


C#  
(C Sharp)

 | DOCUMENTATION

Version: 5.4 ([switch to 5.5b](#))

## Scripting API

- UnityEngine
  - UnityEngine.Advertisements
  - UnityEngine.Analytics
  - UnityEngine.Apple
  - UnityEngine.Assertions
  - UnityEngine.Audio
  - UnityEngine.Diagnostics
  - UnityEngine.Events
  - UnityEngine.EventSystems
  - UnityEngine.Experimental
  - UnityEngine.iOS
  - UnityEngine.Networking
  - UnityEngine.Purchasing
  - UnityEngine.Rendering
  - UnityEngine.SceneManagement
  - UnityEngine.Scripting
  - UnityEngine.Serialization
  - UnityEngine.SocialPlatforms
  - UnityEngine.Sprites
  - UnityEngine.Tizen
  - UnityEngine.UI
  - UnityEngine.VR
  - UnityEngine.Windows
  - UnityEngine.WSA
- Classes
  - AccelerationEvent
  - AnchoredJoint2D
  - AndroidInput
  - AndroidJavaClass
  - AndroidJavaObject

## Camera

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

[SWITCH TO MANUAL](#)

### Description

A Camera is a device through which the player view the scene.

A screen space point is defined in pixels. The bottom-left corner is (0,0) and the top-right corner is (Screen.width, Screen.height).

A viewport space point is normalized and relative to the camera.

A world space point is defined in global coordinates.

See Also: [camera component](#).

### Static Variables

<a href="#">allCameras</a>	Returns the array of all cameras in the scene.
<a href="#">allCamerasCount</a>	The number of cameras in the scene.
<a href="#">current</a>	The currently active camera.
<a href="#">main</a>	The first camera created in the scene.
<a href="#">onPostRender</a>	Event that is triggered after the camera has finished rendering the scene.
<a href="#">onPreCull</a>	Event that is triggered before the camera culls the scene.
<a href="#">onPreRender</a>	Event that is triggered before the camera starts rendering the scene.

### Variables

<a href="#">actualRenderingPath</a>	The rendering path used by the camera. Possible values are: kRenderingPathDefault, kRenderingPathForward, kRenderingPathDeferred, kRenderingPathHybrid.
-------------------------------------	---

- Camera
- Color
- Collider
- Debug
- GameObject
- Light
- Material
- Mathf
- MeshCollider
- MeshRenderer
- MonoBehaviour
- PhysicMaterial
- Random
- ...

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

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

## Returns

**Object** A clone of the original object.

# Casting & Manipulating

```
// Spawning and casting
Vector3 spawnPoint = new Vector3(1f, 0f, 0f);
Quaternion spawnRotation = Quaternion.identity;
GameObject clone = (GameObject) Instantiate(Prefab, spawnPoint, spawnRotation, transform);

// Now we have a GameObject, rather than an Object. We can use any of the methods
// available on a GameObject:

// Apply a random scale
Vector3 randomScale = new Vector3(1f, Random.Range(1f, 3f), 1f);
clone.transform.localScale = randomScale;
```

# Arrays

```
int[] HighScores;
```



ARRAY TYPE



# Ways to Create an Array

```
// Empty integer array  
int[] HighScores;
```

```
// Empty integer array with four element  
int[] HighScores = new int[4];
```

```
// Integer array with specific values  
int[] HighScores = { 10, 12, 15, 20 };
```



# Resources

- Ray Wenderlich – [Video](#) on arrays
- Unity [tutorial](#) on arrays
- Blog [post](#): data structures in Unity and when to use them
- Unity [tutorial](#) on Lists and Dictionaries

# Explosions

# Making the Pokémon “Explodable”

We need a prefab that has colliders and physics:

1. Model settings: check “Generate Colliders”
2. Add model to the scene to create a game object
3. Mesh Collider: check “Convex”  
*(\*any mesh collider with a rigidbody needs to be set to convex)*
4. Add a Rigidbody component to the game object
5. Create a prefab from the game object

# Three Scripts

- RandomlySpawn.cs
  - Attached to an empty game object
  - Randomly place Pokémon in our scene
- FireExplosive.cs
  - Attached to the player
  - Throw an explosive Poké Ball from the player
- Explosive.cs
  - Attached to the Poké Ball
  - Explodes on contact

# Scripting Concepts

- [Destroy\(...\)](#)
- Finding collisions
  - [OnCollisionEnter\(Collision collision\)](#)
  - [Collision](#)
  - [Physics](#) class & [Physics.OverlapSphere\(...\)](#)
  - [Collider](#)
- Applying forces to rigidbodies
  - [Rigidbody.AddForce\(...\)](#)
  - [Rigidbody.AddRelativeForce\(...\)](#)
  - [Rigidbody.AddExplosionForce\(...\)](#)
- Drawing debugging information
  - [Gizmos](#) class & [OnDrawGizmos](#)
  - [Gizmos.color](#)
  - [Gizmos.DrawSphere\(...\)](#)

# Gizmos

class in UnityEngine

## Description

Gizmos are used to give visual debugging or setup aids in the scene view.

All gizmo drawing has to be done in either [OnDrawGizmos](#) or [OnDrawGizmosSelected](#) functions of the script.

[OnDrawGizmos](#) is called every frame. All gizmos rendered within [OnDrawGizmos](#) are pickable. [OnDrawGizmosSelected](#) is called only if the object the script is attached to is selected.

## Static Variables

[color](#) Sets the color for the gizmos that will be drawn next.

[matrix](#) Set the gizmo matrix used to draw all gizmos.

## Static Functions

[DrawCube](#) Draw a solid box with center and size.

[DrawFrustum](#) Draw a camera frustum using the currently set Gizmos.matrix for it's location and rotation.

[DrawGUITexture](#) Draw a texture in the scene.

[DrawIcon](#) Draw an icon at a position in the scene view.

[DrawLine](#) Draws a line starting at from towards to.

[DrawMesh](#) Draws a mesh.

[DrawRay](#) Draws a ray starting at from to from + direction.

[DrawSphere](#) Draws a solid sphere with center and radius.

[DrawWireCube](#) Draw a wireframe box with center and size.

[DrawWireMesh](#) Draws a wireframe mesh.

[DrawWireSphere](#) Draws a wireframe sphere with center and radius.

# Physics.OverlapSphere

public static Collider[] **OverlapSphere**([Vector3](#) position, float radius, int layerMask = AllLayers, [QueryTriggerInteraction](#) queryTriggerInteraction = QueryTriggerInteraction.UseGlobal);

## Parameters

position	Center of the sphere.
radius	Radius of the sphere.
layerMask	A <a href="#">Layer mask</a> that is used to selectively ignore colliders when casting a ray.
queryTriggerInteraction	Specifies whether this query should hit Triggers.

## Description

Returns an array with all colliders touching or inside the sphere.

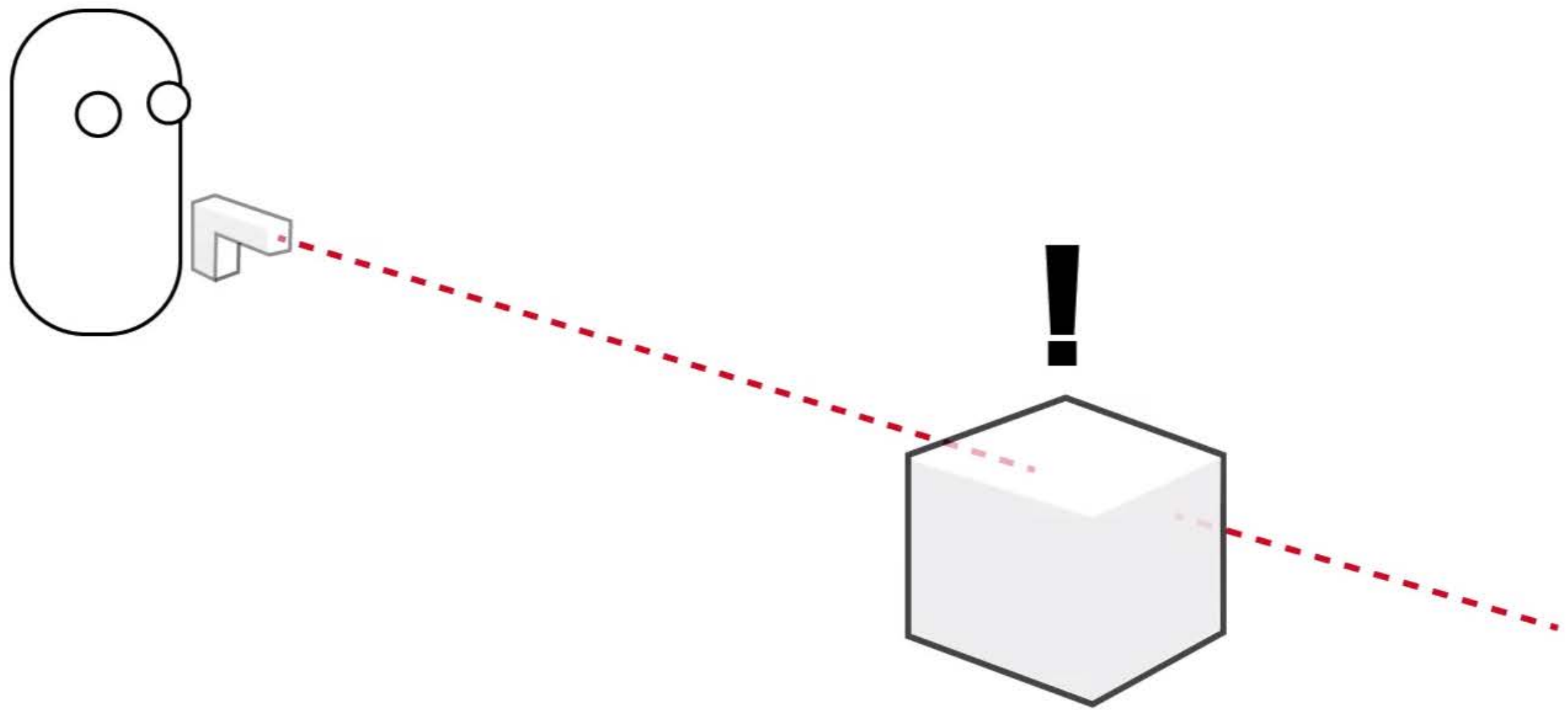
NOTE: Currently this only checks against the bounding volumes of the colliders not against the actual colliders.

```
using UnityEngine;
using System.Collections;

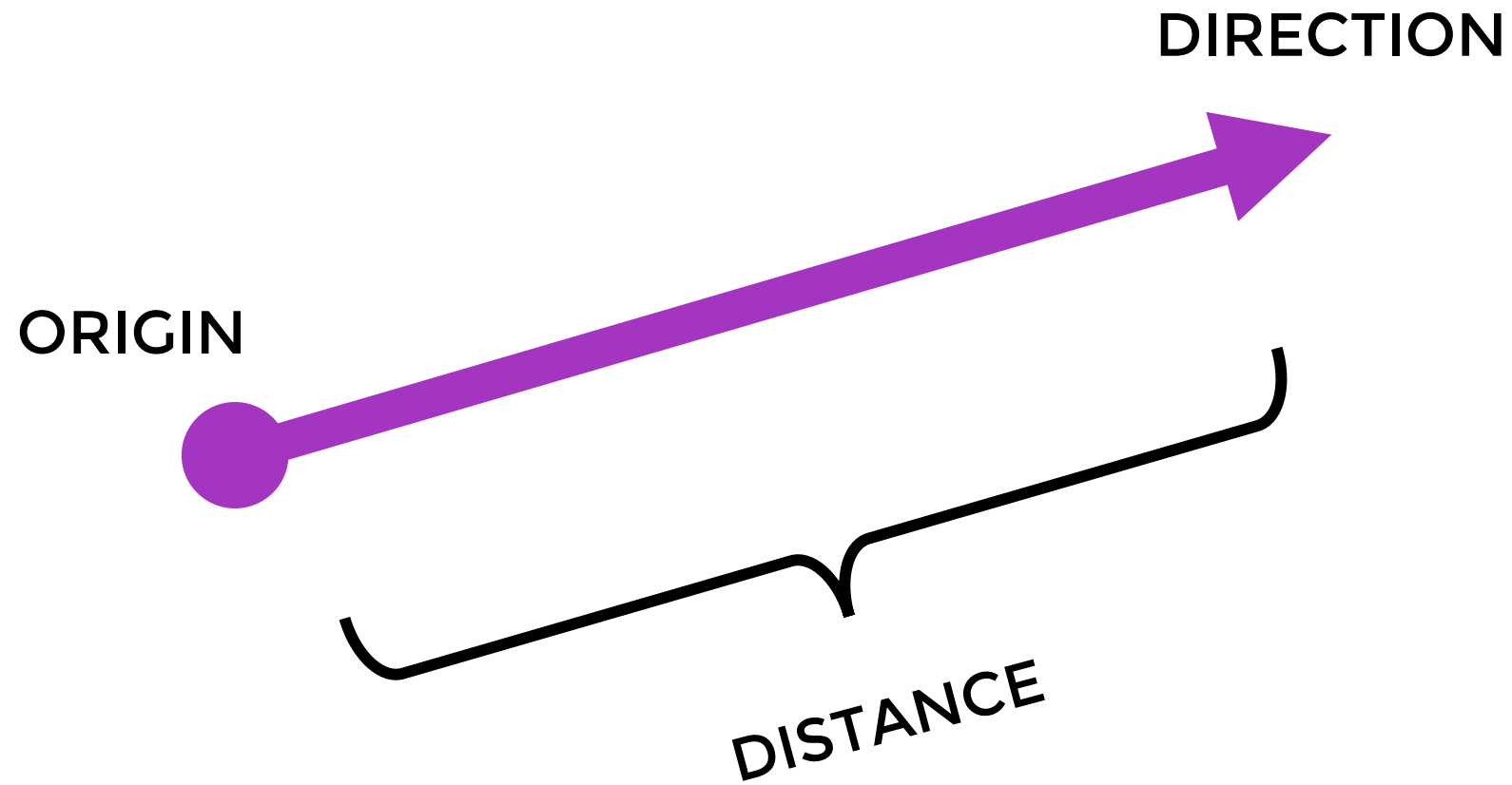
public class ExampleClass : MonoBehaviour {
    void ExplosionDamage(Vector3 center, float radius) {
        Collider[] hitColliders = Physics.OverlapSphere(center, radius);
        int i = 0;
        while (i < hitColliders.Length) {
            hitColliders[i].SendMessage("AddDamage");
            i++;
        }
    }
}
```

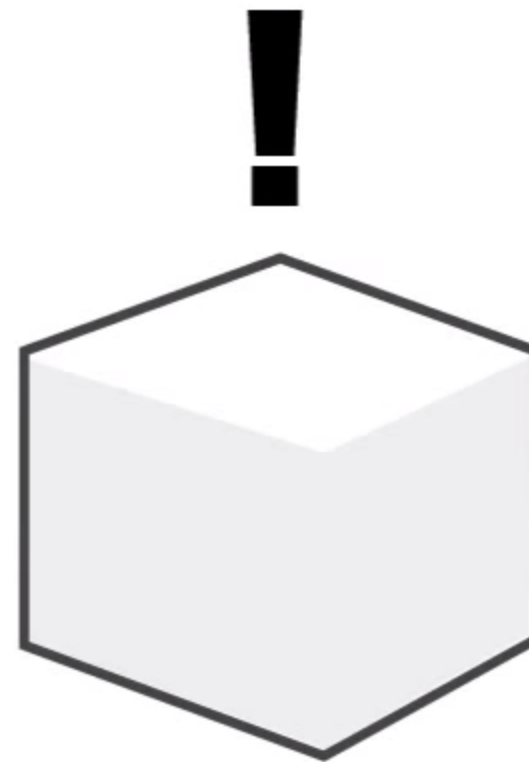
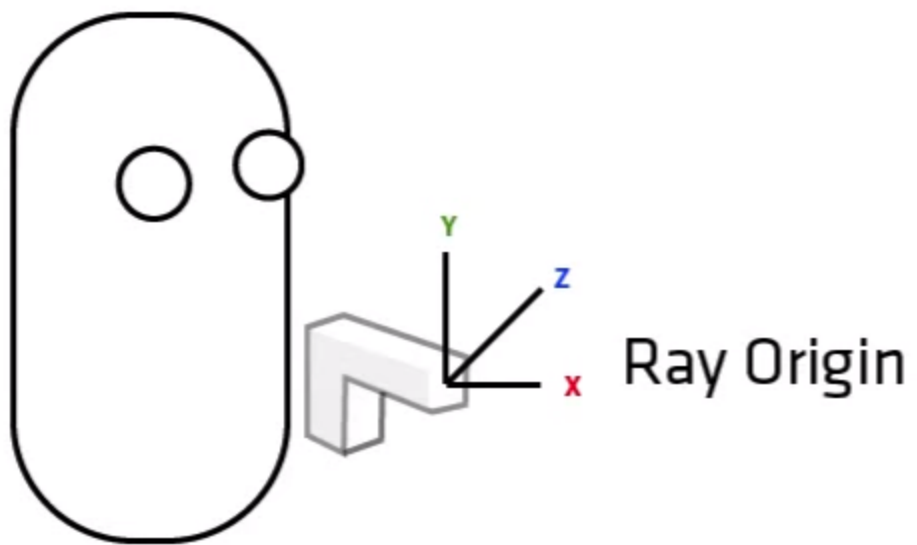
# Raycasting

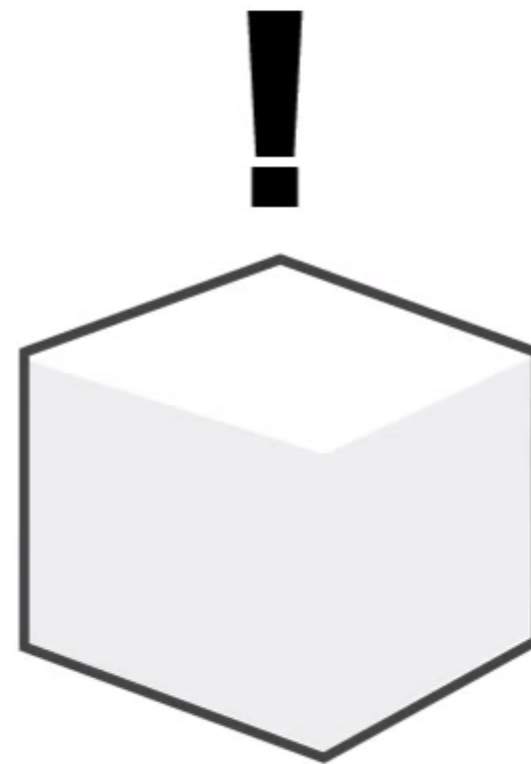
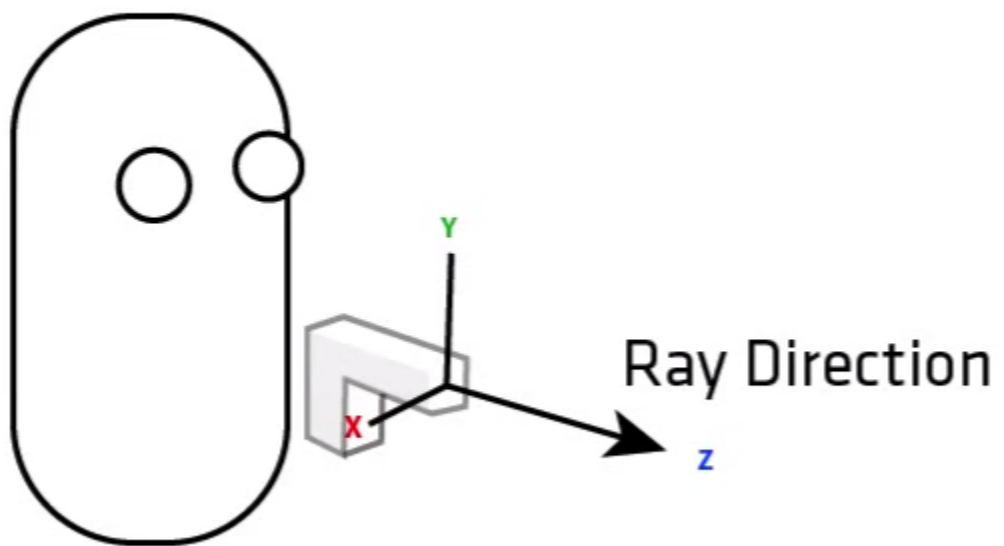


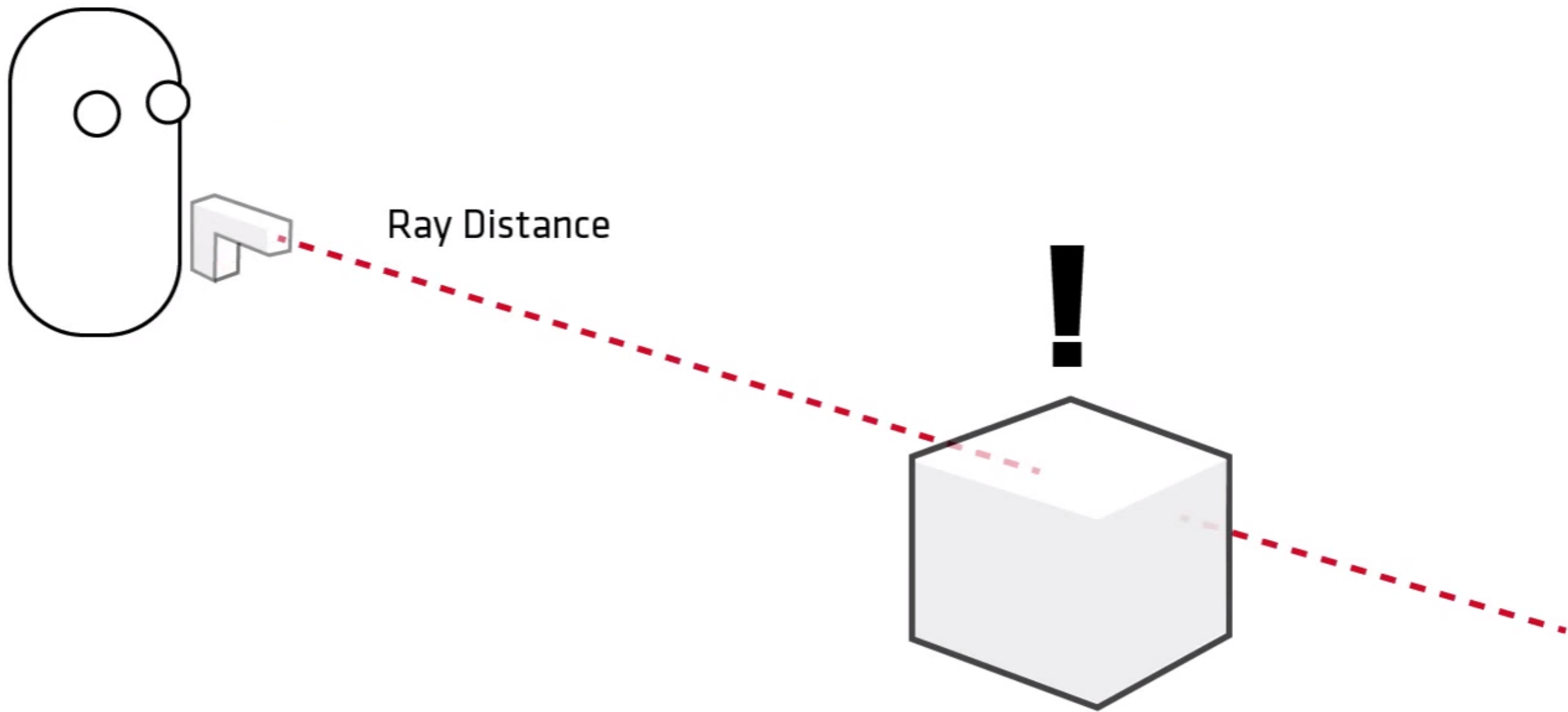


# Ray









# New Scripting Concepts

- Finding a 3D direction from the mouse position
  - [Input.mousePosition](#)
  - [Camera](#) class & [Camera.ScreenPointToRay\(...\)](#)
- Raycasting to find object(s) along a path
  - [Ray](#) struct
  - [Physics.Raycast\(...\)](#)
  - [RaycastHit](#) struct