xCoAx 2023 11th Conference on Computation, Communication, Aesthetics & X
2023.xCoAx.org
Weimar, Germany

c0l1v3: A Collaborative Nonlinear Live Coding Instrument

Bruno Gola
me@bgo.la
Universität der Künste, Berlin, Germany

DOI 10.34626/xcoax.2023.11th.414

c0l1v3 is an improvisational, collaborative, interactive audiovisual performance, and a live coding web-based instrument created by Bruno Gola. The collaboration aspect is implemented in a way that anyone with a device running a web browser can join at any time and play. There is no login or credentials, *c0l1v3* is always running and accessible via its URL. The website works as a frontend to a SuperCollider program running on the server-side. It streams audio and data back to the browser with low latency using WebRTC, so all players are controlling the same sound process. *c0l1v3*'s interface is an experiment in breaking the traditional top-to-bottom linear code writing approach, it presents a canvas where players can add code anywhere, and all players can edit each other's code blocks. In *c0l1v3* there is no distinction between performers and audience.

Keywords: Live Coding, Nonlinear, Audiovisual, Network Music, Collaborative Art, Computer Music, Distributed Performance, Improvised Music.

Description

c0l1v3 is an experiment in collaborative improvised live coding and less traditional ways of collectively writing code to control an audio-visual instrument. It is always available as an open playground on the web, working as an online public square where anyone can enter and listen to others playing and jam together.[1]

Background

During the beginning of the COVID-19 pandemic I started working together with Prof. Alberto de Campo in the context of the *Generative Kunst* / Computational Arts class at the Berlin University of the Arts to develop tools that could support the class on continuing working in its hands-on, live experimentation practice even remotely. During the two months before the beginning of the summer semester of 2020, we developed *tcposcrouter*, *ws2udp* and *HyperDisCo*, all tools that make it simple to collaborate in *SuperCollider* live coding sessions over the internet.

Simultaneously I was playing in different live events online, such as algoraves, and watching many live streamed performances in different formats. During that time I was getting frustrated with the lack of direct feedback in my own performances, it was hard to feel like I was playing live or that there was an audience on the other end, so I started imagining ways to make my performances more alive and reactive.

In the beginning of 2021, as part of the On-The-Fly program, I had the opportunity to be Artist-in-Residency at Ljudmila Art and Science Lab, in Slovenia. There I started working on the project that became *c0l1v3*. During the residence my goal was to build an installation called *Co-op{codes}*. In this installation the visitors could experiment with live coding: there were two computers running the same software, and with those two computers the audience could control the audiovisual instrument that I developed using *SuperCollider* which consisted of 3 projectors and 8 speakers. The software interface was designed in a way that anyone could experiment by just double clicking anywhere on the screen to add a random piece of valid code. Each block of code added to the interface had its own editing space. The interface followed no order between blocks, and each block could be executed independently. The software was running on a web browser, both computers were connected via the Wi-Fi and changes made on each computer would be shown simultaneously on the other.

1. https://c0l1v3.bgo.la/

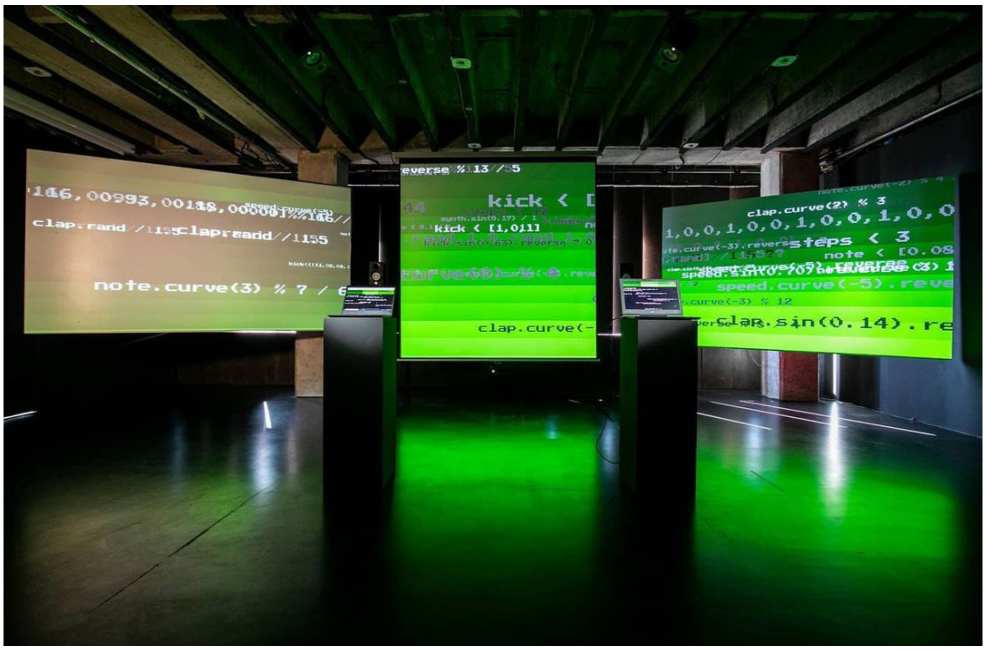


Figure 1: *Co-op{codes}* installation in Ljudmila, Slovenia.

Distributed Performance

At the end of my residency in Ljudmila I took part in the *Algopolis* event by performing live with *Co-op{codes}*. That was a solo performance but opened the way to the next step of that research, transforming the installation into this open, web based instrument, that could be online and I could perform at any time with it, and also invite the audience to join me writing code together.

During the rest of 2021 and beginning of 2022 I kept working on the concepts and software for what became *c0l1v3*. The main challenge was to stream the audio over the web with good quality and low latency, for me it was very important to have as low latency as possible so that everyone playing could react together. The solution for that was WebRTC. I started writing my own solution called *Spatify* for another project in 2019, but in 2021 I found *Janus WebRTC Server* and its streaming plugin. Using *gstreamer* to stream from *SuperCollider* to *Janus* it is possible to have multiple web clients receiving the same audio and data stream with very low latency.

The text interface synchronisation uses *websockets* so that all the players see what each other is typing. Commands are sent to the server via *websocket* and then sent to *SuperCollider*. The *websocket* server-side is connected to *SuperCollider* using *tcposcrouter* so it works as a proxy between OSC messages of *SuperCollider* and *WebSocket* messages from the browser.

Using *tcposcrouter* also enables multiple synchronised *SuperCollider* servers running in different locations, in a more distributed performance approach, being independent of internet connection in case the internet fails. For example one person can run their own full setup consisting of *SuperCollider*, the HTML/JavaScript frontend and a local *ws2udp* to proxy *websocket* messages to OSC. The *SuperCollider* program would connect to other *SuperCollider* instances using *OSCRouter* class and being on the same *OSCrouter* group as the other players, sharing code this way, and the sound comes directly out of the local *SuperCollider* sound engine. If the connection fails at any moment, the sound is not interrupted since the sound is produced locally in this case.

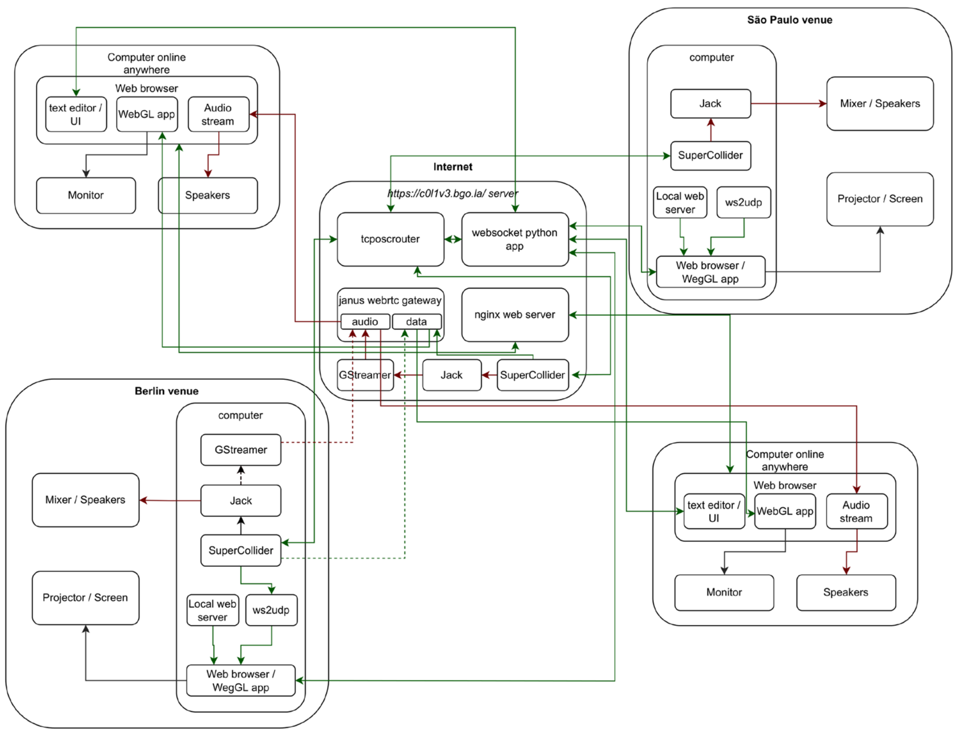
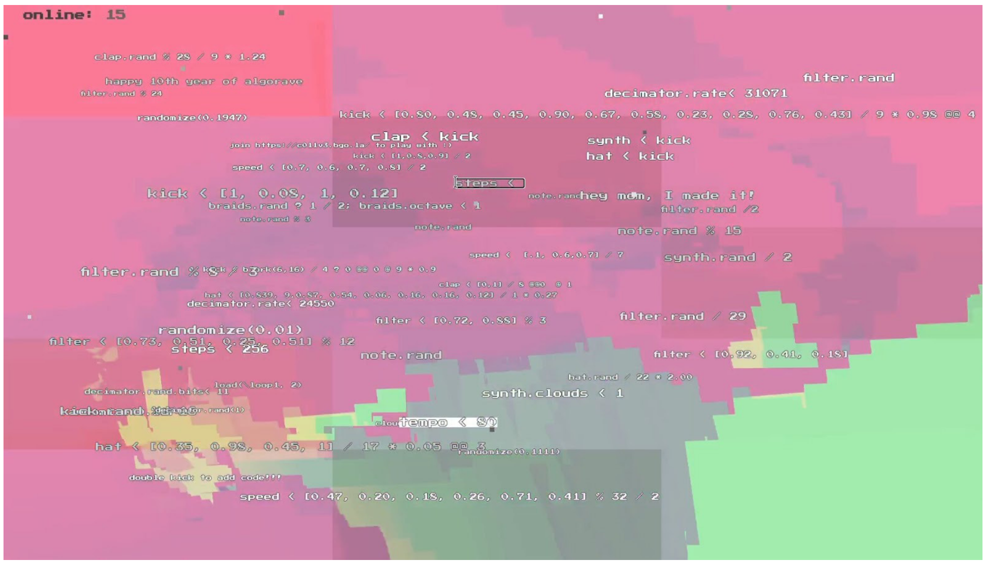


Figure 2: Technical audio and data flow diagram for *c0l1v3*.

Nonlinear Editor & Domain Specific Language

Two essential features of *c0l1v3* are its DSL, or Domain Specific Language, and its nonlinear text editor. Because of my experience with Linux systems and the Python programming language. I started designing the language inspired by REPL environments. REPL stands for Read-Eval-Print-Loop, and is a standard way of communicating with command line programs in UNIX environments. For me REPL provides a simple text-based way for controlling processes. So in *c0l1v3* every block of code is treated as a command in a REPL environment, and the user or player evaluates one command or block each time. This, and the fact that blocks can be added anywhere in the interface, without any line structure, makes it hard to create big blocks of continuous code.

In my experience working with other live coding environments my programmer self would always show up and be in the way of the performer self. I would end up writing big functions and large blocks of code that would do complex musical things, but would lose the improvised quick responsiveness that I was searching for when performing.



```
online: 15
clap.rand % 28 / 9 * 1.24
happy 10th year of algorave
filter.rand % 24
filter.rand
decimator.rate< 31071
randomize(0.1947)
kick < [0.80, 0.48, 0.45, 0.90, 0.67, 0.58, 0.23, 0.28, 0.76, 0.43] / 9 * 0.98 @@ 4
clap < kick
synth < kick
join https://c0l1v3.bgo.la/ to play with :)
hat < kick
kick < [1,0.8,0.9] / 2
speed < [0.7, 0.6, 0.7, 0.8] / 2
steps <
kick < [1, 0.08, 1, 0.12]
note.rand
hey mom, I made it!
braids.rand ? 1 / 2; braids.octave < 1
filter.rand /2
note.rand % 3
note.rand
note.rand % 15
speed < [.1, 0.6,0.7] / 7
synth.rand / 2
filter.rand % 8 / 3
kick / bjork(6,16) / 4 ? 0 @@ 0 @ 9 * 0.9
clap < [0,1] / 8 @@0 @ 1
hat < [0.839, 9,0.87, 0.54, 0.06, 0.16, 0.16, 0.12] / 1 * 0.27
decimator.rate< 24550
filter < [0.72, 0.88] % 3
filter.rand / 29
randomize(0.01)
filter < [0.73, 0.51, 0.25, 0.51] % 12
steps < 256
note.rand
filter < [0.92, 0.41, 0.18]
hat.rand / 22 * 2.00
load(\loop1, 2)
decimator.rand.bits< 11
synth.clouds < 1
kick.rand
decimator.rand(1)
tempo < 80
hat < [0.35, 0.98, 0.45, 1] / 17 * 0.05 @@ 3
randomize(0.1111)
double kick to add code!!!
speed < [0.47, 0.20, 0.18, 0.26, 0.71, 0.41] % 32 / 2
```

Figure 3: *c0l1v3* web interface.

By designing the language and environment focusing on short blocks of code and rapid switching between blocks, the *c0l1v3* environment invites players to experiment and improvise, to go in different directions. Blocks of code that are not evaluated after a certain time are deleted from the screen, also forcing players to move on to new things.

The system also makes heavy use of *JITLib*, a *SuperCollider* library for Just-in-Time, or interactive, programming. With the help of *JILib* it is possible to do controlled random changes to the sound process, and this is encouraged by *c0l1v3*'s philosophy. Be it small random moves through its dimensional space, to explore sounds found during a performance or big random changes to go in a completely different direction. Players can also store and recall the current state, as well as morph in time or step by step between two different states.