Processing Notes

Chapter 17: Sound

Processing does not have sound “built-in” the the programming language, but, there are many libraries that can be used. A programming language library is a set of classes & functions that make it easier to do something. We will use the Minim sound library. We first start with simply playing a .wav file. A .wav file is a sound file commonly used on computers. The following code assume there are a “pop.wav” and “thud.wav” file in the data directory. Given these two files exist, the following code is a complete sketch that will make a popping sound when the user clicks in the application window and a thud sound when the user releases the ‘t’ key:

```
import ddf.minim.* ;
Minim minim;

AudioPlayer au_player1, au_player2 ;

void setup()
{
  minim = new Minim(this) ;
  au_player1 = minim.loadFile("pop.wav") ;
  au_player2 = minim.loadFile("thud.wav") ;
}

void stop()
{
  minim.stop() ;
  super.stop() ;
}

void draw()
{
}

void mousePressed()
{
  au_player1.play() ;
}

void keyReleased()
```

```
{
  if (key == 't')
  {
    au_player2.play() ;
  }
}
```

The first line imports the library. This is necessary because, as said above, Minim is not built-in, you need to give instructions to include it, the import does this. The next line declares “mimim” to be a on object of type “Minim”. The next line creates two objects of type AudioPlayer. We will load a different wave file into each of the AudioPlayer objects, au_player1 and au_player2, and then use these objects to play the loaded sounds. In the setup( ) function we create an instance of Minim and then use this instance to load the pop.wav and thud.wav files into the two AudioPlayer objects. After the sound has been loaded in an AudioPlayer object, all we have to do is call the play( ) method on that object to play the sound file. We do exactly this in the mousePressed( ) and keyReleased( ) methods.

There is a new function here that we have not used before: stop( ). The stop( ) function is called with you exit the running processing application. If we are playing a long sound file, there needs to be a way to stop the playback, which can be done with the minim.stop( ) call inside the stop function.

With this simple code example you now know enough to add sound effects to your Processing sketches, just record all the .wav files you want, create and AudioPlayer object for each sound, load the sounds, and use them!

You may be wondering how do you record a sound? I recommend **Audacity** (google it) since it is free and runs on both Mac and Windows.

Generating Sounds

In the above examples we saw how to play sounds, lets move to generating sounds. As you probably know, sounds are made up of waves. Most of use have a vague memory of simple physics of sound: frequency, amplitude, different wave shapes. To have fun making sounds you don’t need to remember much more than that. Below is a complete sketch to create an amusing synthesizer. When you run the sketch, move your mouse up and down in the window to change the pitch:

```
import ddf.minim.* ;
import ddf.minim.signals.* ;
import ddf.minim.effects.* ;

Minim minim;
```

```
AudioOutput au_out ;

SquareWave sqw ;
LowPassSP lpass ;


void setup()
{
  size(400,600) ;
  minim = new Minim(this) ;
  au_out = minim.getLineOut() ;

  // create a SquareWave with a frequency of 440 Hz,
  // an amplitude of 1 and the same sample rate as out
  sqw = new SquareWave(440, 1, 44100);

  // create a LowPassSP filter with a cutoff frequency of 200 Hz
  // that expects audio with the same sample rate as out
  lpass = new LowPassSP(200, 44100);

  // now we can attach the square wave and the filter to our output
  au_out.addSignal(sqw);
  au_out.addEffect(lpass);
}

void stop()
{
  super.stop() ;
}

void draw()
{
  sqw.setFreq(mouseY) ;
}



void keyPressed()
{
  if ( key == 'm' )
  {
    if ( au_out.isMuted() )
    {
      au_out.unmute();
    }
```

```
    else
    {
      au_out.mute();
    }
  }
}
```

For generating sounds we need to import two more libraries: mimim.signals and minim.effects, hence, lines 2 and 3 above. Now consider the next three lines:

```
AudioOutput au_out ;
SquareWave sqw ;
LowPassSP lpass ;
```

The first declares au_out to be an AudioOutput object. SquareWave will be the wave form we will use. The LowPassSP is a filter which filters out some of the more obnoxious sounds that bother humans.

Now lets consider the following lines in setup( ):

```
au_out = minim.getLineOut() ;
sqw = new SquareWave(440, 1, 44100)l
lpass = new LowPassSP(200, 44100);
au_out.addSignal(sqw);
au_out.addEffect(lpass);
```

The first line initializes an AudioOutput line, or channel. The second creates a square waveform with a frequency of 440 Hz and an amplitude of 1. The next line creates a filter that does not touch signals with a frequency below 200, but reduces the amplitude (smooths out) signals of higher frequency to make them “sound nicer”. The next two lines add the signal in sqw and the filter in lpass to the AudioOutput line. Try commenting out the addEffect( ) call and compare the sounds produced.

The changes to the pitch are made in the draw function:

```
void draw()
{
  sqw.setFreq(mouseY) ;
}
```

The frequency in the signal is changed to the current mouse y-coordinate, i.e. a value between 0 and 600. Also note the keyPressed( ) function. If the user hits ‘m’ it toggles between muted and not muted.

Lets consider another example where we store the frequencies we want to play in an array and then loop over that array repeatedly playing the sounds. In this example we will put random values between 200 and 1000 and use those for our frequencies:

```
import ddf.minim.* ;
import ddf.minim.signals.* ;
import ddf.minim.effects.* ;

Minim minim;
AudioOutput au_out ;
SquareWave sqw ;
LowPassSP lpass ;

int CurrentFreqIndex ;
float [] Frequencies ;
int NUMFREQ = 100 ;


void setup()
{
  size(400,600) ;
  CurrentFreqIndex = 0 ;
  minim = new Minim(this) ;

  au_out = minim.getLineOut() ;

  sqw = new SquareWave(440, 1, 44100);
  lpass = new LowPassSP(200, 44100);
  au_out.addSignal(sqw);
  au_out.addEffect(lpass);

  // create the array of frequencies and initialize with random values
  Frequencies = new float[NUMFREQ] ;
  for (int i = 0 ; i < NUMFREQ ; i++)
    Frequencies[i] = random(200,1000) ;
}

void stop()
{
  minim.stop();
  super.stop() ;
}

void draw()
{
  CurrentFreqIndex++ ;
```

```
  if (CurrentFreqIndex >= NUMFREQ)
    CurrentFreqIndex = 0 ;
  sqw.setFreq( Frequencies[CurrentFreqIndex] ) ;
}

void mousePressed()
{

}


void keyPressed()
{
  if ( key == 'm' )
  {
    if ( au_out.isMuted() )
    {
      au_out.unmute();
    }
    else
    {
      au_out.mute();
    }
  }
}
```

The key to understanding this example is the setup( ) and draw( ) functions. In setup( ) we create the array and initialize each cell with a random number. In draw( ) we increment a counter to cycle through the cells of the array.

As a final example lets generate the scale of whole notes. Each note has a different frequency. For the octave from middle-C to the C above it, the frequencies are:

C	261.63
D	293.67
E	329.63
F	349.23
G	391.99
A	440

B	493.88
C	523.25

If we initialize an array to hold these value we can then play a scale or a simple tune. The following code can be added to one of the sketches above to play a scale:

```
float[] notes ;
int counter ;


void setup()
{
  notes = new float[8] ;   // create the array to hold the note frequencies
  notes[0] = 261.63 ;
  notes[1] = 293.67 ;
  notes[2] = 329.63 ;
  notes[3] = 349.23 ;
  notes[4] = 391.99 ;
  notes[5] = 440 ;
  notes[6] = 493.88 ;
  notes[7] = 523.25 ;
  counter = 0 ;
  frameRate(1) ;    // slow it down so each pitch is held for 1 second
}


void draw()
{
  sqw.setFreq(notes[counter]) ;
  counter++ ;
  if (counter >= 8)
    counter = 0 ;
}
```

We have now only scratched the surface of sound in Processing. For more information read the Minim man pages. Also, reading a little about sound on the web is helpful. We strongly urge the reader to look at these three sources of info:

1) http://code.compartmental.net/tools/minim/
2) http://code.compartmental.net/minim/examples/
3) http://en.wikipedia.org/wiki/Piano_key_frequencies

EXERCISE 17A

Write a sketch to create play sounds that start at a low pitch, constantly shift up until reaching a high pitch, then back down to the starting point, and the repeat indefinitely.

EXERCISE 17B

Write a sketch to play arpeggios. (Don't bother if you do not know what arpeggios are.)

EXERCISE 17C

Create your own game of Simon! Simon was the memory game with 4 buttons. The game tested the players memory. Let **N** be the number of notes in the test. The test starts with **N** equal to 1, then 2, then 3, then 4, and so on. For **N** equal to 3, the game would play a string of 3 buttons. Each button would light up and make a sound. Each of the buttons had an unique sound, and for that button the sound was always the same, and an unique color, again always the same for that button. After the game plays the sequence of three buttons/sounds, the player had to play it back.

Hints:

1) Build the game incrementally
2) First get four colored rectangles to show on the screen
3) Second make it so that when the user click on one of the rectangles (buttons), a different sound is made.
4) To the sound also make the button flash it's color
5) Now have the computer play a sequence: lighting up each button and playing the sound.
6) Have the user enter the sequence and the computer respond yes/no when the user gets it right or wrong.
7) Store the sequence in an array, where each cell is a number 1..4 for the 4 buttons.