

Escuela ORT

Orientación: Informática

Materia: Taller de Videojuegos con Unity



Videojuegos con Unity 3D

<Clase 06: Mi primer personaje 3D />

Introducción

En el encuentro del día de hoy daremos inicio a un nuevo proyecto en el que aprenderemos nuevas formas de trabajar y continuaremos agregando conocimientos a nuestra mente desarrolladora.

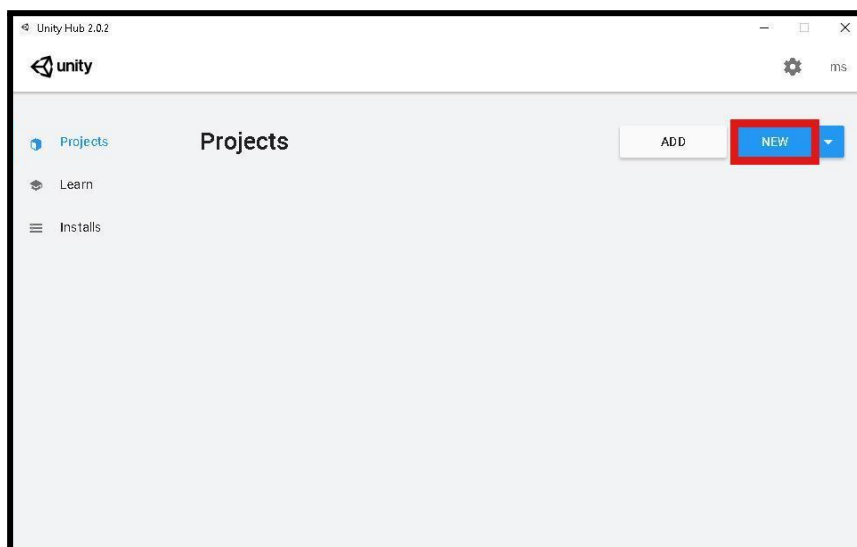
Nuevamente nos embarcamos en la creación de un personaje el cual cumplirá con cierta lógica pero en esta instancia gozará con un agraciado apartado visual.

Daremos uso a una página web que nos ayudará a complementar el aspecto visual del proyecto mediante el uso de un colmado repertorio de modelos y animaciones 3D libres de costo.

Un nuevo comienzo.

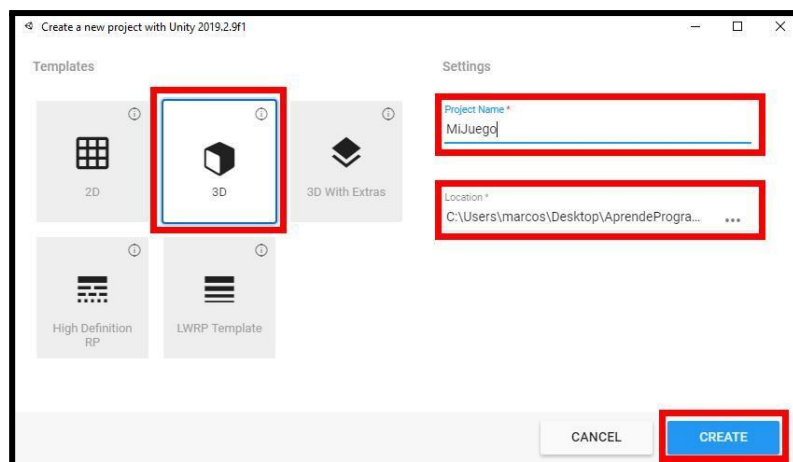
Ya pudimos experimentar con la creación de un primer videojuego el cual cumplió su ciclo. Por eso, perdamos el miedo de volver a empezar, de volver a hacer y crear. Estamos trabajando con videojuegos donde todo está en constante movimiento y un pequeño cambio puede significar lo mejor que nos haya pasado.

- 1) Acceder al programa Unity Hub y ejecutarlo.
- 2) Crearemos un nuevo proyecto haciendo click en New.



Se nos abrirá una ventana a la cual le asignaremos las características de nuestro proyecto:

- ☐ Utilizaremos la template 3D.
- ☐ Nombraremos nuestro proyecto
- ☐ Le asignaremos una dirección de guardado.
- ☐ Por último le daremos click a Create.



Luego de finalizada la carga se nos abrirá el editor, el cual ya debería parecernos muy familiar.

Mi primer personaje 3D.

Cada uno de los videojuegos que conocemos cuenta con al menos un personaje y una historia que va desde la más sencilla hasta el más complejo árbol de narraciones.

Como hablamos antes, el aspecto visual de nuestro jugador está ligado al contexto de la historia que creamos y queremos contar.

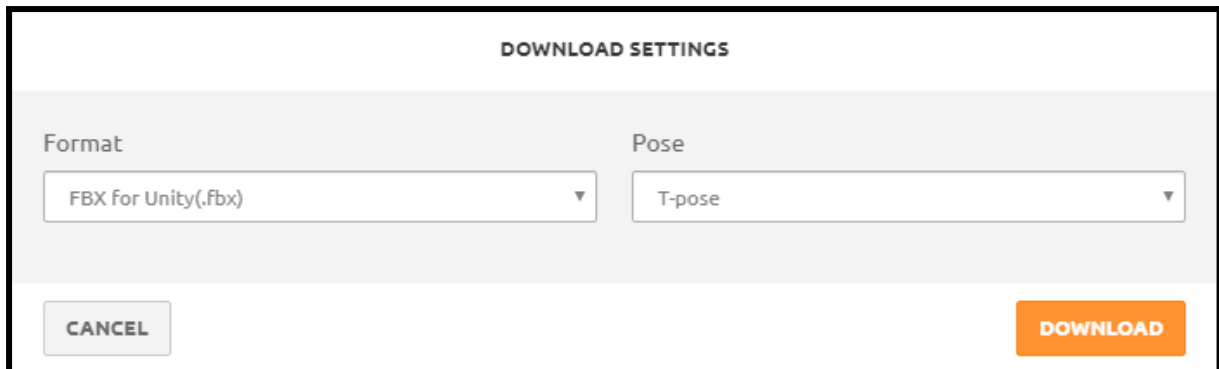
Nuestro personaje puede tener una belleza ilimitada, pero ese atractivo no siempre puede provenir de nuestras manos.

Afortunadamente, hay almas generosas que entienden nuestra situación y nos ayudarán a no trabarnos en el camino.

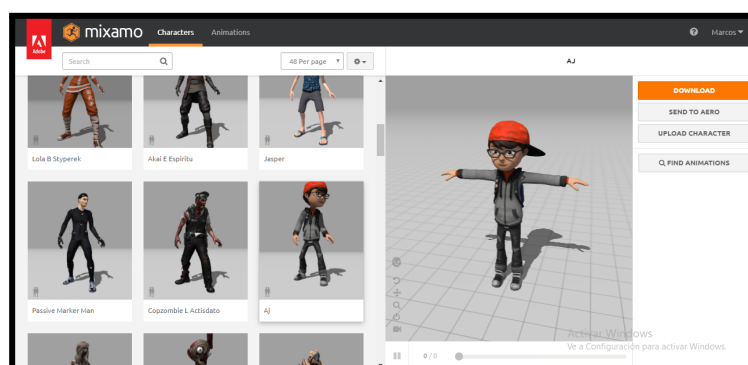
Si ! Estamos hablando de modelos 3D gratuitos que provienen de artistas profesionales que colaboran con la industria.

Estos modelos 3D se pueden encontrar en diferentes páginas con una simple búsqueda e incluso el “Asset Store” de Unity.

Para continuar con el proyecto , utilizaré un personaje que se encuentra en la página : www.mixamo.com. y posee las siguientes características :



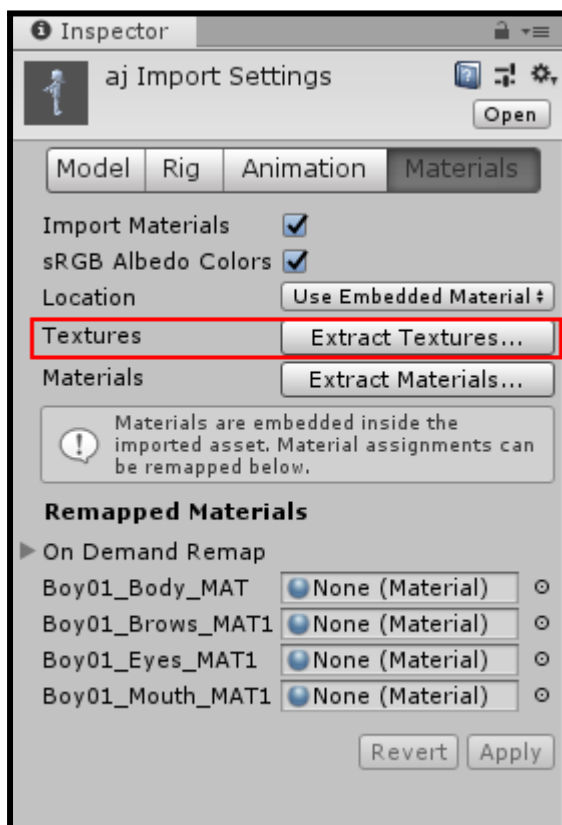
Mixamo es un sitio web, que posee una gran cantidad de modelos y animaciones que podemos utilizar en nuestro videojuego. Recuerda que para ingresar te deberás registrar con el uso de un mail o puedes utilizar [ESTE](#) modelo para continuar con el proyecto,



Mientras esperamos a que finalice la descarga de nuestro personaje procederemos a crear una carpeta en nuestro proyecto de Unity llamada “Modelos 3D”.

Seguidamente, importamos el modelo 3D con solo arrastrarlo sobre la carpeta recientemente creada.

Ahora seleccionaremos el modelo y en la ventana del inspector buscaremos la pestaña Materials y le daremos al botón “Extract Textures”.



Cuando apretemos el botón se nos abrirá una ventana la cual utilizaremos para indicar donde queremos que se guarden las texturas que vienen adjuntas al modelo.

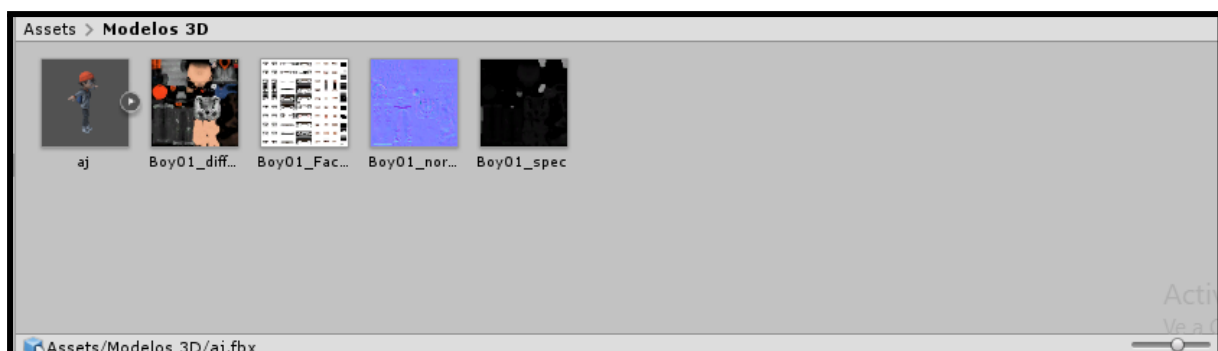
Por el momento seleccionaremos la carpeta “Modelos 3D”.

Recuerda que tenemos que esperar una mínima cantidad de tiempo para que realice la extracción de forma correcta.

Al finalizar la carga, nos aparecerá un cartel que nos preguntará si queremos arreglar los “Normal map” y aceptaremos haciendo click en el botón “Fix Now”.

Un normal map es un tipo de textura el cual le da detalles de iluminación, profundidad y relieve a un material. Aprenderemos sobre ellos más adelante.

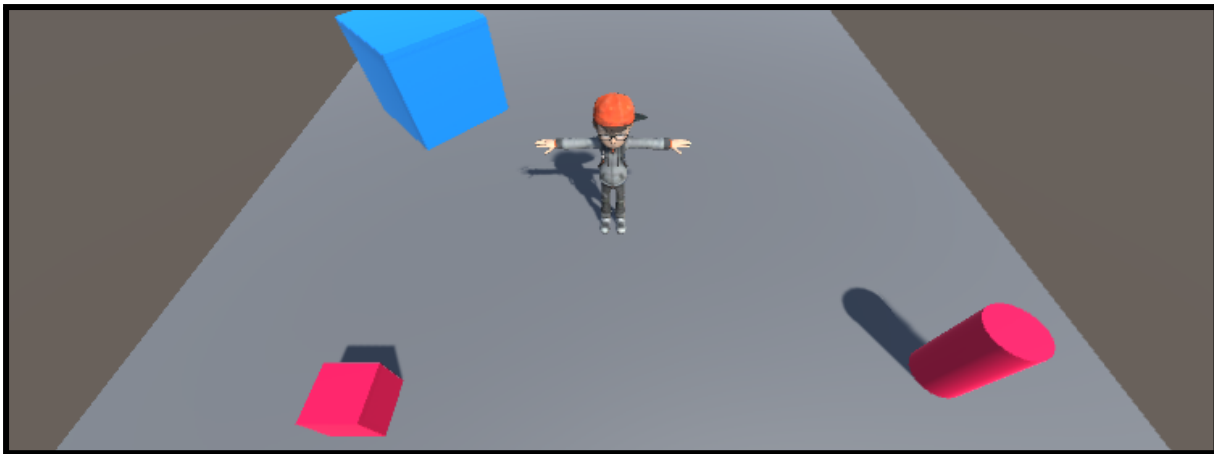
Luego de completar los pasos, nuestra carpeta quedará de la siguiente forma :



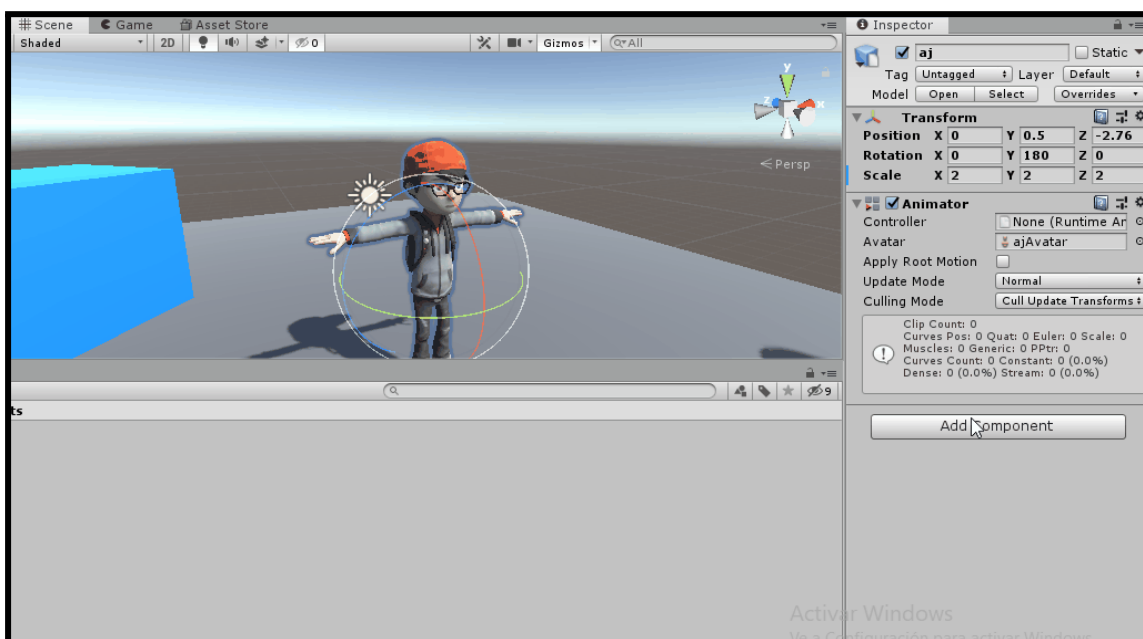
¡ Excelente!

Ahora es momento de comenzar a utilizar nuestro personaje, así que lo arrastraremos a la escena principal.

Pero como comenzamos con un proyecto nuevo el escenario se encontrará vacío. Pero tratemos de cambiarlo agregando un piso y algunos obstáculos sencillos y diferenciados con colores(materiales) para que nuestro personaje pueda moverse. También ajustemos la cámara para poder visualizar todos los elementos del mapa que estamos creando.



Antes de avanzar con la creación del código configuraremos a nuestro protagonista para que posea todos los componentes necesarios :

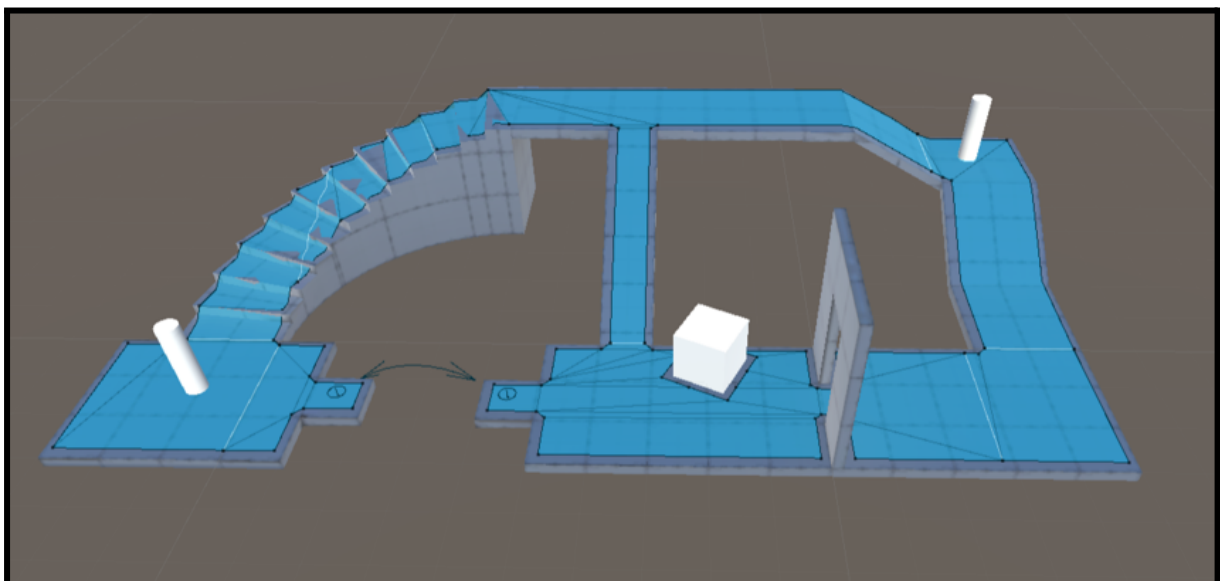


- 1) Seleccionaremos el personaje y en la sección del inspector agregaremos el componente "Capsule Collider" para detectar las colisiones.
- 2) También configuraremos su Radius(radio) y su Height(Altura) y Center para que se ajuste a nuestro jugador.

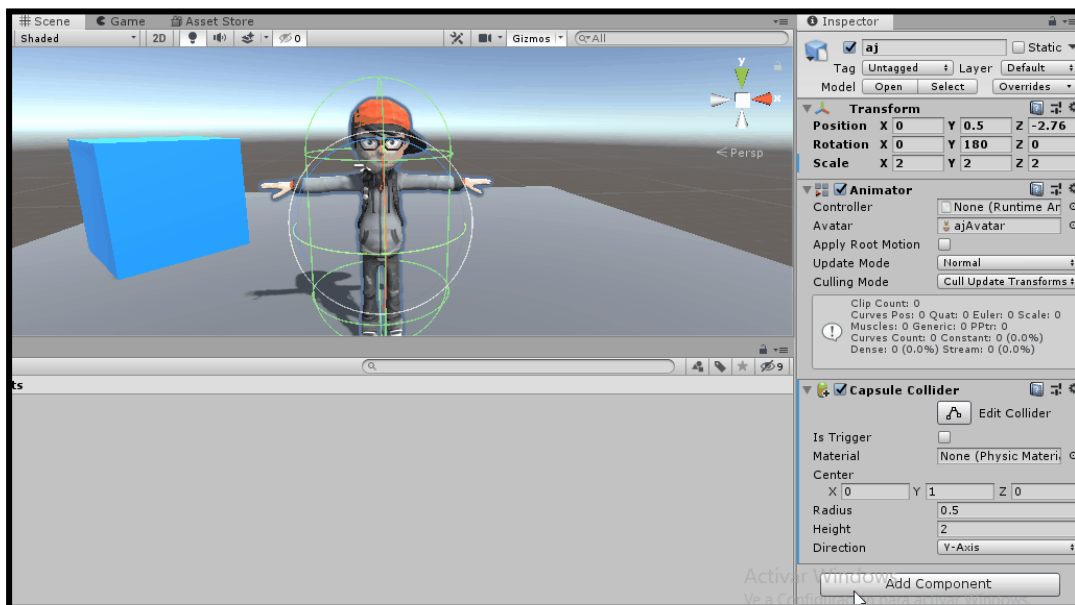
NOTA: El collider es representado por las líneas verdes alrededor del personaje y ellas deben ajustarse lo más cercano posible al jugador de lo contrario tendremos problemas a la hora de detectar colisión.

Sistema de navegación.

Este es un sistema que nos permite crear personajes que se puedan mover de forma inteligente alrededor del mundo utilizando figuras de navegación generadas automáticamente para la geometría de la escena



- 1) Agregaremos un componente nuevo llamado **NavMeshAgent** el cual nos servirá para hacer que nuestro jugador “navegue” a través del escenario.
- 2) De la misma forma que el capsule collider configuraremos el Radius(Radio) y el Height(Altura) para que se ajuste al modelo 3D.



- 3) Seguidamente, buscaremos la ventana de “Navigation” dando click en Window > AI > Navigation.
- 4) Observaremos que en el inspector tenemos varias características que podemos retocar para el uso del NavMesh :

Agents: podemos configurar las propiedades de los agentes que serán utilizados para moverse.

Áreas : Esta sección es utilizada para delimitar los diferentes tipos de áreas y cuáles de ellas podrán ser recorridas por los agentes.

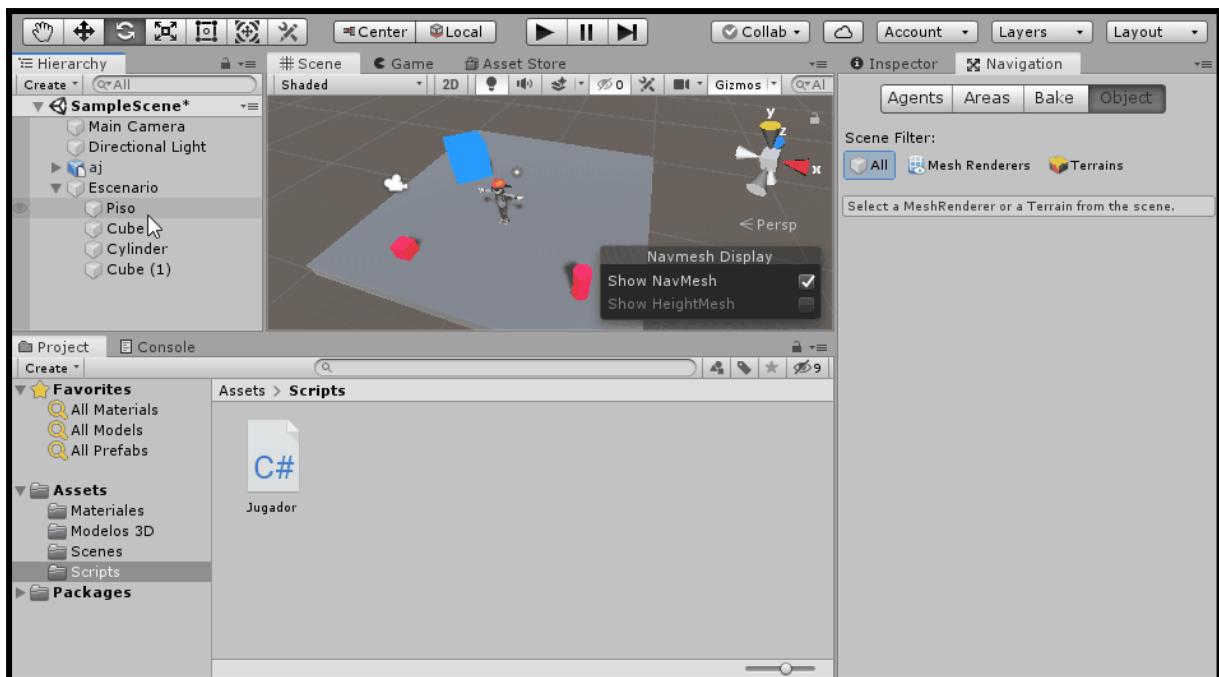
Bake: Sección que generalmente se usa para “bakear”, “cocinar” o delimitar las áreas disponibles de navegación mediante el uso de los objetos.

Object: Aquí se seleccionan los objetos y se los setea como alguno de los tres tipos: “Caminables” “No Caminables” “Saltable”.

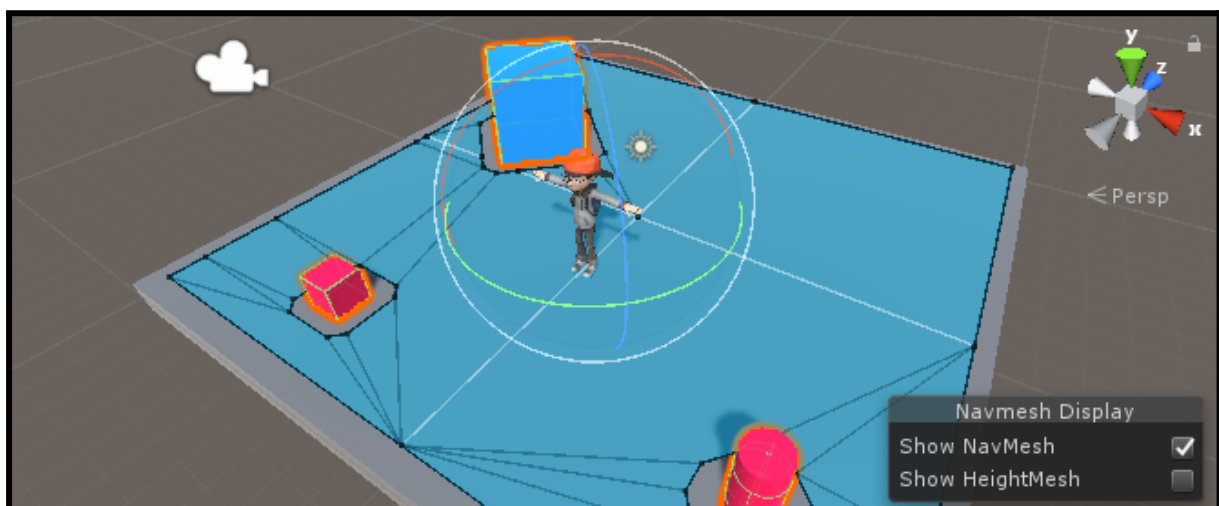
Mantendremos la pestaña **Object** abierta y seleccionaremos el piso (cubo o plano) el cual configuraremos como “Navigation Static” y “Walkable” para que el personaje pueda caminar sobre él.

Posteriormente, clickearemos en los obstáculos y haremos el mismo paso, con la diferencia de que ahora serán “Navigation Static” y “Not Walkable”.

Cuando hayamos terminado de setearlos nos dirigiremos a la pestaña **Bake** y haremos click en el botón “Bake”.



Notarán que Unity comenzará a realizar los cálculos para delimitar las zonas de navegación. Una vez terminada la carga tendremos nuestra escena encuadrada en color celeste los posibles lugares a donde el jugador puede acceder.



Al igual que el proyecto anterior procederemos a crear la carpeta “Scripts” y dentro de ella un nuevo c# script llamado Jugador.

Abriremos el script y luego de esperada la carga comenzaremos a escribir el código :

Lo primero que tenemos que escribir es el llamado a la clase AI (Artificial Intelligence) la cual es la clase “Padre” o contenedora del Nav Mesh.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
```

Luego, codearemos dos variables : una de tipo **NavMeshAgent** llamada “agente” la cual será obtenida en el método Start mediante las líneas GetComponent<>() y la otra será de tipo **LayerMask** pública llamada “sobreQuePuedoCaminar”. Un LayerMask es un tipo de variable que se utiliza de forma similar a un tag. En nuestro caso, su uso será para delimitar sobre qué puntos de una capa podemos hacer click para que nuestro personaje pueda dirigirse hacia ellos.

```
public class Jugador : MonoBehaviour
{
    NavMeshAgent agente;
    public LayerMask sobreQuePuedoCaminar;

    void Start()
    {
        agente = GetComponent<NavMeshAgent>();
    }

    void Update()
    {
    }
}
```

Raycast.

Ahora que tenemos nuestras variables listas, podemos comenzar a escribir la lógica de “Click para movernos”. Emplearemos los RayCast los cuales son líneas invisibles que serán creadas desde la posición de la cámara hasta el punto donde hagamos click con nuestro mouse. Estos Raycast son ampliamente usados en gran variedad de juegos y aplicaciones; Un claro ejemplo es el de los shooters donde se crea un rayo invisible desde el punto A (La cámara) hasta un punto B (Adelante) y si ese rayo encuentra algo de un tag o layer específico reacciona ante tal (Personajes, paredes, bombas, etc).

Comencemos escribiendo un if dentro del método Update() que pregunte por el input del click izquierdo del mouse(0).

```
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        // ...
    }
}
```

Si la sentencia if devuelva true, procederemos a crear dos variables nuevas Ray y RayCastHit.

La primera es la que guarda desde que punto se generará el rayo, en nuestro caso la posición de la cámara principal(A) hasta el lugar donde hacemos click con el mouse (B) y la segunda es la que guarda toda información de los objetos a los cuales el rayo ha dado.

```
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        Ray rayo = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit info;
    }
}
```

Ahora Procederemos a realizar otra sentencia if la cual mediante el uso de una método **generará** el rayo utilizando ciertos parámetros :

```
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        Ray rayo = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit info;
        if (Physics.Raycast(rayo, out info, 100f, sobreQuePuedoCaminar))
        {
            // 
        }
    }
}
```

- 1) Utilizamos el método Raycast() de la clase Physics para crear el rayo.
- 2) Recibe como primer parámetro una variable de tipo Ray que indique desde donde comienza la creación del mismo..
- 3) Parámetro que guarda la información del punto donde ha dado la línea invisible.
- 4) Distancia máxima que posee el rayo(float).
- 5) LayerMask o capa en la que tendrá efecto si se cumple la sentencia if.

Ahora tenemos que agregar la porción de código que hará que nuestro personaje se mueva al destino correcto en el caso de que la condición se cumpla.

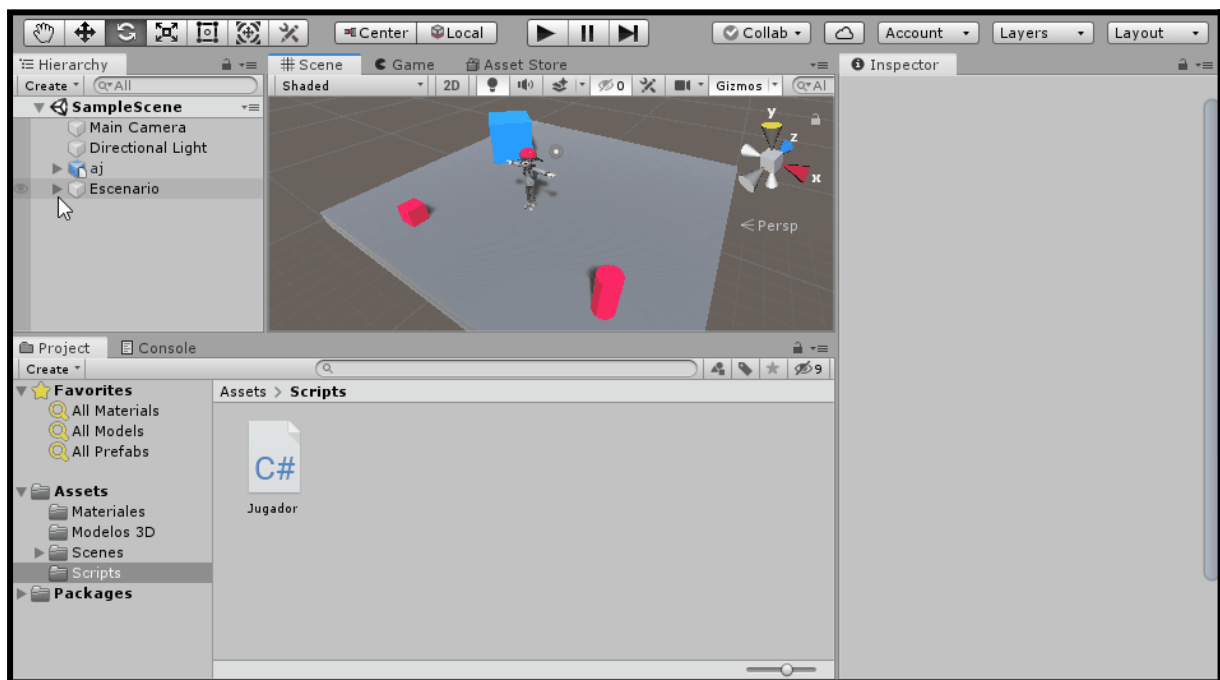
Aquí es donde daremos uso al agente creado :

```
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        Ray rayo = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit info;

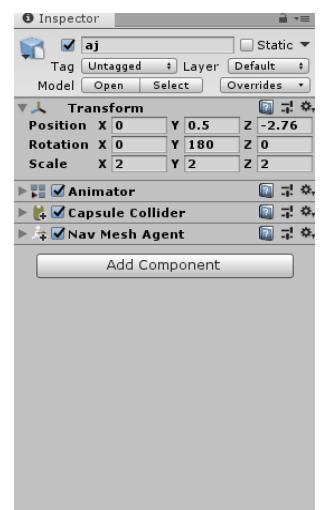
        if (Physics.Raycast(rayo, out info, 100f, sobreQuePuedoCaminar))
        {
            agente.SetDestination(info.point);
        }
    }
}
```

Ahora solo nos faltará configurar la o las capas que nuestro player puede utilizar para moverse :

- 1) Seleccionaremos el GameObject piso y en la pestaña del inspector buscaremos la sección Layers (Arriba a la derecha).
- 2) Notarán que estamos utilizando la capa "Default" pero cambiemosla apretando el botón y luego en "Add Layer".
- 3) Se nos abrirá una ventana con todas las layers de nuestro juego. Luego buscaremos un espacio vacío y escribiremos el nombre de capa que queramos utilizar. En nuestro caso utilizaremos "Piso".
- 4) Volveremos a seleccionar el GameObject y pondremos la capa "Piso".

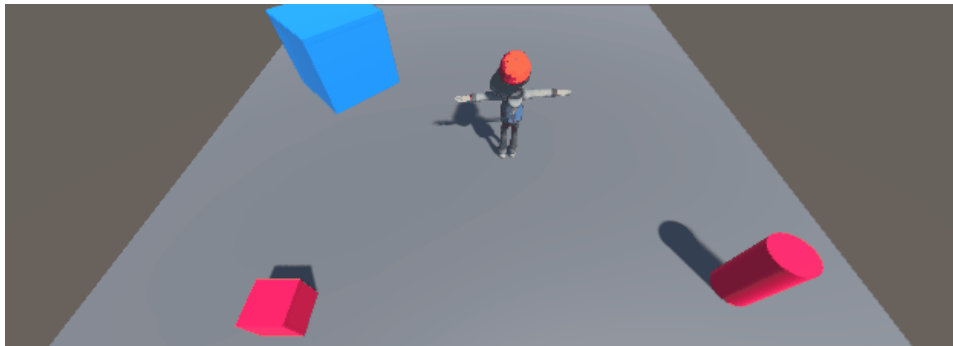


Por último agreguemos como componente el script "Jugador" al protagonista o modelo 3D que importamos y en la sección de LayerMask seleccionaremos "Piso" de lo contrario no funcionará el código que hicimos.



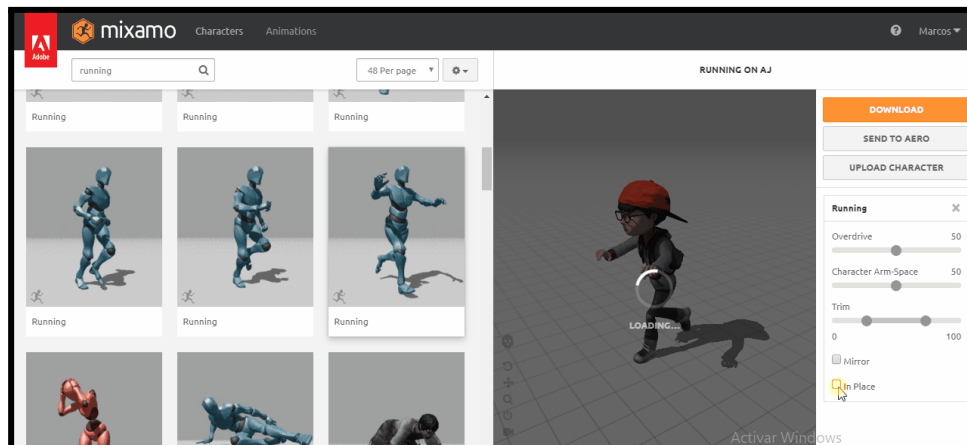
¡Genial! Probemos nuestro script entrando en modo “Play”.

Animaciones.

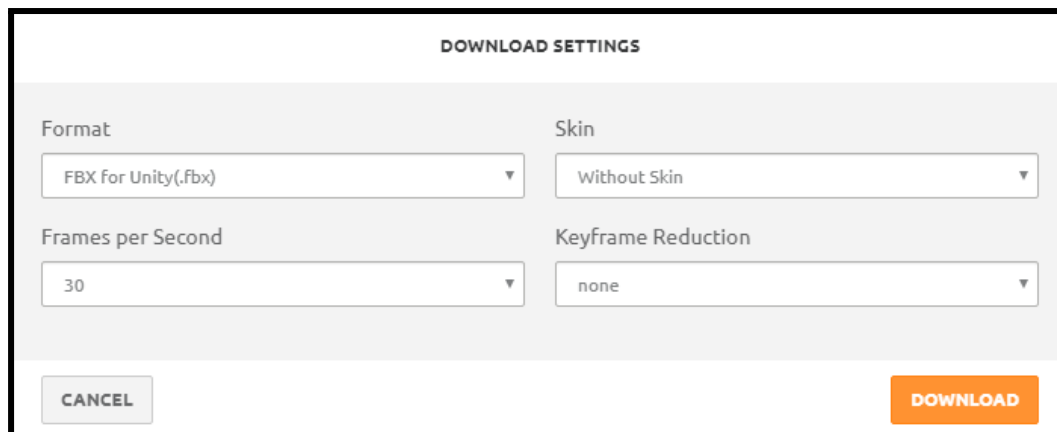


Ya tenemos nuestro personaje respetando la lógica de click que codeamos. Pero como podrán ver el modelo sigue en esa fea pose T. La cual es utilizada por modeladores y artistas para diseñar de forma correcta las articulaciones y huesos de los personajes. Cambiemoslo agregando dos animaciones que serán equivalentes a dos estados : IDLE será utilizada cuando el personaje esté quieto y RUNNING para cuando esté en movimiento.

Pero para llegar a este punto primero debemos descargar las animaciones y lo haremos desde nuestra estimada página www.mixamo.com o podemos descargar las



utilizadas en esta clase ([idle](#) - [running](#)).

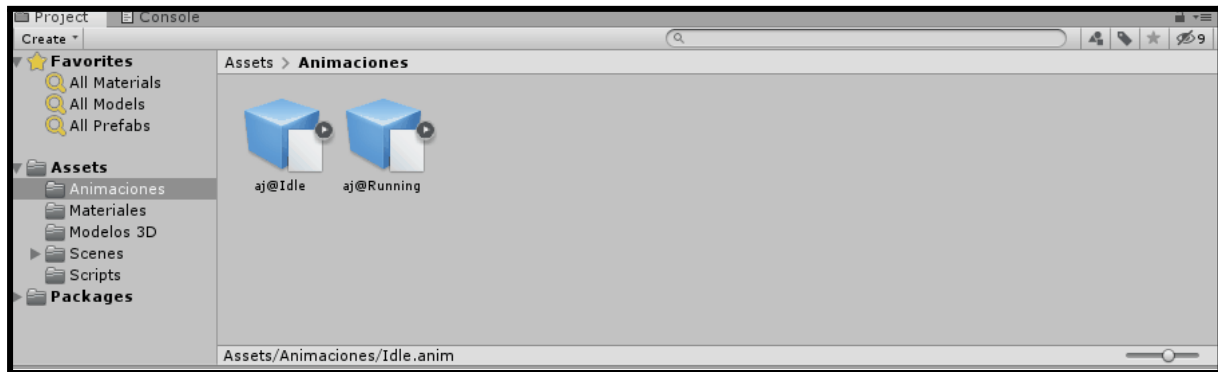


Luego de finalizada la descarga, volveremos al editor y crearemos una carpeta llamada “Animaciones” y arrastraremos nuestras nuevas animaciones.

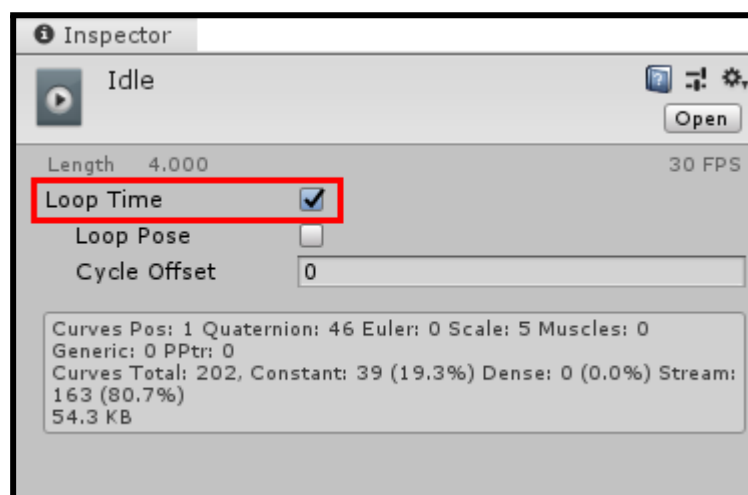
Seleccionaremos el “prefab” que contiene todos los objetos que utiliza la animación y lo expandimos haciendo click en su flecha.

Clickearemos en el “Animation Clip” y lo duplicaremos utilizando el comando CTRL + D. Realizaremos el mismo procedimiento con la segunda animación.

Este paso lo utilizamos para evitar realizar demasiadas configuraciones.



Ahora seleccionaremos los duplicados y en la sección del inspector tildamos la opción “Loop” para que las animaciones se reinicien una vez terminadas.



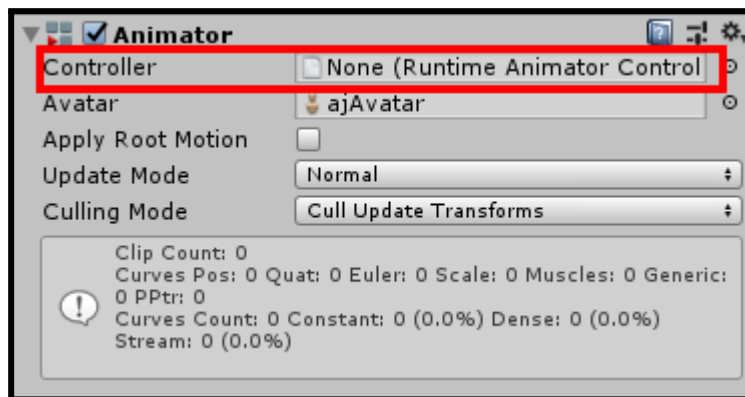
¡Excelente!

Cuando realizamos la animación de los obstáculos en el primer juego habíamos hablado de dos formas de “manejarlas” : la primera era mediante el componente Animation y la segunda utilizando el Animator. En el caso actual, daremos uso a la última ya que estamos usando **varias** animaciones que deberán seguir cierta **lógica** para poder funcionar correctamente.

Seleccionemos el personaje de nuestro juego y observemos si cuenta con el componente Animator (Generalmente se coloca automáticamente cuando lo

importamos a la escena) De lo contrario agreguemoslo mediante el botón “Add Component”.

Como notarán este componente requiere del uso de un **Animator Controller**



Animator Controller.

A menudo los personajes u objetos tienen múltiples animaciones que poseen ciertas condiciones para cambiar. Por ejemplo, la animación de correr puede convertirse en una de saltar. Esto se logra mediante un controlador de animaciones o Animator Controller el cual nos permite arreglar y mantener un conjunto de clips de animaciones así como también las transiciones que ocurren entre ellas.



Comencemos a configurar el controlador :

- 1) En la carpeta animaciones daremos Click Derecho > Create > Animator Controller.

- 2) Lo nombraremos “Jugador”.

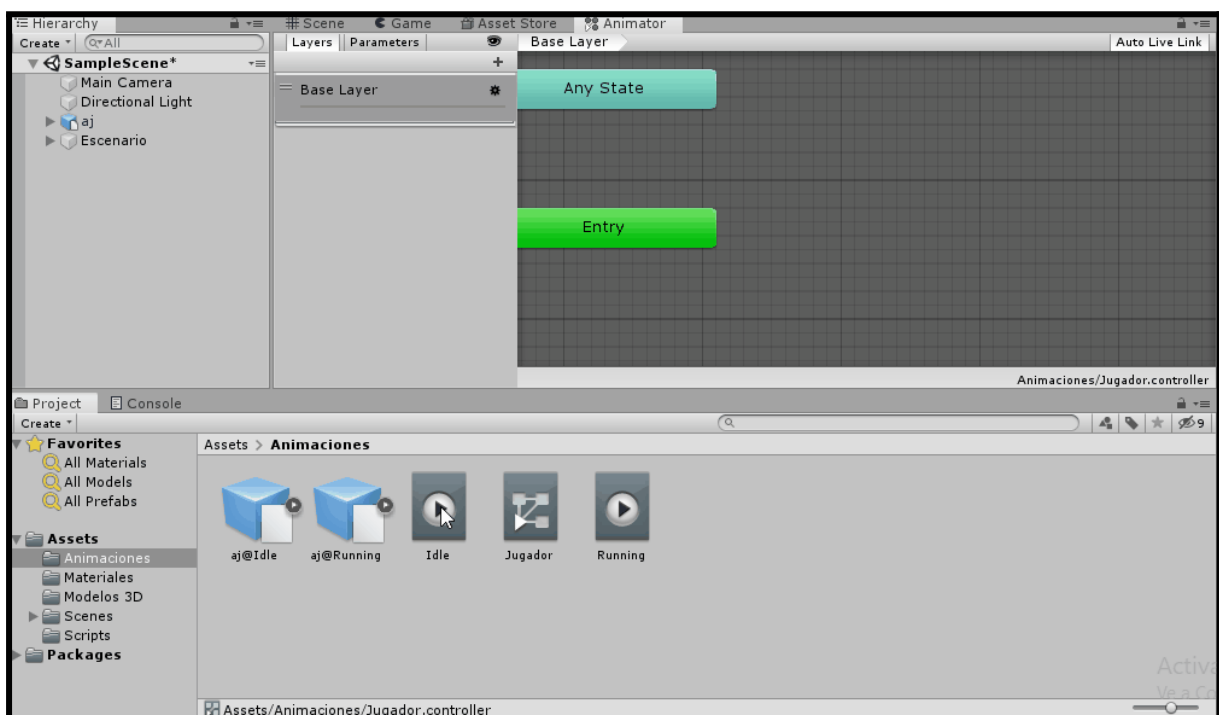
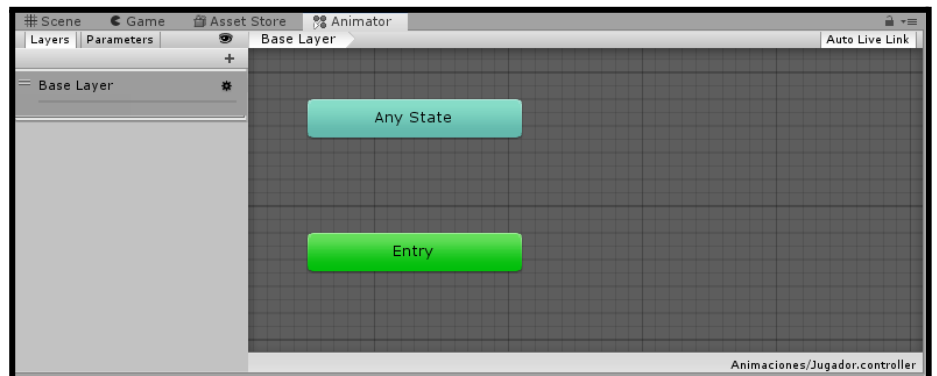
- 3) Lo abriremos haciendo doble click sobre el controlador.

- 4) Notarán que tenemos dos estados de inicio, los cuales nos servirán para realizar las transiciones y acomodar nuestras animaciones.

Entry es el estado inicial por el cual tendrá efecto la primera animación y Any State

funciona para que las animaciones puedan realizar su transición en cualquier momento indicado según nuestros criterios.

Comenzaremos arrastrando las dos animaciones (Idle y Walking) a la grilla del controlador.

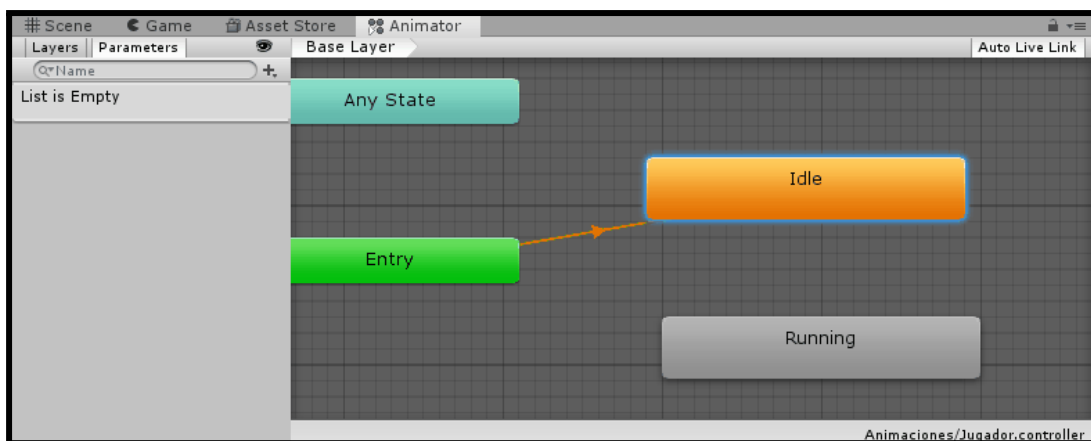


La

flecha de Entry se posicionará en el primer clip que soltemos sobre la grilla

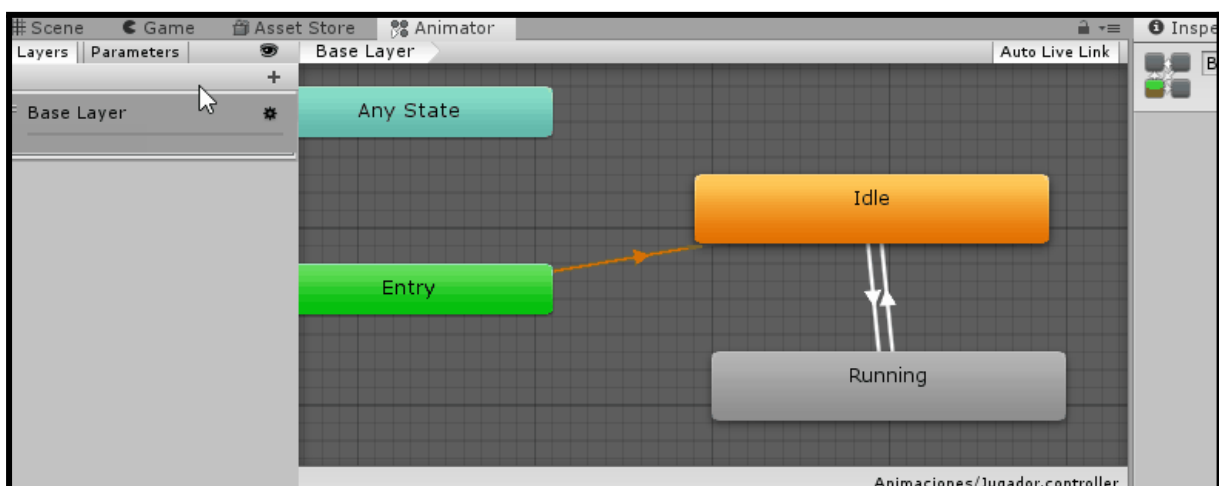
seteandolo como la “Animación” de entrada. También, notarán que Running no se encuentra conectada a nada por más que esté presente en el controlador. Cambiemos esto:

- 1) Daremos click derecho sobre el “estado” Idle y luego en Make Transition.
- 2) Arrastraremos la flecha sobre el estado Running.
- 3) Repetiremos los pasos anteriores pero esta vez empezando desde running hacia idle.



Las flechas que acabamos de crear representan una transición entre las dos animaciones. Pero para que esa transición haga efecto se debe cumplir una condición: Un parámetro determinará si se debe cambiar de estado o no.

Daremos click en botón “Parameters” en la parte superior de la ventana y nos crearemos uno nuevo de tipo **Bool** llamado “EstaCorriendo”.



Ahora configuraremos las “flechas” o transiciones entre los dos estados de la animación en la que preguntaremos por el parámetro “EstaCorriendo” :

En el caso de que el mismo sea **True** significa que nuestro personaje se encuentra en movimiento y esa es nuestra señal para activar el estado de la animación “Running” y si es **False** se mantendrá en la posición “Idle”.

1) Seleccionaremos la flecha que se dirige hacia “Running”.

2) Desactivaremos la opción “Has Exit Time” para que la transición se realice de forma “rápida”.

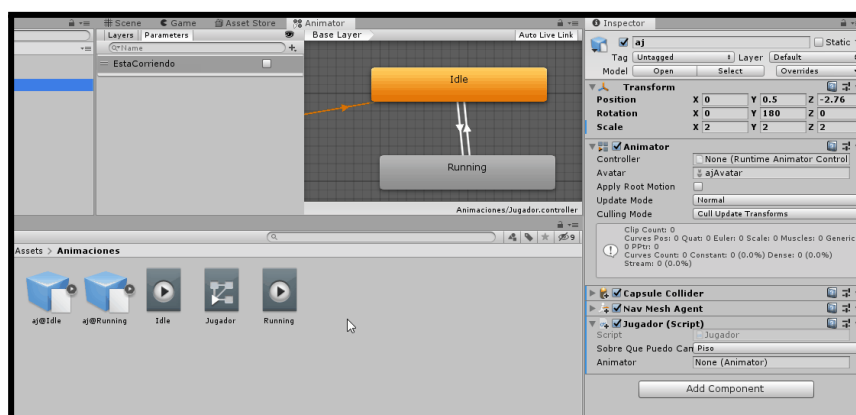
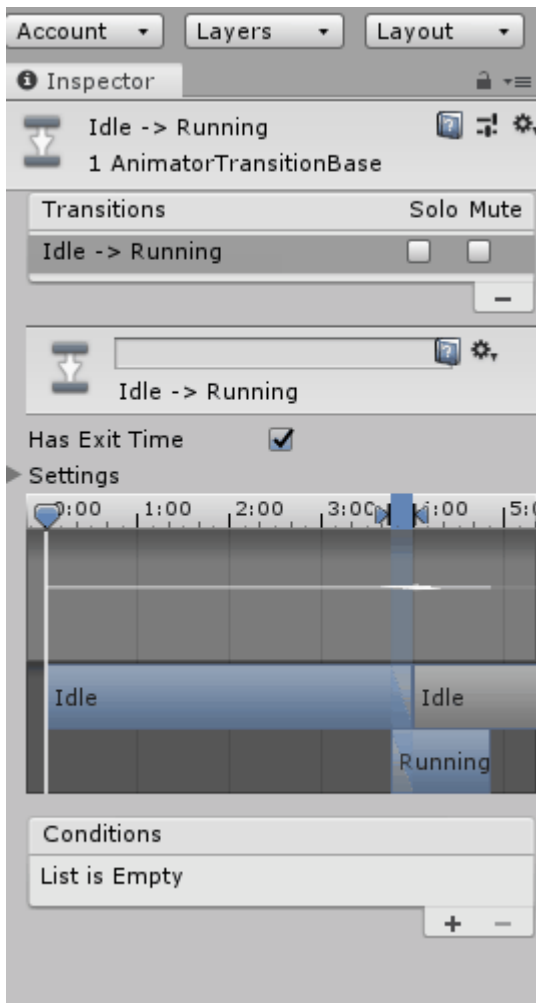
3) La sección de “Conditions” será utilizada para agregar la condición de transición : En nuestro caso será si el parámetro “EstaCorriendo” es igual a True.

4) Ahora seleccionaremos la flecha que apunta hacia “Idle”.

5) Nuevamente desactivaremos la opción “Has Exit Time”.

6) La condición “EstaCorriendo” será igualado a “False”.

7) Luego de terminar de configurar el Controlador lo arrastraremos a la ranura “Animator Controller” del componente agregado en el jugador.



A codear un poco más.

Ya tenemos el Animator Controller seteado para que cambie sus animaciones dependiendo de un parámetro.

Pero el mismo cumple con lógicas que no están adaptadas al movimiento de nuestro jugador : Alteremos el script “Jugador” para que ambas estén conectadas entre sí.

- 1) Comencemos creando una variable de pública de tipo Animator llamada (valga la redundancia) “animator”.

```
NavMeshAgent agente;  
public LayerMask sobreQuePuedoCaminar;  
  
public Animator animator;
```

- 2) Nuevamente, en el método Update() daremos uso del agente creado al inicio de la clase para saber si nuestro personaje se encuentra en movimiento hacia el punto que hicimos click o si ya llegó a su destino. Y lo haremos mediante un if que compara dos variables provenientes de la clase NavMeshAgent:

```
void Update()  
{  
    if (Input.GetMouseButtonDown(0))  
    {  
        Ray rayo = Camera.main.ScreenPointToRay(Input.mousePosition);  
        RaycastHit info;  
  
        if (Physics.Raycast(rayo, out info, 100f, sobreQuePuedoCaminar))  
        {  
            agente.SetDestination(info.point);  
        }  
    }  
  
    if (agente.remainingDistance <= agente.stoppingDistance)  
    {  
    }  
}
```

En resumen, estamos preguntando si la distancia que le queda recorrer al agente es menor a la distancia de seteada como “detenido” (entre 0m y 1m).

En el caso de que la sentencia sea **True** significa que el personaje o agente ha llegado a su destino y deberíamos poner en modo “Play” la animación “idle”.

Y si fuese **False** quiere decir que el protagonista continúa en movimiento dirigiéndose hacia el punto clickeado informándonos que la animación de este caso sería “Running”.

Para completar con lo mencionado previamente utilizaremos el método “SetBool” de la clase animator el cual utiliza dos argumentos: El primero es el nombre del “parámetro” que creamos para realizar las transiciones: En nuestro caso será el “EstaCaminando” y la segunda es el valor que queremos cambiar de ese parámetro (True o False).

NOTA: Recuerda que estamos utilizando este método porque el parámetro que elegimos es de tipo Bool pero el mismo puede variar y ser Trigger, Int y float.

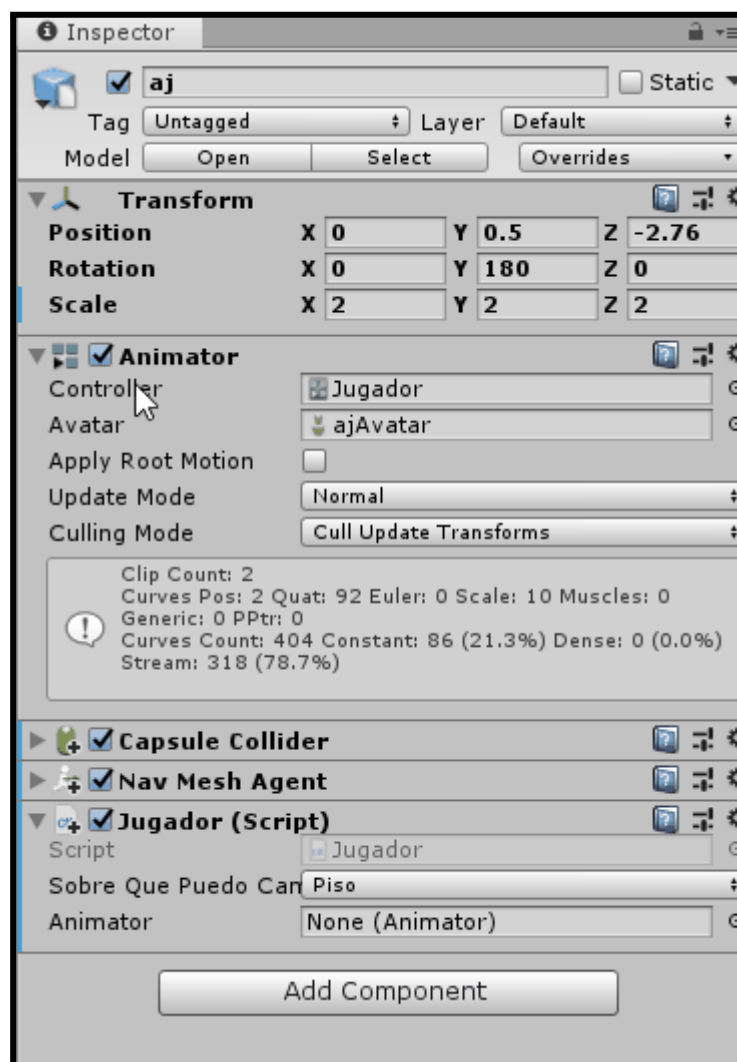
```
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        Ray rayo = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit info;

        if (Physics.Raycast(rayo, out info, 100f, sobreQuePuedoCaminar))
        {
            agente.SetDestination(info.point);
        }
    }

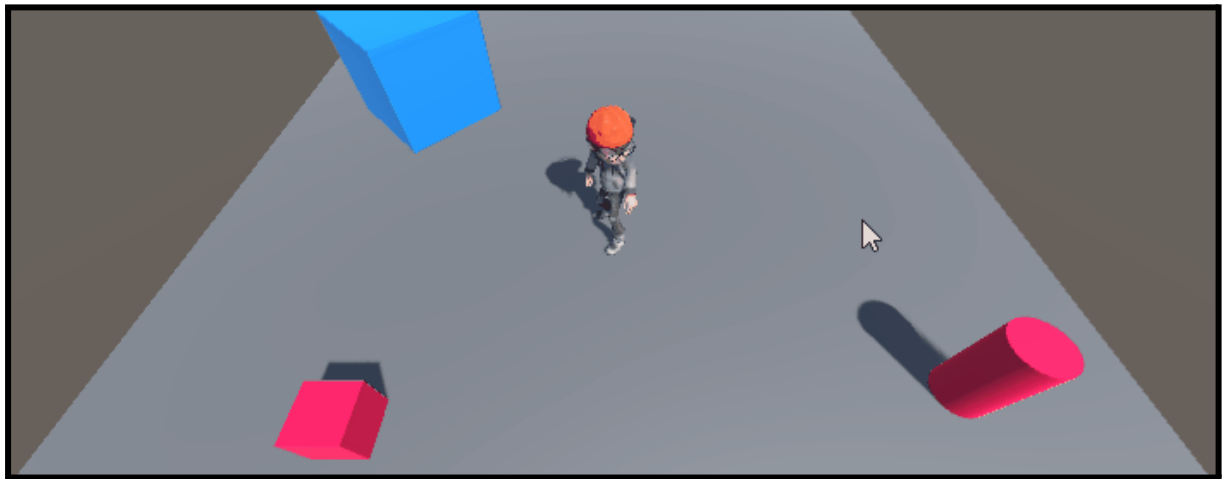
    if (agente.remainingDistance <= agente.stoppingDistance)
    {
        animator.SetBool("EstaCorriendo", false);
    }
    else
    {
        animator.SetBool("EstaCorriendo", true);
    }
}
```

NOTA: El primer argumento que pasamos a la función SetBool deberá estar escrito exactamente igual al que creamos en el Animator Controller de lo contrario no hará efecto el cambio de estado ya que no podrá reconocerlo con el “nuevo nombre”.

Para concretar el uso de animaciones arrastraremos el componente Animator de jugador dentro de la ranura que se acaba de crear con el uso de la variable pública del mismo nombre.



Si todo funciona correctamente deberíamos tener a nuestro jugador animado en la escena. Probemoslo dando click al botón "Play" :



Bibliografía:

Importar modelos:

<https://docs.unity3d.com/530/Documentation/Manual/HOWTO-importObject.html>

<https://docs.unity3d.com/Manual/HOWTO-importObject.html>

Sistema de navegación:

<https://docs.unity3d.com/es/530/Manual/nav-NavigationSystem.html>

<https://docs.unity3d.com/es/530/Manual/class-NavMeshAgent.html>

Raycast:

<https://docs.unity3d.com/ScriptReference/Physics.Raycast.html>

Controlador de animaciones:

<https://docs.unity3d.com/es/current/Manual/class-AnimatorController.html>