Universidad de La Laguna

Escuela Técnica Superior de Ingeniería Informática

Interfaces Inteligentes
Introducción

Práctica 3

Isabel Sánchez Berriel

4º curso del Grado en Ingeniería Informática. Itinerario de Computación

Universidad de La Laguna

Escuela Técnica Superior de Ingeniería Informática

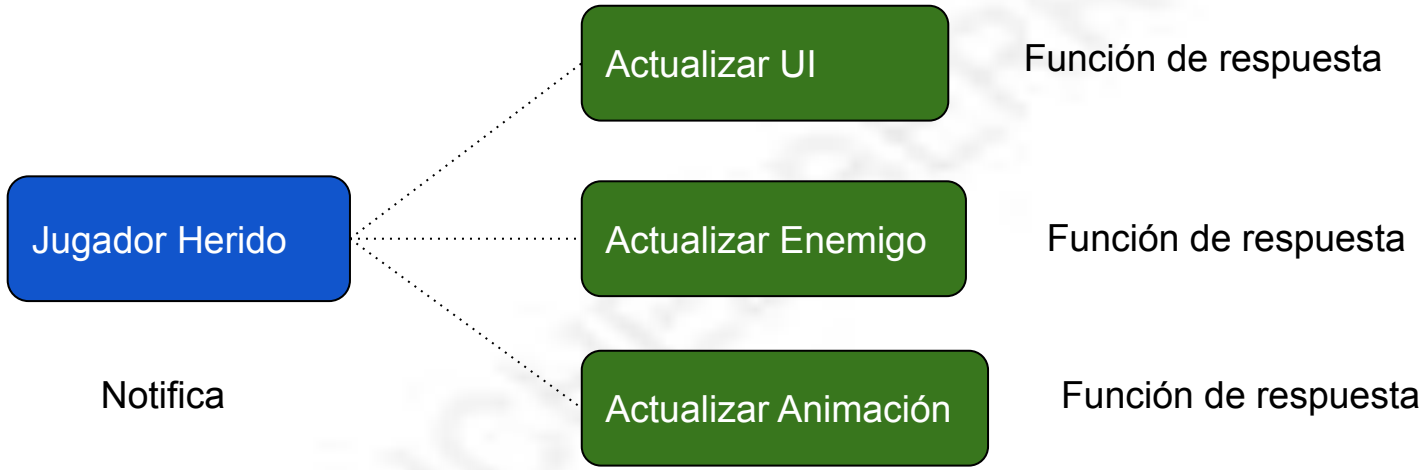
Contenidos

- Patrón Observador
  - *Delegados*
  - *Eventos*
- Patrón Singleton

Patrón Observador

- Si un objeto del juego cambia de estado, notifica a todos los que dependen de él
- Los objetos dependiente (**suscriptores**) se suscriben al objeto al que están observando (**notificador**)
- Los observadores, en Unity, son otros scripts.
- Un observador implementa f**unciones de respuesta** que se ejecutan cuando se le notifica el evento.

Patrón Observador



Cada script responde cuando sucede el evento, en lugar de verificar constantemente si ha sucedido algo

Patrón Observador

- El notificador no necesita saber de los observadores
- Los observadores no necesitan saber de los otros observadores
- El notificador gestiona los estados y cuando se produce el evento envía el mensaje.
- Los observadores sólo tienen que saber qué hacer cuando les llega la notificación.

Tipo delegado

- Tipo delegado de C#

```
public delegate Tipo NombreDelegado(Parámetros);
public NombreDelegado varDelegado;
…
varDelegado = metodo(); //puede ser cualquier función
                        //compatible: devuelve el mismo tipo
                       //mismos parámetros.
```

- Se comporta como un puntero a función.
- Se delega en un método que a priori se desconoce.
- Se debe disponer de un script que actúe de controlador evitando bloques switch.
  - *Se comunican los objetos mediante eventos (delegados)*

Eventos

- Un evento permite utilizar varios delegados.

```
public delegate Tipo NombreDelegado(Parámetros);
public static event NombreDelegado varDelegado;
…
varDelegado = metodo();
```

- Un evento es de tipo delegado.
- Se usan para comunicar a los objetos de la escena algún cambio.
- Cualquier objeto se puede declarar a la escucha del evento.
- Si se elimina el objeto se deben eliminar las suscripciones al evento.

Eventos

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Notificador : MonoBehaviour
{

    public delegate void mensaje();
    public event mensaje OnMiEvento;
    public int contador;
```

Eventos

```
// Start is called before the first frame
update
    void Start()
    {
        contador = 0;
    }
 // Update is called once per frame
    void Update()
    {
        contador = contador + 1;
        if (contador % 1000 == 0) {
            OnMiEvento();
        }
    }
}
```

Eventos

```
public class Respuesta : MonoBehaviour
{
    public Notificador notificador;
    // Start is called before the first frame update
    void Start()
    {
        notificador.OnMiEvento += miRespuesta;
    }
    // Update is called once per frame
    void Update()
    {
     }
    void miRespuesta(){
        Debug.Log("Soy el cilindro");
        Debug.Log(notificador.contador);
    }
}
```

UI

- Sistema de eventos que permite gestionar las respuestas a las entradas desde los dispositivos más comunes.
- El sistema consta de:
  - **Canvas** que actúa de contenedor donde se dibujan los elementos de la UI
  - Elementos que reciben la acción: botones, texto, check box, ...
  - **Input Module**. se especifican e identifican las entradas disponibles.
  - **EventSystem**: component
  - **Raycaster**: component, su función es determinar los objetos que generan el evento.
  - **Input Handlers**