# C#

(C Sharp)

# Classes and Instances

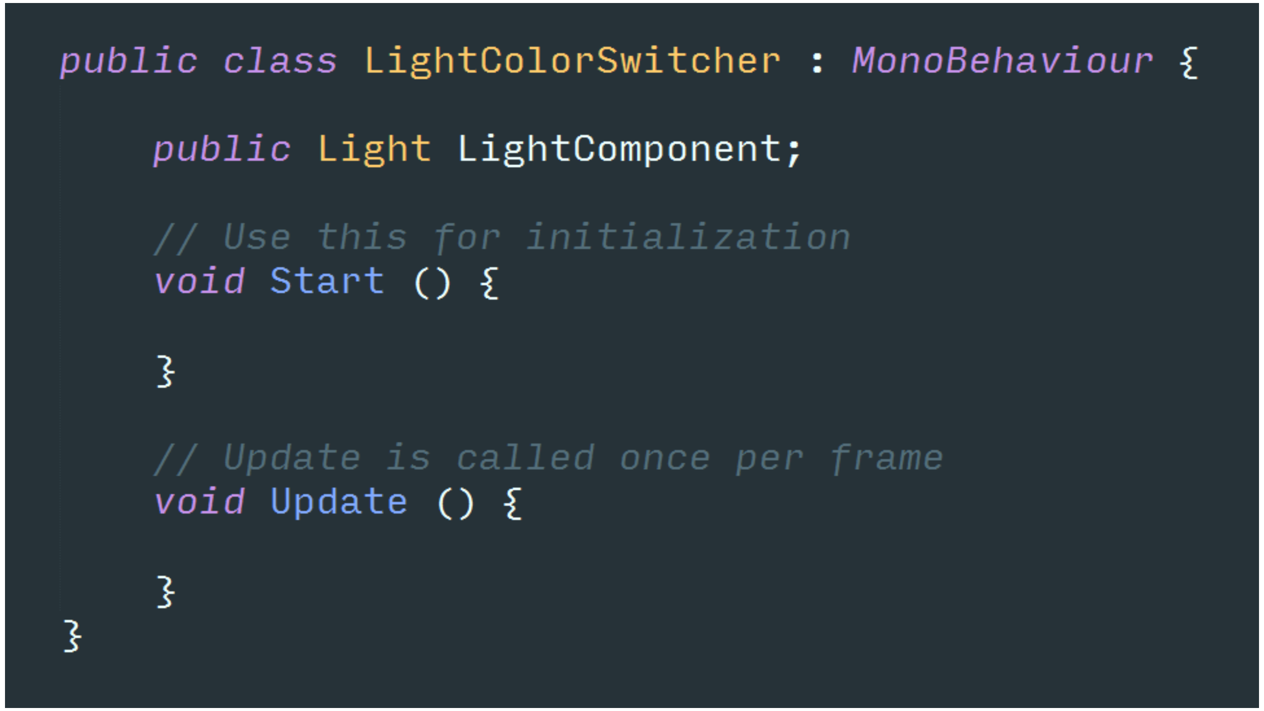
(OOP)

# Classes

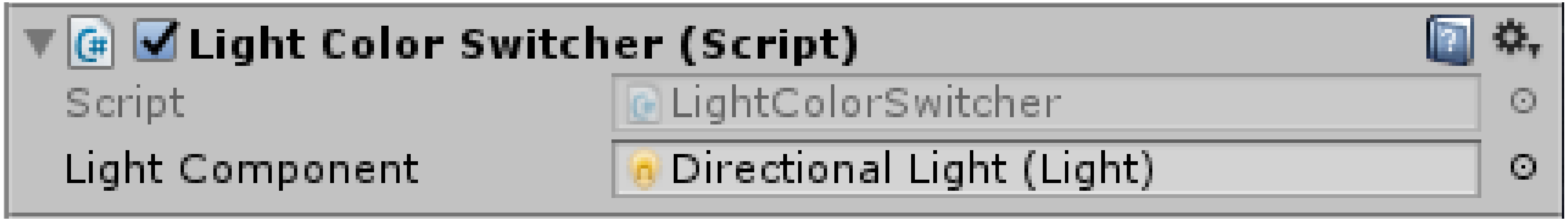
- Encapsulation: organize variables and functions together
- Nearly everything in C#/Unity is a class!

# Accessing Components

# Via Inspector



```
public class LightColorSwitcher : MonoBehaviour {

    public Light LightComponent;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

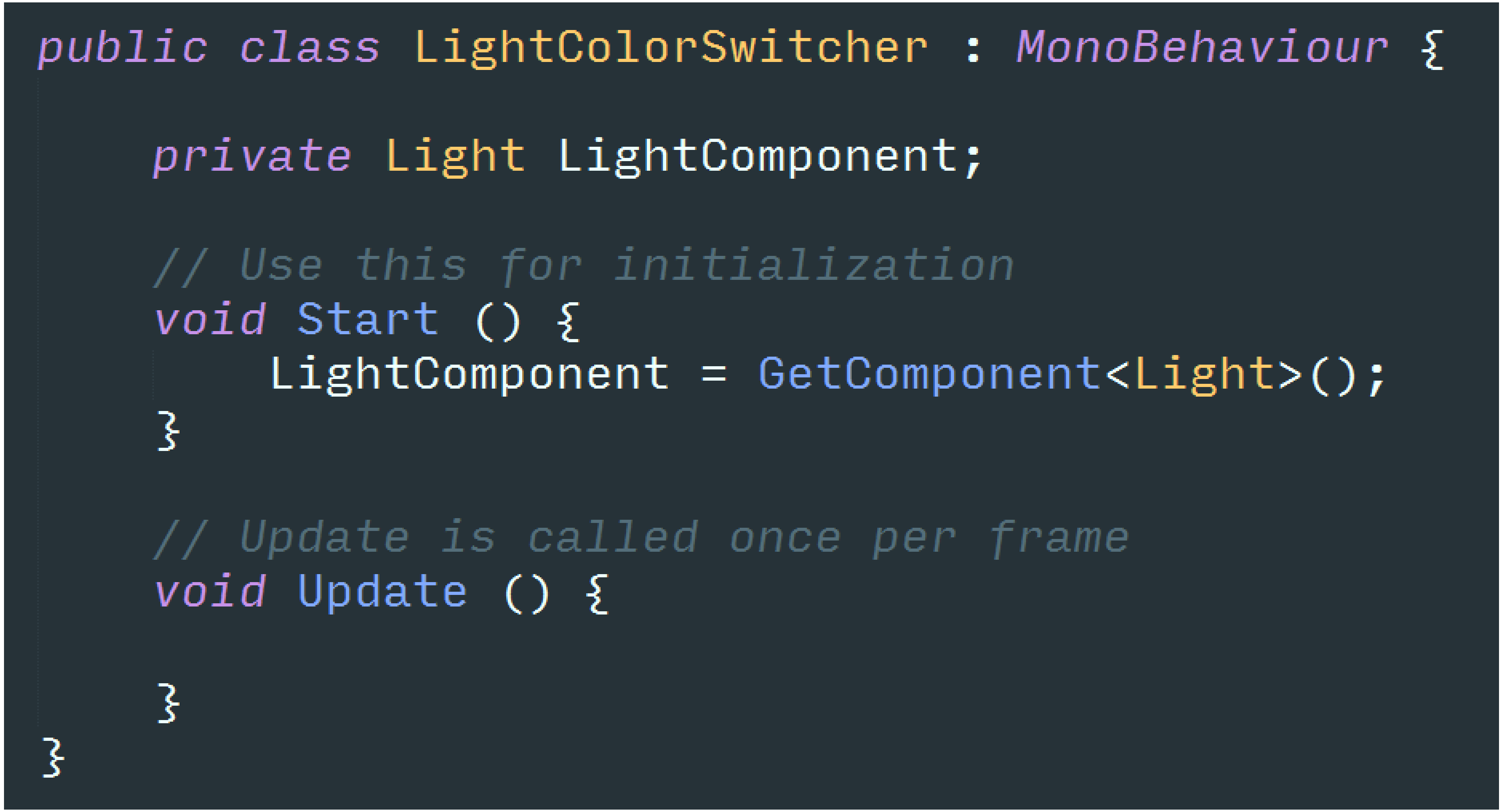


☑ Light Color Switcher (Script)

Script: LightColorSwitcher

Light Component: Directional Light (Light)

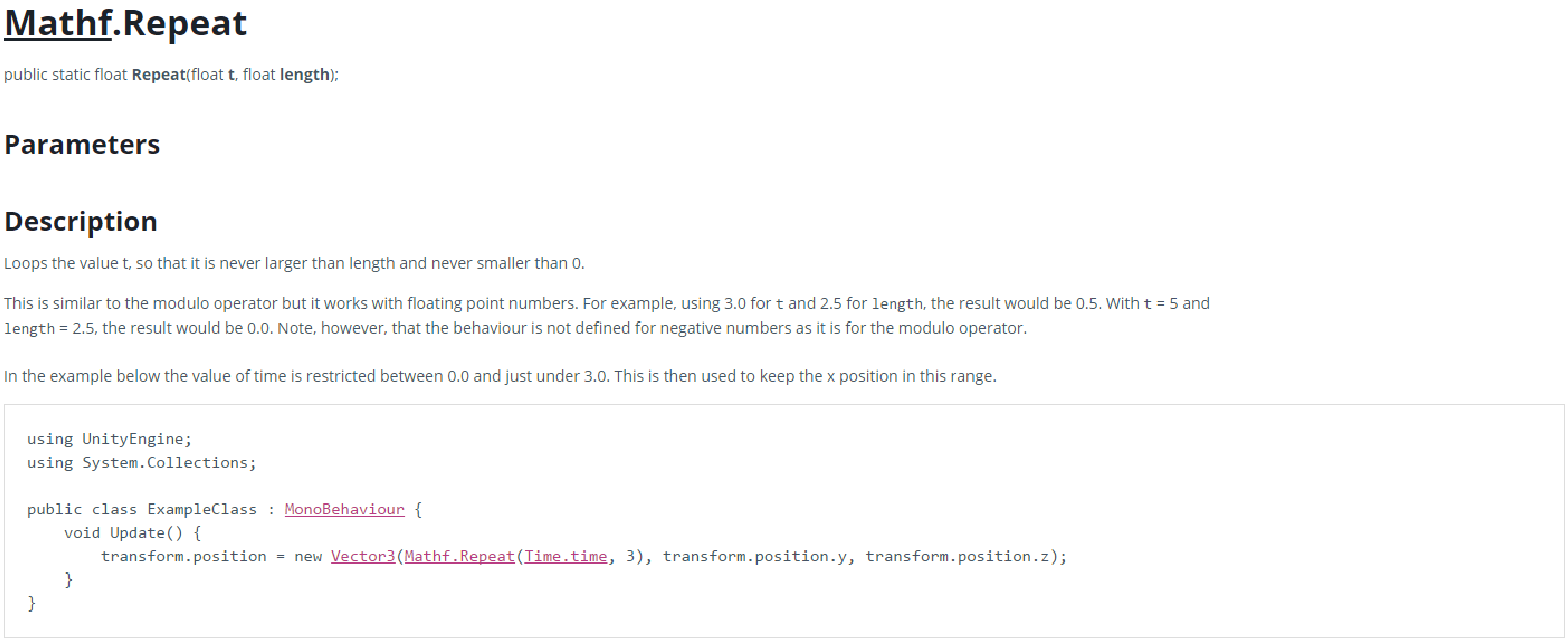
# Via Scripting



```
public class LightColorSwitcher : MonoBehaviour {

    private Light LightComponent;

    // Use this for initialization
    void Start () {
        LightComponent = GetComponent<Light>();
    }

    // Update is called once per frame
    void Update () {

    }
}
```

# Mathf

http://docs.unity3d.com/ScriptReference/Mathf.html



# Mathf.Repeat

public static float **Repeat**(float **t**, float **length**);

## Parameters

## Description

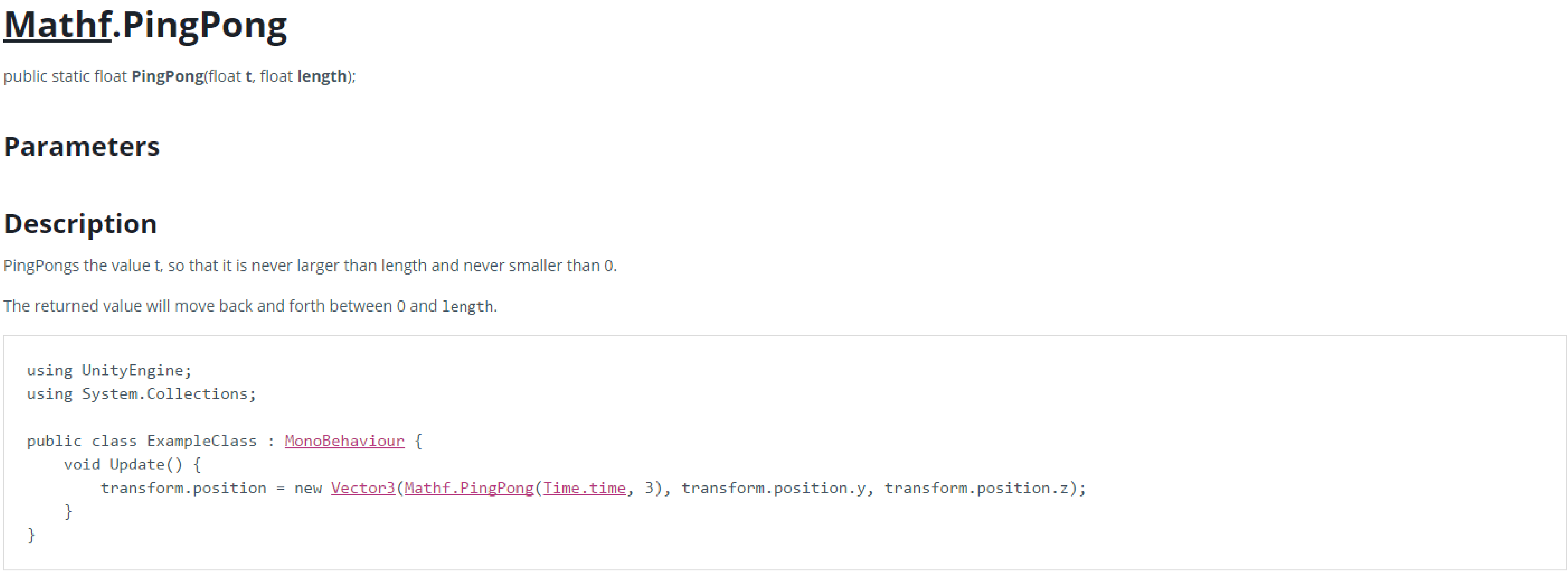
Loops the value t, so that it is never larger than length and never smaller than 0.

This is similar to the modulo operator but it works with floating point numbers. For example, using 3.0 for `t` and 2.5 for `length`, the result would be 0.5. With `t` = 5 and `length` = 2.5, the result would be 0.0. Note, however, that the behaviour is not defined for negative numbers as it is for the modulo operator.

In the example below the value of time is restricted between 0.0 and just under 3.0. This is then used to keep the x position in this range.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.position = new Vector3(Mathf.Repeat(Time.time, 3), transform.position.y, transform.position.z);
    }
}
```

http://docs.unity3d.com/ScriptReference/Mathf.Repeat.html



# Mathf.PingPong

public static float **PingPong**(float **t**, float **length**);

## Parameters

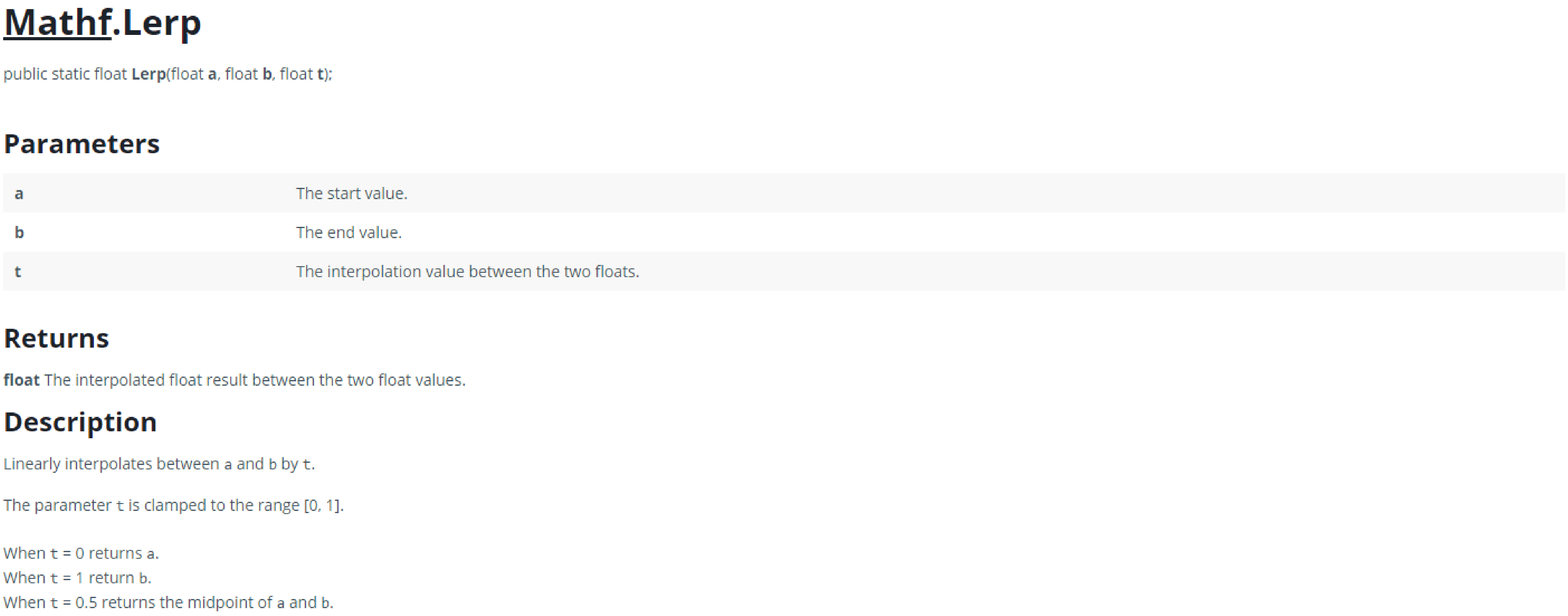
## Description

PingPongs the value t, so that it is never larger than length and never smaller than 0.

The returned value will move back and forth between 0 and `length`.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.position = new Vector3(Mathf.PingPong(Time.time, 3), transform.position.y, transform.position.z);
    }
}
```

http://docs.unity3d.com/ScriptReference/Mathf.PingPong.html



# Mathf.Lerp

public static float **Lerp**(float **a**, float **b**, float **t**);

## Parameters

| | |
|---|---|
| **a** | The start value. |
| **b** | The end value. |
| **t** | The interpolation value between the two floats. |

## Returns

**float** The interpolated float result between the two float values.

## Description

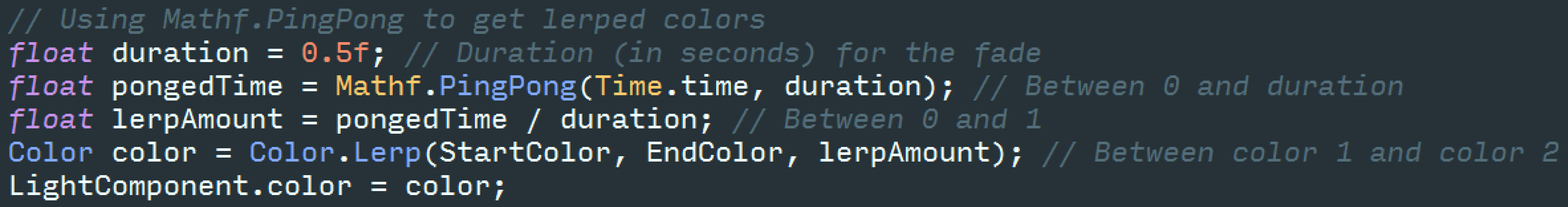
Linearly interpolates between `a` and `b` by `t`.

The parameter `t` is clamped to the range [0, 1].

When `t` = 0 returns `a`.
When `t` = 1 return `b`.
When `t` = 0.5 returns the midpoint of `a` and `b`.

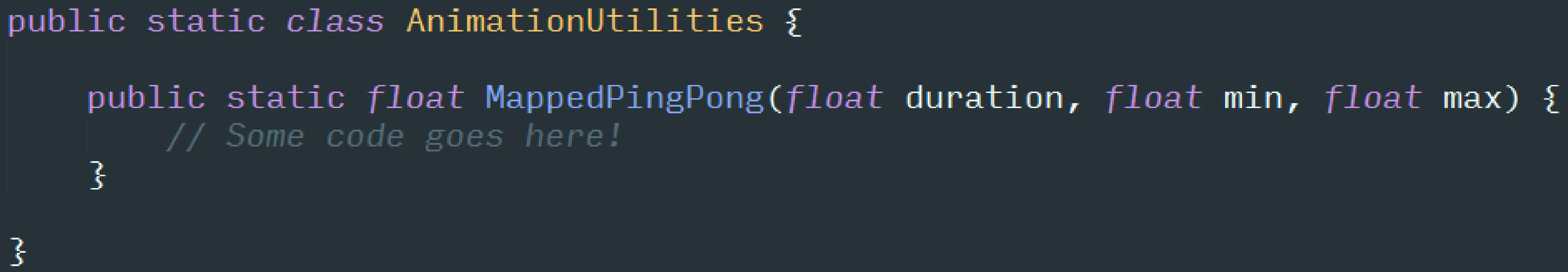
https://docs.unity3d.com/ScriptReference/Mathf.Lerp.html

```
// Using Mathf.PingPong to get intensities between 1 and 5
float duration = 2f; // Duration (in seconds) for the fade
float pongedTime = Mathf.PingPong(Time.time, duration); // Between 0 and duration
float lerpAmount = pongedTime / duration; // Between 0 and 1
float intensity = Mathf.Lerp(1, 5, lerpAmount); // Between 1 and 5
LightComponent.intensity = intensity;
```

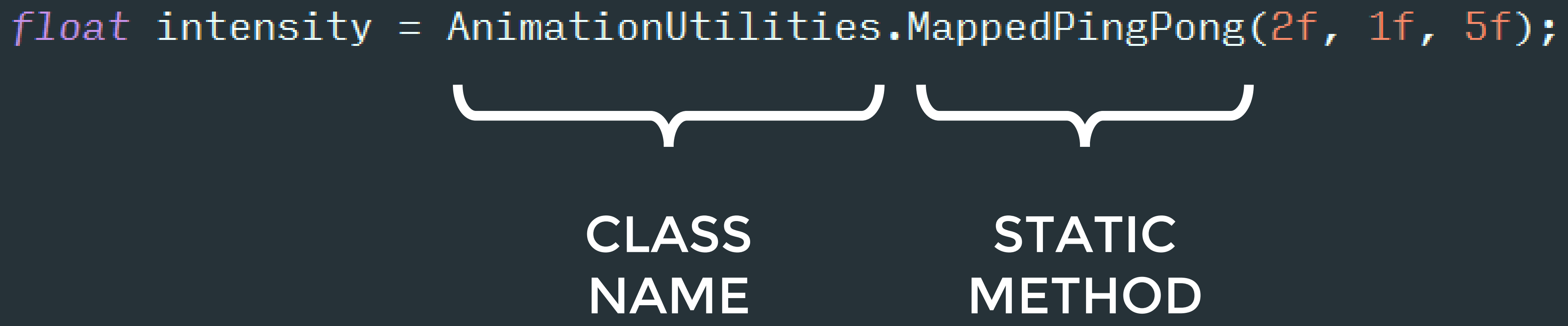


```
// Using Mathf.PingPong to get lerped colors
float duration = 0.5f; // Duration (in seconds) for the fade
float pongedTime = Mathf.PingPong(Time.time, duration); // Between 0 and duration
float lerpAmount = pongedTime / duration; // Between 0 and 1
Color color = Color.Lerp(StartColor, EndColor, lerpAmount); // Between color 1 and color 2
LightComponent.color = color;
```

# Static Classes & Methods



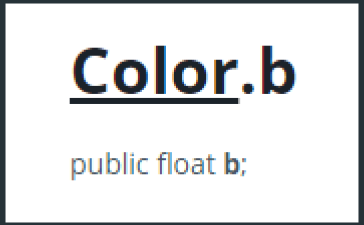
```
public static class AnimationUtilities {

    public static float MappedPingPong(float duration, float min, float max) {
        // Some code goes here!
    }

}
```



```
float intensity = AnimationUtilities.MappedPingPong(2f, 1f, 5f);
```

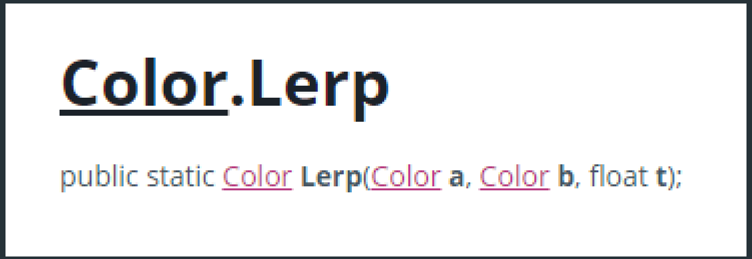
CLASS
NAME

STATIC
METHOD



## Color.b

public float **b**;



## Color.Lerp

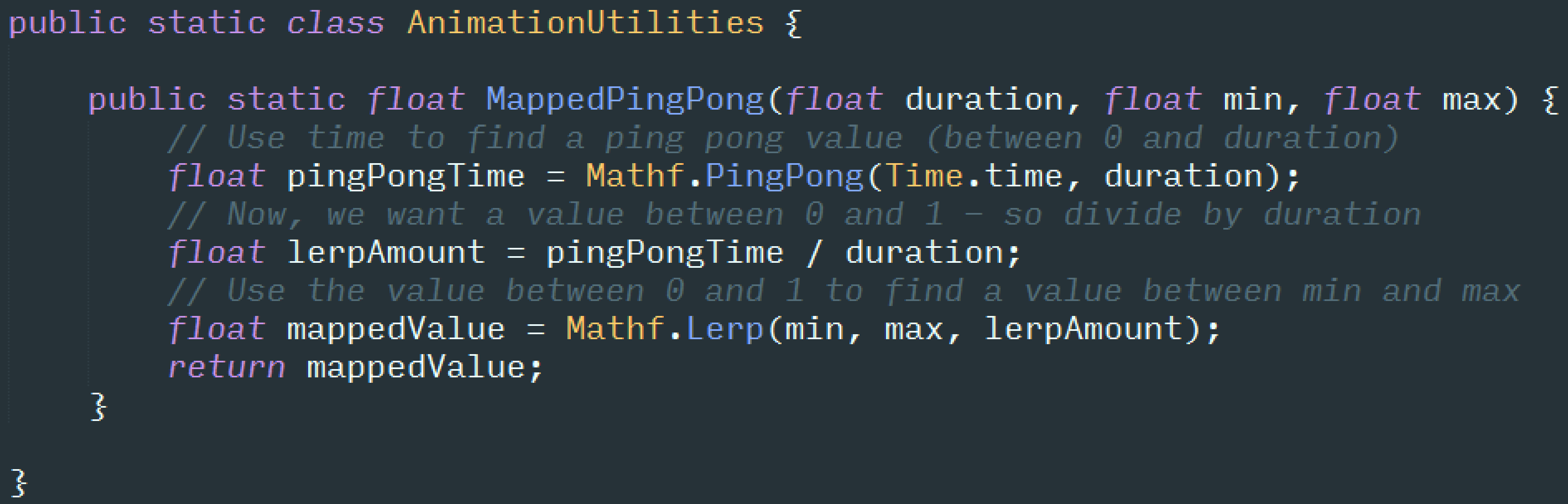
public static Color **Lerp**(Color **a**, Color **b**, float **t**);

```
Color c = new Color(1f, 0f, 0f);
c.b = 1f;
c.g = 0.1f;
```

INSTANCE
FIELD

```
Color c1 = new Color(1f, 0f, 0f);
Color c2 = new Color(0f, 0f, 1f);
Color.Lerp(c1, c2, 0.5f);
```

STATIC
METHOD

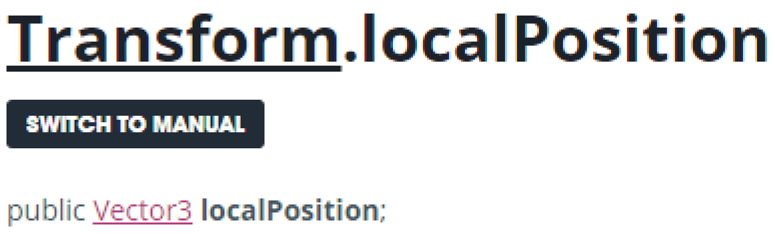


```csharp
public static class AnimationUtilities {

    public static float MappedPingPong(float duration, float min, float max) {
        // Use time to find a ping pong value (between 0 and duration)
        float pingPongTime = Mathf.PingPong(Time.time, duration);
        // Now, we want a value between 0 and 1 - so divide by duration
        float lerpAmount = pingPongTime / duration;
        // Use the value between 0 and 1 to find a value between min and max
        float mappedValue = Mathf.Lerp(min, max, lerpAmount);
        return mappedValue;
    }

}
```

# Vector3

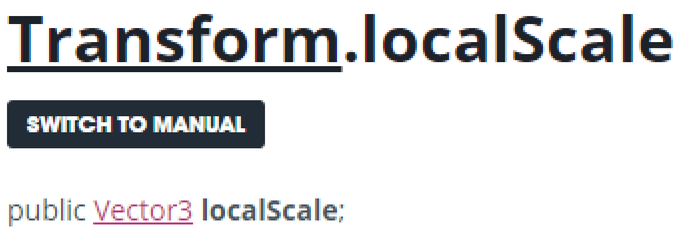
http://docs.unity3d.com/ScriptReference/Vector3.html

# Transform



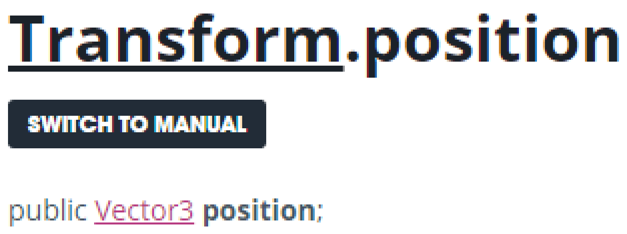
Transform.localPosition

```
public Vector3 localPosition;
```



Transform.localScale

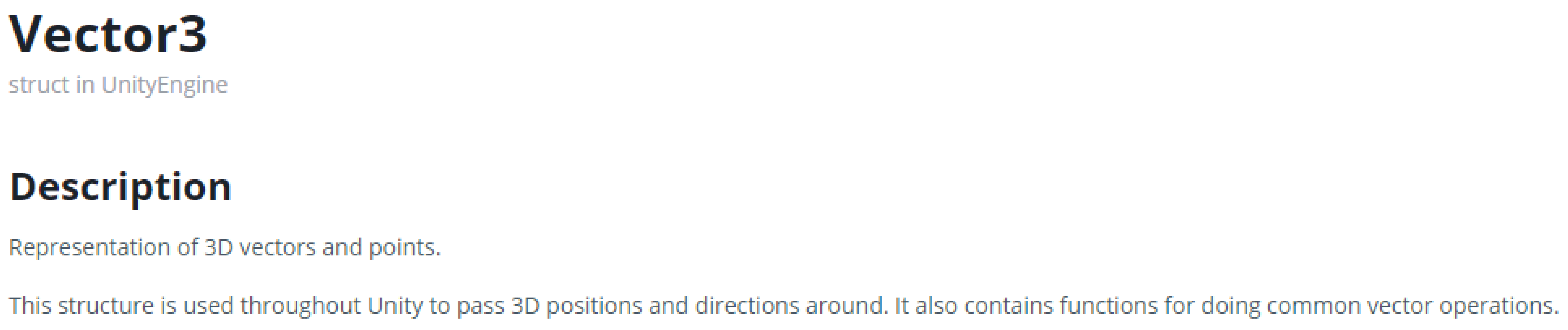
```
public Vector3 localScale;
```



Transform.position

```
public Vector3 position;
```

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



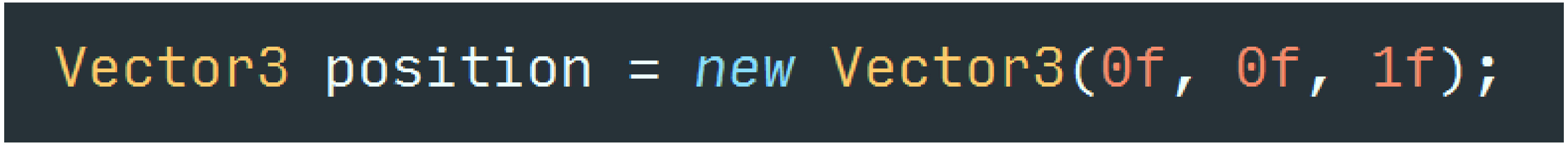
# Vector3

struct in UnityEngine

## Description

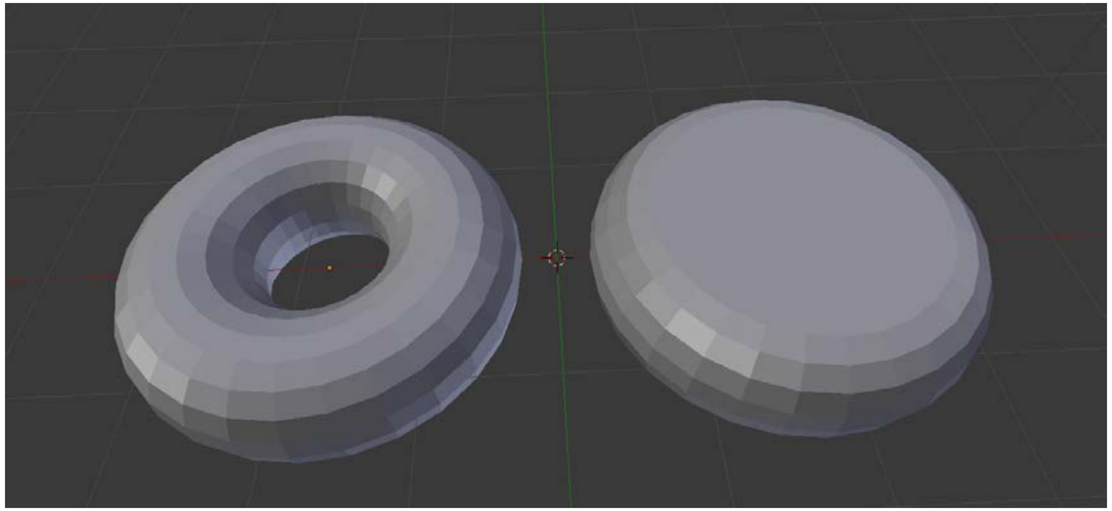
Representation of 3D vectors and points.

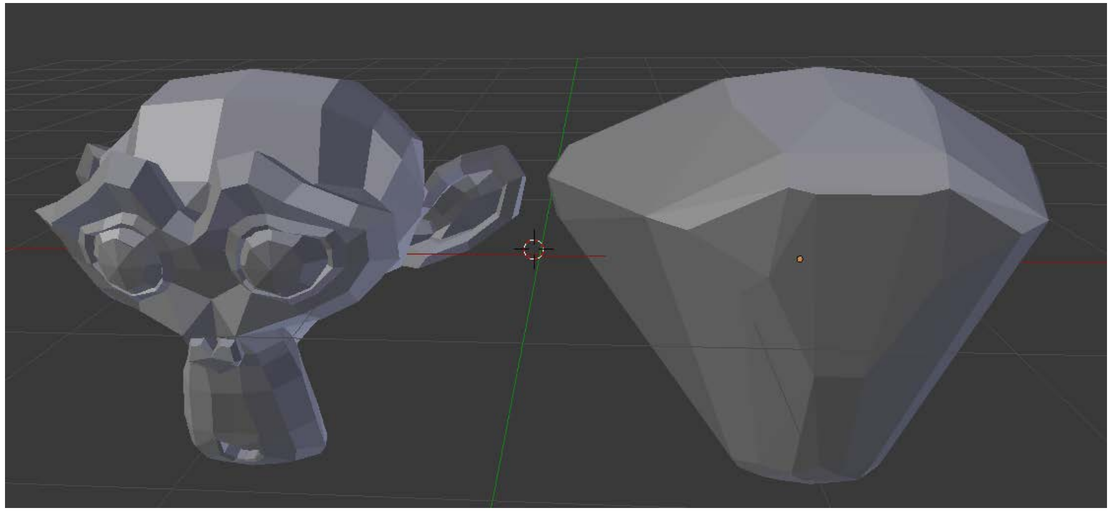
This structure is used throughout Unity to pass 3D positions and directions around. It also contains functions for doing common vector operations.

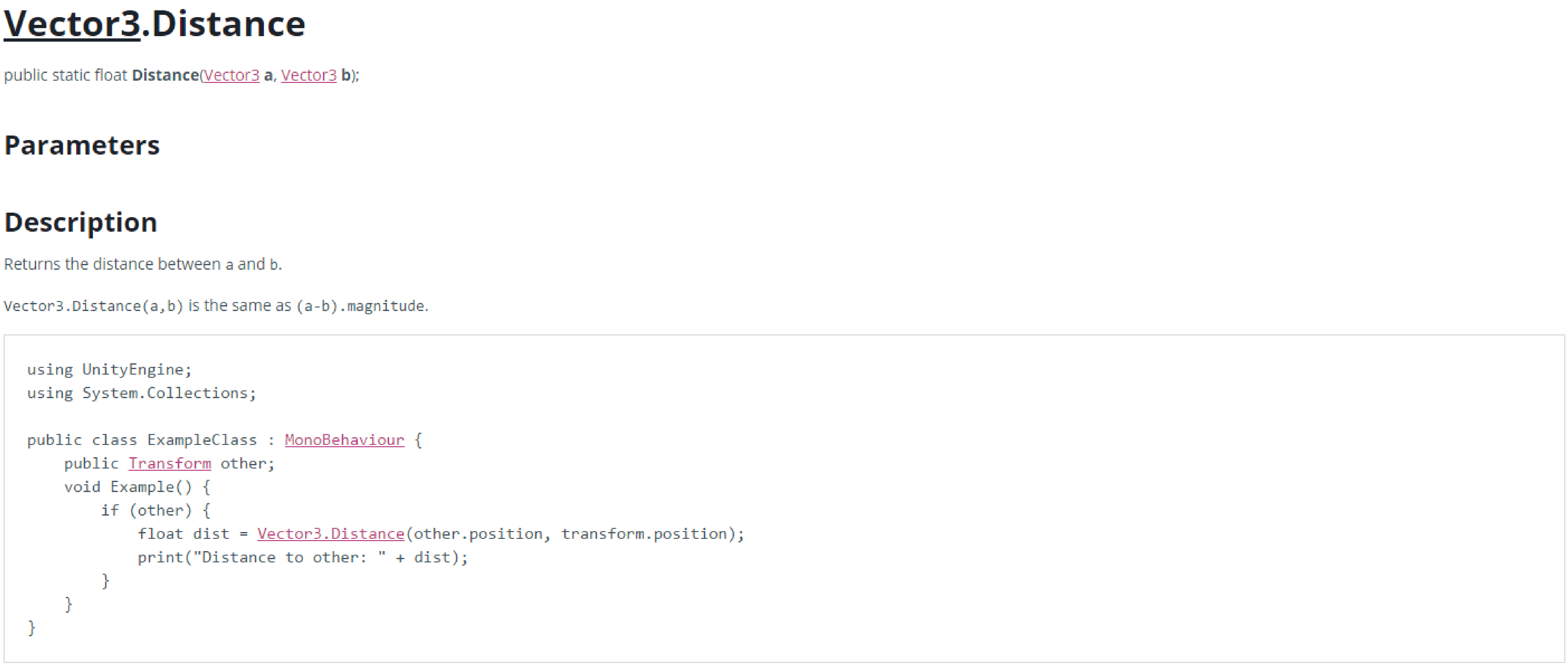


```
Vector3 position = new Vector3(0f, 0f, 1f);
```

# Convex Hull







# Vector3.Distance

public static float **Distance**(Vector3 **a**, Vector3 **b**);

## Parameters

## Description

Returns the distance between `a` and `b`.

`Vector3.Distance(a,b)` is the same as `(a-b).magnitude`.

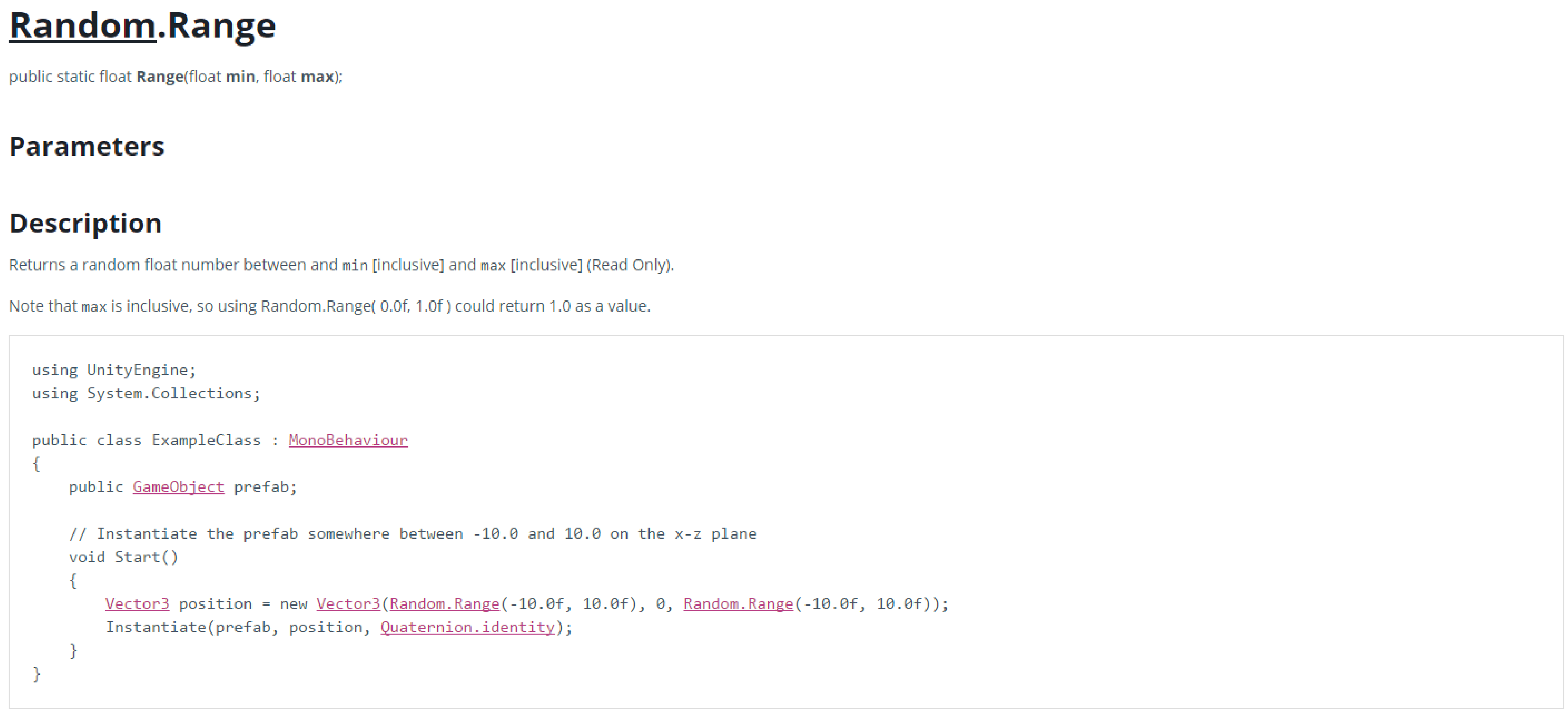
```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Transform other;
    void Example() {
        if (other) {
            float dist = Vector3.Distance(other.position, transform.position);
            print("Distance to other: " + dist);
        }
    }
}
```

http://docs.unity3d.com/ScriptReference/Vector3.Distance.html

```
public class DistanceDemo : MonoBehaviour {

    public Transform PlayerTransform;

    void Update () {

        // Find the distance
        float distance = Vector3.Distance(PlayerTransform.position, transform.position);

        // Check how this object is to the player
        if (distance <= 3f) {
            Debug.Log("Player is close!");
        } else {
            Debug.Log("Player is far!");
        }
    }
}
```

# Random

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



# Random.Range

public static float **Range**(float **min**, float **max**);

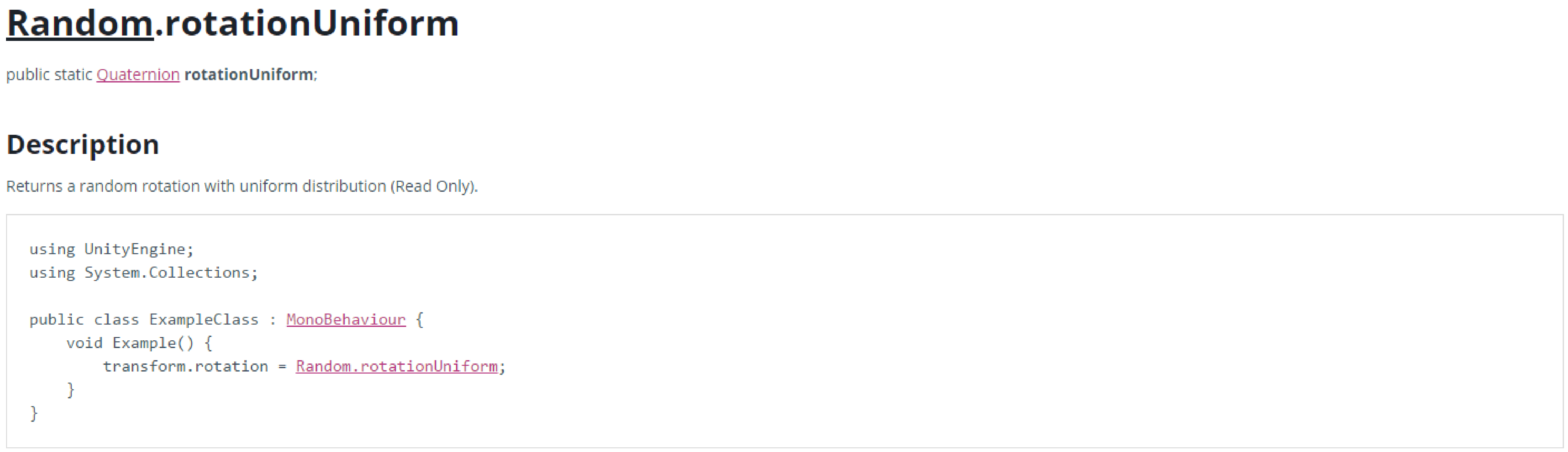
## Parameters

## Description

Returns a random float number between and `min` [inclusive] and `max` [inclusive] (Read Only).

Note that `max` is inclusive, so using Random.Range( 0.0f, 1.0f ) could return 1.0 as a value.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour
{
    public GameObject prefab;

    // Instantiate the prefab somewhere between -10.0 and 10.0 on the x-z plane
    void Start()
    {
        Vector3 position = new Vector3(Random.Range(-10.0f, 10.0f), 0, Random.Range(-10.0f, 10.0f));
        Instantiate(prefab, position, Quaternion.identity);
    }
}
```



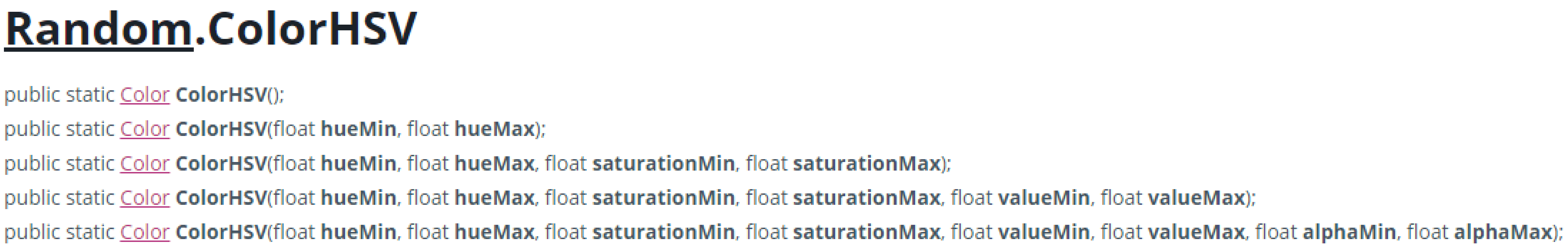
# Random.rotationUniform

public static Quaternion **rotationUniform**;

## Description

Returns a random rotation with uniform distribution (Read Only).

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        transform.rotation = Random.rotationUniform;
    }
}
```



# Random.ColorHSV

```
public static Color ColorHSV();
public static Color ColorHSV(float hueMin, float hueMax);
public static Color ColorHSV(float hueMin, float hueMax, float saturationMin, float saturationMax);
public static Color ColorHSV(float hueMin, float hueMax, float saturationMin, float saturationMax, float valueMin, float valueMax);
public static Color ColorHSV(float hueMin, float hueMax, float saturationMin, float saturationMax, float valueMin, float valueMax, float alphaMin, float alphaMax);
```

http://alloyui.com/examples/color-picker/hsv/

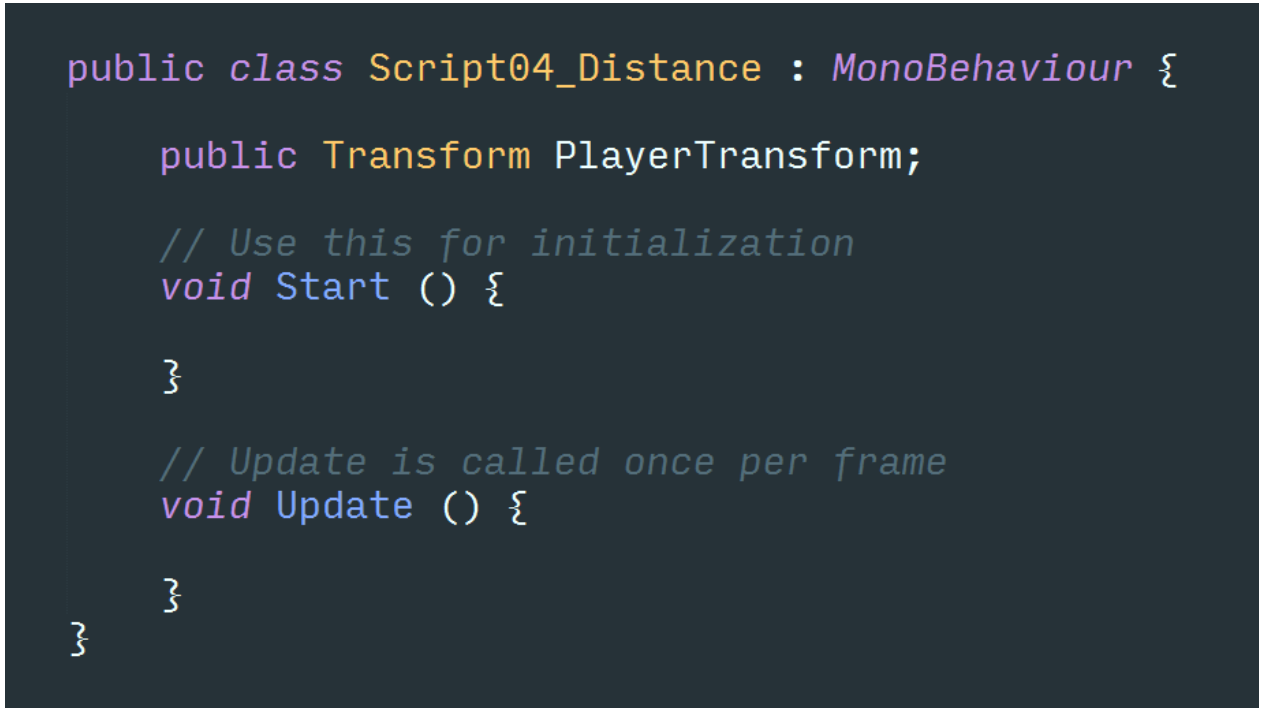
# MeshRenderer & Material

http://docs.unity3d.com/ScriptReference/MeshRenderer.html
http://docs.unity3d.com/ScriptReference/Material.html

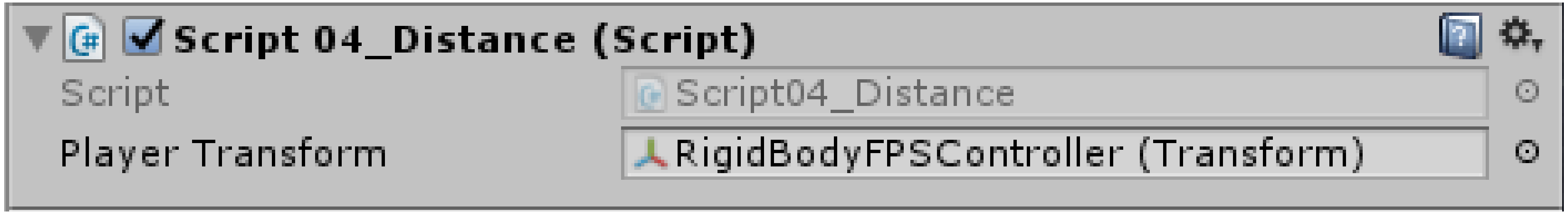
```
public class GettingMaterial : MonoBehaviour {

    private MeshRenderer renderer;
    private Material mat;

    void Start () {
        // Get the MeshRenderer on this game object
        renderer = GetComponent<MeshRenderer>();
        // Get the first material from the renderer
        mat = renderer.material;
        // Change the material's color to red
        mat.color = new Color(1f, 0f, 0f);
    }
}
```

# Accessing Other Game Objects

# Via Inspector



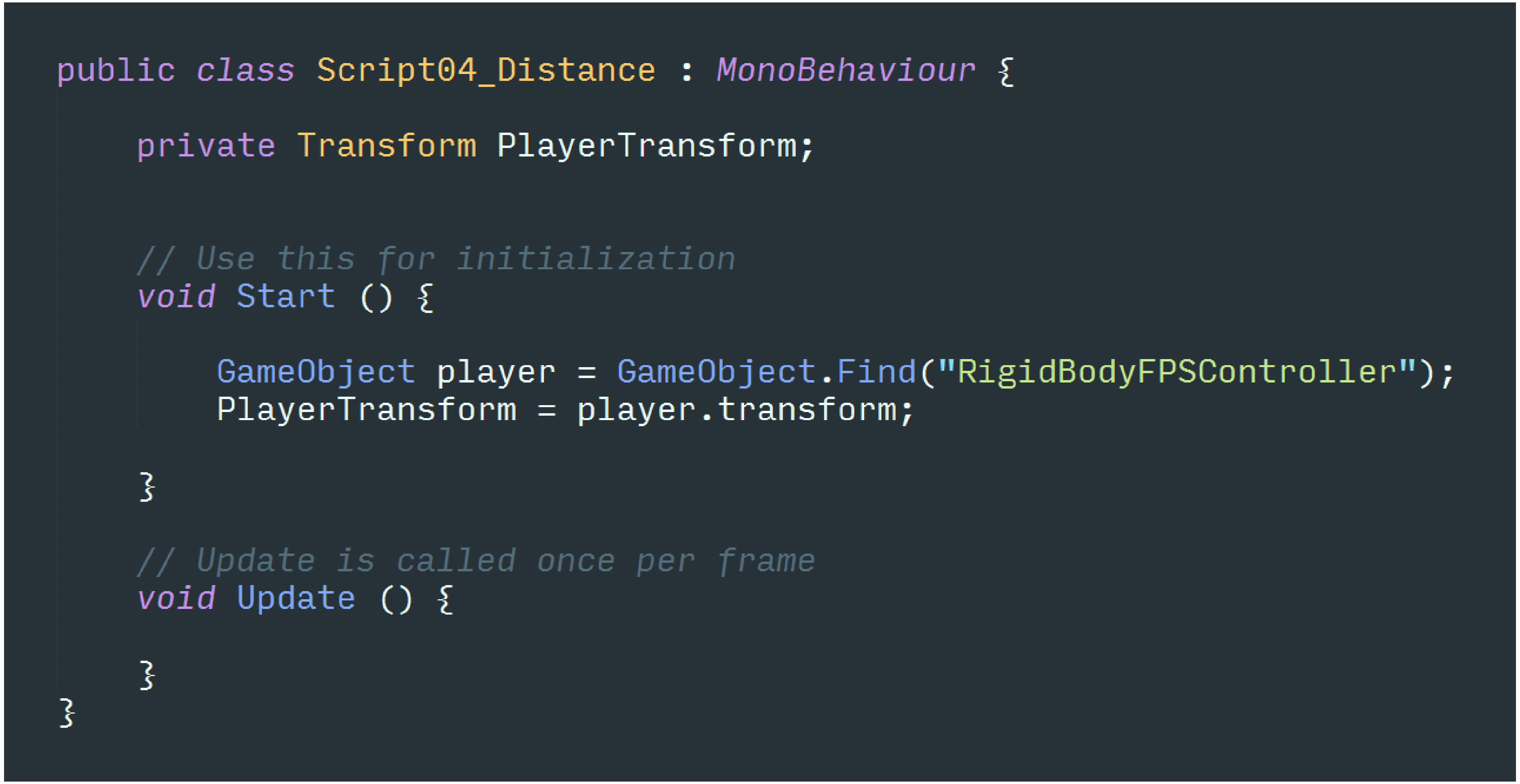
```
public class Script04_Distance : MonoBehaviour {

    public Transform PlayerTransform;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```



☑ Script 04_Distance (Script)

| | |
|---|---|
| Script | Script04_Distance |
| Player Transform | RigidBodyFPSController (Transform) |

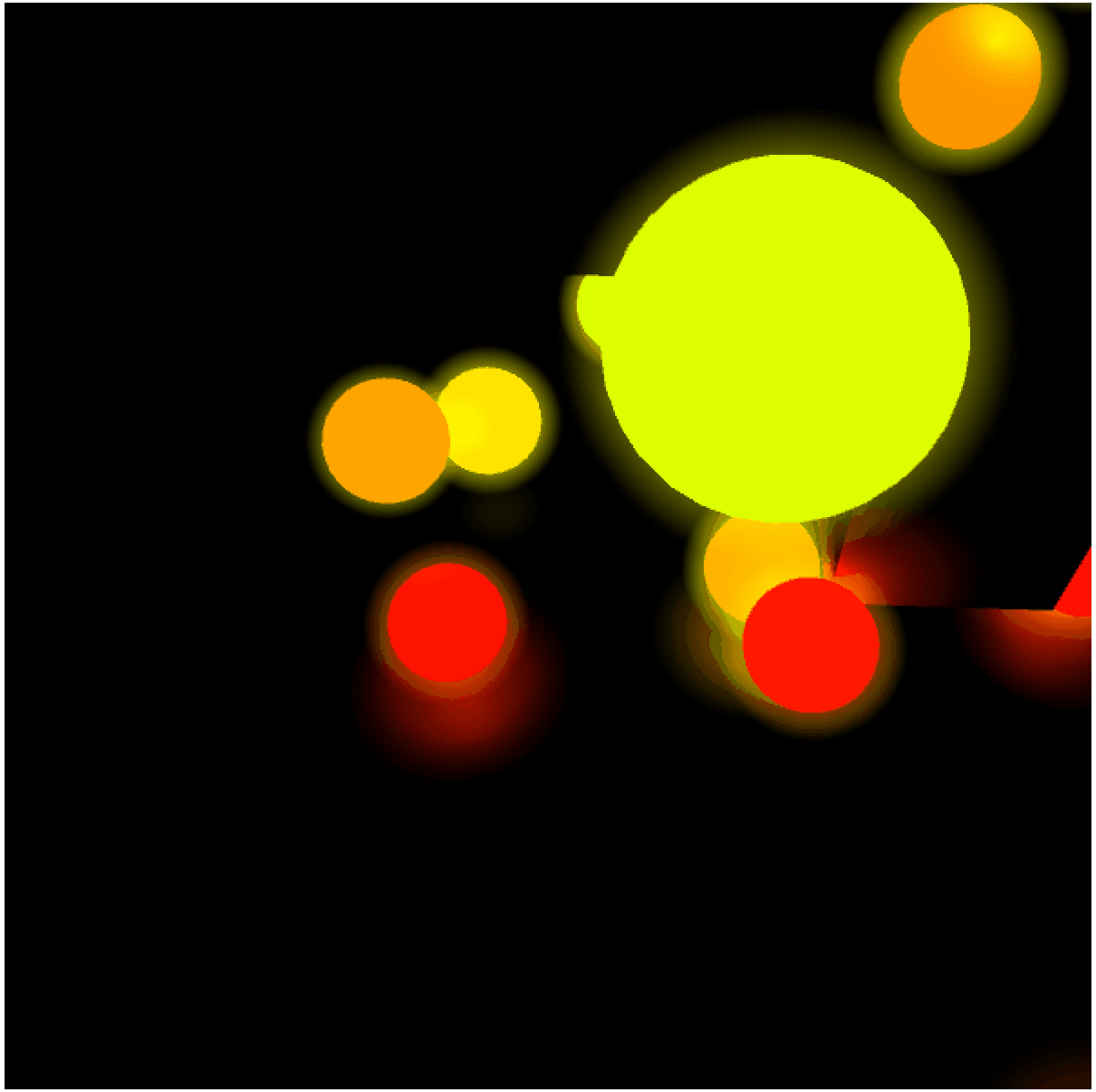
# Via Scripting



```
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 () {

    }
}
```

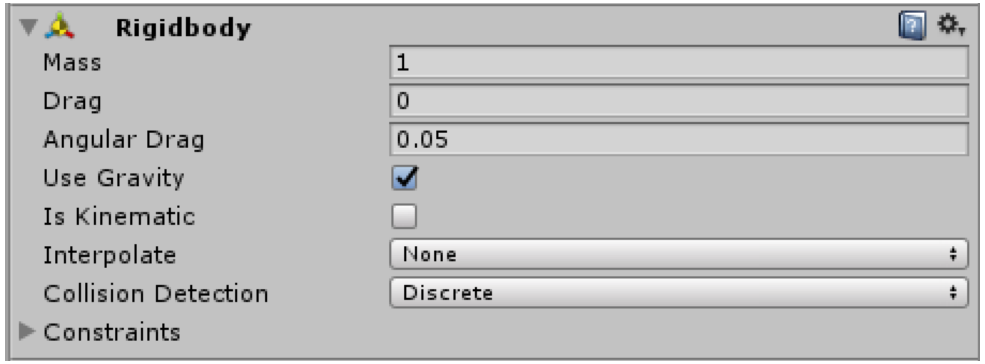
https://docs.unity3d.com/ScriptReference/GameObject.Find.html

# Popcorn Lights



# Bouncy Rigidbodies

1. Add rigidbody component



Rigidbody

Mass: 1

Drag: 0

Angular Drag: 0.05

Use Gravity: ☑

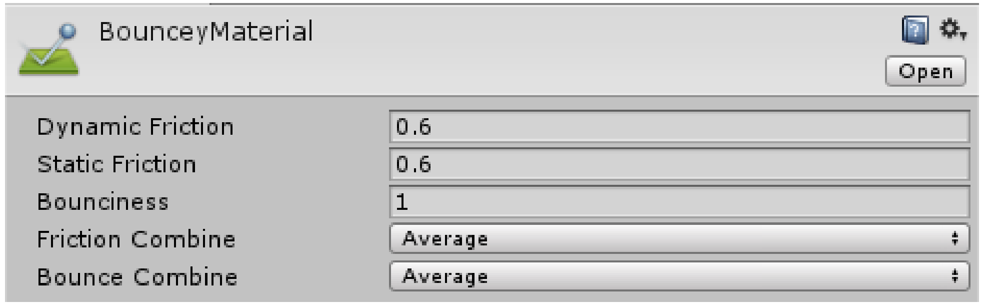
Is Kinematic: □

Interpolate: None

Collision Detection: Discrete

Constraints

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



BounceyMaterial

Dynamic Friction: 0.6

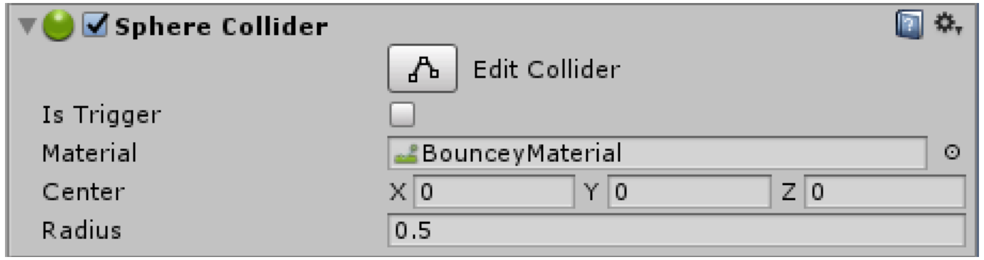
Static Friction: 0.6

Bounciness: 1

Friction Combine: Average

Bounce Combine: Average

3. Apply “physic material” to the collider



☑ Sphere Collider

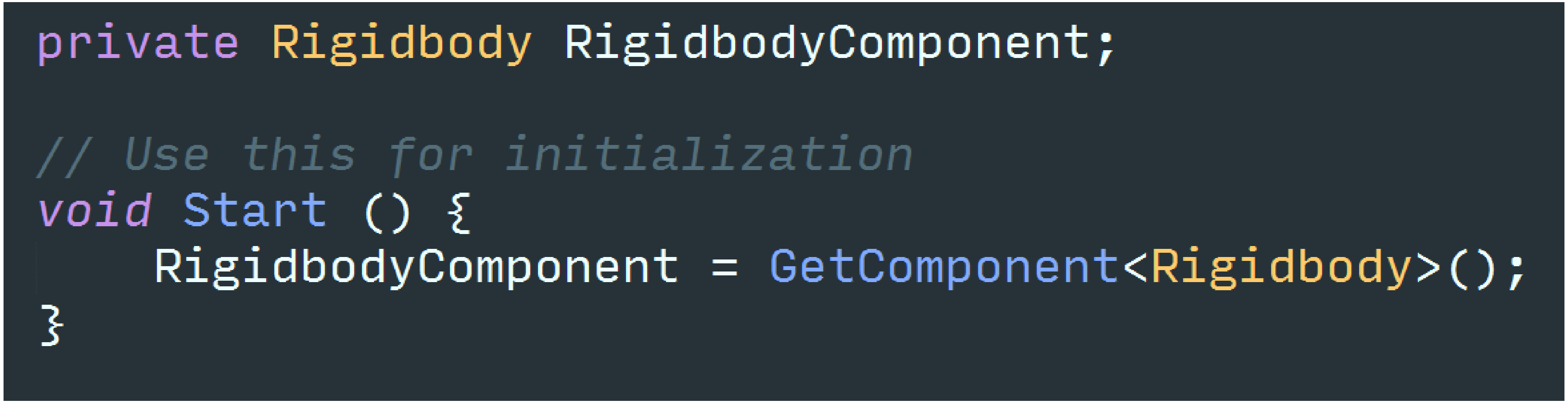
Is Trigger: □

Material: BounceyMaterial

Center: X 0 Y 0 Z 0

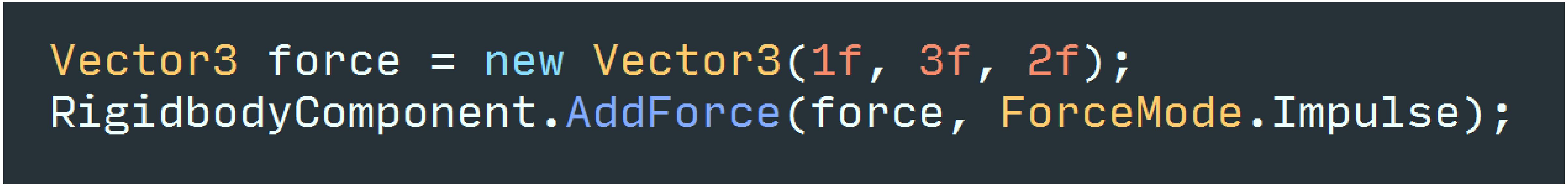
Radius: 0.5

# Getting Rigidbody

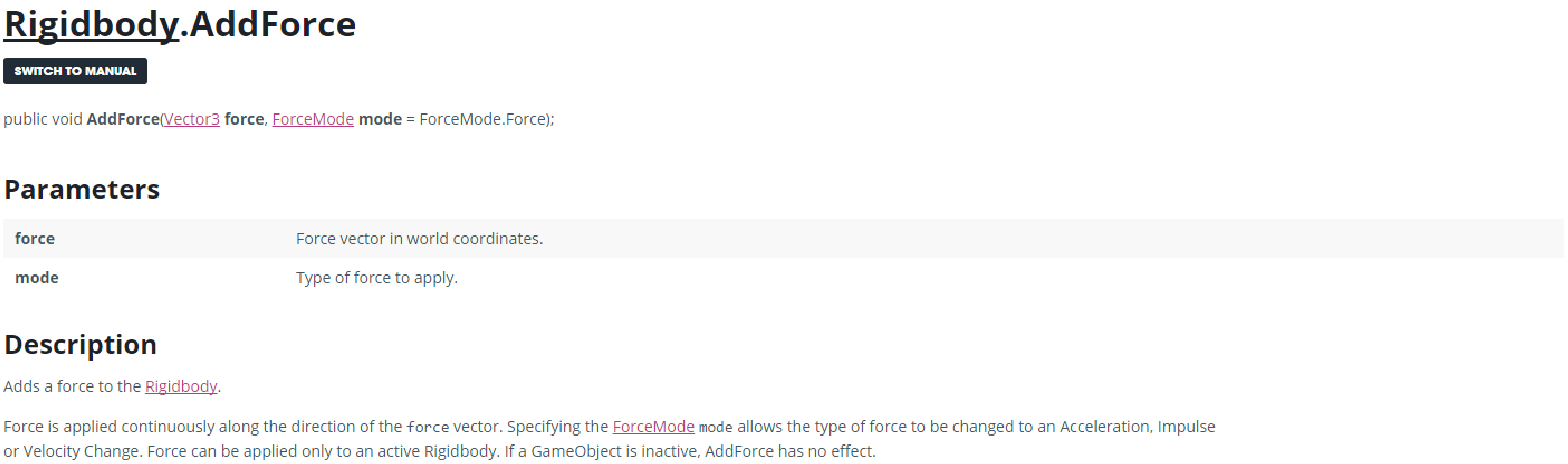


```
private Rigidbody RigidbodyComponent;

// 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);

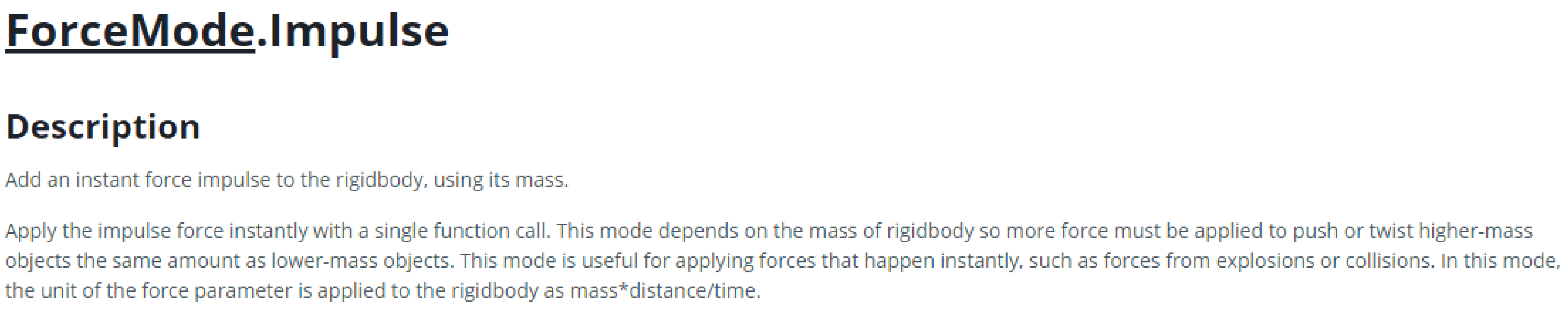
## Parameters

| | |
|---|---|
| **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

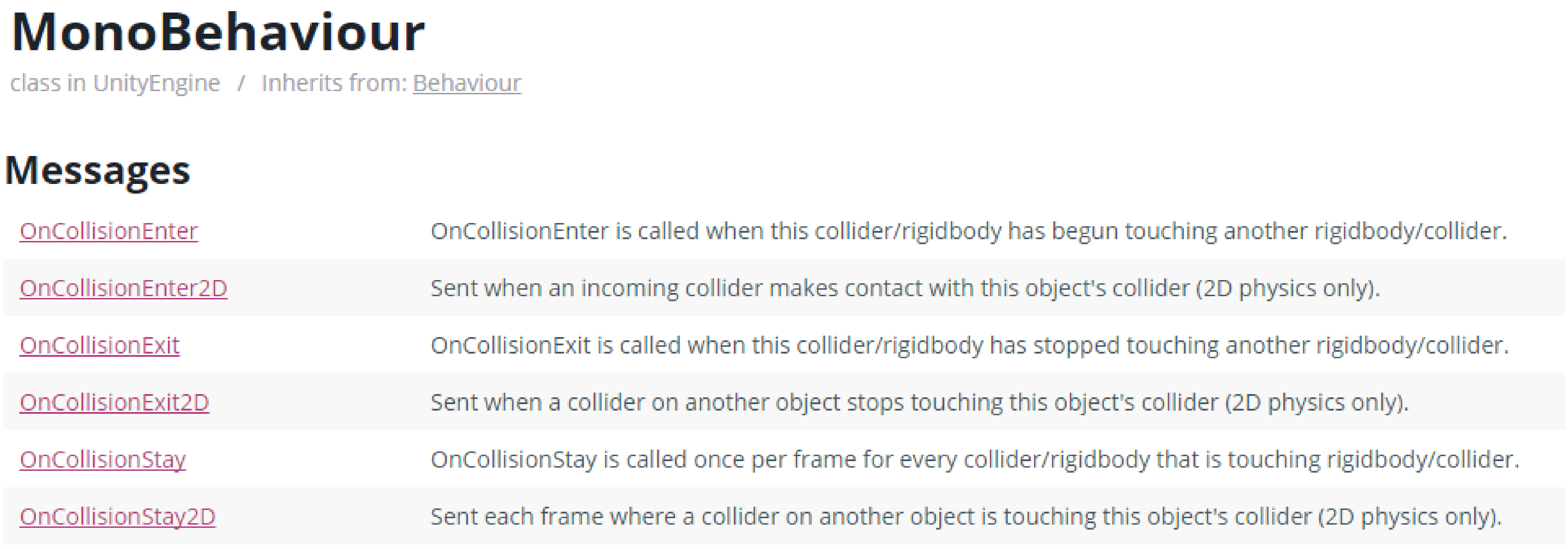
## Description

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 mass*distance/time.

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

# Detecting Collisions



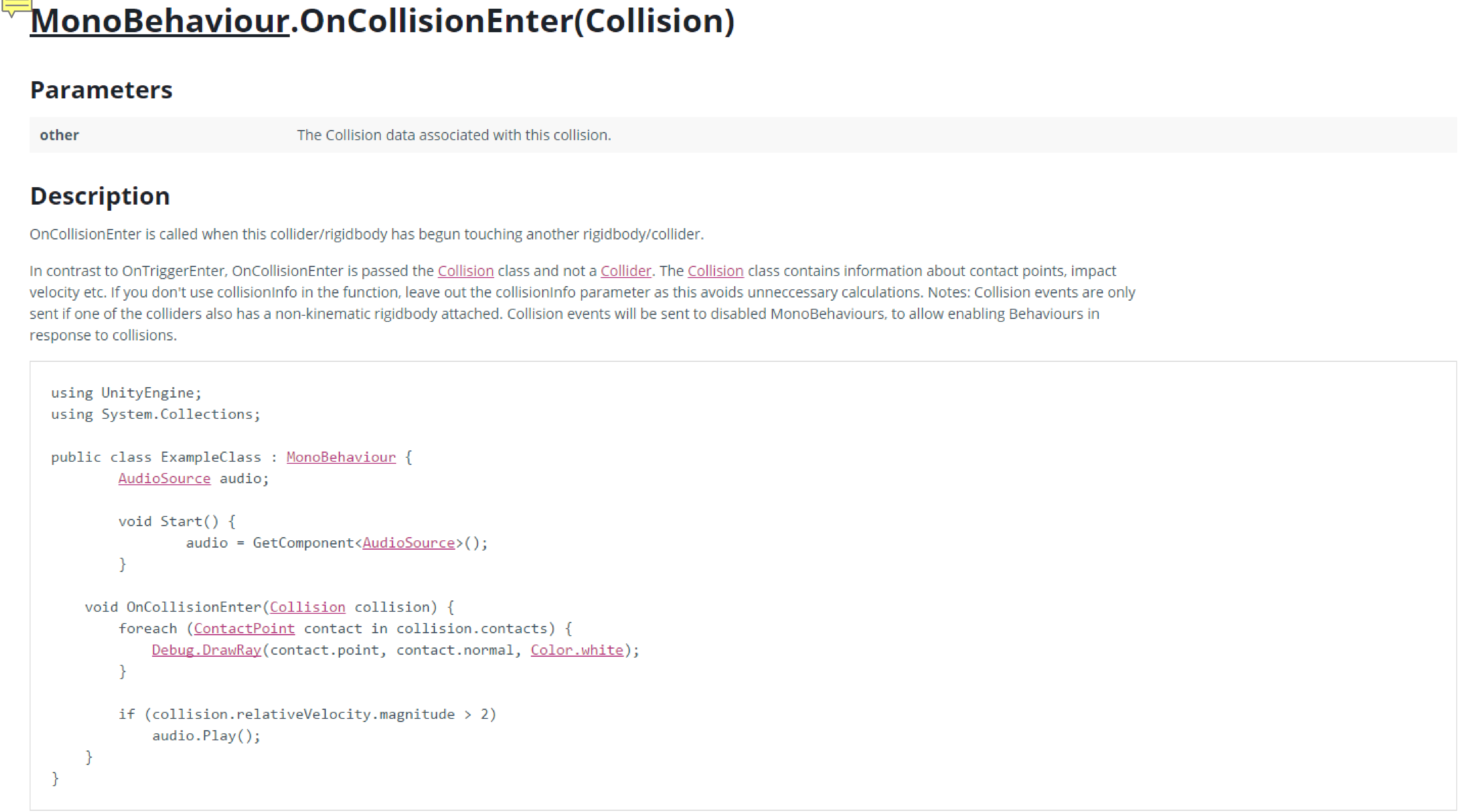
## MonoBehaviour

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). |

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.

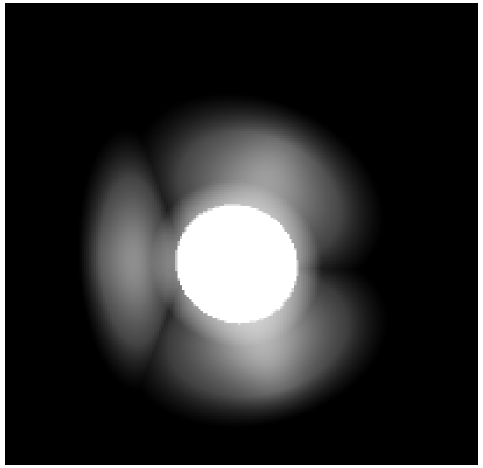
```
using UnityEngine;
using System.Collections;

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();
    }
}
```

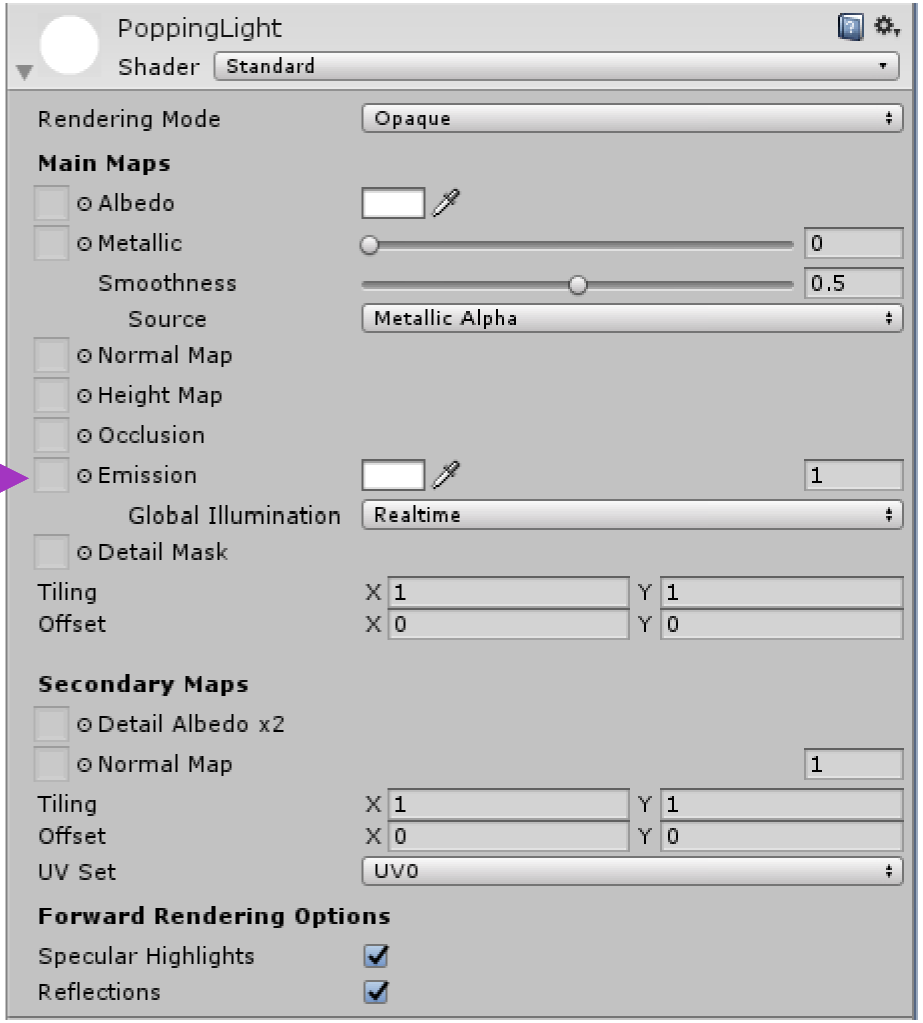
# Prefabs

- 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





PoppingLight
Shader: Standard

Rendering Mode: Opaque

**Main Maps**

- Albedo
- Metallic: 0
- Smoothness: 0.5
- Source: Metallic Alpha
- Normal Map
- Height Map
- Occlusion
- Emission: 1
- Global Illumination: Realtime
- Detail Mask

Tiling: X 1, Y 1
Offset: X 0, Y 0

**Secondary Maps**

- Detail Albedo x2
- Normal Map: 1

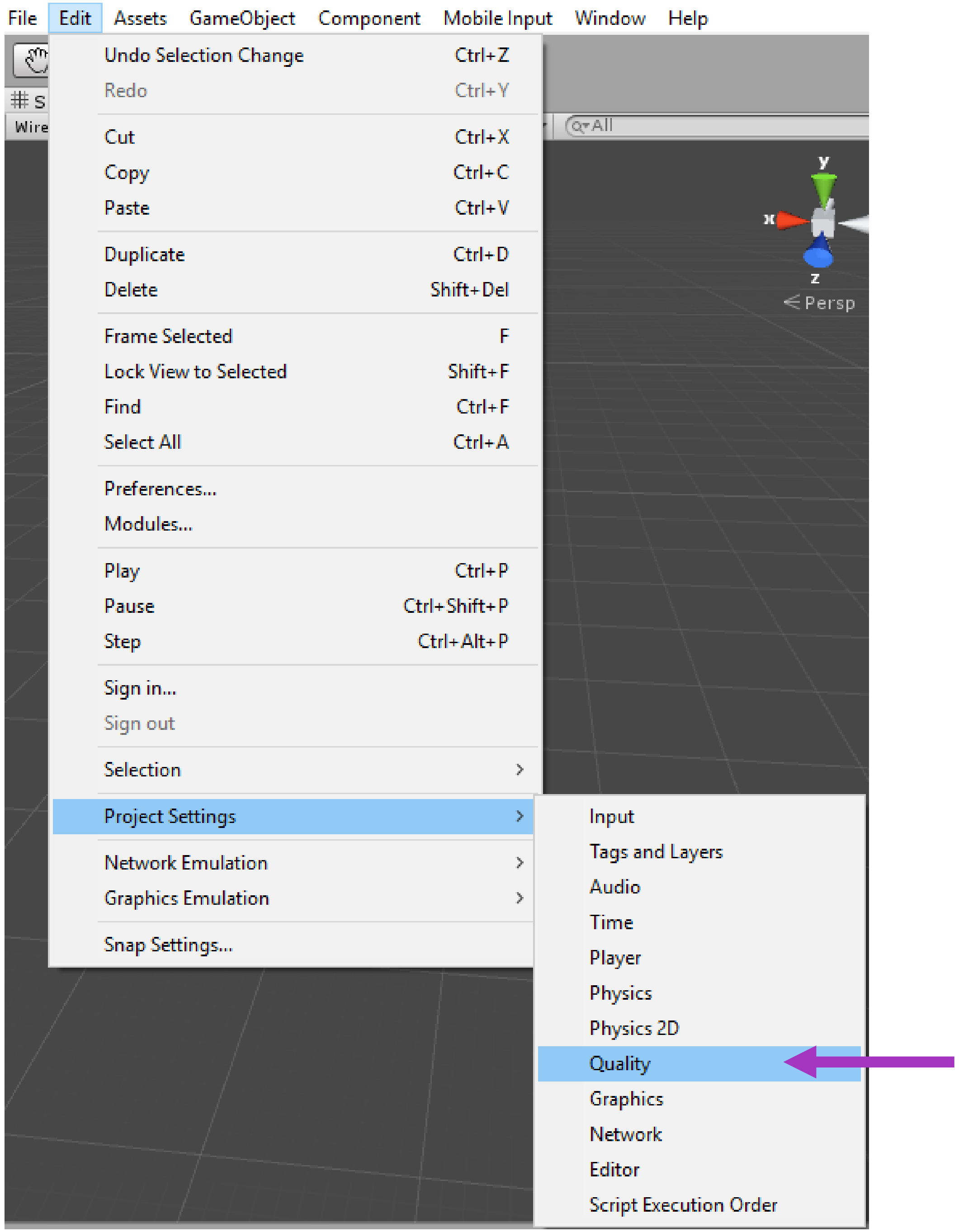
Tiling: X 1, Y 1
Offset: X 0, Y 0
UV Set: UV0

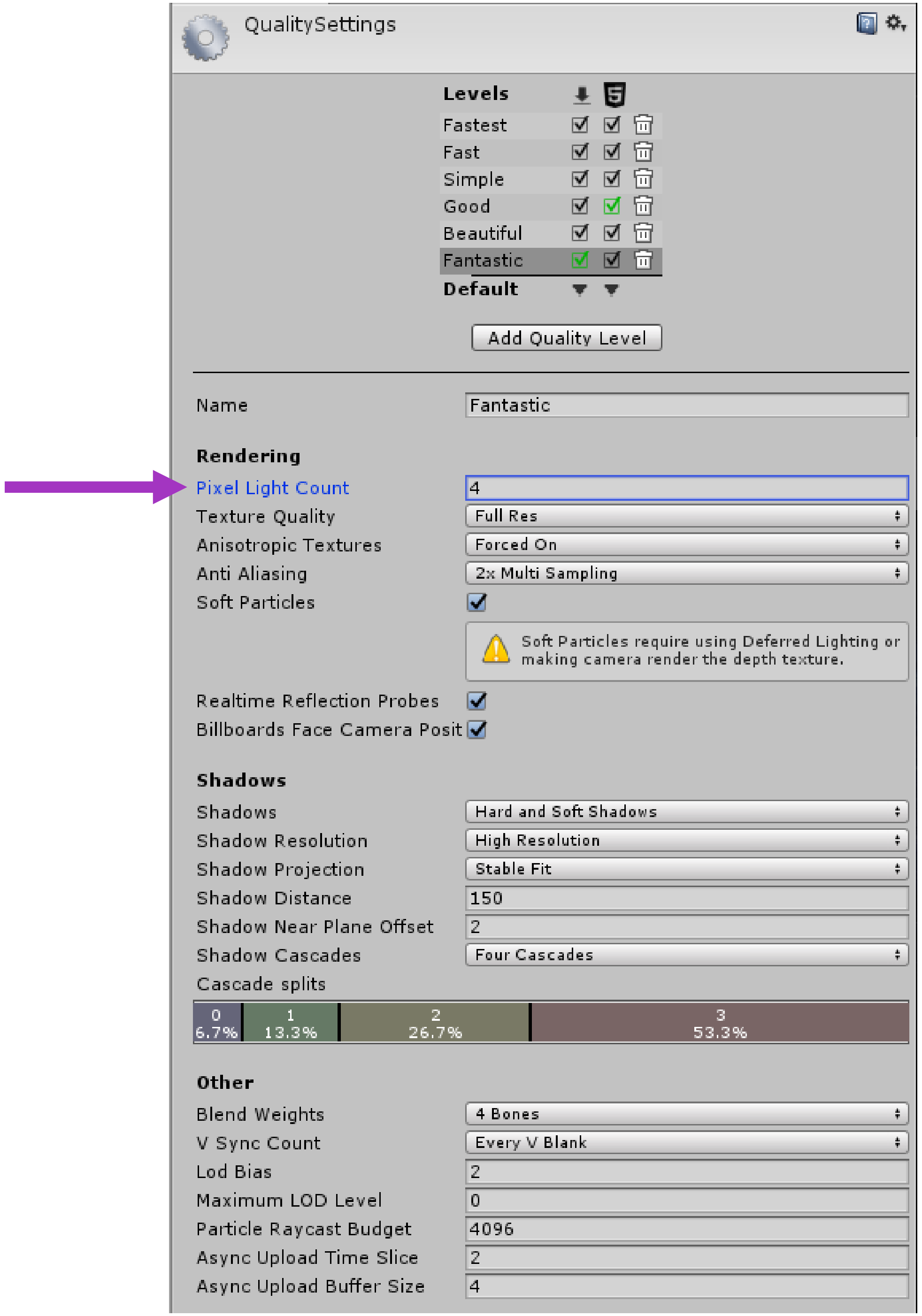
**Forward Rendering Options**

- ☑ Specular Highlights
- ☑ Reflections

# 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




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: ☑

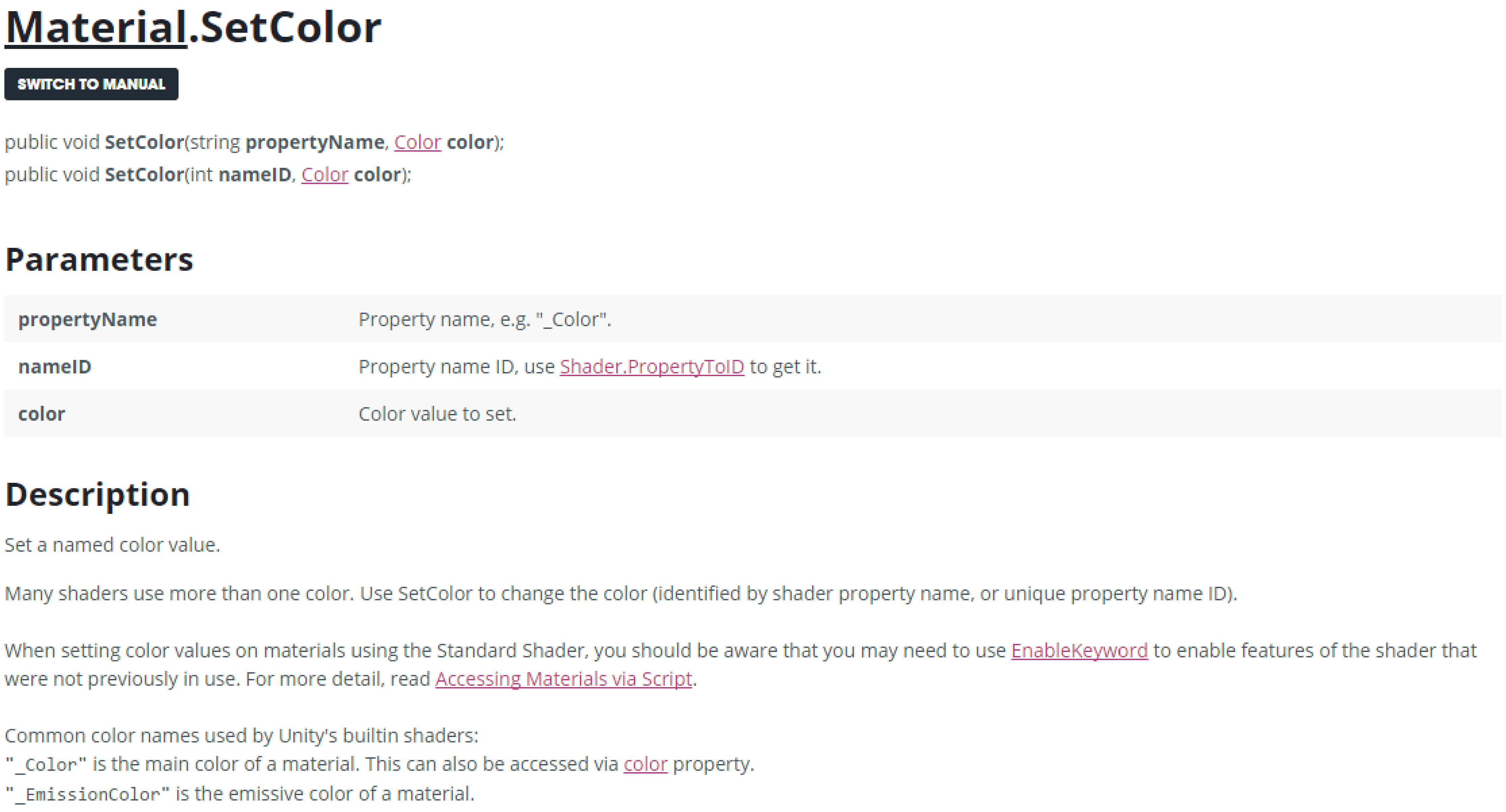
**Shadows**

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 | 1 | 2 | 3 |
|---|---|---|---|
| 6.7% | 13.3% | 26.7% | 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!

# Loops

- Unity tutorial