**Documento de Programa**

**Tools**

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

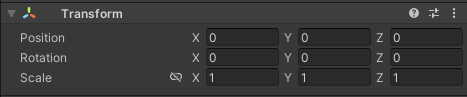
# **Objetos**

## 1. Obj\_Player

Es el objeto principal del objeto, contiene todos los hijos y componentes esenciales para el personaje.

### 1.1 - Transform

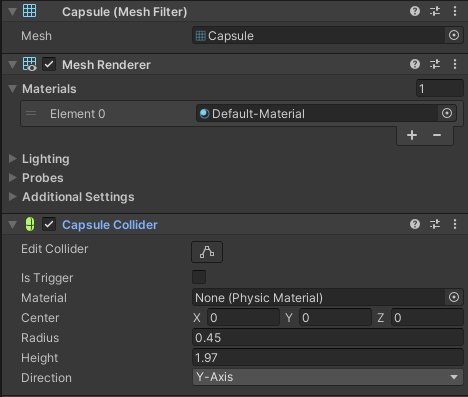
Determina la posición, rotación y escala del objeto.



### 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.

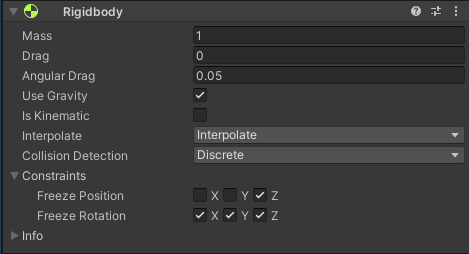
Usamos *Radius* y *Height* para mover los límites.





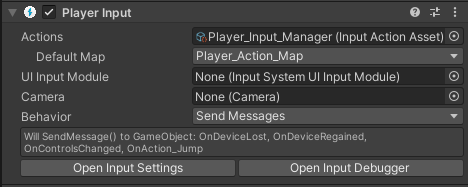
### 1.3 - RigidBody

Proporciona la física a un objeto.

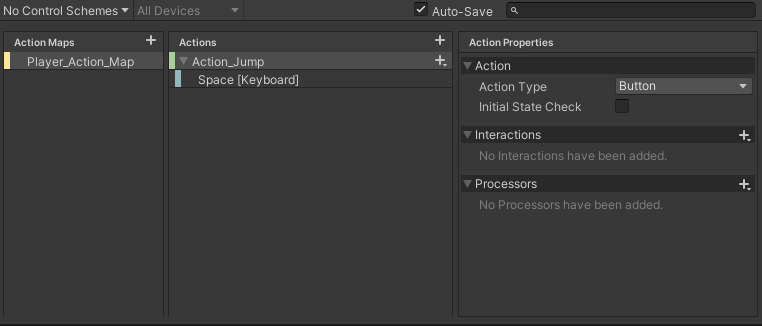


### 1.4 – Player Input

Este componente es el encargado de manejar el Input System de Unity.



*Default Map* se refiere al mapa de acciones al que accedemos por defecto. El mapa de acciones contiene dentro las acciones y inputs que se realizan.



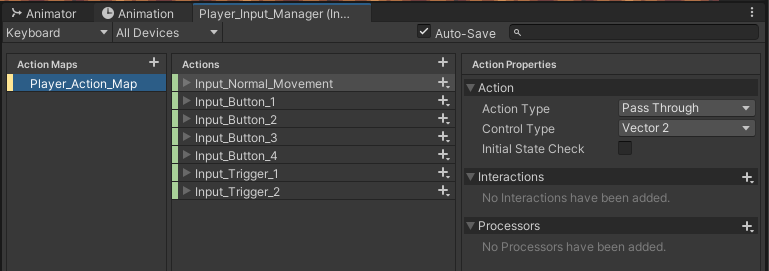
En esta ventana es donde se controlan y asginan los Inputs.

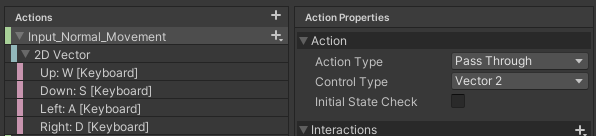
*Action Maps* se refiere al conjunto de accions que tiene un tipo de control, por ejemplo, se puede tener un tipo para las acciones del personaje y otra para cuando se navegan los menus.

*Actions* es cada accion individual que exista, cada una tiene un *Action Type*:

* *Button* - Para entradas ON/OFF (Ej: saltar con un botón).
* *Value* - Para valores continuos (Ej: dirección de un joystick).
* *Pass-Through* - Para flujos de datos sin filtrado.

Debajo de las acciones se define el *Binding*, la tecla/boton la cual ejecuta la accion.

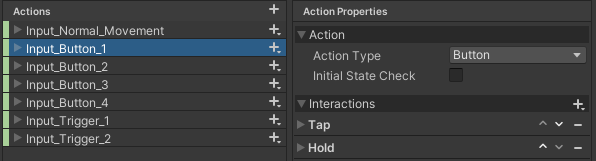




* *Input\_Normal\_Movement* – Contiene los inputs necesarios para el movimiento.

*Actyon Type* es *Pass Through* esto significa que el valor del input se envía sin ningún tipo de procesamiento adicional. (Sin los eventos que se detectan en Button o Value).

*Control Type* es *Vector 2*, lo que significa que la acción espera dos valores flotantes (X, Y).



* *Input\_Button / Input\_Trigger* – Son los inputs para los botones principales del juego. Son *Action Type Button*, esto significa que en el codigo podemos detectar estados como "Started", "Performed" o "Canceled".

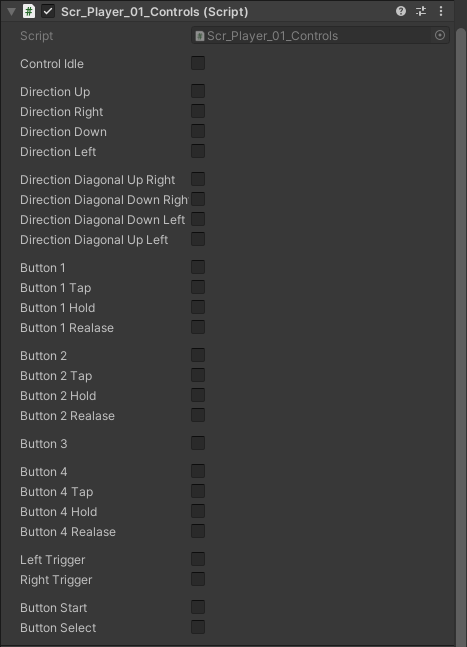
Hay dos tipos:

Button 1, 2 y 4 contienen *Interactions Tap* y *Hold*. Esto permite detectar en estos botones cuando se presiona rapidamente o se deja presionado.

Button 3 y Trigger 1 y 2 por el otro lado no cuenta con estas *Interactions*, lo importante es detectar cuando se usan, por lo que no importa como se presione.

### 1.5 – Scr\_Player\_01\_Controls

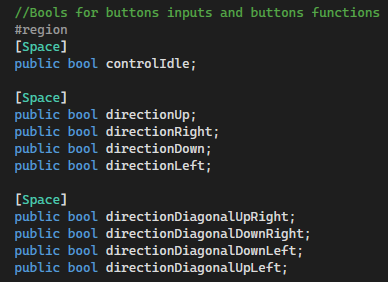
Este script se encarga de manejar el control que usa el jugador y sus inputs.



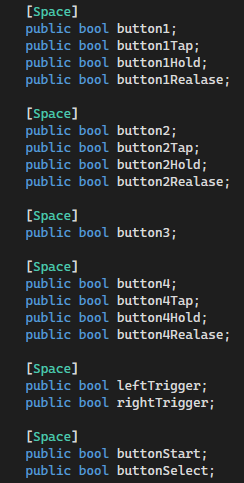
Variables



*playerActionControls* es una referencia al Input Asset generado con el nuevo sistema de Inputs. Permite acceder a los mapas de acción de *Player\_Action\_Map* y sus inputs.

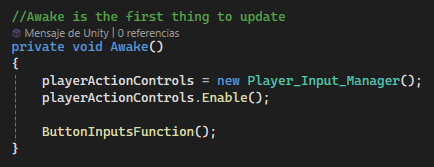


Las direcciones se componen de los cuatro ejes mas cuatro diagonales.



Para los inputs con *Tap y Hold* es necesario detectar *Tap, Hold y Realase.*

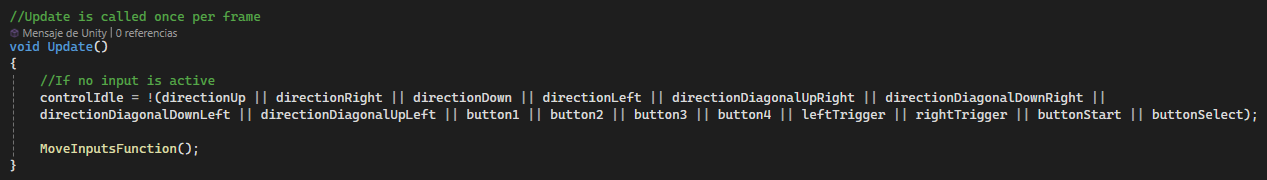
Awake()



Se crea una nueva instancia de *Player\_Input\_Manager* y se habilita.

Despues se llama a la funcion *ButtonInputsFunction()*, que asigna las funciones para detectar la entrada de botones.

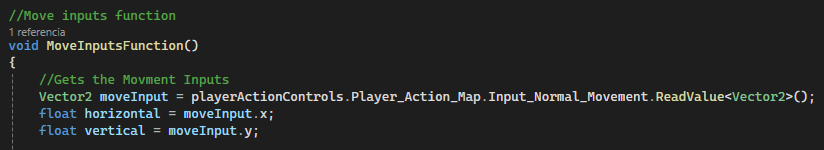
Update()



La bool *controlIdle* se activa cuando ningun otro boton esta activado.

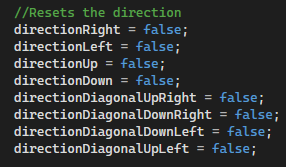
Se llama a la funcion *MoveInputsFunction()*, que actualiza los valores de dirección.

Funcion MoveInputsFunction()

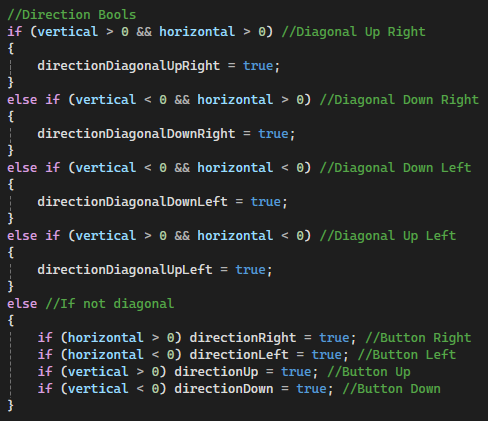


La funcion *MoveInputs()* se ejecuta en *Update()* y controla los Inputs utiliados para el movimiento.

*moveInput* lee el *Vector2* de los input en el *Action Map* (x = izquierda/derecha, y = arriba/abajo).



Se asegura de que las direcciones previas no se queden activas.

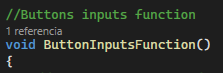


Esta seccion detecta el movimiento.

Si vertical y horizontal son ambos positivos o ambos negativos, significa que el stick se mueve en una dirección diagonal. Se activan las variables correspondientes (*directionDiagonalUpRight*, *directionDiagonalDownRight*, etc.).

Si el movimiento no es diagonal, activa solo una dirección (*directionRight*, *directionLeft*, etc.). Esto evita que se activen múltiples direcciones al mismo tiempo.

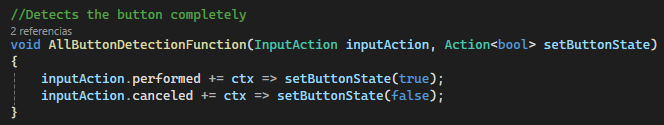
Funcion NewInputsFunction()



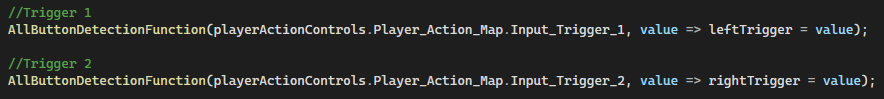
Esta funcion e encarga de asignar las funciones de detección a cada botón. Dentro de esta se llama a diferentes funciones que determinan el tipo de deteccion que tienen los botones.

Funcion AllButtonDetectionFunction()

Esta funcion es la deteccion basica del control. Activa una variable cuando se presiona (performed) y desactiva la variable cuando se suelta (canceled).



Se usa en botones simples como leftTrigger y rightTrigger.



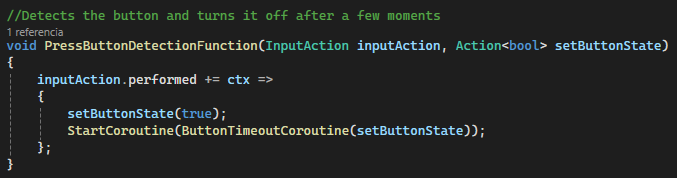
Se pasan los valores *InputAction inputAction* y *Action<bool> setButtonState.*

*inputAction* referencia al Input en el asset.

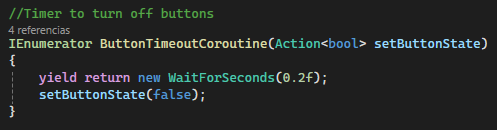
*setButtonState* la bool correspondiente a ese input en el script.

Funcion PressButtonDetectionFunction()

Esta funcion es una deteccion basica pero con un timer para apagarse despues de un tiempo.



Activa el botón cuando se presiona. Después de 0.2 segundos lo apaga la corutina ButtonTimeoutCoroutine().



Se usa en button3 (saltar), porque solo necesita activarse por un instante.

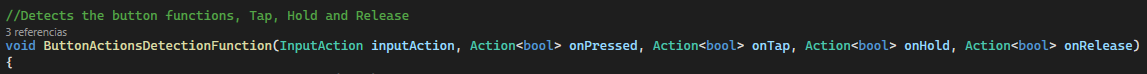


*inputAction* referencia al Input en el asset.

*setButtonState* la bool correspondiente a ese input en el script.

Funcion ButtonActionsDetectionFunction()

Esta funcion es para los botones mas complejos a los que se les tiene que detectar *Tap, Hold y Release.*



Parámetros:

*inputAction* referencia al Input en el asset.

*onPressed* se activa cuando el botón se empieza a presionar.

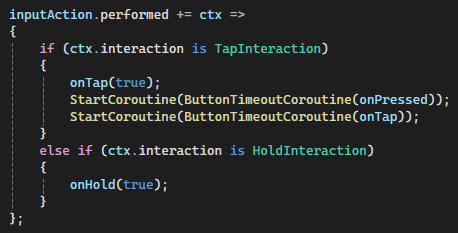
*onTap* se activa si el botón es presionado y soltado rápidamente.

*onHold*: se activa si el botón se mantiene presionado.

*onRelease* se activa cuando se suelta el botón después de haber sido mantenido.



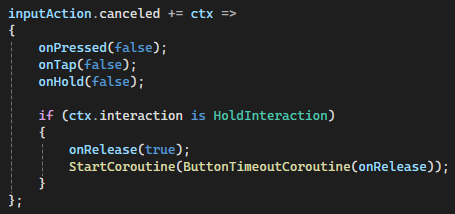
Esto se activa cuando el botón comienza a ser presionado (*inputAction.started*). Marca la variable bool correspondiente como true (ejemplo: button1 = true).



Se activa cuando el boton esta siendo presionado (*inputAction.performed*).

Si la interacción es Tap (presionar y soltar rápido), se activa *onTap(true)* y inicia corrutinas para apagar *onPressed* y *onTap*.

Si la interacción es un Hold (mantener presionado), se activa *onHold(true),* lo que indica que el botón está siendo sostenido.

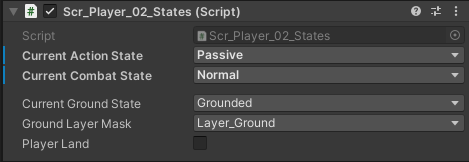


Se activa cuando el boton deja de usarse (*inputAction.canceled*).

Se desactivan todas las variables (*onPressed, onTap, onHold*). Si el botón estaba en modo Hold, al soltarlo se activa *onRelease(true),* la cual se apaga después un tiempol.

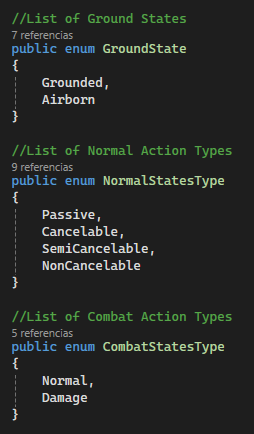
### 1.6 – Scr\_Player\_02\_States

Este script controla los diferentes estados en el que se encuentra el jugador.



Enum

El script define tres *enum* para representar distintos estados.

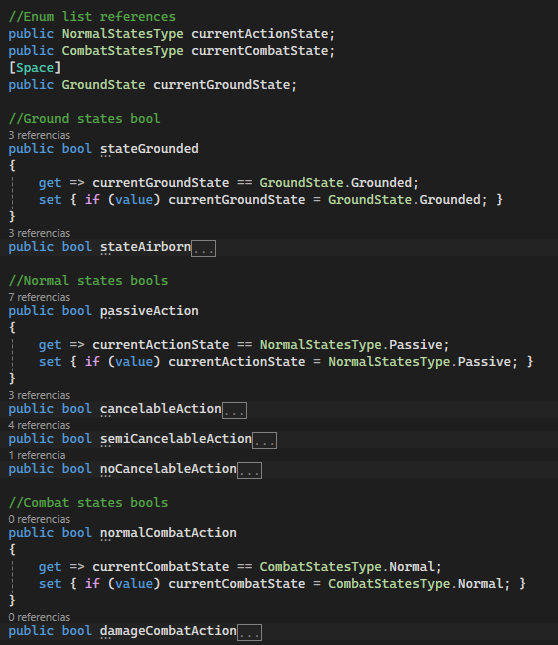


*GroundState* determina si el personaje esta en la superficie o no.

*NormalStatesType* determina el estado del personaje para realizar las acciones.

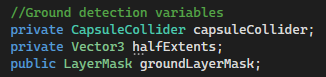
*CombatStatesType* determina el estado al estar en combate.

Variables



*Enum list references* son las referenicas en el editor de los enum.

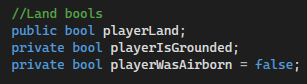
Las bools estan programadas con esta propiedad para setear el valor en true.



*capsuleCollider* referencia al componente que determina el collider del jugador.

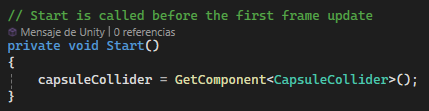
*halfExtents* define la distancia entre el personaje y la superficie para que detecte.

*groundLayerMask* indica en qué Layers de Unity se detectará como superficie.



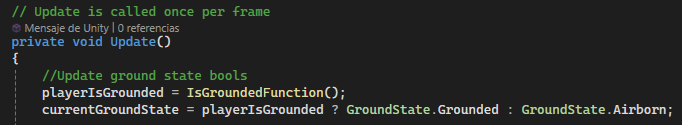
Bools utilizadas para el aterrizaje del jugador.

Start()



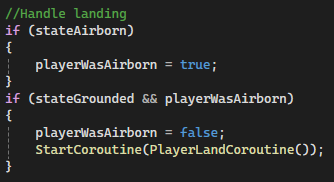
Obtiene el *CapsuleCollider* del personaje para usarlo en la detección de suelo.

Update()



*playerIsGrounded* se actualiza llamando a *IsGroundedFunction(),* que verifica si el personaje está tocando el suelo.

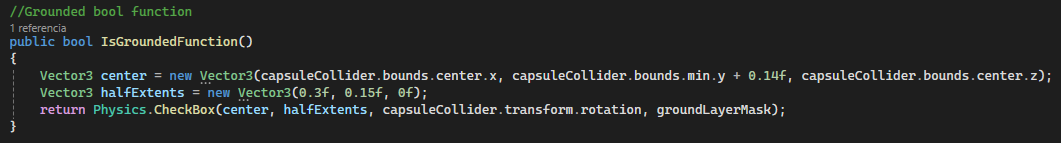
Luego, *currentGroundState* se actualiza con el estado correcto.



Si el personaje esta en el aire (*stateAirborn*), *playerWasAirborn* se pone en true.

Si luego aterriza (*stateGrounded*) y *playerWasAirborn* es true, se inicia la corutina *PlayerLandCouroutine()* para manejar el aterrizaje.

IsGroundedFunction()



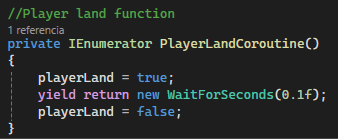
Esta funcion se encarga de detectar si el personaje esta sobre una superficie o no.

Usa *Physics.CheckBox()* para detectar si el personaje está tocando el suelo.

*Vector3 center* define la posición del centro de la caja de detección, ubicada un poco por encima de la base del *CapsuleCollider*. *halfExtents* define el tamaño del área de detección.

Si la caja detecta colisión con un objeto con *groundLayerMask* (Ground Layer), retorna true (el personaje está en el suelo).

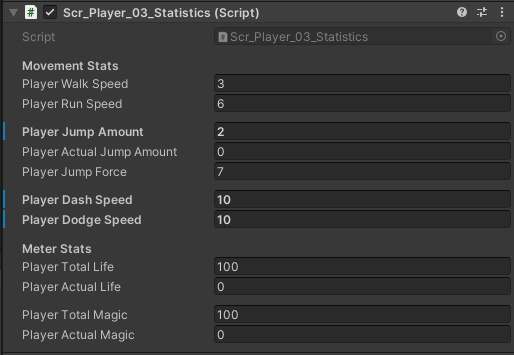
PlayerLandCouroutine()



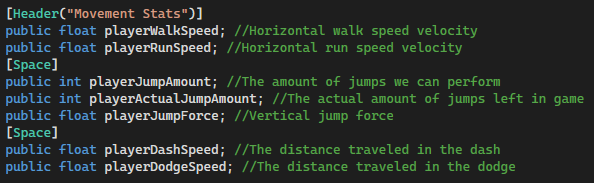
Cuando el jugador aterriza, *playerLand* se pone en true por 0.1 segundos y luego en false.

### 1.7 – Scr\_Player\_03\_Stats

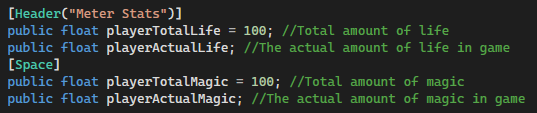
Este script contiene en un mismo lugar el valor numerico de las estadisticas del personaje.



Variables

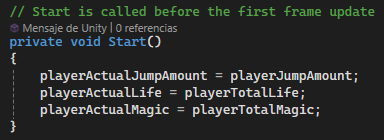


* **playerWalkSpeed** – La velocidad al caminar.
* **playerRunSpeed** – La velocidad al correr.
* **playerJumpAmount** – La cantidad total de saltos.
* **playerActualJumpAmoun** – La cantidad de saltos restantes in game.
* **playerJumpForce** – La fuerza de salto.
* **playerDashSpeed** – La distancia que reocrre al ejecutar un dash.
* **playerDodgeSpeed** – La distancia que recorre al esquivar.



* **playerTotalLife** – El total maximo de la vida del jugador.
* **playerActualLife** – La vida actual del jugador in game.
* **playerTotalMagic** – El total maximo de la magia del jugador.
* **playerActualMAgic** – La magia actual del jugador in game.

Start()

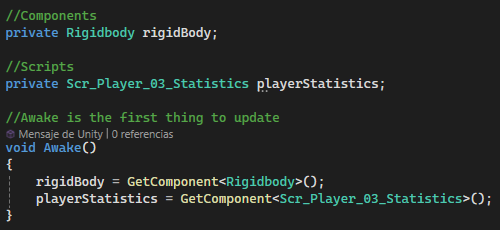


Se asigna la cantidad maxima a la cantidad actual.

### 1.8 – Scr\_Player\_04\_Physics

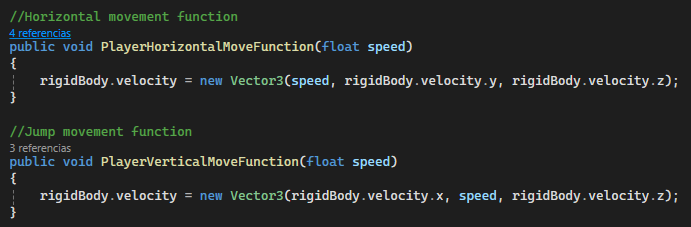
Este script contiene las funciones para el movimiento del objeto del jugador.

Variables



Para mover al jugador usamos el componente *rigidBody* combinado con las estadisticas que tenemos almacenadas en *playerStatistics*.

PlayerMovesFunctions



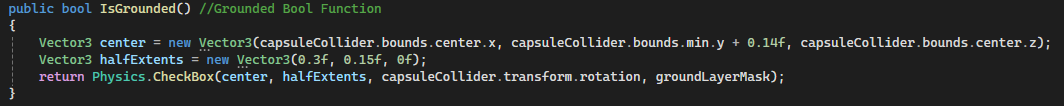
El objeto se mueve en referencia a estas funciones, las cuales pasan un valor y lo pasan al X o Y de un *Vector3*.

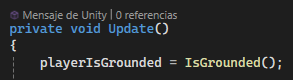
### 1.8 – Scr\_Player\_0?

Este script maneja el movimiento basico del jugador, movimiento horizontal y salto.

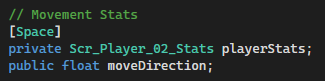
Variables



* **playerCanMove** – Verdadero si el personaje puede moverse.
* **playerIsGrounded** – Verdadero si el personaje esta en una superficie.

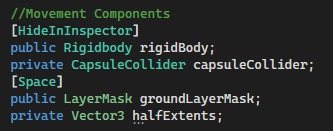


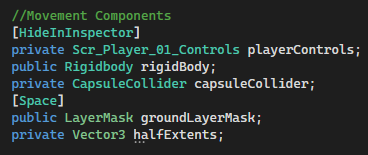
*IsGrounded()* es una funcion que contiene el codigo para saber si el jugador esta en colision con una superficie. Esta funcion regresa verdadero o falso, dependiendo de eso en el metodo *Update()* se prende la bool *playerIsGrounded*.



Las estadisticas de movimiento del jugador.

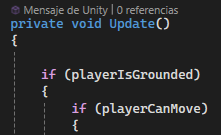
* **playerStast** – El script en donde se almacenan los valores de las estadisticas del jugador.
* **moveDirection** – Le da valor 1 si se presiona la derecha y -1 si se presiona la Izquierda.



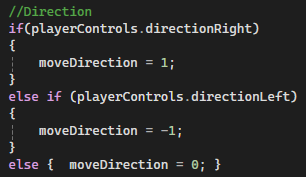


* **playerControls** – El script que contiene los inputs del jugador.
* **rigidBody** – Referencia al componente rigidBody del objeto.
* **capsuleCollider** – Referencia al componente capsuleCollider del objeto.
* **groundLayerMask** – Determina las Layers que son concideradas superficie.
* **halfExtents** – Determina la distanacia para la comprobacion de superficie.

Metodo Update()

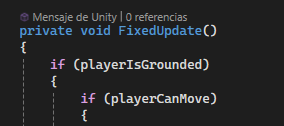


Si el jugador esta en una superifice (*playerIsGrounded*) y puede moverse (*playerCanMove*).

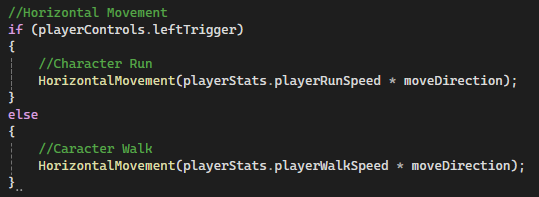


*moveDirection* se calcula usando *playerControls*, su valor depende del input que se ingrese.

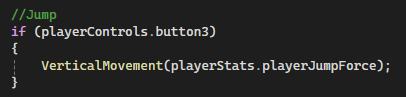
Metodo FixedUpdate()



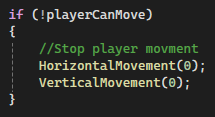
Si el jugador esta en una superifice (*playerIsGrounded*) y puede moverse (*playerCanMove*).



La funcion *HorizontalMovment* se encarga del movimiento horizotnal dependiendo del valor y la direccion que le pasemos, este depende si *leftTrigger* es precionado.

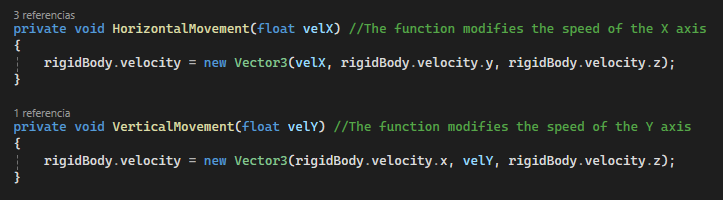


La funcion *VerticalMovment* se encarga del salto dependiendo del valor que le pasemos.



Si el jugador no puede moverse, se les pasa 0 a las funciones de movimiento para que el personaje pare.

Functions



Las funciones *HorizotnalMovement(velX)* y *VerticalMovment(VelY)*, son funciones que pasan un dato al eje X e Y respectvamente, causando que se mueva el personaje.