UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning
Secretaría de Extensión y Cultura Universitaria

III.10. UNIDAD DIDÁCTICA

<< Desarrollo de Videojuegos >>

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

<< Modulo III UnityScript II >>

Programación

Presentación:

En esta unidad continuaremos viendo algunos conceptos fundamentales de programación junto con algunas herramientas para el trabajo con Game Objects y su ubicación espacial dentro de la plataforma de Unity.

Objetivos:

Que los participantes logren:

Asimilar el concepto de bucle

Comprendan la importancia del estudio de conceptos físicos matemáticos.

Puedan ubicarse espacialmente y realizar rotaciones y movimientos de objetos

Bloques temáticos:

1.- Bucles

2.- Física y matemática como herramientas de simulación.

3.- Vector2 y Vector3

4.- Transform

5.- Asignación de nombre en modo juego

6.- Asignar nombre a objeto que contiene el script.

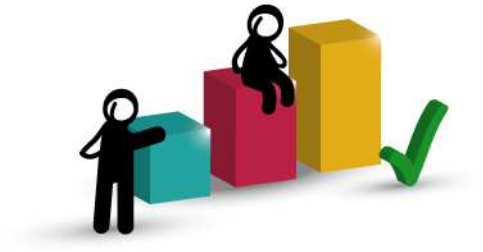
7.- Obtener nombre e identificador de un objeto

8.- Clonación de objetos

9.- Eliminar Objeto – Object.Destroy

10.- Encontrar Objeto de tipo – Object.FindObjectsOfType

11.- Crear lago



Consignas para el aprendizaje colaborativo

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

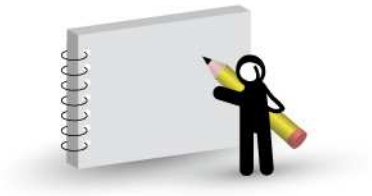
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

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

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

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

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



Tomen nota*

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

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

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

Bloque temático 1: Bucles

Siguiendo con la unidad anterior, vamos ahora a introducir el concepto de Bucles, los cuales son muy útiles cuando necesitamos realizar acciones repetidas mientras se cumpla una determinada condición. Un ejemplo de su uso podría ser el traer datos de una base de datos mientras que los elementos recuperados pertenezcan a un determinado grupo, o destruir un determinado tipo de objetos mientras un contador de tiempo no llega a un cierto valor.

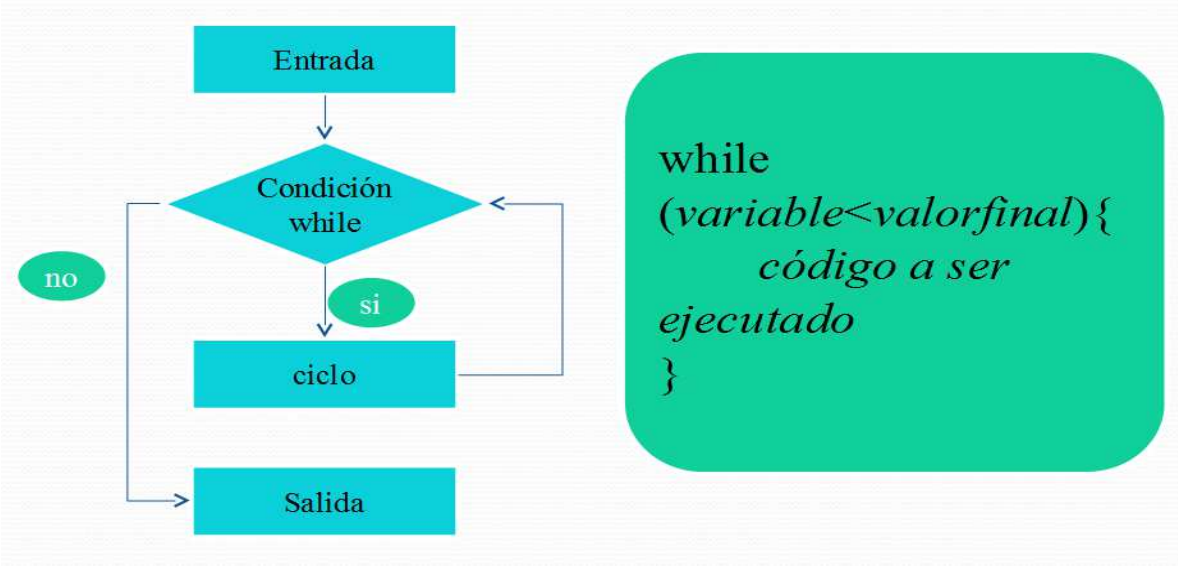
1.1. While

Primero analizaremos el bucle while, cuya estructura es como se indica a continuación:

1.- Se declara el bucle con la palabra While

2.- A continuación entre paréntesis de escribe la condición que se debe cumplir. Mientras esta condición se cumpla, el bucle se seguirá ejecutando.

3.- Entre llaves se escribe el código a ser ejecutado mientras se cumpla la condición.



Nota 1: Los bucles al igual que las estructuras de control se utilizan en todos los lenguajes de programación, y suelen presentar una sintaxis similar, salvo en Python en donde a diferencia de javaScript, java, php, ….. no se utilizan las llaves para delimitar el código.

Nota 2: Con el bucle While hay que tener un cuidado especial, ya que si la condición es verdadera siempre, el bucle se seguirá ejecutando indefinidamente.

Veamos un ejemplo, supongamos que tenemos una variable que puede tomar diferentes valores y que por algún motivo le damos un valor determinado (en el ejemplo 8), la estructura while en este caso permite que mientras el valor de la variable sea menor que 12 se ejecute el método Debug.Log() mostrando el valor que toma la variable en cada vuelta del bucle. Notar que al final del código el valor de la variable se incrementa en una unidad, de esta forma la segunda vez que ingresamos en la condición, el valor de la variable ha cambiado a 9, la tercera vez a 10, la cuarta a 11, y en la quinta no entraría ya que la condición no se cumple pues 12 no es menor que 12.

Recordar que **valor++** es igual a **valor = valor +1**

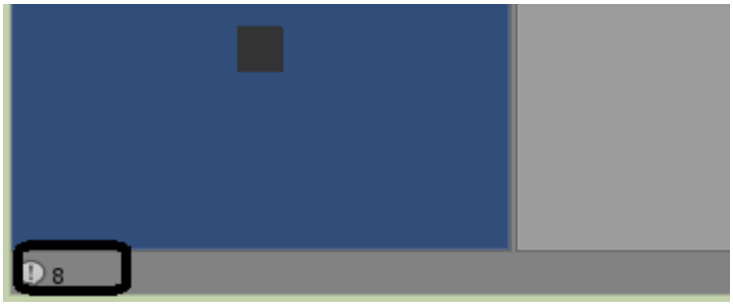
Notar que si al finalizar el código no incrementara el valor en una unidad, la condición del bucle sería siempre verdadera y se seguiria ejecutando indefinidamente.

```
#pragma strict

var valor : int = 8;

function Start () {

      while(valor < 12){
            Debug.Log(valor);
            valor++;
      }
}
```

1.2. do/while

En el caso en el cual queramos que la condición se pueda cumplir al menos una vez, podemos utilizar do/while, en donde en el do va la condición que al menos se ejecuta una vez.

```
#pragma strict

var valor : int = 8;

function Start () {
	do{
		Debug.Log(valor);
		valor++;
	}while(valor < 8);
}
```



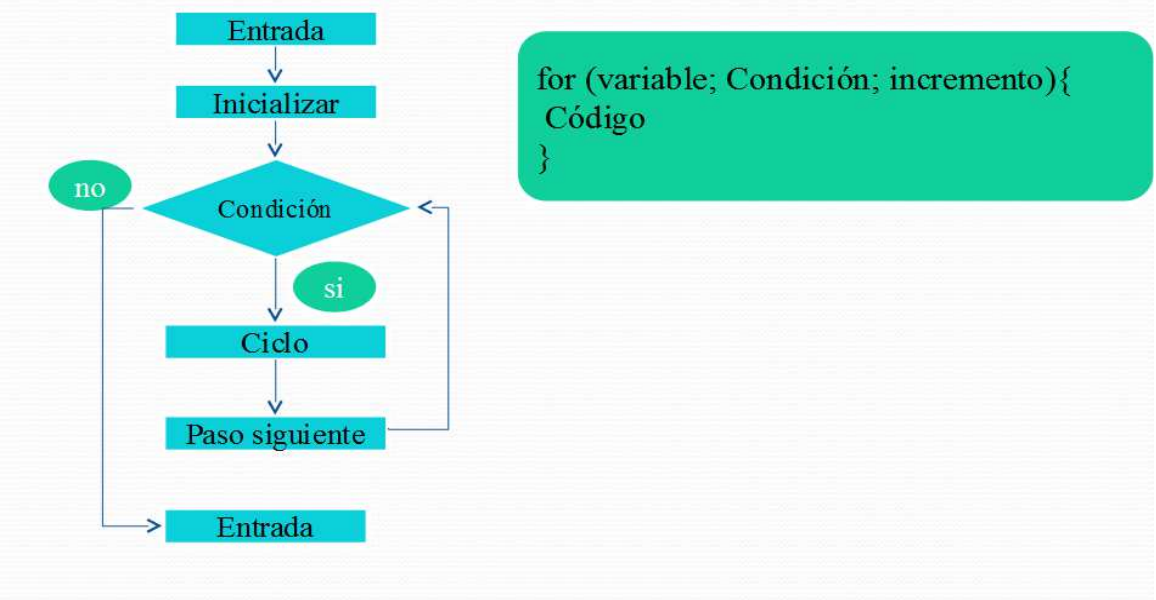
1.3. for

El bucle for, es una estructura muy útil, que posee un condicional con tres términos:

1.- El primer término es la variable que se va tomar en cuenta y por donde se ingresa en un bucle for la primera vez.

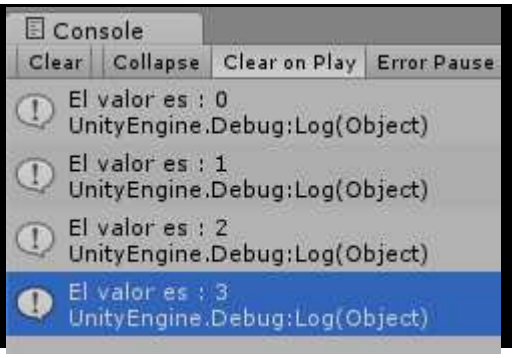
2.- El segundo termino es la condición que la variable debe cumplir para ejecutar el código.

3.- El tercer término es el incremento (decremento u otra forma de variación) que se debe tomar antes de ingresar nuevamente en el condicional.



El siguiente ejemplo imprime los valores del cero al tres en la consola de Unity

```
#pragma strict

var valor : int = 4;
function Start () {
      for(var i: int = 0; i < valor; i++)
      {
            Debug.Log("El valor es : " + i);
      }
}
```



Bloque temático 2: Física y matemática como herramientas de simulación.

Hasta ahora si bien hemos presentado los conceptos de variable, función, operadores, bucles, y estructuras de control, estas son herramientas que son aplicables a un sin fin de situaciones dentro y fuera del mundo de los videojuegos, por ejemplo en una página web se nos muestra las últimas cinco noticias de un diario o las primeras cuatro películas de un sitio destinado a este fin, mediante un bucle del tipo while que recupera la información de una base de datos según el filtro que hemos considerado en la consulta, es más el uso de while para este fin es independiente de si para comunicarnos con el servidor estamos utilizando un lenguaje como C#, Python, JavaScript, Java, Ruby, Fortran90, etc.....

En el mundo de los videojuegos se utilizan estas herramientas con la finalidad de simular movimientos, interacciones, desplazamientos, etc. La simulación se puede aproximar más o menos según la habilidad del programador, y sus conocimientos de la física que nos rodea, en lo que resta de esta unidad trataremos de cubrir algunos conceptos físico matemáticos que son imprescindibles de conocer.

2.1. Cinemática

La cinemática, es la rama de la física que se encarga de estudiar los movimientos de los cuerpos, sin preocuparse de cual es el origen de los mismos. Si bien los movimientos reales son muy complejos, cuando las dimensiones de un objeto son mucho menores que las de su trayectoria, es posible representar al objeto por un punto matemático, tomando al objeto como una partícula. Un ejemplo puede ser el de una persona saltando desde un acantilado, en este caso se puede despreciar los movimientos que realiza con sus extremidades y tomar su cuerpo como si fuera un punto para determinar la trayectoria que realizará.

En una dimensión podemos representar al desplazamiento como:

```
desplazamiento en x = x final – x inicial
```

Si bien no vamos a entrar en conceptos como derivadas e integrales que harían este material muy complejo de comprender, podemos decir que la velocidad (la cual se obtiene como la derivada de la posición) adquiere una expresión de la siguiente forma:

```
velocidad media = (variación en la posición) / (variación en el tiempo)
```

O desarrollando sería:

velocidad media = (posición final – posición inicial) / (tiempo final – tiempo inicial)

De forma similar, se obtiene la aceleración como la derivada de la velocidad, la cual nos quedaría como sigue:

aceleración media = (variación en la velocidad) / (variación en el tiempo)

O desarrollando sería:

aceleración media = (velocidad final – velocidad inicial) / (tiempo final – tiempo inicial)

Tarea 1: Realizar un listado 10 tipos de desplazamientos lineales que se le ocurran.

Tarea 2: Realizar un script que dadas las posiciones finales e iniciales de un cuerpo nos permita calcular su velocidad, considerando que el tiempo transcurrido ha sido de 10 segundo. Representar por consola el resultado asignándoselo a un Game Object vacío.

¿Qué unidades utilizo para la velocidad?

¿Cuáles utilizaría para la aceleración?

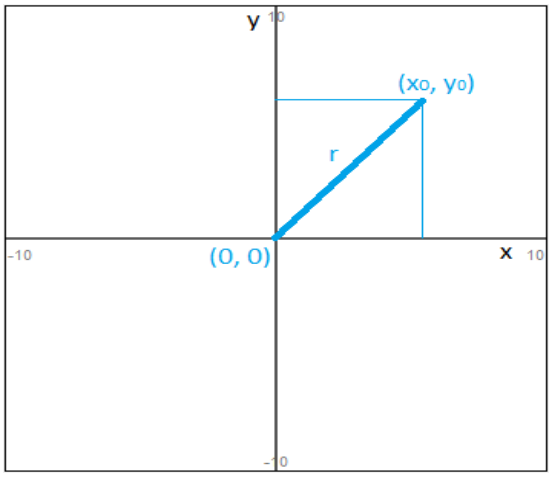
Realice un listado comparativo de las velocidades y aceleraciones promedio de los siguientes vehículos :

Un tren de carga – Un tren de pasajeros – un auto – un avión – un corredor de velocidad.

Bloque temático 3: Vector2 y Vector3

La idea no es que seamos unos expertos en matemáticas y física, si utilizamos una plataforma como Unity que resuelve por nosotros muchos de los inconvenientes de este tipo que podamos encontrar, sin embargo un mayor compromiso por el estudio de la cinemática, la dinámica y las matemáticas nos permitirán sin lugar a dudas obtener los mejores resultados.

Cuando trabajamos en dos dimensiones Unity nos provee del método “Vector2” el cual nos permite ubicar un punto en un espacio de dos dimensiones representados a continuación por los ejes x e y. Un punto en este espacio de dos dimensiones se representa por un par de valores (x_0, y_0), y la distancia del punto al origen (0,0) se puede obtener mediante el teorema de Pitagoras como $r^2 = x^2 + y^2$



En el siguiente ejemplo, estamos declarando una variable “vector” del tipo “**Vector2**” y asignándole el punto (3, 5), el cual tiene como distancia al origen la raíz cuadrada de (3^2 + 5^2). En Unity este valor se obtiene al concatenarle a la variable “**.magnitude**”

```
#pragma strict
function Start () {
    var vector : Vector2 = Vector2(3, 5);
    Debug.Log(vector.magnitude);
}
```

De forma análoga al trabajo con Vector2, cuando vamos a tres dimensiones podemos obtener la posición de un punto mediante “Vector3”

```
#pragma strict
function Start () {
    var vector : Vector3 = Vector3(3, 5, 2);
    Debug.Log(vector.magnitude);
}
```

1. Que representan los ejes de tres dimensiones en Unity

Unity toma la siguiente convención para representar el espacio tridimensional:

1.- El eje “x” representa izquierda derecha.

2.- El eje “z” representan adelante y atrás.

3.- El eje “y” representa arriba y abajo. De esta forma podemos utilizar los siguientes atajos al hacer referencia a un determinado punto.

left = Vector3(-1, 0, 0).

right = Vector3(1, 0, 0).

down = Vector3(0, -1, 0).

up = Vector3(0, 1, 0).

back = Vector3(0, 0, -1).

forward = Vector3(0, 0, 1).

Los atajos de teclado para el origen y el vector de componentes 1 son los siguientes.

zero = Vector3(0, 0, 0).

one = Vector3(1, 1, 1).

2. Acceder a cada componente

En determinadas ocasiones, podemos estar interesados en acceder a un componente dado de un vector, para lo cual podemos utilizar las letras x, y, z para acceder al componente específico de dicho vector.

Por ejemplo, si queremos acceder al componente x lo podemos realizar de la siguiente forma:

```
#pragma strict
function Start () {
    var vector : Vector3 = Vector3(3, 5, 2);
    Debug.Log(vector.x);
}
```

Otra forma de realizar lo mismo, es de la siguiente forma:

```
#pragma strict
function Start () {
    var vector : Vector3 = Vector3(3, 5, 2);
    Debug.Log(vector[0]);
}
```

En este caso:

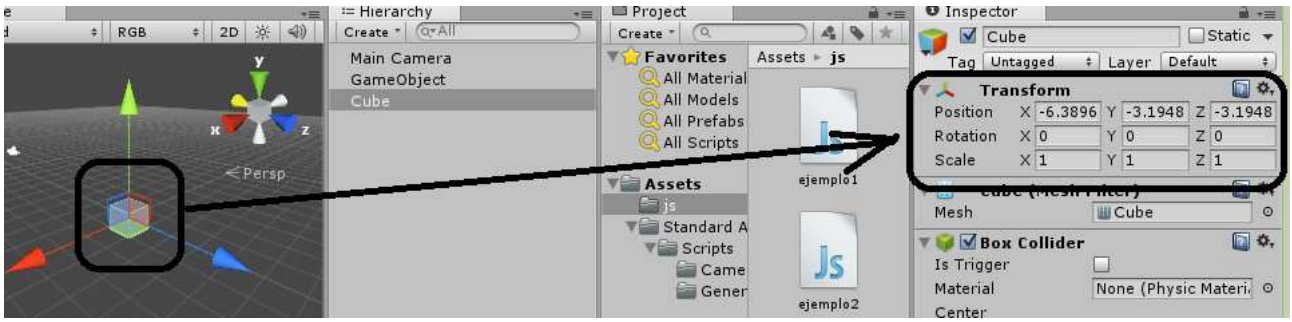
x = [0]

y = [1]

z = [2]

Bloque temático 4: Transform

Cuando creamos un Game Object, podemos observar que el mismo posee asociado en el panel Inspector, una solapa denominada Transform, la cual tiene los datos de posicionamiento, rotación y escala.

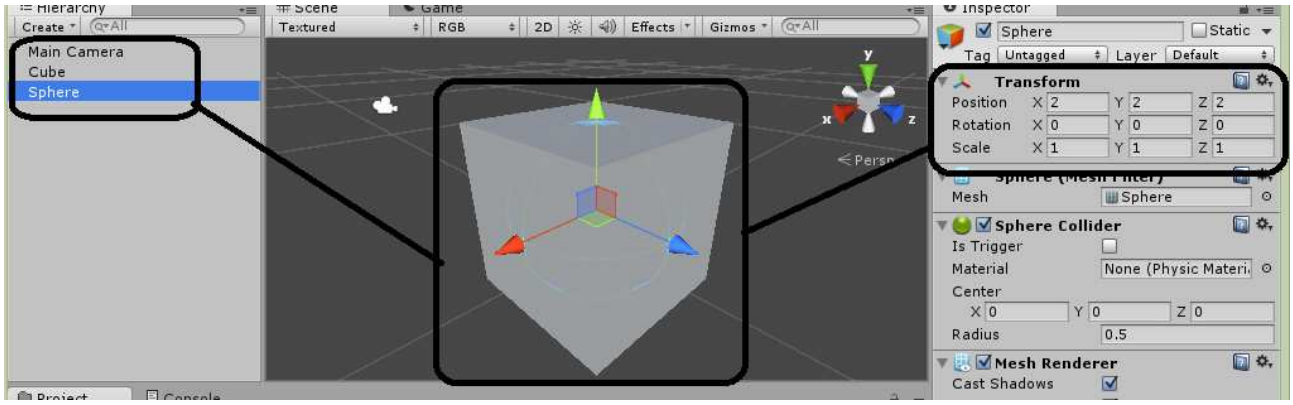


Nota: No puede existir un Game Object sin su solapa Transforma asociada.

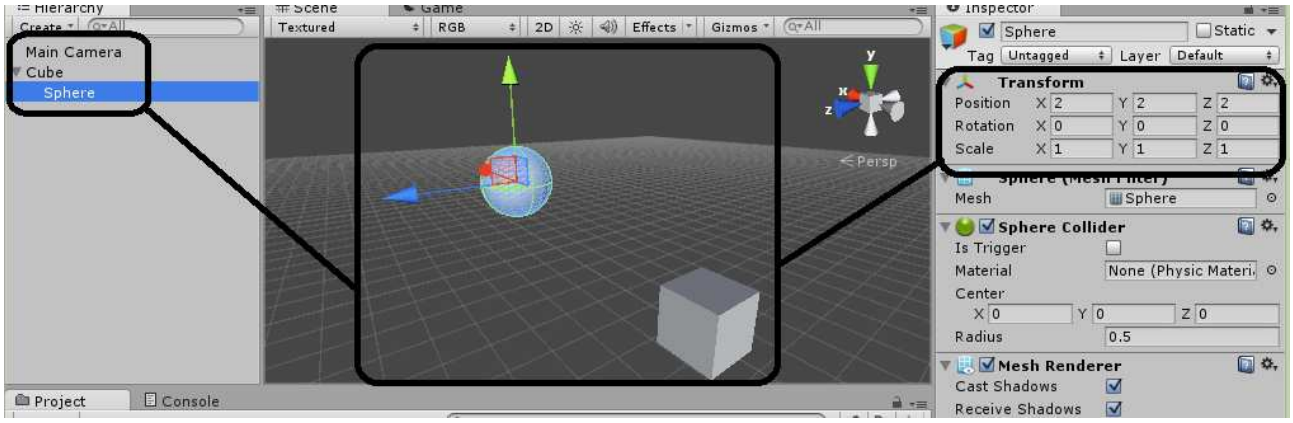
Transform es una clase de Unity que nos permite trabajar con nuestros objetos en un espacio global o local, pero ¿que es un espacio global y un espacio local?

3.1. Espacio global vs espacio local

Un espacio global, es el espacio con relación al cual se ubican todos los objetos inicialmente, cuando creamos un cubo y una esfera dentro de la pantalla de Unity, estos se ubican con relación a un origen de coordenadas (0, 0, 0) global, es decir que si a ambos les doy coordenadas (2, 2, 2) en la pantalla se encuentran superpuestos.



Sin embargo si en el panel de jerarquía arrastro la esfera al cubo, sus coordenada pasaron a ser (0, 0 ,0) pero con relación al cubo. Si ahora lo posiciono en el punto (2, 2, 2) voy a ver como la esfera toma al cubo como origen de coordenadas. En otras palabras, la esfera está ubicada no en relación al sistema de coordenadas global de Unity, sino al sistema de coordenadas local del cubo, y se mueve con relación a este.



3.1. Asignando posición desde script – Uso de: position y localPosition

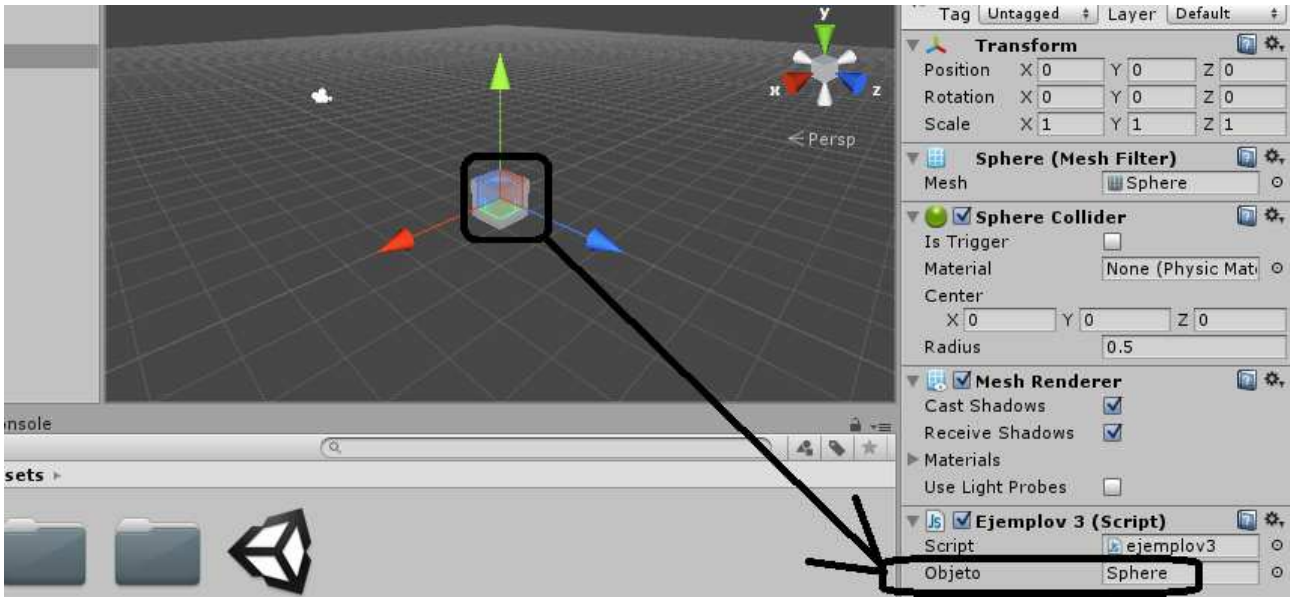
Para modificar la posición de un objeto desde un script, podemos realizarlo de la siguiente forma.

1.- Creemos un Game Object vacío al cual asociar el siguiente script:

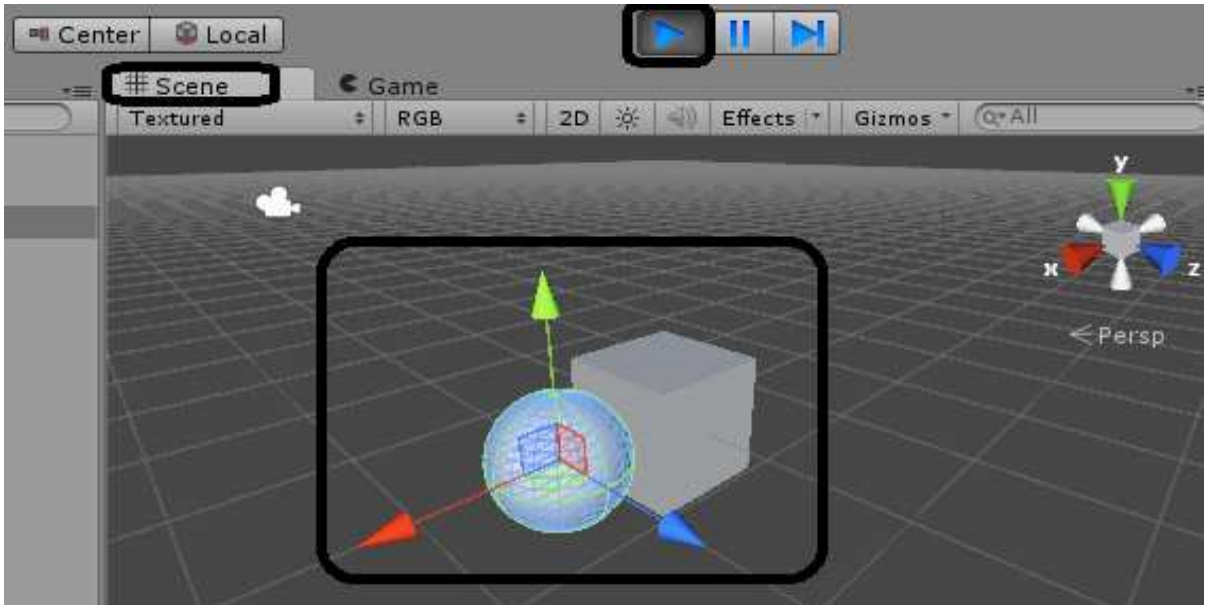
```
#pragma strict
var objeto: GameObject;
function Start () {
     objeto.transform.position = Vector3(1, 0, 0);
}
```

2.- Creemos una esfera y un cubo y posicionemos ambos elementos en origen de coordenadas global.

3.- Asociemos la esfera a la variable objeto del script, simplemente arrastrando la esfera del panel de jerarquía a casillero objeto del inspector cuando previamente se selecciono el Game Object vacío.



Ahora al entrar en modo juego podemos ver como la esfera se ha desplazado una unidad en el eje de las x positivas.

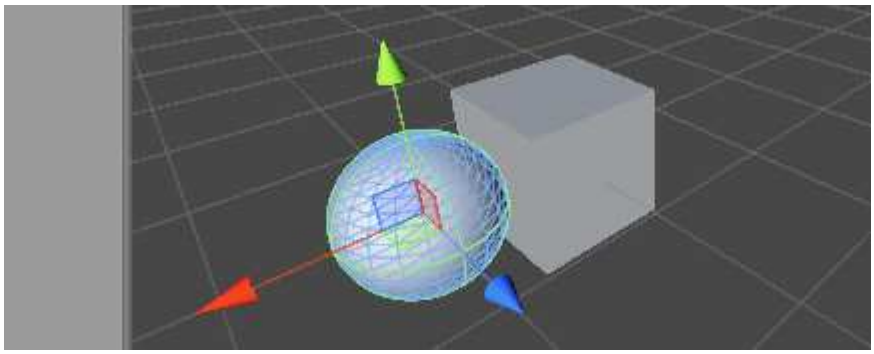


Esta posición final se sigue cumpliendo con relación al eje global, ya que estamos utilizando el atributo position. Podríamos ver como esto es así desplazando el cubo a la posición (1, 0, 0) y arrastrando a la esfera al cubo. Al entrar en modo juego la esfera sigue estado en el la posición (1, 0, 0).

Para hacer que la esfera se mueva con relación al cubo, es decir en coordenadas locales al cubo, podemos modificar el script cambiando “position” por “localPosition”:

```
#pragma strict
var objeto: GameObject;
function Start () {
     objeto.transform.localPosition = Vector3(1, 0, 0);
}
```

En este caso el movimiento es con relación al elemento padre (cubo) por lo que la esfera ahora se encuentra en la posición global (2, 0, 0) y en la posición local (1, 0, 0).

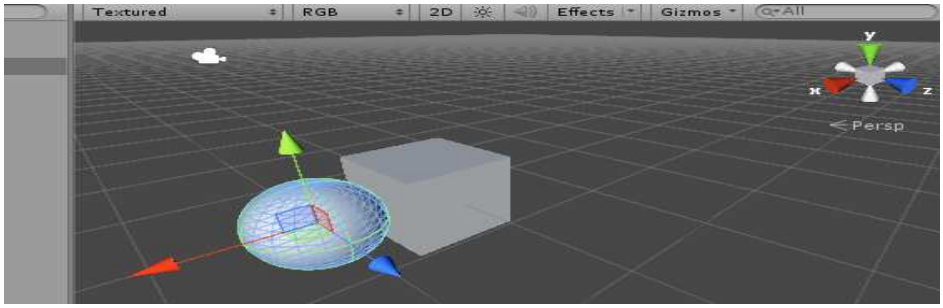


3.2. Asignando rotación desde script – Uso de: eulerAngles y localEulerAngles

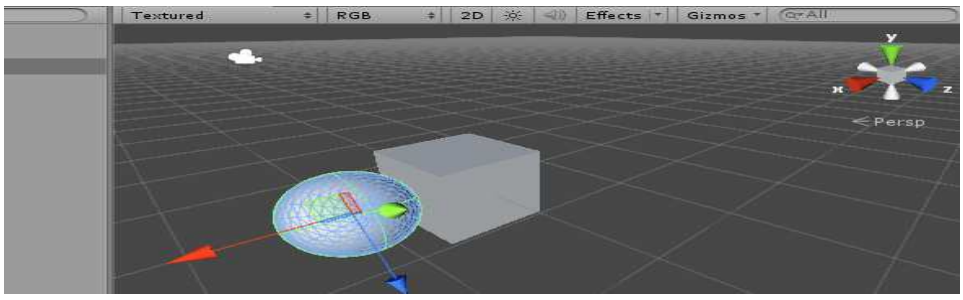
Para agregar rotación se procede en forma análoga que con la traslación, salvo que en este caso se modifica el eje del elemento con relación a los ejes globales “eulerAngles” o con relación al sistema de ejes del elemento padre “localEulerAngles”.

Los valores representan en lugar de unidades de desplazamiento, unidades de ángulo expresados en grados.

En el ejemplo siguiente la esfera se encuentra solo trasladada en una unidad según el eje x



Mientras que en la imagen que se presenta a continuación no solo tiene una traslación de 1 unidad según el eje x con relación a su elemento padre “el cubo”, sino que se le ha agregado una rotación de ejes de 60° según dicho eje relativos al cubo.



Tarea: Realizar un script que permita asignarle a un objeto, una traslación de una unidad según el eje x, y una rotación de 60° según el mismo eje.

Nota: Podríamos utilizar un atajo como objeto.transform.eulerAngles = transform.right*60; Para realizar una rotación de 60° según el eje x, siguiendo la regla de la mano derecha, en donde el pulgar apunta al eje x positivo (en este caso ya que usamos la palagra right), y el resto de los dedos indican la dirección de giro.

Bloque temático 5: Asignación de nombre en modo juego

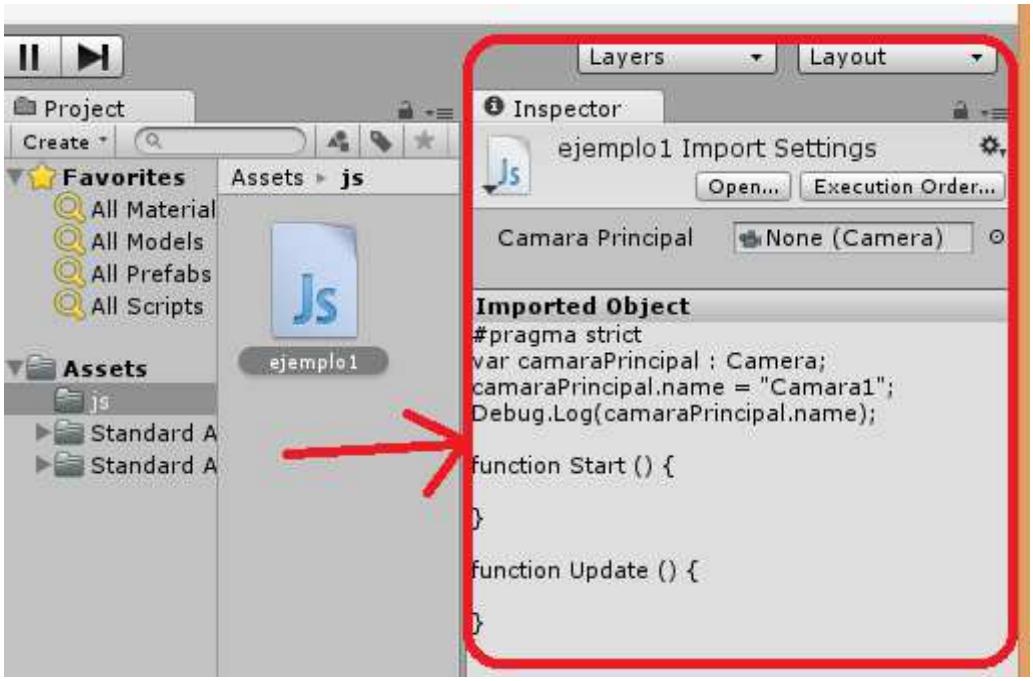
Veamos ahora como realizar asignación de nombres durante el juego, como primer ejemplo, asociemos una cámara al cubo, para lo que vamos a generar el siguiente script:

ejemplo1.js

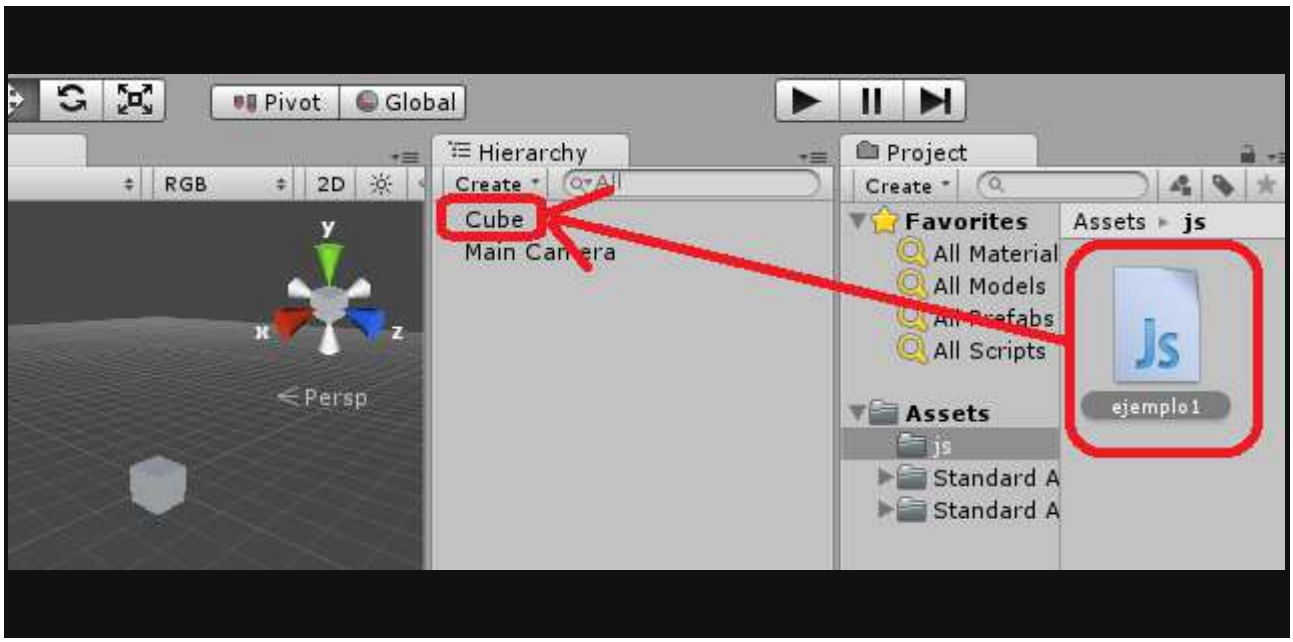
```
#pragma strict
var camaraPrincipal : Camera;
camaraPrincipal.name = "Camara1";
function Start () {
     Debug.Log(camaraPrincipal.name);
}
```

En la primera línea creamos una variable que va a guardar un objeto de tipo “Camera”, luego en la segunda línea le asignamos un nombre “Camara1”. Posteriormente mediante Debug.Log lo imprimimos en pantalla.

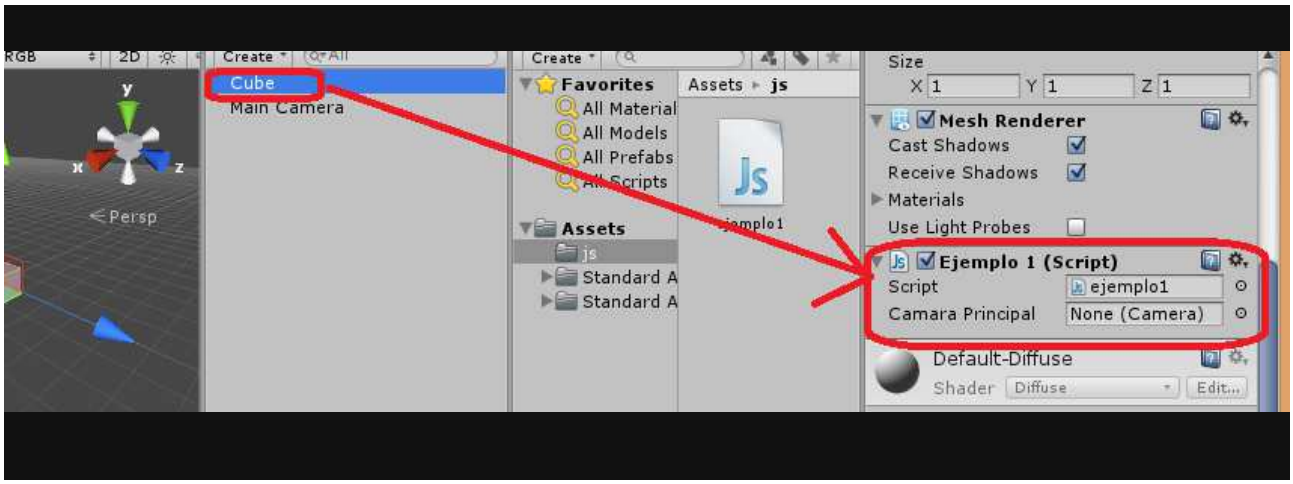
Ahora vamos a la pantalla de Unity y corroboramos que el script se ha actualizado en el Inspector:



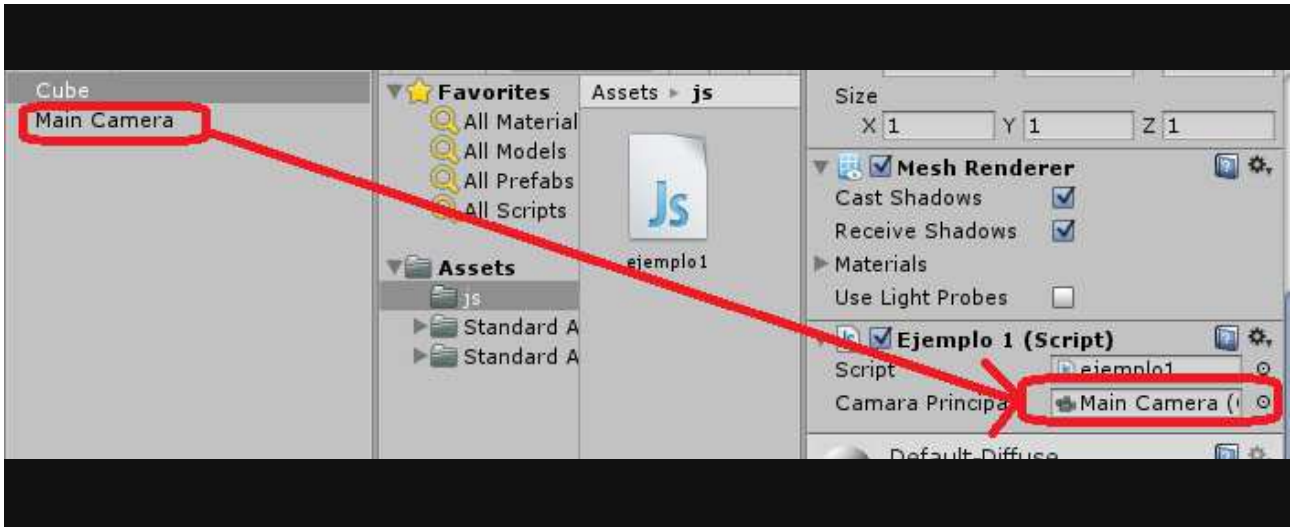
Luego para asignar el script al objeto, arrastramos el script desde el directorio js a el objeto (en este caso creemos un cubo) en el panel de Jerarquía.



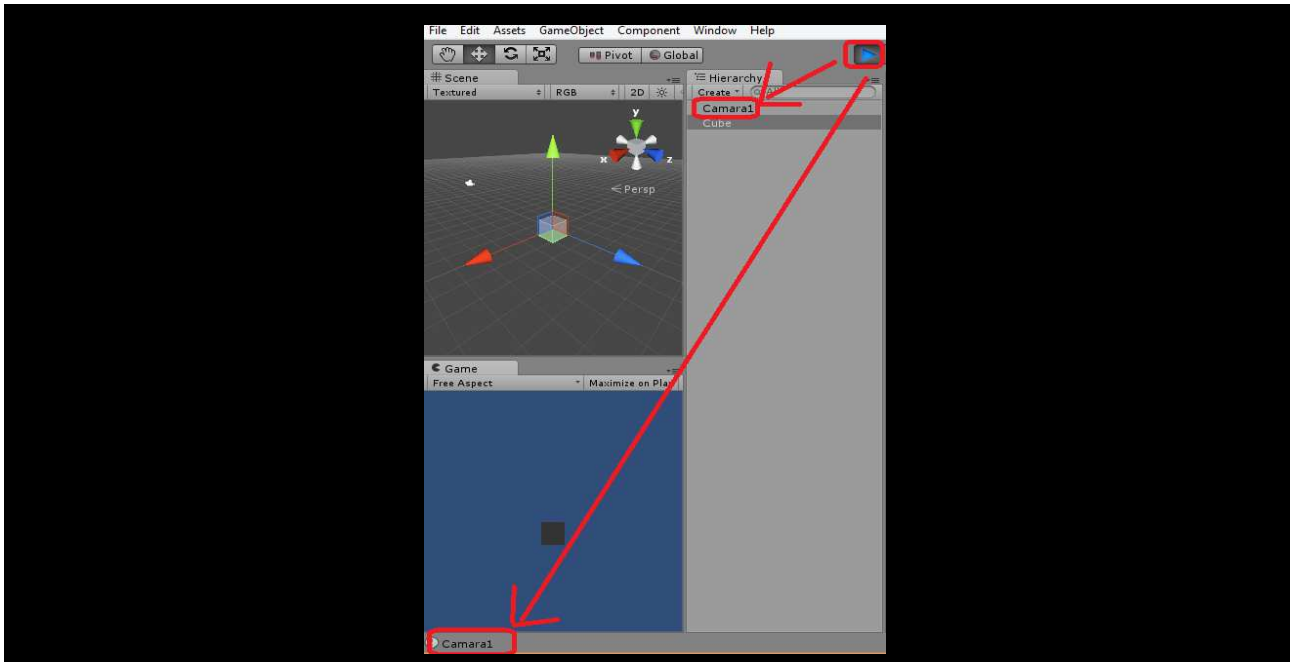
Al seleccionar el cubo en el panel de Jerarquía, ahora podemos ver que en el panel de Inspector figura el script asociado, con visualización de la variable asignada anteriormente.



Ya que hemos creado una variable de tipo “Camera” podemos arrastrar la camara existente en el panel de jerarquía sobre la variable del Inspector para asignarla.



Si ahora entramos en modo juego podemos ver como el nombre de la cámara es asignado en la sección Game a la cámara, y en el panel de jerarquía el nombre se ha modificado al dado en el script.



Bloque temático 6: Asignar nombre a objeto que contiene el script.

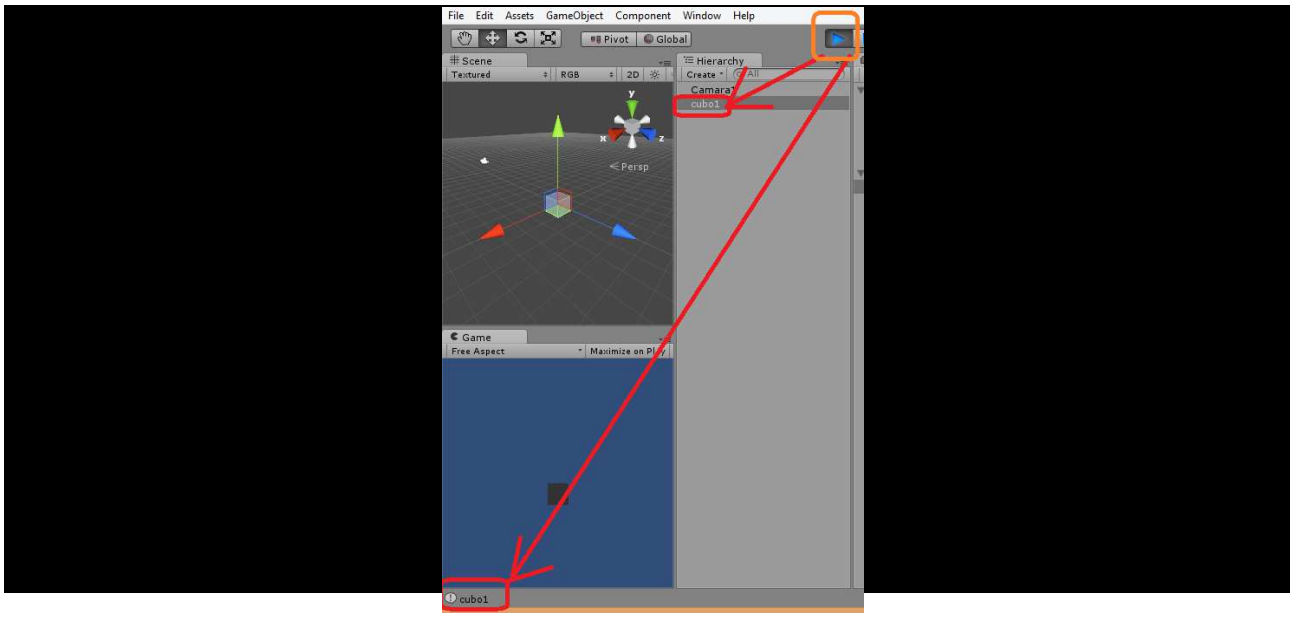
A todos los objetos se les puede asignar un nombre durante el juego, para ello lo único que tenemos que hacer es agregar una variable llamada “name” en el script asociado a un objeto. Veamos esto en funcionamiento, para ello abramos el script del Bloque temático 4 y agreguemos un nombre al cubo.

Nuestro script modificado queda de la siguiente forma:

ejemplo1.js

```
#pragma strict
name = "cubo1";
var camaraPrincipal : Camera;
camaraPrincipal.name = "Camara1";
function Start () {
      Debug.Log(camaraPrincipal.name);
      Debug.Log(name);
}
```

Como podemos ver en la siguiente figura, el nombre del cubo aparece en el panel de jerarquía y en la Escena.

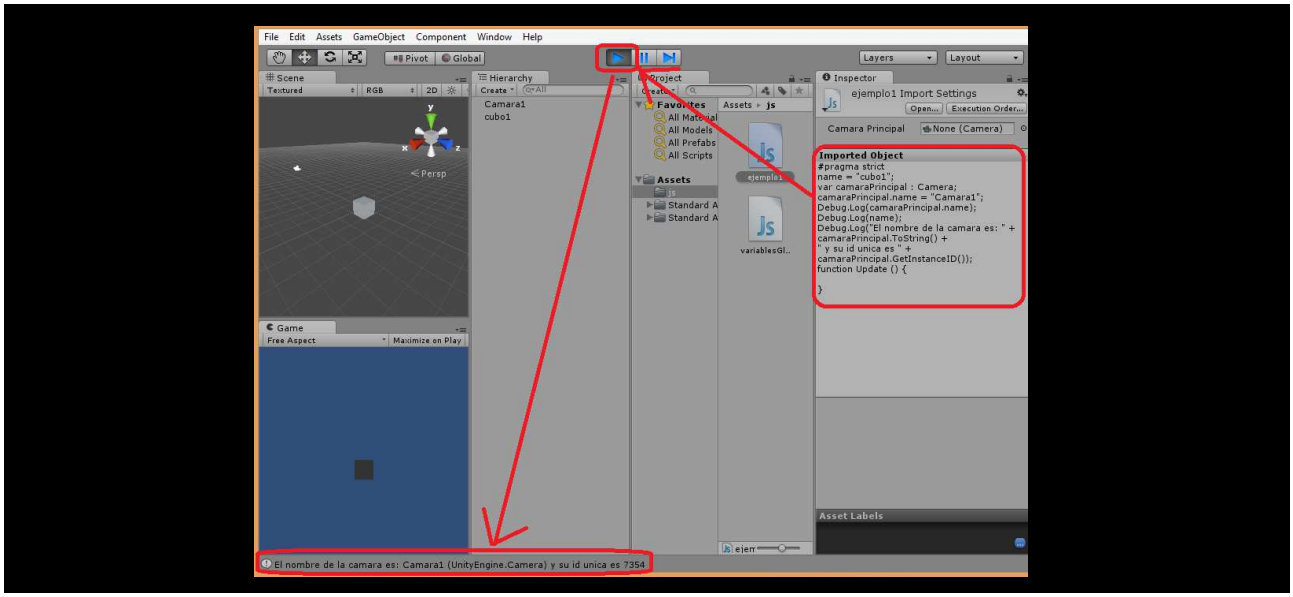


Bloque temático 7: Obtener nombre e identificador de un objeto

Existen dos funciones asignables a cada Game Object, la primera es “ToString” la cual nos retorna el nombre asignado a un Game Object, y la segunda es GetInstanceID la cual nos retorna el identificador único que utiliza Unity para identificar al Game Object. Podemos ver estas funciones en acción, si modificamos nuestro script “ejemplo1.js” de la siguiente forma:

ejemplo1.js

```
#pragma strict

name = "cubo1";
var camaraPrincipal : Camera;
camaraPrincipal.name = "Camara1";

function Start () {
     Debug.Log(camaraPrincipal.name);
     Debug.Log(name);
     Debug.Log("El nombre de la camara es: " +
camaraPrincipal.ToString() +
" y su id unica es " + camaraPrincipal.GetInstanceID());
}
```



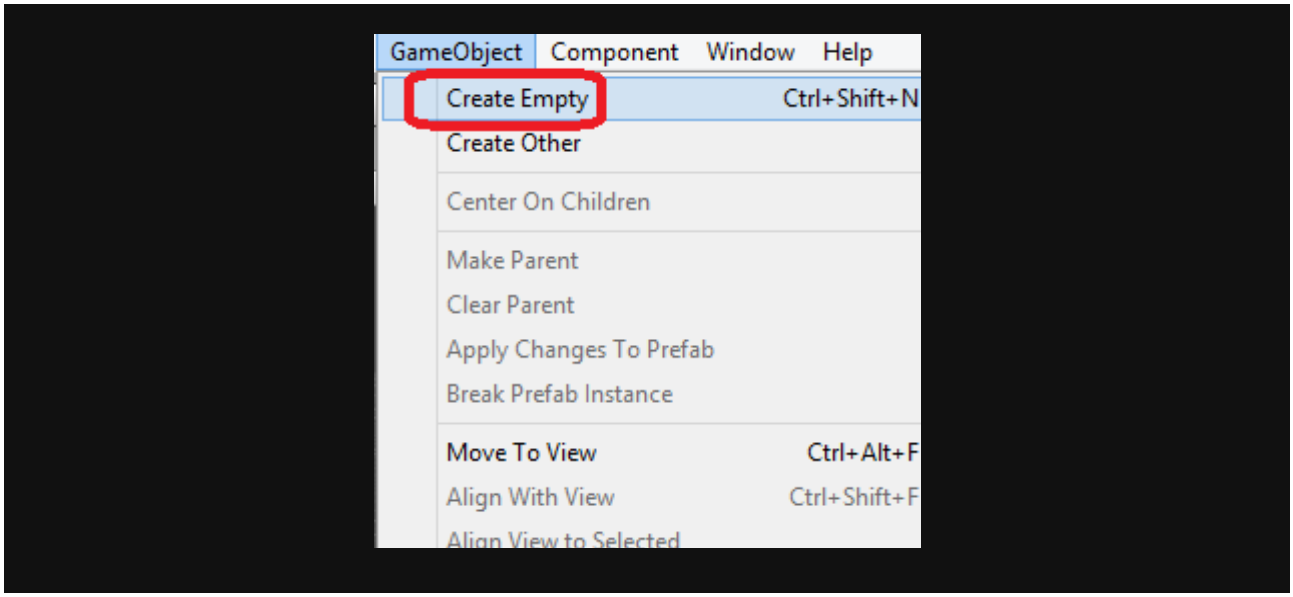
Tarea: Modificar el script ejercicio1.js de forma de que el objeto pueda ser asignado desde el panel de Inspector de Unity mediante arrastrar el Game Object del cual queramos obtener los datos.

Bloque temático 8: Clonación de objetos

Para clonar un objeto (muy utilizado por ejemplo en el uso de proyectiles) podemos realizarlo de la siguiente forma:

Paso 1: Creamos game Objetct vacío.

Vamos a GameObject > Create Empty,



Nos aparece en el panel de jerarquía un Game Object que renombramos como “CubosDuplicados”

Paso 2: Creamos y asignamos script

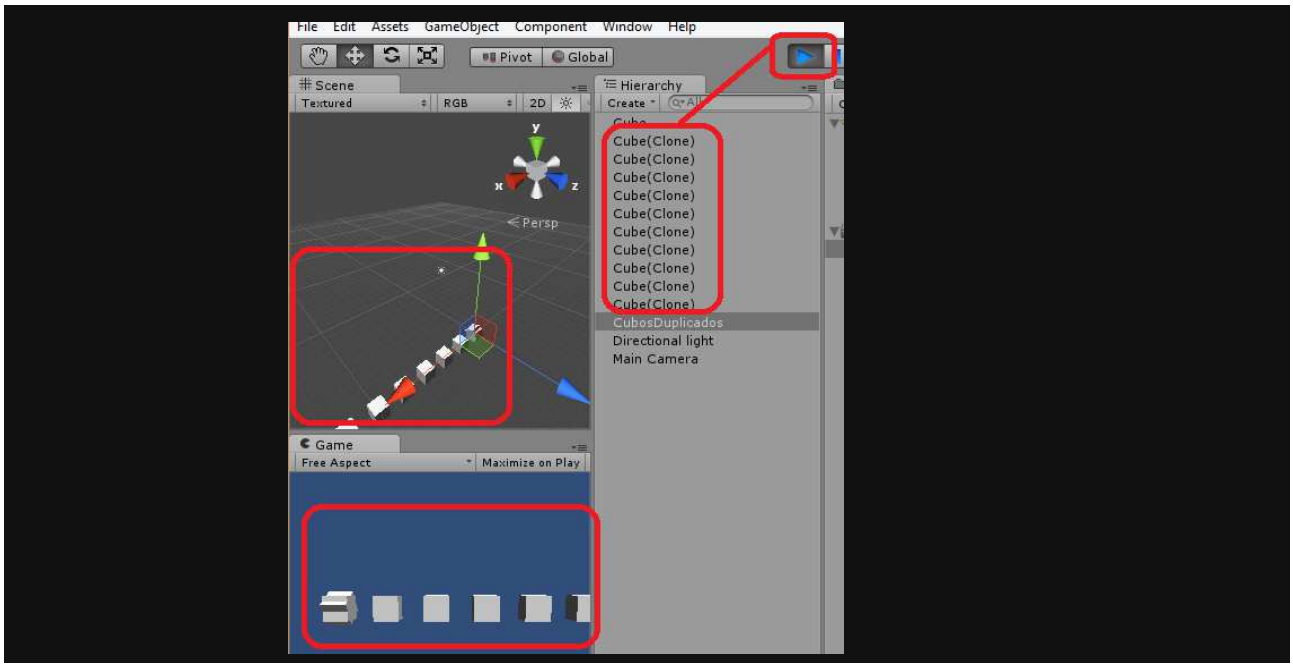
Creamos un script “DuplicarObjeto.js” con el siguiente código:

```
#pragma strict

var prefab : Transform;

function Start () {
     for (var i : int = 0;i < 10; i++) {
          Instantiate (prefab, Vector3(i * 2.0, 0, 0),
Quaternion.identity);
     }
}
```

Aquí Estamos utilizando el loop for para crear mediante el método Instantiate de la clase Object, 10 cubos ubicados a lo largo del eje x. El primer término de Instantiate representa al objeto que queremos duplicar, el segundo término está asociado a la traslación, y el tercer parámetro a la rotación.

Paso 2: Indicamos Game Object a ser copiado

Vamos a Unity y arrastramos el objeto a ser copiado hasta la variable global prefab, y una vez que ingresamos en modo juego, podemos ver las copias del objeto original en pantalla.



Referencia: *http://docs.unity3d.com/ScriptReference/Object.Instantiate.html*

Bloque temático 9: Eliminar Objeto – Object.Destroy

Para eliminar un objeto simplemente le agregamos una función Destroy. Esta función o método de la clase Object, permite especificar el tiempo luego del cual el objeto es destruido, por lo que si ponemos:

```
Destroy (gameObject, 5);
```

El gameObject que estemos pasando como parámetro será destruido luego de 5 segundos (el tiempo se toma como un valor flotante)

Ejercicio:

Crear un scrip con el código anterior y asociarselo al cubo creado, verificar que el mismo desaparece luego de 5seg.

Referencia: ***http://docs.unity3d.com/ScriptReference/Object.Destroy.html***

Bloque temático 10: Encontrar Objeto de tipo – Object.FindObjectsOfType.

Podemos encontrar un objeto de determinado tipo mediante el método de la clase Objeto:

```
FindObjectOfType
```

Como podemos ver en el siguiente script, estamos consultando por elementos del tipo Camera, por lo que lo que tendría que encontrar sería nuestra Main Camera.

```
function Start() {
var buscar : Camera = FindObjectOfType(Camera);
Debug.Log("Encontre:  " + buscar);
}
```

Tarea: Crear un script con el código anterior, asignárselo a un Game Object y verificar su funcionamiento.

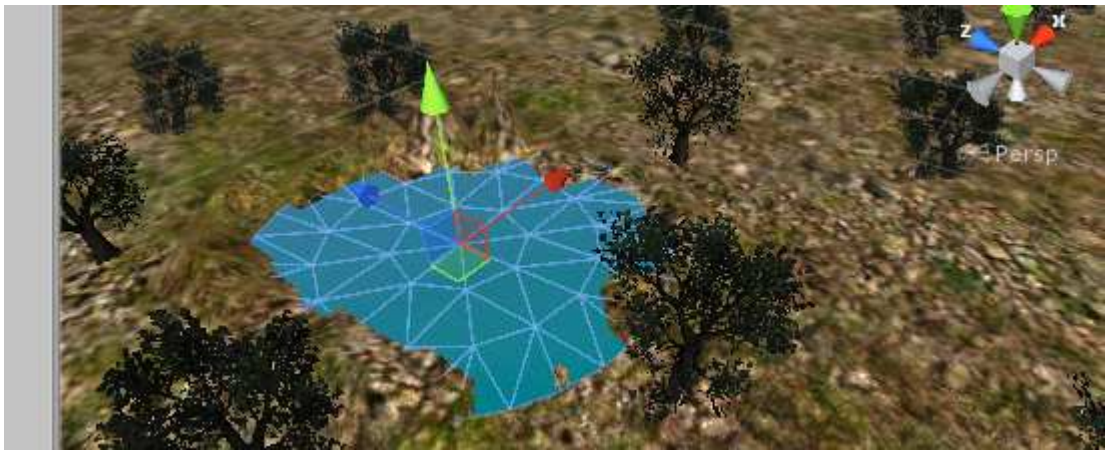
Tarea 2: Investigar sobre el uso de FindObjectsOfType

Bloque temático 11: Crear lago.

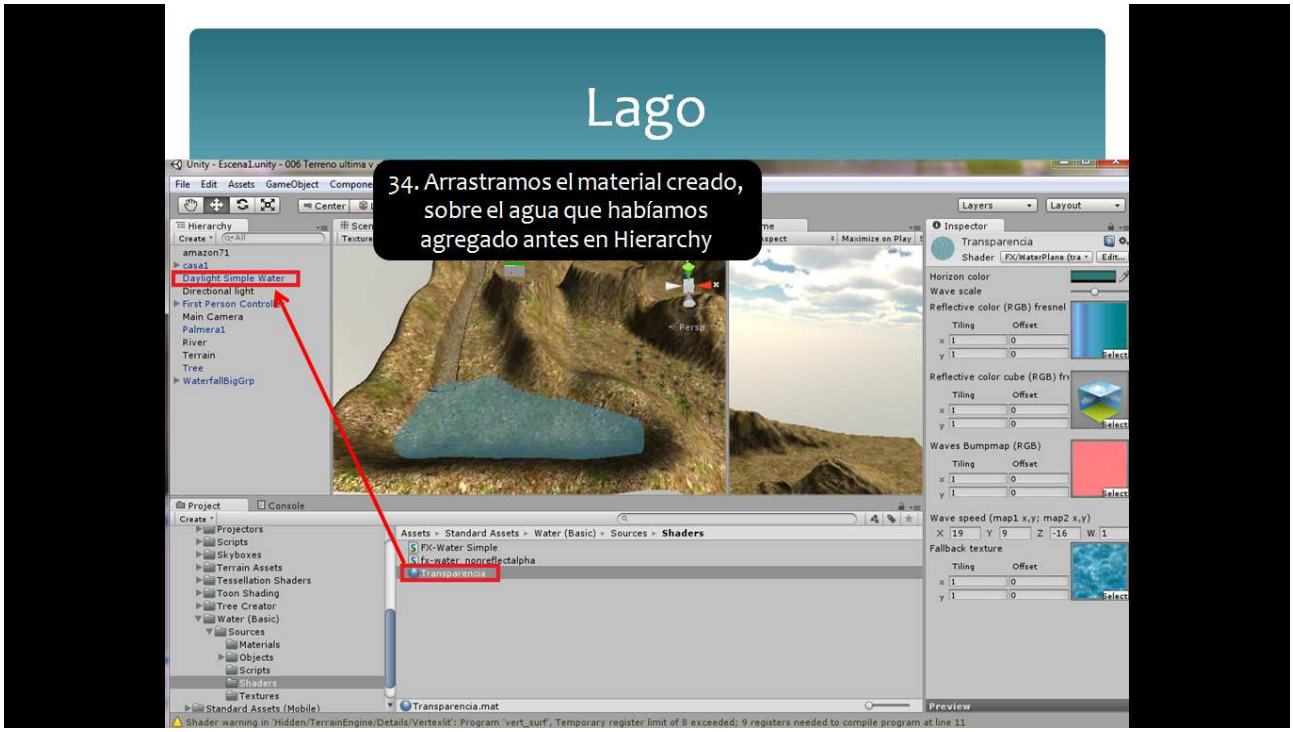
Dejemos ahora un poco de lado la creación de script y veamos como crear un lago. Para comenzar vamos a:

Project > Assets >Standard Assets > Water (Basic)

Ahí tenemos la opción de crear un lago diurno (Daylight Simple Water) o nocturno (Nighttime Simple Water). Para añadirlo, lo podemos arrastrar a la ventana 3D y ubicarlo.



Los efectos de refracción y reflexión de la luz que le dan sensación de agua real, solo están disponibles en Unity Pro, lográndose acabados como el siguiente:


Lago
34. Arrastramos el material creado, sobre el agua que habíamos agregado antes en Hierarchy

Bibliografía utilizada y sugerida

Documentación oficial online -

http://docs.unity3d.com/ScriptReference/Vector2.html

http://docs.unity3d.com/ScriptReference/Vector3.html

http://docs.unity3d.com/ScriptReference/Transform.html

Lo que vimos

En esta unidad hemos visto como trabajar con Loops dentro de javaScript y el uso de herramientas la ubicación espacial de los GameObject dentro de la plataforma de Unity.

Lo que viene:

En la siguiente unidad, vamos a trabajar sobre los personajes realizados, para agregarles animaciones y transiciones de movimientos.