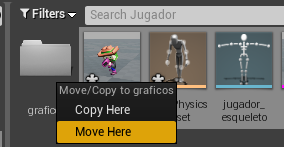
## Jugador (Crear clase)

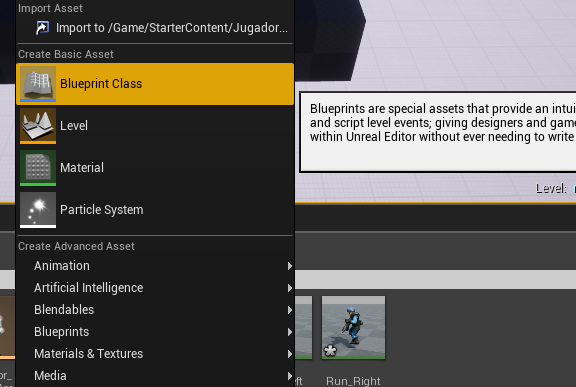
Vamos a crear una carpeta **Graficos** dentro de la carpeta **Jugador**



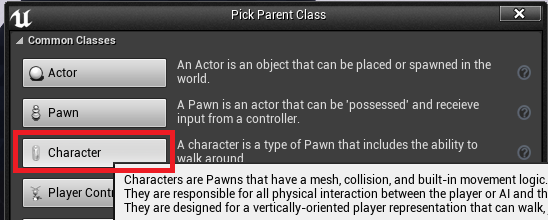
Arrastramos dentro todos los ficheros.

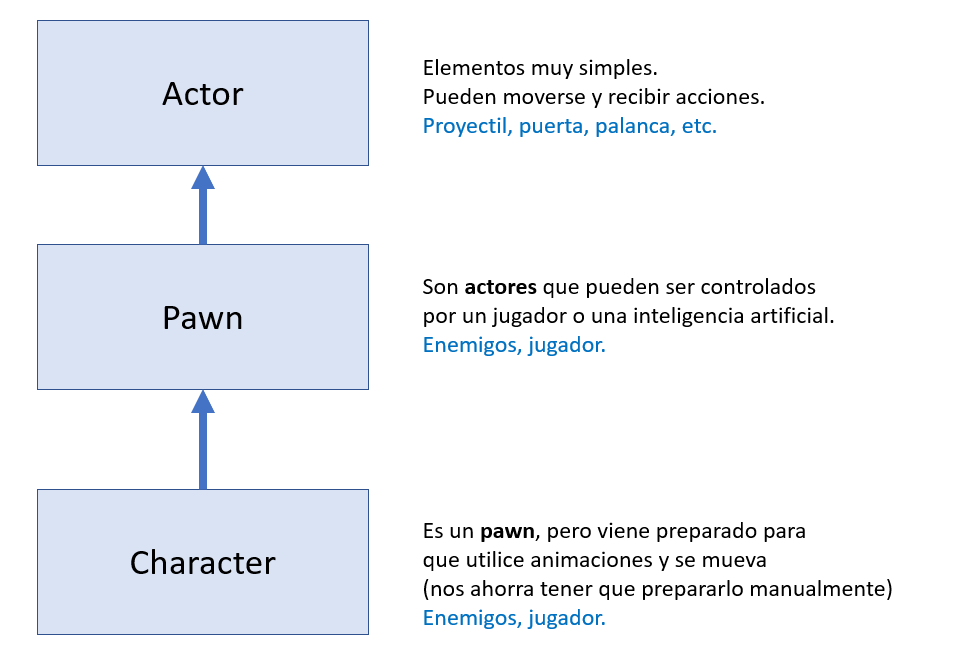


En la propia carpeta **Jugador** creamos una nueva **BlueprintClass** (botón derecho - > BlueprintClass)



Será de tipo **Character**





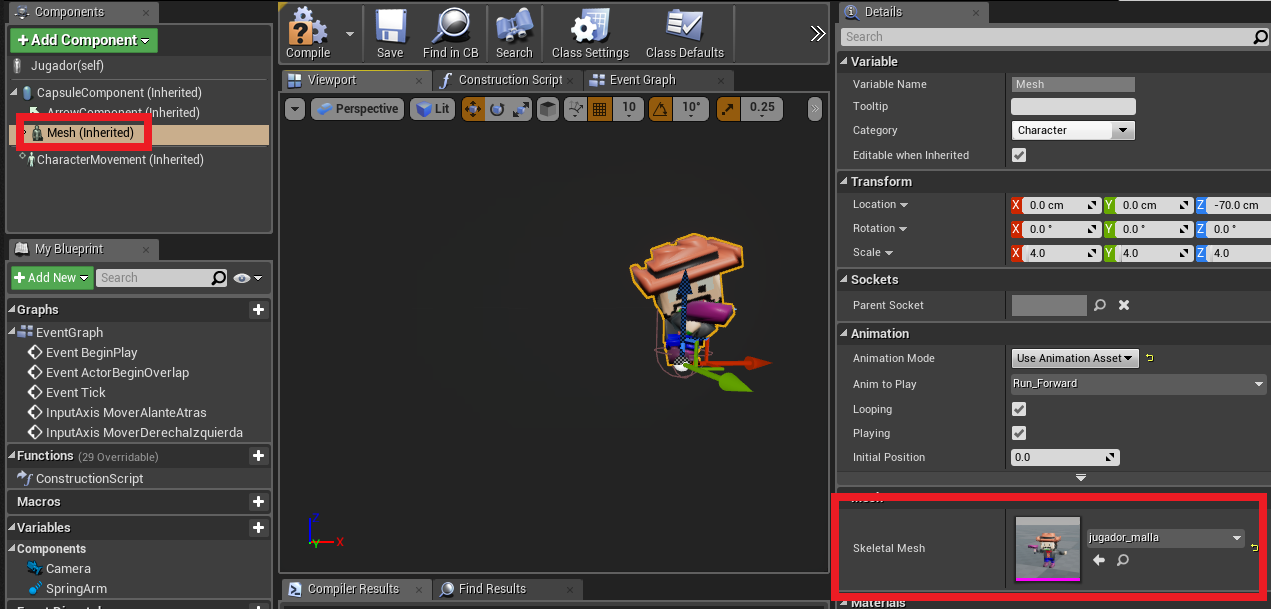
La renombramos a **Jugador.**



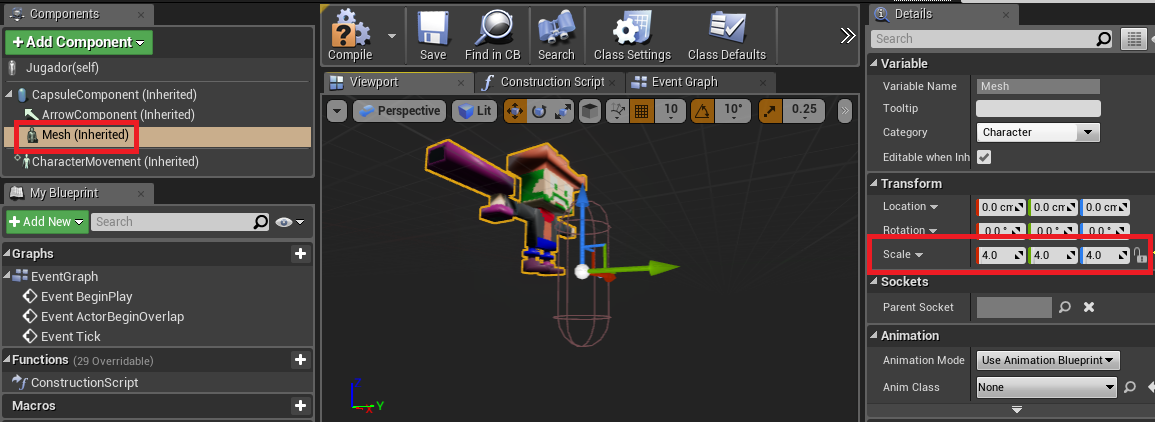
Accedemos a **Jugador** y por el momento únicamente configuramos su **malla (Mesh)**.

Desde la pestaña **Viewport** seleccionamos, el componente **Mesh,** cambiamos las siguientes propiedades:

* **Skeletal Mesh** seleccionamos **jugador\_malla**

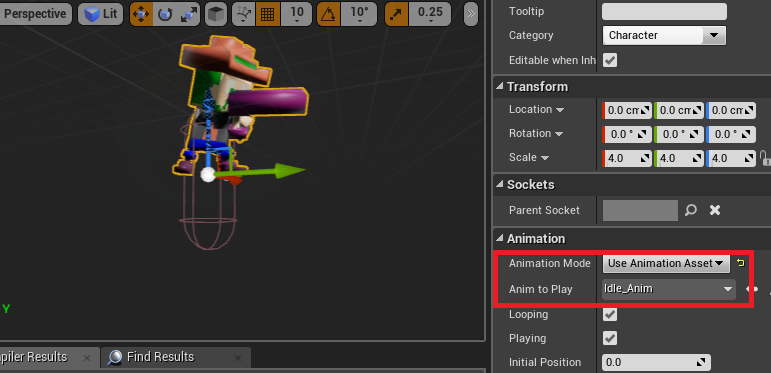


* **Scale**, la malla es muy pequeña, cambiamos el factor a **3 en todos los ejes**

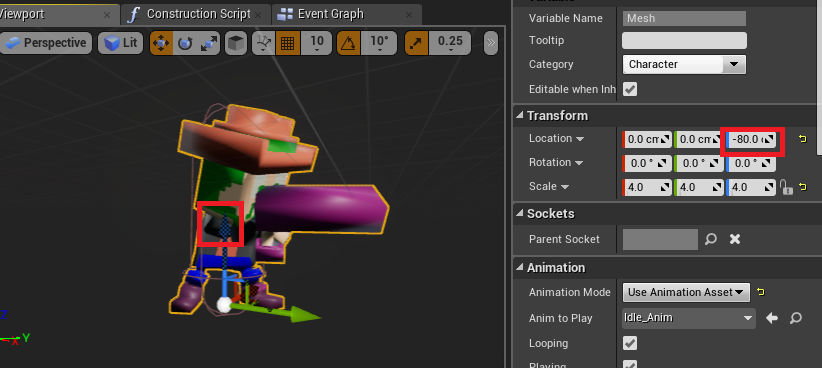


Vamos a probar una animación.

* **Animaction Mode:** Use Animation Asset (usar una animación fija)
* **Anim to Play:** seleccionar la animación que queremos



**Movemos la malla para que encaje dentro del capsule componet.** Movemos la malla para que más o menos dentro de la capsula de colisión.

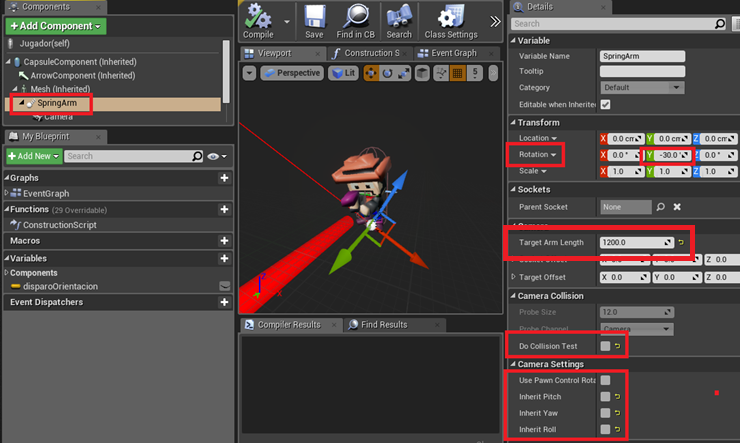


Utilizamos las flechas para colocar al jugador en el medio de la jaula de colisión.

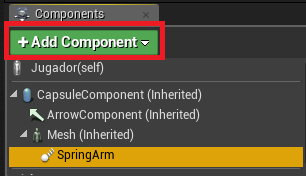
Nos falta añadir un componente **Spring Arm y** una **Camera**. Pulsamos Sobre **+Add Component** y colocamos una **SpringArm** (este componente hace de soporte para la cámara)



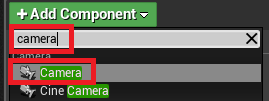
* Tenemos que agregar el componente **Spring Arm** como hijo de la Mesh
* Lo rotamos para que apunte al cielo (la cámara nos grabara desde lo alto)
* Cambiamos la longitud **Target Aim Lenght** a **1200**
* Cambiamos la **rotación** en el eje y a **-50**
* Sí quisiéramos podríamos desactivar el **Do Collision Test**, para que la cámara no haga zoom cuando algun objeto se interponga en la línea de visión.
* **Importante:** Desactivamos todos los parámetros de rotación del **Camera Settings**

****

Para agregar la camará marcamos el **SpringArm** y pulsamos en **+ Add Component**.



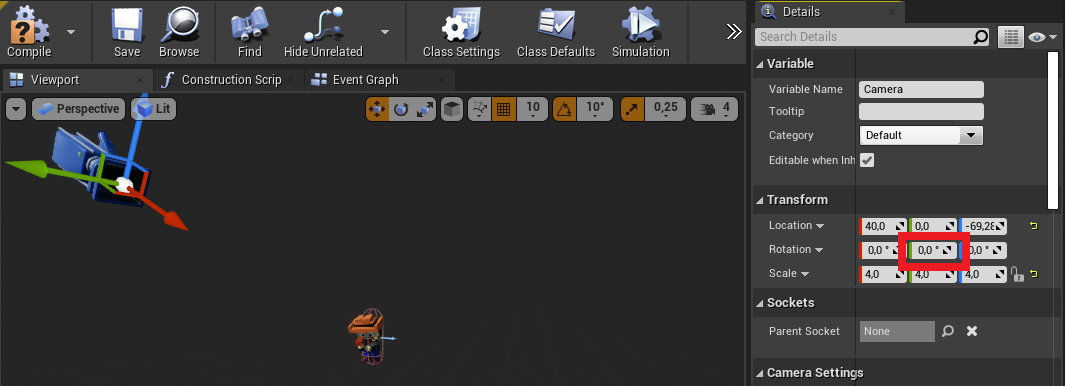
Buscamos el elemento **Camera**



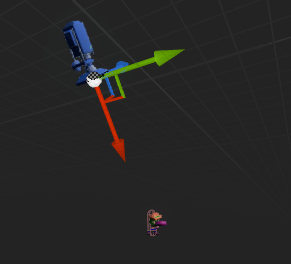
Importante **Camera** debe ser hijo del **SpringArm.** Hacemos que la cámara grabe desde el aire y apunte hacia el suelo (la rotamos hacia abajo).



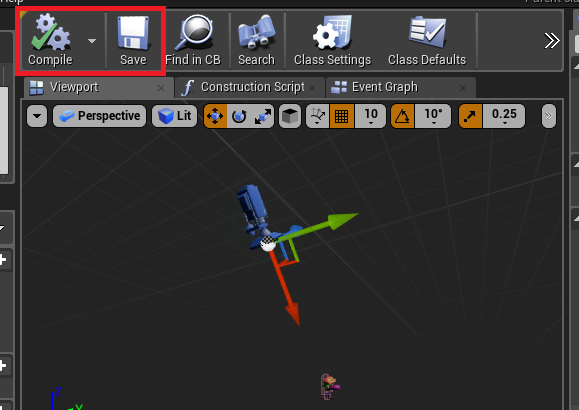
Es posible que haya que cambiar **la rotación de la cámara** , para que apunte al jugador



La cámara debe enfocar al jugador

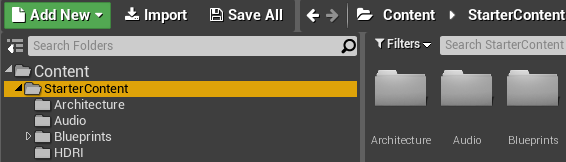


Más adelante implementaremos la funcionalidad del **jugador**, por el momento solo queremos que se vea en pantalla.

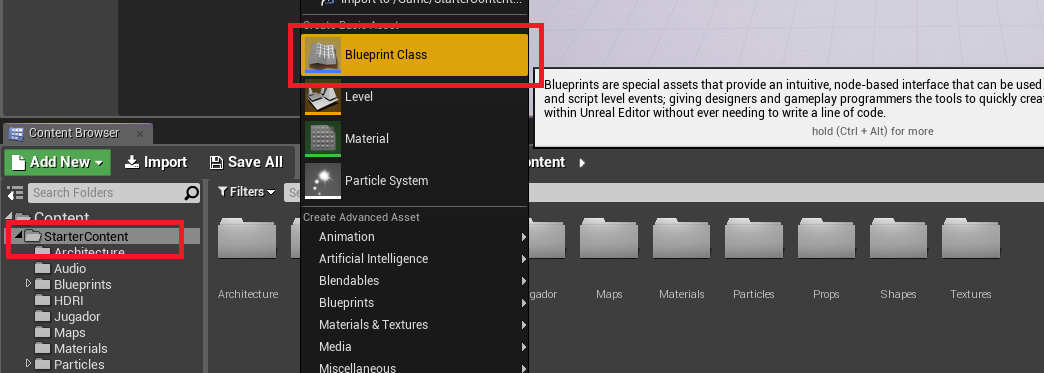


## Modo de Juego

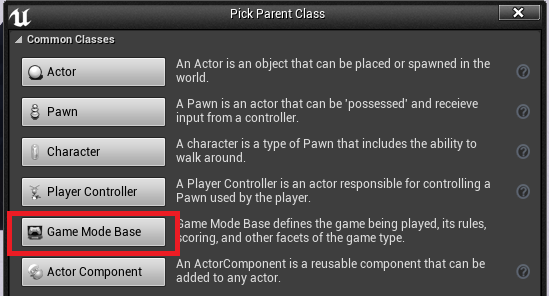
Accedemos la carpeta **StarterContent**



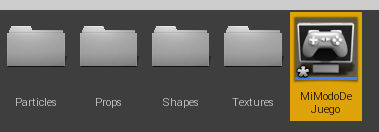
Declaramos una nueva **Blueprint class**.



Será de tipo **Game Mode Base** (un modo de juego)

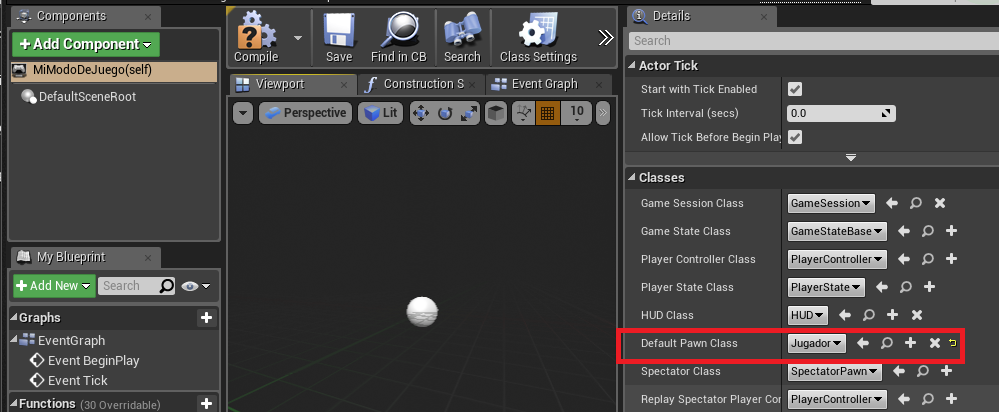


Renombramos el fichero a **MiModoDeJuego**.



|  |
| --- |
| **Modo de Juego:** recordamos que es una clase que se instancia cada vez que se abre un nivel, el código incluido en ella puede ser fácilmente accedido desde cualquier parte del proyecto a través del método **getGameMode** |

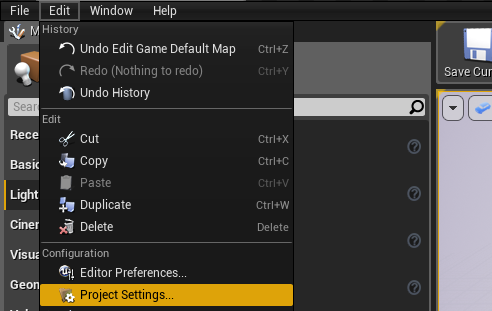
Abrimos el fichero **MiModoDeJuego** y configuramos cual será el **Default Pawn Class**, peón principal del juego.



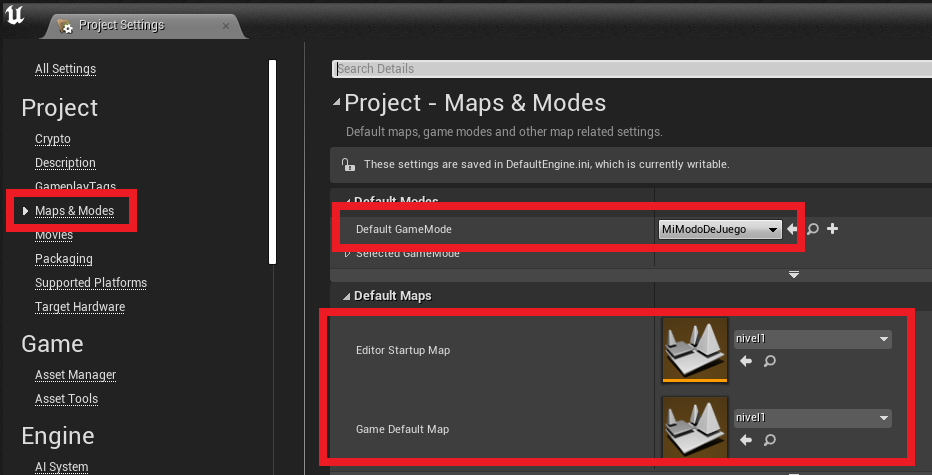
Compilamos y guardamos los cambios.



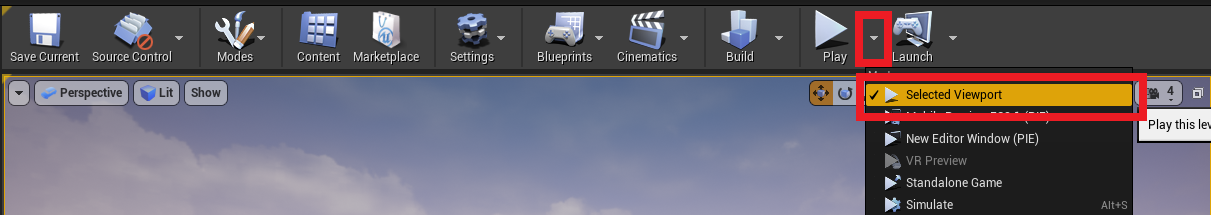
Antes de probar el juego solo falta decirle que utilice **MiModoDeJuego** y el **nivel1** al arrancar el juego. **Edit -> Project Settings**



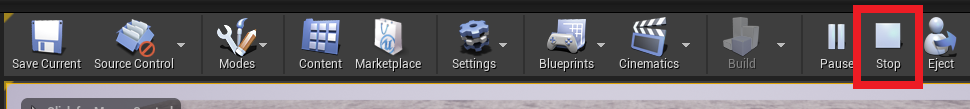
Accedemos a propiedades del proyecto **Maps & Modes**, y establecemos el modo de juego y el nivel.



Probamos a ejecutar el juego.

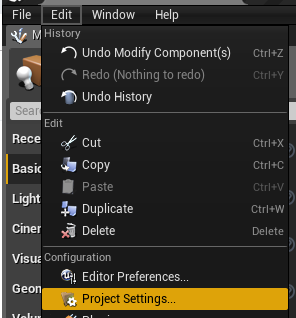




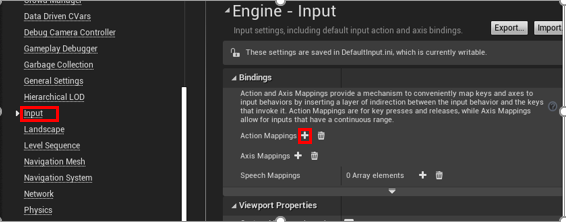
Parar la ejecución del juego, pulsando la **tecla escape**, o el botón **stop** del editor.  
  


## Declaración de mecanismos de entrada (Inputs)

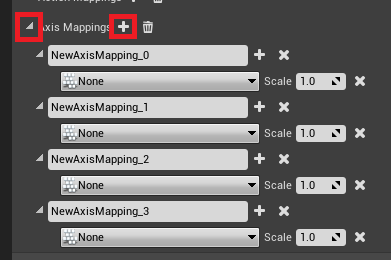
Declaramos los mecanismos de entrada desde **Edit -> Project settings**



Definimos las siguientes **Action Mappings** y **Action Axis,** el juego se va a poder controlar desde Gamepad.

Derecha

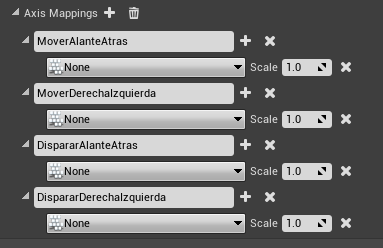
Declaramos 4 Axis Mappings



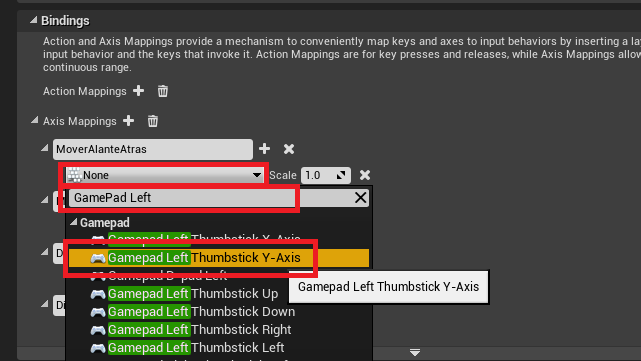
Tendrán los siguientes nombres.

* MoverAlanteAtras
* MoverDerechaIzquierda
* DispararAlanteAtras
* DispararDerechaIzquierda

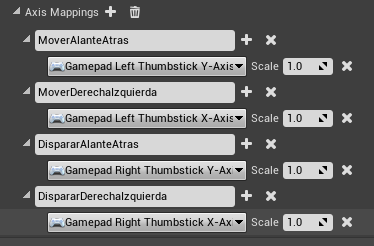
En cada uno vamos a colocar la acción correspondiente del mando.



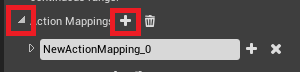
Comenzamos configurando **MoverAlanteAtras**, debemos seleccionar el   
**GamePad Left Thumbstick Y-Axis** (Usar el cuadro de búsqueda para encontrar la opción más rápido)



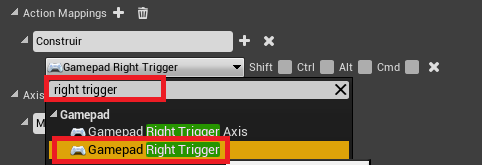
Seleccionamos el modo de control para el resto de los eventos de entrada, tal y como se ve en la siguiente fotografía.



Ahora vamos a agregar una acción.



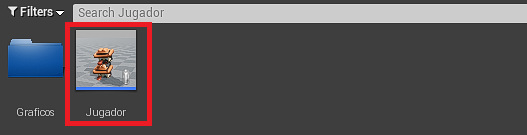
La llamaremos **Construir** y se lanzará con el **Gamepad Right Trigger.**



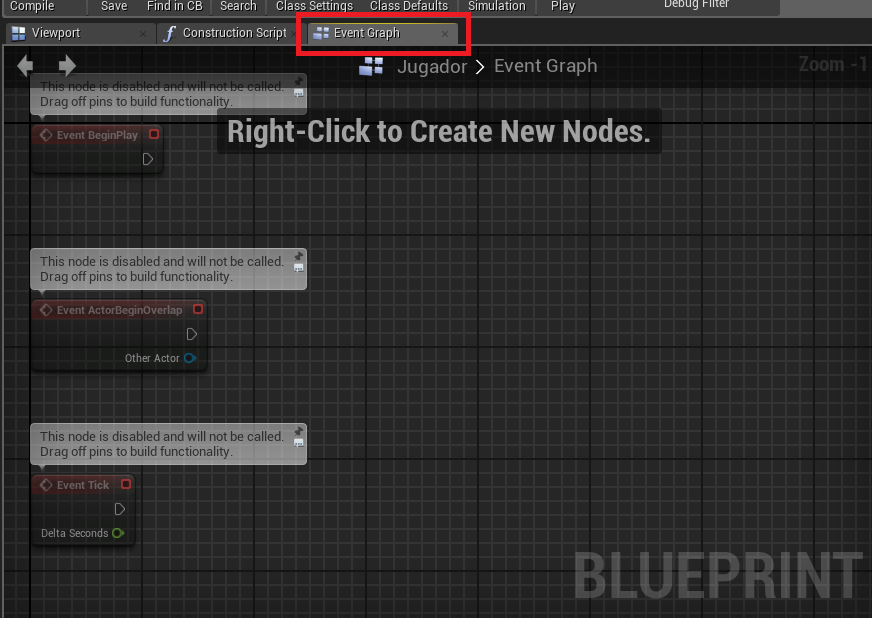
|  |
| --- |
| **Extra – control por teclado**  En el juego que desarrollaremos en próximas semanas utilizaremos el teclado y el ratón en lugar del mando.  Sí en este punto alguien quiere incluir que se juego pueda ser controlado con teclado puede utilizar la siguiente configuración:   * Letras W y S para MoverAlanteAtras * Letras D y A para MoverDerechaIzquierda * Flechas Up y Down para DispararAlanteAtras * Flechas Right y Left para DispararDerechaIzquierda * Barra espaceadora para Construir.   Se pueden agregar mas inputs a cualquier acción pulsando en    El resultado final tiene que ser el siguiente , Fijarse bien en que las letras tienen diferente Scale 1 y -1 . |

## Jugador - Movimiento

Accedemos a la clase **Jugador**,



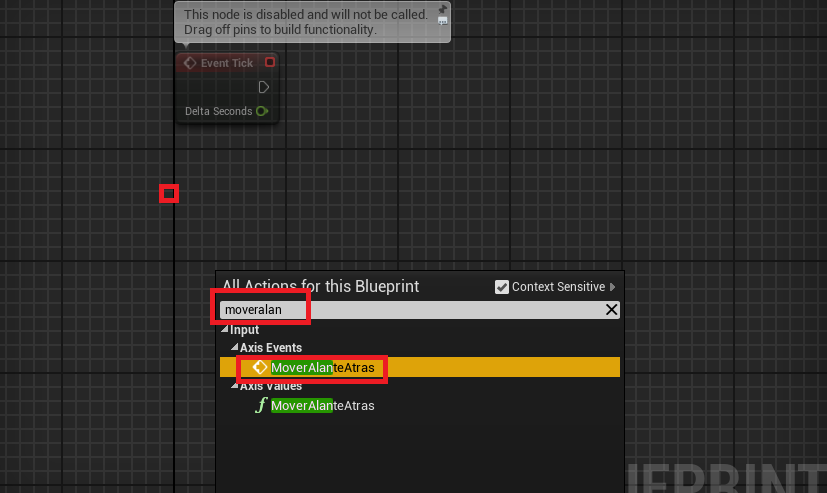
Abrimos la pestaña de **Event Graph.**

****

**Vemos en semitransparente el inicio de los eventos:**

* **Event BeginPlay –** lo que colocamos a continuación se ejecuta una sola vez al iniciar el juego
* **Event ActorBeginOverlap –** se ejecuta cada vez que el jugador se superpone con cualquier otro elemento del juego. El atributo  contendría una referencia al actor contra el que se produce la superposición.
* **Event Tick –** lo que colocamos a continuación se ejecuta en cada FP / frame del juego.

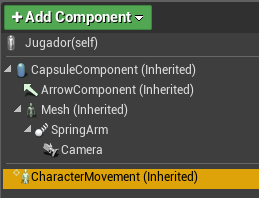
Buscamos una superficie libre, pulsamos botón derecho y buscamos el evento **MoverAlanteAtras**



Seleccionar el evento NO la función (trae una F)

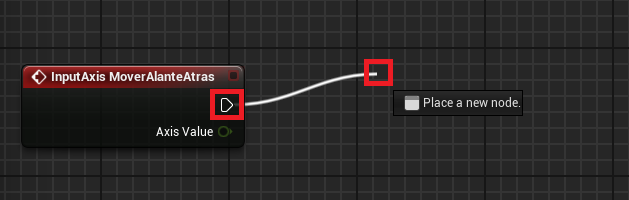
Este evento se va a ejecutar en cada FP del juego y nos dará como resultado el valor del Joysticik que controla el eje 

El Jugador, tiene un componente **CharacterMovement** que trae ya implementado todo el motor de movimiento.

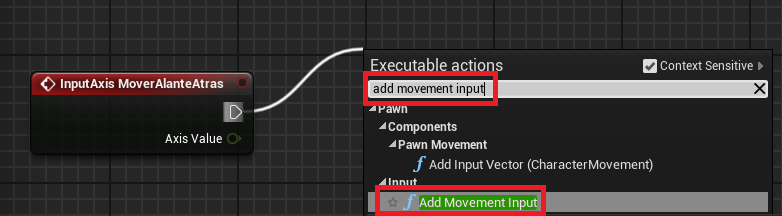


Gracias a este componente podemos hacer uso dentro del jugador de funciones como **AddMovement Input** (la cual le agrega un movimiento).

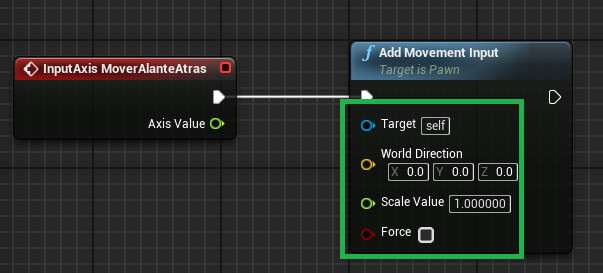
Sacamos un flujo de ejecución del evento **MoverAlanteAtras.**



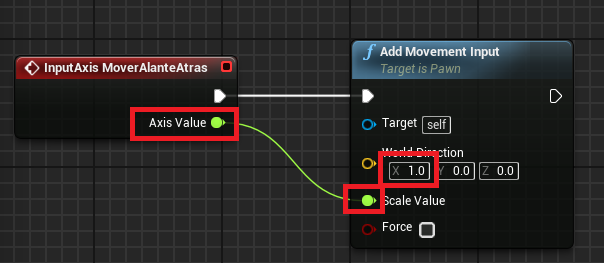
Buscamos la función **AddMovement Input.**



Todos los valores de la parte izquierda de la función son parámetros de entrada (Unreal suele darles un valor por defecto el cual podemos modificar).



* **Target**, es el objeto que queremos mover, Self es una referencia igual al This en otros lenguajes
* **World Direction**, es el vector que marca la línea de movimiento
* **Scale Value**, es la orientación del movimiento dentro de esa línea.

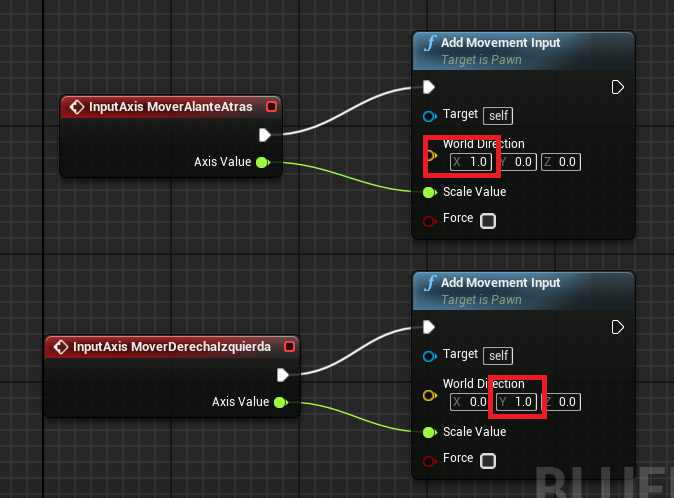


Es posible que pensásemos que la dirección para movernos Adelante y atrás era Y=1 (en lugar de X=1) , pero si nos fijamos en la esquina inferior izquierda del mapa vemos que X e Y, no están orientadas como creíamos.





Repetimos el mismo proceso y creamos el evento **MoverDerechaIzquierda,** el cual llamará a la función **Add Movement Input**, la única diferencia es que está nos moverá en el eje Y.



Compilamos y guardamos los cambios de la clase **Jugador**.



Probamos a ejecutar el juego y a mover el jugador.  