BAIN MUSC 336
Introduction to Computer Music

Project #2
DSP and Synthesis

Part A
Preparatory Homework

Due Date: Mon., Oct. 28, 2019

Directions
This project has two parts: Part A & Part B. To test your knowledge of DSP using MSP, in Part A you will create 12 short MSP patchers and store them in your personal folder on the Studio B hard drive. In Part B you will create original sounds using *additive synthesis* and *FM synthesis* and present them in class on the designated presentation day. It is assumed that you have read Cycling '74's *How MSP Works* & *How Digital Audio Works*. It is also assumed that you have studied the MSP Tutorials we have discussed in class as necessary.

Create a "MSP" sub-folder in your MUSC 336 private folder, and store all of the files associated with this project in that folder. Name each patcher *x*.maxpat: where *x* is the number of the exercise: i.e., 1-1.maxpat, 1-2.maxpat, 1-3.maxpat, etc. Use the self-commenting code style and be sure to comment your work.

Audio output should be directed to the iMac's built-in output. ***BE SURE TO TURN THE SPEAKER VOLUME ALL THE WAY DOWN BEFORE YOU WRITE ANY MSP PROGRAM!*** *Once you have visual confirmation of a signal (e.g., in MSP's main Audio Meter/Gain), slowly turn the volume up from zero to a comfortable level as the program is running.* Be sure to use File > Save As… to make a copy of any tutorial patcher *before* you begin programming (so that you don't alter the MSP Tutorials patcher file on the Studio B drive).

Using MSP Tutorials 1-3 as your starting point, create the following DSP networks:

TEST TONE AND L/R CHANNEL TEST

Using MSP Basics Tutorial 1 *Test Tone* as your starting point:

1.1 *Test Tone*. Create a DSP network that plays a *cosine waveform* (**cycle~**) with default *frequency* 440 Hz. Add an integer number box that allows the user to see the current value of the **gain~** volume slider.[1] Create a *mono* signal by patching left output of **gain~** into the L and R inputs of a **dac~** object.[2] As you work, be sure to keep an eye MSP's master Audio Meter/Gain and Audio On/Off controls (see Right Tool Bar, bottom).

1.2 *Solo/Mute*. Duplicate the previous patcher (*1-1.maxpat*) and add two "down-and-dirty" *solo/mute* toggles that allow the user to hear only the L signal, only the R signal, or both signals simultaneously.[3]

CONTROL SIGNALS

Using MSP Basics Tutorial 2 *Adjustable Oscillator* as your starting point:

2.1 *Fade in/out*. Delete all of the objects associated with oscillator 2, and replace the **ezdac~** with a **dac~**. Add a **toggle** to the **dac~** that serves as an audio on/off switch. Connect the **multiplication** (***~**) object to the **dac~** so that you have a *mono* output signal. To create a 6 sec. *fade in* and *fade out*, respectively, create the following 2 *linear ramp* control messages and send them into the **line~** object's left inlet: **0., 0.25 6000**, and **0.25, 0.0 6000**. Test your fade in and fade out messages to make sure they work. Finally, set the default *frequency* and *amplitude* of the **cycle~** to **440.** and **0.25**, respectively, on program load (**loadbang**).

2.2 *Glissando*. Duplicate the previous patcher. Add a *control rate* **line** object (rather than an *audio rate* **line~** object) and a control message that will cause the frequency of the oscillator to glissando up 1-octave from 440 to 880 Hz over 5 seconds and stay at 880 Hz).[4]

---

[1] See the right outlet of the **gain~** slider object.
[2] L stands for left channel, and R stands for right channel.
[3] Use **gate~** and **toggle** objects to implement the solo/mute toggles.
[4] The format of the message is: **440, 880 5000**: i.e., go from 440 Hz to 880 Hz over 5 sed..

2.3 *Chirp*. A chirp is a *sinusoidal* frequency sweep that is used to test audio equipment. Create a chirp that sweeps from 0 to 10000 Hz over 5 seconds and back down to 0 Hz over another 5 seconds. As in the previous patcher, use the **line** object to feed frequency values to **cycle~**. Two **line** objects are required with the following two linear ramp messages: **0, 10000 5000**, and **10000, 0 5000**.[5] To be sure that the sound doesn't get too loud , set the amplitude of the signal to 0.2 using a message into the right inlet of ***~**. *How does the perceived loudness of the pure tone change as its frequency gets higher?*

WHITE NOISE AND ADSR AMPLITUDE ENVELOPE

Using MSP Basics Tutorial 3 *Wavetable Oscillator* as your starting point:

3.1 *White noise*. Create a program that generates 6 seconds of *white noise*[6] with the following 4-stage *amplitude envelope*: ***A**ttack*: goes from 0 to 1 over 0.5 seconds; ***D**ecay*: goes from from 1 to 0.7 over 0.25 seconds; ***S**ustain*: lasts for 2 seconds; ***R**elease*: goes from 0.7 to 0 over 3.25 seconds.[7] Trigger the envelope with a **button** (b) object. Finally, delete all unnecessary objects.

3.2 *White noise spectrum*. Duplicate the previous patcher and add a **spectroscope~** that allows you to see a real-time *spectrogram* of the final output signal.

---

*Using the **playlist~** object's help file as a reference, create the following MSP patchers:*

AUDIO FILE PLAYBACK

4.1 *Audio file playback*. Open a new patcher file. Use the *Audio* file browser (see Max's Left Toolbar). Find the *stereo* (L/R) audio file: "drumLoop.aif". Drag the audio file into the blank patcher window. This will automatically create a **playlist~** object. Add an **ezdac~** and two **gain~** sliders (one for each channel) to your patcher. To hear the audio file play, connect all of these objects together and press the **playlist~** object's play button. Near the bottom of each **gain~** slider, add a number box that allows you to see the volume level of each channel displayed as an integer (0-157).[8]

4.2 *Metering signals*. Duplicate the previous patcher. Add a **levelmeter~** to each channel. Then chain the two **gain~** sliders together so that the L slider's volume level (right-most outlet) controls the volume level of the R slider. This will guarantee that the *2-channel mix* is always *balanced*. Finally, set the volume to of the L/R channels to a **100** on program load.

4.3 *Looping*. Study the **playlist~** object's help file. Add the **toggle** object and **message** (m) required to turn looping on.[9]

4.4 *Playback speed*. Study the **playlist~** object's help file again. Add the 4 messages required to be able to change the playback speed to: **0.5**, **1.0**, **1.5** and **2.0**.[10] Pass the 4 messages through a floating-point number box so the user can see the current playback speed.

The programming style and grading requirements are the same as for Project 1.
As always, be sure to strive for well aligned, easy-to-read, well-documented code.

Updated: 10/8/2019

[5] To program the up/down sweep, use the right outlet of the first **line** object to trigger the message that is sent to the second **line** object.
[6] Use the **noise~** object as the *sound source*. Or, if you prefer, you may use **pink~** noise.
[7] The syntax for the ADSR control message to line is: **0, 1. 500 0.7 250 0.8 2000 0. 3250**.
[8] 127 is *unity* gain. 128-157 is called *headroom*.
[9] The format of the message is: **loop $1**. The $1 is a variable that, depending on the state of the toggle, will be set to 1 or 0.
[10] The format of the message is: **speed $1**.