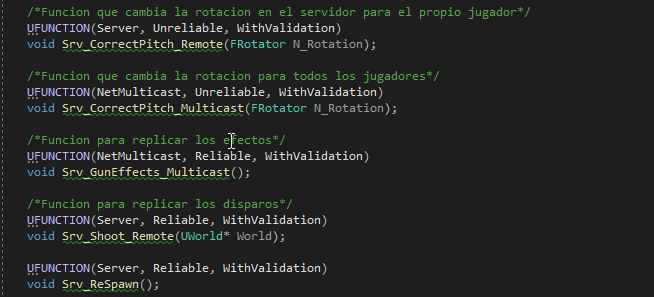
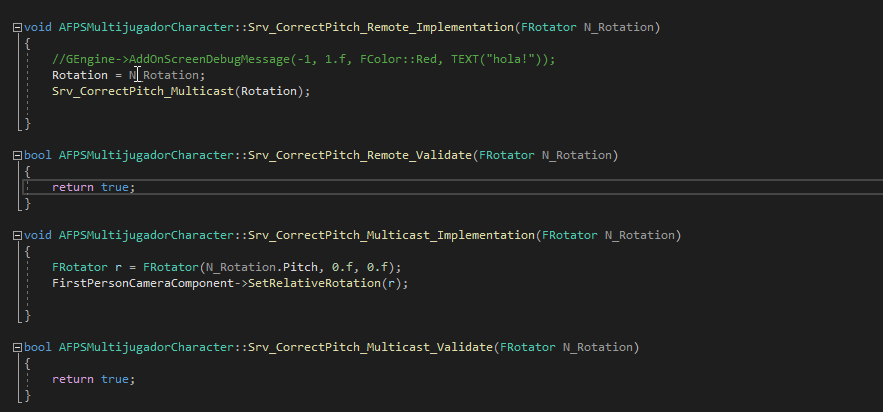
# Sección 1

Se han creado las siguientes funciones para replicar lo requerido:



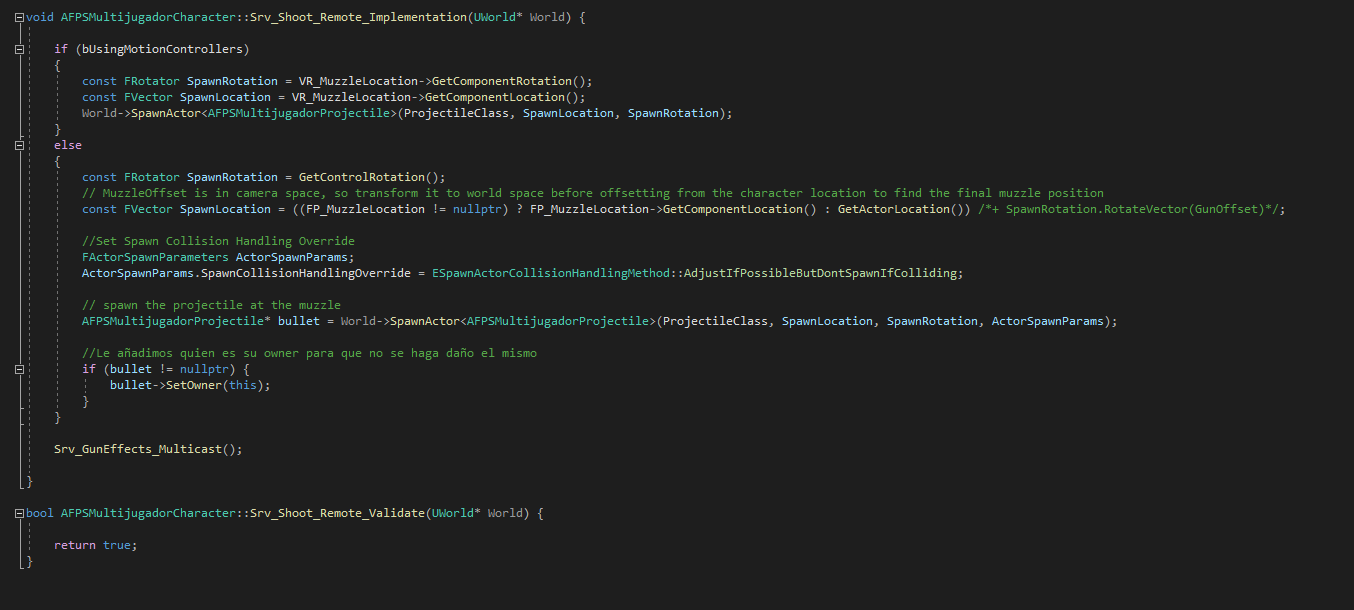
Para esta parte se ha activado la replicación en los objetos pertinentes, tanto como las balas en el blueprint cómo los objetos con físicas del escenario.

Se ha hecho una corrección del pitch en ambos jugadores para que se vea en ambos clientes, las funciones CorrectPitch son para ello. Una que se ejecuta en el servidor y otra en todos:



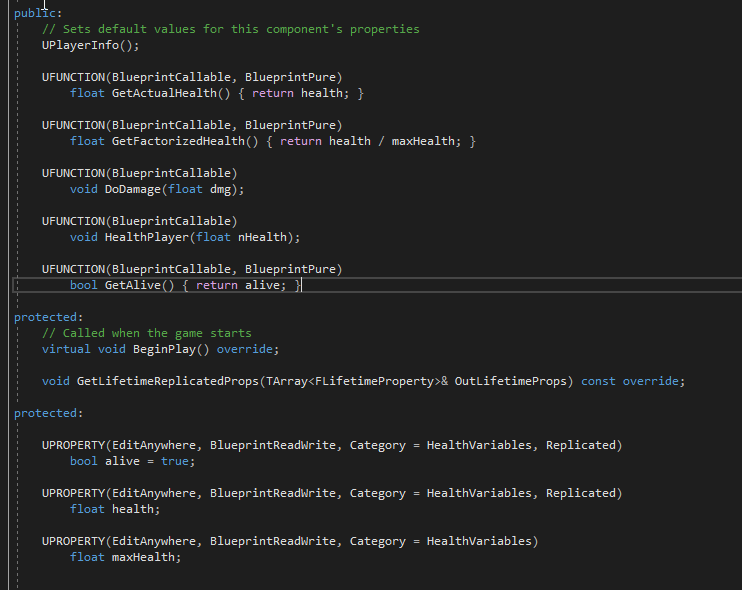
Esta función es llamada en la función del input correspondiente.

Los proyectiles, por su parte se ha separado los efectos en una función multicast y las balas en otra del servidor, ya que son replicadas por blueprint solo nos hacia falta hacer que se muevan en el servidor:



# Sección 2

El componente se ha creado mediante c++ y luego se ha añadido al jugador en el constructor para que el blueprint lo tenga:



Las funciones get se han declarado en el mismo .h y se han hecho puras para cuando sean llamadas en los blueprint.

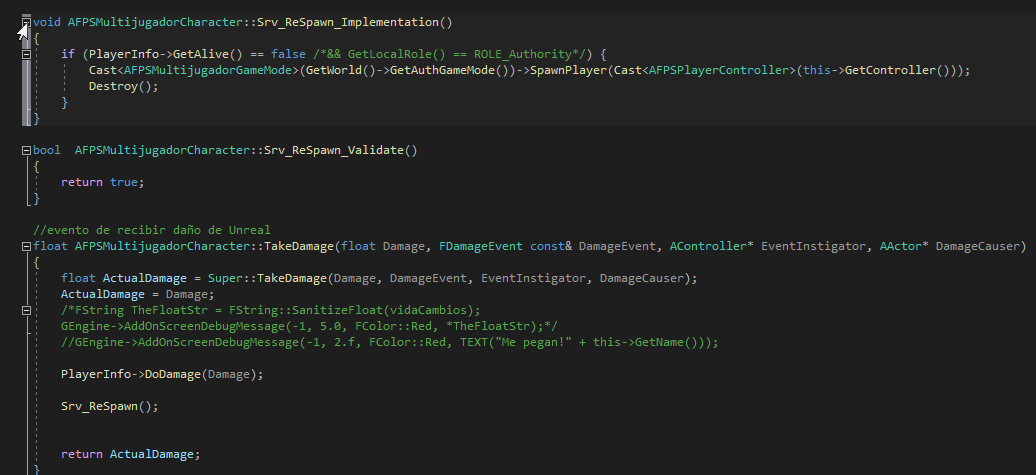
La vida se ha hecho que sea replicable, como el propio estado, y se vaya actualizando cada tick del servidor. Se ha optado por dos funciones distintas de curar y dañar para que sea mas entendible a la hora de llamar dichas funciones. Se podría haber hecho en una sola, pero así se separan las funcionalidades.

# Sección 3:

Para esta parte, he creado mi propio gamemode y controller para poder gestionar mejor la muerte y el respawn de los jugadores.

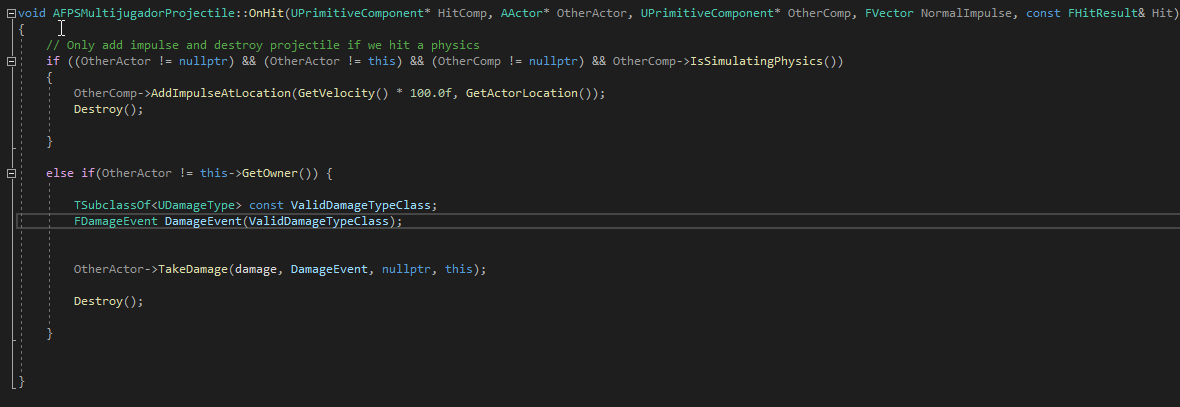
Mirando por la parte del jugador y la propia bala, se ha usado los eventos que proporciona Unreal de TakeDamage and ApplyDamage para hacer daño y recibirlo:

**FPSMultijugadorCharacter.cpp.**

****

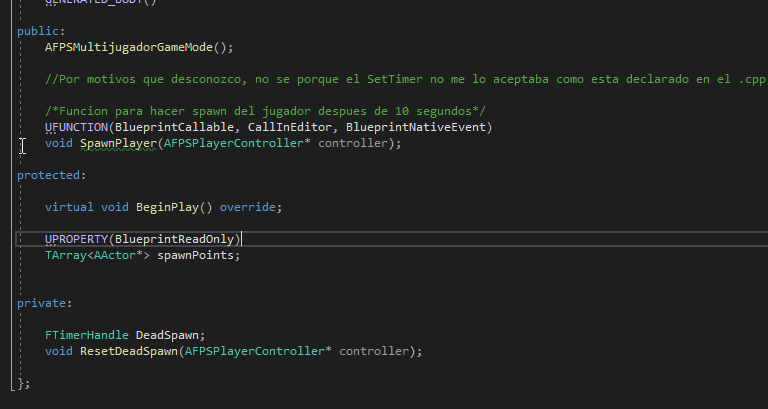
Aquí vemos como se recibe daño y se quita vida. La función de ReSpawn se ejecuta en el servidor donde, usando el Gamemode que se ha creado previamente (AFPSMultijugadorGameMode) para llamar al respawn del jugador muerto.

**AFPSMultijugadorProjectile.cpp**



Aquí no hay mucho misterio, ambas balas se destruyen y si golpea contra un Jugador le aplican daño. Los jugadores no se pueden hacer daño a sí mismos.

**FPSMultijugadorGameMode.cpp/.h**

****

Aquí se me presentó un problema para hacer el delay. Se ve que en esta versión de Unreal (4.26.0) como declaro normalmente la función se ve que no había ninguna sobrecarga para ese método.



GetWorld()->GetTimerManager().SetTimer(DeadSpawn, AFPSMultijugadorGameMode::ResetDeadSpawn(controller), 10.0f, false);

Se ve que la anterior declaración no está presente, así que, por motivos de diseño la función SpawnPlayer se puede implementar en Blueprints y he continuado por ese lado.

**MyFPSMultijugadorGame.BP**

En el Blueprint he hecho que el controller por defecto sea un Blueprint instanciado de la clase de C++.

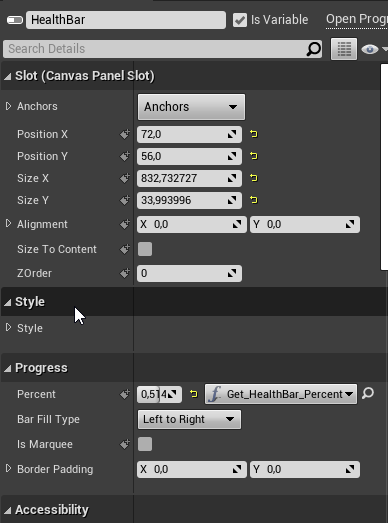
Cuando es llamado en C++ en el momento que el jugador muere, al no tener implementación en C++, se llama al del Blueprint dónde se gestiona que tiene la autoridad para solo crear un jugador. Este método recibe el controlador del jugador muerto antes de ser destruido para crear otra instancia y sea poseída por éste.

Los SpawnPoints es un array de target points que se recogen en el Begin Play en C++ y son usados en el Blueprint para spawnear al jugador.

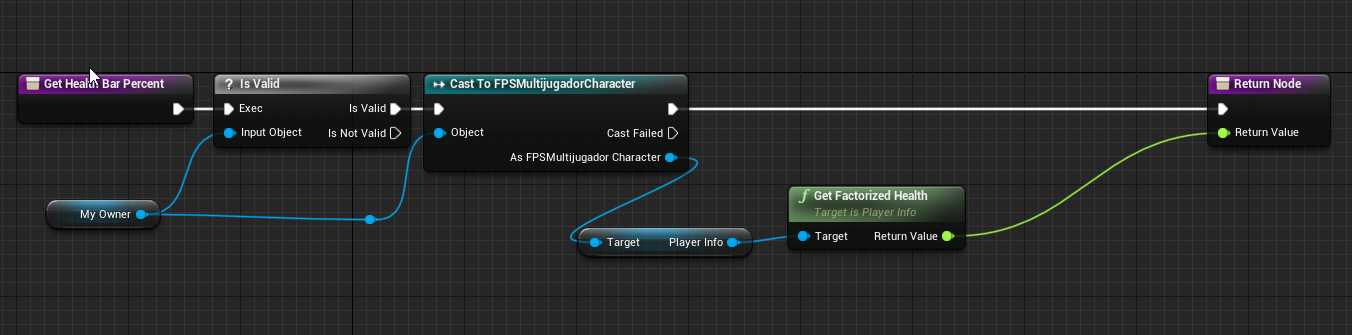
Más tarde, la UI también se reinicia, ya que, se crea con el jugador al principio en el controller.

# Sección 4

Para la UI se ha creado un widget simple con una progress bar:



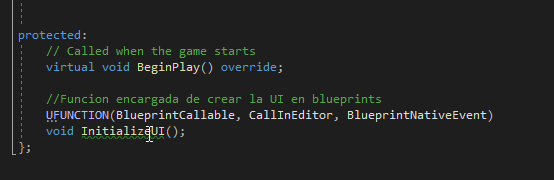
Se ha hecho bind del porcentaje de la barra usando la función de la vida factorizada:



Cuando se crea el widget, se ha hecho una variable del jugador que se añade cuando lo creas.

Aquí se ven unas pequeñas comprobaciones para que no sea nulo el personaje.

**FPSPlayerController.h**



Al igual que en el Gamemode, he creado una función que está declarada y funcional en Blueprint para que sea llamada cuando el jugador vuelva a ser creado, ya que, se le pasa un Controller creado en C++ y así no hacer un cast.

# Sección 5

## Notas y apuntes

Por motivos de falta de tiempo, una vez todo lo importante de la prueba esté completada, los inputs que están en el AFPSMultijugadorCharacter, se deberían mover al propio controller. O por lo menos las básicas que debería tener un jugador (el movimiento y el control de la cámara) por si se decide hacer una herencia de éste y no se quiere que tenga ciertas funcionalidades futuras: como disparar, usar alguna habilidad única, volar…

También se debería haber movido el arma a otra clase dónde se le añadan una skeletal mesh, el punto de spawn de las balas, etc… En esa clase, estaría la creación de las balas como tal y se pondría el daño que hiciera el arma en ese objeto, y no en las balas como está ahora.

Se crearía en el jugador y se spawnearia en el mundo para luego hacerle un attach en un socket en el propio esqueleto del jugador.