Universidad la Salle Noroeste

**Event Functions**

Programación de videojuegos

Ing. En Producción multimedia.

05 de febrero de 2020

Cd. Obregón, Sonora

**First Scene load**

**Awake**

**MonoBehaviur.Awake()**

Se utiliza para inicializar una variable o un state antes de comenzar el juego. Awake se llama solo una vez durante la instancia del script al igual que un constructor. Se utiliza Awake una vez que todos los objetos se inicializan, para poder llamarlos o consultarlos. El Awake de cada objeto inicializado se llama en orden aleatorio. Se utiliza Awake para configurar referencias entre scripts y Start para pasar información.

Awake es la primera función que se llama al inicio para ordenar la inicialización de scripts. Awake no actúa como una rutina.

Ejemplo.

using UnityEngine;

public class ExampleClass : [MonoBehaviour](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html)

{

private [GameObject](https://docs.unity3d.com/ScriptReference/GameObject.html) target;

void Awake()

{

target = [GameObject.FindWithTag](https://docs.unity3d.com/ScriptReference/GameObject.FindWithTag.html)("Player");

}

}

**OnEnable**

**MonoBehaviur.OnEnable()**

Esta función se llama cuando el objeto se habilita y se activa.

Ejemplo.

using UnityEngine;

[[ExecuteInEditMode](https://docs.unity3d.com/ScriptReference/ExecuteInEditMode.html)]

public class PrintOnOff : [MonoBehaviour](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html)

{

void OnDisable()

{

[Debug.Log](https://docs.unity3d.com/ScriptReference/Debug.Log.html)("PrintOnDisable: script was disabled");

}

void OnEnable()

{

[Debug.Log](https://docs.unity3d.com/ScriptReference/Debug.Log.html)("PrintOnEnable: script was enabled");

}

void [Update](https://docs.unity3d.com/ScriptReference/PlayerLoop.Update.html)()

{

#if UNITY\_EDITOR

[Debug.Log](https://docs.unity3d.com/ScriptReference/Debug.Log.html)("[Editor](https://docs.unity3d.com/ScriptReference/Editor.html) causes this [Update](https://docs.unity3d.com/ScriptReference/PlayerLoop.Update.html)");

#endif

}

}

**OnLevelWasLoaded**

**MonoBehaviur.OnLevelWasLoaded(int)**

Esta función es ejecutada para informarle al juego que un nuevo nivel ha sido cargado.

Ejemplo.

using UnityEngine;

using System.Collections;

public class ExampleClass : [MonoBehaviour](https://docs.unity3d.com/es/530/ScriptReference/MonoBehaviour.html) {

void OnLevelWasLoaded(int level) {

if (level == 13)

print("Woohoo");

}

}

Antes que se ejecute el primer frame.

**Start [Inicio]**

La acción de inicializar o comenzar es ejecutada al momento en el que el primer frame (cuadro por segundo) da inicio a realizar la acción para la cual fue diseñada, siempre y cuando la orden dentro del script, se encuentre en activo.

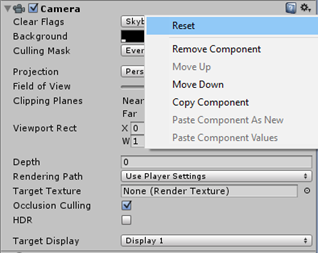


**Dato importante:** Naturalmente esta acción **NO** podrá ser ejecutado cuando un objeto es instanciado durante el gameplay.

**Editor**

**Restablecer [RESET]**

Se le conoce como restablecer o reset, a la reiniciación de propiedades en un script o guiones, nombres que recibe el código. Al aplicar este método, los valores se ajustan a los valores originales, es decir, a los que tenía antes de comenzar una acción.



**In between frames**

OnApplicationPause:

Esto es llamado al final del frame dónde la pausa es detectada, efectivamente entre actualizaciones de frame normales. Un frame extra será despachado después de que OnApplicationPause sea llamado para permitirle al juego mostrarle gráficas que indican el estado pausado.



using UnityEngine;

public class AppPaused : [MonoBehaviour](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html)

{

bool isPaused = false;

void OnGUI()

{

if (isPaused)

[GUI.Label](https://docs.unity3d.com/ScriptReference/GUI.Label.html)(new [Rect](https://docs.unity3d.com/ScriptReference/Rect.html)(100, 100, 50, 30), "Game paused");

}

void OnApplicationFocus(bool hasFocus)

{

isPaused = !hasFocus;

}

void OnApplicationPause(bool pauseStatus)

{

isPaused = pauseStatus;

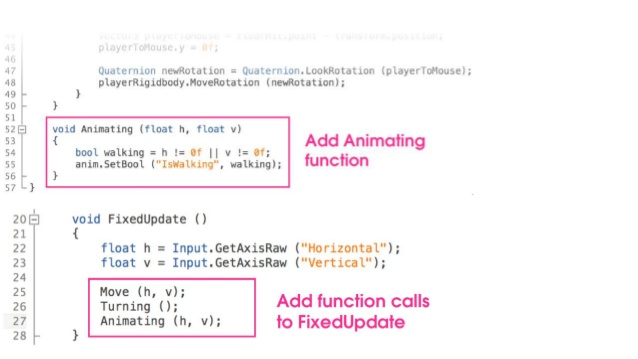
}

}

**Update Order**

FixedUpdate:

FixedUpdate a veces es más llamada que Update. Puede ser llamada varias veces por frame, si la velocidad de frame es baja y puede no ser llamada entre frames en absoluto si la velocidad de frame es alta. Todos los cálculos de física y actualizaciones ocurren inmediatamente después de FixedUpdate. Cuando aplique cálculos de movimiento dentro de FixedUpdate, usted no necesita multiplicar sus valores por Time.deltaTime. Esto se debe a que FixedUpdate en un temporizador fiable, independiente de la velocidad de frames.



using System.Collections;

using System.Collections.Generic;

using UnityEngine;

// GameObject.FixedUpdate example.

//

// Measure frame rate comparing [FixedUpdate](https://docs.unity3d.com/ScriptReference/PlayerLoop.FixedUpdate.html) against [Update](https://docs.unity3d.com/ScriptReference/PlayerLoop.Update.html).

// Show the rates every second.

public class ExampleScript : [MonoBehaviour](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html)

{

private float updateCount = 0;

private float fixedUpdateCount = 0;

private float updateUpdateCountPerSecond;

private float updateFixedUpdateCountPerSecond;

void Awake()

{

// Uncommenting this will cause framerate to drop to 10 frames per second.

// This will mean that [FixedUpdate](https://docs.unity3d.com/ScriptReference/PlayerLoop.FixedUpdate.html) is called more often than [Update](https://docs.unity3d.com/ScriptReference/PlayerLoop.Update.html).

//[Application.targetFrameRate](https://docs.unity3d.com/ScriptReference/Application-targetFrameRate.html) = 10;

StartCoroutine(Loop());

}

// Increase the number of calls to [Update](https://docs.unity3d.com/ScriptReference/PlayerLoop.Update.html).

void [Update](https://docs.unity3d.com/ScriptReference/PlayerLoop.Update.html)()

{

updateCount += 1;

}

// Increase the number of calls to [FixedUpdate](https://docs.unity3d.com/ScriptReference/PlayerLoop.FixedUpdate.html).

void [FixedUpdate](https://docs.unity3d.com/ScriptReference/PlayerLoop.FixedUpdate.html)()

{

fixedUpdateCount += 1;

}

// Show the number of calls to both messages.

void OnGUI()

{

[GUIStyle](https://docs.unity3d.com/ScriptReference/GUIStyle.html) fontSize = new [GUIStyle](https://docs.unity3d.com/ScriptReference/GUIStyle.html)(GUI.skin.GetStyle("label"));

fontSize.fontSize = 24;

[GUI.Label](https://docs.unity3d.com/ScriptReference/GUI.Label.html)(new [Rect](https://docs.unity3d.com/ScriptReference/Rect.html)(100, 100, 200, 50), "[Update](https://docs.unity3d.com/ScriptReference/PlayerLoop.Update.html): " + updateUpdateCountPerSecond.ToString(), fontSize);

[GUI.Label](https://docs.unity3d.com/ScriptReference/GUI.Label.html)(new [Rect](https://docs.unity3d.com/ScriptReference/Rect.html)(100, 150, 200, 50), "[FixedUpdate](https://docs.unity3d.com/ScriptReference/PlayerLoop.FixedUpdate.html): " + updateFixedUpdateCountPerSecond.ToString(), fontSize);

}

// [Update](https://docs.unity3d.com/ScriptReference/PlayerLoop.Update.html) both CountsPerSecond values every second.

IEnumerator Loop()

{

while (true)

{

yield return new [WaitForSeconds](https://docs.unity3d.com/ScriptReference/WaitForSeconds.html)(1);

updateUpdateCountPerSecond = updateCount;

updateFixedUpdateCountPerSecond = fixedUpdateCount;

updateCount = 0;

fixedUpdateCount = 0;

}

}

}

**Update**

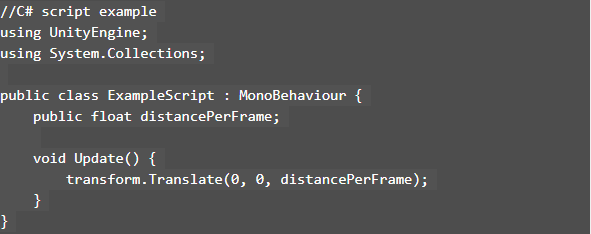
**Update** se llama una vez por frame. Es la función principal para las actualizaciones de frames.

Al correr un juego, Unity automáticamente va a tomar todos los MonoBehaviours que haya en la escena y va a realizar la ejecución de los métodos Update antes de mostrar cada fotograma del juego. De modo que no se debe hacer la ejecución de este método manualmente, el motor se encargará de ello.

Quiere decir que la función Update se ejecutará de manera periódica mientras el juego esté corriendo.

Si el juego funciona a 60 FPS (frames por segundo) la función Update se ejecutará 60 veces por segundo.

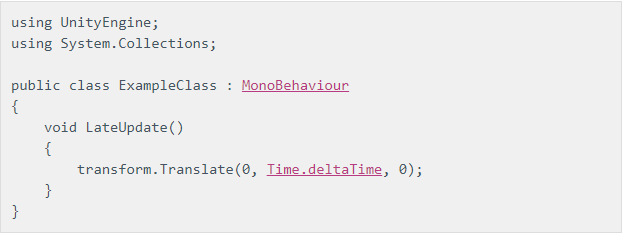
**Ejemplo:**



**Lateupdate**

Lateupdate es llamada una vez por frame, después de que Update haya finalizado. Cualquier cálculo que sea realizado en Update será completado cuando lateupdate comience. Un uso común para lateupdate sería una cámara de tercera persona que sigue. Si usted hace que su personaje se mueva y gire, Update, usted puede realizar todos los cálculos de movimientos de cámara y rotación en lateupdate. Esto asegurará que el personaje haya sido movido completamente antes de que la cámara haga un seguimiento a su posición.

**Ejemplo:**



**Animation Update Loop**

***OnStateMachineEnter****:*

Durante la actualización de la máquina de estado, está devolución de llamada, se realiza en el primer marco de actualización cuando la máquina de estado de un controlador realiza una transición que fluye a través de un estado de **entrada**. No se requiere una transición a un subestado StateMachine.

Esta devolución de llamada se produce sólo si hay un componente controlador (por ejemplo, AnimatorController o AnimatorOverrideController o AnimatorControllerPlayable) en el gráfico de animación.

El flujo de evaluación debe pasar a través de un nodo entrada para llamar a este método. Agregar esta devolución de llamada evita la evaluación de la máquina de estado multiproceso porque se debe invocar un método de script durante la evaluación de la máquina de estado. Se llama a este método mientras se evalúa la máquina de estado, que es un orden de ejecución diferente en comparación con la mayoría de los métodos StateMachineBehaviour.

Ejemplo:

public void **OnStateMachineEnter**([Animator](https://docs.unity3d.com/ScriptReference/Animator.html) **animator**, int **stateMachinePathHash**);

**OnStateMachineExit**:

Durante el paso Actualización de máquina de estado, está devolución de llamada, se llama en el último marco de actualización cuando la máquina de estado de un controlador realiza una transición que fluye a través de un estado de **salida**. No se requiere una transición a un subestado StateMachine.

Esta devolución de llamada se produce sólo si hay un componente controlador (por ejemplo, AnimatorController o AnimatorOverrideController o AnimatorControllerPlayable) en el gráfico de animación.

El flujo de evaluación debe pasar a través de un nodo Entrada para llamar a este método. Agregar esta devolución de llamada evita la evaluación de la máquina de estado multiproceso porque se debe invocar un método de script durante la evaluación de la máquina de estado. Se llama a este método mientras se evalúa la máquina de estado, que es un orden de ejecución diferente en comparación con la mayoría de los métodos StateMachineBehaviour.

Ejemplo:

public void **OnStateMachineExit**([Animator](https://docs.unity3d.com/ScriptReference/Animator.html) **animator**, int **stateMachinePathHash**);

**StateMachineBehaviour**:

Una capa puede tener hasta 3 estados activos: estado actual, estado interrumpido y estado siguiente. Se llama a esta función para cada estado activo con un componente StateMachineBehaviour que define la devolución de llamada OnStateEnter, OnStateUpdate o OnStateExit.

La función se llama primero para el estado actual, luego para el estado interrumpido y finalmente para el siguiente estado.

Este paso ocurre solo si hay un componente controlador (por ejemplo, AnimatorController o AnimatorOverrideController o AnimatorControllerPlayable) en el gráfico de animación.

Ejemplo:

using UnityEngine;

public class AttackBehaviour : [StateMachineBehaviour](https://docs.unity3d.com/ScriptReference/StateMachineBehaviour.html)

{

public [GameObject](https://docs.unity3d.com/ScriptReference/GameObject.html) particle;

public float radius;

public float power;

protected [GameObject](https://docs.unity3d.com/ScriptReference/GameObject.html) clone;

override public void OnStateEnter([Animator](https://docs.unity3d.com/ScriptReference/Animator.html) animator, [AnimatorStateInfo](https://docs.unity3d.com/ScriptReference/AnimatorStateInfo.html) stateInfo, int layerIndex)

{

clone = Instantiate(particle, animator.rootPosition, [Quaternion.identity](https://docs.unity3d.com/ScriptReference/Quaternion-identity.html)) as [GameObject](https://docs.unity3d.com/ScriptReference/GameObject.html);

[Rigidbody](https://docs.unity3d.com/ScriptReference/Rigidbody.html) rb = clone.GetComponent<[Rigidbody](https://docs.unity3d.com/ScriptReference/Rigidbody.html)>();

rb.AddExplosionForce(power, animator.rootPosition, radius, 3.0f);

}

override public void OnStateExit([Animator](https://docs.unity3d.com/ScriptReference/Animator.html) animator, [AnimatorStateInfo](https://docs.unity3d.com/ScriptReference/AnimatorStateInfo.html) stateInfo, int layerIndex)

{

Destroy(clone);

}

override public void OnStateUpdate([Animator](https://docs.unity3d.com/ScriptReference/Animator.html) animator, [AnimatorStateInfo](https://docs.unity3d.com/ScriptReference/AnimatorStateInfo.html) stateInfo, int layerIndex)

{

[Debug.Log](https://docs.unity3d.com/ScriptReference/Debug.Log.html)("On Attack [Update](https://docs.unity3d.com/ScriptReference/PlayerLoop.Update.html) ");

}

override public void OnStateMove([Animator](https://docs.unity3d.com/ScriptReference/Animator.html) animator, [AnimatorStateInfo](https://docs.unity3d.com/ScriptReference/AnimatorStateInfo.html) stateInfo, int layerIndex)

{

[Debug.Log](https://docs.unity3d.com/ScriptReference/Debug.Log.html)("On Attack Move ");

}

override public void OnStateIK([Animator](https://docs.unity3d.com/ScriptReference/Animator.html) animator, [AnimatorStateInfo](https://docs.unity3d.com/ScriptReference/AnimatorStateInfo.html) stateInfo, int layerIndex)

{

[Debug.Log](https://docs.unity3d.com/ScriptReference/Debug.Log.html)("On Attack IK ");

}

}

### **Useful profile markers**

Some of the animation functions shown in the [Script Lifecycle Flowchart](https://docs.unity3d.com/es/2018.4/Manual/ExecutionOrder.html#ScriptLifecycleFlowchart) are not Event functions that you can call; they are internal functions called when Unity processes your animation.

These functions have Profiler Markers, so you can use the [Profiler](https://docs.unity3d.com/es/2018.4/Manual/Profiler.html) to see when in the frame Unity calls them. Knowing when Unity calls these functions can help you understand exactly when the Event functions you do call are executed.

For example, suppose you call [Animator.Play](https://docs.unity3d.com/es/2018.4/ScriptReference/Animator.Play.html) in the **FireAnimationEvents** callback. If you know that the **FireAnimationEvents** callback is fired only after the **State Machine Update** and **Process Graph** functions execute, you can anticipate that your animation clip will play on the next frame, and not right away.

* **State Machine Update:** All state machines are evaluated at this step in the execution sequence. This step occurs only if there is a controller component (for example, [**AnimatorController**](https://docs.unity3d.com/es/2018.4/ScriptReference/Animations.AnimatorController.html) or [**AnimatorOverrideController**](https://docs.unity3d.com/es/2018.4/ScriptReference/AnimatorOverrideController.html) or [**AnimatorControllerPlayable**](https://docs.unity3d.com/es/2018.4/ScriptReference/Animations.AnimatorControllerPlayable.html)) in the animation graph.  
    
  **Note:** State machine evaluation is normally multithreaded, but adding certain callbacks (for example **OnStateMachineEnter** and **OnStateMachineExit**) disables multithreading. See [Animation update loop](https://docs.unity3d.com/es/2018.4/Manual/ExecutionOrder.html#AnimationUpdateLoop) above for details.
* **ProcessGraph:** Evaluates all animation graphs. This includes sampling all animation clips that need to be evaluated, and computing Root Motion.
* **ProcessAnimation:** Blends the results of the animation graph.
* **WriteTransforms:** Writes all animated transforms to the scene from a worker thread.  
    
  Ejemplo:

Performance marker used for profiling arbitrary code blocks.

Use *ProfilerMarker* to mark up script code blocks for the Profiler.

using Unity.Profiling;

public class MySystemClass

{

[ProfilerMarker](https://docs.unity3d.com/2018.3/Documentation/ScriptReference/Unity.Profiling.ProfilerMarker.html) preparePerfMarker = new [ProfilerMarker](https://docs.unity3d.com/2018.3/Documentation/ScriptReference/Unity.Profiling.ProfilerMarker.html)("MySystem.Prepare");

[ProfilerMarker](https://docs.unity3d.com/2018.3/Documentation/ScriptReference/Unity.Profiling.ProfilerMarker.html) simulatePerfMarker = new [ProfilerMarker](https://docs.unity3d.com/2018.3/Documentation/ScriptReference/Unity.Profiling.ProfilerMarker.html)("MySystem.Simulate");

public void UpdateLogic()

{

preparePerfMarker.Begin();

preparePerfMarker.End();

using (simulatePerfMarker.Auto())

}

}

}

**Rendering**

**OnDrawGizmos:**   
Implemente “OnDrawGizmos” si desea dibujar gizmos que también son seleccionables y siempre dibujados.

Esto le permite elegir rápidamente objetos importantes en su escena.

Tenga en cuenta que “OnDrawGizmos” usará una posición del mouse relativa a la Vista de escena.

Esta función no se llama si el componente está contraído en el Inspector. Use “OnDrawGizmosSelected” para dibujar artilugios cuando se selecciona el objeto del juego.  
  
using UnityEngine;

using System.Collections;

public class ExampleClass : [MonoBehaviour](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html)

{

void OnDrawGizmos()

{

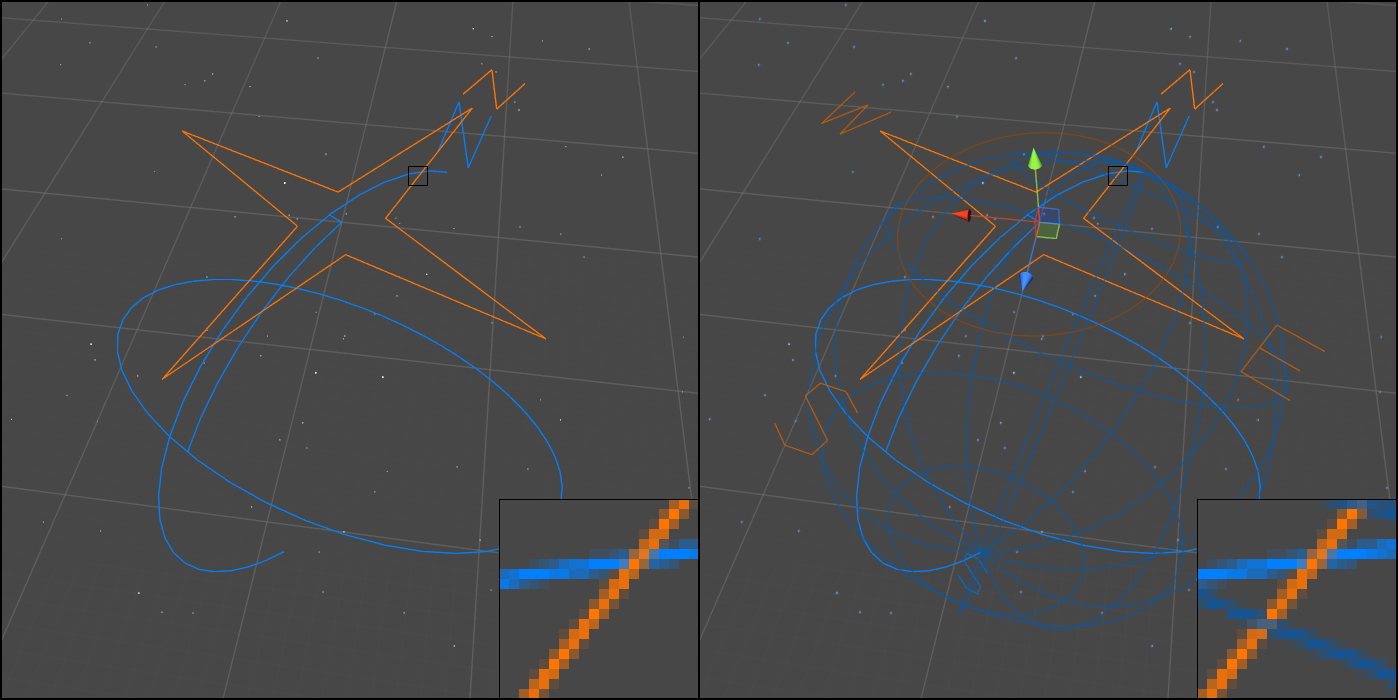
// Draw a yellow sphere at the transform's position

[Gizmos.color](https://docs.unity3d.com/ScriptReference/Gizmos-color.html) = [Color.yellow](https://docs.unity3d.com/ScriptReference/Color-yellow.html);

[Gizmos.DrawSphere](https://docs.unity3d.com/ScriptReference/Gizmos.DrawSphere.html)(transform.position, 1);

}

}



**OnRenderImage:**   
Se llama a “OnRenderImage” después de completar todo el renderizado para representar la imagen.

Efectos de postprocesamiento.

Le permite modificar la imagen final procesandola con filtros basados en sombreadores. La imagen entrante es la textura de renderizado de origen. El resultado debería terminar en la textura de renderizado de destino. Siempre debe emitir un Graphics.Blit () o representar un quad de pantalla completa si anula este método.

Cuando hay varios filtros de imagen conectados a la cámara, procesan la imagen secuencialmente, pasando el destino del primer filtro como fuente al siguiente filtro.

Este mensaje se envía a todos los scripts adjuntos a la cámara.

using UnityEngine;  
  
public class Example : [MonoBehaviour](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html)

{

[Material](https://docs.unity3d.com/ScriptReference/Material.html) material;

void OnRenderImage([RenderTexture](https://docs.unity3d.com/ScriptReference/RenderTexture.html) source, [RenderTexture](https://docs.unity3d.com/ScriptReference/RenderTexture.html) destination)

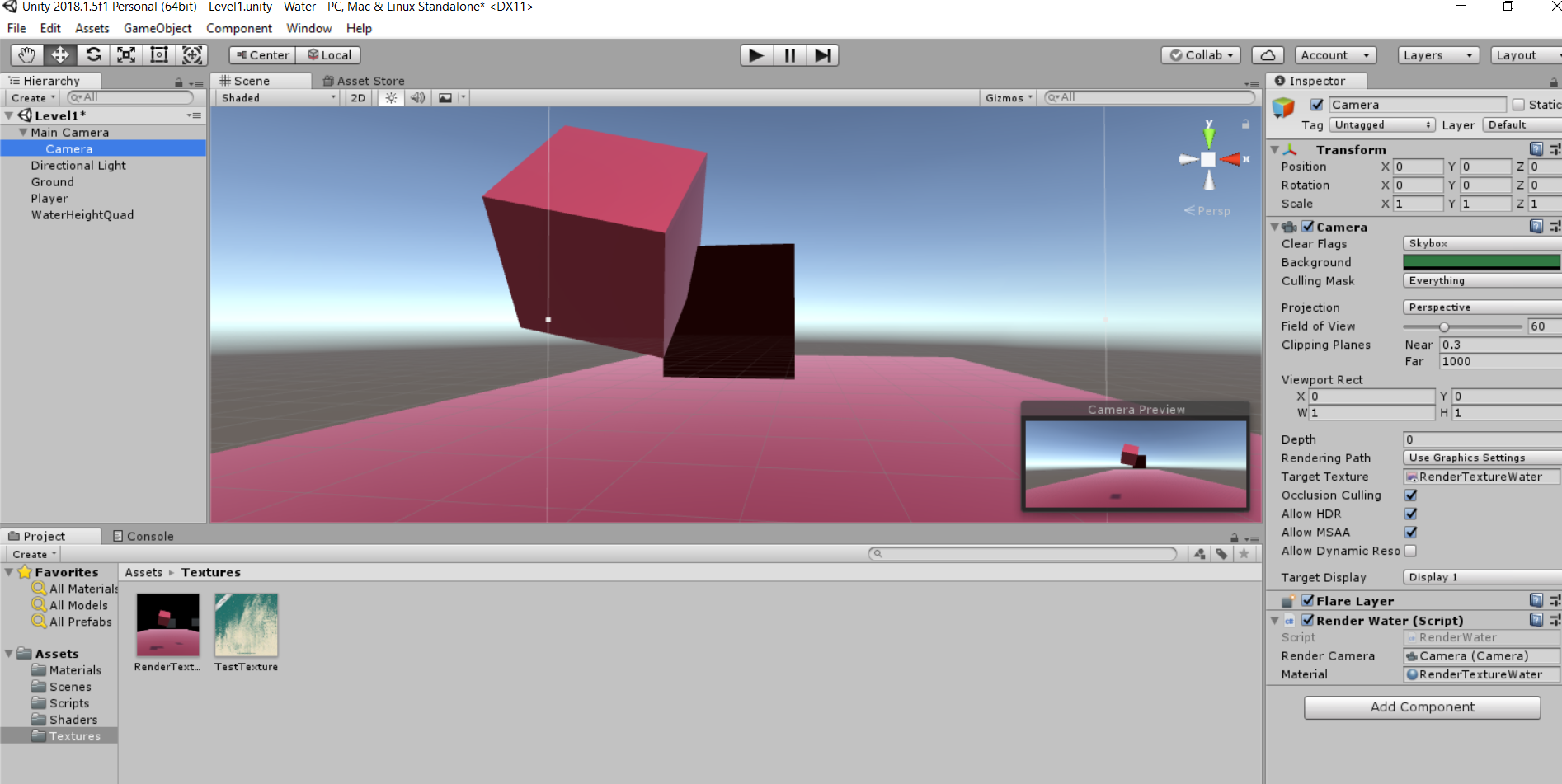
{

// Copy the source Render [Texture](https://docs.unity3d.com/ScriptReference/Texture.html) to the destination,

// applying the material along the way.

[Graphics.Blit](https://docs.unity3d.com/ScriptReference/Graphics.Blit.html)(source, destination, material);

}

}  
  


**When the Object is destroyed**

**OnDestroy:**

Es llamado cuando:

* El objeto va a destruirse.
* La escena actual termina.
* La aplicación del juego se detiene.
* El juego se detiene desde el editor de Unity.

Esta función es llamada después de que todas las actualizaciones de frame para el último frame de la existencia del objeto (el objeto puede ser destruido en respuesta a Object.Destroy o al cerrarse la escena).

OnDestroy ocurre cuando termina una escena o un juego. Al detener el modo Reproducir cuando se ejecuta desde el interior del Editor, finalizará la aplicación. Cuando esto suceda, se ejecutará un OnDestroy. Además, si una escena se cierra y se carga una nueva escena, se realizará la llamada a OnDestroy .

Cuando se construye como una aplicación independiente, [las](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&rurl=translate.google.com.mx&sl=en&sp=nmt4&tl=es&u=https://docs.unity3d.com/2020.1/Documentation/ScriptReference/MonoBehaviour.OnDestroy.html&xid=17259,15700023,15700186,15700190,15700256,15700259,15700262,15700265,15700271&usg=ALkJrhjtd7efa4ldQ4sKTj7HlWt93QKCYg) llamadas de OnDestroy se realizan cuando finalizan las escenas. Una escena finalizada generalmente significa que se carga una nueva escena.

Ejemplo de un script utilizando OnDestroy:

Se ejecuta un botón dentro de ExampleClass1 y este llama a OnDestroy. Luego cambia a ExampleClass2. Una vez que ExampleClass2 esté activo, [OnDestroy](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&rurl=translate.google.com.mx&sl=en&sp=nmt4&tl=es&u=https://docs.unity3d.com/2020.1/Documentation/ScriptReference/MonoBehaviour.OnDestroy.html&xid=17259,15700023,15700186,15700190,15700256,15700259,15700262,15700265,15700271&usg=ALkJrhjtd7efa4ldQ4sKTj7HlWt93QKCYg) se usará cuando se cierre la aplicación. Si ExampleClass1 cierra al cerrar la aplicación, llamará a OnDestroy.

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class **ExampleClass1** : [MonoBehaviour](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/MonoBehaviour.html)

{

private float timePass = 0.0f;

private int updateCount = 0;

void Start()

{

[Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("Start1");

}

// code that generates a message every second

void [Update](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/PlayerLoop.Update.html)()

{

timePass += [Time.deltaTime](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Time-deltaTime.html);

if (timePass > 1.0f)

{

timePass = 0.0f;

[Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("Update1: " + updateCount);

updateCount = updateCount + 1;

}

}

void OnGUI()

{

if ([GUI.Button](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/GUI.Button.html)(new [Rect](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Rect.html)(10, 10, 250, 60), "Change to scene2"))

{

[Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("Exit1");

[SceneManager.LoadScene](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/SceneManagement.SceneManager.LoadScene.html)(1);

}

}

// generate a message before the Start() function

void OnEnable()

{

[Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("OnEnable1");

}

// generate a message when the game shuts down or switches to another [Scene](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/SceneManagement.Scene.html)

// or switched to ExampleClass2

void OnDestroy()

{

[Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("OnDestroy1");

}

}

using UnityEngine;  
using UnityEngine.UI;  
public class **ExampleClass2** : [MonoBehaviour](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/MonoBehaviour.html)  
{  
 void Start()  
 {  
  [Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("Start2");  
 }  
 void OnEnable()  
 {  
  [Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("OnEnable2");  
 }  
 // generate a message when the game shuts down  
 void OnDestroy()  
 {  
  [Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("OnDestroy2");  
 }  
}

**When quitting**

**OnApplicationQuit**: Esta función se llama en todos los game objects antes de que la aplicación se salga. En el editor se llama cuando el usuario detiene el modo de reproducción.

Se envía a todos los objetos del juego antes de que se cierre la aplicación.

Ejemplo:

using UnityEngine;

using System.Collections;

public class ExampleClass : [MonoBehaviour](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/MonoBehaviour.html)

{

void OnApplicationQuit()

{

[Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("[Application](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Application.html) ending after " + [Time.time](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Time-time.html) + " seconds");

}

}

**OnDisable:** Esta función es llamada cuando el comportamiento se vuelve inactivo o deshabilitado.  
  
Esto también se llama cuando el objeto se destruye y se puede usar para cualquier código de limpieza. Cuando los scripts se vuelven a cargar después de que la compilación ha finalizado, se llamará a OnDisable, seguido de un OnEnable después de que se haya cargado el script.  
  
Ejemplo:  
using UnityEngine;  
[[ExecuteInEditMode](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/ExecuteInEditMode.html)]  
  
public class PrintOnOff : [MonoBehaviour](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/MonoBehaviour.html)  
{  
 void OnDisable()  
 {  
  [Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("PrintOnDisable: script was disabled");  
 }

void OnEnable()  
 {  
  [Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("PrintOnEnable: script was enabled");  
 }

void [Update](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/PlayerLoop.Update.html)()  
 {  
#if UNITY\_EDITOR  
  [Debug.Log](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Debug.Log.html)("[Editor](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Editor.html) causes this [Update](https://docs.unity3d.com/2020.1/Documentation/ScriptReference/PlayerLoop.Update.html)");  
#endif  
 }  
}

**OnPreRender:** Este es llamado antes de que la cámara cargue el escenario.  
Ejemplo:

**protected override void OnPreRender(EventArgs e)**

**{**

**base.OnPreRender(e);**

**if (\_message == null)**

**\_message = "Here is some default text.";**

**}**

OnPostObject: Este es llamado cuando la cámara finalmente carga el escenario.  
bool revertFogState = false;

**void OnPreRender()**

**{**

**revertFogState =** [**RenderSettings.fog**](https://docs.unity3d.com/ScriptReference/RenderSettings-fog.html)**;**

[**RenderSettings.fog**](https://docs.unity3d.com/ScriptReference/RenderSettings-fog.html) **= enabled;**

**}**

**void OnPostRender()**

**{**

[**RenderSettings.fog**](https://docs.unity3d.com/ScriptReference/RenderSettings-fog.html) **= revertFogState;**

**}**

**OnPostRender;**se llama después de que una cámara termina de renderizar la escena  
Se llama a esta función sólo si el script está conectado a la cámara y está habilitado. OnPostRender puede ser una rutina conjunta, simplemente use la declaración de rendimiento en la función.

**using UnityEngine;  
// A script that when attached to the camera, makes the resulting  
// colors inverted. See its effect in play mode.  
public class ExampleClass :** [**MonoBehaviour**](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html) **{  
 private** [**Material**](https://docs.unity3d.com/ScriptReference/Material.html) **mat;  
 // Will be called from camera after regular rendering is done.  
 public void OnPostRender()  
 {  
 if (!mat)  
 {  
 // Unity has a built-in shader that is useful for drawing  
 // simple colored things. In this case, we just want to use  
 // a blend mode that inverts destination colors.  
 var shader =** [**Shader.Find**](https://docs.unity3d.com/ScriptReference/Shader.Find.html)**("Hidden/Internal-Colored");  
 mat = new** [**Material**](https://docs.unity3d.com/ScriptReference/Material.html)**(shader);  
 mat.hideFlags =** [**HideFlags.HideAndDontSave**](https://docs.unity3d.com/ScriptReference/HideFlags.HideAndDontSave.html)**;  
 // Set blend mode to invert destination colors.  
 mat.SetInt("\_SrcBlend", (int)UnityEngine.Rendering.BlendMode.OneMinusDstColor);  
 mat.SetInt("\_DstBlend", (int)UnityEngine.Rendering.BlendMode.Zero);  
 // Turn off backface culling, depth writes, depth test.  
 mat.SetInt("\_Cull", (int)UnityEngine.Rendering.CullMode.Off);  
 mat.SetInt("\_ZWrite", 0);  
 mat.SetInt("\_ZTest", (int)UnityEngine.Rendering.CompareFunction.Always);  
 }**  [**GL.PushMatrix**](https://docs.unity3d.com/ScriptReference/GL.PushMatrix.html)**();**  [**GL.LoadOrtho**](https://docs.unity3d.com/ScriptReference/GL.LoadOrtho.html)**();  
 // activate the first shader pass (in this case we know it is the only pass)  
 mat.SetPass(0);  
 // draw a quad over whole screen**  [**GL.Begin**](https://docs.unity3d.com/ScriptReference/GL.Begin.html)**(**[**GL.QUADS**](https://docs.unity3d.com/ScriptReference/GL.QUADS.html)**);**  [**GL.Vertex3**](https://docs.unity3d.com/ScriptReference/GL.Vertex3.html)**(0, 0, 0);**  [**GL.Vertex3**](https://docs.unity3d.com/ScriptReference/GL.Vertex3.html)**(1, 0, 0);**  [**GL.Vertex3**](https://docs.unity3d.com/ScriptReference/GL.Vertex3.html)**(1, 1, 0);**  [**GL.Vertex3**](https://docs.unity3d.com/ScriptReference/GL.Vertex3.html)**(0, 1, 0);**  [**GL.End**](https://docs.unity3d.com/ScriptReference/GL.End.html)**();**  [**GL.PopMatrix**](https://docs.unity3d.com/ScriptReference/GL.PopMatrix.html)**();  
 }  
}**

## **Coroutines**

## Las actualizaciones de rutina normales se ejecutan después de que vuelve la función Actualizar. Una Coroutine es una función que puede suspender su ejecución (rendimiento) hasta que finalice la YieldInstruction dada.

## **Yield**

## La rutina continuará después de que se hayan llamado a todas las funciones de Actualización en el siguiente marco.

Ejemplo:

using UnityEngine;

using System.Collections;

public class WaitForSecondsExample : [MonoBehaviour](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html)

{

void Start()

{

//Start the coroutine we define below named ExampleCoroutine.

StartCoroutine(ExampleCoroutine());

}

IEnumerator ExampleCoroutine()

{

//Print the time of when the function is first called.

[Debug.Log](https://docs.unity3d.com/ScriptReference/Debug.Log.html)("Started [Coroutine](https://docs.unity3d.com/ScriptReference/Coroutine.html) at timestamp : " + [Time.time](https://docs.unity3d.com/ScriptReference/Time-time.html));

//yield on a new [YieldInstruction](https://docs.unity3d.com/ScriptReference/YieldInstruction.html) that waits for 5 seconds.

yield return new [WaitForSeconds](https://docs.unity3d.com/ScriptReference/WaitForSeconds.html)(5);

//After we have waited 5 seconds print the time again.

[Debug.Log](https://docs.unity3d.com/ScriptReference/Debug.Log.html)("Finished [Coroutine](https://docs.unity3d.com/ScriptReference/Coroutine.html) at timestamp : " + [Time.time](https://docs.unity3d.com/ScriptReference/Time-time.html));

}

}

**OnBecameVisible y OnBecameInvisible**

Se llama a **OnBecameVisible** cuando cualquier cámara hace visible el renderizador.

Este mensaje se envía a todos los scripts adjuntos al renderizador. **OnBecameVisible** y **OnBecameInvisible** es útil para evitar cálculos que solo son necesarios cuando el objeto es visible.

Se llama a **OnBecameVisible** una vez para decirle que el objeto, que antes no era visible, ahora ha entrado en el tronco de al menos una cámara. Del mismo modo, **OnBecameInvisible** se llama una vez y le dice que el objeto, aunque anteriormente era visible, ahora ha dejado el centro de todas las cámaras.

**OnBecameVisible**

**// Enables the behaviour when it is visible**

**using UnityEngine;**

**using System.Collections;**

**public class ExampleClass :** [**MonoBehaviour**](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html)

**{**

**void OnBecameVisible()**

**{**

**enabled = true;**

**}**

**}**

OnBecameVisible puede ser una rutina, simplemente use la declaración de rendimiento en la función. Cuando se ejecuta en el editor, las cámaras de vista de escena también harán que se llame a esta función.

**OnBecameInvisible**

**// Disables the behaviour when it is invisible**

**using UnityEngine;**

**using System.Collections;**

**public class ExampleClass :** [**MonoBehaviour**](https://docs.unity3d.com/ScriptReference/MonoBehaviour.html)

**{**

**void OnBecameInvisible()**

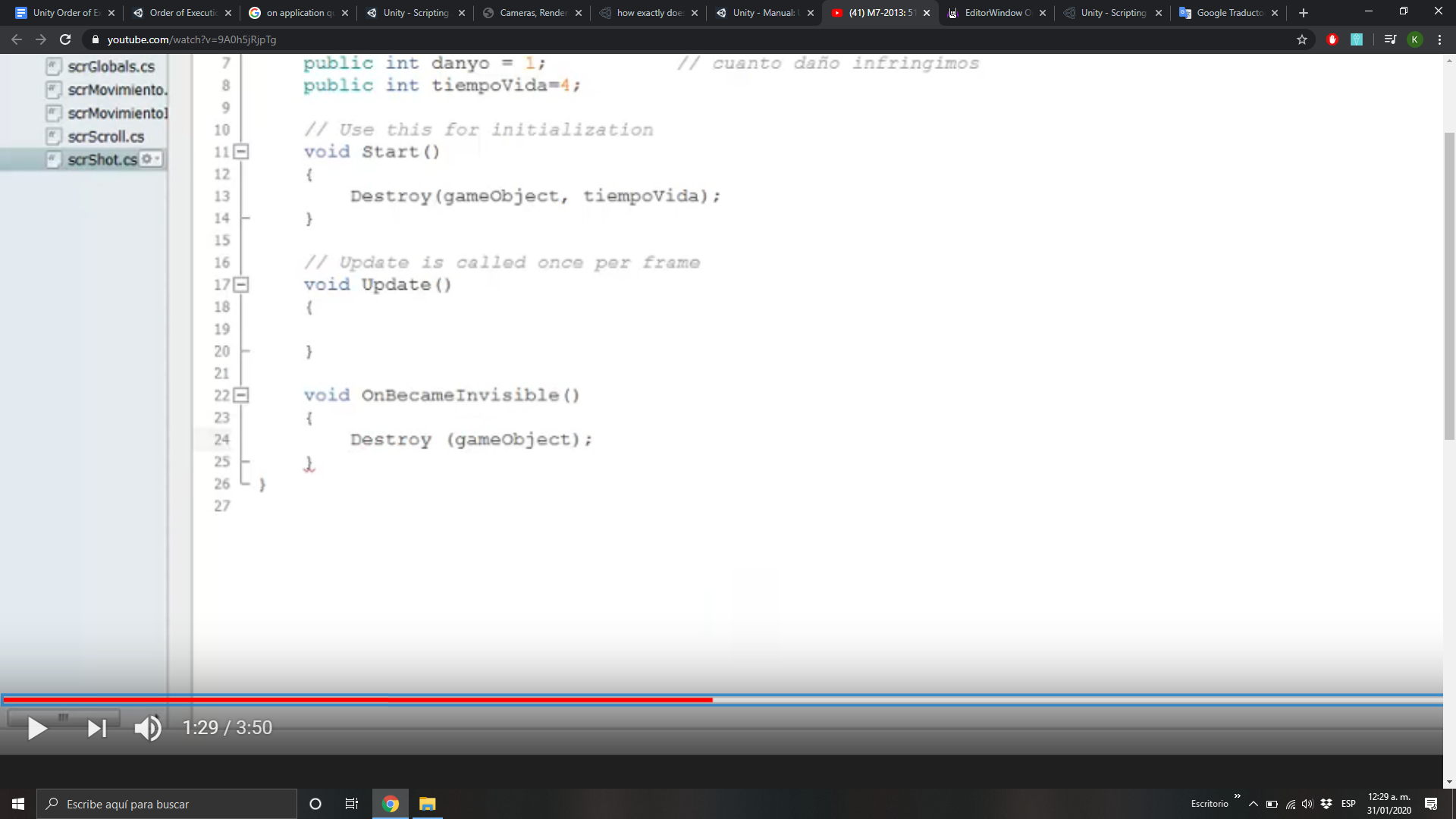
**{**

**enabled = false;**

**}**

**}**

**OnBecameInvisible** puede ser una rutina, simplemente use la declaración de rendimiento en la función. Cuando se ejecuta en el editor, las cámaras de vista de escena también harán que se llame a esta función.

****