Music 257

LAB 3: UNITY, CONTROLLERS, AND THE FREQUENCY DOMAIN

I. **SOLO: GETTING STARTED WITH UNITY: Due 4/19 at Midnight**

Unity is a *game engine*: A giant collection of tools that handles graphics, audio, controller input, and routines for various scripts and such.

Download Unity for your platform here. It's a giant program, so make sure you set aside some time to let it download. You will be prompted to make a Unity account. It's free, as long as you're not making too much money on whatever you're making with it.

After you download and install Unity, go through this tutorial independently : Breakout Demo**.** Breakout is a game that you may know by the name of *Brickbreaker*. It's kind of like Pong, except instead of playing against someone else, you are playing against an endless army of evil bricks.

After you finish the tutorial, change one small thing about the game. It could be color scheme, or the amount of bricks, or the scoring mechanism. Send me a document describing the thing you changed, as well as how you changed it and some accompanying screenshots that show your version off.

II. **IN A GROUP: AUGMENTING BREAKOUT WITH A CONTROLLER: To be shown off 4/22 in class**

Now that you have a basic understanding on the fundamentals of using Unity, build a basic game that uses a specific controller. This game could build off of your Breakout demo. Think about what your controller offers you in terms of input- What are the different ways a player can interface with the controller? Are there any real-world gestures that the controller accepts as input that we can translate to an in-game mechanism, giving the player a greater sense of physical presence in the game?

E-mail Ethan a link to a 30 second pitch video and a video describing your game and how you used the controller. Make sure to include gameplay!

Here are links to the tutorials for each controller:

Microsoft Kinect:
How To
Cool Examples

LEAP Motion:
How To
Cool Examples

Oculus DK2:
How To
Some Extra Tools

III. Solo Matlab: due Thursday 4/21 at midnight

Fourier Transform: Breaking an audio signal down into sinusoids.
You may recall from the last lab that we said we could create any sound by adding together sinusoids, and sinusoids are the most basic building blocks of sound. Here we are going to try to create a function that takes any audio signal and tells us the sinusoids that create that sound.

Create the following function:
`[Y, F]= getSpectrum(input, fs)`

Where input is any input signal, fs is the sampling rate of that signal. Our outputs are Y and F. Y is a list of *amplitudes* of sinusoids evenly spaced from 0 Hz to fs/2 Hz. F is the corresponding frequencies for each Y value.

Before we start, look at the documentation for the fft function in Matlab by typing `help fft`.

`fft(y,N)` is going to output the frequency information we need in complex number. We can just get the amplitudes out of this list of complex numbers but using the `abs` function in Matlab. N is going to be the actual size of our *window*- if it is a bigger number than the length of the signal we are throwing into fft, then the fft function will add zeroes at the end of the signal. The higher the value of N, the higher the *resolution* of our spectrum. N should be the next power of 2 after the length of our signay y.

The absolute value of the output of `fft(y,N)` is a list of the amplitudes of sinusoidal components evenly spaced from 0 Hz to fs/2

Hz, and then from -fs/2 to 0 Hz, respectively. The negative frequency components are going to always be the same as the positive frequency components as long as we are not feeding complex numbers into fft function.

At this point, we know what our *F* value is- it's N/2 evenly spaced numbers from 0 to fs/2.

Furthermore, our *Y* value is going to be the first half of the numbers (1:N/2) outputted by `abs(fft(y,N))`.

Send me your .m file for this function, a short .wav file (3-8 seconds) of your choice, and a plot of the output of your getSpectrum() function (`plot(F,Y);`) used on the wav file. Make sure to label your plot! Here's an example of a way to do that:

```
plot(F,Y);
title('Spectrum of example.wav');
xlabel('Frequency (Hz)');
ylabel('Amplitude');
```