

C#
(C Sharp)

Accessing Other Game Objects

Via Inspector

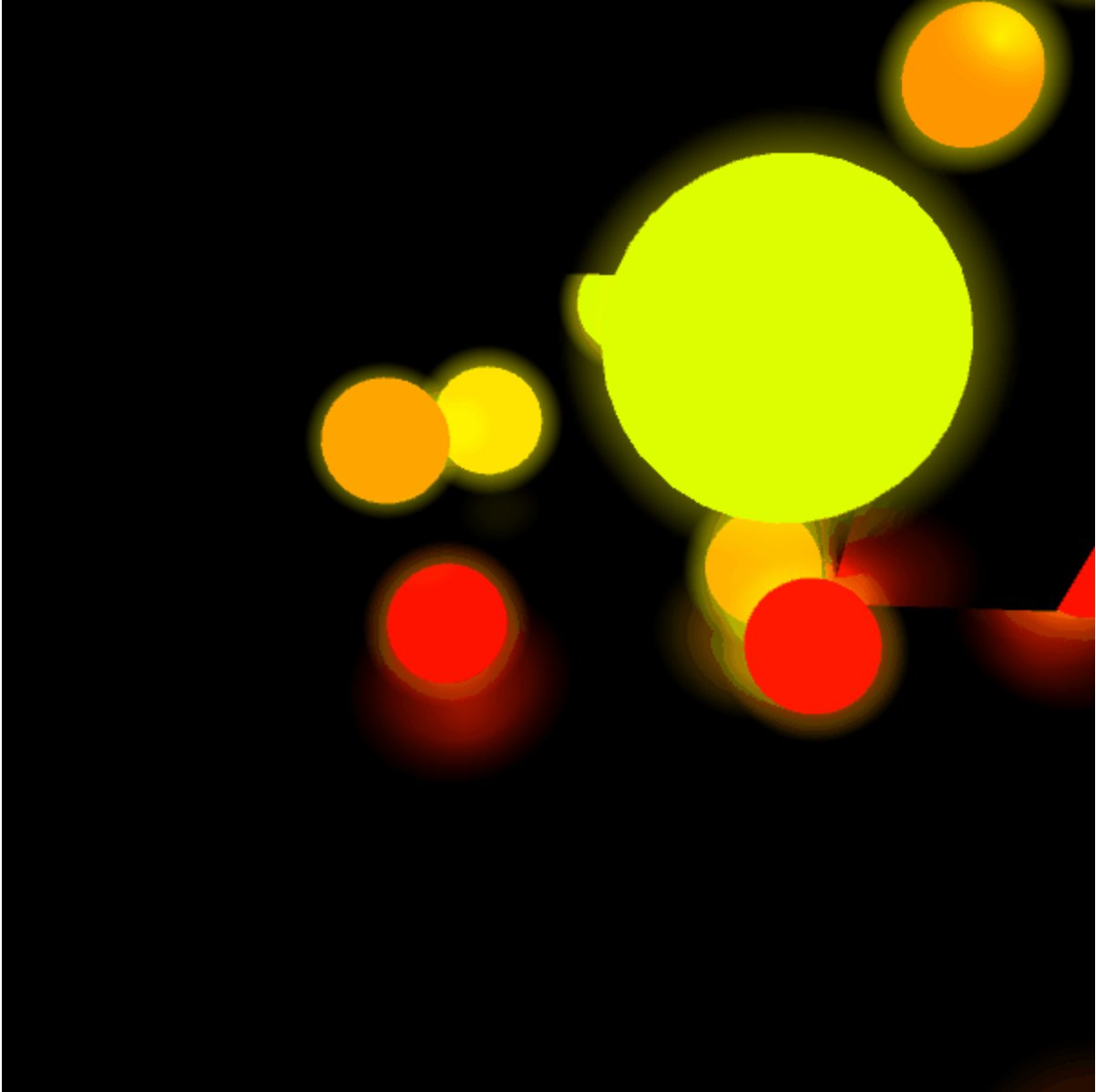
```
public class Script04_Distance : MonoBehaviour {  
  
    public Transform PlayerTransform;  
  
    // Use this for initialization  
    void Start () {  
  
    }  
  
    // Update is called once per frame  
    void Update () {  
  
    }  
}
```



Via Scripting

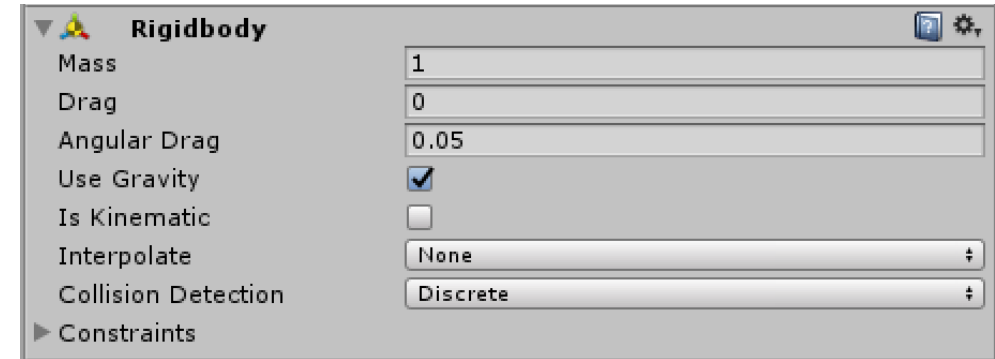
```
public class Script04_Distance : MonoBehaviour {  
  
    private Transform PlayerTransform;  
  
    // Use this for initialization  
    void Start () {  
  
        GameObject player = GameObject.Find("RigidBodyFPSController");  
        PlayerTransform = player.transform;  
  
    }  
  
    // Update is called once per frame  
    void Update () {  
  
    }  
  
}
```

Popcorn Lights

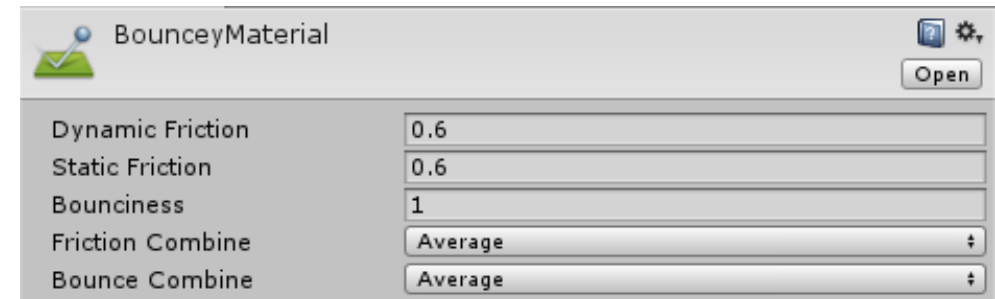


Bouncy Rigidbodies

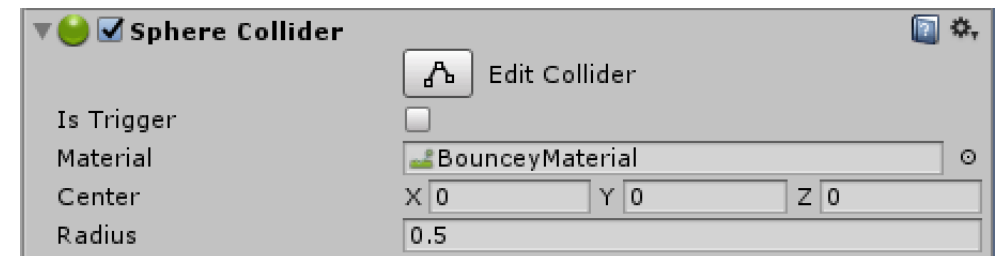
1. Add Rigidbody component



2. Create new "[Physic Material](#)" and set bounciness to 1



3. Apply "Physic Material" to the collider



Getting Rigidbody

```
private Rigidbody RigidbodyComponent;  
  
// Use this for initialization  
void Start () {  
    RigidbodyComponent = GetComponent<Rigidbody>();  
}
```


Applying a Force

```
Vector3 force = new Vector3(1f, 3f, 2f);  
RigidbodyComponent.AddForce(force, ForceMode.Impulse);
```

Rigidbody.AddForce

SWITCH TO MANUAL

```
public void AddForce(Vector3 force, ForceMode mode = ForceMode.Force);
```

Parameters

force	Force vector in world coordinates.
mode	Type of force to apply.

Description

Adds a force to the [Rigidbody](#).

Force is applied continuously along the direction of the force vector. Specifying the [ForceMode](#) mode allows the type of force to be changed to an Acceleration, Impulse or Velocity Change. Force can be applied only to an active Rigidbody. If a GameObject is inactive, AddForce has no effect.

ForceMode.Impulse

Description

Add an instant force impulse to the rigidbody, using its mass.

Apply the impulse force instantly with a single function call. This mode depends on the mass of rigidbody so more force must be applied to push or twist higher-mass objects the same amount as lower-mass objects. This mode is useful for applying forces that happen instantly, such as forces from explosions or collisions. In this mode, the unit of the force parameter is applied to the rigidbody as $\text{mass} \times \text{distance} / \text{time}$.

<https://docs.unity3d.com/ScriptReference/Rigidbody.AddForce.html>



Detecting Collisions

MonoBehaviour

class in UnityEngine / Inherits from: [Behaviour](#)

Messages

[OnCollisionEnter](#)

OnCollisionEnter is called when this collider/rigidbody has begun touching another rigidbody/collider.

[OnCollisionEnter2D](#)

Sent when an incoming collider makes contact with this object's collider (2D physics only).

[OnCollisionExit](#)

OnCollisionExit is called when this collider/rigidbody has stopped touching another rigidbody/collider.

[OnCollisionExit2D](#)

Sent when a collider on another object stops touching this object's collider (2D physics only).

[OnCollisionStay](#)

OnCollisionStay is called once per frame for every collider/rigidbody that is touching rigidbody/collider.

[OnCollisionStay2D](#)

Sent each frame where a collider on another object is touching this object's collider (2D physics only).



MonoBehaviour.OnCollisionEnter(Collision)

Parameters

other

The Collision data associated with this collision.

Description

OnCollisionEnter is called when this collider/rigidbody has begun touching another rigidbody/collider.

In contrast to OnTriggerEnter, OnCollisionEnter is passed the [Collision](#) class and not a [Collider](#). The [Collision](#) class contains information about contact points, impact velocity etc. If you don't use collisionInfo in the function, leave out the collisionInfo parameter as this avoids unnecessary calculations. Notes: Collision events are only sent if one of the colliders also has a non-kinematic rigidbody attached. Collision events will be sent to disabled MonoBehaviours, to allow enabling Behaviours in response to collisions.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    AudioSource audio;

    void Start() {
        audio = GetComponent<AudioSource>();
    }

    void OnCollisionEnter(Collision collision) {
        foreach (ContactPoint contact in collision.contacts) {
            Debug.DrawRay(contact.point, contact.normal, Color.white);
        }

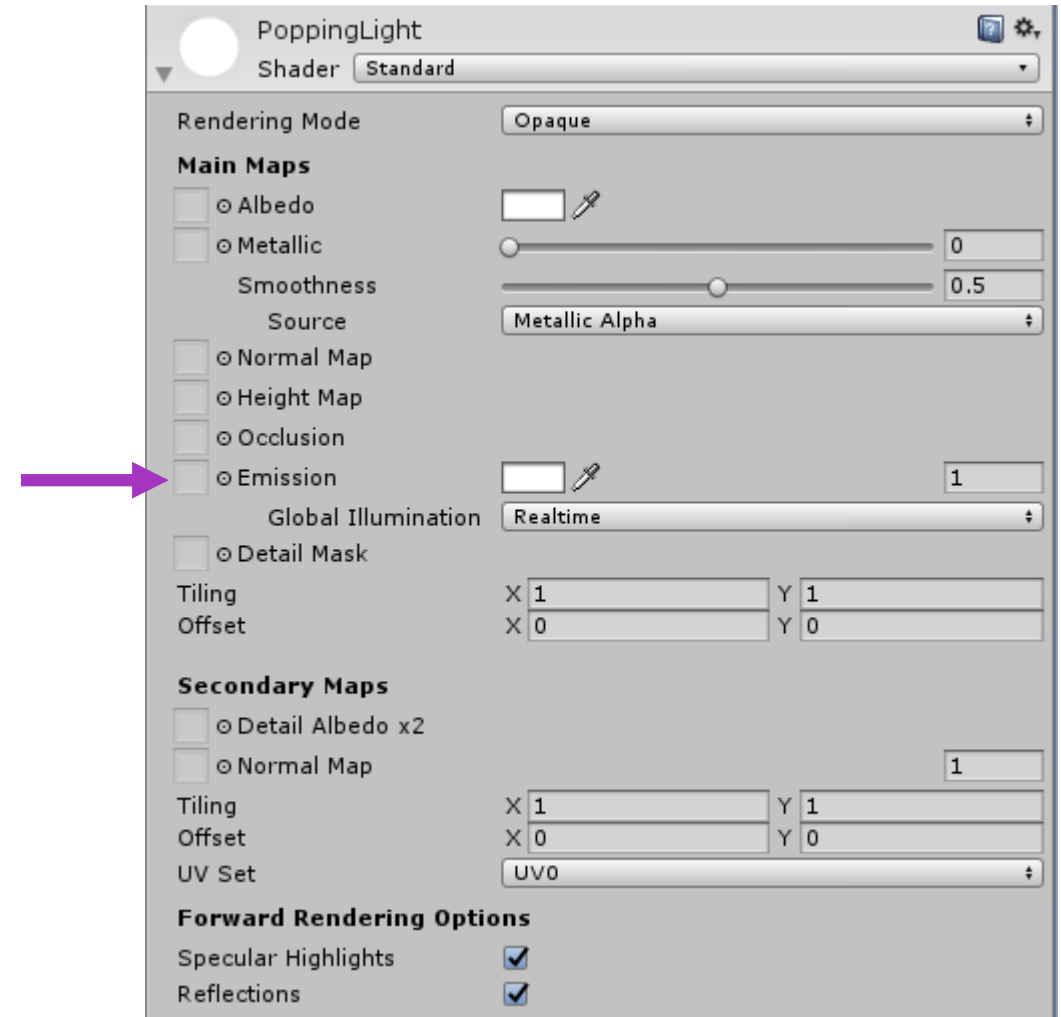
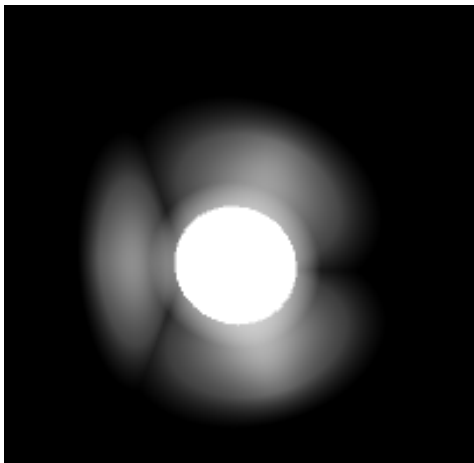
        if (collision.relativeVelocity.magnitude > 2)
            audio.Play();
    }
}
```

Prefabs

- A way to create linked copies of objects
- Watch [Unity tutorial](#)

Real-time “Light Bulb” Effect

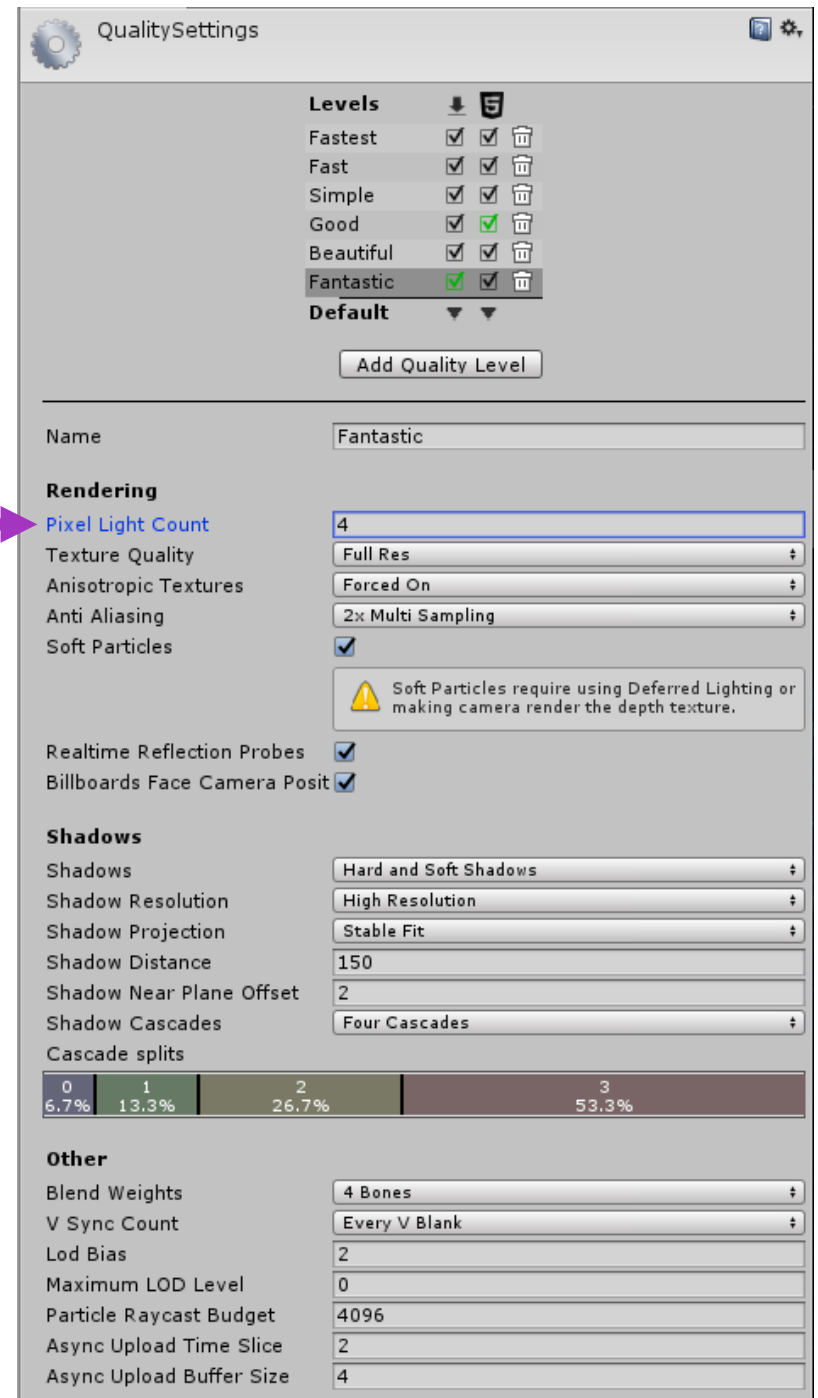
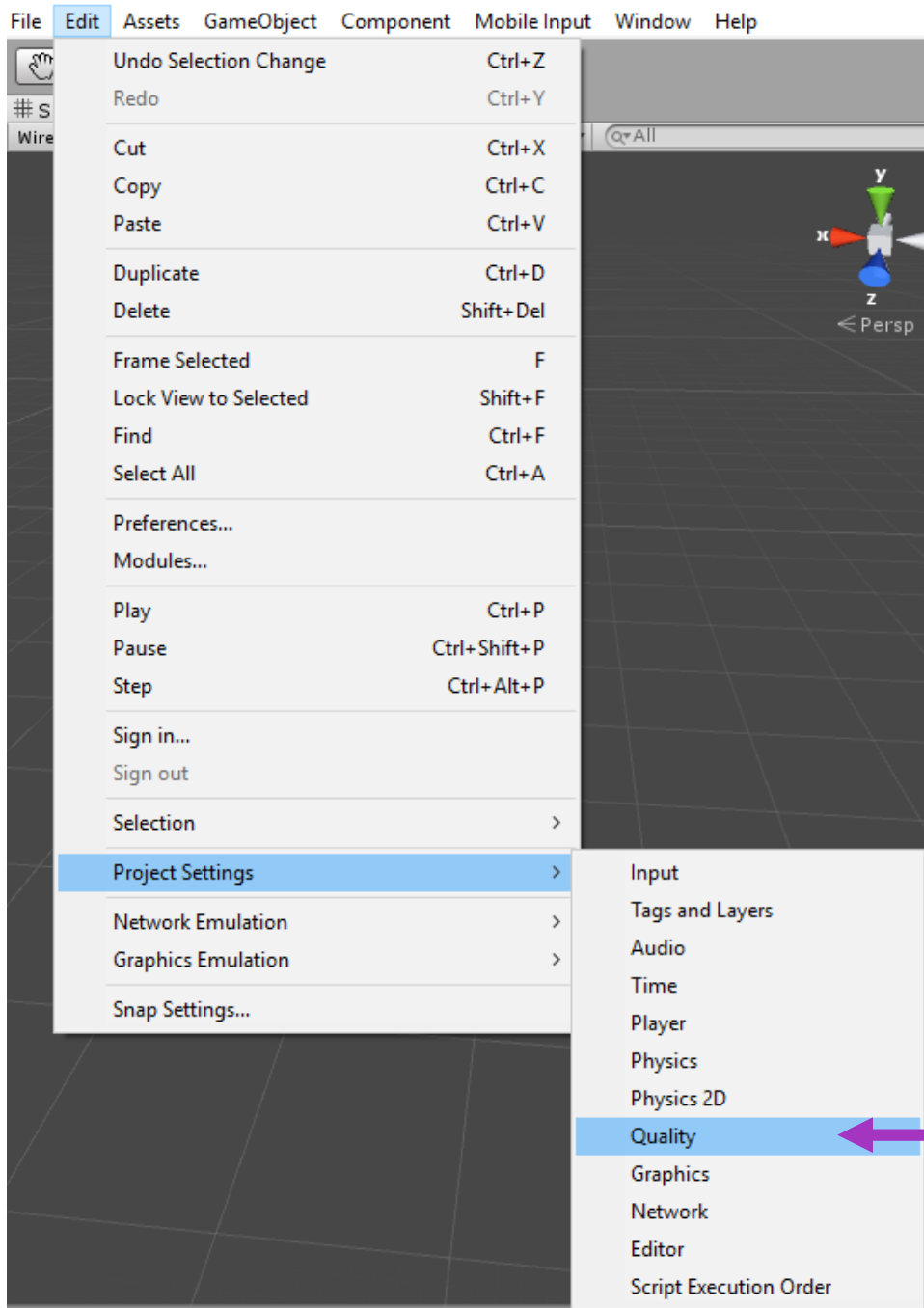
- Two components
 - **Emissive material** – creates the appearance of an illuminated surface
 - **A light source** – creates the cast light from the object





Limits on Real-time Lights

- Only a set number (4) of “pixel light” sources are allowed to illuminate an object
- More than 4 light sources – the least “important” ones are rendered using “vertex lighting”
- More info: [Unity](#)





Material.SetColor

SWITCH TO MANUAL

```
public void SetColor(string propertyName, Color color);
```

```
public void SetColor(int nameID, Color color);
```

Parameters

propertyName	Property name, e.g. "_Color".
nameID	Property name ID, use Shader.PropertyToID to get it.
color	Color value to set.

Description

Set a named color value.

Many shaders use more than one color. Use SetColor to change the color (identified by shader property name, or unique property name ID).

When setting color values on materials using the Standard Shader, you should be aware that you may need to use [EnableKeyword](#) to enable features of the shader that were not previously in use. For more detail, read [Accessing Materials via Script](#).

Common color names used by Unity's builtin shaders:

"_Color" is the main color of a material. This can also be accessed via [color](#) property.

"_EmissionColor" is the emissive color of a material.

