AUD6204 - Programming Environments
Lesson 7.1

PD Week 1 → 6
Recap Exercises

Contents

These exercise are intended to test your current level and knowledge, and to force you to use techniques we have already explored in new ways.

Attempt to complete these steps as best you can without looking back at the patches I have given you. Only use my resources as a guide when you are really stuck.

Before we begin:
First, create a new folder on your desktop. Name it:

21t1_AUD6204_Student#_Surname_Pd_Practice_Week_7

Save all of the code you create within this folder so that you can compress/zip it, and send it to me after you have finished.

Each solution should be **saved as an abstraction** that can be loaded (and used) within another patch.

REMEMBER: Always leave comments in your code!!!

1 Audio-Rate Patches

1.1 ADSR envelope 1

Create a patch that outputs an audio rate ramp suitable for use as an amplitude envelope when it receives trigger.

1.2 ADSR envelope 2

Create a patch that outputs an audio rate ramp suitable for use as an amplitude envelope when it receives MIDI velocities.

1.3 FM Synthesiser

Create a 2 oscillator FM Synthesiser. This synthesiser should:

1. Have an ADSR amplitude envelope.
2. There should be controls for the 'harmonicity ratio' and 'modulation index'
3. Create individual ADSR envelopes for 'harmonicity ratio' and 'modulation index'
4. The synthesiser should receive MIDI notes, and convert these notes to determine the frequency and amplitude of each note, in addition to triggering the amplitude envelope.

1.4 Stereo Delay

Create an audio delay patch. The delay effect should have:

1. **Delay time** and **feedback** for both left and right channels
2. Distortion
3. Filtering
4. Modulation (i.e. audio-rate modulation of the delay times)

1.5 Sampler 1

Create a patch that allows you to load 4 one-shot samples (e.g. kick, snare, hi-hat closed, hi-hat open).

1. The user should be able to load samples.
2. The patch should receive MIDI notes that trigger each sample (pitch should determine which sample is triggered, velocity should determine the amplitude).

1.6 Sampler 2

Create a patch that enables you to load and play samples. The patch should be capable of:

1. Looping sections of the sample (the user should have control over the length of the loops and their starting position in the audio file).
2. The loops should be playable in reverse as well as normally.
3. It should be possible to change the pitch (and, in turn, the playback length) of the samples.
    - You will need to implement the semitone formula for the pitch changes (user inputs semitone shift, e.g. +3 semitones, patch outputs the playback speed necessary to achieve desired pitch change).

2 Creating some useful code

Create code that successfully achieves each of the following.

1. Audio Formulae
   (a) Convert from frequency (Hz) to period (s) & vice versa
   (b) milliseconds to samples & samples to milliseconds
   (c) speed of sound at sea level (user inputs temperature)
   (d) wavelength
   (e) output of a compressor
   (f) phase shift
   (g) aliasing frequency given a sample rate
   (h) BPM to milliseconds & milliseconds to BPM
2. Mathematical formulae
   (a) $\frac{1}{x}$
   (b) $1 - x$
   (c) x^2 (square of a number)
   (d) x^3 (cube of a number)
   (e) x^y (where y is a positive integer)

3. Other useful patches

   (a) Counter
   User should have ability to set the number of integers to cycles through (n)
   Your patch should output an integer every time it receives a bang, and should loop from 0 $\rightarrow$ (n-1).

   (b) Uzi
   This patch should output x bangs as quickly as possible.
   The user should input a number (x) and the code should output that number of bangs. There should be three outlets:

      i. The left outlet outputs the bangs
      ii. The middle outlet outputs a bang **after** the last bang has been sent out the left outlet
      iii. The right outlet should output a count of each bang sent out the left outlet (starting at 0, so the count should be 0$\rightarrow(x-1)$)

   (c) Musical clock/conductor
   Create a patch that outputs a looped count (e.g. 0 $\rightarrow$ 7) at a given tempo.
   The user should be able to:

      i. turn the count on/off
      ii. enter the number of beats in the count (e.g. 8 would mean 0 $\rightarrow$ 7)
      iii. enter the tempo (either as bpm or a delta time(ms))
      iv. reset the count to 0

   (d) Clock
   Your patch should keep track of hours, minutes, and seconds.
   Users should be able to set the correct time manually, or reset to zero.

3 Musical Applications

3.1 Step Sequencer

Create a MIDI sequencer giving us control over both pitch and velocity.

1. We need an object to keep time. (Hint: think of a metronome)
2. We need to count this time:
   - For every bang received we need to "**add 1**" to our count
   - Then we need to keep a record of the result of that calculation (so that we can "add 1" to it the next time we receive a bang).
3. Now we need to convert this, ever increasing count, to a looping count - this count will be between 0 and x-1 (where x is the total number of steps desired).
4. Next we need to output a pitch and velocity for each step in the sequence.
   (a) We looked at two main ways of doing this:
      i. Using multiple interface objects, one for each pitch and one for each velocity.
      ii. Using one UI object to store all of the MIDI pitches and a second to store all of the MIDI velocities.
   (b) Whichever approach you take you will need to be able to read each step in the sequence (Remember: velocity must come before pitch!!!).
5. We then want to 'make a note.' The pitch and velocity is determined by our sequence, Make sure the duration is equal to (or slightly less than) the length of time between each beat (step 1).
6. Finally, we need to send each MIDI **note out** of PD to SimpleSynth.

3.2 Random Sequences

Using our sequencer (see Section 3.1) we want to be able to randomly generate the sequence.

1. We need to generate multiple random numbers. For this we need 2 objects:
   (a) One to generate multiple bang messages (Note: we need a record of the count of each of these bang messages).
   (b) One to generate random numbers.
2. Finally we need to use these random numbers to change the values in our sequence (using the random value as pitch/velocity and the bang count as step number).

3.3 Drum Machine

1. Create another sequencer that allows you to create drum patterns (you can use the same metronome & counter as before to trigger each step).
2. You will need separate sequences for kick, snare, hit closed, and hit open[1] (create other voices as well if you like) → https://msu.edu/course/mus/441/snapshot.afs/sullivan/fs02/MidiFiles/GemMidiDrumMap.html.
3. The sequence for each voice should be editable by the user, and should be read and then output to SimpleSynth, just like the previous sequencer.
4. Extend the drums machine to allow you to create random patterns.

[1] HINT: Unlike our melody sequencer, we do not need control over pitch, if we create a separate sequence for each drum (kick/snare/etc) each step merely needs to be on or off (is the drum hit, yes or no?).