

CONTENIDO

Introducción	13
¿Qué es el scripting en Unity?	13
¿Qué lenguajes puedes usar en Unity?	14
¿Qué función cumplen?	15
Variables	16
Funciones	18
Funciones de escritura	19
Clases	21
Visión General de Scripting	23
1. Creando y usando scripts	23
Creando Scripts	24
Anatomía de un archivo Script.	24
Controlando un GameObject	27
2. Variables y el Inspector	28
3. Controlando GameObjects utilizando Componentes	30
Accediendo Componentes	30
Accediendo Otros Objetos	31
4. Event Functions (Funciones de Evento)	35
Eventos de Inicialización	37
Eventos GUI	37
Eventos de Física	38
5. Administrador del Tiempo y Framerate	39
Fixed Timestep (Timestep fijo)	40
Máximo Timestep Permitido	40
Time Scale (Escala del tiempo)	41
Capture Framerate (Capturar el Framerate)	43

6. Creando y destruyendo GameObjects.	46
7. Corrutinas.	48
8. Carpetas Especiales y Orden de Compilación de Script (Script Compilation Order)	52
9. Namespaces	54
10. Atributos	56
11. Execution Order of Event Functions (Orden de Ejecución de Funciones de Evento)	57
Editor	57
First Scene Load (Cuando carga la primera escena)	57
Antes de la actualización del primer frame	57
Entre frames	58
Update Order (Orden de Actualización)	58
Rendering (Renderización)	59
Coroutines (Corrutinas)	60
Cuando el Objeto es Destruído	60
Cuando Salga	60
Diagrama de Flujo del Ciclo de Vida de Script	61
12. Entender la Gestión Automática de Memoria	63
Tipos de valor y tipos de referencia	63
Asignación y recolección de basura	64
Optimización	64
Solicitar una Recolección	68
Pools de Objetos Reutilizables	70
13. Compilación Dependiente a la Plataforma	72
Directivas de #Defines de plataforma	72
Probando código pre-compilado	74

Defines de plataforma personalizada	77
Defines Global personalizadas	78
14. Funciones genéricas	80
15. Restricciones de Scripting	81
Complicación con anticipación	82
16. Serialización de Script	86
¿Cuáles son estas situaciones dónde el serializado se comporta diferente de lo que yo esperaba?	89
17. UnityEvents (Eventos de Unity)	96
Utilizando UnityEvents	97
UnityEvents Genéricos	98
18. ¿Qué es una Null Reference Exception? (Excepción con Referencia Null)	99
Revisiones de Null	99
Bloques Try/Catch	100
19. Importando Clases	102
“Recetas” para usar Vectores	103
1. Entendiendo la Aritmética de Vectores	103
Adición	103
Substracción	104
Multiplicación y División Escalar	105
Producto Punto	105
Producto Cruz	107
2. Dirección y Distancia de Un Objeto a Otro	109
3. Computando un vector de Normal/Perpendicular	110
4. La Cantidad de la magnitud de un Vector que se encuentra en la dirección de otro Vector	113
Herramientas de Scripting	114

1. Consola.....	114
Advertencias API obsoletas y Actualizaciones Automáticas	115
Registro de seguimiento de pila	116
2. MonoDevelop	117
Configurando MonoDevelop	117
Configurando el Depurador	117
Source Level debugging.....	118
3. Log Files (Archivos de Registro).....	122
Editor	122
Webplayer (Reproductor web)	122
Player (Reproductor)	123
iOS	123
Android	124
Tizen.....	124
Windows Store.....	124
WebGL	124
Accediendo Log Files (Archivos de Registro) en Windows.....	124
4. Editor Test Runner.....	126
Comenzando con NUnit.....	126
¿Cómo funciona Editor Tests Runner?	126
Descripción general de la ventana Runner de Tests del editor ..	127
Carreras sin cabeza (modo por lotes).....	128
5. IL2CPP	131
Opciones del compilador.....	131
Construcciones incrementales	134
EventSystem (Sistema de Eventos)	135
1. Descripción General.....	135

2. Input Modules (Módulos de Input)	135
3. Raycasters	136
4. Messaging System (Sistema de Mensajería)	136
¿Cómo Defino un Mensaje Personalizado?	137
5. InputModules (Módulos de Input)	139
6. Eventos Soportados	139
7. Referencia al Event System (Sistema de Eventos)	142
Event System Manager (Administrador Del Event System)	142
Graphic Raycaster	142
Physics Raycaster (Raycaster de física)	143
Standalone Input Module (Modulo Input Standalone)	143
Touch Input Module (Módulo de Input Táctil)	145
Event Trigger	147

Introducción

¿Qué es el scripting en Unity?

El scripting les indica a nuestros GameObjects cómo comportarse; son los scripts y los componentes añadidos a los GameObjects, y cómo interactúan entre sí, lo que crea tu jugabilidad. Ahora bien, el scripting en Unity difiere de la pura programación. Si has hecho algo de programación pura, por ejemplo, creaste una aplicación en ejecución, debes comprender que en Unity no necesitas crear el código que ejecuta la aplicación, ya que Unity lo hace por ti. En lugar de ello, te concentras en la jugabilidad en tus scripts.

Unity ejecuta un bucle enorme. Lee todos los datos que hay en la escena de un juego. Por ejemplo, lee las luces, las mallas, cuáles son los comportamientos, y procesa toda esta información para ti.

Si piensas en la televisión, donde, por ejemplo en Norteamérica, tienes 29,5 frames/segundo, Unity tiene que hacer lo mismo. Se trata de ejecutar frames específicos únicos, uno tras otro. Orientas a Unity con las instrucciones que escribes en tus scripts, y Unity las ejecuta frame tras frame tan rápido como puede.

Lograr una velocidad de frames alta no solo significa que tu juego se verá más fluid, sino que tus scripts se ejecutarán también con más frecuencia, haciendo que los controles sean más receptivos.

¿Qué lenguajes puedes usar en Unity?

Un script debe estar vinculado con un GameObject en la escena para poder ser invocado por Unity. Los scripts se escriben en un lenguaje especial que Unity puede entender. Y, es a través de este lenguaje que podamos hablarle al motor y darle nuestras instrucciones.

El lenguaje que se usa en Unity se denomina C# (se pronuncia con una C aguda). Todos los lenguajes que Unity emplea son lenguajes de scripting orientados a objetos. Al igual que cualquier lenguaje, los lenguajes de scripting tienen sintaxis, o partes de diálogo, y las partes principales se denominan variables, funciones y clases.

Si estás usando una versión de Unity hasta 2017.3, notarás que tiene un editor de texto llamado MonoDevelop: este puede ayudarnos a completar tu código, nos permitirá saber si estás escribiendo un fragmento de código incorrecto, y nos ayuda a tomar atajos. A partir de 2018.1, también puedes usar [Visual Studio for Unity Community](#), u otros editores de texto como Visual Studio, Notepad o Sublime Text.

Este es un script con un ejemplo de código:

```
1 using System.Collections;
2 using UnityEngine;
3
4 public class DemoScript: MonoBehaviour {
5
6     //Variables
7     //Functions
8     //Classes
9
10    //Use this initialization
11
12    void Start () {
13
14    }
15
16
17    //Update is called once per frame
18
19    void Update () {
20
21    }
22
23 }
```

Ilustración 0-1

¿Qué función cumplen?

Las **variables** contienen valores y referencias a objetos (puedes ver los objetos como variables "más grandes"). Son como una caja que contiene algo que nosotros vamos a usar. Las variables comienzan con una minúscula.

Las **funciones** son colecciones de código que comparan y manipulan estas variables. Las funciones comienzan con una mayúscula. Organizamos el código en funciones de manera que estas puedan reutilizarse fácilmente muchas veces en diferentes partes del programa.

Las **clases** son una forma de estructurar el código para empaquetar colecciones de variables y funciones a fin de crear una plantilla que define las propiedades de un objeto.

El scripting consiste principalmente en comparar estos objetos y sus estados y valores actuales. Se basa en la determinación lógica de un resultado o resolución.

Variables

En Unity, los scripts comienzan por disponer las herramientas que necesitas arriba, y esto normalmente se hace declarando variables. Puedes ver las variables declaradas con la palabra clave visibilidad "pública" o "privada" en el frente, seguida por un tipo y un nombre.

```
5 public class DemoScript: MonoBehaviour {  
6  
7     public Light myLight;  
8     private Light myOtherLight;  
9  
10 }
```

Ilustración 0-1

Cuando declaras tus variables, hay varios tipos de visibilidad, pero los dos más importantes son pública y privada.

Si creas un script con el texto anterior en tu editor de códigos y después regresas a Unity y asignas el script a un `GameObject`, verás que puedes acceder y ver la variable ligera declarada como pública en el Inspector, pero no puedes ver la privada. Y esto ocurre porque solo puede accederse a lo que se define como "privado" dentro de este script particular, dentro de esta clase particular.

Si la haces pública, entonces otros scripts y otras clases podrán acceder, y podrá cambiarse en el Inspector desde el editor de Unity. Por tanto, esto significa que otras personas pueden tener acceso y cambiar su valor.

Hay muchas razones para elegir entre privado o público. Las variables privadas te permiten tener códigos más limpios, ya que

sabes que el valor de esas variables puede ser cambiado solo desde dentro de esa clase. Esto hace que la depuración y el mantenimiento del código sean más sencillos.

Si eliges "público", y experimentas un problema, tendrás que buscar en toda la base de códigos para poder rastrear la fuente debido a cualquier otro objeto tiene acceso a esa variable. Sin embargo, si quieres que los objetos se comuniquen entre ellos, necesitarás que algunas variables (o funciones) sean públicas.

Otro aspecto importante de las variables es el tipo. Un tipo define qué tipo de valor mantiene la variable en la memoria, por ej., puede ser un número, texto, o tipos más complejos, como los que se observan en la imagen a continuación: Transform, Light y Demo Script son, en efecto, referencias a Componentes. Unity necesita saber qué tipo de objeto es y cómo manejarlo.

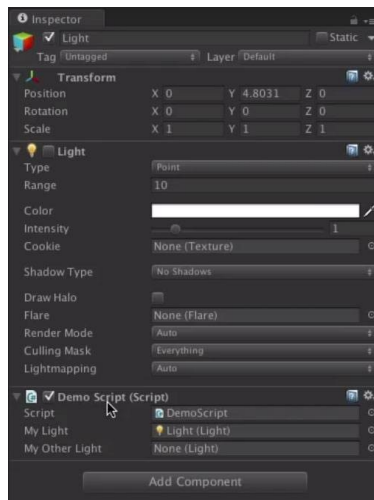


Ilustración 0-II

Otra cosa importante en relación con las variables es el nombre. Lo principal es que tienes que recordar al asignar un nombre a las variables es que no pueden comenzar por un número, y que no pueden contener espacios. Por lo tanto, hay un estilo para

escribir los nombres. En C#, la convención de nomenclatura es camelCase: comienzas con una minúscula y añades palabras, sin espacios, comenzando por una mayúscula, por ej.: "miLuz".

Cuando Unity compila el script, hace que las variables públicas sean visibles en el editor. Observa la imagen a continuación del inspector.

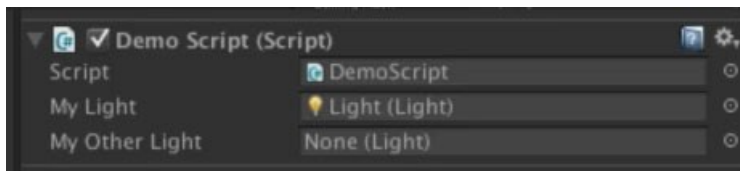


Ilustración 0-III

Funciones

Los scripts manipulan las variables a través de funciones. Hay una serie de funciones que se ejecutan automáticamente en Unity. Observa a continuación:

```

54 /*
55     Awake()
56     Start()
57     Update ()
58     FixedUpdate()
59     LateUpdate
60
61 */
    
```

Ilustración 0-I

Awake se invoca solo una vez cuando se menciona ese componente. Si un GameObject está inactivo, entonces no podrá invocarse hasta que se le active. Sin embargo, Awake se

invoca incluso si el `GameObject` está activo pero el componente no está habilitado (con la pequeña casilla de verificación junto a su nombre). Puedes usar `Awake` para inicializar todas las variables a las que necesitas asignar un valor.

Start - al igual que `Awake`, `Start` se invocará si un `GameObject` está activo, pero solo si el componente está habilitado. Para más información sobre las diferencias con `Awake`.

Update se invoca una vez por frame. Aquí es donde colocas el código para definir la lógica que se ejecuta continuamente, como animaciones, `AI` y otras partes del juego que tienen que actualizarse continuamente.

FixedUpdate es cuando quieres que la física funcione.

Como puedes ver, hay una `Fixed Update` (Actualización fija) y una `Update` (Actualización).

LateUpdate es una función que es similar a `Update`, pero `LateUpdate` se invoca al final del frame. Unity analizará todos los objetos de juego, encontrará todas las `Updates`, e invocará las `LateUpdates`. Esto es bueno para cosas como la cámara. Digamos que quieres mover un personaje en tu juego. Y entonces, él tropieza con otro personaje y termina en otra posición. Si movemos la cámara al mismo tiempo que el personaje, habrá una sacudida, y la cámara no va a estar donde tendría que estar. Así que, básicamente, es un segundo loop que resulta muy práctico.

Funciones de escritura

Cuando escribas una función, recuerda que las funciones comienzan con el tipo de función al principio, seguido por el nombre de la función, y después los parámetros entre paréntesis (si fuese el caso). Los nombres de función comienzan por una

mayúscula y el cuerpo de la función va entre llaves. Este es un ejemplo de cómo escribir una función:

```
13
14 void MyFunction () {
15
16 }
17 }
```

Ilustración 0-I

¿Cómo llamamos a esta función?

```
19
20 void MyFunction () {
21     myLight.enabled = !myLight.enabled;
22 }
23 }
```

Ilustración 0-II

Las funciones hacen cálculos y después presentan un valor. Puedes pedir a una función que haga algo, que procese la información, y que después te dé una respuesta. Si usas el tipo "void", entonces no obtendrás ninguna respuesta.