual, ya que el usuario podría ver sus diseños en escala real, y por otro, porque era un concepto de aplicación innovador, y que permitiría utilizar varios de los módulos de Unreal Engine como pueden ser las interfaces de usuario o el sistema de Blueprints.

En conclusión, la principal motivación de este TFG es aprender a utilizar un motor gráfico desde cero. Y una vez que se tengan unas nociones básicas de su funcionamiento, aplicar todo lo aprendido para desarrollar una aplicación funcional en realidad virtual.

1.2. Objetivos del proyecto

Este TFG tiene como objetivo desarrollar un prototipo de aplicación de diseño de edificios en realidad virtual, utilizando para ello el motor gráfico Unreal Engine 4 y el sistema de realidad virtual Oculus Rift.

La aplicación desarrollada permitirá al usuario diseñar la maqueta de un edificio en un entorno en realidad virtual, permitiéndole rotar y escalar la maqueta para verla desde cualquier ángulo, y crear la estructura básica del edificio. Por otro lado, el usuario podrá 'entrar' en la maqueta, para poder visualizar en tamaño real la estructura del edificio y colocar los muebles y objetos que decorarán las habitaciones de éste.

Por lo que el objetivo de este trabajo es crear una aplicación cuyas principales funcionalidades son:

- Crear la maqueta de un edificio en realidad virtual, en la que se puede posicionar y rotar todos los elementos que componen la estructura de un edificio, y rotar y escalar la propia maqueta.
- 'Entrar' en la maqueta, permitiéndole al usuario ver en tamaño real el edificio creado.
- Un sistema de movimiento con el que el usuario se desplazará por el edificio.
- Añadir objetos a las habitaciones para decorarlas. Pudiéndolos rotar, escalar y posicionar según el gusto del usuario.
- Crear bocetos 3D en el espacio virtual que sirvan de ayuda al usuario a la hora de diseñar el edificio.
- Generar una interfaz de usuario que permita seleccionar las opciones de la aplicación, como cambiar entre la vista de maqueta y la vista en tamaño real, seleccionar y editar los objetos que se van a colocar en el edificio o seleccionar la herramienta de dibujo de bocetos.

- Guardar y cargar los diseños creados por el usuario.

1.3. Estado del arte

1.3.1. Aplicaciones de diseño de edificios en VR

La aplicación a desarrollar se encuentra en un mercado que está en pleno crecimiento, y debido a esta razón las aplicaciones que se encuentran disponibles están sentando las bases de qué funciona, y que no funciona, a la hora de trasladar el diseño de edificios a la realidad virtual.

Entre esas aplicaciones podemos encontrar dos tipos, las que permiten al propio arquitecto crear desde cero el edificio en realidad virtual, y las que son un servicio de empresas que se encargan de transformar un plano o un edificio ya existente en un entorno hiperrealista que se puede explorar en VR.

Dentro del primer grupo destaca la aplicación Prospect Pro [1], de la empresa Iris Vr. Prospect Pro permite importar e integrar archivos 3D dentro del entrono virtual, e ir creando el edificio a partir de estos. Por otro lado, algunas de las funcionalidades que ofrece Prospect Pro son las siguientes:

- Un sistema que permite realizar anotaciones 3D en el entorno virtual que sirven de guía a los arquitectos durante el proceso de diseño de un edificio.
- Un sistema de capas que permite agrupar varios elementos del edificio y controlar sus características, permitiendo clasificar y gestionar de manera rápida los objetos y estructuras que se encuentran en el edificio.
- Un sistema multijugador, que permite que varios diseñadores trabajen al mismo tiempo en el mismo edificio.

Figura 1.1: Ejemplo de diseño realizado con Prospect Pro.

Dentro del segundo grupo destaca la aplicación REinVR (Real State in VR) [2] que consiste en un servicio que permite a los arquitectos y empresas inmobiliarias transformar los planos o edificios existentes en experiencias fotorrealistas, y presentar el edificio a posibles compradores de una forma más realista, aprovechando las virtudes de la realidad virtual.

Figura 1.2: Ejemplo de diseño fotorrealista realizado con REinVR.

Todas las aplicaciones de diseño de edificios en VR tienen algo en común, aceleran el proceso de diseño de edificios y ahorran costes, ya que permiten al diseñador ver una representación muy cercana a la realidad del estado de su diseño, y detectar posibles errores en fases tempranas de este, que más adelante podrían haber tenido un impacto importante en el coste de la construcción del edificio.

En ambas aplicaciones destaca el uso de un sistema de teletransporte como sistema de movimiento, ya que esto reduce significativamente las molestias y mareos que pueden ser causados a los usuarios al moverse por el mapa.

1.3.2. Aplicaciones de dibujo en VR

Debido a que una de las funcionalidades principales que se pretenden implementar en la aplicación es la realización de bocetos en 3D, es importante revisar el estado de las aplicaciones que toman esta funcionalidad como una parte importante de su diseño.

Al contrario de las aplicaciones de diseño de edificios, que están dirigidas al sector profesional, las aplicaciones de dibujo en VR suelen estar dirigidas al sector de consumo. Prueba de ello lo podemos encontrar en aplicaciones como Tilt Brush o Medium, que se encuentran disponibles en plataformas de videojuegos convencionales como Steam u Oculus Store. Entre todas las aplicaciones de dibujo en realidad virtual, la que mayor repercusión ha causado es Tilt Brush.

Tilt Brush es una aplicación creada por Google [3] que permite a los usuarios dibujar sobre un lienzo virtual, poniendo a su disposición una gran variedad de pinceles diferentes, desde cortinas de humo a copos de nieve, permitiendo a los creadores dar rienda suelta a su creatividad.

Entre sus principales características destacan:

- Dos formas de funcionamiento, principiante y avanzado, ajustándose de esta manera a todo tipo de usuarios.
- Un sistema que permite 'clavar' los objetos en una posición para evitar modificarlos por accidente.

- Un sistema de espejo que duplica los trazados del artista para crear diseños simétricos.
- Un sistema de selección de trazados que permite al usuario modificar o eliminar un trazado ya creado.

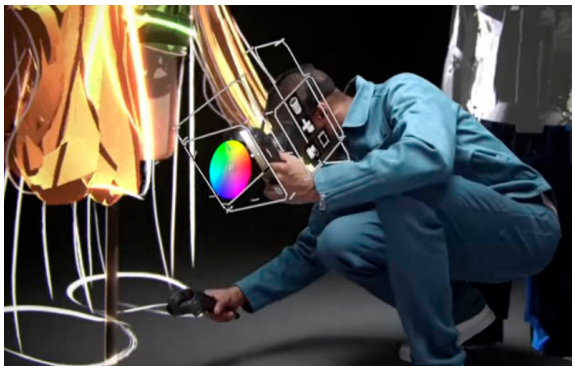


Figura 1.3: Interfaz de Tilt Brush.

Como conclusión, y tras haber analizado las aplicaciones que son referentes en ambos sectores, la realidad virtual es una plataforma a la que aún le cuesta descubrir qué funciona y qué no, y en la que hay una gran ventana para la innovación. Por otro lado, la aplicación que se pretende desarrollar cubre necesidades reales de los usuarios, como puede ser agilizar el proceso de diseño de edificios, o dar rienda suelta a la creatividad de los diseñadores, permitiéndoles probar distintos diseños en un entorno completamente controlado.

1.4. ¿Qué es la realidad virtual?

Ya hemos hablado de las virtudes que presenta la tecnología de realidad virtual. Pero, ¿qué es exactamente la realidad virtual?

Según la RAE la realidad virtual es la "Representación de escenas o imágenes de objetos producida por un sistema informático, que da la sensación de su existencia real."[4]. Esta definición es correcta, pero no recoge todos los puntos claves que hacen que la realidad virtual se diferencie de otras tecnologías, que también permiten representar objetos y darles una sensación de existencia real, como pueden ser los televisores 3D. Buscando otras definiciones podemos encontrar otras más completas cómo que "la realidad virtual (VR) consiste en la inmersión sensorial en un nuevo mundo, basado en entornos reales o no, que ha sido generado de forma artificial, y que podemos percibir gracias a unas gafas de realidad virtual y sus accesorios (cascos de audio, guantes, etc...). El objetivo de esta tecnología es crear un mundo ficticio del que puedes formar parte e incluso ser el protagonista: viendo un coche en un concesionario virtual, siendo protagonista de un videojuego o bien practicando como hacer una operación a corazón abierto."[5].

Por lo que la realidad virtual no solo te permite ver un mundo virtual, sino que te permite 'entrar' en él e interactuar con el entorno de la misma manera que lo harías en el mundo real.

1.4.1. Historia breve de la realidad virtual

Si hablamos de realidad virtual, probablemente nos parezca una tecnología relativamente joven, ya que el primer visor de realidad virtual moderno, el kit de desarrollo de las Oculus Rift, apareció en el año 2010. Pero los orígenes de esta tecnología se sitúan en el año 1962, con la aparición del Sensorama.

"El Sensorama, construido en 1962 por el pionero de la tecnología multimedia Morton Heilig puede ser considerada como la primera máquina de inmersión sensorial, o de realidad virtual. Aunque existió en forma de prototipos prometedores, esta no llegó muy lejos. La idea era muy ambiciosa, se trataba de realizar una experiencia de inmersión sensorial total, a través de imágenes tridimensionales, estímulos visuales, vibraciones, sonidos y hasta olores."[6]

Figura 1.4: Fotografía del Sensorama.

Aunque no fue hasta el año 1994 cuando se comercializó la primera consola de realidad virtual, la Virtual Boy. "Desarrollada y producida por Nintendo, es la primera consola que se podría considerar capaz de generar gráficos en 3D de serie, en una especie de realidad virtual. La Virtual Boy crea la ilusión de profundidad con un efecto llamado 'parallax', generado por distintas profundidades en una máquina construida por espejos."[7] Por desgracia la consola supuso un fracaso comercial, por lo que solo fue comercializada en el mercado japonés y estadounidense, quedándose por lo tanto fuera del mercado europeo.

Figura 1.5: Virtual Boy.

Casi 20 años después, en el año 2010, Palmer Luckey decide revivir la realidad virtual desarrollando el primer kit de desarrollo de las Oculus Rift, y para ello realizó una campaña de Kickstarter [8] en la que buscaba una financiación de 250000 dólares. La campaña fue todo un éxito, logrando recaudar 2,5 millones de dólares, permitiéndole crear un segundo prototipo más avanzado, y el primer visor comercial llamado Oculus Rift. Más tarde, el gigante tecnológico Facebook compraría la compañía por 2000 millones de dólares, demostrando el gran potencial que posee la realidad virtual como negocio.

Figura 1.6: Visor de realidad virtual Oculus Rift.

En la actualidad, la realidad virtual avanza de manera exponencial. Podemos encontrar en el mercado visores como las Oculus Quest [9], un visor enfocado a las masas, totalmente autónomo y que cuenta con cuatro cámaras integradas para realizar el seguimiento de los controladores y el propio visor, o las Valve Index [10], que están enfocadas a un mercado más elitista y requieren un ordenador de alta gama para funcionar, además de ofrecer seguimiento de dedos y una mayor resolución y campo de visión.

1.5. ¿Qué es un motor gráfico?

Ya hemos visto que la realidad virtual nos permite explorar mundos virtuales de forma inmersiva. Pero, ¿cómo se crean esos mundos virtuales?

La respuesta son los motores gráficos. Un motor gráfico es un conjunto de herramientas que facilitan la creación de videojuegos. "Su función básica es renderizar los gráficos (mostrarlos en pantalla), aunque también puede encargarse de simular la física, calcular la iluminación de la escena, gestionar la memoria, las animaciones, la inteligencia artificial, la comunicación en red... Además los motores gráficos suelen incluir una serie de herramientas que simplifican el desarrollo y la comunicación con el propio motor como pueden ser editores de niveles, herramientas de importación de modelos y animaciones, lenguajes de scripting para definir comportamientos, etc. Como se puede deducir, es un componente imprescindible para crear un videojuego."[11]

Entre los principales motores gráficos destacan los siguientes:

1.5.1. Unity

Unity es uno de los motores de videojuegos más populares que existen. Desarrollado por la empresa Unity Technologies, Unity es utilizado en sectores como "la arquitectura, la industria automotriz, de la construcción, ingeniería, cinematografía, juegos y muchas otras. Las experiencias creadas con Unity llegan a aproximadamente 3 mil millones de dispositivos en todo el mundo, y se han instalado 24 mil millones de veces en los últimos 12 meses."[12]. Este motor utiliza C# como lenguaje de programación.

Un videojuego muy popular en realidad virtual que utiliza Unity es Beat Saber [13].

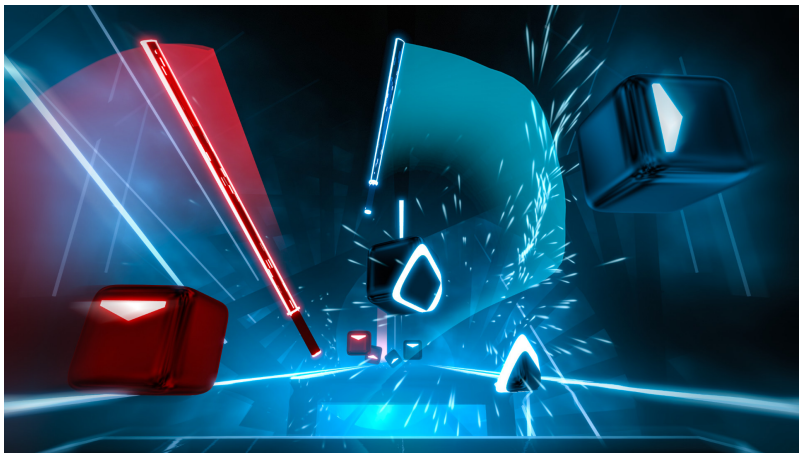


Figura 1.7: Beat Saber.

1.5.2. CryEngine

CryEngine es un motor gráfico creado por la empresa Crytek. Los videojuegos creados con CryEngine destacan por su alta fiabilidad gráfica, aunque este motor no es tan utilizado por los desarrolladores como Unity o Unreal debido a su elevada curva de aprendizaje y a su rendimiento inferior en dispositivos móviles. Este motor utiliza C++ como lenguaje de programación.

Un videojuego popular en realidad virtual que utiliza CryEngine es The Climb [14].

Figura 1.8: The Climb.

1.5.3. Unreal Engine

Unreal Engine es un motor gráfico desarrollado por Epic Games. Se trata de una suite completa de desarrollo de videojuegos escrita en C++ de código abierto. Aunque es más utilizado que CryEngine, no llega a superar a Unity en cuanto a número de desarrolladores, sobre todo en el terreno de videojuegos para móviles. Los lenguajes de programación que utiliza Unreal Engine son C++ y Blueprints.

Algunos videojuegos populares en realidad virtual que utilizan Unreal Engine son Batman: Arkham VR o Robo Recall[15].

(a) Batman: Arkham VR.

(b) Robo Recall.

Figura 1.9: Videojuegos que utilizan Unreal Engine.

Capítulo 2

Justificación de las competencias

A continuación se explicará cómo se han cubierto las competencias específicas relacionadas con este proyecto.

2.1. IS01

"Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software."[16]

Esta competencia ha sido cubierta durante el proceso de análisis previo al desarrollo de la aplicación, realizado para detectar los requisitos que el usuario espera de esta. Por otro lado, se ha tenido en cuenta los estándares de calidad y de rendimiento al utilizar assets con pocos polígonos y al habilitar el Forward Rendering en el motor Unreal Engine, que solo renderiza lo que el usuario está observando en un momento determinado. Además, se ha tratado de realizar el código lo más modular posible, permitiendo de esta manera realizar cambios rápidamente sin afectar al resto de componentes de la aplicación. Por último, se ha utilizado el sistema de log del motor Unreal Engine para depurar el código durante el proceso de desarrollo, para así evitar posibles errores o excepciones que se puedan producir durante la ejecución de la aplicación.

2.2. IS03

"Capacidad de dar solución a problemas de integración en función de las estrategias, estándares y tecnologías disponibles."[16]

Esta competencia ha sido cubierta con la integración del sistema de realidad virtual Oculus Rift en el entorno de desarrollo de Unreal Engine. Utilizando para ello la API de Oculus, y añadiendo los módulos necesarios al archivo de configuración del motor, como el módulo de los controladores de movimiento o del propio dispositivo de realidad virtual. Además, se ha seguido el estándar creado por Google con Daydream [17] para adaptar las interfaces de usuario a un entorno de realidad virtual.

Capítulo 3

Normativa

Para la realización de este proyecto se ha utilizado un motor de videojuegos comercial, el Unreal Engine 4, y varios packs de Assets también comerciales. A continuación se explicará qué tipo de licencia nos ofrece Unreal Engine 4 y cual es la normativa respecto a los Assets e imágenes utilizadas.

3.1. Licencia de Unreal Engine 4

Unreal Engine es gratuito, y no es necesario pagar por publicar un videojuego desarrollado con este motor. Sin embargo, si el videojuego desarrollado genera más de 3000 euros en un trimestre, o supera los 5000000 de euros de beneficio en la tienda de Oculus, se deberá comenzar a pagar un 5 % de los ingresos brutos generados por el videojuego. [18]

Por otro lado, si el motor es utilizado para crear visualizaciones arquitectónicas, animaciones 3D o aplicaciones totalmente gratuitas no es necesario pagar por utilizar el motor. [19]

Debido a que la aplicación desarrollada no va a ser utilizada comercialmente, la licencia de Unreal Engine 4 nos permite utilizar este motor sin que suponga un coste alguno.

3.2. Assets utilizados

Todos los Assets utilizados en la aplicación proceden del Marketplace de Unreal Engine, y se dividen en dos categorías:

3.2.1. Assets gratuitos de Unreal Engine

Estos Assets son ofrecidos por Epic Games de manera gratuita para comenzar a experimentar con el motor. Pueden ser modificados y utilizados en aplicaciones comerciales de forma gratuita. [20]

3.2.2. Packs de Assets de pago del Marketplace

Estos packs de Assets son ofrecidos por particulares en el Marketplace de Unreal Engine, y una vez comprados pueden ser modificados y utilizados en aplicaciones comerciales. [20]

Sin embargo, cada mes Epic Games pone a disposición de los usuarios varios de los packs de pago de forma totalmente gratuita. De manera que si se obtienen a través de estas promociones se pueden utilizar de forma comercial en cualquier aplicación desarrollada sin coste alguno.

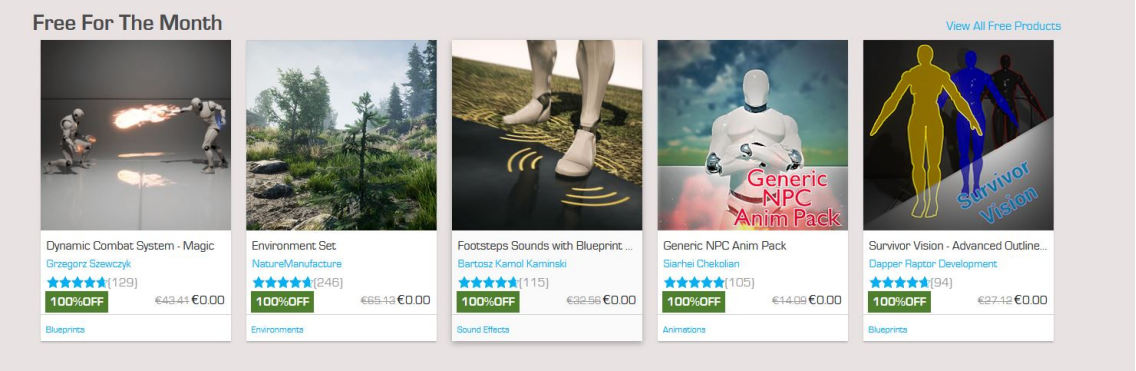


Figura 3.1: Packs de pago que Epic Games ofrece de forma gratuita.

Todos los Assets de pago utilizados para el desarrollo de la aplicación han sido obtenidos a través de estas promociones. Por lo que la licencia de los Assets que se utilizan en la aplicación nos permite utilizarlos sin restricciones.

3.3. Imágenes utilizadas

Para la realización de las interfaces de usuario se han utilizado iconos para representar los botones. Estas imágenes tienen dos tipos de licencia dependiendo de su procedencia:

- Se han utilizado iconos procedentes del pack Material Design, que es Open Source a través de la Apache license version 2.0 [21]. Por lo que la licencia nos permite utilizar estos iconos en todo tipo de aplicaciones, independientemente de si es comercial o no [22].
- También han utilizado iconos procedentes de la página iconfinder.com, que ofrece iconos bajo licencia Creative Commons [23]. Por lo que la licencia nos permite utilizar estos iconos en todo tipo de aplicaciones, independientemente de si es comercial o no, siempre que le demos crédito al autor de los iconos [24].

La lista de imágenes utilizadas en el desarrollo de esta aplicación y sus respectivos autores se encuentra incluida en el código fuente de la aplicación, como indica la normativa Creative Commons.

Capítulo 4

Aportaciones al entorno socio-económico y técnico

Uno de los principales objetivos que se busca al realizar este proyecto es simplificar el proceso de diseños de edificios, a la vez que se exploran nuevos usos de una tecnología emergente como lo es la realidad virtual.

Respecto al entorno técnico, esta aplicación se podría considerar una demostración técnica de la aplicación de la realidad virtual al proceso de diseño de edificios por las siguientes razones:

- Se hace uso de los dos tipos principales de locomoción en realidad virtual, el movimiento libre y el movimiento por teletransporte, y se combinan para ofrecer al usuario varias posibilidades de locomoción en base a sus necesidades.
- Se aplican dos tipos de interacción con interfaces de usuario especialmente diseñadas para la realidad virtual, la selección con puntero y la selección con botones virtuales, y se utiliza una u otra en base a cuál es más cómoda para el usuario en cada situación.
- Se aprovechan los controladores de movimiento que el usuario lleva en cada mano para que pueda interactuar de una forma más natural con el entorno y las interfaces. Como por ejemplo, que a la hora de seleccionar un objeto solo deba apuntar con el controlador hacia él, o que para modificar la posición de un objeto solo deba mover la mano hacia la posición donde lo desea colocar.

Respecto al entorno socio-económico, esta aplicación permite a los arquitectos o inmobiliarias mostrar tanto edificios finalizados como conceptos de edificios a sus potenciales clientes. Aprovechando la inmersión y la escala real del edificio que ofrece la realidad virtual, podrán darle una idea más aproximada al edificio real que la que se consigue con una fotografía o vídeo.

Por otro lado, con esta aplicación los diseñadores podrán acortar los tiempos de desarrollo y reducir costes de producción, ya que al tratarse de un entorno generado por ordenador, se pueden realizar todos los cambios necesarios al edificio si que suponga un coste alguno. Además, los diseñadores podrán dar rienda suelta a su imaginación, ya que al tratarse de un entorno controlado en realidad virtual podrán probar un mayor número de diseños, y experimentar con diseños más arriesgados que en otro caso habrían sido descartados por su complejidad o coste.

Capítulo 5

Tecnologías utilizadas

A continuación se describirán los recursos y las tecnologías que han sido utilizadas para llevar a cabo la realización de la aplicación.

5.1. Hardware

Se ha utilizado el siguiente hardware para el desarrollo y ejecución de la aplicación:

5.1.1. Ordenador de sobremesa

Se ha utilizado un ordenador de sobremesa como herramienta para desarrollar y probar la aplicación.

El ordenador de sobremesa utilizado cumple los requisitos para poder ejecutar aplicaciones en realidad virtual. Además, el motor Unreal Engine 4 se encuentra instalado en una unidad SSD para reducir los tiempos de carga.

Sus especificaciones son las siguientes:

- Procesador: AMD FX-8350 4.0Ghz.
- Tarjeta gráfica: MSI GeForce GTX 1060 6GB.
- Memoria RAM: 16GB DDR4.
- Disco duro SSD: 250 GB.
- Disco duro HDD: 1 TB
- Sistema operativo: Windows 10.

5.1.2. Oculus Rift + Oculus Touch

Se ha utilizado el sistema de realidad virtual Oculus Rift y los controladores Oculus Touch como interfaz entre el usuario y la aplicación. Sus características son las siguientes:

Oculus Rift

El visor de realidad virtual Oculus Rift fue lanzado en 2016, convirtiéndose en uno de los primeros visores de realidad virtual modernos en estar a la venta.

Sus especificaciones técnicas son [25]:

- Cuenta con 2 pantallas OLED con resolución 2160x1200.
- Tiene un ángulo de visión de 110º.
- Presenta una frecuencia de refresco de 90Hz.
- Ofrece un micrófono y auriculares integrados.

Para el desarrollo de la aplicación se ha utilizado el visor para realizar el seguimiento de la cabeza del jugador y para mostrarle el entorno virtual de la aplicación.

Oculus Touch

Los controladores Oculus Touch fueron lanzados en el año 2017, y surgieron como una respuesta por parte de Oculus a los controladores que ofrecía la competencia, como los controladores del HTC Vive.

Cada controlador cuenta con 2 botones de acción, un joystick, un trigger y un botón 'grip', que permite al usuario realizar la acción de coger un objeto.

Para el desarrollo de la aplicación se han utilizado los controladores para realizar el seguimiento de las manos del jugador, y permitirle interactuar con el entorno virtual y las interfaces de usuario.

Figura 5.1: Controladores Oculus Touch.

5.2. Software

El software utilizado para el desarrollo y mantenimiento de la aplicación ha sido el siguiente:

5.2.1. Unreal Engine 4

Se ha usado Unreal Engine 4 como motor gráfico de la aplicación. Como se ha comentado anteriormente, Unreal Engine 4 es una suite completa de desarrollo de videojuegos. Además, tiene soporte para la API de Oculus, por lo que es posible crear aplicaciones de realidad virtual con este motor.

5.2.2. Visual Studio

Como entorno de desarrollo se ha utilizado Visual Studio 2017. Visual Studio es el entorno de desarrollo que utiliza Unreal Engine 4, ya que soporta el lenguaje C++.

"Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno compatible con la plataforma .NET (a partir de la versión .NET 2002). Así, se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y videoconsolas, entre otros."[26]

5.2.3. Github

Para el control de versiones se ha utilizado Git, y como repositorio Github.

"Git es un sistema de control de revisión distribuido, rápido y escalable con un conjunto de comandos inusualmente rico que proporciona operaciones de alto nivel y acceso completo a componentes internos."[27]

Por su parte, "GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de computadora."[28]

5.2.4. Sourcetree

Para simplicar el control de versiones, se ha utilizado el programa Sourcetree. Sourcetree es una aplicación de escritorio gratuita que nos ofrece una representación visual de los repositorios con los que estamos trabajando. Además, elimina la necesidad de utilizar comandos git, ya que todo lo necesario para gestionar los repositorios se puede realizar desde su interfaz.

5.2.5. Trello

Se ha utilizado Trello para organizar y controlar las distintas tareas que se han tenido que realizar durante el proceso de desarrollo de la aplicación.

"Trello es un tablón virtual en el que se pueden colgar ideas, tareas, imágenes o enlaces. Es versátil y fácil de usar pudiendo usarse para cualquier tipo de tarea que requiera organizar información."[29]

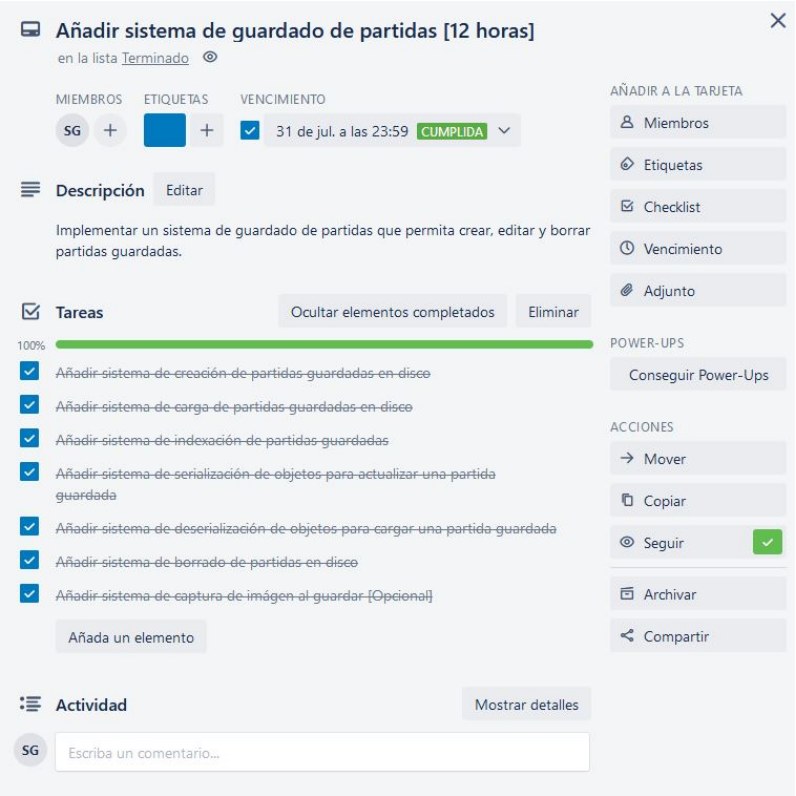


Figura 5.2: Información de una tarea a realizar en Trello.

5.2.6. Figma

Para el diseño de las interfaces de usuario se ha utilizado la aplicación Figma. Figma es un editor online que permite crear diseños de interfaces de usuario, sin tener que preocuparnos por problemas de alineación y ajuste de imágenes. Ya que permite colocar y editar el tamaño los componentes de la interfaz de manera intuitiva, simplemente arrastrando el ratón por el lienzo. Además, permite exportar cada uno de los elementos de la interfaz cómo imágenes, para luego ser importados en los programas que implementen dichas interfaces.

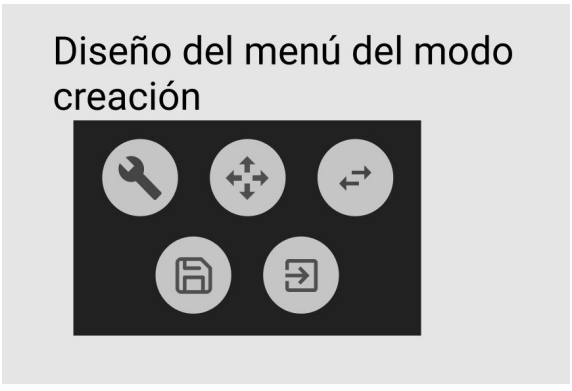


Figura 5.3: Diseño de interfaz de usuario creado en Figma.

5.2.7. Overleaf

Para la realización de la memoria del proyecto se ha utilizado la aplicación Overleaf. Overleaf es un editor de LaTeX online que facilita la creación de documentos de todo tipo, permitiendo organizar los distintos ficheros LaTeX utilizados, y poniendo a disposición de los usuarios una gran cantidad de plantillas y tutoriales sobre como redactar documentos con LaTeX. Además, permite previsualizar y exportar a pdf el documento que se está redactando con solo pulsar un botón.

5.3. Lenguajes

Los lenguajes de programación con los que se ha desarrollado la aplicación han sido los siguientes:

5.3.1. C++

Se ha utilizado el lenguaje C++ como lenguaje principal a la hora de realizar el proyecto, la gran mayoría de funciones de la aplicación están escritas en este lenguaje.

"C++ es un lenguaje de programación diseñado a mediados de los años 80 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitieran la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. En la actualidad, el C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones."[30]

5.3.2. Blueprints

Se ha utilizado el lenguaje de scripting Blueprint para implementar pequeñas características de la aplicación, como la comunicación entre los widgets y sus respectivas clases al producirse un evento, o la modificación de variables en blueprints que heredan de clases en C++.

"Los Blueprints son assests que están en el editor de Unreal Engine y se organizan en nodos conectados entre sí. Tienen multitud de opciones y posibilidades, como por ejemplo:

- Crear, implementar o modificar cualquier elemento
- Creación de mallas, materiales y personajes
- Interacción con menús
- Creación de cámaras y perspectivas, ítems (armas, hechizos, etc.), partículas como lluvia o nieve
- Interacción con el entorno"[31]

Capítulo 6

Planificación inicial

Durante la planificación inicial se ha llevado a cabo la fase de análisis y se han creado varios esquemas de las acciones que realizarán los botones de los controladores en los distintos modos de la aplicación.

6.1. Análisis

Durante la fase de análisis se ha elegido la metodología de desarrollo a seguir, y se ha organizado el desarrollo de la aplicación en base a ella.

Además, se han definido las clases que utilizará la aplicación y las relaciones entre esas clases, creando para ello un diagrama de clases. Por otro lado, se han especificado las historias de usuario, y se ha creado un diagrama de casos de uso a partir de ellas.

Por último, se ha creado una pila de producto con todas las tareas que se deben realizar para completar el desarrollo de la aplicación, y se han ido asignando tareas a cada sprint realizado.

6.1.1. Metodología utilizada

Para el desarrollo del proyecto se ha decidido seguir la metodología Scrum. Pero al tratarse de un desarrollo realizado por una sola persona se han omitido los daily meetings y se han juntado todos los roles de Scrum en la misma persona. Además, debido a que los Sprints no tenían siempre la misma duración, se ha realizado un ajuste del factor de foco para cada Sprint más conservador que el que establece Scrum.

¿Qué es Scrum?

A continuación, se explicará brevemente qué es Scrum, y los distintos actores, eventos y artefactos que forman parte de él.

Según la guía oficial, "Scrum es un marco de trabajo de procesos que ha sido usado para gestionar el desarrollo de productos complejos desde principios de los años 90. Scrum no es un proceso o una técnica para construir productos; en lugar de eso, es un marco de trabajo dentro del cual se pueden emplear varios procesos y técnicas."[32]

Actores en Scrum

En la metodología Scrum intervienen el Dueño del Producto, el Scrum Master y el Equipo de Desarrollo.

- "El Dueño de Producto es el responsable de maximizar el valor del producto y el trabajo del Equipo de Desarrollo, es la única persona responsable de gestionar la Lista del Producto (Product Backlog).
- El Scrum Master es el responsable de asegurar que Scrum se entienda y se adopte. Los Scrum Masters hacen esto asegurándose de que el Equipo Scrum trabaja ajustándose a la teoría, prácticas y reglas de Scrum.
- El Equipo de Desarrollo consiste en los profesionales que realizan el trabajo de entregar un Incremento de producto 'Terminado' que potencialmente se pueda poner en producción al final de cada Sprint. Solo los miembros del Equipo de Desarrollo participan en la creación del Incremento."[32]

Eventos de Scrum

Los eventos que organizan y controlan el desarrollo de una aplicación con Scrum son el Sprint, la Planificación de Sprint, el Scrum Diario, la Revisión del Sprint y la Retrospectiva del Sprint.

- "El corazón de Scrum es el Sprint, es un bloque de tiempo (time-box) de un mes o menos durante el cual se crea un incremento de producto 'Terminado' utilizable y potencialmente desplegable.
- El trabajo a realizar durante el Sprint se planifica en la Planificación de Sprint. Este plan se crea mediante el trabajo colaborativo del Equipo Scrum completo.
- El Scrum Diario es una reunión con un bloque de tiempo de 15 minutos para que el Equipo de Desarrollo sincronice sus actividades y cree un plan para las siguientes 24 horas.
- Al final del Sprint se lleva a cabo una Revisión de Sprint para inspeccionar el Incremento y adaptar la Lista de Producto si fuese necesario.
- La Retrospectiva de Sprint es una oportunidad para el Equipo Scrum de inspeccionarse a sí mismo y de crear un plan de mejoras que sean abordadas durante el siguiente Sprint. La Retrospectiva de Sprint tiene lugar después de la Revisión de Sprint y antes de la siguiente Planificación de Sprint."[32]

Artefactos en Scrum

Finalmente, los artefactos que intervienen en Scrum son la Pila de Producto, la Pila de Sprint y el Incremento.

- "La Pila de Producto es una lista ordenada de todo lo que podría ser necesario en el producto y es la única fuente de requisitos para cualquier cambio a realizarse en el producto.

- La Pila del Sprint es el conjunto de elementos de la Pila de Producto seleccionados para el Sprint, más un plan para entregar el Incremento de producto y conseguir el Objetivo del Sprint.
- El Incremento es la suma de todos los elementos de la Lista de Producto completados durante un Sprint y el valor de los incrementos de todos los Sprints anteriores."[32]

6.1.2. Pila de producto y Sprints realizados

Siguiendo el marco de trabajo de Scrum, se creó una pila de producto para organizar las tareas a realizar durante el desarrollo del proyecto, asignando 1 punto de historia de usuario a cada hora de trabajo. La pila de producto es la siguiente:

id	Nombre	Descripción	Prioridad	Validación	Estimación
H0	Investigar el estado del arte.	Investigar el estado del arte tanto de la industria de realidad virtual en general, como de aplicaciones dedicadas al diseño de edificios en realidad virtual y dibujo 3D.	100	-	7
H1	Análisis de la aplicación	Realizar el análisis de la aplicación para definir la estructura del software y las necesidades del usuario que deberá cubrir la aplicación.	100	-	10
H2	Definir las clases y actores que serán utilizados en el proyecto.	Realizar la especificación de las clases y actores que se utilizarán dentro de Unreal Engine para el correcto funcionamiento de la aplicación.	100	-	15
H3	Crear entorno del menú principal.	Realizar el diseño y creación del entorno al que accederá el usuario al iniciar la aplicación, y que le permitirá crear un nuevo diseño o cargar uno previo.	95	El usuario puede ver el menú principal.	10

Sigue en la página siguiente.

id	Nombre	Descripción	Prioridad	Validación	Estimación
H4	Añadir Pawn.	Añadir el controlador de usuario que 'poseerá' al jugador dentro del juego, y sus variaciones en base a la situación y mapa en el que se encuentre el jugador.	95	El usuario puede 'poseer' un Pawn.	10
H5	Crear el escenario del mapa de creación de edificios.	Crear el mapa en Unreal Engine del entorno de trabajo que utilizará el jugador.	95	El usuario puede ver el entorno de creación de edificios.	15
H6	Añadir sistema de menú principal.	Convertir los diseños creados previamente en Widgets dentro de Unreal Engine y añadir funcionalidad al menú principal.	95	El usuario puede navegar por el menú principal.	10
H7	Añadir sistema de movimiento.	Crear el sistema de movimiento tanto libre como de teletransporte.	70	El usuario puede moverse por el mapa.	10
H8	Añadir sistema de bocetos.	Implementar el sistema de bocetos que permitirá crear y borrar bocetos 3D en el entorno.	80	El usuario puede crear y borrar bocetos.	10
H9	Añadir sistema de spawn de estructuras del edificio.	Crear un sistema de colocación y edición de estructuras.	80	El usuario puede crear y editar estructuras.	20
H10	Añadir sistema de spawn de objetos comunes.	Crear un sistema de colocación y edición de objetos comunes.	80	El usuario puede crear y editar objetos comunes.	20
H11	Añadir sistema de guardado de diseños.	Implementar un sistema de guardado de diseños que permita crear, editar y borrar diseños guardados.	70	El usuario puede guardar, cargar y borrar diseños.	12

Sigue en la página siguiente.

id	Nombre	Descripción	Prioridad	Validación	Estimación
H12	Control de errores y refactorización.	Realizar el control de errores de toda la aplicación y refactorizar las clases.	50	-	30
H13	Crear memoria final.	Crear la memoria final del proyecto.	50	-	20
H14	Crear presentación de la defensa.	Crear la presentación que será utilizada en la defensa del proyecto.	50	-	10
					Total: 209

Cuadro 6.1: Pila de producto.

Planificación del primer Sprint

Tras realizar la pila de producto se calculó la velocidad del Sprint inicial y se realizó la pila del primer Sprint. El primer Sprint abarca desde el día 1 de abril al 15 de abril de 2019.

A la hora de calcular la velocidad del sprint, se puso un límite de 15 días para realizar el sprint, una media de 6 horas de trabajo al día y se asignó un factor de foco de 0,8.

Miembro	Días	Horas/Día	Total Horas (estimación esfuerzo)
Sergio	15	6	90
		Total disponibilidad	90
		Factor de foco sin pasado	0.8
		Velocidad estimada	72

Cuadro 6.2: Velocidad del primer Sprint.

Tras obtener la velocidad estimada se realizó la pila del Sprint, que recogía las historias H0 (Investigar el estado del arte), H1 (Análisis de la aplicación), H2 (Definir las clases y actores que serán utilizados en el proyecto), H3 (Crear entorno del menú principal), H4 (Añadir Pawn), H5 (Crear escenario del mapa de creación de edificios) y H6 (Añadir sistema de menú principal).

Retrospectiva del primer Sprint y planificación del segundo Sprint

La estimación del primer Sprint se quedó un poco corta, ya que no se consiguió terminar la historia H6 (Añadir sistema de menú principal) a tiempo, como consecuencia, se reduce el factor de foco a 0.7 para tratar de ajustar mejor la velocidad estimada con la velocidad real. Además, se amplía el tiempo en el que se desarrolla el Sprint a 30 días y se reduce el número de horas dedicadas al día a 3, para ajustar más a la realidad el desarrollo del Sprint.

El segundo Sprint fue desarrollado entre los días 16 de abril y 15 de mayo. El cálculo de la velocidad del segundo Sprint es el siguiente:

Miembro	**Días**	**Horas/Día**	**Total Horas (estimación esfuerzo)**
Sergio	30	3	90
		Total disponibilidad	90
		Factor de foco	0.7
		Velocidad estimada	63

Cuadro 6.3: Velocidad del segundo Sprint.

Tras obtener la velocidad estimada se realizó la pila del Sprint, que recogía las historias H6 (Añadir sistema de menú principal), H7 (Añadir sistema de movimiento), H8 (Añadir sistema de bocetos), H9 (Añadir sistema de spawn de estructuras del edificio) y H10 (Añadir sistema de spawn de objetos comunes).

Retrospectiva del segundo Sprint y planificación del tercer Sprint

En el segundo Sprint no se lograron los objetivos establecidos, ya que solo se consiguió realizar las historias H6 (Añadir sistema de menú principal) y H7 (Añadir sistema de movimiento). El motivo de que no se completaran todas las historias fue porque no se pudo dedicar el tiempo necesario, y la complejidad de las historias de usuario era mayor a lo calculado. En el tercer Sprint se mantiene el límite de tiempo y las horas dedicadas al día del Sprint anterior. Este Sprint fue desarrollado entre los días 16 de mayo y 15 de junio de 2019.

Como hemos comentado anteriormente, a la hora de calcular la velocidad del Sprint, se puso un límite de 30 días para realizar el Sprint, una media de 3 horas de trabajo al día y se asignó un factor de foco de 0,8.

Miembro	**Días**	**Horas/Día**	**Total Horas (estimación esfuerzo)**
Sergio	30	3	90

Sigue en la página siguiente.

Miembro	Días	Horas/Día	Total Horas (estimación esfuerzo)
		Total disponibilidad	90
		Factor de foco	0.8
		Velocidad estimada	63

Cuadro 6.4: Velocidad del tercer Sprint.

Tras obtener la velocidad estimada se realizó la pila del Sprint, que recogía las historias H8 (Añadir sistema de bocetos), H9 (Añadir sistema de spawn de estructuras del edificio), H10 (Añadir sistema de spawn de objetos comunes) y H11 (Añadir sistema de guardado de diseños).

Retrospectiva del tercer Sprint y planificación del cuarto Sprint

En el tercer Sprint tampoco se consiguió alcanzar el objetivo propuesto, ya que no se pudieron realizar las historias H9 (Añadir sistema de spawn de estructuras del edificio) y H11 (Añadir sistema de guardado de diseños). La razón fue que las historias H8 (Añadir sistema de bocetos) y H10 (Añadir sistema de spawn de objetos comunes) resultaron ser bastante más extensas de lo calculado. Debido a esto, se reduce el factor de foco a 0,7 para este Sprint, y mantenemos el límite de días y las horas dedicadas al día. El cuarto Sprint fue desarrollado entre los días 16 de junio y 15 de julio de 2019.

Como hemos comentado anteriormente, a la hora de calcular la velocidad del Sprint, se puso un límite de 30 días para realizar el Sprint, una media de 3 horas de trabajo al día y se asignó un factor de foco de 0,7.

Miembro	Días	Horas/Día	Total Horas (estimación esfuerzo)
Sergio	30	3	90
		Total disponibilidad	90
		Factor de foco	0.7
		Velocidad estimada	63

Cuadro 6.5: Velocidad del cuarto Sprint.

Tras obtener la velocidad estimada se realizó la pila del Sprint, que recogía las historias H9 (Añadir sistema de spawn de estructuras del edificio), H11 (Añadir sistema de guardado de diseños) y H12 (Control de errores y refactorización).

Retrospectiva del cuarto Sprint y planificación del quinto Sprint

En el cuarto Sprint solo se realizó la historia H9 (Añadir sistema de spawn de estructuras del edificio), debido a que era más compleja de lo calculado, y fue necesario realizar cambios en el código realizado anteriormente para que funcionara correctamente. Debido a esto, y teniendo en cuenta que el proyecto debe estar terminado en agosto, se mantiene el factor de foco, se reduce el límite de días a 15, y se amplía el número de horas dedicadas al día a 4. El quinto Sprint fue desarrollado entre los días 16 de julio y 1 de agosto de 2019.

Como hemos comentado anteriormente, a la hora de calcular la velocidad del Sprint, se puso un límite de 15 días para realizar el Sprint, una media de 4 horas de trabajo al día y se asignó un factor de foco de 0,7.

Miembro	**Días**	**Horas/Día**	**Total Horas (estimación esfuerzo)**
Sergio	15	4	60
		Total disponibilidad	60
		Factor de foco	0.7
		Velocidad estimada	42

Cuadro 6.6: Velocidad del quinto Sprint.

Tras obtener la velocidad estimada se realizó la pila del Sprint, que recogía las historias H11 (Añadir sistema de guardado de diseños) y H12 (Control de errores y refactorización).

Retrospectiva del quinto Sprint y planificación del sexto Sprint

En el quinto Sprint se consiguió lograr los objetivos establecidos, por lo que la estimación de la complejidad de las historias de usuario y el cálculo de la velocidad del Sprint fueron correctos. El sexto y último Sprint está dedicado a la realización de la memoria del proyecto y la presentación de la defensa. Por lo se encuentra actualmente en proceso, y abarca desde el día 2 de agosto al 30 de agosto de 2019.

A la hora de calcular la velocidad del Sprint, se puso un límite de 30 días para realizar el Sprint, una media de 3 horas de trabajo al día y se asignó un factor de foco de 0,7.

Miembro	**Días**	**Horas/Día**	**Total Horas (estimación esfuerzo)**
Sergio	30	3	90

Sigue en la página siguiente.

Miembro	Días	Horas/Día	Total Horas (estimación esfuerzo)
		Total disponibilidad	90
		Factor de foco	0.7
		Velocidad estimada	63

Cuadro 6.7: Velocidad del sexto Sprint.

Tras obtener la velocidad estimada se realizó la pila del Sprint, que recogía las historias H13 (Crear memoria final) y H14 (Crear presentación de la defensa).

6.1.3. Definición de clases

Durante la fase de análisis se definieron las clases iniciales que iba a tener la aplicación, y se creó el diagrama de clases.

Clases a utilizar

Durante la fase de análisis se definieron a alto nivel las clases que serían utilizadas inicialmente para implementar cada funcionalidad de la aplicación. La razón de definir las clases a alto nivel es porque muchas de estas clases heredan de objetos del motor Unreal, y durante la fase de análisis aún no se tenía una idea clara sobre cómo se iba a implementar cada funcionalidad, por lo que el diseño podría variar significativamente durante la fase de desarrollo. Las clases a utilizar son las siguientes:

Para recuperar el diseño guardado al iniciar un nivel se utilizará la clase **GameMode**.

Para el sistema de guardado de diseños se utilizará la clase **LevelSaveGameIndex** para almacenar los diseños guardados, y la clase **LevelSaveGame** para almacenar y recuperar los datos de cada diseño guardado.

Para el jugador serán necesarias dos clases principales. La clase **VRPawn** será utilizada en el menú principal y en el modo creación, ya que en estos casos no es necesario que el usuario se desplace por el mapa. Y la clase **VRCharacter** será utilizada en el modo diseño, ya que en este caso si es necesario que el usuario se pueda desplazar por el mapa. Además, se utilizaran las clases **DesignModeHandController** y **CreationModeHandController** para cada uno de los controladores del jugador dependiendo del modo en el que se encuentre.

Para las interfaces de usuario se utilizaran las clases **MainMenuWidget** , **DesignModeSelectionMenu** y **CreationModeSelectionMenu**, que nos permitirán controlar los eventos producidos al interactuar con las interfaces.

Para la colocación de objetos se utilizarán las clases **PlacedObject** y **PlacedStructure** para representar los objetos y estructuras colocadas.

Para la creación de bocetos se utilizará la clase **Brush**, que representará cada uno de los bocetos realizados y guardará el historial de trazos de cada boceto.

Diagrama de clases

El diagrama de clases es el siguiente:

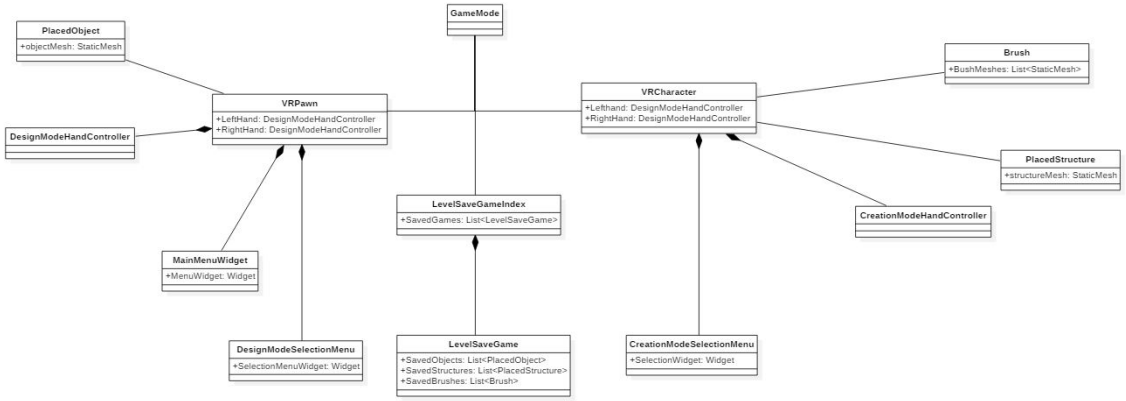


Figura 6.1: Diagrama de clases.

6.1.4. Casos de uso

Tras realizar el diagrama de clases, se especificaron los casos de uso de la aplicación y se realizó el diagrama de casos de uso.

Especificaciones de caso de uso

Las especificaciones de caso de uso realizadas son las siguientes:

Caso de uso 1	Create new design.
Descripción	El usuario crea un nuevo diseño.
Actores	Main menu user.
Precondiciones	El usuario debe encontrarse en el menú principal.
Flujo normal	**Paso y Acción**
	1 El usuario selecciona la opción de crear un nuevo diseño.
Postcondiciones	El sistema crea el nuevo diseño y lo añade al índice de diseños guardados, posteriormente, actualiza la interfaz para mostrar el nuevo diseño guardado.

Cuadro 6.8: Caso de uso 1.

Caso de uso 2	Load design.
Descripción	El usuario carga un diseño guardado.
Actores	Main menu user.
Precondiciones	El usuario debe encontrarse en el menú principal.
Flujo normal	**Paso y Acción**
	1 El usuario selecciona la opción de cargar un diseño.
	2 El usuario selecciona de la lista de diseños el diseño que desea cargar.
Postcondiciones	El sistema recupera el diseño del índice de diseños guardados, carga el diseño y coloca al usuario en el.

Cuadro 6.9: Caso de uso 2.

Caso de uso 3	Delete design.
Descripción	El usuario elimina un diseño guardado.
Actores	Main menu user.
Precondiciones	El usuario debe encontrarse en el menú principal.
Flujo normal	**Paso y Acción**
	1 El usuario selecciona la opción de eliminar un diseño.
	2 El usuario selecciona de la lista de diseños el diseño que desea eliminar.
	3 El sistema pregunta al usuario si realmente desea eliminar el diseño guardado.
Postcondiciones	El sistema elimina el diseño guardado y lo borra del índice de diseños guardados.
Variaciones	**Paso y Acción**
	3-A El usuario confirma que desea eliminar el diseño.
	3-B El usuario cancela la eliminación del diseño.

Cuadro 6.10: Caso de uso 3.

Caso de uso 4	Move by teleportation.
Descripción	El usuario se mueve por el escenario utilizando un sistema de movimiento por teletransporte.
Actores	Design mode user.
Precondiciones	El usuario debe encontrarse en una posición válida para realizar el teletransporte.

Sigue en la página siguiente.

Flujo normal	**Paso y Acción**
	1 El usuario activa la función de teletransporte.
	2 El sistema muestra al usuario un avance de la posición en el espacio a la que se va a teletransportar.
	3 El usuario ajusta la posición a la que se va a teletransportar moviendo el controlador de su mano derecha.
	4 El usuario confirma la posición a la que se desea teletransportar.
Postcondiciones	El sistema cambia la posición del usuario a la seleccionada al teletransportarse.

Cuadro 6.11: Caso de uso 4.

Caso de uso 5	Move by free locomotion.
Descripción	El usuario se mueve por el escenario utilizando un sistema de movimiento de locomoción libre.
Actores	Creation mode user, design mode user.
Precondiciones	El usuario debe encontrarse en una posición válida para realizar el movimiento.
Flujo normal	**Paso y Acción**
	1 El usuario activa la función de movimiento.
	2 El usuario mantiene el joystick de movimiento en la dirección que desea moverse.
Postcondiciones	El sistema cambia la posición del usuario en base a la posición del joystick.

Cuadro 6.12: Caso de uso 5.

Caso de uso 6	Exit aplication.
Descripción	El usuario sale de la aplicación.
Actores	Main menu user.
Precondiciones	El usuario debe encontrarse en el menú principal.
Flujo normal	**Paso y Acción**
	1 El usuario selecciona en el menú principal la opción de salir de la aplicación.
	2 El sistema pregunta al usuario si desea realmente salir de la aplicación.
Postcondiciones	El sistema cierra la aplicación.
Variaciones	**Paso y Acción**

Sigue en la página siguiente.

	2-A El usuario confirma que desea cerrar la aplicación.
	2-B El usuario cancela el cierre de la aplicación.

Cuadro 6.13: Caso de uso 6.

Caso de uso 7	Create Structure.
Descripción	El usuario crea una estructura.
Actores	Creation mode user.
Precondiciones	El usuario debe encontrarse en el modo creación.
Flujo normal	**Paso y Acción**
	1 El usuario selecciona la opción de crear una nueva estructura.
	2 El sistema le muestra al usuario una previsualización de la estructura en el espacio 3D.
	3 El usuario posiciona la estructura en el espacio 3D.
Postcondiciones	El sistema coloca la estructura en el lugar seleccionado por el usuario.

Cuadro 6.14: Caso de uso 7.

Caso de uso 8	UD Structure.
Descripción	El usuario edita o borra una estructura.
Actores	Creation mode user.
Precondiciones	El usuario debe encontrarse en el modo creación.
Flujo normal	**Paso y Acción**
	1 El usuario selecciona una estructura del entorno 3D.
	2 El usuario mueve el controlador para ajustar la posición y rotación de la estructura.
	3 El usuario confirma la nueva posición y rotación de la estructura.
Postcondiciones	El sistema actualiza la posición de la estructura en base al lugar seleccionado por el usuario.
Variaciones	**Paso y Acción**
	2-A-1 El usuario elimina la estructura.
	2-A-2 El sistema borra la estructura del espacio 3D.

Cuadro 6.15: Caso de uso 8.

Caso de uso 9	Change creation mode menu state.
Descripción	El usuario activa o desactiva el menú del modo creación.
Actores	Creation mode user.
Precondiciones	El usuario debe encontrarse en el modo creación.
Flujo normal	**Paso y Acción**
	1 El usuario pulsa el botón de mostrar/ocultar el menú del modo creación.
Postcondiciones	El sistema muestra/oculta el menú del modo creación.

Cuadro 6.16: Caso de uso 9.

Caso de uso 10	Select item from selection menu.
Descripción	El usuario selecciona un elemento del menú de selección del modo creación o diseño.
Actores	Creation mode user, design mode user.
Precondiciones	El usuario debe encontrarse en el modo creación o diseño y el menú debe encontrase visible.
Flujo normal	**Paso y Acción**
	1 El usuario apunta con el controlador de su mano derecha hacia el elemento del menú que desea seleccionar.
	2 El usuario selecciona el elemento deseado.
Postcondiciones	El sistema genera un evento de selección del elemento seleccionado.

Cuadro 6.17: Caso de uso 10.

Caso de uso 11	Update design.
Descripción	El usuario actualiza el diseño guardado en el que se encuentra.
Actores	Creation mode user, design mode user.
Precondiciones	El usuario debe encontrarse en el modo creación o diseño y debe haber un diseño cargado.
Flujo normal	**Paso y Acción**
	1 El usuario selecciona la opción de guardar diseño.
Postcondiciones	El sistema actualiza el diseño guardado.

Cuadro 6.18: Caso de uso 11.

Caso de uso 12	Change mode.
Descripción	El usuario cambia entre el modo creación y el modo diseño.
Actores	Creation mode user, design mode user.
Precondiciones	El usuario debe encontrarse en el modo creación o diseño.
Flujo normal	**Paso y Acción**
	1 En el modo creación, el usuario selecciona en el menú del controlador la opción de cambiar de modo.
	2 El usuario selecciona la posición en la que desea aparecer en el espacio 3D.
Postcondiciones	El sistema cambia el modo y posiciona al jugador en la posición seleccionada.
Variaciones	**Paso y Acción**
	1-A-1 En el modo diseño, el usuario selecciona en el menú del controlador la opción de cambiar de modo.
	1-A-2 El sistema cambia de modo.

Cuadro 6.19: Caso de uso 12.

Caso de uso 13	Exit to main menu.
Descripción	El usuario sale al menú principal.
Actores	Creation mode user, design mode user.
Precondiciones	El usuario debe encontrarse en el modo creación o diseño.
Flujo normal	**Paso y Acción**
	1 El usuario selecciona la opción de salir al menú principal.
	2 El sistema pregunta al usuario si realmente desea salir al menú principal.
Postcondiciones	El sistema carga el menú principal y coloca al usuario en él.
Variaciones	**Paso y Acción**
	2-A El usuario confirma que desea salir al menú principal.
	2-B El usuario cancela la operación.

Cuadro 6.20: Caso de uso 13.

Caso de uso 14	Create object.
Descripción	El usuario crea un objeto.
Actores	Design mode user.

Sigue en la página siguiente.

Precondiciones	El usuario debe encontrarse en el modo diseño.
Flujo normal	**Paso y Acción**
	1 El usuario selecciona la opción de crear un nuevo objeto.
	2 El sistema le muestra al usuario una previsualización del objeto en el espacio 3D.
	3 El usuario posiciona el objeto en el espacio 3D.
Postcondiciones	El sistema coloca el objeto en el lugar seleccionado por el usuario.

Cuadro 6.21: Caso de uso 14.

Caso de uso 15	UD object.
Descripción	El usuario edita o borra un objeto.
Actores	Design mode user.
Precondiciones	El usuario debe encontrarse en el modo diseño.
Flujo normal	**Paso y Acción**
	1 El usuario selecciona un objeto del entorno 3D.
	2 El usuario mueve el controlador para ajustar la posición y rotación del objeto.
	3 El usuario confirma la nueva posición y rotación del objeto.
Postcondiciones	El sistema actualiza la posición del objeto en base al lugar seleccionado por el usuario.
Variaciones	**Paso y Acción**
	2-A-1 El usuario elimina el objeto.
	2-A-2 El sistema borra el objeto del espacio 3D.

Cuadro 6.22: Caso de uso 15.

Caso de uso 16	Create brush.
Descripción	El usuario crea un trazado de pincel.
Actores	Design mode user.
Precondiciones	El usuario debe encontrarse en el modo diseño.
Flujo normal	**Paso y Acción**
	1 El usuario pulsa el botón de crear una nueva traza.
	2 El sistema muestra al usuario la traza a medida que la va creando.
	3 El usuario suelta el botón de crear una nueva traza.

Sigue en la página siguiente.

Postcondiciones	El sistema coloca la traza en las posiciones marcadas por el jugador.

Cuadro 6.23: Caso de uso 16.

Caso de uso 17	Delete brush.
Descripción	El usuario borra un trazado de pincel.
Actores	Design mode user.
Precondiciones	El usuario debe encontrarse en el modo diseño.
Flujo normal	**Paso y Acción**
	1 El usuario pulsa el botón de eliminar la última traza.
Postcondiciones	El sistema elimina la última traza creada por el usuario.

Cuadro 6.24: Caso de uso 17.

Caso de uso 18	Change design mode menu state.
Descripción	El usuario activa o desactiva el menú del modo diseño.
Actores	Design mode user.
Precondiciones	El usuario debe encontrarse en el modo diseño.
Flujo normal	**Paso y Acción**
	1 El usuario pulsa el botón de mostrar/ocultar el menú del modo diseño.
Postcondiciones	El sistema muestra/oculta el menú del modo diseño.

Cuadro 6.25: Caso de uso 18.

Diagramas de casos de uso

A partir de las especificaciones de caso de uso se han creado los siguientes diagramas de casos de uso:

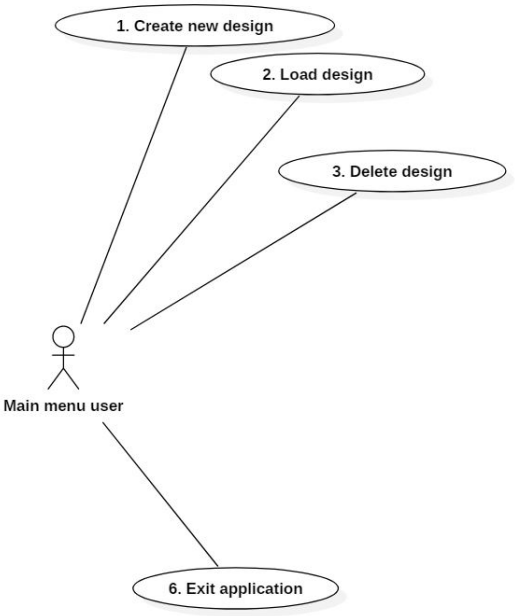


Figura 6.2: Diagrama de casos de uso del Main Menu User.

Figura 6.3: Diagrama de casos de uso del Design Mode User y el Creation Mode User.

6.2. Esquema de controles

Se han realizado una serie de esquemas para planificar las funciones que tendrán los botones de los controladores en cada modo de la aplicación.

En el menú principal, solo es necesario que los controladores permitan al usuario cambiar de página y seleccionar el nivel deseado. Para ofrecerle al usuario la manera más cómoda de interactuar con el menú se ha seguido la siguiente configuración de controles:

Figura 6.4: Esquema de controles del menú principal.

En el modo diseño, se amplía el número de acciones que el usuario puede realizar con respecto al menú principal, por lo que la complejidad del diseño de los controles también aumenta. Además, se dan casos en los que se utiliza el mismo botón para realizar una acción u otra dependiendo del contexto. La configuración de controles del modo diseño es la siguiente:

Figura 6.5: Esquema de controles del modo diseño.

En el modo creación encontramos una situación parecida con el modo diseño, ya que también se dan casos en los que el mismo botón se utiliza para varias acciones dependiendo del contexto. La configuración de controles del modo creación es la siguiente:

Figura 6.6: Esquema de controles del modo creación.

Capítulo 7

Desarrollo

En esta sección, se explicará cómo se han diseñado y desarrollado las principales clases de la aplicación.

7.1. Principales objetos de Unreal Engine utilizados

Debido a que las clases utilizadas heredan de clases del motor Unreal Engine, es necesario explicar los principales objetos de Unreal que han sido utilizados para el desarrollo de la aplicación.

7.1.1. Actor

"Un Actor es todo objeto que puede ser colocado en un nivel. Actor es una clase genérica que soporta transformación 3D, como puede ser la traslación, rotación y escala. Un Actor puede ser creado (spawned) o destruido a través de código del gameplay (C++ o Blueprints). En C++, AActor es la clase base de todos los actores."[33]

Dentro de la aplicación desarrollada la clase Actor será utilizada para representar los objetos y estructuras del edificio.

7.1.2. Pawn

"La clase Pawn es la clase base de todos los actores que pueden ser controlados por los jugadores o la IA. Un pawn es la representación física de un jugador o entidad IA dentro del mundo. Esto no solo significa que un Pawn determina el aspecto visual de un jugador o IA, sino también como interactúa con el mundo en términos de colisiones y otras interacciones físicas. Esto puede ser confuso en algunas circunstancias en las que un juego no tenga una malla de jugador o avatar visible dentro del juego. Sin embargo, un Pawn aún representa la localización física, rotación, etc. de un jugador o entidad dentro del juego. Un Character es un tipo especial de Pawn que tiene la habilidad de caminar por el mundo."[34]

Dentro de la aplicación desarrollada la clase Pawn será utilizada para implementar el jugador en el menú principal y en el modo creación, ya que no es necesario

que el jugador se mueva. Y la clase Character será utilizada para implementar el jugador en el modo diseño, ya que el jugador debe poder moverse por el nivel.

7.1.3. Blueprint

"El sistema de scripting visual Blueprint en Unreal Engine es un sistema de scripting de gameplay basado en el concepto de utilizar una interfaz basada en nodos para crear elementos del gameplay desde el propio editor de Unreal. Como ocurre con varios lenguajes de scripting comunes, es utilizado para definir clases orientadas a objetos u objetos dentro del motor. Normalmente los objetos definidos utilizando el sistema Blueprint son referidos coloquialmente como 'Blueprints'.

El sistema es extremadamente flexible y poderoso ya que proporciona a los diseñadores la habilidad de utilizar virtualmente el rango completo de conceptos y herramientas que generalmente solo están disponibles para los programadores. Además, el lenguaje de marcado específico de Blueprint disponible en la implementación en C++ de Unreal Engine permite a los programadores crear sistemas base que podrán ser extendidos por los diseñadores."[35]

Dentro de la aplicación desarrollada se ha utilizado el sistema Blueprint para ampliar las clases creadas en C++.

7.1.4. UMG - Widgets

"Unreal Motion Graphics UI Designer (UMG) es una herramienta de auditoría visual de interfaces de usuario que puede ser utilizada para crear elementos de la interfaz de usuario, como puede ser HUDs dentro del juego, menús u otros gráficos relacionados con la interfaz que se deseen presentar a los usuarios. En el centro de UMG se encuentran los Widgets, que son una serie de funciones prefabricadas que pueden ser utilizadas para construir las interfaces (botones, checkboxes, sliders, barras de progreso, etc.). Estos Widgets son editados en un Blueprint especializado, que utiliza dos pestañas para la construcción: la pestaña 'Designer' permite crear el diseño visual de la interfaz y sus funciones básicas, mientras que la pestaña 'Graph' proporciona la funcionalidad que se encuentra detrás de los Widgets creados."[36]

Dentro de la aplicación desarrollada se ha utilizado los Widgets para crear las interfaces de usuario que serán utilizadas por el jugador, y para controlar los eventos generados cuando el jugador interactúa con esas interfaces.

7.2. Diseño de las interfaces de usuario

Para el diseño de las interfaces de usuario se ha creado un mockup en la aplicación Figma. Y una vez que el diseño de las interfaces de usuario estaba terminado se han creado los widgets dentro del motor Unreal Engine.

7.2.1. Mockup inicial

Para la creación del mockup se ha seguido el estándar de Daydream creado por Google [17], en el que se indica el tamaño que deben tener los botones y la paleta

de colores que se debe seguir.

Menú principal

Para el diseño del menú principal se ha separado la pantalla de selección de los niveles de la barra de herramientas y de salir del juego. Por otro lado, se ha tratado de simplificar la interfaz lo máximo posible, mostrándole simplemente al usuario los niveles que puede seleccionar, el botón de añadir y borrar niveles y el botón de salir del juego.

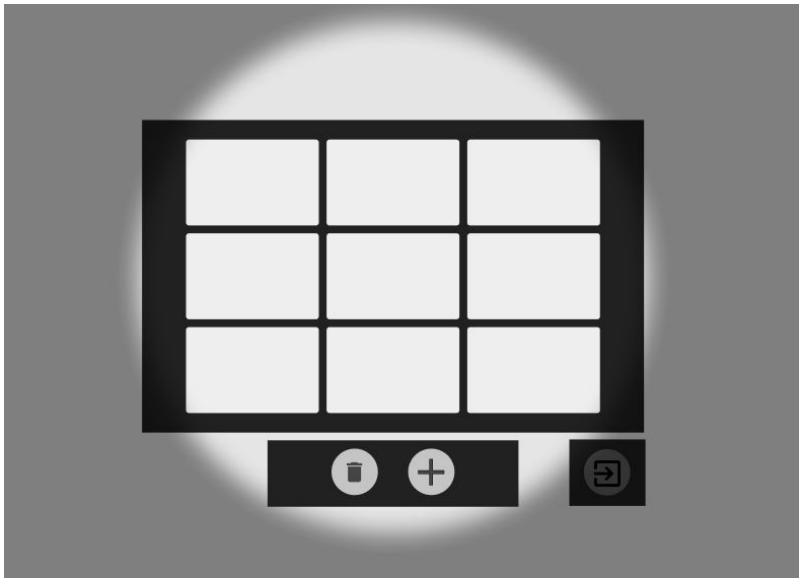
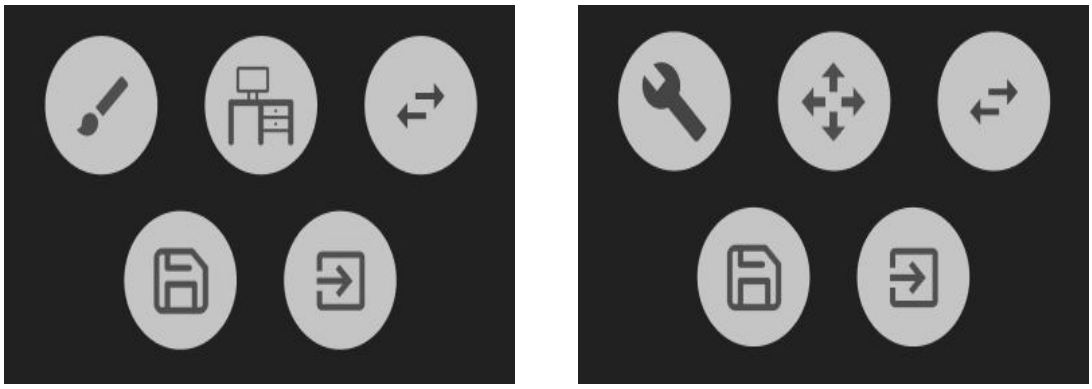


Figura 7.1: Mockup del menú principal.

Además, se ha simulado la porción de interfaz que el usuario podrá ver a través de las gafas de realidad virtual. Para poder ajustar la posición de los elementos y que le resulte cómodo al usuario utilizar la interfaz en todo momento.

Menús de los controladores

Para el diseño de los menús de los controladores del modo diseño y creación se ha realizado una interfaz reducida, ya que se debe tener en cuenta que estos menús irán acoplados en el controlador izquierdo del jugador. Además, se han utilizado sólo botones ya que es la manera más natural de interactuar con este tipo de interfaces en realidad virtual.



(a) Menú del controlador del modo diseño. (b) Menú del controlador del modo creación.

Figura 7.2: Mockups de los menús de los controladores.

Para los botones se han utilizado iconos que representan la acción que realiza cada uno de ellos, de manera que la interfaz se mantiene lo más simple posible.

Menús de selección

Para el diseño de los menús de selección del modo creación y diseño también se trató de simplificar lo máximo posible la interfaz. Por lo que no se utilizó ningún tipo de texto en ella, y se utilizaron iconos para representar la funcionalidad de cada botón.

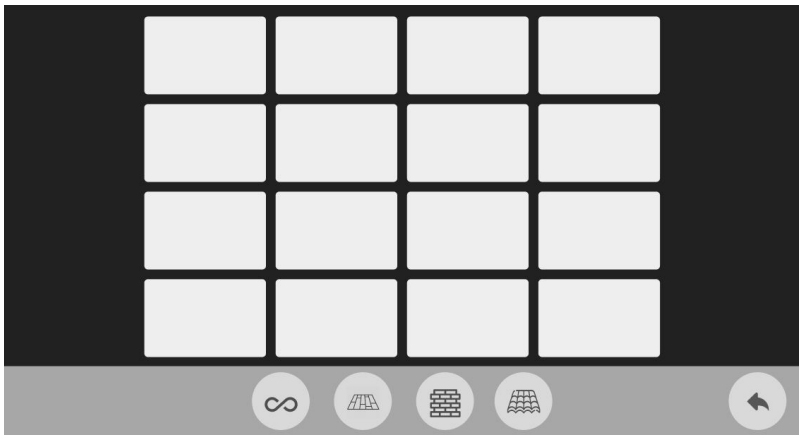


Figura 7.3: Mockup del menú de selección del modo creación.

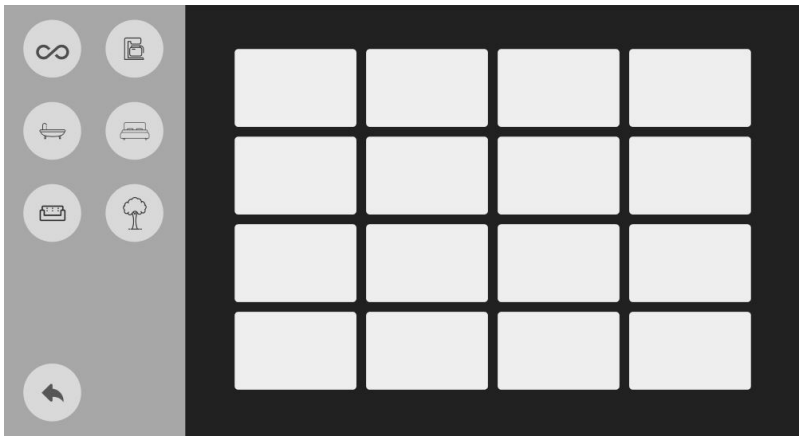


Figura 7.4: Mockup del menú de selección del modo diseño.

Para diferenciar los dos menús, se decidió que para el menú de selección del modo creación los botones irían situados en la parte inferior de la interfaz, y en el menú de selección del modo diseño los botones irían situados en el lateral izquierdo de la interfaz.

Menú de salir al menú principal

Por último, para el diseño del menú de salir al menú principal se creó una interfaz simple, que cuenta con un texto que pregunta al usuario si realmente quiere salir de la aplicación y dos botones para aceptar o rechazar la operación.

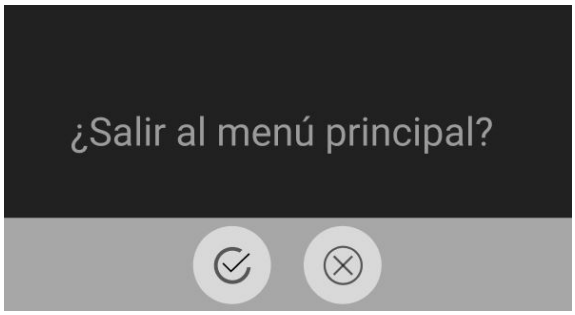


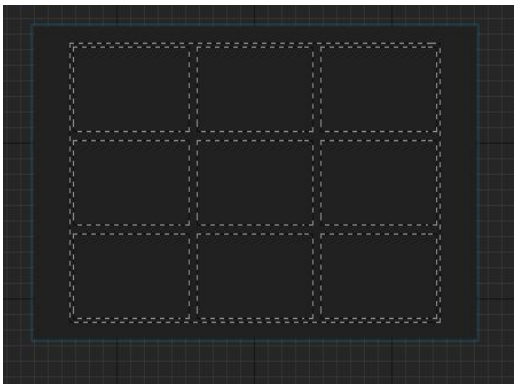
Figura 7.5: Mockup del menú de salir al menú principal.

7.2.2. Adaptación a Unreal Engine

A continuación se explicará cómo se ha adaptado la interfaz del menú principal al motor Unreal Engine. Se explicará solo el apartado visual del diseño del interfaces y cómo se controló uno de los eventos, ya que la lógica de las interfaces se explica más adelante.

En primer lugar, se creó un Widget llamando 'WBP_LevelGid' para el panel de selección del nivel, en este Widget se agregó el fondo creado en el mockup, y se creó un GridPanel, para colocar los niveles de forma ordenada, y un HorizontalBox, para colocar las páginas del menú.

(a) Componentes de WBP_LevelGid.



(b) Aspecto de WBP_LevelGid.

Figura 7.6: Componentes y aspecto del Widget WBP_LevelGid.

En segundo lugar, se creó otro Widget llamado 'WBP_LevelGridCard' para representar cada uno de los niveles que el usuario puede seleccionar, que consiste simplemente en un botón en el que se colocará la imagen que representa cada nivel, y un campo de texto, que indicará el nombre del nivel en caso de que no se encuentre la imagen del nivel. Este Widget será instanciado y añadido dentro del GridPanel del Widget 'WBP_LevelGrid' para cada uno de los niveles guardados, de manera que el panel se irá rellenando o vaciando a medida que se creen o eliminen diseños guardados.

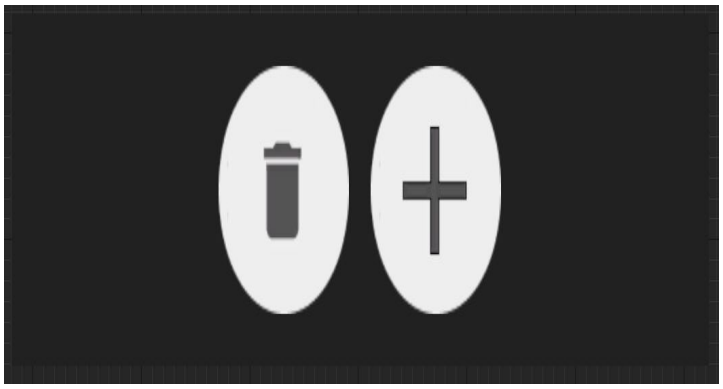
(a) Componentes de WBP_LevelGridCard.

(b) Aspecto de WBP_LevelGridCard.

Figura 7.7: Componentes y aspecto del Widget WBP_LevelGridCard.

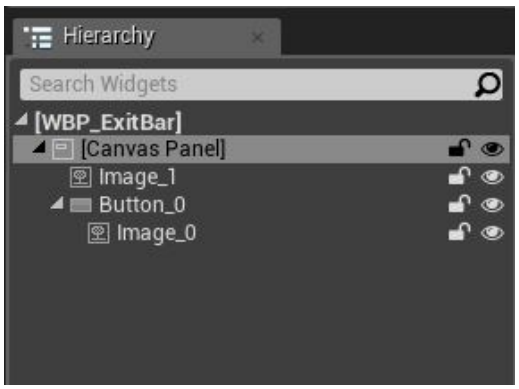
En tercer lugar, se crearon dos Widgets más para representar la barra de herramientas y la barra de salir del juego, llamados 'WBP_ActionBar' y 'WBP_ExitBar' respectivamente. Estos Widgets son muy simples, ya que solo consisten en los botones de añadir y borrar niveles en el caso de 'WBP_ActionBar', y en el botón de salir del juego en el caso de 'WBP_ExitBar'. En cada uno de los botones se ha añadido el icono correspondiente, y se ha asignado un fondo para cada uno de los estados (Normal, Hovered y Pressed) que sirven de guía para el jugador.

(a) Componentes de WBP_ActionBar

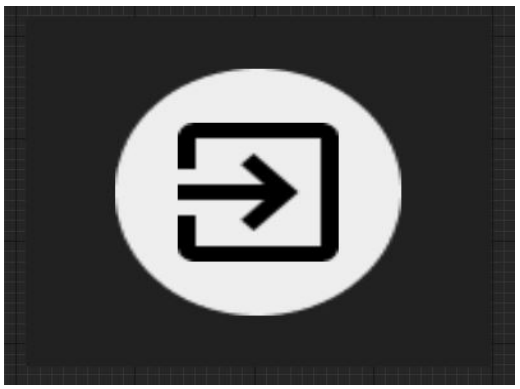


(b) Aspecto de WBP_ActionBar

Figura 7.8: Componentes y aspecto del Widget WBP_ActionBar.



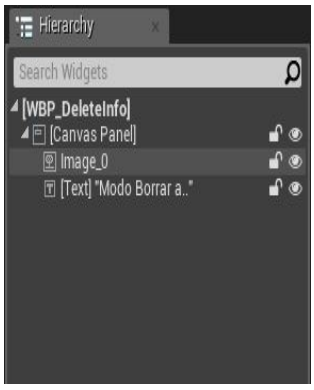
(a) Componentes de WBP_ExitBar.



(b) Aspecto de WBP_ExitBar.

Figura 7.9: Componentes y aspecto del Widget WBP_ExitBar.

En cuarto lugar, se creó un Widget llamado 'WBP_DeleteInfo', que consiste simplemente en un texto que indica que el modo borrar se ha activado.



(a) Componentes de WBP_DeleteInfo.

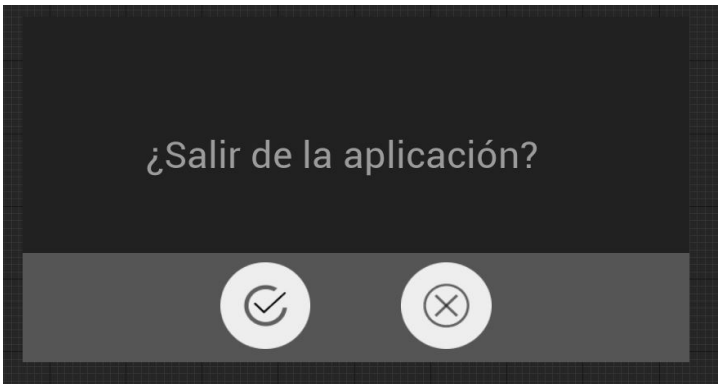
(b) Aspecto de WBP_DeleteInfo.

Figura 7.10: Componentes y aspecto del Widget WBP_DeleteInfo.

En quinto lugar, se creó un Widget llamado 'WBP_ExitGame', que solo es visible cuando el jugador pulsa el botón de salir del juego. Este Widget consiste en un campo

de texto que pregunta al usuario si realmente quiere cerrar la aplicación y los botones de aceptar y cancelar.

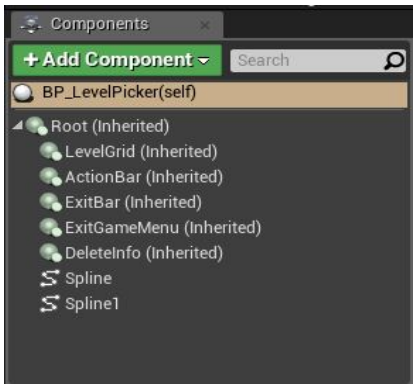
(a) Componentes de WBP_ExitGame.



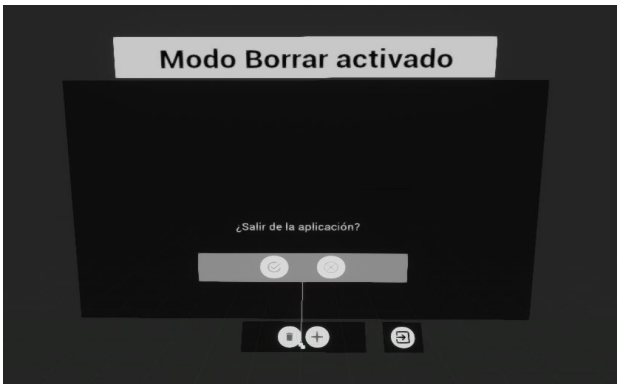
(b) Aspecto de WBP_ExitGame.

Figura 7.11: Componentes y aspecto del Widget WBP_ExitGame.

En sexto y último lugar, se juntaron todos los Widgets en un Blueprint llamado 'BP_LevelPicker', que permitió organizar los distintos elementos de la interfaz del menú principal, y colocarlos en el entorno 3D. Además, permitió controlar el evento de mostrar el menú de salir del juego.



(a) Componentes de BP_LevelPicker.



(b) Aspecto de BP_LevelPicker.

Figura 7.12: Componentes y aspecto del Blueprint BP_LevelPicker.

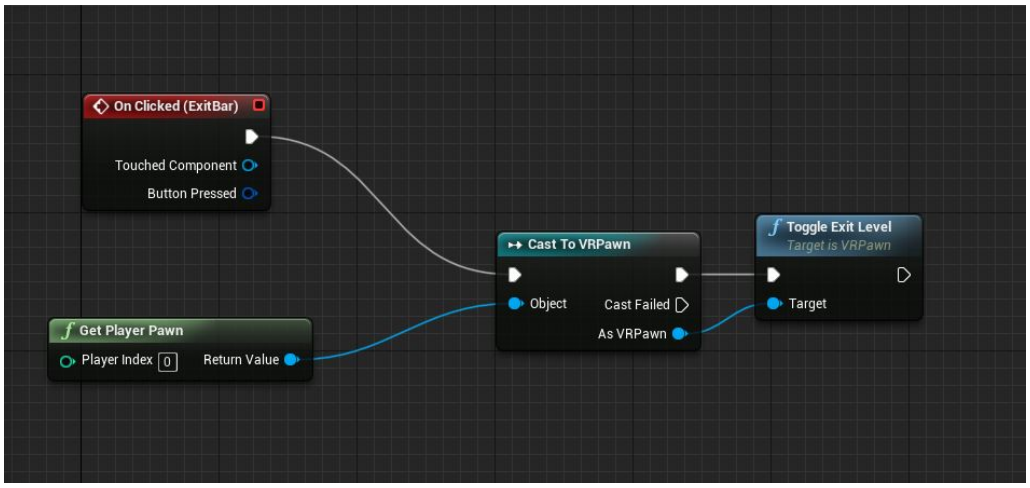


Figura 7.13: Blueprint que muestra el Widget de salir del juego.

La figura 7.13 muestra una función en Blueprint que detecta cuando se pulsa el botón de salir del juego, obtiene el Pawn del jugador, y llama a la función ToggleExitLevel() que se encuentra en la clase 'VRPawn', y se encarga de mostrar/ocultar el menú de salir del juego.

7.3. Diseño de los componentes del jugador

A continuación se explicará qué componentes tienen las clases VRPawn y VRCharacter, cómo se comportan durante los eventos BeginPlay() y Tick() y cómo están diseñados los Blueprints que heredan de estas clases. Además se explicará el funcionamiento de la clase HandControllerBase, de la que heredan todas las clases de los controladores de los modos creación y diseño, y cómo se lleva a cabo la gestión de los eventos producidos por los controladores.

7.3.1. VRPawn

Constructor

La clase VRPawn está compuesta principalmente por:

- Un SceneComponent al que se acoplan el resto de componentes.
- Un CameraComponent que permite al usuario ver el entorno virtual.
- Un StaticMeshComponent que representa el marcador de teletransporte utilizado para cambiar entre el modo creación y el modo diseño.
- Un PostProcessComponent que permite reducir el campo de visión del jugador al modificar su posición para evitar mareos.
- Dos WidgetComponent que representan los menús de selección de estructuras y de salir al menú principal.

Estos componentes han sido previamente declarados en el fichero .h, y luego han sido inicializados en el constructor de la clase.

Función BeginPlay()

Tras inicializar los componentes en el constructor, se procede a realizar los ajustes iniciales de la clase en el evento BeginPlay(), que es llamado al iniciar la aplicación o al instanciar la clase VRPawn.

En esta función se realizan las siguientes acciones:

- Se oculta el marcador de teletransporte.
- Se crean e inicializan los controladores derecho e izquierdo, acoplándolos al jugador, y utilizando las clases que han sido definidas por el Blueprint que hereda de esta clase, para detectar que tipo de controlador se debe asignar a cada mano.
- Se comprueba si el Blueprint que hereda de esta clase es el Blueprint del menú principal o el Blueprint del modo diseño. En caso de ser el Blueprint del menú principal no se realiza ningún ajuste más.
- Se ocultan los menús de selección y de salir al menú principal.
- Se crea el blinder que reducirá el campo de visión del jugador al desplazarse para evitar mareos, se le asigna un material previamente definido en fichero .h, y se le asigna un radio inicial al material que provoca que no sea visible para el jugador.
- Se inicializa el objeto BaseActor que ha sido previamente definido en el fichero .h, y que nos permitirá detectar cuando las estructuras que queremos colocar se encuentran en el suelo.
- Se desactiva la colisión de los objetos que se encuentren colocados en el mapa, para facilitar la selección de estructuras.
- Se rellena el menú de selección de estructuras con todos las estructuras disponibles.
- Se establece la posición inicial del jugador.

Función Tick()

Por último, se asignan las acciones que la clase debe realizar en el evento Tick(), que es llamado cada frame.

En esta función se realizan las siguientes acciones:

- Se actualiza el blinder en base a la velocidad actual del jugador.
- Se desactiva el puntero del controlador derecho, el cual permite detectar cuando se está apuntando hacia una estructura en el modo creación, o seleccionar una opción del menú de selección en el menú principal.
- En caso de que esté activado el modo de movimiento del mapa, se actualiza la posición del jugador respecto al mapa.

- En caso de que esté activado el modo de cambio entre el modo creación y el modo diseño, se actualiza la posición del marcador de teletransporte.
- En caso de que se esté colocando una estructura, se actualiza su posición en el mapa.
- En caso de que no se esté modificando la posición del mapa y el menú de selección no esté activo, se comprueba si el jugador está apuntando hacia una estructura. En caso de se que esté apuntando hacia una estructura se activa el puntero del controlador derecho.
- Si no se cumple nada de lo anterior, nos aseguramos de que el marcador de teletransporte se encuentra oculto.

Una vez que tenemos creada la clase VRPawn base, creamos los Blueprints que heredan de dicha clase y realizamos la configuración de cada uno de ellos.

En el caso del Blueprint del jugador del menú principal llamado 'BP_UIPawn', dejamos sin asignar la gran mayoría de variables ya que no nos son necesarias en el menú principal. La única variable que nos interesa es la del controlador derecho, a la que le asignamos la clase 'BP_UIPointerHandController' que hereda de la clase 'HandControllerBase'. La configuración del Blueprint es la siguiente:

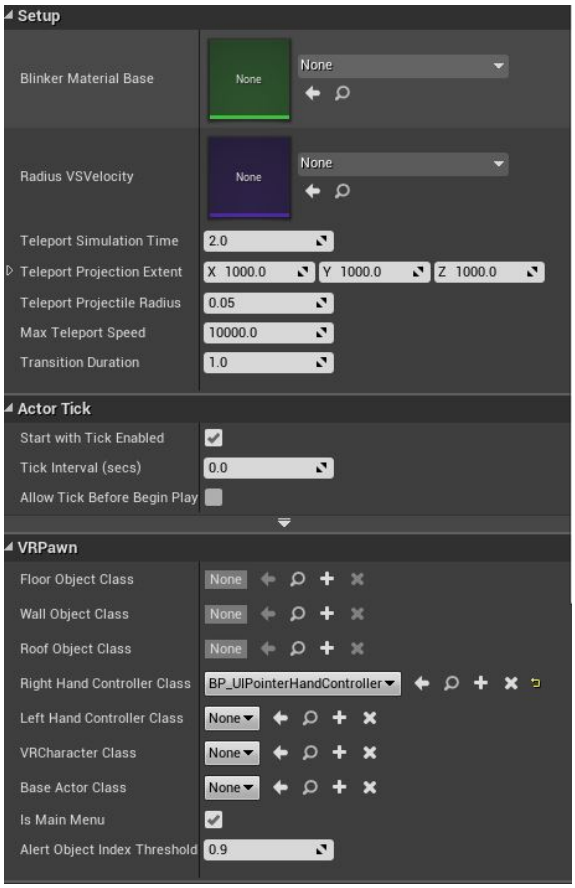


Figura 7.14: Configuración del Blueprint del jugador en el menú principal.

En el caso del Blueprint del jugador en el modo creación llamado 'BP_CreationModePawn', se asignan todas las variables necesarias ya que en el modo creación se hace uso de to-

das ellas. Además, asignamos las clases 'BP_CreationModeSelectHandController' y 'BP_CreationModeMenuHandController', que heredan de la clase 'HandControllerBase', al controlador derecho e izquierdo respectivamente. Por último, asignamos la clase 'BP_DesignModeCharacter' que nos permitirá cambiar al modo diseño, y la clase 'BP_BasePlatform' que nos permitirá detectar el terreno del mapa al colocar una estructura. La configuración del Blueprint es la siguiente:

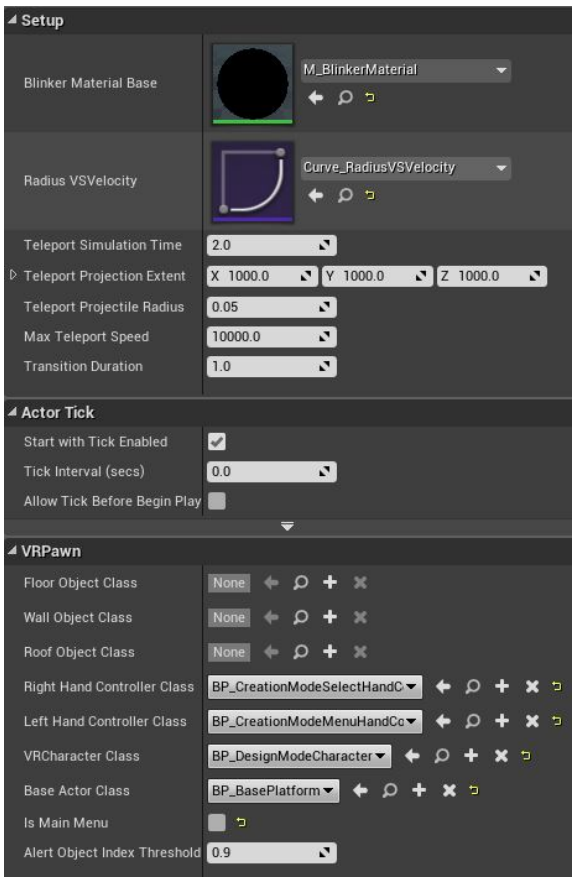


Figura 7.15: Configuración del Blueprint del jugador en el modo creación.

7.3.2. VRCharacter

Constructor

El constructor de la clase VRCharacter es bastante similar al de la clase VRPawn y está compuesto principalmente por:

- Un SceneComponent al que se acoplan el resto de componentes.
- Un CameraComponent que permite al usuario ver el entorno virtual.
- Un StaticMeshComponent que representa el marcador de teletransporte utilizado para mostrar al usuario el punto de teletransporte.
- Un PostProcessComponent que permite reducir el campo de visión del jugador al moverse para evitar mareos.
- Dos WidgetComponent que representan los menús de selección de objetos y de salir al menú principal.

- Un SplineComponent que representa el arco que se muestra junto al marcador de teletransporte durante la previsualización de un teletransporte.

Estos componentes han sido previamente declarados en el fichero .h, y luego han sido inicializados en el constructor de la clase.

Función BeginPlay()

Tras inicializar los componentes en el constructor, se procede a realizar los ajustes iniciales de la clase en el evento BeginPlay():

- Se oculta el menú de selección de objetos, el menú de salir al menú principal y el marcador de teletransporte.
- Se crea el blinder que reducirá el campo de visión del jugador al desplazarse para evitar mareos, se le asigna un material previamente definido en fichero .h, y se le asigna un radio inicial al material que provoca que no sea visible para el jugador.
- Se establece la altura del jugador.
- Se crean e inicializan los controladores derecho e izquierdo, acoplándolos al jugador y utilizando las clases que han sido definidas por el Blueprint que hereda de esta clase, para detectar que tipo de controlador se debe asignar a cada mano.
- Se rellena el menú de selección de objetos con todos los objetos disponibles.
- Se inicializa el objeto BaseActor que ha sido previamente definido en el fichero .h, y que nos permitirá detectar cuando los objetos que queremos colocar se encuentran en el suelo.
- Se recogen los bocetos que se encuentren en el nivel y se añaden al historial de bocetos.

Función Tick()

Por último, se asignan las acciones que la clase debe realizar en el evento Tick():

- Se actualiza el blinder en base a la velocidad actual del jugador.
- Se corrige la deriva de la cámara que se puede producir cuando el jugador se mueve físicamente.
- Se desactiva el puntero del controlador derecho, que nos indica cuando estamos apuntando hacia un objeto.
- En caso de que se esté colocando un objeto, se actualiza su posición en el espacio virtual y se oculta el marcador y el arco de teletransporte.
- Si el modo teletransporte está activado, se actualiza el marcador y el arco de teletransporte.

- Si no se está dibujando un boceto y el menú de selección no está activo, se comprueba si el jugador está apuntando hacia un objeto. En caso de se que esté apuntando hacia un objeto se activa el puntero del controlador derecho.
- Si no se está haciendo nada de lo anterior, nos aseguramos de que no se muestra ni el marcador ni el arco de teletransporte.

Una vez que se ha creado la clase base VRCharacter, creamos el Blueprint que hereda de dicha clase llamado 'BP_DesignModeCharacter' y realizamos su configuración. En esta configuración se asignan las clases 'BP_DesignModeSelectHandController' y 'BP_DesignModeMenuHandController', que heredan de la clase 'HandControllerBase', al controlador derecho e izquierdo respectivamente. Por otro lado, asignamos la clase 'BP_CreationModePawn', que nos permitirá cambiar al modo creación, la clase 'BP_BasePlatform' que nos permitirá detectar el terreno del mapa, y la curva 'RadiusVSVelocity', que nos permitirá ajustar el radio del blinder del jugador en base a la velocidad de este. Finalmente, asignamos los materiales necesarios para el arco de teletransporte y el blinder. La configuración del Blueprint es la siguiente:

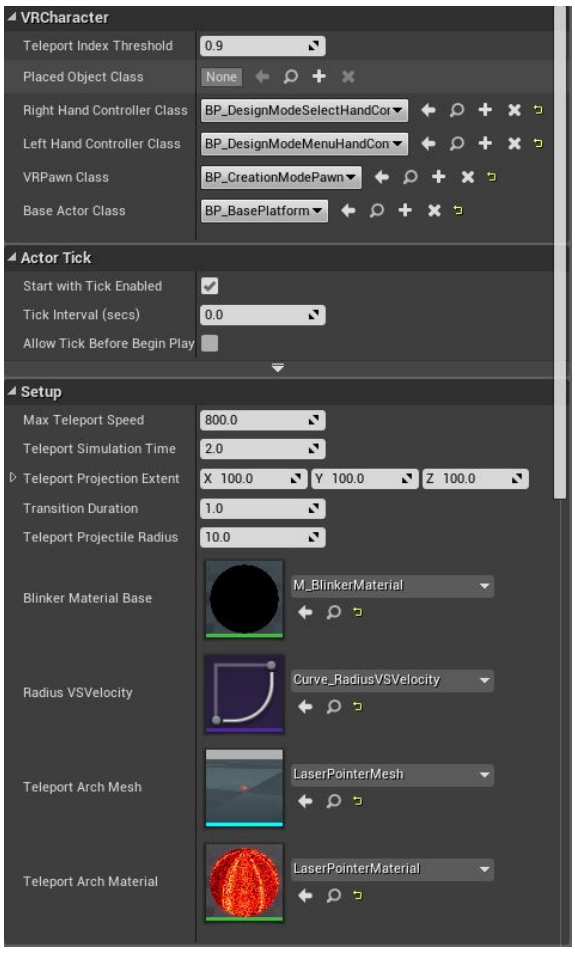


Figura 7.16: Configuración del Blueprint del jugador en el modo diseño.

7.3.3. HandControllerBase

La clase HandControllerBase es la base de la que heredan todas las clases que implementan los controladores. Por lo que es interesante ver cómo está definida esta

clase y cómo es su constructor.

Esta clase hace un gran uso del polimorfismo, ya que todos sus métodos son declarados como virtuales, y son implementados de forma diferente por las clases que heredan de ella. Por lo que se comporta como una interfaz entre las clases del jugador ('VRPawn' y 'VRCharacter') y las clases que implementan los controladores.

```
UCLASS()
class VRCHITECT_API AHandControllerBase : public AActor
{
  GENERATED_BODY()

public:
  // Sets default values for this actor's properties
  AHandControllerBase();

  void SetHand(EControllerHand Hand);
  virtual void RightTriggerReleased() {}
  virtual void RightTriggerPressed() {}
  virtual void LeftTriggerPressed() {}
  virtual void RightAButton() {}
   ...
```

Listing 7.1: Algunos de los métodos virtuales de la clase HandControllerBase.

Por su parte el constructor inicializa el MotionControllerComponent que se ha definido en el fichero .h, y que permite añadir un controlador a una clase, de manera que se ahorra esta declaración e inicialización en las clases que hereden de ella. Además, se desactiva el evento Tick(), ya que no es necesario y nos permite optimizar la aplicación.

```
AHandControllerBase::AHandControllerBase()
{
  PrimaryActorTick.bCanEverTick = false;

  //Creamos e inicializamos el controlador de movimiento.
  MotionController = CreateDefaultSubobject<
   UMotionControllerComponent>(TEXT("MotionController"));
  SetRootComponent(MotionController);
  MotionController->SetShowDeviceModel(true);
}
```

Listing 7.2: Constructor de la clase HandControllerBase.

7.3.4. Control de eventos

Respecto al control de eventos, es necesario crear en el menú de ajustes de Unreal Engine cada uno de los eventos que queremos capturar, ponerle un nombre, y asignarle uno o más botones de los controladores. Algunos de los eventos controlados son los siguientes:

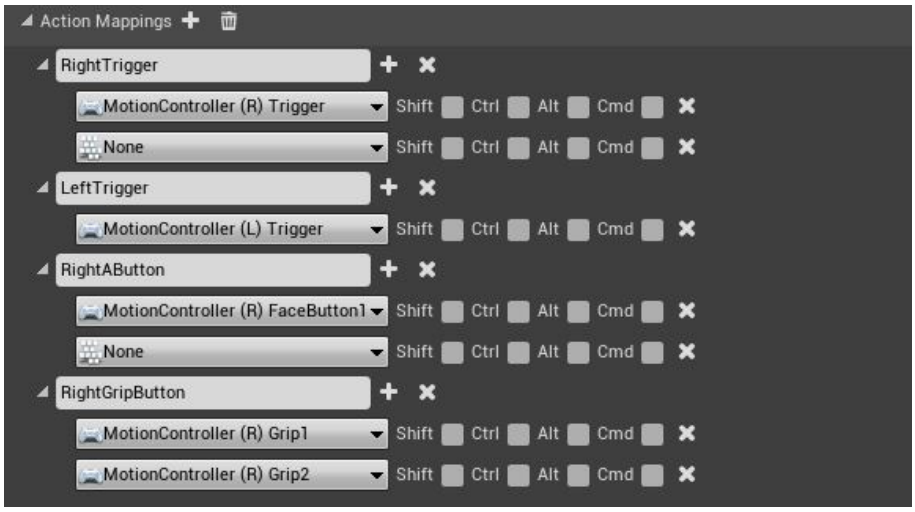


Figura 7.17: Configuración de eventos en el menú de ajustes.

Una vez que hemos creado el evento debemos añadirlo a la clase que va a gestionar ese evento e indicar la función a la que se debe llamar cuando ocurra. Por ejemplo, uno de los eventos controlados en la clase 'VRCharacter' es el siguiente:

```
1 void AVRCharacter::SetupPlayerInputComponent(UInputComponent*
    PlayerInputComponent)
2 {
3   //Asignar los eventos a cada boton de los controladores.
4   Super::SetupPlayerInputComponent(PlayerInputComponent);
5     ...
6   PlayerInputComponent->BindAction(TEXT("RightTrigger"),
     EInputEvent::IE_Pressed, this, &AVRCharacter::
     RightTriggerPressed);
7     ...
8 }
```

Listing 7.3: Control del evento RightTrigger en la clase VRCharacter.

7.4. Diseño de los objetos y estructuras

Para el diseño de las clases de los objetos y estructuras, se ha creado una clase base para cada tipo de objeto y estructura en C++ y luego se ha creado un Blueprint que hereda de esa clase para cada objeto y estructura.

Las clases C++ creadas han sido 'PlacedObject', para los objetos, y 'FloorObject', 'WallObject' y 'RoofObject', para los suelos, paredes y techos respectivamente. La razón por la que se crea una clase C++ para cada tipo es para poder detectar cuando se está apuntando hacia un objeto y cuando se está apuntando hacia una estructura. Y en el caso de que se esté apuntado hacia una estructura, poder diferenciar si se está apuntando hacia un suelo, una pared o un techo. Debido a que el diseño de estas clases es prácticamente idéntico, se explicará solo cómo se realizó la clase 'PlacedObject'.

Diseño de la clase C++

La clase cuenta con un componente de escena y un componente StaticMesh que permiten representar el objeto que se quiere colocar, las funciones para serializar y deserializar el objeto, el constructor y la función BeginPlay(). Las funciones para serializar y deserializar el objeto serán explicadas en el apartado de guardar y cargar diseños.

Por otro lado, el constructor es muy simple. Solo se encarga de inicializar el StaticMesh y el SceneComponent, y de acoplar el StaticMesh al SceneComponent. De esta manera se pueden representar los objetos del edificio.

```
1
2  APlacedObject::APlacedObject()
3  {
4    PrimaryActorTick.bCanEverTick = false;
5
6    //Creamos un componente root al que iran anclados el resto de
       componentes.
7    Root = CreateDefaultSubobject<USceneComponent>(TEXT("Root"));
8    SetRootComponent(Root);
9
10   //Creamos e inicializamos el mesh.
11   ObjectMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("
       ObjectMesh"));
12   ObjectMesh->SetupAttachment(Root);
13
14 }
```

Listing 7.4: Constructor de la clase PlacedObject.

Blueprints

Una vez que tenemos creada la clase 'PlacedObject', procedemos a crear los Blueprints que heredan de esta clase. Algunos de esos Blueprints son los siguientes:

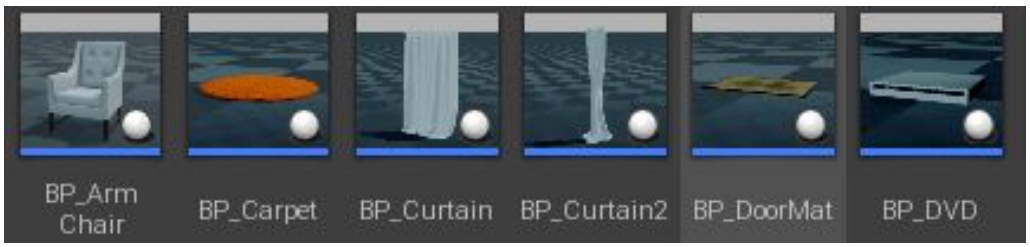


Figura 7.18: Blueprints que heredan de la clase PlacedObject.

Por último, es necesario modificar el canal de colisión de los objetos, de manera que no interfiera ni con el jugador ni con el canal de la cámara, para evitar problemas a la hora de colocar los objetos.

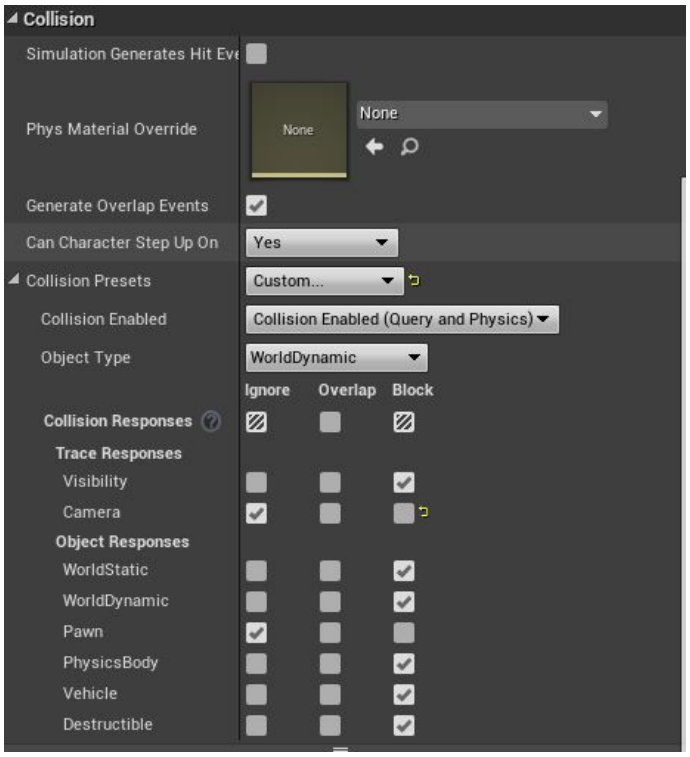


Figura 7.19: Configuración de la colisión de los objetos.

7.5. Sistema de movimiento

El jugador puede utilizar dos sistemas de movimiento en base a sus necesidades. Se explicarán las principales funciones de los sistemas de movimiento del modo diseño. Ya que el modo creación solo implementa el movimiento libre, y es muy similar al movimiento libre del modo diseño.

7.5.1. Movimiento libre

Para realizar el movimiento libre el jugador debe mover el joystick izquierdo hacia la dirección a la que desea moverse.

Como hemos explicado anteriormente, para controlar los eventos generados por los controladores se debe añadir el evento en el menú de ajustes de Unreal Engine, y posteriormente se añade en la clase deseada el control de los eventos.

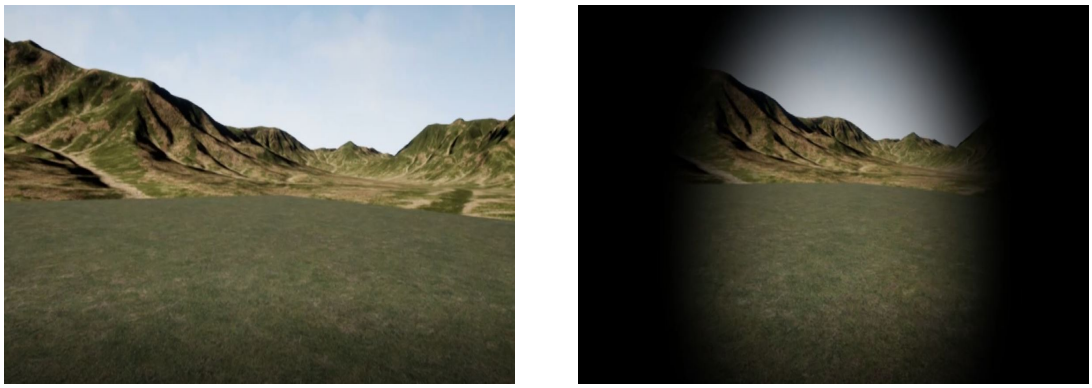
En este caso, se ha añadido en la clase 'VRCharacter' el siguiente control de eventos. En el que se llama a la función MoveForward() al detectar que el joystick se mueve en el eje Y, y a la función MoveRight() al detectar que el joystick se mueve en el eje X. Estas funciones se encargan de actualizar la posición del jugador:

```
1
2 void AVRCharacter::SetupPlayerInputComponent(UInputComponent*
    PlayerInputComponent)
3 {
4   //Asignar los eventos a cada boton de los controladores.
5   Super::SetupPlayerInputComponent(PlayerInputComponent);
```

```
6   PlayerInputComponent->BindAxis(TEXT("Forward"), this, &
      AVRCharacter::MoveForward);
7   PlayerInputComponent->BindAxis(TEXT("Right"), this, &AVRCharacter
      ::MoveRight);
8     ...
9 }
```

Listing 7.5: Configuración de los eventos de movimiento libre en la clase VRCharacter.

Además, en cada frame se actualiza el blinder del jugador, de manera que si el jugador se está moviendo se oscurece su vista. El resultado del oscurecimiento se puede ver en la figura 7.20.



(a) Vista normal. (b) Vista oscurecida.

Figura 7.20: Oscurecimiento de la vista durante el movimiento libre.

7.5.2. Teletransporte

Para realizar el movimiento por teletranspote el jugador debe mover el joystick derecho hacia arriba y mantenerlo en esa posición mientras selecciona el punto donde se quiere teletransportar. Una vez que ha elegido el punto de teletransporte, debe soltar el joystick para realizar el teletransporte.

Para detectar cuando el jugador mueve el joystick derecho en el eje Y, se crea el evento en el editor de Unreal Engine y se gestiona en la clase 'VRCharacter'. Para detectar cuando se mueve el joystick, se compara su posición con un límite establecido, si es superior al límite significa que el usuario está moviendo el joystick hacia arriba, por lo que se activa el modo teletransporte. En caso de que el joystick vuelva a su posición se realiza el teletransporte y se desactiva el modo teletransporte.

Una vez que se activa el modo teletransporte se detecta en la función Tick() que está activado, y se actualiza la posición del indicador de teletransporte y el arco de teletransporte cada frame.

```
1
2 void AVRCharacter::Tick(float DeltaTime)
3 {
4     ...
5   else if (isTeleportActive) {
6     //Si esta activado el modo teletransporte, actualizamos tanto
     el arco como el marcador de teletransporte.
```

```
7       UpdateDestinationMarker();
8     }
9       ...
10  }
```

Listing 7.6: Sección de la función que gestiona el modo teletransporte.

En la función UpdateDestinationMarker(), que gestiona la actualización del marcador y el arco de teletransporte, se comprueba si el usuario está apuntando hacia una posición del terreno válida y se actualiza el marcador y el arco de teletransporte.

Finalmente, una vez que el usuario suelta el joystick se realiza el teletransporte. Para ello se comprueba si el marcador se encuentra en una posición válida, y en caso de estarlo se obtiene su posición, se realiza un fundido de cámara a negro y se inicializa un timer con una duración de 1 segundo. Al finalizar el timer se coloca al jugador en la posición del marcador obtenida anteriormente y se devuelve la vista normal a la cámara. De esta manera se crea un efecto de teletransporte.

En la figura 7.21 se puede ver un ejemplo del marcador y el arco de teletransporte, que permiten al usuario seleccionar el punto al que desea moverse.

Figura 7.21: Marcador y arco de teletransporte.

7.6. Sistema de bocetos

Para realizar un boceto el usuario debe pulsar el trigger del controlador derecho y mantenerlo pulsado mientras realiza el boceto. Para finalizar la creación del boceto debe soltar el botón del trigger. Por otro lado, si desea borrar el último boceto creado debe pulsar el botón 'A' del controlador derecho.

El sistema de bocetos se implementa en las clases 'VRCharacter', 'DesignModeSelectHandController' y 'Stroke'. La clase 'VRCharacter' se encarga de detectar cuando se generan los eventos de pulsación del trigger y del botón 'A' del controlador derecho, la clase 'DesignModeSelectHandController' se encarga de controlar la creación y borrado del boceto y de guardar el historial de bocetos, y la clase 'Stroke' se encarga de crear y ajustar los trazados del boceto y de guardar el historial de posiciones del controlador al crear un boceto.

7.6.1. Creación de bocetos

Como hemos comentado, para la creación de bocetos se deberá detectar el evento de pulsación del trigger derecho, que indica el inicio de la creación del boceto, y el evento de soltar el trigger derecho, que indica el fin de la creación del boceto. Una vez se detectan, se llama al método correspondiente en la clase 'DesignModeSelectHandController'.

Debido a que el evento de soltar el trigger derecho puede tener más de una función en el modo diseño (seleccionar un objeto del mapa, finalizar un boceto, etc.), el control se realiza en la implementación del método en el fichero .cpp. Sin embargo, el evento de pulsar el trigger derecho solo tiene la función de comenzar a realizar un boceto, por lo que se puede realizar el control en la definición del método en el fichero .h.

```
public:
//Realizamos el control en la definicion del metodo.
void RightTriggerPressed() {

  if (RightHandController) { if (bCanDraw) { RightHandController->
   RightTriggerPressed(); } }
};
```

Listing 7.7: Función que controla el evento de soltar el trigger derecho.

```
void AVRCharacter::RightTriggerReleased()
{
    ...
  //Si se esta en el modo dibujar, indicar al controlador derecho
   que se ha pulsado el trigger.
  if (RightHandController) { if (bCanDraw) { RightHandController->
   RightTriggerReleased(); } }
}
```

Listing 7.8: Sección de la función que controla el evento de pulsar el trigger derecho.

Una vez que se ha realizado el control del evento, el resto del proceso de creación y borrado de los bocetos se realiza en las clases 'DesignModeSelectHandController' y 'Stroke'.

Al realizar la pulsación del trigger, se crea el objeto del boceto y se establece su posición al valor de la posición del controlador derecho.

```

void ADesignModeSelectHandController::RightTriggerPressed()
{
  //Creamos un nuevo stroke y lo colocamos en la posicion del
   controlador.
  if (StrokeClass) {
    CurrentStroke = GetWorld()->SpawnActor<AStroke>(StrokeClass);
    CurrentStroke->SetActorLocation(GetActorLocation());
  }
}
```

Listing 7.9: Función que crea un nuevo boceto y establece su posición.

Una vez que se ha definido el boceto, el proceso de creación de este continúa en la función Tick() de la clase 'DesignModeSelectHandController'. Dentro de esta

función se actualiza el boceto en cada frame, de manera que el usuario puede ver al instante los trazados del boceto que realiza. Para realizar la actualización del boceto se utiliza una función auxiliar.

En la función auxiliar que se encarga de actualizar el boceto en cada frame se crean dos static meshes, un cilindro que representa el trazado que el usuario ha realizado en el frame actual y una esfera que sirve como punto de unión entre la sección del trazado actual y la anterior, de manera que no quedan huecos entre los trazados. Además, se ajusta la escala, la rotación y posición tanto de la esfera como del cilindro para permitir al usuario realizar giros y trazados rápidos sin que se produzcan deformaciones en el boceto. Finalmente, se guardan todas las posiciones del controlador registradas durante la creación del boceto, para poder recuperar el boceto al cargar un diseño.

Por último, una vez que se detecta que el usuario deja de pulsar el trigger derecho se guarda el boceto en el historial de bocetos, y se pone a null el objeto del boceto para evitar que se siga actualizando en la función Tick().

En la figura 7.22 se puede ver un ejemplo de creación de bocetos.

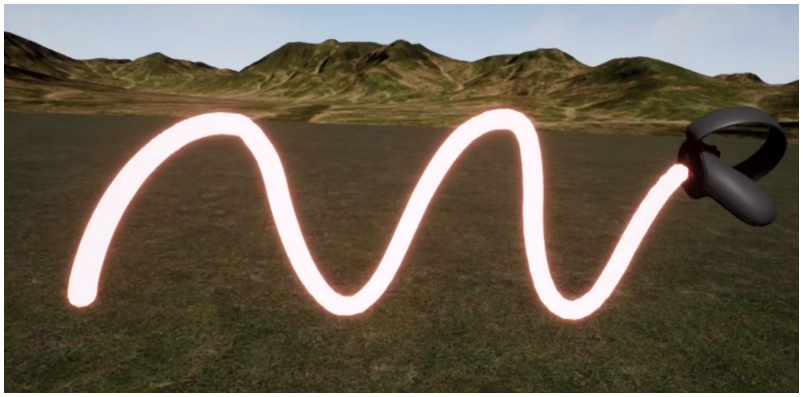


Figura 7.22: Creación de bocetos.

7.6.2. Borrado de bocetos

Para el borrado de bocetos es necesario detectar en la clase 'VRCharacter' el evento de pulsación del botón 'A' en el controlador derecho, y llamar a la función RightAButton() de la clase 'DesignModeSelectHandController'.

Dentro de la función RightAButton() solo es necesario recuperar el último boceto añadido al historial de bocetos y eliminarlo.

```
void ADesignModeSelectHandController::RightAButton()
{
  //Eliminamos el ultimo stroke creado.
  if (StrokeHistory.Num() > 0) {
    AStroke* StrokeTemp = Cast<AStroke>(StrokeHistory.Pop());;
    StrokeTemp->Destroy();
  }
}
```

Listing 7.10: Función que elimina un boceto.

7.7. Colocación de objetos y estructuras

Debido a que el sistema de colocación de objetos y el sistema de colocación de estructuras son muy similares, se explicará solo el sistema de colocación de objetos y se destacarán las principales características de cada modo al colocar los objetos y las estructuras.

Para la colocación de objetos el usuario deberá seleccionar en el menú de selección el objeto que desea colocar, y seleccionar la posición donde colocarlo. Una vez seleccionada la posición deberá pulsar el trigger derecho para confirmar la posición del objeto.

7.7.1. Sistema de menús

Para que el usuario pueda ver los objetos que puede colocar en el entorno es necesario realizar un sistema de menús que le permita filtrar los objetos por categorías y seleccionar el objeto deseado.

Para ello utilizaremos las clases 'VRCharacter', 'DesignModeSelectionMenu' y 'DesignModeSelectionMenuCard'. La clase 'VRCharacter' se encarga de actualizar la interfaz de usuario, la clase 'DesignModeSelectionMenu' implementa el menú de selección, y la clase 'DesignModeSelectionMenuCard' representa cada uno de los botones del menú que el usuario puede seleccionar.

Para realizar el filtrado de los objetos se detecta por medio de Blueprints cuando el usuario pulsa uno de los botones de categoría, y se llama la función ReloadObjectList() de la clase 'VRCharacter'. Esta función actualiza el contenido del menú, y recibe como parámetro el directorio donde se encuentran los objetos de la categoría seleccionada.

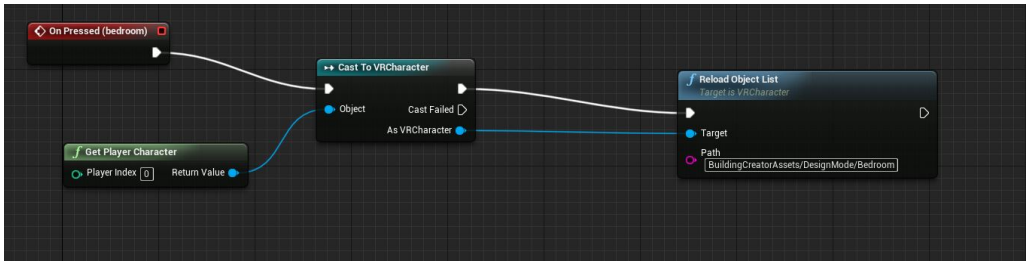


Figura 7.23: Blueprint que actualiza el menú de selección.

Dentro de la función ReloadObjectList() se recuperan los objetos del directorio que se pasa por parámetro, se crean los botones que representan cada uno de los objetos y se actualiza el contenido del menú.

Además, en cada uno de los botones que se añaden al menú de selección se crea un listener que detecta cuando son pulsados. Una vez que se genera el evento de pulsación se llama a la función ActionButtonClicked() de la clase 'DesignModeSelecionMenuCard'.

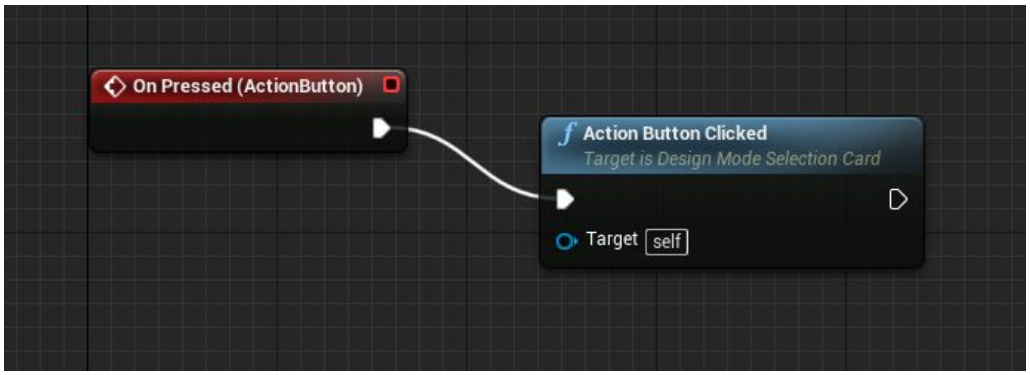


Figura 7.24: Blueprint que detecta la pulsación de un botón del menú de selección.

En la figura 7.25 se pueden ver algunos del los objetos que pueden ser seleccionados en el menú de selección del modo diseño.

Figura 7.25: Contenido del menú de selección del modo diseño.

7.7.2. Sistema de colocación

La colocación de un objeto comienza cuando el jugador selecciona uno de los objetos del menú de selección. Tras realizar la acción de pulsar el botón se genera el evento que llama a la función ActionButtonClicked() de la clase 'DesignModeSelectionMenuCard'.

Dentro de la función ActionButtonClicked() se recupera el Blueprint del objeto correspondiente al botón pulsado, se recupera la instancia de la clase 'VRCharacter' actual, se indica a la clase 'VRCharacter' la clase del objeto que queremos colocar, y llamamos a la función PlaceObject() de la clase 'VRCharacter'.

```
1 void UDesignModeSelectionCard::ActionButtonClicked()
2 {
3     ...
4   //En el design mode  obtenemos el blueprint correspondiente, se
     lo asignamos al PlayerCharacter y le indicamos que coloque el
     objeto.
5   FString res = "/Game/" + Path + "/" + ObjectNameS + "." +
     ObjectNameS + "_C";
6   auto ObjClass = LoadClass<APlacedObject>(nullptr, *FString(res));
7   if (!ObjClass) return;
```

```
8   PlayerCharacter = Cast<AVRCharacter>(UGameplayStatics::
      GetPlayerCharacter(GetWorld(), 0));
9   PlayerCharacter->SetObjectClass(ObjClass);
10  PlayerCharacter->PlaceObject();
11    ...
12 }
```

Listing 7.11: Sección de la función que crea el objeto a colocar.

Dentro de la función PlaceObject() se oculta el menú de selección, se crea una instancia del objeto a colocar para permitir al usuario posicionar el objeto en el mapa, y se activa el modo colocación de objetos.

Una vez que nos encontramos en el modo colocación de objetos, la actualización de la posición del objeto se realizará en la función Tick() de la clase 'VRCharacter'. Que se encarga de llamar en cada frame a la función UpdatePlacedObject() para ajustar la posición del objeto, y de asegurarse de que no se muestra ni el marcador ni el punto de teletransporte.

Dentro de la función UpdatePlacedObject() se modifica la rotación y la posición del objeto en caso de que el usuario mueva el joystick derecho en el eje X o en el eje Y respectivamente. Además, se detecta el punto en el terreno al que el usuario está apuntando con el controlador derecho, y se actualiza la posición del objeto a colocar.

Finalmente, una vez que el usuario está satisfecho con la posición del objeto a colocar, debe pulsar el trigger derecho para crear otra instancia del objeto en la posición seleccionada. La razón por la que se crea otra instancia del objeto, en lugar de colocar la instancia que se utilizó para previsualizar la posición del objeto, es para permitir al usuario colocar múltiples instancias del mismo objeto sin tener que volver a seleccionarlo en el menú.

Por lo que cuando el jugador confirma la posición del objeto a colocar se crea una nueva instancia del objeto, se le asigna la posición del objeto temporal que se utilizó para previsualizar la posición, y se modifica sus canales de colisión. De manera que se pueda seleccionar posteriormente en el modo edición, ya que se activa la colisión con el canal de la cámara y del pawn utilizados para detectar los objetos del mapa.

Una vez que el usuario ha terminado de colocar todas las instancias del objeto que crea necesarias debe pulsar el botón 'A' del controlador derecho. Al producirse el evento de pulsación se elimina la instancia del objeto utilizada como previsualización, se sale del modo colocación de objetos, y se resetea el modificador de posición en el eje Z que permite ajustar la altura a la que se coloca un objeto.

En la figura 7.26 se puede ver un ejemplo de colocación de objetos. En este caso se está colocando una pila de libros sobre un escritorio.

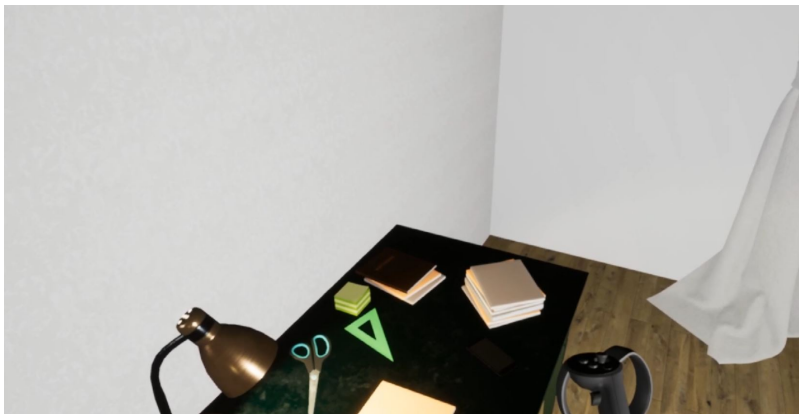
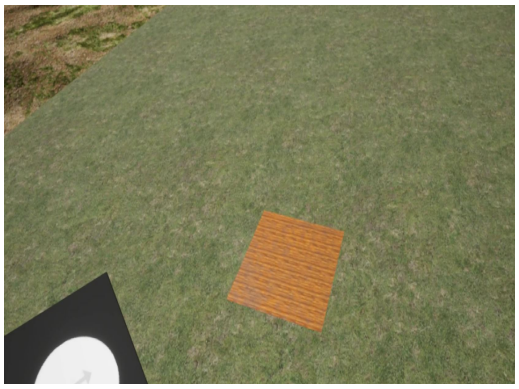


Figura 7.26: Ejemplo de colocación de un objeto.

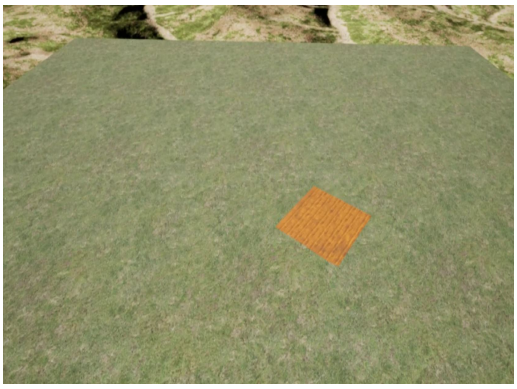
Tras explicar cómo funciona el sistema de colocación de objetos y estructuras, se destacarán los principales ajustes que el usuario puede realizar al colocar un objeto o estructura.

Colocación de estructuras

Una vez que el usuario selecciona una estructura del menú de selección puede colocarla en cualquier posición válida del terreno. Además, puede rotar la estructura o acoplarla a otra estructura previamente colocada. En la figura 7.27 se puede ver un ejemplo de una estructura con distinta posición y rotación.



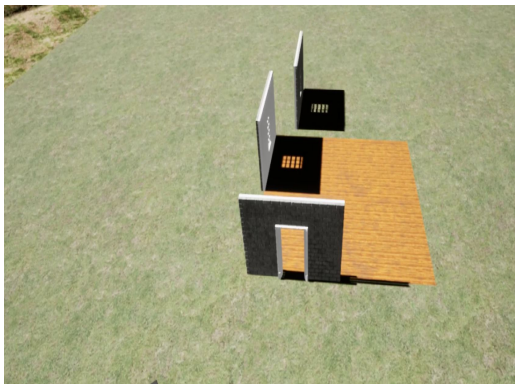
(a) Suelo sin rotar.



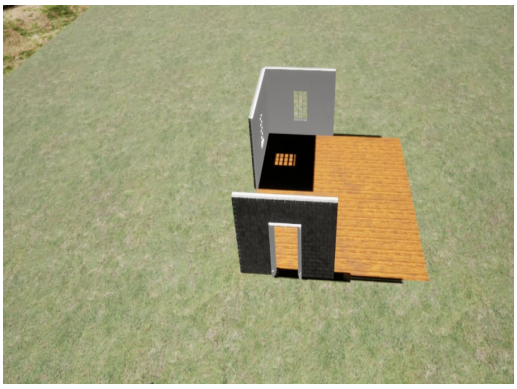
(b) Suelo rotado.

Figura 7.27: Diferentes posiciones y rotaciones al colocar un suelo.

Para acoplar una estructura a otra, el usuario debe apuntar hacia la estructura a la que quiere acoplar la estructura que está colocando actualmente. De esta manera, se ajusta automáticamente la posición y rotación de la estructura que se está colocando. En la figura 7.28 se puede observar un ejemplo de acoplamiento de estructuras, en el que una pared ajusta su rotación y posición automáticamente al detectar que se encuentra sobre un suelo.



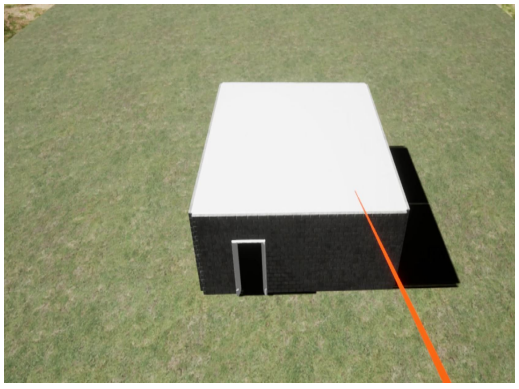
(a) Estructura sin acoplar.



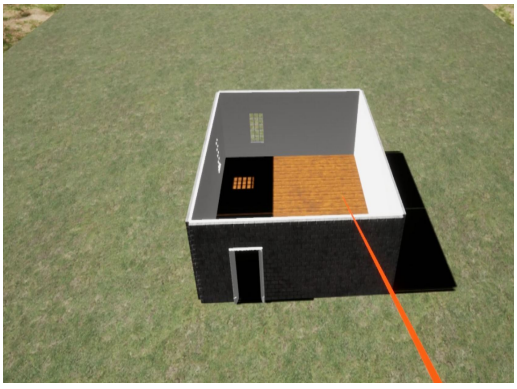
(b) Estructura acoplada.

Figura 7.28: Ejemplo de acoplamiento de una estructura.

Finalmente, el usuario puede ocultar el techo del edificio en cualquier momento, de manera que no obstaculiza a la hora de seleccionar otras estructuras.



(a) Techo visible.



(b) Techo oculto.

Figura 7.29: Modificación de la visibilidad del techo del edificio.

Colocación de objetos

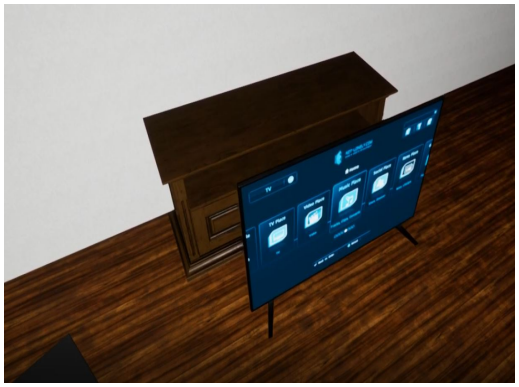
Una vez que el usuario selecciona un objeto del menú de selección puede colocarlo en cualquier posición válida del terreno. Además, puede rotar el objeto o situarlo sobre otro objeto ya colocado. En la figura 7.30 se puede observar un ejemplo de rotación de objetos.

(a) Sofá rotado hacia la derecha.

(b) Sofá rotado hacia la izquierda.

Figura 7.30: Rotación de un objeto.

Para colocar un objeto sobre otro, el usuario debe apuntar hacia el objeto sobre el que quiere colocar el objeto que se está colocando actualmente. De este modo, se ajusta automáticamente la posición del objeto que se está colocando. En la figura 7.31 se puede observar un ejemplo de colocación automático de una televisión sobre un mueble.



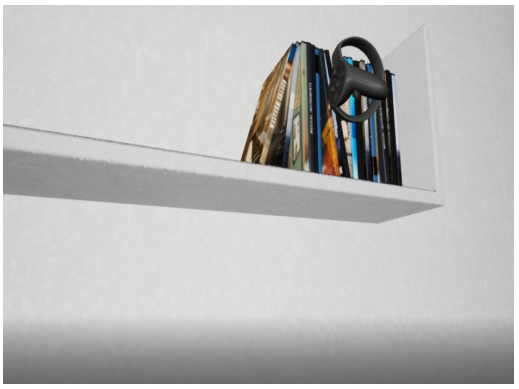
(a) TV antes de colocarla sobre el mueble.

(b) TV después de colocarla sobre el mueble.

Figura 7.31: Ejemplo de colocación de un objeto sobre otro.

Finalmente, el usuario puede ajustar la posición en el eje Z de los objetos al colocarlos. Ya que en el caso de objetos pequeños, o que se encuentren en posiciones complicadas, se puede dar la situación en la que el ajuste automático no coloque el objeto con suficiente precisión. En la figura 7.32 se puede observar como el usuario puede ajustar ligeramente la posición del objeto con precisión, de manera que el objeto quede colocado exactamente donde el usuario desea.

(a) DVDs sobre una repisa sin ajuste.



(b) DVDs sobre una repisa con ajuste.

Figura 7.32: Ejemplo de ajuste del eje Z de un objeto.

7.8. Sistema de guardado de diseños

El jugador puede crear, cargar y borrar diseños guardados a través del menú principal, y guardar un diseño desde el mapa de diseño de edificios. Para gestionar los diseños guardados se utilizan las clases 'LevelSaveGame', para guardar los datos de cada diseño guardado, y 'LevelSaveGameIndex', para almacenar y organizar todos los diseños guardados.

7.8.1. Crear diseños

Para crear un nuevo diseño el usuario deberá pulsar el botón de crear un nuevo diseño que se encuentra en la barra de acción del menú principal. Una vez que se genera el evento de pulsación del botón, se llama a la función AddLevel() de la clase 'LevelPicker', que implementa el menú principal.

```
1 private:
2   UFUNCTION()
3   void AddButtonClicked() { if (ParentPicker) ParentPicker->
     AddLevel(); }
```

Listing 7.12: Función que crea un nuevo diseño.

Dentro de la función AddLevel() se llama a la función Create() de la clase 'LevelSaveGame', que se encarga de crear un nuevo diseño y añadirlo al índice, se actualiza el menú principal para que muestre el nuevo diseño creado, y nos aseguramos de que el modo borrar diseños guardados esta desactivado.

Dentro de la función Create() se crea el fichero del diseño guardado, se le asigna un identificador único, se realiza el primer guardado del diseño, se le asigna una imagen por defecto y se añade el diseño guardado al índice de diseños guardados.

Tras realizar estos pasos ya se ha creado un nuevo diseño y se puede acceder a el a través del menú principal. En la figura 7.27 se puede ver un ejemplo de creación de diseños. Además, se puede observar como se le asigna una imagen por defecto a los dos diseños creados.

Figura 7.33: Creación de diseños.

7.8.2. Guardar diseños

Para guardar un diseño el usuario deberá pulsar el botón de guardado de diseños que se encuentra en el menú del controlador izquierdo. Una vez que se genera el evento se llama a la función SaveGame() de la clase 'VRCharacter' o 'VRPawn', dependiendo de si nos encontramos en el modo creación o diseño.

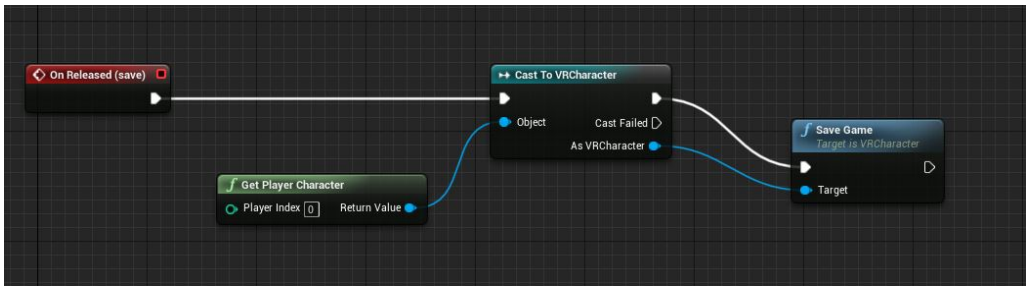


Figura 7.34: Blueprint que llama a la función de guardado de diseños.

Dentro de la función SaveGame() se realiza el guardado del diseño creado a través del GameMode actual y se muestra una alerta que indica que se ha guardado el diseño.

Dentro de la función del GameMode se recupera el diseño actual, se realiza una captura de pantalla y se almacena con el mismo nombre que el de el diseño guardado actual, se serializan todos los objetos, estructuras y bocetos del nivel y se guarda el diseño.

Para guardar los objetos, estructuras y bocetos creados, se obtienen todos y se serializan en un Array de Struct para cada tipo.

```
1
2 void ULevelSaveGame::SerializeFromWorld(UWorld * World)
3 {
4   //Reseteamos los arrays de objetos.
5   Strokes.Empty();
6     ...
7
8   //Rellenamos los arrays de objetos.
```

```
9   for (TActorIterator<AStroke> StrokeItr(World); StrokeItr; ++
      StrokeItr) {
10    Strokes.Add(StrokeItr->SerializeToStruct());
11  }
12 }
```

Listing 7.13: Ejemplo de serialización de los bocetos.

Un ejemplo de serialización es el que se realiza para los bocetos. En él se guarda la clase del boceto y el historial de posiciones del controlador al crearlo, para poder realizar una réplica del boceto original al cargar un diseño.

En la figura 7.29 se puede observar como aparece el mensaje de guardado de diseño en el menú del controlador al guardar un diseño.

Figura 7.35: Mensaje de guardado de diseños.

7.8.3. Cargar diseños

Para cargar un diseño el usuario deberá seleccionarlo en el menú principal. En la figura 7.30 se puede ver el menú de selección de diseños guardados que se encuentra en el menú principal.

Figura 7.36: Menú de selección de diseños.

Una vez que el usuario pulsa sobre uno de los botones del menú de selección se genera el evento de pulsación, se muestra al usuario una pantalla de carga y se carga

el nivel de diseño de edificios, pasándole como parámetro el nombre del diseño que se desea cargar.

Al iniciar el nivel ocurren dos acciones en el GameMode:

- Se llama a la función Init(), que se encarga de recuperar el nombre del diseño guardado pasado por parámetro.
- Se llama a la función BeginPlay(), que se encarga de ocultar la pantalla de carga y llamar a la función Load(), que deserializa e instancia los objetos, estructuras y bocetos guardados en el fichero del diseño guardado.

Para deserializar los objetos, estructuras y bocetos guardados se recorren los Arrays que se crearon durante la serialización, y se recupera e instancia los objetos, estructuras y bocetos guardados en el nivel.

```
1
2 void ULevelSaveGame::DeserializeToWorld(UWorld * World, AActor*
    BaseActor)
3 {
4   //Borramos todos los objetos del world actual.
5   ClearWorld(World);
6
7   //Deserializamos e instanciamos todos los objetos guardados.
8   for (FStrokeState State : Strokes) {
9     AStroke* Stroke = AStroke::SpawnAndDeserializeFromStruct(World,
      State);
10  }
11 }
```

Listing 7.14: Ejemplo de deserialización de los bocetos.

Un ejemplo de deserialización es el que se realiza para los bocetos. En él se instancia un boceto y se recorren las posiciones del controlador guardadas al crearlo, permitiendo crear una réplica del boceto original.

```
1 AStroke * AStroke::SpawnAndDeserializeFromStruct(UWorld * World,
    FStrokeState & StrokeState)
2 {
3   //Deserializamos e inicializamos el objeto.
4   AStroke* Stroke = World->SpawnActor<AStroke>(StrokeState.Class);
5   for (FVector ControlPoint : StrokeState.ControlPoints) {
6     Stroke->Update(ControlPoint);
7   }
8   return Stroke;
9 }
```

Listing 7.15: Función que deserializa un boceto.

7.8.4. Borrar diseños

Para borrar un diseño el usuario debe activar el modo borrado de diseños desde el menú principal. Y una vez que esté activado, pulsar sobre los diseños que desea borrar. Tras pulsar el botón de activación del modo borrado de diseños se llama a través de Blueprints a la función ToggleDelete() de la clase 'VRPawn'.

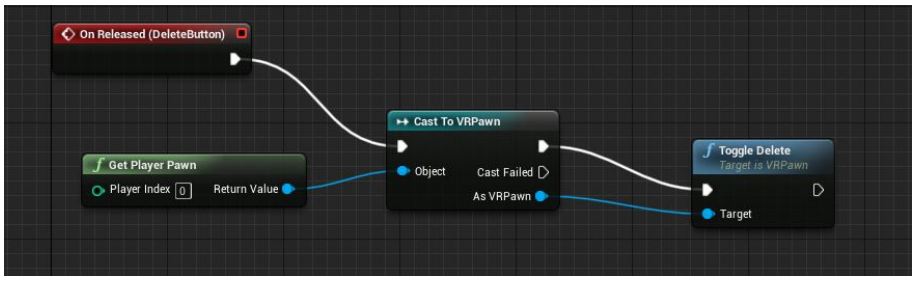


Figura 7.37: Blueprint que llama a la función de borrado de diseños.

Dentro de la función ToggleDelete() se recupera la instancia de la clase 'LevelPicker' actual y se llama a la función ToggleDeleteActive().

```
void AVRPawn::ToggleDelete()
{
  if (bIsMainMenu) {
    //Activamos/desactivamos el modo borrar.
    for (TActorIterator<ALevelPicker> LevelPickerIter(GetWorld());
   LevelPickerIter; ++LevelPickerIter) {
      LevelPickerIter->ToggleDeleteActive();
    }
  }
}
```

Listing 7.16: Función que activa el modo borrado de diseños.

Dentro de la función ToggleDeleteActive() se activa el modo borrado de diseños, se muestra el panel que indica que el modo de borrado de diseños está activo y se actualiza el menú para habilitar el modo borrado en cada uno de los botones de este.

En la figura 7.32 se puede ver el mensaje que aparece en el menú principal al activar el modo borrar.

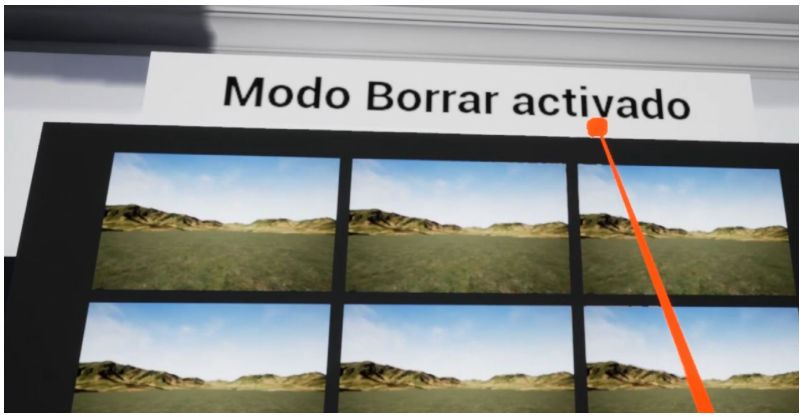


Figura 7.38: Mensaje de activación del modo borrar.

Una vez que se ha activado el modo borrado de diseños, cuando el usuario pulse sobre uno de los botones se eliminará el archivo y la imagen correspondiente al diseño guardado, se eliminará el diseño del índice de diseños guardados, y se actualizará el menú principal.

Tras realizar estos pasos se habrá eliminado el diseño guardado y todos sus ficheros relacionados.

7.9. Flujo de creación de un edificio

Tras haber explicado cómo funcionan los distintos componentes que permiten a un usuario crear el diseño de un edificio, se explicarán los pasos que el usuario debe seguir para realizar el diseño de un edificio de principio a fin.

7.9.1. Crear el nuevo diseño

Al iniciar la aplicación se le presenta al usuario el menú principal, donde puede gestionar todos los diseños creados y crear nuevos diseños.

Para crear un nuevo diseño el usuario debe pulsar el botón de crear un diseño, que se encuentra en la barra de acción. Tras pulsar el botón se añade un nuevo diseño al menú principal.

(a) Botón de añadir un nuevo diseño. (b) Diseño añadido al menú principal.

Figura 7.39: Ejemplo de creación de un nuevo diseño.

Para comenzar a editar el diseño creado el usuario debe seleccionarlo en el menú principal. Una vez lo selecciona se carga el entorno de diseño de edificios vacío, para que el usuario pueda comenzar a diseñar el edificio.

Figura 7.40: Entorno de diseño de edificios.

7.9.2. Colocar las estructuras

Al acceder al entorno de diseño de edificios se coloca al usuario de forma predeterminada en el modo diseño, por lo que si desea colocar estructuras debe cambiar al modo creación. Para cambiar al modo creación debe pulsar el botón de cambio de modo que se encuentra en el menú del controlador izquierdo.

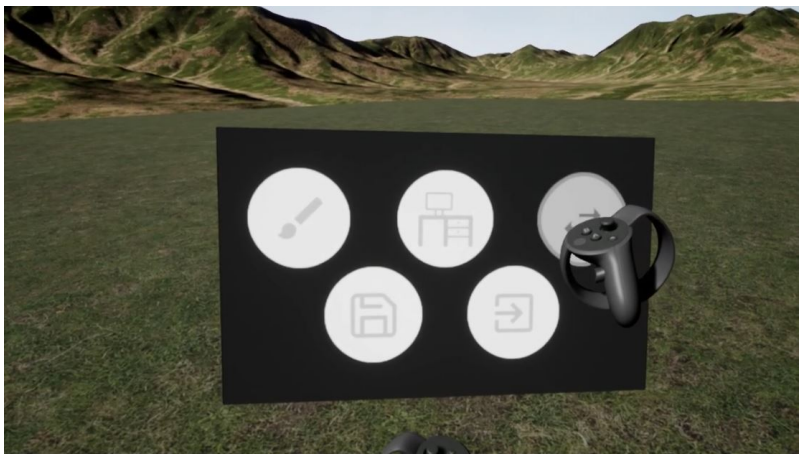
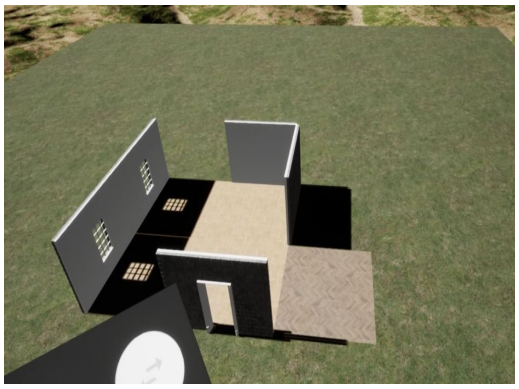


Figura 7.41: Botón de cambio de modo.

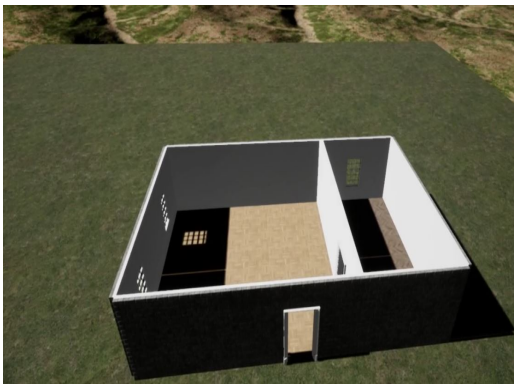
Una vez que el usuario se encuentra en el modo creación puede comenzar a colocar las estructuras del edificio. Para ello debe seleccionarlas en el menú de selección y colocarlas en el lugar que desee.

Figura 7.42: Menú de selección de estructuras.

En la figura 7.43 se puede ver un ejemplo de diseño de estructuras, en el que se han creado dos habitaciones. En la figura 7.43a se observa la vista que tiene el usuario durante la creación de una estructura, y el la figura 7.43b se puede apreciar el diseño final que ha creado el usuario, con los techos ocultos para que se pueda distinguir mejor las dos habitaciones creadas.



(a) Estructura a medio hacer.



(b) Estructura terminada.

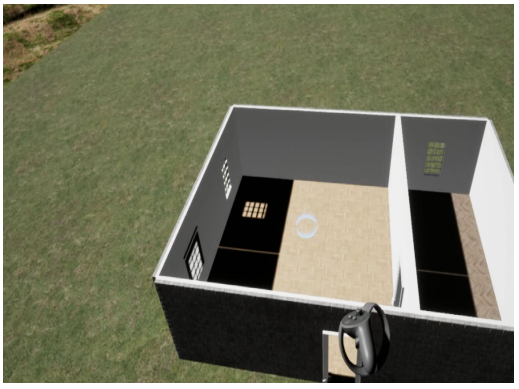
Figura 7.43: Ejemplo de diseño de la estructura de un edificio.

7.9.3. Colocar los objetos

Cuando el usuario esté satisfecho con la estructura creada puede comenzar a colocar los muebles y objetos que componen el edificio. En este caso se ha decido crear una sala de estar en la habitación de mayor tamaño y un baño en la habitación de menor tamaño.

Para comenzar, el usuario debe cambiar al modo diseño pulsando el botón de cambio de modo que se encuentra en el menú del controlador izquierdo. Una vez que ha pulsado el botón debe seleccionar el lugar del terreno en el que desea colocarse. En la figura 7.41 se puede ver un ejemplo de cambio al modo diseño.

(a) Botón de cambio de modo.



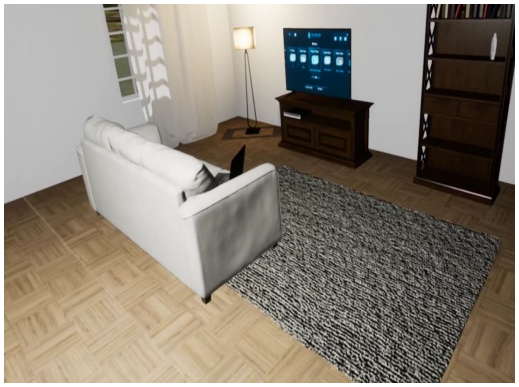
(b) Selección del punto de colocación.

Figura 7.44: Cambio al modo diseño.

Una vez que el usuario se encuentra en el modo diseño puede comenzar a colocar los objetos de cada habitación. Para ello debe seleccionarlos en el menú de selección y colocarlos en el lugar que desee.

Figura 7.45: Menú de selección de objetos.

En la figura 7.46 se puede ver dos ejemplos de colocación de objetos en el edificio. En la figura 7.46a se puede observar como se coloca una alfombra en el suelo de la sala de estar, y en la figura 7.46b se puede ver como se pone un dispensador de jabón sobre un lavamanos en el baño.

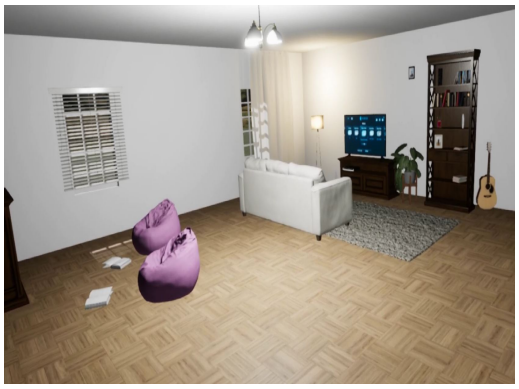


(a) Colocación de un objeto en la sala de estar.

(b) Colocación de un objeto en el baño.

Figura 7.46: Ejemplo de diseño de las habitaciones de un edificio.

En la figura 7.47 se puede observar el aspecto de cada habitación en el modo diseño tras colocar todos los objetos en la sala de estar y el baño.



(a) Sala de estar.

(b) Baño.

Figura 7.47: Aspecto de las habitaciones tras colocar los objetos.

Por otro lado, se puede ver en la figura 7.48 el aspecto del edificio finalizado en el modo creación, con los techos ocultos para que se puedan ver los objetos que componen cada habitación.

Figura 7.48: Aspecto final del edificio en el modo creación.

Capítulo 8

Conclusiones

A continuación se comentarán los resultados obtenidos tras la realización del proyecto, el grado de satisfacción con el resultado final y las posibles ampliaciones que se podrían realizar al proyecto en un futuro.

8.1. Resultados del trabajo

La realización de este proyecto ha resultado un gran reto personal. Ya que no tenía mucha experiencia en el desarrollo de aplicaciones en C++, y nunca había realizado un proyecto de esta envergadura con Unreal Engine, además de que el proyecto tenía el añadido de trabajar con realidad virtual.

Respecto a la aplicación desarrollada, se han logrado alcanzar los objetivos propuestos al principio del desarrollo del proyecto. Y ha servido para aprender una gran cantidad de conceptos de Unreal Engine, como pueden ser:

- El diseño y creación de interfaces de usuario dentro de Unreal Engine.
- La adaptación de los controles a un sistema de realidad virtual, y la gestión de los eventos producidos por los controladores de movimiento.
- El sistema de Blueprints, que permite gestionar los eventos producidos y sirve como interfaz entre una clase C++ y los Blueprints que heredan de ella.
- El sistema de colisión, que permite modificar y controlar la respuesta de un actor ante la colisión con otros actores.
- El sistema de guardado de diseños, y el manejo de ficheros mediante Unreal Engine.

Finalmente, este proyecto ha sido de mucha utilidad a la hora de experimentar con el motor Unreal Engine. Y ha servido para obtener los conocimientos necesarios para el desarrollo de aplicaciones con Unreal Engine, por lo que puedo afrontar proyectos más complejos y continuar experimentando con el motor con seguridad.

8.2. Trabajos futuros

Aunque la aplicación ha cumplido con los objetivos establecidos, hay una serie de características que no se han podido añadir debido a su complejidad y la falta de tiempo. Algunas de esas características son:

- Añadir múltiples plantas a un edificio, permitiendo al usuario cambiar de planta.
- Permitir al usuario importar sus propias estructuras y objetos a la aplicación.
- Ampliar las interfaces creadas, permitiendo al usuario cambiar el color y la forma de los bocetos que crea, y modificar el color de las estructuras y objetos que se encuentran en el entorno.

Bibliografía

[1] Iris VR. *Prospect Pro*. https://irisvr.com/prospect.

[2] Real State Innovation. *REinVR*. https://www.reinvr.com.

[3] Google. *Tilt Brush*. https://www.tiltbrush.com/.

[4] RAE. *Definición de realidad virtual*. https://dle.rae.es/srv/fetch?id=VH7cofQ.

[5] Mundo virtual. *Definición de realidad virtual*. http://mundo-virtual.com/que-es-la-realidad-virtual/.

[6] Proyecto Idis. *Sensorama*. https://proyectoidis.org/sensorama/.

[7] Retro Maquinitas. *Virtual Boy*. https://retromaquinitas.com/virtual-boy/.

[8] Palmer Luckey. *Kickstarter de Oculus Rift*. https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game.

[9] Oculus. *Página de Oculus Quest*. https://www.oculus.com/quest/.

[10] Valve. *Página de Valve Index*. https://www.valvesoftware.com/es/index.

[11] Marcos Díez. *Motor gráfico... ¿y eso qué es?* https://marcosdiez.wordpress.com/2012/11/14/motor-grafico-y-eso-que-es/.

[12] Unity Technologies. *Acerca de Unity*. https://unity3d.com/es/public-relations.

[13] Unity Technologies. *Unity AR and VR games*. https://unity.com/es/solutions/ar-and-vr-games.

[14] Crytek. *Presroom - CryEngine*. https://press.cryengine.com/.

[15] Epic Games. *Unreal Engine for AR, VR & MR*. https://www.unrealengine.com/en-US/vr.

[16] Universidad de Las Palmas de Gran Canaria. *Objetivos y competencias del GII*. https://www2.ulpgc.es/archivos/plan_estudios/4008_40/ObjetivosyCompetenciasdelGII.pdf.

[17] Google. *Daydream Sticker Sheet*. https://developers.google.com/vr/design/sticker-sheet.

[18] Epic Games. *Unreal Engine End User License Agreement*. https://www.unrealengine.com/en-US/eula.

[19] Epic Games. *Royality-free revenue sources FAQ*. https://www.unrealengine.com/en-US/faq.

[20] Epic Games. *MARKETPLACE FREQUENTLY ASKED QUESTIONS (FAQ).* https://www.unrealengine.com/en-US/marketplace-faq.

[21] Material Design. *Icons.* https://material.io/resources/icons/?search=bed&style=baseline.

[22] Apache Software Fundation. *Apache License, Version 2.0.* https://www.apache.org/licenses/LICENSE-2.0.html.

[23] Flat Icon. *Attribution: How, when and where?* https://support.flaticon.com/hc/en-us/articles/207248209-How-I-must-insert-the-attribution-.

[24] Crative Commons. *Attributing Sources.* https://creativecommons.org/use-remix/.

[25] Mejores gafas realidad virtual. *Oculus Rift.* https://mejoresgafasrealidadvirtual.com/gafas-vr/gafas-de-realidad-virtual-para-pc/oculus-rift/.

[26] Wikipedia. *Microsoft Visual Studio.* https://es.wikipedia.org/wiki/Microsoft_Visual_Studio.

[27] Git. *Git - fast, scalable, distributed revision control system.* https://github.com/git/git.

[28] Wikipedia. *Github.* https://es.wikipedia.org/wiki/GitHub.

[29] Wikipedia. *Trello.* https://es.wikipedia.org/wiki/Trello.

[30] Blanchard Space. *Introducción a C++, ¿qué es?* https://blanchardspace.wordpress.com/2013/05/06/introduccion-a-c-que-es/.

[31] Baboonlab. *Blueprints en Unreal Engine 4, funciones y tipos.* http://www.baboonlab.com/blog/noticias-de-marketing-inmobiliario-y-tecnologia-1/post/blueprints-en-unreal-engine-4-funciones-tipos-y-otras-caracteristicas-21.

[32] Ken Schwaber y Jeff Sutherland. *La guía de SCRUM.* https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf.

[33] Unreal Engine Documentation. *Actors.* https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Actors/index.html.

[34] Unreal Engine Documentation. *Pawn.* https://docs.unrealengine.com/en-US/Gameplay/Framework/Pawn/index.html.

[35] Unreal Engine Documentation. *Blueprints.* https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html.

[36] Unreal Engine Documentation. *UMG UI Designer.* https://docs.unrealengine.com/en-US/Engine/UMG/index.html.