UNREAL ENGINE 4

Shooter – Prison Escape

Miguel Cabezas Puerto
70911497-J

Tabla de contenido

Tabla de ilustraciones

Unreal Engine

Unreal Engine es un motor de desarrollo de juegos creado por la compañía Epic Games[4] con licencia propietario[5] utilizado por primera vez en 1998 en el juego FPS Unreal. En un primer momento fue desarrollado para juegos *Shooter* en primera persona, no obstante, pronto ganó éxito en otra gran variedad de géneros como juegos de rol online, videojuegos de sigilo y otros *RPGs* [6]

Codificación y plataformas

Su código está escrito en C++ por lo que tiene un alto grado de portabilidad siendo siendo uno de los motores de videojuegos más ampliamente usados en el mundo. La versión actual es Unreal Engine 4 aunque el mayo de 2020 se reveló que a finales de 2021 se lanzará Unreal Engine 5 compatible con todos los sistemas existentes, así como con la próxima generación de consolas PlayStation 5 y Xbox Series X/S.
Actualmente da soporte a diversas plataformas como Windows, Linux, MAC OS en el ámbito de los ordenadores, Dreamcast, Xbox y PlayStation en consolas y Android e iOS para la tecnología móvil.

Versiones

- ➔ Unreal Engine 1: integraba renderizado, detección de colisiones, IA, manipulación de archivos de sistema y visibilidad. En él se desarrollaron juegos como *Unreal* y *Unreal Tournament*
- ➔ Unreal Engine 2: en él se reescribió completamente el motor de renderización y el código del núcleo además de integrar el SDK de Karma physics[7]. Uno de los juegos desarrollados con este motor fue *Unreal Championship 2*
- ➔ Unreal Engine 3: su renderizado soporta técnicas avanzadas como HDRR [8], iluminación por píxel y sombreado dinámico. En él se desarrollaron juegos como *Unreal Tournament y Gears of War 2*
- ➔ Unreal Engine 4: desde marzo de 2015 está disponible de forma gratuita. Sus capacidades incluyen renderizado basado en físicas, contrucción de niveles, animaciones, efectos visuales, físicas, juego en red y gestión de *assets.* Entre los juegos desarrollados destacan *City of Titans[9], Kingdom Hearts III [10]* o *Bacon Man [11].*

IDE y primeros pasos

Una vez descargado Unreal Engine 4 (UE4) de su página oficial y lanzado se abre inicialmente en *Project Browser.* En *la* pestaña de *projects* podemos encontrar todos los proyectos en los que se está trabajando mientras que la pestaña de *New Project* es utilizada para crear nuevos proyectos basados en plantillas de diferentes modos de juego existentes.

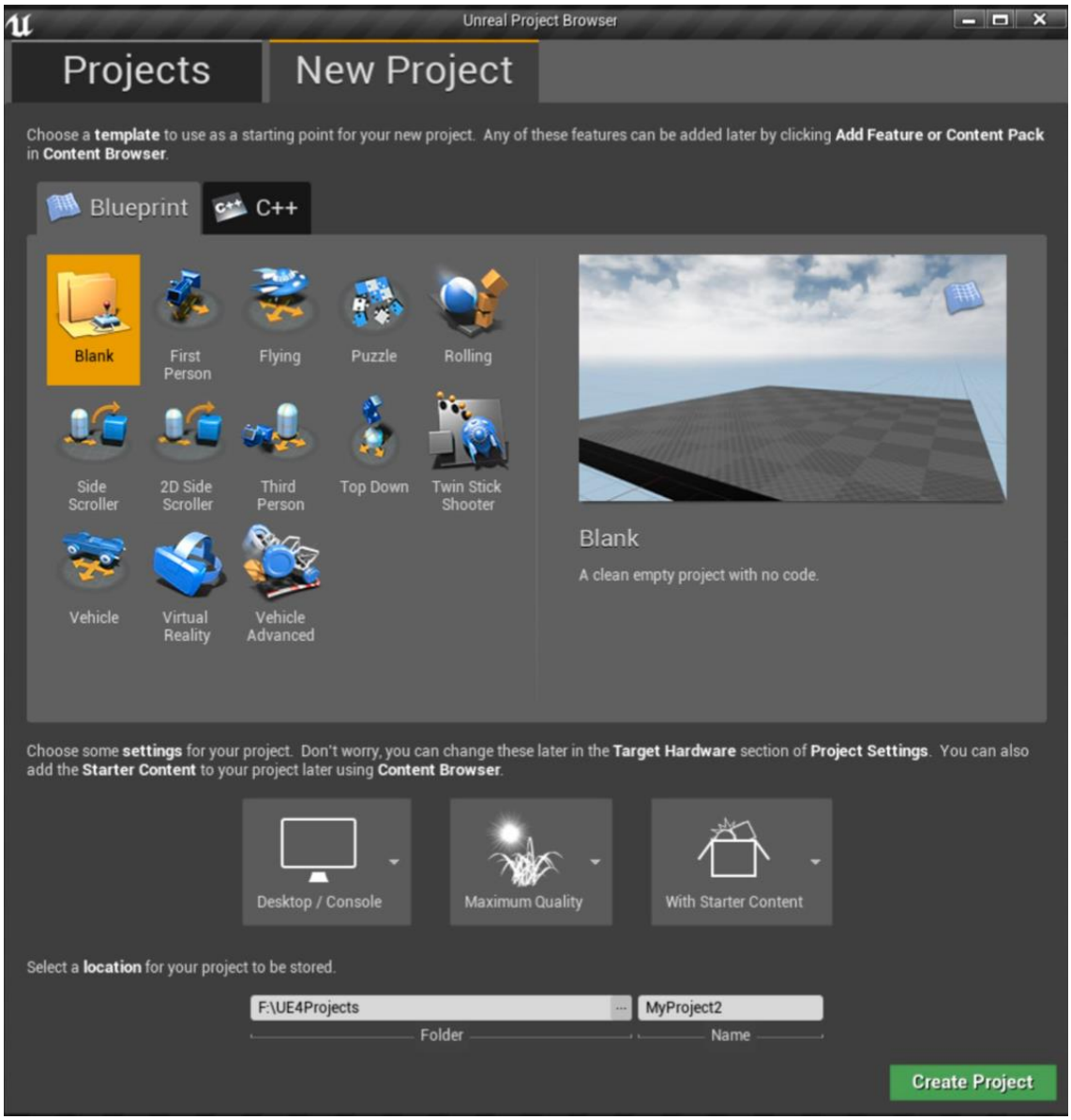

Ilustración 1 Project Browser

Desde la pestaña de *New Project* es posible crear un Proyecto basado en *Blueprints* o en C++. *Blueprint*, el usado para el ejemplo práctico de este trabajo, es un entorno de programación visual basado en nodos desarrollado por UE para permitir tanto a diseñadores como a programadores añadir funcionalidad a los proyectos.

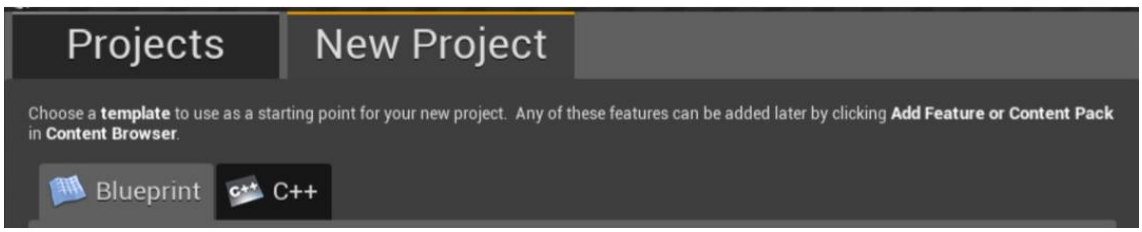

Ilustración 2 Pestaña New Project

Además, se puede partir de diferentes plantillas de juegos denominadas *Game Modes Templates* construidas en torno a Gameplay Framework. Existen plantillas tanto para proyectos basados en Blueprints como para proyectos basados en C++. En el caso práctico, como se detallará en próximos apartados, se ha empleado la plantilla de juego en tercera persona (*Third Person Character)*

Por último, antes de crear el proyecto se pueden cambiar sus ajustes como el hardware de destino del proyecto, la calidad de los gráficos o si el proyecto debe incluir un paquete de assets disponibles en UE4 (*Starter Content*). mediante la pestaña *Project Settings.*

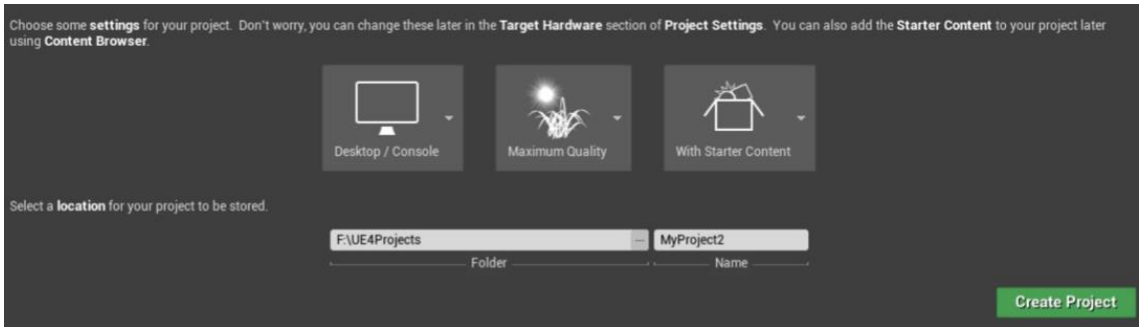

Ilustración 3 Project Settings

Interfaz principal

La interfaz principal de UE4 se conoce como *Level Editor*, esta se utiliza fundamentalmente para la construcción de entornos, niveles, colocar objetos en escenas. En ella podremos encontrar siete grandes partes: *menu bar(1), level editor toolbar(2), modes panel(3), content browser panel(4), viewport panel(5), world outliner panel(6) details panel(7)*

Ilustración 4 Interfaz principal

Menu bar

Barra de menú con los menús habituales *File, Edit, Window, y Help*. *File* contiene operaciones para cargar y guardar proyectos y *Levels, Edit* operaciones clásicas de copiar y pegar así como preferencias del editor (aquí es posible cambiar el idioma) y del proyecto. *Window* aquí es posible ver los distintos *viewports*. *Help* ayuda y enlaces a la documentación.

Toolbar

proporciona un acceso rápido a las opciones y herramientas más comunes como por ejemplo: guardar nivel actual, construir la iluminación para los *Static Actors* en *Build*, cambiar las propiedades del editor, ejecutar el nivel actual con *Play*

Modes Panel

Muestra los distintos modos de edición. Permite acceder a diferentes interfaces para trabajar con distintos tipos de *Actors(assets* colocados en el nivel o escena) con operaciones tales como colocar nuevos *actors* en un nivel o escena, crear y editar paisajes, añadir follaje a un nivel, crear geometrías…

Content Browser

Área principal donde crear importar, organizar, ver y modificar contenidos dentro de Unreal Engine 4. En él es posible organizar las carpetas y realizar otras operaciones sobre *assets* como renombrarlos, moverlos, copiarlos y ver sus referencias.

ViewPorts

Ventana a los mundos que serán creados en UE4. Es posible navegar por ellos como se haría en el propio juego o pueden ser utilizados de forma más esquemática como si de planos de arquitectura se tratase. Es posible cambiar a diferentes vistas del *Viewport* de forma que se muestre sólo la información necesaria en cada momento. Existen diferentes tipos uno en perspectiva, y siete ortográficos (en dos dimensiones) (desde arriba, desde abajo, izquierda, derecha, de frente y de espaldas)

Además, también se puede elegir entre más de trece modos de visualización dependiendo de la iluminación, reflexiones, colisiones… Como ejemplo se muestran en la siguiente imagen algunos de ellos.

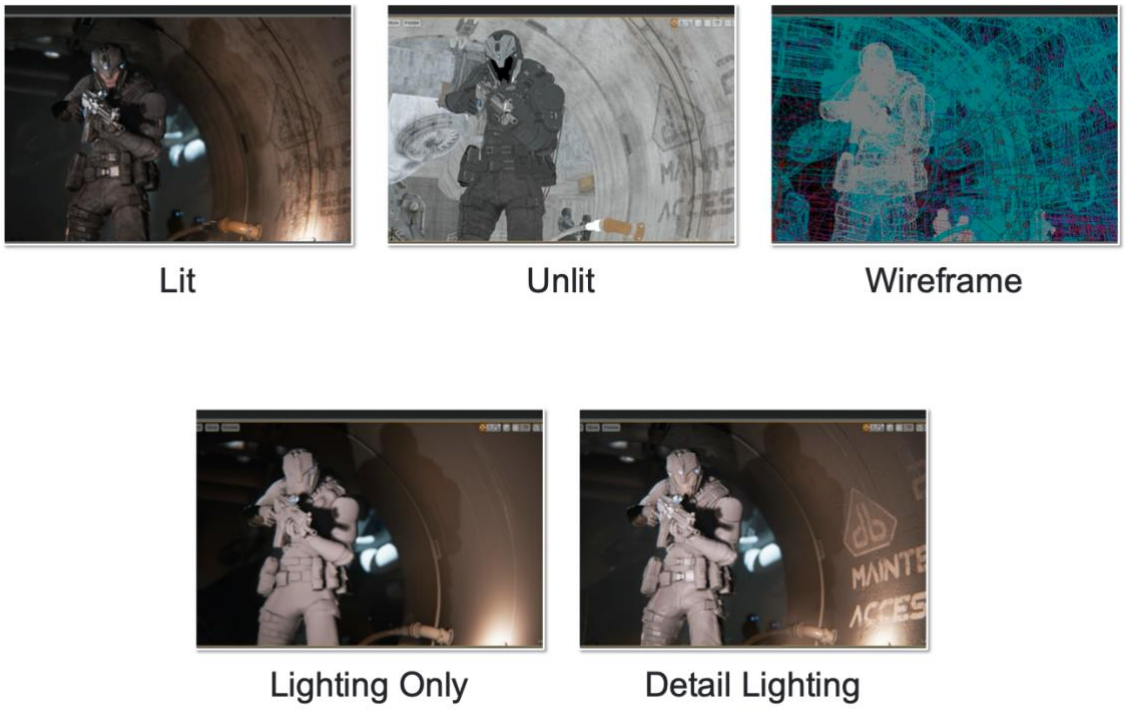

Ilustración 5 View modes

World Outliner

Muestra todos los *Actors* dentro del nivel actual en un árbol jerárquico. Es posible seleccionar un *Actor* haciendo doble clic y hacer foco en el *Viewport* sobre él pulsando la tecla F.

Una vez seleccionado se mostrarán los detalles de dicho *Actor* en el panel *Details*

Details panel

Muestra todas las propiedades editables del *Actor* seleccionado.Algunas propiedades comunes son su nombre, localización, dimensiones, escala, rotación… Dependiendo de la clase del *Actor* se mostrarán diferentes propiedades de igual modo que se mostrarán diferentes propiedades dentro de sus componentes en función del tipo (malla estática, malla de colisión, componentes anidados…)

Navegación en la escena

Por último, se indican las combinaciones de teclas y movimientos de ratón necesarios para moverse en el nivel en el momento de la edición. El resto de los detalles relativos al motor se verán de forma práctica en el ejemplo ya que el objetivo principal de este trabajo es el de realizar un *shooter* similar a Unity pero sobre otro motor, por ello no se profundiza en los aspectos teóricos de Unreal Engine.

CONTROL	**ACCIÓN**
Click derecho + arrastrar	Rota la cámara *Viewport* en el mismo lugar sin trasladarse.

Click izquierdo + arrastrar	Mueve la cámara del *Viewport* hacia adelante y hacia atrás y la rota a izquierda y derecha
Click derecho + click izquierdo + arrastrar	Mueve la cámara del *Viewport* arriba y abajo.
Ctrl + Alt + click + arrastrar	Crea un recuadro de selección múltiple.
Tecla F	Centra el foco de la cámara en el actor seleccionado.
Alt+ botón derecho + arrastrar	Realiza un *zoom* de la cámara del *Viewport* sobre un punto de interés
Alt+ botón izquierdo+ arrastrar	Hace pivotar el *Viewport* alrededor de un punto de interés
Botón izquierdo + ruleta ratón	Permite cambiar la velocidad a la que nos desplazamos por el *Viewport.*

Prison Escape

El presente juego de *Shooter* ha sido desarrollado partiendo de las transparencias sobre Unreal Engine asociadas a la asignatura "Animación Digital" del Grado en Ingeniería Informática de la Universidad de Salamanca[12] y del trabajo realizado por mí mismo para dicha asignatura siguiendo una serie de vídeos explicativos del desarrollo de ciertas funcionalidades [1] [2]

A ello se le han añadido algunos aspectos de desarrollo, así como la introducción explicativa del motor de desarrollo Unreal Engine previamente detallada.

El resultado del juego se puede visualizar bien en el vídeo adjunto o bien descargando Unreal Engine y jugando, haciendo uso del proyecto también adjunto.

Cabe destacar que todos los pasos indicados se muestran con los nombres en castellano resultantes de cambiar el idioma por defecto del motor de desarrollo.

Sinopsis del juego

Un prisionero se ha escapado de la cárcel, debe recorrer una ciudad abandonada donde solo habitan policías *zombies* en busca de una llave que le permita atravesar el portal interdimensional y volver a casa. En el transcurso tendrá que huir de los policías que querrán herirle, recogerá balas y pistola para su defensa, recolectará monedas que harán su vuelta a casa más cómoda y podrá recargar su vida en caso de resultar herido.

Controles

Teclas arriba y abajo para avanzar, izquierda y derecha para girar. Tecla espacio para saltar, tecla P para pausar el juego, tecla C para poder armarse con la pistola.

Desarrollo

El primer paso realizado fue crear un nuevo nivel diferente al de por defecto, que estuviese vacío. Para ello creé una carpeta llamada "MisNiveles" y dentro de ella pulsando botón derecho seleccioné "Nivel*"* y creé uno simplemente con suelo.

A continuación, desarrollé los rasgos básicos del personaje principal.

Personaje principal

Animaciones: he utilizado las mismas animaciones asociadas el *Third Person Character* que se crea por defecto al crear un proyecto de tercera persona, además he añadido una animación para cuando el personaje dispara y otra para cuando el usuario muere, que serán explicadas más adelante. He cambiado el esqueleto asociado a dichas animaciones. Para ello, en primer lugar, he descargado la malla estática de Mixamo [3] y la he importado al proyecto. A continuación, se selecciona *Third Person Character*, botón derecho, editar. En el apartado "Ventana gráfica*"*, seleccionamos *Mesh Inherited*, en *Mesh* debemos seleccionar la malla estática importada.

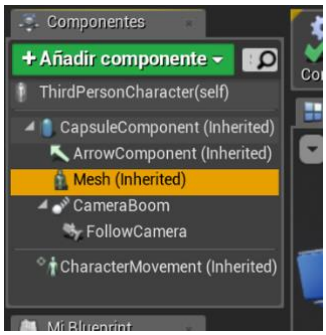

Ilustración 7 Malla estática modificada

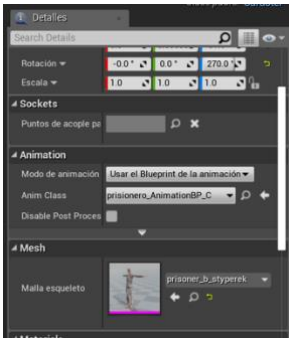

Ilustración 6 Malla estática modificada

En la ventana "Gráfico del evento" simplemente conservamos las animaciones e interacción que ya nos proporcionaba el *Third Person Character*.
Lo siguiente realizado fue el personaje para los enemigos, los pasos seguidos para cambiar la apariencia y animarlo fueron los mismos que para el personaje principal, simplemente se hizo una copia del *Third Person Character* y se siguieron los pasos arriba explicados.
Posteriormente hice que el enemigo, al ver al jugador, lo siguiese por algunas zonas del escenario, para ello introduje el enemigo en el escenario. Primero se debe establecer el área donde el enemigo nos seguirá a través de un objeto *NavMesh Bounds Volume,* este se coloca en el escenario del tamaño deseado y será la zona donde nos podrá seguir el enemigo, en el escenario, en el diseño aparecerá pintado de verde.

Ilustración 8 Objeto límites de malla de navegación

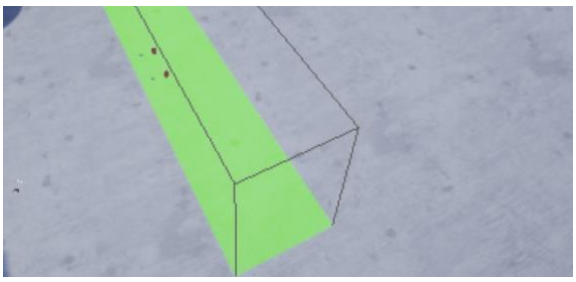

Ilustración 9 Límites sobre escenario

A continuación, pulsando dos veces sobre el enemigo, al abrirlo, buscamos en el menú "Componentes*"*, "*Pawn Sensing"* y se lo añadimos al enemigo. Teniéndolo seleccionado podremos modificar el ángulo de visión periférica donde nos podrá ver.

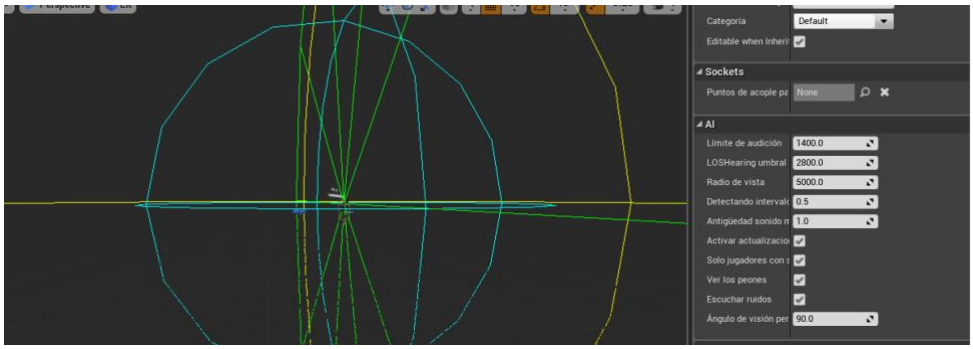

Ilustración 10 Ángulo de visión enemigos

A continuación, en el menú de detalles, en el apartado "Eventos*"*, seleccionamos "En ver peón" para así generar la respuesta al evento cuando el enemigo ve al personaje principal. Ahora en la pestaña *"*Gráfico del evento", lo configuramos de la siguiente manera:

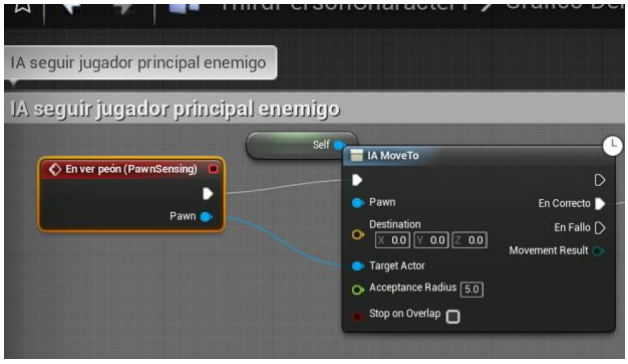

Ilustración 11 Gráfico del evento seguir enemigo IA

De tal forma que el campo *Pawn (pawn* del personaje que está viendo) lo conectamos con *Target Actor* para que así al personaje que ve sea al que siga.
Lo siguiente realizado fue que el jugador tuviese una vida, que se mostrase por pantalla para posteriormente cuando el enemigo que le sigue le alcanzase le fuese quitando vida. Para ello, primero se debe crear una variable en el personaje principal donde se almacene la vida.

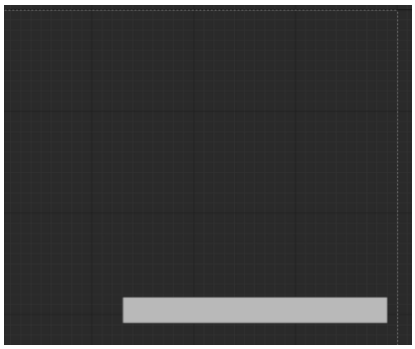

Ilustración 12 Barra de vida

Para mostrar una barra de vida se debe crear un *Widget Blueprint* y dentro de él añadir una barra de progreso personalizándola como se desee.

En el menú de detalles en la barra de progreso en el apartado "*Progress"* seleccionamos que su valor de enlace con una función que se referirá al valor de la variable vida del jugador.

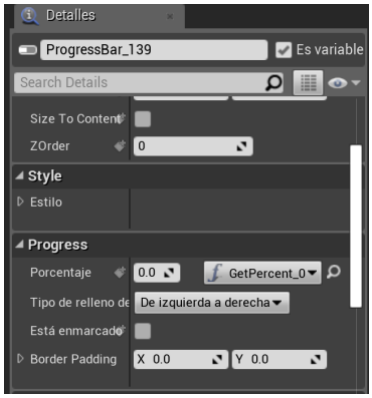

Ilustración 13 Valor de enlace con función de barra de vida

Esta función contendrá

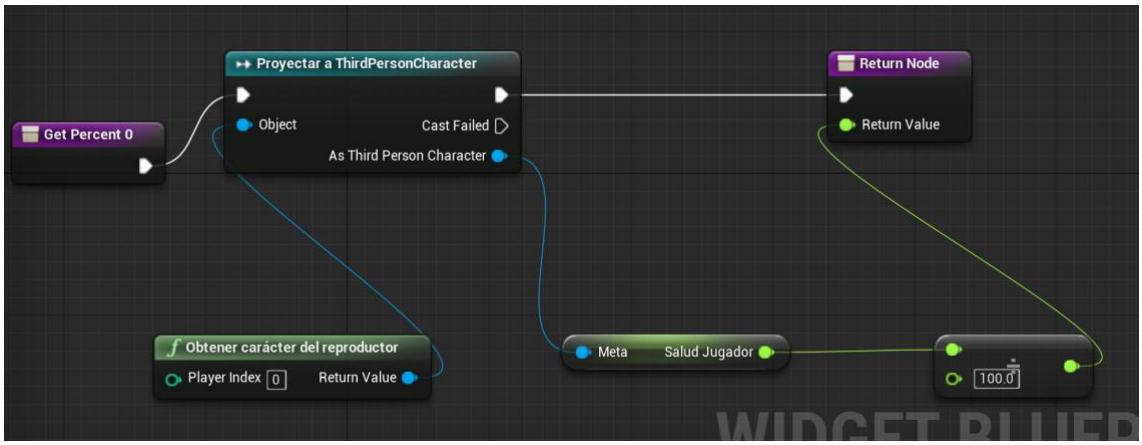

Ilustración 14 Función valor de barra de vida

Donde se obtiene la salud del jugador y se calcula un porcentaje sobre cien y eso será lo que se le devuelva al valor de progreso de la barra de vida.
Por otro lado, en la vida del jugador, se crea el siguiente flujo:

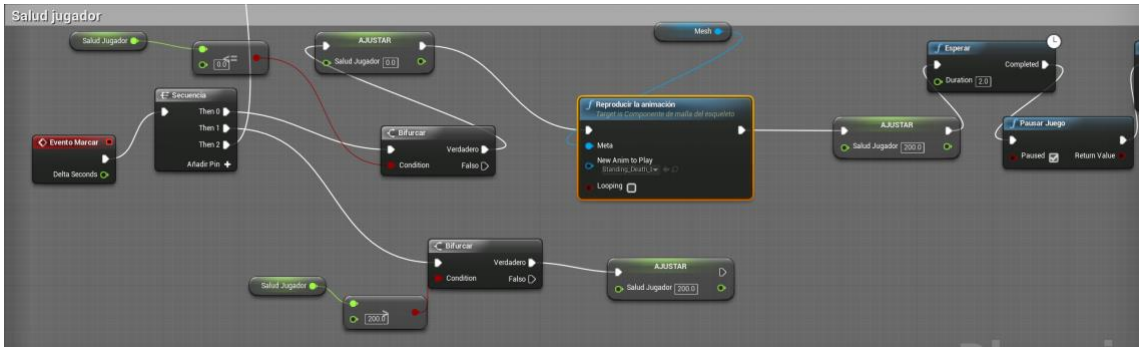

Ilustración 15 Flujo vida del jugador

Donde si la vida del jugador es mayor que 200 (el valor máximo) se establece al valor máximo y si es menor que cero se reproduzca una animación de morir previamente descargada de Mixamo e importada al proyecto, se ajuste al tope la vida y se pause el juego.
A continuación, configuramos al enemigo para que cuando alcance al personaje le vaya restando vida, para ello en el flujo donde hicimos la IA se añade lo siguiente, además se crea otro *Widget Blueprint* con una imagen de mancha de sangre que irá apareciendo con más intensidad cada vez que el enemigo nos reste más vida.

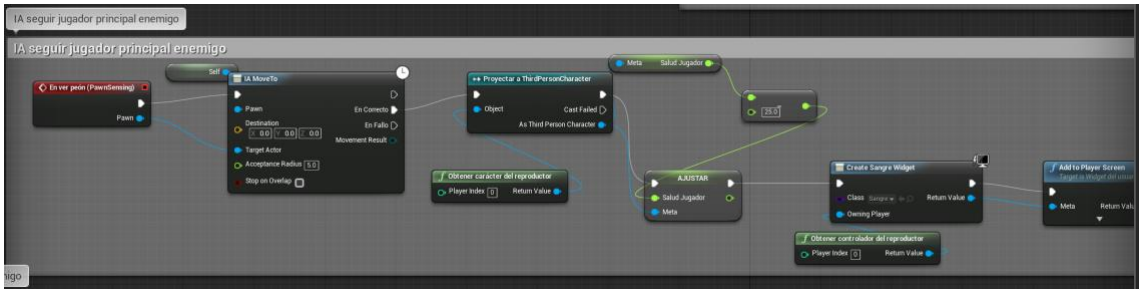

Ilustración 16 Flujo daño realizado por el enemigo

Aquí en el campo "En correcto" de "*IA Move To"*, que significa cuando el enemigo alcanza a su destino, lo unimos con una proyección del *Third Person Character* (el personaje principal) para modificar su variable de vida restándole de 25 en 25 cada vez que el enemigo lo alcanza y mostrando en la pantalla la marca de sangre del *widget* creado al efecto.
A continuación, hice una barra de vida para el enemigo que se situará encima del enemigo y girará para que la podamos visualizar desde cualquier ángulo que lo miremos. Para ello se debe crear un *Blueprint Widget*, crear una barra de progreso como para el personaje principal. A continuación, debemos importarlo a nuestro enemigo. En la ventana gráfica del enemigo añadimos un nuevo componente llamado *widget*, en el campo *User Interface* del menú de detalles seleccionamos el *widget* creado.

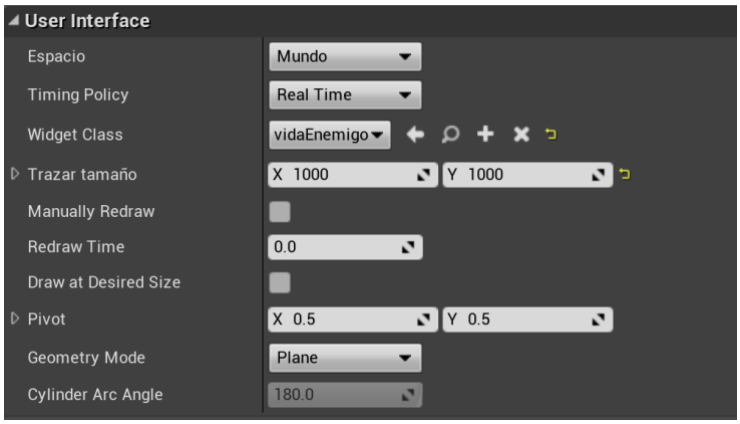

Ilustración 17 Colocación barra de vida del enemigo

Posteriormente la redimensionamos y colocamos. La configuramos en la pestaña gráfico del evento. Para ello creamos una variable de salud en el enemigo. El flujo para controlar la barra de vida del enemigo será:

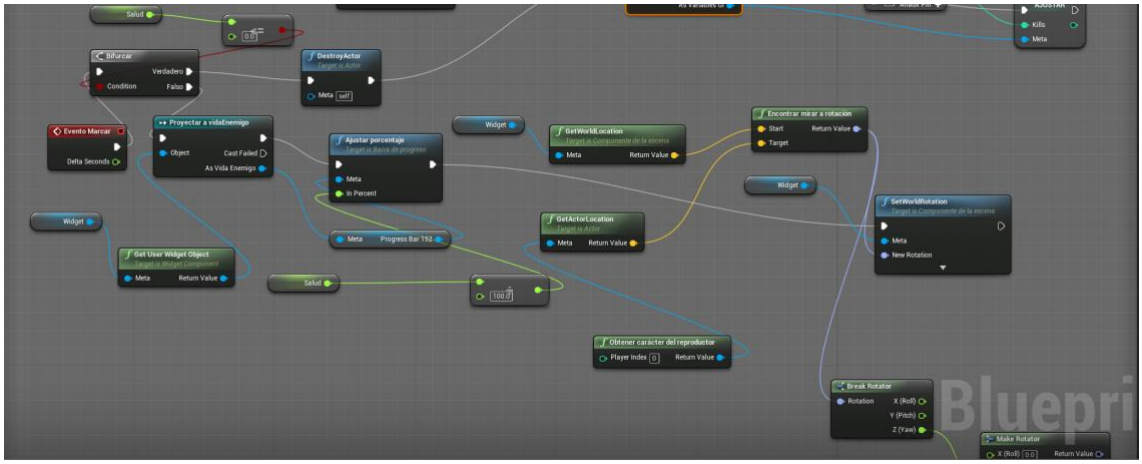

Ilustración 18 Vida del enemigo en el flujo

Donde llamamos a la barra de estado, la salud del enemigo, ajustamos el porcentaje de la barra de salud en función del valor de la variable de salud del enemigo. Posteriormente para ver la barra desde cualquier ángulo debemos obtener la localización del *widget* y en encontrar mirar a rotación hacemos que gire hacia un objetivo / *target* que en este caso es la localización del personaje principal. Finalmente, en "*set world location"* del *widget* lo colocamos en coordenadas del escenario. Para que cuando nos acerquemos no se venga hacia abajo la visualización del widget, con "*Break Rotator"*, en la coordenada Z la unimos a "*Make Rotator"*. A mayores, añadimos que, si la vida del enemigo es menor o igual que cero, entonces se destruya al enemigo de la escena.

Para poder bajar la vida al enemigo, hice que el jugador principal pudiese coger un arma, balas y disparar. El procedimiento es el siguiente:

- Balas: se crea una nueva *Blueprint Class* de tipo Actor, se le añade una malla estática con forma de bala y un *box collider*. En el *box collider* en el evento comenzar superposición se crea el siguiente flujo:

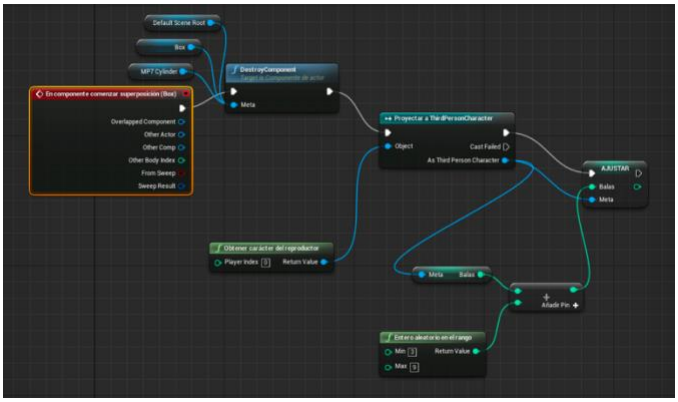

Ilustración 19 Función superposición bala

Cuando se impacte con la bala, esta desaparece y se suma un número aleatorio a una variable del jugador que contendrá el número de balas actuales. Un mecanismo idéntico (más la reproducción de un sonido) es el elegido para recoger monedas del escenario, aunque esta variable se almacena en un objeto que almacenará los valores entre niveles que se explicará posteriormente. Es por ello que, el caso de recoger monedas, por resultar análogo, no se explicará.

- **Coger pistola:** para ello se crea una *Blueprint Class* de tipo actor con la malla en forma de pistola y una malla estática que no tenga colisiones (*NoCollision*) en su apartado *Collision*

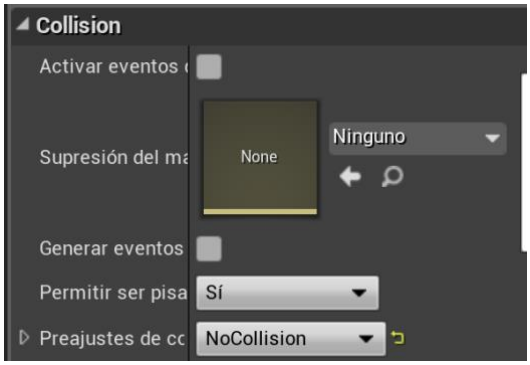

Ilustración 20 Malla estática pistola

Para cogerla cuando nos acerquemos a ella pulsando la tecla C, debemos primero en Editar> ajustes del proyecto > Motor > Entrada establecer una nueva asignación de acción a una tecla

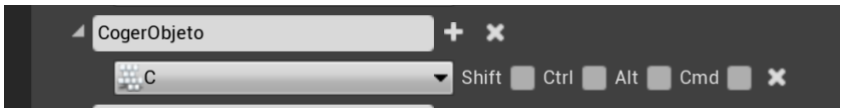

Ilustración 21 Configuración tecla coger pistola

Luego debemos introducir en el mismo lugar donde situemos la pistola un objeto *TriggerBox* y con él seleccionado pulsamos en la pestaña *Blueprint >Open Level Blueprint*

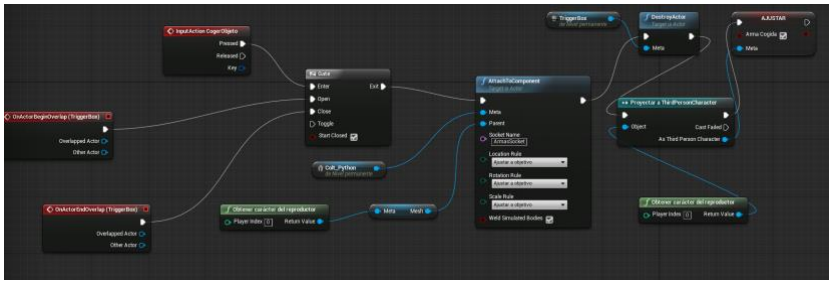

Ilustración 22 Interacción jugador - objetos a recoger

Añadimos el evento de pulsar la tecla C y los eventos cuando el personaje entre y salga de superposición con el *trigger box*. El módulo *Gate* indica que el evento de pulsar la tecla C se activará si estamos en superposición con el *trigger box* y si no lo estamos entonces no se activará. Para coger el objeto, pulsamos la pestaña Ventana, Paleta, *Attach To Component.* En este módulo la meta será el arma y se la anidaremos al personaje (el personaje será su padre) en el lugar del socket creado. Una vez cogida, eliminamos el *trigger box* para que no pueda haber más interacciones con él y ponemos a cierta una variable booleana del personaje como que tiene el arma cogida. Esta variable servirá para indicar si el personaje puede disparar o no, podrá disparar si tiene el arma cogida y la cantidad de balas que posee es superior a cero.
Previamente debemos haber creado un socket en el esqueleto del personaje donde situar el arma cuando la coja. Para ello seleccionamos el personaje, pulsamos dos veces, seleccionamos la pestaña de esqueleto, seleccionamos la parte del cuerpo donde queremos que se sitúe el arma, botón derecho, *add socket*. Posteriormente importamos el arma (seleccionamos el arma (*static mesh*) descargada la arrastramos hacia el socket creado. Para cambiar su posición simplemente modificamos la posición del socket creado.

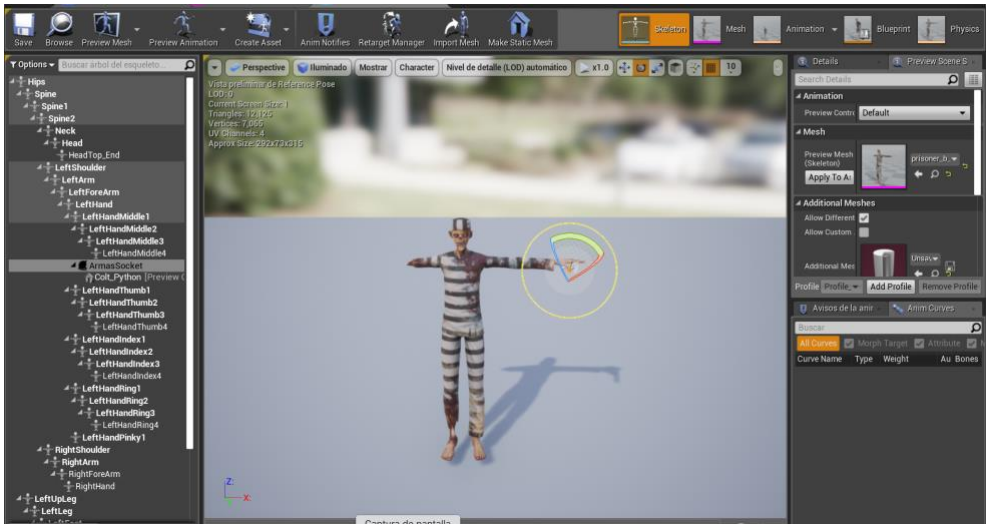

Ilustración 23 Colocación del socket posición arma

Es importante que el objeto del arma, en el menú de detalles, en el apartado Transformación, sea movible y en colisiones que no tenga colisión.

Ilustración 24 Propiedades de transformación del arma

- **Disparar:** en primer lugar, debemos asociar a la pulsación del botón derecho del ratón el evento de disparar, para ello seguimos los pasos explicados previamente para la tecla C de recoger el arma. Llamamos a ese evento en la pestaña "Gráfico del evento*"* de nuestro personaje principal. Obtenemos el gestor de la cámara, la localización de nuestro personaje. Creamos una línea de trazada que seguirá nuestro disparo (una línea donde con lo que se cruce impactará), el punto de origen será la localización del actor y el punto de fin un vector resultante de sumar una distancia a la posición del personaje. Cuando golpee con algo, generamos un emisor de un sistema de partículas que las emitirá en la localización del punto final de la línea de trazada. Para solo poder disparar cuando tengamos balas y la pistola cogida, ponemos una condición *AND* con ambas, resultando que cuando es cierta se reproduce una animación de disparar descargada de Mixamo e importada de forma asociada al personaje principal, si la condición es falsa mostraremos una cadena por pantalla que indicará "Sin balas" y no se reproducirá la animación ni se creará la línea de trazada explicada.

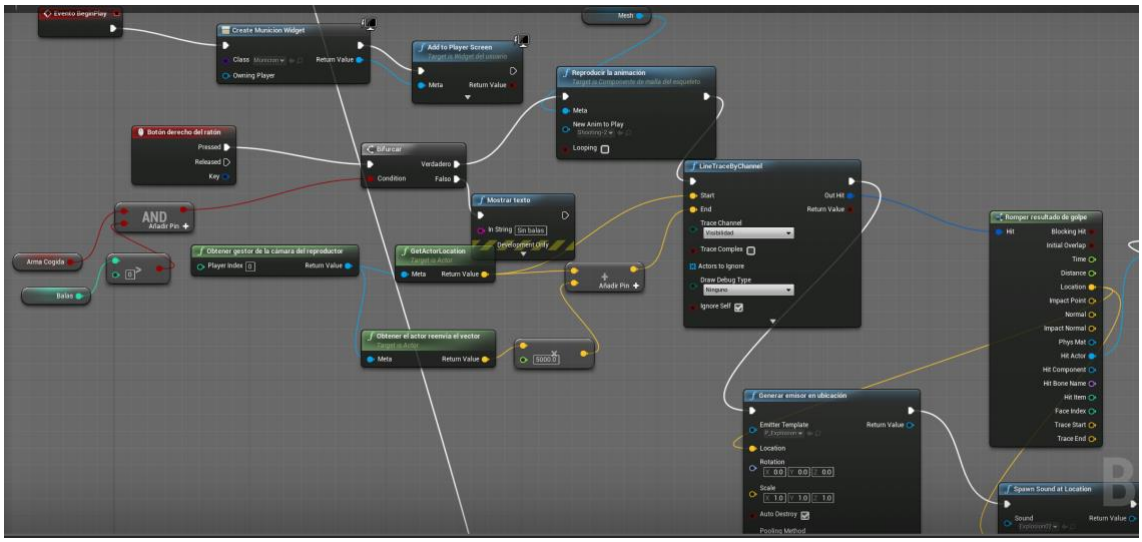

Ilustración 25 Flujo jugador: disparo

Posteriormente debemos bajar la vida al enemigo si la línea de trazado impacta con él. Para ello, en el campo *Hit Actor* del módulo de "Romper resultado de golpe*"* recuperamos el actor del enemigo y posteriormente obtenemos su variable vida y modificamos su valor.

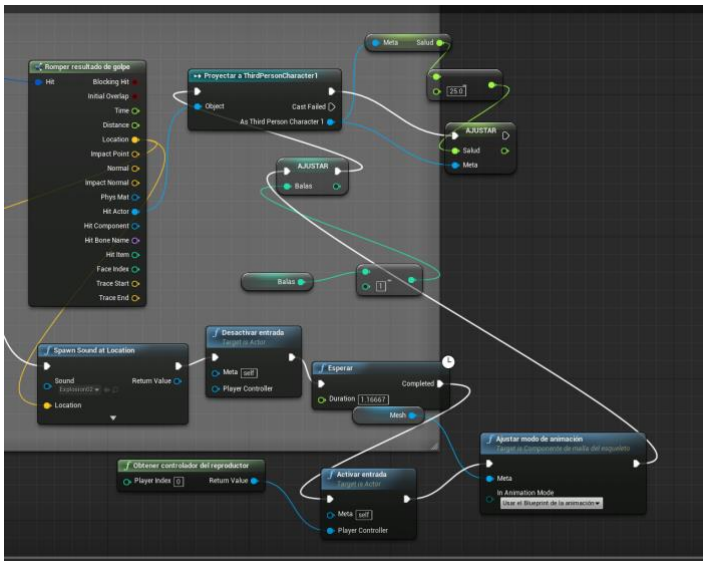

Ilustración 26 Flujo enemigo: impacto del disparo

En la parte inferior del flujo hacemos que mientras dispara no pueda realizar ninguna otra acción.

Posteriormente se realiza lo necesario para representar la muerte del jugador y el menú de derrota. Para el menú de derrota se crea un *Blueprint Widget* que aparezca cuando la vida del jugador sea menor o igual a cero. En él aparecerán los botones de reintentar (que sumará un intento a una variable de un objeto que almacena valores entre niveles) y cargará de nuevo el nivel. Otro botón salir que cerrará el juego. Aparecerán por pantalla los valores de enemigos abatidos y monedas conseguidas recuperados del objeto con información persistente.

- **Botón volver a jugar:** *seteará* las variables de enemigos abatidos y monedas conseguidas a cero y sumará uno a los intentos realizados en el objeto de persistencia de información

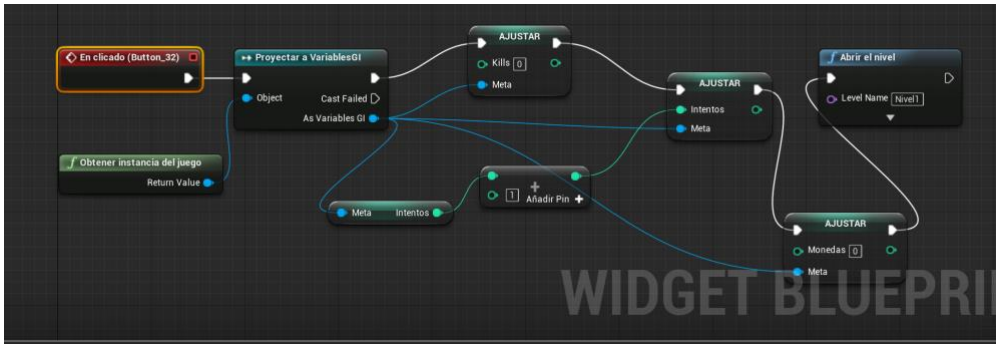

Ilustración 27 Flujo del botón de volver a jugar

- **Botón salir**

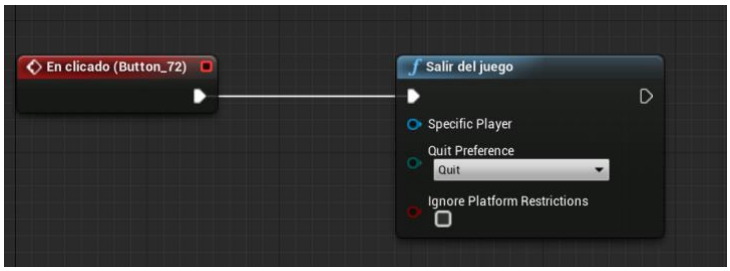

Ilustración 28 Flujo del botón de abandonar partida

Una vez realizado un menú de fin de juego, creé un objeto donde poder almacenar variables entre niveles. Para ello debemos crear una *Blueprint Class* de tipo *Game Instance*. Dentro de ella creamos las variables que queremos almacenar entre niveles (monedas recogidas, enemigos abatidos, intentos realizados y si hemos conseguido la llave para completar el nivel).

Ilustración 29 Objeto manager de variables a traspasar entre niveles

Para recuperar el valor de alguna variable de este objeto en cualquier sitio basta con llamarlo. Para en los resultados mostrar el valor de estas variables hay que enlazar el contenido del texto a mostrar con el valor de la variable a mostrar, por ejemplo, en el caso del menú de muerte, para mostrar los enemigos abatidos:

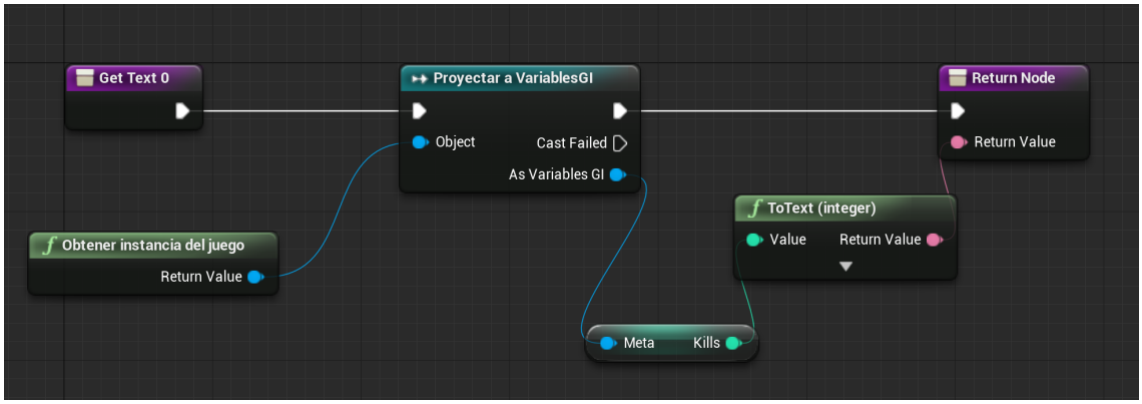

Ilustración 30 Recuperar variables entre niveles

Donde obtenemos el objeto de persistencia, recuperamos la variable deseada y la devolvemos como valor de retorno al cuadro de texto. El proceso para mostrar el valor del resto de variables en este y otro menúes posteriormente explicados resultaría análogo. Un aspecto importante para tomar el objeto creado como el de persistencia del juego es que debemos seleccionar en Ajustes del proyecto qué *Game Instance* queremos. En el apartado *Project* > Mapas y modos > *Game Instance* > Clase de instancia de juego. Ahí seleccionamos el objeto creado.

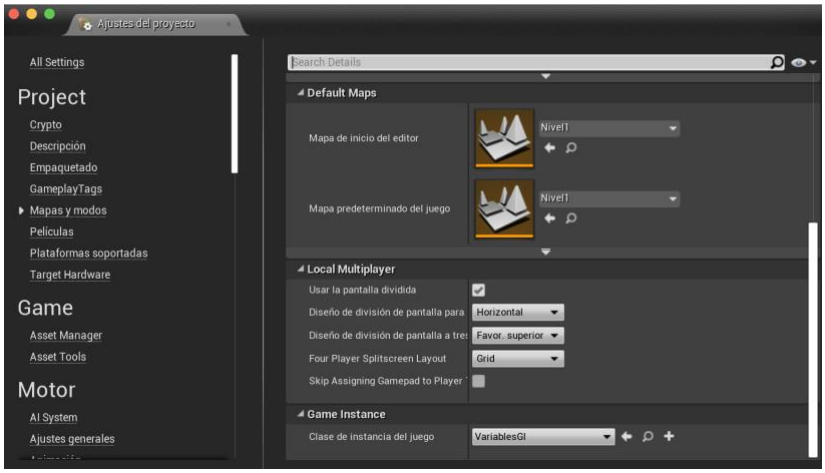

Ilustración 31 Configurar objeto de persistencia

Una vez creado un menú de fin de juego, creé un nivel de inicio de juego. La creación del nivel es análoga a la creación del nivel de juego solo que este nivel no contendrá nada más allá de un *Blueprint widget* donde se mostrarán las acciones de jugar y salir del juego en botones cuyos comportamientos son análogos a los explicados en otro botones previamente. Para establecerlo como nivel por defecto del juego al iniciarlo debemos acudir a ajustes del proyecto, mapas y nodos, y en *Default Maps* en el apartado de Mapa predeterminado del juego seleccionar el que deseemos.

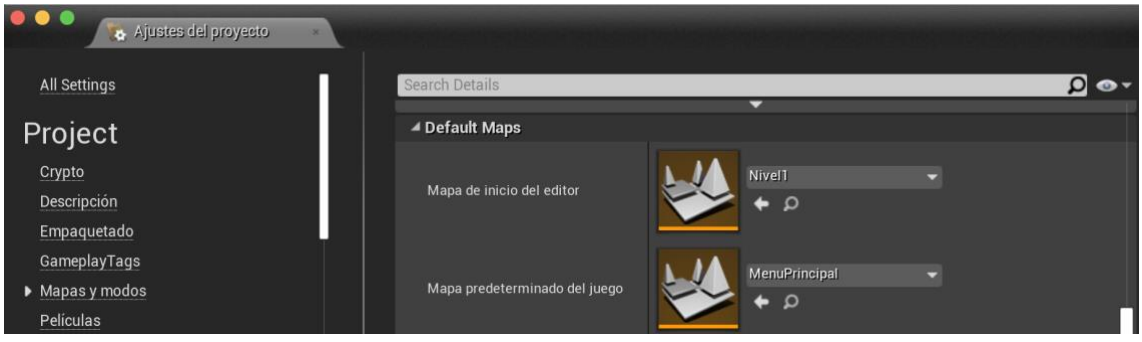

Ilustración 32 Creación y configuración de nivel inicial explicativo

Posteriormente perfeccioné la muerte del jugador, cuando su vida fuese menor que cero, reproducir una animación de muerte y cargar el *widget* de fin de juego previamente

explicado, este es el flujo resultante de completar el flujo ya explicado de la vida del jugador:

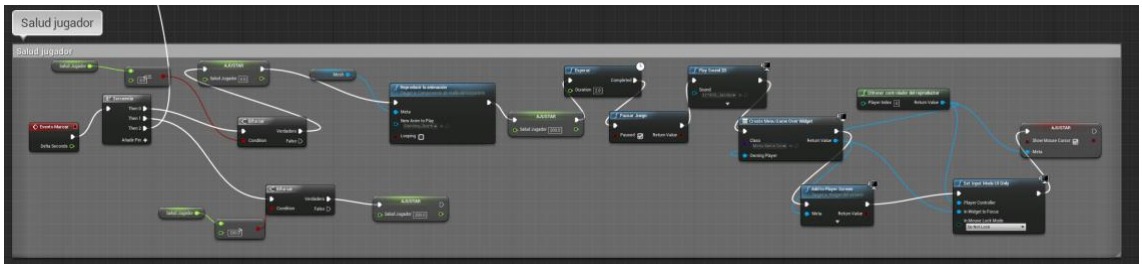

Ilustración 33 Flujo completo del jugador

Donde el módulo *Set Input Mode UI Only* es para deshabilitar la entrada del jugador y solo poder interaccionar con ratón e interfaz de usuario.
Para poder superar el juego debemos crear un nivel donde ir al superar el anterior. Por motivos de espacio (para que el juego no ocupase mucho más) he decidido que cuando supere el nivel, se le redirija a otro donde solo habrá un panel donde se reflejen los enemigos abatidos, monedas recogidas e intentos realizados y un botón que al pulsarlo nos cargue el nivel del menú principal. Para poder superar el juego el usuario deberá encontrar una llave y en uno de los edificios habrá un portal interdimensional que al tener la llave le permitirá la consecución de la victoria. Para pasar de nivel se ha colocado un *trigger box* encima del portal que, de modo análogo a lo explicado para el *trigger box* de coger el arma, cuando el personaje entre en superposición con él y la variable "llaveCogida" del objeto de persistencia sea *true* se cargará el nivel de resultados explicado reproduciéndose una música.

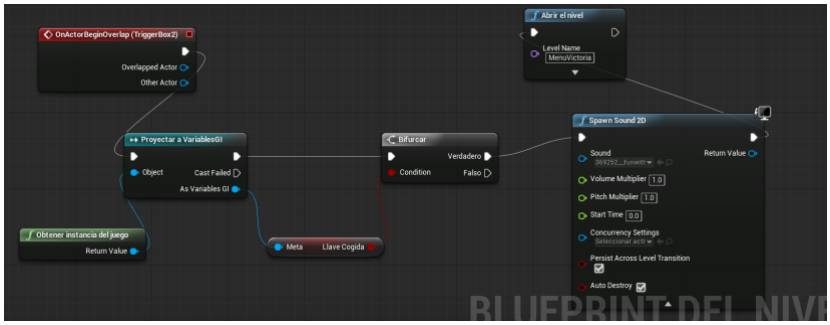

Ilustración 34 Configuración trigger box cambio de nivel

Para coger la llave, el mecanismo es análogo a coger una moneda o una bala, solo que, en vez de sumar a las variables del jugador, se pone a cierta la variable que indica si la llave se ha cogido del objeto de persistencia.
Finalmente, he realizado los siguientes detalles para indicar al jugador que tiene que recoger una llave, poder subir a las azoteas de los edificios, entrar en los edificios y abrir puertas.

- **Escaleras para subir a los edificios:** para ello he creado una *Blueprint Class* a la que le he puesto una malla estática con la forma de escaleras de pared descargada,

a ello le he añadido un *box collision* para detectar la colisión del jugador con las escaleras y hacer que las suba. Cuando el personaje entra en superposición con las escaleras se ajusta a verdadero el valor de una variable del jugador y cuando dejar de estar en superposición se ajusta a falso y se reestablece el modo de movimiento del *Third Person Character* de nuevo a "Caminar".

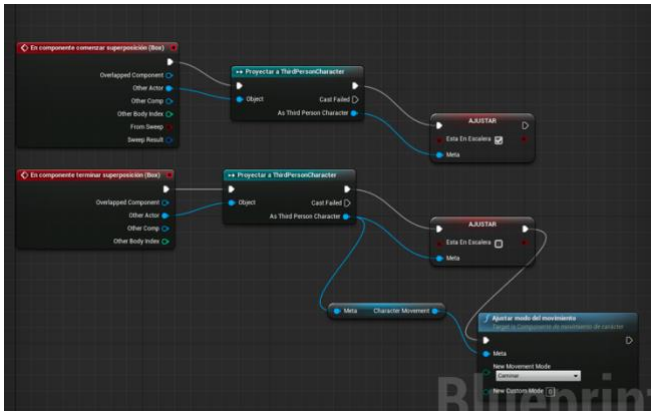

Ilustración 35 Interacción jugador - escaleras

En el jugador, cuando la variable de estar en las escaleras vale verdadera se ajusta el modo de movimiento del personaje a "Volar" para que así pueda ascender y crear un efecto simple de subir escaleras. Habría que incorporar una animación de subir escaleras para dotar de más realismo al efecto, pero no lo he realizado por no encontrar una ya realizada y la complejidad de construirla desde cero.

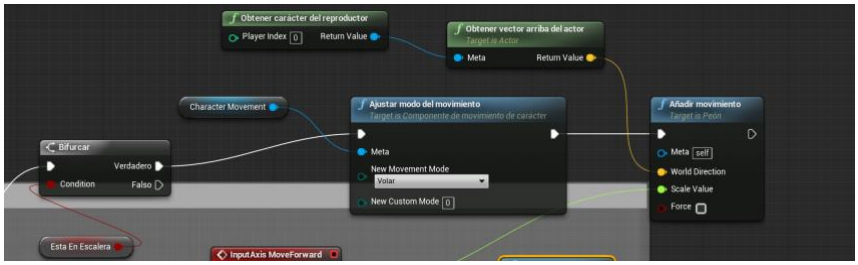

Ilustración 36 Flujo jugador : subir escaleras

Además, para poder ascender a otras azoteas, además de pasar de una a otras, he creado una serie de plataformas móviles mediante el uso de *TimeLines*. Para ello, en primer lugar se añaden al escenario los objetos que se quieren mover, en este caso cubos que, modificando sus dimensiones, se moldean a la forma de plataformas. Posteriormente se les añade una malla estática a estos como se realizó con los personajes (botón derecho > blueprint class > actor > añadir componente > malla estática).
A continuación, a través de la pestaña "Gráfico del evento" se crea el *TimeLine* (Botón derecho > *timeline* > añadir línea de tiempo > doble *click* >añadir pista del vector (V+) > congelamos los ejes que queramos para que el movimiento se realice en el que queramos (eje Y en este caso))

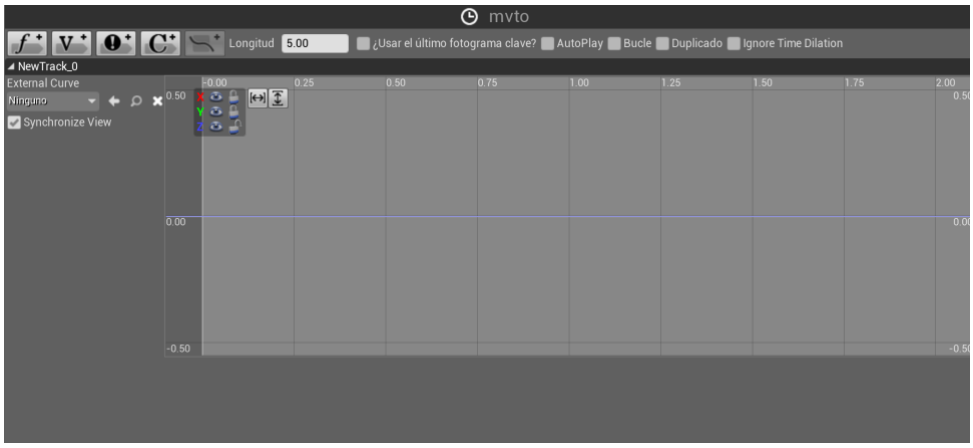

Ilustración 37 TimeLine plataformas

Realizando esto el movimiento podría resultar "a tirones" por lo que procedí a suavizarlo, para ello se seleccionan los tres puntos de movimiento (por los que pasa la plataforma), creados pulsando botón derecho, *add key to Z*, se pulsa botón derecho sobre uno de ellos y se selecciona "auto".
Finalmente se crea el gráfico del evento resultando este:

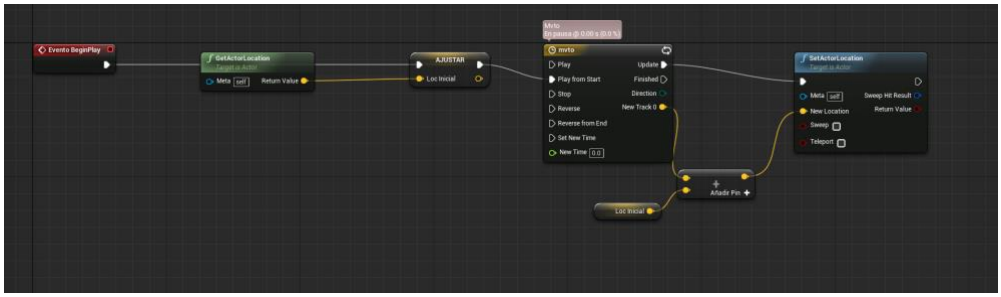

Ilustración 38 Flujo movimiento platformas

- **Entrar en los edificios:** para poder entrar en los edificios simplemente es necesario modificar la malla estática de los edificios importados (Botón derecho > editar) en la pantalla que se despliega en el menú de detalles, apartado de colisiones especificar en complejidad de la colisión "Usar colisión compleja como colisión simple" para que tome los diferentes componentes de la malla como separados y no como un bloque.

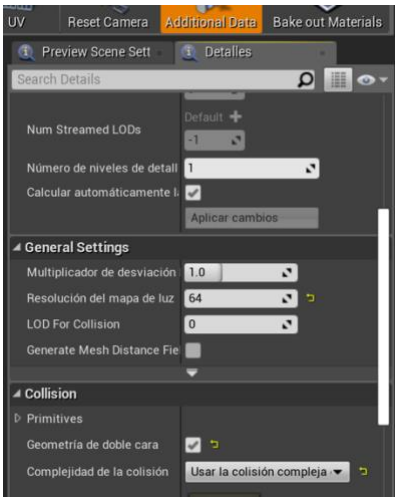

Ilustración 39 Cambiar malla estática edificios

- **Abrir puertas:** importando una malla estática de una puerta, añadiendo un *box collision* que cuando el jugador impacte con él se realice la reproducción de un *TimeLine* y cuando deje de impactar que vuelva a su posición inicial.

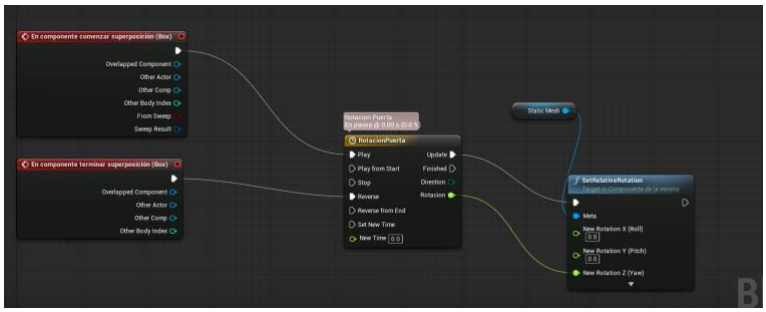

Ilustración 40 Movimiento apertura puertas

- **Personaje ayudante que indica al jugador que debe recoger una llave:** añadimos a la escena un personaje como los enemigos, pero sin IA ni animaciones que esté continuamente en posición de pie. A este personaje le añadimos en la pestaña "Ventana Gráfica" un componente de renderizado de texto donde escribimos el mensaje a transmitir. Añadiremos también un *box collision* al personaje para detectar cuándo el personaje principal se acerque mostrar el mensaje y cuando se aleje, ocultarlo. En el evento comenzar superposición y finalizar superposición realizamos el siguiente flujo:

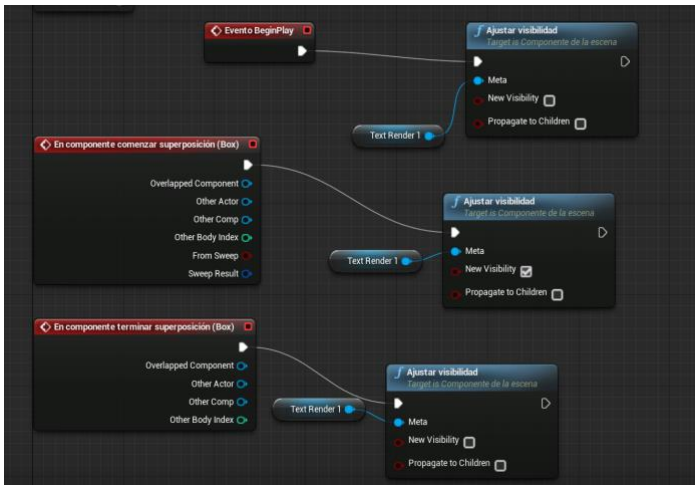

Ilustración 41 Cuadro de diálogo personaje ayudante

Cuando comience la superposición ajustaremos a visible la visibilidad del texto mientras que cuando se termine lo ocultaremos. Además, nada más comenzar no queremos que se visualice por eso marcamos su visibilidad como oculta.

Referencias

[1]*Youtube.com*, 2020. [Online]. Available: https://www.youtube.com/user/HoracioMeza27/videos. [Accessed: 17- Dec- 2020].

[2]*Youtube.com*, 2020. [Online]. Available: https://www.youtube.com/channel/UCjj_5i2y2UcGvxtBkUYWcNg/videos. [Accessed: 17- Dec- 2020].

[3]"Mixamo", *Mixamo.com*, 2020. [Online]. Available: https://www.mixamo.com/#/. [Accessed: 17- Dec- 2020].

[4]"Epic Games Store | Official Site", *Epic Games Store*, 2020. [Online]. Available: https://www.epicgames.com/store/es-ES/. [Accessed: 17- Dec- 2020].

[5]"Tipos de licencias de software: software libre, propietario y demás | Velneo", *Velneo*, 2020. [Online]. Available: https://velneo.es/licencias-software-libre-propietario-otros/. [Accessed: 17- Dec- 2020].

[6]"Videojuego de rol", *Es.wikipedia.org*, 2020. [Online]. Available: https://es.wikipedia.org/wiki/Videojuego_de_rol. [Accessed: 17- Dec- 2020].

[7]"UDN - Two - KarmaReference", *Docs.unrealengine.com*, 2020. [Online]. Available: https://docs.unrealengine.com/udk/Two/KarmaReference.html. [Accessed: 17- Dec- 2020].

[8]"High-dynamic-range rendering", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/High-dynamic-range_rendering. [Accessed: 17- Dec- 2020].

[9]"City of Titans | Answer the Call", *Cityoftitans.com*, 2020. [Online]. Available: https://cityoftitans.com. [Accessed: 17- Dec- 2020].

[10]S. Enix, "KINGDOM HEARTS III", *Kingdomhearts.com*, 2020. [Online]. Available: https://www.kingdomhearts.com/3/es/home/. [Accessed: 17- Dec- 2020].
[11]B. Adventure, "Bacon Man: An Adventure on Steam", *Store.steampowered.com*, 2020. [Online]. Available: https://store.steampowered.com/app/327760/Bacon_Man_An_Adventure/. [Accessed: 17- Dec- 2020].
[12]Á. L. Murciego, *Sesión 1 - Sesión 6 Animación Digital,Grado en Ingeniería Informática,* Salamanca: Dpto Informática y Automática, Universidad de Salamanca, 2019.