Scripting Tutorial - Lesson 12: Advanced: Mouse Control

Download supporting files for this tutorial

Texas Instruments TI-Nspire Scripting Support Page

In lesson 11 we introduced the powerful promise of classes within Lua, making so much more possible. Perhaps the most useful of these new possibilities is the control of screen objects using the mouse in addition to the keyboard controls that we have already studied.

We begin with a little housekeeping. I always like to define variables for my window width and height to make it easy to refer to these later. You might have noticed that I am adopting a convention where global variables are Capitalized, while local variables are lowercase.

More importantly, I begin by introducing a new variable, called TrackedObject, which is initially nil, but will make it easy for us to follow things, especially when we are trying to track more than one object. And we can define our size and starting position a little more precisely.

```
W = platform.window:width()
H = platform.window:height()
TrackedObject = nil
Sq = Square(W/2, H/2, W/10, W/10)
```

Begin by defining the effect of mouse "clicks". Each "click", of course, is actually two actions - a mouse down and a mouse up. Both begin with a simple check - if TrackedObject exists, then set its selected state to off. This takes account of whatever state it happens to be in when you click on it.

In the case of mouse down, then, the effect is to turn the selected state to on, and to make our object the "TrackedObject" - as long as we click INSIDE the object! Here is that lovely "contains" function hard at work.

Finally, we refresh the screen with a platform.window:invalidate() command. When using mouse actions, this forces screen refreshes whenever we use the mouse, so there is no need for other methods (like the timer we have used previously).

```
function on.mouseDown(x,y)
    if Sq:contains(x, y) then
        if TrackedObject ~= nil then
            TrackedObject.selected
            = false
        end
        TrackedObject = Sq
        Sq.selected = true
        platform.window:invalidate()
    end
end
```

The other two mouse actions are very simple. Mouse up simply releases the object. Turns selected off, and sets TrackedObject back to nil.

```
function on.mouseUp(x,y)
    if TrackedObject ~= nil then
        TrackedObject.selected = false
    end
    TrackedObject = nil
    platform.window:invalidate()
end
```

Finally, the marvellous mouse**Move** function. If TrackedObject exists (which it does if we have our mouse down on our object) then the position variables for this object are set to be the same as those of the mouse. Easy!

```
function on.mouseMove(x,y)
    if TrackedObject ~= nil then
```

A nice finishing touch for this lesson might be to display the coordinates of our TrackedObject as we move it around the screen. Just add the following lines to **on.paint**:

```
        if TrackedObject ~= nil then
            gc:setFont("sansserif", "b", 10)
            gc:setColorRGB(unpack(Sq.color))
            gc:drawString(
            "("..TrackedObject.x..","
            ..TrackedObject.y..")",20,20)
        end
                TrackedObject.x = x
                TrackedObject.y = y
                platform.window:invalidate()
            end
        end
```

In our next lesson we will review how to control such an object using keyboard commands.

Back to Top
