

Memoria Práctica Unreal Engine

[Introducción](#)

[Descripción del Juego](#)

[Diseño y Desarrollo](#)

[Mapas del Juego](#)

[Menu Map](#)

[Primer Nivel: ThirdPersonMap](#)

[Segundo Nivel](#)

[1. Introducción al Nivel](#)

[2. Diseño del Entorno](#)

[3. Mecánicas de Juego](#)

[End Game Map](#)

[Blueprints](#)

[Level Blueprints](#)

[MenuMap_Level](#)

[SegundoNivel_Level](#)

[End Game Map_Level](#)

[Characters Blueprints](#)

[BP_Exo](#)

[BP_Enemigo](#)

[Controller Blueprints](#)

[Game Instance Blueprint](#)

[Animation Blueprints](#)

[ABP_Exo](#)

[ABP_Enemy](#)

[UI Blueprints](#)

[UI_Menu](#)

[UI_WidgetInterface](#)

[UI_EndGame](#)

[Actors Blueprints](#)

[BP_VictoryTrigger](#)

[BP_DeathTrigger](#)

[BP_StartFollowingTrigger](#)

[BP_StopFollowingTrigger](#)

[BP_Rock](#)

[BP_Pyramid](#)

[BP_Plane](#)

[BP_Ball](#)

[BP_SpeedItem](#)

BP_JumplItem

Recursos de Terceros Importados desde Unreal Fab

1. Introducción

El juego, titulado "Exo's Escape", es un proyecto sencillo desarrollado para explorar los fundamentos del diseño y desarrollo de videojuegos utilizando Unreal Engine. La idea nació de la intención de crear un entorno interactivo que combinará la acción y el desafío con una narrativa ligera y envolvente.

El objetivo principal del proyecto era aprender y experimentar con mecánicas de juego como la persecución dinámica, el diseño de entornos y el control de personajes.

Además, el juego busca ofrecer una experiencia entretenida y desafiante al jugador, con una historia que mezcla aventura y superación personal.

2. Descripción del Juego

- **Género:** Aventura y acción en tercera persona.
- **Narrativa:** El protagonista, Exo, es un joven futbolista que debe huir de un misterioso enemigo que lo persigue sin tregua. El objetivo final de Exo es llegar a una isla que representa su salvación, enfrentando obstáculos en su camino y utilizando su agilidad y destreza para sobrevivir.
- **Mecánicas Principales:**
 - Movimiento fluido y control del personaje en tercera persona.
 - Evitar y superar obstáculos en un entorno tridimensional.

3. Diseño y Desarrollo

3.1. Mapas del Juego

El juego incluye cuatro mapas que estructuran la experiencia del jugador:

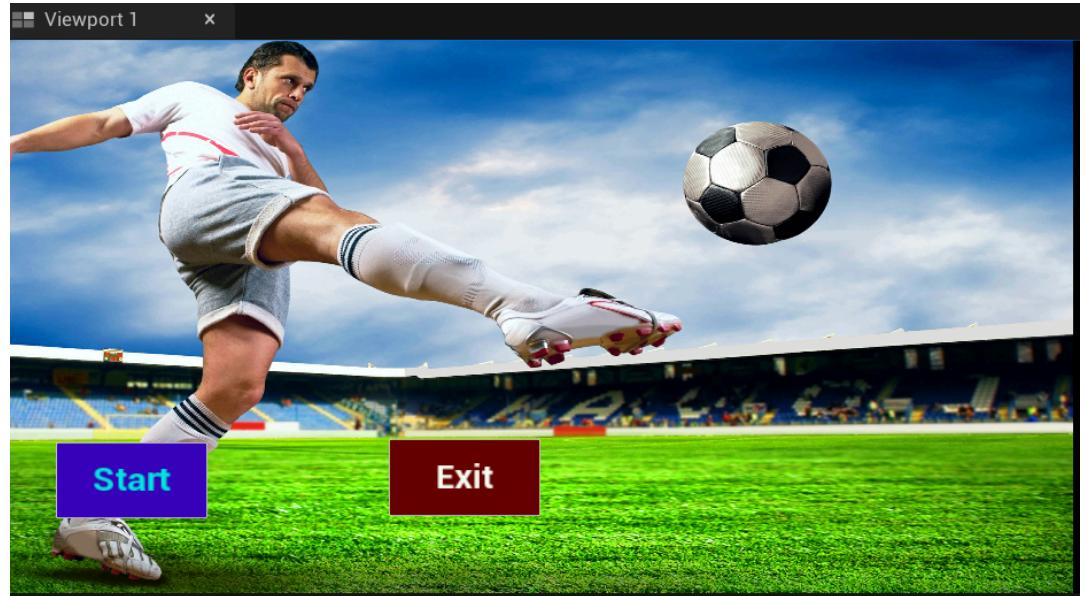
- Menu Map: Mapa inicial que contiene el menú principal.
- Primer Nivel: El primer entorno jugable.
- Segundo Nivel: Un escenario más avanzado en dificultad y diseño.
- End Game Map: Mapa final que presenta la pantalla de conclusión del juego.

3.1.1. Menu Map

El Menu Map es el primer punto de interacción para el jugador. Su diseño es sencillo y funcional, centrado en la accesibilidad.

El menú principal cuenta con dos botones:

- Start: Permite al jugador iniciar el juego, cargando el mapa del primer nivel.
- Exit: Cierra la aplicación.



3.1.2. Primer Nivel: ThirdPersonMap

El primer nivel del juego, llamado ThirdPersonMap (nombre por defecto en Unreal Engine), está diseñado para ofrecer una introducción desafiante al jugador, combinando exploración y objetivos claros.

Diseño del Terreno

El terreno principal es un cubo que actúa como campo de juego y utiliza un material personalizado llamado Pitch Material. Este material fue creado con texturas importadas desde la biblioteca de Unreal (Fab), simulando un campo de fútbol.

Para reforzar la temática futbolística:

Se añadieron cuatro Static Mesh representando porterías de fútbol, colocadas estratégicamente en el campo.

El entorno incluye varias pelotas diseñadas como una clase Blueprint llamada **BP_Ball**.

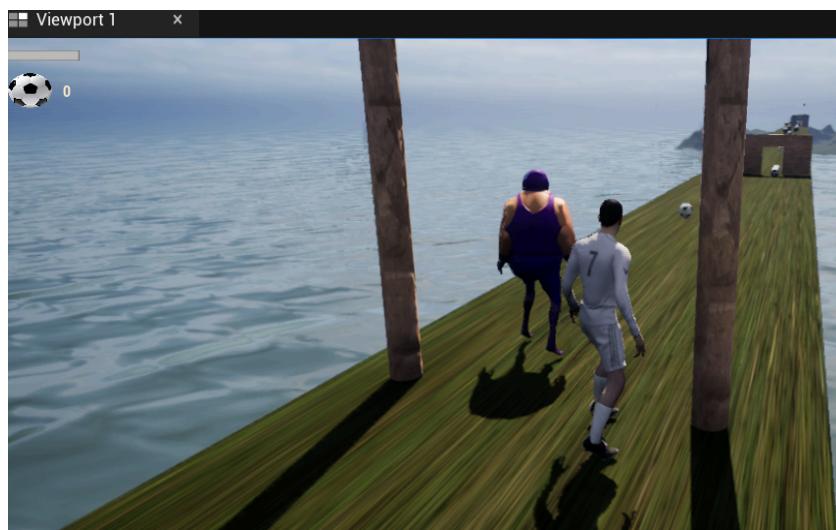


3.1.3. Segundo Nivel

1. Introducción al Nivel

Tema del Nivel: El jugador debe atravesar un puente situado sobre el mar, superando obstáculos y recogiendo ítems mientras un enemigo lo persigue. El objetivo final es alcanzar una isla al otro lado.

Estilo Visual: Entorno marítimo con un puente elevado que combina elementos básicos de arquitectura y formas geométricas.

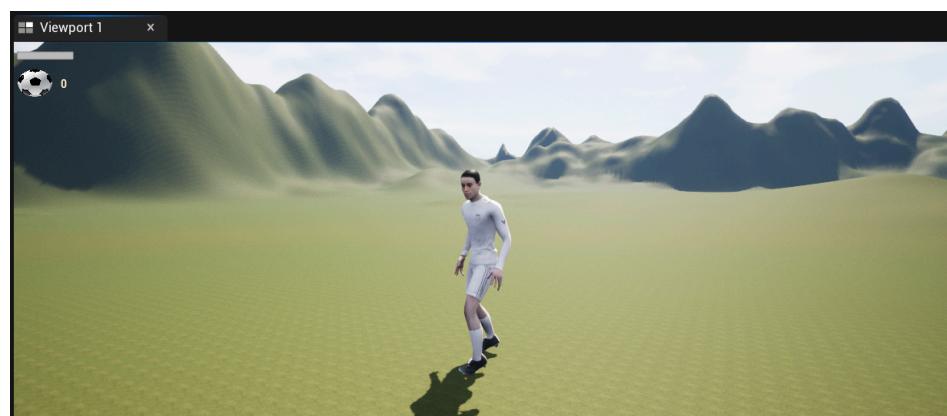


2. Diseño del Entorno

- **Puente:**
 - Creado con un **Shape Plane** del contenido base de Unreal Engine.
 - Representa la base del camino que conecta la entrada del nivel con la isla al final.
- **Obstáculos:**
 - **Formas Geométricas:** Utilizadas desde el **Starter Content**, como esferas que representan rocas y pirámides para estructuras puniagudas.
 - **Elementos Arquitectónicos:** Uso del **Wall Door** de la carpeta "Architecture" para crear pasadizos donde el jugador debe realizar movimientos especiales, como un "sliding" (deslizarse) para pasar debajo.



- **Isla Final:**
 - Representa el objetivo del nivel, una zona segura tras superar todos los desafíos.

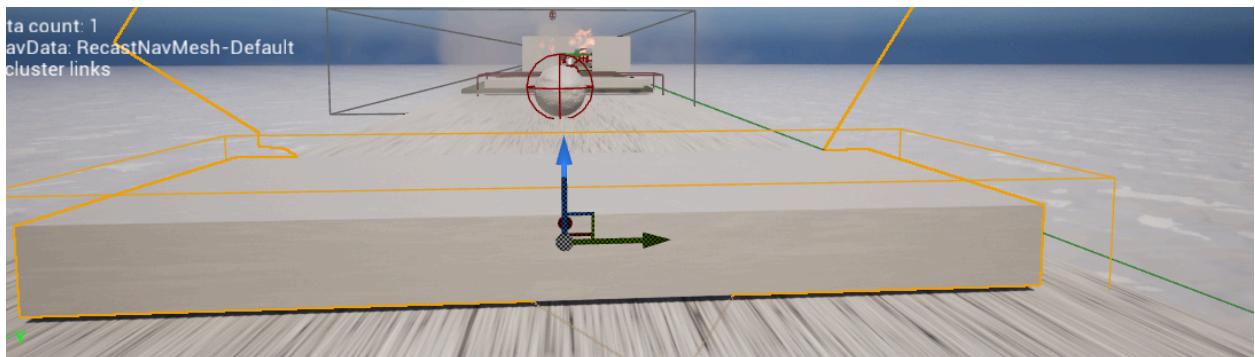
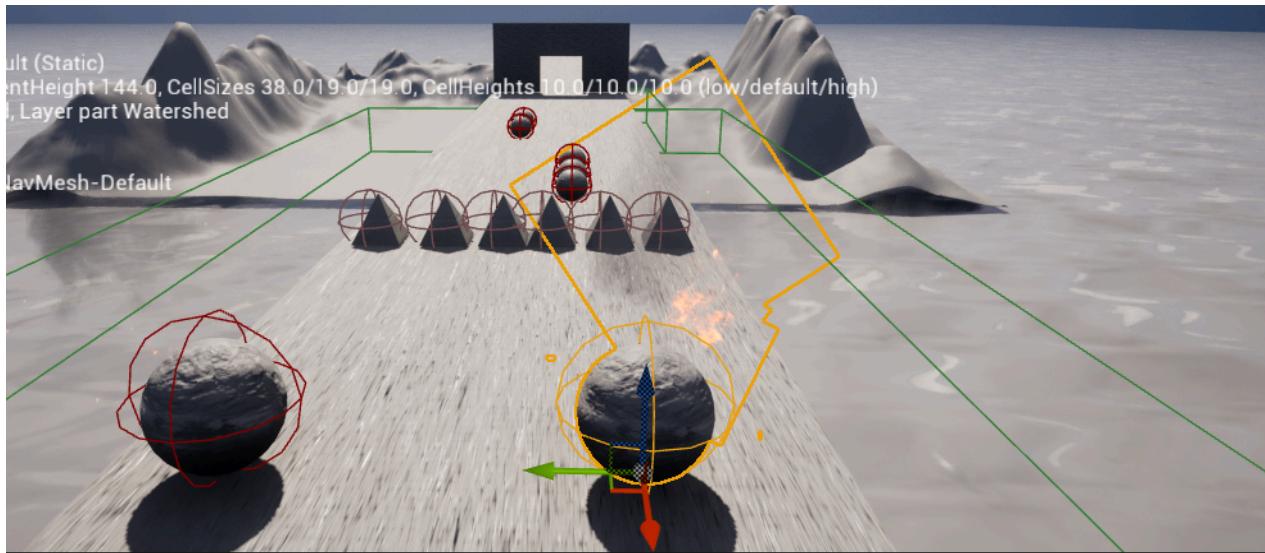


3. Mecánicas de Juego

- Desafíos del Puente:

- Obstáculos:

- **Rocas (Esferas)**: Incorporan un sistema de partículas de fuego que se activa constantemente. Si el jugador entra en contacto con estas rocas, su salud se reduce debido al daño por fuego.
 - **Pirámides**: Actúan como barreras puntiagudas. Reducen la salud del jugador si las toca, pero no tienen partículas asociadas.
 - **Cubo con humo**: Este obstáculo utiliza partículas de humo para simular un efecto de toxicidad. Si el jugador entra en contacto con él, su salud también se reduce.



- Enemigo:

- Persigue al jugador durante el nivel.

- Si alcanza al jugador, se activa un evento de "Game Over".



- **Ítems:**

- **Pelotas:** Recolectables básicos que aumentan la puntuación del jugador.



- **SpeedItem:** Incrementa temporalmente la velocidad del jugador para ayudar a escapar del enemigo o superar secciones rápidas.



- **JumplItem:** Mejora temporalmente la capacidad de salto, útil para superar grandes obstáculos o áreas elevadas.

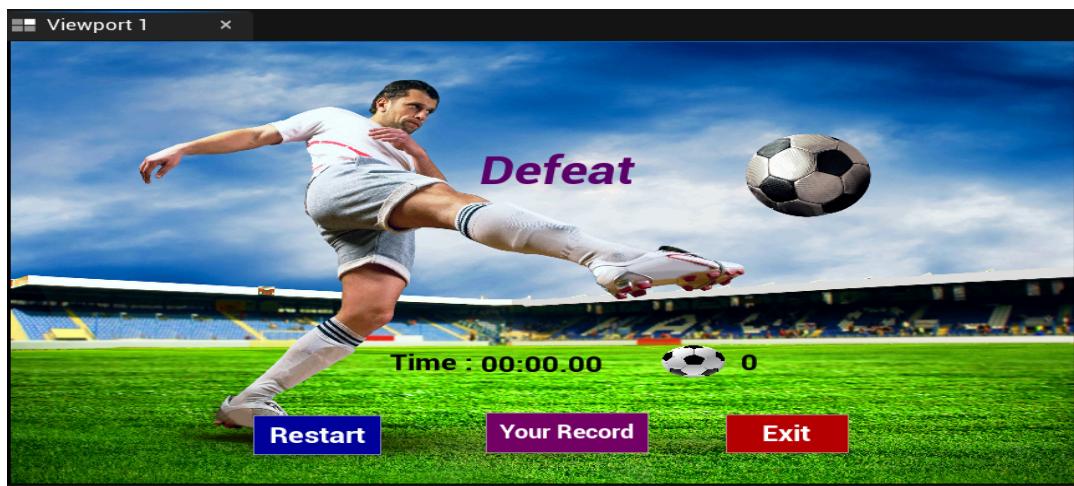


3.1.4. End Game Map

El **End Game Map** sirve como la pantalla final del juego, proporcionando al jugador opciones clave al completar o finalizar su partida. Este nivel está diseñado para ser funcional y centrado en la interacción con la interfaz de usuario.

- **Interfaz del Menú Final:**

- **Diseño:** El menú ocupa toda la pantalla, utilizando un *Canvas Panel* como base para organizar los botones.
- **Botones Disponibles:**
 1. **Restart:** Permite reiniciar el juego desde el primer nivel.
 - Al hacer clic, se carga el mapa inicial (*Map 1*) y se reinician todas las estadísticas del jugador, como tiempo, salud e ítems recolectados.
 2. **Your Record:** Muestra al jugador un resumen de su desempeño:
 - **Tiempo total jugado:** Tiempo desde el inicio hasta la finalización del nivel.
 - **Número de pelotas recolectadas:** Conteo acumulado de ítems recogidos en todos los niveles.
 3. **Exit:** Finaliza el juego cerrando la aplicación.
 - Implementado mediante el nodo "**Quit Game**" para garantizar una salida limpia y directa.



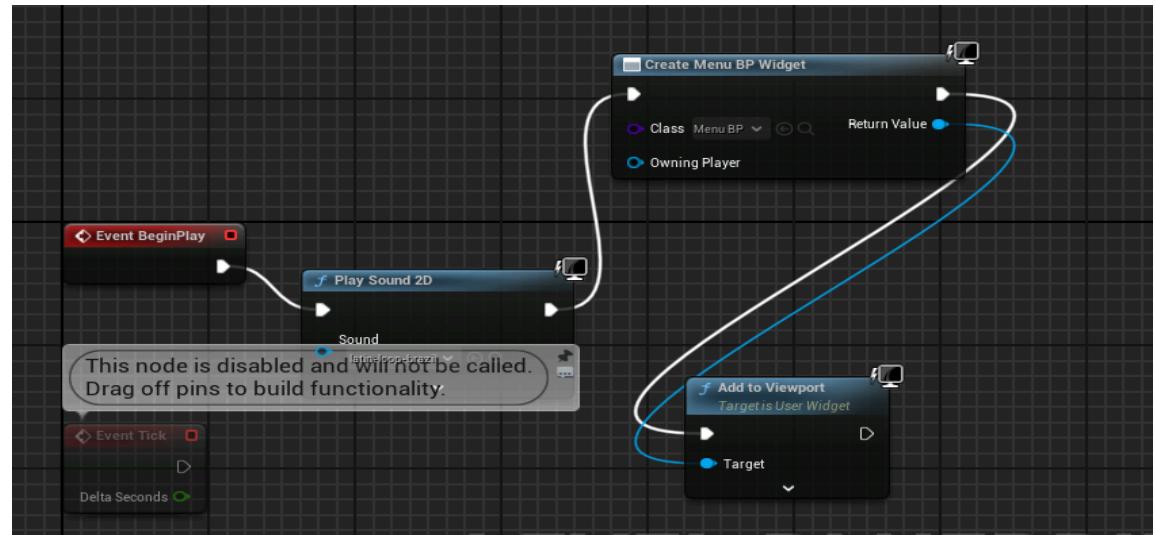
3.2. Blueprints

3.2.1. Level Blueprints

A. MenuMap Level

En el Level Blueprint del Menu Map, se utilizaron los siguientes nodos para mostrar el menú en pantalla:

- **Create Widget:** Este nodo crea una instancia del widget Menu_BP.
- **Add to Viewport:** Este nodo añade el widget a la vista del jugador, haciendo que el menú sea visible y funcional



B. ThirdPersonMap Level

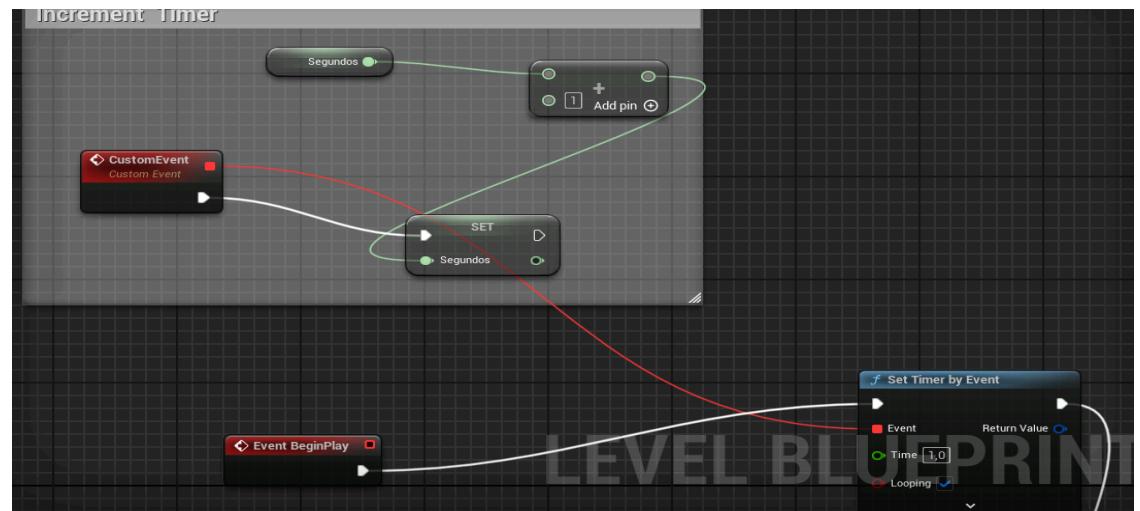
• Cálculo de Tiempo

se utiliza un mecanismo sencillo para medir el tiempo que el jugador permanece en este nivel:

- Inicio del Evento:

- Al comenzar el nivel, se ejecuta el evento Event Begin Play.

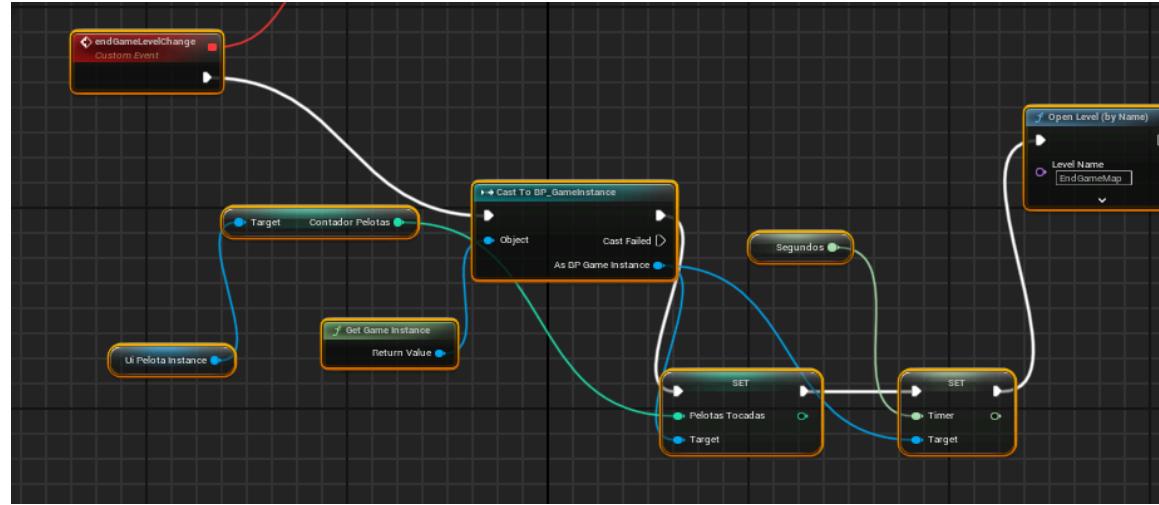
- Configuración del Tiempo:
 - Se utiliza un nodo Set Timer by Event configurado en modo looping, con un intervalo de 1 segundo.
- Incremento de Tiempo:
 - Cada segundo, el evento asociado incrementa la variable segundos en una unidad, llevando el conteo total del tiempo transcurrido.



● Segundo Mecanismo: Evento de Muerte

En el Level Blueprint de ThirdPersonMap, se implementa un mecanismo para detectar cuando el jugador cae:

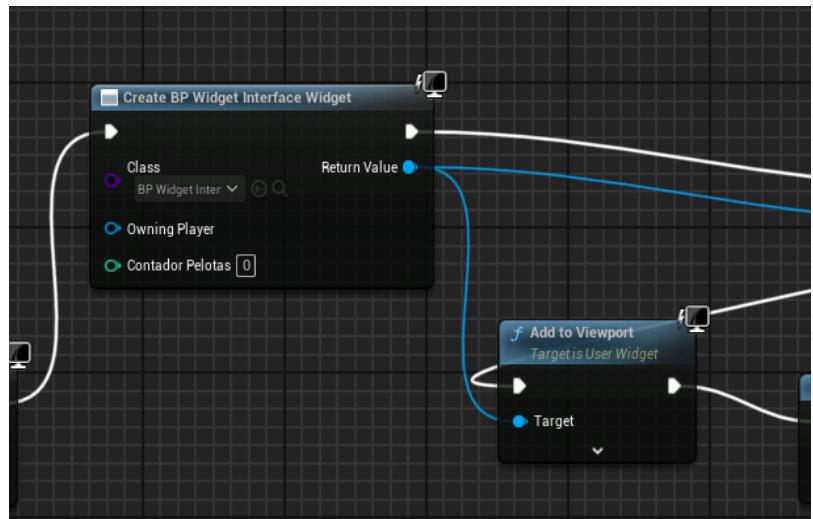
- Obtener Actor de Clase:
 - Se usa el nodo Get Actor of Class para obtener el actor BP_DeathTrigger.
- Event Dispatcher:
 - Se realiza un Bind con el Event Dispatcher llamado Has Fallen.
- Manejador de Evento:
 - Un manejador de este evento se asocia para activar la transición al End Game Map, lo que indica que el jugador ha perdido.
 - se guarda el valor de segundos y de pelotas tocadas en GameInstance



- **Tercer Mecanismo: Barra de Vida y Contador de Pelotas**

En el Level Blueprint de ThirdPersonMap, se añade un widget a la pantalla para mostrar información importante del jugador:

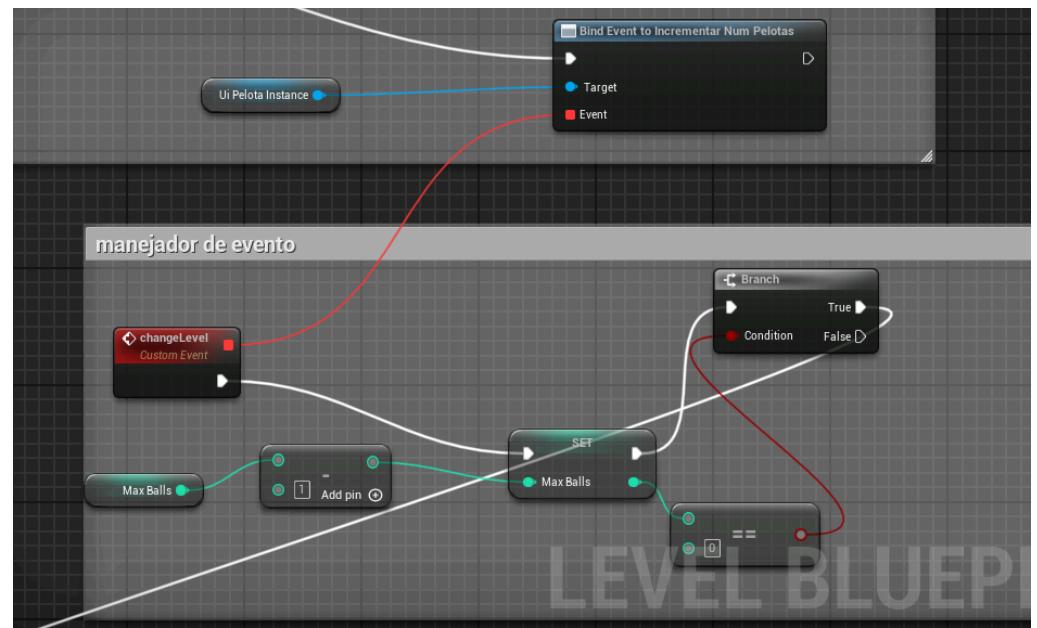
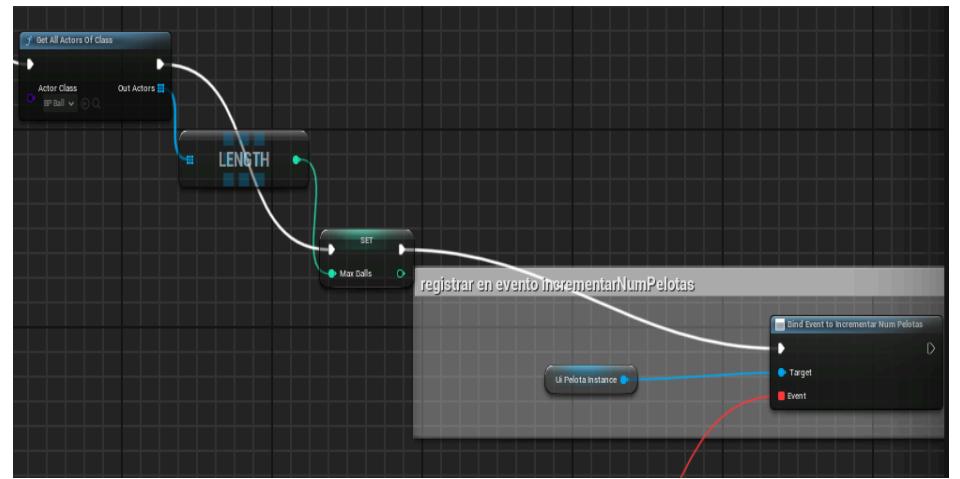
- **Creación del Widget:**
 - Se utiliza el nodo Create Widget para crear una instancia del widget BP_WidgetInterface.
- **Añadir al Viewport:**
 - El widget se agrega al Viewport usando el nodo Add to Viewport.
- **Contenido del Widget:**
 - El widget muestra la barra de vida del jugador y el número de pelotas colocadas en el terreno. Estas dos métricas se posicionan en la parte superior izquierda de la pantalla para facilitar su visualización.



- **Cuarto Mecanismo: Contador de Pelotas y Transición al Siguiente Nivel**

En el Level Blueprint de ThirdPersonMap, se implementa un sistema para manejar la destrucción de pelotas y el avance al siguiente nivel:

- **Obtener Todas las Pelotas:**
 - Se utiliza el nodo Get All Actors of Class para obtener todas las instancias de BP_Ball en el nivel.
- **Almacenar la Cantidad de Pelotas:**
 - La longitud del array resultante se guarda en la variable Max Balls, representando el número total de pelotas en el nivel.
- **Evento de Incrementar Número de Pelotas:**
 - Se hace un Bind con el Event Dispatcher llamado Incrementar Número de Pelotas para actualizar el contador en el widget.
- **Destrucción de Pelotas:**
 - Cada vez que el evento Incrementar Número de Pelotas se llama (cuando el jugador coloca o destruye una pelota), se resta uno de la variable Max Balls.
- **Verificación de Finalización:**
 - Si Max Balls llega a 0, significa que todas las pelotas han sido colocadas y se puede avanzar al siguiente nivel. Si no, el jugador sigue en el nivel actual.



- **Quinto Mecanismo: Actualización de Variables en Game Instance**

Antes de pasar al siguiente nivel, se realiza una verificación y actualización de las variables en la Game Instance:

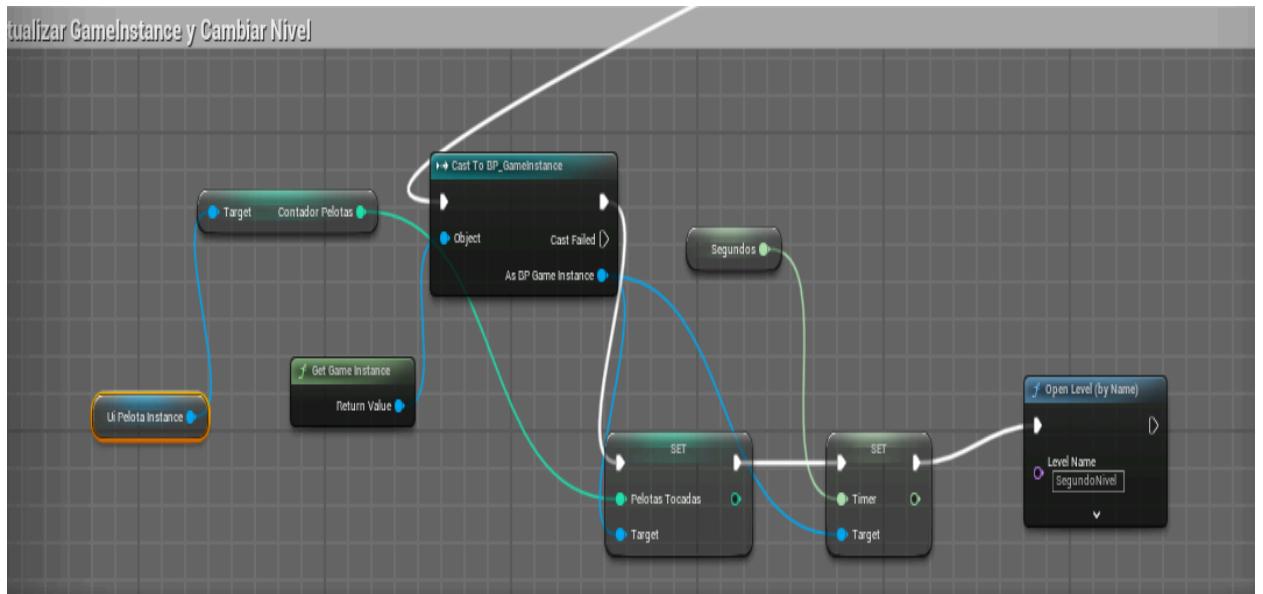
- **Obtención de Game Instance:**

- Se usa el nodo Get Game Instance y se realiza un Cast a BP_GameInstance.

- **Actualización de Variables:**

- Se almacenan los valores en las variables Pelotas Tocadas y Timer de BP_GameInstance:

- **Pelotas Tocadas:** Se actualiza con el valor de la variable Contador Pelotas del widget BP_WidgetInterface.
- **Timer:** Se guarda el valor de la variable Segundos (el tiempo transcurrido en el nivel).



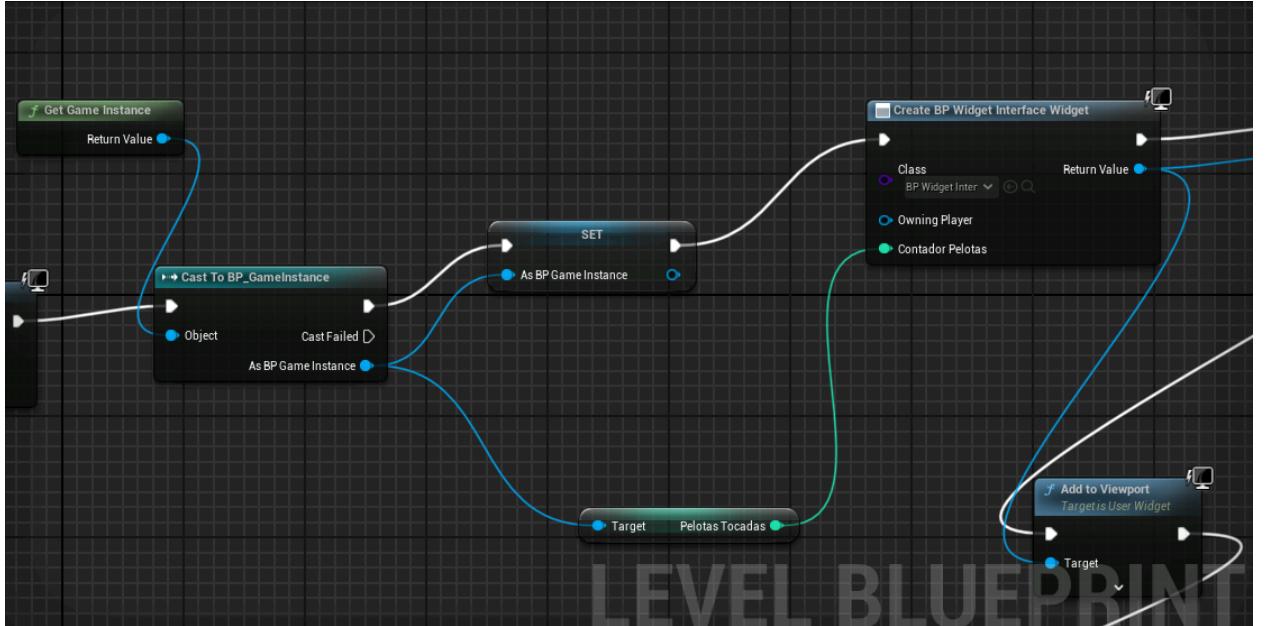
C. SegundoNivel Level

- **Primer Mecanismo :Actualizacion de Widget Interface**
En el Level Blueprint del segundo nivel, se sigue una lógica similar a la del primer nivel, pero con una actualización del contador de pelotas basado en el progreso del jugador
 - **Obtención de Game Instance:**
 - Se obtiene la Game Instance utilizando Get Game Instance y se realiza un Cast a BP_GameInstance.
 - **Actualizar Contador de Pelotas:**
 - Se toma el valor de la variable Pelotas Tocadas de la Game Instance y se actualiza la variable Contador Pelotas en el widget BP_WidgetInterface.

- **Crear y Añadir Widget:**

- Se crea una nueva instancia del widget BP_WidgetInterface con el valor actualizado de pelotas tocadas y se añade al Viewport utilizando Add to Viewport.

Este mecanismo asegura que el jugador vea en la interfaz el número de pelotas tocadas desde el nivel anterior y continúe con el mismo sistema de visualización.



- **Segundo Mecanismo: Bind de Eventos y Transición al Cambio de Nivel**

En el Level Blueprint del segundo nivel, se gestionan los eventos que pueden cambiar el nivel dependiendo de las acciones del jugador:

- **Obtener Actores de Clase:**

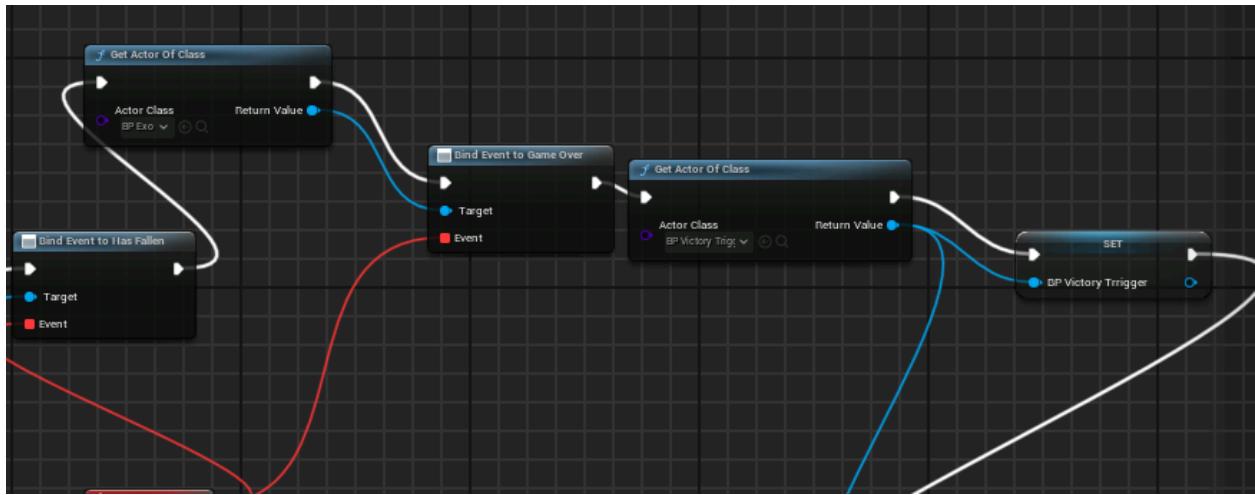
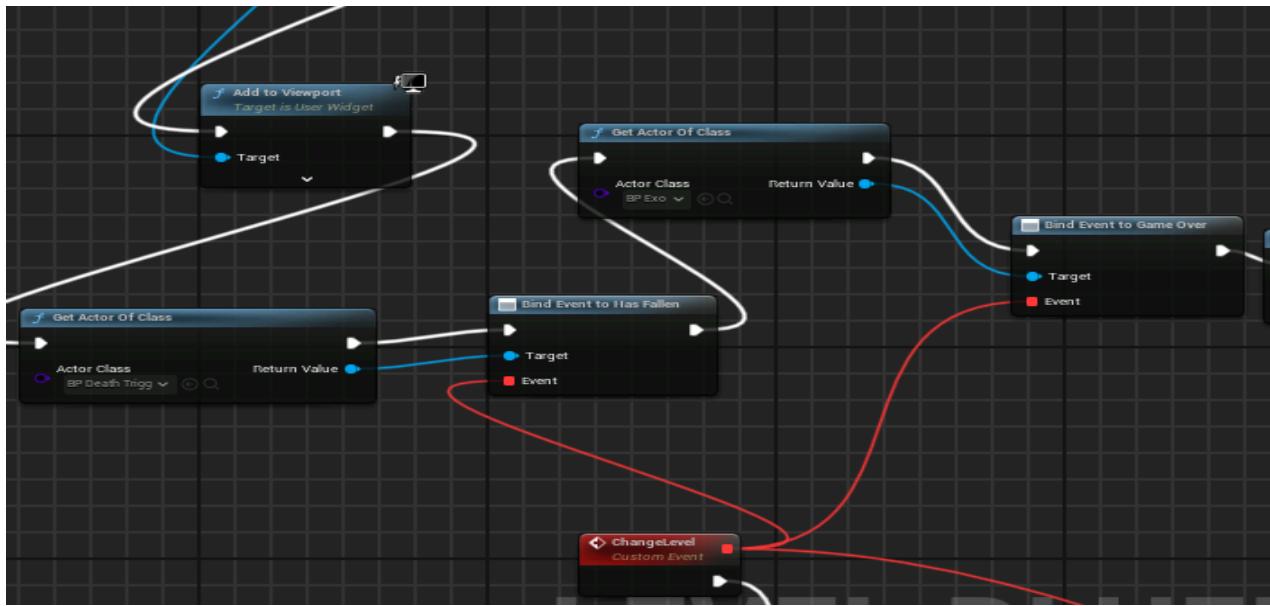
- Se usan los nodos Get Actor of Class para obtener las instancias de los actores BP_VictoryTrigger, BP_DeathTrigger, y BP_Exo (el jugador).

- **Bind a los Event Dispatchers:**

Se realiza un Bind a los siguientes Event Dispatchers:

- **Has Fallen (de BP_DeathTrigger).**
- **Game Over Win (de BP_VictoryTrigger).**
- **Game Over (de BP_Exo).**

Los tres eventos se asocian al mismo manejador, que ejecuta un evento llamado ChangeLevel.

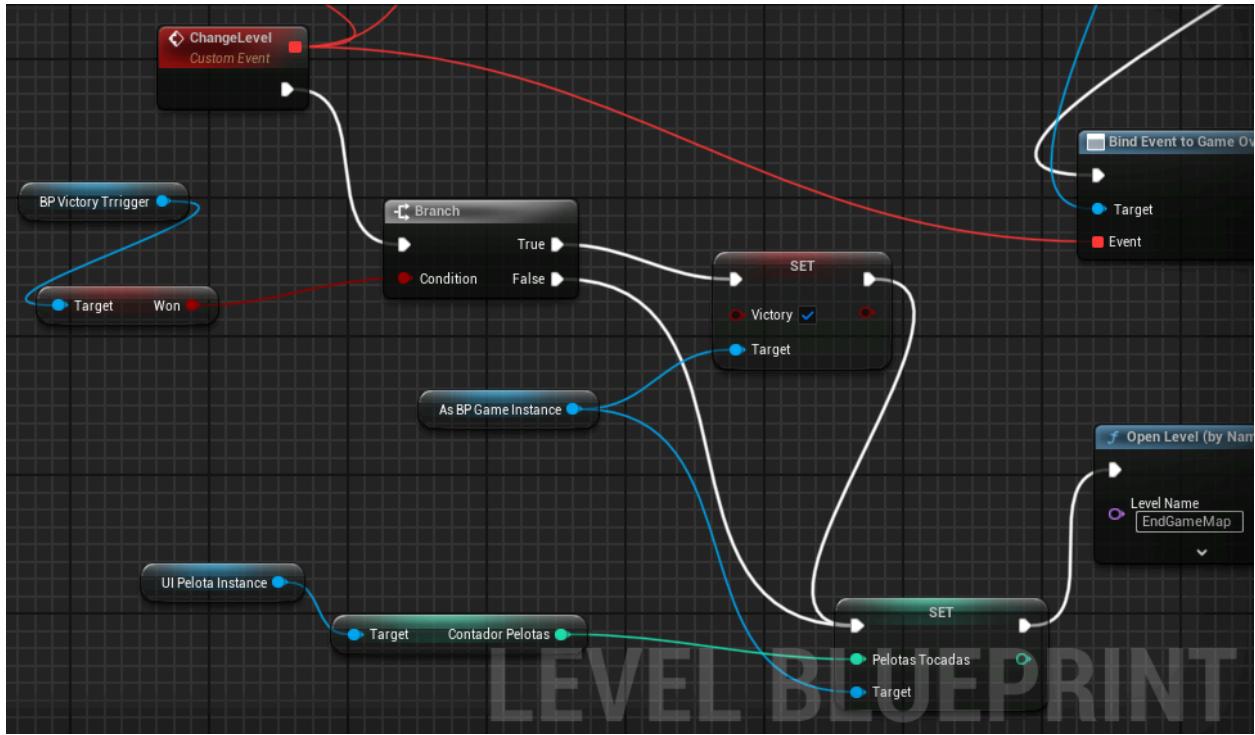


- **Evento ChangeLevel: Transición de Nivel y Actualización de Game Instance**

El evento ChangeLevel gestiona la transición entre niveles y actualiza las variables correspondientes en la Game Instance:

- **Verificación de Victoria:**
 - Se toma la variable **Won** de BP_VictoryTrigger para determinar si el jugador ha ganado.
 - Si Won es true, se actualiza la variable Victory en BP_GameInstance a true.
 - Si Won es false, no se actualiza la variable Victory en Game Instance.
- **Actualización de Pelotas Tocadas:**
 - Se actualiza la variable Pelotas Tocadas de la Game Instance con el valor correspondiente.
- **Transición a EndGame Map:**
 - Despues de la actualización de las variables, se carga el mapa EndGameMap, donde finaliza el juego.

Este evento asegura que las condiciones de victoria o derrota se gestionen correctamente y que las estadísticas del jugador, como las pelotas tocadas y la victoria, se actualicen antes de proceder al final del juego.

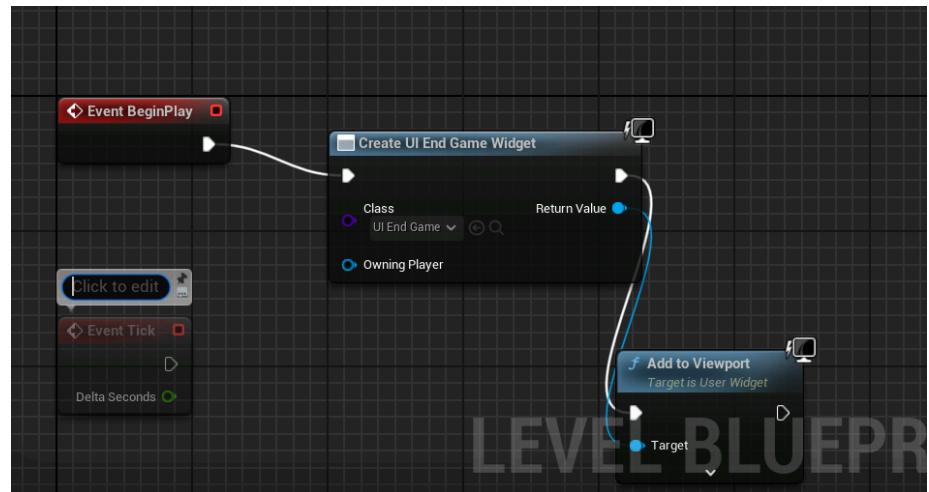


D. End Game Map Level

El **EndGame Map** sigue una lógica similar al **Menu Map**:

- **Creación del Widget:**
 - Se crea un widget de clase **UI_EndGame**.
- **Añadir al Viewport:**
 - Se utiliza **Add to Viewport** para mostrar el widget en pantalla.

Este nivel simplemente muestra la interfaz de finalización del juego al jugador.



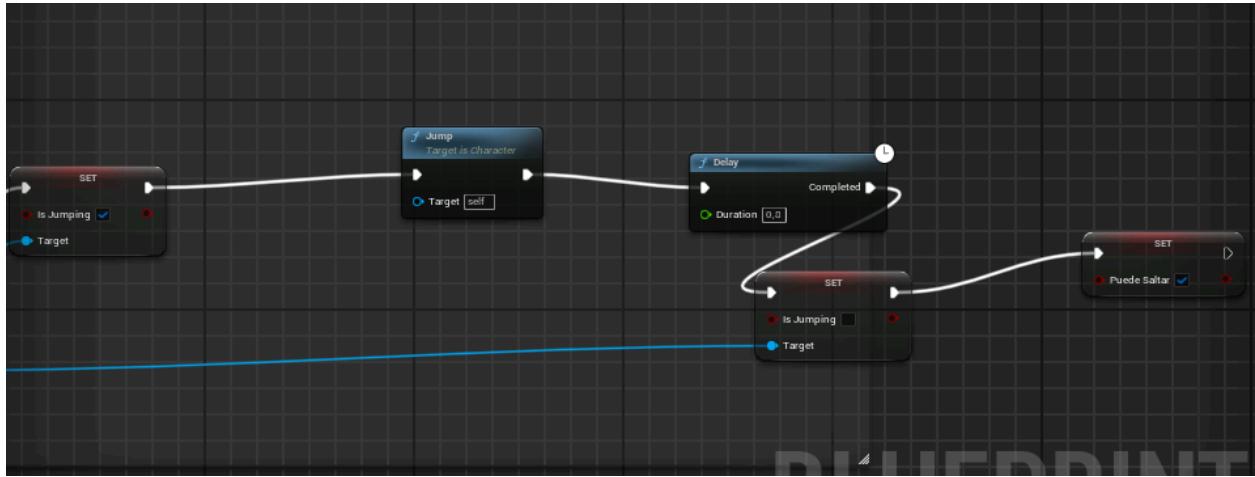
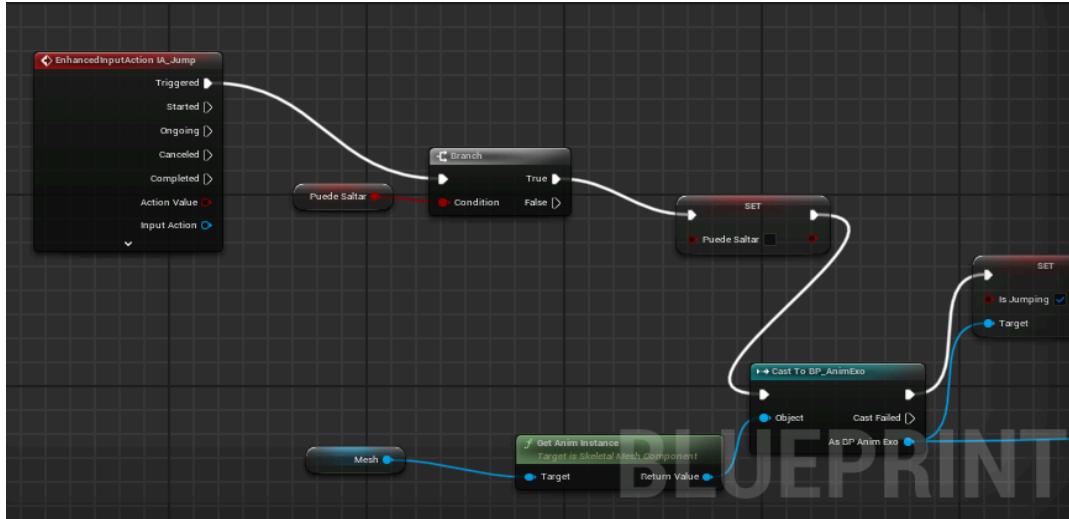
3.2.2. Characters Blueprints

A. BP_Exo

Configuración Inicial:

- El personaje principal, **Exo**, es un modelo importado desde **Mixamo**.
- Se duplicó el **Blueprint ThirdPersonCharacter** por defecto de Unreal y se reemplazó el **Skeletal Mesh** con el modelo de Exo para aprovechar las configuraciones predeterminadas de movimiento y otros sistemas.
- **Primer Mecanismo: Animación de Salto:**
 - **Evento EnhancedInput_IAJump:**
 - Al presionar la tecla asignada al salto, se llama a la función **Jump** (predeterminada).
 - Además, se realiza lo siguiente:

- Obtener la instancia de animación del personaje usando **Get Anim Instance**.
- Realizar un **Cast a BP_AnimExo**.
- Establecer la variable **isJumping** como **true** para activar la animación de salto en el **Animation Blueprint**.



- **Segundo Mecanismo: Movimiento de Agacharse (Crouch Movement)**

Activación por Tecla 'C':

- **Evento Key 'C' Pressed:**

- Detecta si el jugador presiona la tecla para alternar el estado de agacharse.

- **Lógica de Alternancia:**

- Obtener la Anim Instance del personaje y realizar un Cast a BP_AnimExo.

- Verificar la variable isCrouching del Animation Blueprint:

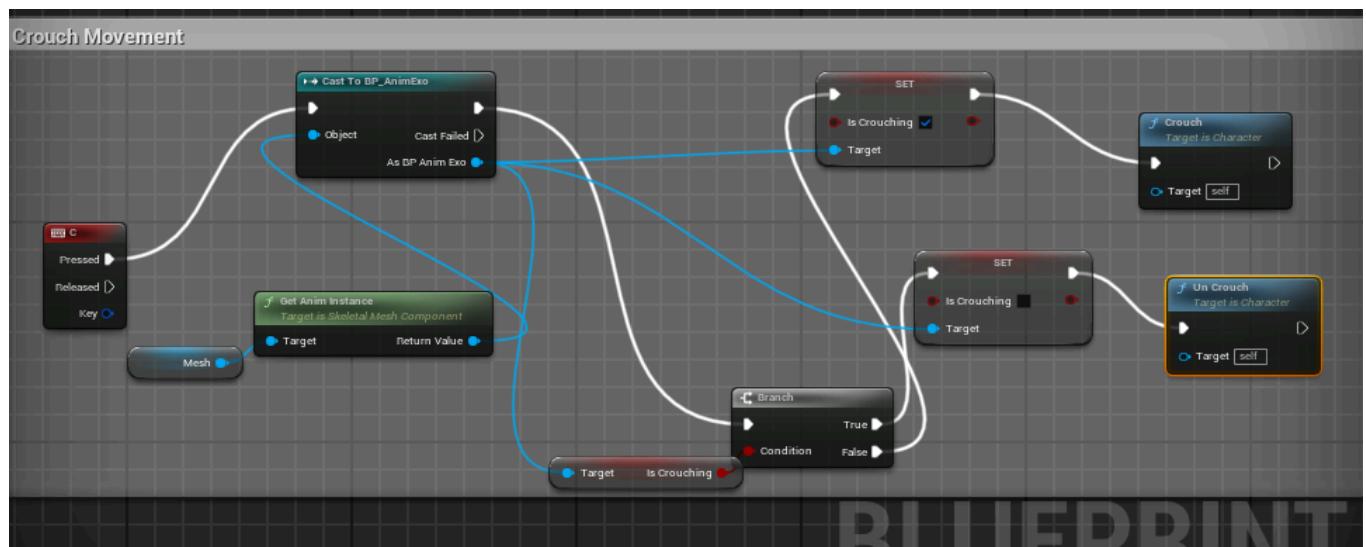
- **Si isCrouching es True:**

- El personaje está agachado, por lo que se cambia la variable a false para detener el movimiento de agacharse.
- Llamar a la función UnCrouch para detener el movimiento de agacharse.

- **Si isCrouching es False:**

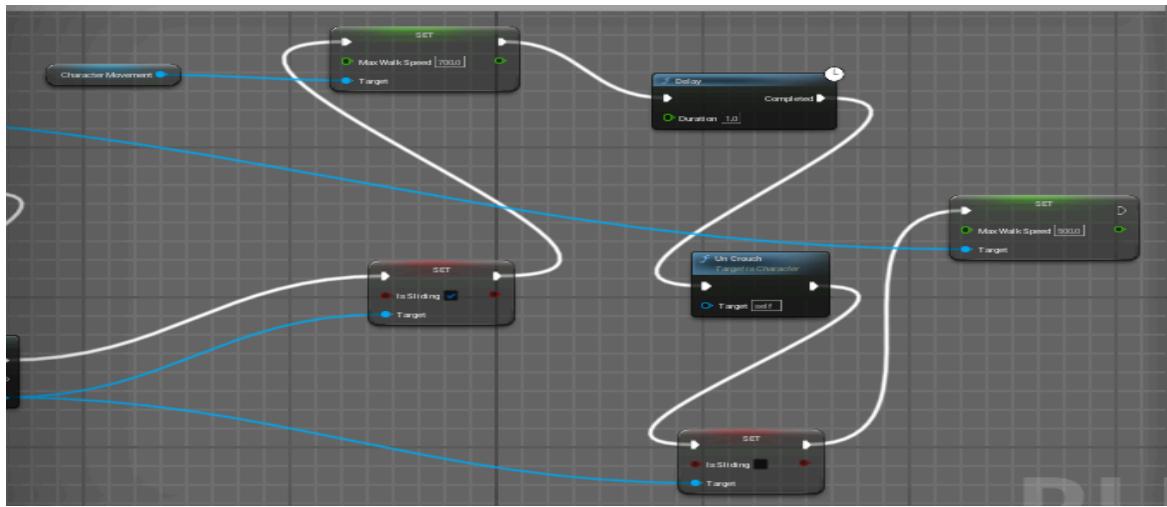
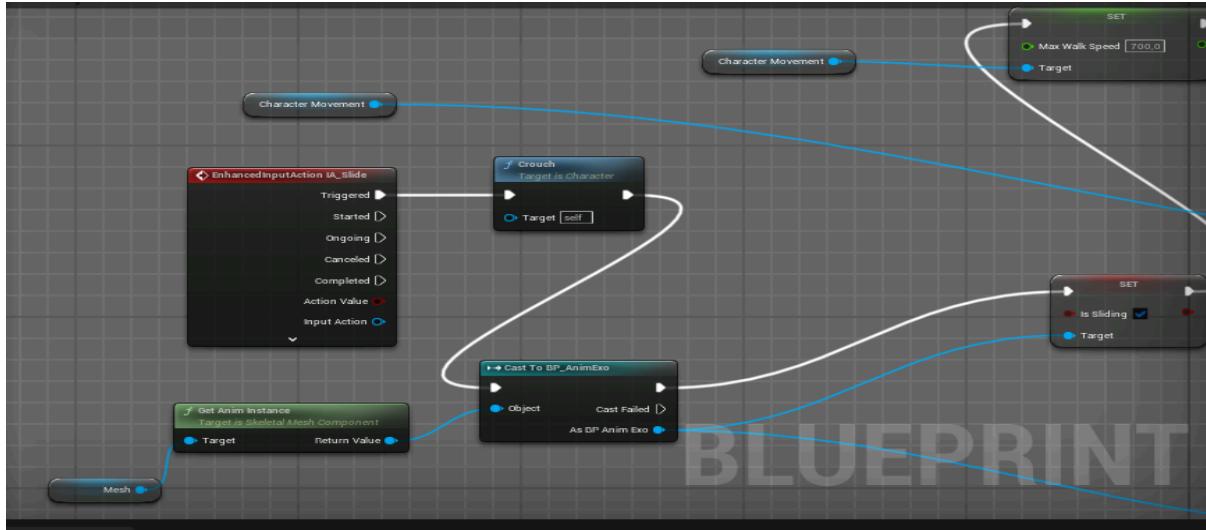
- El personaje no está agachado, se establece la variable en true para activar el movimiento de agacharse.
- Llamar a la función Crouch para activar el movimiento de agacharse.

- Este mecanismo permite alternar entre los estados de estar agachado o de pie y está preparado para posibles usos futuros en el juego.



- **Tercer Mecanismo: Movimiento Deslizante (Sliding Movement)**
- **Activación con Input IA_Slide (Letra G):**
 - **Evento EnhancedInput IA_Slide (letra G):**
 - Detecta cuando el jugador activa el movimiento de deslizamiento.
 - **Lógica del Deslizamiento:**
 - Obtener la Anim Instance y realizar un Cast a BP_AnimExo.
 - Llamar a la función Crouch para iniciar el deslizamiento.
 - Establecer la variable isSliding en true para activar la animación correspondiente.
 - Cambiar la velocidad máxima de caminar (Max Walk Speed) a 700 usando el nodo Set Max Walk Speed.
 - **Finalización del Deslizamiento:**
 - Tras un retraso de 1 segundo (usando Delay):
 - Restaurar la velocidad máxima de caminar a su valor inicial (500).
 - Establecer isSliding en false.
 - Llamar a la función UnCrouch para finalizar el deslizamiento.

Este mecanismo permite una transición fluida entre el movimiento normal y el deslizamiento, proporcionando un dinamismo adicional al personaje.

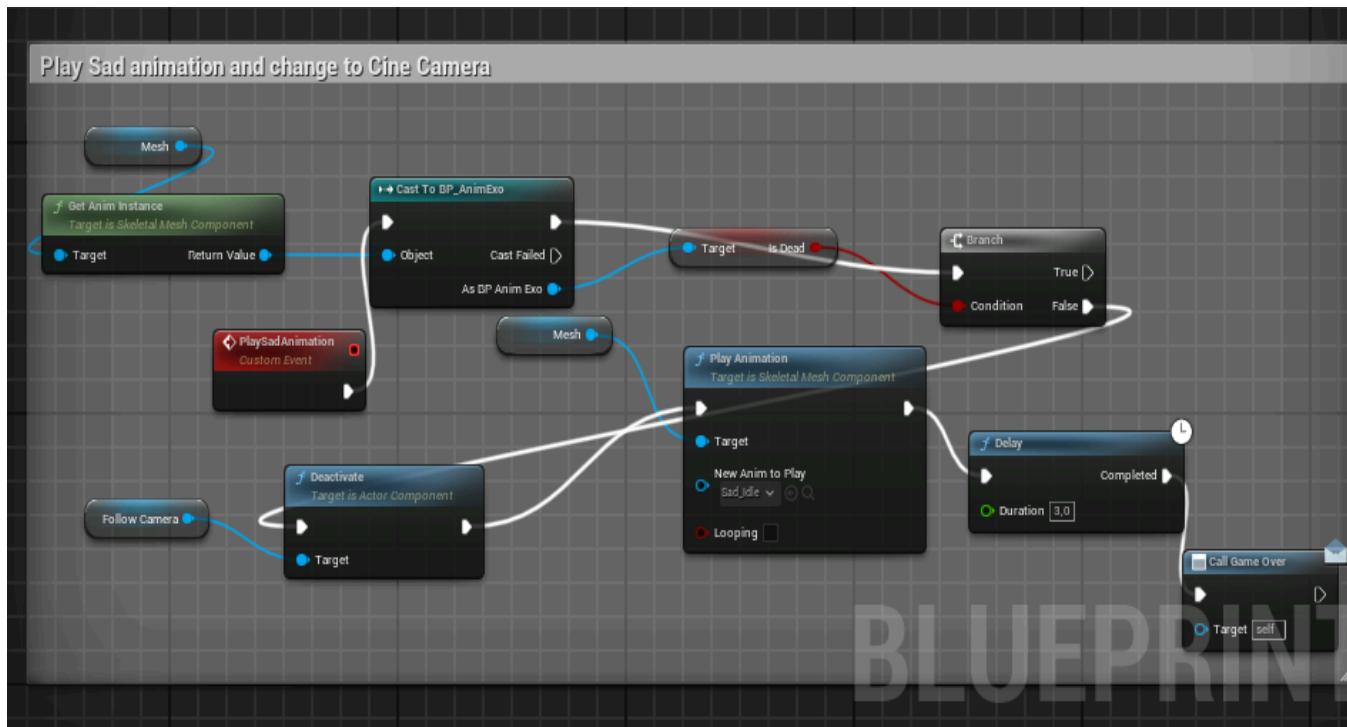


- **Cuarto Mecanismo: Reproducción de Animación Triste (Playing Sad Animation)**

- **Evento Play Sad Animation:**

- Se obtiene la Anim Instance y se realiza un Cast a BP_AnimExo.
 - Comprobación de Estado:
 - Verifica si la variable isDead es false (para evitar reproducir la animación si el personaje está muerto).
 - **Reproducción de Animación:**
 - Si isDead es false:
 - Llama a la función Play Animation con la animación Sad_Idle.

- Desactiva la Follow Camera para ceder el control a la Cine Camera, proporcionando una vista cinematográfica del personaje.
 - Espera 3 segundos usando un Delay.
 - Llama al Event Dispatcher Game Over, indicando el final del juego.
- **Contexto de Invocación:**
- Este evento es activado por el Controller IA del enemigo cuando este alcanza al personaje principal, simulando que ha sido atrapado.



- **Quinto Mecanismo: Recepción de Daño y Actualización de Salud (Take Damage)**

- **Variables Clave:**

- Health (salud actual) y Max Health (salud máxima), ambas de tipo float.

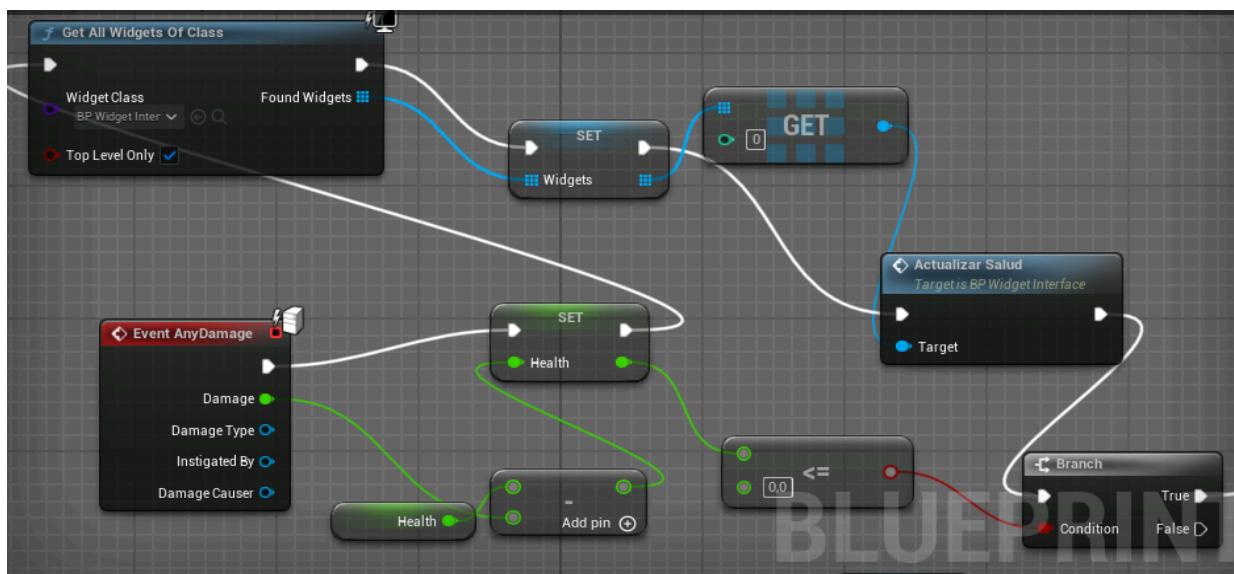
- **Evento Any Damage:**

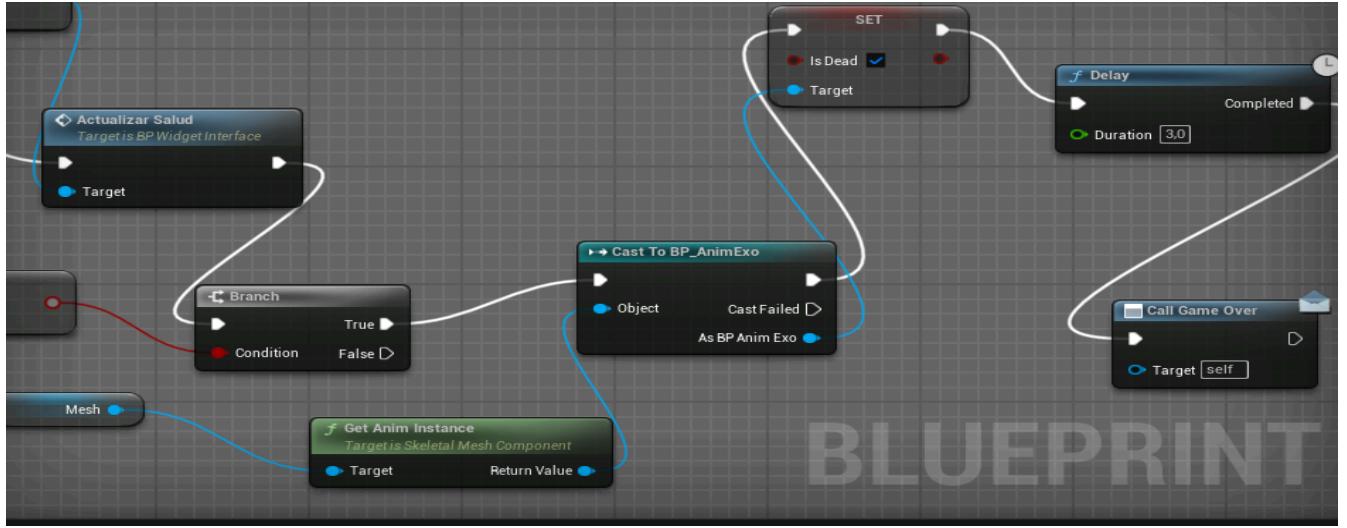
- Resta el valor de Damage de la variable Health.
- Llama a la función Actualizar Salud en el BP_WidgetInterface para reflejar la nueva barra de progreso:
- Se utiliza Get All Widgets of Class para obtener el widget correspondiente y se actualiza con la nueva información.

- **Comprobación de Salud:**

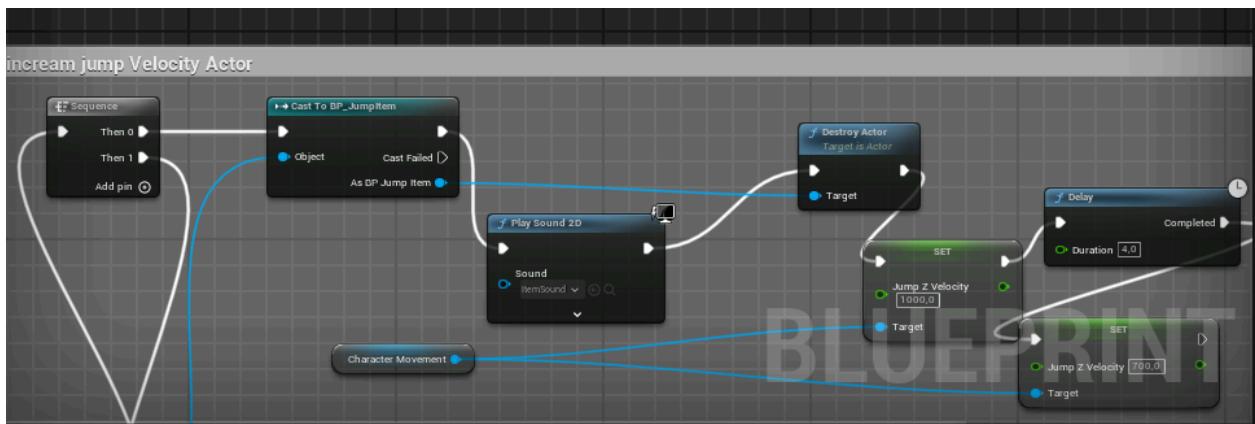
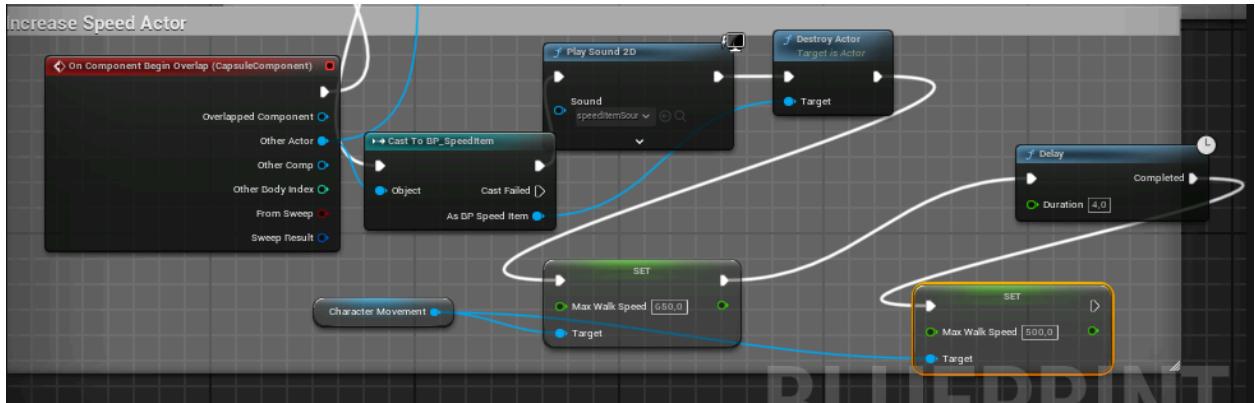
- **Si Health == 0:**

- Cambia la variable isDead del BP_AnimExo a true (activando las animaciones relacionadas con la muerte).
- Inicia un Delay de 3 segundos antes de invocar el Event Dispatcher Game Over, que marca el fin del juego.





- **Sexto Mecanismo: Interacción con Ítems**
 - **Evento: On Component Begin Overlap**
Se conecta a un nodo Sequence para manejar diferentes tipos de ítems.
 - **Caso: BP_JumpItem**
 - Reproduce un sonido específico.
 - Destruye el actor correspondiente (ítem recogido).
 - Ajusta la variable Jump Z Velocity a 1000 (incrementando la altura del salto).
 - Utiliza un Delay de 4 segundos antes de restablecer el valor inicial a 700.
 - **Caso: BP_SpeedItem**
 - Reproduce un sonido específico.
 - Destruye el actor correspondiente (ítem recogido).
 - Ajusta la variable Max Walk Speed a 650 (incrementando la velocidad de movimiento).
 - Utiliza un Delay de 4 segundos antes de restablecer el valor inicial a 500.



B. BP_Enemigo

Es un personaje importado desde Mixamo, basado en una duplicación del blueprint BP_ThirdPerson, al cual se le han realizado modificaciones:

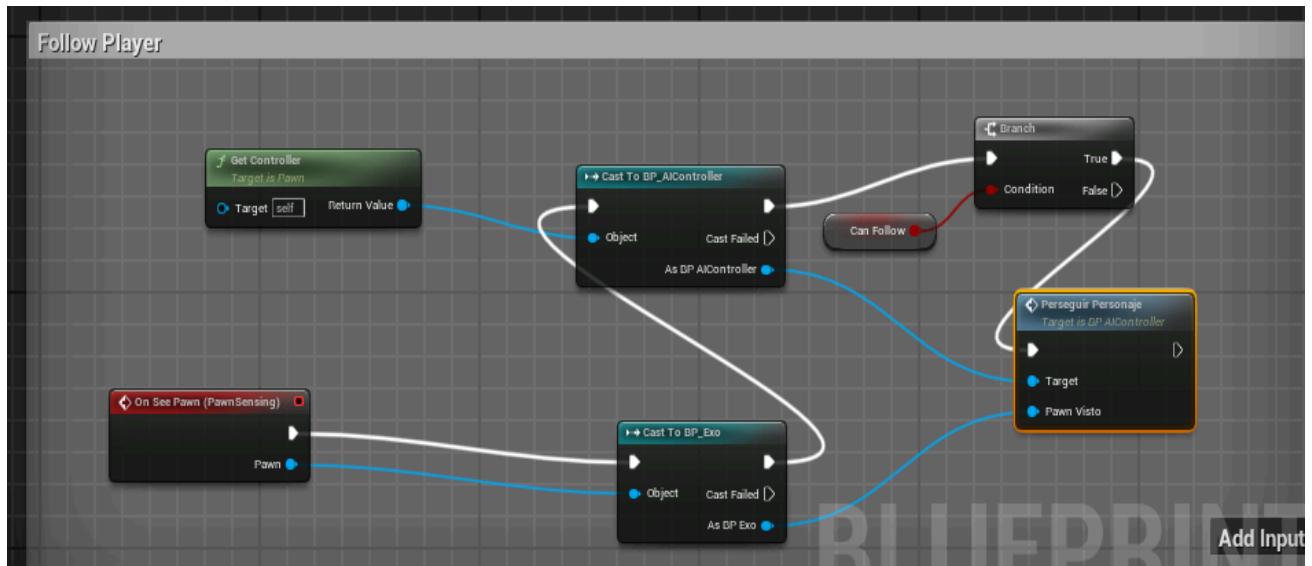
1. Se ha añadido un componente Pawn Sensing para detectar al jugador.
2. Se eliminó la Follow Camera para enfocarse en un comportamiento de IA.
3. Se incorporó la variable CanFollow (booleana) para controlar si el enemigo puede perseguir al personaje.

- **Primer Mecanismo: On See Pawn**

- **Detección de Personaje**

- Utiliza el componente Pawn Sensing para detectar personajes visibles.

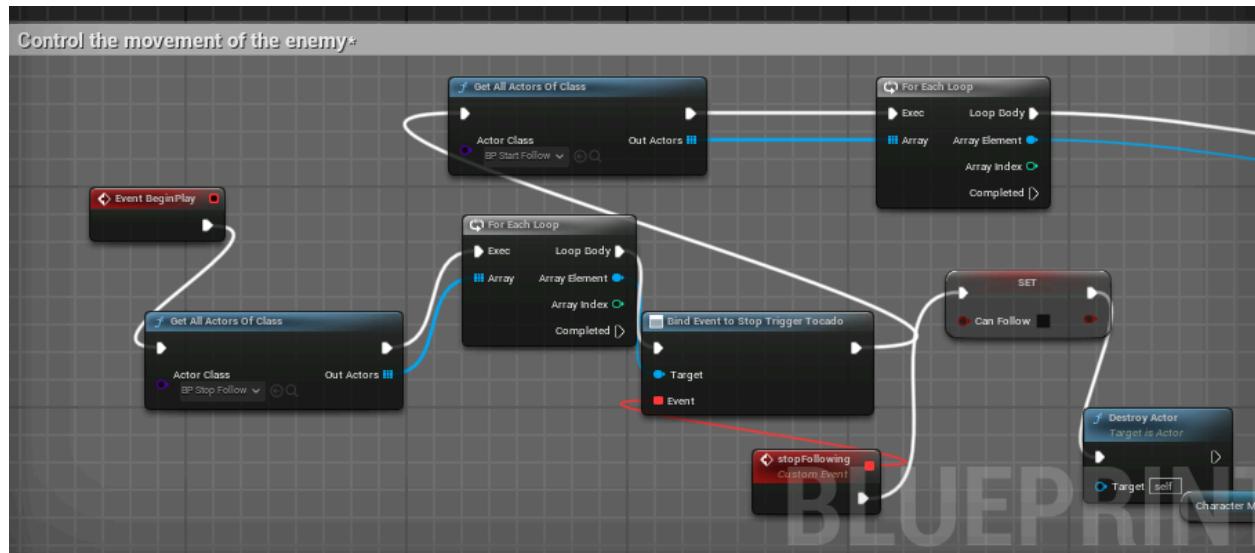
- Cuando se activa el evento On See Pawn, se realiza un cast del actor detectado al blueprint BP_Exo (personaje principal).
- **Verificación y Acción**
 - Si el actor detectado es el personaje principal:
 - Se obtiene el controller del enemigo usando Get Controller.
 - Se realiza un cast a BP_AIController .
 - Se comprueba si la variable CanFollow es True.
 - Si CanFollow = True, se llama a la función Perseguir Personaje del BP_ControllerIA para iniciar la persecución.



- **Segundo Mecanismo: Gestión de Triggers (BP_Enemigo)**
Después del evento Begin Play, el enemigo realiza los siguientes pasos:

- **Obtención de Actores**
Utiliza Get All Actors of Class para obtener todos los actores de las clases:
 - **BP_StopFollowingTrigger**.
 - **BP_StartFollowingTrigger**.
- **Manejo de BP_StopFollowingTrigger**
 - Se realiza un For Each Loop para recorrer todos los actores encontrados.
 - Para cada actor, se realiza un bind con el evento Stop Trigger Tocado.

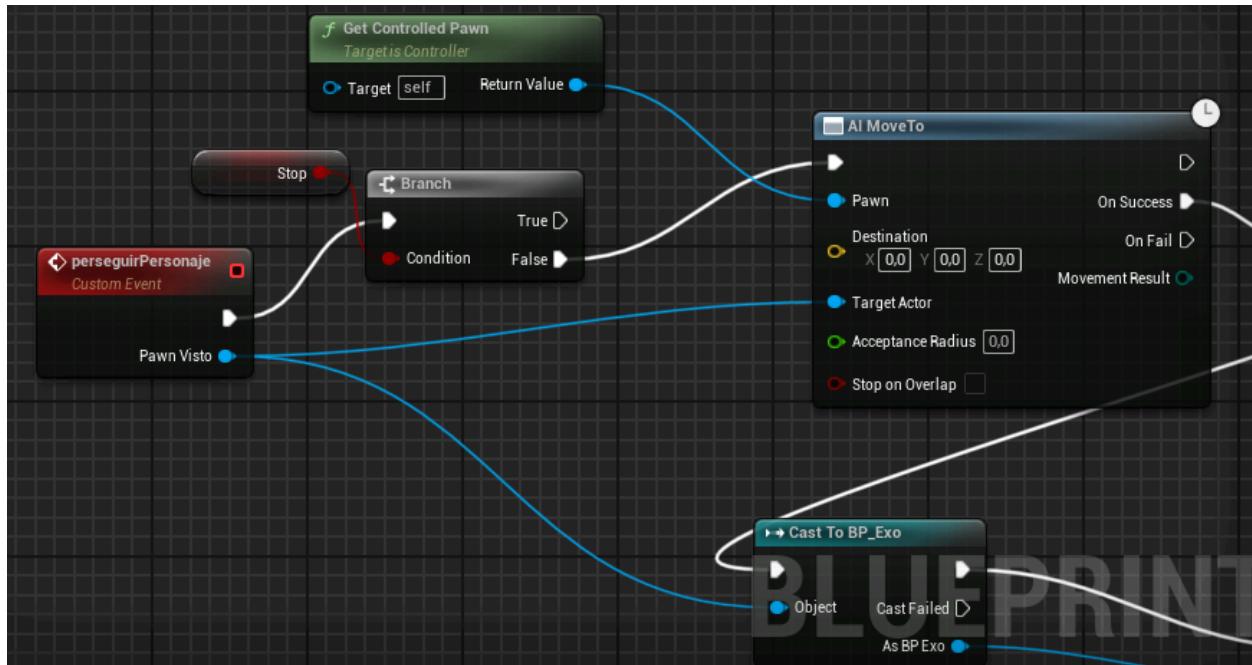
- Este bind está asociado al manejador StopFollowing, que realiza las siguientes acciones:
- Establece la variable CanFollow del enemigo en False.
- Destruye al actor del enemigo con Destroy Actor.
- **Manejo de BP_StartFollowingTrigger**
 - Similar al anterior, se utiliza un For Each Loop para iterar los actores de esta clase.
 - Se realiza un bind con el evento Trigger Tocado.
 - Este bind está vinculado al manejador CanFollowPlayer, que ejecuta las siguientes acciones:
 - Incrementa la velocidad máxima de movimiento (Max Walk Speed) del enemigo en 50 unidades.
 - Actualiza la variable CanFollow del enemigo a True.

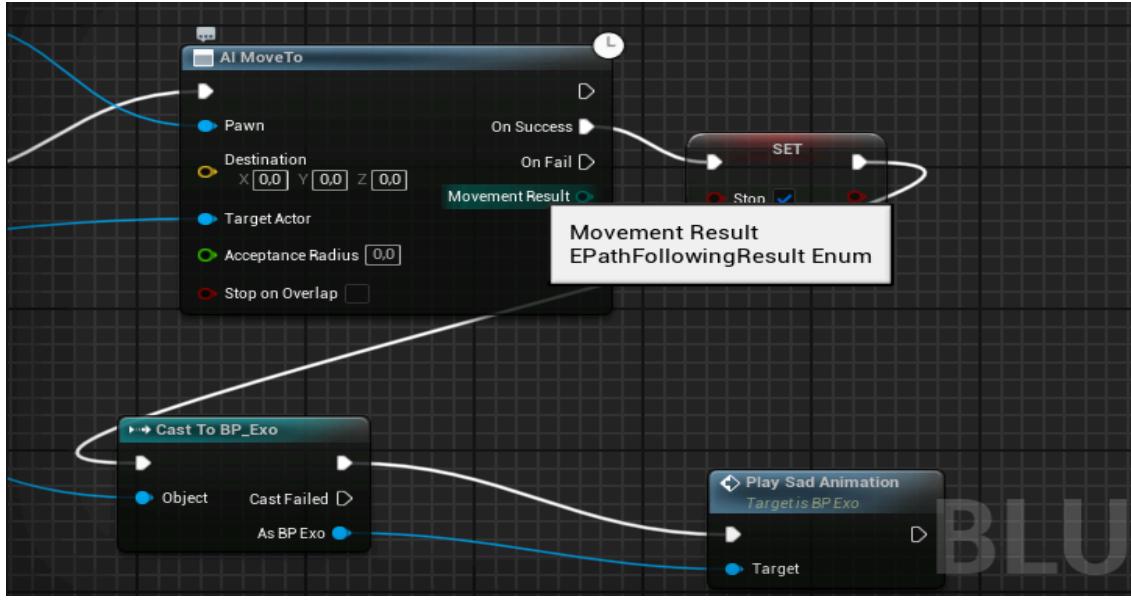


3.2.3. Controller Blueprints

Este controlador se encarga de gestionar la lógica de persecución del enemigo y tiene un evento clave llamado Perseguir Personaje.

- **Evento Perseguir Personaje**
 - **Input del Evento**
 - Recibe como parámetro el Pawn Visto (es decir, el actor detectado).
 - Este parámetro se asigna al campo Target Actor de la función AI Move To.
 - **Lógica del Evento**
 - Comprueba la variable booleana Stop:
 - Si Stop no es True, continúa el flujo; de lo contrario, no realiza ninguna acción.
 - **Configuración del Movimiento**
 - Llama a Get Controlled Pawn, cuyo valor returned se pasa como parámetro Pawn a la función AI Move To.
 - **Alcanzar al Personaje Principal**
 - Si el resultado de AI Move To es Success (es decir, el enemigo logra llegar al personaje principal):
 - La variable Stop se establece en True.
 - Se llama al evento Play Sad Animation del personaje principal (BP_Exo) para ejecutar la animación correspondiente al ser atrapado.





3.2.4. Game Instance Blueprint

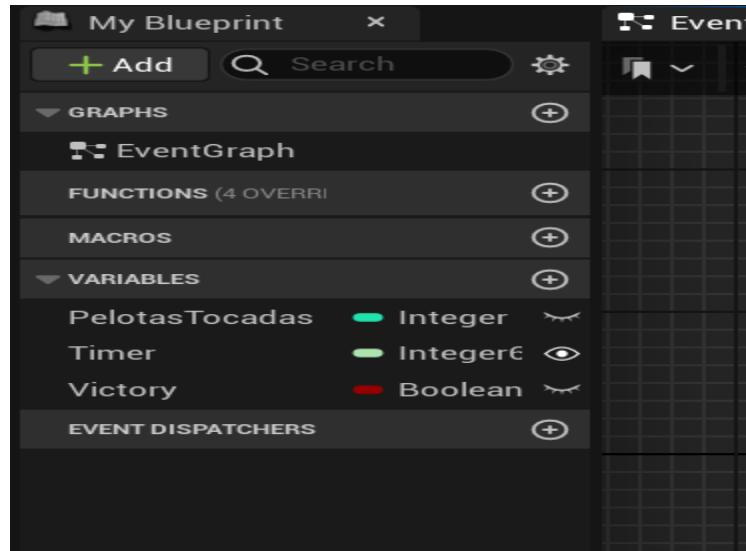
El BP_GameInstance actúa como el contenedor central de datos que persisten entre niveles, permitiendo mantener información clave durante la progresión del juego.

- **Variables**

- Time
 - Tipo: Integer
 - Propósito: Almacenar el tiempo total acumulado que el jugador ha utilizado en los niveles.
- Pelotas Tocadas
 - Tipo: Integer
 - Propósito: Almacenar el número total de pelotas que el jugador ha recogido o tocado en los niveles.
- Victory
 - Tipo: Boolean
 - Propósito: Indicar si el jugador ha ganado el juego (por ejemplo, al completar todos los objetivos o alcanzar el final con éxito).

Uso en el Juego:

Se accede a estas variables desde diferentes niveles y sistemas mediante Get Game Instance y un Cast a BP_GameInstance. Estas variables son actualizadas en eventos clave, como al pasar de un nivel a otro o al completar ciertas acciones específicas.



3.2.5. Animation Blueprints

A. ABP_Exo

El ABP_Exo es el Blueprint de animación específico del personaje Exo. Su configuración aprovecha algunas características predefinidas del ABP_Manny (la clase de animación del Mannequin) y añade modificaciones específicas para personalizar el comportamiento.

- **Configuración del Event Graph**

- **EventGraph**

- Todo el contenido en esta sección es una copia directa del ABP_Manny.
 - Maneja lógica como actualizaciones de variables relacionadas con el movimiento, velocidad, y otras propiedades generales.

- **Configuración del Anim Graph**

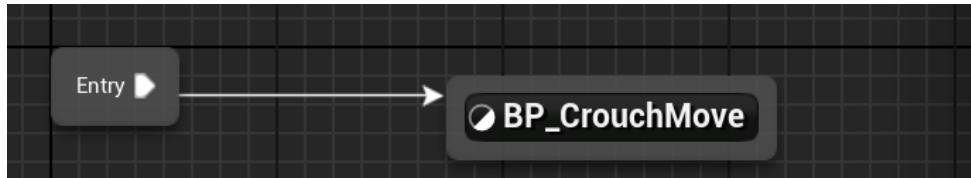
El Anim Graph contiene una configuración personalizada para manejar las animaciones del personaje Exo.

- **Blend Poses by Bool**

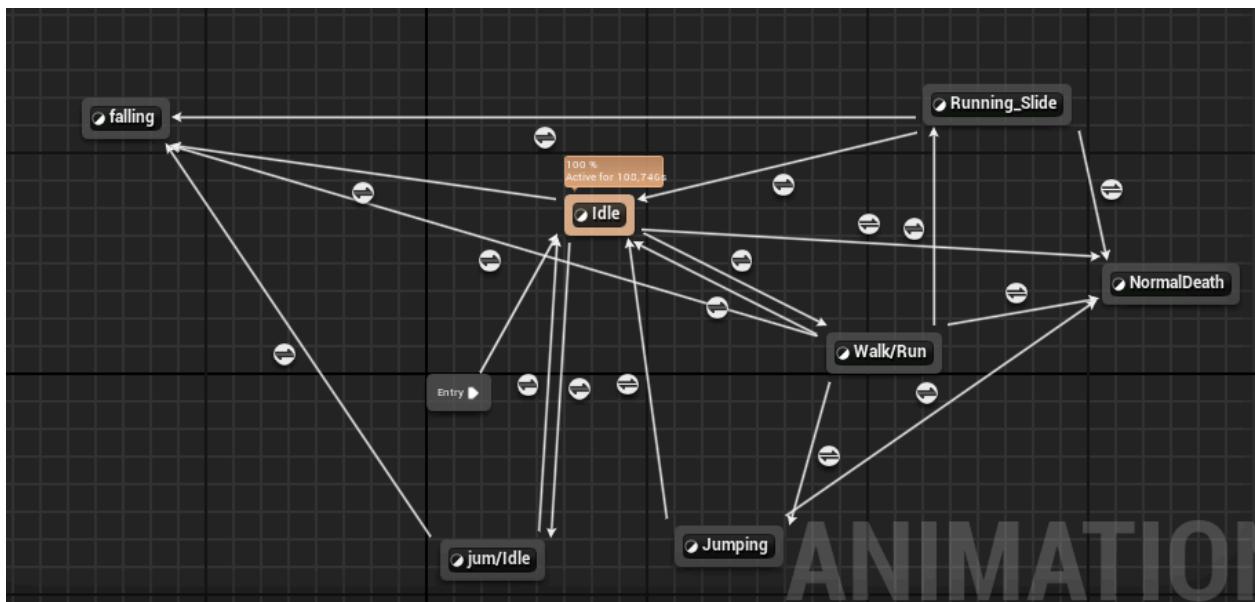
El nodo Blend Poses by Bool conecta dos State Machines principales:

- Crouch Movement: Maneja las animaciones específicas cuando el personaje está en modo agachado.
 - Movement: Maneja las animaciones estándar de movimiento del personaje.
 - Controlado por la variable isCrouching:

- Si isCrouching es true, se utiliza el Crouch Movement.
 - Si isCrouching es false, se activa el Movement.
- State Machines
 - Crouch Movement
 - Contiene animaciones específicas para el estado agachado, como caminar o permanecer estático en esa posición.



- Movement
 - Contiene animaciones estándar de movimiento, como deslizar, caminar, correr, saltar, o el estado inactivo.



En el BP_AnimExo, los dos State Machines (Crouch Movement y Movement) utilizan Blendspaces para gestionar las animaciones de caminata, carrera e inactividad. Estos Blendspaces permiten un control suave sobre las transiciones entre las animaciones de manera que el personaje se mueva de forma natural y fluida según su velocidad.

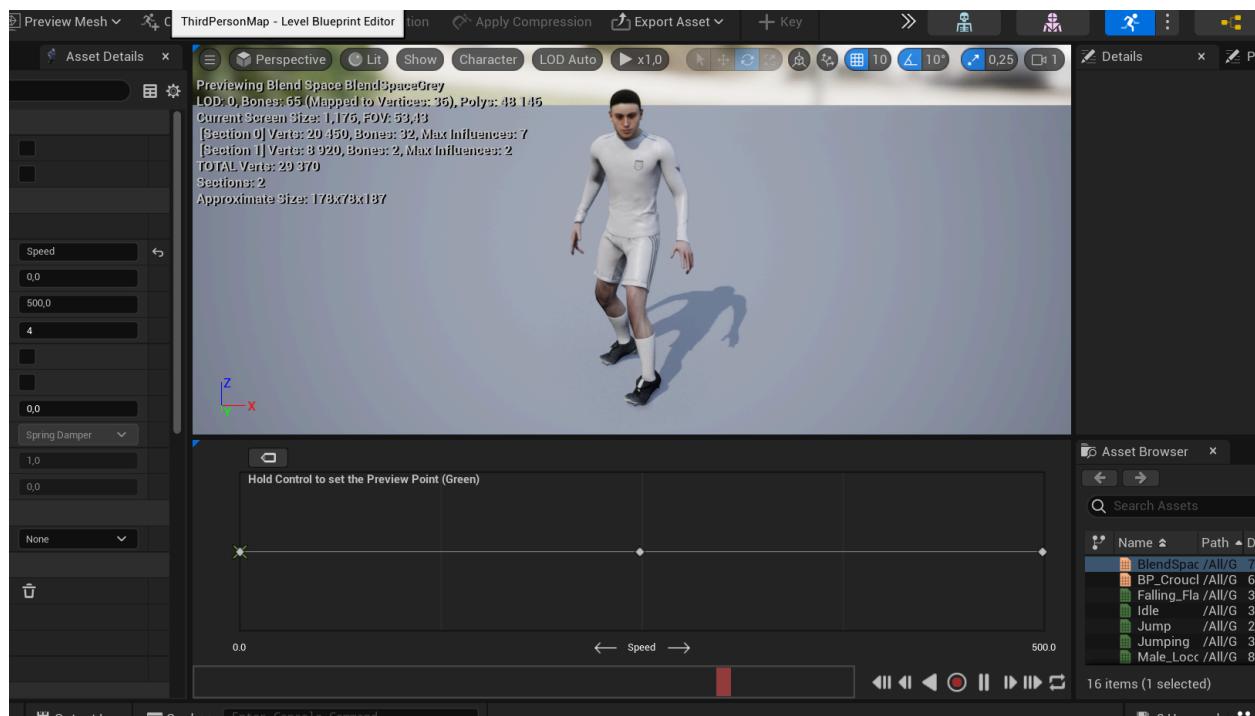
- **Estado Movement: Uso de Blendspace**

Blendspace: Utiliza un Blendspace para combinar las animaciones de caminar, correr e inactividad (idle).

- **Animaciones Controladas:**

- Idle: Cuando el personaje no se mueve (velocidad = 0).
- Walk: Movimiento de caminata cuando la velocidad es baja.
- Run: Movimiento de carrera cuando la velocidad es alta.

Se ajustan dinámicamente según la variable de velocidad del personaje (Speed), proporcionando transiciones suaves entre las diferentes animaciones.



- **Estado Crouch Movement: Uso de Blendspace**

Blendspace para Crouch Movement: De manera similar al estado de movimiento normal, el Blendspace se usa aquí para controlar las animaciones de agachado, permitiendo transiciones suaves entre:

- Idle Crouch: El personaje está agachado y no se mueve.
- Crouch Walk: Movimiento lento mientras el personaje está agachado.

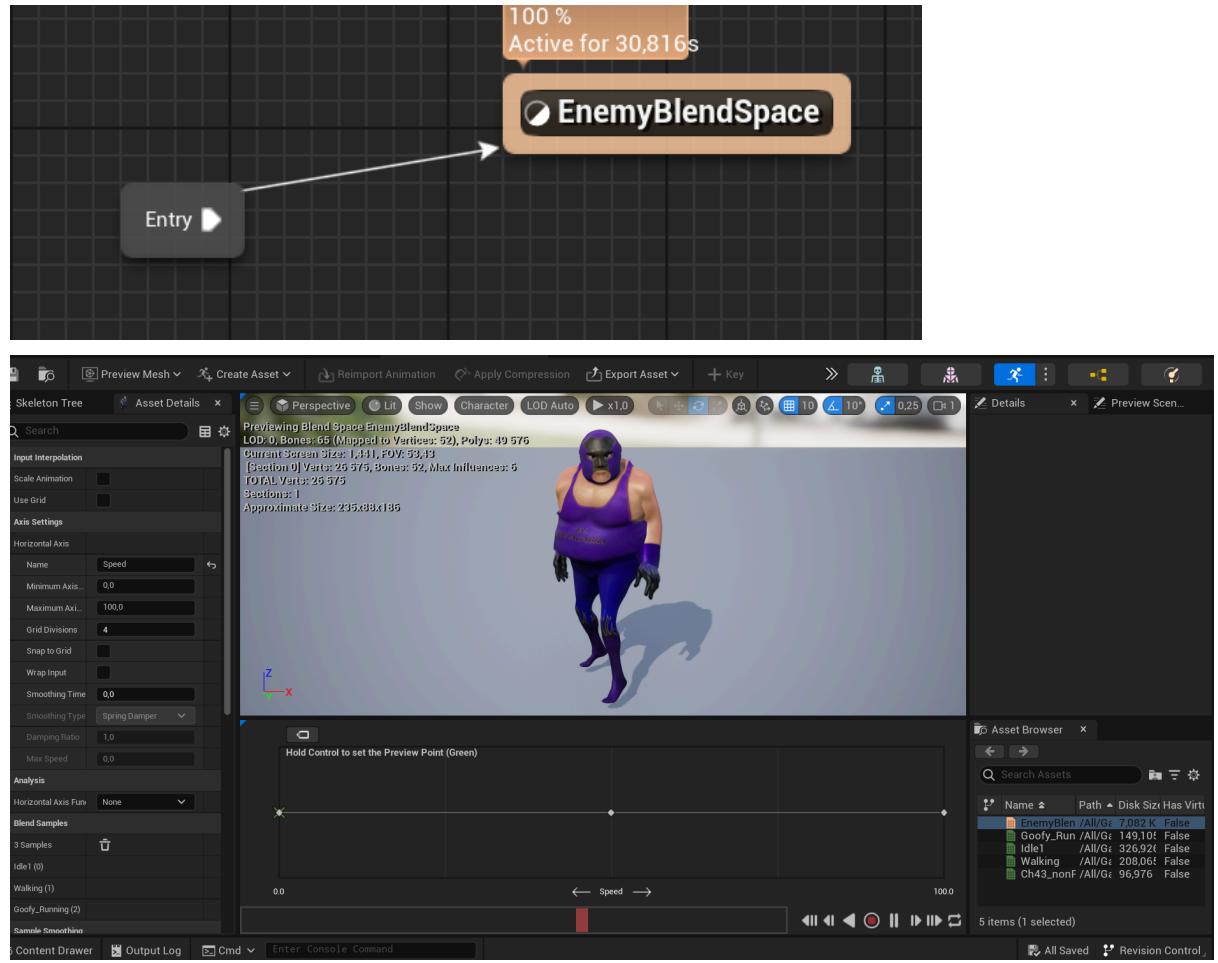


B. ABP_Enemy

El ABP_Enemy es un Blueprint de animación para el enemigo, y su estructura es bastante similar a la del ABP_Manny (que es el Blueprint de animación del maniquí). La principal diferencia radica en su Anim Graph, que está diseñado específicamente para gestionar las animaciones del enemigo.

- **Anim Graph: Estado y Transiciones**

- **State Machine Único:** El ABP_Enemy tiene un único State Machine que gestiona las animaciones de movimiento del enemigo, basado en un Blendspace.
- **Estado único:** Este único estado combina las animaciones de Idle, Walk y Run utilizando el Blendspace, permitiendo que el enemigo se mueva y se comporte de manera natural según la variable Speed.



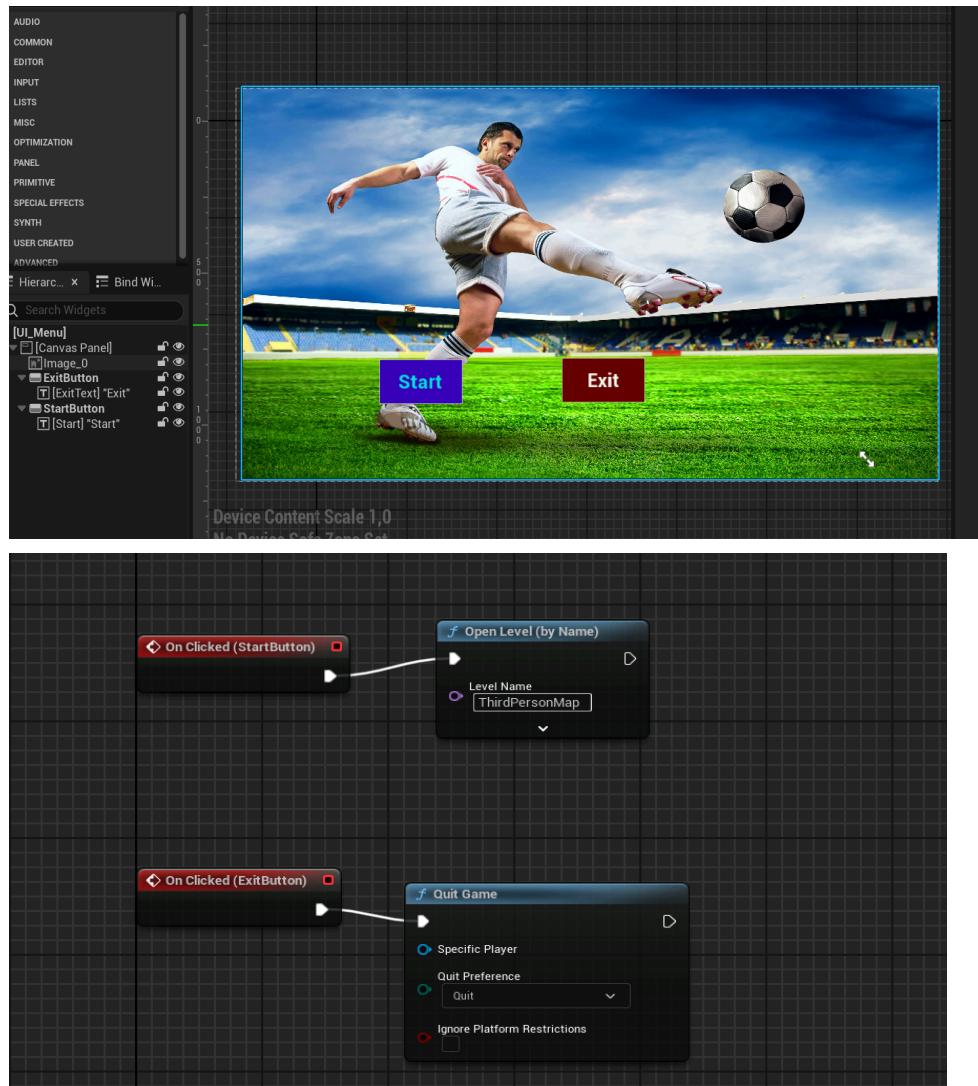
3.2.6. UI Blueprints

A. UI_Menu

En el UI_Menu, que es un Widget Blueprint para el menú principal, se manejan principalmente dos botones: Start y Exit. Aquí está la explicación breve de cómo funcionan los eventos:

- **Componentes:**
 - Imagen: Una imagen de fondo o ícono que se muestra en el menú.
 - Botón Start: Este botón inicia el juego y carga el mapa principal.
 - Botón Exit: Este botón cierra el juego.
- **Event Graph:**
 - **OnClicked (Start Button):**
 - Cuando se hace clic en el botón Start, se ejecuta un evento OnClicked.

- Acción: Este evento carga el nivel ThirdPersonMap (el nivel del juego) usando el nodo Open Level.
- Esto inicia el juego y carga el mapa donde el personaje puede moverse y jugar.
- **OnClicked (Exit Button):**
 - Cuando se hace clic en el botón Exit, se ejecuta el evento OnClicked.
 - Acción: Este evento llama a la función Quit Game, que cierra la aplicación.



B. UI_WidgetInterface

En el UI_WidgetInterface, se muestra información clave del jugador, como el número de pelotas recolectadas y la barra de vida del jugador. A continuación se describen los detalles de cómo funciona este Widget:

- **Componentes del Widget:**

- Imagen de Pelota: Muestra un ícono de pelota en la parte superior izquierda de la pantalla.
- Textbox: Junto a la imagen de la pelota, hay un textbox que está enlazado a la variable contadorPelotas. Este textbox muestra el número de pelotas recolectadas.
- Barra de Progreso: Una barra de progreso que representa la salud del jugador. Está ubicada también en la parte superior izquierda.

- **Event Graph:**

- **Evento IncrementarContador:**

- **Acción:** Este evento se llama para incrementar el número de pelotas recolectadas.

- **Funcionamiento:**

- La variable contadorPelotas se incrementa en 1.
- Luego se llama al evento dispatcher incrementarNumPelotas para que otros elementos o clases puedan reaccionar a este cambio.
- Este evento puede estar vinculado a actualizaciones en la interfaz o a otros elementos de juego.

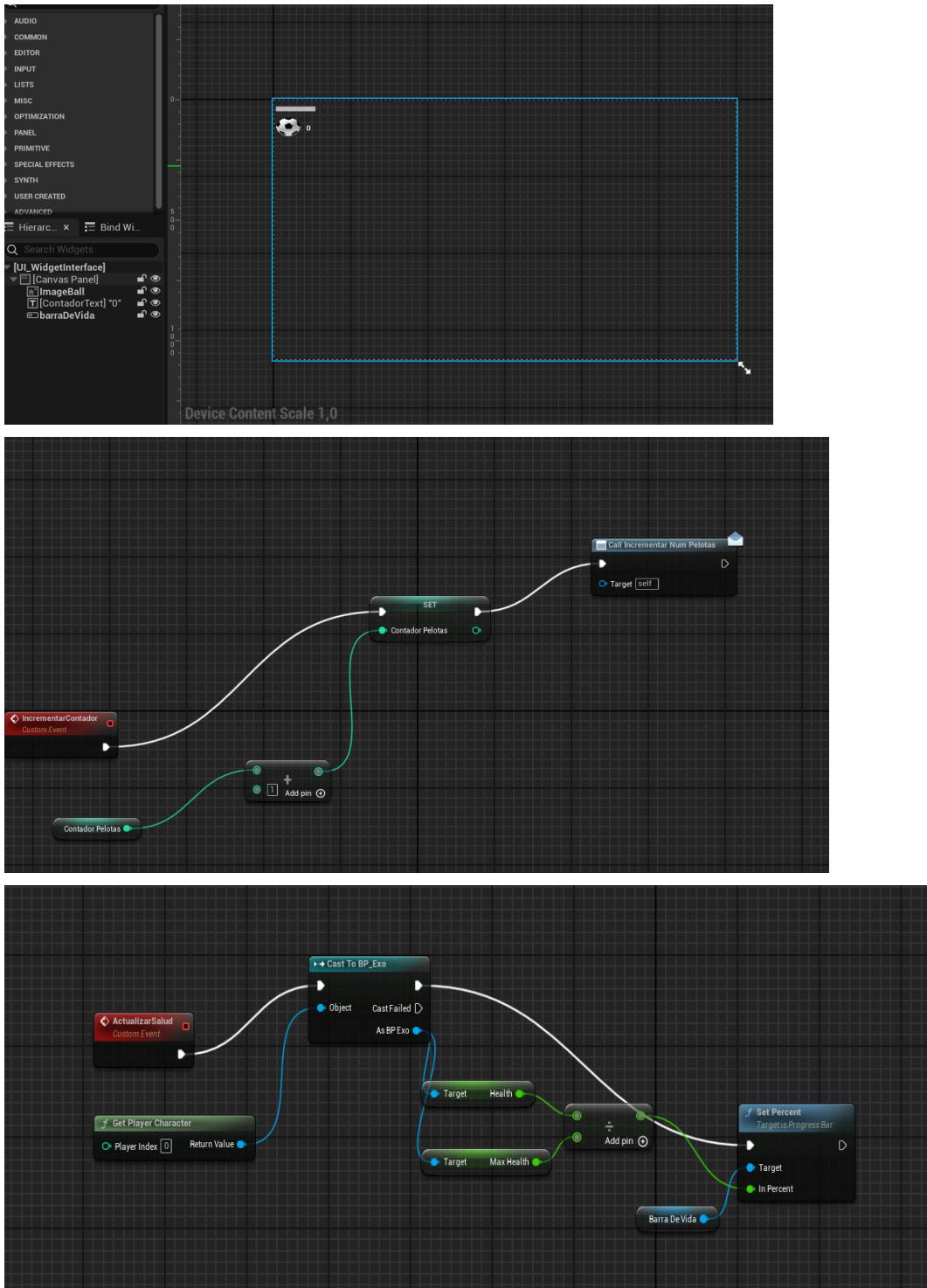
- **Evento ActualizarSalud:**

- **Acción:** Este evento actualiza la barra de progreso de la salud.

- **Funcionamiento:**

- Se obtiene el personaje del jugador mediante Get Player Character.
- Se hace un cast a BP_Exo para obtener las variables health y maxHealth del personaje.
- Luego, se divide la salud actual health entre la maxHealth para obtener el porcentaje de salud restante.

- Se utiliza Set Percent en la barra de progreso para actualizar su valor con el resultado de esta división.

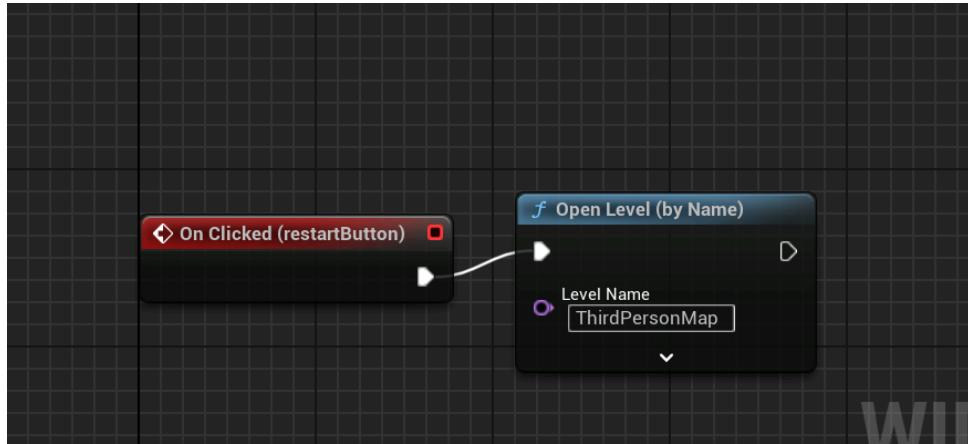


C. UI_EndGame

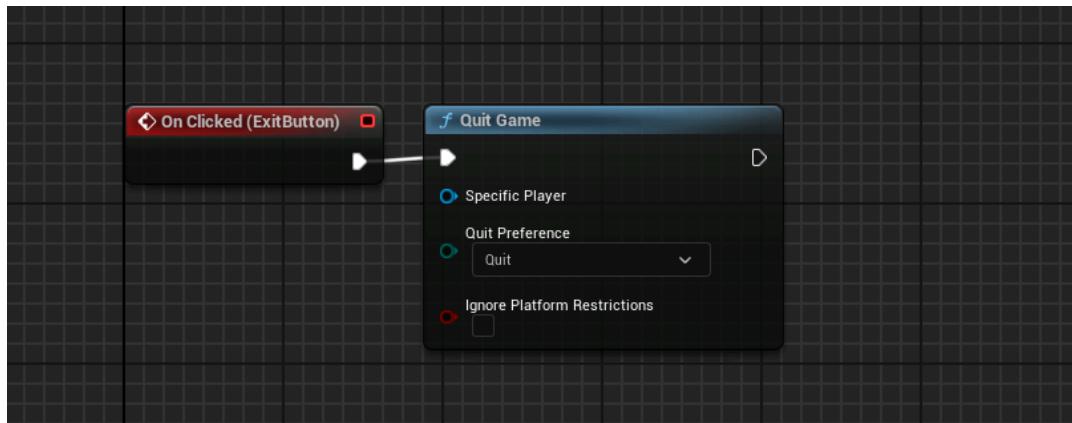
En el UI_EndGame, se gestiona la interfaz que aparece al finalizar el juego, mostrando el resultado y permitiendo al jugador reiniciar el nivel, ver su récord, o salir del juego. Aquí está el desglose:

- **Componentes del Widget:**
 - **Canvas:**
 - Contiene una imagen de fondo.
 - Un Text Block para mostrar el estado del juego: "Victory" o "Defeat".
 - **Tres botones:**
 - Restart: Para reiniciar el nivel.
 - Your Record: Muestra el tiempo y el número de pelotas recolectadas.
 - Exit: Sale del juego.
 - **Elementos adicionales como:**
 - Imagen de pelota.
 - Textos para mostrar el tiempo y el número de pelotas tocadas. Estos elementos son invisibles por defecto y aparecen cuando se presiona el botón Your Record.
- **Event Graph:**
 - **Evento Construct:**
 - Acción: Configura el estado inicial del widget.
 - **Funcionamiento:**
 - Se obtiene la instancia del Game Instance con Get Game Instance y un Cast to BP_GameInstance.
 - Se extrae la variable booleana Victory de BP_GameInstance.
 - Si Victory es true, se asigna el texto "Victory" a una variable string winOrLost.
 - Si es false, se asigna el texto "Defeat".
 - La variable winOrLost está vinculada al Text Block para que el texto se muestre automáticamente.

- **Evento On Clicked(Restart):**
 - Se abre el nivel inicial (ThirdPersonMap) con la función Open Level.

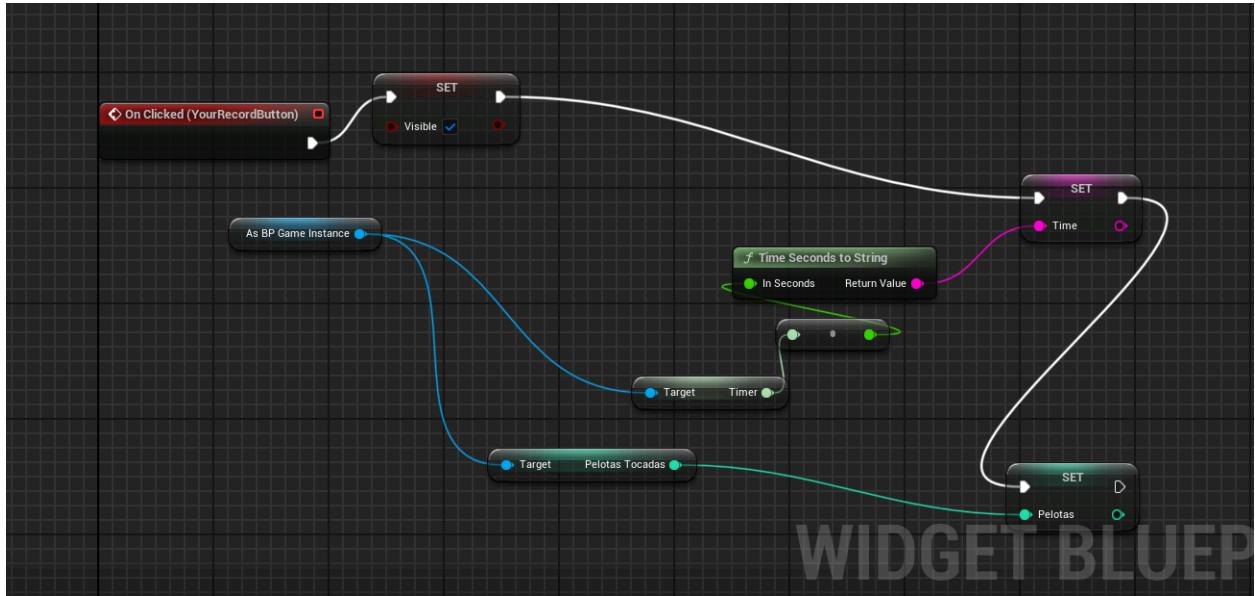


- **Evento On Clicked(Exit):**
 - Llama a la función Quit Game para cerrar el juego.



- **Evento On Clicked(Your Record):**
 - Se obtienen las variables Time y Pelotas Tocadas de BP_GameInstance.
 - La variable Time se convierte a formato de tiempo legible utilizando la función Time Seconds to String.
 - El resultado se almacena en una variable string time, vinculada a un texto que se muestra en el widget.

- También se hace visible un conjunto de elementos en la interfaz, como la imagen de la pelota y los textos asociados.
- Esto se controla con una variable booleana Visible, que se asigna a true.
- La visibilidad de estos elementos está vinculada a funciones que dependen de la variable Visible, permitiendo que los elementos se muestren dinámicamente.



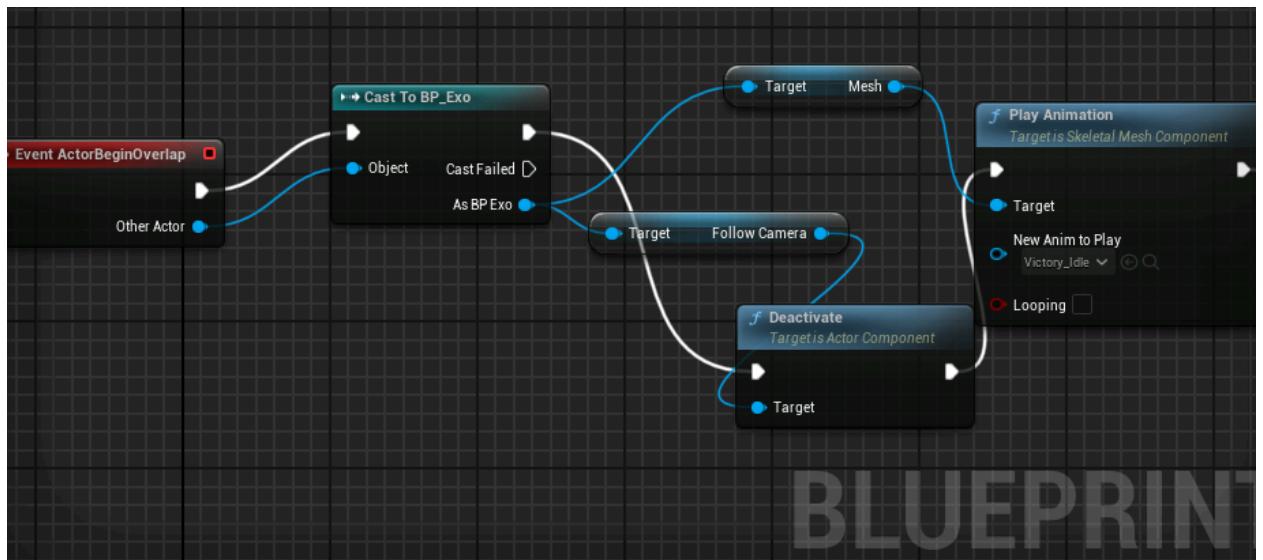
3.2.7. Actors Blueprints

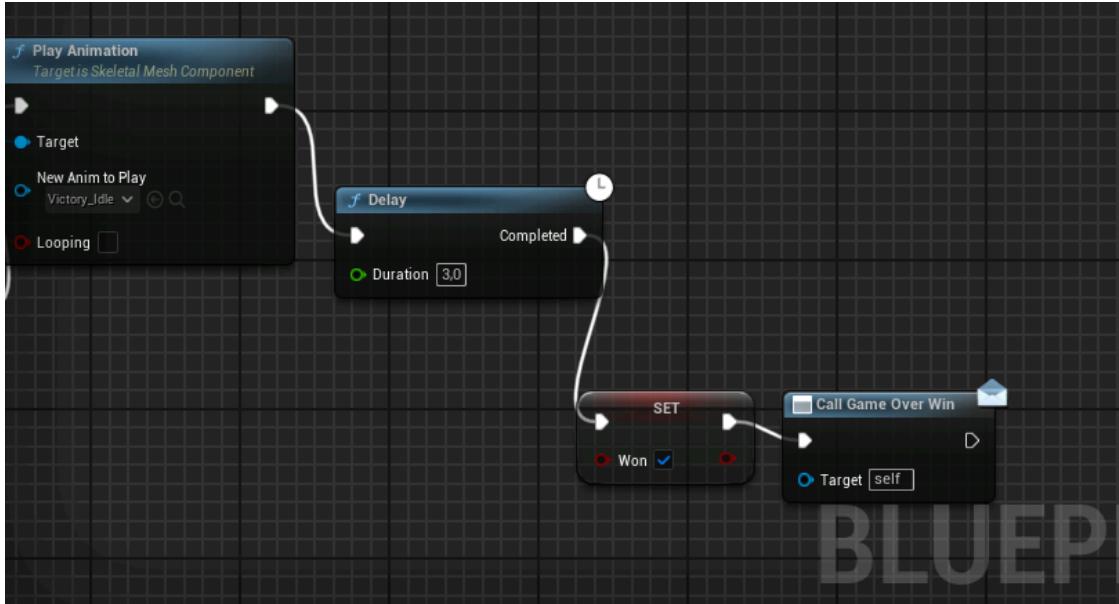
A. BP_VictoryTrigger

En el BP_VictoryTrigger, se gestiona la interacción que ocurre al ganar el juego, activando una animación de victoria y notificando al sistema del resultado. Aquí está el desglose:

- **Event Graph:**
 - **Evento On Begin Overlap:**
 - Acción: Detecta cuando el personaje principal entra en el trigger.
 - **Funcionamiento:**
 - **Casting:**
 - Se realiza un Cast to BP_Exo para verificar si el actor que entró en el trigger es el personaje principal (BP_Exo).

- Si el cast tiene éxito, continúa el flujo.
- Desactivación de Follow Camera:
- Se obtiene el componente Follow Camera del actor BP_Exo.
- Se llama al nodo Desactivar para deshabilitar la cámara de seguimiento y permitir que la Cine Camera tome el control.
- **Reproducción de Animación de Victoria:**
 - Se llama a la función Play Animation en el actor BP_Exo.
 - Se especifica la animación Victory_Idle como la animación a reproducir.
- **Delay:**
 - Se añade un nodo Delay que espera la duración exacta de la animación antes de proceder al siguiente paso.
- **Actualizar Estado del Juego:**
 - Se establece la variable Won a true, indicando que el jugador ha ganado.
- **Notificación del Resultado:**
 - Se llama al Event Dispatcher Game Over Win para notificar al sistema que el juego ha terminado con una victoria.





B. BP_DeathTrigger

El propósito del BP_DeathTrigger es detectar si el personaje principal ha caído y gestionar los eventos asociados, como activar la animación de muerte y notificar al sistema del resultado. A continuación se describe el funcionamiento:

- **Event Graph**

- **Evento On Begin Overlap:**

- Acción: Detecta cuando el personaje entra en el trigger.

- **Funcionamiento:**

- **Casting:**

- Se realiza un Cast to BP_Exo para verificar si el actor que activa el trigger es el personaje principal (BP_Exo).
 - Si el cast tiene éxito, el flujo continúa.

- **Obtener la Anim Instance:**

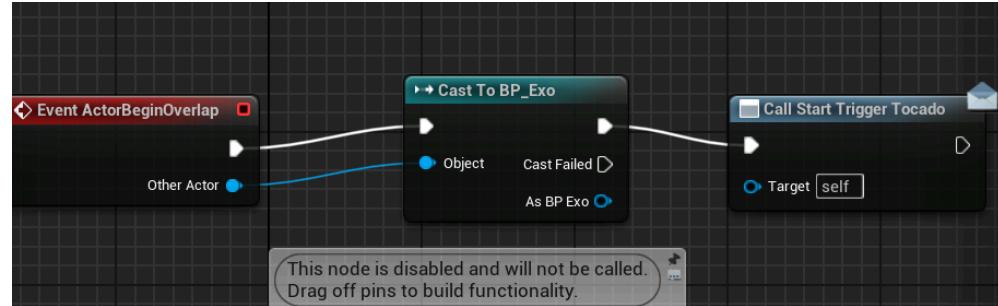
- Se obtiene la Anim Instance del personaje principal (BP_Exo) usando el nodo Get Anim Instance.
 - Se realiza un Cast to ABP_AnimExo para acceder a las variables y funciones del blueprint de animación del personaje.

- **Activar Animación de Muerte:**
 - Se establece la variable IsDead en true dentro de ABP_AnimExo, lo que activa la animación de muerte del personaje.
- **Delay:**
 - Se añade un nodo Delay de 3 segundos para permitir que la animación de muerte se reproduzca completamente antes de continuar.
- **Notificación del Estado:**
 - Se llama al Event Dispatcher Has Fallen para notificar al sistema que el personaje ha caído.

C. BP_StartFollowingTrigger

El BP_FollowingTrigger tiene como objetivo principal indicar al enemigo que debe comenzar a seguir al personaje principal. Su implementación es sencilla y se centra en detectar la entrada del personaje en el trigger y notificar a través de un Event Dispatcher.

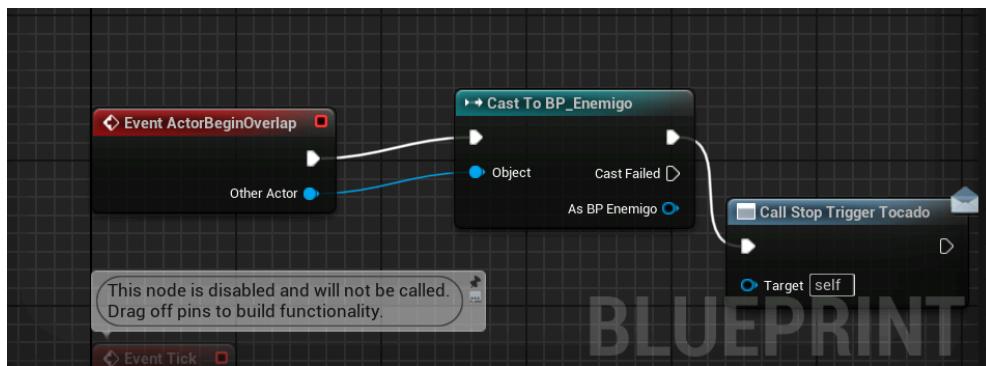
- **Event Graph**
 - **Evento On Begin Overlap:**
 - Acción: Se activa cuando cualquier actor entra en el trigger.
 - **Funcionamiento:**
 - **Casting:**
 - Se realiza un Cast to BP_Exo para verificar si el actor que activa el evento es el personaje principal.
 - Si el cast tiene éxito (es decir, el actor es el personaje principal), el flujo continúa.
 - **Llamar al Event Dispatcher:**
 - Se llama al Event Dispatcher Start Trigger Tocado para notificar a los sistemas relacionados que el trigger ha sido activado.



D. BP_StopFollowingTrigger

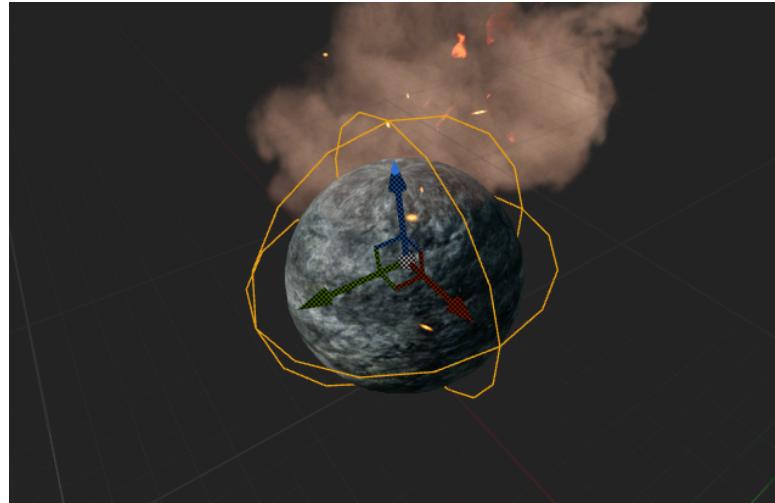
El BP_StopFollowingTrigger tiene como objetivo principal indicar al enemigo cuándo debe detener la persecución del personaje principal. Su funcionamiento es muy similar al BP_StartFollowingTrigger, pero aquí se enfoca en los enemigos en lugar del personaje principal.

- **Event Graph**
- **Evento On Begin Overlap:**
 - Acción: Se activa cuando cualquier actor entra en el trigger.
 - Funcionamiento:
 - Casting:
 - Se realiza un Cast to BP_Enemy para verificar si el actor que activa el evento es un enemigo.
 - Si el cast tiene éxito (es decir, el actor es un enemigo), el flujo continúa.
 - Llamar al Event Dispatcher:
 - Se llama al Event Dispatcher Stop Trigger Tocado para notificar a los sistemas relacionados que el trigger ha sido activado.



E. BP_Rock

El BP_Rock es un blueprint diseñado para representar una roca peligrosa en el juego. Su objetivo es infligir daño al personaje principal si entra en contacto con ella, generar efectos visuales y sonoros, y luego desaparecer.



- **Componentes Principales**
 - Static Mesh (Esfera):
 - Representa la forma física de la roca.
 - Particle System (Fire):
 - Proporciona un efecto visual de fuego para la roca.
 - Sphere Collision:
 - Define el área de colisión que detectará interacciones con otros actores.
- **Event Graph**
 - **Evento On Begin Overlap:**
 - Acción: Se activa cuando un actor entra en el área de colisión de la roca.
 - **Funcionamiento:**
 - **Casting:**
 - Se realiza un Cast to BP_Exo para verificar si el actor que activa el evento es el personaje principal.
 - Si el cast tiene éxito, el flujo continúa.
 - **Aplicar Daño:**

- Se llama a la función Apply Damage, indicando un daño de 25 puntos.

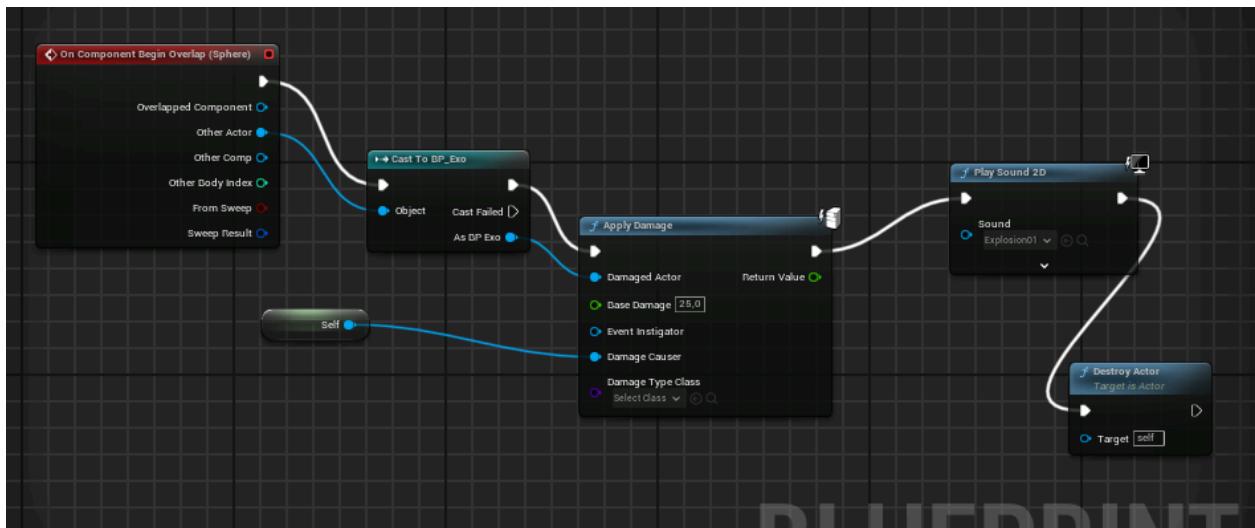
- Esto afecta la variable Health del personaje principal.

■ **Reproducir Sonido:**

- Se ejecuta un Play Sound 2D para reproducir un efecto sonoro de explosión, mejorando la experiencia auditiva.

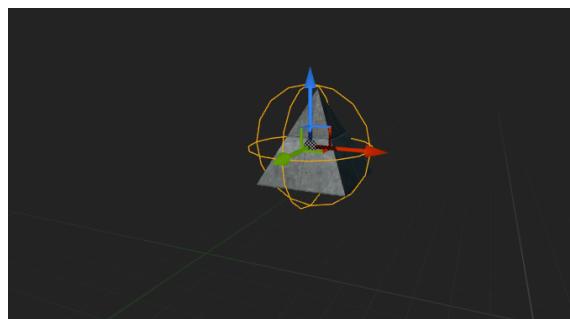
■ **Destrucción del Actor:**

- Se llama a Destroy Actor para eliminar la roca del nivel tras haber cumplido su función.

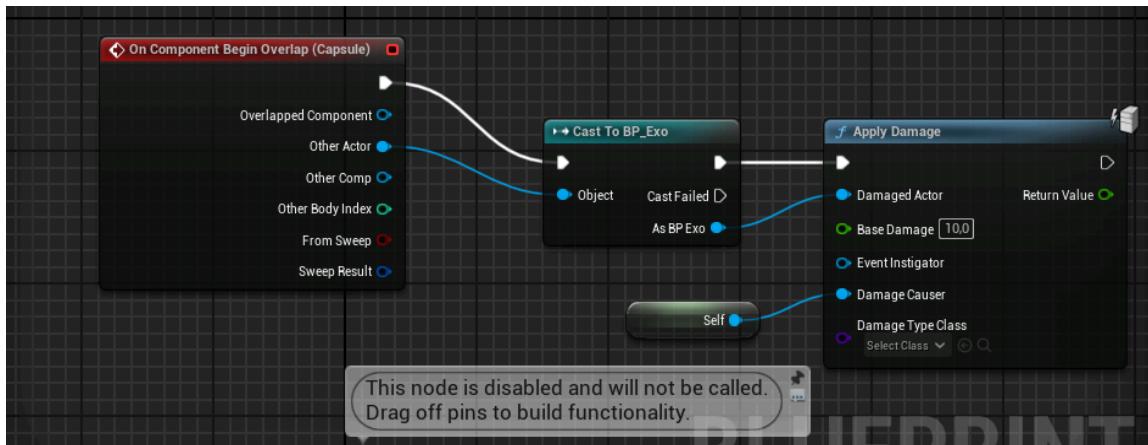


F. BP_Pyramid

El BP_Pyramid es un blueprint que representa un obstáculo estático en forma de pirámide. Su propósito principal es infiligr daño al personaje principal al entrar en contacto con él.

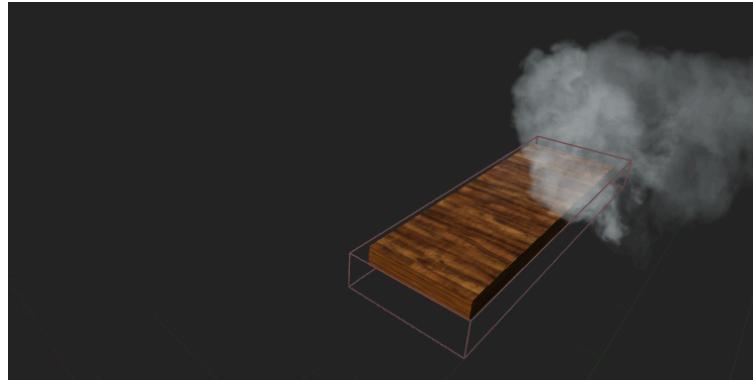


- **Componentes Principales**
 - Static Mesh (Tipo Pirámide):
 - Capsule Collision
- **Event Graph**
 - **Evento On Begin Overlap:**
 - Acción: Se activa cuando otro actor entra en el área de colisión del obstáculo.
 - **Funcionamiento:**
 - **Casting:**
 - Se realiza un Cast to BP_Exo para verificar si el actor que activa el evento es el personaje principal.
 - Si el cast tiene éxito, el flujo continúa.
 - **Aplicar Daño:**
 - Se llama a la función Apply Damage, asignando un daño de 10 puntos.
 - Esto afecta directamente a la variable Health del personaje.



G. BP_Plane

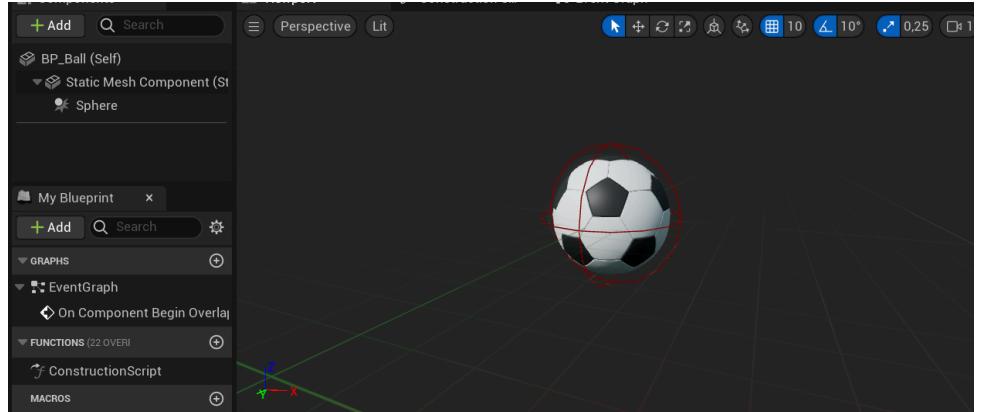
El BP_Plane es un blueprint que representa un obstáculo peligroso en el nivel. Este obstáculo infinge un daño significativo al personaje principal al entrar en contacto con él.



- **Componentes Principales**
- Static Mesh (Tipo Cubo):
- Cube Collision
- Particle System (Smoke):
- **Event Graph**
 - **Evento On Begin Overlap:**
 - Acción: Se activa cuando otro actor entra en el área de colisión del Cube Collision.
 - **Funcionamiento:**
 - **Casting:**
 - Se realiza un Cast to BP_Exo para verificar si el actor que activa el evento es el personaje principal.
 - Si el cast tiene éxito, el flujo continúa.
 - **Aplicar Daño:**
 - Se llama a la función Apply Damage, asignando un daño de 50 puntos al personaje principal.

H. BP_Ball

El BP_Ball es el blueprint que representa una pelota en el juego, diseñada para ser recolectada por el personaje principal. Su interacción incrementa un contador en la interfaz de usuario y reproduce un efecto sonoro.



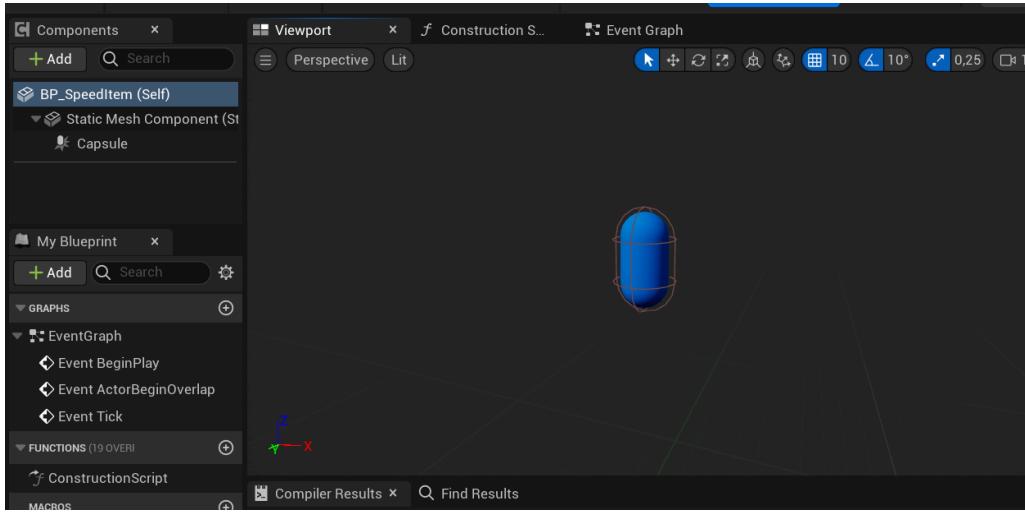
- **Componentes Principales**
 - Static Mesh (Pelota):
 - Modelo de pelota importado desde Unreal Engine, asignado como el aspecto visual del actor.
 - Sphere Collision:
 - Acción: Se activa cuando otro actor entra en el área de colisión del Sphere Collision.
 - Funcionamiento:
 - Casting:
 - Se realiza un Cast to BP_Exo para verificar si el actor que activa el evento es el personaje principal.
 - Si el cast tiene éxito, el flujo continúa.
 - Reproducir Sonido:
 - Se llama a Play Sound para reproducir un efecto sonoro que indica la recolección de la pelota.
 - Destrucción del Actor:
 - Se destruye el actor actual (la pelota) con la función Destroy Actor.
 - Actualizar Contador en UI:
 - Se obtiene el widget de la interfaz de usuario UI_WidgetInterface:

- Utilizando Get All Widgets of Class, se localiza el único widget activo en la escena.
- Se selecciona el primer elemento con Get[0] (ya que hay solo uno en ejecución).
- Se llama a la función Incrementar Contador del UI_WidgetInterface para incrementar el número de pelotas recolectadas.

I. BP_SpeedItem

El BP_SpeedItem es un objeto que incrementa la velocidad del actor (personaje) cuando entra en contacto con él. Este objeto tiene un Static Mesh que representa su forma y una Capsule Collision para detectar la colisión con el personaje principal.

- **Componentes Principales**
 - Static Mesh (Shape_NarrowCapsule):
 - Representa visualmente el SpeedItem, su modelo es un capsule estrecha que sirve como la forma física del objeto en el juego.
 - Capsule Collision:
 - Componente de colisión que detecta el contacto con el personaje principal (BP_Exo).
 - Al entrar en contacto con el jugador, se activa un evento que modifica la velocidad del personaje.
- **Flujo de Interacción**
 - El BP_SpeedItem no tiene lógica dentro de su propio Event Graph, ya que toda la interacción con el jugador está gestionada por el blueprint del personaje principal (BP_Exo).



J. BP_JumpItem

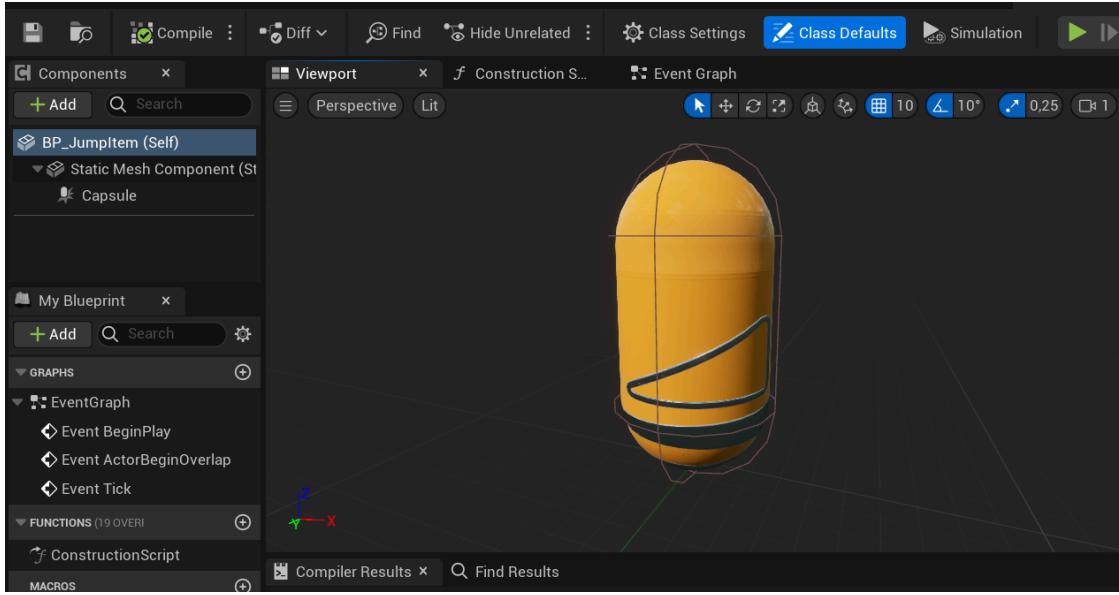
El BP_JumpItem es un objeto similar al BP_SpeedItem, pero en lugar de incrementar la velocidad de movimiento, aumenta la velocidad de salto del personaje cuando entra en contacto con él. Este objeto tiene un Static Mesh para su representación visual y una Capsule Collision para detectar la colisión con el personaje principal.

- **Componentes Principales**

- Static Mesh (Shape_NarrowCapsule):
 - Representa el JumpItem de manera visual, usando un Static Mesh que tiene la forma de una cápsula estrecha. Esta es la forma del objeto en el mundo del juego.
- Capsule Collision:
 - Detecta la colisión con el personaje (BP_Exo) para activar la lógica de aumento de salto.
 - Al detectar la superposición con el personaje, ejecuta el evento para aumentar la altura del salto temporalmente.

- **Flujo de Interacción**

- Al igual que el BP_SpeedItem, la lógica de interacción del BP_JumpItem no reside dentro de este blueprint, sino que es gestionada desde el blueprint del personaje (BP_Exo).



4. Recursos de Terceros Importados

En el proyecto, se han importado varios recursos para agregar contenido visual y enriquecer la jugabilidad. Estos recursos son los siguientes:

- **Modelos Importados**

- Modelos y Animaciones del Personaje Principal y Enemigo:
 - Tanto el personaje principal como el enemigo fueron importados desde Mixamo, una plataforma que proporciona modelos 3D y animaciones prediseñadas.
 - El personaje principal incluye animaciones como idle, run, jump, slide y victory.
 - [enlace:https://www.mixamo.com/#/?page=2&type=Character](https://www.mixamo.com/#/?page=2&type=Character)
 - El enemigo utiliza animaciones básicas para idle y run.
 - [enlace:https://www.mixamo.com/#/?page=3&type=Character](https://www.mixamo.com/#/?page=3&type=Character)
- Pelota (Ball):
 - Se ha importado un modelo de pelota desde Unreal Fab. Este modelo es el objeto que el personaje recolecta durante el juego, representado en el BP_Ball.
 - Este modelo tiene asignado un Sphere Collision para detectar colisiones con el personaje y activar la lógica de incrementar el contador de pelotas tocadas.

[enlace:https://www.fab.com/listings/1becc7fc-1746-41c5-acbd-43acbc6f3001](https://www.fab.com/listings/1becc7fc-1746-41c5-acbd-43acbc6f3001)

- Football Goal (Portería de Fútbol):
 - También se ha importado un modelo de football goal (portería de fútbol) desde Unreal Fab. Este modelo se utiliza en el entorno del juego

enlace:<https://www.fab.com/listings/e3bfd771-64dc-4533-98aa-01eb08284098>

- **Textura Importada**

- **Textura del Pitch (Campo de Fútbol):**
 - La textura del campo de fútbol se ha importado desde Unreal Fab. Esta textura se utiliza para darle un aspecto realista y adecuado al entorno del juego.
 - La textura está asignada a un material que cubre el mesh del campo (cubo) denominado PitchMaterial, dando una apariencia coherente con el tema del juego.

enlace:<https://www.fab.com/listings/459a98c9-6424-4a96-a625-1d915f153c8b>

- **Audios Descargados desde Pixabay:**

- Todos los efectos de sonido utilizados en el proyecto han sido descargados desde Pixabay.

enlace:<https://pixabay.com/es/music/>