TI-nspire™

Scripting Tutorial - Lesson 13: (Advanced) Classes and Keyboard Control

Download supporting files for this tutorial

Texas Instruments TI-Nspire Scripting Support Page

In lesson 12 we saw that defining classes can make something like mouse control of screen objects surprisingly easy. Well, it gets better. Classes tend to make just about everything easier. Since a class object knows where it is on the screen (and, in fact, where other objects are as well) then setting up different control mechanisms tends to be pretty straightforward.

As advertized, in this lesson you will learn how to add arrow keys and other keyboard controls to move our Square around. First, however, a little more housekeeping.

In setting up our document so far, we have defined a bunch of things, some functions, and some global variables (like Color, W, H and even Sq).

Generally, if you need to rely that such global variables exist so that you can call them as required, then you should take steps to ensure this. Usually this means placing them into a function that *must* get called at some point, like **on.create**. Obviously this will always get called at the very beginning (which is good) and that is generally enough when using the handheld.

But we are setting these up to also work on the computer, and in this context the page may get *resized* at any point. If this happens, then we need to recalculate our platform.window:width and height values. If these are defined only when the page is created, they will not be very effective.

A better solution, then, might be to use **on.resize**. this will be called when the page is first created, AND will get called again if the page dimensions change at any point - perfect!

Now we can begin this lesson!

```
function on.resize()
    Color = {
        red = {0xFF, 0x00, 0x00},
        green = {0x00, 0xFF, 0x00},
    }
    W = platform.window:width()
    H = platform.window:height()
    TrackedObject = nil
    Sq = Square(W/2, H/2, W/10, W/10)
end
```

Have a look at the code opposite and pinch yourself if it actually makes sense to you: who are you really, and what have you done with the person who started these tutorials not so long ago?

Hopefully, for some of you at least, what would have been a jumble not so long ago now actually makes some sense.

The only thing that looks different from the arrow controls that we learned to use way back in tutorial 6 should be the use of this **TrackedObject** variable. But once you realize that this is just a way to refer to the currently selected thing (the Square in our case) then you will see that the up and down, and left and right arrows are doing much as we expect them to.

You might want to double-check the conditions imposed here - you should see that these simply stop you running off the visible screen. And, of course, the leading condition for each function - if **TrackedObject** is not **nil**.

This actually raises a very important issue that I have stepped around so far. Copy and paste these lines of code into the script from the previous lesson, adding

```
function on.arrowRight()
    if TrackedObject ~= nil then
        if TrackedObject.x < W - TrackedObject.width / 2 then
            TrackedObject.x = TrackedObject.x + 5
        end
    end
end
function on.arrowLeft()
    if TrackedObject ~= nil then
        if TrackedObject.x > TrackedObject.width / 2 then
```

keyboard commands to your mouse control. What happens?

Well, nothing happens - yet. And the reason is because using just our keyboard, we currently have no way to select our object. With mouse control, we just click.

For such things, I like to use the TAB key. This makes even more sense when we have multiple objects from which to select. In such cases, then pressing tab should jump from object to object - and we will see how to do that shortly. First, study how we define our **tabKey** actions.

```
function on.tabKey()
    if TrackedObject ~= nil then
        TrackedObject.selected = false
    end
    Sq.selected = true
    platform.window:invalidate()
end
```

So what about letting go?

Hopefully you will see that this is a simple but important effect: pressing escape *releases* the currently selected object. Once it is pressed, then the current TrackedObject goes back to being nil.

```
function on.escapeKey()
    if TrackedObject ~= nil then
        TrackedObject.selected= false
        TrackedObject = nil
    end
end
```

```
                TrackedObject.x = TrackedObject.x - 5
            end
        end
    end

function on.arrowDown()
    if TrackedObject ~= nil then
        if TrackedObject.y < H - TrackedObject.height / 2 then
            TrackedObject.y = TrackedObject.y + 5
        end
    end
end

function on.arrowUp()
    if TrackedObject ~= nil then
        if TrackedObject.y > TrackedObject.height / 2 then
            TrackedObject.y = TrackedObject.y - 5
        end
    end
end
```

I hope you don't mind that we are taking baby steps with these tutorials: concentrating on one thing at a time rather than trying to do multiple things. If the pace is painfully slow, I am sorry, but it made sense to me to be extra careful. And you are free to whip through these as fast as you like.

I would strongly suggest that at the end of each tutorial, you play. You use the ideas demonstrated to try and do something of your own. For example, what would be really nice to try at this stage might be to transfer these ideas to a similar context: how about you try doing this with an **image** instead of a drawn graphic object? All should work the same - just using the image definition commands you learned back in Tutorials 6 and 7.

In our next lesson we will see how to apply what we have learned so far to **multiple** objects.

Back to Top

Home ← TI-Nspire Authoring ← TI-Nspire Scripting HQ ← **Scripting Tutorial - Lesson 13**