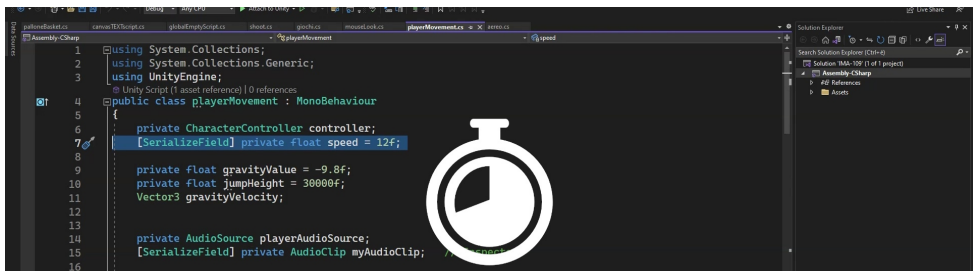


This PDF is available for free; feel free to share it!

[Subscribe to my YT channel](#) & turn on notifications to get updates on new videos!

NOTE - In the tutorial, you'll notice comments on other parts of the code in Italian. This is because I am Italian, and the scenario I'm showing you in this example is from a personal project. Please ignore those comments, as they pertain to parts of the script not relevant to this tutorial.



The screenshot shows a C# script named `playerMovement` in Unity. The script includes `using System.Collections;`, `using System.Collections.Generic;`, and `using UnityEngine;`. It defines a `playerMovement` class that inherits from `MonoBehaviour`. The class contains several private fields: `CharacterController controller`, `float speed = 12f` (marked with `[SerializeField]`), `float gravityValue = -9.8f`, `float jumpHeight = 30000f`, `Vector3 gravityVelocity`, `AudioSource playerAudioSource`, and `AudioClip myAudioClip` (also marked with `[SerializeField]`). A large stopwatch icon is overlaid on the script. Below the script, a table explains the relationship between FPS rate, deltaTime, and movement units.

	fps rate	deltaTime (secs)	units per frame	units per second
device A (slow)	10	0.10	0.10	1
device B	20	0.05	0.05	1
device C (fast)	50	0.02	0.02	1

Below the table, the script continues with `Debug.Log(gravityVelocity.y);`, a comment in Italian, and a `controller.Move` call that uses `Time.deltaTime` to calculate movement per frame.

Hello everyone!

In this tutorial, we will see what ***Time.deltaTime*** is in Unity and how to use it.

This beginner-friendly tutorial is perfect for those who are new to scripting in C# within Unity, as the topic is fundamental and deserves to be well understood, since you will encounter it in many scripts and tutorials.

The tutorial was created using Unity 2022, but ***Time.deltaTime*** has been around for several versions of the software and will likely continue to be in the future.

Time.deltaTime is a variable that represents the time elapsed, in seconds, between the last frame and the current frame (or, if you prefer, the time it took to complete the previous frame).

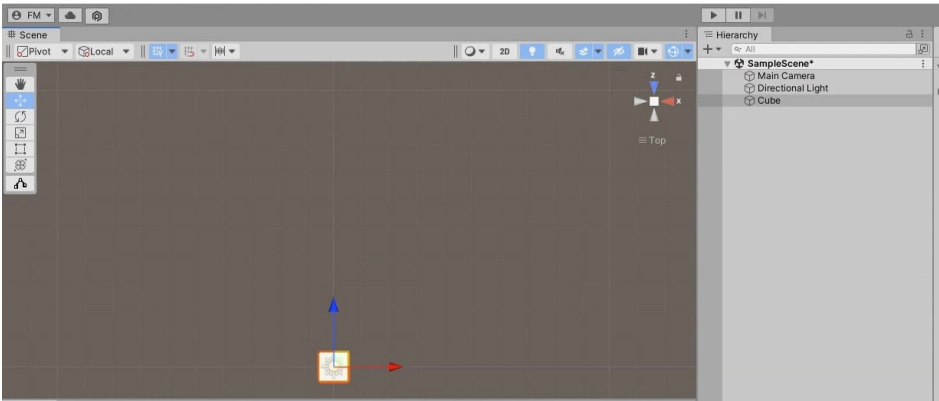
Supposing you want to move the object at 1 scene unit per second...

	fps rate	deltaTime (secs)	units per frame	units per second
device A (slow)	10	0.10	0.10	1
device B	20	0.05	0.05	1
device C (fast)	50	0.02	0.02	1

It is used to make movements and animations **frame rate-independent**, usually by multiplying an object's speed in units per second by *Time.deltaTime* in each frame.

This way, the speed per frame will vary based on the frame rate, **ensuring the same speed** on devices with different performance levels.

Let's see a practical example of what we just described, with a very simple case: **the movement of an object along an axis at a constant speed**. Insert an object into the scene, framing the scene from above with an Isometric view.



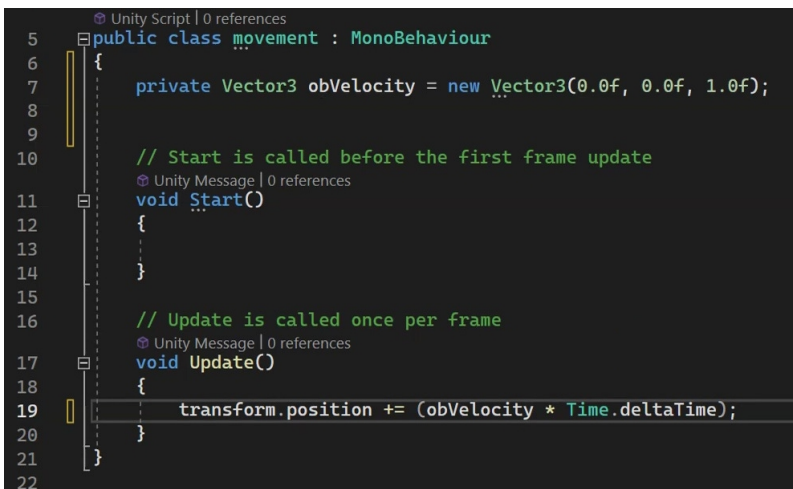
Give the object a **C# script**, calling it, for example, *movement*, then create a **Vector3** to indicate a displacement of one unit (which we want to cover in one second):

```
private Vector3 obVelocity = new Vector3(0.0f, 0.0f, 1.0f);
```

We know that *Update* is called at each frame; however, **we do not know how many frames per second the game will run**, as this varies based on several parameters.

If our frame rate were consistently 50 frames per second, we could modify *obVelocity* to (0.0f, 0.0f, 0.02f), moving 0.02 units with each call to *Update* (and, therefore, each frame), resulting in a displacement of one unit per second.

Since we don't have this possibility, in *Update* we will write:
*transform.position += (obVelocity * Time.deltaTime);*

A screenshot of a Unity C# script named 'movement' which inherits from 'MonoBehaviour'. The script is shown in a dark-themed editor. It includes a private Vector3 variable 'obVelocity' initialized to (0.0f, 0.0f, 1.0f). The 'Start()' method is empty. The 'Update()' method contains the line 'transform.position += (obVelocity * Time.deltaTime);'. Line numbers 5 through 22 are visible on the left margin. Comments indicate that 'Start' is called before the first frame update and 'Update' is called once per frame.

```
5 public class movement : MonoBehaviour
6 {
7     private Vector3 obVelocity = new Vector3(0.0f, 0.0f, 1.0f);
8
9
10    // Start is called before the first frame update
11    void Start()
12    {
13    }
14
15    // Update is called once per frame
16    void Update()
17    {
18        transform.position += (obVelocity * Time.deltaTime);
19    }
20 }
21
22
```

Save the script, return to the **Unity** editor to observe the result: the object moves linearly, maintaining a speed of one unit of the scene per second.

In conclusion: **to move an object at a certain speed** (expressed as **scene units per second**), set this **speed** in a variable and, inside the *Update* function, multiply it by *Time.deltaTime*, as I showed you earlier.

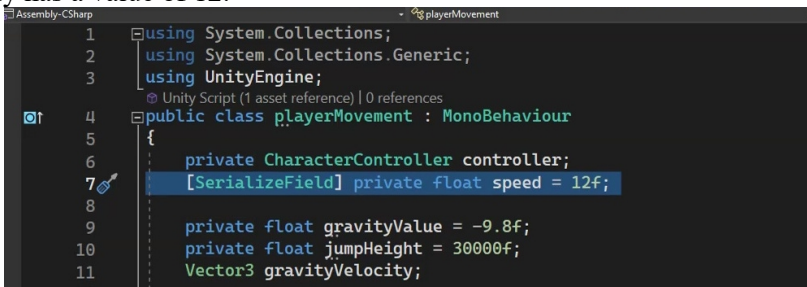
Now let's look at another example.

In the scene I'm using now, the **Main Camera** has been given a movement script that detects the arrow key presses (using *Input.GetAxis* for *Horizontal* and *Vertical*).

The values of **Horizontal** and **Vertical** range from -1 to 1 for both, so if I directly multiplied these values by **Time.deltaTime** in the **Update** function of the script, I would achieve a displacement of one unit in the scene per second, but this solution does not provide the ability to **adjust the speed**.

To solve this problem, the simplest thing to do is **define a private variable** with the **Serialized** modifier, which can be seen and modified in the **Inspector** when testing the game in the Editor.

In the example I'm showing you on video, this variable is called "**speed**" and initially has a value of 12.



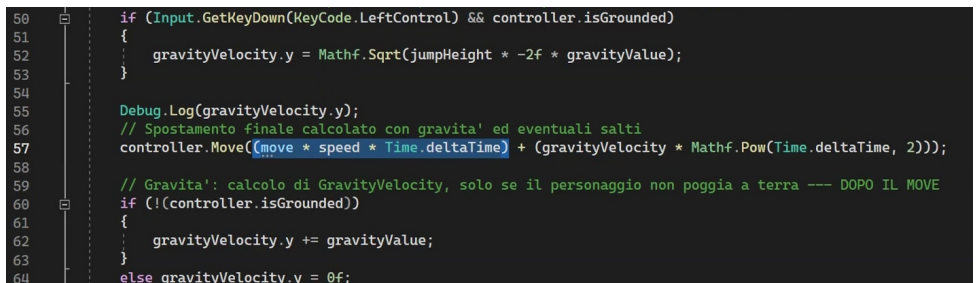
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 public class playerMovement : MonoBehaviour
5 {
6     private CharacterController controller;
7     [SerializeField] private float speed = 12f;
8
9     private float gravityValue = -9.8f;
10    private float jumpHeight = 30000f;
11    Vector3 gravityVelocity;
```

In the **Update** function, the movement is managed through **controller.Move**, because the character is associated with a **Character Controller** component that is moved using the **Move** function.

The function is called with a rather long parameter, divided into two parts.

In the first part, the displacement along the X and Z axes is calculated based on the user's input, stored in the **move** variable (which is a **Vector3**).

In the second part, the force of **gravity** is calculated when a **jump** is performed.



```
50 if (Input.GetKeyDown(KeyCode.LeftControl) && controller.isGrounded)
51 {
52     gravityVelocity.y = Mathf.Sqrt(jumpHeight * -2f * gravityValue);
53 }
54
55 Debug.Log(gravityVelocity.y);
56 // Spostamento finale calcolato con gravita' ed eventuali salti
57 controller.Move((move * speed * Time.deltaTime) + (gravityVelocity * Mathf.Pow(Time.deltaTime, 2)));
58
59 // Gravita': calcolo di GravityVelocity, solo se il personaggio non poggia a terra --- DOPO IL MOVE
60 if (!(controller.isGrounded))
61 {
62     gravityVelocity.y += gravityValue;
63 }
64 else gravityVelocity.y = 0f;
```

Time.deltaTime is used both in the calculation of movement along the axes and in the vertical movement due to gravity, but **we will ignore the second part** (which would require a longer discussion) and focus on the first.

The movement vector *move* is multiplied by *Time.deltaTime*, as seen with the cube in the first example, but it is also multiplied by the float variable *speed*.

I then switch to the Unity editor and start the game to show you how it is possible to change the character's movement speed by modifying, in real time, the value of the *Speed* parameter in the script tab.

This way, it is possible **to calibrate the speed for the start of the game**.

Since this speed is multiplied by *Time.deltaTime*, we know it will be the same on various devices, **regardless of the execution frame rate**.

As "*speed*" is a variable in the code, it will also be possible to **modify it during the game**, based on certain events.

For example, the player might drink a potion that allows them to move at twice the speed or, on the contrary, to proceed more slowly than usual.

The key point is that this speed will be consistent across different devices, as it is independent of the frame rate, thanks to *Time.deltaTime*.

In conclusion, in this video we have seen two simple examples of using *Time.deltaTime*.

Soon, I will be publishing a video about rotations using *Slerp*, where *Time.deltaTime* will be employed to achieve **constant-speed rotations**, regardless of the frame rate.

That's all for this tutorial!
See you soon!
