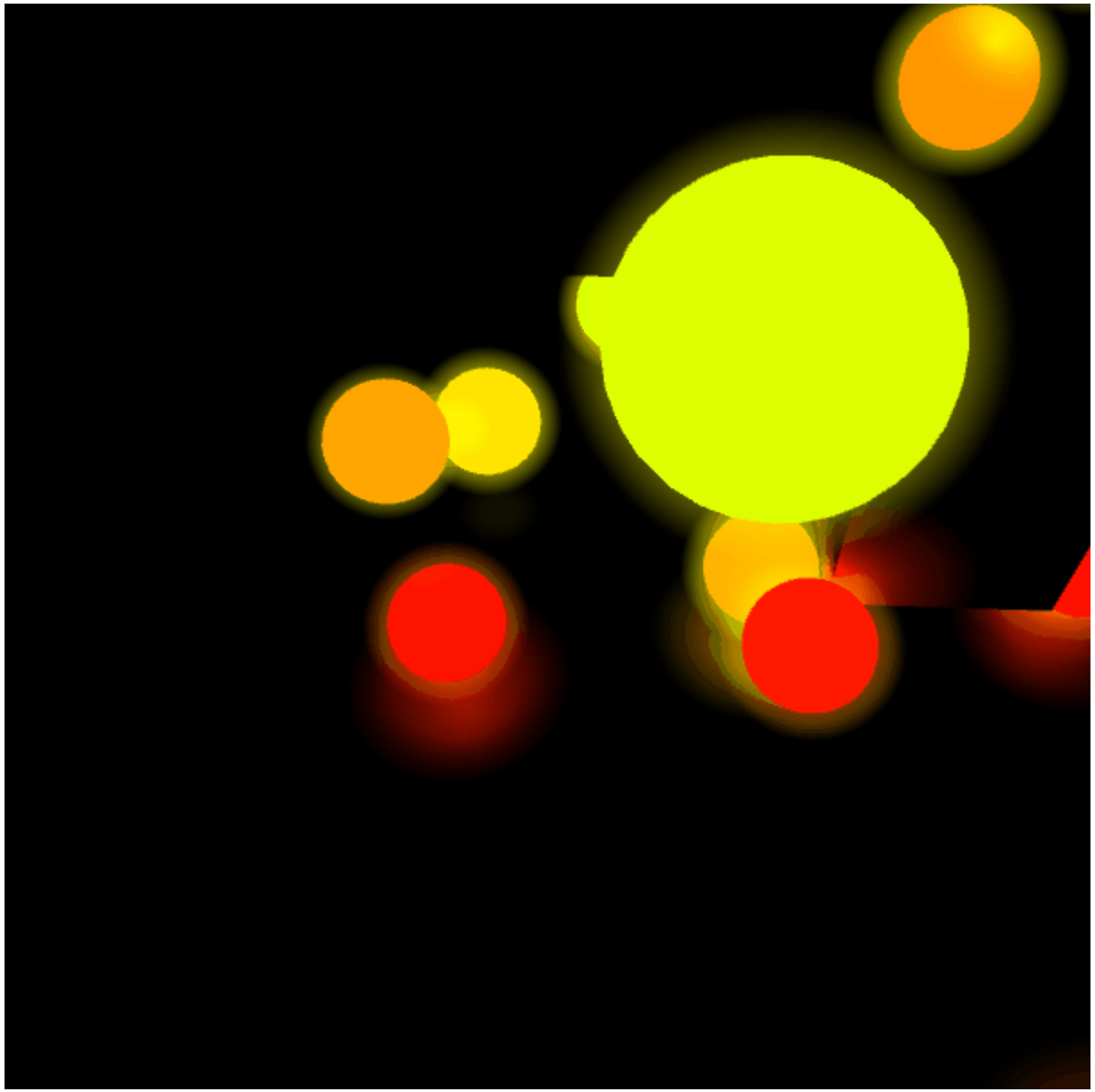
C#

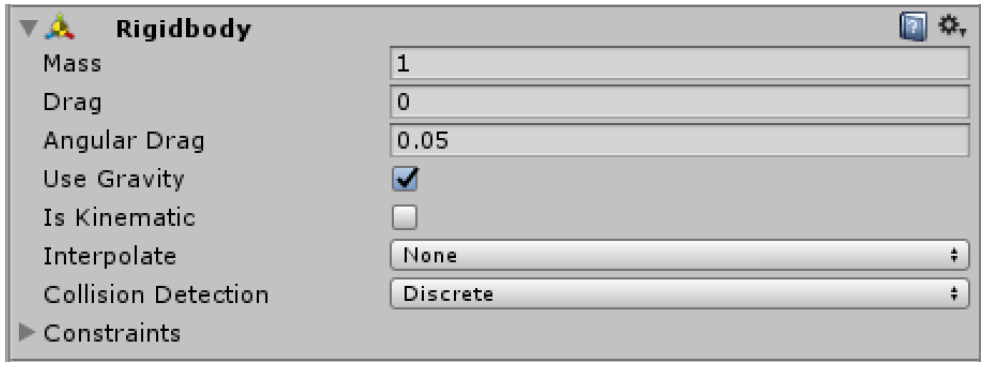
(C Sharp)

Popcorn Lights

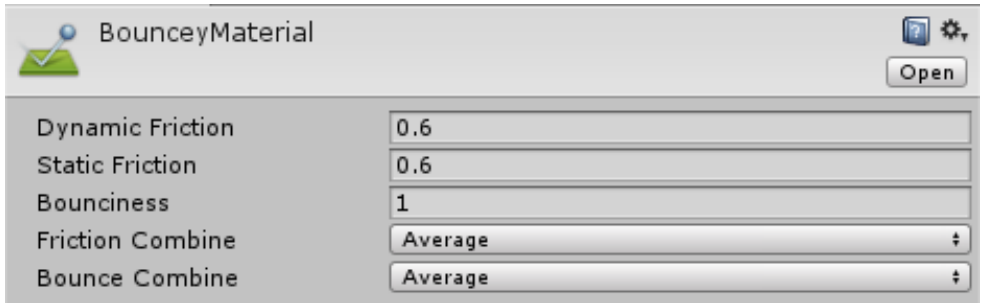


Bouncy Rigidbodies

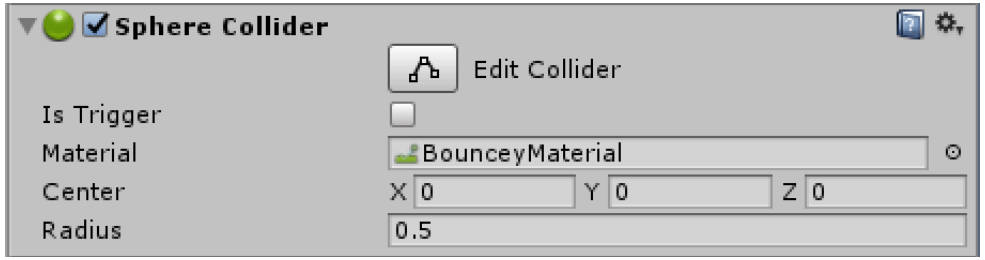
1. Add Rigidbody component



2. Create new "Physic Material" and set bounciness to 0.8



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.

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

ForceMode

enumeration

Description

Option for how to apply a force using Rigidbody.AddForce.

Variables

Force	Add a continuous force to the rigidbody, using its mass.
Acceleration	Add a continuous acceleration to the rigidbody, ignoring its mass.
Impulse	Add an instant force impulse to the rigidbody, using its mass.
VelocityChange	Add an instant velocity change to the rigidbody, ignoring its mass.

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 mass*distance/time.

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).

https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

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 unneccessary 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();
    }
}
```

MonoBehaviour.Invoke

public void **Invoke**(string **methodName**, float **time**);

Description

Invokes the method `methodName` in time seconds.

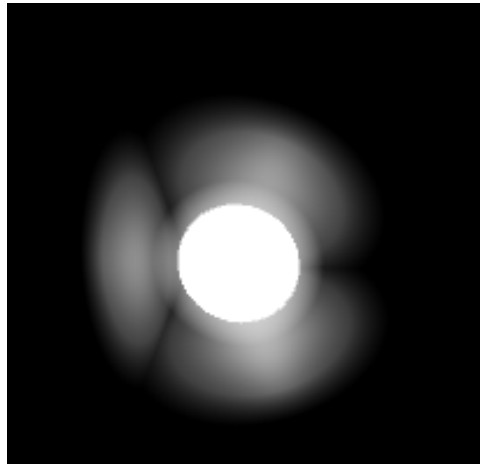
```
using UnityEngine;
using System.Collections.Generic;

public class ExampleScript : MonoBehaviour {

    // Launches a projectile in 2 seconds

    Rigidbody projectile;

    void Start() {
        Invoke("LaunchProjectile", 2);
    }

    void LaunchProjectile () {
        Rigidbody instance = Instantiate(projectile);
        instance.velocity = Random.insideUnitSphere * 5;
    }

}
```

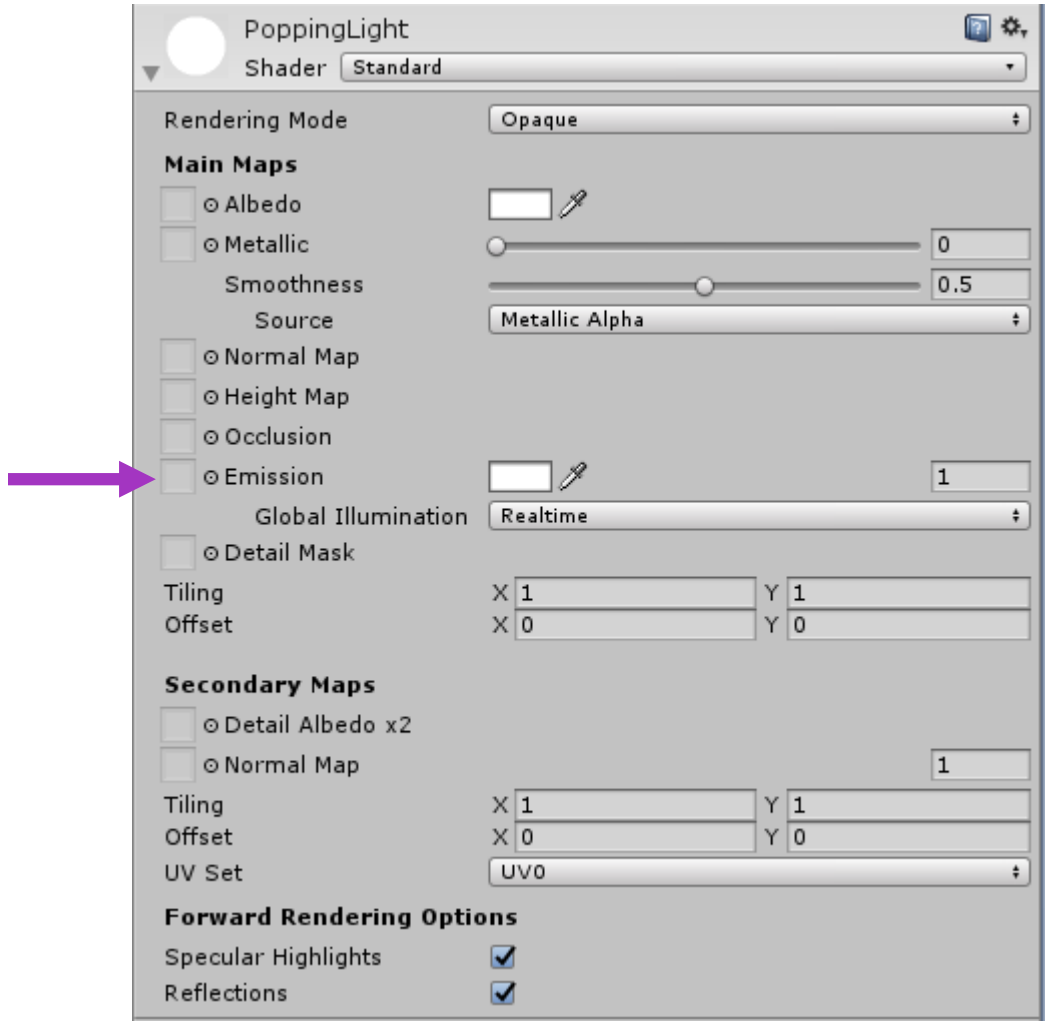
Prefabs

- A way to create linked copies of objects
- Watch Unity tutorial

Real-time “Light Bulb” Effect

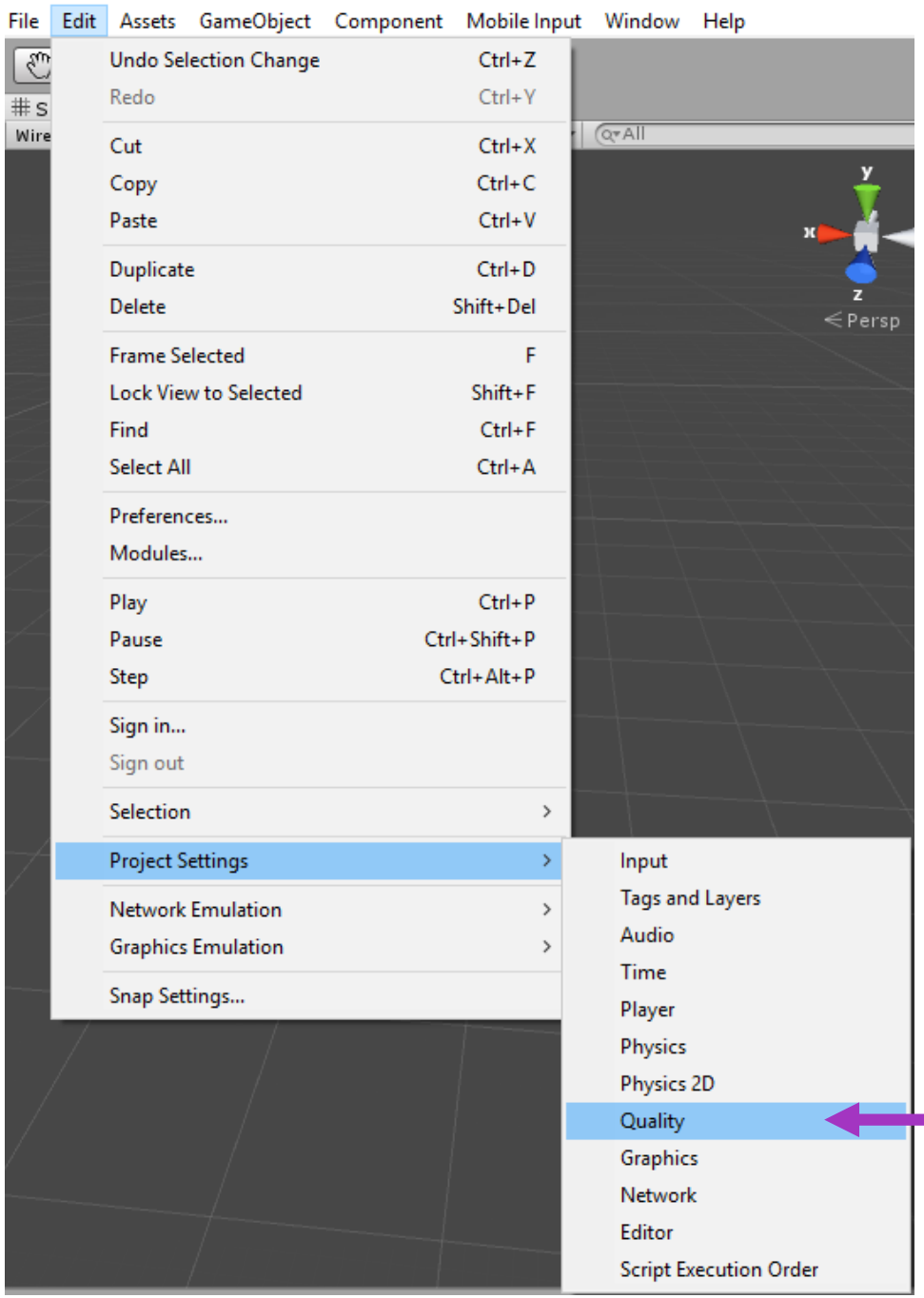
- Two components
  - **Emissive material** – creates the appearance of an illuminated surface
  - **A light source** – creates the cast light from the object

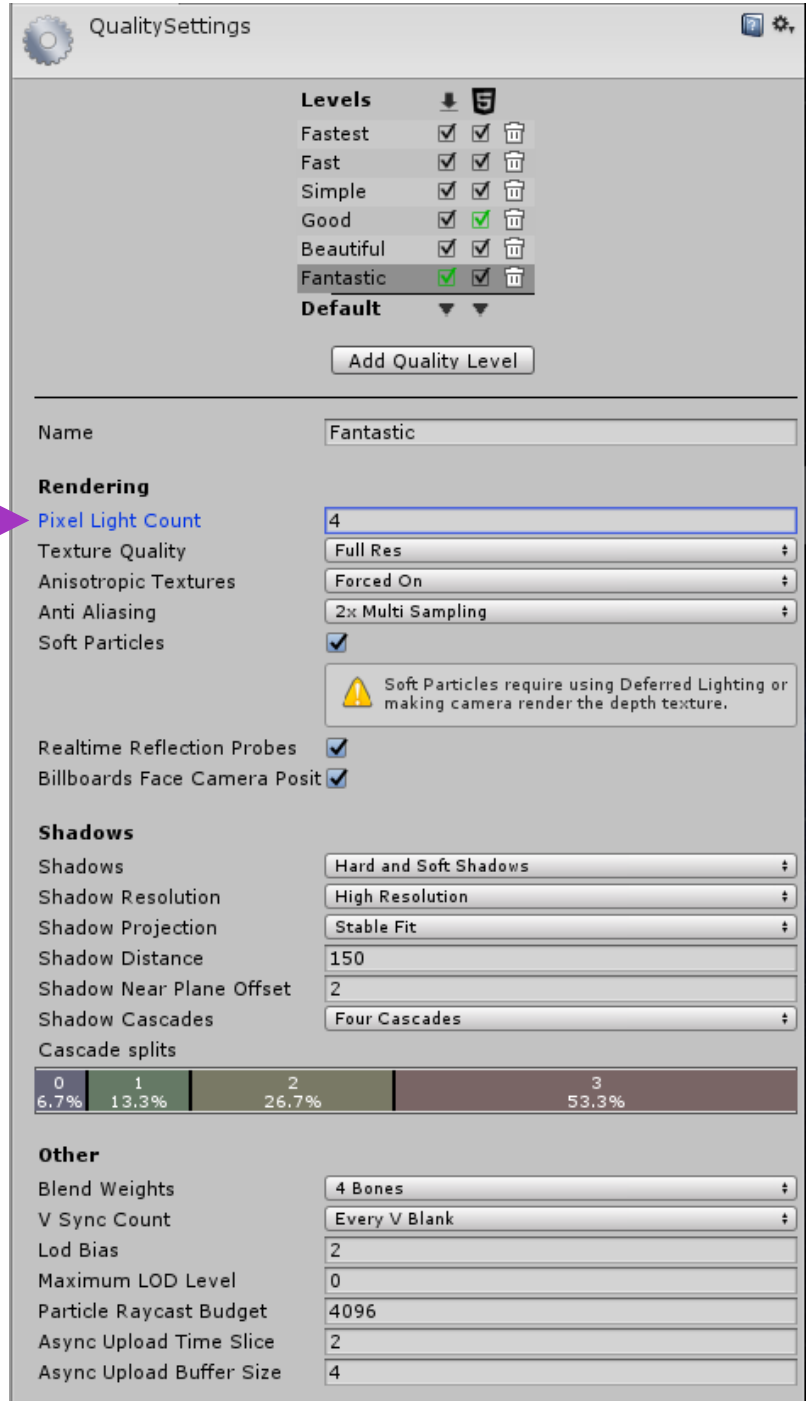




Limits on Real-time Lights

- 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 render using “vertex lighting”
- More info: Unity


File
Edit
Assets
GameObject
Component
Mobile Input
Window
Help
Undo Selection Change Ctrl+Z
Redo Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Duplicate Ctrl+D
Delete Shift+Del
Frame Selected F
Lock View to Selected Shift+F
Find Ctrl+F
Select All Ctrl+A
Preferences...
Modules...
Play Ctrl+P
Pause Ctrl+Shift+P
Step Ctrl+Alt+P
Sign in...
Sign out
Selection
Project Settings
Network Emulation
Graphics Emulation
Snap Settings...
Input
Tags and Layers
Audio
Time
Player
Physics
Physics 2D
Quality
Graphics
Network
Editor
Script Execution Order
Persp


QualitySettings
Levels
Fastest
Fast
Simple
Good
Beautiful
Fantastic
Default
Add Quality Level
Name Fantastic
Rendering
Pixel Light Count 4
Texture Quality Full Res
Anisotropic Textures Forced On
Anti Aliasing 2x Multi Sampling
Soft Particles
Soft Particles require using Deferred Lighting or making camera render the depth texture.
Realtime Reflection Probes
Billboards Face Camera Posit
Shadows
Shadows Hard and Soft Shadows
Shadow Resolution High Resolution
Shadow Projection Stable Fit
Shadow Distance 150
Shadow Near Plane Offset 2
Shadow Cascades Four Cascades
Cascade splits
0 6.7%
1 13.3%
2 26.7%
3 53.3%
Other
Blend Weights 4 Bones
V Sync Count Every V Blank
Lod Bias 2
Maximum LOD Level 0
Particle Raycast Budget 4096
Async Upload Time Slice 2
Async Upload Buffer Size 4

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.

https://docs.unity3d.com/ScriptReference/Material.SetColor.html

Loops!

For Loop

```
for (int i = 0; i < 10; i += 1) {
    Debug.Log("Counting: " + i);
}
```

INITIALIZATION CONDITION INCREMENT

```
for (int i = 0; i < 10; i += 1) {
    Debug.Log("Counting: " + i);
}
```

Loop Flow

1

```
for (int i = 0; i < 2; i += 1) {
    Debug.Log("Counting: " + i);
}
```

1 → 2

```
for (int i = 0; i < 2; i += 1) {
    Debug.Log("Counting: " + i);
}
```

2

3

```
for (int i = 0; i < 2; i += 1) {
    Debug.Log("Counting: " + i);
}
```

```
for (int i = 0; i < 2; i += 1) {
    Debug.Log("Counting: " + i);
}
```

5 ← 4

```
for (int i = 0; i < 2; i += 1) {
    Debug.Log("Counting: " + i);
}
```

5

```
for (int i = 0; i < 2; i += 1) {
    Debug.Log("Counting: " + i);
}
```

6

```
for (int i = 0; i < 2; i += 1) {
    Debug.Log("Counting: " + i);
}
```

8 ← 7

```
for (int i = 0; i < 2; i += 1) {
    Debug.Log("Counting: " + i);
}
```

8

```
for (int i = 0; i < 2; i += 1) {
    Debug.Log("Counting: " + i);
}
```

LOOP OVER

More Loops

- Unity tutorial

Instantiate??

Object.Instantiate

public static Object **Instantiate**(Object **original**);
public static Object **Instantiate**(Object **original**, Transform **parent**);
public static Object **Instantiate**(Object **original**, Transform **parent**, bool **worldPositionStays**);
public static Object **Instantiate**(Object **original**, Vector3 **position**, Quaternion **rotation**);
public static Object **Instantiate**(Object **original**, Vector3 **position**, Quaternion **rotation**, Transform **parent**);

Parameters

original	An existing object that you want to make a copy of.
position	Position for the new object (default Vector3.zero).
rotation	Orientation of the new object (default Quaternion.identity).
parent	The transform the object will be parented to.
worldPositionStays	If when assigning the parent the original world position should be maintained.

Returns

Object A clone of the original object.

https://docs.unity3d.com/ScriptReference/Object.Instantiate.html