2D Physics

Sebastian Grygorczuk - STEM Institute at CCNY

Today's Agenda

- Look at the Camera Component
- Learn about 2D Physics Components: Colliders, Ridigodies, & Effectors
- Create and modify PreFabs

Video

Alongside the slide, you can watch the accompanying video, which covers the same topics in a less detailed manner.

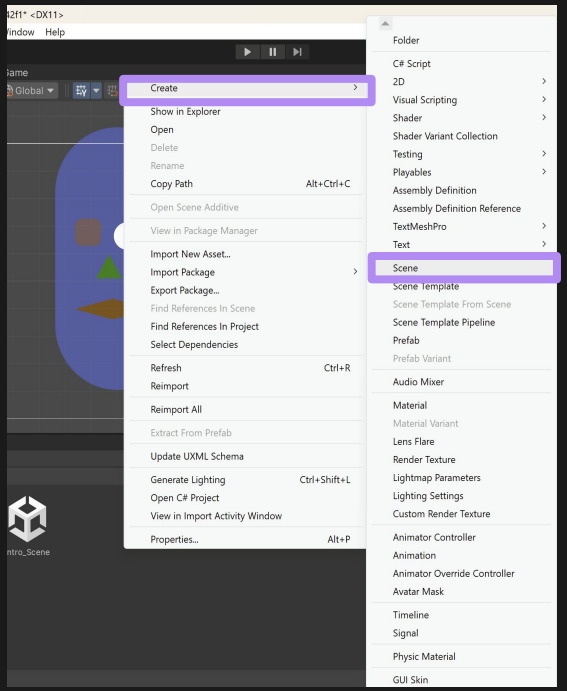
Set Up

Create a New Scene

Let's start by creating a new scene that will hold our physics-based game objects.

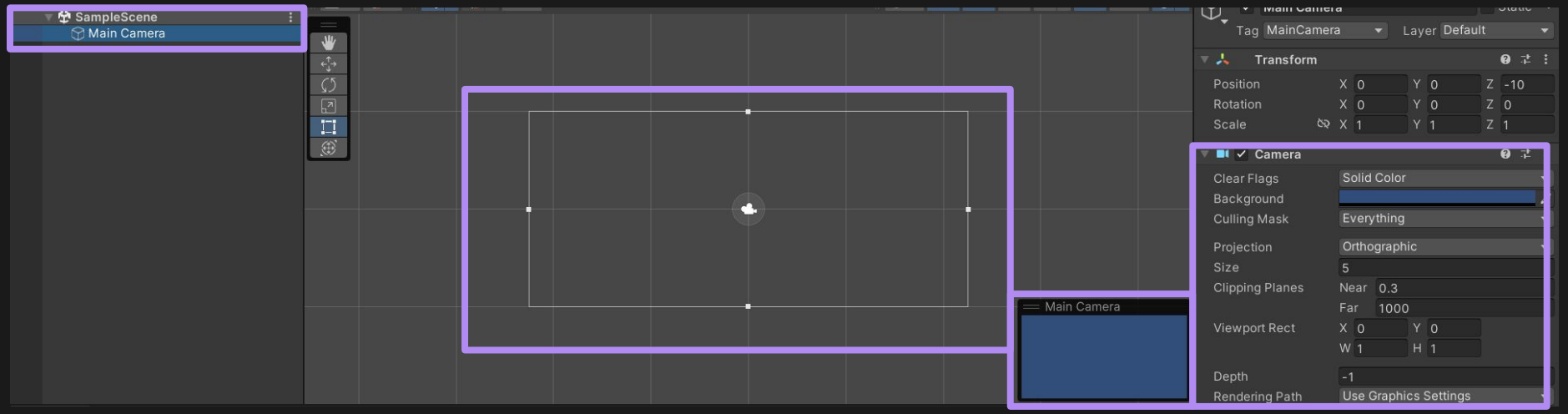
1. Right-click inside the **Project View** and go to **Create > Scene**.
2. Name the scene **Physics_2D** and double-click on it to open.

You should now see an empty scene with just a **Main Camera** game object.



Camera

Camera



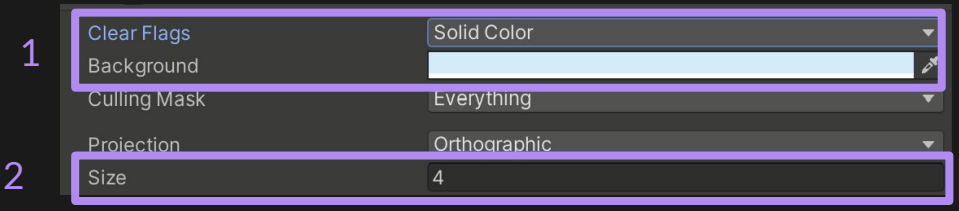
Since we'll be working in a large space, we may need to adjust the camera's position and settings. Whenever you create a new scene, Unity automatically includes a **Main Camera** with a **Camera Component** attached.

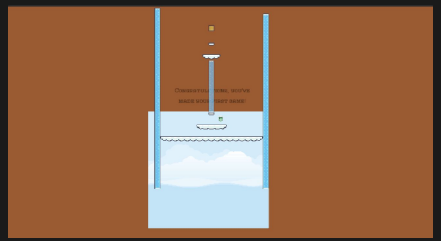
In the **Scene View**, the camera is represented by a white rectangle, and the **Game View** preview (in the bottom right) shows what the camera captures.

Color and Size

The camera has two key properties that control visibility:

1. **Clear Flags** – Defines the background type. We'll use **Solid Color**, determined by the **Background** setting.
2. **Size** – Controls the zoom level. A larger number increases visibility (zooming out), while a smaller number decreases it (zooming in). Adjusting the size is important for presenting the game world properly.

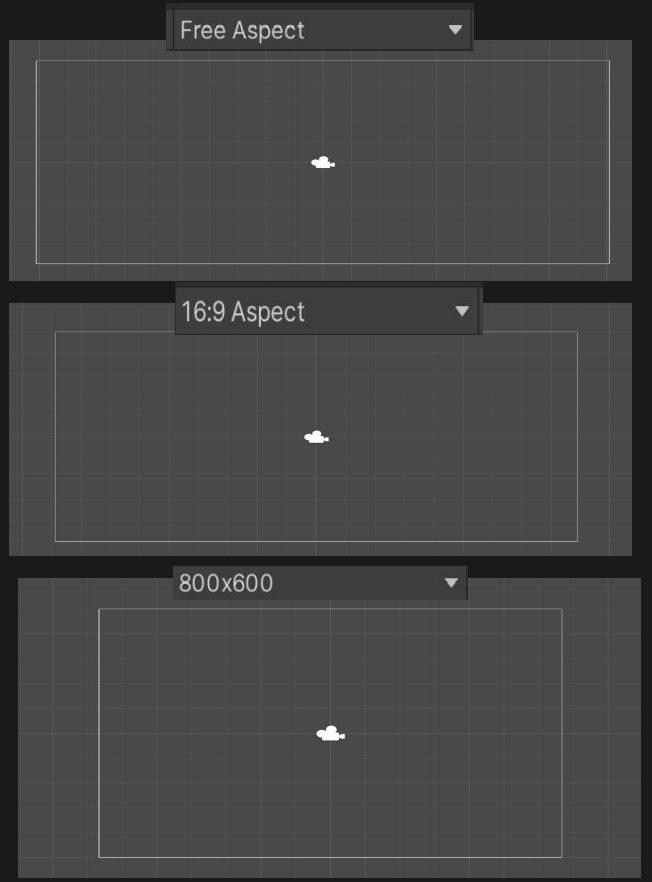




Aspect Ratio

The camera's width and height depend on the **Aspect Ratio** set in the **Game View**. Most computers use a **16:9 aspect ratio**, meaning different resolutions (e.g., **3840x2160, 1920x1080, 1280x720**) will display the game consistently.

For mobile devices, the ratio is typically **9:16**, so it's crucial to adjust the camera settings accordingly. Since we are making a PC game, we'll stick with **16:9**.



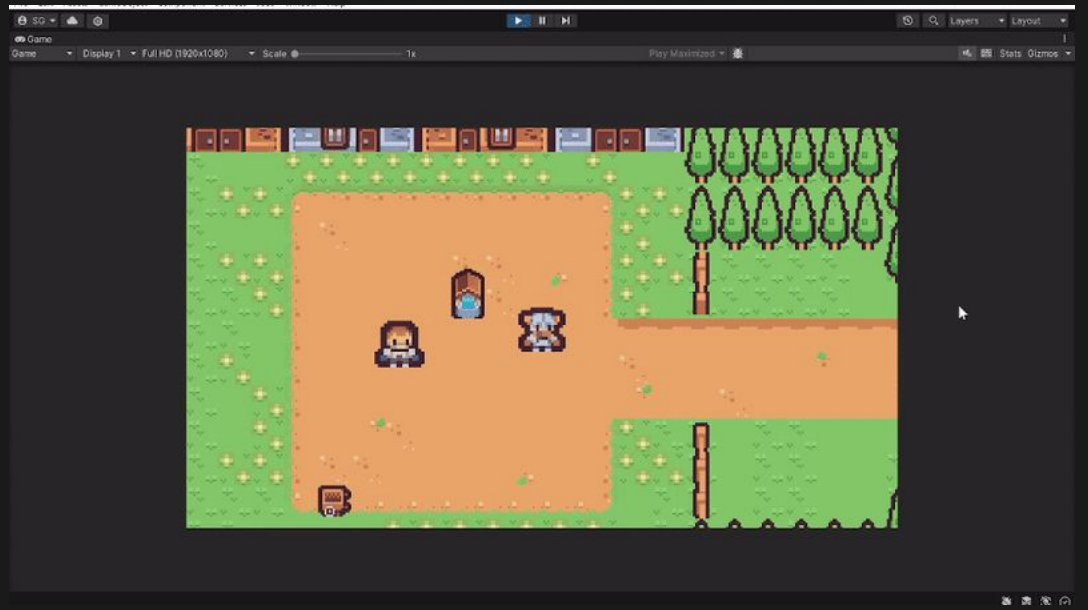
Colliders

Colliders

Colliders are components that give physical properties to objects. So far, our objects have only had **Sprite Renderer** components, meaning they are visible but lack physical interaction. Without colliders, they behave like holograms—visible but intangible.

Once two objects have colliders, they interact physically.

For example, a player cannot walk past a house if both objects have colliders, while they can walk on a floor that only has a sprite.



Adding A Collider Component

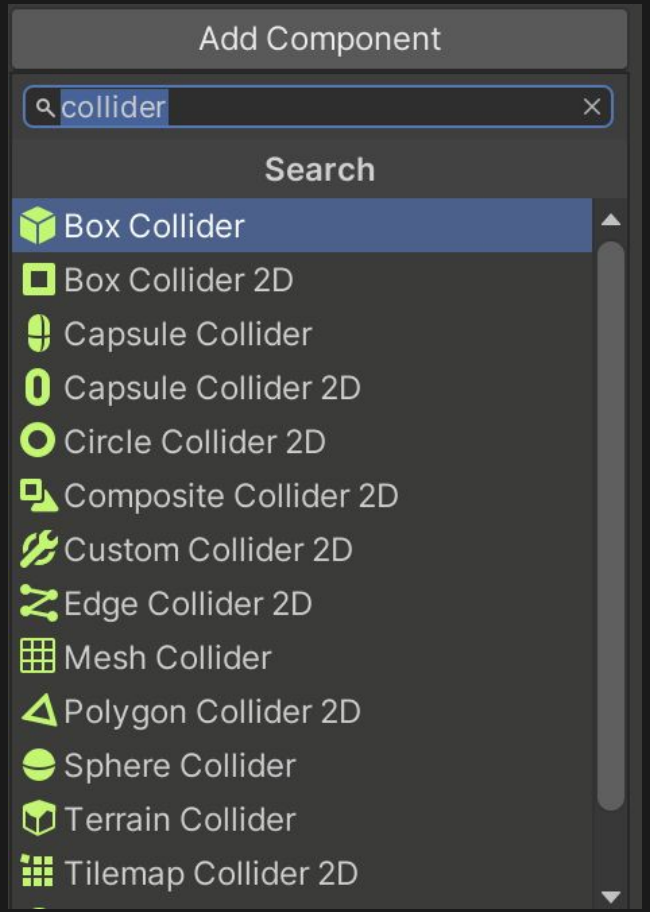
When we go to the **Inspector** of an object and click **Add Component**, we can search for **Colliders**. You will see that there are many different types, but for now, we will focus on three:

- **Box Collider 2D**
- **Circle Collider 2D**
- **Polygon Collider 2D**

Important: Always choose the 2D variant.

A **Box Collider** (without "2D") is a **3D collider** meant for **3D objects**. A **Box Collider 2D** is designed specifically for **2D games**.

If you accidentally use a **3D Collider** in a **2D game**, it **will not register collisions** properly.



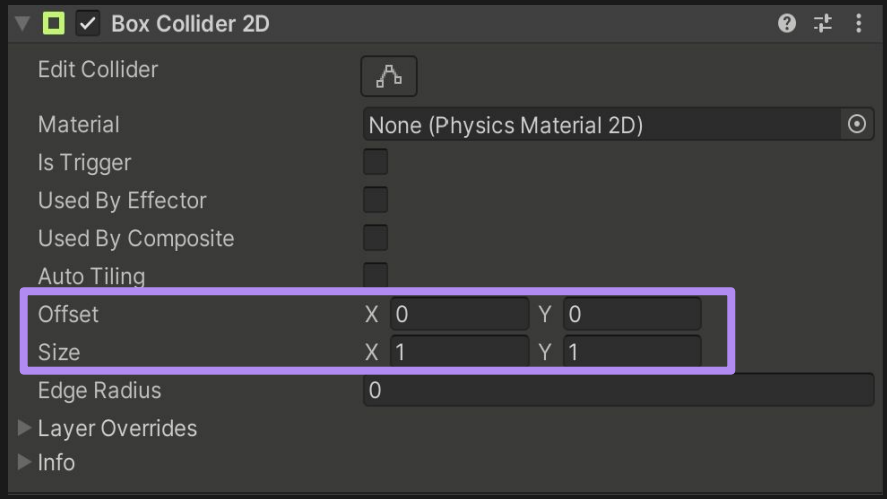
Box Collider 2D

The **Box Collider 2D** is the most basic and commonly used collider in a 2D game. Most objects in your game will use it.

When you add a **Box Collider 2D** to a game object, a **green outline** will appear around it. This outline represents the object's **physical area**. If another object with a collider comes into contact with it, a collision will occur. For example, if a house has a **Box Collider 2D**, the player will not be able to walk through it because their collider will detect the collision.

The collider's area is defined by two main properties:

- **Size (X, Y):** Determines the width and height of the collider.
- **Offset:** Controls how far the collider is positioned from the object's center. By default, the collider is centered on the object.



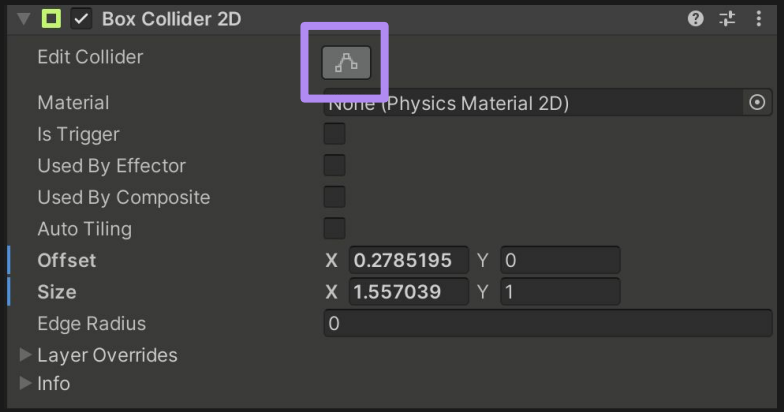
Adjusting the Size

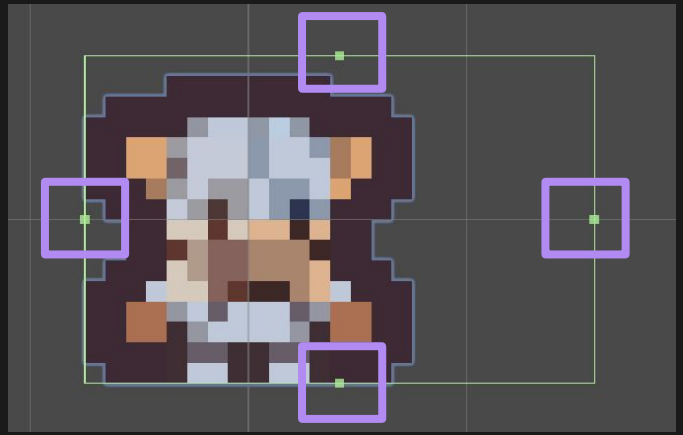
There will be times when you need to adjust a collider directly in the **Scene View** to better fit your object.

To do this:

1. Select the object with the collider.
2. In the **Inspector**, click the **Edit Collider** button.
3. Small handles (nubs) will appear on the edges of the collider.
4. Drag these handles to resize or reshape the collider in any direction.

Any changes you make will automatically update the **Offset** and **Size** properties in the **Inspector**.





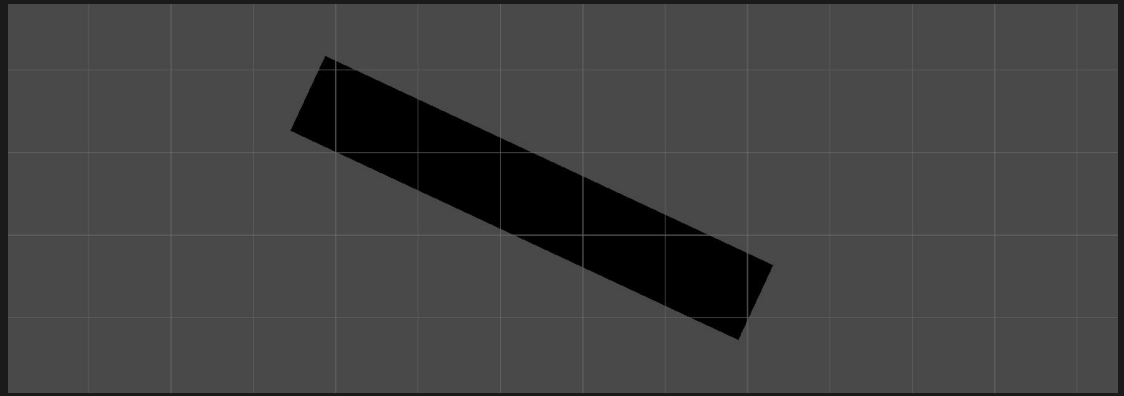
Challenge Create Box and Box Collider

Now, let's put your skills to the test!

Create a Game Object with a **Square Sprite** in the **Sprite Renderer** by going to **2D > Sprite > Square**.

Adjust the shape by **angling it slightly** and **extending it** as needed.

Add a 2D Box Collider Component to the object to enable collision detection.

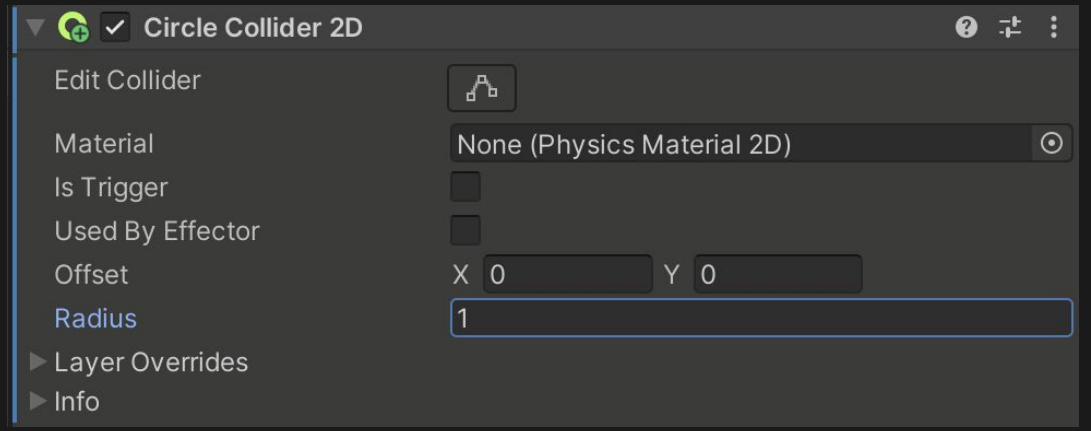


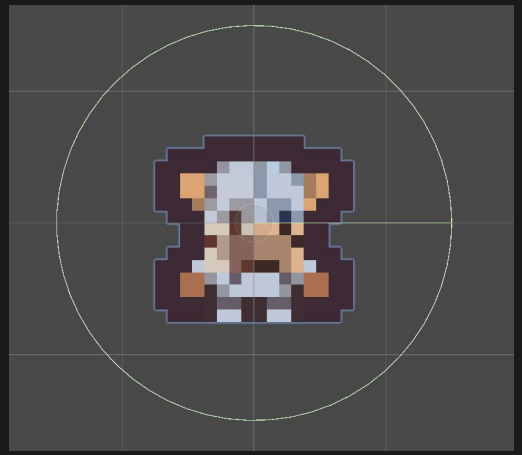
Circle Collider 2D

Another collider we will use is the **Circle Collider 2D**. It functions similarly to the **Box Collider 2D**, but instead of having **width and height**, it has a **radius** that controls the size of the circle.

When added to an object, a **green circular outline** will appear, representing its collision area. Adjusting the **radius** in the **Inspector** will change the size of the collider while keeping it centered on the object by default.

The **Offset** property can also be used to shift the collider's position if needed.



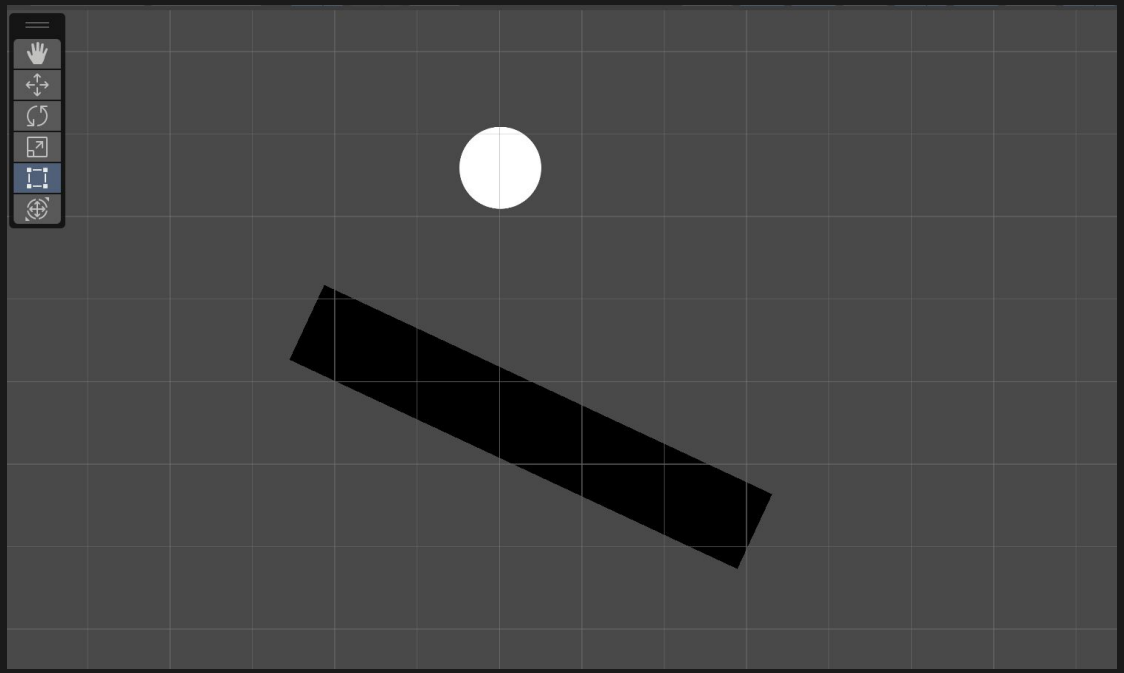


Challenge Create Circle and Circle Collider

Now, let's put your skills to the test!

Create a Game Object with a **Circle Sprite** in the **Sprite Renderer**.

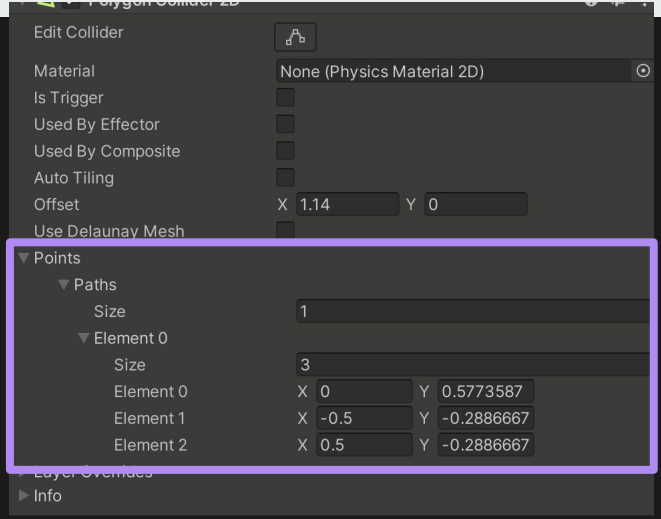
Add a 2D Circle Collider Component to the object to enable collision detection.



Polygon Collider 2D

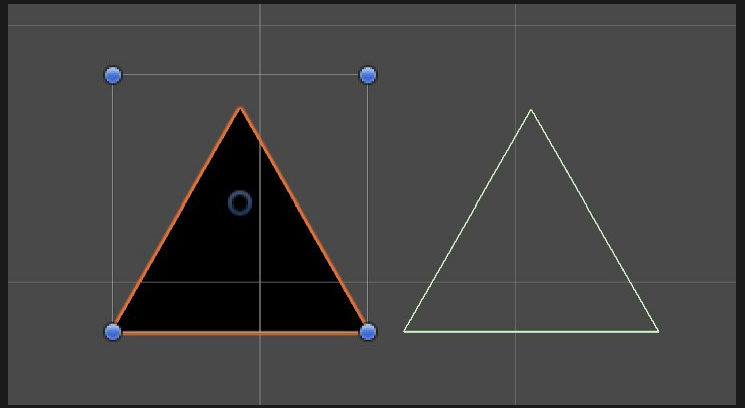
The **Polygon Collider 2D** is unique because it can take on **any shape**. When attached to a game object with a sprite, such as a **triangle**, it will automatically conform to the shape of the image.

This collider works by using a series of **Points**, which define its edges. You can adjust these points manually to create custom collision shapes.



When to Use Polygon Collider 2D

- Use it when **precise collision** is necessary, such as for irregularly shaped objects.
- Avoid overusing it, as it is more complex for the computer to process compared to **Box Collider 2D** or **Circle Collider 2D**.
- In most cases, a box or circle collider is preferred for performance efficiency.



Adding and Removing Points

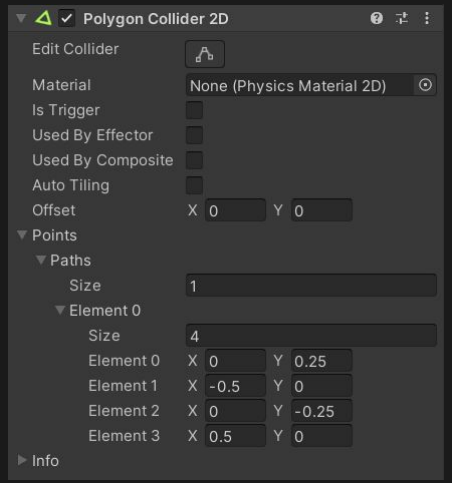
Because the **Polygon Collider 2D** can take any shape, its editing process is a bit different from other colliders.

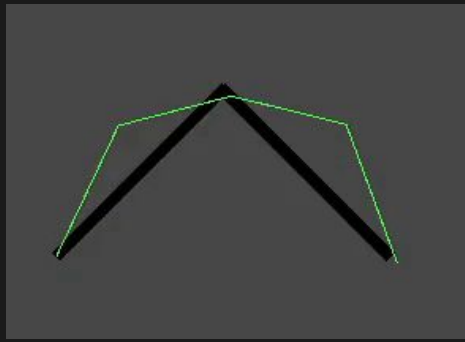
To add a point:

1. **Hover** your mouse over any of the edges or points of the collider.
2. **Drag** the edge to adjust the shape. This action will **create a new point** where two edges meet.

To remove a point:

1. Hold **Left Ctrl** while hovering over the point.
2. Click on the point to **delete it**, which will merge the two edges into one.

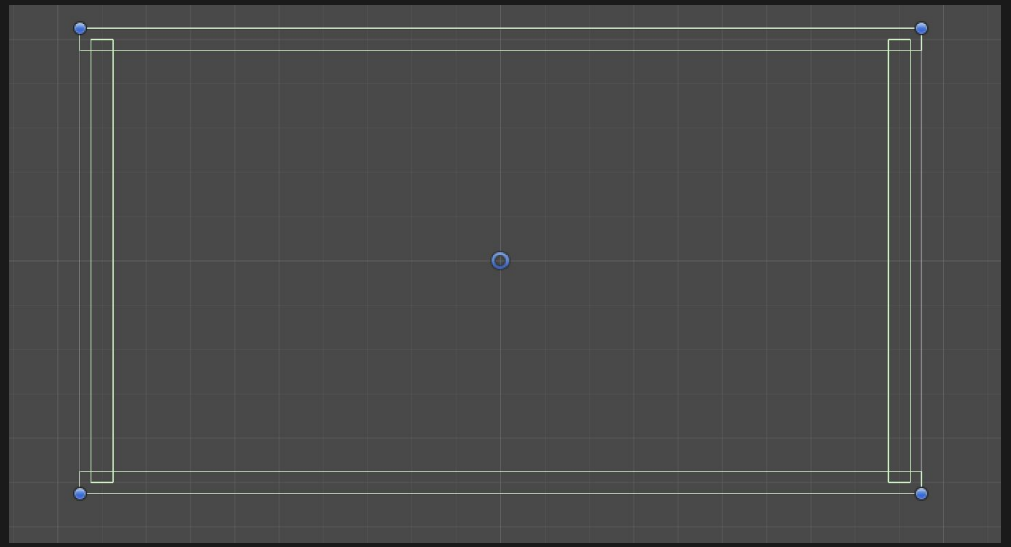




Invisible Walls

A **Game Object** doesn't need to have any visuals attached to it to perform its function. You might have encountered **invisible walls** in games before. These are essentially **colliders with no images** attached.

These invisible colliders can still serve important roles, like preventing the player from wandering off in directions they're not supposed to go. They provide physical boundaries and help shape the gameplay experience, even though they aren't visible to the player.

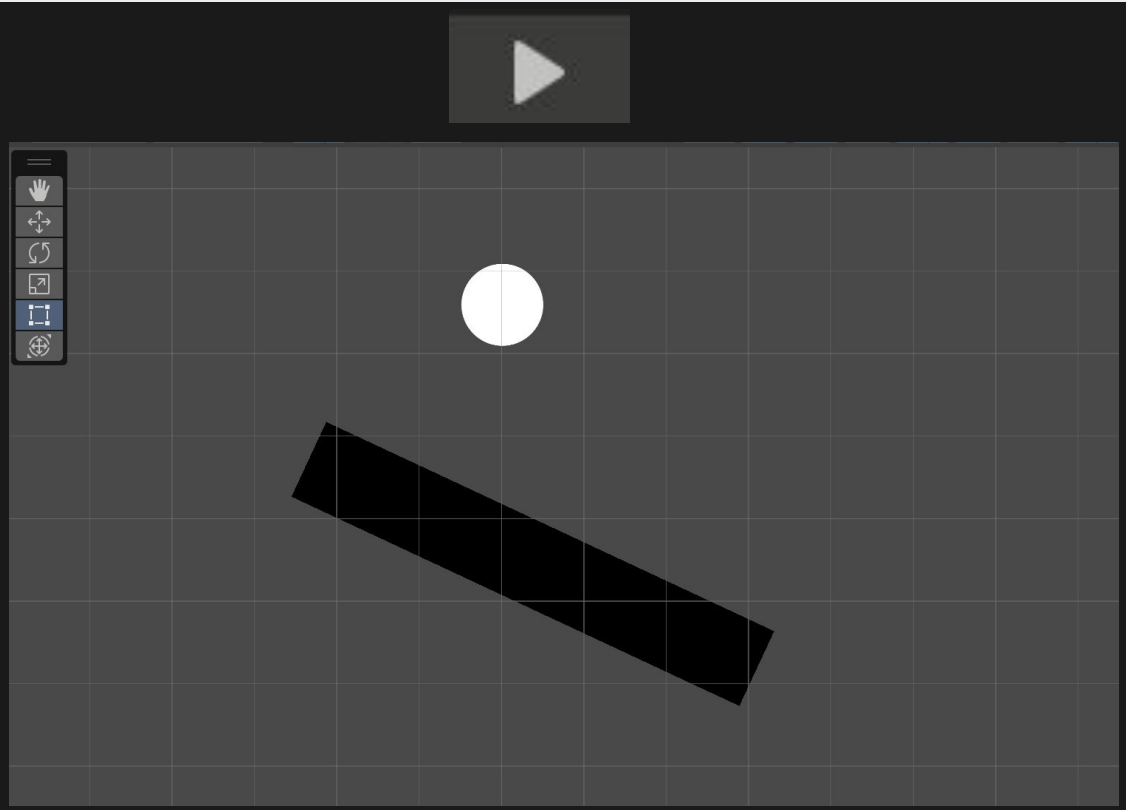


Challenge Ball Drop?

Now, let's put your skills to the test!

Click Play.

What happens?

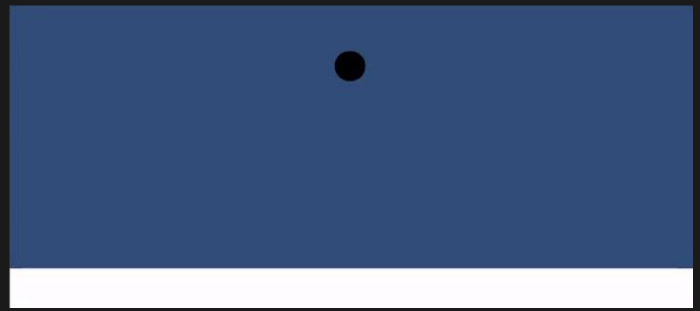


Rigidbody

2D Rigidbody

The **Rigidbody 2D** is essential for making the ball drop.

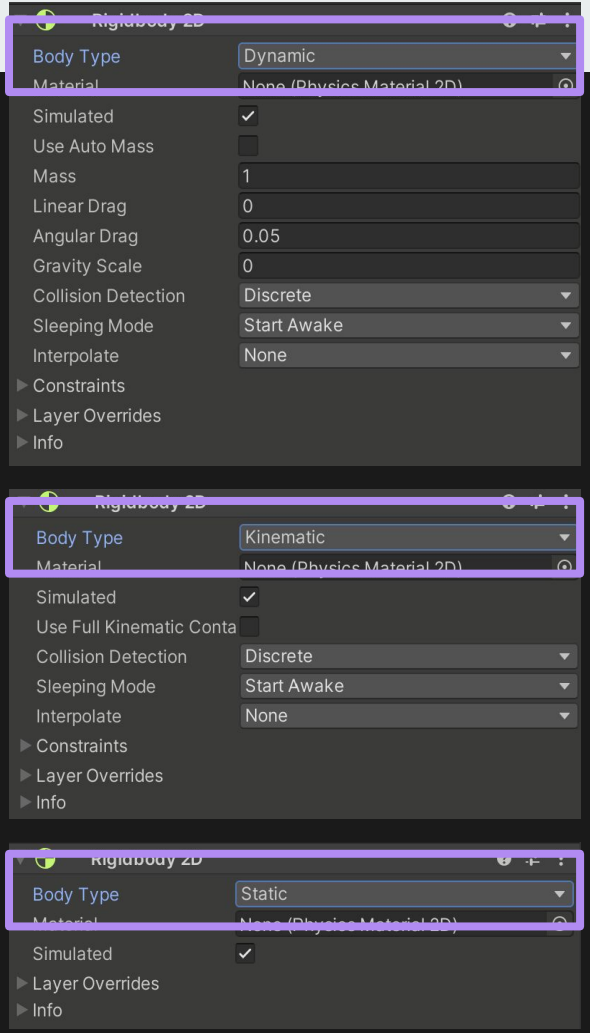
It gives a game object physics properties such as **mass, gravity, velocity, momentum**, and much more.



Rigidbody2d Body Types

There are three **Body Types** that a **Rigidbody 2D** can behave as:

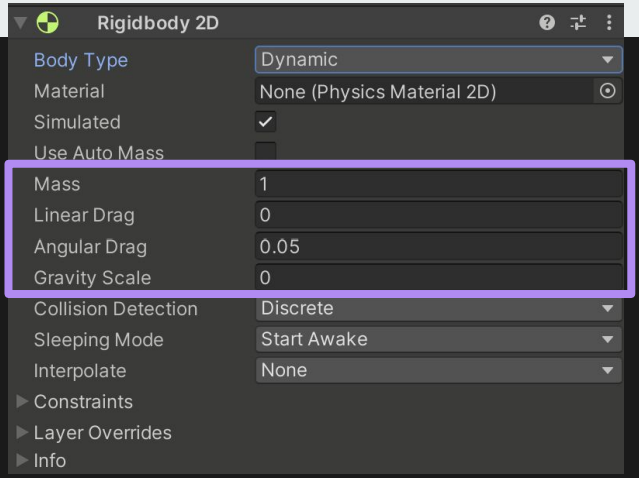
1. **Dynamic** – This is the most complex type. It reacts to other game objects touching it and moves according to the physics provided to it.
2. **Kinematic** – This type can be moved using **scripts** or **code**, but it will not be moved by collisions with other objects.
3. **Static** – This type acts like a wall. It doesn't provide any movement to the object itself, but it's useful for objects that need to be interacted with by other objects.

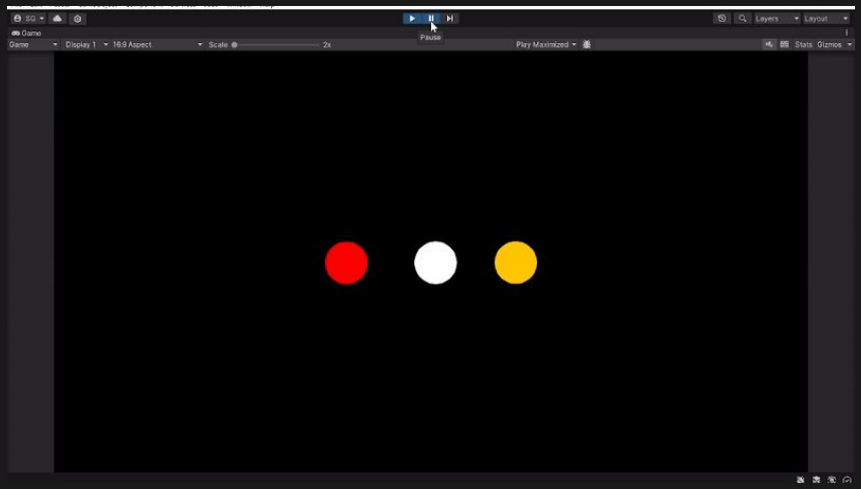


Dynamic Body Type

A **Dynamic Rigidbody** has these four variables that let you choose how the object interacts with other objects:

- **Mass** – This controls the object's density. The denser object will resist being pushed by a less dense object.
- **Linear and Angular Drag** – These control how quickly the ball loses speed as it falls. They're not really important for our class.
- **Gravity Scale** – This controls how fast and in what direction the game object falls. A positive value of 1 will make the object fall down, 0 will keep it in place, and -1 will make it move upward. This is because Unity's base gravity scale works on a negative value, meaning gravity always pulls objects down.





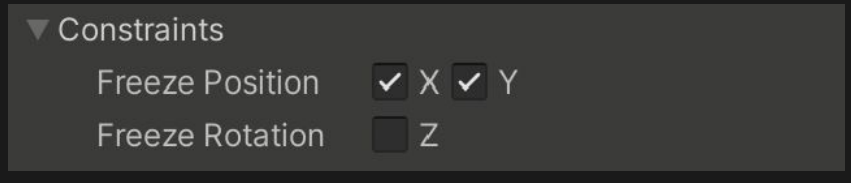
Constraints - Position

The last thing we'll look at in the **Rigidbody** is the **Constraints** tab.

In this tab, you can freeze the position and rotation of the object.

Freezing the **position** allows the object to stay still while still reacting dynamically to the objects that interact with it.

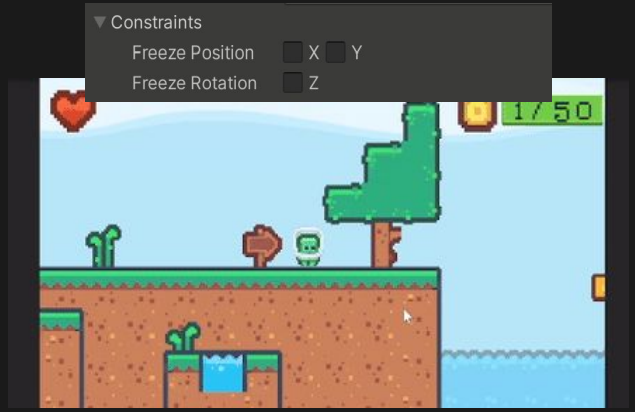
For example, in this case, the paddles spin after being touched, but their position remains fixed.

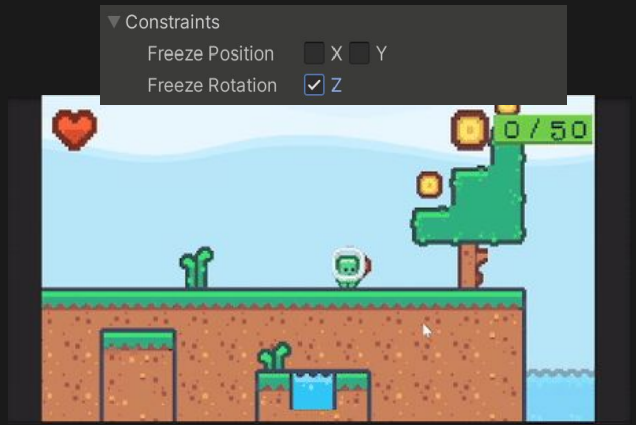


Constraints - Rotation

On the other hand, once we start building out the game, we'll freeze the rotation of our player.

We don't want them to flip over every time they touch any physical object.



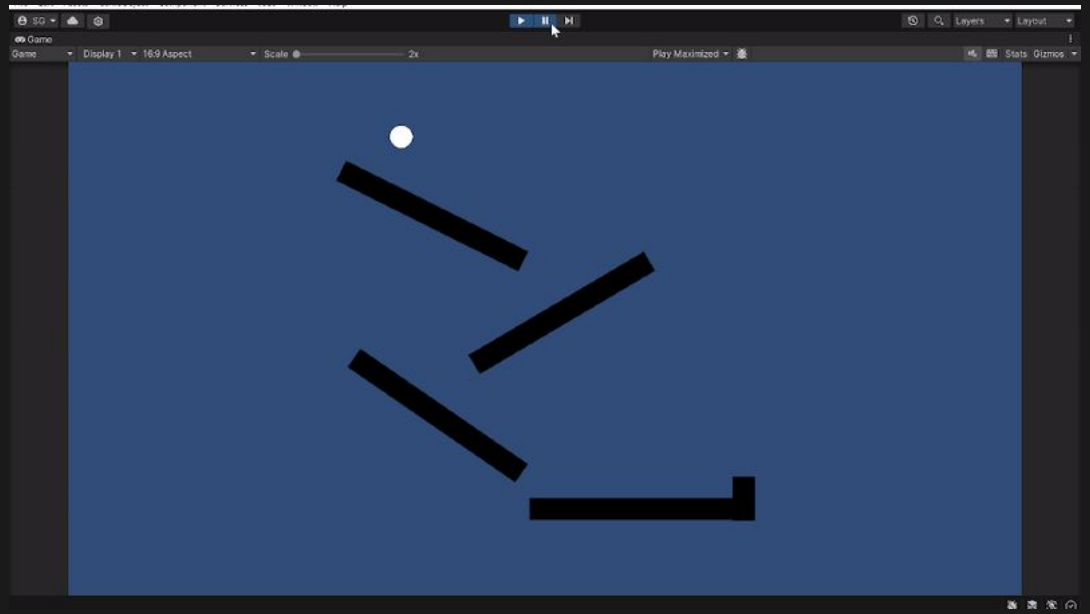


Challenge Ball Drop

Now, let's put your skills to the test!

Let’s add a Rigidbody 2D to the ball, add few more ramps and click play.

Let’s see the ball roll.



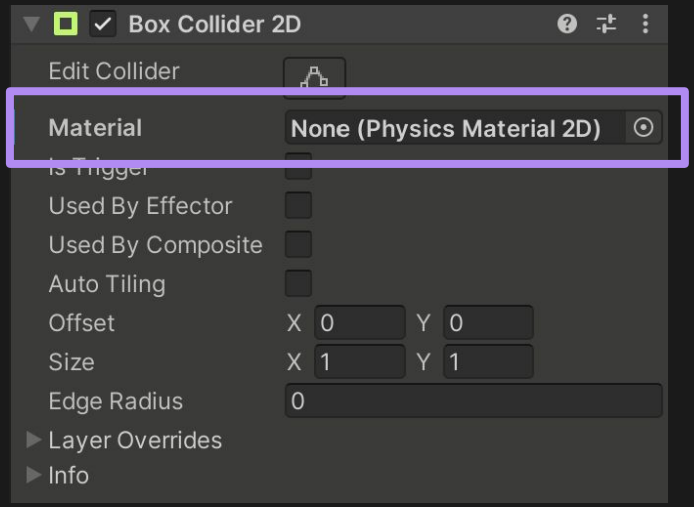
2D Physics Material

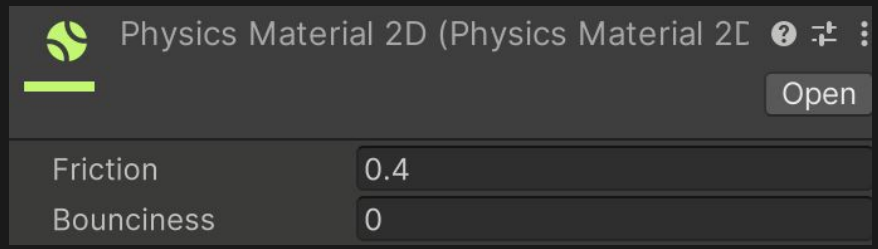
Physics 2D Material

Let's quickly go back to **Colliders**.

There was a field that we skipped called **Material**. Material refers to the **Physics Material 2D**, which defines how much friction and bounciness the object has.

If you don't input one, the default values are a friction of **0.4** and a bounciness of **0**.



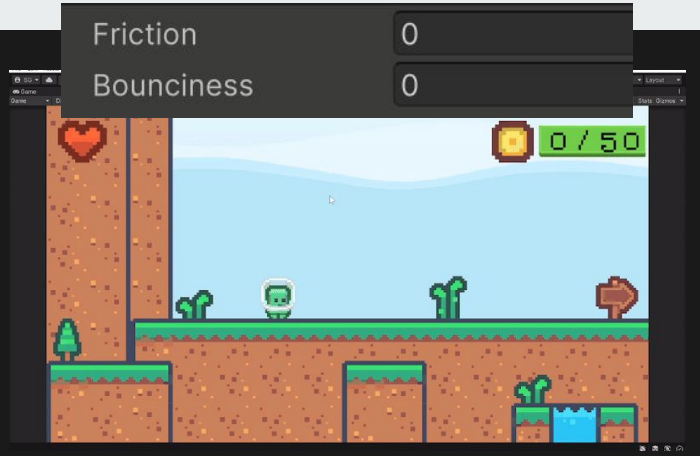


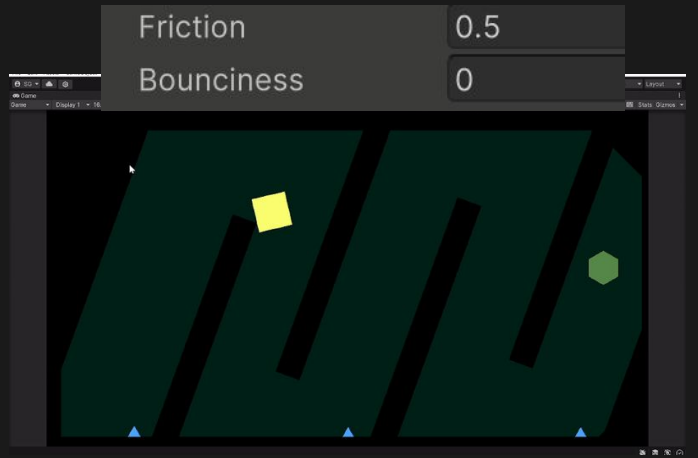
2D Physic Material - Friction

Friction is how much the velocity (both horizontal and vertical speeds) of the player loses when they collide with another collider while pressed against each other.

In **Pixel Quest**, we don't want our player sticking to walls, so we will create a **Physics Material** with **0 friction**.

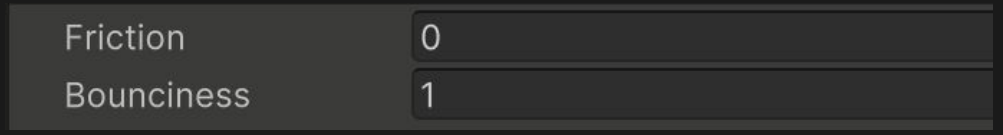
Meanwhile, in **Geo Quest**, we want the player to be able to cling to walls partially, so we'll increase the friction to **0.5**.

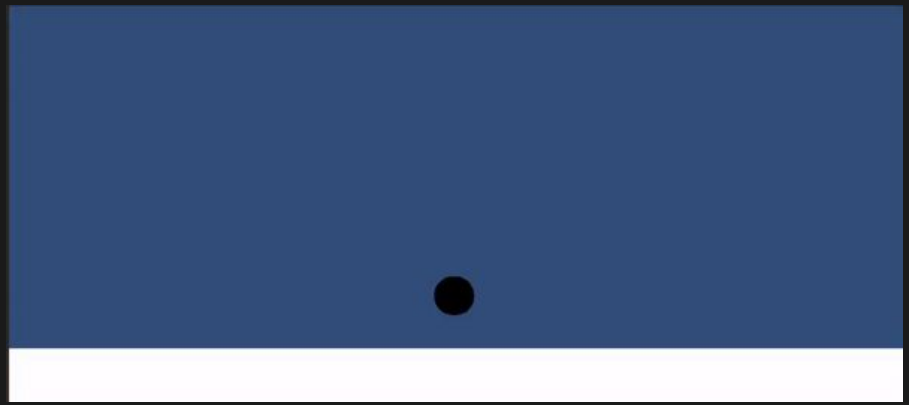




2D Physics Material - Bounciness

On the other hand, we have **Bounciness**, which controls how much the object can bounce without losing energy and allows the object to bounce off any other physical object it interacts with.

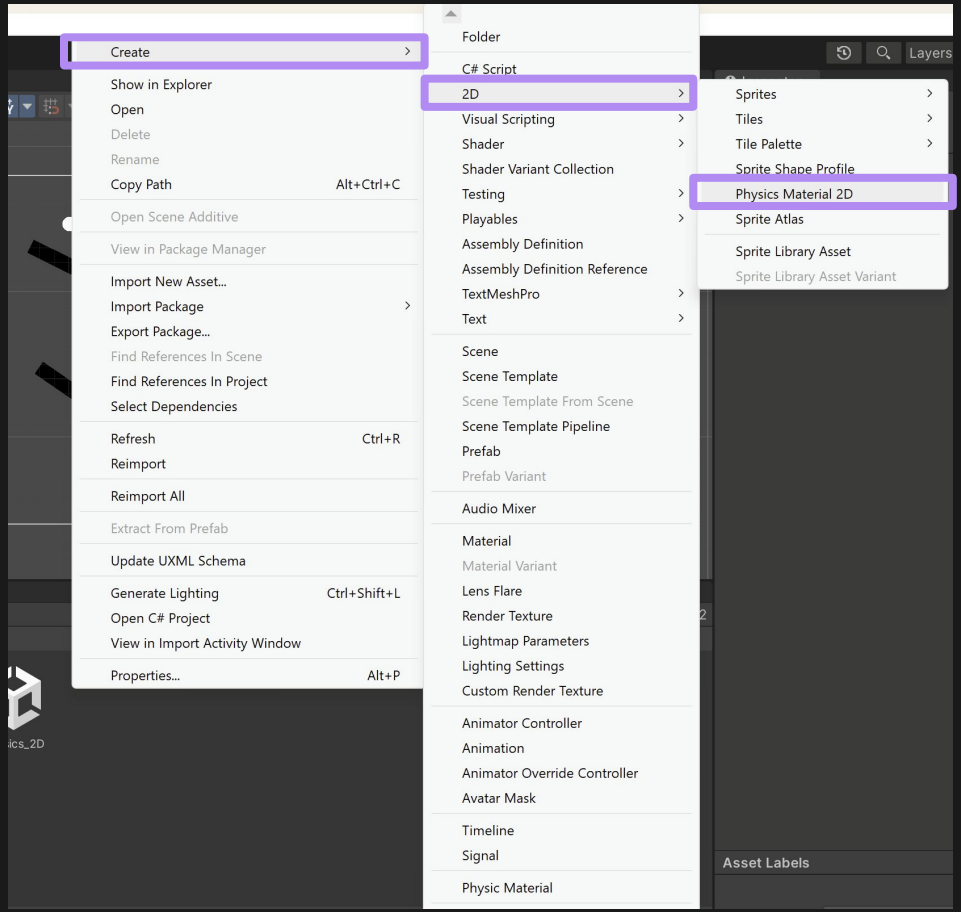




Create 2D Physic Material

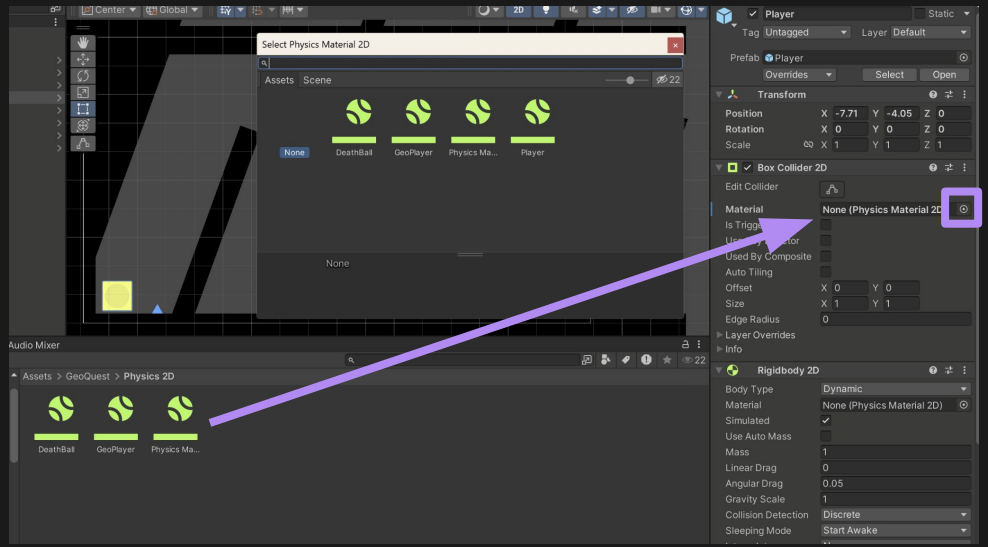
Physics 2D Material is an asset that can be added to any **2D Collider** component.

To create one, right-click in the **Project View**, go to **Create -> 2D -> Physics Material 2D**.



Attaching 2D Physic Material

To attach a **2D Physics Material**, simply drag it from the **Project View** to the slot in the **Inspector**, or click on the target next to the field and select it from the search window that pops up.



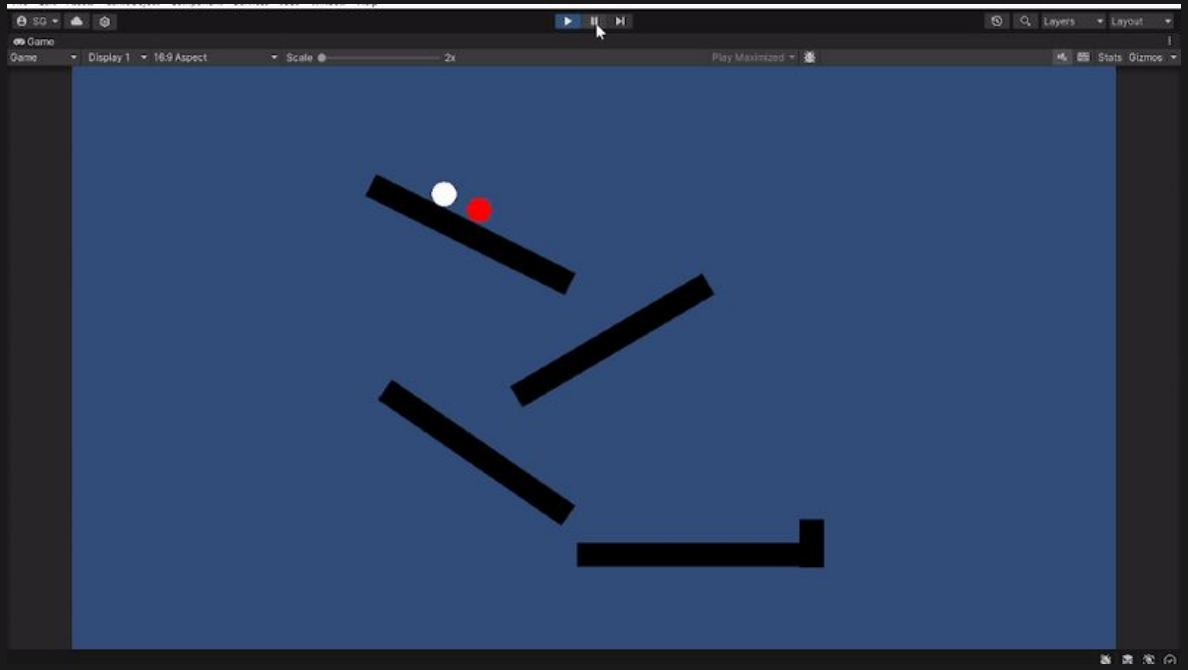
Challenge Physics Material

Now, let's put your skills to the test!

Create another ball, change the color so you have easy time telling the difference between them.

Create a 2D Physics Material, remove all the friction and give it some bounciness, I used 0.5 but feel free to experiment and click play.

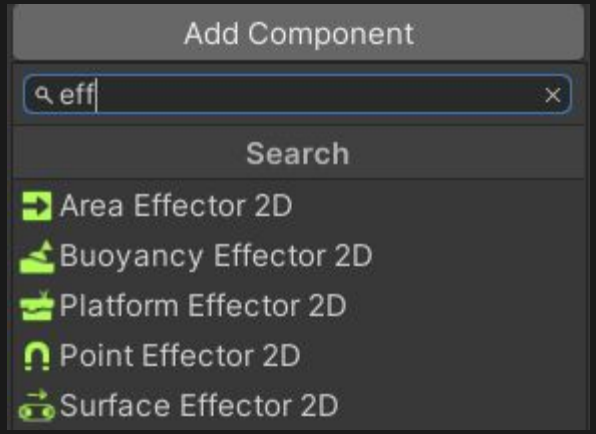
Let’s see the balls roll.



Effectors

2D Effectors

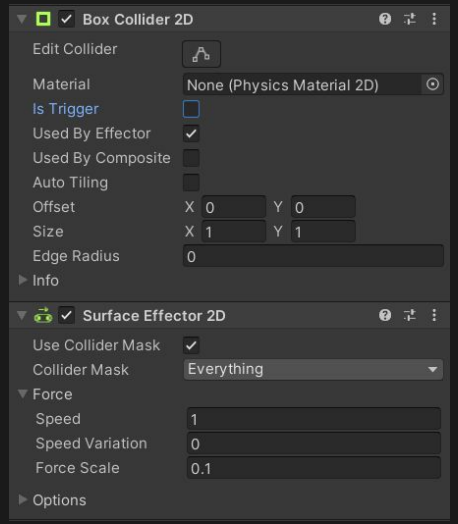
2D Colliders have a unique connection called **Effectors**. Effectors are additional physics components that continuously exert a force. There are five of them: **Surface**, **Area**, **Buoyancy**, **Platform**, and **Point**.

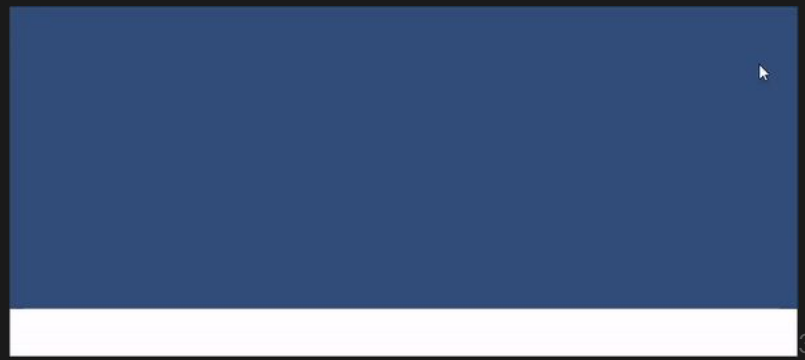


2D Effectors - Surface Effector

The **Surface Effector** gives a constant speed along its **X-axis**. This is connected to the white platform and only requires the effector to be toggled on the **Box Collider**.

- The **Speed** variable will send the object left or right, depending on whether the value is negative or positive.
- The **Force Scale** controls how fast the object accelerates toward the specified speed.
- The **Speed Variation** adds any sudden upticks that occur while accelerating toward the given speed.





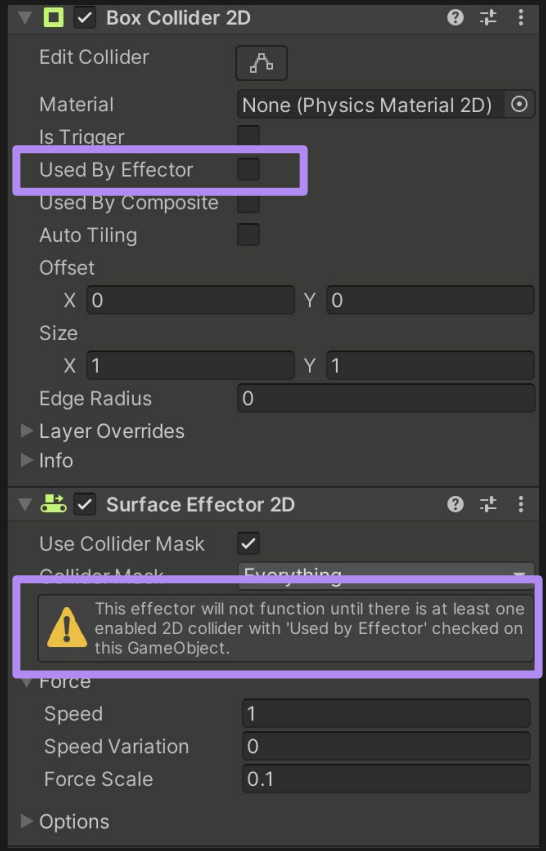
Attaching

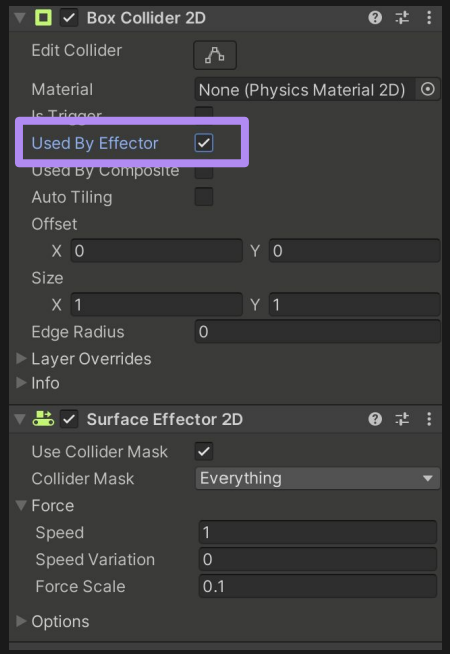
When you add an **Effector**, you'll notice that it will show a warning.

You need to specify which **Collider** is using the effector.

We haven't reached a situation where we add multiple colliders to one Game Object yet, but you can definitely do that.

To ensure the effect works correctly, make sure to check the toggle **Use By Effector** inside the collider you want to use.



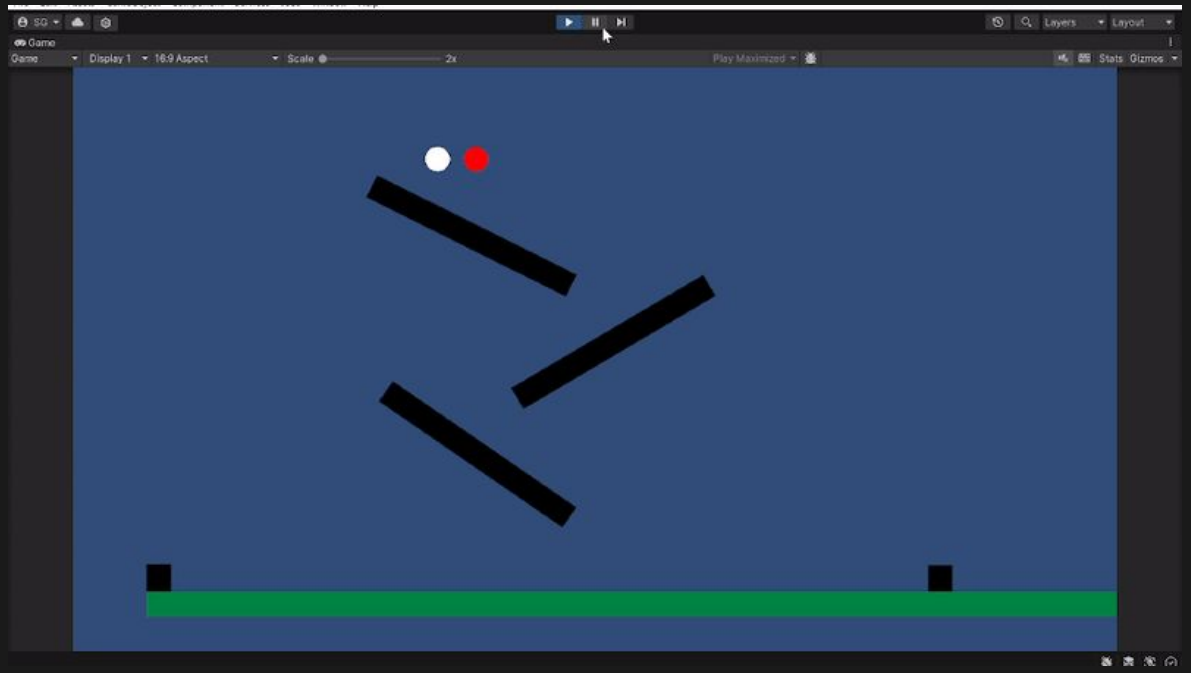


Challenge Surface Effector

Now, let's put your skills to the test!

Let’s extend the floor, change the color of it to tell difference that it behaves differently.

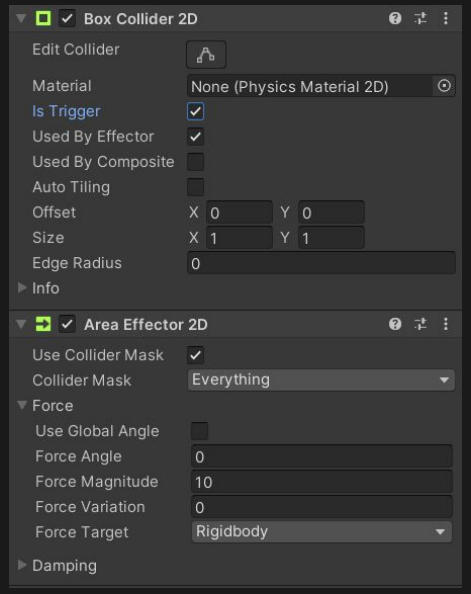
Then add a Surface Effector to it and make the speed negative so that the balls have a negative x velocity upon colluding with it.

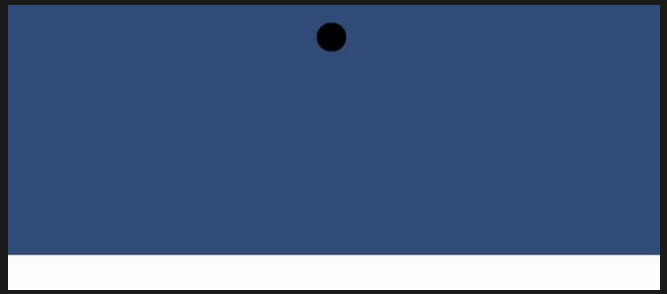


2D Effectors - Area Effector

The **Area Collider** will apply a force in a given direction to any object that enters it. This requires the **Box Collider** to have both the **Trigger** and **Effector** toggled.

- The **Force Angle** determines the direction in which the object will be sent, with **0** being along the **X-axis**.
- The **Force Magnitude** controls how much force is applied to the object.
- The **Force Variation** defines how much random variation can be added to the force. If the value is negative, it determines how much random force will be taken away.

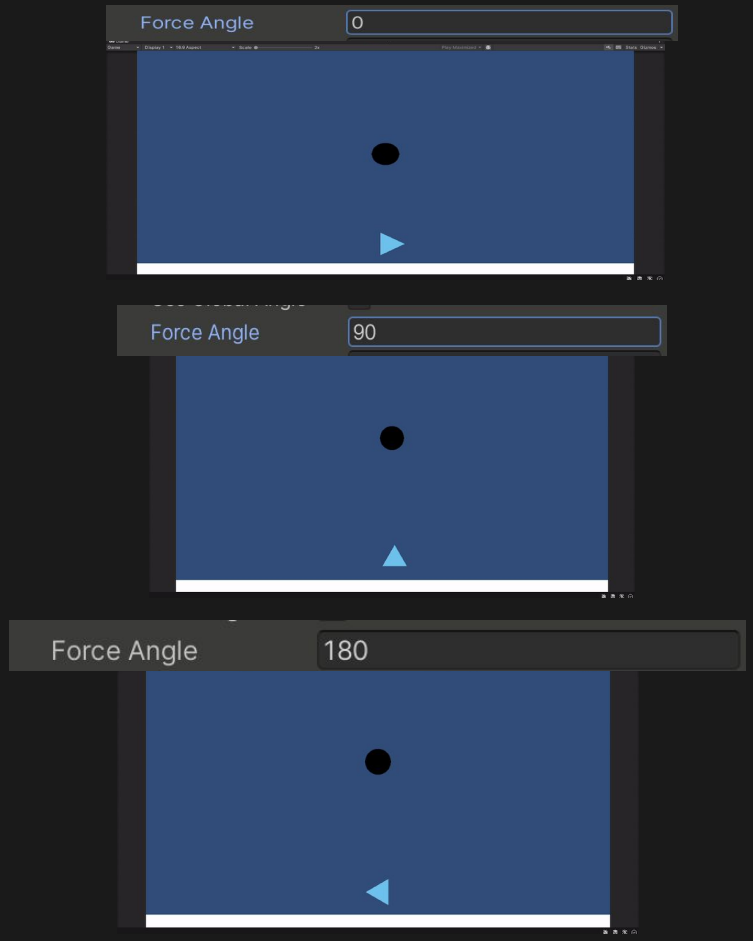




Force Angle

As mentioned on the previous slide, the **Angle** works with **0** being the positive **X direction**, and it follows the regular 360-degree rotation.

You will also notice that, depending on how long the object stays inside the area (like in the triangle), the force exerted on it will be larger.



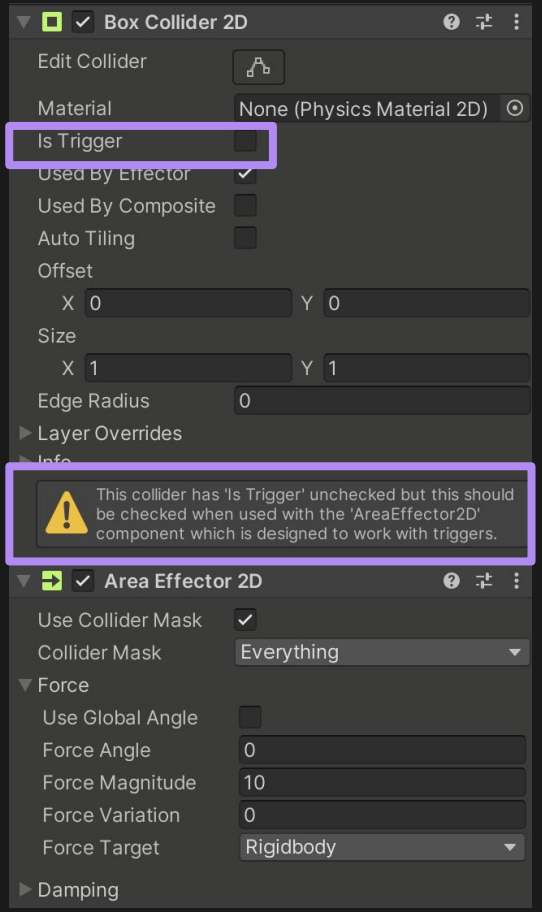
Trigger Collision

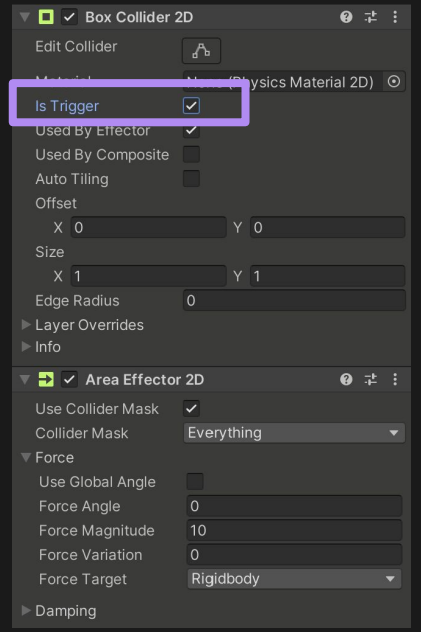
When adding an **Area Effector** and clicking the **Use By Effector** toggle, you will get a new warning inside the collider.

This warning is telling you that the collider needs to be set to **Is Trigger**. **Is Trigger** turns an object from a physical space into a sensing space.

Think of it like when you're walking into a store and there are anti-theft sensors at the doors that beep when something hasn't been paid for. When you walk past them, the unpaid item enters the collider and triggers a response, like the beeping.

We will use many **trigger colliders** to create actions between the player and objects like spikes, coins, hearts, and more.

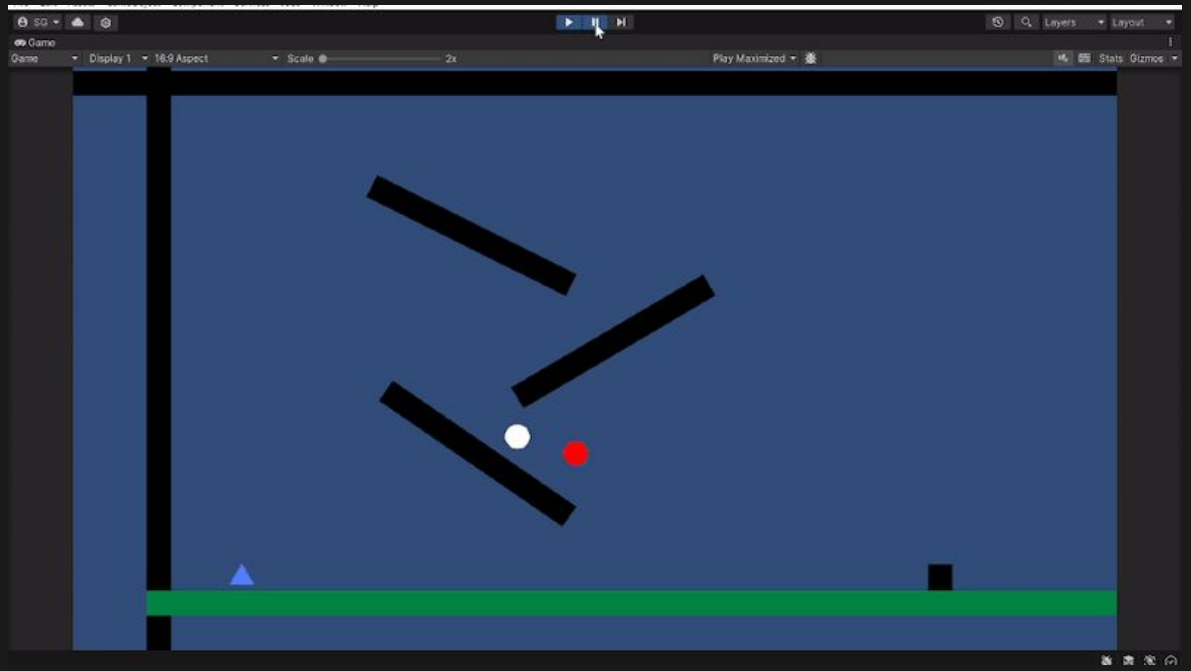




Challenge Area Effector

Now, let's put your skills to the test!

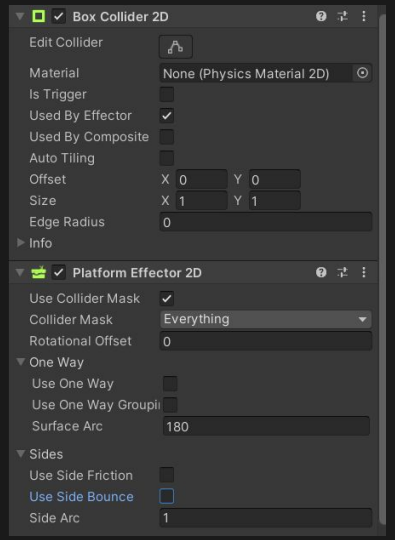
Let’s create a Game Object with Triangle Sprite, attach a Polygonal Collider and Area Effector and make it shoot the balls up into the air upon colliding with it.

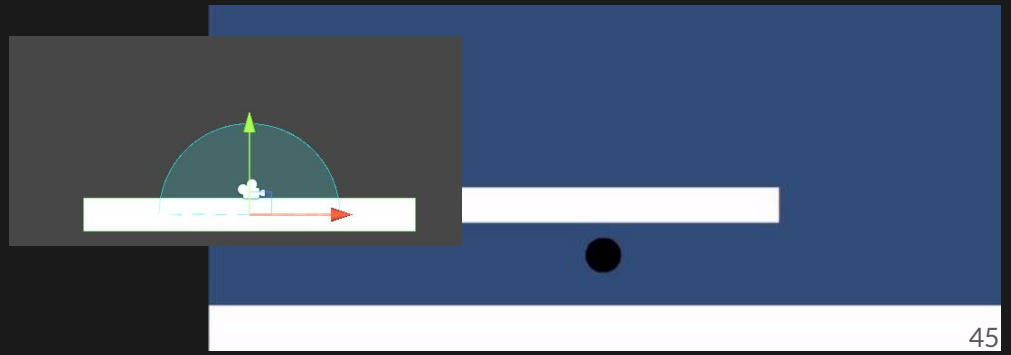


2D Effectors - Platform Effector

The **Platform Effector**, or **One Way Platform**, allows us to create the effect where the player can jump through one side of the platform while keeping the other side solid.

- **Rotation Offset** sets which direction the platform should be facing.
- **Surface Arc** determines which area should count as the solid surface.
- **Sides** allow you to control the behavior when something collides with the platform's side.



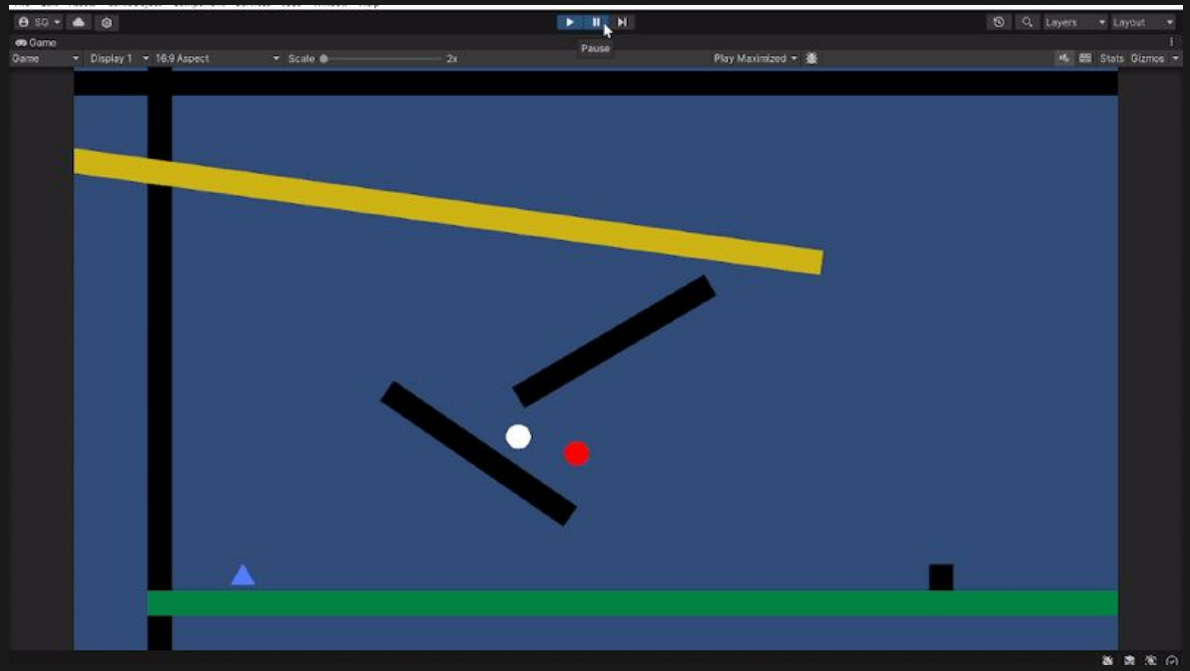


Challenge Platform Effector

Now, let's put your skills to the test!

Let's take that top ramp extend it, change its color and add a Platform Effector to it.

This way the balls should be able to go through the bottom and slide down on top of it.

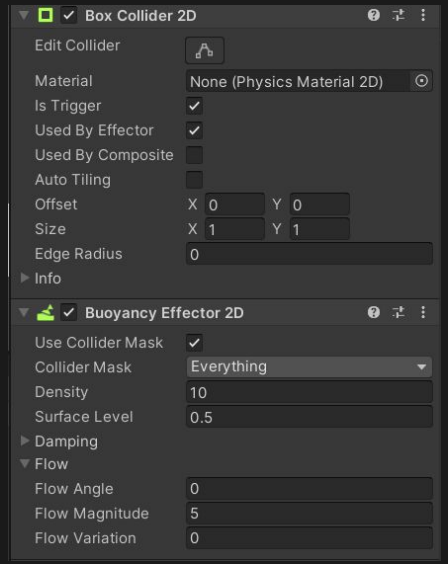


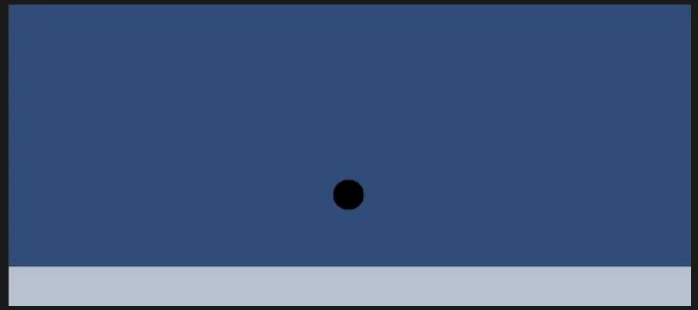
2D Effectors - Buoyancy Effector

The **Buoyancy Effector** is great for simulating bodies of water. It requires the **Box Collider** to have both **Trigger** and **Effector** enabled.

- **Density** determines whether the object that lands in it will sink or float.
- **Surface Level** where the waterline that the object would float around is.

It also has a **Flow** menu that works similarly to the **Surface Effector**, adding force to the left or right of the collider to push the object.



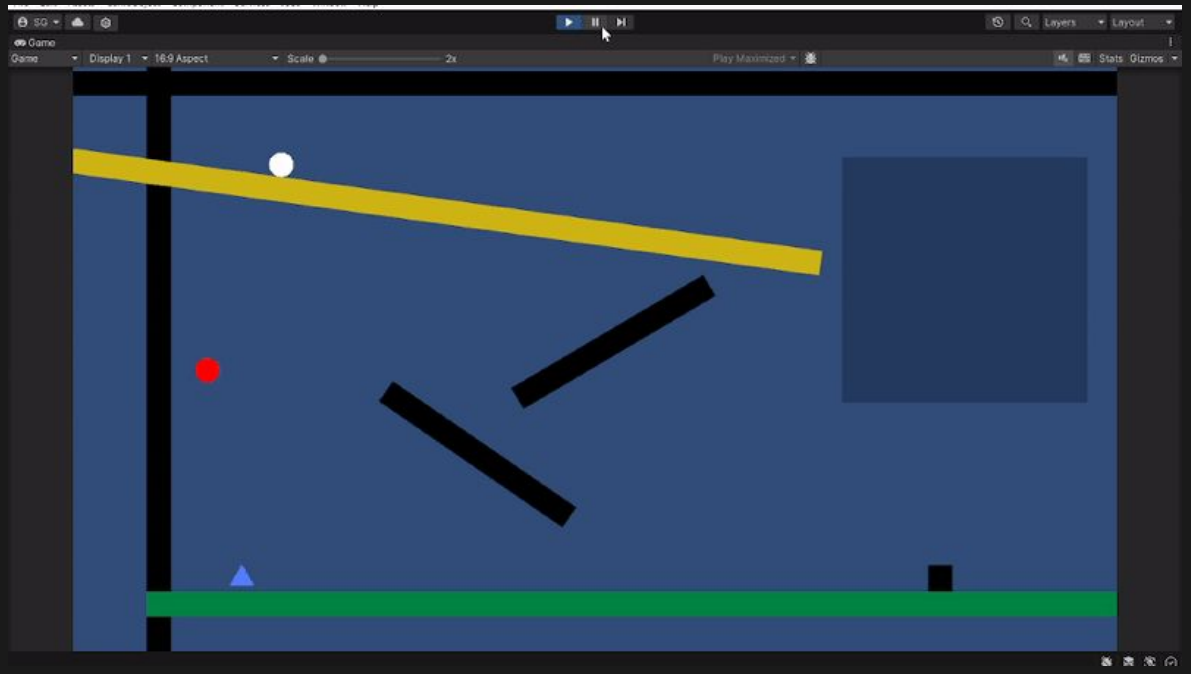


Challenge Buoyancy Effector

Now, let's put your skills to the test!

Lastly let's create a box at the end of the ramp, make sure it has a Collider and a Buoyancy Effector with a high surface level.

This should have the balls run into it and float to the top of our new collider.



PreFabs

What is a PreFab?

A **Prefab** (or **PreFabrication**) is a game asset that serves as a template for a **Game Object**. It contains all the components and variables of the original game object.

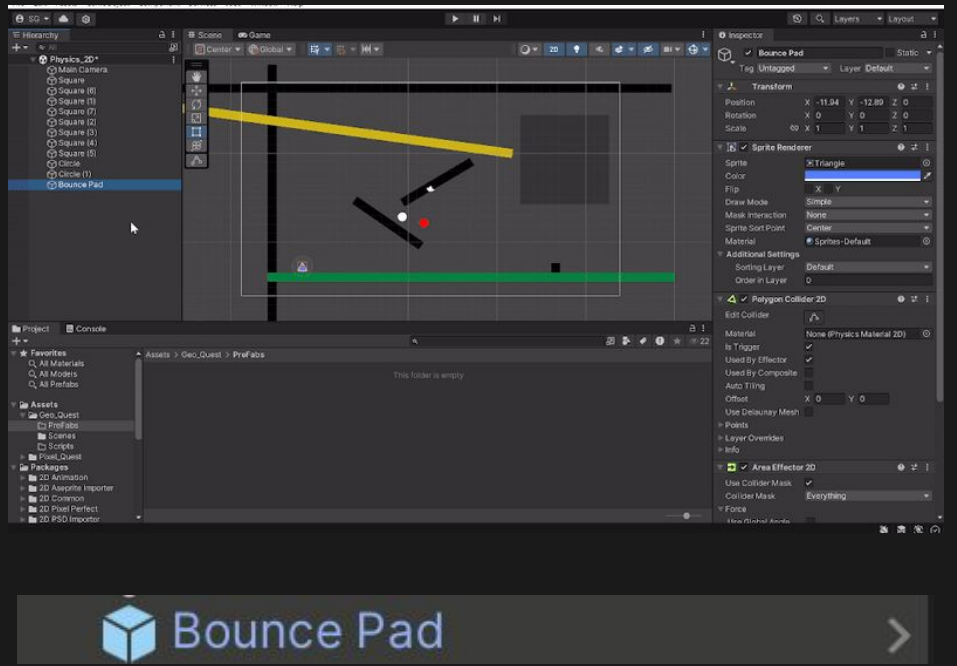
Think of any game object that repeats throughout a game: like enemies, platforms, or obstacles.

Prefabs allow you to create multiple copies of that object quickly, with the ability to edit the original and pass those changes on to all its copies.

Creating a PreFab?

Creating a **Prefab** is super simple, just drag the game object you want to make into a **Prefab** into the **Project View**.

This will create a game asset of the object, and the icon next to the object in the **Hierarchy** will change to a blue cube, indicating it's a **Prefab** copy.

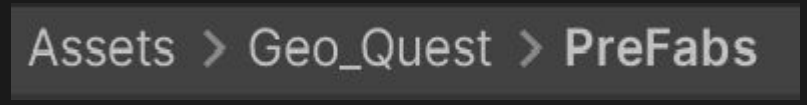


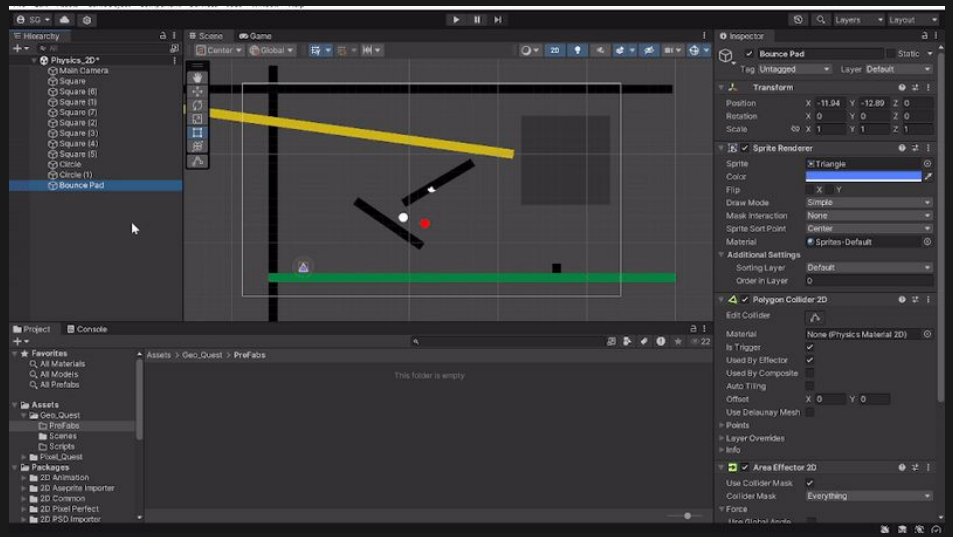
Challenge Create Bounce Pad PreFab

Now, let's put your skills to the test!

Navigate to the Assets > GeoQuest > PreFab folder.

Create a PreFab of the triangle with the area effector connected to it.

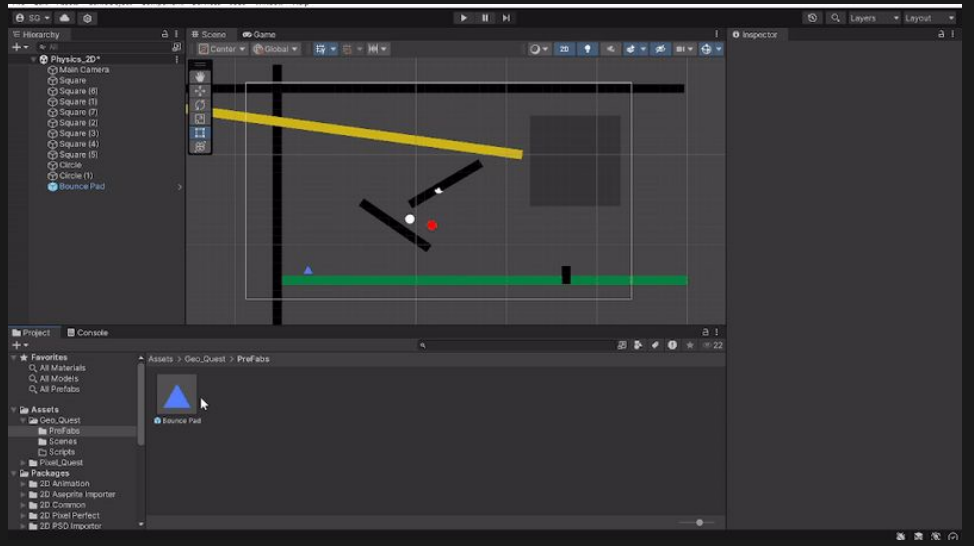




Instantiating PreFabs

Instantiating or creating **Prefab** copies is as easy as dragging the objects back into the scene.

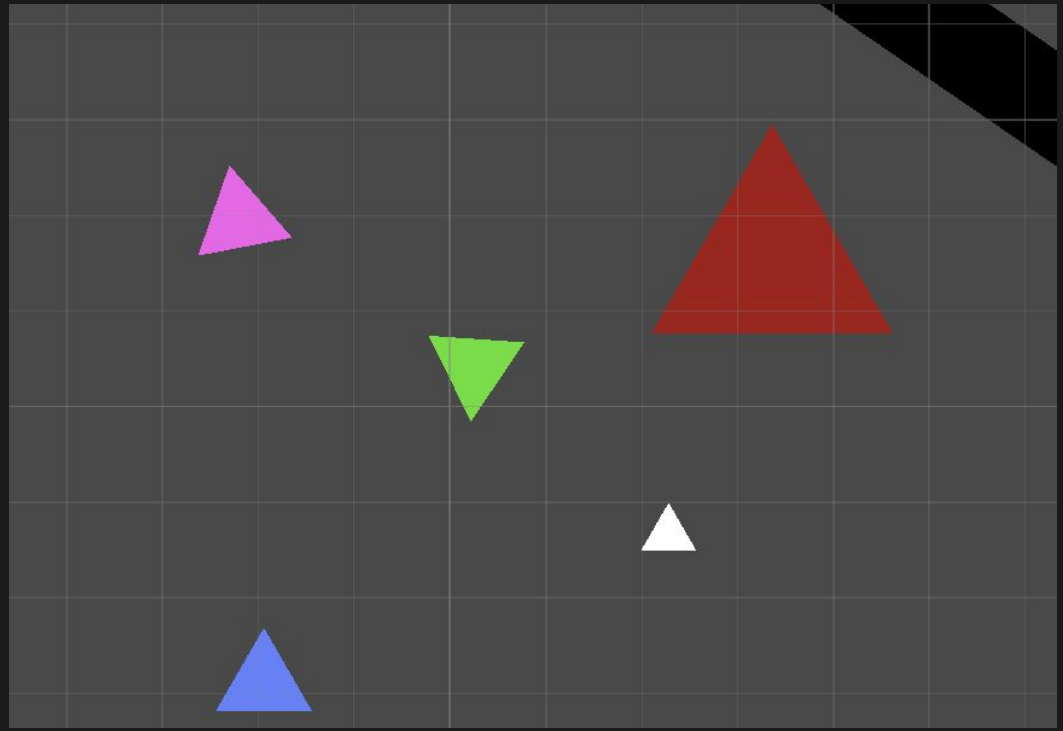
With this, we've created multiple game objects that already have the sprite, collider, and effector attached, behaving exactly as we want them to.



Making Changes to PreFab

Once you've instantiated the **Prefab,** you can change its attributes without affecting the others.

Here, I've changed the scale, rotation, and color of the Bounce Pads, and the changes are localized to each instance.

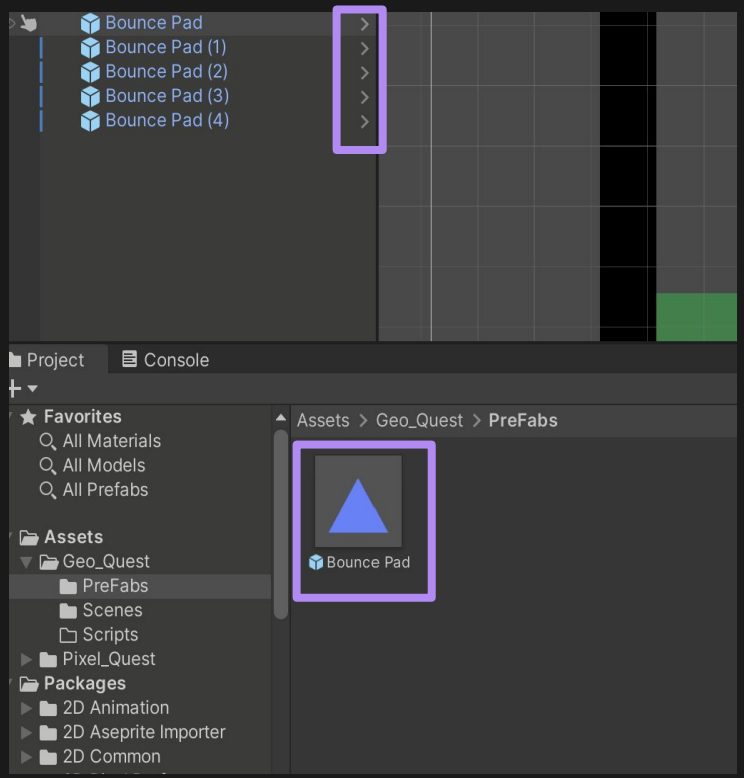


Editing the Original PreFab

When we want to make a change to the original **Prefab**, you can do so in two ways.

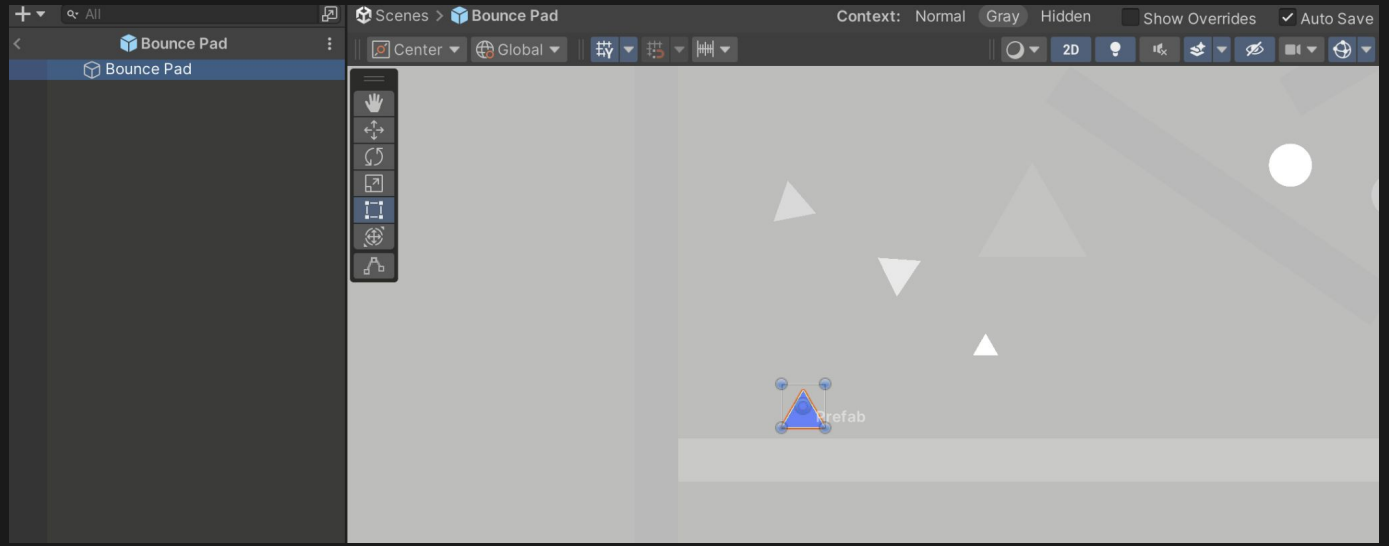
You can either click on the arrow next to the **Prefab** game object in the **Hierarchy**, or double-click the game asset in the **Project View**.

Either option will bring you to a special view that allows you to make changes to the original **Prefab**, and those changes will be passed on to all of its copies.



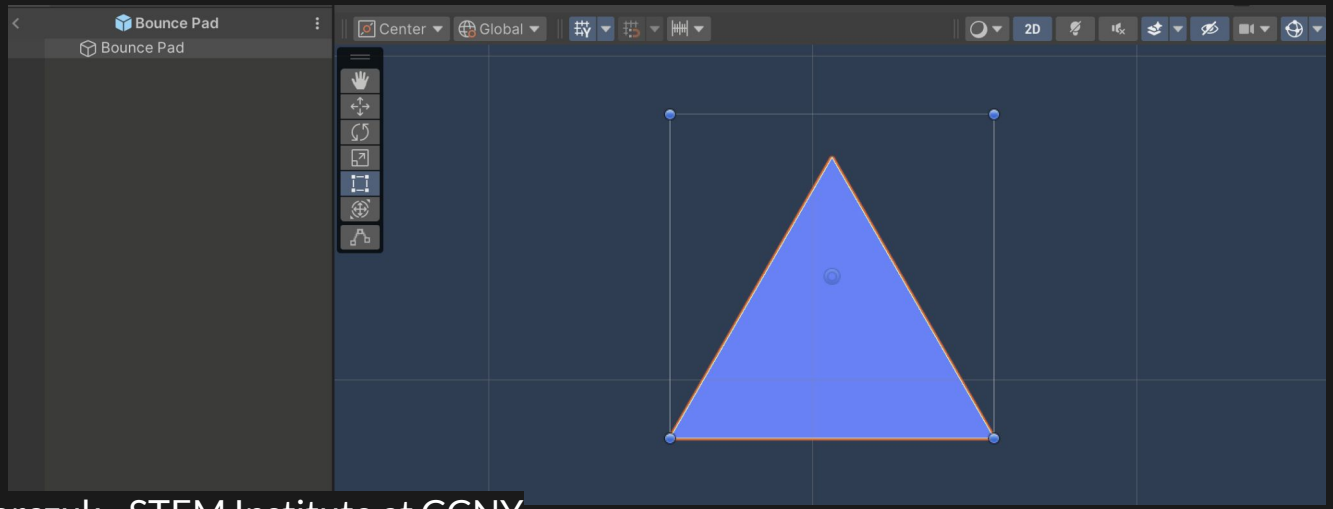
PreFab in Scene

By clicking on the arrow in the **Hierarchy**, you will open the **Prefab** in the Scene, meaning you can make changes to the original **Prefab** while viewing what it would look like in the scene, with all other objects grayed out.



PreFab in Isolation

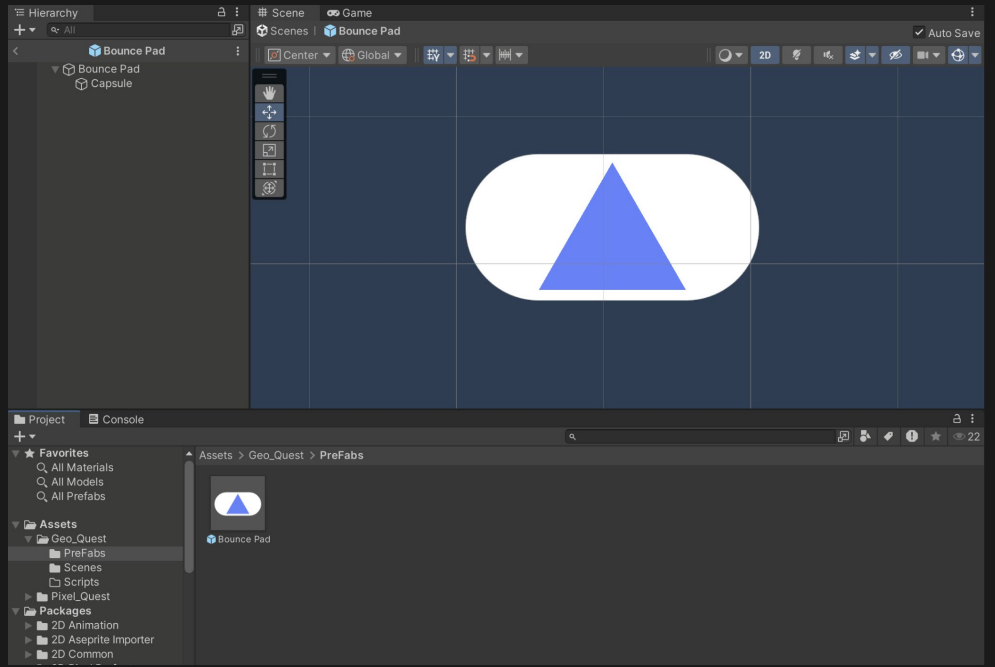
On the other hand, by double-clicking on the asset in the **Project View**, you will view the **Prefab** in isolation to see how the game object behaves on its own, with a blue background indicating you're in the isolated view



Adding to PreFab

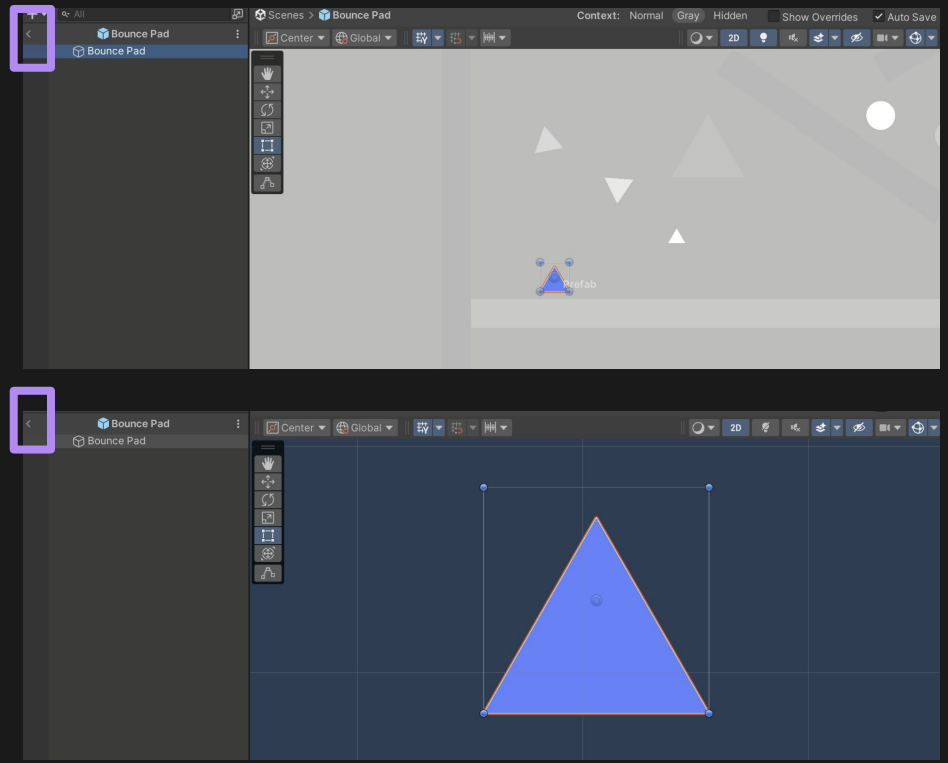
Regardless of where you work from, you can make any changes—adding game objects as children, adding components, etc. Any changes you make here will be passed on to all of the copies you've made across all scenes.

In this case I added a child Game Object that has a Capsule Sprite.



Exit PreFab View

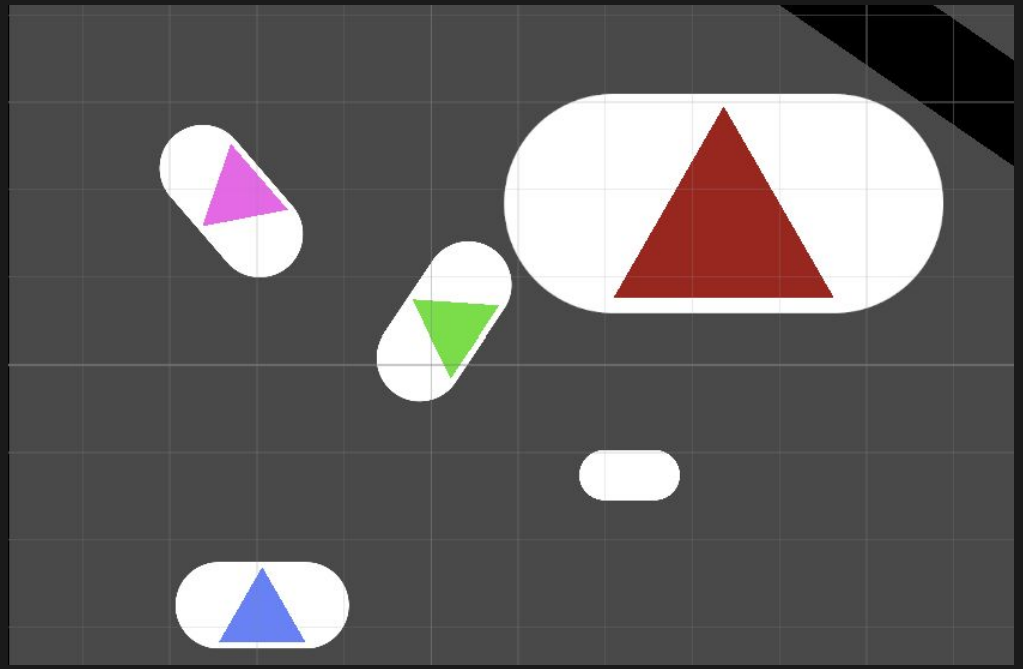
To return to the scene, click on the small arrow at the top of the **Hierarchy** while you're inside the **Prefab** view.



The Changes Going to

All the changes you make to the original **Prefab** will be carried over to all of its instances in all of your scenes.

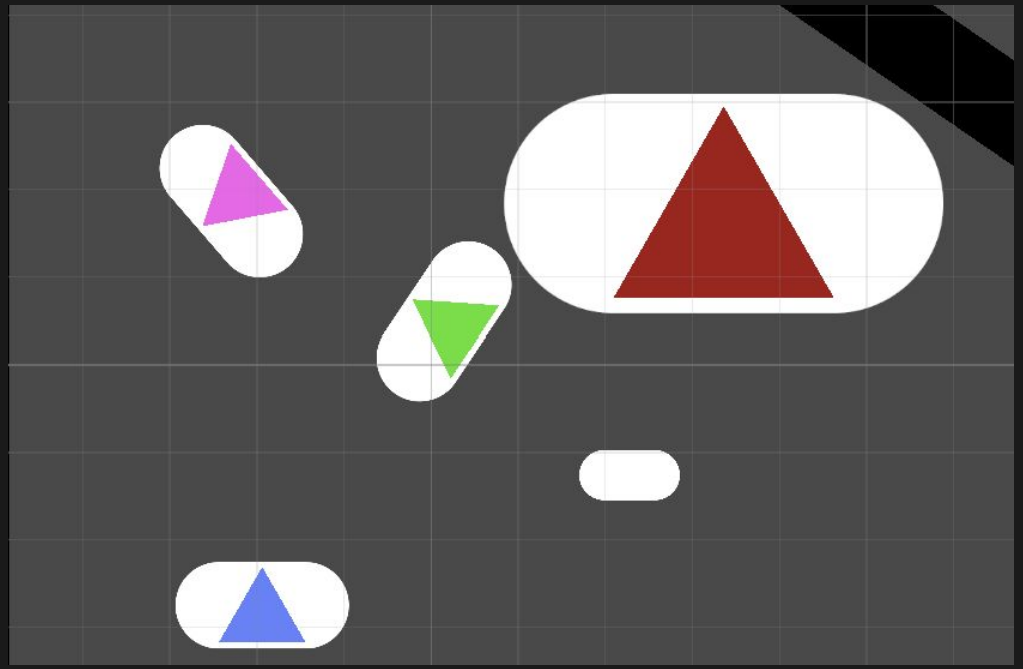
As you can see here, all of the game objects now have that capsule sprite behind them.



Challenge Add and Change PreFabs

Now, let's put your skills to the test!

Instance a few copies of the Bounce Pad, modify them slightly then go into the PreFab view and add a image to the PreFab so it gets sent to all of the copies.

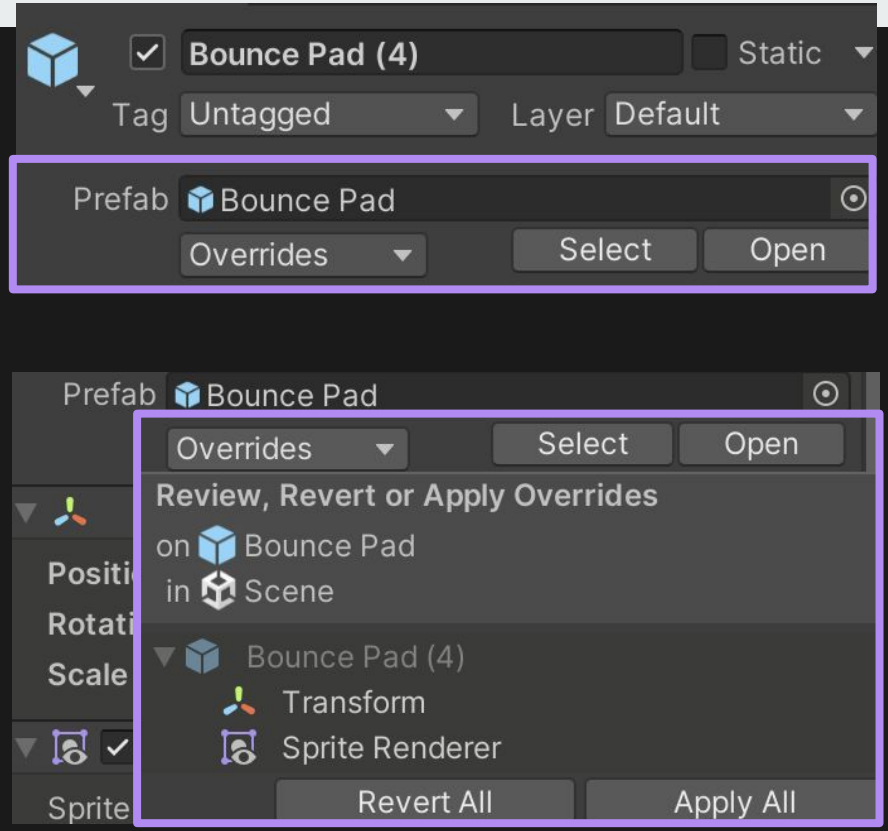


Saving Changes in Scene

Now, let's say that you actually liked the changes you made to an object in the scene and want that change to be passed on to all the other copies.

If we look at the **Inspector** header, we'll see a **Prefab** section now. It tells you the name of the original **Prefab** and allows you to open up the **Prefab** view. What we want to focus on is the **Overrides**.

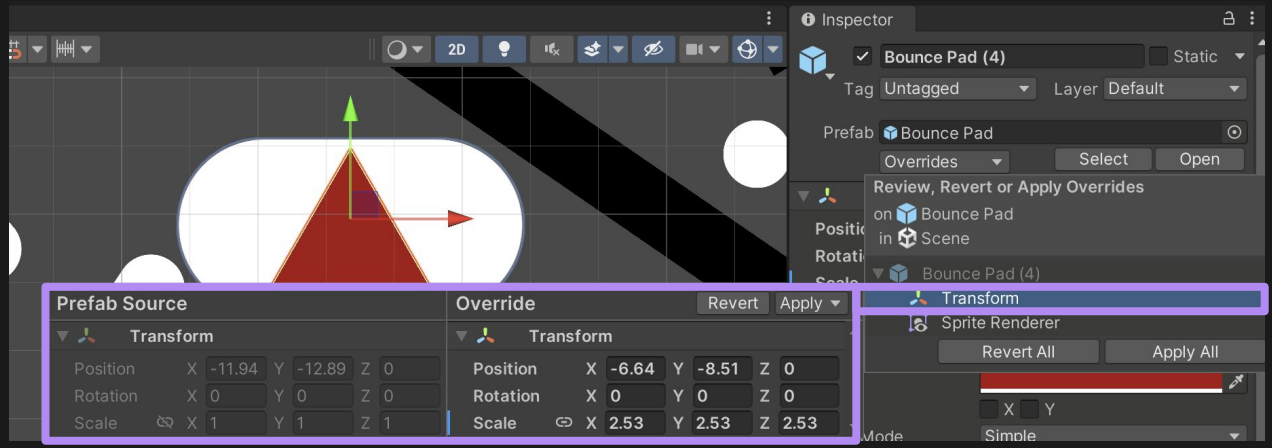
If you made changes that can be passed on, you'll be able to click on **Overrides** and it will show you what changes you've made. From there, you can undo all the changes, apply all of them, or even click on each one to review the changes.



Individual Changes

If you click on one of the changes in the list, it will show you the component that was changed, along with what it used to look like and what it looks like now.

You'll have the ability to either revert the individual changes or approve them individually.



Apply Changes

Now, if we change the **SpriteRenderer**, for example, you'll notice that not all of the copies changed color.

That's because we've already edited that aspect of them.

If we didn't like the one in the bottom left, it will revert to the new original color, but the ones that have already been changed will keep their modifications until reverted to the original version.

All new copies, however, will have that change applied to them.