SONIC PI 2.0

Get your groove on!

Discover new samples, synths, studio effects and Live Coding

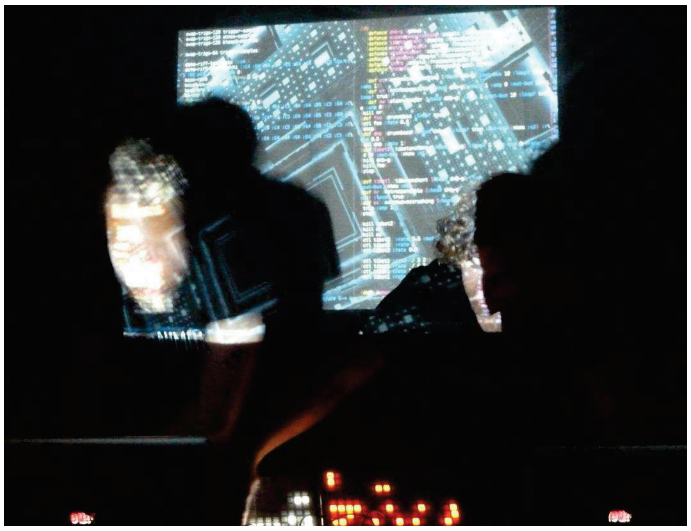
Samuel Aaron

Guest Writer

SKILL LEVEL : BEGINNER

Live Coding

The laser beams sliced through the wafts of smoke as the subwoofer pumped bass deep into the bodies of the crowd. The atmosphere was ripe with a heady mix of synths and dancing. However something wasn't quite right in this nightclub. Projected in bright colours above the DJ booth was futuristic text, moving, dancing, flashing. This wasn't fancy visuals, it was merely a projection of a terminal containing Emacs. The occupants of the DJ booth weren't spinning disks, they were writing, editing and evaluating code. This was a Meta-eX (http://meta-ex.com) gig. The code was their musical interface and they were playing it live.



Coding music like this is a growing trend and is often described as Live Coding (http://toplap.org). One of the recent directions this approach to music making has taken is the Algorave (http://algorave.com) - events where artists code music for people to dance to.

However, you don't need to be in a nightclub to Live Code. As one half of Meta-eX and author of Sonic Pi, I designed version 2 to give you the power to Live Code music anywhere you can take your Raspberry Pi and a pair of headphones, or some speakers. Once you reach the end of this article, you'll be programming your own beats and modifying them live. Where you go afterwards will only be constrained by your imagination.

Getting Sonic Pi v2.0

For this article you will need version 2.0 of Sonic Pi. You can tell if you are using version 2.0 from the opening splash screen. At the time of writing version 2 is in the final stages of development. You can get a copy of the latest release candidate, along with installation instructions, at http://sonic-pi.net/get-v2.0. When version 2.0 is released, you will be able to update with:

```
sudo apt-get install sonic-pi
```

You will then be able to open Sonic Pi by clicking on the main menu, and looking within `Education -> Sonic Pi`.

What's New?

Some of you may have already had a play around with Sonic Pi. Hopefully you all had fun making beeps of different pitches. You can take all the music coding skills you've learned with version 1 and apply it to version 2. For those of you that have yet to open Sonic Pi, now is definitely the time to give it a try. You'll be amazed with what you can do with it. Here's a quick list of the major new features:

* Ships with over 20 synth sounds
* Ability to play any wav or aiff sample file
* Ships with over 70 samples
* Extremely accurate timing
* Support for over 10 studio effects: reverb, echo, distortion, etc.
* Support for Live Coding: changing the code while it runs

Let's have a look at all of these features.

Playing a drum loop

Let's code up a simple drum loop. We can use the `amen` break to get us started. In the main code editor window of Sonic Pi, type the following and then hit the Run button:

```
sample :loop_amen
```

Boom! Instant drums! Go on, press it a few times. Have fun. I'll still be here when you've finished.

But that's not all. We can mess around with the sample. Try this:

```
sample :loop_amen, rate: 0.5
```

Oooh, half speed. Go on, try changing the rate. Try lower and higher numbers. What happens if you use a negative number?

What if we wanted to play the loop a few times over? One way to do this is to call sample a few times with some sleeps in between:

```
sample :loop_amen
sleep 1.753
sample :loop_amen
sleep 1.753
sample :loop_amen
```

However, this could get a bit annoying if you wanted to repeat it 10 times. So we have a nice way of saying that with code:

```
10.times do
  sample :loop_amen
  sleep 1.753
end
```

Of course, we can change the 10 to whatever number we want. Go on, try it! What if we want to loop forever? We simply say `loop` instead of `10.times`. Also, I'm sure you're asking what the magic 1.753 represents and how I got it. Well, it is the length of the sample in seconds and I got it because I asked Sonic Pi:

```
puts sample_duration :loop_amen
```

And it told me 1.753310657596372 - I just shortened it to 1.753 to make it easier for you to type in. Now, the cool thing is, we can combine this code and add a variable for fun:

```
sample_to_loop = :loop_amen
sample_rate = 0.5

loop do
  sample sample_to_loop, rate: sample_rate
  sleep sample_duration sample_to_loop, rate:
sample_rate
end
```

Now, you can change the `:loop_amen` to any of the other loop samples (use the auto-complete to discover them). Change the rate too. Have fun!

For a complete list of available samples click the Help button.

Adding Effects

One of the most exciting features in version 2.0 of Sonic Pi is the support for studio effects such as reverb, echo and distortion. These are really easy to use. For example take the following sample trigger code:

```
sample :guit_e_fifths
```

To add some reverb to this, we simply need to wrap it with a `with_fx` block:

```
with_fx :reverb do
  sample :guit_e_fifths
end
```

To add some distortion, we can add more fx:

```
with_fx :reverb do
  with_fx :disortion do
    sample :guit_e_fifths
  end
end
```

Just like synths and samples, FX also supports parameters, so you can tinker with their settings:

```
with_fx :reverb, mix: 0.8 do
  with_fx :distortion, distort: 0.8 do
    sample :guit_e_fifths
  end
end
```

Of course, you can wrap FX blocks around any code. For example here's how you'd combine the `:ixi_techno` FX and our drum loop:

```
with_fx :ixi_techno do
  loop do
    sample :loop_amen
    sleep sample_duration :loop_amen
  end
end
```

For a complete list of FX and their parameters click the Help button.

Live Coding a Synth Loop

Now we've mastered the basics of triggering samples, sleeping and looping, let's do the same with some synths and then jump head first into live coding territory:

```
loop do
  use_synth :tb303
  play 30, attack: 0, release: 0.3
  sleep 0.5
end
```

So, what do the numbers mean in this example? Well, you could stop it playing, change a number, then start it and see if you can hear the difference. However all that stopping and starting gets quite annoying. Let's make it possible to live code so you can instantly hear changes. We need to put our code into a named function which we loop:

```
define :play_my_synth do
  use_synth :tb303
  play 30, attack: 0, release: 0.3
  sleep 0.5
end

loop do
  play_my_synth
end
```

Now when you run this it will sound exactly the same as the simpler loop above. However, now we have given our code a name (in this case, `play_my_synth`) we can change the definition of our code while things run. Follow these steps:

1. Write the code above (both the define and loop blocks)
2. Press the Run button
3. Comment out the loop block (by adding # at the beginning of each line)
4. Change the definition of your function
5. Press the Run button again
6. Keep changing the definition and pressing Run!
7. Press Stop

For example, start with the code above. Hit Run. Comment out the `loop` block then change the `play` command to something different. Your code should look similar to this:

```
 define :play_my_synth do
   use_synth :tb303
   play 45, attack: 0, release: 0.3, cutoff:
70
   sleep 0.5
 end

 # loop do
 #   play_my_synth
 # end
```

Then hit the Run button again. You should hear the note go higher. Try changing the attack, release and cutoff parameters. Listen to the effect they have. Notice that attack and release change the length of the note and that cutoff changes the 'brightness' of the sound. Try changing the synth too - fun values are `:prophet`, `:dsaw` and `:supersaw`. Press the Help button for a full list.

There are lots of other things we can do now, but unfortunately I'm running out of space in this article so I'll just throw a couple of ideas at you. First, we can try some randomisation. A really fun function is `rrand`. It will return a random value between two values. We can use this to make the cuffoff bounce around for a really cool effect. Instead of passing a number like 70 to the cutoff value, try `rrand(40, 120)`. Also, instead of only playing note 45, let's choose a value from a list of numbers. Try changing 45 to `chord(:a3, :minor).choose`. Your play line should look like this:

```
 play chord(:a2, :minor).choose, attack: 0,
release: 0.3, cutoff: rrand(40, 120)
```

Now you can start experimenting with different chords and range values for cutoff. You can do something similar with the pan value too:

```
 play chord(:a2, :minor).choose, attack: 0,
release: 0.3, cutoff: rrand(40, 120), pan:
rrand(-1, 1)
```

Now throw some FX in, mess around and just have fun! Here are some interesting starting points for you to play with. Change the code, mash it up, take it in a new direction and perform for your friends!

```
 # Haunted Bells
 loop do
   sample :perc_bell, rate: (rrand 0.125,1.5)
   sleep rrand(0.5, 2)
 end
```

```
 # FM Noise
 use_synth :fm
 loop do
   p = play chord(:Eb3, :minor).choose -
[0, 12, -12].choose, divisor: 0.01, div_slide:
rrand(1, 100), depth: rrand(0.1, 2), attack:
0.01, release: rrand(0.1, 5), amp: 0.5
   p.control divisor: rrand(0.001, 50)
   sleep [0.5, 1, 2].choose
 end
```

```
 # Driving Pulse
 define :drums do
   sample :drum_heavy_kick, rate: 0.75
   sleep 0.5
   sample :drum_heavy_kick
   sleep 0.5
 end
 define :synths do
   use_synth :mod_pulse
   use_synth_defaults amp: 1, mod_range: 15,
attack: 0.03, release: 0.6, cutoff: 80,
pulse_width: 0.2, mod_rate: 4
   play 30
   sleep 0.25
   play 38
   sleep 0.25
 end
 in_thread(name: :t1){loop{drums}}
 in_thread(name: :t2){loop{synths}}
```

```
 #tron bike
 loop do
   with_synth :dsaw do
     with_fx(:slicer,freq: [4,8].choose) do
       with_fx(:reverb,room: 0.5, mix: 0.3) do
         n1 = chord([:b1, :b2, :e1, :e2, :b3,
:e3].choose, :minor).choose
         n2 = chord([:b1, :b2, :e1, :e2, :b3,
:e3].choose, :minor).choose
         p = play n1, amp: 2, release: 8,
note_slide: 4, cutoff: 30, cutoff_slide: 4,
detune: rrand(0, 0.2)
         p.control note: n2, cutoff: rrand(80,
120)
       end
     end
   end
   sleep 8
 end
```