UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

Scripts and basic methods

Jordi Linares

A Unity template script

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
   // Use this for initialization
   void Start () {
   }

   // Update is called once per frame
   void Update ()  {
   }
}
```

- In **UnityEngine** we have the **MonoBehaviour** class, basic when defining scripts that will be assigned to **GameObjects** or **Prefabs**
- **System.Collections** belongs to **.NET** and allows us to define lists, dynamic arrays, dictionaries etc.

Developing scripts in C#

- When developing interactive apps and games in **Unity**, not all the **C#** features are suitable:
  - Performance problems
  - Integration problems with the editor
  - Some particularities when deriving from **MonoBehaviour**
  - Some features not still available in **Unity**
  - etc.
- However, most of the features of the language are available in **Unity** and even the most advanced ones can be very useful in many contexts (delegates and events, lambda expressions, linq, extension methods etc.)
- In any case, we have to be careful with what we put inside in methods such as **Update(), FixedUpdate(), LateUpdate()** etc. since they are executed very frequently => we can have performance problems and get a low framerate

Developing scripts in C#

- Scripts in **Unity** derive form the **MonoBehaviour** class
- It is important to take into account that the script is a component of a GameObject and, therefore, **this** in this context makes reference to the **script** and not to the **GameObject**
- We can assign as many scripts as we want to a GameObject
- The most important methods we can define inside a script are:
  - **Awake(), Start()**
  - **Update(), FixedUpdate(), LateUpdate()**
  - **OnCollisionEnter(), OnCollisionStay(), OnCollisionExit() and their equivalents OnTrigger()**
  - **OnBecameVisible(), OnBecameInvisible()**
  - **OnMouseDown(), OnMouseOver()**
  - **OnEnable(), OnDisable()** …

Awake() and Start()

```
void Awake() {
...
}
```

- Awake() is the first method that it is executed when the object is initialized
- IT MUST be used instead of traditional constructors
- It is always executed before any Start() of any object
- It is perfect for initializing cross references among objects

```
void Start() {
...
}
```

- The method Start() is a second initialization step (after all Awake() of all the objects) and all the Start() methods will be executed before any Update() method of any object
- Using Awake() and Start() we can initialize all our objects in two steps
- Start() is called only if the object is active (enabled, a checkbox we can see in the Inspector)

Update(), FixedUpdate(), LateUpdate()

- One of the main objectives of a script is to update somehow the GameObject to which it is assigned
- These changes or updates are generally a change in the GameObject position, scale, rotation, applying new forces, changing colors, new states etc.

- In order to carry out these tasks, **Unity** gives us several alternatives: **Update(), FixedUpdate(), LateUpdate()**

```
void Update() {
...
}
```

- **Update** is the most common one and its main characteristic is that it is called every frame
- It is really important that the code we put in the Update must be very efficient, since the FPS will be affected
- Perfect also for managing the interaction with the user
- It will be only executed if the object is active (enabled)

Update(), FixedUpdate(), LateUpdate()

```
void FixedUpdate() {
...
}
```

- **FixedUpdate** will be called on regular basis and all the physics updates (a physics engine step) will be carried out after the FixedUpdate

- Therefore, it is the perfect place for everything related with changes in the Rigidbody and constant forces
- FixedUpdate can be invoked more or less frequently than the Update, since the Update is executed on a variable rate, while the FixedUpdate is executed on a constant frequency

Update(), FixedUpdate(), LateUpdate()

```
void LateUpdate() {
...
}
```

- The **LateUpdate** will be invoked when all Updates of all the objects have been executed
- So, it is a new opportunity to update the position of the objects after they have carried out their changes in their Updates
- As the Update, it is executed in every frame and it is used when, for example, an object must follow another one. When the main object moves in its Update, the object that follows can do that in the LateUpdate. As LateUpdates are executed when all Updates have been already executed, we are sure that the update of the main object has been executed before the object that pursues the first (in this way, we don't have to be worried about the order on which the Updates are executed)

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

Scripts and basic methods

Jordi Linares Pellicer

unity