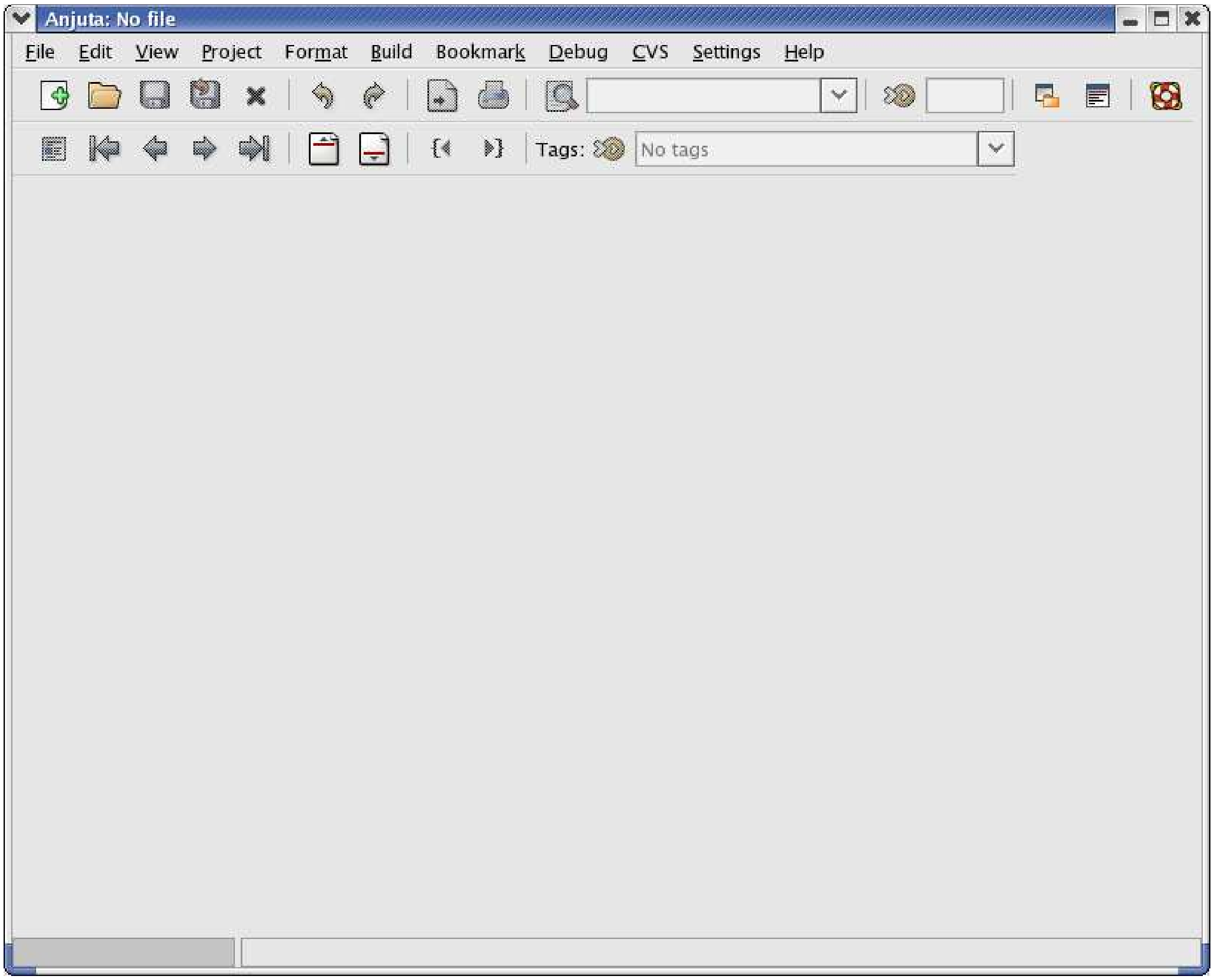
# Scheme-snd-lab1

This document assumes knowledge of the concepts in course documents **scm1-data** and **scm2-iter**. In the examples below, you'll use scheme to make sounds. Instead of typing to guile in a terminal, code development will happen in an editor (with parenthesis matching and code evaluation in the *snd* audio environment). The very first time requires installation of some files. Do this once:

| | |
|---|---|
| 1. Open a terminal. Then change directory to: | **cd ~cc/220a/system** |
| 2. Run install script by writing in your terminal: | **guile -s install.scm** |
| 3. If you haven't already, create a ~/220a directoty: | **cd ~** |
| | **mkdir 220** |
| 4. Copy scheme examples files into it: | **cd 220** |
| | **cp -r ~cc/220a/scm .** |

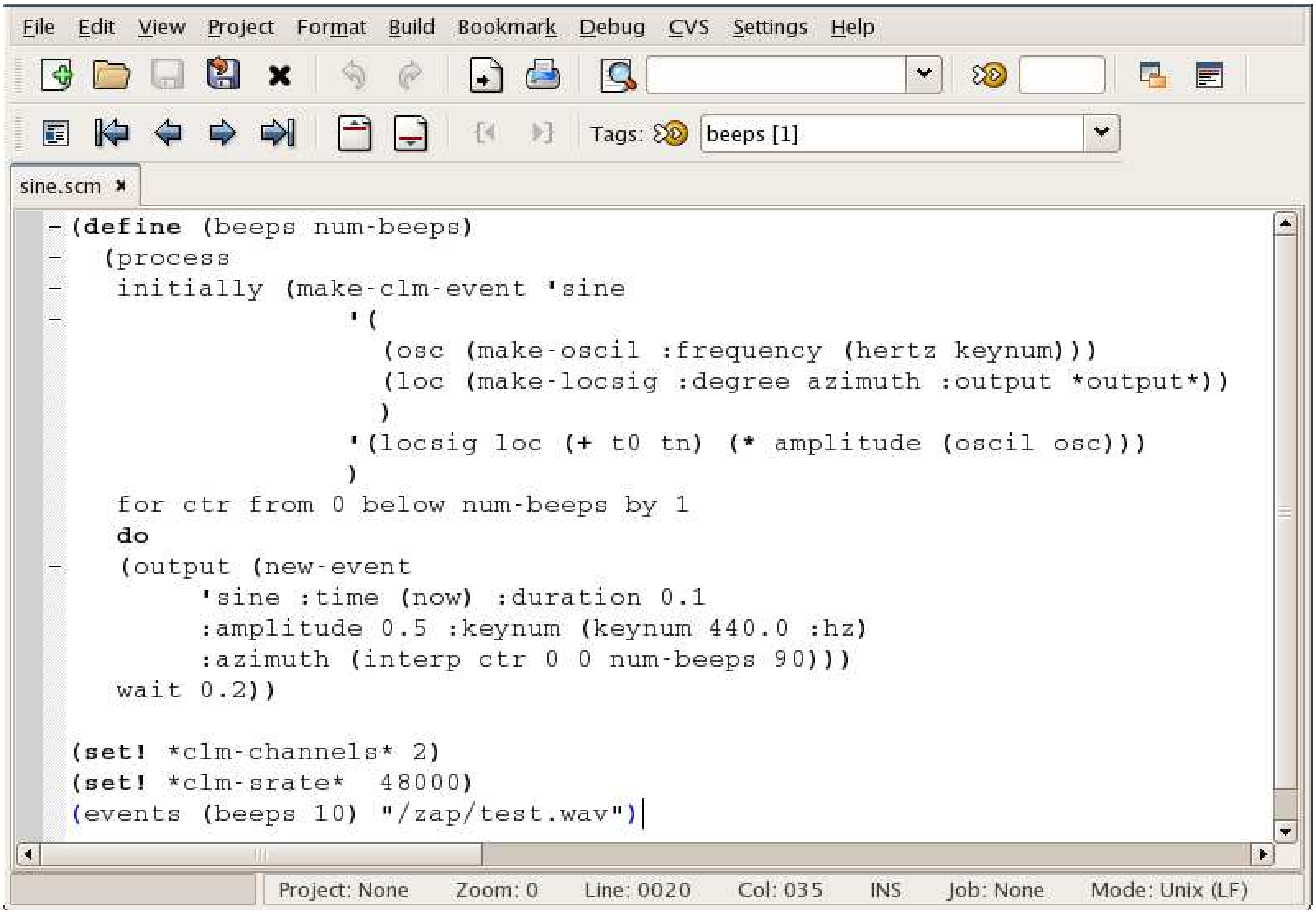
After these steps you will have a new directory in your home directory called "scripts-220a-user" which sets up tools for editing. And you will have a directory for working with 220a examples and stashing your own versions as they accumulate during the course work. Change now to your **scm/sine** subdirectory and type **ls** to list the contents of the directory. There should be several files. Then run *anjuta*, a fully customizable integrated editor, writing in your terminal: **anjuta &**


Anjuta: No file
File Edit View Project Format Build Bookmark Debug CVS Settings Help
Tags:
No tags

You can open mutliple documents from the directory at this point. The rest of this document is aimed at explaining the **sine.scm** file, but before that phase go through the 5 self-documenting tutorials called **minus-sine-0.scm** through **minus-sine-4.scm**. These introductory files progressively approach the features of scheme language and CM (Common Music) that will make the tones in **sine.scm**. To evaluate the code in a file you're looking at, click on Edit: snd-eval (or directly Shift+$ ). That will open a new *snd* window. If you don't yet see snd's guile listener window, click on View: Show listener. The window which appears is a guile session where the code in your editor window has been evaluated. Its current state contains any definitions that may have been specified in the code. You can continue to interact by typing in it directly. In most cases, you will just observe the result, then close snd, edit your code, and evaluate it again with changes. Make some simple changes and test them in each tutorial example. You might want to make a copy of the file under a different name first, so do a File: Save As. Two other operation hints: the editor automatically saves your edits when you evaluate. And you can ask the editor to indent your code with Edit: scheme-indent (then reload and click in the window).

When you're comfortable with those, open and evaluate **sine.scm**, which creates 10 beeps of 0.1 sec duration with a spacing of 0.2 secs between them. The beeps at 440 hz. go from the left channel to the right channel, controlled by the azimuth parameter. The only difference between this example and the

earlier **minus-sine-4.scm** one is that the sine values are sent into the soundfile rather than printing to the listener window. To do that we need one more unit generator in the list of ug's, so we've added **locsig** which directs sound into the output and can vary the position, channel-wise. A good thing to try is to go into a 4-ch studio and run this in a circle. All you need to change is the argument to CM's interp function that goes to 90 degrees in num-beeps, instead put 270 degrees and up the number of clm-channels to 4,
**sine.scm**

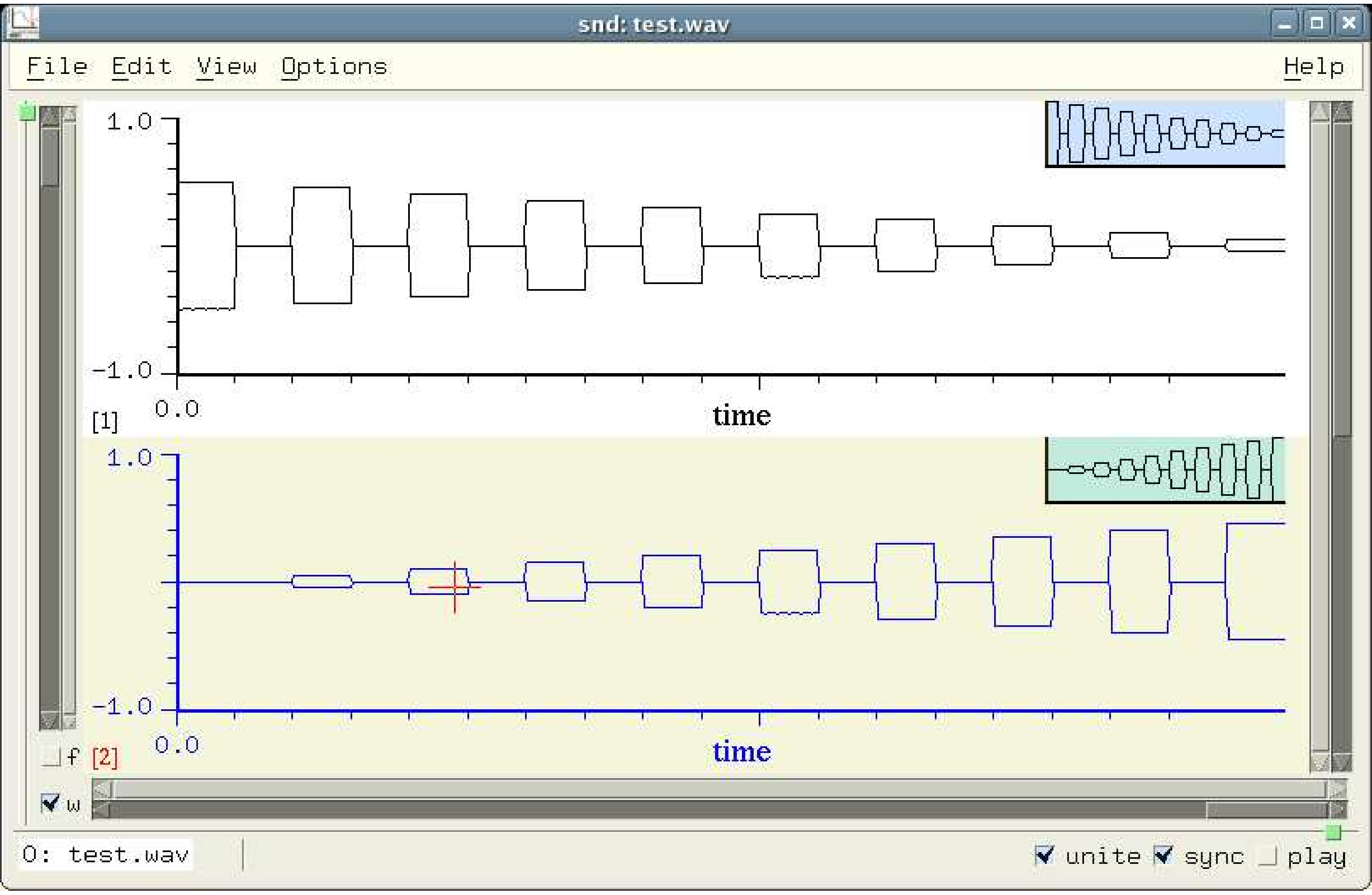

File Edit View Project Format Build Bookmark Debug CVS Settings Help
Tags: beeps [1]
sine.scm
Project: None Zoom: 0 Line: 0020 Col: 035 INS Job: None Mode: Unix (LF)

```
(define (beeps num-beeps)
  (process
   initially (make-clm-event 'sine
                    '(
                      (osc (make-oscil :frequency (hertz keynum)))
                      (loc (make-locsig :degree azimuth :output *output*))
                      )
                    '(locsig loc (+ t0 tn) (* amplitude (oscil osc)))
                    )
   for ctr from 0 below num-beeps by 1
   do
   (output (new-event
        'sine :time (now) :duration 0.1
        :amplitude 0.5 :keynum (keynum 440.0 :hz)
        :azimuth (interp ctr 0 0 num-beeps 90)))
   wait 0.2))

(set! *clm-channels* 2)
(set! *clm-srate*  48000)
(events (beeps 10) "/zap/test.wav")
```

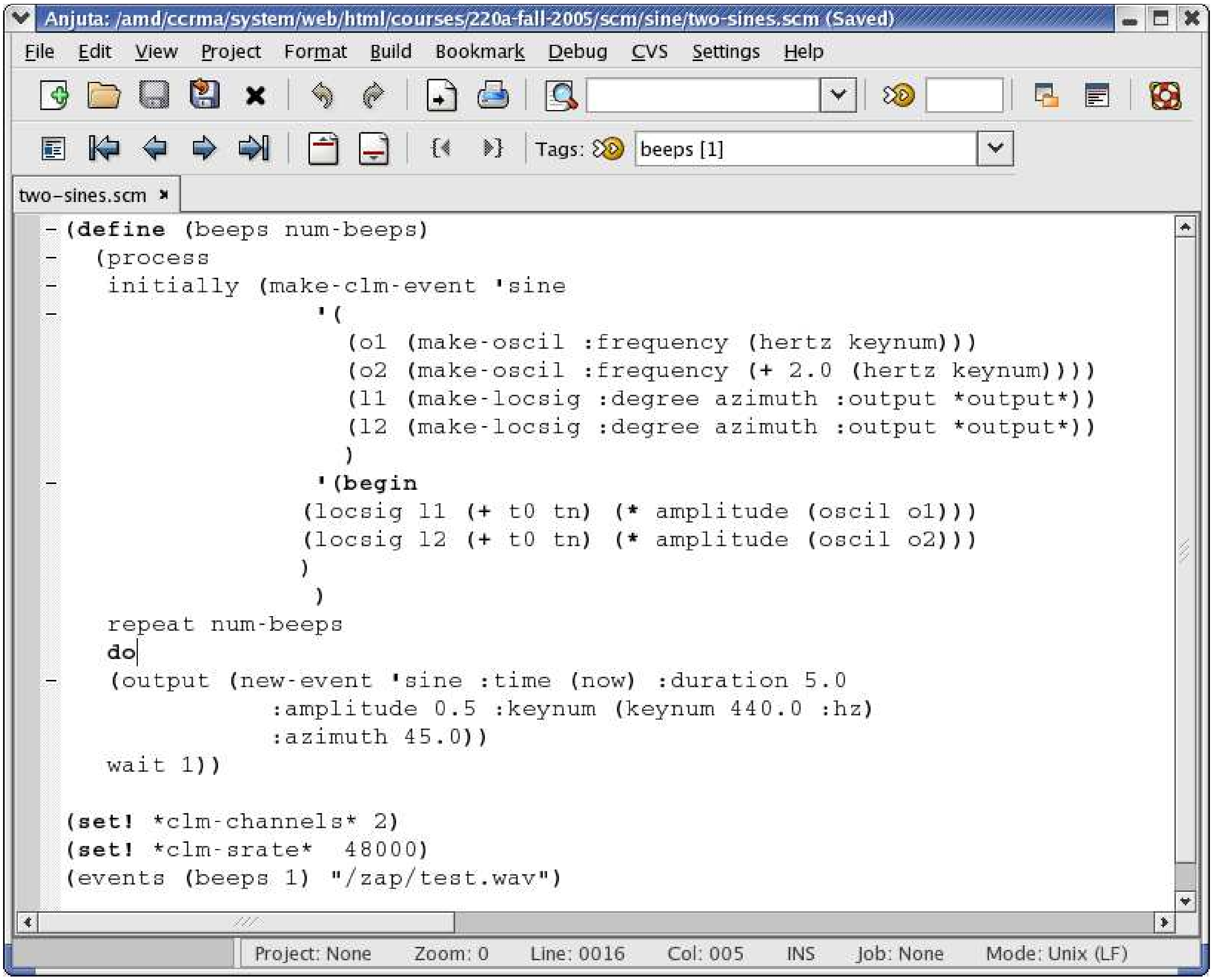
Bill Schottstaedt, snd author, says:

> ``Snd is a sound editor modelled loosely after Emacs and an old, sorely-missed PDP-10 sound editor named Dpysnd. It can accommodate any number of sounds each with any number of channels, and can be customized and extended using either Guile or Ruby."



In **two-sines.scm** file there are two sine waves of 5 seconds at a distance of two hz. Now the azimuth of both is 45 degrees (sound comes at equal amplitude from both speakers). Remember that azimuth goes from 0 (sound comes only from left speaker) to 90 (sound comes from right speaker).

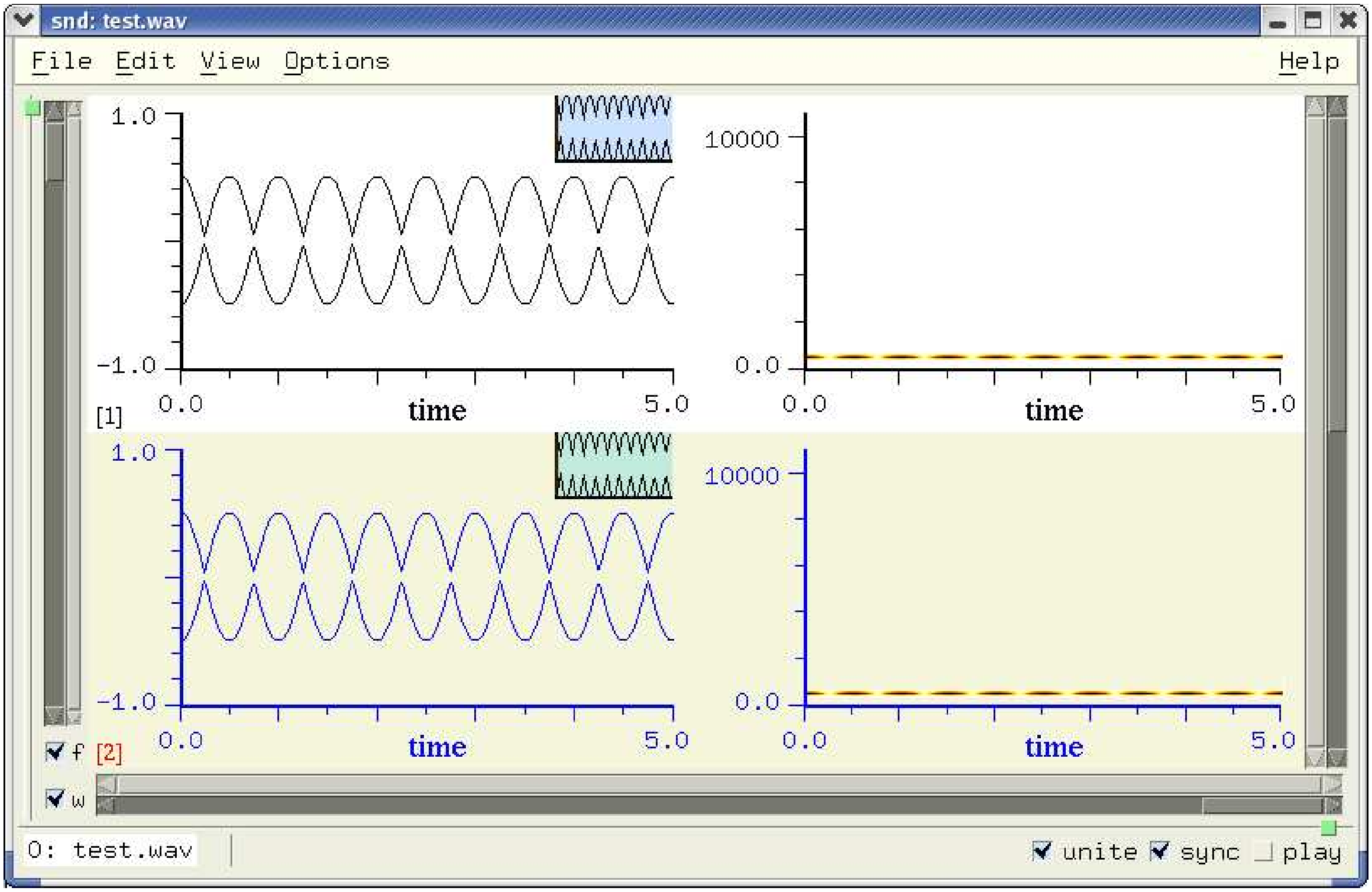
**two-sines.scm**


Anjuta: /amd/ccrma/system/web/html/courses/220a-fall-2005/scm/sine/two-sines.scm (Saved)
File Edit View Project Format Build Bookmark Debug CVS Settings Help
Tags: beeps [1]
two-sines.scm
Project: None Zoom: 0 Line: 0016 Col: 005 INS Job: None Mode: Unix (LF)

```
(define (beeps num-beeps)
  (process
   initially (make-clm-event 'sine
                  '(
                    (o1 (make-oscil :frequency (hertz keynum)))
                    (o2 (make-oscil :frequency (+ 2.0 (hertz keynum))))
                    (l1 (make-locsig :degree azimuth :output *output*))
                    (l2 (make-locsig :degree azimuth :output *output*))
                    )
                  '(begin
                 (locsig l1 (+ t0 tn) (* amplitude (oscil o1)))
                 (locsig l2 (+ t0 tn) (* amplitude (oscil o2)))
                 )
                  )
   repeat num-beeps
   do
   (output (new-event 'sine :time (now) :duration 5.0
              :amplitude 0.5 :keynum (keynum 440.0 :hz)
              :azimuth 45.0))
   wait 1))

(set! *clm-channels* 2)
(set! *clm-srate*  48000)
(events (beeps 1) "/zap/test.wav")
```

You can evaluate again the resulting sound file. Playing it you should hear a 2 Hz beating effect. Have a look in frequency domain by clicking on the **f** box (in the left corner). The default display is a single slice of time transformed into spectra. Sometimes it will be more useful to see the sonogram: go to Options and then click on Transform Options. By clicking on sonogram your right part of the snd window will change to a sonogram view of the sound file, with time-varying spectra. Also you will most likely see more if you specify that the transform should represent amplitude in dB and normalize. You can zoom in on frequency axis by mousing under the 10000 label and scrolling up.



Open another scheme file with similar 2 hz difference in the tones is **two-sine-stereo.scm** and for this you should be using headphones. What happens to the beating effect when the headphones are on properly, and then when you hold the heaphones close together but in front of your nose?

Next, check out ~/220a/scm/**impulse/click.scm** and look for the differences in the structure of the sound computation. Finally, study the file that combines two processes polyphonically, both sines and clicks, ~/220a/scm/**sine/two-sines-and-clicks.scm** and have a look at its sonogram. Zoom in and out, in frequency and time until you see a tic-tac-toe in spectra. Sines are the narrowest in frequency and clicks are the narrowest in time. Why would a click contain all frequencies?