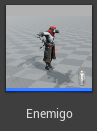
## Eliminar Enemigos

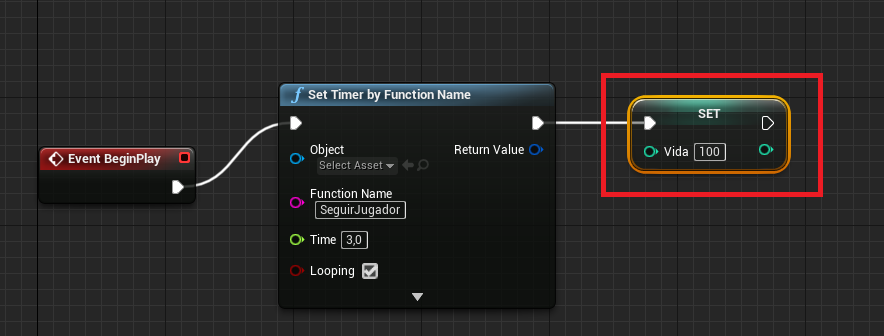
Vamos a implementar un sistema para que los **Enemigos** puedan ser eliminados mediante **Disparos**.

#### Enemigo

Comenzamos implementado el **Enemigo**. Les vamos a dar una **Vida inicial**, cada disparo reducirá esa vida hasta que llegue a cero, en ese momento se destruirá el **Enemigo**.



Abrimos el **Event Graph** de enemigo. Buscamos el evento **BeginPlay,** desde ese evento realizamos un **Set Vida** y le damos el valor 100.



Nos situamos a una zona vacía del Blueprint y creamos un nuevo **Custom Event,** lo llamamos **Impactado**.



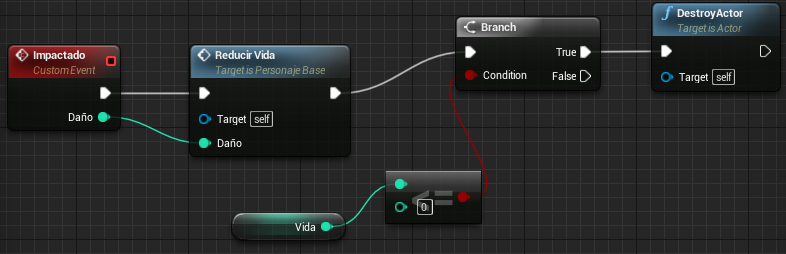
Desde las propiedades del nodo Inputs, pulsamos en **New** y añadimos un nuevo parámetro de entrada (inputs) **Daño**, de tipo Integer.

|  |  |
| --- | --- |
|  |  |

Enviamos el daño al evento **Reducir Vida** (que está implementado en la clase Padre)

Después de eso comprobamos si la **Vida <= 0**. (El **Branch** equivale a un **if )**

Sí la condición se cumple significa que el personaje debe ser eliminado, llamamos al evento  **DestroyActor.**

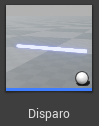


Compilamos y guardamos los cambios.



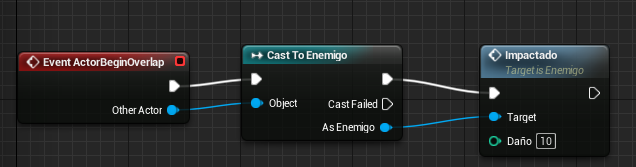
#### Disparo

Abrimos la clase **Disparo**.

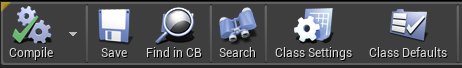


Nos dirigimos al **Event Graph** y buscamos el evento **ActorBeginOverlap**, este evento se ejecuta al inicio de una superposición

Hacemos un **Cast To Enemigo** del Actor contra el que se produce la superposición, sí se trata de un **Enemigo** llamamos al evento **Impactado**.

****

Compilamos y guardamos los cambios.

****

Sí ejecutamos el juego podemos ver como el enemigo es eliminado después de recibir varios disparos.

El evento **ActorBeginOverlap** se ejecuta automáticamente cuando las **áreas de colisión** se superponen.

****

Vamos a hacer también que el **Disparo** se destruya al impactar contra un **Enemigo**, para ello debemos agregar la función **DestroyActor.**

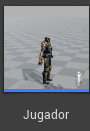


# Eliminar al jugador

Vamos a hacer que un **Enemigo** puedan dañar al **Jugador** cuando están cerca de él.

#### Jugador

Abrimos la clase **Jugador**

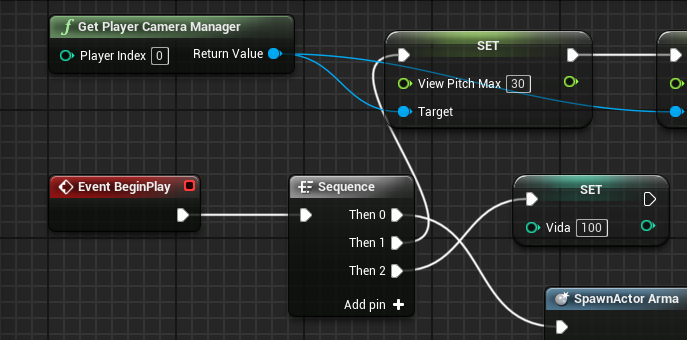


Vamos a un área vacíadel **Event Graph.**

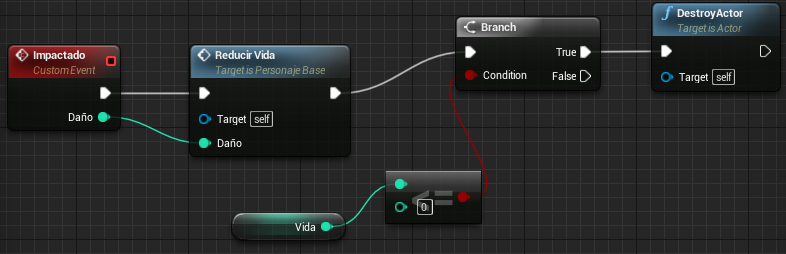
En primer lugar, nos dirigimos al evento **BeginPlay** y agregamos un nuevo nodo al **Sequence**, pulsando en **Add pin +**



Desde el nuevo **Then**, agregamos una llamada a **Set Vida**, y colocamos el valor 100

****

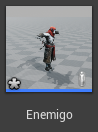
Buscamos una parte vacía y creamos un **Custom Event** de la misma forma que hicimos en el **Enemigo** (Se puede copiar y pegar).

****

Luego veremos que no es muy adecuado hacer un **DestroyActor** del Jugador principal.

#### Enemigo causa daño

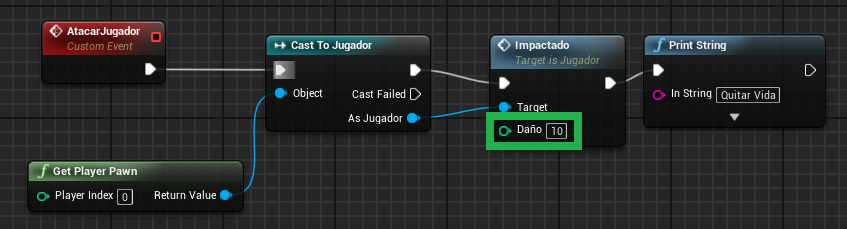
El **Enemigo** va a tener un ataque a corta distancia.



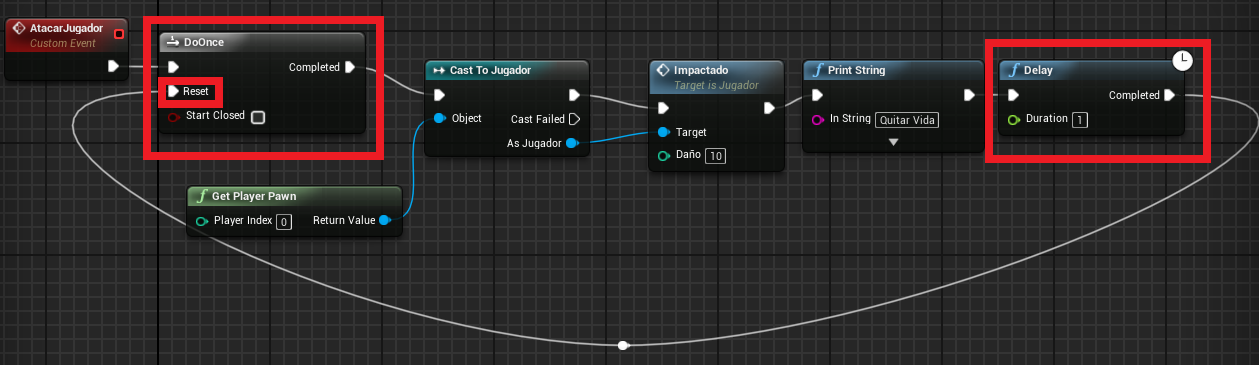
Abrimos el **Event Graph** del **Enemigo.**

Comenzamos agregando un nuevo **Add Custom Event…** al que llamamos **AtacarJugador**

Dentro de este evento obtenemos el **Get Player Pawn** (jugador principal del juego) lo casteamos a **Jugador- Cast To Jugador** y llamamos a su evento **Impactado.**

****

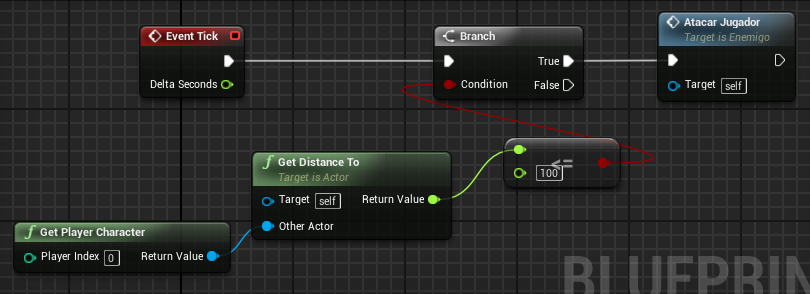
Sí queremos asegurar que no se llama a **AtacarJugador** más de 1 vez por segundo podemos agregar un **DoOnce** y un **Delay de un segundo.**

****

En cada **Tick** del juego comprobamos si el **Jugador** está a menos de 100px del Enemigo, en caso afirmativo ejecutamos **AtacarJugador**.

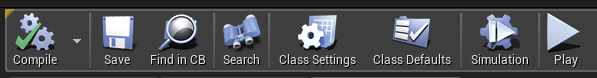
Buscamos el evento **Tick** (esta pre-incluido)

****

****

|  |
| --- |
| **Tick o Timers.** Seguramente hacer la comprobación de atacar al Jugador en cada Tick es excesiva. Podríamos crear un **Timer/hilo** que hiciese la comprobación cada segundo o más. (Algo muy similar a lo que hicimos con la función SeguirJugador) |

Compilamos y guardamos

****

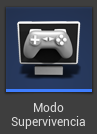
Si ejecutamos el juego podemos comprobar que el **Enemigo** puede destruir al **Jugador.**

Cuando el **Jugador** es destruido ya no podemos interaccionar con el juego.

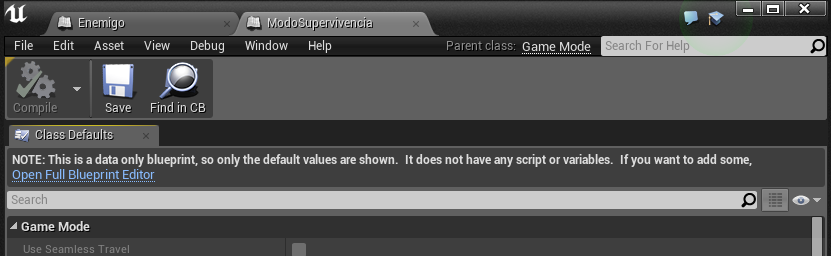
****

#### Reglas del Juego – Reiniciar el nivel

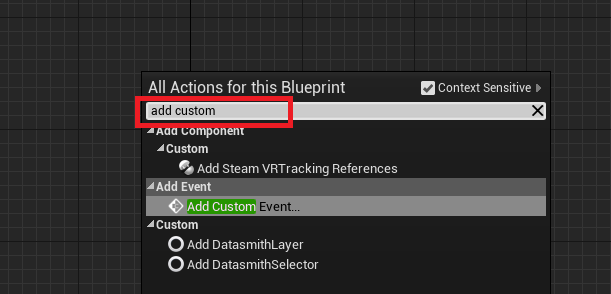
Abrimos el modo de Juego, **Modo Supervivencia**



Abrimos editor completo de Blueprint.

****

Vamos a crear un nuevo **Add Custom Event** al que llamamos **Comprobar Jugador**

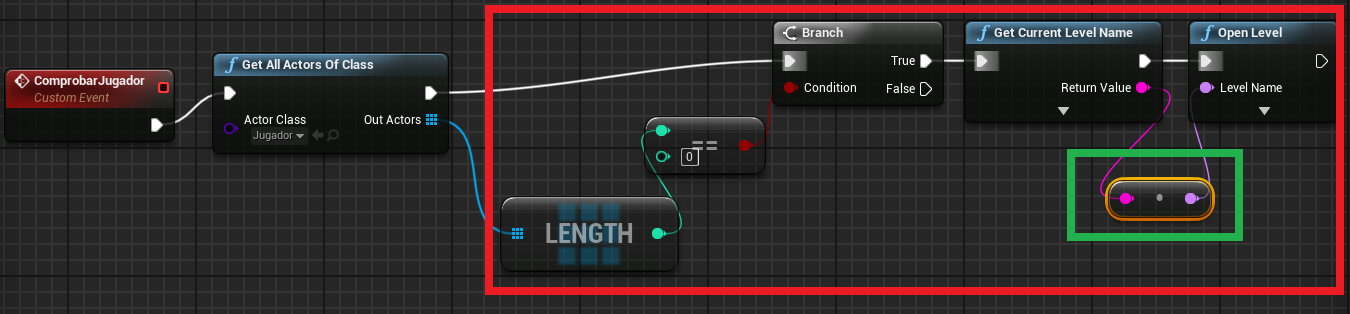


Desde el evento llamamos a un nodo de tipo **Get all Actors Of class**, y seleccionamos la clase **Jugador ,** nos va a retornar un **Array** con todos Jugadores que haya conseguido encontrar en el Nivel.



Sí el tamaño es 0 significa que no queda ningún jugador, volvemos a abrir el nivel en el que estamos actualmente.

El cast marcado en verde, es automático, al conectar los valores



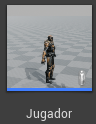
Ahora tenemos varias opciones para invocar al Evento **ComprobarJugador**

* El propio **GameMode** hace esta comprobación en el **Tick**
* Cuando el **Jugador** se destruye accede al GameMode y hace esta comprobación (está es más optima, la comprobación de obtener los actores sobraría porque ya sabemos que esta destruido)

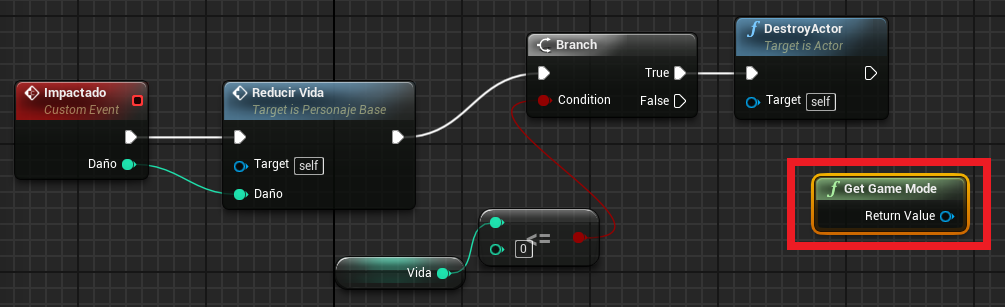
Compilamos y guardamos los cambios



Accedemos al **Jugador**



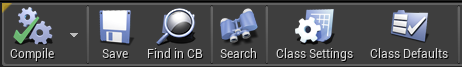
Localizamos el evento **Impactado**, al final del evento obtenemos el modo de juego con **Get Game Mode**



Casteamos el modo de juego **ModoSuperVivencia** y llamamos a la función **Comprobar Jugador**



Guardamos los cambios y compilamos

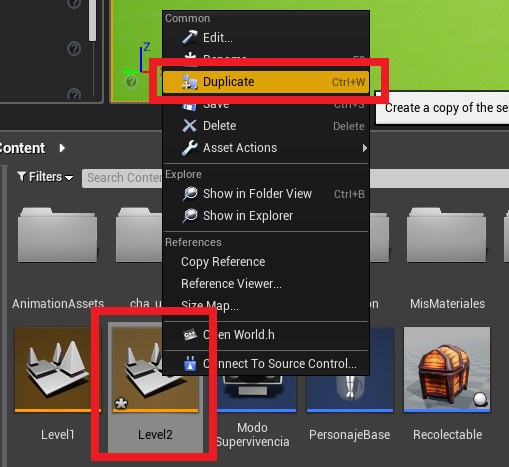


Ejecutamos el Juego y hacemos que destruyan al jugador, el nivel se reiniciara



# Siguiente Nivel (Para hacer en casa)

Accedemos a la carpeta **Content** y duplicamos nuestro fichero **Level2,**



Esto nos genera una copia del nivel **Level3,** para que no sea totalmente igual le quitamos los enemigos. Ambos niveles deben tener al menos un **recolectable**.

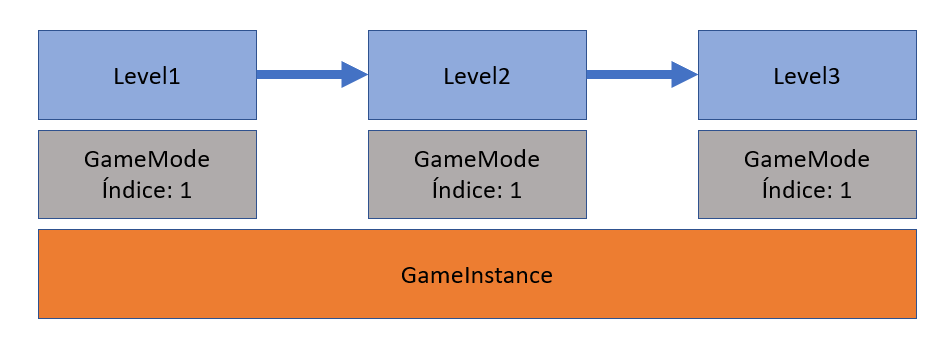


Cuando un nivel se quede sin **recolectables** pasaremos al siguiente (Level2 -> Level3). Lo primero que deberíamos tener un índice del nivel en el que estamos

**¿Dónde guardamos la variable índice, en el GameMode?**

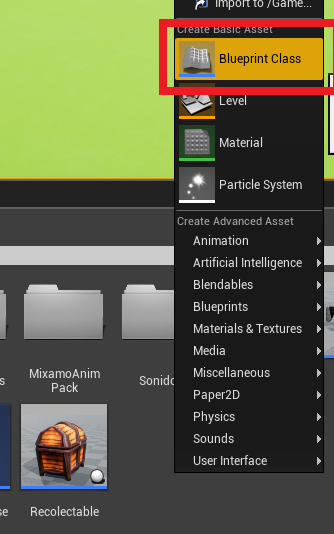
**NO podemos guardarlo en el ModoSuperVivencia ya que se crea de nuevo objeto de este tipo cada vez que se abre un nuevo nivel OpenLevel**

**GameMode** esta pensado para implementar lógica de nivel, y guardar datos referentes al nivel, por ejemplo: cuantas puertas has abierto en el nivel, tiene sentido que se reinicie al cambiar de nivel.

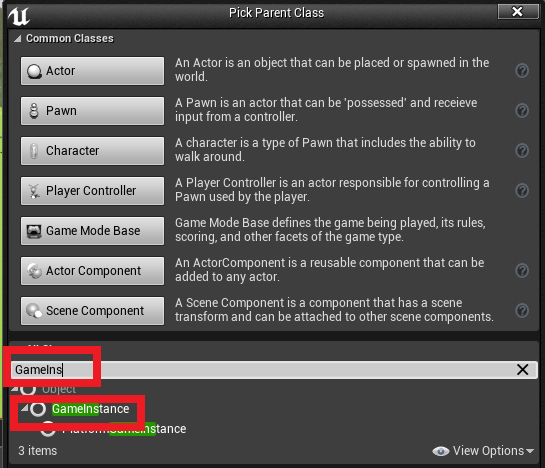
****

Para guardar variables comunes usamos el **GameInstance** ( Se usa de una forma casi idéntica al **GameMode** )

Creamos una nueva Blueprint Class.



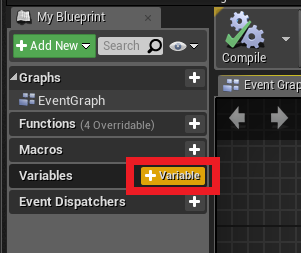
Debemos heredar de la clase base **GameInstance.**

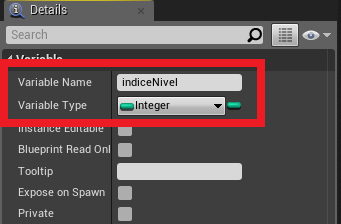


Renombramos el fichero a **InstanciaSupervivencia** .



Entramos en la clase y declaramos una variable nueva **indiceNivel** de tipo integer.

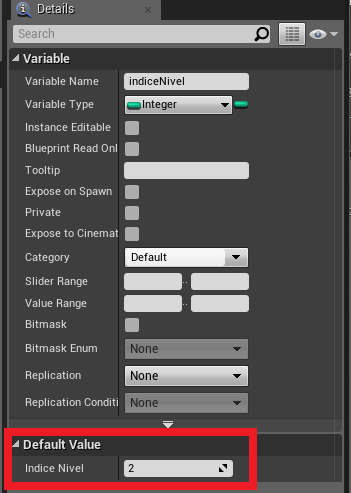




Compilamos para poder darle un valor por defecto a **indiceNivel**.



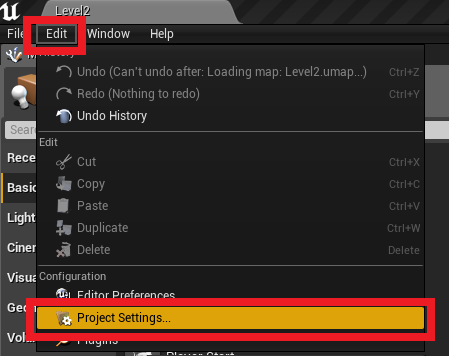
Le asignaremos el valor 2 (en este caso NO queremos comenzar por el Level1, ya que es un mapa de prueba)



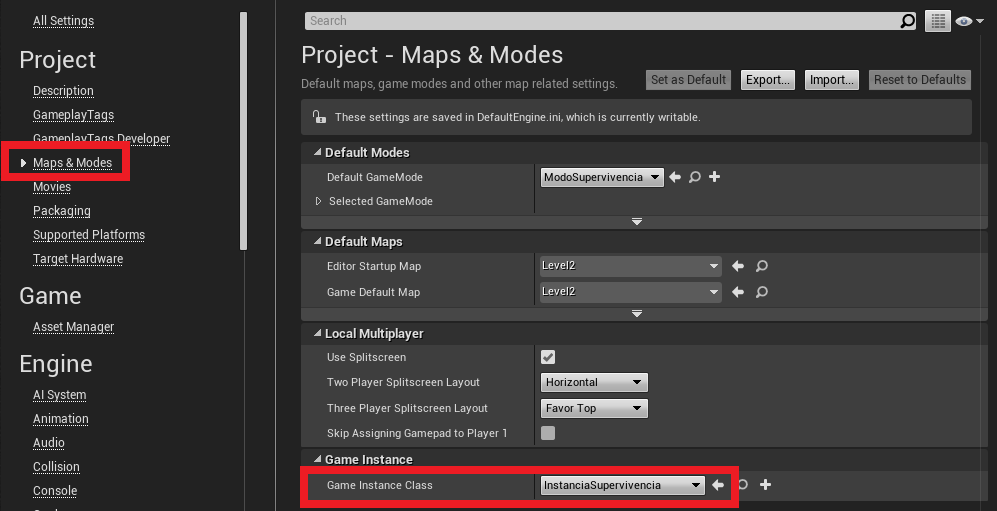
Compilamos y guardamos los cambios.



Al igual que hicimos con el modo de juego debemos **establecer la instancia** en el proyecto. Abrimos las propiedades del proyecto **Edit -> Project Settings …**

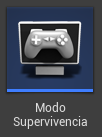


Seleccionamos la categoría **Maps & Modes** y configuramos el valor **de Game Instance Class** para que sea la clase que acabamos de implementar.

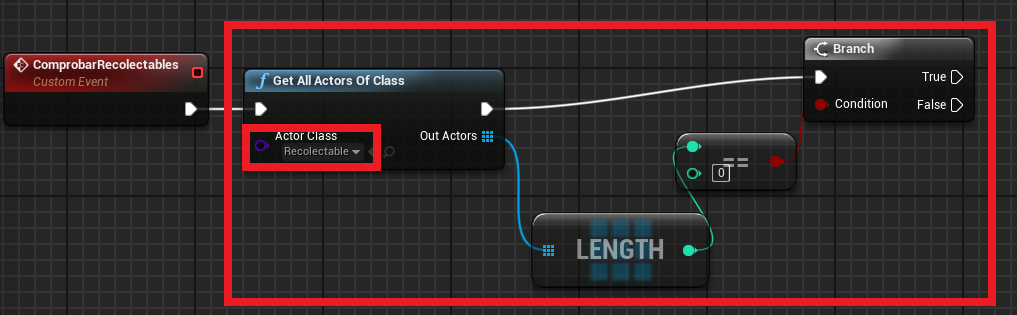


Trataremos de almacenar la mínima cantidad de variables y lógica en la instancia.

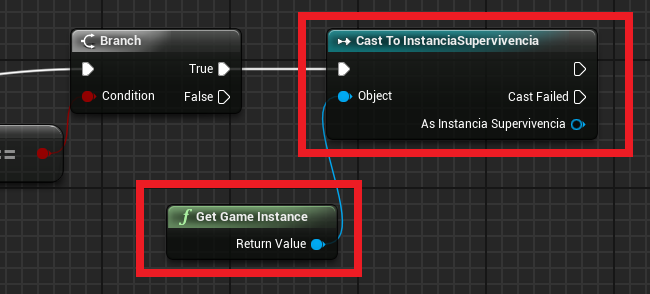
Volvemos al **ModoSupervivencia**



Agregamos un **Add Custom Event**… llamado **ComprobarRecolectables .** Con la función **Get All Actors Of Class** Podemos recuperar todos los **recolectables** del nivel y ver si el número total es 0.



En caso de que sea 0, debemos recuperar la instancia con **Get Instance** y hacer un **Cast to InstanciaSupervivencia**

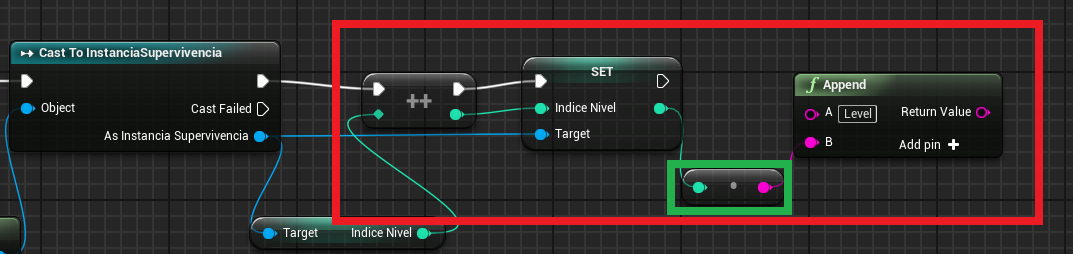


Con la referencia a **Instancia Supervivencia** podemos obtener el **IndiceNivel (Get IndiceNivel)** aplicamos sobre la variable un **increment ( ++)**



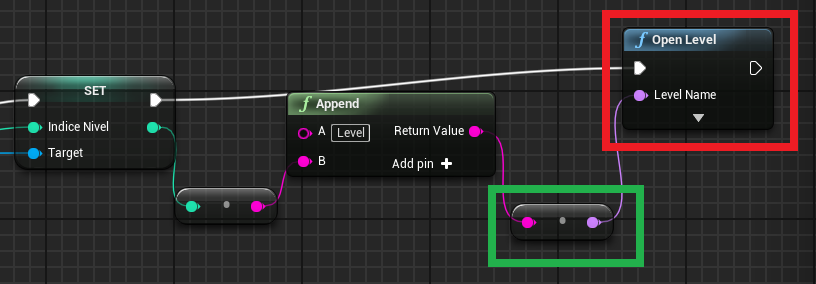
Guardamos el resultado de incrementar la variable de nuevo en **IndiceNivel (Set IndiceNivel).** Usamos la función **Append** para crear un String Level + IndiceNivel

Los Cast marcados en verde son automáticos al unir los puntos



Llamamos a **OpenLevel** enviando como parámetro el valor resultante.

Los Cast marcados en verde son automáticos al unir los puntos

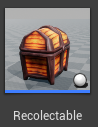


Compilamos y guardamos los cambios.

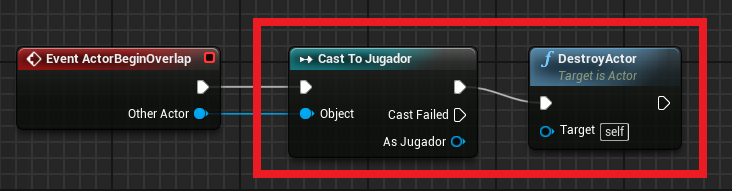


Ahora solo falta llamar a **ComprobarRecolectables** desde los **Recolectables** cuando estos se eliminan.

Abrimos la clase **Recolectable**

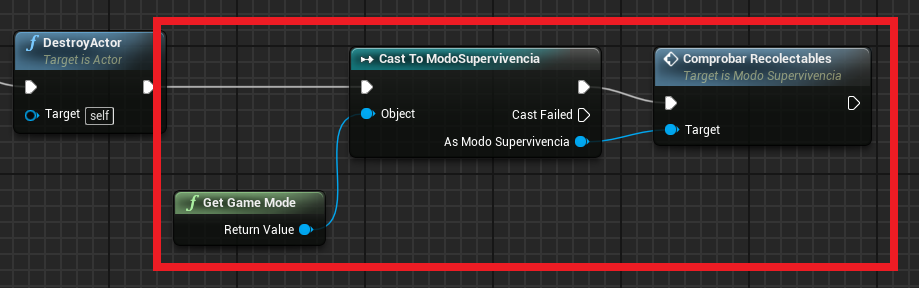
****

Localizamos el evento **ActorBeginOverlap**, al hacer una superposición comprobamos si se trata del **Jugador** y en ese caso eliminamos el propio recolectable con **DestroyActor**.



Después obtenemos el GameMode con **Get Game Mode** y se hace un **Cast to ModoSupervivencia** .

Con la referencia a **ModoSupervivencia** se puede llamar al evento **Comprobar Recolectables** (contenido en el objeto)



Compilamos y guardamos los cambios.



El editor hace que el juego se inicie en el nivel abierto, por ello abrimos el **Level2**. (Para diferenciar el Level2 del Level3 hemos quitado los enemigos en el Level3)



Al pasarnos el Level2 salta al Level3 **de forma correcta**, pero si se acaban los **recolectables** del Levle3



Dará un error, no hemos comprobado cual es el último nivel.

