

## Scripting in Unity3D (vers. 3.4)

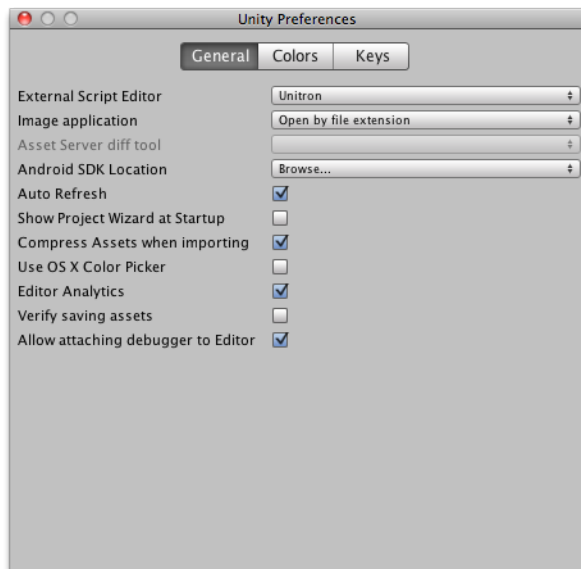
The most basic concepts of scripting in Unity 3D are very well explained in Unity's "Using Scripts" tutorial:

<http://unity3d.com/support/documentation/Manual/Scripting.html>

So please begin this workshop after going through the examples in this workshop first!

The workshop ends after introducing students to the concept of the Update () function in a behavior script, working with variables and attaching scripts to game objects.

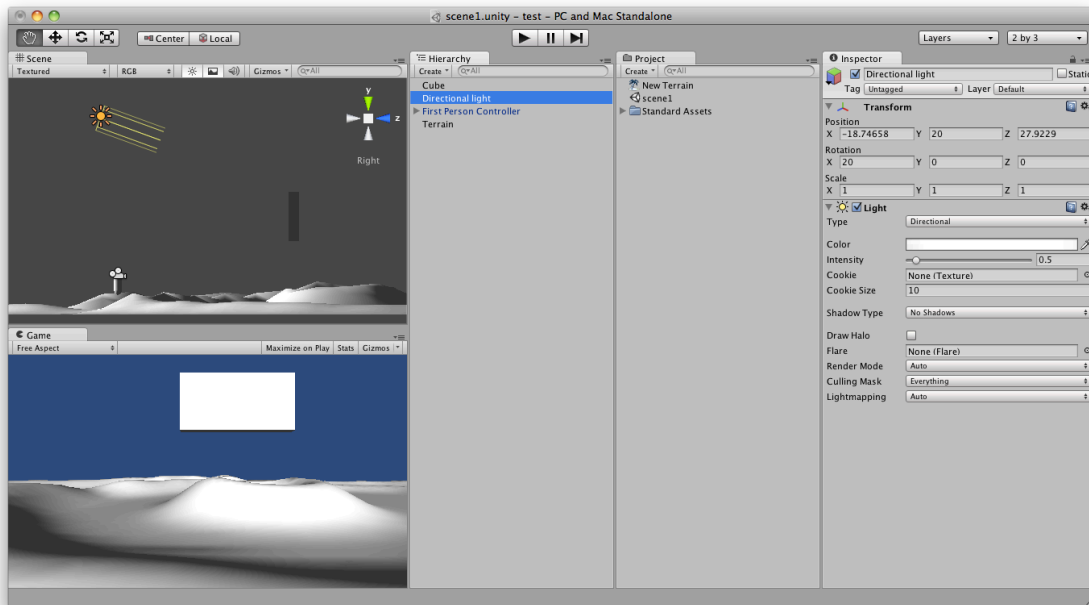
If you are having difficulties launching a proper script editor for your scripts in Unity, go to Unity > Preferences... and in the General panel choose Unitron as your external script editor:



We will start by using the terrain we created in the second part of the first Unity workshop to experiment with 2 things:

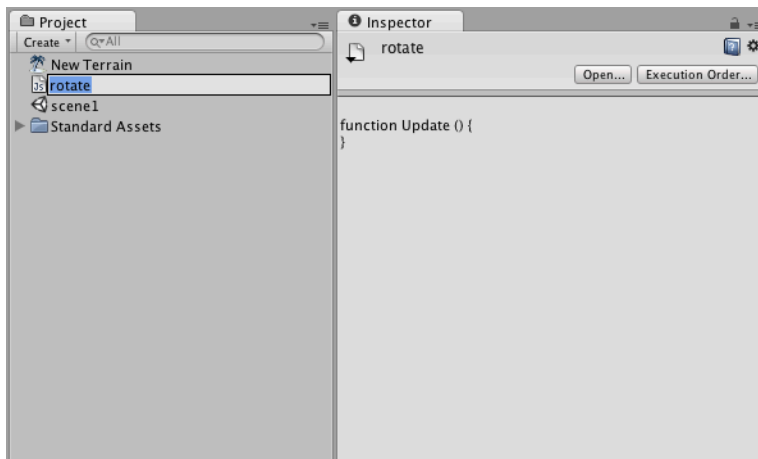
- (1) a simple behavior script that changes the behavior of one game object
- (2) a script that allows us to create a trigger zone by detecting a collision between a game object and the first person controller.

Let's begin with the simple behavior script that rotates a cube using the Update() function. We need a terrain, a first person controller and a cube, so the scene should look something like this (I scaled the cube to make the rotating movement more visible):



Next we will create a new behavior script: Asset > Create > JavaScript

We name this script "rotate" in the Project widow:



You already see the empty Update() function in the Inspector, now double click on the script name in the project window to open it up in the Unitron script editor.

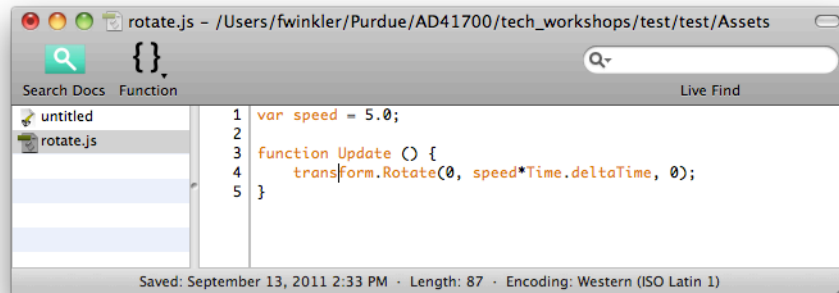
Type in the following script:

```
var speed = 5.0;

function Update () {
    transform.Rotate(0, speed*Time.deltaTime, 0);
}
```

This script updates the Y rotation angle of the game object it is attached to (the cube/box) every time Unity renders a new frame. It is dependent on the time that has passed from the previous frame to the current frame and thus is independent of the frame rate at which your Unity scene will play back (i.e. it won't turn faster on faster computers, only more smoothly).

In the Script editor it should look like this:



```
1 var speed = 5.0;
2
3 function Update () {
4     transform.Rotate(0, speed*Time.deltaTime, 0);
5 }
```

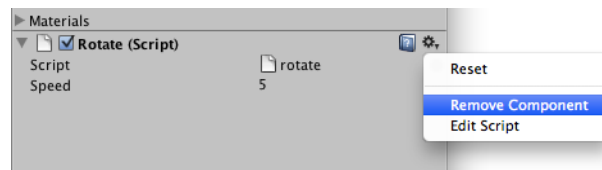
Saved: September 13, 2011 2:33 PM - Length: 87 - Encoding: Western (ISO Latin 1)

Save the script when you close the editor. Then drag and drop the script onto the cube game object in the Hierarchy window. Press the Play button and see the box spin in mid air. Now stop the animation and select the cube that has the script attached to it in the Hierarchy window, notice how in the Inspector the cube game object now has a new property called Rotate (Script). The nice thing about declaring the speed variable previously is that we can change its value interactively in the property inspector without having to open the Unitron script editor.



## Creating Trigger Zones

In the next step, we'll create a trigger zone with the same game object (cube/box). Triggers are useful for triggering other events in your game, like cutscenes, automatic door opening, displaying tutorial messages, etc. For this we need to remove the rotate script and move the box down to the ground of the terrain. To remove a script from a game object, select the game object in the Hierarchy window and then click on the little gear on the top right corner of the script property in the Inspector. Select "remove component" in the pull down menu:



Next we create a new empty script: Asset > Create > JavaScript, name it "trigger\_script", open it up in Unitron, delete the automatically filled in Update(0 function and replace it with the following script:

```

var target : Collider;

function OnTriggerEnter(cubeTrigger : Collider)
{
    if (cubeTrigger == target)

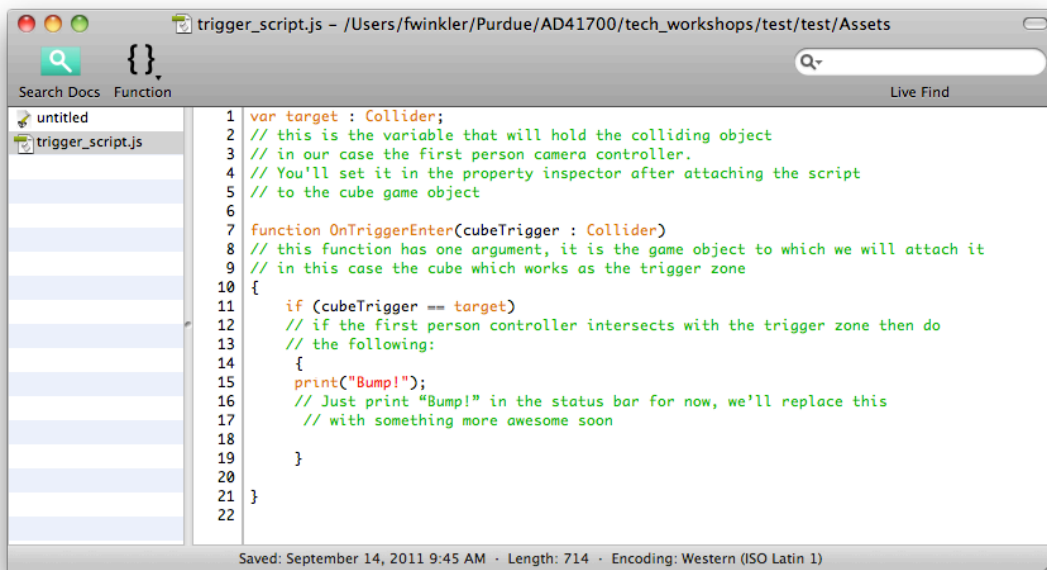
        {
        print("Bump!");
        }

}

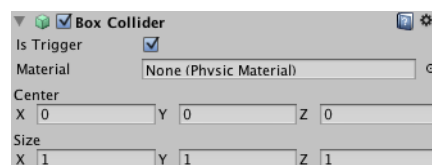
```

This script is doing the following: it checks if the position of the first person controller intersects with the position of the trigger zone (the cube/box game object). If so it simply prints out “Bump!” in Unity’s status bar at the bottom of the screen.

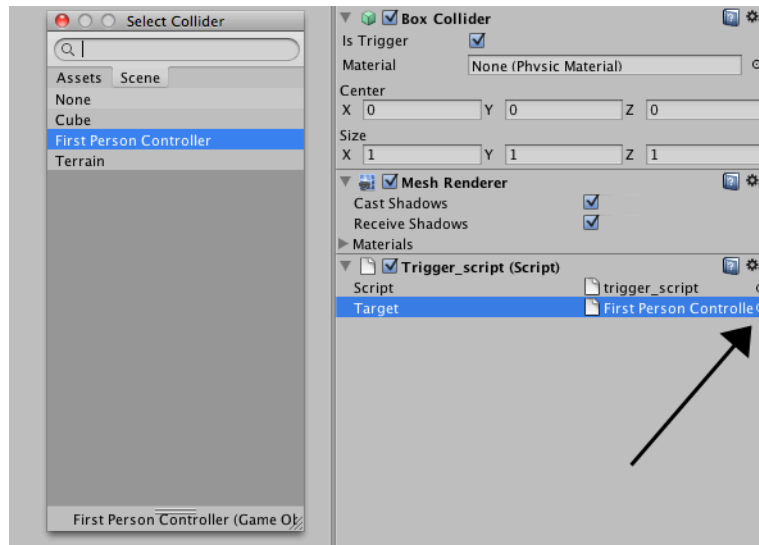
This is what the script looks like in Unitron (note: the green lines are comments):



Now that the script is in place we need to attach it to the game object that we would like to turn into a trigger zone, in this case the cube/box. Take the script in the Property window and drag it onto the cube in the Hierarchy window. For the cube to work as a trigger zone, it is important to select it in the Hierarchy window and then to check the “is Trigger” box in the Box Collider property.



Now, the only thing remaining to do is to set the first person controller to the target variable in the script. We do this by selecting the cube game object (the trigger zone) in the Hierarchy window and navigating to the Trigger\_script (Script) property in the Inspector. Then choose “First Person Controller” from the list next to the “Target” variable:



Now the script can check for collisions between the trigger zone (the game object it is attached to) and the first person controller, the game object that can trigger events by entering the trigger zone.

If you would like to render the trigger zone invisible just uncheck the game object’s “Mesh Renderer” property in the Inspector.

## Counting

Rather than just displaying the same message in the status bar upon a collision, let’s change the script and count the number of collisions that are happening when navigating around in the scene. For this we need a new variable in the trigger\_script. I’ll call it “numberOfHits.”

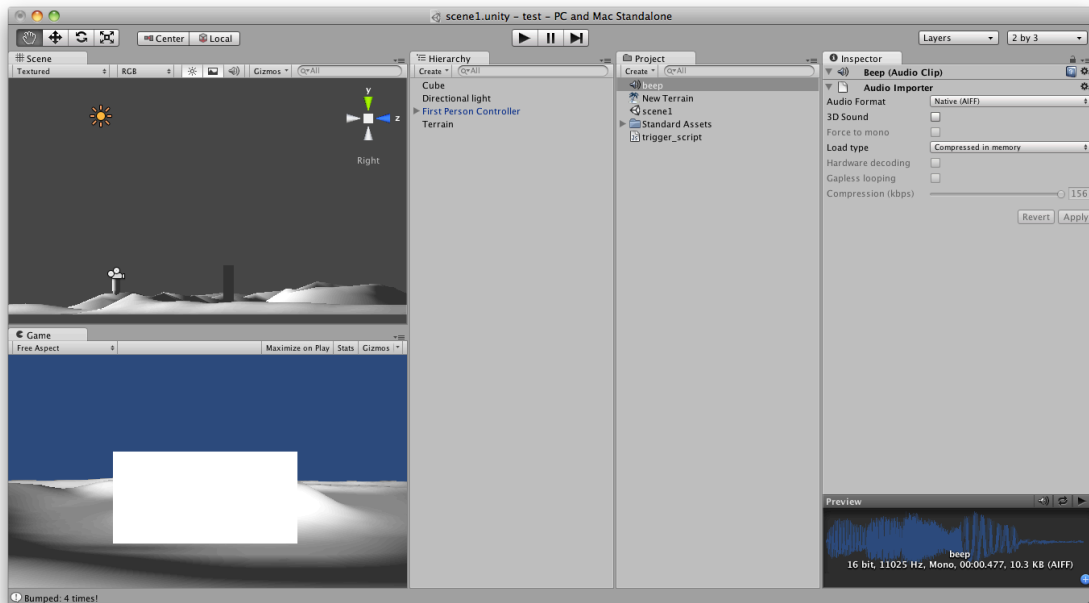
```
var target : Collider;
private var numberOfHits : int = 0;

function OnTriggerEnter(cubeTrigger : Collider)
{
    if (cubeTrigger == target)
    {
        numberOfHits = numberOfHits + 1;
        print("Bumped: " + numberOfHits + " times!");
    }
}
```

Note how declaring the “numberOfHits” variable as private it won’t show up in the Inspector. This script is only triggered upon entering the trigger zone – “OnTriggerEnter” – so we don’t need to worry about multiple counts per visit in the trigger zone.

## Playing Sounds

Next, let's use the trigger zone to play a sound every time we enter it. I downloaded a sample .mp3 file from <http://www.sounddogs.com> converted it in Audacity (<http://audacity.sourceforge.net/>) to AIFF and imported it into Unity as a new asset: Asset > Import New Asset... Unity understand two types of sound files: uncompressed (AIFF or WAV) or compressed (ogg/vorbis). Click on the sound file (mine is named beep) in the Project window to access its properties in the Inspector. Check off the 3D sound option and hit "Apply" in the Inspector window.

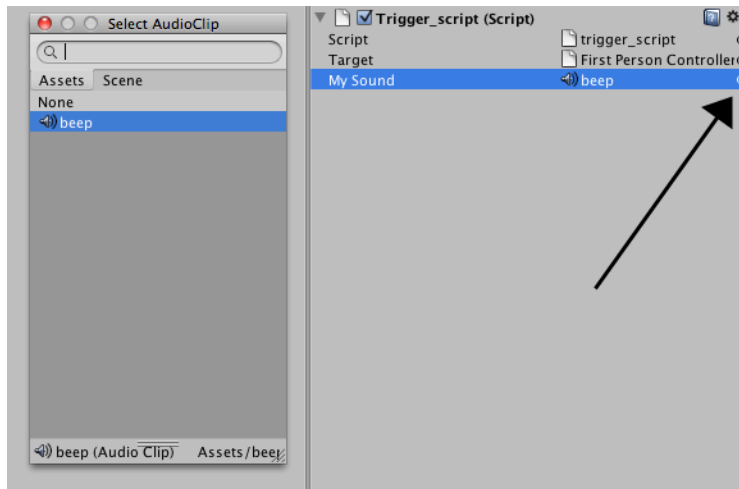


We change the trigger\_script script to include a new audio variable and a line that will play back the sound:

```
var target : Collider;
var mySound : AudioClip;

function OnTriggerEnter(cubeTrigger : Collider)
{
    if (cubeTrigger == target)
    {
        {
            audio.PlayOneShot(mySound);
            print("Bump!");
        }
    }
}
```

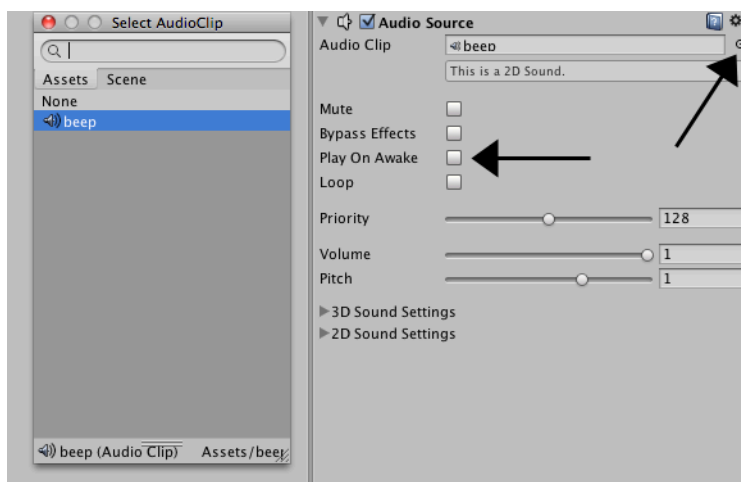
We now have to assign the “Beep” sound file to the mySound variable in the Inspector window. First select the cube in Hierarchy and then go to the Inspector window:



We also need to add an Audio source component to the game object that contains the sound. Select the cube in the Hierarchy and then go to:

Component > Audio > AudioSource

In the Inspector, choose “Beep” as the Audio Clip and uncheck “Play On Awake”



You can now hit the play button and explore the scene. If you would also like to add some background music, simply add an empty game object (Game Object > Create Empty) and then add an Audio source Component to it (keep the empty game object selected in the Hierarchy and then choose Component > Audio > Audio Source. Assign the background sound file in the Inspector and this time make sure the “Play On Awake” box is checked, so the sound loads when the scene loads. Also make sure the “Loop” box is checked for continuous sound playback. You can learn more about audio playback in Unity 3D in Unity’s reference manual at:

<http://unity3d.com/support/documentation/Components/class-AudioSource.html>

## Changing Color

In this example we explore how a script can change the color, first of the game object it is attached to and second of another game object.

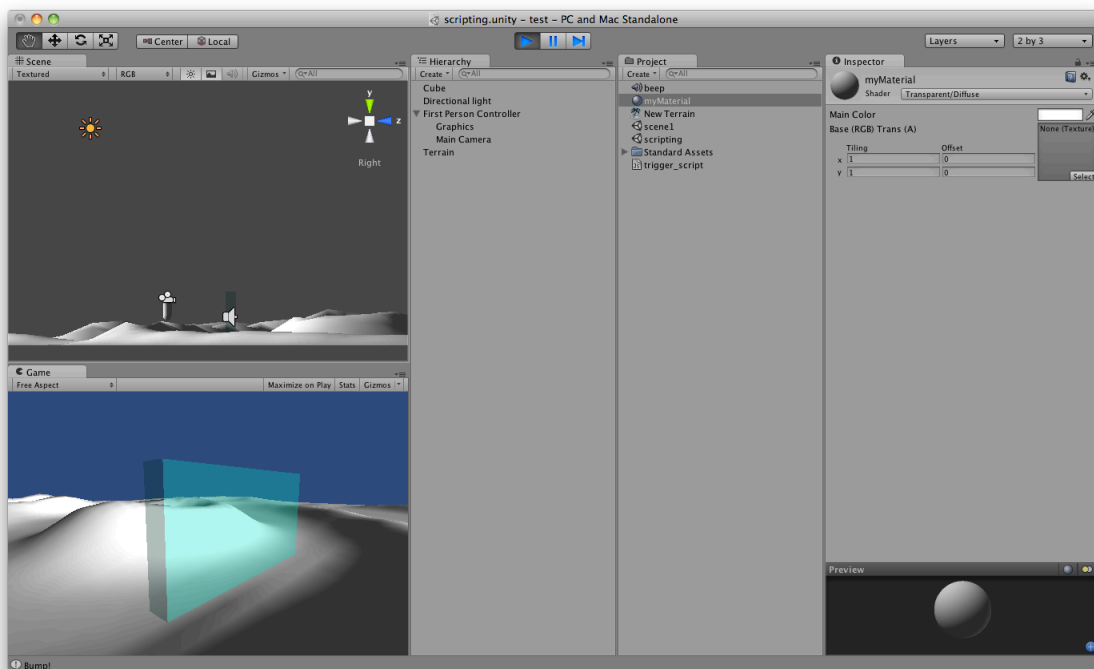
We'll start by creating a plain white material with the option to be rendered transparent (this will allow us to also set the opacity interactively in a script: Assets > Create > Material. Assign this material to the cube (the trigger zone) by dragging it onto the cube game object in the Hierarchy. Next, select the new material (I called it "myMaterial") and set its shader in the Inspector Shader > Transparent > Diffuse.

Now we just have to add a couple of lines to our script:

```
var target : Collider;
var mySound : AudioClip;
private var turquoise : Color = Color(0.0, 0.8, 0.7, 0.3);

function OnTriggerEnter(cubeTrigger : Collider)
{
    if (cubeTrigger == target)
    {
        {
            audio.PlayOneShot(mySound);
            renderer.material.color = turquoise;
            print("Bump!");
        }
    }
}
```

The variable "turquoise" is of the type color and its four arguments are color values for its red, green and blue components as well as its alpha channel (transparency information). When we collide with the trigger zone, its material now changes from a solid white to a transparent turquoise.





## Controlling other Game Object's Components

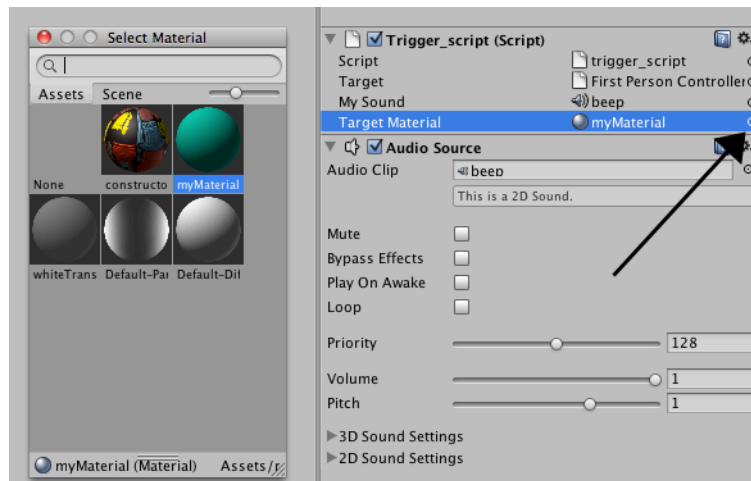
Let's create a second game object behind the trigger zone, so that not the trigger zone changes its material but the primitive behind it: Create a new sphere - Game Object > Create Other > Sphere and place it behind the trigger zone. I also changed the initial transparency of the trigger zone so you can see the sphere behind it but still see the trigger zone as well. For this I created a new material, "whiteTransparent", so I have one material for the trigger zone (whiteTransparent) and one for the sphere (myMaterial).

I'll change my script by adding one more variable that will reference the sphere's Material, called "targetMaterial":

```
var target : Collider;
var mySound : AudioClip;
var targetMaterial : Material;
private var turquoise : Color = Color(0.0, 0.8, 0.7, 1.0);

function OnTriggerEnter(cubeTrigger : Collider)
{
    if (cubeTrigger == target)
    {
        {
            audio.PlayOneShot(mySound);
            targetMaterial.color = turquoise;
            print("Bump!");
        }
    }
}
```

Now we need to assign the right material to "targetMaterial" by selecting "myMaterial" in the Inspector after selecting the cube to which this script is attached to:



Hit the play button and see how the color of the sphere changes when you walk through the trigger zone. You will also see that the color of the sphere is not restored to white once it is changed. To initialize certain components before any other functions of the script are used you can use the Awake() function. In this case we'll use it to always initialize the color of the sphere with white:

```
function Awake () {
    targetMaterial.color = Color.white;
}
```

So the final script looks something like this:

```
var target : Collider;
var mySound : AudioClip;
var targetMaterial : Material;
private var turquoise : Color = Color(0.0, 0.8, 0.7, 1.0);

function Awake () {
    targetMaterial.color = Color.white;
}

function OnTriggerEnter(cubeTrigger : Collider)
{
    if (cubeTrigger == target)

        {
            audio.PlayOneShot(mySound);
            targetMaterial.color = turquoise;
            print("Bump!");
        }
}
```

Another approach would be to use some logic and conditional statements to toggle between two colors whenever you walk through the trigger zone:

```
var target : Collider;
var mySound : AudioClip;
var targetMaterial : Material;
private var turquoise : Color = Color(0.0, 0.8, 0.7, 1.0);

function Awake () {
    targetMaterial.color = Color.white;
}

function OnTriggerEnter(cubeTrigger : Collider)
{
    if (cubeTrigger == target)

        {
            audio.PlayOneShot(mySound);
            if (targetMaterial.color == Color.white){
                targetMaterial.color = turquoise;
            } else {
                targetMaterial.color = Color.white;
            }
        }
    print("Bump!");
}
}
```

Now you are ready for the next tutorial about Instantiation (dynamic creation of game objects) which we will use to create bullets to shoot at things...