David Sushil davidjsushil@gmail.com

Creating Triggers in Torque

Triggers are invisible structures in the Torque engine which have the ability to execute code. You can create triggers using several methods, including by using the World Editor Creator feature in the Torque mission editor. Perhaps the easiest method is through the use of code, which we will explore in this thankfully brief tutorial.

Adding Triggers

To add a trigger through code, create a .cs file and add and modify the following code.

```
datablock TriggerData(FilingCabinetTrigger){
      tickPeriodMS = 1000;
};

function addTrigger(){
      %obj = new Trigger() {
            position = "100 100 100";
            scale =     "1 1 1";
            rotation = "0 0 1 0";
            dataBlock = "FilingCabinetTrigger";
            polyhedron = "0 0 0 1 0 0 0 -1 0 0 0 1";
      };
}
```

As you can see, the first step is to create a datablock called TriggerData. It takes one argument, which will be the name of the parent trigger. In this case, we're calling it FilingCabinetTrigger.

The datablock has one parameter, labeled tickPeriodMS. This indicates how often the datablock executes code. We've assigned it a value of 1000, meaning the trigger will be executed every second. By the way, the smaller this number, the greater its strain on your CPU. Deciding on a reasonable interval is important.

Next, we have a function called addTrigger, which, as you might have guessed, creates an instance of FilingCabinetTrigger.

The first line creates a local variable called %obj, and assigns it the ID of a new Trigger. This new trigger has several parameters of importance. First, its position is where it appears in the mission. It's scale is also reported, as is its rotation. These three properties should be familiar to you, having previously worked with static shapes.

%obj will take on the properties of the FilingCabinetTrigger outlined above, by assigning it to the dataBlock parameter.

Finally, a property called polyhedron creates the shape of the trigger. It is currently set up to represent a box, although additional shapes are possible. To date, I have not found any

resource that explains how this polyhedron parameter works. I tend to manipulate the trigger's scale in order to create the proper size and shape.

By running the addTrigger() function from the console, we can add an instance of the FilingCabinetTrigger to our mission.

Basic Trigger Functions

Next, we need to make it do stuff. The following code offers an example:

```
function FilingCabinetTrigger::onEnterTrigger( %this, %trigger, %obj )
{
      $CabinetInRange = true;
      echo("In range.");
}


function FilingCabinetTrigger::onLeaveTrigger( %this, %trigger, %obj )
{
      $CabinetInRange = false;
      echo("Out of range.");
}
```

This code obviously doesn't do much, other than offering you a framework to build on. Every trigger has two important functions built into it – onEnterTrigger() and onLeaveTrigger(). As you might expect, onEnterTrigger() is called whenever the player moves into the space the trigger occupies. onLeaveTrigger() is called whenever the player moves out of the space the trigger occupies. Remember, we set the tickPeriodMS to 1000, which means these functions will be called only once a second, and only if their assumptions are met. There is a third method, onTickTrigger(), which we will address later.

In this case, when the player enters the trigger, Torque will set the value of a global variable called CabinetInRange to true. It will also print "In range" in the console. Likewise, when the player exits the trigger area, CabinetInRange will revert to false, and the Torque will print "Out of range" in the console.

Notice that these functions take three arguments, which are supplied by Torque. These arguments are %this, %trigger, and %obj, which are essentially the following objects:

Variable	What It Represents
%this	The datablock associated with the trigger.
%trigger	The trigger which executed the function – the one we created in code.
%obj	The object which moved into (or out of) the trigger area – it could be the player, for example.

Now, in order to make this code work effectively, we need to execute the .cs file in which it exists, either through the console, or by placing it in your game.cs file. Either way, the code to run this is as follows:

```
exec("demo/data/shapes/filing-cabinet/filing-cabinet.cs");
addFilingCabinet();
```

Of course, you will need to change the first line to reflect the name and location of the .cs file you have created. You should also add a static shape in roughly the same spot as your trigger.

Animation Control Via Triggers

If you wanted to control a shape's animation using triggers, we need to take a few extra steps. First, we need to create an animated object. Let's assume we've made a filing cabinet, which will open and close whenever the player enters and leaves the trigger area. We might write code like this (and for your sake, I'll boldface what I've changed from the previous examples:

```
datablock StaticShapeData(FilingCabinet){
      shapeFile = "~/data/shapes/filing-cabinet/filing-cabinet.dts";
};

datablock TriggerData(FilingCabinetTrigger){
      tickPeriodMS = 1000;
};

function addTrigger(){
      %FilingCabinetObj = new StaticShape(){
            position = "100 100 100";
            scale =     "1 1 1";
            rotation = "0 0 1 0";
            datablock = FilingCabinet;
      };

      %obj = new Trigger() {
            position = "100 100 100";
            scale =     "1 1 1";
            rotation = "0 0 1 0";
            dataBlock = "FilingCabinetTrigger";
            polyhedron = "0 0 0 1 0 0 0 -1 0 0 0 1";
            cabinetID = %FilingCabinetObj;
      };
}
```

This new code should not be unfamiliar to you at this point. We create a static shape datablock called FilingCabinet, and then create an instance of that datablock in the addTrigger() function. Notice that we are storing the ID of the static shape in a local variable called %FilingCabinetObj, which we then assign to our trigger's cabinetID property. In this way, we can modify our onEnterTrigger() and onLeaveTrigger() functions to control the static shape's animation, like so:

```
function FilingCabinetTrigger::onEnterTrigger( %this, %trigger, %obj ){
      $CabinetInRange = true;
      echo("In range.");
      openCabinet(%trigger.cabinetID);
}

function FilingCabinetTrigger::onLeaveTrigger( %this, %trigger, %obj ){
      $CabinetInRange = false;
      echo("Out of range.");
      closeCabinet(%trigger.cabinetID);
}

function openCabinet(%id){
   %id.playThread(0, "open");
}


function closeCabinet(%id){
   %id.playThread(0, "close");
}
```

Voila.

An Additional Function

Finally, let's look at an addition trigger function, onTickTrigger(), which checks to see if something is inside the trigger area. This function accepts the same three arguments as onEnterTrigger() and onLeaveTrigger(), but will be executed not just once, but every time the tickPeriodMS timer is reset. This could be a good way of assigning damage to a player who has fallen into a pit of lava, for example.