UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning
Secretaría de Extensión y Cultura Universitaria

III.11. UNIDAD DIDÁCTICA

<< Desarrollo de Videojuegos >>

Centro de Formación, Investigación y Desarrollo de Soluciones de e-Learning.
UTN - FRBA. Secretaría de Cultura y Extensión Universitaria
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148 // *e-learning@sceu.frba.utn.edu.ar*
www.sceu.frba.utn.edu.ar/e-learning

<< Modulo III Avatar II >>

Trabajar sobre personajes

Presentación:

En esta unidad nos ocuparemos de los aspectos relacionados a la configuración del avatar para poder realizar movimientos y transiciones entre distintos tipos de movimientos.

Objetivos:

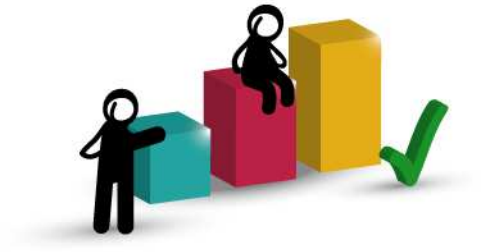
Que los participantes logren:

Agregarle a los personajes movimientos creados por ellos o importados de trabajos de terceros.

Obtener transiciones entre movimientos

Bloques temáticos:

1. Movimientos de Avatar
2. Trabajar con transiciones
3. Trabajar con transiciones con dependencias
4. Inspeccionamos scripts asociados a FPC



Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

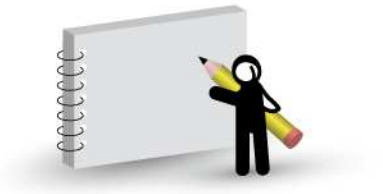
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning constructivista colaborativo de nuestro Centro.***



Tomen nota*

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

**** Está página queda como está. El contenidista no le quita ni le agrega nada.***

Bloque temático 1: Movimientos de Avatar

Repasando lo realizado al animar nuestro primer Avatar, necesitamos:

1.- Un gameObjet (El cual hemos creado en MakeHuman y animado en Blender)

2.- Una animación (la cual creamos en Blender pero que puede ser descargada aparte)

3.- Un Animator Controller (el cual es utilizado para unir la animación al Game Object)

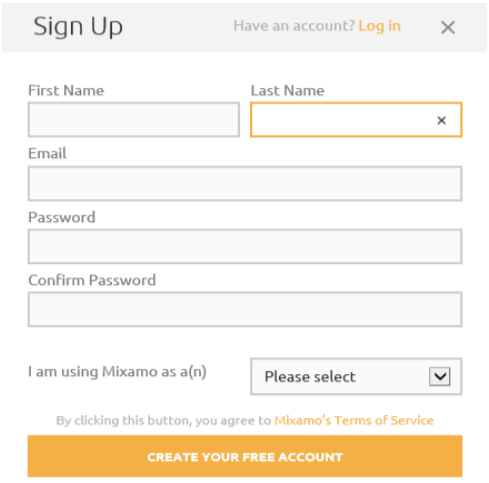
En su momento realizamos dentro del Animator Controller un State al cual le asignamos la animación y luego le asignamos al Game Object en el panel de jerarquía el Animator Controller para hacer que caminara.

Pero ¿qué pasaría si quisiéramos asignarle una animación a otro personaje?

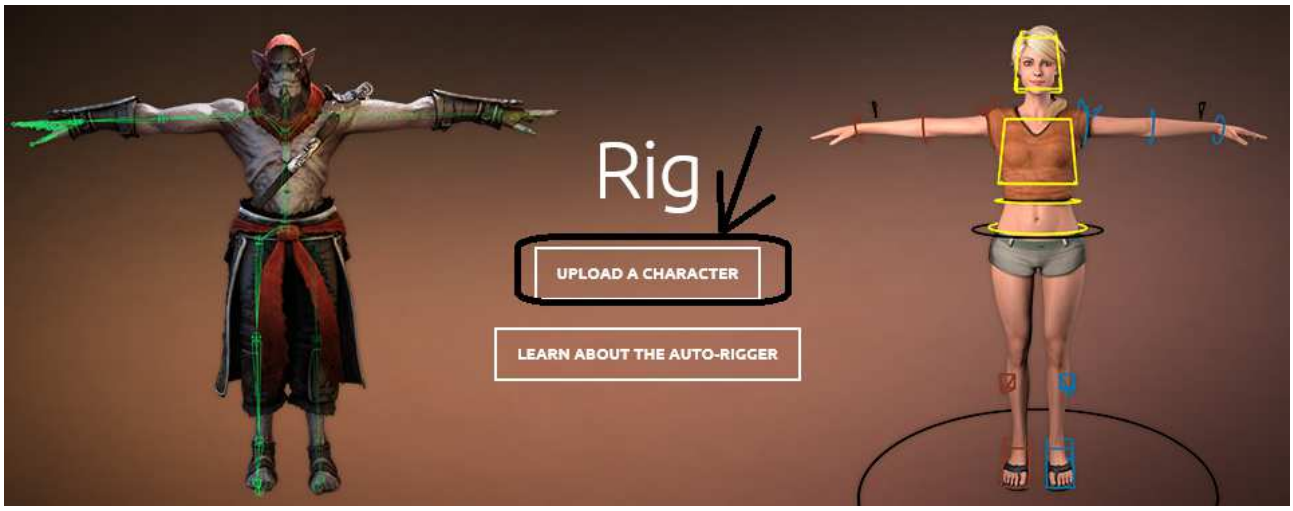
En realidad no hay ningún problema, lo único que tenemos que hacer es configurar el Avatar de ese nuevo personaje para que Unity reconozca el esqueleto del mismo, luego moverlo al panel de jerarquía, y finalmente asignarle el Animator Controller que posee la animación que queramos.

Parece simple, pero ¿Debo crear todas las animaciones que quiero quer realice un personaje?

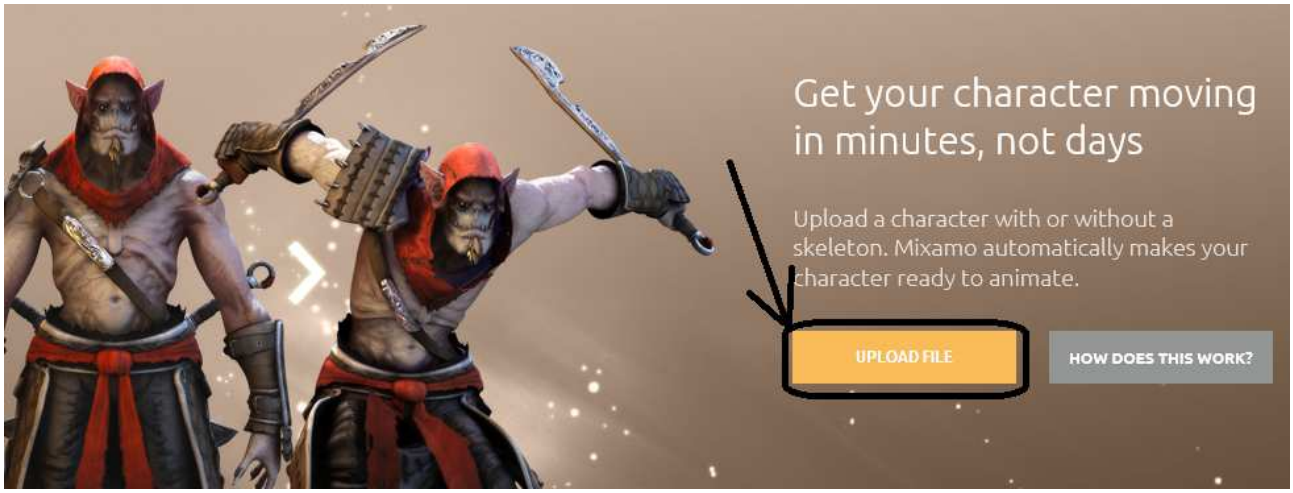
En realidad no, ya que puedo descargar las animaciones por separados de algún assets o de algún sitio en internet, por ejemplo *www.mixamo.com*. Mixano nos provee de personajes y animaciones que podemos utilizar en nuestros trabajos, algunos de uso libre y otros de pago, el primer paso es registrarnos



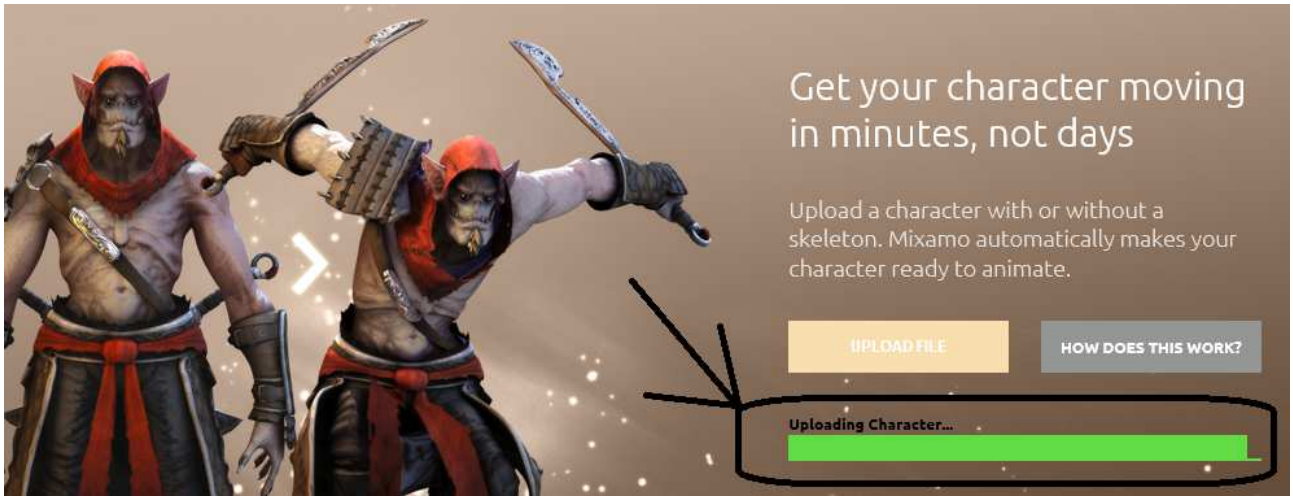
Para comenzar, podemos cargar el carácter que hemos realizado en MakeHuman, para eso vamos a upload a character.



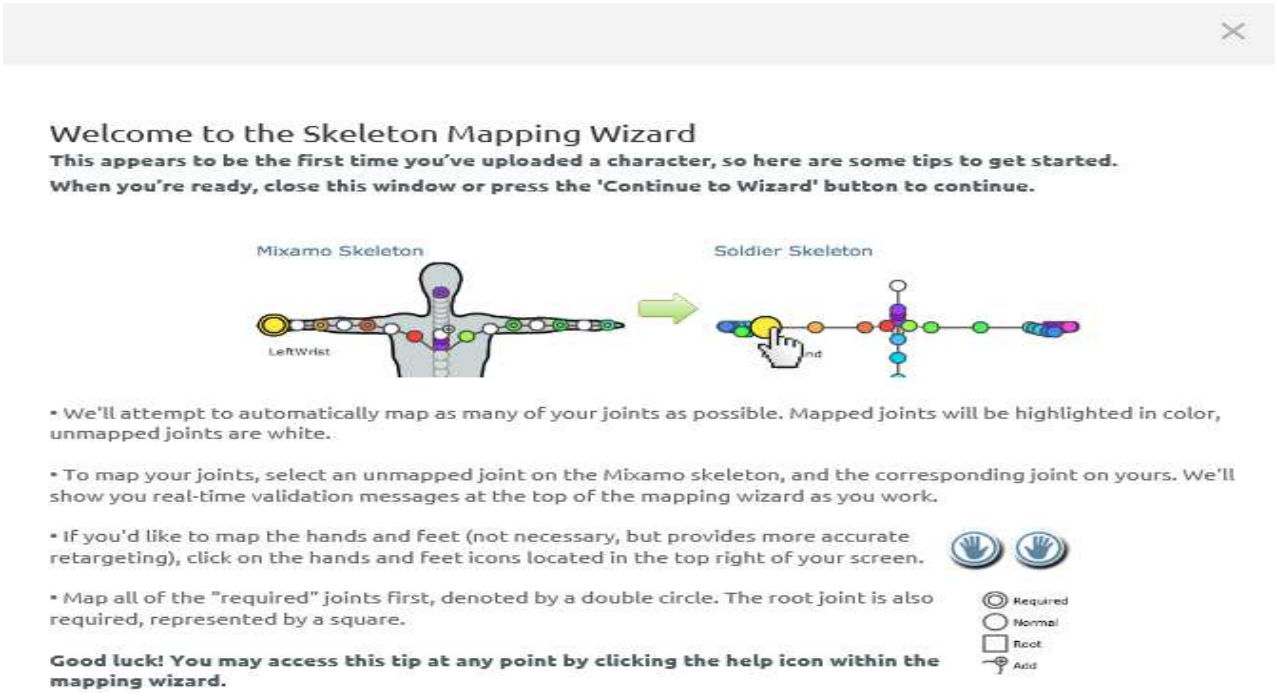
Luego vamos a upload file



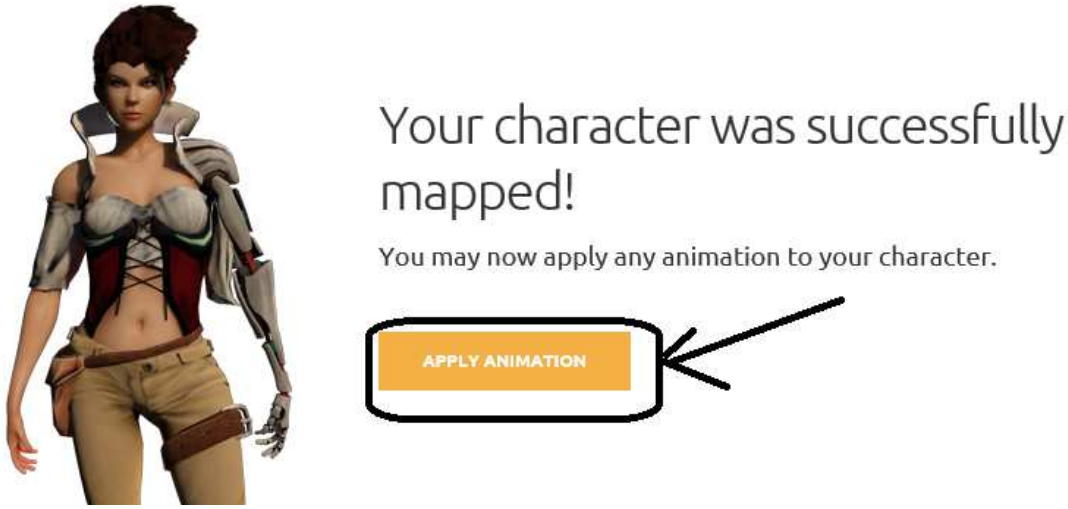
Aquí debemos cargar el personaje realizado en MakeHuman o Modificado en Blender en formato fbx.



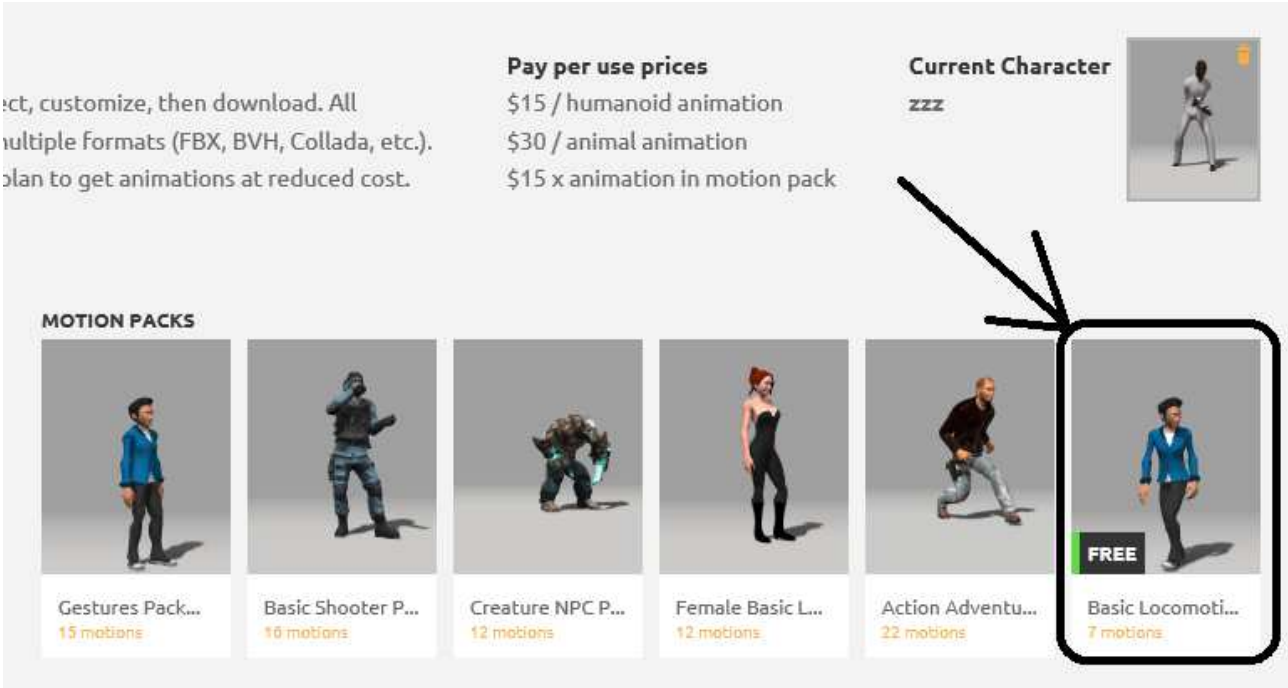
A continuación nos sale una descripción de como mixano interpreta a nuestro personaje si hiciera falta configurar a mano la compatibilidad entre esqueletos.



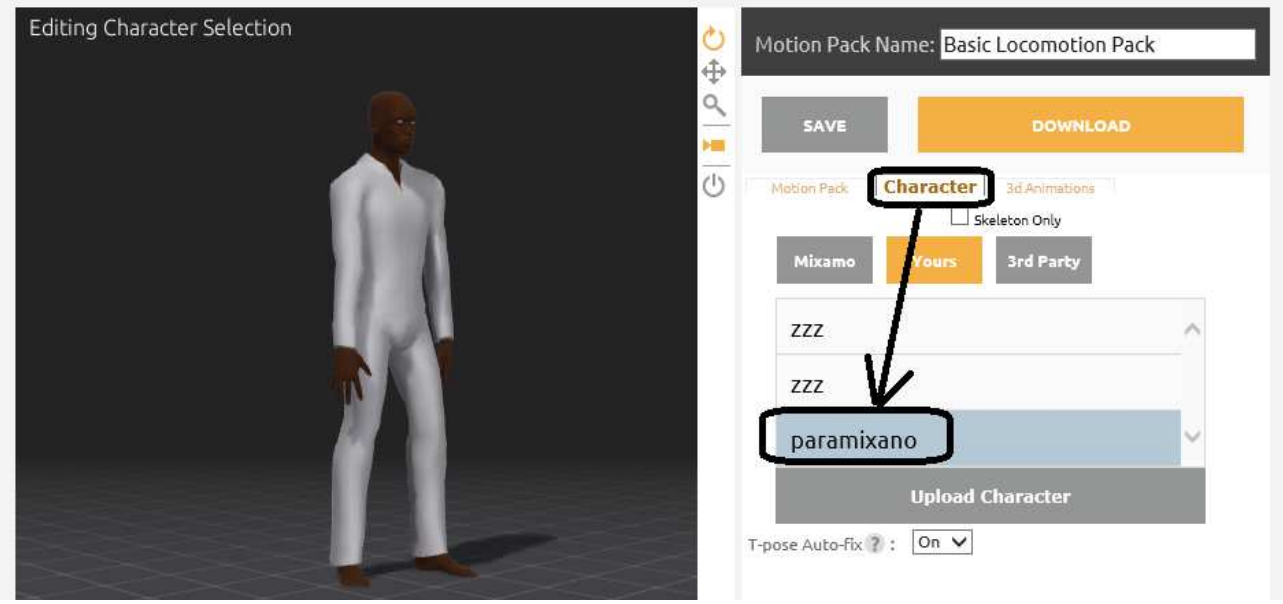
Si la compatibilidad fuera completa, directamente nos aparece una pantalla diciendo que la importación fue exitosa, y podemos comenzar a agregarle animaciones a nuestro personaje.



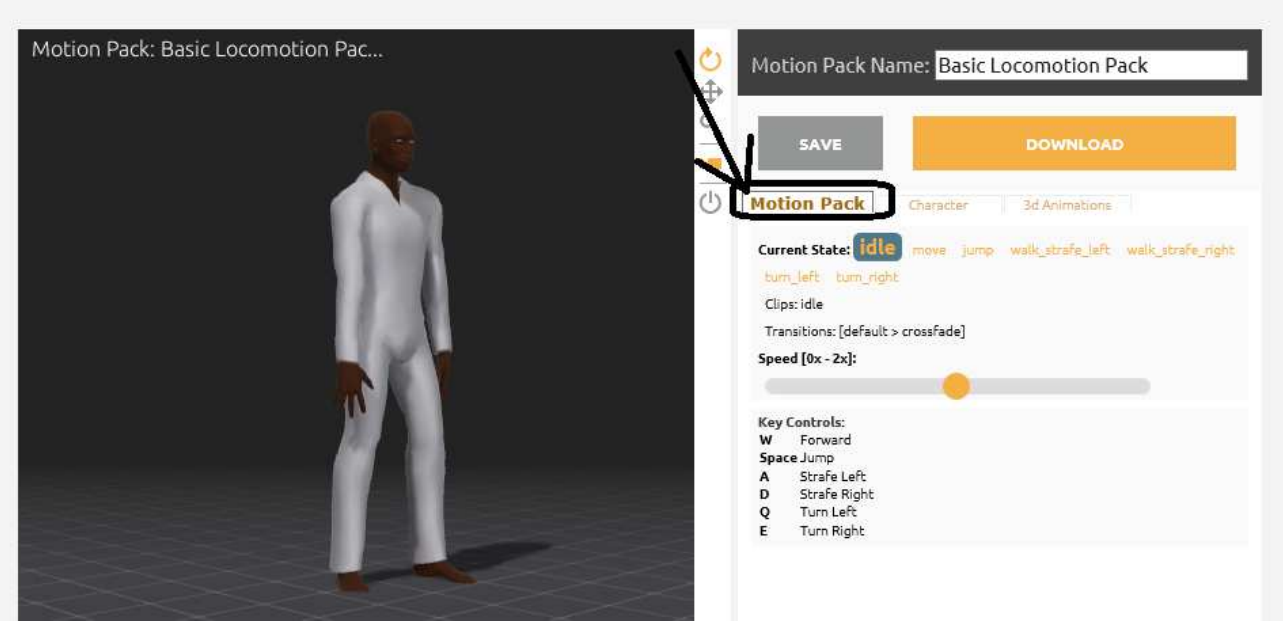
Aquí por unos pocos dolares, tenemos muchísimas animaciones, pero para probar vamos a descargar un paquete de 7 animaciones de uso libre.



Debemos a continuación indicar a cual de los personajes que hemos subido le vamos a aplicar las animaciones. La imagen aparece sin las texturas, lo cual no es problema ya que podemos agregarselas en Unity.

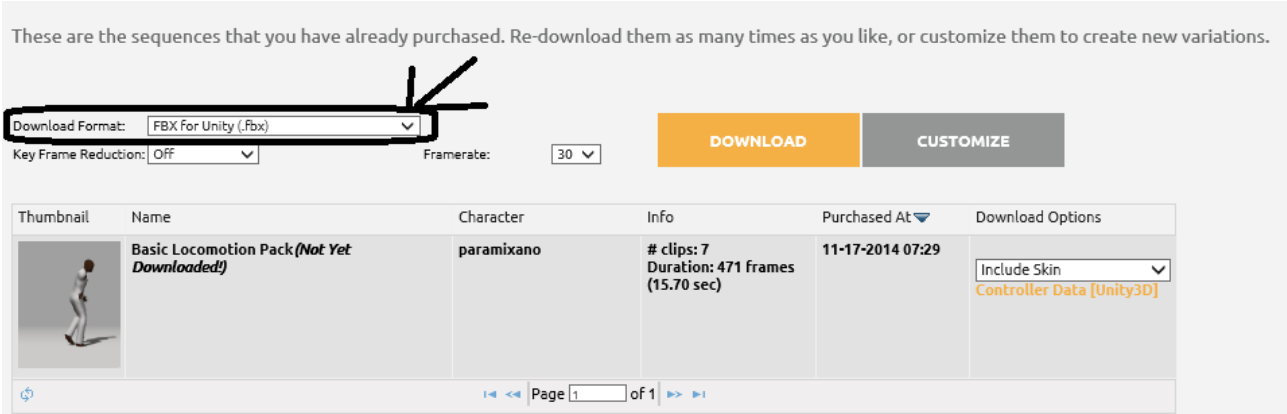


Luego en Motion Pack podemos visualizar las animaciones y ajustar la velocidad de movimientos.



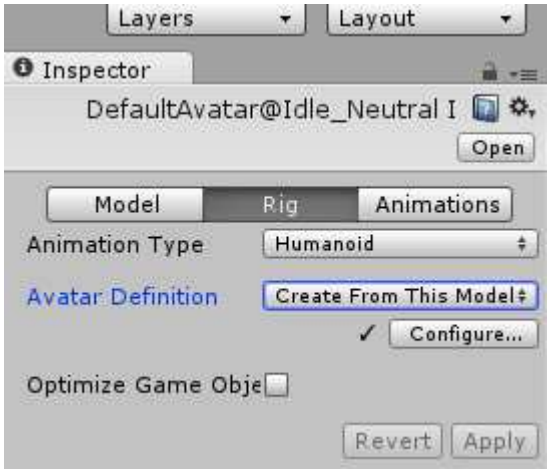
Al finalizar presionamos en Download

Es importante que elijamos formato de descarga fbx para Unity.

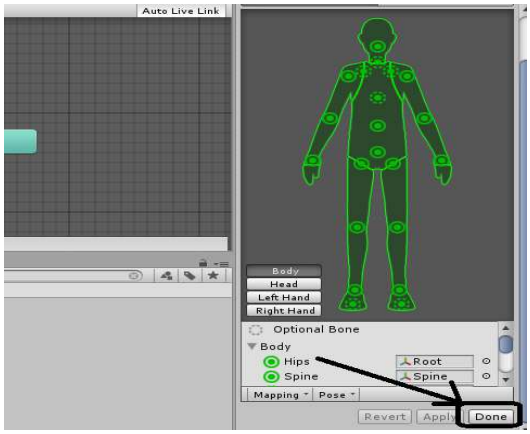


Ahora en Unity creamos un directorio dentro de Assets “personaje_mixano” y guardamos el archivo que hemos descargado, en donde se guardan también el material de la piel.

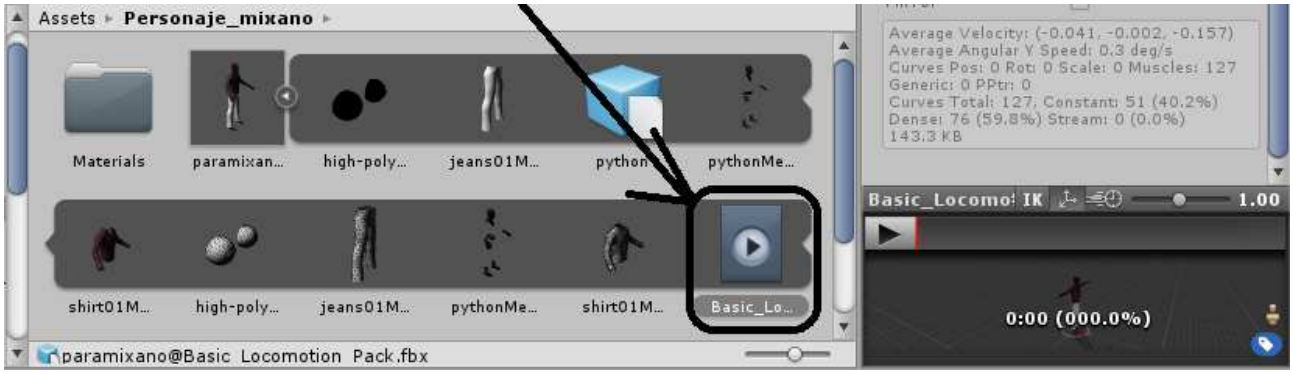
De igual forma que lo hicimos antes, debemos ahora configurar nuestro Avatar, para lo cual vamos a Rig y seleccionamos los valores como se muestran a continuación. Luego de dar Aplay aparece el tilde en Configure, y ya podemos configurar el Avatar.



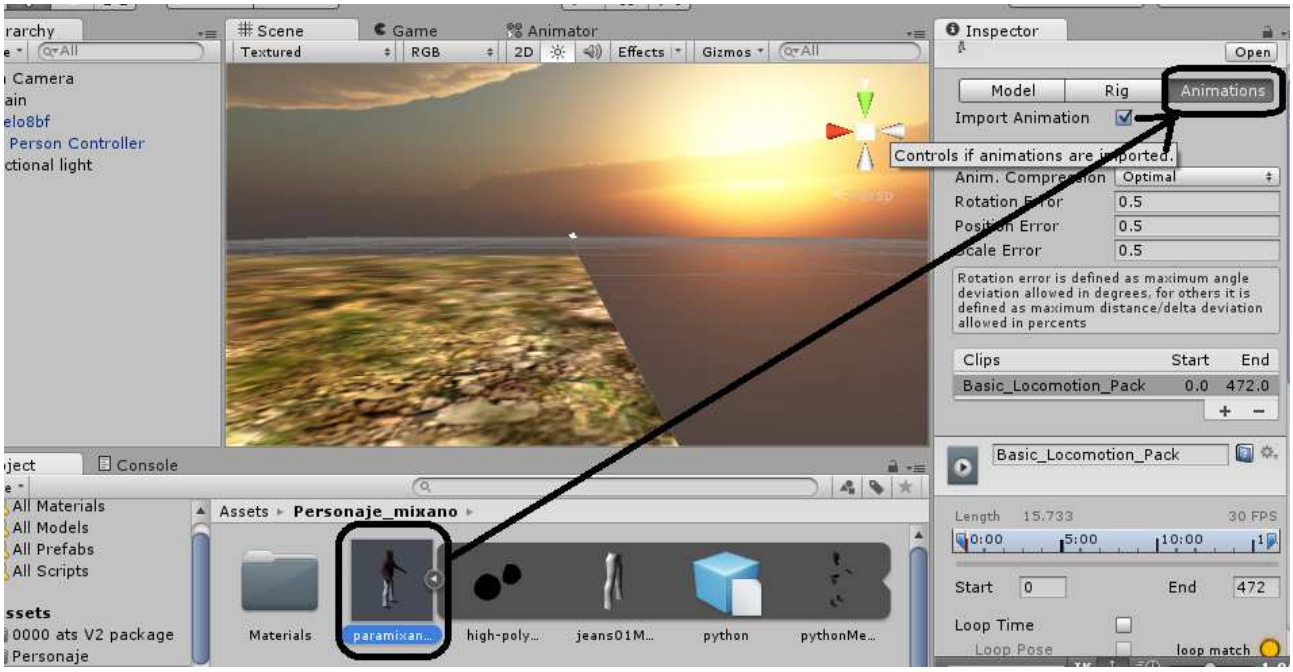
Si todo va bien, presionamos en Done.



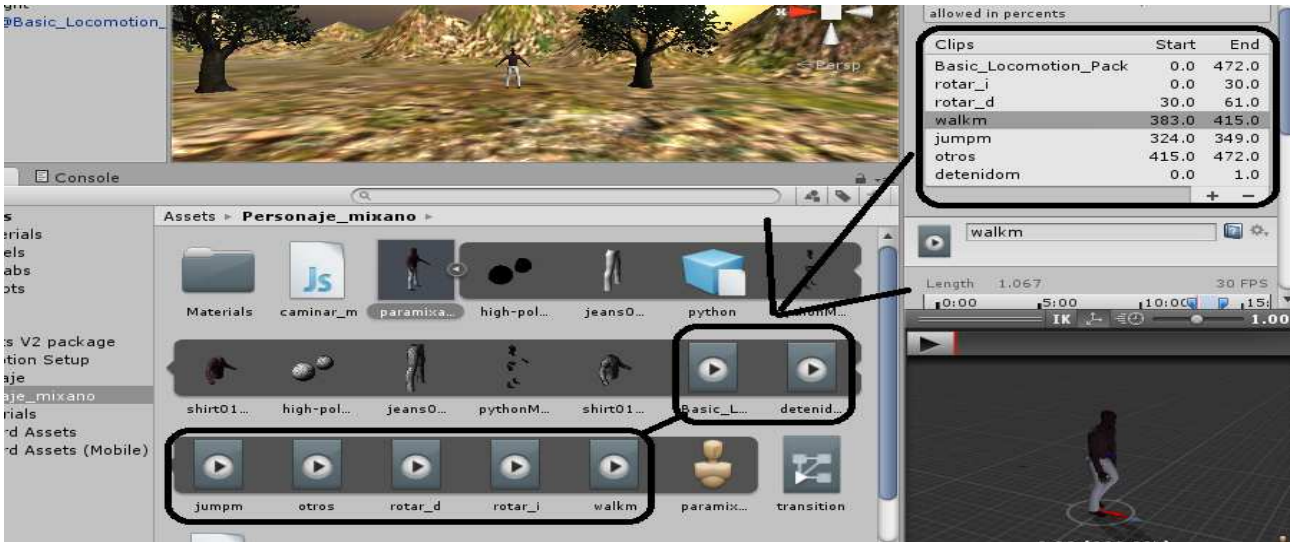
Una vez que tengamos configurado el Avatar, presionamos en el paquete de animaciones y este aparecerá en el panel de inspector, y mediante el botón de play podremos ver las animaciones.



Luego vamos al personaje y configuramos las animaciones en la solapa Animations del panel de Inspector.



Fraccionamos las animaciones, y al dar en aplicar estas se crean en el personaje



Cuando estamos creando las animaciones tenemos que tener en cuenta el significado de cada campo de configuración

1.- El campo Loop tildado hace que la animación se ejecute de forma cíclica.

2.- El campo Root Transform Rotation tildado, evita rotaciones durante la animación.

3.- El campo Root Transform Position (Y) tildado, evita desplazamientos verticales durante la animación, por lo que no tendría que estar tildado por ejemplo en la animación de un salto.

4.- El campo Transform Position (XZ) evita desplazamientos en el plano, por lo que si tenemos una animación de caminata en la cual el personaje se desplaza, y tildamos este campo, vamos a obtener el efecto de que el personaje camine en el lugar al ejecutar la animación.

Bloque temático 2: Trabajar con transiciones

Ya tenemos un par de personajes para trabajar, el que creamos en mixamo, con las siguientes animaciones:

1.- Posición detenido = detenidom

2.- Salto = jumpm

3.- Rotar a la izquierda = rotar_i

4.- Rotar a la derecha = rotar_d

5.- Caminar = walkm (esta caminata es con desplazamiento)

y el que creamos en Blender con las siguientes animaciones:

1.- Posición detenido = detenido

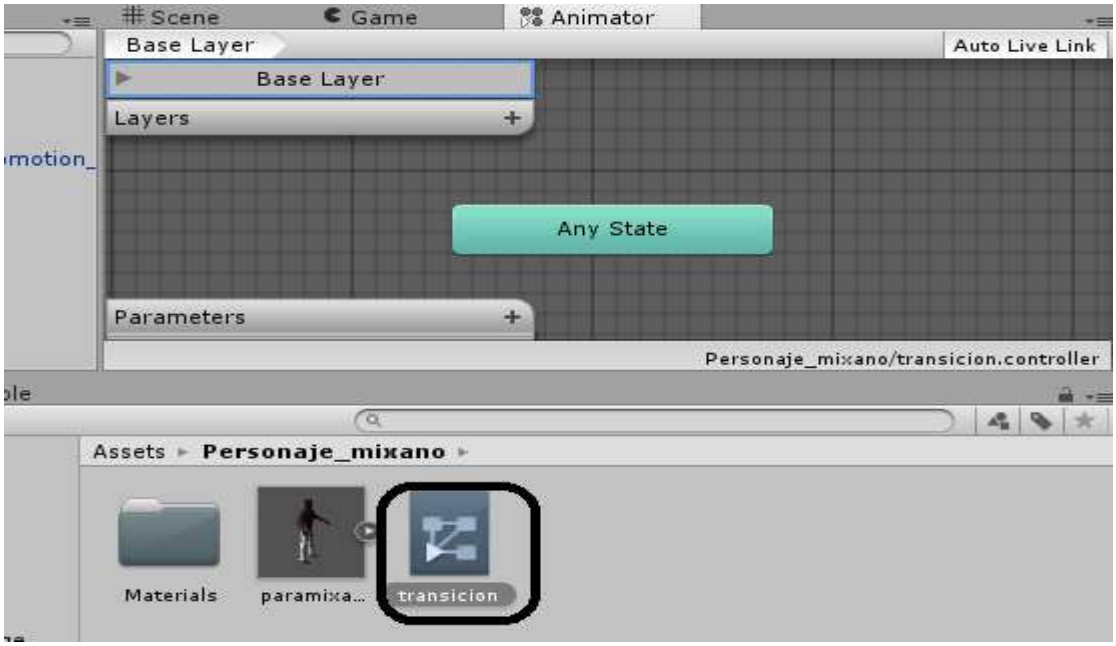
2.- saltar

5.- Caminar = walk (esta caminata es sin desplazamiento)

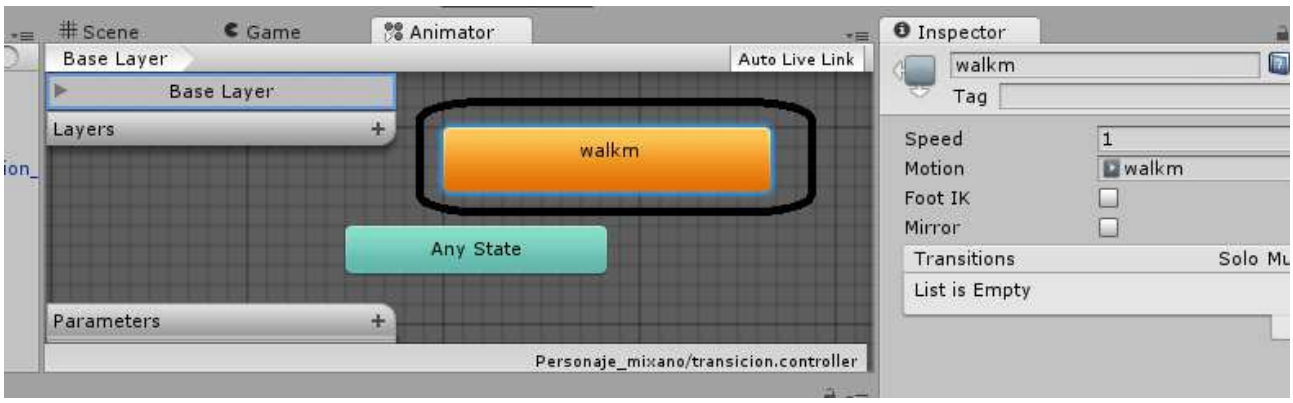
¿Como creamos animaciones controladas?

Este es un punto muy importante que se puede lograr de dos maneras, una de las cuales es trabajar con líneas de tiempo, dado que esta opción se encuentra disponible solo para Unity Pro, nosotros vamos a utilizar la otra forma, que es utilizar scripts y transiciones.

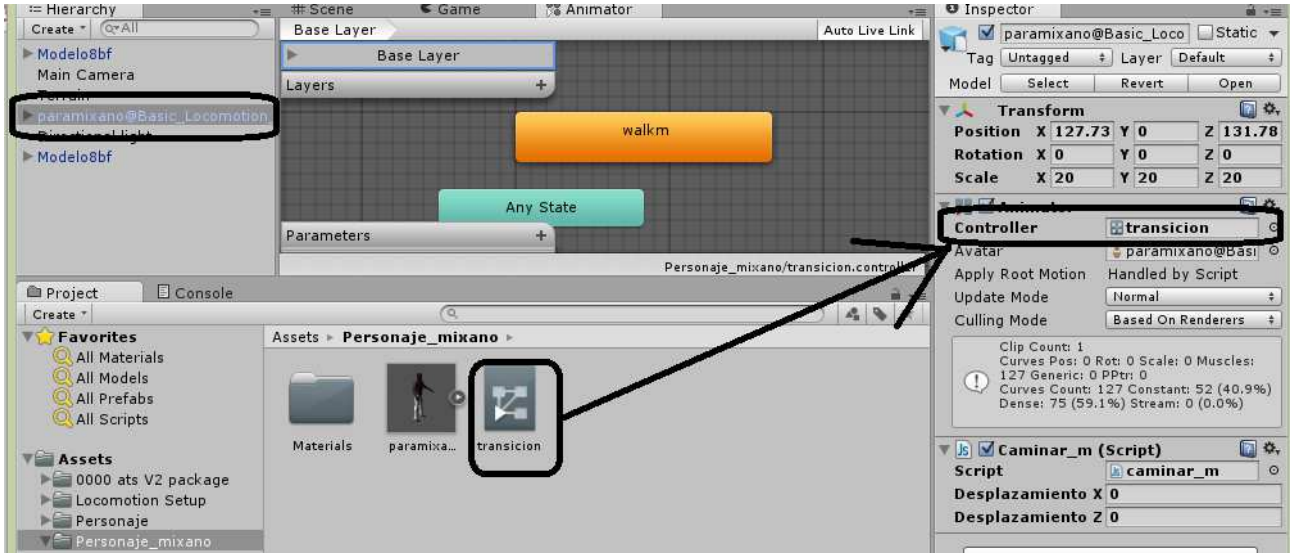
Comencemos por crear un Animation Controller, el cual llamaremos "**transicion**" (no utilizar acentos ni letra ñ para los nombres).



Ahora arrastro la animación de caminar a la ventana de animación, la cual queda marcada como animación por defecto



Arrastremos nuestro personaje al panel de jerarquía y en el panel inspector arrastremos el Animation Controller (transicion) al campo Controller del panel de Inspector (con esto estamos asociando la caminata al pesonaje)



Nota MUY importante: Una vez creado un Animation controller, este puede ser asociado a cualquier personaje que tenga bien configurado el Avatar.

Si ahora pasamos a modo juego, podemos ver que nuestro personaje camina.

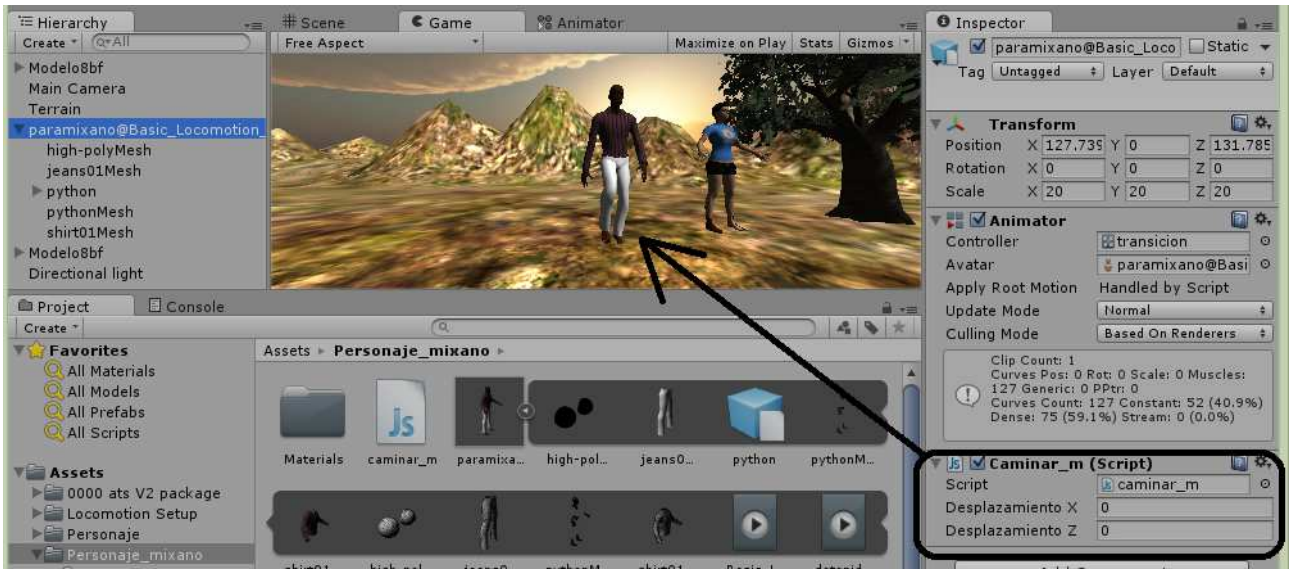
¿Esto está muy bien, pero como controlamos cuando camina?

Bien vamos por partes, primero creemos el siguiente script y se lo asociamos a nuestro personaje:

```
#pragma strict

public var desplazamientoX:float = 0;
public var desplazamientoZ:float = 0;
function OnAnimatorMove() {
     var animator : Animator = GetComponent(Animator);
    if (animator){
     var posicion:Vector3 = transform.position;
     posicion.x += desplazamientoX * Time.deltaTime;
     posicion.z += desplazamientoZ * Time.deltaTime;
     transform.position = posicion;
    }
}
```

En el mismo se crean dos variables para controlar el desplazamiento en x y en z, las cuales son accesibles en modo juego. Aquí hemos agregado un par de componentes nuevos, en primer lugar “OnAnimatorMove()” que es utilizado cuando queremos tener acceso al control de nuestro personaje, y en segundo lugar “Time.deltaTime” que nos permite obtener el desplazamiento recorrido en cada segundo.

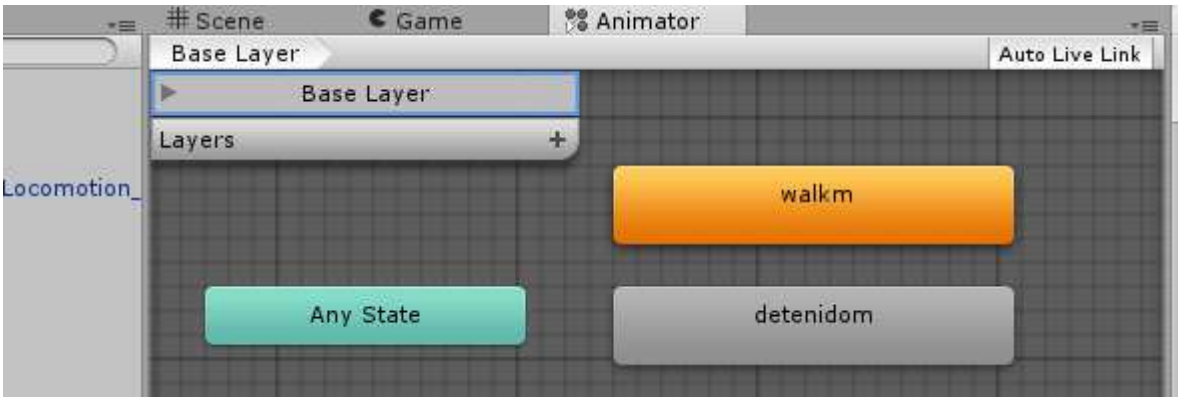
Al observar en modo juego el comportamiento de nuestro personaje, podemos ver que ahora camina en el lugar como si ubieramos tildado durante la configuración de la animación de caminar, el campo “Transform Position (XZ)”, con la diferencia de que ahora podemos modificar el deplazamiento en x o z desde el panel de inspector.



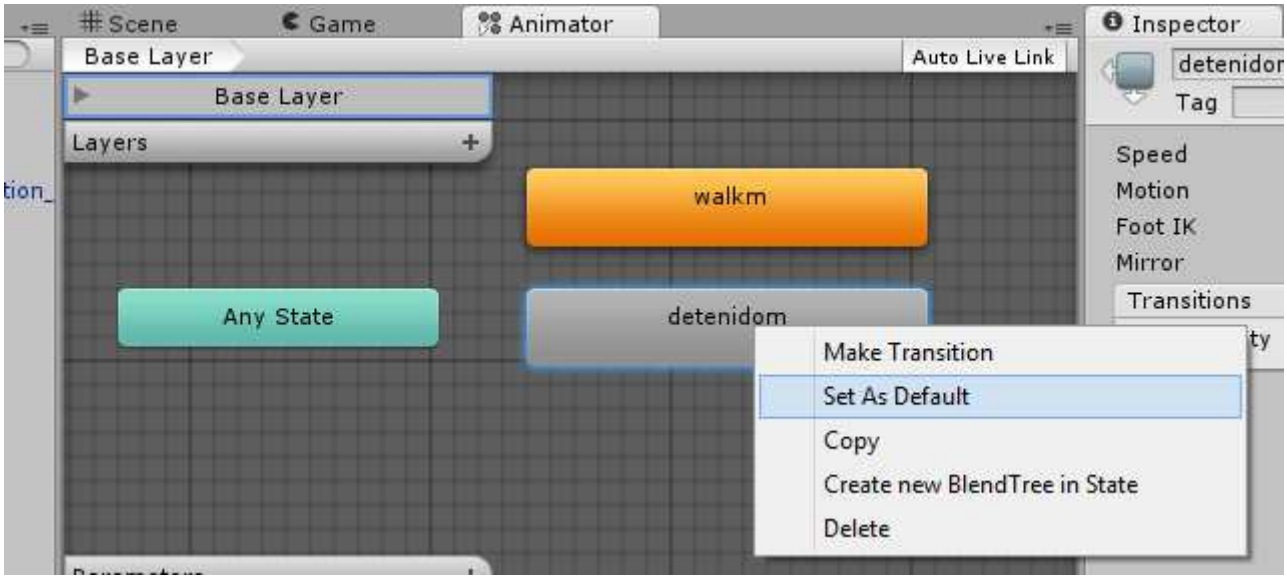
Ok ¿Pero como paso de una animación a la otra? y ¿como controlo la transición?

Bien, veamos ahora como crear una transición entre el estado detenido y caminando. Arrastremos a la ventana del “Animation Controller” que hemos creado (transicion) la acción “detenidom”.

Vemos que esta acción aparece en color gris, ya que la acción por defecto es “walkm”

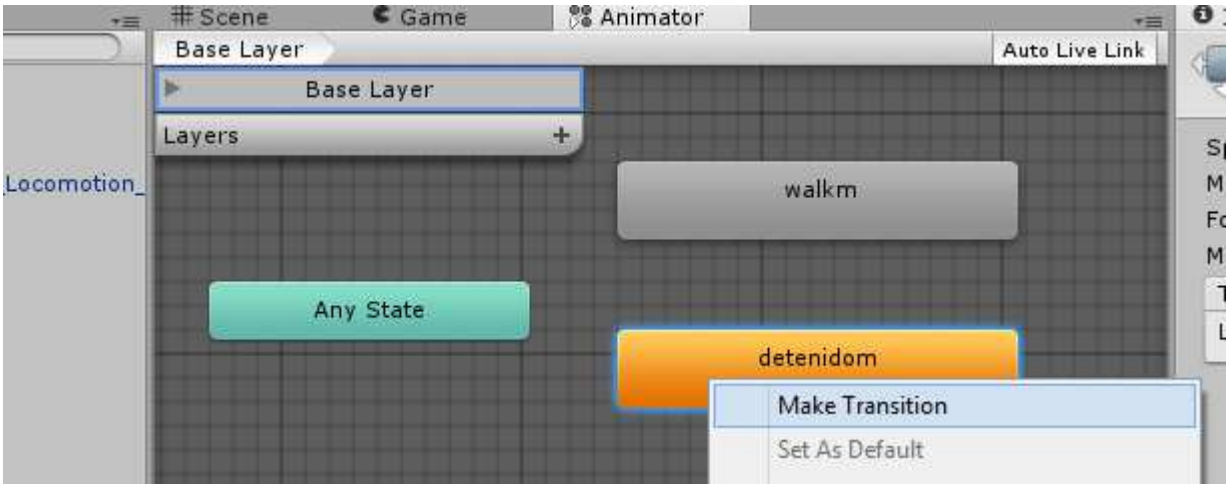


Cambiemos esto para que la acción por defecto sea la del estado detenido, para ello damos click sobre la animación detenidom con el botón derecho del mouse y seleccionamos “Set As Default”.

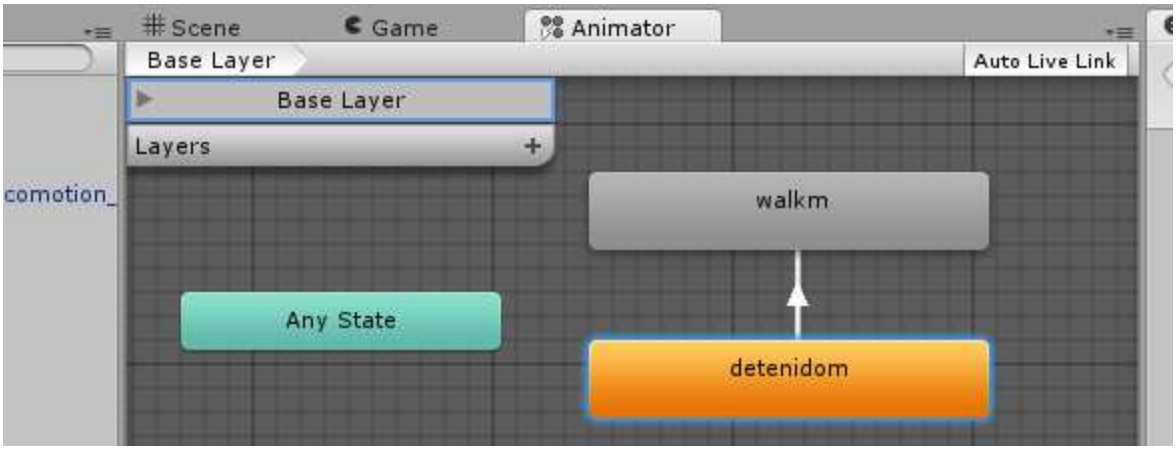


Los colores de las animaciones han cambiado y ahora se encuentran en color naranja “caminarm”, indicando que es la animación por defecto, si entráramos en modo juego ahora veríamos a nuestro personaje en modo de reposo.

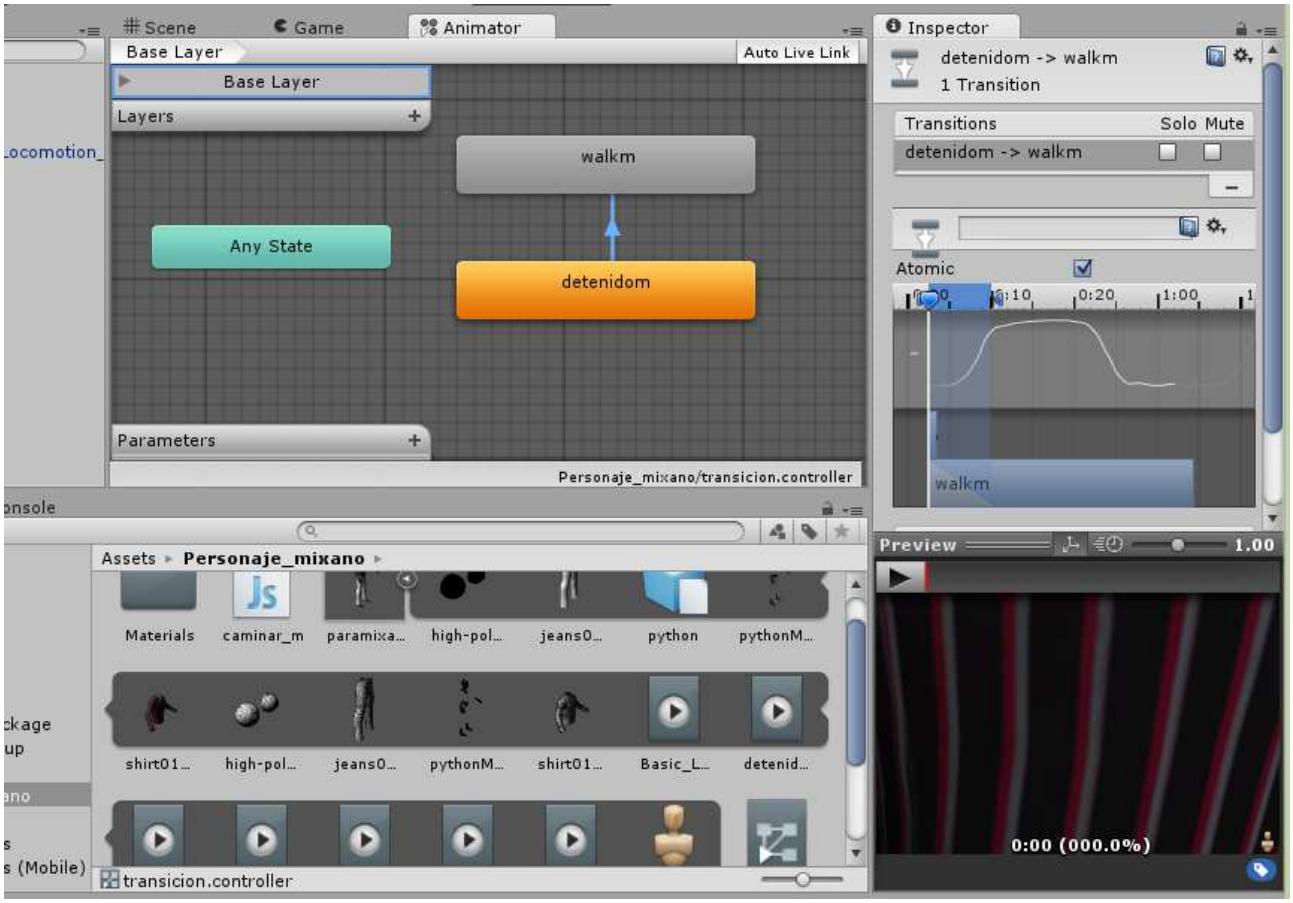
Para agregar una transición desde el estado detenido al de caminar, damos click derecho sobre la animación detenidom y presionamos “Make Transition”, ahora arrastramos la flechita que aparece hasta walkm y soltamos.



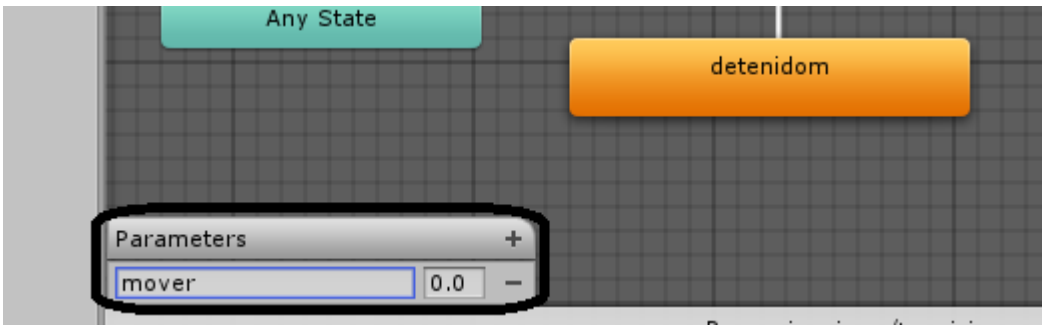
Nos tiene que quedar algo como lo siguiente



Una vez hecho esto podemos dar doble click sobre la flecha que une las animaciones, con lo que la misma se torna celeste indicando que podemos configurar la transición, y en la ventana de la derecha aparece en el panel de Inspector, las herramientas necesarias para su configuración.



Ahora debemos crear un parámetro para controlar la transición, con lo cual vamos a “Parameters” y presionamos sobre el signo “+”, para crear una variable de tipo “Float” la cual luego renombramos como “mover”



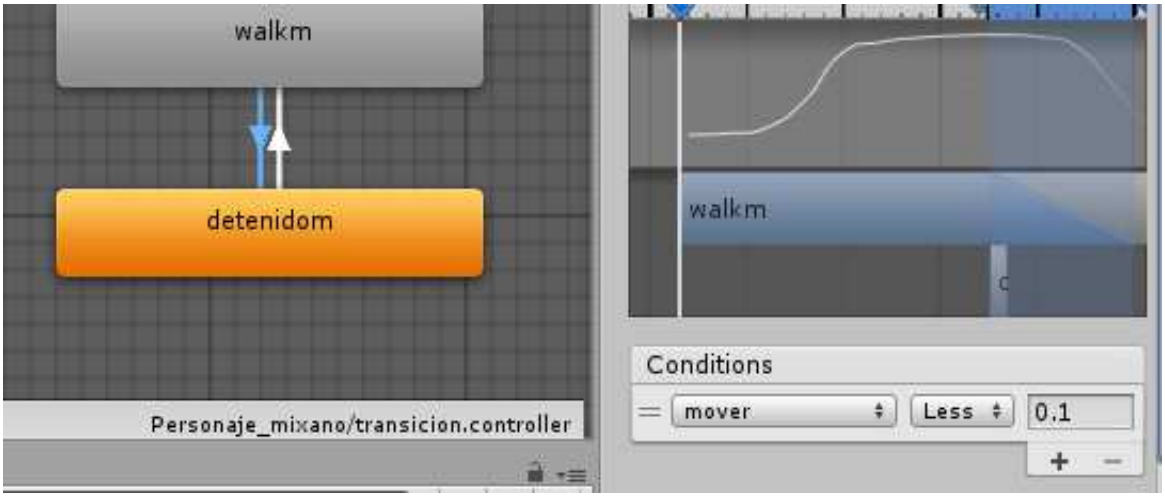
Hecho esto seleccionamos nuevamente la flecha de transición, y al final del panel de control modificamos el campo “Conditions” el cual determina el parámetro que va a controlar la transición y le damos un valor de 0.1, con lo que estaremos diciendo que cuando el valor de mover sea mayor a 0,1 se inicie la transición desde el estad de reposo al de caminar.

Si modificamos nuestro script con el código siguiente y entramos en modo juego, podemos ver que cuando presionamos la flecha vertical o la tecla “w” nuestro personaje se pone en movimiento.

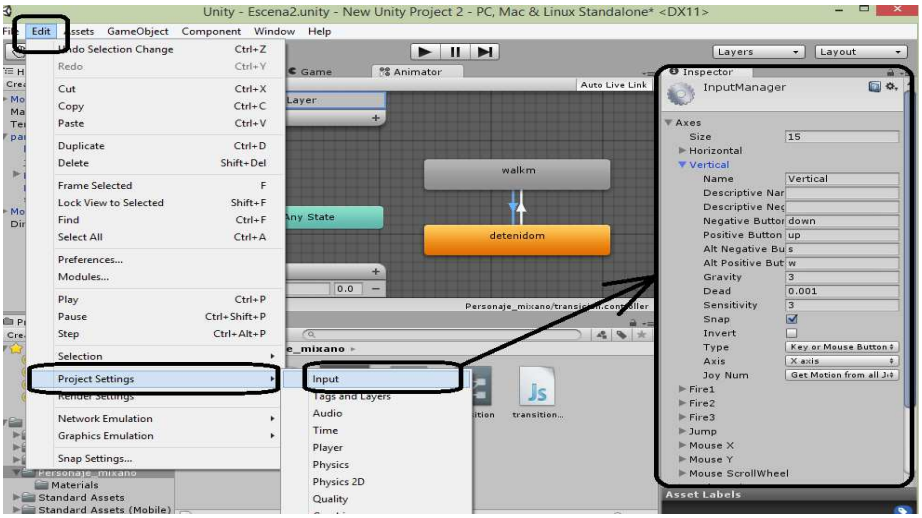
```
#pragma strict

public var desplazamientoX:float = 0;
public var desplazamientoZ:float = 0;
private var animator : Animator;
function OnAnimatorMove() {
     var       animator       :       Animator       =
GetComponent(Animator);
    if (animator){
     var posicion:Vector3 = transform.position;
     posicion.x += desplazamientoX * Time.deltaTime;
     posicion.z += desplazamientoZ * Time.deltaTime;
     transform.position = posicion;
    }
}

function Start () {
animator = GetComponent(Animator);
}
function Update () {
animator.SetFloat("mover", Input.GetAxis("Vertical"));
}
```

En el código agregado hemos creado una variable de tipo animator para poder tener acceso a la animación al declararla dentro de Setup() y posteriormente asociamos el movimiento de las teclas a la variable “mover” para que cuando el valor supere 0.1 el personaje se ponga en movimiento.

Podríamos ahora realizar otra transición del estado caminando al estado de reposo, para lo cual repetimos los pasos anteriores, pero ahora tomando como punto de partida “walkm”. La unica diferencia es que ahora podríamos poner que la transición se de cuando la variable “mover” sea menor a 0.1, con lo que la transición se de cuado suelto la flecha hacia arriba o la tecla w.



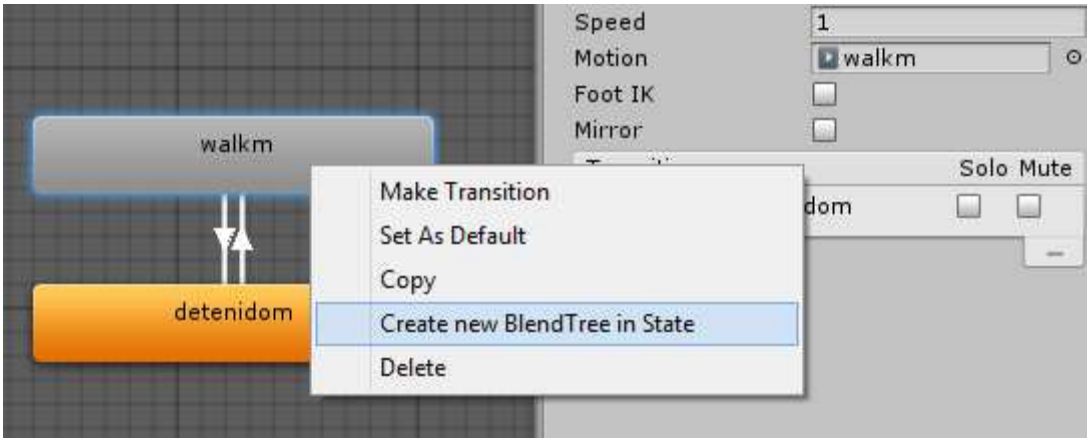
Para configurar los controles de inputs vamos a Edit > Project Seting > Input, con lo que nos aparece en el panel Inspector las opciones de configuración en cada eje.



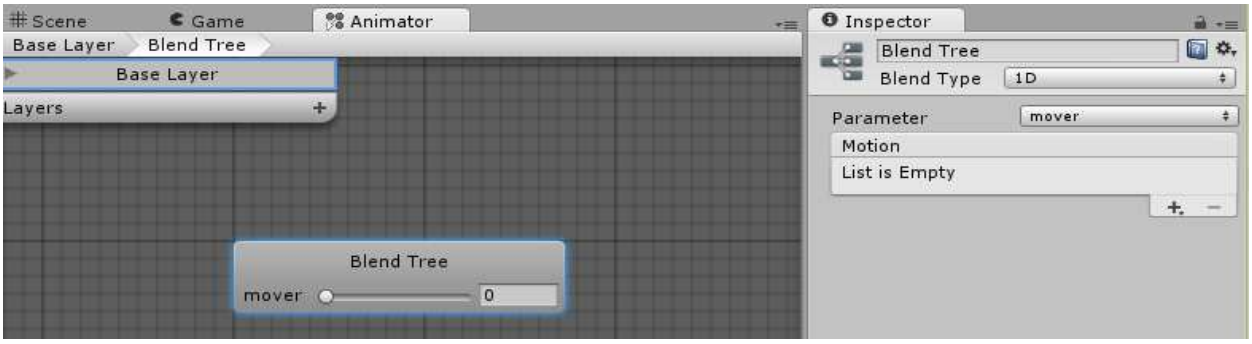
Bloque temático 3: Trabajar con transiciones con dependencias

Algo muy útil, es realizar que transición que presente dependencias, por ejemplo supongamos que tenemos una animación que es caminar y queremos que al presionar las flechas de hacia los costados el personaje rote, podemos realizar esto asociando las animaciones de rotación a la animación de caminar de la siguiente forma.

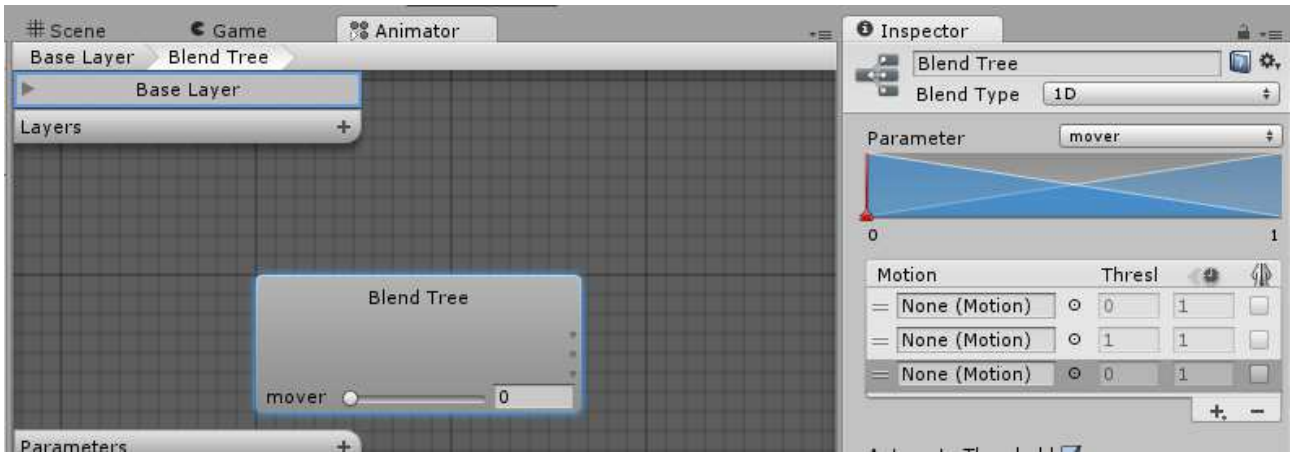
Paso 1: Hacemos click derecho sobre la animación de caminar y presionamos en “Create new Blend Tree in State” con lo que la variable “Motion” del panel de Inspector cambia de “walkm” a Blend Tree.



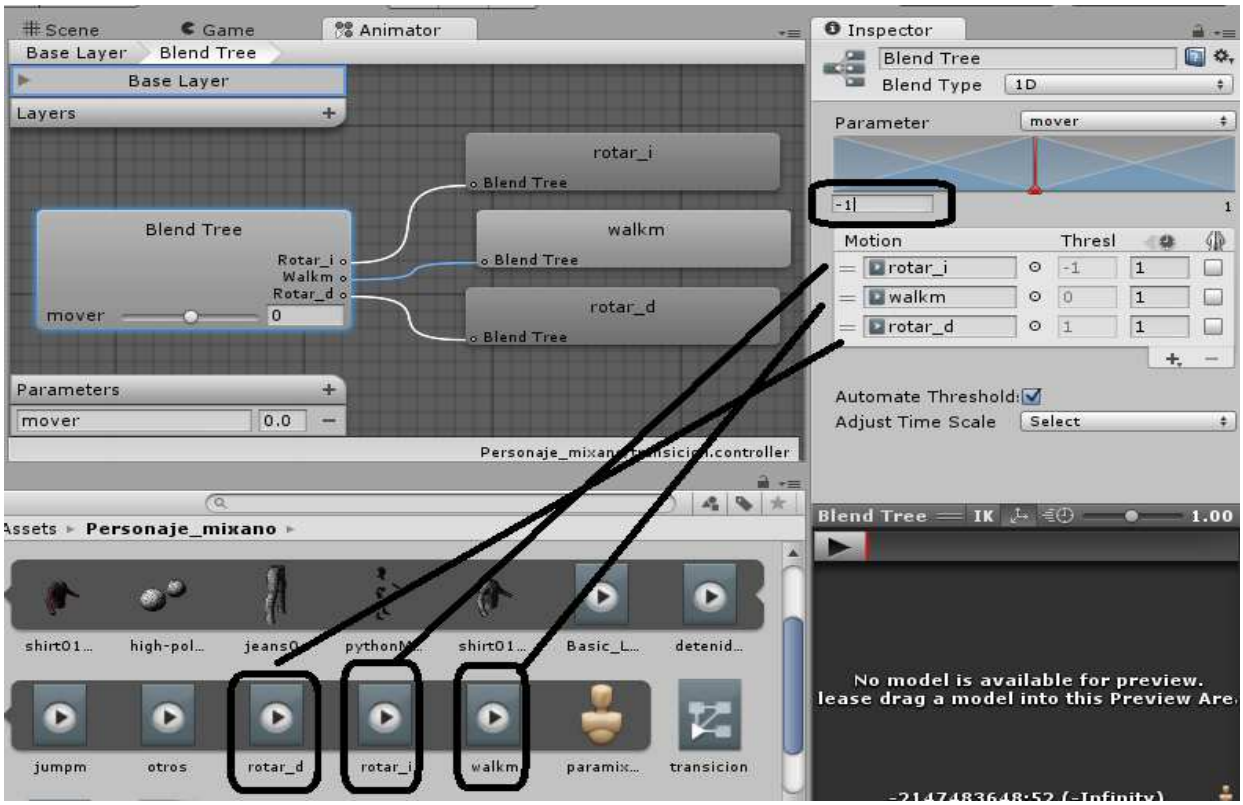
Paso 2: Damos doble click sobre la animación “walkm” y entramos a “Blend Tree” en donde presionamos tres veces en el signo “+” que se encuentra en la lista vacía de Movimientos en el panel de control.



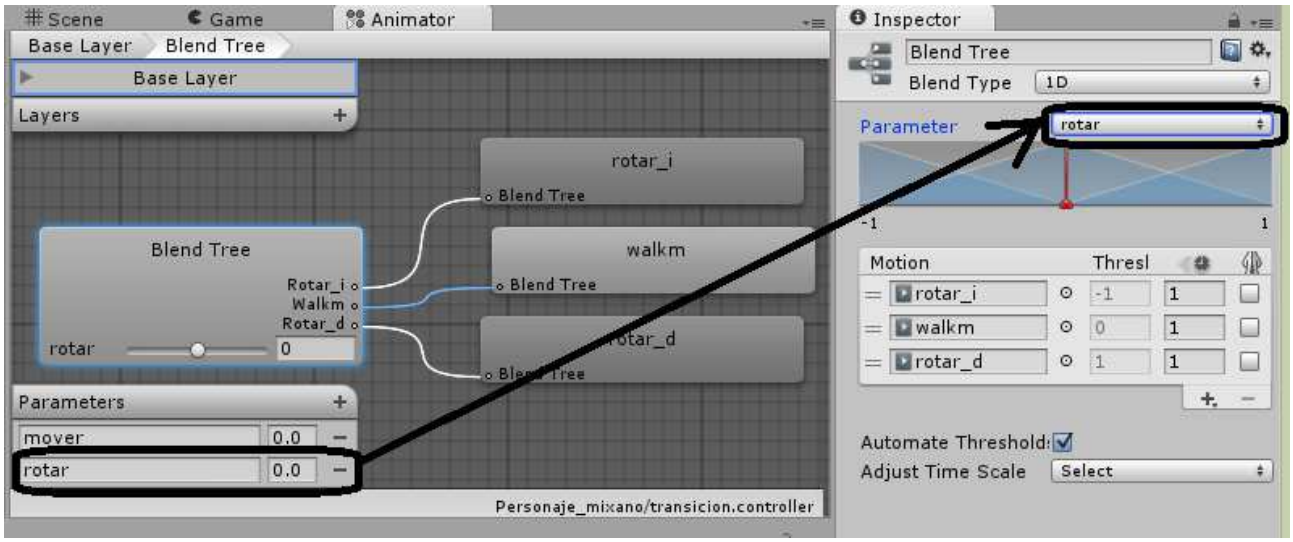
La pantalla nos tendría que quedar de la siguiente formación



Paso 3: Arrastro ahora las animaciones según se indica a continuación y pongo el valor del parámetro izquierdo a -1



Paso 4: Creamos una nueva variable “rotar” y en el panel de Inspector la asociamos a la transición.



Paso 5: Modificamos nuestro script agregando la siguiente línea de código

```
#pragma strict

public var desplazamientoX:float = 0;
public var desplazamientoZ:float = 0;
private var animator : Animator;
function OnAnimatorMove() {
     var animator : Animator = GetComponent(Animator);
    if (animator){
     var posicion:Vector3 = transform.position;
     posicion.x += desplazamientoX * Time.deltaTime;
     posicion.z += desplazamientoZ * Time.deltaTime;
     transform.position = posicion;
    }
}

function Start () {
animator = GetComponent(Animator);
}
function Update () {
animator.SetFloat("mover", Input.GetAxis("Vertical"));
animator.SetFloat("rotar", Input.GetAxis("Horizontal"));
}
```

Con esto hemos logrado que al presionar las flechas hacia los costados o las teclas “a” y “d” (si tenemos presionada la flecha hacia arriba), nuestro personaje rote hacia la derecha o hacia la izquierda.

Bloque temático 4: Inspeccionamos scripts asociados a FPC

El script anterior nos ha permitido comprender que podemos llegar a controlar la animación de nuestro personaje si le agregamos un poco de lógica. Veamos ahora un ejemplo mucho más completo de un script, en realidad vamos a ver tres scripts que se encuentran dentro de:

Assets > Standard Assets > Character Controllers > Sources Scripts

Estos son los scripts que controlan el movimiento del FPC. De todos los scripts, vamos a copiar:

1.- CharacterMotor.js

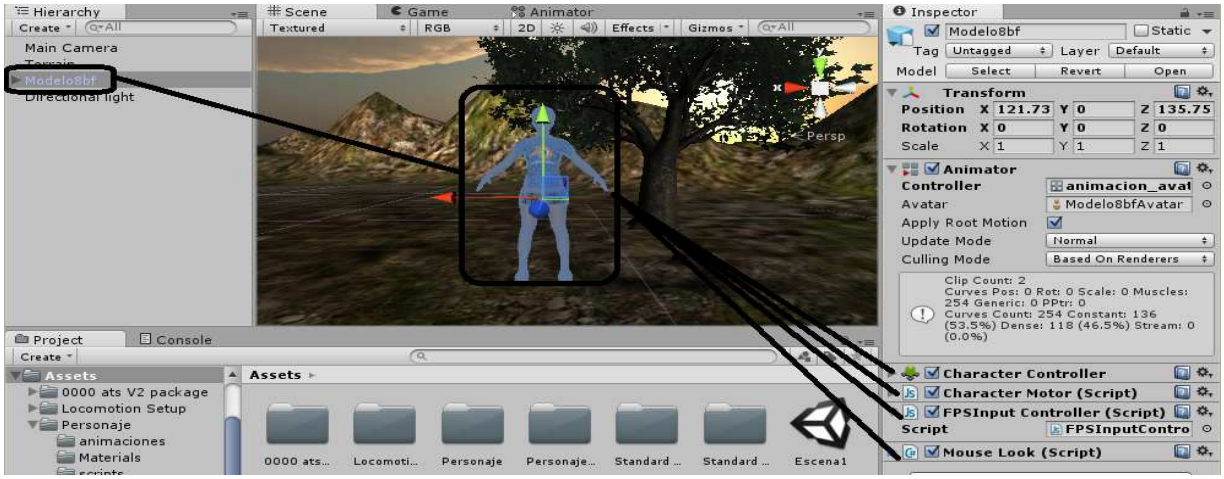
2.- FPSInputController.js

3.- MouseLook.cs

Luego creamos un directorio “Scripts” donde tenemos nuestro personaje traído de Blender y le modificamos los nombres a los scripts para no hacernos lío con los anteriores (Podrían simplemente agregarle una letra al inicio que haga referencia al personaje)

Nota: Para no eliminar el personaje que modificamos en mixamo de la escena que estábamos trabajando, simplemente crear otra escena que contenga solo este personaje

Una vez copiado los scripts, arrastramos nuestro personaje al panel de jerarquía y se los asociamos. Notar que al asociarle el **CharacterMotor.js** en el panel Inspector se crea también un Character Controller



El script CharacterMotor nos permite configurar las velocidades de desplazamiento, el valor de la aceleración de la gravedad, la altura de salto y el movimiento en la plataforma. Si damos play y jugamos con estos valores vamos a ver que nuestro personaje responde al seteo de datos. El análisis de este script requiere de un conocimiento de Programación Orientada a Objetos que excede en mucho el nivel de este curso, por lo que no vamos a entrar en detalle en el código de las clases que se encuentran ahí definidas.

Seguramente al intentar hacer caminar nuestro personaje, nos hemos encontrado que al modificar la dirección de la caminata con el mouse, el movimiento se torna extraño ya que el personaje abandona la posición vertical, y pierde contacto con el suelo. Esto se debe a que el script **MouseLook.cs** que es el que esta controlando la rotación permite la rotación según el eje y

Podemos solucionar esto si alteramos el código un poquito, modificando el valor de la rotación en y a cero, dejándolo como se muestra a continuación (el código en verde son los comentarios y en código en rosa los valores añadidos):

```
using UnityEngine;
using System.Collections;

/// MouseLook rotates the transform based on the mouse delta.
/// Minimum and Maximum values can be used to constrain the possible
rotation

/// To make an FPS style character:
/// - Create a capsule.
/// - Add the MouseLook script to the capsule.
///   -> Set the mouse look to use LookX. (You want to only turn
character but not tilt it)
/// - Add FPSInputController script to the capsule
///   -> A CharacterMotor and a CharacterController component will be
automatically added.

/// - Create a camera. Make the camera a child of the capsule. Reset it's
```

```
transform.
/// - Add a MouseLook script to the camera.
///   -> Set the mouse look to use LookY. (You want the camera to tilt up
and down like a head. The character already turns.)
[AddComponentMenu("Camera-Control/Mouse Look")]
public class MouseLook : MonoBehaviour {

     public enum RotationAxes { MouseXAndY = 0, MouseX = 1, MouseY = 2 }
     public RotationAxes axes = RotationAxes.MouseXAndY;
     public float sensitivityX = 15F;
     public float sensitivityY = 15F;

     public float minimumX = -360F;
     public float maximumX = 360F;

     public float minimumY = -60F;
     public float maximumY = 60F;

     float rotationY = 0F;

     void Update ()
     {
          if (axes == RotationAxes.MouseXAndY)
          {
               float rotationX = transform.localEulerAngles.y +
Input.GetAxis("Mouse X") * sensitivityX;

               rotationY += 0;//Input.GetAxis("Mouse Y") *
sensitivityY;
               rotationY = 0;//Mathf.Clamp (rotationY, minimumY,
maximumY);

               transform.localEulerAngles = new Vector3(-rotationY,
rotationX, 0);
          }
          else if (axes == RotationAxes.MouseX)
          {
               transform.Rotate(0, Input.GetAxis("Mouse X") *
sensitivityX, 0);
          }
          else
          {
```

```
                rotationY += 0;//Input.GetAxis("Mouse Y") *
sensitivityY;
                rotationY = 0;//Mathf.Clamp (rotationY, minimumY,
maximumY);

                transform.localEulerAngles = new Vector3(-rotationY,
transform.localEulerAngles.y, 0);
           }
     }

     void Start ()
     {
          // Make the rigid body not change rotation
          if (GetComponent<Rigidbody>())
               GetComponent<Rigidbody>().freezeRotation = true;
     }
}
```

El script **FPSInputController.js** utiliza al script **CharacterMotor.js** para poder determinar los movimientos.

Con esto nuestro personaje ahora camina normalmente.

Bibliografía utilizada y sugerida

Documentación oficial online -

http://docs.unity3d.com/Manual/class-Avatar.html

https://unity3d.com/es/learn/tutorials/modules/beginner/graphics/textures

Lo que vimos

En esta unidad nos hemos introducido en la tarea de incorporarle animaciones a nuestro personaje, así como en el uso de transiciones.

Lo que viene:

En la siguiente unidad veremos como comenzar a trabajar con interacciones.