Università di Genova | **DIBRIS** DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Augmented Reality

Lecture 5 – Introduction to Unity3D: Scripting

Manuela Chessa – manuela.chessa@unige.it

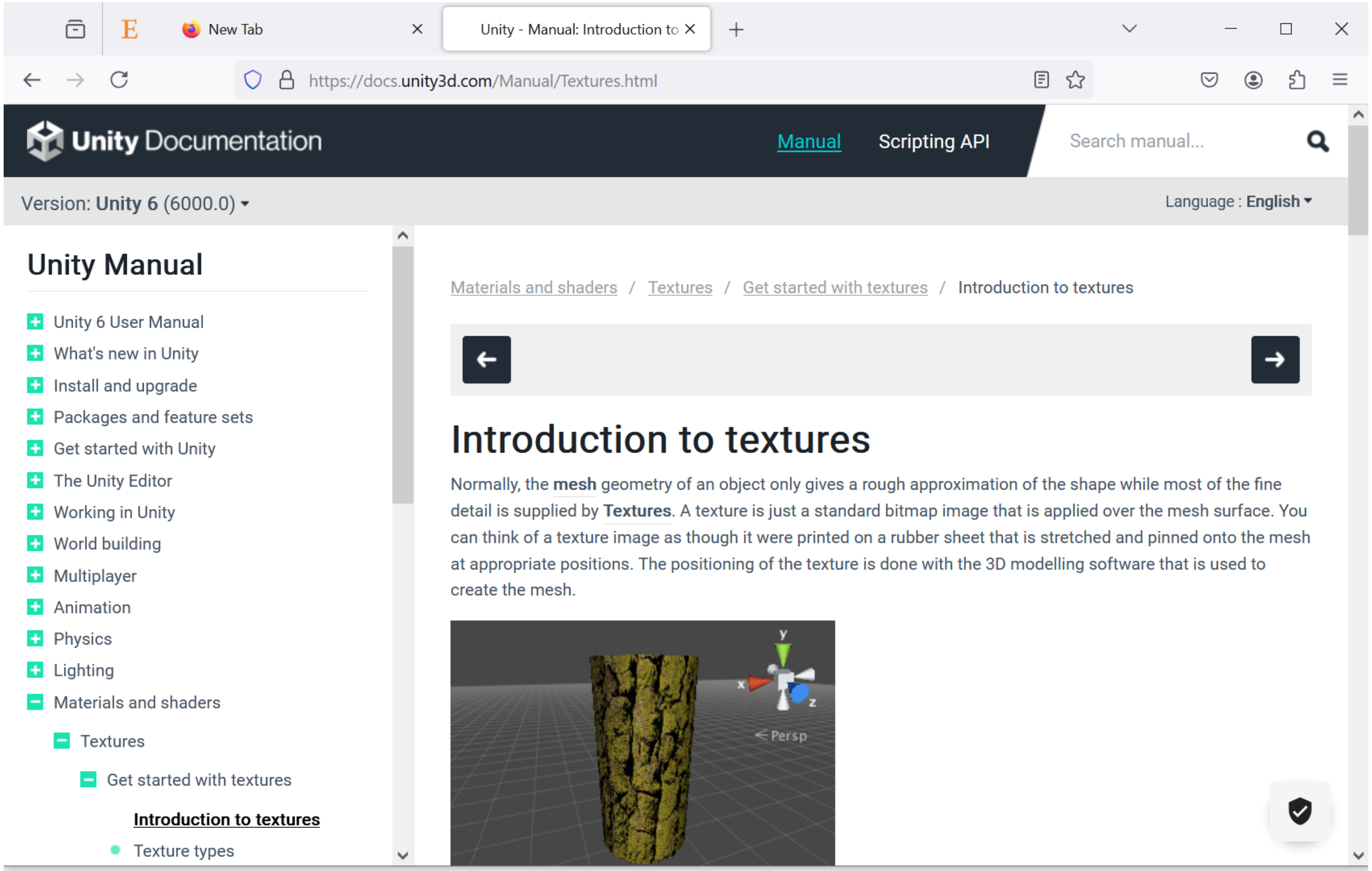
Fabio Solari – fabio.solari@unige.it

Summary

- Textures
- C#: I/O, classes and objects, generics and enumerators
- Scripting in Unity 3D: C# and documentation
- GameObject, Start(), Update() and FixedUpdate() functions
- Examples:
  - Update() and FixedUpdate()
  - External GameObject
  - Empty GameObject
  - Run-time textures
  - Run-time primitives
  - Input: keyboard and mouse

Textures

example


New Tab
Unity - Manual: Introduction to
https://docs.unity3d.com/Manual/Textures.html
Unity Documentation
Manual
Scripting API
Search manual...
Version: Unity 6 (6000.0)
Language : English
y
x
z
< Persp

Unity Manual

- Unity 6 User Manual
- What's new in Unity
- Install and upgrade
- Packages and feature sets
- Get started with Unity
- The Unity Editor
- Working in Unity
- World building
- Multiplayer
- Animation
- Physics
- Lighting
- Materials and shaders
  - Textures
    - Get started with textures
      - **Introduction to textures**
      - Texture types

Materials and shaders / Textures / Get started with textures / Introduction to textures

Introduction to textures

Normally, the **mesh** geometry of an object only gives a rough approximation of the shape while most of the fine detail is supplied by **Textures**. A texture is just a standard bitmap image that is applied over the mesh surface. You can think of a texture image as though it were printed on a rubber sheet that is stretched and pinned onto the mesh at appropriate positions. The positioning of the texture is done with the 3D modelling software that is used to create the mesh.

Scripting in Unity 3D

- All Unity3D games usually include **scripts** written in **C#** or Javascript.
- In this course, we will only use C#, an object-oriented programming language developed by Microsoft within its .NET framework.

- Visual Studio is installed by default when you install Unity on Windows and macOS

C#: I/O

- A class (`Tester`) is defined that contains the `Main()` (*the program entry point*).
- From `Main()` the various *static methods* (tests) that implement *what you want to verify* are called .

```
using System;
using System.Collections.Generic;
using System.Text;

namespace test
{
```

```
class Tester
{
  static void Main(string[] args)
  {
     TestIOKeyboard();
  }
 public static void TestIOKeyboard()
  {
  //...
  }}
```

C#: I/O

```
public static void TestIOKeyboard()
{
 double f;
 string str;
 Console.Write("Enter a double:");
 str = Console.ReadLine();

 f = Convert.ToDouble(str);
 Console.WriteLine("You entered: {0}\n", f);

 Console.Write("Enter a string:");
 str = Console.ReadLine();
 Console.WriteLine("You entered: {0}", str);
}
```

It *converts* the literal (string) representation of a number to its corresponding numeric value.

A possible output

```
Enter a double:-1.2e2
You entered: -120

Enter a string: try
You entered: try
```

C#: file

```
public static void TestFile()
{
 StreamReader fin = new StreamReader("file1.txt");
 StreamWriter fout = new StreamWriter("file2.txt");

 string sInput1,sInput2;
 double price;

 while ((sInput1 = fin.ReadLine()) != null &&
          (sInput2 = fin.ReadLine()) != null)
 {
     fout.WriteLine(sInput1);
     price = Convert.ToDouble(sInput2) * 1.2;
     fout.WriteLine(price);
 }
 fin.Close();
 fout.Close();
}
```

Read a file (`file1.txt`) containing a list of items and relative prices and write a new file (`file2.txt`) with the prices increased by 20%.

`file1.txt`

```
Article1
1.2
Article2
4.6
Article3
10.0
```

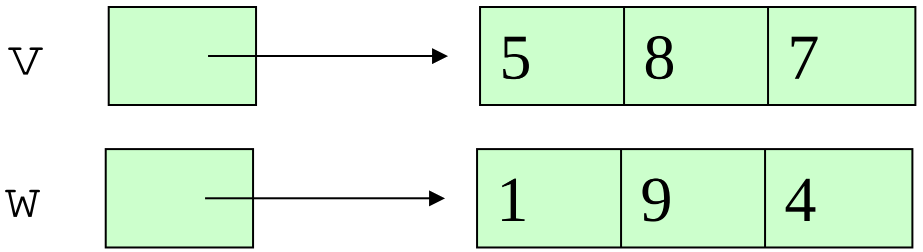
`file2.txt`

```
Article1
1.44
Article2
5.52
Article3
12.0
```

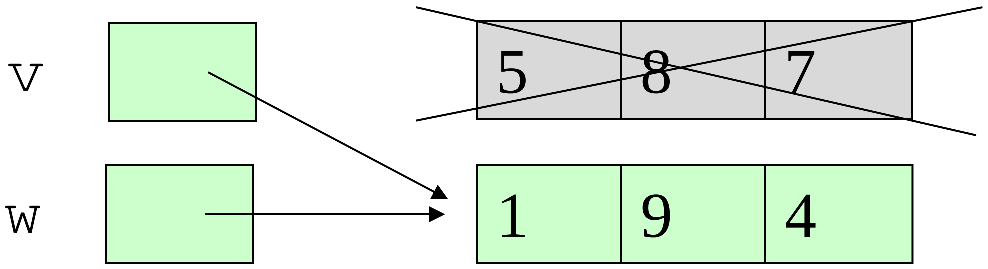
C#: 1D array of primitive data

```
int[] v = new int[3];
v[0]=5; v[1]=8; v[2]=7;
int[] w = {1,9,4};
```

Arrays are types by *reference*



```
v=w;
```



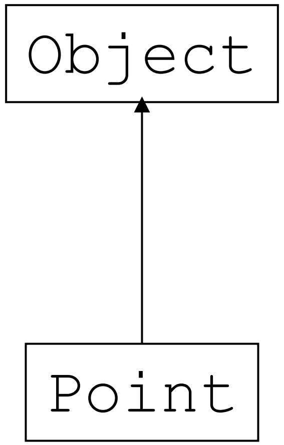
Garbage Collection

C#: classes and objects

All classes automatically derive from the base `Object` class, so they inherit its methods.

For example, the programmer-defined class `Point` derives from `Object`. Then an inheritance hierarchy is automatically created.

```
namespace System
{
  public class Object
  {
    public Object();
    public virtual Boolean Equals(Object obj);
    public virtual Int32 GetHashCode();
    public virtual String ToString();
    ...
  }
}
```



C#: classes and objects

Let's see a possible implementation of a class that represents the points of the plane. The `Point` class is defined within the *Point.cs* file and the class containing the `Main()` in the *PointTester.cs* file .

```
public class Point
{
   private int x;
   private int y;

   public Point() { }

   public Point(int a, int b)
   {
     x = a;
     y = b;
   }
```

```
public Point(int a=0, int b=0)
```

...

C#: classes and objects

```
...
public int X
{
 get
 {
   return x;
 }
 set
 {
   x = value;
 }
}
public int Y
{
 get
 {
   return y;
 }
 set
 {
   y = value;
 }
}
```

Instead of the traditional `Get()` and `Set()` methods, used to modify the state of the object, properties (**properties**) have been used, which *<u>simulate</u>* public access to data fields

Method inherited from `Object` and modified by derived class `Point`

```
public override string ToString()
{
  string tmp;
  tmp = x + " " + y;
  return tmp;
}
}
```

C#: classes and objects

Default constructor

```
static void Main()
{
        Point p = new Point();
        Console.WriteLine(p);

        Point[] pv = new Point[3];
        for (int i = 0; i < pv.Length; i++)
                pv[i] = new Point(5+i, 5+i);

        pv[1].X = 9;
        pv[1].Y = 9;

        for (int i = 0; i < pv.Length; i++)
                Console.WriteLine(pv[i].ToString());
               // Console.WriteLine(pv[i]);//it is ok
}
```

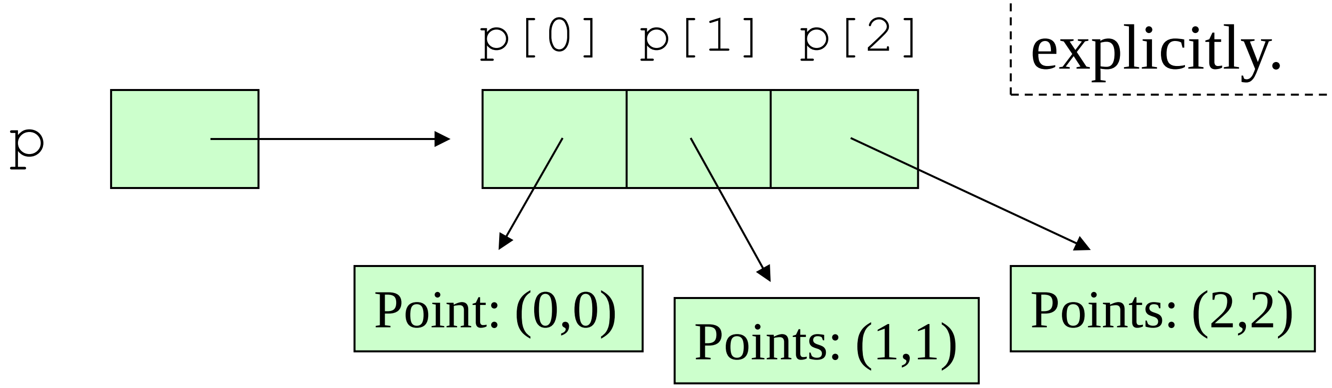
A possible output

```
0 0
5 5
9 9
7 7
```

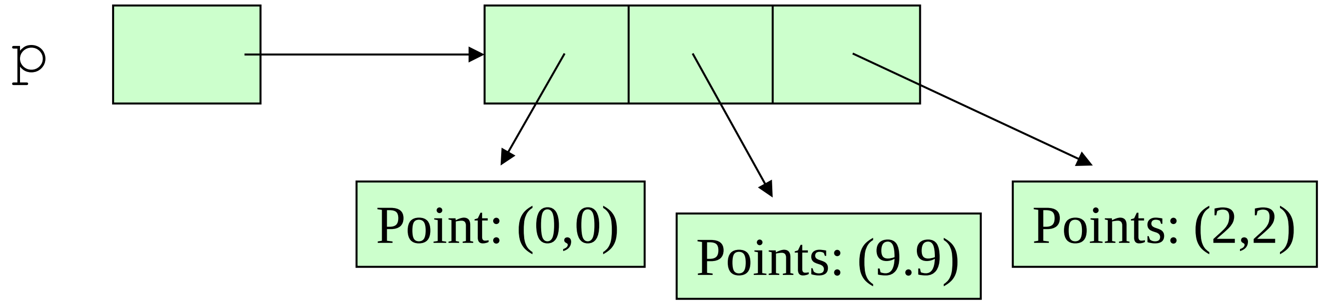
C#: 1D array of objects

```
Point[] p = new Point[3];
p[0] = new Point(0,0);
p[1] = new Point(1,1);
p[2] = new Point(2,2);
```

Object arrays contain *references to objects*, and individual elements *must* be *created* explicitly.



```
p[1].Set(9,9);
```



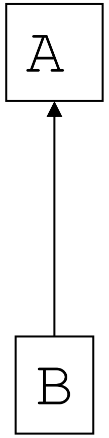
C#: inheritance

```
class A{
  protected string a;
  private int b;
  public void MethodA(){};
  ...
}


class B: A
{
  private int c;
  public void MethodB(){};
...}
```

base class

class B derived from A:
B inherits all members and methods of A and can specify their behavior and by adding others.



C#: the generics

Allows the programmer *to defer the specification of one or more types* until the class or method is declared and instantiated. For example:

```
public class GenericList <T>{

        void Add (T input ) { }

}

class TestGenericList{

        private class AClass{ }

        static void Main (){

                // Declare a list of type int

                GenericList<int> list1 = new GenericList<int>() ;

                // Declare a list of type string

                GenericList<string > list2 = new GenericList< string >() ;

                // Declare a list of type example class

                GenericList<AClass> list3 = new GenericList< AClass>() ;

        }

}
```

C#: the enumerators

Enumerations are used frequently in games to indicate *different states of the game* (e.g., splash screen, paused, etc.):

```
       public enum MyEnum{ TYPE1 , TYPE2 , TYPE3 };
```

They are often used in conjunction with switch statements.

```
       public void PrintEnum( MyEnum t) {

              switch(t) {

              case MyEnum.TYPE1:

                     Console.WriteLine(" T1 ");

                     break ;

              case MyEnum.TYPE2:

                     Console.WriteLine(" T2 ");

                     break ;

              case MyEnum.TYPE3:

                     Console.WriteLine(" T3 ");

                     break ;

} }
```

C#: the enumerators

Each `enum` member has an integral value, following declaration order, from 0 to N.

Explicit values can be assigned in declaration:

```
public enum MyEnum{

     TYPE1 = 10 ,

     TYPE2 = 20

};
```

And retrieved by explicit casting:

```
int t1Val = (int) MyEnum.TYPE1 ;

Console.WriteLine( t1Val ); // =10
```

Scripting in Unity 3D

The documentation of Unity3D is quite complete, and you can access it from:

https://docs.unity3d.com/Manual/index.html?_ga=2.198809647.1675259443.1679475753-2044810871.1677599519&_gac=1.48692564.1678706197.Cj0KCQjwk7ugBhDIARIsAGuvgPaGn85GmbFhl1jUIHtzn0X2OZ-qh3gwE4YzF0ADLOrfMlW9DvSfoH0aAtG4EALw_wcB

Here:
https://docs.unity3d.com/ScriptReference/
you can find the class documentation for Unity3D
On the left, follow UnityEngine -> classes

The Unity3D scripting system and the Unity architecture are explained here:
https://docs.unity3d.com/Manual/ScriptingSection.html

Scripting in Unity 3D

- A `GameObject` is each one of the entities present in a Scene of your game.
- Gameobjects created in Unity’s Editor can be controlled by scripts.
- Each GameObject in Unity contains a collection of Components (lights, colliders, animations, etc.).
- Scripts can be seen as behaviour component of a game object.
- Scripts need to be attached to a game object to work.

Scripting in Unity 3D

- A GameObject may contain multiple scripts.
- Ideally, each script should take care of a particular behaviour of the component (i.e. functionality).
- When creating a new C# Script from the Unity3D editor, the initial code looks like this:

```
using UnityEngine;
using System.Collections;
public classMainPlayer: MonoBehaviour{
      // Use this for initialization
      void Start () { }
      // Update is called once per frame
      void Update () { }
}
```

Scripting in Unity 3D

- A script makes its connection with the internal workings of Unity **by implementing a class which derives from the built-in class** called `MonoBehaviour`.
- The name of the class is taken from the name you supplied when the file was created.
- The class name and filename must be the same to enable the script component to be attached to a `GameObject`.
- The `Start` function will be called by Unity before gameplay begins (i.e. before the `Update` function is called for the first time) and is an ideal place to do any initialization.

Scripting in Unity 3D

- The `Update` function is the place to put code that will handle the frame update for the `GameObject`.
- This might include movement, triggering actions and responding to user input, basically anything that needs to be handled over time during gameplay.
- To enable the Update function to do its work, it is often useful to be able *to set up variables*, read preferences and make connections with other GameObjects before any game action takes place.
- Note that there are NOT constructors: this is because the construction of objects is handled by the editor and does not take place at the start of gameplay as you might expect.
- If you attempt to define a constructor for a script component, it will interfere with the normal operation of Unity and can cause major problems with the project.

Scripting in Unity 3D

- When creating a script, you are essentially creating your own new type of component that can be attached to Game Objects just like any other component.
- Just like other Components often have properties that are editable in the inspector, you can allow values in your script to be edited from the Inspector too.

```
public class CubeScript: MonoBehaviour{
      public string MyName;
      // Use this for initialization
      void Start () {
            Debug.Log("start!!! My name is" + MyName);
      }
      // Update is called once per frame
      void Update () {}
}
```

Scripting in Unity 3D

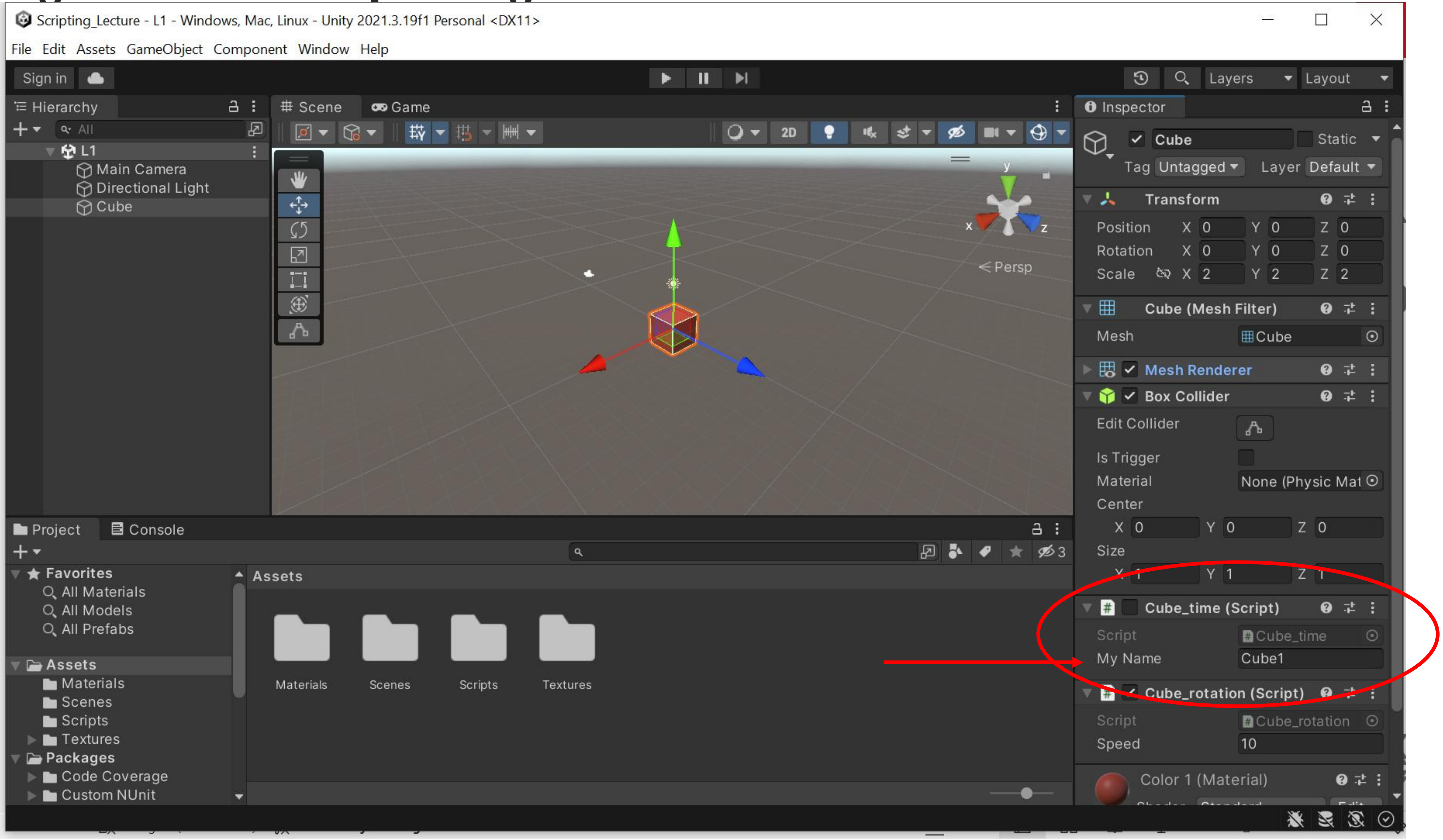
- Unity passes the control to each script intermittently by calling a determined set of functions, called **Event Functions**. The list of available functions is very large, here are the most commonly used ones.
- **Initialization**:
- `void Awake()`: First function to be called, when the first scene is load. It is only called if its game object is active. If not, it will be the first function called when the game object becomes active.
- `void Start()`: Start is called before the first frame update only if the script instance (the component) is enabled.

Scripting in Unity 3D

- **Regular Update Events:**
- `void Update()`: called at every frame, just before the frame is rendered. Used for regular updates. Update time interval can vary.

  `Time.deltaTime`: time in seconds it took to complete the last frame (time since last call to Update()).
- `void FixedUpdate()`: It is called every Physics step. Fixed update intervals are consistent.
- `void LateUpdate()`: called once per frame, after Update has finished. Any calculations that are performed in Update will have completed when LateUpdate begins. Example of use: camera that follows the player.

Update() and FixedUpdate()



Scripting_Lecture - L1 - Windows, Mac, Linux - Unity 2021.3.19f1 Personal <DX11>

Cube

Tag Untagged Layer Default

Transform

Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	2	Y	2	Z	2

Cube (Mesh Filter)

Mesh Cube

Box Collider

Is Trigger □

Material None (Physic Mat

Center X 0 Y 0 Z 0

Size X 1 Y 1 Z 1

Cube_time (Script)

Script Cube_time

My Name Cube1

Cube_rotation (Script)

Script Cube_rotation

Speed 10

Color 1 (Material)

Update() and FixedUpdate()

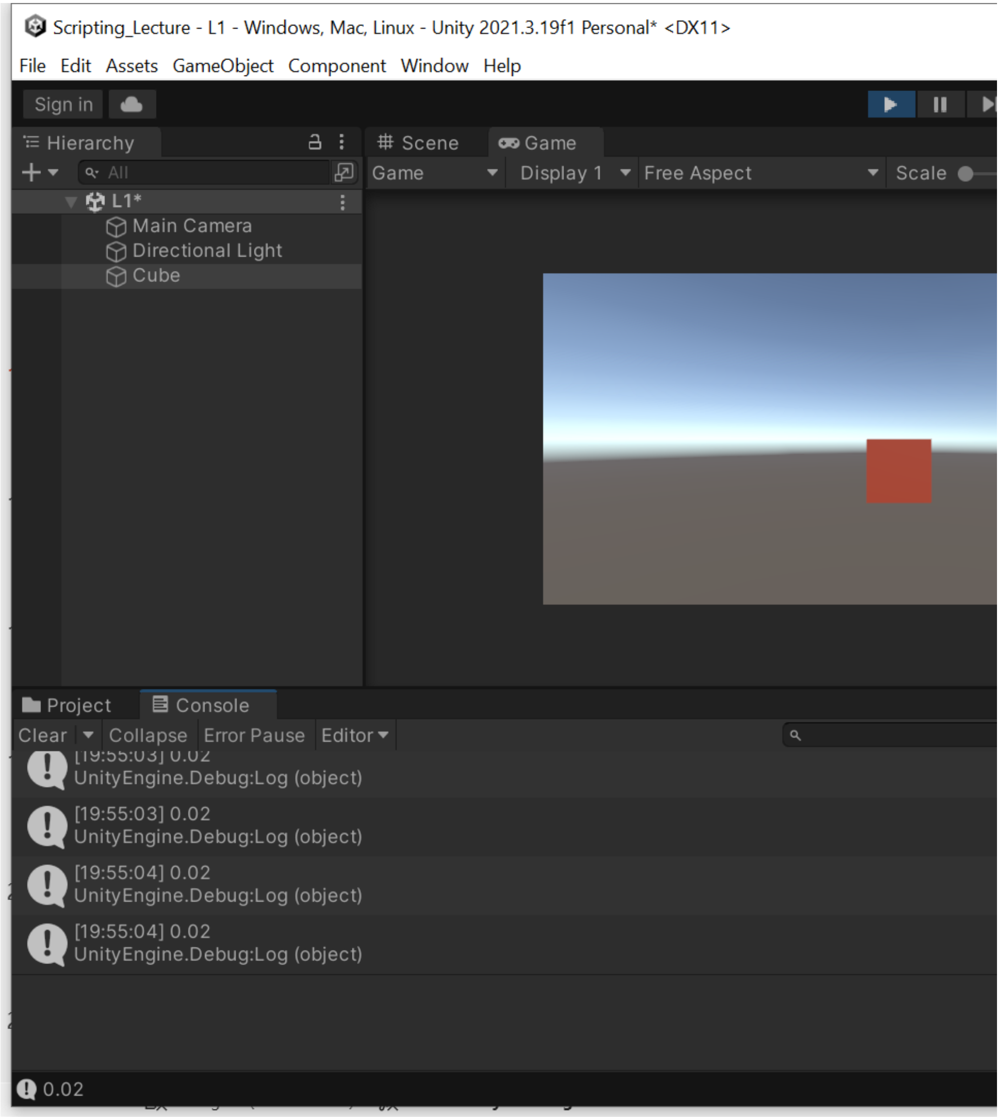
Attached to Cube

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

public class Cube_time : MonoBehaviour
{
    public string MyName;
    // Start is called before the first frame update
    void Start()    {
        Debug.Log("Start!!! My name is " + MyName);
    }

    // Update is called once per frame
    void Update()    {
        //Debug.Log(Time.deltaTime);
    }

    private void FixedUpdate()    {
        Debug.Log(Time.deltaTime);
    }
}
```



```
[19:55:03] 0.02
UnityEngine.Debug:Log (object)
[19:55:03] 0.02
UnityEngine.Debug:Log (object)
[19:55:04] 0.02
UnityEngine.Debug:Log (object)
[19:55:04] 0.02
UnityEngine.Debug:Log (object)
```

0.02

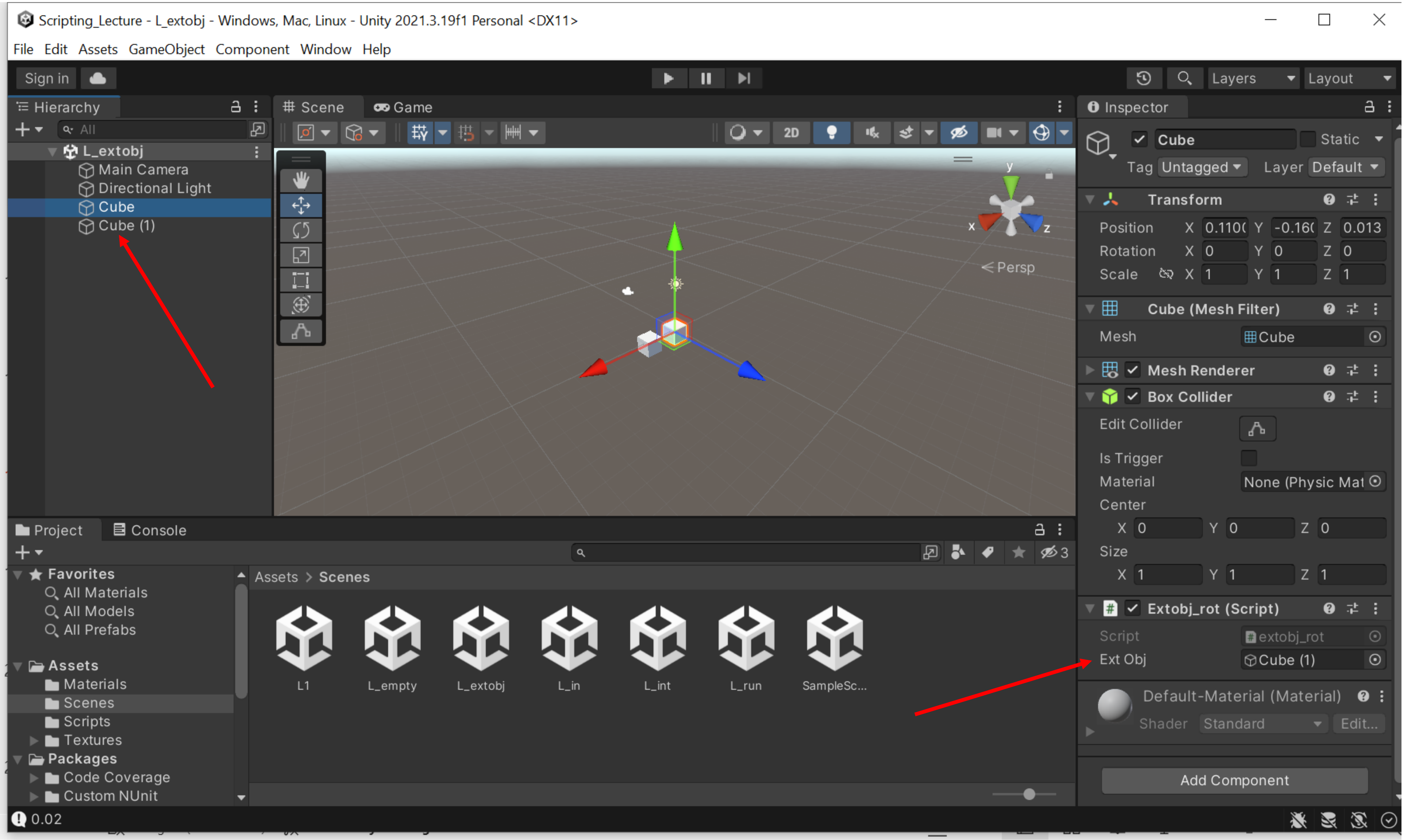
Update() and FixedUpdate()

Attached to Cube

```
public class Cube_rotation : MonoBehaviour
{
    public float speed;
    // Start is called before the first frame update
    void Start()
    {
        Debug.Log("Before transform: " + transform.position);
        transform.position = new Vector3(-2f, 0, 0);
        Debug.Log("After transform: " + transform.position);
    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(new Vector3(0.0f, speed * Time.deltaTime, 0.0f));
    }
}
```

External GameObject



Scripting_Lecture - L_extobj - Windows, Mac, Linux - Unity 2021.3.19f1 Personal <DX11>

Hierarchy

- L_extobj
  - Main Camera
  - Directional Light
  - Cube
  - Cube (1)

Inspector

Cube — Static
Tag Untagged — Layer Default

Transform
Position X 0.110(Y -0.16(Z 0.013
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Cube (Mesh Filter)
Mesh: Cube

Mesh Renderer

Box Collider
Edit Collider
Is Trigger
Material: None (Physic Mat
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1

Extobj_rot (Script)
Script: extobj_rot
Ext Obj: Cube (1)

Default-Material (Material)
Shader Standard — Edit...

Add Component

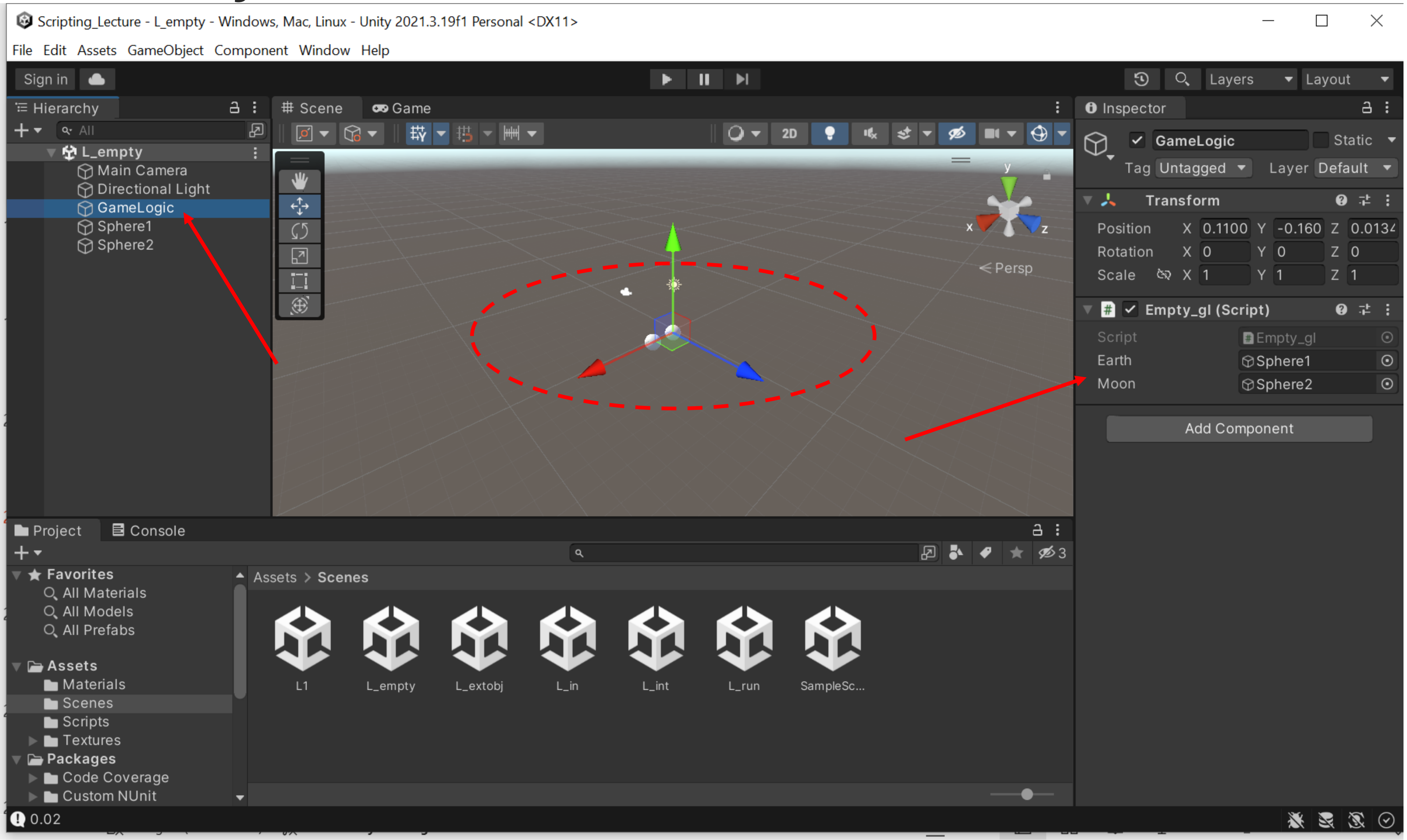
Project
Assets > Scenes: L1, L_empty, L_extobj, L_in, L_int, L_run, SampleSc...

External GameObject

```
public class extobj_rot : MonoBehaviour
{
    public GameObject extObj;
    // Start is called before the first frame update
    void Start()
    {
        GetComponent<MeshRenderer>().material.color = Color.green;
        extObj.transform.position = transform.position + new Vector3(-3.0f, 0.0f, 0.0f);
        extObj.GetComponent<MeshRenderer>().material.color = Color.red;
    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(0.0f, 10.0f * Time.deltaTime, 0.0f);
        extObj.transform.RotateAround(transform.position, Vector3.up, Time.deltaTime * 20.0f);
    }
}
```

Empty GameObject



☑ GameLogic □ Static

Tag Untagged Layer Default

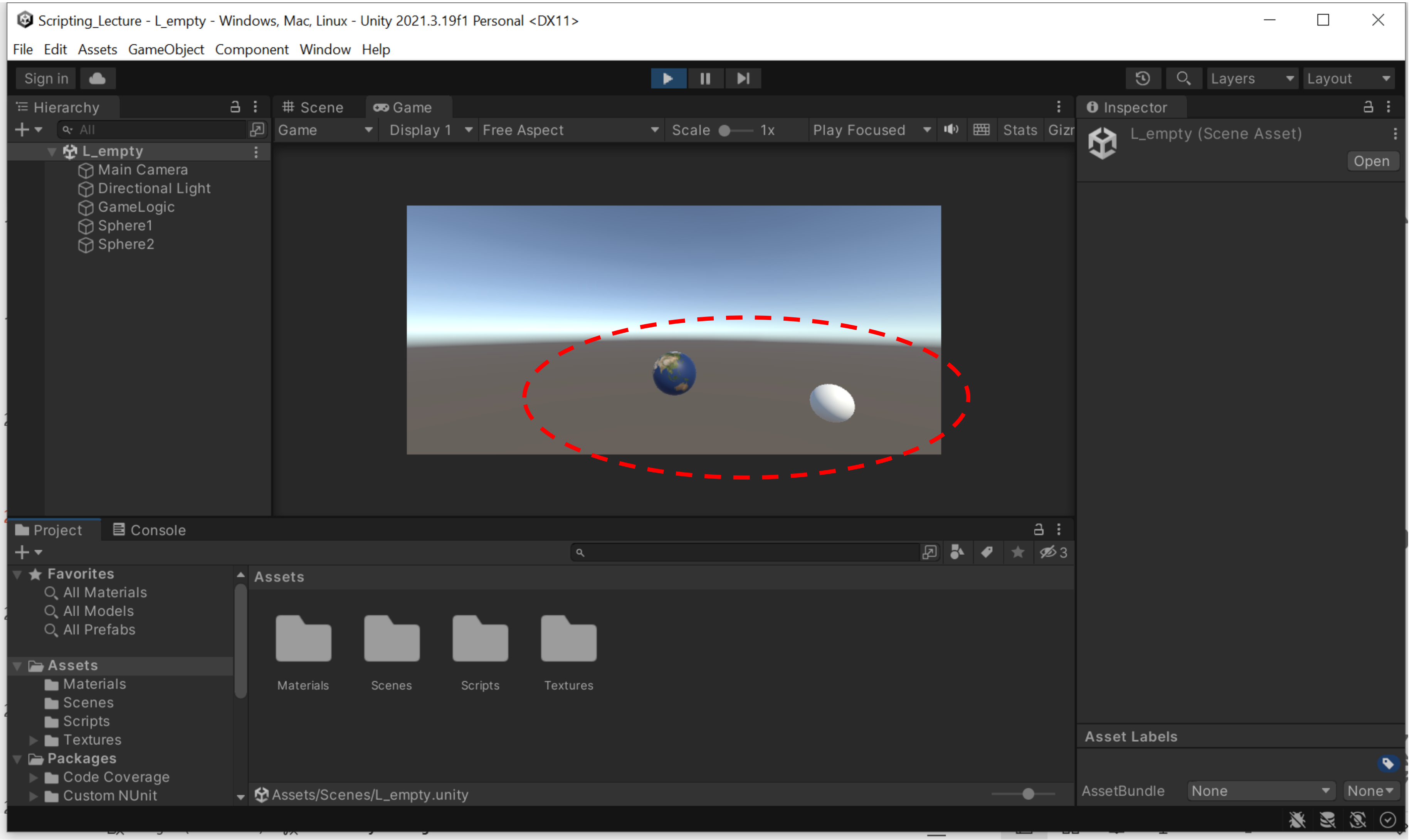
Transform

Position	X	0.1100	Y	-0.160	Z	0.0134
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

☑ Empty_gl (Script)

Script	Empty_gl
Earth	Sphere1
Moon	Sphere2

Empty GameObject and run-time textures

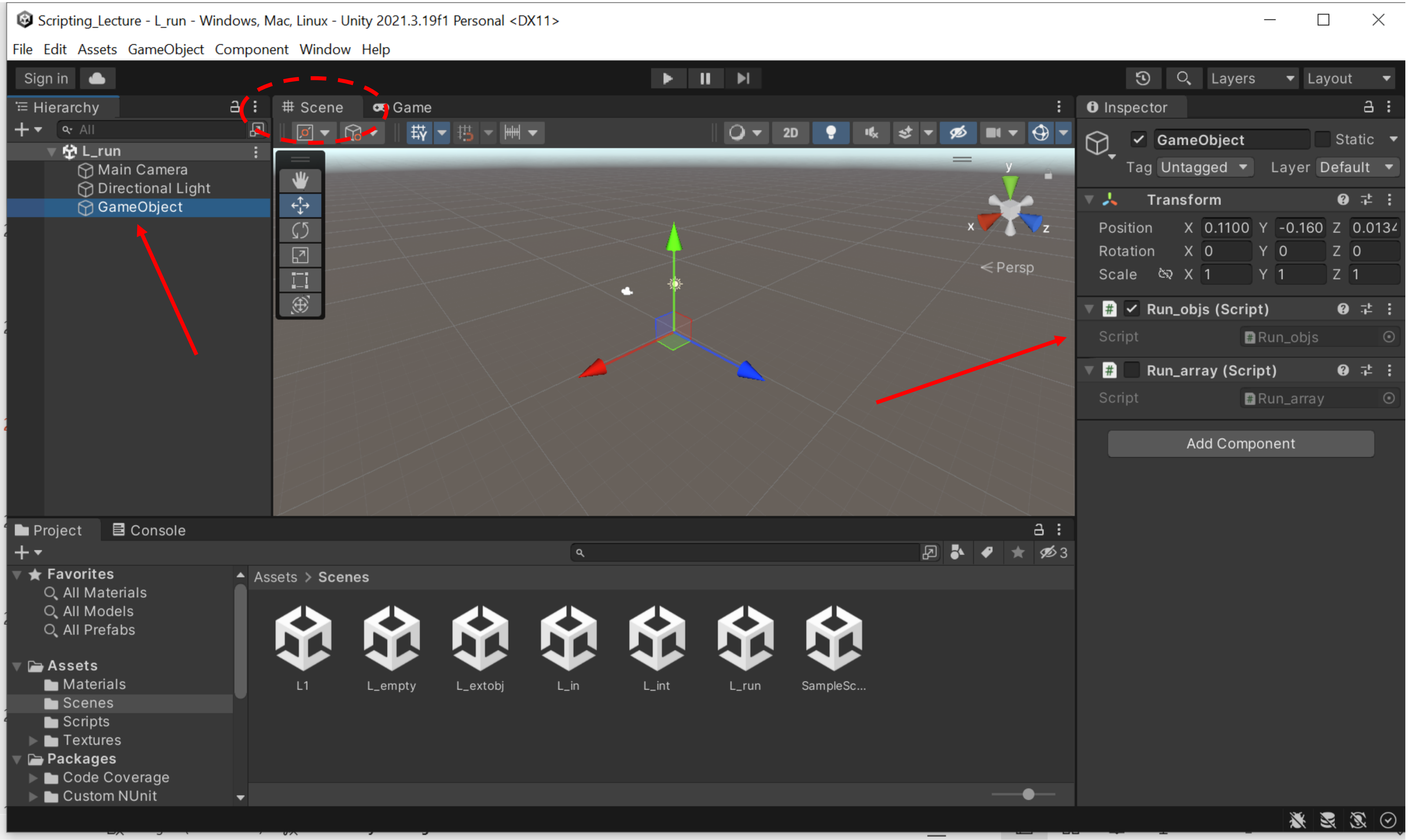


Empty GameObject and run-time textures

```
public class Empty_gl : MonoBehaviour
{
    public GameObject Earth;
    public GameObject Moon;
    // Start is called before the first frame update
    void Start()
    {
        //Earth
        Earth.transform.position = new Vector3(0.0f, 0.0f, 0.0f);
        byte[] bytes = File.ReadAllBytes("./Assets/Textures/2k_earth_daymap.jpg");
        Texture2D texture = new Texture2D(100, 100);
        texture.filterMode = FilterMode.Trilinear;
        texture.LoadImage(bytes);
        MeshRenderer meshRenderer = Earth.GetComponent<MeshRenderer>();
        meshRenderer.material.SetTexture("_MainTex", texture);
        //Moon
        Moon.transform.position= new Vector3(3.0f, 0.0f, 0.0f);
        Moon.transform.localScale= new Vector3(0.5f, 0.5f, 0.5f);
    }

    // Update is called once per frame
    void Update()
    {
        Earth.transform.Rotate(0.0f, 10.0f * Time.deltaTime, 0.0f);
        Moon.transform.RotateAround(transform.position, Vector3.up, Time.deltaTime * 20.0f);
    }
}
```

Run-time primitives



☑ GameObject

Tag Untagged   Layer Default

Position X 0.1100 Y -0.160 Z 0.0134

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

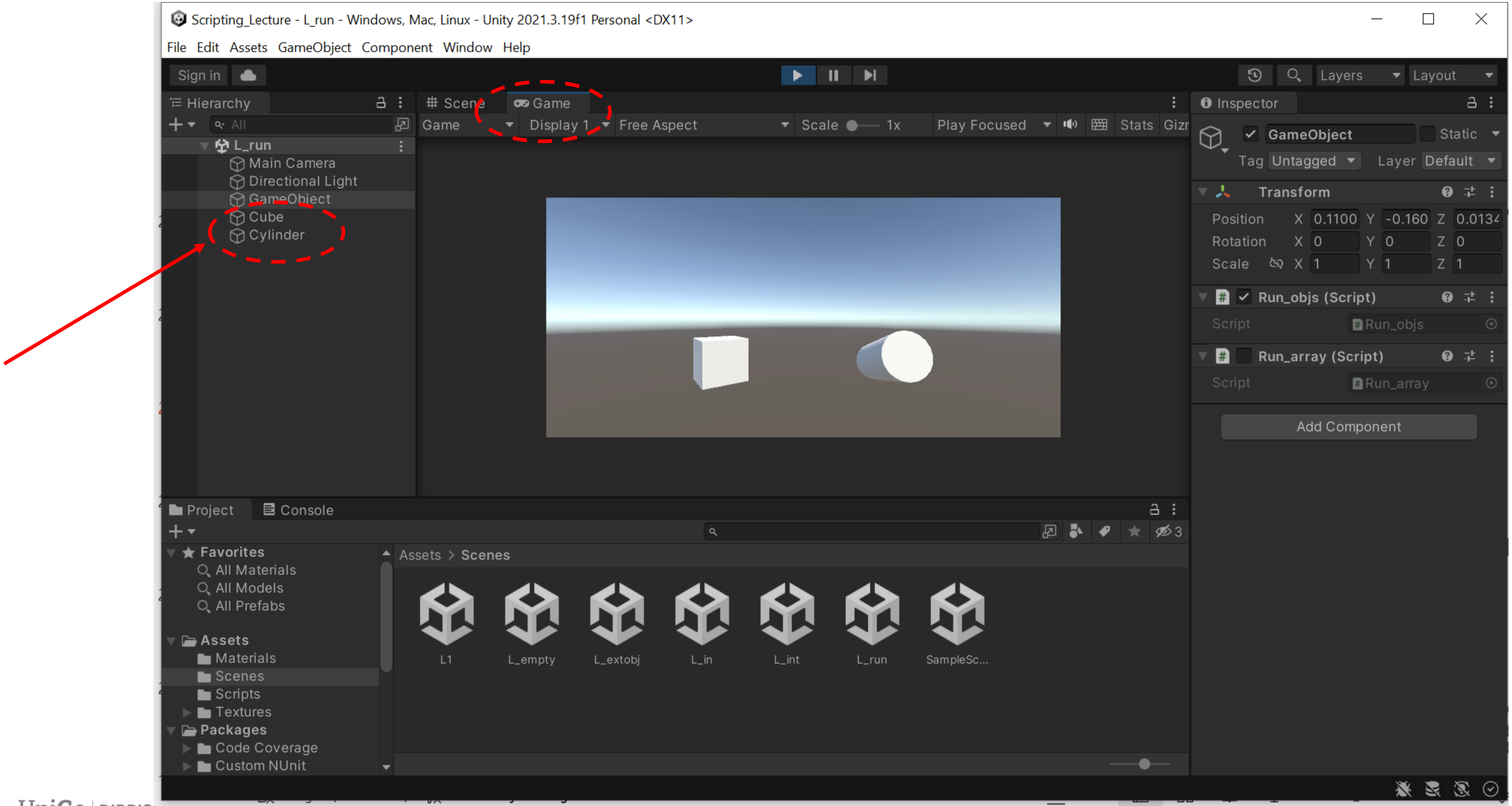
☑ Run_objs (Script)

Script Run_objs

□ Run_array (Script)

Script Run_array

Run-time primitives



Scripting_Lecture - L_run - Windows, Mac, Linux - Unity 2021.3.19f1 Personal <DX11>

GameObject ☑ — Static □ — Tag Untagged — Layer Default

Transform

Position	X	0.1100	Y	-0.160	Z	0.0134
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

☑ Run_objs (Script) — Script: Run_objs

□ Run_array (Script) — Script: Run_array

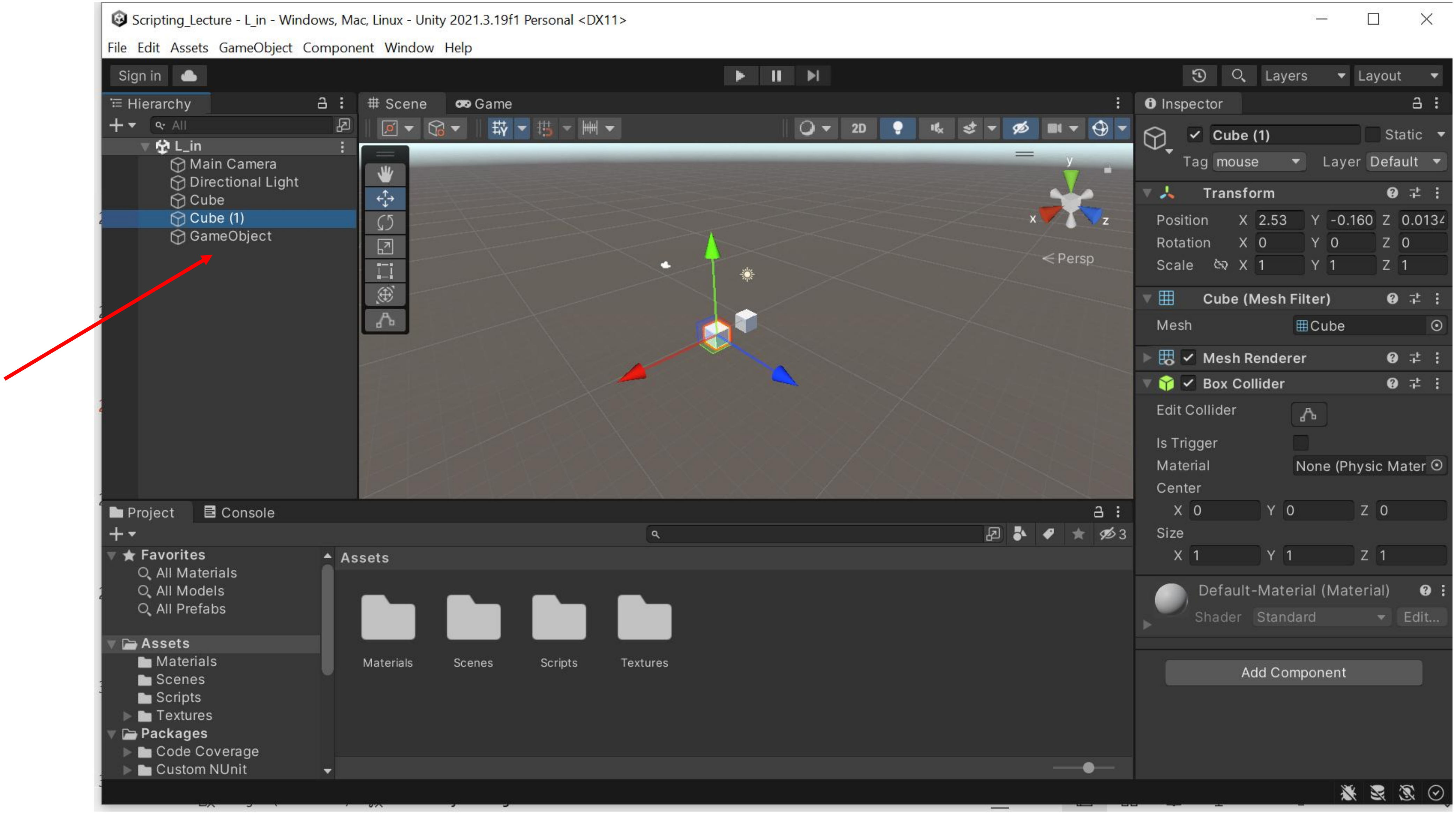
Run-time primitives

```
public class Run_objs : MonoBehaviour
{
    private GameObject MyCube, MyCylinder;
    // Start is called before the first frame update
    void Start()
    {
        MyCube = GameObject.CreatePrimitive(PrimitiveType.Cube);
        MyCube.transform.position = new Vector3(-2.0f, 0.0f, 0.0f);

        MyCylinder = GameObject.CreatePrimitive(PrimitiveType.Cylinder);
        MyCylinder.transform.position = new Vector3(2.0f, 0.0f, 0.0f);
    }

    // Update is called once per frame
    void Update()
    {
        MyCube.transform.Rotate(10.0f*Time.deltaTime * Vector3.up);
        MyCylinder.transform.Rotate(20.0f * Time.deltaTime * new Vector3(1.0f,0.0f,0.0f));
    }
}
```

Input: keyboard and mouse



Cube (1) Static

Tag mouse Layer Default

Transform

Position X 2.53 Y -0.160 Z 0.0134

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

Cube (Mesh Filter)

Mesh Cube

Mesh Renderer

Box Collider

Edit Collider

Is Trigger

Material None (Physic Mater

Center X 0 Y 0 Z 0

Size X 1 Y 1 Z 1

Default-Material (Material)

Shader Standard Edit...

Input: keyboard and mouse

Attached to Cube

```
public class in_key : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        GetComponent<MeshRenderer>().material.color = Color.red;
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown("r"))
            transform.position = transform.position + new Vector3(6f * Time.deltaTime,0.0f,  0.0f);

        if (Input.GetKeyDown("l"))
            transform.position = transform.position + new Vector3(-6f * Time.deltaTime, 0.0f, 0.0f);
    }
}
```

Input: keyboard and mouse

Attached to the empty GameObject

```
public class in_mouse : MonoBehaviour
{
    void Start(){
         }

    void Update()    {
        if (Input.GetMouseButtonDown(0))
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit))
            {
                //Select target
                //if (hit.transform.CompareTag("mouse"))
                if (hit.transform.name =="Cube (1)")
                {
                    hit.transform.gameObject.GetComponent<MeshRenderer>().material.color=
                        new Color(Random.Range(0.5f, 1f), Random.Range(0.5f, 1f),  0);
                }
            }
        }

    }
}
```

UniGe

DIBRIS