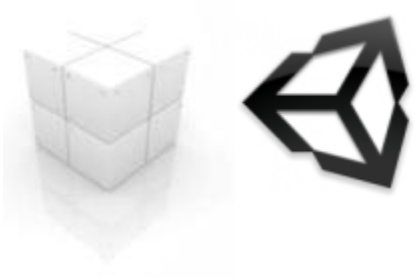
UNITY 3D MANUAL



By: Deodato Pechir.

INDICE DINAMICO – SCRIPTING FOR GAMEPLAY

MENÚ:

1.1 1st/3th PERSON VIEW

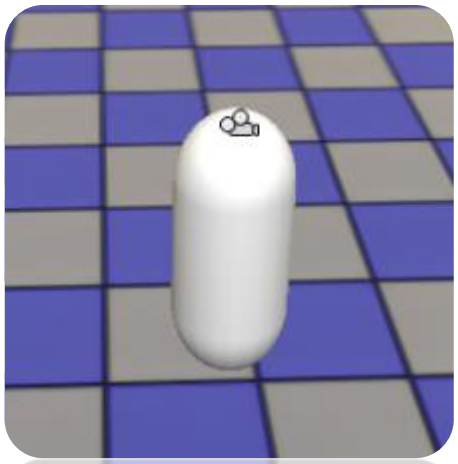
1th & 3th Person View – Projector (Shadows):

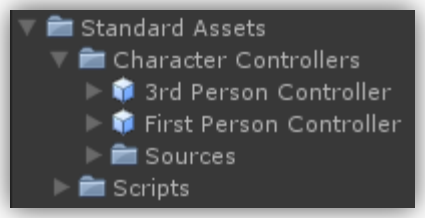
1. Agregamos un Plano (ya tendrá colisión).
2. Agregamos una luz en la escena.
3. Tenemos que tener agregado el paquete de "**Character Controller**", si no se deberá de importar el paquete (Assets>Packaged). vamos a la carpeta de "**Standard Assets>Character Controllers"** y arrastramos "**First/3rd Person Controller**" a la escena.
4. Borrar cualquier otra cámara en la escena. Ahora podras usar al personaje que contiene una cámara integrada.
5. Agregamos el Shadow Projector (paquete), y emparentamos la sombra.
6. **Uso de Projectos como sombras, para incrementar rendimiento y no uso de sombras en tiempo real.**

CREAR COLISIONES CON OBJETOS DINAMICOS DE LA ESCENA:

El personaje contiene el componente de "**Character Controller**" y para que este completo componente interactue con **fisicas** de otros objetos, entramos en la referencia de script en unity y buscamos: OnControllerColliderHit → al ahcer click en la liga este componente del charactercontroller nos permitira que al chocar con un objeto detecte la posicion y lo lanze con una fuerza (golpe entre objetos).

Nota: podrás mover al personaje usando las teclas "WASD" ó las "flechas", así como la barra de espacio para saltar, y no puedes tener más de 2 controles de personajes en la misma escena.



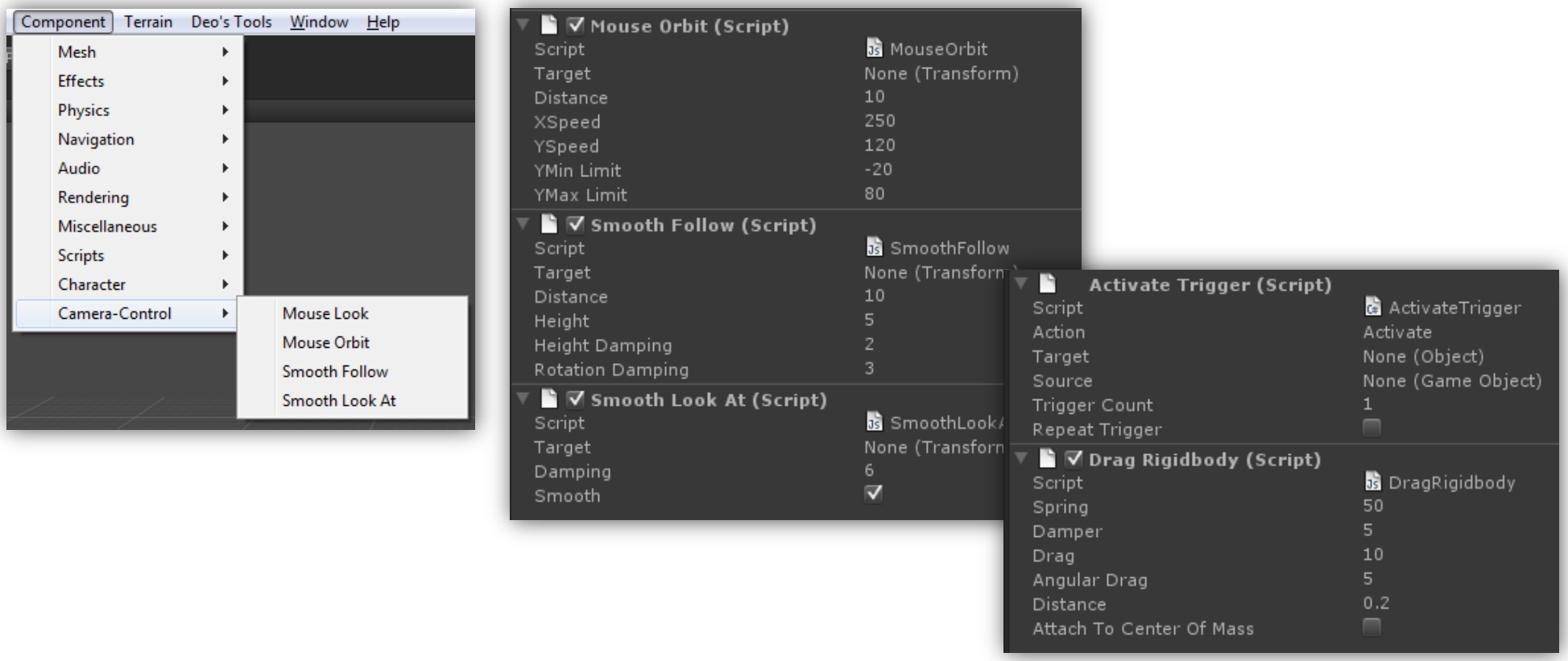


1.2 UNITY SCRIPTS

Unity Scripts:

Principales Scripts Integrados en Unity para control de Cámaras:

- ✓ **Mouse Look**: Permite que la cámara pueda mirar en cualquiera dirección de X & Y o restringir en estos mismos ejes.
- ✓ **Mouse Orbit**: Permite que la cámara siga un target, además de que pueda rotar alrededor del target.
- ✓ **Mouse Follow**: A la cámara se le asigna un Target y lo desplazara a donde este el target, pero además permite que a donde se mueva el mouse el target cuando camine se dirija a esa dirección.
- ✓ **Mouse Look At**: A la cámara se le asigna un Target y este siempre estará mirándolo a donde se mueva.
- ✓ **Activate Trigger**: Nos permite controlar cualquier objeto para activar si es una luz, una animación, etc.
- ✓ **Drag Rigidbody:** Nos permite objetos con Rigidbodies puedan ser arrastrados con la interaccion del Puntero del mouse.



1.3 INTRO TO JAVA & C# SCRIPT.

Intro to Java & C# Scripting:

SINTAXIS:

El Scripting es la forma en la que el usuario crea/define el comportamiento del juego (o las normas) en Unity. El lenguaje de programación recomendado para Unity es **JavaScript**, aunque **C Sharp** o **Boo Script** pueden ser igualmente usados. Pueden pensar en la API como un código que ya ha sido escrito para uno y que permite concentrarte en el diseño de tu juego y acelerar el tiempo de desarrollo. Un buen entendimiento de estos principios básicos es esencial para aprovechar todo el poder de Unity.

Convenciones de nomenclatura:

Antes de empezar es conveniente mencionar algunas convenciones de Unity.

Variables (variables) - empiezan con una letra minúscula. Las variables se usan para almacenar información sobre cualquier aspecto de un estado de juego.

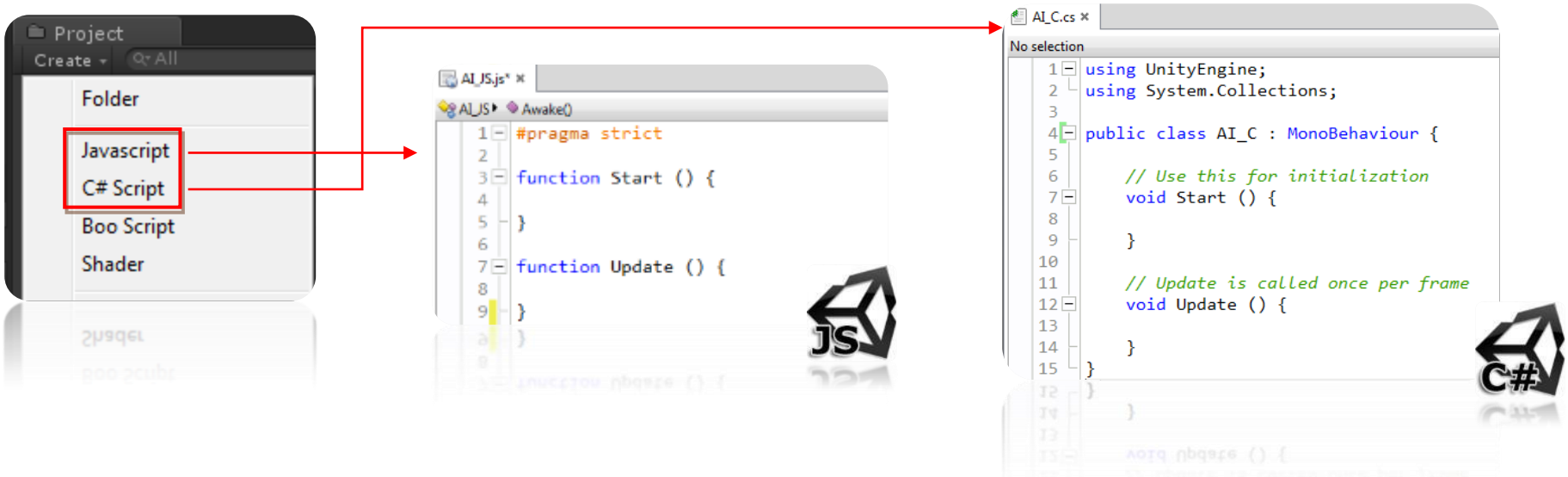
Functions (funciones) - empiezan con una letra mayúscula. Las funciones son bloques de códigos que han sido escritos una vez y que se pueden rehusar tantas veces como sea necesario.

Classes (clases) - empiezan con una letra mayúscula. Éstos pueden tomarse como colecciones de funciones.

Crear Scripts: Selecciona Assets->Create->JavaScrip/C# Script.

Update: Esta función manda llamar cada frame. Esta es la función más usual en juegos, exceptuándolo en el uso de código para **Físicas**.

FixedUpdate: Esta función es para mandar llamar en cada paso de las físicas. Estés es la función para normalmente scripting con físicas.



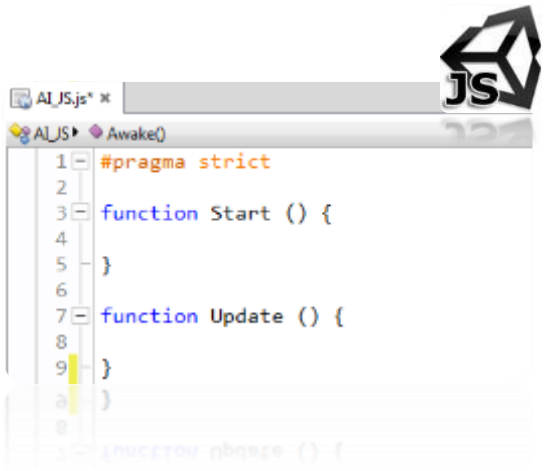
Java Script:

Al iniciar y crear un nuevo C# Script Unity te da los siguiente código:

```
#pragma strict
```
Funcion usada para optimizer procesos para publicar en Flash.

```
function Start () {

}
```
Declarar funcion de Start (al iniciar el juego se ejecutara).

```
function Update () {

}
```
Declarar funcion de Update (Constantemente se ejecuta).



C# Script:

Al iniciar y crear un nuevo C# Script Unity te da los siguiente código:

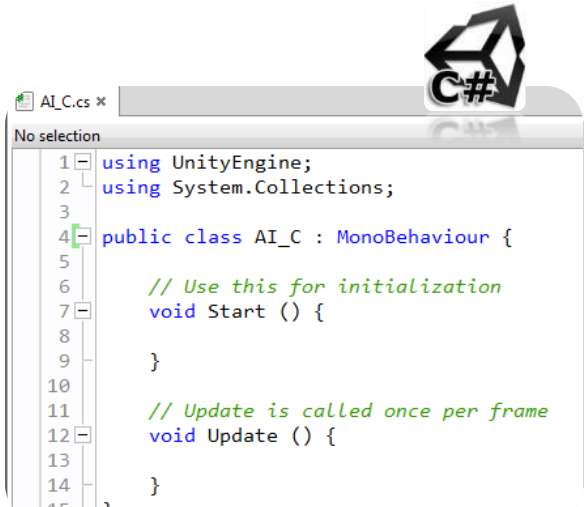
```
using UnityEngine;
using System.Collections;
```
Estas líneas al inicio del script son necesaria para que C# mande llamar directamente al Engine de Unity.

Al iniciar un nuevo documento de C# siempre le pondrá como nombre a la clase "NewBehavoiurScript", este es el default que Unity le asigna como nombre, en este caso el nombre de **Script** y el de la **Clase, tienen que ser el mismo nombre,** debemos de crear una clase y dentro de esta deberá de ir todo el contenido que desarrollaremos para programar.

```
public class AI_C : MonoBehaviour {
        // Use this for initialization
        void Start () {

        }
```
→ La forma en que se declarar funcion de Start (al iniciar el juego se ejecutara).

En **C# TODA** accion que se desee ejecutar debe de ir **dentro de una function** (**void**) forsozamente (Mejor performance).

```
        // Update is called once per frame
        void Update () {

        }
}
```
→ La forma en la que se declarar funcion de Update (Constantemente se ejecuta).



1.4 TYPES OF FUNCTIONS IN JS & C#

Type of Functions in C# Script:

EVENTOS ESCENAS:

void Update () – Es llamado cada frame.
void LateUpdate () – Es llamado cada frame. Si el "Behaviour" esta "activado" - [Usar siempre en camaras].
void FixedUpdate () – Es llamado cada Fixed frame (Físicas).
void Awake() – Es llamado cuando una instancia de script se está cargado (antes).
void Start () – Es llamado al inicio de la escena cuando esta se haya cargado (después).
void Reset () – Reinicia los valores por default.

EVENTOS PARA MOUSE:

void OnMouseEnter () – Es llamado cuando el mouse Entra en el GUIElement o un Collider.
void OnMouseOver () – Es llamado cada frame cuando el mouse esta Encima del GUIElement o Collider.
void OnMouseExit () – Es llamado cuando el mouse ya no está más sobre GUIElement o Collider.
void OnMouseDown () – Es llamado cuando el mouse Presiono botón sobre un GUIElement o Collider.
void OnMouseUp () – Es llamado cuando el mouse Soltó el botón sobre un GUIElement o Collider.
void OnMouseDrag () – Es llamado cuando el mouse Presiono botón sobre un GUIElement o Collider y aun continua presio.

EVENTOS TRIGGERS:

void OnTriggerEnter () – Es llamado cuando el Collider u otros entran en el Trigger.
void OnTriggerExit() – Es llamado cuando el Collider u otros han parado de tocar en el Trigger.
void OnTriggerStay () – Es llamado 1 vez por Frame por cada Collider u otros que están tocando al Trigger.

EVENTOS COLISIONADORES:

void OnCollisionEnter () – Es llamado cuando este Collider/rigidbody a comenzado a tocar otro rigidbody/Collider.
void OnCollisionExit() – Es llamado cuando este Collider/rigidbody ha dejado de tocar a otro Collider/rigidbody.
void OnCollisionStay() – Es llamado 1 vez por frame cada que este Collider/rigidbody está tocando otro Collider/rigidbody.

EVENTOS VISIBLES:

void OnBecameVisible () – Es llamado cuando el Render se ha cambiado a Visible por cualquier cámara.
void OnBecameInvisible() – Es llamado cuando el Render se ha cambiado a Invisible por cualquier cámara.

Type of Functions in Java Script:

EVENTOS ESCENAS:

function Update () – Es llamado cada frame.
function LateUpdate () – Es llamado cada frame. Si el "Behaviour" esta "activado" - [Usar siempre en camaras].
function FixedUpdate () – Es llamado cada Fixed frame (Físicas).
function Awake() – Es llamado cuando una instancia de script se está cargado (antes).
function Start () – Es llamado al inicio de la escena cuando esta se haya cargado (después).
function Reset () – Reinicia los valores por default.

EVENTOS PARA MOUSE:

function OnMouseEnter () – Es llamado cuando el mouse Entra en el GUIElement o un Collider.
function OnMouseOver () – Es llamado cada frame cuando el mouse esta Encima del GUIElement o Collider.
function OnMouseExit () – Es llamado cuando el mouse ya no está más sobre GUIElement o Collider.
function OnMouseDown () – Es llamado cuando el mouse Presiono botón sobre un GUIElement o Collider.
function OnMouseUp () – Es llamado cuando el mouse Soltó el botón sobre un GUIElement o Collider.
function OnMouseDrag () – Es llamado cuando el mouse Presiono botón sobre un GUIElement o Collider y aun continua presio.

EVENTOS TRIGGERS:

function OnTriggerEnter () – Es llamado cuando el Collider u otros entran en el Trigger.
function OnTriggerExit() – Es llamado cuando el Collider u otros han parado de tocar en el Trigger.
function OnTriggerStay () – Es llamado 1 vez por Frame por cada Collider u otros que están tocando al Trigger.

EVENTOS COLISIONADORES:

function OnCollisionEnter () – Es llamado cuando este Collider/rigidbody a comenzado a tocar otro rigidbody/Collider.
function OnCollisionExit() – Es llamado cuando este Collider/rigidbody ha dejado de tocar a otro Collider/rigidbody.
function OnCollisionStay() – Es llamado 1 vez por frame cada que este Collider/rigidbody está tocando otro Collider/rigidbody.

EVENTOS VISIBLES:

function OnBecameVisible () – Es llamado cuando el Render se ha cambiado a Visible por cualquier cámara.
function OnBecameInvisible() – Es llamado cuando el Render se ha cambiado a Invisible por cualquier cámara.

1.5 TYPE OF VARIABLES IN JS & C#

Type of Variables in C# Script (C#):

Al crear variables podemos almacenar dentro de estas valores, y hay diferentes tipos de variables que podemos crear como: Strings (combinación de letras y números), float (Decimales), Int (Enteros), boolean (Boleanos 0-1 / on-off / true-false), Vector3 (XYZ / RGB).

Las variables tienen que ser **declaradass** de la siguiente manera: **C# Script (C#)** → **TipoVariable** NombreVariable **= valor;** una vez declarada la variable no es necesario usar **": TipoVariable"** de nuevo, solo con poner "NombreVariable **= valor".**

- **TipoVariable:** **El tipo de variable a crear.**
- **Valor:** **El valor que tendrá la variable.**

Ejemplos de Variables Publicas: **int** Deo_int **= 200; | float** Deo_float **= 10.5f; | String** Deo_String **= "Hola Mundo";**

```
String          Valor1;     //Textos.
float           Valor2;     //Valores decimales y requieren terminar en "f".
int             Valor3;     //Valores enteros.
boolean         Valor4;     //Valores boleanos 0, 1,true, false.
GameOject       Valor5;     //Objetos dentro de la escena.
Transform       Valor6;     //Valores de transform de un GameObject (mover, rotar, escalar).
Rigidbody       Valor7;     //Valor de un GameObject de tipo rigidbody.
Collision       Valor8;     //Valor de un GameObject cuando Colisiona.
AudioClip       Valor9;     //Valor de un GameObject de tipo sonido.
ParticleEmitter Valor10;    //Valor de un GameObject de tipo partículas.
Texture2D       Valor11;    //Valor de una Textura.
Camera          Valor12;    //Valor de un GameObject de tipo Cámara.
Light           Valor13;    //Valor de un GameObject de tipo Luces.
CharacterController Valor14;//Valor de un GameObject de tipo Control de personaje (1ra y 3ra Persona).
Color           Valor15;    //Valor de cambio/asignación de color en una variable.
Material        Valor16;    //Valor de un GameObject de tipo Material.
AnimationClip   Valor17;    //Valor de un GameObject de tipo Animación (Clips).
Renderer        Valor18;    // Valor de un GameObject de tipo Render materiales/color/visible, etc.
Arreglos CualquierTipoVariable [] ;  // Valor para crear variable de tipo de arreglos.
```

Variables Públicas - Permite comunicarse entre los Scripts del mismo **GameObject u otros GameObjects (Si se publica en Inspector)**:
public String DeoValor **= "Hola a todos"**;

Variables Públicas Ocultas - Para crear variables publicas y estas no sean mostradas en el inspector. Esta línea va arriba de nuestra variable que deseamos no mostrar y si después hay más variables definidas si se mostraran en el inspector: **[System**.NonSerialized**]**

Variables Privadas Permite comunicarse entre los Scripts del mismo **GameObject u otros GameObjects (No se publica en Inspector)**:
private String DeoValor **= "Hola a todos"**; / **String** DeoValor **= "Hola a todos"**;

Variables Globales - Permite crear variables globales para que se peudan comunicar entre todos los Scripts de la escena **(No se publica en Inspector)**:
static String DeoValor **= "Hola a todos"**;

Conversion de tipos entre Variables → Numéricas a Strings | Strings a Numéricas:

int Numerico = 60**;** → Numerico.ToString()**; | String** Alfabetico **= "100" ;** → **int**.Parse (Alfabetico) **;**

Type of Variables in Java Script (JS):

Al crear variables podemos almacenar dentro de estas valores, y hay diferentes tipos de variables que podemos crear como: Strings (combinación de letras y números), float (Decimales), Int (Enteros), boolean (Boleanos 0-1 / on-off / true-false), Vector3 (XYZ / RGB).

Las variables tienen que ser **declaradass** de la siguiente manera: **JavaScript (JS)** → var NombreVariable : **TipoVariable = valor;** una vez declarada la variable no es necesario usar "**var" + "** : **TipoVariable"** de nuevo, solo con poner "NombreVariable **= valor".**

- **Var:** **Definir variable por priemra vez.**
- **Variable:** **El nombre de la variable.**
- **Valor:** **El valor que tendrá la variable.**

Variables Locales desde en el Inspector: **var** Deo_int : **int = 200; | var** Deo_float : **float = 10.5; | var** Deo_String : **String = "Hola Mundo";**

```
var Valor1 : String;                    //Textos.
var Valor2 : float;                     // Valores decimales.
var Valor3 : int;                       //Valores enteros.
var Valor4 : boolean;                   //Valores boleanos 0, 1,true, false.
var Valor5 : GameObject;                //Objetos dentro de la escena.
var Valor6 : Transform;                 //Valores de transform de un GameObject (mover, rotar, escalar).
var Valor7 : Rigidbody;                 //Valor de un GameObject de tipo rigidbody.
var Valor8 : Collision;                 //Valor de un GameObject cuando Colisiona.
var Valor9 : AudioClip;                 //Valor de un GameObject de tipo sonido.
var Valor10 : ParticleEmitter;          //Valor de un GameObject de tipo partículas.
var Valor11 : Texture2D;                //Valor de una Textura.
var Valor12 : Camera;                   //Valor de un GameObject de tipo Cámara.
var Valor13 : Light;                    //Valor de un GameObject de tipo Luces.
var Valor14 : CharacterController;      //Valor de un GameObject de tipo Control de personaje (1ra y 3ra Persona).
var Valor15 : Color;                    //Valor de cambio/asignación de color en una variable.
var Valor16 : Material;                 //Valor de un GameObject de tipo Material.
var Valor17 : AnimationClip;            //Valor de un GameObject de tipo Animación (Clips).
var Valor18 : Renderer;                 // Valor de un GameObject de tipo Render materiales/color/visible, etc.
var Valor[] : CualquierTipoVariable;    // Valor para crear variable de tipo de arreglos.
```

Variables Públicas - Permite comunicarse entre los Scripts del mismo **GameObject u otros GameObjects (Si se publica en Inspector)**:
public var DeoValor : **String = "Hola a todos"**;

Variables Públicas Ocultas - Para crear variables publicas y estas no sean mostradas en el inspector. Esta línea va arriba de nuestra variable que deseamos no mostrar y si después hay más variables definidas si se mostraran en el inspector: **@System.NonSerialized**

Variables Privadas Permite comunicarse entre los Scripts del mismo **GameObject u otros GameObjects (No se publica en Inspector)**:
private var DeoValor : **String = "Hola a todos"**;

Variables Globales - Permite crear variables globales para que se peudan comunicar entre todos los Scripts de la escena **(No se publica en Inspector)**:
static Strign DeoValor **= "Hola a todos"**;

Conversion de tipos entre Variables → Numéricas a Strings | Strings a Numéricas:

```
var Numerico : int = 60;   →   Numerico.ToString(); | var Alfabetico : String = "100";   →   int.Parse (Alfabetico);
```

1.6 MATHEMATICAL OPERATIONS.

Mathematical Operation with variables in JS & C#:

IMPRIMIR EN CONSOLA:

Debug.Log **( );** **ó** print **();**

VARIABLES:

```
//Se definen primero las variables para usarlas después.
JS → var Variable1 : int = 10;              C# → int Variable1 = 10;
JS → var VariableValor2 : int = 50;         C# → int Variable2 = 50;
```

Sumar y Restar:

```
JS & C# →  Variable1 + 1;  →  Variable1 ++;
JS & C# →  Variable1 - 1;  →  Variable1 --;
```

Dividir y Multiplicar:

```
JS & C# →  Variable1 = VariableValor2 / 10;  -- Dividir
JS & C# →  Variable1 = VariableValor2 * 10;  -- Multiplicar
```

Suma de valores en variables:

```
JS & C# →  Variable1 = Variable1 + 100;   →   Variable1 += 100;
JS & C# →  Variable1 = Variable1 - 100;   →   Variable1 -= 100;
JS & C# →  Variable1 = Variable1 * 100;   →   Variable1 *= 100;
```

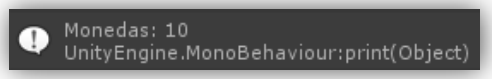
Operaciones entre variables:

```
JS & C# →  Variable1 + VariableValor2;  |  Variable1 - VariableValor2; |  Variable1 * Variable2;  |  Variable1 / Variable2;
```

Concatenar variables de tipo Alfabéticas y Numéricas:

```
JS & C# →  "Monedas: " + Variable1;
```

Resultado de Impresión en Cosola →



1.7 CONDITIONS.

Conditions (IF – ELSE):

- ✓ Las condiciones nos permite hacer comparaciones, de acuerdo al resultado hacemos alguna otra acción.

Java Script:

EJEMPLO: IF

```
var variable_A : int = 1;
var variable_B : int = 2;

Ejemplo de Condiciones:
function Start () {
      if (variable_A > variable_B){
            Debug.Log ("A es mayor que B");
      }else  if (variable_A == variable_B ){
            Debug.Log ("A y B son iguales");
      }else {
            Debug.Log ("B es mayor que A");
      }
}
```

Comparación de 2 o más variables:

```
function Start () {
      if(variable_A > variable_B && variable_A != 0){
            Debug.Log ("La comparación es verdadera");
      }

      if (variable_A > variable_B || variable_A != 0){
            Debug.Log ("La comparación es verdadera ");
      }
}
```

C# Script:

EJEMPLO: IF

```
int variable_A = 1;
int variable_B = 2;

Ejemplo de Condiciones:
void Start () {
      if (variable_A > variable_B){
            Debug.Log ("A es mayor que B");
      }else  if (variable_A == variable_B ){
            Debug.Log ("A y B son iguales");
      }else {
            Debug.Log ("B es mayor que A");
      }
}
```

Comparación de 2 o más variables:

```
void Start () {
      if(variable_A > variable_B && variable_A != 0){
            Debug.Log ("La comparación es verdadera");
      }

      if (variable_A > variable_B || variable_A != 0){
            Debug.Log ("La comparación es verdadera ");
      }
}
```

Funcionaes matematicas de Condiciones:

'>' A es mayor que B
'<' A es Menor que B
'==' A es igual a B
'!=' A No es igual a B

'>=' A Es mayor o igual a B
'<=' A Es menor o igual a B

Modos de comparación de condiciones:
&& 2 Comparaciones al mismo tiempo
|| 1 comparación u otra.

1.8 LOOPS (WHILE & FOR).

Loops (While & For/Foreach):

- ✓ Este tipo de estado debe de ir en una función de tipo **Start**(); While lopps nos permite repetir todo un bloque de código hasta que se cumpla una condición. En si el While sirve para repetir "**Mientras**" se cumple la condición.

Java Script:

EJEMPLO: WHILE

```
var Valor : int = 0;

function Start () {
	// Si es igual a 10 o mayor se cumpla el loop.
	while (Valor <= 10) {
		// se hace aquí la acción que queremos repetir.
		print("Cantidad: " + Valor);
		// Agregamos el valor de 1, cada q se hace un loop.
		Valor ++;
	}
}
```

EJEMPLO: FOR

```
var Valor : int = 0;

// Se define la variable, su límite y el incremento.
function Start () {
	for (Valor = 0; Valor <= 10; Valor ++) {

		// Se hace aquí la acción que queremos repetir.
		print ("Cantidad: " + Valor);

	}
}
```

EJEMPLO: FOR IN

```
var Lista : String;

function Start () {
	for (Lista in nombres){
		// Imprimir informacion
		print (Lista);
	}
}
```

C# Script:

EJEMPLO: WHILE

```
int Valor = 0;

void Start () {
	// Si es igual a 10 o mayor se cumpla el loop.
	while (Valor <= 10) {
		// se hace aquí la acción que queremos repetir.
		print("Cantidad: " + Valor);
		// Agregamos el valor de 1, cada q se hace un loop.
		Valor ++;
	}
}
```

EJEMPLO: FOR

```
int Valor = 0;

// Se define la variable, su límite y el incremento.
void Start () {
	for (Valor = 0; Valor <= 10; Valor ++) {

		// Se hace aquí la acción que queremos repetir.
		print ("Cantidad: " + Valor);

	}
}
```

EJEMPLO: FOR IN

```
String Lista;

function Start () {
	foreach (Lista in nombres){
		// Imprimir informacion
		print (Lista);
	}
}
```

Switch:

- ✓ Este estado nos permite tomar de una sola variable su valor y realizar múltiples operaciones dependiendo del valor de la variable.

Java Script:

EJEMPLO:

```
var CambioArma : String = "A o B";

void Start() {

      switch (CambioArma) {
          case "A":
              Debug.Log ("Cambio a Metralleta");
              // termina la acción en este punto
              break;

              case "B":
              Debug.Log ("Cambio a Lanza Misiles");
              // termina la acción en este punto
              break;

             // si no hay ningún caso, se ejecuta el de default (nada).
              default:
             // termina la acción en este punto
              break;
      }
}
```

C# Script:

EJEMPLO:

```
public class Ejemplo : MonoBehaviour {

      String CambioArma = "A o B";

      void Start() {
            switch (CambioArma){
                case "A":
                Debug.Log ("Cambio a Metralleta");
                // termina la acción en este punto
                break;

                case "B":
                Debug.Log ("Cambio a Lanza Misiles");
                // termina la acción en este punto
                break;

                // si no hay ningún caso, se ejecuta el de default (nada).
                default:
                // termina la acción en este punto
                break;
            }

      }

}
```

1.9 CREATE & CALL FUNCTIONS/VOID

Create & Call Functions/Void:

- ✓ Las funciones nos permiten contener múltiples acciones dentro, y se pueden ejecutar con el nombre de la función.

Java Script:

Explicación:

```
//Creación de una función
function Mi_PrimerFuncion () {
     //doSomething
}

//Ejecutar la función
 Mi_PrimerFuncion ();
```

EJEMPLO:

```
//Variables
var Monedas : int = 100;
var Gastos :  int = 10;


function Mi_PrimerFuncion () {
       // Resta de variables
       Monedas = Monedas - Gastos;

       // Imprimir el valor de monedas
       print (Monedas);
}


function Start (){
    //Ejecutar la funcion
    Mi_PrimerFuncion ();
}
```

C# Script:

Explicación:

```
//Creación de una función
void Mi_PrimerFuncion () {
      //doSomething
}

//Ejecutar la función al iniciar
Mi_PrimerFuncion ();
```

EJEMPLO:

```
public class Ejemplo : MonoBehaviour {

      //Variables
      int Monedas = 100;
      int Gastos = 10;

      void Mi_PrimerFuncion () {
         // Resta de variables
          Monedas -= Gastos;

         // Imprimir el valor de monedas
          print (Monedas);
      }

      void Start (){
         //Ejecutar la funcion
          Mi_PrimerFuncion ();
      }

}
```

Create Overloading Functions/Void:

- ✓ Se pueden crear funciones y sobrecargarlas con información, asi dependiendo del valor enviado se ejecutara una de las funciones

Java Script:

EJEMPLO:

```
function PrintType(item : String){
    print ("Soy del tipo de: String");
}

function PrintType(item : int){
   print ("Soy del tipo de: Int");
}

function PrintType(item : float){
   print ("Soy del tipo de: Float");
}

function PrintType(item : boolean){
   print ("Soy del tipo de: Boolean");
}

function PrintType(item : Array){
   print ("Soy del tipo de: Array");
}

function PrintType(item: GameObject){
//Este captura cualquier otro valor.
   print ("Soy del tipo de: GameObject u otro");
}

function PrintType(){
   print ("Sete olvido poner un argumento en la función");
}

function Start () {
   PrintType();
   PrintType("Hola");
   PrintType(true);
}
```

C# Script:

EJEMPLO:

```
public class Ejemplo : MonoBehaviour {

       void PrintType(string  item){
          print ("Soy del tipo de: String");
       }

       void PrintType(int  item){
          print ("Soy del tipo de: Int");
       }

       void PrintType(float  item){
          print ("Soy del tipo de: Float");
       }

       void PrintType(bool item){
       print ("Soy del tipo de: Boolean");
       }

       void PrintType(ArrayList item){
          print ("Soy del tipo de: Array");
       }

       void PrintType(GameObject  item){
       //Este captura cualquier otro valor.
          print ("Soy del tipo de: GameObject u otro");
       }

       void PrintType(){
          print ("Sete olvido poner un argumento en la función");
       }

       void Start (){
            PrintType();
            PrintType("Hola");
            PrintType(true);
       }
}
```

1.10 SETUP & USE ARRAYS

SetUp & Use Arrays (C# Script)

Crear un nueva lista de Arreglos:

```
ArrayList nombres = new ArrayList ();
```

Agregar nuevos valores en el Arreglo:

```
nombres.Add("Juan");      nombres.Add("Pedro");     nombres.Add("Jesus");     nombres.Add("Martin");
```

Almacenar cada uno de los valores del Arreglo por medio de "For in Array"

```
ArrayList InterfaceGUI = new ArrayList ();

void Start () {
       ArrayList nombresGamePlay = new ArrayList {"Lupa", "Arma", "Vida", "PowerUp"};

       foreach (string ListaTemporal in nombresGamePlay) {
              InterfaceGUI.Add (ListaTemporal);
              print (ListaTemporal);
       }
}
```

Obtener de un arreglo su valor por posición o el Total de valores almacenados:

```
nombres[0]; // Por Posicion.      /      nombres.Count; // Cantidad total.
```

Sobre un arreglo eliminar un valor por posición o limpiar todo el arreglo:

```
nombres.RemoveAt(0); // Eliminar el valor de la posición 0.          /      nombres.Clear();    // Limpiar todo el arreglo.
```

Separar en un arreglo el valor de una variable.

```
string Sentencia = "La clase de videjuegos es interesante.";
string[] palabras = Sentencia.Split(" "[0]);
foreach (string Listatemporal in palabras){
       Debug.Log(Listatemporal);
}
```

Substraer de una variable una cantidad de letras.

```
string Sentencia = "La clase de videjuegos es interesante";
       string InicioMitad = Sentencia.Substring(0, 22);
              Debug.Log(InicioMitad);
```

Convertir un valor numérico en String (Texto):

```
int number = 10;
string text = number.ToString();
```

Convertir un valor en Mayusculas / Minusculas.

```
string Sentencia = "La clase de videjuegos es interesante";
              Debug.Log(Sentencia.ToUpper() );
       Debug.Log( ("VIDEOJUEGOS").ToLower() );
```

SetUp & Use Arrays (Java Script)

Crear un Arreglo y agregarle multiples valores:
```
var nombres = new Array (); // Nuevo arreglo vacio.
var nombres = new Array ("Juan","Pedro","Jesus","Martin"); // Nuevo arreglo con valores predefinidos desde el inicio.
```

Agregar nuevos valores en el Arreglo:
```
nombres.add("Deo"); // Individual
nombres.add ("Juan"," Pedro"," Jesus"," Martin");  // Grupo
```

Agregar nuevos valores a un Arreglo por posición:
```
nombres[0] = "Juan";  nombres[1] = "Pedro"; nombres[2] = "Jesus"; nombres[3] = "Martin";
```

Obtener de un arreglo su valor por posición o el Total de valores almacenados:
```
nombres[0]; // Por Posicion.      /      nombres.Length; // Cantidad total
```

Sobre un arreglo eliminar un valor por posición o limpiar todo el arreglo:
```
nombres.RemoveAt(0); // Eliminar el valor de la posición 0.          /      nombres.Clear();    // Limpiar todo el arreglo.
```

Almacenar cada uno de los valores del Arreglo por medio de "For in Array"
```
function Start () {
	for (var Listatemporal:String in nombres){ // Se transfiere los valores del arreglo por el numero de objetos del arreglo.
		print (Lista);
	}
}
```

Separar en un arreglo el valor de una variable.
```
var Sentencia : String = "La clase de videjuegos es interesante ";
var palabras : String[]  = Sentencia.Split(" "[0]);
Debug.Log("Mira el Inspector");
```

Substraer de una variable una cantidad de letras.
```
var Sentencia: String = "La clase de videjuegos es interesante";
var InicioMitad: String = Sentencia.Substring(0, 22);
Debug.Log(InicioMitad);
```

Convertir un valor numérico en String (Texto):
```
var number: int = 10;
var text : String = number.ToString();
```

Convertir un valor en Mayusculas / Minusculas.
```
var Sentencia : String = "La clase de videjuegos es interesante";
Debug.Log(Sentencia.ToUpper() );
Debug.Log( ("VIDEOJUEGOS").ToLower() );
```

1.11 TRANSFORM POSITION/TRANSLATE & DELTA TIME.

Transforms Position/Translate/Delta Time (C# Script):

Transform.position -- Es para: "obtener/mover" la posición que un objeto con en el método de world space.
Vector3 position

Time.deltaTime -- Se usa para hacer que la velocidad del juego sea independiente en frames/s, para que sea en relación al tiempo y no a frames.
float **deltaTime**

Ejemplos: Vector3.zero → (**0,0,0**) | Vector3.one → (**1,1,1**) | Vector3.up → (0,**1**,0) | Vector3.forward → (0,0,**1**) | Vector3.right → (**1**,0,0,).
Ejemplos: transform.zero → (**0,0,0**) | transform.one → (**1,1,1**) | transform.up → (0,**1**,0) | transform.forward → (0,0,**1**) | transform.right → (**1**,0,0,).

CAMBIO DE POSICION SOBRE LOS GAMEOBJECTS.

```
// Cambiar la posición de un objeto en X,Y,Z.
transform.position = new Vector3 (0, 0, 0);    |    transform.position = Vector3.zero;


// Cambiar la posición de un objeto en  un solo canal de X,Y o Z.
transform.position= new Vector3(10,0,0); | transform.position= new Vector3(0,10,0); | transform.position= new Vector3(0,0,10);
ó
transform.position= transform.right*10; | transform.position= transform.up*10; | transform.position= transform.forward*10;
```

MOVER OBJETOS CONTANTEMENTE DE ACUERDO AL TIEMPO/SEGUNDOS Y NO AL FRAMERATE.
Una manera de usar el Vector3 = **forward** (eje azul), **up** (eje verde), **right** (eje rojo).

```
void Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
      transform.position += transform.forward * 0.2f * Time.deltaTime;
}

void Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
      transform.position += transform.up * 0.2f * Time.deltaTime;
}

void Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
      transform.position += transform.right * 0.2f * Time.deltaTime;
}
```

ANIMACIÓN DE MOVIMIENTO CONSTANTE.

Transform.Translate -- Es para trasladar un objeto en cualquiera de los ejes X, Y, Z con selección del Space.
transform .Translate (Vector3 **translation Axis**, relativeTo : Space = **Space.Self/Space.World**) : void

Ejemplo:

```
void Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
	// Mueve al objeto hacia arriba en el "Local  Space" 1 unidad/segundo.
	transform.Translate (Vector3.up * Time.deltaTime, Space.Self);
}
```

ANIMACIÓN DE MOVIMIENTO LINEAL A --> B.

Vector3.Lerp -- Desplaza al GameObject con una interpolación de un punto de inicio a un destino, con velocidad desacelerada.
Lerp (**from**: Vector3, **to**: Vector3, **t**: float) : Vector3

Vector3.MoveTowards -- Desplaza al GameObject con una interpolación de un punto de inicio a un destino, con velocidad constante.
MoveTowards (**current** : Vector3, **target** : Vector3, **maxDistanceDelta** : float) : Vector3

Ejemplo:

```
void Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
	// Animacion - Inicio acelerado y termina desacelerado.
	transform.position = Vector3.Lerp (transform.position, new Vector3 (10, 0, 2), Time.deltaTime * 2);
}

void Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
	// Animacion - Constante al iniciar y terminar.
	transform.position = Vector3.MoveTowards (transform.position, new Vector3 (10, 0, 2), Time.deltaTime * 2);
}
```

ANIMACIÓN LINEAL PINGPONG (IDA Y REGRESO).

static **function PingPong** (t : float, length : float) : float
"t" es el valor de pingong, de modo que nunca es mayor que "lenght" ni menor a 0. El valor devuelto se moverá hacia atrás y adelante entre 0 y longitud.

```
public float Velocidad;
public float Distancia = 0.01f; // Tiene que tener al menos un minimo la distancia si no marcara error.
public float Altura;
float PosY;

void Update () {
	PosY = Mathf.PingPong ( Time.time * Velocidad, Distancia ) + Altura;
	transform.position = new Vector3 (transform.position.x, PosY, transform.position.z);
}
```

Transforms Position/Translate/Delta Time (Java Script):

Transform.position -- Es para: "obtener/mover" la posición que un objeto con en el método de world space.
var position : Vector3

Time.deltaTime -- Se usa para hacer que la velocidad del juego sea independiente en frames/s, para que sea en relación al tiempo y no a frames.
var **deltaTime :** float

Ejemplos: Vector3.zero → (**0,0,0**) | Vector3.one → (**1,1,1**) | Vector3.up → (0,**1**,0) | Vector3.forward → (0,0,**1**) | Vector3.right → (**1**,0,0,).
Ejemplos: transform.zero → (**0,0,0**) | transform.one → (**1,1,1**) | transform.up → (0,**1**,0) | transform.forward → (0,0,**1**) | transform.right → (**1**,0,0,).

CAMBIO DE POSICION SOBRE LOS GAMEOBJECTS.

```
transform.position = Vector3 (0, 0, 0);    |    transform.position = Vector3.zero; // Cambiar la posición de un objeto en X,Y,Z.
                                                transform.position = transform.zero; // Cambiar la posición de un objeto en X,Y,Z.
```

```
// Cambiar la posición de un objeto en  un solo canal de X,Y o Z.
transform.position.x = 10;  |   transform.position.y = 10;  |  transform.position.z = 10;
```

MOVER OBJETOS CONTANTEMENTE DE ACUERDO AL TIEMPO/SEGUNDOS Y NO AL FRAMERATE.
Una manera de usar el Vector3 = **forward** (eje azul), **up** (eje verde), **right** (eje rojo).

```
function Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
      transform.position += transform.forward * 0.2 * Time.deltaTime;
}

function Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
      transform.position += transform.up * 0.2 * Time.deltaTime;
}

function Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
      transform.position += transform.right * 0.2 * Time.deltaTime;
}
```

ANIMACIÓN DE MOVIMIENTO CONSTANTE.

Transform.Translate -- Es para trasladar un objeto en cualquiera de los ejes X, Y, Z con selección del Space.
transform .Translate (**translation** : Vector3, relativeTo : Space = **Space.Self/Space.World**) : void

Ejemplo:

```
function Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
	// Mueve al objeto hacia arriba en el "Local  Space" 1 unidad/segundo.
	transform.Translate (Vector3.up * Time.deltaTime, Space.Self);
}
```

ANIMACIÓN DE MOVIMIENTO LINEAL A --> B.

Vector3.Lerp -- Desplaza al GameObject con una interpolación de un punto de inicio a un destino, con velocidad desacelerada.
Lerp (**from** : Vector3, **to** : Vector3, **t** : float) : Vector3

Vector3.MoveTowards -- Desplaza al GameObject con una interpolación de un punto de inicio a un destino, con velocidad constante.
MoveTowards (current : Vector3, target : Vector3, maxDistanceDelta : float) : Vector3

Ejemplo:

```
function Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
	// Animacion - Inicio acelerado y termina desacelerado.
	transform.position = Vector3.Lerp (transform.position, Vector3 (10, 0, 2), Time.deltaTime * 2);
}

function Update / LateUpdate () { // Update para personajes y LateUpdate para cámaras.
	// Animacion - Constante al iniciar y terminar.
	transform.position = Vector3.MoveTowards (transform.position, Vector3 (10, 0, 2), Time.deltaTime * 2);
}
```

ANIMACIÓN LINEAL PINGPONG (IDA Y REGRESO).

static **function PingPong** (t : float, length : float) : float
"t" es el valor de pingong, de modo que nunca es mayor que "lenght" ni menor a 0. El valor devuelto se moverá hacia atrás y adelante entre 0 y longitud.

```
var Velocidad : float ;
var Distancia : float = 0.1;     // Tiene que tener al menos un minimo la distancia si no marcara error.
var Altura : float;

function Update () {
	transform.position.y = Mathf.PingPong (Time.time * Velocidad, Distancia) + Altura;
}
```

1.12 TRANSFORM ROTATION & SCALE

Transforms de Rotacion y Escala (C# Script):

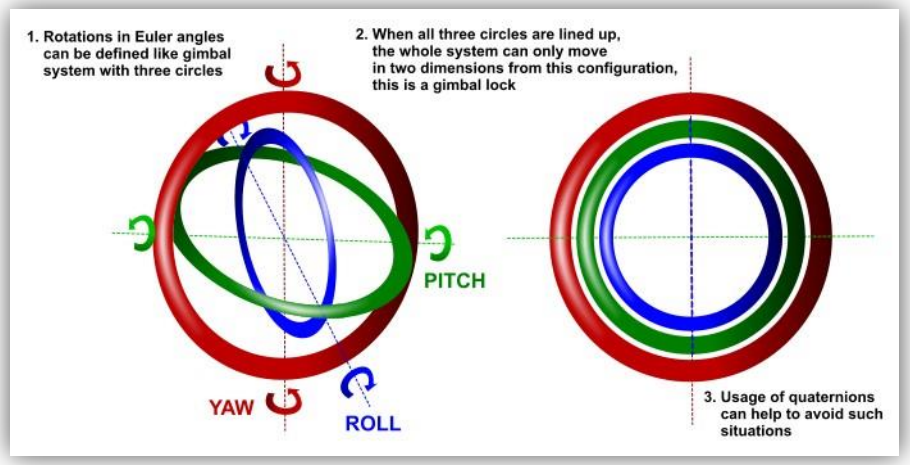
ASIGNACION DE ROTACIÓN DE GAMEOBJETO:

transform.rotation = Quaterion.Euler (x, y, z); → **Quaterion** en ejes de X, Y & Z.

Ejemplo:

```
// Es la rotacion de Angulos de Euler en grados.
void Start () {
    transform.rotation = Quaternion.Euler (0, 45, 0);
}
```

```
// Es la rotacion en Angulos Euler en grados, al parent de manera local.
void Start () {
    transform.rotation = Quaternion. localEulerAngles (0, 45, 0);
}
```



ANIMACION DE ROTACION CONSTANTE:

Transform.Rotate → function **Rotate** (**eulerAngles** : Vector3, **relativeTo** : Space = **Space.Self/World**) : void

Ejemplo:

```
void Update() {
      // Pone al objeto a rotar en el eje de las X 20 grados/segundo.
      transform.Rotate (new Vector3 (20 * Time.deltaTime, 0, 0), Space.Self);
}
```

ANIMACION DE ROTACION SUAVE:

transform.rotation = Quaterion.Lerp (to : Vector3**, from :** Vector3**, t:** float**);** → **Quaterionn** con rotacion con suavidad (Rotaciones Cortas).
transform.rotation = Quaterion.Slerp (to : Vector3**, from :** Vector3**, t:** float**);** → **Quaterionn** con rotacion con suavidad (Rotacion Largas).
transform.rotation = Quaterion.RotateTowards (to : Vector3**, from :** Vector3**, t:** float**);**→ **Quaterionn** con rotacion constante de Inicio a Final.

Ejemplo:

```
public int Rotador;

void Update() {
      // Rotar con suavidad (Fade).
      transform.rotation = Quaternion.Lerp (transform.rotation, Quaternion.Euler (0, Rotador, 0), Time.deltaTime * 1);
}
```

Ejemplo:

```
public int Rotador;

void Update() {
  // Rotar sin suavidad.
  transform.rotation = Quaternion.RotateTowards (transform.rotation, Quaternion.Euler (0, Rotador, 0), Time.deltaTime * 100);
}
```

ANIMACION DE ESCALAMIENTO:

Transform.localScale → Escala al objeto en el espacio local del GameObjetc **Completa** o por **Eje**.

Ejemplo:

```
void Update () {

      // En los 3 ejes estatico.
      transform.localScale = new Vector3 (0.5f * Time.deltaTime, 0.0f, 0.0f);
      // En los 3 ejes Dinámico.
      transform.localScale += new Vector3 (0.5f * Time.deltaTime, 0.0f, 0.0f);      ó
      // En un solo eje Dinámico.
      transform.localScale += Vector3.right * 2 * Time.deltaTime;
}
```

Transforms de Rotacion y Escala (Java Script):

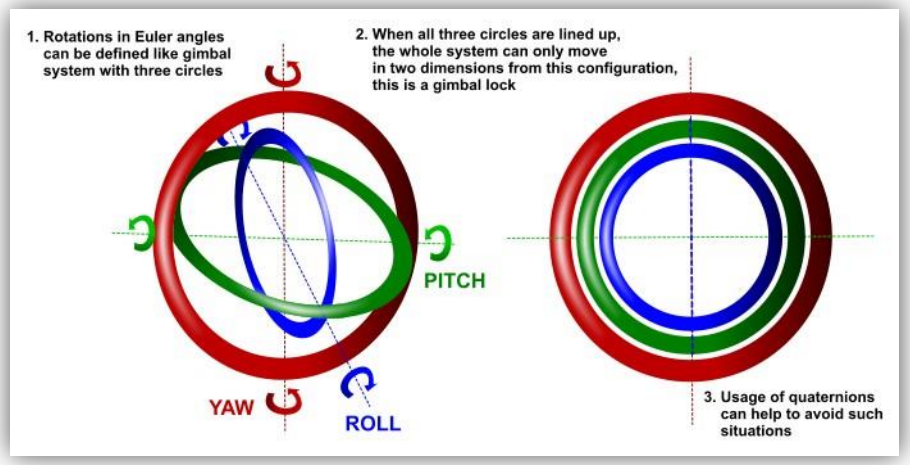
ASIGNACION DE ROTACIÓN DE GAMEOBJETO:

transform.rotation = Quaterion.Euler (x,y,z); → **Quaterion** en ejes de X, Y & Z.

Ejemplo:

```
// Es la rotacion de Angulos de Euler en grados.
function Start () {
    transform.rotation = Quaternion.Euler(0, 45, 0);
}
```

```
// Es la rotacion en Angulos Euler en grados, al parent de manera local.
function Start () {
    transform.rotation = Quaternion. localEulerAngles (0, 45, 0);
}
```



ANIMACION DE ROTACION CONSTANTE:

Transform.Rotate → function **Rotate** (**eulerAngles** : Vector3, **relativeTo** : Space = **Space.Self/World**) : void

Ejemplo:

```
function Update() {
      // Pone al objeto a rotar en el eje de las X 20 grados/segundo.
      transform.Rotate (Vector3(20 * Time.deltaTime, 0, 0), Space.Self);
}
```

ANIMACION DE ROTACION SUAVE:

transform.rotation = Quaterion.Lerp (to : Vector3**, from :** Vector3**, t:** float**);** → **Quaterionn** con rotacion con suavidad (Rotaciones Cortas).
transform.rotation = Quaterion.Lerp (to : Vector3**, from :** Vector3**, t:** float**);** → **Quaterionn** con rotacion con suavidad (Rotacion Largas).
transform.rotation = Quaterion.Lerp (to : Vector3**, from :** Vector3**, t:** float**);** → **Quaterionn** con rotacion constante de Inicio a Final.

Ejemplo:

```
var Rotador : int;

function Update() {
	// Rotar con suavidad.
	transform.rotation = Quaternion.Lerp (transform.rotation, Quaternion.Euler (0, Rotador, 0), Time.deltaTime * 1);

}
```

Ejemplo:

```
var Rotador : int;

function Update() {
   // Rotar constantemente.
  transform.rotation = Quaternion.RotateTowards (transform.rotation, Quaternion.Euler (0, Rotador, 0), Time.deltaTime * 100);
}
```

ANIMACION DE ESCALAMIENTO:

Transform.localScale → Escala al objeto en el espacio local del GameObjetc **Completa** o por **Eje**.

Ejemplo:

```
function Update () {
	// En los 3 ejes
	transform.localScale += Vector3 (0.5f * Time.deltaTime, 0.0f, 0.0f);
	ó
	// En un solo eje
	transform.localScale += Vector3.right * 2 * Time.deltaTime;
}
```

1.13 INPUTS (CONTROLLER).

Inputs Controller (C# Script):

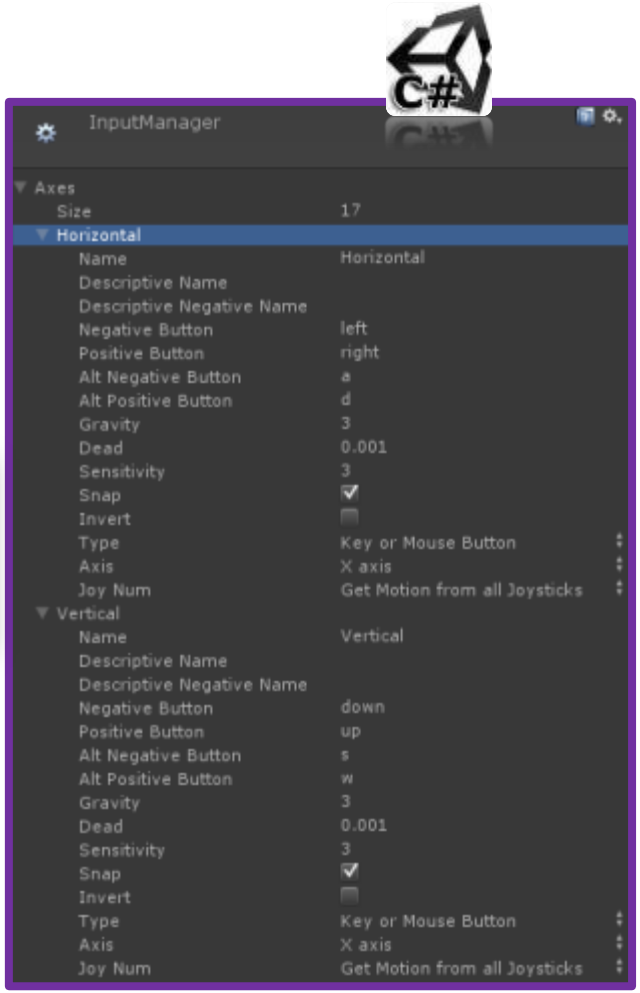
✓ Los inputs nos permiten mapear las teclas que se presionan (teclado/ratón) ó botones de algún control externo de USB.

HACER USO DEL COMPONENTE "AXIS" QUE OSCILA DE 0 A 1 (POSITIVO) / 0 -1 (NEGATIVO).

Input Manager Axis: Permite crear varios inputs, los cuales pueden tener 1 o 2 teclas a usar, y el uso de estas teclas no necesitan estar dentro de un IF para poder usarlas, además permite usar el "Axis", el cual oscila con un valor de 0 a 1.
Edit>Project Setting>Input.

Ejemplo:

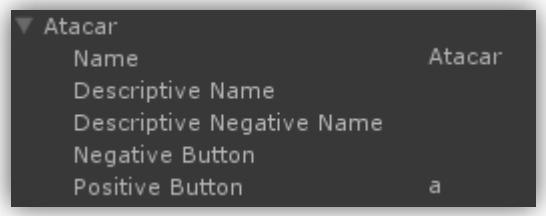
```
void Update () {
      // Mueve al objeto en horizontal/vertical del axis del inputs.
    transform.Translate (Input.GetAxis ("Horizontal"), 0, Input.GetAxis ("Vertical"));
    print (Input.GetAxis ("Horizontal"));  // Mostrar el valor.
    print (Input.GetAxis ("Vertical"));
}
```



DOBLE FUNCIÓN EN UNA MISMA TECLA (BUTTON- Infinito / DOWN / UP):

Input Manager GetKey: esto nos permite detectar alguna "tecla asignada dentro del el Input Manager", usando las existentes o crear nosotros nuevas funciones.

Ejemplo:




```
void Update () {
      if (Input. GetKeyDown/Up ("Atacar")){
            print ("El GameObject la acción!");
      }
}
```

RETARDO AL PRESIONAR UNA TECLA, PARA QUE NO SEA CONSTANTE.

Input Delay: Inputs para controlar el tiempo de esperar para ejecutar la misma función:

Ejemplo:

```
public float Esperar = 0.0f;     // Al iniciar el juego automáticamente se puede disparar.
void Update () {
      if (Input.GetKeyDown ("Fire1") && Time.time > Esperar) {
            print ("Disparo Ejecutado");
            Esperar = Time.time + 2;     // hacer que espere 2 seg para volver a disparar.
      }
}
```

FUNCIONES DE TECLADO (KEYCODE):

Ejemplo:

```
void Update () {
	if ( Input.GetKeyDown (KeyCode.Space) ){
		//acción
	}
}
```

```
void Update () {
	if ( Input.GetKeyUp (KeyCode.Space) ){
		//acción
}
```

FUNCIONES DE TECLADO (STRING):

Ejemplos:

```
void Update () {
	if ( Input.GetKeyDown ("space") ){
		//acción
	}
}
```

```
void Update () {
	if ( Input.GetKeyUp ("space") ){
		//acción
	}
}
```

FUNCIONES DE MOUSE (KEYCODE):

Ejemplos:

Mouse 0 = Izquierdo, Mouse 1 = Derecho, Mouse 2 = Central.

```
void Update () {
	if ( Input.GetKeyDown (KeyCode.Mouse0) ){
		//acción
	}
}
```

```
void Update () {
	if ( Input.GetKeyUp (KeyCode.Mouse0) ){
		//acción
	}
```

FUNCIONES DE MOUSE (STRING):

Ejemplo:
Mouse 0 = Izquierdo, Mouse 1 = Derecho, Mouse 2 = Central.

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

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

Inputs Controller (Java Script):

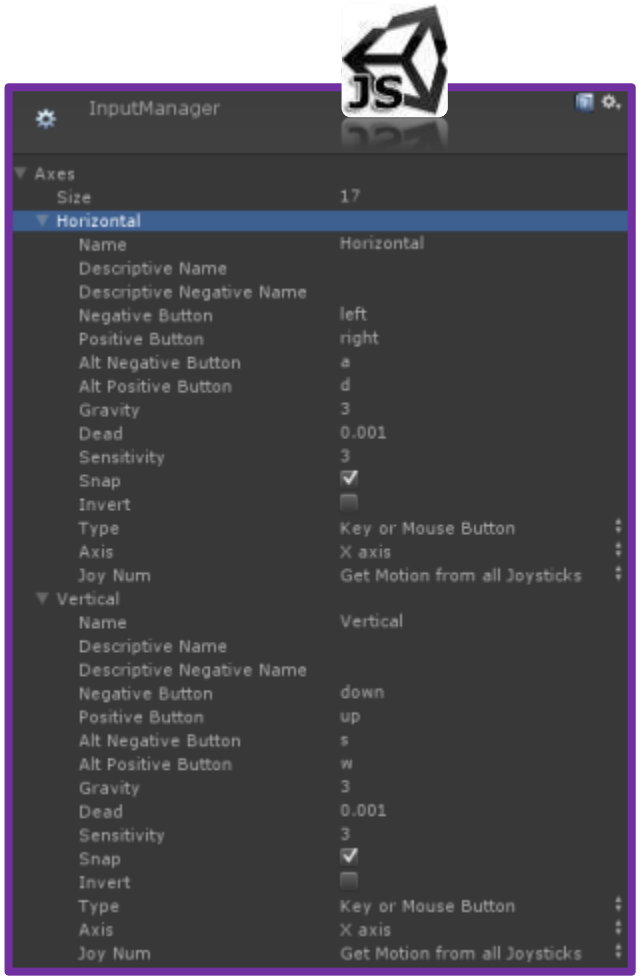
✓ Los inputs nos permiten mapear las teclas que se presionan (teclado/ratón) ó botones de algún control externo de USB.

HACER USO DEL COMPONENTE "AXIS" QUE OSCILA DE 0 A 1 (POSITIVO) / 0 -1 (NEGATIVO).

Input Manager Axis: Permite crear varios inputs, los cuales pueden tener 1 o 2 teclas a usar, y el uso de estas teclas no necesitan estar dentro de un IF para poder usarlas, además permite usar el "Axis", el cual oscila con un valor de 0 a 1.
Edit>Project Setting>Input.

Ejemplo:

```
function Update () {
      // Mueve al objeto en horizontal/vertical del axis del inputs.
    transform.Translate (Input.GetAxis ("Horizontal"), 0, Input.GetAxis ("Vertical"));
    print (Input.GetAxis ("Horizontal"));  // Mostrar el valor.
    print (Input.GetAxis ("Vertical"));
}
```

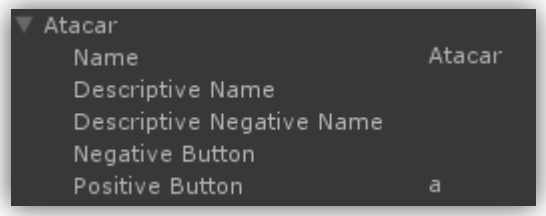


DOBLE FUNCIÓN EN UNA MISMA TECLA (BUTTON- Infinito / DOWN / UP):

Input Manager GetKey: esto nos permite detectar alguna "tecla asignada dentro del el Input Manager", usando las existentes o crear nosotros nuevas funciones.

Ejemplo:

```
function Update () {
      if (Input. GetKeyDown/Up ("Atacar")){
            print ("El GameObject la acción!");
      }
}
```



RETARDO AL PRESIONAR UNA TECLA, PARA QUE NO SEA CONSTANTE.

Input Delay: Inputs para controlar el tiempo de esperar para ejecutar la misma función:

Ejemplo:

```
var Esperar : float = 0.0; // Al iniciar el juego automáticamente se puede disparar.
function Update () {
      if (Input.GetKeyDown ("Fire1") && Time.time > Esperar) {
            print ("Disparo Ejecutado");
            Esperar = Time.time + 2; // hacer que espere 2 seg para volver a disparar
      }
}
```

FUNCIONES DE TECLADO (KEYCODE):

Ejemplo:

```
function Update () {
	if ( Input.GetKeyDown (KeyCode.Space) ){
		//acción
	}
}
```

```
function Update () {
	if ( Input.GetKeyUp (KeyCode.Space) ){
		//acción
	}
}
```

FUNCIONES DE TECLADO (STRING):

Ejemplos:

```
function Update () {
	if ( Input.GetKeyDown ("space") ){
		//acción
	}
}
```

```
function Update () {
	if ( Input.GetKeyUp ("space") ){
		//acción
	}
}
```

FUNCIONES DE MOUSE (KEYCODE):

Ejemplos:

Mouse 0 = Izquierdo, Mouse 1 = Derecho, Mouse 2 = Central.

```
function Update () {
	if ( Input.GetKeyDown (KeyCode.Mouse0) ){
		//acción
	}
}
```

```
function Update () {
	if ( Input.GetKeyUp (KeyCode.Mouse0) ){
		//acción
	}
}
```

FUNCIONES DE MOUSE (STRING):

Ejemplo:

Mouse 0 = Izquierdo, Mouse 1 = Derecho, Mouse 2 = Central.

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

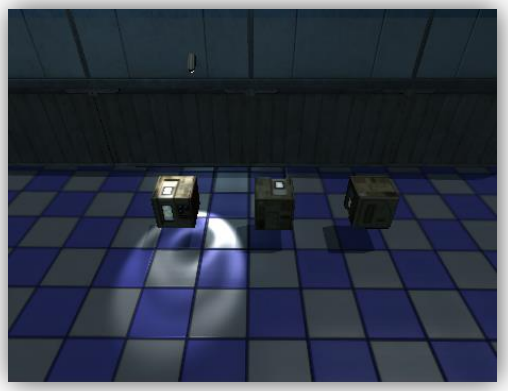
```
function Update () {
	if ( Input.GetMouseButtonUp (0) ) {
		//acción
	}
}
```

1.14 COMMUNICATION WITH GAMEOBJECTS WITH INSPECTOR + LOOKAT

Communication with GameObjects using Inspector & LookAt (C# Script):

Podemos crear variables dinámicas que se pueden tener acceso por medio del **inspector**, de esta manera se tiene control de que se conecta con cada variable de manera visual, este es un buen principio cuando se crean variables sencillas y de accesos para artistas.

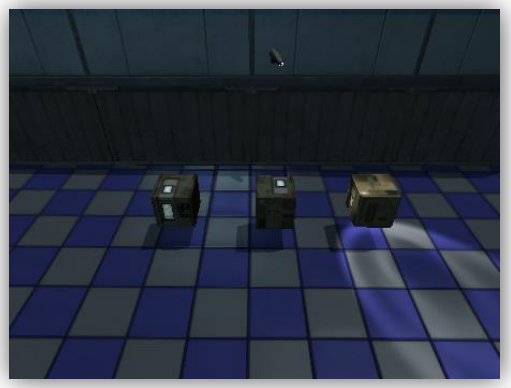
1. Cargamos el paquete llamado " Communication_Inspector".
2. Lo que haremos es al momento de reproducir la escena, cargaremos cada una de las cajas dentro de la variable que hara que se mueva la cámara hacia la caja (opción 1), en la opción 2, hacer que la cámara rote hacia la caja cargada en la variable.
3. Creamos un Script "**AI_Move**" y lo aplicamos en "**_PrefaCamera**" para poder cargar en la variable las cajas desde el inspector.



AI_Move

```
// Variable para insertar al objeto que estaremos mirando.
public Transform Box;

void Update () {
      transform.position = new Vector3
      (Box.position.x, transform.position.y, Box.position.z);
}
```

4. Cambiamos el script "**AI_Move**", y asignamos el script en "**prop_cctvCam_body_001**"



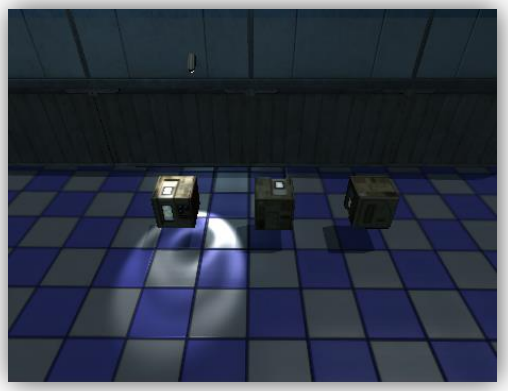
AI_LookAt

```
public Transform Box;

void Update () {
      // Función para mirar hacia un objeto de la escena.
      transform.LookAt (Box);
}
```

Communication with GameObjects using Inspector & LookAt (Java Script):

Podemos crear variables dinámicas que se pueden tener acceso por medio del **inspector**, de esta manera se tiene control de que se conecta con cada variable de manera visual, este es un buen principio cuando se crean variables sencillas y de accesos para artistas.

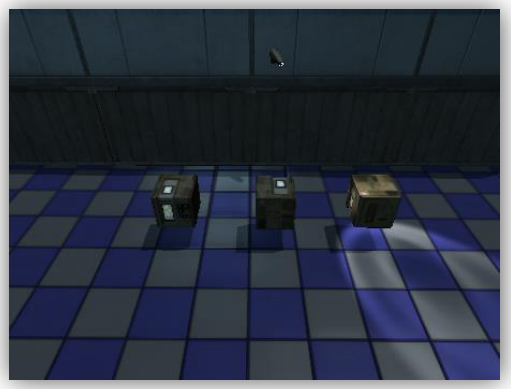
4. Cargamos el paquete llamado " Communication_Inspector".
5. Lo que haremos es al momento de reproducir la escena, cargaremos cada una de las cajas dentro de la variable que hara que se mueva la cámara hacia la caja (opción 1), en la opción 2, hacer que la cámara rote hacia la caja cargada en la variable.
6. Creamos un Script "**AI_Move**" y lo aplicamos en "**_PrefaCamera**" para poder cargar en la variable las cajas desde el inspector.



AI_Move

```
// Variable para insertar al objeto que estaremos mirando.
var Box : Transform ;

function  Update () {
	transform.position = Vector3 (Box.position.x, transform.position.y, Box.position.z);
}
```

4. Cambiamos el script "**AI_Move**", y asignamos el script en "**prop_cctvCam_body_001**"



AI_LookAt

```
var Box: Transform;

function Update () {
	// Función para mirar hacia un objeto de la escena.
	transform.LookAt (Box);
}
```

1.15 COMMUNICATION BETWEEN GAMEOBJECTS WITH FIND/TAG & BASIC ACCESS (On/Off).

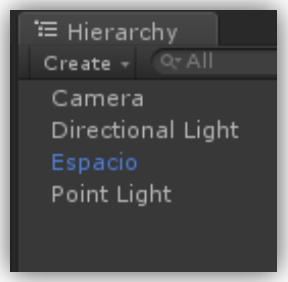
Communication Between Game Objects with Find/Tag [On/Off] (C# Script):

Podemos localizar a cualquier **GameObject** dentro de una escena rápidamente por medio de Scripting, una vez localizado se almacenará en una variable para que posteriormente podamos entrar a sus componentes y manipularlos por medio de scripting.

1. Importamos el **paquete de Escena** de la carpeta y lo insertamos en la escena colocando 2 luces, **pointlight** y un **directional** light.
2. Creamos un Script "**AI_Find**" y se aplicara a la "**Cámara**" para que este sea quien busque al **pointlight** al usar el input creado.

Script: AI_Find (1er Método → .Find)

```
//Buscar el objeto del la luz y guardarlo en la variable
GameObject ObjetoLuz;

void Start () {
	// Asignar a la variable el objeto PointLight de la escena.
	ObjetoLuz = GameObject.Find ("Point Light");
}
void Update() {
	if (Input.GetKeyDown(KeyCode.Space)) {
		//Con el objeto cargado como es luz, entramos al componente de "Light ->light"
		ObjetoLuz.light.enabled = false;
	}
}
```




```
ObjetoLuz.light.enabled = ! ObjetoLuz.light.enabled; //hacer que la luz se apague o prenda al usar la misma tecla (input).
```

Script: AI_Find (2do Método → .FindWithTag)

```
GameObject ObjetoLuz;

void Start () {
	//Mejor Performance.
	ObjetoLuz = GameObject.FindWithTag ("tag_Point Light");
}
void Update() {
	if (Input.GetKeyDown(KeyCode.Space)) {
		// Hacer que al presionar cambie el valor e imprima texto.
		if (interruptor){
			ObjetoLuz.light.enabled = ! ObjetoLuz.light.enabled;
		}
	}
}
```

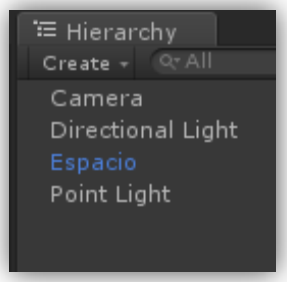
Communication Between Game Objects with Find/Tag [On/Off] (Java Script):

Podemos localizar a cualquier **GameObject** dentro de una escena rápidamente por medio de Scripting, una vez localizado se almacenará en una variable para que posteriormente podamos entrar a sus componentes y manipularlos por medio de scripting.

3. Importamos el **paquete de Escena** de la carpeta y lo insertamos en la escena colocando 2 luces, **pointlight** y un **directional** light.
4. Creamos un Script "**AI_Find**" y se aplicara a la "**Cámara**" para que este sea quien busque al **pointlight** al usar el input creado.

Script: AI_Find (1er Método → .Find)

```
//Buscar el objeto del la luz y guardarlo en la variable
private var ObjetoLuz = GameObject;

function Start () {
	ObjetoLuz = GameObject.Find ("Point Light");   // Asignar a la variable el objeto PointLight de la escena.
}

function Update() {
	if (Input.GetKeyDown(KeyCode.Space)) {
		//Con el objeto cargado como es luz, entramos al componente de "Light ->light"
		ObjetoLuz.light.enabled = false;
	}
}
```




```
ObjetoLuz.light.enabled = ! ObjetoLuz.light.enabled; //hacer que la luz se apague o prenda al usar la misma tecla (input).
```

Script: AI_Find (2do Método → .FindWithTag)

```
var ObjetoLuz GameObject;


void Start () {
     ObjetoLuz = GameObject.FindWithTag ("tag_Point Light"); //Mejor Performance.
}


function Update() {
	if (Input.GetKeyDown(KeyCode.Space)) {
		// Hacer que al presionar cambie el valor e imprima texto.
		if (interruptor){
			ObjetoLuz.light.enabled = ! ObjetoLuz.light.enabled;
		}
	}
}
```

1.16 COMMUNICATION BETWEEN GAMEOBJECTS WITH INPUTS.

Communication Between GameObjects with Inputs (C# Script):

Podemos crear comunicación entre los **GameObjects** dentro de nuestra escena, esto con el propósito de enviar de un objeto a otro información de variables y asi poder ejecutar funciones desde un objeto a otro. Esto apoyándonos del Inspector, para crear la conexión entre los GameObjects y acceder a sus Scripts.

La práctica: será hacer que la cámara de vigilancia enfoque a cada una de los barriles, pero esto será hasta el momento que nosotros presionemos cada una de las teclas **1** (Barrel_1), **2** (Barrel_2), **3** (Barrel_3) para hacer que gire en ese momento.

1. Importamos la escena"**Communication_Inputs**", crearemos 2 scripts, 1 para la "**Main Camera"** el cual controlara las teclas (1, 2, 3) y otro script para el prefab "**prop_cctvCam_body_001"** que será el que tendrá las variables de los objetos a mirar (Barriles).
2. Creamos un script "**AI_LookAt**" para "**prop_cctvCam_body_001"** y probamos "**En GamePlay**" agregar las cajas al transform:

AI_LookAt

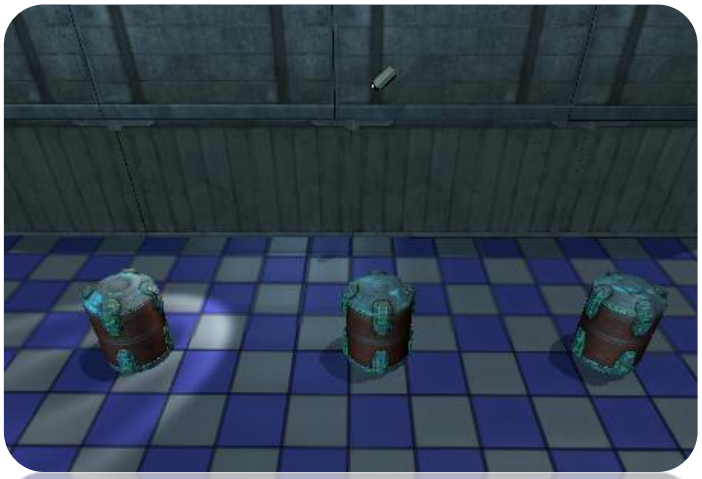
```
//Para poder crear variables publicas y estas no sean mostradas en el inspector agregamos antes de la variable.
[System.NonSerialized]
// Variable para insertar al objeto que estaremos mirando, este debe de ir publico para poder conectarnos con la variable.
public Transform Mirar;

// Creamos una función para solo ejecutarlo cuando lo necesitemos el cambiar la mira sin usar UPDATE (baja rendimiento).
void Update () {
      // Función para mirar hacia un objeto de la escena.
      transform.LookAt (Mirar);
}
```

3. Crear Script "**AI_Switcher**" y asignar a la "**Main camera",** este detectará las teclas 1, 2, 3 y conectaremos estos objetos al script "**AI_LookAt**" en la variable "**Mira**", para que se cambie automáticamente cuando le mande la información de este script al otro script.

AI_Switcher

```
// Creación de variables para guardar las cajas de la escena en cada variable.
public Transform vCaja_1;
public Transform vCaja_2;
public Transform vCaja_3;
// Este debe de contener el mismo nombre del otroscript que queremos Comunicar con el actual script.
```

```
public AI_LookAt OtroScript;

void Update() {
	if (Input.GetKeyDown (KeyCode.Alpha1) ) {
		Debug.Log ("Iluminar Barril 1!");
		//Conexión entre scripts para mandar variables y ejecutar funciones.
		OtroScript.Mirar = vCaja_1;
	}

	if (Input.GetKeyDown (KeyCode.Alpha2) ){
		Debug.Log ("Iluminar Barril 2!");
		//Conexión entre scripts para mandar variables y ejecutar funciones.
		OtroScript.Mirar = vCaja_2;
	}

	if (Input.GetKeyDown (KeyCode.Alpha3) ) {
		Debug.Log ("Iluminar Barril 3!");
		//Conexión entre scripts para mandar variables y ejecutar funciones.
		OtroScript.Mirar = vCaja_3;
	}
}
```



4. En el inspector en la "**Main Camera**" en el componente del script cargamos cada una de los barriles en cada una de las variables.
5. En el inspector en la "**Main Camera** seleccionada, en el componente del script cargamos la variable de **OtroScrip (Prefab Camara)**.

AI_Switcher


```
void Update() {

	if (Input.GetKeyDown (KeyCode.Alpha1) ) {
		Debug.Log ("Iluminar Barril 1!");
		//Conexión entre scripts para mandar variables y ejecutar funciones.
		OtroScript.Mira = vCaja_1;
		OtroScript.Enfocar();     Nota: Esto se repite en el if de caja 2 y 3.
	}
}
```

6. **PERFORMANCE**: En "**AI_LookAt**" Cambiamos **Update** por "**Enfocar ()**" y dejamos publicola función **"public void Enfocar(){}"** de esta manera ahora el buscar que caja iluminara no lo hara constantemente, si no solo hasta presionar las teclas "1, 2 o 3".

Communication Between GameObjects with Inputs (Java Script):

Podemos crear comunicación entre los **GameObjects** dentro de nuestra escena, esto con el propósito de enviar de un objeto a otro información de variables y asi poder ejecutar funciones desde un objeto a otro. Esto apoyándonos del Inspector, para crear la conexión entre los GameObjects y acceder a sus Scripts.

La práctica: será hacer que la cámara de vigilancia enfoque a cada una de los barriles, pero esto será hasta el momento que nosotros presionemos cada una de las teclas **1** (Barrel_1), **2** (Barrel_2), **3** (Barrel_3) para hacer que gire en ese momento.

1. Importamos la escena"**Communication_Inputs**", crearemos 2 scripts, 1 para la "**Main Camera"** el cual controlara las teclas (1, 2, 3) y otro script para el prefab "**prop_cctvCam_body_001"** que será el que tendrá las variables de los objetos a mirar (Barriles).
2. Creamos un script "**AI_LookAt**" para "**prop_cctvCam_body_001"** y probamos "**En GamePlay**" agregar las cajas al transform:

Script: AI_LookAt

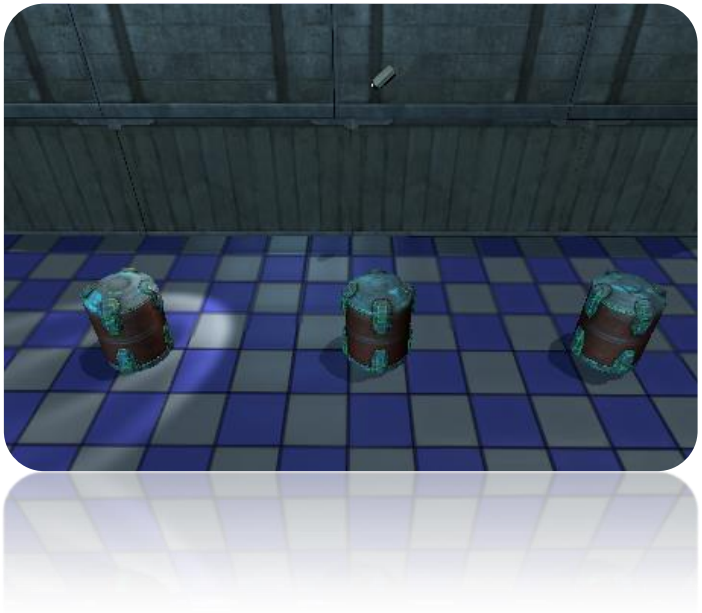
```
// Variable para insertar al objeto que estaremos mirando.
var Mirar : Transform;

// Creamos una función para solo ejecutarlo cuando lo necesitemos el cambiar la mira sin usar UPDATE (baja rendimiento).
function Update () {
	// Función para mirar hacia un objeto de la escena.
	transform.LookAt (Mirar);
}
```

3. Crear Script "**AI_Switcher**" y asignar a la "**Main camera",** este detectará las teclas 1, 2, 3 y conectaremos estos objetos al script "**AI_LookAt**" en la variable "**Mira**", para que se cambie automáticamente cuando le mande la información de este script al otro script.

Script: AI_Switcher

```
// Creación de variables para guardar las cajas de la escena en cada variable.
var vCaja_1 : Transform;
var vCaja_2 : Transform;
var vCaja_3 : Transform;
// Este debe de contener el mismo nombre del otroscript que queremos Comunicar con el actual script.
var OtroScript : AI_LookAt;
```

```
function Update() {
        if (Input.GetKeyDown (KeyCode.Alpha1)  ) {
              Debug.Log ("Iluminar Barril 1!");
              //Conexión entre scripts para mandar variables y ejecutar funciones.
              OtroScript.Mirar = vCaja_1;
       }

       if (Input.GetKeyDown (KeyCode.Alpha2) ){
              Debug.Log ("Iluminar Barril 2!");
              //Conexión entre scripts para mandar variables y ejecutar funciones.
              OtroScript.Mirar = vCaja_2;
       }

       if (Input.GetKeyDown (KeyCode.Alpha3) ) {
              Debug.Log ("Iluminar Barril 3!");
              //Conexión entre scripts para mandar variables y ejecutar funciones.
              OtroScript.Mirar = vCaja_3;
       }
}
```



4. En el inspector en la "**Main Camera**" en el componente del script cargamos cada una de los barriles en cada una de las variables.
5. En el inspector en la "**Main Camera** seleccionada, en el componente del script cargamos la variable de **OtroScrip (Prefab Camara)**.

Script: AI_Switcher

```
function Update() {

       if (Input.GetKeyDown (KeyCode.Alpha1) ){
              Debug.Log ("Iluminar Barril 1!");
              //Conexión entre scripts para mandar variables y ejecutar funciones.
              OtroScript.Mira = vCaja_1;
              OtroScript.Enfocar();     Nota: Esto se repite en el if de caja 2 y 3.
       }
}
```

6. **PERFORMANCE**: En "**AI_LookAt**" Cambiamos **Update** por "**Enfocar ()**" y dejamos publicola función **"public function Enfocar(){}"** de esta manera ahora el buscar que caja iluminara no lo hara constantemente, si no solo hasta presionar alguna de las teclas "1, 2 o 3".

1.17 COMMUNICATION BETWEEN SCRIPTS WITH GETCOMPONENT.

Communication Between Scripts with GetComponent (C# Script).

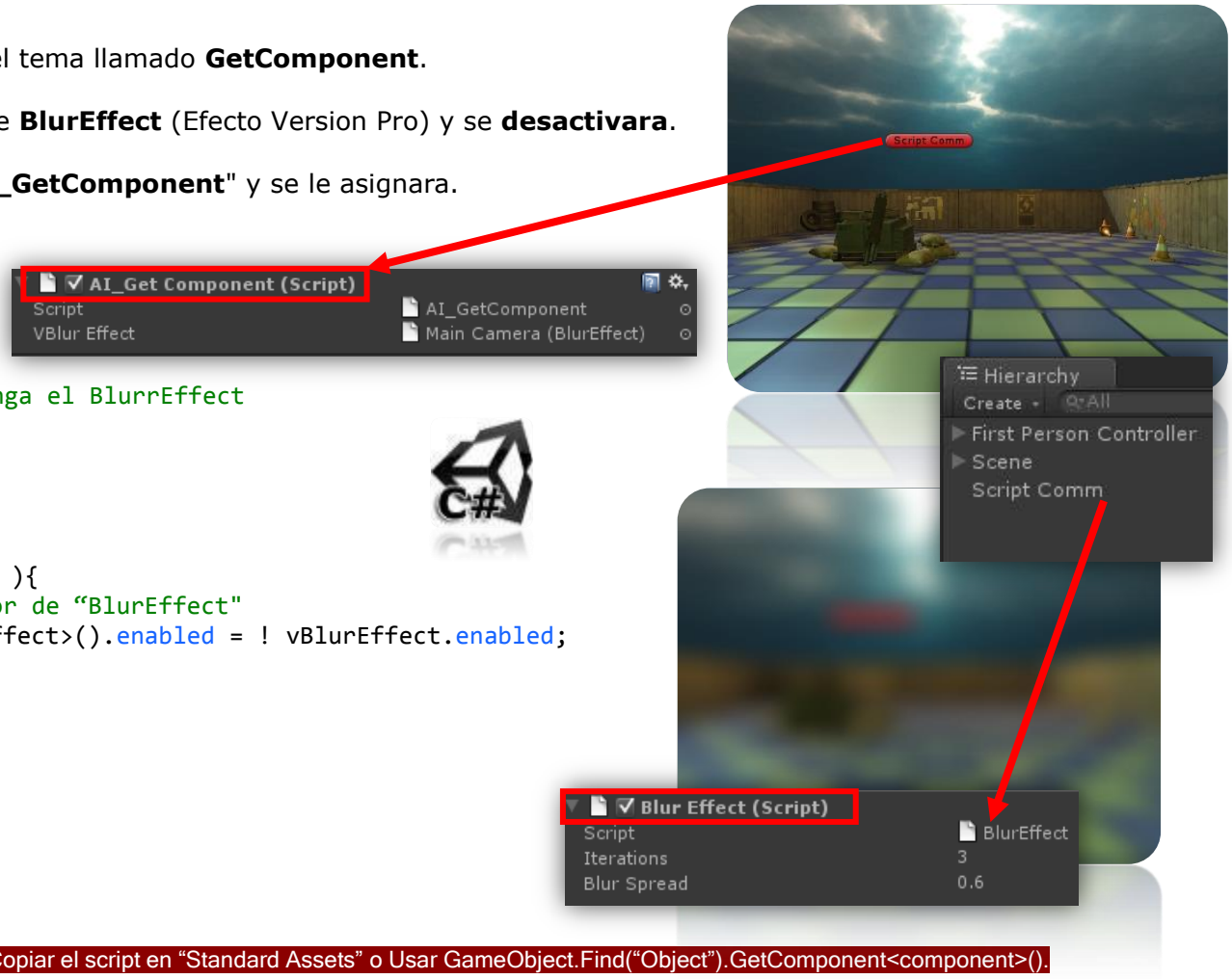
function GetComponent <type : Type>() : Component

Una vez entendido la función del **GetComponent**, procederemos a usar un efecto de cámara en este caso el de Blur, y lo activaremos por medio de Inputs, al hacer click izquierdo se active/desactive. El proceso es desde la caja del centro estará checando si se presiona el Botón Izquierdo del Mouse, y si es así va a buscar a la cámara y entra en el componente de **BlurEffect**, y lo activara o desactivara.

1. Cargamos el paquete dentro de la carpeta del tema llamado **GetComponent**.

2. Sobre la **cámara** se asigno el componente de **BlurEffect** (Efecto Version Pro) y se **desactivara**.

3. Sobre la caja central se creara un Script "**AI_GetComponent**" y se le asignara.



AI_GetComponent:

```
//Variable para  almacenar el objeto que tenga el BlurrEffect
public BlurEffect vBlurEffect;


void Update (){

      if (Input.GetKeyDown (KeyCode.Mouse0) ){
            // Activar o desactivar el valor de "BlurEffect"
            vBlurEffect.GetComponent<BlurEffect>().enabled = ! vBlurEffect.enabled;
      }
}
```

CONEXIONES ENTRE SCRIPT:

Java Script	a	Java Script	→	Si hay Conexion.
Java Script	a	C# Script	→	Si hay Conexion.
C# Script	a	C# Script	→	Si hay Conexion.
C# Script	a	Java Script	→	Si No hay Conexión: Copiar el script en "Standard Assets" o Usar GameObject.Find("Object").GetComponent<component>().

Communication Between Scripts with GetComponent (Java Script).

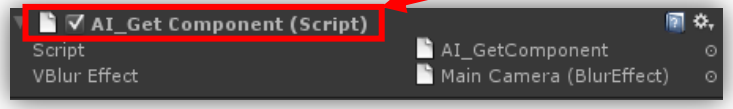
function GetComponent (type : Type) : Component

Una vez entendido la función del **GetComponent**, procederemos a usar un efecto de cámara en este caso el de Blur, y lo activaremos por medio de Inputs, al hacer click izquierdo se active/desactive. El proceso es desde la caja del centro estará checando si se presiona el Botón Izquierdo del Mouse, y si es así va a buscar a la cámara y entra en el componente de **BlurEffect**, y lo activara o desactivara.

1. Cargamos el paquete dentro de la carpeta del tema llamado **GetComponent**.

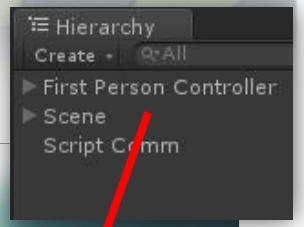
2. Sobre la **cámara** se asigno el componente de **BlurEffect** (Efecto Version Pro) y se **desactivara**.

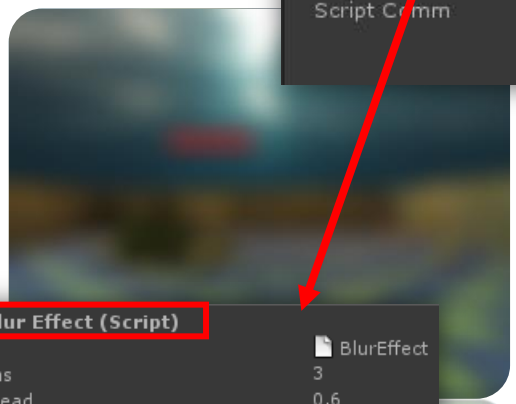
3. Sobre la caja central se creara un Script "**AI_GetComponent**" y se le asignara.



AI_GetComponent:

```
//Variable para  almacenar el objeto que tenga el BlurrEffect
var vBlurEffect : BlurEffect;


function Update (){

      if (Input.GetKeyDown (KeyCode.Mouse0) ){

            // Activar o desactivar el valor de  "BlurEffect"
            vBlurEffect.GetComponent(BlurEffect).enabled = ! vBlurEffect.enabled;
      }
}
```





CONEXIONES ENTRE SCRIPT:

Java Script	a	Java Script	→	Si hay Conexion.
Java Script	a	C# Script	→	Si hay Conexion.
C# Script	a	C# Script	→	Si hay Conexion.
C# Script	a	Java Script	→	Si No hay Conexión: Copiar el script en “Standard Assets” o Usar GameObject.Find(“Object”).GetComponent(component).

1.18 INSTANTIED & YIELD.

Instantiate & Yield (C# Script):

Instantiate (**original**: Object, **position**: Vector3, **rotation**: Quaternion): **Object** → Crear Instancias en tiempo real del Project.

1. Creamos una escena con un plano y un "**Empty Group**", del cual crearemos más objetos.
2. Creamos una Esfera, le asignamos "**RidigBody**" y la meteremos en un "**Prefab**".
3. Creamos un Script "**AI_Creator**", y se lo asignaremos al grupo vacio, eh iniciamos el scripting:

AI_Creator

```
public Rigidbody Prefab;
void Update (){
	Instantiate (Prefab, transform.position, transform.rotation);
}

//-------------------

public Rigidbody Prefab;

void Start (){
	for (int i = 0; i <= 10; I ++){
		Instantiate (Prefab, transform.position, transform.rotation);
		yield return new WaitForSeconds (0.5f);
	}
}
```

El **void** siempre se cambiara por **IEnumerator** cuando este un Yield en la función.

```
public Rigidbody Prefab;
// IEnumerator sirve para poder crear una corrutina básica por el uso del "YIELD" a parte porque esta dentro de un "FOR".
IEnumerator Start () {
	Vector3 Pos = transform.position;

	for (int i = 1; i <= 10; i++){
		Instantiate (Prefab, Vector3 (transform.position, transform.rotation);
		yield return new WaitForSeconds(0.5f);
		transform.position = new Vector3 (Pos.x+i*1.5f, Pos.y, Pos.z );

		// Imprimir la esfera creada.
		Debug.Log ("Prefab Creado :" + i);
	}
}
```

Instantiate & Yield (Java Script):

Instantiate (**original**: Object, **position**: Vector3, **rotation**: Quaternion): **Object** → Crear Instancias en tiempo real del Project.

1. Creamos una escena con un plano y un "**Empty Group**", del cual crearemos más objetos.
2. Creamos una Esfera, le asignamos "**RidigBody**" y la meteremos en un "**Prefab**".
3. Creamos un Script "**AI_Creator**", y se lo asignaremos al grupo vacio, eh iniciamos el scripting:

AI_Creator

```
var Prefab : Rigidbody;
function Update (){
	Instantiate (Prefab, transform.position, transform.rotation);
}

//-------------------

var Prefab : Rigidbody;
function Start (){
	for (var i : int = 0; i <= 10; i ++){
		Instantiate (Prefab, transform.position, transform.rotation);
		yield WaitForSeconds (0.5);
	}
}

//-------------------

var Prefab : Rigidbody;

function Start (){
	var pos : Vector3 = transform.position;

	for (var i : int = 0; I <= 10; i ++){
		Instantiate (Prefab, Vector3 (transform.position, transform.rotation);

		yield WaitForSeconds (0.5);
		transform.position = Vector3 (pos.x * 1.5, pos.y, pos.z);

		// Imprimir la esfera creada.
		Debug.Log ("Prefab Creado :" + i);
	}
}
```

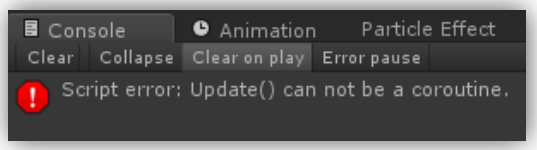
1.19 CORUTINES: YIELD, INVOKE & DEBUG.

Yield, Invokes & Debug (C# Script):

YIELD: Permite **pausar** líneas de código y cuando se termina el tiempo del **yield** pasa a la siguiente línea de código,

1. Ejemplo ----- Sin corrutina (No Funciona - No se puede ejecutar dentro de un Loop→Update una funcion con Espera/Tiempo).

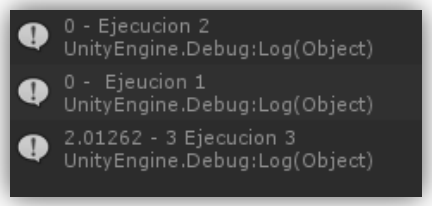
```
// Ejecucion de Yield dentro un Update, como se debe de poner en formato.

void Update  (){
       // Suspender por 2 segundos la ejecucion del codigo.
       yield return new WaitForSeconds (2.0f);
       //Imprimir el valor del tiempo.
       Debug.Log (Time.time);
}
```



IEnumerator Update () → Se debe de usar en lugar "Void" "IEnumerator" para poder crear **corrutinas siempre.**

2. Ejemplo ----- Con corrutina. - (La unica manera de ejecutar Yield dentro de funciones es que no sean de Loop→Update).

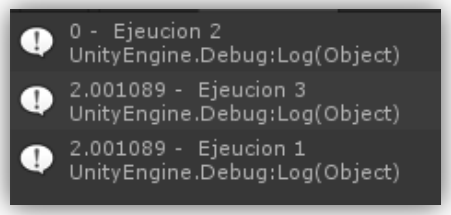
```
// Ejecucion de Funcion Sencilla detenida por tiempo para ejecutar las lineas y posteriormente el resto de la funcion.
void Start () {
       StartCoroutine ( Tiempo () );
       Debug.Log (Time.time + " - Ejeucion 1");
}

IEnumerator Tiempo () {
       Debug.Log (Time.time + " - Ejecucion 2");
       yield return new WaitForSeconds (2.0f);
       Debug.Log (Time.time + " -  Ejecucion 3");
}
```



StartCoroutine (NombreFuncion()) → Se debe de usar para poder ejecutar una función creada que contiene Yield.

3. Ejemplo ----- Con corrutina - (Ejecucion de **Funciones** y despues **de lineas sencillas** fuera de la funcion).

```
// Ejecutar Funcion con Corrutina esto ejecutara todo el contenido de la funcion y luego linea sencilla.
IEnumerator Start () {
       yield return StartCoroutine ("Tiempo");
       Debug.Log (Time.time + " -  Ejeucion 1");
}

IEnumerator Tiempo () {
       Debug.Log (Time.time + " - Ejeucion 2");
       yield return new WaitForSeconds (2.0f);
       Debug.Log (Time.time + " -  Ejeucion 3");
       }
}
```



INVOKES: El Invoke, permite ejecutar "**Funciones"** con un tiempo de ejecucion.

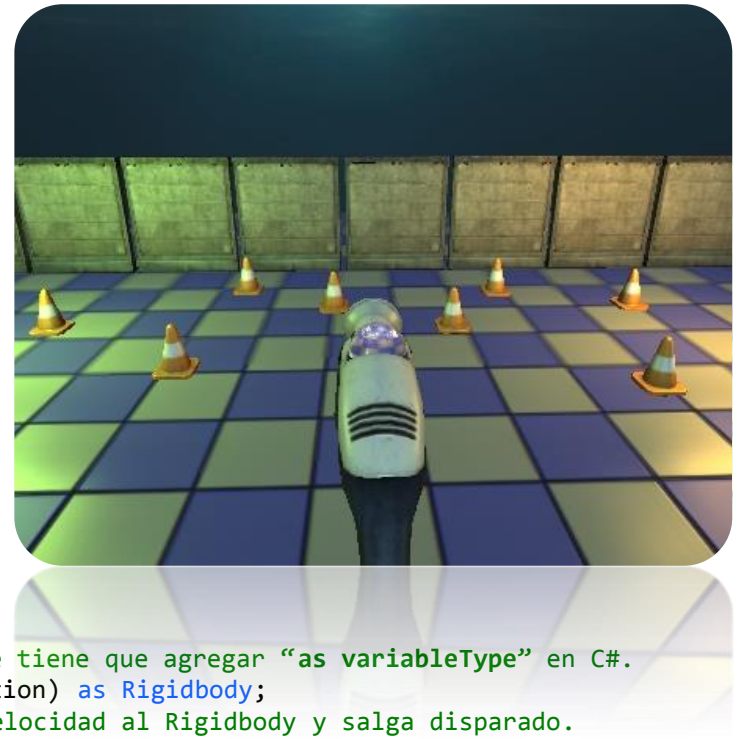
1. Ejemplo ----- Invoke **Sencillo**.

```
//invocar: (función, tiempo ejecución).
void Start (){
	Invoke ("MiFuncion", 2.0f);
}

void MiFuncion () {
	Debug.Log ("Se activo en el segundo: " + Time.time);
}
```

2. Ejemplo ----- Invoke con **Tiempo** de Ejecucion y segundos de **Repetición**.

```
//invocar repetidamente: (función, tiempo ejecución, cada X segundos).
InvokeRepeating ("ElTiempo", 0.0f, 1.0f);

void ElTiempo () {
	Debug.Log ("Se activo en el segundo: " + Time);
}
```

3. Practica ----- Invoke con Tiempo y Repeticion + **Cancelacion de Invoke**.



AI_Shooter

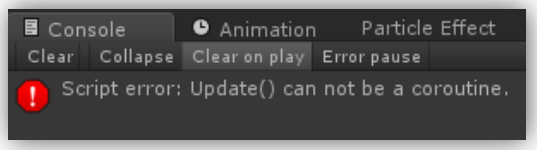
```
public Rigidbody vEsfera;
void Start (){
	InvokeRepeating("LanzarEsfera", 1.0f, 1.0f);	//Activar Invoke
}
void LanzarEsfera () {
	//si la instancia se almacenara en una "VariableType", esta al final se le tiene que agregar "as variableType" en C#.
	Rigidbody ball = Instantiate (vEsfera, transform.position, transform.rotation) as Rigidbody;
	ball.velocity = new Vector3 ( 0, 0, Random.Range (1, 20) ); //Se agrega velocidad al Rigidbody y salga disparado.
}
void Update () {
	if (Input.GetKeyDown (KeyCode.Mouse0)){
		InvokeRepeating("LanzarEsfera", 1.0f, 1.0f);		//Activar de nuevo el Invoke.
	} else if (Input.GetKeyDown (KeyCode.Mouse1)){
		CancelInvoke ();					//Cancelar Invoke.
	}
}
```

Yield, Invokes & Debug (Java Script):

YIELD: Permite **pausar** líneas de código y cuando se termina el tiempo del **yield** pasa a la siguiente línea de código,

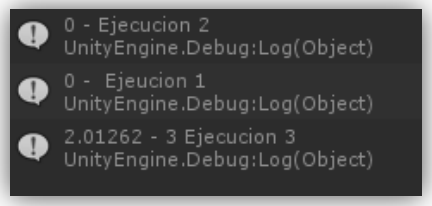
1. Ejemplo ----- Sin corrutina (No Funciona - No se puede ejecutar dentro de un Loop→Update una funcion con Espera/Tiempo).

```
// Ejecucion de Yield dentro un Update, como se debe de poner en formato.

function Update  (){
	// Suspender por 2 segundos la ejecucion del codigo.
	yield WaitForSeconds (2);
	//Imprimir el valor del tiempo.
	Debug.Log (Time.time);
}
```



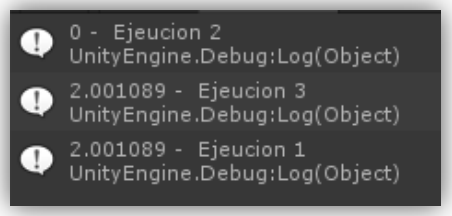
2. Ejemplo ----- Con corrutina. - (La unica manera de ejecutar Yield dentro de funciones es que no sean de Loop→Update).

```
// Ejecucion de Funcion Sencilla detenida por tiempo para ejecutar la linea sencilla y posteriormente el resto de la funcion.
function Start () {
	Tiempo ();
	Debug.Log (Time.time + " -  Ejeucion 1");
}

function Tiempo () {
	Debug.Log (Time.time + " - Ejecucion 2");
	yield WaitForSeconds (2);
	Debug.Log (Time.time + " - 3 Ejecucion 3");
}
```



3. Ejemplo ----- Con corrutina - (Ejecucion de **Funciones** y despues **de lineas sencillas** fuera de la funcion).

```
// Ejecutar Funcion con Corrutina esto ejecutara todo el contenido de la funcion y luego linea sencilla.

function Start () {
	yield  StartCoroutine ("Tiempo");
	Debug.Log (Time.time + " - Ejeucion 1");
}

function Tiempo () {
	Debug.Log (Time.time + " - Ejeucion 2");
	yield WaitForSeconds (2);
	Debug.Log (Time.time + " - Ejeucion 3");
}
```



INVOKES: El Invoke, permite ejecutar "**Funciones"** con un tiempo de ejecucion.

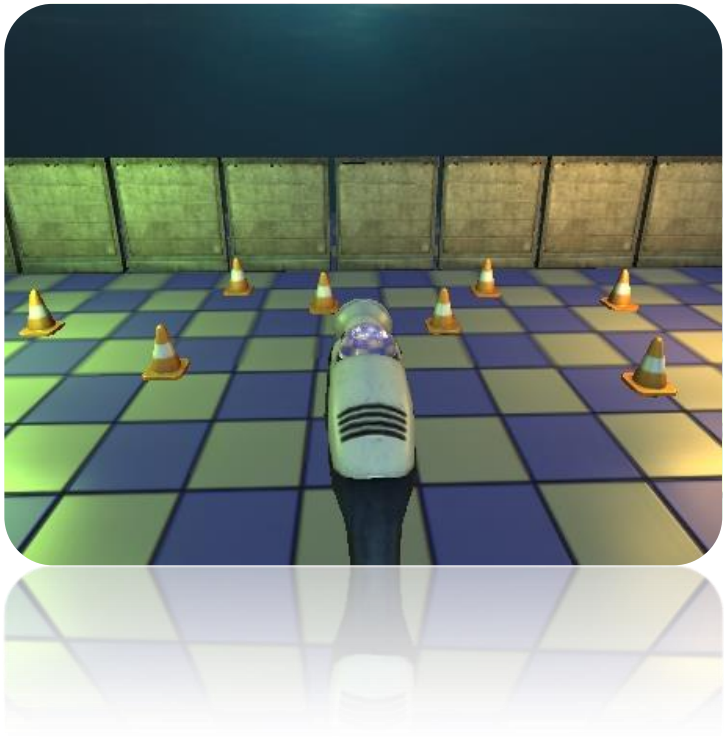
1. Ejemplo ----- Invoke **Sencillo**.

```
//invocar: (función, tiempo ejecución).
Invoke ("MiFuncion", 2);

function FuncionDeo () {
	Debug.Log ("Se activo en el segundo: " + Time.time);
}
```

2. Ejemplo ----- Invoke con **Tiempo** de Ejecucion y segundos de **Repetición**.

```
//invocar repetidamente: (función, tiempo ejecución, cada X segundos).
InvokeRepeating ("ElTiempo", 0, 1);

function ElTiempo () {
	Debug.Log ("Se activo en el segundo: " + Time.time);
}
```

3. Ejemplo ----- Invoke con Tiempo y Repeticion + **Cancelacion de Invoke**.



AI Shooter

```
var vEsfera : Rigidbody;
InvokeRepeating("LanzarEsfera", 0, 1);   //Activar Invoke

function LanzarEsfera () {
	Instantiate(vEsfera, transform.position, transform.rotation).velocity = Vector3 (0, 0, Random.Range(1, 20));
}

function Update () {
	if (Input.GetKeyDown (KeyCode.Mouse0)){
		InvokeRepeating("LanzarEsfera", 0, 1);		//Activar de nuevo el Invoke.
	} else if (Input.GetKeyDown (KeyCode.Mouse1)){
		CancelInvoke ();				//Cancelar Invoke.
	}
}
```

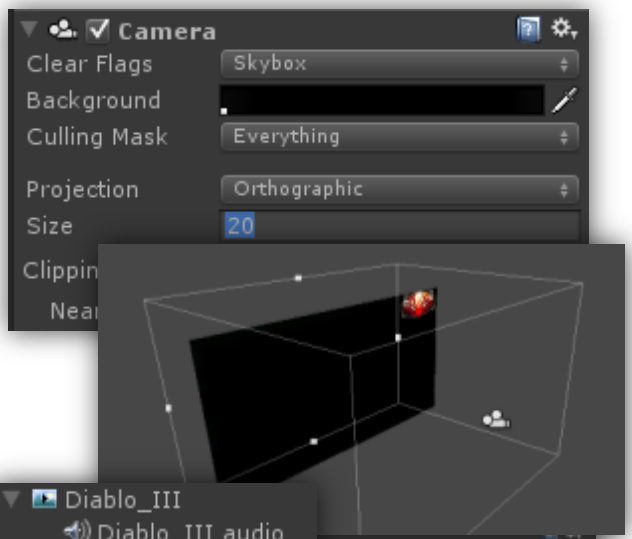
1.20 MOVIE TEXTURE_PRO

Movie Texture (C# Script):

Podemos cargar películas dentro de Unity, solamente en la versión Profesional, esto nos permite poder poner cinemáticos dentro del juego, funciona mucho para los inicios o algún tutorial en video que se requiera insertar.

Nota: Necesitamos por fuerzas tener instalado **Quicktime**, y los formatos que lee son: **.mov**, **.mpg**, **.mpeg**, **.mp4**, **.avi**, **.asf**
Lo ideal es que sea: **Formato: Quicktime Compresor: MPG4**

1. Creamos una escena con una cámara con proyección "**Ortográfica**" y creamos un plano, hacia donde mirara la cámara.
2. Importamos el Video "**Diablo_III.mp4**", este lo importara tal cual como si fuera una textura, y dentro tiene el audio del mismo video.
3. Creamos un material de tipo "**VertexLight**" y aplicamos la textura del video y se aplicará plano al plano que tenemos en la escena.
4. Sobre el plano, creamos un "**Audio Source**" y cargamos el audio del video.
5. Ahora para hacer que la textura se reproduzca al iniciar la escena creamos un Script "**AI_PlayMaterial**" y se lo asignamos al plano:

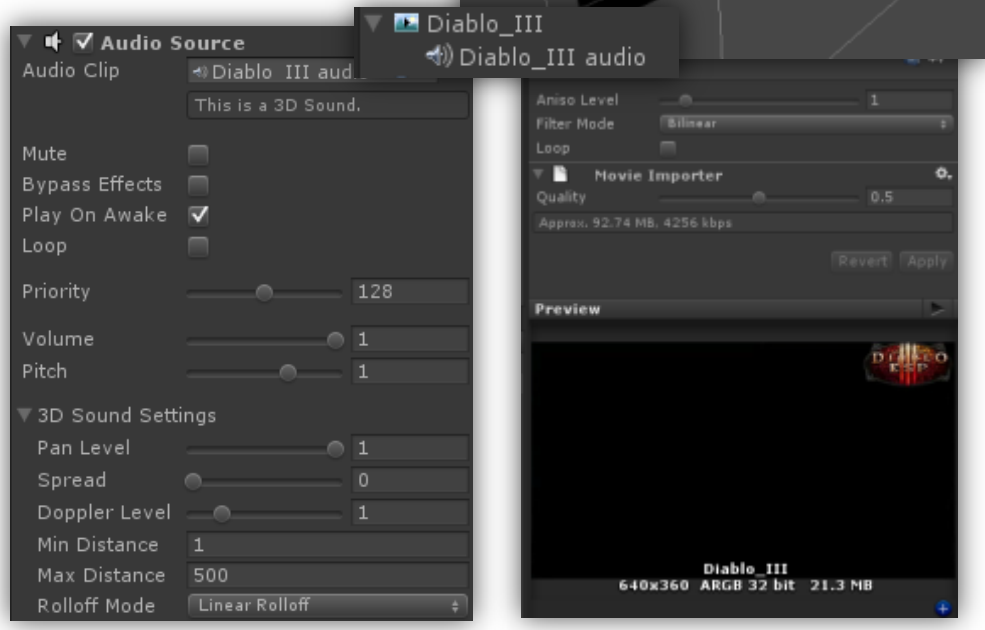


Play Movie

```
public MovieTexture Pelicula;
void Start () {
      Pelicula.Play();
}
```

Play & Pause Movie

```
void Update () {
      if (Input.GetKeyDown (KeyCode.Space)) {
            if (Pelicula.isPlaying) {
                  Pelicula.Pause();
            }
            else {
                  Pelicula.Play();
            }
      }
}
```



At the End Jump to a scene

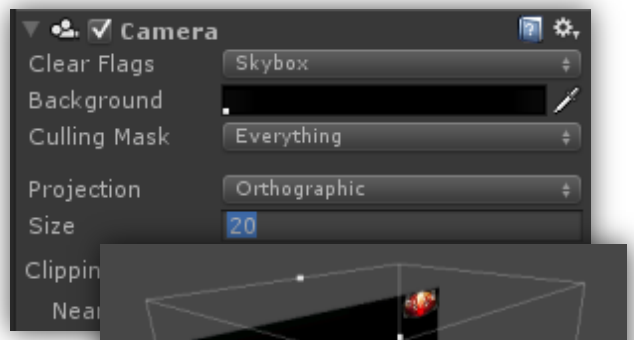
```
if (! Pelicula.isPlaying) {
      // Cargar la escena al terminar
}
```

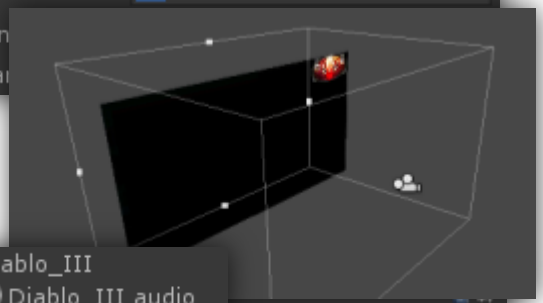
Movie Texture (Java Script):

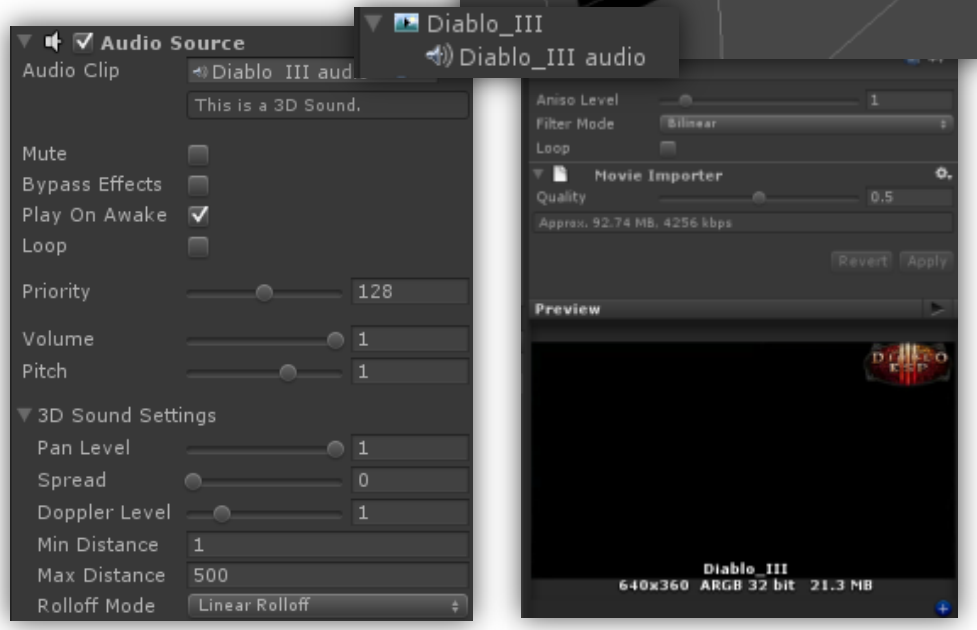
Podemos cargar películas dentro de Unity, solamente en la versión Profesional, esto nos permite poder poner cinemáticos dentro del juego, funciona mucho para los inicios o algún tutorial en video que se requiera insertar.

Nota: Necesitamos por fuerzas tener instalado **Quicktime**, y los formatos que lee son: **.mov**, **.mpg**, **.mpeg**, **.mp4**, **.avi**, **.asf**
Lo ideal es que sea: **Formato: Quicktime Compresor: MPG4**

6. Creamos una escena con una cámara con proyección "**Ortográfica**" y creamos un plano, hacia donde mirara la cámara.
7. Importamos el Video "**Diablo_III.mp4**", este lo importara tal cual como si fuera una textura, y dentro tiene el audio del mismo video.
8. Creamos un material de tipo "**VertexLight**" y aplicamos la textura del video y se aplicará plano al plano que tenemos en la escena.
9. Sobre el plano, creamos un "**Audio Source**" y cargamos el audio del video.
10. Ahora para hacer que la textura se reproduzca al iniciar la escena creamos un Script "**AI_PlayMaterial**" y se lo asignamos al plano:







Play Movie

```
Var Pelicula : MovieTexture;
function Start () {
      Pelicula.Play();
}
```

Play & Pause Movie

```
function Update () {
      if (Input.GetKeyDown (KeyCode.Space)) {
            if (Pelicula.isPlaying) {
                  Pelicula.Pause();
            }
            else {
                  Pelicula.Play();
            }
      }
}
```

At the End Jump to a scene

```
if (! Pelicula.isPlaying) {
      // Cargar la escena al terminar
}
```

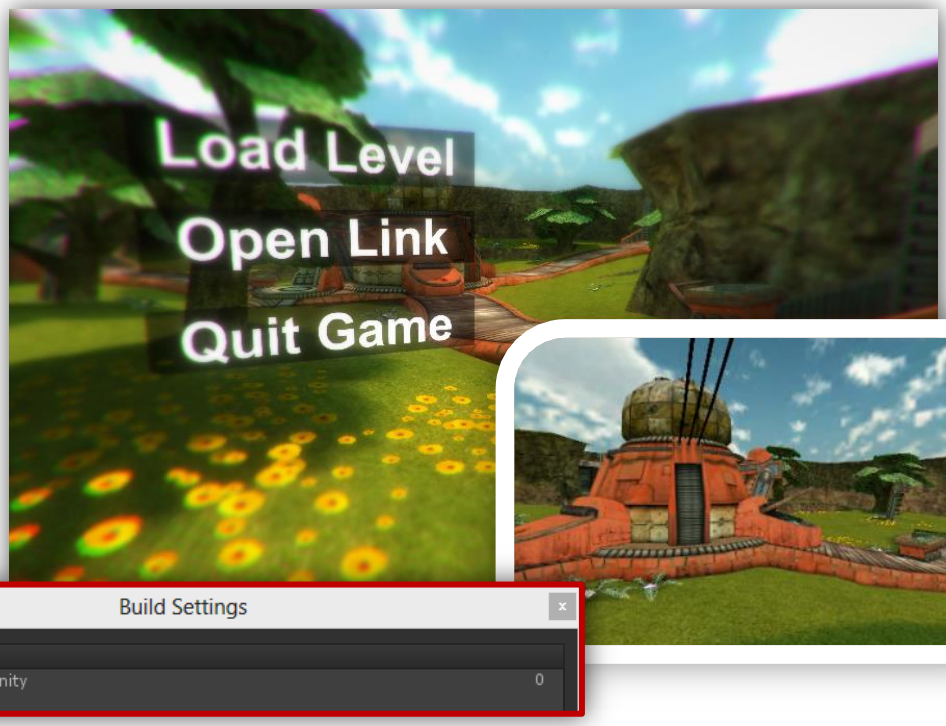
1.21 3D MENUS & LOAD SCENES

3D Menús (C# Script):

Dentro de Unity podemos crear **Menús 3D**, por medio de escenas las cuales en lugar de crear GUI Planos en 2D se sustituyen por escenas **InGame** las cuales fungen como menús interactivos más reales, como puede ser opciones, iniciar juego, créditos, borrar, crear, etc.

1. Cargamos el paquete 3D Menu y creamos un Script "**AI_Botones**" y se lo asignamos a cada uno de los botones.

AI_Botones




```
public bool Boton = false;
Vector3 Pos;

   void OnMouseEnter () {
          Boton = true;
   }

   void OnMouseExit () {
          Boton = false;
   }

   void OnMouseUp () {
          //Esta opcion varía de acuerdo al boton
          Application.LoadLevel ("_Scene" / 0);
          Application.OpenURL ("http://3deobox.com/");
          Application.Quit ();
   }
```

Nota: File>Build Settings… (Cargamos la escena).

```
   void Update () {
          Pos = transform.position;

          if (Boton) {
                 transform.position = Vector3.Lerp (transform.position, new Vector3 (Pos.x, Pos.y, 28), Time.deltaTime * 7f);
          } else {
                 transform.position = Vector3.Lerp (transform.position, new Vector3 (Pos.x, Pos.y, 25), Time.deltaTime * 7);
          }
   }
```

Si queremos que algún objeto **NO** se destruya cuando se carga otra escena:

```
DontDestroyOnLoad (transform.gameObject);
```

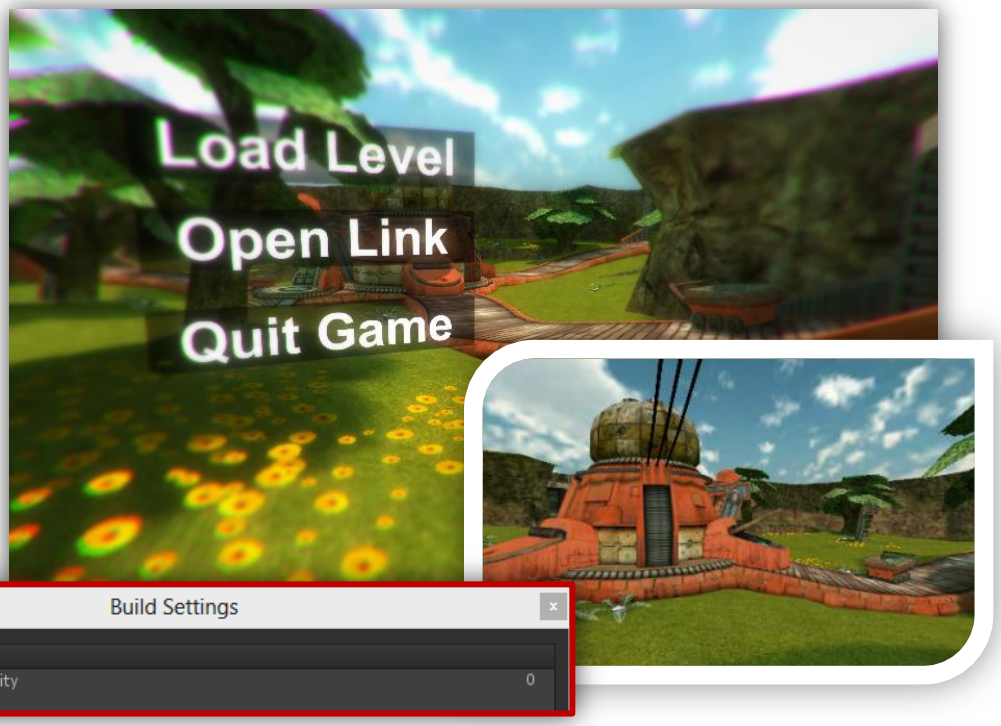
3D Menús (Java Script):

Dentro de Unity podemos crear **Menús 3D**, por medio de escenas las cuales en lugar de crear GUI Planos en 2D se sustituyen por escenas **InGame** las cuales fungen como menús interactivos más reales, como puede ser opciones, iniciar juego, créditos, borrar, crear, etc.

1. Cargamos el paquete 3D Menu y creamos un Script "**AI_Botones**" y se lo asignamos a cada uno de los botones.

AI_Botones

```
var Boton : boolean = false;
private var Pos : Vector3;

   function OnMouseEnter () {
          Boton = true;
   }

   function OnMouseExit () {
          Boton = false;
   }

   function OnMouseUp () {
          //Esta opcion varía de acuerdo al boton
          Application.LoadLevel ("_Scene" / 0);
          Application.OpenURL ("http://3deobox.com/");
          Application.Quit ();
   }
```



Nota: File>Build Settings… (Cargamos la escena).

```
   function Update () {
          Pos = transform.position;

          if (Boton) {
                 transform.position = Vector3.Lerp (transform.position, Vector3 (Pos.x, Pos.y, 26), Time.deltaTime * 7);
          } else {
                 transform.position = Vector3.Lerp (transform.position, Vector3 (Pos.x, Pos.y, 25), Time.deltaTime * 7);
          }

   }
```

Si queremos que algún objeto **NO** se destruya cuando se carga otra escena:

```
DontDestroyOnLoad (transform.gameObject);
```

1.22 PHYSICS & FORCES

Physics & Forces (C# Script):

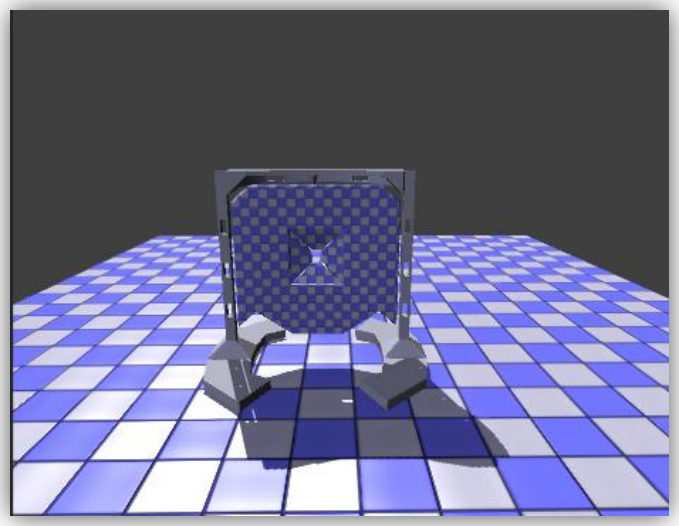
void **AddTorque** (torque : **Vector3**, mode : ForceMode = **ForceMode.Force**) : void
void **AddForce** (force : **Vector3**, mode : ForceMode = **ForceMode.Force**) : void

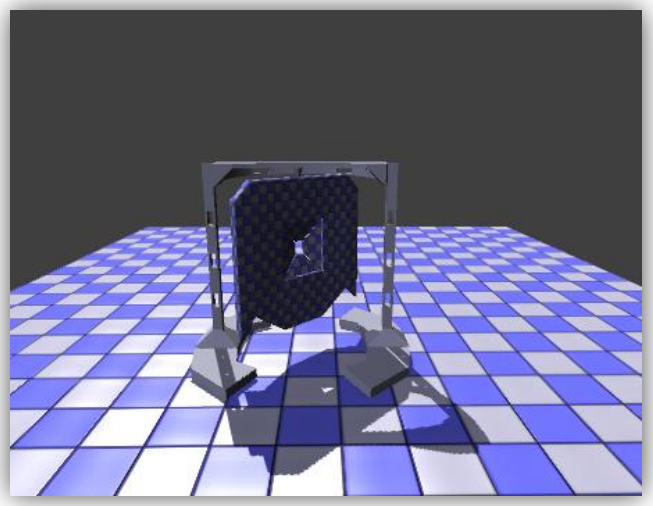
Dentro de Unity podemos crear objetos con físicas y a estos los podemos controlar por medio de fuerzas físicas.

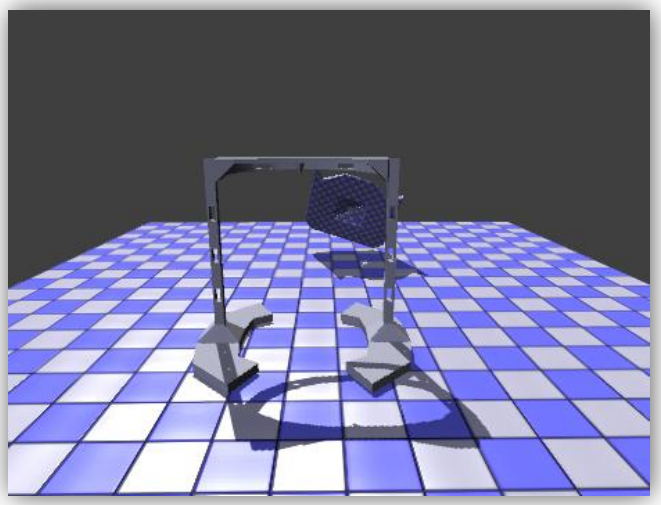
1. Cargamos el paquete Physics_Forces y creamos un Script "**AI_Door**" y se lo asignamos a la puerta.

AI_Door

```
public float TorqueForce = 50f;
public float Force = 400f;

//Funcion para obejotos con físicas.
void FixedUpdate ()
{
	float Hor = Input.GetAxis ("Horizontal") * TorqueForce * Time.deltaTime;
	rigidbody.AddTorque (transform.up * Hor, ForceMode.VelocityChange);
}

// Funcion para cuando se presiona el botón normal del mouse.
void OnMouseDown ()
{
	rigidbody.AddForce (Vector3.forward * Force);
	rigidbody.useGravity = true;
}
```







Physics & Forces (Java Script):

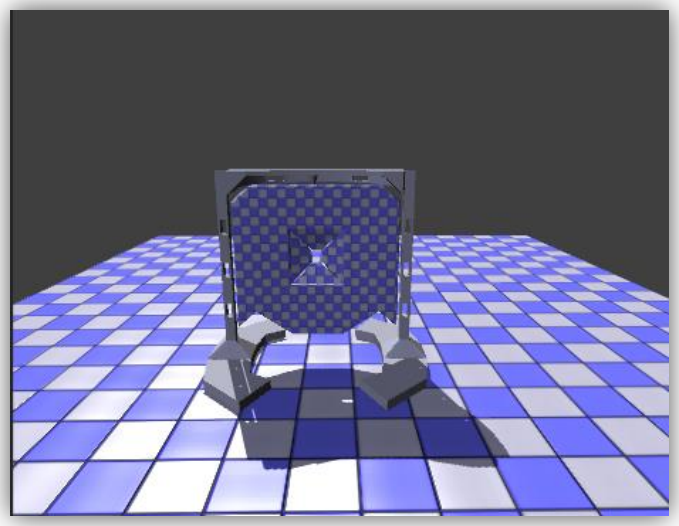
void **AddTorque** (torque : **Vector3**, mode : ForceMode = **ForceMode.Force**) : void
void **AddForce** (force : **Vector3**, mode : ForceMode = **ForceMode.Force**) : void

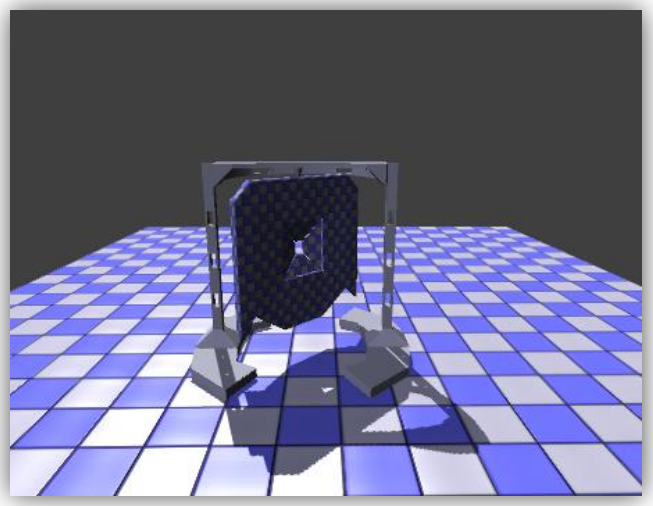
Dentro de Unity podemos crear objetos con físicas y a estos los podemos controlar por medio de fuerzas físicas.

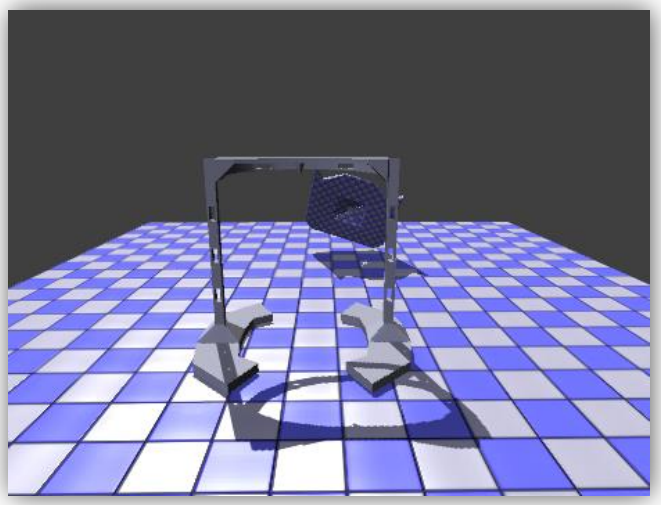
1. Cargamos el paquete Physics_Forces y creamos un Script "**AI_Door**" y se lo asignamos a la puerta.

AI_Door

```
var TorqueForce : float = 50;
var Force : float = 400;

//Funcion para obejotos con físicas.
function FixedUpdate ()
{
   var Hor : float = Input.GetAxis ("Horizontal") * TorqueForce * Time.deltaTime;
   rigidbody.AddTorque (transform.up * Hor, ForceMode.VelocityChange);
}

// Funcion para cuando se presiona el botón normal del mouse.
function OnMouseDown ()
{
      rigidbody.AddForce (Vector3.forward * Force);
      rigidbody.useGravity = true;
}
```







1.23 COLLISION & DESTROY

Collision & Destroy (C# Script):

void OnCollisionEnter (collisionInfo : Collision) : void

Podemos saber cuando un objeto con **Collider** choca con otro objeto cuando tiene el mismo componente (**Collider).**

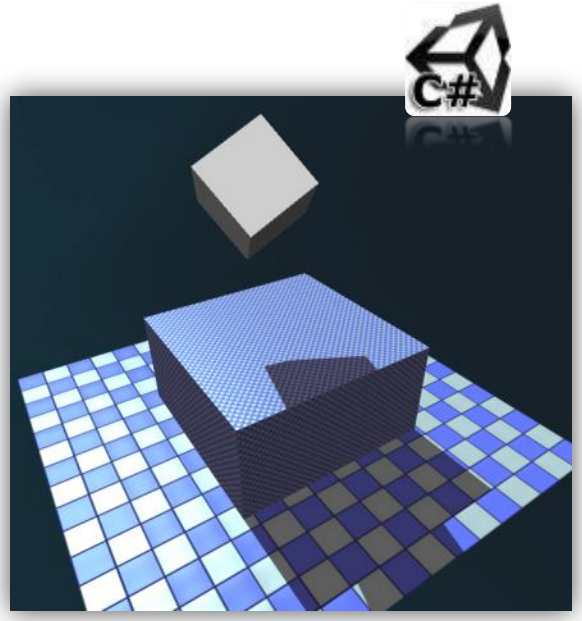
1. Creamos una escena sencilla donde tengamos un plano (piso), un muro (caja) y una caja que caiga de arriba para colisionar.
2. Seleccionamos la caja le asignamos "**Rigidbody**", y le asignamos en "**Box Collider>Material>Bouncy**" (para q rebote mas).
3. Creamos y asignamos un script a la caja "**AI_Collision**" para saber cuándo colisiona la caja con otro objeto con "**collision**".

```
void OnCollisionEnter (){
    print ("Objeto en Colisión");
}
```

4. Ahora queremos saber cuándo colisione con el piso nos indique, esto es bueno para saber cuándo colisiona con un objeto a lo mejor para activar una puerta o interruptor y cuando sea este haga alguna acción.

AI_Collision

```
//creamos una variable de tipo colisión para almacenar aquí con quien colisiona
void OnCollisionEnter (Collision vColisionando){
       if (vColisionando.gameObject.name == "Cube"){
              Destroy (vColisionando.gameObject);
       }
}
```



5. Por último paso si quisiéramos saber con qué objeto colisiona sin importar cual sea:

AI_CollisionContacts

```
void OnCollisionEnter (Collision  vColisionando){
       //variable "contactPoint" para saber x, y, z de del objeto al colisionar.
       ContactPoint Colision = vColisionando.contacts[0];

       Debug.Log ("Posicion: " + Colision.point);
       Debug.Log ("Normal: "   + Colision.normal);
       Debug.Log ("Nombre: "   + Colision.otherCollider.name);
       Debug.Log ("Prefab: "   + Colision.thisCollider.name);

}
```

Destroy:

```
Destroy (gameObject, 3.0f);  // Destruye al gameObject en 3 segundos Después.
Destroy (this);              // Remueve la instancia de script del gameObject.
Destroy (rigidbody);         // Remueve el Componente seleccionado del gameObject.
```

Collision & Destroy (Java Script):

function OnCollisionEnter (collisionInfo : Collision) : function

Podemos saber cuando un objeto con **Collider** choca con otro objeto cuando tiene el mismo componente (**Collider).**

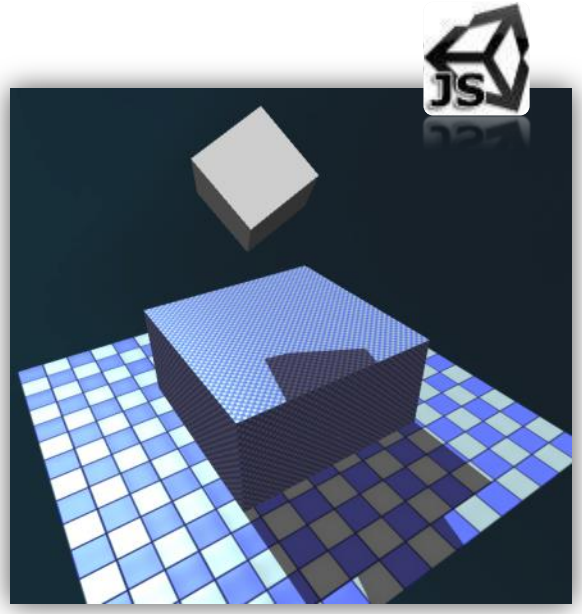
1. Creamos una escena sencilla donde tengamos un plano (piso), un muro (caja) y una caja que caiga de arriba para colisionar.
2. Seleccionamos la caja le asignamos "**Rigidbody**", y le asignamos en "**Box Collider>Material>Bouncy**" (para q rebote mas).
3. Creamos y asignamos un script a la caja "**AI_Collision**" para saber cuándo colisiona la caja con otro objeto con "**collision**".

```
function OnCollisionEnter (){
    print ("Objeto en Colisión");
}
```

4. Ahora queremos saber cuándo colisione con el piso nos indique, esto es bueno para saber cuándo colisiona con un objeto a lo mejor para activar una puerta o interruptor y cuando sea este haga alguna acción.

AI_Collision

```
//creamos una variable de tipo colisión para almacenar aquí con quien colisiona
function OnCollisionEnter (vColisionando : Collision){
       if (vColisionando.gameObject.name == "Cube"){
              Destroy (vColisionando.gameObject);
       }
}
```



5. Por último paso si quisiéramos saber con qué objeto colisiona sin importar cual sea:

AI_CollisionContacts

```
function OnCollisionEnter (vColisionando : Collision){
       //variable "contactPoint" para saber x, y, z de del objeto al colisionar.
       var Colision : ContactPoint = vColisionando.contacts[0];

       Debug.Log ("Posición: " + Colision.point);
       Debug.Log ("Normal: "   + Colision.normal);
       Debug.Log ("Nombre: "   + Colision.otherCollider.name);
       Debug.Log ("Prefab: "   + Colision.thisCollider.name);

}
```

Destroy:

```
Destroy (gameObject, 3.0f);  // Destruye al gameObject en 3 segundos Después.
Destroy (this);              // Remueve la instancia de script del gameObject.
Destroy (rigidbody);         // Remueve el Componente seleccinado del gameObject.
```

1.24 ADD COMPONENTS

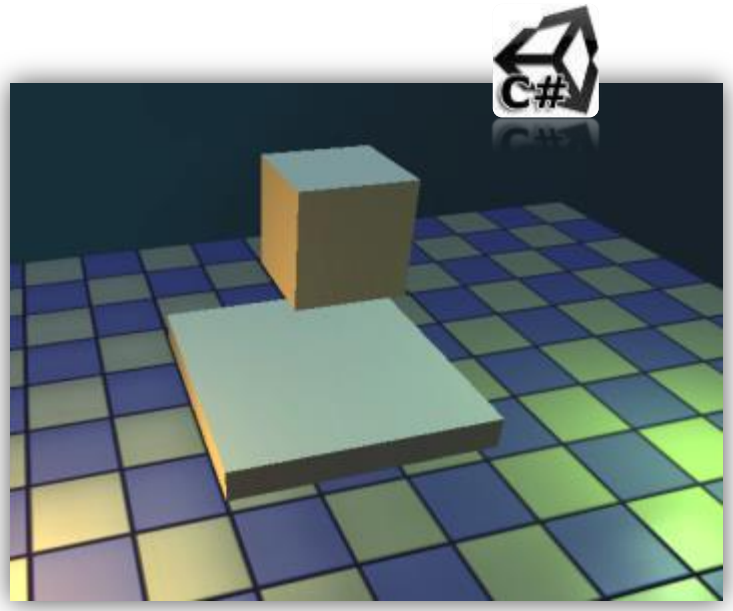
Add Components (C# Script):

void AddComponent (className : String) : Component
Podemos agregar **componentes** directamente **InGame**, por medio de **Scripting**. El ejemplo es que caerá una Caja (Rigidbody) y colisionara con otra que no tiene al iniciar la escena, pero cuando colisiona con esta le agregamos el componente de Físicas.

1. Creamos una escena con 1 plano, 2 cajas, 1 de ellas en medio (**Plataforma**) y la otra que caerá desde arriba con "**Rigidbody**".
2. Corremos la escena para ver como caer la caja de arriba sobre la plataforma, pero esta plataforma no hace nada, para moverla…
3. Creamos un Script "**AI_Rigidbody**" y se lo agregamos a la caja que caerá desde arriba por medio de las físicas.

AI_Rigidbody

```
void OnCollisionEnter (Collision vColisionar){
      if (vColisionar.gameObject.name == "Plataforma"){
            //para saber cuándo colisiona pero nada mas con la plataforma
            Debug.Log ("colisiono con la plataforma");
      }
}

void OnCollisionEnter (Collision vColisionar){
      if (vColisionar.gameObject.name == "Plataforma"){
            //para saber cuándo colisiona pero nada mas con la plataforma
            Debug.Log ("colisiono con la plataforma");
            //Agregamos Rigidbody
            vColisionar.gameObject.AddComponent<Rigidbody>();
      }
}
```



Ahora la cuestión es que cada que colisiona con la plataforma le agrega y le agrega el mismo componente ya que rebotan entre sí, para evitar esto solo hay que agregarlo solo 1 vez si ya lo tiene agregado.

```
Void OnCollisionEnter (Collision vColisionar){
      if (vColisionar.gameObject.name == "Plataforma"){
            //Si el Game Object NO tiene Rigidbody.
            if (! vColisionar.gameObject.rigidbody){
                  //Agregamos Rigidbody
                  vColisionar.gameObject.AddComponent <Rigidbody>();
            }
      }
}
```

Add Components (Java Script):

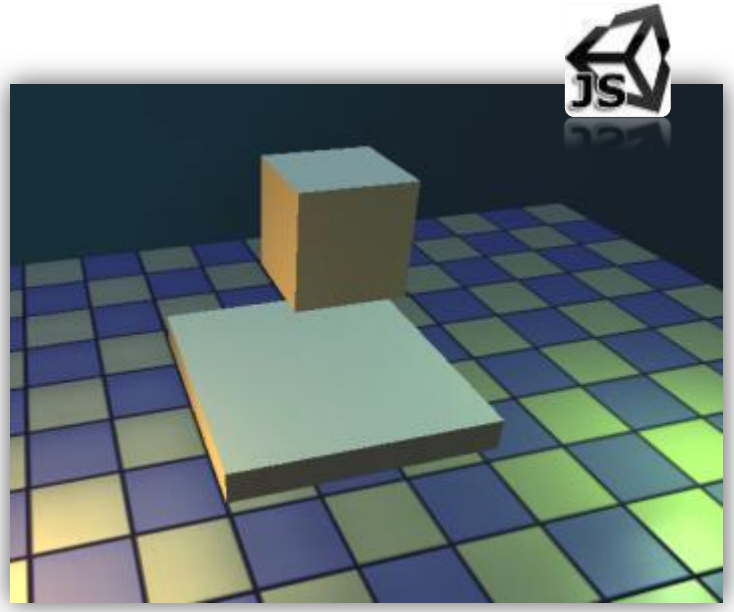
function AddComponent (className : String) : Component

Podemos agregar **componentes** directamente **InGame**, por medio de **Scripting**. El ejemplo es que caerá una Caja (Rigidbody) y colisionara con otra que no tiene al iniciar la escena, pero cuando colisiona con esta le agregamos el componente de Físicas.

1. Creamos una escena con 1 plano, 2 cajas, 1 de ellas en medio (**Plataforma**) y la otra que caerá desde arriba con "**Rigidbody**".
2. Corremos la escena para ver como caer la caja de arriba sobre la plataforma, pero esta plataforma no hace nada, para moverla…
3. Creamos un Script "**AI_Rigidbody**" y se lo agregamos a la caja que caerá desde arriba por medio de las físicas.

AI_Rigidbody

```
function OnCollisionEnter (vColisionar : Collision){
	if (vColisionar.gameObject.name == "Plataforma"){
		//para saber cuándo colisiona pero nada mas con la plataforma
		Debug.Log ("colisiono con la plataforma");
	}
}

function OnCollisionEnter (vColisionar : Collision){
	if (vColisionar.gameObject.name == "Plataforma"){
		//para saber cuándo colisiona pero nada mas con la plataforma
		Debug.Log ("colisiono con la plataforma");
		//Agregamos Rigidbody
		vColisionar.gameObject.AddComponent(Rigidbody);
	}
}
```



Ahora la cuestión es que cada que colisiona con la plataforma le agrega y le agrega el mismo componente ya que rebotan entre sí, para evitar esto solo hay que agregarlo solo 1 vez si ya lo tiene agregado.

```
function OnCollisionEnter (vColisionar: Collision){
	if (vColisionar.gameObject.name == "Plataforma"){
		//Si el Game Object NO tiene Rigidbody.
		if (! vColisionar.gameObject.rigidbody){
			//Agregamos Rigidbody
			vColisionar.gameObject.AddComponent (Rigidbody);
		}
	}
}
```

1.25 AUDIO SOURCE/CLIPS

Audio Source/Clip (C# Script):

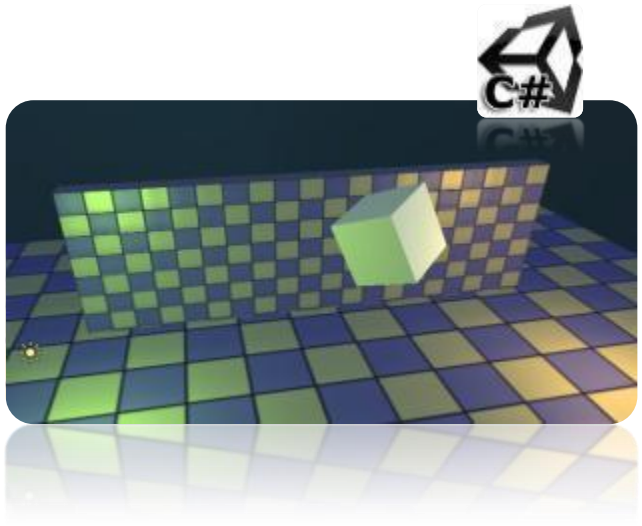
Una manera de **reproducir sonidos** es por medio de **Scripting**, veremos las variaciones para reproducir sonidos en la escena:

1. Creamos una escena con un plano, una caja con "**Rigidbody**" y le cargamos un "**Bounce Material**".
2. Importamos el sonido "**bounce**", lo arrastramos a la caja y automáticamente saldrá el "**Audio Source**">**"Play on Awake: Off"**
3. Creamos un script "**AI_Audio**" para la caja y se lo aplicamos, y haremos que cada rebote se reproduzca el audio "bounce".

Nota: siempre hay que agregar un Audio Source, para usar sin problemas los audios cargados dentro de las variables.

AI_Audio

```
void OnCollisionEnter () {
      // Reproducir el sonido que este en el Audio Source
      audio.Play();
}
```



4. Para poder cambiar los sonidos del Audio Source, usaremos el componente "**PlayOnShot**"

AI_Clips

```
//variables para cargar clips de audio.
public AudioClip MiClips;

void OnCollisionEnter (){
      //reproducir un audioclip en el Audio Source.
      audio.PlayOneShot (MiClips);
}
```

Otra manera de control del audio por medio de script:

```
// Asigna otro clip "sonido" y reproducirlo
AudioClip otherClip;

audio.clip = otherClip;
audio.Play();

// Espera a que el audio haya terminado
yield WaitForSeconds (audio.clip.length);

// Permite reproducir infinitamente el audio.
audio.loop = true/false;
```

```
// Permite silenciar el sonido mientras se reproduce.
audio.mute = true/false;

// Controla el nivel del volumen del sonido (0 a 1).
audio.volume = 0.5f;
// Permite pausar el audio.
audio.Pause();

// Detiene la reproducción del audio.
audio.Stop();
```

Audio Source/Clip (Java Script):

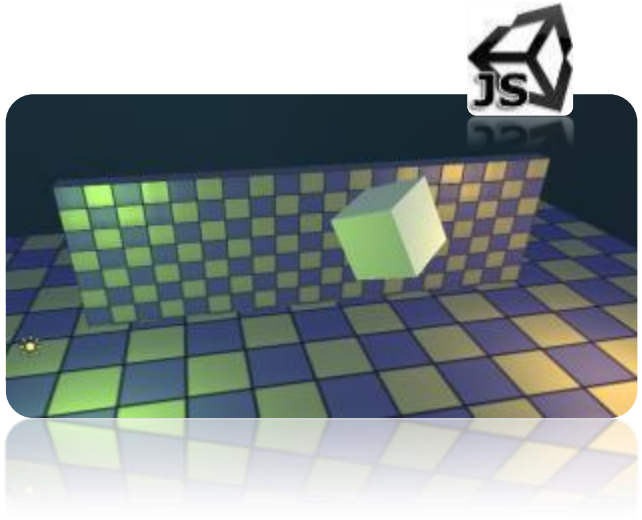
Una manera de **reproducir sonidos** es por medio de **Scripting**, veremos las variaciones para reproducir sonidos en la escena:

1. Creamos una escena con un plano, una caja con "**Rigidbody**" y le cargamos un "**Bounce Material**".
2. Importamos el sonido "**bounce**", lo arrastramos a la caja y automáticamente saldrá el "**Audio Source**">**"Play on Awake: Off"**
3. Creamos un script "**AI_Audio**" para la caja y se lo aplicamos, y haremos que cada rebote se reproduzca el audio "bounce".

Nota: siempre hay que agregar un Audio Source, para usar sin problemas los audios cargados dentro de las variables.

AI_Audio

```
function OnCollisionEnter () {
      // Reproducir el sonido que este en el Audio Source
      audio.Play();
}
```



4. Para poder cambiar los sonidos del Audio Source, usaremos el componente "**PlayOnShot**"

AI_Clips

```
//variables para cargar audios
var MiClips : AudioClip ;

function OnCollisionEnter (){
      //reproducir un audioclip en el Audio Source.
      audio.PlayOneShot(MiClips);
}
```

Otra manera de control del audio por medio de script:

```
// Asigna otro clip "sonido" y reproducirlo
var otherClip: AudioClip;

audio.clip = otherClip;
audio.Play();

// Espera a que el audio haya terminado
yield WaitForSeconds (audio.clip.length);

// Permite reproducir infinitamente el audio.
audio.loop = true/false;
```

```
// Permite silenciar el sonido mientras se reproduce.
audio.mute = true/false;

// Controla el nivel del volumen del sonido (0 a 1).
audio.volume = 0.5;
// Permite pausar el audio.
audio.Pause();

// Detiene la reproducción del audio.
audio.Stop();
```

1.26 SMOOTH VALUES & PARTICLES

Smooth Values & Particles (C# Script):

static function **Instantiate** (original : Object, position : Vector3, rotation : Quaternion) : Object
void **AddForce** (force : **Vector3**, mode : ForceMode = **ForceMode.Force**) : void

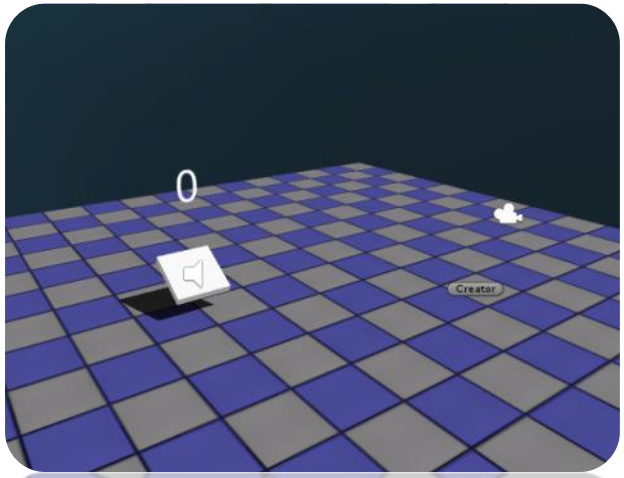
Hay una cámara animada a los lados, que al presionar el mouse soltara una bola con físicas, que al tocar la caja blanca aumente 100 puntos por cada vez que colisione y habrá un sonido al colisionar y al aumentar los puntos, estos puntos se verán en un 3D Text.

1. Cargamos el paquete "**SmoothValues_Particles**" y creamos un script "**AI_Creator**" y lo asignamos al creator de la cámara animado.
2. Asignamos **Prefab_Ball** en la variable **Ball**.

AI_Creator

```
public Rigidbody PrefabtBall;

void Update ()
{
	if (Input.GetMouseButtonDown (0) ) {
		Rigidbody RigidBall = Instantiate (PrefabtBall, transform.position, transform.rotation) as Rigidbody;
		RigidBall.AddForce (Vector3.forward * 800);
	}
}
```

3. Ahora seleccionamos el **Prefab_Ball** en "**Project**" y le creamos un script "**AI_Ball**".
4. Cargamos la **Prefab_Particle** en la variable de "**Particulas**".



AI_Ball

```
public ParticleSystem PrefabParticle;

void OnCollisionEnter ()
{
	Instantiate (PrefabParticle, transform.position, transform.rotation);
	Destroy (gameObject);
}
```

5. Ahora las partículas al terminar de emitir partículas se destruirán.

AI_Particle

```
	void Update ( ) {
		if (! particleSystem.IsAlive) {
			Destroy (gameObject);
		}
	}
```

1. Ahora Seleccionamos la caja y le creamos un script "AI_Box" en donde chocaran las Esferas que salgan con fuerza, estas al colisionar con la caja aumentara de 100 en 100 por cada colision y cambiara el valor sobre el 3D Text.
2. Cargamos el **3DMesh** dentro de la variable de "**Text3D**".

AI_Box

```
float Ponits = 0.0f;
float NextPoints = 0.0f;
float smoothTime = 100.0f;
public TextMesh Text3D;
float RoundPoints;

void Update ()
{
	//Animacion de valores.
	Ponits = Mathf.MoveTowards (Ponits, NextPoints, smoothTime * Time.deltaTime);
	RoundPoints = Mathf.Round (Ponits);
	Text3D.GetComponent <TextMesh> ().text = RoundPoints.ToString ();

	//Sonido
	if (Ponits == NextPoints) {
		audio.Stop ();
	}
}

void OnCollisionEnter ()
{
	//Puntos Extras
	NextPoints += 100;
	audio.Play ();
}
```

Smooth Values & Particles (Java Script):

static function **Instantiate** (original : Object, position : Vector3, rotation : Quaternion) : Object
void **AddForce** (force : **Vector3**, mode : ForceMode = **ForceMode.Force**) : void

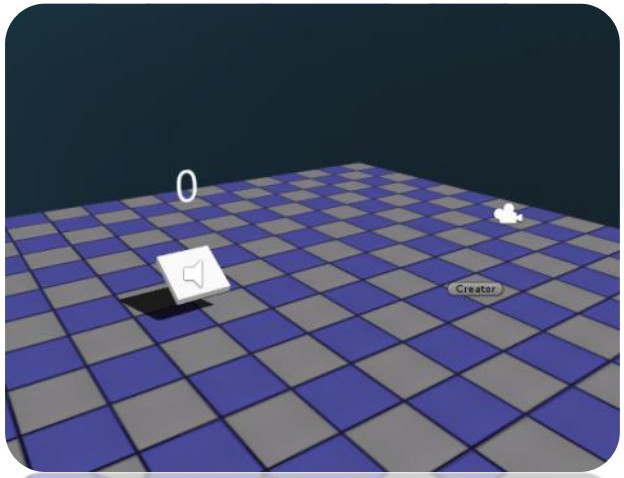
Hay una cámara animada a los lados, que al presionar el mouse soltara una bola con físicas, que al tocar la caja blanca aumente 100 puntos por cada vez que colisione y habrá un sonido al colisionar y al aumentar los puntos, estos puntos se verán en un 3D Text.

1. Cargamos el paquete "**SmoothValues_Particles**" y creamos un script "**AI_Creator**" y lo asignamos al creator de la cámara animado.
2. Asignamos **Prefab_Ball** en la variable **Ball**.

AI_Creator

```
var PrefabtBall : Rigidbody;

function Update ()
{
	if (Input.GetMouseButtonDown (0) ) {
		Rigidbody RigidBall = Instantiate (PrefabtBall, transform.position, transform.rotation);
		RigidBall.AddForce (Vector3.forward * 800);
	}
}
```

3. Ahora seleccionamos el **Prefab_Ball** en "**Project**" y le creamos un script "**AI_Ball**".
4. Cargamos la **Prefab_Particle** en la variable de "**Particulas**".



AI_Ball

```
var PrefabParticle : ParticleSystem;

function OnCollisionEnter ()
{
	Instantiate (PrefabParticle, transform.position, transform.rotation);
	Destroy (gameObject);
}
```

5. Ahora las partículas al terminar de emitir partículas se destruirán.

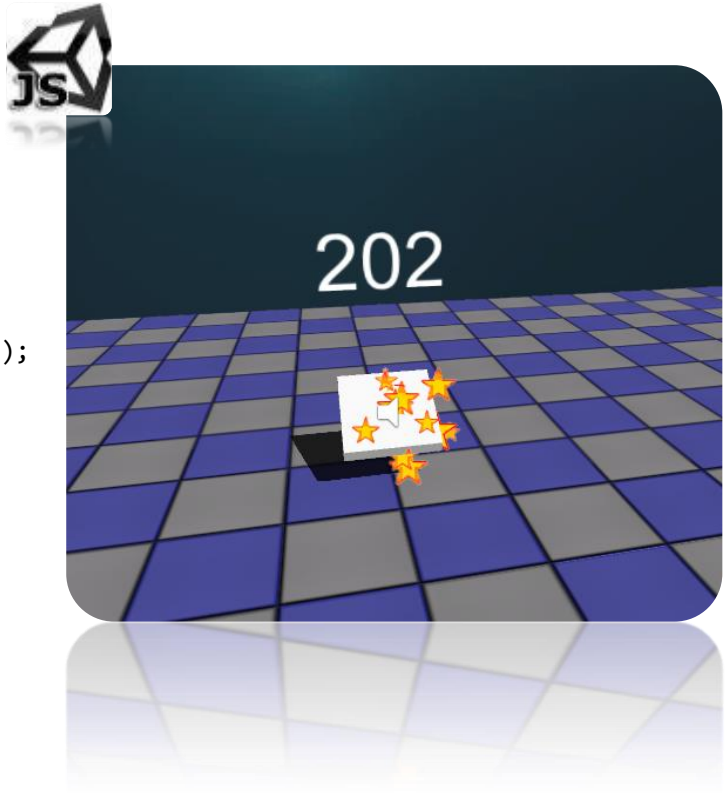
AI_Particle

```
function Update ( ) {
	if (! particleSystem.IsAlive) {
		Destroy (gameObject);
	}
}
```

3. Ahora Seleccionamos la caja y le creamos un script "AI_Box" en donde chocaran las Esferas que salgan con fuerza, estas al colisionar con la caja aumentara de 100 en 100 por cada colision y cambiara el valor sobre el 3D Text.
4. Cargamos el **3DMesh** dentro de la variable de "**Text3D**".

AI_Box

```
private var Ponits : float = 0.0f;
private var NextPoints : float = 0.0f;
private var smoothTime : float = 100.0f;
var Text3D : TextMesh;
private float RoundPoints;

function Update ()
{
	//Animacion de valores.
	Ponits = Mathf.MoveTowards (Ponits, NextPoints, smoothTime * Time.deltaTime);
	RoundPoints = Mathf.Round (Ponits);
	Text3D.GetComponent <TextMesh> ().text = RoundPoints.ToString ();

	//Sonido
	if (Ponits == NextPoints) {
		audio.Stop ();
	}
}

function OnCollisionEnter ()
{
	//Puntos Extras
	NextPoints += 100;
	audio.Play ();
}
```



1.27 TRIGGERS & ANIMATIONS

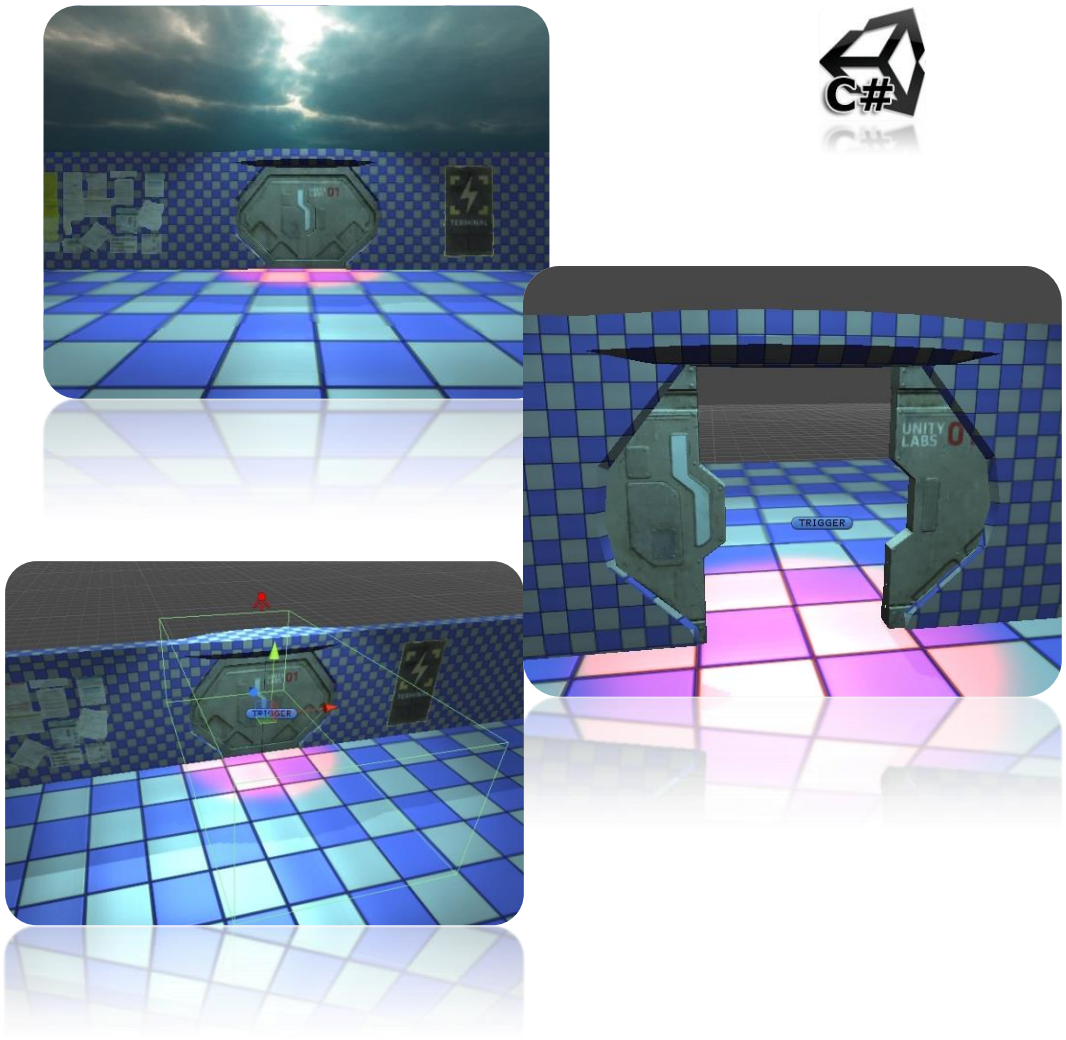
Triggers & Animations (C# Script):

Los triggers nos permiten informar cuando el jugador entra o ítem entran, sa mantienen ó salen de un área específica, dentro de unity basta con poner un objeto con Collider y dentro de sus opciones en el Inspector activamos: **Trigger**.

1. Cargamos la escena Trigger_Animations, el cual contiene una escena con una luz roja que cambiara de color cuando nos acerquemos y las puertas se abrirán por medio de una animación simple que tienen cargadas (Clamp Forever).
2. Creamos unos muros para separar cada área, y en medio ponemos "**Empty Group**" y le asignamos "**Box Collider**" (Puerta).
3. Creamos un Script "**AI_Trigger**", y lo asignamos al trigger que funcionara como cruce de las puertas.

AI_Trigger

```
public GameObject Door_Right;
public GameObject Door_Right;
public Light SpotLight;

void OnTriggerEnter () {
       //Velocidad Animaciones
       Door_Right.animation ["Clip_Door_R"].speed = 1;
       Door_Left.animation ["Clip_Door_L"].speed = 1;
       //Animaciones
       Door_Right.animation.Play ("Clip_Door_R");
       Door_Left.animation.Play ("Clip_Door_L");
       //Cambio de color en luz.
       SpotLight.color = Color.green;
}


void OnTriggerExit () {
       //Velocidad Animaciones
       Door_Right.animation ["Clip_Door_R"].speed = -1;
       Door_Left.animation ["Clip_Door_L"].speed = -1;
       //Animaciones
       Door_Right.animation.Play ("Clip_Door_R");
       Door_Left.animation.Play ("Clip_Door_L");
       //Cambio de color en luz.
       SpotLight.color = Color.red;
}
```



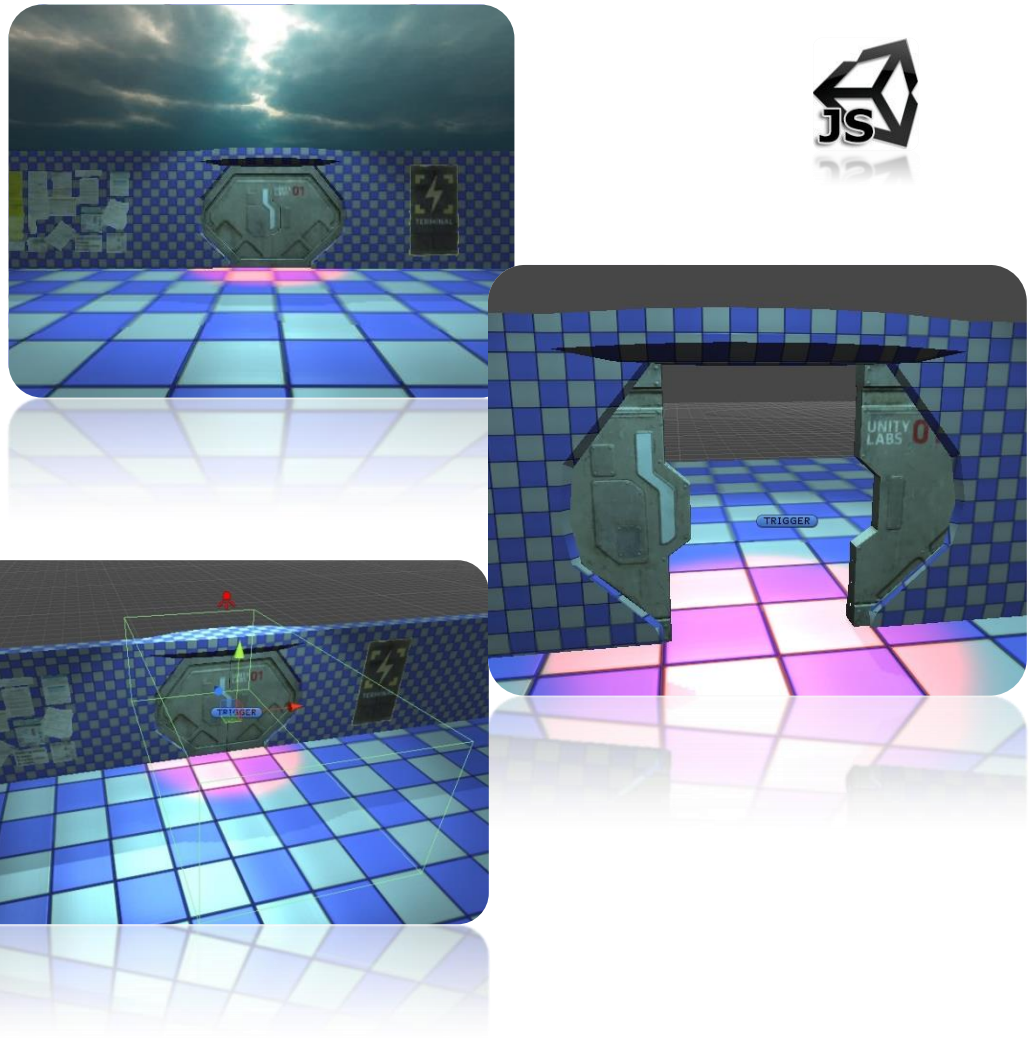
Triggers & Animations (C# Script):

Los triggers nos permiten informar cuando el jugador entra o ítem entran, sa mantienen ó salen de un área específica, dentro de unity basta con poner un objeto con Collider y dentro de sus opciones en el Inspector activamos: **Trigger**.

1. Cargamos la escena Trigger_Animations, el cual contiene una escena con una luz roja que cambiara de color cuando nos acerquemos y las puertas se abrirán por medio de una animación simple que tienen cargadas (Clamp Forever).
2. Creamos unos muros para separar cada área, y en medio ponemos "**Empty Group**" y le asignamos "**Box Collider**" (Puerta).
3. Creamos un Script "**AI_Trigger**", y lo asignamos al trigger que funcionara como cruce de las puertas.

AI_Trigger

```
var Door_Right : GameObject;
var Door_Right : GameObject;
var SpotLight : Light;

function OnTriggerEnter () {
	//Velocidad Animaciones
	Door_Right.animation ["Clip_Door_R"].speed = 1;
	Door_Left.animation ["Clip_Door_L"].speed = 1;
	//Animaciones
	Door_Right.animation.Play ("Clip_Door_R");
	Door_Left.animation.Play ("Clip_Door_L");
	//Cambio de color en luz.
	SpotLight.color = Color.green;
}


function OnTriggerExit () {
	//Velocidad Animaciones
	Door_Right.animation ["Clip_Door_R"].speed = -1;
	Door_Left.animation ["Clip_Door_L"].speed = -1;
	//Animaciones
	Door_Right.animation.Play ("Clip_Door_R");
	Door_Left.animation.Play ("Clip_Door_L");
	//Cambio de color en luz.
	SpotLight.color = Color.red;
}
```



1.28 ANIMATION & EVENTS

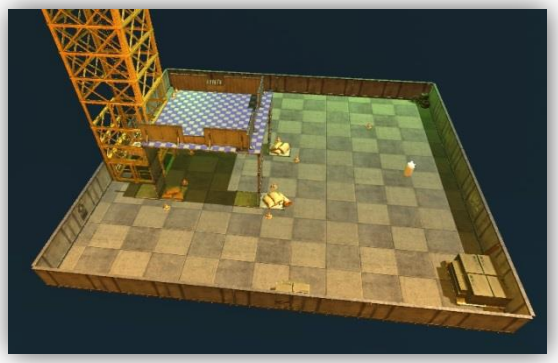
Animation & Events (C# Script):

Los **Animation events** nos permite ejecutar funciones cuando la línea del tiempo pasa por un keyframe en especifico, y cuando esta línea del tiempo llega al evento **colocado** en un **keyframe,** simplemente se ejecuta lo que tenga ese evento.

1. Cargamos la escena Animation Events, que tiene ya un carácter controller (Player) y un pregab (Elevator).
2. En el Elevator cambiamos el **Collider → Trigger** en la **plataforma** del **Elevador** para activar cuando entr solamente el Player.
3. Desactivamos "**Play Automatically**" y Creamos un Script "**AI_Elevador**" y se lo asignamos al Elevator.

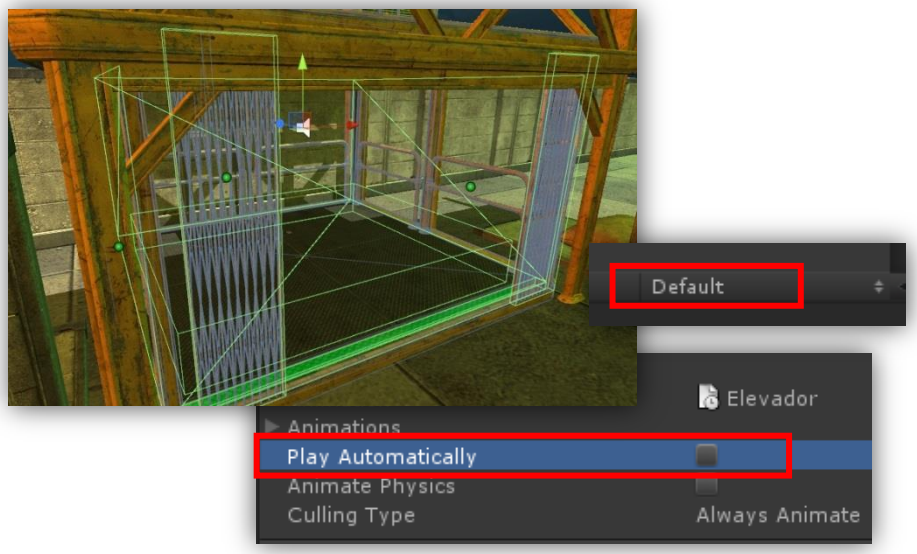
AI_Elevator

```
void OnTriggerEnter (Collider vColisionar){
	//Solamente si el Player colisiona con la plataforma ejecutar.
	if (vColisionar.gameObject.name == "Player"){
		//Activar la animación (Pause).
		animation["Elevator"].enabled = true;
		// Reproducir desde donde este la línea del tiempo
		animation.Play ("Elevator");
	}
}
```



Creamos funciones (**EVENTOS**) para cambiar de colores la plataforma e indique cuando este en movimiento.

```
//Agregamos las barras de color verde del elevador.
	public GameObject AlertColor;

	//Desactivar la animación (Pause).
	void AnimPause(){
		animation["Elevator"].enabled = false;
	}

	//Cambiar a color Rojo
	void ColorRojo(){
		AlertColor.renderer.material.color = Color.red;
	}

	//Cambiar a Color Verde
	void ColorVerde(){
		AlertColor.renderer.material.color = Color.green;
	}
```

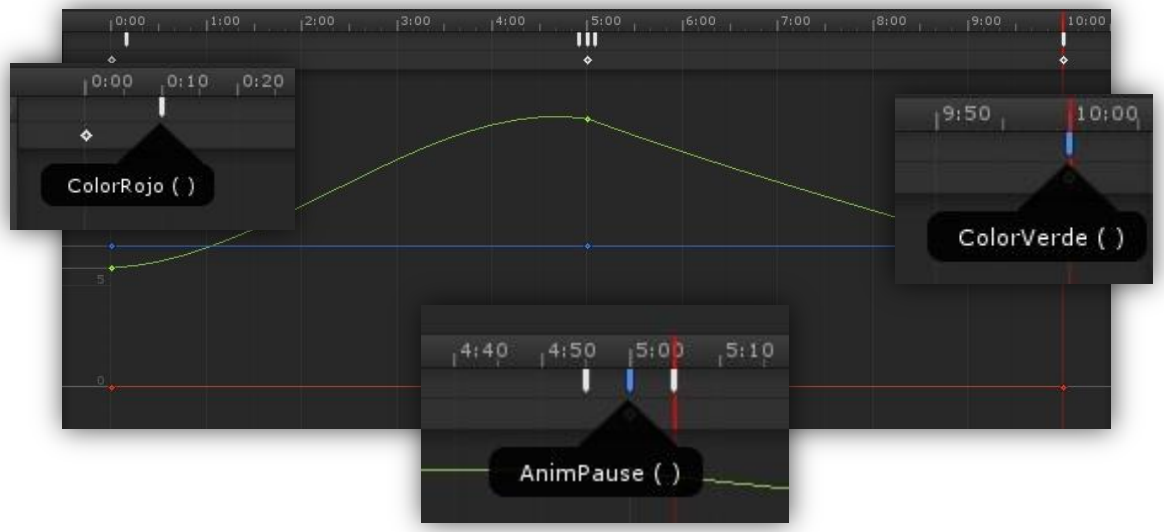


4. Ahora sobre la animación realizada, lo que haremos es agregar eventos para controlar cuando este en movimiento el elevador cambie a color rojo y en verde cuando no esté en movimiento:

- Segundo 0.01 Evento → ColorRojo().

- Segundo 2.29 Evento → ColorVerde().
- Segundo 2.30 Evento → AnimPause().
- Segundo 2.31 Evento → ColorRojo().

- Segundo 4.49 Evento → AnimPause().
- Segundo 5.00 Evento → ColorVerde().



Acciones de la función Animation:

```
//Reproducir la animación general o por animación.
 animation.Play(); / animation.Play("Elevator");


//Detener la animación general y rebobinar o por animación.
animation.Stop(); / animation.Stop("Elevator");


//Detener la animación en general o por animación.
animation.Rewind(); / animation.Rewind("Elevator ");


//Activar/Desactivar (Pause) la animación en general o por animación.
animation.enabled = true/false;  /   animation["Elevator "].enabled = true/false;

//Controla la animación Walk su velocidad = reproducción atrás (-1)/Adelante (1).
animation["Elevator "].speed = -1 ó 1;

//Esperar hasta que la animación termine de reproducir.
yield return new WaitForSeconds (animation.clip.lenght);

//Modo de reproducción de las animaciones.
animation.wrapMode = WrapMode.Loop;     →   (WrapMode.Default, .One, .Loop, .PingPong, .ClampForever).

//Reproducción de Animacion con transición (Blending) de acuerdo a tiempo (se puede tener varios tipos de animación)
animation.CrossFade("Elevator ", 0.5f)
```

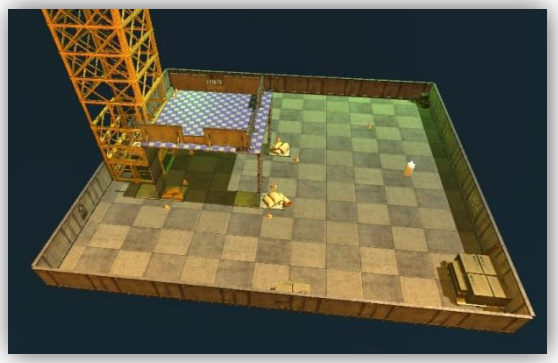
3D AnimationS & Events (Java Script):

Los **Animation events** nos permite ejecutar funciones cuando la línea del tiempo pasa por un keyframe en especifico, y cuando esta línea del tiempo llega al evento **colocado** en un **keyframe,** simplemente se ejecuta lo que tenga ese evento.

1. Cargamos la escena Animation Events, que tiene ya un carácter controller (Player) y un pregab (Elevator).
2. En el Elevator cambiamos el **Collider → Trigger** en la **plataforma** del **Elevador** para activar cuando entr solamente el Player.
3. Desactivamos "**Play Automatically**" y Creamos un Script "**AI_Elevador**" y se lo asignamos al Elevator.

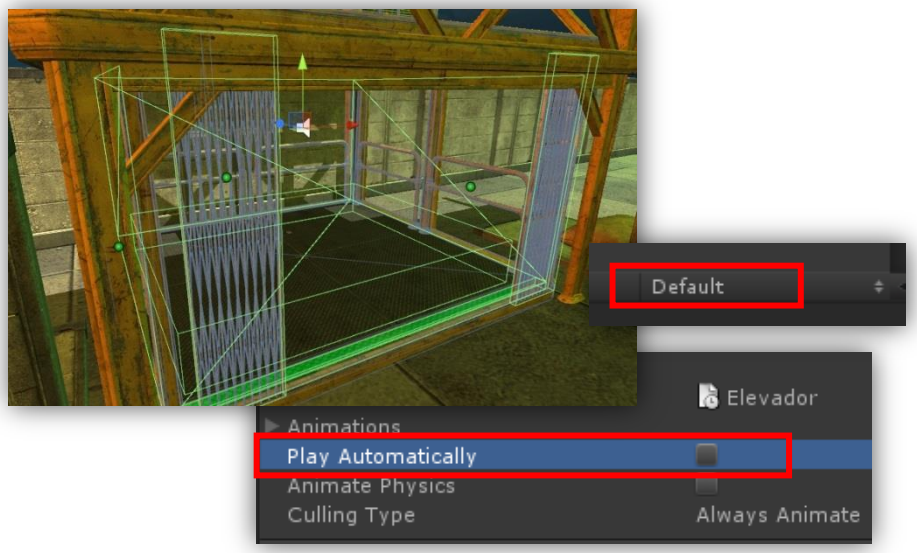
AI_Elevador

```
function OnTriggerEnter (vColisionar : Collider){
	//Solamente si el Player colisiona con la plataforma ejecutar.
	if (vColisionar.gameObject.name == "Player"){
		//Activar la animación (Pause).
		animation["Elevator"].enabled = true;
		// Reproducir desde donde este la línea del tiempo
		animation.Play ("Elevator");
	}
}
```



Creamos funciones (**EVENTOS**) para cambiar de colores la plataforma e indique cuando este en movimiento.

```
//Agregamos las barras de color verde del elevador.
var AlertColor: GameObject;

//Desactivar la animación (Pause).
function AnimPause(){
	animation["Elevator"].enabled = false;
}

//Cambiar a color Rojo
function ColorRojo(){
	AlertColor.renderer.material.color = Color.red;
}

//Cambiar a Color Verde
function ColorVerde(){
	AlertColor.renderer.material.color = Color.green;
}
```

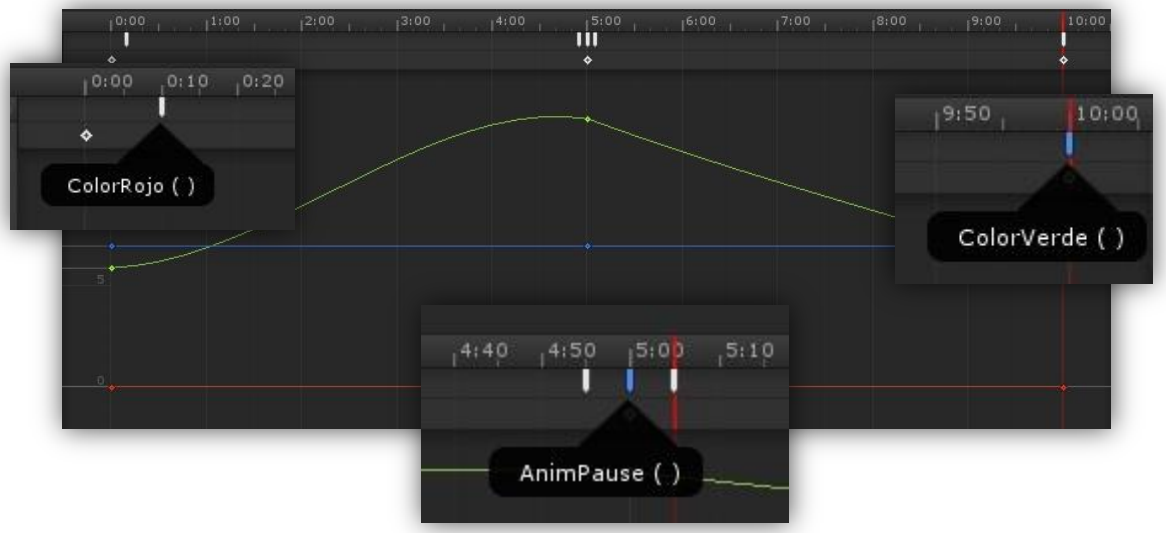


4. Ahora sobre la animación realizada, lo que haremos es agregar eventos para controlar cuando este en movimiento el elevador cambie a color rojo y en verde cuando no esté en movimiento:

- Segundo 0.01 Evento → ColorRojo().

- Segundo 2.29 Evento → ColorVerde().
- Segundo 2.30 Evento → AnimPause().
- Segundo 2.31 Evento → ColorRojo().

- Segundo 4.49 Evento → AnimPause().
- Segundo 5.00 Evento → ColorVerde().



Acciones de la función Animation:

```
//Reproducir la animación general o por animación.
 animation.Play(); / animation.Play("Elevator");


//Detener la animación general y rebobinar o por animación.
animation.Stop(); / animation.Stop("Elevator");


//Detener la animación en general o por animación.
animation.Rewind(); / animation.Rewind("Elevator");


//Activar/Desactivar (Pause) la animación en general o por animación.
animation.enabled = true/false;  /   animation["Elevator"].enabled = true/false;

//Controla la animación Walk su velocidad = reproducción atrás (-1)/Adelante (1).
animation["Elevator "].speed = -1 ó 1;

//Esperar hasta que la animación termine de reproducir.
yield WaitForSeconds (animation.clip.lenght);

//Modo de reproducción de las animaciones.
animation.wrapMode = WrapMode.Loop;     →   (WrapMode.Default, .One, .Loop, .PingPong, .ClampForever).

//Reproducción de Animacion con transición (Blending) de acuerdo a tiempo.
animation.CrossFade ("Elevator", 0.5);
```

1.29 DISTANCE & VISUAL COMPONENTS

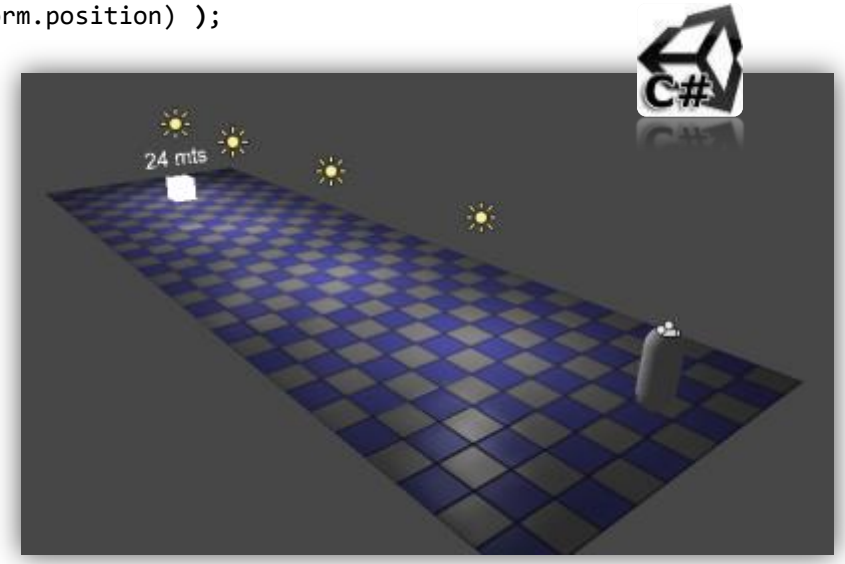
Distance, Enabled & Active Components (C# Script):

static function **Distance** (**a** : Vector3, **b** : Vector3) : **float**
Podemos obtener la distancia entre un punto a otro y poder utilizar el valor para realizar alguna función extra.

1. Cargamos el paquete "Distance" y creamos un script "**AI_Distance**" y se lo asignamos al **3D Text**.

AI_Distance

```
public Transform Character;
public Transform Box;
private float Distance;

void Update () {
	Distance = Mathf.Round ( Vector3.Distance (Character.position, transform.position) );
	GetComponent <TextMesh>().text = Round.ToString () + " mts";

	if (Distance <= 3){
		Box.renderer.enabled = false;		//Render Desactivado.
		Box.collider.enabled = false;		//Activar Collider.
	}else if (Distance >= 3) {
		Box.renderer.enabled = true;		// Render Activado.
		Box.collider.enabled = true;		// Desactivar Collider.
	}
}
```



2DO METODO DESACTIVANDO POR COMPLETO LOS COMPONENTES DEL OBJETO.

```
public Transform Personaje;
public Transform Box;
private float Distance;

void Update () {
	Distance = Mathf.Round ( Vector3.Distance (Character.position, transform.position) );
	GetComponent<TextMesh>().text = Round.ToString () + " mts";

	if (Distance <= 3){
		Box.gameObject.SetActive(false);	//Desactiva al objeto y todos los componentes.
	}else if (Distance >= 3) {
		Box. gameObject .SetActive(true);		//Activa al objeto y todos los componentes.
	}
}
```

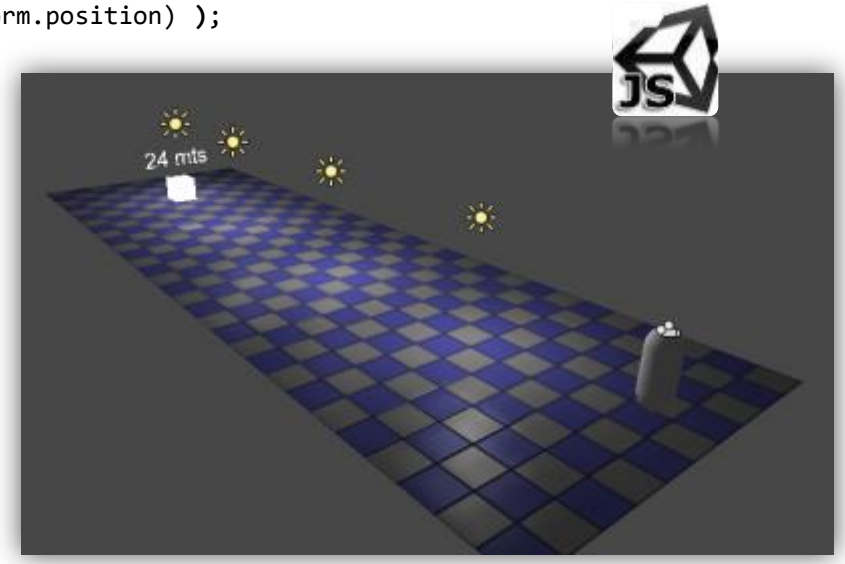
Distance, Enabled & Active Components (C# Script):

static function **Distance** (**a** : Vector3, **b** : Vector3) : **float**
Podemos obtener la distancia entre un punto a otro y poder utilizar el valor para realizar alguna función extra.

1. Cargamos el paquete "Distance" y creamos un script "**AI_Distance**" y se lo asignamos al **3D Text**.

AI_Distance

```
var Character : Transform;
var Box : Transform;
private var Distance : float;

function Update () {
	Distance = Mathf.Round ( Vector3.Distance (Character.position, transform.position) );
	GetComponent <TextMesh>().text = Round.ToString () + " mts";

	if (Distance <= 3){
		Box.renderer.enabled = false;		//Render Desactivado.
		Box.collider.enabled = false;		//Activar Collider.
	}else if (Distance >= 3) {
		Box.renderer.enabled = true;		// Render Activado.
		Box.collider.enabled = true;		// Desactivar Collider.
	}
}
```



2DO METODO DESACTIVANDO POR COMPLETO LOS COMPONENTES DEL OBJETO.

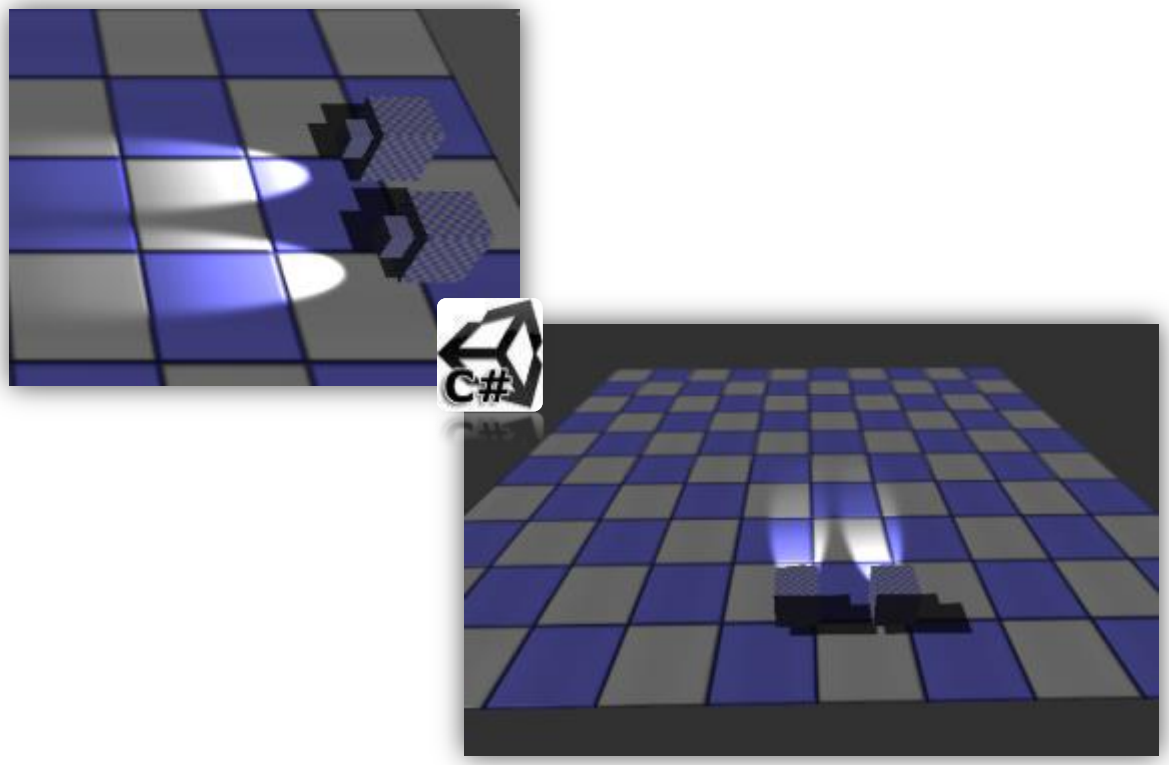
```
var Character : Transform;
var Box : Transform;
private var Distance : float;

function Update () {
	Distance = Mathf.Round ( Vector3.Distance (Character.position, transform.position) );
	GetComponent<TextMesh>().text = Round.ToString () + " mts";

	if (Distance <= 3){
		Box.gameObject.SetActive(false);	//Desactiva al objeto y todos los componentes.
	}else if (Distance >= 3) {
		Box. gameObject.SetActive(true);	//Activa al objeto y todos los componentes.
	}
}
```

1.30 SLOWMOTION.

SlowMotion (C# Script):

Para controlar la velocidad general de la escena se usa Time.timeScale, pero para poder hacerlo individualmente por objeto es diferente:

1. Cargamos el paquete "**ControlSpeed_Pause"** Creamos un Script "**AI_Camera**" y lo asignamos a la cámara.



AI_Camera

```
void Update () {
        print ("DeltaTime: "  Time.deltaTime);
        print ("DeltaTime: "  Time. fixedDeltaTime);

        if (Input.GetKeyDown (KeyCode.Mouse0)) {
                Time.timeScale = 0.1f;
                print ("Slowmotion Speed");
        }
        if (Input.GetKeyDown (KeyCode.Mouse1)) {
                Time.timeScale = 1.0f;
                print ("Normal Speed");
        }

}
```

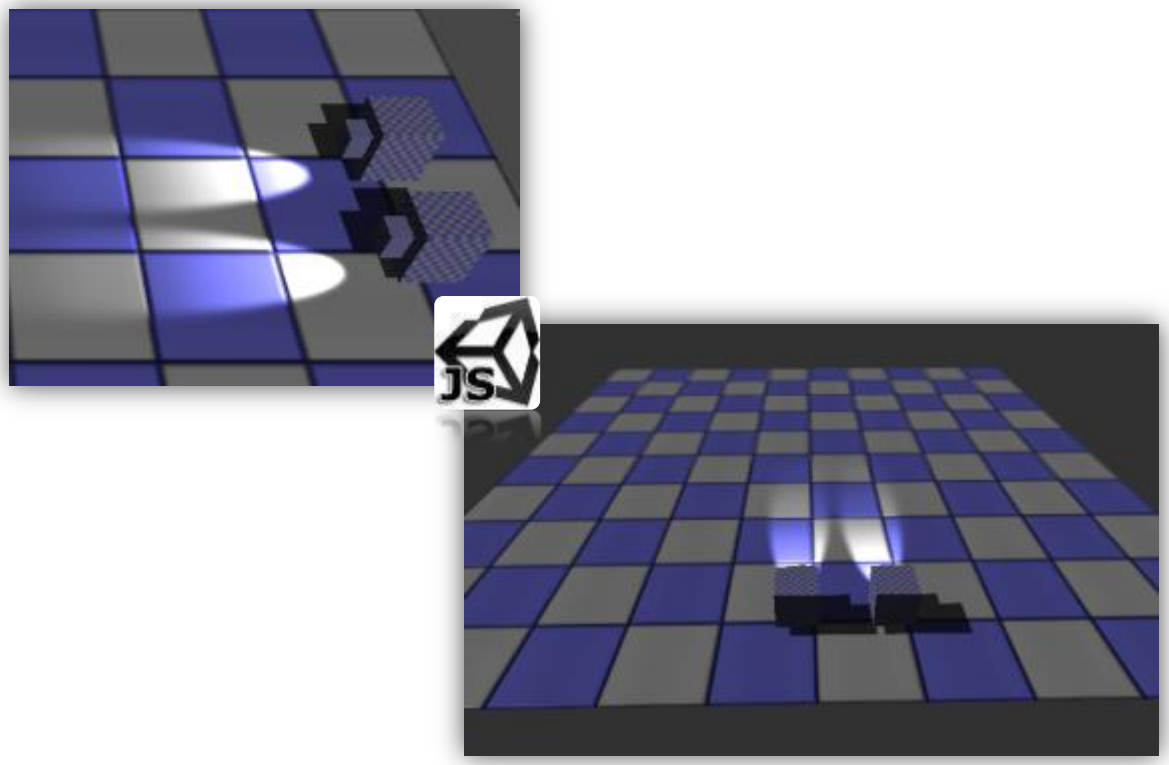
AI_CarSpeed

```
public enum fps {Normal, Slow};
public fps Delta;
float vDeltaTime;

void Update () {
        if (Delta == fps.Normal) {
                        vDeltaTime = Time.fixedDeltaTime;
                } else {
                        vDeltaTime = Time.deltaTime;
                }
        transform.Translate (Vector3.forward * Time. fixedDeltaTime * 0.6f, Space.Self);
}
```

SlowMotion (C# Script):

Para controlar la velocidad general de la escena se usa Time.timeScale, pero para poder hacerlo individualmente por objeto es diferente:

1. Cargamos el paquete "**ControlSpeed_Pause"** Creamos un Script "**AI_Camera**" y lo asignamos a la cámara.



AI_Camera

```
void Update () {
        print ("DeltaTime: "  Time.deltaTime);
        print ("DeltaTime: "  Time. fixedDeltaTime);

        if (Input.GetKeyDown (KeyCode.Mouse0)) {
                Time.timeScale = 0.1f;
                print ("Slowmotion Speed");
        }
        if (Input.GetKeyDown (KeyCode.Mouse1)) {
                Time.timeScale = 1.0f;
                print ("Normal Speed");
        }

}
```

AI_CarSpeed

```
public enum fps {Normal, Slow};
public fps Delta;
float vDeltaTime;

void Update () {
        if (Delta == fps.Normal) {
                        vDeltaTime = Time.fixedDeltaTime;
                } else {
                        vDeltaTime = Time.deltaTime;
                }
        transform.Translate (Vector3.forward * Time. fixedDeltaTime * 0.6f, Space.Self);
}
```

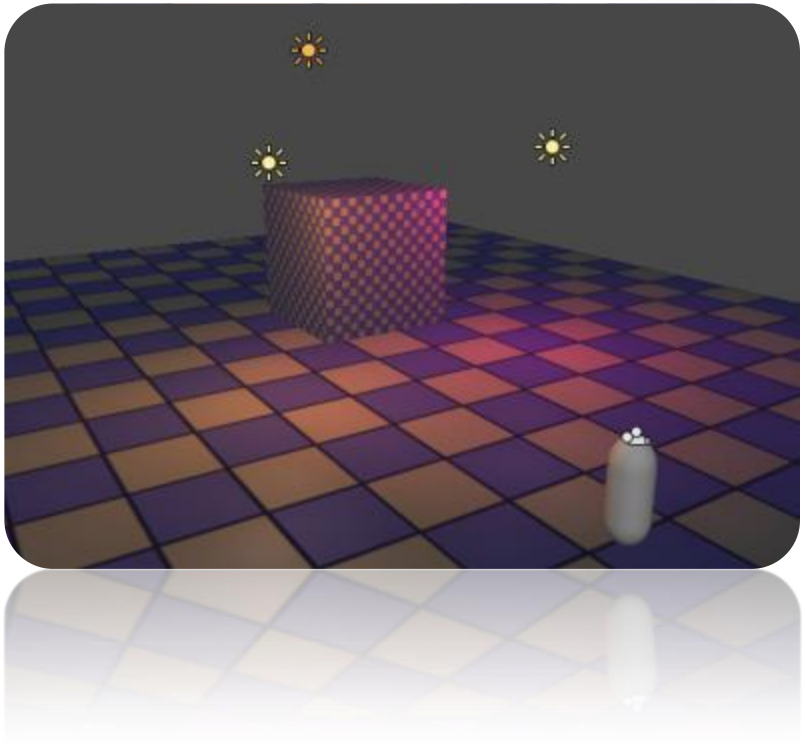
1.31 ONBECAME VISIBLE & INVISIBLE.

OnBecame Visible & Invisible (C# Script):

Podemos Optimizar el performance de los scripts de una manera muy sencilla, si estos no estan dentro del area donde la camara los tenga que ver, lo que se debe de hacer es desactivar el script del objeto, asi ahorramos esos procesos si no se ven y reactivarlos de nuevo al momento que entren en la vision de la camara.

1. Creamos una escena sencilla, un plano, una caja, luces y personaje.
2. Creamos un Script "**AI_Box**" y se lo asignamos a la caja.
3. Este metodo solo sirve cuando la camara de la escena y el Scene view deja de ver al objeto.

Nota: Esto incrementa el performances de manera significativa.



AI_Box

```
//Hacer que la caja este girando
void Update (){
	transform.Rotate (Vector3.up*Time.deltaTime * 15);

}


//Al no ser visible desactivar el script de girar.
void OnBecameInvisible (){
	print("SE DESACTIVO EL SCRIPT!");
	enabled = false;
}


//Al ser visible desactivar el script de girar.
void OnBecameVisible () {
	print("SE ACTIVO EL SCRIPT");
	enabled = true;
}
```

OnBecame Visible & Invisible (Java Script):

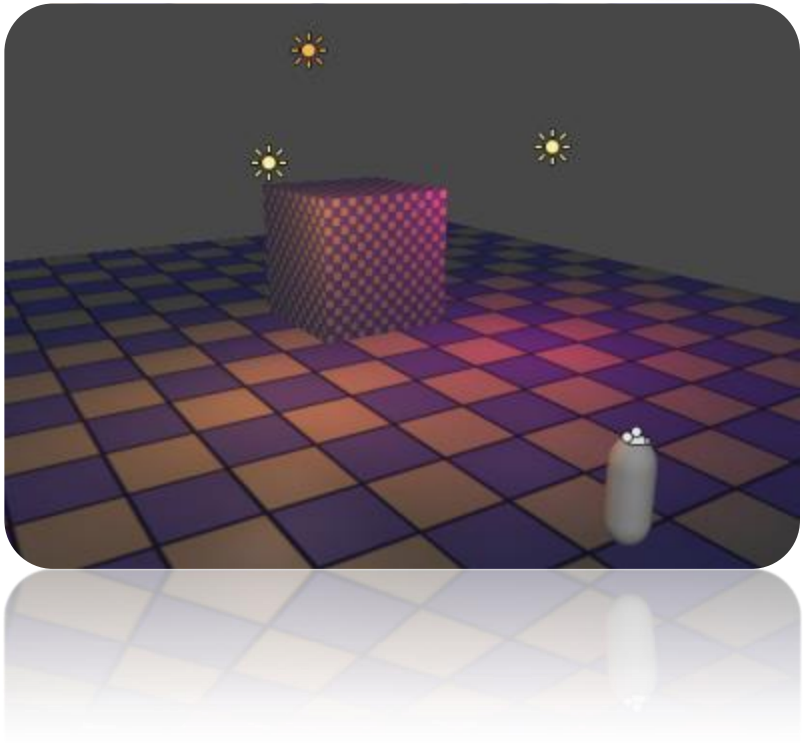
Podemos Optimizar el performance de los scripts de una manera muy sencilla, si estos no estan dentro del area donde la camara los tenga que ver, lo que se debe de hacer es desactivar el script del objeto, asi ahorramos esos procesos si no se ven y reactivarlos de nuevo al momento que entren en la vision de la camara.

1. Creamos una escena sencilla, un plano, una caja, luces y personaje.
2. Creamos un Script "**AI_Box**" y se lo asignamos a la caja.
3. Este metodo solo sirve cuando la camara de la escena y el Scene view deja de ver al objeto.

Nota: Esto incrementa el performances de manera significativa.

AI_Box

```
//Hacer que la caja este girando
function Update (){
	print ("Working");
	transform.Rotate (Vector3.up*Time.deltaTime * 15);

}


//Al no ser visible desactivar el script de girar.
function OnBecameInvisible (){
	print("SE DESACTIVO EL SCRIPT!");
	enabled = false;
}


//Al ser visible desactivar el script de girar.
function OnBecameVisible () {
	print("SE ACTIVO EL SCRIPT");
	enabled = true;
}
```



1.32 RAYCASTING & DRAWRAY.

RayCasting & DrawRay (C# Script):

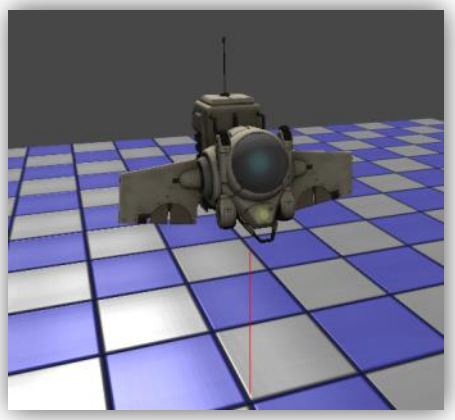
static function **Raycast (origin :** Vector3**, direction :** Vector3**, distance :** float **=Mathf.Infinity, layerMask :** int **= kDefaultRaycastLayers) : bool**

Raycasting permite **lanzar un rayo** desde un **origen** hacia una **dirección**, con una **distancia** de lanzamiento y al **colisionar** puede darnos muchos parámetros de información como **distancia del rayo al colisionar, nombres, collider, etc.**

1. Cargamos el paquete RayCast_DrawRays y creamos un Script "**AI_Creator**" y se lo asignamos al emptygroup Creator.

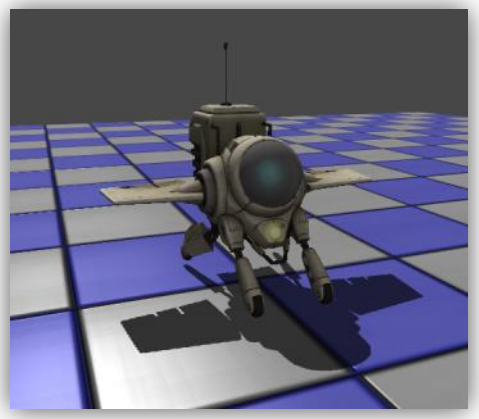
AI_Creator

```
public GameObject Nave;

void Update () {
	if (Input.GetKeyDown (KeyCode.Mouse0) ){
		Instantiate (Nave, transform.position, transform.rotation);
	}
}
```



AI_PrefabNave

```
public float RayDistance = 2f;
private RaycastHit hit;
private bool RayActivator = true;

void Update (){
	if (RayActivator) {
		//Dibujar una línea para verlo InGame.
		Debug.DrawRay (transform.position, Vector3.down * RayDistance, Color.red);

		//True, si el rayo intersecta con algo en la escena a 1.5 unidades.
		if (Physics.Raycast (transform.position, Vector3.down, out hit, RayDistance)) {
			if (hit.distance < 1.9) {
				StartCoroutine (Wings());
			}
		}
	}
}

IEnumerator Wings () {
	RayActivator = false;		// Se desactive el NO poder lanzar más el rayo.
	animation.Play ("Animation");
	yield return new WaitForSeconds (0.05f);
	rigidbody.drag = 15f;
}
```



RayCasting & DrawRay (Java Script):

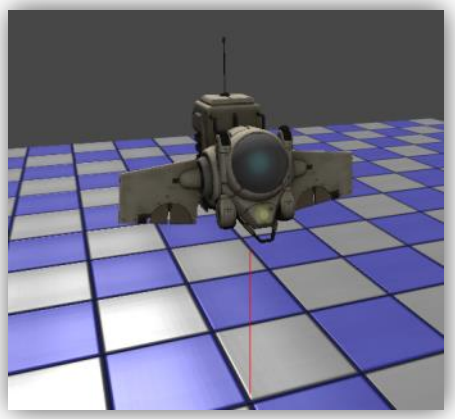
static function **Raycast (origin :** Vector3**, direction :** Vector3**, distance :** float **=Mathf.Infinity, layerMask :** int **= kDefaultRaycastLayers) : bool**

Raycasting permite **lanzar un rayo** desde un **origen** hacia una **dirección**, con una **distancia** de lanzamiento y al **colisionar** puede darnos muchos parámetros de información como **distancia del rayo al colisionar, nombres, collider, etc.**

1. Cargamos el paquete RayCast_DrawRays y creamos un Script "**AI_Creator**" y se lo asignamos al emptygroup Creator.

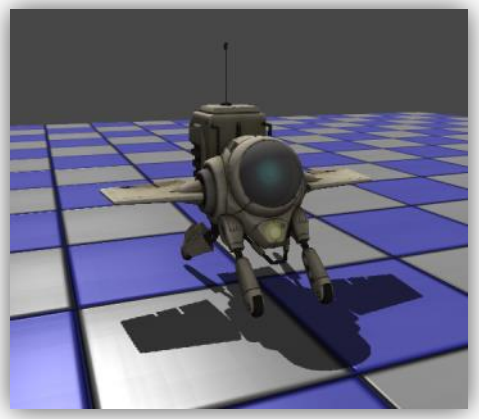
AI_Creator

```
var Nave : GameObject;

function Update () {
	if (Input.GetKeyDown (KeyCode.Mouse0) ){
		Instantiate (Nave, transform.position, transform.rotation);
	}
}
```



AI_PrefabNave

```
var RayDistance : float = 2;
private var RayActivator : boolean = true;
private var hit : RaycastHit;


function Update ()    {
	if (RayActivator) {
		//Dibujar una línea para verlo InGame.
		Debug.DrawRay (transform.position, Vector3.down * RayDistance, Color.red);

		//True, si el rayo intersecta con algo en la escena a 1.5 unidades.
		if (Physics.Raycast (transform.position, Vector3.down, out hit, RayDistance)) {
			if (hit.distance < 1.9) {
				StartCoroutine (Wings());
			}
		}
	}
}

function Wings () {
	RayActivator = false;		// Se desactive el NO poder lanzar más el rayo.
	animation.Play ("Animation");
	yield  WaitForSeconds (0.05);
	rigidbody.drag = 15f;
}
```



1.33 RAYCASTING & ROTATE OBJECTS.

RayCasting & Rotate Objects (C# Script):

static function **Raycast (origin :** Vector3**, direction :** Vector3**, distance :** float **=Mathf.Infinity, layerMask :** int **= kDefaultRaycastLayers) : bool**

En esta practica lo que haremos es hacer que un objeto **Fijo,** rote hacia la direccion del raton, esto sirve para generar un juego donde podriamos tener un personaje o una torre de ataque y esta dispare en direccion a donde este el mouse o al hacer click en una area.

1. Cargamos la escena "**RayCasting_Rotate**" y al objeto Box le crearemos un Script llamado "**AI_RayRotate**".

```
RaycastHit:
-----------
point:        --El punto de impacto en WorldSpace donde colisiono el rayo.
normal:       --La cara de la normal donde imapacto el rayo.
distance:     --La distancia desde el origen del rayo a donde impacto.
collider:     --Con el Collider que colisiono.
rigidbody:    --Rigidbody con collider al que pego (si no tiene rigid el valor es nulo).
```

AI_Box

```
//Variable para controlar la distancia de Rayo.
float Distancia = 30f;
//Variable para obtener informacion cuando el Rayo intersecte.
RaycastHit Hit;

void Update(){
      //variable con la posicion de donde esta el mouse en la patanlla
      Ray Rayo = Camera.main.ScreenPointToRay (Input.mousePosition);

      // Dibujar una linea con los mismos datos del Racast.
      Debug.DrawRay (Camera.main.transform.position, Rayo.direction * Distancia, Color.red);

      if (Physics.Raycast (Rayo, out Hit, Distancia)) {
            //OPCION 1: ROTACION RAPIDA - Solo con la altura en "Y" del objeto (0.5).
            transform.LookAt ( new Vector3 (Hit.point.x, 0.5f ,Hit.point.z) );
      }
}
//OPCION 2: ROTACION SUAVE.
//Obtenemos la rotacion por medio de la posicion de objeto-mouse sin el eje "Y".
Quaternion Rotar = Quaternion.LookRotation (
            ( new Vector3 (Hit.point.x, 0.5f, Hit.point.z) ) - ( new Vector3 (transform.position.x, 0.5f, transform.position.z) ) );
//Rotar con suavidad hacia donde este el mouse.
transform.rotation = Quaternion.Slerp(transform.rotation, Rotar, Time.deltaTime * 2f);
```

RayCasting & Rotate Objects (Java Script):

static function **Raycast (origin :** Vector3**, direction :** Vector3**, distance :** float **=Mathf.Infinity, layerMask :** int **= kDefaultRaycastLayers) : bool**

En esta practica lo que haremos es hacer que un objeto **Fijo,** rote hacia la direccion del raton, esto sirve para generar un juego donde podriamos tener un personaje o una torre de ataque y esta dispare en direccion a donde este el mouse o al hacer click en una area.

1. Cargamos la escena "**RayCasting_Rotate**" y al objeto Box le crearemos un Script llamado "**AI_RayRotate**".

```
RaycastHit:
-----------
point:        --El punto de impacto en WorldSpace donde colisiono el rayo.
normal:       --La cara de la normal donde imapacto el rayo.
distance:     --La distancia desde el origen del rayo a donde impacto.
collider:     --Con el Collider que colisiono.
rigidbody:    --Rigidbody con collider al que pego (si no tiene rigid el valor es nulo).
```

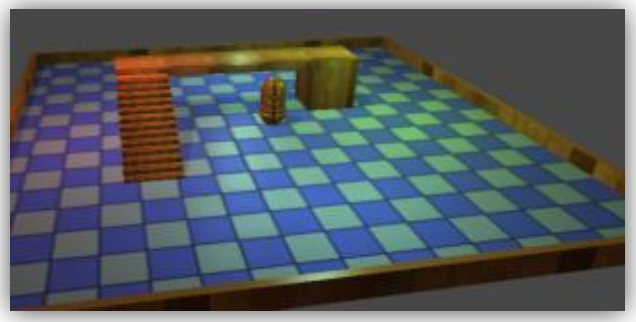
AI_Box

```
//Variable para controlar la distancia de Rayo.
var Distancia : float = 30;
//Variable para obtener informacion cuando el Rayo intersecte.
var Hit : RaycastHit;

function Update(){
	//variable con la posicion de donde esta el mouse en la patanlla
	Rayo : Ray = Camera.main.ScreenPointToRay (Input.mousePosition);

	// Dibujar una linea con los mismos datos del Racast.
	Debug.DrawRay (Camera.main.transform.position, Rayo.direction * Distancia, Color.red);

	if (Physics.Raycast (Rayo, Hit, Distancia)) {
		//OPCION 1: ROTACION RAPIDA - Solo con la altura en "Y" del objeto (0.5).
		transform.LookAt ( Vector3 (Hit.point.x, 0.5 ,Hit.point.z) );
	}
}
//OPCION 2: ROTACION SUAVE.
//Obtenemos la rotacion por medio de la posicion de objeto-mouse sin el eje "Y".
var Rotar : Quaternion  = Quaternion.LookRotation (
		( Vector3 (Hit.point.x, 0.5, Hit.point.z) ) - ( Vector3 (transform.position.x, 0.5, transform.position.z) ) );
//Rotar con suavidad hacia donde este el mouse.
transform.rotation = Quaternion.Slerp(transform.rotation, Rotar, Time.deltaTime * 2);
```

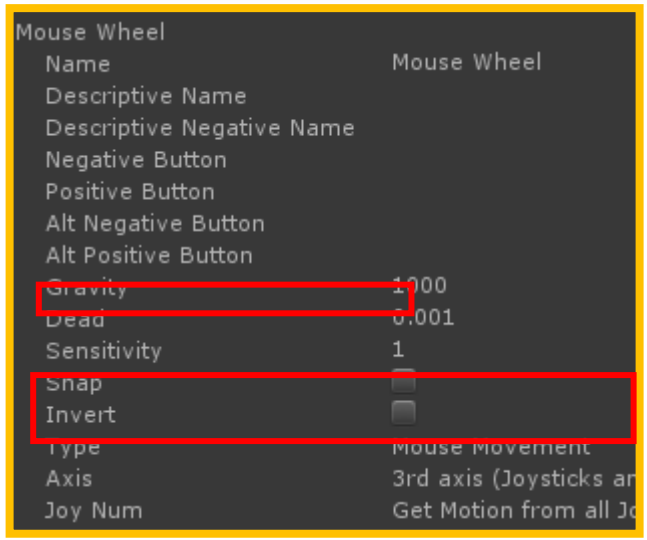
1.34 CLICK & MOVE/ROTATE PLAYER.

Click & Move+Rotate Player (C# Script):

Podemos controlar atributos de movimiento y rotación del jugador por medio del mouse input:



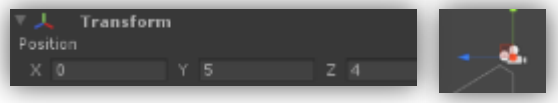
1. **Zoom In/Out**: Sobre la cámara por medio del Scroll.
2. **Player Rotation**: Controlar la rotación del jugador hacia la posición del click del Mouse.
3. **Player Movements**: Controlar la transicion del personaje hacia el click en la escena.
4. **Escena**: Cargamos el paquete "**Click_Move & Rotate Player"** y abrimos la escena.
5. **Inputs:** para el **Mouse Scroll: Edit>Project Settings>Input,** y se llamara "**Mouse Wheel**" y se modificara:
   - **Sensitivity: 1** = Unidades en la que se moverá la cámara cuando usemos el scroll.
   - **Type**: **Mouse Movements** (cuando se usa alguna acción del mouse, en este caso el scroll).
   - **Axis**: **3rd Axis** (activar el uso de botón del Mouse de en medio - Scroll).
6. **Cámara**: creamos un script "**AI_Zoom**" el efecto de usar Scroll y la camara se deplaza suavemente.



AI_Zoom Básico:

```
void Update () {
        //Obtener el valor del uso de Axis = Scroll con su valor de sensitivity =1
        float mouseWheel = Input.GetAxis ("Mouse Wheel");

        //Performance, solo para que al usar el Scroll se active el transform.
        if (mouseWheel != 0f){
                //Mover la cámara de acuerdo a lo que se usa del scroll, con sensibilidad de 1unidad.
                transform.Translate(Vector3.forward * mouseWheel * Time.deltaTime, Space.Self);
        }
}
```

AI_Zoom Avanzado:

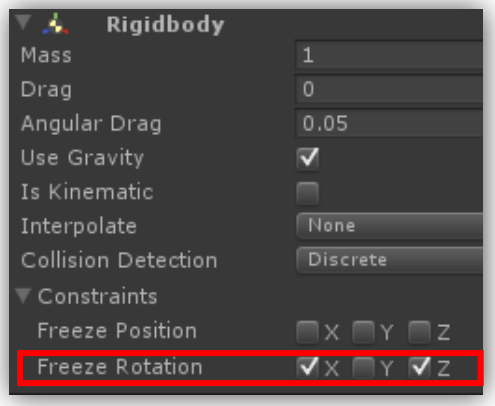



```
//Variable  con la distancia donde este la camara (Checar posicion inicial en Z).
private float vPosicionZ = 4.0f;
private float vPosicionY = 5.0f;
public int vVelocidad;

void Update (){
        // Si se usa el scroll hacia arriba aumentar el valor (Zoom-In).
        if (Input.GetAxis ("Mouse Wheel") > 0f) {

                vPosicionZ -= 1.0f;
                vPosicionY -= 1.0f;
        }
        // Si se usa el scroll hacia abajo recudir el valor (Zoom-Out).
        else if (Input.GetAxis ("Mouse Wheel") < 0f){

                vPosicionZ += 1.0f;
                vPosicionY += 1.0f;
        }
        // Mover la camara hacia el nuevo valor sobre el eje de las "Z".
        transform.position = Vector3.Lerp (transform.position, new Vector3 (transform.position.x, vPosicionY, vPosicionZ), vVelocidad * Time.deltaTime);

}
```

7. **Player**: **Rigidbody>Freeze Rotation → X, Y, Z** (para al desplazar no rote con la velocidad y se caiga).

MOVING:

Physics.Raycast
Raycast (origin : Vector3, direction : Vector3, distance : float=Mathf.Infinity,layerMask : int=kDefaultRaycastLayers) : boolean
Lanza un rayo contra todos los colliders en la escena.



Parametros:
origin --The starting point of the ray in world coordinates.
direction --La direccion del rayo.
distance --La longitus del rayo.
layerMask --MaskLayer se usada para ignorar colliders al lanzar el rayo.

RaycastHit:
---
point: --El punto de impacto en WorldSpace donde colisiono el rayo.
normal: --La cara de la normal donde imapacto el rayo.
distance: --La distancia desde el origen del rayo a donde impacto.
collider: --Con el Collider que colisiono.
rigidbody: --El Rigidbody del collider con el que pego (si no tiene rigid el valor es nulo).

ROTATING:

Plane.Raycast
void Raycast (ray : Ray, out enter : float) : boolean
Intersecta un rayo con el plano, la funcion permite saber la distancia a lo largo del rayo donde intersecta con el plano.

Ray:
---
origin: --El punto de origen del rayo -> Rayo.origin : Vector3.
direction: --La direccion del Rayo -> Rayo.direction : Vector3.
GetPoint: --Devuelve un punto en unidades de distancia a lo largo del rayo --> Rayo.GetPoint(Hit) : Vector3.

8. **Player:** se le dará la habilidad de desplazarse donde se dé un click sobre la escena, creamos un Script "**AI_Moving**":

AI_Moving

```
//variables para mover y rotar con suavidad.
public float Velocidad = 1.0f;

//Raycast: Se usa para al colisionar obtener multiples informaciones como: Pos(point)/normal
private RaycastHit Hit;

//Variable para almacenar la posición donde se haga click.
private Vector3 PosDest;


// Uso de Fixed para objetos con fisicas.
void FixedUpdate () {

	if (Input.GetKeyDown(KeyCode.Mouse0) ){
		// Genera un Rayo desde la posición de la cámara hacia el puntero del mouse.
		Ray rayo = Camera.main.ScreenPointToRay (Input.mousePosition);

		//Si el rayo colisiona con un collider regresara el "True", el rayo se lanza a 100 unidades de distnacia (Sin Distancia es "infinito").
		// Rayo, HitInfo, Distancia.
		if (Physics.Raycast (rayo, out Hit, 100)){
			//Obtener la posición donde colision HIT (x, y, z).
			PosDest = Hit.point;
		}
	}


	//Mover al personaje cada que cambie la PosicionDestino (Vector3.Lerp "Smooth" / Vector3.MoveTowards "Constant").
	transform.position = Vector3.MoveTowards (transform.position, PosDest, Velocidad  * Time.deltaTime);

}
```

```
//PARA EVITAR QUE EL PERSONAJE BRINQUE HACIA ARRIBA, LO MOVEMOS SOLO EN LOS EJES: "X" y "Z"
transform.position = Vector3.MoveTowards(transform.position,new Vector3(PosDest[0],transform.position.y,PosDest[2]),Time.deltaTime * Velocidad);
                                                                        X              Y              Z
```

9. **Player**: Por último agregaremos la habilidad de que pueda **rotar** el personaje en dirección a donde este el mouse "**AI_Rotating**".

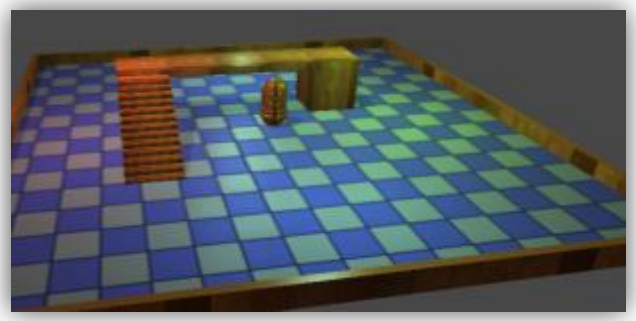
AI_Rotating

```
//Variable para girar con suavidad
public float GirarVelocidad = 4.0f;
//Variable para guardar la distancia del Hit.
private float HitDist;
//Variable para almacenar la posición donde se haga click.
private float PosDet;

void FixedUpdate () {
	// Generar un Plano virtual al centro del personaje sobre el eje "Y"
	Plane PlanoVirtual = new Plane (Vector3.up, transform.position);
	// Lanzar un Rayo desde la posicion de la camara principal hacia la posicion del mouse.
	Ray Rayo = Camera.main.ScreenPointToRay (Input.mousePosition);

	// Se lanza el rayo al plano virtual y al intersectar regresara: true.
	if (PlanoVirtual.Raycast (Rayo, out HitDist)) {
		// Obtener "XYZ" sobre la posicion del mouse sobre el plano virtual.
		Vector3 PosicionPlano = Rayo.GetPoint(HitDist);

		// Angulos de Rotación: Obtener posicion de Mouse en Pantalla "X,Z" --> Conversion a Rotacion para el eje "Y".
		Quaternion AnguloRot = Quaternion.LookRotation(PosicionPlano - transform.position);

		// ROTACAION (Suave/Directa).
		transform.rotation = AnguloRot; // Opcion 1
		transform.rotation = Quaternion.Slerp(transform.rotation, RotarPlayer, Time.deltaTime * GirarVelocidad);
	}
```

Click & Move+Rotate Player (Java Script):

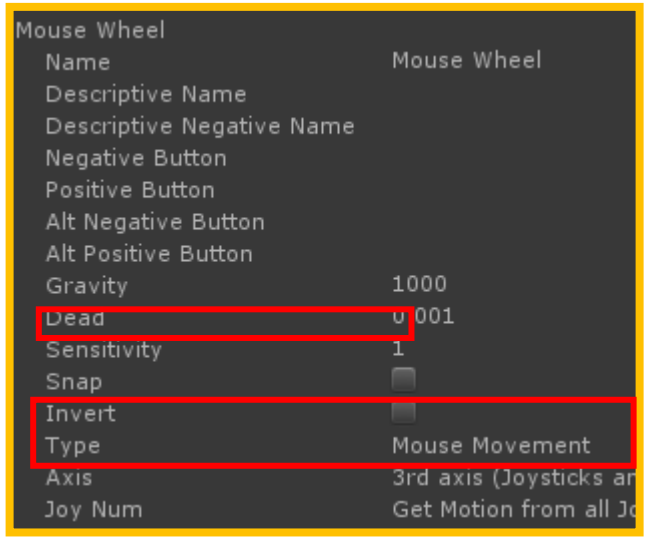
Podemos controlar atributos de movimiento y rotación del jugador por medio del mouse input:

1. **Zoom In/Out**: Sobre la cámara por medio del Scroll.
2. **Player Rotation**: Controlar la rotación del jugador hacia la posición del click del Mouse.
3. **Player Movements**: Controlar la transicion del personaje hacia el click en la escena.
4. **Escena**: Cargamos el paquete "**Click_Move & Rotate Player"** y abrimos la escena.
5. **Inputs:** para el **Mouse Scroll: Edit>Project Settings>Input,** y se llamara "**Mouse Wheel**" y se modificara:
   - **Sensitivity: 1** = Unidades en la que se moverá la cámara cuando usemos el scroll.
   - **Type**: **Mouse Movements** (cuando se usa alguna acción del mouse, en este caso el scroll).
   - **Axis**: **3rd Axis** (activar el uso de botón del Mouse de en medio - Scroll).
6. **Cámara**: creamos un script "**AI_Zoom**" el efecto de usar Scroll y la camara se deplaza suavemente.

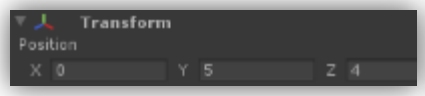


AI_Zoom Básico:

```
function Update () {
        //Obtener el valor del uso de Axis = Scroll con su valor de sensitivity =1
        var mouseWheel : float = Input.GetAxis ("Mouse Wheel");

        //Performance, solo para que al usar el Scroll se active el transform.
        if (mouseWheel != 0){
                //Mover la cámara de acuerdo a lo que se usa del scroll, con sensibilidad de 1unidad.
                transform.Translate (Vector3.forward * mouseWheel * Time.deltaTime, Space.Self);
        }
}
```



AI_Zoom Avanzado:

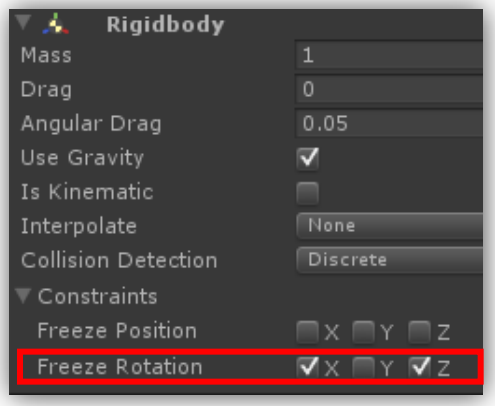



```
//Variable  con la distancia donde este la camara (Checar posicion inicial en Z).
private var vPosicionZ : float = 4.0;
private var vPosicionY : float = 5.0;
var vVelocidad : int;

function Update (){
        // Si se usa el scroll hacia arriba aumentar el valor (Zoom-In).
        if (Input.GetAxis ("Mouse Wheel") > 0.1) {
                vPosicionZ -= 1.0;
                vPosicionY -= 1.0;
        }
        // Si se usa el scroll hacia abajo recudir el valor (Zoom-Out).
        else if (Input.GetAxis ("Mouse Wheel") < -0.1){
                vPosicionZ += 1.0;
                vPosicionY += 1.0;
        }
        // Mover la camara hacia el nuevo valor sobre el eje de las "Z".
        transform.position = Vector3.Lerp (transform.position, Vector3 (transform.position.x, vPosicionY, vPosicionZ), vVelocidad * Time.deltaTime);

}
```

7. **Player**: **Rigidbody>Freeze Rotation → X, Y, Z** (para al desplazar no rote con la velocidad y se caiga).

MOVING:

Physics.Raycast
Raycast (origin : Vector3, direction : Vector3, distance : float=Mathf.Infinity,layerMask : int=kDefaultRaycastLayers) : boolean
Lanza un rayo contra todos los colliders en la escena.



Parametros:
origin --The starting point of the ray in world coordinates.
direction --La direccion del rayo.
distance --La longitus del rayo.
layerMask --MaskLayer se usada para ignorar colliders al lanzar el rayo.

RaycastHit:
---
point: --El punto de impacto en WorldSpace donde colisiono el rayo.
normal: --La cara de la normal donde imapacto el rayo.
distance: --La distancia desde el origen del rayo a donde impacto.
collider: --Con el Collider que colisiono.
rigidbody: --El Rigidbody del collider con el que pego (si no tiene rigid el valor es nulo).

ROTATING:

Plane.Raycast
function Raycast (ray : Ray, out enter : float) : boolean
Intersecta un rayo con el plano, la funcion permite saber la distancia a lo largo del rayo donde intersecta con el plano.

Ray:
---
origin: --El punto de origen del rayo -> Rayo.origin : Vector3.
direction: --La direccion del Rayo -> Rayo.direction : Vector3.
GetPoint: --Devuelve un punto en unidades de distancia a lo largo del rayo --> Rayo.GetPoint(Hit) : Vector3.

8. **Player:** se le dará la habilidad de desplazarse donde se dé un click sobre la escena, creamos un Script "**AI_Moving**":

AI_Moving

```
//variables para mover y rotar con suavidad.
var Velocidad : float = 1.0;

//Raycast: Se usa para al colisionar obtener multiples informaciones como: Pos(point)/normal
private var Hit : RaycastHit;

// Variables para almacenar la  posición de donde se haga click.
private var PosDest : Vector3;


// Uso de Fixed para objetos con fisicas.
function FixedUpdate () {

	if (Input.GetKeyDown(KeyCode.Mouse0) ){
		// Genera un Rayo desde la posición de la cámara hacia el puntero del mouse.
		var rayo : Ray = Camera.main.ScreenPointToRay (Input.mousePosition);

		//Si el rayo colisiona con un collider regresara el "True", el rayo se lanza a 100 unidades de distnacia (Sin Distancia es "infinito").
		// Rayo, HitInfo, Distancia.
		if (Physics.Raycast (rayo, Hit, 100)){
			//Obtener la posición donde colision HIT (x, y, z).
			PosDest = Hit.point;
		}
	}


	//Mover al personaje cada que cambie la PosicionDestino (Vector3.Lerp "Smooth" / Vector3.MoveTowards "Constant").
	transform.position = Vector3.MoveTowards (transform.position, PosDest, Velocidad * Time.deltaTime);

}
```

```
//PARA EVITAR QUE EL PERSONAJE BRINQUE HACIA ARRIBA, LO MOVEMOS SOLO EN LOS EJES: "X" y "Z"
transform.position = Vector3.MoveTowards(transform.position,Vector3(PosDest[0],transform.position.y,PosDest[2]),Velocidad * Time.deltaTime);
                                                                    X             Y             Z
```

9. **Player**: Por último agregaremos la habilidad de que pueda **rotar** el personaje en dirección a donde este el mouse "**AI_Rotating**".

AI_Rotating

```
//Variable para girar con suavidad
var GirarVelocidad = 4.0;
//Variable para guardar la distancia del Hit.
private var HitDist : float;
//Variable para almacenar la posición donde se haga click.
private var PosDet : Vector3;

function FixedUpdate () {
	// Generar un Plano virtual al centro del personaje sobre el eje "Y"
	var PlanoVirtual : Plane  = new Plane(Vector3.up, transform.position);
	// Lanzar un Rayo desde la posicion de la camara principal hacia la posicion del mouse.
	var Rayo : Ray = Camera.main.ScreenPointToRay (Input.mousePosition);

	// Se lanza el rayo al plano virtual y al intersectar regresara: true.
	if (PlanoVirtual.Raycast (Rayo, HitDist)) {
		// Obtener "XYZ" sobre la posicion del mouse sobre el plano virtual.
		var PosicionPlano : Vector3 = Rayo.GetPoint(HitDist);
		// Angulos de Rotación: Obtener posicion de Mouse en Pantalla "X,Z" --> Conversion a Rotacion para el eje "Y".
		var RotarPlayer : Quaternion = Quaternion.LookRotation(PosicionPlano - transform.position);

		// ROTACAION (Suave/Directa).
		//transform.rotation = targetRotation;
		transform.rotation = Quaternion.Slerp (transform.rotation, RotarPlayer, Time.deltaTime * GirarVelocidad);
	}

}
```

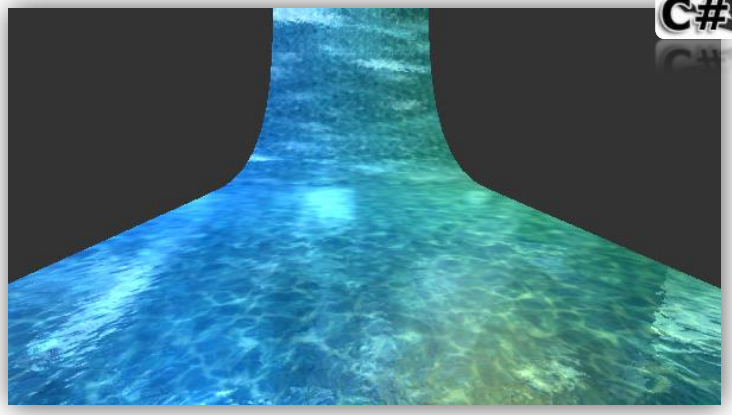
1.35 MATERIALS & TEXTURES.

Material & Properties (C# Script):

TEXTURE PROPERTIES:

SET OFFSET:

"_MainTex" Es la textura del main diffuse. Este solo se acceso vía propiedades de "mainTexture".
"_BumpMap" Es el normal map.
"_Cube" Es el reflection del cubemap.



ANIMACION PARA MOVER UNA TEXTURA CONSTANTEMENTE.

```
// Variable para controlar la velocidad de la textura.
public float  vVelocidad = 0.05f;

void Update () {

	// Para que constantemente este cambiando el tiempo.
	float vOffset = Time.time * vVelocidad;

	// Cambiar el Offset del Diffuse en "X" progresivamente haciendo que su máximo sea el 1 → %.
	renderer.material.SetTextureOffset ("_MainTex", new Vector2 (vOffset % 1, 0));

	// Cambiar el Offset del Bump en "X" progresivamente sin pasar del 1
	renderer.material.SetTextureOffset ("_BumpMap", new Vector2 (-vOffset % 1, 0));

}
```

ANIMACION PARA MOVER UNA TEXTURA DE IDA/REGRESO.

```
void Update () {
	// Variable de tipo ping pong con un Min y Max por el tiempo transcurrido.
	float vPinpong = Mathf.PingPong (Time.time * 0.3f, 1);

	// Cambiar el Offset del material en "Y" de ida y regreso sin pasar de 1 → %.
	renderer.material.SetTextureOffset ("_MainTex", new Vector2 (vPinpong % 1, 0));
}
```

SHADING PROPERTIES:

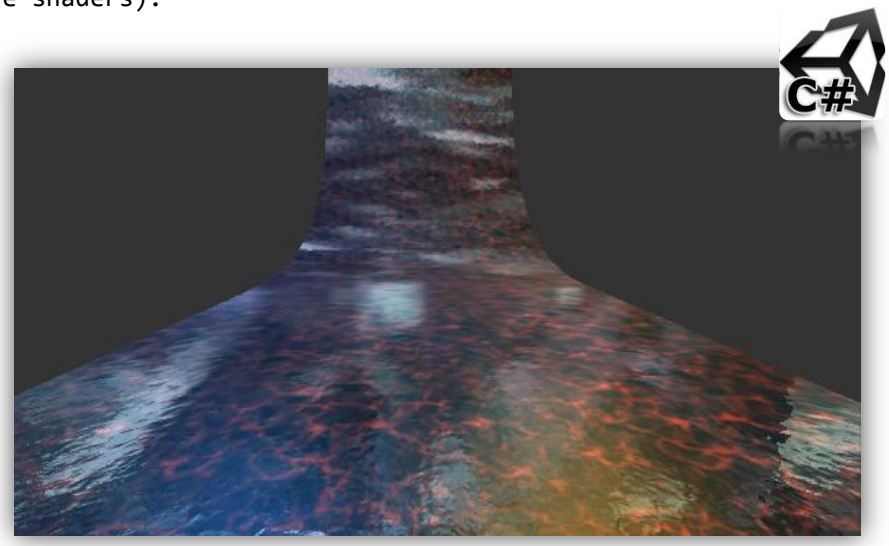
SET COLORS:

```
//Controla el color del material en el canal de "Main Color"
Color vColor = Color.red; → (green, blue, White, Black, yellow, cyan, magenta).
void Start () {
      renderer.material.color = vColor;
}
```

SET SHADINGS:

"_Color" es el main color del material.
"_SpecColor" Es el specular color del material (usado en specular/glossy/vertexlit Shaders).
"_Emission" Es el emissive color del material (usado en vertexlit shaders).
"_ReflectColor" Es el reflection color del material (usado en reflective shaders).

```
void Start () {
      // Control de las características del material.
      renderer.material.SetColor ("_Color", Color.red);
      renderer.material.SetColor ("_SpecColor", Color.green);
      renderer.material.SetColor ("_Emission", Color.blue);
      renderer.material.SetColor ("_ReflectColor", Color.white);
}
```



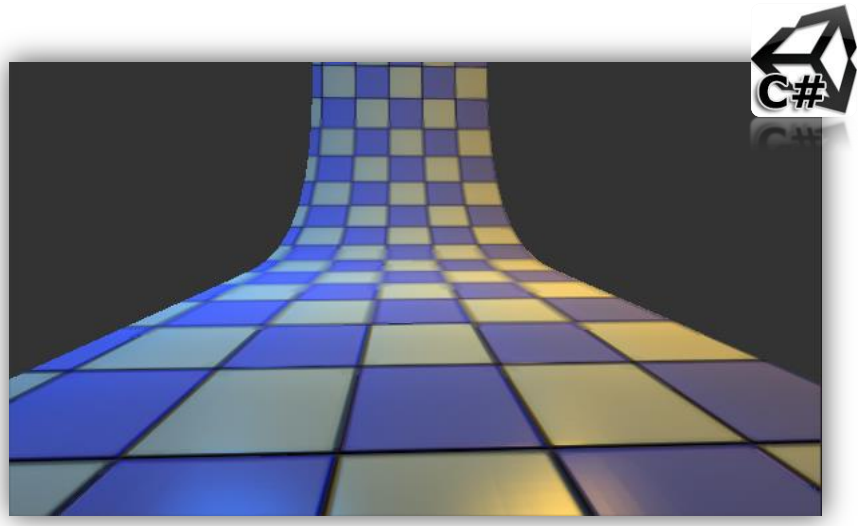
SET BLEND COLOR MATERIAL:

```
//Controla el Blending  entre colores del "Main Color" del material.
Color color1 = Color.red;
Color color2 = Color.gray;

public float duration = 2.0f;


void Update (){
      // Pingpong va de 0 a 1 y regresa, para controlar la velocidad se divide para regresar valor 1.
      float Blend = Mathf.PingPong (Time.time, duration) / duration;
      renderer.material.color = Color.Lerp (color1, color2, Blend);
}
```

MATERIAL PROPERTIES:

SET MATERIALS:

```
//Controla que material quedará activado puesto en una variable.
Material material;
void Start (){
      renderer.material = material;
}
```



SET TEXTURES MATERIALS:

"_MainTex" Es la textura del main diffuse.
"_BumpMap" Es el normal map.
"_Cube" Es el reflection del cubemap.

```
public Texture DiffuseMap;
public Texture BumpMap;
public Texture CubeMap;

void Start (){
      renderer.material.SetTexture ("_MainTex", DiffuseMap);
      renderer.material.SetTexture ("_BumpMap", BumpMap);
      renderer.material.SetTexture ("_Cube", CubeMap);
}
```

SET BLEND COLOR BETWEEN MATERIALS:

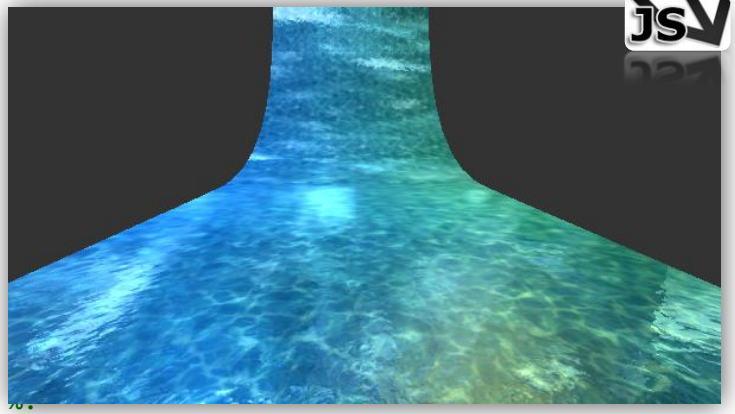
```
// Controla el Blending entre materiales que tengan las mismas texturas (solo cambien características).
Material Material1;
Material Material2;

float duration = 2.0;

void Update (){
      // Pingpong va de 0 a 1 y regresa, para controlar la velocidad se divide para regresar valor 1.
      float Blend = Mathf.PingPong (Time.time, duration) / duration;
      renderer.material.Lerp (Material1, Material2, Blend);
}
```

Material & Properties (Java Script):

TEXTURE PROPERTIES:

SET OFFSET:

"_MainTex" Es la textura del main diffuse. Este solo se acceso vía propiedades de "mainTexture".
"_BumpMap" Es el normal map.
"_Cube" Es el reflection del cubemap.



ANIMACION PARA MOVER UNA TEXTURA CONSTANTEMENTE.

```
// Variable para controlar la velocidad de la textura.
var vVelocidad float = 0.5;

function Update () {

	// Variable para controlar la velocidad de la textura.
	var vOffset : float = Time.time * vVelocidad;

	// Cambiar el Offset del Diffuse en "X" progresivamente sin pasar del 1 → %.
	renderer.material.SetTextureOffset ("_MainTex", Vector2 (vOffset % 1, 0));

	// Cambiar el Offset del Bump en "X" progresivamente sin pasar del 1
	renderer.material.SetTextureOffset ("_BumpMap", Vector2 (-vOffset % 1, 0));

}
```

ANIMACION PARA MOVER UNA TEXTURA DE IDA/REGRESO.

```
function Update () {
	// Variable de tipo ping pong con un Min y Max por el tiempo transcurrido.
	var vPinpong : float = Mathf.PingPong (Time.time * 0.3, 1);

	// Cambiar el Offset del material en "Y" de ida y regreso sin pasar de 1 → %.
	renderer.material.SetTextureOffset ("_MainTex", Vector2 (vPinpong % 1, 0));
}
```

SHADING PROPERTIES:

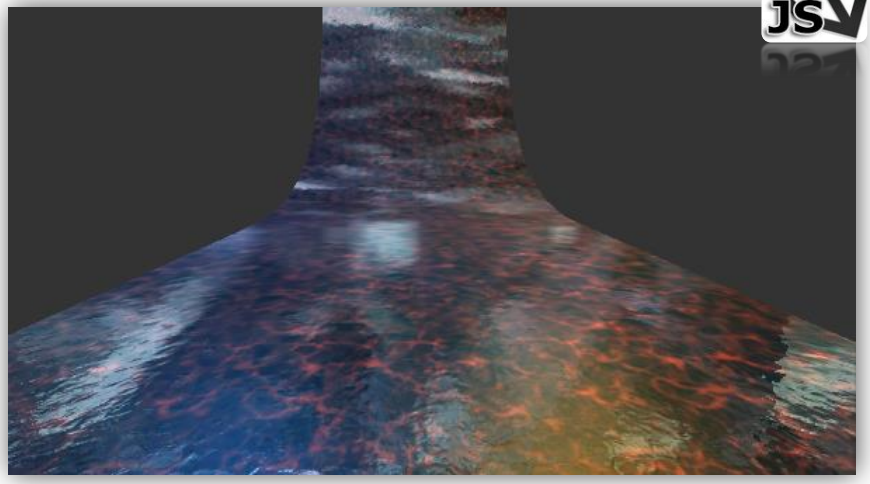
SET COLORS:

```
//Controla el color del material en el canal de "Main Color"
Color vColor = Color.red; → (green, blue, White, Black, yellow, cyan, magenta).
function Start () {
	renderer.material.color = vColor;
}
```

SET SHADINGS:

"_Color" es el main color del material.
"_SpecColor" Es el specular color del material (usado en specular/glossy/vertexlit Shaders).
"_Emission" Es el emissive color del material (usado en vertexlit shaders).
"_ReflectColor" Es el reflection color del material (usado en reflective shaders).

```
function Start () {
	// Control de las características del material.
	renderer.material.SetColor ("_Color", Color.red);
	renderer.material.SetColor ("_SpecColor", Color.green);
	renderer.material.SetColor ("_Emission", Color.blue);
	renderer.material.SetColor ("_ReflectColor", Color.white);
}
```



SET BLEND COLOR MATERIAL:

```
//Controla el Blending  entre colores del "Main Color" del material.
Color color1 = Color.red;
Color color2 = Color.green;

var duration : float = 2;


function Update (){
	// Pingpong va de 0 a 1 y regresa, para controlar la velocidad se divide para regresar valor 1.
	var Blend : float = Mathf.PingPong (Time.time, duration) / duration;
	renderer.material.color = Color.Lerp (color1, color2, Blend);
}
```

MATERIAL PROPERTIES:

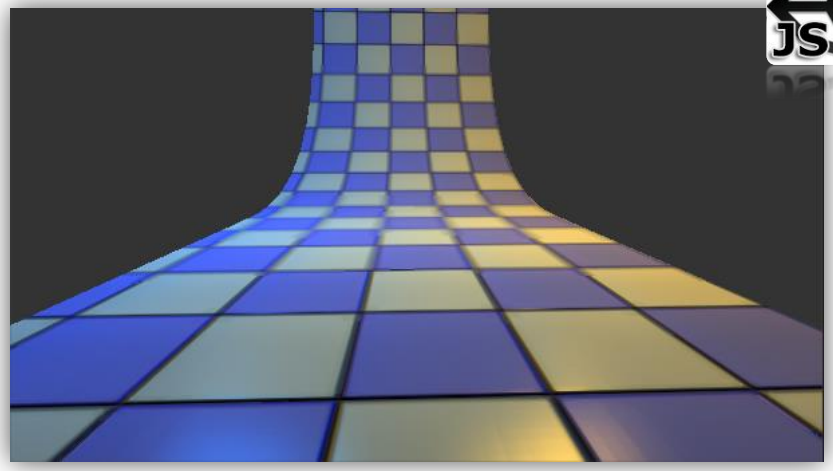
SET MATERIALS:

```
//Controla que material quedará activado puesto en una variable.
var material : Material;
function Start (){
	renderer.material = material;
}
```

SET TEXTURES MATERIALS:

"_MainTex" Es la textura del main diffuse.
"_BumpMap" Es el normal map.
"_Cube" Es el reflection del cubemap.

```
var DiffuseMap : Texture;
var BumpMap : Texture;
var CubeMap : Texture;

function Start (){
	renderer.material.SetTexture ("_MainTex", DiffuseMap);
	renderer.material.SetTexture ("_BumpMap", BumpMap);
	renderer.material.SetTexture ("_Cube", CubeMap);
}
```



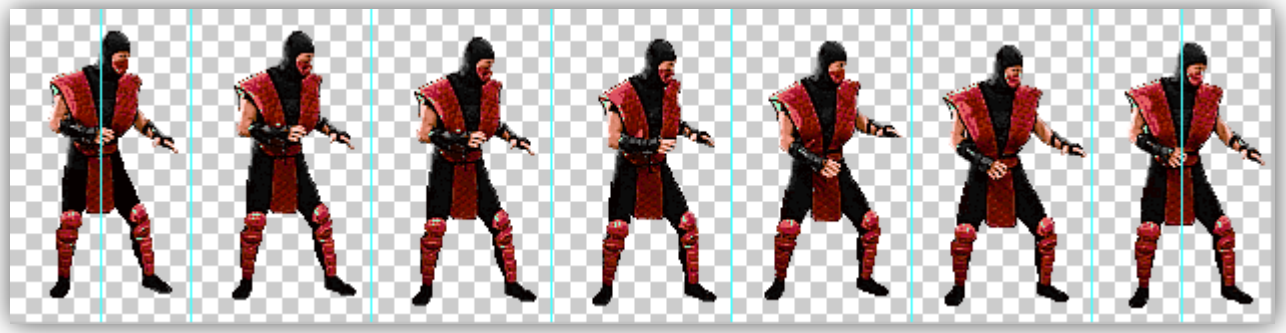
SET BLEND COLOR BETWEEN MATERIALS:

```
// Controla el Blending entre materiales que tengan las mismas texturas (solo cambien características).
var Material1 : Material;
var Material2 : Material;

var duration : float = 2;

function Update (){
	// Pingpong va de 0 a 1 y regresa, para controlar la velocidad se divide para regresar valor 1.
	var Blend : float = Mathf.PingPong (Time.time, duration) / duration;
	renderer.material.Lerp (Material1, Material2, Blend);
}
```

1.36 GUI - ATLAS (SPRITES 2D).

SPRITES/ATLAS TEXTURES (C# Script).

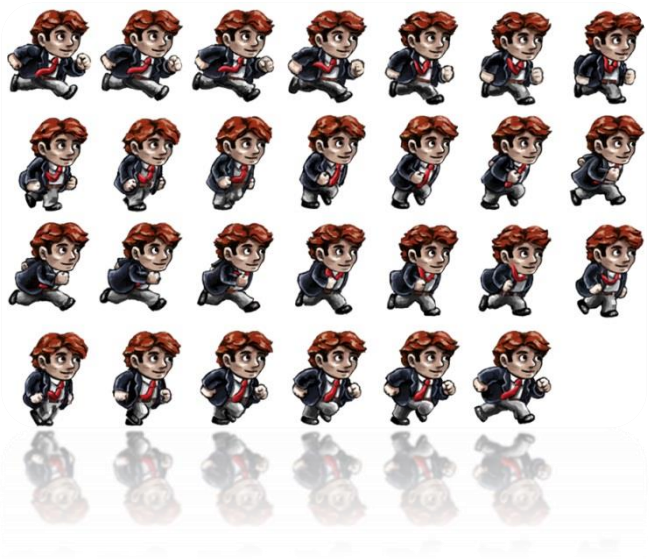
AI_SIMPLE_LINE



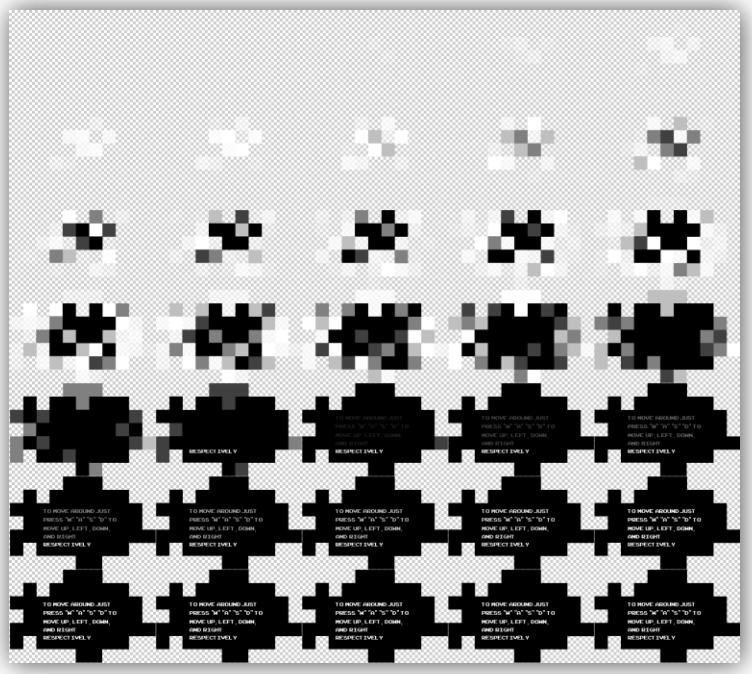
```
// Variables
public int Columns = 7;
public float Framerate = 5f;
//
private float TextureSize;
private float OffsetX;


void Start ()
	{

	//COLUMNAS: Ajustar el tamaño de la textura 1X1 (Unidades de UV).
	TextureSize = 1.0f / Columns;
	//TILLING MAT: Modificar el Tilling (X) para tener alto y ancho por altas.
	renderer.material.SetTextureScale ("_MainTex", new Vector2 (TextureSize, 1f));
	//OFFSET MAT: Modificar el Offset para dezplazarnos al 1er Atlas.
	renderer.material.SetTextureOffset ("_MainTex", new Vector2 (0f, 0f));
	//PLAY: Invocar función con una repetición por segundos
	InvokeRepeating ("Atlas_One_Line", 1f / Framerate, 1f / Framerate);
}


void Atlas_One_Line ()
{
	//OFFSET MAT.
	renderer.material.SetTextureOffset ("_MainTex", new Vector2 (OffsetX, 0));
	OffsetX += TextureSize;   // Avance de Columnas (X)

	//Si es mayor a 1 lo reiniciamos para hacer un Loop
	if (OffsetX >= 1f) {      //Mientras que no pase el limite de Columnas (X)
		OffsetX = 0f;      // Reiniciar Columna (X)
	}

}
```

AI_MULTI_LINES

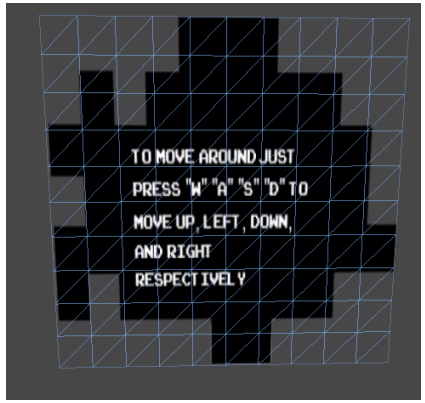
```
//Variables
public int Columns = 7;
public int Rows = 4;
public float Framerate = 30f;
//
private Vector2 TextureSize;
private float OffsetX;
private float OffsetY;

void Start ()
{
    //COLUMNAS: Ajustar el tamaño de la textura 1X1 (Unidades de UV).
    TextureSize = new Vector2 (1f / Columns, 1f / Rows);
    //TILLING MAT: Modificar el Tilling (X) para tener alto y ancho por altas.
    renderer.material.SetTextureScale ("_MainTex", TextureSize);
    //En el caso de este ejemplo de 5 Col y 7 fila -> X:0.2, Y:0.142857
    OffsetX = TextureSize [0];
    OffsetY = -TextureSize [1];
    //OFFSET MAT: Modificar el Offset para dezplazarnos al 1er Atlas.
    renderer.material.SetTextureOffset ("_MainTex", new Vector2 (0, OffsetY));
    //PLAY: Invocar función con una repetición por  segundos
    InvokeRepeating ("Atlas_Multi_GUI", 1f / Framerate, 1f / Framerate);
}

void Atlas_Multi_GUI ()
{
    //OFFSET MAT.
    renderer.material.SetTextureOffset ("_MainTex", new Vector2 (OffsetX, OffsetY));

    OffsetX += TextureSize [0];                 // Valor de Columna (X)
    if (OffsetX >= 1f) {                            // Mientras que no pase el limite de Columnas (X)
          OffsetX = 0.0f;                       // Reiniciar Columna (X)
          OffsetY -= TextureSize [1];           // Regresar 1 fila (Y)
          if (OffsetY < -1f) {                      // Mientras que no pase el limite de Filas (Y)
                OffsetY = -TextureSize [1];     // Reiniciar Filas (Y) como inicio.
          }
    }
}

          1      2      3
```



AI_MULTI_GUI

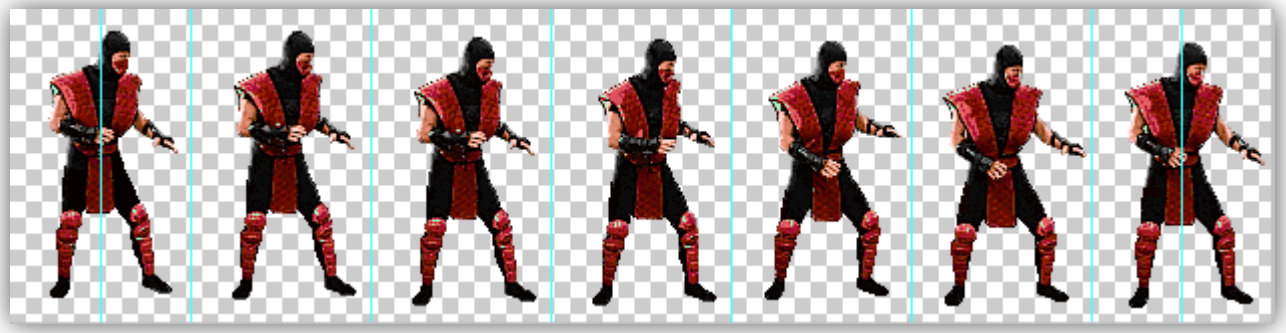


```
public int Columns = 1;
public int Rows = 1;
public float Framerate = 30f;
private Vector2 TextureSize;
private float OffsetX;
private float OffsetY;
public bool Activator = false;

void Start ()
{
        //COLUMNA-FILAS Ajustar el tamaño de la textura 1X1 (Unidades de UV).
        TextureSize = new Vector2 (1f / Columns, 1f / Rows);
        //TILLING MAT: Modificar el Tilling (X,Y) para solo ver el 1er Atlas.
        renderer.material.SetTextureScale ("_MainTex", TextureSize);
        //En el caso de este ejemplo de 5 Col y 7 fila -> X:0.2, Y:0.142857
        OffsetX = TextureSize [0];
        OffsetY = -TextureSize [1];
        //OFFSET MAT: Ver la primer Linea-Columna.
        renderer.material.SetTextureOffset ("_MainTex", new Vector2 (0, OffsetY));
        //Invocar función con inicio y repeticion igual en segundos.
        InvokeRepeating ("Atlas_Multi_GUI", 1f / Framerate, 1f / Framerate);
}

void Atlas_Multi_GUI ()
{
        //OFFSET MAT.
        renderer.material.SetTextureOffset ("_MainTex", new Vector2 (OffsetX, OffsetY));

        //INICIO: 0.142857 a -1 AVANZAR.
        if (Activator) {                                        // Activado - Avanzar
                if (OffsetY > -1f) {                            // Mientras que no pase el limite de Filas (Y)
                        OffsetX += TextureSize [0];             // Valor de Columnas (X)
                        if (OffsetX >= 1) {                     // Mientras que no pase el limite de Columnas (X)
                                OffsetX = 0.0f;                 // Reiniciar columnas (X)
                                OffsetY -= TextureSize [1];     // Avanzar 1 fila (Y)
                        }
                }
        }
        // FINAL: -1 a 0.142857*2 para regresar al 1er Atlas.
        if (! Activator) {                                      // Desactivado - Reversa
                if (OffsetY < -TextureSize [1] * 2) {           // Mientras que no pase el limite de Filas (Y)
                        OffsetX -= TextureSize [0];             // Avance de Columnas (X)
                        if (OffsetX <= 0) {                     // Mientras que no pase el limite de columnas (X)
                                OffsetX = 1f;                   // Reiniciar Columnas (X)
                                OffsetY += TextureSize [1];     // Regresar 1 fila (Y)
                        }
                }
        }
}
        1       2       3       4       5
```



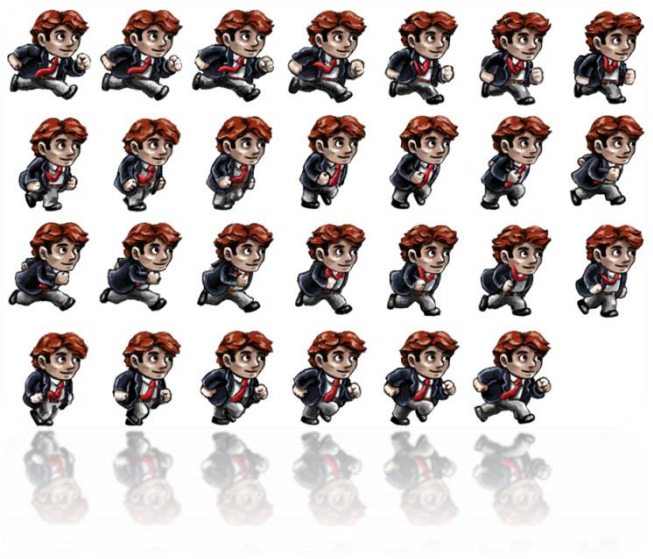
SPRITES/ATLAS TEXTURES (Java Script).

AI_SIMPLE_LINE



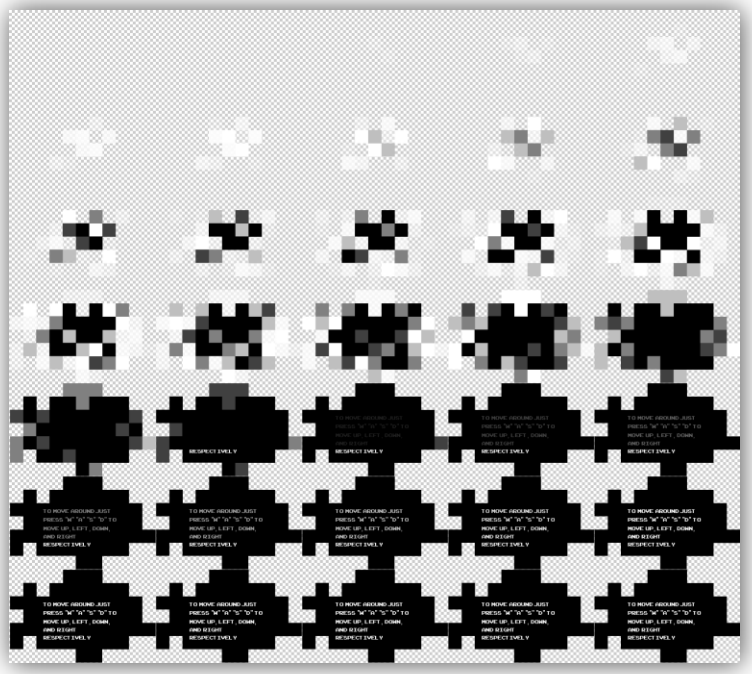
```
// Variables
var Columns : int = 7;
var Framerate : float = 5;
//
var TextureSize : float;
var OffsetX : float;


function Start ()
	{

	//COLUMNAS: Ajustar el tamaño de la textura 1X1 (Unidades de UV).
	TextureSize = 1.0 / Columns;
	//TILLING MAT: Modificar el Tilling (X) para tener alto y ancho por altas.
	renderer.material.SetTextureScale ("_MainTex", Vector2 (TextureSize, 1));
	//OFFSET MAT: Modificar el Offset para dezplazarnos al 1er Atlas.
	renderer.material.SetTextureOffset ("_MainTex", Vector2 (0, 0));
	//PLAY: Invocar función con una repetición por segundos
	InvokeRepeating ("Atlas_One_Line", 1 / Framerate, 1 / Framerate);
}


function Atlas_One_Line ()
{
	//OFFSET MAT.
	renderer.material.SetTextureOffset ("_MainTex", Vector2 (OffsetX, 0));
	OffsetX += TextureSize;   // Avance de Columnas (X)

	//Si es mayor a 1 lo reiniciamos para hacer un Loop
	if (OffsetX >= 1) {	//Mientras que no pase el limite de Columnas (X)
		OffsetX = 0;	// Reiniciar Columna (X)
	}

}
```

AI_MULTI_LINES



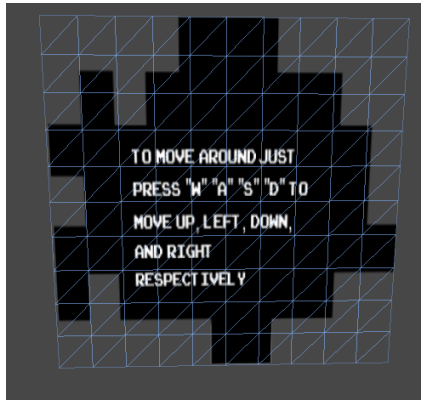
```
//Variables
var Columns : int = 7;
var Rows : int = 4;
var Framerate : float = 30f;
//
private var TextureSize : Vector2;
private var OffsetX : float;
private var OffsetY : float;

function Start ()
{
    //COLUMNAS: Ajustar el tamaño de la textura 1X1 (Unidades de UV).
    TextureSize = Vector2 (1 / Columns, 1 / Rows);
    //TILLING MAT: Modificar el Tilling (X) para tener alto y ancho por altas.
    renderer.material.SetTextureScale ("_MainTex", TextureSize);
    //En el caso de este ejemplo de 5 Col y 7 fila -> X:0.2, Y:0.142857
    OffsetX = TextureSize [0];
    OffsetY = -TextureSize [1];
    //OFFSET MAT: Modificar el Offset para dezplazarnos al 1er Atlas.
    renderer.material.SetTextureOffset ("_MainTex", Vector2 (0, OffsetY));
    //PLAY: Invocar función con una repetición por  segundos
    InvokeRepeating ("Atlas_Multi_GUI", 1 / Framerate, 1 / Framerate);
}

function Atlas_Multi_GUI ()
{
    //OFFSET MAT.
    renderer.material.SetTextureOffset ("_MainTex", Vector2 (OffsetX, OffsetY));

    OffsetX += TextureSize [0];                 // Valor de Columna (X)
    if (OffsetX >= 1) {                         // Mientras que no pase el limite de Columnas (X)
          OffsetX = 0.0;                        // Reiniciar Columna (X)
          OffsetY -= TextureSize [1];           // Regresar 1 fila (Y)
          if (OffsetY < -1) {                   // Mientras que no pase el limite de Filas (Y)
                OffsetY = -TextureSize [1];     // Reiniciar Filas (Y) como inicio.
          }
    }

}               1     2     3
```

AI_MULTI_GUI



```
var Columns : int = 1;
var int Rows : int = 1;
var Framerate : float = 30f;
var TextureSize : Vector2;
//private var OffsetX : float;
private var OffsetY : float;
public var Activator : boolean = false;

void Start ()
{
	//COLUMNA-FILAS Ajustar el tamaño de la textura 1X1 (Unidades de UV).
	TextureSize = Vector2 (1 / Columns, 1 / Rows);
	//TILLING MAT: Modificar el Tilling (X,Y) para solo ver el 1er Atlas.
	renderer.material.SetTextureScale ("_MainTex", TextureSize);
	//En el caso de este ejemplo de 5 Col y 7 fila -> X:0.2, Y:0.142857
	OffsetX = TextureSize [0];
	OffsetY = -TextureSize [1];
	//OFFSET MAT: Ver la primer Linea-Columna.
	renderer.material.SetTextureOffset ("_MainTex", Vector2 (0, OffsetY));
	//Invocar función con inicio y repeticion igual en segundos.
	InvokeRepeating ("Atlas_Multi_GUI", 1 / Framerate, 1 / Framerate);
}

void Atlas_Multi_GUI ()
{
	//OFFSET MAT.
	renderer.material.SetTextureOffset ("_MainTex", Vector2 (OffsetX, OffsetY));

	//INICIO: 0.142857 a -1 AVANZAR.
	if (Activator) {                                    // Activado - Avanzar
		if (OffsetY > -1) {                             // Mientras que no pase el limite de Filas (Y)
			OffsetX += TextureSize [0];                 // Valor de Columnas (X)
			if (OffsetX >= 1) {                         // Mientras que no pase el limite de Columnas (X)
				OffsetX = 0.0;                          // Reiniciar columnas (X)
				OffsetY -= TextureSize [1];             // Avanzar 1 fila (Y)
			}
		}
	}
	// FINAL: -1 a 0.142857*2 para regresar al 1er Atlas.
	if (! Activator) {                                  // Desactivado - Reversa
		if (OffsetY < -TextureSize [1] * 2) {           // Mientras que no pase el limite de Filas (Y)
			OffsetX -= TextureSize [0];                 // Avance de Columnas (X)
			if (OffsetX <= 0) {                         // Mientras que no pase el limite de columnas (X)
				OffsetX = 1;                            // Reiniciar Columnas (X)
				OffsetY += TextureSize [1];             // Regresar 1 fila (Y)
			}
		}
	}
}
```

1 2 3 4 5

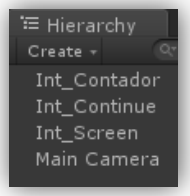


1.37 GUI - MENUS

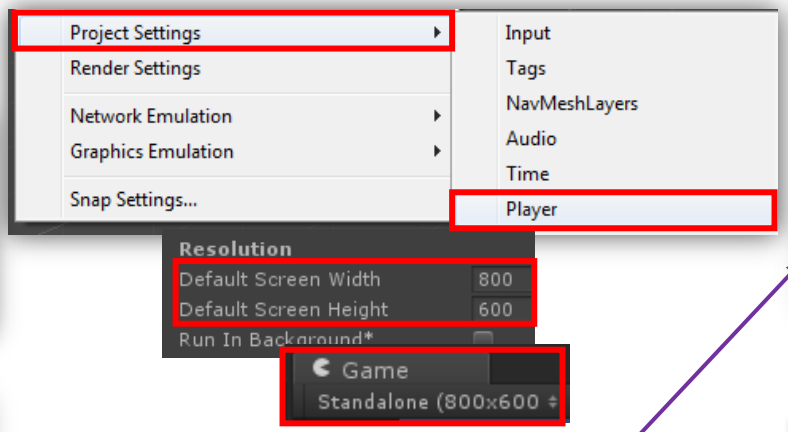
GUI – MENUS Text and Textures (C# Script):

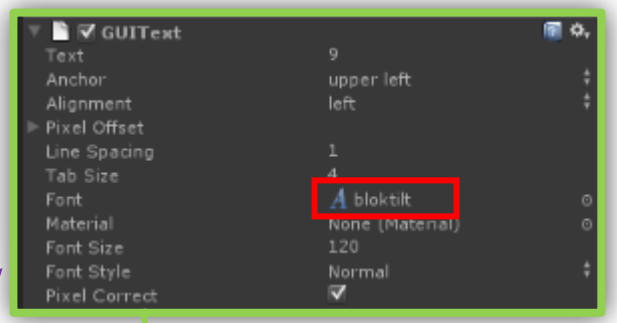
Características para las imágenes a incorporar:

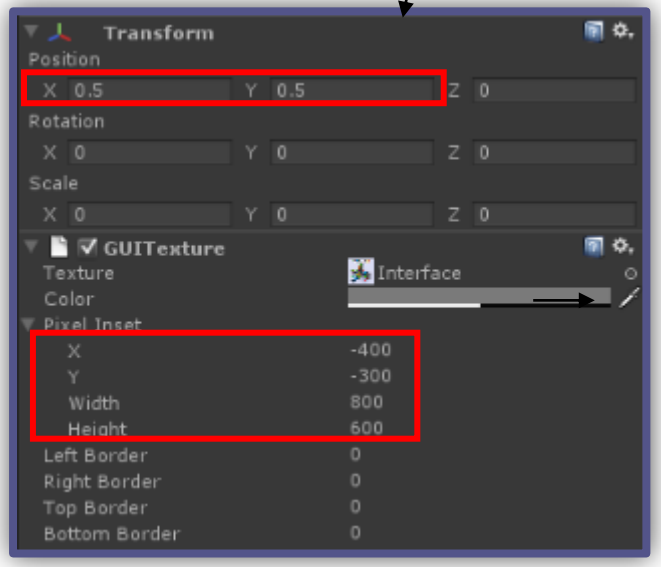
Formato para Imágenes: PSD, PNG TGA, el factor de estos formatos es que pueden guardar información de transparencia.

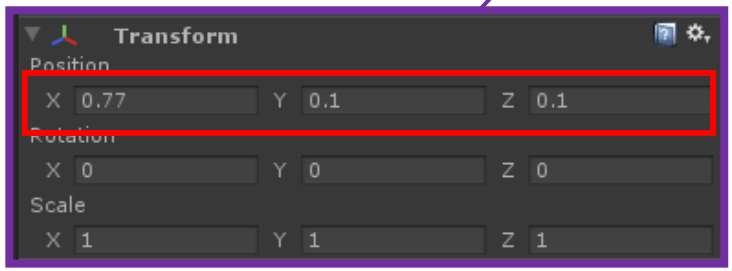


1. Importamos la imagen Interface.jpg y Continue_A y _B, en **Inspector>Format** lo dejamos en **Type: GUI**, **Type: True Color**.
2. Creamos 2 (**Interface**, **Continue_A**): **GameObject>Create Other> GUI Texture**, cargamos cada uno en el valor de **Texture**.
3. Para tener como capaz de GUI, solo se deben de mover en el eje de profundidad de la cámara, en este caso sobre **Z**.
4. Importamos la tipografía **"bloktilt" de Files** y creamos **GameO bject>Create Other> GUI Text** (Le ponemos la tipografía), Para hacer que la imagen de fondo quede siempre al centro en X & Y y/o se ajuste a la resolución Creamos un Script "**AI_Interface**" para la interfaz:









5. Creamos un Script **"AI_Counter"** y se lo asignamos al Int_Counter**:**

Nota: no podemos imprimir variables numéricas en GUIs, hay que convertirlas de Numéricas a Strings.

AI_Counter

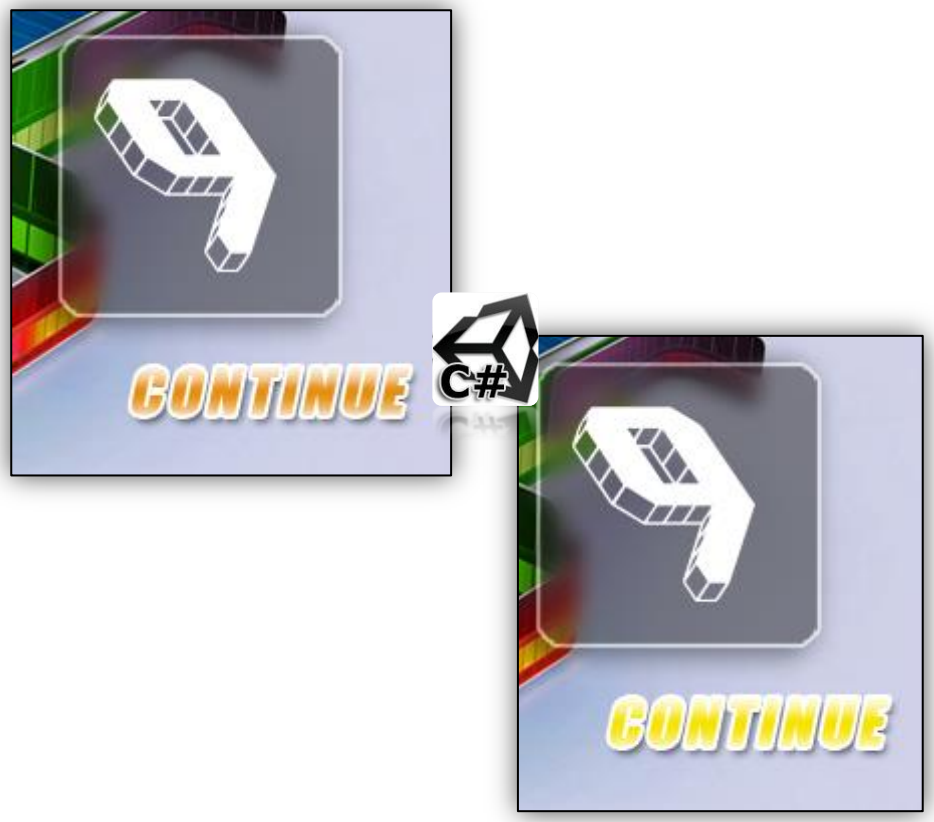
```
IEnumerator Start () {

   //COUNTER
   for (int i = 9; i >= 0; i --){
         yield return new WaitForSeconds (1);
         audio.Play ();
         // Convertir variables numéricas en String para GUI_Text.
         GUI_Text.text = i.ToString ();

         // GAME OVER
         if (i < 0){
               print ("Funciona");
         }
   }
}
```

1 2

6. Creamos un Script **"AI_Continue"** y se lo asignamos a Int_Continue**:**

AI_Continue

```
// Texturas del Botón.
public Texture2D vTexture_A;
public Texture2D vTexture_B;


void OnMouseEnter () {
	GUI_Texture.texture = vTexture_B;
	audio.Play ();
}


void OnMouseExit (){
	GUI_Texture.texture = vTexture_A;
}


void OnMouseUp (){
	print ("Funciona");
}
```



GUI POR MEDIO DE BOTONES NATIVOS DE UNITY.

```
void OnGUI() {
	// Boton Sencillo.
	if (GUI.Button (Rect (10,70,50,30),"Boton") ){
		Debug.Log ("Click con boton con texto");
	}

}

Texture btnTexture;

void OnGUI() {
	// Boton con Imagen.
	if (GUI.Button (Rect (10,10,50,50), btnTexture)) {
		Debug.Log ("Click con boton con imagen");
	}
}
```

MANIPULAR MANUALMENTE GUI CON BASE A RESOLUCIÓN.

```
// Texturas del Botón.
Texture2D vTexture_A;
Texture2D vTexture_B;

//Mover en base a resolución.
float vX;
float vY;

void OnGUI (){
	 // Posición  de GUI par múltiples resoluciones Rect (X, Y, Alto, Ancho).
	GUI_Texture.pixelInset = Rect ( Screen.width*vX,  Screen.height*vY,  Texture_A.width, Texture_A.height );
}
```

GUI TEXT BASIC PROPERTIES.

```
// Cambio de color en el material del texto
GUI_Text.material.color = Color.red/blue/green/white;


// Cambio dinámico del tamaño del texto
GUI_Text.fontSize = 12;

// Cambio dinámico del estilo del texto
GUI_TextfontStyle = FontStyle.Bold;
```

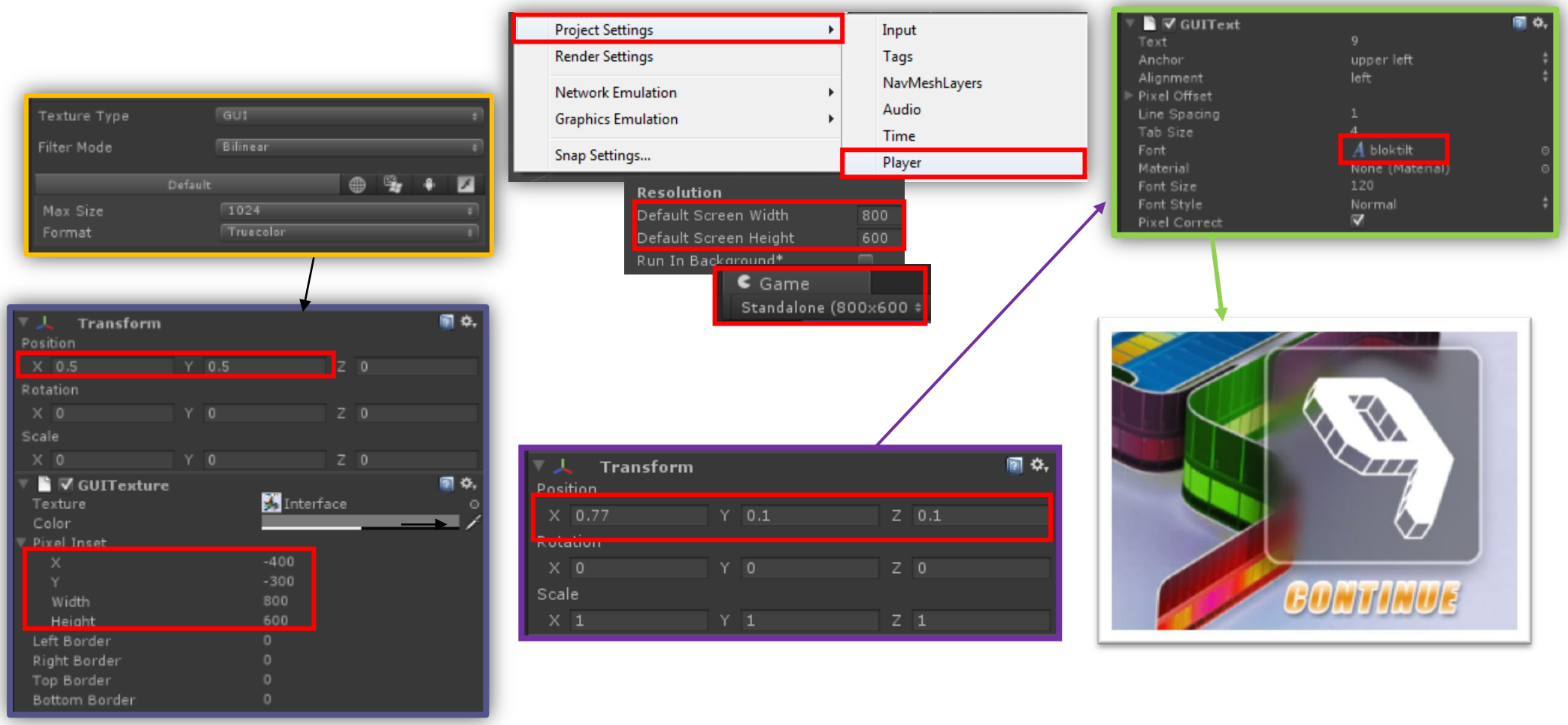
Nota: Siempre debe de estar en 4:3 la proporción par aver que funcione correctamente esta función que creamos.

GUI – MENUS Text and Textures (Java Script):

Características para las imágenes a incorporar:

Formato para Imágenes: PSD, PNG TGA, el factor de estos formatos es que pueden guardar información de transparencia.

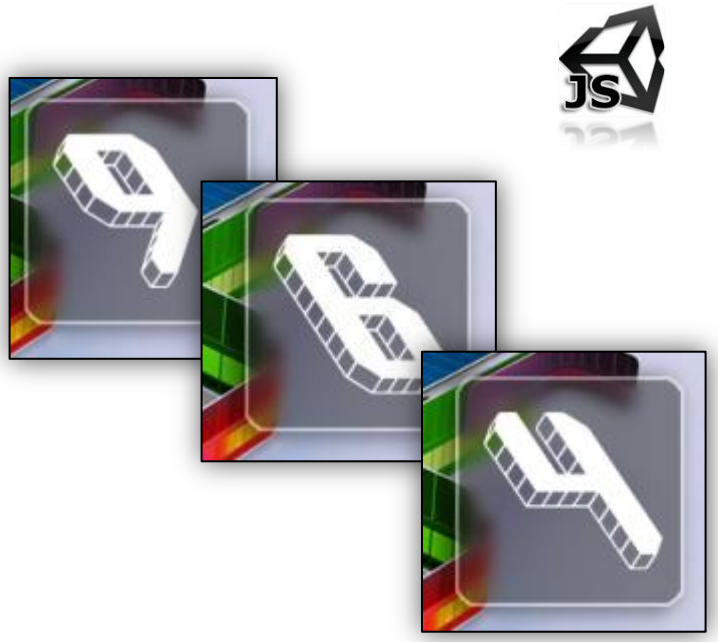
1. Importamos la imagen Interface.jpg y Continue_A y _B, en **Inspector>Format** lo dejamos en **Type: GUI**, **Type: True Color**.
2. Creamos 2 (**Interface**, **Continue_A**): **GameObject>Create Other> GUI Texture**, cargamos cada uno en el valor de **Texture**.
3. Para tener como capaz de GUI, solo se deben de mover en el eje de profundidad de la cámara, en este caso sobre **Z**.
4. Importamos la tipografía **"bloktilt" de Files** y creamos **GameO bject>Create Other> GUI Text** (Le ponemos la tipografía), Para hacer que la imagen de fondo quede siempre al centro en X & Y y/o se ajuste a la resolución Creamos un Script "**AI_Interface**" para la interfaz:



5. Creamos un Script **"AI_Counter"** y se lo asignamos a Int_Counter**:**

Nota: no podemos imprimir variables numéricas en GUIs, hay que convertirlas de Numéricas a Strings.

AI_Counter



```
function Start () {
    //COUNTER
    for (var i : int = 9; i >= 0; i --){
            yield WaitForSeconds (1);
            audio.PlayOneShot (Sonido);
            // Convertir variables numéricas en String para GUI_Text.
            GUI_Text.text = i.ToString ();

            // GAME OVER
            if (i < 0){
                    //Application.LoadLevel (NivelCargar);
            }
    }
}
```

1 2

6. Creamos un Script **"AI_Continue"** y se lo asignamos**:**

AI_Continue

```
// Texturas del Botón.
var vTexture_A : Texture2D;
var vTexture_B : Texture2D;


function OnMouseEnter () {

	GUI_Texture.texture = vTexture_B;
	audio.Play ();
}


function OnMouseExit (){
	GUI_Texture.texture = vTexture_A;
}


function OnMouseUp (){
	print ("Funciona");
}
```

GUI POR MEDIO DE BOTONES NATIVOS DE UNITY.

```
function OnGUI() {
	// Boton Sencillo.
	if (GUI.Button (Rect (10,70,50,30),"Boton") ){
		Debug.Log ("Click con boton con texto");
	}

}

var btnTexture : Texture;

function OnGUI() {
	// Boton con Imagen.
	if (GUI.Button (Rect (10,10,50,50), btnTexture)) {
		Debug.Log ("Click con boton con imagen");
	}
}
```

MANIPULAR MANUALMENTE GUI CON BASE A RESOLUCIÓN.

```
// Texturas del Botón.
var vTexture_A : Texture2D;
var vTexture_B : Texture2D;

//Mover en base a resolución.
var vX : float;
var vY : float;

function OnGUI (){
	 // Posición  de GUI par múltiples resoluciones Rect (X, Y, Alto, Ancho).
	GUI_Texture.pixelInset = Rect ( Screen.width*vX,  Screen.height*vY,  Texture_A.width, Texture_A.height );
}
```

GUI TEXT BASIC PROPERTIES.

```
// Cambio de color en el material del texto
GUI_Text.material.color = Color.red/blue/green/white;
```

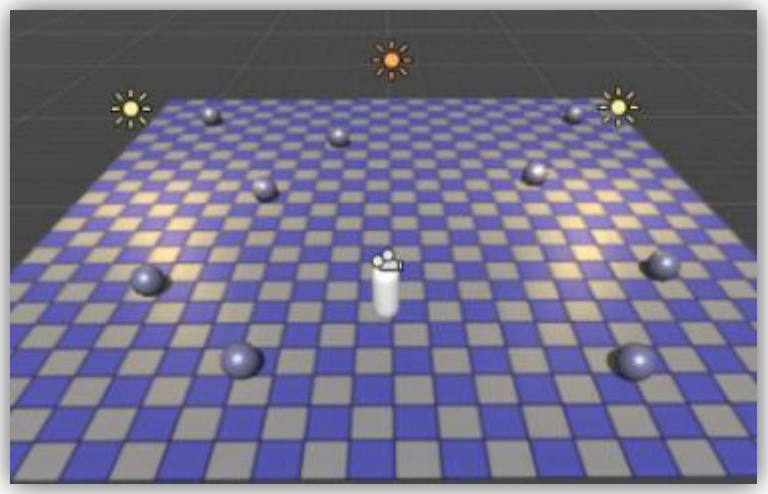
```
// Cambio dinámico del tamaño del texto
GUI_Text.fontSize = 12;

// Cambio dinámico del estilo del texto
GUI_TextfontStyle = FontStyle.Bold;
```

Nota: Siempre debe de estar en 4:3 la proporción par aver que funcione correctamente esta función que creamos.

1.38 GUI – LIFE ENERGY.

GUI – LIFE ENERGY (C# Script):

Podemos crear barras de energías o ítems especiales por medio de GUIs, donde pueden tener transparencias, degradados, y efectos que podemos crear desde Photoshop.



1. Creamos un nuevo proyecto y cargamos el paquete Energy Life_Start.

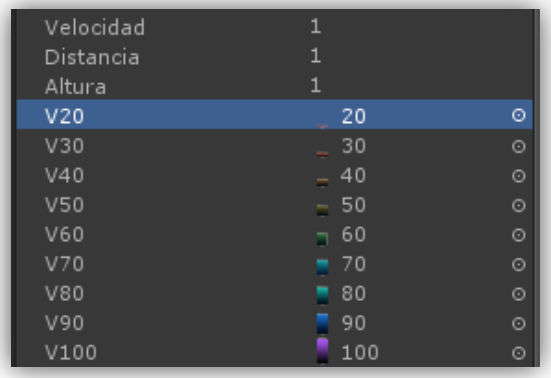
2. Las esferas de la escena son un prefab llamado "Spheres", crearemos un script que se asignara al **Prefab**, y asi animar a todas las esferas hacia arriba y abajo, para poder colocarlas donde sea y tengan esta animación de ida y regreso; asignamos un script "**AI_Ball**":

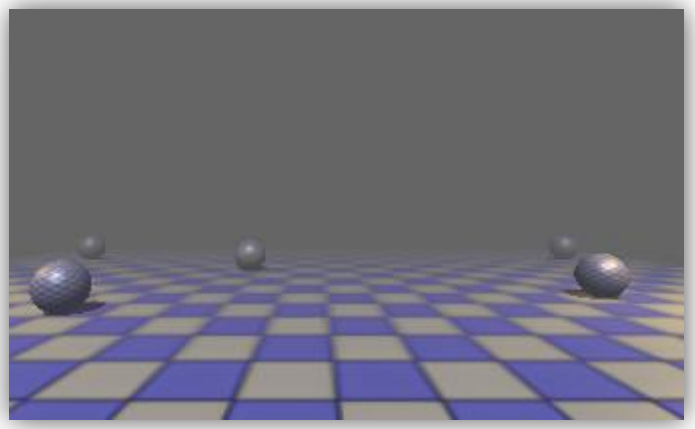
AI_Ball

```
// ANIMACION ESTILO PINGPONG
public float Velocidad;
public float Distancia;
public float Altura;
private float PosY;

void Update () {
      PosY = Mathf.PingPong (Time.time * Velocidad, Distancia) + Altura;
      transform.position = new Vector3 (transform.position.x, PosY, transform.position.z);
}
```

3. Ahora cada que toquemos una de las esferas se destruirá, reproducirá un sonido y asignaremos una variable de vida, para que al tocar cada esfera esta suba un 10%.

```
// CAMBIO DE TEXTURAS
public Texture v20;
public Texture v30;
public Texture v40;
public Texture v50;
public Texture v60;
public Texture v70;
public Texture v80;
public Texture v90;
public Texture v100;

// Variable Global pero No publica en el Inspector.
static int Vida = 10;
```





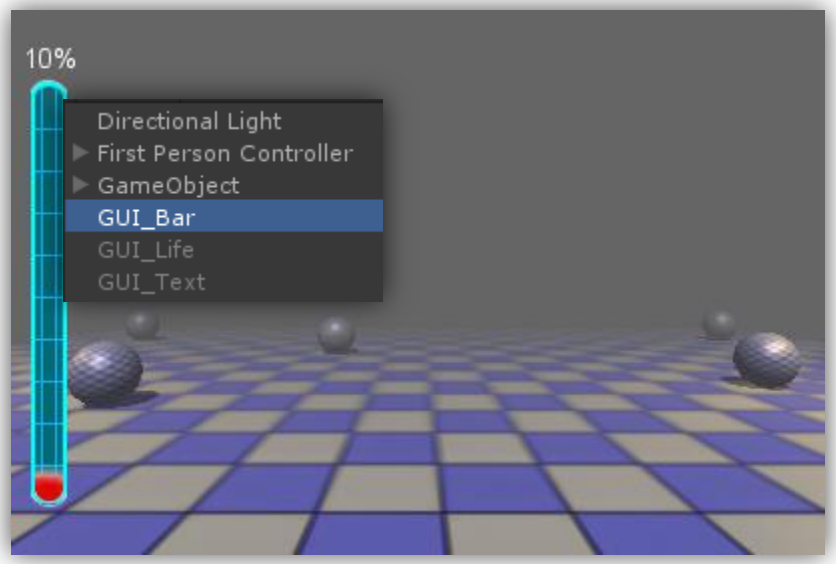
AI_Ball

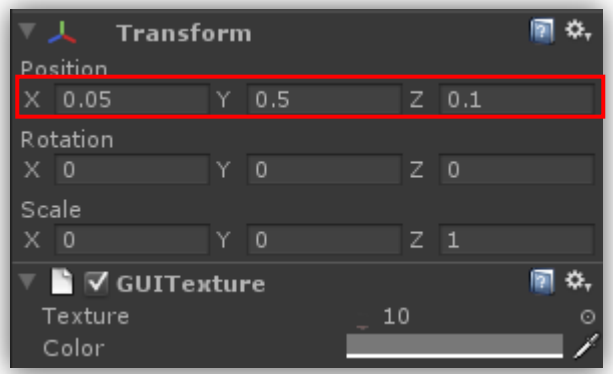
```
void OnTriggerEnter () {

        audio.Play ();
        Destroy (gameObject, 0.5f);

        Vida += 10;

        print (Vida);

}
```

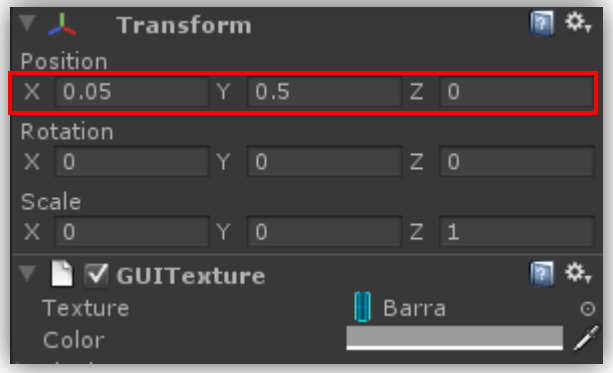
4. Creamos un GUI Texture "**GUI_Bar**", para cargar la **barra** donde se mostraran las texturas con los porcentajes.

5. Creamos un GUI Texture "**GUI_Life**", para cargar los imagenes de los **porcentajes** que se mostraran encima de la barra de energía.

6. Creamos un GUI Text "**GUI_Text**", para mostrar el **porcentaje** de lo que se lleva cargado de manera contextual de la barra de energía.



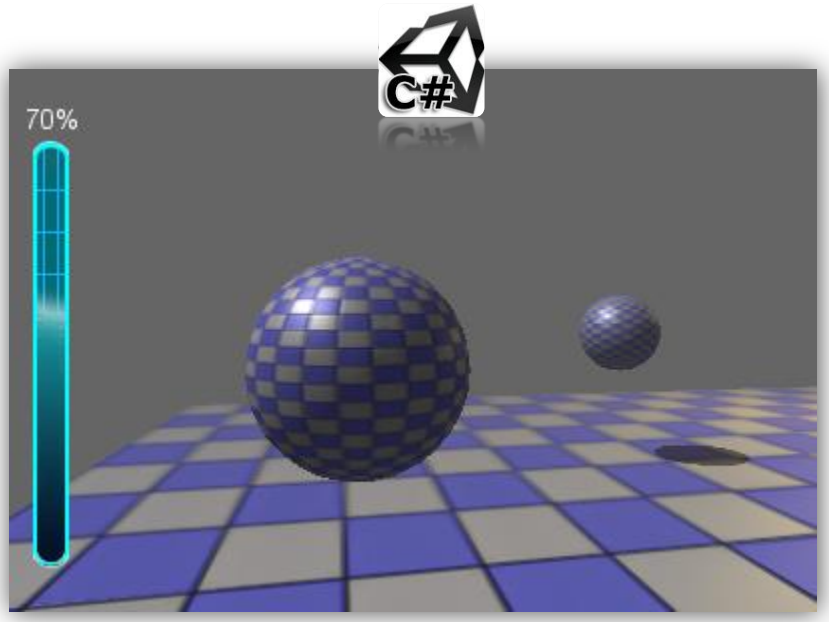




7. Agregamos al OnTrigerEnter la variable de "**Vida**", cada que colisiones, para que aumente de 10 en 10 en su valor Numerico y de Textura, asi de esta manera dara el efecto de que cambia el Porcentaje y la barra de vida va aumentando y cambia de colores.

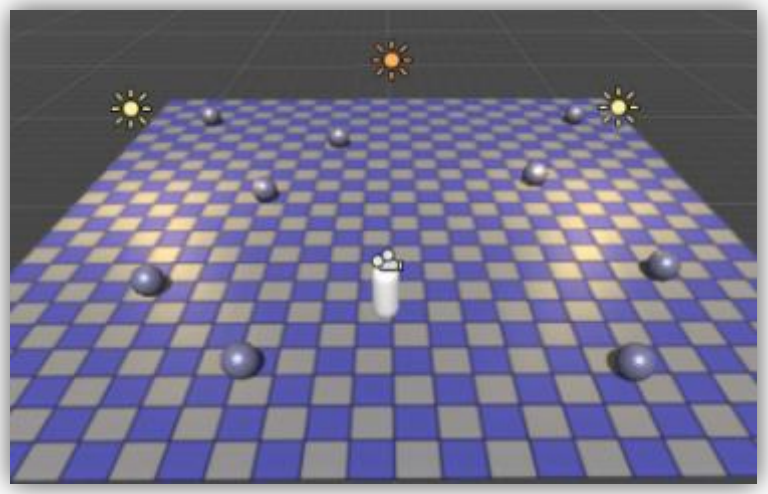
AI_Ball

```
private GameObject GUI_Life;
private GameObject GUI_Text;

void Start () {
        GUI_Life = GameObject.Find ("GUI_Life");
        GUI_Text = GameObject.Find ("GUI_Text");
}

void Update () {
        if (Vida == 20){
                GUI_Life.GUI_Texture.texture = v20;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }else if (Vida == 30){
                GUI_Life.GUI_Texture.texture = v30;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if(Vida == 40){
                GUI_Life.GUI_Texture.texture = v40;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if(Vida == 50){
                GUI_Life.GUI_Texture.texture = v50;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if(Vida == 60){
                GUI_Life.GUI_Texture.texture = v60;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if (Vida == 70){
                GUI_Life.GUI_Texture.texture = v70;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if (Vida == 80){
                GUI_Life.GUI_Texture.texture = v80;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if (Vida == 90){
                GUI_Life.GUI_Texture.texture = v90;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if(Vida == 100){
                GUI_Life.GUI_Texture.texture = v100;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
}
```



GUI – LIFE ENERGY (Java Script):

Podemos crear barras de energías o ítems especiales por medio de GUIs, donde pueden tener transparencias, degradados, y efectos que podemos crear desde Photoshop.



1. Creamos un nuevo proyecto y cargamos el paquete Energy Life_Start.

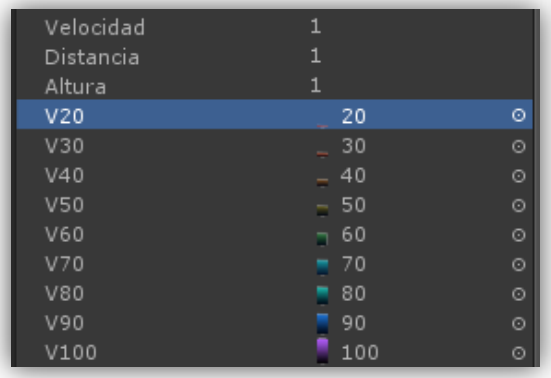
2. Las esferas de la escena son un prefab llamado "Spheres", crearemos un script que se asignara al **Prefab**, y asi animar a todas las esferas hacia arriba y abajo, para poder colocarlas donde sea y tengan esta animación de ida y regreso; asignamos un script "**AI_Ball**":

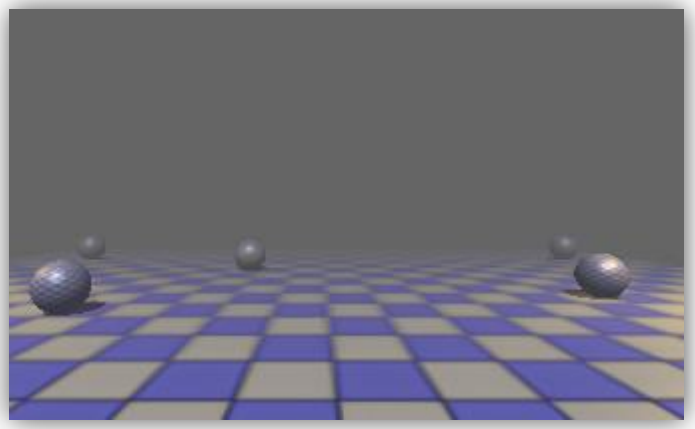
AI_Ball

```
// ANIMACION ESTILO PINGPONG
var Velocidad : float;
var Distancia : float;
var Altura : float;
var PosY : float;

function Update () {
      PosY = Mathf.PingPong (Time.time * Velocidad, Distancia) + Altura;
      transform.position = new Vector3 (transform.position.x, PosY, transform.position.z);
}
```

3. Ahora cada que toquemos una de las esferas se destruirá, reproducirá un sonido y asignaremos una variable de vida, para que al tocar cada esfera esta suba un 10%.

```
// CAMBIO DE TEXTURAS
var v20 : Texture;
var v30 : Texture;
var v40 : Texture;
var v50 : Texture;
var v60 : Texture;
var v70 : Texture;
var v80 : Texture;
var v90 : Texture;
var v100 : Texture;

// Variable Global pero No publica en el Inspector.
static Vida : int = 10;
```





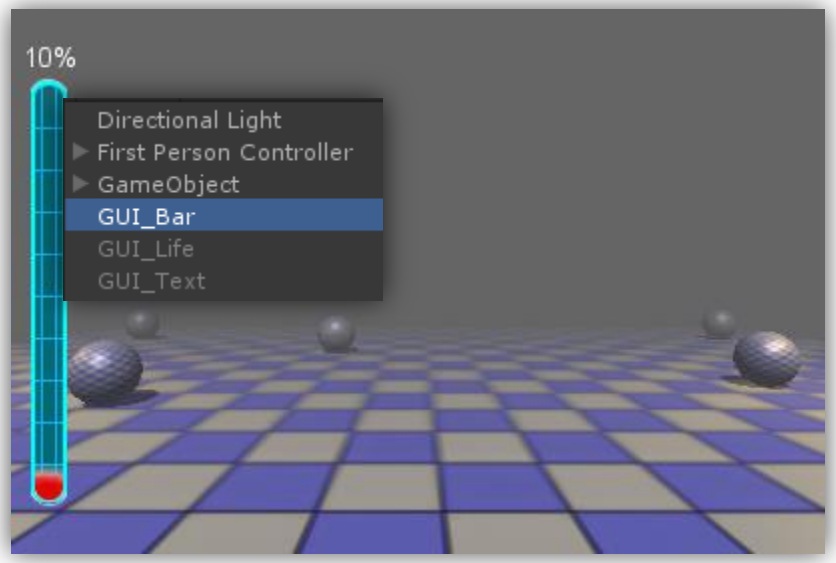
AI_Ball

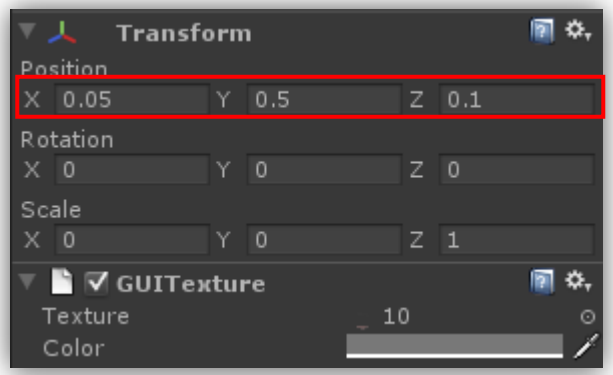
```
function OnTriggerEnter () {

        audio.Play ();
        Destroy (gameObject, 0.5);

        Vida += 10;

        print (Vida);

}
```

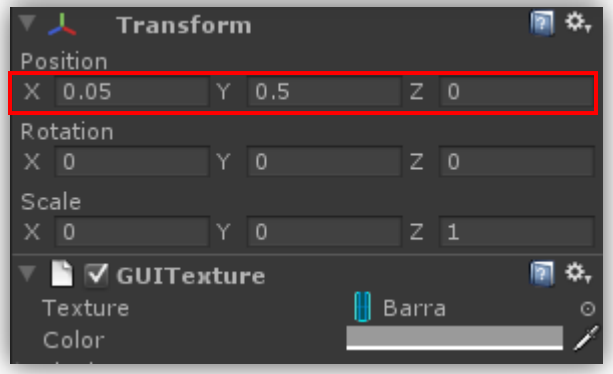
4. Creamos un GUI Texture "**GUI_Bar**", para cargar la **barra** donde se mostraran las texturas con los porcentajes.

5. Creamos un GUI Texture "**GUI_Life**", para cargar los imagenes de los **porcentajes** que se mostraran encima de la barra de energía.

6. Creamos un GUI Text "**GUI_Text**", para mostrar el **porcentaje** de lo que se lleva cargado de manera contextual de la barra de energía.



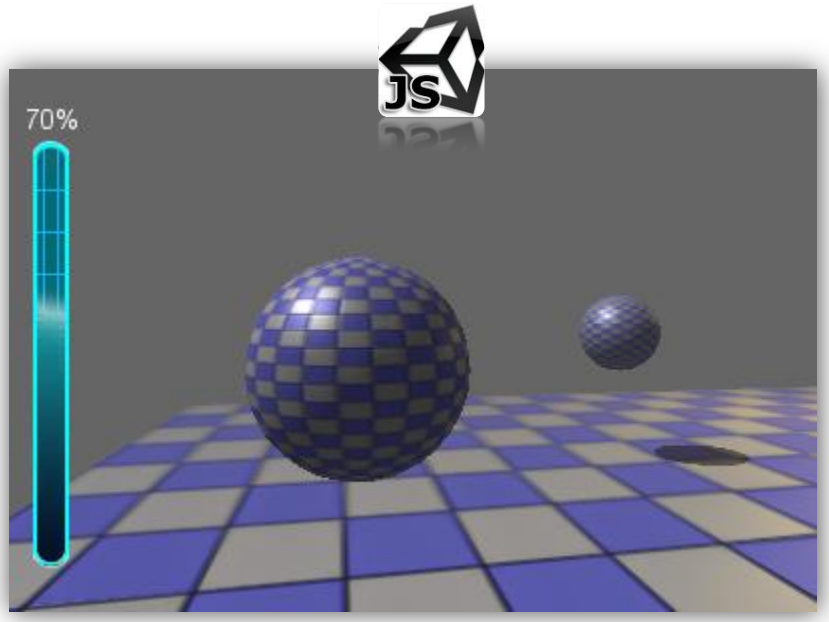




8. Agregamos al OnTrigerEnter la variable de "**Vida**", cada que colisiones, para que aumente de 10 en 10 en su valor Numerico y de Textura, asi de esta manera dara el efecto de que cambia el Porcentaje y la barra de vida va aumentando y cambia de colores.

AI_Ball

```
private GameObject GUI_Life;
private GameObject GUI_Text;

void Start () {
        GUI_Life = GameObject.Find ("GUI_Life");
        GUI_Text = GameObject.Find ("GUI_Text");
}

void Update () {
        if (Vida == 20){
                GUI_Life.GUI_Texture.texture = v20;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }else if (Vida == 30){
                GUI_Life.GUI_Texture.texture = v30;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if(Vida == 40){
                GUI_Life.GUI_Texture.texture = v40;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if(Vida == 50){
                GUI_Life.GUI_Texture.texture = v50;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if(Vida == 60){
                GUI_Life.GUI_Texture.texture = v60;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if (Vida == 70){
                GUI_Life.GUI_Texture.texture = v70;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if (Vida == 80){
                GUI_Life.GUI_Texture.texture = v80;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if (Vida == 90){
                GUI_Life.GUI_Texture.texture = v90;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
        else if(Vida == 100){
                GUI_Life.GUI_Texture.texture = v100;
                GUI_Text.GUI_Text.text = Vida.ToString() + "%";
        }
}
```



1.39 CHARACTER ANIMATION – LEGACY CROSSFADE

Character Animation – Legacy CrossFade (C# Script):

ANIMACIONES: FUNCIONES BÁSICAS PARA DE CONTROL DE REPRODUCCION.

Clip de Animación: Simple control de reproduccion de clips de animaciones de objetos o personajes.

```
// Funciones para el control de Animaciones Basicas:
animation.Play("Walk");                    //Reproducir © de Split Walk
animation.Stop("Walk");                    //Detener la animación del Split Walk y rebobinar.
animation.Rewind("Walk");                  //Regresar la animación del Split Walk

// Reproducir de animación con Espera de reproduccion:
animation.PlayQueued ("Walk", Queued.CompleteOthers);     -- SE USA MAS EN ANIMACIÓN DE GOLPES O PATADAS, ETC.
animation.PlayQueued ("Walk", Queued.PlayNow);            -- SE USA MAS EN ANIMACIÓN DE GOLPES O PATADAS, ETC.
```

Blending de Animación: Es cuando se realiza transiciones entre clips de animaciones en un personaje de manera profesional.

```
animation.CrossFade ("Walk");              //Fade al reproducir Walk con otros clips.
animation.CrossFade ("Walk", 0.2f);        //Fade al reproducir Walk con otros clips, con 2 centesimas de segundo.

// Reproducir de animación Suaves Espera de reproduccion:
animation.CrossFadeQueued ("Walk", Queued.CompleteOthers);       -- SE USA MAS EN ANIMACIÓN CORTAS COMO: GOLPES O PATADAS, ETC.
animation. CrossFadeQueued ("Walk", Queued.PlayNow);             -- SE USA MAS EN ANIMACIÓN CORTAS COMO: GOLPES O PATADAS, ETC.
```

Animacion Velocidad: Podemos modificar la velocidad de cualqueir animacino, y en negativo convierte la animacino atrás (reversa).

```
//Controla la animación Walk su velocidad = reproducción atrás (-1)/Adelante (1).
animation["Walk"].speed = 1 ó -1;
```

CHARACTER CONTROLLER/MOTOR: FUNCIONES PARA CONTROLAR COMPONENTES Y OPCIONES.

```
// Obtener los componentes de "CharacterMotor" y modificar velocidad de movimientos del player.
CharacterMotor Motor = GetComponent<CharacterMotor>();
Motor.movement.maxForwardSpeed = 1.0f;
Motor.movement.maxSidewaysSpeed = 1.0f;
Motor.movement.maxBackwardsSpeed = 1.0f;


// Obtener los componentes de "CharacterController" y saber cuando el personaje toca el piso.
CharacterController controller = GetComponent<CharacterController>();
// Cuando detecta que el personaje esta en el piso
if (!controller.isGrounded){
       //Aplicar animacion de Brinco en el aire, si no el de brincar.
}
```

1. Imporamos el paquete de Unity "**Legacy Animation Scene**" y la carpeta de "**Soldier"**.
2. Seleccionmos "**Soldier**" en **PROJECT**, en **RIG** (Legacy), **Animations BakeAnimation: On** y creamos los siguientes **Clips**:
   - **Idle**: 0 a 190, **Walk**: 193 a 224, **Run** 225 a 240 (**Loop), Jump:** 242 a 261 (**Once**).

3. Insertamos el personaje en el escenario, y en la cámara Smooth Follow cargamos Soldier para que lo siga.

4. Con el **Player** seleccionado agregaremos los siguientes scripts **Componen>Character> FPS InputController**:
   - **Character Controller**: permite agregar dinámicas al personaje para colisionar en la escena.
   - **Character Motor (Script):** agrega componentes para poder controlar el mover, saltar, desplazar, etc.
   - **FPS InputController**: El cual permite controlar por inputs los componentes

5. **Player: Componen>Camera-Control> Mouse Look:** Seleccionar **eje X,** el player rotara sobre el eje X del ratón.

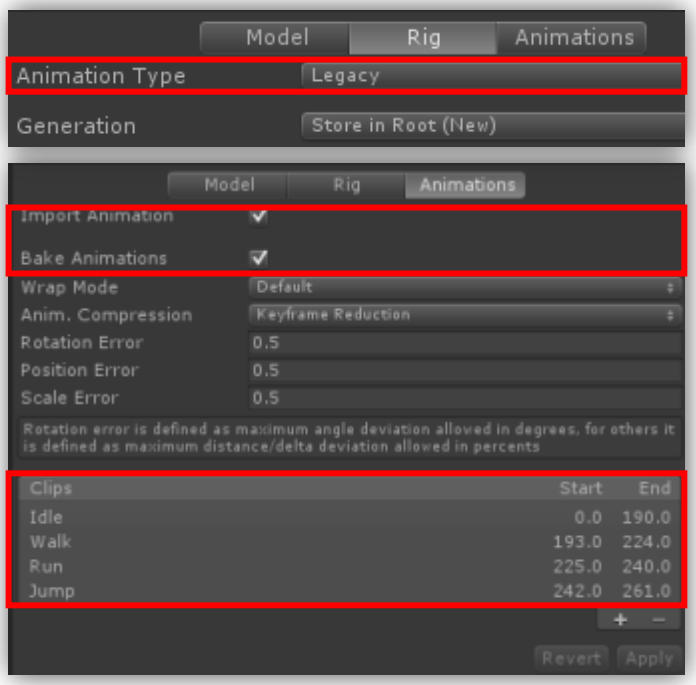
6. Creamos un Script "**AI_Character**" lo asignamos al "player" para crear el control de animaciones **Idle, Walk**:

AI_Character

```
void Update () {
      //CAMINAR ADELANTE.
      if (Input.GetAxis ("Vertical") > 0.1f){
            animation.Play("Walk");
      }
}
```

7. Ahora creamos la animación cuando se suelte el boton cambie a **Idle**.

```
//IDLE AL NO CAMINAR.
else if (Input.GetAxis ("Vertical") < 0.1f) {
      animation.Play("Idle");
}
```

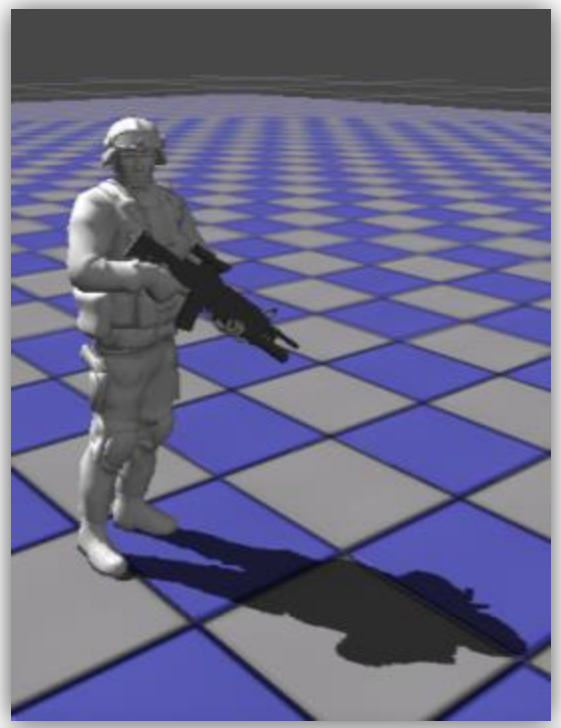


- Como podemos ver cuando cambia de animación, esta cambia de golpe al presionar o soltar el botón del teclado, para evitar esto haremos un **blending** ente las **animaciones** usando el **animation.CrossFade**.

8. Creación de **Blending** entre las animaciones, Se pueden hacer automáticas o nosotros definir el tiempo.

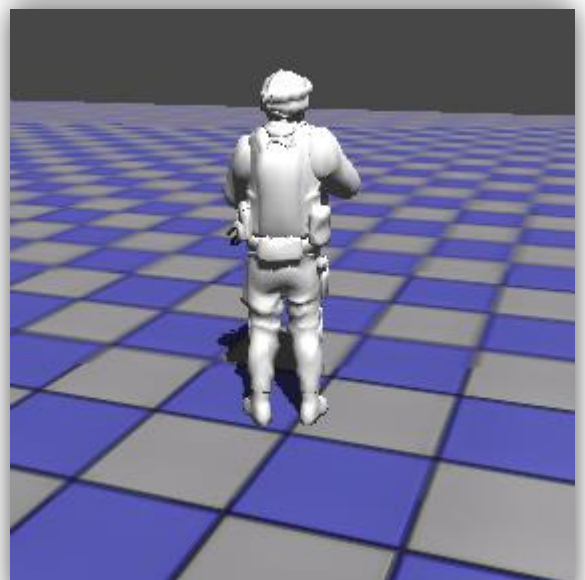
void **CrossFade (animation :** String, **fadeLength :** float **= 0.3F) :** void

```
void FixUpdate () {
	//CAMINAR ADELANTE.
	if (Input.GetAxis ("Vertical") > 0.1f){
		animation.CrossFade("Walk", 0.2f);
	}
	//IDLE AL NO CAMINAR.
	else if (Input.GetAxis ("Vertical") < 0.1f){
		animation.CrossFade ("Idle", 0.1f);
	}
}
```



9. Ahora para regular la velocidad del personaje entrando al Character Motor en Movement de manera dinámica (abrir el Class de CharacterMotorMovement y ver los maxForward, Sideway y Backwards Speeds).

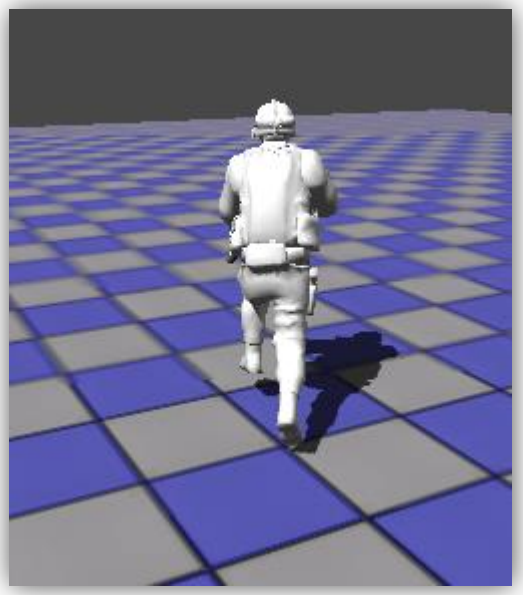
```
// Variables para controlar caminar y correr.
private CharacterMotor Motor;
public float vWalk = 1.5f;

void Start () {
	//Crear variable para obtener los componentes de character controller
	Motor = GetComponent <CharacterMotor>();
}

void FixedUpdate () {
	// CAMINAR ADELANTE.
	if (Input.GetAxis ("Vertical") > 0.1f){
		animation.CrossFade("Walk", 0.2f);

		//Modificar los valores para caminar de forma dinámica.
		Motor.movement.maxForwardSpeed = vWalk;
		Motor.movement.maxSidewaysSpeed = vWalk;
		Motor.movement.maxBackwardsSpeed = vWalk;
	}
	// IDLE AL NO CAMINAR.
	else if (Input.GetAxis ("Vertical") < 0.1f){
		animation.CrossFade ("Idle", 0.1f);
	}

}
```



10. Ahora agregamos un Input para usar la animación de correr: **Run** - "**shift left**".

AI_Character

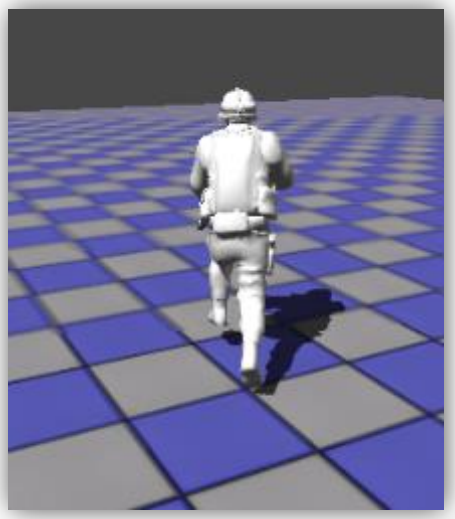
```
// Variables para controlar caminar y correr.
private CharacterMotor Motor;
public float vWalk = 1.0f;
public float vRun = 5.0f;

void FixedUpdate () {

	//Crear variable para obtener los componentes de character controller
	CharacterController controllers = GetComponent<CharacterController>();

	//CAMINAR ADELANTE
	if (Input.GetAxis ("Vertical") > 0.1f){
		animation.CrossFade("Walk", 0.2f);

		//Modificar los valores para caminar de forma dinámica.
		Motor.movement.maxForwardSpeed = vWalk;
		Motor.movement.maxSidewaysSpeed = vWalk;
		Motor.movement.maxBackwardsSpeed = vWalk;
	}

	// IDLE AL NO CAMINAR.
	else if (Input.GetAxis ("Vertical") < 0.1f){
		animation. CrossFade ("Idle", 0.1f);
	}


	//-------------------------------------------------------------

	//CORRER ADELANTE
	if (Input.GetAxis ("Vertical") > 0.1f  && Input.GetKey (KeyCode.LeftShift) ){
		animation.CrossFade("Run", 0.2f);

		//Modificar los valores para caminar de forma dinámica.
		Motor.movement.maxForwardSpeed = vRun;
		Motor.movement.maxSidewaysSpeed = vRun;
		Motor.movement.maxBackwardsSpeed = vRun;
	}
}
```

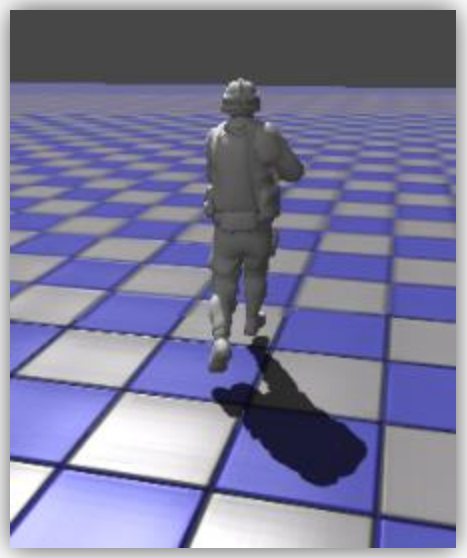


11. Ahora para poder hacer que **camine/corra hacia atrás** con el **Input "Vertical" de tecla negativa y el GetAxis (-1 a 1)**, se **rehusar** la animación de "**Walk**" (adelante) para hacer que vaya hacia atrás por el componente de **Speed**:

AI_Character

```
// Variables para controlar caminar y correr.
private CharacterMotor Motor;
public float vWalk = 1.0f;
public float vRun = 5.0f;

function FixedUpdate () {
        //Crear variable para obtener los componentes de character controller
        CharacterMotor Motor = GetComponent<CharacterMotor<();

        //CAMINAR ADELANTE (Positivo 1).
        if (Input.GetAxis ("Vertical") > 0.1f){
                animation["Walk"].speed = 1;
                animation.CrossFade("Walk", 0.2f);
                //Modificar los valores para caminar de forma dinámica.
                Motor.movement.maxForwardSpeed = vWalk;
                Motor.movement.maxSidewaysSpeed = vWalk;
                Motor.movement.maxBackwardsSpeed = vWalk;
        }
        //CAMINAR ATRAS (Negativo -1).
        else if (Input.GetAxis ("Vertical") < -0.1f) {
                animation["Walk"].speed =  -1;
                animation.CrossFade("Walk", 0.2f);

                //Modificar los valores para caminar de forma dinámica.
                Motor.movement.maxForwardSpeed = vWalk;
                Motor.movement.maxSidewaysSpeed = vWalk;
                Motor.movement.maxBackwardsSpeed = vWalk;
        }
        //-------------------------------------------------------------
        // IDLE AL NO CAMINAR.
        else if (Input.GetAxis ("Vertical") < 0.1f){
                animation.CrossFade ("Idle", 0.1f);
        }
        //-------------------------------------------------------------
        //CORRER ADELANTE (Positivo).
        if (Input.GetAxis ("Vertical") > 0.1f   && Input.GetKey (KeyCode.LeftShift)){
                animation["Run "].speed = 1;
                animation.CrossFade("Run", 0.2f);
                //Modificar los valores para caminar de forma dinámica.
                Motor.movement.maxForwardSpeed = vRun;
                Motor.movement.maxSidewaysSpeed = vRun;
                Motor.movement.maxBackwardsSpeed = vRun;
        }
        //CORRER ATRÁS (Negativo -1).
        if (Input.GetAxis ("Vertical") < -0.1f && Input.GetKey (KeyCode.LeftShift)){
                animation["Run "].speed =  -1;
                animation.CrossFade("Run", 0.2f);

                //Modificar los valores para caminar de forma dinámica.
                Motor.movement.maxForwardSpeed = vRun;
                Motor.movement.maxSidewaysSpeed = vRun;
                Motor.movement.maxBackwardsSpeed = vRun;
        }
}
```

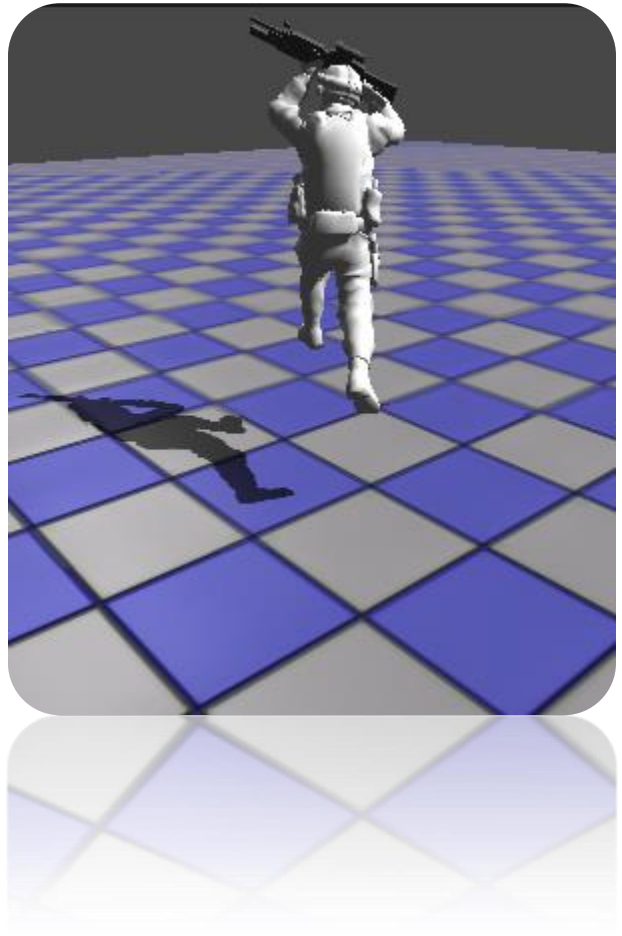




12. Por último cuando el personaje esta **saltando y queda en el aire**, para que **no** haga otra animación más que la del salto, usamos el componente del "**Character Controller**", para saber cuando toca el piso, y cuando no lo toque cambiar a la animación de **Jump**:

AI_Character

```
// Variables para controlar caminar y correr.
private CharacterMotor Motor;
public float vWalk = 1.0f;
public float vRun = 5.0f;

function FixedUpdate () {
        //Crear variable para obtener los componentes de character controller
        CharacterController controllers = GetComponent<CharacterController>();

        if (controllers.isGrounded){
                //Crear variable para obtener los componentes de characterMotor
                CharacterMotor Motor = GetComponent<CharacterMotor>();

                //CAMINAR ADELANTE (Positivo 1).
                if (Input.GetAxis ("Vertical") > 0.1f){
                        animation["Walk"].speed = 1;
                        animation.CrossFade("Walk", 0.2f);
                        //Modificar los valores para caminar de forma dinámica.
                        Motor.movement.maxForwardSpeed = vWalk;
                        Motor.movement.maxSidewaysSpeed = vWalk;
                        Motor.movement.maxBackwardsSpeed = vWalk;
                }
                //CAMINAR ATRAS (Negativo -1).
                else if (Input.GetAxis ("Vertical") < -0.1f) {
                        animation["Walk"].speed =  -1;
                        animation.CrossFade("Walk", 0.2f);
                        //Modificar los valores para caminar de forma dinámica.
                        Motor.movement.maxForwardSpeed = vWalk;
                        Motor.movement.maxSidewaysSpeed = vWalk;
                        Motor.movement.maxBackwardsSpeed = vWalk;
                }
                //-------------------------------------------------------------
                // IDLE AL NO CAMINAR.
                else if (Input.GetAxis ("Vertical") < 0.1f){
                        animation.CrossFade ("Idle", 0.1f);
                }
                //-------------------------------------------------------------
                //CORRER ADELANTE (Positivo).
                if (Input.GetAxis ("Vertical") > 0.1f   && Input.GetKey (KeyCode.LeftShift)){
                        animation["Run"].speed = 1;
                        animation.CrossFade("Run", 0.2f);
                        //Modificar los valores para caminar de forma dinámica.
                        Motor.movement.maxForwardSpeed = vRun;
                        Motor.movement.maxSidewaysSpeed = vRun;
                        Motor.movement.maxBackwardsSpeed = vRun;
                }
                //CORRER ATRÁS (Negativo -1).
                if (Input.GetAxis ("Vertical") < -0.1f && Input.GetKey (KeyCode.LeftShift)){
                        animation["Run"].speed = -1;
                        animation.CrossFade("Run", 0.2f);
                        //Modificar los valores para caminar de forma dinámica.
                        Motor.movement.maxForwardSpeed = vRun;
                        Motor.movement.maxSidewaysSpeed = vRun;
                        Motor.movement.maxBackwardsSpeed = vRun;
                }

        }else {
                animation.CrossFade ("Jump", 0.1f);
        }
}
```



Character Animation – Legacy CrossFade (Java Script):

ANIMACIONES: FUNCIONES BÁSICAS PARA DE CONTROL DE REPRODUCCION.

Clip de Animación: Simple control de reproduccion de clips de animaciones de objetos o personajes.

```
// Funciones para el control de Animaciones Basicas:
animation.Play("Walk");                    //Reproducir © de Split Walk
animation.Stop("Walk");                    //Detener la animación del Split Walk y rebobinar.
animation.Rewind("Walk");                  //Regresar la animación del Split Walk

// Reproducir de animación con Espera de reproduccion:
animation.PlayQueued ("Walk", Queued.CompleteOthers);     -- SE USA MAS EN ANIMACIÓN DE GOLPES O PATADAS, ETC.
animation.PlayQueued ("Walk", Queued.PlayNow);            -- SE USA MAS EN ANIMACIÓN DE GOLPES O PATADAS, ETC.
```

Blending de Animación: Es cuando se realiza transiciones entre clips de animaciones en un personaje de manera profesional.

```
animation.CrossFade ("Walk");              //Fade al reproducir Walk con otros clips.
animation.CrossFade ("Walk", 0.2f);        //Fade al reproducir Walk con otros clips, con 2 centesimas de segundo.

// Reproducir de animación Suaves Espera de reproduccion:
animation.CrossFadeQueued ("Walk", Queued.CompleteOthers);       -- SE USA MAS EN ANIMACIÓN CORTAS COMO: GOLPES O PATADAS, ETC.
animation. CrossFadeQueued ("Walk", Queued.PlayNow);             -- SE USA MAS EN ANIMACIÓN CORTAS COMO: GOLPES O PATADAS, ETC.
```

Animacion Velocidad: Podemos modificar la velocidad de cualqueir animacino, y en negativo convierte la animacino atrás (reversa).

```
//Controla la animación Walk su velocidad = reproducción atrás (-1)/Adelante (1).
animation["Walk"].speed = 1 ó -1;
```

CHARACTER CONTROLLER/MOTOR: FUNCIONES PARA CONTROLAR COMPONENTES Y OPCIONES.

```
// Obtener los componentes de "CharacterMotor" y modificar velocidad de movimientos del player.
CharacterMotor Motor = GetComponent(CharacterMotor);
Motor.movement.maxForwardSpeed = 1.0f;
Motor.movement.maxSidewaysSpeed = 1.0f;
Motor.movement.maxBackwardsSpeed = 1.0f;

// Obtener los componentes de "CharacterController" y saber cuando el personaje toca el piso.
CharacterController controller = GetComponent (CharacterController);
// Cuando detecta que el personaje esta en el piso
if (!controller.isGrounded){
       //Aplicar animacion de Brinco en el aire, si no el de brincar.
}
```

1. Imporamos el paquete de Unity "**Legacy Animation Scene**" y la carpeta de "**Soldier"**.
2. Seleccionmos "**Soldier**" en **PROJECT**, en **RIG** (Legacy), **Animations BakeAnimation: On** y creamos los siguientes **Clips**:
   a. **Idle**: 0 a 190, **Walk**: 193 a 224, **Run** 225 a 240 (**Loop), Jump:** 242 a 261 (**Once**).

3. Insertamos el personaje en el escenario, y en la cámara Smooth Follow cargamos Soldier para que lo siga.

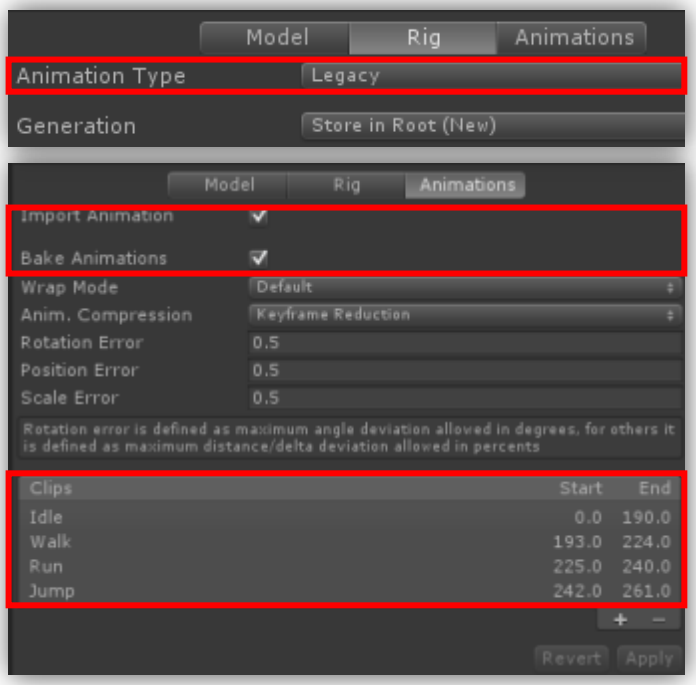
4. Con el **Player** seleccionado agregaremos los siguientes scripts **Componen>Character> FPS InputController**:
   a. **Character Controller**: permite agregar dinámicas al personaje para colisionar en la escena.
   b. Character Motor (Script): agrega componentes para poder controlar el mover, saltar, desplazar, etc.
   c. FPS InputController: El cual permite controlar por inputs los componentes

5. **Player: Componen>Camera-Control> Mouse Look:** Seleccionar **eje X,** el player rotara sobre el eje X del ratón.

6. Creamos un Script "**AI_Character**" lo asignamos al "player" para crear el control de animaciones **Idle, Walk**:

AI_Character

```
function Update () {
	//CAMINAR ADELANTE.
	if (Input.GetAxis ("Vertical") > 0.1){
		animation.Play("Walk");
	}
}
```



7. Ahora creamos la animación cuando se suelte el boton cambie a **Idle**.

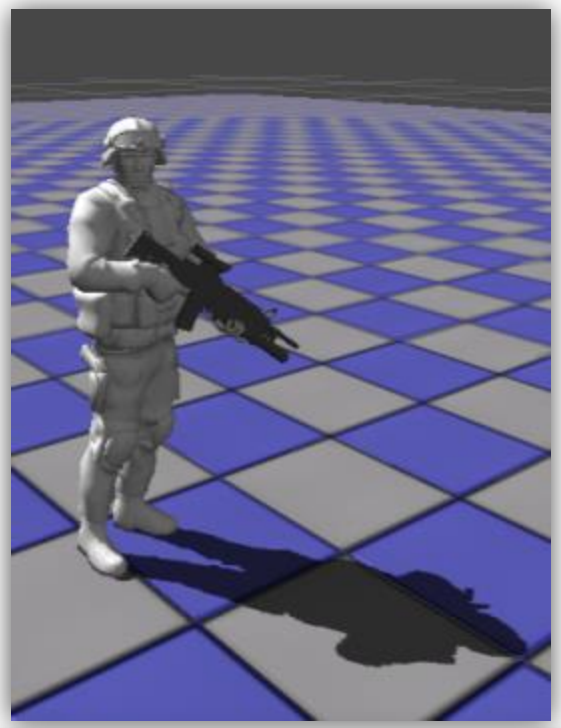
```
//IDLE AL NO CAMINAR.
else if (Input.GetAxis ("Vertical") < 0.1) {
	animation.Play("Idle");
}
```

- Como podemos ver cuando cambia de animación, esta cambia de golpe al presionar o soltar el botón del teclado, para evitar esto haremos un **blending** ente las **animaciones** usando el **animation.CrossFade**.

8. Creación de **Blending** entre las animaciones, Se pueden hacer automáticas o nosotros definir el tiempo.

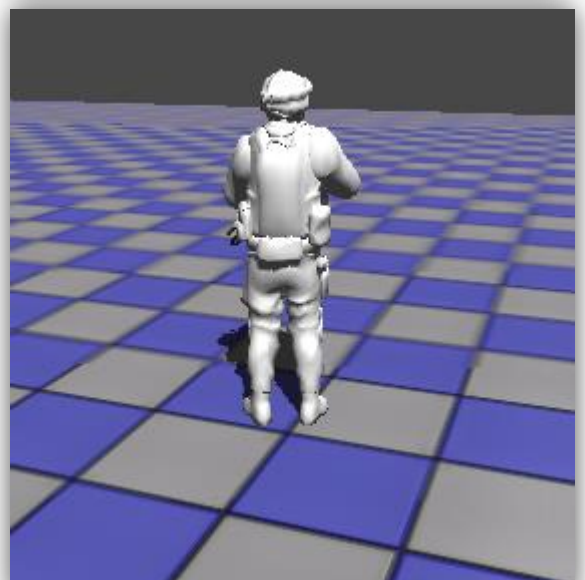
function **CrossFade (animation :** String, **fadeLength :** float **= 0.3F) :** function

```
function FixUpdate () {
	//CAMINAR ADELANTE.
	if (Input.GetAxis ("Vertical") > 0.1f){
		animation.CrossFade("Walk", 0.2f);
	}
	//IDLE AL NO CAMINAR.
	else if (Input.GetAxis ("Vertical") < 0.1f){
		animation.CrossFade ("Idle", 0.1f);
	}
}
```



9. Ahora para regular la velocidad del personaje entrando al Character Motor en Movement de manera dinámica (abrir el Class de CharacterMotorMovement y ver los maxForward, Sideway y Backwards Speeds).

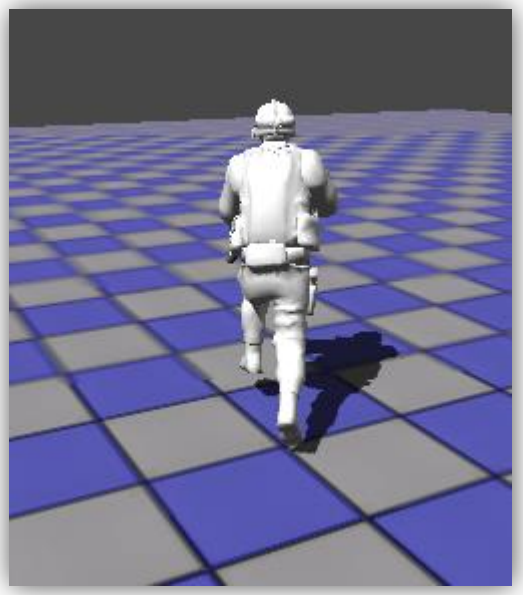
```
// Variables para controlar caminar y correr.
Private var Motor : CharacterMotor;
Public var vWalk : float = 1.5f;

function Start () {
	//Crear variable para obtener los componentes de character controller
	Motor = GetComponent (CharacterMotor);
}

function FixedUpdate () {
	// CAMINAR ADELANTE.
	if (Input.GetAxis ("Vertical") > 0.1f){
		animation.CrossFade("Walk", 0.2f);

		//Modificar los valores para caminar de forma dinámica.
		Motor.movement.maxForwardSpeed = vWalk;
		Motor.movement.maxSidewaysSpeed = vWalk;
		Motor.movement.maxBackwardsSpeed = vWalk;
	}
	// IDLE AL NO CAMINAR.
	else if (Input.GetAxis ("Vertical") < 0.1f){
		animation.CrossFade ("Idle", 0.1f);
	}

}
```



10. Ahora agregamos un Input para usar la animación de correr: **Run** - "**shift left**".

AI_Character

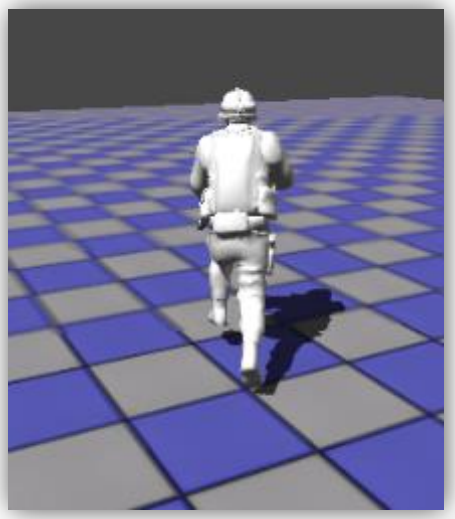
```
// Variables para controlar caminar y correr.
private var Motor : CharacterMotor;
public var vWalk : float = 1.0;
public var vRun : float = 5.0;

function FixedUpdate () {

	//Crear variable para obtener los componentes de character controller
	CharacterController controllers = GetComponent<CharacterController>();

	//CAMINAR ADELANTE
	if (Input.GetAxis ("Vertical") > 0.1){
		animation.CrossFade("Walk", 0.2);

		//Modificar los valores para caminar de forma dinámica.
		Motor.movement.maxForwardSpeed = vWalk;
		Motor.movement.maxSidewaysSpeed = vWalk;
		Motor.movement.maxBackwardsSpeed = vWalk;
	}

	// IDLE AL NO CAMINAR.
	else if (Input.GetAxis ("Vertical") < 0.1){
		animation. CrossFade ("Idle", 0.1);
	}


	//-------------------------------------------------------------

	//CORRER ADELANTE
	if (Input.GetAxis ("Vertical") > 0.1  && Input.GetKey (KeyCode.LeftShift) ){
		animation.CrossFade("Run", 0.2);

		//Modificar los valores para caminar de forma dinámica.
		Motor.movement.maxForwardSpeed = vRun;
		Motor.movement.maxSidewaysSpeed = vRun;
		Motor.movement.maxBackwardsSpeed = vRun;
	}
}
```

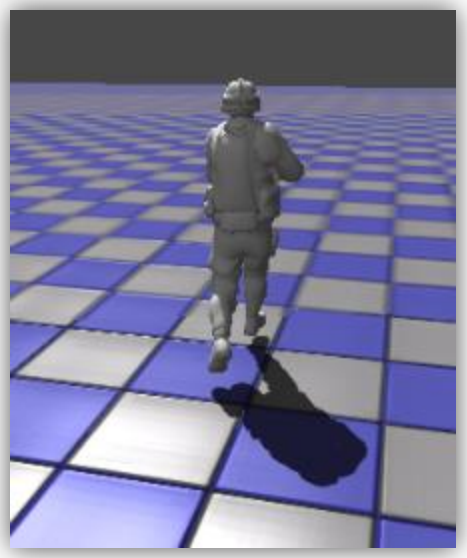


11. Ahora para poder hacer que **camine/corra hacia atrás** con el **Input "Vertical" de tecla negativa y el GetAxis (-1 a 1)**, se **rehusar** la animación de "**Walk**" (adelante) para hacer que vaya hacia atrás por el componente de **Speed**:

AI_Character

```
// Variables para controlar caminar y correr.
private var Motor : CharacterMotor;
public var vWalk : float = 1.0;
public var vRun : float = 5.0;

function FixedUpdate () {
        //Crear variable para obtener los componentes de character controller
        CharacterMotor Motor = GetComponent (CharacterMotor);

        //CAMINAR ADELANTE (Positivo 1).
        if (Input.GetAxis ("Vertical") > 0.1){
                animation["Walk"].speed = 1;
                animation.CrossFade("Walk", 0.2);
                //Modificar los valores para caminar de forma dinámica.
                Motor.movement.maxForwardSpeed = vWalk;
                Motor.movement.maxSidewaysSpeed = vWalk;
                Motor.movement.maxBackwardsSpeed = vWalk;
        }
        //CAMINAR ATRAS (Negativo -1).
        else if (Input.GetAxis ("Vertical") < -0.1) {
                animation["Walk"].speed =  -1;
                animation.CrossFade("Walk", 0.2);

                //Modificar los valores para caminar de forma dinámica.
                Motor.movement.maxForwardSpeed = vWalk;
                Motor.movement.maxSidewaysSpeed = vWalk;
                Motor.movement.maxBackwardsSpeed = vWalk;
        }
        //-------------------------------------------------------------
        // IDLE AL NO CAMINAR.
        else if (Input.GetAxis ("Vertical") < 0.1){
                animation.CrossFade ("Idle", 0.1);
        }
        //-------------------------------------------------------------
        //CORRER ADELANTE (Positivo).
        if (Input.GetAxis ("Vertical") > 0.1   && Input.GetKey (KeyCode.LeftShift)){
                animation["Run "].speed = 1;
                animation.CrossFade("Run", 0.2);
                //Modificar los valores para caminar de forma dinámica.
                Motor.movement.maxForwardSpeed = vRun;
                Motor.movement.maxSidewaysSpeed = vRun;
                Motor.movement.maxBackwardsSpeed = vRun;
        }
        //CORRER ATRÁS (Negativo -1).
        if (Input.GetAxis ("Vertical") < -0.1 && Input.GetKey (KeyCode.LeftShift)){
                animation["Run "].speed =  -1;
                animation.CrossFade("Run", 0.2);

                //Modificar los valores para caminar de forma dinámica.
                Motor.movement.maxForwardSpeed = vRun;
                Motor.movement.maxSidewaysSpeed = vRun;
                Motor.movement.maxBackwardsSpeed = vRun;
        }
}
```

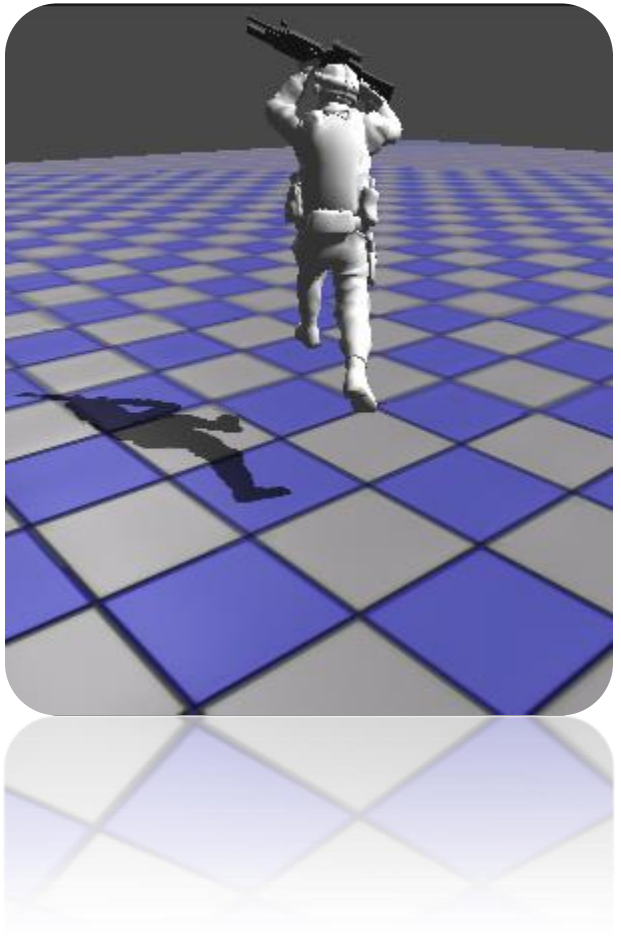




12. Por último cuando el personaje esta **saltando y queda en el aire**, para que **no** haga otra animación más que la del salto, usamos el componente del "**Character Controller**", para saber cuando toca el piso, y cuando no lo toque cambiar a la animación de **Jump**:

AI_Character

```
// Variables para controlar caminar y correr.
Private var Motor : CharacterMotor;
Public var vWalk : float = 1.0f;
Public var vRun : float = 5.0f;

function FixedUpdate () {
        //Crear variable para obtener los componentes de character controller
        CharacterController controllers = GetComponent<CharacterController>();

        if (controllers.isGrounded){
                //Crear variable para obtener los componentes de characterMotor
                CharacterMotor Motor = GetComponent<CharacterMotor>();

                //CAMINAR ADELANTE (Positivo 1).
                if (Input.GetAxis ("Vertical") > 0.1){
                        animation["Walk"].speed = 1;
                        animation.CrossFade("Walk", 0.2);
                        //Modificar los valores para caminar de forma dinámica.
                        Motor.movement.maxForwardSpeed = vWalk;
                        Motor.movement.maxSidewaysSpeed = vWalk;
                        Motor.movement.maxBackwardsSpeed = vWalk;
                }
                //CAMINAR ATRAS (Negativo -1).
                else if (Input.GetAxis ("Vertical") < -0.1) {
                        animation["Walk"].speed =  -1;
                        animation.CrossFade("Walk", 0.2);
                        //Modificar los valores para caminar de forma dinámica.
                        Motor.movement.maxForwardSpeed = vWalk;
                        Motor.movement.maxSidewaysSpeed = vWalk;
                        Motor.movement.maxBackwardsSpeed = vWalk;
                }
                //-------------------------------------------------------------
                // IDLE AL NO CAMINAR.
                else if (Input.GetAxis ("Vertical") < 0.0){
                        animation.CrossFade ("Idle", 0.1);
                }
                //-------------------------------------------------------------
                //CORRER ADELANTE (Positivo).
                if (Input.GetAxis ("Vertical") > 0.1   && Input.GetKey (KeyCode.LeftShift)){
                        animation["Run"].speed = 1;
                        animation.CrossFade("Run", 0.2);
                        //Modificar los valores para caminar de forma dinámica.
                        Motor.movement.maxForwardSpeed = vRun;
                        Motor.movement.maxSidewaysSpeed = vRun;
                        Motor.movement.maxBackwardsSpeed = vRun;
                }
                //CORRER ATRÁS (Negativo -1).
                if (Input.GetAxis ("Vertical") < -0.1 && Input.GetKey (KeyCode.LeftShift)){
                        animation["Run"].speed = -1;
                        animation.CrossFade("Run", 0.2);
                        //Modificar los valores para caminar de forma dinámica.
                        Motor.movement.maxForwardSpeed = vRun;
                        Motor.movement.maxSidewaysSpeed = vRun;
                        Motor.movement.maxBackwardsSpeed = vRun;
                }

        }else {
                animation.CrossFade ("Jump", 0.1);
        }
}
```



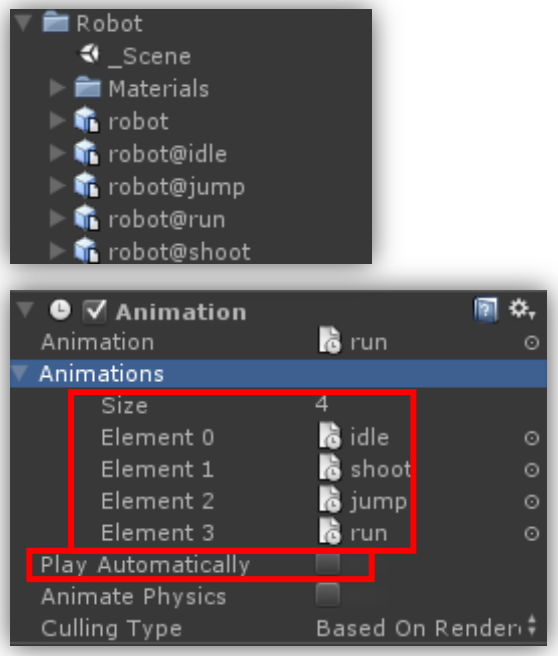
1.40 CHARACTER ANIMATIONS - LEGACY MIX & BLENDING

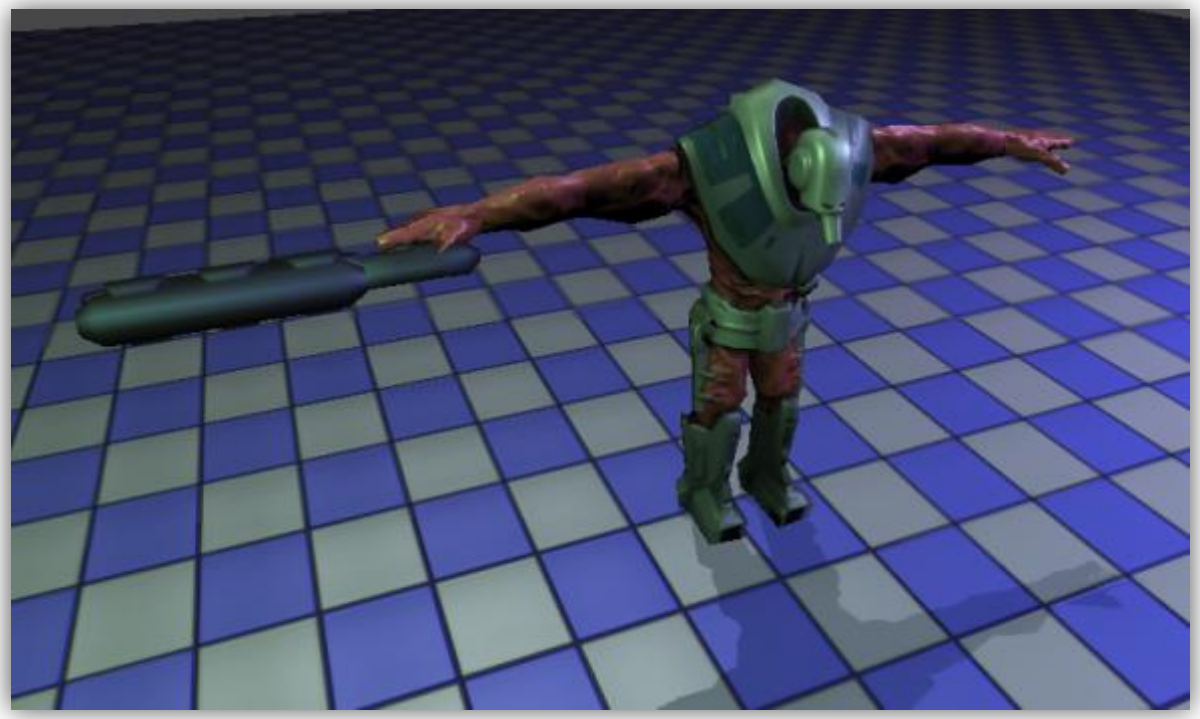
Character Animation – Legacy Mix & Blend (C# Script):

Unity puede ahorrar el trabajo de creación de clips de animación ya que permite mezclar clips de animaciones y esto nos ahorrar animaciones, lo cual se ve impactado en performance. La forma de mezclar animaciones lo más común es dividir de la cadera hacia arriba en una mezcla de clips, así de esta manera puede estar caminando, corriendo o en Idle, y podría estar disparando o haciendo algún movimiento con las manos.

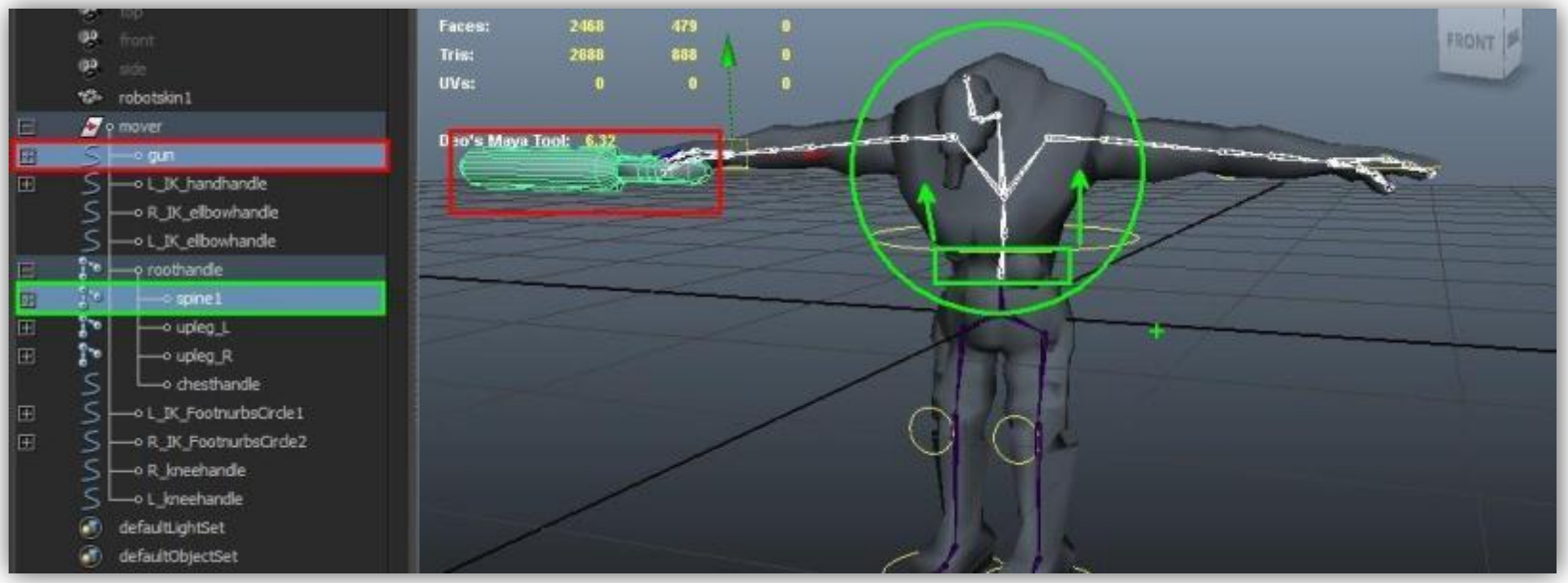
Importamos el paquete del tema: Mix Animations & Blend → "**Character_Start**".

1. En el Personaje dentro del componente de **Animation**, cargamos los clips de animaciones individuales (Idle, shoot, Jump, Run).
2. Activamos "**Play Automatically**" para que no inicie ninguna animación sobre el personaje.
3. Creamos un script para el personaje "**AI_MixAnimations**".





5. Si analizamos al personaje veremos que para controlar la mitad hacia arriba necesitamos saber el nombre del joint o ítems que tenga.



6. Ahora sobre el script creado iniciamos el proceso de insertar los clips, dentro de layers por importancia para poder mezclarlos.

AI_MixAnimations

```
void Update () {
	//Correr - Idle
	if ( Input.GetAxis("Vertical") > 0.1f) {
		animation.CrossFade("run", 0.2f);
		animation["run"].speed = Input.GetAxis("Vertical") + 1;
	}else {
		animation.CrossFade("idle", 0.2f);
	}
	//Brincar - Sin KeyDown para dejar presionado y activar animación
	if (Input.GetKey (KeyCode.Space) ){
		animation.CrossFade("jump", 0.2f);
	}
	// Disparar - Sin KeyDown para dejar presionado y activar animación
	if (Input.GetKey (KeyCode.Mouse0) ){
		animation.Play ("shoot");
	}
}
```

7. Ahora haremos control de clips por medio de inputs, para simular el personaje en movimiento y usando los clips de animación. La idea es que haya Fade entre las animaciones y aparte la mezcla de los clips de la mitad de la cadera hacia arriba, para ahorrar el crear esas animaciones por separado y tener que hacer también mas scripting.

AI_MixAnimations

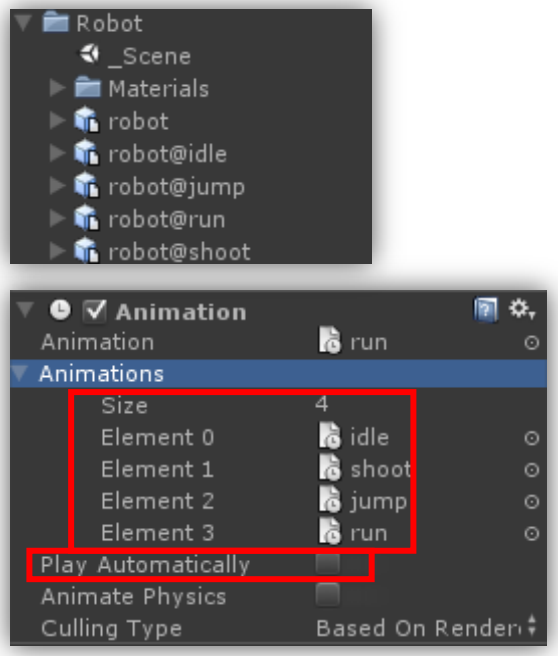
```
void Start () {
        // Hacer que todas los clips esten en modo Loop.
        animation.wrapMode = WrapMode.Loop;
        //Menos estos clips de animacion.
        animation["shoot"].wrapMode = WrapMode.Once;
        animation["jump"].wrapMode = WrapMode.Once;

        //Mezclar el "arma" y de la mitad hacia arriba del personaje "spine1"
        animation["shoot"].AddMixingTransform (transform.Find ("mover/gun") );
        animation["shoot"].AddMixingTransform (transform.Find ("mover/roothandle/spine1") );

        //Layers de Animacion Base para mezclar con otros layers.
        animation["run"].layer = 0;
        animation["idle"].layer = 0;
        //Layers de Mezcla para agregarlos sobre layer base.
        animation["shoot"].layer = 1;
        animation["jump"].layer = 1;

}

void Update () {
    // Clip de Correr con Fade.
    if ( Input.GetAxis("Vertical") > 0.1f) {
                animation.CrossFade("run", 0.2f);
                // Inicio Lento (0) con fade a velocidad de (1+1=2), porque es lenta la animación de Run.
                animation["run"].speed = Input.GetAxis("Vertical")+1;
    }else {
                //Al no presionar nada reproducir Idle Clip.
                animation.CrossFade("idle", 0.2f);
        }
        // Clip de Brincar con Fade.
    if (Input.GetKeyDown (KeyCode.Space) ){
                animation.CrossFade("jump", 0.2f);
        }
        // Shoot Clip al hacer click
    if (Input.GetKeyDown (KeyCode.Mouse0) ){
        //Reproducir la animacion con Fade, y permite reproducir aun que no termine la animación (recomendado par anim. cortas.
                animation.CrossFadeQueued("shoot", 0.15f, QueueMode.PlayNow);
        }

}
```

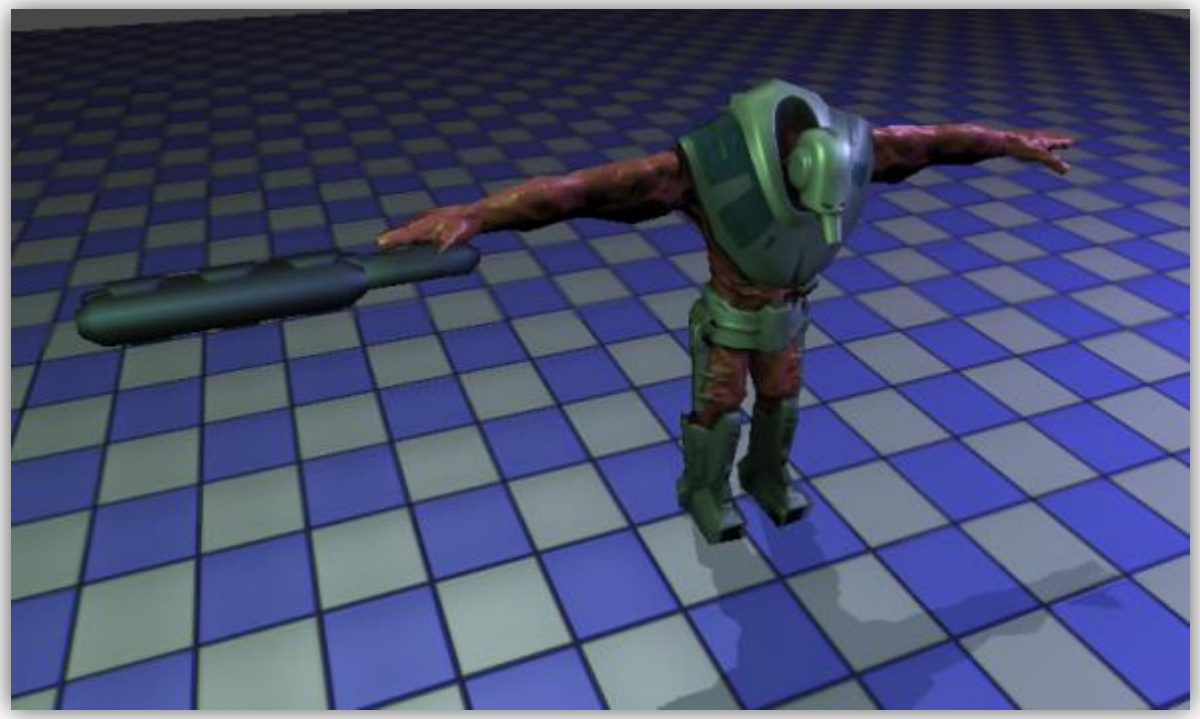
Character Animation – Legacy Mix & Blend (C# Script):

Unity puede ahorrar el trabajo de creación de clips de animación ya que permite mezclar clips de animaciones y esto nos ahorrar animaciones, lo cual se ve impactado en performance. La forma de mezclar animaciones lo más común es dividir de la cadera hacia arriba en una mezcla de clips, así de esta manera puede estar caminando, corriendo o en Idle, y podría estar disparando o haciendo algún movimiento con las manos.

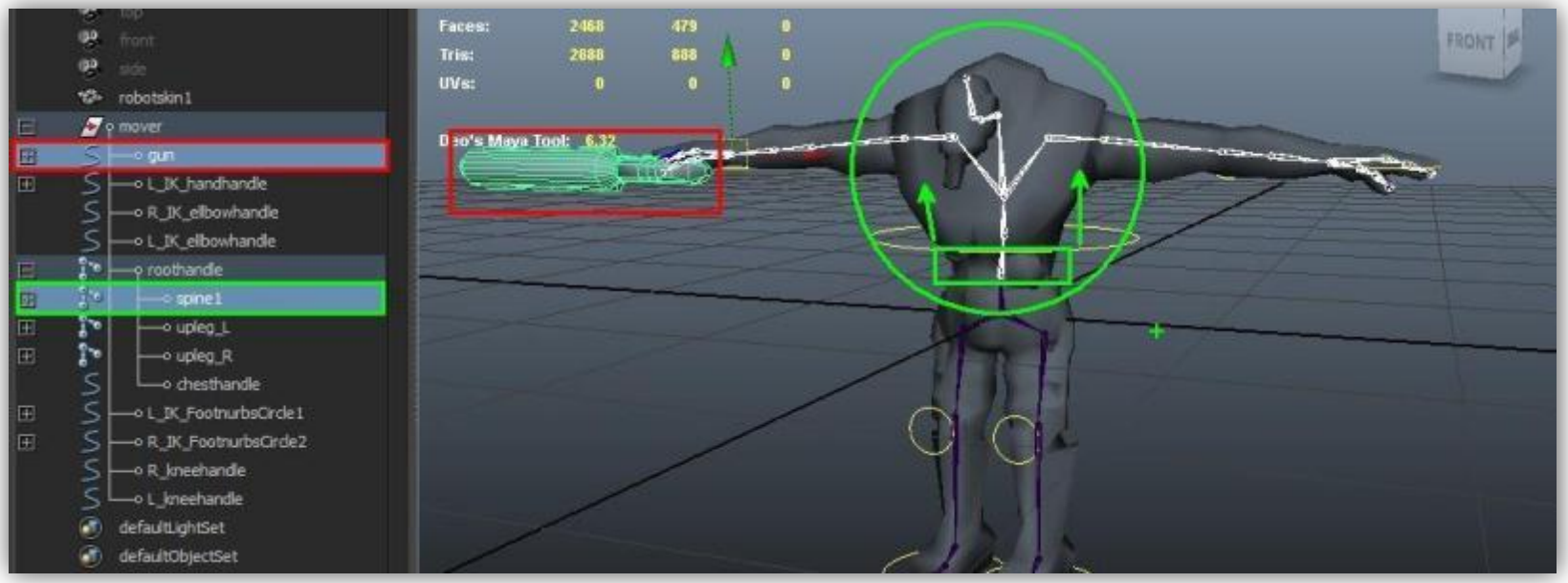
Importamos el paquete del tema: Mix Animations & Blend → "**Character_Start**".

1. En el Personaje dentro del componente de **Animation**, cargamos los clips de animaciones individuales (Idle, shoot, Jump, Run).
2. Activamos "**Play Automatically**" para que no inicie ninguna animación sobre el personaje.
3. Creamos un script para el personaje "**AI_MixAnimations**".





5. Si analizamos al personaje veremos que para controlar la mitad hacia arriba necesitamos saber el nombre del joint o ítems que tenga.



6. Ahora sobre el script creado iniciamos el proceso de insertar los clips, dentro de layers por importancia para poder mezclarlos.

AI_MixAnimations

```
void Update () {
	//Correr - Idle
	if ( Input.GetAxis("Vertical") > 0.1f) {
		animation.CrossFade("run", 0.2f);
		animation["run"].speed = Input.GetAxis("Vertical") + 1;
	}else {
		animation.CrossFade("idle", 0.2f);
	}
	//Brincar - Sin KeyDown para dejar presionado y activar animación
	if (Input.GetKey (KeyCode.Space) ){
		animation.CrossFade("jump", 0.2f);
	}
	// Disparar - Sin KeyDown para dejar presionado y activar animación
	if (Input.GetKey (KeyCode.Mouse0) ){
		animation.Play ("shoot");
	}
}
```

7. Ahora haremos control de clips por medio de inputs, para simular el personaje en movimiento y usando los clips de animación. La idea es que haya Fade entre las animaciones y aparte la mezcla de los clips de la mitad de la cadera hacia arriba, para ahorrar el crear esas animaciones por separado y tener que hacer también mas scripting.

AI_MixAnimations

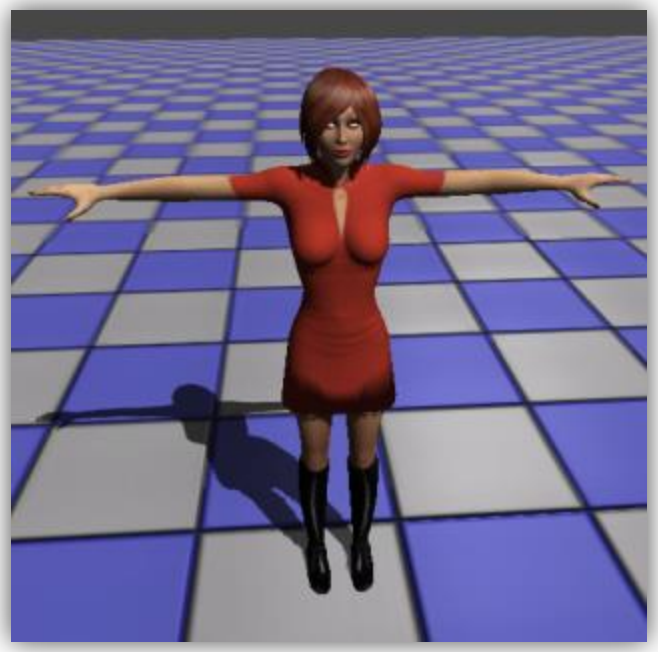
```
function Start () {
        // Hacer que todas los clips esten en modo Loop.
        animation.wrapMode = WrapMode.Loop;
        //Menos estos clips de animacion.
        animation["shoot"].wrapMode = WrapMode.Once;
        animation["jump"].wrapMode = WrapMode.Once;

        //Mezclar el "arma" y de la mitad hacia arriba del personaje "spine1"
        animation["shoot"].AddMixingTransform (transform.Find ("mover/gun") );
        animation["shoot"].AddMixingTransform (transform.Find ("mover/roothandle/spine1") );

        //Layers de Animacion Base para mezclar con otros layers.
        animation["run"].layer = 0;
        animation["idle"].layer = 0;
        //Layers de Mezcla para agregarlos sobre layer base.
        animation["shoot"].layer = 1;
        animation["jump"].layer = 1;

}
function Update () {

    // Clip de Correr con Fade.
    if ( Input.GetAxis("Vertical") > 0.1) {
                animation.CrossFade("run", 0.2);
                // Inicio Lento (0) con fade a velocidad de (1+1=2), porque es lenta la animación de Run.
                animation["run"].speed = Input.GetAxis("Vertical")+1;
    }else {
                //Al no presionar nada reproducir Idle Clip.
                animation.CrossFade("idle", 0.2);
        }
        // Clip de Brincar con Fade.
    if (Input.GetKeyDown (KeyCode.Space) ){
                animation.CrossFade("jump", 0.2);
        }
        // Shoot Clip al hacer click
    if (Input.GetKeyDown (KeyCode.Mouse0) ){
        //Reproducir la animacion con Fade, y permite reproducir aun que no termine la animación (recomendado par anim. cortas.
                animation.CrossFadeQueued("shoot", 0.15, QueueMode.PlayNow);
        }

}
```

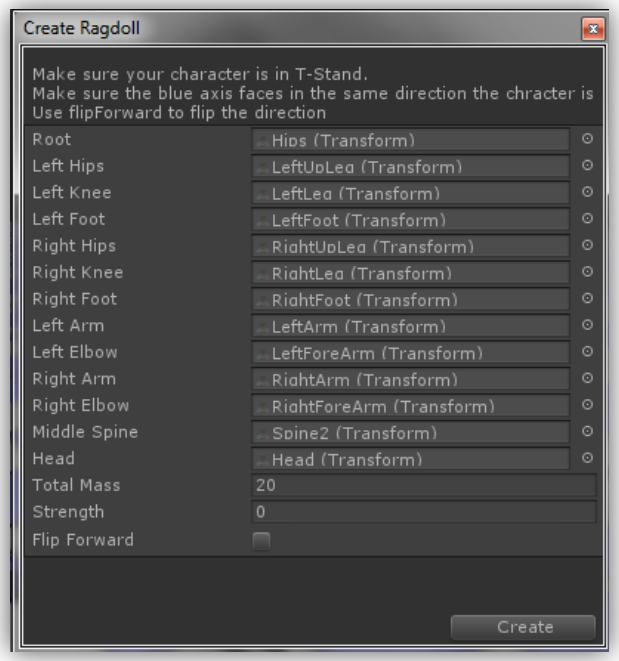
1.41 CHARACTER ANIMATIONS – LEGACY RAGDOLLS

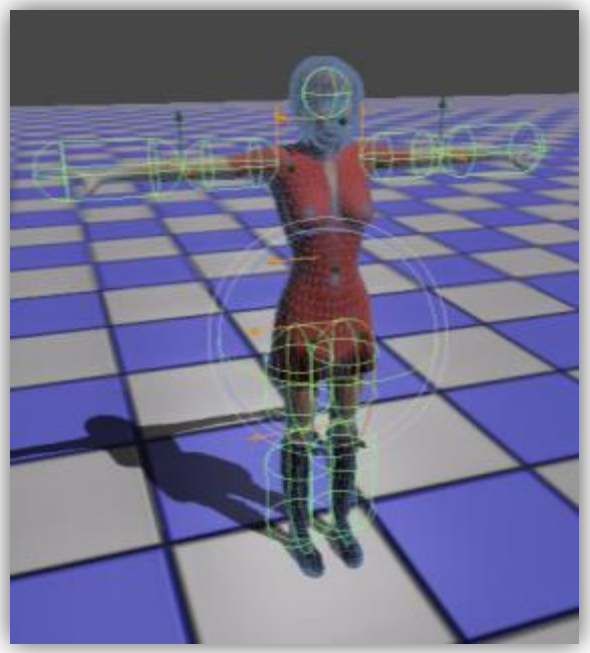
Character Animations – Legacy Ragdoll (C# Script):

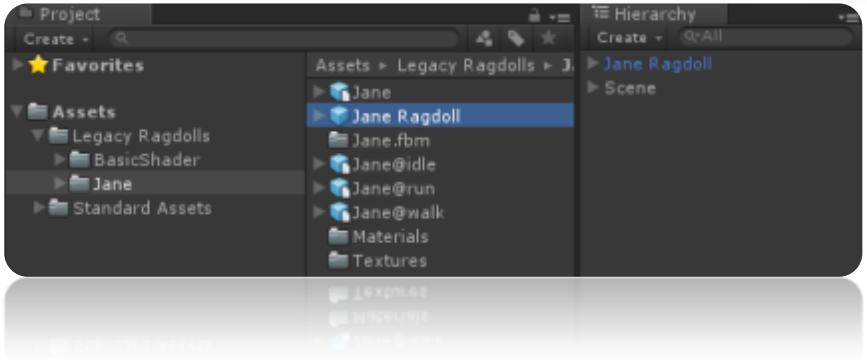
Para hacer que un personaje cuando muera se sustituya por un ragdoll, y este tenga la misma posición de rotación de cada uno de los joints se hace lo siguiente:

1. Cargamos el paquete "**Ragdoll_Scene**" y la carpeta de "**Jane**" que contiene el Mesh y las animaciones.
2. Insertamos el personaje "**Jane**" en la **escena** y creamos un **Ragdoll** de la mismo (**GameObject>Create Other>Ragdoll**).
3. Hacemos todo el **Re Mapeado** de los Joints de acuerdo a la **siguiente tabla** y le damos **Create**.
4. Hay que **Eliminar el Componente de "Animation**" para que solo quede el mesh como Ragdoll, sin componentes.
5. Al terminar el personaje Jane con Ragdoll lo meteremos en un **Prefab** y lo podemos eliminamos de la escena.

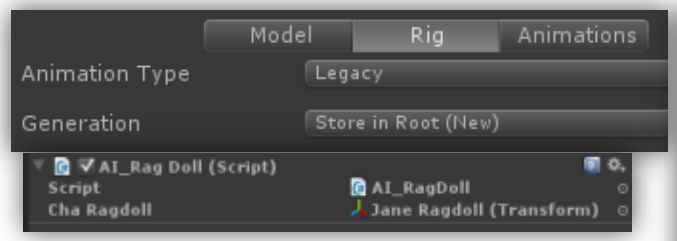


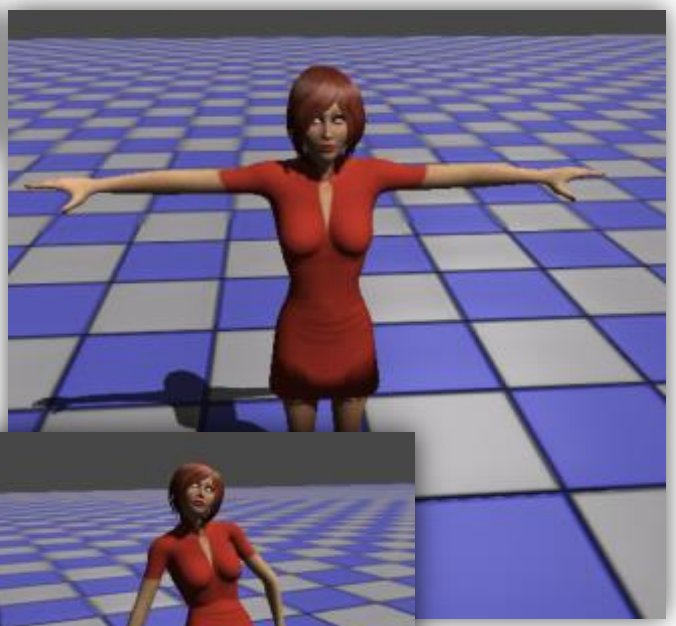






6. Insertamos de nuevo del Project a Jane el cual usaremos como personaje para controlar con Animaciones.
7. Seleccionamos al personaje y animaciones y en **Rig** lo dejamos como: **Legacy**, y activamos en **Loop** todas las animaciones.
8. Creamos un script "**AI_Ragdoll**" y lo asignamos al personaje Jane y cargamos la animación "**Run**":





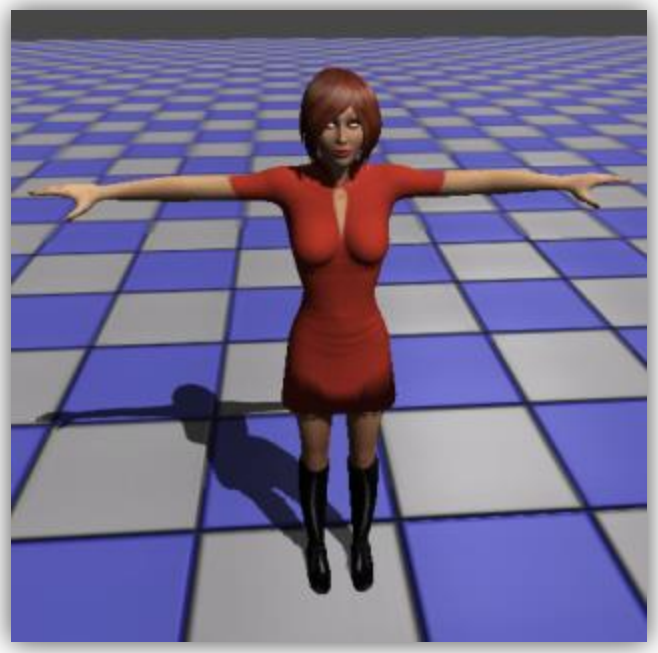
AI_Ragdoll

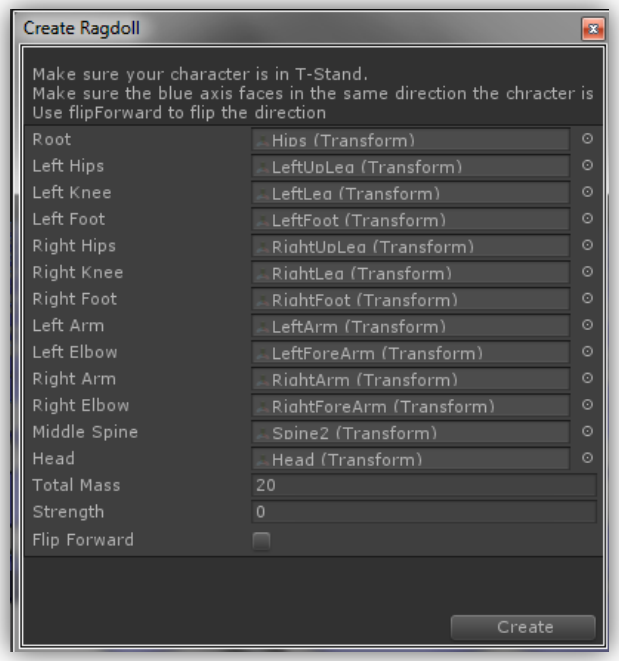
```
//Variable para cargar el RagDoll
public Transform vRagdoll;


void Update (){

    if (Input.GetKeyDown (KeyCode.Mouse0)) {
      // Instancia para cambiar por ragdoll
      Transform Inst_Ragdoll =
      Instantiate (vRagdoll, transform.position, transform.rotation) as Transform;

      // Ejecutar la función y mandarle la posición del ragdoll de cada Joint.
      ActiveRagdoll (transform, Inst_Ragdoll);

      //Destruir Mesh animado
      Destroy (gameObject);
    }

}


// Funcion que contiene la posición del personaje animado y la de ragdoll creado.
void ActiveRagdoll (Transform  JaneAnimated,  Transform JaneRagdoll) {
      // Obtener posición y rotación de cada uno de los Joints.
      JaneRagdoll.position = JaneAnimated.position;
      JaneRagdoll.rotation = JaneAnimated.rotation;

      // Igualar los valores de  "transform" de Jane Animado a Jane Ragdoll.
      foreach (Transform Joints in JaneRagdoll) {
            Transform JointTransform_Names = JaneAnimated.Find (Joints.name);
            //Activar de nuevo pero con la información de cada Joint
            ActiveRagdoll (JointTransform_Names, Joints);
      }

}
```

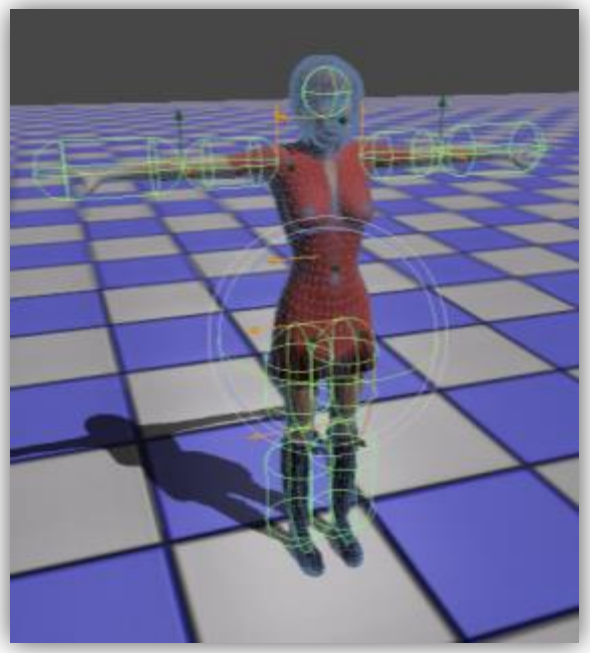
Character Animations – Legacy Ragdoll (Java Script):

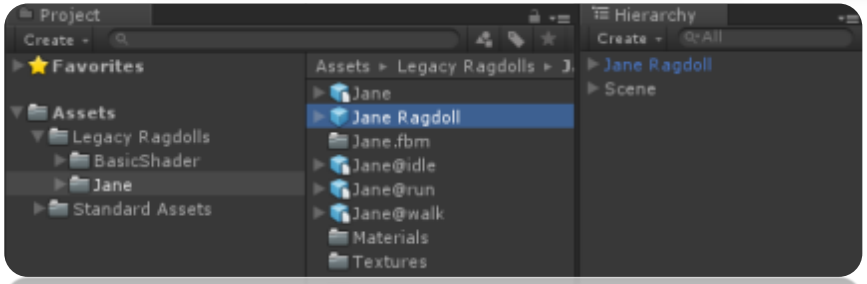
Para hacer que un personaje cuando muera se sustituya por un ragdoll, y este tenga la misma posición de rotación de cada uno de los joints se hace lo siguiente:

1. Cargamos el paquete "**Ragdoll_Scene**" y la carpeta de "**Jane**" que contiene el Mesh y las animaciones.
2. Insertamos el personaje "**Jane**" en la **escena** y creamos un **Ragdoll** de la mismo (**GameObject>Create Other>Ragdoll**).
3. Hacemos todo el **Re Mapeado** de los Joints de acuerdo a la **siguiente tabla** y le damos **Create**.
4. Hay que **Eliminar el Componente de "Animation**" para que solo quede el mesh como Ragdoll, sin componentes.
5. Al terminar el personaje Jane con Ragdoll lo meteremos en un **Prefab** y lo podemos eliminamos de la escena.

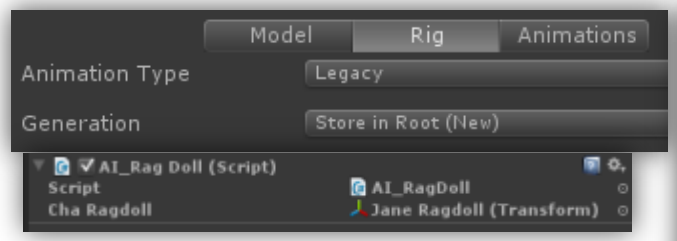


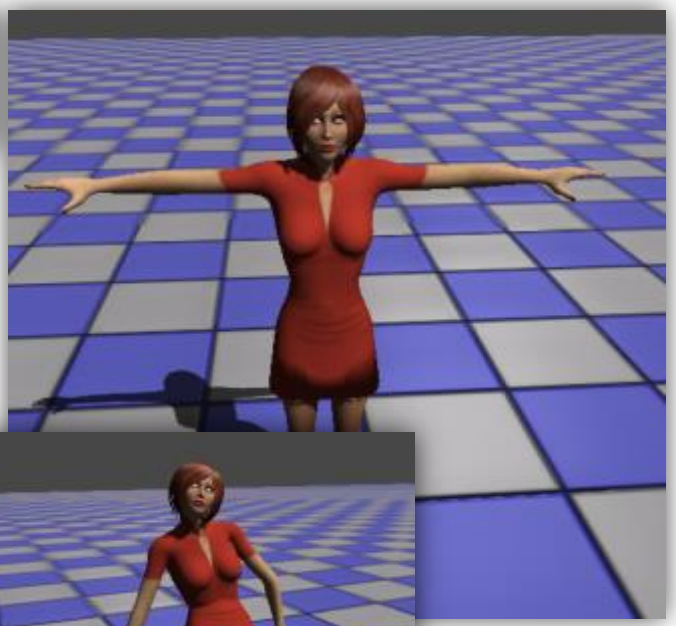






6. Insertamos de nuevo del Project a Jane el cual usaremos como personaje para controlar con Animaciones.
7. Seleccionamos al personaje y animaciones y en **Rig** lo dejamos como: **Legacy**, y activamos en **Loop** todas las animaciones.
8. Creamos un script "**AI_Ragdoll**" y lo asignamos al personaje Jane y cargamos la animación "**Run**":





AI_Ragdoll

```
//Variable para cargar el RagDoll
Var vRagdoll : Transform;


function Update (){

    if (Input.GetKeyDown (KeyCode.Mouse0)) {
      // Instancia para cambiar por ragdoll
      var Inst_Ragdoll : Transform =
      Instantiate (vRagdoll, transform.position, transform.rotation) as Transform;

      // Ejecutar la función y mandarle la posición del ragdoll de cada Joint.
      ActiveRagdoll (transform, Inst_Ragdoll);

      //Destruir Mesh animado
      Destroy (gameObject);
    }

}


// Funcion que contiene la posición del personaje animado y la de ragdoll creado.
Function  ActiveRagdoll (JaneAnimated : Transform,  JaneRagdoll : Transform) {
      // Obtener posición y rotación de cada uno de los Joints.
      JaneRagdoll.position = JaneAnimated.position;
      JaneRagdoll.rotation = JaneAnimated.rotation;

      // Igualar los valores de  "transform" de Jane Animado a Jane Ragdoll.
      foreach (Joints: Transform  in JaneRagdoll) {
            var JointTransform_Names : Transform = JaneAnimated.Find (Joints.name);
            //Activar de nuevo pero con la información de cada Joint
            ActiveRagdoll (JointTransform_Names, Joints);
      }

}
```

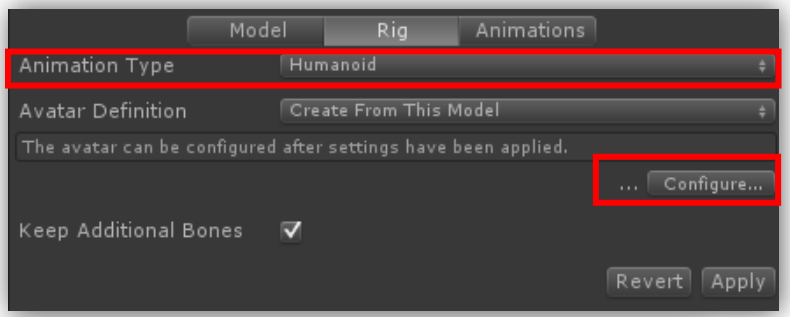
1.42 CHARACTER ANIMATIONS – MECANIMS.

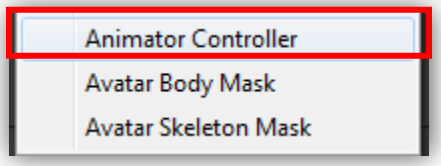
Mecanim – Character Animations System (C#/Java Script):

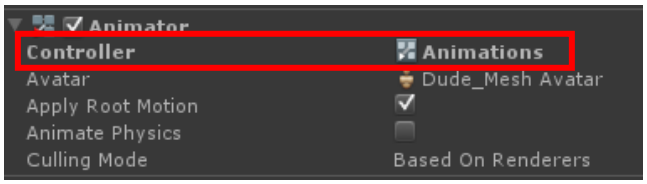
Unity tiene el sistema automatizado para control de Estados de Animacion para personajes con un editor grafico llamado **Mecanims**.

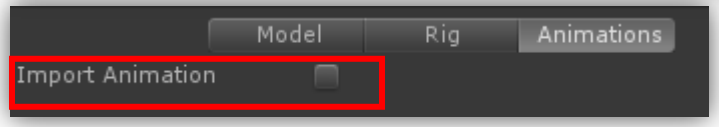
Mecanim es un sistema totalmente visual, para crear Transiciones esntre animaciones, árbol de control de multiples animaciones, herramientas especiales para animaciones por medio de Mocap, asi como Mezcla de animaciones por medio de layers.

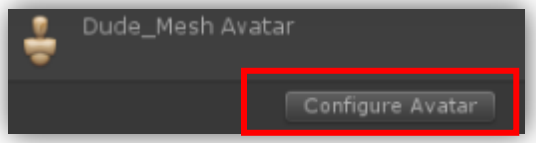
1. Importamos el paquete "**Mecanim Scene**".
2. En **Project>_Mocap** tenemos varias animaciones realizadas en MOCAP que usaremos para controlar a nuestro personaje.
3. Seleccionamos cada uno de las animaciones y en **el menú de RIG>Animation Type: Humanoid.**
4. En el **Mesh_Dude** Desactivamos "**Import Animations: Off**" y activamos la opciones de "…**Configure**".
5. Seleccionamos en "**Mapping**>**Automap**" y **Pose: Enforce TPose** (Esto es para hacer el remapeado si no queda bien de Manual)

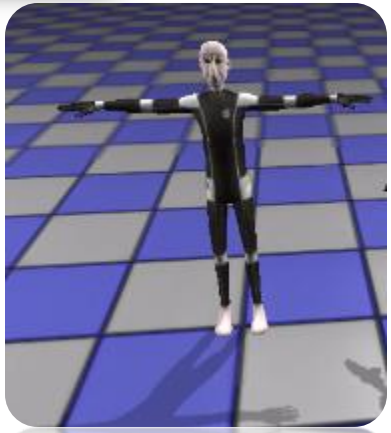


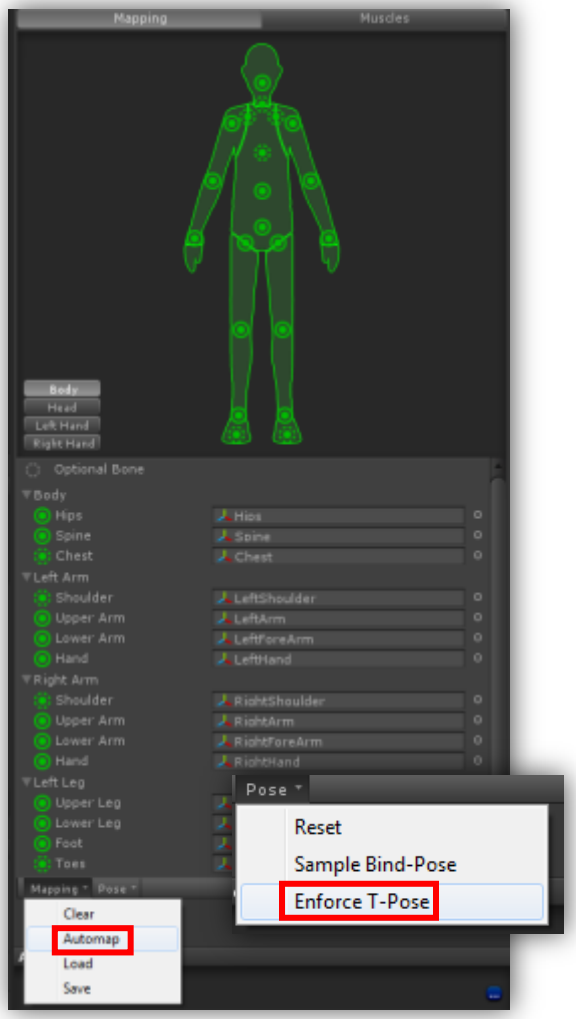




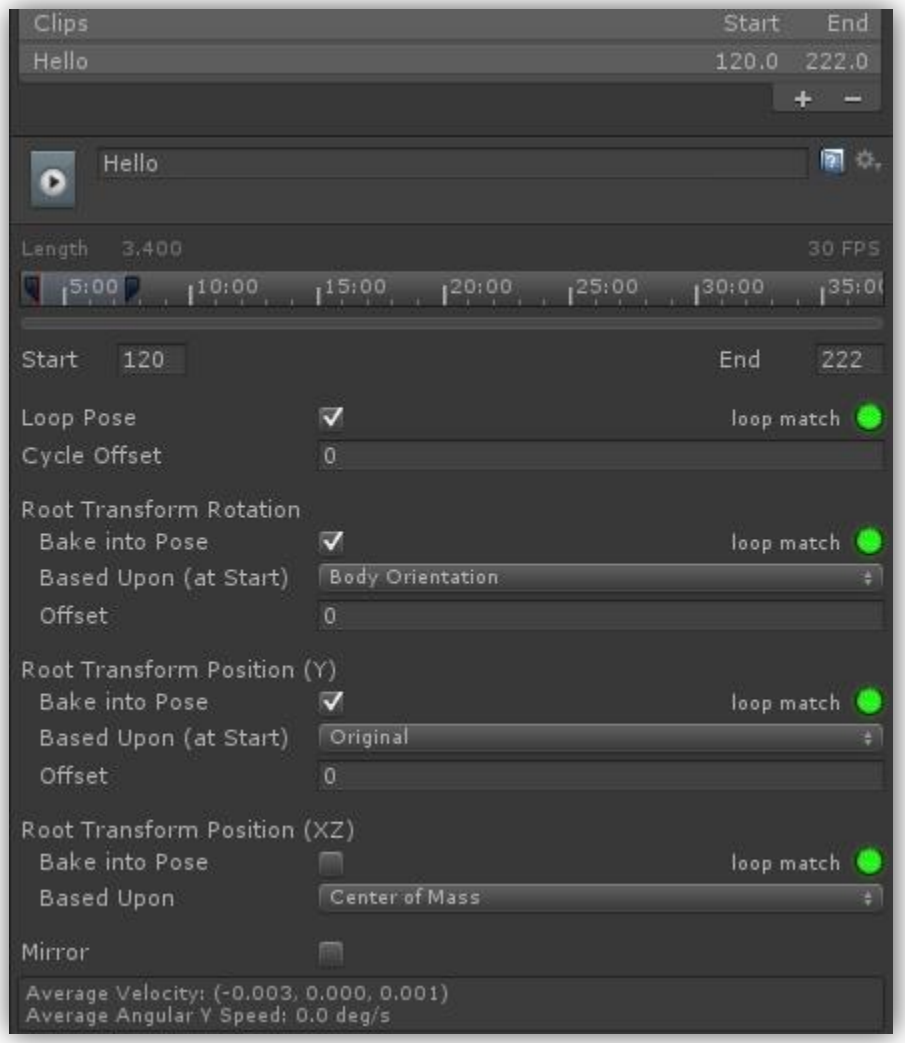


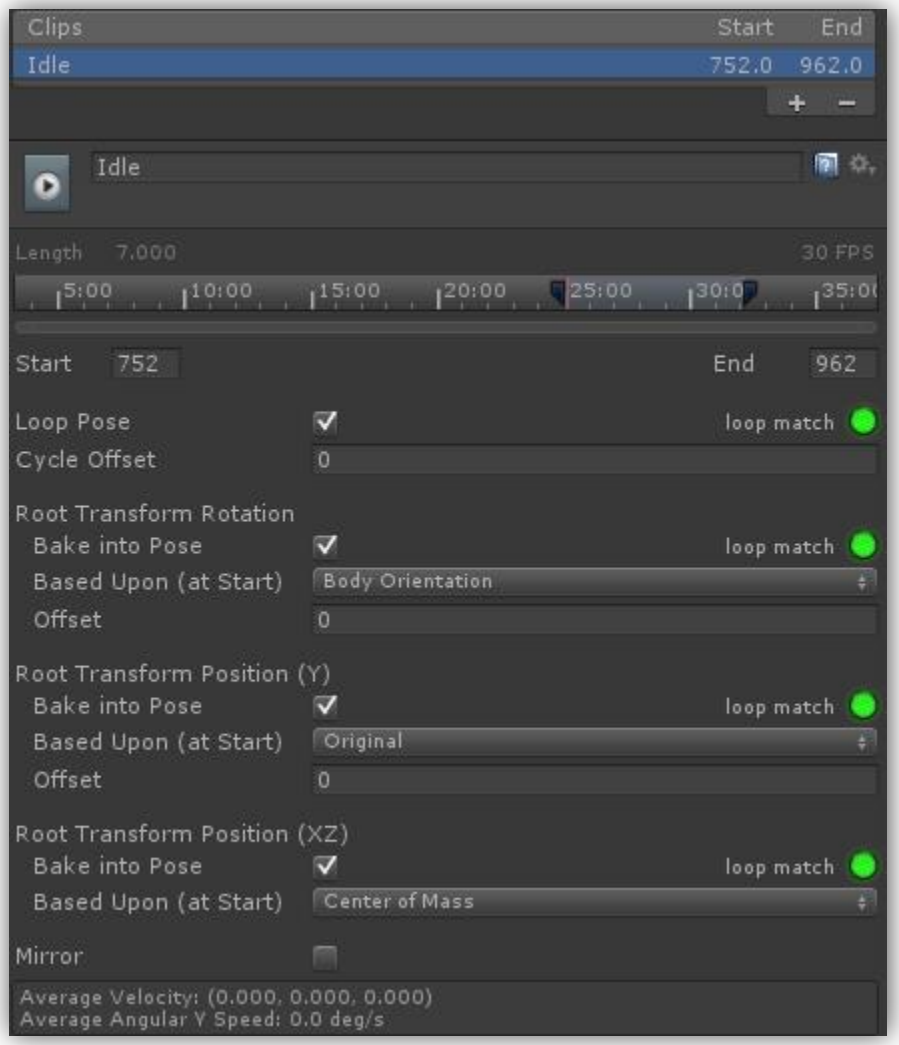


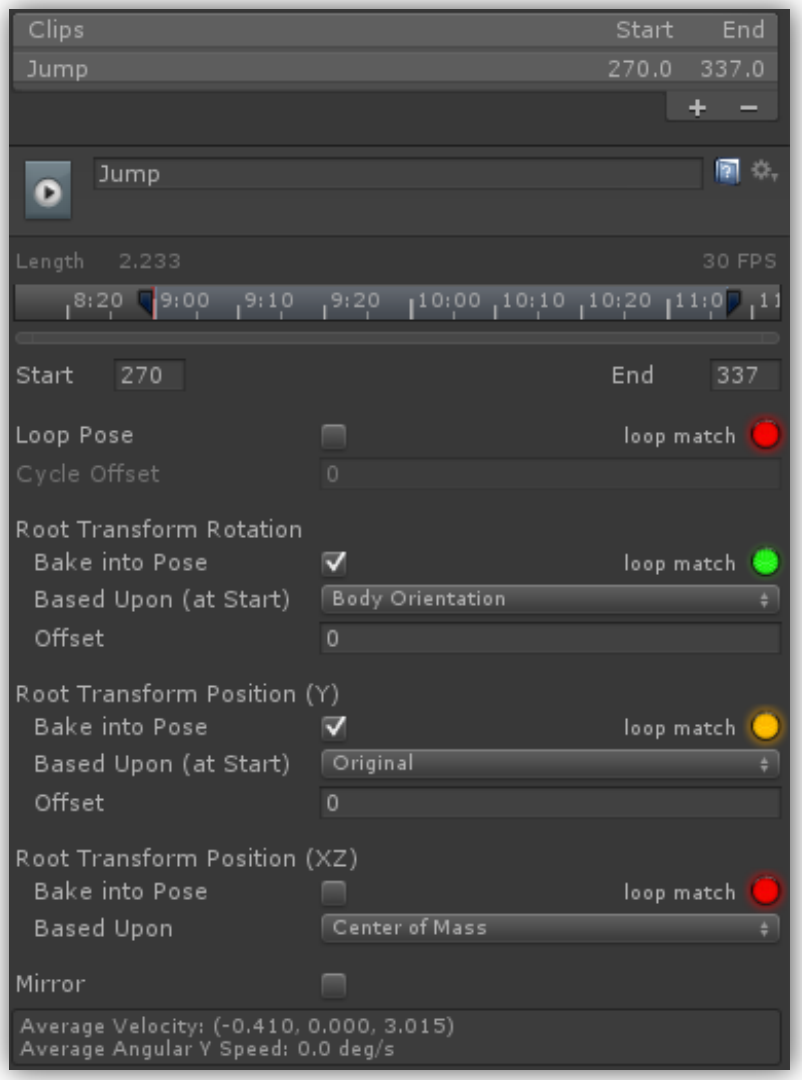


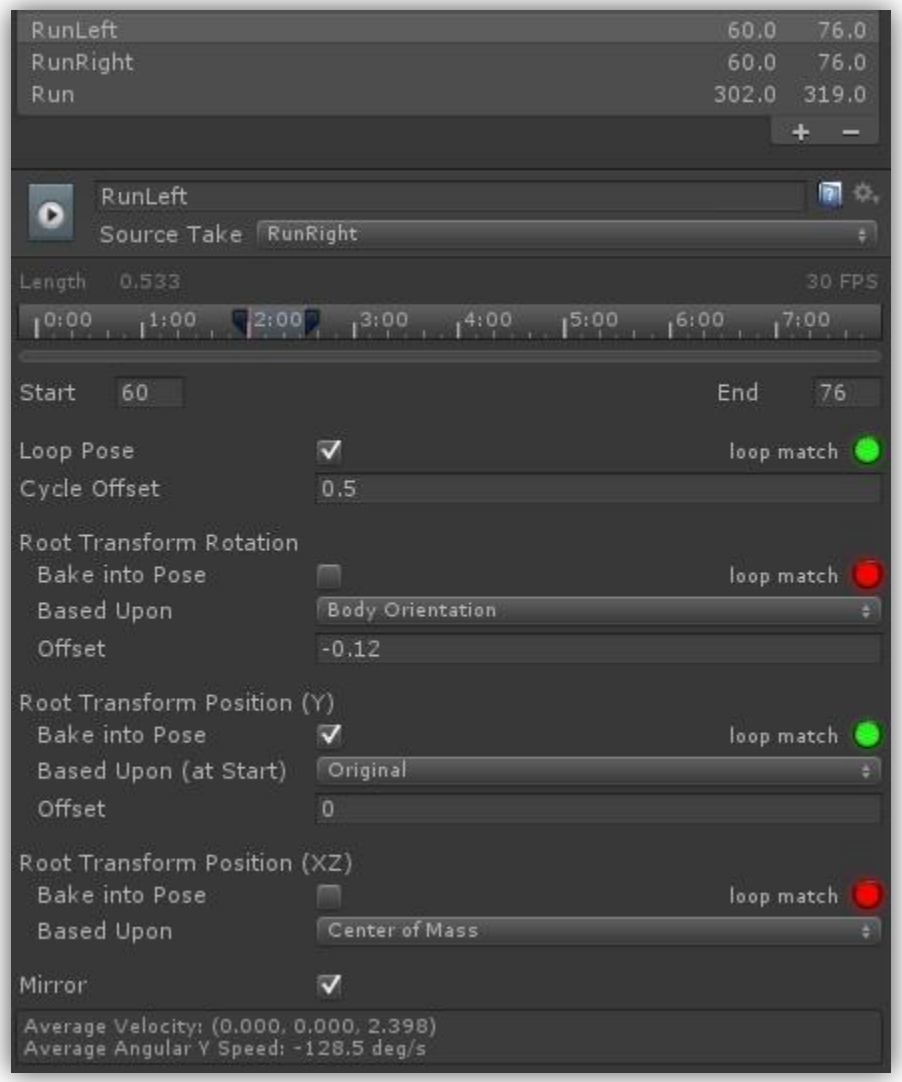


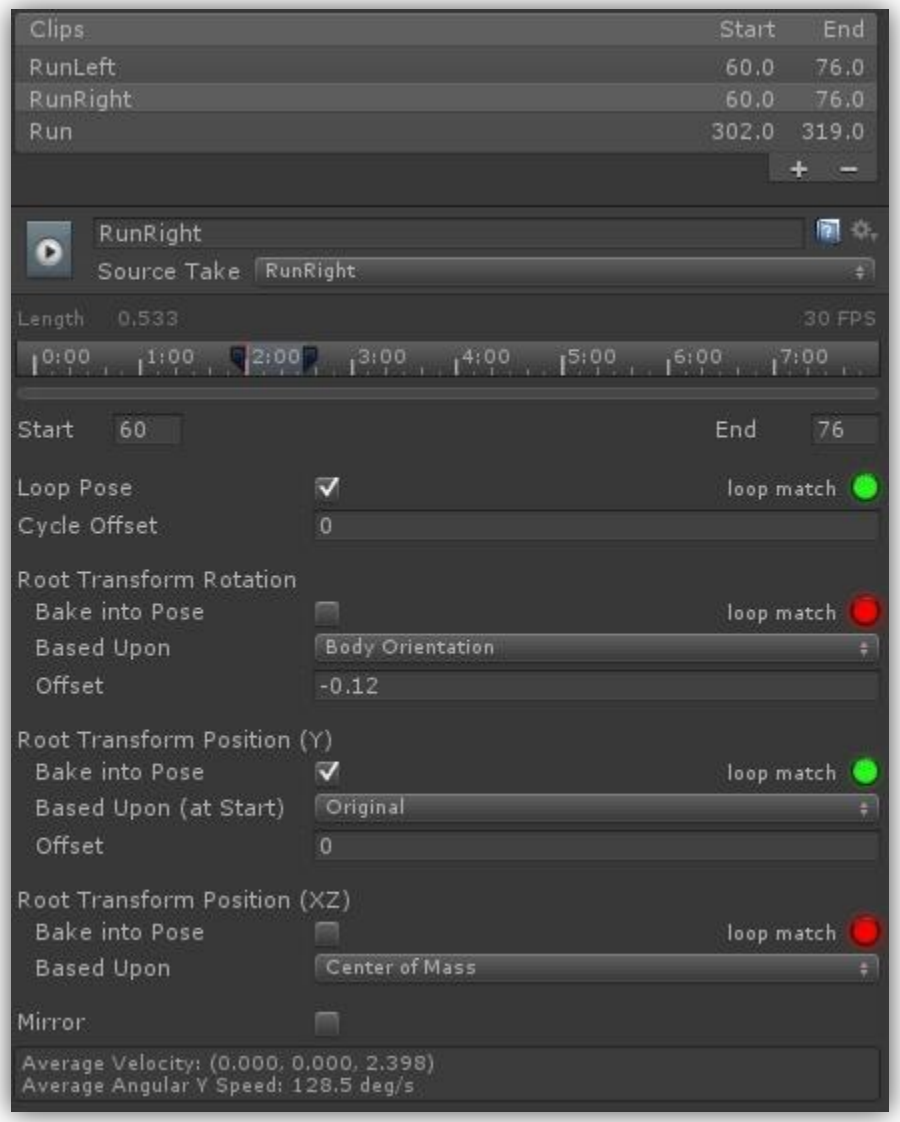
6. Hay que seleccionar todos los "**Anims_"** vamos a la opcion de **RIG**>
    a. **Animation Type: Humanoid.**
    b. **Avatar Definition>Copy from Other Avatar → Mesh_DudeAvatar.**
7. Ahora lo que haremos es configurar cada una de las animaciones y limpiar un poco pequeños errores que tienen par que quden en foramto **Loop**.

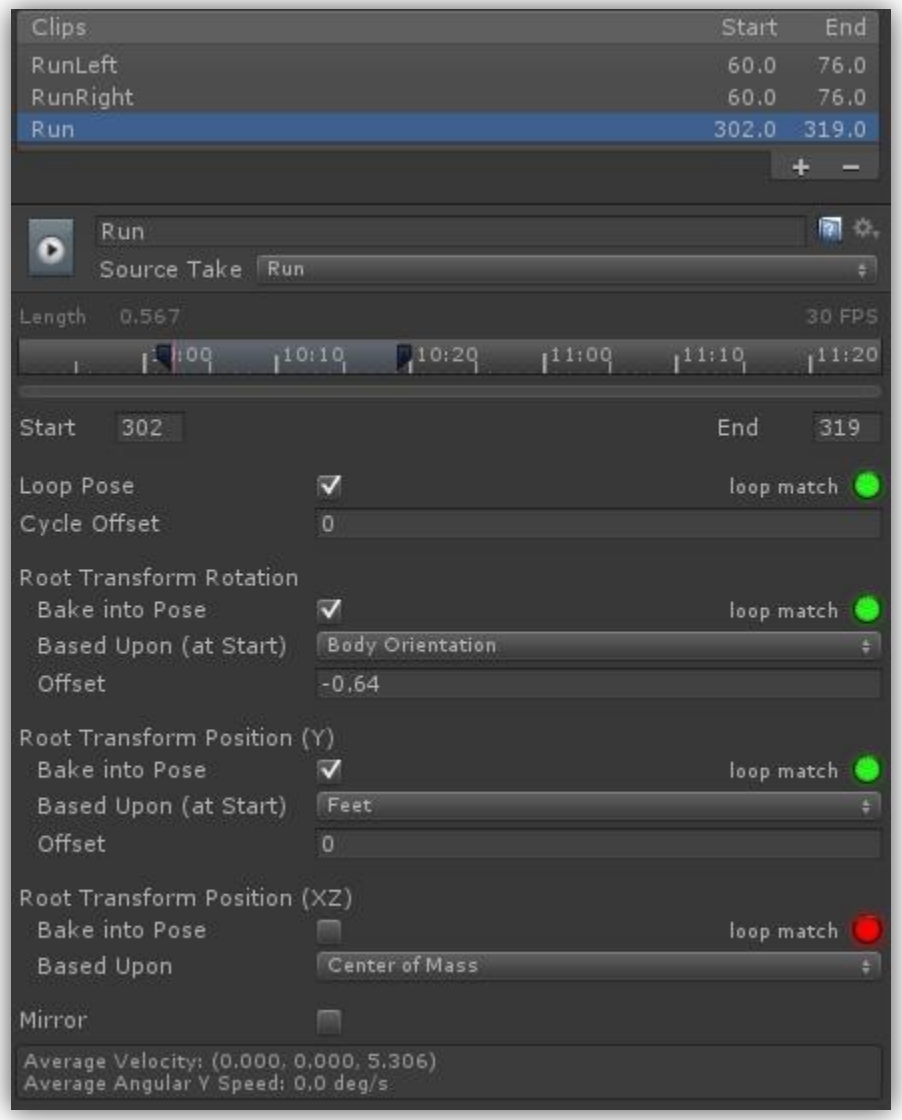


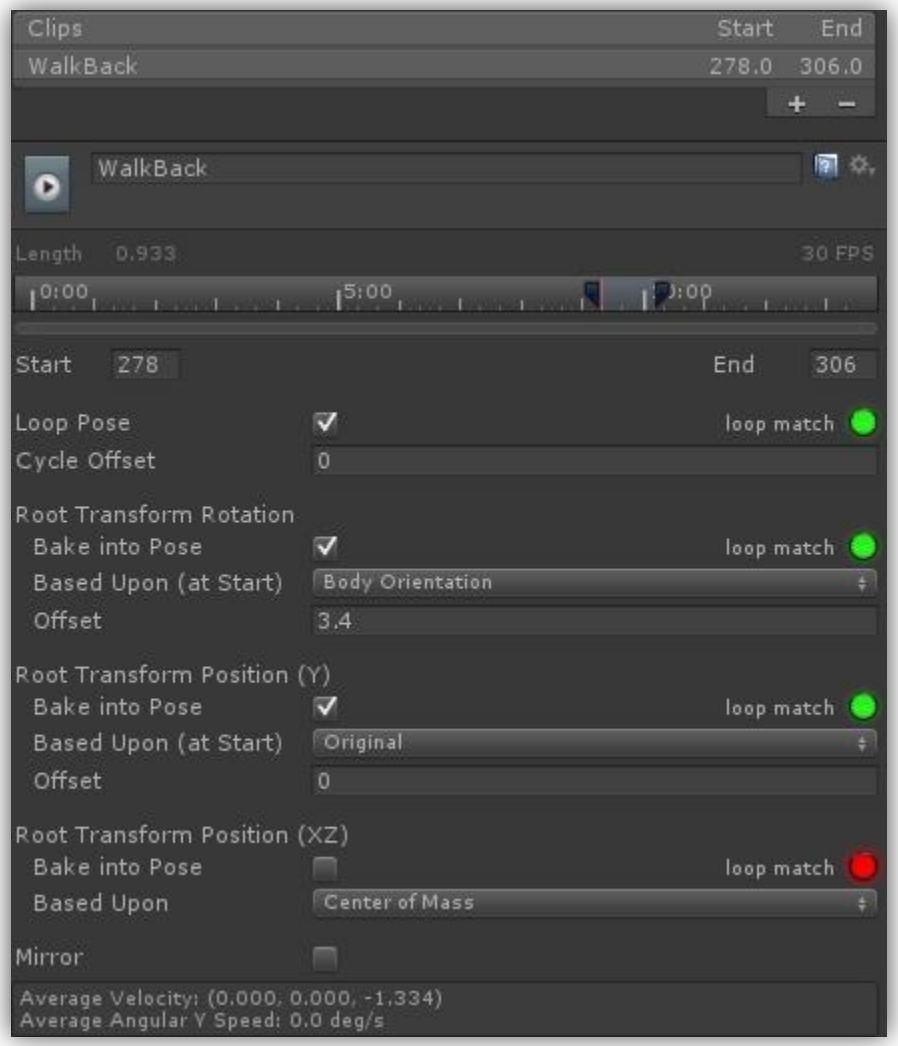


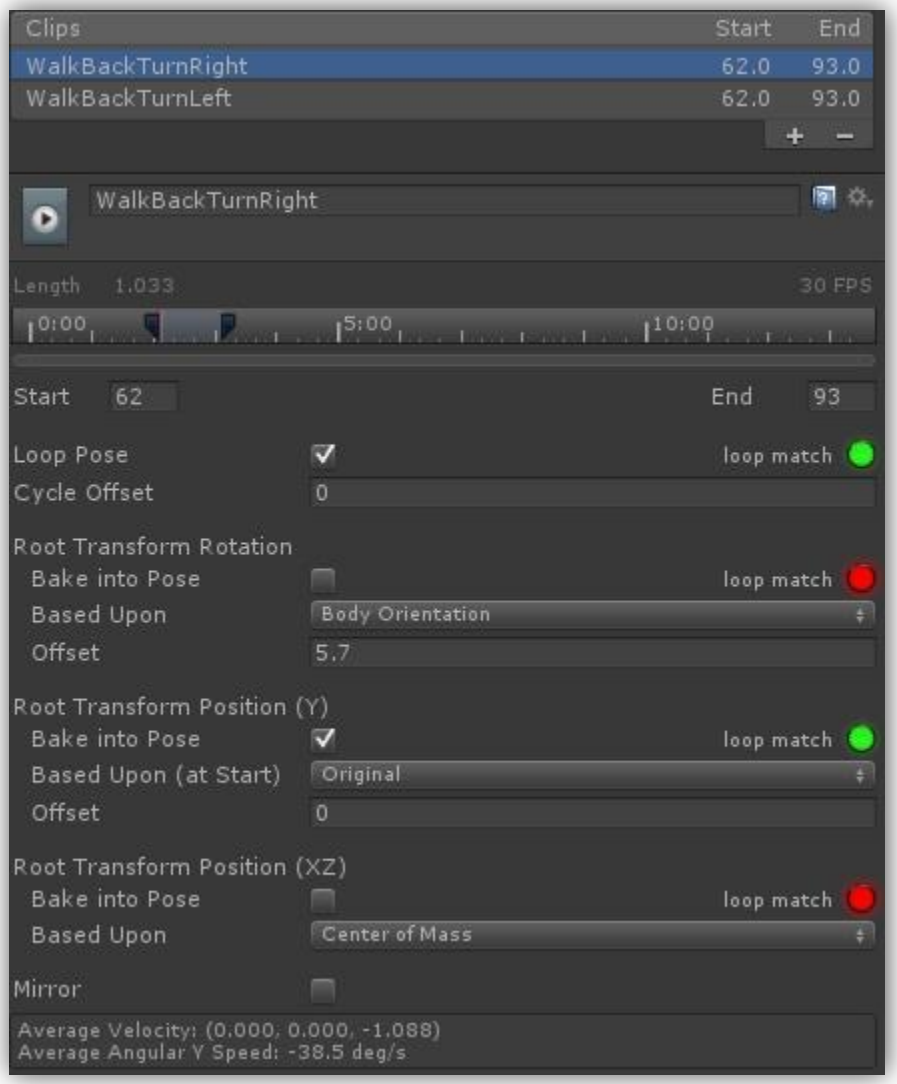

Clips
Start
End
Jump
270.0
337.0
Jump
Length 2.233
30 FPS
8:20
9:00
9:10
9:20
10:00
10:10
10:20
11:0
Start 270
End 337
Loop Pose
loop match
Cycle Offset
0
Root Transform Rotation
Bake into Pose
loop match
Based Upon (at Start)
Body Orientation
Offset
0
Root Transform Position (Y)
Bake into Pose
loop match
Based Upon (at Start)
Original
Offset
0
Root Transform Position (XZ)
Bake into Pose
loop match
Based Upon
Center of Mass
Mirror
Average Velocity: (-0.410, 0.000, 3.015)
Average Angular Y Speed: 0.0 deg/s

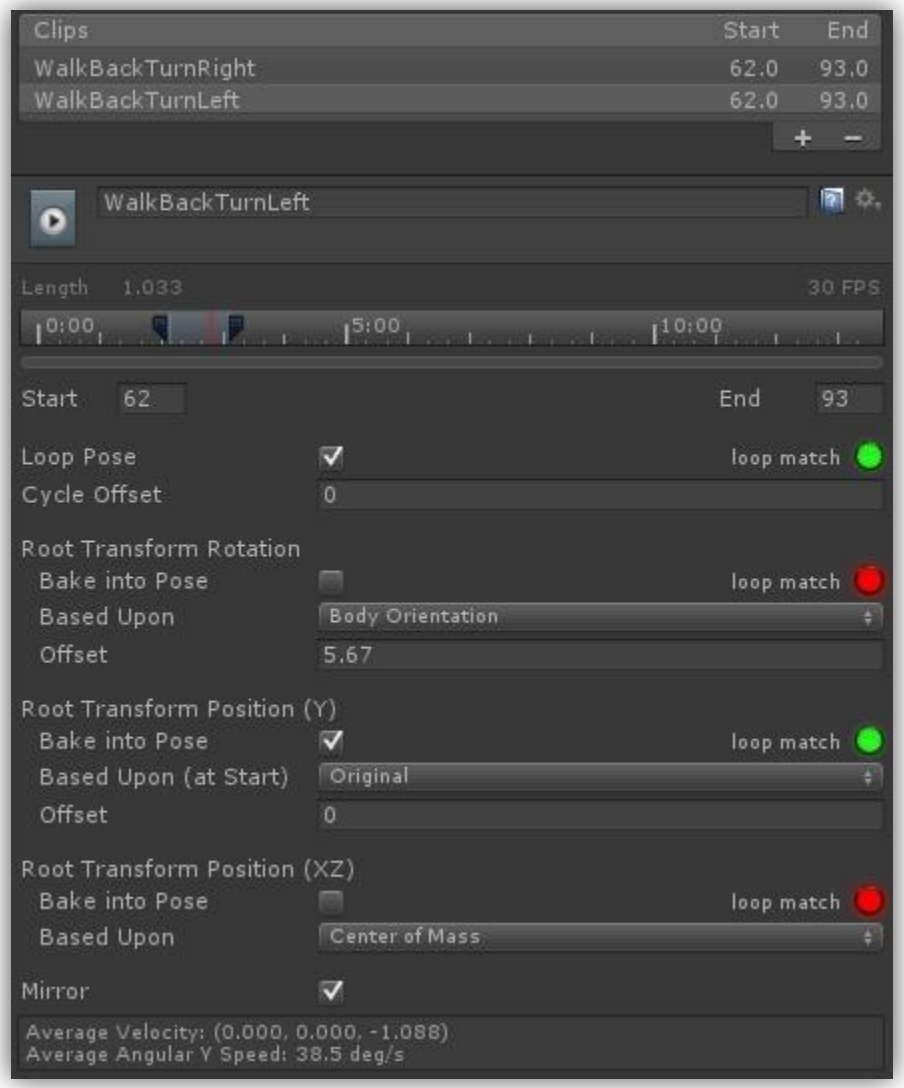

RunLeft
60.0
76.0
RunRight
60.0
76.0
Run
302.0
319.0
RunLeft
Source Take
RunRight
Length 0.533
30 FPS
0:00
1:00
2:00
3:00
4:00
5:00
6:00
7:00
Start 60
End 76
Loop Pose
loop match
Cycle Offset
0.5
Root Transform Rotation
Bake into Pose
loop match
Based Upon
Body Orientation
Offset
-0.12
Root Transform Position (Y)
Bake into Pose
loop match
Based Upon (at Start)
Original
Offset
0
Root Transform Position (XZ)
Bake into Pose
loop match
Based Upon
Center of Mass
Mirror
Average Velocity: (0.000, 0.000, 2.398)
Average Angular Y Speed: -128.5 deg/s

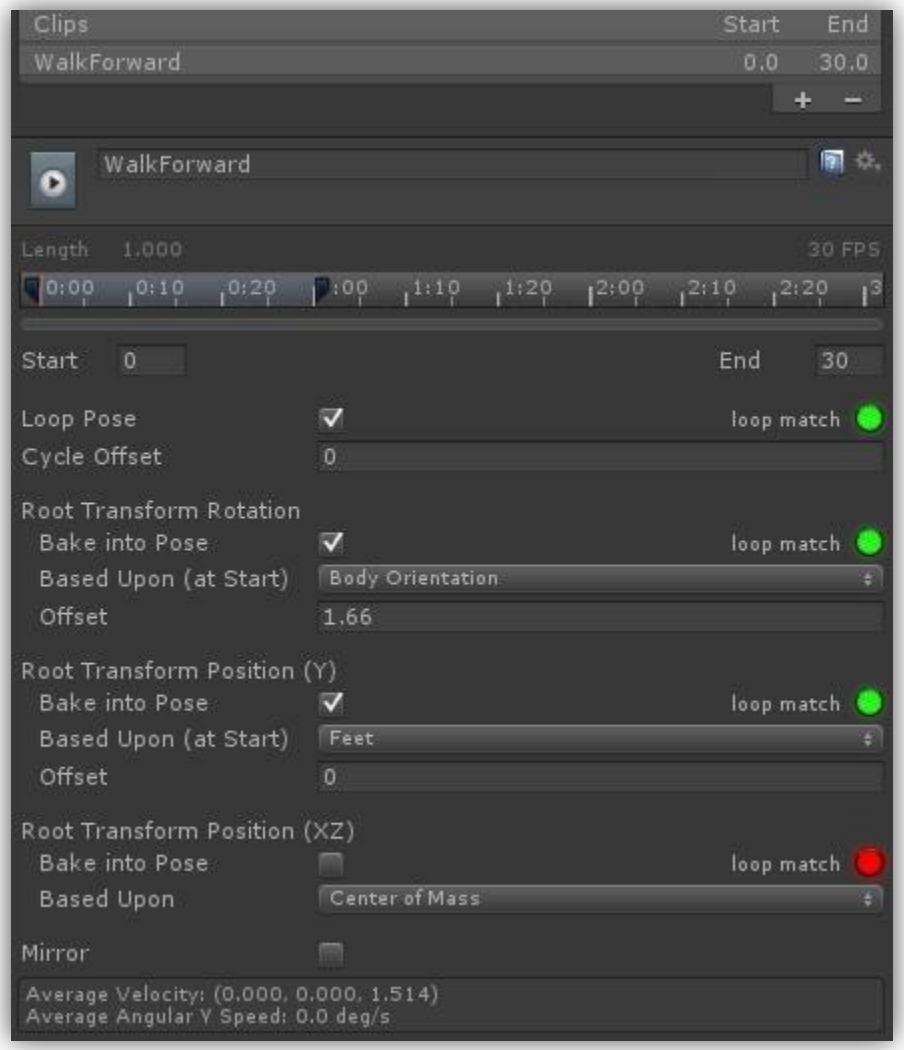

Clips
Start
End
RunLeft
60.0
76.0
RunRight
60.0
76.0
Run
302.0
319.0
RunRight
Source Take
RunRight
Length
0.533
30 FPS
0:00
1:00
2:00
3:00
4:00
5:00
6:00
7:00
Start
60
End
76
Loop Pose
loop match
Cycle Offset
0
Root Transform Rotation
Bake into Pose
loop match
Based Upon
Body Orientation
Offset
-0.12
Root Transform Position (Y)
Bake into Pose
loop match
Based Upon (at Start)
Original
Offset
0
Root Transform Position (XZ)
Bake into Pose
loop match
Based Upon
Center of Mass
Mirror
Average Velocity: (0.000, 0.000, 2.398)
Average Angular Y Speed: 128.5 deg/s

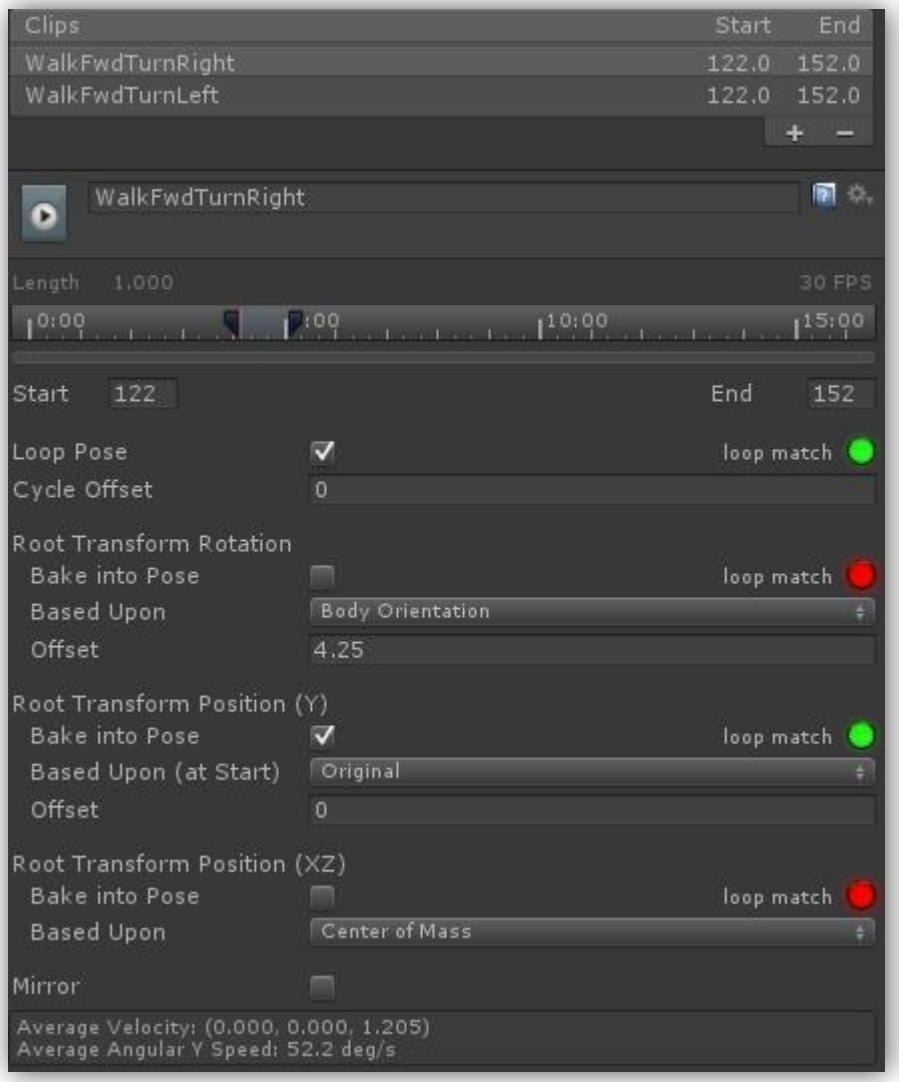

Clips
Start
End
RunLeft
60.0
76.0
RunRight
60.0
76.0
Run
302.0
319.0
Run
Source Take
Run
Length
0.567
30 FPS
10:10
10:20
11:00
11:10
11:20
Start
302
End
319
Loop Pose
loop match
Cycle Offset
0
Root Transform Rotation
Bake into Pose
loop match
Based Upon (at Start)
Body Orientation
Offset
-0.64
Root Transform Position (Y)
Bake into Pose
loop match
Based Upon (at Start)
Feet
Offset
0
Root Transform Position (XZ)
Bake into Pose
loop match
Based Upon
Center of Mass
Mirror
Average Velocity: (0.000, 0.000, 5.306)
Average Angular Y Speed: 0.0 deg/s

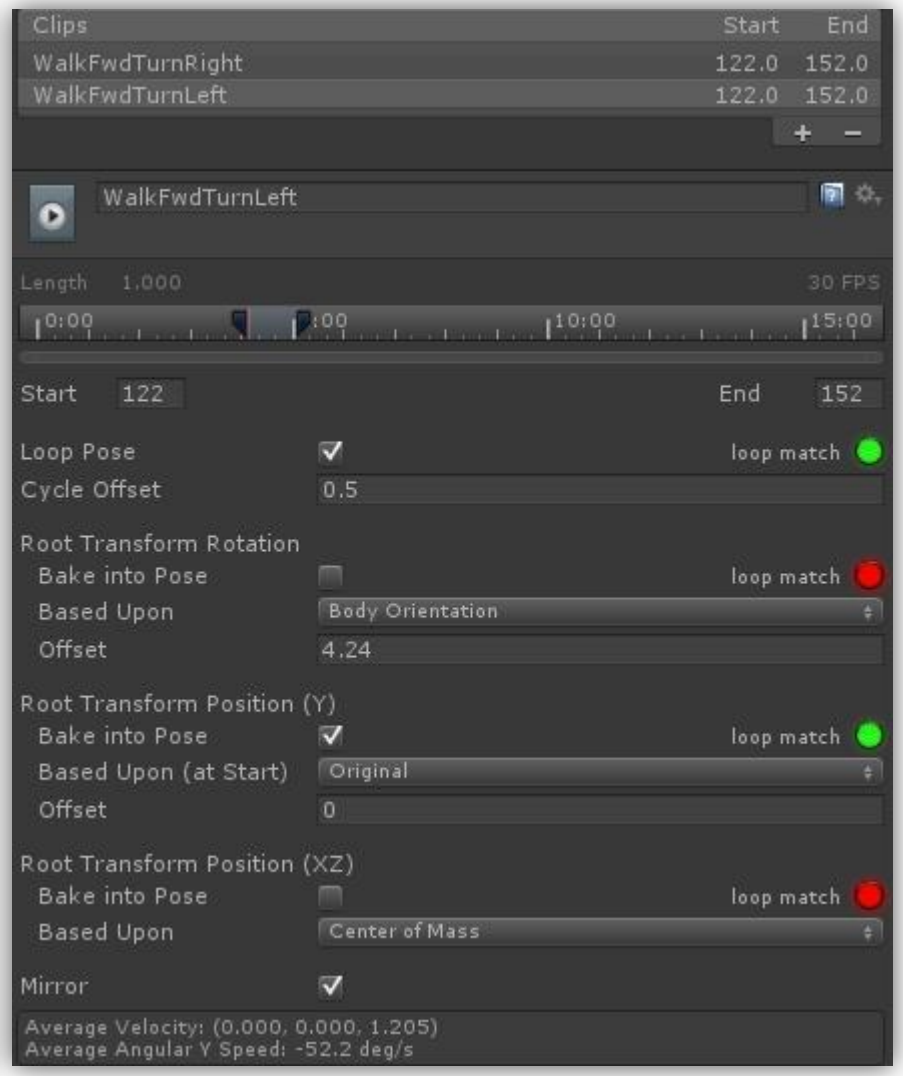

Clips
Start
End
WalkBack
278.0
306.0
WalkBack
Length 0.933
30 FPS
0:00
5:00
0:00
Start 278
End 306
Loop Pose
loop match
Cycle Offset 0
Root Transform Rotation
Bake into Pose
loop match
Based Upon (at Start) Body Orientation
Offset 3.4
Root Transform Position (Y)
Bake into Pose
loop match
Based Upon (at Start) Original
Offset 0
Root Transform Position (XZ)
Bake into Pose
loop match
Based Upon Center of Mass
Mirror
Average Velocity: (0.000, 0.000, -1.334)
Average Angular Y Speed: 0.0 deg/s


Clips
Start
End
WalkBackTurnRight
62.0
93.0
WalkBackTurnLeft
62.0
93.0
WalkBackTurnRight
Length 1.033
30 FPS
0:00
5:00
10:00
Start 62
End 93
Loop Pose
loop match
Cycle Offset 0
Root Transform Rotation
Bake into Pose
loop match
Based Upon Body Orientation
Offset 5.7
Root Transform Position (Y)
Bake into Pose
loop match
Based Upon (at Start) Original
Offset 0
Root Transform Position (XZ)
Bake into Pose
loop match
Based Upon Center of Mass
Mirror
Average Velocity: (0.000, 0.000, -1.088)
Average Angular Y Speed: -38.5 deg/s


Clips
Start
End
WalkBackTurnRight
62.0
93.0
WalkBackTurnLeft
62.0
93.0
WalkBackTurnLeft
Length 1.033
30 FPS
0:00
5:00
10:00
Start 62
End 93
Loop Pose
loop match
Cycle Offset 0
Root Transform Rotation
Bake into Pose
loop match
Based Upon Body Orientation
Offset 5.67
Root Transform Position (Y)
Bake into Pose
loop match
Based Upon (at Start) Original
Offset 0
Root Transform Position (XZ)
Bake into Pose
loop match
Based Upon Center of Mass
Mirror
Average Velocity: (0.000, 0.000, -1.088)
Average Angular Y Speed: 38.5 deg/s

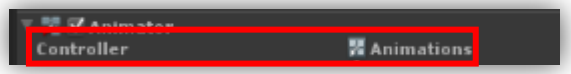

Clips
Start
End
WalkForward
0.0
30.0
WalkForward
Length 1.000
30 FPS
0:00
0:10
0:20
1:00
1:10
1:20
2:00
2:10
2:20
Start 0
End 30
Loop Pose
loop match
Cycle Offset 0
Root Transform Rotation
Bake into Pose
loop match
Based Upon (at Start) Body Orientation
Offset 1.66
Root Transform Position (Y)
Bake into Pose
loop match
Based Upon (at Start) Feet
Offset 0
Root Transform Position (XZ)
Bake into Pose
loop match
Based Upon Center of Mass
Mirror
Average Velocity: (0.000, 0.000, 1.514)
Average Angular Y Speed: 0.0 deg/s


Clips
Start
End
WalkFwdTurnRight
122.0
152.0
WalkFwdTurnLeft
122.0
152.0
WalkFwdTurnRight
Length 1.000
30 FPS
0:00
:00
10:00
15:00
Start 122
End 152
Loop Pose
loop match
Cycle Offset
0
Root Transform Rotation
Bake into Pose
loop match
Based Upon
Body Orientation
Offset
4.25
Root Transform Position (Y)
Bake into Pose
loop match
Based Upon (at Start)
Original
Offset
0
Root Transform Position (XZ)
Bake into Pose
loop match
Based Upon
Center of Mass
Mirror
Average Velocity: (0.000, 0.000, 1.205)
Average Angular Y Speed: 52.2 deg/s


Clips
Start
End
WalkFwdTurnRight
122.0
152.0
WalkFwdTurnLeft
122.0
152.0
WalkFwdTurnLeft
Length 1.000
30 FPS
0:00
:00
10:00
15:00
Start 122
End 152
Loop Pose
loop match
Cycle Offset
0.5
Root Transform Rotation
Bake into Pose
loop match
Based Upon
Body Orientation
Offset
4.24
Root Transform Position (Y)
Bake into Pose
loop match
Based Upon (at Start)
Original
Offset
0
Root Transform Position (XZ)
Bake into Pose
loop match
Based Upon
Center of Mass
Mirror
Average Velocity: (0.000, 0.000, 1.205)
Average Angular Y Speed: -52.2 deg/s

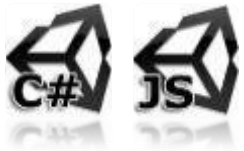
Ahora para poder cargar todas las animaciones en le personaje necesitamos crear un contenedor y colocar ahí todas las animaciones que necesitsamos.

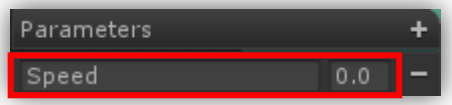
8. En **Project**>**Create**>**Animator Controller** y lo asignamos al **Dude** en el componente de **"Animator" en Controller**.
9. Ahora con el **"Animation Controller"** creado lo seleccionamos y abrimos el panel **Window>Animator** para que muestre sus opciones.
10. Agregamos la animacion de "**WalkForward**" y creamos un "**Parameters>Speed (float)**".
11. Crearemos un Script "**AI_Cha_Animation**" y lo asignamos al personaje.



AI_Cha_Animations

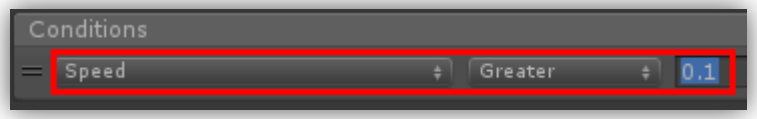
```
private Animator Anim; | private var Anim : Animator;

void | function Start (){
       // Obtener la informacion del Animtor y sus opciones.
       Anim = GetComponent<Animator> ();
}

void | function Update (){
       Anim.SetFloat ("Speed", Input.GetAxis ("Vertical"));
}
```



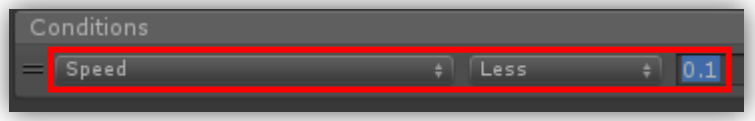


12. Creamos y Configuraremos las Transicion de **ida** y de **regreso**:

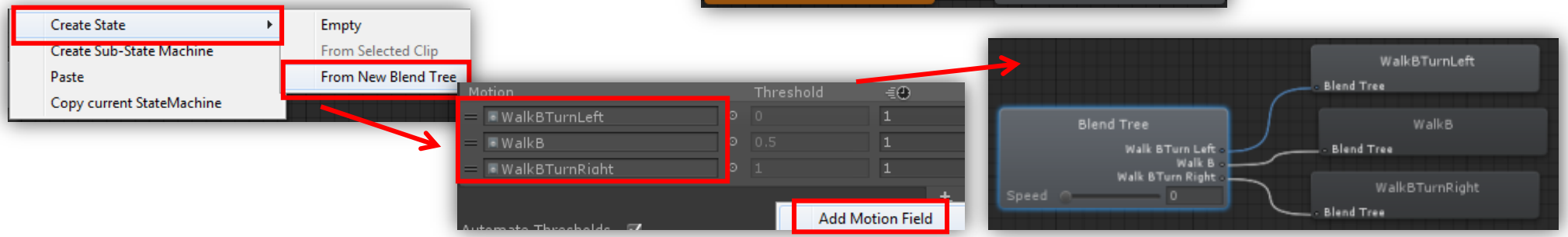
Idle - Speed>0.1 →



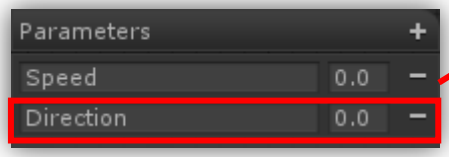
← WalkForward - Speed<0.1

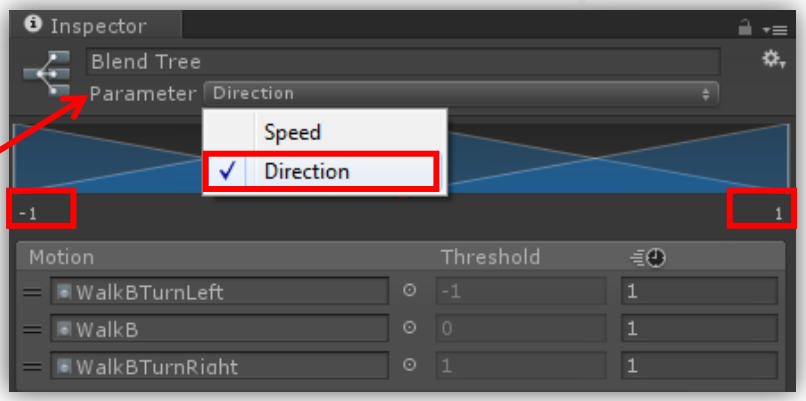


13. Ahora crearemos "**Blend Tree**" para agregar dentro **WalkB, WalkBTurn Left y Right** y lo conectamos directo en → **Idle**.
14. Sobre el nuevo Blend tree cambiamos el nombre a "**Walks Back**", samos 2click y entramos en el Clip.
15. Dentro agregamos "**Add Motion Field**" (**+**), para crear 3 Slots de Clips vacios y asignar ahí los siguientes clips de animación.
16. Ahora cambiamos la conexión de **Idle** con el nuevo Blend Tree (**Walks Back – 3 Clips**) y eliminamos el clip de solo **WalkBack.**



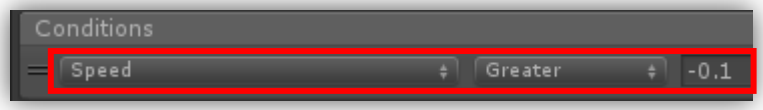
17. Para poder controlar el cambio de animaciones hacia los lados tanto para camianr como correr crearemos una variable dentro de mecanims, "**Parameters>Direction (Float)**", ahora lo asignamos en dentro de "**BlendTree - Walks Back**" en "**Parameter>Direction**", además de cambiar el valor de **Motions de**: **-1 a 1,** el cual será el valor que obtendremos del input de GetAxis que va de -1 a 1.



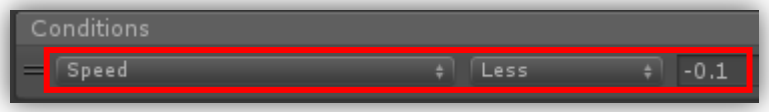


18. Transiciones La variable **Direction** nos permitirá cambiar de: **(-1) → WalkBTurnLeft, (0) → WalkB, (1) → WalkBTurnRight.**

"Walk Back" →



← "Idle"



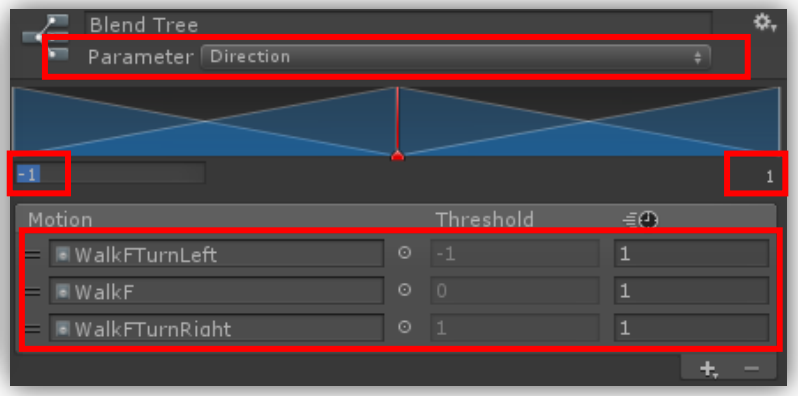
19. Regresamos al Script del Personaje y agregamos una línea para controlar la nueva variable "**Direction**".

AI_Cha_Animations

```
private Animator Anim; | private var Anim : Animator;
void | function Start (){
    // Obtener la informacion del Animtor cargado.
    Anim = GetComponent<Animator> ();
}

void | function FixedUpdate (){
    // Enfrente / Atras -> Input Vertical
    Anim.SetFloat ("Speed", Input.GetAxis ("Vertical") );
    // Lados -> Input Horizontal
    Anim.SetFloat ("Direction", Input.GetAxis ("Horizontal") );

}
```

20. Ahora Copiamos el Blend Tree, cambiando el nombre "**Walks Fordward",** asignamos las animaciones de WalkF y WalkLeft/Right.



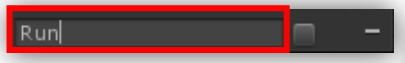


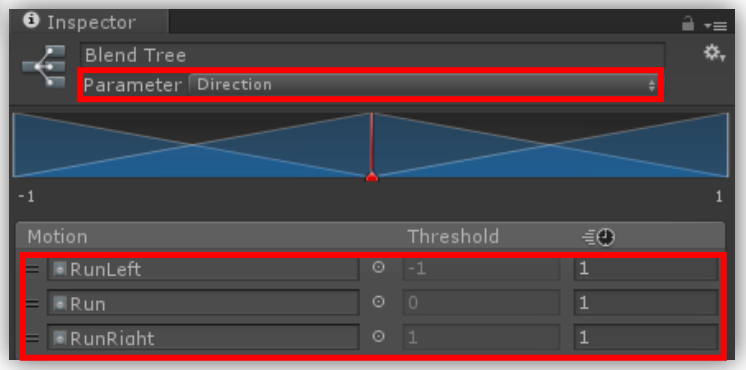
21. Ahora ajustamos los valores de las **nuevas transiciones [Speed**>-0.1 (→), **Speed**<-0.1(←)**].**

//RUN

Una vez más creamos otro nuevo "**Blend Tree"** o copiamos uno de los anteriores y lo nombramos "**Runs"** y entramos en el.

22. Habra **3 Slots** (**Run, RunLeft y RunRight**), **Motion**: **Min -1 a Max 1** (GetAxis) y usamos en Parameter la variable "**Direction**".
23. Creamos una Nuevo "**Parameters**>**Run"** (Bool).

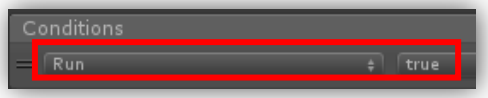




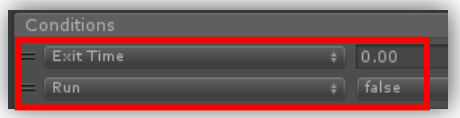
AI_Cha_Animations

```
private Animator Anim; | private var Anim : Animator;

void | function Start (){
    // Obtener la informacion del Animtor cargado.
    Anim = GetComponent<Animator> ();
}

void | function FixedUpdate (){
    //Caminar -> Input Vertical
    Anim.SetFloat ("Speed", Input.GetAxis ("Vertical") );
    //Lados -> Input Horizontal
    Anim.SetFloat ("Direction", Input.GetAxis ("Horizontal") );
    //Correr -> Left shit - Continuo
    Anim.SetBool("Run", Input.GetKey (KeyCode.LeftShift) );
}
```

26. Creamos transiciones: **"Walk Forward"** →



← " **Runs"**

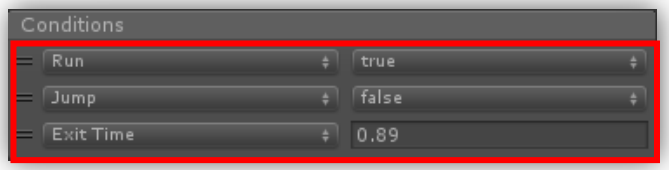


Por ultimo agregaremos la animación de **Saltar** (**Jump**) que estará conectado tanto a **Walk Forward** y **RUNs**.

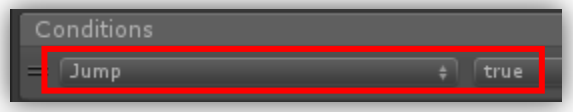
24. Creamos nuevo **Parameters>Jump** (Bool) e insertamos el Clip **Jump.**
25. Creamos las transiciones entre **Walk Forwad** y **Runs** con las condiciones:

- **Runs → Jump:**

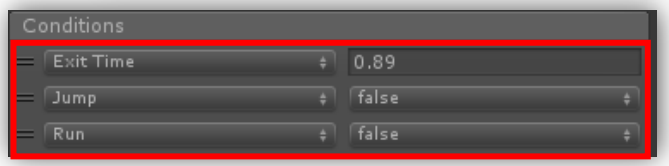
- **Jump →Run:**

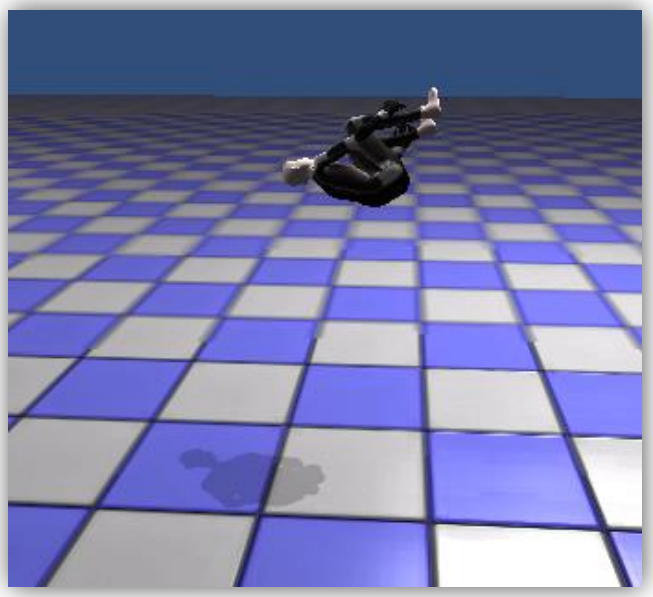


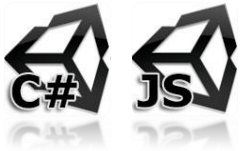
- **WalkFordward → Jump:**



- **Jump → WalkFordward:**

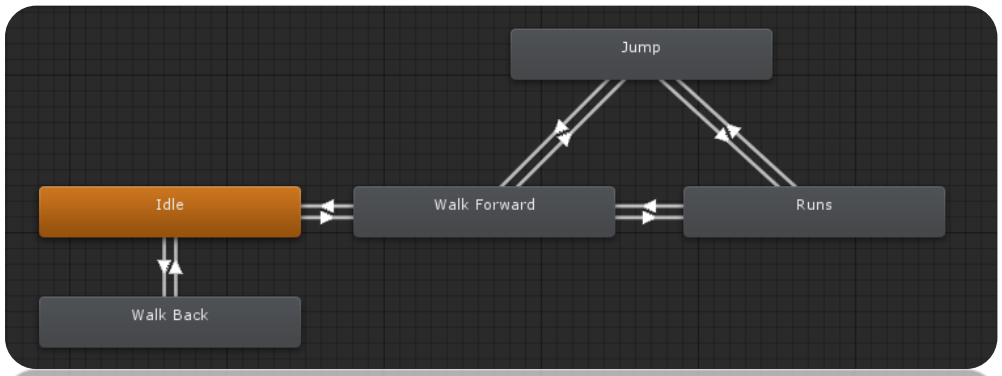






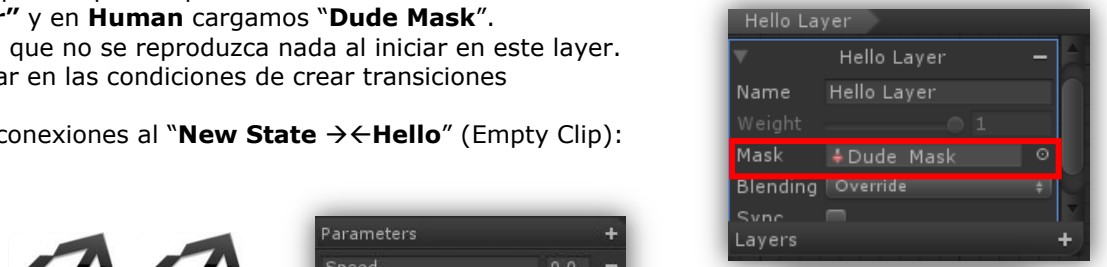
AI_Cha_Animations

```
void | function FixedUpdate (){
    //Caminar -> Input Vertical.
    Anim.SetFloat ("Speed", Input.GetAxis("Vertical"));
    //Lados -> Input Horizontal.
    Anim.SetFloat ("Direction", Input.GetAxis("Horizontal"));
    //Correr -> Left shit – Input Continuo.
    Anim.SetBool("Run", Input.GetKey (KeyCode.LeftShift) );
    //Saltar -> Space – Input Down.
    Anim.SetFloat ("Jump", Input.GetKey (KeyCode.Space) );
}
```



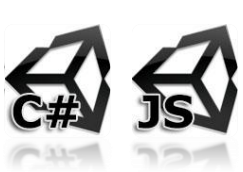
Para dar casi por terminado lo que haremos será crear mezcla de **animaciones** con **Blending**, esto con el propósito de poder tener **una Base de animaciones** que se le pueda sobrescribir una **mezcla** de animacion por medio de otro layer de animacion.

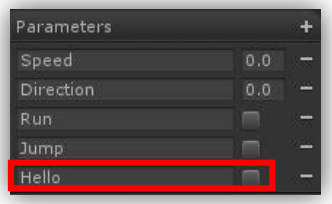
26. Creamos un nuevo "**Avatar Body Mask**" y desactivamos las partes que no queremos mezclar con las animaciones.
27. Vamos a **Base Layer** y creamos un nuevo layer **"Hello Layer"** y en **Human** cargamos "**Dude Mask**".
28. Ahora sobre el **"Hello Layer"** creamos un "**Empty Clip"** para que no se reproduzca nada al iniciar en este layer.
29. Creamos una nuevo "**Parameters>Hello (Bool)**" para asignar en las condiciones de crear transiciones (Conexiones).
30. Ahora insertamos el Clip de animación "**Hello**" y creamos las conexiones al "**New State →←Hello**" (Empty Clip):

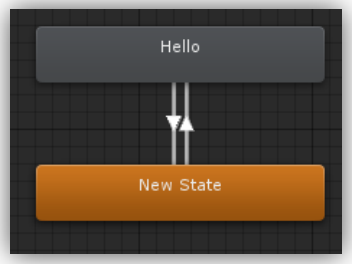


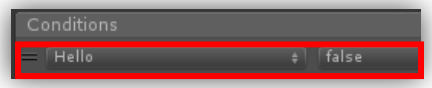
AI_Cha_Animations

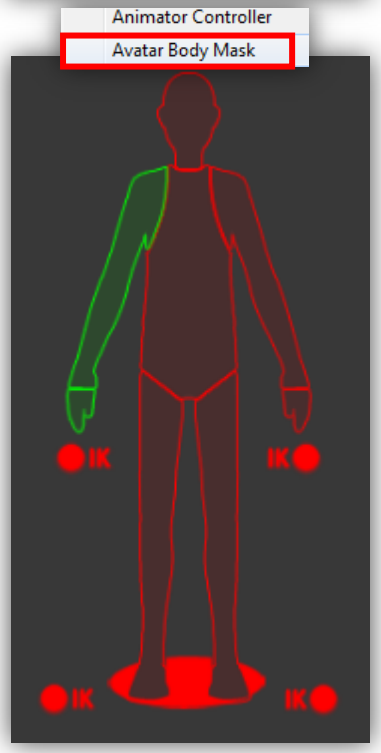
```
private Animator Anim; | private var Anim : Animator;

void | function Start (){
    // Obtener la informacion del Animtor cargado.
    Anim = GetComponent<Animator> ();
    // Si hay 2 layers hacer que tenga blending layer 2, al 100% (1,1f)
    Anim.SetLayerWeight (1, 1f);
}
void | function FixedUpdate (){
    //Caminar -> Input Vertical.
    Anim.SetFloat ("Speed", Input.GetAxis ("Vertical"));
    //Lados -> Input Horizontal.
    Anim.SetFloat ("Direction", Input.GetAxis ("Horizontal"));
    //Correr -> Left shit – Input Continuo.
    Anim.SetBool ("Run", Input.GetKey (KeyCode.LeftShift));
    //Saltar -> Space – Input Down.
    Anim.SetFloat ("Jump", Input.GetKey (KeyCode.Space));
    //Hello – Input Continuo.
    Anim.SetBool ("Hello", Input.GetKey (KeyCode.Mouse0));
}
```



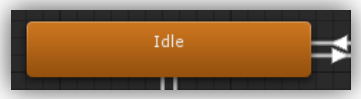


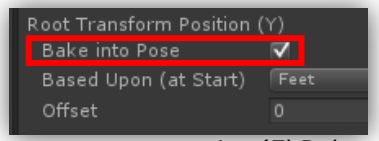


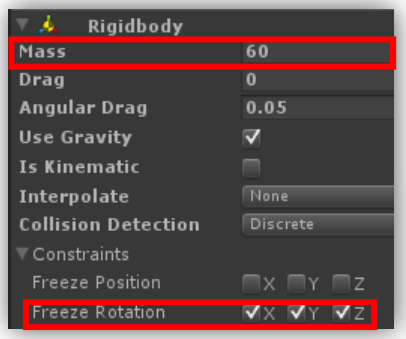


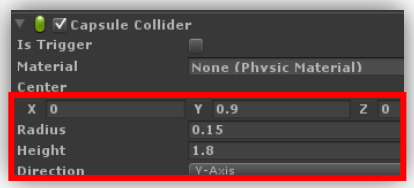


31. Al personaje le agregaremos **RigidBody (Mass 60 y Freeze Rotation en (X, Y, Z)** y un **Capsule Collider** (**Y: 0.9, Radius: 0.15, Height 1.8**).
32. Hay que agregar el Clip de **Idle** con con "**Bake into Pose: on**" para que al poner físicas caiga normal.









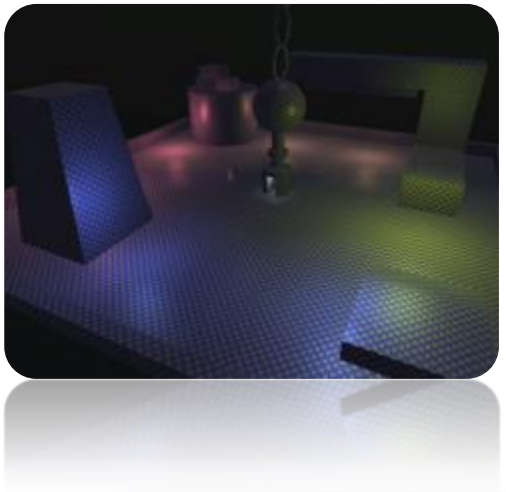
Nota: Ya se pueden ompartir las animacion con otros personajes (El Robot – deberá tener su Propio Avatar ya que es otro personaje con diferente setup).

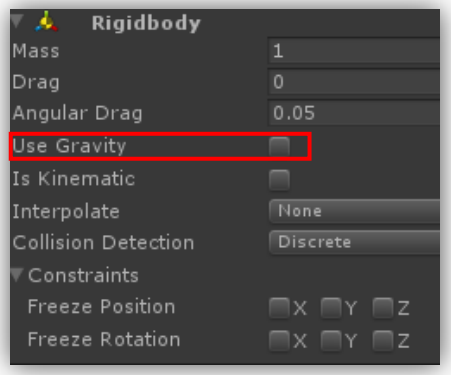
1.43 FPS MINI GAME

First Person Shooter - Mini Game (C# Script):

static function IgnoreCollision (collider1 : Collider, collider2 : Collider, ignore : boolean = true) : void

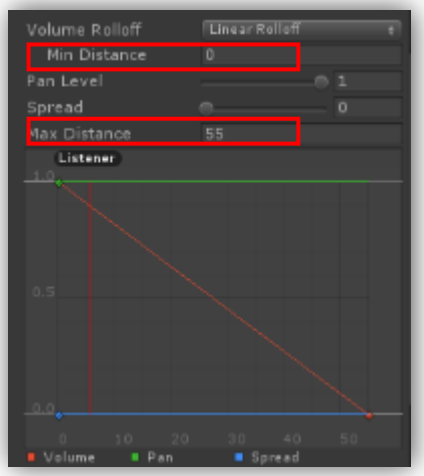
1. Cargamos el paquete con la escena "**FPS Scene**", se creará un personaje que cargue un arma, lance misiles explosivos.
2. Insertamos el "**rocketLauncher**" y lo acomodamos como arma al personaje para emparentarlo después a la **cámara**.
3. Insertamos el "**Rocket**" en la escena y le Asignamos **Rigidbody** y desactivamos la gravedad (**Use Gravity: Of**).
4. **Rocket:** Le emparentamos un "**Point Light**" y le ajustamos los parámetros que mas nos gusten.
5. **Rocket:** Le agregamos un **Audio Soucer**, cargamos el audio "**RocketLauncherFire**".
6. **Rocket:** Le asignamos un **Box Collider** y se ajustara al tamaño del **Rocket**.
7. **Gun:** Creamos un Script "**AI_Gun**" y se lo asignamos al **_Prefab_Gun>RocketLauncher**.





AI_RocketLauncher

```
//Cargar el Misil desde el data

public Rigidbody Rocket;
public int Speed = 30;


void Update () {

  //Click Izquierdo.
   if( Input.GetKeyDown ( KeyCode.Mouse0 ) ){

      //Usar la variable con el objeto de misil (Rigidbody) y crearlo dinámicamente en la escena
    Rigidbody RocketInst = Instantiate (Rocket, transform.position, transform.rotation) as Rigidbody;

      //Agregar velocidad constante al Misil creado en la escena dirigido hacia adelante (Z).
      RocketInst.velocity = transform.forward * Speed;

      // Normalmente se disparara desde un personaje y para evitar que colision con el personaje:
      Physics.IgnoreCollision (RocketInst.collider, transform.root.collider, true);
      Nota: Aqui probar con una caja antes de las partículas para mostrar la dirección del objeto.
   }

}
```



static function **OverlapSphere (position :** Vector3**, radius :** float**, layerMask :** int **= kAllLayers) :** Collider**[]**
function **AddExplosionForce (explosionForce :** float**, explosionPosition :** Vector3**, explosionRadius :** float**, upwardsModifier :** float **= 0.0F, mode :** ForceMode **= ForceMode.Force) :** void

Ahora se creara el efecto de que cuando la bala se impacte y genere el efecto de explosión, haremos que el sistema de partículas se emita cuando colisione la bala, y se coloque en el polígono que colisiono (posición y cara normal).

8. En el "**Prefab_P.Explotion**", en este activamos "**One Shot**" y "**Autodesctruct**" (Legacy Particles).
9. **Explotion**: Asignamos "**Audio Sorce**" y asignamos el sonido "**RocketLauncherImpact**" con **Min**: 1 **Max**: 55.
10. **Rocket:** Creamos un Script "**AI_Rocket**" y se lo aplicaremos a **_Prefab_Rocket**

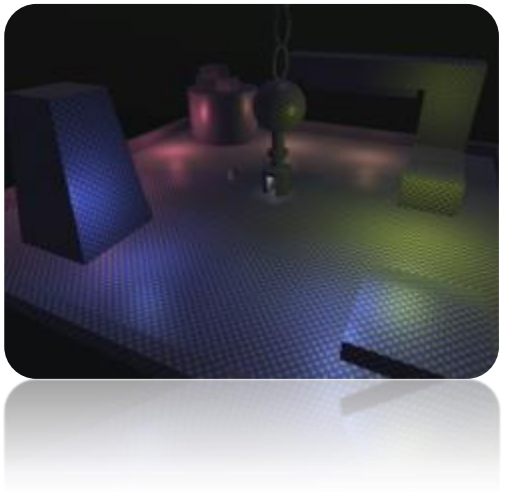
AI_Rocket

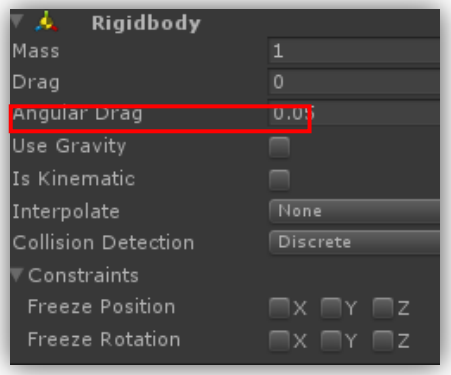
```
public GameObject ParticleExplotion;
public int ExplosionRadius = 5;
public float ExplosionPower = 2000f;


void OnCollisionEnter(Collision vCollision) {
	//Destruir el misil para que solo después aparezcan las partículas con su audio.
	Destroy( gameObject );

	//POSICIÓN: "contactPoint" permite obtener la posición "x,y,z" con lo que colisiona, (primer objeto = 0).
	ContactPoint ObjImpactPos = vCollision.contacts[0];
	//ROTACIÓN: Saber sobre qué cara de la normal colisiono (Vector3.Up = Cara de la Normal Frontal).
	Quaternion ObjImpactRot = Quaternion.FromToRotation (Vector3.up, ObjImpactPos.normal);

	// OBJETO (POS, ROT): crear instancia de la explosión, en la posición donde colisiono el rocket.
	Instantiate (ParticleExplotion, ObjImpactPos.point, ObjImpactRot);


	//Al impactar el misil creara una esfera, y los objetos que esten dentro de su radio con collider se almacenaran en un arreglo.
	Collider[] ObjColl = Physics.OverlapSphere (transform.position, ExplosionRadius);

	//Descargamos cada uno de los "Objeto con collider" dentro de la variable "hit".
	foreach (Collider Hit in ObjColl){
		//Y Si el objeto colisionado tiene asignado "collider" y "Rigidbod".
		if (Hit.rigidbody){
			//Agregamos impulso desde donde colisiono el Misicl hacia el objeto para hacerlo volar.
			Hit.rigidbody.AddExplosionForce (ExplosionPower, transform.position, ExplosionRadius);
		}
	}
}
```

First Person Shooter - Mini Game (C# Script):

static function IgnoreCollision (collider1 : Collider, collider2 : Collider, ignore : boolean = true) : void

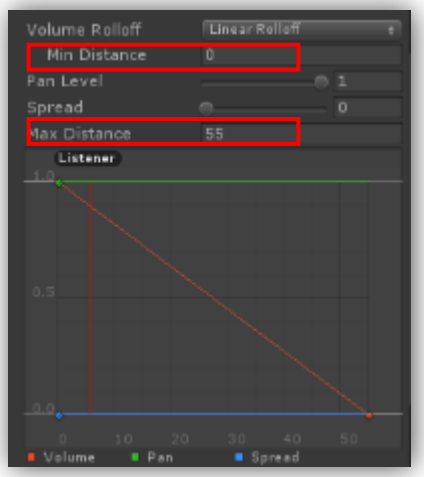
1. Cargamos el paquete con la escena "**FPS Scene**", se creará un personaje que cargue un arma, lance misiles explosivos.
2. Insertamos el "**rocketLauncher**" y lo acomodamos como arma al personaje para emparentarlo después a la **cámara**.
3. Insertamos el "**Rocket**" en la escena y le Asignamos **Rigidbody** y desactivamos la gravedad (**Use Gravity: Of**).
4. **Rocket:** Le emparentamos un "**Point Light**" y le ajustamos los parámetros que mas nos gusten.
5. **Rocket:** Le agregamos un **Audio Soucer**, cargamos el audio "**RocketLauncherFire**".
6. **Rocket:** Le asignamos un **Box Collider** y se ajustara al tamaño del **Rocket**.
7. **Gun:** Creamos un Script "**AI_Gun**" y se lo asignamos al **_Prefab_Gun>RocketLauncher**.





AI_RocketLauncher

```
//Cargar el Misil desde el data

var Rocket : Rigidbody;
var Speed : int = 30;


function Update () {

  //Click Izquierdo.
   if( Input.GetKeyDown ( KeyCode.Mouse0 ) ){

      //Usar la variable con el objeto de misil (Rigidbody) y crearlo dinámicamente en la escena
    Var RocketInst : Rigidbody = Instantiate (Rocket, transform.position, transform.rotation);

      //Agregar velocidad constante al Misil creado en la escena dirigido hacia adelante (Z).
      RocketInst.velocity = transform.forward * Speed;

      // Normalmente se disparara desde un personaje y para evitar que colision con el personaje:
      Physics.IgnoreCollision (RocketInst.collider, transform.root.collider, true);
      Nota: Aqui probar con una caja antes de las partículas para mostrar la dirección del objeto.
   }

}
```



static function **OverlapSphere (position :** Vector3**, radius :** float**, layerMask :** int **= kAllLayers) :** Collider**[]**
function **AddExplosionForce (explosionForce :** float**, explosionPosition :** Vector3**, explosionRadius :** float**, upwardsModifier :** float **= 0.0F, mode :** ForceMode **= ForceMode.Force) :** void

Ahora se creara el efecto de que cuando la bala se impacte y genere el efecto de explosión, haremos que el sistema de partículas se emita cuando colisione la bala, y se coloque en el polígono que colisiono (posición y cara normal).

8. En el "**Prefab_P.Explotion**", en este activamos "**One Shot**" y "**Autodesctruct**" (Legacy Particles).
9. **Explotion**: Asignamos "**Audio Sorce**" y asignamos el sonido "**RocketLauncherImpact**" con **Min**: 1 **Max**: 55.
10. **Rocket:** Creamos un Script "**AI_Rocket**" y se lo aplicaremos a _**Prefab_Rocket**.

AI_Rocket

```
var ParticleExplotion : GameObject;
var ExplosionRadius : int = 5;
var ExplosionPower : float = 2000;


void OnCollisionEnter(vCollision : Collision) {
	//Destruir el misil para que solo después aparezcan las partículas con su audio.
	Destroy( gameObject );

	//POSICIÓN: "contactPoint" permite obtener la posición "x,y,z" con lo que colisiona, (primer objeto = 0).
	var ObjImpactPos : Collision = vCollision.contacts[0];
	//ROTACIÓN: Saber sobre qué cara de la normal colisiono (Vector3.Up = Cara de la Normal Frontal).
	var ObjImpactRot : Quaternion = Quaternion.FromToRotation (Vector3.up, ObjImpactPos.normal);

	// OBJETO (POS, ROT): crear instancia de la explosión, en la posición donde colisiono el rocket.
	Instantiate (ParticleExplotion, ObjImpactPos.point, ObjImpactRot);


	//Al impactar el misil creara una esfera, y los objetos que esten dentro de su radio con collider se almacenaran en un arreglo.
	var ObjColl : Collider[] = Physics.OverlapSphere (transform.position, ExplosionRadius);

	//Descargamos cada uno de los "Objeto con collider" dentro de la variable "hit".
	foreach (Collider Hit in ObjColl){
		//Y Si el objeto colisionado tiene asignado "collider" y "Rigidbod".
		if (Hit.rigidbody){
			//Agregamos impulso desde donde colisiono el Misicl hacia el objeto para hacerlo volar.
			Hit.rigidbody.AddExplosionForce (ExplosionPower, transform.position, ExplosionRadius);
		}
	}
}
```

1.44 PAUSE GAMES.

Pause Game (C# Script):

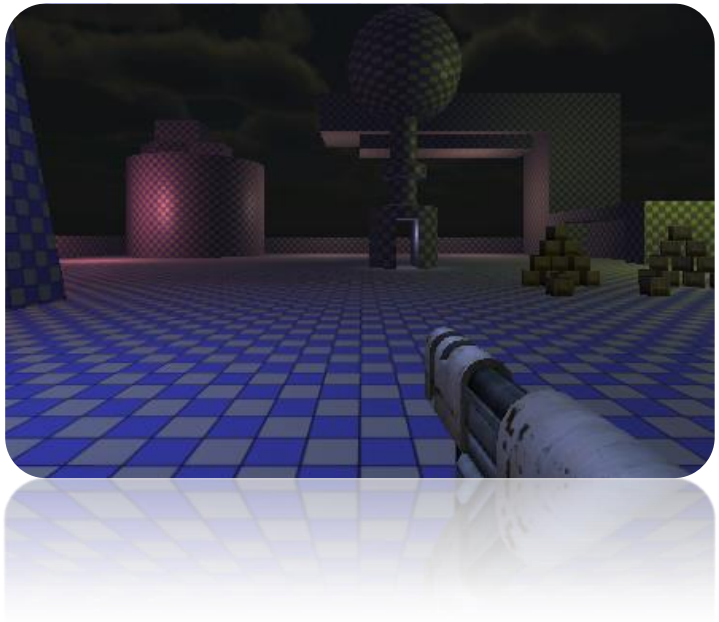
Para Pausar el juego necesitamos detener el audio y los movimiento del carácter controller por medio del "MouseLook" en X & Y creando un Script "**AI_Pause**" y lo asignamos a la cámara.

AI_Pause

```
private bool GamePause = false;
public GameObject CharacterMouseX;
public GameObject CameraMouseY;

void Start ()
{
	//Ocultar Mouse
	Screen.showCursor = false;
}

void Update ()
{
	if (Input.GetKeyDown (KeyCode.Escape) ) {
		GamePause = !GamePause;

		if (GamePause) {
			//PAUSE
			Time.timeScale = 0f; //Pause
			AudioListener.pause = true; //Mute
			Screen.showCursor = true; //show
			//Nodejar que se mueva el Character controller con el Mouse.
			CharacterMouseX.GetComponent <MouseLook> ().enabled = false;
			CameraMouseY.GetComponent <MouseLook> ().enabled = false;
		} else {
			//NO PAUSE
			Time.timeScale = 1f; // No Pause
			AudioListener.pause = false; // sound
			Screen.showCursor = false; // Hide
			// Dejar que se mueva el Character controller con el Mouse.
			CharacterMouseX.GetComponent <MouseLook> ().enabled = true; //Activar Mouse Look en X en el personaje.
			CameraMouseY.GetComponent <MouseLook> ().enabled = true; //Activar Mouse Look en Y en la cámara.
		}
		print ("Pause: " + GamePause);
	}
}
```

Pause Game (Java Script):

Para Pausar el juego necesitamos detener el audio y los movimiento del carácter controller por medio del "MouseLook" en X & Y creando un Script "**AI_Pause**" y lo asignamos a la cámara.



AI_Pause

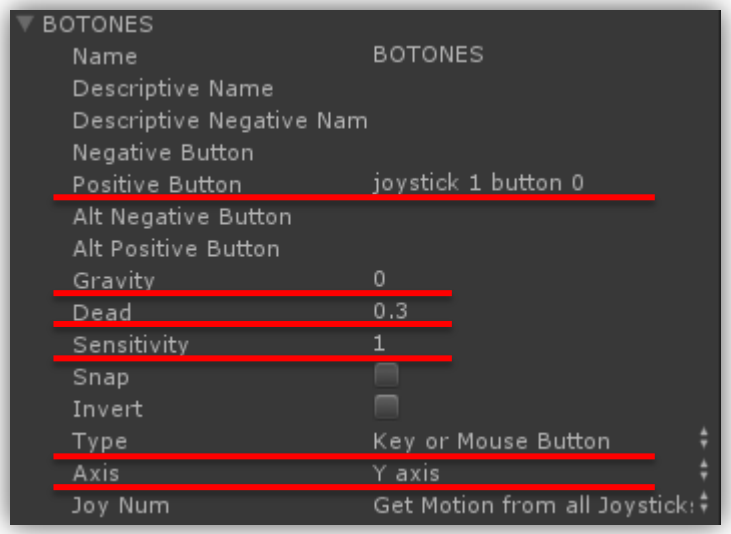
```
var GamePause : boolean = false;
var CharacterMouseX : GameObject;
var CameraMouseY : GameObject;

function Start ()
{
	Screen.showCursor = false; //Ocultar Mouse
}

function Update ()
{

	if (Input.GetKeyDown KeyCode.Escape) ) {
		GamePause = !GamePause;

		if (GamePause) {
			//PAUSE
			Time.timeScale = 0; //Pause
			AudioListener.pause = true; //Mute
			Screen.showCursor = true; //show
			//Nodejar que se mueva el Character controller con el Mouse.
			CharacterMouseX.GetComponent (MouseLook).enabled = false;
			CameraMouseY.GetComponent (MouseLook>).enabled = false;
		} else {
			//NO PAUSE
			Time.timeScale = 1; // No Pause
			AudioListener.pause = false; // sound
			Screen.showCursor = false; // Hide
			//Dejar que se mueva el Character controller con el Mouse.
			CharacterMouseX.GetComponent (MouseLook).enabled = true; //Activar Mouse Look en X en el personaje.
			CameraMouseY.GetComponent (MouseLook).enabled = true; //Activar Mouse Look en Y en la cámara.
		}
		print ("Pause: " + GamePause);
	}

}
```

1.45 CONTROLLER MAPPING.

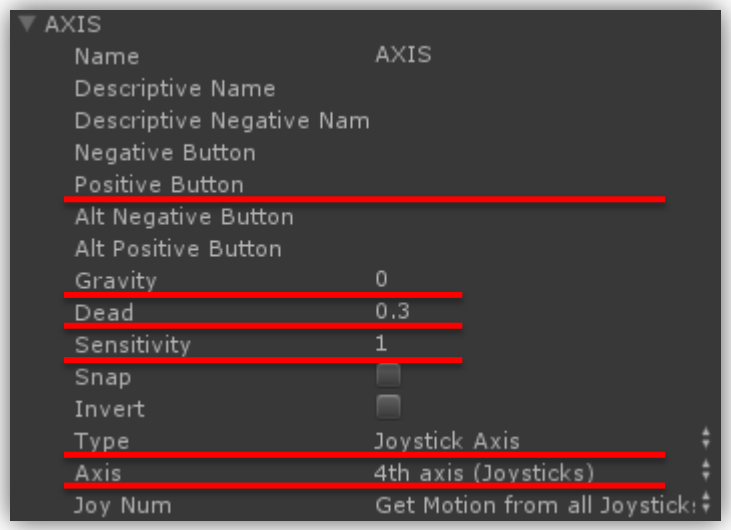
GamePad Controller (C# Script):

Se puede usar Game Controller de Generico, Xbox y Playstation para poder controlar las acciones de nuestro juego re mapeando los botones del control y sustituirlos por el uso del teclado y mouse, generando juegos con orientación a consolas o PC NextGen. **Edit>Project Settings>Inputs. → Configuracion de Inputs: Gravity: 0.0 | Dead: 0.3 | Sensitivity: 1 ←**

BOTONES (Buttons del 0 – 11 en Y Axis)



AXIS (3TH, 4TH, 5TH, 6TH 7TH, 8TH, X & Y en Axis)



UNITY INPUTS	SCRIPT KEYCODE
joystick 1 button 0	KeyCode.JoystickButon0
joystick 1 button 1	KeyCode.JoystickButon1
joystick 1 button 2	KeyCode.JoystickButon2
joystick 1 button 3	KeyCode.JoystickButon3
joystick 1 button 4	KeyCode.JoystickButon4
joystick 1 button 5	KeyCode.JoystickButon5
joystick 1 button 6	KeyCode.JoystickButon6
joystick 1 button 7	KeyCode.JoystickButon7
joystick 1 button 8	KeyCode .JoystickButon8
joystick 1 button 9	KeyCode .JoystickButon9
joystick 1 button 10	KeyCode .JoystickButon10
joystick 1 button 11	KeyCode .JoystickButon11

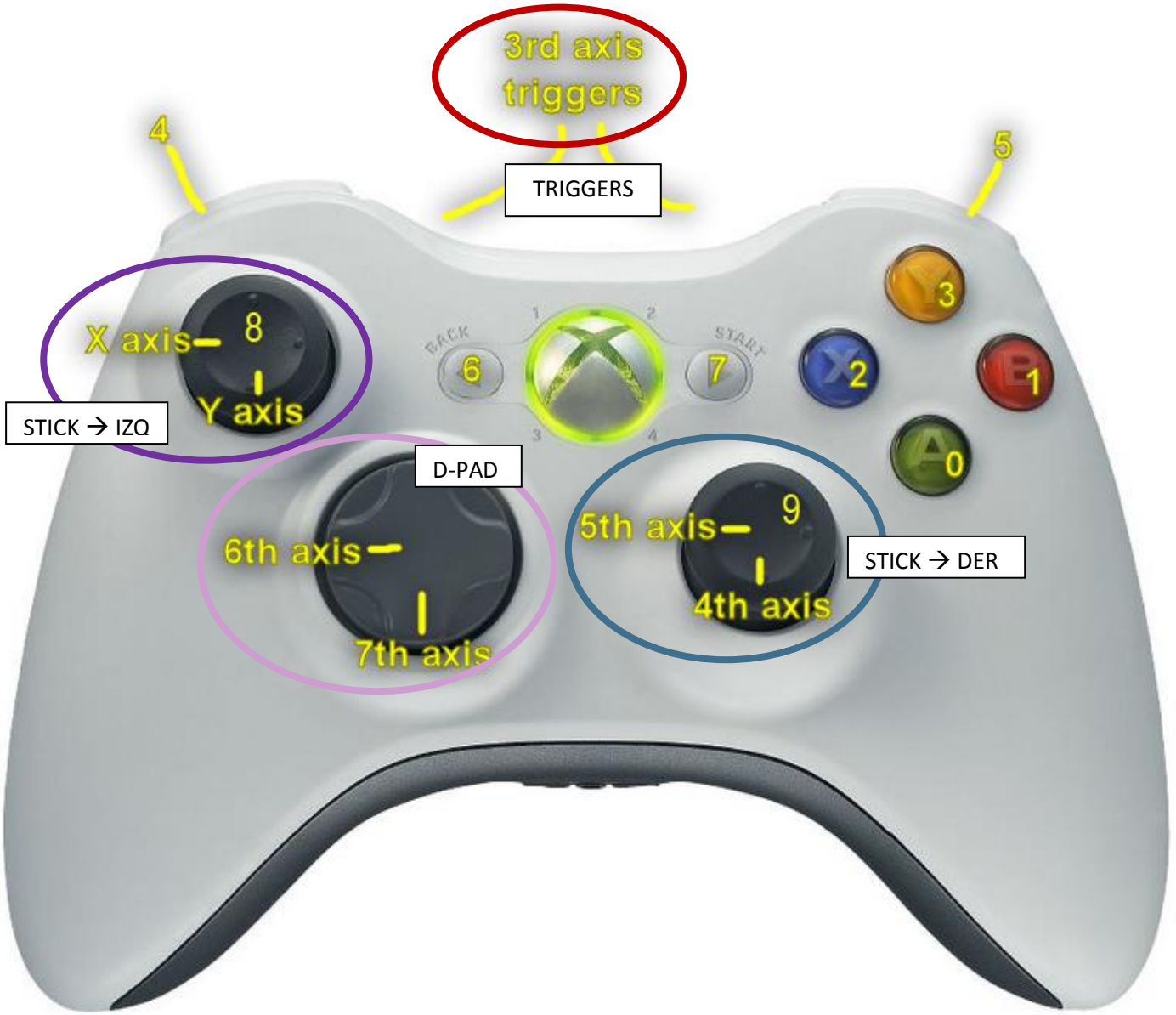
Gravity: Cuan rapido se centrara el Input (teclado/Mouse).

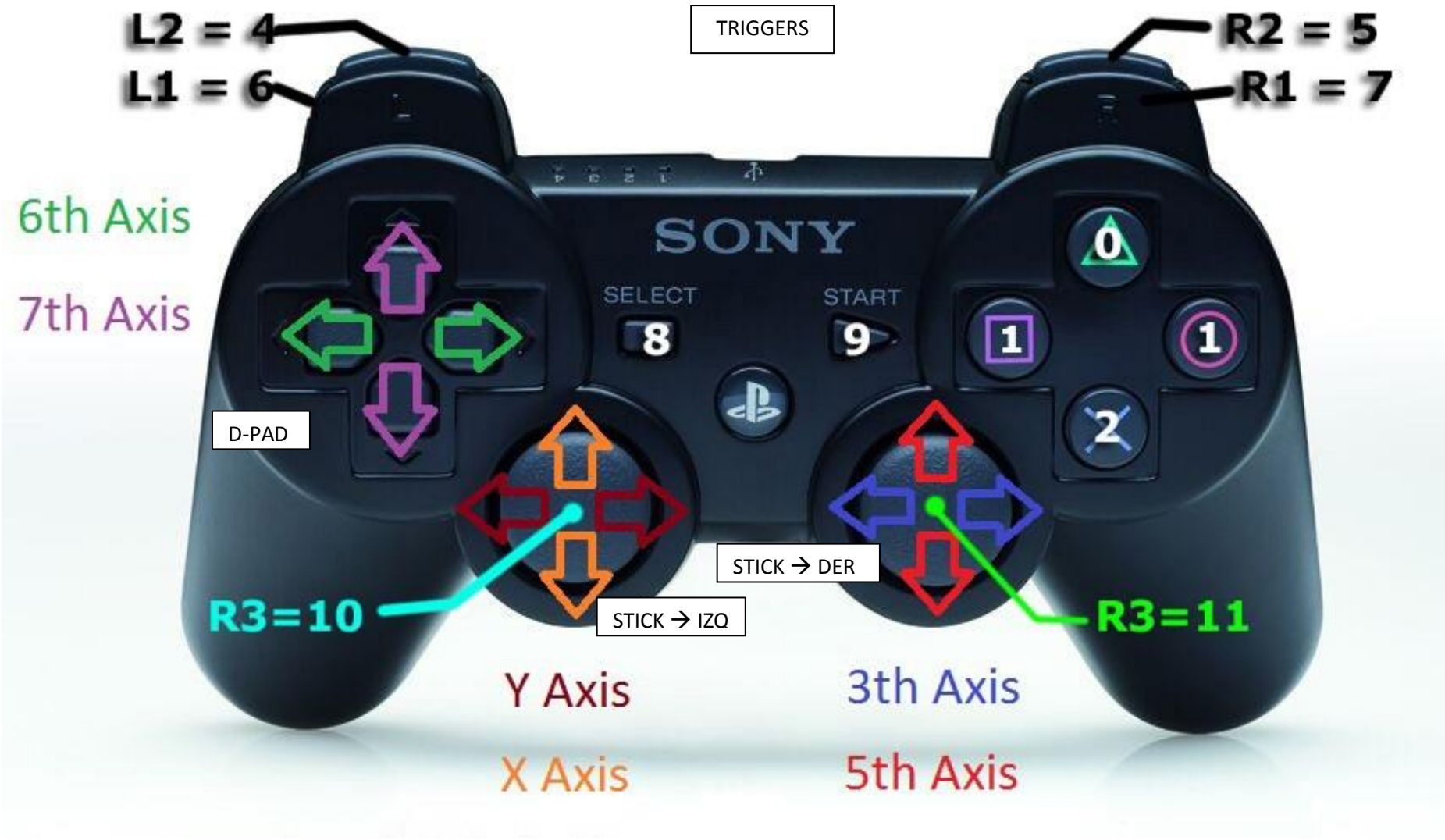
Dead: Los valores positivos o negativos que son menos que este número se registrarán como cero (Joysticks).

Sensivity: Para Teclado/Joysticks, controla la suavidad, en valores altos es rápido y en bajos son suaves (GetAxis -1 a 0 a 1).

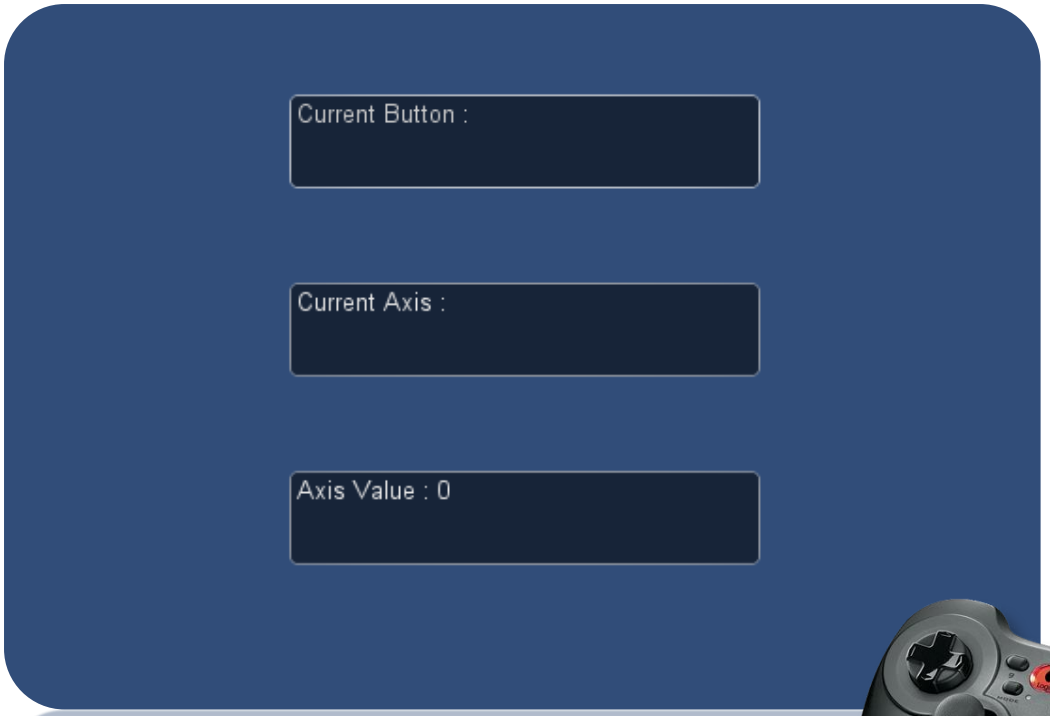
```
Input.GetAxis ("AXIS")    <= | >= | != | ==   → float
Input.GetButtonDown ("AXIS")                  → Boolean
Input.GetButtonUp ("AXIS")                    → Boolean
```

Nota: Copiar el archivo "**InputManager.asset**" del Game Controller que deseas usar (Xbox/Play/Generico).


3rd axis
triggers
TRIGGERS
4
5
X axis
8
Y axis
STICK → IZQ
BACK
6
START
7
3
2
1
0
D-PAD
6th axis
7th axis
5th axis
9
4th axis
STICK → DER


L2 = 4
L1 = 6
TRIGGERS
R2 = 5
R1 = 7
6th Axis
7th Axis
SONY
SELECT
8
START
9
0
1
1
2
D-PAD
STICK → DER
R3=10
STICK → IZQ
R3=11
Y Axis
3th Axis
X Axis
5th Axis

DETECTOR DE MAPEADO DE CONTROLES GENERICOS PARA VIDEOJUEGOS:



Lo que haremos es cambiar los controles de Disparar, Saltar, Mover y rotar cámara directamente con el Game Controller.

1. Abrimos "AI_Rocket_Launcher" en la línea del Input agregamos lo siguiente:

AI_Rocket_Launcher

```
if (Input.GetKeyDown (KeyCode.Mouse0) || Input.GetKeyDown (KeyCode.Joystick1Button7)) {}
```

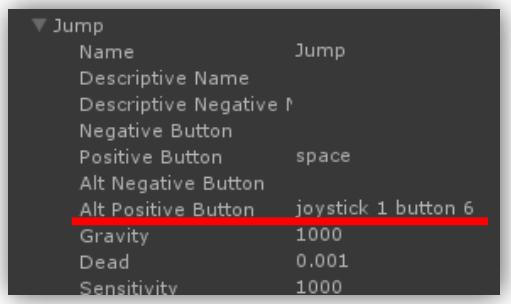
Nota: Los inputs los estoy haciendo con base a mi Game Controller Generico.

2. Abrimos "AI_Pause" en la línea del Input agregamos lo siguiente:

AI_Pause

```
if (Input.GetKeyDown (KeyCode.Escape) || Input.GetKeyDown (KeyCode.JoystickButton9) ) {}
```

3. Agregamos en Input existente de Unity en **Jump → joystick 1 button 6** para brincar.



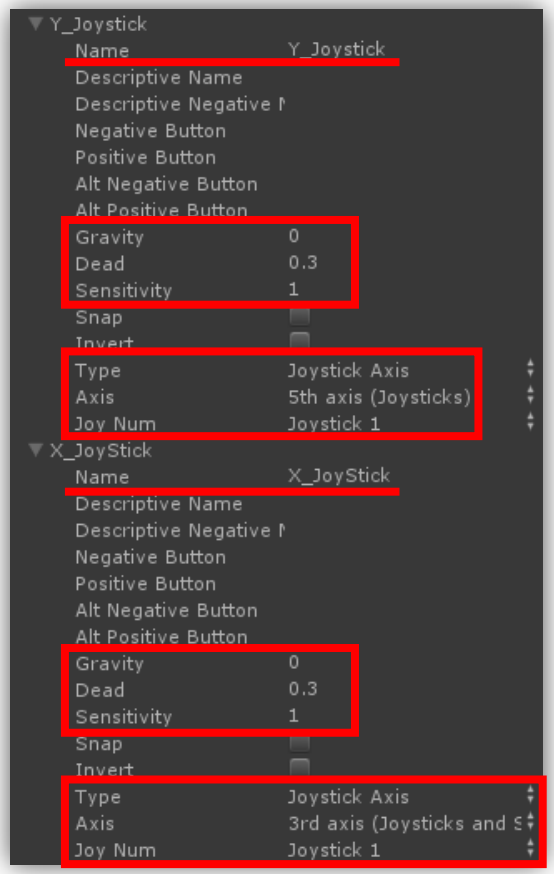
INPUTS DE UNITY – Jump

Edit>Project Settings>Input:

4. Creamos 2 Inputs nuevos con los siguientes valores:

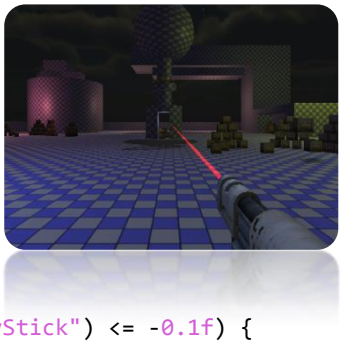
X_JoyStick: Gravity: 0 | Dead: 0.3 | Sensitivity: 1 | Type: Joystick Axis | Axis: 3rd axis.

Y_JoyStick: Gravity: 0 | Dead: 0.3 | Sensitivity: 1 | Type: Joystick Axis | Axis: 5th axis.



5. Ahora lo que hacemos esc rear un Script **"AI_JoystickRotate"** y se asigna al **Player**.

AI_JoystickRotate

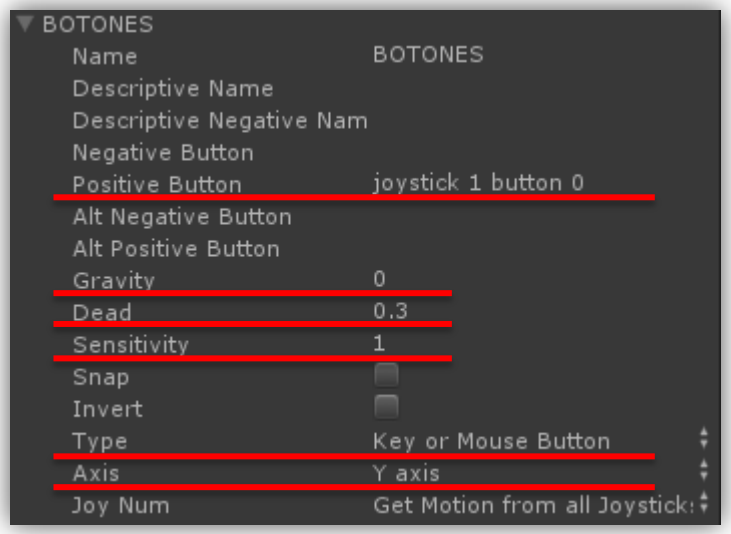


```
public GameObject vCamera;
public float SpeedJoystick = 100f;
private float RotationY = 0f;
public int MaxY = 45;
public int MinY = -45;

void Update ()
{
        //JOYSTICK SIDES – CONTINUO (Rotate).
        if (Input.GetAxis ("X_JoyStick") >= 0.1f || Input.GetAxis ("X_JoyStick") <= -0.1f) {
           transform.Rotate (Vector3.up * Input.GetAxis ("X_JoyStick") * SpeedJoystick * Time.deltaTime);
        }
        //JOYSTICK UPDOWN - NO CONTINUO (localEulerAngles).
        if (Input.GetAxis ("Y_Joystick") >= 0.1f || Input.GetAxis ("Y_Joystick") <= -0.1f) {
                RotationY += Input.GetAxis ("Y_Joystick") * SpeedJoystick * Time.deltaTime;
                RotationY = Mathf.Clamp (RotationY, MinY, MaxY);
                vCamera.transform.localEulerAngles = Vector3.right * RotationY;
        }
}
```

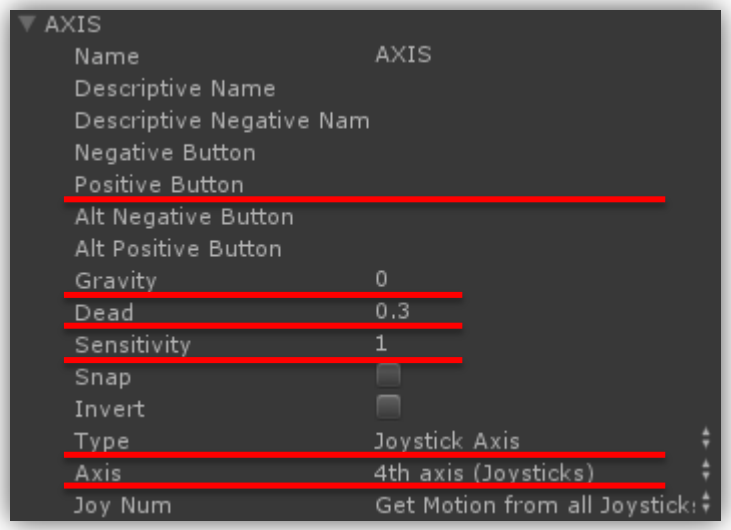
GamePad Controller (Java Script):

Se puede usar Game Controller de Generico, Xbox y Playstation para poder controlar las acciones de nuestro juego re mapeando los botones del control y sustituirlos por el uso del teclado y mouse, generando juegos con orientación a consolas o PC NextGen.
Edit>Project Settings>Inputs. → Configuracion de Inputs: Gravity: 0.0 | Dead: 0.3 | Sensitivity: 1 ←

BOTONES (Buttons del 0 – 11 en Y Axis)



AXIS (3TH, 4TH, 5TH, 6TH 7TH, 8TH, X & Y en Axis)



UNITY INPUTS	SCRIPT KEYCODE
joystick 1 button 0	KeyCode.JoystickButon0
joystick 1 button 1	KeyCode.JoystickButon1
joystick 1 button 2	KeyCode.JoystickButon2
joystick 1 button 3	KeyCode.JoystickButon3
joystick 1 button 4	KeyCode.JoystickButon4
joystick 1 button 5	KeyCode.JoystickButon5
joystick 1 button 6	KeyCode.JoystickButon6
joystick 1 button 7	KeyCode.JoystickButon7
joystick 1 button 8	KeyCode .JoystickButon8
joystick 1 button 9	KeyCode .JoystickButon9
joystick 1 button 10	KeyCode .JoystickButon10
joystick 1 button 11	KeyCode .JoystickButon11

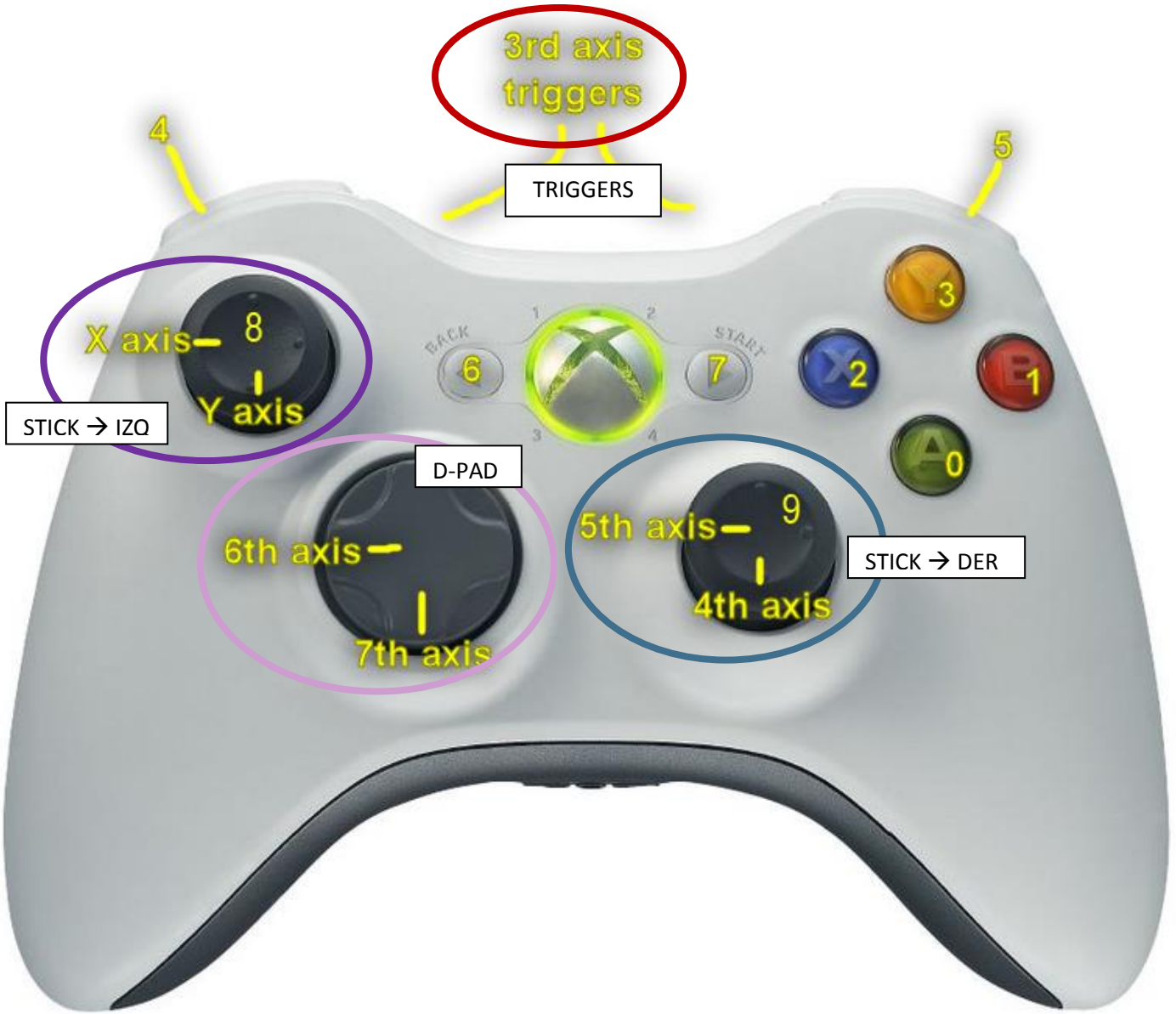
Gravity: Cuan rapido se centrara el Input (teclado/Mouse).

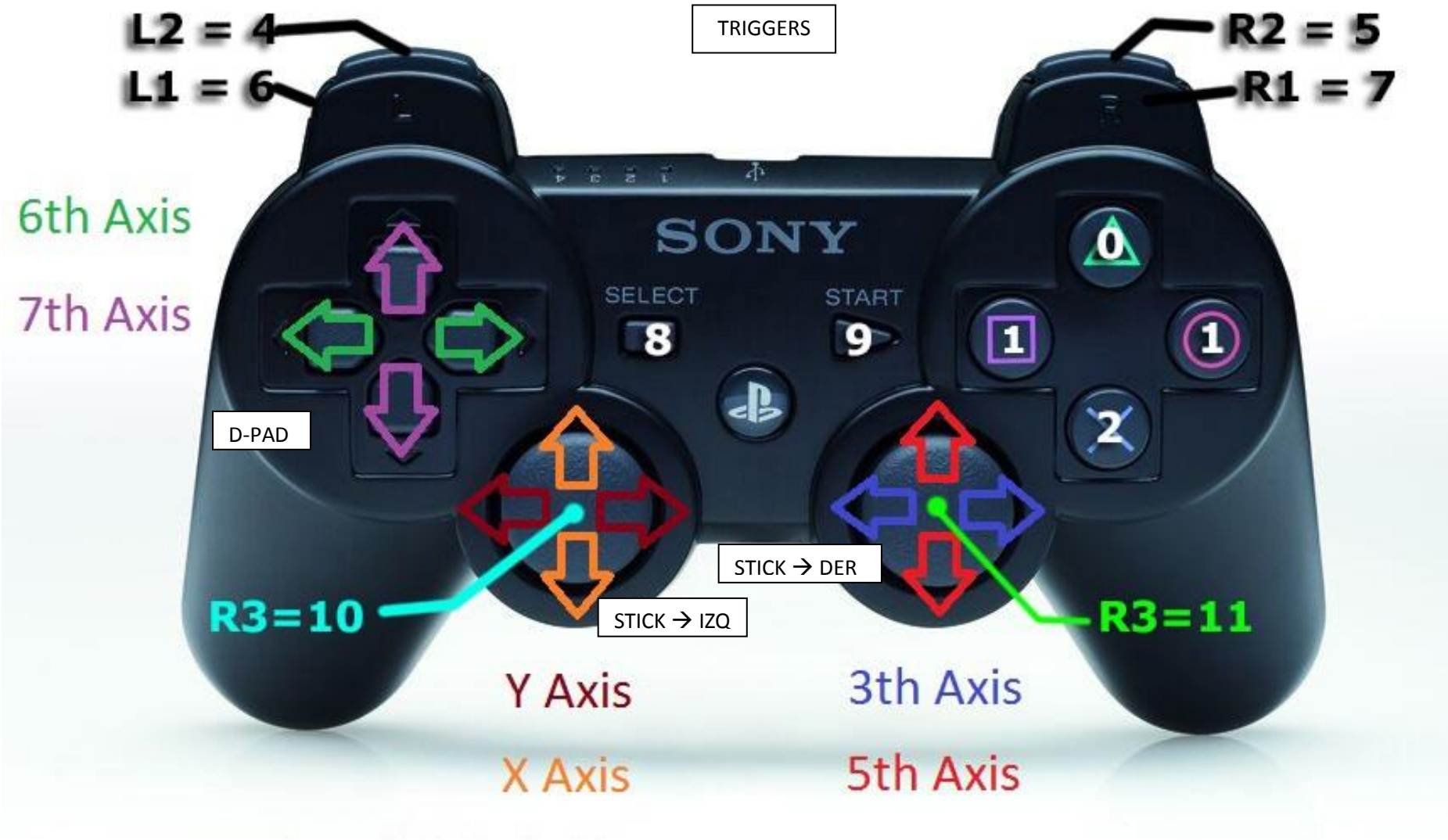
Dead: Los valores positivos o negativos que son menos que este número se registrarán como cero (Joysticks).

Sensivity: Para Teclado/Joysticks, controla la suavidad, en valores altos es rápido y en bajos son suaves (GetAxis -1 a 0 a 1).

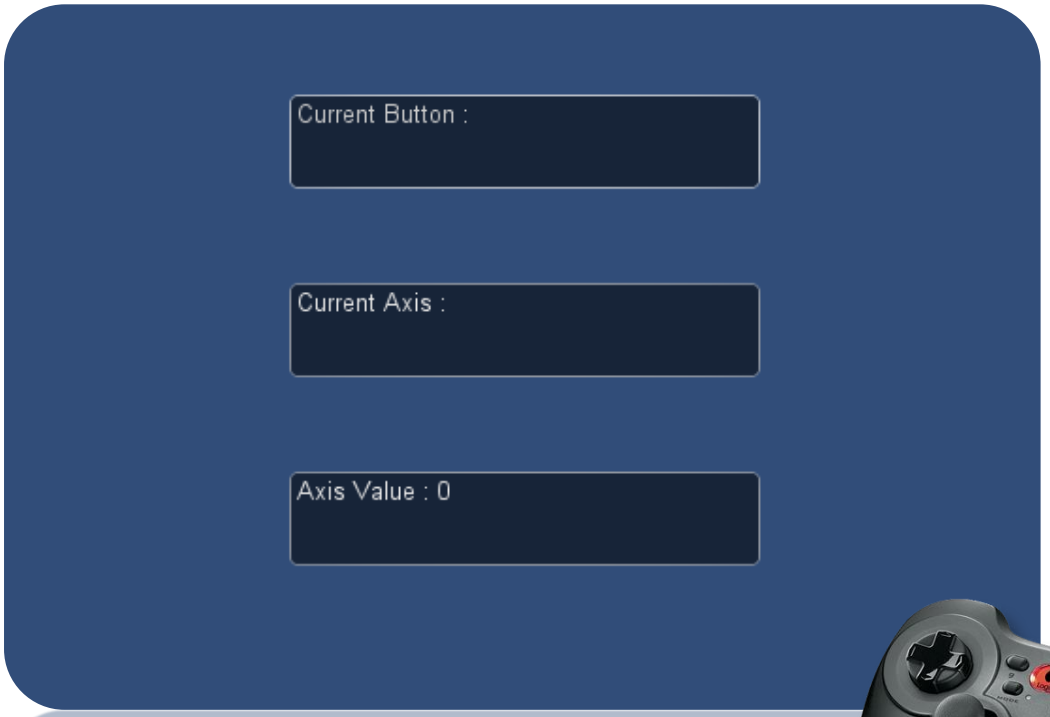
```
Input.GetAxis ("AXIS")     <= | >= | != | ==   → float
Input.GetButtonDown ("AXIS")                    → Boolean
Input.GetButtonUp ("AXIS")                      → Boolean
```

Nota: Copiar el archivo "**InputManager.asset**" del Game Controller que deseas usar (Xbox/Play/Generico).


3rd axis
triggers
TRIGGERS
4
5
X axis
8
Y axis
STICK → IZQ
BACK
6
START
7
3
2
1
0
D-PAD
6th axis
7th axis
5th axis
9
4th axis
STICK → DER


TRIGGERS
L2 = 4
L1 = 6
R2 = 5
R1 = 7
6th Axis
7th Axis
SONY
SELECT
8
START
9
0
1
1
2
D-PAD
STICK → DER
R3=10
STICK → IZQ
R3=11
Y Axis
3th Axis
X Axis
5th Axis

DETECTOR DE MAPEADO DE CONTROLES GENERICOS PARA VIDEOJUEGOS:



Lo que haremos es cambiar los controles de Disparar, Saltar, Mover y rotar cámara directamente con el Game Controller.

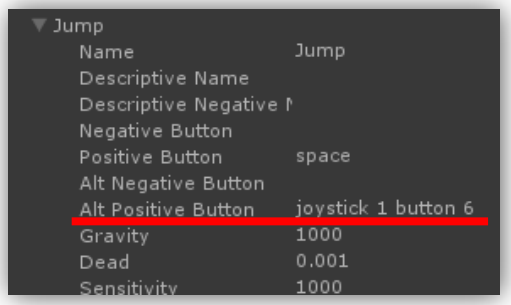
1. Abrimos "AI_Rocket_Launcher" en la línea del Input agregamos lo siguiente:

AI_Rocket_Launcher
```
if (Input.GetKeyDown (KeyCode.Mouse0) || Input.GetKeyDown (KeyCode.Joystick1Button7)) {}
```
Nota: Los inputs los estoy haciendo con base a mi Game Controller Generico.

2. Abrimos "AI_Pause" en la línea del Input agregamos lo siguiente:

AI_Pause
```
if (Input.GetKeyDown (KeyCode.Escape) || Input.GetKeyDown (KeyCode.JoystickButton9) ) {}
```

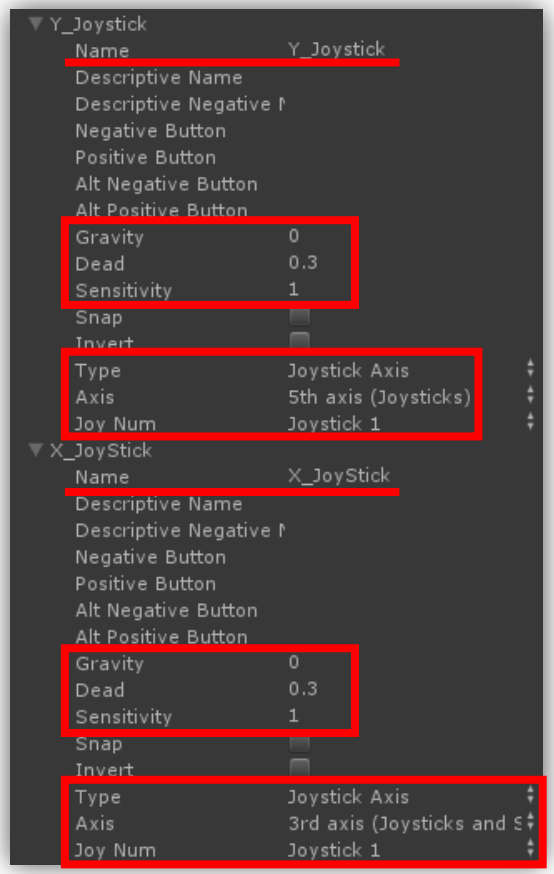
3. Agregamos en Input existente de Unity en **Jump → joystick 1 button 6** para brincar.



INPUTS DE UNITY – Jump
Edit>Project Settings>Input:

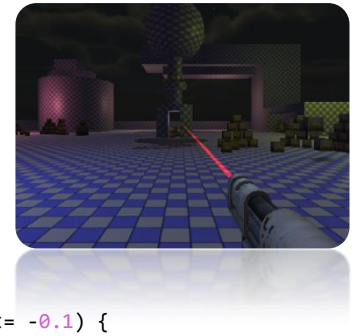
4. Creamos 2 Inputs nuevos con los siguientes valores:

X_JoyStick: Gravity: 0 | Dead: 0.3 | Sensitivity: 1 | Type: Joystick Axis | Axis: 3rd axis.
Y_JoyStick: Gravity: 0 | Dead: 0.3 | Sensitivity: 1 | Type: Joystick Axis | Axis: 5th axis.



5. Ahora lo que hacemos esc rear un Script **"AI_JoystickRotate"** y se asigna al **Player**.

AI_JoystickRotate



```
var vCamera : GameObject;
var SpeedJoystick float = 100f;
var RotationY : float = 0f;
var MaxY : int = 45;
var MinY : int = -45;

function Update ()
{
        //JOYSTICK SIDES - CONTINUO (Rotate).
        if (Input.GetAxis ("X_JoyStick") >= 0.1 || Input.GetAxis ("X_JoyStick") <= -0.1) {
           transform.Rotate (Vector3.up * Input.GetAxis ("X_JoyStick") * SpeedJoystick * Time.deltaTime);
        }
        //JOYSTICK UPDOWN - NO CONTINUO (localEulerAngles).
        if (Input.GetAxis ("Y_Joystick") >= 0.1 || Input.GetAxis ("Y_Joystick") <= -0.1) {
                RotationY += Input.GetAxis ("Y_Joystick") * SpeedJoystick * Time.deltaTime;
                RotationY = Mathf.Clamp (RotationY, MinY, MaxY);
                vCamera.transform.localEulerAngles = Vector3.right * RotationY;
        }
}
```

1.46 SAVE & LOAD INFORMATION.

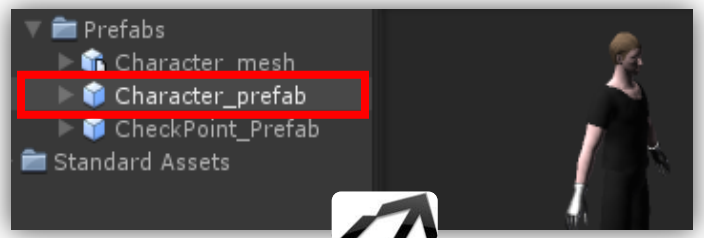
Save & Load Information – PlayerPrefs (C# Script):

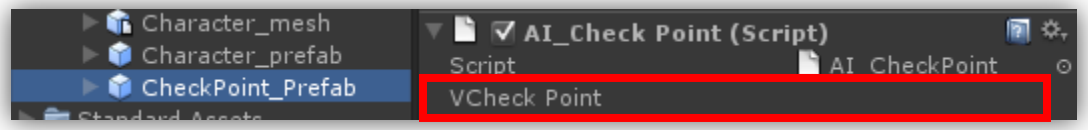
Para poder crear la opción de guardar una escena de un juego sobre la partida en la que estamos jugando, necesitamos usar una función llamada **PlayerPrefs**, la cual nos permite guardar **solamente variables del juego**, en donde estas serán variables con información de lo que queremos guardar como: posición del personaje, vida, sangre, ítems, checkpoint u opciones activadas del juego.

1. **General**: Se quiere que el personaje cuando pase por el ultimo Checkpoint activado y se use la opción de Salvar escena, este lo haga a partir del último Checkpoint activado, así al reiniciar el juego el personaje inicie en la posición del Checkpoint Salvado.
2. Cargamos el paquete "**Save_Scene_Start"**, en **data** en **Prefabs** esta "**Character_Prefab**", este lo insertamos en la escena.
3. En **Data**>**Checkpoint_Prefab** crearemos y asignaremos un script llamado "**AI_CheckPoint**", así todos los prefab de este tendrán el script dentro de la escena y no se tendrá que asignar de uno por uno el Script.

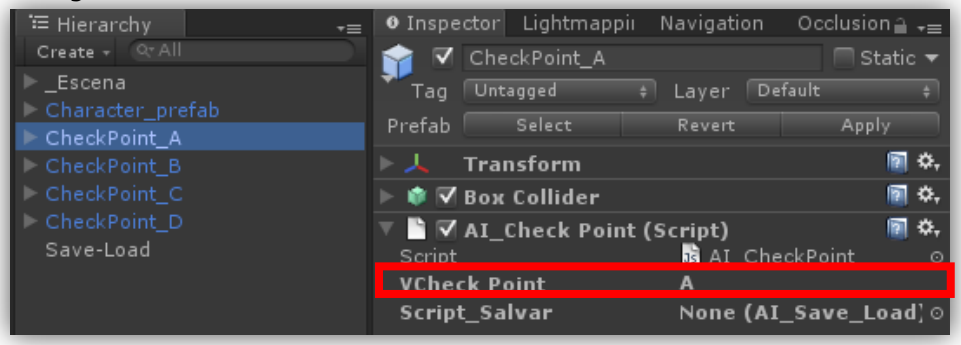
AI_CheckPoint

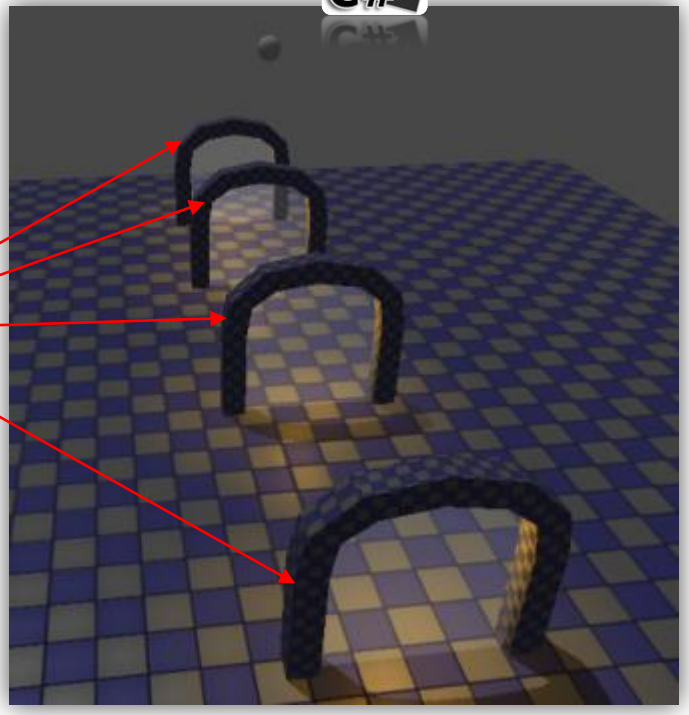
```
//Variable para saber que Checkpoint ha sido activado
public string vCheckPoint;
//Variable para cargar el objeto a donde mandaremos la info del CP activado.
public AI_Save_Load Script_Salvar;

void OnTriggerEnter () {
    print ("Checkpoint >"+vCheckPoint+"< Activado");
    // Mandar al gameObject Esfera el CP activado a su variable para guardar

}
```





4. Ahora lo que haremos es que cada **CheckPoint** en su script (escena) le asignaremos una letra de identificación: **VCheck Point** → "**A a D**".



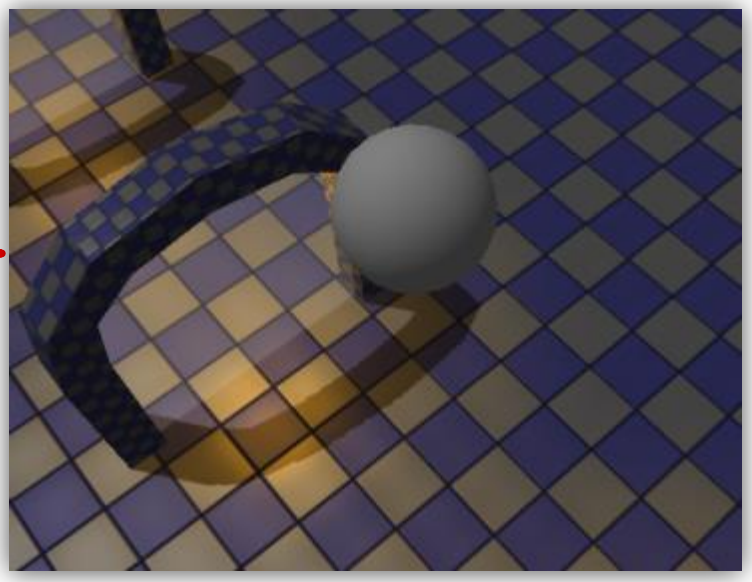


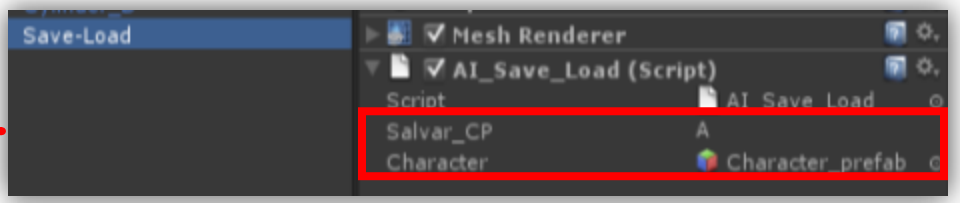
```
static function SetString (key : String, value : String) : void
```
Save Game Values: Windows: HKeyCurrentUser\Software\[company name]\[product name] | Mac: ~/Library/Preferences in a file named unity.[company name].[product name].

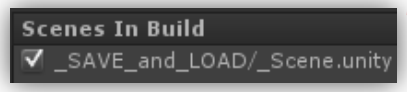
5. Creamos "**KeyCodes" → "Salvar" (Click Izq)** y "**Cargar" (Click Der).**
6. Creamos un script llamado "**AI_Save_Load**" y lo asignamos a la esfera de arriba que servirá para mandarle información de cuando se active cada Checkpoint.

AI_Save_Load

```
//Variable que indica que Checkpoint se guardara
public  string Salvar_CP = "A";

//Variable para insertar al personaje para después moverlo a otro CP.
public GameObject Character;

Opciones de PlayerPrefs: Set/GetInt, Set/GetFloat, Set/GetString,
DeleteKey, DeleteAll, Save (Save in Quit or Crash).

void Update () {
    //SAVE - Click Izquierdo, creamos Info para ahí almacenar la info
    if (Input.GetKeyDown (KeyCode.Mouse0) ) {

          PlayerPrefs.SetString("info", Salvar_CP);
          print ("CheckPoint: "+Salvar_CP+" Salvado");
    }

    //LOAD - Click Derecho, ReCargamos la  escena
    if (Input.GetKeyDown (KeyCode.Mouse1) ){
          Application.LoadLevel ("_Scene");
    }
}
```





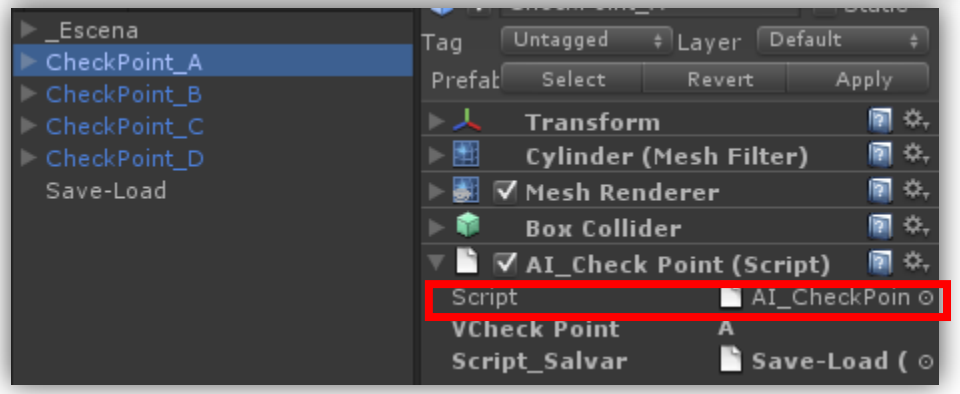


7. Cargamos nuestro archivo de **"_Scene"** en el **Build Settings**.

8. La Variable de **Salvar_CP**, le asignamos "**A**", ya cuando inicie la escena por primera vez será a partir de este CheckPoint.

9. En la variable "**Character**" cargamos al personaje de la escena, para después mover esta variable a cada CP activado.

10. Ahora que esta creado el Script "**AI_Save_Load**", haremos que cada **CheckPoint (A, B, C, D)** en su variable de "**Script_Salvar**" carguen al script de la Esfera: **"AI_Save_Load".**

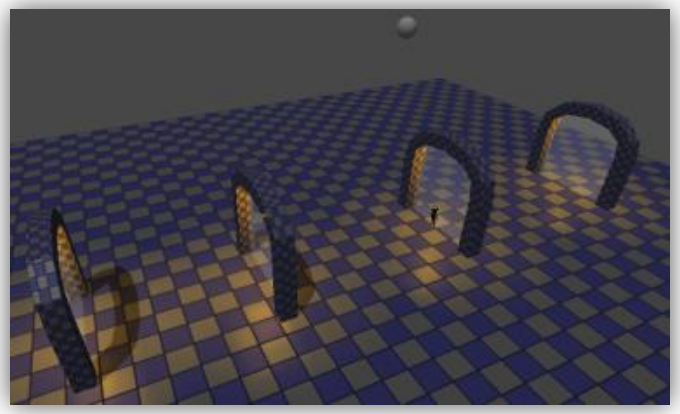


```
static function GetString (key : String, defaultValue : String = "") : String
```

```
// PARA HACER QUE EL PERSONAJE INICIE DONDE SE CARGO EL ULTIMO CP.
void Start (){
	CharacterMove ();	Ó	CharacterInstance ();
}
```

Opciones PlayerPrefs: Get/SetInt, Get/SetFloat, Get/SetString, DeleteAll, DeleteKey, Save.

OPCION A: CHARACTERMOVE → MOVER AL PERSONAJE QUE YA ESTA CREADO EN LA ESCENA.



```
void CharacterMove () {
	//Obtener el valor desde el registro de windows
	string Save_Load = PlayerPrefs.GetString("info");

	print ("Check Point Cargado: " + Save_Load);

	if (Save_Load == "A"){
		Character.transform.position = new Vector3 (0, 0, 45);
	}else if (Save_Load == "B"){
		Character.transform.position = new Vector3 (0, 0, 30);
	}else if (Save_Load == "C"){
		Character.transform.position = new Vector3 (0, 0, 17);
	}else if (Save_Load == "D"){
		Character.transform.position = new Vector3 (0, 0, 7);
	}
}
```

OPCION B: CHARACTERINSTANCE → CREAR PERSONAJE DINÁMICAMENTE Y MOVERLO.

```
void CharacterInstance () {
	//Obtener el valor desde el registro de windows
	string Save_Load = PlayerPrefs.GetString("info");

	print ("Check Point Cargado: " + Save_Load);

	if (Save_Load == "A"){
		Instantiate (Character, new Vector3 (0, 0, 45), transform.rotation);
	}else if (Save_Load == "B"){
		Instantiate (Character, new Vector3 (0, 0, 30), transform.rotation);
	}else if (Save_Load == "C"){
		Instantiate (Character, new Vector3 (0, 0, 17), transform.rotation);
	}else if (Save_Load == "D"){
		Instantiate (Character, new Vector3 (0, 0, 7), transform.rotation);
	}
}
```

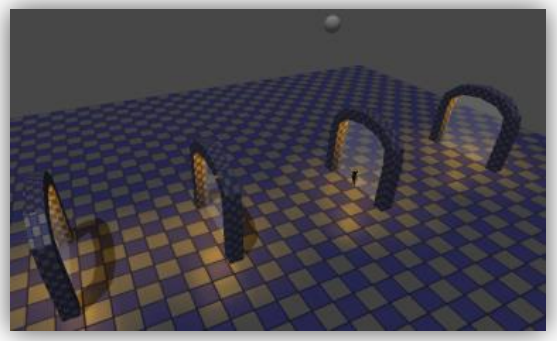
Nota: Permite NO destruir el objeto al iniciar (función de "Awake"), esto permite cargar otra escena sin que se pierdan los valores del CP.
`DontDestroyOnLoad ();` | `DontDestroyOnLoad (transform.gameObject);`

Save & Load Information – File Txt (C# Script):

Otra manera de salvar la información directamente en un archivo de forma Local es la siguiente:

1. Creamos una Escena nueva junto con un Empty Group y asignamos "**AI_Save_Load_File**" :

AI_Save_Load_File



```
//Permite usar librerias del sistema (File).
using System.IO;

public string vName = "Mago";
public int vPoitions = 49;
public float vEnergy = 99.9f;
public bool vShell = false;

void Update ()
{
	if (Input.GetKeyDown (KeyCode.Mouse0)) {
		Save ();
	}
	if (Input.GetKeyDown (KeyCode.Mouse1)) {
		Load ();
	}
}

void Save ()
{
	//Crear Carpeta si ya esta ignorarlo
	if (!Directory.Exists (Application.dataPath + "/SaveLoad/Save/")) {
		Directory.CreateDirectory (Application.dataPath + "/SaveLoad/Save/");
	}
	//Crear archivo txt con el valore de la variable
	File.WriteAllText (Application.dataPath + "/SaveLoad/Save/Save.txt", vName +"\n"+ vPoitions +"\n"+ vEnergy +"\n" + vShell);

	print ("Archivo Salvado");
}

	void Load ()
{
	//Cargar el archivo que contiene multi lineas con valores en tipo string a convertir.
	string[] FileTxt = File.ReadAllLines (Application.dataPath + "/SaveLoad/Save/Save.txt");
	vPoitions = int.Parse (FileTxt [1]);
	vEnergy = float.Parse (FileTxt [2]);
	vShell = bool.Parse (FileTxt [3]);

	print ("Nombre: " + (vName+" Deo") + ", Pociones: " + (vPoitions+1) + ", Energia: " + (vEnergy+0.1f) + ", Escudo: " + (!vShell));
}
```

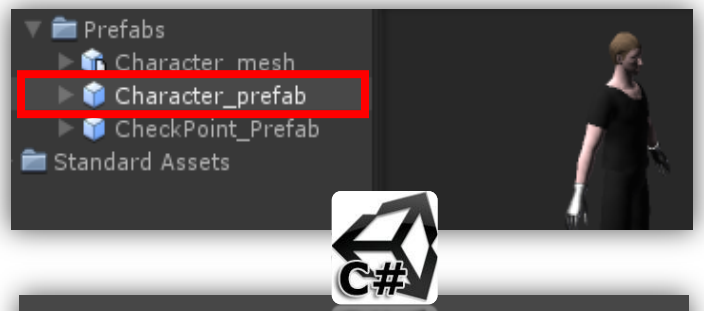
Save & Load Information Scene – PlayerPrefs (Java Script):

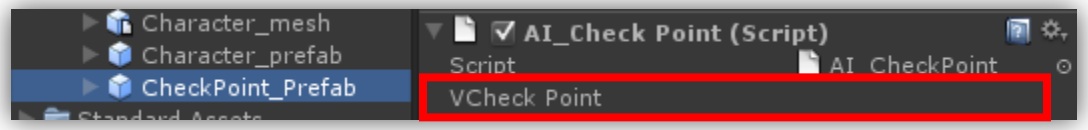
Para poder crear la opción de guardar una escena de un juego sobre la partida en la que estamos jugando, necesitamos usar una función llamada **PlayerPrefs**, la cual nos permite guardar **solamente variables del juego**, en donde estas serán variables con información de lo que queremos guardar como: posición del personaje, vida, sangre, ítems, checkpoint u opciones activadas del juego.

1. **General**: Se quiere que el personaje cuando pase por el ultimo Checkpoint activado y se use la opción de Salvar escena, este lo haga a partir del último Checkpoint activado, así al reiniciar el juego el personaje inicie en la posición del Checkpoint Salvado.
2. Cargamos el paquete "**Save_Scene_Start"**, en **data** en **Prefabs** esta "**Character_Prefab**", este lo insertamos en la escena.
3. En **Data**>**Checkpoint_Prefab** crearemos y asignaremos un script llamado "**AI_CheckPoint**", así todos los prefab de este tendrán el script dentro de la escena y no se tendrá que asignar de uno por uno el Script.

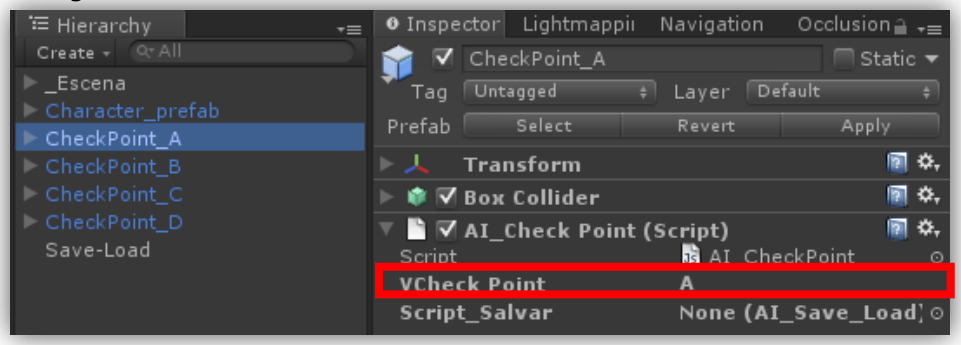
AI_CheckPoint

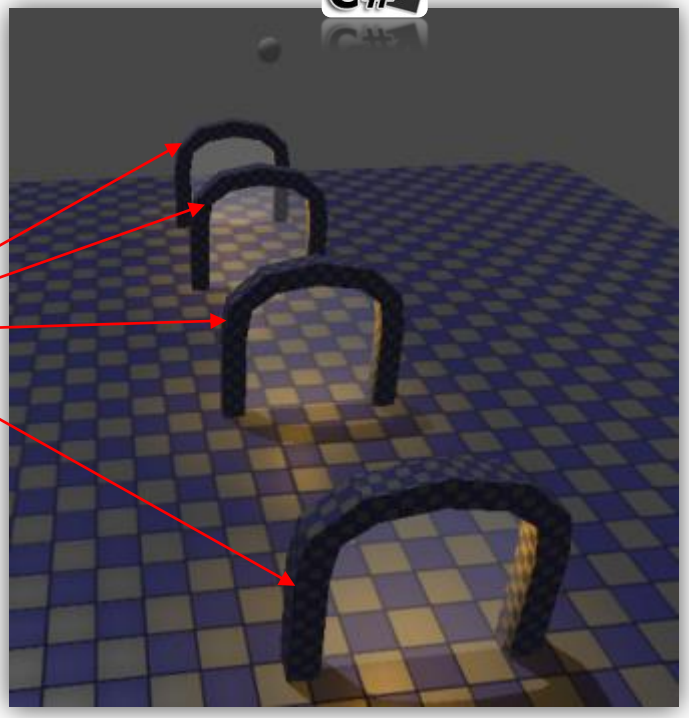
```
//Variable para saber que Checkpoint ha sido activado
var vCheckPoint : string;
//Variable para cargar el objeto a donde mandaremos la info del CP activado.
var Script_Salvar : AI_Save_Load;

function OnTriggerEnter () {
    print ("Checkpoint >"+vCheckPoint+"< Activado");
    // Mandar al gameObject Esfera el CP activado a su variable para guardar

}
```





4. Ahora lo que haremos es que cada **CheckPoint** en su script (escena) le asignaremos una letra de identificación: **VCheck Point** → "**A a D**".



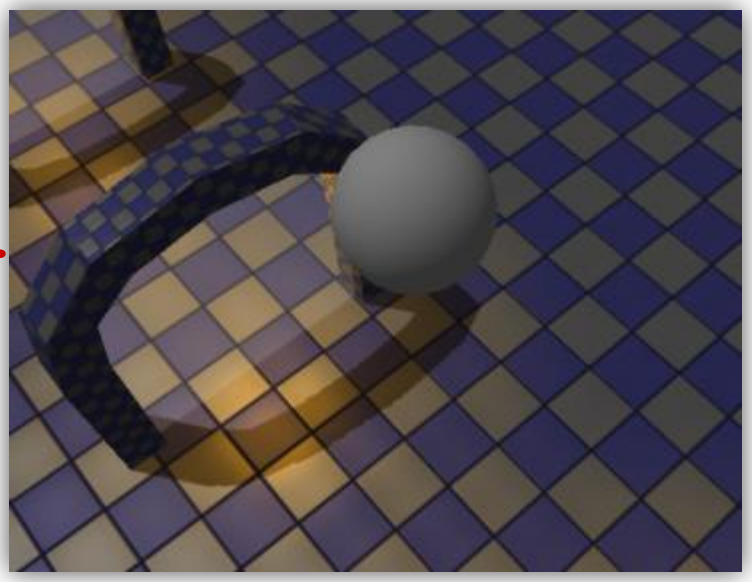


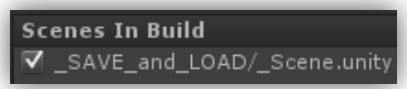
`static function SetString (key : String, value : String) : void`
Save Game Values: Windows: HKeyCurrentUser\Software\[company name]\[product name] | Mac: ~/Library/Preferences in a file named unity.[company name].[product name].

5. Creamos "**KeyCodes" → "Salvar" (Click Izq)** y "**Cargar" (Click Der).**
6. Creamos un script llamado "**AI_Save_Load**" y lo asignamos a la esfera de arriba que servirá para mandarle información de cuando se active cada Checkpoint.

AI_Save_Load

```
//Variable que indica que Checkpoint se guardara
var Salvar_CP : string = "A";

//Variable para insertar al personaje para después moverlo a otro CP.
var Character : GameObject;

Opciones de PlayerPrefs: Set/GetInt, Set/GetFloat, Set/GetString,
DeleteKey, DeleteAll, Save (Save in Quit or Crash).

function Update () {
    //SAVE - Click Izquierdo, creamos Info para ahí almacenar la info
    if (Input.GetKeyDown (KeyCode.Mouse0) ) {

        PlayerPrefs.SetString("info", Salvar_CP);
        print ("CheckPoint: "+Salvar_CP+" Salvado");
    }

    //LOAD - Click Derecho, ReCargamos la  escena
    if (Input.GetKeyDown (KeyCode.Mouse1) ){
        Application.LoadLevel ("_Scene");
    }
}
```



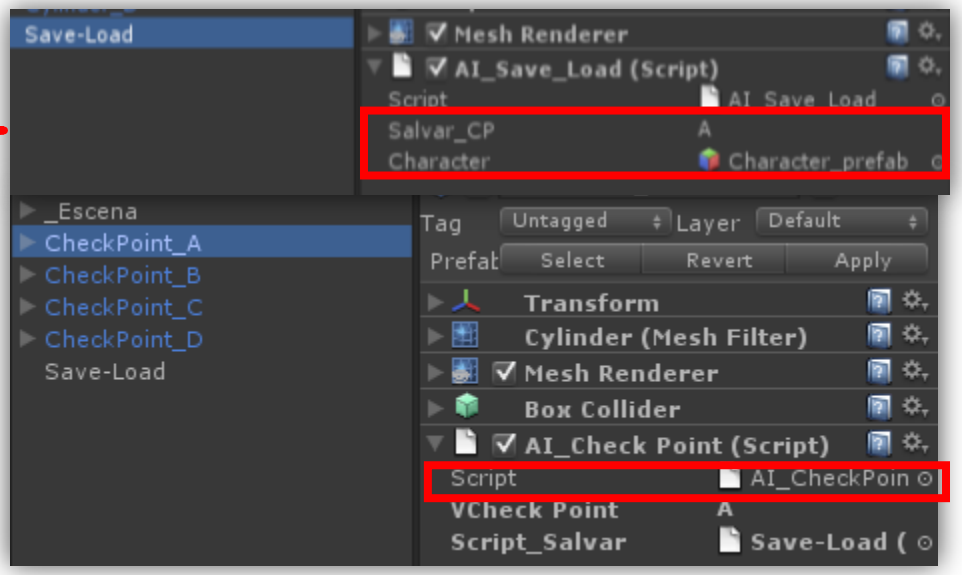


7. Cargamos nuestro archivo de **"_Scene"** en el **Build Settings**.

8. La Variable de **Salvar_CP**, le asignamos "**A",** ya cuando inicie la escena por primera vez será a partir de este CheckPoint.

9. En la variable "**Character"** cargamos al personaje de la escena, para después mover esta variable a cada CP activado.

10. Ahora que esta creado el Script "**AI_Save_Load**", haremos que cada **CheckPoint (A, B, C, D)** en su variable de "**Script_Salvar**" carguen al script de la Esfera: **"AI_Save_Load".**

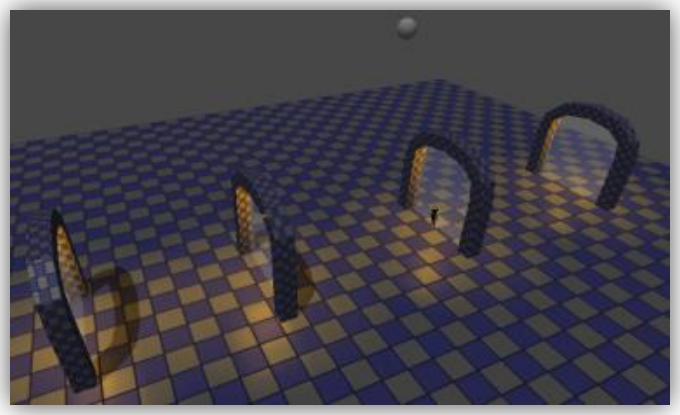


```
static function GetString (key : String, defaultValue : String = "") : String
```

```
// PARA HACER QUE EL PERSONAJE INICIE DONDE SE CARGO EL ULTIMO CP.
function Start (){
	CharacterMove ();	Ó	CharacterInstance ();
}
```

Opciones PlayerPrefs: Get/SetInt, Get/SetFloat, Get/SetString, DeleteAll, DeleteKey, Save.

OPCION A: CHARACTERMOVE → MOVER AL PERSONAJE QUE YA ESTA CREADO EN LA ESCENA.



```
function CharacterMove () {
	//Obtener el valor desde el registro de windows
	string Save_Load = PlayerPrefs.GetString("info");

	print ("Check Point Cargado: " + Save_Load);

	if (Save_Load == "A"){
		Character.transform.position = new Vector3 (0, 0, 45);
	}else if (Save_Load == "B"){
		Character.transform.position = new Vector3 (0, 0, 30);
	}else if (Save_Load == "C"){
		Character.transform.position = new Vector3 (0, 0, 17);
	}else if (Save_Load == "D"){
		Character.transform.position = new Vector3 (0, 0, 7);
	}
}
```

OPCION B: CHARACTERINSTANCE → CREAR PERSONAJE DINÁMICAMENTE Y MOVERLO.

```
function CharacterInstance () {
	//Obtener el valor desde el registro de windows
	string Save_Load = PlayerPrefs.GetString("info");

	print ("Check Point Cargado: " + Save_Load);

	if (Save_Load == "A"){
		Instantiate (Character, new Vector3 (0, 0, 45), transform.rotation);
	}else if (Save_Load == "B"){
		Instantiate (Character, new Vector3 (0, 0, 30), transform.rotation);
	}else if (Save_Load == "C"){
		Instantiate (Character, new Vector3 (0, 0, 17), transform.rotation);
	}else if (Save_Load == "D"){
		Instantiate (Character, new Vector3 (0, 0, 7), transform.rotation);
	}
}
```

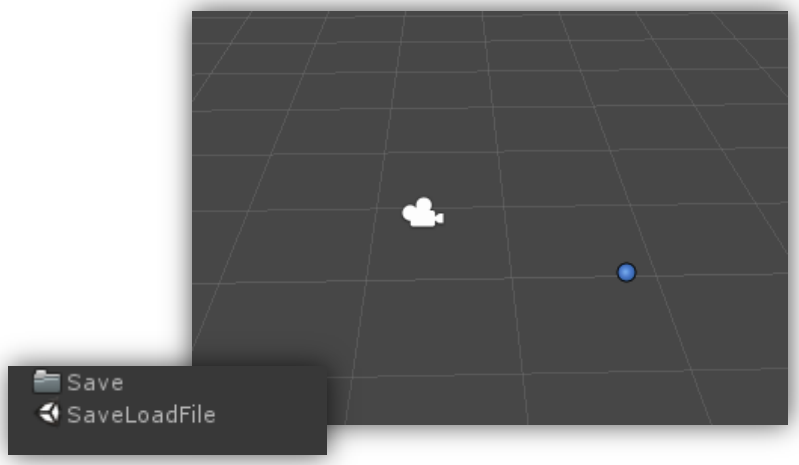
Nota: Permite NO destruir el objeto al iniciar (función de "Awake"), esto permite cargar otra escena sin que se pierdan los valores del CP.

```
DontDestroyOnLoad (); | DontDestroyOnLoad (transform.gameObject);
```

Save & Load Information Scene – File Txt (C# Script):

Otra manera de salvar la información directamente en un archivo de forma Local es la siguiente:

2. Creamos una Escena nueva junto con un Empty Group y asignamos "**AI_Save_Load_File**" :



AI Save Load File

```
//Permite usar librerias del sistema (File).
import System.IO;

var vName : String = "Mago";
var vPoitions : int = 49;
var vEnergy : float = 99.9f;
var vShell : boolean = false;

function Update () {
	if (Input.GetKeyDown (KeyCode.Mouse0)) {
		Save ();
	}
	if (Input.GetKeyDown (KeyCode.Mouse1)) {
		Load ();
	}
}

function Save () {
	//Crear Carpeta si ya esta ignorarlo
	if (!Directory.Exists (Application.dataPath + "/SaveLoad/Save/")) {
		Directory.CreateDirectory (Application.dataPath + "/SaveLoad/Save/");
	}
	//Crear archivo txt con el valore de la variable
	File.WriteAllText (Application.dataPath + "/SaveLoad/Save/Save.txt", vName +"\n"+ vPoitions +"\n"+ vEnergy +"\n" + vShell);

	print ("Archivo Salvado");
}

function Load () {
	//Cargar el archivo que contiene multi lineas con valores en tipo string a convertir.
	var FileTxt : String[] = File.ReadAllLines (Application.dataPath + "/SaveLoad/Save/Save.txt");
	vPoitions = int.Parse (FileTxt [1]);
	vEnergy = float.Parse (FileTxt [2]);
	vShell = boolean.Parse (FileTxt [3]);

     print ("Nombre: " + (vName + " Deo") + ", Pociones: " + (vPoitions + 1) + ", Energia: " + (vEnergy + 0.1f) + ", Escudo: " + (!vShell));
}
```

1.47 BUILDING GAME

Building Settings:

1. Cargar nuestra escena al Build Settings.
2. Entrar En Edit>Project Settings>Player: predeterminar nuestros atributos