Unity Game Engine

Introduction to Unity – Collision and Physics

https://docs.unity3d.com/Manual/CollidersOverview.html
https://docs.unity3d.com/Manual/PhysicsSection.html
https://unity3d.com/learn/tutorials/topics/physics (3D Physics - Colliders, Colliders as Triggers, Rigidbody)
https://docs.unity3d.com/Manual/LayerBasedCollision.html
https://docs.unity3d.com/ScriptReference/Rigidbody.html

Collision Basics

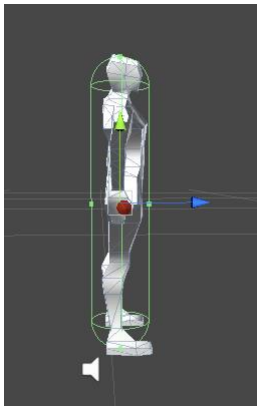
- Introduce Unity 3D's built-in collision system.
- Two important components:
  - Collider Component
  - Rigidbody Component
- Configuration of Collider and Rigidbody Components
  - Static Collider
  - Rigidbody Collider
  - Kinematic Rigidbody Collider

Collision Basics

- Introduce Unity 3D's built-in collision system.
- Two important components:
  - Collider Component
  - Rigidbody Component
- Configuration of Collider and Rigidbody Components
  - Static Collider
  - Rigidbody Collider
  - Kinematic Rigidbody Collider

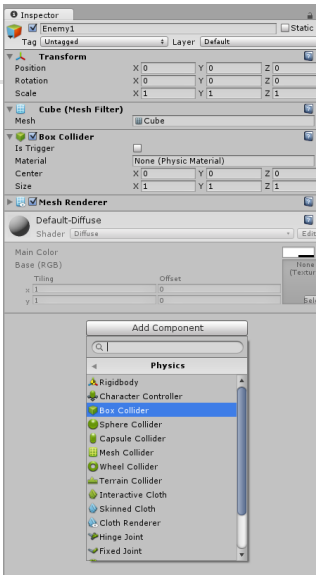
Colliders

- **Collider** components define the shape of an object for the purposes of physical collisions.
- A **collider**, which is *invisible*, need not be the exact same shape as the object's mesh and in fact, a rough approximation is often more efficient and indistinguishable in gameplay.



Defining a collider

- We can add a collider component to a selected object.
- Inspector > Add Components > Physics > Colliders.
  - The primitive ones are **Box**, **Sphere** and **Capsule** collider components.



Primitive Colliders

- The **Box Collider** defines a cube area where collisions will be detected. You can define the box's center and size.
- The **Sphere Collider** defines a spherical volume where collisions will be detected.
- The **Capsule Collider** defines a Capsule volume for collision detection. In this case, you'll also be able to define the height of the capsule, and the axis for orientation.
- The collider volume and position does not have to correspond with your game object's mesh, and frequently it won't.

Mesh Colliders

- **Mesh Colliders** are a particular type of collider which uses an actual mesh for collision detection.
- They are quite expensive as they compute collision against every face of the mesh.
- You should avoid using Mesh colliders if possible.
- If you really need to, you can supply a separate mesh to the component, which has the same shape and structure of the game object's mesh but with fewer faces.

Mesh Colliders (cont'd)

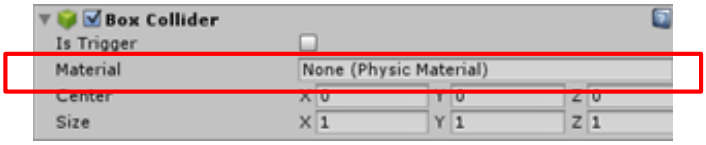
- Normally, collisions between two mesh colliders are ignored.
- If you want to detect collisions between mesh colliders, you need to set them as Convex in the Inspector. In this case, you'll need to supply a mesh with less than 255 faces.

Multiple Colliders for an Object

- You can define multiple colliders for a single object to define complex collision areas.
  - To do so, you need to create empty child objects of your game object, and define colliders for each child.

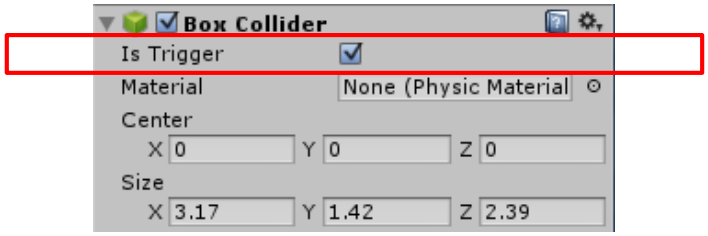
Collider Material

- All Colliders have a Material field which determines how the collider will react to collisions, in regard to friction and bounciness. We'll ignore it for now.



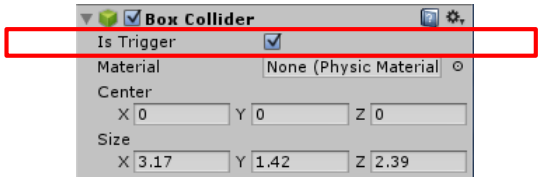
Colliders as Triggers

- If you're going to use the collider as a trigger area, you need to tick "**Is Trigger**."
- A collider configured as a **Trigger** does not behave as a solid object and will simply allow other colliders to pass through.



Colliders as Triggers (2)

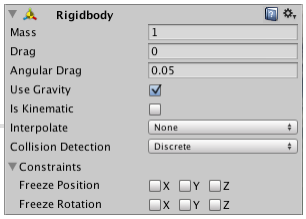
- The physics engine simply detects when one collider enters the space of another without creating a collision for the physics engine.
- This way, collisions will be ignored by the physics engine but they'll still generate events that you can listen to in your scripts.
- When a collider enters its space, a trigger will call the **`OnTriggerEnter`** function on the trigger object's scripts.



Collision Basics

- Introduce Unity 3D's built-in collision system.
- Two important components:
  - Collider Component
  - Rigidbody Component
- Configuration of Collider and Rigidbody Components
  - Static Collider
  - Rigidbody Collider
  - Kinematic Rigidbody Collider

Rigidbody



- If you want your object to **<u>react to physical collision with other objects</u>** and the game world, you'll need to add a **Rigidbody** component.
- A game object with a rigid body will be influenced by gravity and external forces.
- **You <u>won't need a Rigidbody</u> if your object's collider is set as <u>trigger</u>.**

Collision Basics

- Introduce Unity 3D's built-in collision system.
- Two important components:
  - Collider Component
  - Rigidbody Component
- Configuration of Collider and Rigidbody Components
  - Static Collider
  - Rigidbody Collider
  - Kinematic Rigidbody Collider

Collider Interactions

- Colliders interact with each other differently depending on how their Rigidbody components are configured.
  - **Static Collider**
    - Collider only
  - **Rigidbody Collider**
    - Collider and non-kinematic Rigidbody
  - **Kinematic Rigidbody Collider**
    - Collider and kinematic Rigidbody

(C.F) Kinematics *is a mechanics that describes the motion of objects without considering the forces that caused the motion.*

Static Collider

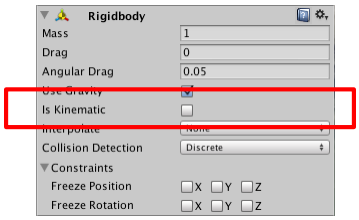
- This a GameObject that has a **Collider but no Rigidbody**.
- Static colliders are used for level geometry which always stays at the same place and never moves around.
  - For your FPS, you created your room with Cube objects coming with Box collider.

Static Collider (2)

- Incoming rigidbody objects will collide with the static collider but will not move it.
- The physics engine assumes that static colliders never move or change and can make useful optimizations based on this assumption.
- Consequently, static colliders should not be disabled/enabled, moved or scaled during gameplay.

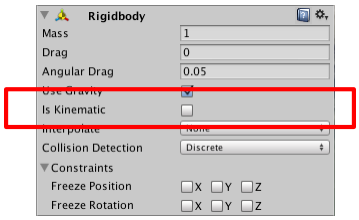
Rigidbody Collider

- This is a GameObject with a **Collider** and a **normal, *non-kinematic* Rigidbody** attached.
- **Rigidbody colliders** are <u>fully simulated by the physics engine</u> and can react to collisions and forces applied from a script.



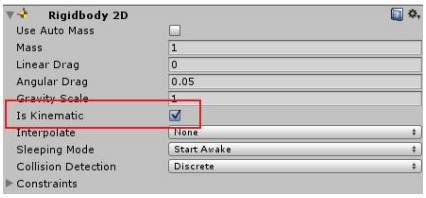
Rigidbody Collider (2)

- They can collide with other objects (including static colliders).
- They are the most commonly used Collider configuration in games that use physics.



Kinematic Rigidbody Collider

- This is a GameObject with a **Collider** and a ***kinematic*** **Rigidbody** attached (ie, the *IsKinematic* property of the Rigidbody is enabled).



- A Rigidbody component can be switched between normal and kinematic behavior at any time using the *IsKinematic* property.

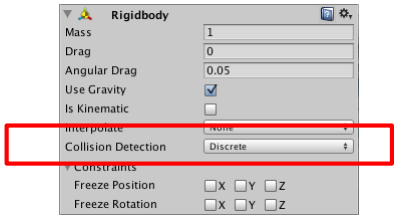
Kinematic Rigidbody Collider (cont'd)

- Use case1 :
  - Kinematic rigidbodies should be used for colliders that can be moved or disabled/enabled occasionally but that should otherwise behave like static colliders.
    - e.g a sliding door that should normally act as an immovable physical obstacle but can be opened when necessary.
- Use case2 :
  - Another common example of this is the "ragdoll" effect where a character normally moves under animation but is thrown physically by an explosion or a heavy collision.

Rigidbody collision detection

Rigidbody Collision Detection

- From the Inspector, you can set whether you want **discrete** or **continuous collision detection**.



- **Continuous collision** detection is used when dealing with fast moving objects.

Rigidbody Collision Detection (2)

- When using **Discrete collision detection**, some collisions may not be detected as they may go through the other object collider between the time another check is performed.
- **By default,** this is set to **Discrete** and you should leave it as continuous collision detection may really slow down your game.

Programming with Collisions

Collision Action Matrix

- When two objects collide, a number of different script events can occur depending on the configurations of the colliding objects' rigidbodies.
- **Collision Action Matrix** gives details of which event functions are called based on the components that are attached to the objects.
- Some of the combinations only cause one of the two objects to be affected by the collision, but **the general rule is that <u>physics will not be applied to an object that doesn't have a Rigidbody component attached</u>**.

Collision Action Matrix : Collision Detection

Collision detection occurs and messages are sent upon collision						
	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider		Y				
Rigidbody Collider	Y	Y	Y			
Kinematic Rigidbody Collider		Y				
Static Trigger Collider						
Rigidbody Trigger Collider						
Kinematic Rigidbody Trigger Collider						

<u>Collision detection</u> occurs iff one of colliding objects is a Rigidbody objects.

Collision Action Matrix: Trigger Message

Trigger messages are sent upon collision						
	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider					Y	Y
Rigidbody Collider				Y	Y	Y
Kinematic Rigidbody Collider				Y	Y	Y
Static Trigger Collider		Y	Y		Y	Y
Rigidbody Trigger Collider	Y	Y	Y	Y	Y	Y
Kinematic Rigidbody Trigger Collider	Y	Y	Y	Y	Y	Y

Trigger messages occur iff one of colliding objects is a Rigidbody (including kinematic) objects.

Non-trigger Object Collision

- **For non-trigger collision**, you'll use
  - `OnCollisionEnter`
  - `OnCollisionExit`
  - `OnCollisionStay`.

Collision detection occurs and messages are sent upon c			
	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider
Static Collider		Y	
Rigidbody Collider	Y	Y	Y
Kinematic Rigidbody Collider		Y	

- `OnCollisionEnter` is called when the game object collider starts touching another game object with a collider and rigidbody attached.

Non-trigger Object Collision (2)

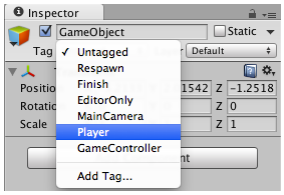
- While colliding, **`OnCollisionStay`** will be called once per frame.
- Finally, when the collision stops, **`OnCollisionExit`** will be called.
  - If the collision stops as a result of one of the objects being destroyed using Destroy(), **`OnCollisionExit`** won't be called.


```
using UnityEngine;
using System.Collections;

public class CollisionTest : MonoBehaviour
{

    void OnCollisionEnter(Collision collisionInfo) {
        print("Detected collision between " +
              gameObject.name +
              " and " + collisionInfo.collider.name);
        print("There are " + collisionInfo.contacts.Length +
                  " point(s) of contacts");
        print("Their relative velocity is " +
                  collisionInfo.relativeVelocity);
    }

    void OnCollisionStay(Collision collisionInfo) {
        print(gameObject.name + " and " +
              collisionInfo.collider.name
              + " are still colliding");
    }

    void OnCollisionExit(Collision collisionInfo) {
        print(gameObject.name + " and " +
              collisionInfo.collider.name
              + " are no longer colliding");
    }
}
```

Utilizing Tags for Collision

- When dealing with collisions, it's useful to set different tags for the game objects in your game world.
- This way, you can quickly determine how to react with different types of collisions in your game, whether it's a collectable, an enemy or anything else.

* You can assign a tag to a game object through Inspector or add a tag. Tag drop-down menu just below the game object name in Inspector.




```
void OnCollisionEnter(Collision collisionInfo) {
    if(collisionInfo.collider.tag == "Enemy") {
        print("Lose health point");
    }
    else if (collisionInfo.collider.tag == "Powerup") {
        print("Collect powerup");
    }
}
```

Trigger Object Collision

- If the game object's collider is set as Trigger, use
  - **OnTriggerEnter**
  - **OnTriggerStay**
  - **OnTriggerExit**
- They work roughly the same way, but supply the Collider object (e.g box collider) directly instead of giving a Collision object (e.g enemy).
- <u>For trigger collision event to be triggered, one of the two colliders needs to have a rigid body attached.</u>


```
using UnityEngine;
using System.Collections;

public class CollisionTest : MonoBehaviour
{

    void OnTriggerEnter(Collider other) {
        print("Collision detected with trigger object "
            + other.name);
    }

    void OnTriggerStay(Collider other) {
        print("Still colliding with trigger object " +
                other.name);
    }

    void OnTriggerExit(Collider other) {
        print(gameObject.name + " and trigger object "
            + other.name + " are no longer colliding");
    }
}
```

Filtering Collisions

- You can force the collision system to ignore certain type of collisions, either specifying the actual objects collider or using layers.

Ignoring Collisions between Objects

- To ignore collisions between the game object and another game object you'll need to use IgnoreCollision, supplying the respective colliders:

```
Physics.IgnoreCollision(gameObject.collider,
                        anotherGameObject.collider);
```

Ignoring Collisions using Layers

- IgnoreLayerCollision lets you specify two layers (using their IDs as integers) and will tell the collision system to ignore collisions between layers.

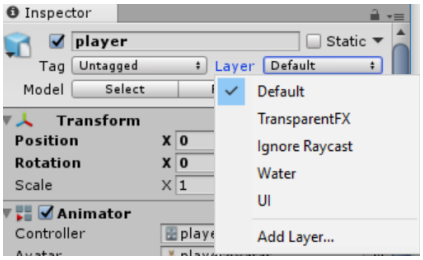
```
Physics.IgnoreLayerCollision(1, 2);
```

- To check if collision between two layers are ignored, use GetIgnoreLayerCollision:

```
bool areIgnored =
Physics.GetIgnoreLayerCollision(1, 2);
```


Layers

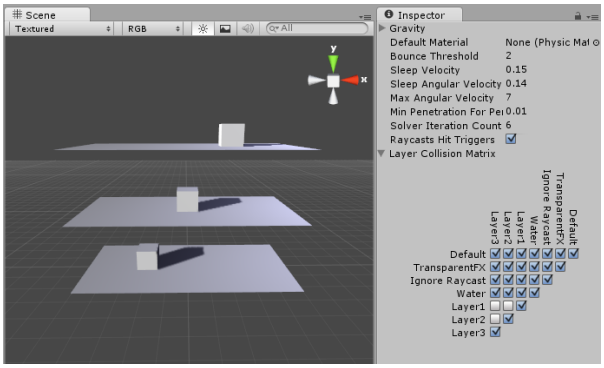
- **Layers** are most commonly used by **Cameras** to render only a part of the scene, and by **Lights** to illuminate only parts of the scene.



- But **layers** can also be used by raycasting to selectively ignore colliders or to create collisions.

Layer-Based Collision Detection

- Layer-based collision detection is a way to make a GameObject collide with another GameObject that is set up to a specific Layer or Layers.

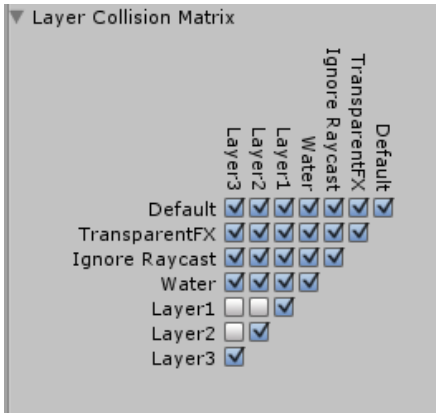


6 GameObjects (3 planes, 3 cubes) in the Scene view

Layer Collision Matrix in the window to the right

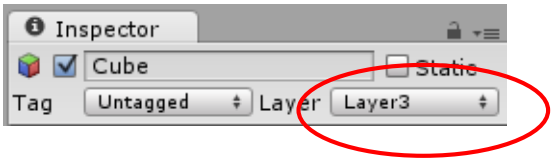
Layer Collision Matrix

- The Layer Collision Matrix is set up so that only GameObjects that belong to the same layer can collide:
- Layer 1 is checked for Layer 1 only
- Layer 2 is checked for Layer 2 only
- Layer 3 is checked for Layer 3 only



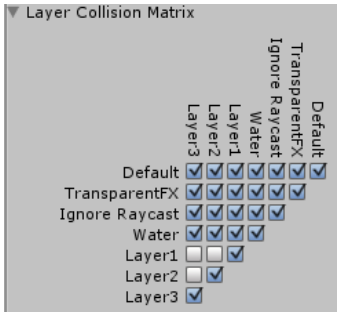
Setting up layer-based collision detection

- Step 1 - Select a Layer for your GameObjects to belong to.
  - Select the GameObject, navigate to the Inspector window, select the **Layer** dropdown at the top, and either choose a Layer or add a new Layer.
  - Repeat for each GameObject.



Setting up layer-based collision detection (cont'd)

- Step 2 - Open the **Physics Manager** window (**Edit** > **Project Settings** > **Physics)** and select which layers on the Collision Matrix will interact with the other layers by checking them.



Collision and Physics

Rigidbody Component

- A **Rigidbody** is the main component that enables physical behaviour for a GameObject.
- With a Rigidbody attached, the object will immediately respond to gravity. If one or more **Collider** components are also added, the GameObject is moved by incoming collisions.

Rigidbody Component (2)

- Since a Rigidbody component takes over the movement of the GameObject it is attached to, you shouldn't try to move it from a script by changing the Transform properties such as position and rotation. Instead, you should apply **forces** to push the GameObject and let the physics engine calculate the results.


```
using UnityEngine;
public class ExampleClass : MonoBehaviour {
    public float thrust;
    public Rigidbody rb;
    void Start() {
        rb = GetComponent<Rigidbody>();
    }
    void FixedUpdate() {
        rb.AddForce(transform.forward *
            thrust);
    }
}
```

This example applies a forward force to the GameObject's Rigidbody.

Rigidbody Component (3)

- There are some cases where you might want a GameObject to have a Rigidbody without having its motion controlled by the physics engine.
  - For example, you may want to control your character directly from script code but still allow it to be detected by triggers. This kind of non-physical motion produced from a script is known as *kinematic* motion.
  - The Rigidbody component has a property called **Is Kinematic** which removes it from the control of the physics engine and allow it to be moved kinematically from a script.
  - It is possible to change the value of **Is Kinematic** from a script to allow physics to be switched on and off for an object, but this comes with a performance overhead and should be used sparingly.

Rigidbody Class

Some Variables

centerOfMass	The center of mass relative to the transform's origin.
collisionDetectionMode	The Rigidbody's collision detection mode.
isKinematic	Controls whether physics affects the rigidbody.
mass	The mass of the rigidbody.
position	The position of the rigidbody.
rotation	The rotation of the rigidbody.
useGravity	Controls whether gravity affects this rigidbody.
velocity	The velocity vector of the rigidbody.

Rigidbody Class

Some Public Functions

AddForce	Adds a force to the Rigidbody.
AddTorque	Adds a torque to the rigidbody.
ClosestPointOnBounds	The closest point to the bounding box of the attached colliders.
GetPointVelocity	The velocity of the rigidbody at the point worldPoint in global space.
GetRelativePointVelocity	The velocity relative to the rigidbody at the point relativePoint.
MovePosition	Moves the rigidbody to position.
MoveRotation	Rotates the rigidbody to rotation.
ResetCenterOfMass	Reset the center of mass of the rigidbody.

Character Controllers

- The character in a game will often need some collision-based physics so that it doesn't fall through the floor or walk through walls.
- **Character Controller** component gives the character a simple, capsule-shaped collider that is always upright.
- The controller has its own special functions to set the object's speed and direction but unlike true colliders, **a rigidbody is not needed** and the momentum effects are not realistic.

Character Controllers (cont'd)

- A character controller cannot walk through static colliders in a scene, and so will follow floors and be obstructed by walls.
- It can push rigidbody objects aside while moving but will not be accelerated by incoming collisions. This means that you can use the standard 3D colliders to create a scene around which the controller will walk but you are not limited by realistic physical behaviour on the character itself.