**PlayerController**: acciones/input

**PlayerState**: atributos del jugador que no forman parte del personaje que controla.

* Vida

Kills por ejemplo no. Pq al morir las kills siguen siendo las mismas

**GameState**: podría incluir estadísticas de todos los players.

**Pawn**: es un objeto que puede ser controlado por un jugador. Dentro del juego podría meterme dentro de una torreta y ponerme a disparar. Ahí controlaría la torreta. La torreta es un Pawn.

Puede ser controlador por:

* Jugador
* IA

**Character**: hereda de Pawn y es humanoide.

Crea un nuevo Character y le pone estas dos atributos

A screen shot of a computer program

Description automatically generated with low confidence

Para decir que cuando le demos al play salga un Character custom, se hace desde el GameMode.

# Input

### Forma rápida:

Graphical user interface, application

Description automatically generated

NO HACER UN COMMIT ASI, ES PARA PRUEBAS.

Si quiero buscar quien le da a la P, encontrar ese nodo es MUY difícil.

Y cuidado también con el Consume Input (si está activado nadie más sabe si se presiona la P).

### Enhanced Input

Graphical user interface, application

Description automatically generated

Se necesita un **Mapping Context y dos Input Action**: (podíamos tener varios mapping action, por ejemplo si tengo unas acciones de personaje normal y luego si cojo un arco otras distintas)

* IA\_CharacterMovement
* IA\_CameraMovemen

Y dentro del mapping añadimos las actions:

A screenshot of a computer

Description automatically generated with medium confidence

Vuelve al código C++ y hace el Begin Play para añadir el mapping context al PlayerController

Text

Description automatically generated

Y en el setup nos vinculamos a Acciones (como el MCVGameCharacter que había ya creado)

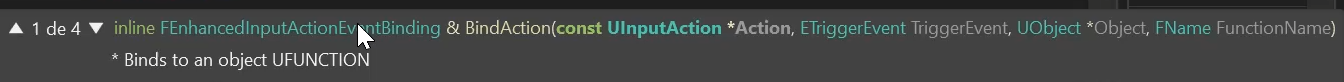
Text

Description automatically generated

**Delegate**: variable con una lista de funciones. Cuando la variable se activa se llama a todas las funciones de la lista.

Los delegates tienen una **signature** o firma (unos parametros). Si creo un delegate q manda un InputComponent por parámetro, todas las funciones que añadimos al delegate deben tener ese parámetro.

El bind action tiene los parámetros



* Ptr a Action que queremos registrarnos (CameraMovementAction)
* Enumeration para ver cuando se llama a esa acción
  + Started: justo al pulsar sobre un botón
  + Trigger: le puedo poner que se lance a los 2 segundos
    - Dead Zone: para el mando se puede poner que si va de 0 a 0.2 no haga nada.
  + …
  + None:
* Objeto: this normalmente
* Función: &Clase::NameFunc

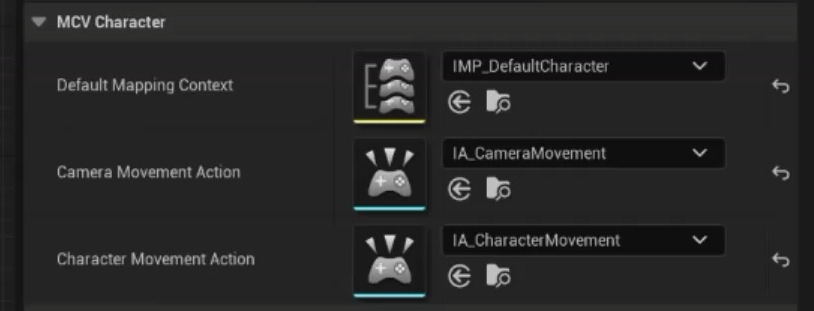
## Categorias

En los details de la BP aparecen las variables creadas con un nombre por defecto de la clase. Pero le podemos añadir una categoría.

Text

Description automatically generated

Así está como por defecto con el nombre de la clase:



Ahí en los details añadimos las variables creadas.

## Rotar Camara

Hay un delegate interno a Unreal que ya viene con el parámetro FInputActionValue



El Value tiene un Get template y consigo mi Fvector2D

Text

Description automatically generated

OJO:

* Camera Settings – use pawn control rotation debe estar activo

Graphical user interface, application

Description automatically generated

* Y el tipo de input debe ser 2d

En el Mapping podemos añadir modificadores para negar. Si la cámara la queremos invertida al mover el ratón por ejemplo.

Graphical user interface, application

Description automatically generated

Función para el movimiento del personaje

Text

Description automatically generated

GetControlRotation coge la rotación del Player. Para moverlo no queremos coger su dirección porq si está mirando al suelo no queremos que se mueva hacia abajo.

Debemos coger solo la rotación en el eje vertical. Con la librería matemática cojo el vector fwd y right.

## Swizzle

Todavía no mueve bien porq necesitamos un **Swizzle** para intercambiar XYZ por YXZ y un negate

Graphical user interface, application, table

Description automatically generated

Se puede rotar el pawn hacia donde se mueve (Character Movement Details)

# Animations

## Skeleton, Mesh, Weighs, Animation

**Skeleton** es una Jerarquía de huesos. La **animación** se encarga de mover estos huesos. Por encima de este hueso está la **malla**.

A close-up of water drops

Description automatically generated with low confidence

Skeletal mesh: es la malla

PhysicsAssets: para hacer colisiones con la malla

* Para ragdolls: si tiene que colisionar el brazo contra el suelo y el cuello colisiona con otra cosa
* O si necesitamos precisión para disparar una bala por debajo del brazo por ejemplo.

Cada vertice tiene más **influencia** o menos con respecto a los huesos más cercanos. Puede ser varios huesos. Por ejemplo un vertice de la mano puede tener un 80% al hueso del dedo y un 20% al del codo

Importar FBX

Al importar el modelo FBX podemos activar esta opción para no tener que rotar el character.

Graphical user interface, text, website

Description automatically generated pero si lo hacemos tenemos que hacerlo tb en las anims

Marcar **Import Normals** por si los de arte las modifican.

* **Speed**: magnitud
* **Velocity**: include direction

## Gestión de animaciones

**Clase**: Animations Instance (c++) / Animation Blueprint

La lógica del Character va en el character. El animGraph lo que hace es coger variables para saber el estado y a través de las variables mezcla animaciones.

Nodo Blend para hacer un blend de 2 animaciones.

Graphical user interface, application, Teams

Description automatically generated