COMP151 - Project 2 - Get your Synth On.

Spring 2021

Your final project is all about sound synthesis, song writing, and beat creation. By the time it's done you'll invent a new synthesized sound or two and program some music and/or beats. Like your first project, you'll be directed to complete a series of pre-designated versions, each of which will increase your potential grade.

As discussed in class on Monday 4/19, you'll begin by taking simple sine waves and programing a short melody. From there you'll start creating sounds above and beyond pure sine waves while starting to add multiple parts to your song. This means each version beyond the first will typically involve creating some new audio systhesis effect, applying that to your existing sound, and expanding upon your test song.

Version 0

The first version of your program is really about setting up a simple musical sequence with basic waves that you can then begin to enhance. You'll be given a starting file that contains a sine wave generator, a copy function, and the blend function from our last lab. Your job is to duplicate a bit of what we did on 4/19 but make a more involved melody. Here's your to do:

- Create a function called *synth* like what you saw in class. It should initially return a sine wave of the same frequency, amplitude, and duration that it is passed.
- Create a main function (or modify testsong) so that it generates and plays the follow 8 beat note sequence at 200 bpm: A3, C4, E4, G4, A4, G4, E4, C4. Each note should last for half a beat and occur at the beginning of each beat.

Version 0.5

Modify your synth so that it adds in a mixture of at least the first 2 harmonic frequencies, i.e. the integer multiples of the given frequency. For example, if the user passes 440 as the main frequency, then your synth should mix in a bit of 880 (2x440) and 1320 (3x440). Play around with the relative volumes and porportions. If you want to go above the first two harmonics, then you should use a loop to count out the harmonic numbers. For example *range(1,5)* could be used to generate the multiples 1, 2, 3, 4, 5 and thereby generate the f, 2f, 3f, 4f, and 5f for the user's desired frequency f.

Version 0.75

Now that we have a basic addative synthesizer, let's start to mold the amplitude in a more natural envelope. When a sound is created in nature it tends to get louder up to some peak volume then drop down in volume until it stops. The part of the sound that is increaing in volume is the *attack* and the part that's decreasing in volume is the *decay*. The combination of attack and decay is called the sound's envelope. We've discussed this once or twice in class.

- Write a function called *envelope* that implements the envelope effect described in the effects section below.
- Add an envolope to *synth*. The envelope would typically be applied after you've mixed together the main tone with its harmonics. If you want to get crazy, then you're welcome to apply envelopes to each individual note and possibly use different envelopes on different notes. It's up to you. You just have to use *envelope* at least once in *synth*.
- Add a second part to your melody. This part is a simple bass line where A2 is played on every other beat for 8 beats. That's the first, third, fifth, and seventh beat. Because the new part is very repetitive, *use a loop to create the 4 note pattern*. You should construct each part as it's own 8 beat long piece of audio and then use *blend* to mix them together. You can play around with the volume of each part in the final mix to get something that sounds right to you.

Version 1.0

Time for some beats.

- Design a new synth function called *drum*. It should be based on white noise (think snare drum) or some very low frequencies (think kick/bass drum) and have a very short envelope. Because these synths are percussive, their pitch doesn't change and

because their envelopes are so short, it doesn't make a lot of sense to pass a desired duration. This means our drum synth will take only one input, the desired amplitude.
- Construct a drum beat to go with your song by coming up with a two beat or four beat pattern that repeats for a total of 8 beats. Here's an easy way to make a 2 beat pattern. Divide each beat into half so that two beats gives you 4 half beats. For example, if a beat takes 0.5 seconds, then the half beats are each 0.25 seconds. Now, just choose some, but not all, of those half beats and sound your *drum* at those times leaving silence during the unused half beats. Repeat that pattern again three more times and boom, 8 beat rhythm. You can also get some good beats by breaking up a beat into quarters giving you 8 quarter beats within a 2 beat time frame.
- Modify your program to add your drum part to the song. Be sure to use the same techinque as before. Constuct each part individually and then mix them together.
- Make it so your program saves this sound to a .wav file on the hard drive.

Version 2.0

Play time! Use your synth and your drum, or invent new sounds, and construct a 32 beat song. The only requirements are that it involves at least 3 parts, like our little tune from version 1.0, and that it's 32 beats long. It can be purely percussive if you want. It does not need to involve notes. The key to making this musical, rhythmic, and not too tedious is repetition. Repetition allows programmable patterns and is also one of the cornerstones of music. If you need some more structure, then I recommend this approach:

1. Come up 2, 4 beat sequences, each with 3 parts. We'll call the first part the **A** part and the second the **B** part.
2. Your song is just the **A** part twice, then the **B** part, then **A** again.
   This is called an AABA song. It's a very well documented song structure and you can find lots of examples and variations by searching the web for *AABA song form*.

Grades and Dates

We'll dedicate two or three lab periods and some class time to this project. Use that time wisely. The final grade for the project is determined by how many versions you complete and the quality of the code you submit.

Important Dates

Date	Event
4/20	Lab for Version 0
4/27	Lab time for Project
5/4	Optional Lab
5/4	**Project Due by 11pm**

Project code should be submitted to GradeScope by 11:00 midnight on 5/4. Your sound files (version 1 and 2) should be uploaded to Moodle.

Grades

Your grade is determined based on the highest version number you complete (see the chart below) and the overall quality of the work submitted.

Version	Grade
0	D
0.5	C
0.75	C+

Version	Grade
1.0	B
2.0	A

Quality is measured by the cleanliness and clarity of the code and the appropriate use of sub-functions that exhibit reusable design. The versions determine the range in which you grade will fall and the quality of your code will determine where your grade lands within that range. For example, let's say you've finished version 0.75 and your code is neatly presented and well designed. Your grade starts at a C+ due to the version that you're presenting and could go up as high as a low B- based on the quality. Conversely, if your code is all crammed into a single function and your variables are poorly named, then your grade could go down as low as a high C due to poor quality.

Attack-Decay Envelope

Natural sounds do not typically start at one amplitude and stay at that amplitude for as long as you'd like them to. This is, however, the kind of sound our wave generators produce. A naturally occurring sound typically has a volume *envelope* where the volume grows to a peak and then shrinks back to nothing. The growth phase is called the *attack*. The shrinking phase is called the *decay*. A basic envelope effects lets you set the length of attack and decay.

Your *envelope* function should take a sound object, an attack length in seconds, and a decay length in seconds. It should then increase the amplitude of the sound from 0% to 100% for the duration of the attack phase and then decrease from 100% back to 0% for the duration of the decay phase. The catch here is that the input sound does not need to be as long as the sum of the attack and decay time. Imagine banging a gong. It has a short attack and a long decay. If you hit it once it and leave it alone you'll hear a loud tone that slowly decays to nothing. If you hit it again before it stops ringing, then you are effectively stopping the old envelope mid decay and starting a new envelope. Your *envelope* function should support this kind of musical sound by computing as much of the envelope as fits the given sound. Suppose your envelope has an attack of 0.25 seconds and a decay of 1 second. If the given sound is 0.5 seconds, then you get all of the attack phase and only 0.25 seconds of decay. If the given sound is 2 seconds you get the full envelope including 0.75 seconds of silence created at the end of the sound.