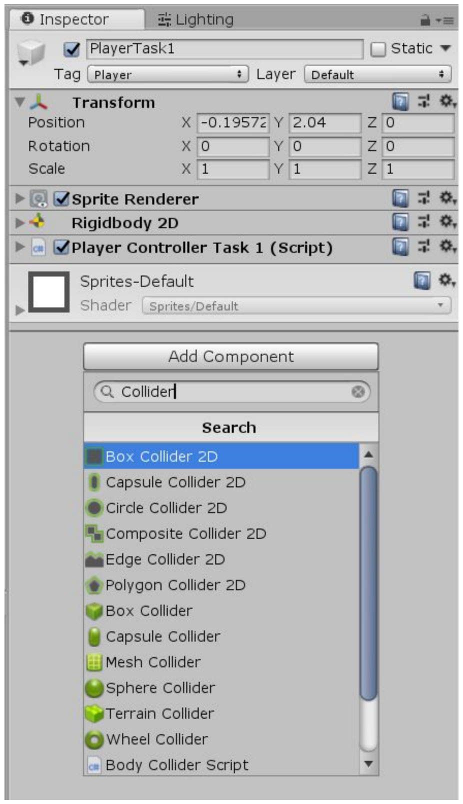
Colliders

Table of Contents

Overview

Colliders are one of the basic building blocks of games. Two of its main uses are as physical boundaries and as event triggers, both of which you will learn in this lab.



PlayerTask1
Tag Player Layer Default
Position X -0.19572 Y 2.04 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Shader Sprites/Default
Collider

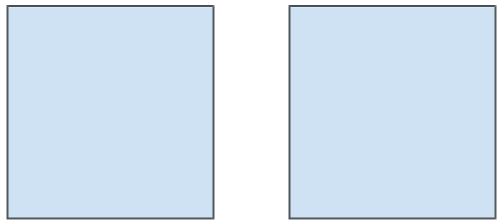
To add a collider to a GameObject:

1. select the object in the hierarchy and then in the Inspector
2. click the Add Component button
3. in the search bar, type “Collider”, and select the shape that you want to use

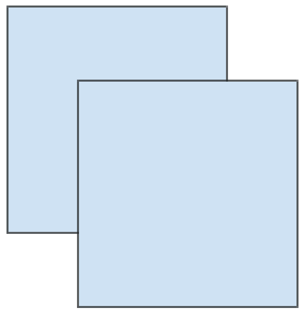
For this lab, we will be using the **Box Collider 2D** and the **Circle Collider 2D** components to implement parts of a naive jump function and an enemy turret that shoots projectiles at the player.

Colliders & Triggers

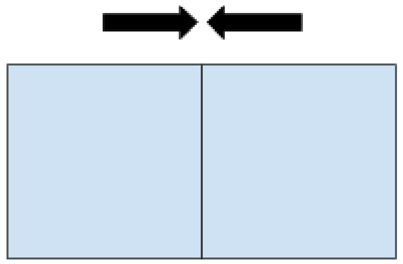
If you already have a solid understanding of how colliders work, the OnCollisionEnter/Stay/Exit() functions, and the OnTriggerEnter/Stay/Exit() functions, then feel free to skip this section.



Imagine these are two GameObjects moving towards each other.



If neither of them have colliders, then there will be no physical collision when they bump into each other, which may not necessarily be the behavior we want. Instead, they will continue through each other.



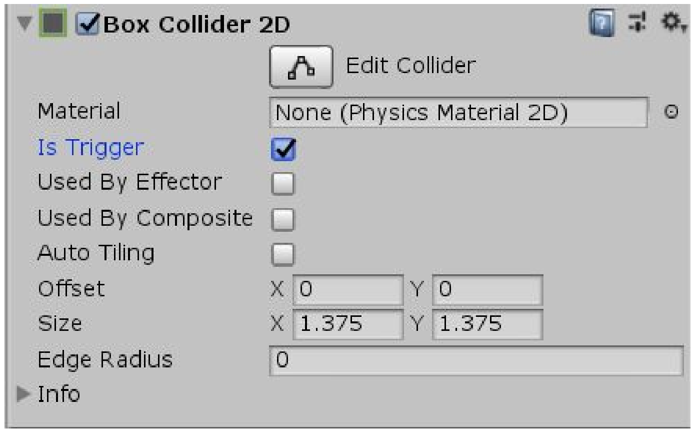
However, with colliders (NOT triggers) attached, these two squares will stop when they come into contact.

There are three methods that will execute whenever a collision occurs:

- On the frame that they come into contact, the OnCollisionEnter() function is called.
- For every frame that they remain in contact, OnCollisionStay() is called.
- On the frame that they leave contact, OnCollisionExit() is called.

To define what happens when two colliders interact, you must implement these three functions in the scripts that you assign to the GameObjects with colliders. It is not necessary to implement all three; for example, if you are only concerned with what happens when collision begins, then feel free to only implement OnCollisionEnter.

Triggers are colliders that do not cause physical collisions, obtained by selecting the "Is Trigger" option in the collider component. Their main use case is when you want a collision to trigger something without any physical interactions or barriers.

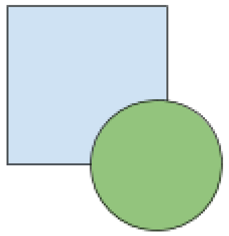


Box Collider 2D

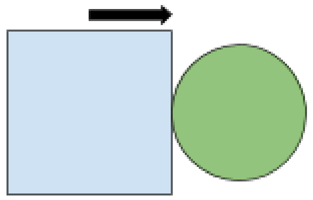
Edit Collider

Material	None (Physics Material 2D)
Is Trigger	✓
Used By Effector	☐
Used By Composite	☐
Auto Tiling	☐
Offset	X 0 Y 0
Size	X 1.375 Y 1.375
Edge Radius	0

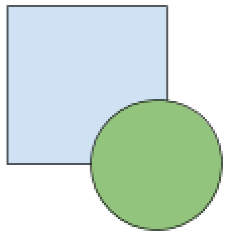
▶ Info



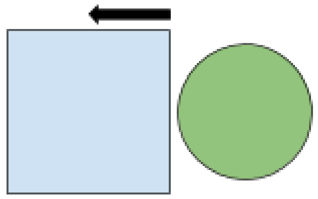
If you have a trigger collider (Green), it will not physically collide with any other colliders (Blue).



Instead, the trigger functions will be called in the same way as the collision functions without a physical collision. OnTriggerEnter() is called on the frame when the trigger comes into contact with another collider.



OnTriggerStay() is called for every frame that the trigger and another collider remain in contact.



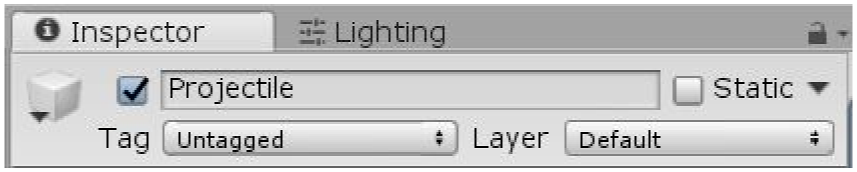
OnTriggerExit() is called on the frame when the trigger and another collider leave contact.

It’s important to note that these collider functions will only be called if at least one of the GameObjects involved in the collision has a Rigidbody component attached.

No Rigidbody = No collision!

Physics Layers

Every object in Unity is assigned to a layer:

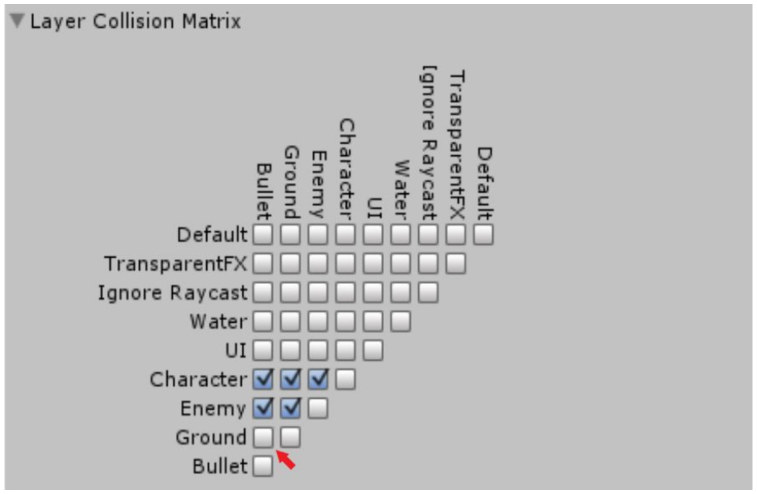


☑ Projectile ☐ Static

Tag Untagged Layer Default

By default, everything is on the Default layer and Game Objects from all layers can collide with one another. However, when your game starts having multiple types of objects, collision interactions can start becoming increasingly complicated.

For example, let's say that you want a special bullet object that is able to pass through the ground but still able to hit enemies. Instead of modifying the collision functions of the bullet and ground objects, you can instead utilize Unity's layer collision matrix, which allows you to toggle collisions between specific layers:



	Bullet	Ground	Enemy	Character	UI	Water	Ignore Raycast	TransparentFX	Default
Default	☐	☐	☐	☐	☐	☐	☐	☐	☐
TransparentFX	☐	☐	☐	☐	☐	☐	☐	☐	
Ignore Raycast	☐	☐	☐	☐	☐	☐	☐		
Water	☐	☐	☐	☐	☐	☐			
UI	☐	☐	☐	☐	☐				
Character	✓	✓	✓	☐					
Enemy	✓	✓	☐						
Ground	☐	☐							
Bullet	☐								

By unchecking the Bullet X Ground box, Game Objects on the Bullet and Ground layers will no longer collide with each other (physical and trigger). To access the collision matrix, go to Edit > Project Settings > Physics2D (for 2D games). This is a very useful tool in specifying your collider interactions!

Lab Instructions

Task 1:

Open up the JumpDemo scene if you haven't already. For this task, you will need to enable the following:

- PlayerTask1, MainCamera, and Environment
- Disable all the others by clicking on them in the hierarchy and unchecking the box to the left of its name in the inspector

a) Hit the play button. The player should fall right through the floor. This doesn't seem like something we want, so let's fix it. Add a **Box Collider 2D** component to PlayerTask1 and size it appropriately by modifying the variables in the inspector OR clicking "Edit Collider" and adjusting the collider in the scene view.

b) Hit the play button again. Now, the player should be able to physically collide with all of the platforms and walls. Try moving around and jumping (Space). **Are there any times when you can jump when you shouldn't be able to?**

- Be able to answer this during the checkoff.

c) We will now fix some problems with jumping. In the Scripts folder, open PlayerControllerTask1.cs. This is the script that controls the PlayerTask1 object. Modify the sections labeled 'Task 1' so that you **should only be able to jump if you are touching a platform, wall, or floor**.

- Be able to answer why our player can jump while touching these locations.

Task 2:

For this task, you will need to enable the following:

- PlayerTask2, MainCamera, and Environment
- Disable all others

a) We will now implement a smarter check for jumping. Notice that PlayerTask2 itself does not have a box collider -- you can accomplish this task without a collider directly on the GameObject. Make modifications so that PlayerTask2 **can jump only when its *feet* are touching the ground**. If you need hints, take a look at the children of PlayerTask2 and the other scripts.

Task 3:

For this task, you will need to enable the following:

- PlayerTask2, ShooterBoss, MainCamera, and Environment
- Disable all others

a) We will now implement a **ShooterBoss** that behaves like a turret. The ShooterBoss keeps a list of all targets within a radius (of your choosing). Upon entering the radius, the player should be added to that list. Upon exiting the radius, the player should be removed from that list. The shooting function is already completed, so you just need to worry about adding and removing targets from the list. *HINT: This task will require use of a trigger collider.*

Lab Summary

You should now have an understanding of what Colliders are, the differences between colliders and triggers, and how to use them. They are simple but incredibly powerful building blocks that are used in practically every game, and will be invaluable as you build your own games!

Checkoff

1. Show functionality of PlayerTask1, and that it can jump when touching any physical barrier.
    a. Explain why this is the case.
2. Show functionality of PlayerTask2, and that it can jump only when its feet are touching the ground.
    a. Explain why this is the case.
3. Show that the ShooterBoss only fires projectiles while you are within its radius.

Challenge

This challenge is highly recommended, especially if you are interested in making any sort of shooting game!

a) You might notice from Task 3 that the ShooterBoss projectiles don't actually interact with anything. Change that by giving the projectile prefab a collider (trigger or not is up to you) and defining its interactions.

- One idea is to have it physically push the player if it hits them. Another would be to cause an explosion on impact. Be creative!

b) Create different types of projectiles and practice with the layer collision matrix.

- One idea is a projectile that only ignores platforms.

c) Give the player a method of fighting back. This might involve creating health scripts for the player/boss. Feel free to use the ShooterScript as reference for shooting projectiles if you'd like.