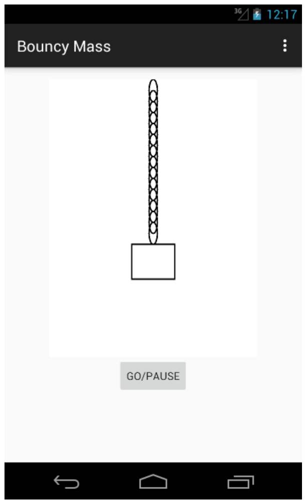
CSCD 372: Android Development
Lab 5: Animation (Bouncy Mass)

Lab Overview

This lab covers animation in a custom scalable view, by simulating a mass bouncing on a spring. In a later lab we will add some options to the simulation using the Preferences API. You may use any of the animation methods covered in lecture other than delayed invalidates in onDraw. If you choose to use an AsyncTask or thread, make sure you kill it using an appropriate lifecyle method in your Activity. Otherwise it will outlive your Activity, which creates problems, including memory leaks.



I use 3 coordinate systems here. The first is *real-world coordinates* for simulating the physics. That only needs to cover the up/down direction, which I call y in the equations below. The units for y is meters and I define positive y to be in the down direction in order to avoid sign confusion when converting to logical or screen coordinates. The origin for y should map to the center of the window. *Logical coordinates* are the coordinates that I do my drawing in. I define a final variable that controls the scaling from real-world to logical coordinates, and used the value 25 (logical) pixels/meter. My logical screen size is 600 x 800. This is then scaled to the actual screen space (*screen coordinates*) as in the previous lab, along with an enforced aspect ratio.

On each time step (I used 100 msec), we need to simulate the new y position of the mass and redraw the screen. The force equations that govern this case define the second derivative of y, which is what we will use to drive the simulation. With positive y (and gravity) in the down direction, the equations may be derived as follows. You will use the last one to simulate the physics:

$$F_{tot} = F_{grav} - F_{spring}$$
$$ma = mg - ky$$
$$m\ddot{y} = mg - ky$$
$$\ddot{y} = g - \frac{ky}{m}$$

where g is the acceleration of gravity (9.80665 m/sec^2), m is the mass (use 1 kg), k is the spring constant (use 1.5 N/m = 1.5 kg/sec^2), and y is the position of the top of the mass (in meters). On each time step, we therefore do the following: (a) compute the acceleration ($\ddot{y}$), which I put in a (local) variable called ypp, from the current position and gravity; (b) update the velocity ($\dot{y}$) by multiplying the acceleration by the time interval and adding it to the previous velocity; (c) update the position (y) by multiplying the velocity by the time interval and adding it to the previous position. You are then ready to redraw the window.

From the above, note that at each time step you are updating position (y) and velocity ($\dot{y}$) by adding to the previous value. That means that they must be initialized to some value when the simulation is first started. For now, set both the initial position and initial velocity to 0.

For the time interval, you can use either the actual elapsed time, or the time interval you specified for a simple timer. The latter is less accurate since message timing is not guaranteed and it doesn't account for your computation time, but you shouldn't notice any significant difference here.

When drawing the spring, use a reasonable number of reasonably sized ovals with a 50% overlap. For example, for 10 "links", a strategy is to: (a) convert the mass position to logical coordinates and add half the logical window height; (b) compute the spacing between links as that location divided by 11; (c) compute the height of an individual link as twice that; (d) draw 10 ovals of that height (and a fixed width), using that spacing, stepping "down" from y=0.

Put a button on your main screen that starts and pauses the simulation, as shown above. When it is all working the mass should bounce up and down indefinitely, as we have no simulation of air resistance here.

You can either restart or pause the animation on rotation (one choice is easier than the other). Make sure your aspect ratio is preserved when rotated. Also make sure that you have an about box and turn in your apk file and your java source.