UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Realidad Virtual y Aumentada

Programación e interacción. Lenguaje C# en Unity 3D, control de elementos multimedia y 3D.

Autor: Javier Córdoba Romero y Juan José Corroto Martin

Profesor: Javier Alonso Albusac Jiménez

Diciembre, 2019

Índice

1. C#

```
public class Cosa
{
        public static void main(String[] args)
        {
                Console.WriteLine("Hello World");
        }
}
```

Listado 1: Hello world

Citacion: [1].

1.1. Programación basada en eventos

En algún momento he dicho que los scripts son componentes En algún momento deberías decir que las variables públicas se pueden ver y modificar desde el editor. En algún momento deberías decir que es posible que los componentes tengan componentes hijos. En algún momento deberías hablar de los eventos más importantes de los comportamientos.

2. Uso de C# en Unity

La primera versión del motor de juegos Unity fue lanzada en el 2005, desde entonces mucho ha cambiado en el motor, pasando por el motor de scripting, nuevas tecnologías gráficas y el modelo de licencias.

El lenguaje de scripting de Unity pasó por varias etapas, la primera de ellas fue *Boo*, un lenguaje orientado a objetos usando el *Common Language Runtime* (CLR) de .NET Framework, tecnología similar a la usada por *Java* en su máquina virtual. La siguiente etapa pasó por usar una variante de *Javascript* como lenguaje de scripting y, por último, se pasó a usar C# como lenguaje de scripting con *Visual Studio* como uno de los IDEs soportados.

C# es un lenguaje de programación orientado a objetos y ejecutado sobre el *Common Language Runtime*, usando principalmente con tipos estáticos aunque con soporte para tipos dinámicos, también se puede enfocar a una programación basada en eventos gracias a sus *delegados*.

Ha conseguido obtener la condición de estándar ISO, su últimya revisión fue en 2018, con

referencia: ISO 23270:2018[1]

Actualmente Unity soporta más de 20 plataformas[2], entre las que más destacan podemos encontrar: Windows, Linux, Mac, Playstation 4, Xbox One, Nintendo 3Ds, Oculus Rift, Android, iOS, WebGL, ARKit y ARCore.

C# en Unity también permite interaccionar con los diferentes componentes del motor, como por ejemplo *Mecanim*, su sistema de animación, su sistema de físicas o su sistema de componentes, lo que proporciona un control total sobre el motor y no sólo tareas de scripting in-game.

2.1. Estructura de clases de Unity

2.2. Manejo de objetos de Unity desde C#

En esta sección se va a tratar cómo manipular objetos del mundo virtual desde un *script* en *Unity*. Como ya se ha dicho, los *scripts* son componentes de los objetos. Desde cualquier *script* podemos manipular cualquier otro objeto del mundo, o el mismo objeto. Esto último es especialmente fácil si queremos aplicar una transformación, pues tenemos acceso al componente *Transform* como variable de clase. En 2 podemos ver como se puede acceder al componente *transform* del objeto e invocar sus funciones para aplicarle una traslación. La variable *transform* de ese listado no es necesario que se defina en ningún sitio del *script*, sino que se tiene acceso a ella directamente. Esto es porque el componente *Transform* es inherente a todos los objetos que están en la escena, puesto que todos los objetos deben de tener una transformación para saber dónde hay que dibujarlos. El resto de componentes (que deben ser añadidos manualmente), pueden ser accedidos simplemente con la función *GetComponent<Tipo>()*. De esta forma podemos fácilmente aplicar transformaciones o fuerzas (en el caso de rigidBody) sobre el mismo objeto.

```
Vector3 direction = new Vector3(mainCamera.transform.forward.x, 0,
   mainCamera.transform.forward.z).normalized * speed *
   Time.deltaTime;
Quaternion rotation = Quaternion.Euler(new Vector3(0,
   -transform.rotation.eulerAngles.y, 0));
transform.Translate(rotation * direction);
```

Listado 2: Acceso al componente transform para modificar el propio objeto

[1] https://www.iso.org/standard/75178.html
[2] https://unity3d.com/es/unity/features/multiplatform

En el caso de querer modificar otro objeto, por ejemplo, en el caso de querer crear un enemigo a cierta distancia de otro, simplemente tenemos que encontrar la referencia al objeto. Esto se puede hacer muy fácilmente haciendo una variable pública y asignándola desde el editor. Sin embargo, si queremos acceder al objeto de forma dinámica o únicamente usando *C#*, se puede hacer de varias formas, todas usando funciones estáticas de la clase *GameObject*:

1. Por nombre: usando la función *GameObject.Find()*. Se puede buscar de forma directa un objeto en la escena virtual mediante su nombre, simplemente es necesario conocerlo de antemano. Este último matiz puede que sea la mayor inconveniencia, pues es probable que no sepamos de antemano el nombre del objeto, sobre todo si se ha generado dinámicamente. Esta función tratará los caracteres ¨ïo como parte del nombre, sino como parte de una jerarquía de objetos. La documentación oficial de *Unity* [3] desaconseja usar esta función cada frame.

2. Por tipo: usando la función *GameObject.FindObjectOfType(Type type)*. En este caso, buscamos por tipo de objeto, teniendo un ejemplo en el listado 3. Esta función retorna el **primer** objeto cargado de el tipo especificado, o *null* si no existe ninguno de dicho tipo. Para conseguir un iterador de todos los objetos de ese tipo, también existe la función *GameObject.FindObjectsOfType(Type type)*. La documentación oficial desaconseja utilizar estas funciones por ser más lentas que el resto.

3. Por etiqueta o *tag*: usando la función *GameObject.FindGameObjectWithTag(String tag)*. Esta función, al igual que la anterior, devuelve el **primer** objeto cargado con dicha tag, o *null* si no encuentra ninguno. Además, lanzará una excepción si la tag no existe. Esta función es muy fácil de usar, pues *Unity* tiene un sistema nativo de tags para los objetos muy simple de usar, a parte de tener tags predefinidas. El hecho de que sean simples cadenas de caracteres también lo hace más sencillo de utilizar.

```
using UnityEngine;
using System.Collections;

// Search for any object of Type GUITexture,
// if found print its name, else print a message
// that says that it was not found.
```

[3] https://docs.unity3d.com/ScriptReference/GameObject.html

```
public class ExampleClass : MonoBehaviour
{
    void Start()
    {
        GUITexture texture =
            (GUITexture)FindObjectOfType(typeof(GUITexture));
        if (texture)
            Debug.Log("GUITexture object found: " + texture.name);
        else
            Debug.Log("No GUITexture object could be found");
    }
}
```

Listado 3: Búsqueda de objetos por tipo

3. Ejemplos

4. Conclusion

5. Referencias

Referencias

[1] M. Bowman y L.L. Debray S.K.and Peterson. “Reasoning about naming systems”. En: *ACM Trans. Program. Lang. Syst.* 15.5 (nov. de 1993), págs. 795-825.