PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
MECOLAB

Diego Andreu Casas Ubilla
dacasas@uc.cl
1er semestre del 2016

Tutorial Unreal Engine 4

Índice

    b. Iluminación

6. Inteligencia Artificial
    a. Mesh del jugador
    b. Movimiento “inteligente”

7. Video 360°
    a. Multimedia necesaria
    b. Media Texture
    c. Sky Sphere

8. Comentarios y documentación de apoyo

1. Introducción

En el siguiente tutorial se intentará abarcar ciertos temas relacionados con el desarrollo tridimensional en Unreal Engine, un motor gráfico para desarrollar juegos en las diferentes plataformas existentes hoy en día.

El desarrollo en esta plataforma fue principalmente con el motivo de implementarlo en realidad virtual, a través del dispositivo Gear VR de Samsung. Por lo que también se abarcarán temas relacionados con este.

a. Unreal Engine

Unreal Engine es un motor de juegos muy famoso utilizado prácticamente en todo el mundo por desarrolladores de muchas plataformas, debido a su capacidad de crear juegos para la gran mayoría de las plataformas disponibles actualmente.

En el tutorial se explicarán las de características principales del programa, primero cómo instalarlo, y luego cómo configurarlo para poder desarrollar para el Gear VR. Luego se verán temas más técnicos, como lo son la construcción de estructuras, donde se explicará un poco los componentes básicos de Unreal (meshes, materiales, texturas, luces y sonidos), también la manipulación de objetos en el editor y, finalmente, la importación de objetos 3D desde otros programas.

También se verá el movimiento de un jugador, tanto por entradas físicas (teclas), como también a través de una inteligencia artificial. A esto también se suman las animaciones que pueda requerir el movimiento.

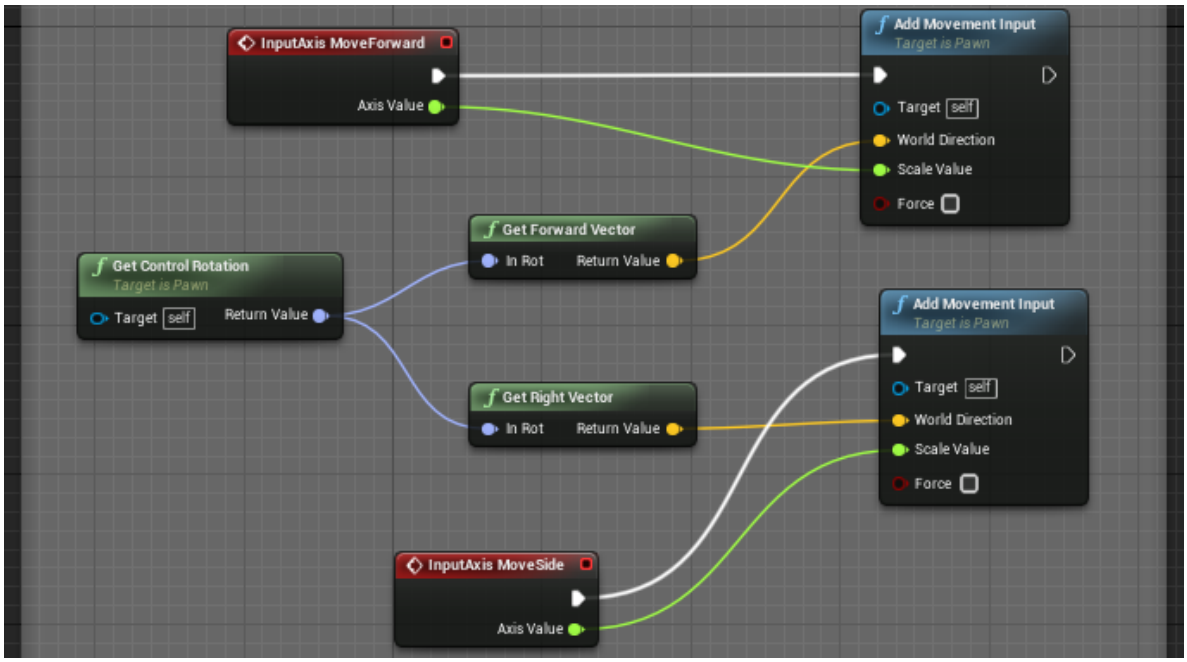
Finalmente se mostrará como reproducir un video 360 dentro del juego. Vale mencionar que esto último es sólo para computador.

b. Blueprints

En el tutorial se utilizará el nuevo modelo de programación que introdujo Unreal Engine en su cuarta versión, Blueprints. Este consiste en una especie de diagramas de flujo que son el símil de las líneas de código en un programa tradicional. Esto se hizo para acercar más a programadores y diseñadores, ya que se hacía difícil para los diseñadores entender el código.

A modo de introducción, esta nueva forma de “programar” es una muy buena forma de representar la ejecución de código.

A modo de ejemplo, así luce un pequeño trozo de “código” de Blueprints.



Debido a la gran variedad de elementos que posee Unreal Engine, sólo se tocarán superficialmente los temas mencionados, ya que una profundización en ellos requiere prácticamente un tutorial completo dedicado a cada uno.

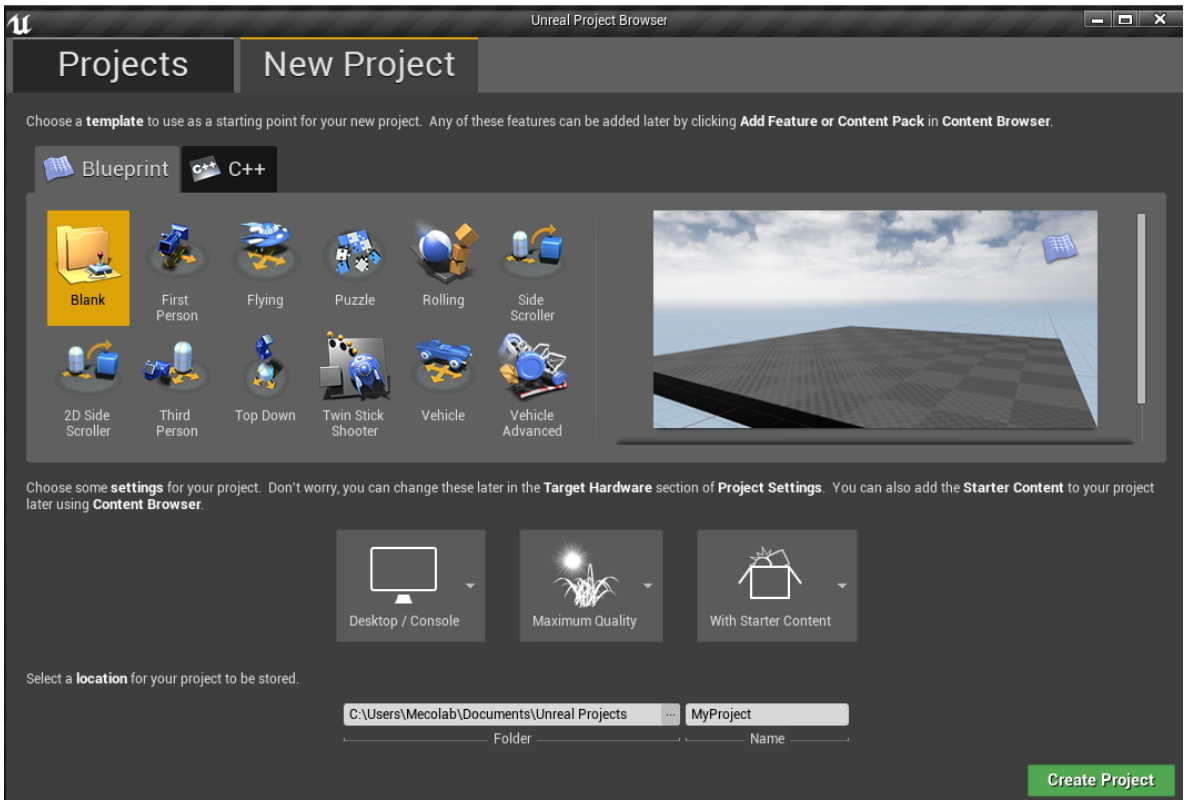
2. Preparación **del ambiente de desarrollo**

a. Instalación de Unreal Engine

Para instalar unreal engine, basta sólo con ingresar a la página https://www.unrealengine.com/ y presionar en “Get Unreal”. Luego se debe crear una cuenta para poder desarrollar. También es necesario el software gratuito de Microsoft, Visual Studio, de preferencia la última versión.

b. Creación del proyecto

Al momento de crear un nuevo proyecto, debes seleccionar la configuración inicial de tu proyecto, dependiendo para que plataforma deseas desarrollar. La pantalla inicial debiese verse así.



Es altamente recomendado que, si estás creando un proyecto por primera vez, lo crees con contenido para principiantes (última configuración de selección). Esto ya que en el contenido mencionado viene una gran cantidad de material para ayudar a desarrollar mucho más rápido, como formas básicas, paredes, pisos, y muchos materiales y texturas.

También se recomienda mucho explorar los proyectos de ejemplo que salen ahí, ya que muestran de forma muy simple lo potente que es el motor, y la gran cantidad de aplicaciones que se pueden crear con él.

Se puede notar, en la parte superior donde sale el tipo de proyecto que se desea crear, sale también el lenguaje (de programación) a utilizar. Para efectos de este tutorial, todos los proyectos se crearon con Blueprints, porque se deseaba aprender este nuevo lenguaje introducido por Unreal Engine 4.

c. Desarrollo en Gear VR

El desarrollo para Gear VR requiere algunas modificaciones al proyecto, que se explicarán a continuación.

i. Software y Hardware necesario

En temas de hardware obviamente se necesita el Gear VR, además del celular con el que se utilizará. Los dispositivos disponibles aparecen en la página oficial de Oculus

(https://www3.oculus.com/en-us/gear-vr/). Las aplicaciones de este tutorial fueron todas desarrolladas para el Samsung Galaxy S6, y el Gear VR "Innovator Edition for S6".

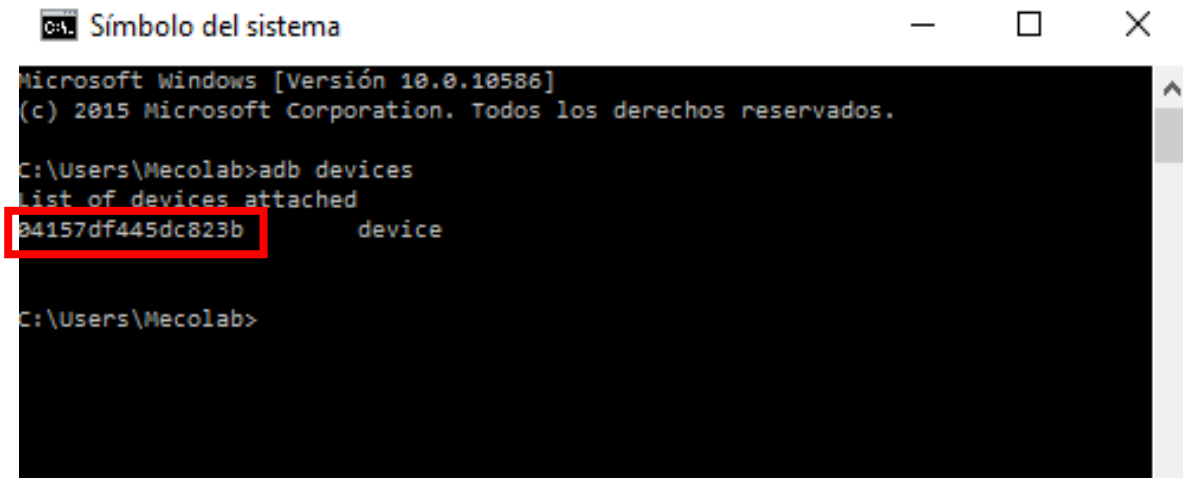
Para temas de software, se necesita primero que el celular se encuentre en modo desarrollador, con el "USB debugging" activado (*Settings* → *About Device* → *Software Info* → Presionar en "*Build Number*" 7 veces y deberían aparecer las opciones para desarrollador. Nuevamente se va a *Settings* → *Developer options* → *USB debugging*). Esto es para poder instalar aplicaciones no oficiales en el terminal, ya que los terminales Android por defecto no dejan instalar aplicaciones que no provengan de la Play Store.

También se necesita un software fundamental para la comunicación entre en computador y el celular, llamado "Tegra Android Development Pack", que está en: https://developer.nvidia.com/gameworksdownload , se debe buscar en la lista y descargar para el sistema operativo deseado. Para poder descargarlo se debe estar registrado en la página como desarrollador. Este software trae consigo el SDK, NDK y JDK, que son herramientas necesarias para poder crear aplicaciones para la plataforma de Android.

Una vez instalado este último programa, luego de reiniciar el computador (si fue necesario), se debe descargar el "Oculus Signature File (osig)". Para esto primero es necesario la creación de una cuenta en https://www.oculus.com/ . Luego se ingresa a https://dashboard.oculus.com/tools/osig-generator/ , donde pide el ID del dispositivo (celular). Este ID lo pide, ya que las aplicaciones de realidad virtual, deben estar construidas con una firma única (con el ID del dispositivo), de tal forma de poder acceder a funcionalidades de bajo nivel del dispositivo.

Para obtener el ID, hay que abrir la consola (shell o cmd para Windows), y con el celular conectado al computador, se escribe el comando "adb devices". Este comando, llama al programa adb (Android debug bridge), el cual permite la comunicación entre el computador y un dispositivo Android, y con el comando devices, entrega una lista de los dispositivos conectados al computador.

El ID del dispositivo es el que aparece marcado en la imagen.



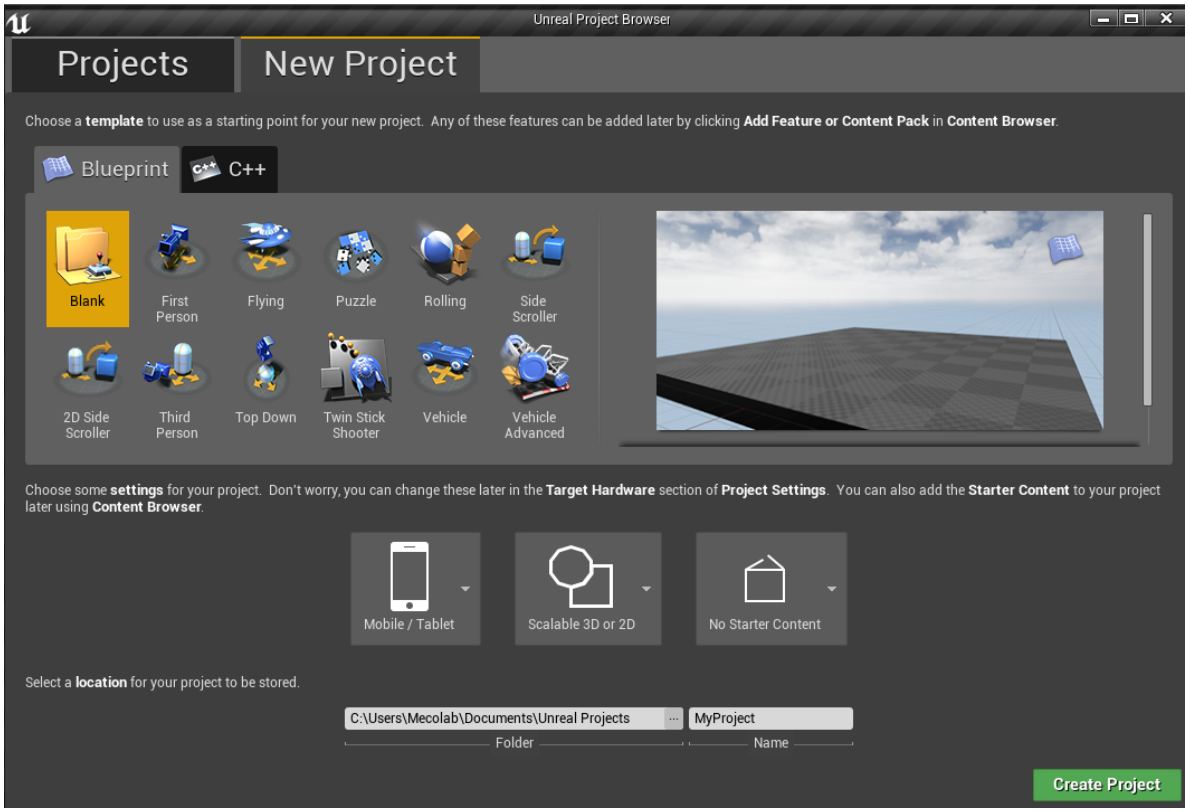
Una vez descargado el archivo, debe ser puesto en la siguiente ruta:

C:\Program Files (x86)\Epic Games\[Versión]\Engine\Build\Android\Java\assets

Algunas cosas a tomar en cuenta son que la carpeta assets debe ser creada en caso de no encontrarse, y donde dice [Versión], debe ser reemplazado con la versión del motor en la que se desea desarrollar. Para efectos del tutorial, el principal proyecto se creó en la versión 4.11.

ii. Configuración del proyecto para Gear VR

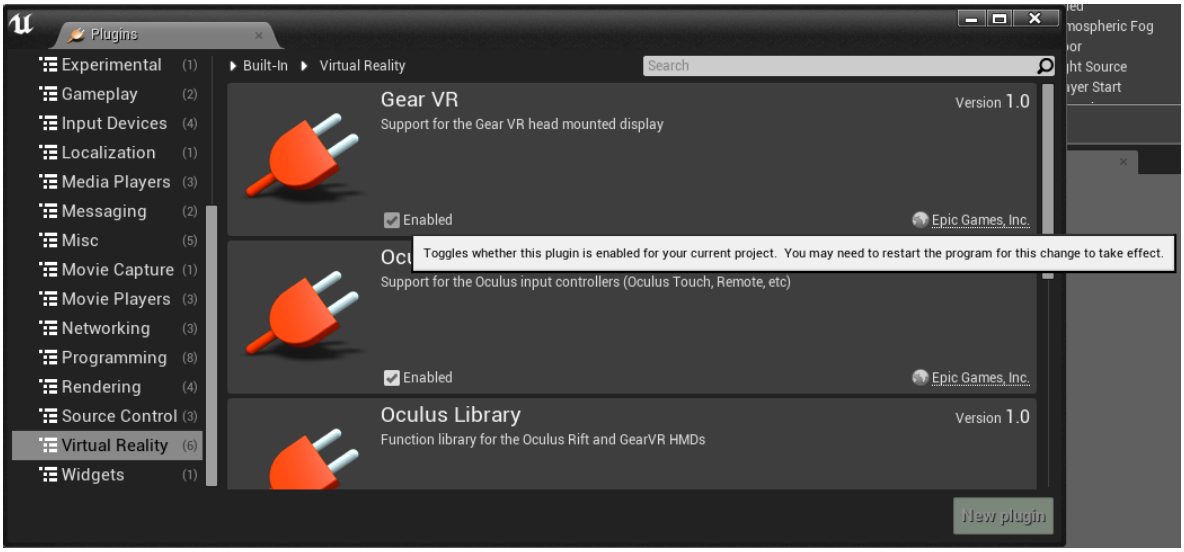
Ahora, para la creación de proyectos enfocados a Gear VR se debe realizar una serie de configuraciones. Primero, la ventana de creación de un nuevo proyecto debiese verse de esta manera.



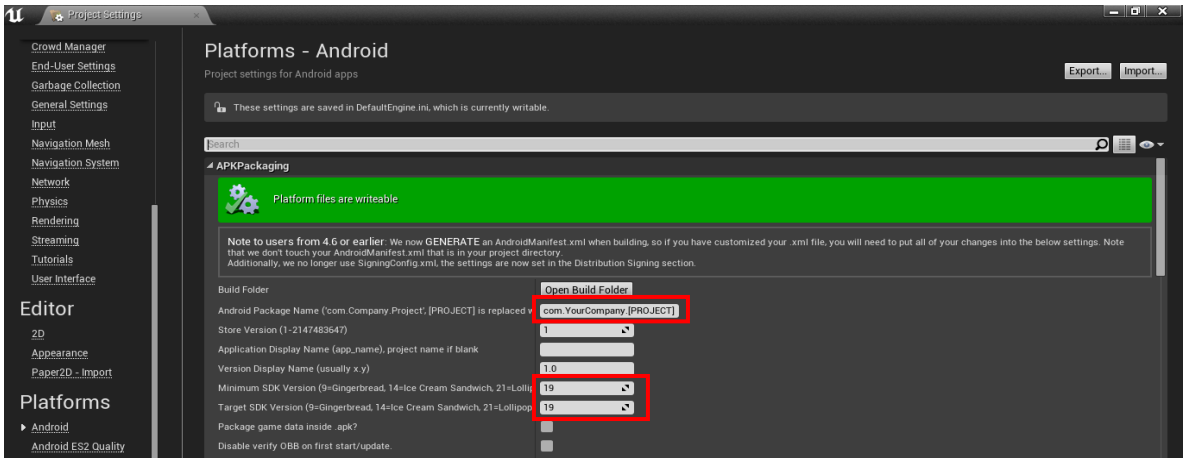
Primero, la plataforma a la que va enfocado el proyecto, es a dispositivos móviles, luego no se desea la máxima calidad para estos dispositivos debido a que no tienen tanto poder de procesamiento como lo posee un computador; y, finalmente, no se incluye el contenido de principiantes ya que este utiliza mucho espacio (debido a la gran cantidad de materiales que trae), cuestión que debe evitarse lo más posible, ya que los celulares tienen un almacenamiento muy limitado.

Como comentario, en un principio (Blueprints salió junto a la cuarta versión del editor) sólo se admitían proyectos (para móviles) desarrollados en C++, ya que es un lenguaje muy eficiente y los dispositivos móviles necesitan la mayor eficiencia posible. Pero ahora ya se pueden crear proyectos con Blueprints, ya que se ha hecho este lenguaje más eficiente de lo que era en un principio.

El siguiente paso a realizar se hace una vez creado el proyecto, en "*Edit*" → "*Plugins*". Luego se presiona "*Virtual Reality*" en la columna de la izquierda, y el plugin para Gear VR debe encontrarse activado. Si se encontraba desactivado, será necesario reiniciar el programa para el correcto funcionamiento del plugin. Los demás plugins pueden activarse, dependiendo de para que plataformas se desea desarrollar.



Luego, se debe ir a "*Edit*" → "*Project Settings*", y en el menú izquierdo se presiona "*Android*" que se encuentra bajo el título de "*Platforms*". Una vez ahí, si aparece un mensaje acerca de que el proyecto no está configurado para Android, solo basta con presionar en el botón que dice "Configurar Ahora". Luego, las configuraciones resaltadas en la siguiente imagen son las que se deben modificar.



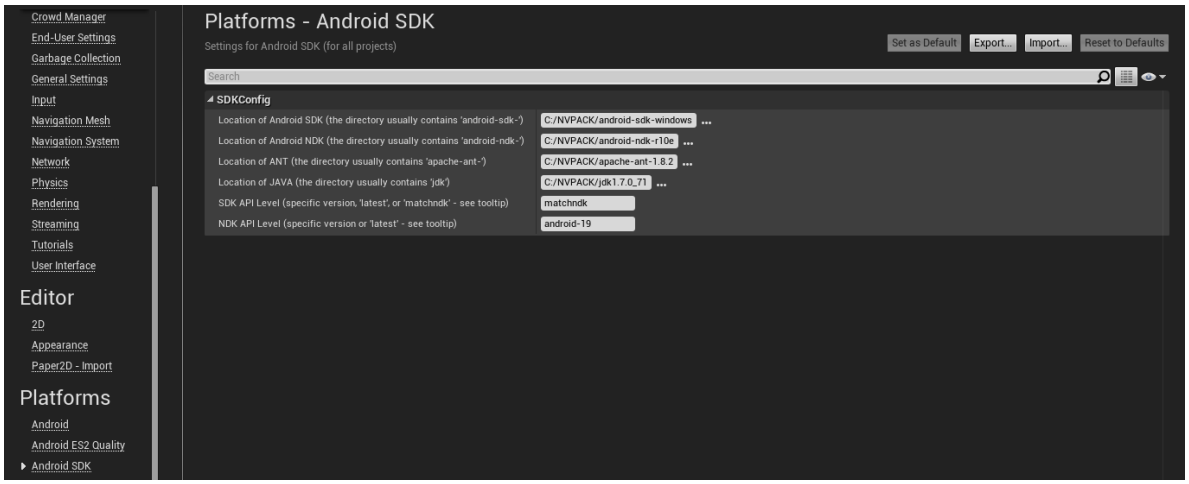
La primera resaltada, se debe reemplazar, tal como dice ahí con el nombre de la empresa, y el nombre del proyecto. Y las dos siguientes son la mínima versión del SDK soportada por la aplicación, y la versión del SDK a la que esta apuntada por el proyecto. En la versión mínima, debe ponerse la 19, ya que es la mínima que soporta esta característica; y en la versión apuntada es recomendado poner la última versión disponible, ya que es la máz exenta de bugs.

Luego, tal vez el cambio más importante que se debe efectuar, es el completar la casilla que dice "*Configure the AndroidManifest for deployment to GearVR*", ya que esta permite el correcto despliegue de la aplicación para el Gear VR.

Finalmente, se debe indicar al programa donde se encuentra el SDK y NDK, así como otros componentes necesarios para el desarrollo en Android. Este último paso se debe hacer sólo una vez, ya que el editor guarda las ubicaciones para los demás proyectos.

Esto se hace de manera similar a la anterior, se va a "*Edit*" → "*Project Settings*", y ahí se va a Android SDK, que está bajo el mismo título de "*Platforms*". Estas rutas se encuentran donde haya sido instalado el Tegra Android Development Pack. Puedes usar los de la imagen para guiarte, en caso de no saber dónde fue instalado.

Las otras configuraciones "SDK y NDK API Level", deben estar de acuerdo con las anteriores, que en caso del tutorial era la versión 19.

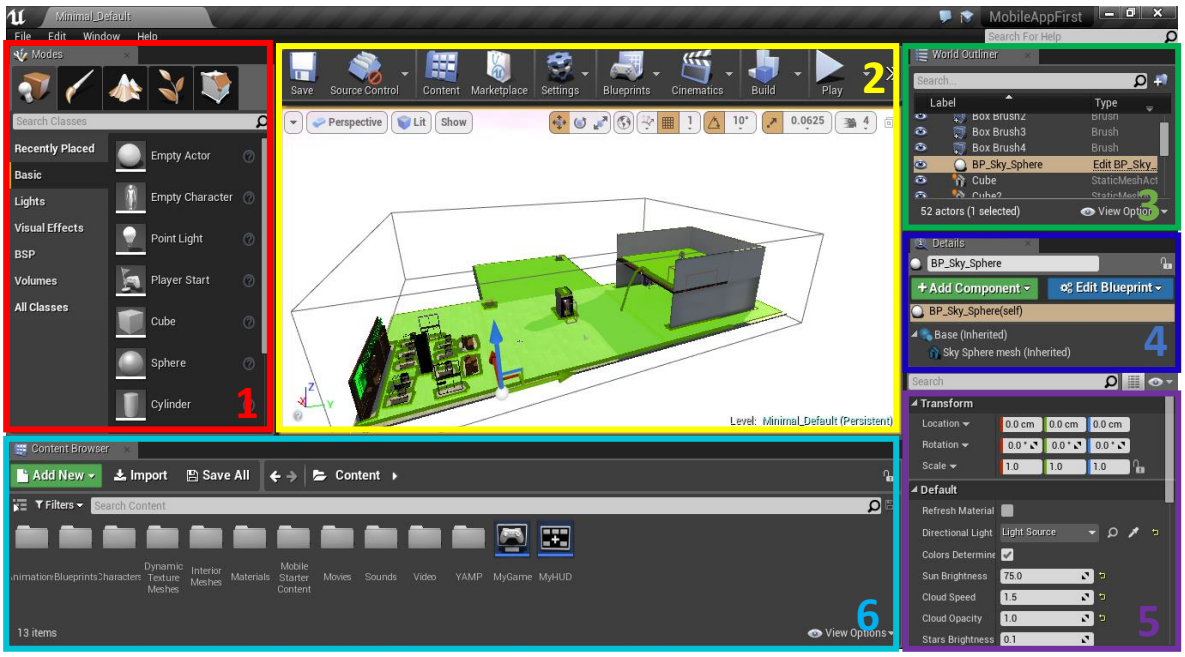


Ahora, para cargar la aplicación desarrollada en el celular, se debe ir a "*File*" → "*Package Project*" → "*Android*" → "*Android (ETC2)*". Esta última, es el tipo de compresión utilizado para el proyecto, pero se recomienda ETC2. Luego pregunta el lugar donde se desea desplegar el proyecto.

Una vez terminada la Build del proyecto, en la ubicación seleccionada con anterioridad existirá un archivo de extensión .bat que es para instalar la aplicación en el celular. Al presionar ese archivo, con el celular conectado al computador, se abre una consola en la que se ve el proceso de instalación de la aplicación en el celular. Una vez finalizada la instalación, la consola se cierra de manera automática, y la aplicación ya debería estar instalada en el celular. Luego, para ejecutarla se abre desde el celular, se pone el celular en el Gear VR y debería comenzar a correr la aplicación.

d. Interfaz de usuario

En esta sección explicaremos la interfaz de usuario del programa.



En la sección 1 se encuentra el selector de modos, que es desde donde se puede modificar el nivel, principalmente, de donde se pueden obtener componentes (actores) para agregar al mapa. Se encuentran tanto actores básicos, como también el objeto para definir la posición inicial del jugador. También se encuentran los distintos tipos de luces que serán explicados más adelante, diferentes tipos de volúmenes (de audio, "gatilladores" de eventos, etc.), objetos de tipo personaje; en fin, se encuentra todo lo que se puede agregar al mapa.

Los otros modos que incluye son el de pintura (para editar los colores), terreno (para editar el terreno del nivel), follaje (para ver opciones de follaje en el nivel) y edición geométrica (para editar las formas presentes en el nivel).

La sección 2 es el editor del nivel que, como se puede deducir fácilmente, es donde se ve el nivel/mapa en el que se está trabajando. Desde ahí se puede editar directamente la ubicación de los objetos (actores), su ángulo de inclinación y su tamaño. Tiene diferentes tipos de vistas configurables directamente, para adecuar la vista a lo que se desee hacer. Finalmente, en la barra de herramientas (parte superior) están los botones de acción que se utilizarán mucho a lo largo del desarrollo, como el botón de guardar el mapa, el para acceder al Blueprints del mapa (como el diagrama de ejecución del mapa, que se ejecuta al iniciar la aplicación), el para hacer la Build del nivel, principalmente para temas de iluminación, ya que estos programas "pre-procesan" las luces con los objetos estáticos, para no hacer que el dispositivo en el que se ejecute el programa tenga que hacer trabajo de más. Y el último que se ve en la imagen, es para iniciar el nivel, dentro del mismo editor, de forma de poder ir probando las características nuevas rápidamente.

La sección 3 es el "World Outliner", que es donde se encuentran todos los actores que se encuentran (componen) en el nivel, tanto materiales, como luces y sonidos, entre otros.

El menú 4, es parte de los detalles del componente seleccionado, e indica los detalles de la estructura, es decir, los componentes por los que está compuesto, valga la redundancia.

La sección número 5 es la de Detalles, que indica las propiedades del actor que está seleccionado. Dentro de estas propiedades se encuentra la ubicación espacial, la rotación, su tamaño, sus propiedades físicas (estático, con gravedad), su capacidad de colisión, el material del que está compuesto, la forma que tiene (Mesh), y muchas más.

Finalmente, el sector 6 es el explorador de contenido que tiene el proyecto. Es donde podemos buscar los materiales para los actores, sus meshes (forma) y texturas. Ahí también se encuentra la multimedia que deseemos agregar al proyecto, como videos, sonidos o imágenes.

3. Construcción de Estructuras

La construcción de estructuras en Unreal es bastante amigable e intuitiva, pero para hacerla aún más clara, primero explicaremos los conceptos básicos que involucran a los actores y sus propiedades visuales. Entendiendo como actores a todo objeto que se encuentre en el nivel, ya sea un personaje, una pared, un sonido, luz, etc.

a. Brush, Mesh, Material, Textura

Los brushes son las formas más básicas que puede tener un nivel (esfera, cubo, cono, entre otras).

Los Meshes son figuras tridimensionales más complejas que los brushes, usados mayoritariamente en programas de diseño 3D.

Al Mesh se le llama estático, cuando son guardados en la memoria de video y son rendereados por la tarjeta gráfica, por lo que no pueden cambiar su forma. Esto permite que tengan formas más complejas que los *brushes*, ya que su rendereo es mucho más eficiente.

Otro tipo de Mesh, es el Mesh esquelético, que es aquel que tiene asociado a él una serie de huesos. Para explicar de mejor forma esto, primero pensemos en un Mesh con forma de persona, luego existe un componente que asocia el brazo del Mesh con un hueso, sus piernas con otro, y así con las extremidades que se deseen mover. Finalmente, para que se logre mover, es necesario una animación, que exprese que huesos se deben mover, de tal manera que cuando la animación es aplicada al Mesh, este se mueva de la forma deseada.

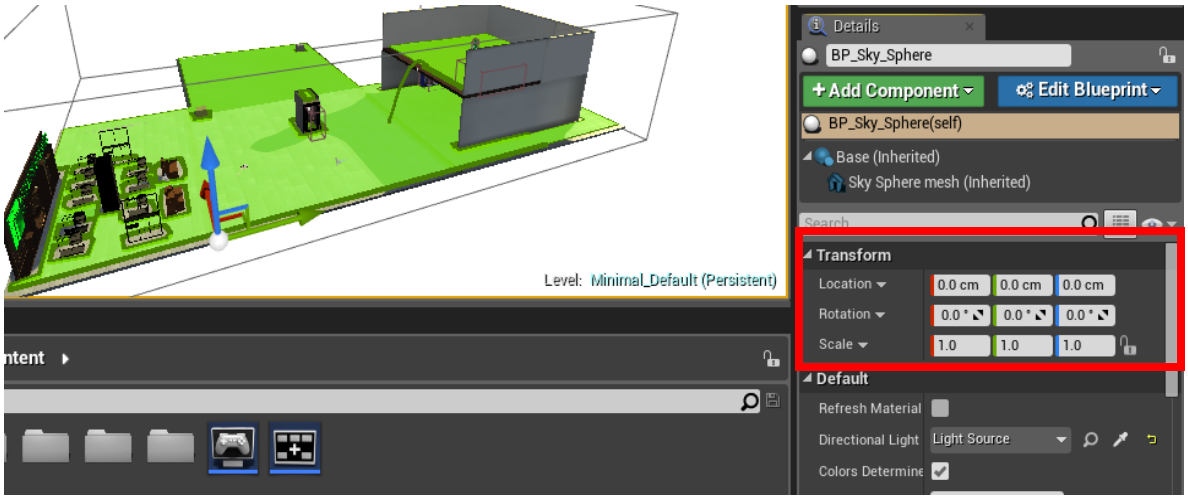
El material es un componente que se le aplica al Mesh, para controlar la apariencia visual de este. Puede ser pensado como la "pintura" del objeto al que se le aplica, pero también controla el brillo del objeto, transparencia, etc. En el fondo, también define el material del que el objeto está hecho. Y cada material puede tener asociado una textura, que básicamente es una imagen que se le es aplicada al material.

b. Construcción

El primer punto importante en la construcción de un nivel, es la navegación a través de este. Que puede parecer muy complicado en un principio, pero se hace muy cómodo luego de usarlo un tiempo.

Si mantenemos el clic izquierdo presionado, y movemos el mouse horizontalmente, la cámara rota hacia el lado correspondiente. Pero si movemos el mouse verticalmente, vemos que la cámara avanza o retrocede, cambiando su posición. Luego, si mantenemos el botón derecho presionado, podemos girar la cámara hacia cualquier dirección; ahora, manteniendo el mismo botón presionado, y además presionamos las teclas W, A, S, D o las flechas de dirección, la cámara se moverá en esa dirección. La velocidad de este movimiento se ajusta con el mismo botón derecho presionado, pero con la rueda del mouse.

Segundo, si ya experimentaste con el editor, ya debes haber descubierto como mover los objetos y dejarlos en el lugar donde quieres. Si no, cuando seleccionas un actor en el editor, aparecen los tres ejes en los que lo puedes mover, y si quieres aún más exactitud, puedes ir al panel de propiedades de ese objeto y ubicarlo de acuerdo a las coordenadas deseadas (imagen a continuación).



Un tema no menos importante, es como poner actores (objetos) en el mapa, cuestión que es bastante sencilla ya que basta con arrastrarlos desde el explorador de contenido, o desde el menú izquierdo, en el que aparecen las formas básicas para colocar en un nivel. Es recomendado que se usen meshes estáticos en vez de brushes, ya que como se mencionó antes, los meshes son mucho más eficientes.

c. Importación desde programas externos

El programa también permite la importación de meshes construidos en otros programas de diseño tridimensional. Esto es fundamental, ya que de otra forma serían siempre las mismas formas en los juegos desarrollados con el editor.

Para efectuar esto, primero se debe exportar el modelo deseado a un archivo de extensión FBX, ya que esta extensión es usada por la mayoría de los programas de modelado 3D. Luego se importa desde el editor, en "*File*" → "Import Scene", y seleccionando el archivo a importar; o simplemente arrastrar el archivo que deseas importar directamente al explorador de archivos del editor. Finalmente se seleccionan las propiedades deseadas del Mesh a importar.

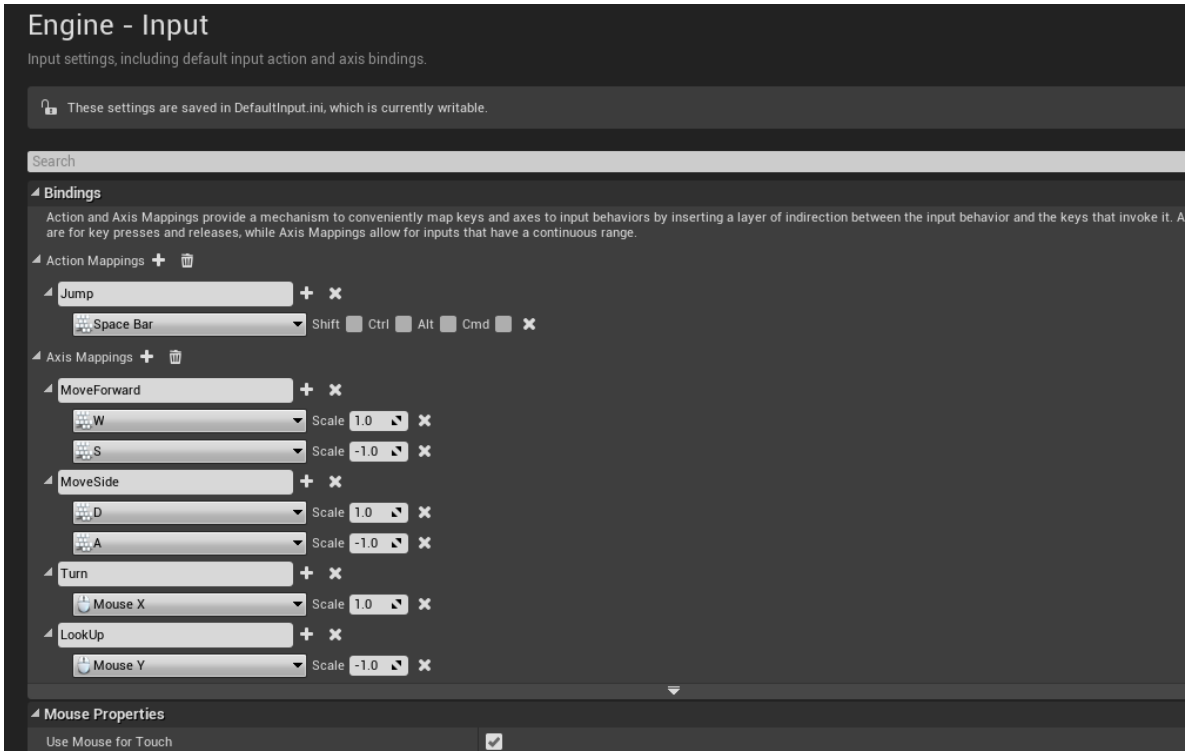
4. Movimiento del Jugador

En este capítulo veremos cómo crear un jugador, y hacer que se mueva por el nivel, con las teclas W, A, S, D o las flechas de dirección, además del mouse para mover la cámara que se encontrará en modo de primera persona. También le daremos a nuestro jugador la habilidad de saltar con la tecla espacio.

a. Configuración de inputs

Primero necesitamos decirle al programa que teclas presionaremos para efectuar el movimiento, así como también que input corresponderá al movimiento de la cámara.

Para esto nos vamos a "*Edit*" → "*Project Settings*", y luego en "*Input*", bajo el subtítulo de "*Engine*". En esta sección, agregamos los "*Axis Mapping*" y "*Action Mappings*" como aparecen en la imagen.



Se puede notar que es bastante simple la interfaz para agregar inputs al proyecto, se presiona en el signo más, y además se pueden agregar varios tipos de inputs para un cierto evento ("*Axis o Action Mapping*"), para poder soportar acciones en dos direcciones, o simplemente para soportar otros tipos de controles, como los son cualquier gamepad, o touchpad.

Se nota también el factor de escala que tiene cada evento, esto para efectos del movimiento significará las dos direcciones posibles (positiva y negativa) en un eje, por ejemplo.

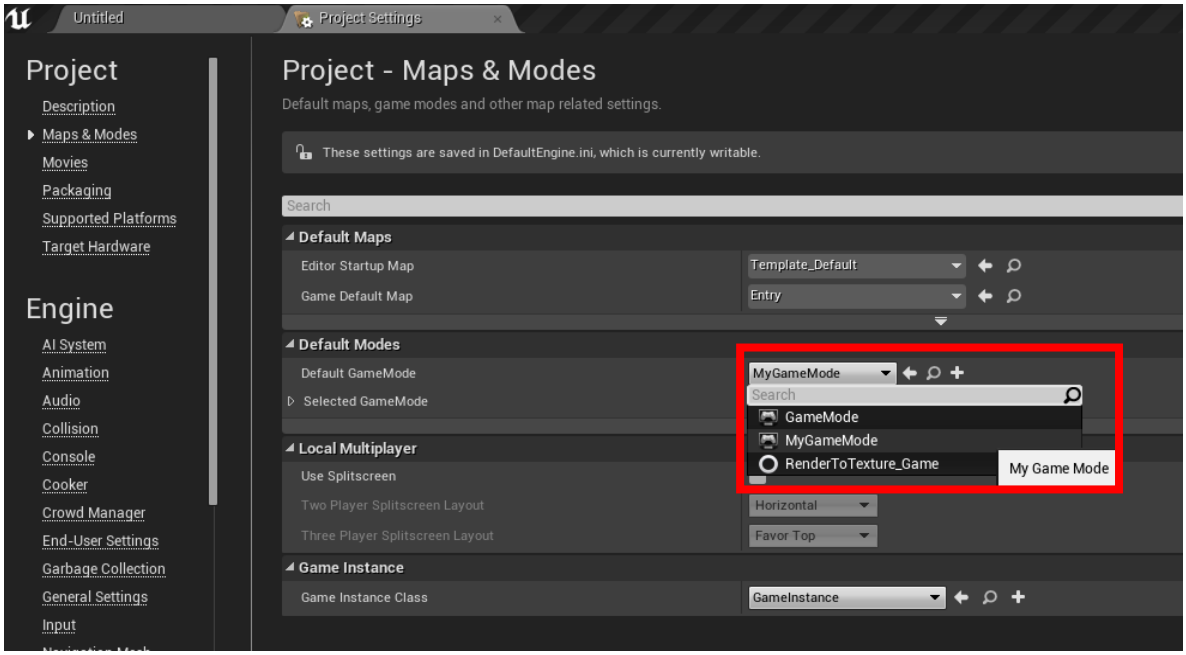
b. Creación del jugador

Primero debemos crear una nueva clase de Blueprints, la cual se puede crear fácilmente desde el explorador de contenido, arriba a la izquierda hay un botón que dice "*Add New*", luego seleccionamos "*Blueprint Class*". Se abre un menú para seleccionar que tipo de clase deseamos, y seleccionamos la clase "*Character*". Esta clase, como lo dice el mismo editor es

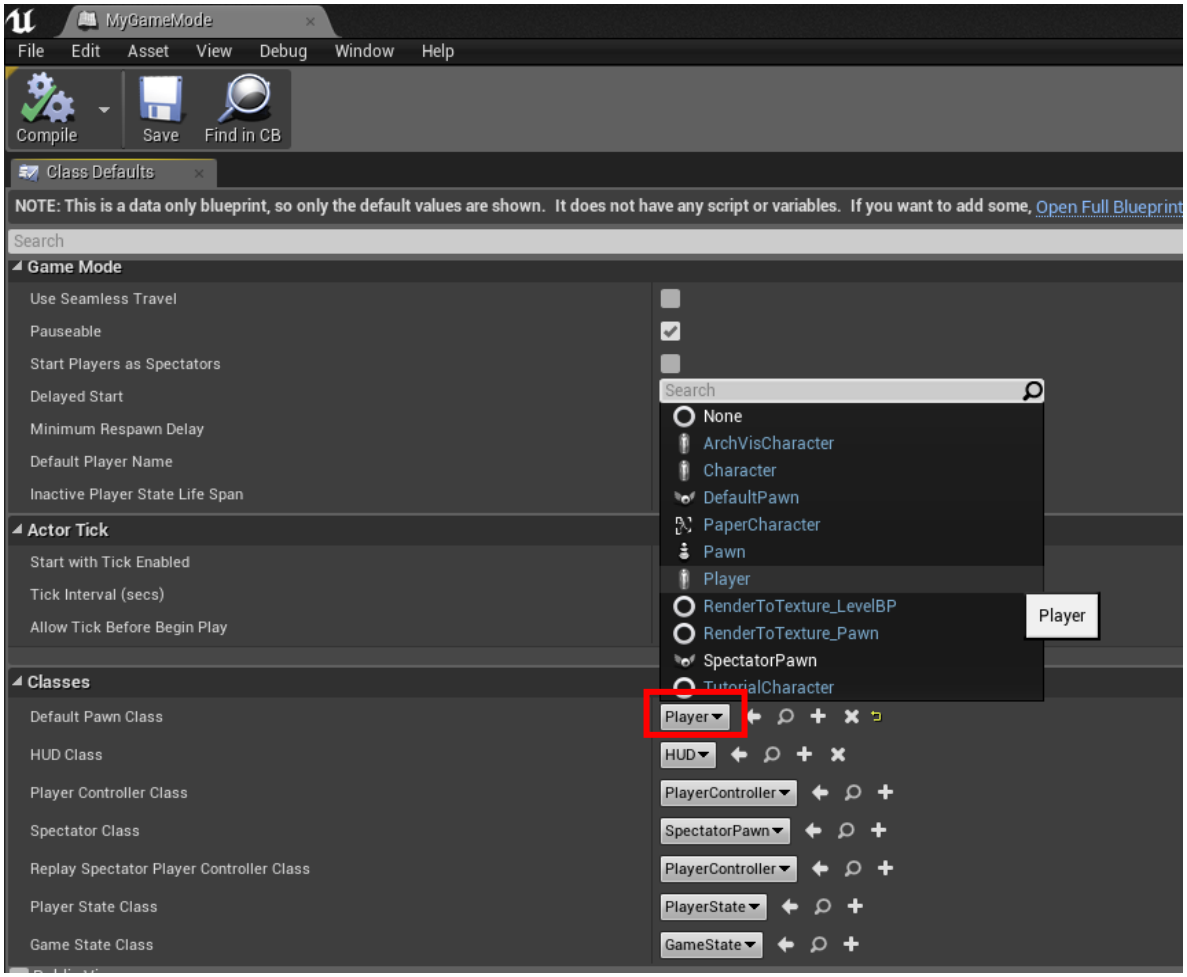
un Peón que tiene la capacidad para moverse alrededor, y un Peón es simplemente un actor que puede ser poseído por algún controlador.

Luego de nombrar nuestra nueva clase Character, debemos crear un modo de juego para hacer que ese personaje (Character) sea el principal del juego. Esto se hace creando una nueva clase Blueprints, pero esta vez seleccionamos en la opción que dice "*Game Mode*", y le ponemos el nombre que deseemos.

Ahora, con los elementos creados, debemos primero decirle al editor que deseamos usar el Modo de juego que creamos recién y no el por defecto que trae el editor. Esto lo hacemos en "*Project Settings*", pero en "*Maps & Modes*". Y debemos seleccionar el modo de juego que creamos antes, como predeterminado, como se indica en la siguiente imagen, en la que el Modo de juego creado se nombró "*MyGameMode*".



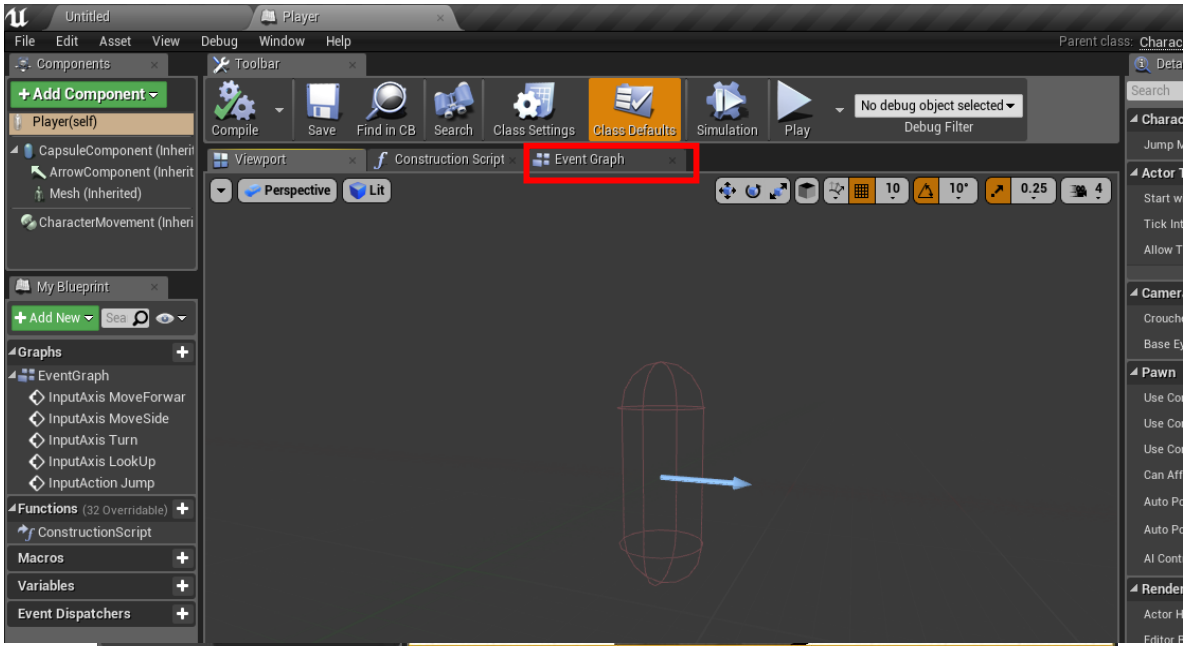
Luego, debemos decirle al modo de juego que acabamos de crear, que el jugador principal es el que creamos antes. Para esto, hacemos doble clic en el Modo de Juego creado, desde el explorador de contenido, y seleccionamos al "*Character*" que creamos. Justo como aparece en la siguiente imagen, donde a la clase se le llamó "*Player*".



Finalmente, en el mapa debemos poner un "*Player Start*" en el lugar que deseemos que el jugador aparezca (al crear un proyecto en blanco, este componente viene por defecto).

c. Movimiento con Blueprints

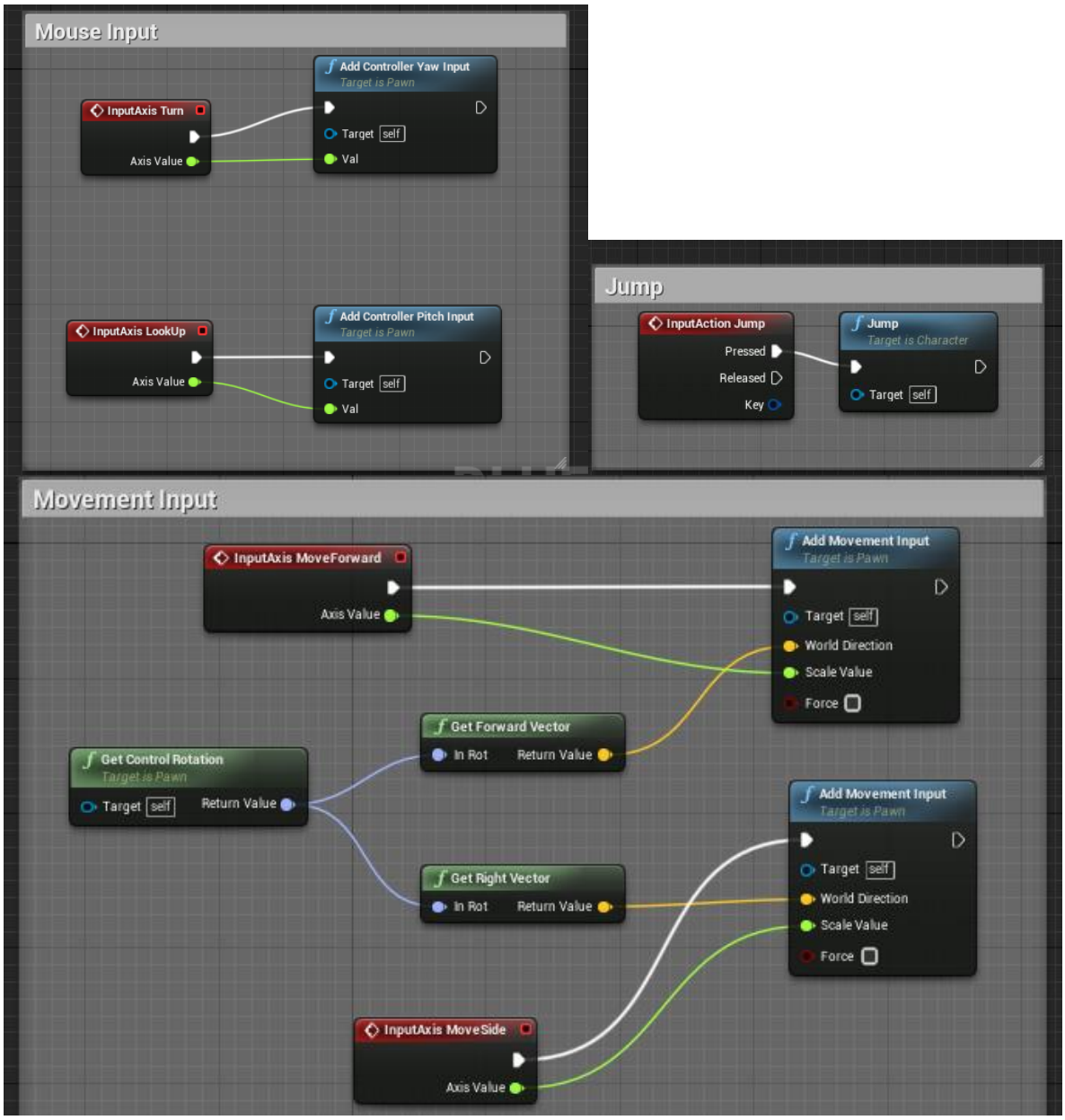
Una vez conectado todos los componentes, debemos programar el movimiento del personaje principal. Esto lo hacemos haciendo doble clic en el personaje principal, desde el explorador de contenido, y en las pestañas superiores, vamos a la que dice "*Event Graph*".



Esta "malla de grafo" es donde programaremos el movimiento del personaje. La forma de utilizarlo (la más simple en mi opinión), es simplemente hacer clic derecho sobre cualquier lugar, y buscar el componente (función, variable, constante, etc.) requerido.

Los eventos que aparecen medios transparentes al principio son: "*BeginPlay*", que es aquel que marca el comienzo del nivel; el evento tic, que es un evento que se ejecuta cada vez que se refresque la pantalla (cada frame), y finalmente el evento de comenzar el sobre posicionamiento, que es utilizado para ejecutarse una vez el personaje intercepte a otro actor en el nivel.

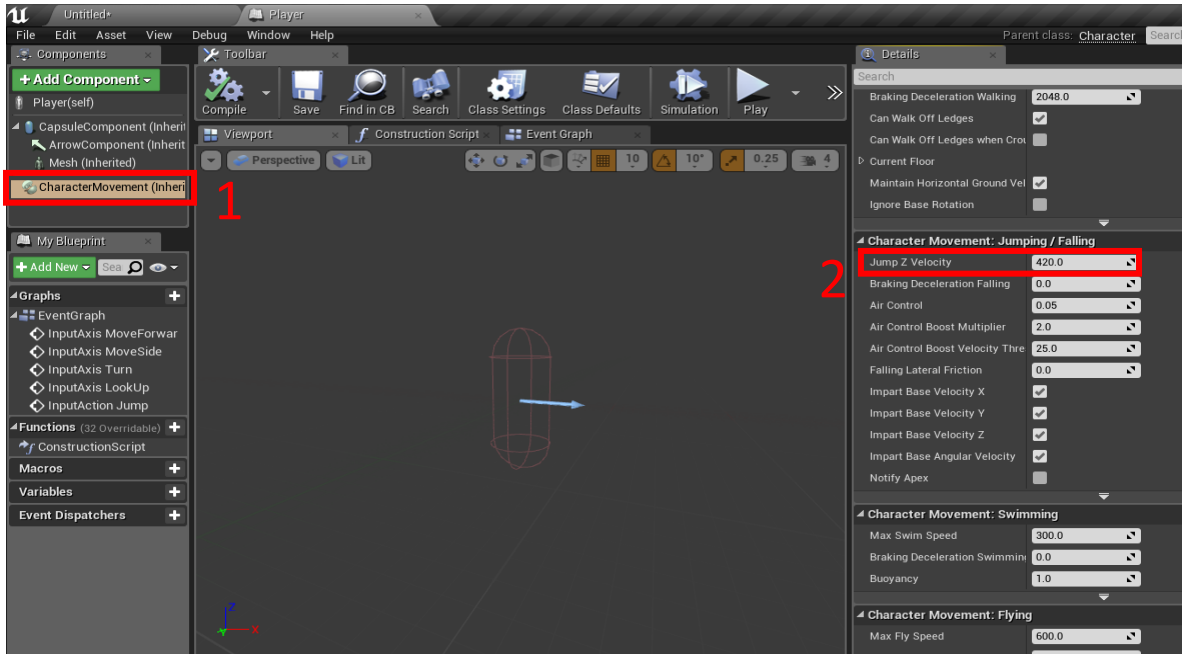
Ahora, en el Blueprint del personaje debemos escribir y conectar los siguientes componentes de la siguiente forma (luego de las imágenes se explicará el funcionamiento del Blueprint).



Como se puede apreciar, el flujo que se ve es bastante intuitivo. Pero básicamente, lo que se dice en el flujo del movimiento (que es el más complejo), es simplemente que cuando exista input del evento "*MoveForward*", se agregue un input de movimiento al personaje (Target = self), en la dirección en que está apuntando el personaje. Esta última se obtiene con la función "*Get Control Rotation*", y del valor retornado por esta obtenemos el vector hacia adelante o hacia el lado (dependiendo del input que reciba).

El color blanco es porque esos componentes están comentados. Esto se puede hacer seleccionando los componentes deseados, presionando botón derecho, y seleccionando comentar ("*Create comment from selection*").

Si se desea modificar el movimiento del personaje, como hacerlo más rápido o que salte más alto, se debe hacer doble clic en el personaje desde el explorador de contenido, y una vez ahí seleccionar el "*Character Movement Component*" del personaje, y las propiedades aparecerán a la derecha de la pantalla. En la siguiente imagen se explica con mayor claridad, y se resalta donde modificar la velocidad de salto o, en efecto, altura a la que llega el personaje con el salto.



d. Animaciones

Si se desea hacer que el personaje tenga cuerpo, y este se mueva, requiere un largo procedimiento, el cual escapa del alcance de este tutorial. Pero puedes obtener más información en la siguiente lista de reproducción de YouTube, que explica de manera muy simple y sencilla las animaciones en Unreal Engine: https://www.youtube.com/playlist?list=PLZlv_N0_O1ga0IoRrpI4xkX4qmCrhGu56 .

5. Audio e Iluminación

El audio y la iluminación son componentes bastante sencillos de incluir en un mapa, en este capítulo describiré un poco en que consiste cada tipo de audio e iluminación, e intentaré ilustrar con algunos ejemplos la diferencia entre los distintos tipos (obviamente esto solo aplica a la iluminación, ya que con el audio no se puede hacer mucho más, sólo se podrá notar habiendo experimentado ya con los diferentes tipos de audio).

a. Audio

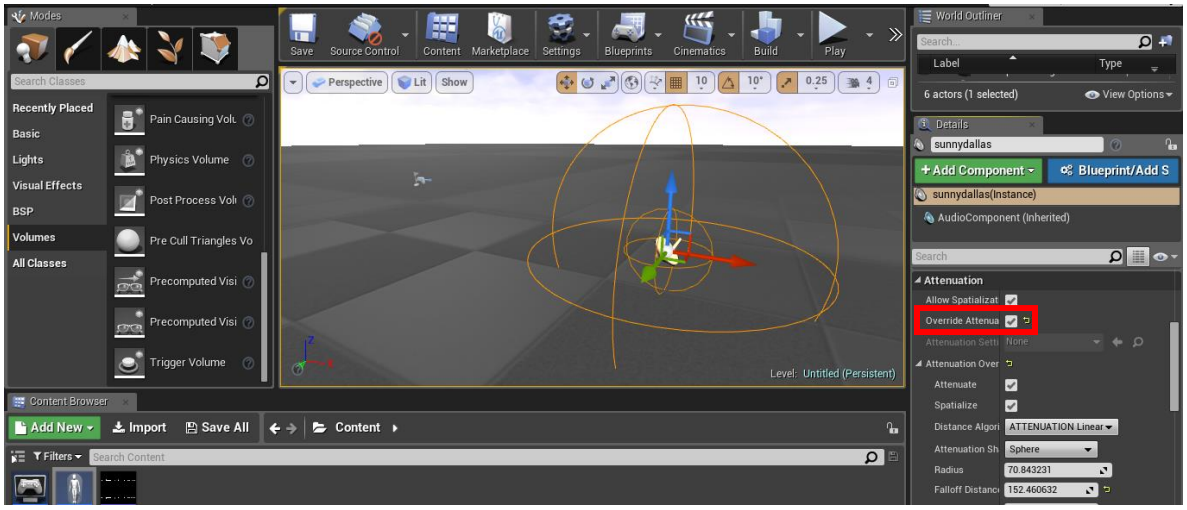
Los dos principales tipos de audio que tocaremos en esta sección, serán los sonidos tridimensionales, y los sonidos sin atenuación.

Un sonido sin atenuación, como el mismo nombre lo explica, es aquel que una vez colocado en el mapa, este se escucha en cualquier lugar. Es decir, se esté muy lejos o muy cerca de la fuente sonora, en ambos casos se escucha con el mismo volumen.

Por otro lado, encontramos los sonidos 3D, o con atenuación, que como se podrá pensar, son aquellos que su volumen e intensidad si cambia con la distancia a la que se encuentre de la fuente.

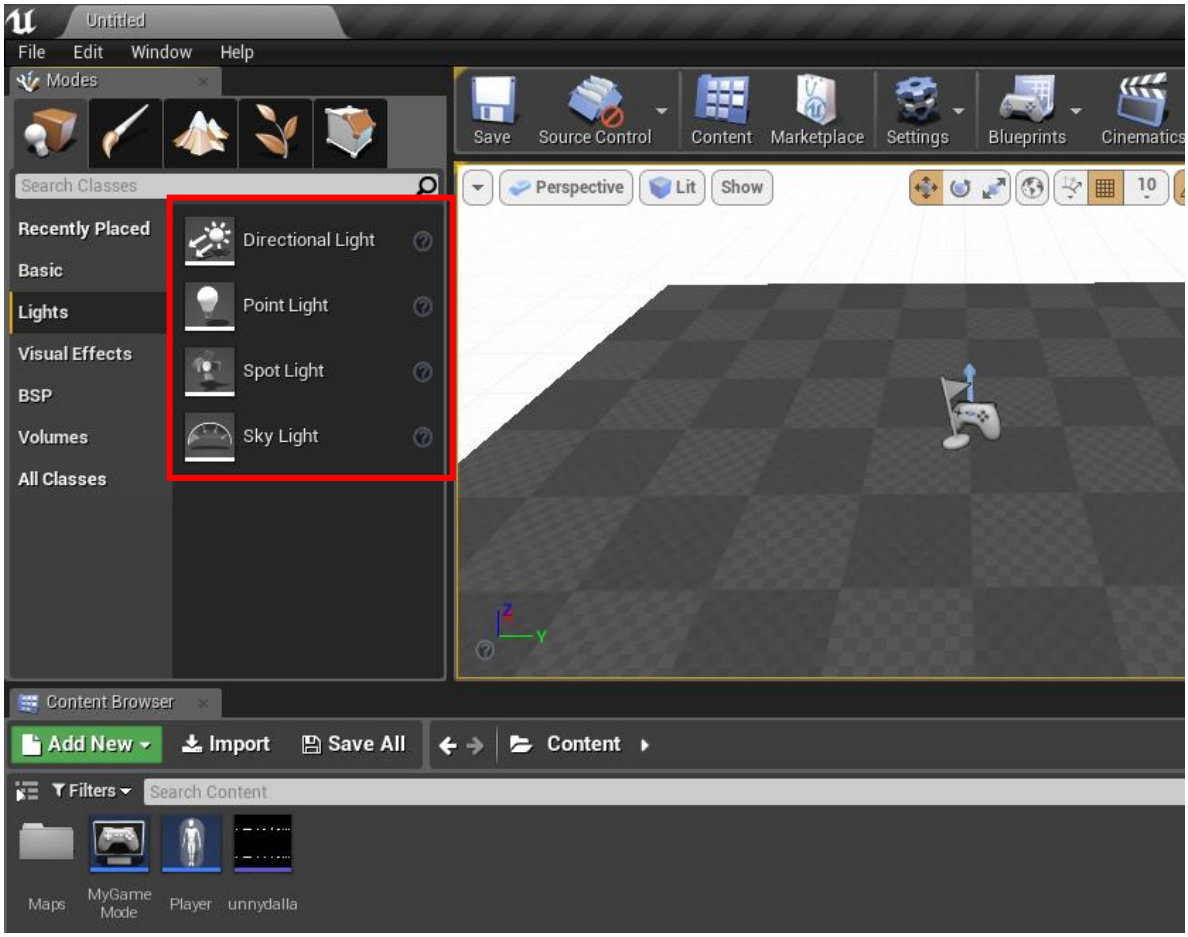
Para utilizar un sonido cualquiera en el editor, primero se debe asegurar que el formato del archivo sea WAV, y luego se debe importar el archivo (arrastrarlo desde el explorador de archivos del sistema, hasta el explorador de contenido del editor).

Una vez importado el archivo en el editor, solo se debe arrastrar desde el explorador de contenido, hasta dentro del mapa y moverlo a la ubicación deseada. Ahora, para hacer que un sonido sea tridimensional o no, debemos modificar un ajuste que se llama "*Override Attenuation*", que se encuentra donde se resalta la imagen. Al seleccionar la caja, se transforma en un sonido 3D, y más abajo se pueden configurar los radios de atenuación. Si la caja no está seleccionada, el sonido se escuchará sin atenuación por todo el nivel.



b. Iluminación

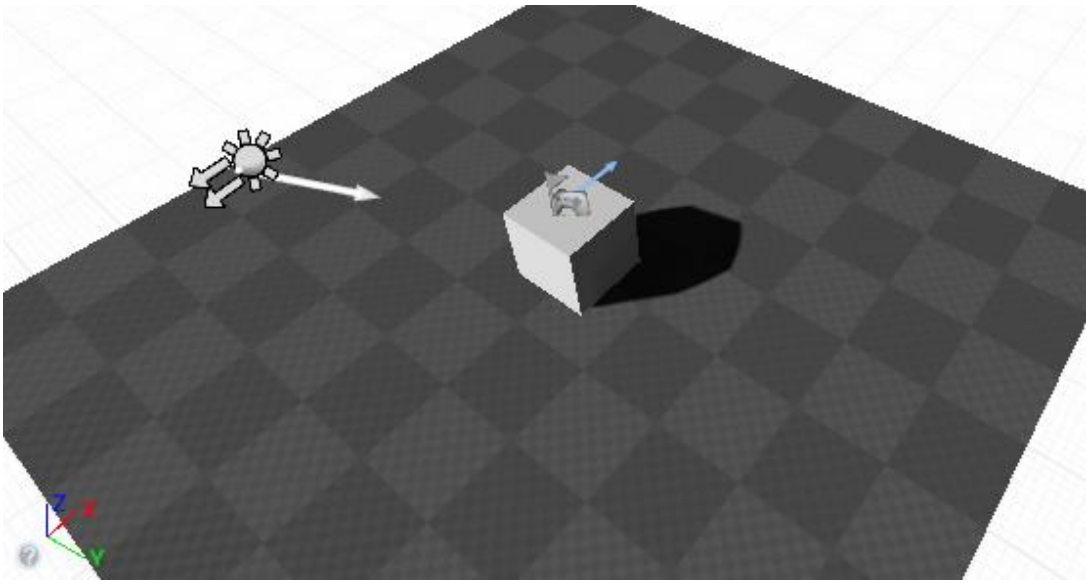
En unreal engine existen 4 tipos de iluminación, los cuales permiten simular las luces de la realidad, dando resultados impresionantes. Cada tipo de iluminación se puede obtener del menú de la izquierda, en la sección "*Ligths*", y sólo basta con arrastrarlas hacia el nivel. En la siguiente imagen se muestra la ubicación de estas.



Para mayor claridad las explicaremos en el mismo orden en que aparecen en el programa.

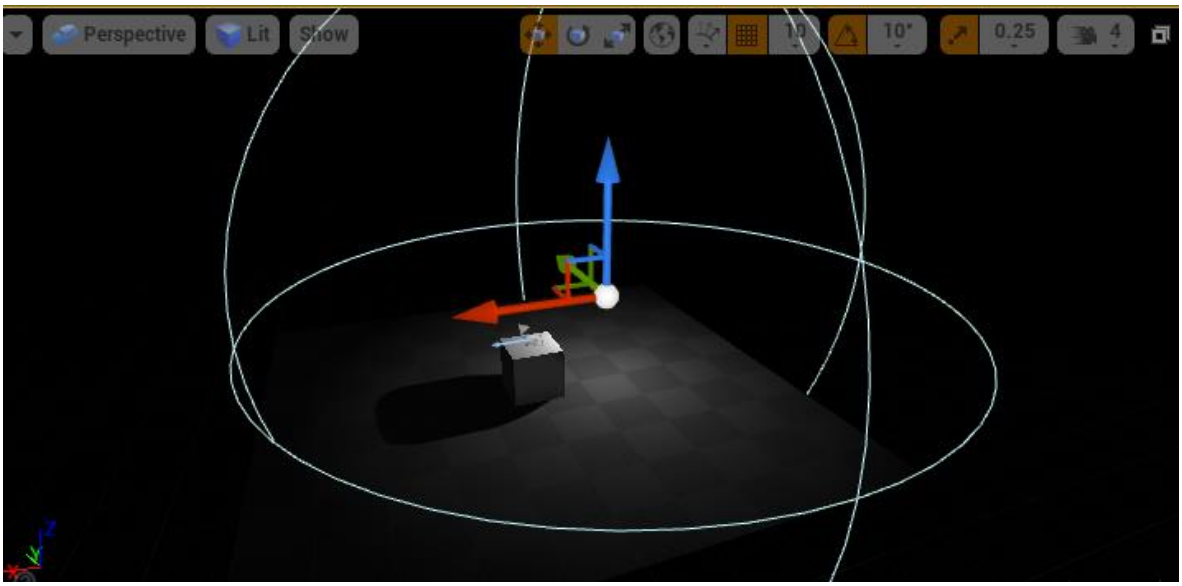
El primer tipo, las luces direccionales, son aquellas que simulan una fuente que se encuentra a una distancia infinita del nivel. Por esto, todas las sombras producto de esta luz, son paralelas a la dirección que se le dé, y el lugar donde se encuentre en el nivel no tiene ninguna importancia, sólo importa el ángulo de incidencia de la luz.

Debido a sus características, esta luz es la perfecta candidata para simular la luz solar. A continuación, se muestra una imagen de ejemplo con luz direccional.



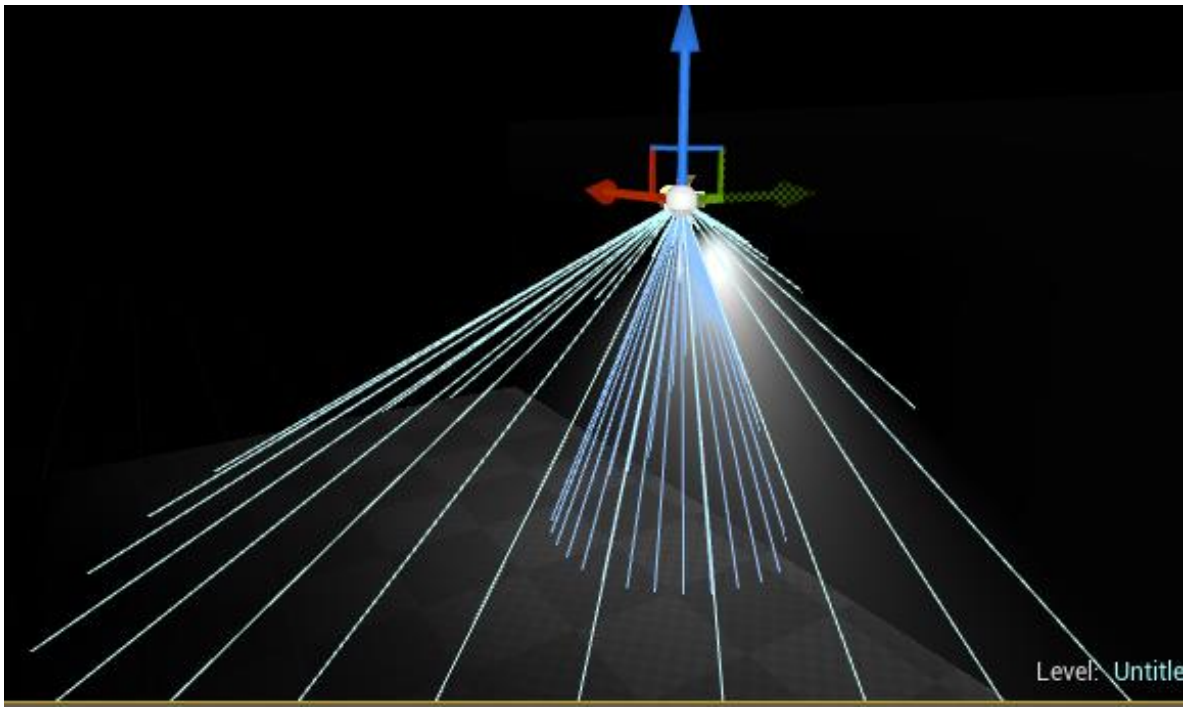
El siguiente tipo de iluminación es la puntual, la cual representa en mayor medida a las ampolletas de la vida real, ya que como su nombre lo dice es puntual, pero además ilumina en todas las direcciones de igual forma. Obviamente tiene un radio en el que es efectiva, el cual se puede modificar en sus propiedades.

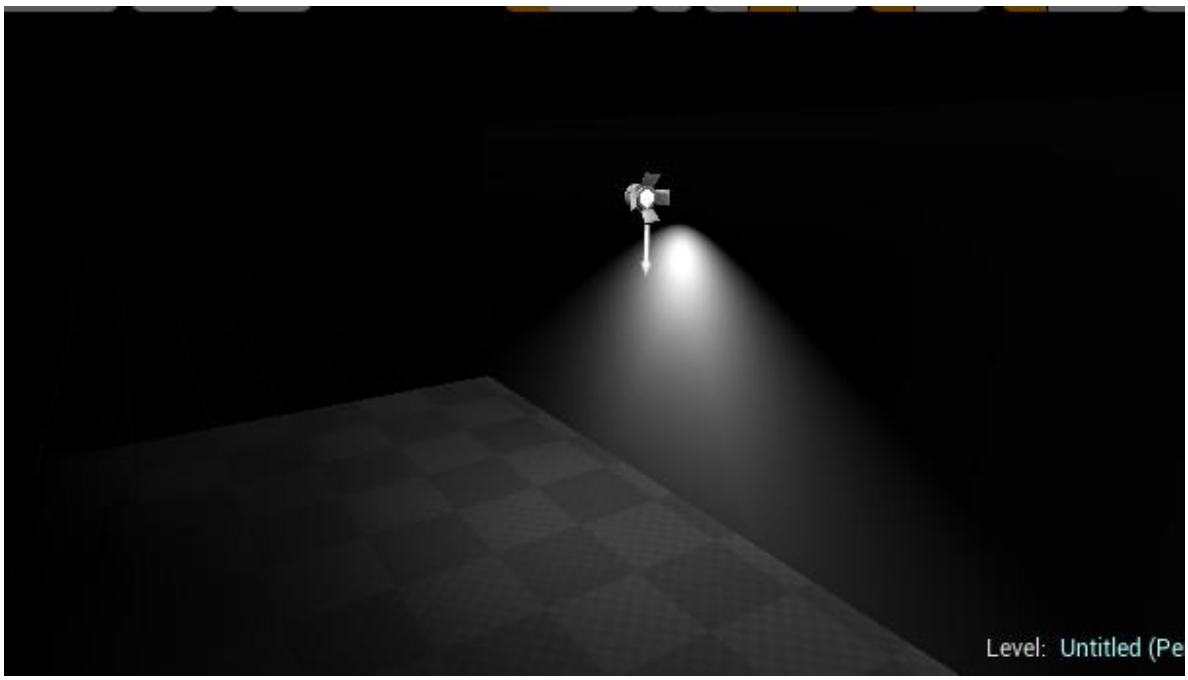
A continuación, se muestra una luz puntual, junto con su radio de efectividad.



Las siguientes luces, son las luces de lugar o "*Spot Lights*", y son aquellas que emiten iluminación en forma de cono, por lo que son las más aptas para representar la luz de una linterna o de un foco de escenario. Para el ajuste de su iluminación, estas tienen el cono exterior y el cono interior. El cono interior, es en el que la luz ilumina completamente; en cambio, en el cono exterior, la luz va iluminando cada vez menos, dejando un estado de penumbra bastante parecido a la realidad.

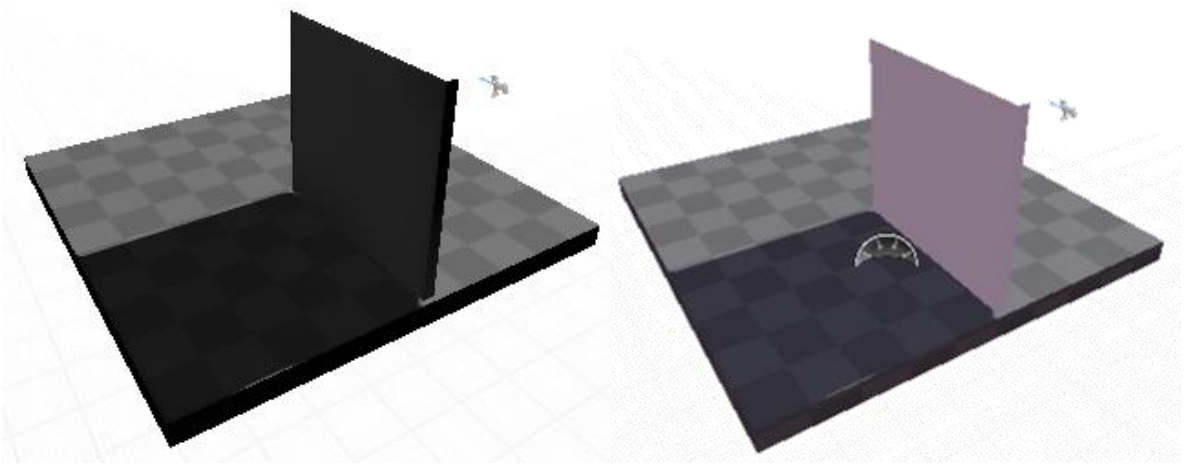
A continuación, se muestra una imagen ejemplificando la luz de lugar. En la de arriba se aprecia el cono interior y exterior, mientras que en la de abajo se muestra el resultado final.





Por último, se encuentra las luces del cielo o "*Sky Lights*", que en pocas palabras se utilizan para iluminar interiores que en la vida real no son tan oscuros. Y para lograr esto lo que hace es capturar luz desde lugares distantes del mapa (todo lo que está más lejano a una distancia definida en sus propiedades), y la aplica a todo el mapa obviamente con menos intensidad. Como su definición es medio complicada, a continuación, se muestra una imagen de un lugar sin "*Sky Lights*", y otra imagen del mismo lugar, pero con "*Sky Lights*".

Se aprecia claramente como la imagen de la izquierda (sin "*Sky Lights*") tiene una sombra mucho más oscura que la imagen de la derecha, que es justamente la que contiene "*Sky Lights*".



Como comentario final, me gustaría agregar que, para poder ver correctamente la iluminación en todo momento, recomiendo altamente reconstruir la iluminación cada vez que se modifiquen los componentes lumínicos de la escena. Esto se puede realizar fácilmente en el menú superior, en "*Build*" → "*Build Lightning Only*". Debo decir que cuando se tiene un nivel de grande tamaño, esta acción toma un tiempo considerable, por lo que solo se debe efectuar cuando se desee ver exactamente cómo quedará el nivel con esa iluminación, luego de haber hecho cambios considerables.

6. Inteligencia Artificial

En este tutorial solo veremos lo más básico de inteligencia artificial, ya que como se ha reiterado en varias ocasiones, y sobre todo este punto en particular, da mucho de qué hablar, por lo que se perdería un poco el enfoque de este tutorial que es simplemente dar una pincelada del editor.

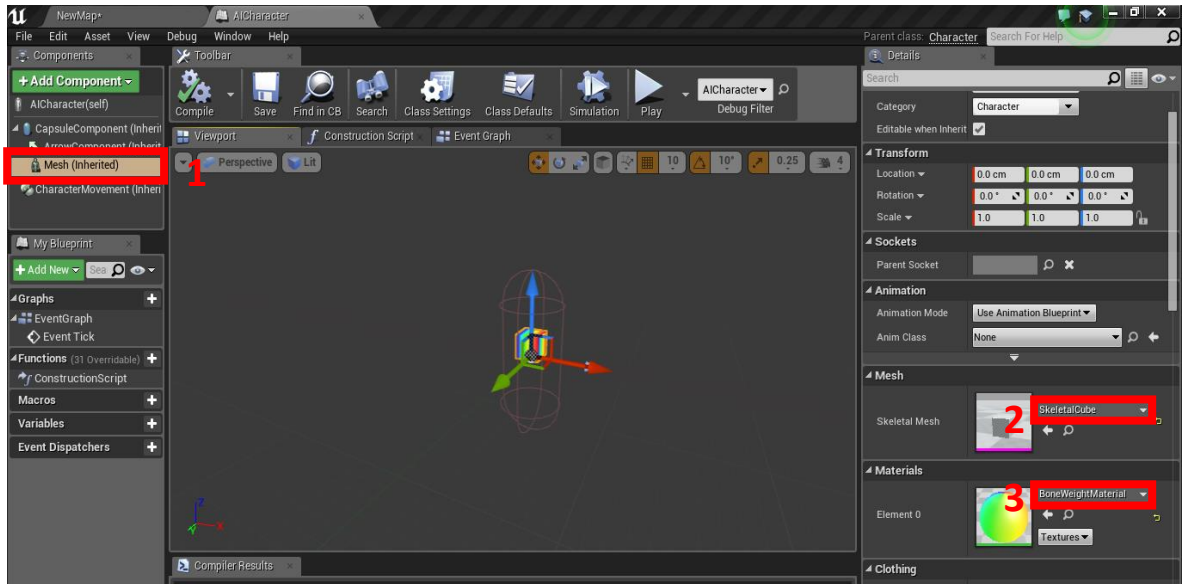
a. Mesh del jugador

Lo primero que debemos preocuparnos es del Mesh del jugador, es decir, el cuerpo del ente al que deseamos otorgarle movimiento de por sí solo.

Para esto debemos crear una nueva clase Blueprint de tipo "*Character*", igual a la que creamos un poco más arriba para explicar el movimiento de un jugador. De hecho, podemos copiar esa misma clase y llamarla de otra manera (además de borrar el Blueprint que maneja su movimiento de acuerdo a los inputs, ya que no los necesitaremos más).

Cuando tengamos nuestro personaje listo, debemos hacer doble clic sobre el objeto, y estaremos en el editor de la clase. Una vez aquí, debemos seleccionar el Mesh del jugador (para poder verlo en el juego). Para esto debemos seleccionar el componente de Mesh del personaje, y saldrán sus propiedades en la parte derecha. Ahí debemos seleccionar el Mesh que deseamos utilizar, y su material.

En la siguiente imagen se ve más claro dónde están las opciones que hay que ir seleccionando.



Para elegir el Mesh, basta con cualquier Mesh que tengan en su proyecto, y de la misma forma para el material. Si no tienen ningún componente, pueden seleccionar la opción "*View options*" → "*Show Engine Content*", que se muestra abajo a la derecha al desplegar el menú para elegir un Mesh o un material. Con esto les aparece una gran cantidad de contenido que pueden usar en su proyecto.

Si desean hacer que la IA tenga las mismas animaciones y el mismo Mesh que su personaje que crearon más arriba (en la lista de reproducción de YouTube que sugerí), solo deben copiar esa clase que crearon, y borrar el contenido del Blueprint que maneja su movimiento a través de inputs.

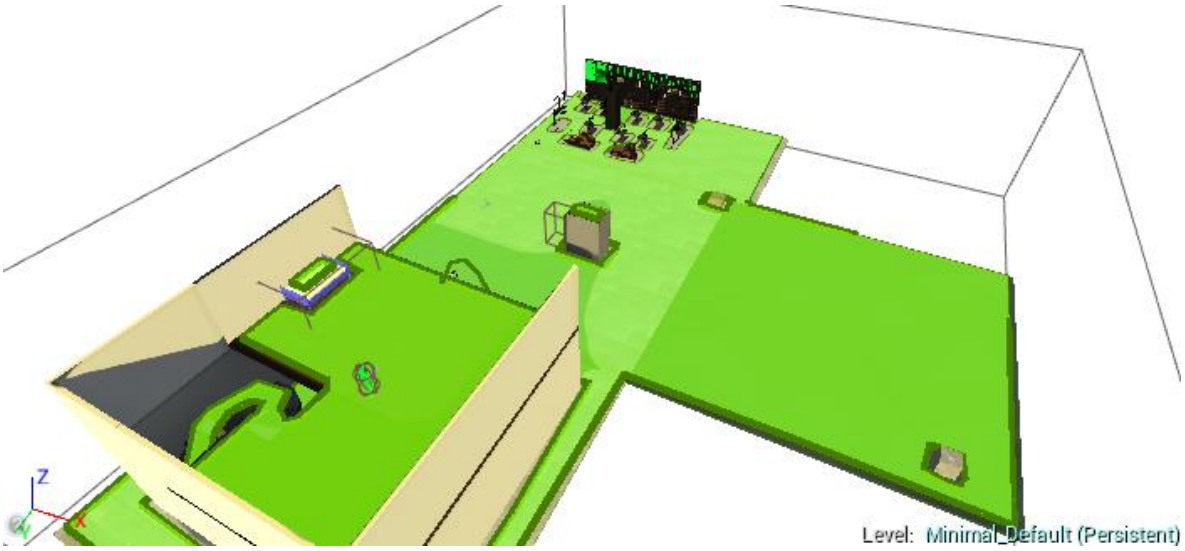
b. Movimiento "inteligente"

Como se dijo anteriormente, sólo se verá un aspecto muy simple de esto, y lo que se hará será hacer que la inteligencia artificial persiga al jugador que es controlado a través de

inputs. También se explicará cómo hacer que la inteligencia pueda patrullar entre varios puntos diferentes del mapa.

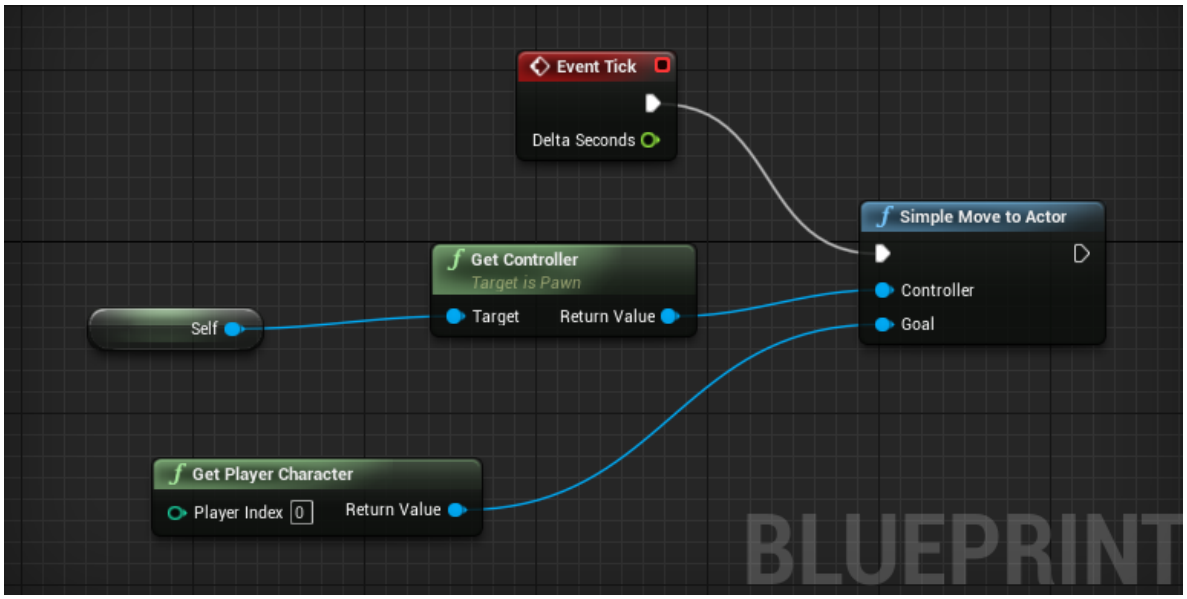
Lo primero es situar la inteligencia artificial en el mapa, esto lo hacemos arrastrándola desde el explorador de contenido hacia el mapa, y ubicándola donde corresponda. Además, debe estar definido el lugar de inicio del jugador (con el objeto "*Player Start*").

Ahora, para definir el espacio donde la inteligencia se puede mover, debemos incluir en el mapa un "Nav Mesh Bounds Volume", que se encuentra en el menú izquierdo. Se puede buscar, o encontrar en la sección de volúmenes. Este lo podemos agrandar, rotar y mover hacia donde deseemos, y este volumen será el que definirá los espacios en los que se puede mover la inteligencia, esto es, todos los lugares que estén dentro de este serán procesados y se analizará si la inteligencia puede pasar por ellos.



En este mapa, el volumen de límites de navegación es la caja que cubre el mapa en su totalidad, y también se puede notar el verde particular que cubre los pisos del mapa. Este verde cubre toda la zona que puede ser recorrida por la inteligencia, de hecho, en la esquina inferior izquierda de la imagen hay una escalera de caracol, y se puede notar que el verde sigue la línea de la escalera, es decir, la inteligencia puede subir y bajar por ahí.

Una vez definido el espacio en el que se moverá la inteligencia, debemos programar el Blueprint que hace que la inteligencia siga efectivamente al jugador principal. Para esto, vamos nuevamente a la configuración de nuestra inteligencia, pero nos vamos a la pestaña de su "*Event Graph*". Una vez en el Blueprint, creamos el flujo que aparece en la siguiente imagen.



Lo que hacemos en este flujo simplemente es decirle al controlador del movimiento de "*Self*" (que en este caso es la inteligencia) que se mueva hacia donde está el jugador de índice 0, que es el jugador que movemos a través de inputs. Esto lo ejecutamos cada vez que el evento "*Event Tick*" sea llamado, esto es permanentemente durante la ejecución del juego, ya que como mencionamos anteriormente en el tutorial, este evento se llama cada vez que la pantalla se refresca (cada frame).

Ahora, existe otra función que se llama "*Simple Move to Location*", que recibe una ubicación en el mapa, y mueve a la inteligencia hacia ese lugar. Además, existen actores que son sólo puntos en el mapa, y se pueden usar fácilmente para hacer que la inteligencia vaya de un lugar a otro, o patrulle en un lugar permanentemente.

Para finalizar esta sección, me gustaría recomendar un tutorial mucho más detallado de inteligencia artificial, que habla acerca del árbol de comportamiento de una inteligencia, que es aquel que determina la acción que efectuará la inteligencia en un momento dado. Ya que lo que expliqué en este tutorial no es ciertamente una inteligencia que "piense" por sí misma.

El tutorial lo pueden ver en el siguiente link: https://www.youtube.com/watch?v=evYE7tfWXUY.

7. Video 360°

Primero debo mencionar que el método que usé para reproducir un video 360, hace que el video se vea con un ligero cambio. El video aparece como si se viera en un espejo, es decir la parte izquierda aparece a la derecha y viceversa, pero no afecta para nada a la visualización general, sólo se nota cuando aparecen palabras al revés.

a. Multimedia necesaria

Lo primero que necesitamos es descargar un video 360. Esto se puede hacer fácilmente con programas que descargan videos de YouTube, y cualquier video 360 de YouTube descargado a través de estos programas servirá.

Una vez descargado el video, debemos colocarlo en la carpeta del proyecto de Unreal Engine, dentro de la carpeta "*content*" (a través del explorador de archivos del sistema operativo), de forma que se vea en el explorador de contenido del editor, una vez puesto ahí.

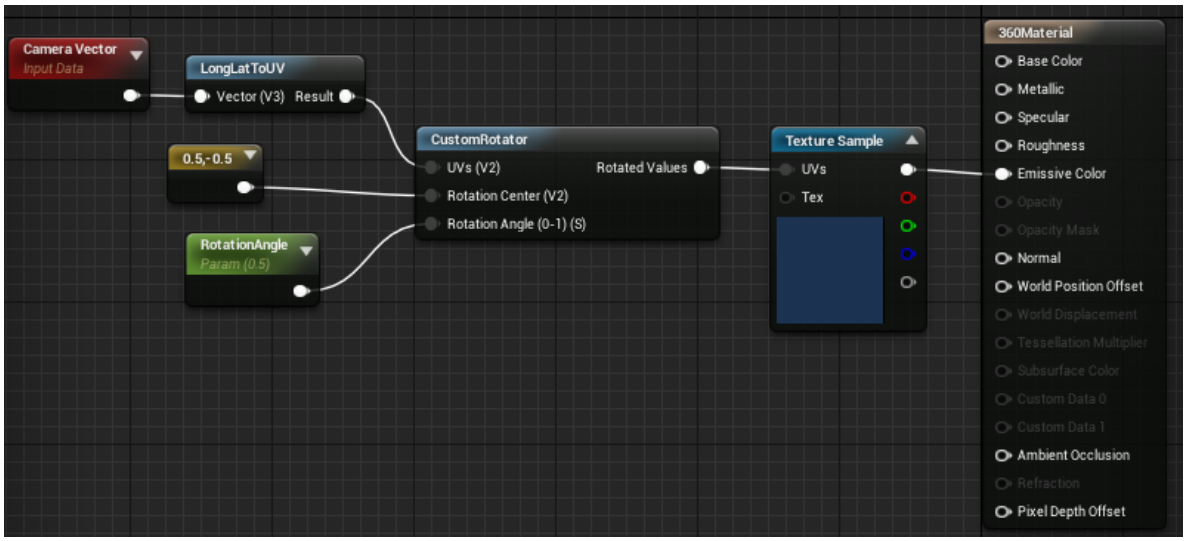
b. Media Texture y Material

Una vez que tengamos el video en la ubicación anterior, debemos hacer clic derecho en él, desde el explorador de contenido, y debemos presionar "*Create Media Texture*" y luego nuevamente en el mismo menú, pero ahora seleccionamos "*Create Media Sound Wave*". Esto generará dos archivos que son el video en sí, y el sonido del video respectivamente.

Luego, debemos hacer clic derecho sobre la textura creada, y presionamos "*Create Material*". Con esto creamos el material donde se reproducirá el video, y este debemos ponerlo en un "*Static Mesh*" que se adecúe a nuestras necesidades, una esfera, pero con las normales invertidas, es decir, que se vea el material desde dentro de la esfera.

Pero antes de poner el material en el Mesh, debemos hacerle unos ajustes al momento de mostrarlo. Estos ajustes son precisamente los que requieren mayor poder de procesamiento, ya que van a acomodar la textura para que ocupe todo el interior de la esfera, es decir, estiraremos el video para que ocupe el interior de la esfera. Para esto, hacemos doble clic sobre el material, y se abrirá lo que parece ser una planilla de Blueprints.

Esta planilla Blueprints es un tanto especial, ya que posee funcionalidades exclusivas de los materiales. Y justamente se usa para crear y modificar un material de la forma que deseemos. A continuación, se muestra la transformación que se debe efectuar al material para su correcta visualización en la esfera.



El componente de color amarillo, se busca con el nombre de "*Constant2Vector*", y el de color verde con el nombre de "*ScalarParameter*". Todos los demás se encuentran con el nombre que tienen (haciendo clic derecho en cualquier lugar del Blueprint, y escribiendo el nombre).

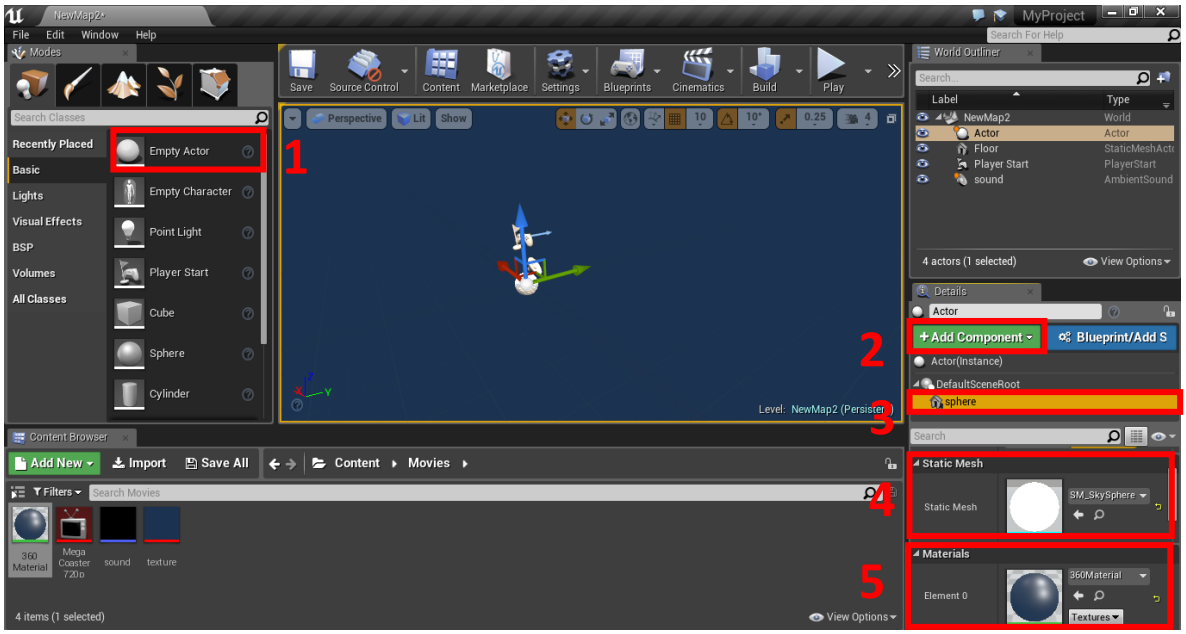
c. Sky Sphere

Lo último que nos falta, es aplicarle el material a una esfera con las normales invertidas, la cual encontraremos fácilmente gracias a la gran cantidad de contenido de apoyo que tiene el editor. Primero, en este mapa, el piso debe ser transparente, para lo que debemos irnos a sus propiedades a la derecha, y activar el ver contenido del editor ("*View Options*" → "*Show Engine Content*"), y seleccionamos el material llamado "*BaseColor*".

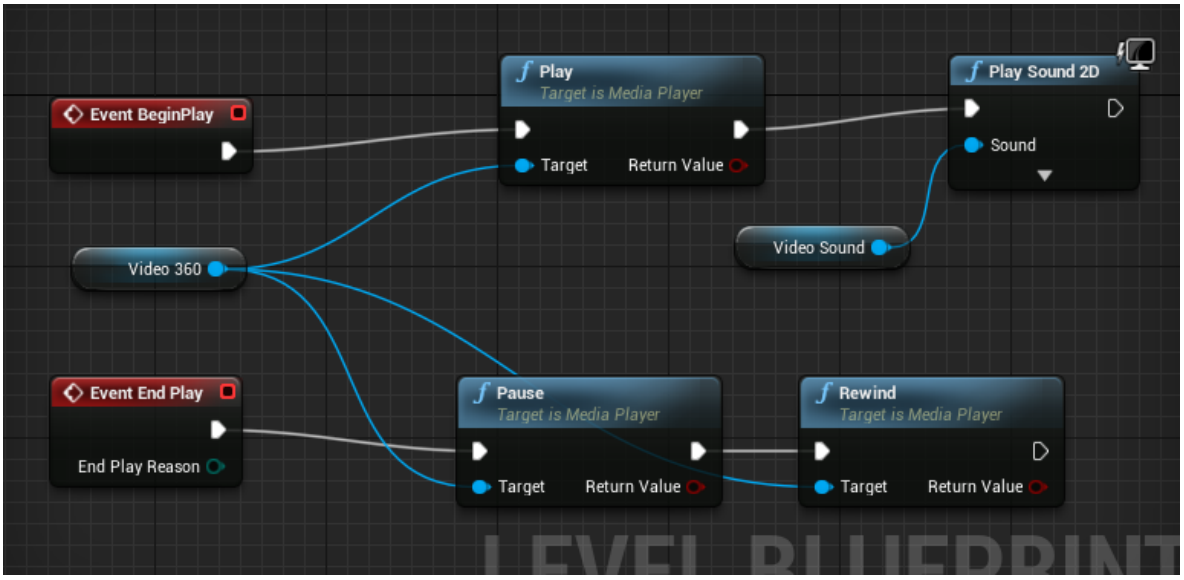
Luego, agregaremos el sonido al mapa, para lo que sólo debemos arrastrarlo desde el explorador de contenido hacia dentro del mapa. Pero también debemos quitar su reproducción automática, la cual se desactiva a través de la opción "*Auto Activate*", en el panel de propiedades abajo a la derecha.

El siguiente paso es agregar la "*SkySphere*" al mapa, para lo que primero debemos agregar un "*Empty Actor*", al que le agregaremos un componente de tipo "*Static Mesh*". Y en sus propiedades, seleccionamos como su Static Mesh uno llamado "*Sky Sphere*" (si no aparece el Static Mesh, debes activar la opción para ver contenido del editor). Y obviamente, seleccionamos como su material, al material que creamos del video 360.

A continuación, se muestra en una imagen la secuencia de pasos a seguir.



Como último paso, para hacer que el video se empiece a reproducir al iniciar el nivel, al igual que su sonido, y ambos detenerse una vez terminado el nivel, debemos entrar al Blueprint del nivel, a través de la opción "*Blueprints*" en el menú superior, y seleccionando la opción "*Open Level Blueprint*". Una vez en el Blueprint del nivel, debemos hacer el flujo que aparece en la siguiente imagen.



El diagrama anterior es muy intuitivo, si lo analizamos paso por paso, vemos que lo que hacemos es que una vez comenzado el juego, con el evento comenzar juego, se comienza a reproducir el video, y luego comienza a reproducirse el sonido.

Y una vez terminado el juego, a través del evento fin del juego, se pausa el video, y se retrocede al inicio. El sonido no es necesario retrocederlo, ya que la función "*Play Sound 2D*" comienza a reproducir el sonido desde el principio.

En este diagrama, los componentes "*Video 360*" y "*Video Sound*" son variables. Las variables se pueden crear en el menú izquierdo, y una vez creadas se les asigna un tipo ("*Media Player*" y "*Media Sound Wave*" respectivamente), y luego de compilar el Blueprint se les puede asignar un valor por defecto, donde se debe elegir el componente que se desea. En este caso fue el video y el sonido del video respectivamente.

8. Comentarios y documentación de apoyo

Para cerrar el tutorial, espero que les haya sido de utilidad para el desarrollo en Unreal Engine. Como comentarios finales, me gustaría agregar que como estas tecnologías son nuevas, y se renuevan cada semana prácticamente, es bueno estar al tanto de las noticias respecto de esto, de las nuevas versiones del editor que van saliendo, y obviamente de las funciones que agregan en cada versión.

Esto es porque, por ejemplo, quizás hoy no se pueda reproducir un video 360 en el Gear VR de la forma en que se explicó en este tutorial. Pero puede ser que en un tiempo más (semanas o meses) sí se pueda, y no será necesario buscar otras formas más rebuscadas de hacerlo. Finalmente, las nuevas versiones se dan justamente para implementar funciones que no están en versiones anteriores y pueden ser de gran utilidad para el desarrollo.

Otra cosa que me gustaría agregar, es que lo que se vio en este tutorial es una parte muy pequeña de todo lo que se puede hacer con Unreal Engine, sólo fue una pincelada de las utilidades más comunes de la plataforma. Y por esto mismo dejaré el link de la documentación original del programa, que contiene todas las posibles funcionalidades de Unreal muy bien explicadas y organizadas: https://docs.unrealengine.com/latest/INT/.

Si tienes cualquier duda respecto del tutorial, o encuentras algún error, me puedes contactar a dacasas@uc.cl para aclarar dudas.