Assignment 3

Overview

In Lecture, we learned about displaying simple graphics, using the kivy canvas to draw a primitive shape, and connecting that to user input. We also learned about different types of animation. In this assignment, you will enhance the behavior of the animated systems and make them musical. This assignment builds on previous ones. You will need to use code and wave files from Assignments 1 and 2 to complete this one.

Part 1: Bubbles Music

Start with the Key-Frame animated bubbles code we saw in class and add the following features:

- Whenever a touch_down event happens, play a note with the synth you wrote from Assignment 1 so that a note is heard when the Bubble appears. Choose the note's pitch based on the touch position using a mapping that makes sense.
- Make the duration of the note and the animation time of the Bubble match.
- Choose another note parameter (or parameters) to be dynamically set when the note is played. For example – duration, timbre, gain/volume. Map that new parameter (or parameters) in some interesting way as well. You are not limited to just looking at touch_pos. You can look at keyboard modifier keys (see kivy: *Window.modifiers*) as well like shift-touch_down, ctrl-touch_down, etc... You can also use randomness, or pick parameters from a pre-arranged sequence.
- Make other visual aspects of the Bubble (color, color animation, position, velocity, radius, radius animation) match or compliment the sonic parameters of the note being played.
- When playing a "note" consider using your snippets from Assignment 2 as well as your NoteGenerator from Assignment 1.

Part2: Bouncy Bubbles

Start with the Physics-based bubbles code and add the following features:

- Make the bubbles bounce off the sides in addition to the bottom.
- Whenever the bubble collides with the bottom or side, play a note. After a certain number of notes played, "remove the collision planes" and have the bubble go off-screen and disappear. At this point, indicate that it should be de-listed by having on_update return False.
- Try two different note-playing schemes. For example, a particular bubble instance may always plays the same note (maybe getting softer with each bounce?). Or, each bubble instance may play a little sequence and disappears after the sequence ends. These are just examples. Feel free to come up with a different behavior.

Part3 optional: Collision Detection

If you are thinking: hey, shouldn't these bubbles collide off of each other as well? Writing a collision system is fun and the results are very gratifying, but it can be a bit of work, so this part is optional. The basic approach is:

- Test for collisions on every update. You need to loop through every possible pair of bubbles. That means a total of N * (N-1) / 2 checks. A collision check with two circles is very straight-forward. Find the distance between the centers of the 2 circles. A collision is defined as that distance being small than the sum of the their radii. When you detect a collision, change the color of the Bubbles to see it working.
- Once you know two bubbles are collided, you apply an equal and opposite force to each bubble so that they move away from each other. I recommend modeling the force as a spring temporarily connected to the two bubbles.
  $F = -k * d$, where d is the penetration distance and k is a constant (this is hook's law). F is a vector pointing in the same direction as the vector connecting the two bubble center-points. Get acceleration via $acc = F / m$. Use *radius^2* for m. Apply *acc* to *velocity*, just like gravitational acceleration (*vel += acc * dt*).
- Once that all works, you can make a note play when two bubbles collide.
- If you decide to do this part, you can get extra credit to offset "imperfections" in other assignments.

Part4: Flower Music

Start with the Flower code and add the following features:

- Make the flower rotate by adding a top-level *Rotate()* object to the canvas and then animating it's angle parameter in *on_update()*. Add an additional speed parameter (in units of Hz – ie, # of full rotations per second).
- Make each flower a melody flower. It rotates at a steady speed and each petal is a note that plays when it reaches theta = 0.
- Create some type of graphical effect when the note plays. For example, the petal that played the note can vibrate, emit a shape based the note played, and/or change color.
- Add an additional circle (ie, Ellipse object) to the center of the flower. When you mouse-click (ie, touch_down) on that circle, the flower toggles between playing/rotating, and not-playing/not-rotating.
- Instantiate a few different flowers on the screen with different note sequences of different lengths.

Briefly document how to control your system in a README file (for Parts 1 and 2). No need to have a video recording of this one.

Finally...

Remember to add a paragraph about how you collaborated. Please have good comments in your code. When submitting your solution, submit a zip file that has all the necessary files. For example, if you used other files that I provided (like core.py), re-provide those files back to me in your submission. Upload your zip file to Stellar / Homework.