Universidad San Jorge

Escuela de Arquitectura y Tecnología

Grado en Diseño y Desarrollo de Videojuegos

Proyecto Final

Desarrollo de NeonHat: Un videojuego arcade de carreras y vuelo para PlayStation VR

Autor del proyecto: Javier Verón Mérida

Director del proyecto: Jaime Font Burdeus

Zaragoza, 08 de septiembre de 2021

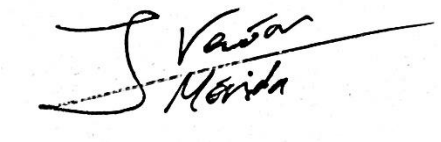
universidad SANJORGE
GRUPO SANVALERO

Este trabajo constituye parte de mi candidatura para la obtención del título de Graduado en Diseño y Desarrollo de Videojuegos por la Universidad San Jorge y no ha sido entregado previamente (o simultáneamente) para la obtención de cualquier otro título.

Este documento es el resultado de mi propio trabajo, excepto donde de otra manera esté indicado y referido.

No doy mi consentimiento para que se archive este trabajo en la biblioteca universitaria de Universidad San Jorge, donde se puede facilitar su consulta.

Firma



Fecha

08/09/2021

Dedicatoria y Agradecimiento

Gracias a mi familia por todo su apoyo. Sin ellos, nada de lo que he conseguido sería posible.

Gracias a mi pareja, Lucía Martínez, por disfrutar conmigo en los días buenos y aguantarme en los días malos.

Gracias a mis amigos y compañeros de *Entalto Studios* por confiar en mí tanto como yo en ellos y embarcarse conmigo en este proyecto.

Gracias a los amigos que he hecho durante la carrera, que me han ayudado siempre sin dudarlo.

Gracias a la Universidad San Jorge por habernos apoyado desde el principio, cuando comenzamos a acercarnos a este mundo con la empresa Junior *4FreaksFiction*.

Gracias a mi tutor del TFG, Jaime Font; y a mi tutora a lo largo de mi estancia en la Universidad, África Domingo.

Tabla de contenido

Resumen

Desarrollo; centrado en la programación; del videojuego *NeonHat*: un videojuego arcade de carreras y peleas contra jefes para la plataforma *PlayStation 4* y el visor de Realidad Virtual *PlayStation VR* controlable con *Move Controllers* o *DualShock 4*.

Este proyecto forma parte de un videojuego real desarrollado por *Entalto Games S.L.* en colaboración con *Lanzadera Emprendedores, S.L.U.* y *Sony Interactive Entertainment* que ha sido publicado entre julio y agosto de 2021 a escala mundial.

Los principales puntos de interés son: la mejora del movimiento del jugador y su control para Realidad Virtual; la implementación diversos sistemas como el de audio, el de localización, o el de guardado; la programación de jefes y menús; el cambio total de diseño que el videojuego sufre a los meses de desarrollo; y la portabilidad a la consola objetivo.

Abstract

Development; focused on programming; of the video game *NeonHat*: an arcade video game about racing and fighting bosses for the platform *PlayStation 4* and the Virtual Reality visor *PlayStation VR* controllable with *Move Controllers* or *DualShock 4*.

This project is part of a real video game developed by *Entalto Games S.L.* in collaboration with *Lanzadera Emprendedores, S.L.U.* and *Sony Interactive Entertainment* that has been published between July and August 2021 worldwide.

The main points of interest are the improvement of the player's movement and control for Virtual Reality; the implementation of various systems such as audio, localisation, or storage; the programming of bosses and menus; the total change of design that the video game undergoes after some months of development; and the portability to the target console.

1.1. Palabras clave

VR (*Virtual Reality*), Realidad virtual, *Motion Sickness*, jefe (ámbito de los videojuegos), *PlayStation*, arcade

2. Introducción

NeonHat (Entalto Games S.L., 2021) es un videojuego arcade[1], exclusivo de VR[2], jugable con *DualShock 4* (Sony Interactive Entertainment, 2013) y *Move Controllers* (Sony Interactive Entertainment, 2010) para *PlayStation* 4 *(Sony Interactive Entertainment, 2013)*. Se lanzó al mercado durante el verano de 2021 y está traducido a 8 idiomas.

Ilustración 1: Captura de pantalla durante una carrera en el juego NeonHat. Fuente: NeonHat - PSTalents: The Moment Teaser (Entalto Games S.L., 2021)

En este juego se compite en 63 carreras diferentes a lo largo de 9 circuitos diferentes y se lucha contra 5 jefes[3]. Es un juego para un solo jugador, pero con tablas de clasificación mundiales, en el que la principal mecánica es el vuelo.

Tras años de desarrollo de una tecnología de vuelo en VR perfeccionada por parte del equipo de *Entalto* (Entalto Games S.L., 2021), se ha seleccionado como mecánica principal en este videojuego arcade competitivo. El estudio y desarrollo de esta tecnología se han asentado en

1 Un género que engloba a todos los videojuegos centrados en la jugabilidad y basados en un diseño minimalista, dificultad ascendente, puntuación como objetivo, y pocas interrupciones.
2 Realidad Virtual (*Virtual Reality*).
3 En el contexto de los videojuegos, un jefe se refiere a un desafío de dificultad superior que se presenta al jugador generalmente antes de un cambio de escenario/fase del juego.

evitar el efecto de *Motion Sickness*[4] en el jugador cuando este se mueva por el escenario, algo bastante común en los juegos de VR.

Este trabajo, por tanto, forma parte de un proyecto real con otros 6 compañeros y diversos colaboradores externos.

Mi papel en el proyecto ha sido el de programador, encargándome de una enorme variedad de aspectos del juego: desde los menús, hasta las peleas contra jefes finales, pasando por mecánicas principales como el giro.

4 *Motion Sickness* es como se denomina a la sensación que ocurre cuando tu cerebro no encuentra sentido a la información enviada por tus sentidos, provocando mareos, dolor de cabeza, fatiga, o náuseas entre otros problemas (Cleveland Clinic, 2021).

3. Estado del arte

3.1. Inmersión mediante los sistemas de control

Los sistemas de control para un videojuego son una parte esencial de la interacción de los jugadores con el mismo, al ser la forma en que el jugador comunica sus acciones al juego.

De igual forma, la inmersión del jugador siempre ha sido un punto importante que se ha tratado de mejorar con abundantes y variados sistemas; como la mejora en el apartado gráfico de un juego, o aportando características a los sistemas de control que conecten directamente con la experiencia que el jugador espera. Este subapartado se centra en estas últimas adiciones a los sistemas de control.

Ilustración 2: DualDigital Controller para Sony PlayStation. Fuente: https://es.wikipedia.org/wiki/DualDigital

Los controladores más básicos tienen únicamente botones y, en algunos casos, gatillos; pero debido a esta carrera por conseguir la mayor inmersión de los jugadores, a día de hoy podemos encontrar controladores con acelerómetros, giroscopios, diferentes formas de retroalimentación háptica, o incluso seguimiento espacial (con cámaras infrarrojas u otros sistemas).

3.1.1. Seguimiento espacial en los controladores

A lo largo de la primera década de los años 2000, las grandes desarrolladoras de consolas han creado diferentes sistemas para seguir al mando en el espacio y tratar de entender sus movimientos. El primer gran exponente de esto fue el controlador para la consola *Wii* (Nintendo Company, Ltd., 2006): el *Wii Remote* (Nintendo Company, Ltd, 2006).

Este controlador cuenta con un acelerómetro, que le permite detectar la aceleración a lo largo de tres ejes; y un sensor óptico que le permite determinar el lugar al que se está apuntando al detectar la luz de la barra sensor de la consola, la cual cuenta con diez LED infrarrojos y cinco LED dispuestos en cada extremo de la barra. (Turner, 2007)

Ilustración 3: Videojuego Wii Sports siendo jugado con el Wii Remote, utilizando su seguimiento del espacio. Fuente: https://www.techradar.com/news/gaming/consoles/eight-things-you-didn-t-know-the-wii-could-do-270149

Durante la década de los 2010, los controladores con seguimiento espacial continuaron su evolución para alcanzar mayor exactitud en la determinación de su posición y dirección en el espacio; como podemos ver con los controladores *Joy-Con* (Nintendo Company, Ltd., 2017) de la consola *Nintendo Switch* (Nintendo Company, Ltd., 2017); pero la evolución de esta tecnología hizo posible la aparición de un nuevo sistema de juego: la Realidad Virtual.

Algunos fabricantes, como *HTC* (HTC Corporation, 1997) y *Oculus VR* (Oculus VR, 2012), comenzaron a desarrollar controladores centrados en el seguimiento espacial y cascos de Realidad Virtual, también con su propio sistema de seguimiento espacial, haciendo posible una nueva experiencia de juego.

Ilustración 4: Una persona utilizando el dispositivo Oculus Quest 2. Fuente: https://www.pocket-lint.com/es-es/ra-y-rv/noticias/oculus/154036-el-escritorio-virtual-te-permitira-jugar-juegos-de-pc-vr-en-el-oculus-quest-2-a-90-hz

3.1.2. PlayStation Move Motion Controller

En 2010, *SIE* (Sony Interactive Entertainment Inc., 1993) sacó al mercado los controladores *PlayStation Move* (Sony Interactive Entertainment Inc., 2010) para su consola *PlayStation 3* (Sony Interactive Entertainment Inc., 2006) para competir con los *Wii Remote* (Nintendo Company, Ltd, 2006).

Ilustración 5: Controladores PlayStation Move de Sony. Fuente: https://www.pushsquare.com/news/2017/10/sonys_tweaking_the_playstation_move_motion_controller_too

Este mando tiene una esfera capaz de iluminarse y ser encontrada por la cámara *PlayStation Eye* (Sony Interactive Entertainment Inc., 2007), haciendo a la consola capaz de detectar la posición en el espacio del controlador con un único sensor óptico.

Para mayor precisión en el seguimiento de los mandos en un espacio tridimensional, *SIE* (Sony Interactive Entertainment Inc., 1993) reemplazó esta cámara con la *PlayStation Camera* (Sony Interactive Entertainment Inc., 2013) al salir al mercado su siguiente consola: la *PlayStation 4* (Sony Interactive Entertainment, 2013). Esta nueva cámara tiene dos sensores ópticos de mayor calidad, que permiten calcular la profundidad con mayor exactitud.

Ilustración 6: Cámara PlayStation Eye de Sony. Fuente: https://www.amazon.es/Sony-Cámara-playstation-eye-ps3/dp/B000W3YQ1Y

Ilustración 7: Cámara PlayStation Camera de Sony. Fuente: https://www.playstation.com/es-es/accessories/playstation-camera/

3.1.3. DualShock 4

La salida de la consola *PlayStation 4* (Sony Interactive Entertainment, 2013) también significó la salida de un nuevo controlador principal para la consola: el *DualShock 4* (Sony Interactive Entertainment, 2013).

Ilustración 8: Controlador DualShock 4 de Sony para su consola PlayStation 4. Fuente: https://as.com/meristation/2020/01/16/noticias/1579174537_957572.html

Este controlador cuenta con acelerómetro y con una luz en su lateral más alejado del jugador, que combinada con el sistema de seguimiento de *PlayStation Camera* (Sony Interactive Entertainment Inc., 2013) hace que la consola pueda saber la posición del mando de una forma tan exacta como al utilizar los controladores *Move* (Sony Interactive Entertainment, 2010); algo que podría ser útil en ciertos juegos que lo requirieran, pero que no pudo demostrar todo su potencial hasta que *SIE* (Sony Interactive Entertainment Inc., 1993) sacó al mercado en 2016 su dispositivo de Realidad Virtual: *PlayStation VR* (Sony Interactive Entertainment, 2016).

3.1.4. Sinergia con PlayStation VR

Con la aparición de *PlayStation VR* (Sony Interactive Entertainment, 2016), tanto los controladores *Move* (Sony Interactive Entertainment, 2010) como la posibilidad de seguimiento espacial del *DualShock 4* (Sony Interactive Entertainment, 2013) cobraron un nuevo sentido: ahora el jugador podría jugar a juegos de Realidad Virtual con los dos controladores siempre que

comprara el nuevo dispositivo, que dispone de dos pantallas para situar en los ojos del jugador y permitir un nuevo salto de inmersión.

Ilustración 9: Imagen publicitaria de PlayStation VR en la que el jugador aparece utilizando este dispositivo y los controladores Move de Sony. Fuente: https://www.playstation.com/es-es/ps-vr/ps-vr-games/

Ilustración 10: Imagen publicitaria de PlayStation Camera en la que el jugador aparece utilizando el dispositivo PlayStation VR y el controlador DualShock 4 de Sony. Fuente: https://www.playstation.com/es-es/accessories/playstation-camera/

3.2. Trabajo Previo

A día de hoy, existe una cantidad ingente de videojuegos, con cada día más y más títulos en el mercado. Por esta razón, únicamente voy a hablar sobre algunos juegos anteriores a *NeonHat* (Entalto Games S.L., 2021) que cumplan con buscar un público objetivo y puntos de interés similares.

3.2.1. Juegos de carreras y arcade

Crazy Taxi (Hitmaker, 1999) es un videojuego clásico arcade con un público objetivo similar al de *NeonHat* (Entalto Games S.L., 2021). En este videojuego tu objetivo es conseguir el mayor dinero posible en el tiempo dado guiado por una flecha tridimensional que apunta en la dirección del destino para guiar al jugador, quien controlará al taxista buscando llevar a sus clientes al punto indicado en el menor tiempo posible. Esto lo hace un juego frenético y relacionado con la velocidad.

Ilustración 11: Crazy Taxi siendo jugado en una máquina arcade. Fuente: http://www.keithsarcade.com/crazy-taxi.html

En la idea original de *NeonHat* (Entalto Games S.L., 2021), este videojuego era la principal referencia en cuanto a flujo de juego y, aunque la idea cambiara, el componente frenético y arcade de *Crazy Taxi* (Hitmaker, 1999) sigue estando presente.

La idea final para *NeonHat* (Entalto Games S.L., 2021), sin embargo, es un juego de carreras con peleas contra jefes. Como ejemplo en el ámbito de carreras podemos encontrar la serie de videojuegos *Wipeout* (Psygnosis / SCE Studio Liverpool, 1995-2017). Esta serie de juegos de carreras es conocida por tener un ritmo rápido, una estética futurista, y una banda sonora electrónica.

Ilustración 12: Captura de pantalla durante una carrera en videojuego Wipeout. Fuente: https://www.vidaextra.com/conduccion/wipeout-omega-collection-se-vuelve-desde-hoy-compatible-con-playstation-vr-por-medio-de-una-actualizacion-gratuita

Otro ejemplo de trabajo previo entre los juegos de carreras centrados en la diversión del jugador por encima de otros aspectos como el realismo es la saga de juegos *Mario Kart* (Nintendo Company, Ltd., 1992-2020). Como escribe en su análisis de la novena entrega de esta serie principal de esta saga A. Arribas (Arribas, 2017), "*Mario Kart* ha sido siempre sinónimo de diversión y calidad […] Lo primero que debemos tener muy claro es que, aunque nos parezca una propuesta simple, estamos ante un arcade de conducción realmente exigente. Lo más brillante de su diseño es que cada jugador podrá profundizar todo lo que quiera en su jugabilidad hasta encontrar un reto a su nivel".

Ilustración 13: Dos personas jugando a Mario Kart Wii, utilizando el acelerómetro del Wii Remote. Fuente: https://www.themycenaean.org/mario-kart-wii/

Ese componente arcade y la posibilidad de ser juego para gente que busque un modo de juego más difícil y profundo o menos han sido esenciales en el éxito de *Mario Kart* (Nintendo Company, Ltd., 1992-2020), lo que nos ha hecho convertir estos principios en partes esenciales para *NeonHat* (Entalto Games S.L., 2021).

En la mayor parte de las peleas contra jefes, sin embargo, la forma de jugar a *NeonHat* (Entalto Games S.L., 2021) es distinta a las carreras acercándose más al género *Endless Runner*[5].

Un videojuego a tener en cuenta debido al parecido que guarda con *NeonHat* (Entalto Games S.L., 2021) tanto en su estética como en la pertenencia a este género es *Sayonara Wild Hearts* (Simogo, 2019).

En la mayor parte de niveles de este videojuego avanzas hacia delante mediante diversos elementos como monopatines, motos, coches, o incluso volando mientras esquivas elementos de colisión y recoges corazones que aumentan tu puntuación. La acción se desarrolla en una gran variedad de espacios con una estética neón y electrónica que se ha empleado como inspiración hasta cierto punto en la dirección artística de *NeonHat* (Entalto Games S.L., 2021).

Ilustración 14: Captura de pantalla del videojuego Sayonara Wild Hearts. Fuente: https://store.steampowered.com/app/1122720/Sayonara_Wild_Hearts/

5 Un género que engloba a todos los videojuegos donde el jugador debe avanzar continuamente en un recorrido infinito y con una dirección dada, generalmente esquivando obstáculos y recogiendo recompensas.

3.2.2. Juegos de VR

Beat Saber (Beat Games, 2019) es un juego arcade exclusivo para VR; como *NeonHat* (Entalto Games S.L., 2021); y disponible también para *PlayStation 4* (Sony Interactive Entertainment, 2013). Sin embargo, en *Beat Saber* (Beat Games, 2019) el jugador no vuela, ni se mueve del sitio; sino que corta cubos que vuelan hacia él siguiendo el ritmo de la música que suene y rodeado por luces de neón. Este juego es todo un éxito en la industria de la VR, con más de 4 millones de copias vendidas y más de 40 millones de canciones que pueden ser compradas como contenido adicional del juego por los jugadores; sirviendo como claro ejemplo de que, en VR, el videojuego arcade tiene una fuerza muy notoria, haciendo que buscáramos como parte de la esencia que queríamos en *NeonHat* (Entalto Games S.L., 2021) ese componente arcade desde el principio.

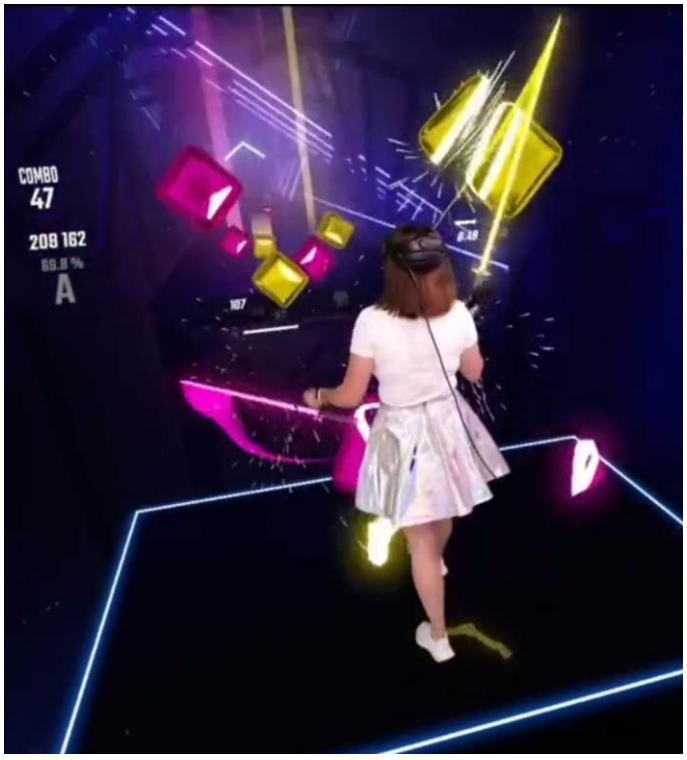


Ilustración 15: Persona jugando a Beat Saber con un dispositivo de Realidad Virtual Oculus. Fuente: https://vm.tiktok.com/ZMRSKuekJ/

Si nos centramos en la otra parte de *NeonHat* (Entalto Games S.L., 2021), en las mecánicas de vuelo y disparo, podemos encontrar a *Marvel's Iron Man VR* (Camouflaj, 2020), que salió al mercado en julio de 2020, cuando llevábamos trabajando sólo unos meses en la primera prueba de concepto.

En este videojuego, el jugador controla a "*Iron Man*" (Lee, Kirby, Lieber, & Heck, 1963), quien es capaz de volar y disparar utilizando sus manos; y es exclusivo para *PlayStation VR* (Sony Interactive Entertainment, 2016), lo que lo convierte en un ejemplo perfecto de trabajo previo. Este juego, sin embargo, no es arcade. Dispone de una historia principal que hay que completar navegando por el espacio virtual del juego y combatiendo mientras se vuela. *Marvel's Iron Man VR* (Camouflaj, 2020), además, no soporta el uso de *DualShock 4* (Sony Interactive Entertainment, 2013), teniendo como única opción de controlador los *Move Controllers* (Sony Interactive Entertainment, 2010). Además, el jugador tiene que girar físicamente como única opción para girar en el espacio del juego, haciendo así que sea común enredarse con el cable de

PlayStation VR (Sony Interactive Entertainment, 2016) y, potencialmente, disminuyendo la inmersión.

Ilustración 16: Persona jugando a Marvel's Iron Man con PSVR y controladores Move. Fuente: https://www.youtube.com/watch?v=5nq7kuPQKhU

También encontramos como trabajo previo un videojuego que se encontró en una situación muy similar a la de *NeonHat* (Entalto Games S.L., 2021): *ADroneLine* (Dinamita Works, 2017-2021). Este proyecto de videojuego fue creado por tres alumnos de la "Escuela Universitaria de Diseño, Innovación y Tecnología" (ESTUDIOS SUPERIORES INTERNACIONALES, S.L, 1988-2021) bajo el nombre de estudio "*Dinamita Work*". El juego pretendía llegar a ser un juego arcade de carreras de drones para *PlayStation VR* (Sony Interactive Entertainment, 2016); una descripción que lo hace un producto bastante similar a *NeonHat* (Entalto Games S.L., 2021). Sin embargo, pretendía ser mucho más realista, acercándose más a un simulador de vuelo de drones. Quedaron finalistas de los Premios *PlayStation* 2017 (Hobby Industria, 2017) y prolongaron su desarrollo durante 4 años, hasta que el 7 de enero de 2021 cancelaron el desarrollo del videojuego (Dinamita Works, 2021).

Ilustración 17: ADroneLine siendo jugado. Fuente: https://www.youtube.com/watch?v=DmFk08x-bus

Videojuego	Año	Similitudes con NeonHat	Acogida del público	Consola	Controles
Crazy Taxi	1999	Arcade, frenético	Extremadamente positiva	Arcade	Arcade
Wipeout	1995 - 2017	Carreras, frenético	Extremadamente positiva	PlayStation, Nintendo 64, PlayStation 2, PSP, PlayStation 3, PlayStation Vita, PlayStation 4, etc.	DualDigital, DualShock, NUS-005, DualShock 2, PSP, DualShock 3, PlayStation Vita, DualShock 4, etc.
Mario Kart	1992 - 2020	Arcade, carreras complejidad oculta	Extremadamente positiva	SNES, N64, GameBoy Advance, N GameCube, NDS, Wii, N3DS, N Wii U, NSwitch, etc.	SNES Controller, NUS-005, GameBoy Advance, GameCube Controller, NDS, Wii Remote, N3DS, Wii U GamePad, Joy-Con, Nintendo Switch Pro Controller, etc.
Sayonara Wild Hearts	2019	Endless run, frenético, estética neón	Extremadamente positiva	NSwitch, PlayStation 4, Windows, Xbox One, Apple Arcade	Joy-Con, DualShock 4, Mouse & Keyboard, Xbox One Controller, iPhone, iPad
Beat Saber	2019	Arcade, VR, estética neón	Extremadamente positiva	Oculus Quest, PlayStation 4, Windows	VR con 2 controladores
Marvel's Iron Man VR	2020	VR, vuelo, disparo	Extremadamente positiva	PlayStation 4	PSVR con Move Controllers
ADroneLine	Cancelado	VR, vuelo	_	PlayStation 4	PSVR
NeonHat	2021	_	Positiva	PlayStation 4	PSVR

Ilustración 18: Comparación de diferentes juegos previos a NeonHat. Fuente: Propia.

3.2.3. Otros medios

Para la versión final de *NeonHat* (Entalto Games S.L., 2021) planteamos una estética que guarda cierta semejanza con la película *Tron* (Lisberger, 1982). Esta película ha pasado a la historia como una obra de culto y parte de este título se debe a la estética que utiliza.

Como explica Laura Gómez en su crítica de la película (Gómez, 2013), "La idea central es la de un humano que ha sido digitalizado y absorbido por un universo digital en el interior de un ordenador. Nos metemos dentro de un videojuego".

Esta descripción guarda relación directa con la historia que se sigue a lo largo de la versión final de *NeonHat* (Entalto Games S.L., 2021), que seguirá la estética de neones e internet vista en *Tron* (Lisberger, 1982) y tendrá peleas contra algunos jefes finales como virus informáticos.

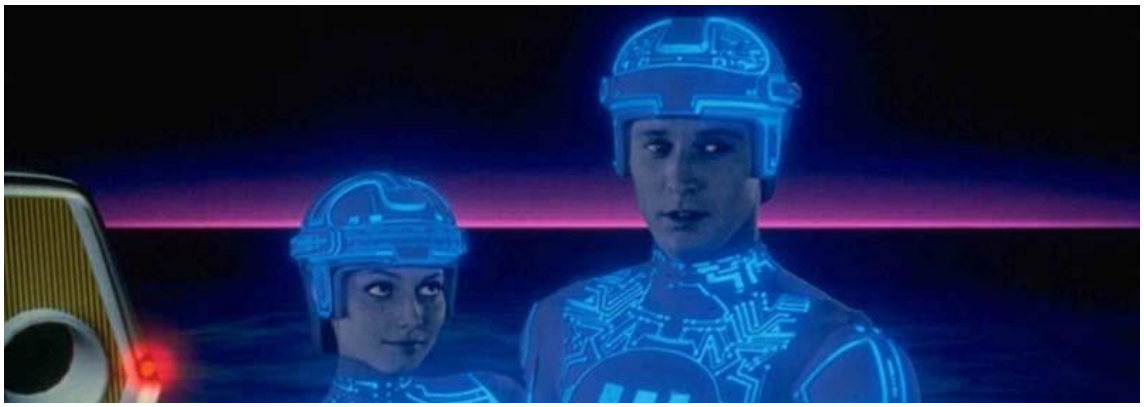


Ilustración 19: Recorte de un fotograma de la película Tron (1982). Fuente: https://cualia.es/tron-1982-de-steven-lisberger/

3.3. La aportación de este trabajo

Cuando empezamos el desarrollo de *NeonHat (Entalto Games S.L., 2021)* había una gran cantidad de videojuegos que llevaban años en desarrollo desde el programa *PlayStation Talents (Sony Interactive Entertainment Europe, 2015-2021)*, como era el caso de *ADroneLine* (Dinamita Works, 2017-2021). Esto nos hizo decidir desde el principio que nuestro desarrollo duraría menos de un año, ya que no queríamos vernos en esa situación. Eso fue, en parte, lo que a los dos meses de desarrollo nos hizo cambiar el rumbo del juego por completo.

Respecto a las mecánicas, decidimos que el jugador siempre estuviera de frente a la pantalla para un seguimiento más fácil por parte de *PlayStation VR* (Sony Interactive Entertainment, 2016), evitando además enredos con cables y problemas derivados. Con esto pretendíamos conseguir una inmersión más profunda que algunos de los videojuegos nombrados durante Trabajo Previo, pero esto requirió una gran dedicación y tiempo enfocados en la integración del

giro y adiciones al vuelo para tratar de evitar por completo cualquier sensación de *Motion Sickness* mientras se mantenía la sensación de velocidad de estar en una carrera.

Como adición respecto a los juegos más clásicos de carreras, decidimos añadir modos de juego diferentes como el de persecución y las peleas contra jefes, las cuales requirieron también un trabajo de ensayo y error hasta conseguir la sensación deseada.

También implementamos para todos los modos de juego sus propias tablas de clasificación global, buscando la competitividad para jugadores que quieran profundizar más en el juego.

Para acabar, *NeonHat* (Entalto Games S.L., 2021) , haciendo honor al idioma en el nombre de su estudio[6], es el primer videojuego con traducción al aragonés, además de ser el primer juego aragonés para *PlayStation VR* (Sony Interactive Entertainment, 2016).

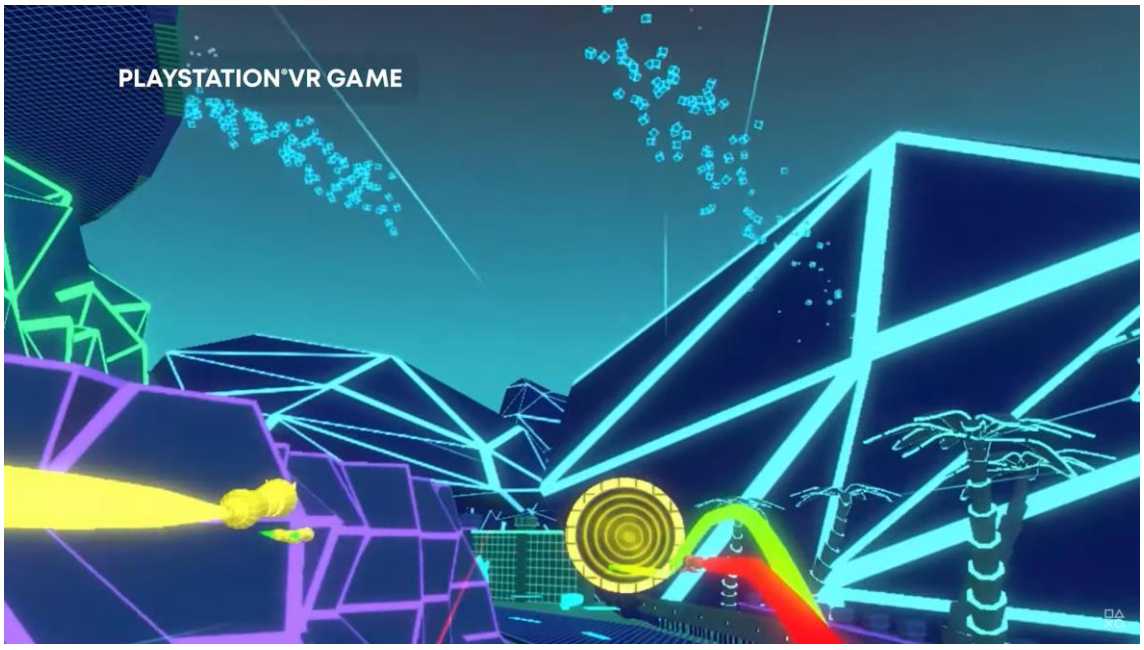


Ilustración 20: Fotograma del tráiler de lanzamiento de NeonHat. Fuente: https://www.youtube.com/watch?v=KSKg3pYcXsE

6 "*Entalto*" significa "Hacia arriba" en aragonés.

4. Objetivos

Este proyecto, al ser parte de un videojuego real, tiene unos objetivos que variaron durante el desarrollo de este, por lo que voy a hablar de objetivos iniciales y de objetivos finales por separado.

4.1. Objetivos iniciales

El objetivo principal es adaptar el sistema de vuelo existente para este nuevo videojuego, incluyendo:

- Controles adaptados para *PlayStation* VR (Sony Interactive Entertainment, 2016), tanto para controladores *Move* (Sony Interactive Entertainment, 2010) como para *DualShock 4* (Sony Interactive Entertainment, 2013).
- Sistema de guiado y búsqueda de rutas en la ciudad.
- Mejoras del sistema de vuelo:
    - Derrape,
    - Turbo,
    - Estabilización.
- Pruebas de rendimiento en la consola objetivo.

4.2. Modificación en objetivos iniciales

Al comenzar el desarrollo del juego, los objetivos iniciales se vieron aumentados con el sistema de guardado y progresión del juego, incluyendo:

- Sistema de economía.
- Sistema de gestión de misiones.
- Carga y guardado desde otro hilo.
- Soporte para varios espacios de guardado.

Finalmente, también se planeó la implementación de un sistema de audio que facilitara la adición de efectos de sonido y música en el proyecto.

4.3. Objetivos finales

Debido a problemas de falta de recursos durante el desarrollo, hubo que cambiar el diseño del videojuego, resultando en una modificación de los objetivos finales desde mi parte del proyecto.

El objetivo principal del equipo en conjunto era poder lanzar el juego a lo largo del verano de 2021 con los recursos disponibles, por lo que hubo que recortar en algunos departamentos recargando otros como el de programación, por lo que finalmente mis objetivos finales son más que los iniciales.

Se mantuvieron prácticamente iguales los siguientes apartados:

- Controles adaptados para *PlayStation* VR (Sony Interactive Entertainment, 2016), tanto para controladores *Move* (Sony Interactive Entertainment, 2010) como para *DualShock 4* (Sony Interactive Entertainment, 2013).
- Mejoras del sistema de vuelo:
    - Derrape,
    - Turbo,
    - Estabilización.
- Pruebas de rendimiento del sistema en la consola objetivo.
- Sistema de gestión de misiones.
- Carga y guardado sin interrumpir el juego.
- Soporte para varios espacios de guardado.
- Sistema de audio.

Y se añadieron los siguientes objetivos para mi proyecto:

- Desarrollo de los menús del juego.
- Desarrollo de las batallas contra jefes (incluyendo dinámicas nuevas y patrones de combate diferentes para cada uno):
    - "Troyano Titánico",
    - "Tejedora Terrible",
    - "Asaltante Acorazado",
    - "Devoradora Desatada"
- Sistema de traducciones y localización.

Sin embargo, el cambio en el diseño del videojuego supuso también la pérdida de ciertos apartados que ya se habían desarrollado: el sistema de economía y el sistema de guiado y búsqueda de rutas en la ciudad.

5. Metodología

5.1. Implicados en el desarrollo de NeonHat

Los primeros implicados somos el equipo desarrollador del juego: Entalto Studios (Entalto Games S.L., 2021).

Además, incluimos en el desarrollo diversos colaboradores externos que se encargaron de apartados como apoyo en Arte 3D, diseño de sonido, composición musical, localización, y traducción.

También contamos con el apoyo y ayuda del programa *PlayStation Talents* (Sony Interactive Entertainment Europe, 2015-2021), creado por *Gammera Nest* (Gammera Nest, 2013) y *PlayStation* España. *PlayStation Talents* participó al trabajar con el programa de emprendimiento "*Corporate*" (LANZADERA EMPRENDEDORES, S.L.U., 2019-2021) junto a Lanzadera (LANZADERA EMPRENDEDORES, S.L.U., 2011): una aceleradora de empresas que actualmente apoya a Entalto Studios (Entalto Games S.L., 2021).

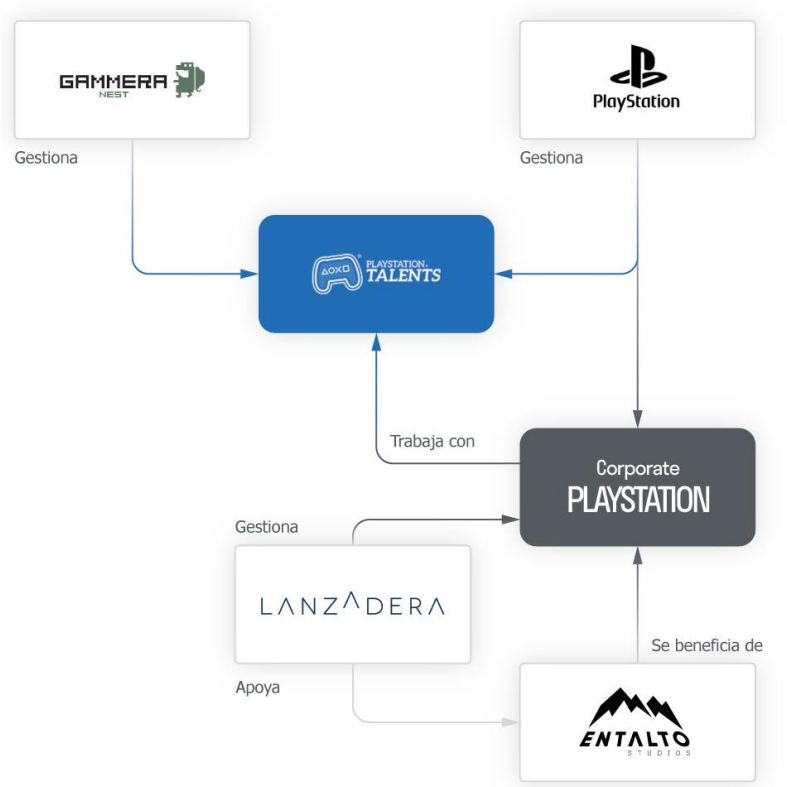


Ilustración 21: Relación entre programas y empresas implicadas en el desarrollo de NeonHat. Fuente: Propia.

En la Ilustración 22 podemos ver un organigrama con la gente implicada de forma directa en el desarrollo de *NeonHat*.

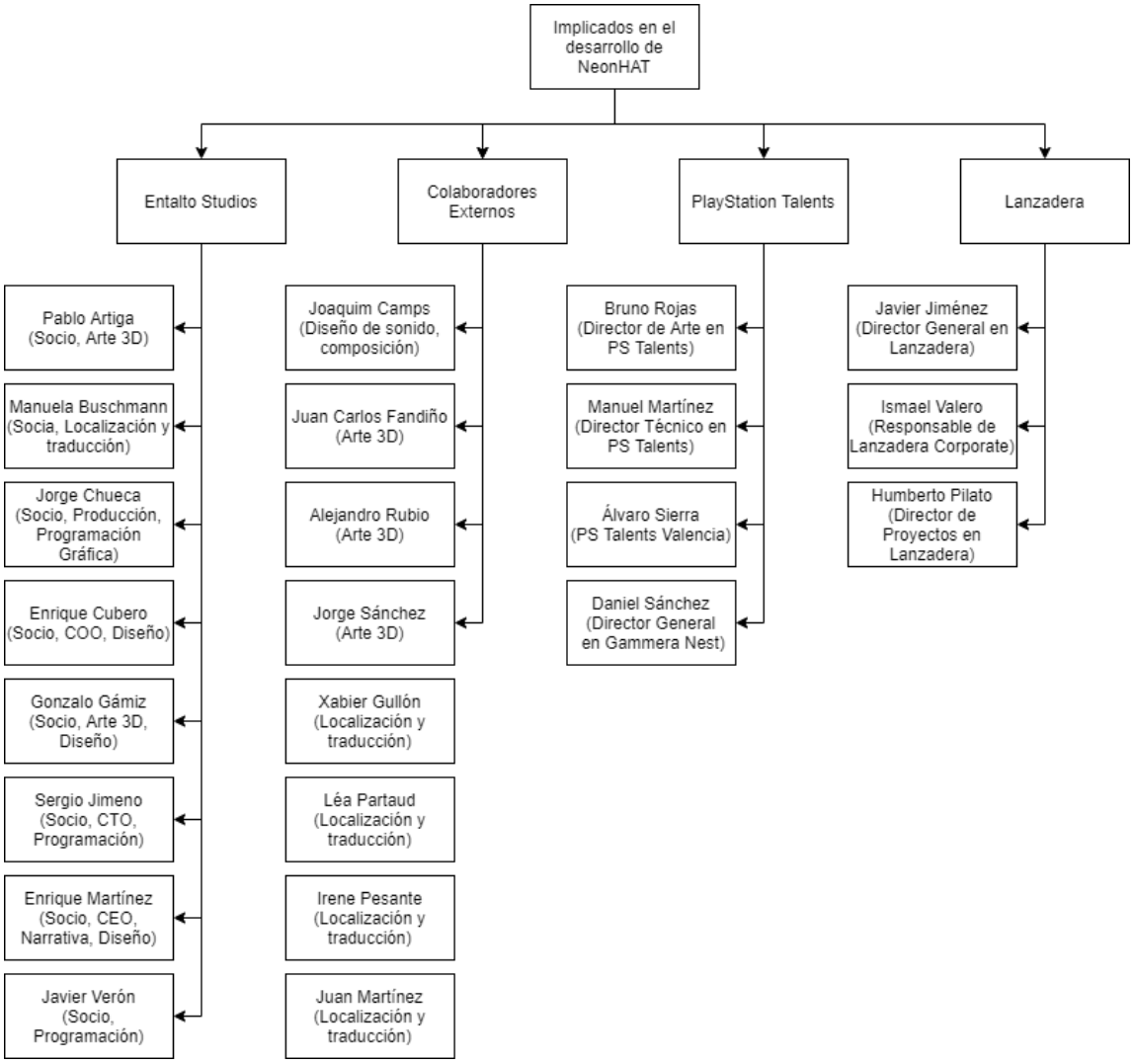


Ilustración 22: Organigrama de personas implicadas en el desarrollo de NeonHat. Fuente: Propia.

5.2. Metodología empleada y por qué ha sido elegida

Para el desarrollo del proyecto hemos aplicado la metodología *SCRUM*. El encargado de definir la metodología para este proyecto fue Jorge Chueca, productor del videojuego.

Según la definición dada por el equipo de "ProyectosAgiles.org" (ProyectosAgiles.org, Consultado en 2021), "En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los

requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales".

Teniendo en cuenta que éramos un equipo pequeño en el desarrollo y teníamos fechas que cumplir, J. Chueca decidió que *SCRUM* era la metodología adecuada para el proyecto.

SCRUM es una metodología con mucha facilidad para la adaptación. Esto se debe a que antes de cada *sprint*[7] se planifica el trabajo de este, y al final de cada *sprint* se realiza una revisión y una retrospectiva para comprobar si los resultados son satisfactorios.

El equipo *SCRUM* se compone de las siguientes figuras:

- Actuando como "Propietario del Producto" se encuentra *Entalto Games S.L.* (Entalto Games S.L., 2021) junto al programa de emprendimiento "*Corporate*" (LANZADERA EMPRENDEDORES, S.L.U., 2019-2021) de *PlayStation Talents* y Lanzadera.
- Actuando como "*Scrum Master*" se encuentra Jorge Chueca, productor en *Entalto Games S.L.* (Entalto Games S.L., 2021) durante el desarrollo de *NeonHat* (Entalto Games S.L., 2021).
- El "Equipo de Desarrollo *SCRUM*" está compuesto por las siguientes personas:
    - Desde *Entalto Games S.L.* (Entalto Games S.L., 2021):
        - Manuela Buschmann (Localización y traducción)
        - Enrique Cubero (Diseño)
        - Gonzalo Gámiz (Diseño y arte 3D)
        - Pablo Artiga (Arte 3D)
        - Jorge Chueca (Programación gráfica y producción)
        - Enrique Martínez (Narrativa, Diseño, y CEO)
        - Sergio Jimeno (Programación y CTO)
        - Javier Verón (Programación)
    - Y como colaboradores externos a *Entalto Games S.L.* (Entalto Games S.L., 2021), se encuentran:
        - Juan Carlos Fandiño (Arte 3D)
        - Joaquim Camps (Diseño de sonido y composición)
        - Alejandro Rubio (Arte 3D)
        - Jorge Sánchez (Arte 3D)
        - Xabier Gullón (Localización y traducción)
        - Léa Partaud (Localización y traducción)

7 "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". (Schwaber & Sutherland, 1991-2016)

- Irene Pesante (Localización y traducción)
- Juan Martínez (Localización y traducción)

Para ser conscientes del progreso del proyecto y poder entender el videojuego en toda su complejidad, una parte esencial durante los *sprints* fueron los "*SCRUM* diarios": reuniones cortas realizadas a diario para sincronizar las actividades de cada miembro del "Equipo de Desarrollo *SCRUM*" y poder planear la jornada de trabajo tras inspeccionar el trabajo avanzado desde el anterior "*SCRUM* diario".

5.3. Herramientas

Para poder desarrollar *NeonHat* (Entalto Games S.L., 2021) se han necesitado una serie de herramientas que veremos a continuación.

5.3.1. Discord

Para poder realizar los "SCRUM diarios", nosotros nos decantamos por la plataforma *Discord* (Discord Inc., 2015-2021), ya que ofrece una amplísima variedad de herramientas como diversos canales de voz y texto, y roles; a la vez que es muy sencillo de comenzar a utilizar.

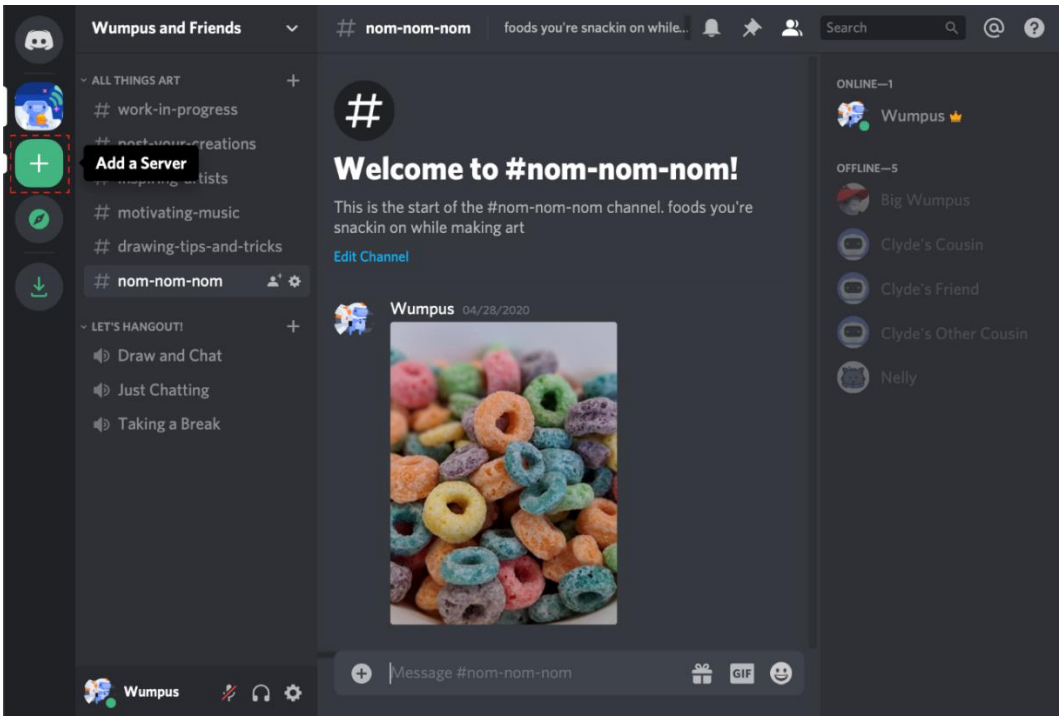


Ilustración 23: Ejemplo de uso en Discord. Fuente: https://support.discord.com/

5.3.2. hacknPlan

Otra herramienta externa que utilizamos para la planificación de los *sprints* fue *hacknPlan* (Estévez, 2015-2021), una excelente herramienta para simplificar el diseño y planificación del desarrollo de software, que además está enfocada para el desarrollo de videojuegos.

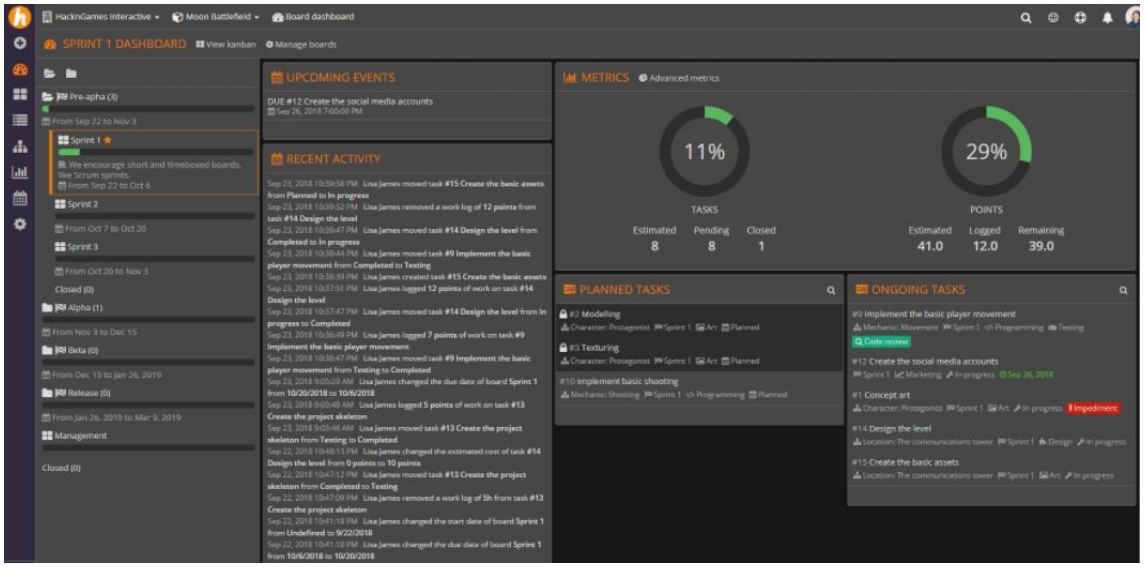


Ilustración 24: Ejemplo de uso en hacknPlan. Fuente: https://hacknplan.com/press

El flujo de trabajo en hacknPlan era el siguiente:

Jorge Chueca crea todas las tareas del *sprint* que vamos a comenzar en la columna para tareas "Planeadas" de cada departamento, con una prioridad (urgente, alta, media, o baja) y una duración estimada en cada tarea.

Cuando comenzamos una tarea, la movemos a la columna para tareas "En progreso"; donde vamos especificando las horas que trabajamos en cada tarea y marcando diferentes subtareas como completadas hasta finalizarla. En este momento tenemos que cambiar la tarea a la columna "*Testing*".

En cuanto la tarea se aprueba por una persona encargada de ello (que en mi caso particular solían ser Sergio Jimeno o Gonzalo Gámiz), se cambia a la columna "Completadas". Automáticamente hacknPlan actualizará el progreso y las métricas de cada apartado del proyecto para que el productor pueda comprobar el estado del *sprint* de forma más sencilla. Se puede ver un ejemplo de esto último en la Ilustración 24.

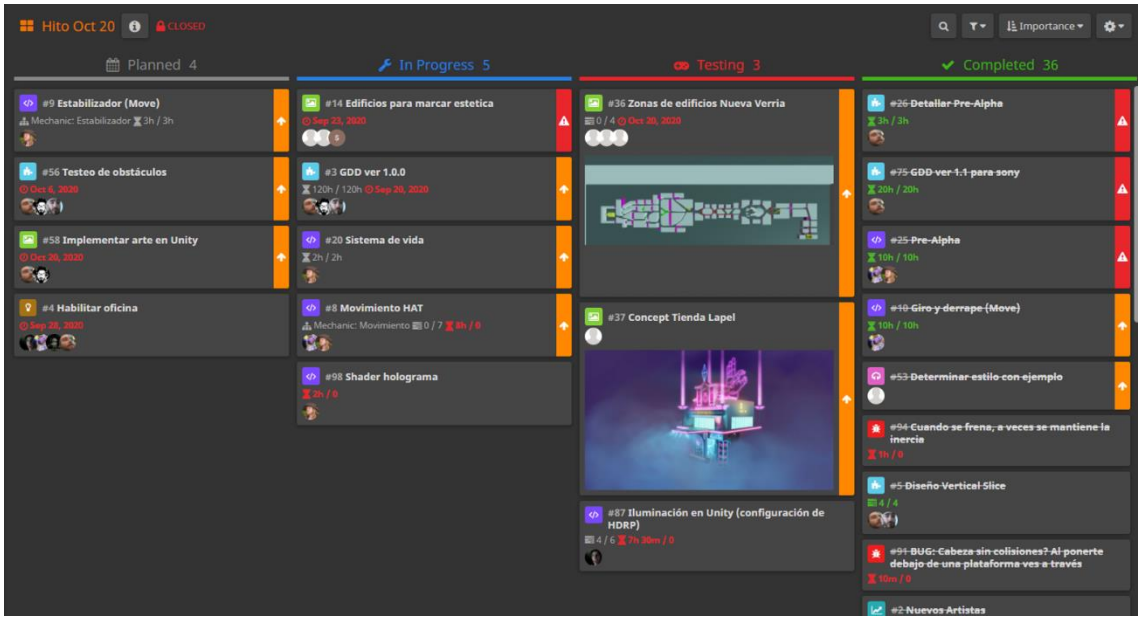


Ilustración 25: Uso real de hacknPlan en el proyecto NeonHat. Fuente: Jorge Chueca (productor de NeonHat)

5.3.3. Unity

El motor que hemos empleado para el desarrollo del videojuego completo ha sido *Unity* (Unity Technologies, 2005-2021).

Este motor provee de una interfaz gráfica donde modificar valores y entender la escena del juego de forma visual. Además, provee de un sistema de portabilidad que hace mucho más sencilla la compilación para diversas plataformas, como *PlayStation 4* (Sony Interactive Entertainment, 2013).

El lenguaje de programación con el que hemos trabajado en este motor es C# (Microsoft Corporation, 2000). Tanto mi compañero Sergio Jimeno como yo estamos muy familiarizados con este lenguaje de programación, por lo que su soporte en *Unity* (Unity Technologies, 2005-2021) ha supuesto una enorme ventaja a la hora de elegir este motor por encima de otros competidores.

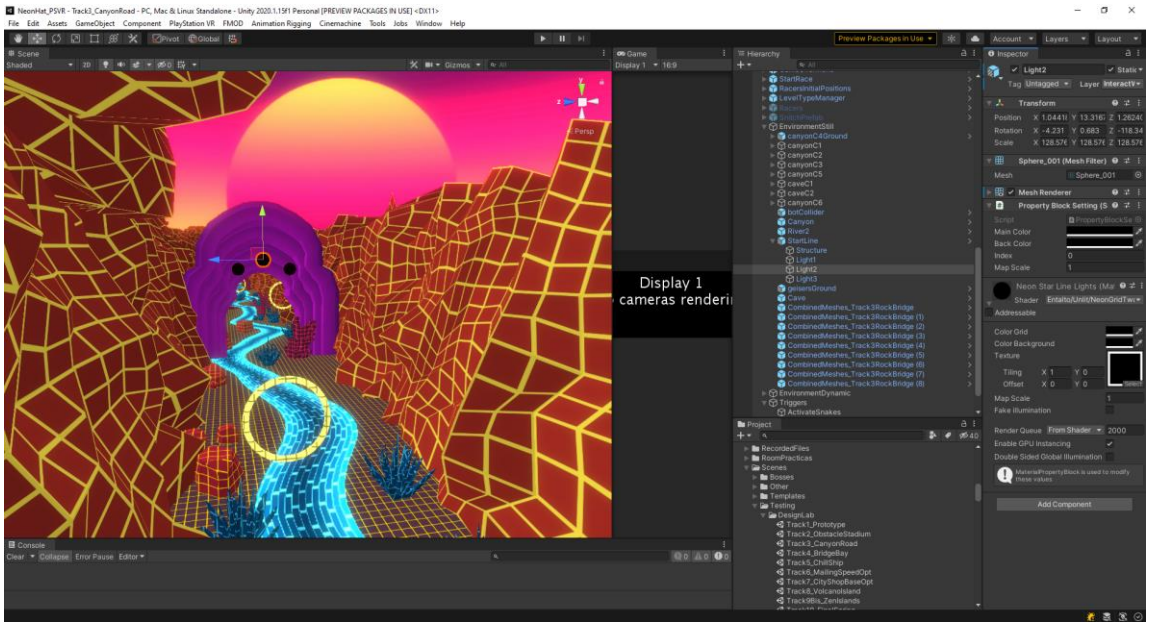


Ilustración 26: Una escena del proyecto NeonHat abierto desde el motor Unity. Fuente: Propia.

5.3.4. JetBrains Rider

Para escribir el código, un entorno de desarrollo integrado (o IDE) es una herramienta que facilita muchos aspectos como la navegación, coloreado del código para mejor legibilidad, y autocompletado. Para programar en el lenguaje que hemos utilizado, C# (Microsoft Corporation, 2000), he elegido *JetBrains Rider* (JetBrains s.r.o, 2000-2021).

La decisión final que me ha hecho escoger este IDE por encima de otros como Visual Studio (Microsoft Corporation, 1997-2019) ha sido la sencillez para personalizar a mi gusto la autosangría y autocompletado.

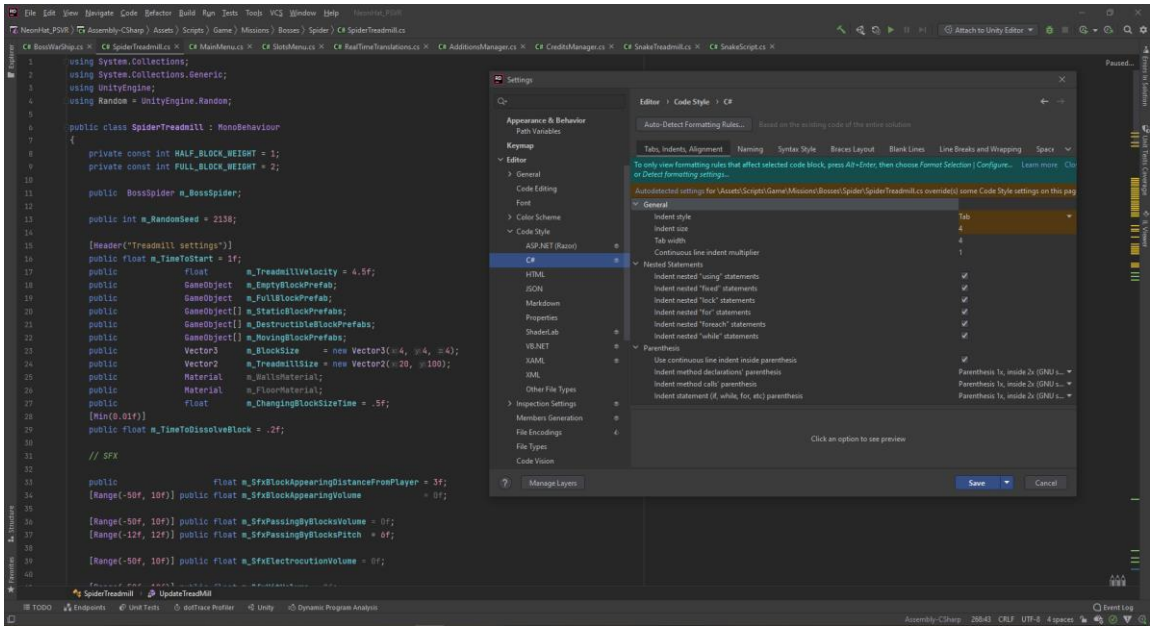


Ilustración 27: Clase SpiderTreadmill del proyecto NeonHat y ventana de ajustes abiertos en JetBrains Rider. Fuente: Propia.

5.3.5. TortoiseSVN

TortoiseSVN es un cliente de subversion[8] muy sencillo de usar. Esta fue la principal razón por la que lo utilizamos. Con este cliente, cada vez que una persona nueva se una al proyecto podrá comenzar a utilizar el control de versiones y subir su trabajo a la rama principal sin problemas.

8 Apache *Subversion* (SVN) es una herramienta de control de versiones de código abierto basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros. Es software libre bajo una licencia de tipo Apache/BSD. (The Apache Software Foundation, 2000-2021)

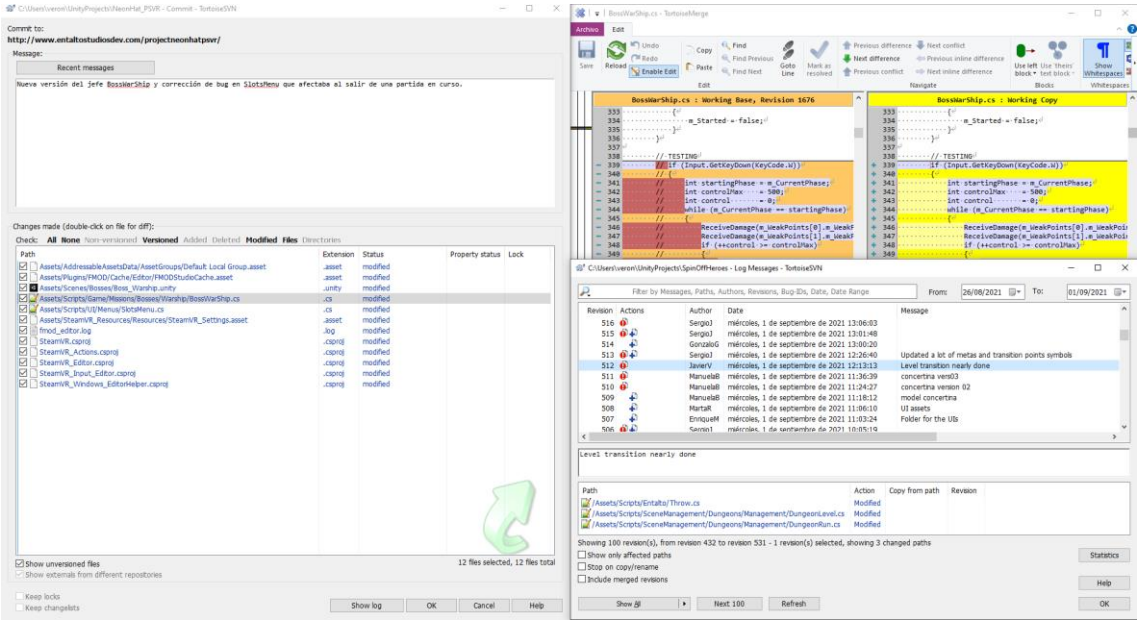


Ilustración 28: Cambios en la versión de NeonHat vistos desde el cliente TortoiseSVN. Fuente: Propia.

5.4. Planificación inicial

Desde el principio, el proyecto se había acordado con el programa "*Corporate*" (LANZADERA EMPRENDEDORES, S.L.U., 2019-2021) como un videojuego arcade, exclusivo de VR, y jugable tanto con *DualShock 4* (Sony Interactive Entertainment, 2013) como con *Move Controllers* (Sony Interactive Entertainment, 2010). También acordamos una duración de alrededor de 180 minutos como mínimo.

Para cumplir estos objetivos, inicialmente apuntamos a hacer un videojuego localizado en una gran ciudad con estética ciberpunk, donde el jugador debería completar diversas misiones volando con un dron; al que denominamos *HAT*[9]; en un mapa abierto para conseguir dinero con el que mejorar el propio *HAT*.

Para ello, mi tarea desde programación se centraba en desarrollar las diversas mejoras necesarias en el sistema de vuelo, crear todos los sistemas de juego de las diferentes misiones, un sistema de navegación, y un sistema de gestión de recursos para desbloquear las mejoras en el dron.

Durante el primer *sprint*, "Pre-alfa (Sprint 1)", me centré en las mejoras del sistema de vuelo y el desarrollo de un primer sistema de audio. El sistema de vuelo es la base más esencial del juego, al ser la mecánica principal, por lo que fue uno de los sistemas que más horas llevó hasta tener un resultado que admitimos como final.

Para el segundo *sprint*, "Vertical Slice (Sprint 2)", el sistema de navegación era la tarea más importante. Además, la versión del juego para el final de este *sprint* que ser cercana a una versión

9 *Hyper-Accelerated Terminal*.

final del juego, por lo que también fueron esenciales el sistema de controles que nos permitiera manejar diferentes tipos de entrada (para poder utilizar indistintamente un controlador u otro), el sistema de guardado y carga de la partida, y el sistema de progresión y economía.

Tras esto, dedicaría el resto de *sprints* a crear diferentes misiones y adiciones que surgieran para hacer el juego más completo, hasta llegar al final del desarrollo: el *sprint* de "*Porting* a *PlayStation* 4".

Con esto, el plan inicial en lo respectivo a programación acabó siendo el que podemos ver en la Tabla 1.

Tabla 1: Planificación original de NeonHat. Fuente: Propia.

Sprint	Objetivos	Horas
Septiembre	Pre-alfa	397
Octubre	Vertical Slice	176
Noviembre	Primer distrito con sus misiones	168
Diciembre	Mitad de segundo distrito con sus misiones	128
Enero	Segundo distrito con sus misiones	120
Febrero	Tercer distrito con sus misiones	120
Marzo	Cuarto distrito con sus misiones y sistema de mejoras	160
Abril	*Porting* a *PlayStation* 4	152
Mayo	QA y arreglo de bugs	144
	TOTAL HORAS	**1565**

5.5. Planificación final

Después del *Vertical Slice*[10] del sprint de octubre, nos dimos cuenta de que hacer una ciudad llevaba una carga de trabajo muy superior a la que teníamos prevista y no podíamos seguir con los tiempos planeados haciendo este juego.

Aquí nos encontramos con dos opciones: Alargar el desarrollo rompiendo los hitos negociados con el programa "*Corporate*" (LANZADERA EMPRENDEDORES, S.L.U., 2019-2021) y yendo en contra de nuestro acuerdo interno de no mantener este desarrollo más de un año; o seguir con los tiempos previstos, rompiendo con el juego que estábamos desarrollando y quedándonos únicamente con lo que considerábamos esencial en el juego.

Tras debatirlo internamente, llegamos a la conclusión de que la segunda opción era la correcta, con lo que planteamos el desarrollo del nuevo proyecto reutilizando todo lo posible de entre las partes que ya habíamos desarrollado.

Debido a estos problemas y a otros que surgieron durante el desarrollo, la planificación final no fue tan exacta como la anterior; sino que fue algo más flexible; y los objetivos iniciales de mi proyecto se vieron modificados:

Tabla 2: Planificación final de NeonHat. Fuente: Propia.

Sprint	Objetivos	Horas
01/10/2020	Pre-alfa	397
20/10/2020	Vertical Slice	176
07/01/2021	Demo de juego final	344
02/02/2021	Detención de producción y arreglos para *porting*	128
19/04/2021	Finalización de la producción del juego	408
10/05/2021	*Porting* a *PlayStation* 4, QAs, y arreglo de bugs	112
	TOTAL HORAS	**1565**

10 Un *Vertical Slice* se refiere a una porción de un juego que actúa como prueba de concepto. No es exactamente un prototipo ya que se espera que el producto final sea igual en calidad y mecánicas.

6. Desarrollo e Implementación

El desarrollo de *NeonHat* (Entalto Games S.L., 2021) ha estado lleno de baches que nos han hecho a todo el equipo aprender que, para trabajar en la industria de los videojuegos, hay que saber adaptarse.

La idea para este juego comenzó en marzo de 2020, cuando se declaró en España el estado de alarma con sus consecuentes restricciones de movilidad debido a la pandemia mundial por SARS-CoV-2. Mis compañeros Sergio Jimeno y Enrique Cubero habían comenzado a plantear la idea para un videojuego y me propusieron unirme a ellos para desarrollar un videojuego de misiones sencillas cuyos principales atractivos se centraban en la exclusividad para VR y la comodidad en el movimiento gracias al sistema de vuelo que S. Jimeno había desarrollado durante varios años, como podemos ver en su Proyecto Final "Diseño e Implementación de un Sistema de Navegación en Entornos de Realidad Virtual para Videojuegos Comerciales" (Jimeno Navarro, 2020). El nombre para este proyecto fue "*Neon Hat*".

6.1. Pre-alfa (Sprint 1)

La idea original de *Neon Hat* se centró en ser un videojuego en el que el jugador se encontraba encerrado en una habitación con un ordenador y unas gafas de Realidad Virtual que le permitían controlar un dron de mensajería, llamado sistema H.A.T., para recorrer las calles de *Neon City* y realizar diversas entregas con un flujo de juego parecido a *Crazy Taxi* (Hitmaker, 1999), pero pudiendo comprar mejoras para el H.A.T. desde el ordenador de la habitación con el dinero recaudado en las misiones.

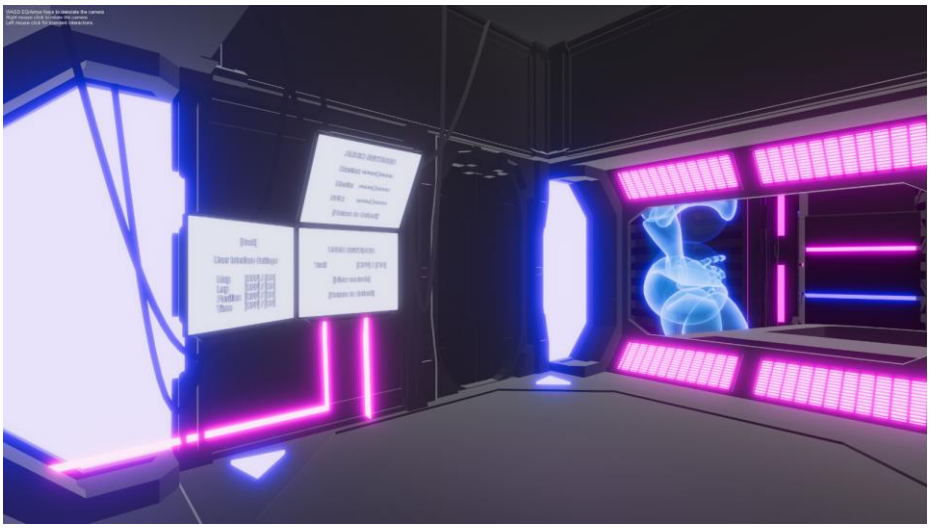


Ilustración 29: Habitación del jugador en la primera versión de Neon Hat. Fuente: Propia.

Esta versión de *Neon Hat* duró hasta que, en mayo, aplicamos para entrar al programa *Corporate* (LANZADERA EMPRENDEDORES, S.L.U., 2019-2021).

Tras varios intercambios de e-mails y reuniones con *Lanzadera*, entramos definitivamente al programa e íbamos a desarrollar *Neon Hat* para *PlayStation 4* (Sony Interactive Entertainment, 2013). Entonces decidimos ampliar el equipo para hacer este trabajo posible, añadiendo a todos los demás miembros del equipo que han desarrollado el juego y a nuestros colaboradores externos.

Al quedar enfocado ahora a ser un videojuego para *PlayStation VR* (Sony Interactive Entertainment, 2016), empezamos a pensar en soluciones para las limitaciones de movimiento que esto supuso (principalmente al girar el jugador) mientras se comenzaba a diseñar el juego, esta vez con más contenido y siendo un juego bastante más complejo y grande que la idea original. Ahora sí, estábamos empezando a desarrollar *NeonHat* (Entalto Games S.L., 2021), y durante este primer sprint me centraría en las mejoras del sistema de vuelo y el desarrollo de un primer sistema de audio.

6.1.1. Primeros pasos: movimiento y base

En esta tarea integramos mejoras como el derrape al sistema de vuelo. El jugador podrá acelerar con el gatillo derecho, estabilizarse en el espacio con el gatillo izquierdo, y derrapar con el botón X; tanto con los controladores *Move* (Sony Interactive Entertainment, 2010) como con el controlador *DualShock 4* (Sony Interactive Entertainment, 2013).

Desde el principio hemos querido que el juego tenga cada mecánica en una clase separada para mayor limpieza y para poder realizar las modificaciones necesarias de forma separada.

El movimiento base del jugador estará repartido, por tanto, en tres clases: *FlySystem, Stabilizer,* y *Skidding*.

FlySystem se encarga de acelerar al jugador y frenarlo cuando sea necesario. En cada *frame*, se comprueban los inputs y se realizarán diferentes acciones dependiendo de si se está pulsando el gatillo de aceleración o no.

En caso de que se esté pulsando, se desactivará la gravedad y se comprobará la dirección de los propulsores, que luego se aplicará como fuerza multiplicada por la intensidad de la aceleración (que depende a su vez de cuánto se pulsen los gatillos) al *Rigidbody* del jugador. *Rigidbody* es una clase del espacio de nombres *UnityEngine* que se encarga de la gestión de las físicas en el motor. Cuando no se esté acelerando, se reactivará la gravedad.

Esta clase también tiene un método público llamado "*public void Boost()*" que se llamará externamente cuando se quiera aplicar un impulso al jugador. Este método modificará tanto la velocidad como la velocidad máxima del jugador temporalmente.

La clase *Stabilizer* se encarga de la estabilización del jugador en el aire. Cuando se pulse el gatillo del estabilizador, se decrementará la velocidad en el tiempo hasta los 10 metros por segundo, que será el momento en el que se estabilice al jugador y se dé el valor de 0 a la velocidad.

Skidding es la clase relacionada con el movimiento que más ha costado decidir como definitiva, ya que las dos clases previas están basadas en el trabajo previo de S. Jimeno (Jimeno Navarro, 2020) pero esta parte de cero. Esta clase se encarga del giro del jugador en el espacio de mundo virtual y el derrape cuando se esté girando mientras se acelere.

En cada fotograma, lo primero que se hace desde Skidding es comprobar si se está pulsando el botón de rotación. En caso de que se esté haciendo, se llamará al método "*private void RotateDrone()*". Este método comprueba el ángulo actual del controlador (que podrá ser el *DualShock 4* (Sony Interactive Entertainment, 2013) o los *Move Controllers* (Sony Interactive Entertainment, 2010)) y se asegura de que el ángulo que se están girando es superior al umbral especificado. Si se supera el umbral, se ajusta a un máximo y un mínimo y se comprueba si se está pulsando el gatillo de aceleración o no. Si no se está pulsando, simplemente se rotará al jugador en el sitio, pero si se está pulsando, se rotará al jugador en un ángulo diferente y se llamará a "*private void ControlSkid()*", que se encargará de controlar el derrape.

El derrape tendrá un número variable de fases para que desde el departamento de Diseño puedan probar diferentes fases hasta quedar contentos. En "*ControlSkid()*" se comprueba en qué fase está actualmente el jugador y se cambia de fase si ha pasado el tiempo suficiente. Después, se cambia la velocidad del jugador en función de la dirección a la que esté mirando (la cual, como hemos girado antes de llamar al método, estará actualizada). También se irá frenando la velocidad con el tiempo para evitar que el jugador lo tenga activado constantemente sin ningún tipo de repercusión.

Las diferentes fases del derrape están definidas en la clase *SkidPhase*. Esta clase guardará información como el nombre de la fase (para facilitar el trabajo a Diseño), la duración, si tiene un impulso de velocidad al acabar o no, y; si tiene el impulso de velocidad; la nueva velocidad máxima mientras dure el impulso. Con este sistema, Diseño podrá añadir y quitar fases del derrape cuando quieran.

Cuando se deje de derrapar parando de pulsar el input de giro, se llama al método "p*rivate void FinishSkid()*". Este método comprobará si la fase del derrape en la que nos encontramos tiene impulso y si la velocidad actual del jugador es suficientemente alta como para superar un umbral dado (para, de nuevo, castigar el no parar de pulsar el derrape). Si se da el caso de que la fase del derrape tiene impulso y la velocidad del jugador está por encima del umbral, se llamará al método "*private void BoostSkid()*". En caso opuesto, se llamará al método "*private void CutSkid()*".

"*CutSkid()*" devolverá a sus valores iniciales a todas las variables relacionadas con el derrape, dejándolo preparado para el siguiente derrape. "*BoostSkid()*", por otro lado, se encargará de dar un impulso al jugador deteniendo durante unas centésimas de segundo su velocidad para después lanzarlo impulsado con el método público "*Boost()*" de *FlySystem*, a la que se le especificará la nueva velocidad máxima temporal. Una vez hecho eso, llamará también a la función "*CutSkid()*".

El código del movimiento del jugador hasta el final del desarrollo es el aquí descrito, aunque para dar unos valores correctos se requirieron muchas horas de prueba.

6.1.2. Sistema de Audio

Uno de los primeros sistemas necesarios era un sistema de audio que se encargase de la gestión de efectos de sonido, música, y ambiente, debido a que al ser un juego de Realidad Virtual la inmersión es algo esencial.

Para las primeras versiones del juego programé un sistema de audio para *Unity* (Unity Technologies, 2005-2021) bastante completo.

Este sistema de audio se basa en dos clases: *AudioManager* y *MusicManager*, siendo *AudioManager* autosuficiente y *MusicManager* sirviendo únicamente para facilitar el cambio y división de la música.

AudioManager es capaz de ayudar en diferentes tareas relacionadas con el sonido, como las siguientes:

- Controlar los volúmenes de música, sonido, y general.
- Aleatorizar efectos de sonido.
- Hacer sonar efectos de sonido de diferentes índoles (no interrumpibles, continuos, tridimensionales, bidimensionales, etc.).
- Controlar la música que está sonando, e incluso hacer sonar música concurrente.
- Crear una cola de música.
- Fundidos de música y efectos de sonido.
- Crear subclips de sonido desde una canción, pudiendo dividirla en varias partes para hacer bucles diferentes o darle otro uso.

MusicManager permitirá elegir qué canciones sonarán, cuántas partes tienen y cuál es la duración de cada una, podrá cambiar de fragmento en una canción o de canción con una simple llamada a un método, etc. Su cometido es realizar acciones complejas relacionadas con la música del juego de forma sencilla en diferentes métodos que actuarán con *AudioManager*.

Ilustración 30: Opciones de AudioManager en el editor de Unity. Fuente: Propia.

6.1.3. Final de hito y sprint review

Una vez teníamos funcionales la base mecánica del juego, el movimiento, y el sistema de audio, llegó el momento de cumplir con el primer hito. Necesitábamos una versión jugable de una parte del juego para grabarlo y enseñárselo a nuestros encargados dentro del programa *Corporate* (LANZADERA EMPRENDEDORES, S.L.U., 2019-2021).

Mientras el equipo de diseño preparaba el nivel de demostración utilizando las ya programadas mecánicas de movimiento y sistema de audio, yo investigué acerca de optimización y gestión de recursos para juegos de VR y para juegos de mundo abierto (ya que era la idea para *NeonHat* a estas alturas del desarrollo) teniendo en cuenta que íbamos a trabajar con una consola del año 2013. Para ello investigué cómo teníamos que enfocar el desarrollo del juego desde el principio en todos los departamentos y preparé una presentación en la que hablé sobre los siguientes temas:

- Arte: animaciones, materiales y texturas, LOD[11], modelado, cartelería y pantallas en el juego.
- Gráficos: cómo analizar el rendimiento, post-procesado de Unity en consolas, vértices y *shaders* en cada *frame* para consola, uso de múltiples núcleos para gráficos de escenas grandes en consola, modificación de sombras, qué *Rendering Pipeline* emplear.
- Diseño: Jerarquía en las escenas, UI y lienzos, luces y niebla, reutilización de otros objetos de la escena para evitar instanciación o carga de los mismos durante la ejecución, y administración de memoria con 6 normas a seguir:
    - Dividir el terreno en varios terrenos de menor tamaño.
    - Uso de LODs.
    - No tener demasiados objetos.

11 *Level Of Detail*: Es la cantidad de niveles de detalle visibles basándose en la distancia que hay entre la cámara y el objeto.

  - Uso de objetos estáticos y batching.
  - Uso de *Occlusion Culling* [12].
  - Uso de la menor cantidad de objetos diferentes en el mapa posible.
- Programación: Instanciación de objetos en la raíz de la escena, código de objetos animados, consejos diversos de optimización, administración de la memoria con reutilización de objetos y uso manual del *Garbage Collector*[13], comprobación constante de rendimiento y *profiling*[14].

El resultado final de la demostración fue satisfactorio, acabando el *sprint* tal y como estaba planeado.

6.2. Vertical Slice (Sprint 2)

Una vez la primera demostración estaba presentada, teníamos que pensar en preparar el *Vertical Slice*. Este sí sería un nivel que tendríamos que enviar como demostración, y debería tener arte y sonido, además de sistemas más finales y mecánicas más pulidas.

Lo primero en lo que trabajé fue en el sistema de navegación. Este sistema está dividido entre el sistema de navegación real y la representación que se le mostrará al jugador y tuvo dos versiones: una basada en puntos de ruta definidos por el diseñador y otra automática funcionando mediante una malla tridimensional.

Después desarrollé un sistema de controles que nos permitiera manejar diferentes tipos de entrada (para poder utilizar indistintamente un controlador u otro), el sistema de guardado y carga de la partida, y el sistema de progresión y economía.

6.2.1. Sistema de Navegación, versión 1

El sistema de navegación se compone principalmente de:

- Un sistema de puntos de ruta definidos en el espacio e interconectados entre los que tengan acceso directo entre sí.

12 No renderizado de los objetos que estén ocluidos por otros objetos (Unity Technologies, 2020).

13 El *Garbage Collector* se encarga de administrar automáticamente la memoria, liberando los objetos que ya no estén en uso ni vayan a volver a ser utilizados.

14 El *Profiler* es una herramienta que permite conseguir información sobre el rendimiento de la aplicación. (Unity Technologies, 2020).

- Un sistema de búsqueda de ruta (o *pathfinding*) utilizando el algoritmo de búsqueda "A*"[15] desde otro hilo[16].
- Un sistema de seguimiento del jugador que sepa entre qué puntos de ruta se encuentra, si ha pasado del punto de ruta actual, ha retrocedido, o ha salido de la ruta y hay que recalcular.
- Varios sistemas de guía que se verán brevemente a continuación.

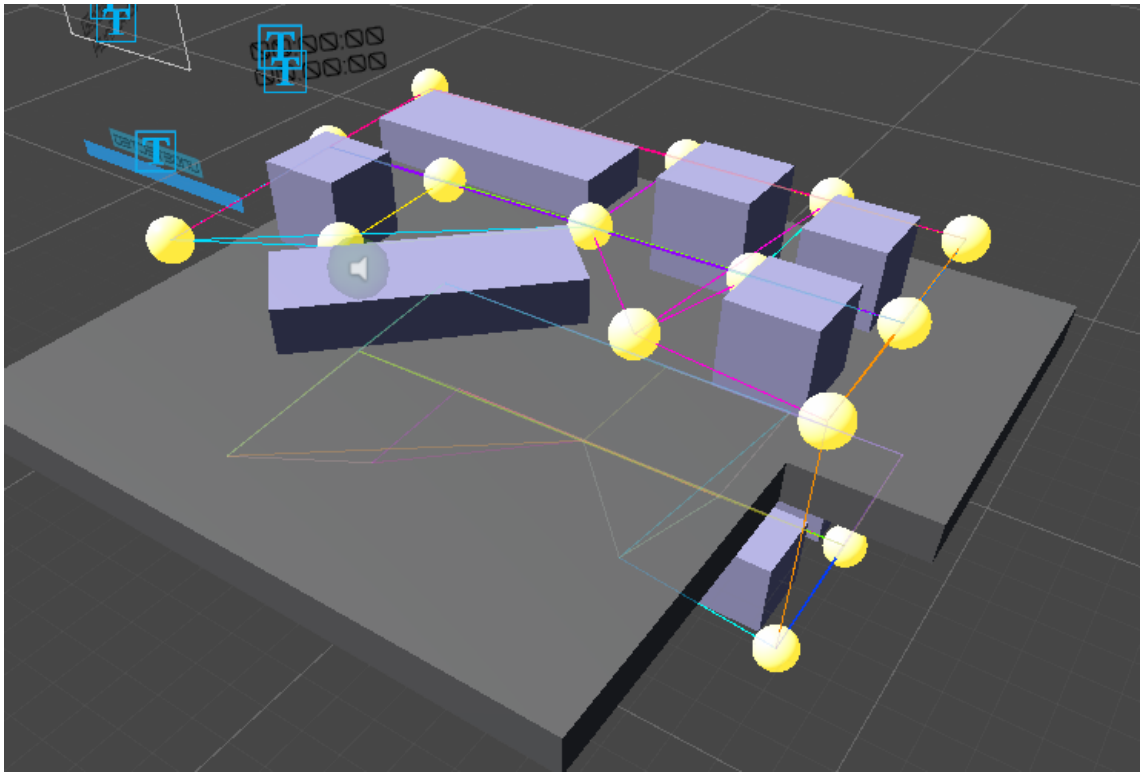


Ilustración 31: Ejemplo del sistema de puntos de ruta interconectados utilizado en el sistema de navegación de la primera versión de NeonHat.

El primero de los tres sistemas de guía es una brújula tridimensional que estará en todo momento en el brazo izquierdo del jugador. Esta brújula estará formada por una esfera en cuyo interior se encuentran los cuatro puntos cardinales, los tres ejes cartesianos, y la dirección del movimiento del jugador representada con una flecha. La dirección hacia el destino quedará representada con una flecha en el exterior de la cápsula. También tendrá diferentes colores en la esfera

15 A* es un algoritmo de búsqueda que comienza en un nodo específico de una red de nodos para encontrar otro nodo dado con el menor coste posible entre ellos (por ejemplo, menor distancia).

16 Un hilo, en contexto de computación, se refiere a una tarea (o subproceso) que puede ser ejecutada al mismo tiempo que otra tarea, compartiendo recursos del proceso que lo crea. Pueden ser utilizados, por ejemplo, para hacer tareas en segundo plano o para procesar tareas de forma asíncrona.

dependiendo de si el jugador está fuera de ruta, sobre el objetivo, debajo del objetivo, o a la misma altura que el objetivo.

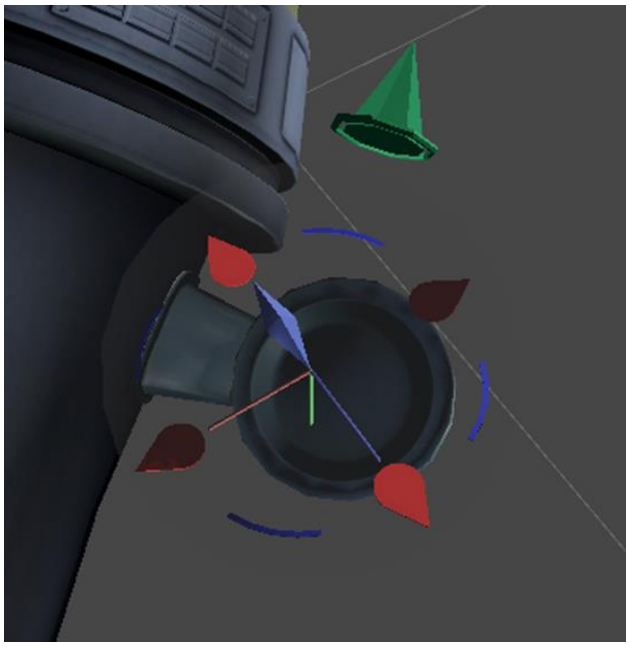


Ilustración 32: Brújula tridimensional utilizada en el sistema de navegación de la primera versión de NeonHat. Fuente: Propia.

También se cuenta con un sistema de flechas que aparecen en el punto de ruta al que el jugador se dirige apuntando hacia el siguiente. Al ser un juego con gran velocidad en muchas ocasiones, es importante mostrar al jugador hacia dónde tendrá que girar al llegar al punto al que está yendo.

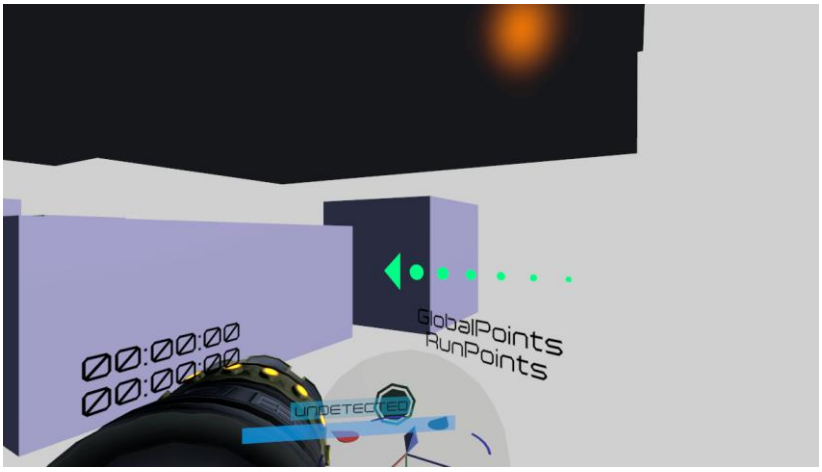


Ilustración 33: Flecha de guía e indicador de dirección directa hacia el objetivo (en naranja) utilizados en el sistema de navegación de la primera versión de NeonHat. Fuente: Propia.

Finalmente, el jugador podrá guiarse también con un indicador que muestra la dirección directa hacia el objetivo cuando este no se encuentre en el campo de visión (como se puede ver en la Ilustración 33) y, si se encuentra en el campo de visión, muestra su posición exacta y los metros de distancia (como se puede ver en la Ilustración 34).

Ilustración 34: Indicador de dirección directa hacia el objetivo y distancia hasta el mismo utilizado en el sistema de navegación de la primera versión de NeonHat. Fuente: Propia.

Las clases involucradas en este sistema de navegación son las siguientes:

- *Waypoint*: a esta clase pertenecerá cada uno de los puntos de ruta que se guardarán para el sistema de navegación. En cada instancia de *Waypoint* se guarda información como la posición en el espacio de mundo, el índice de la instancia en el *array* con todos los puntos de ruta, los índices de todas las instancias vecinas (los puntos a los que se puede acceder directamente desde el actual), y el coste del punto para el algoritmo de búsqueda de ruta (o *pathfinding)*. Este último parámetro se podrá utilizar para que, por ejemplo, tengan mayor coste los puntos de ruta de una zona peligrosa que los de una zona segura y se evite esa ruta de ser posible.

- *WaypointsGenerator*: Esta clase será la encargada de generar las instancias de *Waypoint* a partir de las posiciones dadas en las que se encontrará cada punto de ruta en el espacio de mundo. También se encargará de generar los vecinos de cada punto de ruta y establecerlos en cada uno. Esto lo hará recorriendo el array de puntos en el espacio de

mundo desde cada punto de ruta y comprobando si se puede acceder en línea recta hasta cada uno utilizando el método estático *public static bool Raycast()* de la clase *Physics* del espacio de nombres[17] *UnityEngine*, el cual devolverá un valor *true* si se ha colisionado con algo entre los dos puntos o un valor *false* si no se ha colisionado. En este caso, si este método devuelve *false*, los dos puntos serán vecinos.

- *Astar*: Esta clase será la encargada de calcular la ruta óptima entre todos los puntos de ruta desde una posición en el mundo hasta otra utilizando el algoritmo de búsqueda "A*". Funcionará desde un hilo aparte en segundo plano con el fin de no bloquear la ejecución del juego durante el cálculo.

- *Navigation*: Esta clase será la principal en el sistema de navegación y la que llevará el control del resto de clases involucradas en dicho sistema. Guardará una referencia a una instancia de *Astar* y al resto de instancias de las otras clases. Cuando se llame a su función *public void NewDestination()* indicando una nueva posición de destino, comenzará a funcionar. Primero pondrá a la instancia de *Astar* a calcular la nueva ruta. Cuando la ruta esté lista, los 3 tipos de indicadores se pondrán en marcha para saber hacia dónde deben indicar al jugador. Mientras tanto, se comprobará periódicamente la posición del jugador dentro de la ruta y si hay que realizar alguna acción al respecto.
  Si el jugador se ha salido de la ruta, se recalculará una nueva ruta. Esto se determinará calculando la distancia respecto a los dos puntos de ruta entre los que el jugador se encuentre. Si se aleja más de una distancia indicada, se considerará que se ha salido de la ruta.
  Si, por otro lado; y siendo el punto de ruta actual el punto hacia el que el jugador se estaba dirigiendo; tiene menos distancia respecto al punto de ruta siguiente al actual desde su posición, que la que hay desde la posición del punto de ruta actual hasta el siguiente, se determinará que el jugador ha pasado el punto de ruta actual, por lo que el punto de ruta actual pasará a ser el siguiente.
  Si, por el contrario, está más lejos del punto de ruta actual que este punto del anterior, se determinará que el jugador ha retrocedido en la ruta, por lo que el punto actual pasará a ser el anterior.

17 "La palabra clave *namespace* se usa para declarar un ámbito que contiene un conjunto de objetos relacionados. Puede usar un espacio de nombres para organizar los elementos de código y crear tipos únicos globales." (Microsoft Corporation, 2015)

- *CompassController*: Esta clase controla la brújula tridimensional del jugador y se encarga de mover los indicadores de la brújula en tiempo real y de cambiar los colores de la esfera.

- *ObjectiveArrow*: Esta clase controla los estados de la flecha indicadora, que aparecerá en el espacio de mundo indicando con su posición el punto de ruta actual y con su dirección el punto de ruta siguiente (o el objetivo final, de no haber más puntos de ruta hasta el mismo). Estos estados se utilizarán para que la flecha aparezca y desaparezca de forma progresiva en los diferentes puntos.

- *HelmetUIManager*: Esta clase actúa como mánager para diversas indicaciones para el jugador, ya que controla toda la interfaz gráfica de usuario con cosas como la puntuación o los puntos de vida restantes. En lo relativo a la navegación, sólo importa su método *public void UpdateCurrentObjective()*. Este método calculará el plano en el que se proyecta la interfaz y calculará la proyección de la posición del objetivo y la posición de las esquinas del *frustum* de la cámara en este plano. Con esto se comprobará si el objetivo se encuentra dentro del *frustum* y, de ser así, aparecerá en la posición de la proyección en el plano un indicador con la distancia en metros hasta el objetivo. Si, por otro lado, el objetivo se encuentra fuera del *frustum*, aparecerá un indicador en la posición más cercana a la real dentro del *frustum*, dando a entender que hacia esa dirección se encuentra el objetivo. Esta nueva posición será calculada mediante la intersección de líneas coplanares entre los cuatro bordes de la proyección del *frustum* y la dirección desde la proyección del centro del *frustum* en el plano hacia la proyección del objetivo. La intersección más cercana será la posición en la que se situará el indicador.

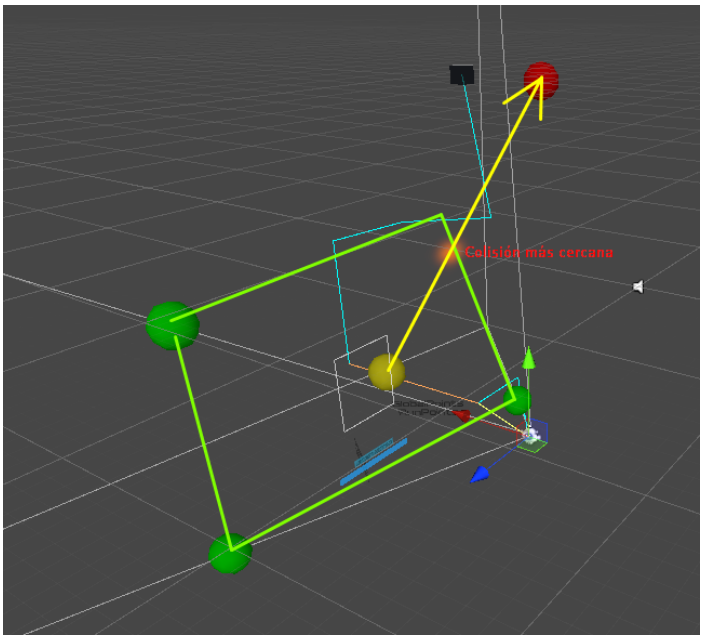


Ilustración 35: Representación del funcionamiento del método UpdateCurrentObjective de la clase HelmetUIManager de la primera versión de NeonHat. Las esferas verdes representan las proyecciones de las esquinas del frustum en el plano de la GUI del jugador, la esfera amarilla representa la proyección del centro del frustum en el mismo plano, y la esfera roja representa la proyección del destino en el mismo plano. Fuente: Propia.

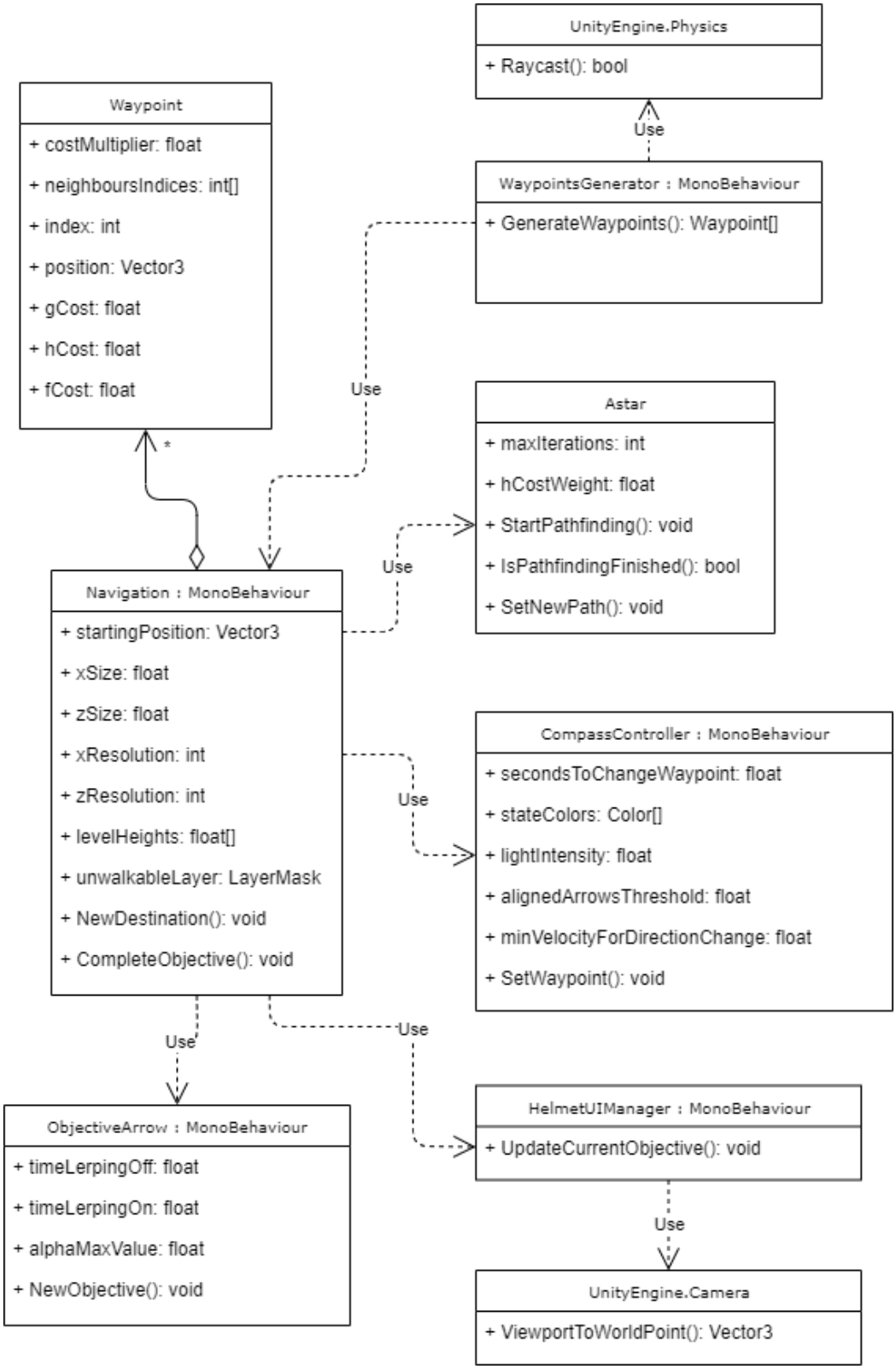


Ilustración 36: Relación entre clases implicadas en la primera versión del sistema de navegación de la primera versión de NeonHat. Fuente: Propia.

6.2.2. Sistema de Navegación, versión 2

Para esta segunda versión que realicé del sistema de navegación decidí utilizar una malla tridimensional de puntos de ruta generados con unos parámetros dados en vez de las posiciones de los puntos de ruta predefinidas. Este cambio requirió una modificación importante en la funcionalidad de varias clases, aunque desde el punto de vista del jugador el sistema continuó funcionando de forma similar.

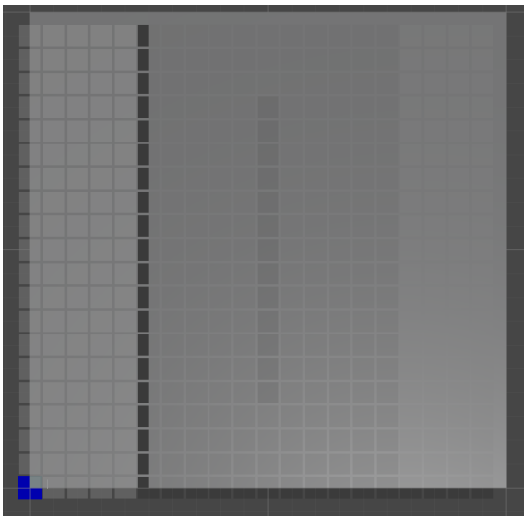


Ilustración 37: Ejemplo de la malla tridimensional utilizada en la segunda versión del sistema de navegación de la primera versión de Neonhat vista desde arriba. Fuente: Propia.

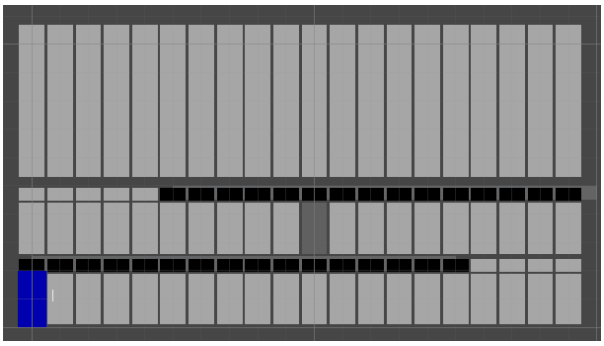


Ilustración 38: Ejemplo de la malla tridimensional utilizada en la segunda versión del sistema de navegación de la primera versión de Neonhat vista desde el frente. Fuente: Propia.

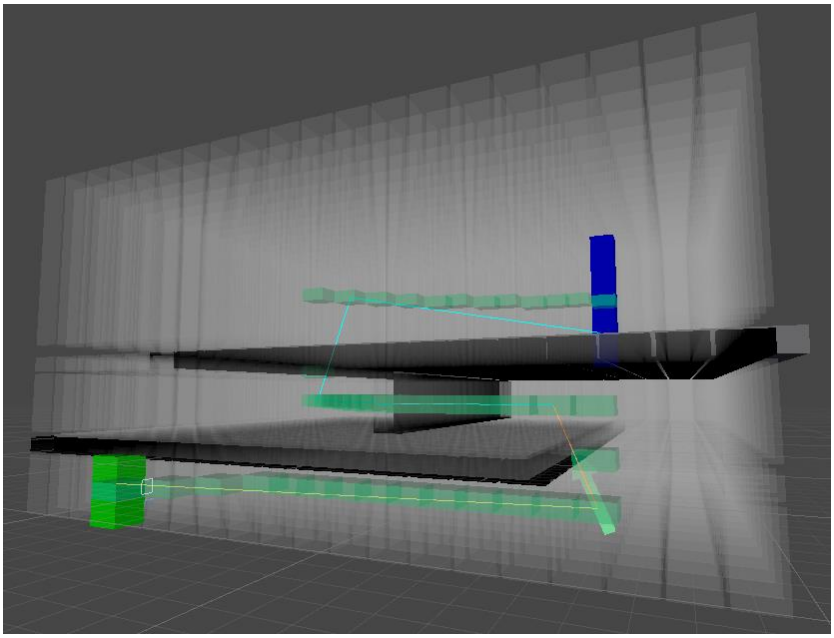


Ilustración 39: Visualización tridimensional de un ejemplo de la malla tridimensional utilizada en la segunda versión del sistema de navegación de la primera versión de NeonHat. Fuente: Propia.

Las clases involucradas y sus respectivas modificaciones para esta segunda versión del sistema de navegación son las siguientes:

- *Waypoint*: En la anterior versión se escogían a mano todas las posiciones en las que se crearían puntos de ruta, considerando así que todos los que existieran eran transitables, pero en esta versión ese ya no es el caso. Al generarse automáticamente en una malla tridimensional habrá que comprobar si los puntos son transitables o no. Por ello, ahora hay un nuevo atributo en esta clase indicando si lo es con un valor booleano. El índice guardado ahora tampoco será un entero, sino tres: el índice en X, en Y, y en Z. El resto de los valores guardados seguirán siendo igual que antes; incluyendo el índice de los vecinos. Esto se debe a que los puntos de ruta se guardan en un *array*, y no en una matriz ni en un array multidimensional. Por esto, la razón de que se guarde el índice propio en X, Y, y Z no es únicamente la localización en el *array*; ya que para eso sería más sencillo con un solo entero; sino que se debe a que se necesitan estos valores al utilizar una versión tridimensional del algoritmo de Bresenham (Bresenham, 1965), del cual se hablará más adelante.

- *WaypointsGenerator*: Esta clase ahora sólo alberga el método estático *public static Waypoint[] GenerateWaypoints()*, al que se le dará una posición inicial, un tamaño para

la malla en X y otro en Z, la resolución de la malla en X y en Z, un *array* con las alturas de los pisos que haya en el nivel, y la *LayerMask*[18] de objetos no transitables.

Este método devuelve un *array* del tamaño de la multiplicación de las dos resoluciones X y Z, y el número de alturas que hay (la longitud del *array* de alturas). La decisión de tratar a las alturas de forma diferente a la X y la Z es para ahorrar en memoria y en procesamiento, ya que nunca se daría el caso de necesitar más de un punto de ruta por nivel de altura en esta primera versión de *NeonHat*.

A la hora de comprobar si cada punto de ruta es transitable o no, se utiliza el método estático *public static bool CheckSphere()* de la clase *Physics* del espacio de nombres *UnityEngine*, que devolverá *true* si detecta algo dentro de una esfera de posición y radio dados y con una *LayerMask* a comprobar dada. Si devuelve *true*, significa que hay algo en ese espacio, por lo que no será transitable. Tras esto se guardarán los ocho vecinos de la misma altura y los dos vecinos de arriba y abajo.

- *Astar*: Esta clase conserva el código que tenía para la anterior versión con ciertos añadidos. La diferencia principal se debe a que antes no se necesitaba suavizar las rutas tras calcularlas, ya que el resultado de utilizar el algoritmo "A*" con los puntos de ruta de la versión anterior siempre sería el camino óptimo por cómo se guardaban los vecinos de cada punto. Esto ya no ocurre por el tipo de puntos de ruta que utilizaremos ahora, como se puede ver en la Ilustración 40. Para solucionar esto suavizando la ruta en una malla bidimensional, se puede aplicar el algoritmo de Bresenham (Bresenham, 1965); sin embargo, aquí estamos hablando de una malla tridimensional, por lo que he tenido que utilizar una versión tridimensional del algoritmo de Bresenham (Proyectos Robóticos, 2010). Cuando la búsqueda y el suavizado acaban, se devuelve tanto la ruta suavizada como la ruta sin suavizar, ya que se utilizarán ambas para tareas distintas.

18 Un *GameObject* puede utilizar hasta 32 capas de tipo *LayerMasks* soportadas por Unity. Las primeras 8 de estas capas son especificadas por Unity. Se utilizan máscaras de bits que representan las 32 capas definiéndolas como *true* o como *false*. Cada máscara de bits describe si una capa se está utilizando. Por ejemplo, si el bit 5 de una *LayerMask* es igual a 1 (*true*), se entenderá que el objeto que utilice esa *LayerMask* tendrá la configuración de "Agua" del motor. (Unity Technologies, 2021)

Ilustración 40: Ejemplo de las dos rutas proporcionadas por la clase Astar en la segunda versión del sistema de navegación de la primera versión de NeonHat. En verde, la ruta original fruto del uso del algoritmo de búsqueda A. En azul y naranja, la ruta suavizada mediante el uso de la variación tridimensional del algoritmo de trazado de líneas de Bresenham. Fuente: Propia.*

- *Navigation*: En la nueva versión de sistema de navegación, esta clase tiene nuevos parámetros para los diseñadores. En estos parámetros se podrá indicar todo lo relativo al tamaño de la malla tridimensional. La única otra diferencia se encuentra en el seguimiento del jugador. Ahora, para saber si el jugador ha pasado un punto o ha retrocedido se utilizan los puntos de ruta de la versión suavizada con Bresenham 3D (Proyectos Robóticos, 2010), mientras que para evaluar si el jugador se ha salido de la ruta y hay que recalcular se utilizará la versión sin suavizar. Se considerará que se ha salido de la ruta cuando no se encuentre en ningún punto de ruta vecino del punto de ruta en el que se encontraba anteriormente (o en el mismo).

- *CompassController*: Esta clase continúa funcionando igual.

- *ObjectiveArrow*: El único cambio con respecto a la versión anterior es que ahora la flecha del espacio de mundo sigue al jugador en el Y; es decir, siempre estará a la misma altura que el jugador. Esto hará que sea más fácil de ver y seguir para el jugador sin importar la altura de cada piso o nivel.

- *HelmetUIManager*: Tras ver que en la versión anterior se volvía molesto tener un indicador todo el rato en el borde de la visión en Realidad Virtual, decidimos dejar únicamente el indicador con los metros de distancia sobre el objetivo.

Ilustración 41: Indicador de distancia hasta el objetivo de la segunda versión del sistema de navegación de la primera versión de NeonHat. Fuente: Propia.

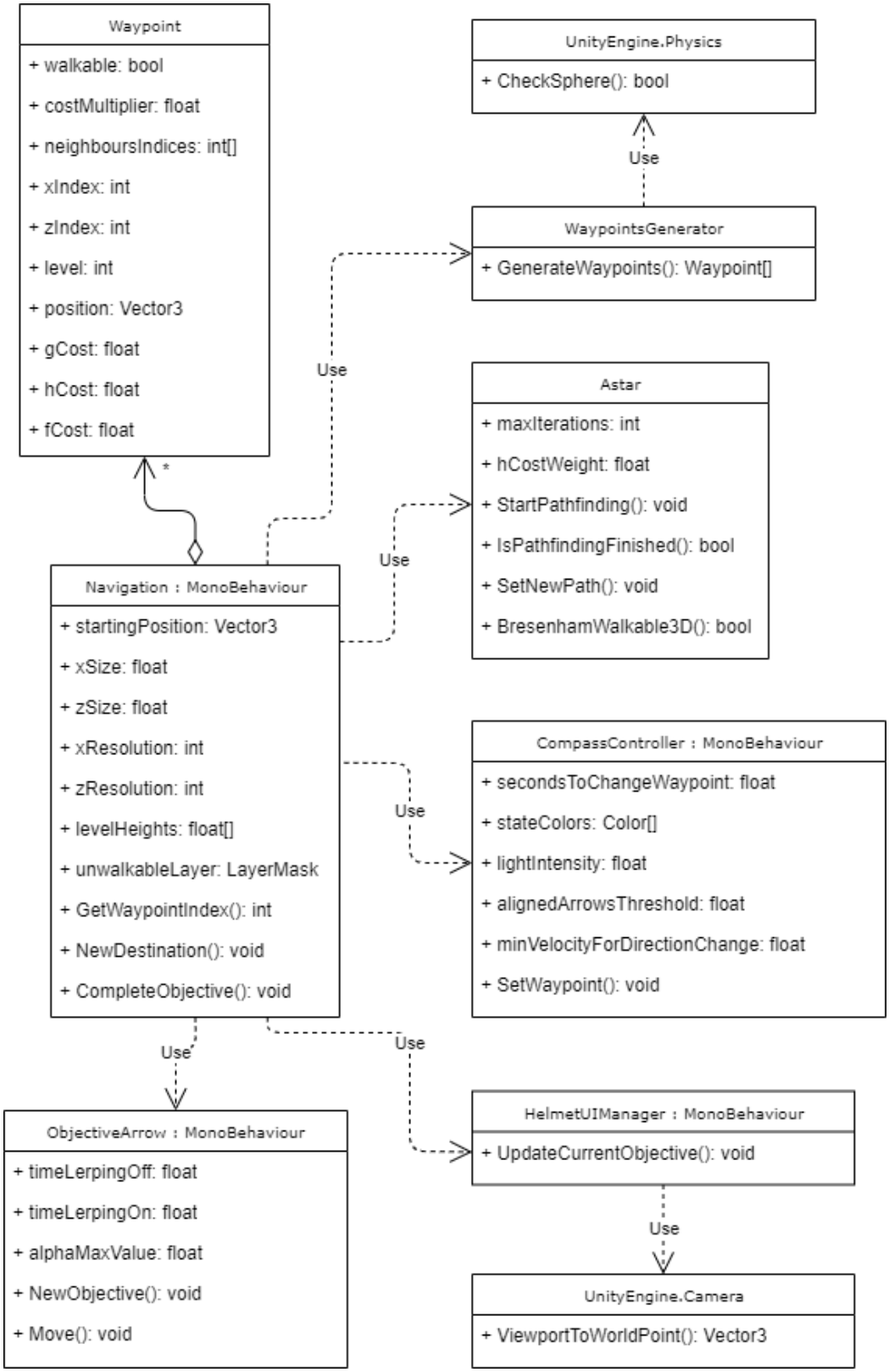


Ilustración 42: Relación entre clases implicadas en la segunda versión del sistema de navegación de la primera versión de NeonHat. Fuente: Propia.

6.2.3. Controles

Para comprobar el tipo de input de forma externa y uniforme para todas las clases que necesitan input, como el movimiento y el estabilizador, escribí la clase *DSMInputSystem*, que es un *singleton*[19] con directivas de compilación condicional[20] y referencias a todos los tipos de input programados: *DualShock 4* (Sony Interactive Entertainment, 2013) y *Move Controllers* (Sony Interactive Entertainment, 2010) para la directiva de compilación condicional "#if UNITY_PS4"; y *SteamVR* (Valve, 2015-2021) en caso contrario, para pruebas desde PC durante el desarrollo. Esta clase también se encargará de la retroalimentación háptica, como la vibración de los mandos, cuando se llame como corrutina[21] a su método "*public IEnumerator ControllerHapticPulse()*".

6.2.4. Sistema de guardado y progresión

Para contener toda la información de la partida del jugador escribí la clase *AdditionsManager* con el patrón de diseño *singleton*. Esta clase guardará todos los niveles jugables dividiéndolos entre niveles bloqueados, adquiribles, y accesibles. Tendrá también la descripción y datos de todos los niveles y el dinero actual del jugador. La información de cada nivel jugable se guardará en una instancia de la clase *Level*, que tendrá el nombre, precio, número identificador, puntuación máxima local, y dos valores booleanos para saber si está bloqueado y si es accesible (es decir, está adquirido). Esta distinción viene porque los niveles estaban bloqueados hasta cumplir con ciertos requisitos. En ese momento se desbloqueaban, pero habría que comprarlos, por lo que hasta entonces estarían marcados como adquiribles, pero no como accesibles.

Para guardar el progreso del jugador en memoria y cargarlo posteriormente, escribí la clase *SaveLoad*, que guardaría y cargaría desde un hilo aparte y está diseñada también con el patrón *singleton*. Esta clase tendrá un *array* de instancias de *SlotInformation*, que tendrá toda la información del jugador actual; un entero para saber cuál se está utilizando en este momento; tres booleanos de control para saber cuándo se ha comenzado a utilizar una de las instancias de *SlotInformation*, cuándo se ha guardado y cuándo se ha cargado; una referencia al hilo (clase

19 S*ingleton* o instancia única es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella (Gupta, 2012).

20 "Cuando el compilador de C# encuentra una directiva #if, seguida en última instancia por una directiva #endif, compila el código entre las directivas solo si se ha definido el símbolo especificado" (Wagner & "olprod", 2021).

21 "Una corrutina es una función que tiene la habilidad de pausar su ejecución y devolver el control a Unity para luego continuar donde lo dejó en el siguiente *frame*" (Unity Technologies, 2016).

Thread del espacio de nombres *System.Threading*) que se utilizaría para cargar y guardar; y una referencia a la instancia de *AdditionsManager*, con el que estaría intercambiando información en cada carga o guardado. En este intercambio de información se utilizaría a la instancia de *SaveLoad* únicamente como intermediaria entre la instancia de *AdditionsManager* y la instancia del *array* de *SlotInformation* en la posición indicada por el "entero para saber cuál se está utilizando en este momento". Este sistema permite que varios jugadores puedan tener su propio progreso y partida en el juego con una sola copia de este comprada.

6.2.5. Final de hito y sprint review

Finalmente, con toda la funcionalidad previa sumada a todos estos nuevos sistemas permitió el funcionamiento del *Vertical Slice*. Este *Vertical Slice* contaba con un menú parcialmente funcional (con diferentes espacios de guardado y selección del único nivel disponible) y un nivel en el que se enseñaban los controles al jugador y se le pedía que entregara un paquete en una localización del distrito de la ciudad en el que se desarrolla la acción. Tras entregarlo, descubrimos que era una bomba y, tras la explosión del edificio, comienza una persecución a toda velocidad de la policía tras nosotros.

Tras finalizar el desarrollo de este *Vertical Slice* tuvimos una reunión en la que llegamos a la conclusión de que no íbamos a poder cumplir fechas con un producto tan complejo, principalmente por falta de personal en el departamento de Arte. Comenzamos a iterar sobre la idea original y a buscar las partes esenciales del videojuego que queríamos desarrollar: ¿qué lo hacía divertido?

Después de unos días de deliberación, decidimos que lo importante del juego era la sensación de velocidad y el gran control en el vuelo, por lo que comenzamos a hacer un juego arcade de carreras, con mucha menos carga en el departamento de Arte y de un tamaño más acorde a lo que una empresa independiente de alumnos recién egresados podía intentar hacer como primer título y con unos estándares de calidad más realistas.

6.3. Demo de juego final (Sprint 3)

De repente nos habíamos visto con un calendario bastante más apretado, ya que teníamos que tirar todo el diseño de niveles, bastante diseño de sistemas, la programación de varios sistemas (como el de navegación), y la mayor parte del arte, ya que íbamos a hacer un juego totalmente diferente en el que mantendríamos únicamente lo relativo al movimiento.

Durante este *sprint* nos centramos en planear el nuevo videojuego, salvar ciertos sistemas de la versión anterior, y crear menús para acabar con un nuevo *Vertical Slice* que nos permitiera ver la viabilidad del nuevo *NeonHat*.

6.3.1. El nuevo NeonHat

Decidimos hacer un juego de carreras con una estética sencilla y nada realista: Internet.

Aún con este cambio, habíamos perdido varios meses de trabajo, por lo que para llegar a las fechas acordadas con *PlayStation* España y Lanzadera decidimos que tendríamos que crear más horas de juego ahorrando todo lo posible en todos los departamentos.

Una forma era añadir dificultades, pero la gente suele jugar en una sola dificultad los juegos, así que esto no serviría en general. Sin embargo, nuestro juego iba a ser un juego de carreras arcade, y tenemos un gran ejemplo de este tipo de juegos: *Mario Kart* (Nintendo Company, Ltd., 1992-2020). En los títulos de esta saga suele haber tres dificultades divididas en, tomando como ejemplo el título *Mario Kart DS* (Nintendo Entertainment Analysis and Development, 2005), 50 CC[22], 100 CC, y 150 CC. Cada una de estas dificultades comparte los mismos circuitos y aumenta la velocidad de los vehículos, pero tienen sus propias copas y puntuaciones. Esto hace que los jugadores lo consideren niveles aparte y quieran lograr las mejores puntuaciones en todas las dificultades.

Decidimos basarnos en esta estrategia y, dado que nuestro juego tiene como temática "Internet", nos pareció una buena idea denominar a las tres dificultades como "Kbps", "Mbps", y "Gbps". Más tarde, repasamos estos nombres y concluimos que lo mejor sería llamar a las dificultades simplemente "Kilobyte", "Megabyte", y "Gigabyte", ya que son términos con los que la mayor parte de usuarios están más familiarizados.

En esta versión del juego, el jugador perdía puntos de vida al chocar con una pared, y al llegar a cero puntos de vida, reaparecía en el último punto de control por el que hubiera pasado. Tras probar las primeras versiones con gente externa (familiares y amigos), nos dimos cuenta de que esto cortaba bastante el flujo del juego y frustraba más que aportaba, por lo que decidimos quitar esta funcionalidad y reemplazarla por otro tipo de carrera: una cuarta dificultad llamada "*Derby*

22 Centímetros cúbicos.

Extremo". En esta dificultad se mantendría la velocidad de "Gigabyte", pero al chocar con las paredes se perderían puntos de vida. Al llegar a cero, ahora el jugador no reaparecería, sino que perdería la carrera. En contraposición, las tres dificultades originales no tendrían puntos de vida y no habría ninguna repercusión en los choques más allá de la pérdida de velocidad.

Con estas cuatro dificultades ya teníamos más tiempo de juego; pero necesitábamos algo más para llegar a las horas de juego que habíamos acordado, lo que nos llevó a crear otro tipo de modo de juego: el modo "persecución". En este modo de juego, el jugador podría disparar y perseguiría a un enemigo escudado por tres enemigos protectores. Al disparar y derribar a los tres enemigos protectores, el enemigo principal perdería el escudo temporalmente, con lo que se le podría disparar. Tras repetir esto una serie de veces, el enemigo principal acabaría derribado y el jugador acabaría la carrera.

Sorprendentemente, este nuevo modo de juego gustó mucho a todo el que lo probaba, y tras recibir la petición de poder disparar a los rivales en las carreras normales en diversas ocasiones, nos lo planteamos seriamente. Al final decidimos que los rivales tendrían un mejor nivel de juego que el que tenían en el momento y serían bastante mejores que un jugador novato, con la contrapartida de que el jugador podría dispararles y ralentizarlos temporalmente con ello.

Con esto y tras varias horas de prueba, descubrimos lo divertido que era volar y esquivar mientras se disparaba, por lo que para cumplir con el poco tiempo que nos faltaba de juego para llegar al mínimo acordado de horas decidimos implementar 5 jefes finales, de los que se hablará más adelante, quedando así la estructura final de *NeonHat* (Entalto Games S.L., 2021) cerrada.

6.3.2. Salvando y perdiendo ideas y trabajo

Cuando teníamos claro cómo iba a ser el producto final, era el momento de hacer una labor dura: ver qué sistemas se migran al nuevo proyecto y qué sistemas se quedan atrás.

El sistema de gestión de niveles y el menú se redujeron y mantuvieron. Ya no habría que desbloquear los niveles, por lo que se quitó todo el sistema de economía, pero se mantendrían los niveles como bloqueados o desbloqueados.

De igual forma, el sistema de guardado y carga se mantuvo también casi por completo, con pequeñas reducciones.

El sistema de movimiento fue lo único que se mantuvo al completo, ya que la razón principal por la que íbamos a hacer un juego de carreras era que teníamos un movimiento pulido y con buena sensación para el jugador.

Para controlar las mejoras, los paquetes que el jugador tenía en cada momento, y demás extras que el jugador podría recoger, había un sistema de inventario, recogida, y lanzamiento de objetos. Este sistema no tenía cabida en el juego, por lo que no pasó al siguiente proyecto.

Otro sistema que no pasó al siguiente proyecto tras una enorme cantidad de horas invertidas fue el sistema de navegación, ya que en un juego de carreras cerradas no tendría ninguna forma sencilla de ser introducido desde diseño.

Finalmente, el sistema de audio que hice se desechó también ya que Joaquim Camps; el músico externo con el que hemos colaborado para el desarrollo del juego; nos propuso utilizar *FMOD* (Firelight Technologies Pty Ltd, 1995-2021). Tras evaluarlo y estudiar el funcionamiento de este software con *Unity* (Unity Technologies, 2005-2021), decidimos que era una buena idea aprender a usar un motor de audio profesional y comenzamos a utilizarlo.

6.3.3. Nuevo sistema de audio

Debido a que habíamos estado utilizando el sistema de audio desarrollado por mí hasta el momento, ya teníamos bastante implementación de audio añadida al proyecto, por lo que añadir *FMOD* (Firelight Technologies Pty Ltd, 1995-2021) fue más sencillo de lo que podría haber sido. Mi acercamiento fue crear una nueva clase llamada *FMODManager*, que se encargaría de guardar todas las referencias e instancias de los diferentes sonidos e implementar todos los métodos que antes se llamaban de la otra clase de una forma lo más similar posible, con el fin de cambiar la menor cantidad de código en el resto del proyecto. Podemos ver ejemplos de esto en la Ilustración 43 y en la Ilustración 44.

```
public float MasterVolume
{
    get { return m_MasterVolume / m_InitialMasterVolume; }
    set
    {
        m_MasterVolume = Mathf.Clamp01(value) * m_InitialMasterVolume;
        m_MasterBus.setVolume(m_MasterVolume);
    }
}
```

```
private Bus m_MasterBus
in class FMODManager
```

Ilustración 43: Propiedad MasterVolume en la clase FMODManager y su ajuste del volumen en el bus máster del motor de audio FMOD. Fuente: Propia.

```
Frequently called  19 usages
public EventInstance PlayOneShotWithParameters(EventRef eventName, Vector3 position, ParameterRef[] parameterNames, float[] newValues)
{
    EventInstance currEvent = RuntimeManager.CreateInstance(m_EventGuids[(int)eventName]);
    for (int i = 0; i < parameterNames.Length; ++i)
    {
        currEvent.setParameterByID(m_EventsParameters[(int)eventName].m_ParametersIDs[(int)parameterNames[i]], newValues[i]);
    }
    currEvent.set3DAttributes(RuntimeUtils.To3DAttributes(position));
    currEvent.start();
    currEvent.release();

    return currEvent;
}
```

Ilustración 44: Método PlayOneShotWithParameters() de la clase FMODManager, encargado de lanzar un efecto de sonido que se utilice una única vez sin cortes con unos parámetros dados (como el volumen o el tono), como ejemplo del tipo de métodos que se encuentran en esta clase. Fuente: Propia.

En esta clase se encuentran dos enumeraciones[23] llamadas *EventRef* y *ParameterRef*. Estas servirán para poder indicar de forma humanamente comprensible a qué evento (canción o efecto de sonido) y a qué parámetro de un evento (volumen, tono, fase, sección, …) nos referimos en cada momento.

6.3.4. Menús

Estando todos los sistemas a punto desde programación, sólo quedaba esperar a que desde Diseño tuvieran un primer circuito con el que terminar un nuevo *Vertical Slice* que nos dejara ver si el cambio de rumbo había sido acertado. Hasta ese momento, pude adelantar trabajo de la versión final que también se añadiría en el *Vertical Slice*.

Preparé una escena que mostrara los logotipos e imágenes que quisiéramos al principio utilizando la clase *StartLogos*. Esta clase mostraría todos los logos que tuviera referenciados uno detrás de otro en un tiempo dado, con transiciones fluidas entre ellos, y; al acabar; cambiaría de escena al menú principal. El cambio de escena se realizaría con una también nueva clase llamada *SceneChange*, que se encargaría de mantener la referencia a todas las escenas y de cambiar de escena cargando la siguiente desde una corrutina de *Unity*, siempre con un fundido a negro, seguido del logotipo de *NeonHat* (indicando que se está cargando la siguiente escena), y acabando con un fundido a la escena. También durante el cambio de escena se limpiarían los efectos visuales, de sonido y las corrutinas en funcionamiento.

Se implementó también un menú de pausa, de final de partida, y se pulieron el menú de selección de jugador y de selección de nivel.

Para el menú de selección de jugador implementé un teclado funcional en Realidad Virtual, para que el jugador pudiera elegir su nombre de partida y que esta no se llamara "1", "2", o "3". Por esto, también añadí un nuevo elemento para guardar en memoria y cargar junto al resto de información: el nombre del espacio de juego.

Fue en este punto cuando también implementé respuesta visual y háptica en los mandos controladores cuando se pase por encima de un botón, ya que en el resto de los botones se notaba su falta; pero fue en el teclado cuando se volvió esencial. La respuesta visual fue un cambio de color en el láser que sale del mando para saber si estás o no encima de un botón, y un cambio en el color del botón afectado. La respuesta háptica fue solamente una pequeña

23 "Un tipo de enumeración es un tipo de valor definido por un conjunto de constantes con nombre del tipo numérico integral subyacente. Para definir un tipo de enumeración, use la palabra clave *enum* y especifique los nombres de miembros de enumeración". (Microsoft Corporation, 2019)

vibración del mando cada vez que el láser note un botón sobre el que no estaba antes. Con estos dos cambios tan simples todos los menús se volvieron mucho más agradables de navegar.

6.3.5. Cambios en el gameplay

Como he dicho previamente, uno de los menús que implementé fue el menú de pausa. Esto desembocó en que el primer cambio en el *gameplay*[24] al que me tuve que enfrentar fue implementar la pausa de forma que pare toda la ejecución menos la que nos interese.

La forma en que implementé esto fue haciendo uso de la clase *Time* (Unity Technologies, 2021) del espacio de nombres *UnityEngine.CoreModule*.

Modificando el valor *timeScale* de *Time* a 0, se puede hacer que el valor de *deltaTime* siempre dé como resultado 0; mientras que si el valor de *timeScale* era 1, el valor de *deltaTime* es el tiempo entre el anterior fotograma y el actual. También existe *unscaledDeltaTime*, que siempre tendrá como valor el tiempo entre el anterior fotograma y el actual, sin importar el valor de *timeScale*.

Utilizando en el paso del tiempo de las clases que siempre habrán de estar activas el valor de *Time.unscaledDeltaTime* me aseguro de que siempre estarán en funcionamiento; mientras que utilizando en el resto de clases *Time.deltaTime* me aseguro de que se pausarán siempre que *Time.timeScale* valga 0, por lo que será el valor al que pondré esta variable cuando quiera pausar el juego, mientras que lo devolveré a 1 cuando quiera que el flujo del juego continúe.

El resto de los cambios en el *gameplay* que implementé para esta versión de *NeonHat* fueron diferentes añadidos para una revisión del movimiento:

- Creé aros de turbo que situar a lo largo de las carreras para un aumento temporal de la velocidad, aprovechando el método *Boost()* de la clase *FlySystem*.
- Añadí movimiento en el eje Y durante los derrapes para poder derrapar mientras se sube o baja.
- Hice la aceleración en *FlySystem* dependiente de la velocidad actual. Esto se decidió tras realizar diferentes pruebas y concluir que se sentía mejor acelerar despacio cuando se tiene la velocidad actual baja, y acelerar más rápido cuanto más rápido vaya el jugador.

24 Este término hace referencia a la manera en la que el jugador interactúa con el juego o a la manera en la que el juego interactúa con el jugador. (Cervera, 2019)

6.3.6. Final de hito y sprint review

Con todos estos cambios logramos tener un *Vertical Slice* mucho más cercano a lo que sería el videojuego final. Este *Vertical Slice* contaba con todos los sistemas que hemos visto implementados, pero sólo tenía un nivel.

Esto se debe a que queríamos un único nivel que se pareciera lo máximo posible al juego final para probar la viabilidad del proyecto según el tiempo que hubiera requerido realizar el trabajo que había hasta el momento, y queríamos ver si era realmente divertido de jugar.

El *Vertical Slice* resultó ser un éxito en tiempos de desarrollo y nos gustó mucho al probarlo, por lo que decidimos realizar un primer *port*[25] a *PlayStation 4* (Sony Interactive Entertainment, 2013), para comprobar que todo funcionara correctamente.

Tras adaptar los controles a los mandos *Move* (Sony Interactive Entertainment, 2010), compilamos el juego para la consola desde *Unity* (Unity Technologies, 2005-2021) y lo ejecutamos en un *Development Kit*[26] que nos fue facilitado por *Lanzadera* y *PlayStation Talents* (LANZADERA EMPRENDEDORES, S.L.U., 2019-2021), en Valencia. Al lanzarlo en el kit, no se podía controlar y no se renderizaba correctamente ningún elemento visual, por lo que decidimos ponernos en contacto con Manuel Martínez; el director técnico en *PlayStation Talents*.

Aunque el *sprint* su hubiera completado satisfactoriamente, la consola objetivo no soportaba el videojuego, por lo que durante el próximo *sprint* habría que solucionar este problema.

6.4. Detención de producción y arreglos para *porting* (Sprint 4)

Tras ver de cerca varios problemas, Manuel Martínez nos sugirió que la opción más viable sería detener la producción para arreglar lo que había de videojuego y, desde el punto en el que tuviéramos todo lo existente arreglado y funcionando, hacer el juego enfocándonos únicamente en la plataforma final (*PlayStation 4* (Sony Interactive Entertainment, 2013)) para poder llegar a fechas sin necesidad de utilizar un mes en hacer el *porting* al final del desarrollo.

Como en ese momento no disponíamos de un *Development Kit* en Zaragoza, decidimos hacer todo lo posible por que funcionase antes de volver a Valencia para probarlo.

25 *Port* es el término utilizado cuando un videojuego diseñado para funcionar en una plataforma específica se modifica para funcionar en una plataforma diferente. (Wolf, 2008)

26 Los *Development Kits* "son máquinas específicas para compilar código en una máquina con las especificaciones finales de la consola." (Rosa Fernández, 2019)

6.4.1. Arreglando el proyecto hasta el punto actual

Nuestro actual proyecto se cimentaba sobre todo el trabajo anterior, lo que generaba parte de los problemas que nos surgieron. Debido a este motivo, lo primero que hice fue empezar un proyecto limpio desde cero y añadir únicamente lo necesario. Con esto, sabríamos exactamente todo lo que se ejecutaba y había en el proyecto.

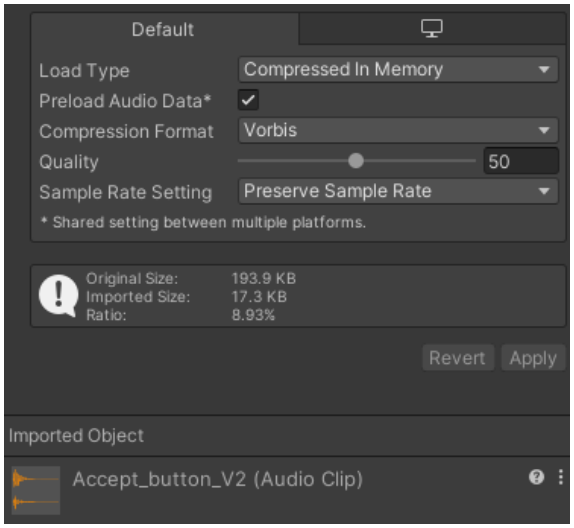


Ilustración 45: Ajustes de importación de un SFX en NeonHat. Fuente: Propia.

Después de esa limpieza, uno de los principales problemas fue el apartado gráfico, ya que la tarjeta gráfica de la consola no tenía capacidad suficiente para soportar los niveles de nuestro juego en Realidad Virtual.

Mi parte del trabajo, sin embargo, no fue en esa dirección.

Lo primero que hice fue modificar todo el audio que teníamos en el proyecto para tratar de hacer que los archivos de sonido ocupasen mucha menos memoria sin perder calidad de forma notable, ya que parte del problema era la cantidad de información en RAM.

Con esta acción, conseguimos ahorrar más de un 90% de memoria en algunos casos.

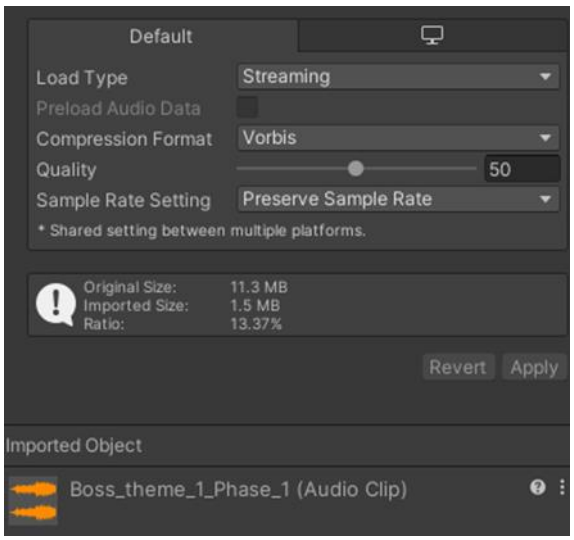


Ilustración 46: Ajustes de importación de una canción en NeonHat. Fuente: Propia.

Después de esto, me dediqué junto a mis compañeros a cambiar diferentes elementos estéticos en los niveles a otros con menos consumo de memoria *VRAM* en la tarjeta gráfica.

Finalmente, descubrimos que otro problema era el uso de caracteres especiales del castellano, por lo que reuní a todo el equipo y dirigí una migración completa de lenguaje al inglés en todo tipo de texto: desde el nombre de los archivos de sonido hasta los comentarios en el código.

6.4.2. Final de hito

Tras aproximadamente un mes en el que estuvimos realizando este proceso de mantenimiento, conseguimos tener el proyecto totalmente enfocado a *PlayStation 4* (Sony Interactive Entertainment, 2013) y llegó el momento de la prueba definitiva.

Tras compilar y ejecutar *NeonHat* en la consola, el nivel que teníamos hecho funcionaba casi a la perfección. Fallaban algunos controles que arreglamos más tarde; pero se veía perfectamente fluido, con una velocidad que de entre 100 y 200 fotogramas por segundo durante una carrera.

La detención de la producción mereció la pena: habíamos arreglado el juego.

6.5. Finalización de la producción del juego (Sprint 5)

Con lo que teníamos hasta la fecha funcionando perfectamente, ya sólo faltaba pulir algunos apartados y añadir contenido al juego.

Durante este *sprint*, mientras mi compañero en Programación; Sergio Jimeno; se centró principalmente en todo lo relativo a las carreras, yo ayudé a diseñar los jefes y me centré en programar una gran parte de las peleas contra estos jefes, diversas opciones de personalización, varias fuentes de retroalimentación, y parte del sistema de traducción y localización.

6.5.1. Primera aproximación a los jefes: Troyano Titánico

El primer jefe final que programé fue el "Troyano Titánico". La idea es que fuera un robot inmenso que actuara de forma diferente dependiendo de la posición del jugador en un coliseo tridimensional.

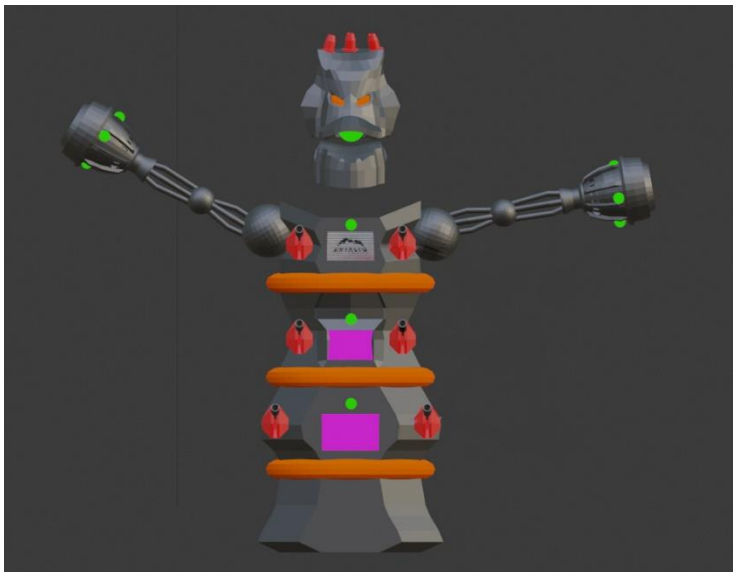


Ilustración 47: Primera versión para jefe Troyano Titánico de NeonHat. Fuente: Gonzalo Gámiz.

El jugador tendría que rodearlo atacando a sus puntos débiles mientras esquivaba sus ataques, y tiene la siguiente estructura:

- El jefe está compuesto por tres cilindros que giran durante los ataques y una cabeza que intenta mirar siempre al jugador.

- Entre los cilindros y entre el tercer cilindro y la cabeza, tiene un aro (sumando un total de tres).
- En la cabeza tiene tres lanzamisiles y en cada cilindro 4 cañones.
- En cada uno de los dos primeros cilindros tiene dos puertas cerradas y dos puntos débiles.
- En el tercer cilindro tiene dos puertas cerradas y dos brazos.
- En cada mano, al final de cada brazo, tiene cuatro puntos débiles.
- Al destruir todos los puntos débiles exteriores de cada cilindro, se abren las puertas de este, mostrando un punto débil interior.
- Al destruir el punto débil de cada cilindro, este se destruye.
- Al destruir todos los cilindros, los aros se colocan alrededor de la cabeza y comienza la fase final; en la que el último punto débil está en el interior de la boca. Al acabar con ese punto débil, el jefe es derrotado.

Tiene los siguientes ataques, combinables entre sí:

- Burbujas: El jefe dispara por los cañones de sus cilindros esferas de energía que dañan al jugador. Puede disparar desde un cilindro, desde dos, o desde los tres. Los cañones siempre apuntan a la altura a la que se encuentre el jugador.
- Misiles: Dispara misiles dirigidos hacia el jugador desde su cabeza, desde uno hasta tres. Este ataque se ejecuta siempre disparando tres misiles cuando el jugador está muy alejado del jefe.
- Pulsos: El jefe emite pulsos desde uno, dos, o los tres aros, creando anillos que dañan al jugador.

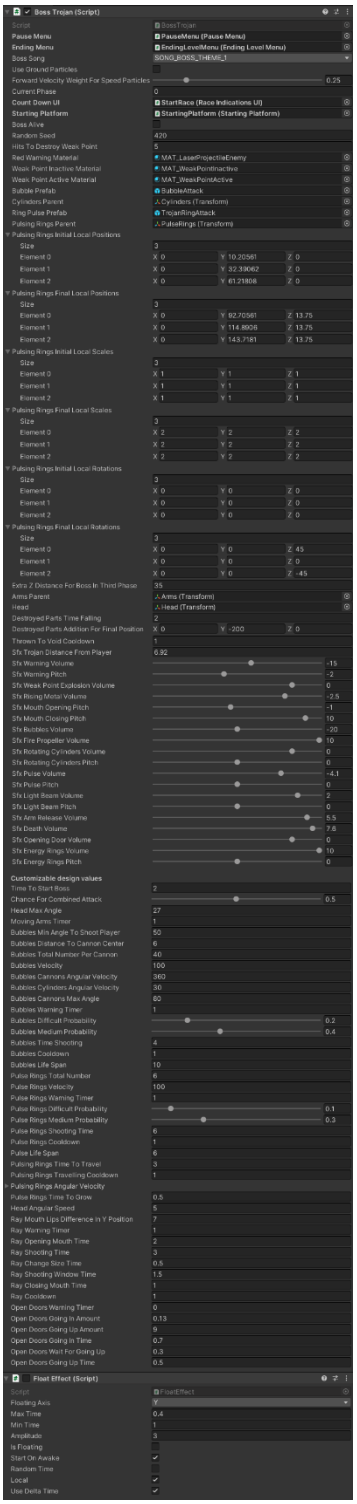


Ilustración 48: Parámetros del jefe Troyano Titánico en NeonHat. Fuente: Propia.

- Puñetazo: Cuando el jugador está muy cerca, el jefe le asesta un puñetazo, dañando al jugador. El movimiento del brazo está logrado utilizando Cinemática Inversa[27].
- Rayo: Desde que se destruye el primer punto débil interior del jefe, el jefe puede disparar un rayo desde su boca que daña al jugador al contacto.

Todos los proyectiles de este jefe (y de los siguientes), se generan en una *pool*[28] calculando el máximo posible en pantalla para procurar su reutilización y disminuir la carga de trabajo durante la ejecución.

Este primer jefe fue el más costoso de todos. Tanto que, mientras yo lo acababa de programar, mi compañero Sergio Jimeno terminó con la programación de las carreras y comenzó a programar los jefes Asaltante Acorazado y Devoradora Desatada.

Cuando terminamos y probamos este jefe, nos dimos cuenta de que se hacía bastante confuso de jugar y entender en Realidad Virtual, pero no podíamos permitirnos perder tantos días de trabajo como había costado, así que decidimos que lo revisaríamos más adelante, dejando en el funcionamiento del jefe todo parametrizado y modificable desde editor, ya que tenía que seguir haciendo el siguiente jefe por el momento. Se puede ver la gran cantidad de parámetros modificables de los que dispone este jefe en su clase *BossTrojan* (Ilustración 48).

6.5.2. Tejedora Terrible

Para este jefe el movimiento se vio modificado, pasando a ser parecido a un *Endless Runner*: el jugador se encuentra en una cinta infinita como si estuviera desplazándose hacia el frente y con la mecánica de giro desactivada para poder ir únicamente en línea recta.

Este jefe es una araña que comienza atacando con unos bloques que se generan en el camino del jugador, quien deberá esquivarlos mientras avanza. De golpear al jugador, le dañan.

Para usar los bloques también se hace uso de una *pool*. La velocidad va en aumento haciendo cada vez mayor el espacio ocupado por bloques en cada línea de bloques hasta el final de la fase.

Durante esta fase, la araña está oculta bajo el suelo y el jugador deberá mantenerse con vida hasta que el jefe se muestre.

27 Cinemática Inversa (IK) es el uso de ecuaciones de cinemática para determinar los parámetros de las articulaciones de un manipulador para que el efector final se mueva a una posición deseada. (Aristidou, Lasenby, Chrysanthou, & Shamir, 2018)

28 El patrón de "Pooling" describe cómo la cara adquisición y liberación de recursos se puede evitar reciclando recursos que ya no se necesitan. (Kircher & Jain, 2002)

Una vez se muestre, comenzará a atacar combinando sus patas (también mediante el uso de Cinemática Inversa) y los bloques que genera. Mientras tanto, el jugador deberá destruir los seis puntos débiles. Estos puntos débiles están situados en los ojos de la araña.

El código del funcionamiento de la cinta con los bloques y sus parámetros de dificultad está en la clase *SpiderTreadmill* (Ilustración 49), mientras que los parámetros relacionados únicamente con la araña se encontrarán en *BossSpider* (Ilustración 50).

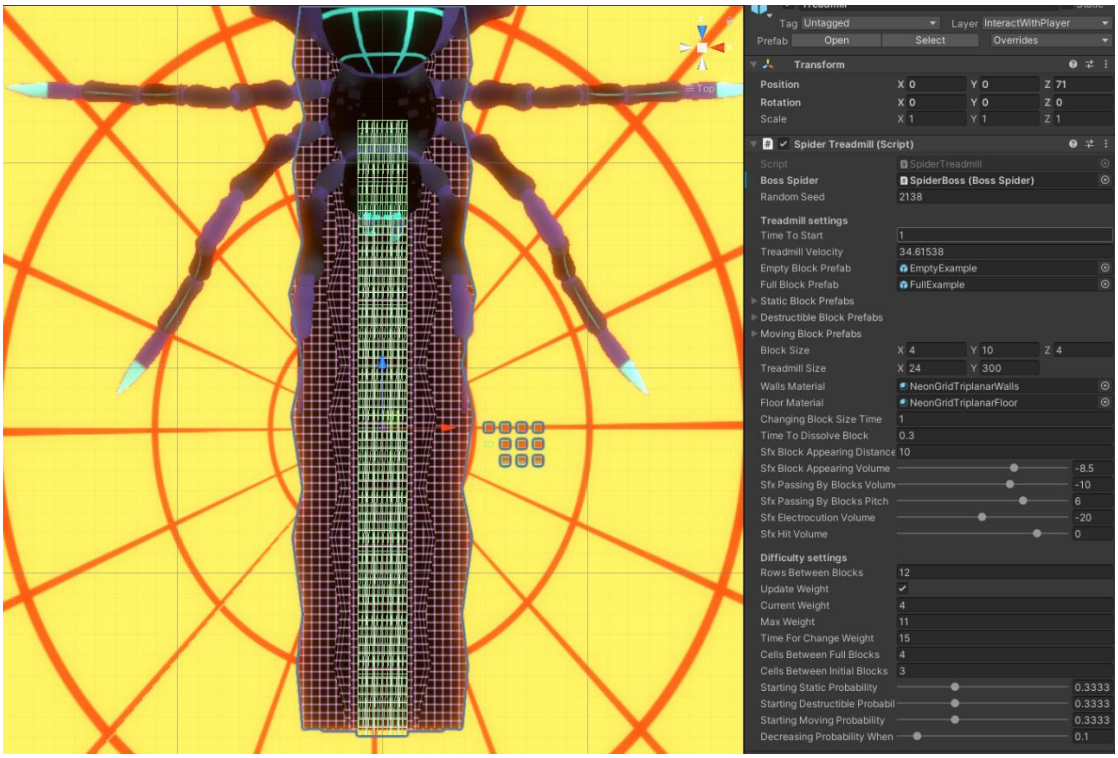


Ilustración 49: Cinta de bloques en el jefe Tejedora Terrible de NeonHat. Fuente: Propia.

Para el funcionamiento de la cinta de bloques, se crea una matriz con todas las posiciones posibles para los bloques y dichas posiciones se actualizan en cada fotograma modificando su posición en el eje Z. Cuando pasan de un límite dado, se recolocan en una posición Z inicial y; tras ocurrir este proceso un número dado de veces; se ponen bloques dependiendo de la dificultad actual en la hilera a lo largo de X que se acabe de colocar en la posición inicial.

Para determinar la dificultad se utiliza un sistema de pesos. La dificultad comienza en peso 4 y asciende hasta 11 con el tiempo. Cada hilera que se llene de bloques deberá cumplir con el peso de dificultad que haya en el momento, costando un bloque completo 2 y un bloque por la mitad 1.

Siempre se colocará el bloque con menor peso de la hilera a una distancia máxima en X de 3 bloques desde el bloque con menor peso de la anterior hilera generada, para asegurar como posible el paso del jugador por la hilera de bloques sin recibir daño.

Cuando este jefe estuvo finalizado y lo probé, me pareció muy notable lo divertido e intuitivo que era en comparación al Troyano Titánico. Avisé a mis compañeros para probarlo todos y decidimos que habría que cambiar los diseños del resto de jefes para adaptarlos a ser más parecidos al de la Tejedora Terrible: habíamos dado con la clave para hacer jefes divertidos en nuestro juego.

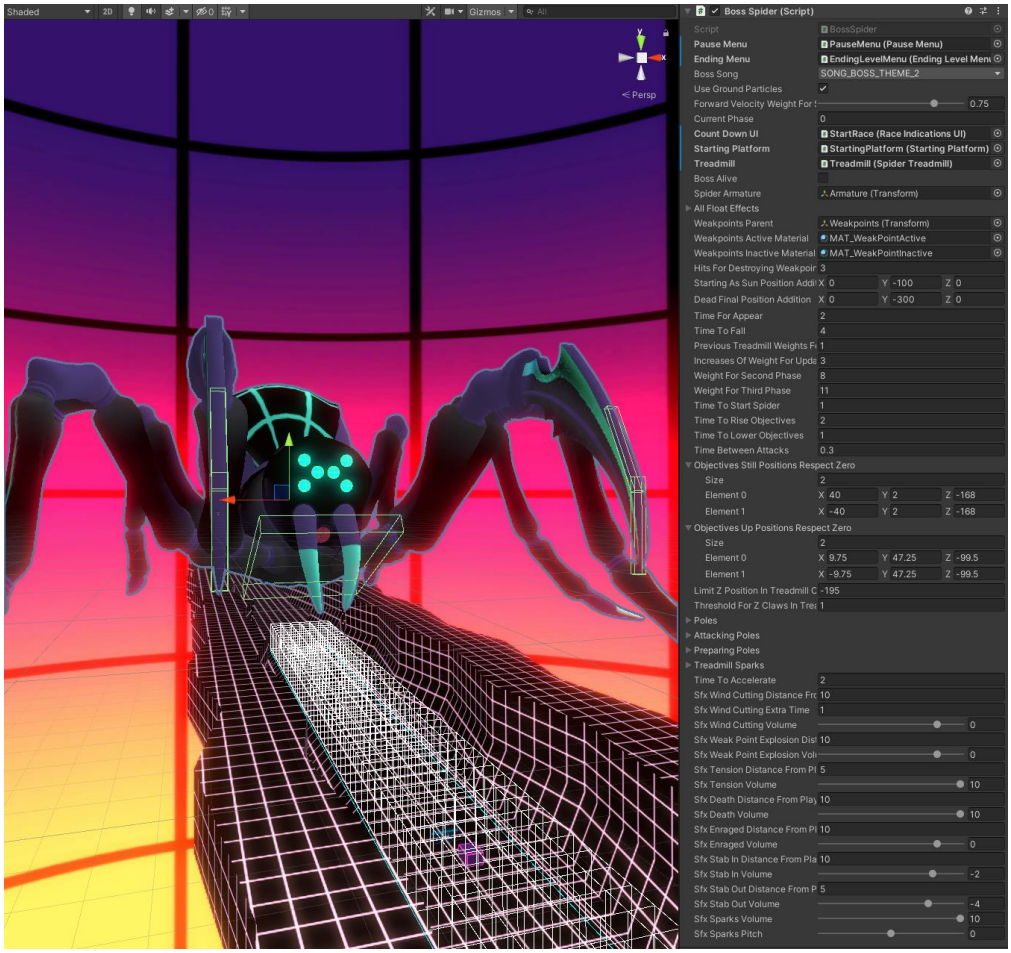


Ilustración 50: Parámetros del jefe Tejedora Terrible en NeonHat. Fuente: Propia.

6.5.3. Giro en el diseño de los jefes: Troyano Titánico versión 2

Copiamos el concepto de la araña, haciendo que el jugador no pudiera girar y se encontrara enjaulado en un espacio.

Algunos ataques ya no tendrían sentido, así que los eliminamos:

- Los misiles seguían siendo confusos y ya no tenían sentido a larga distancia porque el jugador no podía alejarse.
- Los puñetazos tampoco servían ningún propósito ahora que el jugador no podría acercarse.

Para no desechar la utilizad de la Cinemática Inversa en los brazos, hicimos que fueran capaces de levantarse y lanzar fuego desde la punta, que dañaría al jugador al contacto.

Además, los cilindros ahora estarían siempre girando para permitir al jugador atacar a los puntos débiles sin tener que esperar a que el jefe ataque aleatoriamente, lo que combina perfectamente con el nuevo ataque de los brazos.

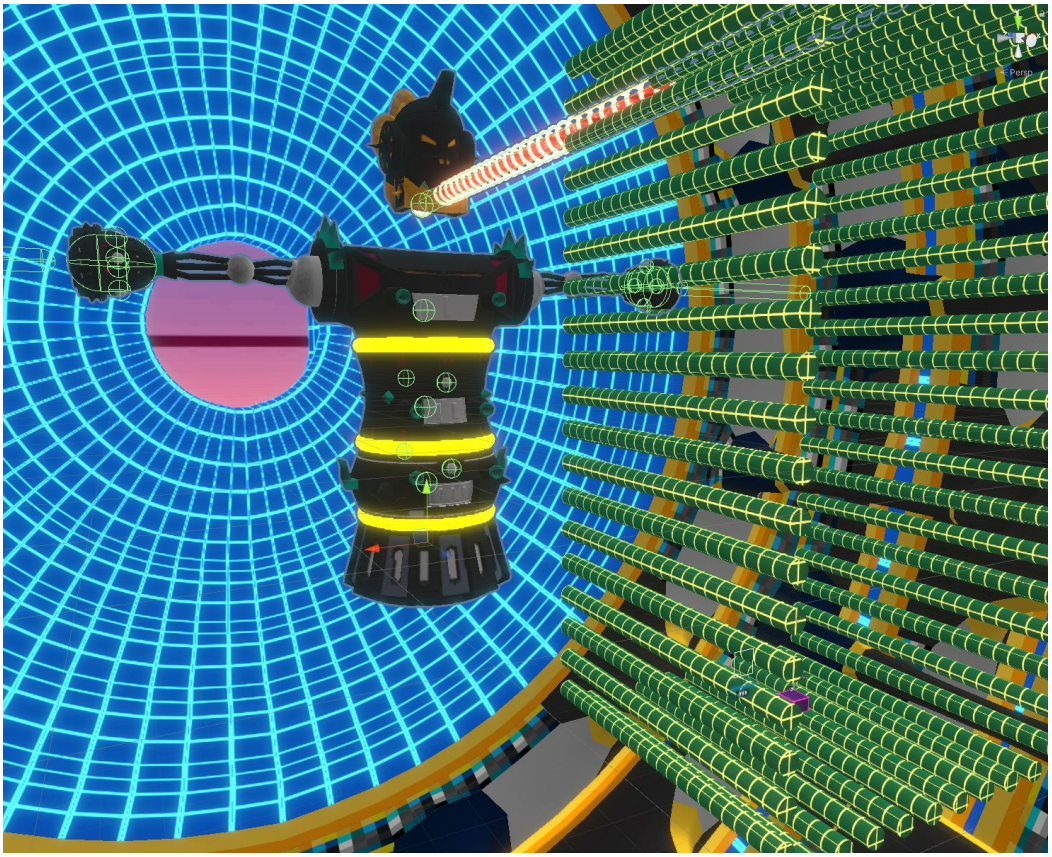


Ilustración 51: Segunda versión del Troyano Titánico (sin lanzamisiles y con el jugador enjaulado). Fuente: Propia.

6.5.4. Asaltante Acorazado

Cuando terminé con la nueva versión del Troyano Titánico, Sergio comenzó a hacer la Devoradora Desatada, que sería el penúltimo jefe, mientras yo pasaba a hacer el Asaltante Acorazado.

Este jefe es un barco que se mueve más rápido que el jugador y se acoraza al estar a cierta distancia del jugador, por lo que para dañarlo habrá que acercarse a él utilizando anillos de turbo como los que se pueden encontrar en las carreras.

Este jefe tiene sólo dos puntos débiles: uno en la proa y otro en la popa. El punto débil en la proa estará constantemente acorazado mientras el otro no haya sido destruido.

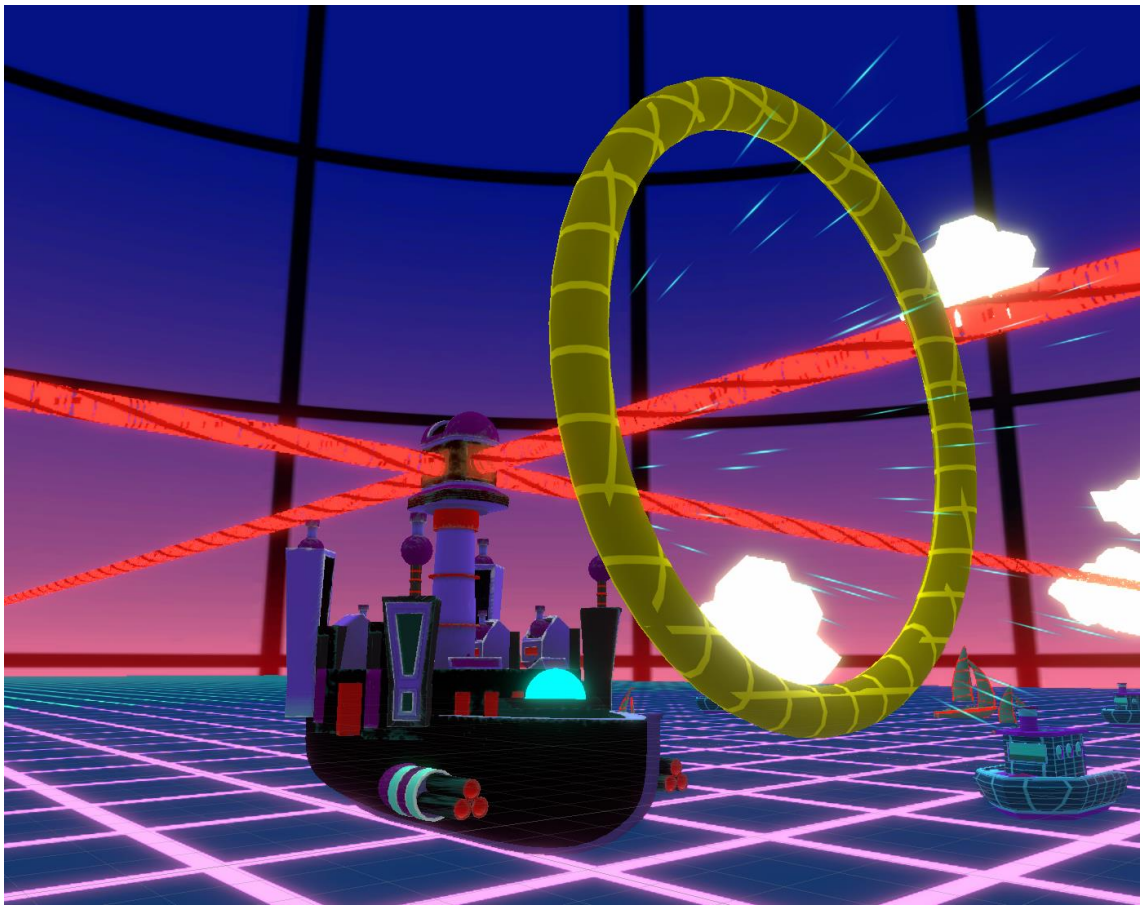


Ilustración 52: Jefe Asaltante Acorazado en el fondo, anillo de turbo en primer plano. Fuente: Propia.

De nuevo, este jefe funciona bloqueando al jugador en una jaula sin dejarle girar y moviendo todo hacia atrás para hacer creer al jugador que se mueve hacia delante.

Para los anillos de turbo y los ataques del jefe se han utilizado también *pools*.

Estos ataques son un total de 5 ataques diferentes, teniendo un total de 3 en su primera fase, 4 en su segunda fase, y sólo el último de los 5 en la tercera fase.

Los ataques de la primera fase son los siguientes:

- Burbujas: De forma similar al Troyano, puede disparar desde sus dos cañones traseros al jugador burbujas que lo dañan al contacto.
- Bombas: Desde los dos propulsores traseros, el jefe deja caer de forma alterna bombas que se mantienen sin explotar durante unos segundos, hasta que lo hacen y emiten una onda expansiva lenta que daña al jugador mientras permanezca en su interior.
- Rayo de faro: Desde el faro, el jefe emite cuatro rayos que dañan al jugador al contacto.

Cuando el primer punto débil (situado en la popa) ha perdido la mitad de la vida, comienza la segunda fase, que contará con los mismos ataques que la primera con un pequeño añadido:

- Barcas de apoyo: El jefe deja caer desde sus laterales dos barcas de apoyo que actuarán por su cuenta para atacar al jugador. Cada barca de apoyo cuenta con un cañón de burbujas y con un punto débil que se puede destruir de un solo golpe.
  Si se destruye uno de los dos, el jefe dejará caer una nueva barca de apoyo pasado un tiempo.

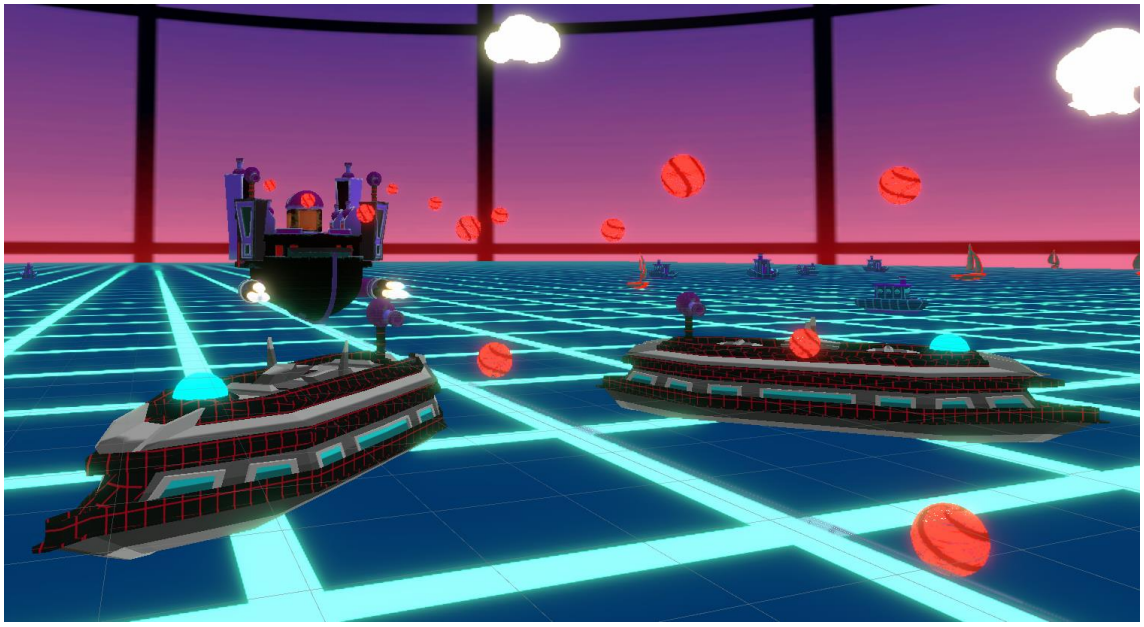


Ilustración 53: Jefe Asaltante Acorazado y dos barcas de apoyo disparando burbujas. Fuente: Propia.

Cuando se destruya el primer punto débil, el jefe sufre una transformación y comienza a volar, pasando a su fase final. En esta fase sólo cuenta con un ataque:

- Burbujas+: Esta evolución del ataque anterior cuenta con 6 cañones más, haciendo un total de 8 cañones disparando al jugador y convirtiendo esta fase en un auténtico *bullet hell*[29].

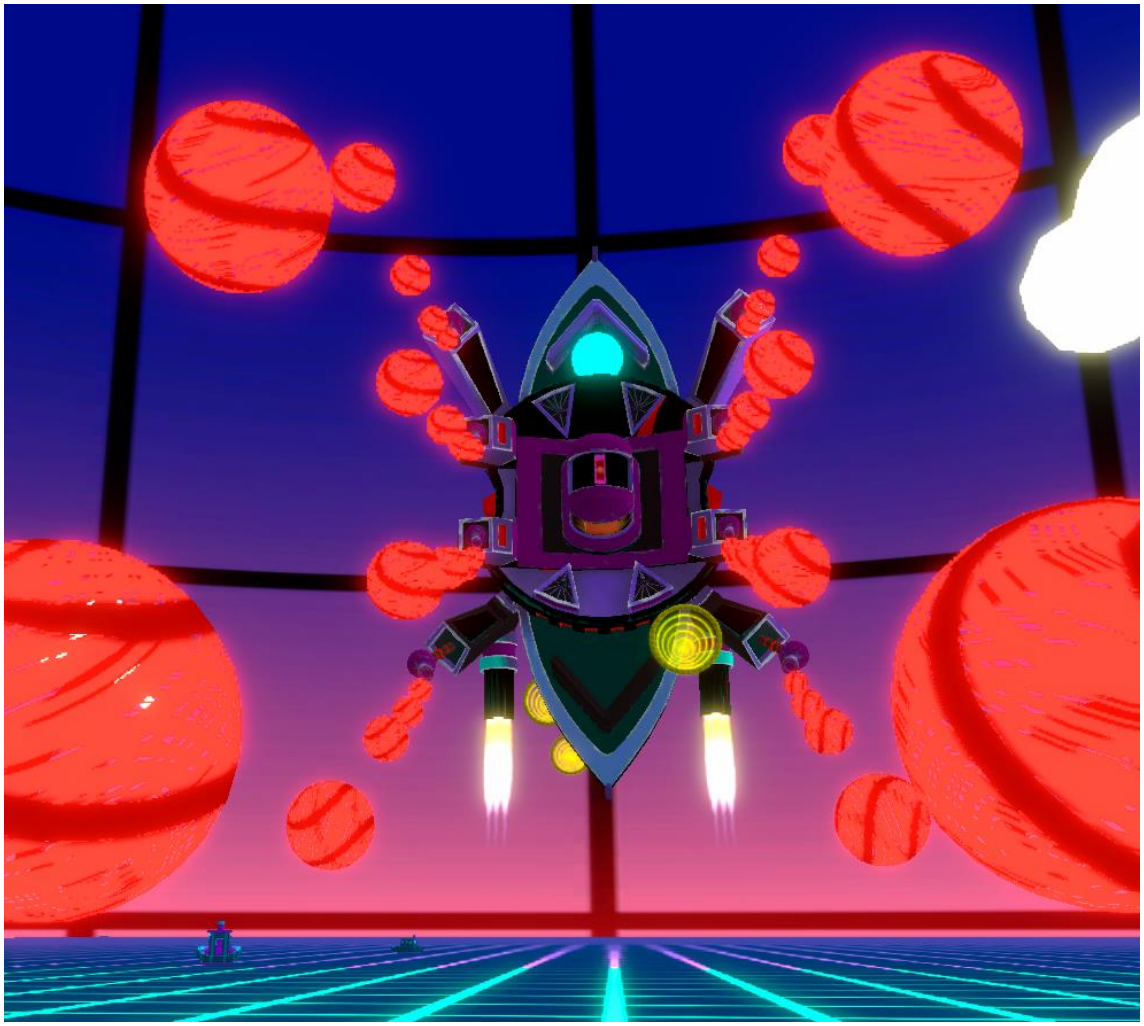


Ilustración 54: Jefe Asaltante Acorazado durante la tercera fase, disparando burbujas al jugador. Fuente: Propia.

6.5.5. Devoradora Desatada

Este se divide en dos fases: el combate contra la serpiente y la búsqueda del punto débil en su interior.

29 Juegos en los que los enemigos invaden toda la pantalla de balas.

La primera fase, el combate contra la serpiente, lo programó mi compañero Sergio Jimeno, por lo que únicamente hablaré de la segunda fase.

Ilustración 55: Jefe Devoradora Desatada. Fuente: Propia.

Al final de la primera fase, este jefe devora al jugador. Con esto lleva la pelea a su interior, cuando el jugador tiene que esquivar los bloques que tiene en su interior el jefe hasta llegar al final y destruir el punto débil; acabando con el jefe.

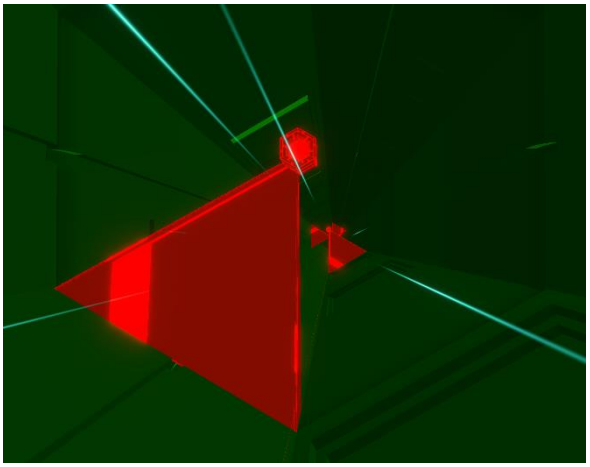


Ilustración 56: Interior del jefe Devoradora Desatada durante la segunda fase de la pelea contra este jefe. Fuente: Propia.

La idea para el funcionamiento es bastante similar a la cinta de la araña; sin embargo, parte de la programación es diferente.

Al ser todos los bloques similares, en esta versión de la cinta hay una *pool* de bloques completos como el que puede verse en la Ilustración 57. Cada triángulo se puede desactivar por separado, y para hacerlo se calcula en una máscara de bits qué bloques están vacíos y cuáles no dependiendo de la dificultad actual (que va en aumento con el tiempo, y cuanta más dificultad, menos bloques libres).

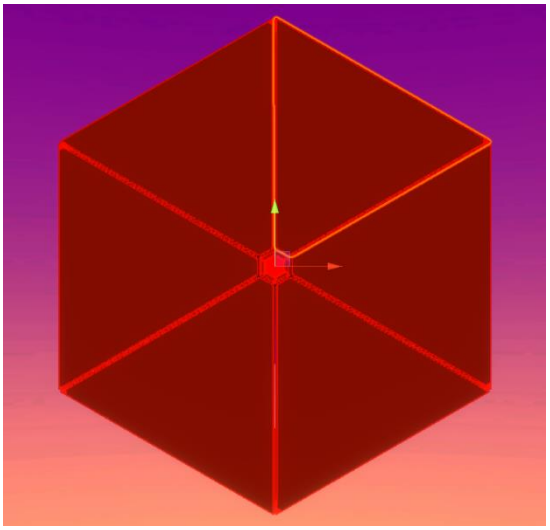


Ilustración 57: Bloque completo para la segunda fase del jefe Devoradora Desatada. Fuente: Propia.

Cuando el jugador llega al final de la cinta, encuentra el punto débil y acaba con el jefe.

6.5.6. Modo de juego personalizable

Mientras mi compañero Sergio Jimeno se centró en hacer el jefe final del juego, yo comencé a preocuparme por los detalles del juego, como la personalización.

En Realidad Virtual es importante dejar que el jugador personalice su experiencia de forma que el juego sea lo más cómodo para cada uno sin modificar la base de este, así que integré un nuevo sistema de giro automático, accesible desde el menú de opciones.

Hasta ahora, para girar había que pulsar un botón en el mando. Al pulsarlo, se permitía al jugador hacer que el HAT girara al girar el mando. Sin embargo, si la opción se pone en "Automático" en vez de en "Manual", no habrá que pulsar este botón y con tan sólo girar el mando por encima de un umbral el HAT comenzará a girar.

También seguía habiendo gente que se mareaba al probar el juego. Dedujimos que era por falta de puntos de referencia, y tras varias pruebas de vuelo y combate con una cápsula (Ilustración 58 e Ilustración 59) rodeando al jugador vimos que era viable y la experiencia de juego era la misma. Tras probar que la gente se mareaba menos con la cápsula, la añadí también como algo opcional desde el menú de opciones.

También hicimos que se pudiera modificar el tamaño de la mira y el color de los propulsores. Esto, junto a las opciones de idioma y de volumen que ya había, dejó un menú de opciones bastante completo y personalizable, como se puede apreciar en la Ilustración 60.

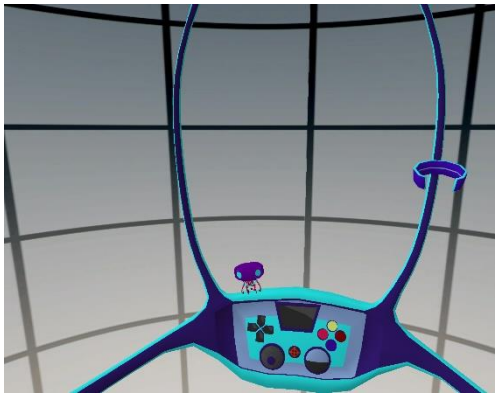


Ilustración 58: Cápsula opcional para el jugador vista desde dentro. Fuente: Propia.

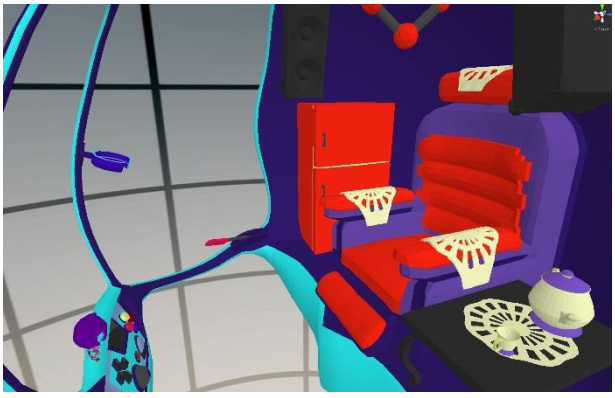


Ilustración 59: Cápsula opcional para el jugador vista desde fuera. Fuente: Propia.

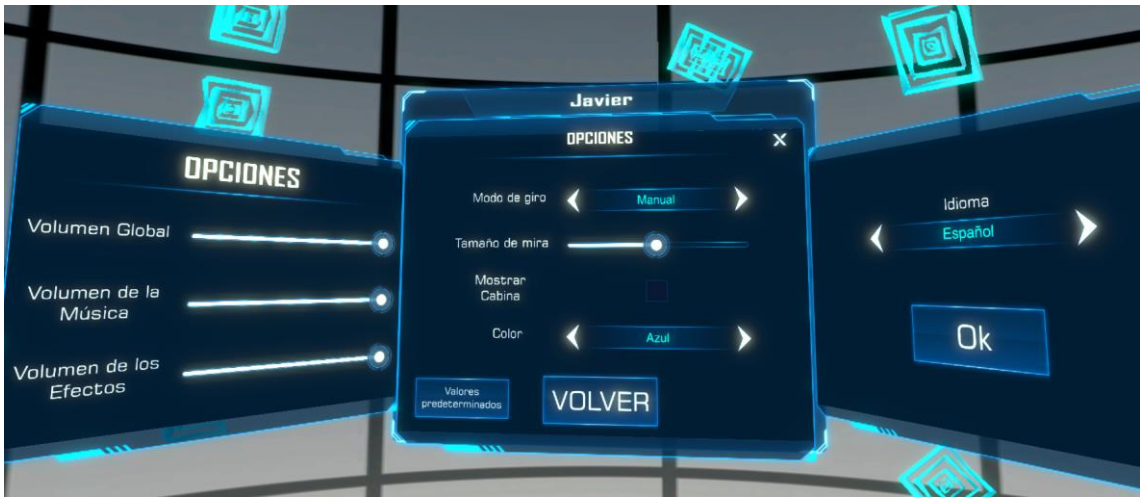


Ilustración 60: Menú de opciones de NeonHat. Fuente: Propia.

6.5.7. Retroalimentación

Otro elemento muy importante en los videojuegos siempre es la retroalimentación que reciba el jugador con cualquier acción. Por eso decidí también pulir este apartado, empezando por las partículas de viento.

Una forma muy efectiva de generar en el jugador sensación de velocidad es mediante partículas moviéndose a su alrededor hacia atrás a gran velocidad. Esto lo sabíamos desde el principio del desarrollo, por lo que teníamos una versión temprana de estas partículas, que se perfeccionó en el final del desarrollo. Estas partículas también decidimos que aparecieran en jefes que simulen movimiento como el Asaltante Acorazado o la Tejedora Terrible.

De igual forma, un método elegante para avisar al jugador cuando esté cerca del suelo es levantar polvo, por lo que hice que al estar próximo al suelo se generaran particular saliendo del mismo. El efecto queda muy vistoso, pareciendo ser chispas al chocar con el suelo.

Cuando se está derrapando, una forma muy fácil de saber cuándo está listo el turbo del derrape es con partículas de diferentes colores. Es por esto por lo que al derrapar los propulsores comienzan a emitir partículas, que cambian de color cuando se pueda utilizar el turbo.

Finalmente, se volvía un poco confuso cuándo dábamos con los láseres a los enemigos y cuándo no. Por ello, decidimos hacer vibrar al mando y aportar un añadido visual a la mira cuando se golpeara a un enemigo al disparar.

6.5.8. Traducciones

Para el sistema de localización, dividimos todos los textos entre textos fijos y textos variables.

Los textos fijos serían todos los textos que no se verían modificados durante la ejecución (excepto al cambiar de idioma), y los variables serían todos los que se verían con texto diferente dependiendo de qué hubiera que mostrar en cada momento, como: descripciones de niveles, espacios de guardado, valores de menú de opciones, o tablas de puntuaciones.

Para traducir la mayor parte de textos variables hice la clase estática *RealTimeTranslations*, que permite acceder a traducciones para todos estos apartados desde cualquier otro código en el que necesite traducir palabras, ya que guardará todas las traducciones de las palabras sueltas que se puedan necesitar.

Sin embargo, para los títulos de los niveles y las descripciones, utilicé otro sistema que ya había programado cuando hice el sistema de progreso: en cada nivel, hay un apartado para su nombre, un título a mostrar, y una descripción a mostrar, para cada idioma que se necesite.

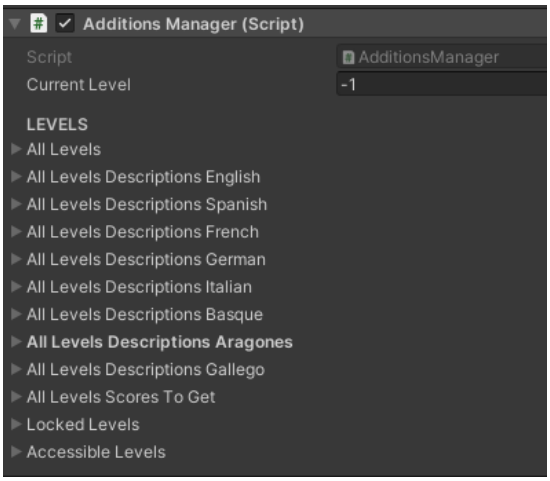


Ilustración 61: Clase AdditionsManager desde el editor de Unity, con todas las descripciones de los niveles en los diferentes idiomas. Fuente: Propia.

Finalmente, todos los textos fijos los decidimos localizar con un tercer sistema: *Localization* (Unity Technologies, 2020), en su versión 0.10.0.

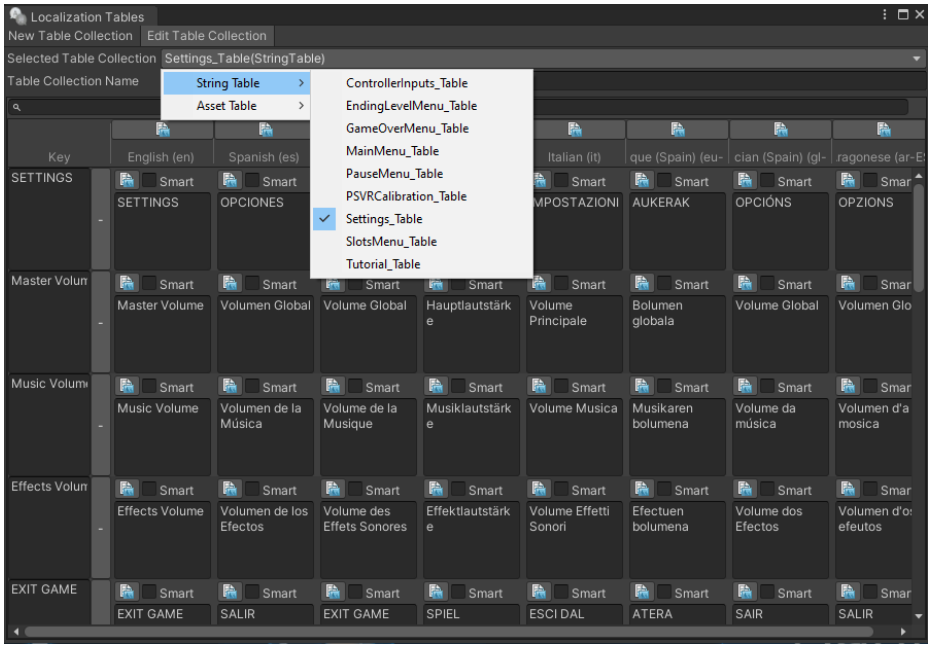


Ilustración 62: Tablas de localización del paquete Localization de Unity Technologies en su versión 0.10.0 en el proyecto de Unity de NeonHat. Fuente: Propia.

6.5.9. Últimos retoques

Los últimos días antes del final del hito me dediqué enteramente a la corrección de diversos *bugs*[30] que habían surgido a lo largo de estos meses, como perder los menús detrás de paredes.

Este problema se daba porque; al ser un juego de Realidad Virtual y tener los menús en el espacio de mundo; cuando el jugador abría un menú estando cerca de un suelo o pared, y esto resultaba en el menú abriéndose debajo del muro, con lo que se perdía parcial o completamente. Lo arreglamos poniendo a todos los menús un mismo material creado específicamente para la interfaz de usuario con una prioridad en la cola de renderizado diferente al resto de materiales. Tras esto, hubo que hacer que los mandos y láseres también se renderizaran encima de los menús.

También tuvimos diversos problemas con los menús, ya que el control de cosas como botones deslizadores en espacio de mundo utilizando el apuntado de los mandos no funcionaba de forma tan consistente como debiera, perdiendo en ocasiones los colisionadores que habíamos puesto en estos. Sin embargo, acabamos por solucionar estos problemas también.

30 "Los bugs informáticos son errores o fallas de un programa o sistema que produce resultados inesperados, es decir, que trabaja de una forma para la que no estaba diseñado originalmente". (Gómez Baray, 2020)

6.5.10. Final de hito y sprint review

Finalmente teníamos el juego y funcionaba casi todo perfectamente en la consola.

Le enseñamos lo que teníamos a Manuel Martínez (director técnico en *PlayStation Talents*) y nos dio su aprobado, por lo que estábamos mucho más cerca de poder enviar el juego a *QA*[31].

Este *sprint* lo conseguimos completar con todos los objetivos finalizados.

6.6. *Porting* a *PlayStation* 4, *QA*s, y arreglo de bugs (Sprint 6)

Para este sprint, *PlayStation Talents* nos envió un *Development Kit* a Zaragoza, por lo que pudimos comenzar a probar diferentes sistemas en la consola objetivo de forma más continua, ya que aunque hubiéramos acabado con la producción, quedaban algunos detalles por arreglar. Una vez estos detalles estuvieran solucionados, habríamos de pasar la prueba de calidad de *Sony* "*Final Quality Assurance*" (*FQA*) para poder publicar el juego en *PlayStation 4* (Sony Interactive Entertainment, 2013).

Debido a que este es un videojuego de Realidad Virtual, también deberemos superar otro proceso similar denominado "*VR Consultation*".

Así, los objetivos de este último *sprint* fueron: la modificación del sistema de guardado para adaptarse a la consola objetivo, la programación de logros, y pasar satisfactoriamente todos los *QAs* necesarios.

6.6.1. Modificación del sistema de guardado

En PlayStation había problemas para guardar de la misma forma que en PC, así que Sergio Jimeno se puso de nuevo en contacto con Manuel Martínez, que le dijo la forma recomendada de guardar los datos: utilizando *PlayerPrefs* (Unity Technologies, 2021).

Esto hizo que tuviéramos que cambiar también la forma en que se guarda: hasta ahora lo hacíamos en un hilo aparte y guardando la información cada vez que se modificara cualquier elemento de información (por ejemplo, en el menú de opciones), pero sólo se puede acceder a PlayerPrefs desde el hilo principal. Solucionamos este problema utilizando corrutinas y descendiendo el número de veces en que guardábamos los datos a hacerlo únicamente al cambiar de escena, para evitar problemas de rendimiento al cerrar el menú o similares.

31 "*QA* significa *Quality Assurance* y su función no solo abarca el control de calidad del desarrollo del software sino también ayudar a todo el equipo para que las cosas funcionen correctamente". (F. Coelho, 2019)

6.6.2. Logros

De la mayor parte de los logros se encargó enteramente Sergio Jimeno, pero yo ayudé en un apartado en particular: para ciertos logros, es necesario saber de forma sencilla y rápida los niveles que se han completado en cualquiera de los espacios de guardado (ya que los logros de *PlayStation* no van por espacio de guardado, sino por copia del juego).

Como tenemos 69 niveles diferentes, desarrollé un sistema de máscara de bits con 3 enteros para un total de 96 bits disponibles con el que saber qué niveles se habían completado en cualquier momento con esa copia del juego.

6.6.3. Pre-Gold

El juego debía cumplir con las especificaciones descritas en el documento *Technical Requirements Checklist* de *Sony* (Sony Interactive Entertainment Inc., 1993); un documento de 257 páginas con todos los requerimientos que un juego debe cumplir para ser publicado en *PlayStation* 4 (Sony Interactive Entertainment, 2013) del que, por motivos de confidencialidad, no puedo dar detalles.

Cuando acabamos de ultimar detalles, enviamos el juego al equipo técnico de *PlayStation Talents*, para que pasara su propio *QA* antes de enviarlo a los oficiales en Europa y América.

En el primer envío hubo ciertos problemas, y yo me encargué de corregir fallos con el tutorial: no cambiaban correctamente las imágenes dependiendo de qué mando utilizaras y algunos textos se traducían incorrectamente.

En el siguiente envío, pasó correctamente este *QA*, y enviamos el juego al *VR Consultation* de *Sony* (Sony Interactive Entertainment Inc., 1993); el cual también superamos; por lo que pudimos enviar el proyecto al *FQA* de *Sony* Europa para entrar en fase *Gold*[32].

6.6.4. Final de hito

Recibimos un informe con una lista de errores encontrados y su importancia para pasar el *FQA*: los errores catalogados como "*MUST FIX*" deberán ser corregidos para la superación de este *QA*, mientras el resto de los errores podrán corregirse más adelante, en futuras actualizaciones del juego.

32 "El *status Gold* indica que la creación del videojuego ha terminado y se ha creado el master, el disco que se mandará para que, a partir de él, se tiren y distribuyan las copias que llegarán a los usuarios". (Quesada, 2018)

Los errores con la catalogación de "*MUST FIX*" fueron detalles como una calificación *PEGI*[33] errónea: Habíamos puesto el juego como *PEGI* 3, pero al haber disparos de láseres debía ser *PEGI* 7 por violencia leve. Tras corregir estos fallos pasamos el *FQA* el 17 de julio de 2021, entrando por fin en fase *Gold* para Europa.

Tras esto, procedimos a enviar el proyecto al *FQA* de *Sony* América, para el cual tuvimos que cambiar la calificación *PEGI* por la calificación *ESRB*[34]. Tras superar también este *QA*, entramos también en fase *Gold* para América.

Ahora sólo quedaba crear los materiales para nuestra página en la *PlayStation Store* (Sony Interactive Entertainment Europe Ltd., 2021): tráileres, capturas de fotogramas del juego, logotipos, etc.

Por fin *NeonHat* (Entalto Games S.L., 2021) estaba acabado e íbamos a publicar nuestro primer videojuego en la *PlayStation Store*.

33 "PEGI proporciona clasificaciones por edad para videojuegos en 38 países europeos. La clasificación por edad confirma que el juego es apropiado para jugadores de cierta edad. PEGI considera la idoneidad de edad de un juego, no el nivel de dificultad". (Pan European Game Information, 2003)

34 Las calificaciones ESRB proveen información sobre qué hay en un juego o aplicación para que los padres y consumidores puedan realizar elecciones con información sobre qué juegos son apropiados para sus familias. (Entertainment Software Association, 1994)

7. Estudio económico

En el subapartado "5.2. Metodología empleada y por qué ha sido elegida" se ha visto el equipo de desarrollo, que en la Ilustración 63 se ve reflejado en los apartados "*Design and management"*, "*Development"*, "*2D Art"*, "*Employees"*, y "*Outsource"*.

En el apartado de la Ilustración 63 "*Operational costs"* se encuentran diversos costes derivados de otros elementos ya vistos: los costes de las diferentes herramientas de software vistas en el subapartado "5.3. Herramientas" están incluidos en "*Hardware and Software"*; el coste de los servidores empleados para el control de versiones descrito en el subapartado "5.3.5. TortoiseSVN" está incluido en *"Servers"*; y los viajes a Valencia descritos en los subapartados "6.3.6. Final de hito y sprint review" y "6.4. Detención de producción y arreglos para *porting* (Sprint 4)" están incluidos dentro de *"Travel"*.

Finalmente, podemos observar en esta Ilustración 63 que el total del coste de *NeonHat* (Entalto Games S.L., 2021) asciende hasta los 138.810 euros.

Team Plan	Development Plan									
	Sep.21	Oct.21	Nov.22	Dec.22	Jan.23	Feb.23	Mar.24	Apr.24	May.25	
	NeonHAT V.1			NeonHAT + Arreglos			Beta		Gold	
Position	**Cost Gross $**									**Total**
Design and management										
CEO	1500	1500	1500	1500	1500	1500	1500	1500	1500	13.500
Producer	1500	1500	1500	1500	1500	1500	1500	1500	1500	13.500
Game Designer	1500	1500	1500	1500	1500	1500	1500	1500	1500	13.500
Development										
CTO + Lead Backend Dev	1500	1500	1500	1500	1500	1500	1500	1500	1500	13.500
Lead Unity Dev	1500	1500	1500	1500	1500	1500	1500	1500	1500	13.500
2D Art										
3D Artist	1500	1500	1500	1500	1500	1500	1500	1500	1500	13.500
3D Artist							1500	1500	1500	4.500
Employees	6	6	6	6	6	6	7	7	7	
Salary Spends GROSS $	9000	9000	9000	9000	9000	9000	10500	10500	10500	85.500
Total Labour Cost $	11700	11700	11700	11700	11700	11700	13650	13650	13650	111.150
Social Tax	30%									
Operational costs										
Servers	30	30	30	30	30	30	30	30	30	270
Office rent + spends	500	500	500	500	500	500	500	500	500	4.500
Bookkeeping	60	60	60	60	60	60	60	60	60	540
Hardware and Software	150	150	150	150	150	150	150	150	150	1.350
Unexpected expenses	200	200	200	200	200	200	200	200	200	1.800
Travel				400	400	400	400			1.600
Outsource										
Localization								800	800	1.600
Sound design						1000	1000	1000	1000	4.000
3D Art		1500	1500	1500	1500	1500	1500	1500	1500	12.000
Total Monthly Costs	12.640	14.140	14.140	14.540	14.540	15.540	17.490	17.890	17.890	**138.810**

Ilustración 63: Precio y desglose del desarrollo del juego completo. Fuente: Jorge Chueca.

Debido a la confidencialidad del proyecto con *PlayStation Talents* (Sony Interactive Entertainment Europe, 2015-2021), no es posible dar detalles sobre las ventas actuales ni porcentajes de regalías.

8. Resultados

8.1. Proyecto final

NeonHat comenzó como un videojuego totalmente diferente al que se acabó desarrollando, pero se mantuvo el objetivo del proyecto: publicar en *PlayStation 4* (Sony Interactive Entertainment, 2013) un videojuego arcade de vuelo en Realidad Virtual.

Por mi parte, pude desarrollar por completo todas las tareas que dependían de mí, incluso algunas que quedaron fuera el proyecto final como el sistema de economía y el sistema de guiado y búsqueda de rutas en la ciudad.

Mi trabajo en el proyecto final ha acabado incluyendo la adaptación de los controles para *PlayStation* VR (Sony Interactive Entertainment, 2016), tanto utilizando controladores *Move* (Sony Interactive Entertainment, 2010) como el controlador *DualShock 4* (Sony Interactive Entertainment, 2013); he realizado mejoras en el sistema de vuelo como el giro, el derrape, el turbo, y la estabilización; he desarrollado una gran diversidad de sistemas: gestión de misiones, guardado y carga de datos, espacios de guardado, audio, y traducción y localización; 5 menús y un sistema para interactuar con ellos utilizando dispositivos de VR; funcionalidad de *Endless Run* para las peleas contra jefes; el comportamiento de 4 jefes y las peleas contra los mismos; y he ejecutado diversas pruebas de rendimiento en la consola objetivo con el fin de superar el *VR Consultation* y los *FQA* europeos y americanos de *Sony* (Sony Interactive Entertainment Inc., 1993).

Como consecuencia, todo este trabajo excede el total de 300 horas requeridas para la realización del Proyecto de Fin de Grado con creces, con un total final de 1565 horas.

Quiero justificar esta diferencia precisando que este proyecto ha sido más que un Proyecto de Fin de Grado. Este proyecto ha sido también un proyecto tanto personal como profesional: un videojuego acabado y coherente con la base que se concibió inicialmente:

Un videojuego arcade de carreras y peleas contra jefes exclusivo para Realidad Virtual.

8.2. Presente y futuro de *NeonHat*

NeonHat finalmente se lanzó a la venta el día 29 de julio de 2021 para *PlayStation* Europa y el 4 de agosto de 2021 para *PlayStation* América, recibiendo su primer parche con problemas en los marcadores solucionados y otros pequeños arreglos el 23 de agosto de 2021.

Ilustración 64: Mapa con los países en los que NeonHat está publicado coloreados en azul. Fuente: Sergio Jimeno.

El juego, a día de hoy, ya se puede ver y comprar en la *PlayStation Store* (Sony Interactive Entertainment Europe Ltd., 2021).

También hay diversos *gameplays* en *YouTube* (YouTube LLC, 2005) y varias críticas en revistas virtuales especializadas de todo el mundo como *The VR Grid* (Ouellette, 2021), *Desconsolados* (Calzada, 2021), *Cosas de Chicas Gamers* (Sánchez, 2021), o *Guardado Rápido* (Cervantes, 2021).

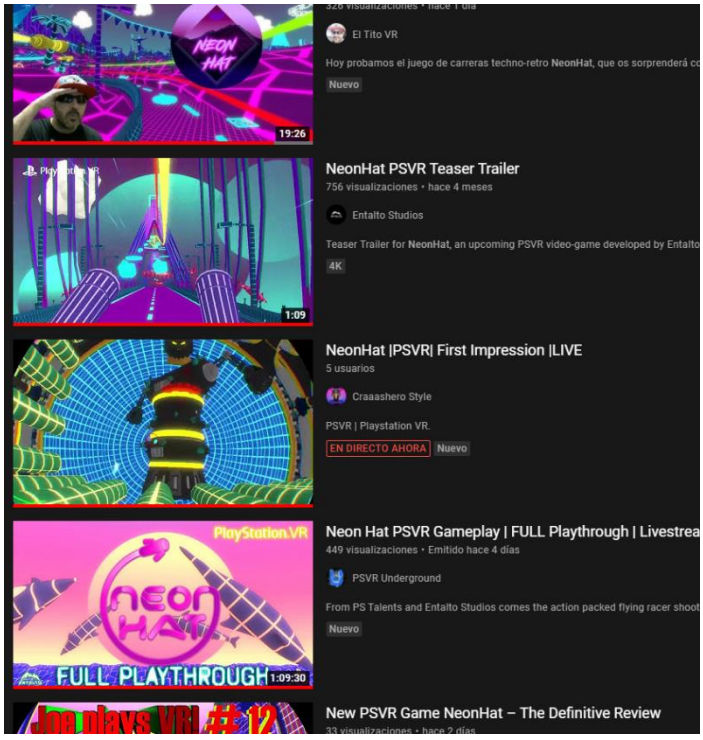


Ilustración 65: Vídeos en YouTube sobre NeonHat. Fuente: Propia.

En cuanto al futuro del juego; tenemos pensado que salga para diferentes plataformas en algún momento del futuro y no sabemos si habrá continuaciones del juego, ya que esto dependerá de la acogida por el público.

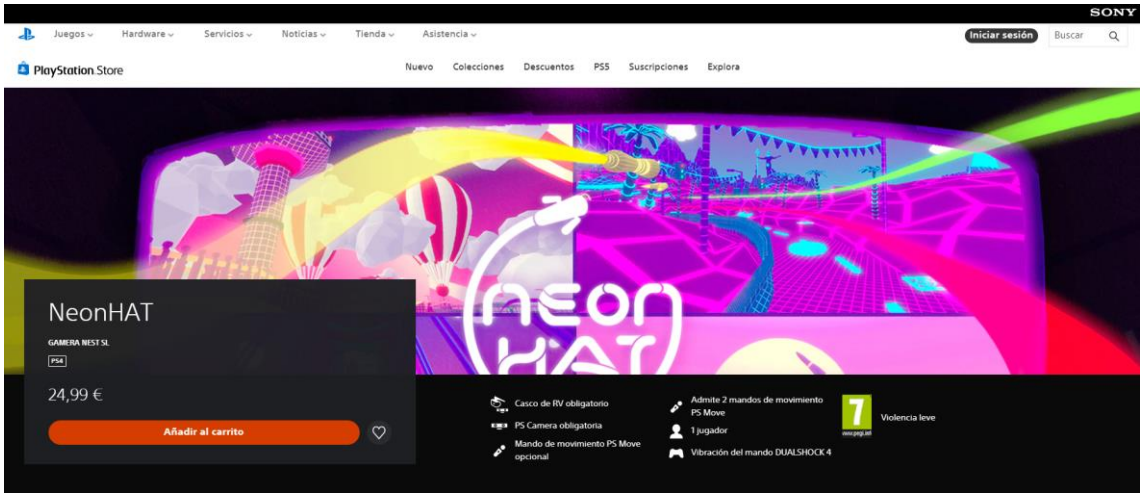


Ilustración 66: NeonHat disponible para su compra en la PlayStation Store. Fuente: https://store.playstation.com/es-es/product/EP0811-CUSA25749_00-8213788485881595/

9. Conclusiones

Haber sacado un videojuego al mercado es un logro bastante importante y un gran reto que todos los miembros del equipo hemos superado juntos. Que nuestro primer título, además, se haya publicado en la *PlayStation Store* es bastante importante.

Estoy muy contento con el resultado y creo que es un juego divertido. Además, haberlo desarrollado y publicado en tan poco tiempo nos demuestra que el saber prever los problemas y analizar cómo avanza el desarrollo es algo realmente importante, ya que gracias a estos esfuerzos nos dimos cuenta a tiempo de que había que cambiar por completo el diseño del juego y pivotar en la idea base del juego si no queríamos atascarnos con este proyecto durante más de lo previsto.

A lo largo de este desarrollo hemos podido poner en práctica una gran cantidad de materias aprendidas a lo largo del grado, pero también hemos aprendido mucho sobre cómo desarrollar un proyecto real. Estoy seguro de que si volviéramos a comenzar el proyecto no cometeríamos el desacierto de intentar la primera versión del juego. Viéndolo con perspectiva, se escapaba de nuestro alcance. Era un proyecto inabarcable para un equipo tan pequeño y nuevo en el desarrollo de videojuegos como el nuestro. Cuando nos dimos cuenta de este error, nadie quiso utilizar en adelante el trabajo que ya no hiciera falta; en mi caso, el sistema de Navegación; sino al contrario: ninguno entendíamos cómo no habíamos sido capaces de ver antes la gran falta de arte que íbamos a sufrir para llenar una ciudad entera. Y lo importante era sacar el videojuego adelante en su mejor versión posible.

Sin embargo; también hicimos cosas bien, como utilizar una metodología ágil que nos permitió darnos cuenta de este error cuando aún no era demasiado tarde y mantener a todo el equipo informado en todo momento del estado del juego y lo que se esperaba en el momento de cada uno; incluyendo a las personas encargadas de todo el trabajo externalizado.

Además, fuimos capaces de mostrar lo prometido en todas las presentaciones y reuniones con *PlayStation* España y Lanzadera (LANZADERA EMPRENDEDORES, S.L.U., 2011), algo que ha sido esencial en el buen trato que recibimos por ambas partes tanto al tener que parar la producción del juego como a la hora de pedir recursos (como el *Development Kit*).

Por último, he de decir que la sensación de ver el resultado de tanto trabajo funcionando correctamente mientras otras personas lo juegan y disfrutan es sin duda una de las razones por las que quiero seguir desarrollando videojuegos y, desde luego, este ha sido un punto de partida ideal.

10. Bibliografía

Aristidou, A., Lasenby, J., Chrysanthou, Y., & Shamir, A. (septiembre de 2018). Inverse Kinematics Techniques in Computer Graphics: A Survey. *Computer Graphics forum, 37*(6), 35-58. Obtenido de https://onlinelibrary.wiley.com/doi/10.1111/cgf.13310

Arribas, A. (20 de abril de 2017). Análisis de Mario Kart 8 Deluxe (Switch). *Vandal Videojuegos*. Obtenido de https://vandal.elespanol.com/analisis/switch/mario-kart-8-deluxe/45256#p-83

Beat Games. (21 de mayo de 2019). Beat Saber. Obtenido de https://beatsaber.com

Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. *IBM SYSTEMS JOURNAL, 4*, 25-30. Obtenido de https://web.archive.org/web/20080528040104/http://www.research.ibm.com/journal/sj/041/ibmsjIVRIC.pdf

Calzada, L. M. (3 de agosto de 2021). *Análisis de NeonHat - PSVR ··· Desconsolados.* Obtenido de DESCONSOLADOS: NeonHat: https://www.desconsolados.com/analisis/neonhat/

Camacho, M. (2021). *El coste de un trabajador para la empresa [+ fórmula + vídeo] - Factorial*. Obtenido de Factorial Blog: https://factorialhr.es/blog/coste-empresa-trabajador/

Camouflaj. (3 de julio de 2020). Marvel's Iron Man VR. Obtenido de https://store.playstation.com/es-es/product/EP9000-CUSA16206_00-MIMVRGAME0000001

Campana, N. (23 de mayo de 2018). *¿Qué distingue a un programador junior de un programador senior?* Obtenido de freelancermap: https://www.freelancermap.com/blog/es/diferencias-programador-junior-y-senior/

Cervantes, R. (06 de agosto de 2021). *Análisis de NeonHat (PS4/PSVR). Un juego de carreras para PSVR de corte ochentero - Guardado Rápido.* Obtenido de Guardado Rápido: Análisis de NeonHat (PS4/PSVR). Un juego de carreras para PSVR de corte ochentero: https://www.guardadorapido.com/analisis-de-neonhat-ps4-psvr/

Cervera, I. (12 de abril de 2019). *Geekno | Gameplay*. Obtenido de Gameplay | ¿Qué significa Gameplay? | Vocabulario gamer: https://www.geekno.com/glosario/gameplay

Cleveland Clinic. (18 de enero de 2021). *Motion Sickness: Symptoms & Treatment*. Obtenido de Cleveland Clinic: Motion Sickness: https://my.clevelandclinic.org/health/articles/12782-motion-sickness#:~:text=Motion%20sickness%20occurs%20when%20your,or%20airsick%20is%20motion%20sickness.

Dinamita Works. (2017-2021). Adroneline. Madrid, Madrid, España.

Dinamita Works. (7 de enero de 2021). *ADroneline -The Game en Twitter*. Obtenido de Twitter: https://twitter.com/ADroneline/status/1347239401764642822

Discord Inc. (2015-2021). *Discord | Tu sitio para hablar y pasar el rato*. Obtenido de Discord: https://discord.com

Entalto Games S.L. (2021). *Entalto Studios | Videogame Development*. Obtenido de Entalto Studios: Qué es SCRUM – Proyectos Ágiles

Entalto Games S.L. (2021). *Projects - ENTALTO STUDIOS*. Obtenido de Entalto Studios: Projects: https://www.entaltostudios.com/projects

Entertainment Software Association. (1994). *ESRB Game Ratings - ESRB Ratings.* Obtenido de ESRB: Entertainment Software Rating Board: https://www.esrb.org

Estévez, C. (septiembre de 2015-2021). *Project management for game development - HacknPlan*. Obtenido de HacknPlan: Where project management meets game design: https://hacknplan.com

ESTUDIOS SUPERIORES INTERNACIONALES, S.L. (1988-2021). *ESNE - Escuela Universitaria de Diseño, Innovación y Tecnología*. Obtenido de ESNE: https://www.esne.es

F. Coelho, F. (4 de agosto de 2019). *¿Qué es un QA y por qué no debe faltar en un proyecto de desarrollo web? - DIGITAL55.* Obtenido de DIGITAL55| ¿Qué es un QA y por qué no debe faltar en un proyecto de desarrollo web?: https://www.digital55.com/business/qa-no-debe-faltar-proyecto-desarrollo-web/

Ferro, M. (23 de abril de 2021). *▷¿Cuál es el precio del agua en España y quién lo regula?* Obtenido de Tarifasdeagua by Selectra: Precio de agua en España: Toda la información: https://tarifasdeagua.es/info/precio

Firelight Technologies Pty Ltd. (1995-2021). FMOD. Melbourne, Australia. Obtenido de https://www.fmod.com/unity

Gammera Nest. (2013). *Gammera Nest*. Obtenido de Gammera Nest: https://gammeranest.com

Gómez Baray, K. (2 de febrero de 2020). *qué es un bug | El Economista.* Obtenido de El Economista | ¿Qué es un bug y cómo afecta a tu computadora?: https://www.eleconomista.com.mx/tecnologia/que-es-un-bug-20200131-0079.html

Gómez, L. (3 de abril de 2013). *Cine para gamers: crítica de Tron - HobbyConsolas Juegos.* Obtenido de HobbyConsolas: Cine para gamers: crítica de Tron: https://www.hobbyconsolas.com/reviews/cine-para-gamers-critica-tron-49786

Gupta, L. (22 de octubre de 2012). *Java Singleton Pattern Explained - HowToDoInJava*. Obtenido de HowToDoInJava: Java Singleton Pattern Explained: https://howtodoinjava.com/design-patterns/creational/singleton-design-pattern-in-java/

Hitmaker. (1999). Crazy Taxi.

Hobby Industria. (31 de diciembre de 2017). Juegos de deporte, escape room, disparos y puzles para PS4 y PSVR. *PlayStation Talents*. Obtenido de https://www.playstationtalents.es/blog-item/juegos-de-deporte-escape-room-disparos-y-puzles-para-ps4-y-psvr

HTC Corporation. (15 de mayo de 1997). *HTC.* Obtenido de HTC: https://www.htc.com/

Indeed. (2004-2021). *Ofertas de trabajo, bolsa de trabajo | Buscar empleo en Indeed España*. Obtenido de indeed: https://es.indeed.com/

Janssens, D. (23 de julio de 2015). *What is a bitmask and a mask? - Stack Overflow*. Obtenido de Stack Overflow: What is a bitmask and a mask?: https://stackoverflow.com/questions/31575691/what-is-a-bitmask-and-a-mask

JetBrains s.r.o. (2000-2021). *Comprar Rider: Precios y licencias, Descuentos - Suscripción a JetBrains Toolbox*. Obtenido de JetBrains Rider: https://www.jetbrains.com/es-es/rider/buy/#personal?billing=monthly

Jimeno Navarro, S. (2020). *Diseño e Implementación de un Sistema de Navegación en Entornos de Realidad Virtual para Videojuegos Comerciales.* Proyecto Final de Grado en Diseño y Desarrollo de Videojuegos, Zaragoza. Obtenido de https://repositorio.usj.es/bitstream/123456789/432/1/Sergio%20Jimeno%20-%20Diseño%20e%20Implementación%20de%20un%20Sistema%20de%20Navegación%20en%20Entornos%20de%20Realidad%20Virtual%20para%20Videojuegos%20Comerciales.pdf

Kircher, M., & Jain, P. (2002). *Pooling.* Obtenido de Pooling: http://www.kircher-schwanninger.de/michael/publications/Pooling.pdf

LANZADERA EMPRENDEDORES, S.L.U. (21 de 03 de 2011). *Aceleradora e incubadora de empresas - Lanzadera*. Obtenido de LANZADERA: https://lanzadera.es

LANZADERA EMPRENDEDORES, S.L.U. (2019-2021). *terminos-condiciones-playstation-190502.pdf*. Obtenido de PROGRAMA "CORPORATE CON SONY PLAYSTATION": https://inscripcion.lanzadera.es/hubfs/PLAY-STATION/terminos-condiciones-playstation-190502.pdf

Lee, S., Kirby, J., Lieber, L., & Heck, D. (marzo de 1963). Iron Man. *Tales of Suspense, 39*.

Lisberger, S. (Dirección). (1982). *Tron* [Película].

Microsoft Corporation. (1997-2019). *IDE de Visual Studio, editor de código, Azure DevOps y App Center - Visual Studio*. Obtenido de Microsoft | Visual Studio: https://visualstudio.microsoft.com/es/

Microsoft Corporation. (2000). *Documentos de C#: inicio, tutoriales y referencias. | Microsoft Docs*. Obtenido de Microsoft | Docs: Documentación de C#: https://docs.microsoft.com/es-es/dotnet/csharp/

Microsoft Corporation. (2011-2021). *Comparar todos los planes de Microsoft 365 (anteriormente Office 365): Microsoft Store*. Obtenido de Microsoft Office 365: https://www.microsoft.com/es-es/microsoft-365/buy/compare-all-microsoft-365-products?rtc=1

Microsoft Corporation. (20 de julio de 2015). *abstract: Referencia de C# | Microsoft Docs*. Obtenido de Microsoft | Docs: abstract (Referencia de C#): https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/abstract

Microsoft Corporation. (20 de julio de 2015). *Microsoft | Docs: espacio de nombres (Referencia de C#).* Obtenido de Palabra clave namespace: Referencia de C#: https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/namespace

Microsoft Corporation. (2015-2021). *Buy & Download Windows 10 - Microsoft Store*. Obtenido de Microsoft Windows 10: https://www.microsoft.com/en-us/store/b/windows?activetab=tab%3ashopwindows10

Microsoft Corporation. (9 de abril de 2019). *Instrucción switch de C# | Microsoft Docs*. Obtenido de Microsoft | Docs: switch (referencia de C#): https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/switch

Microsoft Corporation. (13 de diciembre de 2019). *Microsoft | Docs: Tipos de enumeración (referencia de C#).* Obtenido de Tipos de enumeración: referencia de C# | Microsoft Docs: https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/builtin-types/enum

Microsoft Corporation. (Consultado en 2021). *List<T> Clase (System.Collections.Generic) | Microsoft Docs*. Obtenido de Microsoft | Docs: List<T> Clase: https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.list-1?view=net-5.0

Nintendo Company, Ltd. (19 de noviembre de 2006). *Accessories | Wii | Nintendo UK | Wii | Nintendo.* Obtenido de Nintendo: Wii Accessories: https://www.nintendo.co.uk/Wii/Accessories/Accessories-Wii-Nintendo-UK-626430.html

Nintendo Company, Ltd. (1992-2020). Mario Kart.

Nintendo Company, Ltd. (19 de noviembre de 2006). *Wii | Nintendo.* Obtenido de Nintendo: Wii: https://www.nintendo.es/Wii/Wii-94559.html

Nintendo Company, Ltd. (3 de marzo de 2017). *Choose Your Joy Con Color.* Obtenido de Nintendo Switch: Create your favorite color combo: https://www.nintendo.com/switch/choose-your-joy-con-color/

Nintendo Company, Ltd. (3 de marzo de 2017). *Nintendo Switch™ Family - Nintendo - Official Site.* Obtenido de Nontendo | Nintendo Switch: https://www.nintendo.com/switch/

Nintendo Entertainment Analysis and Development. (14 de noviembre de 2005). Mario Kart DS.

O2 España. (2018-2021). *O2 | 300Mb de fibra y 10GB para tu móvil con la mejor cobertura.* Obtenido de O2: Tarifa Fibra 300Mb y Móvil 10GB: https://o2online.es/fibra-y-movil-300/

Oculus VR. (julio de 2012). *Oculus | Equipos, juegos y gafas de realidad virtual.* Obtenido de Oculus: https://www.oculus.com

Ouellette, R. (3 de agosto de 2021). *NeonHAT - THE VR GRID.* Obtenido de THE VR GRID: NeonHAT: https://www.thevrgrid.com/neonhat/

Pan European Game Information. (2003). *Home | Pegi Public Site.* Obtenido de PEGI: https://pegi.info/es

Proyectos Robóticos. (14 de marzo de 2010). *Robótica: Bresenham 3D.* Obtenido de Robótica - Bresenham 3D: https://sites.google.com/site/proyectosroboticos/bresenham/bresenham-3d

ProyectosAgiles.org. (Consultado en 2021). *Qué es SCRUM – Proyectos Ágiles*. Obtenido de ProyectosAgiles.org: Qué es SCRUM: https://proyectosagiles.org/que-es-scrum/

Psygnosis / SCE Studio Liverpool. (1995-2017). Wipeout.

Quesada, D. (04 de mayo de 2018). *Qué significa que un juego ya es Gold - Hobby Basics - HobbyConsolas Entretenimiento.* Obtenido de HobbyConsolas - Qué significa que un juego ya es Gold - Hobby Basics: https://www.hobbyconsolas.com/reportajes/que-significa-que-juego-ya-es-gold-hobby-basics-206308

Roosendaal, T., & others. (1998-2021). *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. Obtenido de blender: https://www.blender.org

Rosa Fernández, B. (21 de octubre de 2019). *¿Qué es un Devkit para consolas? Con esto se crean los juegos - GamEsfera.* Obtenido de gamesfera | ¿Qué es un Devkit para consolas? Con esto se crean los juegos: https://www.gamesfera.es/noticias/xboxone/que-es-un-devkit-para-consolas-asi-se-crean-los-juegos/

Sánchez, S. (06 de agosto de 2021). *NEONHAT - ANÁLISIS EN PS VR | Cosas de Chicas Gamers.* Obtenido de Cosas de Chicas Gamers: NEONHAT - ANÁLISIS EN PS VR: https://www.chicasgamers.com/2021/08/analisis-review-neonhat-psvr.html

Schwaber, K., & Sutherland, J. (1991-2016). *2016-Scrum-Guide-Spanish.* Obtenido de La Guía de Scrum: La Guía Definitiva de Scrum: Las Reglas del Juego: https://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf#zoom=100

Simogo. (septiembre de 2019). Sayonara Wild Hearts. Obtenido de https://simogo.com/work/sayonara-wild-hearts/

Sony Interactive Entertainment. (2010). *PlayStation Move motion controller US*. Obtenido de PlayStation: PlayStation Move motion controller: https://www.playstation.com/en-us/accessories/playstation-move-motion-controller/

Sony Interactive Entertainment. (15 de noviembre de 2013). *Mando inalámbrico DUALSHOCK 4 ES*. Obtenido de PlayStation: Mando inalámbrico DUALSHOCK 4: https://www.playstation.com/es-es/accessories/dualshock-4-wireless-controller/

Sony Interactive Entertainment. (15 de noviembre de 2013). *PS4*. Obtenido de PS4 | Juegos increíbles, entretenimiento sin fin | PlayStation: https://www.playstation.com/es-es/ps4/

Sony Interactive Entertainment. (13 de octubre de 2016). *PlayStation VR | Vive los juegos en increíbles mundos de realidad virtual | PlayStation ES*. Obtenido de PlayStation: PlayStation VR: https://www.playstation.com/es-es/ps-vr/

Sony Interactive Entertainment Europe. (2015-2021). *PlayStation Talents*. Obtenido de PlayStation Talents: https://www.playstationtalents.es

Sony Interactive Entertainment Europe Ltd. (29 de julio de 2021). *NeonHAT.* Obtenido de PlayStation Store - NeonHAT: https://store.playstation.com/es-es/product/EP0811-CUSA25749_00-8213788485881595/

Sony Interactive Entertainment Inc. (16 de noviembre de 1993). *SIE Top Page | Sony Interactive Entertainment Inc.* Obtenido de Sony Interactive Entertainment - PlayStation: https://www.sie.com

Sony Interactive Entertainment Inc. (11 de noviembre de 2006). *PlayStation®3 España .* Obtenido de PlayStation®3: https://www.playstation.com/es-es/support/hardware/ps3/

Sony Interactive Entertainment Inc. (octubre de 2007). *Sony Cámara Playstation Eye PS3: Amazon.es: Videojuegos.* Obtenido de Amazon: Sony Cámara Playstation Eye PS3: https://www.amazon.es/Sony-Cámara-playstation-eye-ps3/dp/B000W3YQ1Y

Sony Interactive Entertainment Inc. (15 de agosto de 2010). *Mando de movimiento PlayStation Move España.* Obtenido de Mando de movimiento PlayStation Move: https://www.playstation.com/es-es/accessories/playstation-move-motion-controller/

Sony Interactive Entertainment Inc. (15 de noviembre de 2013). *PlayStation Camera | Retransmite tus partidas al mundo y conéctate a PS VR España .* Obtenido de PlayStation Camera: https://www.playstation.com/es-es/accessories/playstation-camera/

The Apache Software Foundation. (2000-2021). *Apache Subversion*. Obtenido de SUBVERSION: Apache® Subversion®: https://subversion.apache.org

Turner, D. (1 de julio de 2007). *Hack: The Nintendo Wii | MIT Technology Review.* Obtenido de MIT Technology Review - Hack: The Nintendo Wii: https://www.technologyreview.com/2007/07/01/271887/hack-the-nintendo-wii/

TutorialsTeacher. (10 de mayo de 2020). *C# Arrays (With Easy Examples)*. Obtenido de TutorialsTeacher C# Arrays: https://www.tutorialsteacher.com/csharp/array-csharp#:~:text=An%20array%20is%20the%20data,%2C%20multidimensional%2C%20and%20jagged%20array.

Unity Technologies. (2005-2021). *Unity*. Obtenido de Unity Real-Time Development Platform | 3D, 2D VR & AR Engine: https://unity.com

Unity Technologies. (2016). *Unity - Manual: Corrutinas*. Obtenido de Unity | Documentation: Corrutinas: https://docs.unity3d.com/es/530/Manual/Coroutines.html

Unity Technologies. (31 de julio de 2018). *Unity - Manual: Prefabs*. Obtenido de Unity | Documentation: Prefabs: https://docs.unity3d.com/Manual/Prefabs.html

Unity Technologies. (2020). *About Localization | Localization | 0.10.0-preview .* Obtenido de Unity | About Localization: https://docs.unity3d.com/Packages/com.unity.localization@0.10/manual/index.html

Unity Technologies. (2020). *Unity - Manual: Occlusion culling*. Obtenido de Unity | Documentation: Occlusion culling: https://docs.unity3d.com/Manual/OcclusionCulling.html

Unity Technologies. (2020). *Unity - Manual: Profiler overview*. Obtenido de Unity | Documentation: Profiler overview: https://docs.unity3d.com/Manual/Profiler.html

Unity Technologies. (23 de 08 de 2021). *Unity - Scripting API: PlayerPrefs.* Obtenido de Unity | Documentation | PlayerPrefs: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html

Unity Technologies. (03 de agosto de 2021). *Unity | Documentation: LayerMask.* Obtenido de Unity - Scripting API: LayerMask: https://docs.unity3d.com/ScriptReference/LayerMask.html

Unity Technologies. (06 de agosto de 2021). *Unity | Documentation: Time.* Obtenido de Unity - Scripting API: Time: https://docs.unity3d.com/ScriptReference/Time.html

Valve. (2015-2021). *SteamVR en Steam*. Obtenido de Steam: SteamVR: https://store.steampowered.com/app/250820/SteamVR/

Wagner, B., & "olprod". (17 de marzo de 2021). *Directivas de preprocesador de C# | Microsoft Docs*. Obtenido de Microsoft | Docs: Directivas de preprocesador de C#: https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/preprocessor-directives

Wolf, M. J. (2008). Glossary. En M. J. Wolf, *The Video Game Explosion: A History from PONG to Playstation and Beyond* (pág. 315). Westport: United States of America. Obtenido de https://books.google.es/books?id=XiM0ntMybNwC&pg=PA315&redir_esc=y#v=onepage&q&f=false

YouTube LLC. (14 de febrero de 2005). *YouTube*. Obtenido de neonhat psvr - YouTube: https://www.youtube.com/results?search_query=neonhat+psvr

11. Tablas de figuras

11.1. Tabla de ilustraciones

11.2. Tabla de tablas

12. Anexos

12.1. Propuesta de proyecto final

Nombre alumno: Javier Verón Mérida
Titulación: Grado en Diseño y Desarrollo de Videojuegos
Curso académico: 5º (Doble grado)

1. TÍTULO DEL PROYECTO

Adiciones, optimización y sistema de navegación para un sistema de vuelo 3D en un proyecto real de Realidad Virtual.

2. DESCRIPCIÓN Y JUSTIFICACIÓN DEL TEMA A TRATAR

Entalto Studios está creando Neon Hat: un juego arcade para Realidad Virtual de PS4 en el que vuelas por una ciudad ciberpunk para cumplir diferentes misiones cortas.

Para ello, se utilizará como base un sistema de vuelo ya desarrollado por el equipo que está orientado para campo abierto en PC (utilizando HTC Vive o similares) y se adaptará para la Realidad Virtual de PS4 y para ser eficiente como mecánica principal de un juego arcade. También, al desarrollar tareas cortas en una ciudad grande, será necesario un sistema de guiado eficiente.

Este proyecto se va a centrar en la adaptación del sistema de vuelo y la implementación del sistema de guiado para una ciudad 3D.

3. OBJETIVOS DEL PROYECTO

El objetivo principal es adaptar el sistema de vuelo existente para este nuevo videojuego, incluyendo:

- Controles adaptados para PS4 VR.
- Sistema de guiado y búsqueda de rutas en la ciudad.
- Optimizaciones y mejoras del sistema de vuelo (derrapes, frenazos, estabilización).
- Pruebas de rendimiento del sistema en la consola PS4.

4. METODOLOGÍA

Será definida cuando el proyecto empiece.

5. PLANIFICACIÓN DE TAREAS

Será definida cuando el proyecto empiece.

6. OBSERVACIONES ADICIONALES

El proyecto será dirigido por Jaime Font.

12.2. Reuniones

1. **Elabora acta:** Javier Verón
   Convocados: Jaime Font
   Fecha: 13 de marzo de 2020.
   Modo: online.
   Duración: 1h.
   Programa: Microsoft Teams.
   Contenido: Se han hablado unas primeras ideas sobre el tema del TFG.

2. **Elabora acta:** Javier Verón
   Convocados: Jaime Font
   Fecha: 22 de septiembre de 2020.
   Modo: online.
   Duración: 1h.
   Programa: Microsoft Teams.
   Contenido: Comienzo del TFG. Idea definida, ya que hemos hablado de que el trabajo se enfocará finalmente en mi trabajo como programador para el videojuego de PlayStation VR NeonHat, desarrollado por Entalto Studios.

3. **Elabora acta:** Javier Verón
   Convocados: Jaime Font
   Fecha: 06 de octubre de 2020.
   Modo: online.
   Duración: 1h.
   Programa: Microsoft Teams.
   Contenido: Hemos hablado de los primeros pasos del proyecto y cómo va a ser el videojuego.

4. **Elabora acta:** Javier Verón
   Convocados: Jaime Font
   Fecha: 27 de octubre de 2020.
   Modo: online.
   Duración: 1h.
   Programa: Microsoft Teams.
   Contenido: Reunión para ver cómo ha avanzado este mes el desarrollo.

5. **Elabora acta:** Javier Verón
   Convocados: Jaime Font
   Fecha: 4 de marzo de 2021.
   Modo: online.
   Duración: 1h.
   Programa: Microsoft Teams.
   Contenido: El alcance del videojuego se ha visto reducido y ha cambiado por completo el diseño de este, por lo que esta reunión ha sido para hablar sobre los cambios que ha sufrido tanto el juego como mi TFG.

6. **Elabora acta:** Javier Verón
   Convocados: Jaime Font
   Fecha: 28 de junio de 2021.
   Modo: online.
   Duración: 1h.
   Programa: Microsoft Teams.
   Contenido: Se ha preparado una estructura de la que partir para la elaboración de la memoria.