

# Trabajo Unreal (CR7: The Game)



# Índice

<b>Índice</b>	<b>2</b>
<b>Introducción</b>	<b>3</b>
<b>Creación del menú principal</b>	<b>4</b>
<b>Diseño del primer nivel</b>	<b>10</b>
<b>Añadiendo música a nuestro menú principal y a nuestro primer nivel</b>	<b>12</b>
<b>Creación del personaje y aplicación de animaciones básicas</b>	<b>16</b>
<b>Añadiendo más animaciones a nuestro personaje</b>	<b>36</b>
<b>Creación de objetos o pickups</b>	<b>40</b>
Objeto coleccionable “Balon de Oro”	40
Objeto de daño “Coca-Cola”	43
Objeto de curación “Agua”	47
Objeto de velocidad “Ferrari”	51
<b>Creación de HUD</b>	<b>55</b>
<b>Creación del menú GameOver</b>	<b>60</b>
<b>Creación del escenario de juego</b>	<b>65</b>
<b>Creación de enemigos</b>	<b>67</b>
<b>Creación del entorno de fútbol</b>	<b>73</b>
<b>Creación del menú de Éxito y pasó al siguiente nivel</b>	<b>81</b>
<b>Desarrollo del segundo nivel</b>	<b>84</b>
Implantación de un sistema de fuera de juego	85
Implantación de plataformas móviles	89
Incremento de las capacidades de los enemigos	90
Diseño del segundo nivel	91
Menú Final	93
<b>Correcciones</b>	<b>94</b>

# Introducción

En primer lugar, voy a realizar una breve descripción del juego que voy a desarrollar para la práctica entregable de Unreal.

El juego que voy a desarrollar consistirá en un juego de plataforma donde tendremos un personaje con sus correspondientes animaciones y el objetivo de este personaje es avanzar por los distintos niveles que tiene el juego. Este personaje tendrá la textura y animaciones de Cristiano Ronaldo por lo que la temática del juego será futbolera. El objetivo principal del juego es que Cristiano Ronaldo supere las entradas en segada de los enemigos que estén dispuestos por el escenario y conducir el balón hasta introducirlo en la portería, la cual se encuentra en el final del nivel.

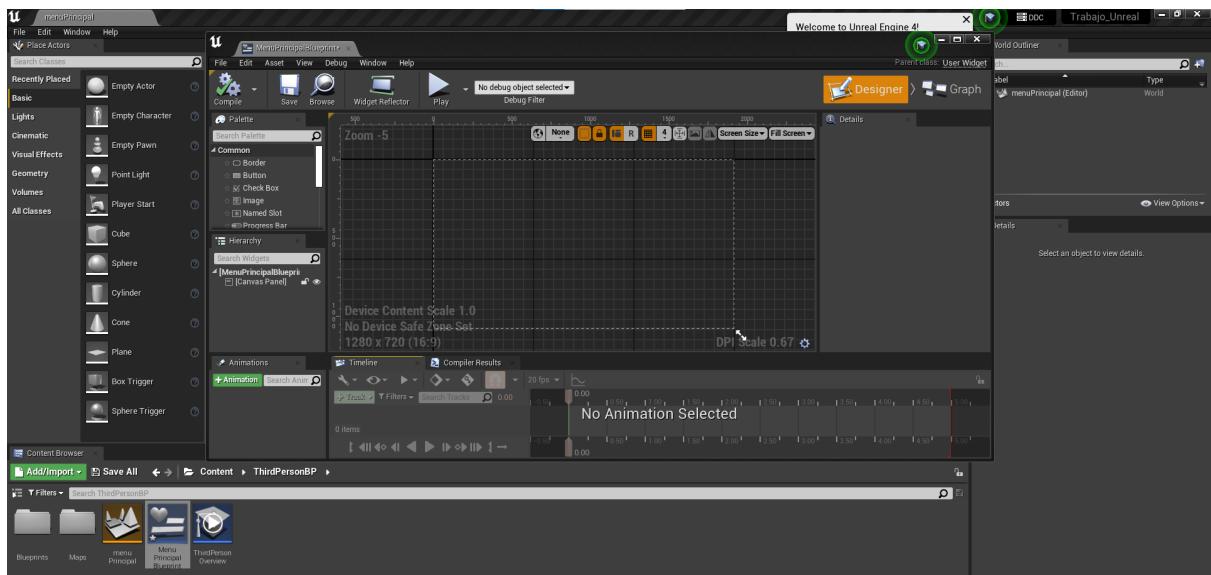
Cada vez que nuestro personaje pierda por causa de algún obstáculo o enemigo, nuestro personaje volverá a aparecer en la plataforma de inicio del nivel correspondiente. De manera adicional, existirán un conjunto de objetos (de cura, de daño, colecciónables, etc) que dotarán al juego de un mayor grado de entretenimiento.

Nota: Aunque el desarrollo de este juego está solamente realizado por mi (el trabajo es individual), a lo largo del todo el documento hablaré en 1º persona del plural para hacer referencia a las acciones que voy realizando a lo largo del desarrollo del juego.

# Creación del menú principal

Una vez hemos creado nuestro juego por defecto en tercera persona, tendremos cargado un nivel básico y un personaje básico. Lo primero que vamos a querer hacer en el desarrollo de este juego es crear un menú principal que se muestre al ejecutar el juego y que nos permita realizar distintas opciones.

Para ello el primer paso es crear un nuevo nivel vacío (click derecho -> nivel) el cual le vamos a dar el nombre de menuPrincipal. Una vez hayamos creado este nivel, nos dirigimos a este nivel que acabamos de crear haciendo doble click sobre él. Despues vamos a crear una interfaz llamada “MenuPrincipalBlueprint” (User interface -> Blueprint Widget) el cual será la interfaz en 2D donde vamos a diseñar el diseño del menú principal.



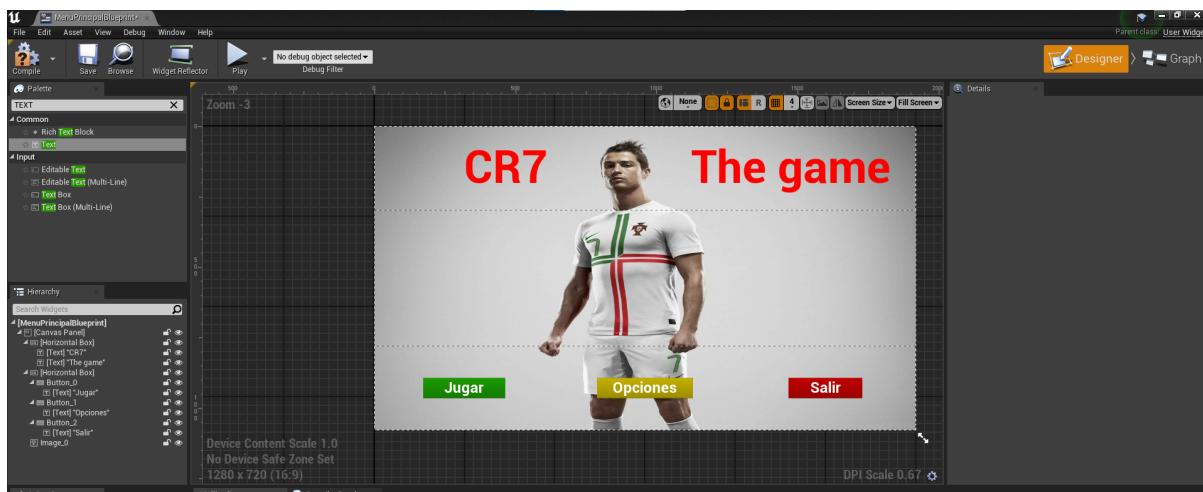
Ahora, para realizar el diseño del menú principal, en primer lugar, añadiremos una Horizontal Box, la cual se colocara en la parte superior de la interfaz y un Texto que contenga el título del Menú Principal el cual se coloque de manera centrada dentro de esta Horizontal Box



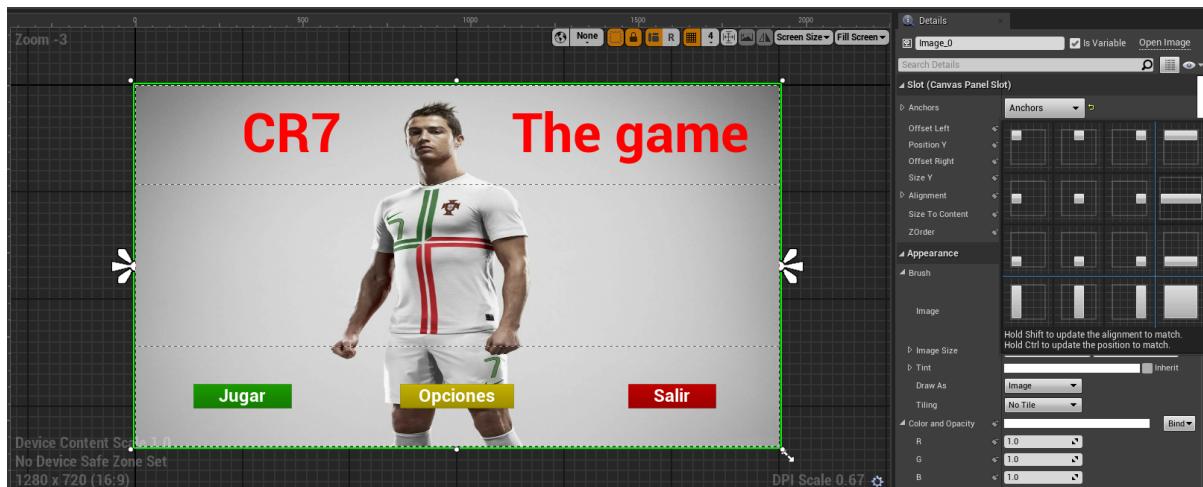
Después, vamos a añadir una Horizontal Box la cual va ocupar la parte inferior de la interfaz y vamos a añadir 3 botones los cuales van a estar contenidos dentro de esta Horizontal Box. No olvidar de añadir un componente Texto a cada botón.



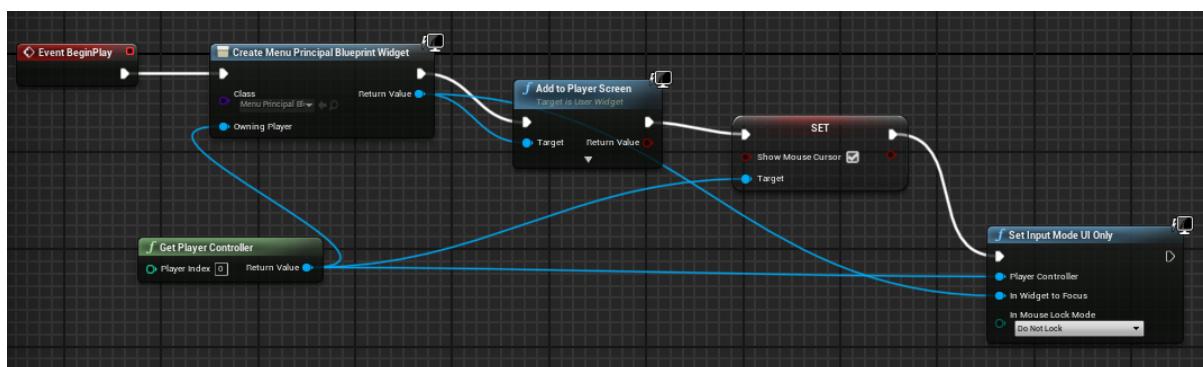
Para concluir con el diseño de este Menú Principal, vamos a añadir una imagen de fondo que ocupe la totalidad de la interfaz. Buscaremos en todo caso que la imagen esté por detrás de los botones, y que la legibilidad de los elementos de la interfaz no se vea alterada por la imagen. Para añadir una imagen a la interfaz, añadimos un componente imagen y a este componente imagen le asociamos la imagen que nosotros queramos pero es imprescindible importar esta imagen al proyecto. Para que la imagen se vea por detrás del texto, cambiamos el parámetro Zorder a -1 para que se coloque detrás de todos los objetos cuyo valor de Zorder es 0. Al final el aspecto final de nuestro menú principal sería el siguiente:



Para que el menú ocupe la totalidad de la pantalla, no olvidar de ajustar los anchors de los objetos de manera adecuada para que los objetos del menú se extiendan por toda la pantalla.

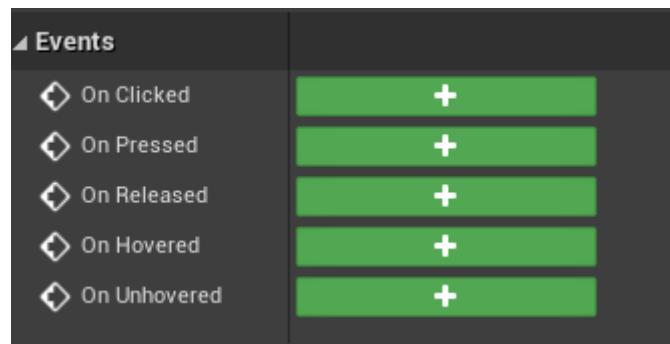


Ahora, el siguiente paso es mostrar esta interfaz del menú principal cuando se abra el nivel MenuPrincipal. Para ello accedemos al Level Blueprint del nivel principal y crearemos el siguiente código:

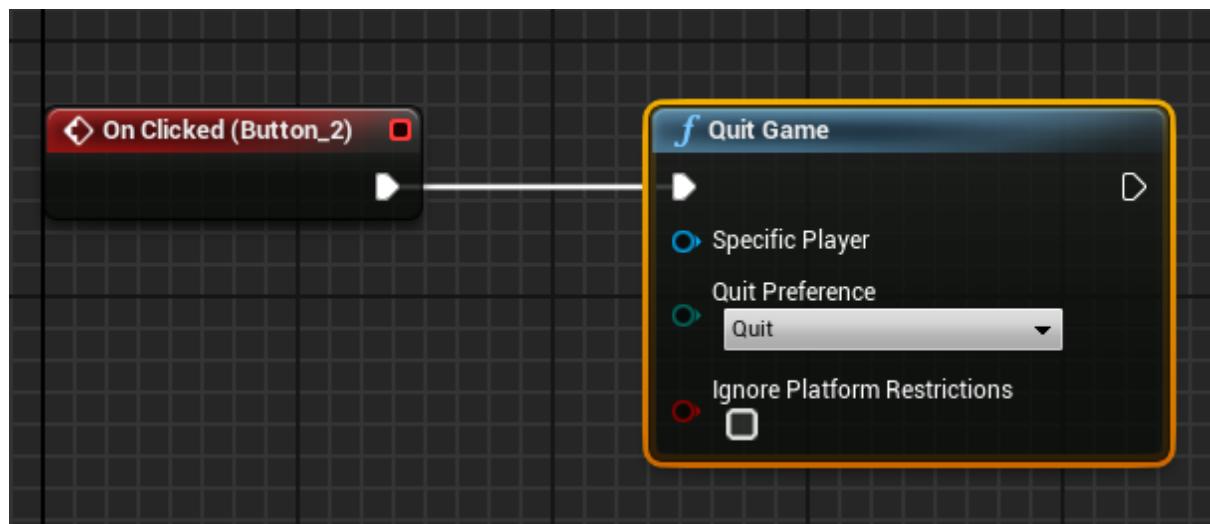


Básicamente con este código, creamos nuestro Widget del menú principal (Create Menu Principal Blueprint Widget) indicando en la clase que widget se desea crear. Después añadimos este widget que hemos creado a la pantalla a través de la acción Add to Player Screen. Después añadimos la acción Set Show Mouse Cursor para mostrar el cursor del ratón en la pantalla cuando se muestre el menú. Por último, configuramos un modo de entrada que permita que solo la interfaz de usuario responda a la entrada del usuario a través de la acción Set Input Mode UI Only.

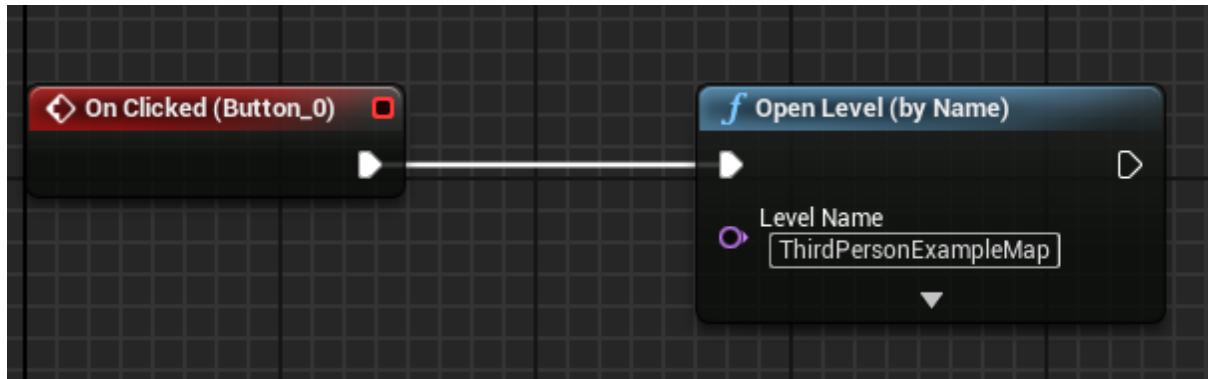
Ahora, vamos a configurar las acciones de los distintos botones para que cuando sean pulsados, realicen la funcionalidad correspondiente. En primer lugar, vamos a configurar la acción del botón Salir. Para ello, nos dirigimos a la interfaz y seleccionamos el botón de salir. En el panel de la derecha, en la sección de Events, pulsaremos en la acción del evento On Clicked y nos saldrá un blueprint en el que podemos configurar las acciones correspondientes cuando el jugador pulse el botón de Salir.



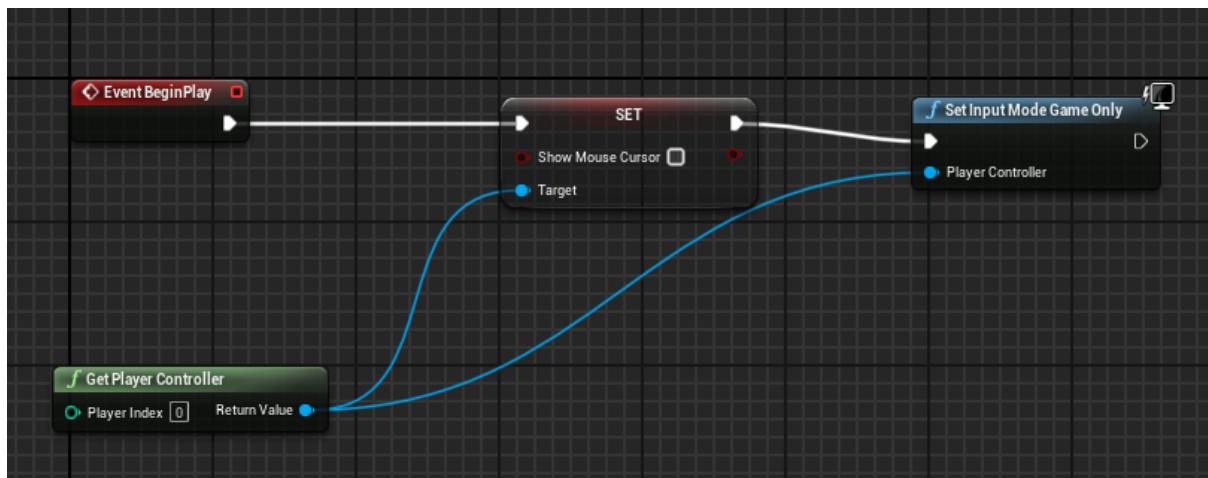
En nuestro caso el blueprint es tan fácil como añadir una acción “Quit Game” cuando se produzca el evento On Clicked en el botón de salir.



Para configurar el botón de Jugar, realizaremos los mismos pasos que en el botón de Salir con la diferencia de que ahora, cuando se produzca el evento On Clicked en el botón de Jugar, añadiremos la acción de abrir el nivel cuyo nombre del nivel es ThirdPersonExampleMap. Cuando configuremos el primer nivel, cambiaremos esta acción para que nos abra nuestro primer nivel del juego.

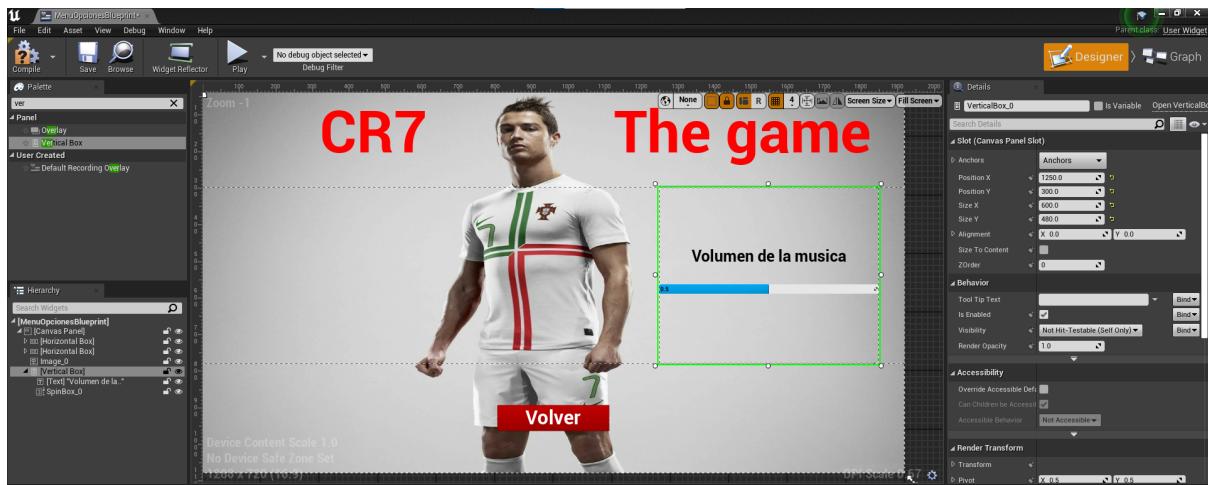


Si ejecutamos el menú principal tal como está y pulsamos al botón de jugar, se abrirá el primer nivel pero no podremos desplazarnos dentro de este nivel. Esto se debe a que para mostrar el menú activamos el input en modo de solo interfaz gráfica. De esta manera, en el blueprint del primer nivel del juego vamos a añadir una pequeña porción del código para cuando se abra el primer nivel, se establezca el input en modo juego.

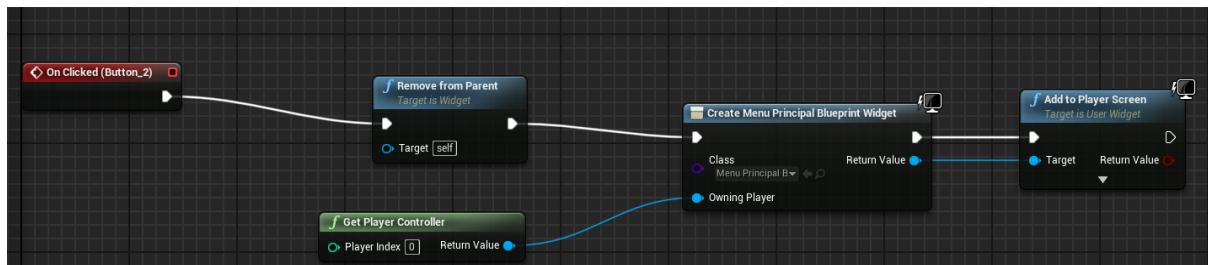


En este código, cuando se ejecute el primer nivel (evento BeginPlay) se ocultará el cursor del ratón con el nodo Set Show Mouse Cursor y se establecerá el modo de input en solo juego a través del nodo Set Input Game Only.

Por último, vamos a configurar el botón de Opciones. Antes de configurarlo, vamos a crear otro Widget Blueprint similar al del Menú Principal pero que nos muestre las opciones del juego. En nuestro caso la interfaz de opciones estará formada por un botón para volver a la pantalla anterior y un slider para ajustar el volumen del juego que más adelante configuraremos.

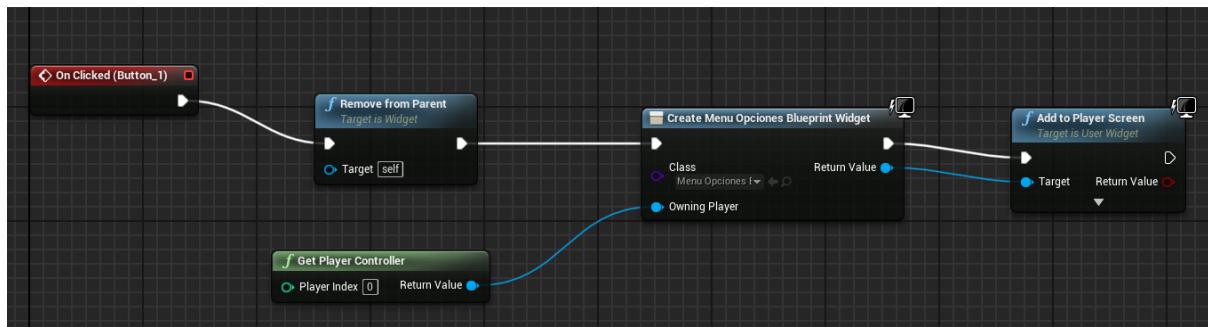


Como hemos dicho, este menú va a estar formado por un solo botón el cual nos va a permitir volver a la pantalla anterior, que será el menú principal. Para ello, registramos el evento On Clicked del botón realizando la siguiente secuencia de acciones:



Básicamente, cuando el usuario pulse el botón de salir, se quitara el widget del menú de opciones como widget principal (Remove from Parent) y se establecerá el widget del menú principal como widget principal y después añadiremos este widget del menú principal a la pantalla.

Una vez hemos hecho y configurado de manera básica el menú de opciones, volvemos al menú principal y establecemos el comportamiento del botón de opciones de manera similar a como hemos configurado el botón de salir. La secuencia de acciones será la siguiente:



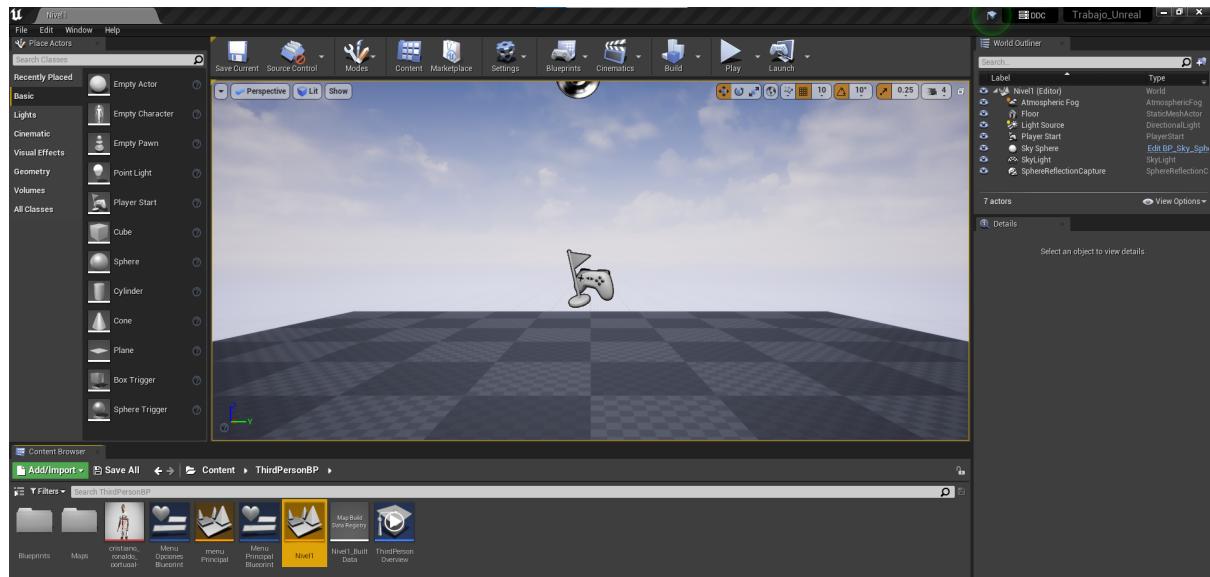
Básicamente, cuando el usuario pulse el botón de opciones, se quitara el widget del menú principal como widget principal (Remove from Parent) y se establecerá el widget de

opciones como widget principal y después añadiremos este widget de opciones a la pantalla.

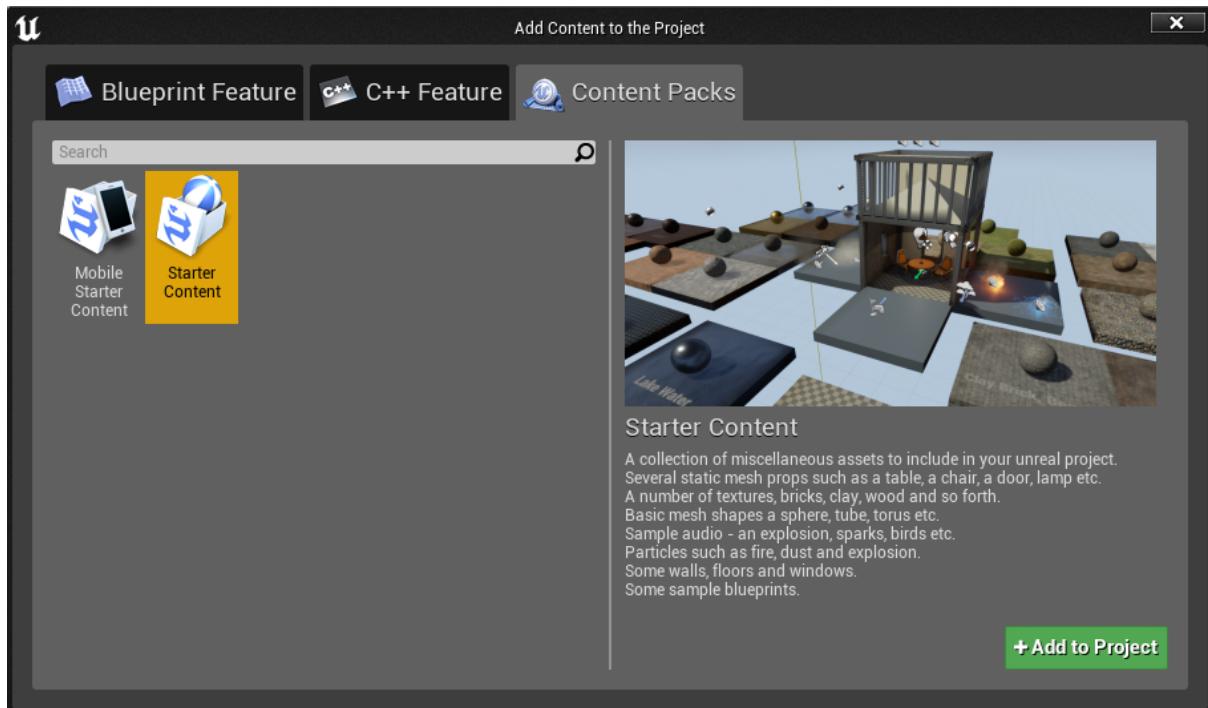
Con esto tendríamos la funcionalidad básica del menú principal.

## Diseño del primer nivel

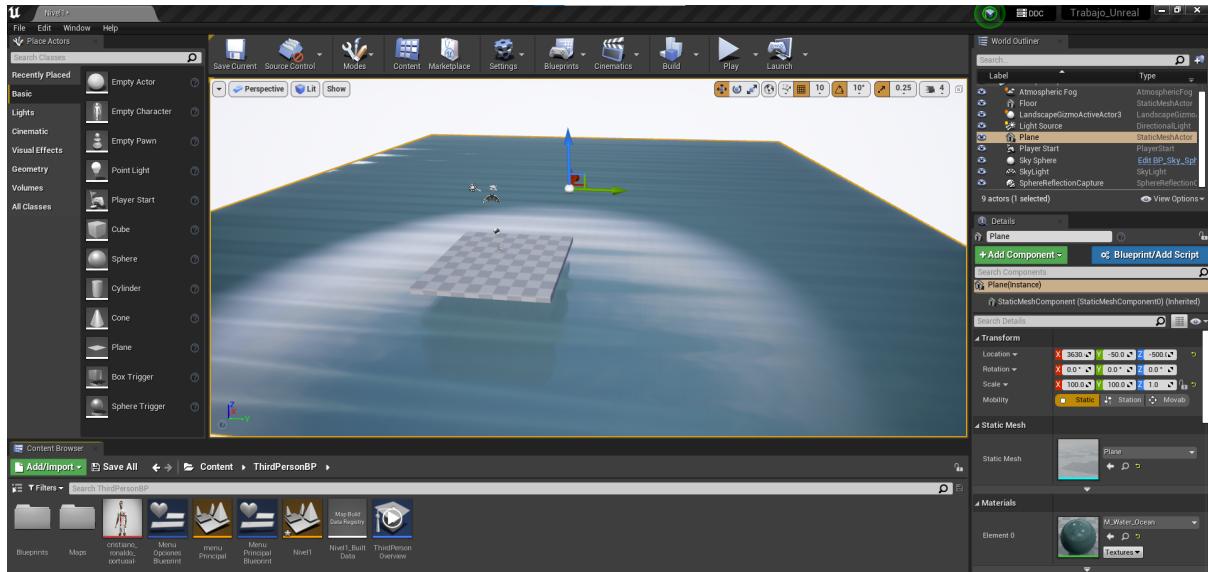
Una vez hemos diseñado el menú principal, el siguiente paso es diseñar el primer nivel del juego, al cual se va a acceder cuando el usuario pulse jugar en el menú principal. Para ello iremos a la pestaña File y la opción New Level y crearemos un nuevo nivel por defecto. Nos quedará un nivel con el siguiente aspecto:



Para que el diseño de nuestra nivel tenga unas texturas interesantes, vamos a añadir un paquete de texturas. Para ello, en el Content Browser (ventana de la parte inferior) pulsamos el botón de Add y seleccionamos la opción de Add Feature Or Content Pack. Se nos desplegará una ventana donde podremos añadir distintos packs. Nosotros vamos a añadir el Starter Pack el cual cuenta entre otras cosas con una serie de Texturas básicas.



En primer lugar, vamos a crear una superficie de agua que ocupe todo el nivel. Para ello, vamos a crear un plano, el cual sea bastante grande y en este plano en el componente Materials, le vamos a asociar una textura de agua de las que vienen en el Starter Pack. En nuestro caso, utilizaremos la textura M\_Water\_Ocean.



Una vez tenemos esto ya tenemos una superficie de pruebas donde probar el personaje. Ahora el siguiente paso sería configurar el personaje y su movimiento. Cuando ya tengamos lista la funcionalidad del personaje se seguirá con el desarrollo de la infraestructura del nivel.

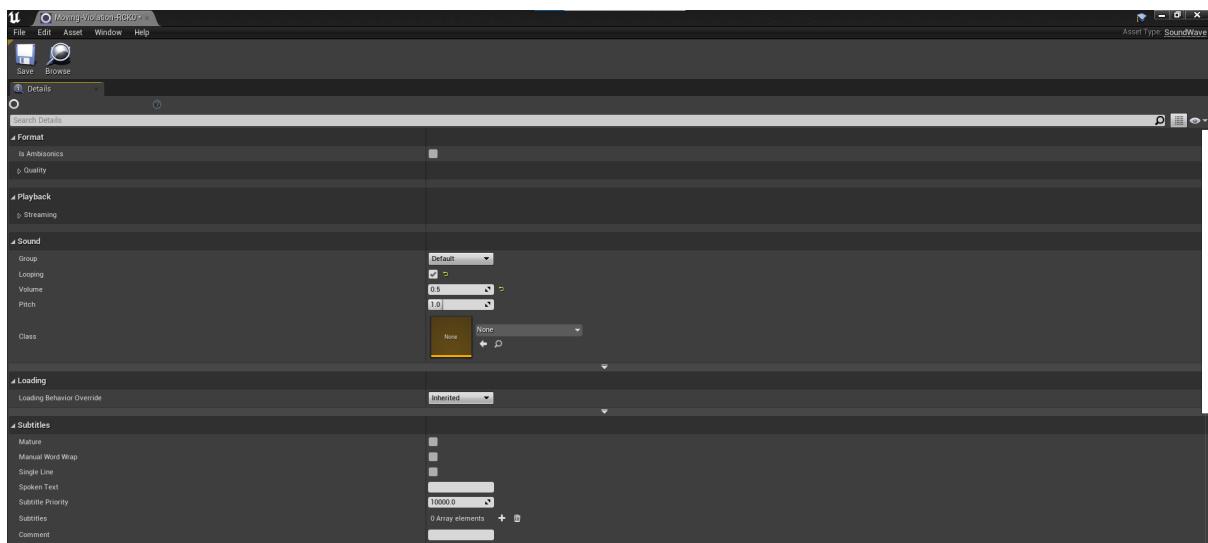
# Añadiendo música a nuestro menú principal y a nuestro primer nivel

Ahora, vamos a añadir música al menú principal y vamos a ajustar el slider del menú de opciones para que a través de este slider se pueda controlar el volumen de la música del juego.

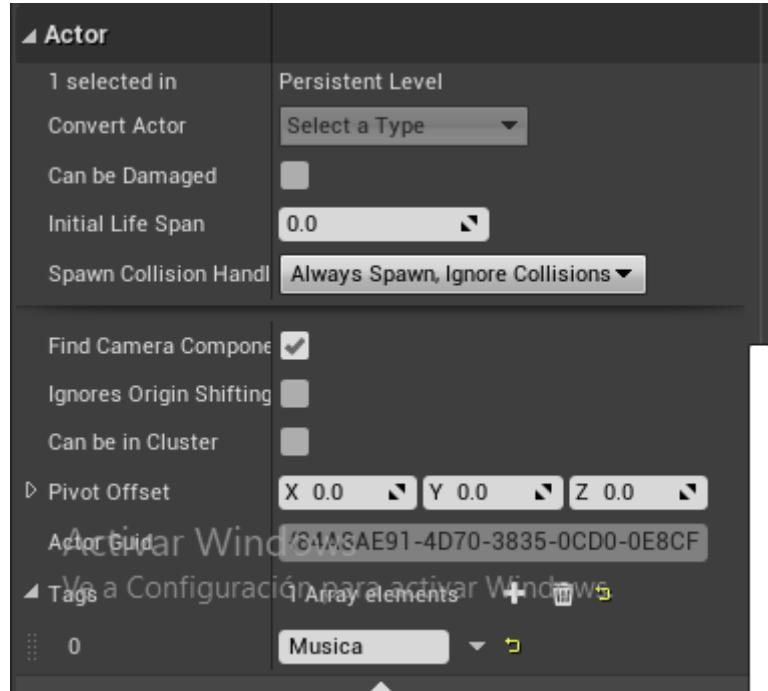
Para implementar esta funcionalidad nos hemos ayudado del siguiente tutorial ya que no hemos usado los slider en clase y nos sabemos bien cómo utilizarlos

[https://www.youtube.com/watch?v=6-GR5YxBrA4&ab\\_channel=GomVoDeveloper](https://www.youtube.com/watch?v=6-GR5YxBrA4&ab_channel=GomVoDeveloper)

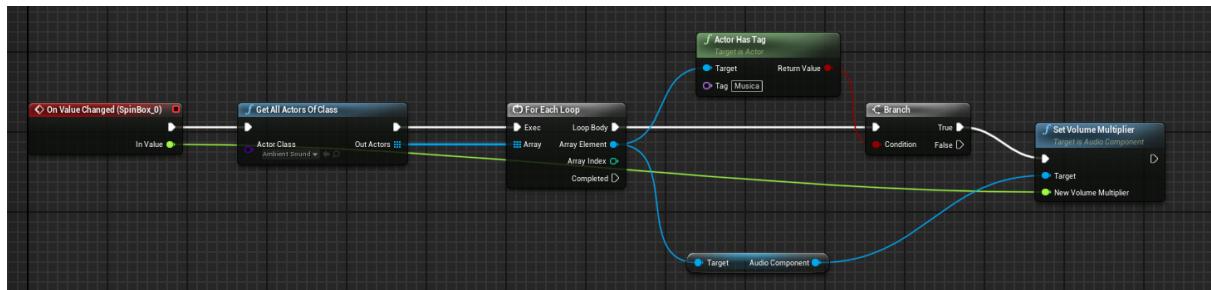
En primer lugar, importamos a nuestro proyecto un fichero de tipo .wav con la música y arrastramos este audio al nivel del menú principal para que se escuche nuestra canción, cuando se ejecute el menú principal. Cambiaremos las propiedades de la música para que se reproduzca la canción en bucle (activar parámetro looping) y cambiaremos el volumen de la música a 0.5 (valor por defecto).



En segundo lugar, seleccionamos la música que hemos introducido en el nivel MenuPrincipal y dentro del componente Actor añadimos una etiqueta llamada “Música” para controlar únicamente el volumen de los ficheros de audio que tenga esta etiqueta de “Música”.



En tercer lugar, vamos a establecer un conjunto de acciones que se van a ejecutar cuando cambie el valor del slider. Esta secuencia de acciones evidentemente irá dirigida a cambiar el volumen de la música. Para ello, registramos un evento llamado On Value Changed en el slider del menú de opciones. La secuencia de acciones necesaria para cambiar el volumen de la música cuando se produzca este evento es el siguiente:

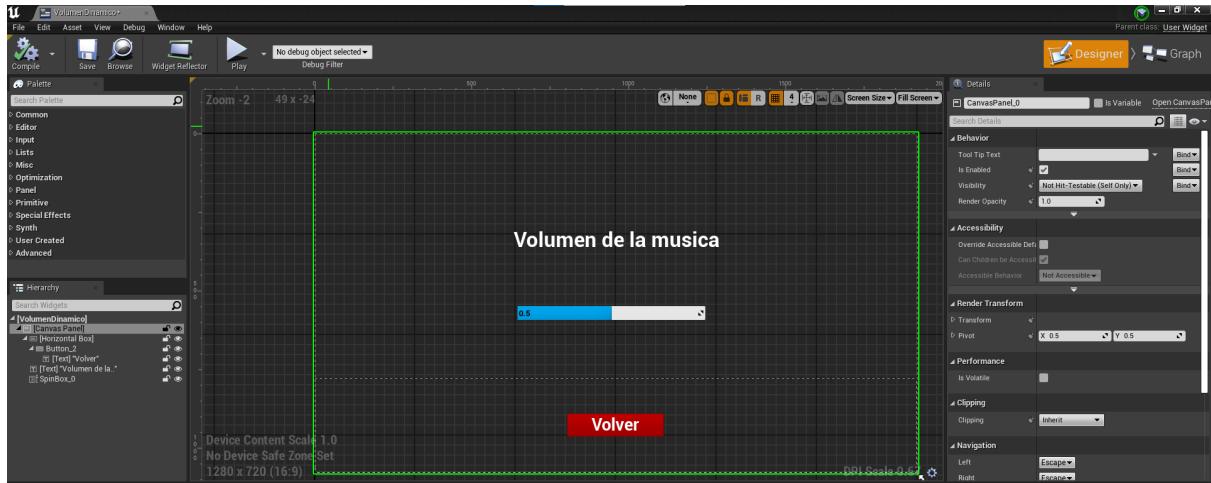


En este código, cuando el valor del slider cambie (On Value Changed), se obtendrá una referencia a todos los actores de tipo música de ambiente a través del nodo Get All Actors of Class y con los nodos For Each Loop y Actor Has Tag seleccionamos solo los actores música de ambiente que tenga la etiqueta de música. Por último, con Set Volume Multiplier se ajusta el valor del volumen de la música en función del valor registrado por este slider.

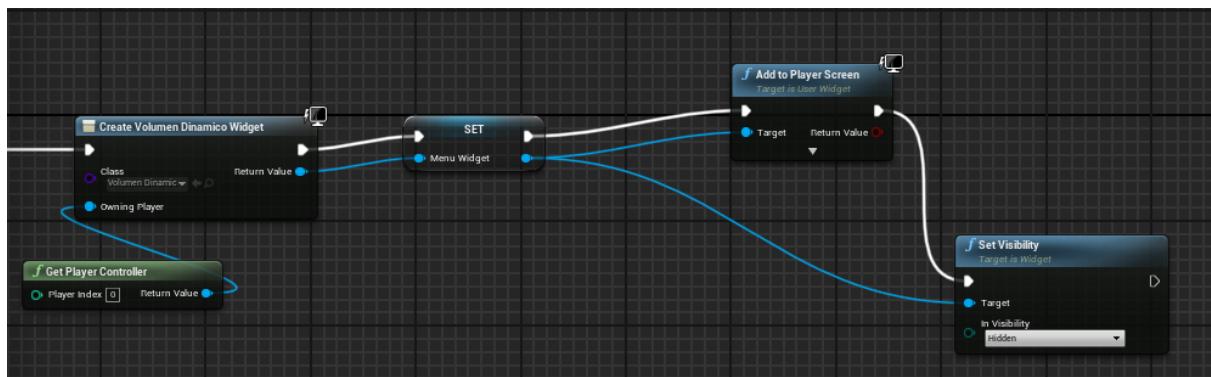
Ahora, nos vamos a dirigir al Nivel1 del juego para configurar la música del primer nivel del juego así como un menú rápido el cual nos permite realizar ajustes sobre el volumen del juego mientras estemos ejecutando el nivel. Para ello, lo primero que haremos evidentemente será importar la música del primer nivel y configurar esta música con el parámetro looping activado y el volumen a 0.5 (igual que con la música del menú).

En segundo lugar, crearemos un Blueprint Widget parecido al menú de opciones pero eliminando la foto de fondo y centrando el slider ya que va a ser un menú rápido y dinámico

que va usar el personaje cuando este en medio del nivel. La interfaz tendrá el siguiente aspecto:

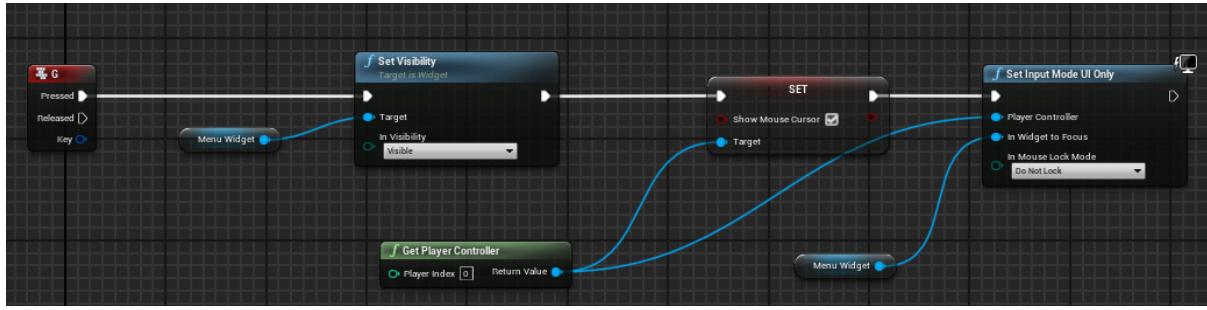


Una vez hecho esto, nos dirigimos al Blueprint del personaje y vamos a crear el widget de este menú dinámico que nos va a permitir el cambio de volumen. Dentro del evento Begin Play de nuestro personaje vamos a añadir la secuencia de acciones necesaria para crear el widget del menú de opciones dinámico y ocultarlo en primera instancia. La secuencia de acciones es la siguiente:



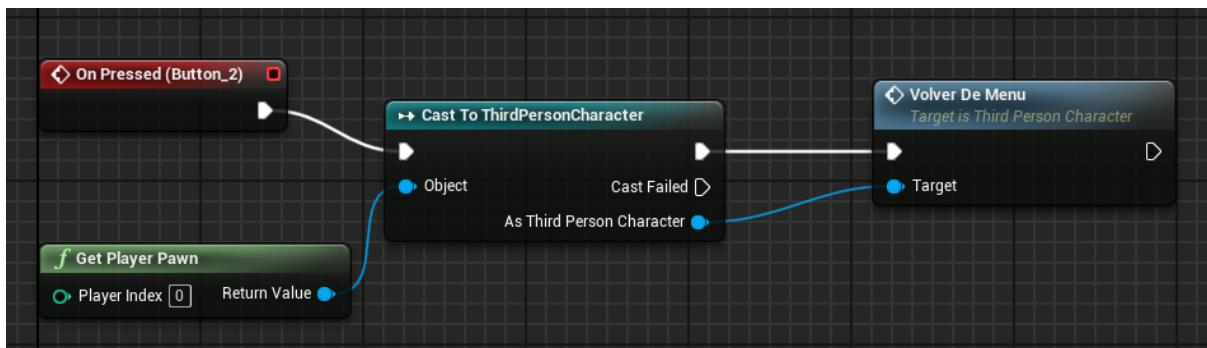
Básicamente, en este código, cuando se produzca el evento BeginPlay y se cargue el nivel, se creará el widget del menú de volumen dinámico (Create Volumen Dinámico Widget) y el almacenará en una variable llamada MenuWidget. Después, con el nodo Add To Player Screen se muestra en la pantalla y con el nodo Set Visibility, ocultamos el menú en primera instancia pues no queremos que se vea al empezar el nivel.

Después, vamos a añadir una porción de código para mostrar el menú de volumen cuando el Jugador pulse una tecla determinada.



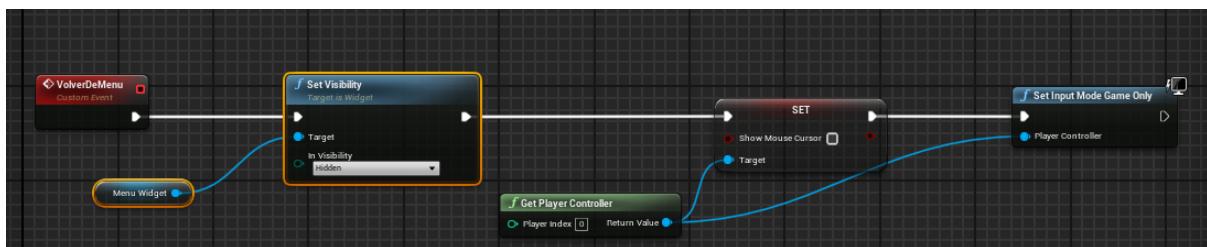
Como se puede observar, en este código cuando se pulsa la tecla G (posteriormente se va a cambiar por la tecla M pues es una tecla más intuitiva para poner el menú) hace visible el menú con el nodo Set Visibility. También se hace visible el cursor del ratón con el nodo Set Show Mouse Cursor y se establece el modo de input a solo interfaz gráfica.

Una vez hecho esto, vamos a configurar el botón de salir del menú de volumen para que nos permita volver al juego. Para ello, nos vamos al menú de volumen y registramos el evento On Pressed (similar a On Clicked) para el botón de salir. Una vez hecho esto incluimos el siguiente código:



Básicamente, en este código, se obtiene una referencia al personaje (Cast To ThirdPersonCharacter y Get Player Pawn) y una vez se obtiene la referencia, se llama al evento Volver de Menú de las acciones necesarias para volver de este menú.

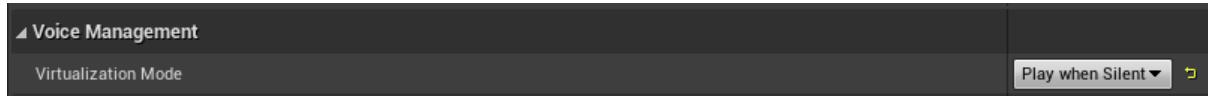
Por último, nos dirigimos de nuevo al blueprint del personaje y registramos el evento customizado Volver de Menú con el objetivo de añadir el código necesario para regresar de este menú.



De esta manera, cuando se pulsa el botón de salir en el menú de volumen, se activa este evento de forma que se vuelve a ocultar la visibilidad del menú con el nodo Set Visibility.

También, se vuelve a ocultar el cursor del ratón con Set Show Mouse Cursor y se establece el modo de input en solo juego.

Como último detalle, debemos habilitar la opción play in silent tanto en la música del menú principal como en la música del juego ya que de no hacerlo, cuando quitemos el volumen de la música (valor del slider a 0) y luego lo volvamos a poner se reproducirá desde el principio de nuevo. Si habilitamos esta opción, la música seguirá sonando aunque esté en silencio.



## Creación del personaje y aplicación de animaciones básicas

Uno de los puntos que se pide para el desarrollo de este proyecto es la creación y animación de un personaje propio. Debido a esto, hemos decidido crear un personaje 3D de Cristiano Ronaldo a partir de una imagen en 2D.

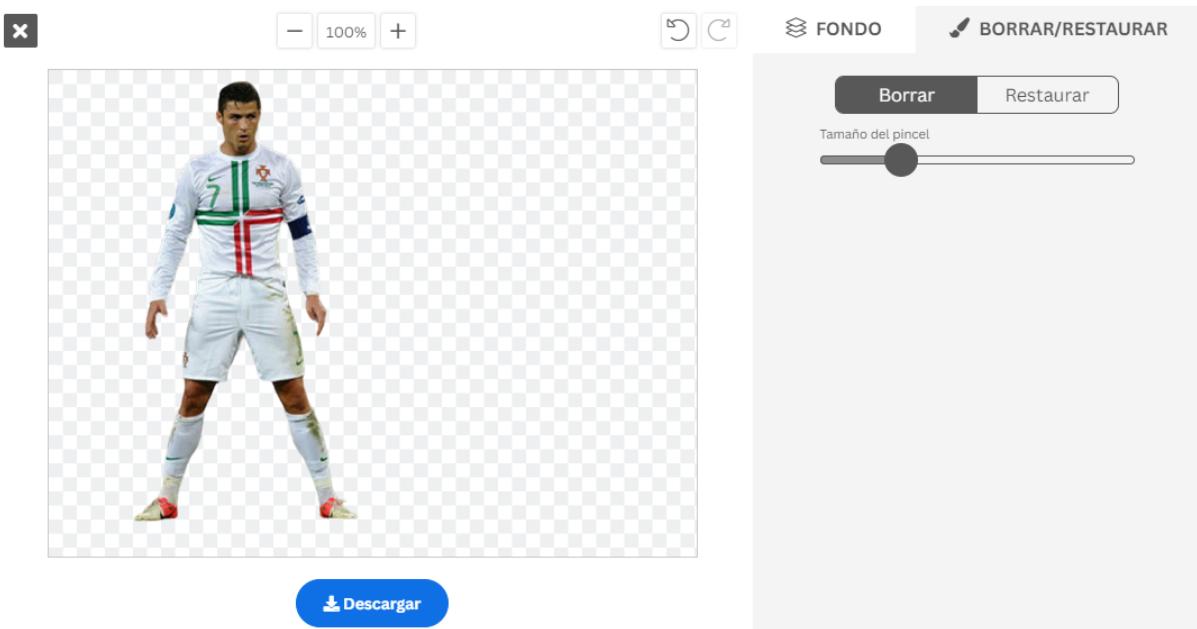
Para obtener un modelo 3D de Cristiano Ronaldo a partir de una foto en 2D lo primero que vamos a hacer es buscar un foto de Cristiano Ronaldo en la que parezca en T-Pose o A-Pose (posición del cuerpo en forma de T o de A). En nuestro caso, hemos encontrado una imagen de Cristiano Ronaldo antes de chutar una falta en la Eurocopa de 2012 donde se encuentra en A-Pose lo cual nos viene muy bien para generar el modelo.



Una vez hemos descargado la foto de Cristiano Ronaldo en A-Pose el segundo paso es eliminar el fondo de la imagen y solo quedarnos con el cuerpo de Cristiano Ronaldo. Para ello, hemos utilizado una herramienta de recorte de fondos dinámica llamada remove.bg cuyo enlace se adjunta a continuacion:

<https://www.remove.bg/>

Tras realizar el recorte del cuerpo de Cristiano Ronaldo, nos descargamos la foto recortada y la guardamos al lado de la foto original.

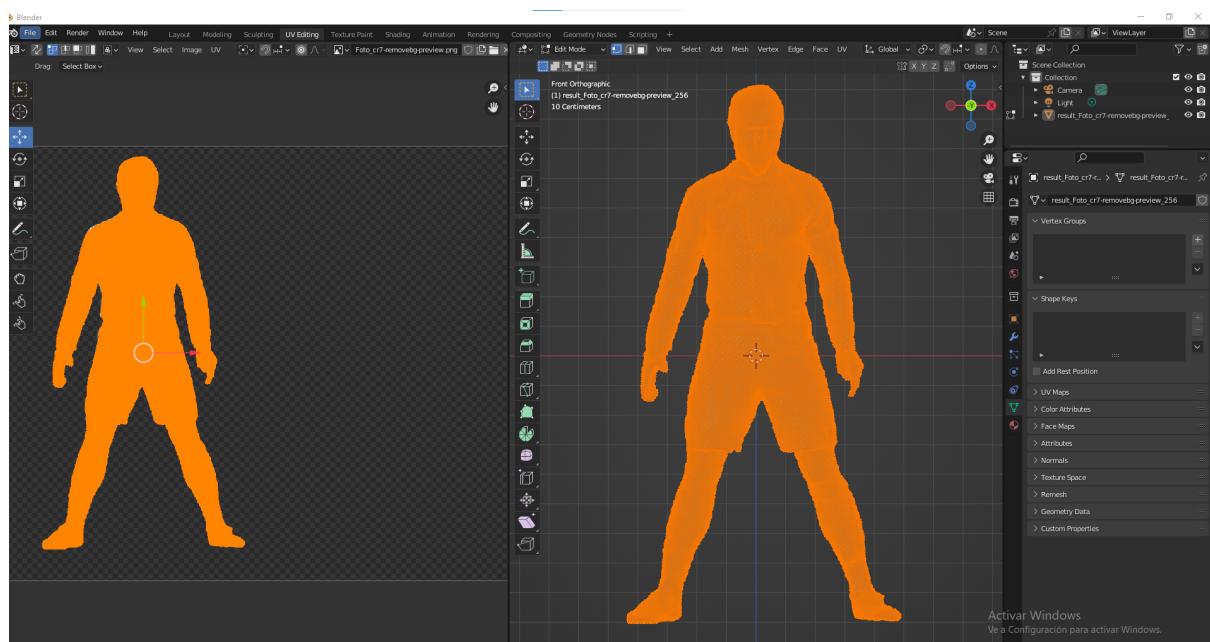


En tercer lugar, vamos a utilizar la herramienta PiFuHD para generar el modelo 3D a partir de la foto del cuerpo de Cristiano Ronaldo. Para generar este modelo importaremos la foto del sin fondo de Cristiano Ronaldo y ejecutaremos los comandos que nos aparezcan en la página hasta generar el modelo 3D. No olvidar de realizar primero una conexión con una cuenta de Google Drive antes de generar el modelo 3D. Para generar este modelo nos hemos ayudado del siguiente tutorial:

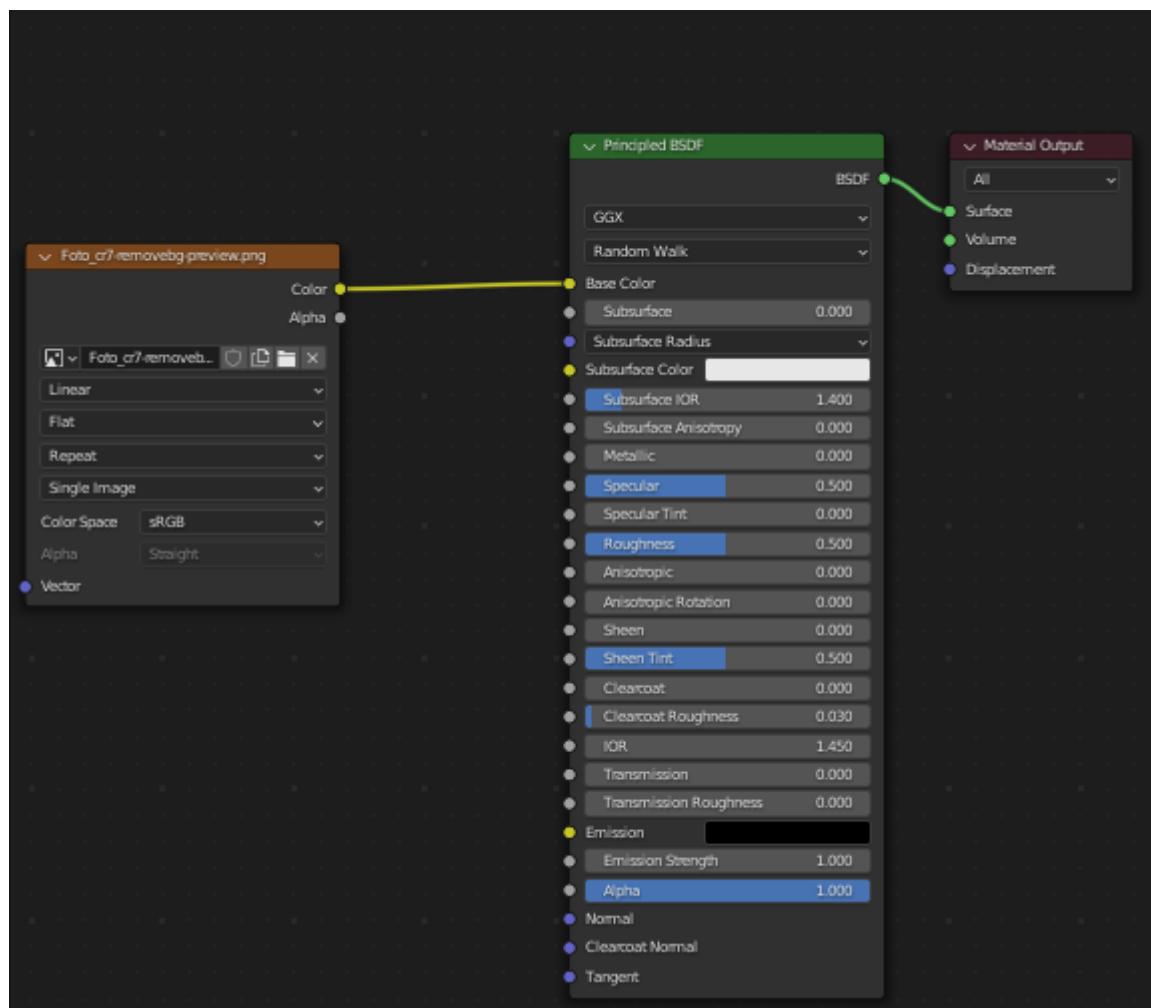
[https://www.youtube.com/watch?v=EEcxAdzVo5s&t=310s&ab\\_channel=Emiliusvgs](https://www.youtube.com/watch?v=EEcxAdzVo5s&t=310s&ab_channel=Emiliusvgs)

Una vez hemos generado el modelo 3D el cuarto paso es añadirle texturas a este modelo 3D para así añadirle la indumentaria de Cristiano Ronaldo. Para ello, importamos nuestro modelo de Cristiano Ronaldo y seleccionaremos la totalidad del cuerpo para aplicarle una textura. Esta textura será generada a partir de una imagen (la imagen de Cristiano Ronaldo sin fondo) y se aplicará a las partes del cuerpo de Cristiano Ronaldo seleccionadas (en este caso se ha seleccionado el cuerpo entero).

En este proceso, cuadraremos la selección del cuerpo de Cristiano Ronaldo con el cuerpo de Cristiano Ronaldo en la foto. En la siguiente imagen adjunta una captura del proceso.



Después, nos iremos al Shader Editor y asociaremos a nuestro modelo la textura que acabamos de crear a partir de esta imagen de Cristiano Ronaldo.



Una vez hecho esto, se nos plantea un pequeño problema, ya que la textura que hemos creado de Cristiano Ronaldo se va a aplicar tanto por delante como por detrás. Y nosotros queremos que por la parte trasera de nuestro modelo 3D se muestre el dorsal de Cristiano Ronaldo. Para arreglar esto, hemos encontrado una foto de Cristiano Ronaldo en el mismo contexto pero realizada desde atrás por lo que tenemos una vista del dorsal y de la parte trasera de Cristiano Ronaldo.



De esta manera, recortaremos esta foto con la herramienta remove.bg y crearemos una nueva textura a partir de esta foto sufriendo los mismos pasos que en el caso anterior. La diferencia en este caso, es que esta textura debe asociarse a un nuevo material para así luego combinar ambos materiales para obtener la textura final.

De esta manera, vamos a crear un segundo material y una vez lo hemos creado vamos a seleccionar la parte trasera del modelo de Cristiano Ronaldo. Una vez hemos seleccionado la parte trasera, procedemos a crear una textura para este material a partir de la foto de Cristiano Ronaldo de la parte trasera de la misma manera que hemos hecho antes.

Una vez hemos creado este segundo material con esta textura delantera, simplemente tenemos que aplicar ambos materiales a nuestro modelo 3D para tener la textura completa de nuestro modelo de Cristiano Ronaldo.

Aplicar texturas a modelos 3D a partir de imágenes en 2D es complicado y debido a esto las partes laterales del modelo no tienen aplicadas las texturas de manera correcta. Debido a esto, vamos a utilizar el modo Texture Paint para ajustar las texturas al modelo 3D del personaje de mejor manera, a través de la difuminación de las texturas.

Una vez hecho esto el resultado del modelo 3D es el siguiente:

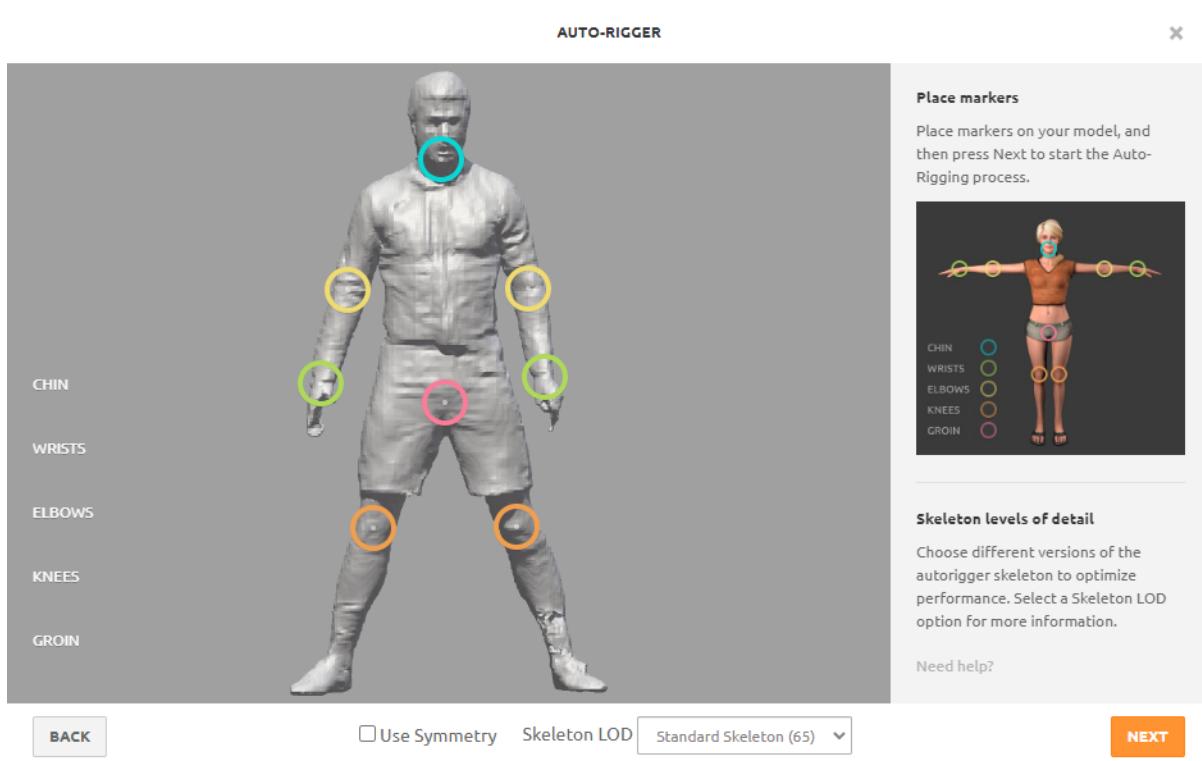


Obsérvese en la parte derecha de la imagen, como el modelo tiene dos materiales asociados donde uno tiene el material con la textura delantera y otro tiene el material con la textura trasera de Cristiano Ronaldo.

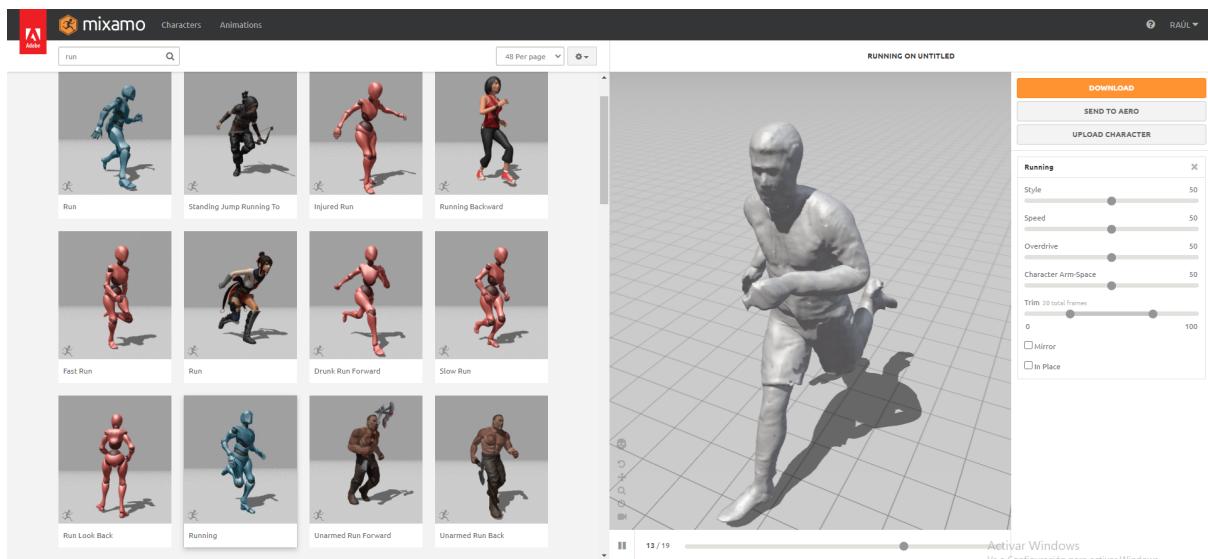
El proceso para aplicar las texturas a nuestro modelo de Cristiano Ronaldo se ha inspirado en este tutorial:

[https://www.youtube.com/watch?v=EEcxAdzVo5s&t=310s&ab\\_channel=Emiliusvgs](https://www.youtube.com/watch?v=EEcxAdzVo5s&t=310s&ab_channel=Emiliusvgs)

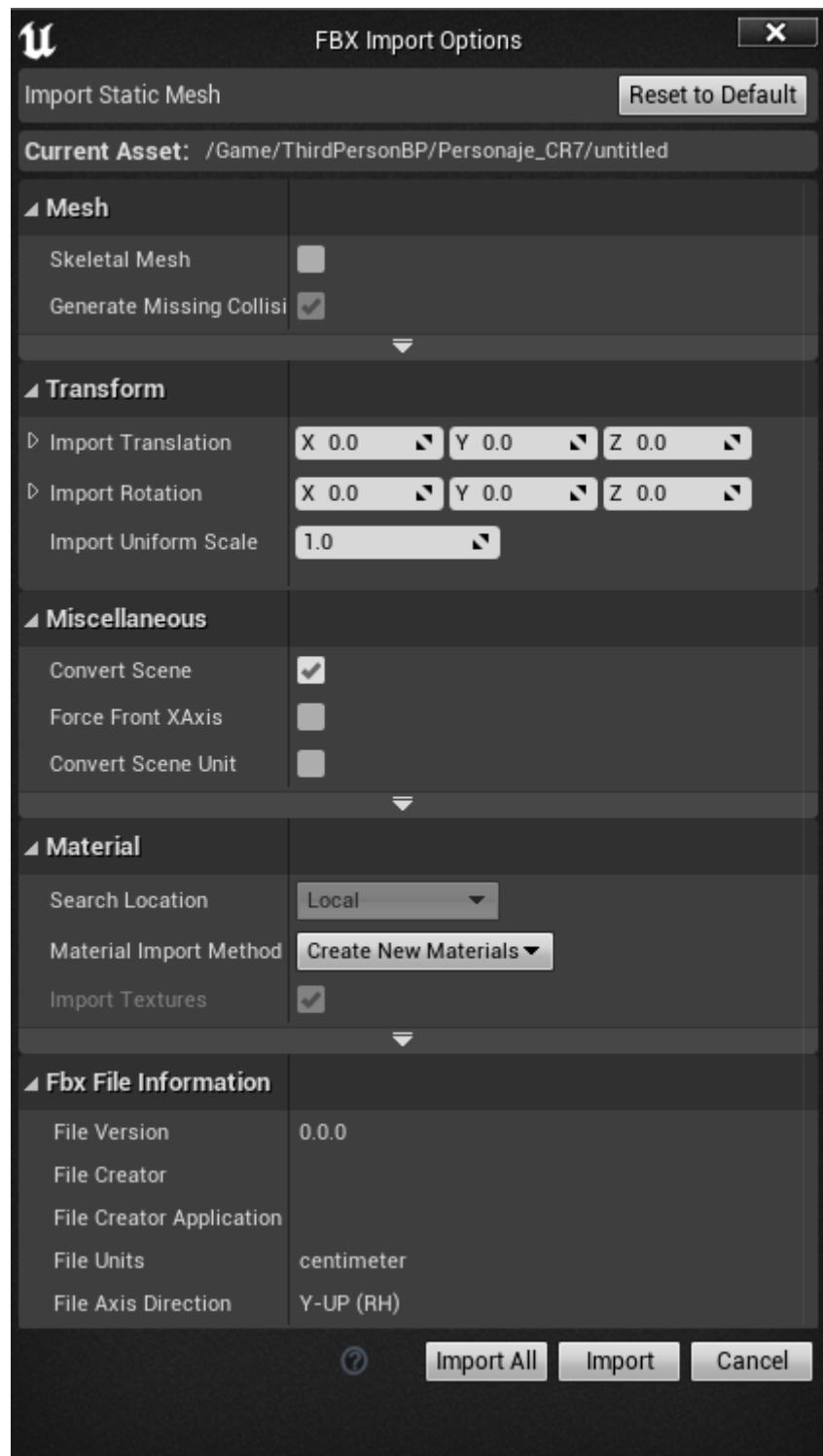
Una vez hemos diseñado el modelo 3D con textura de Cristiano Ronaldo, el siguiente paso es subir este modelo 3D a Mixamo para hacer el autorigging y aplicar animaciones a nuestro modelo de Cristiano Ronaldo. En este proceso de autorigging, vamos a indicar las extremidades y partes del cuerpo importantes para realizar las animaciones. En concreto vamos a indicar dónde se encuentra la barbilla, los codos, las muñecas, las rodillas y la ingle de nuestro modelo de Cristiano Ronaldo.



Una vez hecho esto podemos utilizar este modelo 3D para aplicar distintas animaciones sobre este modelo y descargarlas para usarla en nuestro proyecto de Unreal. Nosotros utilizaremos distintas animaciones para Cristiano Ronaldo como correr, saltar, caerse, levantarse, etc.

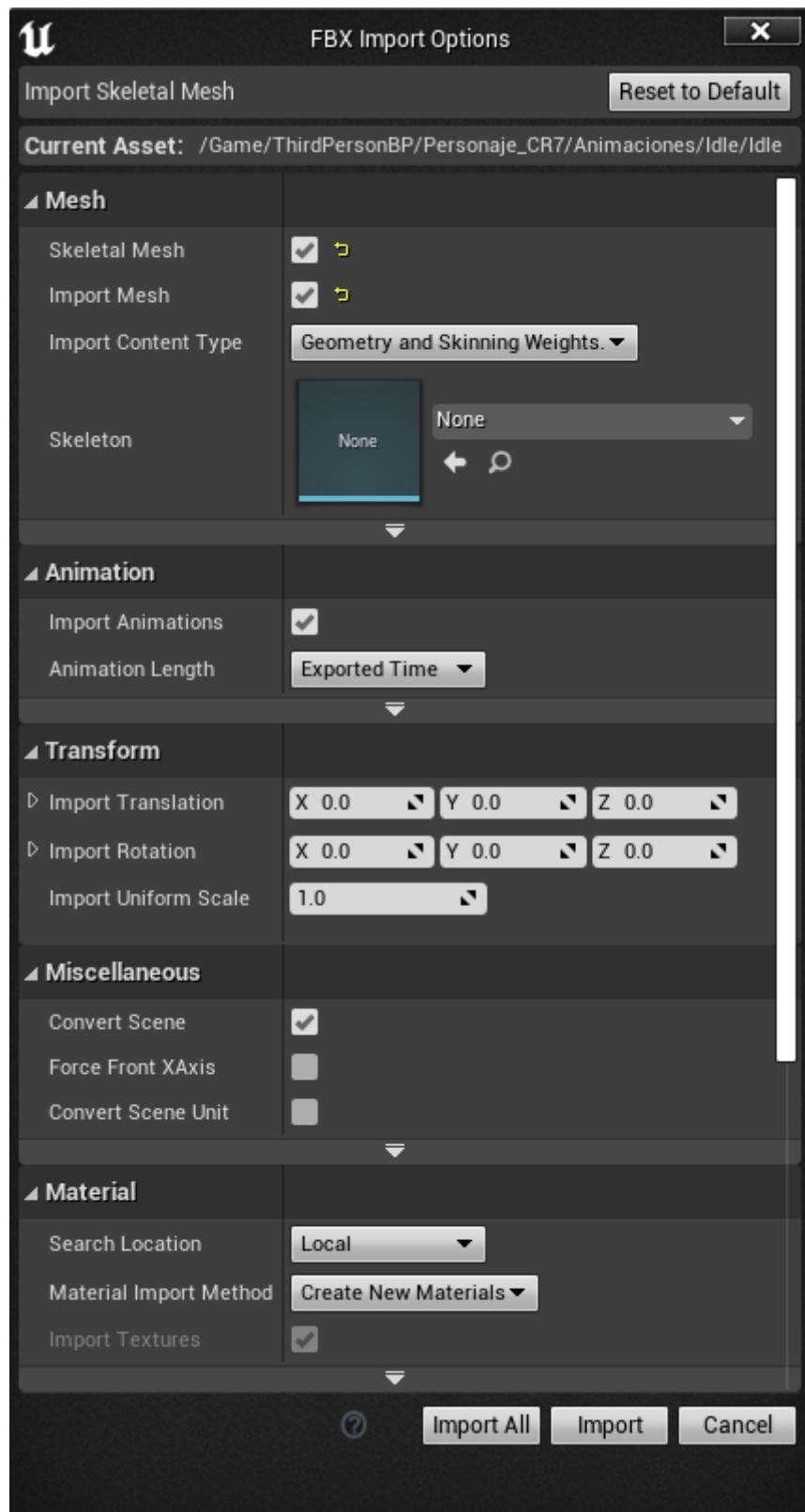


Por último, vamos a incorporar nuestro modelo 3D con textura de Cristiano Ronaldo y las correspondientes animaciones de este modelo que acabamos de generar con Mixamo. Para agregar nuestro modelo 3D de Cristiano Ronaldo, simplemente tenemos que arrastrar el contenido del modelo 3D en el proyecto, y pulsar en Import All.

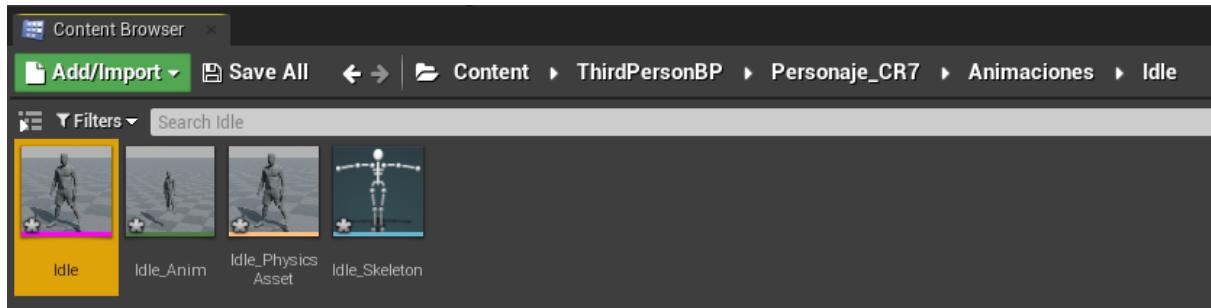


En este caso, hemos decidido no importar la Skeletal Mesh del personaje ya que al utilizar esta skeletal mesh para luego importar las animaciones da fallos de compatibilidad ya que la Skeletal Mesh de las animaciones Mixamo no son compatibles con la Skeletal Mesh de nuestro personaje. Para evitar estos problemas de compatibilidad, hemos decidido no importar el skeletal mesh del personaje para importar posteriormente la Skeletal Mesh del personaje con la propia animación.

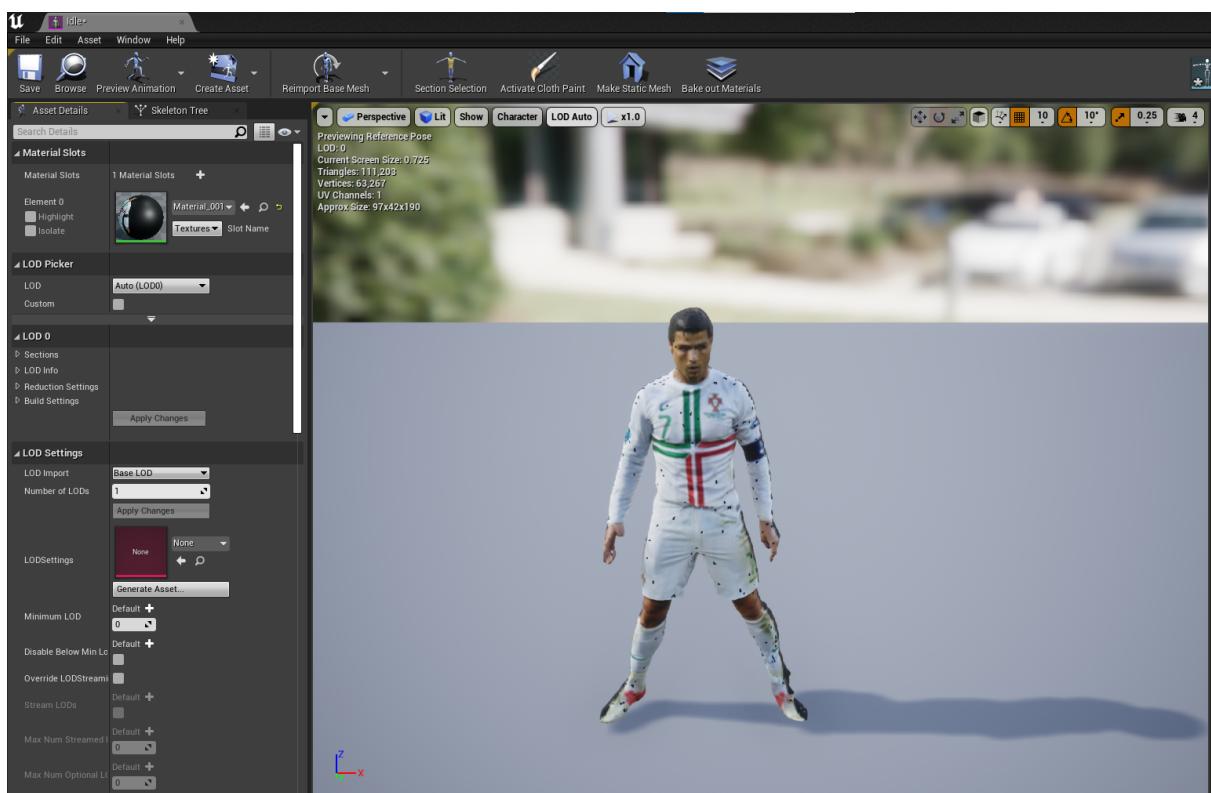
Una vez hemos importado el personaje sin la Skeletal Mesh, vamos a importar la primera animación de nuestro personaje, la cual será la animación de Idle. Todas las animaciones que importemos al proyecto estarán dentro de la carpeta Animaciones dentro de la carpeta del Personaje. Al realizar esta primera importación vamos a marcar la opción Skeletal Mesh para añadir el Skeletal Mesh del personaje en la animación.



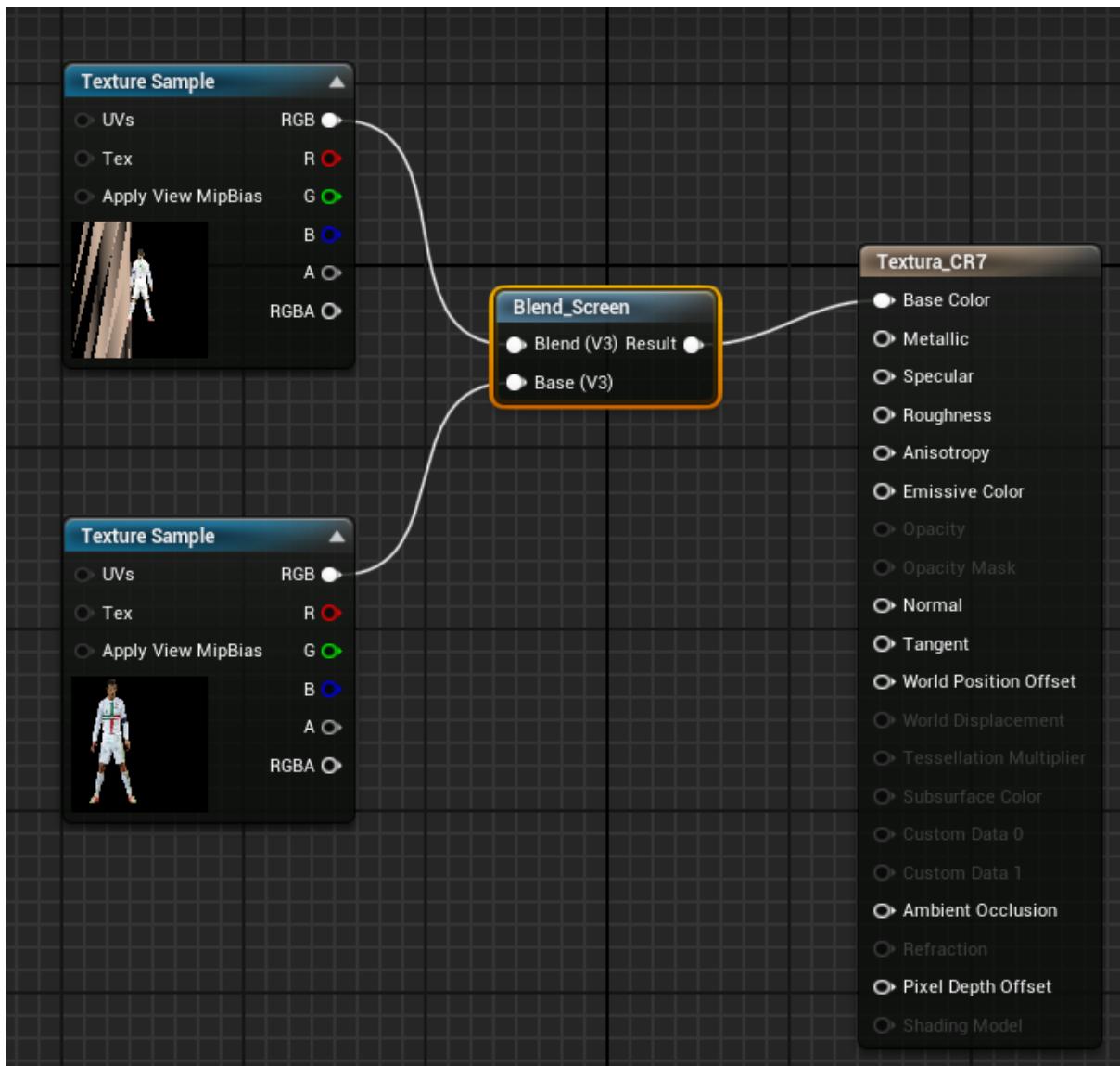
Al realizar la importación de esta animación, nos aparecerán los siguientes archivos:



Como se puede observar, al importar la animación en nuestro proyecto, se han creado 4 archivos (el Mesh, el Skeletal Mesh, la animación y el paquete de físicas). Sin embargo, como se puede observar en las miniaturas, la animación de Idle no incorpora las texturas de Cristiano Ronaldo. Debido a esto, vamos a abrir el Mesh de esta animación y en el panel de la derecha vamos a añadir las texturas de Cristiano Ronaldo.



Sin embargo, como se puede observar, no podemos aplicar las 2 texturas a nuestro personaje al mismo tiempo para la parte delantera y para la trasera. Debido a esto vamos a crear un nuevo material formado por las 2 texturas (parte delantera y trasera). Para ello, vamos a crear un nuevo material llamado Textura\_CR7 y dentro de este material vamos a añadir el siguiente código:

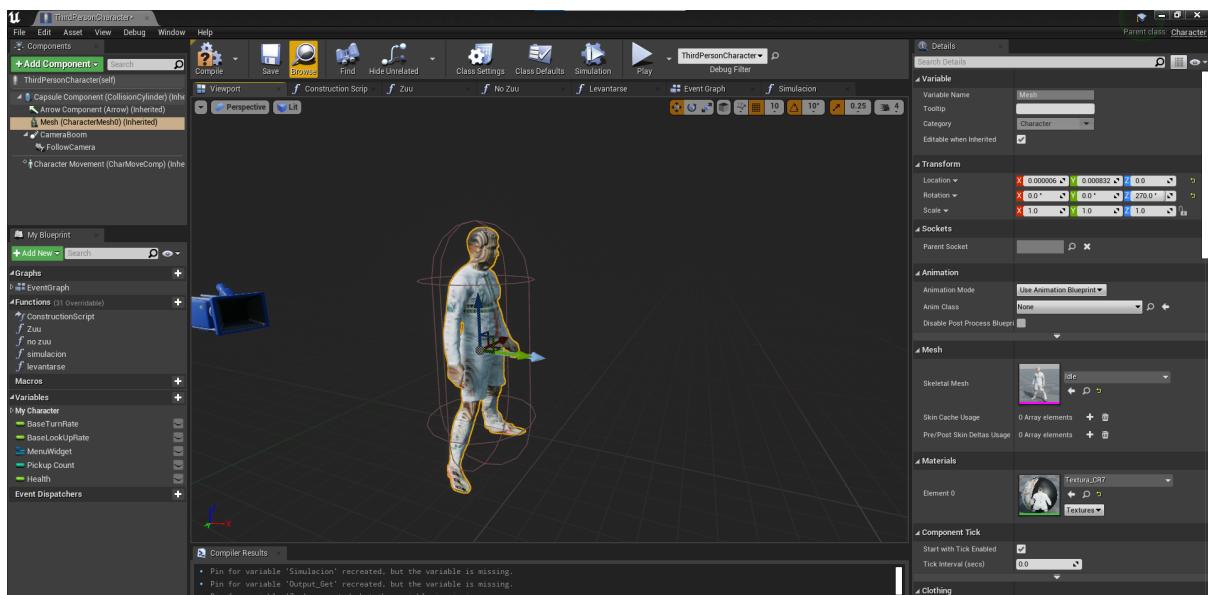


En este código, básicamente se obtiene las texturas de la parte trasera y delantera de cristiano ronaldo y a través del nodo Blend\_Screen se combina ambas texturas en el material Texturas\_CR7.

Una vez hecho esto nos dirigimos al Mesh de la animación Idle y sustituimos el material de la parte delantera de Cistiano Ronaldo por este nuevo material que incorpora las 2 texturas de Cristiano Ronaldo.

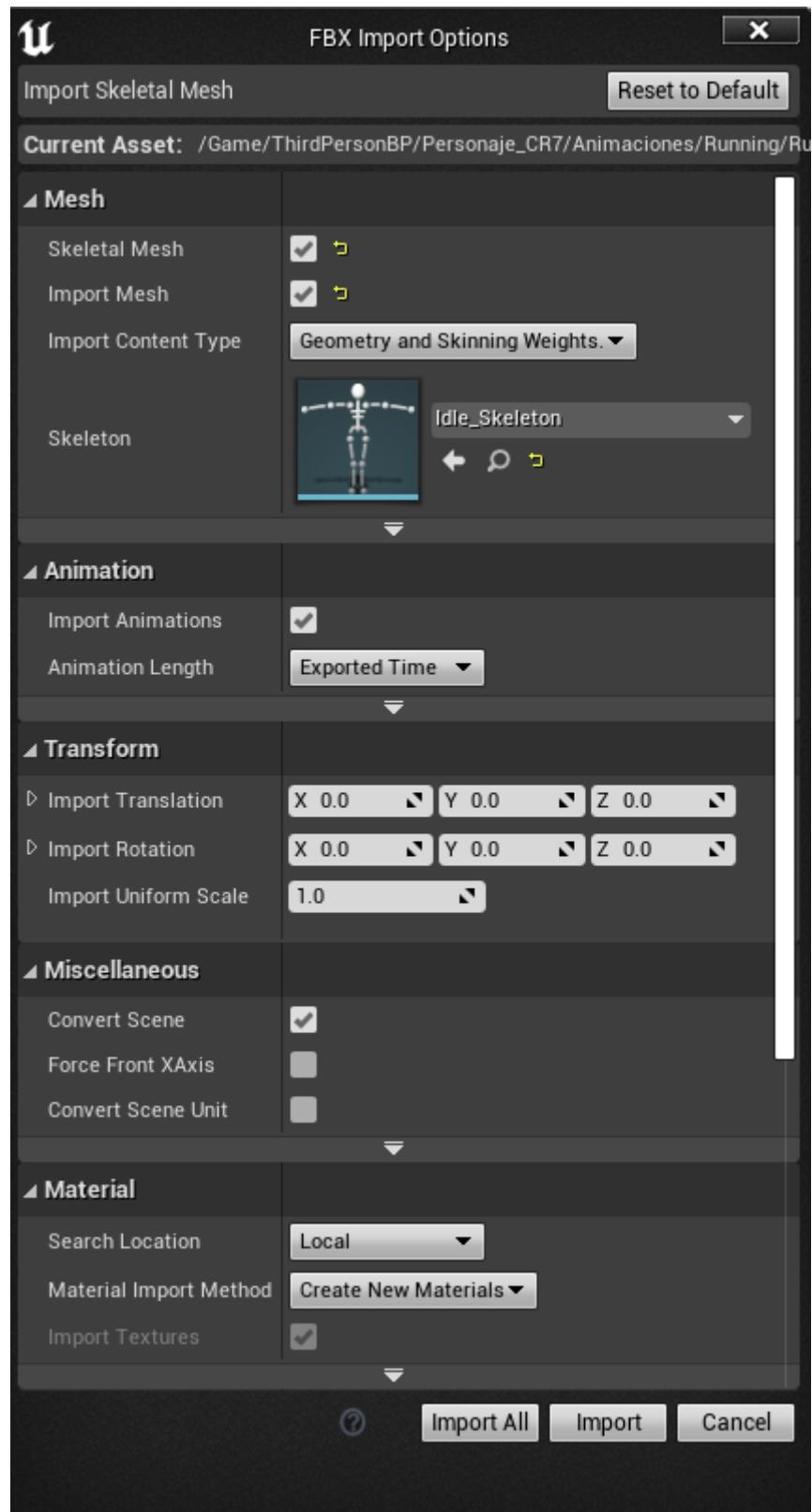


Una vez que hemos importado la Skeletal Mesh de Cristiano Ronaldo, vamos a asociar esta Skeletal Mesh al ThirdPersonCharacter para que tome la referencia del cuerpo de Cristiano Ronaldo y no la del modelo por defecto. Simplemente tenemos que seleccionar el Mesh de nuestro ThirdPersonCharacter y en la sección de Skeletal Mesh asociar el Skeletal Mesh de la animación de Idle.



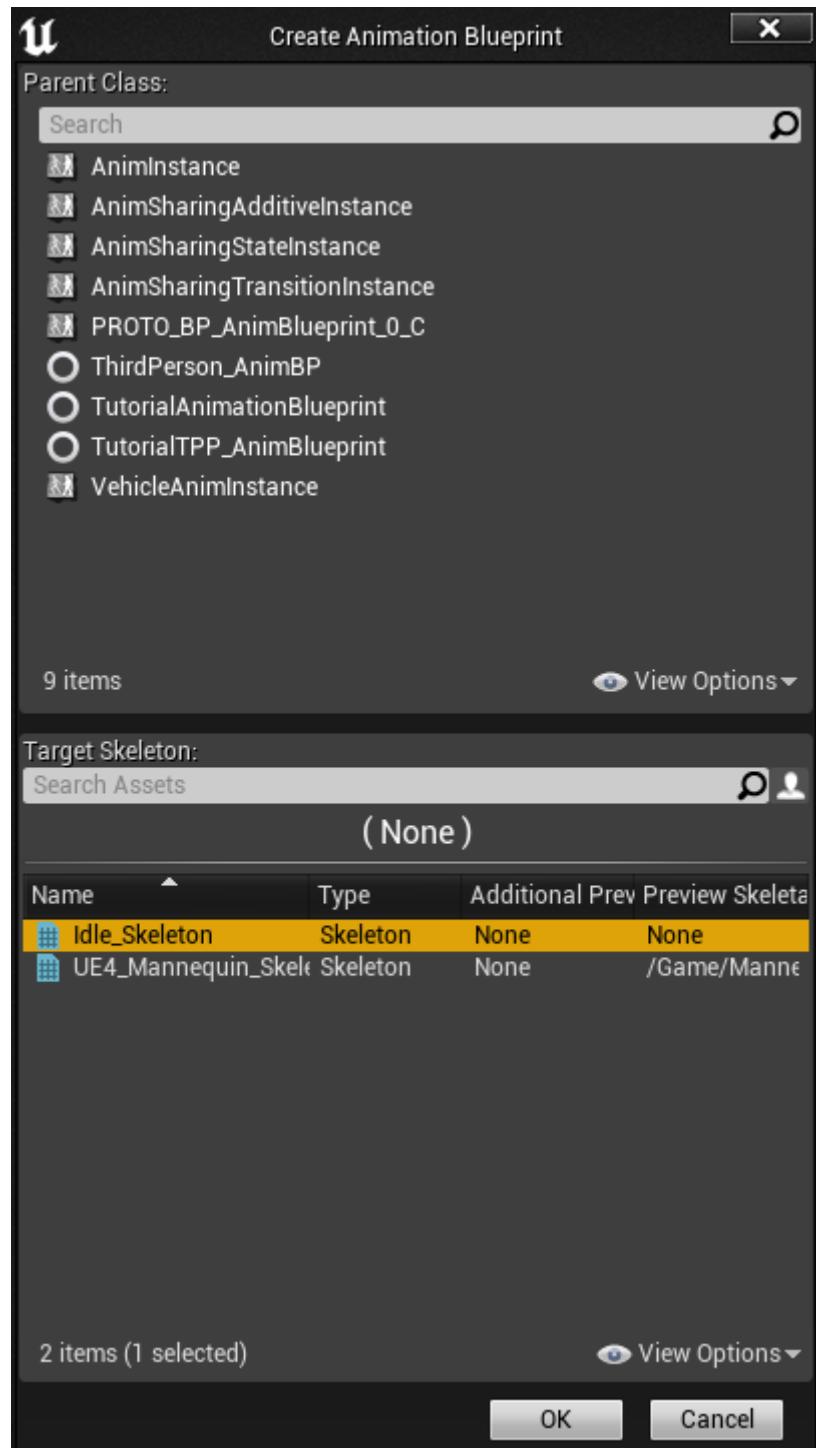
Ahora, vamos a importar las animaciones de correr y saltar y a través de estas animaciones vamos a conseguir que nuestro personaje pueda moverse por el escenario y saltar. Para realizar la importación de estas animaciones hay que tener en cuenta un pequeño detalle.

Ese detalle consiste, en seleccionar la Skeletal Mesh de nuestro personaje de forma que así esta animación use la referencia de este Skeletal Mesh para realizar las animaciones.



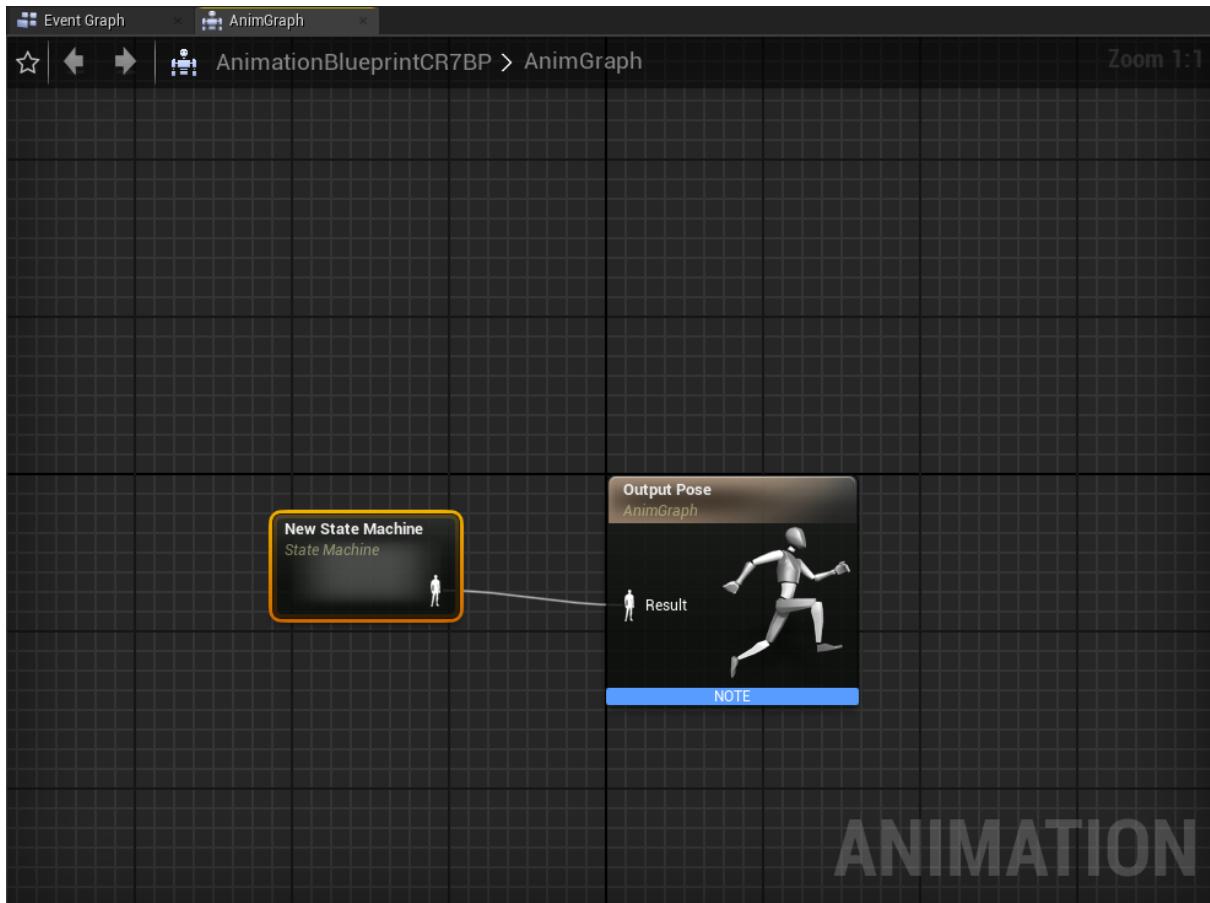
Tampoco olvidar se asociar la textura de Cristiano Ronaldo a todas las animaciones que vayamos a importar. El proceso para realizar este cambio de textura es el mismo que el seguido para la animación de Idle.

Ahora una vez hemos importado las animaciones básicas, vamos a cerrar un Animator Blueprint el cual se va a encargar de gestionar las animaciones de nuestro personaje. De esta manera, vamos a crear un Animation Blueprint llamado AnimationBlueprintCR7BP. A este Animation Blueprint vamos a asociar el Skeletal Mesh de nuestro personaje para que use esta referencia para realizar todas las animaciones.

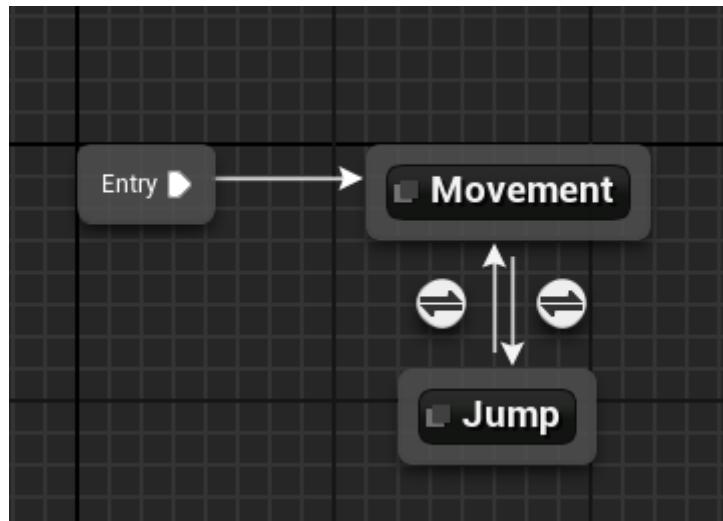


Dentro de este Animation Blueprint, vamos a crear un máquina de estados la cual nos permita realizar transiciones entre las distintas animaciones. Esta máquina de estados va a tener un funcionamiento parecido al Animator Controller que hemos visto en Unity solo que esta herramienta de la máquina de estados es más potente.

Esta máquina de estados, la vamos a conectar con la pose de animación final, y dentro de esta máquina de estados realizaremos esta transición entre las distintas animaciones.

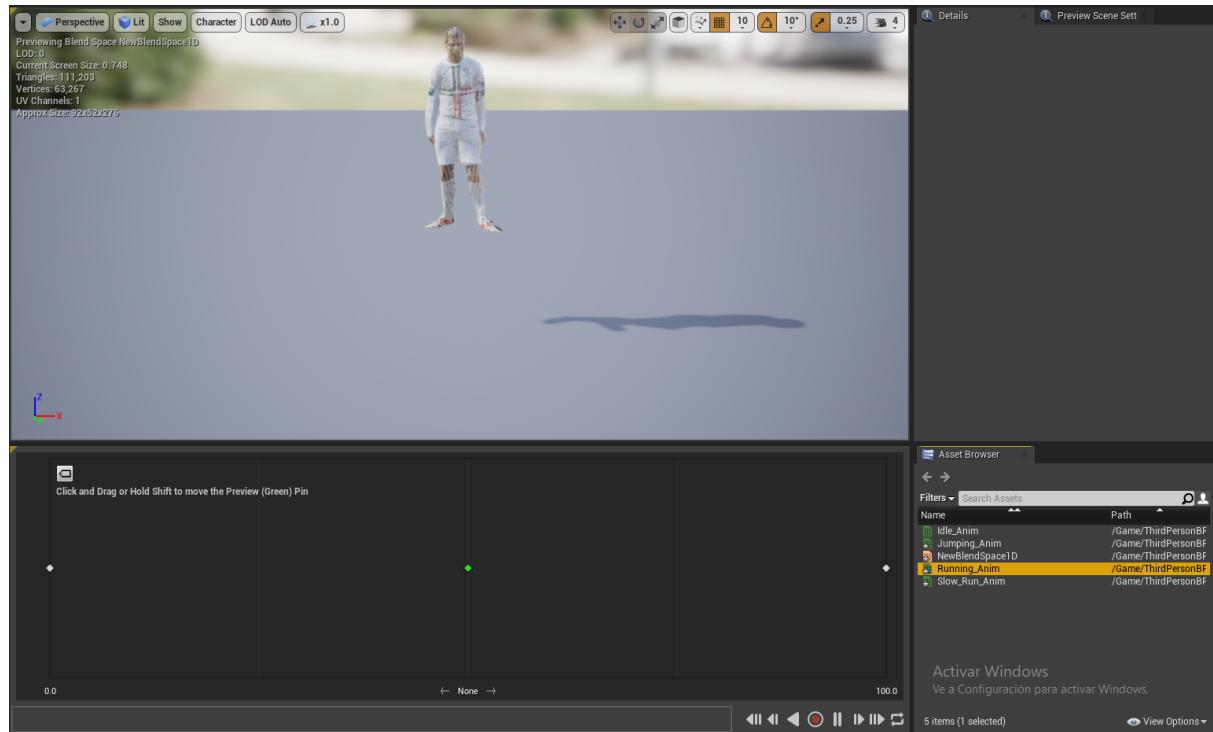


Ahora, vamos a definir los estados de nuestra máquina de estados para controlar las animaciones. Como en primera instancia solo hemos importado las animaciones de Idle, de correr y de saltar, nuestra máquina de estados va a ser tan simple como un estado de movimiento y un estado de salto.



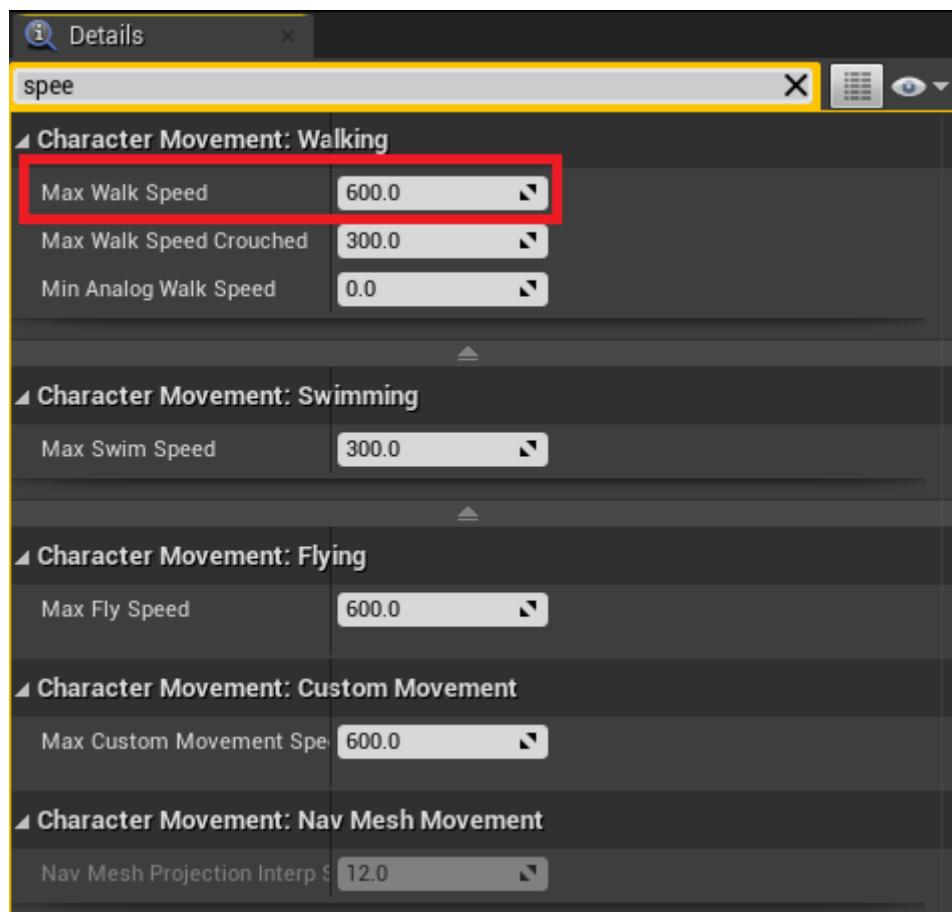
Ahora, vamos a entrar en el estado de Movement y vamos a definir la animación que se reproducirá cuando nuestro personaje esté moviéndose. Para definir esta animación, vamos a crear un blend de animaciones ya que queremos realizar un mezcla de varias animaciones para crear la animación final que se va a reproducir cuando se mueva nuestro personaje. No olvidar de asociar el Skeletal Mesh de nuestro personaje a este blend de animaciones.

Una vez hayamos creado el blend de animaciones, arrastramos las animaciones de Idle, Run Slow y Running a la línea horizontal del Blend de animaciones. Vamos a colocar evidentemente estas tres animaciones a la misma distancia una de otra para que las transiciones entre animaciones se hagan de manera suave y con la misma velocidad.

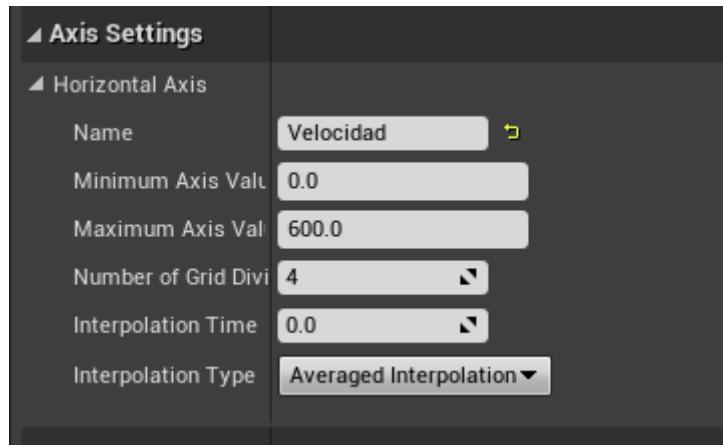


Esta línea horizontal en el que hemos arrastrado las distintas animaciones representa la velocidad del personaje. Debido a esto, nos vamos a ir al panel de la izquierda en el panel de Axis Settings y vamos a cambiar el nombre de esta línea horizontal para que se llame Velocidad. Al cambiar este nombre de la línea horizontal, podremos acceder a esta línea horizontal.

Además de cambiar el nombre de esta línea horizontal, tenemos que ajustar el valor máximo de esta línea horizontal para adecuarla con la velocidad máxima definida para nuestro personaje. Para conocer la velocidad máxima de nuestro personaje, accedemos al personaje y seleccionamos el componente del movimiento del personaje y en el panel de detalles buscamos la velocidad máxima del personaje.



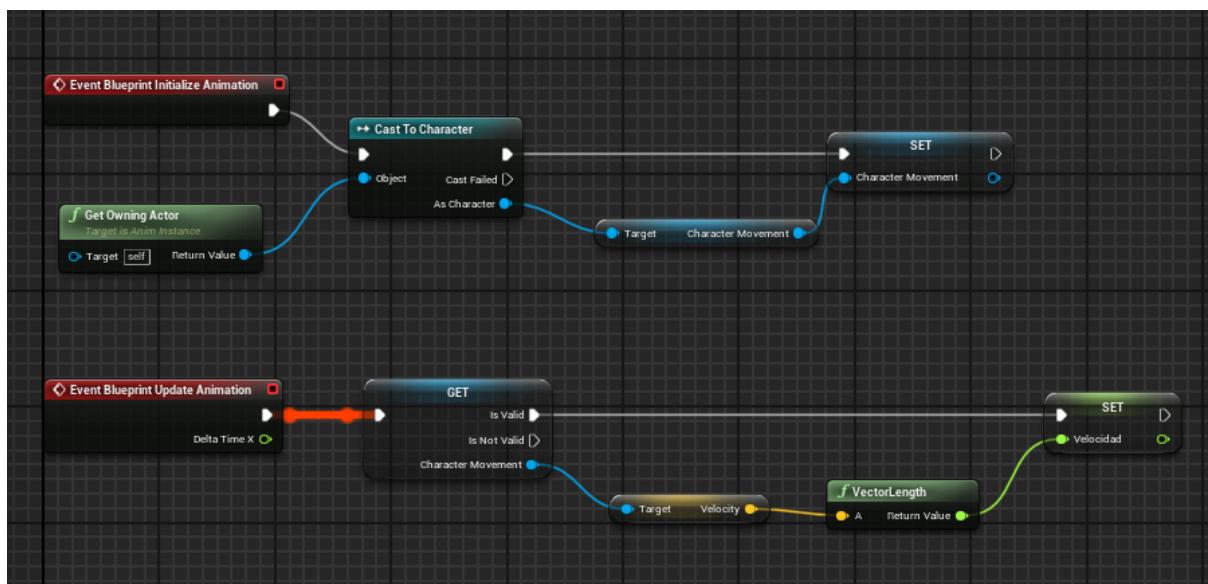
De esta manera, el valor máximo de nuestra línea horizontal del blend de animaciones debe ser de 600 para que se ajuste a la velocidad máxima del personaje y que la mezcla de animaciones se realice correctamente.



Con esto ya hemos hecho la mezcla de animaciones que queremos que se reproduzca cuando se mueva el personaje. De esta manera, ahora vamos a entrar en el estado de Movement de la máquina de estados y vamos a asociar nuestro blend de animaciones de movimiento a la pose de animación de este estado.



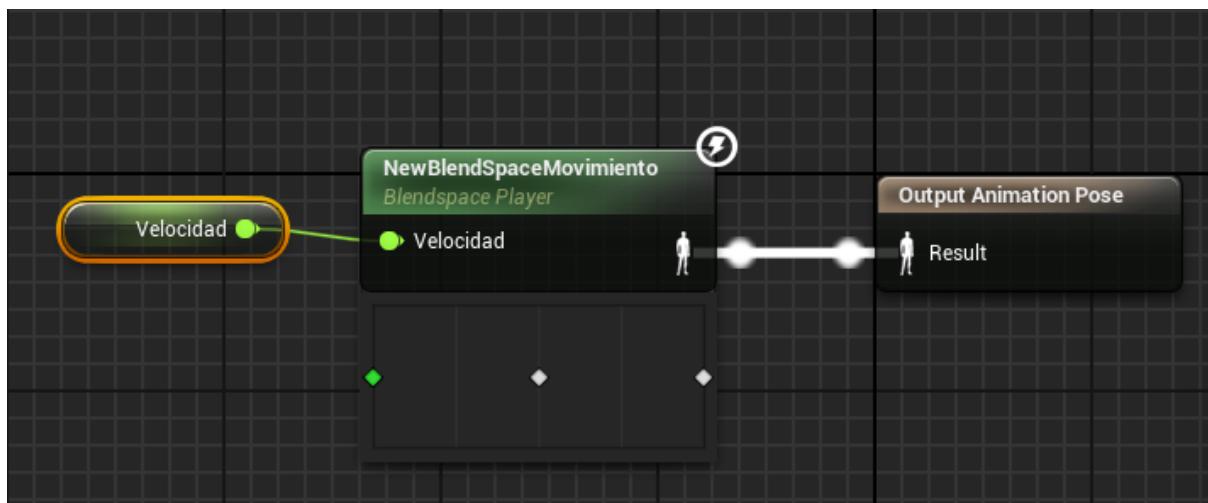
Una vez hecho esto, ahora vamos a controlar mediante programación la actualización de este valor de velocidad en función de la velocidad del personaje para así realizar esta mezcla de transiciones correctamente. Para ello utilizaremos el siguiente código:



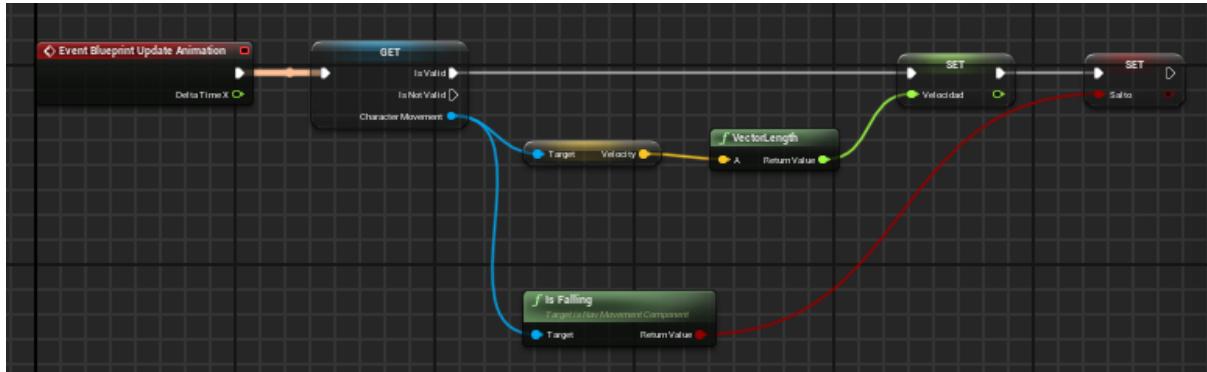
En este código, cuando se produce el evento de inicialización de la animación, se obtiene el blueprint del personaje a través de Cast to Character y Get Owner Actor que hace referencia al actor propietario del Animation Blueprint. Una vez obtenemos esta referencia del personaje, vamos a acceder al movimiento del personaje a través del nodo Get Character Movement. Como esta información del movimiento del personaje se va a utilizar varias veces, se va almacenar en una variable y para ello utilizaremos la opción “Promote to variable” desde el nodo Get Character Movement.

Después, cuando se produzca el evento de actualización de la animación, a través de un get del Movimiento del Personaje, vamos a extraer el valor de la velocidad del carácter a través del nodo Get Velocity. Como este valor de velocidad se trata de un vector, vamos a convertir este valor en un flotante a través de la función Vector Length. Por último, creamos una variable float llamada velocidad en el que almacenamos este valor de velocidad del personaje.

Por último, en el estado de Movement vamos a obtener la velocidad de este personaje (que está almacenada en la variable velocidad) a través de un get y vamos a enviar al blend de animaciones este valor de velocidad para que haga la mezcla de animaciones de manera correcta en función de la velocidad del personaje.

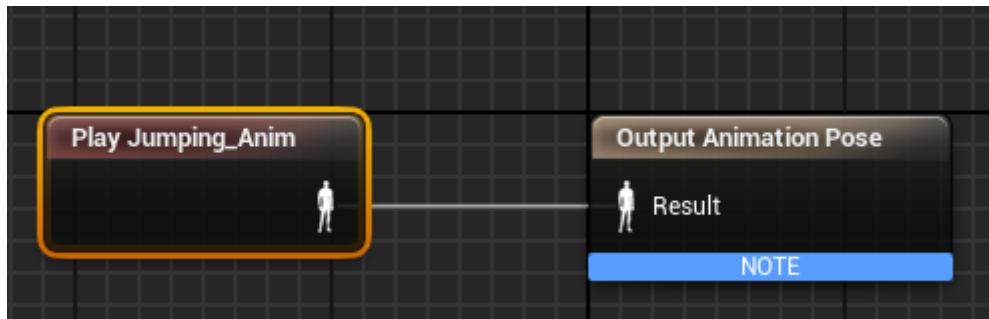


Con esto ya hemos configurado perfectamente la animación de movimiento. Ahora, vamos a configurar la animación de salto. Para ello, volvemos al código del Animation Blueprint y vamos a añadir un par de modificaciones.



Estas modificaciones consisten en acceder al personaje (al igual que hicimos con la velocidad) para determinar si el personaje está en el aire o no. Para ello se utilizará el nodo Is Falling que nos determina automáticamente si el personaje está en el aire o no. Este nodo nos devuelve un booleano indicándonos si el personaje está en el aire o no. Lo almacenaremos en una variable booleana llamada Salto que nosotros crearemos.

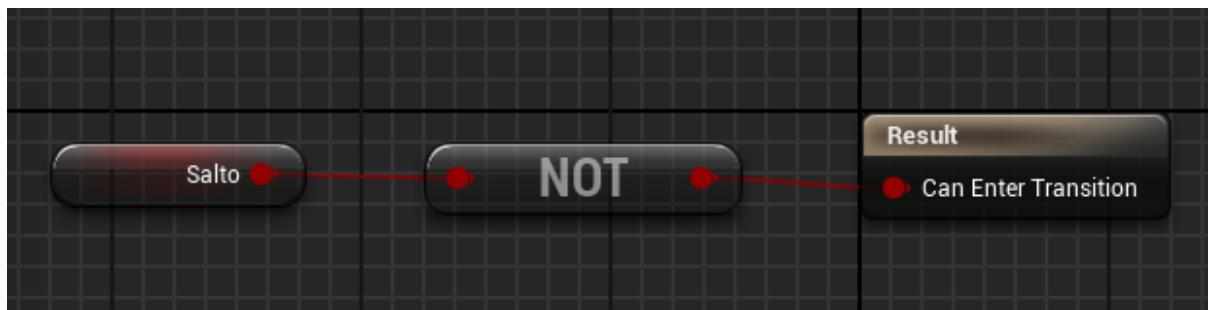
Una vez hecho esto, nos vamos al estado de Jump y unimos la animación de salto a la pose de animación de este estado.



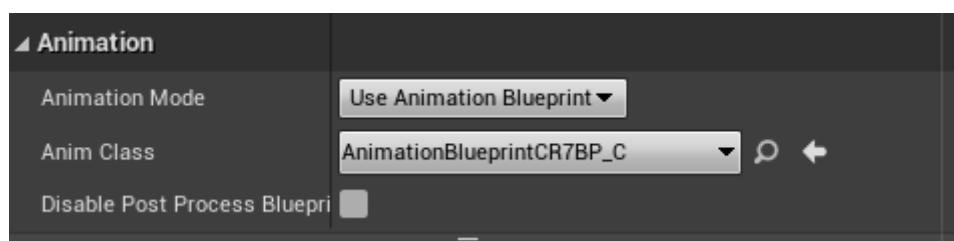
Al añadir esta animación, solo nos interesa que se reproduzca una vez por lo que desactivaremos la opción de Loop Animation la cual se encuentra en la pestaña de la derecha al seleccionar la animación.

Por último, vamos a definir las transiciones entre este estado de Movimiento y este estado de Salto. Para ello, nos vamos a ayudar de esta variable booleana que nos indica si el jugador está en el aire o no. De esta manera, vamos a realizar la transición del estado de Movement al estado de Jump cuando la variable saltar sea true y vamos a realizar la transición del estado de Jump al estado de Movement cuando la variable saltar sea false.





Por último, no olvidar asociar a nuestro personaje, el Animation Blueprint que controla todas las animaciones del personaje. De no hacer este último paso, todo lo que hemos hecho hasta ahora no serviría de nada. Para asociar nuestro Animation Blueprint al personaje accedemos de nuevo al componente Mesh del personaje y en la sección de Animación, en Anim Class seleccionamos nuestro Animation Blueprint.



Con esto ya hemos realizado todos los pasos necesarios para controlar las animaciones básicas de nuestro personaje. A partir de esta base vamos a ir incorporando distintas animaciones e incrementando esta máquina de estados para que el comportamiento de nuestro personaje sea mucho más completo.

## Añadiendo más animaciones a nuestro personaje

### Animación de celebración

Una de las señas de identidad de Cristiano Ronaldo es su famosa celebración en la que salta con los 2 brazos extendidos. Debido a esto, vamos a importar una animación que recrea más o menos el estilo de esta celebración y queremos que esta animación se reproduzca cuando el Jugador pulse la tecla Z.

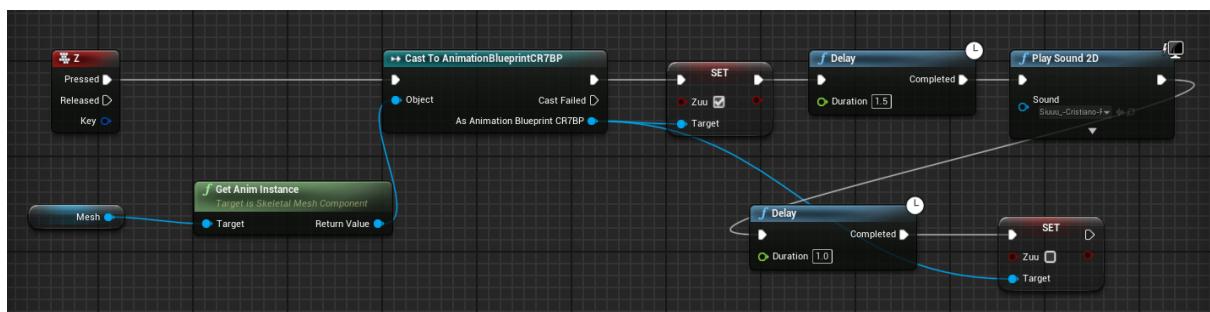
Para ello, en primer lugar vamos a importar la animación de Celebración (llamada jumping) y después vamos a añadir un estado llamado Celebración en el cual se reproducirá la animación de celebración. (Nota: No olvidar de asociar las texturas a la animación e importar la animación utilizando el esqueleto del personaje).



Al añadir esta animación, solo nos interesa que se reproduzca una vez por lo que desactivaremos la opción de Loop Animation la cual se encuentra en la pestaña de la derecha al seleccionar la animación.

Después, vamos a crear las transiciones que nos van a llevar y sacar del estado de celebración. Para ello vamos a crear una variable booleana llamada zuu la cual va ser evaluada para realizar las transiciones con el estado de Celebración. Si la variable zuu es verdadera nuestro personaje pasa del estado de Movimiento al estado de Celebración y si la variable zuu es falsa nuestro personaje pasa del estado de Celebración al estado de Movimiento.

Por último, vamos a acceder al blueprint del personaje y vamos a añadir el siguiente código:



En este código, cuando el jugador pulsa la tecla Z, se obtiene una referencia del Animation Blueprint del mesh de nuestro personaje a través de los nodos Get Anim Instance y Cast To AnimationBlueprintCR7BP. Una vez accedemos al contenido del Animation Blueprint, vamos a establecer el valor de la variable booleana zuu a verdadero. Después, esperamos 1.5 segundos para que el jugador caiga y se reproduzca el sonido del grito de Cristiano Ronaldo, el cual, hemos importado a nuestro proyecto. Por último, se espera 1 segundos y se establece el valor del booleano zuu a falso para volver al estado de Movimiento.

## Animación de simulación

Otra de las señas de identidad de Cristiano Ronaldo es su fama por simular penaltis y tirarse al suelo sin que ningún rival le haya hecho nada. Debido a esto, vamos a importar una animación que recrea un “piscinazo” de Cristiano Ronaldo y queremos que esta animación se reproduzca cuando el Jugador pulse la tecla P.

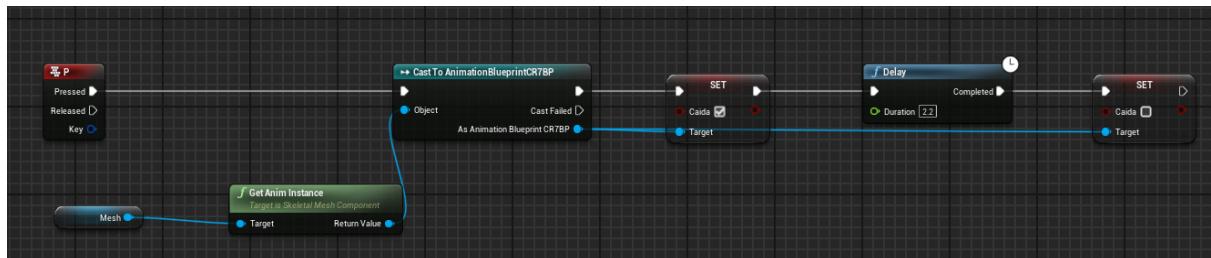
Para ello, en primer lugar vamos a importar la animación de Simulación (llamada Soccer Tackle) y después vamos a añadir un estado llamado Simulación en el cual se reproducirá la animación de simulación. (Nota: No olvidar de asociar las texturas a la animación e importar la animación utilizando el esqueleto del personaje).



Al añadir esta animación, solo nos interesa que se reproduzca una vez por lo que desactivaremos la opción de Loop Animation la cual se encuentra en la pestaña de la derecha al seleccionar la animación.

Después, vamos a crear las transiciones que nos van a llevar y sacar del estado de simulación. Para ello vamos a crear una variable booleana llamada caída la cual va ser evaluada para realizar las transiciones con el estado de Simulación. Si la variable caída es verdadera nuestro personaje pasa del estado de Movimiento al estado de Simulación y si la variable caída es falsa nuestro personaje pasa del estado de Simulación al estado de Movimiento.

Por último, vamos a acceder al blueprint del personaje y vamos a añadir el siguiente código:



En este código, cuando el jugador pulsa la tecla P, se obtiene una referencia del Animation Blueprint del mesh de nuestro personaje a través de los nodos Get Anim Instance y Cast To AnimationBlueprintCR7BP. Una vez accedemos al contenido del Animation Blueprint, vamos a establecer el valor de la variable booleana caída a verdadero. Después, esperamos 2.2 segundos para que el jugador caiga y se establece el valor del booleano caída a falso para volver al estado de Movimiento.

## Animación de disconformidad

Cristiano Ronaldo también tiene fama por sus airadas protestas. Debido a esto, vamos a importar una animación que va a recrear un gesto de disconformidad y queremos que esta animación se reproduzca cuando el Jugador pulse la tecla C.

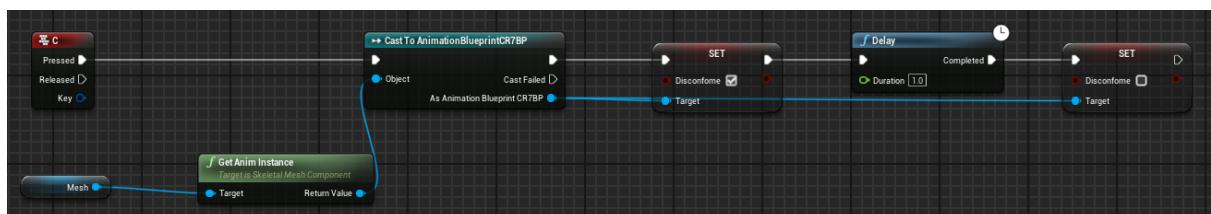
Para ello, en primer lugar vamos a importar la animación de Disconformidad (llamada Dismiss Gesture) y después vamos a añadir un estado llamado Disconformidad en el cual se reproducirá la animación de disconformidad. (Nota: No olvidar de asociar las texturas a la animación e importar la animación utilizando el esqueleto del personaje).



Al añadir esta animación, solo nos interesa que se reproduzca una vez por lo que desactivaremos la opción de Loop Animation la cual se encuentra en la pestaña de la derecha al seleccionar la animación.

Después, vamos a crear las transiciones que nos van a llevar y sacar del estado de Disconformidad. Para ello vamos a crear una variable booleana llamada disconforme la cual va ser evaluada para realizar las transiciones con el estado de Disconformidad. Si la variable disconforme es verdadera nuestro personaje pasa del estado de Movimiento al estado de Disconformidad y si la variable disconforme es falsa nuestro personaje pasa del estado de Disconformidad al estado de Movimiento.

Por último, vamos a acceder al blueprint del personaje y vamos a añadir el siguiente código:



En este código, cuando el jugador pulsa la tecla C, se obtiene una referencia del Animation Blueprint del mesh de nuestro personaje a través de los nodos Get Anim Instance y Cast To AnimationBlueprintCR7BP. Una vez accedemos al contenido del Animation Blueprint, vamos a establecer el valor de la variable booleana disconforme a verdadero. Después, esperamos 1 segundo para que el jugador caiga y se establece el valor del booleano disconforme a falso para volver al estado de Movimiento.

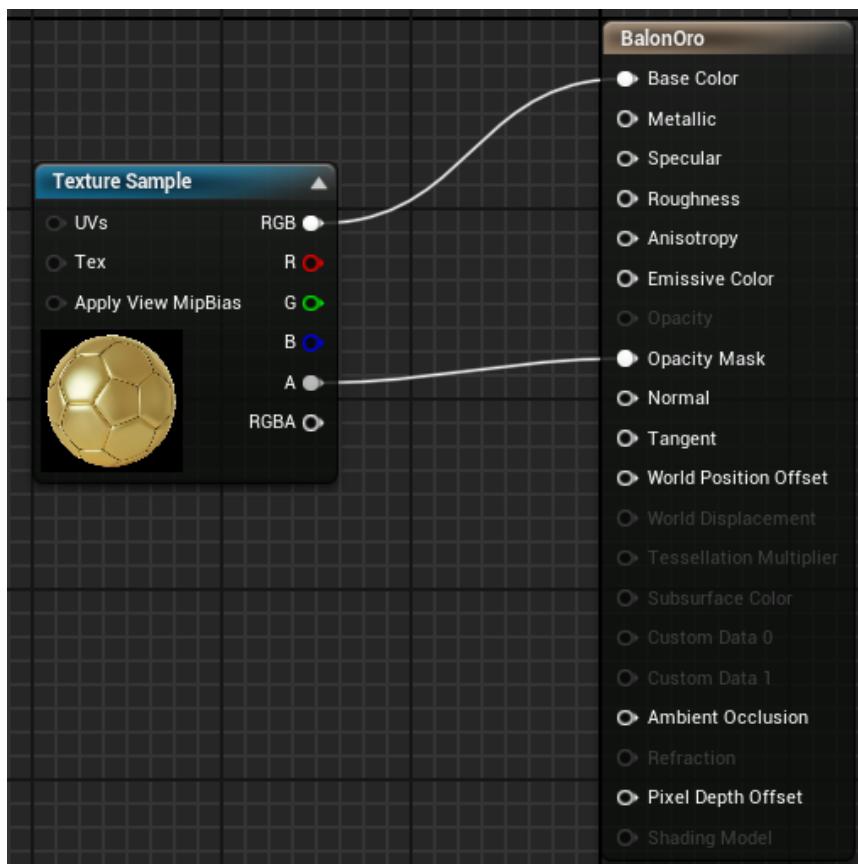
# Creación de objetos o pickups

Una vez hemos importado el modelo de nuestro personaje a Unreal y hemos establecido las animaciones básicas y las animaciones “complementarias”, el siguiente paso que vamos a realizar es la introducción de diferentes objetos que pueda recoger nuestro personaje. Cada vez que nuestro personaje coja un objeto se producirá una acción determinada dependiendo del tipo de objeto.

## Objeto coleccional “Balon de Oro”

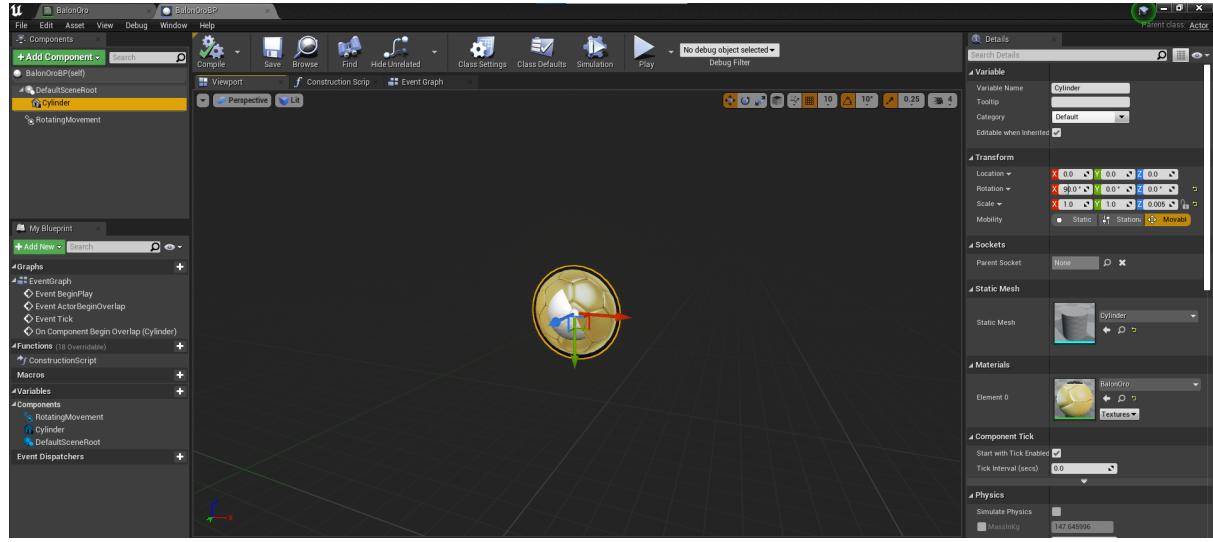
En primer lugar, vamos a diseñar un objeto “coleccional” de forma que cada objeto de este tipo que recoja nuestro jugador tendrá de alguna manera influencia en la puntuación final del juego. Como a nuestro personaje Cristiano Ronaldo le gusta recoger balones de oro (ha ganado 5 balones de oro en su carrera) este objeto o pick up en cuestión será un balón de oro de forma que cada vez que el personaje coja un balón de oro, se incrementará en una unidad el número de balones de oro recogidos por el jugador. Más adelante, también crearemos un menú HUD en el que se muestre de manera dinámica el número de balones de oro recogidos por el jugador.

Para crear este objeto, en primer lugar, vamos a diseñar la textura que utilizará este objeto. En nuestro caso, para crear la textura del balón de oro, nos descargaremos una imagen recortada sin fondo de un balón de oro y después crearemos un material llamado BalonOro. Dentro de este material, crearemos una textura a partir de la imagen del balón de oro recortada y esa textura se aplicará a nuestro material.

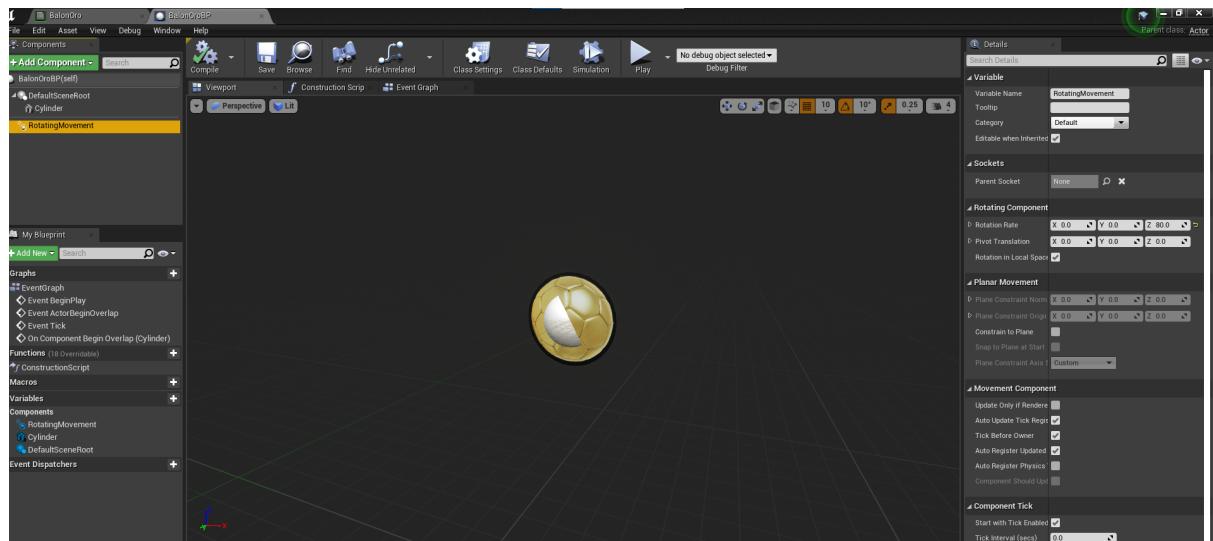


Para crear esta textura, se ha modificado el parámetro Blend Mode del material al valor masked de forma que se utiliza el valor alfa de nuestra fotografía para definir la máscara de opacidad del objeto y así conseguir un mejor texture para nuestro objeto.

En segundo lugar, crearemos un Blueprint Class de tipo Actor cuyo nombre será BalonOroBP. Dentro de este Blueprint Class, añadiremos un cilindro el cual giraremos 90 grados y dejaremos su componente Z casi a 0 para que tenga la forma de un círculo en 2D. Después, aplicaremos la textura del balón de oro que acabamos de crear para así tener nuestro objeto balón de oro perfectamente diseñado.

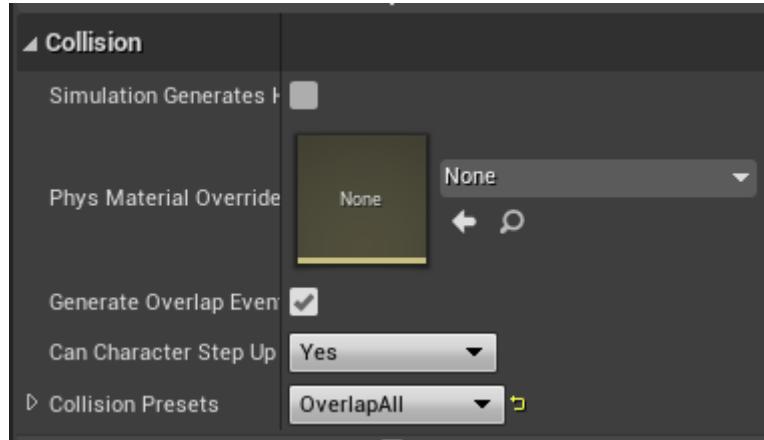


Ahora, podemos arrastrar el contenido de este Blueprint Class al escenario de forma que el objeto balón de oro se visualiza perfectamente. Sin embargo, queremos que este objeto esté rotando continuamente para poder visualizarlo desde todos los ángulos (ya que es un objeto 2D) y para darle un componente de dinamismo al objeto. Para ello, vamos a añadir al Blueprint Class un componente RotatingMovement para establecer una rotación en el eje Z a nuestro objeto.

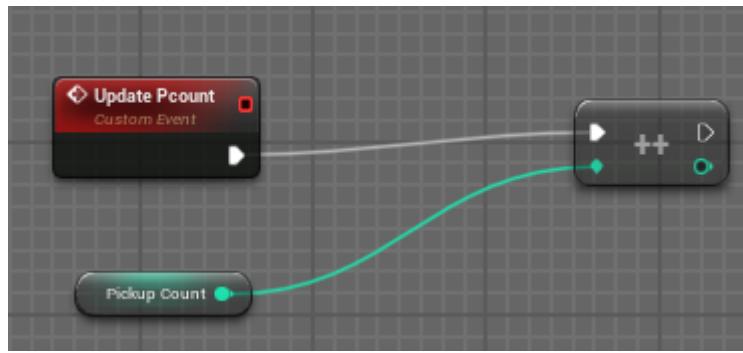


En nuestro caso, hemos elegido una tasa de rotación de 80 grados para el eje Z para que así el objeto se mueva a una velocidad moderada y se pueda visualizar sin ningún tipo de problema.

Una vez hecho esto, queremos que nuestro personaje pueda atravesar el objeto y recogerlo (ya que el objeto detecta las colisiones). Para ello, en primer lugar, vamos a configurar el objeto de forma que nuestro personaje pueda atravesarlo. Para ello, lo único que tenemos que hacer es acceder a nuestro Blueprint Class y en la sección de colisiones del objeto establecer el valor de OverlapAll.

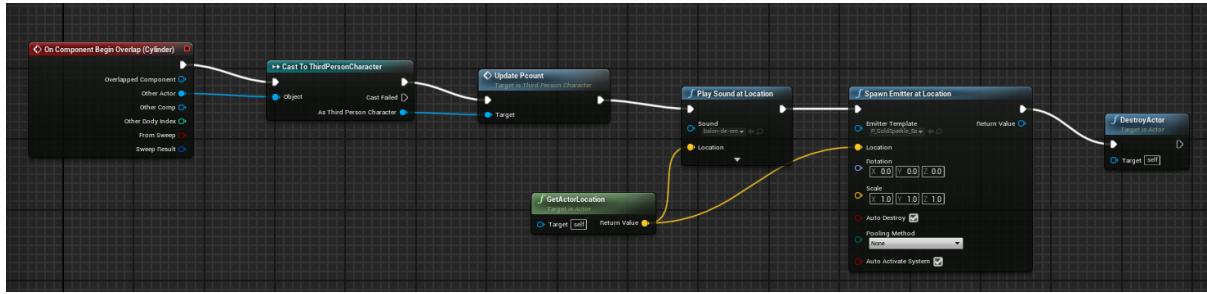


Con esto nuestro personaje puede atravesar el objeto. Ahora, queremos que cuando nuestro personaje atraviese el objeto, lo recoja y se incremente un contador con el número de ítems recogidos. Para ello, en el Blueprint del personaje vamos a crear el siguiente código:



Básicamente con este código, definimos un evento customizado llamado Pcount que se ejecutará desde el blueprint del objeto el cual aumentará en una unidad la cantidad de balones de oro cogidos por el jugador. Este valor del número de balones de oro será almacenado en una variable entera llamada Pickup Count la cual habremos creado para este cometido.

Por otra parte, en el blueprint del objeto Balón de Oro se ha utilizado el siguiente código:

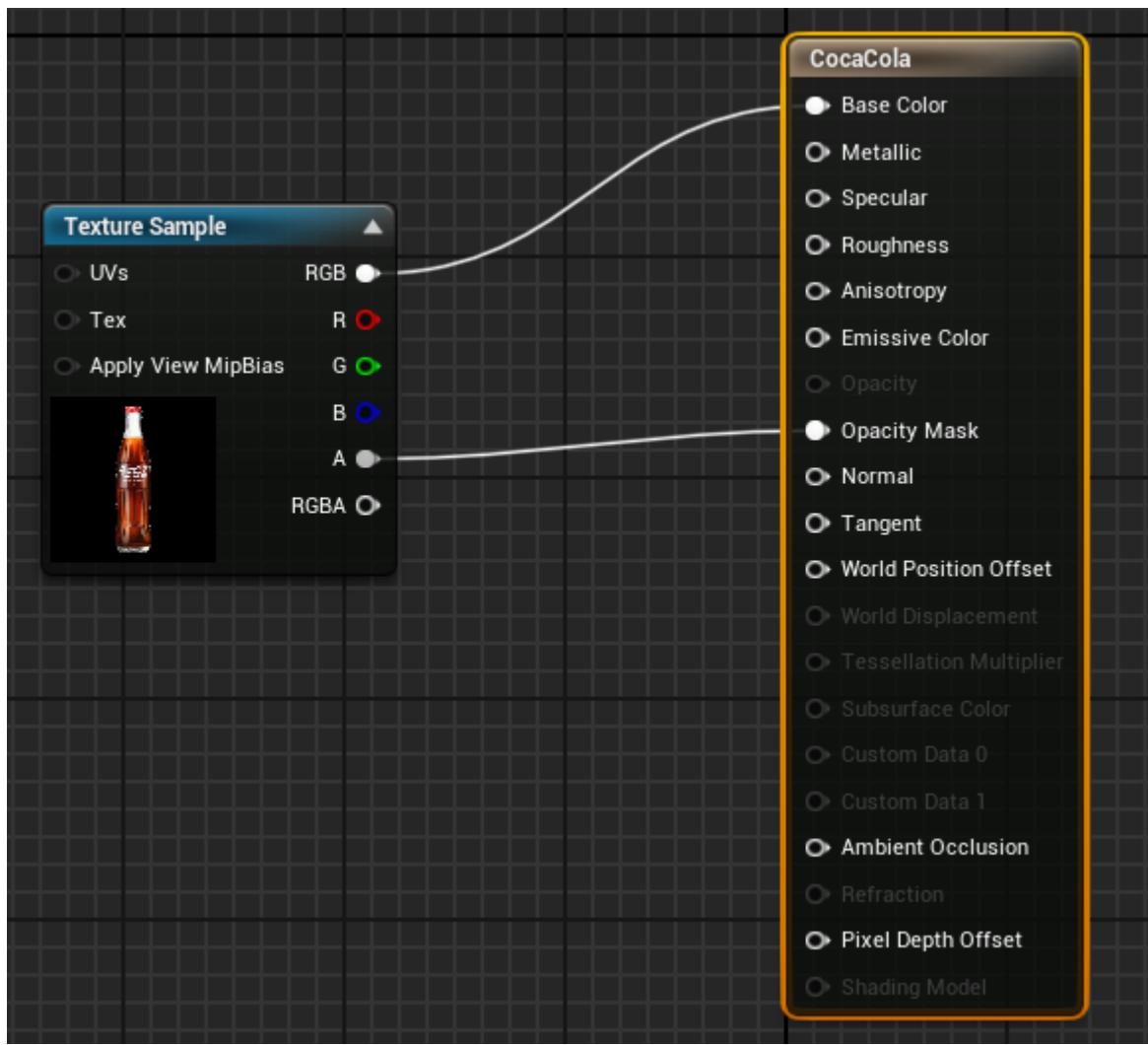


Básicamente, en este objeto cuando se produce el evento de Overlap, es decir, cuando el jugador atraviesa el objeto, se obtiene la referencia a nuestro personaje y se llama al evento Pcount que hemos definido en nuestro personaje para incrementar el número de balones de oro recogidos por el personaje. Después, vamos a reproducir un sonido que hemos importado donde Crisitano Ronaldo dice Balon de Oro. Más adelante, con el nodo Spawn Emitter at Location se reproduce un efecto de partículas de oro sobre el objeto cuando es recogido. Para aplicar este efecto de partículas hemos añadido el asset **Infinity Blade: Effects** a nuestro proyecto el cual se encuentra disponible para su descarga en este [enlace](#). Por último, se destruye el objeto pues ya ha sido recogido y no queremos que aparezca más.

## Objeto de daño “Coca-Cola”

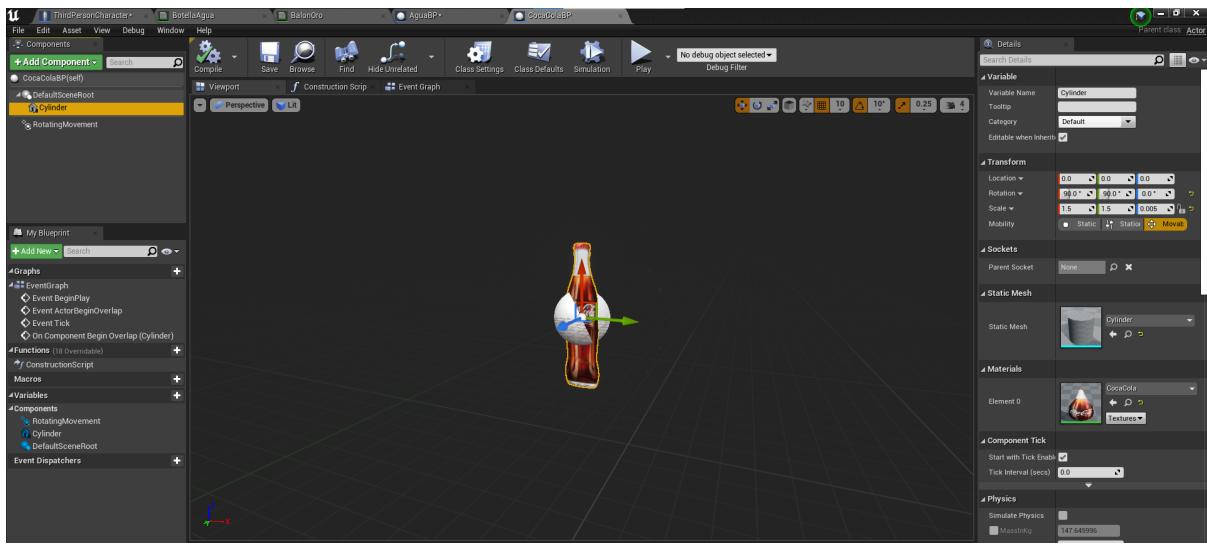
En segundo lugar, vamos a diseñar un objeto de daño de forma que cada vez que nuestro personaje coja este objeto, se quite parte de su vida. Como a nuestro personaje Crisitano Ronaldo no le gusta la Coca-Cola ya que tiene muchos azúcares y a él lo que le gusta es beber agua para mantenerse sano y en forma, este objeto o pick up en cuestión será una botella de Coca-Cola de forma que cada vez que el personaje coja una botella de Coca-Cola, se decrementaran los puntos de vida que tenga el jugador. Más adelante, también crearemos un menú HUD en el que se muestre de manera dinámica los puntos de vida que tiene el jugador.

Para crear este objeto, en primer lugar, vamos a diseñar la textura que utilizará este objeto. En nuestro caso, para crear la textura de la botella de Coca-Cola, nos descargaremos una imagen recortada sin fondo de una botella de Coca-Cola y después crearemos un material llamado CocaCola. Dentro de este material, crearemos una textura a partir de la imagen de la botella de Coca-Cola recortada y esa textura se aplicará a nuestro material.

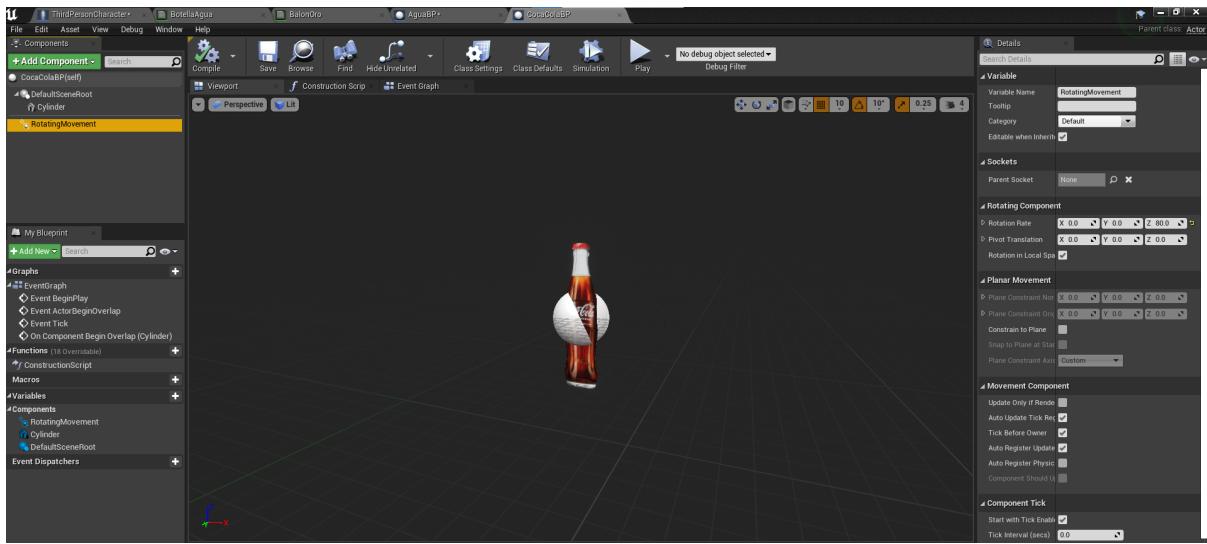


Para crear esta textura, se ha modificado el parámetro Blend Mode del material al valor masked de forma que se utiliza el valor alfa de nuestra fotografía para definir la máscara de opacidad del objeto y así conseguir un mejor textura para nuestro objeto.

En segundo lugar, crearemos un Blueprint Class de tipo Actor cuyo nombre será CocaColaBP. Dentro de este Blueprint Class, añadiremos un cilindro el cual giraremos 90 grados y dejaremos su componente Z casi a 0 para que tenga la forma de un círculo en 2D. Después, aplicaremos la textura de la botella de Coca-Cola que acabamos de crear para así tener nuestro objeto botella de Coca-Cola perfectamente diseñado.

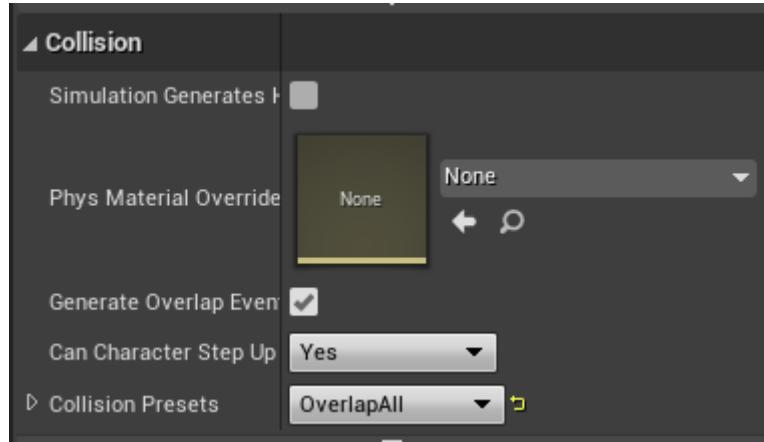


Ahora, podemos arrastrar el contenido de este Blueprint Class al escenario de forma que el objeto botella de Coca-Cola se visualiza perfectamente. Sin embargo, queremos que este objeto esté rotando continuamente para poder visualizarlo desde todos los ángulos (ya que es un objeto 2D) y para darle un componente de dinamismo al objeto. Para ello, vamos a añadir al Blueprint Class un componente RotatingMovement para establecer una rotación en el eje Z a nuestro objeto.

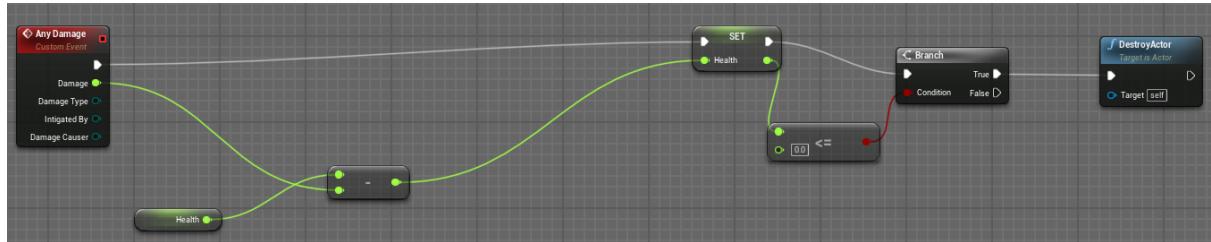


En nuestro caso, hemos elegido un tasa de rotación de 80 grados para el eje Z para que así el objeto se mueva a una velocidad moderada y se pueda visualizar sin ningún tipo de problema.

Una vez hecho esto, queremos que nuestro personaje pueda atravesar el objeto y recogerlo (ya que el objeto detecta las colisiones). Para ello, en primer lugar, vamos a configurar el objeto de forma que nuestro personaje pueda atravesarlo. Para ello, lo único que tenemos que hacer es acceder a nuestro Blueprint Class y en la sección de colisiones del objeto establecer el valor de OverlapAll.

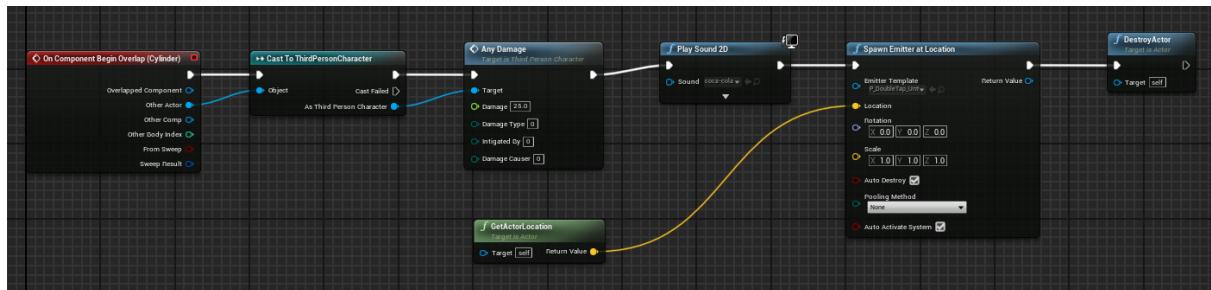


Con esto nuestro personaje puede atravesar el objeto. Ahora, queremos que cuando nuestro personaje atravesie el objeto, lo recoja y se decremente el valor de los puntos de vida del jugador. Para ello, en el Blueprint del personaje vamos a crear el siguiente código:



Básicamente con este código, definimos un evento customizado llamado Any Damage que se ejecutará desde el blueprint del objeto el cual modificará el valor de los puntos de vida del jugador. Este valor del número de puntos de vida será almacenado en una variable flotante llamada Health cual habremos creado para este cometido. Como se puede observar se obtienen los puntos de vida del personaje y se restan por el daño recibido por el personaje. Por último, se evalúa si el personaje tiene puntos de vida y en el caso de que no tenga, se destruirá el jugador (esta funcionalidad cambiará más adelante).

Por otra parte, en el blueprint del objeto Botella de Coca-Cola se ha utilizado el siguiente código:



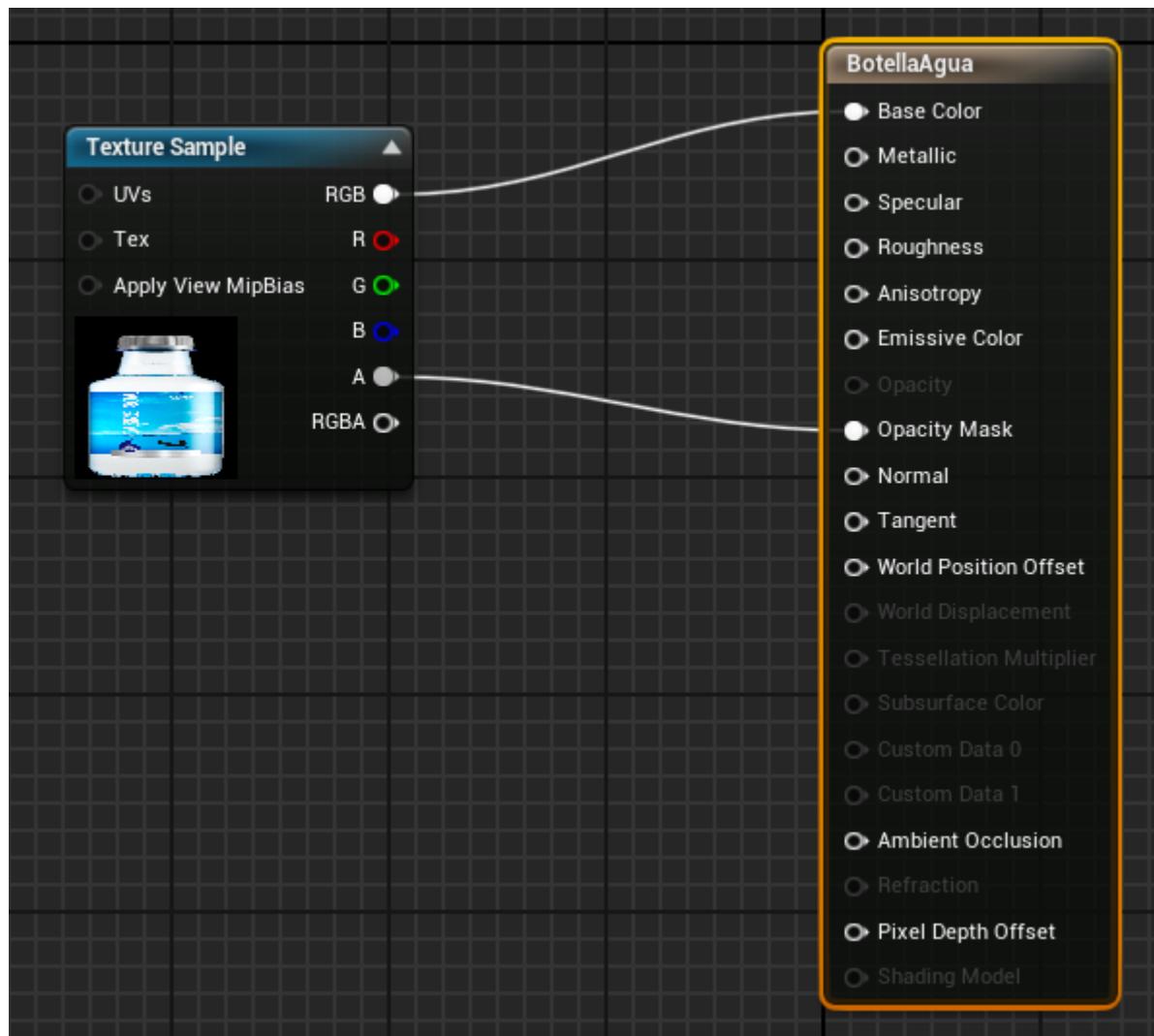
Básicamente, en este objeto cuando se produce el evento de Overlap, es decir, cuando el jugador atraviesa el objeto, se obtiene la referencia a nuestro personaje y se llama al evento Any Damage que hemos definido en nuestro personaje para decrementar el número de puntos de vida del personaje.

Después, vamos a reproducir un sonido que hemos importado donde Crisitano Ronaldo dice Coca-Cola. Más adelante, con el nodo Spawn Emitter at Location se reproduce un efecto de partículas de daño sobre el objeto cuando es recogido. Para aplicar este efecto de partículas hemos añadido el asset **Infinity Blade: Effects** a nuestro proyecto el cual se encuentra disponible para su descarga en este [enlace](#). Por último, se destruye el objeto pues ya ha sido recogido y no queremos que aparezca más.

## Objeto de curación “Agua”

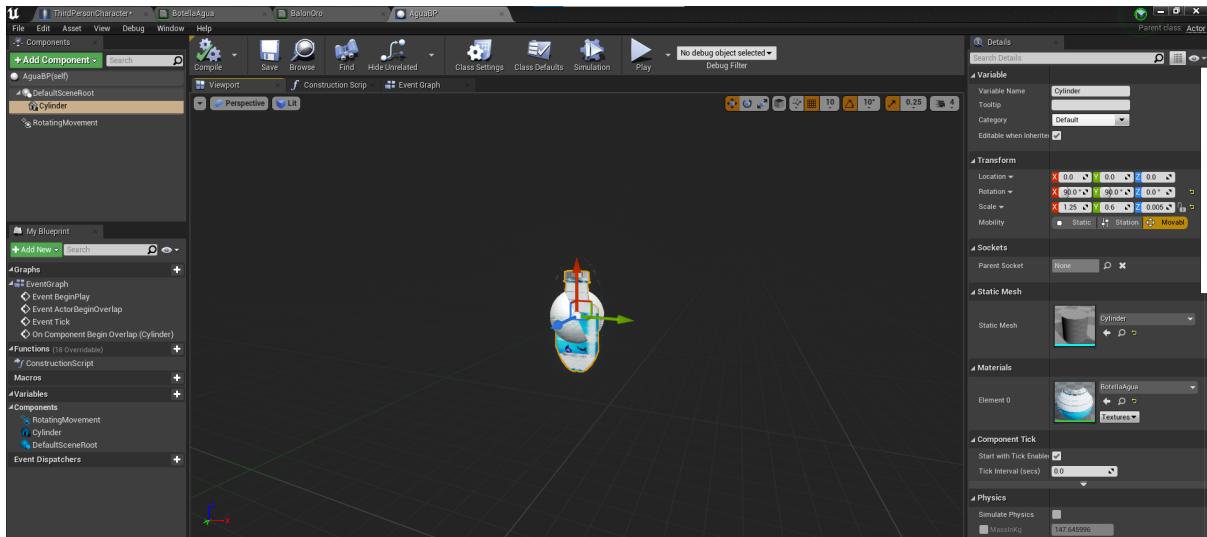
En tercer lugar, vamos a diseñar un objeto de curación de forma que cada vez que nuestro personaje coja este objeto, se regenere parte de su vida. Como a nuestro personaje Crisitano Ronaldo le gusta beber agua para mantenerse sano y en forma este objeto o pick up en cuestión será una botella de agua de forma que cada vez que el personaje coja una botella de agua, se incrementarán los puntos de vida que tenga el jugador. Más adelante, también crearemos un menú HUD en el que se muestre de manera dinámica los puntos de vida que tiene el jugador.

Para crear este objeto, en primer lugar, vamos a diseñar la textura que utilizará este objeto. En nuestro caso, para crear la textura de la botella de agua, nos descargaremos una imagen recortada sin fondo de una botella de agua y después crearemos un material llamado BotellaAgua. Dentro de este material, crearemos una textura a partir de la imagen de la botella de agua recortada y esa textura se aplicará a nuestro material.

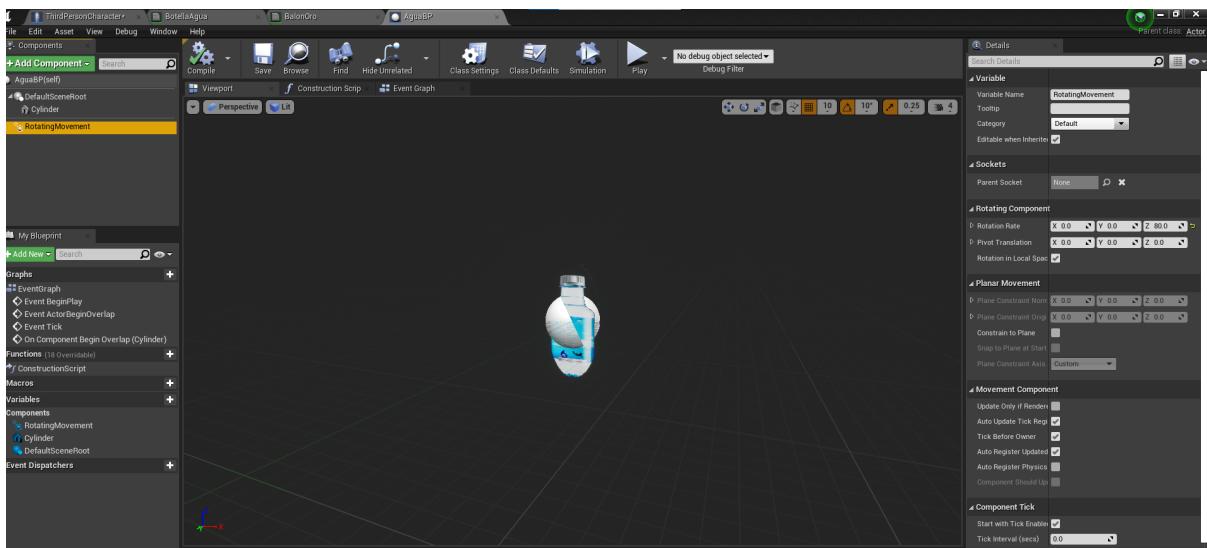


Para crear esta textura, se ha modificado el parámetro Blend Mode del material al valor masked de forma que se utiliza el valor alfa de nuestra fotografía para definir la máscara de opacidad del objeto y así conseguir un mejor textura para nuestro objeto.

En segundo lugar, crearemos un Blueprint Class de tipo Actor cuyo nombre será AguabP. Dentro de este Blueprint Class, añadiremos un cilindro el cual giraremos 90 grados y dejaremos su componente Z casi a 0 para que tenga la forma de un círculo en 2D. Después, aplicaremos la textura de la botella de agua que acabamos de crear para así tener nuestro objeto botella de agua perfectamente diseñado.

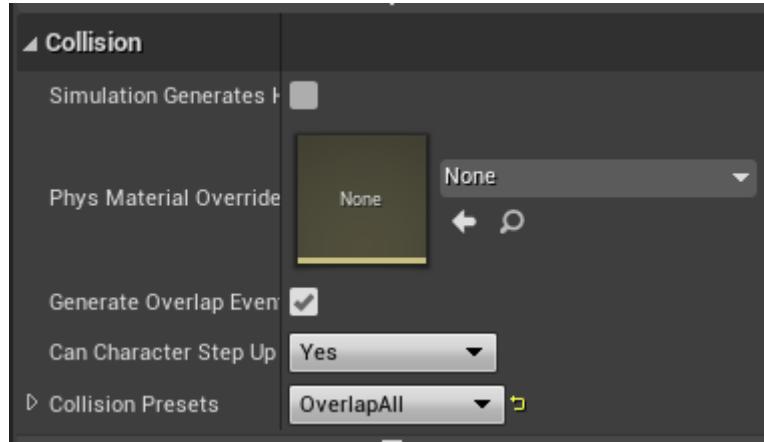


Ahora, podemos arrastrar el contenido de este Blueprint Class al escenario de forma que el objeto botella de agua se visualiza perfectamente. Sin embargo, queremos que este objeto esté rotando continuamente para poder visualizarlo desde todos los ángulos (ya que es un objeto 2D) y para darle un componente de dinamismo al objeto. Para ello, vamos a añadir al Blueprint Class un componente RotatingMovement para establecer una rotación en el eje Z a nuestro objeto.

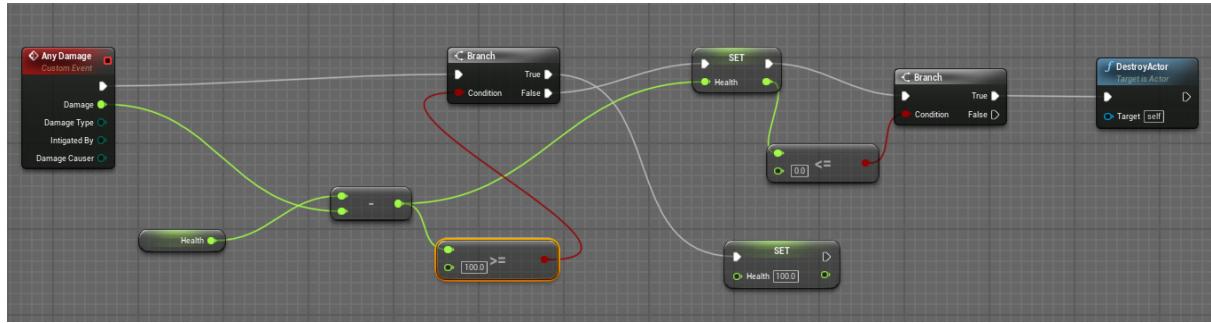


En nuestro caso, hemos elegido un tasa de rotación de 80 grados para el eje Z para que así el objeto se mueva a una velocidad moderada y se pueda visualizar sin ningún tipo de problema.

Una vez hecho esto, queremos que nuestro personaje pueda atravesar el objeto y recogerlo (ya que el objeto detecta las colisiones). Para ello, en primer lugar, vamos a configurar el objeto de forma que nuestro personaje pueda atravesarlo. Para ello, lo único que tenemos que hacer es acceder a nuestro Blueprint Class y en la sección de colisiones del objeto establecer el valor de OverlapAll.



Con esto nuestro personaje puede atravesar el objeto. Ahora, queremos que cuando nuestro personaje atravesese el objeto, lo recoja y se incremente el valor de los puntos de vida del jugador. Para ello, en el Blueprint del personaje vamos a crear el siguiente código:

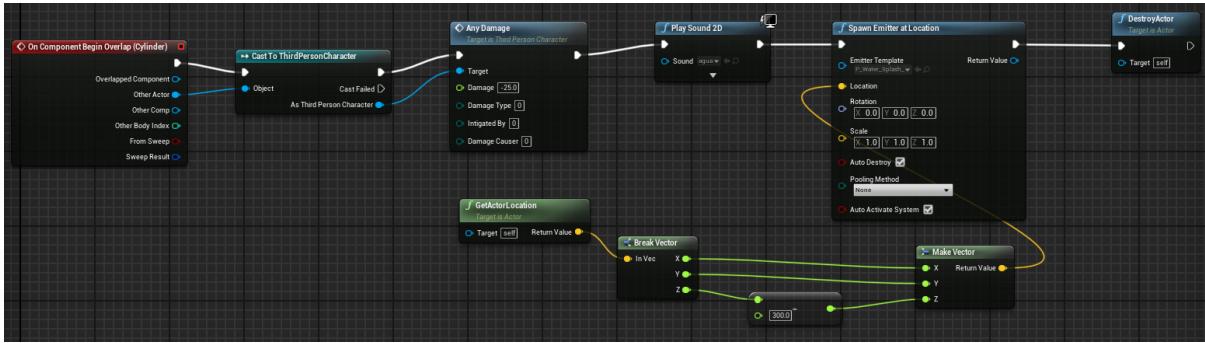


Básicamente con este código, definimos un evento customizado llamado Any Damage que se ejecutará desde el blueprint del objeto el cual modificará el valor de los puntos de vida del jugador. Este valor del número de puntos de vida será almacenado en una variable flotante llamada Health cual habremos creado para este cometido. Como se puede observar se obtienen los puntos de vida del personaje y se restan por el daño recibido por el personaje. En nuestro caso, como es un objeto de curación, se utilizarán puntos de daño negativos por lo que esta resta se convertirá en una suma.

Al aumentar los puntos de vida se compara el valor de la resta (en este caso suma por el daño negativo) con el valor máximo de vida del personaje con el objetivo de que el personaje no pueda superar este límite de 100 puntos de vida. En el caso de que se superen los puntos de vida máximo, se asignará a la variable Health el valor máximo de 100 puntos de vida.

Como se puede observar, hemos sido capaces de gestionar tanto el objeto de curación como el de daño utilizando el mismo evento y utilizando distintos caminos de ejecución en función de si se aumentan los puntos de vida o se disminuyen.

Por otra parte, en el blueprint del objeto Botella de Agua se ha utilizado el siguiente código:



Básicamente, en este objeto cuando se produce el evento de Overlap, es decir, cuando el jugador atraviesa el objeto, se obtiene la referencia a nuestro personaje y se llama al evento Any Damage que hemos definido en nuestro personaje para incrementar el número de puntos de vida del personaje. Obsérvese cómo en este caso introducimos los puntos de vida a sumar en negativo para que la resta se convierta en una suma como hemos dicho antes.

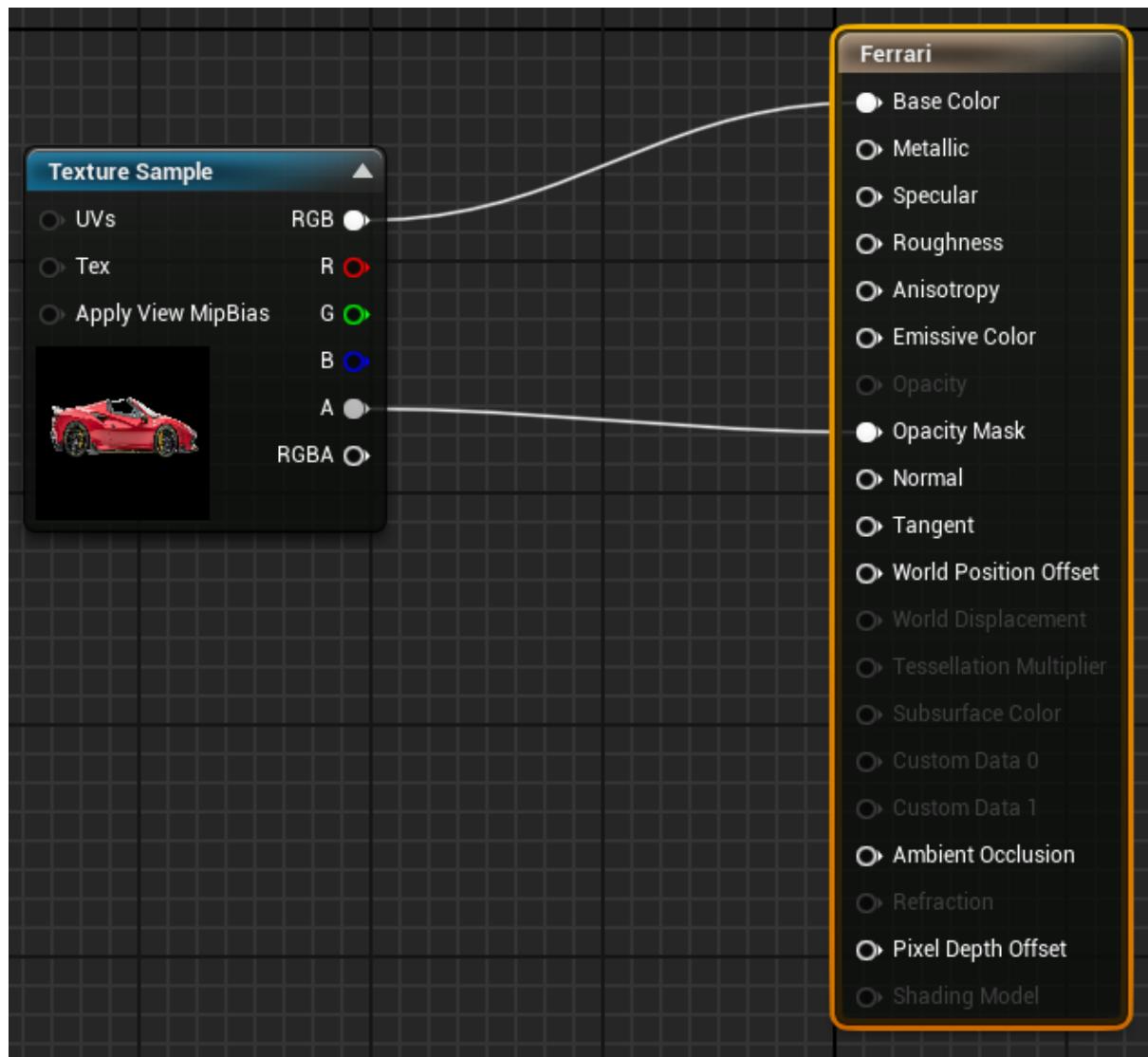
Después, vamos a reproducir un sonido que hemos importado donde Crisitano Ronaldo dice agua. Más adelante, con el nodo Spawn Emitter at Location se reproduce un efecto de partículas de agua sobre el objeto cuando es recogido. Para aplicar este efecto de partículas hemos añadido el asset **Infinity Blade: Effects** a nuestro proyecto el cual se encuentra disponible para su descarga en este [enlace](#). Por último, se destruye el objeto pues ya ha sido recogido y no queremos que aparezca más.

Obsérvese cómo se ha editado el valor Z en el cual se va a reproducir el efecto de partículas de agua para que así las partículas de agua no vayan tan arriba en el mapa y el efecto sea más bonito de ver.

## Objeto de velocidad “Ferrari”

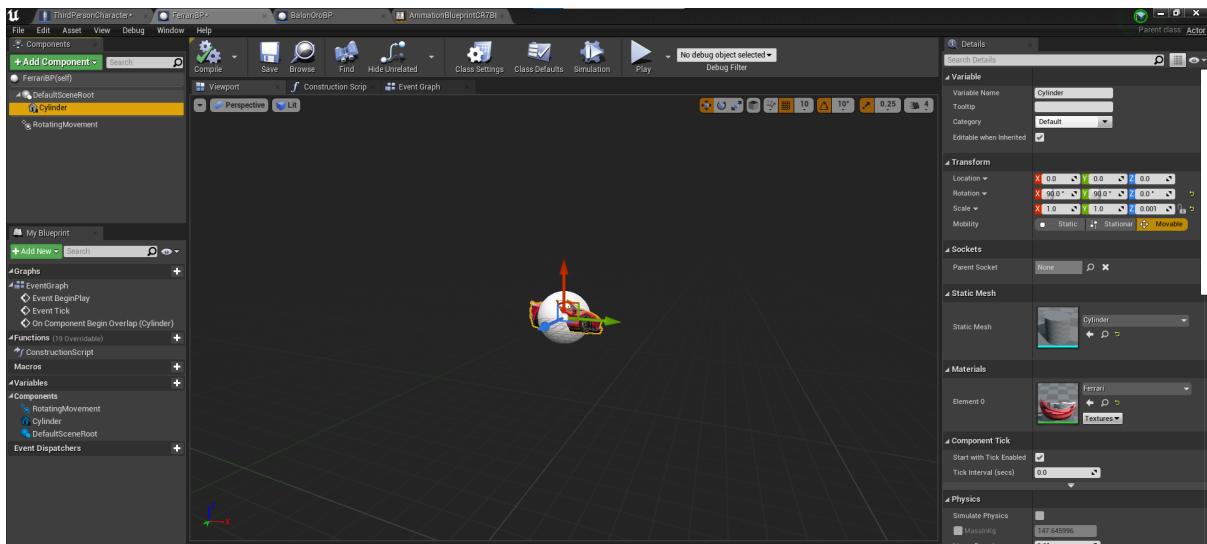
En cuarto lugar, vamos a diseñar un objeto de velocidad de forma que cada vez que nuestro personaje coja este objeto, se incremente al doble la velocidad del jugador. Como a nuestro personaje Crisitano Ronaldo le gusta mucho invertir su dinero en coches de lujo como Ferraris o Lamborghinis este objeto o pick up en cuestión será un Ferrari de forma que cada vez que el personaje coja un Ferrari, se duplicará la velocidad del jugador. Más adelante, también crearemos un menú HUD un ícono para que nos muestre cuando se nos ha duplicado la velocidad.

Para crear este objeto, en primer lugar, vamos a diseñar la textura que utilizará este objeto. En nuestro caso, para crear la textura del Ferrari, nos descargaremos una imagen recortada sin fondo de un Ferrari y después crearemos un material llamado Ferrari. Dentro de este material, crearemos una textura a partir de la imagen del Ferrari recortada y esa textura se aplicará a nuestro material.

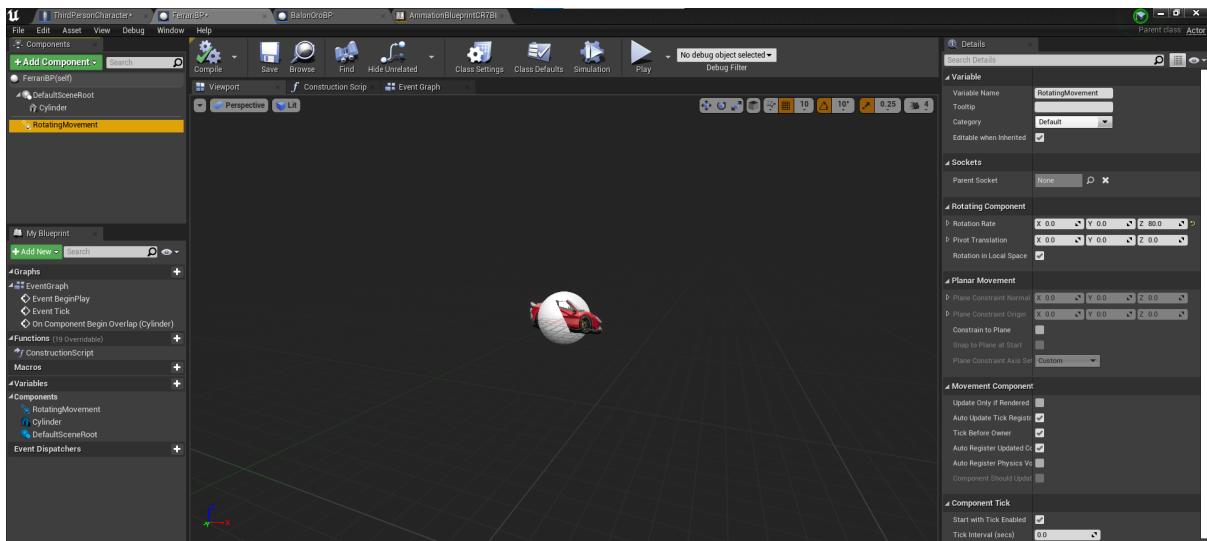


Para crear esta textura, se ha modificado el parámetro Blend Mode del material al valor masked de forma que se utiliza el valor alfa de nuestra fotografía para definir la máscara de opacidad del objeto y así conseguir un mejor textura para nuestro objeto.

En segundo lugar, crearemos un Blueprint Class de tipo Actor cuyo nombre será FerrariBP. Dentro de este Blueprint Class, añadiremos un cilindro el cual giraremos 90 grados y dejaremos su componente Z casi a 0 para que tenga la forma de un círculo en 2D. Después, aplicaremos la textura del Ferrari que acabamos de crear para así tener nuestro objeto botella de agua perfectamente diseñado.

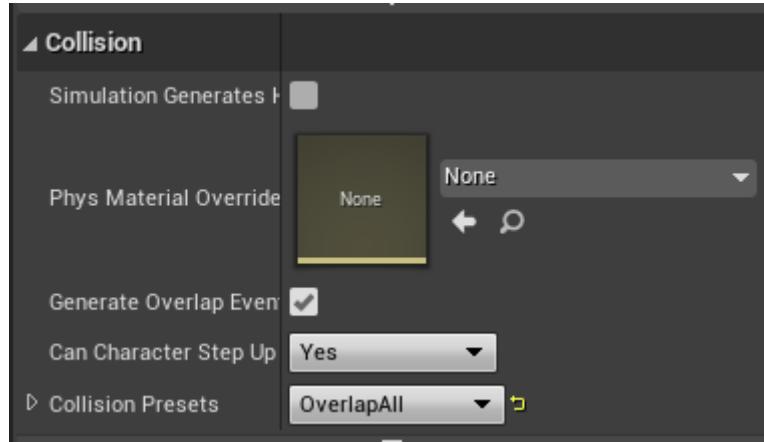


Ahora, podemos arrastrar el contenido de este Blueprint Class al escenario de forma que el objeto Ferrari se visualiza perfectamente. Sin embargo, queremos que este objeto esté rotando continuamente para poder visualizarlo desde todos los ángulos (ya que es un objeto 2D) y para darle un componente de dinamismo al objeto. Para ello, vamos a añadir al Blueprint Class un componente RotatingMovement para establecer una rotación en el eje Z a nuestro objeto.

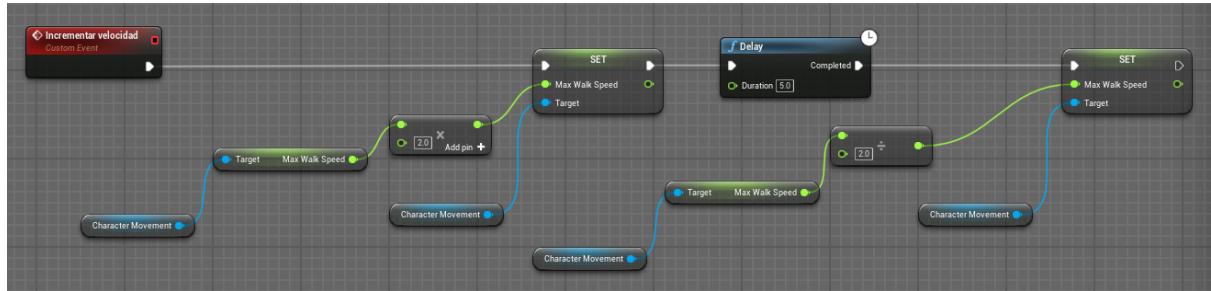


En nuestro caso, hemos elegido una tasa de rotación de 80 grados para el eje Z para que así el objeto se mueva a una velocidad moderada y se pueda visualizar sin ningún tipo de problema.

Una vez hecho esto, queremos que nuestro personaje pueda atravesar el objeto y recogerlo (ya que el objeto detecta las colisiones). Para ello, en primer lugar, vamos a configurar el objeto de forma que nuestro personaje pueda atravesarlo. Para ello, lo único que tenemos que hacer es acceder a nuestro Blueprint Class y en la sección de colisiones del objeto establecer el valor de OverlapAll.

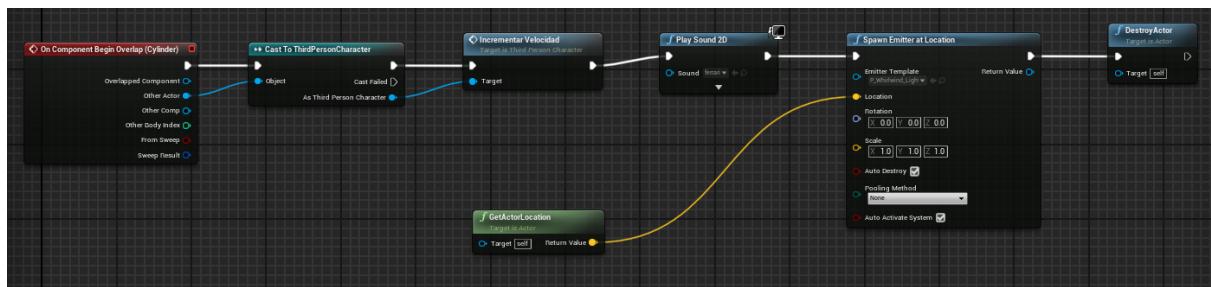


Con esto nuestro personaje puede atravesar el objeto. Ahora, queremos que cuando nuestro personaje atravesese el objeto, lo recoja y se duplique el valor de la velocidad del jugador. Para ello, en el Blueprint del personaje vamos a crear el siguiente código:



Básicamente con este código, definimos un evento customizado llamado Incrementar Velocidad que se ejecutará desde el blueprint del objeto el cual modificará la velocidad máxima del jugador. De esta manera, se obtendrá el valor de la velocidad máxima del jugador, y se multiplicara este valor de la velocidad máxima por 2. Una vez se ha obtenido el nuevo valor de la velocidad máxima se asignará este valor a la velocidad máxima del personaje. Este aumento de la velocidad durará 5 segundos y cuando acabe el delay se restablecerá el valor de la velocidad máxima del personaje a su valor original.

Por otra parte, en el blueprint del objeto Ferrari se ha utilizado el siguiente código:

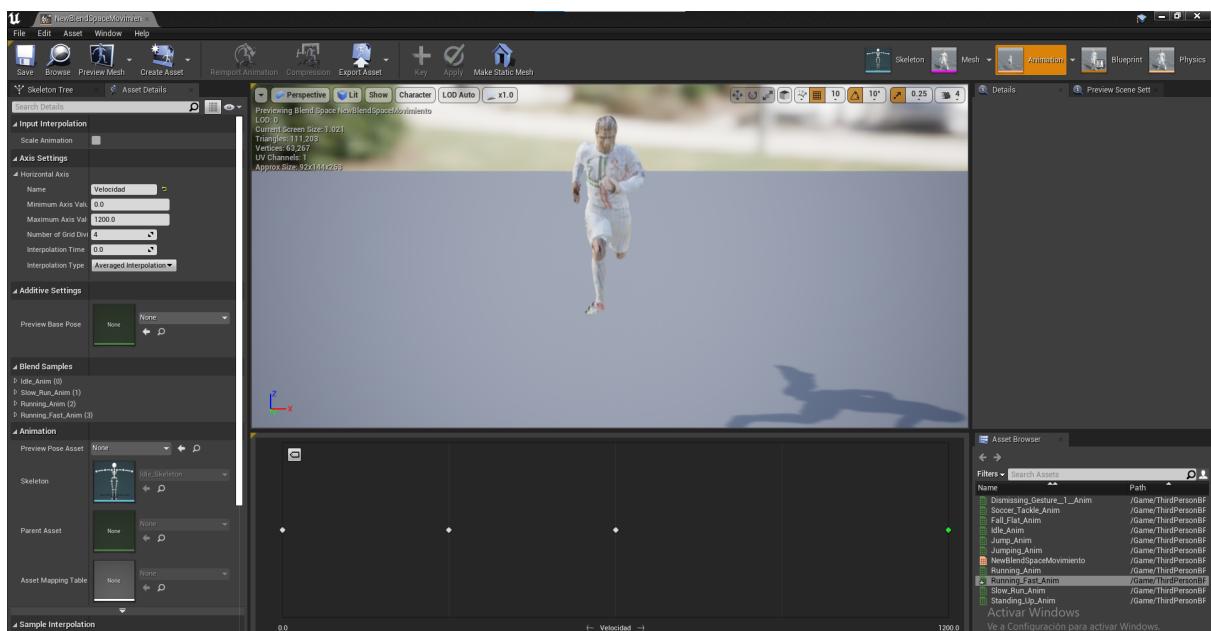


Básicamente, en este objeto cuando se produce el evento de Overlap, es decir, cuando el jugador atraviesa el objeto, se obtiene la referencia a nuestro personaje y se llama al evento

Incrementar Velocidad que hemos definido en nuestro personaje para duplicar la velocidad del personaje.

Después, vamos a reproducir un sonido con la aceleración de un coche Ferrari. Más adelante, con el nodo Spawn Emitter at Location se reproduce un efecto de partículas de viento sobre el objeto cuando es recogido. Para aplicar este efecto de partículas hemos añadido el asset **Infinity Blade: Effects** a nuestro proyecto el cual se encuentra disponible para su descarga en este [enlace](#). Por último, se destruye el objeto pues ya ha sido recogido y no queremos que aparezca más.

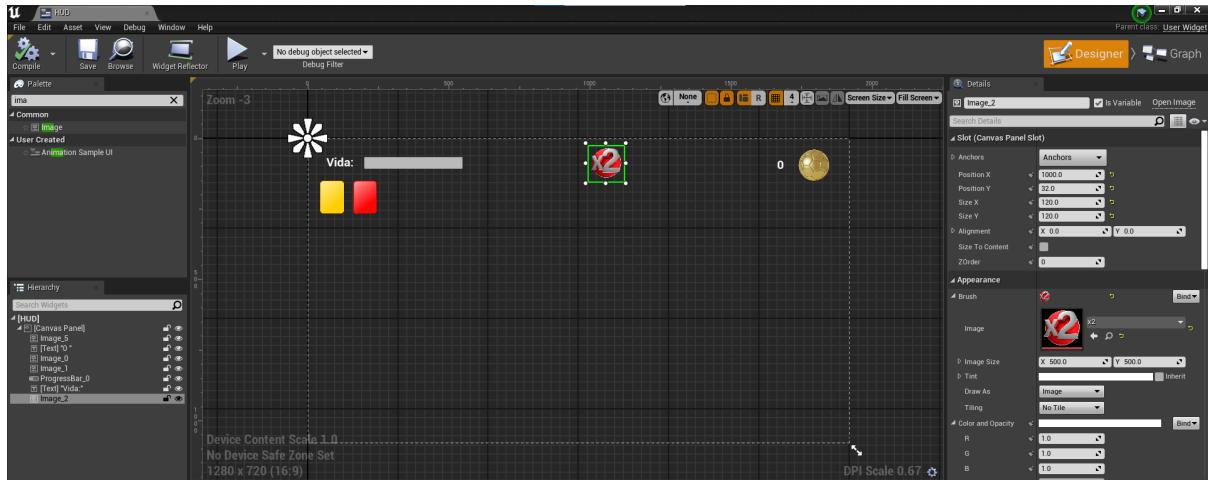
De manera adicional, vamos a añadir una animación extra llamada Running Fast para que se reproduzca cuando el personaje tiene el doble de velocidad. Esta animación es similar a la animación normal de correr pero el movimiento de las extremidades es el doble de rápido. Para conseguir reproducir esta animación, cuando nuestro personaje haya cogido el Ferrari, nos vamos a dirigir al Blend de animaciones que creamos anteriormente y vamos a aumentar el valor máximo de la línea horizontal de 600 a 1200 y en el valor de 1200 vamos a incluir la animación de Running Fast. De esta manera, cuando Cristiano Ronaldo coja el Ferrari, su valor de velocidad máxima pasará de 600 a 1200 y por tanto se ejecutará la animación de Running Fast.



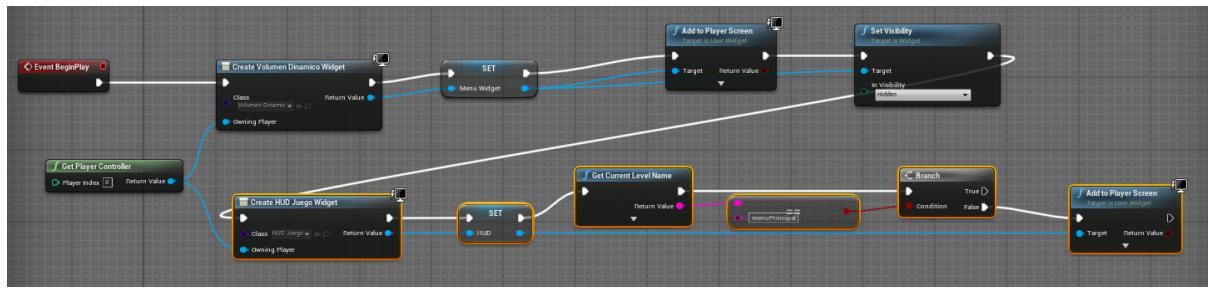
## Creación de HUD

Una vez hemos creado los distintos objetos que puede coger el Jugador, el siguiente paso que vamos a dar es crear un menú HUD el cual va a indicar de manera dinámica distinta información, como el número de balones de oro recogidos, los puntos de vida del jugador o si el jugador está en estado de doble velocidad.

Para ello, en primer lugar, vamos a crear un Blueprint Widget y dentro de este Blueprint Widget vamos a incluir toda la información que deseamos mostrar con este menú HUD. El aspecto de nuestro HUD será el siguiente:

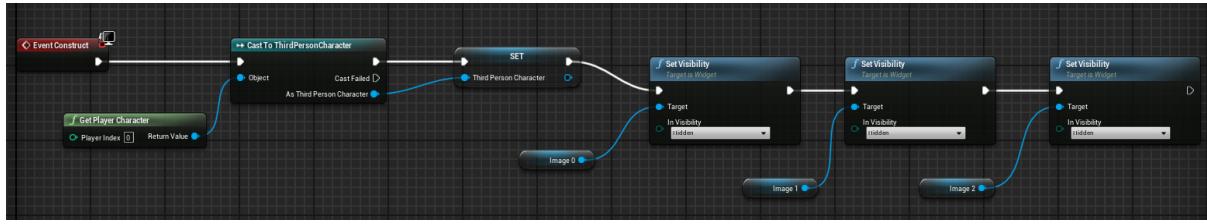


Para mostrar el contenido del HUD cuando se reproduzca el primer nivel del juego, añadiremos una porción de código en el Blueprint del personaje.



Esta modificación del Blueprint, después de ajustar el input del juego, vamos a crear el menú HUD del juego con el nodo Create HUD Widget y después lo vamos a sacar por la pantalla a través del nodo Add To Player Screen. Sin embargo, al crear el HUD desde el personaje, el menú HUD también se dibujara en el nivel menuPrincipal ya que ese nivel cuenta con un ThirdPersonCharacter. Debido a esto, en este código obtenemos el nombre del nivel actual para que en caso de que si el nivel que se esté ejecutando es el menuPrincipal que no se muestre el HUD por la pantalla. Esta solución no es la mejor (podríamos haber creado el HUD en el blueprint del nivel) pero no nos queda más remedio que utilizarla ya que luego necesitamos acceder a determinados eventos del HUD desde el personaje y esto solo va a ser posible si tenemos directamente la referencia del HUD en el personaje.

Si ahora ejecutamos el juego, todos los objetos que se han incluido en el HUD se visualizarán cuando realmente queremos que algunos objetos permanezcan ocultos hasta que se produzca un evento determinado. En concreto, queremos eliminar las imágenes de las tarjetas y del símbolo de x2. Para ello, incluiremos el siguiente código en el blueprint del HUD:

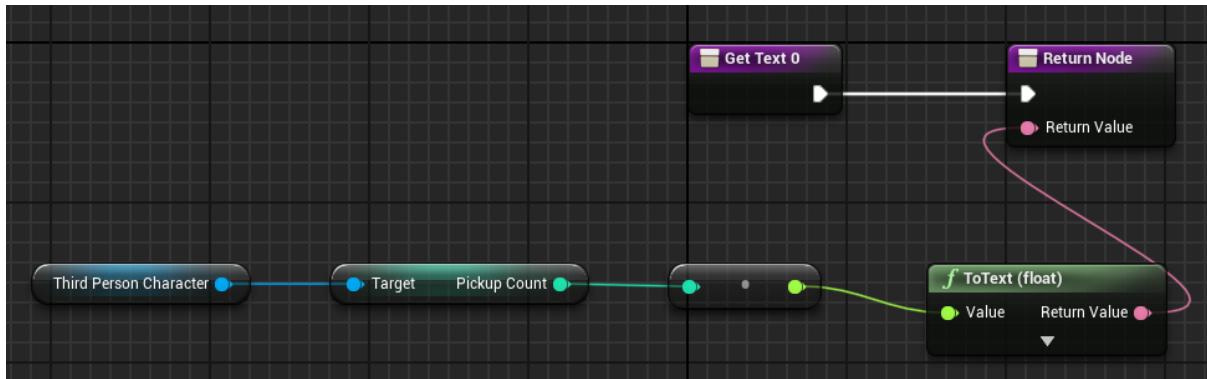


Con este código, en primer lugar obtenemos una referencia de nuestro personaje a través de los nodos Cast to ThirdPersonCharacter y Get Player Character y almacenamos esta referencia en una variable. Esto lo hacemos porque posteriormente el HUD utilizará variables de este personaje. Por último, a través de Set Visibility ocultamos las imágenes de las tarjetas y del x2.

Ahora, nos interesa que cada vez que nuestro personaje recoja un balón de oro, se incremente en el HUD el texto que lleva la cuenta de los balones de oro recogidos por el personaje. Para hacer esto, en el HUD vamos a hacer bind de este bloque de texto que lleva la cuenta de los balones de oro.



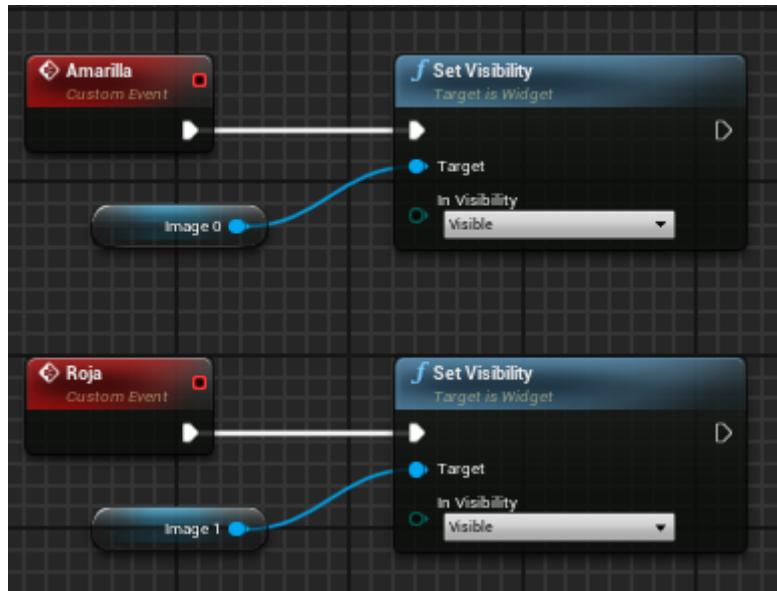
Al hacer este bind, se va a crear una función con la que vamos a determinar cuál debe ser el contenido de este bloque de texto en cada momento. Para controlar el contenido de este bloque de texto se utilizará el siguiente código:



Básicamente, en este código, obtenemos la variable Pickup Count de nuestro personaje, la cual lleva la cuenta de los balones de oro recogidos, y lo convertimos a texto a través del nodo To Text.

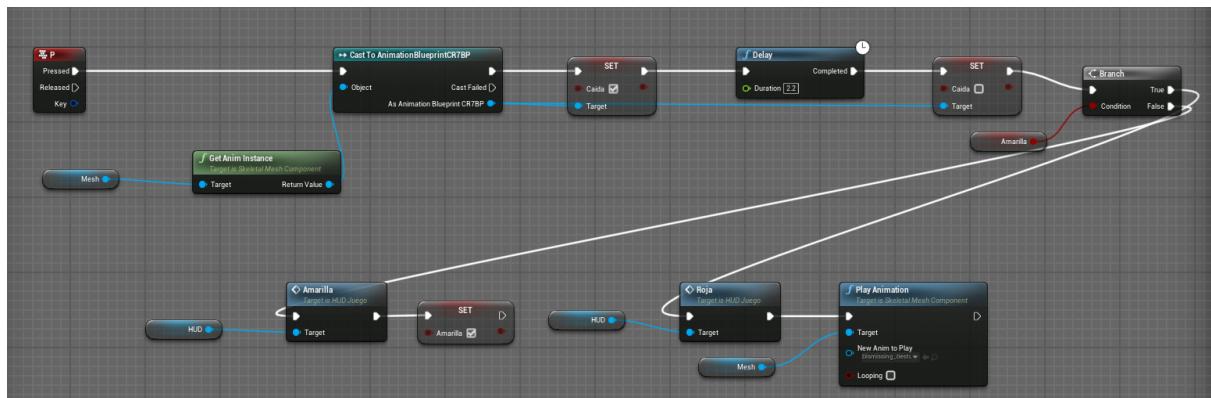
Ahora, vamos a gestionar la aparición de las tarjetas en el HUD. Anteriormente, en el diseño del personaje creamos la posibilidad de que Cristiano Ronaldo pudiera tirarse simulando un penalti con la tecla P. Debido a esto, hemos decidido que cada vez que simule Cristiano Ronaldo un penalti, nuestro personaje será sancionado con una tarjeta amarilla. Por lo tanto, la primera vez que Cristiano Ronaldo simule un penalti recibirá tarjeta amarilla, y si simula un penalti una segunda vez recibirá una segunda amarilla por lo que será expulsado

y el juego finalizará. Para controlar esta situación se definirán los siguientes eventos en el HUD:



Con el evento Amarilla, hacemos visible la imagen de la tarjeta amarilla y con el evento Roja, hacemos visible la imagen de la tarjeta roja.

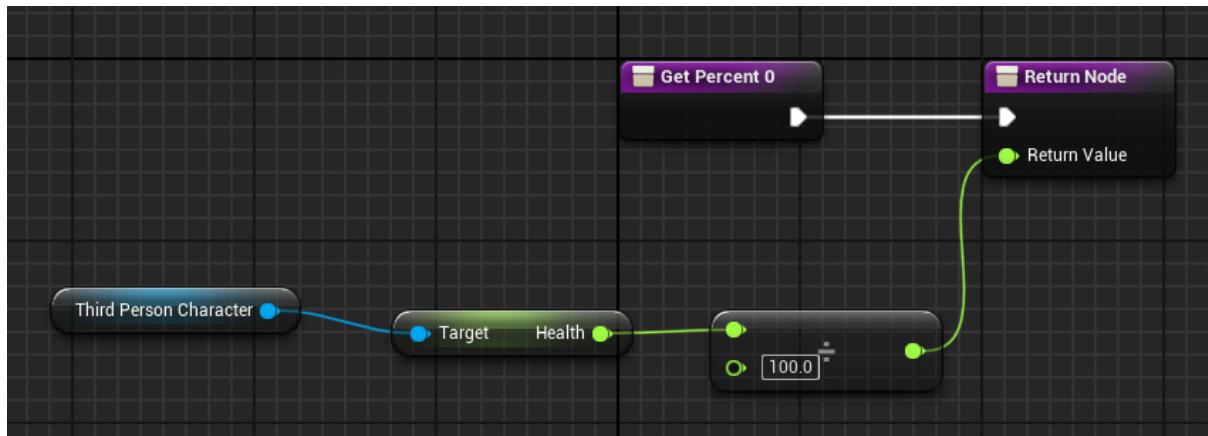
Estos eventos Amarilla y Roja serán llamados desde el blueprint del personaje dentro del código en el que gestionamos la simulación del personaje. La gestión de las tarjetas en el blueprint del personaje es el siguiente:



Como se puede observar, hemos añadido código para la gestión de evento de pulsación de la tecla P. Básicamente este código nuevo se basa en la creación de una nueva variable booleana llamada Amarilla de forma que si esta variable es falsa, se activa el evento Amarilla para sacar tarjeta amarilla al personaje y se cambia el valor de la variable Amarilla a verdadero y si esta variable Amarilla es verdadera, se activa el evento Roja para sacar tarjeta roja al personaje y se reproduce una animación de disconformidad.

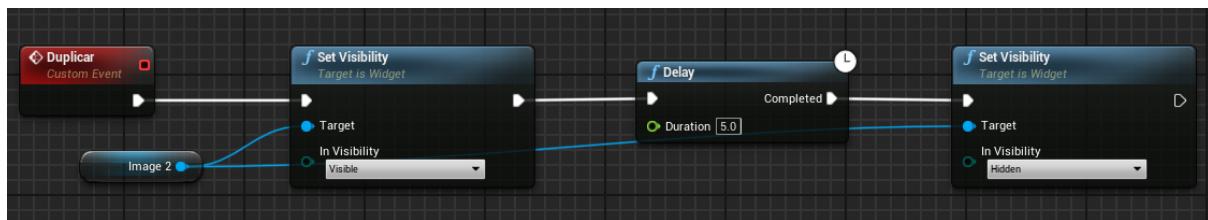
Una vez hecho esto, lo siguiente que vamos a hacer es gestionar la barra de vida del personaje. Para ello, vamos a proceder de manera similar a cómo gestionábamos los balones de oro y vamos a hacer un binding de la Progress Bar que gestiona la vida del

personaje. De esta manera, vamos a asociar el porcentaje de esta barra de progreso a una función en la que vamos a determinar el valor de esta barra de vida a través de los puntos de vida del personaje. El código de esta función sería el siguiente:



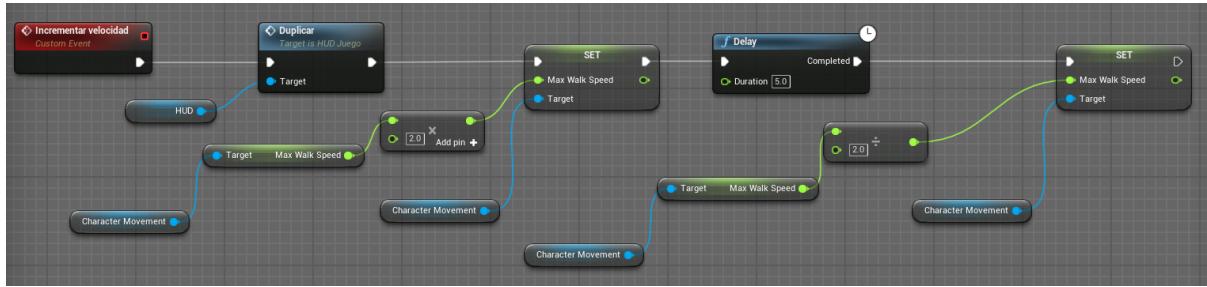
Básicamente, en este código, se obtiene una referencia a nuestro personaje, y se obtiene el valor de los puntos de vida del personaje. Estos puntos de vida del personaje se dividirán entre 100 que son los puntos máximos de vida del personaje obteniéndose así el porcentaje de vida del personaje que debe mostrarse en la barra de progreso.

Por último, vamos a gestionar la aparición de la imagen x2 en el HUD. Anteriormente, en el diseño del personaje creamos la posibilidad de que Cristiano Ronaldo pudiera correr al doble de velocidad durante 5 segundos si cogía un objeto Ferrari. Debido a esto, hemos decidido que cada vez que Cristiano Ronaldo coja un objeto Ferrari, se muestre la imagen indicando que nuestro personaje tiene el doble de velocidad. Para controlar esta situación se definirán los siguientes eventos en el HUD:



Con el evento Duplicar, hacemos visible la imagen del x2 y cuando transcurran 5 segundos y finalice el efecto del Ferrari, volvemos a ocultar la imagen.

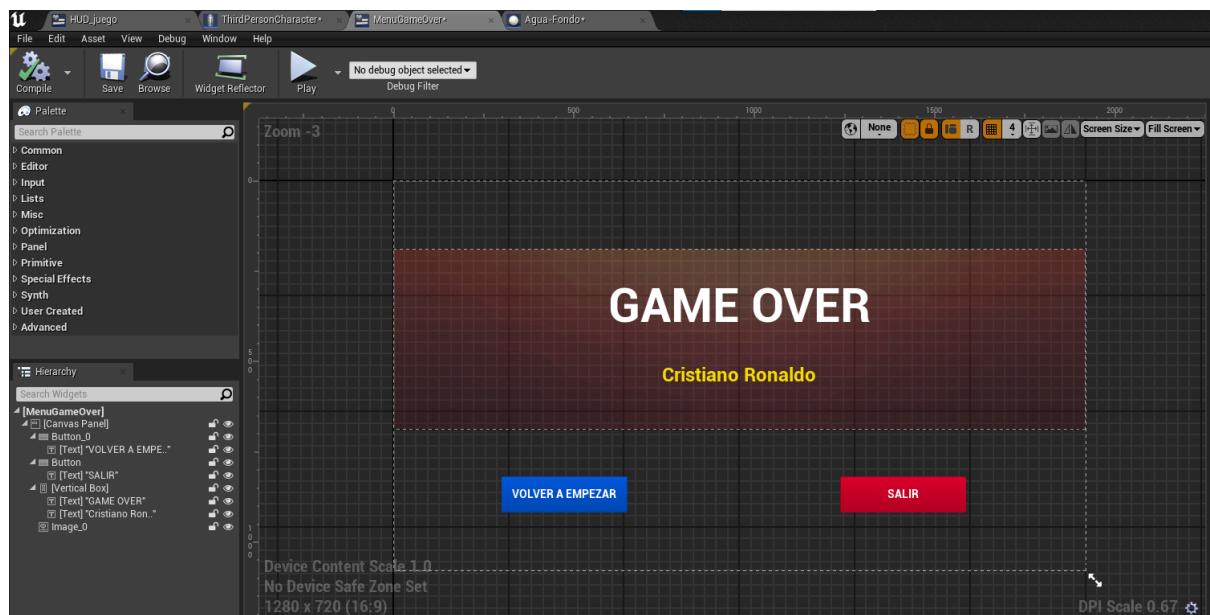
Este evento Duplicar será llamado desde el blueprint del personaje dentro del código en el que gestionamos la recolección del objeto Ferrari.



Como se puede observar, lo único que hemos añadido es un llamada al evento Duplicar que nos muestra la foto de x2 durante un tiempo, antes de incrementar la velocidad del personaje. El resto del código permanece igual.

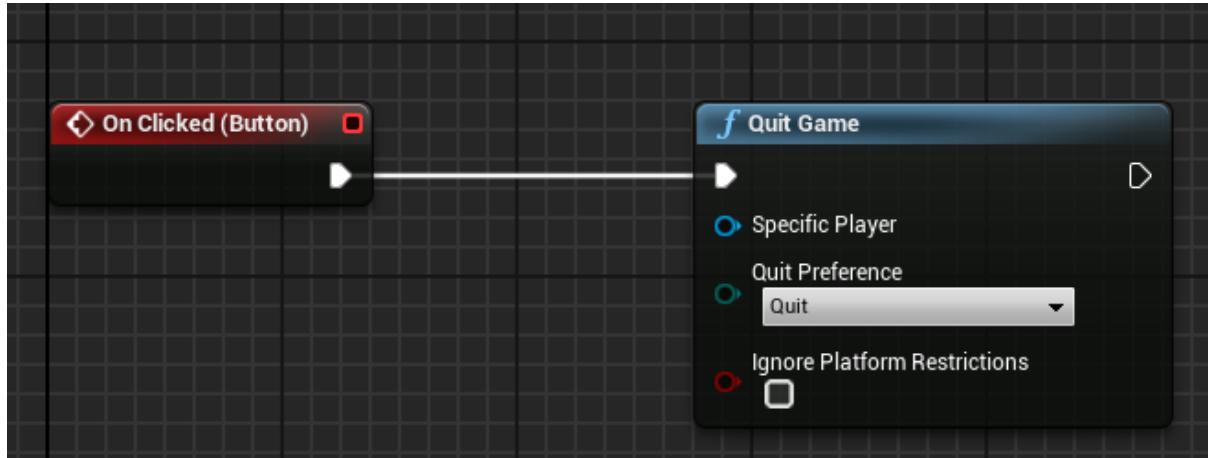
## Creación del menú GameOver

Una vez hemos creado y diseñado el HUD de nuestro juego, vamos a crear un menú de Game Over el cual queremos que se despliegue cuando nuestro personaje pierda en el juego. Para ello, como en casos anteriores, vamos a crear un Widget Blueprint el cual va a tener el siguiente aspecto:

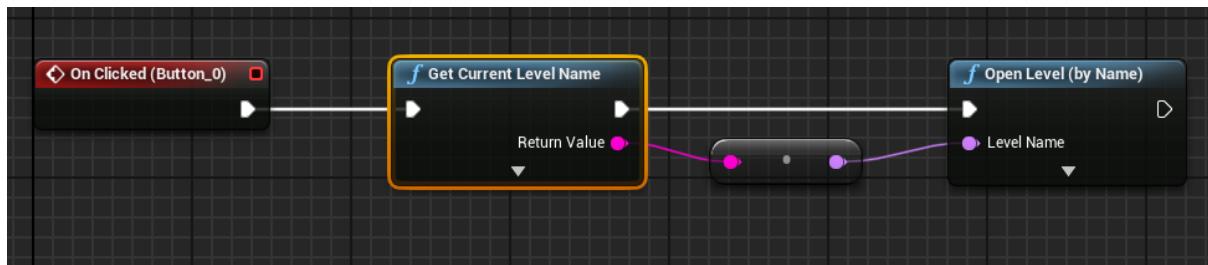


Como se puede observar, este menú cuenta con 2 botones cuya funcionalidad se va a programar a continuación. También contará con un texto en el que se explicará la razón por la cual nuestro personaje ha perdido en el juego. Este menú de Game Over cuenta con una imagen con un fondo rojo para darle más “ambientación” al menú de Game Over.

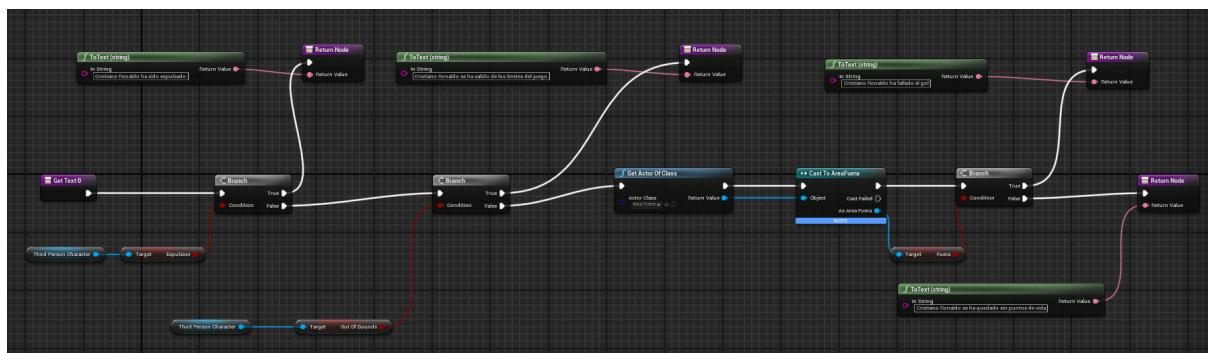
Para programar el botón de Salir simplemente tenemos que registrar el evento On Clicked en el panel derecho en la sección de Eventos. Cuando se produzca este evento, ejecutaremos un nodo llamado Quit Game el cual nos sacará del juego.



Para programar el botón de Volver a empezar simplemente tenemos que registrar el evento On Clicked en el panel derecho en la sección de Eventos. Cuando se produzca este evento, obtendremos el nombre del nivel actual con el nodo Get Current Level Name y utilizaremos este nombre del nivel actual para volver a cargar el nivel actual con el nodo Open Level.



Para gestionar y mostrar en pantalla el motivo por el cual nuestro personaje ha perdido en el juego, vamos a hacer un binding al bloque de texto de color amarillo y vamos a crear una función que va a definir el contenido de este bloque de texto.

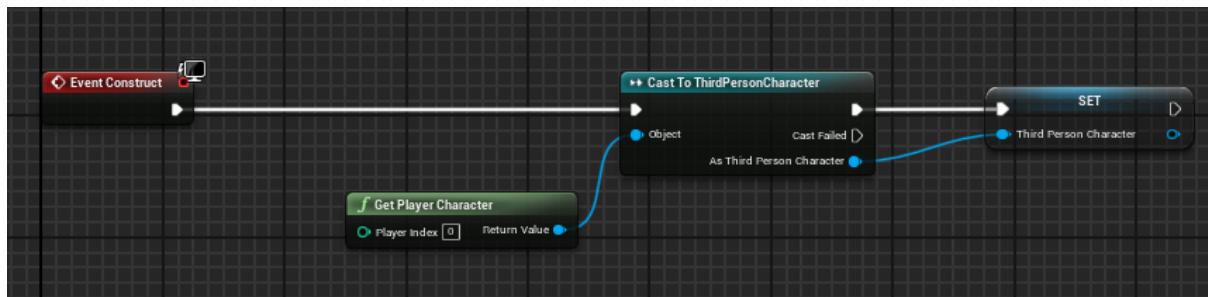


En nuestro juego, hay 4 motivos por los que nuestro jugador puede perder. El primer motivo es que le saquen tarjeta roja por simular un penalti 2 veces. El segundo motivo es que nuestro personaje salga de los límites del mapa. El tercer motivo es que nuestro jugador dispare fuera de la portería (más adelante hablaremos sobre esta funcionalidad). El cuarto motivo es que nuestro jugador se quede sin puntos de vida.

Una vez hemos definido las posibles maneras en las que nuestro personaje puede perder, vamos a crear y hacer uso de 3 variables booleanas las cuales nos van a ayudar a definir por qué motivo ha perdido nuestro jugador. La primera variable booleana se llama Expulsión. Esta variable se activará cuando nuestro personaje reciba tarjeta roja. La segunda variable booleana se llama Out Of Bounds. Esta variable se activará cuando nuestro personaje se salga de los límites del nivel. La tercera variable booleana se llama Fuera. Esta variable se activará cuando nuestro personaje falle el disparo a portería. Si ninguna de estas 3 variables booleanas se ha activado, entonces la razón del gameOver se debe a que nuestro personaje se ha quedado sin puntos de vida.

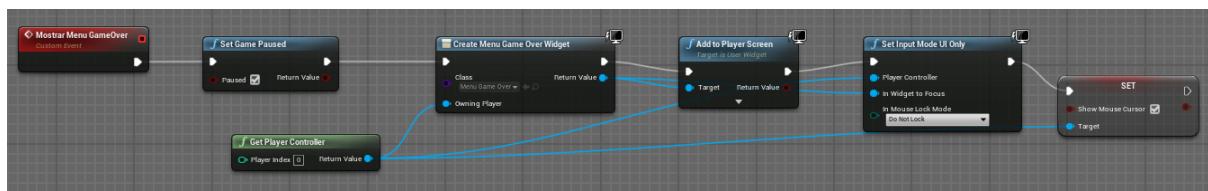
Obsérvese cómo las variables booleanas Out Of Bounds y Expulsión pertenecen al personaje ya que dependen sobre él mientras que la variable Fuera pertenece a un Blueprint Class llamada AreaFuera la cual se encarga de controlar cuando un disparo se ha ido fuera. Más adelante explicaremos cómo funciona.

Para obtener la referencia al Blueprint class AreaFuera se ha utilizado el nodo Get Actor of Class indicando como parámetro nuestro blueprint AreaFuera y luego utilizando el nodo Cast To AreaFuera. En cambio, para obtener la referencia del personaje, se ha incluido el siguiente código el cual se ejecuta antes de mostrar el menú de gameOver y decidir el motivo de la muerte.



En este código, básicamente hemos obtenido una referencia de nuestro personaje, y después de almacenamos esta referencia en una variable que hemos creado al efecto.

Para mostrar el menú de GameOver cada vez que el personaje pierda, vamos a definir un evento customizado llamado Mostrar Menu GameOver el cual seguirá la misma lógica que la representación de otros menús.

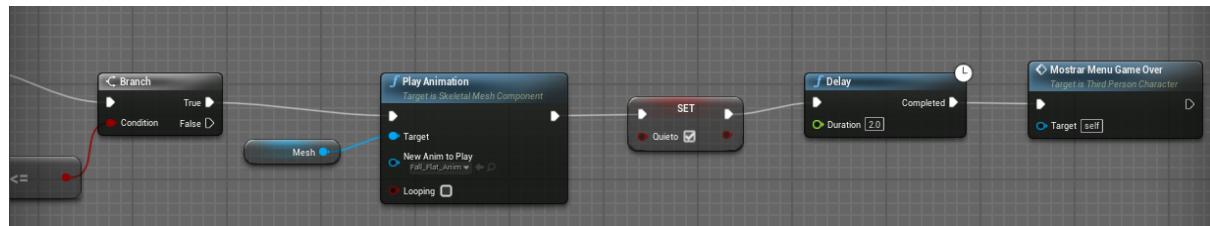


Básicamente, en este código, cuando se llama a este evento customizado Mostrar Menu GameOver se pausará el juego a través del nodo Set Game Paused, y después se creará el Widget Menu GameOver (Create Menu GameOver Widget) y se mostrará por pantalla (Add to Player Screen). Por último, se establecerá el modo de juego a solo interfaz gráfica (Set

Input Mode UI) y se mostrará el cursor del ratón por pantalla (Set Show Cursor Mouse) para que el usuario pueda elegir una de las opciones del menú.

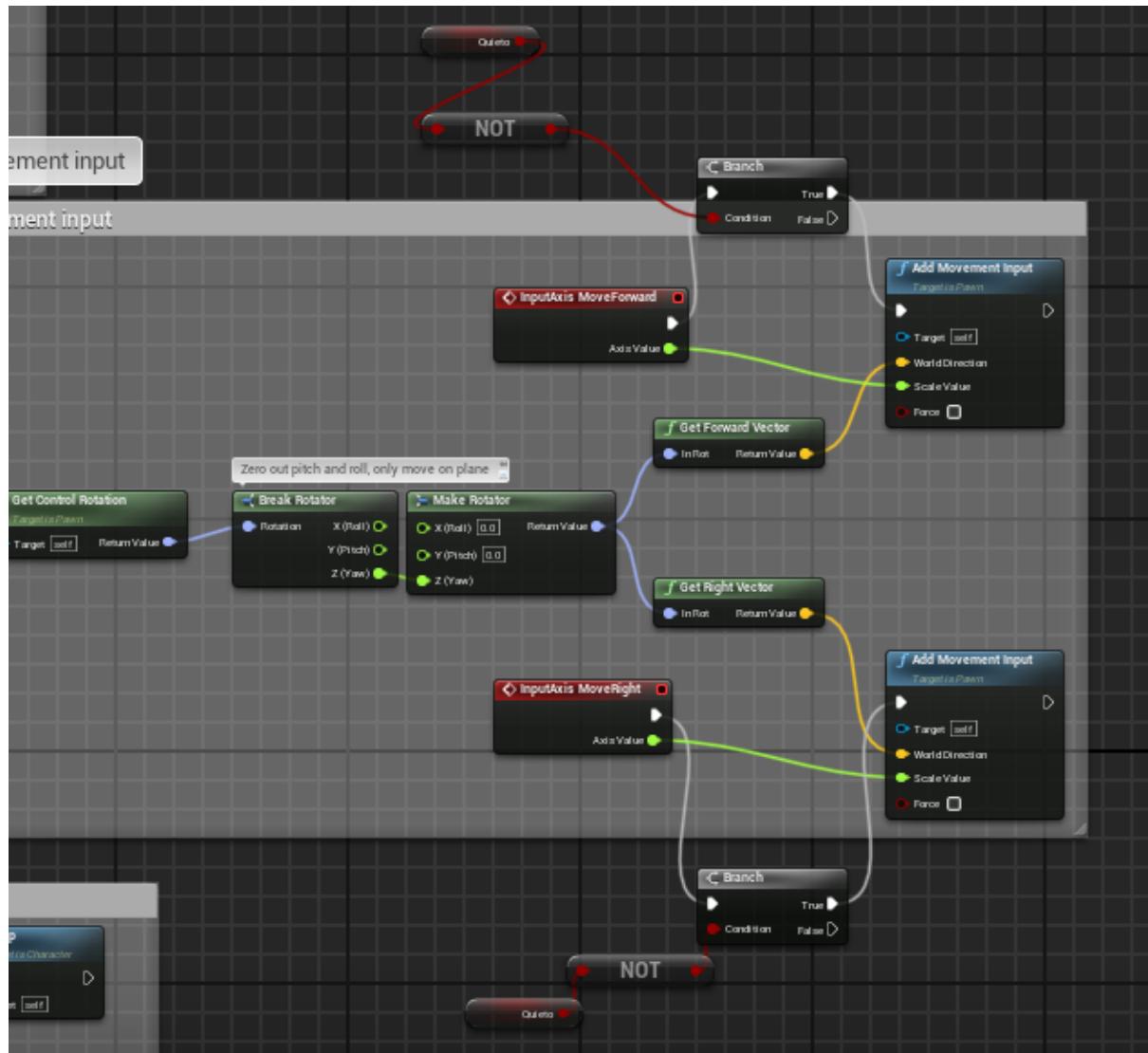
Una vez hemos definido de qué maneras puede perder el personaje, y cómo se representará este menú GameOver en función de cada una de estas razones, debemos de realizar algunos ajustes en el código para activar este evento Mostrar Menu GameOver cuando se produzca una situación que haga perder a nuestro jugador.

Cuando definimos el evento Any Damage en el personaje, diseñamos el código de manera que si el personaje se quedaba sin puntos de vida, el personaje sería destruido. Ahora, vamos a cambiar este código para que el personaje en vez de ser destruido, reproduzca una animación de muerte y después se muestre el menú de GameOver.



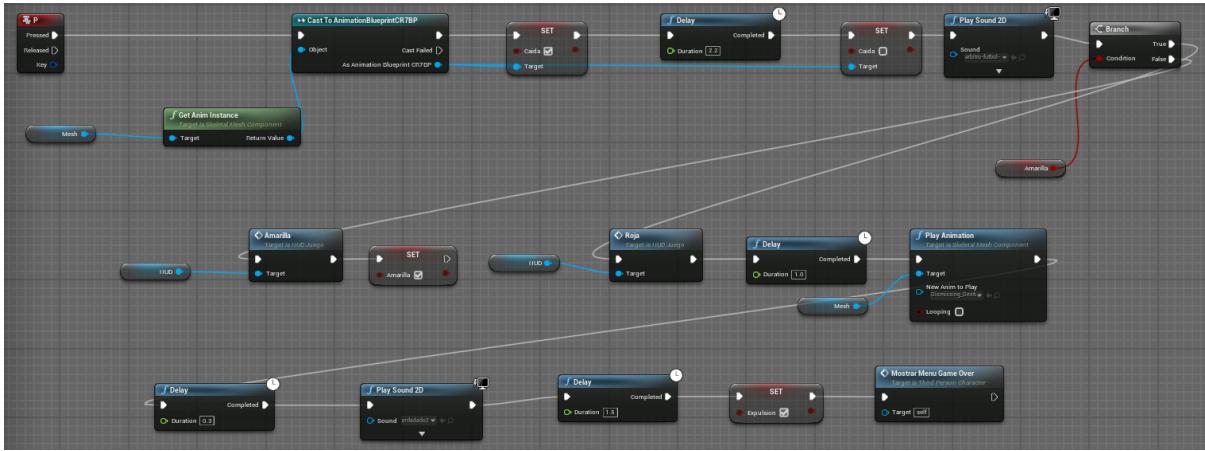
Con esta modificación, ahora cuando el personaje se quede sin puntos de vida se reproducirá una animación de caída (la cual hemos importado anteriormente a nuestro proyecto) y se congelará el movimiento del personaje a través de la activación de una variable booleana llamada **quieto**. Después se esperan 2 segundos y una vez transcurran estos 2 segundos se llamará al evento Mostrar Menu GameOver de forma que se nos mostrará este menú de GameOver.

Para poder congelar el movimiento de nuestro personaje, hemos introducido unas modificaciones en el código de movimiento del personaje el cual estaba creado por defecto.



Estas modificaciones se basan en la creación de una variable booleana llamada *quieto* de forma que si esta variable es verdadera, el personaje no podrá moverse. El objetivo de congelar el movimiento del personaje en algunas situaciones se debe al deseo de reproducir una animación sin que el Jugador pueda mover el personaje.

Cuando definimos el evento P en el personaje, diseñamos el código de manera que si el personaje simulaba un penalti, recibía tarjeta amarilla y si volvía a simular otro penalti recibía tarjeta roja. Ahora, hemos realizado varias modificaciones sobre este evento las cuales se muestran en la siguiente imagen:



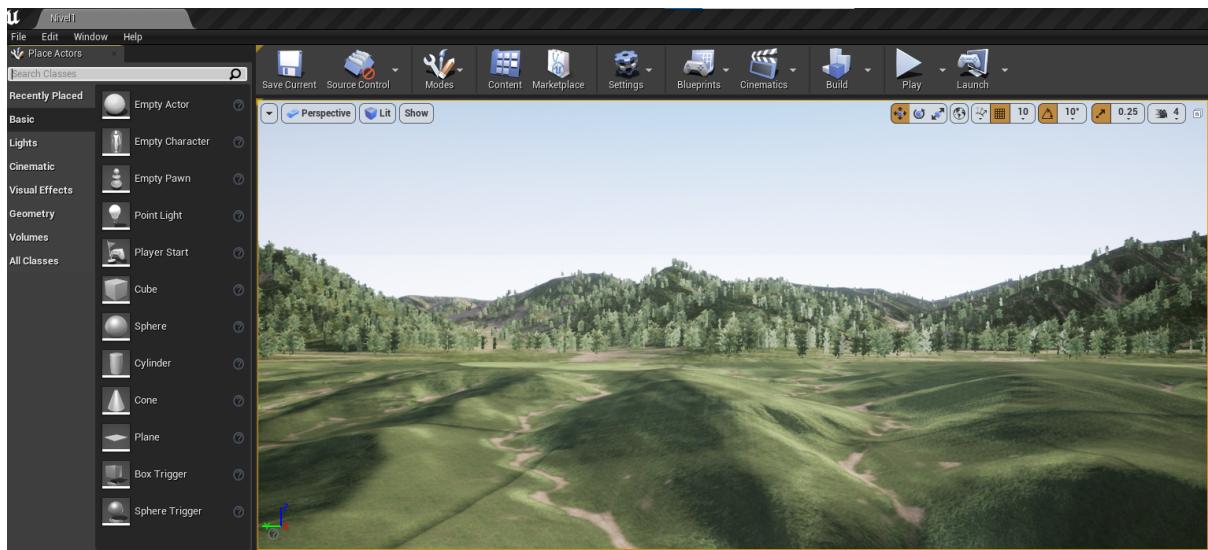
La primera modificación consiste en la inclusión de un ruido de silbato cuando se muestra una tarjeta (da igual de qué color) al jugador. La segunda modificación consiste en la inclusión de un sonido de disconformidad que acompañe a la animación de disconformidad. La última modificación consiste en la llamada al menú GameOver y activación de la variable booleana Expulsión para indicar al menú cuál es la razón por la que ha mostrado este menú de GameOver.

Aún nos quedan por definir 2 situaciones en las que el personaje puede perder y desencadenar en la activación del menú de GameOver. Sin embargo, estas situaciones no se pueden dar todavía ya que no se ha añadido las funcionalidades necesarias para que esto ocurra. Cuando estas funcionalidades sean añadidas se incluirá y se explicará el código que desencadene en la activación del menú de GameOver.

## Creación del escenario de juego

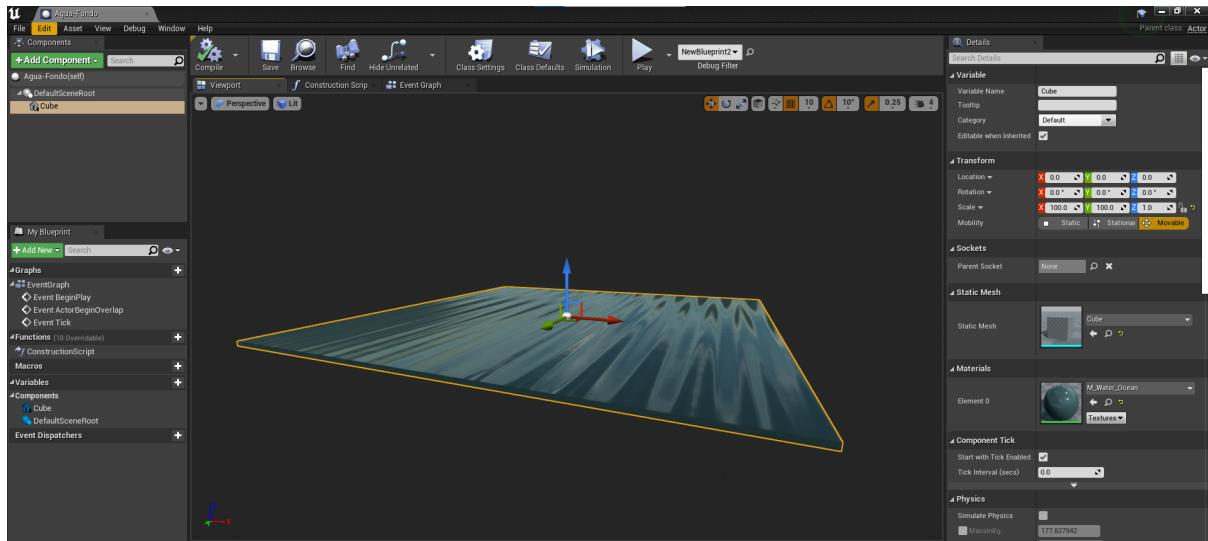
Una vez tenemos gestionado los movimientos y animaciones del jugador, la mecánica de recolección de objetos y la gestión del HUD del jugador, el siguiente paso es definir por así decirlo la infraestructura del nivel. Para crear esta infraestructura o terreno de juego nos vamos a ayudar de un Asset dedicado a la representación de distintos paisajes llamado **Landscape Backgrounds** cuyo link de descarga se adjunta en el siguiente [enlace](#).

Este asset cuenta con 12 escenarios o paisajes ya diseñados por lo que nosotros vamos a utilizar la ventaja que nos da este asset, y vamos a copiar los componentes de uno de estos escenarios (en concreto el escenario Showcase\_mountain\_5) y los vamos a pegar en nuestro nivel. El resultado sería algo parecido a esto:

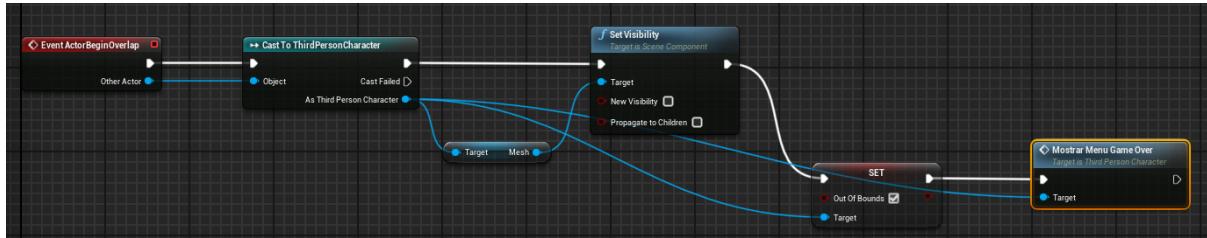


Si observamos el contenido de esta infraestructura, podemos observar que tenemos un paisaje central de color verde y a los bordes distintas montañas con arboles. La idea para crear la infraestructura del nivel es hacer un hoyo en este paisaje central y colocar agua debajo de forma que cada vez que el personaje se caiga de las plataformas, caiga al agua y al tocar el agua el juego finalice.

Para crear los hoyos en el paisaje, es tan sencillo como elegir el modo de editor Landscape y con la brocha pulsar la parte del terreno que se quiere hundir presionando el botón SHIFT cuando se hace click. Para incluir el agua en el fondo del hoyo, vamos a crear una blueprint class llamada AguaFondoBP de tipo Actor la cual va a estar formada por un cubo inmenso al que le asociaremos una textura de agua.



Ahora, vamos dentro de este blueprint, vamos a añadir una porción de código de forma que cuando caiga nuestro personaje y toque el agua, este muera y se muestre un menú de GameOver.



En este código, cuando nuestro personaje atravesese el agua (evento BeginOverlap) se cogerá esta referencia al actor personaje y se hará un casting con el nodo Cast to ThirdPersonCharacter para obtener la referencia de nuestro personaje. Después, se oculta nuestro personaje a través del nodo Set Visibility. Por último, se pone a verdadero una variable del personaje llamada Out Of Bounds la cual será utilizada para determinar en el menú de GameOver la razón por la que el personaje ha perdido. Por último, activamos el evento Mostrar Menu GameOver, de manera que se mostrará el menú de GameOver y la razón por la que hemos perdido.

Una vez hecho esto, lo único que nos quedaría por hacer es desplegar libremente distintas plataformas y objetos por el mapa para así culminar con la representación del nivel. Para crear las plataformas, se hará copia y pega de la plataforma creada por defecto para el nivel y se aplicará una textura de hierba.

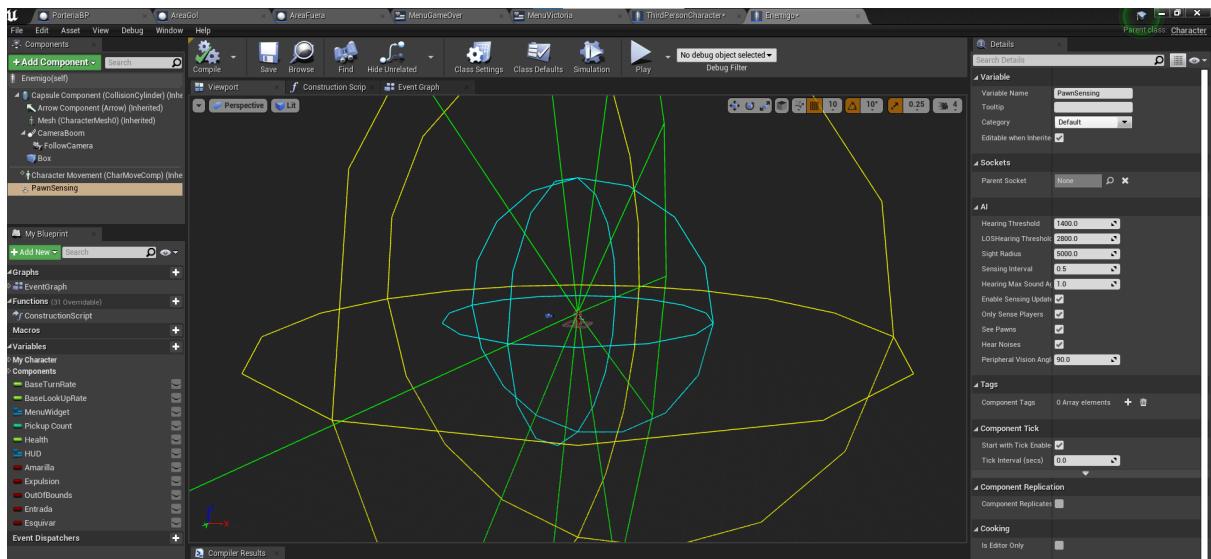


## Creación de enemigos

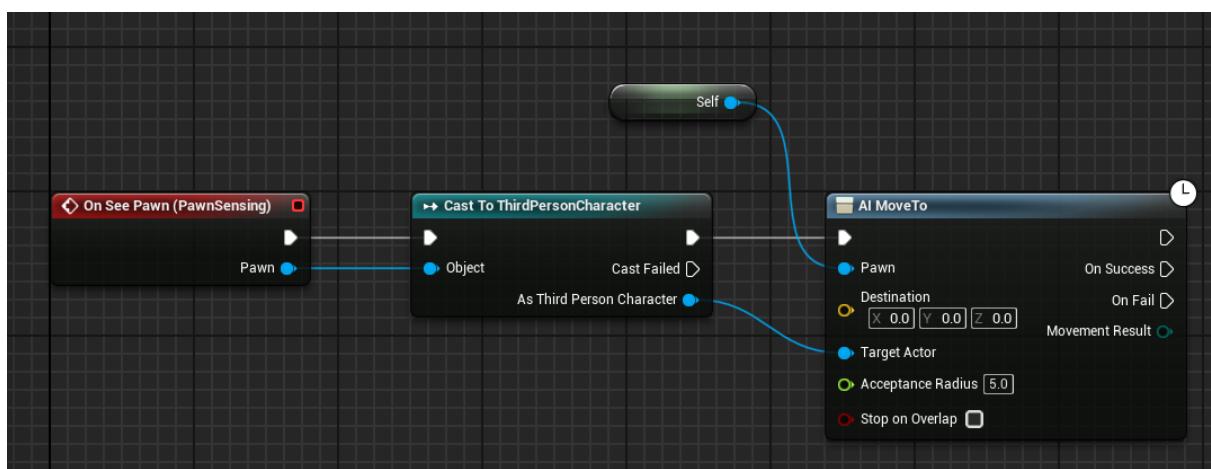
Llegados a este punto, nuestro personaje Crisitano Ronaldo tiene todas las funcionalidades implementadas pero falta crear a los rivales de nuestro personaje. Es por esto por lo que el siguiente paso es diseñar los enemigos de nuestro Jugador. Estos enemigos van a tener como objetivo hacer entradas a nuestro personaje con el objetivo de quitarle vida al mismo y que no llegue al final del nivel. Para ello, vamos a tener que programar una pequeña IA para que los enemigos puedan seguir a nuestro personaje y hacerle entradas en segada.

En primer lugar, para crear el enemigo, vamos a hacer un duplicado de nuestro personaje, pero le vamos a quitar todos los blueprints del personaje (ya que el enemigo no los va a necesitar) y la textura de nuestro personaje. Posteriormente, en el caso de que tengamos tiempo, intentaremos que estos enemigos tengan texturas de futbolistas, pero como este proceso de texturización es tedioso, lo vamos a dejar para más adelante.

Una vez hemos creado el enemigo, vamos a acceder al Blueprint del enemigo y vamos a añadir un componente Pawn Sensing el cual nos proporciona un área en la que nuestro enemigo va a poder detectar nuestra presencia. Este área de presencia puede ser modificada por ejemplo, modificando la visión de nuestro personaje. En nuestro caso, le vamos a dar una visión de 90 grados a nuestro personaje.



Una vez hecho esto, registramos el evento On See Pawn el cual se va activar cuando entremos el en área de visión del enemigo y vamos a añadir el siguiente código:



Básicamente, con este código, obtenemos una referencia a nuestro personaje y luego vamos a usar esta referencia para indicar al enemigo a quien tiene que perseguir.

Una vez hecho esto, nos vamos rápidamente al componente Character Movement y le reducimos la velocidad máxima de 600 a 400 para que corra más lento que mi personaje y así el personaje pueda ir superando a los enemigos con mayor facilidad.



Ahora, vamos a crear un Blueprint de animación para gestionar las animaciones de nuestro enemigo. Este Blueprint de animación será más sencillo que el de nuestro personaje pero contará con una serie de aspectos comunes. Debido a ello, en vez de crear un Blueprint de Animación nuevo vamos a duplicar el que ya tenemos y después haremos los cambios necesarios para adaptarlo al enemigo.

En nuestro caso, solo nos interesa que nuestro enemigo reproduzca la animación de correr cuando nos persigue y reproducir la animación de realizar una entrada en segada cuando el personaje está a una distancia cercana al personaje. De esta manera, nuestra máquina de estados va a contar únicamente con estos dos estados de Movimiento y Entrada.

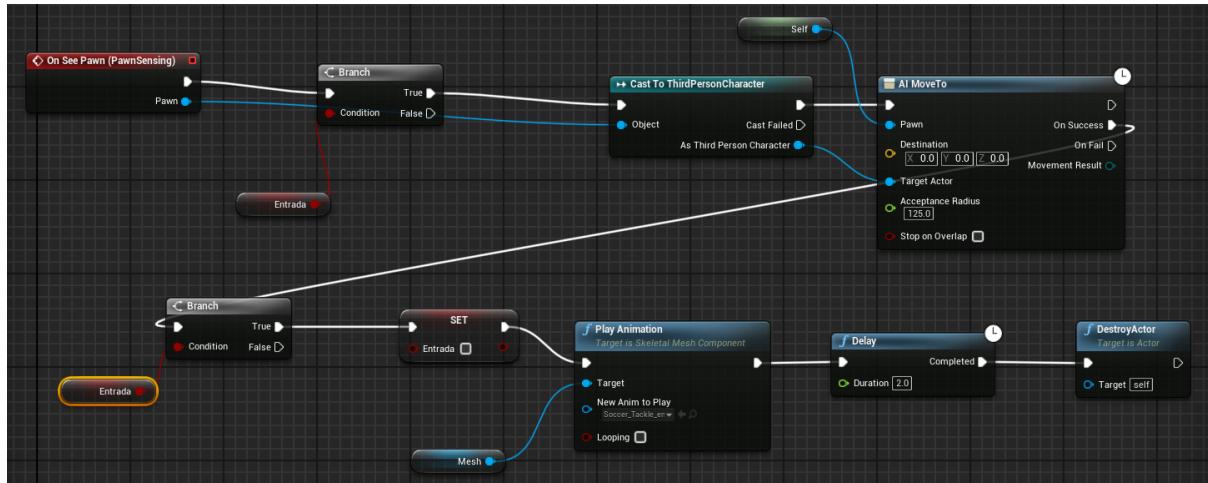


En este caso, podemos ver que no existe ninguna transición entre el estado de Movimiento y el estado de Entrada. Esto se debe a que el personaje una vez realice la animación de segada va a ser destruido. Si tenemos en cuenta esto, es más sencillo en nuestra opinión controlar la transición de movimiento a segada con un nodo Play Animation y después destruir al enemigo.

Para realizar correctamente la animación del movimiento, se utilizará el mismo código que el utilizado en el blueprint de animación de nuestro personaje. Como nosotros hemos creado este blueprint de animación a través de un duplicado del blueprint de animación del personaje, no tenemos que incluir ningún tipo de código en este blueprint de animación.

Una vez hecho esto, no hay que olvidar de asociar el blueprint de animación de nuestro enemigo al enemigo. Si no hacemos esto, este control de las animaciones no se aplicará para nuestro enemigo.

Una vez hemos conseguido que el enemigo nos persiga además de controlar la animaciones del enemigo, vamos a añadir más código al blueprint del enemigo para que el enemigo se tire en segada cuando esté a una distancia prudencial de nuestro personaje. Para ello, vamos a utilizar el siguiente código:



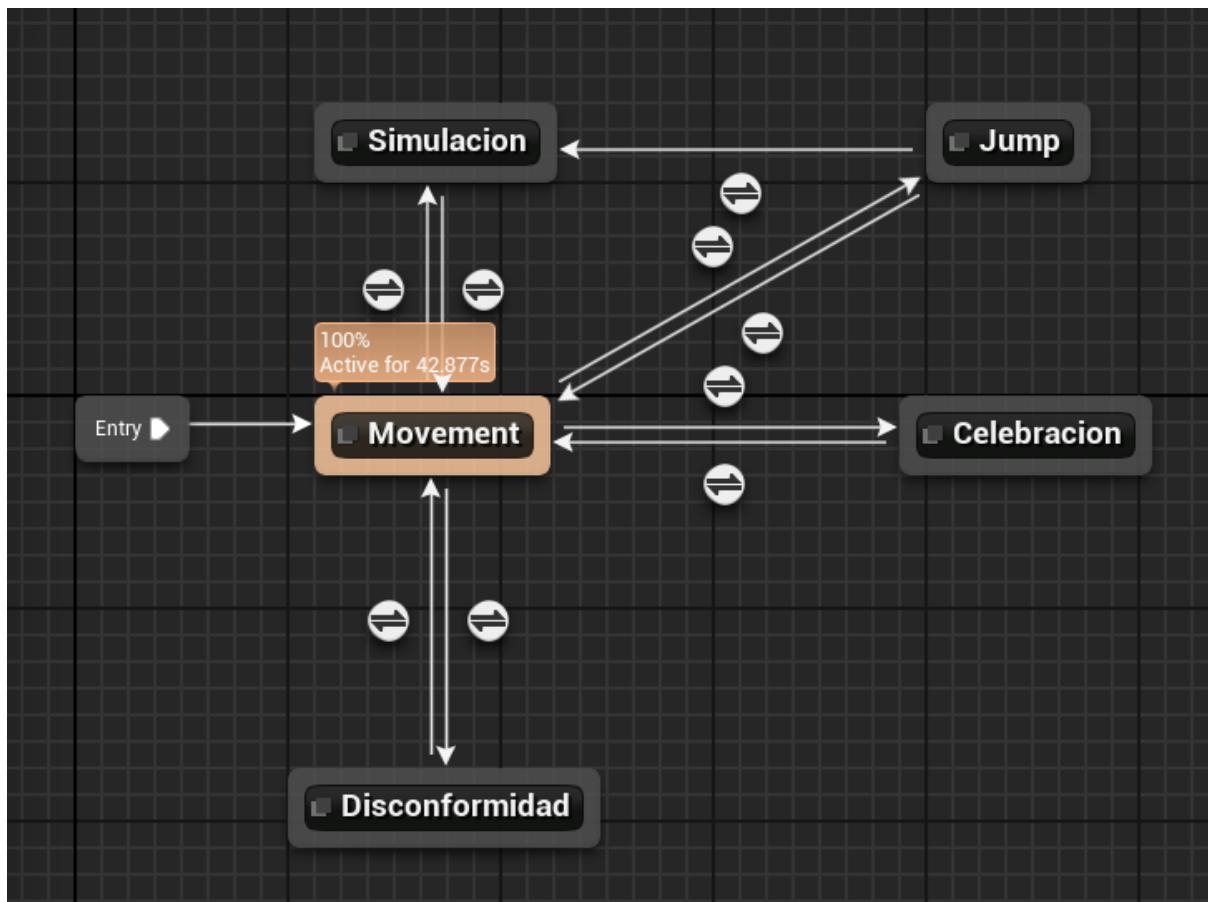
Anteriormente, en el blueprint de enemigo utilizamos un código simple en el que usábamos el nodo AI Move To para que nuestro enemigo persiguiera a nuestro personaje. Profundizando en la utilidad de este nodo, podemos indicar que el enemigo nos persiga hasta que esté a una determinada distancia o rango de nosotros. De esta manera, hemos indicado que el enemigo nos persiga hasta que esté a una distancia menor que 125 de nuestro personaje. Cuando el enemigo entre dentro de este radio, se activará el evento On Success lo cual nos permitirá gestionar que hará nuestro enemigo cuando alcance al jugador.

De esta manera, cuando el enemigo se haya acercado lo suficiente, reproducirá la animación de segada y una vez culmine esta animación el enemigo se destruirá (Destroy Actor). Sin embargo, si no controlamos la ejecución de esta animación, esta se realizará múltiples veces y esto no es lo que queremos. Debido a esto, vamos a crear una variable booleana llamada entrada, la cual le daremos inicialmente un valor verdadero (en el Evento BeginPlay).

Así, cuando nuestro enemigo se disponga a realizar una entrada en segada, en primer lugar se revisa si la variable entrada es verdadera. Si la variable es verdadera, se cambia su valor a falso para que el enemigo no vuelva a realizar la animación de entrada en segada. También se utiliza esta variable al principio del evento On Pawn para que una vez el enemigo haya decidido realizar la entrada, que no pueda seguir más al personaje para que así la ejecución de la segada sea en línea recta. De no controlar esto último, el enemigo nos seguirá durante toda la animación de la segada y esto queda muy extraño en la ejecución del juego.

En último lugar, debemos configurar el jugador para que reciba daño cada vez que reciba un impacto de una entrada y ejecute una animación en la que nuestro personaje se caiga al suelo como consecuencia de la entrada. Para la caída del personaje, utilizaremos la misma animación de caída que la utilizada en el estado de simulación, es más, reutilizaremos el

propio estado de simulación. Simplemente, tenemos que incluir una transición desde el estado de Jump al estado de Simulación porque es posible que la entrada del enemigo impacte al jugador cuando este ha empezado a saltar. De esta manera, nos aseguramos de que se reproduzca la animación de caída cuando el enemigo nos impacte con una de sus entradas.



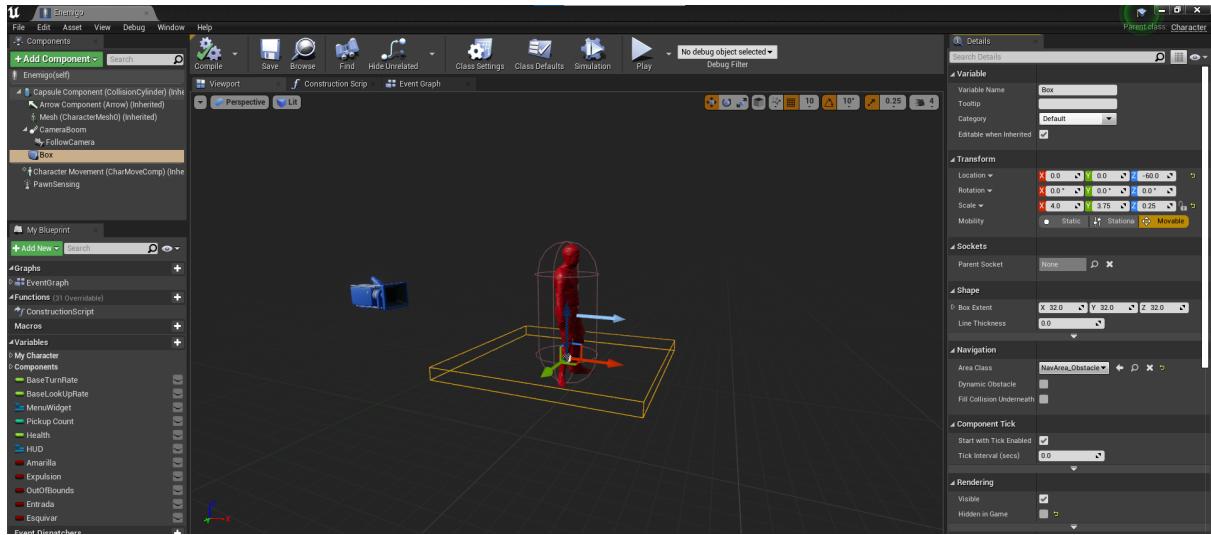
También sería conveniente añadir esta misma transición desde el estado de celebración y de disconformidad pero de momento por simplicidad no vamos añadir estas transiciones.

Ahora, nos dirigimos de nuevo al blueprint del enemigo y vamos añadir un componente box collision al enemigo para detectar cuando el enemigo impacta con el personaje. Este box collision lo vamos a colocar en los pies del enemigo.

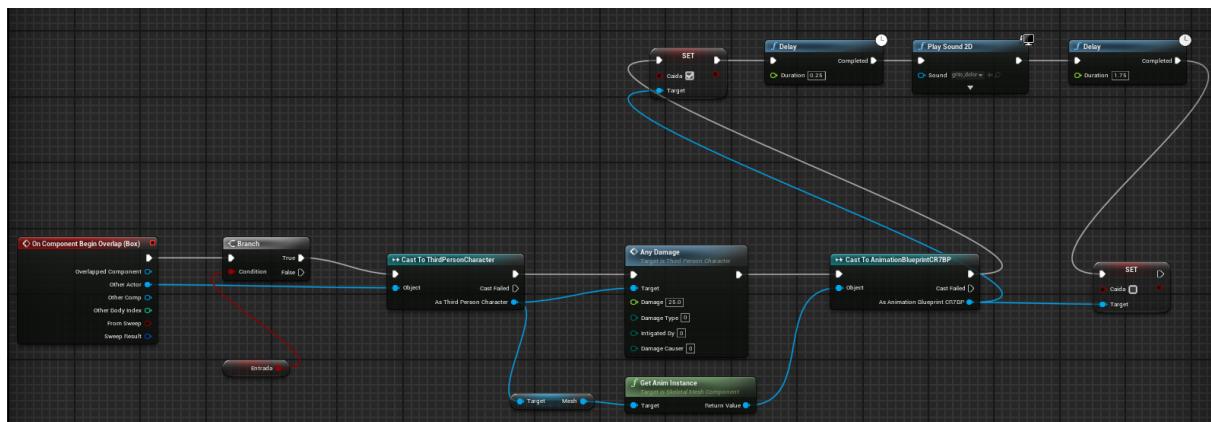
Sin embargo, debido a la construcción de la animación de entrada en segada del enemigo, tenemos un problema ya que durante la reproducción de esta animación, este box collision se mantiene fijo en la posición del enemigo justo antes de hacer la segada.

Debido a esto, el objetivo es que la superficie de esta box collision sea un poco más grande que el radio de acción de la segada. De esta manera, si nuestro personaje se encuentra en el aire justo antes de que el enemigo haga la segada, nuestro personaje no recibirá daño. En cambio, si el personaje estaba en el suelo antes de que el enemigo realice la segada, entonces el personaje recibirá daño del enemigo pues la box collision entró en contacto con

el personaje y se considera que aunque salte el personaje no le dará tiempo a esquivar la entrada.



Ahora partiendo de la idea de cuando las segadas quitan daño y cuando no quitan daño al personaje, vamos a añadir un código que se ejecute solamente cuando la box collision del enemigo entre en contacto con el personaje. Para ello vamos a gestionar el evento BeginOverlap del box collision del enemigo con el siguiente código:



Como se puede observar, lo primero que hacemos en este código es revisar si la variable entrada es verdadera. Con esto conseguimos evaluar solo las colisiones del box collision con el personaje antes de que el personaje haya comenzado a realizar la segada. En segundo lugar, llevamos directamente la referencia del actor con el que se ha producido la colisión con el nodo Cast to ThirdPersonCharacter. Con esto conseguimos tratar sólo las colisiones con nuestro personaje.

Una vez hecho, si se cumplen las condiciones que hemos impuesto, se quita 25 puntos de vida al personaje llamado al evento Any Damage del personaje. Por último, vamos a obtener la referencia del Blueprint de animaciones del personaje (Cast to AnimationBlueprintCR7BP y Get Anim Instance) con el objetivo de activar la transición que

nos lleve al estado de simulación y así reproducir la animación de caída. Para activar la transición simplemente tenemos que poner a verdadero la variable booleana caída. Mientras se reproduce la animación de caída reproduciremos un sonido de un grito de dolor (el cual hemos importado al proyecto) para darle más realismo a la situación. Una vez termina la animación, se pone a falso la variable booleana caída de manera que volvemos al estado de Movimiento.

## Creación del entorno de fútbol

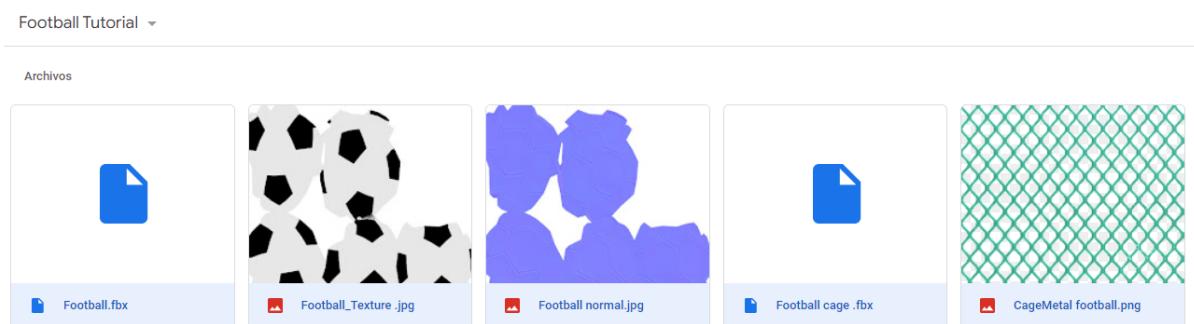
Como hemos dicho en la introducción, este es un juego de fútbol por lo que el último paso para que este juego realmente sea un juego de fútbol es añadir una pelota y un portería de forma que nuestro personaje pueda mover y chutar esta pelota a la portería.

Para crear este entorno de fútbol, nos hemos ayudado de un tutorial de youtube. En este tutorial, se incluye un enlace a una carpeta de Drive con el material necesario para nuestro juego. En enlace tanto al tutorial como a la carpeta de Drive con el material se encuentra en los siguientes enlaces:

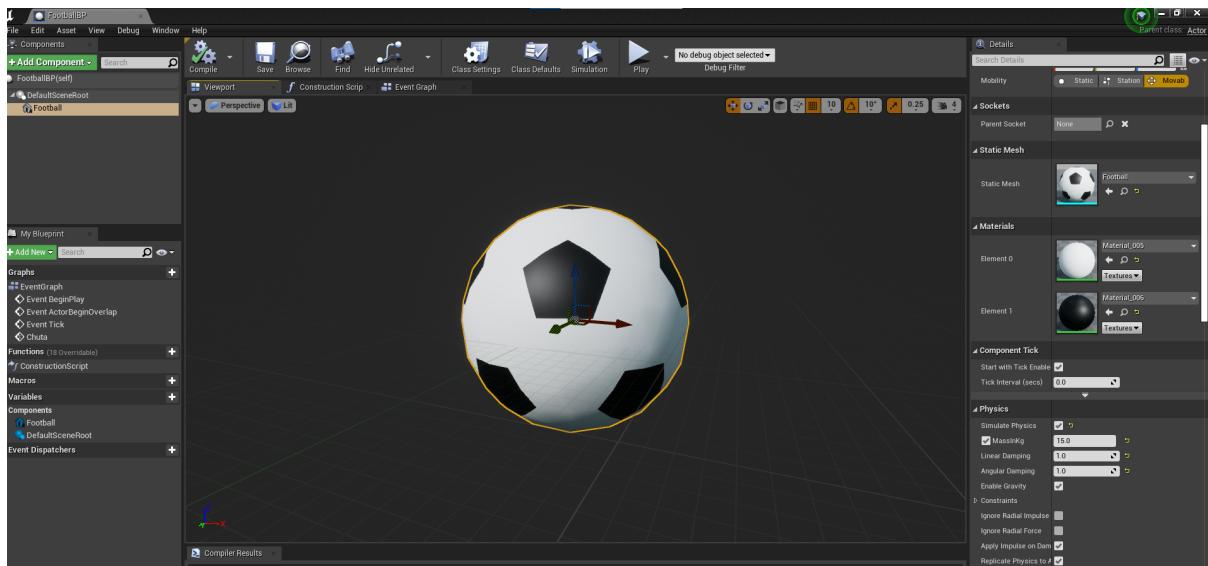
[https://www.youtube.com/watch?v=-viRKf9zDFY&ab\\_channel=Realtime3DNowYoshi](https://www.youtube.com/watch?v=-viRKf9zDFY&ab_channel=Realtime3DNowYoshi)

[https://drive.google.com/drive/folders/0Bz5Qf8WvxssuxRXhaNW5DM1kwSW8?resourcekey=0-4Pt9NQsqDfg\\_3Z2JRYk\\_EA](https://drive.google.com/drive/folders/0Bz5Qf8WvxssuxRXhaNW5DM1kwSW8?resourcekey=0-4Pt9NQsqDfg_3Z2JRYk_EA)

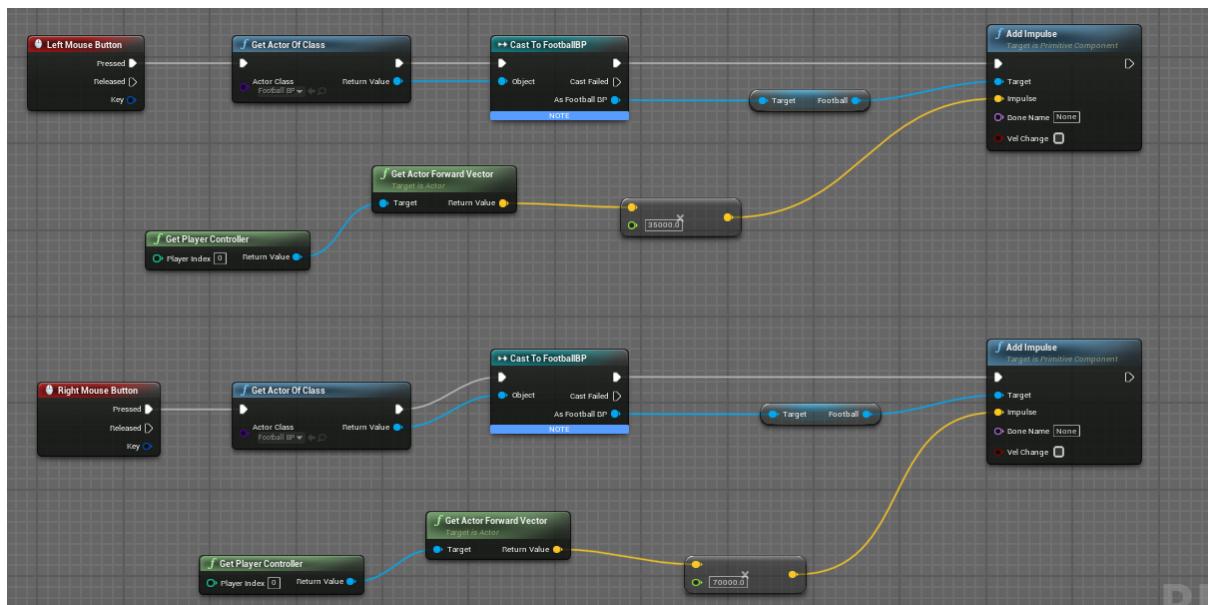
En primer lugar, vamos a descargar el contenido de la carpeta de Drive y vamos a importarla a nuestro proyecto. Esta carpeta cuenta con un modelo 3D de balón, un modelo 3D de una portería y texturas tanto para el balón como para la portería.



Una vez hecho esto, vamos a crear un Blueprint Class de tipo actor llamado FootballBP al cual vamos a arrastrar el modelo 3D de la pelota. Después, nos vamos al panel derecho y buscamos el componente de físicas. Dentro de este componente vamos a habilitar el uso de físicas y vamos a darle una masa de 15 kg a nuestra pelota. También cambiaremos los parámetros Linear Dumping y Angular Dumping a 1. Tampoco olvidar de aplicar las texturas adecuadas a la pelota en el caso de que no estén bien aplicadas por defecto.



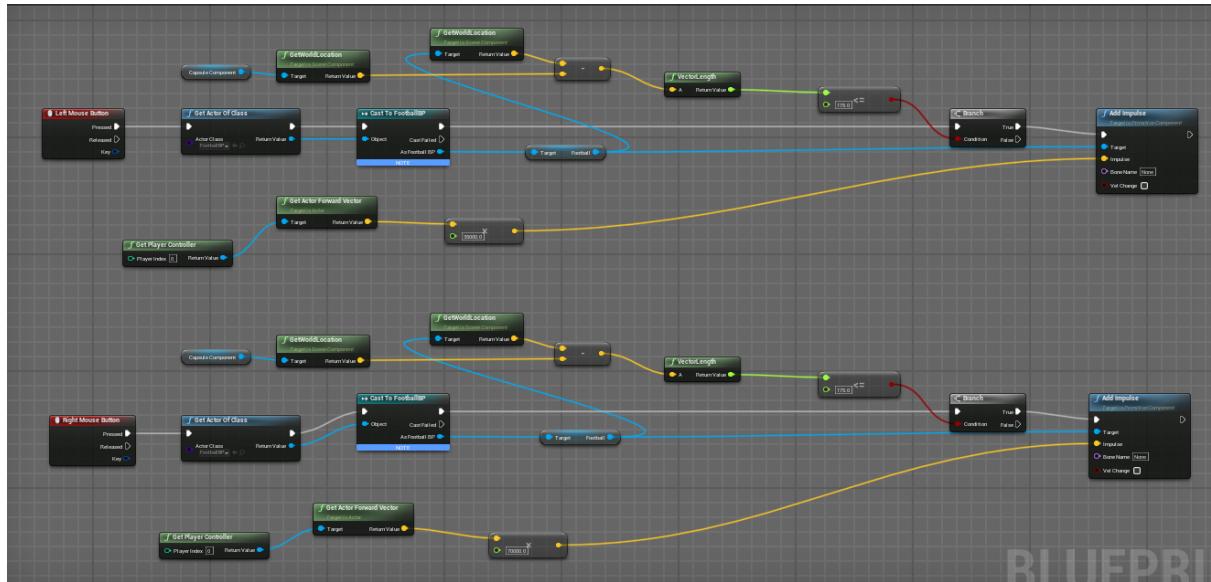
Ahora, vamos a definir en el blueprint del personaje cómo hacer uso de esta pelota. Básicamente queremos que nuestro personaje pueda chutar la pelota y que pueda avanzar con la pelota. De esta manera, vamos a definir 2 eventos que se ejecuten cuando se hagan click a los botones de ratón de forma que el botón derecho del ratón se utilizará para chutar y el botón izquierdo del ratón se utilizará para avanzar. Cuando se produzcan estos eventos, se ejecutará el siguiente código:



Para avanzar o chutar la pelota, en primer lugar, se obtiene la referencia a la clase FootballBP a través de los nodos Get Actor Of Class y Cast To FootballBP. Después se utilizará la referencia de la pelota para añadir un impulso a la pelota con el nodo Add Impulse. Este impulso estará determinado por el movimiento de nuestro personaje (obtenido a través de Get Player Forward Vector) y multiplicado por una constante.

Como se puede observar, la diferencia entre el código de chutar y avanzar difiere en un único aspecto que es la potencia con que se impulsa la pelota y el valor de esta constante.

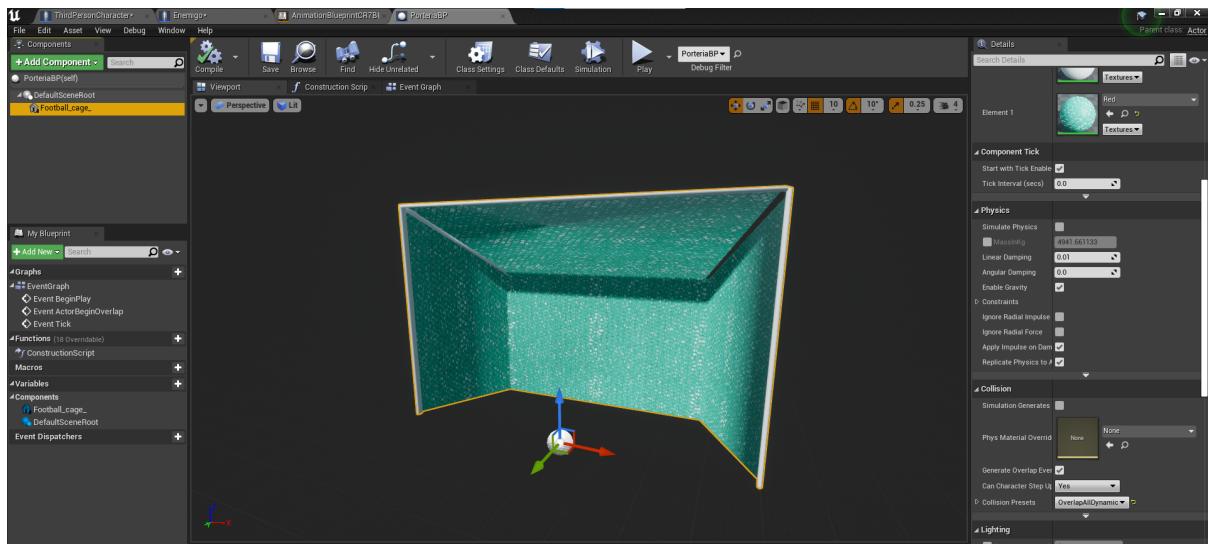
Sin embargo, con este código se puede chutar o avanzar la pelota independientemente de donde se encuentre el jugador, lo cual es muy poco estético. Debido a esto, vamos a modificar el código anterior y vamos a tener en cuenta la posición del jugador y de la pelota antes de realizar el chut o avance



Básicamente, esta modificación consiste en obtener cada vez que se quiera chutar o avanzar la pelota, la posición de la pelota y del jugador. Para obtener la posición de la pelota y el jugador vamos a utilizar el nodo GetWorldLocation donde se le va a pasar respectivamente la referencia del balón y del jugador. Después restamos el valor de ambas posiciones y calculamos el módulo del vector resultante. Si el módulo del vector diferencia es menor que un valor de referencia se permite realizar las acciones de chutar o avanzar pero si el módulo del vector diferencia es mayor que el valor de referencia la acción queda invalidada y no se hace nada.

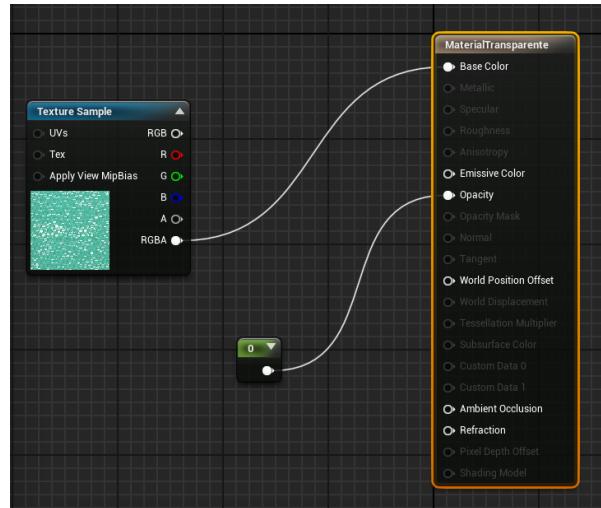
Una vez hemos diseñado perfectamente las físicas de la pelota y su interacción con el personaje, lo último que nos queda es añadir la portería al nuestro nivel. El objetivo de esta portería evidentemente es determinar si el disparo de nuestro jugador ha entrado o no. El objetivo de nuestro jugador en el nivel es superar a los rivales y meter el balón en la portería. Es por eso por lo que debemos configurar bien esta ya que la diferencia entre un disparo fallido y uno anotado define el éxito o fracaso de nuestro personaje.

Para añadir la portería, en primer lugar, vamos a crear un blueprint class de tipo actor llamado PorteriaBP a la cual vamos a añadir el modelo 3D de la portería que hemos descargado. Después añadiremos las textura de red a la portería, ya que no viene aplicada esta textura por defecto. Por ultimo definiremos las colisiones con la portería en el modo OverlapAllDynamic para que el balón pueda atravesar la portería.

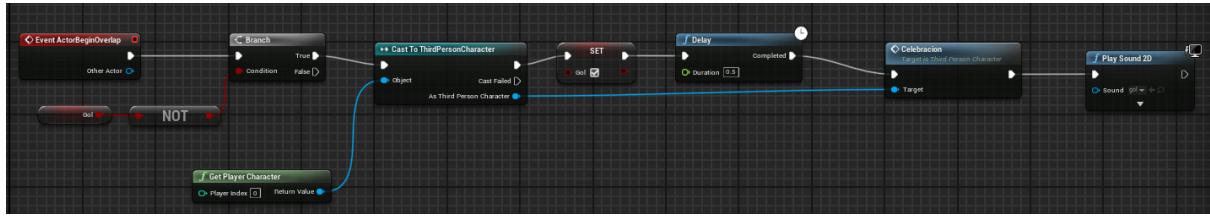


Ahora, para detectar cuando un balón ha entrado en la portería vamos a utilizar un plano transparente que se va a colocar entre los palos de la portería. De esta manera, cuando el balón atraviese este plano, el gol será detectado y a través del evento BeginOverlap podemos gestionar las acciones a realizar tras marcar el gol.

De esta manera, vamos a crear un Blueprint Class de tipo actor llamada AreaGolBP a la cual vamos a añadir un plano con una textura transparente. Para aplicar esta textura transparente al plano se ha creado un material donde el valor de la opacidad es 0.



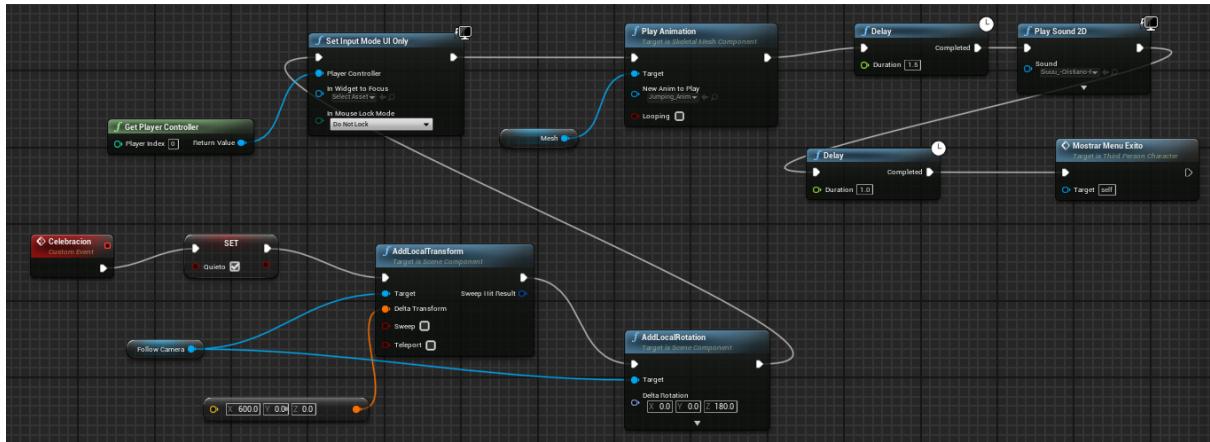
Una vez hemos colocado el plano de forma que abarque toda la superficie de la portería y este situada entre los palos, vamos a cambiar la propiedad de colisión del plano a OverlapAllDynamic y vamos a definir un evento Begin Overlap para gestionar las acciones a realizar cuando el personaje marque un gol. El código del blueprint de AreaGolBP es el siguiente:



En este código, cuando el balón entre en la portería, se obtiene una referencia al personaje (Cast to ThirdPersonCharacter y Get Player Character) y a través de esta referencia se activa el evento Celebración el cual está definido en el personaje y veremos más adelante. Por último, se reproducirá un sonido de fondo de un estadio celebrando un gol para darle más dramatismo al juego.

Obsérvese que se ha creado una variable booleana llamada Gol que gestiona si nuestro personaje ha marcado gol o no. Esta variable se utiliza para que el gol solo se registre una vez ya que la pelota es posible que atraviese varias veces el área de gol debido a un rebote.

Una vez nuestro personaje ha metido un gol, queremos que lo celebre como es debido. Por eso hemos creado un evento customizado en el personaje el cual se encargará de celebrar el gol como se merece. El código de celebración será el siguiente:



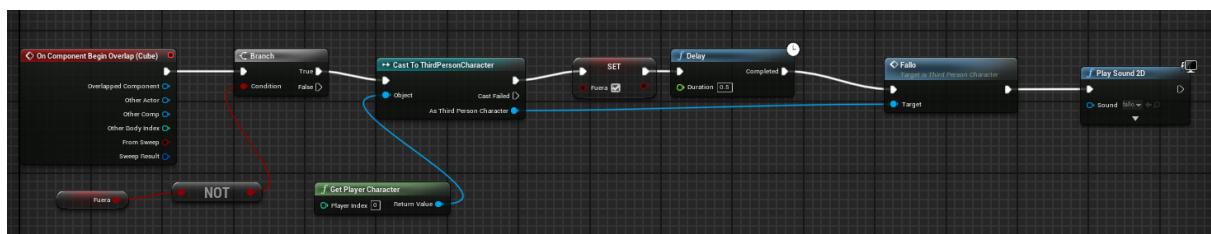
Como se puede observar, en este código lo primero que hacemos es establecer la variable booleana *quieto* a true. Esto es porque queremos que el jugador no pueda mover el personaje mientras está celebrando el gol. Después, a través de los nodos *AddLocalTransform* y *AddLocalRotation* se ajusta la posición y rotación de la cámara del personaje para que lo enfoque desde la parte delantera y no desde la parte trasera.

Más adelante, se establece el input del juego a solo interfaz gráfica para que el jugador no pueda mover la cámara del personaje. Por último, se reproduce la animación de celebración junto con el sonido de celebración característico de Crisitano Ronaldo. Al final se observa que llamamos a un evento llamado Mostrar Menu Éxito. Este evento es responsable de mostrar un menú de “victoria” el cual nos permite avanzar al siguiente nivel. Más adelante hablaremos sobre este menú de “victoria”.

Ahora, para detectar cuando nuestro personaje ha fallado en su disparo a portería vamos a utilizar un plano transparente que se va a colocar detrás de la portería. De esta manera, cuando el balón atraviese este plano, el fallo será detectado y a través del evento BeginOverlap podremos gestionar las acciones a realizar tras fallar el gol.

De esta manera, vamos a crear un Blueprint Class de tipo actor llamada AreaFuerabp a la cual vamos a añadir un plano con una textura transparente al igual que hicimos con AreaGolBP.

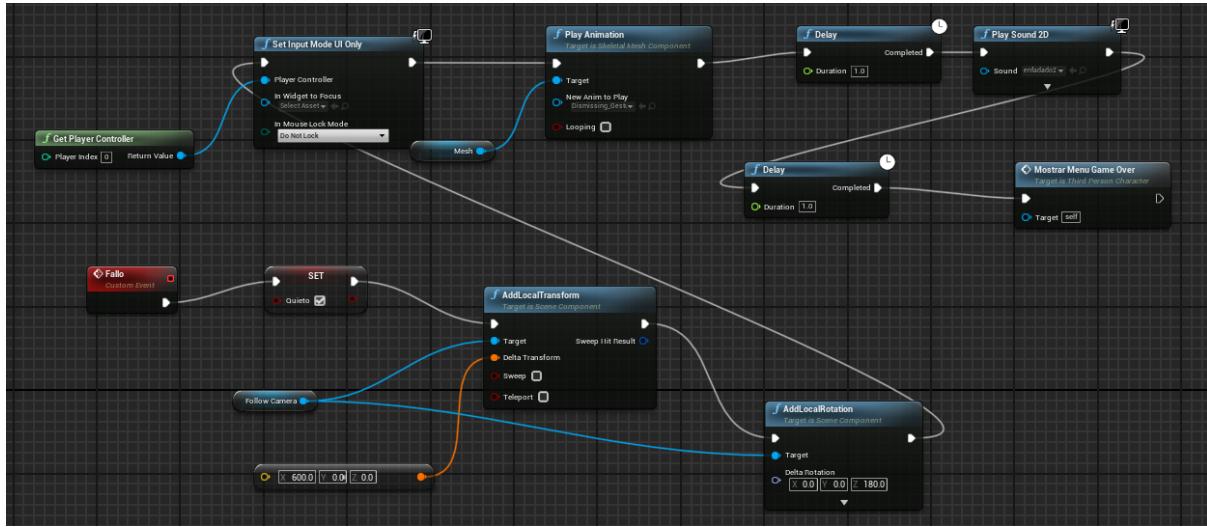
Una vez hemos colocado el plano detrás de la portería, vamos a cambiar la propiedad de colisión del plano a OverlapAllDynamic y vamos a definir un evento Begin Overlap para gestionar las acciones a realizar cuando el personaje falle un gol. El código del blueprint de AreaFuerabp es el siguiente:



En este código, cuando el balón atraviese este plano, se obtiene una referencia al personaje (Cast to ThirdPersonCharacter y Get Player Character) y a través de esta referencia se activa el evento Fallo el cual está definido en el personaje y veremos más adelante. Por último, se reproducirá un sonido de fondo de un estadio lamentando el fallo para darle más dramatismo al juego.

Obsérvese que se ha creado una variable booleana llamada Fallo que gestiona si nuestro personaje ha fallado o no. Esta variable se utiliza para que el fallo solo se registre una vez ya que la pelota es posible que atraviese varias veces el área de fallo debido a un rebote.

Una vez nuestro personaje ha fallado el gol, queremos que se lamente. Por eso hemos creado un evento customizado en el personaje el cual se encargará de lamentarse del fallo. El código de lamento será el siguiente:

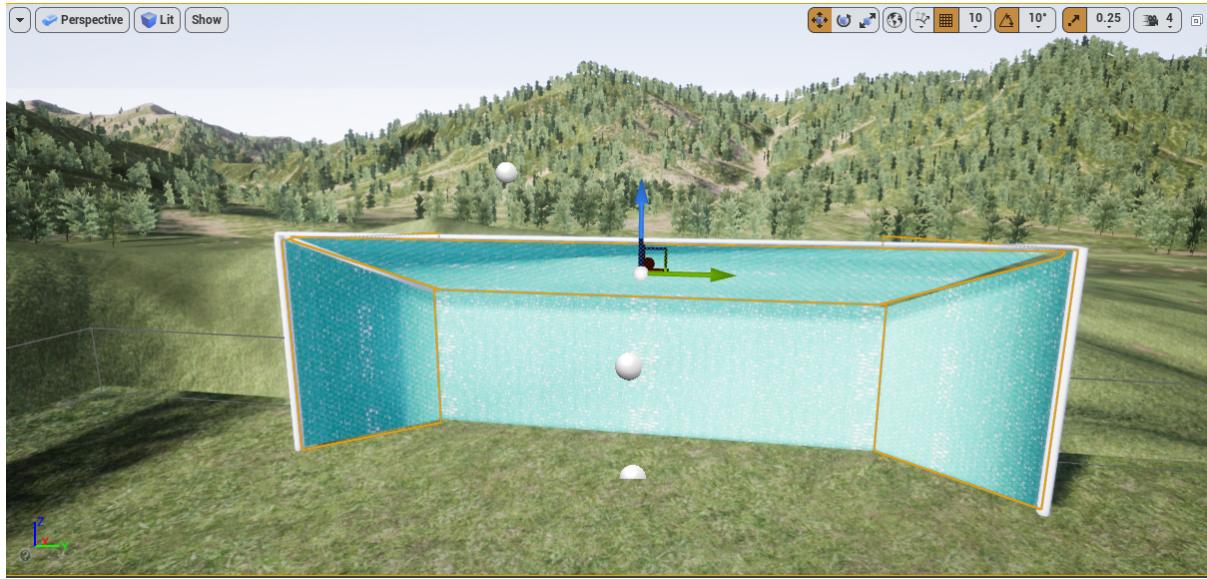


Como se puede observar, en este código lo primero que hacemos es establecer la variable booleana quieto a true. Esto es porque queremos que el jugador no pueda mover el personaje mientras está lamentándose del fallo. Despues, a través de los nodos AddLocalTransform y AndLocalRotation se ajusta la posición y rotación de la cámara del personaje para que lo enfoque desde la parte delantera y no desde la parte trasera.

Más adelante, se establece el input del juego a solo interfaz gráfica para que el jugador no pueda mover la cámara del personaje. Por último, se reproduce la animación de desaprobación junto con el sonido de desaprobación. Al final llamamos al evento Mostrar Menu GameOver para que se muestre el menú de GameOver ya que nuestro personaje ha perdido.

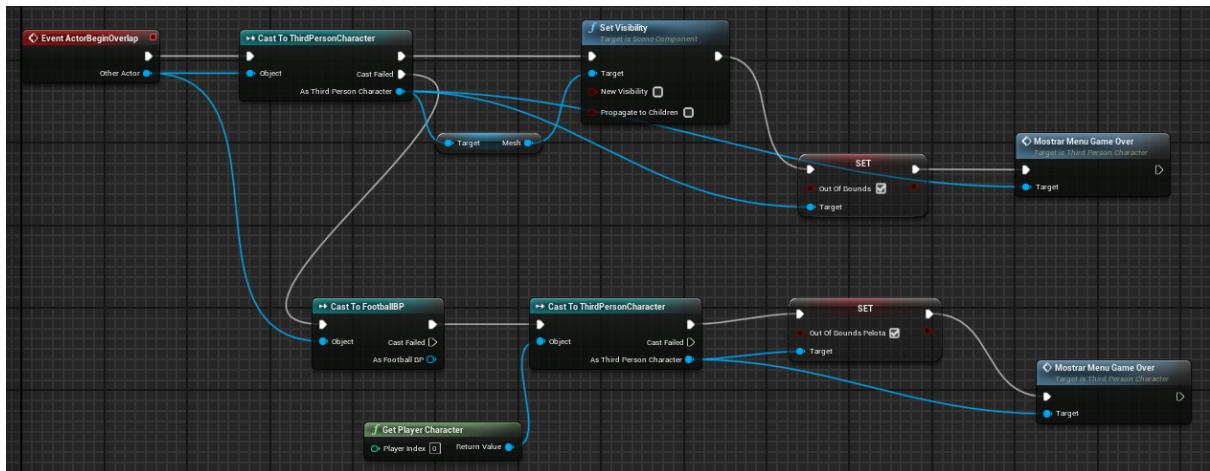
Con esto ya tenemos toda la configuración con respecto a la mecánica de fútbol implementada en el juego. Sin embargo, hay una cosa que nos ha quedado en el aire. Al crear el blueprint de la portería, definimos las colisiones de la portería como OverlapAllDynamic. De esta manera, la pelota atravesará la portería y posteriormente atravesará el área de fallo y esto no es lo que queremos ya que se reproducirá tanto la animación de celebración como la de desaprobación.

Para evitar este problema, hemos creado una serie de planos transparentes que están colocados en la red de la portería. Estos planos si detectan las colisiones por lo que mantendrán el balón dentro de la portería cuando nuestro personaje meta un gol.



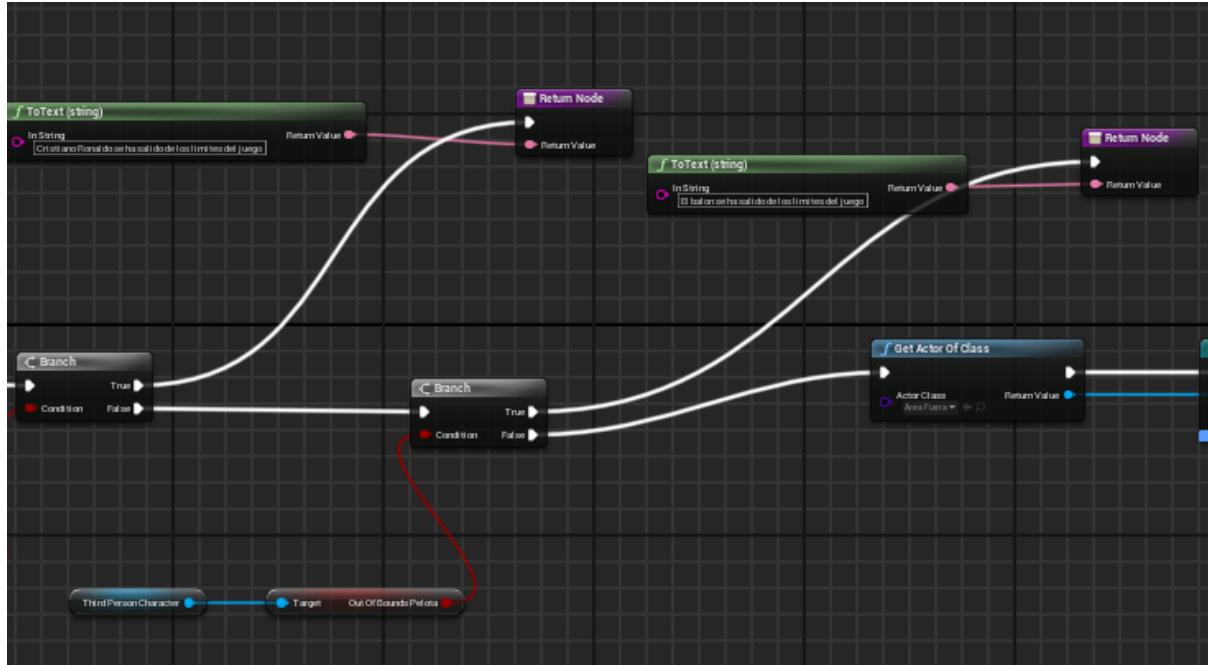
En condiciones normales, habíamos definido la colisión de la portería en el modo BlockAllDynamic para que detecte las colisiones con la pelota. Sin embargo, el modelo está definido de forma que la Static Mesh de la red está a la misma altura o nivel de los palos y de haber optado por esta solución no habríamos podido dotar a la portería de un carácter tridimensional.

Por último, vamos a realizar una cambios en el blueprint del agua del fondo de forma que cuando caiga el balón al agua, el personaje pierda y se muestre el menú de GameOver con la razón de la derrota.



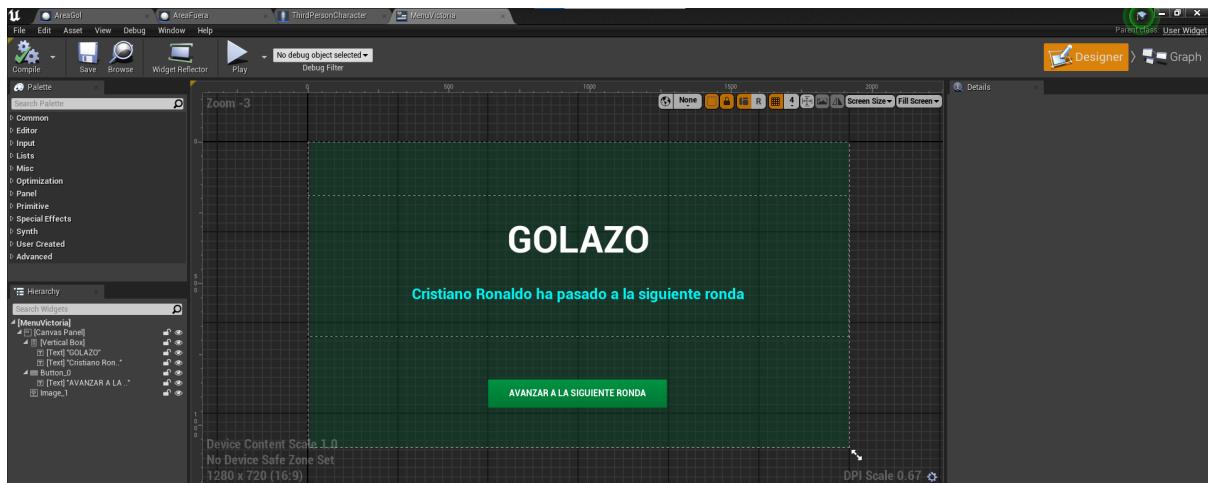
Con estos cambios, cuando un objeto atraviesa el agua de fondo y este objeto no se trata de nuestro personaje pues el Cast to ThirdPersonCharacter falla, se comprueba a través del nodo Cast To FootballBP si el objeto que ha atravesado es un balón. Si esto es así, el casting es correcto por lo que se pone a true la variable booleana Out of Bounds Pelota del personaje la cual será utilizada por el menú de GameOver para determinar la razón de la derrota. Por último, se llamará al evento de mostrar el menú de GameOver.

Por otro lado en el menú de GameOver se ha modificado el código de la función de binding para evaluar si la pelota se ha salido de los límites, ya que ahora es un motivo por el cual nuestro personaje puede perder en el juego. La mecánica es similar a la utilizada para determinar otras razones de pérdida en el juego.



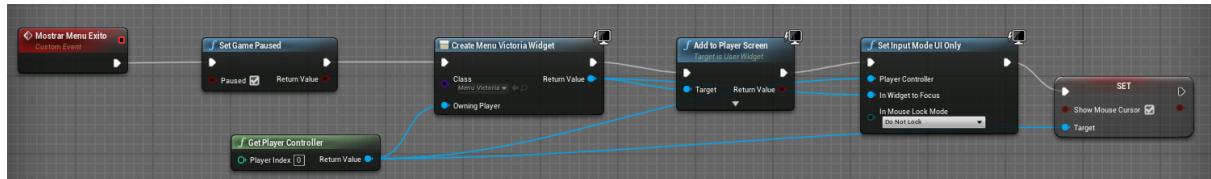
## Creación del menú de Éxito y pasó al siguiente nivel

Antes de pasar al desarrollo del siguiente nivel hay que comentar un par de puntos. El primero de ellos es la creación de un menú de éxito el cual se muestra cuando el personaje mete un gol en la portería. Para crear este menú de éxito, creará un nuevo blueprint widget llamado MenuVictoria el cual tendrá el siguiente aspecto:



Como se puede observar, este menú cuenta con un botón cuya funcionalidad se va a programar a continuación. También contará con un texto informativo. Este menú de Exito cuenta con una imagen con un fondo verde para darle más “ambientación” al menú de Éxito.

Cuando definimos el evento de celebración del gol en el personaje, al final de la gestión de este evento hicimos una “llamada” al evento de Mostrar Menu Éxito. El código asociado a este evento de Mostrar Menu Éxito es el siguiente:



Básicamente, en este código, cuando se llama a este evento customizado Mostrar Menu Éxito se pausará el juego a través del nodo Set Game Paused, y después se creará el Widget Menu Victoria (Create Menu Victoria Widget) y se mostrará por pantalla (Add to Player Screen). Por último, se establecerá el modo de juego a solo interfaz gráfica (Set Input Mode UI) y se mostrará el cursor del ratón por pantalla (Set Show Cursor Mouse) para que el usuario pueda elegir una de las opciones del menú.

En cuanto a la programación de la funcionalidad del botón, al igual que hemos hecho en menús anteriores, registramos el evento On Clicked de forma que cuando ocurra ese evento se realice el cambio de nivel.

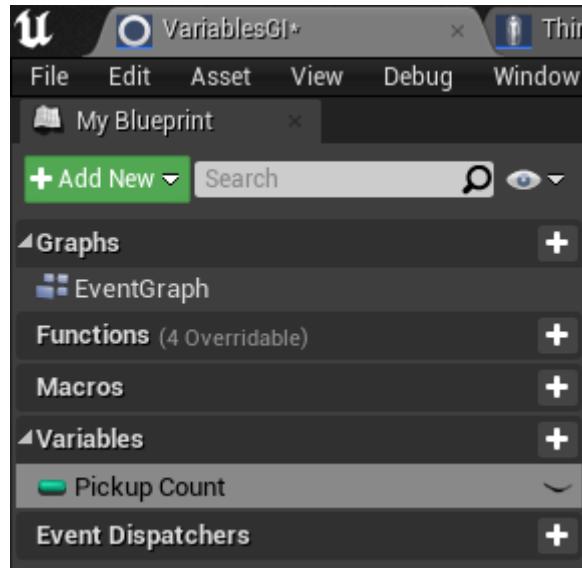


Como se puede observar, realizar el cambio de nivel es tan simple como llamar al nodo Open Level e indicar el nombre del nivel.

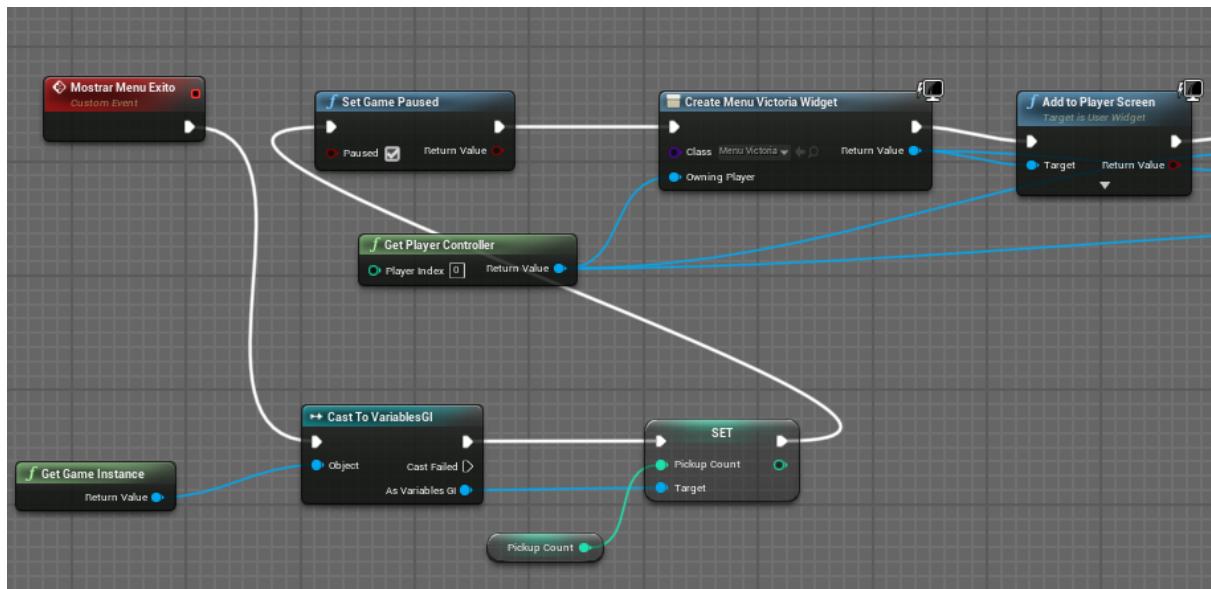
Tal como tenemos el juego, cuando nuestro personaje marque un gol, le aparecerá este menú y cuando pulse el botón de avanzar a la siguiente ronda, se cargará el segundo nivel del juego. Sin embargo, al cambiar de nivel perderá la información de los balones de oro recogidos ya que la variable que lleva la cuenta de los balones de oro recogidos se encuentra en el personaje y este se destruye al cambiar de nivel.

Para evitar perder esta información, vamos a hacer uso de un blueprint llamado GameInstance. Un GameInstance es un objeto que permanece por así decirlo creado

durante toda la ejecución del juego. De esta manera, podemos utilizar este GameInstance para almacenar distintas variables que queremos conservar durante todo el transcurso del juego. En nuestro caso, vamos a crear un nuevo Blueprint Class de tipo GameInstance llamado VariablesGI y en este GameInstance vamos a crear una variable entera llamada Pickup Count para almacenar el número de balones de oro recogidos por el personaje en todos los niveles.



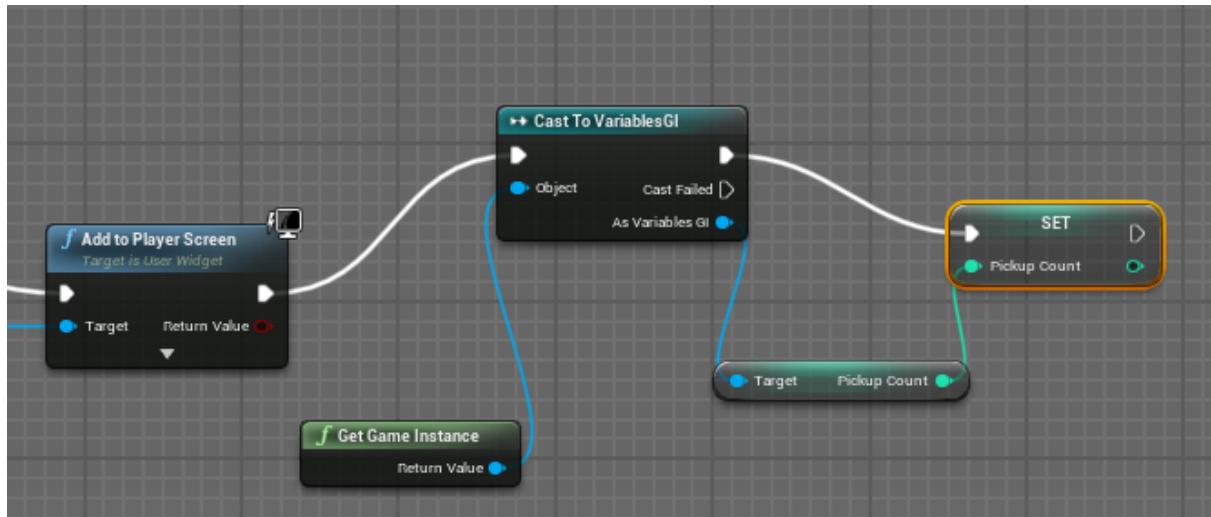
Ahora vamos a realizar unos cambios en el blueprint del personaje, para llevar el número de balones de oro recogidos por el jugador a nuestro GameInstance antes de realizar el cambio de nivel. El primer cambio se va producir en el código en el que se muestra el menú de Éxito, ya que antes de mostrar el menú de Éxito, vamos a trasladar el número de balones de oro recogidos al GameInstance.



Básicamente, con estas modificaciones, se obtiene una referencia al GameInstance VariablesGI a través de los nodos Cast to VariablesGI y Get Game Instance. Una vez se ha

obtenido la referencia del GameInstance, se actualiza el valor de su variable Pickup Count con el número de balones de oro recogidos por el jugador hasta el momento.

Por otro lado, vamos a añadir otra porción de código en el evento BeginPlay para que en el momento en el que se cargue el nivel 2, podamos recuperar el número de balones de oro recogidos en niveles anteriores.



Básicamente, con estas modificaciones, se obtiene una referencia al GameInstance VariablesGI a través de los nodos Cast to VariablesGI y Get Game Instance. Una vez se ha obtenido la referencia del GameInstance, se actualiza el valor de la variable Pickup Count del personaje con el número de balones de oro recogidos almacenados en la variable del GameInstance.

De esta manera, si el nivel que se está ejecutando es el primero, la variable Pickup Count del GameInstance vale 0, por lo que la variable Pickup Count se le asignará un 0. Por otro lado, si el nivel que se está ejecutando es el segundo, la variable Pickup Count del GameInstance vale x, por lo que la variable Pickup Count se le asignará un x siendo x el número de balones de oro recogidos en el nivel 1.

Con estas modificaciones, actualizaremos el GameInstance justo antes de cambiar de nivel y recuperaremos su información al empezar el siguiente nivel. Con esto también conseguimos que la cuenta de balones de oro dentro de un nivel siga a cargo del jugador y que el GameInstance sólo haga de puente entre niveles.

## Desarrollo del segundo nivel

Una vez hemos creado la infraestructura del primer nivel y la mecánica de juego necesaria para este nivel, ahora vamos a crear la infraestructura del segundo nivel. De momento, para construir el segundo nivel vamos a crear un duplicado de la infraestructura del primer nivel. De esta manera, a partir de lo realizado en el primer nivel, podemos ir añadiendo más infraestructura y nuevas funcionalidades.

El objetivo de este segundo nivel es mantener la esencia del primer nivel pero añadirle un mayor grado de dificultad mediante la incorporación de nuevas mecánicas y funcionalidades, además de añadir un escenario de juego más complejo.

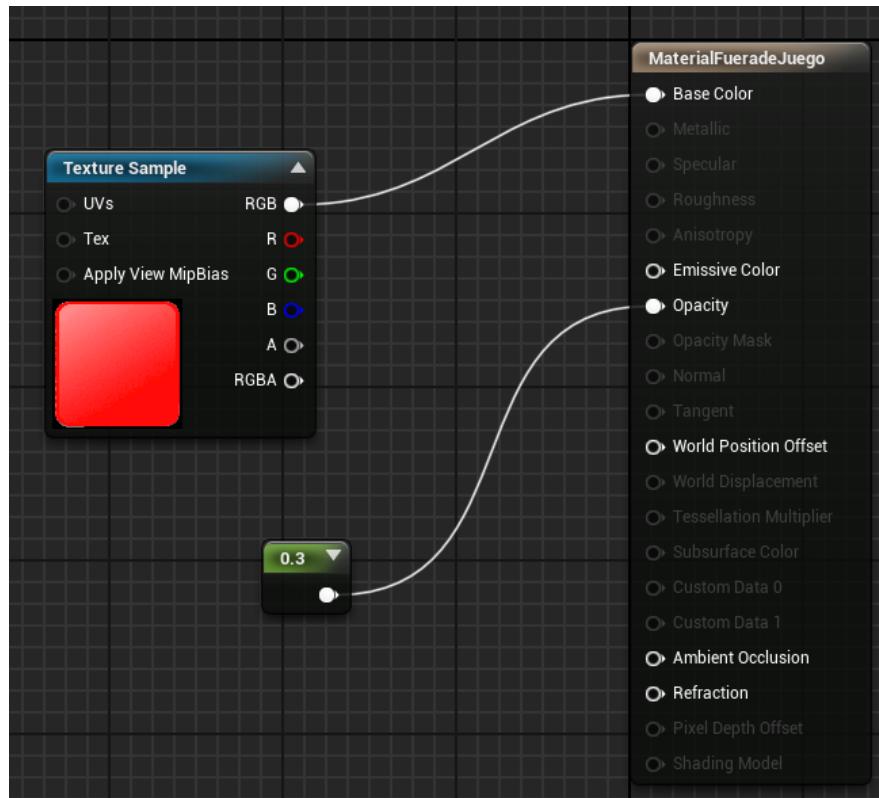
## Implantación de un sistema de fuera de juego

La primera funcionalidad que vamos a añadir para incrementar la dificultad del juego es un sistema de fuera de juego. Originariamente, el fuera de juego es una norma del fútbol en la que un jugador no puede recibir un pase de un compañero si en el momento de ese pase se encuentra por delante de todos los defensas. Sin embargo, como en nuestro juego solo tenemos un solo jugador y no podemos recibir pases, tenemos que redefinir este concepto de fuera de juego.

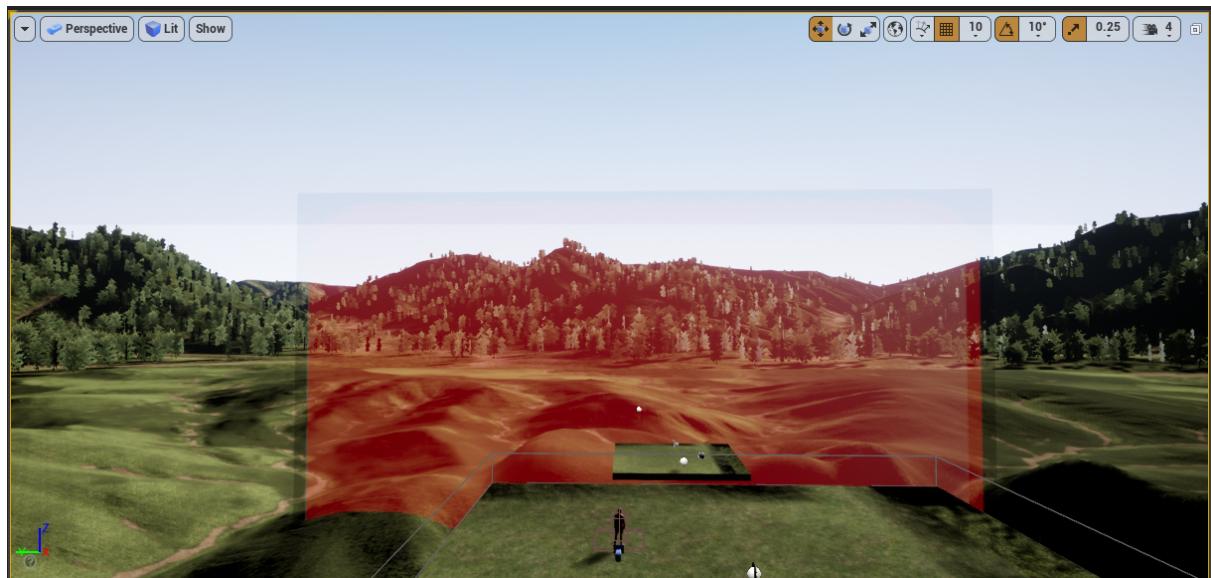
En nuestro caso, vamos a llamar fuera de juego a las situaciones en las que nuestro personaje no avanza por el escenario a una velocidad adecuada. Es decir, vamos a disponer de un plano que va a ir barriendo todo el escenario lentamente y si este plano alcanza a nuestro personaje, este inmediatamente se encontrará en fuera de juego y perderá.

Para diseñar este sistema de fuera de juego, en primer lugar, vamos a crear un nuevo blueprint class de tipo actor llamado AreaFueradeJuego el cual va a estar formado por un plano del mismo estilo al utilizado en AreaGol y AreaFuera. Sin embargo, en este caso, no queremos que el plano sea transparente del todo si no que sea visible pero deje pasar la mayor parte de la luz.

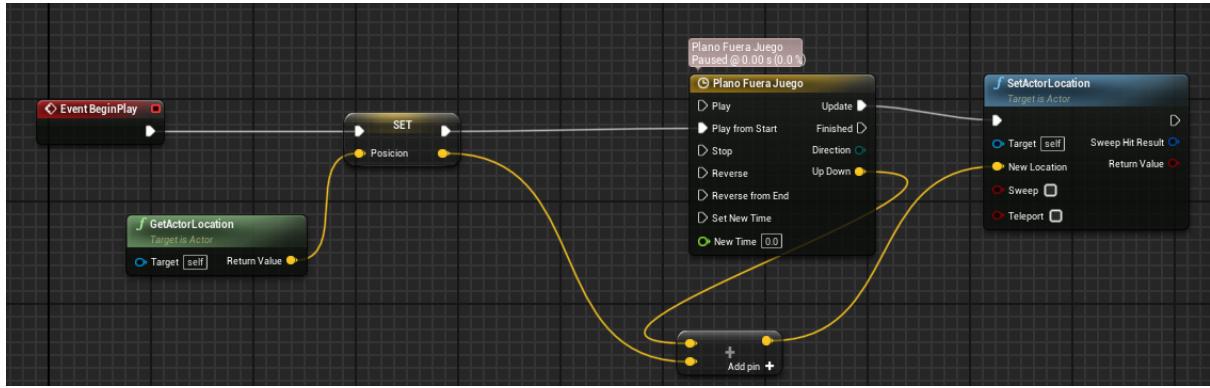
Para ello, vamos a crear un nuevo material, donde en primer lugar, definiremos el Blend Mode de este material en modo translúcido. En segundo lugar, crearemos una textura a partir de la foto de la tarjeta roja con el objetivo de aplicar esta textura al material. En tercer lugar, le daremos un valor de opacidad de 0.3 al material para que el material se vea rojo pero deje pasar la mayor parte de la luz.



Una vez hemos añadido la textura al plano de fuera de juego, cambiamos la gestión de las colisiones en OverlapAllDynamic para que no detecte las colisiones con los objetos y arrastramos este blueprint al escenario con las dimensiones adecuadas para que pueda barrer todo el nivel.

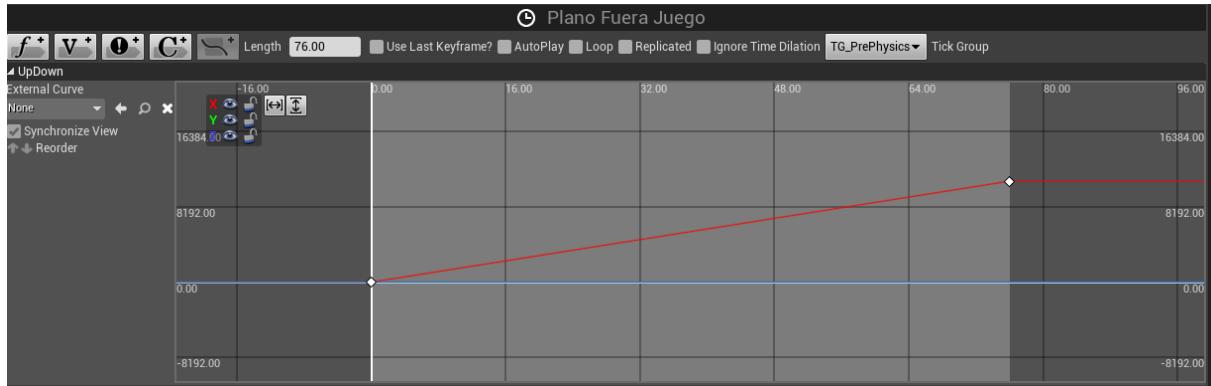


Ahora, vamos a añadir un código al blueprint `AreaFueradeJuego` para que el plano de fuera de juego se vaya desplazando lentamente y vaya barriendo todo el campo de juego. Para ello nos vamos a ayudar de una timeline la cual va a gestionar el movimiento en el eje X del plano de fuera de juego en función del tiempo.



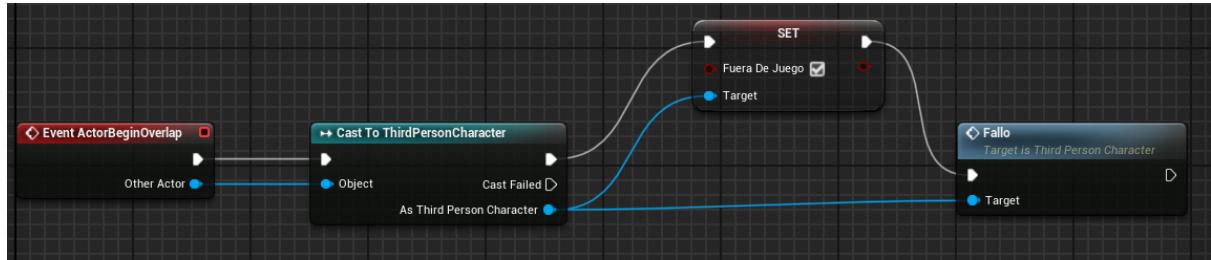
En este código, cuando se ejecute el nivel (evento BeginPlay), se va a obtener la posición del plano de fuera de juego (nodo GetActorLocation) y se va a almacenar en una variable llamada Posición. Después, vamos a definir una Timeline llamada Plano FuerJuego en la cual vamos a definir cuál va a ser el valor del componente X (con respecto a la posición inicial) del plano de fuera de juego en función del tiempo. Este valor del componente X se va a sumar a la posición inicial del plano continuamente de forma que el resultado de la suma se va a ir utilizando para ir ajustando la posición del plano con el nodo SetActorLocation.

La variación del componente X del plano de fuera de juego (con respecto a la posición inicial) va a variar siguiendo la siguiente función:



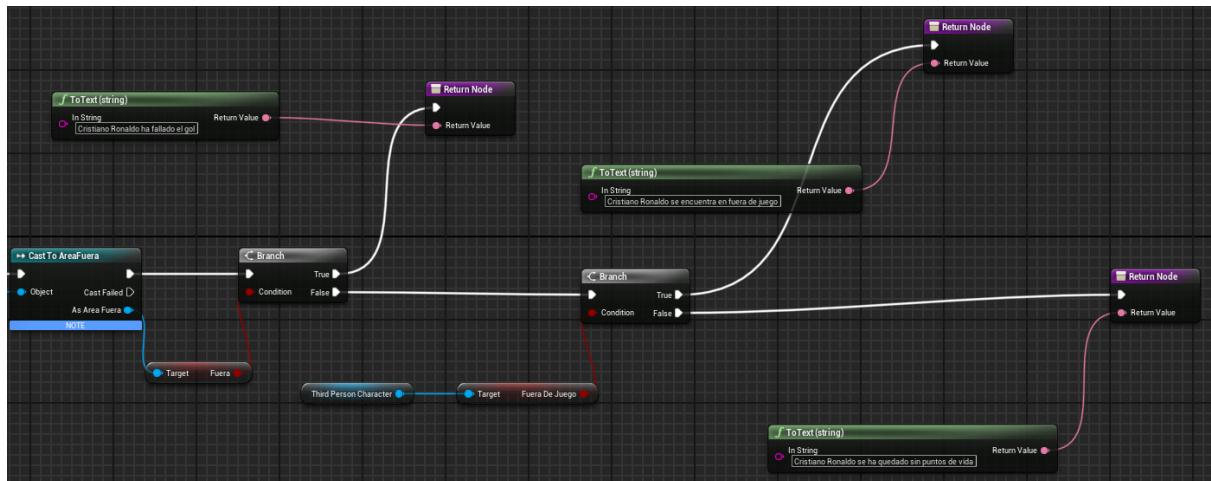
Como se puede observar, el plano se mueve desde la posición X 0 a la posición X 11000 en un intervalo de 76 segundos. Para variar la velocidad en la que se mueve este plano, simplemente hay que regular el tiempo que emplea el plano en ir desde la posición X 0 a la posición X 11000.

Una vez hemos conseguido que el plano de área de juego se mueva, el siguiente paso es conseguir que cuando el plano nos atraviese, el jugador pierda automáticamente y se muestre el menú de GameOver. Para ello, vamos a incluir el siguiente código en el blueprint del actor AreaFueradeJuego.

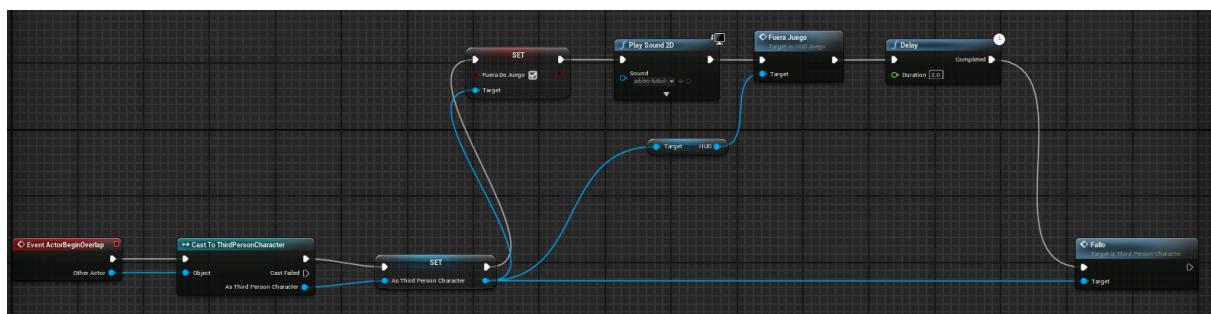


En este código, básicamente, cuando el plano atraviesa un objeto, se hace un cast al ThirdPersonCharacter para ver si el objeto atravesado es nuestro personaje. En caso de que así sea, se pone a verdadero una variable booleana fuera de juego que hemos creado en el personaje, la cual será utilizada por el menú de GameOver para determinar el motivo de la derrota. Por último, llamamos al evento Fallo ya que queremos que se reproduzcan las mismas acciones que cuando nuestro personaje falla el gol.

Por último, incluimos una porción de código en la función del binding del menú GameOver para indicar que si nuestro personaje está en fuera de juego, que muestre un mensaje indicando que el personaje ha perdido por estar en fuera de juego.



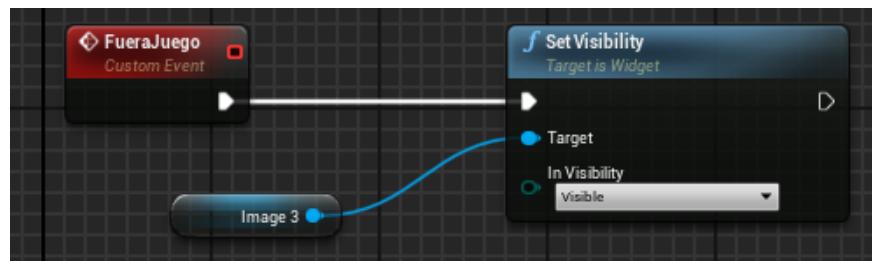
Con esto el sistema de fuera de juego está correctamente implementado. Sin embargo, queremos darle un mayor efecto al fuera de juego. Debido a ello, vamos a añadir una porción de código al blueprint AreaFueradeJuego para que se reproduzca un sonido de silbato (al igual que con las tarjetas) cuando entremos en fuera de juego y para añadir al HUD un imagen alertando que estamos en fuera de juego.



Para conseguir este efecto, vamos a reproducir el sonido de silbato con Play Sound 2D y después vamos a llamar al evento Fuera Juego el cual está definido dentro del HUD y se encarga de hacer visible una imagen con un banderín de fuera de juego.

Por otro lado, en el HUD vamos a añadir una imagen de un banderín de fuera de juego (la cual hemos importado a nuestro proyecto) y vamos a ocultar en primera instancia esta imagen al igual que hacemos con las tarjetas y con la imagen de x2.

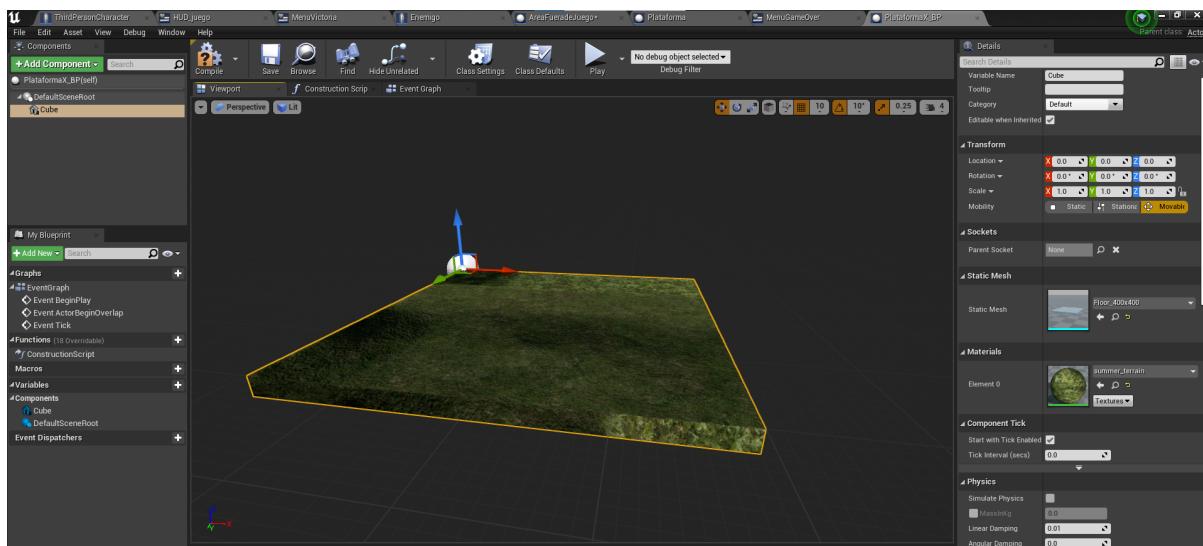
Por último, añadiremos un evento customizado llamado Fuera Juego el cual hará visible esta imagen del banderín de fuera de juego. Este evento como hemos visto se invoca desde el blueprint del Área de Fuera de Juego.



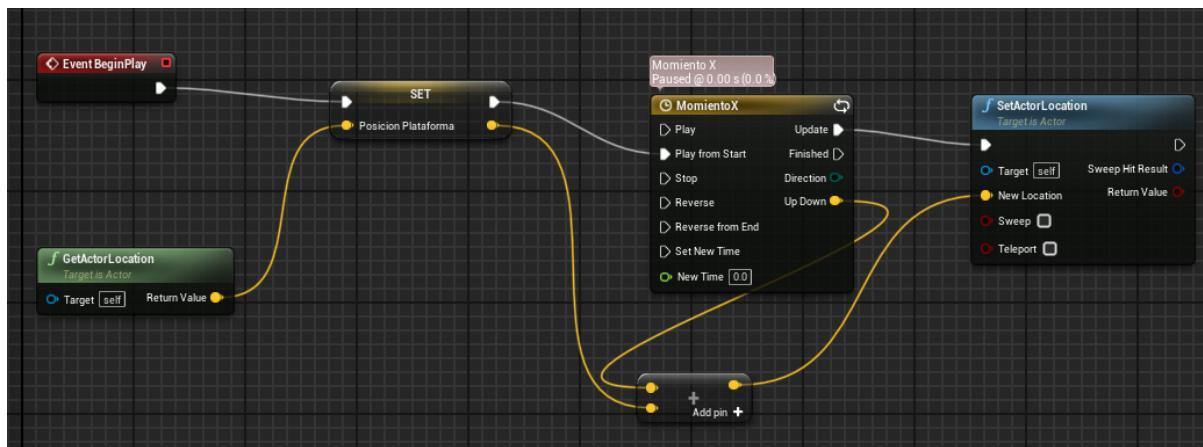
## Implantación de plataformas móviles

Una vez hemos implantado el sistema de fuera de juego, lo siguiente que vamos a hacer es implantar una serie de plataformas móviles de forma que nuestro personaje tenga que saltar en el momento adecuado para ir de una plataforma a otra. Para ello, utilizaremos una mecánica similar a la del fuera de juego ya que necesitamos el uso de timelines para gestionar el movimiento de las plataformas.

En primer lugar, vamos a crear un nuevo blueprint class de tipo actor llamado PlataformaX\_BP el cual implementará una plataforma que se desplaza en el eje X. Esta plataforma estará formada por un cubo donde le asociaremos como malla estática un suelo de 400x400 y este cubo tendrá una textura de hierba (la misma textura que las utilizadas para las otras plataformas).

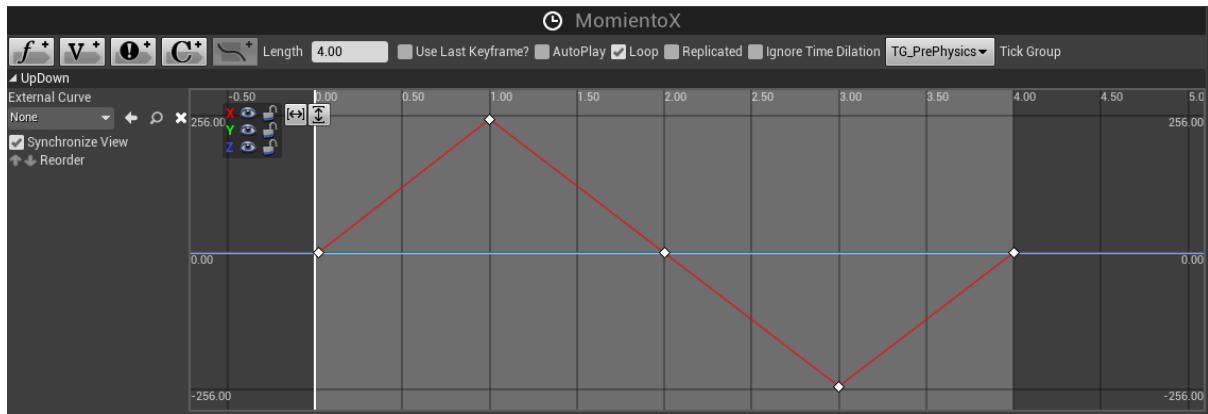


Ahora, vamos a añadir un código al blueprint PlataformaX\_BP para que la plataforma se vaya desplazando de un lado a otro en el eje X. Para ello nos vamos a ayudar de una timeline la cual va a gestionar el movimiento en el eje X de la plataforma en función del tiempo.



En este código, cuando se ejecute el nivel (evento BeginPlay), se va a obtener la posición de la plataforma (nodo GetActorLocation) y se va a almacenar en una variable llamada Posición Plataforma. Después, vamos a definir una Timeline llamada MovimientoX en la cual vamos a definir cuál va a ser el valor del componente X (con respecto a la posición inicial) de la plataforma en función del tiempo. Este valor del componente X se va a sumar a la posición inicial de la plataforma continuamente de forma que el resultado de la suma se va a ir utilizando para ir ajustando la posición de la plataforma con el nodo SetActorLocation.

La variación del componente X de la plataforma (con respecto a la posición inicial) va a variar siguiendo la siguiente función:



Como se puede observar, la plataforma se mueve de manera oscilatoria entre las posiciones -250 y 250 en un intervalo de 4 segundos. Para variar la velocidad en la que se mueve esta plataforma, simplemente hay que regular este periodo de oscilación. En este caso, hemos activado la opción de loop pues queremos que el movimiento se realice de manera repetitiva. Esto con el plano de fuera de juego no nos interesaba y por eso no lo marcamos en su momento.

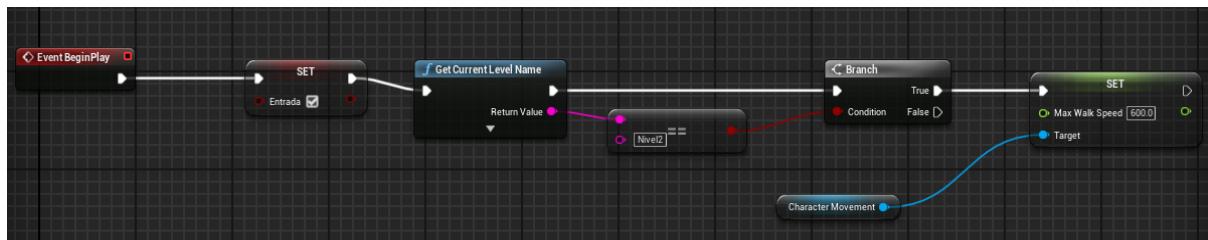
Ahora repetiremos el proceso para conseguir plataformas que se muevan en el eje Y y en el eje Z. El proceso es exactamente el mismo pero cambiando la variable a tratar en la timeline por lo que no lo vamos a volver a explicar. Una vez repetido el proceso tendremos dos blueprint class llamadas PlataformaY\_BP y PlataformaZ\_BP las cuales se mueven en el eje Y y en el eje Z respectivamente.

## Incremento de las capacidades de los enemigos

Por último, para aumentar la dificultad del nivel, vamos a incrementar la velocidad a la que se mueven los enemigos para que puedan perseguir con mayor rapidez a nuestro personaje y que sea más difícil esquivarlos.

Para igualar la condiciones de nuestros enemigos a las nuestras, vamos a proporcionar a los enemigos de una velocidad máxima de 600 al igual que el personaje. Recordamos que la velocidad de los enemigos en el nivel 1 era de 400.

Para aumentar la velocidad de los enemigos en el nivel 2 vamos a añadir el siguiente código en el evento BeginPlay del enemigo:

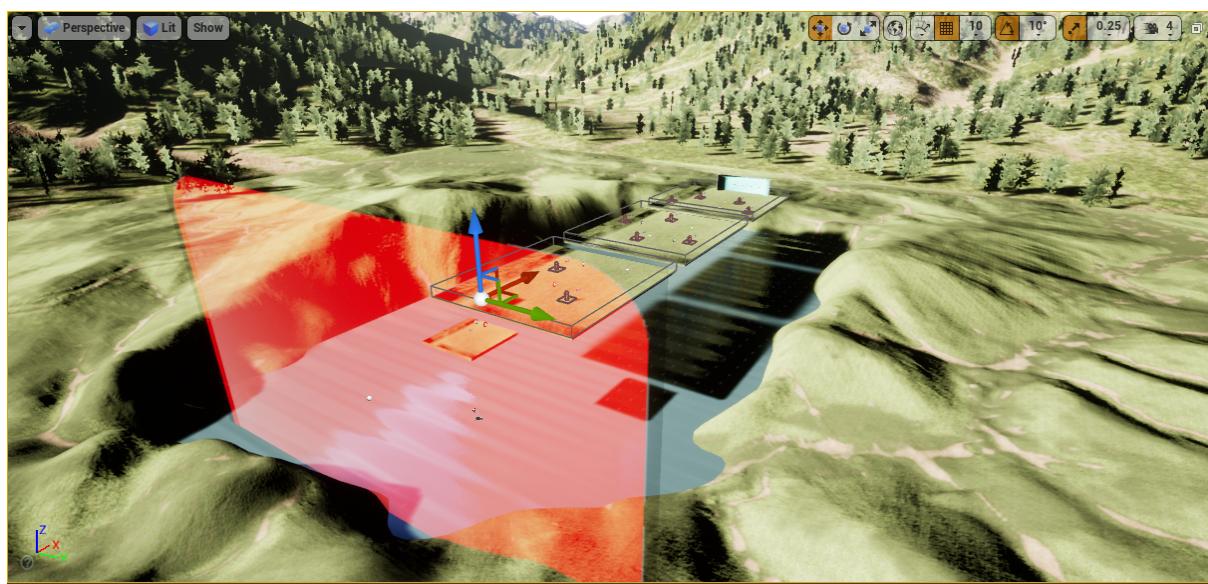


En este código, se obtiene el nombre del nivel actual con Get Current Level Name y se compara el nombre del nivel actual con la cadena Nivel2. Si estamos en el nivel 2, la comparación dará un valor verdadero, por lo que establecemos la velocidad máxima del enemigo a 600.

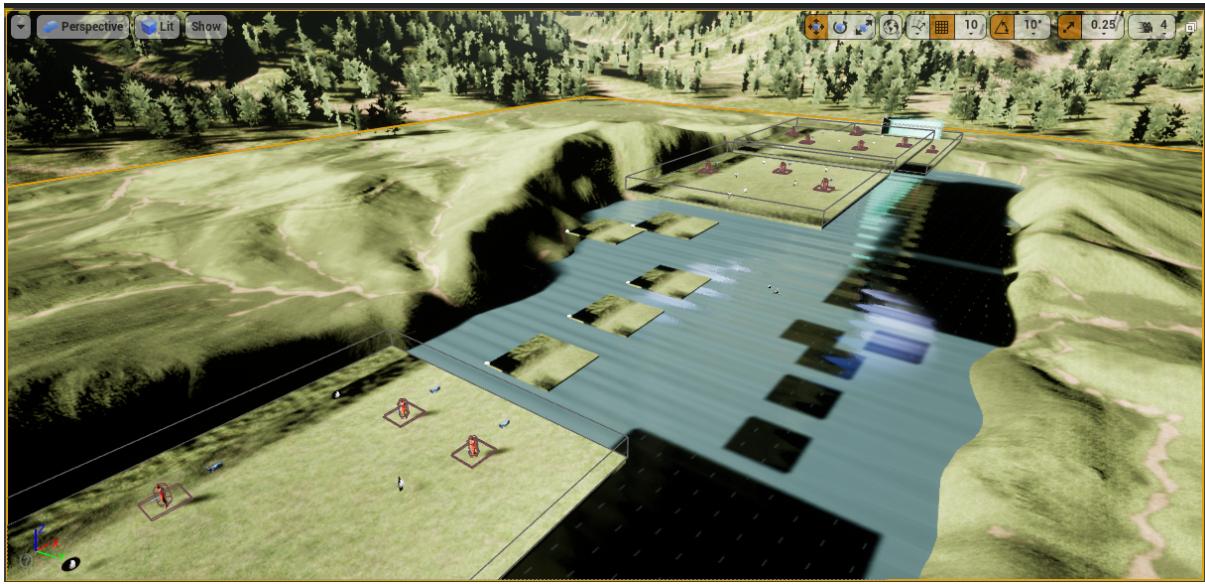
## Diseño del segundo nivel

Una vez hemos incorporado estas funcionalidades adicionales para aumentar la complejidad de nivel, ahora vamos a complicar el nivel de manera estructural añadiendo más plataformas, más enemigos, más objetos y haremos uso de las plataformas móviles.

Inicialmente la disposición de nuestro nivel 2 es el siguiente (recordamos que es una copia del escenario del nivel 1):

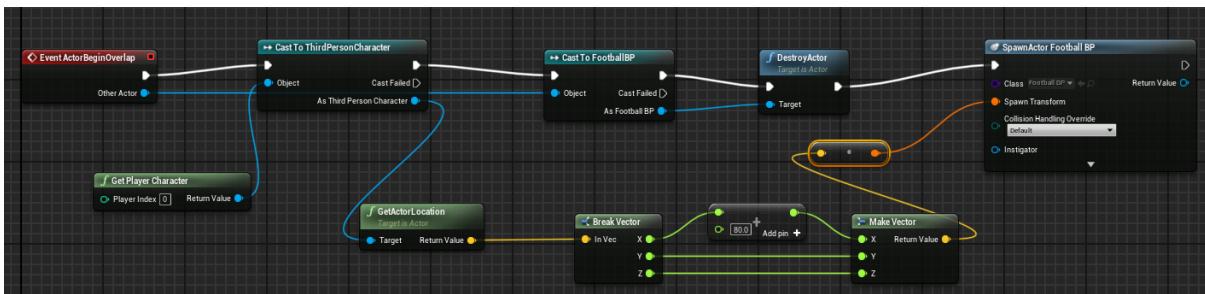


De esta manera, vamos a utilizar la misma lógica que la utilizada en el nivel 1 pero ahora, vamos a hacer un nivel más largo y vamos a hacer uso de plataformas móviles. Tras realizar el diseño del nivel 2 nos queda algo como esto:



Como se puede observar, en la mitad del nivel hay una serie de plataformas móviles que el personaje debe superar con el balón. Como saltar de plataforma en plataforma móvil con el balón es algo bastante difícil de hacer sin que se caiga el balón al agua, hemos decidido que en esta zona cada vez que se le caiga el balón, inmediatamente se le colocará un nuevo balón en los pies del personaje. Así, el jugador puede tener muchos intentos de saltar las plataformas con el balón y el nivel no se hace así tan difícil.

Para reponer los balones en esta zona, vamos a crear un blueprint class de tipo actor llamado Devolver\_BalonBP que tendrá la misión de reponer los balones al personaje. Este actor será un plano transparente que abarca toda la zona de las plataformas móviles, el cual tendrá sus colisiones en AllOverlapDynamic. De esta manera, cuando el balón atraviese este plano se ejecutará el siguiente código:



En este código, cuando un objeto traspase este plano (evento BeginOverlap) primero se obtiene una referencia al personaje con los nodos Cast to ThirdPersonCharacter y Get Player Character. Después, se comprueba con el nodo Cast To FootballBP si el objeto que atravesó el plano es el balón. En caso de que otro objeto atraviese el plano, este casting fallará y finalizará la ejecución de este código.

Después, se obtendrá la posición del personaje con el nodo GetActorLocation y se incrementará en 80 en la coordenada X para obtener la posición en la que debe reponerse el balón (justo delante del personaje). Por último, se destruirá la instancia del balón que está

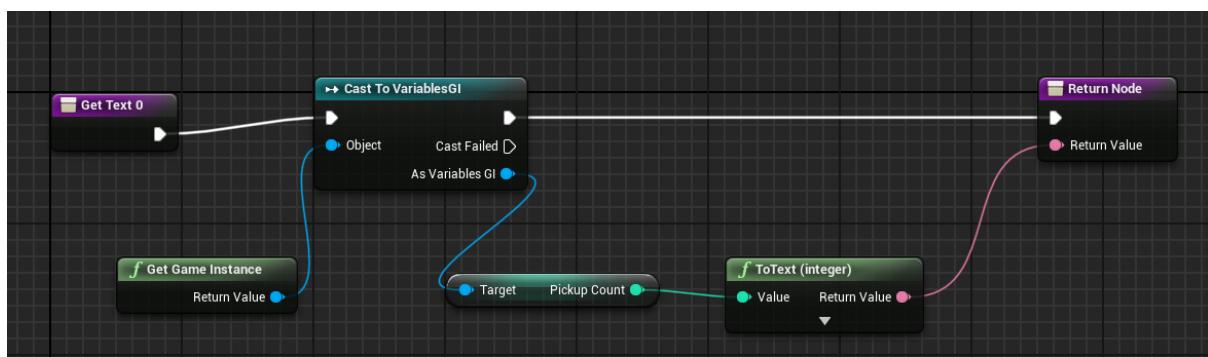
cayendo al agua (Destroy Actor) y se creará otra nueva instancia del balón justo delante del personaje (Spawn Actor FootballBP).

## Menú Final

Para concluir el segundo nivel, vamos a diseñar un menú final el cual nos indique que el juego ha finalizado además de indicarnos la puntuación final del juego. En nuestro caso, la puntuación va a ir determinada por los balones de oro recogidos por el jugador, que será la información que se muestre por pantalla. El diseño del menú final será el siguiente:

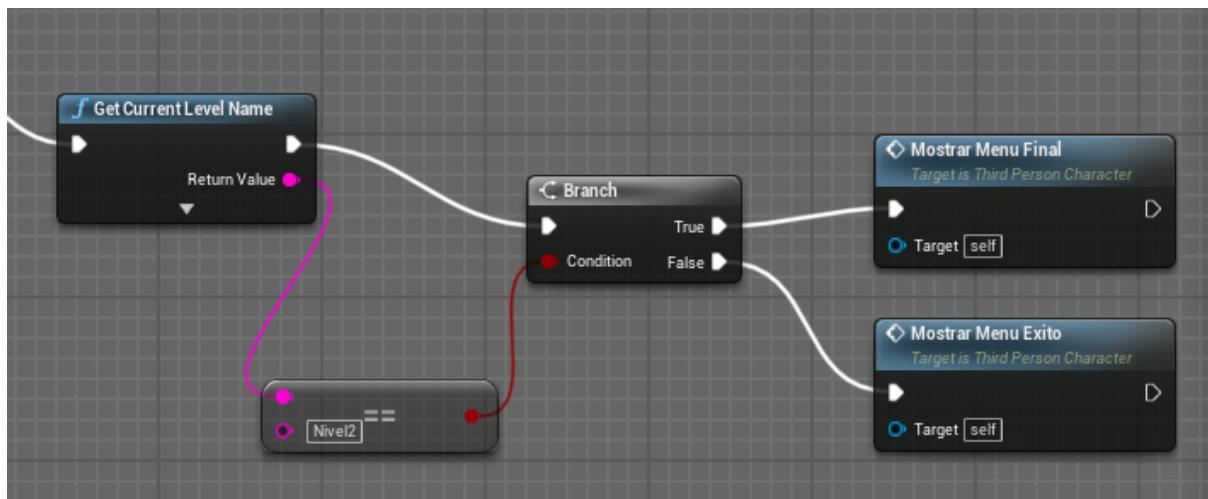


Como se puede observar, en la parte de la derecha tenemos un texto que va a indicar el número de balones de oro recogidos por el jugador en todo el juego. Para mostrar esta cantidad de balones de oro recogidos en total, vamos a hacer un binding a esta cadena de texto y vamos a definir una función que nos devuelva el número total de balones de oro recogidos por el jugador.



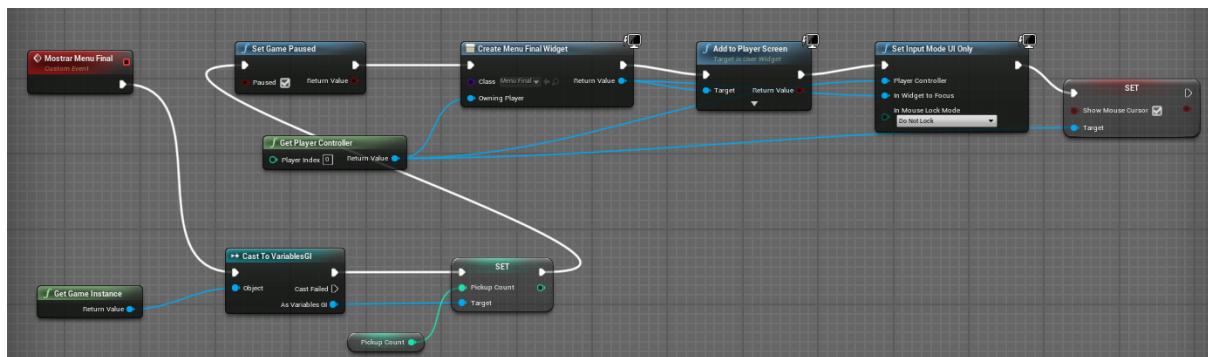
En esta función, básicamente se obtiene una referencia del GameInstance a través los nodos Cast To VariablesGI y Get Game Instance y después se utiliza esta referencia al GameInstance para obtener la cantidad total de los balones de oro recogidos por el juego para mostrar esta información por pantalla.

Por último, vamos a configurar el Blueprint del personaje de forma que se muestre el menú de éxito cuando el nivel 1 finalice y que se muestre el menú final cuando finalice el nivel 2. Para ello hemos utilizado el siguiente código:



Básicamente, en esta porción de código, se obtiene el nombre del nivel actual con Get Current Level Name y se evalúa el nombre del nivel activo de forma que si el nivel activo es el nivel 2 se llama al evento Mostrar Menu Final y si el nivel activo es el nivel 1 se llama al evento Mostrar Menu Éxito.

Para mostrar el menú final, se utilizan el evento Mostrar Menu Final el cual tendrá el siguiente código:



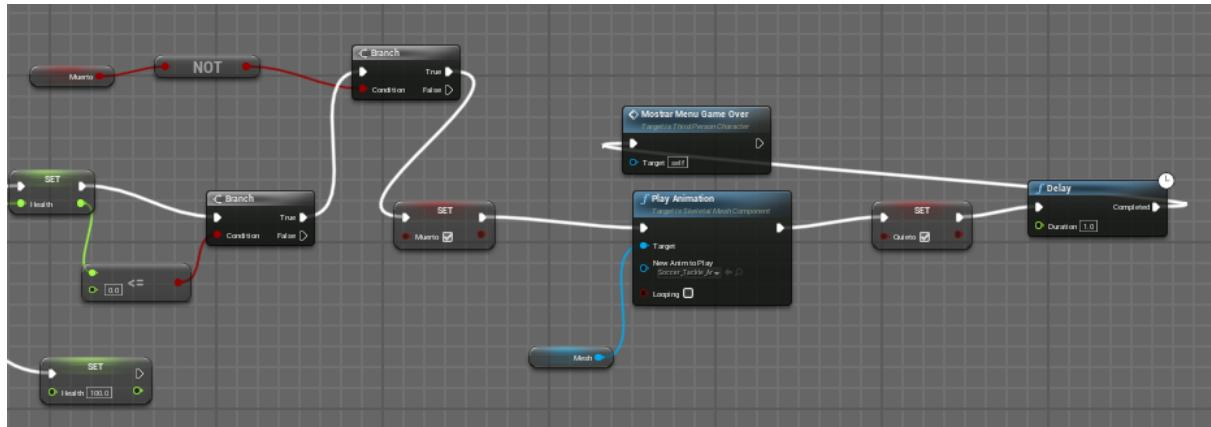
Las acciones realizadas en este código son iguales a las realizadas en el evento Mostrar Menu Éxito por lo que no vamos a volver a explicarlas.

Con esto el juego estaría terminado. Lo único que quedaría es revisar el tiempo asignado para la gestión del fuera de juego para que el plano se mueva a una velocidad adecuada teniendo en cuenta la dificultad del nivel.

## Correcciones

Tras probar la mecánica y jugabilidad del juego hemos detectado algunos pequeños aspectos que pueden mejorar. Para no complicar más la explicación vamos a contar de manera breve los cambios realizados y el código modificado o añadido para corregir estos fallos.

En primer lugar, hemos detectado que nuestro personaje cuando es alcanzado por varios enemigos de forma seguida es posible que reproduzca la animación de “muerte” varias veces ya que puede seguir recibiendo entradas cuando está realizando la animación de morir. Para corregir esto, hemos realizado estos cambios en la gestión del evento Any Damage.

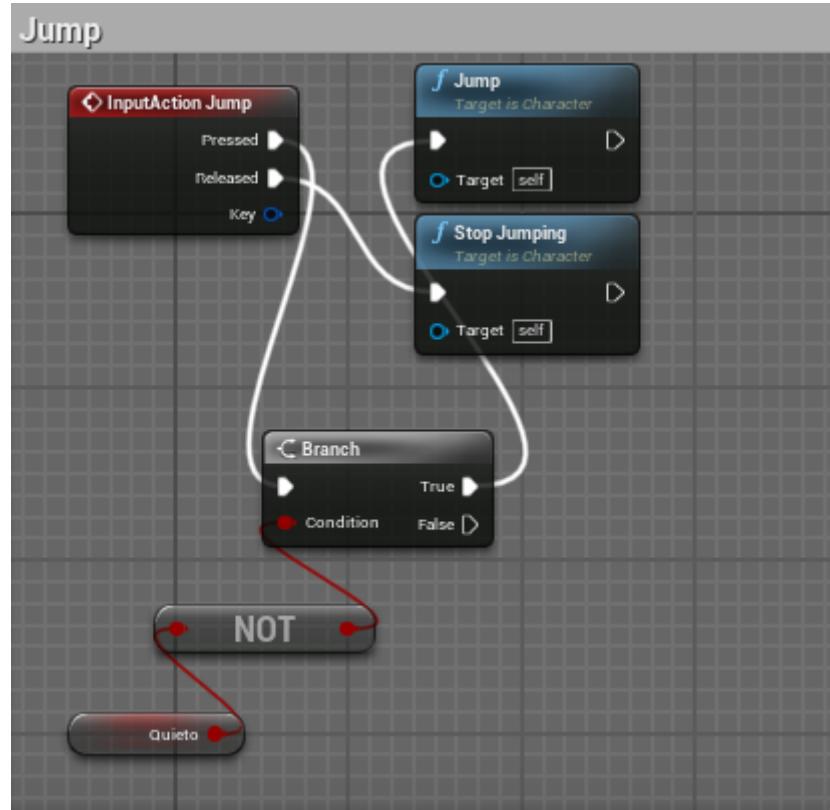


Estos cambios se basan en la introducción de una variable booleana llamada muerto la cual se pone a true cuando nuestro personaje se queda sin puntos de vida. De esta manera, si inmediatamente otro enemigo impacta al jugador cuando este ya no tiene puntos de vida, la variable muerto será true por lo que no se reproducirá la animación de “muerte”.

También se puede observar que hemos congelado el movimiento del personaje cuando el personaje se ha quedado sin puntos de vida, ya que poder mover el personaje cuando está muerto es algo raro.

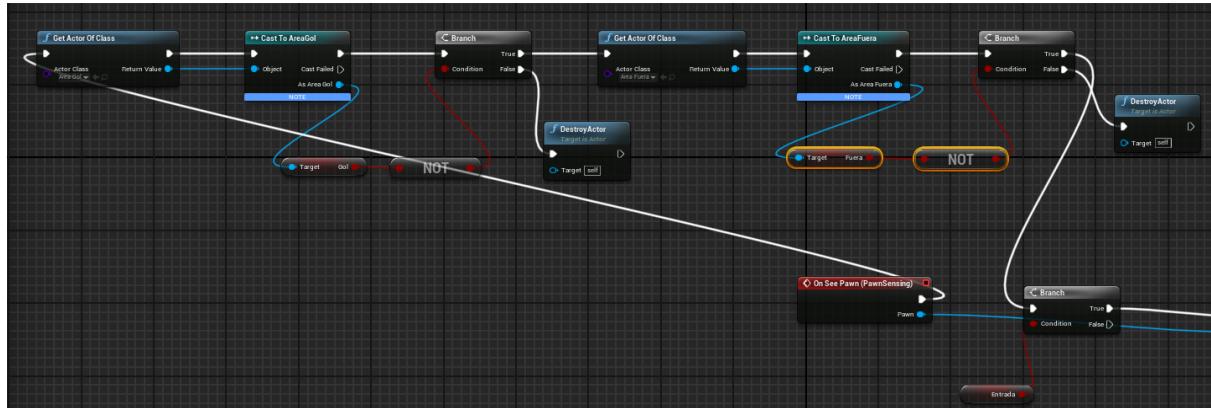
También hemos decidido cambiar la animación de muerte, pues la que estaba puesta no nos convencía y preferimos que se reproduzca la misma animación de caída en todas las situaciones. De esta manera, cuando muera el personaje, la animación de caída será la misma que cuando reciba una entrada con la diferencia de que en esta caída de “muerte” el personaje no gritará pues estará “muerto”.

En segundo lugar, hemos modificado el código de salto que venía creado por defecto por el personaje de forma que vamos a poder bloquear también el salto del personaje en algunas animaciones en las cuales no nos interesa que el personaje se mueva o salte.



Con esto vamos a poder bloquear el movimiento en la animación de caída de forma que el personaje no pueda ni saltar ni moverse cuando reciba el impacto de una entrada. También utilizamos este concepto para bloquear el movimiento y salto del personaje cuando está realizando la caída de muerte como hemos visto antes.

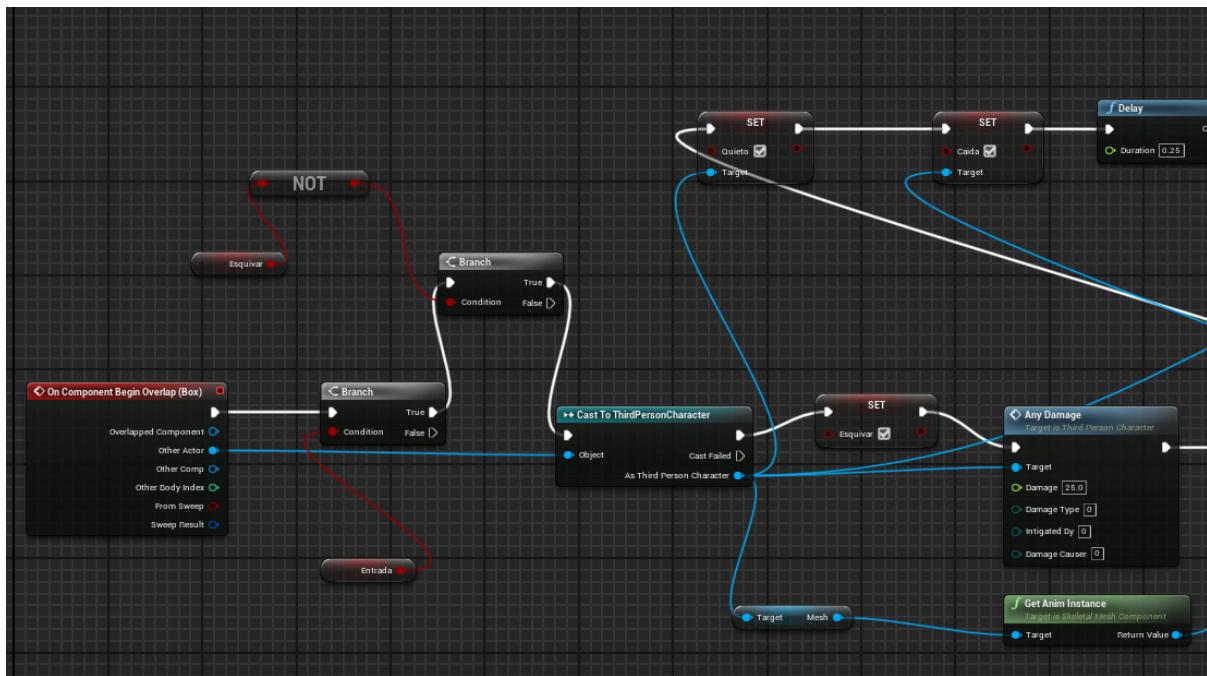
En tercer lugar, hemos detectado que al marcar o fallar un gol, los enemigos que quedan en el campo nos siguen persiguiendo y realizando entradas por lo que nos pueden matar mientras estamos celebrando o lamentando el fallo. Es por esto por lo que hemos modificado el Blueprint del enemigo para destruir a todos los enemigos cuando nuestro personaje marque o falle un gol. El código utilizado es el siguiente:



Con este código, vamos a obtener la referencia del Blueprint AreaGol para evaluar si el personaje ha marcado un gol. En el caso de que se haya marcado el gol (variable gol a true) se destruyen todos los actores de la clase enemigo. Después se va a obtener una

referencia del Blueprint AreaFuera para evaluar si el personaje ha fallado un gol. En el caso de que se haya fallado el gol (variable fuera a true) se destruyen todos los actores de la clase enemigo.

En cuarto lugar, hemos detectado que hay veces que un enemigo al realizar una entrada nos impacta varias veces, por lo que nos quita más vida de la que debería quitarnos. Esto sucede cuando entramos y salimos del box collision del enemigo varias veces antes de que realice la entrada. Para corregir esto, hemos creado una variable booleana llamada Esquivar para asegurarnos que sólo llamamos una vez al evento Any Damage.



Ahora, con estas modificaciones, antes de llamar al evento Any Damage, ponemos a true la variable Esquivar de forma que nos aseguramos así que para este enemigo no se vuelve a llamar al evento Any Damage de nuevo para aplicar más daño. Obsérvese también cómo bloqueamos los movimientos del personaje cuando recibe daño del enemigo de la misma manera que hacíamos en la muerte del personaje.

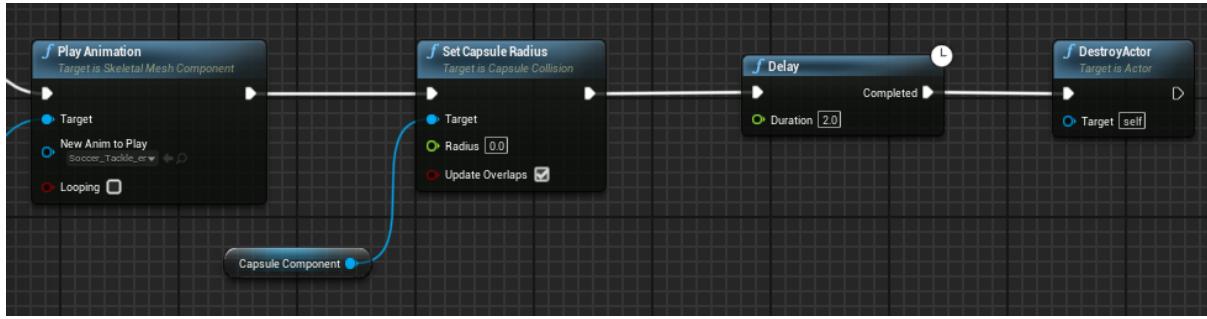
En quinto lugar, hemos detectado que las colisiones de nuestro personaje con el balón no son buenas ya que nuestro personaje tiene un mesh collider en forma de cápsula y al entrar en contacto con el balón, esta superficie redondeada de la cápsula no interactúa bien con el balón. Debido a esto hemos decidido añadir un box collider en los pies de nuestro personaje para que así tenga una superficie lisa para entrar en contacto con el balón y así realizar de mejor manera la conducción de balón.



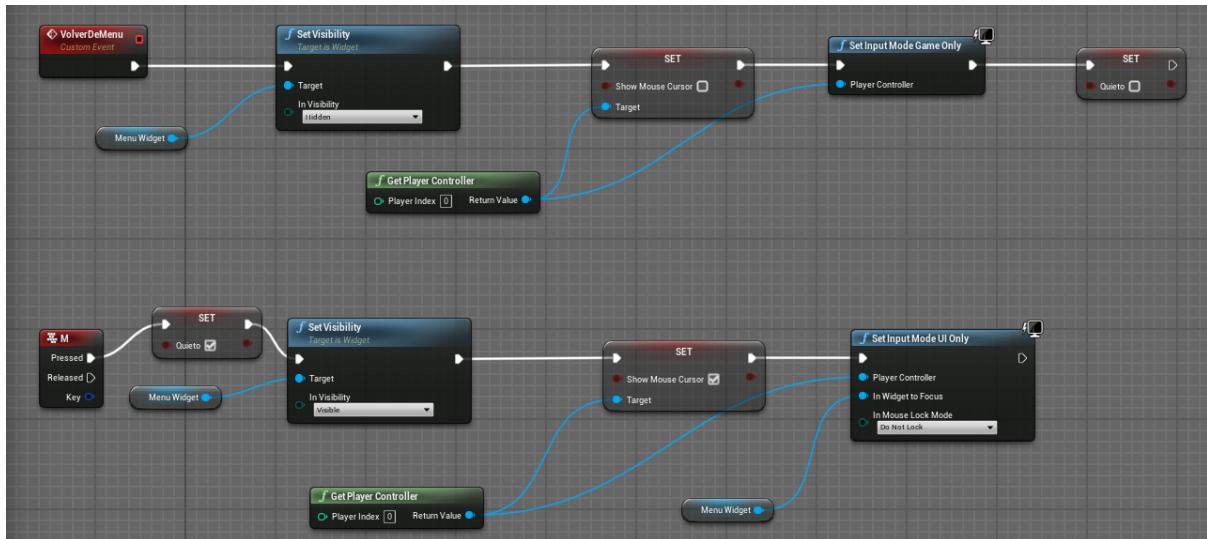
En sexto lugar, hemos observado que a la hora de realizar las entradas, el capsule collider del enemigo permanece en la posición en la que estaba el enemigo justo antes de realizar la segada. Como ya comentábamos antes esta es la razón por la que utilizamos un sistema tan peculiar para detectar las colisiones.

Ya que no podemos mover el capsule collider del enemigo en la animación, hemos decidido que cuando el enemigo realice la entrada, se elimine este capsule collider ya que no hacerlo el personaje se chocara con el capsule collider, aunque realmente no haya enemigo enfrente si no que está en el suelo haciendo la segada.

Para eliminar el capsule collider del enemigo cuando realice la entrada añadiremos el nodo Set Capsule Radius donde reducimos el radio del capsule collider a 0.



El último error que hemos detectado está asociado con el menú de volumen dinámico ya que si mostramos este menú (pulsando la tecla M) mientras estamos corriendo, el personaje seguirá corriendo mientras el menú de volumen está representado en la pantalla. Para solucionar esto bloquearemos el movimiento del personaje cuando el jugador pulse la tecla M y volveremos a habilitar el movimiento cuando se esconda de nuevo el menú de volumen.



Básicamente el cambio es poner a true la variable quieto cuando se pulsa la tecla M y desactivar la variable quieto cuando se esconde el menú (evento VolverdeMenu).