Virtual and Augmented Reality

Introduction to Unity

Jordi Linares Pellicer

Juan Jesús Izquierdo Doménech

Jorge Orta López

Index

Introduction to Unity

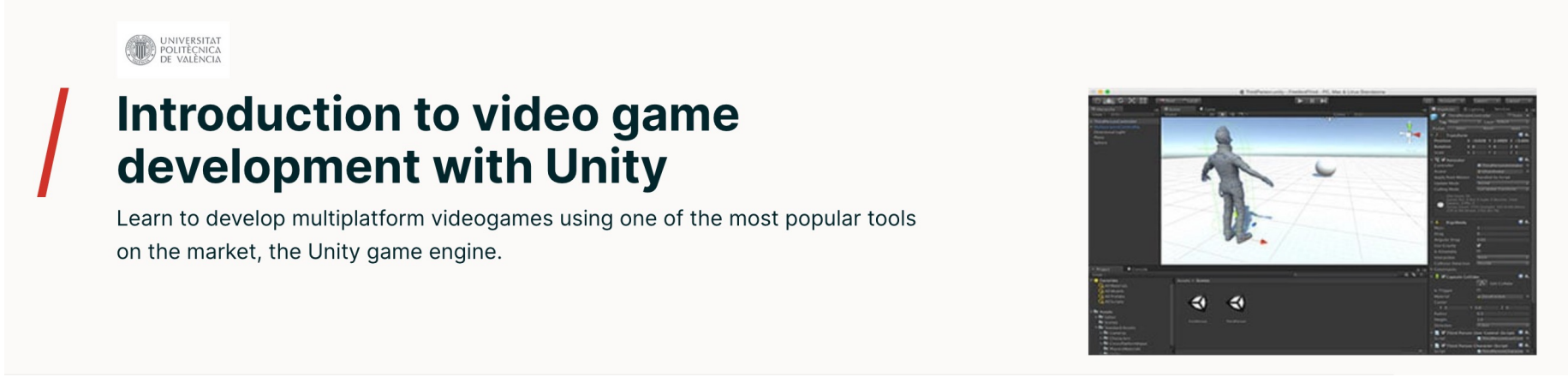
Objectives:

- Learn the basics of Unity
- Focus on the most important parts to develop simple examples
- Learn some basics of its interface design tools
- Learn to integrate Vuforia and its basic operation
- Learn how to develop VR apps with Unity and OpenXR
- Carry out practical examples
- We will use the C # programming language that is easy to learn from C ++ or Java

Introduction to Unity

To deepen:

- My MOOC in edX
- https://www.edx.org/es/course/introduction-to-video-game-development-with-unity



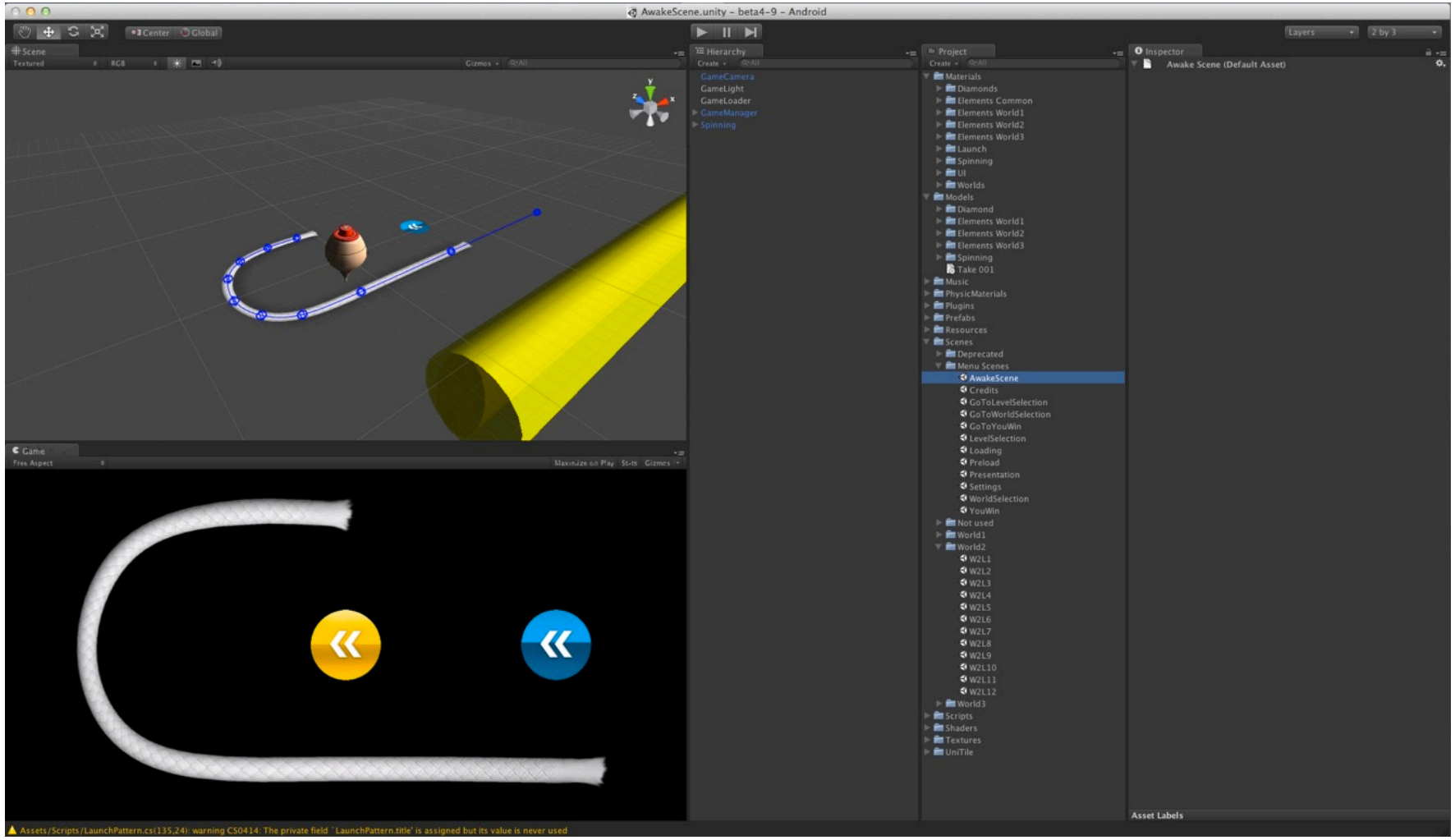
Introduction to Unity

Unity in VR / AR / XR:

- Unity is a video game engine, and video game development has exactly the same needs and characteristics that we need for the development of interactive applications and VR / AR. In particular:
  - High-quality, optimized real-time graphics
  - Deployment on a wide range of platforms
  - Compatibility with different interactive devices
  - Physics engine
  - Development of 2D / 3D graphical user interfaces
  - Multi-user application development
  - Powerful scene and environment editor
  - Powerful development tool, programming language and libraries
  - Support of the main content creation tools 2D, 3D, audio, animations, etc.
  - And more

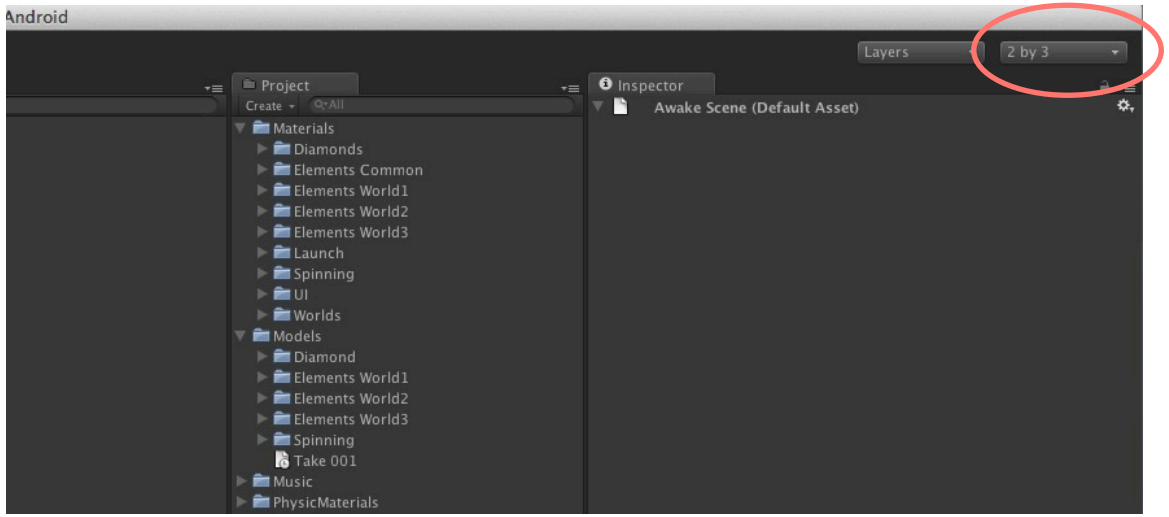
The interface of Unity

Divided in Views



The interface of Unity

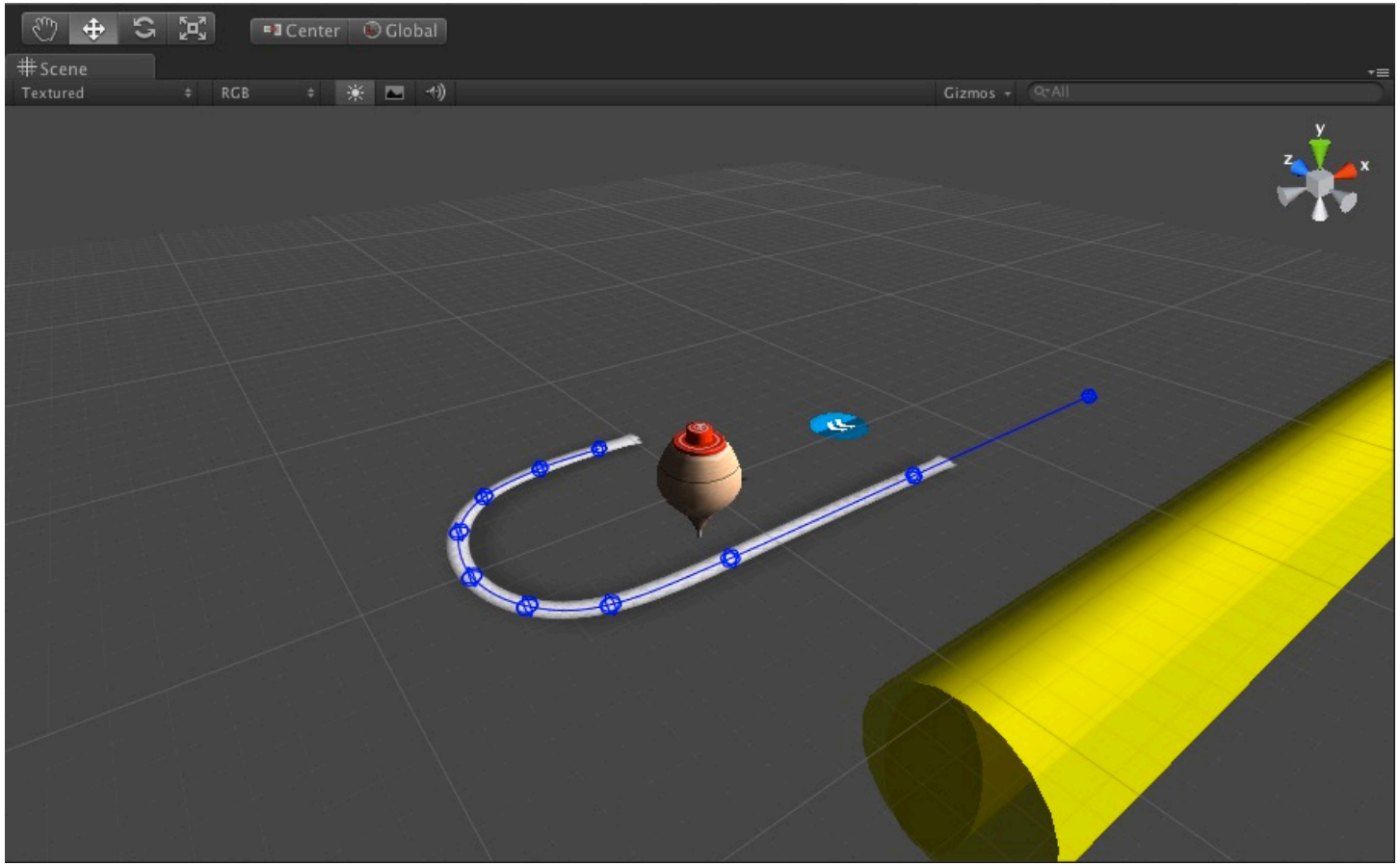
- Any view can be activated or deactivated from the menu Windows
- They can be relocated and even be independent windows
- There are preset configurations and we can create our own:



The interface of Unity

Scene View

- Where we will distribute, we will locate, and we will modify our objects (**GameObjects**)



The interface of Unity

Scene View

- Where we will distribute, locate and modify our objects (**GameObjects**)

Object manipulation tools (keys **q, w, r, t**)

- The first is the 'Hand tool'
- The other 3 tools allow the manipulation of objects: **move** (key '**w**'), **rotate** ('**e**'), **scale** ('**r**'), scale interface elements, and sprites ('t')
- For its operation, an object must be selected, and the corresponding '**Gizmo**' will appear
- The Gizmo of the tool will allow us to control the position, scaling or rotation of the object globally or on a particular axis
- The manipulation tools act on a component of the **GameObject** called **Transform** and that stores the **position**, **scale** and **rotation** of the object

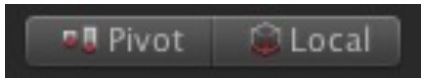
Proposed exercise: ccreate a **GameObject** (sphere, cube ...) and modify its position, scale and rotation, becoming familiar with its possibilities and observing in the view '**Inspector**' the component values '**Transform**'. See what happens when it's done click in the center of Gizmo.

*Experience the effect of pressing the space bar + Shift, and selecting an object and the key **'F'** (click-double over the object in **Hierarchy**)*

The interface of Unity

Scene View

Position control tools



- **Pivot / Center** allow changes to be made with respect to the pivot of the object or its center. They are also especially interesting when we have several objects selected.'Pivot' will cause changes to be made regarding the 'pivot' of one of them, 'Center' will be with respect to the center of all those selected
- **Local / Global** They will be especially useful when we apply rotation to an object. 'Local' will make future changes take into account this previous rotation (the transformation will be done with respect to the local coordinate system of the object), while 'Global' will make the transformations always take place with respect to the global coordinate system of the scene.
- Multiple selection with Shift

Proposed exercise: *Create multiple GameObject, select them and experience the differences of Pivot / Center. With the same example, apply rotation to an object and change its position experiencing the differences of Local / Global*

The interface of Unity

Scene View

- Scene navigation may change based on operating system and available mouse (2 or 3 buttons)

- The tool 'hand tool' will change based on the different possibilities: drag (hand), orbital (eye), magnifying glass (zoom)
- Ctrl / Cmd or mouse wheel-> zoom mode
- Alt -> orbital mode
- The mouse wheel can also be used for zooming (Alt + right button)
- Cursor arrows -> mode 'walking'
- (Right button + WASD + QE) -> Model 'flythrough'
- IT IS GENERALLY NOT necessary to select the tool 'hand tool' to interact with the scene, as these keys are still active in the other modes.

Exercise: Create various objects, manipulate them and navigate the scene using all the possibilities of the tool 'hand tool'

The interface of Unity

Scene View

- **Snapping** (Edit -> Snap Settings)
Allows you to set 'change values' for translations, rotations, and scaling.
For the object to undergo a change with 'snap', you have to press the key Ctrl/Cmd(Mac)

On 'Snap Settings' you can change the properties and also force a specific selected object to adjust to the increments of snap.

- **Vertex Snapping**
With 'translate' and pressing the key 'V' we can select a vertex and snap it to another vertex of another object. This option is especially useful in the creation of levels when we want to make different sections fit together perfectly (without overlaps or gaps)

In general, and more specifically in the level creation, the tools of Snapping and Vertex Snapping are essential.

The interface of Unity

Scene View

- When we want to visualize the scene from a certain axis (very useful in the precise location of objects in the scene), we can use the following gizmo:

- With it, we can choose the axis in question or return to the perspective projection by clicking on the text that appears under the Gizmo.
- It also allows the use of isometric projection

The interface of Unity

Scene View

- In the toolbar of the scene view we find the following:

- The first menu allows us to establish the display mode of the elements of the scene (not on the game, only in this view). The most important are the modes 'shaded' (with texture), Wireframe (wired mode) and ShadedWireframe (mixed).
- The others are of an advanced nature, Render Paths (kind of rendering of objects, green for deferred lighting, yellow for forward rendering and red for vertex lit), the channel alpha (Alpha), the areas of greatest graphic load (Overdraw, where the objects are render with transparency to appreciate accumulation) and Mipmaps can guide us on the appropriate size of the textures of the objects (red indicates that the texture is larger than necessary, blue indicates that the texture should be larger).
- The next button allows us to see the scene in 2D or 3D mode.
- The following icon allows us to visualize the scene with the lights actually assigned or with an ambient light by default (and that would not be applied in the final game)
- The audio button allows us or not to reproduce the sounds in the scene view
- Gizmos allows us to establish their properties
- Finally, a search area to identify elements of the scene

The interface of Unity

Game View

- The objective is to show what you will see in the game from their cameras
- We must provide at least one camera so that the scene can visualize elements

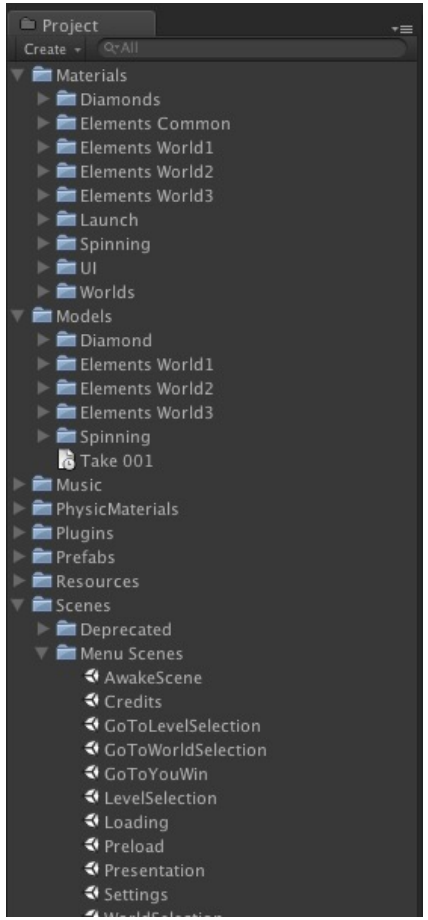
- With these buttons we will start the game, pause it or execute it step by step.
- IT IS IMPORTANT note that Unity allows us to make modifications to the elements of the scene while we are running the game, BUT these changes will be undone when the game is stopped (useful behavior during development but dangerous many times)

- In the scene view options we can change the aspect ratio and its resolution, enable it to be maximized every time we run and show statistics regarding the resources consumed in its execution
- Aspect ratio changes can be very interesting to evaluate the behavior of our game on different devices

The interface of Unity

Project view

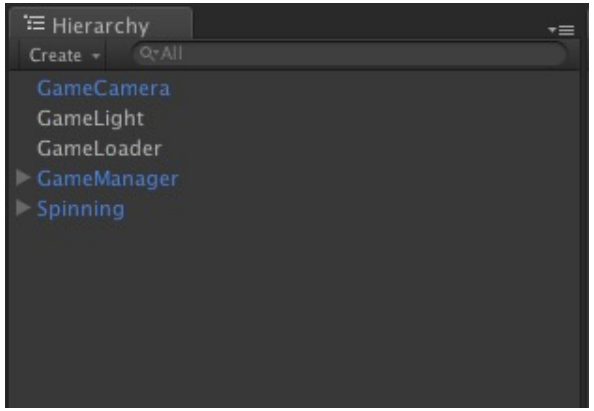
- Folders where we will group our resources (Assets)
- Our project is actually a directory, and 'Project' shows what we can find in the directory 'Assets'
- Many operations can be done alternatively on this view or the directory directly (although it is always better to do it on this view, especially when it comes to moving or deleting things)
- Allows two types of views: one column or two columns (with preview of the assets)



The interface of Unity

Hierarchy view

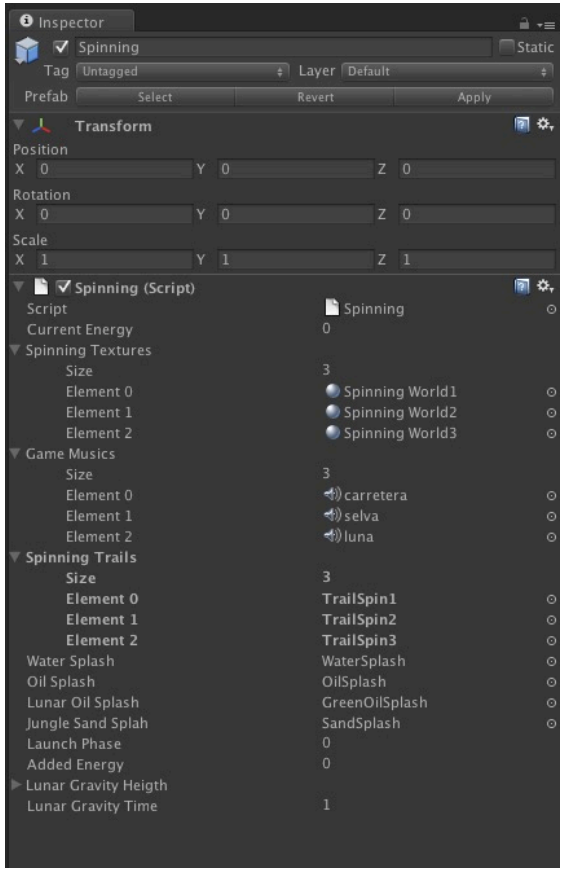
- We can see the 'scene graph' of the current game scene
- Shows the elements of the current scene and their parent / child relationships
- Unity uses the very common scene graph strategy to organize, classify and manipulate the different elements of the scene
- There is a direct correspondence between what we can see in this view and the scene view



The interface of Unity

Inspector view

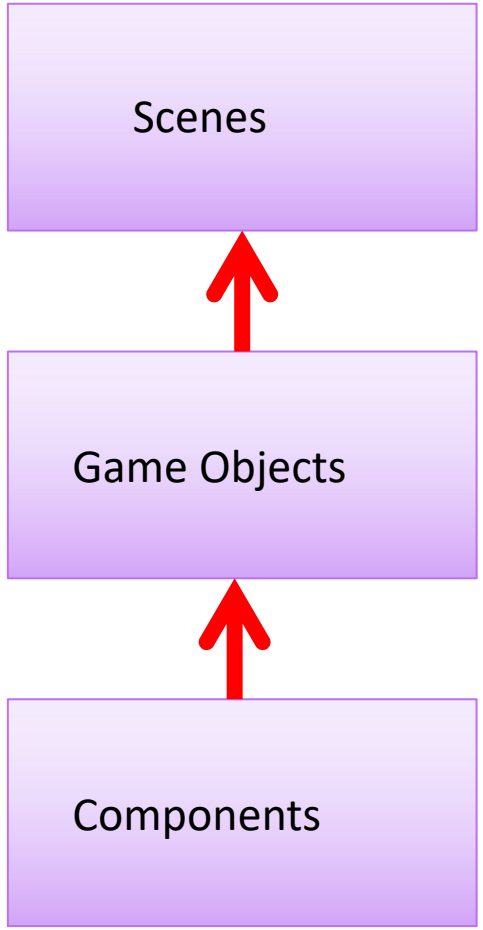
- When we select an object, an asset or other elements, the inspector view will allow us to access their properties and modify them
- The information displayed will depend on the selected item



Basic elements of Unity

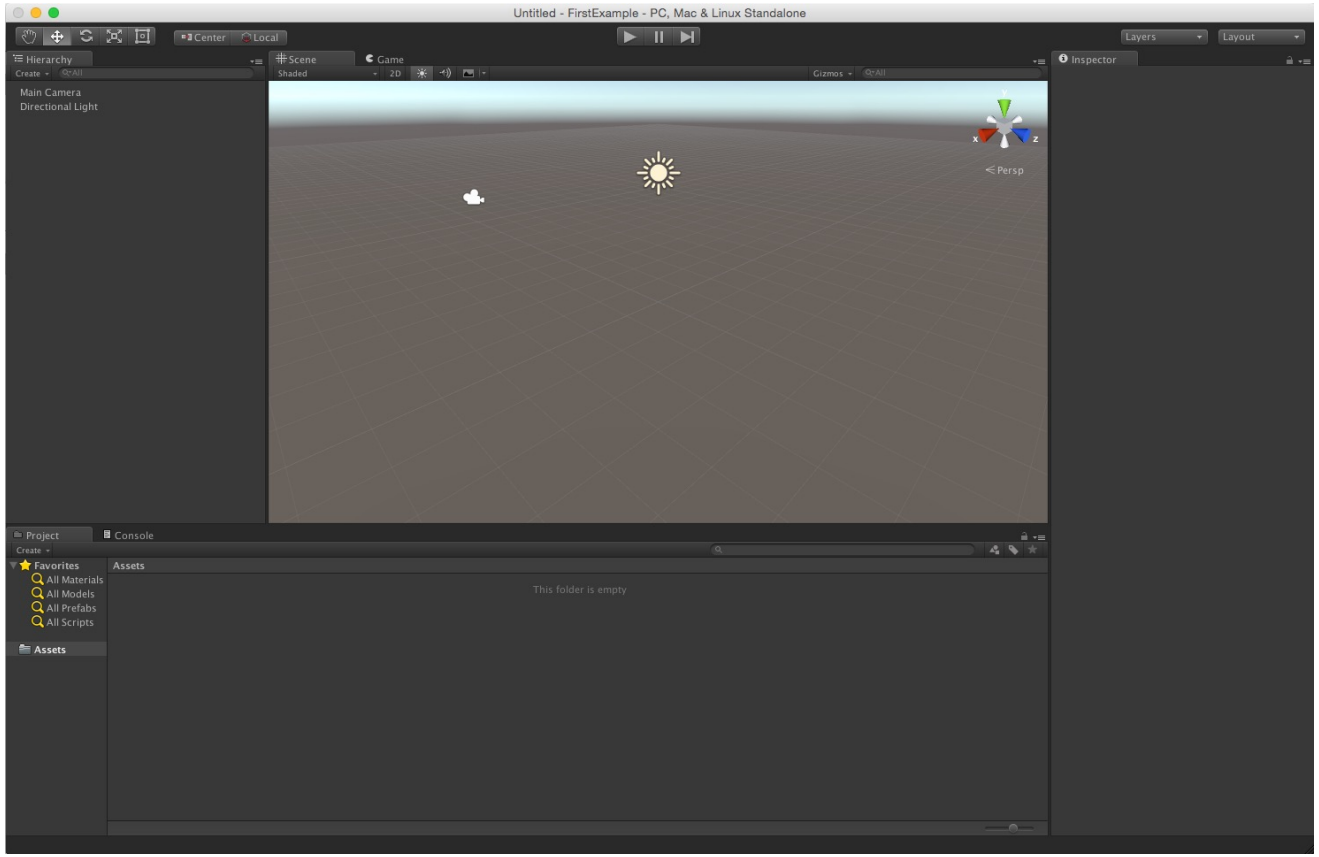
- The fundamental element of Unity is the GameObject, game objects such as cameras, 2D and 3D objects, particle systems, interface elements etc.
- These elements will be located in a specific scene.
- The scenes will help us divide our game or application into levels, different interaction screens, etc. It is a fundamental tool in the division into parts of our game.
- The GameObject will be created from Assets: resources of all kinds (3D geometries, textures, sounds, etc.) together with other elements created in Unity (scenes, prefabs, scripts, shaders etc.)
- The GameObject will have components. Based on the components that we add, the GameObject will be able to do some things or others.
- One of the most important components that we can associate with a GameObject is a Script, which will shape their behavior during the evolution of the game (we will use C # in our case)

Basic elements of Unity



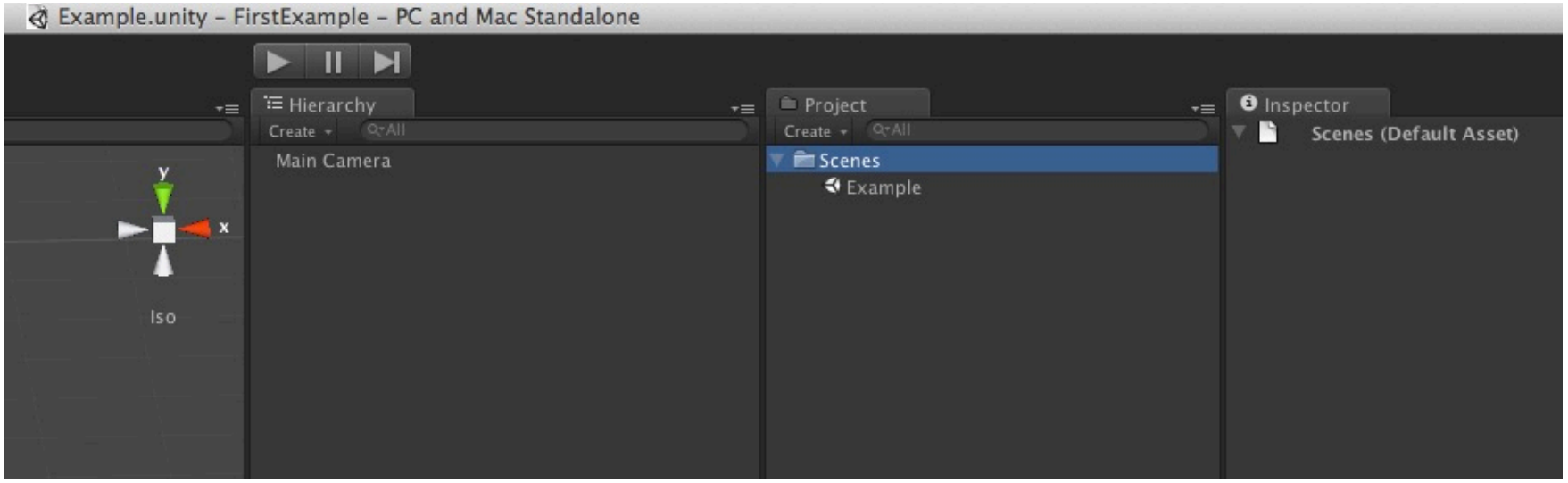
First example with Unity

- File -> New Project -> FirstExample



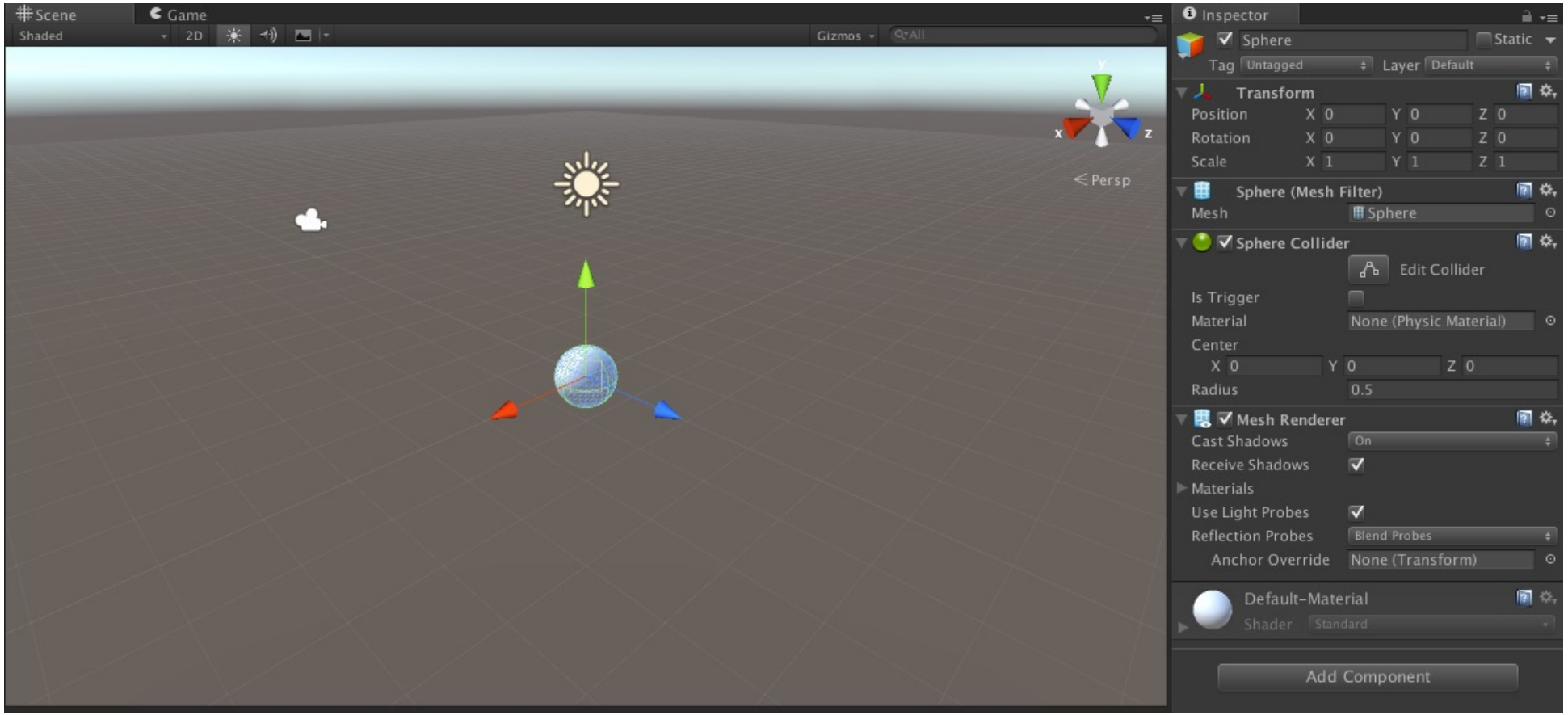
First example with Unity

- We create a new folder in 'Project' named 'Scenes' (now appears by default)
- You can change the name of the scene by going to Scenes folder in the Porject view and change the name of the scene (.unity extension)



First example with Unity

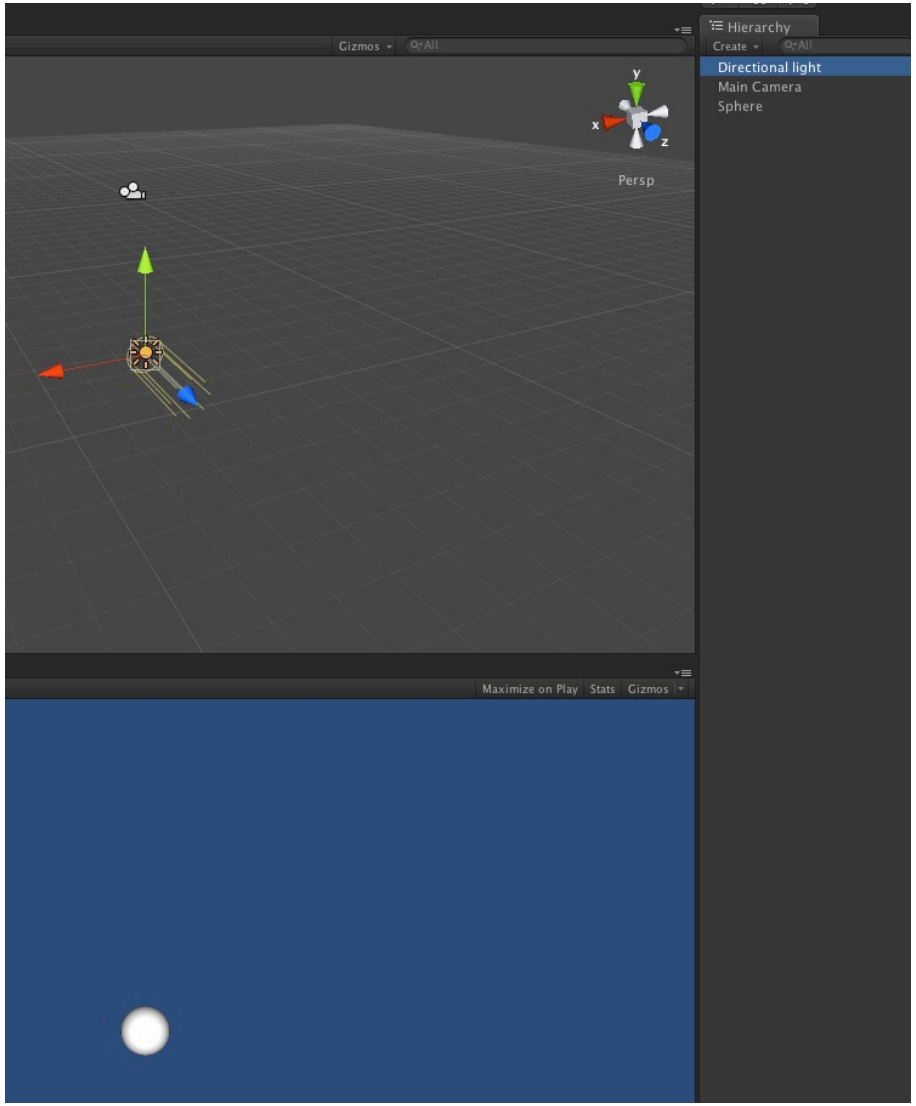
- We create a sphere GameObject->Create other->Sphere
- We make sure you have position 0,0,0



First example with Unity

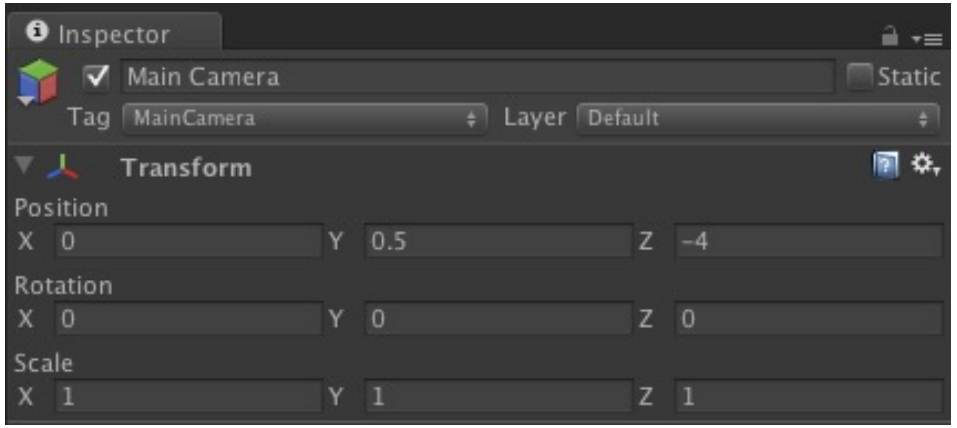
- A direct light (GameObject->Create other->Directional Light) appears by default

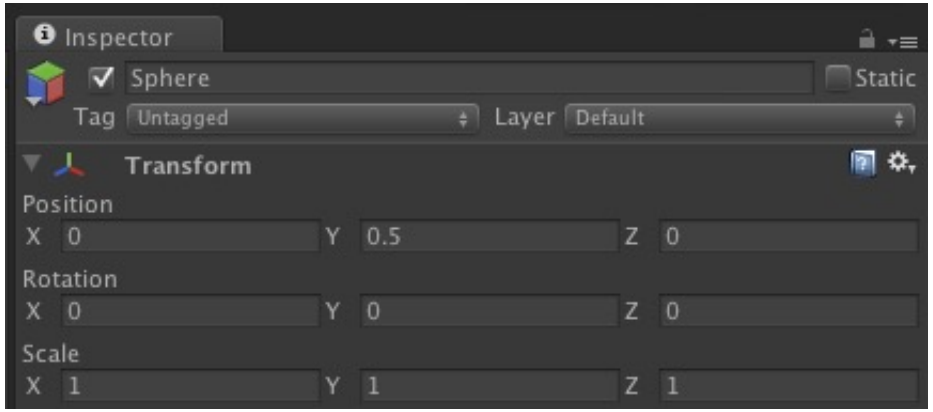
- We can try to move the light and see its effect (?)
- We can try to apply a rotation... any difference?

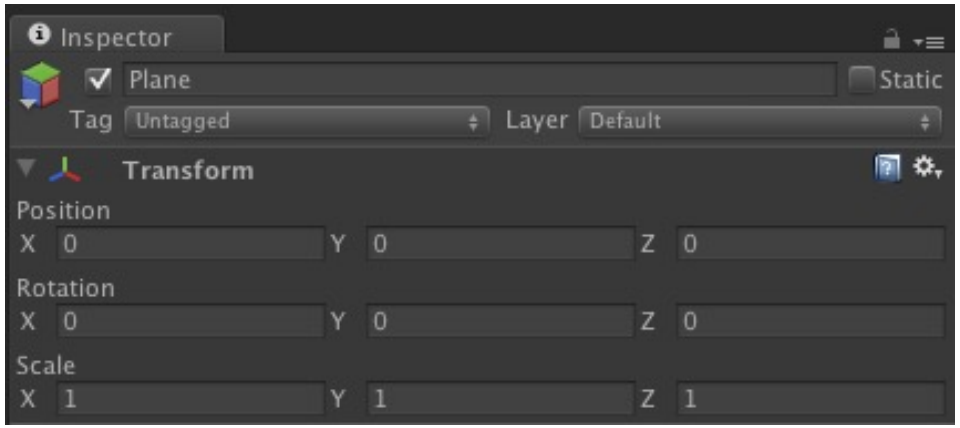


First example with Unity

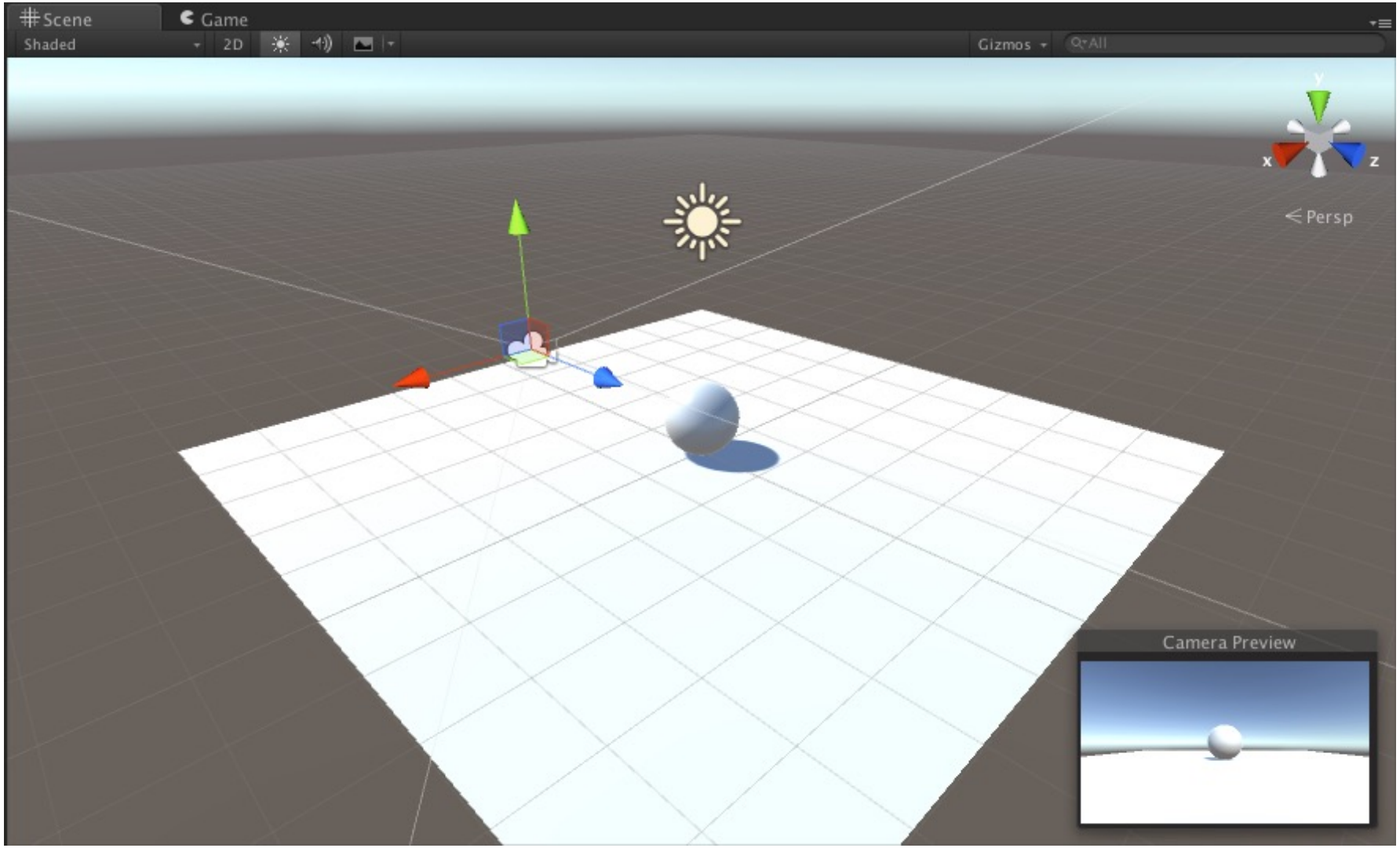
- We create a plane
- Using the manipulation tools or directly setting values in the inspector, we leave the different elements of the scene with the following values:





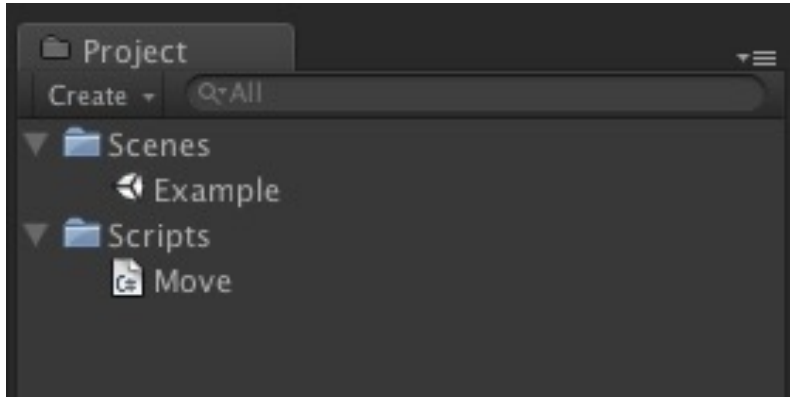


First example with Unity



First example with Unity

- We create a folder for scripts and create our first script in C#, we call Move
- We edit it with Visual studio:



Move.cs

No selection

```
using UnityEngine;
using System.Collections;

public class Move : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

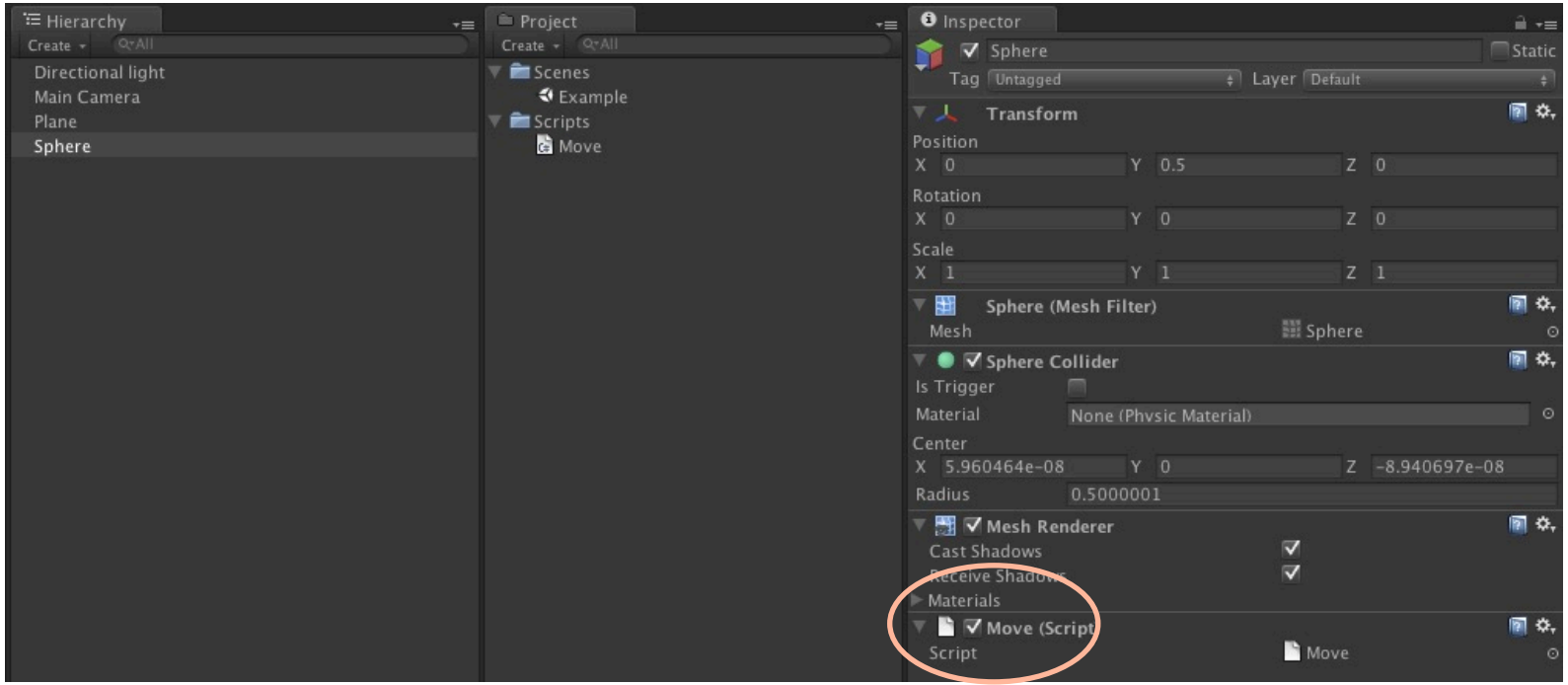
First example with Unity

- We edit the following:

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class Move : MonoBehaviour {
5
6      // Use this for initialization
7      void Start () {
8
9      }
10
11     // Update is called once per frame
12     void Update () {
13         transform.Translate(Input.GetAxis ("Horizontal"), 0, 0);
14     }
15 }
16
```

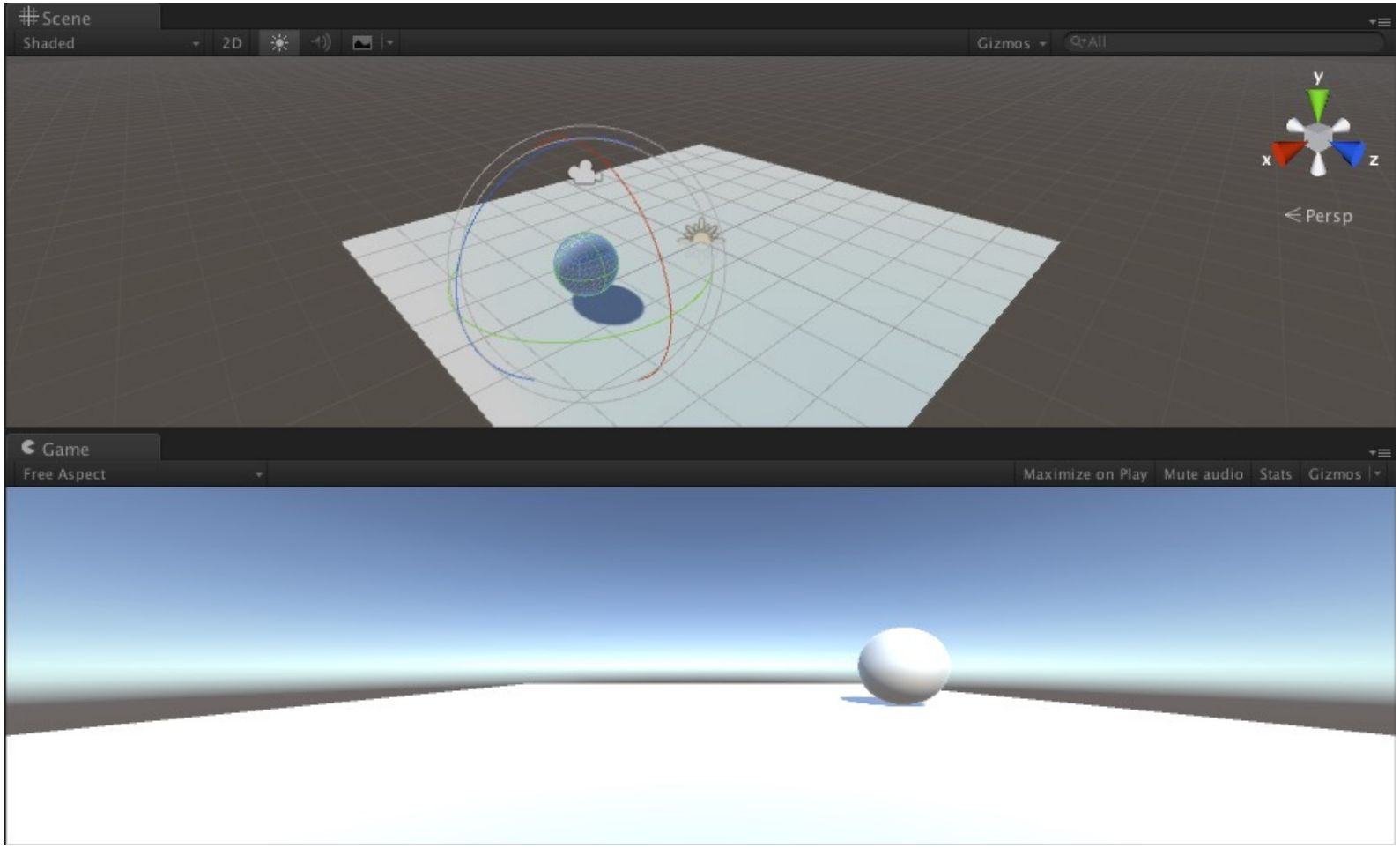
First example with Unity

- In the previous step we have created a script able to move an object on the X axis
- For an object to be controlled by the script we have to add this script as a component to the object (the sphere)
- This linking can be done in the editor of Unity, dragging the script on the sphere (in the hierarchy or scene view), or by selecting the sphere and dragging the script on the inspector



First example with Unity

- If we click 'Run', we will see the effect when pressing on the horizontal cursor keys



First example with Unity

- The units we use in Unity are meters and seconds
- How can we guarantee that the sphere will move a certain number of meters per second and also independently of how many frames per second get our endgame on the target platform?
- We can use Time.deltaTime, which measures the time elapsed since the last frame (last method Update() call of the GameObject)

```
// Update is called once per frame
void Update () {
    transform.Translate(Input.GetAxis ("Horizontal") * Time.deltaTime, 0, 0);
}
```

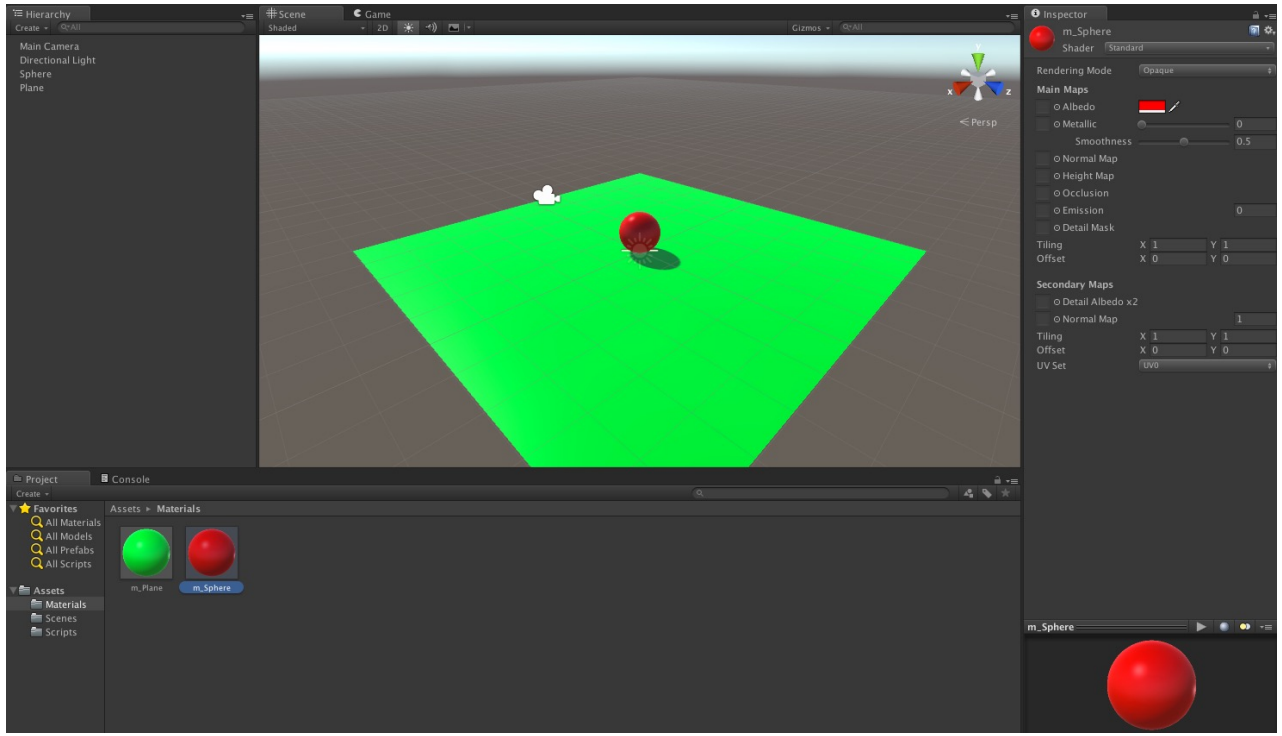
- GetAxis gives us values [-1, +1] with what multiplied by Time.deltaTime we can see in the Inspector that the sphere moves one unit (meter) per second in one direction or another
- This behavior will be maintained regardless of the final FPS(although more or less smooth)

First example with Unity

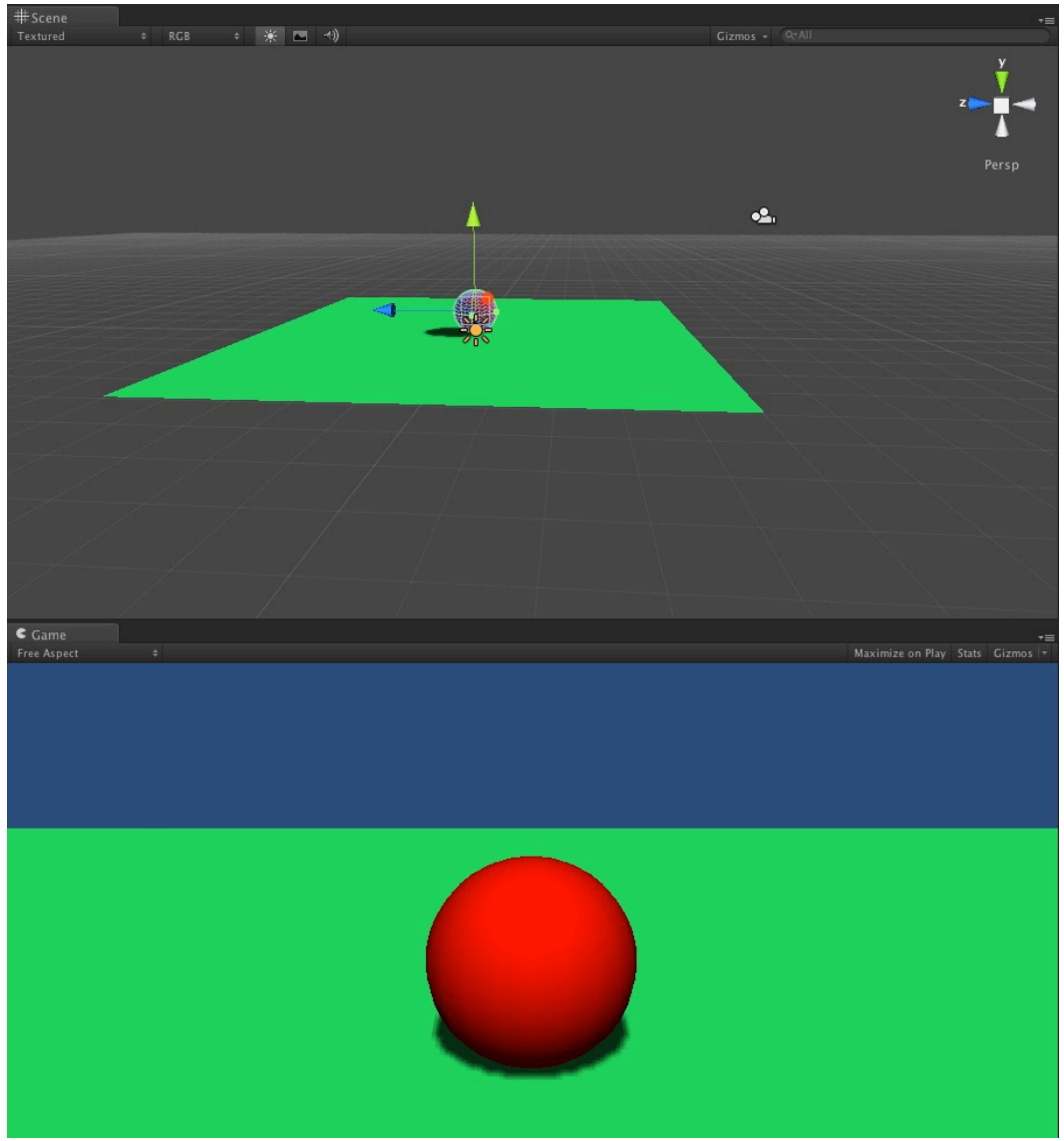
- The use of Time.deltaTime it's especially important in kinematic systems, that is, when we are the ones who directly control the movement of our objects
- Any amount or value of displacement that we apply in a method Update (that is executed every frame), implies a value by frame and, therefore, dependent on the rate of frames per second that our game or application is achieving at a specific time on a specific device
  - Let's imagine that on a platform I am getting 60 fps. This means that the method Update is called every 1/60 seconds. This will be precisely the value of Time.deltaTime.
  - If we want to move to 2 meters per second, the offset to be carried out by the method Update is 2 * 1.0f / 60.0f, that is to say, 2 * Time.deltaTime.
  - If, for whatever reason, fps vary during play, the value of Time.deltaTime will also do it and we will ALWAYS get to move 2 meters per second (and on any platform)

First example with Unity

- We create a new folder 'Materials' for materials (Create Folder on the Project view)
- We create two materials m_Sphere and m_Plane (Right button, Create Material) and select a color (Albedo) for them
- We assign these materials to the sphere and plane respectively (again, we have several possibilities, drag the material over the object or select the object and drag the material over the Inspector)



First example with Unity



First example with Unity

Exercise:

- Make the sphere also move in the Z axis with the cursor