

Dice Making in Unity

Part 2: A Beginner's Tutorial Continued

Overview

This is part 2 of a tutorial to create a six sided die that rolls across a surface in Unity. If you haven't looked at part 1, you should probably go back and do that now. In this second part we will look at adding behavior to GameObjects via Javascripts.

This tutorial was made for Unity 3.0 and assumes that the user has at least watched the first few Unity tutorial videos found here: <http://unity3d.com/support/documentation/video/>

This tutorial attempts to be more than a list of steps to follow, I tried to record some of the reasons for why I make choices. If you don't want to follow my logic and just want steps, look at the stuff in bold. At the end of each section is a list of the new Game Objects, Components or Functions covered.

In the last part of the tutorial the dice were created out of Unity primitives and textures, a ground plane and lighting were added, and physics enabled for the dice. In the next part of the tutorial, we will add the scripting to enable the dice values to be visualized through some on screen text.

Determining die value

To determine the value of the side facing up on the die, we'll use a combination of scripting and physics. We need to attach a script to SixSideDie to store the current value of the die. Unity allows us to use "Triggers" for the physics engine to determine when objects touch, but they won't bounce off each other. Triggers can then send messages to scripts which in turn determine behavior.

Before we get started with scripts, let's make that red error message at the bottom of the window go away. When we added the rigid body to the SixSidedDie, Unity computed physics based on the meshes built-in colliders, but because we have multiple meshes parented together, there is an error. We can

make it go away by simply removing the colliders for the side faces. **Select side1 in the Hierarchy, right click on the Mesh Collider Component, and select Remove Component.** Repeat this for each side. The error should be gone. Now we are ready for adding some scripts.

The value of the die needs to be stored in a Component of the SixSideDie, and for this we'll create a new script from the **Project tab Create->JavaScript** and name it "DieValue". We'll add to this later, but here's the script for now:

```
public var currentValue = 0;
```

That's it. **Select the script in the Project tab and click the Edit button in the Inspector.** You can paste the text script text into the editing window and save the script. The script just has a single variable, `currentValue`, which we use to store the current value of the die. Now that the script is in the Project tab, **drag DieValue to the SixSidedDie Game Object** just like any other Component.

To determine the die face value, we need to add a trigger to each side of the die. Think of the trigger as an invisible button on the surface of each side of the die. The trigger can be just a tiny cube with the Mesh Renderer component turned off. When that side rolls around to the ground, the trigger will intersect with the ground and send a message to a script we'll add in this next step. **Add a cube, scale to 0.1,0.1,0.1, move it to 0,0.56,0, deactivate the mesh renderer, and rename it to side1Trigger.** In the Box Collider Component, **check the Is Trigger box. Parent side1Trigger to SixSidedDie** by dragging onto side1 in the Hierarchy. Now repeat for the other six sides.

The trigger needs to talk to a script, for this we'll create a new script from the **Project tab Create->JavaScript** and name it "**SideTrigger**". This is a generic script which gets reused on each trigger. The script has two main parts: a variable to store the value of the side with the trigger, and a function that is called when the trigger is activated. Here's the script we'll use for this purpose:

```
public var faceValue = 0;
```

```
function OnTriggerEnter( other : Collider ) {  
  
    dieGameObject = GameObject.Find("SixSidedDie");  
  
    dieValueComponent = dieGameObject.GetComponent("DieValue");  
  
    dieValueComponent.currentValue = faceValue;  
  
    Debug.Log(faceValue);  
  
}
```

The variable `faceValue` is declared public at the top of the script; when the script is added as a component to the trigger, we can set this to a different value for each face within the Inspector tab. We'll actually give each trigger the value of the face on the opposite side of the die. Go ahead and **drag this script onto each trigger** and set the `faceValue` through the Inspector.

The function `OnTriggerEnter` is a built in Unity function that is called whenever the `GameObject` with this script attached fires a trigger event. There are four things happening inside that function.

The first is that we assign a temporary variable that represents the Game Object that we call `SixSidedDie`. Unity allows searching for Game Objects by name using the `GameObject.Find()` function. That's handy because we need to find the `DieValue Component` that we just added to `SixSidedDie`.

Like searching for Game Objects, we can search for Components by name, but we need to use the variable for the particular Game Object we're searching from, hence the `dieGameObject.GetComponent` function on the functions second line. This function returns a variable which is the scripting representation of the Component which is itself a script.

The third line sets the `currentValue` variable inside the `DieValue Component`. The dot after the variable name indicates we want to access a variable within that component, so we just set it equal to the `faceValue` for this trigger.

The last line is for convenience. `Debug.Log` is very helpful when you start scripting; it will print messages out in a special Unity window called the console. **Access the console window by clicking Window -> Console**. Run your project and watch the numbers get printed as your die rolls around.

Components Used in this Section:

JavaScript

Functions Used in this Section:

OnTriggerEnter() -

<http://unity3d.com/support/documentation/ScriptReference/Collider.OnTriggerEnter.html>

GameObject.Find() -

<http://unity3d.com/support/documentation/ScriptReference/GameObject.Find.html>

GetComponent() -

<http://unity3d.com/support/documentation/ScriptReference/GameObject.GetComponent.html>

Debug.Log() - <http://unity3d.com/support/documentation/ScriptReference/Debug.Log.html>

Making values visible

Our last task is to make our die value visible using a Game Object called TextMesh. TextMesh is a 3D text object in Unity that is very useful for things like GUIs and messages to users. So that the TextMesh knows what value is on our die, we'll update the DieValue script.

Add the TextMesh by clicking **GameObject -> CreateOther -> 3D Text** and transform it to **6,-1.5,-4**.

Rename the TextMesh to DieText. The TextMesh will come into the scene aligned perpendicular to the Z axis, which is not where we can see it, so **rotate 90,0,0** to orient it correctly. Rotate the camera so the game view matches the scene view.

In the Inspector tab you'll see many font settings, we won't mess with those, but notice that there is one item called text. If you change this value, it will change the value on the screen. This is the value to update with the die information.

Our last step is to update the DieValue script to look for the DieText GameObject and update the Text value of the Text Mesh Component. **Edit the DieValue script** to look like this:

```
public var currentValue = 0;

function Update() {
    dieTextGameObject = GameObject.Find("DieText");
    textMeshComponent = dieTextGameObject.GetComponent("TextMesh");
    textMeshComponent.text = currentValue.ToString();
}
```

The Update() function is another built in Unity function; it is called every time the scene is rendered (more or less). Inside the update function, the first two lines are similar to the Game Object and Component search from the SideTrigger script, but with the goal of finding the Text Mesh. Once the textMeshComponent variable is assigned, we just set the text value to the currentValue for the die. But there is one last trick here, text is a variable of type string, and currentValue is implicitly an integer, so we use the ToString() function to change currentValue into a string. Try taking away the .ToString() part of the last line to see what happens.

Function Used in this Section:

Update() - <http://unity3d.com/support/documentation/ScriptReference/MonoBehaviour.Update.html>

ToString() -

Keeping the die on the board

If you toss the dice around a bit, you will notice that the die can disappear off screen if thrown with too much speed. To solve this problem, we could use a script to constrain the transform property of the SixSidedDie, or we can use the physics engine. To use Unity's physics engine for this task, we can add

four walls to our scene, one to block the bottom, one to block the top, one for the left and one for the right. Cubes work easily enough for this task, just make sure to scale them high enough to constrain the movement of the die, and turn off the Mesh Renderer Component to make it invisible.

End

These steps take you most of the way to a usable dice for your game, but there is room for further improvement. For instance, adding an additional die; for most games you will want two or more die. The FaceTrigger and DieValue scripts currently make that difficult because the object names are coded directly into the script. Changing those scripts to use a variable for the Game Object names would be a big improvement.

A plain white die could be a little boring. Changing the color is certainly an option, or experiment with other materials. Watch out if you have multiple dice that need to be different colors: Materials are shared between objects, so new Materials are necessary for different colored objects.

The interaction with the die is a little hacky and not very useful. How could it be improved? Is scripting required? Is there a physics hack?

That's it for our super basic die tutorial. We covered many basic Unity concepts but there is much more depth and breadth to the game engine. In addition to many more Game Objects not covered in this tutorial, scripting will be key to making your games function. Check out the scripting reference at Unity3d.com for much more information:

<http://unity3d.com/support/documentation/ScriptReference/index.html>

initial part 1 - 1/11/2011 - Jonathan Cecil . jonathancecil@ucla.edu