Scripting Tutorial – Lesson 14: (Advanced) Using Keyboard Controls with Multiple Classes

Download supporting files for this tutorial

Texas Instruments TI-Nspire Scripting Support Page

And now the fun begins!

By lesson 13 we had learned how to define a class and to control movement of such objects using both keyboard and mouse controls. Not much fun with just a single class. So now it is time to work with multiple class objects.

First we need another class to work with. This is a good excuse for a little review and revision of where we have been recently.

You will see below the code to create a Circle class to complement our existing Square. It should be sensible and even familiar, if you took my advice and tried to create this yourself at the end of Lesson 11.

Things to study closely: look at the way that the **contains** function is defined here: quite different from that of the Square, but perfectly sensible for a circle. In fact, this is a fair bit simpler than that for the Square.

As with the Square, I have defined a generic Circle – one which could just as easily be an ellipse, just as the Square could easily be turned into a Rectangle, if desired. Makes life easier later on if you need to vary from the original simple plan. This is a good rule of thumb for any programming (or, indeed, document creation) that you do: try to avoid creating **specific** instances when, with just

a little more time and effort, you could create a **general** case. For a quiz document, for example, avoid entering specific questions if, with a little use of a random function, you can have the software generate multiple questions. Much more powerful and useful as a learning tool.

Finally, review the syntax for the **drawArc** and **fillArc** commands that are used here. Note that this is, in fact, a much more general command than just a "circle creator". The circle is just a specific instance that occurs when the arc extends from 0 to 360 degrees (or perhaps from -180 to 180? Think about this!).

We now arrive at the heart of this lesson: a simple way to work with multiple objects is to create a table which contains them! Then to move from object to object, we just step through the table. I hope, like me, that you can appreciate the elegance of this approach.

```
Circle = class()

function Circle:init(x, y, width, height)
	self.x = x
	self.y = y
	self.width = width * 2
	self.height = height * 2
	self.radius = height
	self.color = Color.red
	self.selected = false
end

function Circle:contains(x, y)
	local r = self.radius
	local d = math.sqrt((self.x - x)^2 + (self.y - y)^2)
	return d <= r
end

function Circle:paint(gc)
	local cx = self.x - self.radius
	local cy = self.y - self.radius
	local diameter = 2*self.radius
	gc:setColorRGB(unpack(self.color))
	gc:fillArc(cx, cy, diameter, diameter, 0, 360)
	if self.selected then
		gc:setPen("medium","smooth")
		gc:setColorRGB(0, 0, 0)
		gc:drawArc(cx, cy, diameter, diameter, 0,
		360)
	end
end
```

```
Objects = {
	Circle(W/2, H/2, W/20, W/20),
	Square(W/2, H/2, W/10, W/10),
}
```

In this case, we create a table called **Objects** which consists, at present, of a Square and a Circle. It might be about now that the importance of the variable **TrackedObject** might become apparent. Whatever is the currently selected member of the Objects group becomes the TrackedObject. We just need a mechanism to move from one object to the next – and that will be our **tabKey** for the keyboard, or mouseDown for the mouse controls.

This Objects table should be defined up in the **on.resize** function. You will notice that I actually use it to define my Square and my Circle, doing away with the intermediate variable "Sq" that I used previously. This will mean a little rewrite of some of our previous code, but will be worth it.

The best way to see how this process will work is by redefining our **tabKey** function to take advantage of it, as shown

here. You will see the new code in **bold**. The rest of the code, using TrackedObject, remains just fine.

Perhaps the syntax looks a little strange, but still we are looking at a simple **for** loop which steps through the objects in the Objects table.

What is that -1 in the first line of the loop? The loop commences with the number of objects in the Objects table (#Objects) and then steps backwards (-1) to the first object. Nice.

The effect of pressing TAB here is to move focus from one object in the table to the next object. The **break** serves to stop that continuing indefinitely - it moves forward once, and then stops until the next press of the tabKey.

We can then update our **on.paint** function to use the Objects table rather than the specific global variable Sq, as previously. You will see that this **for** loop uses the dummy variable "_" to count through

---

```
function on.tabKey()
    if TrackedObject ~= nil then
        TrackedObject.selected = false
    end
    for i = #Objects, 1, -1 do
        local obj = Objects[i]
        if obj == TrackedObject then
            TrackedObject = Objects[i+1]
            break
        end
    end
    if TrackedObject == nil then
        TrackedObject = Objects[1]
    end
    TrackedObject.selected = true
    platform.window:invalidate()
end
```

```
function on.paint(gc)
    for _, obj in ipairs(Objects) do
        obj:paint(gc)
    end
    if TrackedObject ~= nil then
        gc:setFont("sansserif", "b", 10)
        gc:setColorRGB(unpack(TrackedObject.color))
        gc:drawString("("..TrackedObject.x..","
        ..TrackedObject.y..")",20,20)
    end
end
```

the Objects table in "ipairs", which means counting an index as well as each object (obj[1], obj[2], etc). Finally, we replace the Sq reference in the last drawString commands so that the coordinate position displayed is that for the TrackedObject, and it will be displayed in that object's color. Pretty cool.

These changes should now make it possible to use keyboard controls to tab from object to object, and to move the selected object around. What remains is to update the **mouseDown** function in a similar way to the tabKey so that our mouse controls will also work with multiple classes.

In our final lesson for this sequence we will see how this is done, and finish this introduction to classes and to mutliple controls.

Back to Top

Home ← TI-Nspire Authoring ← TI-Nspire Scripting HQ ← **Scripting Tutorial - Lesson 14**