SAM AARON

Sam is the creator of Sonic Pi. By day he's a research associate at the University of Cambridge Computer Laboratory; by night he writes code for people to dance to.
sonic-pi.net

Sonic Pi

LIVE CODING EDUCATION

Join Sonic Pi creator and live performer **Sam Aaron** as he introduces us to the joys of live-coding music on the Raspberry Pi...

You'll Need

- Raspberry Pi running Raspbian
- Sonic Pi v2.9+
- Speakers or headphones with a 3.5mm jack
- Update Sonic Pi: `sudo apt-get update && sudo apt-get install sonic-pi`

The lights sliced through the mist as the rhythms moved the crowd, the atmosphere ripe with a heady mix of synths. However, something wasn't quite right. Projected in bright colours above the performer was futuristic text, moving, dancing, flashing. This wasn't fancy visuals; it was merely a projection of Sonic Pi running on a Raspberry Pi. The musician wasn't spinning discs – she was writing, editing, and evaluating code. Live. This is live coding.

It may sound like a far-fetched scene from a futuristic music festival, but is actually a description of a pupil performing at their primary school assembly. Coding music like this is a growing trend and is often described as live coding. However, this approach to manipulating code isn't just useful for performance; it's also a fabulous way of engaging a new generation of coders in the classroom.

Engaging a new generation of coders

Across the educational landscape, it's becoming clear that teaching programming is an increasingly important aspect of our curriculum. What is not obvious, however, is exactly how we go about engaging students in order to effectively teach the core fundamentals of programming. One approach to solving this problem has been to ask expert programmers what excited them when they started learning. This has yielded responses ranging from sorting algorithms to binary arithmetic. Whilst these may be deeply interesting and engaging topics for computer scientists, it's not clear that they are effective topics for exciting a broad and diverse range of students in introductory computing. This is especially significant considering that most students will not embark in a career in programming in either industry or academia.

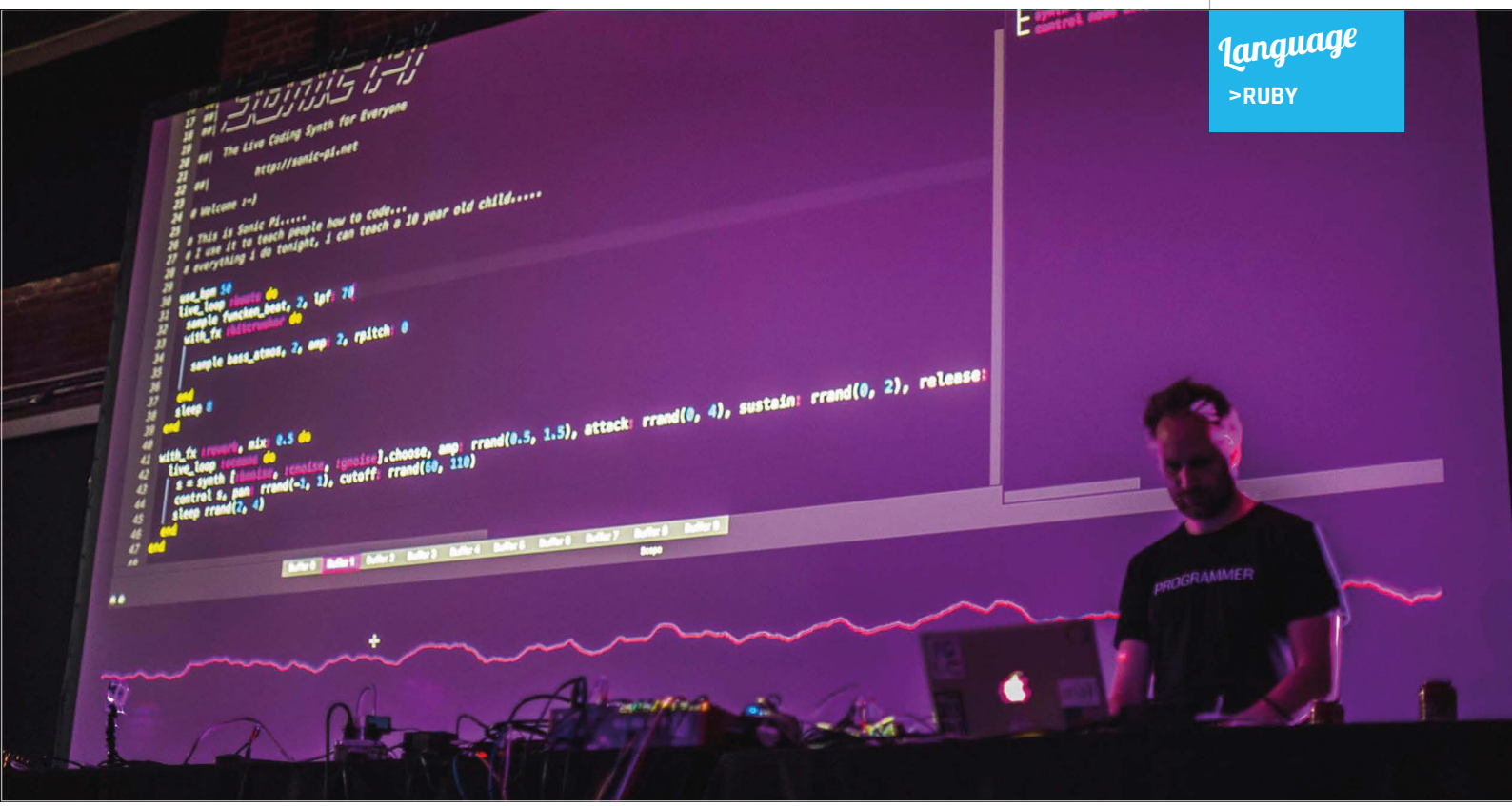
Sonic Pi takes a different approach. It turns code into a powerful new kind of musical instrument with a focus on fast feedback and iterative learning. It enables students to code the kinds of music they're typically used to listening to. If this claim sounds a little ambitious, take a look at this quotation from the *Rolling Stone* magazine (**magpi.cc/2ciPcr**) covering a recent Sonic Pi performance at Moogfest USA:

"[The set] – which sounded like Electric Café-era Kraftwerk, a little bit of Aphex Twin skitter and some Eighties electro – was constructed through typing and deleting lines of code. The shadowy DJ sets, knob-tweaking noise and fogbank ambient of many Moogfest performers was completely demystified and turned into simple numbers and letters that you could see in action. Dubbed 'the live coding synth for everyone', it truly seemed less like a performance and more like an invitation to code your own adventure."

Powerful yet simple

Of course, there's very limited educational value in a system that's powerful yet incredibly difficult to learn. However, Sonic Pi was conceived and built within classrooms in close collaboration with teachers and has benefited from many design iterations based on hundreds of hours of student observation. Simplicity has even been baked into

Language
>RUBY



Above Sonic Pi creator Sam Aaron performing at Moogfest 2016

the core design philosophy by only allowing features that may be easily understood by and taught to a ten-year-old child.

For example, in order to get started, you only need to learn two extremely simple commands: **play**, which allows you to play different notes, and **sleep**, which enables you to choose how long to wait before playing the next note. Once you've mastered **play** and **sleep**, the next command to learn is **sample**, which gives you the ability to play any pre-recorded sound. Sonic Pi includes a large number of built-in recordings to use, such as drums, guitars, and atmospheric noises. However, the real fun starts when you record your own sounds to play back and manipulate. Finally, you also have access to a full set of professional studio effects, such as echo and reverb, to manipulate your sounds. For instance, playing a sample of a guitar with reverb is as simple as:

```
with_fx :reverb do
  sample :guit_harmonics
end
```

Explore the full computing curriculum

Considerable care and attention has been placed to ensure that Sonic Pi allows educators to deliver all of the core concepts in the UK's new computing curriculum. It is an ideal follow-on language to Scratch due to its simple block-like, text-based syntactic structure. It is already seeing extensive use worldwide in all levels of education, from primary through to university. There are even primary schools which encourage their pupils to give performances with Sonic Pi during assemblies.

There is already a lot of support for educators who wish to use Sonic Pi in the classroom. You can easily use Sonic Pi to teach basic computing concepts such as sequencing, iteration, selection, functions, data structures, and algorithms. However, rather than teaching these ideas in a distant, abstract way, you are invited to deliver them using a simple, engaging musical narrative which gives the constructs extra meaning and motivation. For example, instead of sequencing, you could teach melodies; instead of iteration, repeating rhythms; instead of lists, bass lines or riffs. This is possible because each and every musical idea in Sonic Pi is immediately represented by one or more core computer science techniques. You can even go much further than the basic curriculum and explore concurrency, determinism, and even live coding. Before we dive into these advanced topics, let's take a quick look at how Sonic Pi brings new meaning to the basic concepts of sequencing, iteration, and selection.

> "You can easily use Sonic Pi to teach a range of basic computing concepts"

Sequencing melodies

From the outset, Sonic Pi encourages the learner to code sequences of instructions. This is because the most simple programs consist of sequencing calls to **play** and **sleep** to create simple melodies:

```
play 70
sleep 1
play 75
sleep 0.5
play 82
```

What's exciting here from a musical perspective is that by sequencing **play** and **sleep** like this, you have access to most of Western music. In other words, with just two commands you can recreate any known melody or rhythm, and of course you're entirely free to compose your own.

Iterating rhythms

Iteration is also very simple to represent and work with. For instance, imagine we wish to play a bass drum five times followed by a cymbal three times:

```
5.times do
  sample :bd_haus
  sleep 0.5
end

3.times do
  sample :drum_cymbal_open
  sleep 1
end
```

In the example above, all the code between the first **do** and **end** lines is repeated five times, and the code within the second block is repeated three times. We therefore play the **:bd_haus** sample and wait for half a second repeatedly five times, before finishing by playing the **:drum_cymbal_open** sample and waiting for one second repeatedly three times.

Below It's amazing what interesting sounds can be created with just a few lines of code

Selecting sounds

Selection is also easy to represent. Let's use some randomisation to repeatedly choose whether to play a cowbell or a snare drum with a one in two chance of playing either percussion instrument:

```
loop do
  if one_in 2
    sample :drum_cowbell
  else
    sample :drum_snare_hard
  end
  sleep 0.125
end
```

In this example we use an infinite **loop** to repeatedly call the function **one_in** with the parameter **2**. This will return **true** or **false**, with a 50% chance of either. We then use this value to select one of two possible samples to play. If **one_in 2** returns **true** we hear a cowbell, otherwise we hear a snare drum. We then wait for **0.125** seconds before repeating. Executing this code creates an interesting non-repeating rhythm and is the basic building block of a probabilistic drum sequencer.

A brief introduction to live coding

In order to get a taste of what Sonic Pi can do, let's now dive straight into the deep end and take a look at one of the most exciting aspects on offer: live coding. This is a new style of programming that provides many new learning opportunities in the classroom due to its immersive nature and incredibly fast feedback cycle. The basic idea is simple: in addition to the traditional start/stop model of programming, we can also continually tweak and modify the program as it runs. This allows us to rapidly explore the effects of small tweaks to the program, which both encourages tinkering and increases engagement. For example, once you have mastered the live coding workflow, it is easy to get into a flow-like state for long periods of time, modifying and remodifying the code.

The live loop

The key to live coding is mastering Sonic Pi's unique programming construct, the **live_loop**. At a quick glance, it looks no more complicated than a 'forever' block in Scratch. Let's look at one:

```
live_loop :beats do
  sample :bd_haus
  sleep 0.5
end
```

There are four core ingredients to a **live_loop**. The first is its name. Our **live_loop** above is called **:beats**. You're free to call yours anything you want. Go crazy. Be creative. I often use names that

communicate something about the music I am making to the audience. The second ingredient is the **do** word, which marks where the **live_loop** starts. The third is the **end** word, which marks where the **live_loop** finishes, and finally there is the body of the **live_loop**, which describes what the loop is going to repeat – that's the bit between the **do** and **end**. In this case we're repeatedly playing a bass drum sample and waiting for half a second. This produces a nice regular bass beat. Go ahead, copy it into an empty Sonic Pi buffer and hit **Run**. Boom, boom, boom!

Redefining on-the-fly

OK, so what's so special about the **live_loop**? So far it just seems like a glorified **loop**! Well, the beauty of **live_loop**s is that you can redefine them on-the-fly. This means that whilst they're still running, you can change what they do. This is the secret to live coding with Sonic Pi. Let's have a play:

```
live_loop :choral_drone do
  sample :ambi_choir, rate: 0.4
  sleep 1
end
```

Type in the code above and then press the **Run** button or hit **ALT+R**. You're now listening to some gorgeous choir sounds. Now, whilst it's still playing, change the rate from **0.4** to **0.38**. Don't hit **Stop** but instead hit **Run** again. Whoa! Did you hear the choir change note? Change it back up to **0.4** to return it to how it was. Now, drop it to **0.2**, down to **0.19**, and then back up to **0.4**. See how changing just one parameter on the fly can give you real control of the music? Now have a go at playing around with the rate yourself. Choose your own values. Try negative numbers, really small numbers, and large numbers. Most importantly, have fun!

Sleeping is important

One of the core lessons about **live_loop**s is that they need rest. Consider the following **live_loop**:

```
live_loop :infinite_impossibilities do
  sample :ambi_choir
end
```

If you try running this code, you'll immediately see Sonic Pi complaining that the **live_loop** did not sleep. This is a safety system kicking in! Take a moment to think about what this code is asking the computer to do. That's right, it's asking the computer to play an infinite amount of choir samples in zero time. Without the safety system, the poor computer will try to do this and crash and burn in the process. So remember, your **live_loop**s must always contain a **sleep**.

Above Sonic Pi can be used alongside traditional musical instruments

Combining sounds with concurrency

Music is full of things happening at the same time. Drums at the same time as bass at the same time as vocals at the same time as guitars.... In computing we call this concurrency, and Sonic Pi provides us with an amazingly simple way of writing concurrent code. Just use more than one **live_loop**!

```
live_loop :beats do
  sample :bd_tek
  with_fx :echo, phase: 0.125, mix: 0.4 do
    sample  :drum_cymbal_soft, sustain: 0, release: 0.1
    sleep 0.5
  end
end

live_loop :bass do
  use_synth :tb303
  synth :tb303, note: :e1, release: 4, cutoff: 120, cutoff_attack: 1
  sleep 4
end
```

Here, we have two **live_loop**s: one looping quickly making beats, and another looping slowly to make a crazy bass sound.

One of the interesting things about using multiple **live_loop**s is that they each manage their own time. This means it's really easy to create complex polyrhythmical structures and make very interesting music with only a few lines of code.

Use Sonic Pi in your classroom today

This ends our whistle-stop tour of live-coding music with Sonic Pi. If you are excited about this new approach to engaging students with computing, you can download the app from **sonic-pi.net**. It is completely free and cross-platform, working identically on Windows, Mac, and the Raspberry Pi. It comes with a complete built-in tutorial which assumes you know nothing about either code or music. There is also an accompanying (free) book published by *The MagPi* (**magpi.cc/1VGIOux**) which includes many ideas and invitations to experiment and learn. Finally, there is a full scheme of work targeting the new computing curriculum, which is free to download from the Raspberry Pi website.