Creative Coding - Week 3

Making Music with Strudel.cc

Lecturer: Dr Michael Cook (mike.cook@kcl.ac.uk)

Figure 1: A group called 'algobabez' livecoding at an algorave. Some of the code can be seen projected behind them (this is not Strudel however) Photo credit: smokeghost/flickr.

In this week's session we're going to look at *livecoding*, a way of writing music using a domain-specific programming language. The *live* part refers to the fact that this language can be partially recompiled at runtime, which means that you can write code which creates music, and then change and execute part of that code without interrupting the rest of the sounds being produced. Live-coders use this to perform in real-time, coding on stage at events called *algoraves* (see the picture above). Livecoding tools can also be used for writing static songs or producing at home, but their main appeal is for improvisational music performance.

In today's session we're going to be learning how to use Strudel. Strudel is a Javascript port of a language called TidalCycles, which is a very popular livecoding language. TidalCycles was developed by Alex McLean, who lives in Sheffield and regularly performs at algoraves around the UK. Because Strudel is Javascript-based, it also runs in the browser, which makes it very convenient for us. **You'll need headphones to make the most of today's session.** You don't need music knowledge though, just a willingness to get messy. Find Strudel here:

https://strudel.cc/

1 Running Strudel

Programs in Strudel won't look like any other code you've ever written anywhere else, but you'll get the hang of it. First of all, when you load the website up, you may see an example program written by a member of the Strudel community. To get a feel for what Strudel can do, press 'play' in the top-right of the window, or hit CTRL+ENTER. Music should start to play!

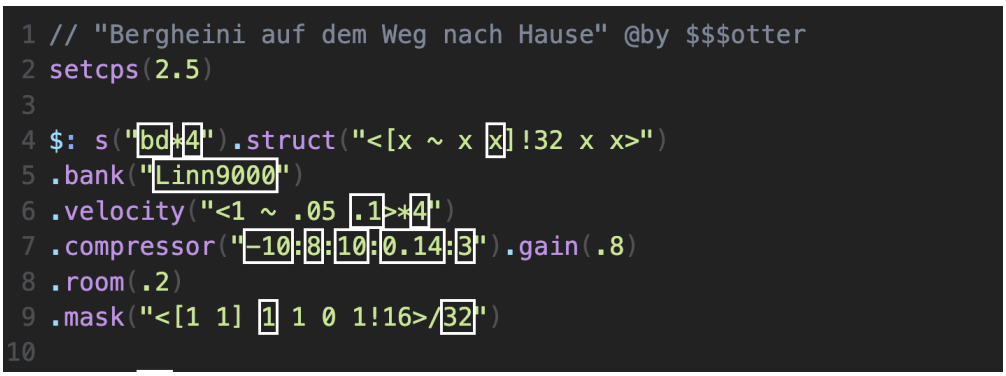

Figure 2: A Strudel program during execution.

As the music plays, you'll see white boxes appear over parts of the program. This is showing you which bits of the program are being excecuted right now. In Figure 2 you can see several areas are active. In the last line, for example, the white box around the third 1 digit shows that we are in this part of the sequence of numbers. We'll get to what these sequences mean in a bit. Feel free to listen to the song play for as long as you like. Some songs have a beginning and end, but most play forever. CTRL + . (the full stop/period symbol) traditionally stops Strudel, but your browser/platform may vary. Before you start, try playing one of the examples under the Patterns tab. Here's a recent one.

1.1 Writing Strudel: Basic Sounds

Delete all the code in this program and write the following (remember to click 'play', or press CTRL + ENTER when you're done):

```
sound("bd")
```

This is the simplest possible Strudel program - it tells the computer to play a bass drum sound (`bd`) once per *cycle*. You can think of a cycle like the framerate in a p5js sketch. If a cycle if four seconds long, then this pattern will play one bass drum note, then wait until four seconds have passed. This means that if we give Strudel more to play, it will play them faster, because the whole program is fit into one looping cycle. Try this:

```
sound("bd bd bd bd bd bd bd bd")
```

This is now playing eight bass drum sounds in the same space of time. Here's a list of other drum kit sounds you can use, try playing around with them for a bit, adding and removing them from your program, and seeing what you can make:

```
/*
    bd = bass drum
```

```
    sd = snare drum
    rim = rimshot
    hh = hihat
    oh = open hihat
    lt = low tom
    mt = middle tom
    ht = high tom
    rd = ride cymbal
    cr = crash cymbal
    - = rest (play nothing)
*/
```

Here's a little example:

```
sound("bd - cp - bd bd cp hh")
```

We can also specify what *kind* of drum machine we want it to use samples from when it plays, for example, the sound of a bass drum. If you click "drum-machines" you'll see a list of all the drum machines that Strudel has samples from. Try this:

```
sound("bd - cp - bd bd cp hh").bank("RolandTR909")
```

`.bank` is an example of a modifier that you tack onto the end of a command in Strudel to change it in some way. There are a lot of modifiers, that do all kinds of different things, and can be applied in many different ways. I understand about 2% of them! You'll learn more by looking at examples and experimenting. Try this:

```
sound("bd - cp - bd bd cp hh")
    .bank("RolandTR909")
    .fast(2)
    .room(0.5)
    .lpf(1000)
```

`.fast(2)` plays the sequence at double the speed of a regular cycle. It has a counterpart, `slow`, and you can pass it any number (including negative and decimal numbers). `room(0.5)` is reverb - it adds an echo effect. 0.5 is a light effect, you can increase the number to make it stronger. `lpf` is a low pass filter. This removes any component of the sound's frequency that is higher than the number given, in hz. Low pass filters make a sound muffled, like you're hearing it on the other side of a nightclub door. Try running the above example, then changing `lpf` to `hpf` – a high pass filter, which does the opposite – to see the difference. In Strudel there are usually two or three ways to write exactly the same thing, like `fast()` and `slow()`.

2 Writing Strudel: Basic Notes

Drum kits are *samples*, recordings of specific noises (like hitting a bass drum) that we play on command and modify. Other instruments are synthesised, meaning that we can ask for a specific musical note that sounds like it was played through that instrument. This uses a different syntax to the drum kit, but the principle is the same. First, specify a sequence (this time of numbered notes) and then an instrument to use:

```
note("48 49 50 51 52 53 54 55").sound("piano")
```

You can also write letters to play notes that way too, and add flats ('b') and sharps ('#') to play the black piano keys.

```
note("d e f# g a bb c").sound("piano")
```

We can also define the scale a sequence is played in which is useful for lots of effects. Note that this uses `n` rather than `note` which is a different function for reasons I could not figure out before writing this workshop:

```
n("0 1 2 3 4 5 6")
  .scale("C:minor")
  .sound("piano")
```

You can find a full list of instruments under the 'sounds' tab on the right-hand side of Strudel's main page. There are a lot of them! Click them to hear an example sound from one. Note that some instruments can't play certain kinds of sequence, or can't express different keys/scales. Try things out and see how it goes!

3 Sequences and Compositions

Let's go back to our bass drum. Try this sequence:

```
sound("bd bd bd bd bd bd bd bd")
```

This plays eight bass drum hits in one cycle. Now try this:

```
sound("[bd bd bd bd] bd")
```

By wrapping four bass drum sounds in [square brackets], we tell Strudel to treat it like a single item. So the sequence now has two things in it: one is a subsequence of four bass drum hits, and one is a single hit. It gives half a cycle to each of these sequences. Then, *within that subsequence*, the four bass drums get one quarter of that time period each. So you get four sounds at one eighth of a cycle, and one at half a cycle. Ok, now try this:

```
sound("<bd sd hh cp> bd")
```

Angled brackets are treated like a rotation of instructions. Each time Strudel comes to execute the contents of what's in the angled brackets, it takes the item after the one it chose last time. We can combine these:

```
sound("bd <[hh hh] hh> - bd [hh <hh rim>] -")
```

Strudel has a lot of other ways to play with sequences. We can multiply (`*`) and divide (`/`) sequences by numbers to change their duration too, a bit like using `fast()`. We can play two sequences in parallel by separating them by commas (but we'll see other ways to do multiple sequences later). Here's a nice example from the Strudel workshop page:

```
sound("bd*4, [- cp]*2, [- hh]*4").bank("RolandTR909")
```

The clap (`cp`) plays at half the speed of the other two sequences, and square brackets are used to make some samples play half as long, and at the end of the beat to create a syncopated effect (I think). This is all really cool but Strudel's expressiveness gets even richer. One of the reasons why almost everything in Strudel is expressed as strings passed to functions is that anything in Strudel

can be turned into a varying sequence. All of the rules we just learned above can be applied to *any string* in Strudel. For example:

```
sound("bd - cp - bd bd cp hh")
    .bank("RolandTR909")
    .fast("<2 1>")
    .room("[1 0.5]")
    .lpf("1000 800")
```

Run this and you can see how the `fast` command varies between two speeds, because we're changing it once per cycle with the angled brackets, and how the room effect changes twice within a single cycle because we're telling it to vary between two values. This can be used to write really compact, complex sequences, which is also really handy when you're changing something live.

4 Samples

One last thing for this week, as a bit of extra fun for you when you start making things. Strudel allows you to load your own sample banks into your file, and there are a lot of open-source sample banks available on the internet. You can find some in the example songs, but I've also prepared some samples you might want to use today:

```
samples('github:gamesbyangelina/samples')
```

Important! Note that these are single quotes, not double, around the GitHub link. Strudel is very fussy about this sometimes.

For this section, you can also find some sample code I wrote here.

This loads samples direct from a github repo. The second part is my github username, and the last part is the repo name. There are other ways to load samples - ask me or look up the 'samples' section of the Strudel site for more info.

```
s("charli:0")
```

My repo has a `strudel.json` file that defines sample groups and links them to files in the repo. So this creates a new sample bank called 'charli'. By appending a colon and a number, you can play that number sample from the pack. The above code plays the first (0th) sample. Samples behave like any other sound, you can apply modifiers and do whatever with them. For example:

```
s("charli:2").slice(2, "0 1 0 1 0 0 0 1")
```

`slice` is one of my favourite functions. It takes two arguments, the first is a number to chop the sample into - so in this case, two parts. The second argument is a pattern to play using those parts. So this chops the sample into two, and then plays them according to the sequence I've given. Try it out. There are several sample banks in this set:

- `vg` (0-14) - samples of videogame sound effects
- `charli` (0-11) - samples of Charli XCX
- `chap` (0-7) - samples of Chappell Roan
- `comp` (0-15) - samples from a radio broadcast about computers

5 This Week: Create a 10 Second Loop

Strudel is a lot harder to code with than p5js (if you were here for those lessons) because its syntax is very compact and doesn't directly correspond to an output. You'll spend a lot of your time running code just to see what it does. That's okay! A lot of livecoders do the same even during performances - they'll try something, realise they don't like it, and undo the change all on stage.

This week your goal is to make a ten second loop of some kind and (optionally) share it with us at the end if you feel comfortable. You can do this task entirely using the drum kit and just focus on creating a really nice layered beat, but if you want to experiment with melodies or samples please go for it!

The Strudel learning page is very useful and has a lot of examples for every single feature of the language. If you need more inspiration, head on over there:

https://strudel.cc/workshop/first-sounds/

6 What Next

6.1 Share your work with me!

I'd love to see (and hear) what you make, and I'd also love to be able to share people's work at future department events, or show to students in future years for inspiration! I would love it if you shared what you make - anonymously if you prefer, or with your name for full credit.

Ways you can share your work:

- Email the code, or a link to your Strudel project to (mike.cook@kcl.ac.uk)
- Post it to GitHub and share with me (username: gamesbyangelina)
- Drop it anonymously in this form: https://forms.gle/qh8UWnqW1fNJyfxz8.

6.2 Sharing online

Strudel allows you to save and share your work with others, using the 'share' button in the top right of the screen. The livecoding community is small but very dedicated and helpful. Strudel has a number of Discords dedicated to it where performers and the tool developers answer questions!

6.3 Further Reading

London has algoraves on regularly, which you can attend usually very cheaply (or for free) and see livecoders perform – I can almost guarantee you'll see a Strudel performer, or at least a TidalCycles performer, at any algorave. A popular place in London for them is Corsica Studios. Most algoraves have an open mic session at the start for an hour where *anyone* can plug in a laptop and perform for a few minutes. It's a great way to try out livecoding!

Some people also use Strudel/TidalCycles for making recorded/static songs, or record their live work and put it online. Graham Dunning is a nice example of this, on Bandcamp here.

There are also a lot of guides and performances recorded on YouTube. Watching someone make music live is really useful for seeing what techniques they use. I went to a workshop run by Lucy Cheesman (Heavy Lifting) who said that for most of her performances she uses maybe ten different effects and a few drum kits and instruments. You don't need to know everything about music, or everything about Strudel, to make great sounds - just get comfy with a little palette and see what you can make with it!

A Creating Samples

This is a quick guide to making and uploading your own samples so you can use them in Strudel. I don't recommend doing this during our lesson as it takes time, but you can come back to it later!

A.1 Step 1: Acquire Samples

First you need to get the music files you want to play! I put mine in mp3 form, but that might not be optimal if you're an audio person. I got them from a variety of different sources, and you'll probably have your own ideas too!

- **Archive audio** I found on Archive.org which is a treasure trove of all sorts of audio. I searched for terms like 'computer', downloaded longer recordings, and chopped them up.
- **Videogame samples** I found on Sounds Resource which has a lot of old/classic videogame sound effects and music. Often they come entirely unlabelled.
- **Music samples** I found on YouTube, and downloaded them using yt-dlp, a successor to youtube-dl. In particular, I searched for 'acapella' or 'karaoke' along with the song title, as often people have already done the work of extracting the audio/instrument tracks from songs.

Once your have your audio, you may need to chop it up into something smaller to extract the bits you want. Especially for Strudel, samples tend to be better if they are shorter. I recommend using Audacity for this, an extremely powerful and free audio editing tool. I won't go into a tutorial for that here, but the short version is:

- Open the file in Audacity.
- Highlight the bit you want by clicking and dragging on the track (you may need to zoom in).
- Go to File → Export and choose *Selected Audio* – really important you remember to click this, otherwise it will export the entire thing. I forget this about 50% of the time.

A.2 Create Repository

You can load samples into Strudel locally, but it's easier to put them online. The easiest way is on GitHub. You need to create a repository and upload your samples to it. However, you also need a special file that tells Strudel where to look for everything, which must be called `strudel.json`. Here's an example:

```
{
    "_base": "https://raw.githubusercontent.com/gamesbyangelina/samples/main/",
    "categoryname":
    [
        "sample1.mp3",
        "sample2.mp3",
    ],
    "categoryname2": "sample3.mp3"
}
```

In the above example, this would create two sample packs when loaded into strudel. The one called `categoryname2` just has a single sound in it, so when you call it it plays that sound. The once called `categoryname` has two sounds in it, so you can ask for `categoryname:0` and `categoryname:1` to play different sounds. How you organise your samples is up to you.

You can find my strudel.json file by clicking here. You can find better examples of strudel.json files on the Strudel workshop pages.

A.3 Including Samples

Once you've added the samples files and the json file to your repository, and uploaded it to GitHub, you can now reference it in your Strudel files just like I did. The exact line will depend on your username and the repository name:

```
samples('github:yourusername/yourreponame')
```

Then you can start playing samples, based on the category names you gave. Note that Strudel first has to download the sample to play it, so it may take a few seconds. Strudel also caches things, so if you change your repository you may need to force a refresh (hold SHIFT and press refresh on the page) to try and trigger it to reload. If something seems wrong, click the Console button on the right-hand side of Strudel to see if there are any useful messages.