Assignment 3: Graphics

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, or use the instructors' versions which are provided in the common directory.

Part 1: Bubbles Music

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

Music:

- Whenever a `touch_down` event happens, play a note with the `NoteGenerator` from Assignment 1 (or see common/note.py).
- When choosing the parameters of the notes to play, create a system that varies **at least two** note parameters (e.g., pitch, gain, duration, timbre).
- Choose these note parameters based something that is not random. For example, you can look at touch position, or keys-held-down, or lookup values from a sequence.

Graphics:

- Instantiate a bubble for each note played, but create a new animation for that bubble that is different from the examples in class. Some ideas:
    - Different shapes: non-circular ellipses, number of segments
    - Different sizes
    - Different colors and color animations. Look up Color in Kivy docs (`Color.rgb` and `Color.hsv`)
    - Bubble motion can be based on keyframes or on calculated dynamics.
- Whatever you do, make sure there is a **clear mapping** between the visual aspects of a Bubble and the musical/sonic properties of the generated noted.

Feel free to change anything you need to about Bubble and `MainWidget`, including `__init__` parameters and member variables.

Part2: Bouncy Music

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

- Make the bubbles bounce off the sides in addition to the bottom.
- After a fixed number of bounces, disable the collision planes and have the bubble fall off-screen and disappear. Indicate that the animation is done by returning *False* in `on_update`.
- Implement a bounce callback. Whenever `PhysBubble` detects that a collision happened, it will call a function with parameters `(self, velocity)` – where `self` is

the `PhysBubble` instance and velocity represents the size of the bounce. See `listen_func` in audio.py for an example of a callback function. This callback function is passed into `PhysBubble` as an initialization argument.

- Implement `on_collide()` as the callback function in `MainWidget` and make it play a note. For each bubble bounce, you should hear a note playing. Make the note playing scheme interesting (i.e., not just random notes). Some ideas:
    - Each particular bubble instance plays the same note. But all together, they form some nice sounding chord/arpeggio, which can evolve over time.
    - Each bubble instance plays different notes. Maybe a little sequence? And then it disappears after the sequence ends.
    - Do something with the velocity parameter generated by each bounce.
    - As before, feel free to change whatever you need in the code, including function parameters, callback parameters, and member variables.

Part3: Creative Music

You have seen a variety of graphics techniques in lecture. Now that you have warmed up with Parts 1 and 2, create your own unique system that marries graphics and music. Create something graphical **that is different** from colored circles of Parts 1 and 2. Look into using Lines, Rectangles, Particle Systems (in addition to ellipses). For music, use `NoteGenerators` or `WaveGenerators`, or both.

Briefly document how to control your system in a README file. Create a video for Part 3.

Finally...

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.