UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning
Secretaría de Extensión y Cultura Universitaria

II.9. 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 II UnityScript I >>

Primeros pasos

Presentación:

Unity nos brinda la posibilidad de programar en C#, JavasScript o Boo, presentando los componentes, como variables y funciones de forma clara y con ejemplos que podemos ir incorporando poco a poco. Programar en Unity requiere no solo conocer el lenguaje en el cual estamos desarrollando, sino también comprender la estructura interna de la plataforma, ya que posee una programación orientada a objetos compuesta por una estructura de clases que proveen muchas funcionalidades ajenas al lenguaje en si.

El javaScript que vamos a utilizar, no es exactamente el lenguaje que podríamos utilizar en una página web, ya que posee algunas diferencias considerables, por ejemplo en la declaración de una variables, en lugar de simplemente declararla como:

var nombreDeVariable;

Lo hacemos como:

var nombreDeVariable : Tipo = Valor;

Esto puede confundir un poco al inicio a quien viene del mundo web, por lo que vamos a ir introduciendo los conceptos de a poco.

Objetivos:

Que los participantes logren:

Comprender los conceptos de variable, array y función.

Aprendan a trabajar con estructuras de control y bucles

Sepan crear scripts y asociarlos a GameObjets

Bloques temáticos:

1.- Crear script y asignarlo a un objeto

2.- Texto enriquecido

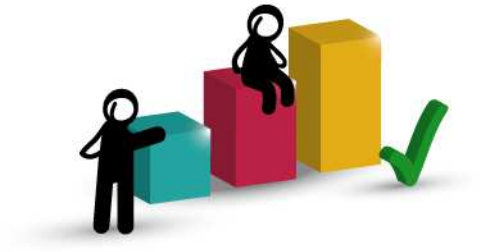
3.- Variables y funciones

4.- Tipo de declaración de variables

5.- Quitar script a objeto

6.- Operadores aritméticos, de asignación, de comparación y lógicos

7.- Estructuras de control



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:

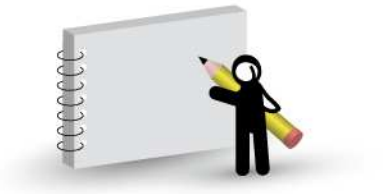
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

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.

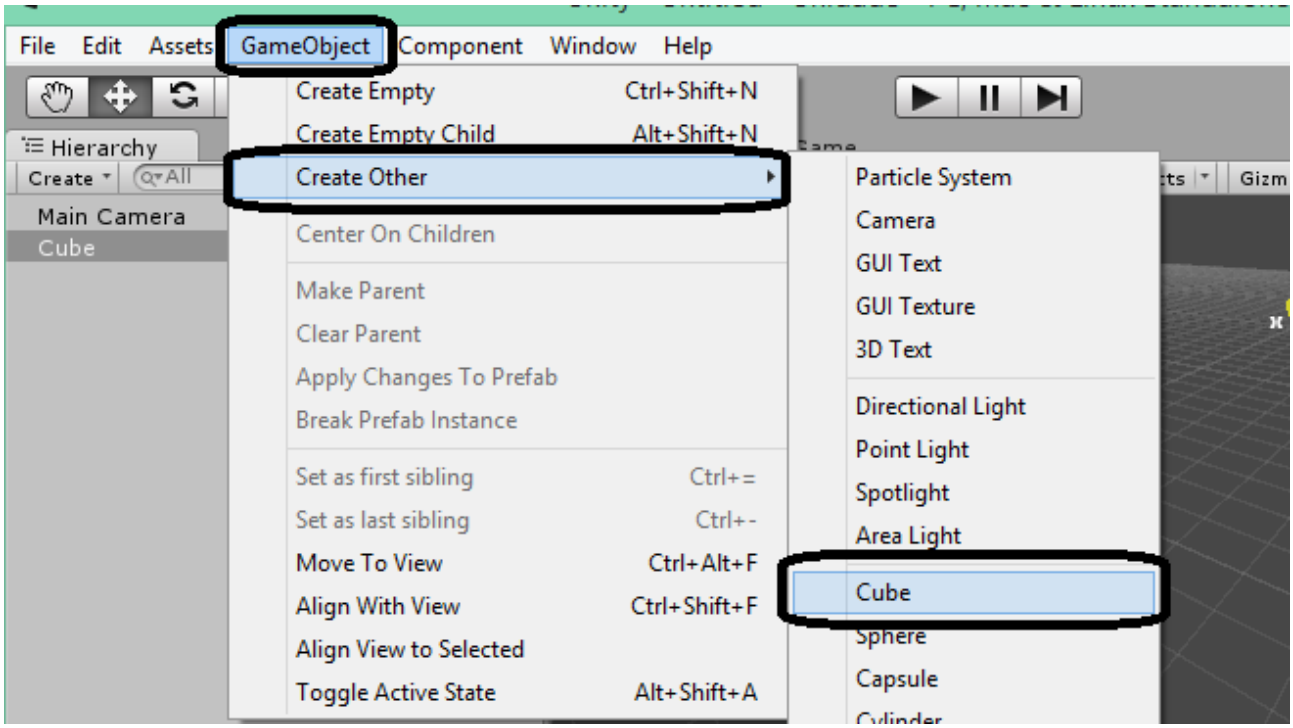
**** Está página queda como está. El contenidista no le quita ni le agrega nada.***

Bloque temático 1: Crear script y asignarlo a un objeto

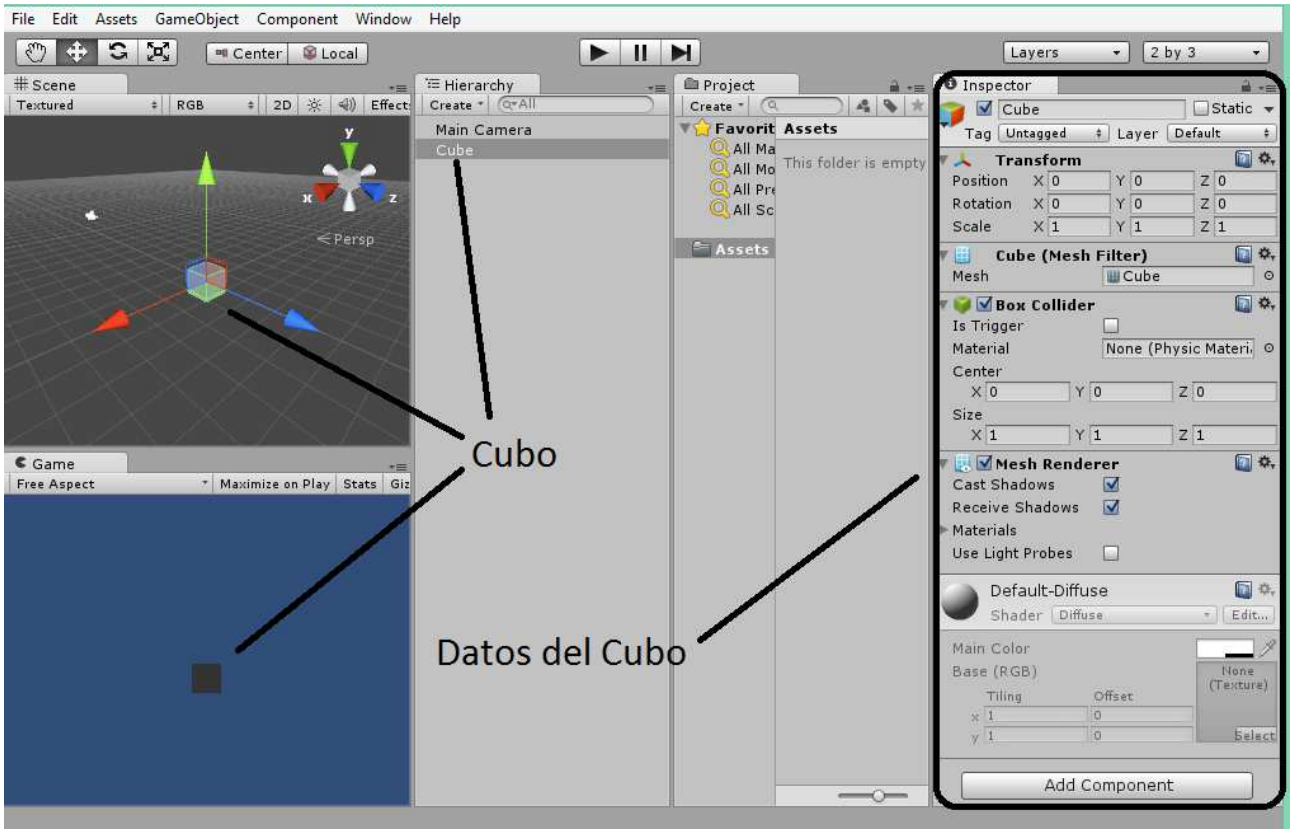
En Unity podemos desarrollar interacciones mediante la generación scripts en C#, Boo y UnityScript. Para crear un script vamos a seguir una serie de pasos.

Paso1: Crear Game Objet

Los script deben ser agregados a objetos dentro del juego, por lo que como primer paso vamos a crear un objeto (Game Object). Contamos con diferentes tipos de objetos, como cubos, lamparas, luces, etc. A modo de ejemplo seleccionemos un cubo aún cuando podríamos seleccionar un Game Object vacío.



Como podemos ver en la siguiente imagen, se ha agregado un cubo al panel de jerarquía (Hierarchy) que puede ser visualizado en la escena (Scene) y en el Juego (Game), y cullos datos se encuentran en desplegados en el inspector de objetos (Inspector). Los datos pueden ir desde la posición del objeto, hasta las variables incorporadas a un script asociado al objeto.

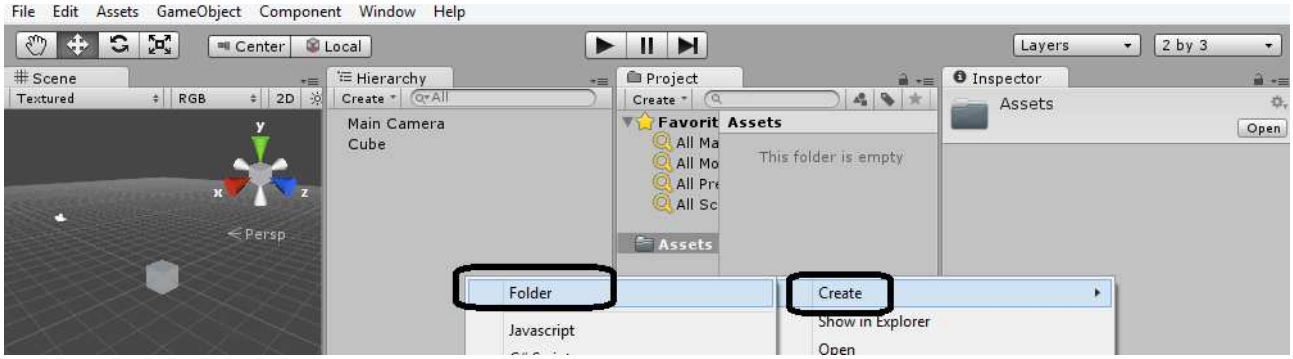


Nota 1: Podemos activar o desactivar de una escena un determinado objeto, destildando el checkbox que aparece en la parte superior del Inspector.

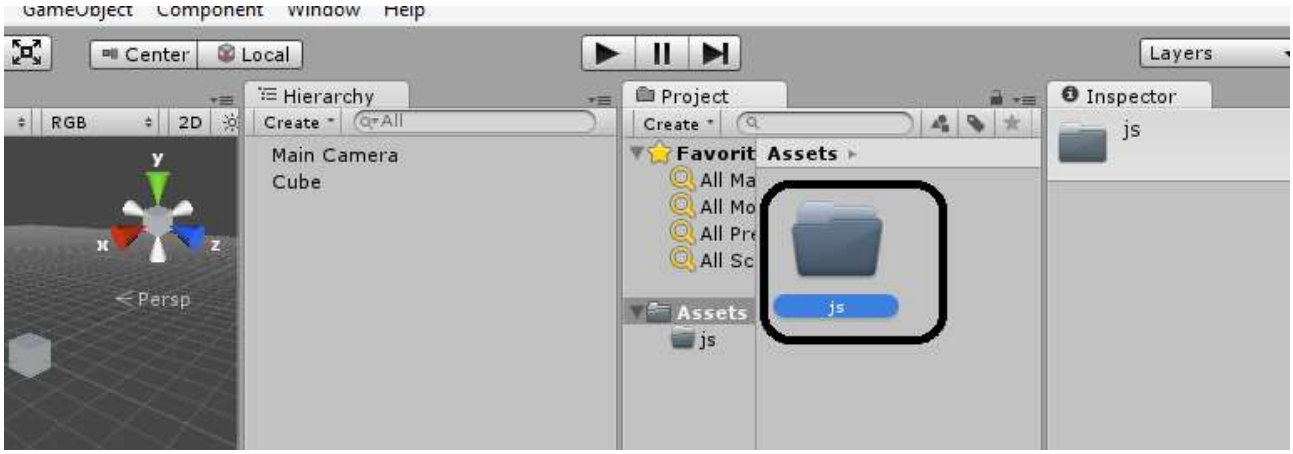
Nota 2: Al modificar el nombre en el Inspector, también se modifica su nombre en el panel de jerarquía.

Paso2: Crear carpeta para script

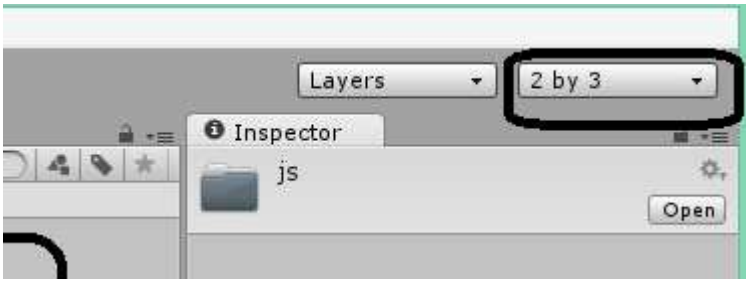
Como segundo paso vamos a crear una carpeta para contener los script, a la cual podemos llamar “js” como se suele llamar en los desarrollos web. Lo único que debemos hacer para generarla es hacer click derecho en el panel Assets y seleccionar “new Folder”, el nombre puede ser modificado mediante F2.



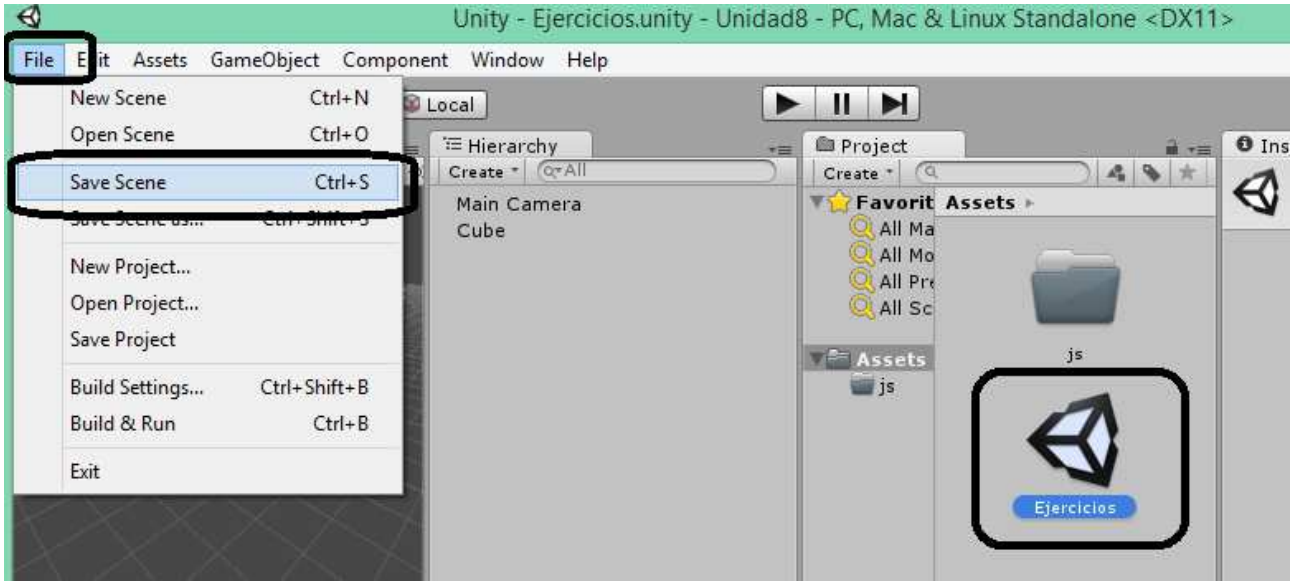
Nos debería quedar como en la siguiente figura.



Nota 1: Estamos utilizando un tipo de visualización 2 by 3, la cual puede ser seleccionada desde el margen superior derecho de la pantalla.



Antes de seguir avanzando sería conveniente salvar la escena, con lo cual vamos a **File > Save Scene** en el margen superior derecho y le damos un nombre. En mi caso "Ejercicios" y lo guardamos en la raíz de Assets. Ahora en el panel de Assets tenemos nuestra scena junto al directorio **js.**

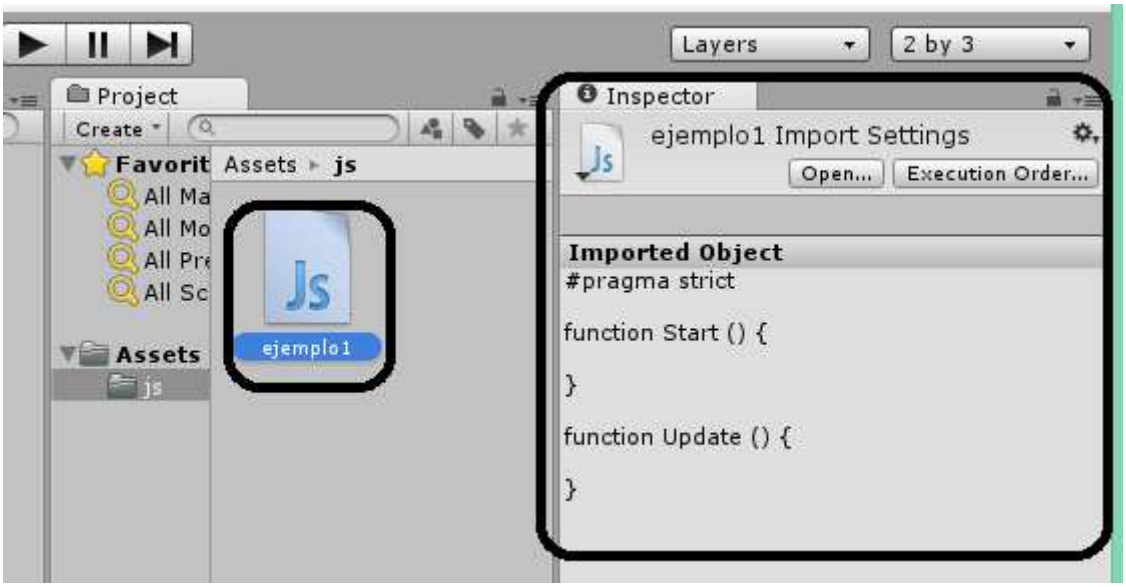


Nota: Las escenas juegan el papel de las diferentes pantallas, en una unidad posterior trataremos este tema, por ahora simplemente crearemos una para trabajar con scripts.

Paso3: Crear script

Para crear un script es similar a lo realizado para crear el directorio “js”. En este caso entramos al directorio js y hacemos click derecho dentro del directorio, luego seleccionamos: Create > Javascript.

Le damos el nombre: **ejemplo1**

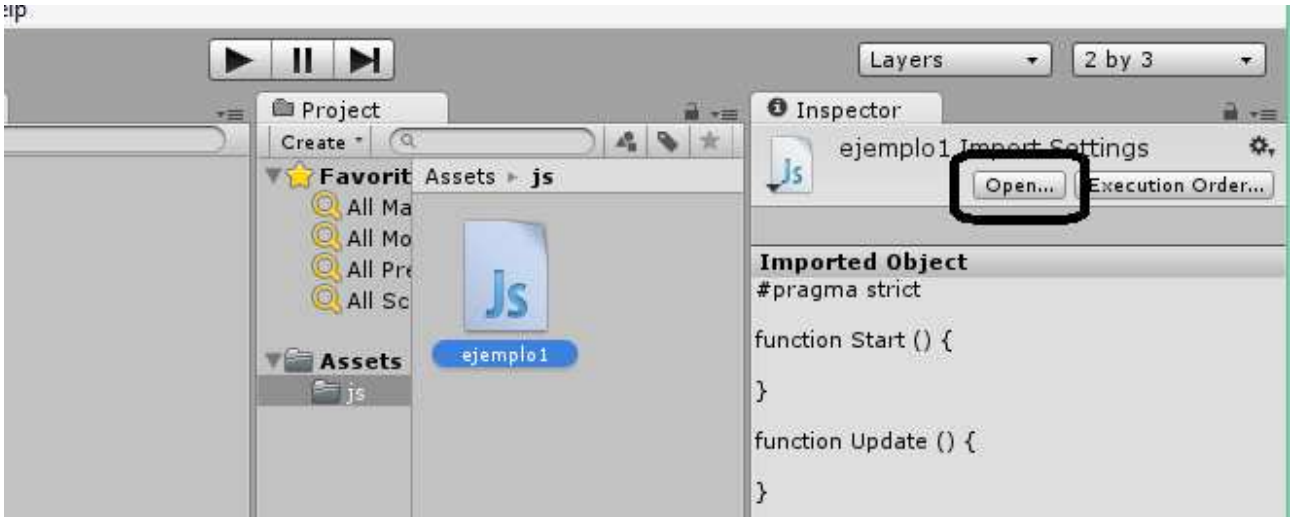


Nota 1: Podemos ver como en el Inspector aparece el código del script y aparecen declaradas dos funciones: Start() y Update().

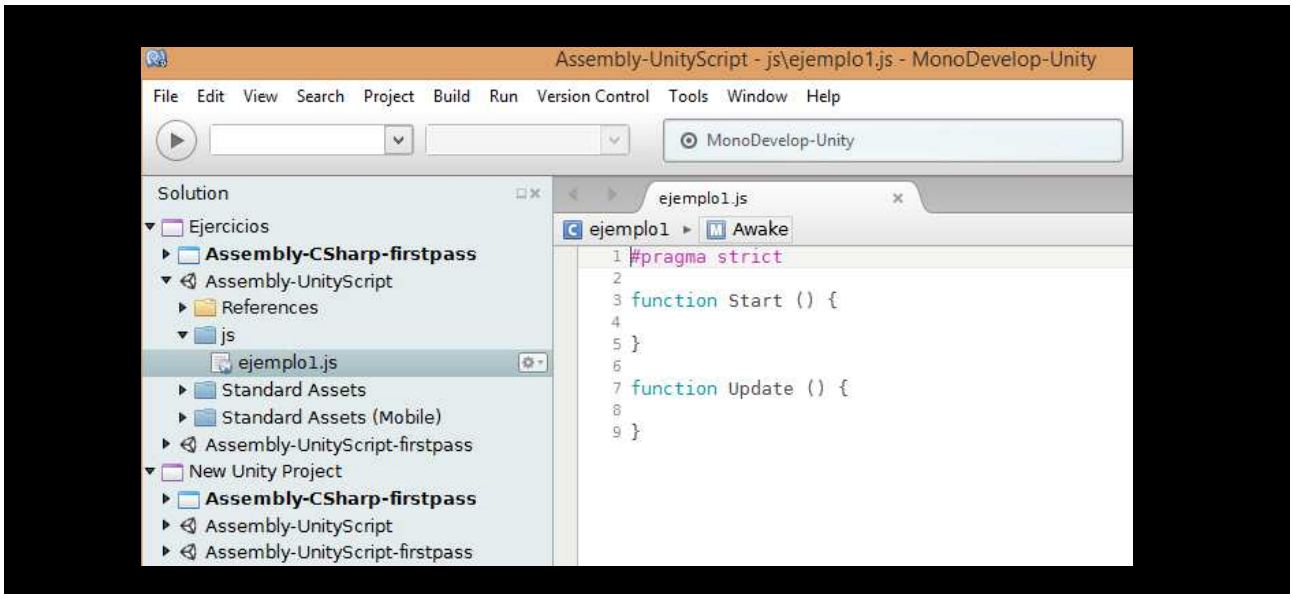
Nota 2: Comenzaremos a trabajar asumiendo que quien lee este material no sabe absolutamente nada de programación, por lo que el concepto de función será introducido un poco más adelante.

Paso4: Abrir script

Para abrir el script podemos utilizar cualquier editor de texto, sin embargo Unity trae un editor que nos permite realizar un Debbug (buscar errores en el código) y nos sugiere soluciones por lo que es la mejor opción para desarrollar en esta plataforma. Para abrir el editor de Unity presionamos en Open, con lo que el script que podemos visualizar en el Inspector se abre en una ventana nueva.



El editor que se habré se llama MonoDevelop y presenta la siguiente apariencia:



Paso5: Crear código

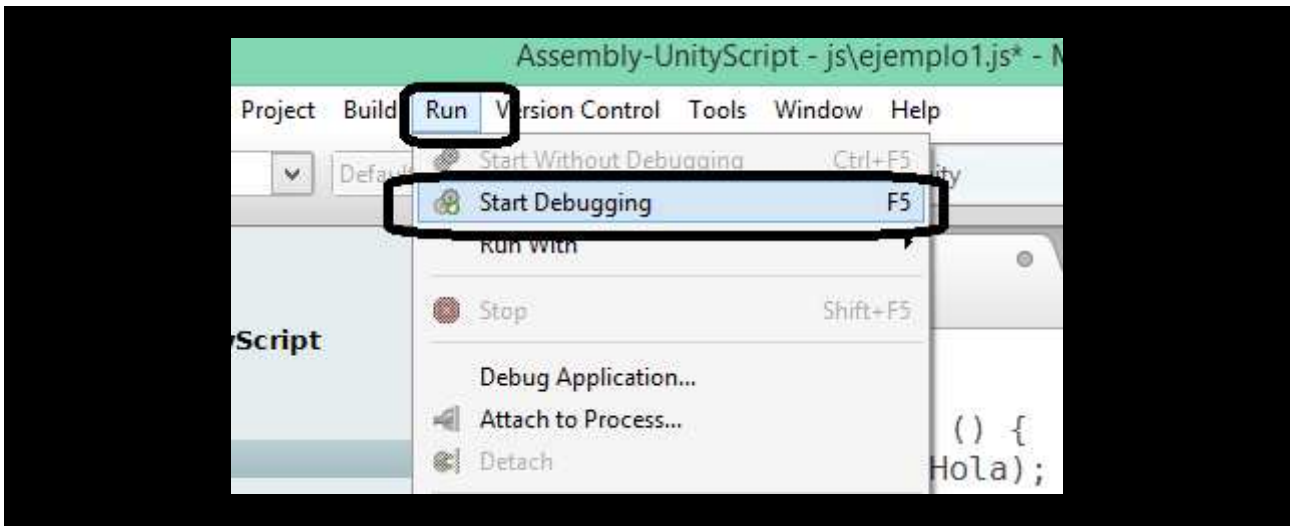
Como primer ejemplo de la generación de código, vamos a asociar un mensaje al cubo el cual será visible al ingresar a modo juego. Dejamos el código como se muestra a continuación, eliminando la función Update, de la cual nos ocuparemos más adelante.

```
#pragma strict
function Start () {
     Debug.Log('Hola');
}
```

Referencia Debug: http://docs.unity3d.com/ScriptReference/Debug.html

Paso6: Debuggear script

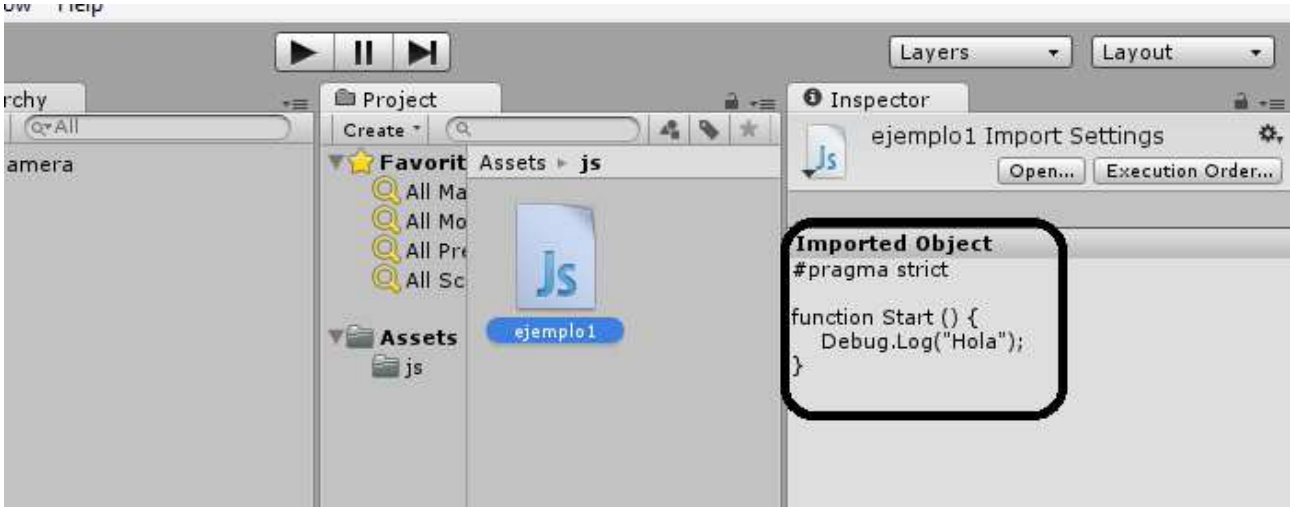
Para chequear que el código realizado no contenga ningún error luego de salvar los cambios, vamos a Run > Start Debugging. De existir algún error este se indicaría sobre el código que escribimos.



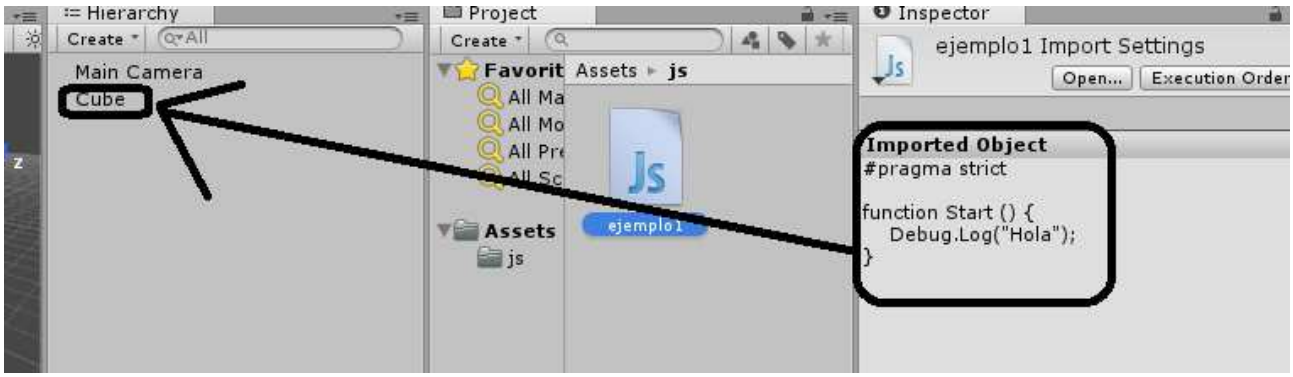
Nota: Si todo está bien, el editor abre una ventana emergente, la cual cerramos.

Paso7: Asignar Script a Game Objet

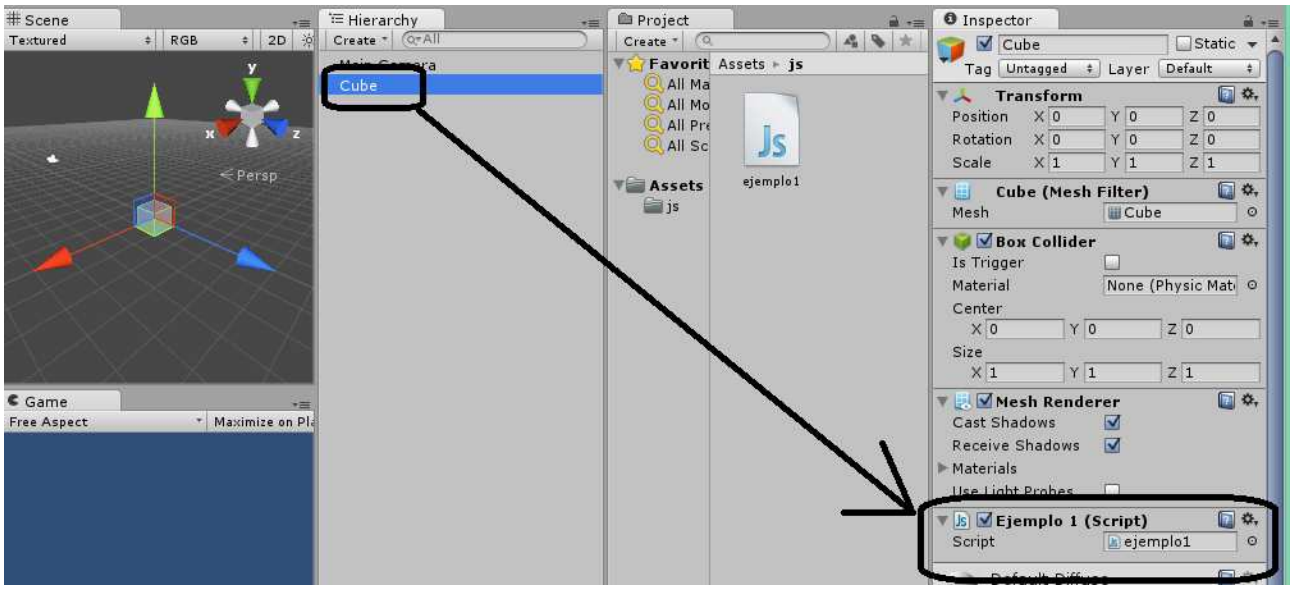
Ahora vamos a la pantalla de Unity y corroboramos que el script se ha actualizado en el Inspector:



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

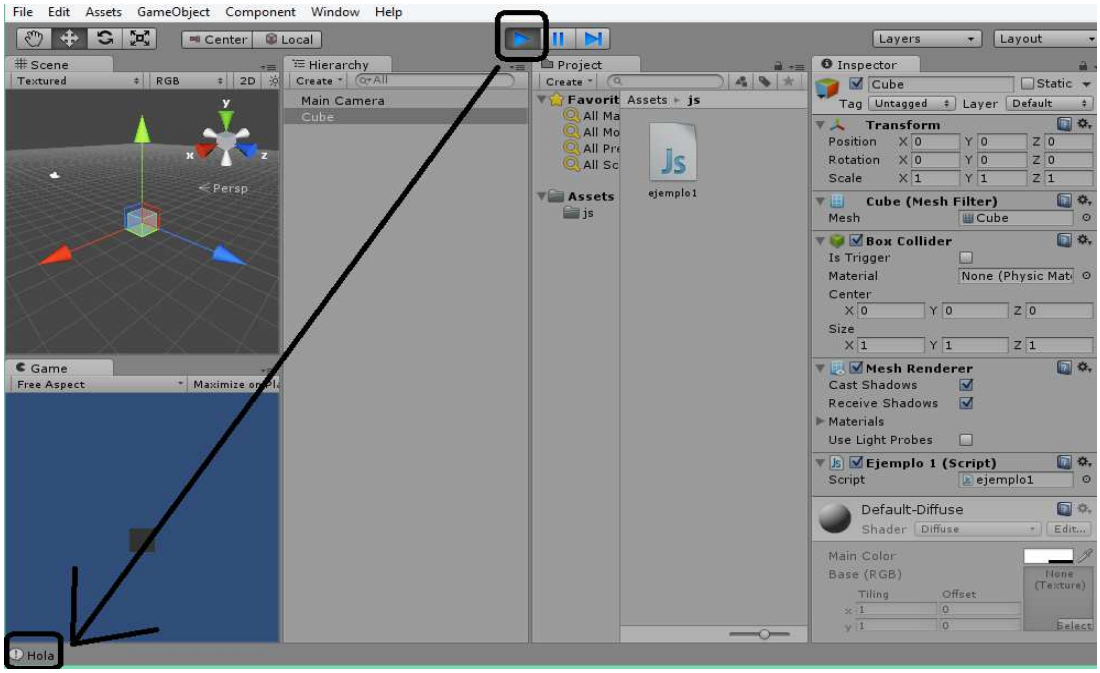


Al seleccionar el cubo en el panel de Jerarquía, ahora podemos ver que en el panel de Inspector figura el script asociado.



Paso8: Visualización en modo juego

Si ahora ingresamos a modo juego, podemos ver en la parte inferior de la pantalla el contenido de Debul.Log.

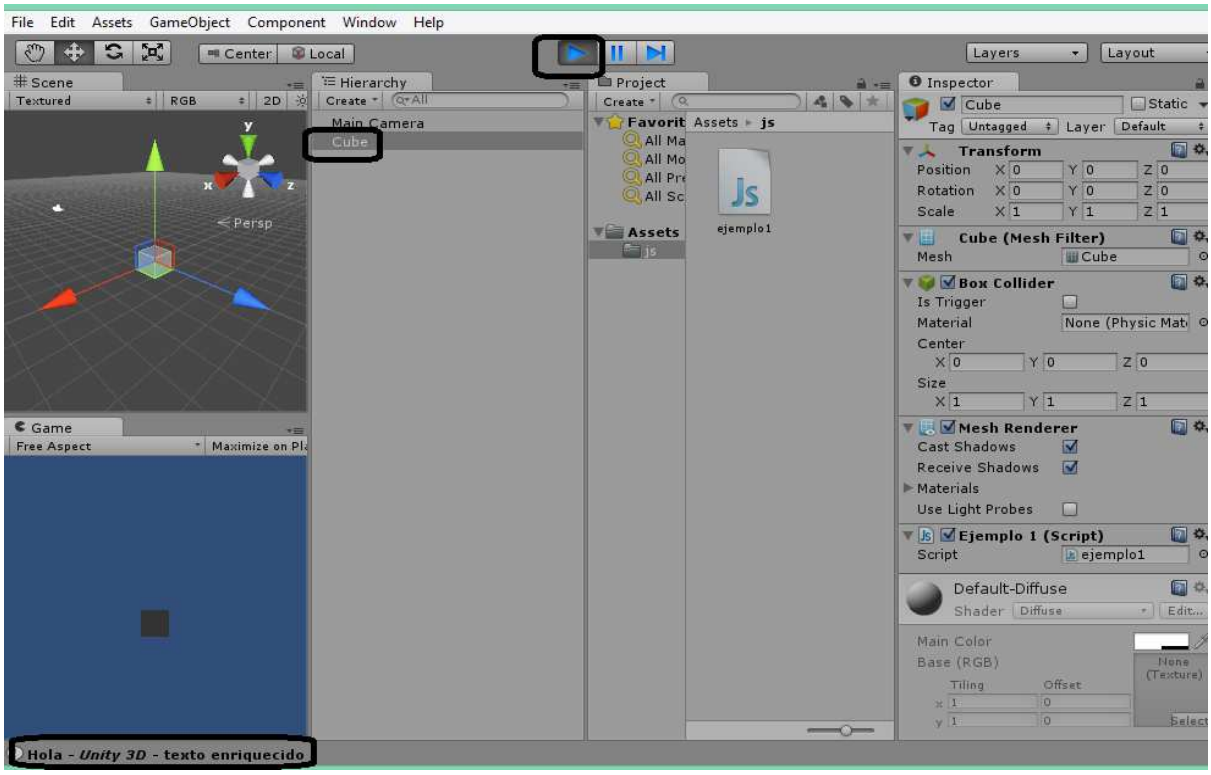


Bloque temático 2: Texto enriquecido.

Unity posee un sistema de texto enriquecido para presentar información inspirado en el formato html, el cual usa un sistema de etiquetas para representar la información. A modo de ejemplo, si modificamos nuestro script con el siguiente código:

```
#pragma strict
function Start () {
     Debug.Log("<b>Hola - <i>Unity 3D</i> - texto enriquecido</b>");
}
```

Podemos ver que al guardar los cambios y ejecutarlo nuevamente, la información se presenta en negrita ya que está entre las etiquetas <b> y Unity 3D aparece en italica ya que utilizamos una etiqueta de tipo <i>.



Referencia: *http://docs.unity3d.com/Manual/StyledText.html*

Tarea 1: Investigar el uso de estilos en la página de referencia, y modificar el script para agregarle colores al texto.

Bloque temático 3: Variables y funciones

Una herramienta fundamental para realizar scripts en cualquier lenguaje de programación, son las variables, estos elementos nos permiten almacenar información de diferente naturaleza, como: strings – números enteros – números booleanos, etc. Por otro lado las funciones nos van a permitir agregarle lógica a nuestro trabajo, desde crear operaciones aritméticas hasta complicados procedimientos.

Por ejemplo, si analizamos un auto podríamos establecer una serie de variables en las cuales almacenar el color, la marca, la velocidad máxima, etc, y una serie de funciones destinadas a indicar la forma en la que el auto acelera o frena, como realiza la combustión, cual es la secuencia de encendido de la radio, etc.

Si sustituimos ahora el código de nuestro script por el siguiente:

```
#pragma strict
var color = " <color=#00ffffff>…Cyan</color>";
function Start () {
      Debug.Log("<b>Hola - <i>Unity 3D</i> - texto enriquecido</b>"  + color);
}
```

Podemos observar dos puntos importante:

1.- Hemos pasado el valor de una variable que contiene una etiqueta y un texto dentro de Debug.Log().

2.- Utilizamos para concatenar el valor de la variable y el texto el signo de +.

Nota: Los strings van entre comillas simples o dobles, mientras que las variables y números no van entre comillas.

Las funciones por su lado presentan la siguiente estructura:

```
function nombreDeFunción(atributo1, atributo2, …....., atributoN){
      código a ejecutar;
}
```

1.- Comienzan con la declaración de función: function

2.- Le sigue el nombre de la función

3.- Entre paréntesis se declaran los atributos a ingresar dentro de la función para ser utilizados en el código.

4.- El código a utilizar por la función se escribe entre llaves.

5.- Cada línea de código finaliza con punto y coma.

Para ejecutar una función se escribe su nombre seguido de los parámetros a ser pasados y finalizando con punto y coma:

```
nombreDeFunción(parámetro1, parámetro, …....., parámetroN);
```

Veamos un ejemplo en donde la función realiza la suma de dos números.

```
#pragma strict


var valor1 : int = 3;
function Start () {
var suma = Sumar(valor1, 4);
Debug.Log(suma);
}

function Sumar(primerValor : int, segundoValor : int) {
var resultadoSuma : int;
var resultadoSumaFormato;
resultadoSuma = primerValor + segundoValor;
resultadoSumaFormato    =    "<b><color=cyan>"    +    resultadoSuma    +
"</color></b>";
return resultadoSumaFormato;
}
```

En el ejemplo anterior agrupamos muchos conceptos:

1.- Se declara una variable **global** (fuera de cualquier función) llamara **valor1**

2.- Creamos la función **Sumar**, la cual es invocada desde la función Start y la cual se le pasan dos parámetros, el primero es la variable valor1 y el segundo es un número entero.

3.- La función Sumar recibe dos atributos declarados de tipo entero (int)

4.- Dentro de la función Sumar se crea una variable **local** (**resultadoSuma**) de tipo entero, que realiza la suma de los atributos que ingresan a la función.

5.- La función **resultadoSumaFormato**, le agrega color y negrita al resultado.

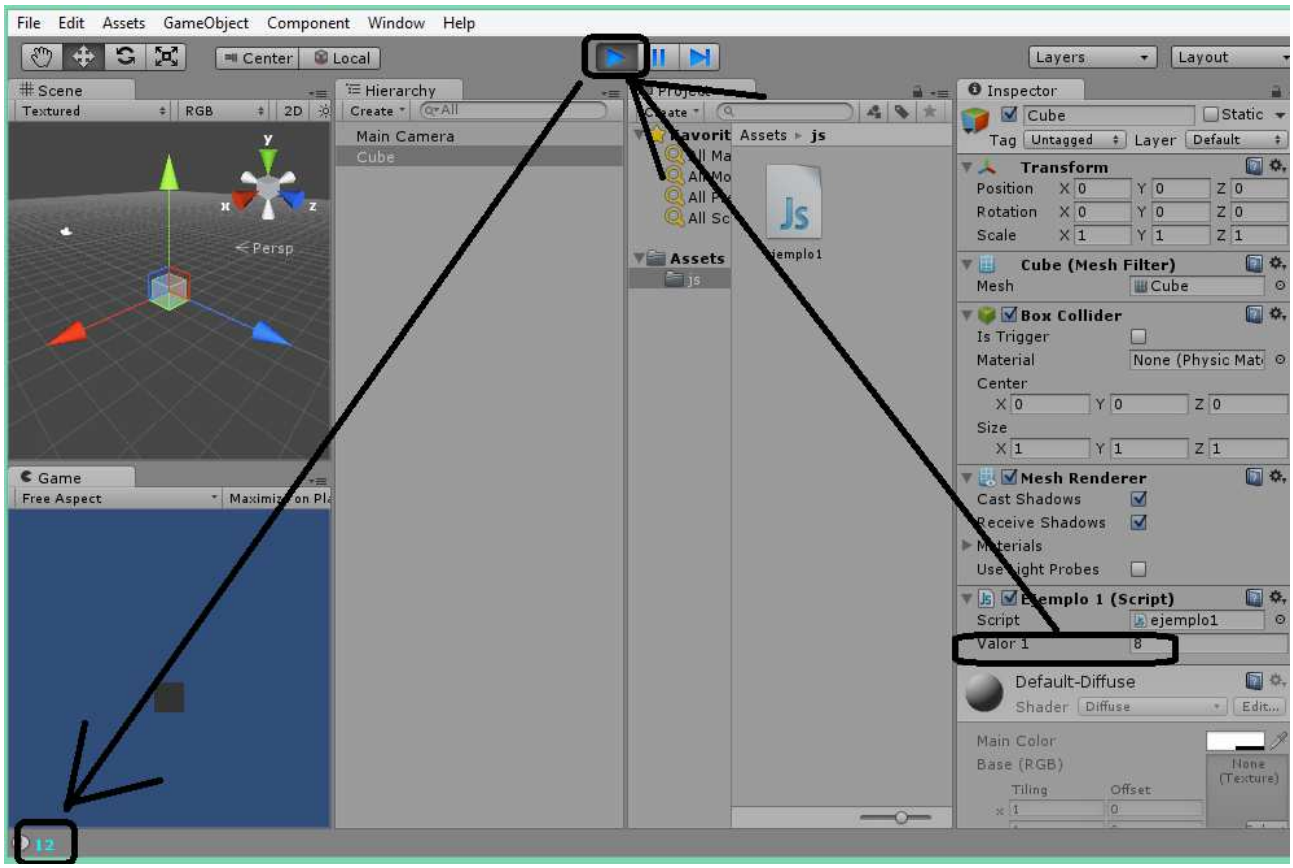
6.- El contenido de la variable anterior, lo retornamos mediante **return**.

El resultado se muestra a continuación:

Nota: Existen tres funciones fundamentales asociadas al juego: Awake(), Start() y Update

1.- Awake() se ejecuta al iniciar el juego y su función es instanciar todos los objetos (instanciar es un concepto que viene de la Programación Orientada a Objetos, que significa crear un objeto de una determinada clase) y a continuación se leen y ejecutan los scripts que tengan asociados.
2.- una vez que Awake() finalizo de ejecutarse, se ejecuta Start(), esta función se ejecuta cuando comenzamos el nivel en el cual nos encontramos, todo lo que esta dentro de esta función se ejecuta al entrar en una nueva pantalla.
3.- Finalmente Update() se ejecuta en cada render de la escena, permitiendo no solo la visualización, sino también la interacción con los elementos de la pantalla en cada frame.

El valor de la variable puede ser modificado desde el panel “Inspector” antes de ingresar a modo juego



Bloque temático 4: Tipo de declaración de variables

Las variables que declaremos fuera de una función, son consideradas globales, y pueden ser accedidas desde el panel Inspector. A continuación vamos a realizar un script que asociaremos al cubo anterior, en el cual declararemos una serie de variables globales de diferentes tipos, y veremos como se presentan y como podemos acceder a ellas en el panel Inspector.

Variables globales:

Tipo 1: Sin tipo de datos ni inicialización

Este tipo de variables no aparecen en el panel Inspector

Ejemplo: `var variable1;`

Tipo 2: Con tipo de datos y sin inicializar

Este tipo de variables si aparece en el panel Inspector y aparecen por defecto con un valor de cero.

Ejemplo: `var variable2 : int;`

Tipo 3: Sin tipo de datos pero inicializada

Este tipo de variables si aparece en el panel Inspector de Unity en base al valor dado por defecto.

Ejemplo: `var variable3 = 7;`

Tipo 4: Con tipo de datos e inicializada

Este tipo de datos aparece en el panel de gerarquía.

Ejemplo: `var variable4 : int = 4;`

Tipo 5: Inicializada durante el juego

Cuando declaramos una variable, y la inicializamos en otra línea, esta adquiere el valor por defecto recién cuando apretamos play.

Ejemplo:

```
var variable5 : int;

variable5 = 3;
```

Tipo 6: Variables privadas

Cuando anteponemos el prefijo “private” a una variable, esta se torna privada y no aparece en el inspector.

Ejemplo: `private var variable6 : int;`

Tipo 7: Variables de tipo arrastrar

Este tipo de variables al ser del tipo de Unity, aparece con una punto dentro de un circulo a la derecha, el cual indica que se puede puede importar desde el juego un componente del tipo declarado para asignar el valor a la variable. La asignación se realiza o bien arrastrando el objeto desde la carpeta o el panel de gerarquía, o seleccionando mediante hacer click en el circulito.

Ejemplo7: `var variable7 : GameObject;`

Variables locales:

Las variables locales son declaradas dentro de las funciones y no son públicas, por lo que no aparecen en el inspector.

Ejemplo:

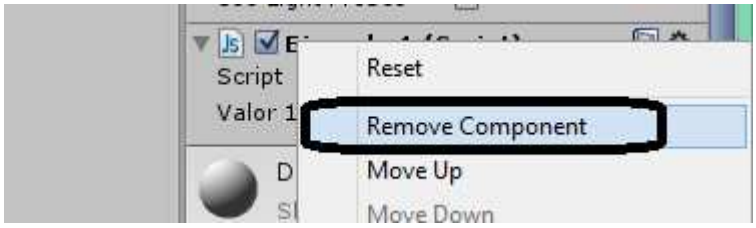
```
function Update() {
var variable8 : int;
}
```

En particular la función Update() es llamada en cada frame por Unity.

Tarea 2: crear script que posea todos los tipos de variables indicados aquí y analizar su comportamiento.

Bloque temático 5: Quitar script a objeto

Si por algún motivo debemos quitarle el script a un objeto, lo podemos hacer rápidamente seleccionando el objeto en el panel del jerarquía y luego en el panel Inspector hacemos click derecho sobre el script y presionamos: Remove Component



Bloque temático 6: Operadores aritméticos, de asignación, de comparación y lógicos.

Cuando comenzamos a darle lógica a nuestros scripts, necesitamos contar con operadores matemáticos que nos permitan sumar, restar, comparar etc. Estas herramientas se agrupan según cuatro categorías y se listan a continuación, no es necesario su aprendizaje de memoria, deben de ser incorporadas paulatinamente a medida que avanzamos:

6.1. Operadores aritméticos

Operator	Descripción	Ejemplo	Valor de x	Valor inicial y	Resultado de x	Resultado de y
+	Suma	y=x+2	2			4
-	Resta	y=x-2	3			1
*	Multiplicación	y=x*2	2			4
/	División	y=x/2	6			3
%	Resto	y=x%2	5			1
++	Incremento	y=++		3		4
		y=++x	4		5	5
		y=x++	4		5	4
	Decremento	y=--		3		2
		y=--x	4		3	3
		y=x--	4		3	4

6.4. Operadores lógicos.

Operador	Descripción	Ejemplo
&&	and	(x < 10 && y > 1) es true
||	or	(x==5 || y==5) es false

6.2. Operadores de asignación

Operator	Ejemplo	Igual a:	Resultado
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

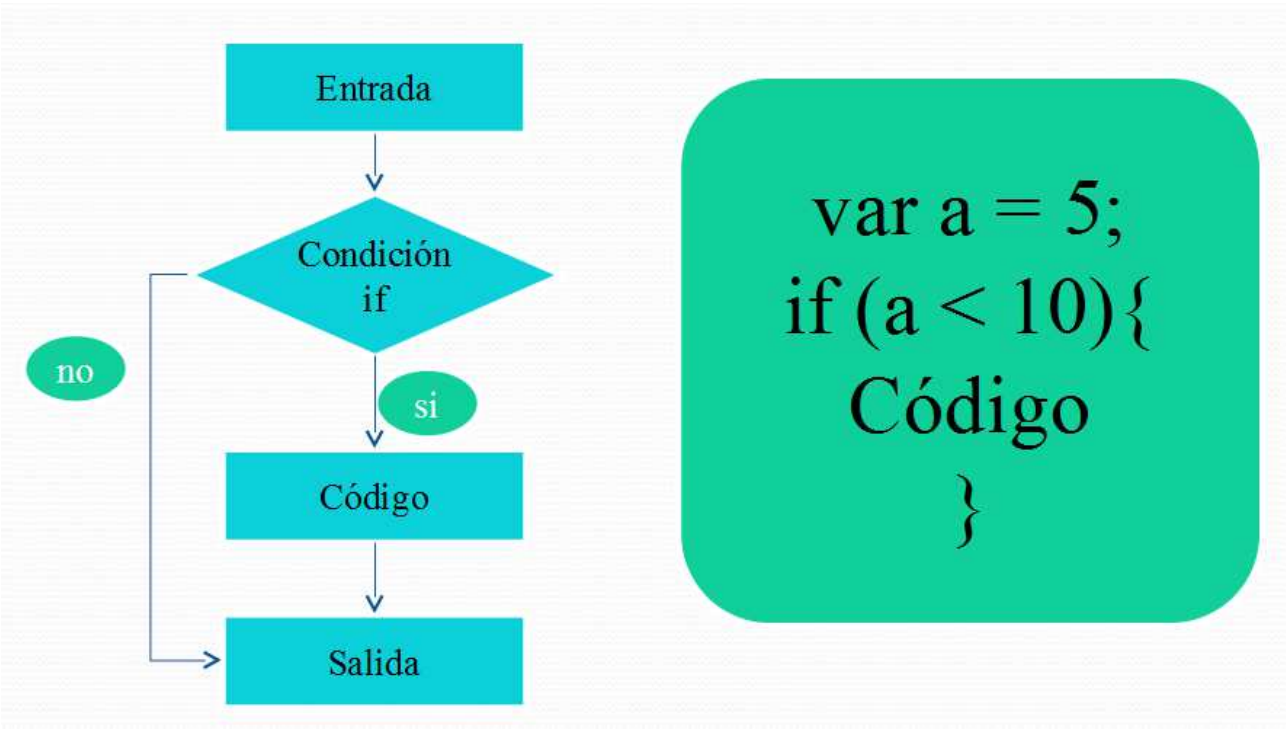
6.3. Operadores de comparación

Operador	Descripción	Comparación	Resultado
==	Es igual a:	x==8	*false*
		x==5	*true*
===	Es exactamente igual a: (Valor y tipo)	x==="5"	*false*
		x===5	*true*
!=	No es igual a:	x!=8	*true*
!==	No es igual: (ni en valor ni en tipo)	x!=="5"	*true*
		x!==5	*false*
>	Es mayor que:	x>8	*false*
<	Es menor que:	x<8	*true*
>=	Es mayor o igual a:	x>=8	*false*
<=	Es menor o igual a:	x<=8	*true*

Bloque temático 7: Estructuras de control

Otro componente fundamental en nuestros scripts lo constituyen los bucles y las estructuras de control, diseñados para cumplir con la tarea de tomar decisiones, seleccionar elementos, etc. Todos los lenguajes de programación cuentan con estas herramientas aún cuando su sintaxis puede variar un poco entre lenguajes. A continuación veremos el uso de: if – if/else – while – do/while – for – switch.

7.1. if



La estructura if nos sirve para tomar decisiones, en donde si una determinada condición indicada entre parentesis se cumple, se ejecuta el código que se encuentra entre llaves, sino el código no se ejecuta, por ejemplo : si la llave introducida en una puerta es la que habré la puerta, que salga un mensaje de felicitaciones. A continuación se presenta la estructura lógica y un ejemplo de código.

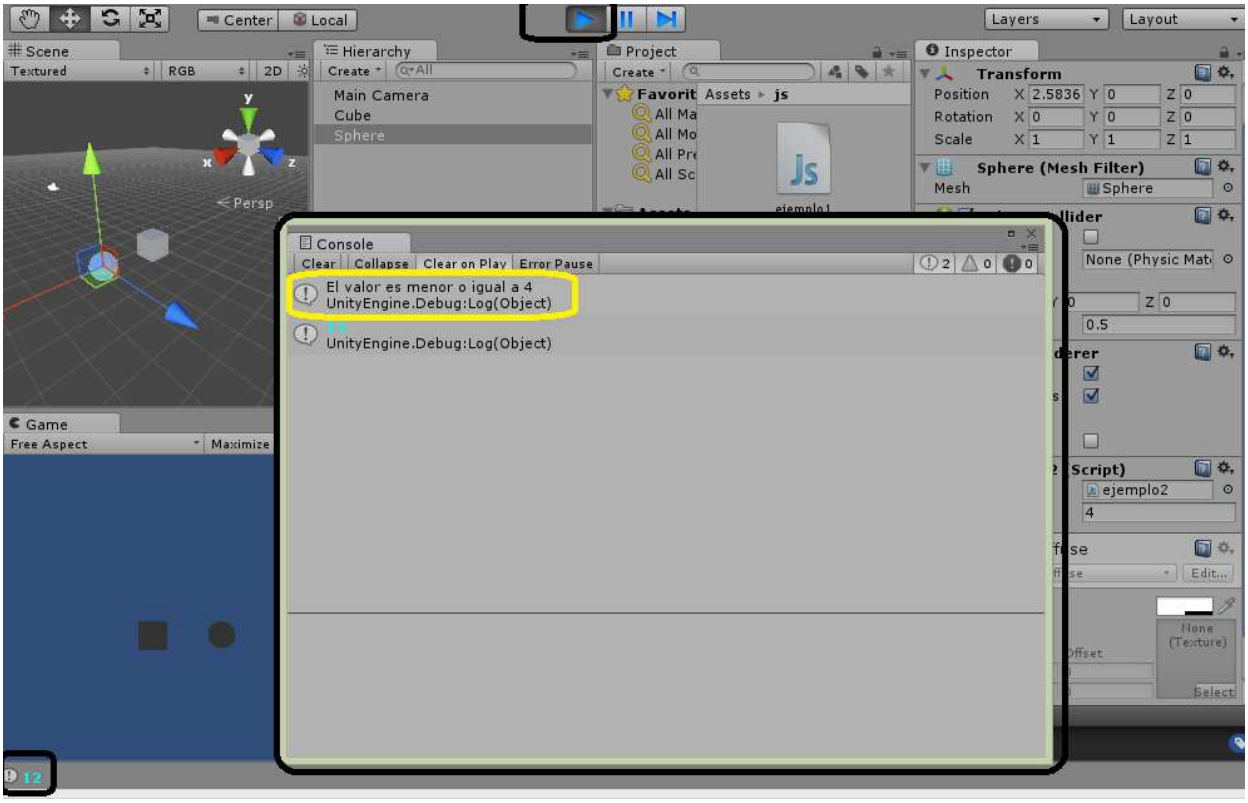
Tarea 3: Crear una esfera y asociarle el siguiente script:

```
#pragma strict
var variable : int = 4;
function Start () {
     if(variable<=4){
          Debug.Log("El valor es menor o igual a 4");
     }
}
```

Nota1: Prestar atención que al seleccionar la esfera, en el panel inspector podemos modificar el valor de la variable.

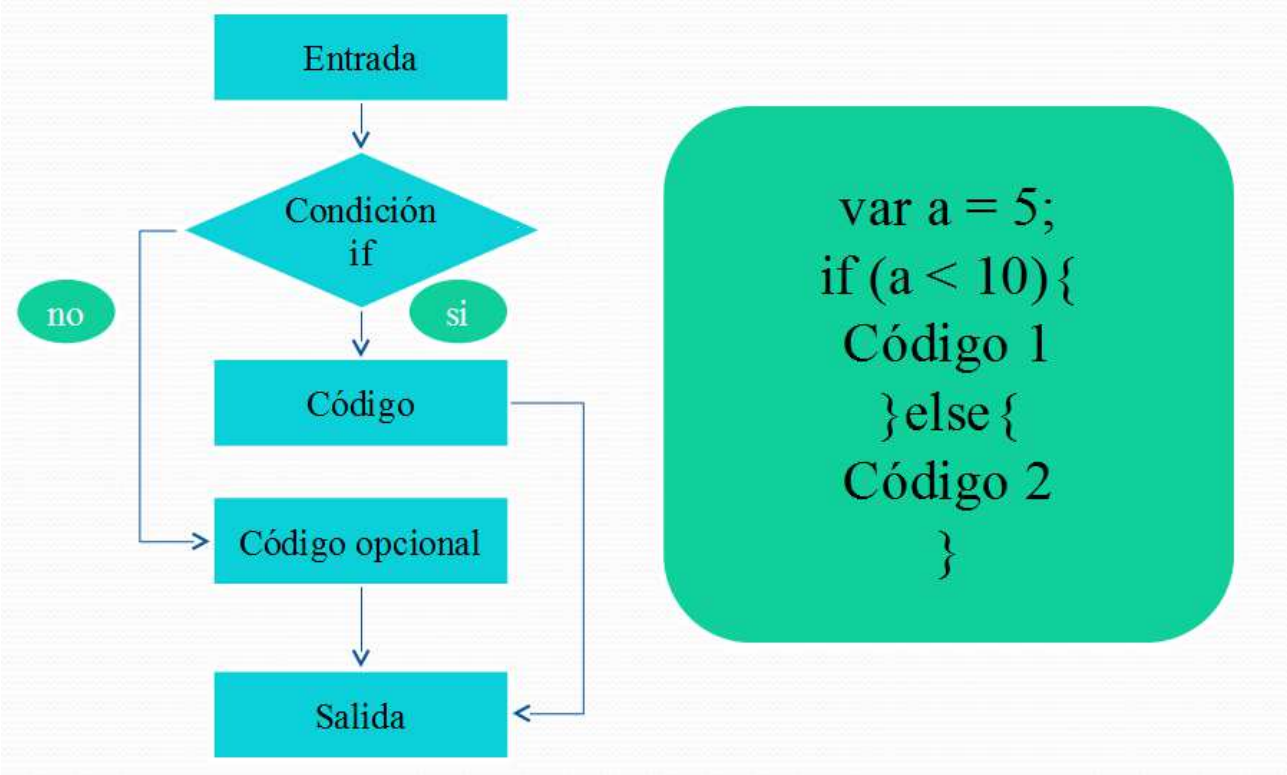
Nota2: En windows podemos presionar (shift + ctrl + c) para ver el panel de control con todos los mensajes

Nota3: Si el valor de la variable no cumple la condición, entonces el mensaje no sale en el panel de control.



7.2. if/else if/else

Una modificación al caso anterior es la estructura if/else, en la cual si la condición en el if no se cumple, entonces se ejecuta la segunda condición que se encuentra dentro del else. Este tipo de estructura lo utilizamos todo el tiempo en diferentes plataformas, cada vez que nos sale una ventana emergente en la cual tenemos que aceptar o cancelar.



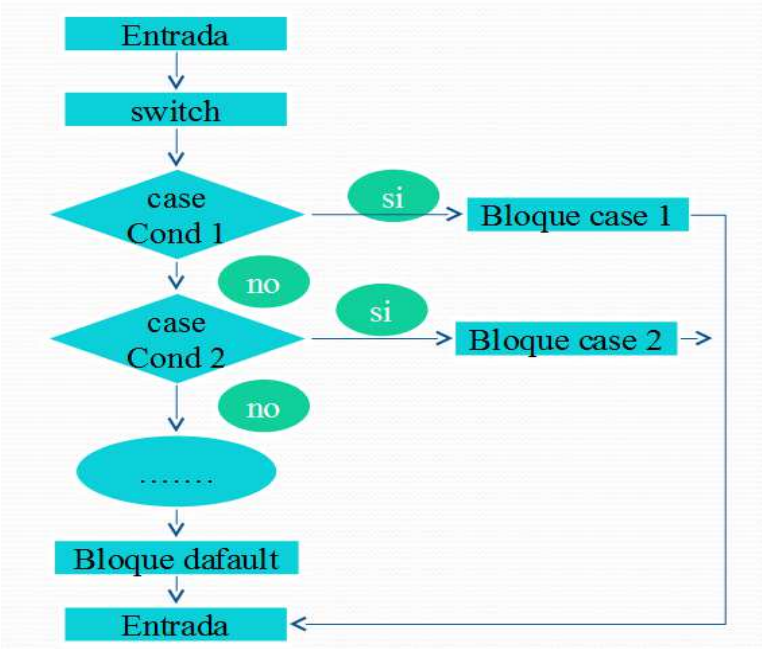
Una variante es considerear un if luego de un else como en siguiente ejemplo del cual queda como tarea ejecutar y testear:

```
#pragma strict

var variable : int = 4;

function Start () {
     if(variable<=4){
          Debug.Log("El valor es menor o igual a 4");
     }else if(variable<=7){
          Debug.Log("El valor es menor o igual a 7");
```

```
    }else{
        Debug.Log("El valor es mayor a 7");
    }
}
```

7.3. switch

Una opción que podemos utilizar en ciertos casos con una eficiencia mayor a la estructura anterior, es la estructura switch, mientras que en la estructura if/else if necesitamos recorrer todas las opciones hasta encontrar la condición que se debe ejecutar, en este caso de forma más directa se ejecuta la opción que coincide con la variable de entrada.



Tarea 4: En el siguiente ejemplo:

1.- Notar que tenemos la opción de determinar un valor por defecto, el cual se ejecuta si ninguna de las condiciones anteriores se cumplen.

```
#pragma strict

var i : int = 5;

     function Start () {
          switch (i) {
          case 1:    Debug.Log("El valor es igual a 1");
          break;
          case 2:    Debug.Log("El valor es igual a 2");
          break;
          case 3:    Debug.Log("El valor es igual a 3");
          break;
          case 4:    Debug.Log("El valor es igual a 4");
          break;
          default: Debug.Log("El valor es mayor a 4");
     }
}
```

2.- Asignarle a un Game Object el ejemplo anterior y analizar su funcionamiento.

3.- ¿Qué pasa si el valor de la variable es 1 pero elimino las instrucciones break?

4.- Investigar el uso de “Random.Range” para generar una variable i aleatoria.

Resumen de tareas:

Tarea 1 - Investigar el uso de estilos en la página de referencia, y modificar el siguiente script para agregarle colores al texto.

```
#pragma strict
function Start () {
     Debug.Log("<b>Hola - <i>Unity 3D</i> - texto enriquecido</b>");
}
```

Tarea 2 - crear script que posea todos los tipos de variables indicados en esta unidad

Tarea 3 - Crear una esfera y asociarle el siguiente script:

```
#pragma strict
var variable : int = 4;
function Start () {
     if(variable<=4){
          Debug.Log("El valor es menor o igual a 4");
     }
}
```

Nota1: Prestar atención que al seleccionar la esfera, en el panel inspector podemos modificar el valor de la variable.

Nota2: En windows podemos presionar (shift + ctrl + c) para ver el panel de control con todos los mensajes

Nota3: Si el valor de la variable no cumple la condición, entonces el mensaje no sale en el panel de control.

Tarea 4 - En el siguiente ejemplo:

1.- Notar que tenemos la opción de determinar un valor por defecto, el cual se ejecuta si ninguna de las condiciones anteriores se cumplen.

```
#pragma strict

var i : int = 5;

     function Start () {
          switch (i) {
```

```
            case 1:    Debug.Log("El valor es igual a 1");
            break;
            case 2:    Debug.Log("El valor es igual a 2");
            break;
            case 3:    Debug.Log("El valor es igual a 3");
            break;
            case 4:    Debug.Log("El valor es igual a 4");
            break;
            default: Debug.Log("El valor es mayor a 4");
        }
}
```

2.- Asignarle a un Game Object el ejemplo anterior y analizar su funcionamiento.

3.- ¿Qué pasa si el valor de la variable es 1 pero elimino las instrucciones break?

4.- Investigar el uso de "Random.Range" para generar una variable i aleatoria.

Subir como entregable un print de pantalla en cada caso, y un archivo Word con las respuestas de las preguntas y los trabajos de investigación.

Bibliografía utilizada y sugerida

Documentación oficial online -

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

Lo que vimos

En esta unidad nos hemos introducido en algunos elementos básicos de la programación en JavaScript aplicado a Unity.

Lo que viene:

En la siguiente unidad seguiremos ampliando los conocimientos de programación.