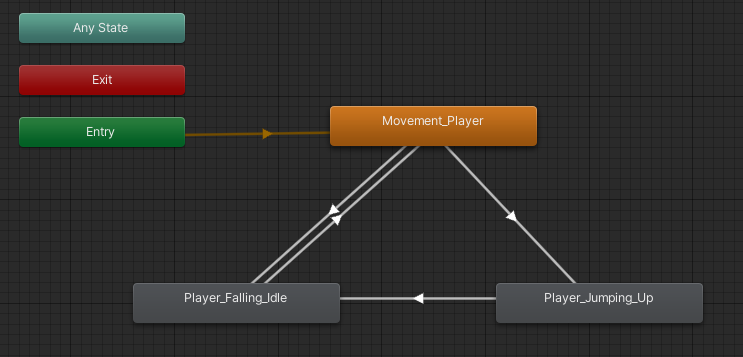
Lógica salto Player

Para poder realizar la lógica del salto del Jugador se han realizado los siguientes pasos:

Buscas animaciones en mixamo:

* Falling Idle
* Jumping Up

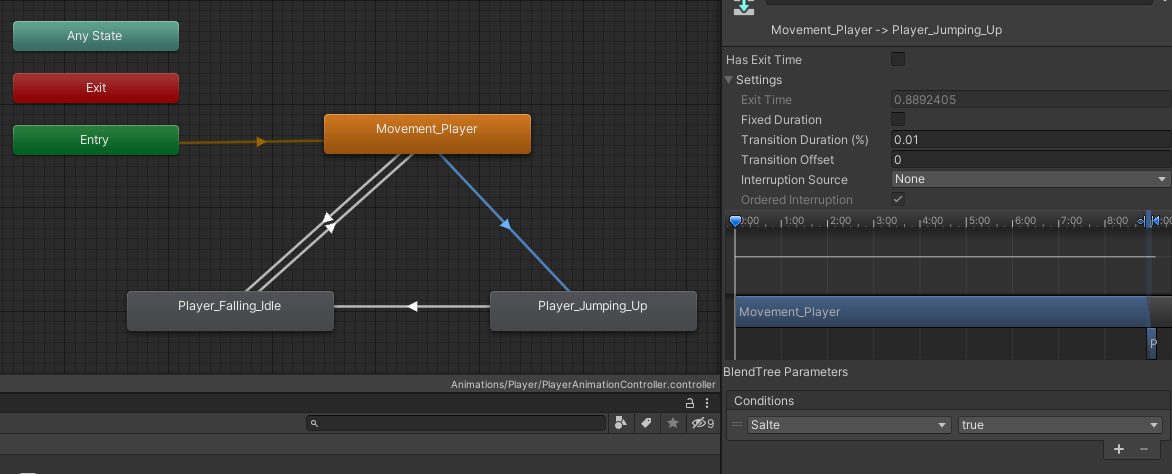
Las animaciones se han incluido en el animator del jugador de la siguiente manera:

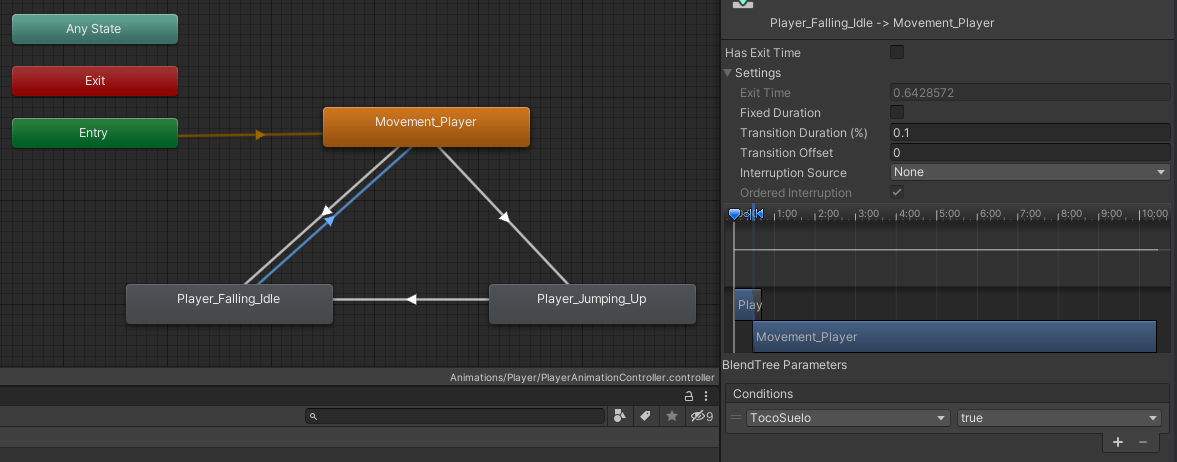


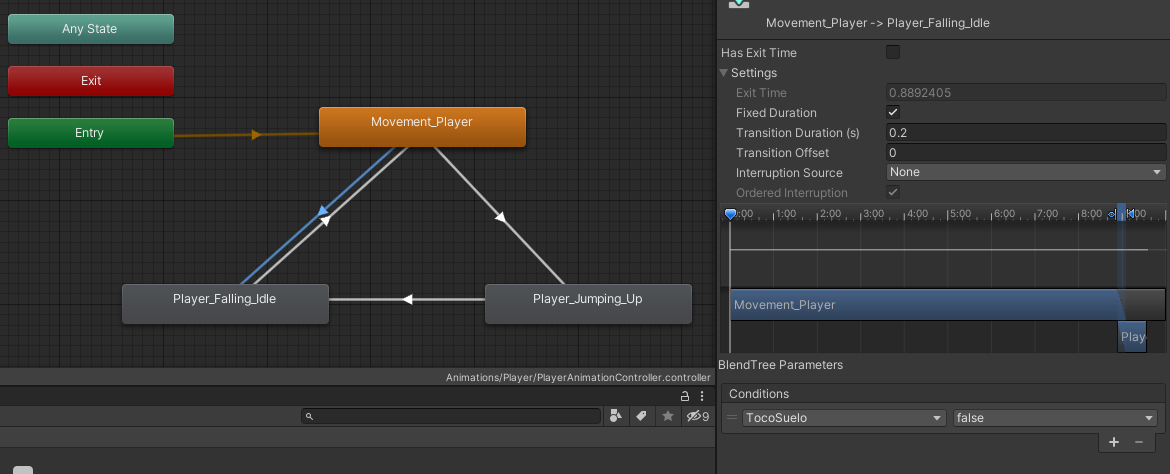
En el animator crear 2 parámetros de tipo bool que hemos llamado Salte y TocoSuelo



Lógicas de las transiciones:



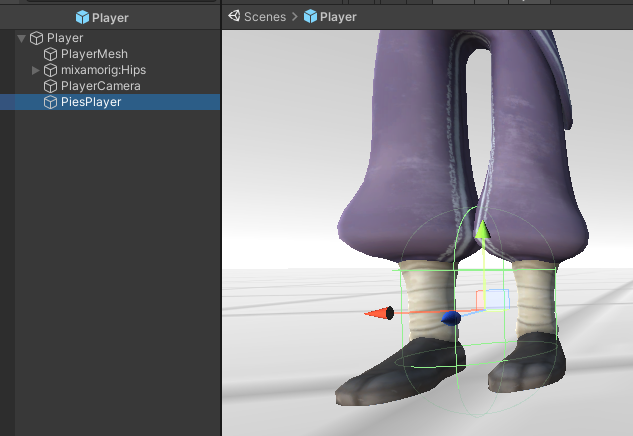




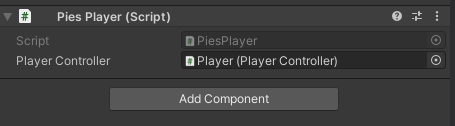
Dentro de la jerarquía del jugador se ha creado un empty en la posición 0 el cual hemos llamado PiesPlayer.

En este empty se ha añadido un capsule collider copiado del capsule collider general del jugador, en el capsule collider activamos la opción de **Trigger**.

Seguidamente se ha ajustado a un tamaño correcto



Por otra parte este empty tiene un script asignado el cual hemos llamado PiesPlayer



### Lógica **PiesPlayer.cs**

| **using System;**  **using System.Collections;**  **using System.Collections.Generic;**  **using UnityEngine;**  **public class PiesPlayer : MonoBehaviour**  **{**  **public PlayerController playerController;**  **private void OnTriggerStay(Collider other)**  **{**  **playerController.CanJump = true;**  **}**  **private void OnTriggerExit(Collider other)**  **{**  **playerController.CanJump = false;**  **}**  **}** |
| --- |

Por otra parte la lógica del jugador tiene el siguiente script:



**PlayerController.cs** contiene la siguiente lógica para el salto:

**Variables** implicadas en la lógica de salto:

| private const string **SALTE\_STR** = "Salte";  private const string **TOCO\_SUELO\_STR** = "TocoSuelo";  private bool canJump; |
| --- |

### Dentro **del Update()** miramos si podemos saltar de la siguiente manera

| *// Comprobamos si el jugador puede saltar y aplicamos la lógica correspondiente*  if (canJump)  {  if (Input.GetKeyDown(KeyCode.**Space**))  {  Jump();  }  \_animator.SetBool(**TOCO\_SUELO\_STR**, true);  }  else  {  IAmFalling();  } |
| --- |

### El método **Jump()** contiene la siguiente lógica:

| */// <summary>*  */// Método para aplicar una fuerza de impulso al jugador y simular un salto*  */// </summary>*  private void Jump()  {  \_animator.SetBool(**SALTE\_STR**, true);  \_rigidbody.AddForce(Vector3.up \* jumpForce, ForceMode.**Impulse**);  } |
| --- |

### El método **IAmFalling()** contiene la siguiente lógica:

| */// <summary>*  */// Método que analiza si el jugador esta en el aire*  */// </summary>*  private void IAmFalling()  {  \_animator.SetBool(**TOCO\_SUELO\_STR**, false);  \_animator.SetBool(**SALTE\_STR**, false);  } |
| --- |

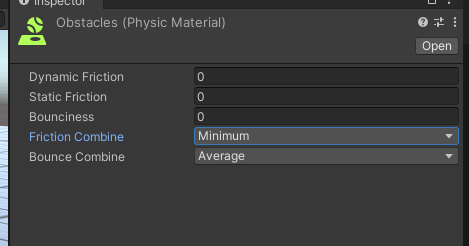
Resumen:

El [script](#_p9zqya3qpqb7) que tiene los pies del jugador comprueba, si el collider está colisionando con algo, en caso que esté colisionando permitirá al jugador saltar a indicando el valor de la variable **canJump** a true (ya que quiere decir que está pisando suelo), en el caso contrario dicha variable tendrá el valor de false y no podrá saltar.

En la lógica [Update](#_k5b7m9ml0ebu) del script de playerController, se comprueba dicha variable **canJump** para permitir al jugador saltar o no.

En caso que sí y que el jugador pulse la tecla Space ejecutará el método [**Jump()**](#_o2p51zrstgbn) y de lo contrario, si el jugador no puede saltar quiere decir que se está ejecutando el método [**IAmFalling()**](#_24qcefz31goy)**.**

Para evitar colisiones erróneas cuando chocamos con obstáculos creamos un physic material con las siguientes propiedades:



Este material podemos asignarlo en los obstáculos de la siguiente manera:

