**Documentos de Programación del Enemigo**

**Tools**

* Unity (ver. 2021.3.30f)
* Visual Studio 2022 (C#)

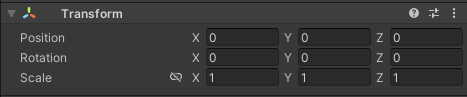
# **Objeto**

## 1. Enemy

### 1.1 Enemy\_GameObject (Principal)

#### 1.1.1 - Transform

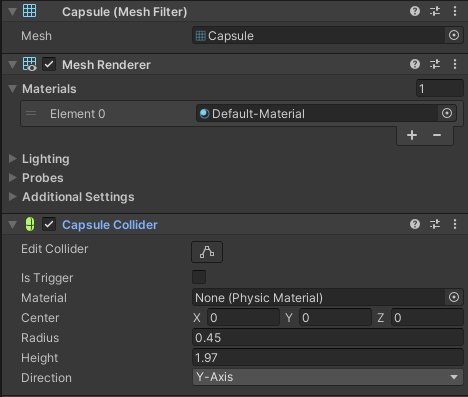
Determina la posición, rotación y escala del objeto.



#### 1.1.2 – Capsule Collider y Renderer

Determina el espacio que ocupa el personaje. Solo aspectos visuales son en 2D, pero la forma del jugador es en 3D.

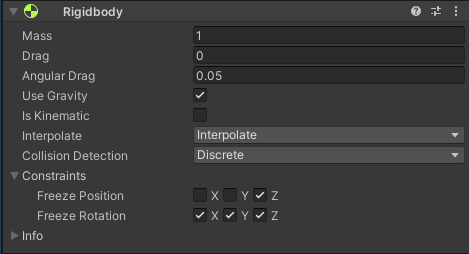
Usamos *Radius* y *Height* para mover los límites.





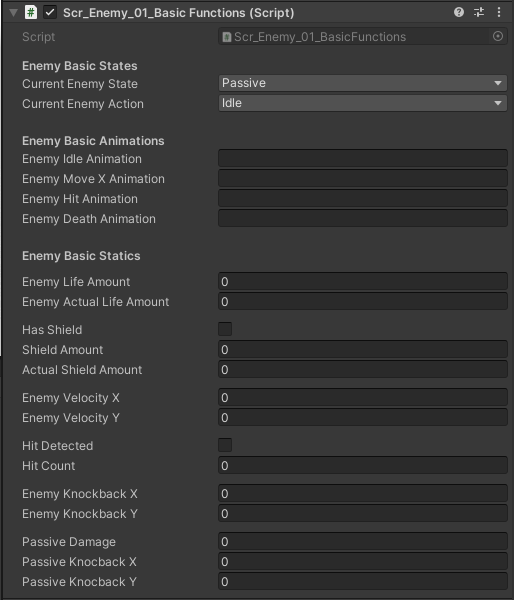
#### 1.1.3 - RigidBody

Proporciona la física a un objeto.

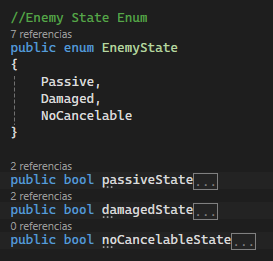


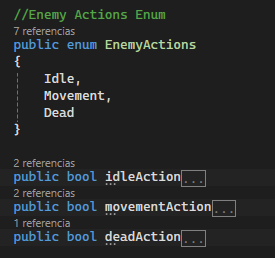
#### 1.1.4 – Scr\_Enemy\_01\_BasicFunctions

Este script se encarga de manejar las funciones basicas de un enemigo.



##### Enum

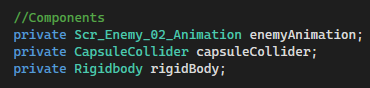




*EnemyState* determina el estado actual del enemigo.

*EnemyAction* determina la accion actual del personaje.

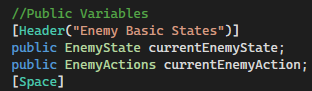
##### Variables



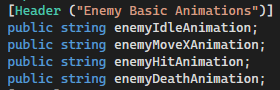
*enemyAnimation* - El script que controla las animaciones del enemigo.

*capsuleCollider* - El collider del enemigo.

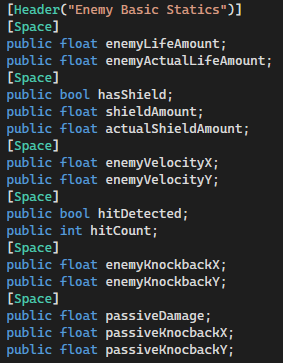
*rigidBody* - Se usa para el movimiento físico.



Referencia a los Enum.



Strings con los nombres del *Animation Clip* correspondiente a las animaciones basicas del enemigo.



*enemyLifeAmount / enemyActualLifeAmount* – La vida total y actual del enemigo.

*hasShield* – Una bool que indica si el enemigo tiene escudo.

*shieldAmount / actualShieldAmount* – La cantidad de escudo total y actual del enemigo.

enemyVelocityX / enemyVelocityY – La velocidad de movimiento del enemigo.

*hitDetected* – Una bool que se prende al detectar un golpe.

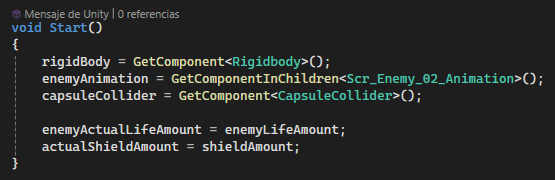
*hitCount* – La cantidad de golpes detectados.

*enemyKnocbackX / enemyKnocbackY* – La distancia de retroseso al ser golpeado.

*passiveDamage* – El daño que le da al jugador al chocarse.

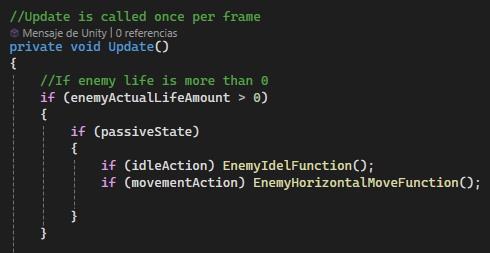
*passiveKnocbackX / passiveKnocbackY –* La distancia de retroceso que le da al jugador al chocarse.

##### Start()

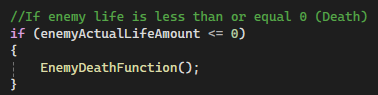


Localiza componentes y setea los valores iniciales (*actualLifeAmount* y *actualShieldAmount*).

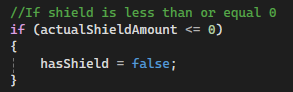
##### Update()



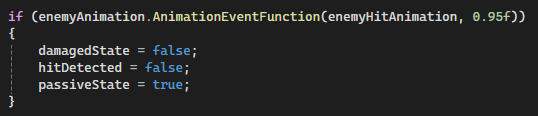
Si *enemyActuaLifeAmount* es mayor a cero y esta en *passiveState*, dependiendo de en que accion este el enemigo, *idleAction* o *movementAction*, se ejecuta la funcion correspondiente, *EnemyIdleFunction() o EnemyHorizontalMoveFunction().*



Si *enemyActuaLifeAmount* es cero o menos, es decir cuando muere el enemigo, se ejecuta la funcion *EnemyDeathFunction().*

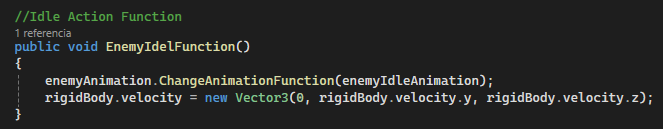


Si *actualShieldAmount* es cero o menos, enotnce se apaga *hasShield*.



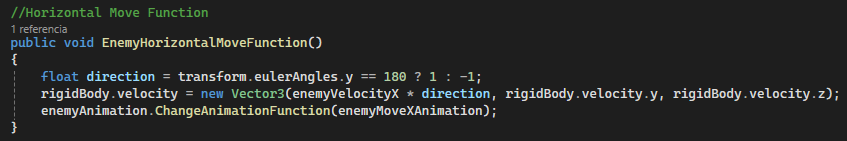
Cuando la animación de daño esté en 95%, se cambia de *damageState* a *passiveState* y se reinicia *hitDetected*.

##### EnemyIdleFunction()



Esta funcion se ejecuta cuando *EnemyActions* del enemigo es *idleAction*. Cambia la animacion a *enemyIdleAnimation* y detiene el movimiento del *rigidBody*.

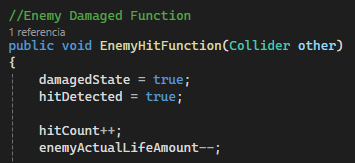
##### EnemyHorizontalMoveFunction ()



Esta funcion se ejecuta cuando *EnemyActions* del enemigo es *movementAction*. Se determina la dirección del movimiento dependiendo de la rotación del enemigo en el eje Y. Si *transform.eulerAngles.y* es 180, significa que está mirando a la izquierda, si es 1, esta mirando a la derecha.

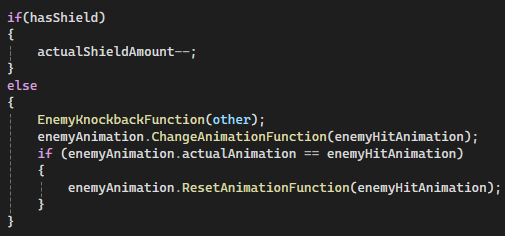
Mueve al objeto a traves de *rigidBody.velocity* y cambia la animacion a *enemyMoveXAnimation* y detiene el movimiento del *rigidBody*.

##### EnemyHitFunction()



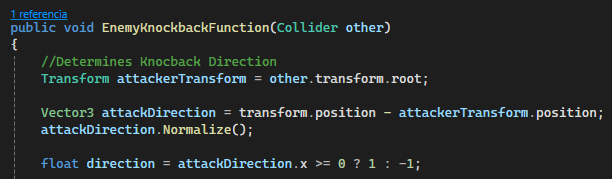
Esta funcion se ejecuta cuando se daña al enemigo, se llama en *Scr\_Enemy\_03\_Hurtbox.*

Cambia *EnemyState* a *damagedState* y prende la bool *hitDetection*. Tambien se le suma uno a *hitCount* y se resta uno a *enemyActualLifeAmount*.



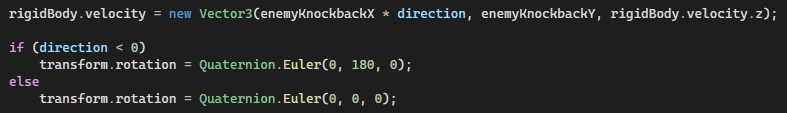
Si el enemigo *hasShield* entonces se le resta uno a *actualShieldAmount*. Si no entonces se llama al funcion *EnemyKnocbackFunction* y se cambia la animacion a *enemyHitAnimation*. Si la animacion ya es *enemyHitAnimation,* entonces se llama a *ResetAnimationFunction* para reiniciar la animacion.

##### EnemyKnockbackFunction()



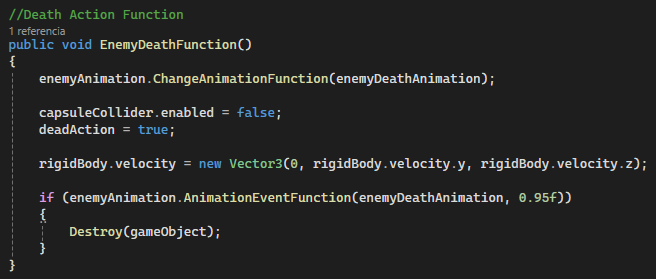
*attackerTransform* obtiene el objeto raíz del player.

*Vector3* calcula la dirección del ataque, que va desde el atacante hacia el enemigo. Luego la normaliza, para obtener un vector de dirección sin magnitud (1 o -1). Por ultimo, *float direction* determina si el ataque fue a izquierda o a la derecha del enemigo.



*rigidBody.velocity*, aplica el knocback del enemigo usando *enemyKnockbackX \* direction*. Se ajusta la rotación del enemigo dependiendo si *direction* es mayor o menor a 0.

##### EnemyDeathFunction()



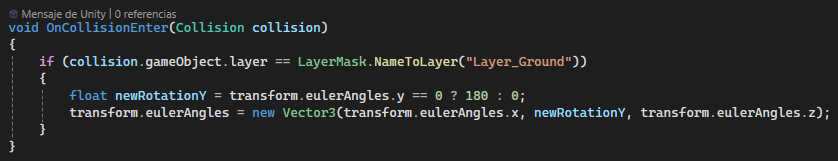
Cambia la animacion a *enemyDeathAnimation*.

Desactiva la colision del capsuleCollider y cambia *EnemyActions* a *deadAction*.

Detiene el movimiento de *rigidBody*.

Cuando la animacion llegue al 95% entonces se destruye el objeto.

##### OnCollisionEnter()

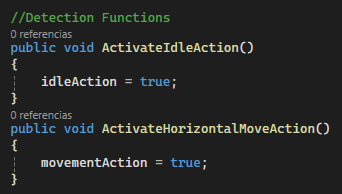


Esta funcion esta para detectar las paredes o esquinas del escenario para que el enemigo se de vuelta al moverse.

Al detectar un objeto con *Layer\_Ground* cambia *transform.eulerAngles* para rotar al objeto.

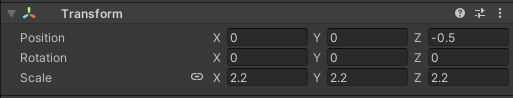
##### Detection Fuhnctions

El script Scr\_Enemy\_04\_Detection ejecuta funciones al detectar al jugador, para que funcione tenemos que crear funciones que, en este caso, active las acciones correspondientes.



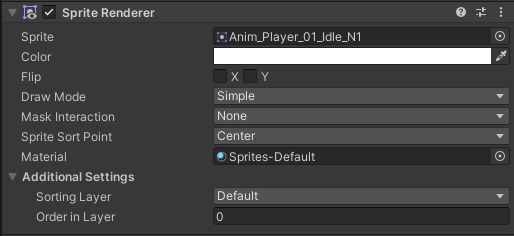
### 1.2 Enemy\_Sprite (Hijo)

#### 2.2.1 - Transform



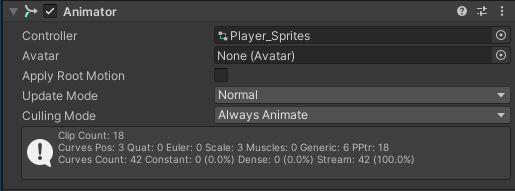
Determina la posición, rotación y escala del objeto.

#### 2.2.2 – Sprite Renderer



Este componente se usa para renderizar las imágenes en 2D.

#### 2.2.3 - Animator



Este compoente se encarga de las animaciones del jugador.

El tipo de Controller define el flujo de las animaciones y sus transiciones.

##### Animation Window (Como crear animaciones)

Las animaciones se hacen en la ventana Animation.



Para crear una nueva animacion tenemos que ir a la caja debajo de *Preview* y despues *Create New Clip.*



Cada animación es un archivo anim. Necesitamos crear uno con la estructura de nombre correcta:

*Anim\_[Number]\_[Character]\_[Animation Name]*

Ej: *Anim\_01\_Player\_Idle*



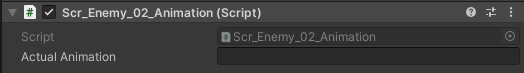
Luego, necesitamos los sprites que componen cada fotograma de animación, representados por rombos en la línea de tiempo.

Los *Samples* representa la velocidad a la que se reproduce cada fotograma. La base es 60, pero la velocidad adecuada es 15.

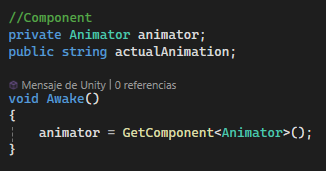


#### 2.2.4 – Scr\_Enemy\_02\_Animation

Este script contiene las funciones para la reproduccion de animaciones y los eventos que se pueden ejecutar.

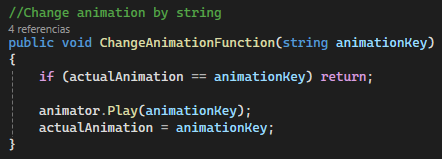


##### Variables



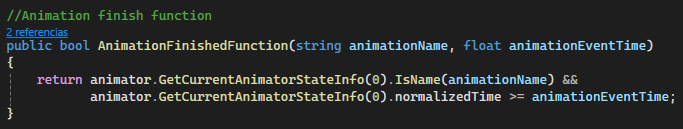
Se hace referencia al componente *Animator* de Unity y se hace un *string* que determina cual es la animacion actual que se esta ejecutando.

##### ChangeAnimationFunction()



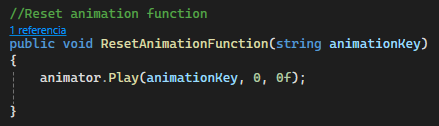
Esta funcion es la que se usa para cambiar de animacion, le pasamos *animationKey*, el cual es comparado a *actualAnimation* para evitar que se reinicie la animacion, y luego se ejecuta en el *animator* y cambia *actualAction*.

##### AnimationFinishedFunction()



A esta funcion es una bool que para ser true se le pasa dos valores, *animationName*, el nombre de la animacion, y *animationEventTime*, el tiempo dentro de esa animacion en el que pasa a true.

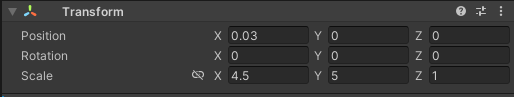
##### ResetAnimationFunction()



Reinicia la animacion en el *animator* regresandola a 0%.

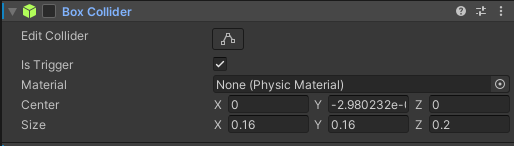
### 1.3 Enemy\_HurtBox (Hijo)

#### 2.3.1 - Transform



Determina la posición, rotación y escala del objeto.

#### 2.3.2 – Box Collider



Este compoente se encarga de demarcar la zona activa de nuestra hitbox en los ataques.

##### Animation Window (Como crear animaciones)

Las animaciones se hacen en la ventana Animation.



Para crear una nueva animacion tenemos que ir a la caja debajo de *Preview* y despues *Create New Clip.*



Cada animación es un archivo anim. Necesitamos crear uno con la estructura de nombre correcta:

*Anim\_[Number]\_[Character]\_[Animation Name]*

Ej: *Anim\_01\_Player\_Idle*



Luego, necesitamos los sprites que componen cada fotograma de animación, representados por rombos en la línea de tiempo.

Los *Samples* representa la velocidad a la que se reproduce cada fotograma. La base es 60, pero la velocidad adecuada es 15.

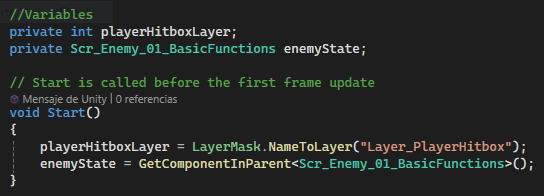


#### 2.3.3 – Scr\_Enemy\_03\_HurtBox



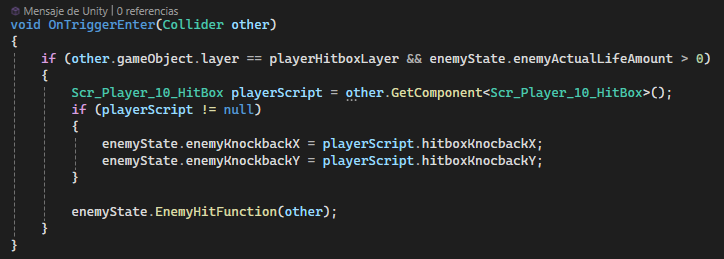
Este script controla la deteccion de la Hurt Box.

##### Variables



Se determina al player a traves de su layer en *playerHitboxLayer* y se hace referencia al script principal *Scr\_Enemy\_01\_BasicFunctions*.

##### OnTriggerEnter()

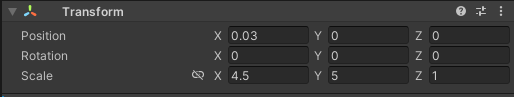


Si se entra en el trigger de un objeto con la misma layer que *playerHitboxLayer*, se intetna buscar el script *Scr\_Player\_10\_HitBox* en el objeto para extraer el knocback que causa el ataque en X (*playerScript.hitboxKnocbackX*) e Y (*playerScript.hitboxKnocbackY*).

Llama al funcion *EnemyHitFunction,* se le pasa *other* para determinar la direccion de la colision.

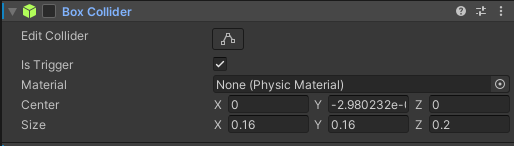
### 1.4 Enemy\_Detection (Hijo)

#### 2.4.1 - Transform



Determina la posición, rotación y escala del objeto.

#### 2.4.2 – Box Collider



Este compoente se encarga de demarcar la zona activa de nuestra hitbox en los ataques.

##### Animation Window (Como crear animaciones)

Las animaciones se hacen en la ventana Animation.



Para crear una nueva animacion tenemos que ir a la caja debajo de *Preview* y despues *Create New Clip.*



Cada animación es un archivo anim. Necesitamos crear uno con la estructura de nombre correcta:

*Anim\_[Number]\_[Character]\_[Animation Name]*

Ej: *Anim\_01\_Player\_Idle*

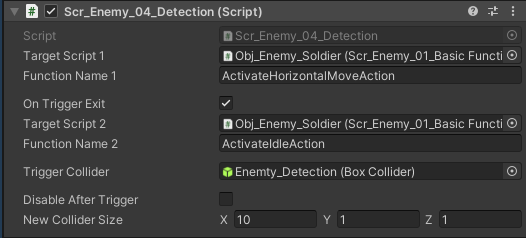


Luego, necesitamos los sprites que componen cada fotograma de animación, representados por rombos en la línea de tiempo.

Los *Samples* representa la velocidad a la que se reproduce cada fotograma. La base es 60, pero la velocidad adecuada es 15.

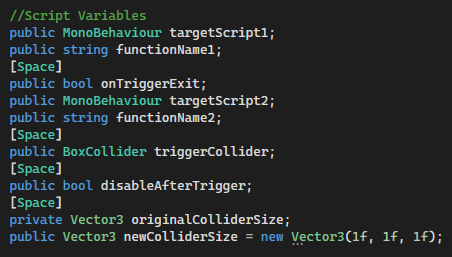


#### 2.4.3 – Scr\_Enemy\_04\_Detection



Este script controla la deteccion del objeto player y las acciones relacionadas.

##### Variables



*targetScript1* – El script que contiene la funcion que se va a ejecutar al detectar al jugador.

*functionName1* – El nombre de la funcion que se va a ejecutar al detectar al jugador.

*bool onTriggerExit* – Una bool que determina si se ejecuta una accion al dejar de detectar al jugador.

*targetScript2* – El script que contiene la funcion que se va a ejecutar al dejar de detectar al jugador.

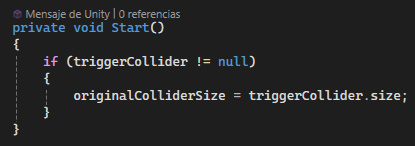
*functionName2* – El nombre de la funcion que se va a ejecutar al dejar de detectar al jugador.

*triggerCollider* – Referencia al box collider que se usa para la deteccion.

*originalColliderSize* – Referencia al tamaño original del box collider.

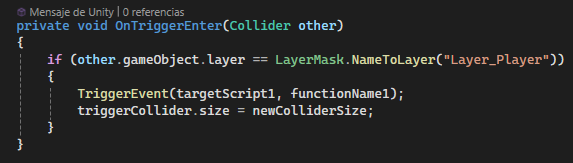
*newColliderSize* – Referencia al nuevo tamaño del box collider.

##### Start()



Se guarda el tamaño original del collider para luego poder restaurarlo.

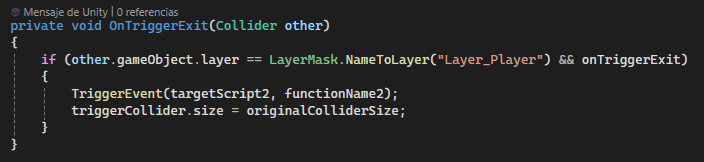
##### OnTriggerEnter()



Comprueba si el objeto que entró es el jugador (Layer\_Player).

Se ejecuta *TriggerEvent()* y se cambia el tamaño del *triggerCollider* a *newColliderSize*.

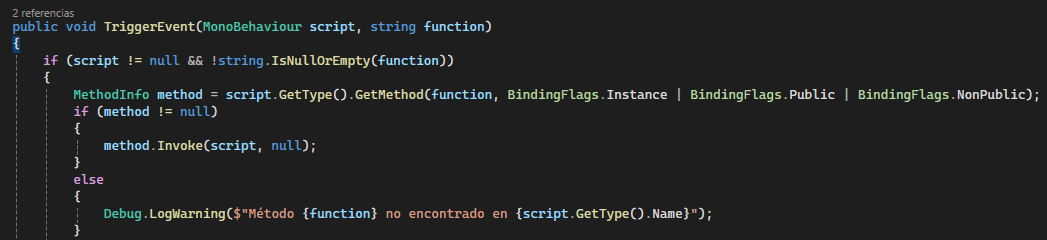
##### OnTriggerExit()



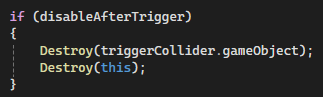
Comprueba si el objeto que salio es el jugador (Layer\_Player) y si *onTriggerExit* esta prendido.

Se ejecuta *TriggerEvent()* y se cambia el tamaño del *triggerCollider* a *originalColliderSize*.

##### TriggerEvent()



Este método llama a una función por nombre. Busca el método en *scrip*, aunque sea privado y si lo encuentra, lo ejecuta. Si no lo encuentra, lanza un mensaje de advertencia.



Si *disableAfterTrigger* es true, destruye el collider y el script después de ejecutarse.