UNITY 2019 - Basics

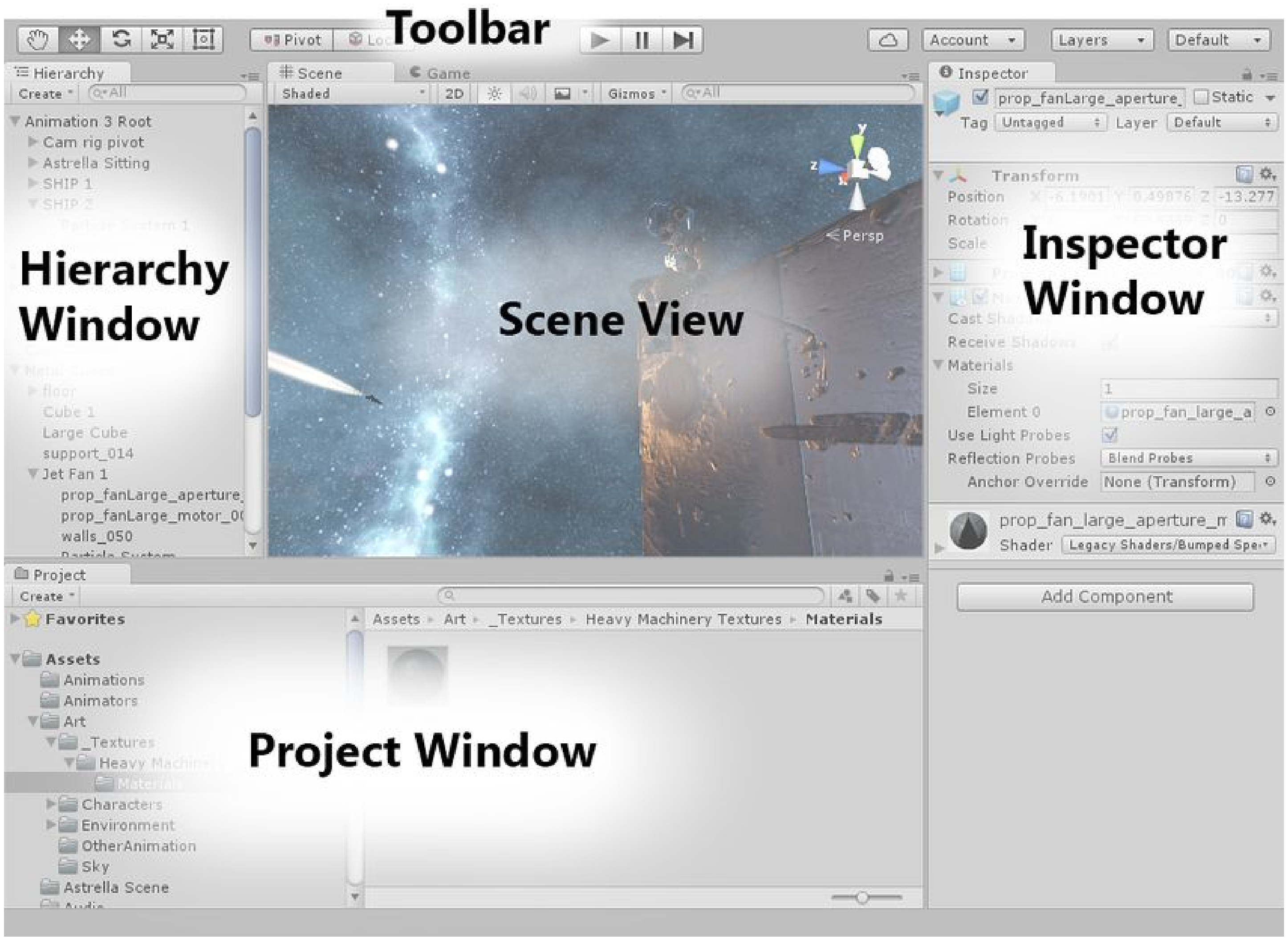
This script is a mix between my own texts, the official documentation and other tutorials. The complete documentation can be found here: https://docs.unity3d.com/Manual/UnityOverview.html

Learning the interface

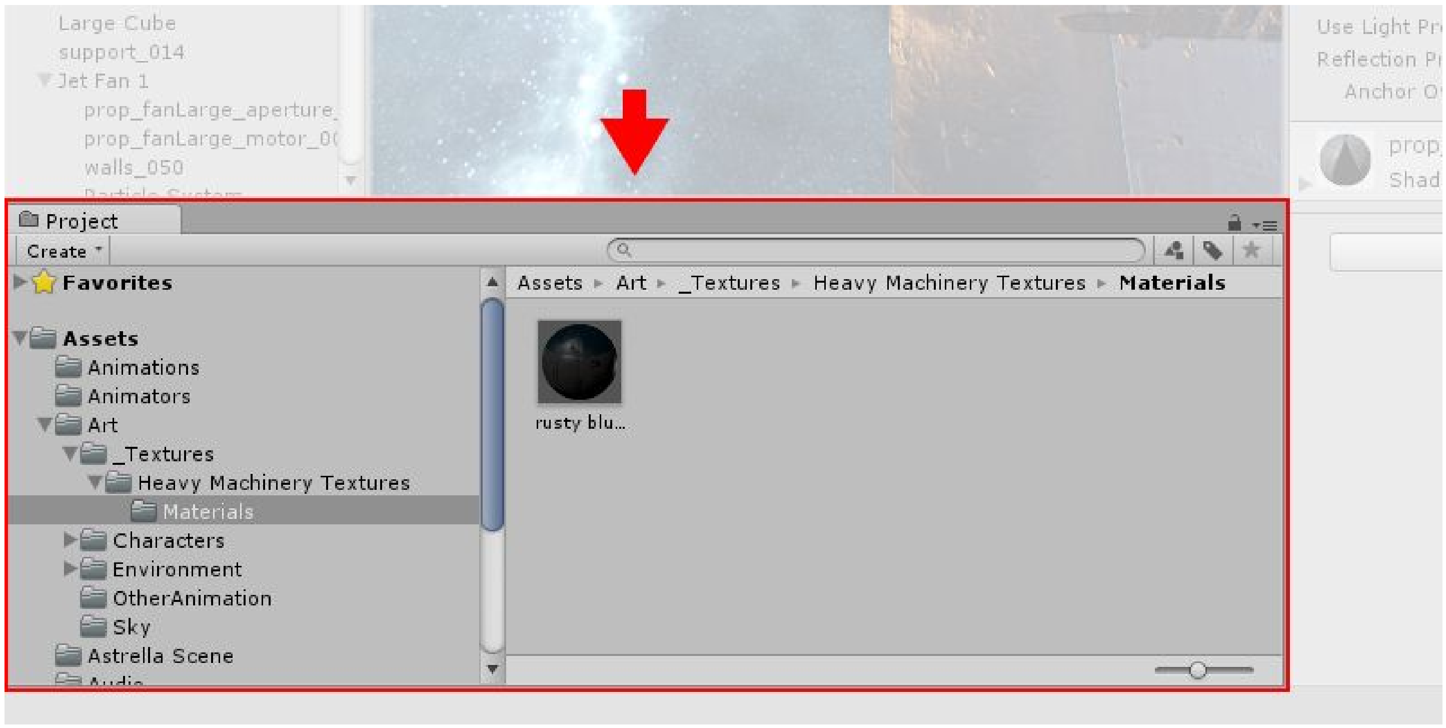
Take your time to look over the editor interface and familiarize yourself with it. The main editor window is made up of tabbed windows which can be rearranged, grouped, detached and docked.

This means the look of the editor can be different from one project to the next, and one developer to the next, depending on personal preference and what type of work you are doing.

The default arrangement of windows gives you practical access to the the most common windows. If you are not yet familiar with the different windows in Unity, you can identify them by the name in the tab. The most common and useful windows are shown in their default positions, below:

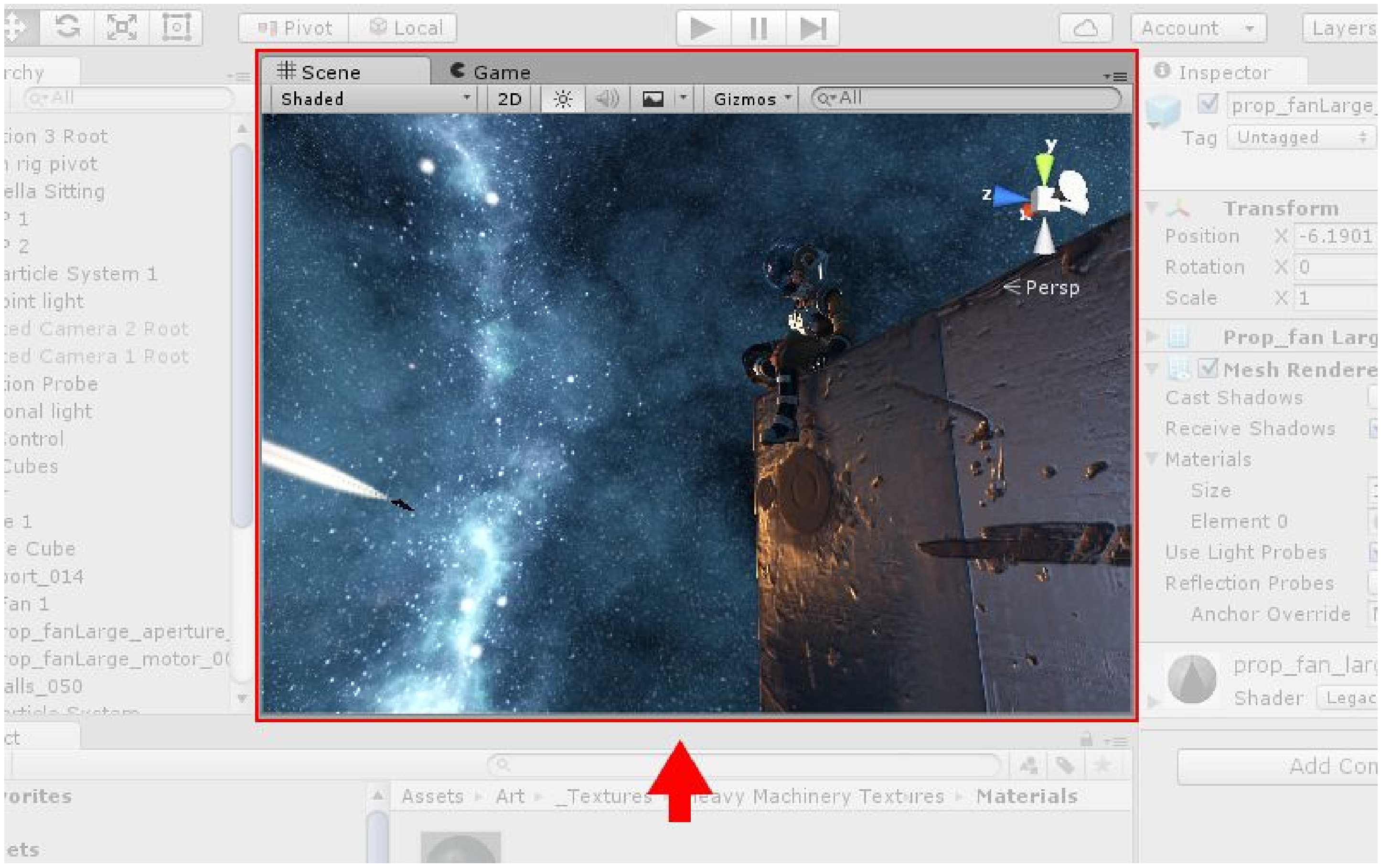


The Project Window



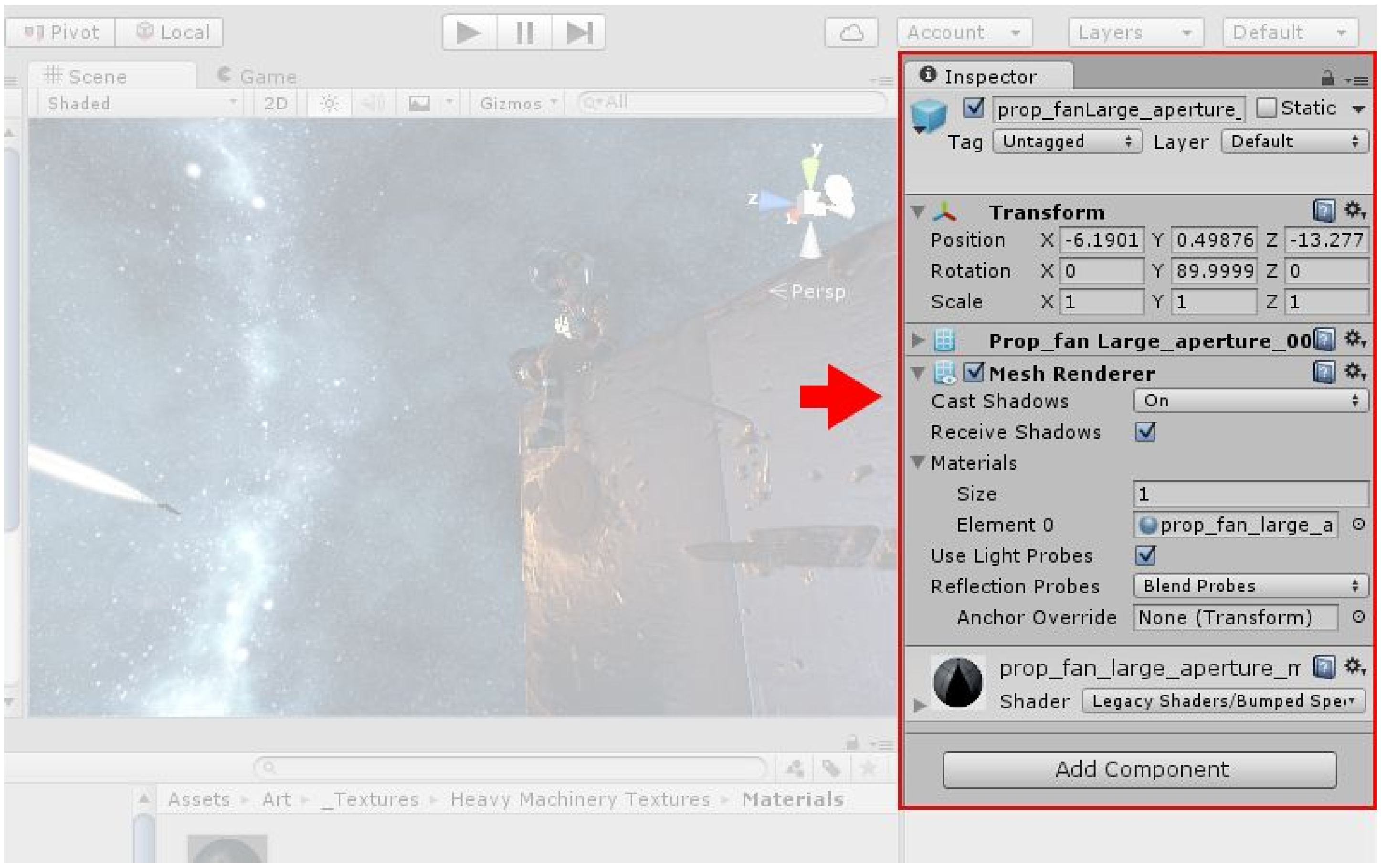
The Project Window displays your library of assets that are available to use in your project. When you import assets into your project, they appear here. Find out more about the Project Window.

The Scene View



The Scene View allows you to visually navigate and edit your scene. The scene view can show a 3D or 2D perspective, depending on the type of project you are working on. Find out more about the Scene View and the Game View.

The Inspector Window


Pivot
Local
Account
Layers
Default
Scene
Game
Shaded
2D
Gizmos
Persp
Assets ▸ Art ▸ _Textures ▸ Heavy Machinery Textures ▸ Materials

Inspector

prop_fanLarge_aperture_ Static

Tag Untagged Layer Default

Transform

Position	X	-6.1901	Y	0.49876	Z	-13.277
Rotation	X	0	Y	89.9999	Z	0
Scale	X	1	Y	1	Z	1

Prop_fan Large_aperture_00

Mesh Renderer

Cast Shadows	On
Receive Shadows	✓
Materials	
Size	1
Element 0	prop_fan_large_a
Use Light Probes	✓
Reflection Probes	Blend Probes
Anchor Override	None (Transform)

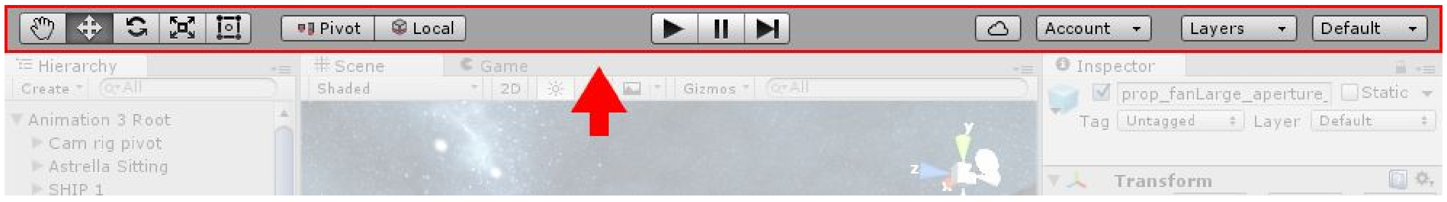
prop_fan_large_aperture_m

Shader Legacy Shaders/Bumped Spe

Add Component

The Inspector Window allows you to view and edit all the properties of the currently selected object. Because different types of objects have different sets of properties, the layout and contents of the inspector window will vary. Find out more about the Inspector Window.

The Toolbar

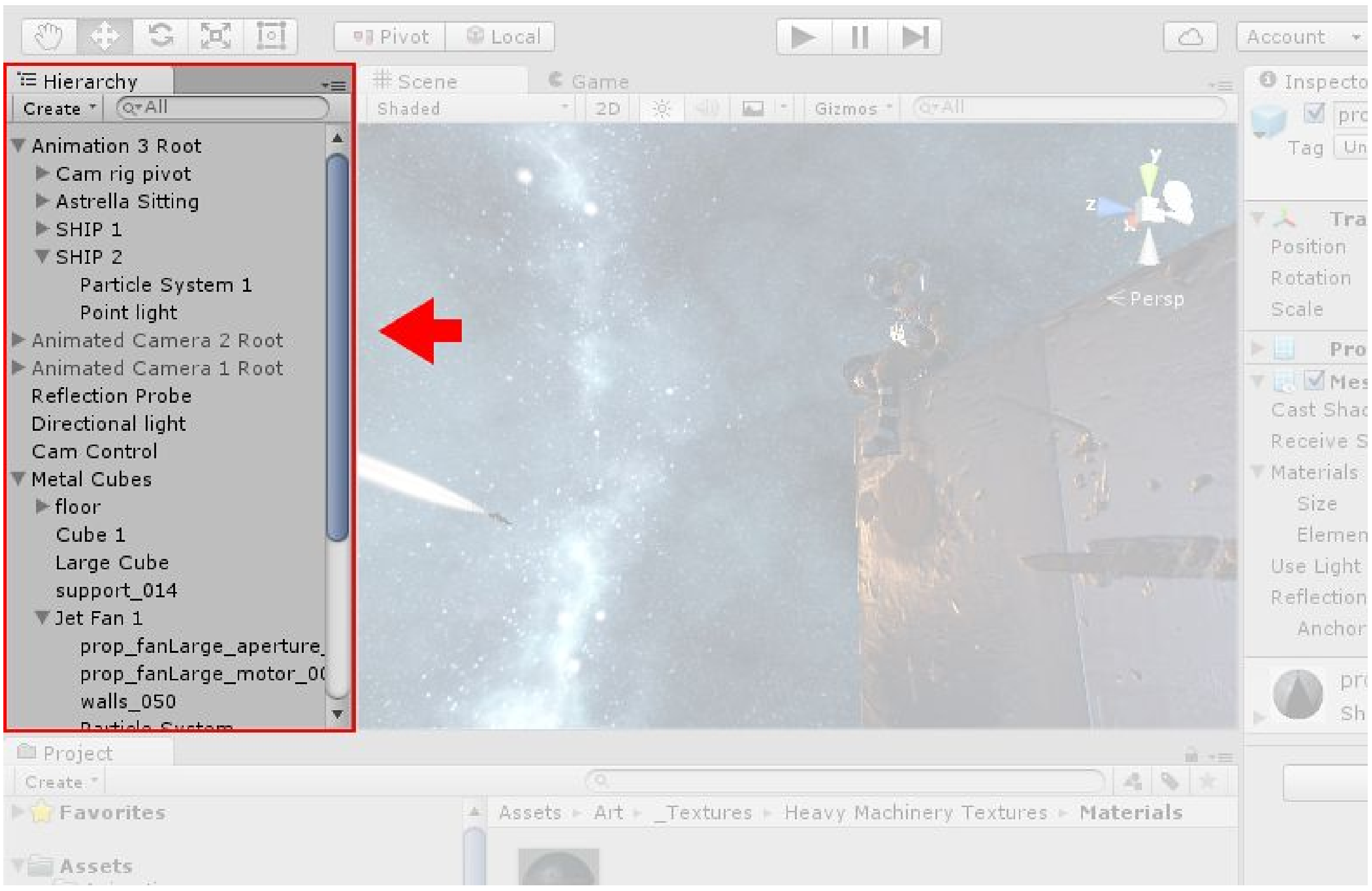


The Toolbar provides access to the most essential working features. On the left it contains the basic tools for manipulating the scene view and the objects within it. In the centre are the play, pause and step controls. The buttons to the right give you access to your Unity Cloud Services and your Unity Account, followed by a layer visibility menu, and finally the editor layout menu (which provides some alternate layouts for the editor windows, and allows you to save your own custom layouts).

The toolbar is not a window, and is the only part of the Unity interface that you can’t rearrange.

Find out more about the Toolbar.

The Hierarchy Window



```
▼ Animation 3 Root
  ▶ Cam rig pivot
  ▶ Astrella Sitting
  ▶ SHIP 1
  ▼ SHIP 2
      Particle System 1
      Point light
▶ Animated Camera 2 Root
▶ Animated Camera 1 Root
  Reflection Probe
  Directional light
  Cam Control
▼ Metal Cubes
  ▶ floor
    Cube 1
    Large Cube
    support_014
  ▼ Jet Fan 1
      prop_fanLarge_aperture_
      prop_fanLarge_motor_0(
      walls_050
```

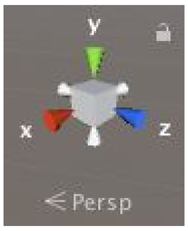
The Hierarchy Window is a hierarchical text representation of every object in the scene. Each item in the scene has an entry in the hierarchy, so the two windows are inherently linked. The hierarchy reveals the structure of how objects are attached to one another. Find out more about the Hierarchy Window.

Scene View navigation

The Scene View has a set of navigation controls to help you move around quickly and efficiently.

Scene Gizmo

The Scene Gizmo is in the upper-right corner of the Scene View. This displays the Scene View Camera's current orientation, and allows you to quickly modify the viewing angle and projection mode.



The Scene Gizmo has a conical arm on each side of the cube. The arms at the forefront are labelled X, Y and Z. Click on any of the conical axis arms to snap the Scene View Camera to the axis it represents (for example: top view, left view and front view). You can also right-click the cube to bring up a menu with a list of viewing angles. To return to the default viewing angle, right-click the Scene Gizmo and click Free.

You can also toggle Perspective on and off. This changes the projection mode of the Scene View between Perspective and Orthographic (sometimes called 'isometric'). To do this, click the cube in the centre of the Scene Gizmo, or the text below it. Orthographic view has no perspective, and is useful in combination with clicking one of the conical axis arms to get a front, top or side elevation.

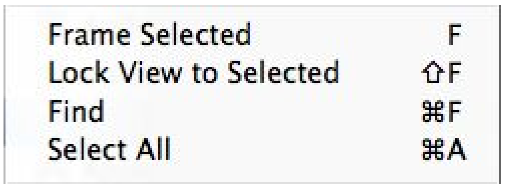
Movement shortcuts

For extra efficiency, all of these controls can also be used regardless of which transform tool is selected. The most convenient controls depend on which mouse or track-pad you are using:

Action	3-button mouse	2-button mouse or track-pad	Mac with only one mouse button or track-pad
Move	Hold Alt+middle mouse button, then drag	Hold Alt+Control+left-click, then drag	Hold Alt+Command+left-click, then drag
Orbit (Not available in 2D mode)	Hold Alt+left-click, then drag	Hold Alt+left-click, then drag	Hold Alt+left-click, then drag
Zoom	Use the scroll wheel, or hold Alt+right-click, then drag	Hold Alt+right-click, then drag	Use the two-finger swipe method to scroll in and out, or hold Alt-Control+left-click, then drag

Centering the view on a GameObject

To center the Scene View on a GameObject, select the GameObject in the Hierarchy, then move the mouse over the Scene View and press **F**. This feature can also be found in the menu bar under **Edit** > **Frame Selected**. To lock the view to the GameObject even when the GameObject is moving, press **Shift+F**. This feature can also be found in the menu bar under **Edit** > **Lock View to Selected**.



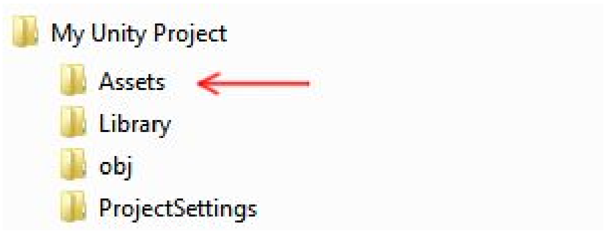
Assets

An asset is representation of any item that can be used in your game or project. An asset may come from a file created outside of Unity, such as a 3D model, an audio file, an image, or any of the other types of file that Unity supports. There are also some asset types that can be created within Unity, such as an Animator Controller, an Audio Mixer or a Render Texture.

Importing Assets

Assets created outside of Unity must be brought in to Unity by having the file either saved directly into the “Assets” folder of your project, or copied into that folder. For many common formats, you can save your source file directly into your project’s Assets folder and Unity will be able to read it. Unity will notice when you save new changes to the file and will re-import as necessary.

When you create a Unity Project, you are creating a folder - named after your project - which contains the following subfolders:

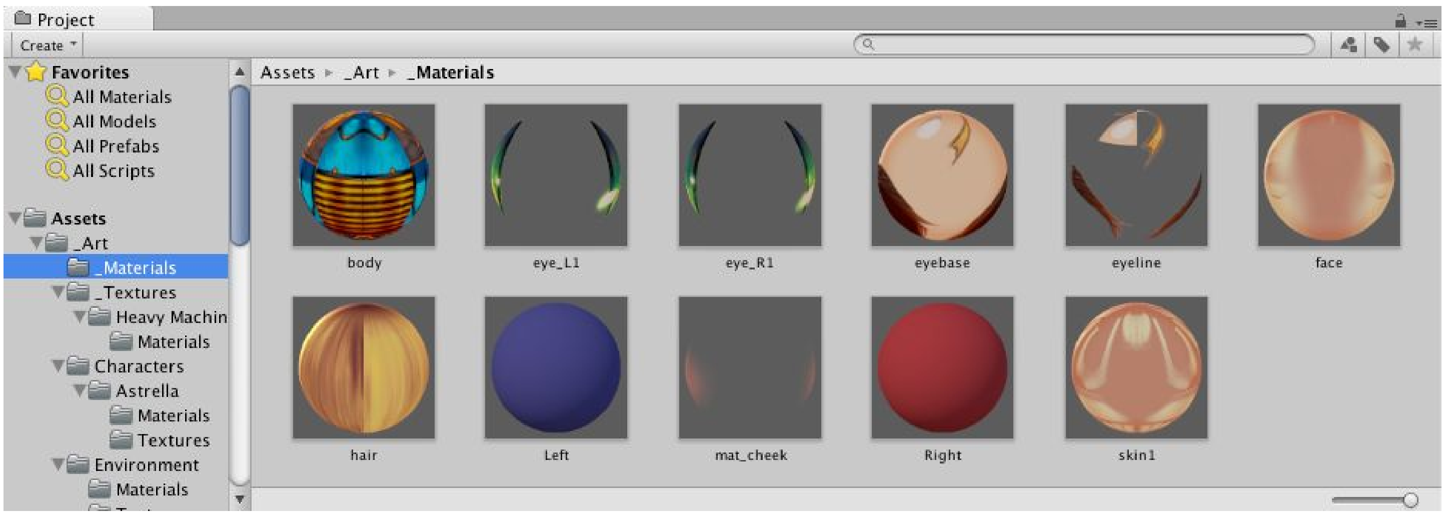


The basic file structure of a Unity Project

The Assets folder is where you should save or copy files that you want to use in your project.

The contents of the Project Window in Unity shows the items in your Assets folder. So if you save or copy a file to your Assets folder, it will be imported and become visible in your Project Window.

Unity will automatically detect files as they are added to Assets folder, or if they are modified. When you put any asset into your Assets folder, you will see the asset appear in your Project View.

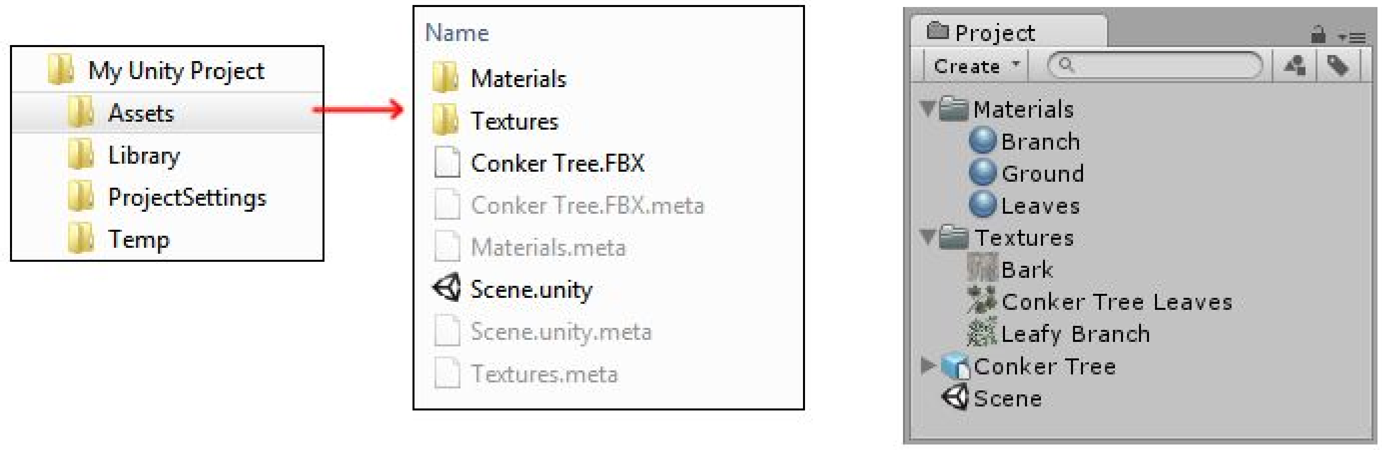


The Project Window shows assets that have been imported into your project

If you drag a file into Unity’s Project Window from your computer (eg, from the Finder on Mac, or from Explorer on Windows), it will be copied into your Assets folder, and will appear in the Project window.

The items you see in your Project window represent (in most cases) actual files on your computer, and if you delete them within Unity, you are deleting them from your computer too.

The relationship between the Assets Folder in your Unity Project on your computer, and the Project Window within Unity



The above image shows an example of a few files and folders inside the Assets folder of a Unity project. You can create as many folders as you like and use them to organise your Assets.

You'll notice in the image above that there are .meta files listed in the file system, but not visible in Unity's Project Window. Unity creates these .meta files for each asset and folder, but they are hidden by default, so you may not see them in your Explorer/Finder either.

They contain important information about how the asset is used in the project and they must stay with the asset file they relate to, so if you move or rename an asset file in Explorer/Finder, you must also move/rename the meta file to match.

The simplest way to safely move or rename your assets is to always do it from within Unity's project folder. This way, Unity will automatically move or rename the corresponding meta file. If you like, you can read more about .meta files and what goes on behind-the-scenes during the import process.

If you want to bring collections of assets into your project, you can use Asset Packages. See Asset Packages for more details.

Some common types of Asset

Image Files

Most common image file types are supported, such as BMP, TIF, TGA, JPG, and PSD. If you save your layered Photoshop (.psd) files into your Assets folder, they will be imported as flattened images. You can find out more about importing images with alpha channels from photoshop, or importing your images as sprites

3D Model Files

If you save your 3D files from most common 3D software packages in their native format (eg, .max, .blend, .mb, .ma) into your Assets folder, they will be imported by calling back to your 3D package's FBX export plugin (*). Alternatively you can export as FBX from your 3D app into your Unity project. Read more about importing 3D files from your 3D app.

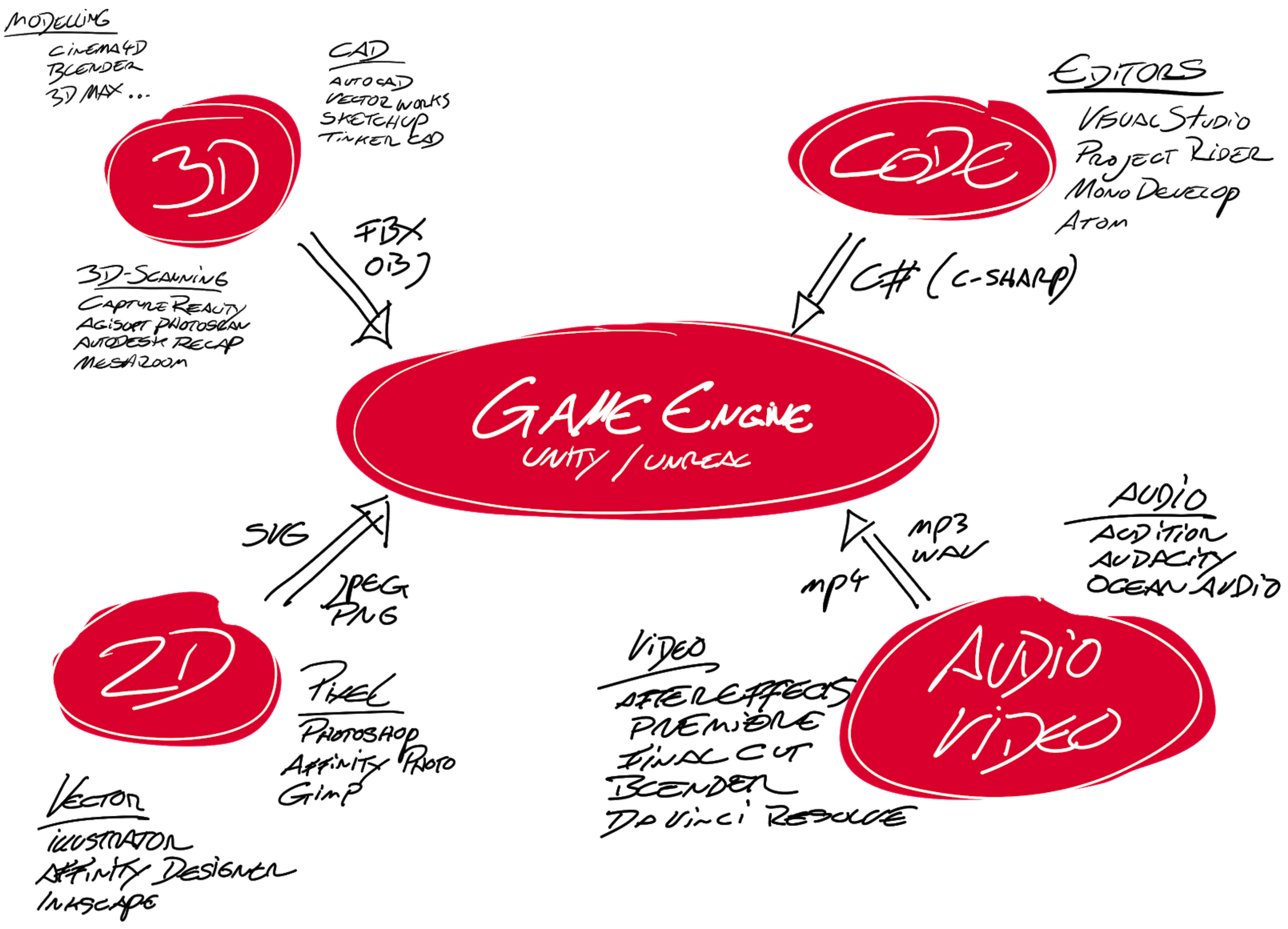
Meshes & Animations

Whichever 3D package you are using, Unity will import the meshes and animations from each file. For a list of applications that are supported by Unity, please see this page.

Your mesh file does not need to have an animation to be imported. If you do use animations, you have your choice of importing all animations from a single file, or importing separate files, each with one animation. For more information about importing animations, please see Importing animations.

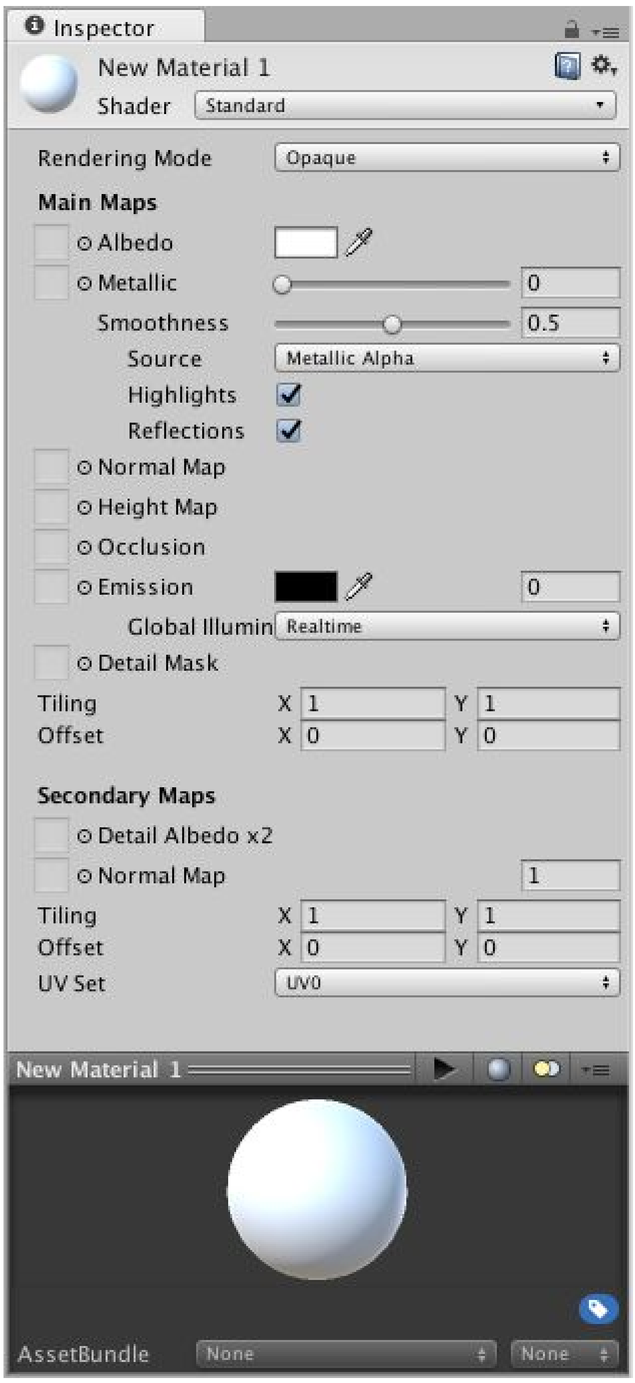
Audio Files

If you save uncompressed audio files into your Assets folder, they will be imported according to the compression settings specified. Read more about importing audio files.



Creating and Using Materials

To create a new Material, use Assets->Create->Material from the main menu or the Project View context menu.
By default, new materials are assigned the Standard Shader, with all map properties empty, like this:



Inspector

New Material 1
Shader: Standard

Rendering Mode: Opaque

Main Maps
- Albedo
- Metallic: 0
- Smoothness: 0.5
- Source: Metallic Alpha
- Highlights: ✓
- Reflections: ✓
- Normal Map
- Height Map
- Occlusion
- Emission: 0
- Global Illumin: 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

New Material 1

AssetBundle: None, None

Once the Material has been created, you can apply it to an object and tweak all of its properties in the Inspector. To apply it to an object, just drag it from the Project View to any object in the Scene or Hierarchy.

Setting Material Properties

You can select which Shader you want any particular Material to use.

Simply expand the **Shader** drop-down in the Inspector, and choose your new Shader.

The Shader you choose will dictate the available properties to change.

The properties can be colors, sliders, textures, numbers, or vectors.

If you have applied the Material to an active object in the **Scene**, you will see your property changes applied to the object in real-time.

There are two ways to apply a Texture to a property:

1. Drag it from the Project View on top of the Texture square
2. Click the Select button, and choose the texture from the drop-down list that appears

The Unity **Standard Shader** is a built-in shader with a comprehensive set of features. It can be used to render “real-world” objects such as stone, wood, glass, plastic and metal, and supports a wide range of shader types and combinations. Its features are enabled or disabled by simply using or not using the various texture slots and parameters in the material editor.

Some of the **Standard Shader** settings are:

Rendering Mode. This allows you to choose whether the object uses transparency, and if so, which type of blending mode to use.

- Opaque - Is the default, and suitable for normal solid objects
- Cutout - Allows you to create a transparent effect with hard edges between. No semi-transparent areas.
- Transparent - Suitable transparent materials (glass).
- Fade - Allows the transparency values to entirely fade an object out, including any specular highlights or reflections.

Albedo Color and Transparency. The Albedo parameter controls the base color of the surface.
Specifying a single color for the Albedo value is sometimes useful, but it is far more common to assign a texture map for the Albedo parameter. This should represent the colors of the surface of the object. It’s important to note that the Albedo texture should not contain any lighting, since the lighting will be added to it based on the context in which the object is seen.

The alpha value of the Albedo colour controls the transparency level for the material. This only has an effect if the Rendering Mode for the material is set to one of the transparent mode, and not Opaque. As mentioned above, picking the correct transparency mode is important because it determines whether or not you will still see reflections and specular highlights at full value, or whether they will be faded out according to the transparency values too. A range of transparency values from 0 to 1, using the Transparent mode suitable for realistic transparent objectsWhen using a texture assigned for the Albedo parameter, you can control the transparency of the material by ensuring your albedo texture image has an alpha channel. The alpha channel values are mapped to the transparency levels with white being fully opaque, and black being fully transparent. This will have the effect that your material can have areas of varying transparency.

Asset Store access and navigation

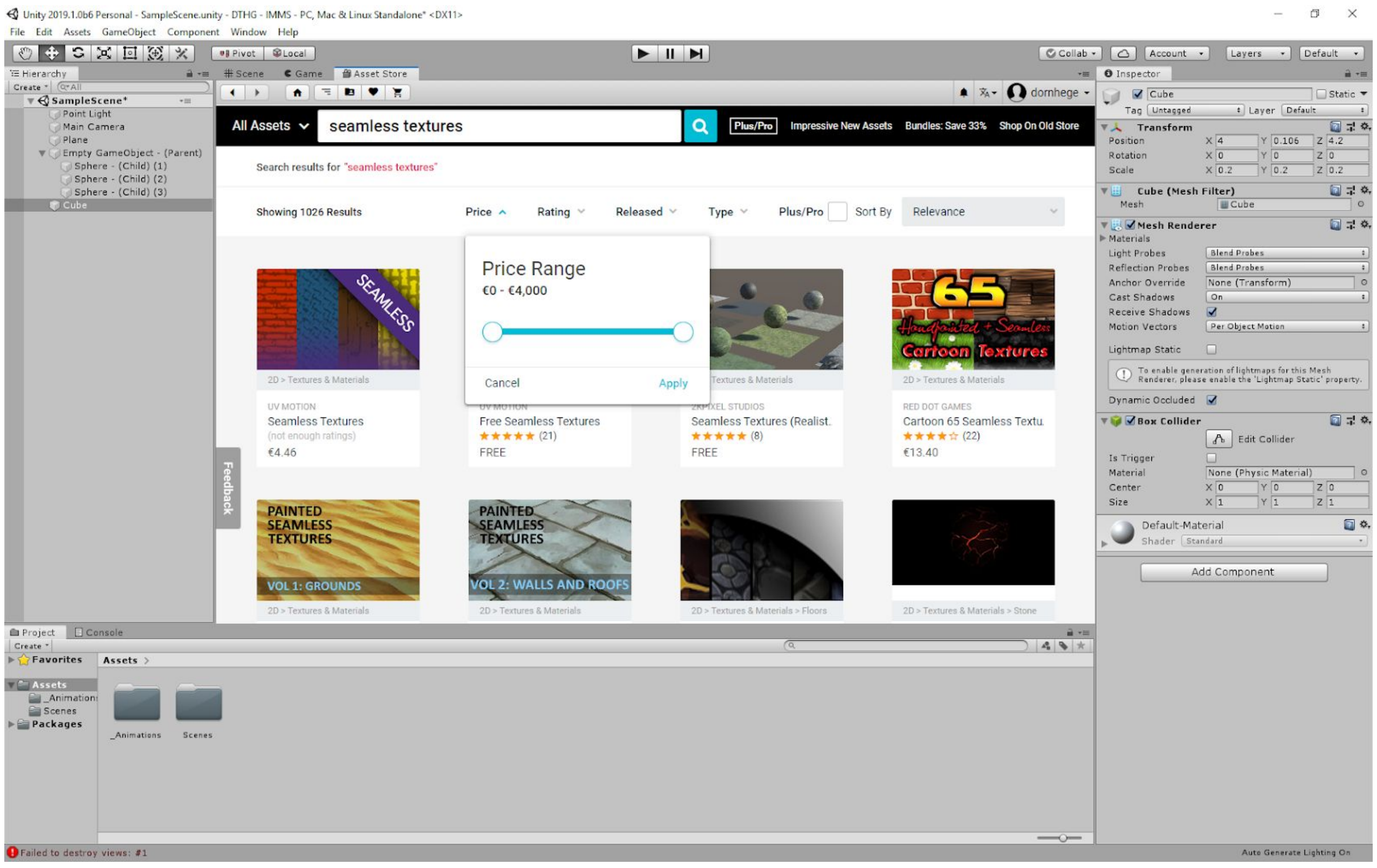
The Asset Store is home to thousands of free or affordably priced assets that save critical time and effort for Unity creators.

You can open the Asset Store window by selecting Window > Asset Store from the main menu. On your first visit, you will be prompted to create a free user account which you will use to access the Store subsequently.

There are categories at the top of the page. Click on one of these categories to filter the assets.

Use the search bar to search through every asset in the Asset store. Click the All Assets dropdown to view the Asset categories. Click on one of the categories to narrow down your search.

Also, you can use the dropdown menus under the banner to filter your search. For example, use the Price menu to look for assets within your price range by using the slider.



Unity 2019.1.0b6 Personal - SampleScene.unity - DTHG - IMMS - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Window Help
Pivot Local
Collab Account Layers Default
Hierarchy
SampleScene*
Point Light
Main Camera
Plane
Empty GameObject - (Parent)
Sphere - (Child) (1)
Sphere - (Child) (2)
Sphere - (Child) (3)
Cube
Scene Game Asset Store
dornhege
All Assets
seamless textures
Plus/Pro Impressive New Assets Bundles: Save 33% Shop On Old Store
Search results for "seamless textures"
Showing 1026 Results
Price Rating Released Type Plus/Pro Sort By Relevance
Price Range
€0 - €4,000
Cancel
Apply
UV MOTION
Seamless Textures
(not enough ratings)
€4.46
Free Seamless Textures
(21)
FREE
Seamless Textures (Realist.
(8)
FREE
RED DOT GAMES
Cartoon 65 Seamless Textu
(22)
€13.40
2D > Textures & Materials
2D > Textures & Materials > Floors
2D > Textures & Materials > Stone
Feedback
Inspector
Cube
Static
Tag Untagged Layer Default
Transform
Position X 4 Y 0.106 Z 4.2
Rotation X 0 Y 0 Z 0
Scale X 0.2 Y 0.2 Z 0.2
Cube (Mesh Filter)
Mesh Cube
Mesh Renderer
Materials
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Cast Shadows On
Receive Shadows
Motion Vectors Per Object Motion
Lightmap Static
To enable generation of lightmaps for this Mesh Renderer, please enable the 'Lightmap Static' property.
Dynamic Occluded
Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
Default-Material
Shader Standard
Add Component
Project Console
Favorites
Assets
_Animations
Scenes
Packages
_Animations Scenes
Failed to destroy views: #1
Auto Generate Lighting On

Use the Sort By dropdown menu to change the order of the Assets according to your preferences.

*.unitypackage - files

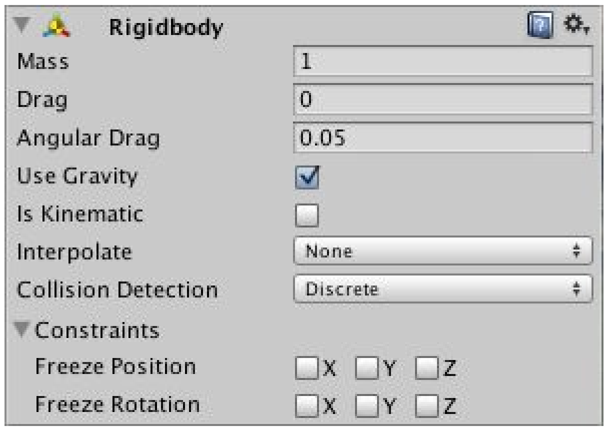
Assets that already have been downloaded are stored as *.unitypackage files on your hard disk - you can find the under:

macOS: ~/Library/Unity/Asset Store
Windows: C:\Users\accountName\AppData\Roaming\Unity\Asset Store

You can copy them to a different location so you can store and install them at a later time.

Rigidbody-Component

Rigidbodies enable your GameObjects to act under the control of physics. The Rigidbody can receive forces and torque to make your objects move in a realistic way. Any GameObject must contain a Rigidbody to be influenced by gravity, act under added forces via scripting, or interact with other objects through the NVIDIA PhysX physics engine.


Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	☐
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	☐ X ☐ Y ☐ Z
Freeze Rotation	☐ X ☐ Y ☐ Z

Property:	Function:
Mass	The mass of the object (in kilograms by default).
Drag	How much air resistance affects the object when moving from forces. 0 means no air resistance, and infinity makes the object stop moving immediately.
Angular Drag	How much air resistance affects the object when rotating from torque. 0 means no air resistance. Note that you cannot make the object stop rotating just by setting its Angular Drag to infinity.
Use Gravity	If enabled, the object is affected by gravity.
Is Kinematic	If enabled, the object will not be driven by the physics engine, and can only be manipulated by its Transform. This is useful for moving platforms or if you want to animate a Rigidbody that has a HingeJoint attached.

Physic Material

The Physic Material is used to adjust friction and bouncing effects of colliding objects. To create a Physic Material select Assets > Create > Physic Material from the menu bar. Then drag the Physic Material from the Project View onto a Collider in the scene.

Dynamic Friction	The friction used when already moving. Usually a value from 0 to 1. A value of zero feels like ice, a value of 1 will make it come to rest very quickly unless a lot of force or gravity pushes the object.
Static Friction	The friction used when an object is laying still on a surface. Usually a value from 0 to 1. A value of zero feels like ice, a value of 1 will make it very hard to get the object moving.
Bounciness	How bouncy is the surface? A value of 0 will not bounce. A value of 1 will bounce without any loss of energy.
Friction Combine	How the friction of two colliding objects is combined.
- Average	The two friction values are averaged.
- Minimum	The smallest of the two values is used.
- Maximum	The largest of the two values is used.
- Multiply	The friction values are multiplied with each other.
Bounce Combine	How the bounciness of two colliding objects is combined. It has the same modes as Friction Combine Mode

Types of light

Point lights
A point light is located at a point in space and sends light out in all directions equally. The direction of light hitting a surface is the line from the point of contact back to the center of the light object. The intensity diminishes with distance from the light, reaching zero at a specified range. Light intensity is inversely proportional to the square of the distance from the source. This is known as 'inverse square law' and is similar to how light behaves in the real world.

Point lights are useful for simulating lamps and other local sources of light in a scene.

Spot lights
Like a point light, a spot light has a specified location and range over which the light falls off. However, the spot light is constrained to an angle, resulting in a cone-shaped region of illumination. The center of the cone points in the forward (Z) direction of the light object. Light also diminishes at the edges of the spot light's cone. Widening the angle increases the width of the cone and with it increases the size of this fade, known as the 'penumbra'.

Spot lights are generally used for artificial light sources such as flashlights, car headlights and searchlights.

Directional lights
Directional lights are very useful for creating effects such as sunlight in your scenes. Behaving in many ways like the sun, directional lights can be thought of as distant light sources which exist infinitely far away. A directional light does not have any identifiable source position and so the light object can be placed anywhere in the scene. All objects in the scene are illuminated as if the light is always from the same direction. The distance of the light from the target object is not defined and so the light does not diminish.

Directional lights represent large, distant sources that come from a position outside the range of the game world.

By default, every new Unity scene contains a Directional Light. In Unity 5, this is linked to the procedural sky system defined in the Environment Lighting section of the Lighting Panel (Lighting>Scene>Skybox). Rotating the default Directional Light (or 'Sun') causes the 'Skybox' to update. With the light angled to the side, parallel to the ground, sunset effects can be achieved. Additionally, pointing the light upwards causes the sky to turn black, as if it were nighttime. With the light angled from above, the sky will resemble daylight. If the Skybox is selected as the ambient source, Ambient Lighting will change in relation to these colors.

Area lights
An Area Light is defined by a rectangle in space. Light is emitted in all directions uniformly across their surface area, but only from one side of the rectangle. There is no manual control for the range of an Area Light, however intensity will diminish at inverse square of the distance as it travels away from the source. Since the lighting calculation is quite processor-intensive, area lights are not available at runtime and can only be baked into lightmaps.

Since an area light illuminates an object from several different directions at once, the shading tends to be more soft and subtle than the other light types. Light is emitted across the surface of an Area Light producing a diffuse light with soft shadowing.

Emissive materials
Like area lights, emissive materials emit light across their surface area. They contribute to bounced light in your scene and associated properties such as color and intensity can be changed during gameplay. Whilst area lights are not supported by Precomputed Realtime GI, similar soft lighting effects in realtime are still possible using emissive materials.

'Emission' is a property of the Standard Shader which allows static objects in our scene to emit light. Emission will only be received by objects marked as 'Static' or "Lightmap Static' from the Inspector.

Emissive materials only directly affect static geometry in your scene. If you need dynamic, or non-static geometry - such as characters, to pick up light from emissive materials, Light Probes must be used.

Ambient light
Ambient light is light that is present all around the scene and doesn't come from any specific source object. It can be an important contributor to the overall look and brightness of a scene. Ambient light can be useful in a number of cases, depending upon your chosen art style. An example would be bright, cartoon-style rendering where dark shadows may be undesirable or where lighting is perhaps hand-painted into textures. Ambient light can also be useful if you need to increase the overall brightness of a scene without adjusting individual lights.

Ambient light settings can be found in the Lighting window.

Skyboxes

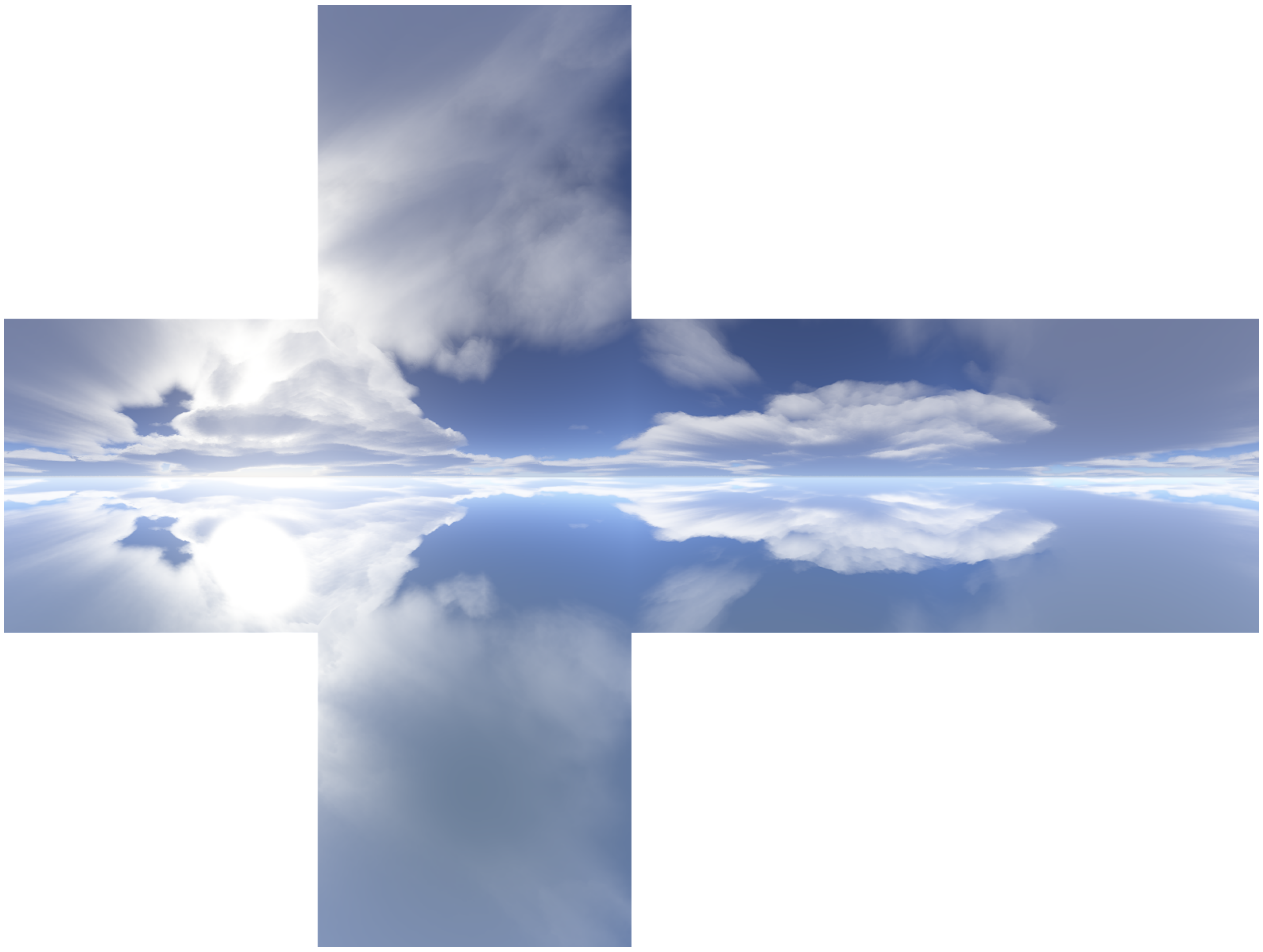
A skybox is a panoramic texture drawn behind all objects in the scene to represent the sky or any other vista at a great distance. This lesson explains how to use skyboxes in Unity.

A skybox is a panoramic view split into six textures representing six directions visible along the main axes (up, down, left, right, forward and backward). If the skybox is correctly generated, the texture images will fit together seamlessly at the edges to give a continuous surrounding image that can be viewed from "inside" in any direction. The panorama is rendered behind all other objects in the scene and rotates to match the current orientation of the camera (it doesn't vary with the position of the camera, which is always taken to be at the centre of the panorama). A skybox is thus an easy way to add realism to a scene with minimal load on the graphics hardware.

You can find free skyboxes in the asset store or create your own Sykbox Materials:
https://docs.unity3d.com/Manual/HOWTO-UseSkybox.html

The Skybox Material can be defined separately for the scene and each individual camera:

Scene:	**Window > Lighting > Settings** **Environment, Skybox Material: _______**
Camera	**Inspector, Component Properties, Camera, Background**



Lighting Overview

https://unity3d.com/learn/tutorials/topics/graphics/introduction-lighting-and-rendering?playlist=17102

Realtime Lighting
By default, lights in Unity - directional, spot and point, are realtime. This means that they contribute direct light to the scene and update every frame. As lights and GameObjects are moved within the scene, lighting will be updated immediately. This can be observed in both the scene and game views.

Realtime LightsThe effect of realtime light alone. Note that shadows are completely black as there is no bounced light. Only surfaces falling within the cone of the Spotlight are affected.

Realtime lighting is the most basic way of lighting objects within the scene and is useful for illuminating characters or other movable geometry.

Unfortunately, the light rays from Unity's realtime lights do not bounce when they are used by themselves. In order to create more realistic scenes using techniques such as global illumination we need to enable Unity's precomputed lighting solutions.

Baked GI Lighting
When 'baking' a 'lightmap', the effects of light on static objects in the scene are calculated and the results are written to textures which are overlaid on top of scene geometry to create the effect of lighting.

These 'lightmaps' can include both the direct light which strikes a surface and also the 'indirect' light that bounces from other objects or surfaces within the scene. This lighting texture can be used together with surface information like color (albedo) and relief (normals) by the 'Shader' associated with an object's material.

With baked lighting, these light textures **(lightmaps) cannot change during gameplay** and so are referred to as 'static'. Realtime lights can be overlaid and used additively on top of a lightmapped scene but cannot interactively change the lightmaps themselves.

With this approach, we trade the ability to move our lights at gameplay for a potential increase in performance, suiting less powerful hardware such as mobile platforms.

Precomputed Realtime GI Lighting
Whilst traditional, static lightmaps are unable to react to changes in lighting conditions within the scene, Precomputed Realtime GI does offer us a technique for updating complex scene lighting interactively.

With this approach it is possible to create lit environments featuring rich global illumination with bounced light which responds, in realtime, to lighting changes. A good example of this would be a time of day system - where the position and color of the light source changes over time. With traditional baked lighting, this is not possible.

In order to deliver these effects at playable framerates, we need to shift some of the lengthy number-crunching from being a realtime process, to one which is 'precomputed'.

Enabling Baked GI or Realtime GI

Unity enables both Baked GI and Realtime GI by default. Baked GI is all precomputed; Realtime GI carries out some precomputation when indirect lighting is used. With both enabled, you then use each individual Light in your Scene to control which GI system it should use (in the Light component, use the Mode setting to do this). To make your game less resource-intensive, you can choose to disable Realtime GI or Baked GI.

To manually enable or disable Global Illumination, open the Lighting window (Window > Lighting Settings > Scene). Tick Realtime Global Illumination to enable Realtime GI, and tick Baked Global Illumination to enable Baked GI.

Per-Light Settings

The default baking mode for each light is 'Realtime'. This means that the selected light(s) will still contribute direct light to your scene, with indirect light handled by Unity's Precomputed Realtime GI system.

However, if the baking mode is set to 'Baked' then that light will contribute lighting solely to Unity's Baked GI system. Both direct and indirect light from those lights selected will be 'baked' into lightmaps and cannot be changed during gameplay.

Selecting the 'Mixed' baking mode, GameObjects marked as static will still include this light in their Baked GI lightmaps. However, unlike lights marked as 'Baked', Mixed lights will still contribute realtime, direct light to non-static GameObjects within your scene. This can be useful in cases where you are using lightmaps in your static environment, but you still want a character to use these same lights to cast realtime shadows onto lightmapped geometry.

Precomputed realtime global illumination

A detailed introduction can be found here: https://unity3d.com/learn/tutorials/topics/graphics/realtime-resolution?playlist=17102
(Some settings have changed in newer versions - for example Realtime Resolution is now Indirect Resolution)

When working with Precomputed Realtime GI, a lighting precompute is the process of calculating the bounce of light around the static geometry within a Scene in the Unity Editor and storing this data for use at run time. This process reduces the number of lighting calculations that must be performed at run time, allowing realtime bounced lighting while maintaining interactive framerates.

Indirect Resolution
When setting up a Scene it is important to have some idea of the unit scale your project will need. And set the Indirect Resolution accordingly:

(Window > Lighting Settings > Scene > Lightmapping Settings > Indirect Resolution

Suitable values are around 2 - 3 texels per unit for indoor Scenes, and 0.5 - 1 for outdoor environments.

Prepare objects for Precomputed Realtime GI
For the precomputing process to begin, we must have at least one object marked as Static or Lightmap Static in our Scene. Select the GameObject that is going to be Static > In the Inspector window, check the box marked Static.

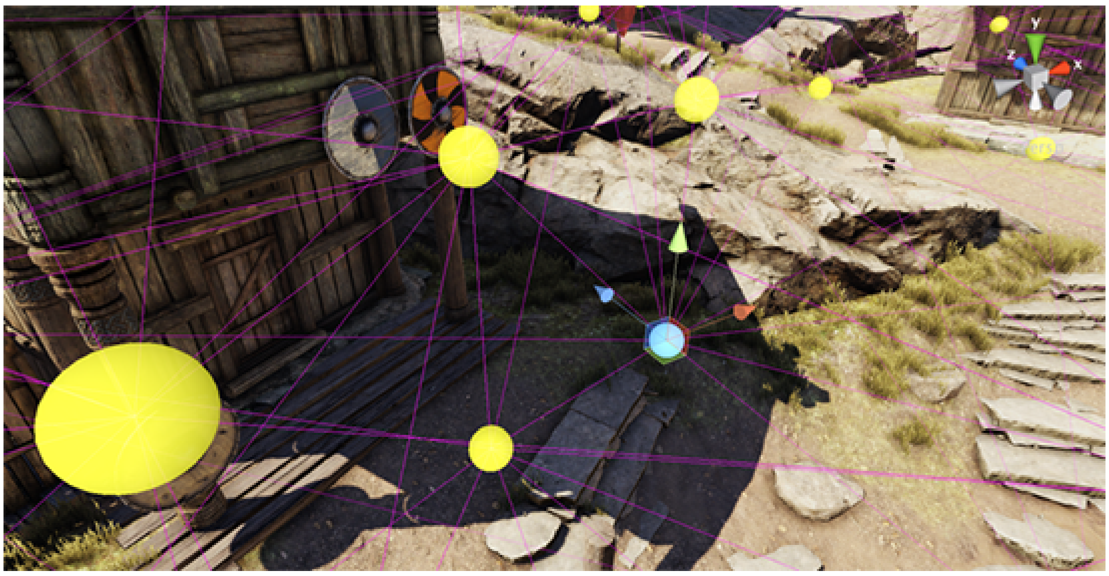
Probe lighting
Probe lighting is a fast technique for approximating lighting in realtime rendering. It is commonly used for non-Static (dynamic) objects. Probe lighting is very performant at run time and has the added benefit that it is quick to precompute.

Probe lighting works by sampling the incoming lighting at a specific point in 3D space and encoding this information across a sphere using mathematical functions known as spherical harmonics. These coefficients have low storage cost and can be then be quickly 'unpacked' at gameplay and used by shaders within the Scene to approximate surface lighting. In Unity, this functionality is offered by Light Probes.

To set up Light Probes, begin by creating a Light Probe Group from the GameObject menu:

- GameObject > Light > Light Probe Group
- Select the newly created Light Probe Group in the Hierarchy view.
- In the Inspector panel, select Edit Light Probes from the Light Probe Group component.
- You can select the probes themselves in the Scene view.
- Position the probes – you can duplicate existing probes by pressing Ctrl+D (Cmd+D on Mac).

When deciding what is an area of lighting interest, look for areas of shadow, or perhaps where there is a strong change in the color of the Terrain material. Remember we are aiming to sample indirect, or bounced, lighting throughout the Scene. In order to justify the cost of adding a new probe, we must ensure that it is sampling some noticeable change in the lighting. If we were to use our Light Probes to sample areas with general or consistent lighting, then we are unlikely to see much change as the receiving object passes these probes. Like many aspects of game optimisation, we have to ensure every item we include in our Scene offers some benefit.



The Particle System

A detailed tutorial can be found here: https://www.raywenderlich.com/138-introduction-to-unity-particle-systems
or here: https://unity3d.com/de/learn/tutorials/topics/graphics/introduction-and-goals?playlist=17102

A particle system generally emits particles in random positions within a predefined space, which can have a shape like a sphere or a cone. The system determines the lifetime of the particle itself, and when that lifetime expires, the system destroys the particle.

Create a particle emitter

- Create a new empty GameObject or select an existing one
- Inside the Inspector, click the Add Component button. Search for Particle System and click to add it

Note: You might see pink particles instead of white, which seems to be a Unity bug with setting the default texture. If that's the case, don't worry: you will set the proper fire texture shortly. If you'd like to fix it now, click the Renderer section of the particle system, click the Dot next to the Material field and double-click Default-Particle in the window that pops up.

When you select a GameObject with an attached particle system, you'll notice a black dialog in the lower right hand corner of the scene view. This dialog lets you simulate or stop the particle system. Clicking Simulate activates your particle system and changes the button to a "Pause" button. To stop the simulation, click the "Stop" button.

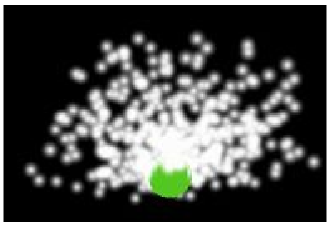
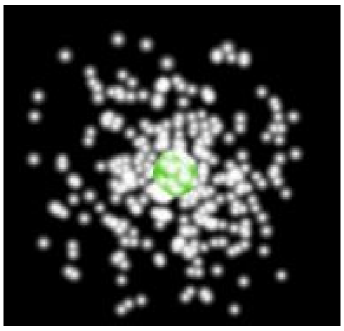
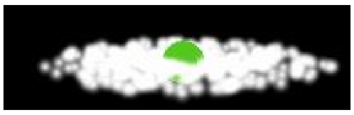
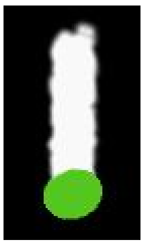
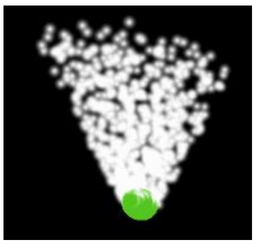
The particle system Component you added has several subsections. Each of these subsections is called a Module. These Modules contain the settings for the particle system.

Note: As you modify the settings of the particle system, you'll see a preview of it in the Game Window.

Main module (excerpt)
Duration: The length of time in seconds for the particle system to run
Looping: Repeatedly emit particles until the particle system stops. The cycle restarts once the Duration time is reached
Prewarm: Only used when Looping is enabled. The Particle System will act as if it's already completed a full cycle on start-up
Start Delay: The delay in seconds before the particle system starts emitting
Start Lifetime: The initial lifetime in seconds for the particles. The particle is destroyed after this elapsed time
Start Speed: The initial speed of the particles. The greater the speed of the particles, the more spread out they will be

Emission Module (excerpt)
Rate over Time: represents the number of particles emitted per second

Shape Module (excerpt)
The Shape module, as the name implies, controls the shape and the behavior of particles in that shape. You can choose from several different shapes; each has their own particular settings. Here are some examples:

HemiSphere	Sphere	Circle	Box	Cone

Render Module (excerpt)
All particles have a particle material and a texture that defines how they look. There's only so much you can do with the default texture. By changing the texture, you can create effects like magic stars, smoke and of course, fire.
Material: Defines the material used for the particles

Size Over Lifetime Module
With the Size over Lifetime module, you can create particles that grow or shrink during their lifetime, or even pulsate.

Audio Listeners & Sources

Unity's audio system is flexible and powerful. It can import most standard audio file formats and has sophisticated features for playing sounds in 3D space, optionally with effects like echo and filtering applied.

To play sounds in Unity, you use an Audio Listener, an AudioSource and an AudioClip:

The **AudioListener** acts as a microphone-like device. It receives input from any given Audio Source in the scene and plays sounds through the computer speakers. For most applications it makes the most sense to attach the listener to the Main Camera. Every scene can contain various Audio Sources, but only one Audio Listener.

An **AudioSource** is what will actually play the sound in 2D or 3D space. In 3D space, the sound's volume can vary based on how far the AudioSource is from the object listening to it.

You can set an AudioSource to play sound in 2D space, which means it will play at a consistent volume regardless of the distance from the AudioListener.

An **AudioClip** is the actual audio file that the AudioSource will play.

Important Settings for AudioSources

● Audio Clip	Reference to the sound clip file that will be played
● Output	The sound can be output through an audio listener or an audio mixer
● Mute	If enabled the sound will be playing but muted
● Play On Awake	If enabled, the sound will start playing the moment the scene launches (If disabled, you need to start it using the Play() command from scripting.)
● Loop	Enable this to make the Audio Clip loop when it reaches the end
● Volume	How loud the sound is at a distance of one world unit (one meter) from the Audio Listener
● Pitch	Amount of change in pitch due to slowdown/speed up of the Audio Clip. Value 1 is normal playback
● Spatial Blend	Sets how much the 3D engine has an effect on the audio source
● Doppler Level	Determines how much doppler effect will be applied to this audio source (if is set to 0, then no effect is applied)
● Min Distance	Within the MinDistance, the sound will stay at loudest possible. Outside MinDistance it will begin to attenuate.
● Max Distance	The distance where the sound stops attenuating at. Beyond this point it will stay at the volume it would be at MaxDistance

Audio Mixer and Audio Mixer Groups

The Unity Audio Mixer allows you to mix various audio sources, apply effects to them, and perform mastering.

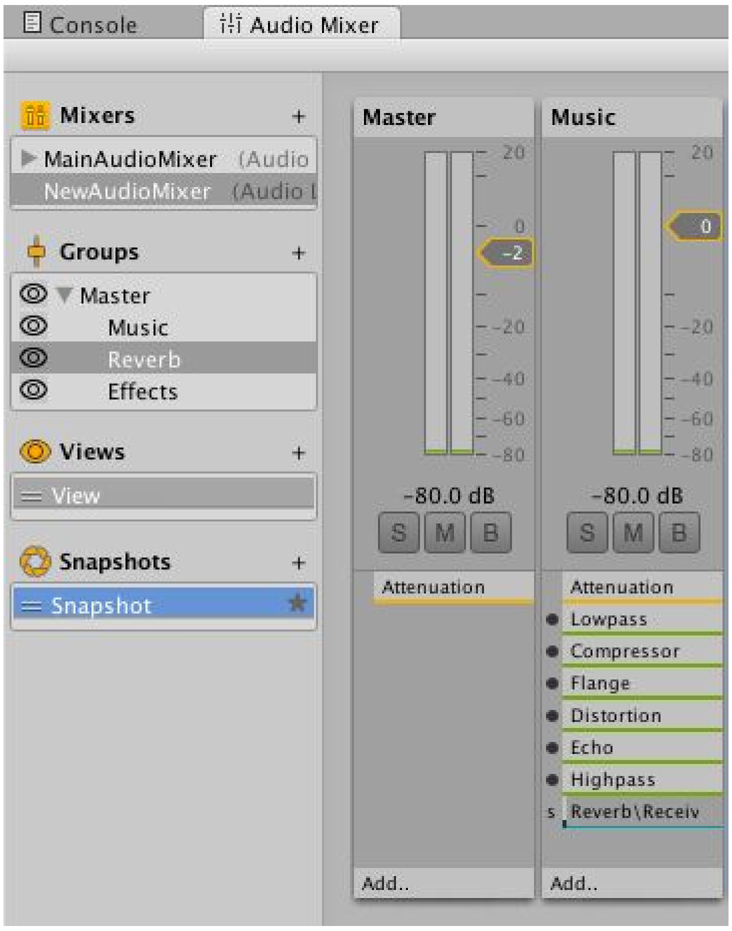
To create a new Audio Mixer Asset in your assets folder:
(Right Click > Create > Audio Mixer)
You can open the Audio Mixer window by double clicking on an Audio Mixer Asset or selecting:
Window > Audio > Audio Mixer

Channels are referred to as "Groups". You can create new »Groups« by clicking on the + next to »Groups«

Once you have set up your Mixer and Groups you can assign those groups to different Audio Sources:
Activate a Audio Source in the
Hierarchy > Inspector > Audio Source > Output:

Select the desired Mixer Group by dragging & dropping it from the Audio Mixer window or through the context search (the small Icon next to the empty field).

You can now level the specific sound groups and add additional sound effects.



The concept of Colliders as Triggers

Colliders are components that allow GameObjects they're attached to to react to other colliders - provided that one of the game objects has a rigidbody component attached.

In order to make a collider into a trigger we simply check the 'Is Trigger' checkbox on the Collider-Component setting in the inspector. When a collider is a trigger things will no longer bump into it. Instead they will pass through it and this can be detected via code.

The following states can be detected:

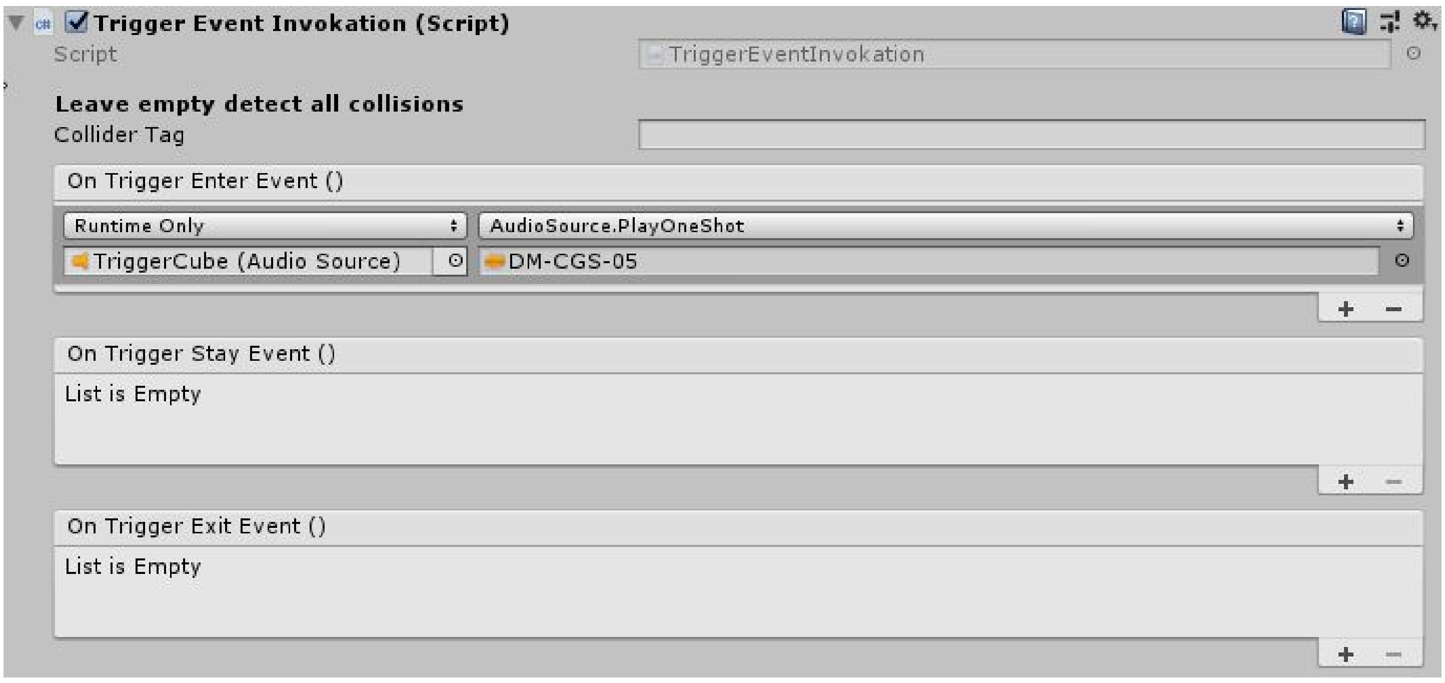
OnTriggerEnter	OnTriggerEnter is called when the Collider other enters the trigger.
OnTriggerExit	OnTriggerExit is called when the Collider other has stopped touching the trigger.
OnTriggerStay	OnTriggerStay is called almost all the frames for every Collider other that is touching the trigger. (The function is on the physics timer so it won't necessarily run every frame.)

One way to react to Trigger-Collisions is to use code - a simpler way is using a script provided with the Workshop Material: **TriggerEventInvokation.cs**

Unity Events

The Event System is a way of sending events to objects in the application based on input, be it keyboard, mouse, touch, or custom input through scripts.

You can control other GameObjects through by attaching a script using the EventSystem and providing open Event slots.


Trigger Event Invokation (Script)
Script: TriggerEventInvokation
Leave empty detect all collisions
Collider Tag:
On Trigger Enter Event ()
Runtime Only | AudioSource.PlayOneShot
TriggerCube (Audio Source) | DM-CGS-05
On Trigger Stay Event ()
List is Empty
On Trigger Exit Event ()
List is Empty

- Select the + icon to add a slot for a callback
- Select the UnityEngine.Object you wish to receive the callback
  (You can use the object selector for this)
- Select the function you wish to be called
- You can add more then one callback for the event

Animations in Unity

An in depth tutorial about Animation can be found here: https://www.raywenderlich.com/1494-introduction-to-unity-animation

Unity has a rich and sophisticated animation system (sometimes referred to as ‘Mecanim’). It provides:

- Easy workflow and setup of animations for all elements of Unity including objects, characters, and properties
- **Support for imported animation clips and animation created within Unity**
- Humanoid animation retargeting - the ability to apply animations from one character model onto another
- Simplified workflow for aligning animation clips
- **Convenient preview of animation clips, transitions and interactions between them**. This allows animators to work more independently of programmers, prototype and preview their animations before gameplay code is hooked in
- **Management of complex interactions between animations with a visual programming tool**
- Animating different body parts with different logic
- Layering and masking features

Animation workflow

Unity’s animation system is based on the concept of **Animation Clips**, which contain information about how certain objects should change their position, rotation, or other properties over time. Each clip can be thought of as a single linear recording.

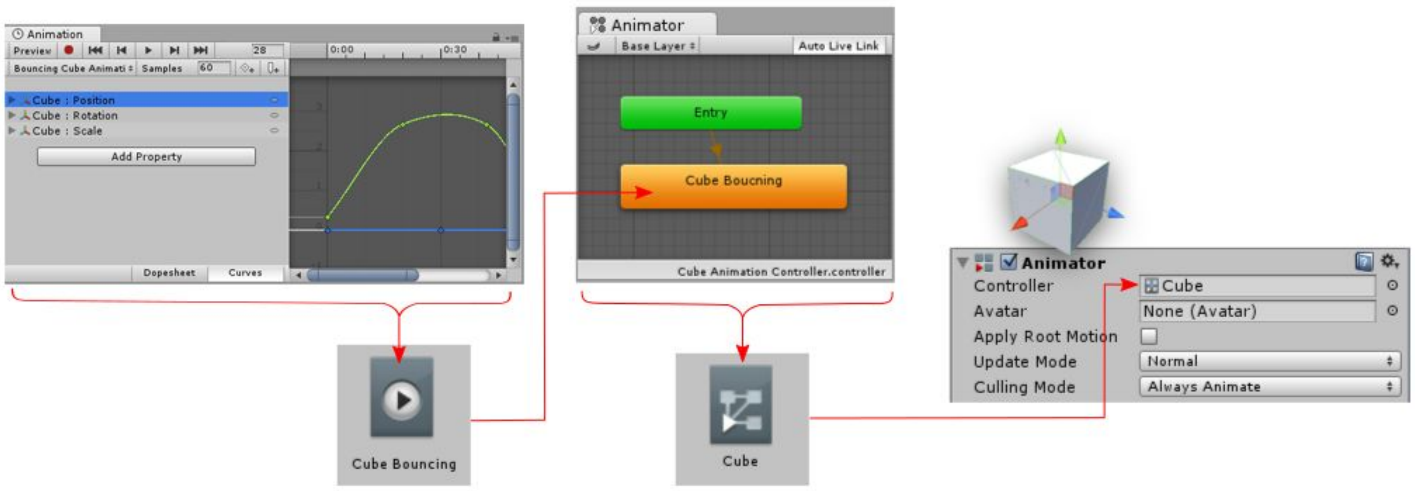
Animation Clips are then organised into a structured flowchart-like system called an **Animator Controller**. The Animator Controller acts as a “State Machine” which keeps track of which clip should currently be playing, and when the animations should change or blend together.

Animation Clips and the Animator Controller * are brought together on a GameObject via the **Animator Component**. This component has a reference to an Animator Controller. The Animator Controller, in turn, contains the references to the Animation Clips it uses. *)There is an additional element: the Avatar that is used for Character Animation

In short:

- A GameObject must have an Animator component
- The Animator component must have an Animator Controller asset assigned
- The Animator Controller asset must have one or more Animation Clips assigned

The diagram below shows how these pieces are assigned, starting from the new animation clip created in the Animation Window:



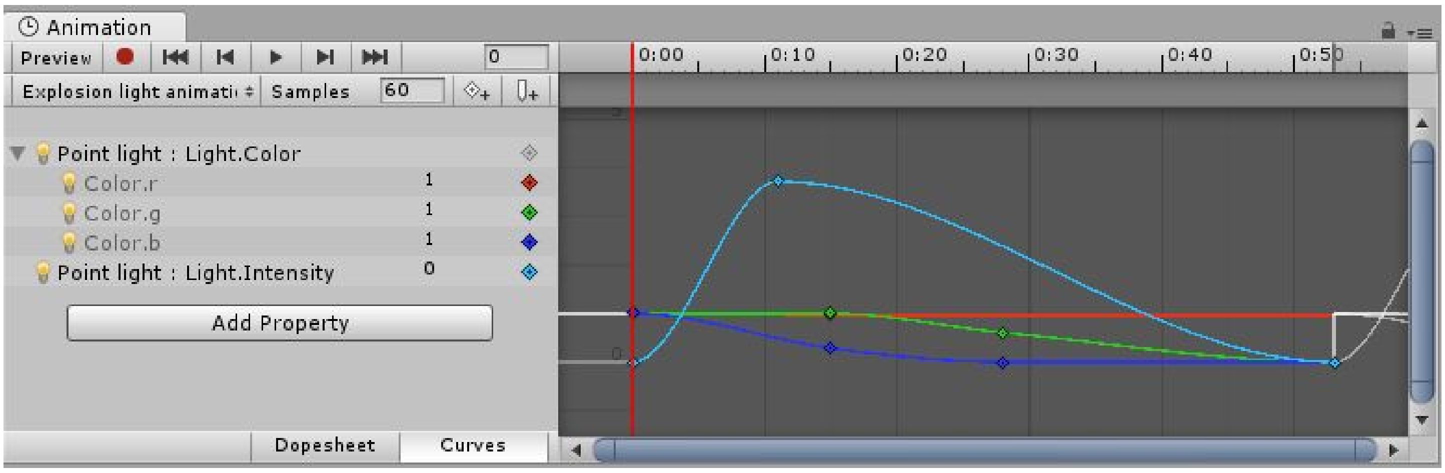
Controller: Cube
Avatar: None (Avatar)
Apply Root Motion
Update Mode: Normal
Culling Mode: Always Animate

Create and edit Animations within Unity

Visit: https://docs.unity3d.com/Manual/animeditor-CreatingANewAnimationClip.html for more details.

Unity's Animation Window also allows you to create and edit animation clips. These clips can animate:

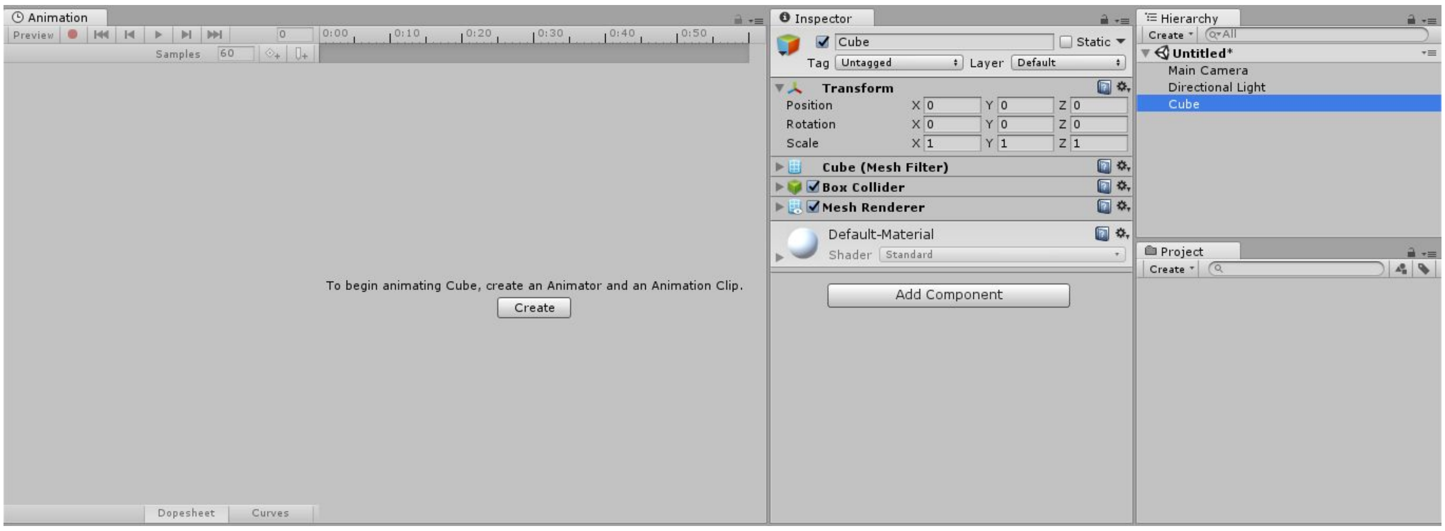
- The position, rotation and scale of GameObjects
- Component properties such as material colour, the intensity of a light, the volume of a sound
- Properties within your own scripts including float, integer, enum, vector and Boolean variables
- The timing of calling functions within your own scripts



An example of Unitys Animation window being used to animate parameters of a component - in this case, the intensity and range of a point light

Creating a New Animation Clip

- To create a new Animation Clip for the selected GameObject, and make sure the Animation Window is visible.
- If the GameObject does not yet have any Animation Clips assigned, you will see the "Create" button in the centre of the Animation Window timeline area. Click the Create button.
- You will then be prompted to save your new empty Animation Clip somewhere in your Assets folder.
  Once you have saved this new empty Animation Clip, a number of things happen automatically:
  - A new Animator Controller asset will be created
  - The new clip being created will be added into the Animator Controller as the default state
  - An Animator Component will be added to the GameObject being animated
  - The Animator Component will have the new Animator Controller assigned to it
- (If you want to create a new Animation Clip on an object that already has animations, you must select "Create New Clip" from the menu on the top left corner of the Animation Window. Again, you will be prompted to save your new empty Animation Clip before being able to work with it.)



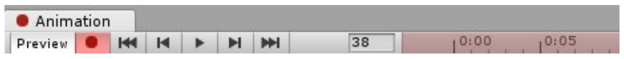
To begin animating Cube, create an Animator and an Animation Clip.

Cube						
Tag	Untagged	Layer	Default			
Transform						
Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1
Cube (Mesh Filter)						
Box Collider						
Mesh Renderer						
Default-Material						
Shader	Standard					

Animating a GameObject

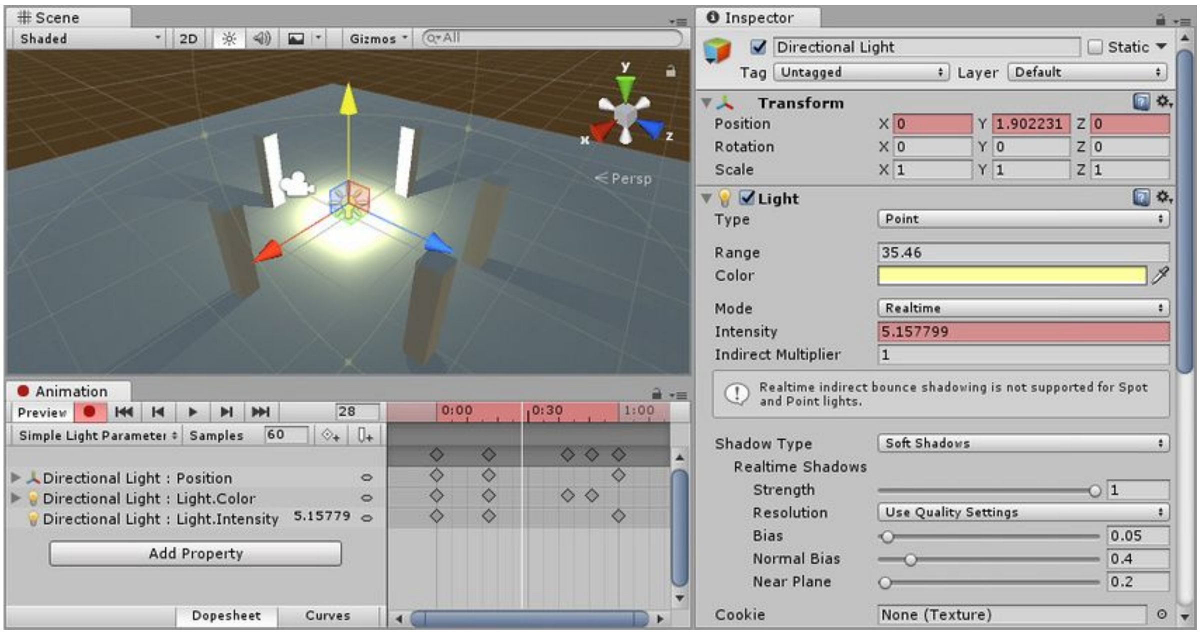
Once you have saved the new Animation clip Asset, you are ready to begin adding keyframes to the clip. There are two distinct methods you can use to animate GameObjects in the Animation window: Record Mode and Preview Mode. To get informations on how to use Preview mode, go to: https://docs.unity3d.com/Manual/animeditor-AnimatingAGameObject.html

To begin recording keyframes for the selected GameObject, click on the Animation Record button. This enters Animation Record Mode, where changes to the GameObject are recorded into the Animation Clip

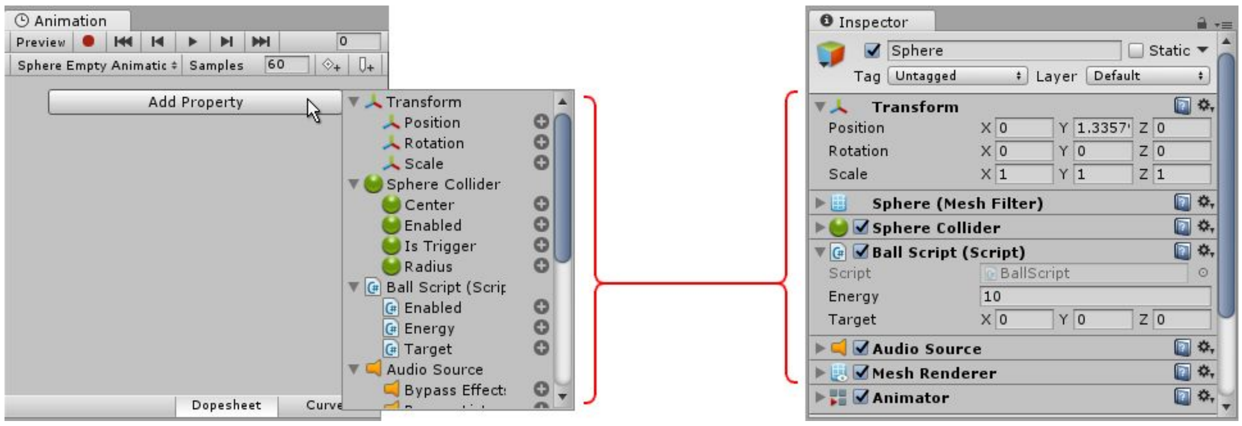


- Once in Record mode you can create keyframes by setting the white Playback head to the desired time in the Animation timeline, and then modify your GameObject to the state you want it to be at that point in time.
- The changes you make to the GameObject are recorded as keyframes at the current time shown by the white line (the playback head) in the Animation Window.
- Any change to an animatable property (such as its position or rotation) will cause a keyframe for that property to appear in the Animation window.
- Clicking or dragging in the timeline bar moves the playback head and shows the state of the animation at the playback head's current time.

In the screenshot below you can see the Animation window in record mode. The timeline bar is tinted red, indicating record mode, and the animated properties show up with a red background in the inspector.



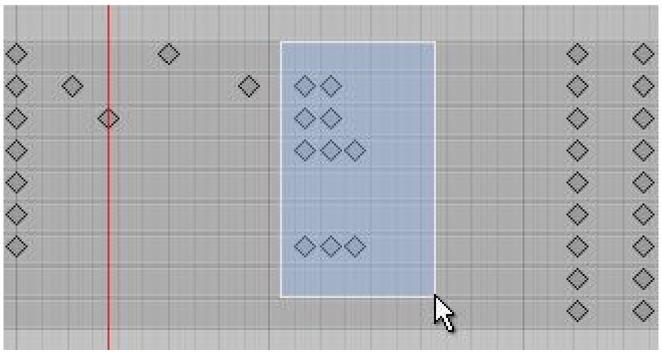
You can stop the Record Mode at any time by clicking the Record button again. When you stop Record mode, the Animation window switches to Preview mode, so that you can still see the GameObject in its current position according to the animation timeline. You can also add animatable properties to the current GameObject (and its children) by clicking the Add Property button. Clicking this button shows a pop up list of the GameObject's animatable properties. These correspond with the properties you can see listed in the inspector.



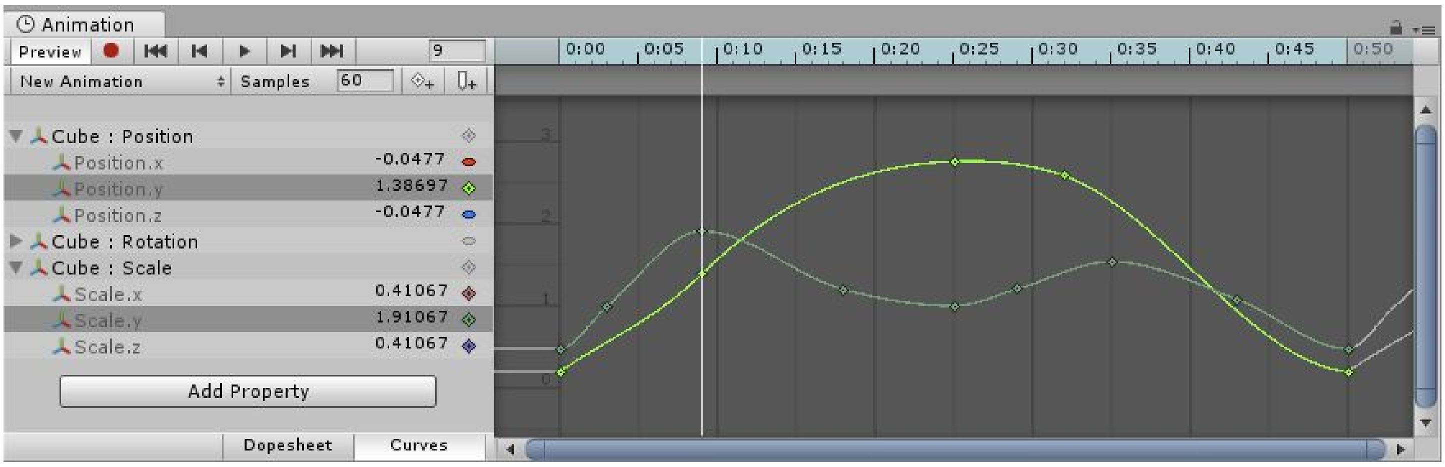
Editing Animation Clips

Animation Clips can be edited in a variety of methods. To get detailed information on each refer to the manual:

- Key manipulation in Dopesheet mode: https://docs.unity3d.com/Manual/animeditor-AdvancedKeySelectionAndManipulation.html

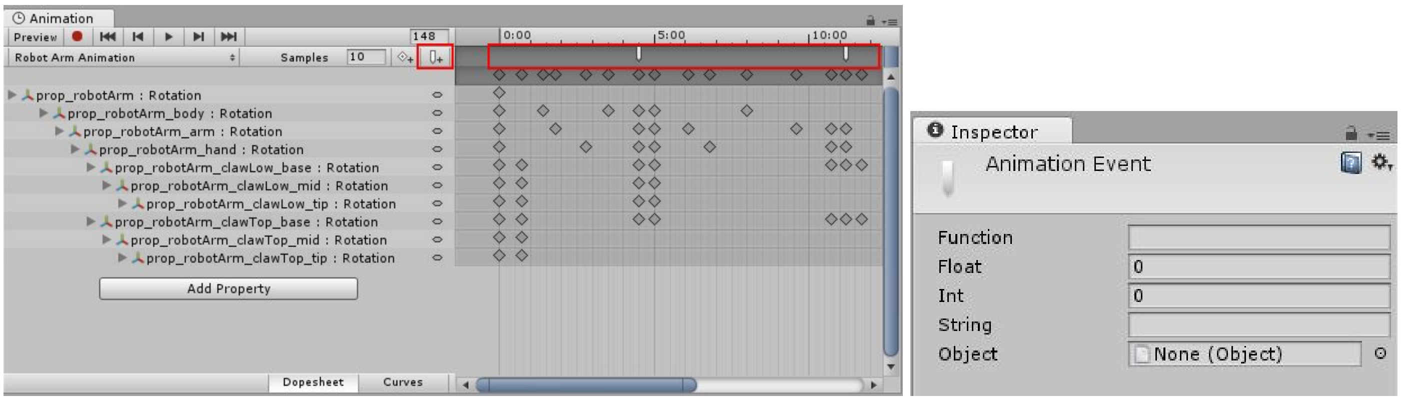


- Animation Curves:
    - https://docs.unity3d.com/Manual/animeditor-AnimationCurves.html
    - https://docs.unity3d.com/Manual/EditingCurves.html
    - https://docs.unity3d.com/Manual/animeditor-KeyManipulationInCurvesMode.html



Cube : Position
Position.x -0.0477
Position.y 1.38697
Position.z -0.0477
Cube : Rotation
Cube : Scale
Scale.x 0.41067
Scale.y 1.91067
Scale.z 0.41067

- Animation Events: https://docs.unity3d.com/Manual/animeditor-AnimationEvents.html

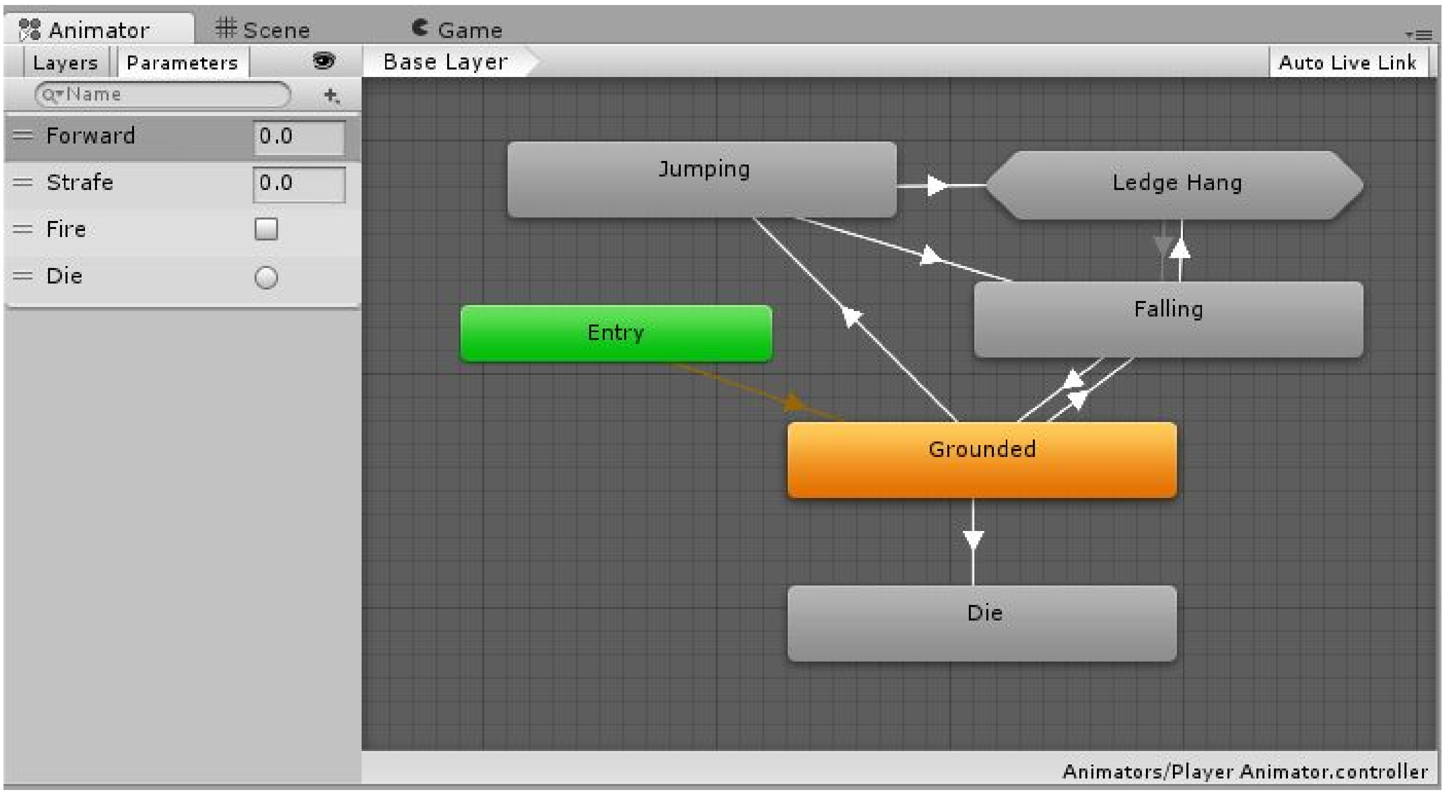

Animation
Preview
148
Robot Arm Animation
Samples 10
0:00
5:00
10:00
prop_robotArm : Rotation
prop_robotArm_body : Rotation
prop_robotArm_arm : Rotation
prop_robotArm_hand : Rotation
prop_robotArm_clawLow_base : Rotation
prop_robotArm_clawLow_mid : Rotation
prop_robotArm_clawLow_tip : Rotation
prop_robotArm_clawTop_base : Rotation
prop_robotArm_clawTop_mid : Rotation
prop_robotArm_clawTop_tip : Rotation
Add Property
Dopesheet
Curves
Inspector
Animation Event

Function
Float 0
Int 0
String
Object None (Object)

Animator Controller

An Animator Controller allows you to arrange and maintain a set of animations for a character or other animated Game Object. The controller has references to the animation clips used within it, and manages the various animation states and the transitions between them using a so-called State Machine, which could be thought of as a kind of flow-chart, or a simple program written in a visual programming language within Unity.

When you create a new Animation Clip, an Animator Controller is automatically generated. It can be created manually from the Assets menu, or from the Create menu in the Project window.



The Animator window has two main sections: the main gridded layout area, and the left-hand Layers & Parameters pane. The main section with the dark grey grid is the layout area. You can use this area to create, arrange and connect states in your Animator Controller.

- You can right-click on the grid to create a new state nodes or Transitions between States
- Use the middle mouse button or Alt/Option drag to pan the view around
- Click to select state nodes to edit them, and click & drag state nodes to rearrange the layout of your state machine

The left-hand pane can be switched between Parameters view and Layers view. The parameters view allows you to create, view and edit the Animator Controller Parameters. These are variables you define that act as inputs into the state machine.

- To add a parameter, click the Plus icon and select the parameter type from the pop up menu.
- To delete a parameter, select the parameter in the lists and press the delete key
  (on macOS use fn-Delete to delete the selected parameter)

When the left-hand pane is switched to Layers view, you can create, view and edit layers within your Animator Controller. This allows you to have multiple layers of animation within a single animation controller working at the same time, each controlled by a separate state machine. A common use of this is to have a separate layer playing upper-body animations over a base layer that controls the general movement animations for a character.

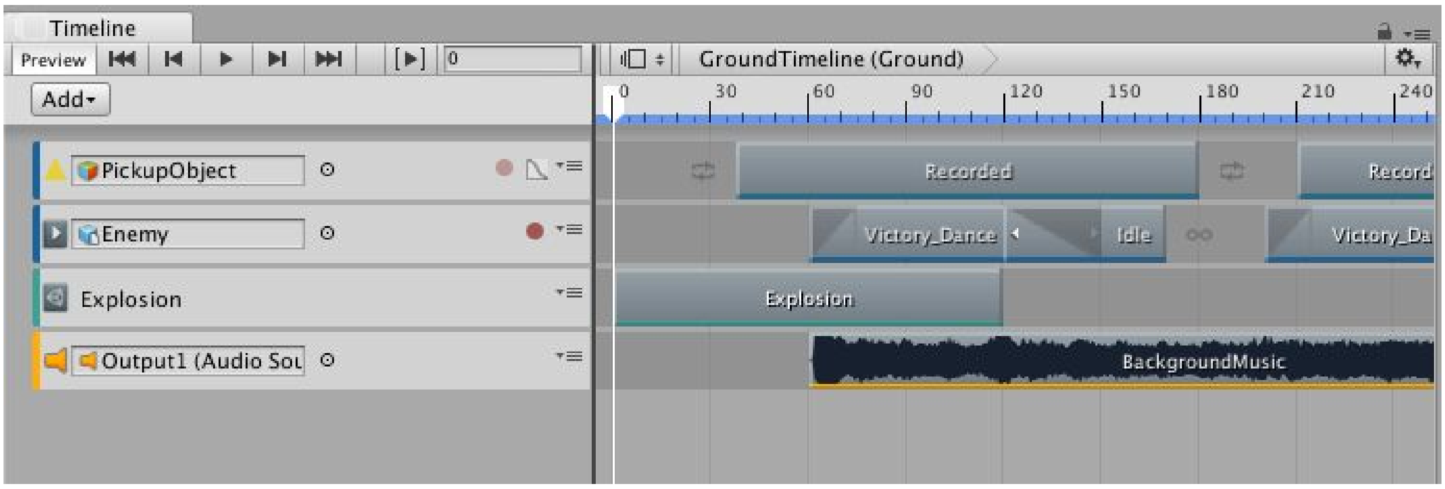
- To add a layer, click the plus icon
- To delete a layer, select the layer and press the delete key
  (on macOS use fn-Delete to delete the selected parameter)

To get detailed descriptions on how to use the Animation State machine, go to:
https://docs.unity3d.com/Manual/AnimationStateMachines.html

Timeline

In difference to Animations, that mot of the time affect single GameObject, the Timeline window allows you to create and direct cinematic content, game-play sequences, audio sequences and complex particle effects. You can animate many different GameObjects within the same sequence, such as a cut scene or scripted sequence where a character interacts with scenery. In the timeline window you can have multiple types of track, and each track can contain multiple clips that can be moved, trimmed, and blended between. It is useful for creating more complex animated sequences that require many different GameObjects to be choreographed together.

This is done in theTimeline Editor window by visually arranging tracks and clips linked to GameObjects in your scene.



Creating a Timeline Asset and Timeline instance

To create a new Timeline Asset and Timeline instance, follow these steps:

1. In your scene, select the GameObject that you want to use as the focus of your cinematic or other gameplay-based sequence.
2. Open the Timeline Editor window (menu: Window > Sequencing > Timeline)
3. If the GameObject does not yet have a Playable Director component attached to a Timeline Asset, a message in the Timeline Editor window prompts you to click the Create button.
4. Click Create. A dialog box prompts you for the name and location of the Timeline Asset being created.
5. Click Save.

Unity now does the following:

- Saves a new Timeline Asset to the Project. If you did not change the name and location of the Timeline Asset being created, the name of the Timeline Asset is based on the selected GameObject with the “Timeline” suffix. For example, selecting the GameObject named “Enemy”, by default, names the asset “EnemyTimeline” and saves it to the Assets directory in your project.

- Adds an empty Animation track to the Timeline Asset.

- Adds a Playable Director component to the selected GameObject and sets the Playable property to the Timeline Asset. This creates a Timeline instance.

- In the Playable Director component, the binding for the Animation track is set to the selected GameObject. The Animation track does not have any clips, so the selected GameObject is not animated.

- Adds an Animator component to the selected GameObject. The Animator component animates the GameObject through the Timeline instance. The GameObject cannot be animated without an Animator component.

Recording basic animation with an Infinite clip

You can record animation directly to an Animation track. When you record directly to an empty Animation track, you create an Infinite clip. An Infinite clip is defined as a clip that contains basic key animation recorded through the Timeline Editor window. An Infinite clip cannot be positioned, trimmed, or split because it does not have a defined size: it spans the entirety of an Animation track.

Before creating an Infinite clip, you must add an empty Animation track for the GameObject that you want to animate.

In the Track list,click the Record button for the empty Animation track to enable Record mode. The Record button is available for Animation tracks bound to simple GameObjects such as cubes, spheres, lights, and so on.

When a track is in Record mode, the clip area of the track is drawn in red with the “Recording…” message. Any modification to an animatable property of the GameObject sets a key at the location of the Timeline Playhead. Animatable properties include transforms and the animatable properties for all components added to the GameObject.

Track List

Use the Track List to add, select, duplicate, delete, lock, mute, and reorder the tracks that comprise a Timeline Asset.
You can also organize tracks into Track groups.

The simplest method of adding a track is to click the Add button and select the type of track from the Add Track menu. You can also Right-click in an empty area of the Track list and select the type of track from the Add Track menu.

The Timeline Editor window also supports dragging a GameObject into the Track list. Drag a GameObject into an empty area in the Track list and select the type of track to add from the context menu.

Depending on the type of track selected, the Timeline Editor window performs different actions:

Select **Animation Track** and the Timeline Editor binds the GameObject to the Animation track. If the GameObject doesn’t already have an Animator component, the Timeline Editor creates an Animator component for the GameObject.

Select **Activation Track** and the Timeline Editor binds the GameObject to the Activation track. There are some limitations when creating an Activation track when dragging a GameObject.

Select **Audio Track** and the Timeline Editor adds an Audio Source component to the GameObject and binds this Audio Source component to the Audio Track.

In detail Informations about how to add, select, duplicate, delete, lock, mute, and reorder the tracks can be found here:
https://docs.unity3d.com/Manual/TimelineTrackList.html

Clips view

Navigating the Clips view - Use one of the following methods to pan, zoom, or frame clips in the Clips view:

- To pan, Middle-drag, or hold Alt and drag.
- To zoom vertically, move the scroll-wheel, or hold Alt and right-drag.
- To zoom horizontally, hold Command/Control and zoom vertically.
- To frame all selected clips, select one or multiple clips and press F.
- To frame all clips vertically, press A.

Adding clips

The Timeline Editor window supports different methods of adding clips to tracks depending on the type of track.The quickest method is to Right-click on an empty area within a track and select the appropriate Add option from the context menu. Depending on the track, the options for adding clips change. The clip is added after the last clip on the track.

You can also drag an Animation clip to an empty area in the Timeline Editor window to automatically create a track and add the Animation clip to the track.

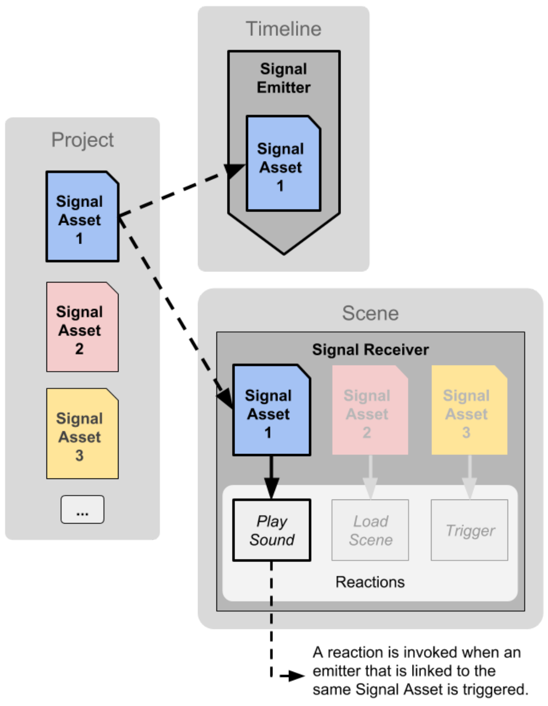
In detail Informations about how to add, position, and manipulate clips for each track in the Track list can be found here:
https://docs.unity3d.com/Manual/TimelineTrackList.html

Timeline Signals

Timeline Signals are an easy way for Timeline to interact with objects in the scene.Signal emitters are used to trigger a change in state of the scene when the timeline passes a given point in time. Using a signal emitter and a signal asset, you can trigger a signal receiver in a game object that will define a set of pre-configured reactions to your Timeline.

A Signal Emitter can be added on the timeline and on tracks (with some restrictions).

- On the timeline: This is a new concept for this release. Signals can be added on the timeline itself. No tracks are needed in this case.
- On a track: Signals can also be added to any track, as long as the track accepts a binding. We've also introduced a new type of track, a Signal Track. This track accepts only signals but no clips.



In order to properly set up signals on a timeline, you need three pieces: a Signal Asset, a Signal Emitter and a Signal Receiver:

- Signal Asset: This asset is the link between an emitter and a receiver. Basically, the Signal Asset acts as an identifier.
- Signal Emitter: A Signal Emitter is placed on the Timeline. It contains a reference to a Signal Asset. When played, if the current time is greater than the emitter's time, the emitter is triggered; it will send its Signal Asset to a Signal Receiver.
- Signal Receiver: The Signal Receiver is a component that has a list of reactions, each of those reactions are bound to a Signal Asset. When a receiver is notified that a signal has been triggered, it will invoke the reaction bound to the corresponding Signal Asset.

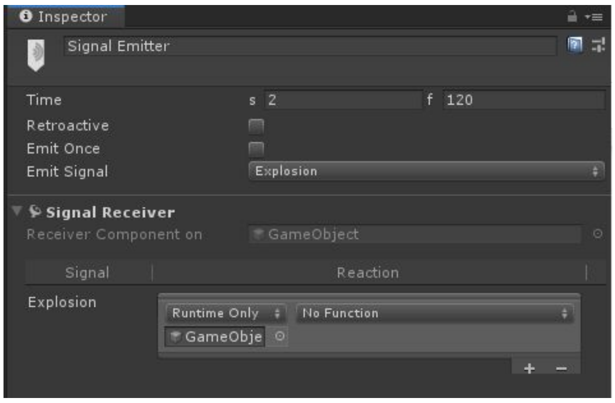
How to setup a Signal Track:

- Right-click on the Tracklist of your Timeline, then choose Signal Track
- Add a GameObject (like your "SceneDirector") to the empty GameObject slot on the new track
- Choose "CreateSignalReceiver". This will add a Signal Receiver Component to your GameObject
- Right-click on the new track, then choose Add Signal Emitter
- In the Inspector window, click on Create Signal Asset, choose a file name and hit Enter
- Still in the Inspector, click the Create Reaction button.
- Now you can setup what is necessary for the reaction associated with the signal you just created

The important part is that the Signal Emitter inspector first shows you properties related to the Signal Emitter:

- The Retroactive property, when enabled, will trigger the signal if the timeline begins playing after the signal's time.
- The Emit Once property, when enabled, will trigger the signal only once when the timeline loops.

It also displays, for your convenience, the Signal Receiver that will receive the signals.

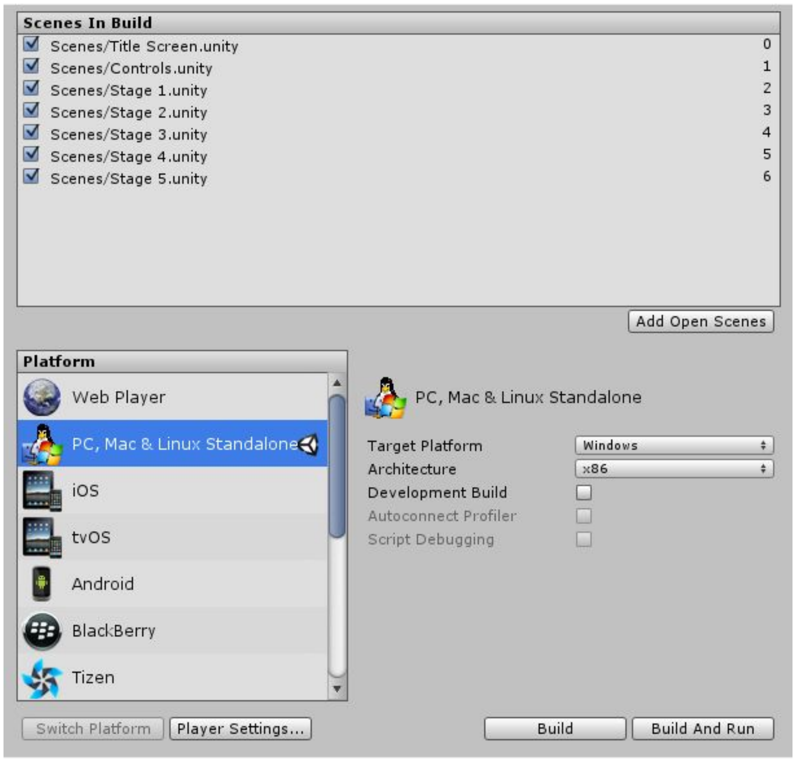

Inspector
Signal Emitter
Time s 2 f 120
Retroactive
Emit Once
Emit Signal Explosion
Signal Receiver
Receiver Component on GameObject
Signal | Reaction
Explosion | Runtime Only | No Function
GameObje

Building an Application

At any time while you are creating your game, you might want to see how it looks when you build and run it outside of the editor as a standalone. This section will explain how to access the Build Settings and how to create different builds of your games.

File > Build Settings… is the menu item to access the Build Settings window. It pops up an editable list of the scenes that will be included when you build your game.

The Build Settings window



Scenes In Build

✓	Scenes/Title Screen.unity	0
✓	Scenes/Controls.unity	1
✓	Scenes/Stage 1.unity	2
✓	Scenes/Stage 2.unity	3
✓	Scenes/Stage 3.unity	4
✓	Scenes/Stage 4.unity	5
✓	Scenes/Stage 5.unity	6

PC, Mac & Linux Standalone

Target Platform	Windows
Architecture	x86
Development Build	☐
Autoconnect Profiler	☐
Script Debugging	☐

The first time you view this window in a project, it will appear blank. If you build your game while this list is blank, only the currently open scene will be included in the build. If you want to quickly build a test player with only one scene file, just build a player with a blank scene list.

It is easy to add scene files to the list for multi-scene builds. There are two ways to add them. The first way is to click the Add Open Scenes button. You will see the currently open scenes appear in the list. The second way to add scene files is to drag them from the Project View to the list.

At this point, notice that each of your scenes has a different index value. Scene 0 is the first scene that will be loaded when you build the game. When you want to load a new scene, use SceneManager.LoadScene inside your scripts.

If you've added more than one scene file and want to rearrange them, simply click and drag the scenes on the list above or below others until you have them in the desired order.

When you are ready to publish your build, select a Platform and make sure that the Unity logo is next to the platform. You will be able to select a name and location for the game using a standard Save dialog. When you click Save, Unity builds your game pronto. It's that simple. If you are unsure where to save your built game to, consider saving it into the projects root folder. You cannot save the build into the Assets folder.

Loading scenes

When you work with different scenes you need a script to access them. A simple script is available in the Workshops Library folder: LoadNextSceneOn.cs

It will load the next scene from the "Scenes in Build"-Window when pressing a specified Key.

Player Settings

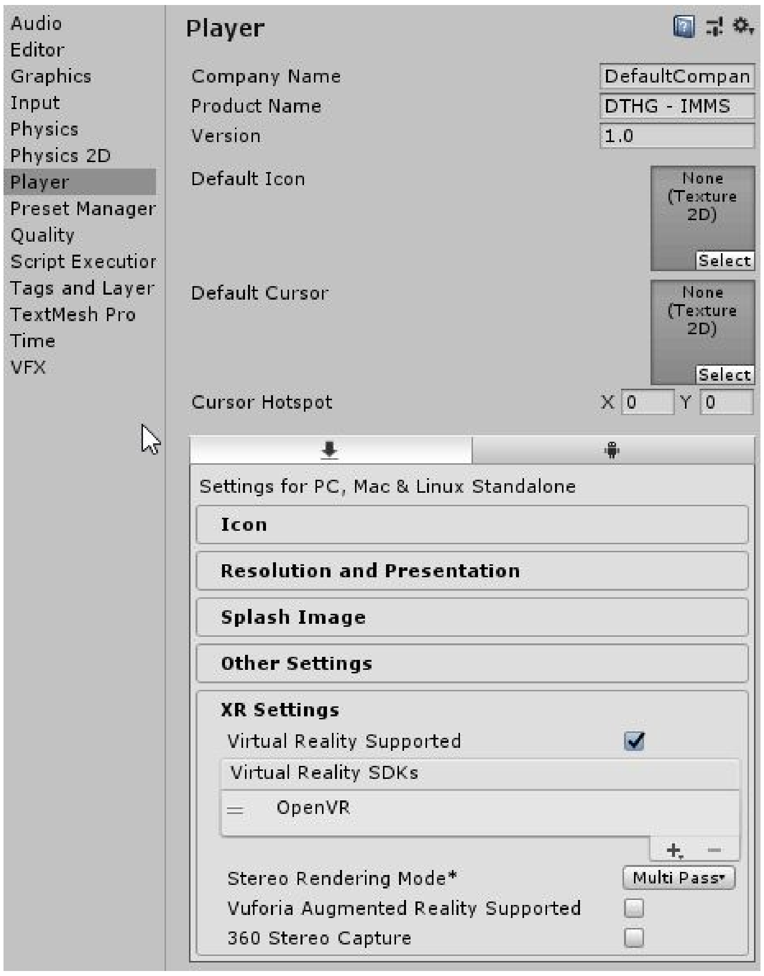
Use the Player settings to set various options for the final game built by Unity.
There are a few general settings that are the same regardless of the build target.
The edit the Player Settings, go to: **Edit > Project Settings > Player**

General settings

Property	Function
Company Name	Enter the name of your company. This is used to locate the preferences file.
Product Name	Enter the name that appears on the menu bar when your game is running. Unity also uses this to locate the preferences file.
Version	Enter the version number of your application.
Default Icon	Pick the the Texture 2D file that you want to use as a default icon for the application on every platform. You can override this for specific platforms.
Default Cursor	Pick the the Texture 2D file that you want to use as a default cursor for the application on every supported platform.
Cursor Hotspot	Set the offset value (in pixels) from the top left of the default cursor to the location of the cursor hotspot.

Platform Specific Settings

- Icon: the game icon(s) as shown on the desktop.
- Resolution and Presentation: settings for screen resolution and other presentation details such as whether the game should default to fullscreen mode.
- Splash Image: the image shown while the game is launching. This section also includes common settings for creating a Splash Screen, which are documented in the Splash Screen section.
- Other Settings: any remaining settings specific to the platform.
- Publishing Settings: details of how the built application is prepared for delivery from the app store or host webpage.
- XR - Settings: settings specific to Virtual Reality, Augmented Reality, and Mixed Reality applications.



Player

Company Name: DefaultCompan
Product Name: DTHG - IMMS
Version: 1.0
Default Icon: None (Texture 2D) Select
Default Cursor: None (Texture 2D) Select
Cursor Hotspot: X 0 Y 0

Settings for PC, Mac & Linux Standalone

Icon
Resolution and Presentation
Splash Image
Other Settings
XR Settings
Virtual Reality Supported: ✓
Virtual Reality SDKs
OpenVR
Stereo Rendering Mode*: Multi Pass
Vuforia Augmented Reality Supported: ☐
360 Stereo Capture: ☐

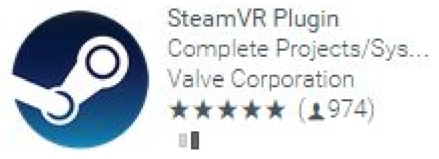
Quality Settings

Make sure to set the correct quality levels as default
in the Quality Settings.

For more information on the Quality Settings, go to:
https://docs.unity3d.com/Manual/class-QualitySettings.html

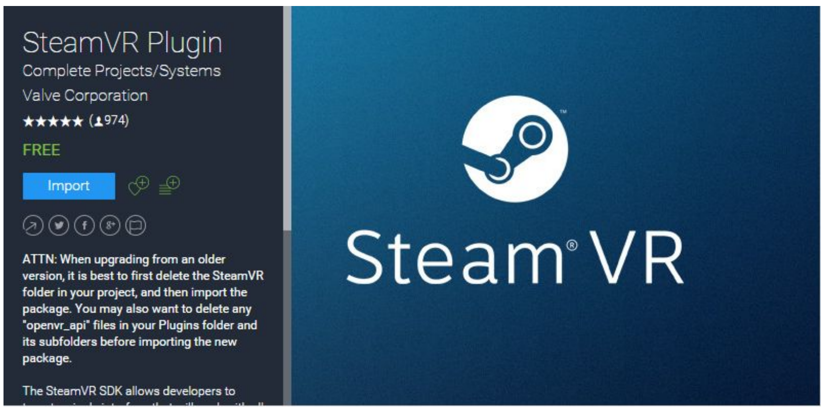
Setting up a Project for SteamVR (HTC Vive, Oculus Rift, WMR…)

- Edit > Project Settings > Player > XR Settings
  - Virtual Reality Supported [✓]
  - Virtual Reality SDKs remove Oculus
    (unless you want to specifically want to create for the Oculus Go or other Oculus VR Systems)
  - Keep “OpenVR” in the SDKs to create a VR Application that will run on several systems (Oculus, Vive, Windows Mixed Reality)



SteamVR Plugin
Complete Projects/Sys...
Valve Corporation
★★★★★ (974)

- File > Build Settings >
  - Target Platform: Windows
  - Architecture: x86_64
- add SteamVR Plugin (Asset Store)
- Open the »SteamVR Input« window
  Window > Steam VR Input
- When asked about using the example files: choose [YES]
- For now just click on [Save and generate]
- Wait for the compilation process to finish



SteamVR Plugin
Complete Projects/Systems
Valve Corporation
★★★★★ (974)
FREE

ATTN: When upgrading from an older version, it is best to first delete the SteamVR folder in your project, and then import the package. You may also want to delete any "openvr_api" files in your Plugins folder and its subfolders before importing the new package.

The SteamVR SDK allows developers to

Setting up a Scene for SteamVR

- Delete ‘Main Camera’
- Add Player Prefab (SteamVR > InteractionSystem > Core > Prefabs)
- A VR Scene setup with the Player can be tested without VR Equipment!

VR Locomotion

VR locomotion are methods that enable movement from one place to another (locomotion) within a virtual reality environment. Locomotion through a virtual environment is enabled by a variety of methods including head bobbing and arm swinging, as well as other natural movements that translate to in-game movements.

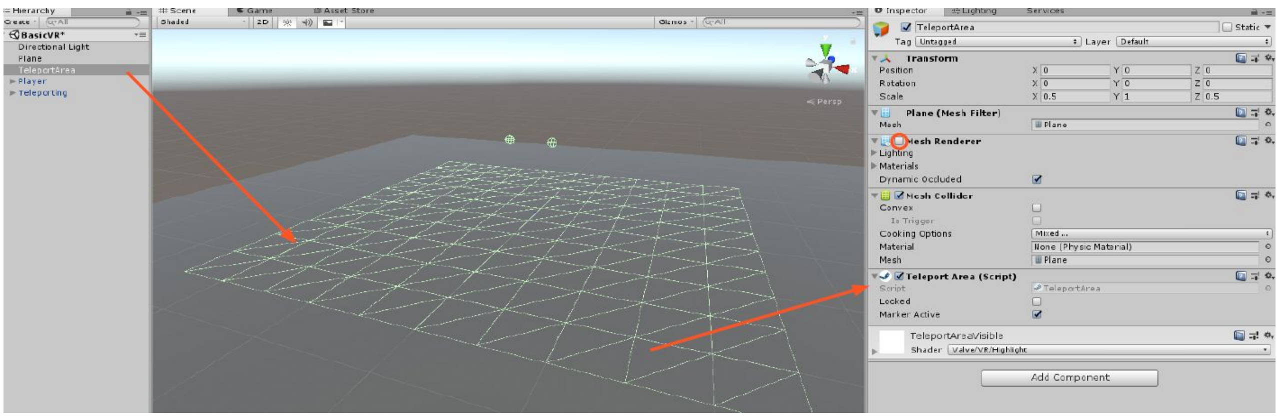
A few examples of VR locomotion:

Artificial locomotion involves the use of controllers to navigate through an environment. One problem with that method is that it tends to cause VR sickness by creating a discrepancy between what the user detects through vision and what the movement-related systems within the inner ear detect.

For teleportation, another method of VR locomotion, the user might point to their desired destination and click a button to automatically move there. In room-scale VR, for example, the user might come to the physical limits of the room and then choose to teleport to a different virtual location.

Locomotion - Teleporting

- import the Teleporting Prefab to your scene: SteamVR > InteractionSystem > Teleport > Prefabs
- add a plane to the scene and call for example »Teleport Area«
- add the Teleport Area script to this plane
- If you want the plane to be invisible deactivate the Mesh Renderer (otherwise it will be visible as blue square)



TeleportArea — Static
Tag: Untagged — Layer: Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 0.5 Y 1 Z 0.5

Plane (Mesh Filter)
Mesh: Plane

☐ Mesh Renderer
Lighting
Materials
Dynamic Occluded ✓

✓ Mesh Collider
Convex ☐
Is Trigger ☐
Cooking Options: Mixed ...
Material: None (Physic Material)
Mesh: Plane

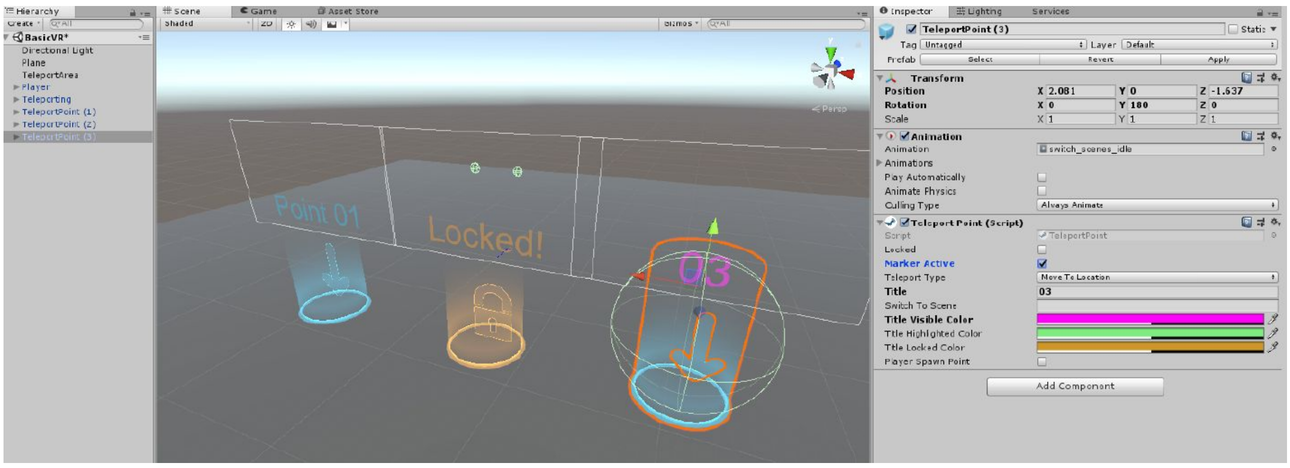
✓ Teleport Area (Script)
Script: TeleportArea
Locked ☐
Marker Active ✓

TeleportAreaVisible
Shader: Valve/VR/Highlight

Locomotion - Teleporting - Fixed Points

- import the Teleporting Prefab to your scene: SteamVR > InteractionSystem > Teleport > Prefabs
- add a one or more Teleport Point Prefabs: SteamVR > InteractionSystem > Teleport > Prefabs

  A player can only jump to unlocked Points or Areas



TeleportPoint (3) — Static
Tag: Untagged — Layer: Default
Prefab: Select | Revert | Apply

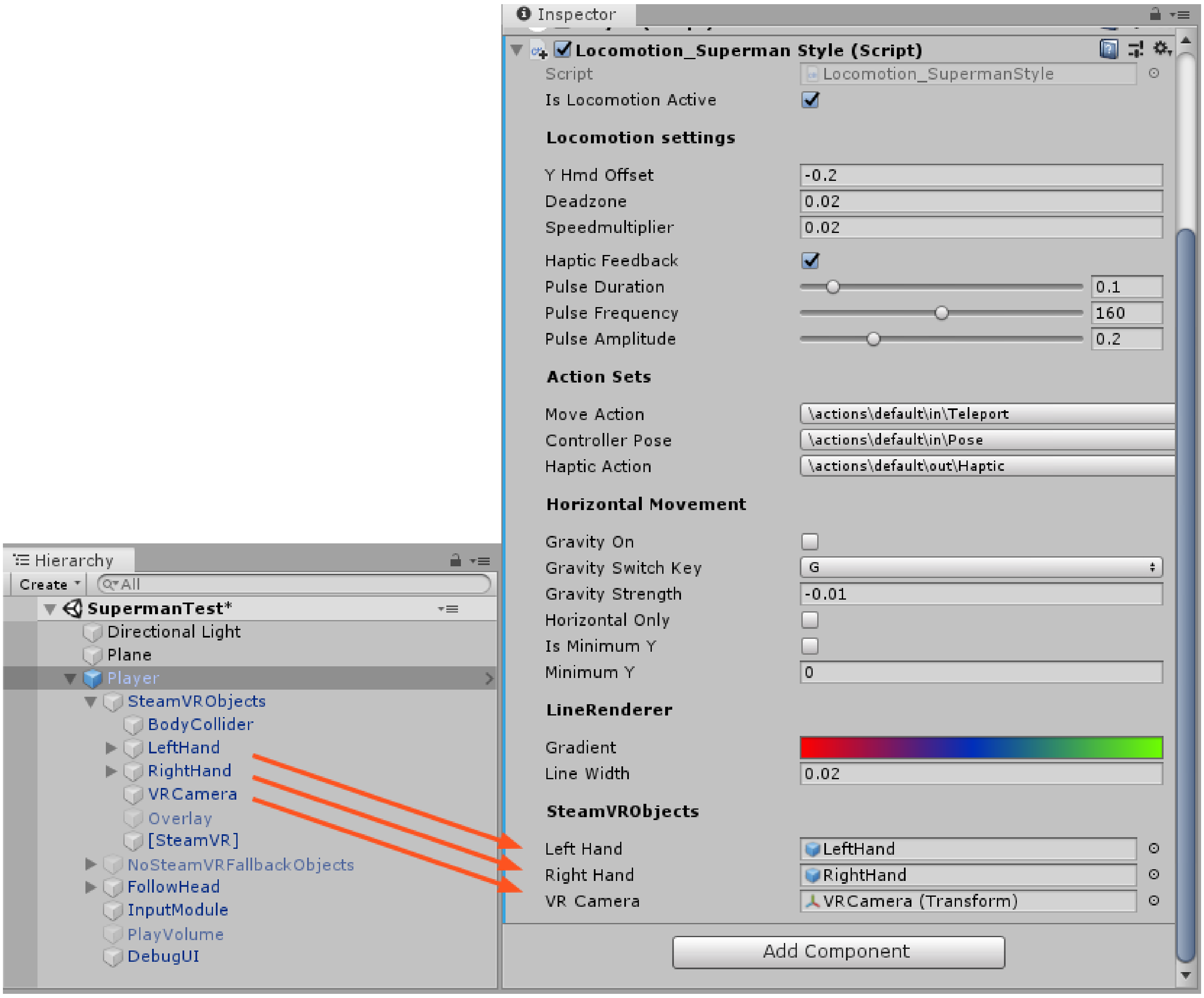
Transform
Position X 2.081 Y 0 Z -1.637
Rotation X 0 Y 180 Z 0
Scale X 1 Y 1 Z 1

✓ Animation
Animation: switch_scenes_idle
Animators
Play Automatically ☐
Animate Physics ☐
Culling Type: Always Animate

✓ Teleport Point (Script)
Script: TeleportPoint
Locked ☐
Marker Active ✓
Teleport Type: Move To Location
Title: 03
Switch To Scene:
Title Visible Color
Title Highlighted Color
Title Locked Color
Player Spawn Point ☐

Locomotion - Superman Style

- Import the Locomotion_SupermanStyle.cs script from the workshops library
- Add the script to the Player
- Populate the script's empty slots for SteamVRObjects (LeftHand, RightHand VRCamera)
- Setup the gradient for the LineRenderer



Locomotion_Superman Style (Script)

Script	Locomotion_SupermanStyle
Is Locomotion Active	✓

Locomotion settings

Y Hmd Offset	-0.2
Deadzone	0.02
Speedmultiplier	0.02
Haptic Feedback	✓
Pulse Duration	0.1
Pulse Frequency	160
Pulse Amplitude	0.2

Action Sets

Move Action	\actions\default\in\Teleport
Controller Pose	\actions\default\in\Pose
Haptic Action	\actions\default\out\Haptic

Horizontal Movement

Gravity On	
Gravity Switch Key	G
Gravity Strength	-0.01
Horizontal Only	
Is Minimum Y	
Minimum Y	0

LineRenderer

Gradient	
Line Width	0.02

SteamVRObjects

Left Hand	LeftHand
Right Hand	RightHand
VR Camera	VRCamera (Transform)

```
SupermanTest*
  Directional Light
  Plane
  Player
    SteamVRObjects
      BodyCollider
      LeftHand
      RightHand
      VRCamera
      Overlay
      [SteamVR]
    NoSteamVRFallbackObjects
    FollowHead
    InputModule
    PlayVolume
    DebugUI
```

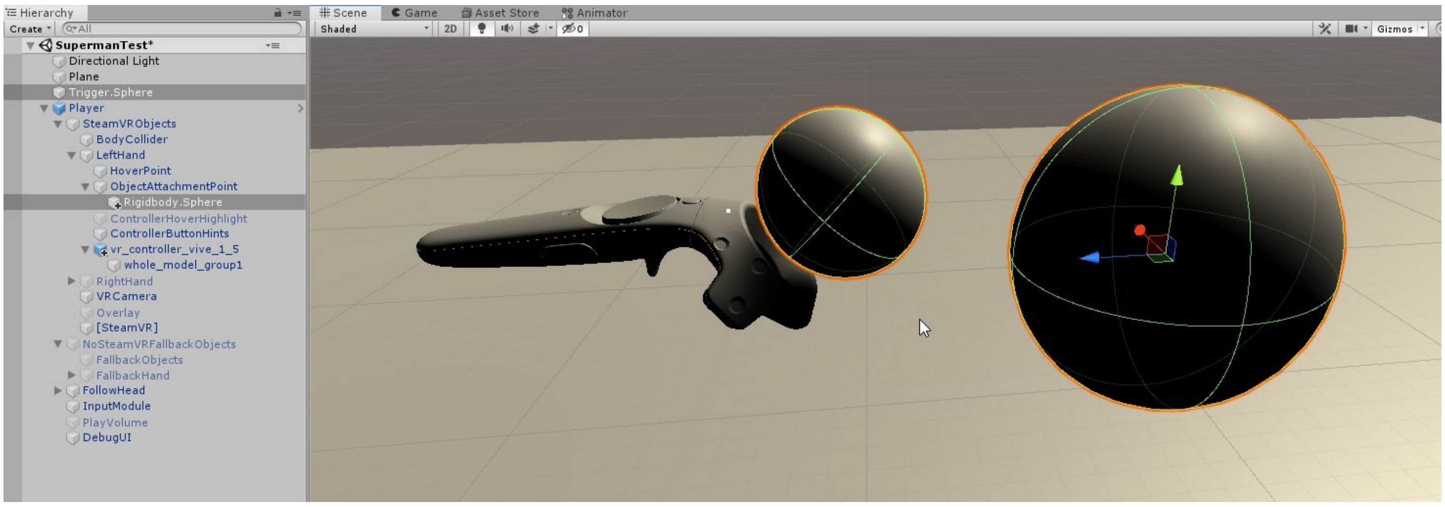
Locomotion - Dash Movement

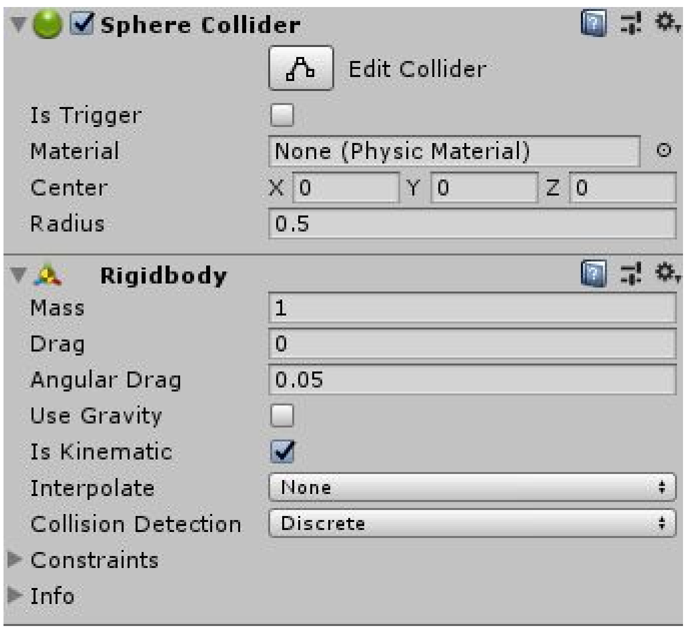
A good Tutorial about locomotion with Dash Movement can be found here:
https://unity3d.college/2017/07/25/vr-dash-movement-locomotion-steamvr-unity3d/

Interaction

Simple interaction using Colliders

A simple way to interact with your environment is to use Collision Objects, that activate an Event when triggered.
To trigger those Collision Objects you have to attach a Rigidbody (kinematic) to your controller.





Sphere Collider

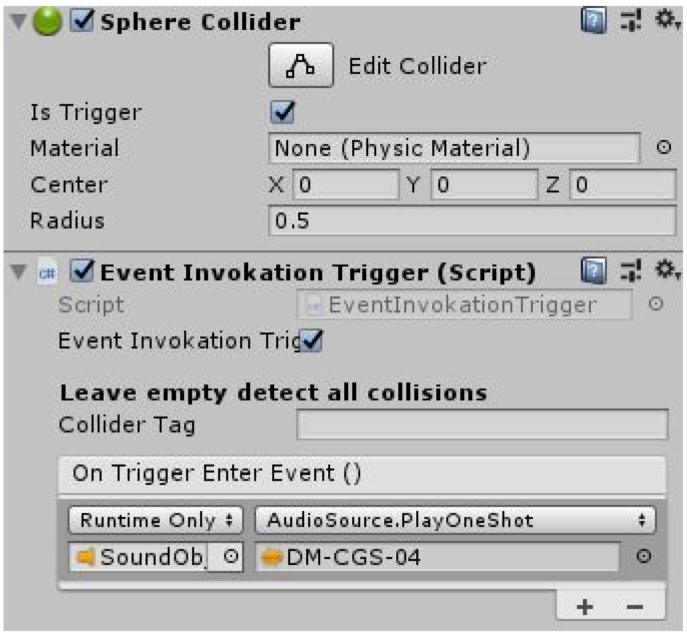
Edit Collider

Is Trigger	☐
Material	None (Physic Material)
Center	X 0 Y 0 Z 0
Radius	0.5

Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	☐
Is Kinematic	✓
Interpolate	None
Collision Detection	Discrete

Constraints
Info



Sphere Collider

Edit Collider

Is Trigger	✓
Material	None (Physic Material)
Center	X 0 Y 0 Z 0
Radius	0.5

Event Invokation Trigger (Script)

Script	EventInvokationTrigger
Event Invokation Trig	✓

Leave empty detect all collisions

Collider Tag	

On Trigger Enter Event ()

Runtime Only	AudioSource.PlayOneShot
SoundOb	DM-CGS-04

Rigidbody.Sphere on controller

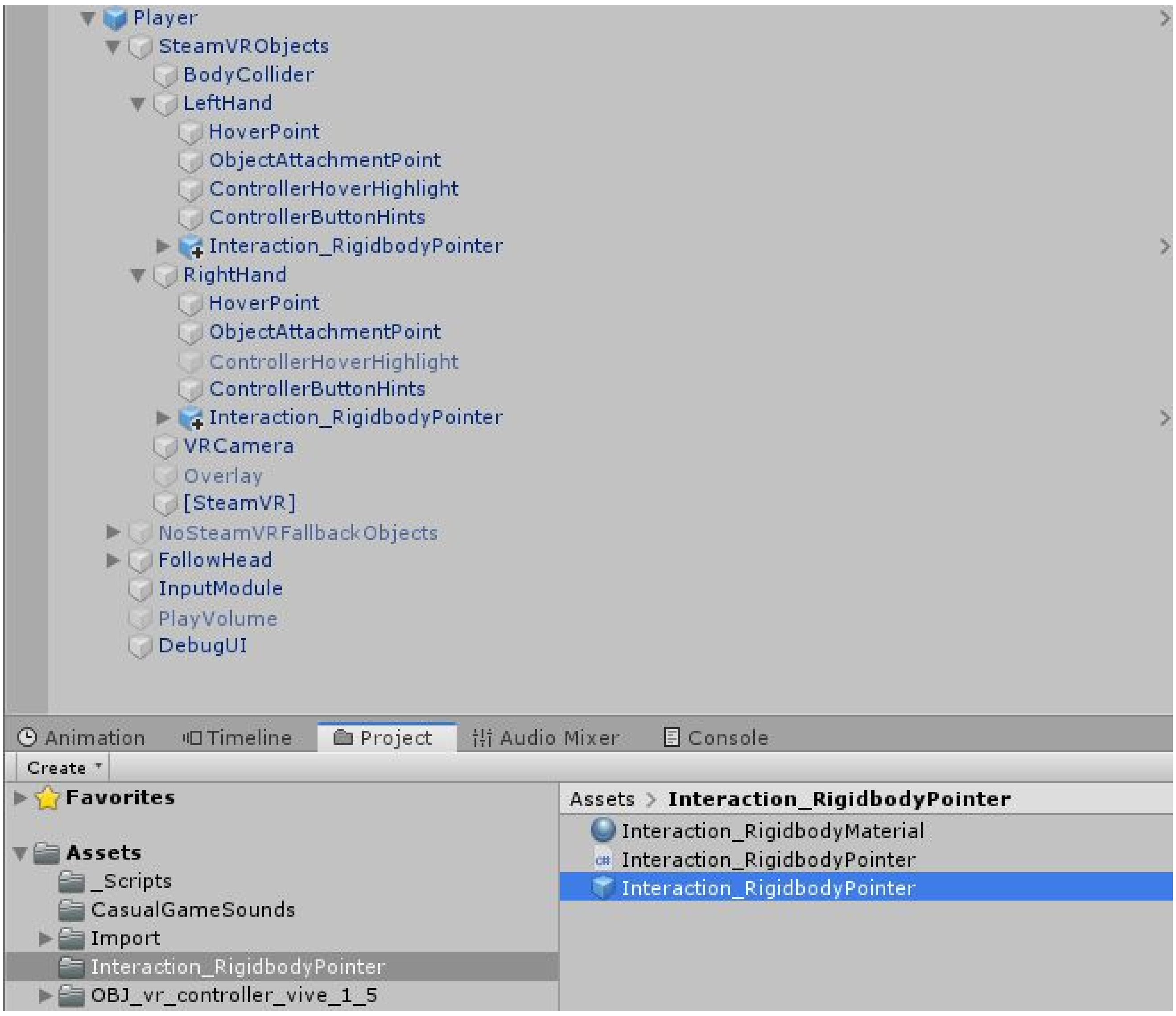
Trigger.Sphere with Event Invokation on Trigger Script

Simple interaction using a Laser Pointer

I prepared a simple Laser Pointer Prefab that you can use in your project and adapt to your liking.
It is based on the “SteamVR Canvas Pointer” tutorial from “VR with Andrew”, but adapted in a way so it can trigger a Collision Object instead of a UI Canvas. If you want to create a graphical User Interface, have a look at his videos:
https://www.youtube.com/watch?v=3mRI1hu9Y3w

To add a Laser Pointer to your SteamVR Player follow these steps:

- Import the Interaction_RigidbodyPointer.unitypackage from the workshops material folder
- In the hierarchy window, navigate to your Player’s Hands (Player > SteamVR Objects > Left Hand / Right Hand)
- Make the Interaction_RigidbodyPointer Prefab a child of one or both of the hands



```
Player
  SteamVRObjects
    BodyCollider
    LeftHand
      HoverPoint
      ObjectAttachmentPoint
      ControllerHoverHighlight
      ControllerButtonHints
      Interaction_RigidbodyPointer
    RightHand
      HoverPoint
      ObjectAttachmentPoint
      ControllerHoverHighlight
      ControllerButtonHints
      Interaction_RigidbodyPointer
    VRCamera
    Overlay
    [SteamVR]
  NoSteamVRFallbackObjects
  FollowHead
  InputModule
  PlayVolume
  DebugUI
```

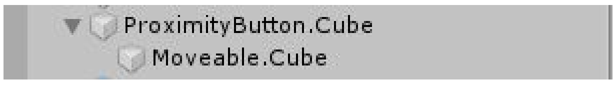
```
Favorites
Assets
  _Scripts
  CasualGameSounds
  Import
  Interaction_RigidbodyPointer
  OBJ_vr_controller_vive_1_5
```

```
Assets > Interaction_RigidbodyPointer
  Interaction_RigidbodyMaterial
  Interaction_RigidbodyPointer
  Interaction_RigidbodyPointer
```

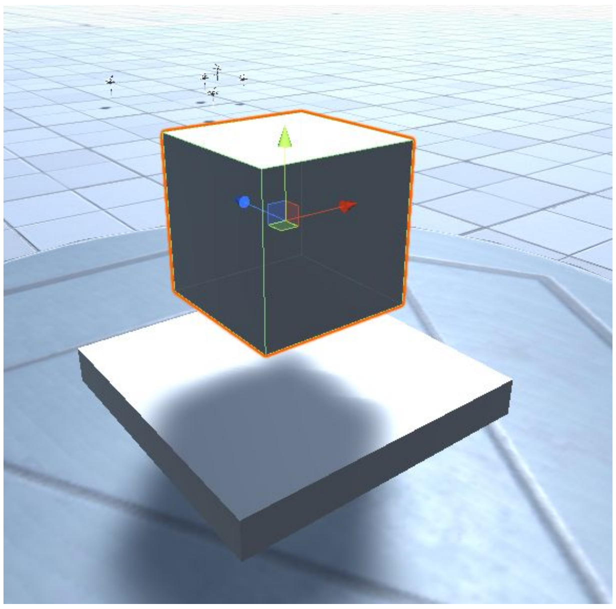
SteamVR Interactables - ProximityButton

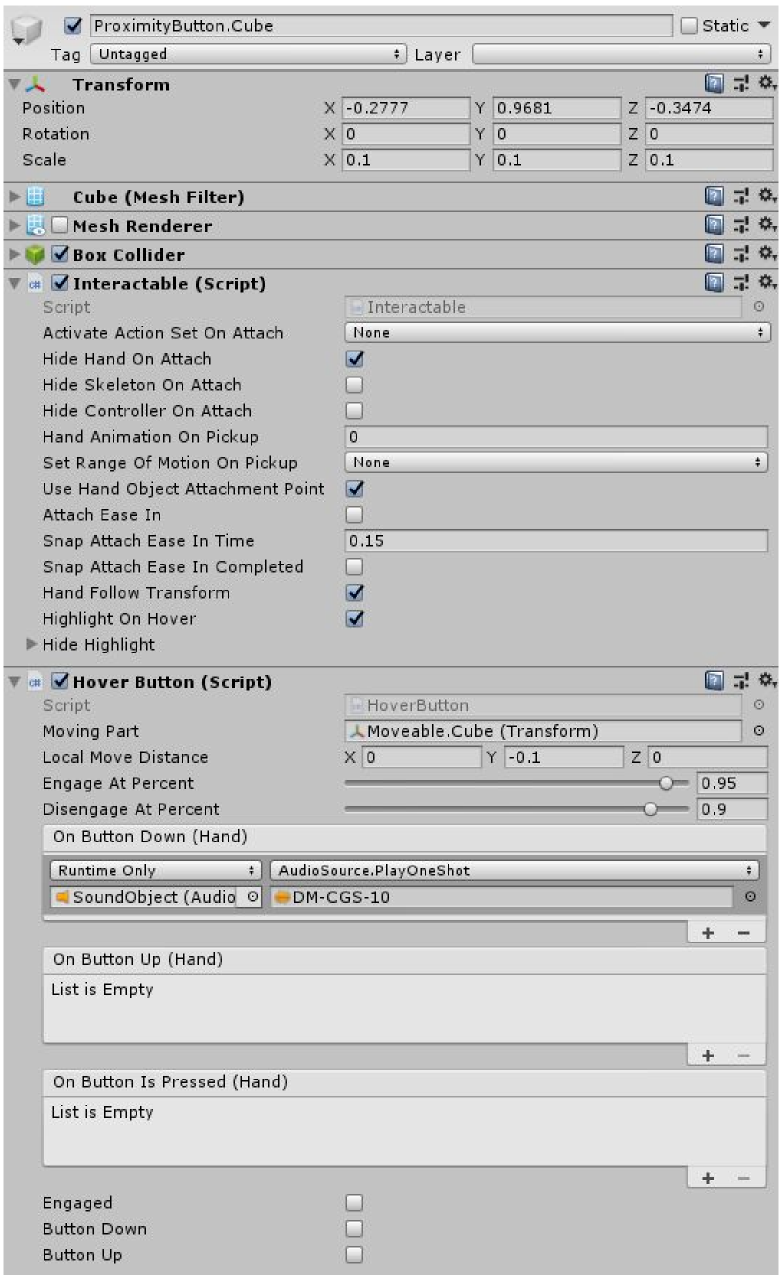
To make a simple Proximity Button:

- Create a Cube and duplicate it. Rename the Copy “MovableCube” and make it the Child of the original cube
- **Deactivate the original’s Mesh Renderer!**
- Add the Hover Button script to the original cube (Add Component > Hover Button)
    - The Interactable script will be added automatically
- Populate the “Moving Part” slot with the “MovableCube”
- Add an Event to OnButtonDown



ProximityButton.Cube
Moveable.Cube





ProximityButton.Cube ☐ Static
Tag Untagged Layer

Transform

	X	Y	Z
Position	-0.2777	0.9681	-0.3474
Rotation	0	0	0
Scale	0.1	0.1	0.1

Cube (Mesh Filter)
☐ **Mesh Renderer**
✓ **Box Collider**
✓ **Interactable (Script)**

Script	Interactable
Activate Action Set On Attach	None
Hide Hand On Attach	✓
Hide Skeleton On Attach	☐
Hide Controller On Attach	☐
Hand Animation On Pickup	0
Set Range Of Motion On Pickup	None
Use Hand Object Attachment Point	✓
Attach Ease In	☐
Snap Attach Ease In Time	0.15
Snap Attach Ease In Completed	☐
Hand Follow Transform	✓
Highlight On Hover	✓
Hide Highlight	

✓ **Hover Button (Script)**

Script	HoverButton
Moving Part	Moveable.Cube (Transform)
Local Move Distance	X 0 Y -0.1 Z 0
Engage At Percent	0.95
Disengage At Percent	0.9

On Button Down (Hand)
Runtime Only | AudioSource.PlayOneShot
SoundObject (Audio | DM-CGS-10

On Button Up (Hand)
List is Empty

On Button Is Pressed (Hand)
List is Empty

Engaged	☐
Button Down	☐
Button Up	☐

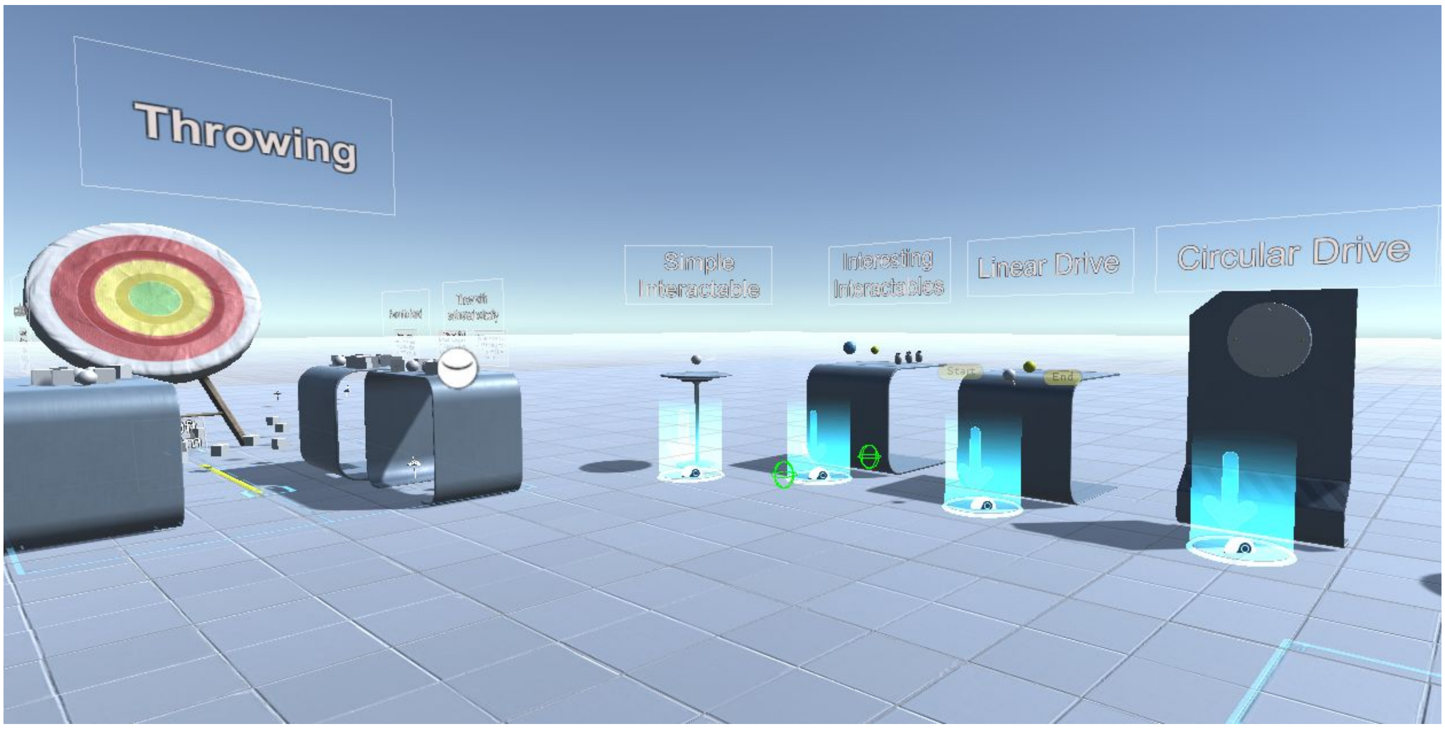
SteamVR Interactables - Throwable (& Movable)

Another way to interact with objects using the SteamVR Interactables is by attaching the “Throwable” script to an object.
By doing so you can move and throw objects with the controllers

- Create a Cube, reduce its scale to 0.2/0.2/0.2
- Add the “Throwable” script to it (Add Component > Throwable)
    - The Interactable script will be added automatically
- Basically you are done - you can now pick up the object and throw it around
- But you can also play around with the object’s Rigidbody Settings:
    - Deactivating it’s gravity setting will make it float in the air as if it is weightless
    - Making it Kinematic lets you move it from one place to another, where it will stay put even in mid air.

SteamVR Interactables - Example Scene

Many more examples of Interaction in VR are demonstrated in the “Interactions_Example” Scene. You can find the main scene in your project under **Assets/SteamVR/InteractionSystem/Samples/Scenes/Interactions_Example.unity**



Here you can test a variety of interaction methods, learn how they are set up. And then copy and paste them to your project and adapt them to your needs.

Developing for the Oculus Go

Adapted from https://scriptable.com/blog/oculus-go-unity-setup-quick-start

Assumptions

- You own or have access to an Oculus Go headset
- You have played around with Virtual Reality (VR)
- You may have used Unity before
- You are using a Windows machine (explained next)
- You have enough rights on your machine to install software
- You understand that this is about the Oculus Go standalone headset and NOT the Oculus Rift

Why you need a Windows machine

I've worked on many Web projects and I prefer using my Mac. But there are several reasons why you are much better off using a Windows machine when developing apps for the Oculus Go (and preferably one with a good GPU) :

- Oculus has made it clear they don't officially support the Mac with statements in their forums
- I have also run into issues with Unity where things broke after a Mac OS update and discovered it was low on their priority list
- To get real power out of a Mac you need an external GPU

Enable Developer Mode on the Go

(Taken from the Oculus Developer documentation)

To begin development locally for Oculus Go, you must enable Developer Mode in the companion app. Before you can put your device in Developer Mode, you need to have created (or belong to) a developer organization on the Oculus Dashboard.

To join an existing organization:

1. You'll need to request access to the existing Organization from the admin.
2. You'll receive an email invite. Once accepted, you'll be a member of the Organization.

To create a new organization:

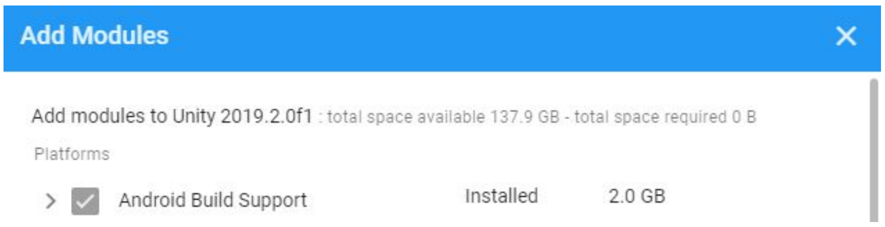
1. Go to: https://dashboard.oculus.com/organization/create
2. Fill in the appropriate information.

Enable Developer Mode:

1. Open the Oculus app on your mobile device.
2. In the Settings menu, select your Oculus Go headset that you're using for development.
3. Select More Settings.
4. Toggle Developer Mode on.

Basic Unity Setup

1. Make sure you have Android Build support and the Android SDK and NDK Tools installed.
   You can add the modules to your installation through the Unity Hub: Installs > ⁝ > Add Modules



Add Modules

Add modules to Unity 2019.2.0f1 : total space available 137.9 GB - total space required 0 B

Platforms

☑ Android Build Support	Installed	2.0 GB

2. Download and import the latest version of the Oculus Integration through the asset store

3. Open Unity and create a new project

4. Go to File > Build Settings

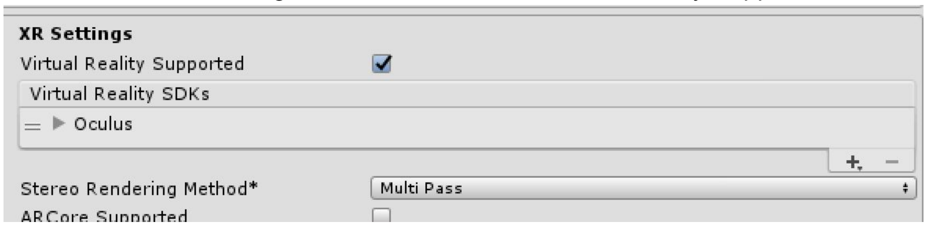
   a. Select Android and then Switch Platform. (If you did not add Android support when you first installed Unity, you will have to do so now, then restart Unity).

   b. Close the Build Settings window

5. Go to Edit > Project Settings > Player

   a. Remove “Vulkan Graphics API” from the list.

   b. Scroll down to XR Settings. Click the box next to Virtual Reality Supported.



XR Settings

Virtual Reality Supported	✓
Virtual Reality SDKs	
Oculus	
Stereo Rendering Method*	Multi Pass
ARCore Supported	☐

   Verify that Oculus appears in the SDK’s list. If not, click the “+” to the right, and select Oculus.

   c. Scroll back up to the top and set your Company Name and Product Name.

   d. Then scroll down to Other Settings . Set the Package Name field to com.[CompanyName].[ProductName]

   e. For Minimum API Level — I would set this API Level 21

Import Oculus Samples Scenes

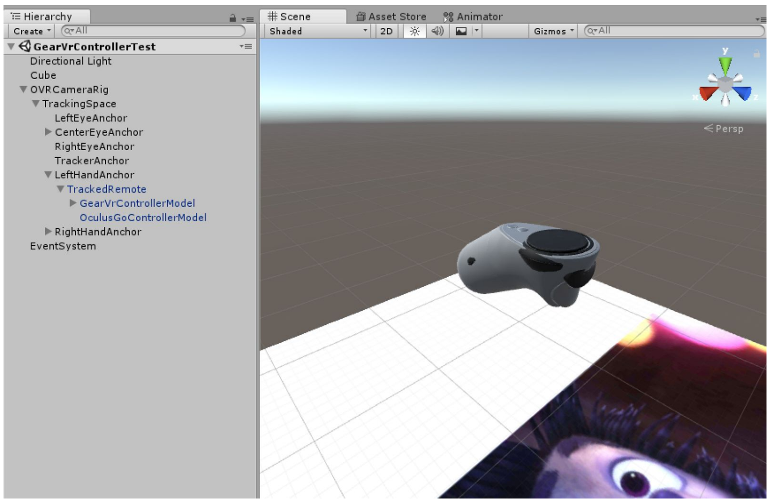
To get started with your app, we’ll import some premade sample scenes that Oculus was nice enough to make for us.

- Grab the OculusUtilities Unity package that you downloaded in step 2

- Install the package into your project

- Now, in your Assets folder, go to
  Oculus > VR > Scenes.
  Double click GearVrControllerTest.

- In the hierarchy of the scene, dive down within the OVRCameraRig object to find the OculusGoControllerModel within the scene.
  Double click it to focus on it in the scene view



```
GearVrControllerTest
  Directional Light
  Cube
  OVRCameraRig
    TrackingSpace
      LeftEyeAnchor
      CenterEyeAnchor
      RightEyeAnchor
      TrackerAnchor
      LeftHandAnchor
        TrackedRemote
          GearVrControllerModel
          OculusGoControllerModel
      RightHandAnchor
  EventSystem
```

Set up Android Debug Bridge

- We're almost there (I promise)! Before we can actually move our Unity app from our computer to our Go, we need to install Android Debug Bridge (ADB) and the drivers for the Oculus Go.
- Download and install the Oculus Go ADB driver:
    - Download the zip file containing the driver.
    - Unzip the file.
    - Right-click on the .inf file and select Install.
- Download and install ADB
    - version ADB 1.4.3. is supplied in this folder
    - or download here: https://www.xda-developers.com/install-adb-windows-macos-linux/
    - Install ADB:
        - Do you want to install ADB and Fastboot? (Y/N): Y
        - Install ADB system-wide? (Y/N): Y
        - Do you want to install device drivers? (Y/N): Y
        - A window will appear asking you to install the drivers. Do so.

Connect your Go and Build!

- Connect your go to your PC via micro-USB.
    - Bring up ADB window from the last step
    - Or start a new command-line window and go to the ADB folder
- Type and run adb devices to confirm that your Go has been detected.
- Back in Unity, go to File > Build Settings
- Click "Add Open Scenes", and ensure only the scene GearVrControllerTest has the check mark next to it checked.
- Click Build
- Select the folder where you would like to place the .APK file that's generated. I like to create a "Builds" folder in the project. Save.
- After the .APK is generated, navigate to it in File Explorer. Copy it to the same directory where you installed ADB in Step 8.
- Bring the ADB window back up.
- Type and run adb install [NameOfYourApp].apk
- You should now see the file get transferred to your Oculus Go. In a minute or two, it should then show the message Success.
- Congrats — you did it!!
- Now, to run your app, put on your Go and go to Library > Unknown Sources. You should see the Bundle ID of your app on the last page of the Unknown Apps list. Click on it to launch it, and enjoy!
- To uninstall run adb uninstall *package name*. For example, adb uninstall com.headjack.myapp.
- In order to access Oculus Go when you connected to your computer, you have to open a command-line window, go to the ADB folder and type and run and run adb devices

Best Practice Settings for Oculus Quest and Oculus Go

The following settings are based on recommendations by Oculus and may have to be adjusted according to your project.

- Use the **SetXRSettings.cs** script in your scene and set the “Texture Resolution Scale” to **1.5** – The higher render scale value, the better visual effect you get. Remember: This will tradeoff the performance.
- Set your VR camera to **Forward Rendering** and **Allow MSAA**.

- Project Settings > Player > Other Settings

Color Space:	Linear (optional, a change here while take a while)
Auto Graphics API:	OFF
Graphics API:	OpenGLES3, delete Vulkan
Multithreaded Rendering	OFF
Static Batching	ON
Dynamic Batching	ON
GPU Skinning	ON
Graphics Jobs	OFF
Scripting Backend	IL2CPP

- Project Settings > Player > XR Settings
    - Virtual Reality Supported ON (Duh)
    - Stereo Rendering Method should be set to **Single Pass (Preview)**
    - Minimum API Level 21

- Project Settings > Player > Optimization

Prebake Collision Meshes	ON
Keep Loaded Shaders Alive	ON
Optimize Mesh Data	ON

- Project Settings > Audio
    - Set DSP Buffer Size to Good latency
    - Set Spatializer Plugin to Oculus Spatializer

- Project Settings > Quality
    - To make sure mobile builds always use a consistent quality setting, uncheck all the levels, except for Medium for the android build. Medium is a good one to start from, it should be the only level that is checked.
    - Pixel light count should be 1.
    - Anisotropic Textures should be disabled
    - Anti Aliasing should generally be set to 4x Multi Sampling. If this has a large enough impact on performance, using 2x is also acceptable. The “Use Recommended MSAA Level” setting in OVRManager will override this setting when enabled.
    - Shadows should be set to Hard Shadows Only or Disable Shadows.
    - Skin Weights should not be set to more than 2 bones.
    - V Sync Count should be set to Don't Sync. The oculus runtime handles it's own V Sync.

- Project Settings > Graphics
    - Graphics on all tier levels should use Low for the standard shader quality. If possible, avoid using the standard shaders and use mobile shaders instead.
    - The rendering path for all tier levels should be set to Forward. The performance cost of deferred rendering is too high to make it a viable option on mobile.
    - Real Time GI CPU Usage should be set to Low for all tiers.
    - All the other settings should be good with their default values. Consider adding shaders to the Preloaded Shaders list located here. Adding shaders to this list will impact loading times but, will help avoid needing to load and compile these shaders on demand.

In the recent version of the Oculus Integration some important settings have to be made:

- Edit the “AndroidManifest.OVRSubmission” (Assets > Oculus > VR > Editor) and add the line:
  `<uses-feature android:name="android.hardware.vr.headtracking" android:required="true" android:version="1" />`
  before the closing </manifest>. Save and close
- In Unity’s main menu select “Oculus > Avatars > Edit Settings” and set Rift App Id and Go/Quest Id to 999999
- In Unity’s main menu select “Oculus > Tools > Build Mobile Quest-Expansion File
- In Unity’s main menu select “Oculus > Tools > Create Store-compatible AndroidManifest-xml

Official Unity Resources and help

If you need additional help try all these resources:
https://unity3d.com/learn/tutorials a set of step-based tutorials, and Topics dividing up additional lessons in more detail.
https://docs.unity3d.com/Manual/index.html The Unity User Manual helps you learn how to use the Unity Editor.
http://answers.unity3d.com/index.html The best place to ask and answer questions about development with Unity.
And never forget: google is magic! You have a specific problem? Just google: Unity + YourProblem.

Recommended Tutorials:

Andrew has many good video tutorials on: SteamVR, Oculus Go, Vuforia and Kinect
https://www.youtube.com/channel/UCG8bDPqp3jykCGbx-CiL7VQ/videos

Weekly live streams, tutorials and interviews on FusedVR:
https://www.youtube.com/channel/UCLO98KHpNx6JwsdnH04I9yQ/videos

Good in-depth tutorials on a lot of Unity and VR related topics:
https://www.raywenderlich.com/library?domain_ids%5B%5D=3

Free Resources

Free Audiofiles: http://bbcsfx.acropolis.org.uk/

Free Audiofiles: https://www.musicradar.com/news/tech/free-music-samples-download-loops-hits-and-multis-627820

Free 3D Objects: https://sketchfab.com/search?features=downloadable&sort_by=-pertinence&type=models

Free 3D Objects: https://3dwarehouse.sketchup.com/

Free 3D Objects: https://www.turbosquid.com/ (Free registration required)

Free 3D Objects: https://free3d.com/3d-models/furniture

Free 3D Objects: http://www.3dsmodels.com/

Free 3D Objects: https://answers.unity.com/questions/16650/game-asset-website-list-free-and-paid-textures-mod.html

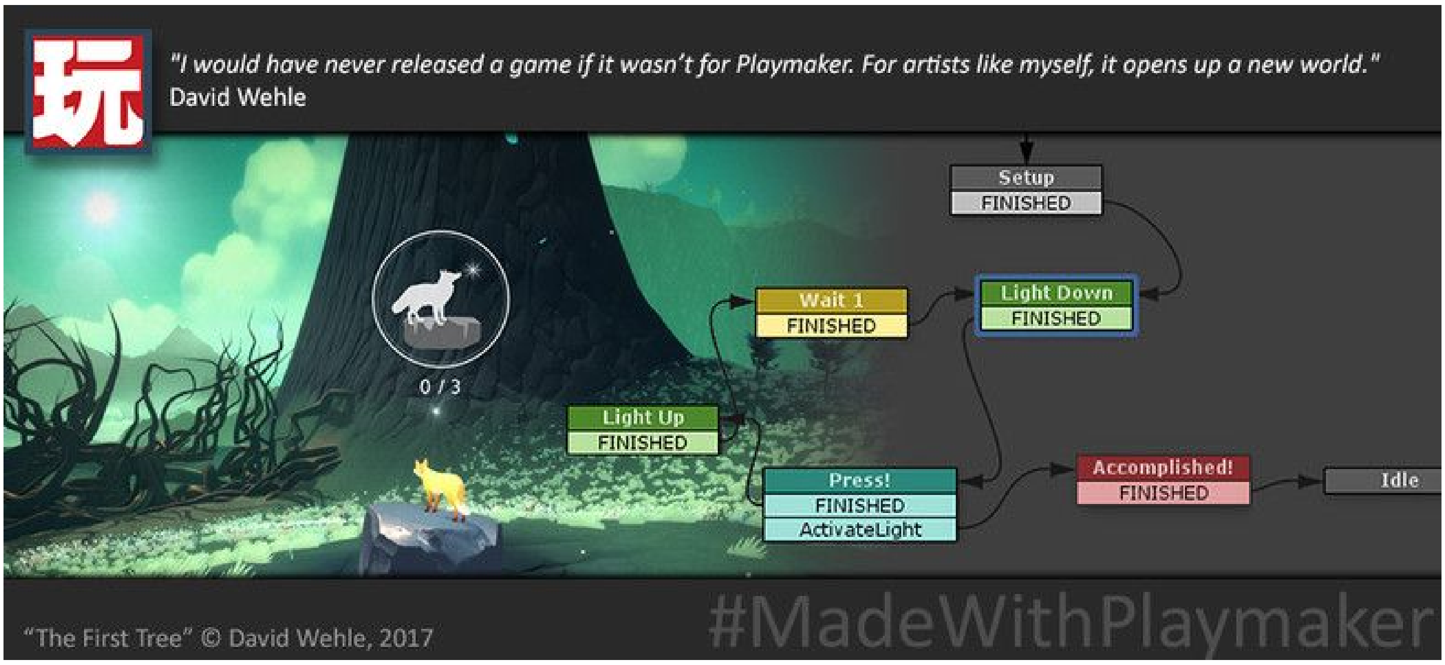
Free 3D Editors

TinkerCAD: https://www.tinkercad.com (you can export STL and OBJ Files)

SketchUp Free Online: https://app.sketchup.com (you can Export only STL-Files that can not be imported into Unity3D

Convert STL Files to OBJ: http://www.greentoken.de/onlineconv/

Playmaker


玩
"I would have never released a game if it wasn't for Playmaker. For artists like myself, it opens up a new world."
David Wehle
Setup
FINISHED
Wait 1
FINISHED
Light Down
FINISHED
Light Up
FINISHED
Press!
FINISHED
ActivateLight
Accomplished!
FINISHED
Idle
0 / 3
"The First Tree" © David Wehle, 2017
#MadeWithPlaymaker

Quickly make gameplay prototypes, A.I. behaviors, animation graphs, interactive objects, cut-scenes, walkthroughs...

Artists and Designers: Realize your creative vision without coding! Unlock the power of Unity.

Programmers: Add a powerful visual state machine editor to your toolbox. Interface with scripts or extend Playmaker with Custom Actions.

A limited version of Playmaker is free with the workshop - but limited to educational use only!

You can buy a full Playmaker Version for a limited time with a special discount using the promo code **virtualspatialsystems :**

https://sites.fastspring.com/hutonggames/instant/playmaker?coupon=virtualspatialsystems

Tutorials and Manuals:

https://hutonggames.fogbugz.com/default.asp?W1

http://www.hutonggames.com/tutorials_game_design_with_playmaker.php

https://hutonggames.fogbugz.com/default.asp?W548

Youtube Tutorials:

intro: https://www.youtube.com/watch?v=1Eh7aPTcq1I&t=14s

01 States - Light Switch: https://www.youtube.com/watch?v=I9VwsVtbgFU

02 States - Open and Close: https://www.youtube.com/watch?v=dmUEN3txNPY

03 Triggers: https://www.youtube.com/watch?v=BYwkd80T5Zk

…