Universidad de Valladolid

Escuela de Ingeniería Informática (Segovia)

Grado en Ingeniería Informática de Servicios y Aplicaciones

Unreal Engine 4: Desarrollo de Videojuegos en 3D

Alumno: Manuel Sastre Cebrián

Tutor: Luis María Fuentes García

ÍNDICE

Índice de ilustraciones:

Glosario Tablas:

AGRADECIMIENTOS:

A mis padres y a Laura. Unos por ayudarme y apoyarme en aquello que de verdad me importa y a ti, Laura, por estar a mi lado durante todos estos años de esta travesía. Cada uno con su granito de arena.

A todos mis compañeros de la Universidad. Los viejos y los nuevos. Adrián, Ángel, Miguel, Jorge, Raquel, Dani, Libertad, Marta, Isabel, Pablo, etc… y el resto de amigos que siempre estuvieron para salir a tomar algo durante tantos años.

A los profesores que me abrieron las puertas de la informática: Miguel Ángel y Anibal.

Vosotros me enseñasteis más de lo que podría haber pedido. En aquella clase de Multimedia descubrí mi verdadera pasión y camino.

Gracias.

No intentes seguir en el camino que te impone quien no te conoce. El destino es tu futuro.

-Artyom

ABSTRACT

Este documento contendrá lo relativo a un trabajo de fin de grado relacionado con la creación de un videojuego para ordenador mediante el uso del motor gráfico Unreal Engine 4. Se ha elegido este software debido a su potencia y versatilidad, puesto que permite desarrollar videojuegos para cualquier plataforma. Además, el proyecto incluirá un Documento de Diseño del Videojuego (Game Design Document).

El videojuego pertenece al género Aventura 3D, donde el jugador avanzará a lo largo de niveles tridimensionales para completar puzles y superar enemigos que tratarán de frenar su avance.

This document contains a thesis related to the creation of a videogame for computer using the graphic motor Unreal Engine 4. This software has been chosen due to its power and versatility, since it allows to develop videogames suitable for any platform. Furthermore, this project includes a GDD (Game Design Document).

The videogame belongs to the genre 3D Adventure, where the player advances through tridimensional levels in order to complete puzzles and overcome enemies who will try to stop his progress.

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

El mundo de los videojuegos siempre ha sido para mí una de las partes más atractivas y creativas de la informática.

Desde el año 2015 los motores gráficos más importantes del mercado han comenzado a liberalizar las licencias de uso, permitiendo su utilización fuera de las grandes empresas o los estudios con recursos. El poder usar Unreal Engine 4 sin tener que pagar una licencia me hizo pensar sobre por donde tenía que ir mi TFG y me animó la idea de investigar e introducirme más en el mundo de la informática gráfica.

Al comprobar que el diseño de videojuegos no es un tema muy tratado en los TFG de la escuela y tras hacer un estudio comparativo de motores gráficos, me decidí por la creación de un videojuego con la herramienta Unreal Engine 4.

1.2. OBJETIVOS

El objetivo de este TFG es desarrollar una versión funcional de un videojuego mediante Unreal Engine 4. Se creará toda la estructura y componentes de un videojuego. Los subobjetivos de este TFG son:

- Diseñar el videojuego: movimientos del personaje, enemigos, elementos del escenario.
- Crear la interfaz gráfica del videojuego: HUD (Heads-Up Display) del personaje y menús.
- Crear la banda sonora y los sonidos del videojuego.
- Crear los diferentes niveles del videojuego: el jugador podrá moverse a lo largo del mundo.
- Crear un sistema de guardado: se podrá guardar parte de los avances.
- Crear el hilo argumental del videojuego.

1.3. ESTADO DEL ARTE

En esta sección se hará un pequeño análisis de la industria de los videojuegos en la actualidad.

Según un estudio de *NewZoo Global Games*[1] los beneficios han aumentado desde el 2012 en casi un 32% y se estima que en el año 2017 la industria de los videojuegos genere casi 103 mil millones de dólares y ya ha superado al cine como industria de creación de contenidos.

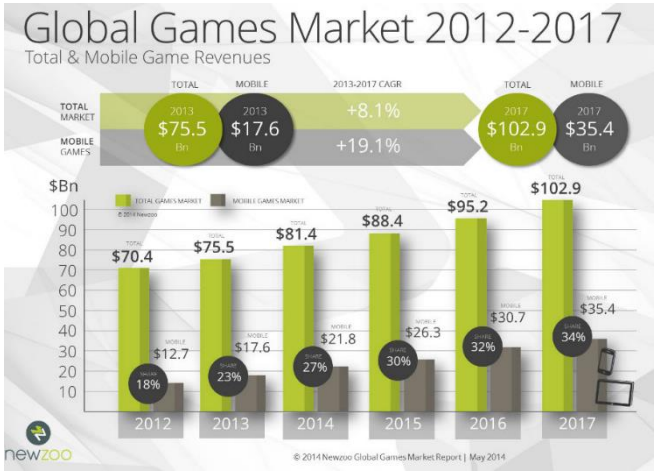


Ilustración 1: mercado global de videojuegos 1

Este mismo estudio realiza una comparación[2] entre plataformas de videojuegos (ordenador, smartphone, videoconsolas de sobremesa y videoconsolas portátiles). Se puede observar que el ordenador sigue liderando los ingresos anuales, aunque los dispositivos móviles tienen un fuerte crecimiento. Por el contrario, las consolas de sobremesa se mantienen estancadas.

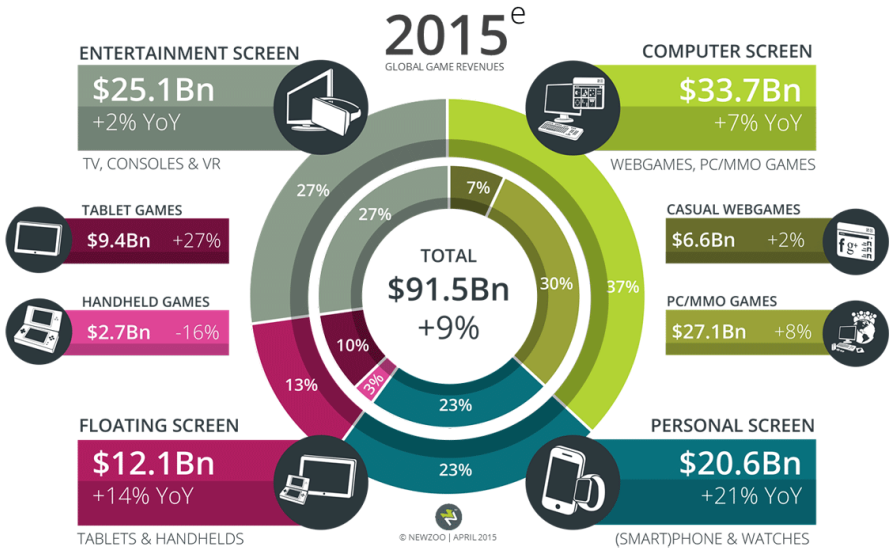


Ilustración 2: mercado global de videojuegos 2

Aunque los dispositivos móviles han irrumpido en el mercado con mucha fuerza, el ordenador sigue siendo la punta de lanza de una industria que no para de crecer año a año.

1.4. TECNOLOGÍA EMPLEADA

Para el desarrollo de este TFG se va a emplear las siguientes herramientas.

- Hardware:
  - Procesador Intel Core i5-3750k.
  - RAM 16 GB.
  - Tarjeta gráfica Nvidia 970 con memoria de 4GB GDDR5.
- Software:
  - Microsoft Office 2016.
  - Windows 10.
  - Blender.
  - Unreal Engine 4.
  - Autodesk.
  - Space Scape.
- Otros:
  - Piano electrónico: composición de algunas partes de la música.
  - Mynoise.net: creación de sonidos y música.

1.5. CONTENIDO DEL CD-ROM

El CD-ROM que se entrega junto a la documentación tendrá en su interior 3 carpetas:

- Software: contendrá la totalidad del proyecto exportado de Unreal Engine 4.
- Documentación: Contendrá la versión en PDF de este documento.
- Recursos: Se añadirán los archivos comprimidos de aquellas bibliotecas de libre distribución que se han usado para la realización de este TFG.

2. PLANIFICACION Y PRESUPUESTO

2.1. METODOLOGÍA

El TFG seguirá un modelo incremental. Cada mes de desarrollo será una iteración y se irán añadiendo funcionalidades con el propósito de mejorar el resultado final. Este tipo de desarrollo es muy utilizado en la actualidad por los equipos de desarrollo de videojuegos pequeños, ya que permite crear versiones jugables de cara a que los usuarios finales puedan probarlo antes del lanzamiento.

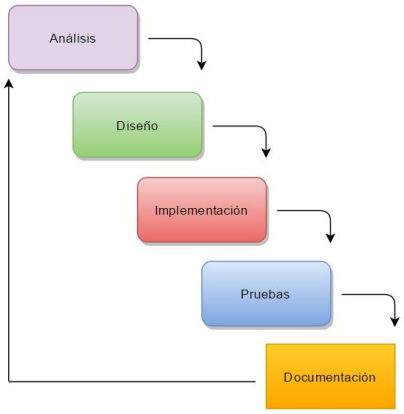


Ilustración 3: Modelo Incremental

Estas versiones jugables se conocen como *Alphas* y *Betas* dependiendo de lo completo que esté el desarrollo. También se acuña el término *Early Access*[3] para aquellas ocasiones en las que los ciclos del modelo incremental se acortan y se lanzan versiones jugables cada menos tiempo.

2.2. DESARROLLO

El desarrollo del videojuego estará marcado por varias iteraciones, cada una de las cuales consta de las siguientes fases:

- Análisis.
- Diseño.
- Desarrollo.
- Implementación.
- Pruebas.
- Documentación.

Las iteraciones se planificaron de forma que cada una fuera un mes natural, como podemos ver en este diagrama de Gantt:

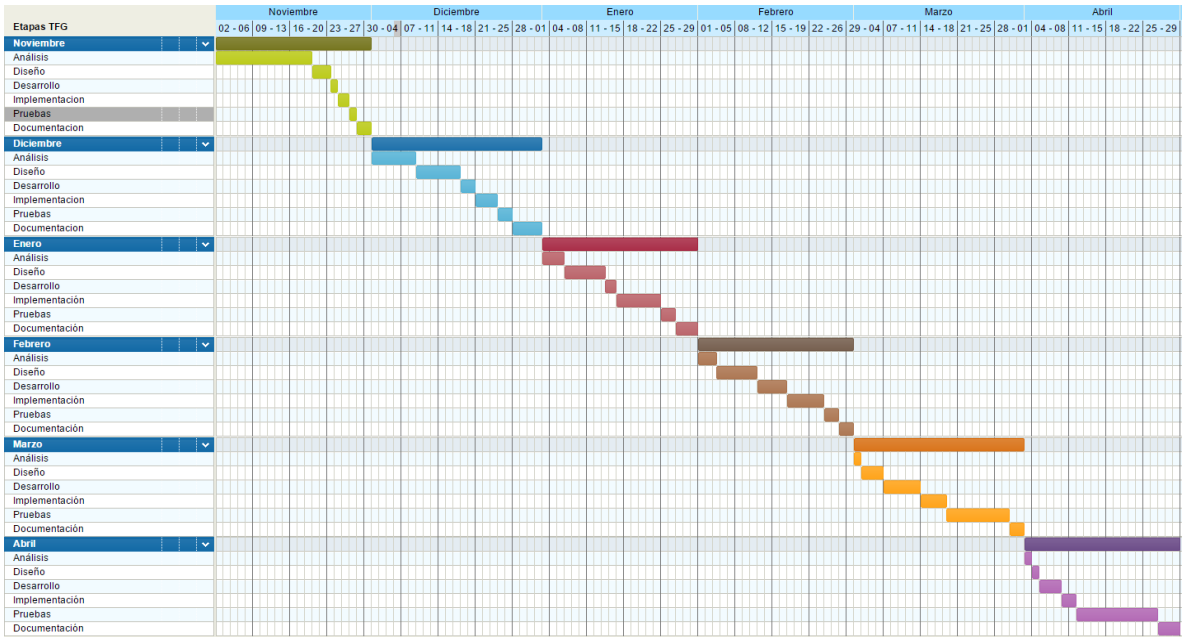


Ilustración 4: Diagrama de Gantt

- Primera iteración: enfocar el problema y comenzar la aproximación a las necesidades del sistema.
- Segunda iteración: se tendrá gran parte del diseño estructural realizado.
- Tercera iteración: se empezará a implementar el diseño del videojuego (mundo, personajes, ambientación...).
- Cuarta iteración: primera versión del videojuego. Toda la parte visual estará en su mayoría terminada. Se comenzará con la parte más técnica del TFG, trabajando en la parte funcional del videojuego.
- Quinta iteración: testeo y pruebas en entorno real. El videojuego estará en un estado de *beta* y será 100% funcional. Se probarán posibles errores y se realizarán las últimas tareas de programación de las diferentes acciones del videojuego.
- Sexta iteración: se realizarán pruebas con usuarios reales y se implementarán correcciones. Se terminará la documentación.

2.3. PRESUPUESTO INICIAL

Se tratará el coste económico y temporal de este proyecto teniendo en cuenta el diagrama de Gantt creado antes de comenzar el desarrollo del mismo.

Estará dividido en:

- Presupuesto Hardware: elementos físicos que son necesarios para el correcto desarrollo del proyecto (ordenadores, dispositivos móviles, conexiones de internet…).
- Presupuesto Software: coste total del software usado.
- Salarios: coste total de los trabajadores.

2.3.1. Presupuesto Hardware

Para el desarrollo del videojuego serán necesarios:

1. Ordenador personal: se requiere un ordenador con el sistema operativo Windows 10, la vida media de un ordenador y sus componentes es de 4 años. Se va a hacer un uso de 6 meses, es decir durante todo el desarrollo, por lo que:

Porcentaje de uso	Tiempo
100%	48 meses en 4 años
X%	6 meses
X = 12.5% de uso.	

Tabla 1: porcentaje de uso PC 1

2. Conexión a internet: UE4 necesita conexión a internet para poder actualizarse y poder acceder a la documentación y ayuda necesarias. El coste por mes es de 35€ y solo se usará la mitad del tiempo de desarrollo, por lo que:

Porcentaje de uso	Tiempo
100%	6 meses
X%	3 meses
X = 50% de uso.	

Tabla 2: porcentaje de uso Internet 1

3. Impresora: usaremos el modelo Ricoh SP112 para imprimir la documentación por valor de 40€.

4. Teclado: piano necesario para crear la música del videojuego. La vida media de un piano de estas características es de 5 años. Se utilizará durante un mes.

Porcentaje de uso	Tiempo
100%	60 meses (en 5 años)
X%	1 mes
X = 1.7 % de uso.	

Tabla 3: porcentaje uso piano 1

Total:

Hardware	Uso	Coste	Coste final (coste x uso)
Ordenador	12.5%	1300€	162€
Conexión a internet	50%	35 x 6 meses	105€
Impresora	100%	40	40€
Piano	1.7%	450	7,65€
Coste total			314.15€

Tabla 4: presupuesto hardware 1

2.3.2. Presupuesto Software

Para el desarrollo del videojuego se usarán los siguientes programas:

- Sistema operativo Windows 10 Pro.
- Unreal Engine 4.
- Blender.
- Visual Studio.
- Google Drive.
- Audacity.
- Space Scape.
- Autodesk.

Los programas UE4, Blender, Visual Studio, Google Drive, Audacity y Space Scape son gratuitos y no requieren una inversión económica. Autodesk tiene licencias mensuales, y aunque solo lo vayamos a usar durante una semana tenemos que pagar la mensualidad completa de 200€.

Software	Uso	Coste total	Coste final (coste x uso)
Windows 10	100%	45€	45€
Unreal Engine 4	-----	-----	0€
Blender	-----	-----	0€
Visual Studio	-----	-----	0€
Google drive	-----	-----	0€
Audacity	-----	-----	0€
Space Scape	-----	-----	0€
Autodesk	100%	200€	200€
Coste total			245€

Tabla 5: coste Software 1

2.3.3. Salarios

Este proyecto será realizado por una única persona, la cual llevará a cabo tres trabajos diferentes. Los salarios pertenecen a la media global[4] del año 2015

- Diseñador de videojuegos: 21€ por hora trabajada.
- Programador de videojuegos: 25€ por hora trabajada.
- Evaluador/testing y documentación de videojuegos: 11€ por hora trabajada.

Teniendo en cuenta que trabajará durante 8 horas a lo largo de los 5 días laborables de la semana tenemos un total de 1056 horas de trabajo a repartir según el diagrama de Gantt.

Trabajadores	*Horas trabajadas*	*Coste total (horas trabajadas x € por hora trabajada)*
Diseñador	440 h	9240€
Programador	495 h	12375€
Evaluador	120 h	1320€
Coste total	1056 h	22935€

Tabla 6: coste humano 1

2.3.4. Coste final

	Coste
Presupuesto Hardware	314.15€
Presupuesto Software	221€
Presupuesto humano	22935€
Coste total	23470.15€

Tabla 7: coste final 1

2.4. PRESUPUESTO REAL

En esta sección veremos el presupuesto teniendo en cuenta el tiempo real de desarrollo del TFG. El coste temporal ha aumentado de 6 meses a 7 meses y 5 días.

Estará dividido en:

- Presupuesto Hardware: elementos físicos que son necesarios para el correcto desarrollo del proyecto (ordenadores, dispositivos móviles, conexiones de internet…).
- Presupuesto Software: coste total del software usado.
- Presupuesto de desarrollo: coste personal.

2.4.1. Presupuesto Hardware

Para el desarrollo del videojuego serán necesarios:

1. Ordenador personal: se ha utilizado un ordenador con el sistema operativo Windows 10. Teniendo en cuenta que la vida media de un ordenador y sus componentes es de 4 años y que se ha hecho uso del ordenador durante todo el proyecto:

Porcentaje de uso	Tiempo
100%	48 meses en 4 años
X%	7 meses y 5 días
X = 15.41% de uso.	

Tabla 8: porcentaje uso PC 2

2. Conexión a internet: UE4 necesita conexión a internet para poder actualizarse y poder acceder a la documentación y ayuda necesarias. El coste por mes es de 35€ y en la realidad se ha usado 7 meses:

Porcentaje de uso	Tiempo
100%	7 meses
X%	4 meses
X = 66.7% de uso.	

Tabla 9: porcentaje uso Internet 2

3. Impresora: se ha utilizado el modelo Ricoh SP112 para imprimir la documentación por valor de 40€.

4. Teclado: piano necesario para crear la música del videojuego. La vida media de un piano de estas características es de 5 años. Finalmente, el teclado ha sido usado solo durante medio mes.

Porcentaje de uso	Tiempo
100%	60 meses (en 5 años)
X%	0.5 mes
X = 0.84 % de uso.	

Tabla 10: porcentaje uso piano 2

Total:

Hardware	Uso	Coste	Coste final (coste x uso)
Ordenador	15.41%	1300€	200.33€
Conexión a internet	66.7%	35 x 7 meses	140€
Impresora	100%	40	40€
Piano	0.84%	450	3.78€
Coste total			384.11€

Tabla 11: coste Hardware 2

2.4.2. Presupuesto Software

Para el desarrollo del videojuego se usaron los siguientes programas:

- Windows 10 Pro.
- Unreal Engine 4.
- Blender.
- Visual Studio.
- Google drive.
- Audacity.
- Space Scape.
- Autodesk.

Los programas UE4, Blender, Visual Studio, Google Drive, Audacity y Space Scape son gratuitos y no requieren una inversión económica. Autodesk tiene licencias mensuales, y aunque solo lo hemos usado durante una semana hemos tenido que pagar la mensualidad completa de 200€.

El presupuesto Software, por tanto, no ha variado, ya que el único programa de coste lo hemos usado, como se presupuestó, una sola semana.

Software	Uso	Coste total	Coste final (coste x uso)
Windows 10	100%	45€	45€
Unreal Engine 4	-----	-----	0€
Blender	-----	-----	0€
Visual Studio	-----	-----	0€
Google drive	-----	-----	0€
Audacity	-----	-----	0€
Space Scape	-----	-----	0€
Autodesk	100%	200€	200€
Coste total			245€

Tabla 12: coste Software 2

2.4.3. Salarios

Este proyecto ha sido realizado por una única persona, la cual ha llevado a cabo tres trabajos diferentes.

- Diseñador de videojuegos: 21€ por hora trabajada.
- Programador de videojuegos: 25€ por hora trabajada.
- Evaluador/testing y documentación de videojuegos: 11€ por hora trabajada.

El desarrollo completo se ha realizado en 7 meses y 5 días, por lo que asciende a un total de 1216 horas de trabajo. Los roles de trabajo también han cambiado bastante ya que el tiempo de trabajo del programador y evaluador ha subido mientras que el de diseñador ha bajado. El resultado final es:

Trabajadores	*Horas trabajadas*	*Coste total (horas trabajadas x € por hora trabajada)*
Diseñador	340 h	7140€
Programador	695 h	17375€
Evaluador	181 h	1991€
Coste total	1056 h	26506€

Tabla 13: coste humano 2

2.4.4. Coste final

	Coste
Presupuesto Hardware	384.11€
Presupuesto Software	221€
Presupuesto humano	26506€
Coste total	27111.11€

Tabla 14: coste final 2

2.5. EJEMPLO REAL

En esta sección vamos a hablar del caso del videojuego *Shovel Knight*, realizado por el estudio *Yacht Club Games*. La razón por la que incluyo este ejemplo[5] es comparar el presupuesto estimado y el final con la realidad para ver cómo al final los tiempos y el coste de desarrollo de un videojuego son complicados de inferir.

Sus creadores decidieron llevar a cabo un pequeño estudio para mostrar el coste total del desarrollo económico del videojuego y cómo este dista mucho de cualquier estimación posible. Mostraron la fórmula que usaron para establecer el presupuesto inicial del videojuego:

Número de gente en el equipo x 10.000 dólares al mes

Estos 10.000 dólares incluyen tanto sueldo como seguro médico, impuestos y necesidades primarias del desarrollador (equipo, electricidad, agua, comida…), además de licencias, alquiler de suelo para mostrar el videojuego en ferias, testadores para las fases finales, publicidad, etc.

Esta cifra no es muy precisa, ya que en palabras del propio director de desarrollo «infravaloramos de manera criminal el tiempo y esfuerzo de cada miembro del equipo. Lo típico es compensar esto sobreestimando un poco el tiempo, salario y cantidades a la hora de calcular el presupuesto».

La estimación temporal fue de casi dos años, por lo que teniendo a 6 personas trabajando salió un presupuesto final de 1.440.000 dólares.

El problema de estos presupuestos es que solo son asumibles por grandes estudios, pero un estudio pequeño o con poco recorrido no se puede permitir esta cifra. En concreto, Yacht Club Games contaba con un presupuesto de 35.000 dólares.

Para poder cumplir los dos años de desarrollo se decidió aumentar las horas de trabajo diarias a casi 13 horas diarias durante los 7 días de la semana, además de bajar el sueldo mensual de los desarrolladores.

Aun así, el coste temporal seguía sin coincidir con el real y se decidió dividir el desarrollo en dos mitades. La primera parte del juego sería lanzada siguiendo el plan temporal establecido (dos años de desarrollo) y la otra mitad se lanzaría a lo largo de los siguientes años a modo de actualizaciones gratuitas.

Con esta idea se pretendió reducir el coste final de los dos primeros años de desarrollo y reinvertir el dinero obtenido tras la venta de la primera versión del videojuego.

3. VIDEOJUEGOS Y MOTORES GRÁFICOS

3.1. ¿QUÉ ES UN VIDEOJUEGO?

Un videojuego es un juego electrónico en el que una o varias personas interactúan a través de imágenes de vídeos con un dispositivo. Los videojuegos son actualmente la principal industria de arte y entretenimiento a nivel global.

3.2. ¿QUÉ ES UN MOTOR GRÁFICO?

Un motor gráfico es un software que permite el diseño, creación y representación de un videojuego. Su funcionalidad básica es la de proveer al videojuego de un motor de renderizado gráfico, motor de físicas, audio y animación de personajes, además de dar la posibilidad de reutilizar el contenido creado para un videojuego concreto en otros diferentes.

Los primeros motores gráficos surgieron en los años 90 y su finalidad era lograr una separación muy definida entre los diferentes componentes del videojuego. Diferenciar el núcleo del videojuego (sistema de colisiones, audio, sistema de renderizado 2D o 3D) del resto de recursos (el arte, los elementos físicos o las reglas de juego) permitía crear una estructura inicial y a partir de ella desarrollar varios videojuegos distintos.

Un ejemplo de esta tendencia fue la desarrolladora *Id Software*[6]. Su videojuego *Doom* fue creado mediante el uso del motor id *Tech 1*, y tras el éxito de esta primera parte desarrollaron los videojuegos *Doom 2*, *Hexen*, *Heretic*, *Strife* y *HacX* utilizando el mismo motor, por lo que pudieron reciclar el núcleo del *Doom* original. Es decir, a partir de un solo videojuego inicial lograron producir otros 5 diferentes, con el consiguiente ahorro de tiempo y dinero.

3.3. COMPARATIVA DE MOTORES GRÁFICOS

La proliferación de videoconsolas y la llegada de los videojuegos para teléfonos móviles ha resultado en la creación de un gran número de motores gráficos, cada uno con sus ventajas y sus inconvenientes.

A continuación, realizaré una comparativa entre 4 de los principales motores gráficos actuales.

3.3.1. Unity3D

Ilustración 5: Logotipo Unity

Unity3D es un motor gráfico multiplataforma creado por la empresa *Unity Technologies*. Actualmente se puede obtener de forma gratuita para uso personal, y permite el desarrollo de videojuegos para Windows, OS X, Linux, Xbox 360, PlayStation 3, PlayStation Vita, Wii, Wii U, iPad, iPhone, Android y Windows Phone. Esto lo convierte en uno de los motores con más compatibilidad con los dispositivos más usados en la actualidad.

Pros:

- Fácil de usar y compatible con todas las plataformas de juego.
- Comunidad muy grande.
- Fácil de aprender.
- Muchas salidas profesionales.

Contras:

- Solo es gratuito para uso personal.
- En comparación con otros motores, se necesita más tiempo para hacer juegos complejos.
- Está más dirigido a juegos para teléfonos móviles o pequeños.

3.3.2. Unreal Engine 4

Ilustración 6: Logotipo Unreal Engine 4

Unreal Engine 4 es un motor gráfico creado por la compañía *Epic Games*. Su primera aparición es en el año 1998 con el videojuego *Unreal*, y desde entonces ha sido la base para videojuegos de la talla de *Bioshock*, *Deus Ex* o *Mass Effect* entre otros.

Pros:

- Ofrece una comunidad gigantesca y cuenta con un amplio número de tutoriales en la plataforma YouTube (tanto oficiales como de desarrolladores).
- Actualizaciones constantes.
- Gran número de herramientas que facilitan el desarrollo.
- Compatible con la mayoría de plataformas de juego (menos Nintendo).
- Gratuito.

Contras:

- Complejo de aprender.
- Exige un equipo potente: configuraciones con menos de 16 Gb pueden tener problemas de rendimiento.

3.3.3. CryEngine V

Ilustración 7: Logotipo CryEngine

CryEngine V es un motor gráfico creado por la compañía alemana *Crytek*. En sus inicios fue un motor de *demostración* para la empresa gráfica *Nvidia*, pero al demostrar sus grandes capacidades técnicas se decidió usarlo en el videojuego *Far Cry*. Actualmente pertenece a *Ubisoft*.

Pros:

- Uno de los motores más fotorrealistas que podemos encontrar en el mercado.
- Cuenta con un apartado sonoro muy fuerte, de la mano de FMOD.
- Curva de aprendizaje muy pequeña para desarrollos pequeños.
- Gratuito.

Contras:

- Menor comunidad que los anteriores.
- Aunque es idóneo para proyectos grandes, el desarrollo es complejo.

3.3.4. HeroEngine

Ilustración 8: Logotipo Hero Engine

HeroEngine es un motor gráfico creado por la compañía *Simutronics Corporation*. Este motor está dirigido a la creación de videojuegos MMO (*Massively Multiplayer Online*) y una de sus características más importantes es la compatibilidad con servicios en la nube.

Pros:

- Optimizado para ambientes multijugador, permite la transición entre niveles de forma más rápida.
- Fácil y práctico: curva de aprendizaje muy baja si conoces este tipo de herramientas.
- Conexión con un servicio en la nube propio: HeroCloud.

Contras:

- Motor de scripting poco intuitivo.
- No es gratuito.
- Complejo para un desarrollador novato.

3.3.5. Havok Game Dynamics

Ilustración 9: Logotipo Havok

Aunque lo incluimos en la comparativa, *Havok Game Dynamics* es un motor de físicas, complemento para el desarrollo de videojuegos, y no sustituye a ningún motor gráfico. Fue creado por la empresa *Havok* y adquirido por Microsoft en el año 2015. Ha sido empleado en juegos como *Assasin's Creed*, *Fallout*, *DeusEx* o *Half-Life* entre otros.

Pros:

- Admite todas las plataformas a excepción de móviles.
- Facilita la creación de eventos dinámicos (explosiones, choques, interacciones con objetos).
- Libera de cálculos a la CPU.

Contras:

- No es gratuito.
- Curva de aprendizaje difícil.

3.4. ¿POR QUÉ UNREAL ENGINE 4?

La primera pregunta que nos hacemos como desarrolladores es qué motor gráfico elegir. En la actualidad existen muchos motores gráficos, siendo los más usados los mostrados con anterioridad. Sin embargo, hay que estudiar más en profundidad estos para elegir aquel que se adapte mejor a nuestras necesidades, por lo que los motivos que han hecho que eligiera Unreal Engine 4 (en adelante, UE4) han sido:

- Buena documentación: en la página oficial[7] podemos encontrar documentación acerca de todas las funcionalidades del motor gráfico. Cualquier herramienta está explicada de tal forma que no encontremos dificultades al toparnos por primera vez con el motor.
- Comunidad de usuarios: UE4 se caracteriza por tener un foro oficial con una actividad diaria muy importante. Es muy probable que el problema que tengas esté solucionado ya por algún usuario, y si no es así puedes escribirlo para que la comunidad te ayude de forma rápida y eficiente.
- UE4 da todas las herramientas que se pueden necesitar para el desarrollo de un videojuego. Los *Blueprints* nos permiten implementar cualquier idea de forma más rápida que mediante el uso de C++, y el soporte por parte de la comunidad y la propia empresa les convierte en una herramienta muy importante.
- La plataforma es totalmente gratuita y podemos obtener resultados con un nivel muy alto.
- La tienda[8] de *Epic Games* ofrece un sinfín de material gratuito a disposición de cualquier desarrollador.
- YouTube[9] se ha convertido en una gran biblioteca de contenido, facilitando el aprendizaje de nuevas ideas y conceptos. No solo la comunidad ofrece estos video-tutoriales, también la tienda de *Epic Games* tiene contenido audiovisual.

UE4 permite crear un videojuego completo sin necesidad de programar código, y teóricamente aumenta la velocidad y eficiencia del desarrollo. Por estas razones todo el desarrollo se va a basar en el motor gráfico UE4 y sus Blueprints.

3.4.1. Descripción

UE4 es una suite que permite desarrollar animaciones y videojuegos mediante el uso de herramientas de *scripting* visual (conocidas como *Blueprints*) o programación con C++. Está catalogado como *real engine* debido a su capacidad de desarrollar escenas con gran similitud respecto al mundo real.

3.4.2. *Blueprints*

Los *Blueprints* son un sistema de *scripting* visual basado en el concepto de usar tarjetas o nodos para crear funcionalidades dentro del editor de UE4. Este lenguaje establece uniones de tipo objeto a objeto con elementos pertenecientes al motor gráfico, pudiendo tener diferentes salidas y entradas. El recorrido de un *Blueprint* comienza en el nodo que está situado a la izquierda y continúa siguiendo los enlaces que lo recorren.

Cada nivel tiene un Blueprint asociado y podremos encontrar blueprints dentro del general de cada nivel o por separado en forma de archivos. Aquellas funcionalidades que vayan a ser reutilizadas en otros niveles se guardaran fuera del general en forma de archivo externo, a diferencia de las funciones que sean especificas del nivel que en cuyo caso se guardan dentro de este.

Los *Blueprints* se usan para sustituir a la programación con código.

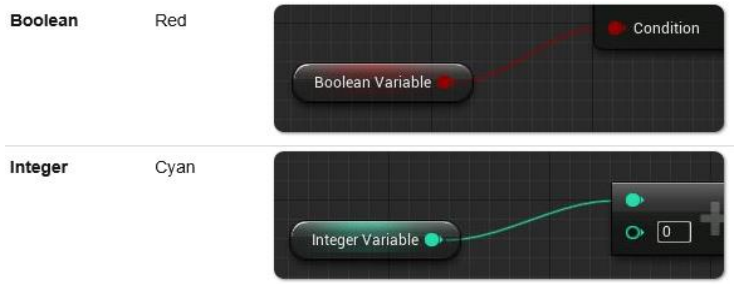


Ilustración 10: Ejemplo de variables

Se puede obtener más información acerca de los colores y variables en la documentación oficial[7]

3.4.2.1. Clases

A la hora de crear un nuevo *Blueprint* podemos elegir entre diferentes clases. Estas se podrían asemejar a una superclase de Java. Una vez creado el *Blueprint*, este contiene las funciones y variables básicas que tenga la clase escogida.

Existen muchos tipos de *Blueprint* con funcionalidades muy específicas que se pueden encontrar en la documentación de Unreal Engine (https://docs.unrealengine.com/latest/INT/Engine/*Blueprints*/UserGuide/Types/Class*Blueprint*/index.html).

Veremos ejemplos a continuación.

3.4.2.2. Variables

Los *Blueprints* también pueden usar variables como cualquier otro lenguaje y además métodos *Get* y *set*. Un ejemplo del uso de las variables:

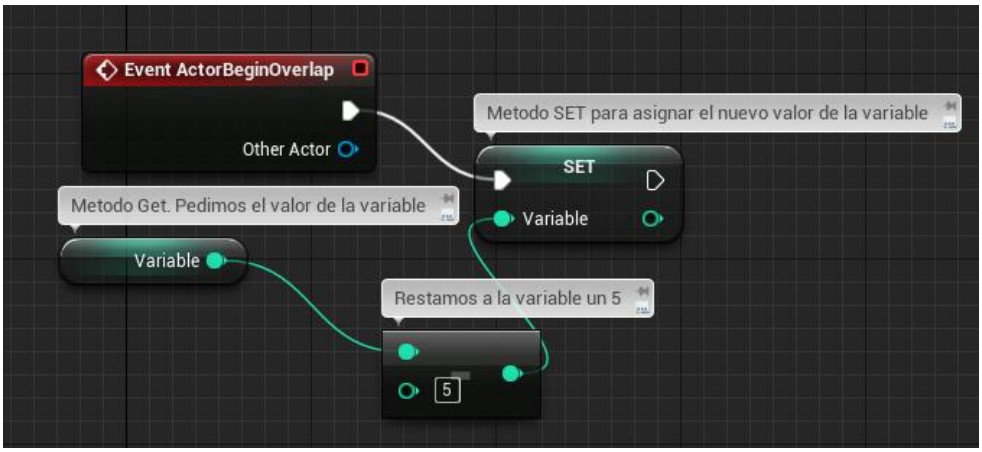


Ilustración 11: Variables

3.4.2.3. Funciones

Las funciones pueden recibir y devolver diferentes parámetros. Podrían entenderse como los métodos de Java. En el ejemplo tenemos una función llamada *Take Damage* que va a actualizar la vida del personaje.

Ilustración 12: Funciones

Esta función la llamamos desde otro punto del *Blueprint* como podemos ver en la imagen a continuación.

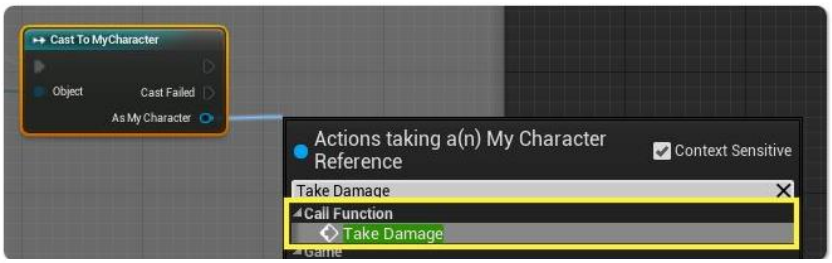


Ilustración 13: Funciones 2

3.4.2.4. Eventos

Los eventos se usan para responder a acciones tales como colisiones. No devuelven ningún valor y son similares a las funciones, con la diferencia de que una función siempre se va a ejecutar y un evento depende de una acción para activarse. Un ejemplo sería:

Cuando el usuario active el evento *activar altar* se imprimirá por pantalla un texto y se destruirá un portón.

Ilustración 14: Evento

3.4.2.5. Actor Component

La clase Actor Component se usa para reutilizar comportamientos, sonidos, movimientos o características muy específicas en diferentes actores.

Un ejemplo sería el de heredar un comportamiento de seguir al enemigo por parte de varios tipos diferentes de enemigos. Podemos hacer una clase Actor Component y asociarla a cada uno de los enemigos sin tener que hacer una para cada uno.

3.4.2.6. Widget

UE4 tiene una herramienta de diseño de interfaces de usuario llamada *Unreal Motion Graphics IU Designer* (UMG). Los widgets son el centro de esta herramienta y definen los diferentes elementos de una interfaz de usuario del videojuego, como botones, menús o pantallas de carga.

La forma de trabajar es muy simple y visual ya que permite añadir elementos a la interfaz arrastrándolos desde la paleta de elementos que vemos a la derecha.

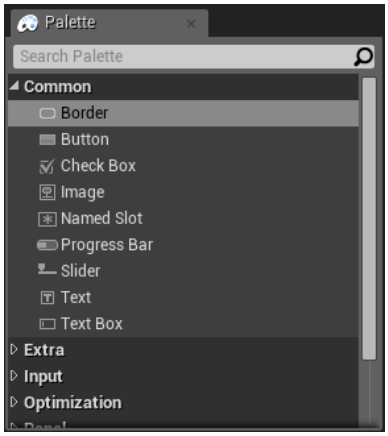


Ilustración 15: Widget

Una vez elegido el elemento que queremos colocar, tan solo hay que arrastrarlo a la pantalla de diseño.

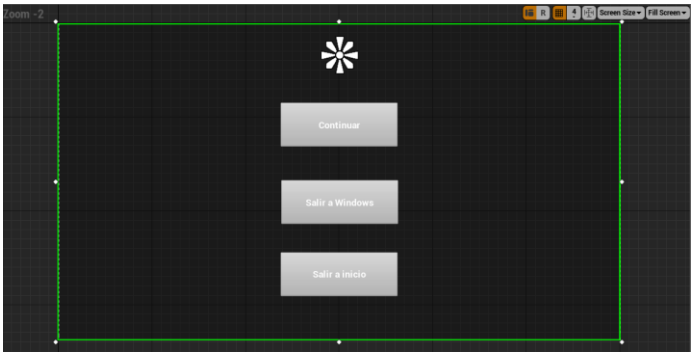


Ilustración 16: Widget 2

Una vez diseñado el aspecto visual se pueden añadir funcionalidades por medio de eventos específicos para cada botón.

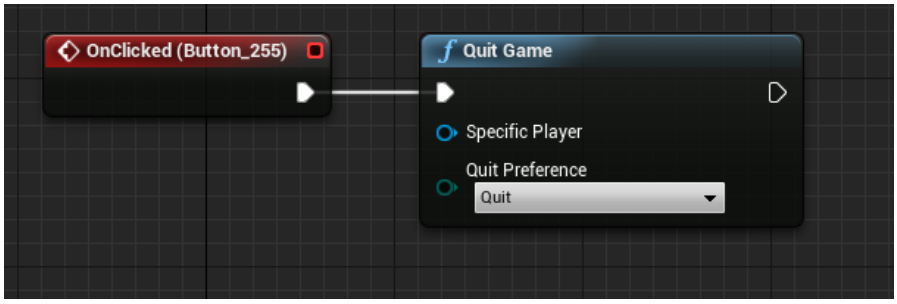


Ilustración 17: Widget 3

En la imagen de arriba podemos ver cómo hemos creado un evento para cuando se presione el botón que consistirá en salir del juego.

4. DOCUMENTO DE DISEÑO DEL VIDEOJUEGO (GDD)

El documento de diseño de videojuego (*Game Design Documento* o GDD) es la base para el desarrollo de un videojuego. Sintetiza la estructura general de lo que será el videojuego (concepto, historia, género…). Formará el esqueleto del proyecto en vistas a un desarrollo más fluido.

El desarrollo de un videojuego difiere en gran medida de la creación de aplicaciones o páginas web, por lo que se ha estandarizado el uso de este tipo de documentos. Juegos como GTA[10], Larry[11] o Rise of the dead[12] tienen sus documentos en público.

La idea de usar esta estructura viene dada de los diferentes TFG y artículos relacionados con el diseño de videojuegos que se pueden encontrar en la bibliografía.

4.1. EL JUEGO EN TÉRMINOS GENERALES

4.1.1. Concepto del juego

El personaje principal se despierta en un planeta desconocido. Lo único que recuerda es que estaba en un viaje interplanetario en dirección al exoplaneta Shepard-4, en el que se sospechaba que podía haber vida, tras haber recibido una radiobaliza de origen extraterrestre. Cerca encuentra su nave destruida: parece que el impacto ha destrozado el sistema de propulsión y ha comenzado un fuego que amenaza con hacer explotar los depósitos de combustible.

La única alternativa para el jugador es internarse por el desierto rojo que le rodea y buscar cobijo durante la noche, ya que no conoce los peligros que puede encontrar en este planeta. Caminando sin rumbo se encuentra con un edificio misterioso que no puede abrir. Debe reunir 4 fragmentos de una llave para poder entrar, y solo así conseguirá averiguar qué ha ocurrido con la antigua civilización que habitaba el planeta y con sus compañeros de viaje.

4.1.2. Género

El género del juego es aventura 3D[13] en primera persona, caracterizado por la necesidad de investigar, solucionar problemas, interactuar con otros personajes y seguir una línea histórica. Este tipo de género también tiene subgéneros muy parecidos a los de la literatura o el cine, estando este proyecto dentro de la rama de la ciencia ficción y misterio.

4.1.3. Clasificación de contenido

El sistema PEGI[14] europeo clasifica los videojuegos según su contenido y por categorías de edad mínima. Este proyecto tiene los siguientes descriptores de contenido:

Logo	Nombre del descriptor	Explicación
	Violencia	Puede contener escenas de personas que sufran lesiones o que mueran, a menudo mediante el uso de armas. También pueden contener derramamiento o gotas de sangre.
	Miedo	Puede contener escenas que se consideran demasiado perturbadoras o aterradoras para los más jóvenes o los jugadores emocionalmente vulnerables.
	PEGI 12	En esta categoría pueden incluirse los videojuegos que muestren violencia de una naturaleza algo más gráfica hacia personajes de fantasía y/o violencia no gráfica hacia personajes de aspecto humano o hacia animales reconocibles, Así como los videojuegos que muestren desnudos de naturaleza algo más gráfica. El lenguaje soez debe ser suave y no debe contener palabrotas sexuales.

Por lo tanto, este videojuego no es recomendado para menores de 12 años.

4.1.4. Flujo de juego

El flujo de juego[15] es la representación de la secuencia de pantallas que un jugador debe pasar para progresar en el juego en forma de diagrama de flujo. En él se incluyen también los menús de juego.

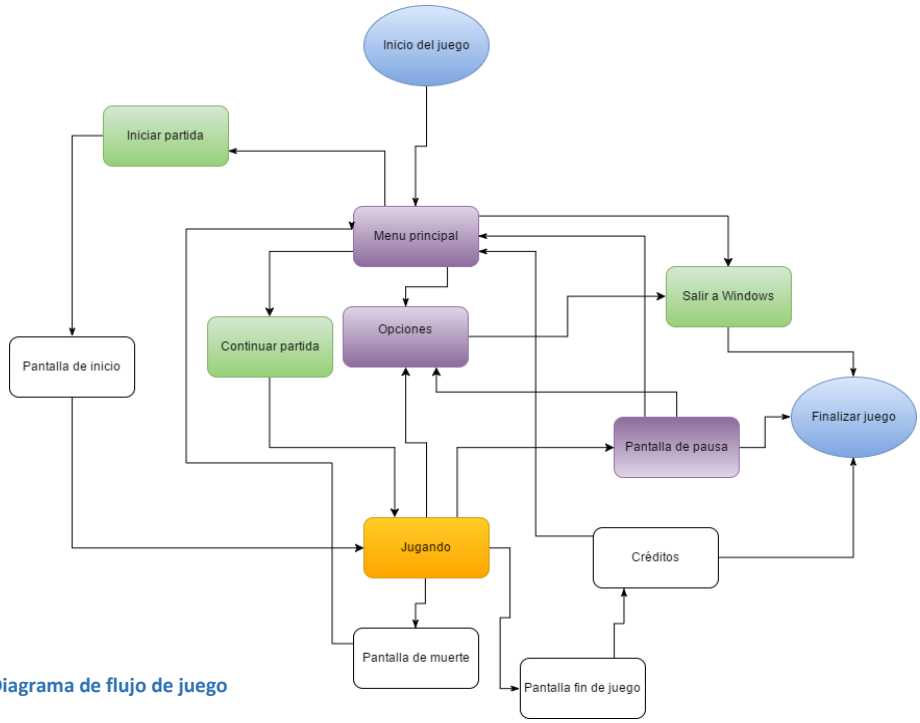


Ilustración 18: Diagrama de flujo de juego

4.1.5. Apariencia del juego

El juego intenta introducir al jugador dentro de un mundo de ciencia ficción de temática espacial con la libertad de moverse a lo largo de un mapa. Los elementos de ciencia ficción están influenciados por el videojuego *LifeLess Planet*[16], donde el jugador se encuentra en un mundo del que no conoce nada y sin saber lo que le deparará el transcurso de su aventura.

Por otro lado, el audio y la banda sonora del juego tratará de crear una atmósfera de misterio e incomodidad que supondría estar solo en un mundo desconocido.

4.2. HISTORIA

Nos situamos 200 años en el futuro. Los viajes espaciales se han convertido en algo normal y la humanidad está preparada para enviar una misión al exoplaneta Shepard-4.

Las mediciones indican que Shepard-4 puede tener vida inteligente, por lo que se convertiría en el primer contacto con vida extraterrestre.

En mitad del viaje la nave tiene un problema en los motores y la explosión hace que la curvatura espacio-temporal por la que viajaba se curve aún más, por lo que al llegar al nuevo planeta no han pasado 3 años como estaba previsto… sino casi 60 años.

Tras un aterrizaje forzoso, el jugador despierta en un planeta totalmente desconocido y ha perdido toda la memoria. Tan solo recuerda que pertenecía a una misión internacional en busca de vida extraterrestre y el nombre que viene bordado en su traje: Sujeto 1.

En este mundo desconocido tendrá que encontrar cobijo y una forma de comunicarse con la Tierra o con la supuesta vida inteligente de Shepard-4.

4.3. MUNDO SHEPARD-4

4.3.1. Resumen

El mundo donde transcurre la historia se dividirá en cuatro zonas. En las tres primeras toda la exploración será en primera persona y en la cuarta será mediante el uso de un vehículo.

4.3.2. Mapa

- Llegada al planeta: esta es la zona de inicio del juego. Se trata de un paisaje desértico envuelto en una esfera que simula el espacio exterior.

- El viejo edificio: es la única construcción que el jugador puede ver desde donde está. Dentro no habrá más luz que la de pequeñas lámparas y la linterna del personaje. Hay enemigos que perseguirán al jugador.

- La zona antigua: parte subterránea del viejo edificio. En ella se encuentran diferentes pruebas propuestas por un personaje no jugador (PNJ en adelante) llamado Bob el superviviente.

- La zona de conducción: de vuelta en el exterior del edificio, el jugador tendrá que sortear peligros y buscar miembros de la tripulación a mandos de un vehículo, con la finalidad de llegar al final del mapa con el mayor número posible de supervivientes.

4.3.3. Final del juego

Tras finalizar todas las pruebas se mostrarán los créditos finales. Una vez visualizados se volverá al menú inicial.

4.4. PERSONAJES

4.4.1. Personaje principal

- **Personaje a pie**: el jugador es el único superviviente conocido de la misión internacional. Tendría que haber llegado a Shepard-4 hace casi 54 años por lo que en la tierra se dio por muerta a toda su tripulación. Es el personaje que realiza todas las acciones del videojuego. Posee vida y energía para poder correr además de ser capaz de usar los power-ups que encuentre a lo largo del mapa

- **Vehículo**: tras encontrar un viejo asentamiento de la civilización originaria del planeta, el jugador logra encontrar un transporte que promete llevarle a una máquina extraterrestre capaz de llevarle de vuelta a la Tierra junto con el resto de miembros de la tripulación vivos.

4.4.2. Resto de tripulación

- **Tripulación** viva: el jugador descubre en la última parte del juego que parte de su tripulación sigue viva y tendrá que recoger a los supervivientes.

- **Tripulación** muerta: no todo el mundo corrió la misma suerte. Aquellos que murieron en este planeta mutaron en una máquina que intenta recuperar su cuerpo. Una vez comienzan a andar pierden la poca energía que les queda a los 8 segundos.

4.4.3. Bob el superviviente

Bob es el único superviviente conocido de la civilización que habitaba el planeta. Nos contará qué pasó durante todos los años de viaje y el porqué de la desolación del planeta.

4.5. MECÁNICAS DE JUEGO

4.5.1. Objetivos del juego

El juego contará con diferentes retos o misiones en los distintos niveles. Para poder pasar de nivel hay que superar todas las misiones.

La lista de misiones es:

1. Llegada al planeta (primer nivel): tras los créditos iniciales hay un pequeño tutorial para aprender los controles básicos. Después el jugador debe buscar los fragmentos de una llave que le permitirá abrir una puerta que le lleva al siguiente nivel.

2. El viejo edificio (segundo nivel): tras lograr entrar a través de la puerta se escucha una voz pidiendo auxilio. En este nivel hay que encontrar unos explosivos para derribar una pared, lo

que permite continuar a lo largo del mapa. Dentro de este edificio se encuentran algunos miembros de la tripulación del personaje principal, sin vida y transformados en enemigos que lo persiguen.

3. La zona antigua (tercer nivel): la voz de auxilio se escucha cada vez más cerca. Se descubre que su origen es un antiguo habitante del planeta llamado Bob. Este pide al jugador que le ayude con una serie de retos de habilidad. Una vez superados, Bob hablará de un vehículo y de un antiguo teletransporte que podría llevar al personaje principal de nuevo a la tierra.

4. La zona de conducción (cuarto nivel): una vez el jugador suba al vehículo, tendrá que recorrer una gran zona sorteando peligros y buscando supervivientes. Dependiendo de cuántos rescate el juego tendrá un final u otro.

4.5.2. Enemigos y vida

El jugador encontrará diferentes tipos de enemigos que podrán matarlo de un golpe o restarle pequeños porcentajes de vida. También habrá pequeños botiquines que le permitirán recuperar parte de la vida perdida.

4.5.3. Acciones del personaje

Modo primera persona: Las acciones que podrá realizar en las fases de primera persona serán:

- Moverse:
    - Hacia delante.
    - Hacia atrás.
    - Hacia la izquierda.
    - Hacia la derecha.
- Girarse:
    - Girarse 360º en el eje Y.
    - Girarse 90º en el eje X.
- Saltar.
- Correr.
- Ir lento.
- Interactuar con el medio:
    - Hablar con personajes.
    - Interactuar con elementos del escenario.

Modo vehículo:

- Moverse:
    - Hacia delante.
    - Hacia detrás.
    - Hacia la izquierda.
    - Hacia la derecha.
- Acercar o alejar la cámara.

4.5.4. Controles de juego

Es posible jugar usando raton y teclado. Eventualmente se puede usar un mando de Xbox gracias a que este es capaz adaptarse a los controles del ordenador.

Ilustración 19: Controles de juego

1. Movimiento del jugador:

   - Teclas W/↑ movimiento hacia delante.
   - Teclas A/← movimiento hacia la izquierda.
   - Teclas S/↓ movimiento hacia atrás.
   - Teclas D/→ movimiento hacia la derecha.

2. Acciones del jugador: por ejemplo, activar los altares o comenzar la comunicación con Bob.

3. Activar y desactivar la linterna.

4. Caminar despacio (mantener pulsado).

5. Correr (mantener pulsado).

6. Turbo del vehículo (mantener pulsado).

4.5.5. Mecánicas e interacciones

- **Fragmentos de llave** (primer nivel): se necesitan 4 fragmentos de llave para abrir la puerta de entrada que lleva del primer al segundo nivel.

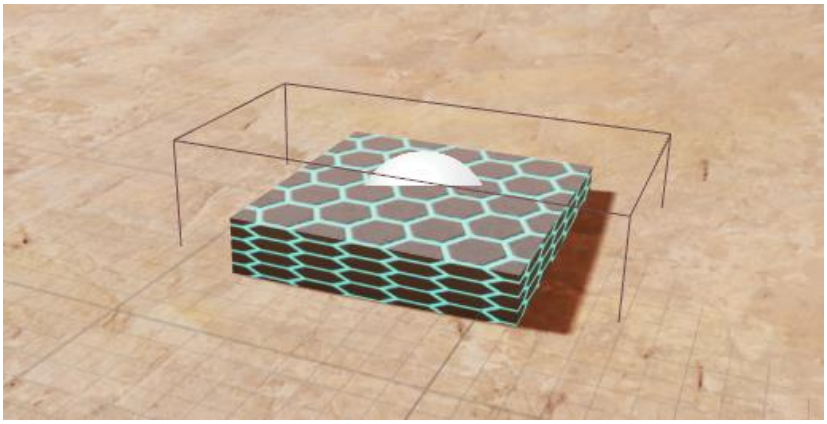


Ilustración 20: Llave

- **Portón** (segundo nivel): si el jugador se acerca a este portón se mostrará un mensaje de ayuda.

Ilustración 21: Portón

- **Teletransportador**: hay varios teletransportadores a lo largo del mapa. Sirven para mover al personaje de un nivel a otro. Cada uno de ellos tiene un dispositivo de origen y otro de destino. Para activarlos el jugador debe entrar en contacto con la caja que marca el lugar en el que se encuentra el teletransportador.

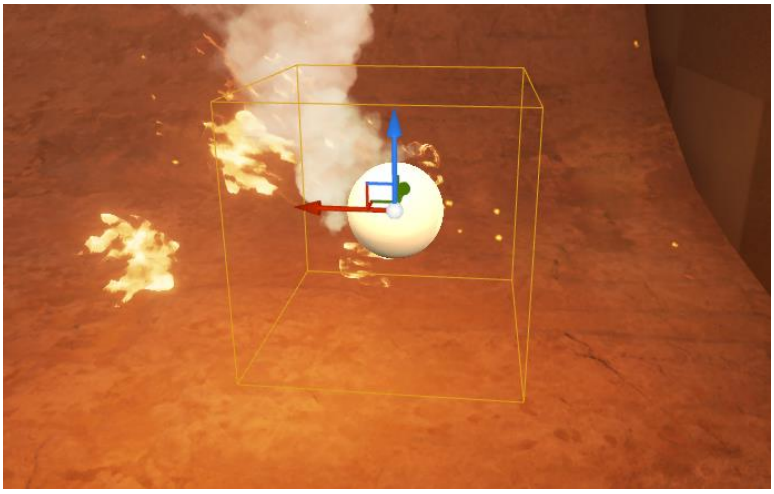


Ilustración 22: Teletransportador en el primer nivel

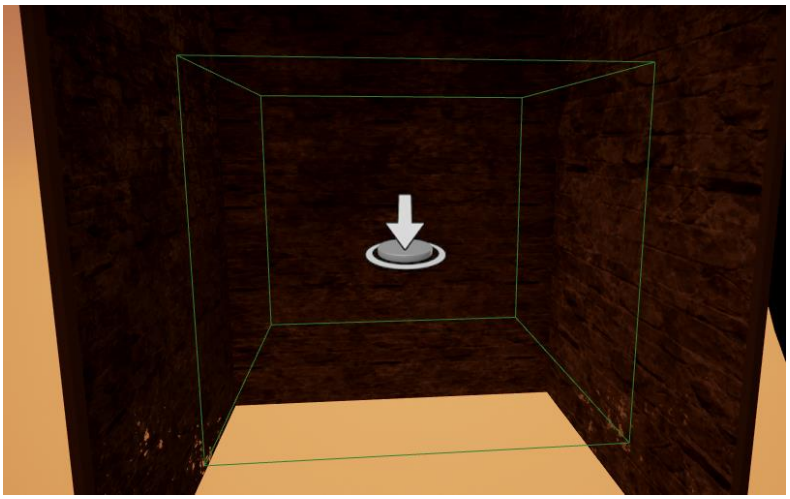


Ilustración 23: Teletransportador en el tercer nivel

- **Girador** (tercer nivel): el jugador podrá girar 90 grados pulsando la tecla E siempre que esté en contacto con el rectángulo amarillo.

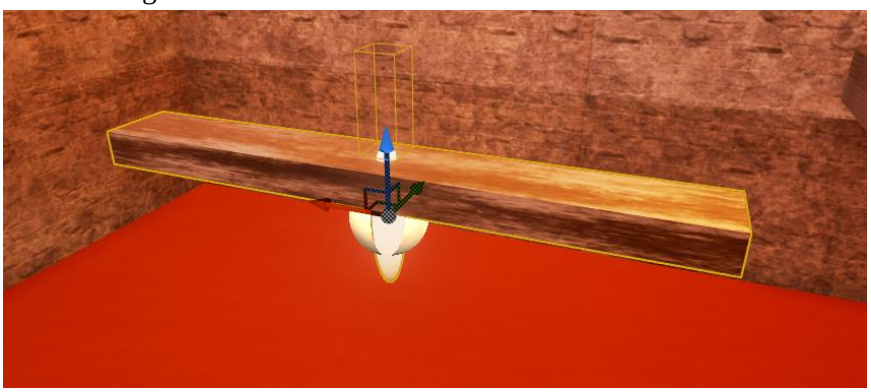


Ilustración 24: Girador

- **Altar mágico**: hay diferentes altares mágicos a lo largo de los niveles, su función es abrir nuevas zonas del mapa dentro del mismo nivel. Se activan pulsando la tecla E siempre que el jugador esté dentro del área gris. Algunos altares mágicos generan enemigos.

Ilustración 25: Altar mágico

- **Botiquín**: este elemento se encuentra a lo largo de todo el juego y añade un pequeño porcentaje de vida en caso de que el jugador haya sufrido daños.

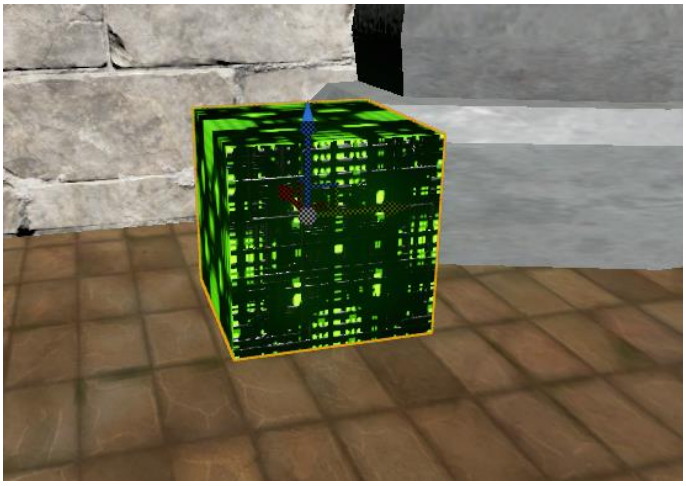


Ilustración 26: Botiquín

- **Bob el superviviente** (tercer nivel): este personaje no jugador (PNJ en adelante) nos ofrece un diálogo en el cual podemos descubrir más sobre la historia del juego. No se mueve y no interactúa con ningún otro elemento.

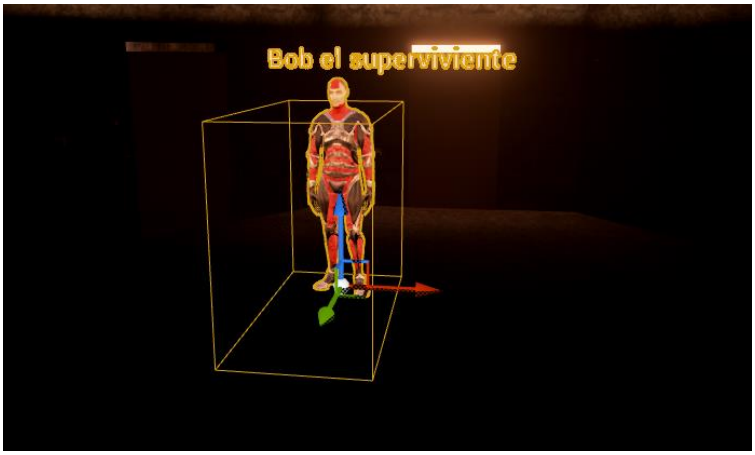


Ilustración 27: Bob el superviviente

- **Encendedor de antorcha** (tercer nivel): área que desactiva la linterna y enciende una luz tenue. Será la única luz disponible una vez sea activada.

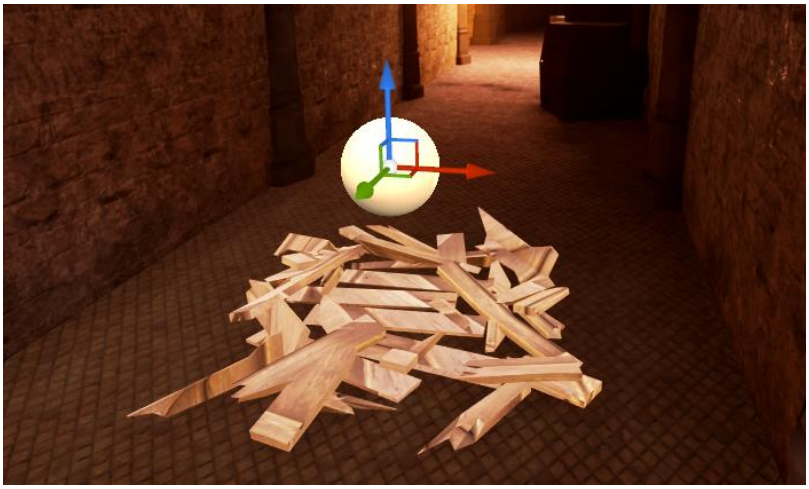


Ilustración 28: Encendedor de antorcha

- **Tripulación viva** (cuarto nivel): PNJs que deben ser rescatados en el nivel de vehículo. Modelo obtenido del pack libre de *Mixamo animation pack.*

Ilustración 29: Tripulación viva

- **Explosivos** (segundo nivel): objeto necesario para derribar una pared que permite continuar por el nivel.

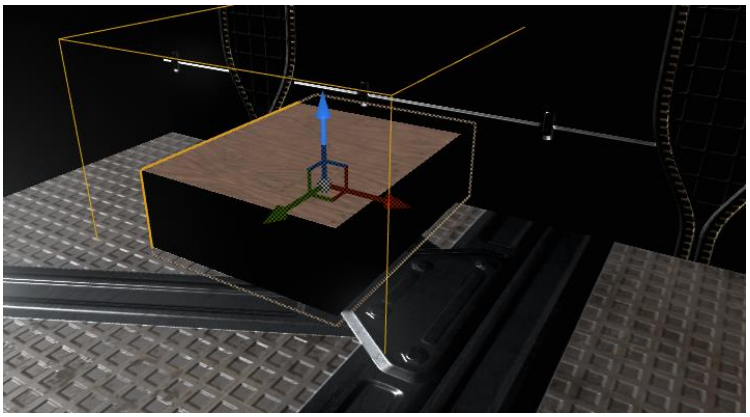


Ilustración 30: Explosivo

- **Pared explosiva** (segundo nivel): elemento que bloquea el acceso a la siguiente zona del nivel. Solo puede ser derribada mediante los explosivos.

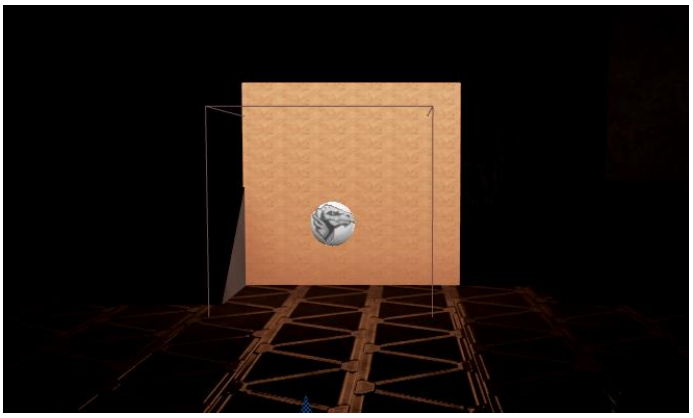


Ilustración 31: Pared explosiva

4.5.6. Elementos que quitan vida

- **Minas**: objetos que restan vida y desaparecen al entrar en contacto con ellos. Se encuentran en todos los niveles menos en el último.

Ilustración 32: Minas

- **Lava**: en caso de caer y entrar en contacto se teletransportará al jugador a las cercanías de la lava. Se encuentra en todos los niveles menos en el segundo.

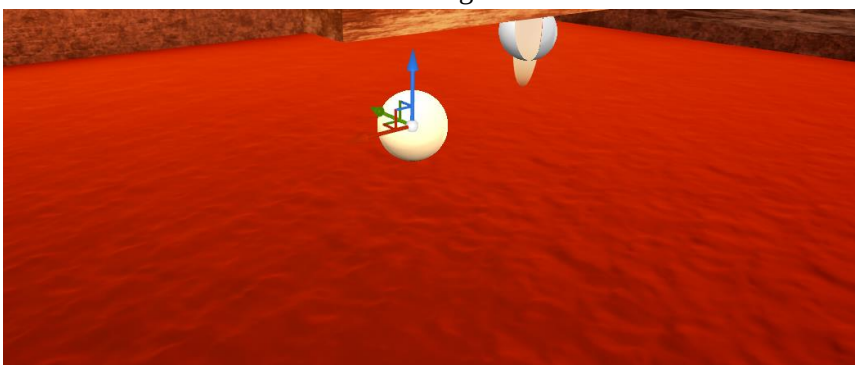


Ilustración 33: Lava

- **Antigua tripulación**: enemigo principal del juego. Perseguirá al jugador en cuanto éste entre en su campo visual, hasta quedarse sin energía. Si toca al jugador perderá toda la energía, caerá al suelo y quitará vida al jugador. Se encuentra en todos los niveles menos en el último.

Ilustración 34: Antigua tripulación

- **Picos** (tercer nivel): si el jugador entra en contacto con ellos perderá vida. Tienen una animación de subida y de bajada.

Ilustración 35: Picos

4.5.7. Power-up

- **Supervelocidad** (primer y segundo nivel)**:** si el jugador entra en contacto con este objeto obtendrá inmediatamente una mejora de velocidad durante 5 segundos. Se avisará al jugador de la obtención de esta mejora por pantalla y mediante un breve audio.

Ilustración 36: power-up velocidad

- **Supersalto (tercer nivel):** si el jugador entra en contacto con este objeto obtendrá inmediatamente una mejora de salto durante 5 segundos. Se avisará al jugador de la obtención de esta mejora por pantalla y mediante un breve audio.

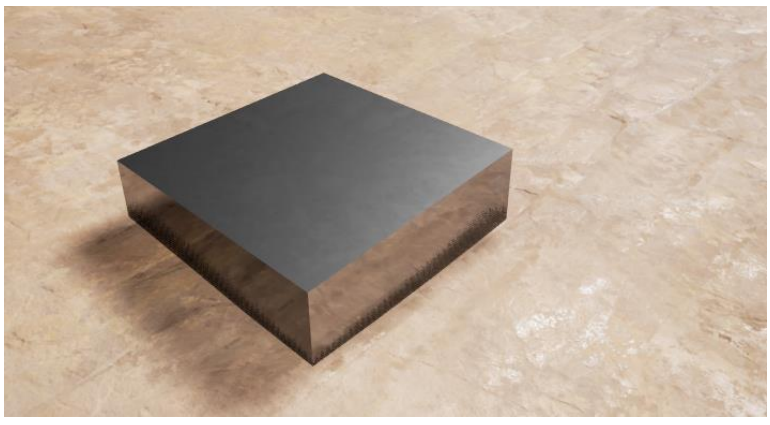


Ilustración 37: power-up salto

4.5.8. Otros elementos

- Luz parpadeante 1 y 2 (segundo nivel): estas lucen se encienden y se apagan de forma aleatoria simulando que están rotas o defectuosas. El jugador no puede interaccionar con ellas.

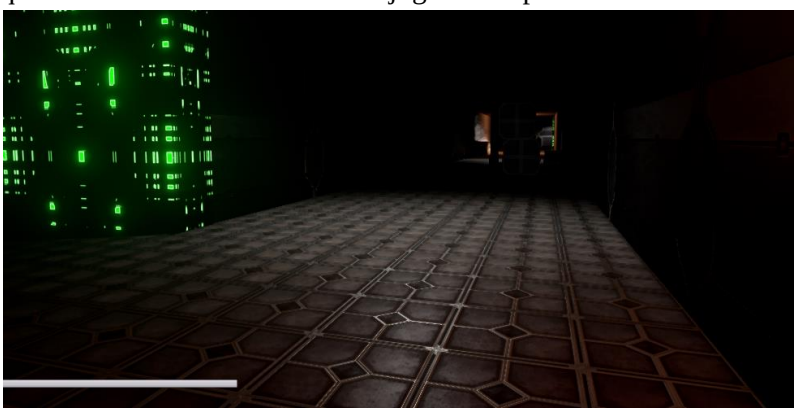


Ilustración 38: Luz parpadeante

- Fuego: elemento formado por particulas que no interacciona con el usuario y emite una tenue luz roja. Se usa tanto en zonas incendiadas como en las antorchas. Se encuentra en todos los niveles.

Ilustración 39: fuego

- Chispas eléctricas (segundo nivel): elemento formado por partículas que no interacciona con el usuario. A diferencia del fuego este no emite luz.

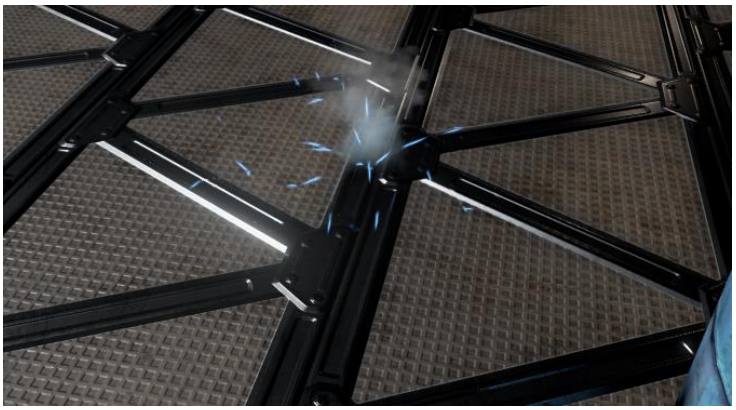


Ilustración 40: chispas eléctricas

4.6. MENÚS E INTERFACES

La interfaz es conocida como HUD17 (del inglés “Head-Up Display”) y muestra información por pantalla durante la partida. Puede mostrarse siempre (vida disponible) o en determinados eventos.

4.6.1. Interfaz durante la partida

El HUD durante la partida es muy sencillo y limpio. Solo se mostrará la vida del jugador y las alertas por eventos (recoger explosivo, recoger vida …).

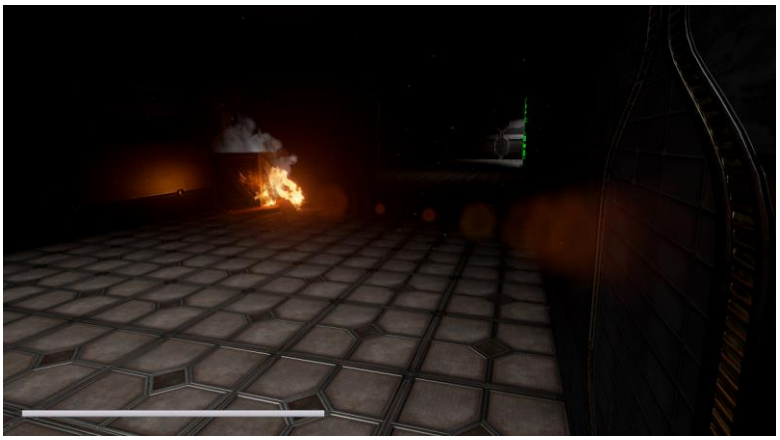


Ilustración 41:HUD durante la partida

4.6.2. Tutorial

Se activa en el inicio del juego y tiene por objetivo que el jugador aprenda los controles del videojuego. Mientras este menú este activo, el jugador no podrá moverse.

Ilustración 42: controles del juego

4.6.3. Interfaz fin de partida

Si el jugador tiene el contador de vida a cero, se mostrará el menú de fin de partida. En él se encontrarán las siguientes opciones.

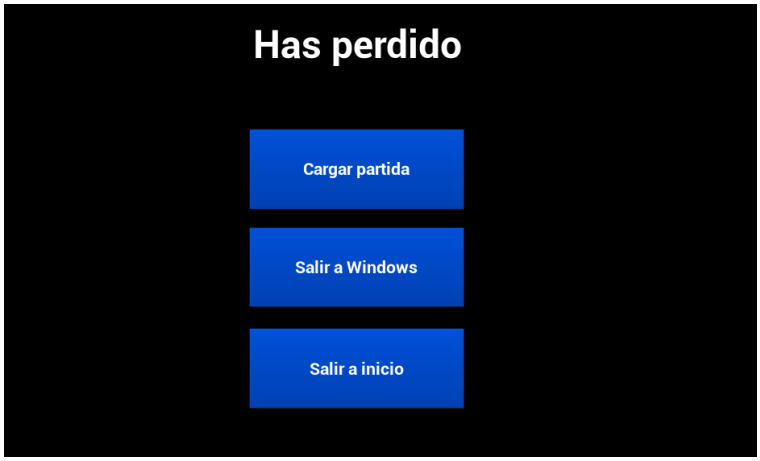


Ilustración 43: fin partida

4.6.4. Menú principal

Es el menú que se abre al comenzar el juego. En el vamos se encuentran las siguientes opciones:

Ilustración 44: Interfaz Menu principal

4.6.5. Menú de pausa

A este menú se accede por medio de la tecla “Escape” durante la partida. En él se encuentran las siguientes opciones:

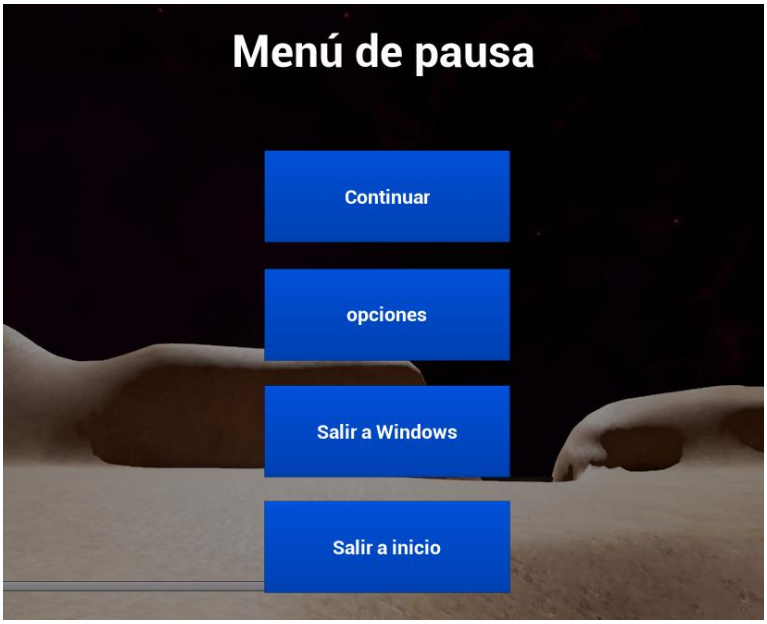


Ilustración 45: Interfaz Menú pausa

4.6.6. Menú de configuración

Dentro de las opciones podremos modificar los siguientes parámetros de video y audio:

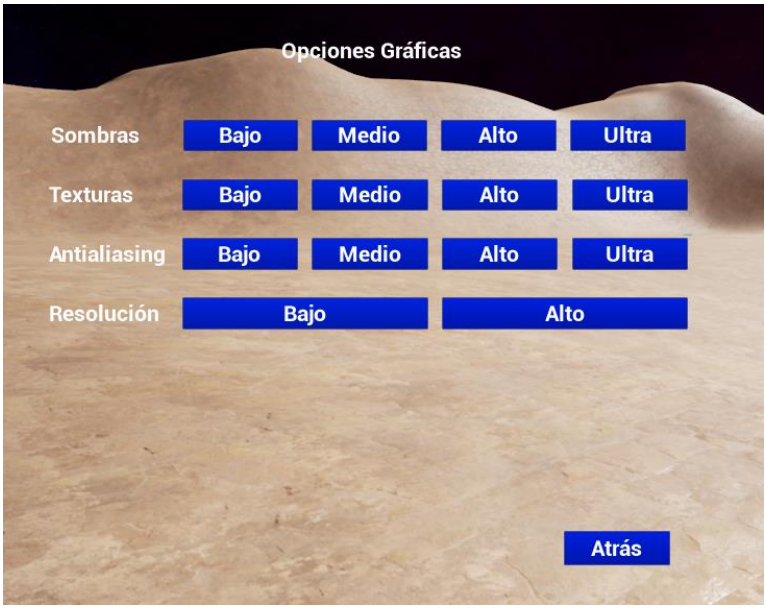


Ilustración 46: Interfaz configuración gráfica

4.6.7. Interfaz chat con Bob

Bob es un PNJ con el que podremos mantener un dialogo. Se permite responder a Bob con dos opciones diferentes.

Ilustración 47: interfaz Bob

4.6.8. Menú de cargar y guardado

A este menú se accede desde el menú de pausa y en él se encuentran las opciones de guardar partida y de continuar otra partida.

Ilustración 48: Interfaz cargar/guardar

4.7. PANTALLAS FINALES

Ambas pantallas se muestran al completar el juego. En caso de rescatar a todos los personajes se mostrará la ilustración 2, en el caso contrario se mostrará la ilustración 1

Ilustración 49: Interfaz final 1

Ilustración 50: Interfaz final 2

5. DESARROLLO E IMPLEMENTACIÓN

En esta sección se hablará del desarrollo e implantación de la sección 4 mediante el uso de la programación grafica de UE4.

5.1. ACCIONES DEL PERSONAJE

5.1.1. Primera persona

Dentro del Blueprint en el que configuramos las acciones del personaje, encontramos el componente llamado charactermovement. Este componente se encarga de configurar todas las propiedades de movimiento más usadas dentro de un videojuego y aunque nos ofrece muchas características solo se muestran las 3 más importantes.

- **General settings** (ver Ilustración 44): en esta parte encontramos las variables generales del jugador. Las variables más importantes son:
    - Gravity scale: valor personalizado de la gravedad. La gravedad que afecta al jugador es la del proyecto se multiplicada por este valor.
    - Mass: masa del jugador. Se usa para simular impactos y aceleraciones
    - Default Land Movement: movimiento estándar para cuando el jugador no se encuentra en una zona señalizada como agua. Es el valor predeterminado para el inicio del juego o cuando se modifica su posición al teletransportarlo a otro punto del mapa.

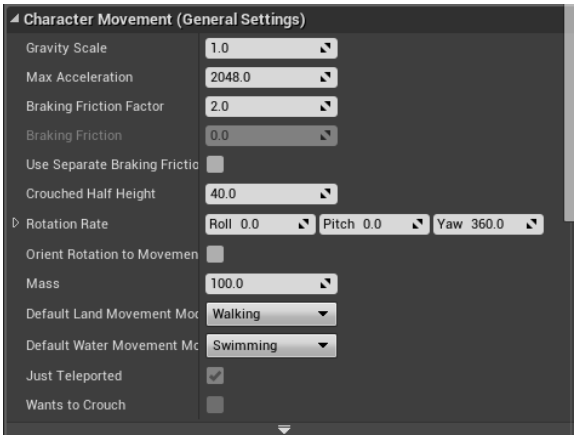


Ilustración 51: Character Movement

- **Walking** (ver Ilustración 45): en esta parte configuramos la forma de andar/correr del jugador. Las variables más importantes son:
    - Max Step Height: establece cual es la pendiente máxima por la que el jugador puede caminar.
    - Walkeable Flor Angle: establece cual es la pendiente máxima del suelo en la que el jugador puede andar.
    - Ground friction y Braking Deceleration Walking: establece la resistencia del jugador a modificar su dirección de movimiento. Si en ambas variables el valor es 0 no existirán fuerzas de rozamiento con el suelo y el movimiento se asemejaría a andar sobre hielo.

- Max Walk Speed: establece la velocidad máxima a la que se puede mover el jugador.

Ilustración 52: Character Movement Walking

- **Jumping / Falling** (ver Ilustración 46): en esta parte configuramos el salto y la forma que tiene el jugador de interactuar con el medio cuando está en el aire. Las variables más importantes son:

  - Jump Z Velocity: valor inicial al saltar. Si la variable fuese 0 no se separaría del suelo.

  - Air control: establece la capacidad que tenemos de modificar nuestra trayectoria en el aire.

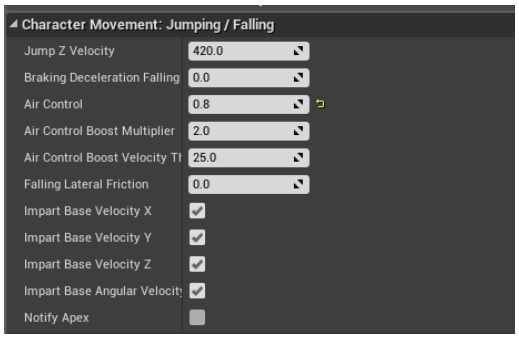


Ilustración 53: Character Movement Jumping

5.1.1.1. Eventos con pulsación de tecla

Para configurar el movimiento del personaje tenemos que realizar un *Mapping* de aquellas teclas que vayamos a usar para dar movimiento. Para ello tendremos que entrar en las opciones del proyecto y seleccionar la opción *Input*.

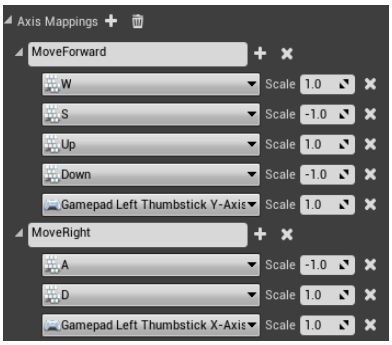


Ilustración 54: Axis Mappings

- **Moveforwad** (ver Ilustración 47): movimiento hacia delante y hacia atrás.

En la tecla W elegimos un valor positivo (movimiento hacia delante) y en la tecla S seleccionamos un valor negativo (movimiento haca detrás). Se da también la misma propiedad a las teclas flecha ya que es estándar poder mover el personaje con ambos conjuntos de teclas.

Seleccionamos también el *Stick* derecho de un mando. En esta ocasión solo tenemos que dar el valor positivo ya que el *Stick* asignará automáticamente un valor negativo si lo movemos hacia atrás.

- **MoveRight** (ver Ilustración 47): se aplica lo mismo que para el anterior movimiento, pero para los giros. En este caso el valor positivo del giro es cuando se hacia la derecha y el valor negativo es cuando se gira hacia la izquierda.
- **Jump**: Se ha asociado la barra espaciadora a la acción llamada *Jump*.
- **Sprint**: Se ha asociado la tecla *shift* izquierdo a la acción llamada sprint usada para correr.
- **Slow**: la tecla C a la acción llamada *slow* usada para ir más lento.
- **Flashlight**: Se ha asociado la tecla F a la acción llamada *flashlight* usada para encender y apagar la linterna.

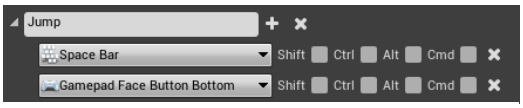


Ilustración 55: Jump

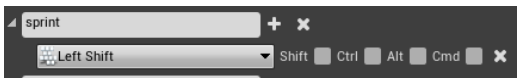


Ilustración 57: Sprint

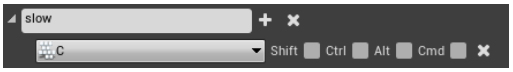


Ilustración 58: Slow

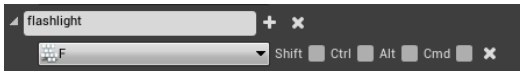
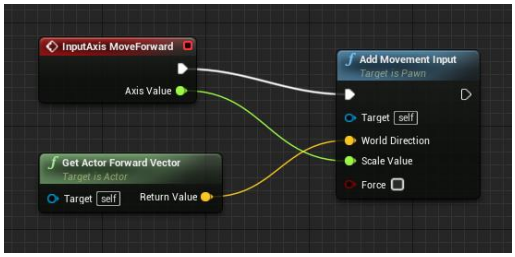


Ilustración 56: Flashlight

- **Movimiento hacia delante y hacia atrás** (ver Ilustración 52): cuando hay un evento *MoveForward* (pulsar W, S, flechas o *Stick*) se generará un movimiento con la dirección del vector de movimiento delantero del personaje.

En caso de que se pulse W, flecha delantera o *Stick* inclinado hacia delante, el movimiento será positivo y el jugador se moverá hacia delante. Si se pulsa S, flecha hacia abajo o el *Stick* inclinado hacia abajo, el movimiento será negativo y hacia atrás.

Ilustración 59: Movimiento delante/atrás



- **Girar** (ver Ilustración 53): cuando hay un evento *MoveRight* (pulsar A, D, flechas o *Stick*) se generará un movimiento con la dirección del vector de movimiento lateral del personaje.

  En caso de que se pulse D, flecha derecha o *Stick* inclinado hacia la derecha, el movimiento será positivo y el jugador girara hacia la derecha. Si se pulsa A, flecha hacia la izquierda o el *Stick* inclinado hacia la izquierda, el movimiento será negativo y girará hacia la izquierda.

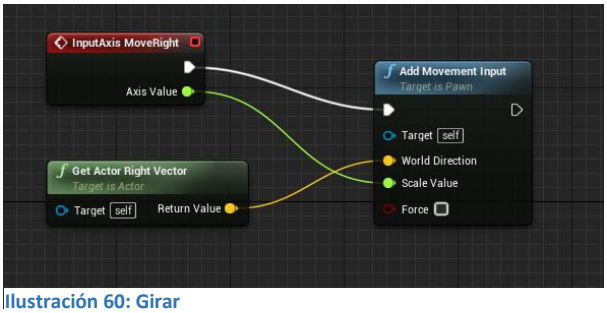


Ilustración 60: Girar

- **Saltar** (ver Ilustración 54): el evento *InputAction Jump* se activa cuando se pulsa la barra espaciadora (se activa una función predeterminada por UE4 que hace que el jugador se eleve rápidamente durante unos pocos segundos).

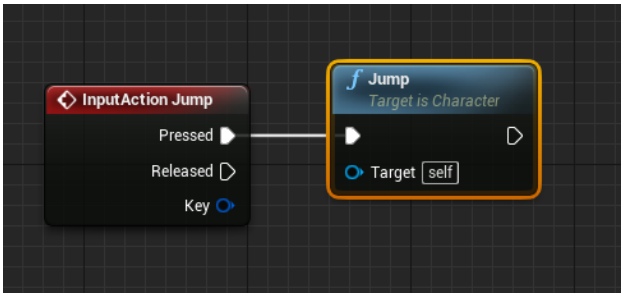


Ilustración 61: Saltar

- **Correr** (ver Ilustración 55): El evento *InputAction* Sprint se activa al pulsar la tecla *Shift* izquierda y mientras este activada la variable *Max Walk Speed* aumenta a 750. Cuando se deja de pulsar la variable vuelve a su valor predeterminado de 500.

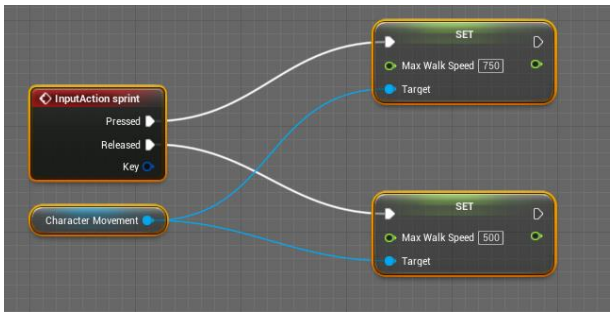


Ilustración 62: Correr

- **Caminar lento** (ver Ilustración 56): El evento *inputAction slow* se activa al pulsar la tecla "c" y mientras este activada la variable *Max Walk Speed* disminuye a 250. Cuando se deja de pulsar la variable vuelve a su valor predeterminado de 500

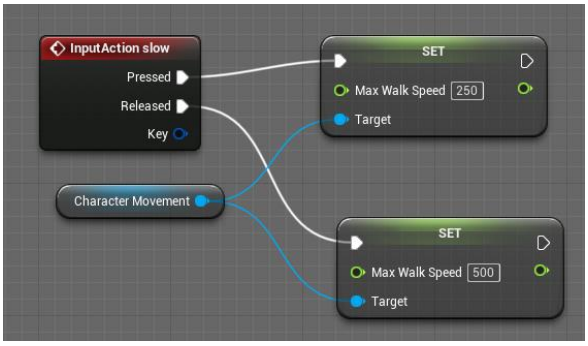


Ilustración 63: Caminar lento

- **Linterna** (ver Ilustración 57): El evento i*nputAction flashlight* se activa al pulsar la tecla "f". El nodo *Flipflop* nos va a permitir alternar entre las tarjetas *Toggle Visibility* (que van a actuar sobre un punto de luz que se sitúa delante del jugador) ejecutando primero el camino "A" y luego el "B".

  La primera vez que pulsemos la tecla "f" se ejecutara el camino "A", por lo que cambiar el estado del punto de luz de visible a no visible y mostrara un mensaje por pantalla indicando que la linterna está apagada.

  La siguiente vez que pulsemos se ejecutara el camino "B" por lo que volverá a cambiar el estado del punto de luz de no visible a visible y mostrara un mensaje por pantalla indicando que la linterna está encendida.

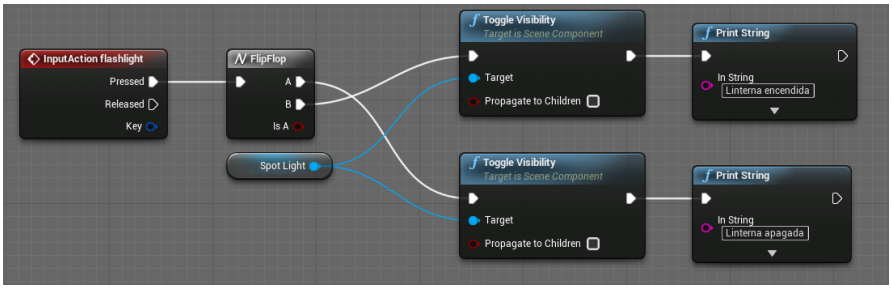


Ilustración 64: Linterna

5.1.1.2. Eventos sin pulsación de letra

- **Encender antorcha** (ver Ilustración 58, 59): este evento enciende una antorcha que ilumina el área cercana.

  Cuando se activa el evento *OnComponentBeginOverlap* (sobre el disparador de encender la antorcha) usamos un Casting sobre el *Blueprint* del jugador (nodo *Cast To FirstPersonCharacter*) para poder hacer la llamada al evento Encendido (nodo encendido *Event*) que veremos a continuación.

  Una vez llamamos a este evento se imprime por pantalla un mensaje (antorcha encendida) y se hace visible un punto emisor de luz con partículas de fuego. De este modo aparece a la derecha del jugador la luz de una antorcha.

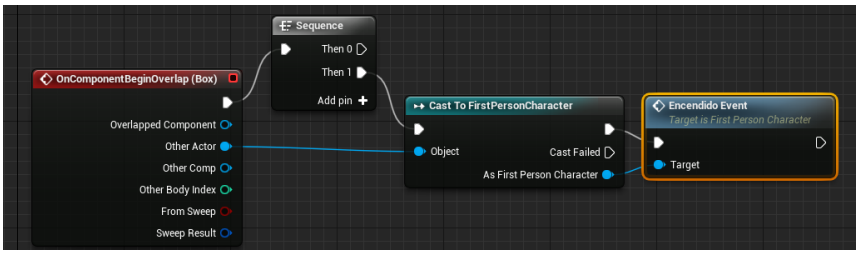


Tabla 15: antorcha

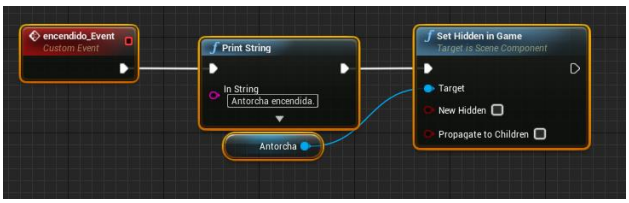


Ilustración 65: antorcha 2

5.1.1.3. Eventos con pulsación de la letra E

- **Altar mágico** (ver Ilustración 60, 61): este *Blueprint* tiene la función de eliminar el muro del pasillo para que el jugador pueda continuar con el juego y se puede activar tras pulsar la letra E siempre y cuando estemos cerca del altar (Variable Encima Altar sea *true*).

  El nodo "Encima altar" es una variable de tipo booleana que va a identificar si el usuario está dentro del área (nodo *onComponenteBegin Overlap* asigna true con el nodo *SET* y el nodo *EndOverlap* lo cambia a false) como podemos ver en la figura inferior.

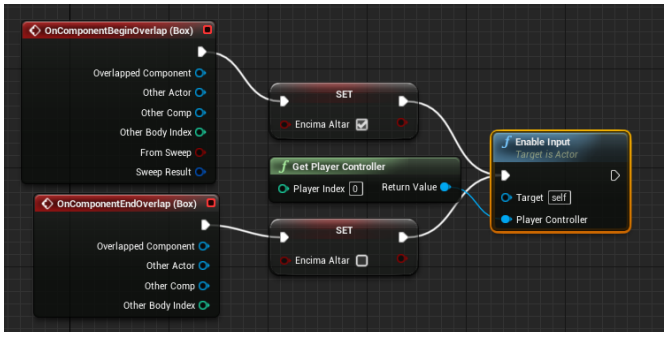
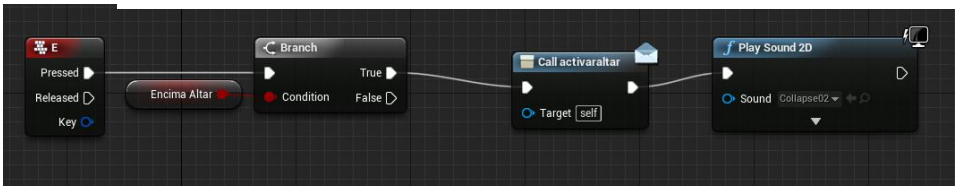


Ilustración 67: altar mágico



- **Explosión de pared** (ver Ilustración 62): este *Blueprint* se encarga de crear la explosión que destruye la pared. Una vez el jugador se acerca a la pared, se mostrará un aviso por pantalla y se producirá la explosión.

  El nodo secuencia divide la ejecución en dos partes:

  1- En la primera se crea una fuerza de tipo radial (nodo Fire Impulse) que destruye la pared.

  2- En la segunda se añaden partículas de fuego (nodo Activate Particle System) y se elimina el activador de la explosión para evitar que la acción vuelva a ocurrir.

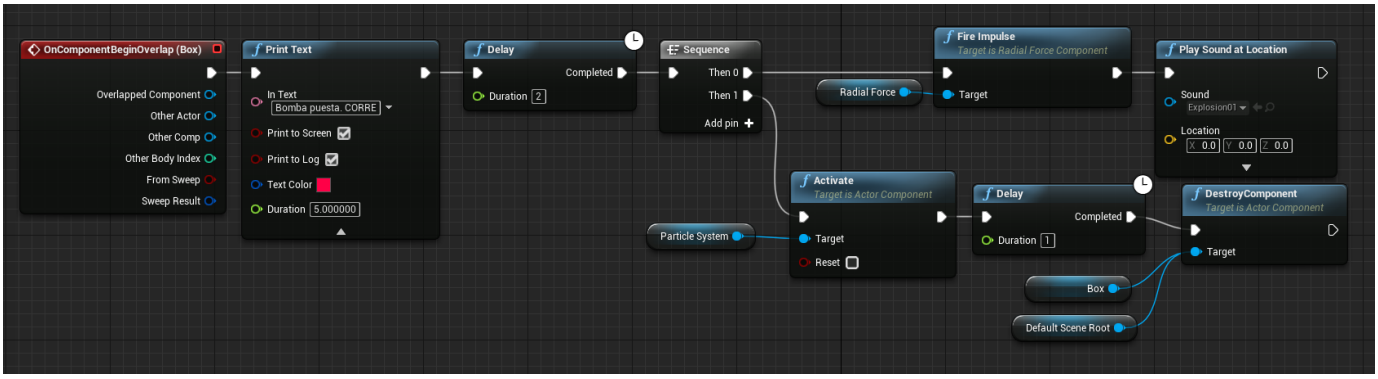


Ilustración 68: Explosión pared

- **Girador** (ver Ilustración 63, 64, 65): el girador está formado por 2 elementos, una caja para detectar cuando estamos encima de él y el girador como elemento físico dentro del juego.

  Este *Blueprint* lo dividimos en 3 partes:

  1- Detección de jugador: cuando el jugador esta encima del girado se activa la posibilidad de interaccionar con el elemento "girador", en caso de que salga del área de activación quitamos la capacidad de interaccionar con el "girador"

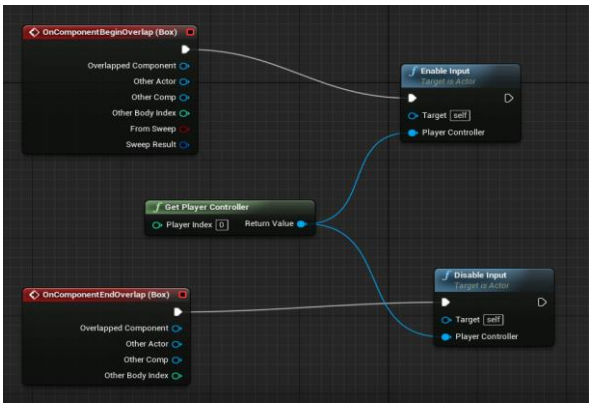


Ilustración 69: Girador

  2- Interacción con el girador con el uso de la tecla E:

  - Girostatus 2 es una variable booleana que nos va a indicar el estado del girador, siendo true el estado inicial y false el estado girado.

- Dentro del nodo *Branch* diferenciamos el valor de esta variable de cara a cambiarla tras presionar E.

- *Timeline_0* se encarga junto al nodo *make Rotator* de realizar la rotación del elemento elegido de aplicar la rotación sobre el objeto (el nodo *SetActorRotation* tiene como objetivo el propio girador)

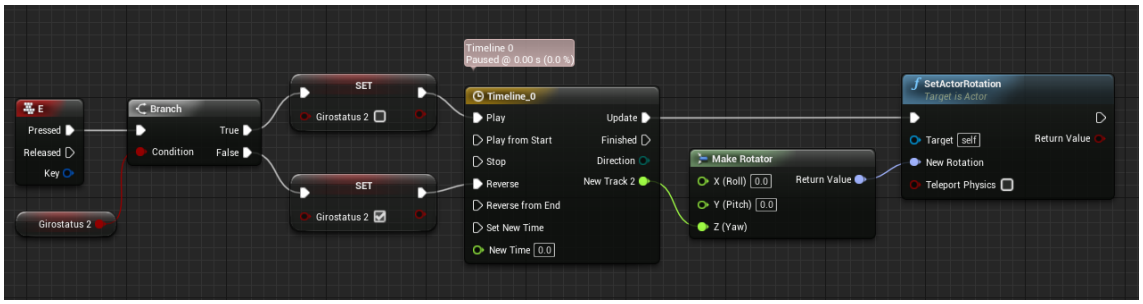


Ilustración 70: girador 2

3- Timeline_0: en este nodo está programada la velocidad y la forma que tiene de realizar la animación de giro. En el eje X se sitúa la duración (1 segundo). En el eje Y se sitúa el porcentaje de giro (cuanto más pendiente tenga más rápido se efectúa el giro).

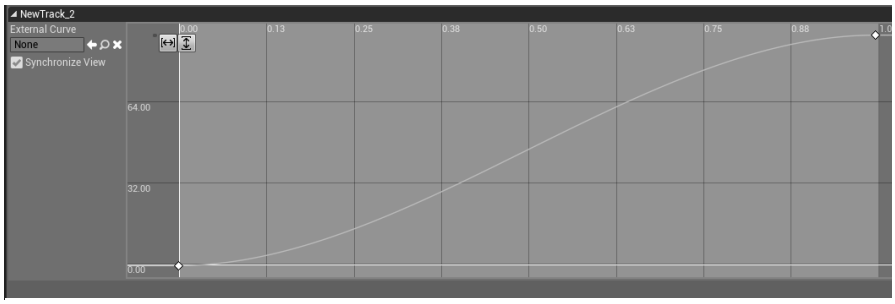


Ilustración 71: Timeline Girador

5.1.2. Vehículo

Aunque UE4 ofrece un vehículo ya programado, para este proyecto se ha creado uno desde cero.

5.1.2.1. Acciones del vehículo

El vehículo está formado por el *Blueprint* Vehiculocomponente y Vehiculo.

- **Vehiculocomponente** (ver Ilustración 66, 67, 68): este *Blueprint* se va a encargar de calcular la fuerza de cada componente que hará que el vehículo levite y se configurará la resistencia a ir hacia delante y a girar. Podría entenderse como las ruedas del vehículo (solo que en este caso serían invisibles).

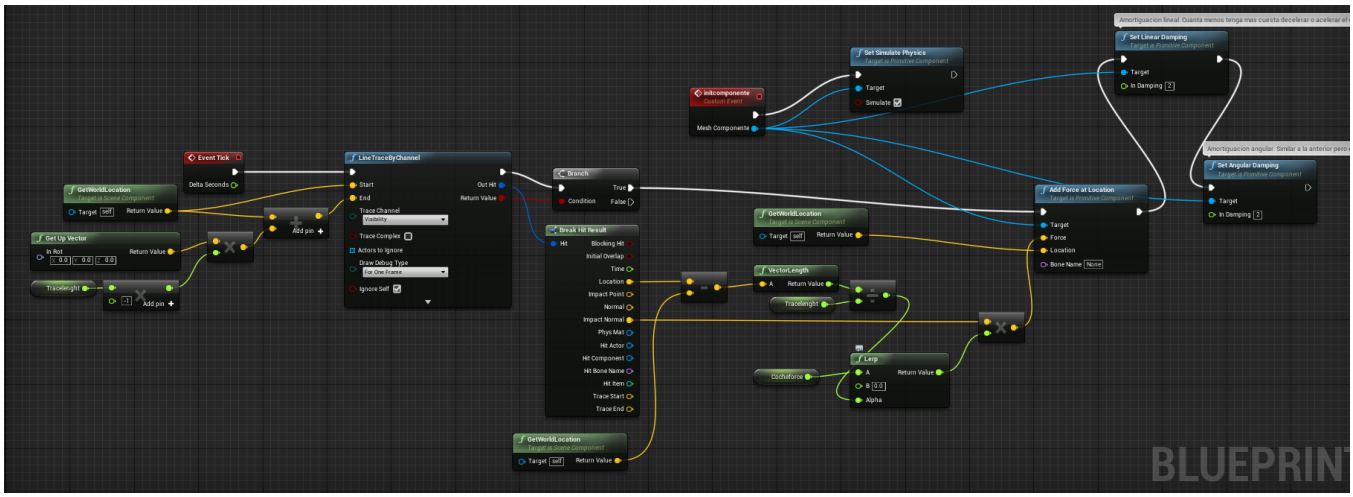


Ilustración 72: Vehículo componente

Se dividirá el diagrama en dos partes para explicarlo mejor.

1- Primera parte:

Esta parte se encarga de calcular la fuerza que tiene opuesta a la gravedad del vehículo y comienza con el nodo *Event Tick* por lo que se activará varias veces por segundo (los *ticks* son las ejecuciones del videojuego por segundo). Se calcula varias veces por segundo la posición del vehículo respecto del suelo de cara a calcular si el vehículo tiene que bajar (en ese caso estaría demasiado despegado del suelo) o subir (en ese caso estaría demasiado pegado al suelo).

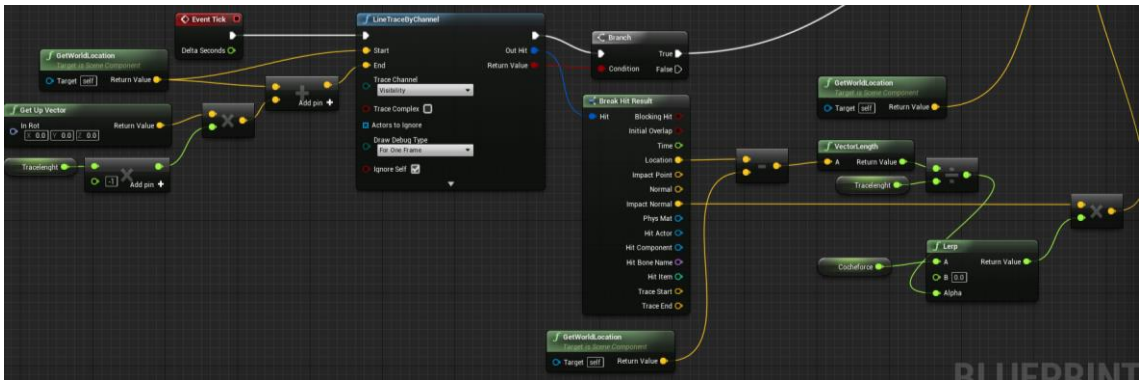


Ilustración 73: vehículo componente 2

2- Segunda parte.

En esta parte encontramos el nodo *Set Simulate Physics* que habilita al vehículo para simular efectos físicos (como la gravedad), el nodo *add Force at location* que permite efectuar las fuerzas de compensación (subir el vehículo o bajarlo) que vimos en el grafico anterior, el nodo *Set Linear Damping* o amortiguación lineal que se usa para indicar al vehículo la resistencia que tiene a acelerar (cuanto mayor sea el número más cuesta aumentar su velocidad) y el nodo *Set Angular Damping* o amortiguación angular que se usa para indicar al vehículo la resistencia que tiene a girar (cuanto mayor sea el número más cuesta realizar giros sobre su eje).

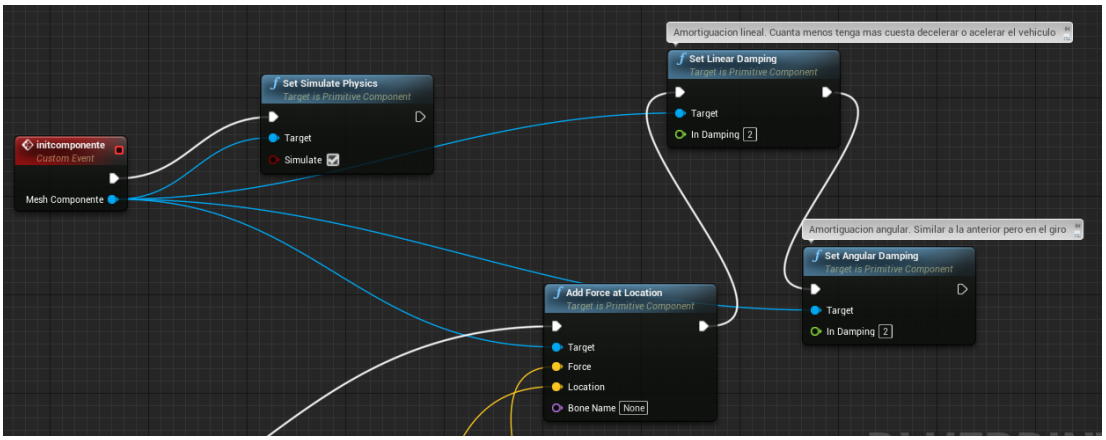


Ilustración 74: vehículo componente 3

- **Vehículo** (ver Ilustración 69, 70, 71, 72, 73): Este *Blueprint* está compuesto por el vehículo como tal (objeto físico y texturas) y sus acciones. Se dividirá en secciones para facilitar su explicación.

1- **Inicialización de las físicas del vehículo**: se inician los 4 componentes de generación de físicas para el vehículo.

El nodo Event *BeginPlay* se activa cuando comienza el juego y está unido a los nodos que activan el *Blueprint Vehiculocomponente*

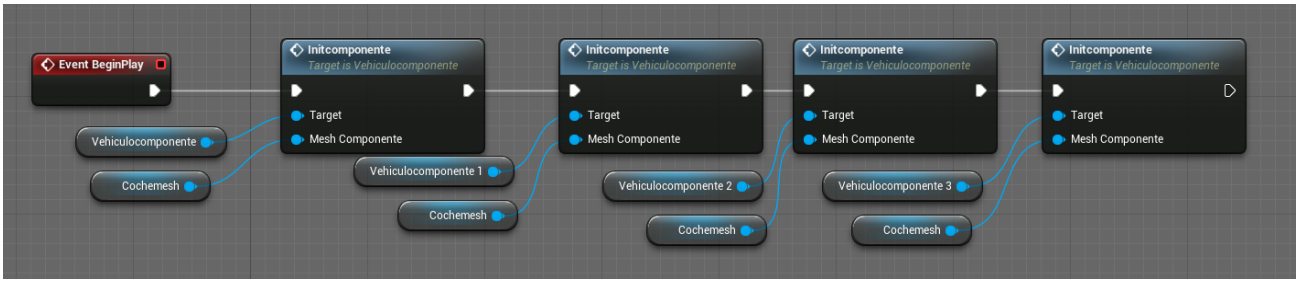


Ilustración 75:Vehiculo 1

2- **Movimiento hacia delante/atrás**: Al igual que el personaje en 1 persona, necesitamos vincular la acción de la tecla W/S, flecha hacia arriba/abajo o Stick inclinado hacia delante/atrás al movimiento correspondiente.

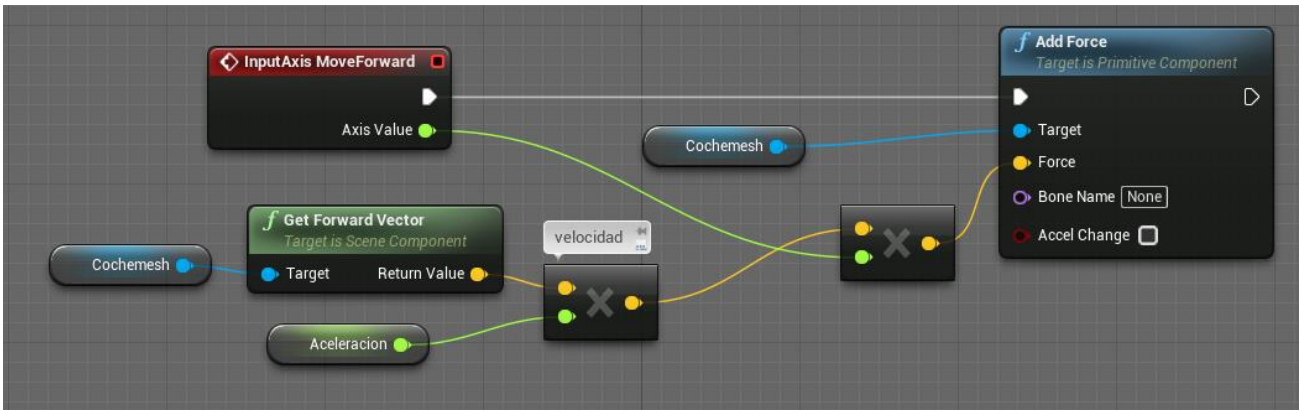


Ilustración 76: Movimiento Vehiculo 1

El nodo InputAxis MoveForward va a activarse cuando usemos los botones asignados en la configuración del proyecto para el movimiento hacia delante/atrás. En este caso contamos con una variable llamada aceleración (que podemos modificar como veremos en la sección del "turbo" que multiplicaremos por el vector de movimiento del vehículo.

El valor resultante se multiplica por el valor de Axis Value (Este valor es positivo si elegimos mover hacia delante y negativo si lo hacemos hacia atrás) y se envía al nodo Add Force que actúa sobre el vehículo (llamado en el Blueprint cochemesh).

3- **Movimiento izquierdo/derecha**: El movimiento lateral del vehículo es más complejo que el del personaje en primera por cómo está diseñado el propio Blueprint.

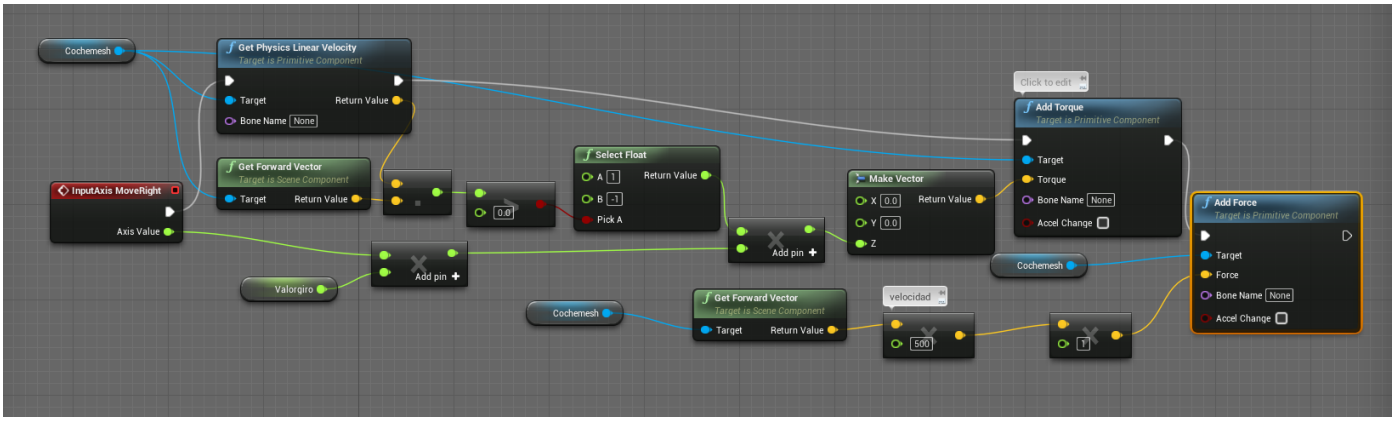


Ilustración 77: Movimiento Vehiculo 2

El nodo *InputAxis MoveRight* se activa siempre que se usen los botones configurados para mover al vehículo de forma lateral (son los mismos que se usan en el movimiento lateral del personaje en primera persona).

Dependiendo de si el giro es a la derecha (valor positivo) o a la izquierda (Valor negativo) crearemos un vector de movimiento (nodo Make Vector) cuyo valor será el producto del vector de movimiento actual del vehículo (siempre que sea diferente de cero, ya que si está parado el vector será

0 y el vector resultante sería nulo) y el valor total del giro (valor que está establecido entre -1 y 1 dependiendo de si es izquierda o derecha). Este valor resultante se envía al nodo *Add Torque* que aplica la fuerza de giro al vehículo.

Además, se le añade una fuerza hacia delante debido a un error que hace que el vehículo no gire más de 180º si está parado (nodo *Add Force*).

4- **Zoom de la cámara**: se puede acercar y alejar la visión del vehículo usando el *scroll* del ratón.

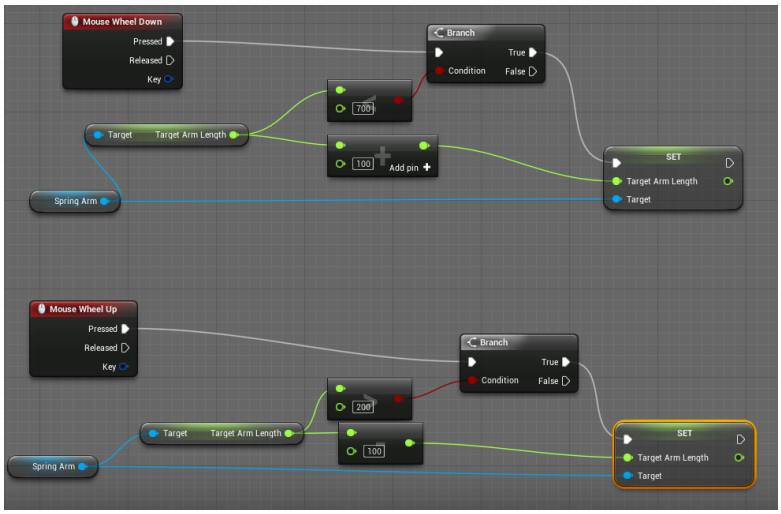


Ilustración 78: Zoom vehículo

Para los dos casos (alejar y acercar) usamos los nodos establecidos por UE4 para el uso de la rueda del ratón (Mouse Wheel Down y up).

La variable predeterminada por UE4 Arm Lenght establece la distancia entre la cámara y el vehículo. En caso de alejar la cámara (girando la rueda del ratón hacia delante) se usa el nodo suma para añadir 100 puntos por giro a la distancia que tenga la variable Arm Lenght (nodo set). Para el acercar la cámara se realizan las mismas operaciones con la diferencia de que se usa el nodo resta.

5- **Velocidad extra**: Esta operación trata de añadir un efecto de velocidad extra al vehículo.

Se activa con el evento pulsar barra espaciadora (nodo *Space Bar*) y tiene dos acciones. La primera es pulsar la tecla, que tras un retraso de 0.2 segundos (nodo *Delay*) ajustará la velocidad de vehículo en 320.000 unidades (nodo *SET*).

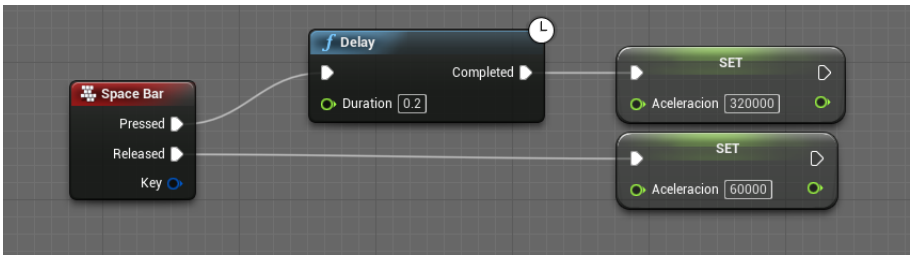


Ilustración 79: Vehículo velocidad

En caso de que se deje de pulsar se volverá a ajustar la velocidad a la preestablecida (nodo SET con 60.000 unidades).

5.1.2.2. Interacciones

- **Supervivientes** (ver Ilustración 74): al pasar por encima de los supervivientes los recogerá.

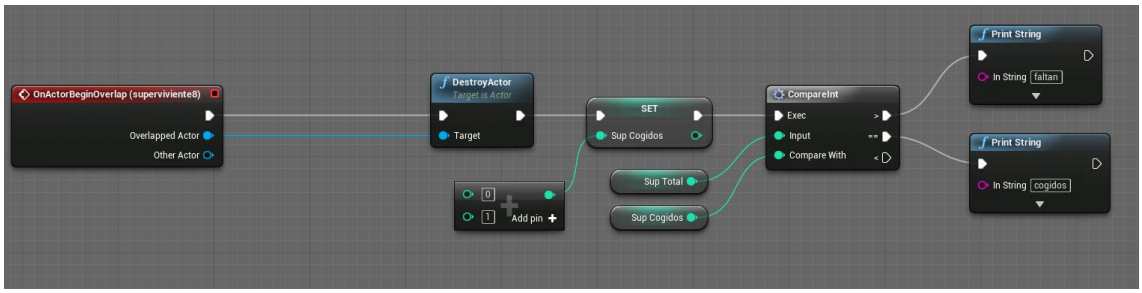


Ilustración 80: supervivientes

El evento *OnActorBeginOverlap* (superviviente X) se activa cuando el jugador pasa por encima de algún superviviente. El nodo DestroyActor elimina el superviviente recogido y actualiza la variable Sup cogidos (es el número de supervivientes recogidos).

El nodo Compareint va a comparar el número de supervivientes recogidos con el número de supervivientes totales para mostrar por pantalla si se han recogido todos o aún faltan.

- **Contacto con lava y reinicio**: Estos dos eventos tienen el mismo diagrama, por lo que están unidos. En un caso se activa cuando tocamos la lava y en el otro se activa en caso de que el vehículo este inutilizado y haya que reiniciar el nivel.

En ambos casos se usa el nodo *Move Component To* para cambiar de posición al vehículo. La posición a la que se mueve esta preestablecida por la posición en la que se encuentra el elemento Puntoreinicio.

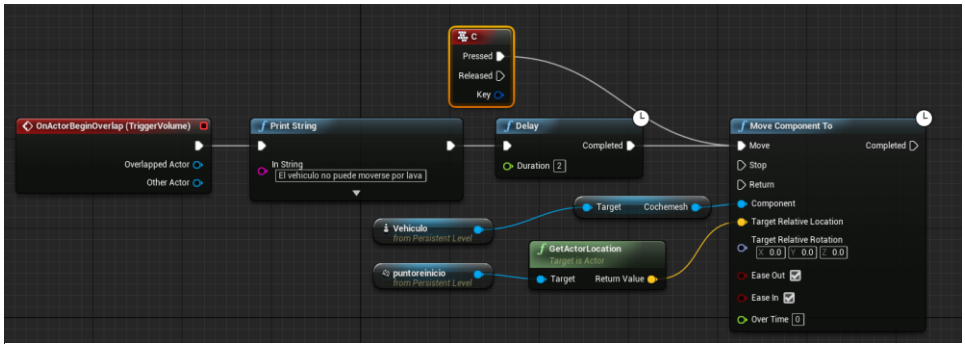


Ilustración 81: Lava vehículo

5.2. ENEMIGOS

Esta sección describe el diseño de los *Blueprints* relacionados con enemigos con más detalle. Se ha dividido en dos secciones para diferenciar aquellos enemigos que tienen movimiento de los que carecen de él.

5.2.1. Con movimiento

En muchas ocasiones se confunde IA (inteligencia artificial) con seguimiento del jugador. Si bien es verdad que la inteligencia artificial se define como "Programa de computación diseñado para realizar determinadas operaciones que se consideran propias de la inteligencia humana, como el autoaprendizaje" y que cualquier *Blueprint* o código diseñado para perseguir al jugador sin ningún tipo de consideración o razonamiento no puede considerarse inteligencia artificial, este tipo de mecánicas sí que se consideran como inteligencia artificial desde el punto de vista de la estructuración de las clases y nodos en UE4.

Este enemigo está diseñado para perseguir al usuario en caso de que este entre dentro de un cono de detección y que muera a los 8 segundos de iniciarse la persecución. En caso de que el enemigo entre en contacto con el jugador, este perderá un porcentaje de vida.

5.2.2. COMPONENTES DEL ENEMIGO

- **CapsuleComponent y Capsule** (ver Ilustración 76, 77): estas dos capsulas establecen el área que interactúa con el jugador. Al entrar en contacto con estas dos capsulas se genera el evento que desencadena el daño del enemigo al jugador
- **ArrowComponent** (ver Ilustración 78): muestra la dirección del vector de movimiento (flecha azul).
- **Mesh y SkeletalMesh** (ver Ilustración 78): Modelo 3d del esqueleto del enemigo.
- **CharacterMovement**: da la propiedad al personaje de moverse. Vamos a encontrar dentro de este componente todas las variables relacionadas con movimiento divididas en las diferentes formas en que podemos encontrarlo (andar, saltar/caer, nadar, volar, movimiento personalizado, interacción de físicas, etc.).

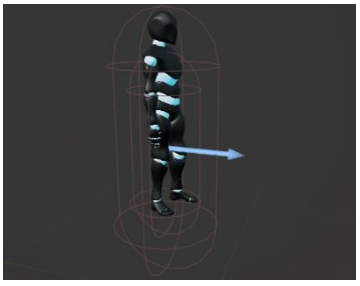


Ilustración 82: Enemigo 3

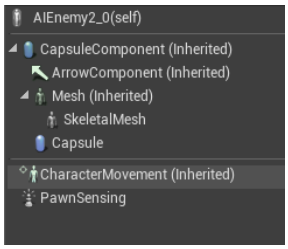


Ilustración 83: Enemigo 1

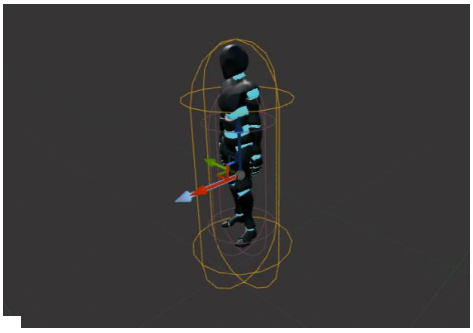


Ilustración 84: Enemigo 2

- **PawnSensing** (ver Ilustración 79): componente que añade un cono de detección (cono de color verde) y genera el evento *OnSeePawn* siempre que el jugador se encuentre dentro de esa área.

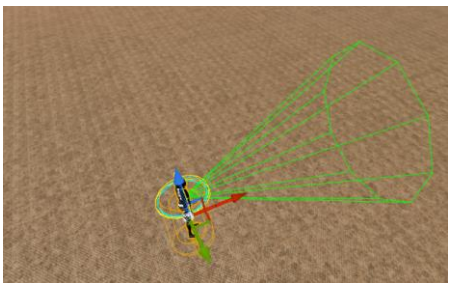


Ilustración 85: Enemigo 4

5.2.2.1. Blueprint

En esta sección se explicarán las dos funciones que se encuentran en el *Blueprint* del enemigo.

5.2.2.1.1. Enemigo que persigue al jugador

Cuando el jugador entra en contacto con el cono de detección se activará un sonido (nodo *play sound*) y el enemigo se moverá hacia el jugador (nodo *Simple Move to Actor*).

El siguiente elemento es un temporizador de 8 segundos, tras los cuales se destruirán las capsulas de detección del enemigo (nodo *Destroycomponent* con objetivo *Capsule y Capsule component*). Finalmente el nodo *Set Simulate Physics* hace que se activen las físicas de movimiento del esqueleto del enemigo y caiga al suelo simulando que ha muerto.

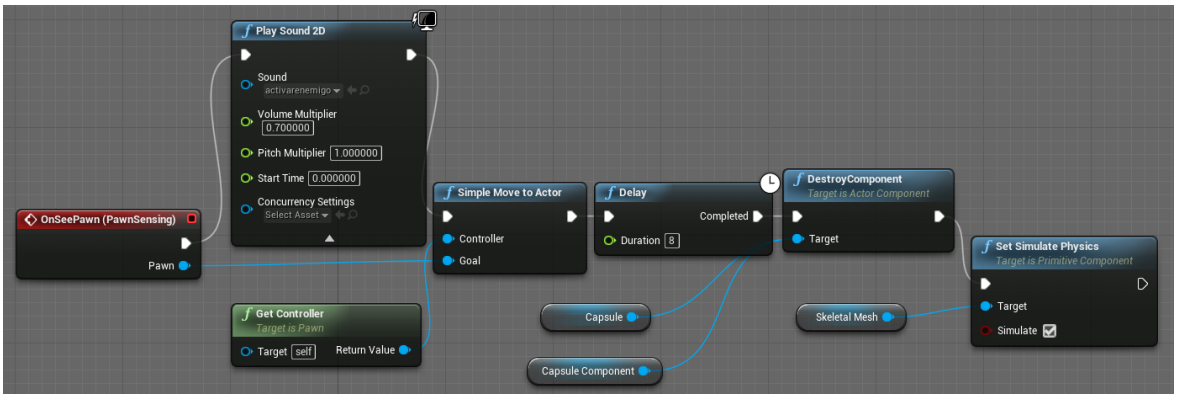


Ilustración 86: Enemigo 4

5.2.2.1.2. Daño al jugador

Cuando el jugador entra en contacto con la capsula del enemigo se va a actualizar su vida mediante el nodo set (el valor que será la vida actual menos 0.5), posteriormente tendremos una secuencia (nodo *Sequence*) con dos caminos.

- Then 0: en el primer camino volvemos a hacer lo mismo que en "Seguir al jugador". El enemigo perderá sus capsulas de detección y simulará físicas para fingir su muerte.

- Then 1: el primer nodo (*Comparefloat*) comparamos la vida actual del jugador con 0 para ver si el jugador está muerto (entonces se muestre un aviso de muerte y saldremos al menú principal).

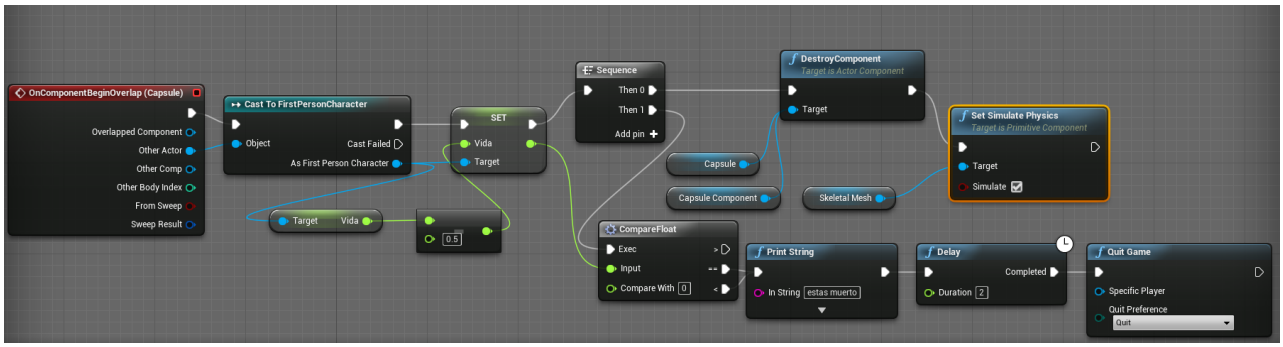


Ilustración 87: Enemigo 5

5.2.3. Sin movimiento (minas)

Las minas serán junto a la lava el único enemigo estático que nos reste vida. Su diagrama será el siguiente.

Las minas se activan con el nodo *OncomponentBeginOverlap* (cuando entramos en contacto con el modelo de la mina). El nodo *Play Sound 2D* emite un sonido de explosión.

El siguiente paso es realizar un casting sobre el *Blueprint* del jugador (nodo *Cast To FirstPersonaCharacter*) con el fin de actualizar su vida (Nodo *set* con un valor de la vida del jugador menos 0.1). El siguiente paso es, mediante el nodo *CompareFloat* (comparamos la vida del jugador con el valor 0), ver si el jugador ha perdido toda su vida.

Si su vida total no es igual a 0 se elimina la mina y si el jugador ha perdido toda su vida se muestra el menú de muerte.

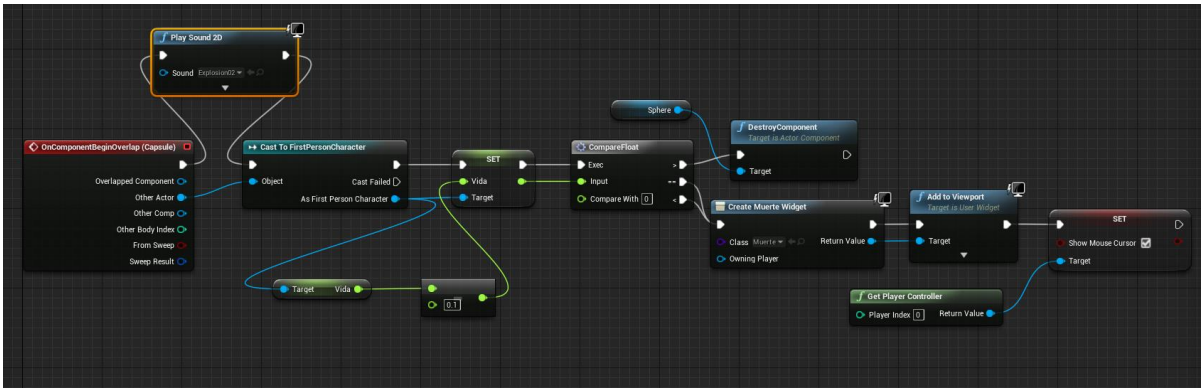


Ilustración 88: Minas daño

5.3. POWER-UP

5.3.1. Supervelocidad

Cuando el jugador entra en contacto con la caja de colisión del power-up se muestra por pantalla el mensaje de velocidad aumentada (nodo *create Supervelocidad*) y posteriormente se realiza un casting para modificar el valor de la velocidad máxima del personaje (nodo *cast to FirstPersonCharacter* y nodo *SET con valor 1900*).

Una vez aumentada la velocidad se establece un retraso de 2 segundos (nodo *Delay)* tras los cuales se elimina el mensaje mostrado por pantalla, tras otros 3 segundos se vuelve a asignar el valor normal de la velocidad máxima del personaje (nodo *SET con valor 750)*. Finalmente se elimina del nivel tanto el activador como el objeto que representa el power-up (nodo *DestroyComponent*).

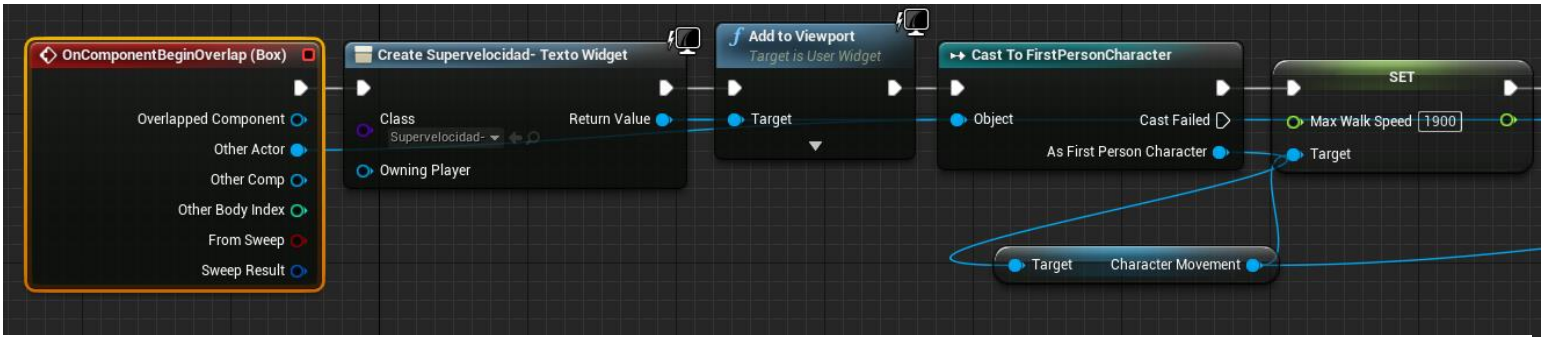


Ilustración 90: velocidad extra 1

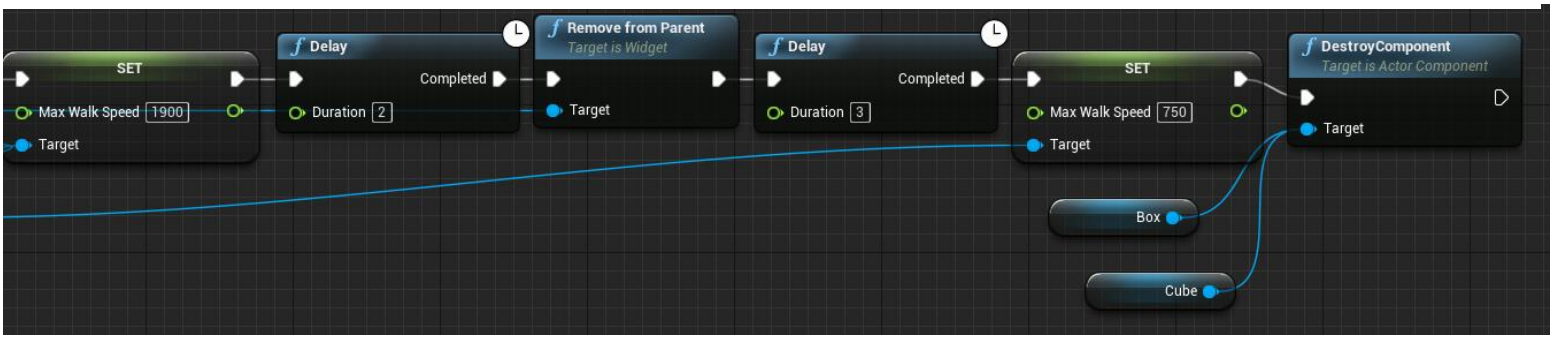


Ilustración 89: velocidad extra 2

5.3.2. Supersalto

Cuando el jugador entra en contacto con la caja de colisión del power-up se muestra por pantalla el mensaje de salto aumentado (nodo create Supersalto) y posteriormente se realiza un casting para modificar el valor de la velocidad máxima a la que el personaje se mueve en el eje Z (nodo cast to FirstPersonCharacter y nodo SET con valor 700) permitiendo de este modo subir hacia arriba más rápido que antes y generando el efecto de que el jugador salta más.

Una vez aumentada la velocidad en el eje Z se establece un retraso de 2 segundos (nodo Delay) tras los cuales se elimina el mensaje mostrado por pantalla, tras otros 3 segundos se vuelve a asignar el valor normal de la velocidad máxima del personaje (nodo *SET con valor 450)*. Finalmente se elimina del nivel tanto el activador como el objeto que representa el power-up (nodo *DestroyComponent*).

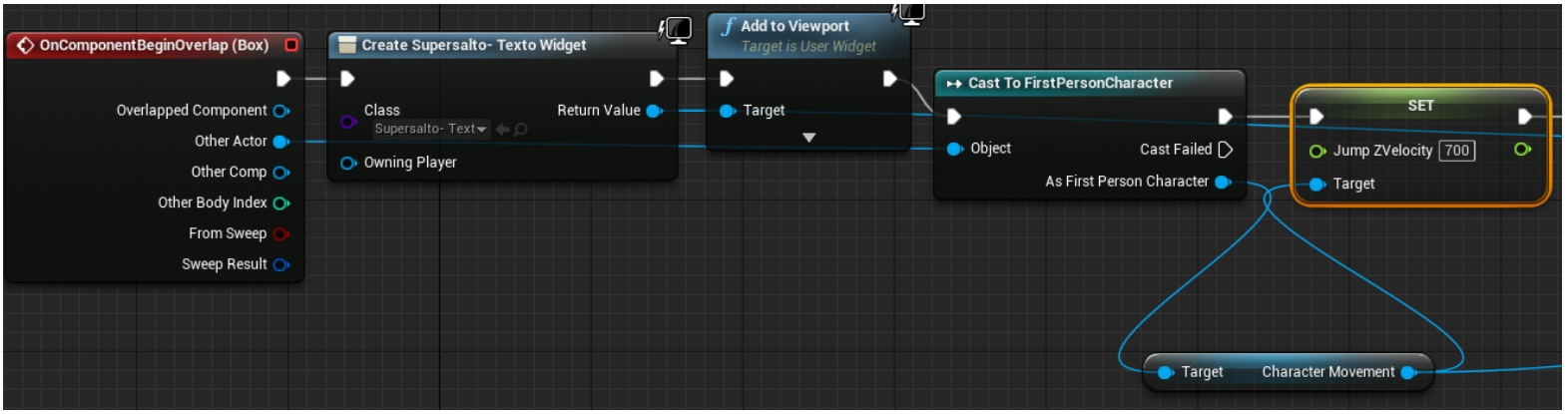


Ilustración 92: supervelocidad 1

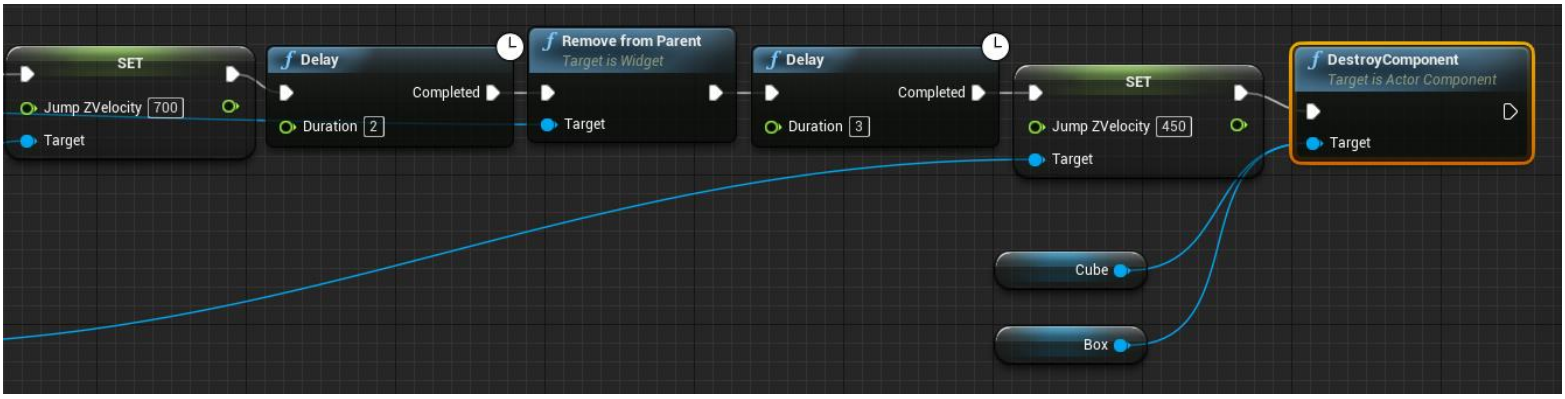


Ilustración 91: supervelocidad 2

5.4. OTROS OBJETOS

5.4.1.Luces

Mediante el uso de *Event Tick (*evento que se activa de forma aleatoria*)*, la lámpara se activa o desactiva (nodo *Toggle Visibility* de forma aleatoria. Para evitar que parpadeen todas igual, se han creado dos *Blueprint*. En uno de ellos se añade un nodo *Random Float In Range* mas un nodo *Delay* para que el encendido y apagado no dependa solo del nodo *Event Tick*.

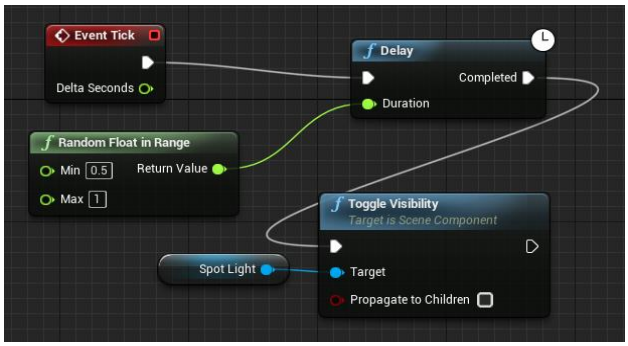


Ilustración 94: Luces 2

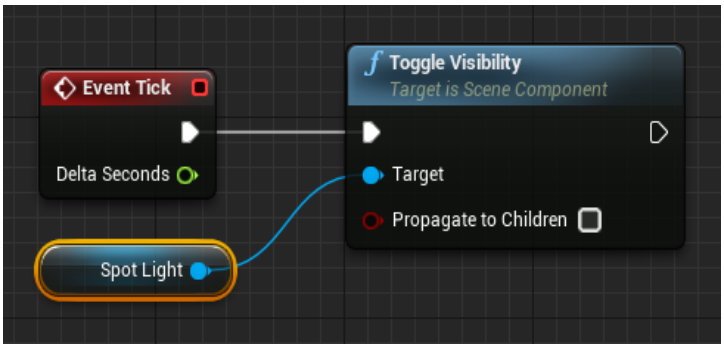


Ilustración 93: Luces 1

5.5. BLUEPRINT DEL MUNDO

En esta sección se encuentran los blueprints que pertenecen a los diferentes niveles

5.5.1. Teletransporte al caer a la lava

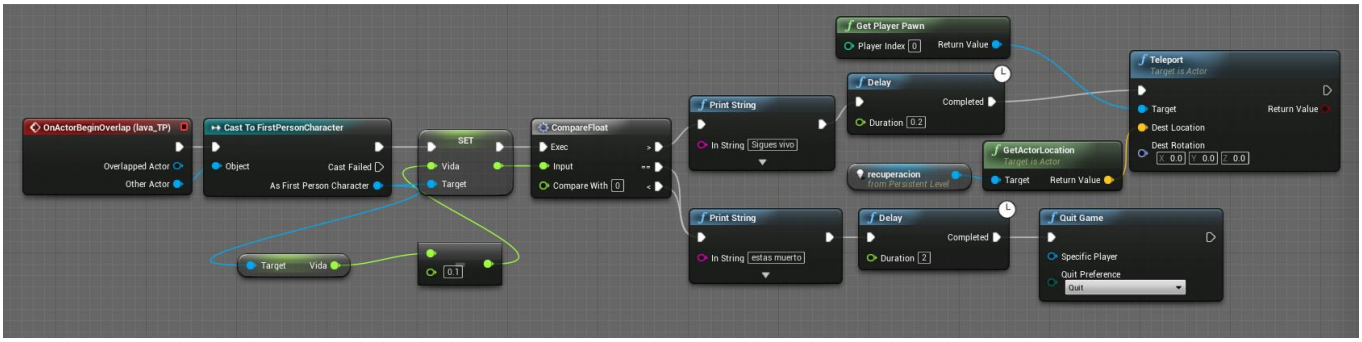


Ilustración 95: Teletransporte lava

Esta función entra en acción si el jugador cae a la lava en el primer laberinto. Cuando entra en contacto con la lava (nodo *Onactorbeginvoverlap*) actualizamos su vida (nodo set con valor: vida – 0.1) para luego en comparar la vida y detectar si aún le queda (nodo *compareFloat*).

En caso de que la vida no sea igual a 0 teletransportamos al jugador al inicio de la prueba y en el caso contrario mostramos el mensaje de muerte y volvemos al menú de inicio.

5.5.2. Recoger llaves

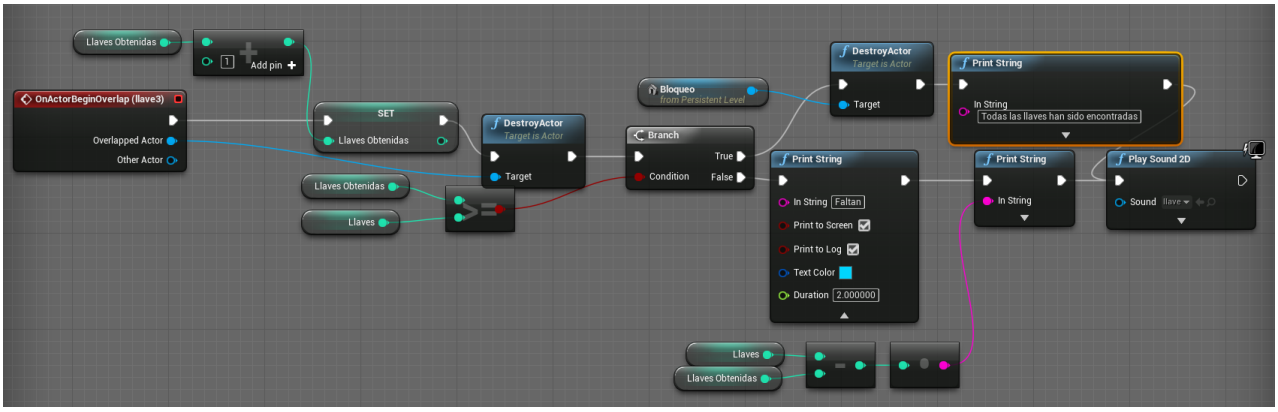


Ilustración 96: Recoger llave

Esta función permite el acceso al siguiente nivel siempre que se encuentren las 3 llaves repartidas a lo largo del nivel.

Cuando se activa el evento OnActorBeginOverlap (cuando entramos en contacto con la llave) sumamos al contador de llaves obtenidas un uno (nodo SET llaves obtenidas) y destruimos el actor llave para no poder volverá a cogerlo.

El siguiente nodo es un I cuya condición es si el número de llaves obtenidas es mayor o igual que las llaves totales.

Si el resultado es afirmativo eliminamos el actor que bloque el acceso al siguiente nivel y si por el contrario el resultado es negativo indicamos en pantalla cuantas llaves faltan (restando las que tenemos a las totales) En ambas opciones se reproduce el sonido correspondiente a coger la llave.

5.5.3. Recoger explosivo

Esta función se activa cuando pasamos encima del explosivo.

Su objetivo es el de eliminar el objeto "explosivo" y permitir el acceso al *Blueprint* "explosión de pared"

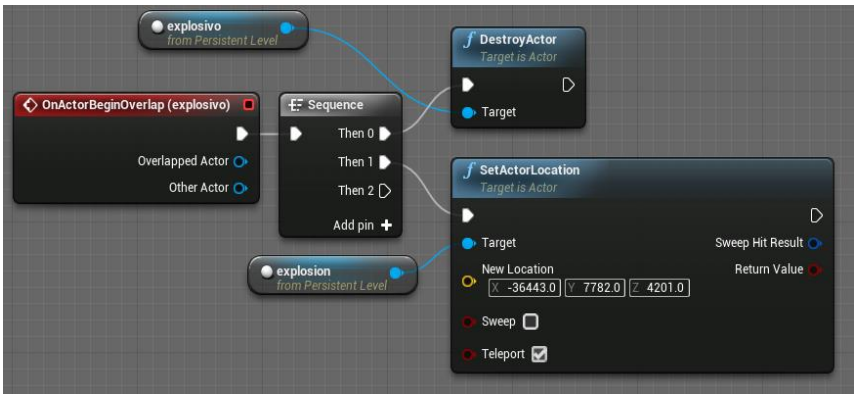


Ilustración 97: Recoger explosivo

5.5.4. Altar mágico

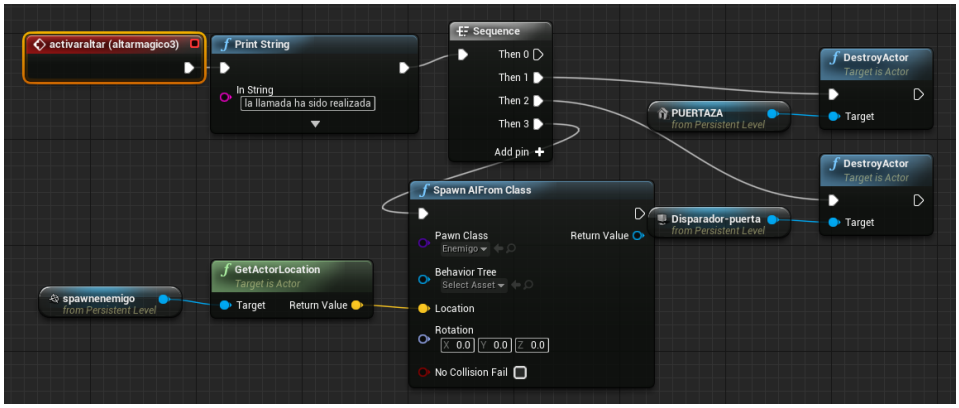


Ilustración 98: Altar mágico 1

Accedemos a esta función a partir del *Blueprint* Altar mágico y se encuentra fuera de este debido a que modifica elementos situados en el mundo y un *Blueprint* no puede modificar elementos ajenos a sí mismo.

Como vimos en "altar mágico" había una llamada a un evento llamado "activaraltar" que es el que encontramos en esta función dentro del *Blueprint* del nivel.

Esta función comienza con una impresión de texto en la pantalla y la siguiente secuencia

1. Se destruye el actor PUERTA (puerta que encontramos en el pasillo)

2. Eliminamos el disparador (la “box” del altarmagico) para evitar que pueda ser activada de nuevo.

3. Con la tarjeta *SpawnIA* creamos un enemigo y lo posicionamos haciendo uso de una variable de localización.

   Esta variable de localización se posiciona en el mapa y de este modo recogemos la posición exacta donde queremos posicionar al enemigo.

   Encontramos otros 3 altares más que no se incluye en esta documentación al ser la misma función.

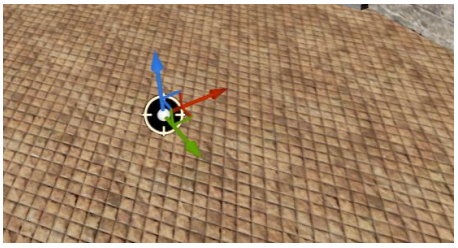


Ilustración 99: Altar mágico 2

5.5.5. Abrir el menú InGame

Esta función comienza con el evento “pulsar tecla escape”. Una vez pulsado se pausa el juego (nodo Set Game Paused) y se muestra mediante la interfaz creada (nodo Create Ingame Menu Widget con la clase Ingame Menu, que es la clase donde está la interfaz).

Además, con el nodo *SET* permite al jugador mover el ratón y poder seleccionar las opciones mostradas.

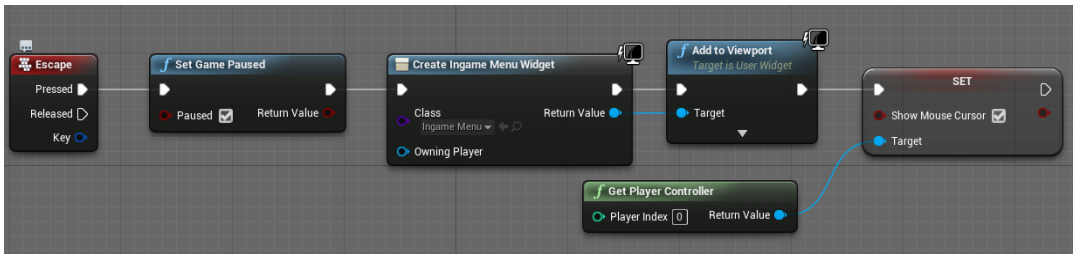


Ilustración 100: Menú 1

5.5.6. Elevador

Cuando entramos en contacto con la plataforma elevadora (imagen superior derecha) se activa un el nodo movimiento. Este es un *Blueprint* que realiza una animación de movimiento vertical.

Ilustración 102: Elevador 1

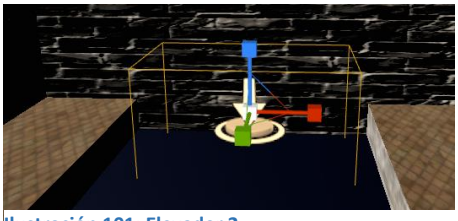


Ilustración 101: Elevador 2

5.5.7. Teletransporte a nuevo nivel

Cuando entramos en contacto con el objeto Siguientenivel (imagen superior derecha) abrimos el siguiente nivel mediante el uso del nodo *Open Level*

Ilustración 104: Siguiente nivel 1

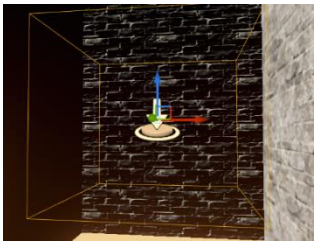


Ilustración 103: Siguiente nivel 2

5.5.8. Teletransportación dentro de un nivel

Esta función está diseñada para mover al jugador entre diferentes tipos de zonas dentro del juego.

Se hace uso del nodo Teleport predefinido por UE4 4 y que tiene tres entradas. La primera es el nodo OnActor-BeginOverlap o lo que es lo mismo, cuando el jugador entre en contacto con la caja naranja del teleport.

La segunda es el objetivo de la teleportación, en este caso usamos el nodo Get Player Pawn para indicar que es el propio jugador. La tercera es el punto donde vamos a transportar al jugador. En este caso hemos usado otro objeto similar al de la imagen superior derecha para posicionarlo en el mundo y recoger mediante el nodo *GetActorLocation* su localización.

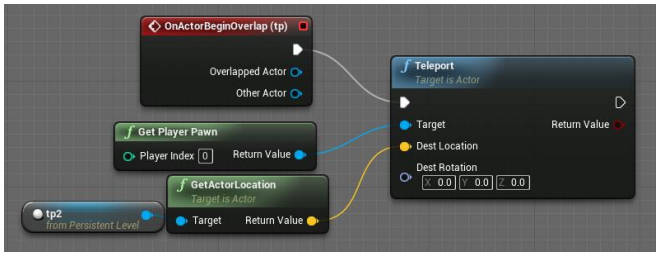


Ilustración 106: Teletransportación 1

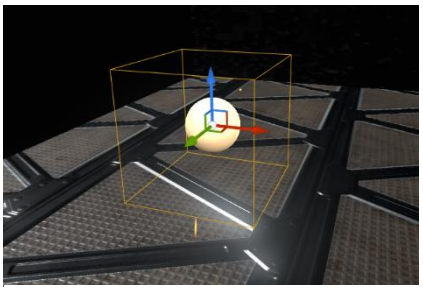


Ilustración 105: Teletransportación 2

5.6. BOB Y EL CHAT

El sistema de diálogos con el personaje no jugador está realizado con 4 *Blueprints* que veremos a continuación

5.6.1. *Blueprint* principal de Bob

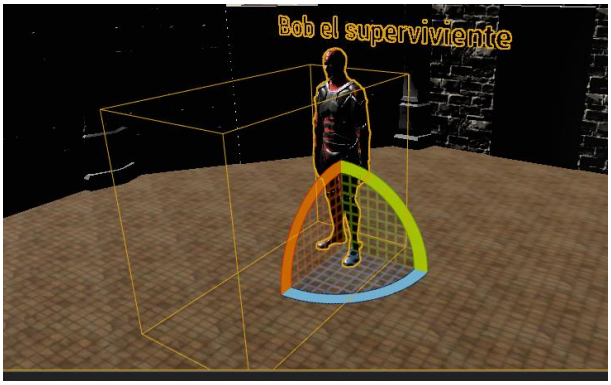


Ilustración 107: Bob 1

Este *Blueprint* es de tipo "actor" por lo que tendrá asociado un esqueleto, textura y su función será la de activar el diálogo.

Para ello tenemos 2 funciones importantes:

- Detectar inicio de dialogo: para indicar al jugador cuando puede comenzar el dialogo con Bob, este tendrá un cartel de color amarillo que cambiará de color cuando el jugador entre dentro de la caja de activación.
  Haciendo el uso del nodo *OncomponenteBeginOverlap* (cuando el jugador se sitúe dentro de la caja naranja que podemos ver al inicio de este punto) se cambia el color del texto mediante el nodo *Set Text Render Color*. En el momento en el cual se active el nodo *OncomponentEndOverlap* (dejemos de estar dentro de la caja naranja) volveremos a cambiar del mismo modo el color del texto, tornándolo al color amarillo inicial.

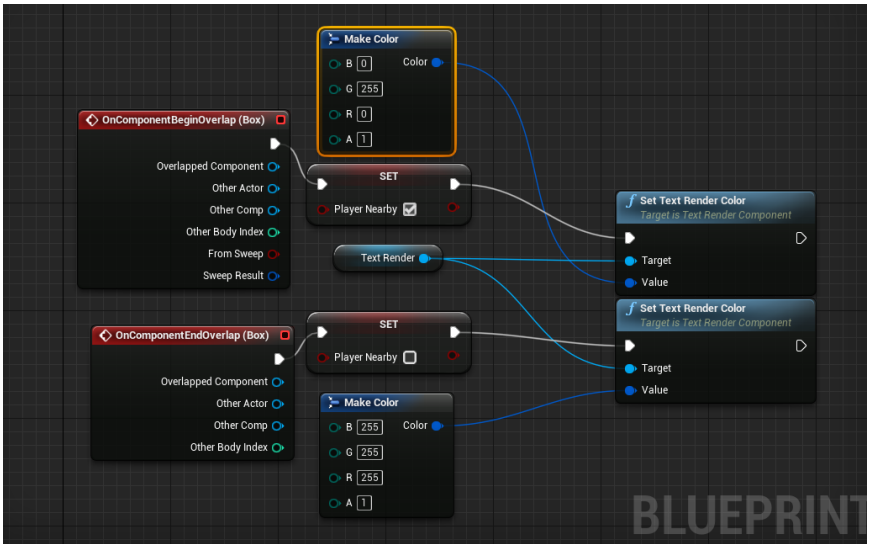


Tabla 16: Bob 2

- Iniciar el chat con la tecla E: se podrá iniciar el chat pulsando la tecla "E".

Esta función comienza al pulsar la letra E y comprobar en el nodo *Branch* (bucle *if*) que estamos dentro de la caja amarilla. En caso de que esta comprobación devuelva el valor true desactivaremos la posibilidad de mover al jugador (nodo *Disable input*) y mostraremos el *Blueprint* de tipo *Widget Dialogue* (en el cual tenemos la interfaz del chat).

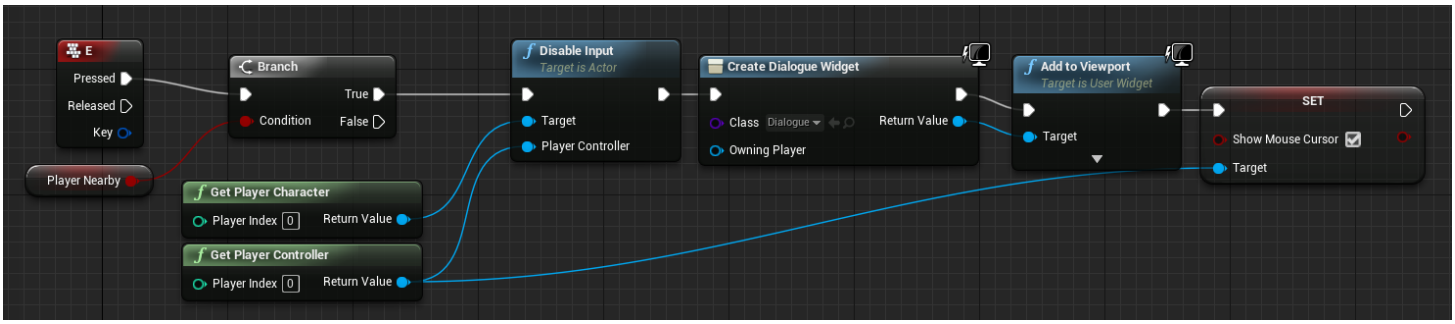


Ilustración 108: Bob 3

5.6.2. Estructura de variables

En este *Blueprint* encontramos las variables que usaremos para guardar los textos que más tarde se utilizan para mostrar el diálogo.

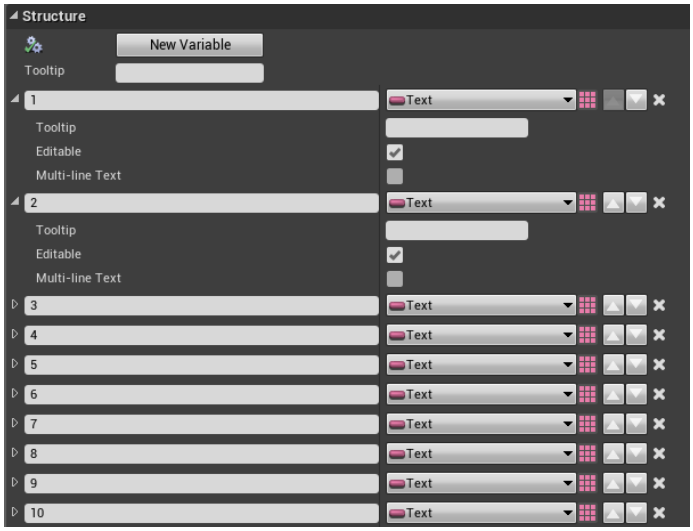


Ilustración 109: Bob 4

En esta estructura se indican las variables a usar en la lista de diálogos. Como en este dialogo solo se usará texto plano, todas las variables son de tiempo Text.

5.6.3. Lista de diálogos

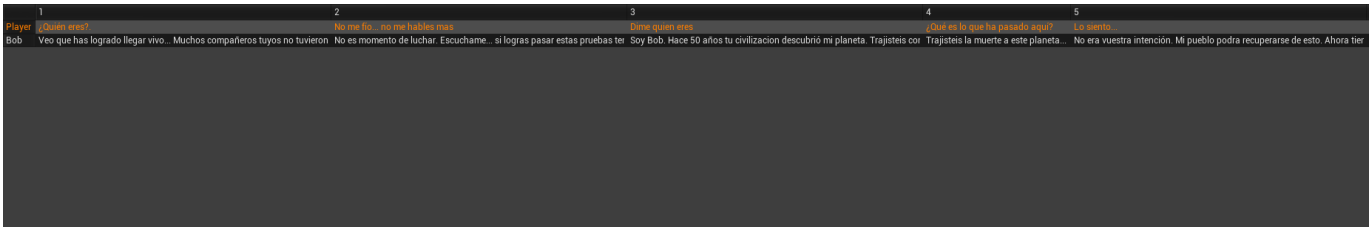


Ilustración 110: Bob 5

Aquí encontramos todos los diálogos escritos y numerados para su uso en el *Blueprint* que veremos a continuación. La tabla está dividida en una fila para Bob y otra para el propio jugador. En cada fila se encuentra todas frases que podremos seleccionar para poner en el dialogo

5.6.4. *Blueprint* Widget

Este Blueprint se divide en dos:

1. Interfaz:

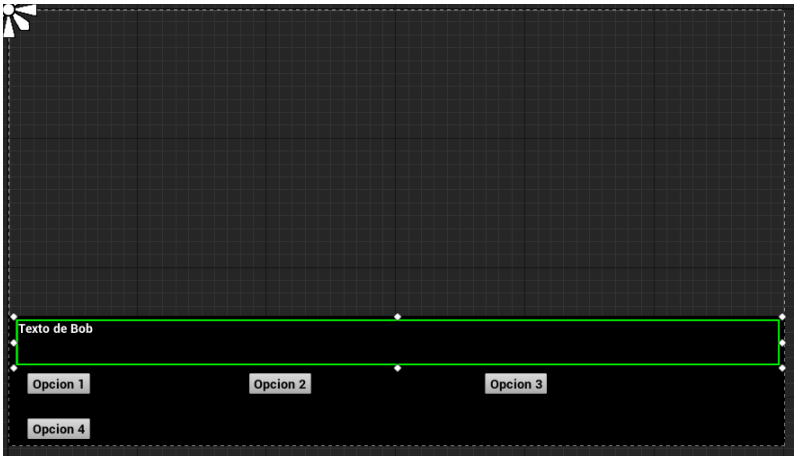


Ilustración 111: Interfaz Bob 1

- Parte superior: en la imagen se identifica por el recuadro verde y es la zona donde se mostrará el texto de Bob.
- Parte inferior: aquí se encuentran los botones donde se mostrarán las opciones asociadas al texto de la parte superior.

  En el caso de que un texto no tenga asociado más de dos opciones, los sobrantes no se mostrarán.

2. Código visual

Este *Blueprint* es el más importante ya que es el que va a mostrar en la pantalla los diálogos y nos permitirá elegir las diferentes respuestas.

Al ser muy extenso se explica por partes.

- Cargar los datos de la lista de diálogos.

  En la imagen inferior vemos el nodo *Event Construct* que da comienzo al dialogo y a continuación se encuentran los nodos que se encargaran de recoger los valores de la tabla de dialogo (nodos *Get Data Table Row* dialogo_lista).

  En este caso hay dos nodos ya que uno se usará para recibir la fila correspondiente al dialogo de Bob y el otro nodo para la fila correspondiente al dialogo del propio jugador.

En los dos nodos que encontramos abajo (*Break* Dialogo_estructura) se encuentran las variables de dialogo numeradas para poder elegir cual se quiere mostrar.

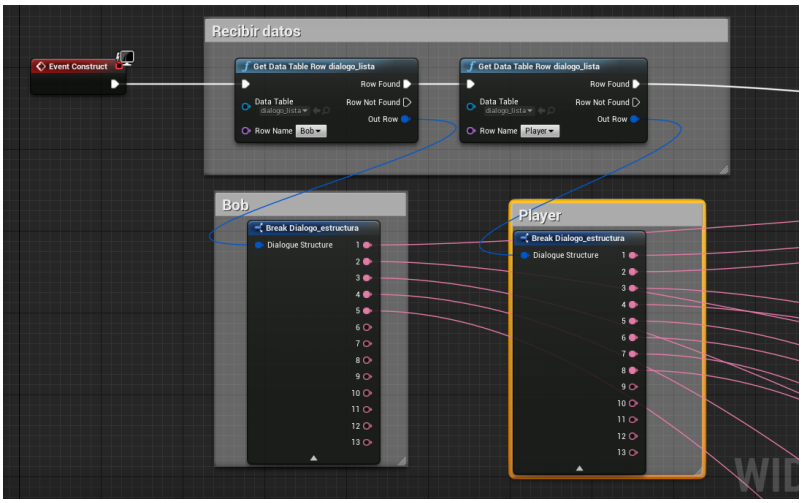


Ilustración 112: Interfaz Bob 2

- Mostrando los textos en pantalla: esta parte se encarga de mostrar los textos tanto en los botones como en la parte del texto de bob. Solo se explica la primera parte del dialogo con Bob ya que el resto de conversaciones se estructuran del mismo modo.

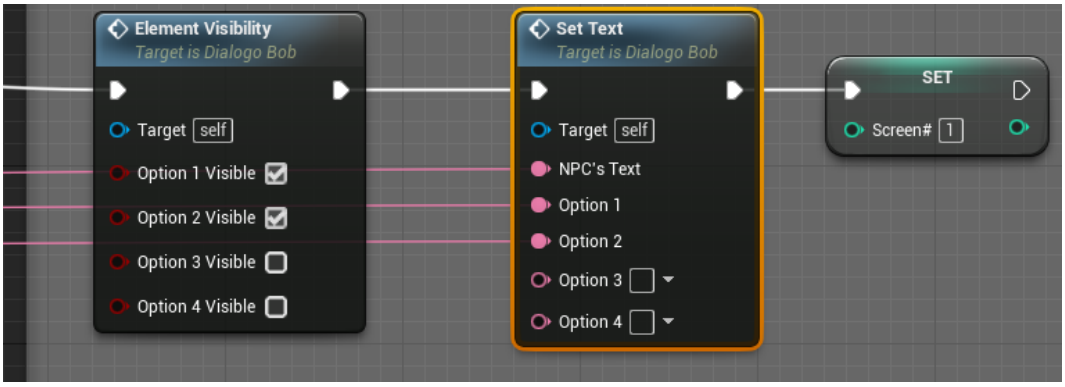


Ilustración 113: interfaz Bob

El primer nodo (Element Visibility) nos permite seleccionar que partes de la interfaz queremos mostrar. Como en esta pantalla solo tenemos dos opciones, están marcadas como True la opción 1 y la opción 2 (las otras dos no están marcadas, por lo que no saldrán).

El siguiente nodo (Set Text) se encarga de mostrar en cada sección de la interfaz (texto de bob y botones de opción) el texto seleccionado.

Para elegir el texto primero hay que elegirlo en la tabla de dialogo y posteriormente engancharlo a la opción deseada.

EJEMPLO:

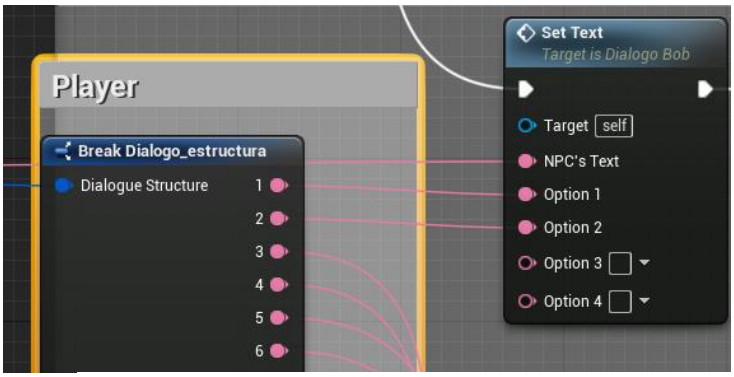


Ilustración 114: Interfaz Bob 4

El nodo de la derecha es el que estamos viendo en esta sección.

El nodo de la izquierda pertenece a la sección anterior y se en él se guarda el texto perteneciente al jugador en la tabla de dialogo en variables de texto.

Es por ello por lo que la primera variable de dialogo se une con Opción 1. Eso significa que en el botón 1 de nuestra interfaz se encontrará el texto perteneciente a la primera posición de la tabla de dialogo.

Lo mismo pasará con la opción dos, que está unida con la segunda posición. Se puede observar que también hay un enganche llamado *NPC's Text*. Este será el texto que se mostrará en la parte de Bob (parte superior de la interfaz) y estará enganchado al nodo en el que se guardan las variables de texto pertenecientes a Bob.

- Cambiando entre niveles de dialogo:

Cuando se elige una de las diferentes opciones del dialogo se mostrarán diferentes respuestas con sus correspondientes opciones.

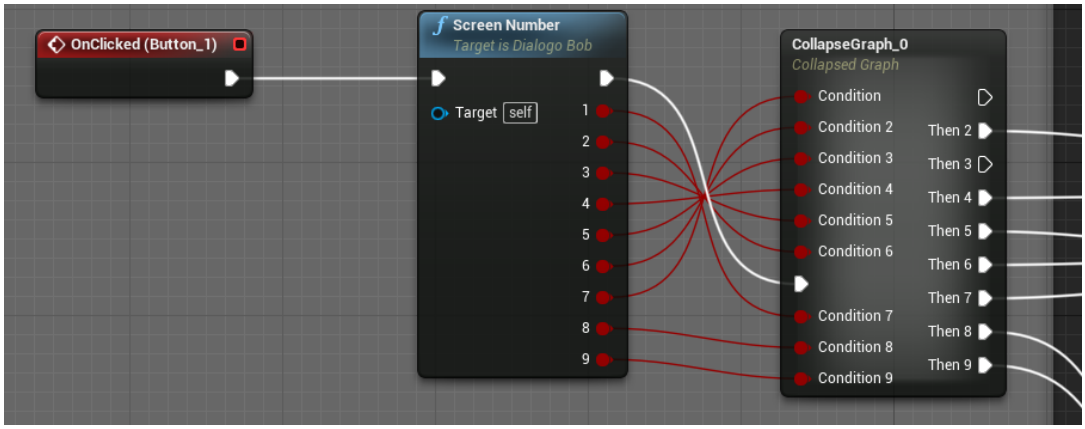


Ilustración 115: Interfaz Bob 5

Este es el ejemplo de si seleccionamos el botón uno en la conversación con Bob y habrá uno para cada opción.

Al generarse el evento *OnClicked* (hacer *click* en el botón 1) entramos en un nodo (*Screen Number*) que detecta en que pantalla estamos. Como estamos en la pantalla 1, el nodo devuelve el valor 1 y en el siguiente nodo (*CollapseGraph_0*) selecciona la condición que corresponde a la pantalla uno (tenemos que seguir la línea que une el valor 1 del nodo *Screen Number* con el nodo *CollapseGrap_0*).

En este caso el valor es 7 por lo que la pantalla que se abrirá a continuación será la que sigue a la línea que sale desde *Then* 7.

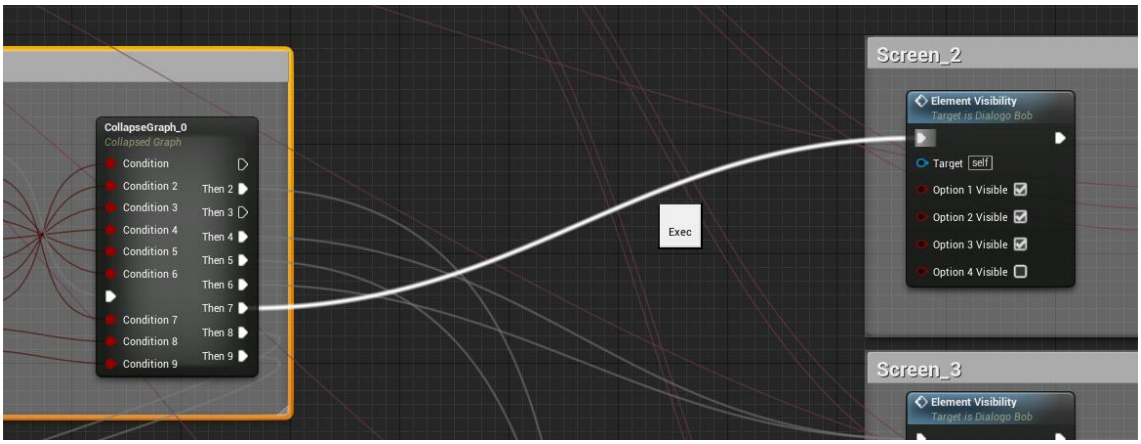


Ilustración 116: interfaz Bob 6

Será entonces cuando la segunda pantalla de dialogo se abra y muestre el texto correspondiente. Esta estructura continua para el resto de ventanas de dialogo.

- Botón de cierre del dialogo:

  En el caso del botón que cierra el dialogo con bob, el enlace que se puede ver en la imagen superior no apunta a otra ventana si no a un nodo de tipo *Dialogue_Exit* que se encarga de cerrar el dialogo y permitir al usuario retomar el control del personaje.

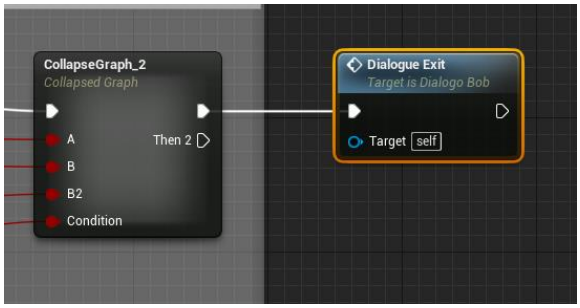


Ilustración 117: interfaz bob 7

5.7. PERSISTENCIA DEL JUEGO

En esta sección se explicarán los Blueprints de carga y guardado del videojuego. Este juego guarda la posición donde estaba situado el jugador y su vida. Se crea un fichero de guardado donde se guarda una variable de tipo *Float* (vida del jugador) y otra de tipo vector de posición (posición del jugador en el eje XYZ).

Guardar es un *Blueprint* en el que solo se encuentran las variables que se van meterse en el archivo de guardado, su única función es la de ser el contenedor de dichas variables. Los diagramas que se mostrarán a continuación se encuentran en todos los menús donde haya botón de cargar y guardar.

5.7.1. Guardar partida

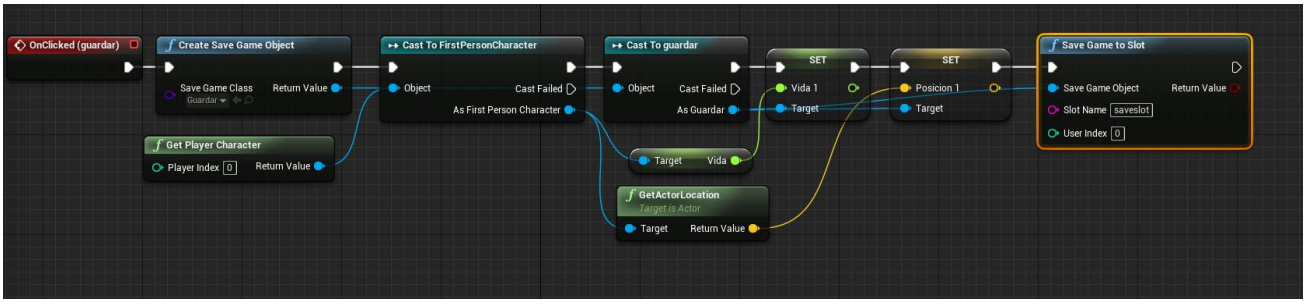


Ilustración 118: Guardar partida

La primera operación que hay que realizar es crear un objeto de guardado (nodo *Create Save Game Object*) con las variables del *Blueprint* Guardar (en este caso guardamos la vida como *float* y la posición como vector de posición). Mediante los dos nodos SET (set vida 1 y set posición 1) actualizamos las variables del objeto de guardado con la vida y posición actuales del jugador. Finalmente se crea un fichero de guardado a partir del objeto de guardado con nombre *saveslot* (nodo *Save Game To Slot*)

5.7.2. Cargar partida

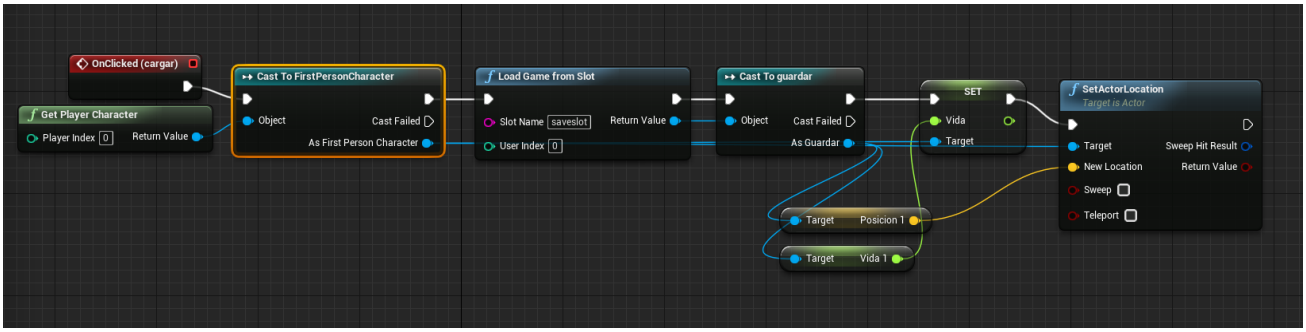


Ilustración 119: Cargar partida

En este diagrama se carga el archivo de guardado que generamos en el punto anterior. Con el nodo *SET* se asigna al jugador el valor de la variable vida 1 del archivo de guardado y con el nodo *SetActorLocation* posicionamos al jugador usando el vector de posición guardado en la variable posición 1.

5.8. MENÚS

En este apartado se explicará la creación mediante Blueprints de los menús del juego

5.8.1. Menú de inicio

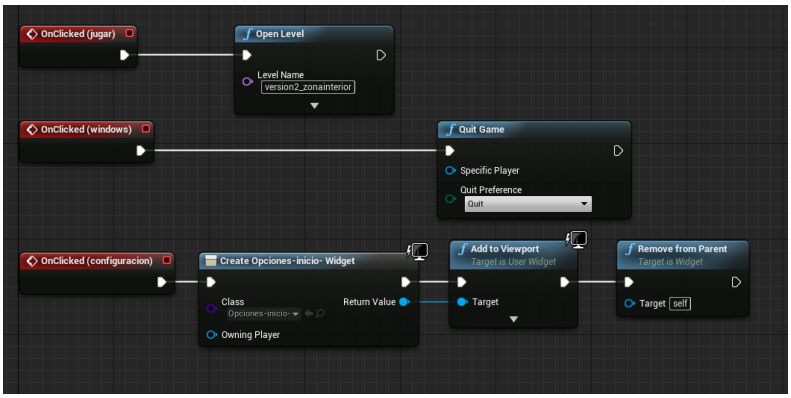


Ilustración 120: Menú inicio

Encontramos eventos de botón (nodos rojos) para cada botón creado en la interfaz del menú.

- **Botón Jugar**: se abre el juego en el primer nivel de juego mediante el uso del nodo *Open Level*.
- **Botón Windows**: cierra el juego y nos devuelve al escritorio de Windows mediante el uso del nodo *Quit Game*.
- **Botón Configuración**: el nodo *Create widget* carga un nuevo interfaz (en este caso el de configuración) el cual se añade a la pantalla (nodo *Add to Viewport*) y se elimina la interfaz del menú con el nodo *Remove from Parent*.

5.8.2. Menú de configuración

Este menú será igual para los tres mapas, por lo que solo se explicará en una ocasión y se dividirá para exponer mejor la totalidad del Blueprint.

UE4 permite modificar los parámetros gráficos principales de forma sencilla mediante el uso de comandos. En este caso se permite modificar la calidad de las sombras, las texturas, el antialiasing y la resolución en tiempo real de ejecución del juego.

Asociado a cada configuración se ha creado una variable de tipo *String* llamada Sombra_C donde guardaremos el comando necesario para el cambio.

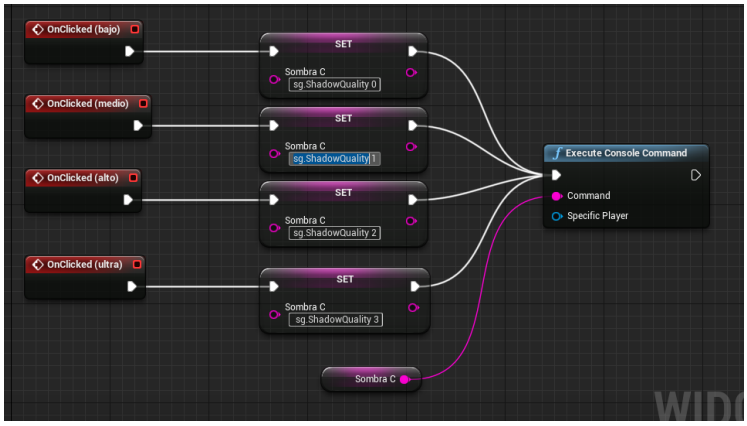


Ilustración 121: Configuración sombras

- **Calidad de las sombras**: El comando para cambiar la calidad de las sombras es *sg.ShadowQuality* [valor numérico entre 0 y 3]
  Para cada botón de la interfaz tenemos un evento de tipo *Onclicked* y cada evento está unido a un nodo SET que asigna a la variable de tipo *String* SombraC el valor asociado al botón. Si seleccionamos el botón "bajo" se asignara el comando *sg.ShadowQuality 0*.

  El nodo que precede al nodo SET es el mismo para los cuatro y se encarga de ejecutar los comandos que recibe de la variable Sobra_C. Recordamos que con anterioridad hemos guardado en esa variable el texto asociado al botón que se haya pulsado.

- **Calidad del Antialiasing**: El comando para cambiar la calidad del *Antialiasing* es *r.PostProcesssAAQuality* [0,2,4,6] y el proceso del diagrama es el mismo que con la calidad de las sombras.

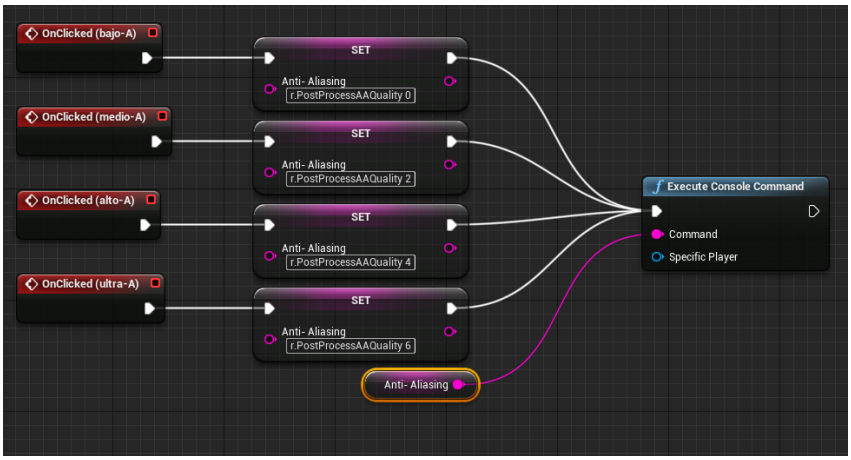


Ilustración 122: Configuración Antialiasng

- **Cambiar la resolución**: El comando para cambiar la resolución es *r.setRes* [resolución deseada] y el proceso del diagrama es el mismo que con la calidad de las sombras con la diferencia de que aquí solo se tienen 2 botones para seleccionar la calidad.

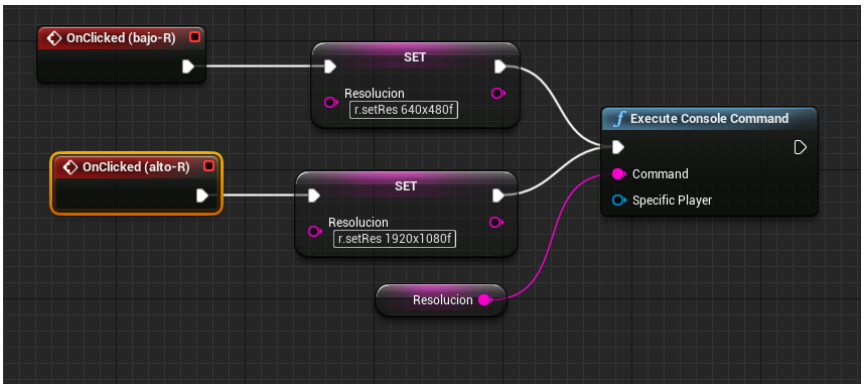


Ilustración 123: Configuración resolución

5.8.3. Menú de muerte del jugador

Este menú aparece cuando el jugador se queda sin vida. Aunque tiene tres opciones solo se mostrarán 2 aquí ya que la opción de cargar partida esta explicada en la página.

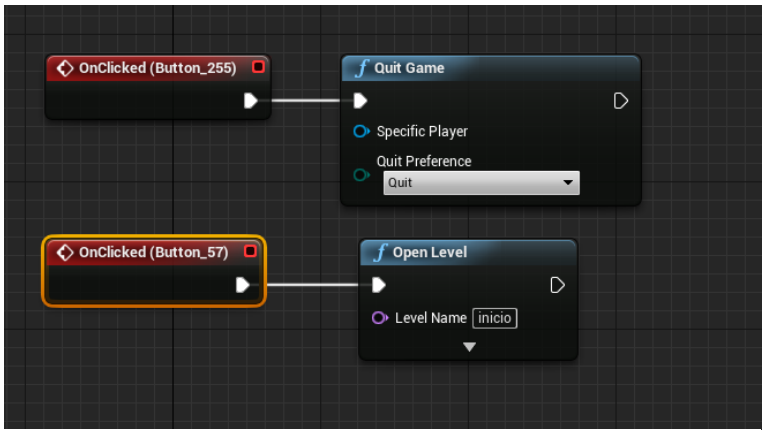


Ilustración 124: Menú muerte jugador

En este menú se encuentran dos botones. Uno de salir del juego (nodo *Quit Game*) y otro de abrir el menú inicial (nodo *Open level*).

5.8.4. Menú durante el juego

A este menú se accede pulsando la tecla "escape" durante la ejecución de la partida. Se encuentran 4 botones más los botones de carga y guardado, aunque estos dos últimos están explicados en la página.

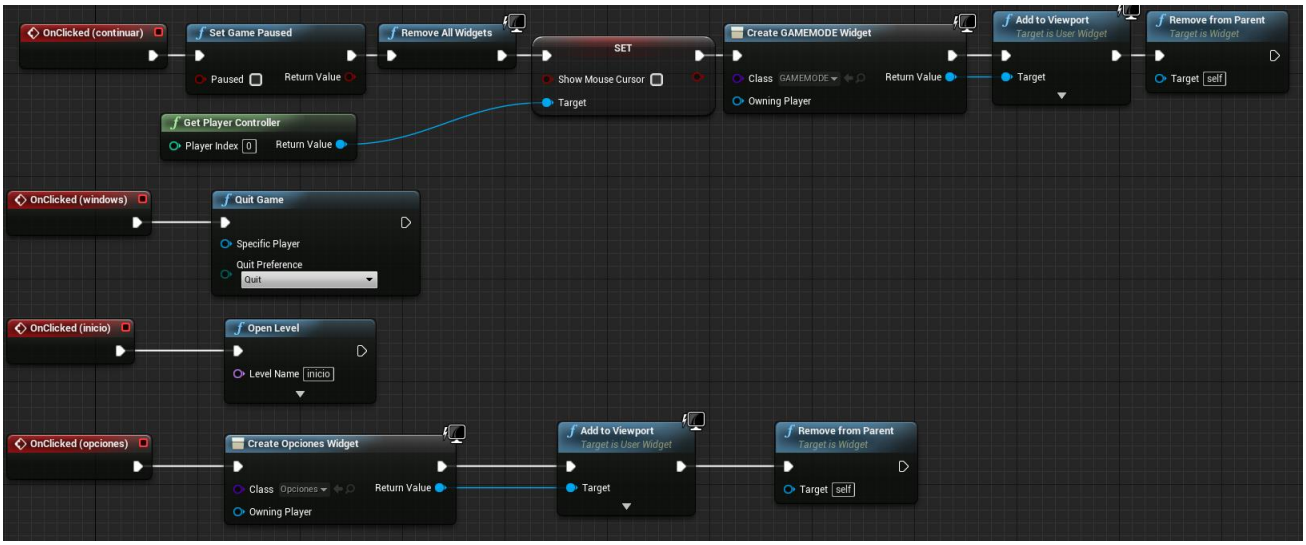


Ilustración 125: Menú Ingame

- El botón continuar comienza quitando el modo pausa (nodo *Set Game Paused* con la condición *paused* deseleccionada), eliminando la interfaz del menú (nodo *Remove All Widgets*), dando el control al jugador (Nodo set recibiendo el nodo Get Player controller) y añadiendo la barra de vida (nodos *Create GAMEMODE Widget* y *Add to Viewport*).
- El botón Windows se une al nodo *Quit Game* para terminar la ejecución del juego y volver al escritorio.

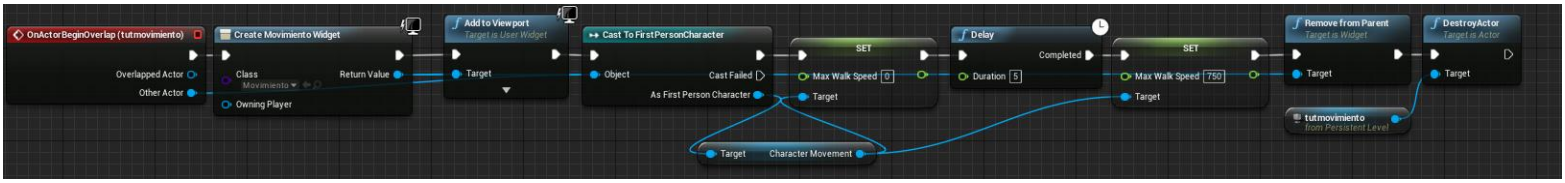
- El botón Inicio devuelve al jugador al menú de inicio

- El botón opciones abre el menú de opciones graficas mediante el nodo *Create Opciones Widget* y *add to Viewport*, además borra el nodo *Remove From Parent* borra de pantalla el menú anterior para que no se superponga.

5.8.5. Tutorial

El tutorial es un blueprint de tipo Widget (como los menús) que se activa al entrar en contacto con un disparador situado a pocos pasos del inicio del juego. Una vez activado muestra la interfaz (nodo *Create Movimiento Widget y Add to Viewport*). El jugador se queda parado para que pueda apreciar mejor la interfaz, esto se logra haciendo *casting* (nodo *Cast To FirstPersonCharacter*) para acceder a la variable Max Walk Speed y actualizar su valor a 0 (Nodo *SET)*, de este modo la velocidad de movimiento del jugador será 0.

Para dar tiempo a leer todos los controles se aplica un retraso (nodo *Delay)* tras el cual el jugador vuelve a recuperar su velocidad estándar (Nodo SET con valor 750*)* y tras eliminar la interfaz del tutorial (Nodo *Remove from Parent*) y eliminar el activador del tutorial (Nodo *DestroyActor* con objetivo el activador) vuelve tener control de sus acciones.



5.9. CREACIÓN DE TERRENOS

Para la creación del terreno de los niveles se ha usado las herramientas propias de UE4.

Una vez seleccionamos el modo *Landscape* se crea un nuevo terreno (opción *New Landscape*)

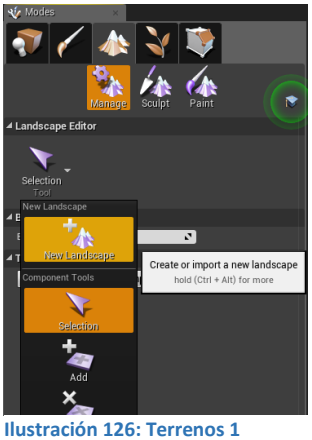


Ilustración 126: Terrenos 1

Una vez se crea el terreno (se crea plano) usamos las herramientas de la pestaña Sculpt para dar la forma deseada.

En este caso estamos aplicando un pincel de presión (zona azul) con diferentes modos de uso (erosionar, subir, bajar, suavizar…). También se ha usado esta técnica para diseñar paredes, aunque hay que destacar que sería más eficiente usar Blender para ello.

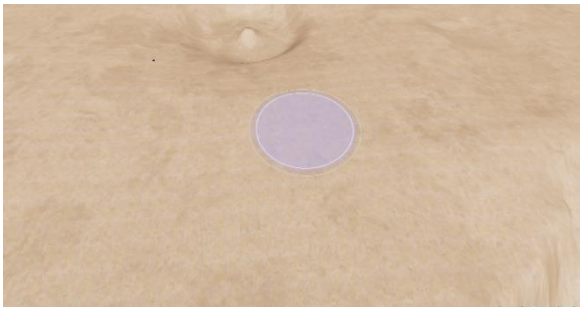


Ilustración 127: terrenos 2

5.10. BLENDER

Blender es un software gratuito de libre distribución creado para diseño gráfico en 3D, animaciones e incluye un pequeño motor para diseñar videojuegos.

Se ha utilizado para diseñar aquellos objetos que no se podían conseguir de forma gratuita en la biblioteca de UE4 (como por ejemplo las antorchas). Aunque hay varios objetos diseñados con Blender, solo vamos a explicar el desarrollo de la antorcha:

- Para su diseño nos hemos basado en las antorchas del videojuego *Skyrim*.

Ilustración 128: Blender 1

- Partimos de un modelo básico, en este caso un cilindro

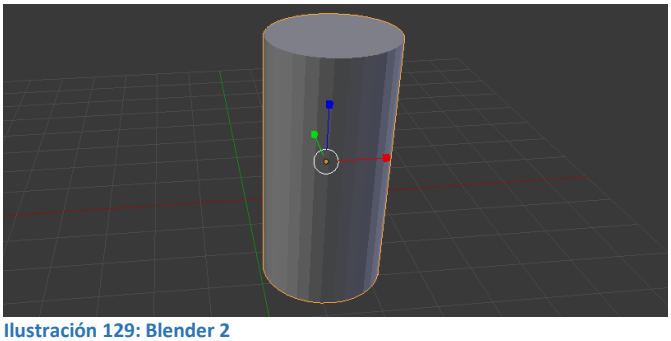


Ilustración 129: Blender 2

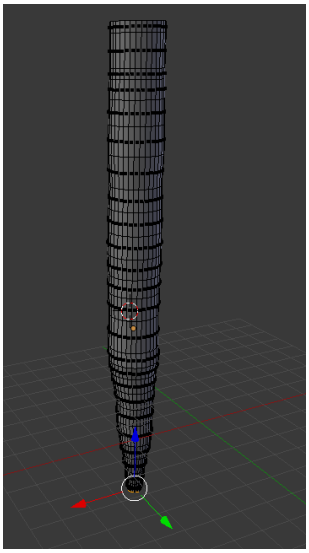


Ilustración 131: Blender 3

Ilustración 132: Blender 4

Ilustración 133: Blender 5

Ilustración 130: Blender 6

- Con la herramienta *Extrude Region* seccionamos el cilindro en el eje Z (altura) haciéndolo así más alto y dando algo de forma (ilustración 124).
- Mejoramos la superficie usando la herramienta *Randomize*. De este modo la superficie se deformará de forma aleatoria y se asemejará un poco más a la superficie de la madera.
- Creamos las abracaderas de metal usando el elemento Toro. Se deforman con la herramienta *Randomize* para mejorar su apariencia (ilustración 123).
- Aplicamos una textura simple del color de la madera al palo (ilustración 121).
- Finalmente añadimos la zona de fuego en la parte superior usando un cubo (le aplicamos una subdivisión en los vértices para redondearlo un poco) y dándole una textura de color rojo (ilustración 122).
- Exportamos el objeto y lo importamos en UE4 4 para añadirle un efecto de partículas y de luz siendo el resultado final el siguiente:

Ilustración 134: Blender 7

5.11. CREACIÓN DEL CIELO CON SPACE SCAPE Y AUTODESK

Space scape es un software de libre distribución con el que podemos generar imágenes de nebulosas de forma aleatoria. Una vez obtengamos la imagen tendremos que usarla para texturizar una esfera usando la herramienta Autodesk. Los pasos son los siguientes:

1. Crear la imagen con *Space Scape*. Este programa nos ofrece una serie de variables (tamaño de estrellas, tamaño de nebulosas, colores aleatorios, etc.) para crear una imagen simulada del espacio.

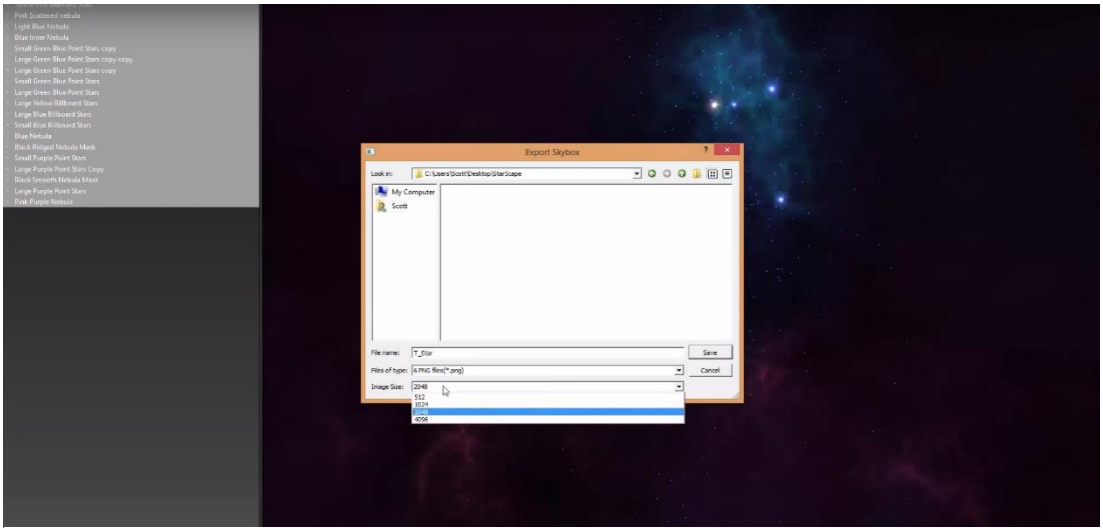


Ilustración 135: Space Scape 1

2. Una vez creada la imagen procedemos a guardarla en 6 trozos (el primer paso para texturizar una esfera es crear un cubo texturizados) y con el tamaño más grande posible para evitar ver la imagen pixelada.

   Las 6 imágenes tienen que colocarse de una forma específica en el cubo, por lo que para evitar problemas las renombramos poniendo la posición en la que tendrá que estar en el cubo.

Ilustración 136: Space Scape 2

3. Aplicar textura a un cubo en *Autodesk*: para ello se crea un cubo y dentro del editor de materiales se hace *click* en el botón *Standard* y se cambia por la opción *Multi/sub-ojbect*.

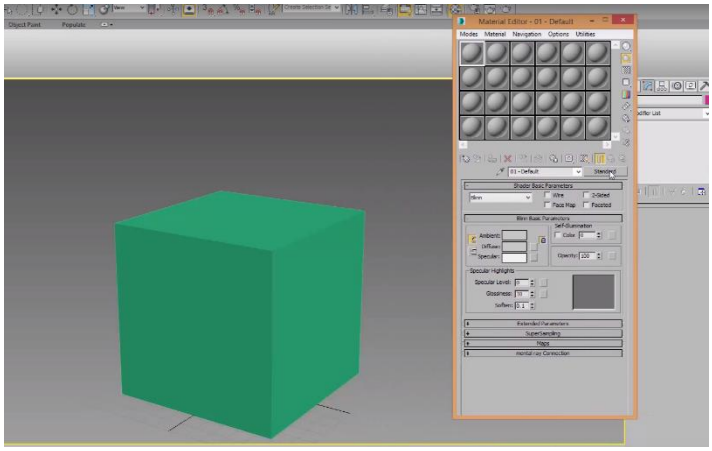


Ilustración 137: Autodesk 1

Seleccionamos la opción *Set Number* of *materials* e indicamos que solo habrá 6 materiales diferentes.

4. A cada material le asignamos una cara (siempre siguiente el orden indicado en la imagen).

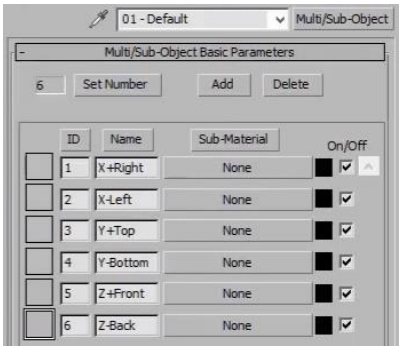


Ilustración 138: Autodesk 2

5. Asignamos a cada material la imagen correspondiente que anteriormente generamos y guardamos con *Space-Scape*.

   Ejemplo: En la ID 1 con nombre X + *Right* pulsamos el botón "*none*" y seleccionamos la imagen guardad como T_start_right1.png siendo el resultado el siguiente cubo

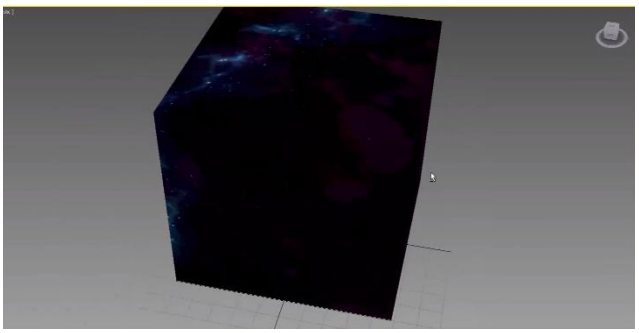


Ilustración 139: Autodesk 3

6. Una vez tenemos el cubo texturizado tenemos que transformarlo en esfera. Para ello seleccionamos el cubo y dentro y nos metemos en el menú de modificadores.

   El siguiente paso es elegir el modificador *TurboSmooth.* Este modificador nos transformara el cubo en un icosaedro. Para crear una esfera habrá que añadir iteraciones al modificador dentro del mene que nos sale al seleccionarlo.

Ilustración 141: Autodesk 4

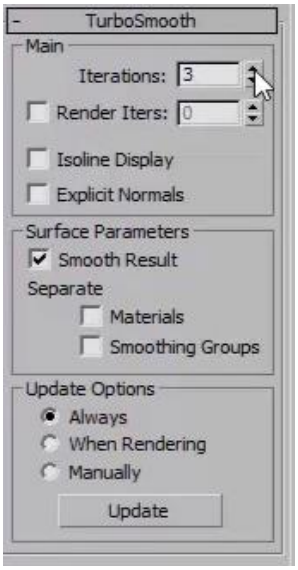


Ilustración 140: Autodesk 5

7. Una vez creada la esfera tenemos que exportarla desde *Autodesk* usando la extensión. FBX

8. Dentro de UE4 importamos el archivo teniendo en cuenta que hay que deseleccionar la opción *Auto Generate Collision* para evitar problemas tanto con la iluminación como con errores con otros elementos del juego.

El resultado de la importación serán 6 materiales, una esfera texturizada y las 6 imágenes.

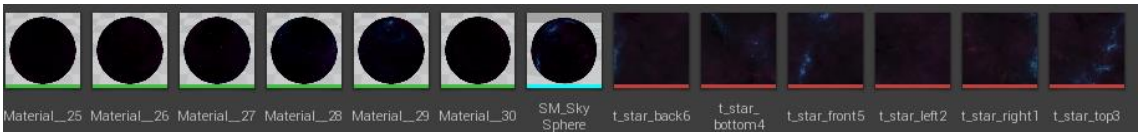


Ilustración 142: cielo UE 1

9. Abrimos la esfera texturizada (SM_SkySphere) para deseleccionar las colisiones: *Collision*/*remove Collision*

10. Seleccionamos las 6 imágenes (en la imagen de arriba tienen una barra roja).

    Dentro de la ventana que se abre cambiamos el valor de *Texture Group* a *Skybox*. De esta forma hacemos saber a UE4 que nuestra esfera está formada por imágenes que van a crear el cielo.

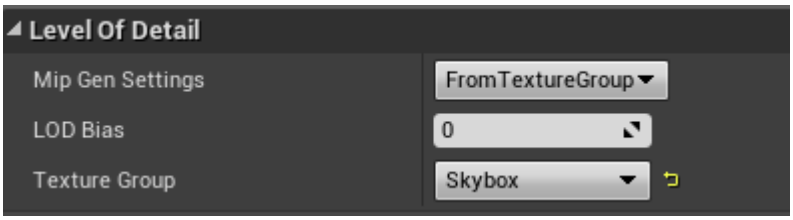


Ilustración 143: cielo UE 2

11. La última parte es añadir a cada una de las texturas la unión entre *Emissive Color* y *Texture* simple. En la imagen inferior podemos ver la unión con una línea blanca. Esto va a hacer que las imágenes brillen sin necesidad de que un foco de luz las ilumine.

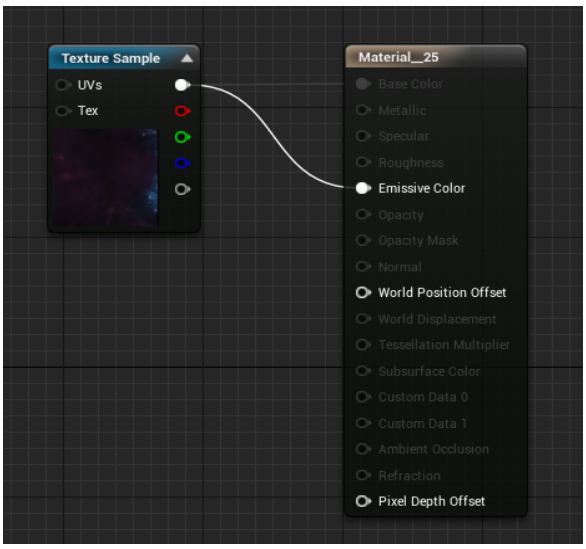


Ilustración 144: cielo UE 3

12. Finalmente añadimos la esfera en nuestro mundo y ajustamos su tamaño final.

Ilustración 145: Cielo UE 4

5.12. AUDIO

UE4 cuenta con un motor de audio que permite crear diferentes tipos de efectos. En esta sección hablaremos de los audios más importantes de este TFG y de su creación.

Se ha usado audio creado por mí mismo e importado de bibliotecas de libre acceso.

5.12.1. Música inicio

Ha sido creada de forma manual con un piano y editada con Audacity. Se inicia al cargar el menú de inicio dentro del Blueprint usado para crear el Menú y por medio del siguiente nodo

- **Sound**: elegimos el audio dentro de todos los importados
- **Volume multiplier:** aplicamos un multiplicador en la potencia del audio.
- **Start time**: indicamos si queremos que suene al instante o después de un tiempo.

Ilustración 146: Musica inicio

5.12.2. Música ambiente

Ha sido creada con la página mynoise.net, con los recursos gratuitos de UE4 y composición propia.

Pistas que comienzan al iniciarse el nivel: la mayoría de la banda sonora del videojuego comienza tras empezar un nivel, por lo que sus Blueprints son idénticos. En este *Blueprint* activamos el audio (nodo *activate*) y en caso de que el audio finalice lo volvemos a activar (nodo *OnAudioFinished*).

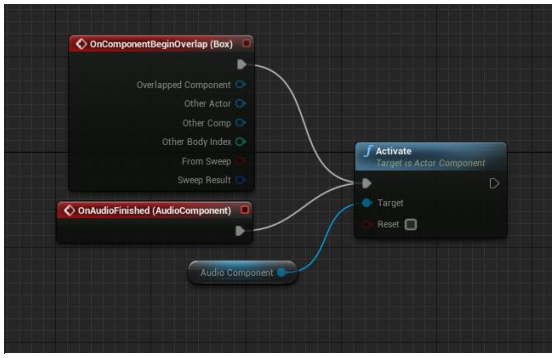


Ilustración 147: Música ambiente 1

5.12.3. Efectos de audio

Una parte importante de la inmersión en un videojuego es el audio del mismo. En esta sección se verá una muestra de cómo se asocian audios a eventos del juego sacados en su mayoría de la biblioteca gratuita de audio *Sonnis Free Pack.*

- Audios sin modificadores: tienen como origen el propio jugador, por lo que carecen de localización[18] sonora y es la clase más simple de audio en UE4.

  Ejemplo del altar mágico:

  Activar un altar supone abrir un nuevo camino, por ello se añadió un audio similar al de un muro rompiéndose mediante el uso del nodo *Play Sound 2D* dentro de la secuencia de acciones que desencadena activar el altar.

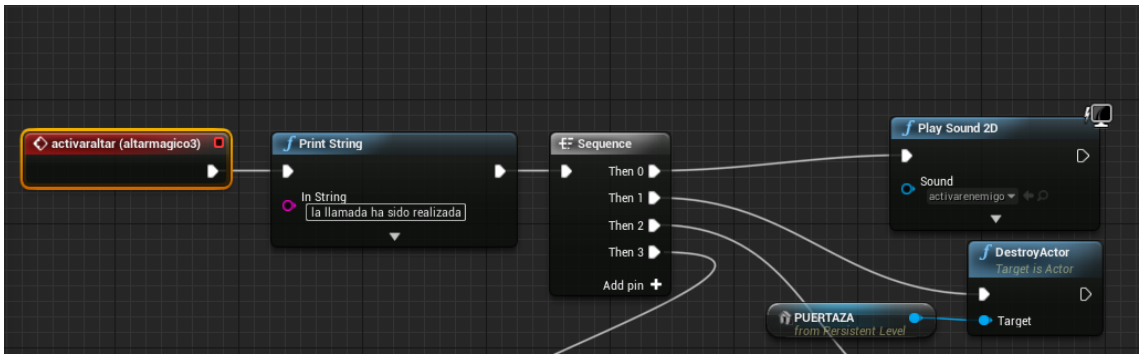


Ilustración 149: audio altar mágico

Otros objetos que tienen asociados este tipo de audios son: llaves, botiquines, minas, audio inicial del juego y el sonido ambiente.

- Audio con modificadores: tienen como origen un elemento externo al personaje, por lo que se le pueden aplicar características como la atenuación[18] o el sonido biaural[19] entre otros.

  Ejemplo de audio de fuego:

  Este tipo de audio se encuentra dentro de los objetos predefinidos por UE4. Se puede posicionar en cualquier punto del nivel, desde el cual emite el audio. Sus variables más importantes son:

  - Sound: archivo de audio que va a reproducirse.
  - Volume Multiplier: multiplicador del volumen, un valor por encima de uno lo aumenta y por debajo lo disminuye.
  - Override Attenuation: activa las opciones de atenuación.
  - Distance Algorithm: elección de algoritmo que se usara para la atenuación. En este caso se ha elegido la función logarítmica, pero se pueden elegir atenuaciones constantes o inversas del logaritmo.
  - Falloff Distance: radio a partir del cual la atenuación es máxima y no se escucha el audio.

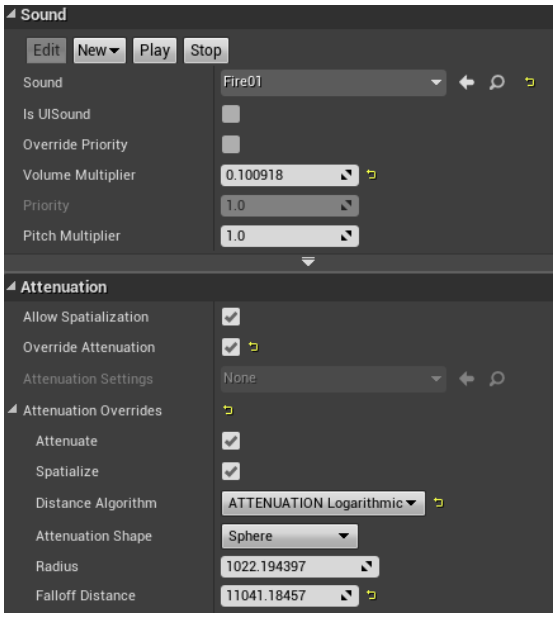


Ilustración 150: variables del audio

6. PUESTA EN MARCHA

6.1. INSTALANDO UE4

El primer paso para su descarga es registrarse en la página[18] de *Epic Games* ya que es requisito indispensable para desarrollar proyectos en UE4.

Una vez estamos registrados podemos acceder al fichero de instalación desde la web[18]. Cuando finaliza la instalación encontramos el menú donde podemos instalar la versión que queramos de UE4 (en este caso la 4.12.3), organizar nuestros proyectos o revisar la tienda.

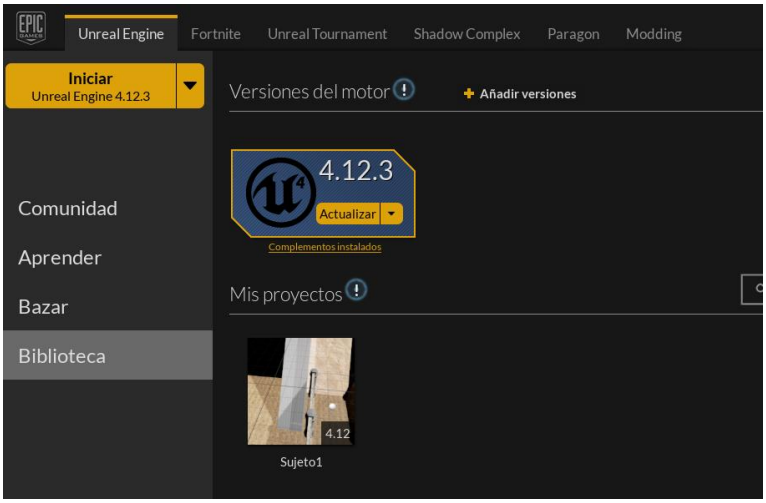


Ilustración 151: puesta en marcha 1

Una vez instalado el motor gráfico tenemos la oportunidad de crear un nuevo proyecto a partir del siguiente menú:

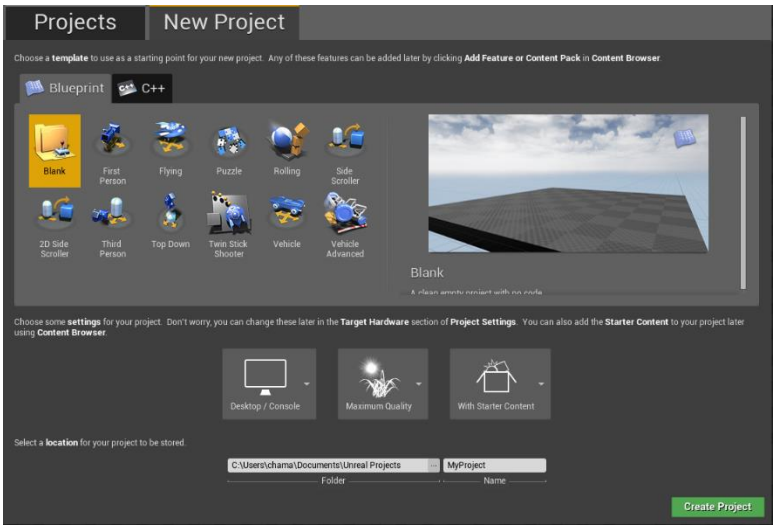


Tabla 17: puesta en marcha 2

Nos da la opción de elegir un proyecto con Blueprints o con C++ además de elegir el estilo de juego que queramos. Estos estilos de juego vienen con configuraciones predeterminadas. Si elegimos Vehicle, UE4 nos ofrecerá un vehículo simple con las funciones básicas programadas.

Si por el contrario elegimos, como ha sido el caso, *first person* tendremos configurado un actor en primera persona.

En todos los ejemplos la clase principal se puede modificar para configurarlo de la forma que el videojuego requiera. Se elige para que dispositivo vamos a desarrollar el juego (*Desktop/console*), la calidad y si queremos comenzar un con pack de contenido inicial (pack que contienen objetos y texturas para los primeros pasos con el software.).

6.2. COMPILACIÓN

Cuando el desarrollo ha finalizado se procede a la compilación final del juego. En el mundo del desarrollo esta última compilación se la conoce como *gold copy* ya que es la versión que se usará como original para las copias que se distribuirán en físico y en formato digital.

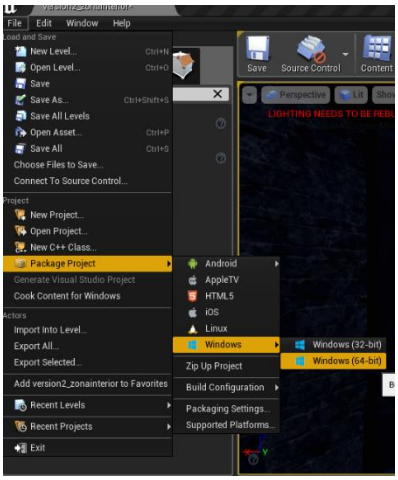


Ilustración 152: compilación 1

Para ello seguimos la ruta de la imagen superior (file → P*ackage Project*).

Una vez estamos en el submenú de *Package Project* nos pregunta qué tipo de compilación queremos. UE4 permite crear proyectos para Android, Apple, Html5, iOS, Linux y Windows 10 (aunque existen *plugins* para consolas). En este caso usaremos la compilación para Windows 10 y la versión 64 bits.

El resultado final de la compilación es la siguiente carpeta:

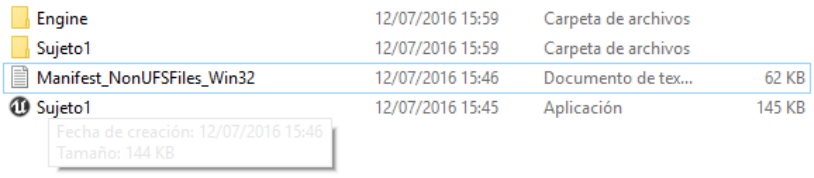


Ilustración 153: Compilación 2

1. Engine: contiene toda la información del núcleo del motor gráfico. Estos ficheros serán prácticamente similares en cualquier videojuego producido con UE 4.

2. Sujeto1: contiene todo el contenido creado por nosotros.

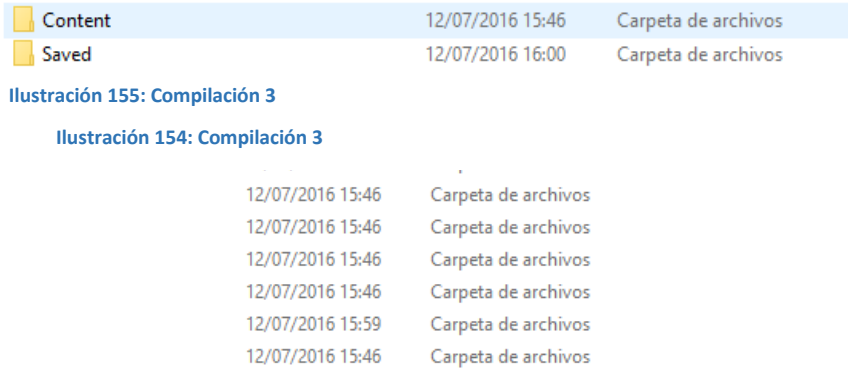


Ilustración 155: Compilación 3

Ilustración 154: Compilación 3

Dentro de la carpeta Content estará situado todo el contenido del videojuego (estructuras, *Blueprints*, esqueletos, objetos, audio …) y en la carpeta *Saved* estará el fichero de guardado. Si tuviéramos varios ficheros de guardado se guardarían también en la misma carpeta.

El tiempo de compilación de las dos versiones es de:

	Versión 64 bits	**Versión 32 bits**
Tiempo de compilación	16 minutos	13 minutos

Tabla 18: tiempo de compilación

6.3. PRUEBAS

En esta sección se realizarán pruebas para comprobar el correcto funcionamiento del videojuego. La metodología incremental permite hacer pruebas en cada iteración, por lo que se llega a la fase final del desarrollo con gran parte de la fase de pruebas realizada.

Las pruebas se han realizado con un ordenador con características inferiores para validar la optimización del videojuego en equipos con menos potencia grafica que el usado para su desarrollo.

- Intel Core I5 – 2.4 Ghz
- 8 GB de memoria RAM
- Disco SSD
- Nvidia 850M 2GB.

El videojuego permite tener una tasa de FPS (fotogramas por segundo) de 60 estables y asegura este número gracias a la capacidad para cambiar la calidad gráfica, por lo que se considera que presenta un rendimiento grafico óptimo.

El juego es capaz de abrir todas interfaces de usuario. Aquí se ha detectado un error del que no se la solución, ya que al abrir un menú hay que hacer click con el botón derecho del ratón para poder seleccionar opciones dentro de la interfaz.

El videojuego guarda de forma correcta el avance del usuario. Una vez se guarda la partida, se vuelve a recuperar de forma correcta esta, dejando al jugador con la misma vida y en la misma posición.

6.4. MANUAL DE INSTALACIÓN

En este capítulo se expondrá como ejecutar el videojuego.

Los requisitos mínimos para el correcto funcionamiento de este videojuego son:

- Bibliotecas DirectX 10, 11 o 12 instaladas o actualizadas.
- Sistema Operativo Windows 7, 8.1 o 10.
- Drivers gráficos actualizados (Nvidia, AMD o Intel).
- 3 Gb disponibles en el disco duro.

Una vez comprobados estos requisitos, se accede a la carpeta del videojuego y se ejecuta el archivo Sujeto1, mostrando el menú principal.

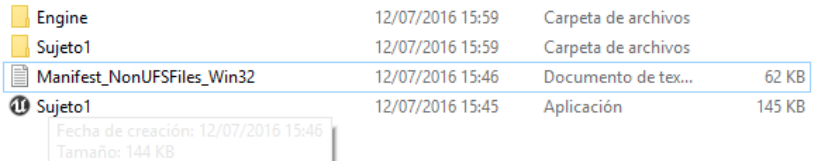


Ilustración 156: Manual 1

7. CONCLUSIÓN Y MEJORAS

7.1. CONCLUSIÓN

Cuando empecé este proyecto pensaba que no existía otro tipo de programación de videojuegos que no pasara por escribir código tal y como lo conocemos. Aunque ya sabía de la existencia de los motores gráficos y su funcionalidad, no conocía el scripting gráfico.

Esta forma de programación, mediante el uso de tarjetas o nodos, suponía un cambio en la forma en la que yo entendía la programación, y tras leer la documentación de UE4 me di cuenta de la potencia que suponía el uso de esta tecnología.

Hasta hace relativamente poco la programación del videojuego se basaba en lenguajes de programación como C++ o C# y en estructuras de programación complejas, que hacían que el desarrollo de videojuegos dependiese de grandes grupos de trabajadores. En la actualidad hay muchos estudios pequeños, también conocidos como estudios indies, que desarrollan videojuegos con menos presupuesto del que podrían tener grandes empresas pero que remueven el mercado con sus lanzamientos.

La programación visual (o Blueprints) que ofrece UE4 es una tecnología muy madura, y cuanto más investigaba más me asombraba de las cosas que permitía hacer, ya que al principio pensaba que al no ser código tendría limitaciones técnicas. Sin embargo, cuando comencé a programar y a crear las ideas que tenía en mente me di cuenta de que no había problema en desarrollar todo el proyecto con programación visual. Por otra parte, descubrí que muchos proyectos con presupuestos millonarios estaban hechos solo con esta tecnología, relegando el código en C++ para aquellos problemas específicos que los Blueprints no pudieran solucionar.

Si ya en la época de los noventa la llegada de los grandes motores gráficos produjo una revolución, la facilidad para acceder gratuitamente a ellos, sobre todo los más importantes, ha hecho más fácil la creación de videojuegos a pequeña escala.

Creo que los videojuegos son una parte muy importante dentro de la informática, ya que facilitan la creación y el desarrollo de nuevas tecnologías, como la realidad virtual. Vivimos una revolución cultural marcada por el avance imparable de estos y empeñada en unir la vida cotidiana el entretenimiento.

Desde un punto de vista más persona, este TFG me ha abierto las puertas del diseño de videojuegos. Mucha gente no entiende el trabajo que hay por detrás de un videojuego y que la mayor dificultad que presentan es el tiempo. También me ha ayudado a contactar con gente relacionada con el diseño de videojuegos, interesándose incluso en mi creación y ayudándome siempre que lo he necesitado.

Es por ello que la última estrofa de agradecimiento va para dos personas:

El creador de *mynoise.com* por permitirme usar sus generadores de audio de forma gratuita.

Josemaría Egea por guiarme y no tener problemas por contactar conmigo por Linkedin y mostrarme su TFG de desarrollo de videojuegos. Al final tu trabajo fue útil a otro.

Seguiré especializándome en esta área ya que solo he raspado la superficie de todo lo que UE4 puede ofrecer.

7.2. MEJORAS

Una vez terminado el desarrollo y haber entendido con más profundidad UE4 estaría en disposición de realizar las siguientes mejoras para versiones futuras:

- Mejorar el sistema de guardado: el juego solo guarda la posición del jugador y su vida, pero en una implementación optima de un sistema de guardado habría que mantener otros valores como enemigos ya activados, llaves recogidas o minas destruidas.

- Implementar enemigos más complejos: los enemigos actuales distan mucho de tener inteligencia artificial y tan solo buscan colisionar con el jugador. La idea de añadir enemigos más complejos con diferentes mecánicas para superarlos fue algo no pudo añadirse en la versión actual, pero que con los conocimientos actuales si podría.

- Solucionar problema con el control: actualmente hay un error que se produce al dar el control al jugador dentro de cualquier menú. Este error hace que haya que dar doble click para poder pulsar cualquier tecla.

- Perfeccionar las texturas: me gustaría poder en un futuro usar mejores texturas, las usadas en este proyecto son reutilizadas de otros proyectos.

8. BIBLIOGRAFÍA

En esta sección está listada toda la bibliografía consultada para realizar el proyecto.

- Documentación oficial de UE4 [última consulta 12/06/2016] - https://docs.unrealengine.com/latest/INT/
- Desarrllo de un videojuego con UE4: Universidad Politécnica Superior de Alicante [última consulta 10/07/2016] - https://rua.ua.es/dspace/bitstream/10045/49409/1/Desarrollo_de_un_videojuego_con_Unreal_Engine_4_EGEA_CANALES_JOSE_MARIA.pdf
- Project NORS A Multiplayer Online Battle Arena Game Implemented in Unreal Engine 4 [última consulta 20/06/2016] - https://brage.bibsys.no/xmlui/bitstream/id/353813/JNess_AOlsen_MRoedland_CASand_2015.pdf
- GDD?! Game Design Document Examples: características generales de un GDD [última consulta 07/06/2016] - http://seriousgamesnet.eu/assets/view/238
- Documentación en Videojuegos: Documento de diseño (GDD): GDD [última consulta 05/06/2016] - https://eldocumentalistaudiovisual.com/2015/02/06/documentacion-en-videojuegos-documento-de-diseno-gdd/
- Como crear un documento de diseño de videojuegos (GDD) [última consulta 06/062016] - http://www.hagamosvideojuegos.com/2015/06/como-crear-un-documento-de-diseno-de.html
- Kitatus book: UE4 & Blueprints [última consulta 11/06/2016] - http://kitatus.co.uk/
- Autodesk: comunidad online para resolución de problemas [última consulta 14/04/2016] - http://www.autodesk.com/community?_ga=1.33935713.1330856840.1468254079
- ¿Cuánto gana un desarrollador de videojuegos? [última consulta 10/07/2016] - https://www.niubie.com/2011/01/cuanto-gana-un-desarrollador-de-videojuegos/
- Canal oficial en YouTube de Unreal Engine: recursos gratuitos para desarrolladores [última consulta 13/07/2016] - https://www.youtube.com/user/UnrealDevelopmentKit
- The online computer and Videogame Industry: documento de investigación sobre la industria de los videojuegos [última consulta 01/05/2016]- https://www.oecd.org/sti/ieconomy/34884414.pdf.
- Tesladev YouTube: videos de desarrollo de videojuegos con Blueprints [última consulta 05/06/2016]- https://www.youtube.com/channel/UC3QBWg9pMnaFF-q0qjXPDEg.
- Space Scape: configuración y uso del programa [última consulta 14/04/2016] - https://www.youtube.com/channel/UC3QBWg9pMnaFF-q0qjXPDEg.
- Canal Blender: tutoriales en español del uso básico de Blender [última consulta 29/06/2016] - https://www.youtube.com/channel/UCDkwKVawKoC8lpKgfMvHMpw
- Página oficial de Blender: información sobre las herramientas básicas del programa [última consulta 12/03/2016] - https://www.blender.org/support/tutorials/

- Apuntes de la asignatura de Multimedia impartida por el profesor Miguel Angel. Universidad de Valladolid escuela de informática de Segovia [última consulta 13/03/2016]

8.1 Referencias

[1], [2]: Estudio sobre la industria de los videojuegos:

https://newzoo.com/insights/articles/global-games-market-will-grow-9-4-to-91-5bn-in-2015/

[3]: Early Access, su utilidad:

https://hipertextual.com/archivo/2013/03/steam-early-access/

[4]: ¿Cuánto se gana creando videojuegos?

https://www.niubie.com/2011/01/cuanto-gana-un-desarrollador-de-videojuegos/

[5]: ¿Cuánto cuesta desarrollar un videojuego?

http://yachtclubgames.com/2014/08/sales-one-month/

[6]: Idtech y los motores gráficos

https://en.wikipedia.org/wiki/Id_Tech

[7]: documentación Unreal Engine 4:

https://docs.unrealengine.com/latest/INT/

[8]: Tienda Unreal Engine 4:

https://www.unrealengine.com/marketplace

[9] Canal Oficial en Youtube de Epic Games:

https://www.youtube.com/user/UnrealDevelopmentKit

[10] GDD videojuego GTA:

http://classes.dma.ucla.edu/Winter12/157A/doc2/gta_gdd.pdf

[11] GDD videojuego Larry

http://allowe.com/gamedesign/Larry6%20Design.pdf

[12] GDD videojuego Rise of the dead.

http://www.gamefiction.com/docs/Rise_Prospectus_Doc.pdf

[13] Genero aventura en tres dimensiones

https://en.wikipedia.org/wiki/Adventure_game

[14] Sistema PEGI europeo

http://www.pegi.info/es/

[15]Diagrama de flujo de juego

http://www.gamasutra.com/blogs/EmanuelMontero/20091106/85856/Story_Flowchart_Diagrams.php

[16] Videojuego LifeLess Planet:

http://es.ign.com/lifeless-planet-pc/76946/review/lifeless-planet-analisis-para-pc

[17] HUD ¿Qué es?:

https://es.wikipedia.org/wiki/HUD_(inform%C3%A1tica)

[18] Pagina de descarga de UE4

https://www.unrealengine.com/dashboard

[19] Atenuación de sonidos en la transmisión

https://es.wikipedia.org/wiki/Atenuaci%C3%B3n

[20] Localización de sonidos 3D

http://www.theverge.com/2015/2/12/8021733/3d-audio-3dio-binaural-immersive-vr-sound-times-square-new-york

9. GLOSARIO

- Blueprint: programación grafica de scripting en la que se basa UE4.
- Real Engine: motor gráfico que se caracteriza por su semejanza grafica a la realidad.
- HUD: estructura grafica que representa datos importantes del personaje en la pantalla de juego.
- IU: interfaz de usuario, dentro de este término se encuentran los menús, pantallas de carga y similares.
- FPS: fotogramas por segundo a los que es capaz de funcionar el juego.
- Gold copy: copia final de un videojuego en formato DVD o blu-ray usada por los estudios para la producción en masa.
- Antialiasing: suavizado de bordes en las imágenes que conforman un videojuego.
- Power-ups: mejoras para el jugador que le permiten ir más rápido o saltar más entre otras.