Assignment 1: Sonic Board Game

Welcome to Music 257! We will be having weekly lab assignments up until the beginning of the final projects. There will be a mix of group work and individual work in the subsequent assignments. We would like you to work in groups for every game that we ask you to make.

We'll be asking you to make a basic game or prototype of a game in each assignment. We will provide some starter code and artwork, so that most of your time is spent designing the game and thinking about how to use sound in it. You are always welcome to make your own code and artwork (we'll show you some resources for game artwork).

This assignment will get you set up with all of the developing environments and start thinking about sound and game design.

I. Downloading and Setting Up Technology

In this course we're more interested in applying the concepts taught in class and making fun interactions rather than going in depth into programming of video games. That being said we encourage everyone to extend the assignment and make awesome and creative games.

The core technologies we will be using are Matlab, Processing and Chuck. For students that do not have as much programming experience, we're also going to start out with Game Salad, a drag and drop game editor, while concurrently learning the basics of programming. Feel free to use Game Salad whenever you see fit, but it might not work with a few labs in which we will be using sensors or novel controllers.

Matlab is available to everyone through remote login to either the myth computers or the ccrma machines. We would like you to use Matlab on a machine rather than remotely so you can hear the audio output. If you would like to work on your own computer feel free to set up Octave.

For an intro Matlab tutorial please watch Jimmy's tutorial on YouTube:
https://www.youtube.com/watch?v=ihBTKXWYNJc

GameSalad is a drag and drop game creation engine. You can use this engine for the beginning labs while learning how to program with ChucK and Processing. ChucK and Processing are necessary for the later labs when we start working with sound analysis and alternative controllers!

Download and go through the examples to get a feel for how GameSalad works.
http://gamesalad.com/download

ChucK is a digital audio programming language created by CCRMA's Ge Wang. We'll have Ge come in to talk about ChucK and how to use it with controllers! Next week we'll go through some ChucK examples to get everyone thinking about how to make sound with ChucK.

http://audicle.cs.princeton.edu/mini/

Processing is a high-level graphics library written in java. We'll be using this as our graphics engine because it interacts very easily with ChucK and with controllers like the NeuroSky.

Getting started is easy by looking at some examples. Explore the organizations website and check out the reference page: http://www.processing.org/

To see some cool examples (with code) check out this gallery:
http://www.openprocessing.org/browse/

II. Sonic Board Game

The Game Design Process:

- Inspiration
- Design Loop
- Implement
- Playtest

Design Loop:

- State Problem
- Brainstorm
- Prototype
- Assess

In a group of 4-5 please create a first prototype (low resolution paper prototype) of a board game. You must go through each step of the design process and finally hold a playtest. Pay attention to the mechanics that you are using in this game. What can a player do at every point in the game? Is that intuitive? Most importantly, what is the overall goal the player is trying to achieve?

We would like a main part of the game to use sound in some way. Think about how you can use sound to either affect the game (sound as a controller) or as a part of how the game is played (sound as a mechanic).

Deliverables:

The board game and a set of rules that clearly state the game's goal. Please bring them to next Tuesday's lab. Also please upload two videos to YouTube and send Nolan a link. One video of your group members pitching your game (30secs-1min) and one of your playtest (1-2 mins). Send the links to Nolan.

III. MATLAB

Grab this script to test out your functions, and make sure that it's in the same folder as your functions.

Part 1:

Create a matlab function `sineTone.m` of the following format:

```
output = sineTone(frequency, duration, fs)
```

This function should take the following parameters:

- Frequency: the frequency of a sinusoid in Hz (cycles per second)
- Duration: The number of seconds long the tone is
- fs: the sampling rate of the signal, in Hz (Samples per second)

Here are some things you need to know:

- A sinusoid can be expressed with the following equation:

$$y = \sin(2\pi f t)$$

Where f is our frequency in Hz, and t is the current time.

- Since we are in the digital realm, we can define the time t as n/f_s, where n is our sample number, and f_s is our sampling rate.
- Since we know the total number of seconds our sine tone needs to be, and we know how many samples we need per second, how many samples long should our sine tone be? (if fs = 44100 (which is typical for audio) and we want 2 seconds of audio, we need 2*44100 samples for 2 seconds of audio).

Part 2:

Create two functions:

```
output = rampUp(duration, fs, input)
output = rampDown(duration, fs, input)
```

rampUp should be able to take the output of our sineTone function and fade the signal in at the beginning of the tone. rampDown should make the signal fade out at the beginning of our tone.

Here are the parameters given:

- duration: the number of seconds long the fade in or fade out is
- fs: the original sampling rate of our input signal
- input: our input signal.

Here are some things you need to know:

- For either of these functions, we need to create an *envelope*: A set of numbers from 0.0 to 1.0 that will scale our signal at specific amounts at specific times. For example, an envelope entirely consisting of zeroes will silence any signal it's applied to, because any number times 0 = 0. Similarly, an envelope made entirely out of 1.0s will not affect our signal in any way, and an envelope entirely made out of 0.5's will cut the *amplitude* of the signal it is applied to in half.
- Given our duration and sampling rate, how many samples long will our envelope be?
- Look up the matlab function `linspace(a,b,n)`. It creates an n-length vector of numbers from value a to b. How could we use this to create an envelope that will ramp up

our signal from volume 0.0-1.0? Similarly, how could we use this to create an envelope to ramp down from 1.0-0.0?

- We need to apply the envelope we create to specific areas of our signal. We can select the first *n* samples of our input signal by writing `input(1:n)`, and the last *n* samples of our signal by writing `input(end-n+1:end)`

Check out the YouTube tutorial linked above. This is also a very good tutorial: http://www.math.utah.edu/lab/ms/matlab/matlab.html

Deliverables:

Please send Nolan (nlem {at} ccrma {dot} stanford {dot} edu) out the .m file where you have written your scripts and a plot of the first **1000** (not the entire signal) samples. Please put all of these into a folder with naming convention: **LASTNAME_FIRSTNAME_LAB1**