Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

BIONICAPE
GAME STUDIOS

MÁSTER UNIVERSITARIO EN DESARROLLO DE VIDEOJUEGOS

Trabajo Fin de Máster

“Sistemas de portales multijugador con Unreal Engine.”

“Multiplayer portal systems with Unreal Engine."

Autor: Francisco Moreno Jiménez

Badalona, 17 de mayo de 2023

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43.826.207-Y profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Sistemas de portales multijugador con Unreal Engine”

ha sido realizada bajo su dirección por D. **Francisco Moreno Jiménez**, con N.I.F. 48.132.774-F, en la entidad colaboradora **Bionic Ape, S.L**. situada en Calle Fina de Calderón 25 5B 28055, Madrid y han sido tutorizadas por D. **Javier Sevilla Sánchez**, Director en la Empresa **Bionic Ape, S.L.**, en calidad de tutor externo.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos, firman la presente en San Cristóbal de La Laguna a 17 de mayo de 2023.

Agradecimientos

A David Llorca,

por ayudarme a encontrar mi vocación al mostrarme el maravilloso mundo de la programación.

A Javi Agenjo,

por enseñarme a amar el desarrollo de videojuegos.

A Juan Solana,

por ayudarme a empezar a dedicarme profesionalmente al desarrollo de videojuegos.

A Javier Sevilla,

por ver algo en mí, por decidir darme la oportunidad de convertirme en un súper guerrero y por todo lo que has invertido en mí.

A Francisco Mesas,

por cambiar mi forma de ver el mundo y por enseñarme que solos avanzamos más deprisa, pero que acompañados llegamos más lejos.

A Míriam Suñé,

por todo el apoyo incondicional desinteresado que me has regalado diariamente durante tanto tiempo y por ser mi salvavidas en mis peores momentos.

A mis padres y familia,

por hacerlo todo posible.

Resumen

Unreal Engine es un motor diseñado y desarrollado por ***Epic Games***. Este motor se utiliza generalmente para el diseño y desarrollo de videojuegos, pero puede tener otras aplicaciones. Este motor está preparado para ser usado fácilmente en modo **multijugador**, es decir, que varios jugadores pueden estar en la misma partida y las acciones de uno tienen un impacto en la partida del otro. Aun con todas las facilidades que el motor otorga, es necesaria una buena comprensión para poder desarrollar sistemas complejos.

Unreal Engine está demostrando ser cada día más versátil, siendo usado en infinidad de sistemas y aplicaciones diversas. Por ello, es fundamental **trabajar** de una forma que **maximice la reutilización de nuestro trabajo**. Esto facilita no tener que resolver el mismo problema una y otra vez. Una forma de lograr esto es **dividir el código en *plugins* y módulos independientes y agnósticos** que puedan ser fácilmente migrados de un proyecto a otro.

Definimos un portal como un punto en el espacio que está conectado con otro punto en el espacio en una distinta ubicación, de tal manera que, cuando cruzamos uno de ellos, la parte de nosotros que lo ha cruzado ha viajado en el espacio hasta la posición del portal conectado. Además, un portal puede llegar a tener una representación visual, que sería lo que se vería si estuviésemos mirando a través del portal conectado, como si la luz también pudiera viajar a través de él.

El **objetivo** de este proyecto ha sido **la realización de un plugin que incorpore en cualquier proyecto la funcionalidad de portales con soporte de multijugador**. Este plugin debe poder ser fácilmente **migrado de un proyecto a otro y debe ser claro e intuitivo de utilizar**, en especial por diseñadores.

Palabras clave: Unreal Engine, multijugador, portal, plugin

Abstract

Unreal Engine is an engine designed and developed by **Epic Games**. This engine is used generally for the design and development of video games, however it can have other uses. This engine is prepared for being easily used in **multiplayer** mode, that is, two players can be in the same game and their actions have an impact on each other's game. However the system is much aiding for developers, one must have a very good understanding of it for developing a complex system.

Unreal Engine is proving to be more versatile each day, as it keeps getting used in many different systems and applications. For that, it is important to **work** in a way that **maximizes reusing our work**. That makes it possible to not have to solve the same problem over and over again. A way of achieving this is to **divide the code in independent and agnostic *plugins* and modules** that can be easily migrated from one project to another.

We define a portal as a point in space that is connected to another point in space in a different location in such a way that, if we cross one of them, the part of us that has crossed it has traveled in space to the position of the connected portal. Moreover, a portal may have a visual representation of what we would see if we were looking through the connected portal, as if the light could also travel through them.

The **goal** of this project was **the development of a plugin that provides in any project the functionality of portals with support for multiplayer**. This plugin must be easily **migrated from one project to another and must be clear and intuitive to use**, specially for designers.

Keywords: Unreal Engine, multiplayer, portal, plugin.

Índice

Índice de figuras

Capítulo 1
Introducción

A lo largo de este Trabajo de Fin de Máster se desarrollará un **proyecto real** en el entorno de un **estudio de videojuegos**, mediante la aplicación de los conocimientos adquiridos durante el máster, en especial los relativos a diseño, producción, patrones de diseño, 3D y videojuegos multijugador. Aunque todas las asignaturas cursadas han aportado un granito de arena en la elaboración de este proyecto, las dos más representativas serían **Desarrollo de Videojuegos 3D** y **Temas Avanzados de la Tecnología de los Videojuegos**.

La existencia de portales es una característica **fácilmente incorporable a cualquier videojuego**, lo que puede dar pie a mecánicas de videojuegos muy interesantes, especialmente en juegos de puzzles.

En este proyecto, elaboraremos un plugin para Unreal Engine que permitirá añadir portales con soporte de multijugador en cualquier proyecto.

1.1. Antecedentes

La entidad colaboradora, **Bionic Ape S.L.**, es una empresa dedicada al outsourcing que nació cuando su creador, Javier Sevilla, que había estado trabajando como autónomo en proyectos de outsourcing, decidió formar una empresa y contratar a gente con la que repartir el trabajo.

Esta empresa tiene la filosofía de **dividir todo el código desarrollado en plugins y conservar la propiedad intelectual del mismo**, y así poder **reutilizarlo con futuros clientes**, optimizando el tiempo invertido en cada uno de ellos.

Uno de los plugins realizados por la empresa como desarrollo propio, que fue creado con la intención de ser publicado en el marketplace de Unreal Engine, implementa funciones de portales en multijugador. Sin embargo, debido a la necesidad de trabajar en proyectos externos para poder subsistir, no se había podido dedicar suficiente tiempo en pulir el plugin lo suficiente como para poder hacerlo público.

1.2. Periodo y Departamentos

El estudiante **trabaja a tiempo parcial** en la entidad colaboradora desde **agosto del 2022**. Durante el mes de **febrero de 2023**, se le asignó como tarea la **revisión y mejora de un plugin de portales con soporte multijugador** que se encontraba en estado preliminar, así como la elaboración de un nivel de ejemplo

introductorio que explicase el uso del plugin. La entidad colaboradora cuenta con menos de 10 trabajadores en total, por lo que por ahora no se encuentra dividida en departamentos.

1.3. Objetivo General

El proyecto de este Trabajo de Fin de Máster consiste en la elaboración de la tarea asignada descrita en el apartado anterior, **aplicando los conocimientos adquiridos durante el paso del estudiante por el máster** para lograr un producto final reutilizable, fácilmente migrable, desacoplado, claro e intuitivo de usar.

1.4. Objetivos Específicos

Concretamente, los objetivos del presente Trabajo de Fin de Máster se podrían dividir en los siguientes:

- **Valoración del estado actual** del proyecto.
- **Diseño y producción** del trabajo a realizar.
- **Desarrollo del trabajo**.
- **Redacción** de la presente **memoria** del proyecto.
- Valoración de **líneas de desarrollo futuras**.

Capítulo 2
Portales

Antes de entrar en materia, debemos **definir** uno de los conceptos más relevantes que aparecen a lo largo de este Trabajo de Fin de Máster: **¿Qué es un portal?** A lo largo de este apartado, discutiremos sobre esta cuestión y lo ilustraremos con ejemplos representativos de portales en diferentes ámbitos.

2.1. Definición

Si buscamos en la RAE la palabra *portal* [1], no encontraremos una definición que se adecúe exactamente a lo que buscamos.

Definimos un portal como un **punto en el espacio que está conectado con otro punto en el espacio** en una distinta ubicación, de tal manera que, cuando cruzamos uno de ellos, la parte de nosotros que lo ha cruzado ha viajado en el espacio hasta la posición del portal conectado. Además, **un portal puede llegar a tener una representación visual**, que sería lo que se vería si estuviésemos mirando a través del portal conectado, **como si la luz también pudiera viajar a través de él**.

2.2. Ejemplos

Podemos encontrar ejemplos del concepto descrito en el apartado anterior tanto en videojuegos como en televisión como en cine. En este apartado, expondremos y analizaremos algunos de estos ejemplos.

2.2.1. Portales en videojuegos

El primero de los ejemplos de portales en videojuegos que analizaremos es del archiconocido **Pacman**. Si nos fijamos en el mapa de la *Figura 1*, veremos que **en la parte central de las paredes izquierda y derecha hay una especie de pasillo sin final**. Si alguna vez hemos jugado a este videojuego y hemos tratado de cruzarlos, sabremos que, **al cruzar uno de ellos,** nuestro personaje **se teletransporta al pasillo opuesto**. En otras palabras, al final de cada uno de los pasillos hay un portal y ambos están conectados.

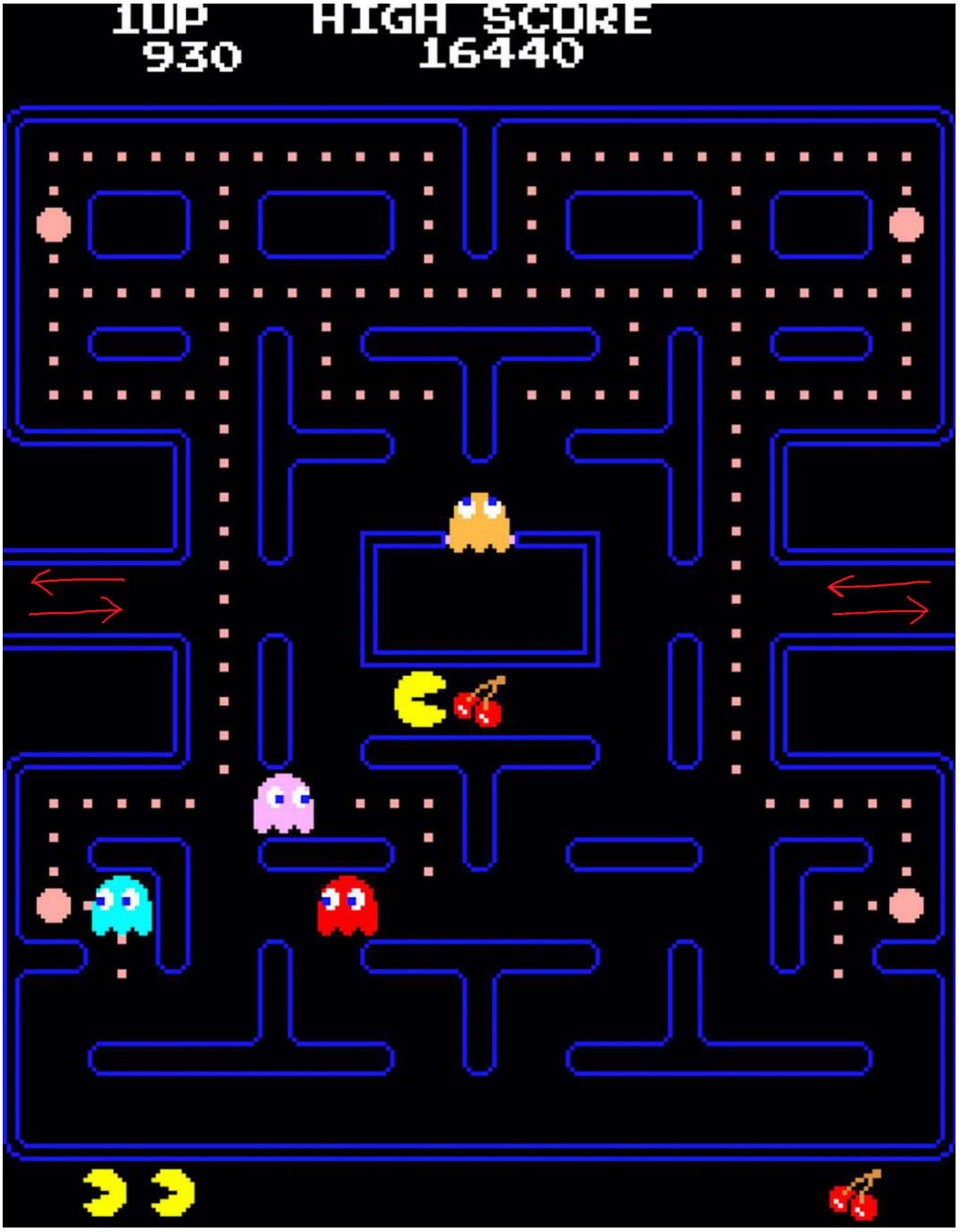

Figura 1: Mapa del videojuego Pacman. [2]

Otro de los ejemplos de videojuegos con lo que podríamos considerar como portales es **Snake**. Si alguna vez lo hemos jugado, sabremos que, en ocasiones, cuando la serpiente alcanza frontalmente uno de los bordes de la pantalla y sigue avanzando sin cambiar de dirección, aunque en algunas variantes del juego hay un muro y la serpiente morirá, en otras, como podemos ver en la *Figura 2*, **su cabeza aparecerá en la pared opuesta** y, a medida que siga avanzando, su cuerpo eventualmente lo hará también. Así, podemos considerar que **cada uno de los límites de la pantalla es un portal que está conectado con el de la pared opuesta**.

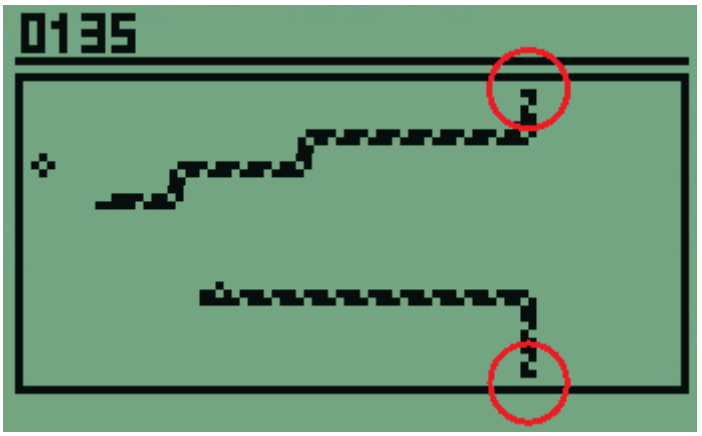

Figura 2: Partida de Snake en la que se ha cruzado un portal. [3]

El último de los ejemplos de portales en videojuegos, tal vez **el más representativo** y el que podríamos considerar el mayor modelo a seguir **en este proyecto** es el videojuego (o la saga) **Portal**. A diferencia de en los ejemplos anteriores, en este caso los portales sí tienen una representación visual de lo que se vería a través del portal conectado, como podemos ver en la *Figura 3*.

Figura 3: Ejemplo de portales en el videojuego Portal. [4]

Un ejemplo de videojuego con portales que tienen una representación visual, pero esta no guarda relación con el portal conectado lo podemos encontrar en **Minecraft**. En este videojuego es posible construir un portal, el *Portal de Nether*, que **es capaz de transportar al jugador** entre la *Superficie* y el *Nether*. Como podemos ver en la *Figura 4*, dicho portal muestra una especie de **efecto visual**, pero **no muestra información sobre el portal de destino**.

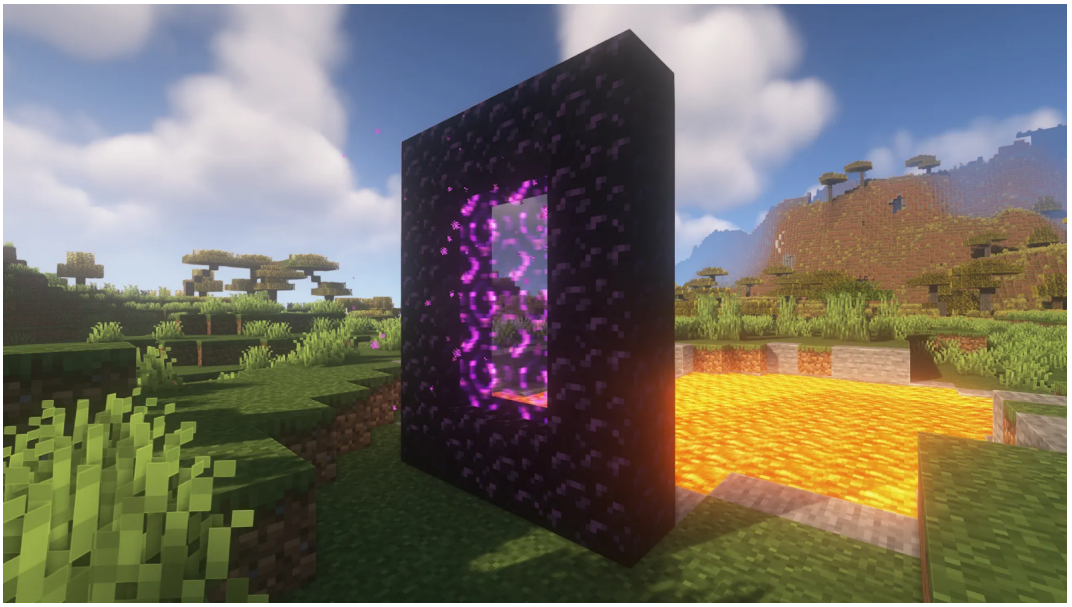

Figura 4: Portal de Nether en el videojuego Minecraft. [5]

2.2.2. Portales en televisión

En **televisión** también es posible encontrar **ejemplos** de portales. El primero que veremos pertenece a la serie **Dragon Ball Z**, durante la saga de **Majin Buu**. En ciertos capítulos, los personajes de Piccolo, Goten y Trunks se encierran en una dimensión paralela accesible desde la ***Sala del Tiempo Hiperbólico*** junto con Super Buu con el objetivo de derrotarlo sin que la Tierra sufra graves consecuencias. Ante la desesperanza por ganar la batalla, Piccolo decide hacer el sacrificio de **destruir la sala que conecta las dos dimensiones** para que ninguno de ellos pueda salir y así mantener encerrado el mal que alberga Super Buu. Ante esta situación, el antagonista, de alguna forma, **abre** con su boca una especie de brecha o **portal que une ambas dimensiones**, que podemos ver en la *Figura 5*, y así poder regresar a la dimensión de origen, dejando a Goten, Trunks y Piccolo encerrado en la dimensión paralela [6].

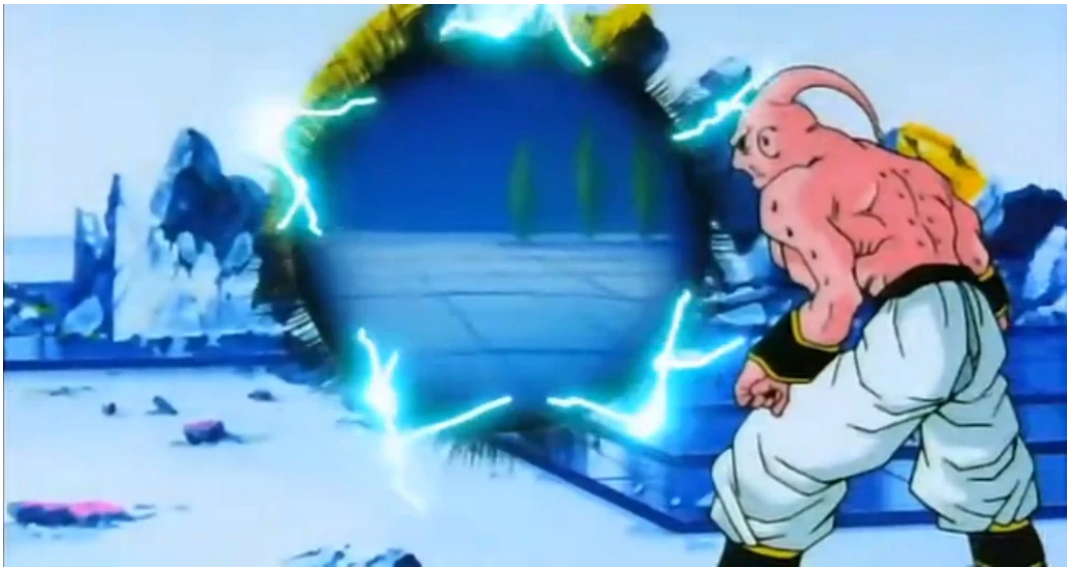

Figura 5: Portal abierto por Super Buu en Dragon Ball Z. [7]

Otro ejemplo de **portales en televisión**, y uno de los más icónicos, lo podemos encontrar en la serie **Rick y Morty**. En esta serie, Rick, un científico chiflado, inventa una **pistola de portales**, con la que va de aventuras con su nieto Morty por cantidad de **dimensiones extrañas**. En la *Figura 6* podemos ver uno de los portales, que **no tiene representación visual** de su portal conexo.

Figura 6: Portal en la serie Rick y Morty. [8]

2.2.3. Portales en cine

Así como hemos podido encontrar ejemplos de portales tanto en videojuegos como en televisión, **en cine también es posible encontrarlos**. En la adaptación cinematográfica que se ha hecho de los cómics de Marvel se encuentra, entre muchos otros, el personaje de **Dr. Strange**. Uno de los poderes de este personaje, como podemos ver en la *Figura 7*, que corresponde a una escena de lucha entre Dr. Strange y Spider-Man extraída de la película *Spider-Man: No Way Home*, es el de crear portales. **Estos portales**, como en *Portal* y en *Dragon Ball Z*, **tienen una representación visual del portal con el que están conectados**. En esta figura, además, podemos ver un concepto muy importante en este proyecto, que es el de **recursividad de portales**. Este efecto se produce cuando **desde un portal se ve ese mismo portal a través del portal con el que está conectado**, creando un efecto de espejo infinito. Más adelante hablaremos más detalladamente de este concepto.

Figura 7: Portales recursivos en Spider-Man: No Way Home. [9]

Otro de los ejemplos que podemos ver de portales en el mundo del cine pertenece a la **película *Suzume***. En esta película, los protagonistas se encargan de viajar por Japón cerrando "**puertas**" que en realidad son **portales a otra dimensión** en la que habita un gran mal, al que llaman gusano, que, de lograr su objetivo de cruzar uno de los portales, causaría estragos en la Tierra. Como podemos ver en la Figura 8, en la misma portada de la película aparecen los protagonistas cruzando uno de estos portales.

Figura 8: Suzume y Sōta cruzando un portal en Suzume. [10]

Capítulo 3
Multijugador

En este capítulo, hablaremos de otro de los conceptos más importantes en este proyecto: los **videojuegos multijugador**. Comentaremos qué son y sus tipos, hablaremos de su historia y analizaremos qué hay que tener en cuenta a la hora de trabajar con juegos multijugador en Unreal Engine.

3.1. Videojuegos multijugador

Si buscamos en la RAE la palabra *multijugador* [11], no encontraremos resultados, por lo que intentaremos definirla nosotros. Se dice que un videojuego es **multijugador** cuando **permite la interacción, simultánea o no, de más de un jugador en una misma partida**. En el siguiente apartado, veremos los diferentes tipos de videojuegos multijugador que podemos encontrar.

3.2. Historia

Algunos de los primeros videojuegos multijugador en la historia incluyen *Tennis For Two* (1958), *Pong* (1972), *Spacewar!* (1962) o *Astro Race* (1973) [12]. Los primeros videojuegos multijugador en tiempo real, como *Empire* (1973) o *Spasim* (1974) fueron desarrollados para el sistema *PLATO*.

Más adelante, durante la década de 1980, los videojuegos multijugador se popularizaron gracias al gran éxito de las máquinas arcade. Como ejemplos de esta época podemos destacar *Gauntlet* (1985) o *Quartet* (1986).

El primer videojuego conocido con multijugador LAN es *Spectre* (1991), para Apple Macintosh, que soportaba hasta 8 jugadores simultáneos.

Durante la década de 1990, con la llegada de internet, empezaron a abrirse paso los primeros videojuegos multijugador en línea. Es el caso de juegos como la saga de *Doom* (1993), *Duke Nukem* (1996) o *Quake* (1996).

3.3. Tipos de videojuegos multijugador

Podemos encontrar diferentes tipos de juegos multijugador según los dispositivos en los que se estén jugando:

- **Multijugador local**: Todos los jugadores que están participando lo están haciendo desde el mismo dispositivo, con diferentes dispositivos de entrada o, en ocasiones, el mismo.

- **Multijugador en red de acceso local**: Todos los jugadores que están participando lo están haciendo cada uno desde su propio dispositivo, y estos están conectados entre sí a través de la red de acceso local, o LAN (Local Access Network). Por norma general, todos los jugadores deberán poseer una copia del juego para poder jugar. En otras ocasiones, solo uno de los jugadores deberá disponer una copia del juego y podrá invitar a jugar a otros jugadores que no la tengan. Un ejemplo de esto es la funcionalidad *DS Download Play* de la que disponen los dispositivos *Nintendo DS*, que permite jugar a videojuegos como *Mario Kart* o *New Super Mario Bros* solo con que uno de los jugadores disponga de los juegos.
- **Multijugador en línea**: Es un tipo de multijugador similar al anterior, pero en este caso los jugadores podrán conectarse entre sí desde cualquier parte del mundo a través de internet. En algunas consolas, para poder acceder a esta característica, será necesario pagar por una suscripción temporal. Es el caso de consolas como *Nintendo Switch* o *PlayStation 4*. En este caso también es posible que no todos los jugadores deban haber comprado una copia del juego para poder jugar. Es el caso de juegos como *A Way Out*. Este juego dispone de una versión de pago y una versión gratuita. Sin embargo, para que un jugador pueda jugar con la versión gratuita, es necesario que reciba una invitación de algún jugador que haya adquirido la versión de pago y así poder jugar juntos.
- **Multijugador por correo electrónico**: En esta modalidad de juego multijugador, propia de los juegos multijugador por turnos, cada jugador juega durante su turno la partida localmente en su máquina, para posteriormente sincronizar el nuevo estado de la partida con sus contrincantes a través de correo electrónico. En la actualidad, se podría usar cualquier método de mensajería instantánea para jugar en este modo, no necesariamente con correo electrónico, que es como surgió. En la *Figura 9*, podemos ver como ejemplo una partida del clásico *Tres en Raya* jugada a través de *Whatsapp*.

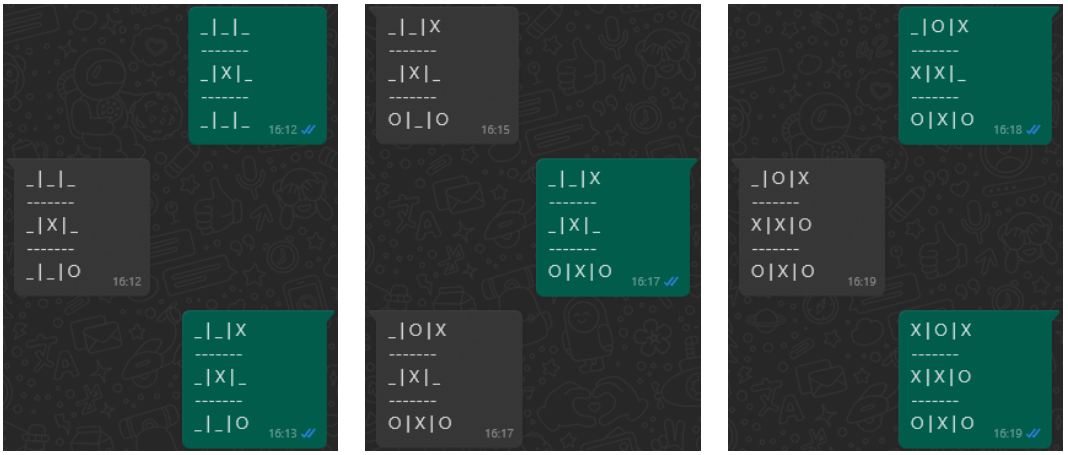

Figura 9: Partida de Tres en Raya a través de Whatsapp.

Por otra parte, también podemos clasificar los **videojuegos multijugador**, y de hecho cualquier juego multijugador ya sea analógico o digital, según cómo se organicen los equipos en cada partida:

- **Todos contra todos**: En esta modalidad no se forman equipos, sino que todos los jugadores compiten contra todos los demás.
- **Cooperativo**: Todos los jugadores cooperan por un objetivo común.
- **Por equipos**: Esta categoría podría considerarse una mezcla de las dos anteriores. Los jugadores se dividen en diferentes equipos de igual o distinto tamaño. Los jugadores de cada equipo cooperan con los de su mismo equipo para competir contra los demás equipos.

Finalmente, podemos clasificar también los juegos multijugador según la dinámica de la partida:

- **Multijugador en tiempo real**: Todos los jugadores de la partida pueden efectuar sus acciones en el juego de manera simultánea.
- **Multijugador por turnos**: Cada jugador en la partida dispone de un turno de tiempo limitado durante el cual solo él y nadie más puede efectuar acciones en la partida. Cada vez que todos los jugadores han terminado su turno, se vuelve a iniciar el ciclo completo hasta que la partida concluye.

3.4. Red

Antes de entrar en profundidad en temas de red, debemos definir ciertos conceptos básicos que es primordial conocer de antemano:

- **Cliente**: En una conexión de red, a cada uno de los usuarios que se conectan desde su máquina a internet se les llama clientes. En realidad, el "cliente" no es la persona, sino el software que se está ejecutando en su máquina.
- **Servidor**: Aunque a priori pueda sonar como un concepto relativamente abstracto, un servidor no es más que un ordenador que está conectado a internet y que está abierto a recibir conexiones entrantes con tal de compartir información.

En sistemas en los que hay múltiples clientes que deben poder comunicarse entre sí, hay dos formas como pueden establecer dicha comunicación:

- **Cliente-Servidor**: En este tipo de comunicación, el servidor actúa como intermediario entre todos los clientes, por lo que toda la comunicación entre ellos pasa por él. Los clientes le piden al servidor que envíe un mensaje a otro cliente y él lo hace. En este caso, **los clientes dependen de que el servidor esté activo** y funcionando para poder comunicarse con el resto de clientes.

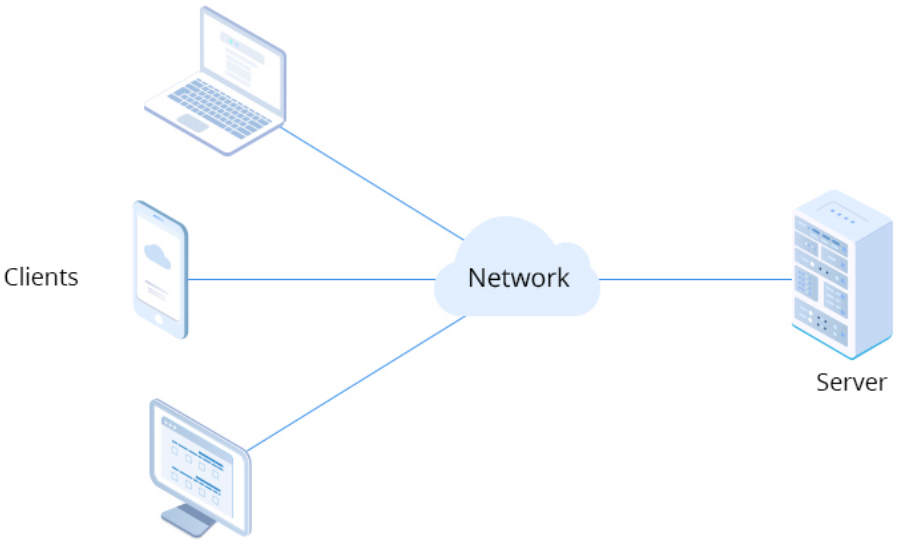

Figura 10: Comunicación Cliente-Servidor. [13]

- **Peer-to-peer (P2P)**: En las comunicaciones P2P no es necesaria la existencia de un servidor, ya que **los clientes se comunican directamente entre sí**. Al ser un sistema más descentralizado, la comunicación es más desordenada. Además, este tipo de comunicación es más inseguro porque

cada cliente debe conocer a los demás. Sin embargo, la conexión es más confiable porque no depende de la conexión constante de un servidor.

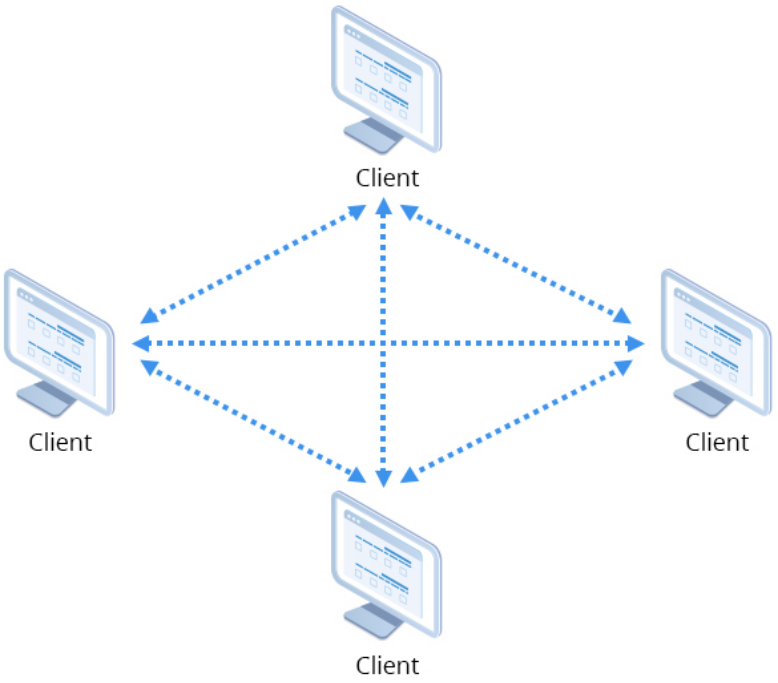

Figura 11: Comunicación Peer-to-peer. [13]

3.5. Multijugador en Unreal Engine

Unreal Engine es un motor de videojuegos desarrollado por **Epic Games** del que hablaremos en más detalle más adelante. Este motor de videojuegos está diseñado desde sus fundamentos para **soportar características de multijugador online de manera nativa**. Por ello, hacer cosas sencillas como generar un efecto desde una máquina y que se manifieste en todos los clientes es algo relativamente asequible de realizar. No obstante, para desarrollar sistemas más complejos que dispongan de características multijugador, es primordial tener una comprensión adecuada de cómo funciona el sistema multijugador nativo de Unreal Engine y saber ponerlo en práctica.

A lo largo de este apartado analizaremos todos los aspectos básicos que deberemos tener en cuenta a lo largo de este proyecto en relación a las características **multijugador en Unreal Engine**.

3.5.1. Cliente y Servidor

La comunicación multijugador online que se produce con Unreal Engine sigue siempre una estructura Cliente-Servidor.

Dentro de una partida, el servidor actúa como autoridad y, como tal, se encarga de ejecutar el juego en su propia instancia y compartir con los clientes las actualizaciones en el estado del juego. Por defecto, el servidor posee a prácticamente todos los actores en la escena y solo él puede actualizar su estado. Todo lo que ocurra en el estado del juego del servidor se transmitirá a los clientes, siempre y cuando lo hayamos marcado como replicable.

Por otro lado, los clientes se limitan a recibir actualizaciones del servidor y a comunicarse con él cada vez que quieran cambiar el estado del juego a través de una acción del jugador. Aparte, opcionalmente el cliente puede encargarse de la **representación visual y cosmética del estado de la partida**. Como esta representación visual es algo propio de cada jugador, esta se gestiona normalmente a través del HUD, una interfaz de usuario, que **solo existe** en la instancia de **cada cliente** y el servidor no sabe nada de todo esto. Es a través de estos como el cliente debe comunicarse con el servidor. En ocasiones, como veremos más adelante, un cliente puede actuar a la vez como servidor. Todo lo que ocurra en el estado del juego del cliente se quedará en ese cliente a no ser que este se lo comunique al servidor, por lo que, en general, solo ejecutaremos en cliente efectos visuales de los que solo deba ser conocedor el propio cliente.

Desde el mismo **editor de Unreal Engine** descargado desde el lanzador de Epic Games, **es posible empaquetar el juego**, es decir, generar el fichero ejecutable, aunque este fichero ejecutable solo puede ser ejecutado por clientes. Para empaquetar un juego que sea apto para ser ejecutado por un **servidor dedicado**, es necesario seguir los siguientes pasos:

1. Crear una cuenta de Github, en caso de que no tengamos una.
2. Vincular nuestra cuenta de Epic Games con nuestra cuenta de Github.
3. En ese momento, recibiremos una invitación de Epic Games para unirnos a su comunidad de Github, lo cual es necesario para realizar el paso siguiente.
4. Acceder al repositorio de Github de Unreal Engine y descargar la versión *Release* que se adecúe con la versión de Unreal Engine que tengamos instalada desde el lanzador de Epic Games.
5. Seguir los pasos descritos en [14] para compilar el código fuente del motor que acabamos de descargar.

6. Desde el motor que acabamos de descargar e instalar desde Github será posible empaquetar nuestro juego para servidor.

Si no quisiéramos tener un servidor dedicado o tener que empaquetar la versión de Github, nos valdría con ejecutar un **cliente** en modo "Escucha", es decir, como **Host de la partida**. Para que otros clientes se pudiesen conectar, deberíamos conocer la IP y el puerto de dicho Servidor o utilizar las herramientas que proveen sistemas Online como el de Epic o el de Steam, que harán esta resolución de IP y puerto por nosotros.

3.5.2. Tipo de conexión

Cuando vayamos a ejecutar un juego desde el editor de Unreal Engine en modo de red, veremos que hay 3 modos diferentes entre los que deberemos escoger:

- **Play Standalone**: En este modo **no hay modo multijugador**. El jugador juega en solitario una partida del juego y no hay ningún tipo de conexión con otras instancias del mismo. Las acciones de cada jugador solo tendrán un efecto en su propia instancia. Es el modo por defecto para los juegos para un solo jugador o *single player*.
- **Play as Listen Server**: En este modo, **uno de los clientes actúa también como servidor**, por lo que el resto de clientes en la partida se comunican a través de este.
- **Play as Client (Dedicated Server)**: En este modo, **el servidor es una instancia que se ejecuta por separado** de la de los clientes para que estos se conecten con la misma. Este es el modo que simula el servidor dedicado que podemos crear con la versión del código de Git.

Capítulo 4
Alcance y desarrollo

Una vez presentado todo el contexto que da pie a nuestro proyecto, en este capítulo **expondremos el propio proyecto** en sí: cada una de las tareas que se han llevado a cabo, cómo se han organizado y cómo se ha llevado a cabo toda la parte de desarrollo.

4.1. Tareas realizadas

En este apartado, describiremos las tareas que fueron realizadas a lo largo de este proyecto, así como las metodologías de trabajo y un cronograma con los distintos periodos dedicados a cada una de las tareas.

4.1.1. Descripción

En consonancia con los objetivos específicos de este proyecto, se han realizado las siguientes tareas:

- **Revisar el estado actual del plugin**: se hizo una revisión del estado actual del plugin de portales en multijugador.
- **Diseñar las mejoras a implementar para publicar el plugin**: una vez hecha la valoración del estado actual del plugin, se diseñaron posibles arreglos y mejoras a partir de lo que estaba ya hecho para dejarlo listo para ser publicado en el marketplace de Unreal Engine.
- **Programar el desarrollo de las mejoras propuestas**: una vez hecho el diseño, se hizo una programación del tiempo que se dedicaría a cada uno de los arreglos y mejoras propuestos.
- **Desarrollar las mejoras propuestas**: tras diseñar y planificar el desarrollo a realizar, se procedió al mismo.
- **Elaborar nivel explicativo del plugin**: esta tarea fue realizada paralelamente a la anterior. A la vez que se desarrollaban los arreglos y mejoras propuestos, se iba trabajando en un nivel de ejemplo para Unreal Engine que hiciese una introducción al plugin y mostrase casos de uso.
- **Redactar la memoria del proyecto**: una vez se dio por finalizada la fase de desarrollo, dio comienzo la redacción de la memoria.
- **Valorar las líneas de desarrollo futuras**: como parte de la redacción de la memoria y una vez documentada la fase de desarrollo del proyecto, se procedió a la valoración de posibles líneas de desarrollo futuras.
- **Preparar y realizar la defensa del TFM**: una vez redactada y publicada la memoria, se dio paso a la preparación y posterior defensa de la misma.

4.1.2. Metodología

A lo largo del proyecto, se ha usado la metodología ágil. La metodología ágil es un enfoque de desarrollo de software que se basa en la entrega temprana y continua de software funcional. La metodología ágil se enfoca en la colaboración y la comunicación entre los miembros del equipo de desarrollo y el cliente.

Esta metodología tiene grandes ventajas, como su gran flexibilidad de adaptación ante cualquier imprevisto, una mayor calidad de software debido a la frecuente realización de pruebas o una mayor productividad consecuencia de un enfoque más dedicado en las tareas más importantes. Por estos motivos, creemos que esta metodología es la más adecuada para este proyecto.

4.1.3. Cronograma

Como podemos observar en el siguiente cronograma del proyecto, este tuvo inicio la semana del 30 de enero de 2023 y una duración de 3 meses y medio.

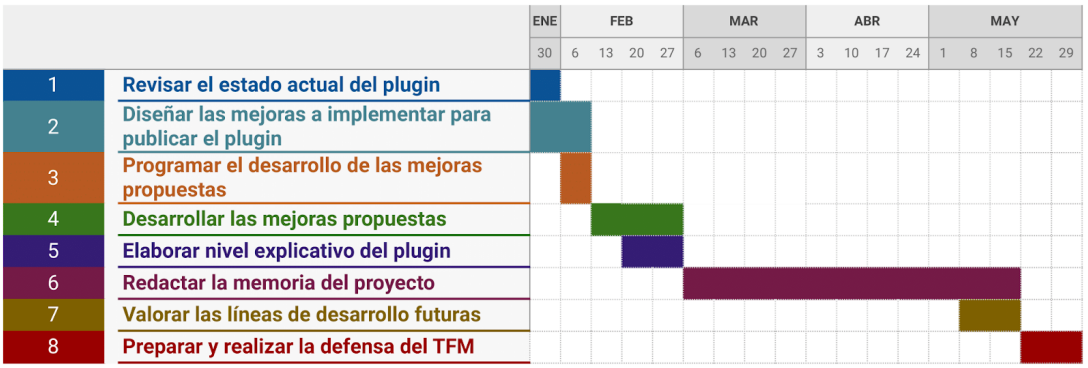

Figura 12: Cronograma del proyecto.

4.2. Herramientas utilizadas

En este apartado hablaremos de las herramientas que han sido utilizadas a lo largo de la elaboración de este proyecto.

4.2.1 Unreal Engine

Unreal Engine es un motor de videojuegos desarrollado por Epic Games en 1998 que fue utilizado inicialmente para los videojuegos *Unreal* (1998) y *Unreal Tournament* (1999) [15]. Este motor se desarrolló inicialmente con el foco puesto en videojuegos de disparos en primera persona, aunque se ha utilizado para

muchos otros géneros con gran éxito. Este motor está desarrollado en C++, que es el lenguaje de programación a usar por defecto.

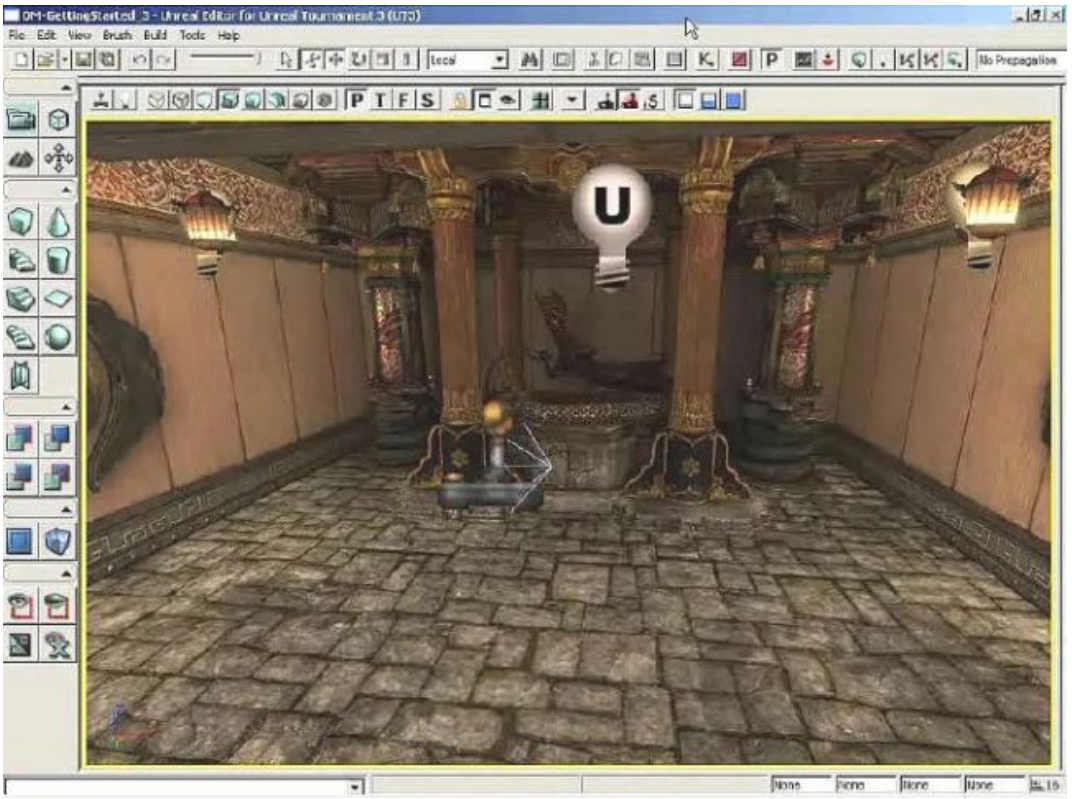

Figura 13: Primera versión del motor Unreal Engine. [16]

Entre los motivos por los que hemos escogido este motor para el desarrollo de este proyecto, destacan que es gratuito, de código disponible (o *source available*), fácil de utilizar, está diseñado para soportar fácilmente proyectos multijugador, está diseñado **especialmente para proyectos 3D**, tiene la versatilidad de **permitir usar código en C++** para secciones de código que requieran **más performance** o **Blueprints** para **tareas más sencillas** o para **prototipar**, es posible debugar el código fuente con total libertad y tiene **una gran cantidad de documentación** tanto oficial como no oficial, e incluso foros, muy sencilla de encontrar.

4.2.2. JetBrains Rider

Unreal Engine es un editor que permite configurar muchas cosas a nivel visual, pero a la hora de escribir código nos daremos cuenta de que vamos a necesitar utilizar un **IDE** (Integrated Development Environment), o editor de código, que

esté dedicado exclusivamente a ello. **La opción por defecto** suele ser el editor **Visual Studio**, de Microsoft, aunque hay otras opciones según lo que necesitemos.

Una de ellas es **Rider, de JetBrains**. Este es un editor relativamente reciente que incorpora de forma nativa asistencia de código y tiene la posibilidad de instalar un plugin en alguna versión del motor o en algún proyecto para tener una **integración total con Unreal Engine**. Lo equivalente a este IDE sería Visual Studio con el plugin *Visual Assist*, con la diferencia de que Rider dispone de una licencia gratuita para estudiantes. Rider se ha hecho tan popular que cada vez son más los usuarios de Visual Studio que se pasan a este editor tras ver su potencial. Por otro lado, hay cierto sector que asegura usar Rider para programar y Visual Studio + Visual Assist para debugar.

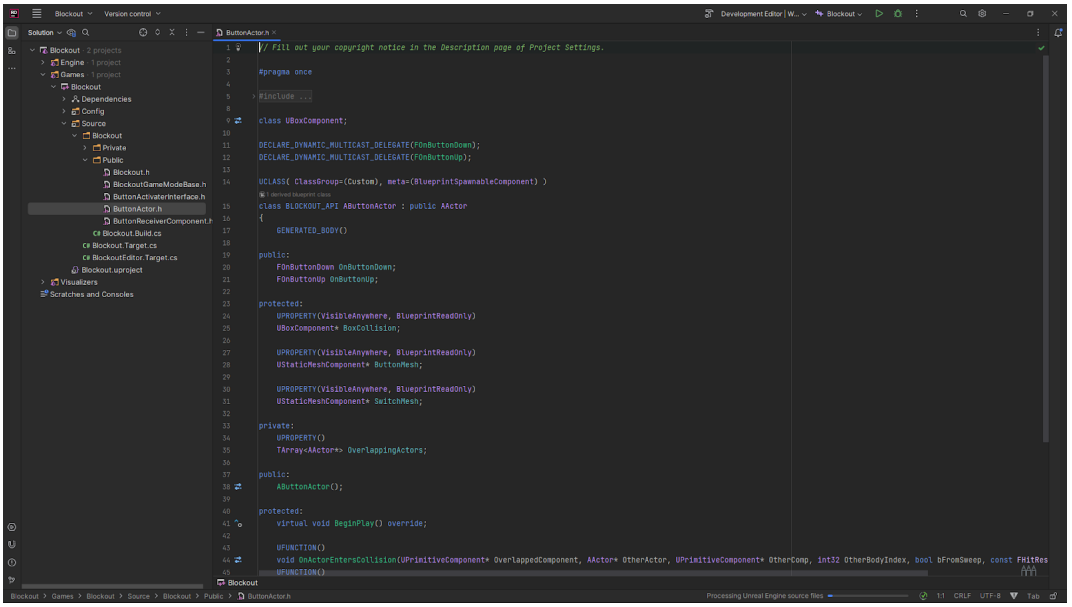

Figura 14: Interfaz gráfica de JetBrains Rider.

En resumen, consideramos **JetBrains Rider el IDE adecuado para este proyecto** por su gran potencial y por ser una opción gratuita para nosotros.

4.2.3. Plastic SCM

Plastic SCM es un sistema de control de versiones desarrollado por Códice Software en 2006 [16]. No fue hasta 2020, tras la compra de Códice Software por

parte de **Unity** y su establecimiento como sistema de control de versiones oficial del mismo, que este software **adquiriera la popularidad que tiene hoy en día**.

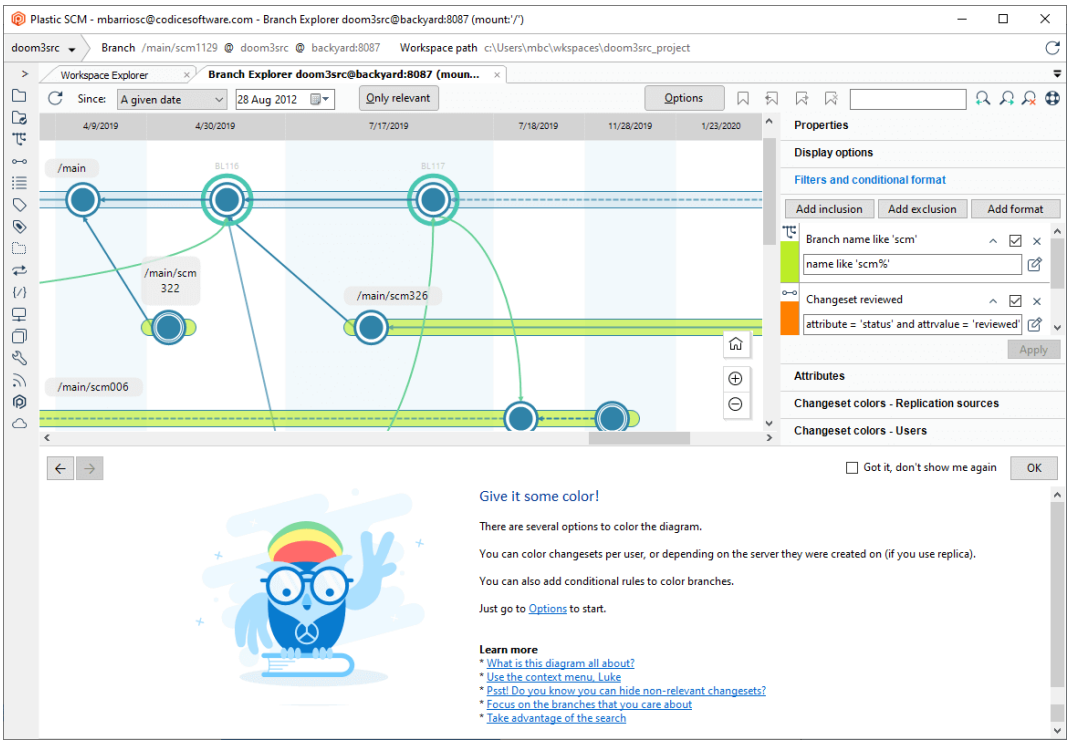

Figura 15: Interfaz gráfica de Plastic SCM. [17]

Entre los **puntos fuertes** de Plastic SCM, destacamos **su facilidad de uso**, su intuitiva interfaz gráfica, su eficiencia para grandes proyectos y su manejabilidad a la hora de **gestionar y mezclar ramas** [18]. Por estos motivos creemos que es el software adecuado para trabajar en un entorno de desarrollo orientado a plugins, como es el caso de este proyecto. A parte, cuenta con los **Xlinks**, que han resultado de gran utilidad en este proyecto dado que permiten guardar los plugins en repositorios separados para luego referenciarlos desde cada proyecto en los que se vayan a usar y mezclar los cambios hechos en los mismos desde los diferentes proyectos.

4.2.4. Hubstaff

Hubstaff es una herramienta para empresas y empleados que ayuda a realizar un control de las horas trabajadas. **Permite asignar tareas y proyectos** a los **empleados** y que éstos indiquen con precisión el tiempo dedicado a cada uno de

ellos. **Esta herramienta ha sido fundamental en este proyecto para llevar un control sobre las horas dedicadas al mismo.**

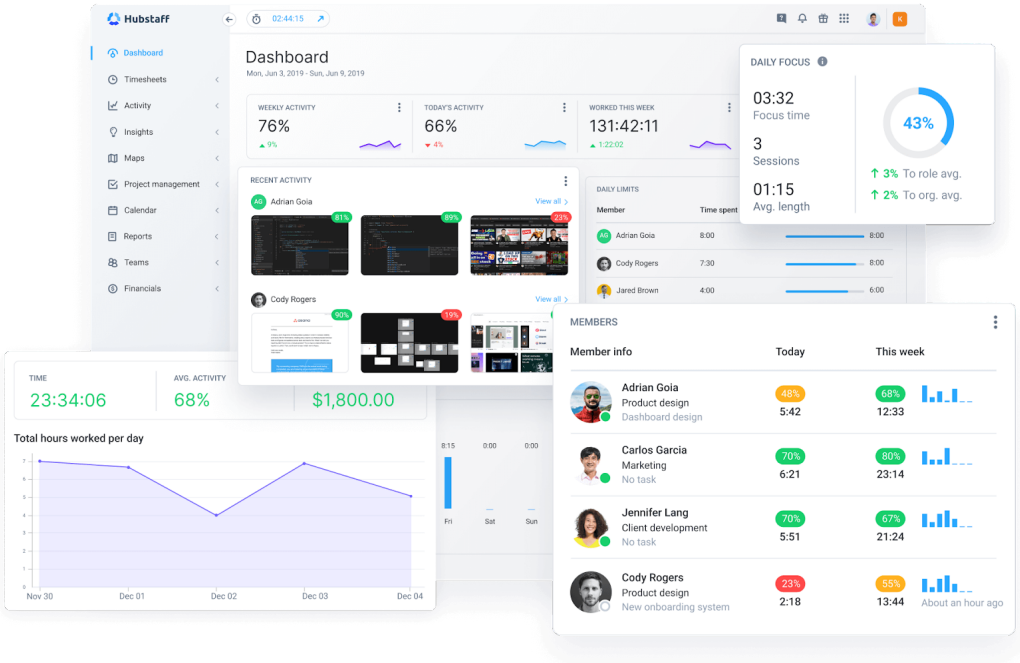

Figura 16: Diferentes interfaces de Hubstaff. [19]

4.2.5. Google Workspace

Toda la comunicación dentro de la empresa se realiza a través de **Google Chat**. Asimismo, **diariamente a las 10:00 se realiza una reunión "Standup"** donde se comentan los objetivos actuales, el trabajo realizado el día anterior y el trabajo planeado para el día en curso. Esta unificación de las vías de comunicación permite una mayor efectividad en la misma. Por otra parte, el uso de Google Workspace optimiza el uso del disco duro al no requerir la instalación de ningún software más allá de un navegador de internet de los que, por norma general, cualquier ordenador dispone.

4.3. Desarrollo

En este apartado, expondremos a grandes rasgos el desarrollo realizado a lo largo de este proyecto. El estado del plugin de portales previo al inicio de este proyecto se puede ver en [20] y [21].

4.3.1. Portales

El origen de todo es la clase **Portal**. Todos los tipos de portales que se creen deberán heredar de esta clase. Esta clase hereda de **Actor** y contiene toda la funcionalidad de un portal: la **conexión con otro portal**, la **representación visual** del propio portal y el **control de la interacción** de los actores con el mismo.

Para generar la **representación visual**, el portal dispone de un componente **Portal Camera Component** que hereda de Actor Component y se encarga de generar a cada frame un fotograma que corresponde a lo que veríamos si al mirar a dicho portal nuestra línea de visión se teletransportarse al portal conexo; además de una malla estática donde mostrar el output de la Portal Camera.

Para controlar la interacción de los actores con el mismo, **el portal dispone de tres colisiones** diferentes que se encargan de diferentes asuntos:

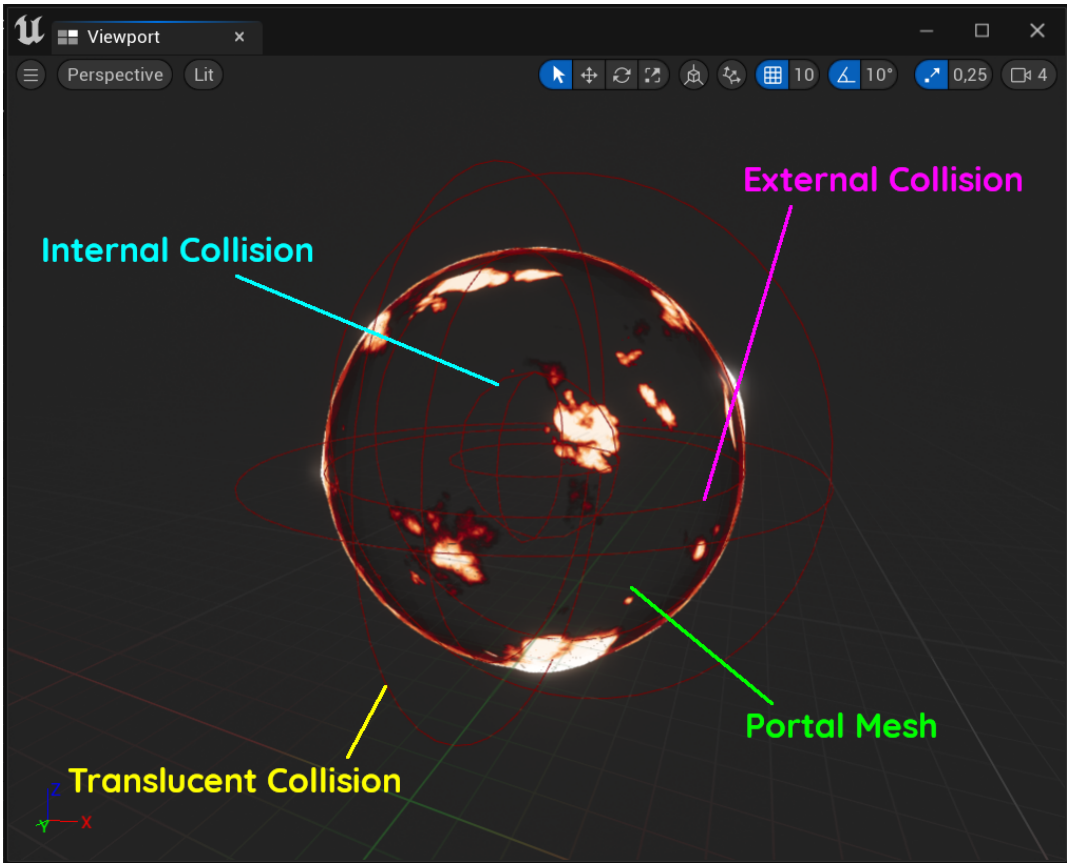

Figura 17: Diferentes colisiones en un portal esférico.

- **Internal Collision**: Hereda de *Shape Component*. Cuando un **actor solapa** con esta colisión, este **es teletransportado** al portal conectado en caso de que no haya una transacción de teletransporte en curso. Si no se hiciera esta última comprobación, al solapar con esta colisión el actor sería teletransportado instantánea e infinitamente de un portal a otro, porque en cada teletransporte se estaría detectando un solapamiento con esta colisión. Esto se producirá desde el servidor para que todos los clientes reciban la actualización en el estado del juego.
- **External Collision**: Hereda de *Shape Component*. Cuando un jugador solapa con esta colisión, se spawnea en la escena o, en caso de que hubiese sido creado con anterioridad, se hace visible; un actor de post procesado que lo que hace es mostrar en todo el viewport, ya no solo en el portal, lo que el jugador vería desde el portal conectado. Además, se interpola entre la vista del jugador y el output de este actor de post procesado para que, cuanto más cerca esté el jugador del centro del portal, más peso tenga el punto de vista de destino. Esto se ejecutará desde el cliente del jugador que esté solapando con el portal porque **solo para él debe ser visible la transición visual causada por el portal**.
- **Translucent Collision**: Hereda de *Shape Component*. Cuando un jugador solapa con esta colisión, se ejecuta un efecto de rematerialización del portal. Este efecto es necesario, ya que, cuando un jugador se ha teletransportado a la ubicación de un portal, este ha sido ocultado por motivos visuales. Esto se ejecutará desde el cliente del jugador que esté solapando con el portal porque solo para él debe ser visible el efecto de rematerialización del portal.

En el caso de las tres colisiones previamente descritas, solo se detectarán solapamientos con actores que contengan un **Teleport Component**, por lo que, como es lógico, los portales solo podrán interactuar con actores que se puedan teletransportar.

4.3.2. Recursividad

En muchas ocasiones a lo largo de esta memoria hemos mencionado el concepto de **recursividad** en portales, pero ¿A qué nos referimos exactamente cuando hablamos de recursividad en portales?

A través de un portal podemos ver el mundo desde el punto de vista del portal con el que está conectado. **¿Qué ocurre cuando desde un portal vemos ese mismo portal porque entra dentro del campo de visión del otro portal?** Ahí es cuando se produce la **recursividad**.

En estas situaciones, para poder generar la vista del portal, antes tenemos que generar la vista del portal que hay dentro (nivel de recursividad 2), pero a su vez para generar dicha vista antes tenemos que generar la del portal que hay dentro de ese (nivel de recursividad 3), y así infinitamente.

Como no es posible de ninguna manera replicar el comportamiento que acabamos de exponer, la **solución** sería **limitar el nivel de recursividad máximo al que queremos llegar**. Entonces, el portal en dicho nivel de recursividad lo renderizaremos como si dentro de la vista de ese portal no existiese dentro ningún portal recursivo. Si lo hacemos así, nunca obtendremos una vista perfecta con infinitos portales recursivos, pero si el nivel máximo de recursividad que configuramos es lo suficientemente alto, lograremos que no se note. **Alternativamente, podemos decidir no ser tan precisos con los portales recursivos en favor de un mayor rendimiento**. En la figura 18, se muestra una comparativa de un mismo portal con diferentes niveles máximos de recursividad:

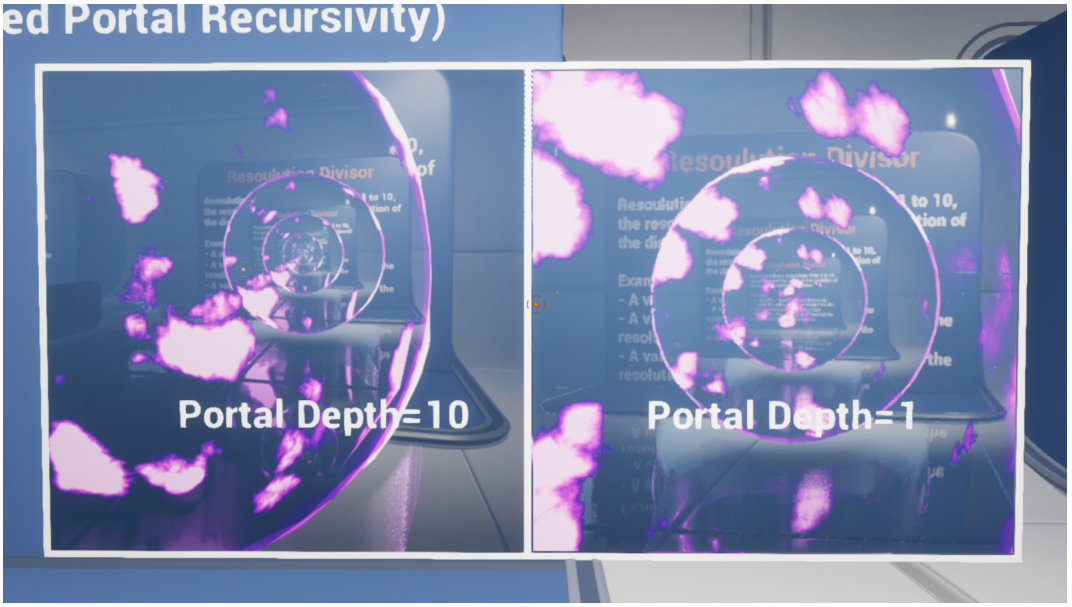

Figura 18: Demostración de diferentes ajustes para portales recursivos.

Si nos fijamos en el que tiene un valor más bajo, se nota claramente que en el último portal recursivo faltan más portales dentro, pero, por temas de rendimiento, podemos decidir que no nos importa. Por otra parte, podemos observar que, en el que tiene el valor más alto, es prácticamente imposible distinguir cuál es el último nivel de recursividad, aunque depende mucho de la resolución y el tamaño a los que lo estemos viendo.

4.3.3. Portal Performance Settings

El Portal Camera Component que mencionamos en el apartado anterior es una de las piezas clave en este proyecto. Uno de sus atributos más importantes es la estructura Portal Performance Settings, ya que está definirá cómo se van a pintar los portales. Estos **ajustes** nos permitirán ajustar de muchas maneras el renderizado de los portales **para adaptar el rendimiento de nuestro plugin de portales a cualquier tipo de dispositivo**.

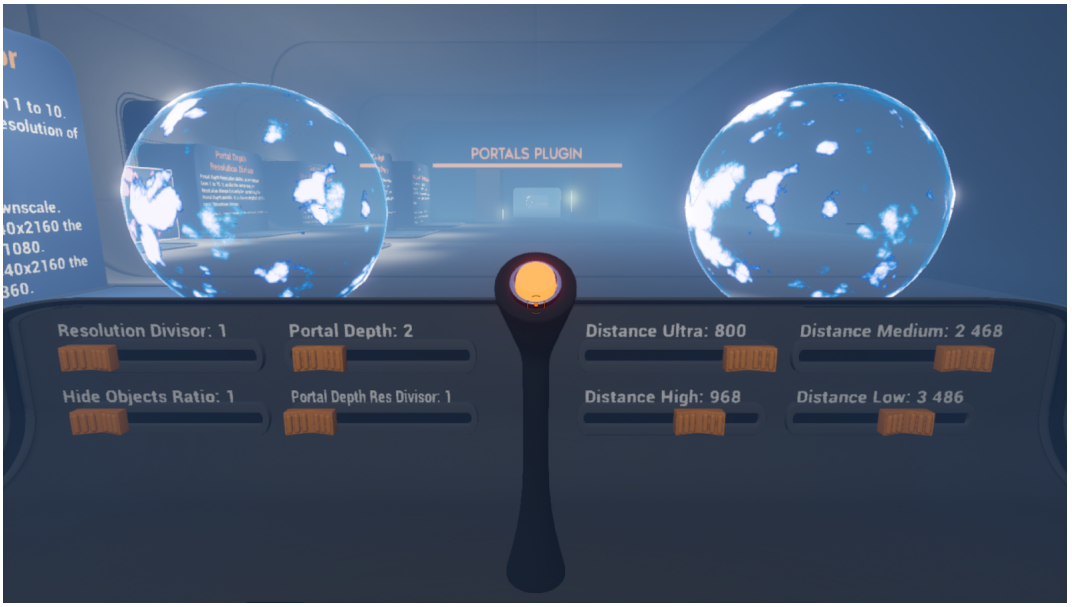

Figura 19: Portal Performance Settings.

Los diferentes ajustes que se pueden realizar son los siguientes:

- **Resolution Divisor**: Por defecto, las texturas donde se renderizarán los outputs de los diferentes Portal Camera Component en la escena tendrán un tamaño en píxeles igual al tamaño del viewport. El Resolution Divisor actuará como divisor de ese tamaño y así, **cuanto mayor sea, menor será el tamaño de la textura generada**, por lo que se consumirán menos recursos a cambio de perder resolución.
- **Portal Depth**: Este atributo define la **cantidad de portales recursivos** que se renderizarán por cada portal. Más adelante hablaremos sobre la recursividad en portales.
- **Hide Objects Ratio**: Este atributo define el ratio de fotogramas al que se actualizará qué objetos están ocultos para el portal en cuestión. Es algo que debe actualizarse con frecuencia, ya que depende de la posición y el

punto de vista del jugador. Podemos definir un valor mayor para este atributo para que esta actualización se haga cada más fotogramas y así ahorrar recursos.

- **Portal Depth Resolution Divisor**: Este atributo haría **lo mismo que el Resolution Divisor**, pero solo se aplicaría a **portales recursivos**, adicionalmente al ya mencionado Resolution Divisor.
- **Maximum Distance for (Ultra/High/Medium/Low)**: Los portales se pueden renderizar a cuatro posibles resoluciones diferentes: Ultra, High, Medium o Low. Cuando más cerca estemos de un portal, este usará la calidad Ultra y, a medida que nos alejemos de este, irá reduciendo la calidad a High, Medium y Low para ahorrar recursos. **Estos atributos definen la distancia máxima a la que se usará cada una de las resoluciones**. Si el jugador está a más distancia de la máxima para la resolución Ultra, se intentará con la High, y así sucesivamente. Cuando la distancia entre el portal y el jugador supere la distancia máxima para la menor de las resoluciones, la Portal Camera dejará de renderizar y el portal seguirá mostrando el renderizado del último frame en que lo hizo. Esto causa problemas visuales que comentaremos más adelante.

4.3.4. Portal Camera Component

El **Portal Camera Component** es el componente **responsable** de la **representación visual de los portales**. Hace todo lo necesario para generar el render final del portal que lo posee.

Un portal consta de varias cámaras: la del portal principal y la de cada uno de los portales recursivos. El Portal Camera Component es el responsable de generarlas dinámicamente y de actualizar su posición cuando es necesario. Además, también se encarga de actualizar la resolución de renderizado dependiendo de la distancia entre el portal y el jugador. Además, el Portal Camera Component hace lo necesario para actualizar, según los parámetros de rendimiento que le hayamos configurado, la lista de objetos ocultos para el portal, la cual depende del *near clipping plane* del portal, el cual a su vez depende de la posición del jugador.

El *near clipping plane* de una cámara es un plano perpendicular al vector *forward* (hacia adelante) de la cámara que actúa de tal manera que todo lo que se encuentra situado entre este y la cámara se deja de renderizar.

El Portal Camera Component actuará exclusivamente en cliente, ya que la representación visual que controla es única para cada uno de los clientes.

4.3.5. Láseres

Otra característica disponible en el plugin y que, aunque se aleja un poco del tema central de portales, permite ver el potencial de estos, es la de láseres.

Llamamos **láser** a **cualquier rayo que se pueda lanzar en una escena y que sea capaz de cruzar un portal**. Este "láser" tiene múltiples aplicaciones, como **activar mecanismos que activan puertas** (como en el videojuego *Portal*) o incluso **realizar un disparo en un videojuego de tipo *shooter***.

Este láser funciona básicamente lanzando un *line trace* desde el punto de origen con la dirección que le demos y, si detectamos una colisión con un portal, definimos el punto de impacto como la posición final del rayo y lanzaremos otro rayo diferente desde la posición de impacto teletransportada al portal conectado que tenga una longitud igual que el original sustrayéndole la distancia que ya ha recorrido hasta el portal con el que ha colisionado. El rayo, si decidimos que la tenga, tendrá como representación visual un sistema de partículas con la textura de rayo que le apliquemos. El láser se calculará desde el servidor y será replicado a todos los clientes porque su efecto es común y compartido para todos los clientes.

En la figura 19 podemos ver el resultado de este efecto:

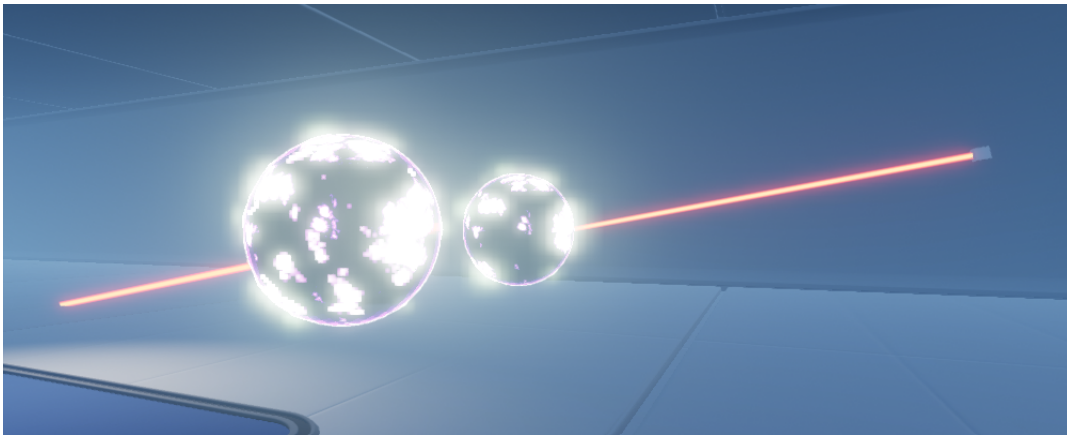

Figura 20: Láser cruzando un portal.

¿Qué ocurre si, tras haber sido teletransportado a través de un portal, un rayo vuelve a colisionar con ese mismo portal? Lo que pasará es lo que vemos en la figura 20: que el rayo cruzará ese mismo portal una y otra vez. Como en el caso de los portales recursivos, también **hay que definir un nivel de recursividad**

máximo para evitar situaciones en las que el rayo cruzaría el portal infinitamente.

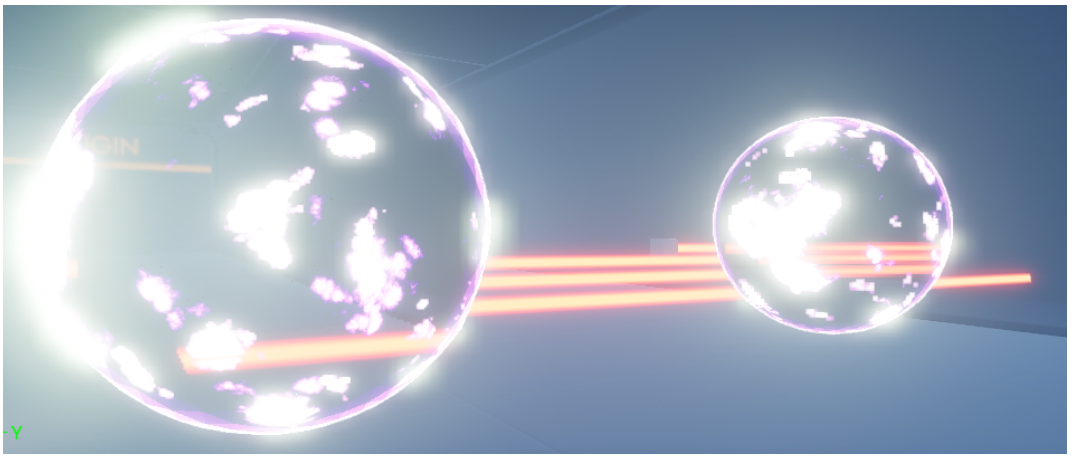

Figura 21: Láser cruzando un portal recursivamente.

4.3.6. Problemas encontrados

En este apartado, expondremos los principales problemas que encontramos y solucionamos en el plugin de portales.

4.3.6.1. Portales recursivos

En el momento de empezar este proyecto, los portales recursivos se veían extraños. Cuando uno los veía, se notaba claramente que había algo mal con ellos, pero era complicado decir exactamente el qué.

El problema que había era que la cámara de todos los portales recursivos, independientemente de su nivel de recursividad, estaban usando un punto de vista de recursividad de nivel 1. Esto es, la cámara de todos los portales recursivos estaban ubicadas en la misma posición, que era la posición que debía corresponder a la cámara del primer portal recursivo.

Para facilitar un mejor entendimiento, lo ilustraremos con un fragmento de pseudocódigo:

```
TransformAcumulado = CamaraJugador.Transform
for IndiceRecursividad = 1:NumeroPortalesRecursivos
    TransformAcumulado = TeleportarTransform(TransformAcumulado,
PortalOrigen, PortalDestino)
```

```
    PortalesRecursivos[IndiceRecursividad].Camara.Transform =
TransformAcumulado
```

Es decir, por cada nivel de recursividad había que “teletransportar” el punto de vista de la cámara con respecto al nivel de recursividad anterior, lo cual no se estaba haciendo y por eso se veían mal.

4.3.6.2. Salto visual

En el momento de empezar este proyecto, cuando uno cruzaba un portal se producía una especie de salto visual que hacía evidente en qué momento el jugador había sido teletransportado. Esto se debía a que los Scene Capture Component no se renderizan exactamente igual que la cámara actual de la escena. No había un solo motivo por el que se produjese este salto visual, era más bien una combinación de varios factores.

Dos de los factores que producían un salto visual eran la aplicación tanto de Motion Blur como del método de antialiasing TAA (Temporal Antialiasing). El TAA lo que hace es combinar el frame actual con frames pasados para producir una imagen con menos aliasing. El uso de estos efectos generaba artefactos que provocaban una peor visualización de los portales, además del salto visual que comentábamos con anterioridad. Lo que se hizo fue deshabilitar Motion Blur y escoger otro método de antialiasing.

Una vez corregidos estos factores, se seguían viendo saltos visuales, sobre todo en las paredes y suelos, que eran superficies especialmente brillantes. La conclusión fue que había que deshabilitar los reflejos en todos los materiales de la escena porque, mientras que la cámara del jugador es capaz de capturarlos, no está soportado por defecto en el motor de Unreal que un Scene Capture Component los capture porque ello implicaría un coste de rendimiento insostenible.

4.3.7. Mejoras añadidas

En este apartado, expondremos las principales mejoras que añadimos al plugin de portales.

4.3.7.1. Más precisión

Hasta el momento de iniciar este proyecto, las colisiones entre los portales y los actores se detectaban con cualquier componente de colisión del actor, lo que hacía que, cuanto más grande fuera esta, más lejos estaría el actor del portal en el momento en el que el portal decidiese teletransportar al actor. Lo ideal sería que, en el momento en que un actor se acercase a un portal, se le creara una colisión

muy pequeña y el portal solo se fijase en esa colisión para decidir en qué momento teletransportar al actor. Pues esto es precisamente lo que se realizó durante este proyecto.

4.3.7.2. Niebla

Como medida de ahorro de recursos, cuanto más se alejase un jugador de un portal, menor sería la calidad de renderizado de este, hasta tal punto que el renderizado del mismo se acabaría pausando. En estos casos, la visualización del portal se rompía y se le empezaban a ver las costuras. Sumado a este efecto era necesaria la aplicación de uno de los recursos más usados en videojuegos: la niebla.

En la gran mayoría de los videojuegos, para ahorrar recursos, cuando un objeto está muy lejos de la cámara se deja de renderizar. Para disimular este efecto, se añade un efecto de "niebla" que lo que hace es teñir todos los actores en la escena de un color de niebla de tal manera que, cuanto más alejados están estos actores de la cámara, más intenso es este color.

Con los portales, aplicamos este mismo principio: a la vez que se está jugando con la calidad de los portales a medida que nos acercamos y alejamos de estos, aplicamos un efecto de "niebla" tiñendo el portal de negro a medida que nos alejamos. Así, conseguimos disimular el efecto.

4.3.7.3. Portales planos

En el momento de iniciar este proyecto, los únicos portales que existían eran de forma esférica. Lo que hicimos fue extraer toda la funcionalidad de portales esféricos de la clase Portal y meterlo en una clase Sphere Portal que heredase de esta. Tras esto, creamos una clase Plane Portal y adaptamos la malla y las colisiones que usaba el portal esférico para que tuviera forma de portal plano.

Hubo que hacer ajustes prácticamente milimétricos para cada una de las colisiones para que el plano Near de la cámara del jugador no rompiese el portal plano en el momento de acercarnos demasiado. También hubo que hacer ajustes muy precisos para que el jugador se teletransportase justo en el punto exacto que buscábamos.

4.3.8. Documentación

Una documentación más detallada sobre el uso del plugin se puede encontrar en [22]. Un vídeo introductorio al plugin con el nivel de ejemplo con el que se ha trabajado en este proyecto puede verse en [23].

Conclusiones y Líneas futuras

Como hemos podido ver a lo largo de este proyecto, **trabajar de una manera modular orientada a plugins que facilite la reutilización** de nuestro trabajo es una manera de **optimizar** el mismo.

Los **portales** son un recurso muy útil que no se casa con un solo estilo de videojuego y **puede utilizarse para un gran número de géneros** aportando una mecánica muy atractiva que puede dar mucho juego a cualquier juego que la utilice.

Los videojuegos **multijugador** son un sector que actualmente tiene una **gran fuerza** debido al **gran valor añadido** que llega a aportar a un videojuego la participación de **diferentes jugadores que rompan la monotonía del videojuego** gracias a la **naturaleza** en muchas ocasiones **aleatoria del ser humano**.

Estos tres factores pueden ser ingredientes para nuestra receta del éxito. Cuando trabajamos, es importante no solo que lo hagamos de **manera óptima**, sino que el **producto** que generamos sea **atractivo**. Sabemos que **una fórmula funciona cuando perdura en el tiempo** y muchos tratan de imitarla añadiendo su toque personal, pero nunca debemos dejar a un lado la **búsqueda de la originalidad**. Quién sabe, a lo mejor algún día es uno de nosotros el precursor de una nueva tendencia.

Con respecto al desarrollo del plugin, nos vimos obligados a tener que dejar algunas cosas en el tintero por falta de tiempo. Por ejemplo, queríamos hacer que el plugin fuera compatible con distintas versiones del motor de Unreal para poder llegar a un público mayor. Esta y otras mejoras menores aparecerán en **versiones futuras** del plugin.

Summary and Conclusions

As we have seen throughout this project, **working in a modular plugin-oriented way that eases the reuse** of our work is a way of **optimizing** it.

Portals are a very useful resource that is not bound to any specific video game style and **can be used in many different genres** for providing an attractive mechanic that can work well with any kind of game.

Nowadays, **multiplayer** video games are a really strong sector due to the **great value given** to any video game by the involvement of **many different players that can break the monotony of the video game** thanks to the, in many occasions, **arbitrary human nature**.

These three factors can be ingredients for our recipe of success. When we work, it is not only important to do it in an **optimal way** but to make an **attractive product**. We know that **a formula works when it persists over time** and many try to replicate it by adding their personal touch, but we can **never stop looking for originality**. Who knows, maybe one day it could be one of us who starts a new trend.

With respect to the development of the plugin, we found ourselves forced to leave some things out due to lack of time. For example, we wanted to make the plugin compatible with different versions of the Unreal Engine to reach a wider audience. This and other minor improvements will be part of **future versions** of the plugin.

Bibliografía

[1] RAE. (s. f.). *Portal*. Diccionario de la lengua española. https://dle.rae.es/portal

[2] Neyra, A. (2020, 22 mayo). 40 años de Pac-Man: cómo el comecocos acabó con el sexismo en las salas de juegos. *El País*. https://elpais.com/elpais/2020/05/22/icon_design/1590159484_898128.html

[3] *Play Snake 2 Nokia 3310 Online*. (s. f.). FunHTML5Games.com. https://funhtml5games.com/?play=snake2

[4] Archer, J. (2022, 22 diciembre). Portal with RTX is a flawed diamond of a ray traced remaster. *Rock Paper Shotgun*. https://www.rockpapershotgun.com/portal-with-rtx-is-a-flawed-diamond-of-a-ray-traced-remaster

[5] Toms, O. (2023, 8 febrero). How to make a Nether Portal in Minecraft (fastest method). *Rock Paper Shotgun*. https://www.rockpapershotgun.com/minecraft-how-to-make-a-nether-portal

[6] EvaNiNorimasu. (2020, 25 mayo). *Super Buu breaks out the Hyperbolic Time Chamber* [Vídeo]. YouTube. https://www.youtube.com/watch?v=vrY61JEgOfQ

[7] *Vice Shout | Dragon Ball Wiki | Fandom*. (s. f.). Dragon Ball Wiki. https://dragonball.fandom.com/wiki/Vice_Shout

[8] Casademont, R. S. (2021, 17 julio). «Rick y Morty» temporada 5: El avance del capítulo 5 soluciona una gran duda de los fans. *Fotogramas*. https://www.fotogramas.es/series-tv-noticias/a37055471/rick-y-morty-temporada-5-opening-pistola-portales/

[9] FIGHTING CINEMA. (2022, 23 mayo). *Peter Parker vs Dr. Stephen Strange in the movie Spider-Man: No Way Home (2021)* [Vídeo]. YouTube. https://www.youtube.com/watch?v=yknvwyHiz4Q

[10] Powster. (s. f.). *Suzume | Official Website*. Suzume | Official Website. https://www.suzume-movie.com/synopsis/

[11] RAE. (s. f.). *Multijugador*. Diccionario de la lengua española. https://dle.rae.es/multijugador

[12] Wikipedia contributors. (2023). Multiplayer video game. *Wikipedia*. https://en.wikipedia.org/wiki/Multiplayer_video_game

[13] Moris. (s. f.). *Client-Server vs. Peer-to-Peer Networks | FS Community.* Knowledge. https://community.fs.com/blog/client-server-vs-peer-to-peer-networks.html

[14] *Downloading Unreal Engine Source Code.* (s. f.). Unreal Engine 5.1 Documentation. https://docs.unrealengine.com/5.1/en-US/downloading-unreal-engine-source-code/

[15] Colaboradores de Wikipedia. (2023b). Unreal Engine. *Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/wiki/Unreal_Engine

[16] Wikipedia contributors. (2023a). Plastic SCM. *Wikipedia*. https://en.wikipedia.org/wiki/Plastic_SCM

[17] Scm, P. (s. f.). *Plastic SCM version control · GUI guide*. https://docs.plasticscm.com/gui/plastic-scm-version-control-gui-guide

[18] TrustRadius. (2022, 2 marzo). *Make workflow flexible wit Plastic SCM | TrustRadius*. https://www.trustradius.com/reviews/plastic-scm-2022-03-02-06-35-18

[19] Hubstaff. (s. f.). *Hubstaff | Time Tracking and Productivity Monitoring...* https://hubstaff.com/

[20] Bionic Ape. (2019, 6 octubre). *[WIP] Portal Plugin - Unreal Engine #Devtober* [Vídeo]. YouTube. https://www.youtube.com/watch?v=A6Ev0OeLf1A

[21] Bionic Ape. (2020, 14 febrero). *Portals Plugin Promo* [Vídeo]. YouTube. https://www.youtube.com/watch?v=KtvWMe8FGSo

[22] *PortalsPluginDoc*. (s. f.). Google Docs. https://docs.google.com/document/d/17Dv1AmRjg4xdSyrdXjgqu0PG1c4qtIoAViAt3xUZOa8/edit

[23] Bionic Ape. (2023, 14 abril). *Demo Portals* [Vídeo]. YouTube. https://www.youtube.com/watch?v=-wTN4bb1Iaw