Game Programming Using Unity 3D

Lesson 7: Prefabs

Ferdinand Joseph Fernandez

Chief Technological Officer, Dreamlords Digital Inc.

Admin and Co-founder, Unity Philippines Users Group

September 2011

Overview

Again, we'll be continuing where we left off from the previous lesson. If you haven't done Lesson 6, you need to do it before you can continue here.

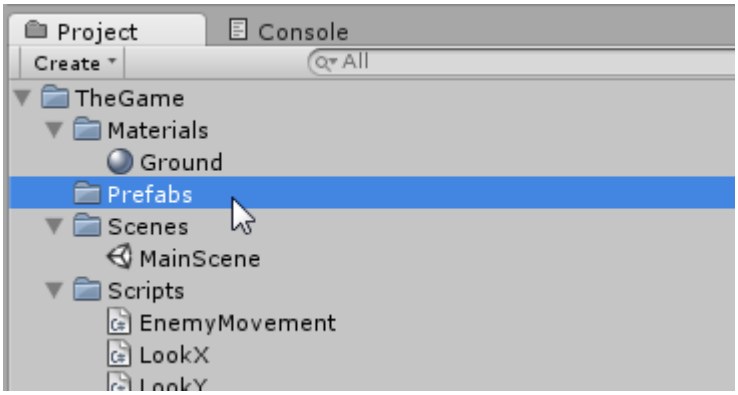
We now have an enemy that chases our player. Now how about making more of that enemy?

Unity features an easy way to create templates out of any of your game objects. Unity calls this a “prefab”.

Think of a prefab as a template, or a master copy of one of your game objects. Its that master copy that you keep on duplicating if you want more copies of it.

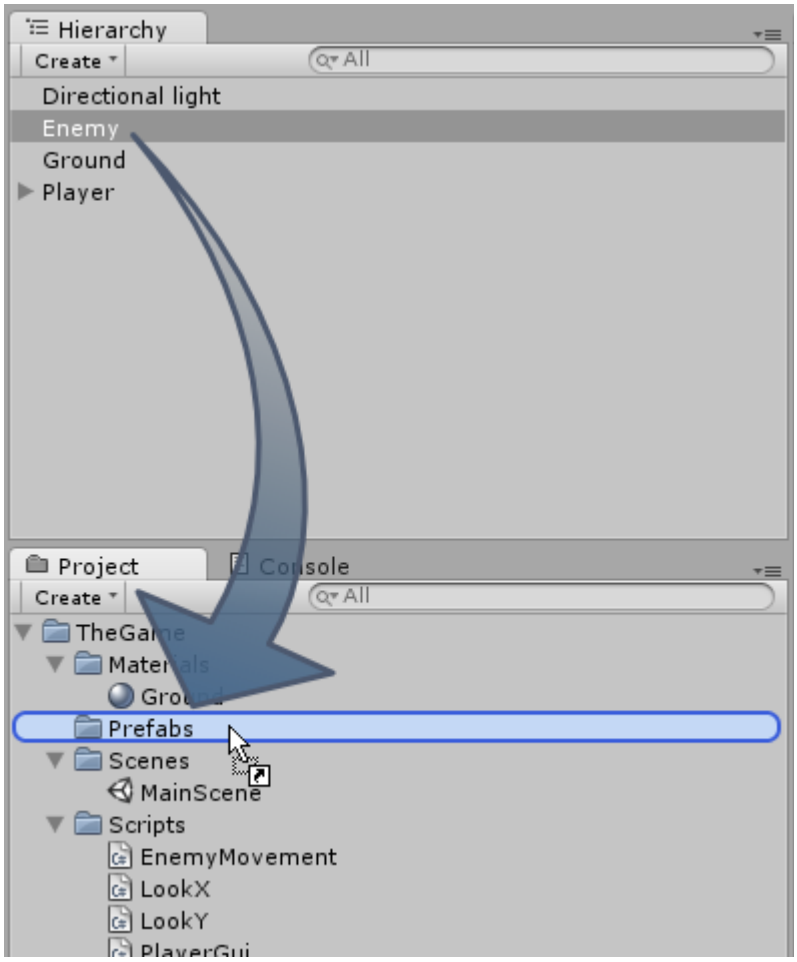
Making A Prefab Folder

Prefabs exist in the Project View. First let's make a folder to store all our prefabs. Inside the “TheGame” folder, make a new folder. Name it “Prefabs”.

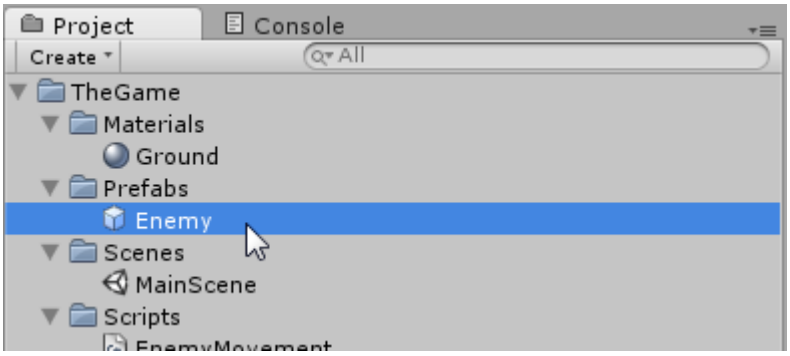


Creating A Prefab In Unity Version 3.4 Or Later

Since Unity 3.4, creating a prefab out of a game object is easy. Drag the “Enemy” game object from the Hierarchy View into the “prefabs” folder in the Project View.



When you drop it there, the “Enemy” prefab will be created. Prefabs are indicated by a blue box icon.



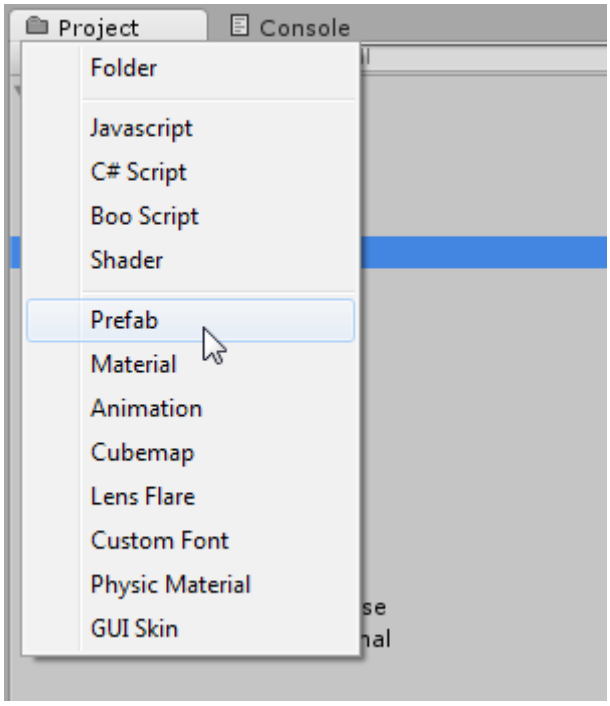
Creating A Prefab In Older Versions Of Unity

If you don't have at least version 3.4 of Unity, creating prefabs is done in a different way.

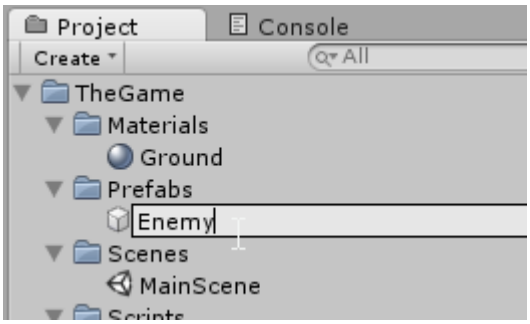
If you already created a prefab using the method mentioned earlier, you may skip this section.

First, go to the Project View. Make sure to select your Prefabs folder.

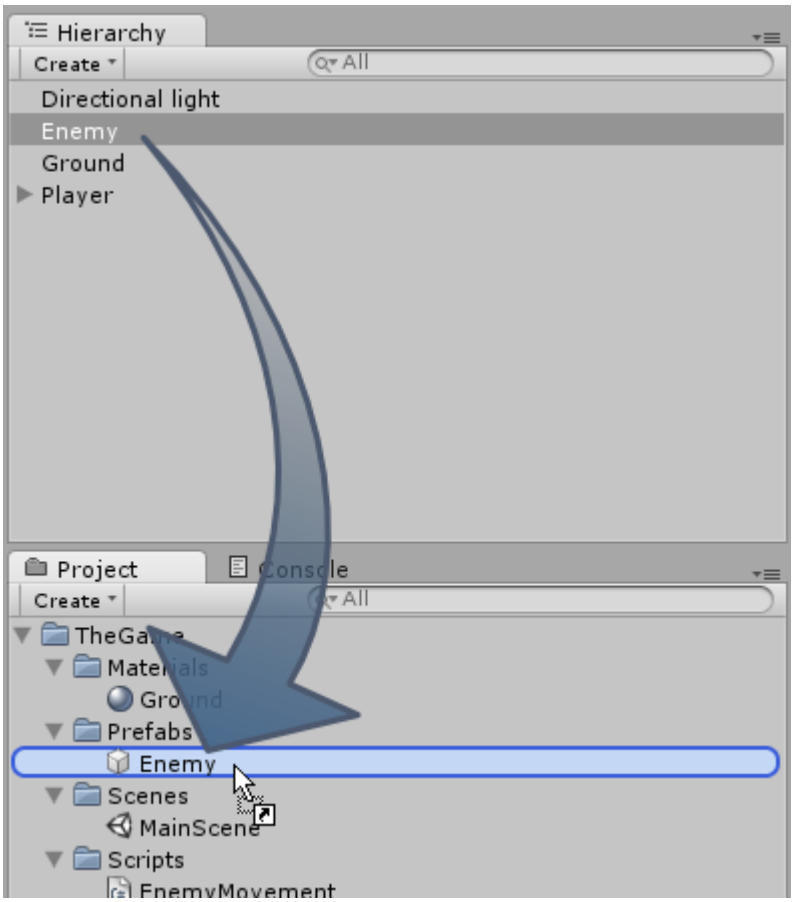
Now, click on the “Create” button, and select Prefab.



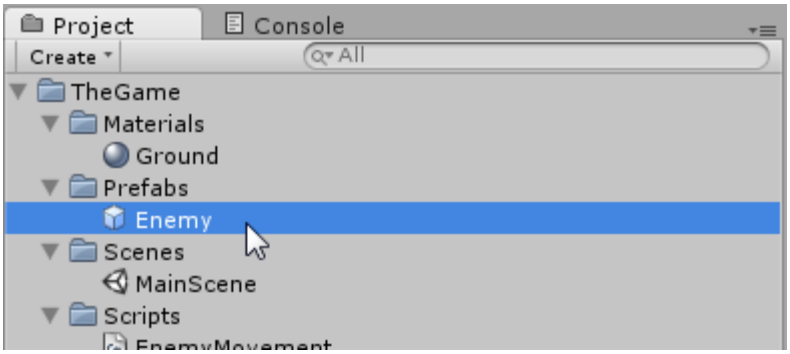
You'll end up with a new empty prefab that you can rename. Name it “Enemy”. Notice the Enemy prefab right now has a gray box as its icon. This means its empty.



Now drag the “Enemy” game object from the Hierarchy View into your empty “Enemy” prefab in the Project View.



When you drop it there, the Enemy prefab will now have a blue box as its icon, indicating it has a value now.



Identifying Prefabs

Now check the Enemy game object that you have on the scene. If you look at its entry in the Hierarchy View, its name is not in blue. This indicates that your Enemy game object is a prefab instance.

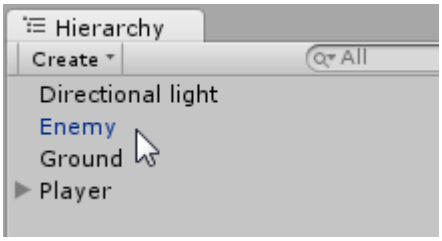


Illustration 1: Prefab instances are indicated by their blue name in the Hierarchy.

Think of a prefab instance as a “live” copy, whereas the prefab is the master copy that doesn't exist anywhere in the scene. A prefab's job is only to be duplicated into live copies to the scene.

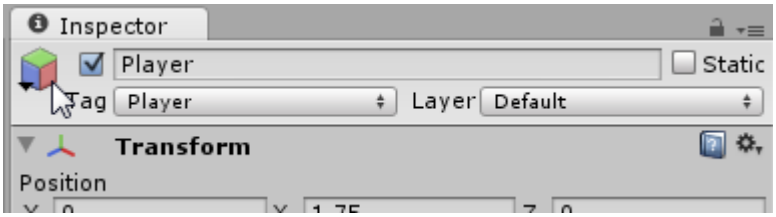


Illustration 2: Notice the icon of a regular game object is a box with red, green, and blue sides.

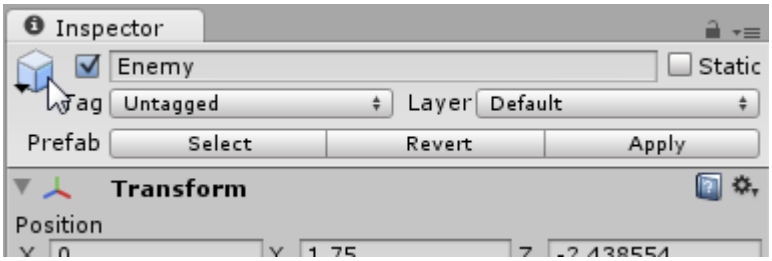
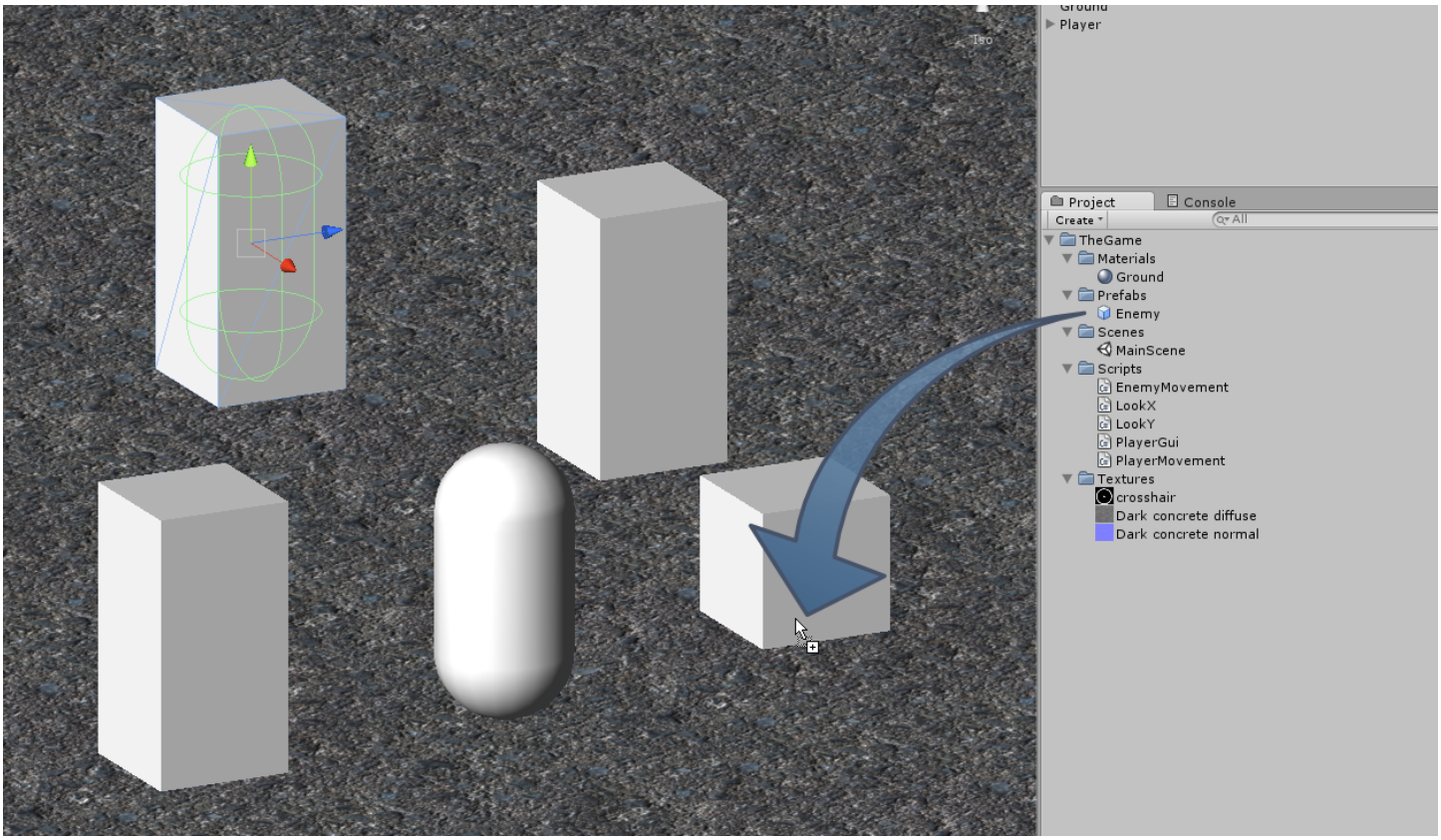


Illustration 3: A game object created from a prefab on the other hand, has an icon of a completely blue box.

Prefabs are still the same game objects that you know. You can add or remove components from them. You can move them and edit their values like usual.

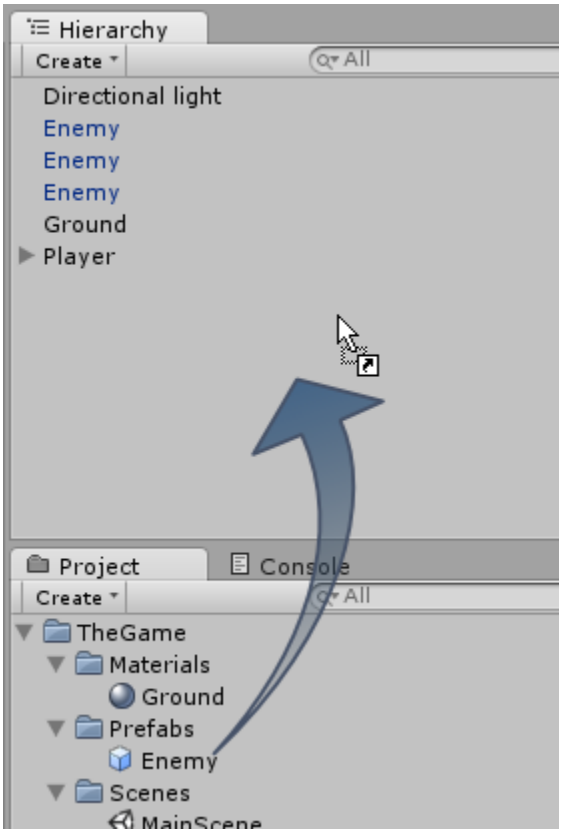
Creating Prefab Instances

To create new instances of a prefab, drag the prefab from the Project View into anywhere in the scene.



Alternatively, you can drag the prefab from the Project View into the Hierarchy View.

Go ahead and drop at least two more enemies onto the scene.



Modifying Prefabs

Now, if we wanted duplicates of our Enemy, actually, we could have just copy-pasted it without bothering with prefabs. Why are we bothering with prefabs then?

The convenience of a prefab is that you can change its master copy, and all live copies will have the changes propagated to them.

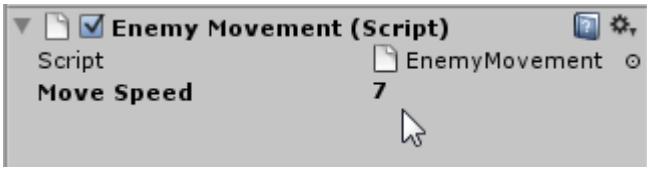
(The most important reason you'll want prefabs is that you can create live copies of a prefab on-the-fly during runtime, but we'll tackle that later.)

Now, we'll try modifying the prefab. Click on the Enemy prefab in your Project View. The Inspector will show the Enemy prefab's properties, just like a regular game object.

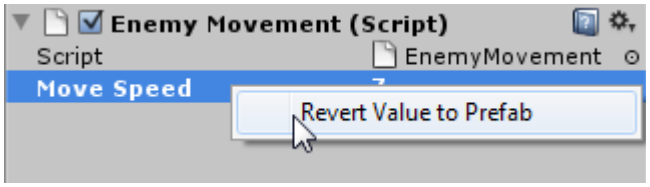
Change the “Move Speed” to 5.25. If you select any of your Enemy game objects in the scene, you'll see it now has 5.25 in its Move Speed also!

Overriding Prefab Values

How about if you want only one of the enemies to have a different speed? You can still edit each Enemy game object individually. When you modify only one of them, the value will show in bold letters. That indicates that you've overriden its default prefab value.

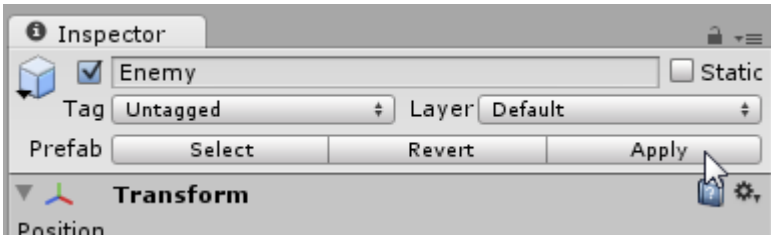


If you ever want to revert a property back to its default value, right-click on it and choose “Revert Value to Prefab”.



Turning An Override Value Into Its Default Value

If you ever decide that the override value you gave to one of your prefab instances should be applied to all copies of that prefab, you can do so by pressing the “Apply” button found at the top of the Inspector.



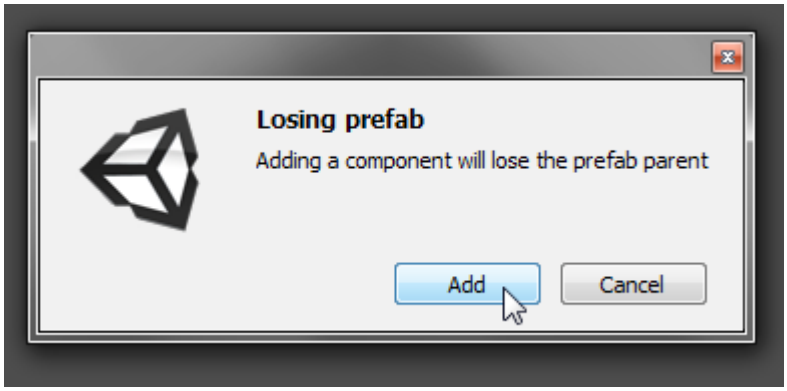
You'll also find other prefab-specific buttons there:

“Select” highlights the prefab (the master copy) in the Project View.

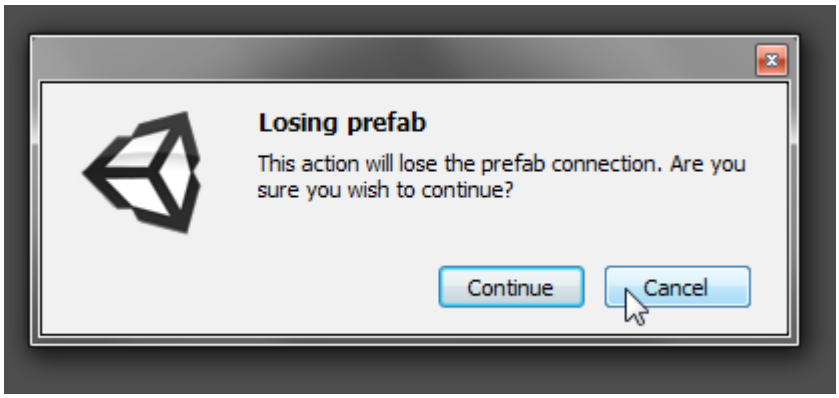
“Revert” reverts all overridden values in all components of that prefab instance to the defaults.

Changing Components In Prefabs

If you add a new component to one of your prefab instances, you'll be warned that it will break the “link” to your prefab.



Also, if you try to remove an existing component from your prefab instance, you'll be greeted with pretty much the same warning:



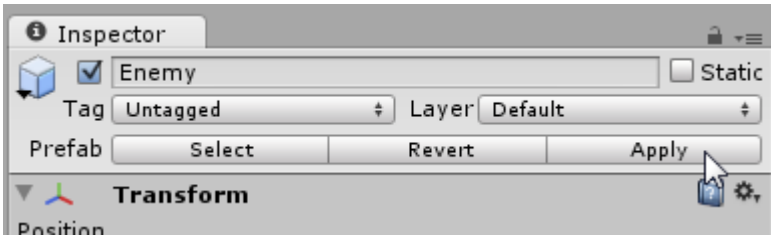
If your intention was to add/remove components to all copies of that prefab, don't worry. Go ahead and continue the process.

Try it. Select one of your enemies in the scene. Add an Audio Source component (Component > Audio > Audio Source) to your Enemy.

Go ahead and click “Add” on the warning message that appears.

What happened is, since you added an Audio Source, your edited prefab instance will then no longer be “linked” to the original prefab. You just need to “link” it back. If you “link” it back, the prefab (master copy) will now also have an Audio Source.

Simply press the “Apply” button in the top part of the Inspector to do so:



When that's done, all copies of that prefab (and the prefab itself) will now have an Audio Source component.

In Conclusion...

This lesson focuses solely on prefabs. You now know how to make and edit prefabs.

In the next lesson, we'll start adding code to allow the player to damage enemies using a gun.