**Laboratory 8: Playing Sound using the Minim Library**

**Introduction:** The aim of this section of the course is to learn how audio is manipulated in software programs and how we can control sound. Any sketch not available in Processing 2 is available on moodle. For .pde files on moodle, copy the code for the sketch to a new sketch and save it with the same name as the moodle sketch in a CS171 folder or subfolder. To add the sound file for the sketch, from the sketch menu choose Add File and add the file(e.g., groove.mp3) to the sketch. It will be placed in a new data folder in the sketch folder. Choose sketch, show sketch folder to see the new data folder.

**Part 1a:** Playing a Sound File

Sound Files are stored in many different audio ‘formats’, which define the bit layout for storing the audio as digital data on a computer. This can be an uncompressed format such as a .wav file or a compressed format(to reduce file size), such as the popular .mp3 format. The Minim sound library in Processing allows us to play sound files in many formats. There are a number of sample sounds on the moodle page for this Laboratory in the .wav and .mp3 format. It is important to specify the correct extension (.wav or .mp3) when referencing the samples in a Processing sketch. Download all the samples to a subfolder for this Laboratory.

Open the examples in Processing, go to Contributed Libraries, then Minim and Basics and open the example PlayAFile. Save the sketch to your own CS171 folder or a subfolder for this Laboratory. Study the code for the sketch below:

import ddf.minim.\*;

Minim minim;

AudioPlayer player;

void **setup**()

{

  size(512, 200, P3D);

 // we pass this to Minim so that it can load files from the data directory

  minim = new Minim(this);

 // loadFile will look in all the same places as loadImage does.

  player = minim.loadFile("groove.mp3");

}

void **draw**()

{

  background(0);

  stroke(255);

  // draw the waveforms

  // the values returned by left.get() and right.get() will be between -1 and 1,

  // so we need to scale them up to see the waveform

  // note that if the file is MONO, left.get() and right.get() will return the same value

  for(int i = 0; i < player.bufferSize() - 1; i++)

  {

    float x1 = map( i, 0, player.bufferSize(), 0, width );

    float x2 = map( i+1, 0, player.bufferSize(), 0, width );

    line( x1, 50 + player.left.get(i)\*50, x2, 50 + player.left.get(i+1)\*50 );

    line( x1, 150 + player.right.get(i)\*50, x2, 150 + player.right.get(i+1)\*50 );

  }

  // draw a line to show where in the song playback is currently located

  float posx = map(player.position(), 0, player.length(), 0, width);

  stroke(0,200,0);

  line(posx, 0, posx, height);

  if ( player.isPlaying() )

  {

    text("Press any key to pause playback.", 10, 20 );

  }

  else

  {

    text("Press any key to start playback.", 10, 20 );

  }

}

void **keyPressed**()

{

  if ( player.isPlaying() )

  {

    player.pause();

  }

  // if the player is at the end of the file,

  // we have to rewind it before telling it to play again

  else if ( player.position() == player.length() )

  {

    player.rewind();

    player.play();

  }

  else

  {

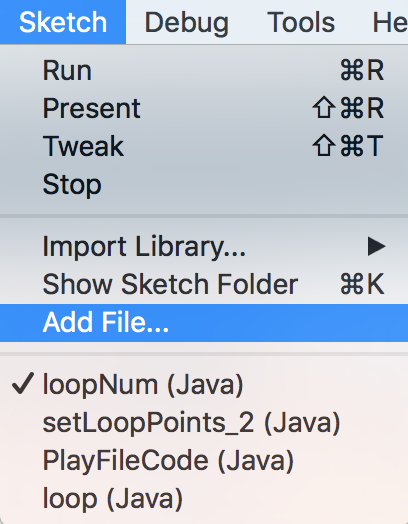
    player.play();

  }

}

**CodeSection 1:** The PlayAFile sketch for the examples library Minim

**To Do:** Run the Sketch to hear the sound and see the sound waveform drawn in the window. Note the changing shape of the waveform for the song playing, where it has higher peaks in louder sections. Next add another sound file to your sketch by browsing from the sketch->Add File.. menu (see the figure below) in processing. Browse to the folder where you placed the downloaded sound files from moodle and choose one to use in the sketch.



The AudioPlayer in Minim has a number of useful methods we can call to control the audio. In the keyPressed() function above you can see the methods isPlaying(), pause(), play() and rewind(). Another function is skip(milliseconds) allowing you to skip back and forth in the song a number of milliseconds.

**ToDo.** Open the getMetaData sketch from the Minim basic examples subfolder. Run the sketch on groove.mp3 and note each line of the metadata text.

**ToDo:** Open the skip.pde sketch from the Minim examples in the subfolder AuidoPlayer. The keyPressed() function allows you to rewind or fastforward the song based on whether you press the ‘r’ key or the ‘f’ key. Add this functionality to the first sketch so that pressing ‘p’ will pause the song, ‘f’ will skip forward a number of milliseconds(+1000) and ‘r’ will skip backwards(-1000) the same amount. Add a text line on the screen giving instructions on the key to control skipping. Use the spacebar to take a snapshot of the wave playing. Save it as sketch1a2.png by placing the keyReleased() method at the end of the scketch.

void **keyReleased**() {

  if (key == ‘ ‘)

     save("sketch1a2.png");

}

**Part 1b:** Looping the Sound

The AudioPlayer in Minim has a number of useful methods we can call to control the audio. In the keyPressed() function above you can see the methods isPlaying(), pause(), play() and rewind(). There is also a loop() function. The loop() function can be empty (loop()) or can take an integer parameter loop(int numloops) indicating the number of times to loop.

**To Do:** Replace the code in keyPressed() above with code to set the number of times the song should play/loop based on selecting a number key from the keyboard. First we need to convert the String value of the keyPressed to an integer. Then run the sketch and press a number key to test it working.

void **keyPressed**()

{

 String keystr = String.valueOf(key);

  int num = int(keystr);

//if num is between 0 and 10

// loop the player

**To Do:** Open another Minim example, setLoopPoints from the Audioplayer folder. Save this sketch in your own home folder for CS171 or subfolder. In this example the mousePressed() function is used to set loop points for the song as it is playing. Firstly, the map function is used to map the mouseX position to the length of the song, snip.length(). The map function returns a float type so we convert the result to an integer to match a position in the song which is in milliseconds. The RIGHT mouse button sets the loop end point, otherwise the mouse sets the loop start point.

  int ms = (int)map(mouseX, 0, width, 0, snip.length());

**ToDo:** Combine the code for setting the number of loops based on a number key pressed with this sketch. Run the sketch a few times with a different number of loops and loop points.

Answer the following question in your word document under the heading “Sound File Types”.

Name one uncompressed and one compressed sound file type format?

**Part 2a:** Programming Sound with Oscillators

You are now familiar with the Minim library for manipulating sound files. Another way of producing sound is to use well-known basic wave shapes that produce different sound colours. Firstly, we create a simple Sine wave at 440Hz (A4 note, the A above middle C on the Piano). We do this using an Oscil object from UGens in the Minim library:

wave = new Oscil( 440, 0.25f, Waves.SINE );

The input parameters to Oscil are the frequency(pitch=44Hz), the amplitude (loudness 0.25f), and the type of wave, a SINE wave. Copy the following program into a new sketch and run it.

import ddf.minim.\*;

import ddf.minim.ugens.\*;

Minim minim;

AudioOutput out;

Oscil       wave;

Oscil       mod;

void **setup**()

{

  size(512, 200, P3D);

  minim = new Minim(this);

  // use the getLineOut method of the Minim object to get an AudioOutput object

  out = minim.getLineOut();

  // create a SINE wave Oscillator, set to 440 Hz, at 20% amplitude

  wave = new Oscil( 440, 0.25f, Waves.SINE );

  wave.patch( out );

}

void **draw**()

{

  background(0);

  stroke(255);

  // draw the waveforms

  for(int i = 0; i < out.bufferSize() - 1; i++)

  {

    line( i, 50 + out.left.get(i)\*50, i+1, 50 + out.left.get(i+1)\*50 );

    line( i, 150 + out.right.get(i)\*50, i+1, 150 + out.right.get(i+1)\*50 );

  }

}

**ToDo:** Add a void **keyPressed**() method to the end of the sketch. In the keyPressed() method add code to change the wave shape to one of Waves.SINE, Waves.SQUARE, Waves.TRIANGLE, Waves.SQUARE, Waves,QUARTERPULSE) when the keys 1,2,3,4,are pressed respectively. Use the command:

   wave.setWaveform(Waves.TRIANGLE);

to change the wave shape, replacing TRIANGLE with one of the waveform names. Ensure to turn on code completion in Processing. From the Processing menu, choose Preferences and ensure that Code completion with Ctrl-space is checked. Then run the sketch and test it by hitting the different keys. Notice the waveform shapes.

Next we will program 2 oscillators. Here one oscillator will control how the amplitude changes of the other oscillator. This is called amplitude modulation(AM). The oscillator used to control the amplitude is called the modulator waveform.

import ddf.minim.\*;

import ddf.minim.ugens.\*;

Minim minim;

AudioOutput out;

Oscil       wave;

Oscil       mod;

void **setup**()

{

  size(512, 200, P3D);

  minim = new Minim(this);

  // use the getLineOut method of the Minim object to get an AudioOutput object

  out = minim.getLineOut();

  // create a triangle wave Oscil, set to 440 Hz, at 0.5 amplitude

  wave = new Oscil( 440, 0.5f, Waves.TRIANGLE);

  // create a sine wave Oscil at 2Hz for modulating the amplitude of wave

  mod  = new Oscil( 2, 0.25f, Waves.SINE );

  // connect up the modulator

  //we patch the modulator to the amplitude of our main waveform

  mod.patch( wave.amplitude );

  // patch the modulated wave to the output

  wave.patch( out );

}

void **draw**()

{

  background(0);

  stroke(255);

  // draw the waveforms

  for(int i = 0; i < out.bufferSize() - 1; i++)

  {

    line( i, 50 + out.left.get(i)\*50, i+1, 50 + out.left.get(i+1)\*50 );

    line( i, 150 + out.right.get(i)\*50, i+1, 150 + out.right.get(i+1)\*50 );

  }

}

void **mouseMoved**()

{

  float modulateAmount = map( abs(height-mouseY), 0, height, 0, 1 );

  float modulateFrequency = map( mouseX, 0, width, 1, 100 );

  mod.setAmplitude( modulateAmount );

  mod.setFrequency( modulateFrequency );

}

**ToDo:** We can change the modulation amount(the amount the pitch changes) and the modulation Frequency (the speed it changes) in the map functions in **mouseMoved**. Change both map functions as follows and then run the sketch again, moving the mouse across the screen to hear the difference.

float modulateAmount = map( abs(height-mouseY), 0, height, 0, 10 );

float modulateFrequency = map( mouseX, 0, width, 1, 10 );

**Part 2b:** A Simple Drum Machine

The following sketch uses the AudioSample object, which is used for short sampled sounds. It uses the method trigger() to play a sound rather than the play() method of AudioPlayer. Copy the following code to a new sketch and run the program.

// import Minim

import ddf.minim.\*;

Minim minim;

AudioSample d1;

AudioSample d2;

AudioOutput out;

// track when a drum has been struck

boolean drum1struck;

boolean drum2struck;

void **setup**() {

  // initialize the screen

  size(210, 120);

  smooth();

  // initialize sound

  minim = new Minim(this);

  out = minim.getLineOut();

  d1 = minim.loadSample("bongo1.wav");

  d2 = minim.loadSample("bongo7.wav");

  // set boolean variables to initialize the graphics

  drum1struck = false;

  drum2struck = false;

}

void **draw**() {

   background(255);

  // draw the drums: if a drum has just been struck

  // then fill its ellipse with color as visual feedback for the user

  // drum 1

  if (drum1struck == true) {

    fill(255,0,0);

    drum1struck = false;

  } else {

    fill(255);

  }

  ellipse(50, 55, 100, 100);

  // drum 2

  if (drum2struck == true) {

    fill(0);

    drum2struck = false;

  } else {

    fill(255);

  }

  ellipse(160, 55, 100, 100);

}

void **keyPressed**() {

  if (key == ‘a’ || key == ‘A’) { //note you may have to retype the

    drum1struck = true; //single quote marks after copying

    d1.trigger();

  }

  else if (key == ‘b’ || key == ‘B’) {

    drum2struck = true;

    d2.trigger();

  }

}

**ToDo:** Rather than using particular keys to control the sound, code a mousePressed() function which will play each drum when clicked with the mouse. To do this note where each ellipse is on the screen. If mouseX is over the left part of the screen it should play drum1 otherwise it should play drum2.

void **mousePressed**() {

  if(mouseX< half the width of screen){

       code here to trigger the first drum

       //set Boolean for drum1 to true

drum1struck = true;

  }

  else {

       code here to trigger the second drum

//set Boolean for drum1 to true

       drum2struck = true;

  }

**ToDo:** Now add two more ellipses below the first two which wlll trigger two more drum sounds when clicked with the mouse. Add the samples, kick and snare, to the sketch from moodle. You will need to code the mousePressed() function so that when you are over any of the 4 ellipses a mouse click will trigger that drum sound. You will need to declare:

* two extra AudioSample objects:

AudioSample d3;

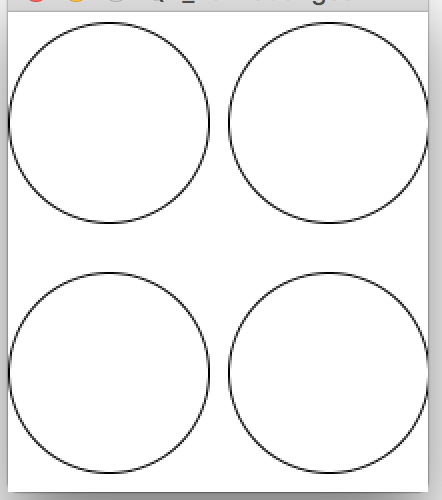
AudioSample d4;

* two extra booleans:

boolean drum3struck;

boolean drum4struck;

Load the two new samples in setup() and set the Booleans to false similar to the first two drums. In draw() place two new ellipses below the first two and code and if else for each to fill the ellipse when drum3struck or drum4struck is true. Then in mousePressed() change the if else block so that each of the four drums are triggered when the mouse is over that part / quarter of the screen and setting the drumstruck variable to true for the particular drum.

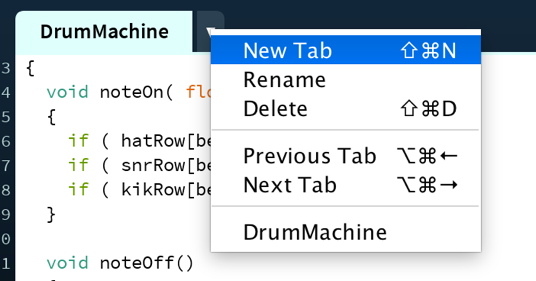


**Part 3:** A Drum Sequencer

From the Minim Examples open the subfolder, Advanced, and then open the sketch DrumMachine. Run the sketch to see the interface and click on some of the red rectangles to hear the result. Notice that there is a matrix with 3 rows of 16 rectangles. Each row represents a different drum sound. Notice the single green rectangle running along the top of the matrix. The green rectangle triggers the sound of whichever red rectangles are turned on(green).

Next, scroll through the code in the sketch. Notice that there are two classes inside the DrumMachine sketch, Class Tick and Class Rect. Class Rect draws one rectangle and decides whether it is turned on(green) or off(red). Class Tick decides whether a drum sound is turned on (noteOn()) or off(noteOff()).

**ToDo**: First we are going to move the Rect class to a new tab. Click on the white upside-down triangle beside the DrumMachine tab and choose New Tab. Give it the name Rect (the same as the class Rect)



Now highlight and cut all the code for the class Rect in DrumMachine and paste it into the new tab Rect

class Rect

{

  int x, y, w, h;

  boolean[] steps;

  int stepId;

  public Rect(int \_x, int \_y, boolean[] \_steps, int \_id)

  {

    x = \_x;

    y = \_y;

    w = 14;

    h = 30;

    steps = \_steps;

Copy all of this code for class Rect and place in the new tab Rect. Then cut the code from DrumMaching.

    stepId = \_id;

  }

  public void draw()

  {

    if ( steps[stepId] )

    {

      fill(0,255,0);

    }

    else

    {

      fill(255,0,0);

    }

    rect(x,y,w,h);

  }

  public void mousePressed()

  {

    if ( mouseX >= x && mouseX <= x+w && mouseY >= y && mouseY <= y+h )

    {

      steps[stepId] = !steps[stepId];

    }

  }

}

You will now have two tabs with code, one for Drum Machine and one for class.Rect. This is known as modularization of code and helps to separate the functionality of the program so that it is easier to read, understand and also to reuse. From the sketch menu choose Show Sketch Folder and notice that there are now two .pde sketches, DrumMachine.pde and Rect.pde. Run the sketch again to ensure it works. Study the code below for DrumMachine.pde.

import ddf.minim.\*;

The variables for our DrumMachine

3 arrays each to hold 16 Boolean variables

An expandable array (ArrayList) of rectangle objects

import ddf.minim.ugens.\*;

Minim       minim;

AudioOutput out;

3 variables of type Sampler to hold our drum samples

Sampler     kick;

Sampler     snare;

Sampler     hat;

boolean[] hatRow = new boolean[16];

boolean[] snrRow = new boolean[16];

boolean[] kikRow = new boolean[16];

ArrayList<Rect> buttons = new ArrayList<Rect>();

int bpm = 120; // the tempo of our DrumMachine

//(60bpm is one beat per second)

int beat; // which beat we’re on now

Out instrument for playing the samples. AudioOutput calls noteOn()based on the values sent to playNote().

AudioOutput calls noteOff()based on the values sent to playNote().

class Tick implements Instrument

{

  void noteOn( float dur )

  {

    if ( hatRow[beat] ) hat.trigger();

    if ( snrRow[beat] ) snare.trigger();

    if ( kikRow[beat] ) kick.trigger();

  }

  void noteOff()

  {

    // next beat

    beat = (beat+1)%16;

    // set the new tempo

    out.setTempo( bpm );

    // play note now, with a sixteenth note duration

    out.playNote( 0, 0.25f, this );

  }

}

void **setup**()

{

  size(395, 200);

  minim = new Minim(this);

  out   = minim.getLineOut();

  // load all of our samples, using 4 voices for each.

  // this will help ensure we have enough voices to handle even

  // very fast tempos.

  kick  = new Sampler( "BD.wav", 4, minim );

  snare = new Sampler( "SD.wav", 4, minim );

  hat   = new Sampler( "CHH.wav", 4, minim );

// patch/connect samplers to the output (speakers)

  kick.patch( out );

  snare.patch( out );

  hat.patch( out );

//Create the matrix of buttons to trigger samples

//This will be a 3x16 button grid

  for (int i = 0; i < 16; i++)

  {

    buttons.add( new Rect(10+i\*24, 50, hatRow, i ) );

    buttons.add( new Rect(10+i\*24, 100, snrRow, i ) );

    buttons.add( new Rect(10+i\*24, 150, kikRow, i ) );

  }

//We start the sequencer at beat 0

//We will have 16 beats per sequence

  beat = 0;

  // start the sequencer at 80bpm tempo

  out.setTempo( bpm );

// Start the sequener by calling playNote at beat 0 for

// one 16th note length, played on our instrument Tick

//Tick then iteratively calls playNote again in the noteOff() method

  out.playNote( 0, 0.25f, new Tick() );

}

void **draw**()

{

  background(0);

  fill(255);

Continually draw the matrix of buttons

  for(int i = 0; i < buttons.size(); ++i)

  {

    buttons.get(i).**draw**();

  }

Set the color for our running beat marker to dark red every 4 beats and green otherwise.

Draw the Beat marker 16 times across the screen

  stroke(128);

  if ( beat % 4 == 0 )

  {

    fill(200, 0, 0);

  }

  else

  {

    fill(0, 200, 0);

  }

  // beat marker – the running rectangle

// at the top of the drum matrix

  rect(10+beat\*24, 35, 14, 9);

}

When the mouse is pressed call mousePressed for all the buttons to determine which ones have been turned on or off.

void **mousePressed**()

{

  for(int i = 0; i < buttons.size(); ++i)

  {

    buttons.get(i).**mousePressed**();

  }

}

**ToDo:** Change the bpm at the top of the sketch to change the speed of the sequence. Add one or more rows of 16 rectangles to the DrumMachine matrix to trigger another sample from the samples on Moodle(Choose one from tom1.wav, tom2.wav, shake.wav, cymb.wav, handclap.wav, WB .wav). To do this you will need to:

* set up a new Sampler variable,
* a new array of 16 boolean values
* another call to load a sample: e.g.,shake   = new Sampler(. .
* patch the sample to out
* add a new row of Rect to the matrix, buttons.add( new Rect(. . .
* add a new if condition in noteOn().