**MUS/COS 314 – Fall 2016 – Programming Assignment 2**

Due 9/30 (11:59PM)

Work with your partner!

Some useful documentation you should take a look at:

Types, values, and variables

http://chuck.cs.princeton.edu/doc/language/type.html

Unit Generators in ChucK

http://chuck.cs.princeton.edu/doc/language/ugen.html

Operators

http://chuck.cs.princeton.edu/doc/language/oper.html

Manipulating Time in ChucK

<http://chuck.cs.princeton.edu/doc/language/time.html>

**1) Create a ChucK patch that, when launched, plays a little mini-piece all by itself!**

**(These will be played in class!)!**

The mini-piece should be roughly around 60-120 seconds long, have distinct sections

(though they can overlap), and then shut itself off. The idea is to use user-written functions to alter the characteristics/variables of your synth patch over time.

Use at least 4 different UGens (not including the dac). Some notable synthesis or effect UGens to consider if you need ideas:

-Oscillators, each with different timbre: SinOsc, SqrOsc, SawOsc, TriOsc, Noise

-Filters: LPF (low pass filter), HPF (high pass filter), BPF (band pass filter)

-Envelope generator: ADSR (a little more advanced than just turning gain on and off!)

-Using an low frequency oscillator (LFO) to alter another parameter over time: any oscillator, query its .last() function for a sample value -1.0 to 1.0, scale it to a parameter range you need, chuck it to a UGen parameter.

-Effects: JCRev (reverb unit)

You can then alter the parameters of these things with numbers. They can be randomly generated, or stored in an array, or based off of user arguments, or from LFOs, it’s up to you.

In terms of sections of the piece, for example, you could have certain parameters changing quickly and randomly for a while, then stop that and have other parameters change slowly and repetitively. I suggest encoding each section as its own function, i.e.:

//20 times, each one second long

function void theMellowPart()

{

.1=>myReverb.mix; // low reverb

for(0=>int i; i<20; i++)

{

//choose at random from 16-element “mellowFrequences” array

Std.rand2(0,15)=>int myRandomIndex;

mellowFrequencies[myRandomIndex]=>myOsc.freq;

//every time through, crank up reverb a little

i \* .1 + .1 =>myReverb.mix;

1::second=>now;

}

}

//for 30 seconds, change filter frequency quickly

function void theChaoticPart()

{

now=>time myStartTime;

//another way of defining time!

//“while now is less than 30 seconds after then”

while(now< myStartTime +30::second)

{

Std.rand2(300,6000)=>myLPFilter.freq;

125::ms=>now;

}

}

Each of these functions run for a set amount of time, then return. If you’re feeling adventurous, have the functions take in parameters that define their behavior.

Remember that there are two primary ways of structuring time at the top level.

Linearly:

theChaoticPart(); //runs for 30 seconds, then returns and we go to the next line

theMellowPart(); //runs for 20 seconds, then returns and we go to the next line

Concurrently (by “sporking shreds”):

spork~ theChaoticPart(); //it runs for 30 seconds, but we immediately //go to the next line!

spork~ theMellowPart(); //20 seconds, ditto!

so in that concurrent example, both functions will be running at the same time (along with whatever happens next in your code). With sporking, you can have your functions overlap in time, yielding interesting effects (and possibly unanticipated ones, especially if you have a parameter being affected by two functions at the same time!)

You could also have functions that don’t do anything over time, but merely set parameters to a value and immediately exit (for example, one that sets all the frequencies to a particular chord, or another that sets them all to the same pitch, etc).

//input a pitch number, i.e. middle C is 60, build a major chord on it

function void setToMajorChord(int rootPitch)

{

Std.mtof(rootPitch) => myOsc1.freq; //set root

//root plus 4 semitones is major third

Std.mtof(rootPitch + 4) => myOsc2.freq;

//root plus 7 semitones is the perfect fifth

Std.mtof(rootPitch + 7) => myOsc3.freq;

}

**2) After you’ve done the above, modify your patch to accept an argument from the**

**command line (in the terminal, or miniaudicle).**

The code will use me.arg(0) to grab the first argument. Use Std.atoi() to convert a numeric argument from type string (aka text characters) to an integer.

Std.atoi(me.arg(0)) => int anInputVariable;

You will run it in terminal by going to directory the patch is in (using the “cd” command), and then typing “mypatch.ck:1337” to enter 1337 as the argument.

Use this argument to alter \_something\_ about the piece, so that when it is run with different arguments, it behaves differently (durations, pitches, any parameter you like...).

Make sure you explain what it does in your comments! And define in the comments what a “nice” range for that value might be (for us when we test it during grading).

Quick note on sound files, in case you want to use them (not required!): miniaudicle and chuck-from-the-terminal look for sound files in different places. MiniAudicle looks in a specific default directory (defined/alterable in miniaudicle’s preferences window), whereas when run from the terminal, it looks in the current folder (i.e. the same folder as the chuck file you are running).

**3) Get creative/get your hands dirty.** **Surprise people. Don’t just copy and paste the suggestions in this document!**

Formatting:

While the flexibility of chuck means you can put functions and variables anywhere, let’s agree to code structure of:

-comment section at the top, with your names, the date, and a very brief description, plus what your argument does

-define the synth chain(s) (unless they happen within a function)

-global variables

-your functions

-the main part of the patch

Make sure you comment your code generously, explaining what is going on!

WHAT TO TURN IN:

- your one chuck file

- include any additional files if there are any (audio samples, etc)

- Just turn in one copy of the assignment for every pair of students. Make sure you put both your names at the top of the file in comments

HOW TO TURN IT IN:

If it is just a chuck file, submit that. If you have any additional files, then zip everything together into one archive and submit that.