Creating a Mole Whack Game with CoronaSDK - Part 2

This is part 2 of a multi-part tutorial series. For the first part, visit: Creating a Mole Whacking Game with CoronaSDK – Part 1. In this tutorial, we will learn how to move our Mole in the game and also a little about Object Oriented Programming (OOP). If this sounds scary, don't be scared. This is just a fancy way of breaking different parts of a program into smaller parts so they can be re-used. Let's begin...

This sample project uses the Mole Whack artwork provided Vicki Wenderlich.

1. Download the starter project from the APPlied Club projects page. Unzip the project.
2. Open the Corona Simulator and open the project in the **Part 2** folder. Then open the `main.lua` file in **Notepad++** or **TextWrangler**
3. We are going to move our Mole character to it's own file so we can separate all of the code related to the mole. This will help us in the future to find any changes we need to make to the mole.
4. Create a new file called "MoleGenerator.lua" in the same folder as the main.lua file for MoleWhack. This is going to be a text file but instead of having the .txt extension, it should have the .lua extension.
5. Now open up main.lua and remove the lines that created the "myMole" object. **Remove this:**

```
local myMole=display.newImageRect("images/mole_1.png",178,200)
myMole.x=display.contentCenterX
myMole.y=display.contentCenterY
```

6. For the time being, go down and find the following line and comment it out (that means add -- to the beginning of the line):

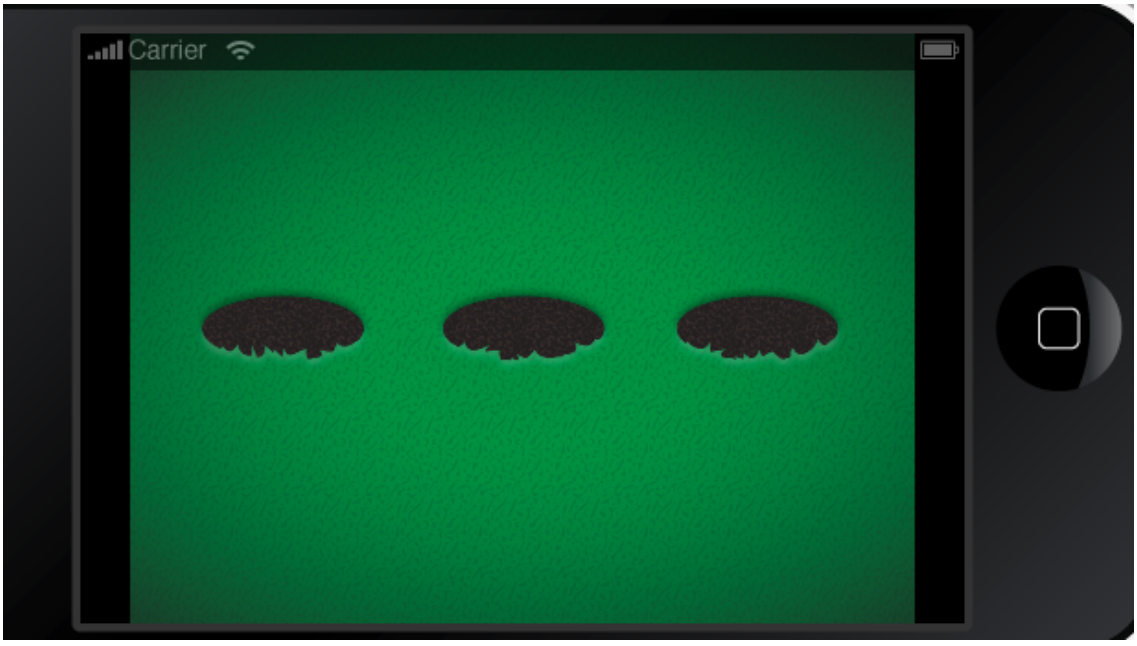
```
myMole:toFront()
```

7. Your main.lua should now look like this:

```
--------------------------------------------------------------------------------
----------
--
-- main.lua
--
--------------------------------------------------------------------------------
----------

-- Your code here
```

```
local grassUpper=display.newImageRect("images/grass_upper.png",1024,384)
grassUpper.x=display.contentCenterX
grassUpper:setReferencePoint(display.TopCenterReferencePoint)
grassUpper.y=0

local grassLower=display.newImageRect("images/grass_lower.png",1024,384)
grassLower.x=display.contentCenterX
grassLower:setReferencePoint(display.BottomCenterReferencePoint)
grassLower.y=display.contentHeight

grassUpper:toBack()
--myMole:toFront()
grassLower:toFront()

local bgDirt=display.newImageRect("images/bg_dirt.png",1024,768)
bgDirt.x=display.contentCenterX
bgDirt.y=display.contentCenterY
bgDirt:toBack()
```

8. Save and re-launch the project. Your mole should be gone but also make sure you don't have any errors in the terminal window.



9. Now open up *MoleGenerator.lua* in **Notepad++**(Win) or **TextWrangler**(Mac)

10. MoleGenerator.lua is what is called a module or class. To start a new module, add the following line to the top of the file:

```
module(..., package.seeall)
```

11. Now, we will create an object that represents our module. To do that, add the following:

```
local MoleGenerator={}

return MoleGenerator
```

12. So far, this is simple enough. What will eventually happen is, we will include MoleGenerator.lua into our main.lua file. main.lua will read the contents of MoleGenerator.lua and include whatever it returns.

13. Now that we have our MoleGenerator object defined, we are going to add a function to it that actually creates a new mole for us. In fact, that sounds like a great name for our function "newMole". Before the line `return MoleGenerator` add the following function:

```
function MoleGenerator.newMole()
    local newMole=display.newImageRect("images/mole_1.png",178,200)
    newMole.x=display.contentCenterX
    newMole.y=display.contentCenterY
    return newMole
end
```

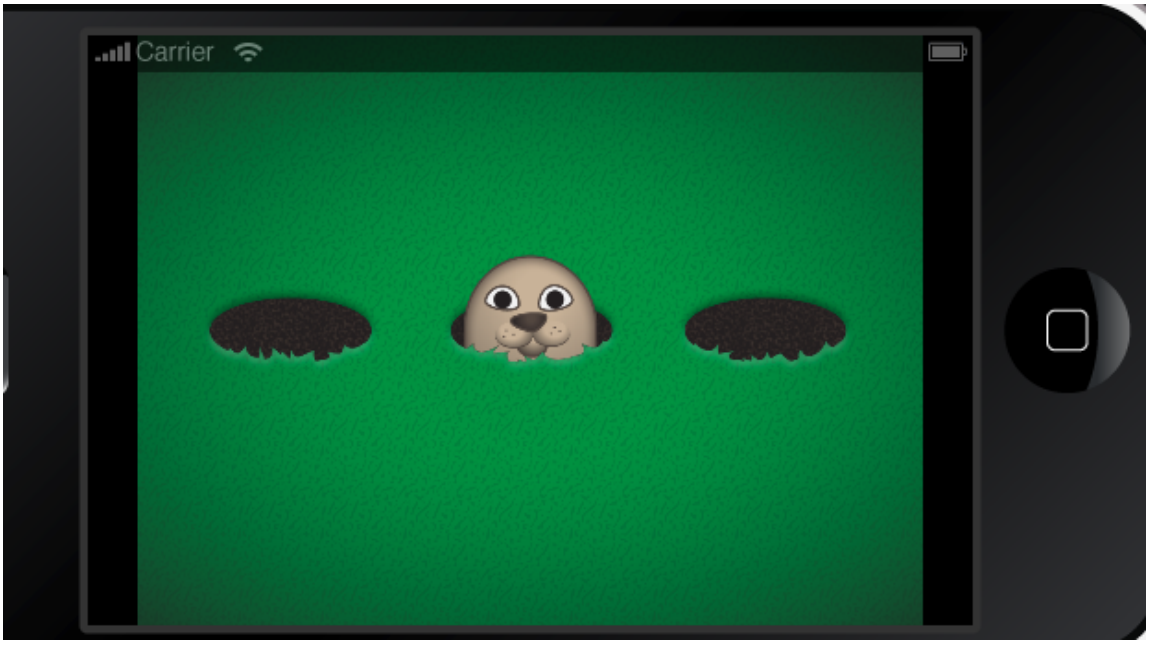
14. This is just a super simple function to create our mole object and center him. It probably looks very similar to the code you deleted in Step 5. At this point, your MoleGenerator.lua file should look like this:

```
module(..., package.seeall)

local MoleGenerator={}

    function MoleGenerator.newMole()
        local newMole=display.newImageRect("images/mole_1.png",178,200)
        newMole.x=display.contentCenterX
        newMole.y=display.contentCenterY
        return newMole
    end

return MoleGenerator
```

15. Switch back to the `main.lua` file and at the top, add the following lines:

```
local MoleGenerator=require("MoleGenerator")
local myMole=MoleGenerator.newMole()
```

16. Save and re-launch your project and your friend the mole should be back.



17. Alright, let's get that mole moving!

18. The first thing we want to do is make the mole hidden when the game launches, so change his starting point from `newMole.y=display.contentCenterY` to `newMole.y=display.contentHeight`, this will put the mole's center point at the bottom of the screen (and hide him behind the grass in the front). Hop over to MoleGenerator.lua and make that change on line 8.

19. Save and re-launch the app and you shouldn't see your mole anymore (you also shouldn't see any errors in the terminal).

20. Go back to MoleGenerator.lua. We are going to take this slow and in chunks. Let's start with just animating the mole coming out of the hole. After the line `newMole.y=display.contentHeight`, add the following code:

```
--Configure how fast our mole will move
local moleSpeed=5
local distanceToTravel=newMole.y - display.contentCenterY
local travelTime=distanceToTravel/(1/moleSpeed)
```

21. This block of code controls how fast our mole moves. The first line: `local moleSpeed=5` is just a value that I set the speed to initially. I found this number by trial and error and seems to work pretty well.

22. The next line: `local distanceToTravel=newMole.y - display.contentCenterY`, measures how far the mole has to travel. It subtracts the mole's destination (the center of the screen) from the current position (the bottom of the screen). Remember, the bottom of the screen is the height of the device, so on an iPad device that is laid out sideways, the vertical height is 768 points. This line is the equivalent of writing: `local distanceToTravel= 768 - 768/2`.

You might ask why we just don't write that, the reason is, if you move this code to a different device, those values might not work. An iPhone 5 has a height of 640 points so the same values wouldn't work.

23. Finally, the last line: `local travelTime=distanceToTravel/(1/moleSpeed)`, divides the distance by our speed. This object ends up being how long it should take our mole (in miliseconds) to travel to the center of the screen. If we used values for the iPad, the math would like like this: `local travelTime=384/(1/5)` or `local travelTime=1920` (or 1.92 seconds)

24. That is the most math you will need for this tutorial. I use those three lines in a lot of my projects when I need to move objects around the screen at a constant rate.

25. Okay, let's get this mole moving! Below those three lines above, add the following code:

```
newMole.transition=transition.to(newMole,{
                                                time=travelTime,
                                                y=display.contentCenterY,

                                            }
                                    )
```

26. This code creates a transition object. Transition objects move other objects around on the screen. We use the command `transition.to` to create our transition object, and then we assign it to a property on our new mole called `.transition`. Assigning the transition to a property will come in handy in the future.

27. The `transition.to` command takes a few arguments. The first argument, is the object to move, which in our case is `newMole`. The next argument is a lua table (signified with the open `{` and close `}` characters). This table includes a parameter named `time` which is how long the transition should take (we calculated that above in step 19. The other argument is what you want to change about the object. You can actually change multiple properties with the same transition command but for now, we just want to move it's `y` value to the center of the screen `display.contentCenterY`.

28. Save and re-launch. You should now see our little mole buddy pop his head up out of the ground.

29. Just in case you had issues, here is what MoleGenerator.lua currently looks like:

```
module(..., package.seeall)

local MoleGenerator={}

    function MoleGenerator.newMole()
        local newMole=display.newImageRect("images/mole_1.png",178,200)
        newMole.x=display.contentCenterX
        newMole.y=display.contentHeight
```

```
        --Configure how fast our mole will move
        local moleSpeed=5
        local distanceToTravel=newMole.y - display.contentCenterY
        local travelTime=distanceToTravel/(1/moleSpeed)

        newMole.transition=transition.to(newMole,{
                                                time=travelTime,
                                                y=display.contentCenterY,

                                            }
                                        )

        return newMole
    end

return MoleGenerator
```

30. Now we need to move the mole back down after he pokes his head up, otherwise he's going to get whacked all the time. We are going to need to make some changes to our existing code. First, we will want to put the the code that makes the mole move up into a function, so we can call it anytime we need it (and we will need it).

31. To create the function, wrap the code in the following:

```
--a function to move the mole up
local function moveMoleUp()
    newMole.transition=transition.to(newMole,{
                                                time=travelTime,
                                                y=display.contentCenterY,
                                            }
                                    )
end
```

32. We are going to create an equivalent function to move the mole down. I'm going to create this above the function to move him up and I will explain why in the next step. So above our "moveMoleUp" function, add the following code:

```
--a function to move the mole down
local function moveMoleDown()
    newMole.transition=transition.to(newMole,{
                                                time=travelTime,
                                                y=display.contentHeight
                                            }
                                    )
end
```

33. This function creates a new transition object that is performed on `newMole` and sets

newMole's y value to the bottom of the screen (also known as `display.contentHeight`)

34. We have two functions in our code now, one to move the mole up and one to move it down. If you think about the order of things:
    1. Start Game
    2. Mole is hidden
    3. Move mole up
    4. Mole is visible
    5. Move mole down
    6. Repeat steps 2-5
35. We will call the moveMoleUp function first and then call the moveMoleDown function once the mole has been moved up. But how do we know when the mole is visible? There are two ways, first, we know how long the mole is going to be moving up (it's our variable `travelTime`), so we could call the moveMoleDown function after 2 seconds has passed. And that would work okay. But there is a better way.
36. The `transition.to` accepts another parameter called `onComplete`. This allows you to tell the transition to call a certain function when the transition is done. If we use that new knowledge we can change the `function moveMoleUp` to the following:

```
--a function to move the mole up
local function moveMoleUp()
    newMole.transition=transition.to(newMole,{
                                                time=travelTime,
                                                y=display.contentCenterY,
                                                onComplete=moveMoleDown
                                            }
                                    )
end
```

37. Now all we need to do is call our moveMoleUp function and when it completes, it will move the mole back down. You can do this by adding the line `moveMoleUp()` after the end of the moveMoleUp function. So our MoleGenerator.lua file looks like this:

```
module(..., package.seeall)

local MoleGenerator={}

    function MoleGenerator.newMole()
        local newMole=display.newImageRect("images/mole_1.png",178,200)
        newMole.x=display.contentCenterX
        newMole.y=display.contentHeight

        --Configure how fast our mole will move
        local moleSpeed=5
        local distanceToTravel=newMole.y - display.contentCenterY
```

```
        local travelTime=distanceToTravel/(1/moleSpeed)

        --a function to move the mole down
        local function moveMoleDown()
            newMole.transition=transition.to(newMole,{
                                                    time=travelTime,
                                                    y=display.contentHeight
                                                }
                                            )
        end

        --a function to move the mole up
        local function moveMoleUp()
            newMole.transition=transition.to(newMole,{
                                                    time=travelTime,

y=display.contentCenterY,
                                                    onComplete=moveMoleDown
                                                }
                                            )
        end

        moveMoleUp()

        return newMole
    end

return MoleGenerator
```

38. Save and re-launch. Now our mole buddy is a little smarter. He pops up and then ducks back down. But that's it. We want to repeat this behavior (and maybe randomize it a little).

39. We are going to create a new function to control when our mole pops up. Comment out the line that reads `moveMoleUp()` (should be line 35 from above).

40. Add the following function:

```
local function randomMoveMole()
      local randomSeconds=math.random(1000,5000)
      timer.performWithDelay(randomSeconds,moveMoleUp)
end
randomMoveMole()
```

41. Let's talk about what this does. The first line, `local function randomMoveMole()`, just declares our function, nothing new there.

42. The next line, `local randomSeconds=math.random(1000,5000)`, uses the `math.random`

function to generate a random number between 1,000 and 5,000 (inclusive, inclusive means that it is possible for the function to generate a number equal to 1,000 or equal to 5,000). Why are these numbers so big? Because the next line takes a value in miliseconds and there are 1,000 miliseconds in a second, so 1,000ms=1 second.

43. This line, `timer.performWithDelay(randomSeconds,moveMoleUp)`, creates a timer object that performs a function after a certain delay. In our case, we are calling our `moveMoleUp` function after the delay.

44. This line, `end` just closes our function from step 41.

45. This line, `randomMoveMole()`, calls our new function and causes the mole to start to move after a random number of seconds (between 1 and 5 seconds, inclusive).

46. This is great so far, but it will only happen once. We need some way of calling our `randomMoveMole` function after the mole hides again. If only there was some argument or parameter we could send to do that after the mole moved down. Hmmm... thoughts? ideas? onComplete? Yes! Exactly!

47. We need to change our `moveMoleDown` function. But before we change it, it needs to know about our `randomMoveMole` function. Lua reads from the top of the file to the bottom so when it reads the lines about our `moveMoleDown` function, it has no idea that `randomMoveMole` is further down. To fix this, above our line that reads `local function moveMoleDown()` we need to add: `local randomMoveMole`:

```
 local randomMoveMole
--a function to move the mole down
local function moveMoleDown()
```

48. Now lets add that onComplete parameter to our transition in the `moveMoleDown` function, (**NOTE** Don't forget to add a comma at the end of the line `y=display.contentHeight` to let lua know there is another parameter):

```
newMole.transition=transition.to(newMole,{
                                                time=travelTime,
                                                y=display.contentHeight,
                                                 onComplete=randomMoveMole
                                            }
                                    )
```

49. And one last thing before we save, because we told lua about randomMoveMole above `moveMoleDown`, we don't need to call it local when we create the function, so go find the line that looks like this: `local function randomMoveMole()` and change it to this: `function randomMoveMole()`

50. Save and relaunch and your mole should be moving up and down at random intervals between 1 and 5 seconds. If you are stuck, here is the code from MoleGenerator.lua:

```
module(..., package.seeall)
```

```
local MoleGenerator={}

    function MoleGenerator.newMole()
        local newMole=display.newImageRect("images/mole_1.png",178,200)
        newMole.x=display.contentCenterX
        newMole.y=display.contentHeight

        --Configure how fast our mole will move
        local moleSpeed=5
        local distanceToTravel=newMole.y - display.contentCenterY
        local travelTime=distanceToTravel/(1/moleSpeed)

        local randomMoveMole
        --a function to move the mole down
        local function moveMoleDown()
            newMole.transition=transition.to(newMole,{
                                                    time=travelTime,

y=display.contentHeight,

onComplete=randomMoveMole
                                                }
                                            )
        end

        --a function to move the mole up
        local function moveMoleUp()
            newMole.transition=transition.to(newMole,{
                                                    time=travelTime,

y=display.contentCenterY,
                                                    onComplete=moveMoleDown
                                                }
                                            )
        end

        function randomMoveMole()
        local randomSeconds=math.random(1000,5000)
        timer.performWithDelay(randomSeconds,moveMoleUp)
        end
        randomMoveMole()

        return newMole
    end

return MoleGenerator
```

51. For fun and feel good points, how do you think you would make the mole change the hole he came out of? I will give some hints:

- It should be done in the `function randomMoveMole`.
- You will need to change the .x value of the newMole object.

See you next time!