**Hit Box Program Document**

**General**

This document keeps track of the implementation of the game mechanics.

**Tools**

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

# **Player Object**

## 1. Principal Object - Character\_00\_Name\_Object

Is the main object of the object, it contains all the children and essential components for the character.

### 1.1 - Transform

Determines the position, rotation, and scale of the object.



### 1.2 - RigidBody

Provides the physics to an object.

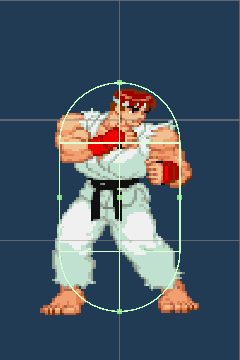
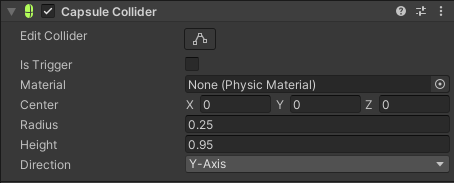


*Collision Detection –> Continuous:* The best method for accuracy in detections.

*Constraints:* The game is in 2D so we only need the X and Y Axis.

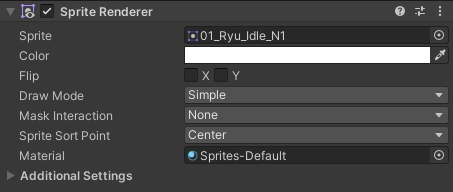
### 1.3 - Capsule Collider

Determines the space the character occupies. We use *Radius* and *Height* to move the boundaries.



### 1.4 – Sprite Renderer

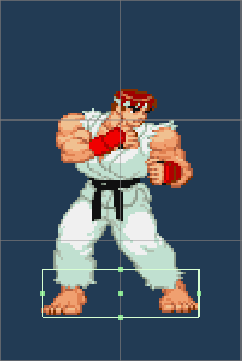
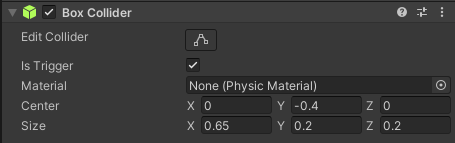
This component shows the sprites and images of the object.



### 1.4 - Box Collider

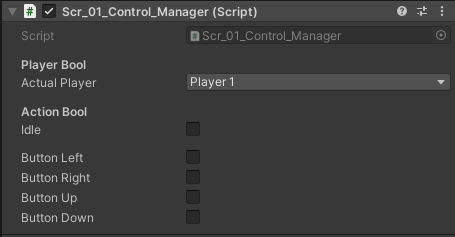
Determines the space of the ground detection for the Scr\_02\_State\_Manager, it always goes at the bottom of the character.

The Collider is a Trigger because we don’t need to interact with solid objects. We use *Center* and *Size* to move the boundaries.



### 1.5 - Scr\_01\_Control\_Manager

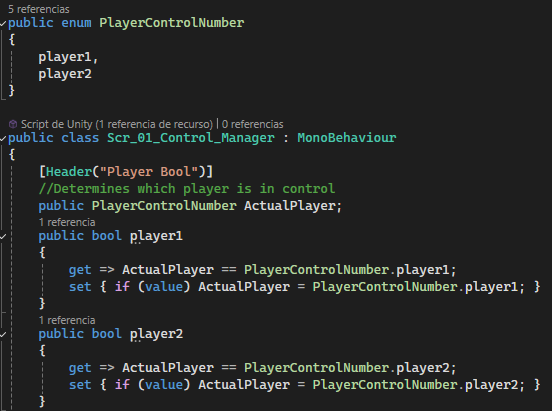
Is a universal Scrpit that manages the inputs of Player 1 and Player 2.



*Actual Player:* Determines the set of controls of that object, the options are *Player 1 and Player 2.*

*Action Bool:* Depending on the set of controls, when an input is detected, its respective box will be marked. When no input is detected the default box *Input* will be marked.

#### Code: Player Asignation

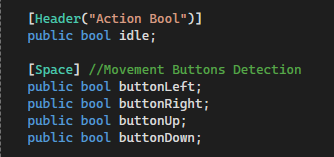


PlayerControlNumber contains the number of control configurations that may exist.

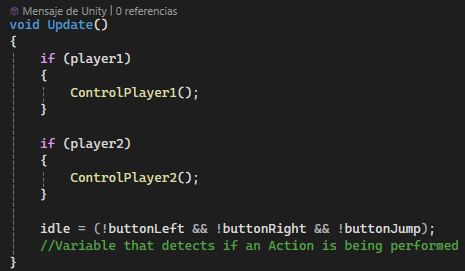
ActualPlayer Determines the set of controls of that object.

The rest of the code works so that it appears in the Unity UI and that only one option can be chosen.

#### Code: Inputs

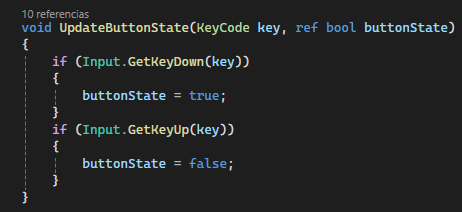


Action Bools refers to all the buttons on the control.

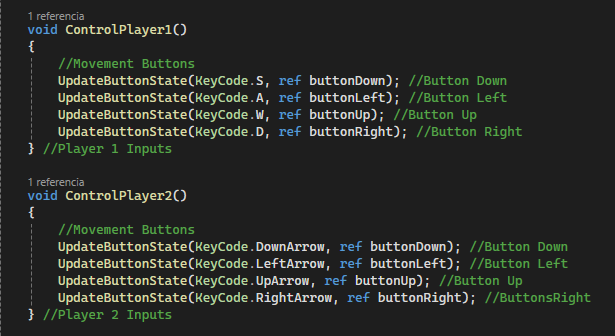


The Update method checks what bool is active (player1 or player2) and assigns the controls stored in methods ControlPlayer1() or ControlPlayer2().

Idle is activated when the main inputs are not activated, the more main inputs exist, the longer the section after the equal sign will be.



The UpdateButtonState method is what detects the press of the buttons. The parameter KeyCode takes the input (Ej: KeyCode.A, KeyCode.UpArrow) and buttonState refers to the the Action Bool that activates (Ej: buttonUp, buttonJump).

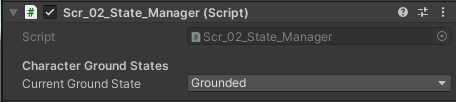


The ControlPlayer1 and ControlPlayer2 method is where the control inputs are gathered.

Assigning to the UpdateButtonState method the key and the bool of the corresponding action.

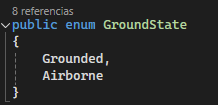
### 1.6 - Scr\_02\_State\_Manager

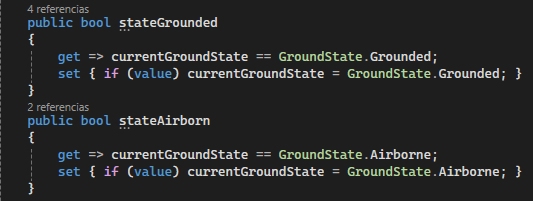
Is a universal Scrpit that manages states of the Object.



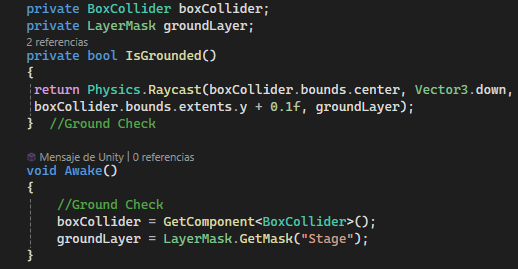
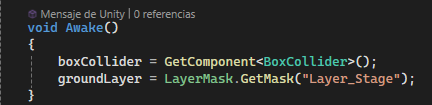
*Current Ground Check* determines if the player is *Grounded* or *Airborn*.

#### Code: Ground Check

**

**

The ground check works with two bools stateAirborn and stateGrounded, these activate depending on what state the object is in.

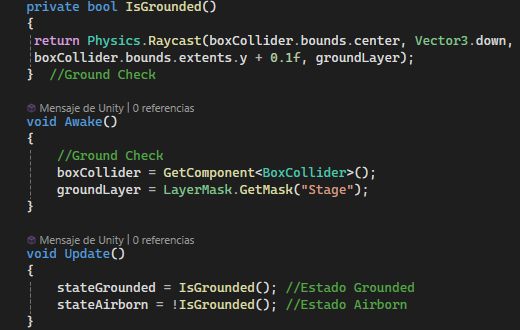
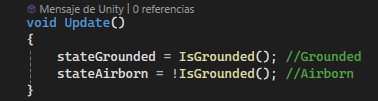
 

The detection works by combining the variable boxCollider and groundLayer.

The variable boxCollider is assigned to the component 1.4 - Box Collider, which determines the location and size of the detection box.

The variable groundLayer determines which layer the boxCollider should interact with to activate and deactivate bools, in this case, "Layer\_Stage".

The bools stateAirborn and stateGrounded will work with every object with the Layer “Layer\_Stage”.

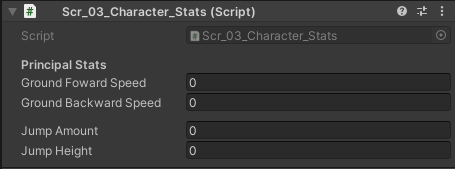
IsGrounded() is the principal method of the ground check.

It performs a ray cast from the center of the boxCollider downwards that detects objects with the Layer “Layer\_Stage”.

If this method is true, the bool stateGrounded is true, if the method is false, the bool stateAirborn is true.

### 1.7 - Scr\_03\_Character\_Stats

Is a universal Scrpit that manages the numeric value of every stat the character has.



*Ground Foward Speed:* Determines the velocity of the character moving forward in the ground.

*Ground Backward Speed:* Determines the velocity of the character moving backward in the ground.

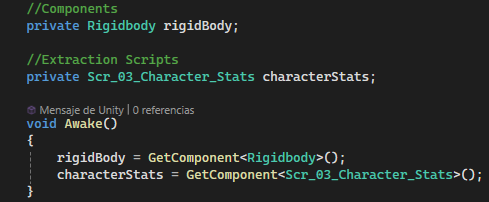
*Jump Amount:* Determines the amount of jumps a character has, the minimum is 2.

*Jump Height:* Determines height and distances of the jumps.

### 1.8 - Scr\_04\_Universal\_Physics\_Manager

Is a universal Scrpit that manages the basic physics of a character, those that everyone shares without exception.

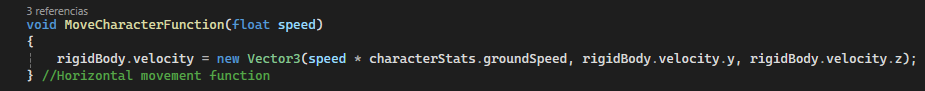
#### Code: Components



To control the object's movement, we need to manipulate the X and Y axis of the rigidBody.

The characterStats determines the parameters that will be used to control the character.

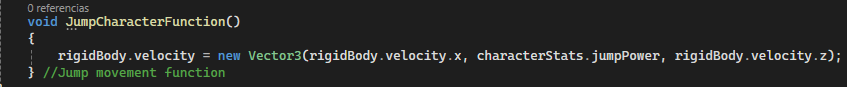
#### Code: Character Walk

**

The MoveCharacterFunction(speed) function is what makes the character move horizontally.

Take the Speed ​​parameter and determine the movement in the X axis of rigidBody.velocity. The two other axes stay the same.

#### Code: Character jump



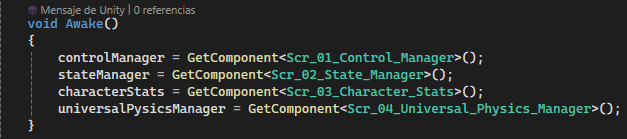
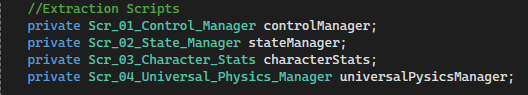
The JumpCharacterFunction() function is what makes the character jump.

Uses the characterStats.jumpPower in rigidBody.velocity to move the character.

### 1.9 - Scr\_05\_Universal\_Action\_Manager

Is a universal Scrpit that manages the basic actions of a character, those that everyone shares without exception.

#### Code: Components



The controlManager detects which key is being pressed, determining the character's action.

The stateManager determines the state of the character.

The characterStats determines the parameters that will be used to control the character.

The universalPysicsManager manages the physics of actions shared between all characters.

#### Code: Actual Action

The variable actualAction is a string that shows what action is being performed by the character.



We have to update it since other scripts are going to use the variable. The most important actions are documented **[03 – List of Labels -> 03 – Character Actions].**

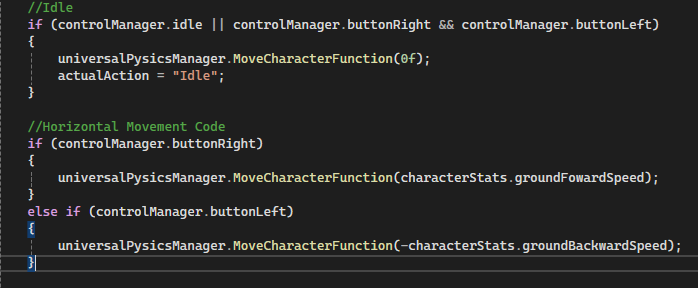
#### Code: Void

We use the method FixedUpdate() to code everything related to the character's physics.

#### Code: State Grounded

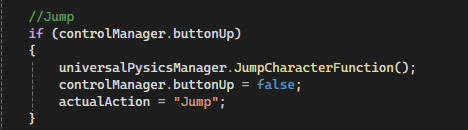
We put our code in stateManager.stateGrounded if we are performing an action on the stage.

The horizontal movement is composed of a series of If statements.



The first two, check what button is being pressed, if controlManager.buttonWright or controlManager.buttonLeft, which determines whether the value ​​is positive or negative.

The if that combines controlManager.buttonWright and controlManager.buttonLeft prevents the character for moving if both or none inputs are pressed.



controlManager.buttonUp is set to false so that doesn't trigger another jump right away.