Level 11 - UNIFY MY WORLD

Getting Familiar With Unity's Structure

This award-winning game development engine is a relative newcomer to the game engine field, starting in 2005 with the first version emerging after 4 years of development, debuting at Apple's WWDC (Worldwide Developer Conference). By 2007 they had launched version 2 of the software which had support for console development on the Nintendo Wii. In 2008 they announced iPhone support for the program, one of the first engines to do so. By 2009 they were offering a free fully functional trial version for educational and non-commercial purposes, and by 2010 they had 250,000 developers registered. In fall of that year they released the redesigned version of Unity 3D, and as of this writing there are now well over two million registered developers using Unity 3D.

There are currently a few versions of Unity 3D. The free version is called Unity Personal while the paid versions are Unity Plus and Unity Pro, paid for via a monthly subscription fee. These flavors of Unity can be downloaded for free with a limited timed trial license for Unity Plus and Pro. Presently, the differences in terms of features between Unity Personal and Unity Plus/Pro are fewer than most Free and Pro program combinations. For a complete list of the differences check out the Unity website **www.unity3d.com**.

Platform Development Support

Here's a basic list of the supported platforms as of this publishing:

• Mac/Linux Desktop application
• PC Desktop application (including Windows 8 support)
• WebGL (HTML 5 / Javascript based, requires browser support)
• Nintendo Wii and Wii U
• PlayStation 3, Playstation Vita, Playstation 4
• Xbox 360, Xbox One
• iOS (iPod Touch, iPhone, iPad)
• Android OS
• Windows Phone and Windows 8 Mobile
* Samsung Smart TV
* Tizen

UNITY'S AUDIO ENGINE: UNDER THE HOOD

As with a lot of game engines the Unity developers chose to use an existing middleware audio toolset in order to generate sounds in game. Unity's audio duties are thus handled by an older version of the FMOD audio engine (called FMOD Ex), and inherit a lot of terminology from it as well. Note that this is NOT the same as the FMOD Studio version that is in more common usage today. It is older technology .

Additionally Unity only comes with a basic low-level access to the FMOD Ex API, and Unity's programmers have implemented only a portion of its more advanced functionality and none of its GUI-based features that are available in the FMOD Designer app. Access to these FMOD-derived features in Unity (beyond the basics, which we'll be covering) is usually done in code via Unity's API, based around a C# code interface (plus Unity's two shorthand scripting variants, Javascript(UnityScript) and Boo. While the API uses a few FMOD terms, it does not resemble FMOD code or scripting in any significant way.

To be clear, this means that there is no direct support for the older FMOD Designer functionality in Unity. The good news is some of this functionality can be enhanced through third party middleware and extensions. Additionally Unity's own audio engine can itself be replaced entirely by a different one through the use of plugins. These are described in the paragraph below.

Extending Unity's Engine: Supported Audio Formats

Like quite a few other game engines, Unity supports the use of plug-ins. These provide additional or alternate functionality than can be provided through the C# scripting interface. This extends to audio engines as well. Both **Audiokinetic's Wwise** and **FMOD Studio** (the newest version of FMOD) have some level of integration available in Unity's Editor. This essentially bypasses Unity's own audio engine, replacing it with the plugin's version. **Tazman Audio's Fabric** also extends functionality using the native Unity audio engine. While all these integrations are improving greatly in terms of usability, it still will require some coding expertise to use them in a full fledged game.

Supported audio formats for Unity include:

• **Uncompressed Audio support** : Unity supports WAV and AIFF, the two most common audio formats in use today.
• **Compressed Audio Support** : Unity supports both import and playback of MP3 and Ogg Vorbis files. Ogg files are supported in desktop applications (Mac, PC and Linux) while MP3 files are supported on import for all platforms. ADPCM is also supported, which can be useful in certain cases.

• **Modtracker Support** : Unity supports four formats of Modtracker files which offer incredibly compressed file sizes (these are actually MIDI files combined with compressed sample data, made popular in the nineties). Flexibility is extremely limited however. You can only stop or start the modtracker file.
• **Audio Encoding** : Unity can encode uncompressed audio into Ogg Vorbis for desktop and some mobile applications, and uses MP3 and ADPCM for mobile applications due to the presence of hardware decoders that support it (especially on iOS).
• **Support for Filtering/Effects** : Because Unity uses FMOD's audio capabilities, it does come with a lot of extra DSP-oriented features. These features are supported in every version from Unity 5.x onward and include most of the plugins covering EQ, filtering, reverb, compression, volume ducking, etc, and much more are now available with Unity's open plug-in SDK. Recent to Unity 5 has been the addition of a full fledged flexible Mixer object that uses automation snapshots in combination with the plugins to provide a full fledged game mixing system.

HOW SOUND WORKS IN UNITY

The basic principle of sound in Unity is determined by two Game Object Components—the Audio Listener and the Audio Source. Both of these components work in combination with the sound file itself. You can think of the Audio Listener as you—or, more specifically, as your ears, or perhaps a microphone—while the Audio Source is, as you might expect, the source of the audio, which can be any object in the game world.

Audio Listener

The Audio Listener Component is usually attached to a first- or a third-person controller combined with a camera (which represents your eyes or your view of yourself). The Listener serves as both your ears and as the master output of the game's audio, whether it's 2D or 3D. Unity limits you to only one Audio Listener per scene.

Audio Source

The Audio Source Component is just that, a source of audio in the game world. Audio sources can be set up to play 2D or 3D sounds in a variety of ways. One way is a simple ambient background that loops. You've already heard examples of this in the Mushroom Forest game environment we covered in a previous level.

If the Audio Source is attached to an object and plays continuously, distance from the object is important in the game, and the volume settings of the Audio Source can be customized to a wide variety of settings based on the distance. Audio Filters, if available, can also be employed by this same method, enabling more realistic depictions of occlusion and obstruction.

Triggering an Audio Source

An Audio Source can be triggered by a number of conditions, but only a very few of them can be configured using no coding or scripting expertise. The simplest of these is an object appearing in the game space for the first time. The sound is tied to the source by a sound trigger, and looping can be turned on or off as well.

Any other type of triggering will require some scripting knowledge and usage of other components. For example, an object might make a sound if the player runs into it. The object will not play the sound on its own, however; the sound requires the trigger event, which in this case is the collision, and the direction to play a sound comes from the script. Another example would be a sound that occurs when you click on the mouse to fire in a first-person shooter. In this case, a script is setup to look for mouse events, so when a mouse event is detected, the script triggers the sound to play.

Hands On With Unity

In order to give you a better idea of how audio sources can function and be triggered inside a game environment, you really need to understand at least a little bit about how Unity is put together. We're going to take you step-by-step through Unity to help you re-create an extremely simple game function. A player will walk around a simple virtual environment with an ambient background and walk into an object, which will make a sound of some kind. In the process you'll learn at least a little bit about how Unity itself works and begin to understand the logic underlying its engine.

To continue, you'll need to download Unity and install it first. Open a web browser and go to **www.unity3d.com/downloads**.

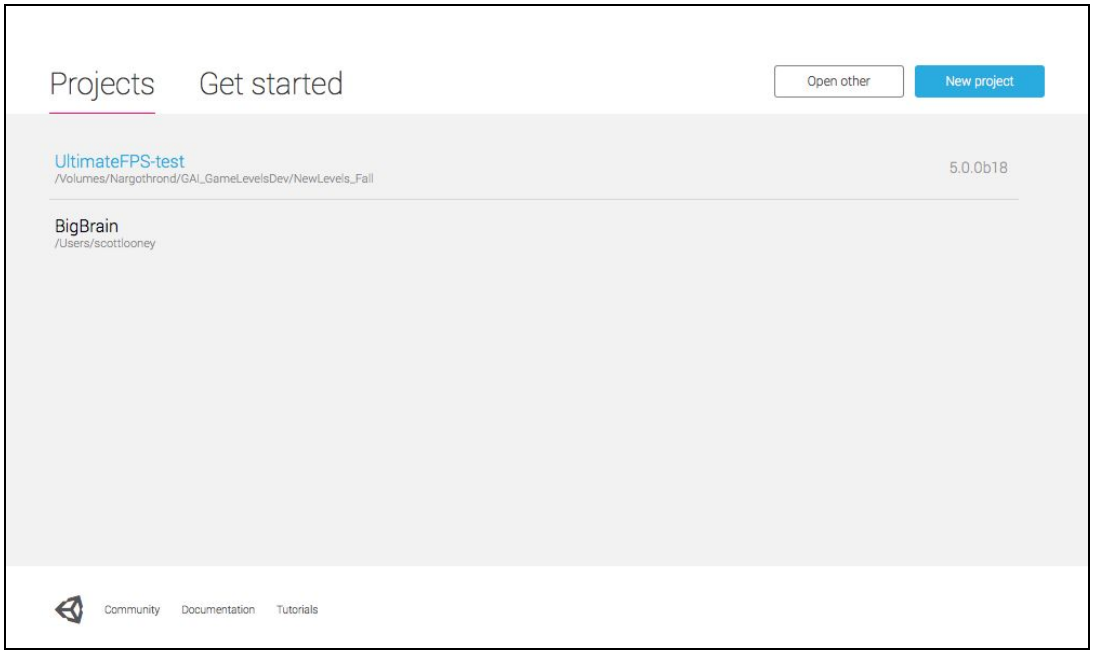
NOTE: Generally you will be using the Unity Download Assistant which will allow you to install Unity and also optionally install build support for other platforms. Unity 5.x will only build on WebGL by default, so if you want to build a standalone Mac/PC/Linux app or iOS, Android, etc, you will have to check those options in the Assistant.

Once you've downloaded and installed the program, go to the following website to get the free demo level project file (**http://gameaudioinstitute.com/unitywalkthroughlevel/**). Download that zip file, uncompress it into its folder and we'll be ready to go.

Getting To Know the Unity Interface:Hands On

I'll start off by saying that this is one complex program to use for sure, and we are only going to look at the barest essentials that specifically relate to importing and adding sounds within the program. It's extremely deep, but that being said, the sound part is relatively easy to deal with.

Let's open up Unity. On a PC you will find it under C: (or it may also be 'My Computer') \Program Files\Unity. On a Mac its normal location is in the /Applications/Unity folder on your hard disk. Open this folder and double click the App to start it up. If this is your first time starting up Unity you'll be presented with a registration screen. You can either register online or take the registration key to another machine that is connected to the Internet. The basic download of the game comes with a license for Unity Personal and an option for activating the limited demo of the Unity Plus and/or Pro versions. Once activated you should see this window.

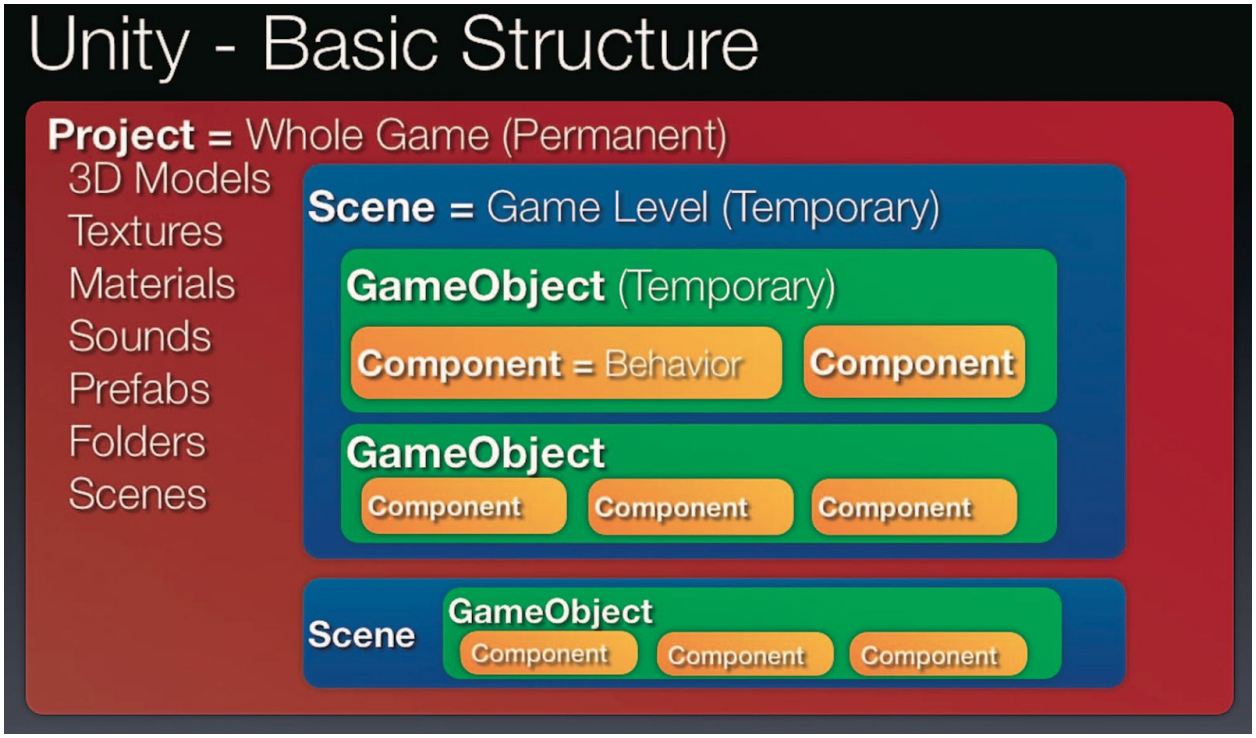


Anatomy of a Unity Project

If you close the Welcome screen what you will be looking at is a Unity Project in the Unity Editor. Most likely you will be looking at the demo project that comes with Unity. While it's pretty cool to look at, there are a lot of objects in it and it may get a bit confusing. So let's strip things down and give you a somewhat simpler project, courtesy of the Unity Asset Store and Mixed Dimensions and modified by the Game Audio Institute(GAI).

In this window you'll see the list of recently opened projects (which should be largely empty). We want to open a new Project, so choose **Open Other** to load a Unity Project and navigate to wherever you saved the folder you downloaded earlier from the GAI website. In most cases this will be in the /Users/(username)/Downloads folder on a Mac, or on the PC it's usually under C:\Users\(username)\Downloads. When you have found this folder just click Open on the folder itself—you don't need to go into the folder to find a file. Let this file open up, and we'll be ready to go!

NOTE: If Unity gives a warning that it's going to update the project and you can't go back to an earlier version, this is normal, as Unity is updating fairly frequently. Just choose the OK or Continue button to move ahead.

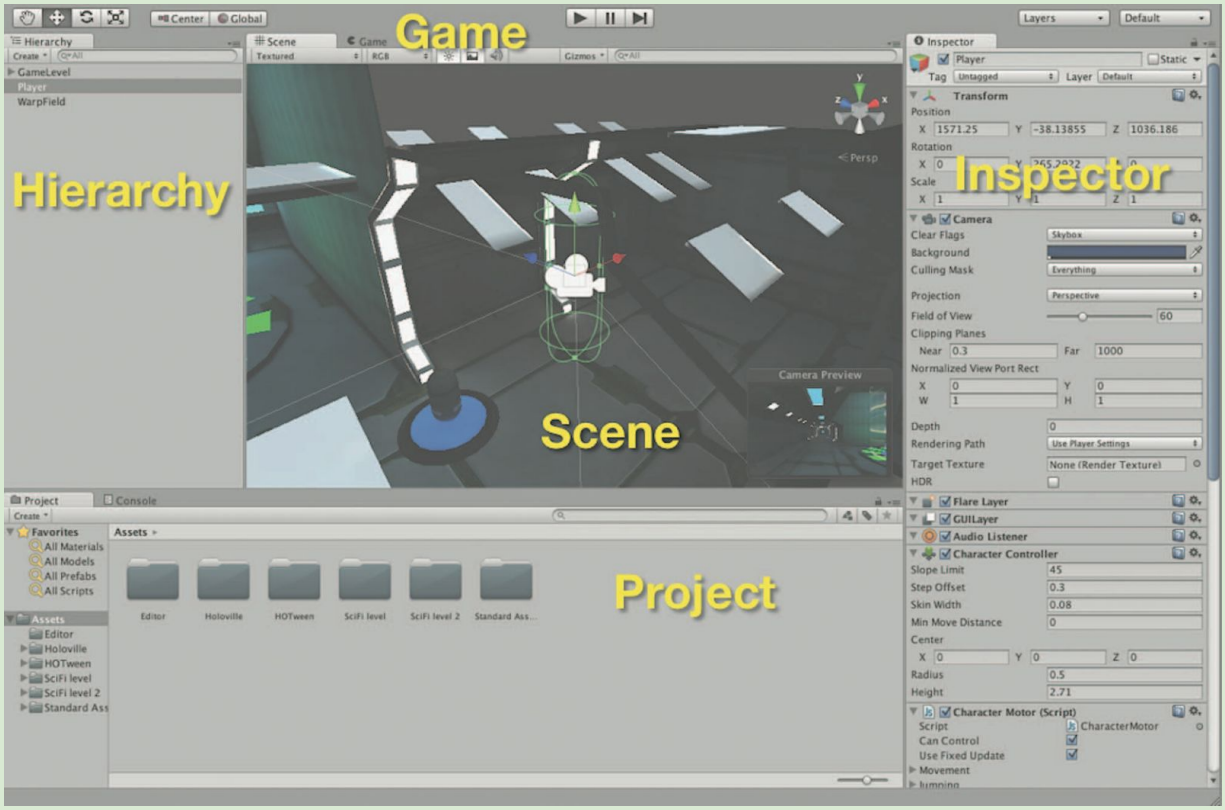


STRUCTURE OF UNITY

The Project

Let's talk about the basic structure of Unity, Starting with the Project. The Project is the largest file structure available in Unity. Everything in Project goes into a master folder with the project's name. This means all of your tangible game assets, as well as all other kinds of utility files and stored settings. The analogy of a project is that it is basically your whole game. Copying the entire Project folder will effectively duplicate your game, and is a good idea as a backup plan, should your original Project folder get corrupted. The keyword here is permanent . The Project is a place for things you will use in the game and they generally won't be deleted or altered in any way.

A View To Your Project



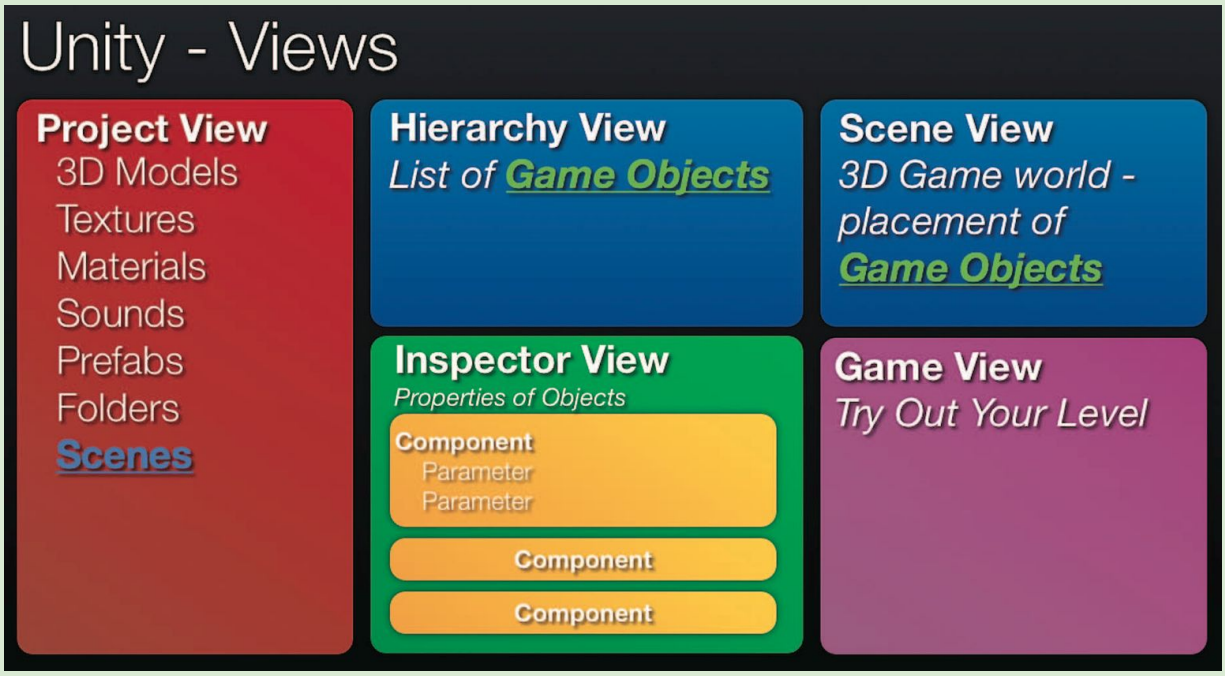
Unity organizes the contents of your project visually into what are called Views . These are extremely flexible—you can adjust boundaries on any View by dragging its edges around the screen and it will resize accordingly. Additionally any View can easily be maximized by simply mousing over its tab (you don't have to click) and pressing Shift and Spacebar. The maximized View can then be toggled to return back to its original size by pressing the spacebar again. Try it out—it's fun! Each View has its name printed clearly on its Tab. There

are a quite a few of these, but we'll be covering the Project , Hierarchy , Scene Inspector and Game Views in this level.

To get ourselves on the same visual page as it were, look in the far upper right corner of the main editor for the Default label. The Default layout is the Hierarchy View on the upper left, Scene/Game Views in the top center, the Inspector View on the right side, and the Project and Console Views on the bottom left and bottom center. This is what is indicated in the screenshot.

This screenshot shows the placement of Views in the Default Layout. Views can be customized to suit your needs and preferences, and you can save and recall Layouts as well.

Let's check out the Project View first. The most common part of any Unity project that you will be dealing with is the Assets folder, found directly under the main project folder in the View. The Assets folder contains all of the aforementioned tangible things like 3D models, textures, scripts and of course, sound files. To import assets into a project, you will generally be dealing with this View. Doing so will make a copy of the asset, which is the recommended method. You can also add things like scripts and folders to the Project as well as many other things, but we need to be moving on. Let's talk about the Scene file.



The Scene

Significantly, the Project's Assets folder will contain at least one Scene file. The Scene is where all of the less tangible things take place in a game. The keyword to think of here is temporary . A Scene file is similar to a single game level in a game. In the Scene environment things become less tangible and more virtual in nature. Opening up a Scene file will display its

contents in the Hierarchy View as a name and also visually as an object placed in the Scene View. One thing you may notice is that when we opened the Project we ended up in an Untitled Scene . If you look at the contents of the Hierarchy View in the upper left side and the Scene View you'll see that they will only have a Main Camera object in them and nothing will appear in the Scene or Game Views. We need to open a Scene file, so look for the file called **TestSound-SciFi.unity** in the Assets folder in the Project View and double click that. Now that we've done so, we can see the list of objects on the left and the visual representation of our game space just to the right side in the center in the Scene View.

It's extremely important to point out here that these Objects in a Scene can change constantly. For example, if I'm playing the game and shoot a robot and it explodes, the Robot Game Object is removed from the Scene. Or if suddenly a dozen magic fireballs appear and flame out, this means they get created, go through their animation and then get deleted. The Scene is a totally dynamic place where anything can happen (and usually does).

The Game Object

This is the virtual building block of a Scene. Everything listed in the Hierarchy View you're looking at is a Game Object . These can be physically visible inside the game space, or they can be invisible. Every Game Object possesses characteristics and/or behaviors. Unity refers to each of these specific behaviors as Components.

The Component

Click on any object in the Hierarchy to see these Components on the right side in the Inspector View. Every Game Object in a Scene will have at least one Component called a Transform (which governs every object's position, size, and rotation), and more Components can be added.

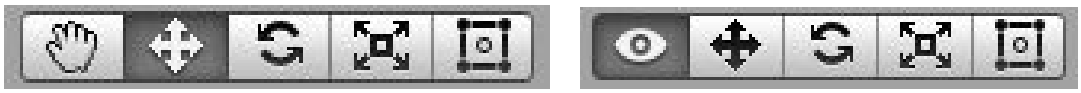
Some Components are predefined, and others can be customized, such as scripts. Components will have various settings or parameters associated with them and these of course can change. To do audio in Unity you will also need to add Components (the aforementioned Audio Listener and Audio Source).

So that's the basic structure in a nutshell—the Project contains all your real assets, including Scene files, which contain virtual Game Objects. These in turn contain Components that define the behavior of a particular object. Now let's examine how all of these Game Objects in a Scene are visually represented by getting into Scene View.

Moving Around in the World In Scene View

This is where the vast majority of graphic editing and arranging is done for objects within Unity (although Unity is not a graphic editor, remember that!). Let's examine the objects in the Scene—if you take a look on the left side you'll notice fairly few Game Objects, and the visual

material in our Scene view is equally sparse. You have a terrain, a wall, a player object and a sphere representing a basic Audio Source. Now in this window your view is controlled by the camera, which you can zoom, pan and rotate. Since navigating areas of a scene in Unity is pretty vital to being able to select objects in the scene, let's go over the ways we can control our view. There are a lot of methods but I find the most usable way is to have a two-button mouse or to have a right-click setup on laptops that have multi-touch trackpads. This enables Flythrough Mode , and at this point you can use keyboard keys to navigate through the scene. First, though, locate the Toolbar section on the top left—it should look something like this:



Click on the Hand object to select the Camera Move mode, then right-click (you cannot use Control-click for this, there has to be a dedicated right-click option) and hold. As you are holding, you have entered Flythrough mode, and the Hand tool should turn into an Eye, and the cursor in the screen should also be changed to an eye.

Now you can efficiently navigate through the scene. Using the mouse you can change view and pan rotation similar in a manner used by RTS games. You can use this now to change angle and elevation with the addition of the following keys:

• W Zooms into the scene
• S Zooms back out of the scene
• A Tracks camera to the left
• D Tracks camera to the right
• E Tracks camera up
• Q Tracks camera down

You can use the mouse or trackpad plus the Hand tool to move around the scene as well, and there are alternate controls, which you can use if these don't work or you're really curious:

• Use the arrow keys to move around on the X and Z axes
• X Axis (left/right, with the left and right arrows)
• Z Axis (forward/back with the up-arrow forward and the down-arrow back)

• Hold Option/Alt and click-drag to orbit the camera around the current pivot point

• Hold Option/Alt and middle click-drag to drag the Scene View camera around. You need a three-button mouse (or trackpad shortcut) for this.

• Hold Option/Alt and right click-drag to zoom the Scene View. This is the same as scrolling with your mouse wheel or using the W and S keys in Flythrough.

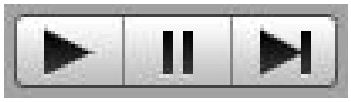
One extremely important thing to do is to be able to center your view on something you've selected. So, once you have found the object you want to select, click on it and press the F (Frame) key with the cursor in the Scene View. This will center the Scene View and pivot point on the selection.

Help, I'm Lost In Space!

We've all been there. Suddenly your camera is pointing towards some random direction and you're totally lost. So another extremely nifty thing, is that if you know the name and location of a Game Object in the Hierarchy, you can click on it, hover the mouse within the Scene View (don't click in the Scene View, as this could potentially select another object), press F and the view will instantly zoom to that object and center it in the view. You can also double click on the GameObject in the Hierarchy.

OK—Take a few minutes and try flying through the scene using the controls mentioned here.

Game View



We have one more important view to discuss and that's Game View. This View is, as you might guess, the view that is active when playing the game or Scene itself. It is inactive until you start the Scene using the playback controls at the top center of the Edit View. When you click the Play arrow, the arrow turns blue and Game View becomes active. If you click the Maximize on Play button on the right side of the Scene View, the game will be played in the maximum screen space possible. In the Game View, you can use many of the controls you used in Flythrough mode (the mouse and the W, A, S, and D keys), but you cannot use the up and down controls (Q and E). You can also use the left and right arrows to track left or right (without turning), and the up and down arrows to move forward or back. If your screen is not maximized, you can select and edit objects in the Hierarchy, Scene and Project while you play the game.

However, there is a very important point to make if you do this. ALL objects in the Hierarchy and Scene Views will revert to their original states when you stop playing the game (including deleting Objects). The Project View will remain unaffected however. So, if you want changes you make to be permanent, you should make changes to assets in the Project View, and not in the Scene Hierarchy View. If you have to remember a change to an Object in the Scene View, taking a screenshot of your Inspector settings is a convenient way to do this. To turn Game View off, simply click the Play arrow again to disable playback.

So now that we've covered this, click the Play button at the top, and play your scene. Walk around using the keyboard keys and mouse/trackpad, and notice the various things in the environment. You won't hear any sound in the project though, because we haven't yet associated any sound files or created any Audio Sources.

Conclusion

As you can see, the Unity game engine is an extremely deep, complex, and involved environment to create games, and we have barely even begun to scratch the surface as to what's possible here. Whether you are just using the App or if you are getting your hands dirty with our demonstration level, at this point, you should have a solid grasp of the basic sound functionality that is available within Unity. You should also know how to get around inside the 3D space. Next, we will actually implement sound into the demo provided. So turn the page, audio adventurer, and prepare yourself for the next level!

Level 12 - UNITY TOO

Basic Audio Implementation in Unity 3D

We've covered the basics of how Audio operates in Unity, the basic overall project structure and how to get around inside the game. We'll continue now with how to actually implement audio in a game environment, as well covering a few more advanced sound triggering options for those interested in what's available currently within the scripting API. This Level is going to involve scripting, and while you may not be doing this as a career, it's very helpful to know the basics of how triggering audio works. It's okay if you don't understand this the first time—scripting and coding are difficult things to grasp. Rest assured that there is enough general information in this chapter so that even if you can't code like an expert, you can at least sound like one in a conversation. Note: If you want to follow along with the step-by-step instruction below, you will need to have already downloaded Unity and our simple example project covered in Level 11.

Importing Sounds as Audio Clips

To take our next step, let's import some sounds. Find any sound file you have around and just drop it directly into the Project View. You'll notice it becomes a file that can be viewed and previewed. Click on it and let's cover a few of the options available when importing sounds into Unity.

The first thing you should notice when you click on any audio file is the Inspector View fills up to show the Importing options. Unity refers to any audio file imported as an **AudioClip**. Let’s take a look at the different audio importing and configuring options available to us. You're looking at the Audio Importer settings for this **AudioClip**. This is an area in which you will likely be spending some time configuring settings for audio files, but for now we'll just touch on some highlights and move on to getting these audio clips in the game environment.

The Audio Importer allows you to configure settings for any imported audio assets in your game. At the top there are some common controls:

- **Force To Mono** will mix any stereo assets down to mono in game, which saves space and can be a good idea if the sound has no perceivable stereo channel separation in game. You can also choose to Normalize the summed file which can be useful. Also any changes applied get done to a copy of the file and not the original.

- **Load In Background** is disabled by default. When enabled it allows these AudioClips in a scene to be loaded via a separate process rather than the main one, which can be beneficial in certain situations.

- **Preload Audio Data** is enabled by default and will preload AudioClips when the scene loads. If this is disabled, you will have to have separately preloaded the data manually, or it will load into memory when the sound receives a Play command.

In the next area down, you can configure how AudioClips are handled in memory, as well as define your compression settings and even sample rate conversion (yes, it's still a viable option even in the 21st century).

NATIVE VS COMPRESSED MODES

Native Mode in Unity means that Unity will not touch the format of the audio file (uncompressed or not) and simply play it back as it was imported. Native Mode is usually recommended for short uncompressed audio sounds. Compressed Mode by contrast will compress the audio assets either as Ogg, MP3(on iOS), or ADPCM. when the project is compiled into an application. Note that Unity will not care if the files are already compressed when you import them—if you select Compressed Mode it will re-compress the audio again, which is usually undesirable and will likely create more artifacts.

AUDIO LOADING MODES

These loading options are available when selecting either Mode:

• **Decompress On Load** : Unity will the decompress file when loading it. This is recommended only for shorter compressed files, as longer ones will potentially create a processing delay.

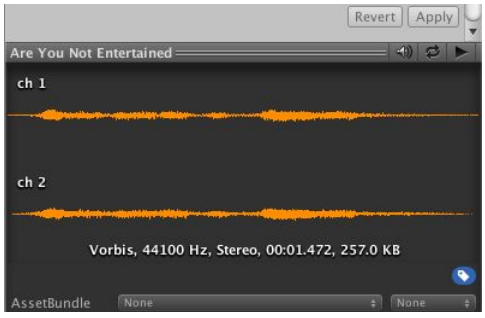
• **Compressed In Memory** : Unity will keep these files compressed until playback when they will be decompressed. Recommended for larger compressed files.

• **Stream From Disc** : This method is mainly desired for continuous sounds like music soundtracks. It uses lower amounts of memory but it's usually a good idea to limit the number of items that are continually streaming to one or two items.

The nice thing about this section is that you can create a default setup that will be applied to the asset for all supported platforms, but if necessary you can create individual overrides for each of the many platforms that Unity supports.

Note that you have to install the build support for each of these platforms to see the icons as shown in the screenshot. Again, we'll look at the specifics a bit later, but this section is really important to saving resources as well as having them triggered efficiently in game.

And lastly, any changes to an Asset like an AudioClip MUST be applied. Don't worry though, Unity will remember that you changed something and ask if you want to Apply or Revert your changes. Other really nice things with Unity 5 are that now you can select multiple AudioClips and apply the same changes to them if desired. Multi- editing is awesome!



Preview Section

Beneath the Import Settings is the Preview section. There are some simple tools in the right-hand corner of this area. Within the Preview area, you can do the following (going left to right).

- **AutoPlay Toggle** automatically begins playback of any sound as soon as the sound is selected in the Project view.
- **Loop Toggle** allows you to turn looping on or off.
- **Play and Pause** allows you to start and stop playback. The arrow turns blue if the sound is playing.

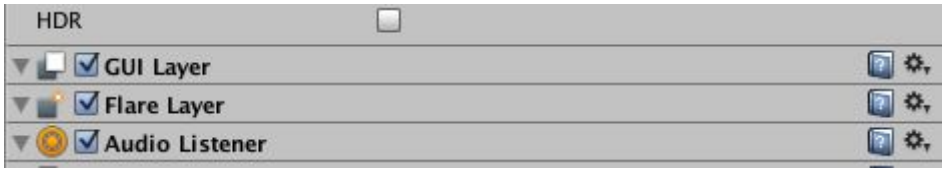
You can click and drag on the waveform of the sound to locate it at any point. Note that none of these settings—playback, looping, and so forth—are attributes of the file; they simply allow you to listen to the sound. Also, you will notice that you cannot adjust the volume or perform any editing on the sound, as you could in Pro Tools or a dedicated audio editor. Unity is NOT an

audio-editing application. It can automate certain aspects of the sound in relation to the game environment, but all serious editing requires a dedicated audio-editing application.

NOTE: Although you can create a Listener Component you should first ALWAYS check to make absolutely sure there isn't already a Listener Component already implemented. This is very important because Unity only supports one Listener Component in the scene at a time. So make sure there isn't one already working, or you can definitely get into issues with multiple Audio Listeners in the scene.

Check For The Audio Listener

So, we first need to start off by making sure we haven't already an Audio Listener Component installed. One of the most common GameObjects where you will find an Audio Listener attached is the Main Camera. Let's look for this in the Hierarchy. Open the Hierarchy, and in the Search field in the upper right, type the following "Main Camera". This will show the Main Camera object and its Components. Look carefully and you will see an Audio Listener Component. It's definitely not anything fancy, and you can't really configure it much, but it's vitally important that it's there.



Actually, to be truthful, the Audio Listener settings are more global in nature for the whole project. As such, they can be found under Edit > Project Settings > Audio. We already have an Audio Listener in the Scene (it's usually found as an included Component attached on the Main Camera Object . In this case it's attached to the object named Player).

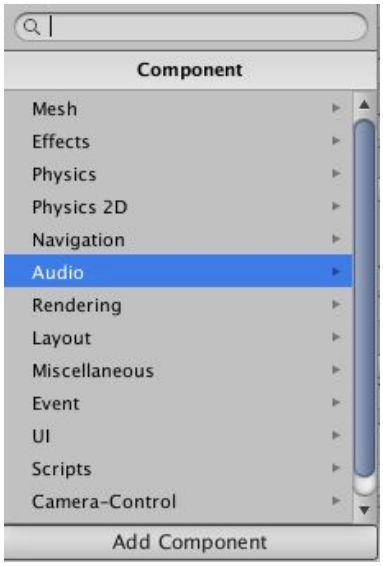
Creating the Audio Source

Now that you've established that the Listener Component is present, let's move on to the Audio Source. This is pretty much the most important audio-related component you need for hearing audio playback in Unity. As the name implies, it is the source of audio, but in Unity 5 there have been some significant changes as to how it works.

To add an Audio Source to a GameObject there are three main methods. We'll rank these in order of complexity, most involved to least:

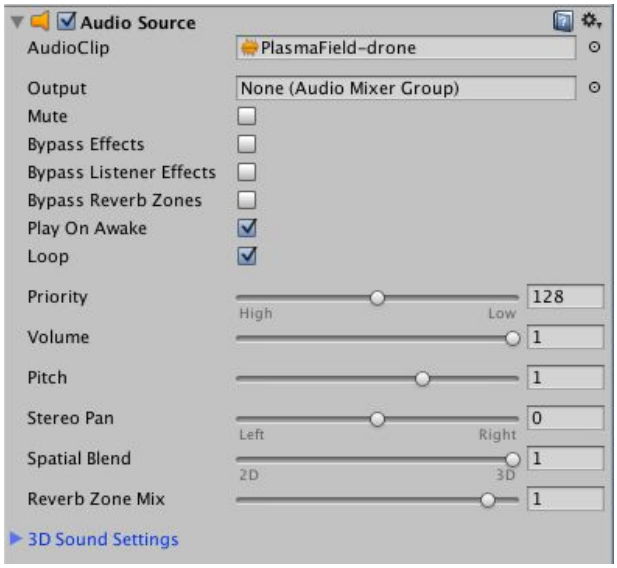
1. Click on the PlasmaField object in the Hierarchy, go to the Component menu, and choose Audio and then Audio Source.

2. Click on the PlasmaField object in the Hierarchy, then in the Inspector for the object click the Add Component button, select Audio, then Audio Source.



3. Drag an Audio Clip over a GameObject. This can be either the listed object in the Hierarchy or the visible object in the Scene View. Note this latter method may not work if the object is hidden from view or is otherwise not visible.

Let's go over the basic controls:



Audio Source Component Settings

The name of the Audio Source Component is at the top. Clicking on the name will minimize that part of the view. There's a check mark before the name. Use this check mark to deactivate or activate the Component. Activating and deactivating Components can be useful in troubleshooting.

The basic Audio Source settings are directly below the name, in the area above the colored graphic display indicating 3D Sound Settings. Note that by default, the AudioClip setting says None (AudioClip). You can change this by dragging the desired AudioClip directly over the None label and dropping it. You can also click the dot just to the right of this field to select the sound you want from the files listed there. Pick the file marked **PlasmaFieldDrone.aif** or select an ambient loop of your own choosing. When you do this, the None setting will change to the name of the AudioClip.

Let's go over the rest of the controls available:

• **Mute** temporarily mutes the sound. Mute can be useful when you need to isolate a particular sound from others. Leave this setting alone.
• **Bypass Effects/Bypass Listener Effects/Bypass Reverb Zones** temporarily shuts off all effects that have been applied to the sound. Leave this setting alone as well.
• **Play On Awake** immediately plays the sound as soon as the scene is loaded. This setting is useful for ambient sounds. Turn this option on.
• **Loop** will loop the sound in the game. Note that checking this setting is different from using the Loop option in the Preview area, though it has the same effect. The Loop setting causes the Audio Source to loop continuously, so it is useful for background sounds. Turn this option on.
• **Priority** allows you to rank sounds according to their importance. You can rank sounds from 0 to 256: 0 indicates the most important sounds and is reserved for music soundtracks and other audio that must be heard, while 256 indicates a sound that does not need to be heard much at all. Leave this alone for now.
• **Volume** is the maximum volume the Audio Source will have: 1 is the highest volume possible, and 0 mutes the sound.
• **Pitch** is like a record speed control. Effectively, it adjusts the playback speed of a sound. Because speed and pitch are linked, a faster speed (above 1.0) creates a higher pitch with a shorter length, while a lower number (below 1.0) creates a lower pitch with a longer length.
*** Spatial Blend** is a brand new control for Unity 5 and replaces the AudioClip based method in Version 4. All the way to left the sound is considered 2D and will not be affected by distance from the Listener, and to the right the sound will be entirely 3D. Set this control all the way to the right.

3D Sound Settings

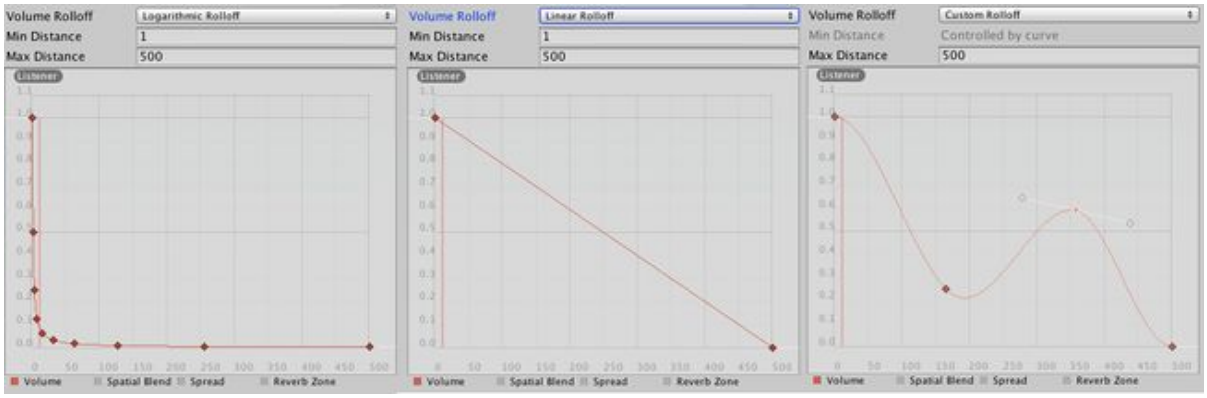
Unity uses the Min and Max Distances concepts in order to determine how sound is perceived within the 3D game space. These concepts are directly inherited from FMODs functionality. To think about this,imagine two spheres placed one inside the other—you have an inner sphere

and an outer sphere. The inner sphere represents the minimum distance that you are from the sound while the outer sphere represents the maximum distance. Note also that the number settings of Minimum and Maximum distance are expressed in GU or Game Units. This measurement is mainly metric in nature, so you can think of the settings in meters rather than feet, which is an important distinction to consider.

The **Min Distance** is the distance at which the sound will be at its maximum volume, and no matter how much closer you would be able to get to the Audio Source, the sound will not get any louder. Essentially it's the maximum volume that can possibly be obtained from the source as perceived by the Listener.

The **Max Distance** is the distance at which the sound stops attenuating. Whatever volume is set at Max Distance, it will stay at that volume. As you come closer to the Audio Source, the volume will gradually increase. Unity defaults Max distance to 500 GU (meters) or about 1600 ft which is quite a distance, so it's usually a good idea to reduce this if you're relying on 3D distance to affect the sound.

The difference in volume between the minimum and maximum distance of an Audio Source is governed by the rolloff curve, available in the 3D controls section. Let's take a look at this.



The 3D Settings shown here indicate different types of rolloff curves. The curve of the graph essentially governs the way that the volume will be reduced between the Min and Max Distance from an Audio Source. There are two preset modes— Linear mode, and Logarithmic mode. Although Logarithmic mode is considered more realistic to the way the human ear hears sound, Linear mode may be preferred in some cases, as the volume will drop off more gradually in comparison.

You can also customize the curve by adjusting any of the nodes as shown on the right, or double-clicking the mouse to create more nodes. When you do so the menu will indicate that you are now creating/editing a Custom curve, and the Min Distance setting will blank out, because it is to be controlled by your custom curve rather than a preset distance. In this case we've started with a Linear setting and then we adjusted it by adding a point to curve the line just slightly.

There are a few more of these 3D settings to mention. Most of these can also be controlled by curves drawn in the same graph area as Volume:

• **Doppler Level :** This is a static control (no curve available) that changes the pitch of a sound based on the velocity (speed) of the Listener or the Source, enabling you to get semi-realistic effects when moving fast. It's not very convincing, however, as it only changes pitch and not phase. Best to use it sparingly or not at all.

• **Pan :** The pan control does not behave the same as a standard pan control in a typical stereo configuration. In effect, it governs how well the stereo sound is perceived from the Audio Source within the 3D Game space. High settings will result in the stereo channel separation clearly being perceived, while low settings will tend to create more of a monaural sound, although still distributed through the 3D audio engine and 'heard' through the Listener.

• **Spread :** The spread control is most useful for surround configurations such as 5.1 or 7.1 surround systems. It governs how effectively the separation of individual speakers will be perceived inside the game from that Audio Source. Low settings will tend to make it sound more centralized and toward the front, while higher settings will distribute the sound more equally between all the speakers. Let's now play our Scene so that we can hear the sound that we just put in our Audio Source. First, make sure that you save your Scene—this is going to be very important going forward. Then click the Play button at the top to play our Scene and listen to the looped AudioClip we placed in the scene as we walk around.

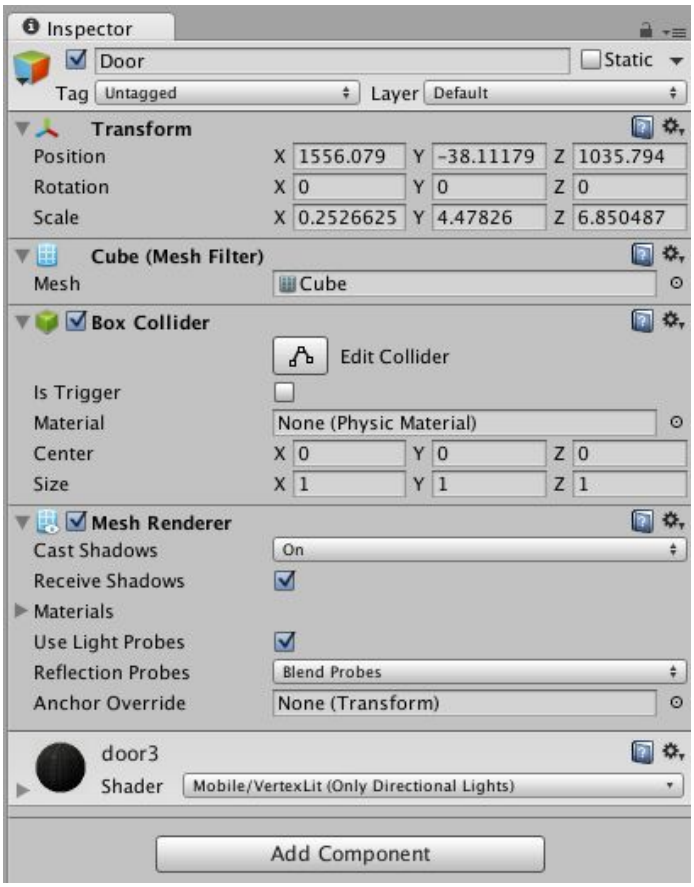
You'll recall the rule we specified last level that all changes to a Game Object in the Scene during Play Mode are lost. Actually here's where we should mention one significant exception of sorts, specific to Audio Sources that are set to both Loop and Play On Awake. First, make sure you're not playing your Scene. Then look at the Scene View bar at the top. You might have noticed a little speaker icon on the top bar towards the left.

Turning this on as shown will automatically enable you to preview these Audio Sources in the game itself, although you'll have to move your view camera around to hear the effect of it. Essentially it puts the Listener on the Scene camera, so that it can be heard while moving around in the scene area. This can be a useful feature as you can then set and edit your Min and Max Distances and try things out before going into Game mode.

Triggering a Sound in Unity

Now, for our next trick, we're going to make it so that the character walking around will run into a door, and that will trigger a sound. You might recall in the last level, we talked about the importance of physics inside a game engine. That's what I'm going to use in order to generate

this sound. Game objects inside Unity can have a Component known as a Collider attached to them. In fact for most simple objects that can be created by the Game Object menu, this Component is already automatically added to them.



Let's take a look at the **Door** object. Search for this object in the Hierarchy, and click on it to review the inspector view for it. Look for the Box Collider component attached to the object. Now, find the Is Trigger option and click on it to enable the trigger functionality.

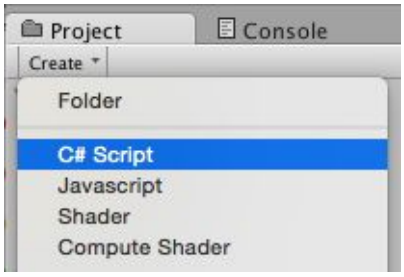
Add an Audio Source to this via **Add Component Button >Audio>Audio Source**.

In the Audio Source leave the AudioClip field blank and turn off Play On Awake and Loop if they're not already off . You might wonder how we're going to hear a sound now, since the Clip field is blank. The answer is that an Audio Source can just be a source of audio for a group of AudioClips. In fact you could trigger dozens of AudioClips to play through a single Audio Source. The significant limitation here is that it cannot adjust volume independently for all those files playing through them.

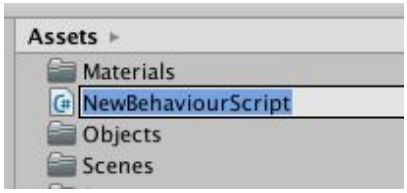
Setting Up A Script

Now comes the slightly complex part—or the fun part, depending on what you think about programming script files. And In this version we're going to use C# and not Javascript/Unity Script as we showed in the book. C# is a bit more tricky but luckily, this is a fairly short script. Let's get started:

1. Create a script in the Project View by clicking and holding the Create button and selecting C# Script from the drop-down menu.



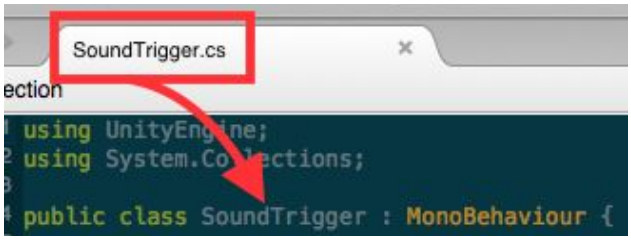
2. You should then see this:



3. You need a better name than "NewBehaviourScript", so rename the script "SoundTrigger" (or any name you want, it does not matter, but make sure you don't click anywhere else. This is the best way to name the file).

4. Now you need to edit the script. Select the SoundTrigger script from the Project View and look at the Inspector View. In the upper left, click on the **Open** button to open MonoDevelop, the default text editor in Unity. This will take several seconds.

5. At the top in MonoDevelop, look very carefully at the name after the words "public class". It should be on Line 4. It must exactly match the name of the file. There can be situations where you do not immediately change the script name from "NewBehaviourScript" in the Project View at creation time and instead rename it later. If you do this, the script's public class will still read "NewBehaviourScript" and must be changed by hand to the actual file name of the script.



6. Next in MonoDevelop skip these first four lines, and then delete the bit of script that already exists after this including their opening and closing braces:

Void Start() { }

void Update (){ }.

Make sure to not delete the last closing brace as it is needed for the main script class. Here's what you should see when finished editing:

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class SoundTrigger : MonoBehaviour {
5
6 }
```

7. Then type the following code exactly starting on Line 6:

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class SoundTrigger : MonoBehaviour {
5
6 public AudioClip Sound;
7 public AudioSource triggerSource;
8
9     void OnTriggerEnter(){
10         triggerSource.PlayOneShot(Sound);
11     }
12 }
```

So what is going on in the code? Let's go through a step-by-step explanation.

Lines 1 and 2

Using UnityEngine;

Using SystemCollections;

These two lines refer to what are called namespaces. Essentially, this means a library of methods the script will refer to. The first one of these refers to functions in the Unity Engine itself, while the second refers to a standard C# library for common generic methods. Other namespaces can be added as needed.

Line 6

public AudioClip Sound;

The first line defines a variable called Sound. The type of this variable is an AudioClip. The script will refer to this variable in the future. The prefix "public" means the parameter will be

exposed, meaning visible in the Inspector, which is going to be crucial for assigning it a value in the Unity Editor later on.

Line 7

```
public AudioSource triggerSource;
```

The next line declares a public AudioSource. This will be assigned later, and prevents us having to type "GetComponent<AudioSource>()" before each audio system command. Note this is a significant change from earlier versions of Unity as well as from the book, which could use the "audio" keyword for the local Audio Source.

Line 9

```
void OnTriggerEnter(){
```

The function OnTriggerEnter() is an existing function that is associated with collision events. A collision event occurs when two three-dimensional objects collide. In this case, the collision occurs when anything with a collider and Rigidbody encounters the Door object. Collider data can contain a lot of useful information. However the trigger variation is much simpler in nature. It's merely a single method that checks (in this case) if the object has entered the Door area.

Note the opening brace and the closing brace after the next line. This is the conventional part of the code that says, "If an object collides with another object, then do whatever is within the braces."

Line 10

```
triggerSource.PlayOneShot(Sound);
```

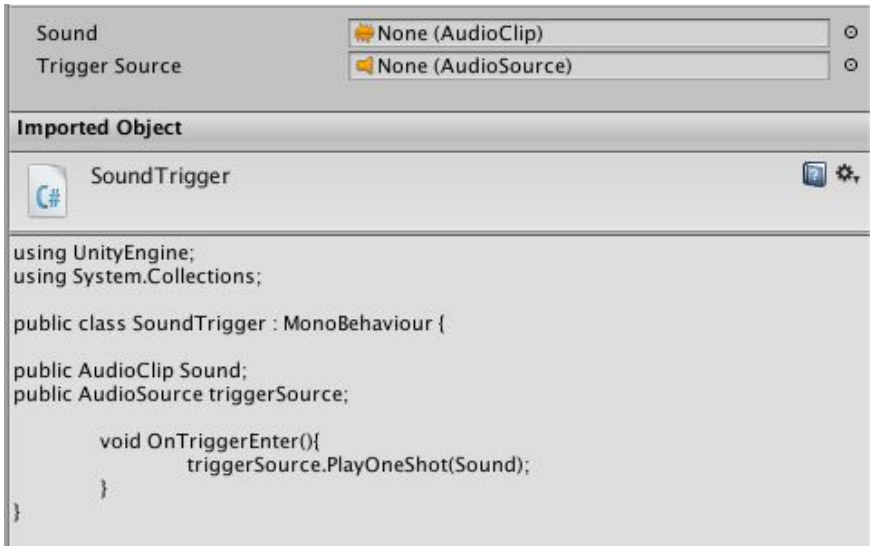
The triggerSource.PlayOneShot (Sound) command will play an audio file one time. What sound will it play? It will play the Sound variable created on Line 6. What sound is that? Any sound you define as the variable in the Inspector.

Line 11

```
}
```

Don't forget to close the braces to end the OnTriggerEnter() function:

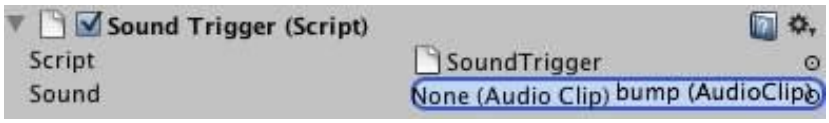
Save this function in MonoDevelop by going to File > Save or by using the keyboard shortcut Command+S. Then close the editor. The text for the function should appear in the Inspector View.



Adding the Script and the Sound

Now you need to add the script to the object, and the sound to the script in the Inspector.

1. Select the **Door** object in the Hierarchy.
2. Drag the Sound Trigger script into the Inspector for the Door. Wait for the plus sign to appear, release the mouse button, and voila! Your script is now added to the Inspector as a component.
3. Now you will need to tell the Sound Trigger script which sound to play. Drag any appropriate one-shot AudioClip sound from the Project View into the Inspector over the Sound Trigger script where it currently says "None (Audio Clip)". You can also click the small circle to the right to select from a variety of sounds in the Assets folder including the Sounds folder you created earlier.



4. Almost done! Your last step is to set your Audio Source. Drag the Door object over the Trigger Source setting in the script. This will tell the script to use the local Audio Source for this GameObject. As long as you have an Audio Source present, you should see something like this:

5. Now, Save your scene, and test it out by going into Game View and walking into the Door . The sound should work perfectly. If not, go back over your work to see whether you missed any steps.

Help! I Can't Trigger the Sound

If you experience errors in compiling or anything else, ask yourself the following questions:

1. Is your code completely accurate? It has to be exactly as detailed earlier. If it's missing one character, such as a curly bracket, it can fail compilation and you won't be able to run your Scene.

2. Did you drag an AudioClip to the script variable? In this case you don't want to drag an AudioClip to the heading in the Audio Source Component because the method PlayOneShot requires a variable. So make sure you drag it instead over the Sound variable in the Script Component, not the Audio Source.

Trigger Dilemmas

So one thing that you might notice is that any object that you turn into a trigger automatically becomes transparent to whatever objects are passing through. This may not be the most desirable situation...

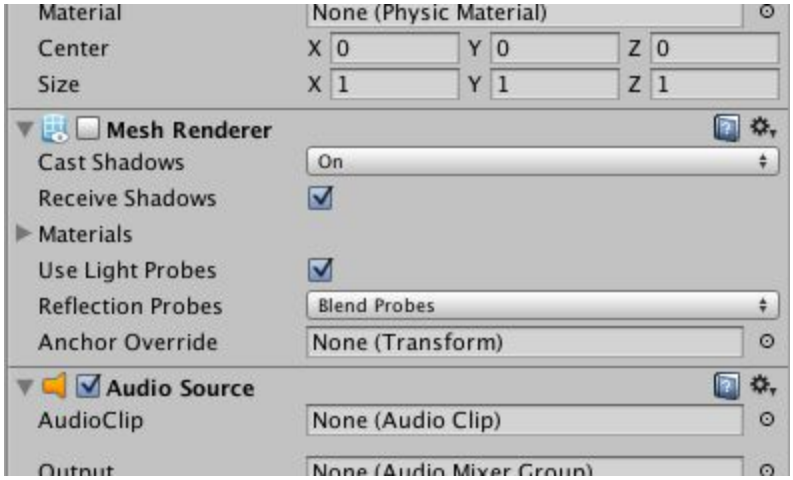
For example, if you were to run into a door you would not want to necessarily pass through the door in order for the sound to play, but rather to literally collide with it. How would you do that? There are a lot of options at your disposal—coming strictly from a position involving only triggers, what you would do would be to create another object just in front but make it invisible. That way when you encounter the invisible trigger the sound will play and then you can take the trigger option on the Door object off, so that it behaves more like a real door would.

Fortunately a lot of this work is already done for you. Notice that the game object **Door** has a triangle next to it. Click the triangle to expose the object underneath it. It's called **Door Trigger** , and as you can see it's currently grayed out. This is because the game object has been deactivated. To activate and deactivate a game object click on it in the Hierarchy and look in the upper left corner of the Inspector View. You should see a blank checkbox. Click the checkbox to activate the object. You'll also notice that this new object is indented and underneath our original Door object in the Hierarchy. In Unity's terms, this object is referred to as a Child , while the object above it is referred to as a Parent . This is the method by which game objects can be organized in the Hierarchy, rather than files and folders in the Project View. In general children inherit behaviors from parents, but children can also be independent of parents if desired.

To see an example of this, click on the Door object, press the W key to change to the Translate Tool and then click on the colored arrows to move the object. You'll notice that when you move the Door object, its child (Door Trigger) will move along with it. However, you can also click on the child and move it independently of the Door object if you desire.

Let's return to the Scene. In the Door object, turn off its Is Trigger function in the Box Collider. This will make it into a solid object. To be safe you should also Remove your trigger script and Audio Source component by clicking on the Gear icon in the upper right-hand corner of the component and selecting Remove Component .

Now what we want is for the Door Trigger object to act like a trigger but not be visible. How do we do this? In this case, what we want to do is to turn off rendering for this particular object so that it remains active but invisible. To do this, find the Door Trigger object and look for the Component called the Mesh Renderer . Deactivate that Component by clicking the check box in the upper left-hand corner.



This makes the game engine not draw the object in question, but its functionality is left intact. Add the trigger script that you created in the earlier steps to the Door Trigger object instead. You'll also have to add an

Audio Source Component as well to complete the setup. Now, start up Game Mode and try it out. You will be able to run into the Door and not pass through it, but the collision sound will still be able to play.

Under the Hood: More Advanced Audio Triggering Methods
The following examples are for more advanced and code-savvy individuals wishing to know more specifically about the various ways that Unity can trigger audio. This is not for beginners, but it gives a more detailed look at the functioning of Unity's API and may be useful when explaining what you want to have happen to a programmer or integrator.

More Methods of Triggering Audio and Scripting Terms in Unity
There are a number of methods to trigger audio in Unity, and some are more useful for certain purposes than others are. Since version 4, a number of methods have been added that are not as well documented. We'll try to give a basic description of the usage cases as well as test examples to try out. Note that all of these methods will work with all versions of Unity past version 5. We'll be using C# for this demonstration as well, for consistency.

This means set up the variable " myAudioSource " as a type of Audio Source. The term **public** makes it visible in the Inspector, otherwise it will be private:

```
public AudioSource myAudioSource;
```

Play the sound (with optional delay time in parentheses (obsolete)):

```
myAudioSource.Play();
```

Any Audio Source must exist as a Component attached to a Game Object, so defining this variable means you would either drop the Game Object with the Source you wanted on it, or specify exactly which GameObject you want to use its Audio Source on.

```
myAudioSource.Play(<optional delay time>);
```

The simplest playback method, and the code equivalent to any Audio Source triggering an AudioClip, as we showed with the ambience loop example earlier. Play() will play whatever audio clip has been selected, either by dropping it directly on the Audio Source Component, or specifying it in code with myAudioSource.Clip , as in the following:

```
Public AudioClip sound;
public AudioSource myAudioSource;
myAudioSource.clip = sound;
loop = true;
myAudioSource.Play();
```

Use Play() ; for continuous sounds like music or background ambiences. It isn't generally good at retriggering sound effects. If Play() is invoked repeatedly on an Audio Source, the sound will interrupt and restart from the beginning. This method is also best for sounds that are moving in the space, or that you need continuous pitch or volume change on. Additionally an audio.Pause() and audio.Stop() command are available.

```
myAudioSource.PlayOneShot(<audioClip>);
```

This method is preferred for any layered sound effects you wish to create. Essentially what happens each time this method is called is that an individual unseen Audio Source is created, plays the sound, and is destroyed. PlayOneShot is great for when you want multiple copies, or

instances, of a sound to exist. It is also important to know that while you can change the pitch or volume of a sound before triggering, it cannot be changed during playback.

```
myAudioSource.PlayClipAtPoint(<AudioClip, Vector3, float>);
```

This method is somewhat similar to audio.PlayOneShot but will create an Audio Source and play the AudioClip at a specified physical location in the game represented by the Vector3 value. This is actually three numbers—for X,Y and Z locations). After playing it will clean itself up by destroying the Audio Source afterwards.

Note the differences here— PlayOneShot requires an Audio Source to be present but PlayClipAtPoint does not. This method also fixes a sound at the location specified (it can't move) and you cannot loop or change pitch of the sound before or after. You can set the volume in this method with the float value, or leave it out if desired.

This method is great for something like a basic explosion sound to go along with a Prefab animation. The animation plays at the same time as the sound effect does, and the Audio Source is destroyed when it finishes playing.

Example - any Game Object can be dragged over the Transform variable. It will use that object's location at that point. For example, this will play the clip specified at the location at volume 0.9 (the f suffix is needed for any float numbers).

```
public Transform someLocation;
public AudioClip mySound;
public AudioSource myAudioSource;
myAudioSource.PlayClipAtPoint(mySound, someLocation, 0.9f);
```

```
myAudioSource.PlayScheduled(<double AudioSettings.dspTime>);
```

This is a new method that allows better much scheduling of an AudioClip, in combination with a separate absolute sample time called AudioSettings.dspTime , expressed as a double (double precision number—basically a very precise number).

This is much more accurate than the older method which is attached to frame rate. This rate can fluctuate depending on a number of factors, which can easily throw the music timing off . This method is independent of that timing and similar to the way that Wwise and FMOD Studio work. It works similar to audio.Play in that it needs an Audio Source and a AudioClip specified.

Example

```
Play a sound exactly 10 seconds later in the game:

AudioSettings.dspTime time;
public AudioSource myAudioSource;
public AudioClip clip;
```

```
clip = myAudioSource.clip;
myAudioSource.PlayScheduled(time + 10.0);
```

There is a lot more material on various ways to trigger audio in Unity, but as this is not a programming or scripting course we can't get into any more specifics here. This is as far as we're going to go for now.

Getting (More) Help

Now that we've just barely covered the basics of audio implementation in Unity, it does leave us with the question of "Where do I learn more about Unity?" If you should decide to learn more about how Unity works in the broader sense or want to start making a game, there are a plethora, a smorgasbord, a dizzying array, a boatload—superlatives fail me—of documentations,

tutorials, and videos available. There are four main sources of Unity knowledge referenced on their own website, under the Learn link:

Tutorials

A new section on the site is the increasingly comprehensive Learn section, with tutorial videos and demo projects to download on subjects like Audio, Editor, Scripting and more.

Documentation

This area consists of the Manual , Component Reference Manual and Scripting Reference Manual . These are all useful, but I find I refer to the Scripting Reference more often, as it has code examples I think are very helpful. These can also be accessed from the Help menu.

Community

This area consists of the Forum, the Unity Answers section, and Feedback section. Of these, the most useful for new developers is the Answers section, where coders from newbies to veterans pose questions that are answered by other members. The Forum and Feedback areas will become useful over time for connecting with other developers and for posting your availability as a composer or sound designer there.

Still not enough? Google "Unity 3D YouTube" and just see how many hits you get. There are literally tens of thousands of videos out there. Also, check the back of the book for all of our helpful links and publications or visit our Game Audio Resources page at the Game Audio Institute website for more info.

Conclusion

We've now come to the end of our time in Unity. We've spent time learning about the specifics of importing audio and the various settings available on Audio Sources. We've also investigated the basics of simple interactivity via scripting in order to trigger sound events, which is a very common thing to do inside games, and we've even covered a few other methods by which sounds can be triggered more accurately.

There's a ton more that you can cover yourself if you're curious by investigating some of the sources we've mentioned. At the next Level we'll be looking at the recent explosion in the mobile, casual and social game markets.