School of Science

CSP2108 Introduction to Mobile Application Development

Workshop: Animation with Image Sheets

In this workshop you explore a tutorial example program that uses Lua control structures as well as Image Sheets to create a simple, fun animation.

Instructions

For this workshop, it is suggested that you work in pairs.

Download JungleScene.zip from Blackboard and unzip it to a suitable location. You should find a Corona project, which you can open in Corona. If all goes well, the app will run and you will see a simple animation of a green man eternally running away from a puma (?) in the jungle.

Understanding the code

Your first task is to read over the code and make sure you understand how the app works. So read over it yourself, doing your best to work out what each chunk does, and try to answer the following questions (Answers at the end of this document):

1. What does this function call do?

```
display.setStatusBar( display.HiddenStatusBar )
```

You can read about this in the Corona docs, at
https://docs.coronalabs.com/api/library/display/index.html .

2. In this line

```
local sky = display.newImage( "sky.jpg", centerX, centerY )
```

What is the size of this image, and where is it positioned on the screen?

3. The line

```
local tree = {}
```

creates an empty table, and then

```
tree[1] = display.newImage( "Palm-arecaceae.png" )
tree[1].xScale = 0.7; tree[1].yScale = 0.7
tree[1].anchorY = BOTTOM_REF
tree[1].x = 20; tree[1].y = baseline
tree[1].dx = 0.1
```

loads in an image and stores it in the table, with index (key value) 1. What do each of the last 4 lines of code do?

Hint: the first 3 lines set properties that every DisplayObject has – you can read about those in the Corona docs. The last line adds a new property to each of the tree images, which is then used later on (can you find where?).

4. This chunk:

```
-- an image sheet with a cat
local sheet1 = graphics.newImageSheet( "runningcat.png", { width=512, height=256,
numFrames=8 } )

-- play 8 frames every 1000 ms
local instance1 = display.newSprite( sheet1, { name="cat", start=1, count=8, time=1000 } )
instance1.x = display.contentWidth / 4 + 40
instance1.y = baseline - 75
instance1.xScale = .5
```

```
instance1.yScale = .5
instance1:play()
```

What will be the size in pixels of the image of the cat?
What is the name of the sequence that will play when the last line is executed?

5. In this line:

```
ground:setFillColor( 0x31/255, 0x5a/255, 0x18/255 )
```

the numbers 0x31, 0x5a and 0x18 are *hexadecimal numbers*. Approximately what are the values of the red, green and blue components of the fill colour of *ground*?

6. In this line:

```
local tPrevious = system.getTimer()
```

what does *system.getTimer()* return? You can find out from the Corona docs.

7. After that line, a function called *move()* is defined. We will come back to that. After the function, this line starts the action:

```
Runtime:addEventListener( "enterFrame", move );
```

We will start to learn about events next week, but you can try reading about this one in the Corona docs. What it does here is to set things up so that the *move()* method is called every time a new animation frame is about to be drawn (the timing will be determined by the timing of the sequences of the two sprites, instance1 (the cat) and instance2 (the man).

8. OK now we will have a look at the *move()* method – remember this will be called just before each animation frame is drawn. These two lines:

```
local tDelta = event.time - tPrevious
tPrevious = event.time
```

are to figure out how much time (in milliseconds) have passed since the last frame was drawn. This will then be used to work out how far everything should have moved. *event.time* is the current time, and *tPrevious* is used to store the previous time.

9. Now these lines:

```
if (grass.x + grass.contentWidth) < 0 then
        grass:translate( 480 * 2, 0)
```

```
end
if (grass2.x + grass2.contentWidth) < 0 then
        grass2:translate( 480 * 2, 0)
end
```

use *if* statements to determine whether the grass images need to be rearranged. At any time, one of the grass images will be on the left of screen, and the other one will be on the right, giving the impression of a single image continuously moving from right-to-left (and the trees moving with them). Actually our brains interpret this as the grass and trees remaining still while the cat and man move from left-to-right!

See if you can figure out how this trick of moving and swapping the two grass images works.

10. This chunk:

```
local i
for i = 1, #tree, 1 do
        tree[i].x = tree[i].x - tree[i].dx * tDelta * 0.2
        if (tree[i].x + tree[i].contentWidth) < 0 then
                tree[i]:translate( 480 + tree[i].contentWidth * 2, 0 )
        end
end
```

uses a *for* loop to do a similar thing with all the tree images, moving them from right-to-left, and then shifting them from the left edge of the screen to the right edge so that they keep cycling around forever.

How many times will the body of this loop be executed for each frame?

Adding some drama

Now that you have a good idea how the app works, we are going to modify it a little to add some drama to the story. We are going to add some code so that the cat gradually catches up to the man. Don't worry – at the last moment, the man puts on a spurt of speed and manages to keep out of the cat's clutches.

Here are the changes to be made:

1. First, I'd like to fix something that is wrong with the existing code. This line:

```
local i
```

immediately before the *for* loop is not necessary. To see this, comment it out and relaunch the app. The programmer has forgotten that the loop control variable *i* behaves like a local variable whose scope is the *for* loop. So the line above simply declares another local variable, also called *i*, which the program does not use.

2. That's better. Now your first task is to modify this chunk:

```
local instance2 = display.newSprite( sheet2, { name="man", start=1, count=15, time=500 })
```

Instead of just one sequence (called "man"), we want this sprite to have two sequences, one called "man", with a time between frames of 500 ms, and a second one called "manFast" with a time between frames of 200 ms.

Make this change. The app should run the same as before.

3. Now to add the drama. In the *move()* method, add a chunk of code after the *for* loop end before the *end* of the loop. The chunk should do the following:

```
if the current sequence is "man"
        move the man xOffset/10 to the left
if the current sequence is "manFaster"
        move the man xOffset/10 to the right

calculate the gap between the centre of the cat and the centre of the man

if the gap is less than 150
        set the sequence to "manFast"
if the gap is more than 250
        set the sequence to "man"
```

That's it! Of course you have to convert the "pseudocode" description above into Lua code, using *if* statements. Note that to make this work, after you have set the sequence, you will have to call *play()* on the man's sprite.

Answers:

1. Hides the status bar on most devices.
2. The image is 480x320 (fills the screen) and is placed in the centre of the screen.
3.

```
tree[1].xScale = 0.7; tree[1].yScale = 0.7
tree[1].anchorY = BOTTOM_REF
tree[1].x = 20; tree[1].y = baseline
```

The first line reduces the size of the image to 70%. The second sets the vertical anchor of the image for bottom alignment. The third line sets the initial location of the tree's anchor point (center bottom).

```
tree[1].dx = 0.1
```

This is used to how much to adjust the position of this tree image each ms, with this code

```
tree[i].x = tree[i].x - tree[i].dx * tDelta * 0.2
```

4.
What will be the size in pixels of the image of the cat?

Actually we can't tell. It depends on the resolution of the device and on what scaling method is used. But if the device is 480x320 (same as the content area) then it will be 50% of the size of the original image from file.

What is the name of the sequence that will play when the last line is executed?

"cat"

5. For example, 0x31 hexadecimal is 3*16 + 1*1 = 49 decimal. So the red component is 49/255 or about 0.192. Similarly working out green is about 0.353 and blue is about 0.094.
6. The time in milliseconds since the app started.
7. No question to answer!
8. No question to answer!
9. Draw yourself some diagrams to work this out.
10. That will be 8 times. *#tree* is the length of the array, *tree*.