School of Science

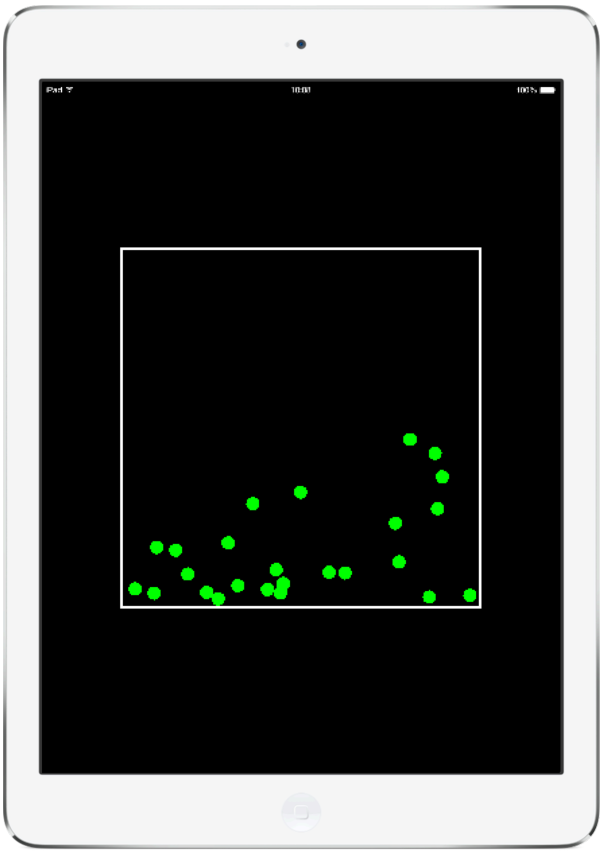
CSP2108 Introduction to Mobile Application Development

Workshop: Sensors and Physics

In this workshop you will add extra functions to an executive toy app. You will add a collision listener that plays sound, and a tap listener that toggles between draw modes.

Instructions

Download Tumble.zip from Blackboard and unzip it to a suitable location. This is the Corona project. Try it out. It looks like this:



This is the example from lectures.

Add a tap listener to show "debug" mode for physics objects

1. First, add in a global event listener to listen for "tap" events. The listener should look something like:

```
local function onTap( event )
    -- do something e.g. print out event info
end
```

and you could register it in the function *setupEvents()*. It should be added to the *Runtime* object.

To test, click on the simulator window.

2. Once you have this working, modify the listener so that it only responds to double-taps. You can do this by examining *event.numTaps*, and using an *if* statement.

3. Next, instead of your print statement, add code that toggles between "normal" and "debug" drawing mode. You will need to create a variable to keep track of which mode is currently being used (the default is "normal"), and you will need to call *physics.setDrawMode()* to switch between the two modes.

Add a collision listener that plays a "clink" sound when two gems collide

We are going to add a collision listener to play a sound effect when two gems collide, but we have to be careful, as there will be many, many collisions. So we are going to do a couple of things to limit the number of collisions that are reported, and the number that we respond to.

1. First, we will use this function

```
physics.setAverageCollisionPositions(true)
```

to limit the number of collisions per object to one per frame. All the collisions for that frame are "averaged" and only reported once (read about it in the online Corona Docs). A good place to put this is in *setupPhysics()*.

In lectures, we saw how to add a listener to listen for "collision" events. Instead, we are going to add a listener for "postCollision" events. This is because post-collision events have extra information available, including the force of the collision, in *event.force*. We can use this to only respond to larger collisions, say with force > 10.

2. Add the listener by creating a function that looks like

```
local function onPostCollision( event)
   -- print something
end
```

and then registering it as a "postCollision" listener on the global *Runtime* object. Read about this in the online Corona Docs.

3. Add in a test (an *if* statement) so that the printing only happens if the force is more than 10.

4. Finally, the code adds a property "type" to each wall and each gem, which we can now use so that we only respond to collisions between two gems, and not between a gem and a wall.

Add extra conditions to the *if* statement, so that printing only happens if both objects are gems (examine even.object1.type and event.object2.type).

5. Now let's get the audio working. Download the file "glass-click-4.wav", create a subfolder called "sounds" in the Tumble folder, and then add code to load the sound (use *audio.loadSound*), and to play it when collisions occur (using *audio.playSound*).

6. Just to be a bit fancier, it would be nice to have harder collisions sound louder. This can be done by setting the volume level on the channel before playing the sound. Something like this:

```
local channel = audio.findFreeChannel()
if channel ~= 0 then
   local volume = event.force/100
   audio.setVolume(volume, {channel = channel})
   audio.play(clinkSound, {channel = channel})
end
```

Read about this in the online Corona Docs, and then implement it.

7. When you have completed all these tasks, install the completed app on an Android device for testing. Rotate the device to change the direction of gravity and keep the gems tumbling.

Go nuts

If you finished this and found it too easy, have a go at enhancing the app. Use your imagination! How about adding different kinds of gems that have different effects (e.g. splitting into two, or joining together to make bigger gems), or adding extra “walls”. You could explore what can be done to customize collisions (see the online Docs to find out how).