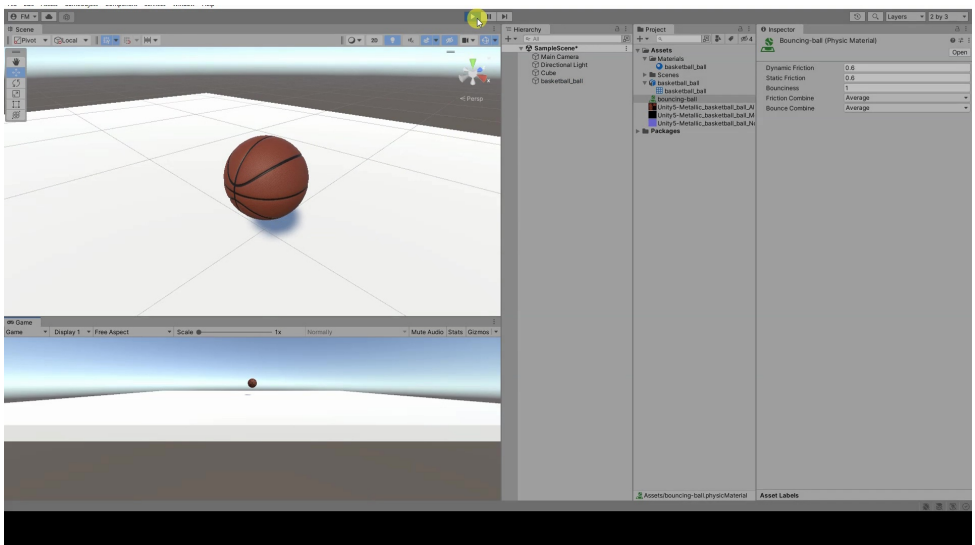


This PDF is available for free; feel free to share it!

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



Hello everyone!

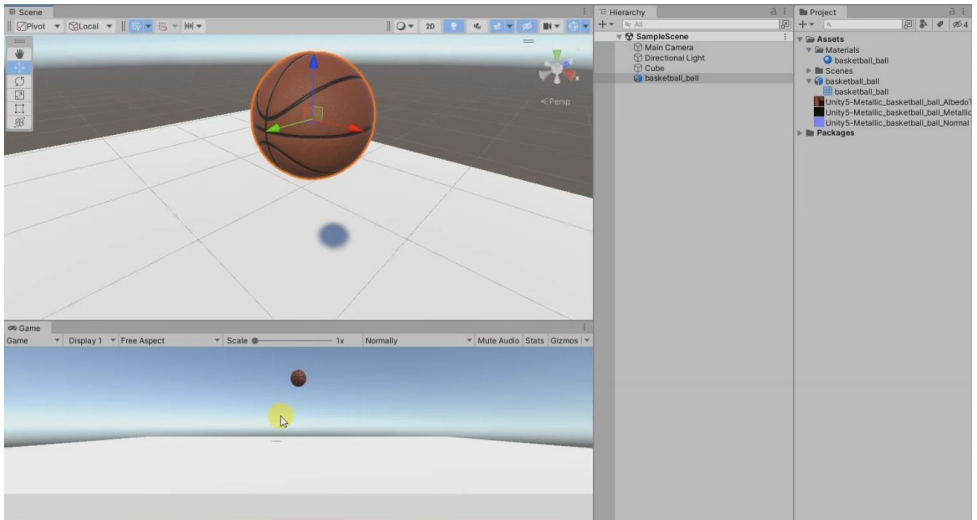
This is a basic tutorial on how to use **Rigid Bodies** and **Physics Materials** to make objects **fall and bounce** in our **Unity** projects.

Unity provides a physics simulation engine, but in order to access these features, it is necessary to use a couple of components (such as **Rigid Body** and **Colliders**) and other elements.

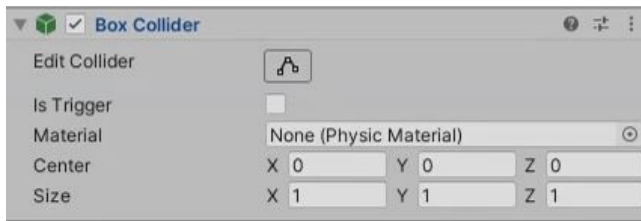
This is because physical simulation is computationally expensive, so by default, objects do NOT participate in it. It's up to us to specify if and how an object should behave physically.

Okay, let's get started!

I am setting up a simple scene in Unity with a **Cube** as the floor and a **basketball** model.

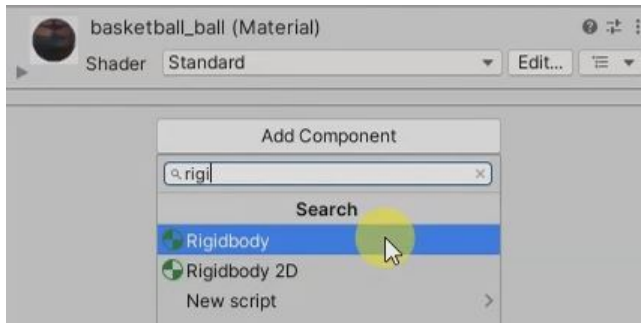


We immediately notice that the Cube object, which is a native Unity object, already has a **Box Collider** component, while my 3D model, which I created from an external file, does not have any **Collider component** of any kind. We'll come back to this later.

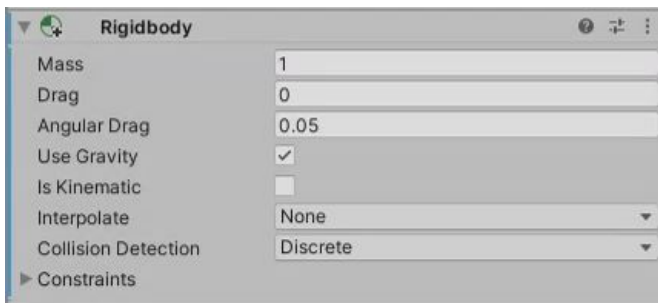


I positioned the **basketball** model right above the **Cube**, at a certain distance, but when I press the **Play** button, I immediately notice that the ball... does not fall.

To make the **ball** participate in the physical simulation of the scene, we just need to provide it with a **Rigid Body component**, which can be easily added in the object's **Inspector**.



In the component's tab, we notice some interesting features; among them, the **mass** (which we can leave as is for this example) and the "**Use Gravity**" checkbox, which is selected by default, so this time we can **start the game** and notice that, in fact, **the ball falls...**

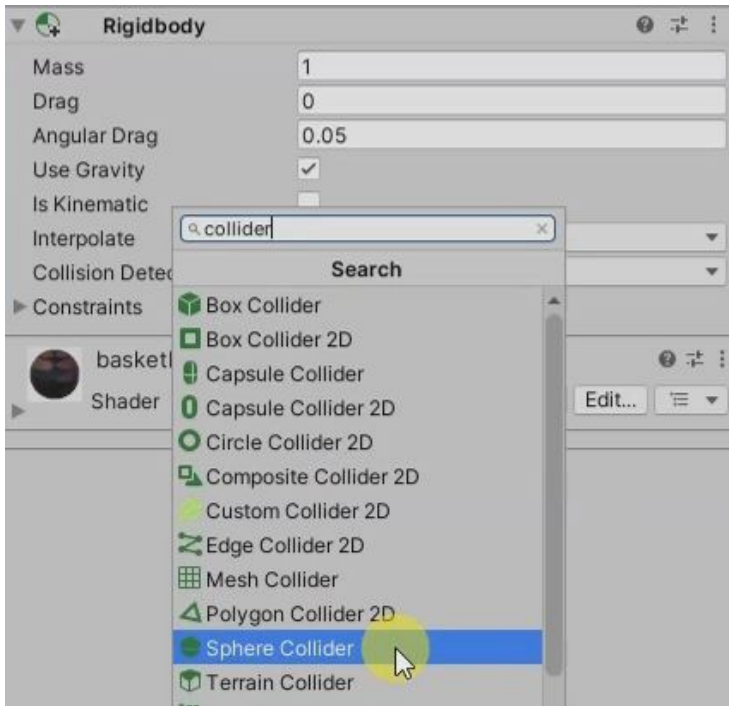


... but passes **through the underlying cube**, falling into the void!

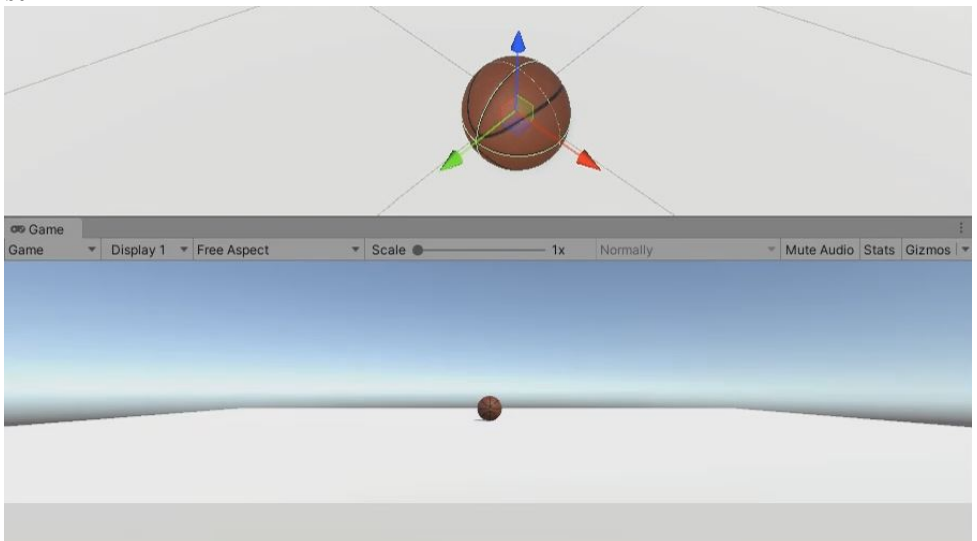
You might think of adding a Rigid Body component, possibly without gravity, to the underlying Cube object as well, to make both objects participate in the physical simulation... however, we can immediately verify that this assumption does not solve the problem (so, remove the Rigid Body from the cube, to avoid confusion).

Simply put, the ball passes through the cube... yet the **Cube has a Box Collider**, so it should detect the **collision...**

The fact is that **it's the basketball model that does NOT have its own Collider**, so we need to provide it with one in the **Inspector**. Since the basketball has a spherical shape, we choose a **Sphere Collider** for this object.



Starting the simulation now, we will see the ball fall onto the Cube and remain **still** on it.



So, keep in mind that even the Cube (and, in general, any scene object that will interact with the ball) must have its own **Collider**!

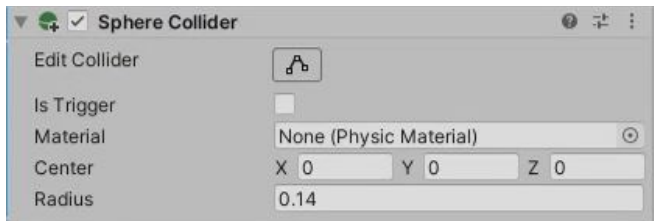
So, to recap:

- in order **to interact** with each other, **two scene objects must both have Colliders**, otherwise they will behave like ghosts;
- furthermore, in order to participate in the **physical simulation** (and thus be affected by **gravity and other forces**), an object must also have a **Rigid Body component**.

However, the result achieved so far is **not satisfactory: the ball falls and impacts the underlying cube, but it does not bounce!**

The properties of the **object's material**, however, should not be assigned to the Rigid Body, but **to the Collider**: these properties concern ANY scene object, not just those that participate in the physical simulation (i.e., Rigid Bodies), so we need to set them in the **Collider tab**.

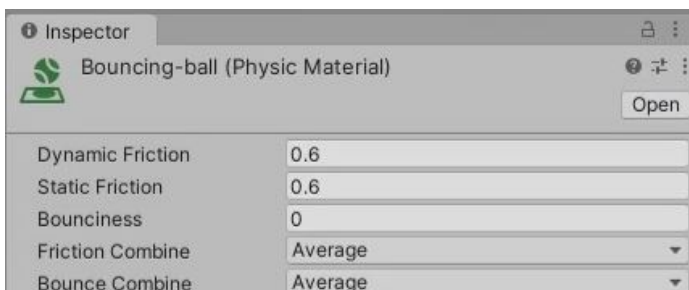
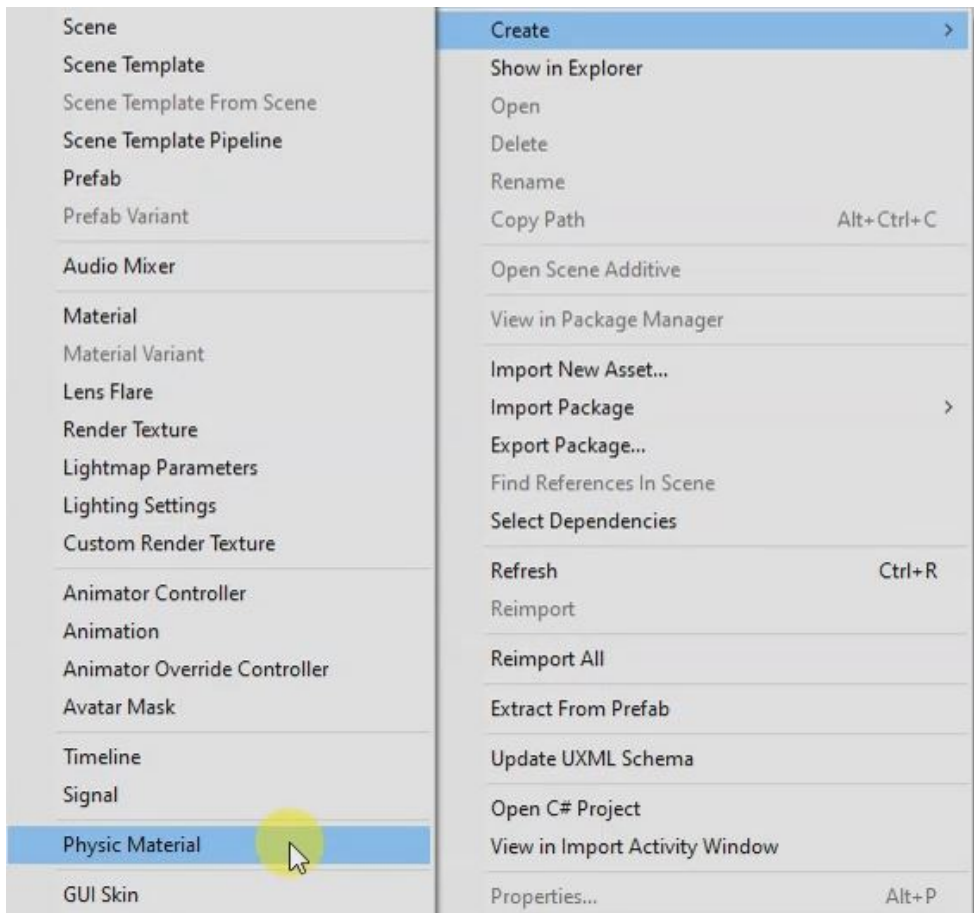
In this tab, we find a field called **Material**, which is set to **None** by default but suggests that we need to provide it with a "**Physic Material**" object.



So let's add a **Physic Material** object to the **Project**; attention: "**Physic Material**", not simply "**Material**", which instead concerns the visual aspect of an object.

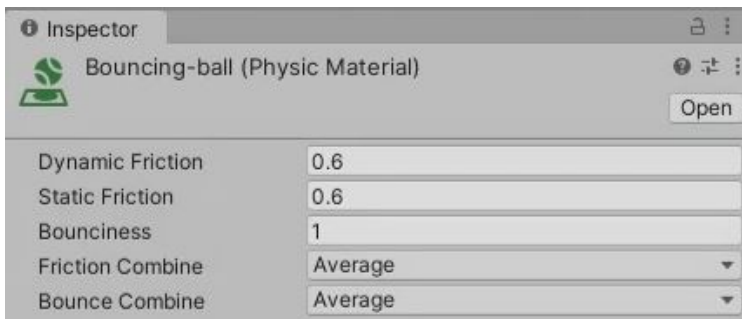
We then assign this new **Physic Material** to the basketball's **Sphere Collider**, press **Play**, and... notice that the ball **does not bounce**.

This is because, by default, the **bounciness** of a new **Physic Material** is set to **0**, as we can see by selecting such object and observing its few properties in the **Inspector**.



The **Physic Material** has two main properties: "**Bounciness**" and "**Friction**," which can be adjusted to create a more or less realistic bouncing behavior; for example, increasing the "**Bounciness**" can increase the force of the bounce, while increasing friction can decrease the distance traveled after each bounce.

So let's try setting **Bounciness** to **1** (maximum value for this field) and start the game: this time, the ball bounces a couple of times before stopping!



The behavior is realistic, but **how can we make the ball bounce more?**

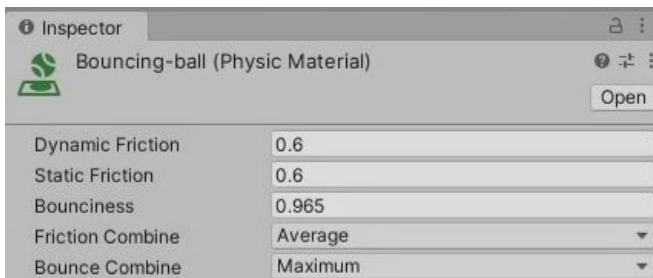
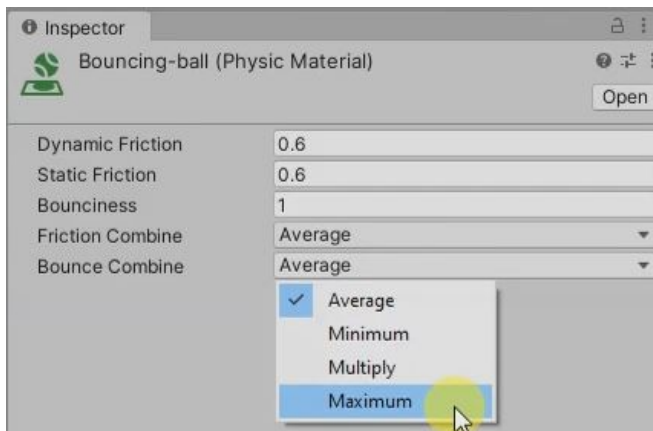
We can act, for example, on the **Friction** parameters and, above all, on the two **Combine** parameters: these parameters determine **how to combine the physical parameters of two objects** when they come into contact.

The default value for **Bounce Combine** is **Average**, so it will average between the basketball (for which we set **Bounciness** to 1) and the underlying cube, which does not have a *Physics Material* and will therefore be seen by Unity as 0, so **the final Bounce value will actually be 0.5**.

If we change **Bounce Combine** to **Maximum**, this time the actual **Bounciness** in play will be 1 and the ball will bounce much more...

... but now the problem is that it will keep bounce **more!**

It's actually a strange behavior; apparently, to maintain **constant bounces**, the **Bounciness value must be 0.965...** don't ask me why.



Obviously, in simulating bounces, **the surface is also important**: so we need to provide a **Physic Material** to the Cube (but the same goes for floors, walls, and other objects made of different materials and therefore with different physical properties) and set its **Bounciness** appropriately...

... but I leave this to you: try various combinations, with different values for **Bounciness**, **Friction**, and **Combine**!

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