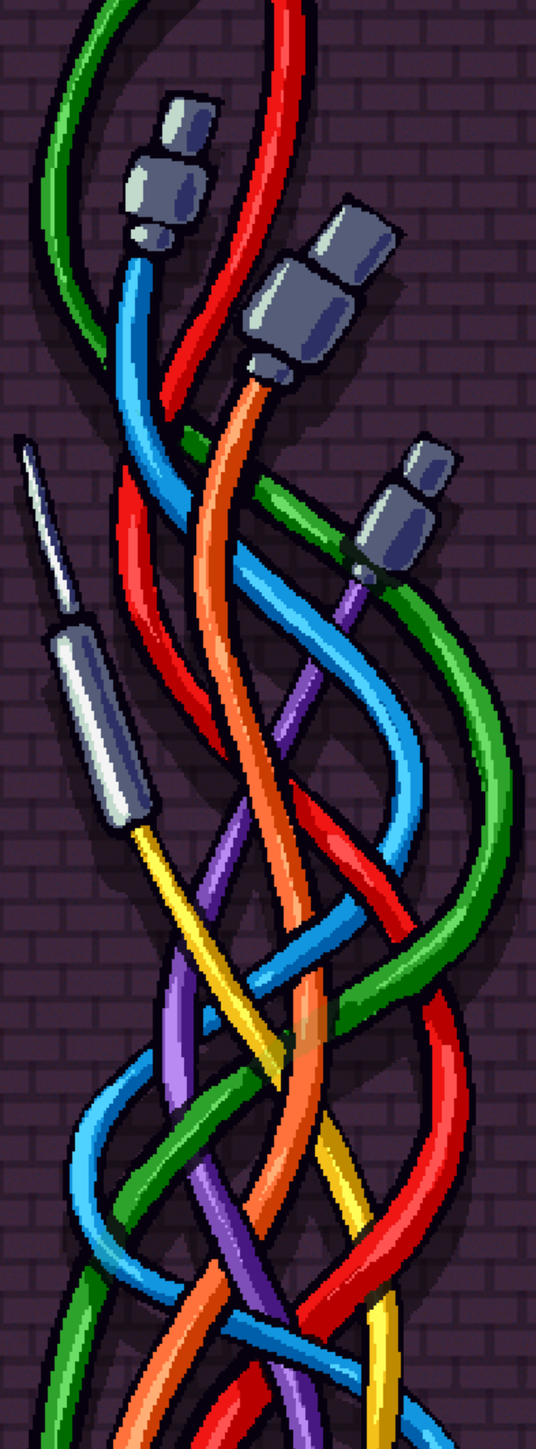


ADCx GATHER

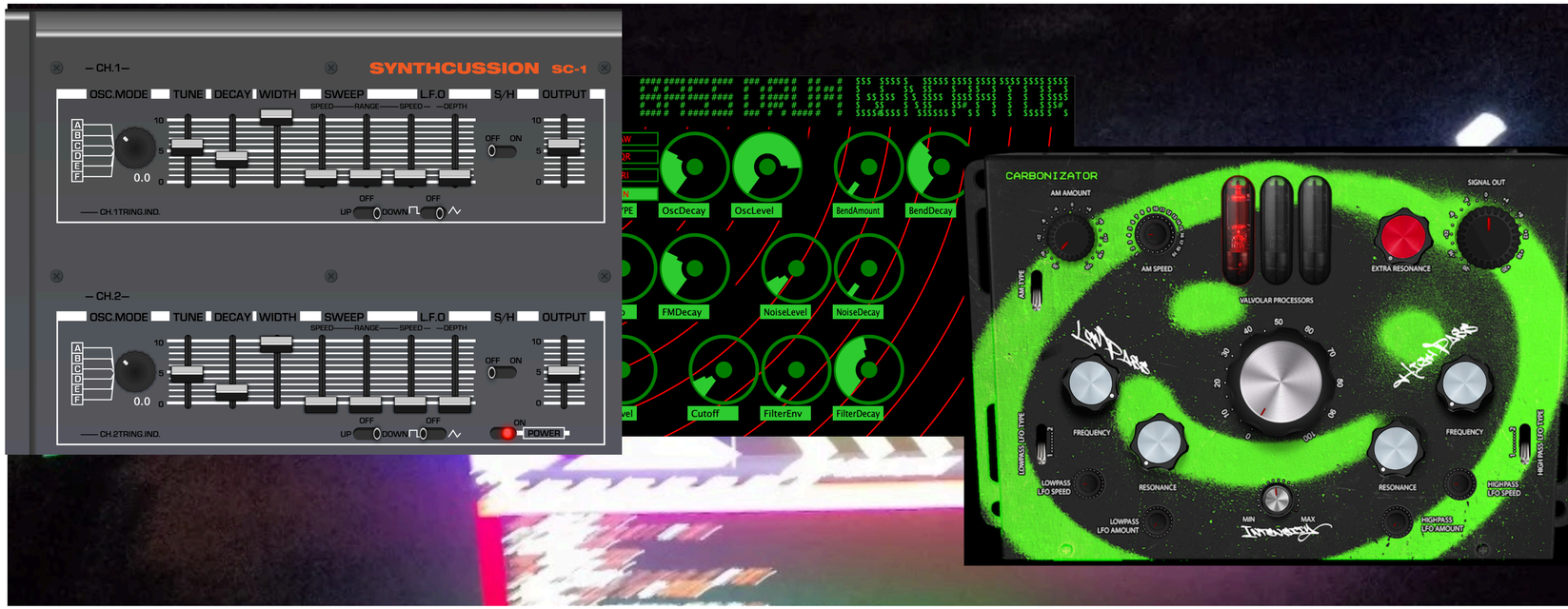
SOUND OVER BOILERPLATE

ACCESSIBLE PLUG-INS DEVELOPMENT WITH PHAUSTO AND CMAJOR

DOMENICO CIPRIANI

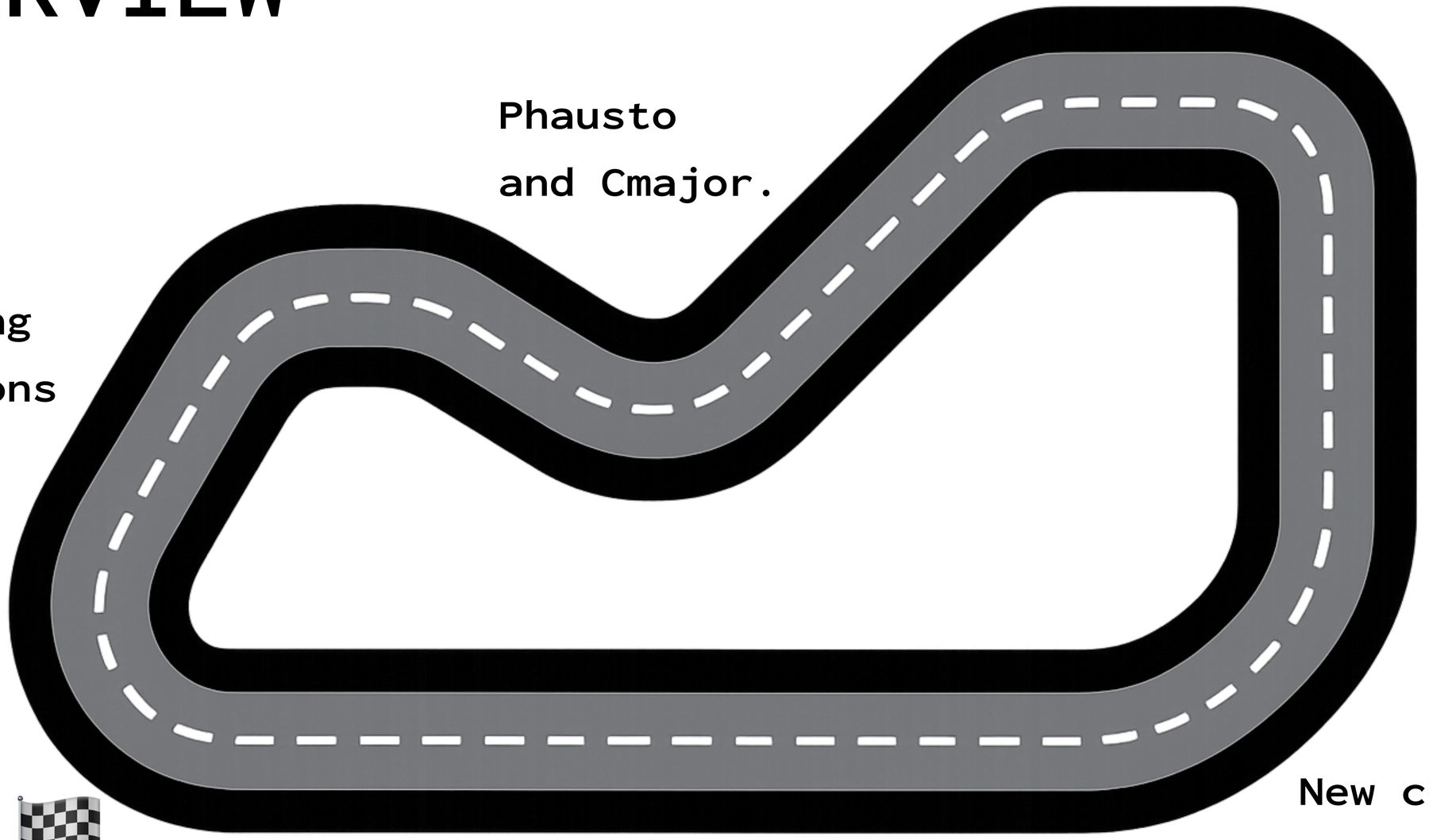
ABOUT ME

- Dj producer, live performer, since early 2000.
- Live coder and audio developer since 2020.
- Currently working for Evref/Inria.



Domenico Cipriani - 2025

OVERVIEW



Phausto
and Cmajor.

Monetisation

Existing
Solutions

New challenges

The problem.

Domenico Cipriani - 2025

THE PROBLEM

Made with FAUST

Audio Developers

Sound Artists and Musicians

- C++ is hard to learn. Hard to master. Dangerous.

- JUCE ≠ everything

☠ Memory management

☠ Data races

Inria Pharo

PHAUSTO AND CMAJOR

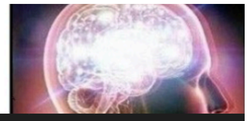
Made with FAUST

C/C++

```
#include "PluginEditor.h"

//==============================================================================
PulseOSCAudioProcessor::PulseOSCAudioProcessor()
    : AudioProcessor (BusesProperties()
#if ! JucePlugin_IsMidiEffect
#if ! JucePlugin_IsSynth
                      .withInput  ("Input",  juce::AudioChannelSet::stereo(), true)
#endif
                      .withOutput ("Output", juce::AudioChannelSet::stereo(), true)
#endif
                      ),
      apvts(*this, nullptr, "PARAMETERS", {
          std::make_unique<juce::AudioParameterFloat>("attack", "Attack", juce::NormalisableRange<float>(0.001f, 5.0f, 0.001f), 0.01f),
          std::make_unique<juce::AudioParameterFloat>("decay", "Decay", juce::NormalisableRange<float>(0.001f, 5.0f, 0.001f), 0.1f),
          std::make_unique<juce::AudioParameterFloat>("sustain", "Sustain", juce::NormalisableRange<float>(0.0f, 1.0f, 0.001f), 0.8f),
          std::make_unique<juce::AudioParameterFloat>("release", "Release", juce::NormalisableRange<float>(0.01f, 10.0f, 0.001f), 0.5f),

          std::make_unique<juce::AudioParameterFloat>("pulseWidth", "Pulse Width", juce::NormalisableRange<float>(0.01f, 0.99f, 0.001f), 0.5f),
          std::make_unique<juce::AudioParameterFloat>("freq", "Frequency", juce::NormalisableRange<float>(20.0f, 20000.0f, 0.01f, 0.3f), 440.0f),
          std::make_unique<juce::AudioParameterFloat>("gain", "Gain", juce::NormalisableRange<float>(0.0f, 1.0f, 0.001f), 0.8f),
      })
{
    // initial smoothing values
    pulseWidthSmoothed.reset(currentSampleRate, 0.02);
    gainSmoothed.reset(currentSampleRate, 0.02);
    freqSmoothed.reset(currentSampleRate, 0.02);
}

PulseOSCAudioProcessor::~PulseOSCAudioProcessor()
{
}

//==============================================================================
const juce::String PulseOSCAudioProcessor::getName() const { return JucePlugin_Name; }
bool PulseOSCAudioProcessor::acceptsMidi() const { return true; }
bool PulseOSCAudioProcessor::producesMidi() const { return false; }
bool PulseOSCAudioProcessor::isMidiEffect() const { return false; }
double PulseOSCAudioProcessor::getTailLengthSeconds() const { return 0.0; }

int PulseOSCAudioProcessor::getNumPrograms() { return 1; }
int PulseOSCAudioProcessor::getCurrentProgram() { return 0; }
void PulseOSCAudioProcessor::setCurrentProgram (int) {}
const juce::String PulseOSCAudioProcessor::getProgramName (int) { return {}; }
void PulseOSCAudioProcessor::changeProgramName (int, const juce::String&) {}

//==============================================================================
void PulseOSCAudioProcessor::prepareToPlay (double sampleRate, int)
{
    currentSampleRate = sampleRate;
    adsr.setSampleRate(sampleRate);

    phase = 0.0;
    phaseIncrement = 0.0;

    float pw = *apvts.getRawParameterValue("pulseWidth");
    float g = *apvts.getRawParameterValue("gain");
    float f = *apvts.getRawParameterValue("freq");

    pulseWidthSmoothed.reset(sampleRate, 0.02);
    gainSmoothed.reset(sampleRate, 0.02);
    freqSmoothed.reset(sampleRate, 0.02);

    pulseWidthSmoothed.setTargetValue(pw);
    gainSmoothed.setTargetValue(g);
    freqSmoothed.setTargetValue(f);
}

void PulseOSCAudioProcessor::releaseResources()
{
}

#ifndef JucePlugin_PreferredChannelConfigurations
bool PulseOSCAudioProcessor::isBusesLayoutSupported (const BusesLayout& layouts) const
{
  #if JucePlugin_IsSynth
    juce::ignoreUnused (layouts);
    return true;
  #else
    // only allow stereo layout
    if (layouts.getMainOutputChannelSet() != juce::AudioChannelSet::stereo())
        return false;
   #if ! JucePlugin_IsSynth
    if (layouts.getMainOutputChannelSet() != layouts.getMainInputChannelSet())
        return false;
   #endif
    return true;
  #endif
}
#endif

void PulseOSCAudioProcessor::handleMidiCC(int controllerNumber, int controllerValue)
{
    // Map CC -> parameter (you can change these CC assignments)
    // CC 20 -> attack, 21 -> decay, 22 -> sustain, 23 -> release
    // CC 24 -> pulseWidth, 25 -> freq, 26 -> gain
    float normalized = controllerValue / 127.0f;

    switch (controllerNumber)
    {
        case 20: // attack
            apvts.getParameter("attack")->setValueNotifyingHost(normalized);
            break;
        case 21: // decay
            apvts.getParameter("decay")->setValueNotifyingHost(normalized);
            break;
        case 22: // sustain
            apvts.getParameter("sustain")->setValueNotifyingHost(normalized);
            break;
        case 23: // release
            apvts.getParameter("release")->setValueNotifyingHost(normalized);
            break;
        case 24: // pulse width -> normalized maps to 0.01..0.99
        {
            float pw = juce::jmap(normalized, 0.0f, 1.0f, 0.01f, 0.99f);
            apvts.getParameter("pulseWidth")->setValueNotifyingHost(apvts.getParameter("pulseWidth")->convertTo0to1(pw));
            break;
        }
        case 25: // freq (map to log range)
        {
            float minF = 20.0f, maxF = 20000.0f;
```

485 lines / generic UI

CMAJOR



```
graph PulseSynth [[main]]
{
    input event std::midi::Message midiIn;
    output stream float out;

    let voiceCount = 8;

    node
    {
        voices = Voice[voiceCount];
        voiceAllocator = std::voices::VoiceAllocator (voiceCount);
    }

    connection
    {
        midiIn -> std::midi::MPEConverter -> voiceAllocator;
        voiceAllocator.voiceEventOut -> voices.eventIn;
        voices -> out;
    }
}

graph Voice
{
    input event (std::notes::NoteOn, std::notes::NoteOff) eventIn;
    output stream float out;

    node
    {
        noteToFrequency = NoteToFrequency;
        env = std::envelopes::ADSR;
        osc = PulseOscillator;
    }

    connection
    {
        eventIn -> noteToFrequency -> osc.frequencyIn;
        eventIn -> env.eventIn;
        (env.gainOut * osc.out) -> out;
    }
}

// simple MIDI note -> Hz converter
processor NoteToFrequency
{
    input event std::notes::NoteOn eventIn;
    output event float32 frequencyOut;
```

50 lines / generic UI

FAUST

```
import("stdfaust.lib");

// parameters
attack  = hslider("attack [unit:s] [style:knob]",0.01,0.001,5,0.001);
decay   = hslider("decay [unit:s] [style:knob]",0.1,0.001,5,0.001);
sustain = hslider("sustain [style:knob]",0.8,0,1,0.001);
release = hslider("release [unit:s] [style:knob]",0.5,0.01,10,0.001);

pulseWidth = hslider("pulseWidth [style:knob]",0.5,0.01,0.99,0.001);
gain       = hslider("gain [style:knob]",0.8,0,1,0.001);

// frequency from MIDI (use midi keyon/off)
freq = nentry("freq",440,20,20000,1) : si.smoo; // fallback if no MIDI
freqMIDI = pm.freq;  // polyphonic MIDI frequency
gateMIDI = pm.gate;  // polyphonic MIDI gate

// oscillator
osc = os.pulse(freqMIDI : si.smoo,pulseWidth);

// ADSR envelope
env = en.adsr(attack,decay,sustain,release,gateMIDI);

// apply gain and envelope
process = osc * env * gain <: _,_;
```

24 lines / generic UI

PHAUSTO

```
synth := (PulseOsc new freq: #freq ; uLevel: #gain ) => (ADSREnv new trigger: #gate ).
dsp := synth  stereo asDsp.
dsp init.
myPatch := CmajorMIDIPatch new name: 'PulseSynth'; dsp: dsp.
myPatch export.
```

5 lines / generic UI

Domenico Cipriani - 2025

Inria

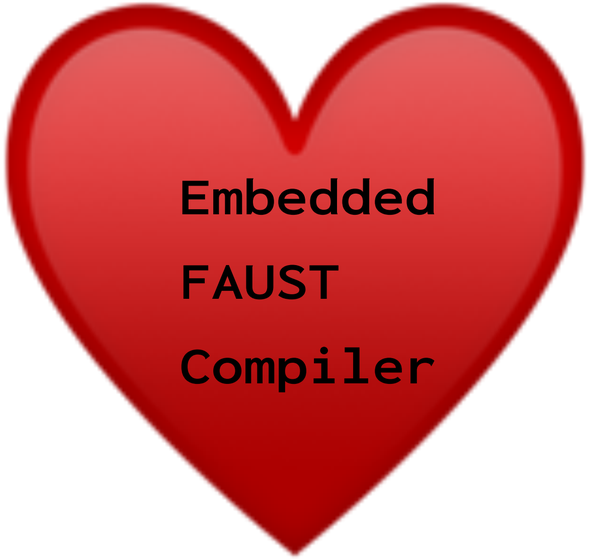
Pharo

THE SOLUTION(S)

Max4Live	Not free, only runs in Ableton Live
PlugData	Sparse libraries, questionable performance
FAUST(VST)/w HISE	Functional style can be hard to learn
Cmajor	Easier to learn but still tricky for beginners. JS required for UIs

PHAUSTO AND CMAJOR

- Phausto is a library and API for Pharo Smalltalk for easy DSP programming.



Huge amount of Unit Generators from the Standard Libraries

Exporters to many different languages including Cmajor!

- Cmajor patches can easily run on the free loader plug-in made

Inria

Cmajor

Made with FAUST

- C-family language designed for writing DSPs.
- It aims to attract beginners but also to match C/C++ performances.
- The Cmajor VST/AU plugin can be used to load patches inside any regular DAW.

Inria Pharo

PHARO AND PHAUSTO

- Pharo is a pure OOP language
- Free / open-source / multi-platform

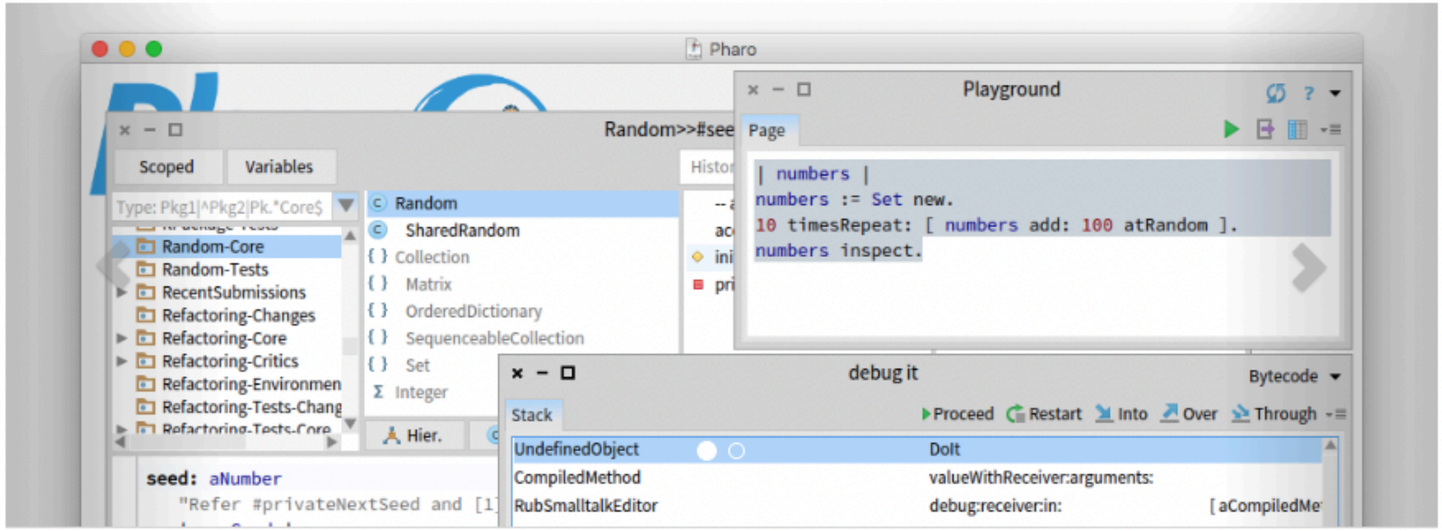
+

Simple language syntax

Powerful IDE

Integrated Git support

Advanced run-time reflection



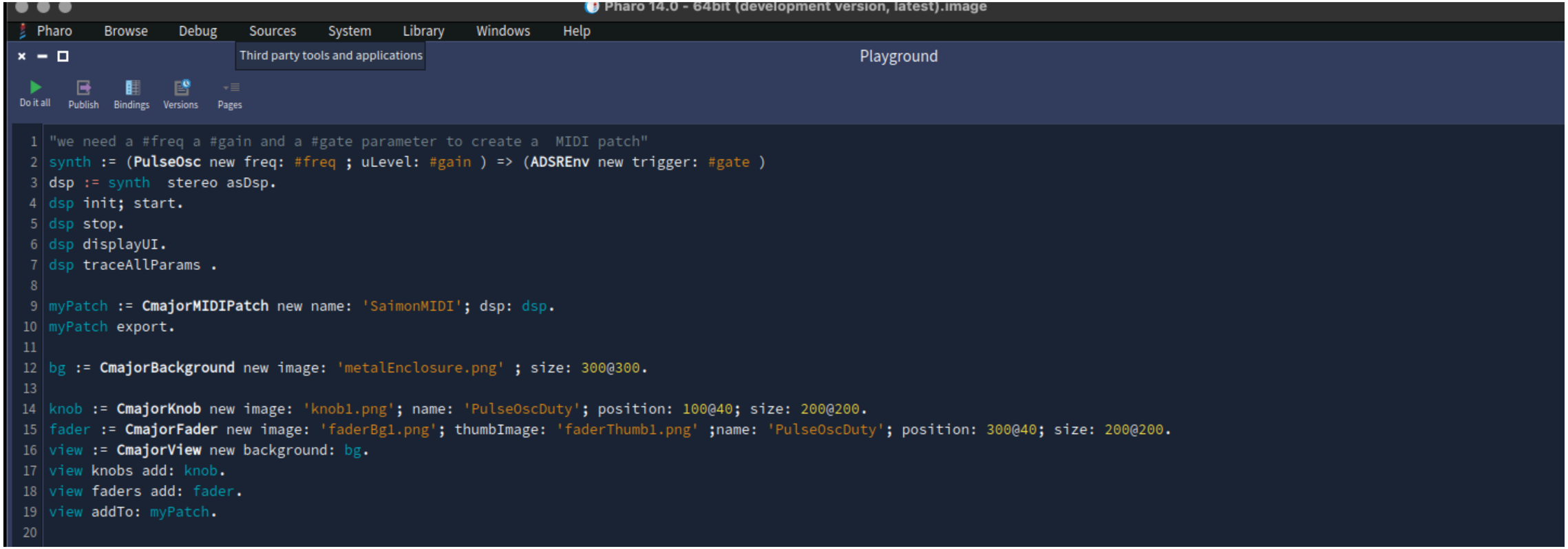
- 7 weeks of online MOOC.
- 7 days of MasterLu

Domenico Cipriani - 2025

Inria

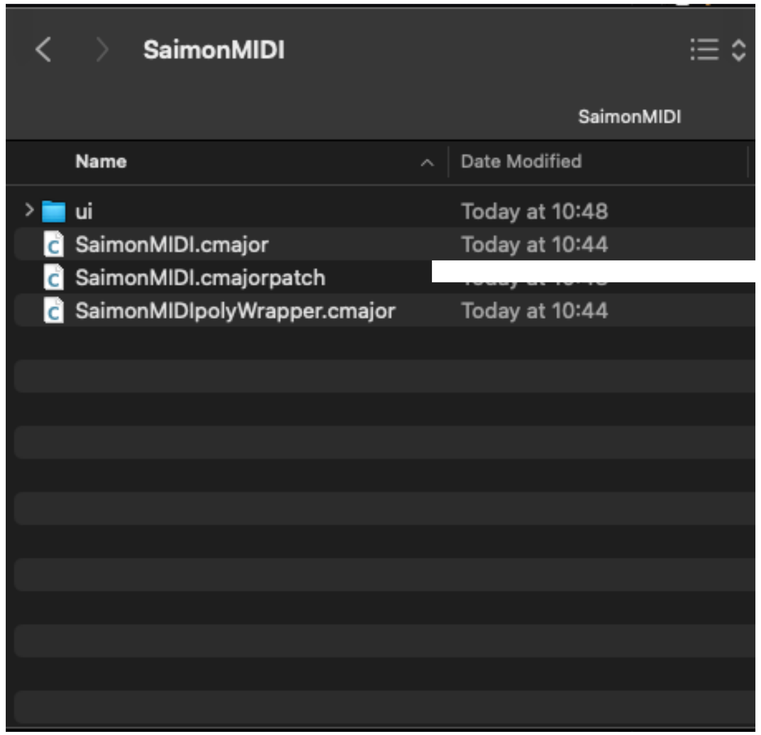
Pharo

EXPORT TO CMAJOR IN A LAP



Domenico Cipriani - 2025

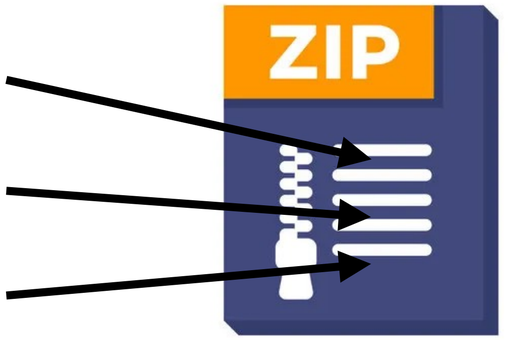
USE A CMAJOR PATCH IN A DAW



=

Domenico Cipriani - 2025

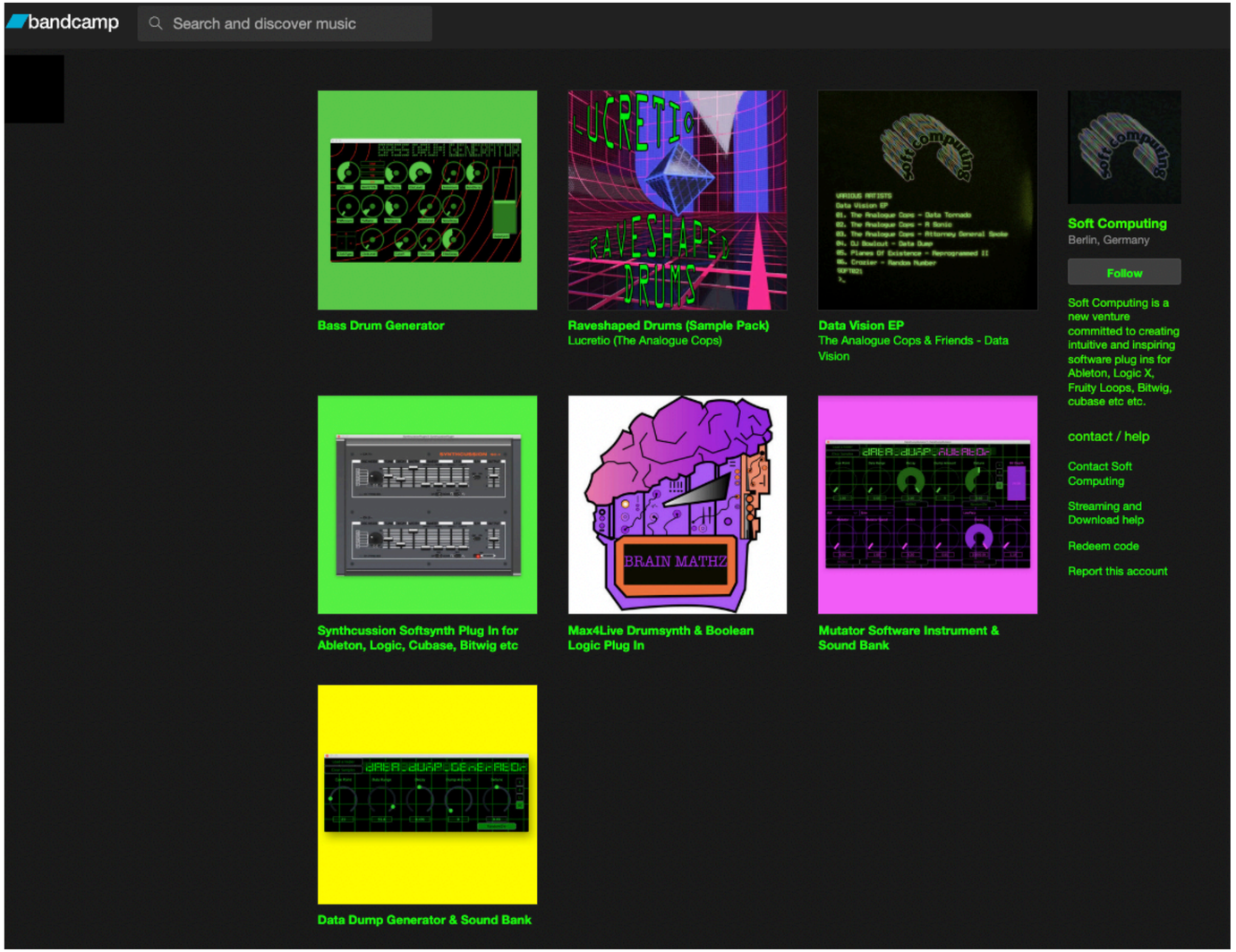
THE MONETISATION



Download Cmajor plug-ins - drag & drop your patches
(https://github.com/cmajor-lang/cmajor/releases)

Parts of the FAUST libraries are copyrighted ⚠

THE MONETISATION



Bandcamp lets you attach bonus items with our music.

Bundle plugins/assts into a ZIP file and upload them as a bonus item.

Domenico Cipriani - 2025

THE CHALLENGES

Isn’t AI coding plug-ins for us ?

More complex UIs / simple API.

Native DAW support for Cmajor patches.

Grazie!
See you in GatherTown

https://github.com/lucretiomsp/phausto
https://pharo.org/
https://cmajor.dev/
https://faust.grame.fr/

Domenico Cipriani - 2025