UJI Players

Año 2020, Número 1, Artículo 1

Introducción a *UnrealEngine 4* -Creando un personaje en primera persona con C++

Juan José Gómez Simón

Resumen

En este artículo haré una pequeña introducción a *UnrealEngine 4* mediante la explicación y creación de un personaje en primera persona que se mueva por un entorno 3D y pueda mover la cámara libremente.

Para ello, se necesita saber previamente varios conceptos con respecto a la programación orientada a objetos y C++ tales como:

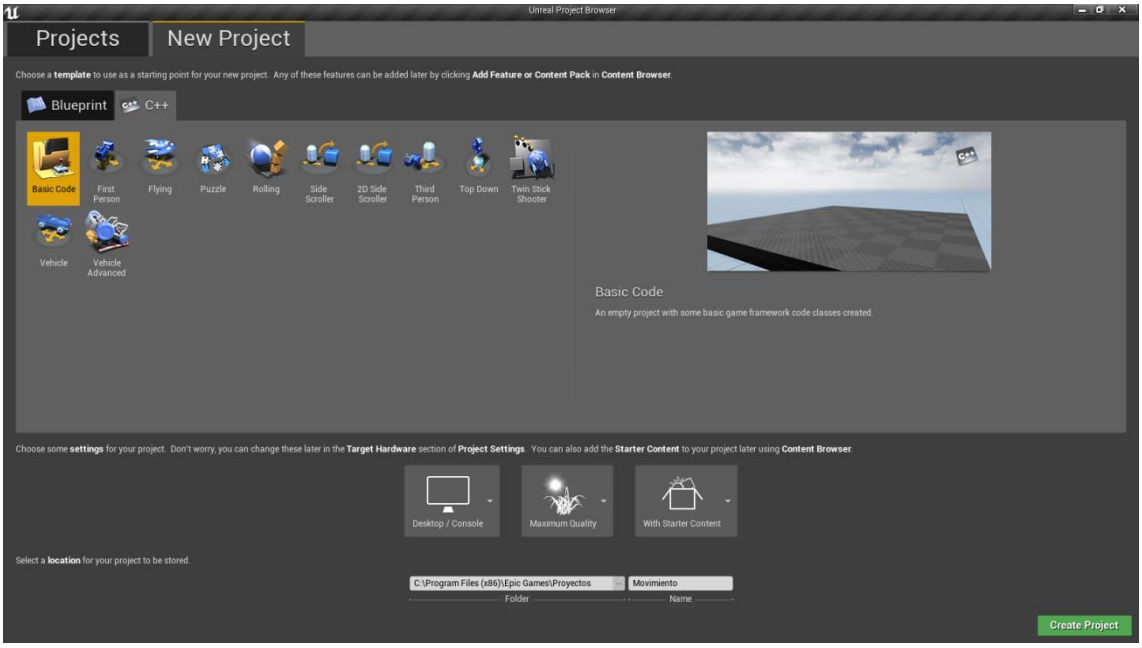
- Clases: https://www.masqueteclas.com/articulo/el-concepto-de-clase/
- Herencia: https://es.ccm.net/contents/411-poo-herencia
- Visibilidad: https://victorroblesweb.es/2016/06/11/php-poo-visibilidad-public-protected-private/ (este articulo es del lenguaje PHP pero conceptualmente es lo mismo para todos los lenguajes)
- Constructores:https://www.ecured.cu/Constructor_(C%2B%2B)
- Archivos de cabecera:https://docs.microsoft.com/es-es/cpp/cpp/header-files-cpp?view=vs-2019 (resumidamente dice que el código se divide en dos ficheros: el .h(header) y el .cpp(el fuente), el .h tiene las declaraciones de variables, funciones etc y el .cpp sus inicializaciones y definiciones)
- Punteros: https://www.programarya.com/Cursos/C++/Estructuras-de-Datos/Punteros

También antes de comenzar cabe mencionar dos cosas; en primera instancia para programar necesitarás *VisualStudio* o *VisualStudioCode* y en segunda instancia cabe decir que estoy usando *UnrealEngine 4.14.3* y *VisualStudio2015*debido a que mi ordenador no es muy potente.

Creación del proyecto

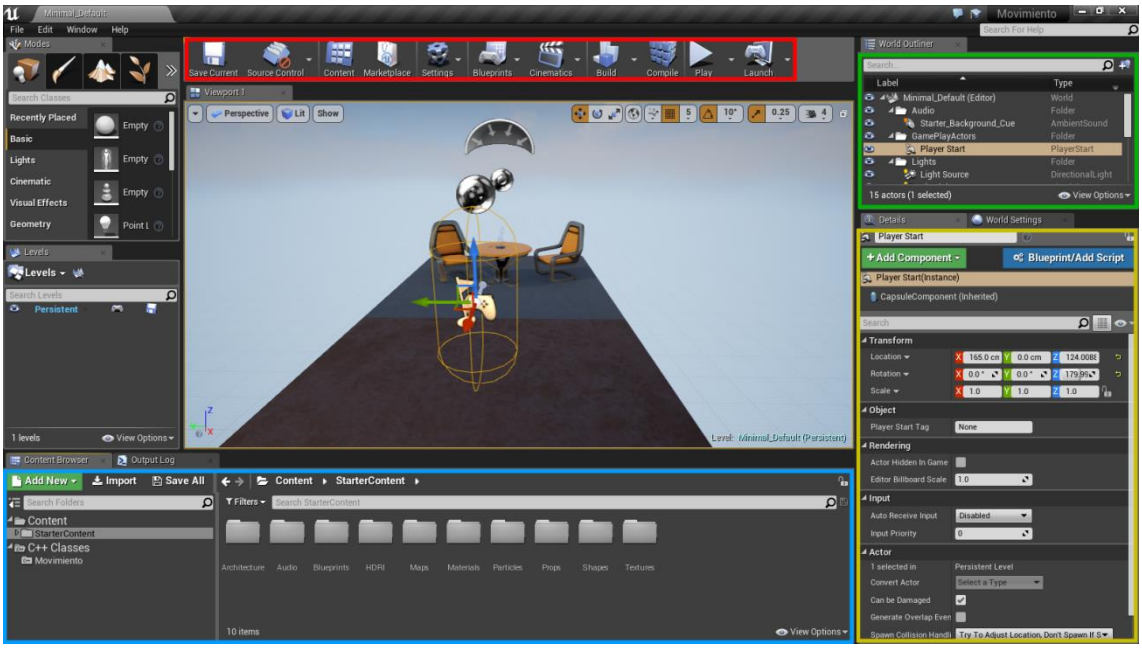
Antes de nada, lo primero que hay que hacer es crear un proyecto nuevo, como se puede apreciar en la imagen, se pueden crear dos tipos de proyectos, los basados en BluePrints(ignoraremos este concepto de momento) y los que están basados en C++(elegiremos este tipo).

Y finalmentepara este artículo, con el fin de enseñar a hacer un personaje desde cero, se elegirá una plantilla vacía(*Base Code*).Sin embargo, como se puede ver, hay gran variedad de proyectos ya creados para tener una pequeña base.



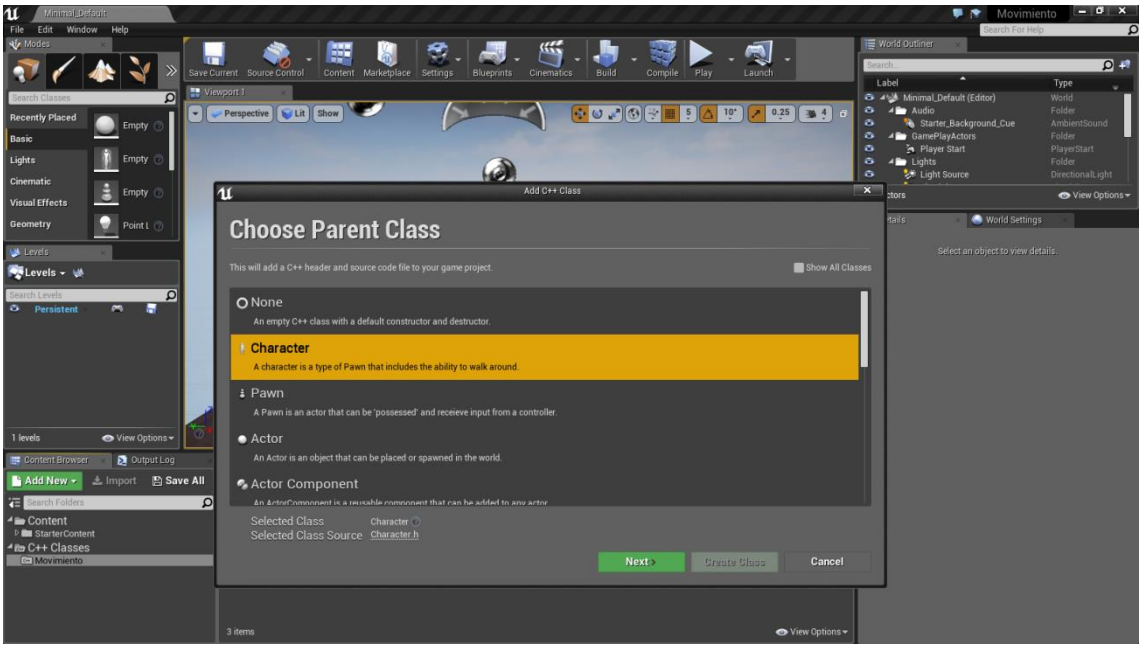
Primeros pasos

Una vez creado el proyecto lo primero que mostrará es esta ventana de la cual solo nos interesa cuatro partes:



- La zona roja es la barra de herramientas de las cuales solo usaremos "Settings"(para ver las propiedades del proyecto), "Compile"(para compilar el código),"Blueprints" y "Play"(para ir probando el juego).
- La zona azul es el "Navegador de Contenido" que es donde crearemos y guardaremos todos nuestros "assets"(modelados, texturas, sonidos, clases, objetos, etc).
- La zona verde muestra todos los objetos que hay colocados en escena y al clicar sobre uno de ellos, en la zona amarilla aparecerá sus propiedades.

Bien, para comenzar a crear nuestro personaje lo primero que tenemos que hacer es ir al "Navegador de Contenido " y dirigirnos a la carpeta "C++ Classes" (si no te aparece pulsa en el ícono que hay a la izquierda debajo de "Add New"), después abrir la única carpeta que hay(que se llamará como nuestro proyecto, en mi caso "Movimiento"). Una vez dentro, pulsamos click derecho y seleccionamos la opción "New C++ Class" y aparecerá esta ventana:



De aquí seleccionaremos que nuestro personaje herede de la clase "Character" ya que esta posee un componente fundamental para el movimiento llamado "CharacterMovement", el cual tiene parámetros fundamentales tales como la velocidad del jugador, su inercia, la fricción etc.

Cabe decir que también podemos elegir la clase "Pawn"(que es una clase básica que lee inputs) pero no posee toda esa información que nos ofrece el "CharacterMovement". Sin embargo, eso puede ser una ventaja si lo que buscas es otro tipo de movimiento, ya que te da más libertad a la hora de programar.

Nota: He llamado a mi objeto Jugador.

Hora de programar

Una vez que ha cargado Visual Studio lo primero que haremos será ir al .h y añadir debajo de `GENERATED_BODY()`el siguiente código:

```
protected:

///Se define los componentes del jugador
//1.Camara del jugador
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Componentes jugador")
	UCameraComponent* camaraJugador;

//2.SpringArm para un movimiento fluido de la camara
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Componentes jugador")
	USpringArmComponent* springArmCamara;

//3.SkeletalMesh, sera el "avatar" del jugador
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Componentes jugador")
	USkeletalMeshComponent* aspectoJugador;
```

1. El primer componente es la cámara del jugador que se encargará de mostrar por pantalla todo lo que sucede en la escena.
2. El segundo componente está relacionado con la cámara, lo que hace es un movimiento de cámara más fluido, es decir cuando el jugador se desplaza lo que hace es darle un retardo al movimiento de la cámara para que vaya un poco por detrás del jugador.
3. Este componente es para que después se seleccione un modelado y que además se pueda animar.

Nota:El UPROPERTYes para indicar que puede ser visible en el editor del motor para que así si queremos modificar una propiedad del jugador (como por ejemplo el FOV de la cámara o la velocidad del jugador) lo podamos hacer sin necesidad de tocar el código ni de compilar nada.

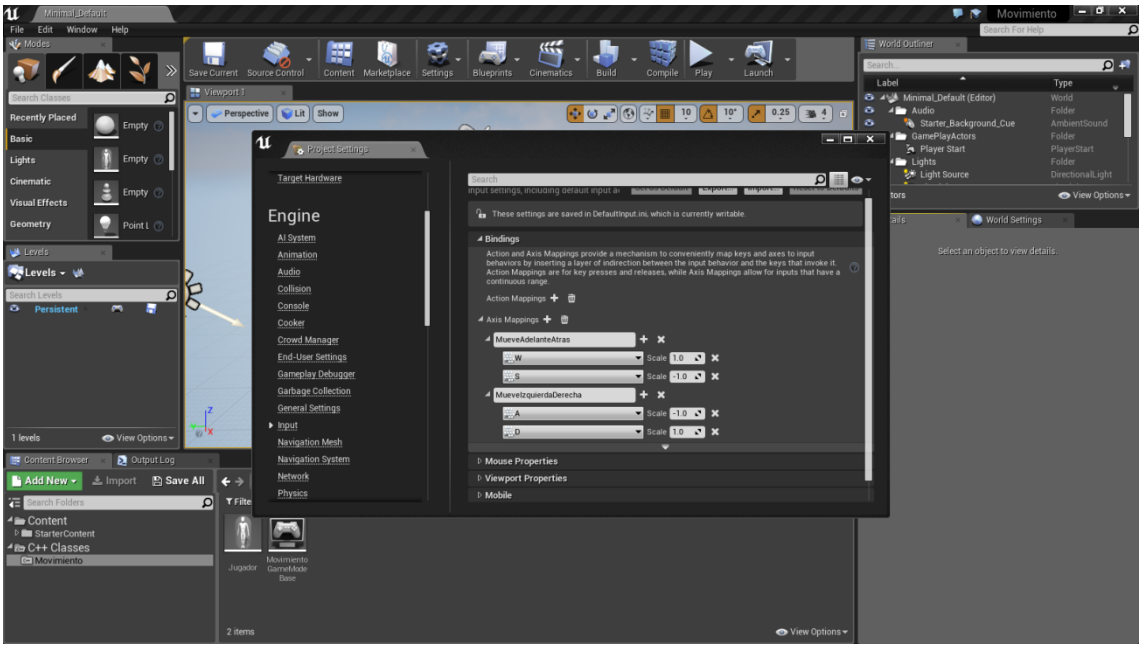
Después, tras definir los componentes lo que haremos será ir al .cppy en el constructor poner este código(sustituyendo`PrimaryActorTick.bCanEverTick = false;`):

```
// Activa o desactiva el evento Tick(evento que se llama 30 veces por fotograma),por defecto esta en True pero se recomienda desactivarlo
// a no ser de un caso muy excepcional.
PrimaryActorTick.bCanEverTick = false;

///Se crea los componentes del jugador, o mejor dicho se inicializan
//Camara
CamaraJugador=      CreateDefaultSubobject<UCameraComponent>(TEXT("Camara      del jugador"));

//SpringArm
springArmCamara = CreateDefaultSubobject<USpringArmComponent>(TEXT("SpringArm de la camara"));

//SkeletalMesh
aspectoJugador = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("Aspecto del jugador"));

camaraJugador->bUsePawnControlRotation = true;

///Une los componentes al objeto o mejor dicho creamos la jerarquia de componentes que tendra el jugador

//Unimos a este componente raiz tanto el "avatar" del jugador y el  springArm
aspectoJugador->SetupAttachment(RootComponent);
springArmCamara->SetupAttachment(RootComponent);

//Unimos la camara al springArm para que se aplique ese efecto de fluidez explicado previamente
camaraJugador->SetupAttachment(springArmCamara);
```

Perfecto, hasta hora lo que tenemos hecho son los componentes que formarán el jugador(el avatar, la cámara, la colisión(esta ya viene incluida porque hemos heredado de la clase "Character") etc.).Ahora, lo que queda es que se desplace y que se pueda rotar la cámara libremente.

Desplazamiento del jugador

Lo primero que tenemos que hacer es volver a *UnrealEngine* e ir a la barra de herramientas, apartado "Settings" y dentro ir "Engine"->"Inputs", dejando una ventana tal que así:



Lo que haremos será añadir dos "Axis Mappings" que serán los grupos de teclas que llamarán a los correspondientes métodos para desplazarse tanto hacia adelante/atrás como izquierda/derecha. Y las llamaremos como queramos, dentro de cada grupo pondremos dos teclas con un valor de 1 y -1 siendo 1 ir hacia delante(o hacia la derecha) y -1 ir hacia atrás(o izquierda), estos valores nos servirán para más adelante.

Una vez hecho esto, volvemos a *Visual Studio*, más concretamente al .h y debajo del `GENERATED_BODY()` pondremos el siguiente código:

```
private:

	//Metodo para mover al jugador hacia adelante o hacia atras
	voidmoverAdelanteAtras(float velocidad);

	//Metodo para mover al jugador hacia la izquierda o hacia la derecha
	voidmoverIzquierdaDerecha(float velocidad);
```

Después, nos dirigiremos al .cpp y dentro del método"SetupPlayerInputComponent" (que se encarga de controlar los eventos sobre los inputs del jugador) pondremos el siguiente código:

```
//Al pulsar W o S se llamara a la metodomoverAdelanteAtras
PlayerInputComponent-
>BindAxis("MueveAdelanteAtras",this,&AJugador::moverAdelanteAtras);

//Al pulsar A o D se llamara a la metodomoverAdelanteAtras
```

```
PlayerInputComponent->BindAxis("MueveIzquierdaDerecha",this,
&AJugador::moverIzquierdaDerecha);
```

Nota:"MueveAdelanteAtras"es el nombre de mi grupo de teclas para desplazarme en el eje X.

Nota2: &AJugador::es la referencia del objeto que llamará la función, aquí debe estar &ANombreDelObjetoQueLeHasPuesto::con la A incluida muy importante por temas de herencia.

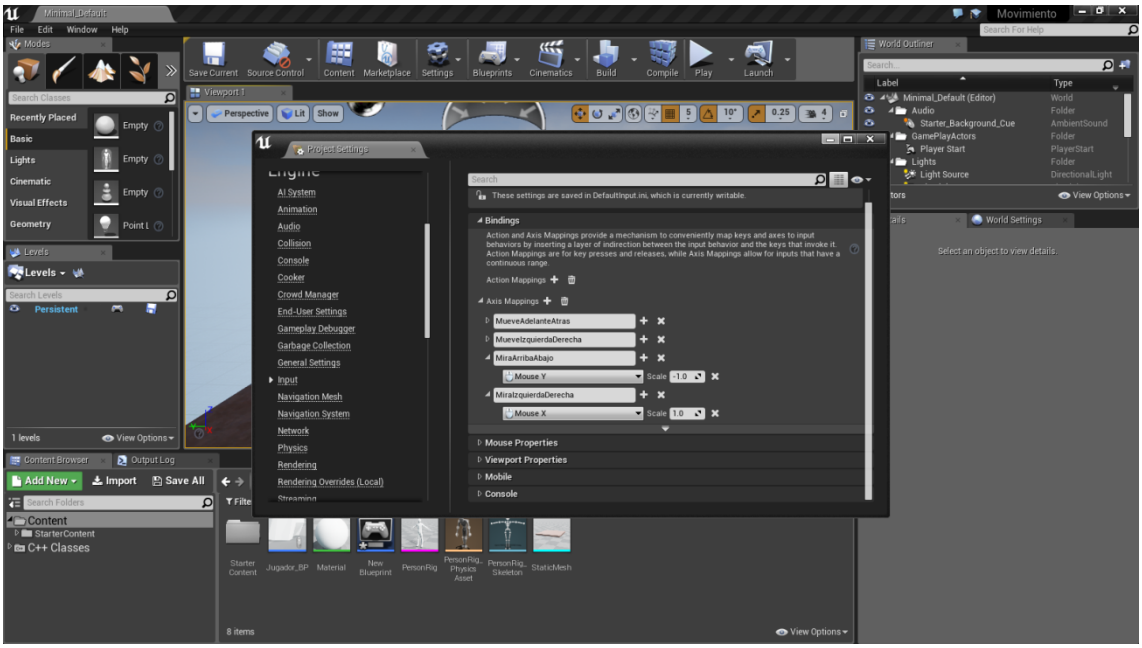
Finalmente, iremos al final del archivo .cpp y pondremos el siguiente código que son los métodos para desplazar al jugador:

```
//Mover al jugador de adelante y hacia atras
voidAJugador::moverAdelanteAtras(floatvelocidad) {
//Si el componente que contiene todos los parametros de velocidad y movimiento
existe y el jugador no esta parado...
	if (Controller != NULL&&velocidad != 0.0f) {
	//...entonces se desplaza hacia adelante o hacia atras dependiendo del
parametroscale visto en ProyectSettings->Input
		AddMovementInput(GetActorForwardVector(), velocidad);
		//GEngine->AddOnScreenDebugMessage(-1,	15.0f,	FColor::Red,
FString::Printf(TEXT("Adelante")));
	}
}

//Metodo para mover al jugador hacia la izquierda o hacia la derecha
voidAJugador::moverIzquierdaDerecha(floatvelocidad) {
	//Si el componente que contiene todos los parametros de velocidad y
movimiento existe y el jugador no esta parado...
	if (Controller != NULL&&velocidad != 0.0f) {
		//...entonces se desplaza hacia la izquierda o derecha dependiendo
del parametroscale visto en ProyectSettings->Input
		AddMovementInput(GetActorRightVector(), velocidad);
		//GEngine->AddOnScreenDebugMessage(-1,	15.0f,	FColor::Red,
FString::Printf(TEXT("Atras")));
	}
}
```

Nota: La expresiónAJugador::brevemente explicado es para evitar conflictos con los nombres de los métodos, a esto se denomina "scoperesolution" y cabe decir que se llama Jugadorporque así se llama el objeto.

Movimiento de la cámara

Hay poco que decir al respecto, es exactamente el mismo proceso que el previamente explicado:



En el .h:

```
//Metodo para mover la camara hacia arriba o hacia abajo
voidmoverCamaraArribaAbajo(floatrotacion);

//Metodo para mover la camara hacia la izquierda o derecha abajo
voidmoverCamaraIzquierdaDerecha(floatrotacion);
```

En el .cpp, en el método SetupPlayerInputComponent:

```
	//Al mover el mouse arriba o hacia abajo mueve la camara
	PlayerInputComponent->BindAxis("MiraArribaAbajo", this,
&AJugador::moverCamaraArribaAbajo);

	//Al mover el mouse izquierda o derecha mueve la camara
	PlayerInputComponent->BindAxis("MiraIzquierdaDerecha", this,
&AJugador::moverCamaraIzquierdaDerecha);
```

Al final del archivo .cpp:

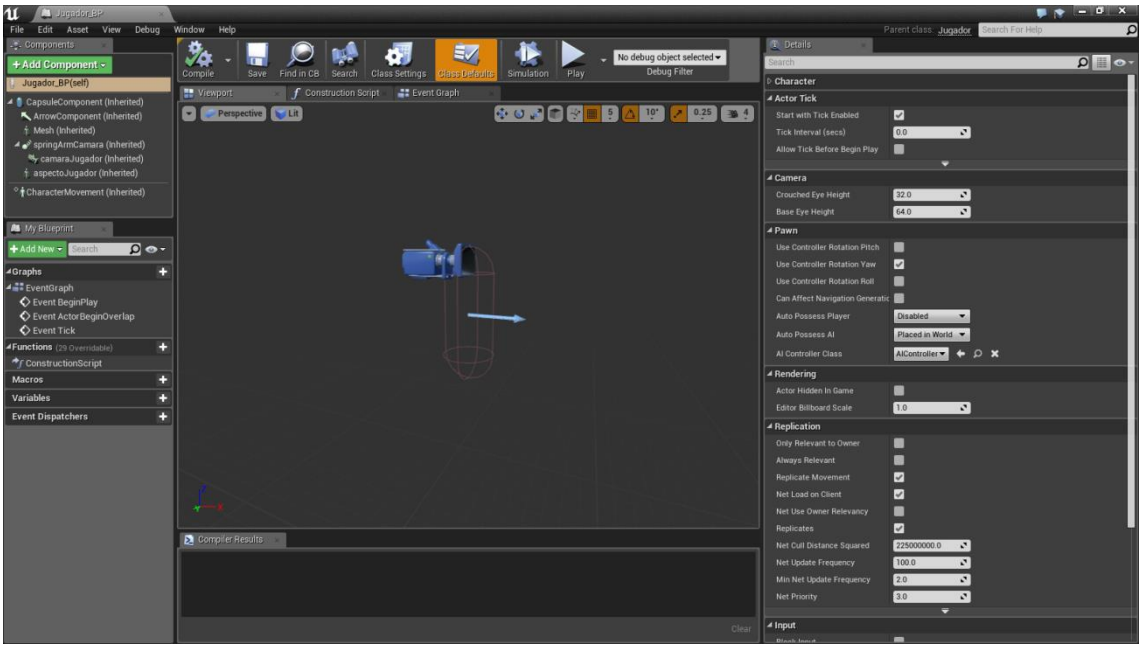
```
//Metodo para mover la camara del jugador arriba o abajo
voidAJugador::moverCamaraArribaAbajo(floatrotacion) {
	//Si el componente no es nulo mueve la camara, este componente tambien
tiene parametros relacionados con la camara
	if (Controller != NULL) {
		//Mueve la camara aplicando el scale para diferenciar de arriba
hacia abajo
		AddControllerPitchInput(rotacion);
		//GEngine->AddOnScreenDebugMessage(-1,	15.0f,	FColor::Red,
FString::Printf(TEXT("Arriba")));
	}
}

//Metodo para mover la camara del jugador de derecha a izquierda
voidAJugador::moverCamaraIzquierdaDerecha(floatrotacion) {
```

```
	//Si el componente no es nulo mueve la camara, este componente tambien tiene parametros relacionados con la camara
	if (Controller != NULL) {
		//Mueve la camara aplicando el scale para diferenciar de derecha hacia izquierda
		AddControllerYawInput(rotacion);
		//GEngine->AddOnScreenDebugMessage(-1, 15.0f, FColor::Red, FString::Printf(TEXT("abajo")));
	}
}
```

Retoques finales

Para acabar, volvemos a *UnrealEngine* y pulsaremos la opción "Compile" y si todo ha salido bien nos compilará correctamente nuestro objeto.

Tras esto, pulsaremos click derecho sobre nuestro nuevo objeto y seleccionaremos la opción de "CreateBluePrintclassbasedon " seleccionaremos una ruta, guardamos y con esto hemos conseguido un entorno más gráfico en el que podemos editar los parámetros y propiedades de los componentes y variables que hayamos hecho públicos sin necesidad de editar código ni compilar.

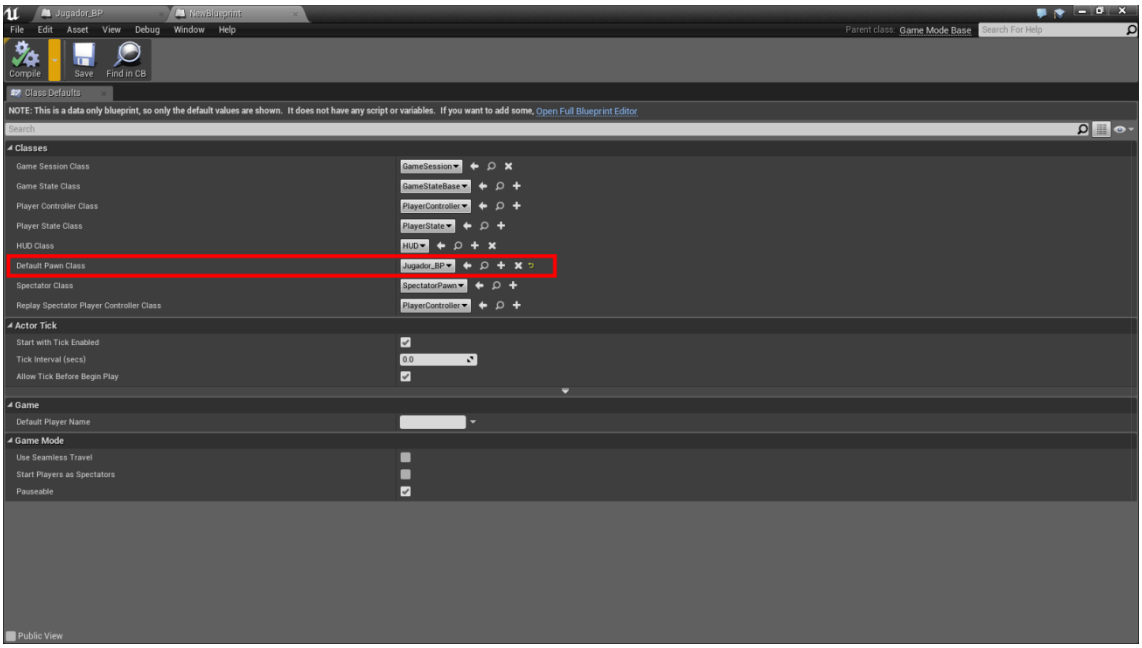


Nota: Si nos dirigimos al "SpringArm" y marcamos la opción "Enable Camera Lag" podréis ver a lo que me refería con movimiento más fluido y suave.

Nota2: Si le damos de nuevo al "SpringArm" y modificamos "Target ArmLength" podremos modificar la distancia de la cámara respecto al jugador.

Finalmente, queda colocar en escena nuestro jugador.Sin embargo, no será tan simple como cogerlo y arrastrarlo a escena(que se puede hacer pero es una mala práctica). Lo primero que haremos será ir al "Navegado de Contenido", pulsar click derecho y

seleccionar la opción "BlueprintClass", después se selecciona la clase "GameMode Base". Una vez creado lo abrimos y en la opción que aparece señalada en la imagen a continuación, colocamos nuestro objeto creado (pero la versión BluePrint):



Finalmente, nos dirigimos a la barra de herramientas y seleccionamos la opción Blueprints->GameMode: ->SelectGameModeBase->El Blueprint que acabamos de crear.

Con esto, hemos conseguido que el objeto seleccionado en el parámetro "Default PawnClass" aparezca en escena,¿dónde? bueno en el "PlayerStart" que hay en el nivel y que se puede apreciar en la segunda imagen.

Y con esto se acabó, espero que te haya servido y muchas gracias por leerme.

Si quieres preguntar algo puedes hacerlo vía Twitter(@GJuanjo1999), e-mail o por mi página web(http://juanjosegomez.x10host.com/).

PD: Dejaré los dos ficheros para descargar aquí:

https://drive.google.com/file/d/1g2FTb5gvBk-Vc-vmR1m_wasYT2rUKKz8/view?usp=sharing

https://drive.google.com/file/d/1mjM7drXvo9W7jaAb9boweWqDq4gVMev3/view?usp=sharing