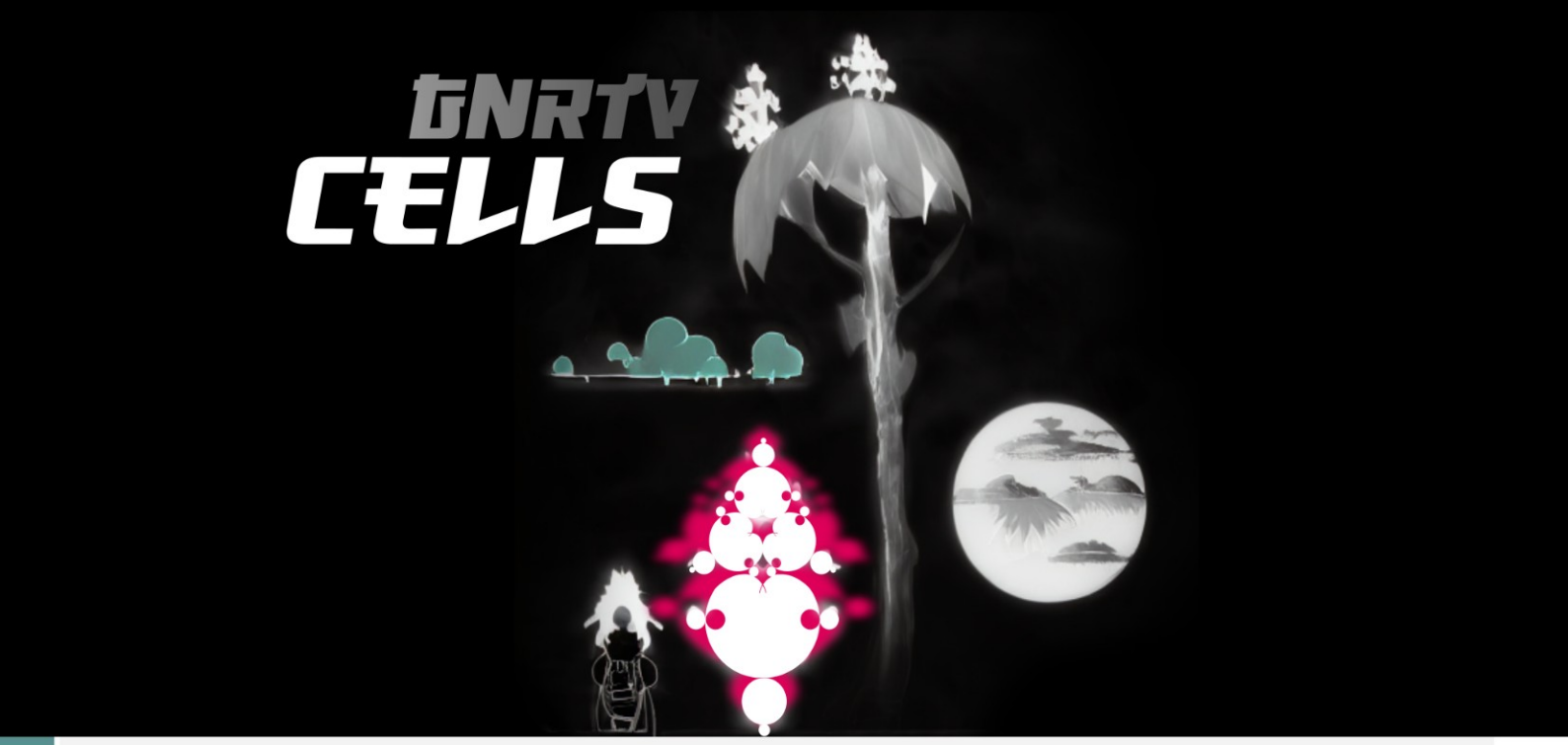
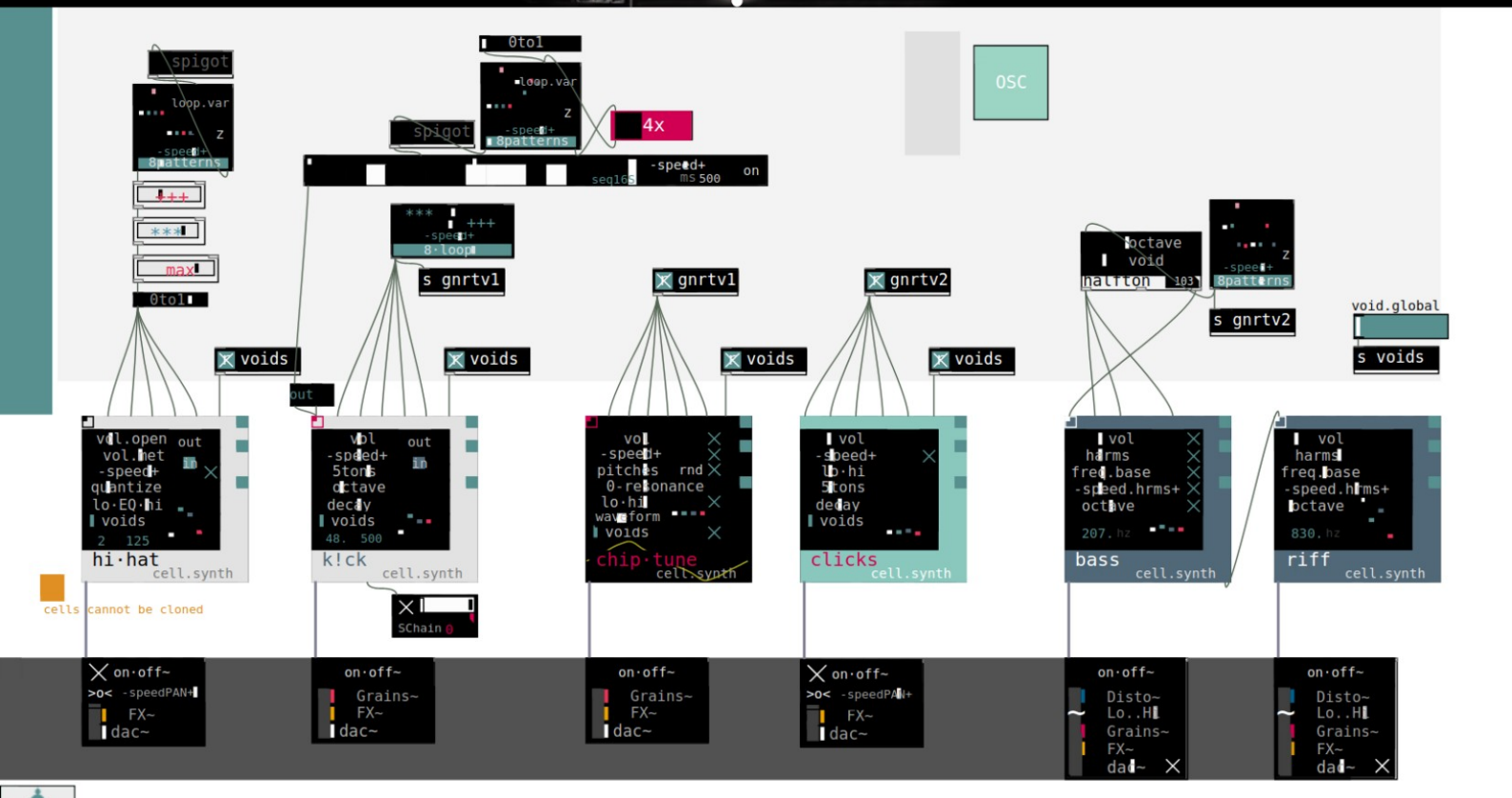
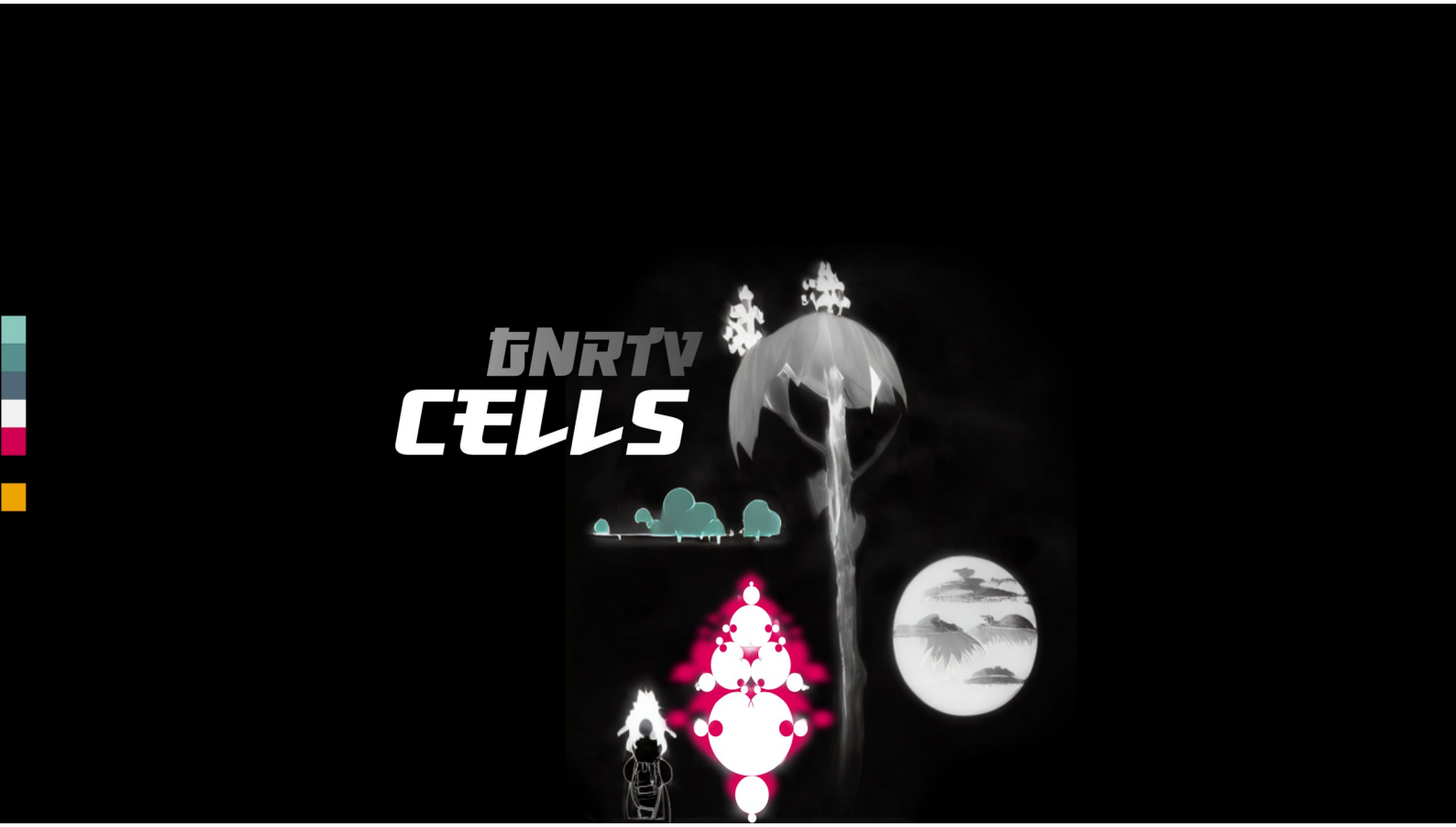

GNRTV
CELLS


spigot
loop.var
z
-speed+
8patterns
0to1
max
s gnrtv1
s gnrtv2
gnrtv1
gnrtv2
voids
4x
OSC
seq16s
ms 500
on
octave
void
halfton
s voids
void.global
vol.open
vol.het
quantize
lo·EQ·hi
hi·hat
cell.synth
k!ck
chip·tune
clicks
bass
riff
cells cannot be cloned
SChain
on·off~
FX~
dac~
Grains~
Disto~
Lo..HI
butxaca
pocket
clØck
bpm 120 ms 125
core
~mixer
Low·Pass
delay.time
vol.Reverb
kut.Schain
vol.Delay
reinjection
468
vol.TOT
click or Space
rec~
start
stop
mic
GNRTV
cells
reset

2 // Toolkit GNRTV.CELLS

GNRTV.CELLS (Generative Cells) is a 'Toolkit' inspired by the ***modular synthesizers*** (which have been so successful in the last decade) in the framework of the algorithmic music, to be able to build **generative algorithmic instruments** (i.e. which can potentially be partially or fully automated inspired by physical and biological models).
The Toolkit has applications in music and live performance, NewMedia Art projects and interactive and/or immersive installations.
GNRTV.CELLS is programmed with the ***pd-l2ork* / purr-data**** language, in order to install the toolkit you must follow the instructions at the end of this chapter in section 2.2 INSTALL
https://github.com/xamanza/GNRTV.CELLS

pd-l2ork* / purr-data*
are versions of the pd (pure-data) language that implement more current UI UX features with js technology.



Note: This is the first version of GNRTV.CELLS, a process that has been overwhelmed by time
It will continue to be able to build a compact tool for artistic, creative and educational applications.
https://github.com/xamanza/GNRTV.CELLS

GNRTV.CELLS Toolkit is aimed at building:

> Algorithmic Sound design and composition
// sound production to be able to use in other resources and media projects
// live performance

> Automata // Sonic Generative Engines
// Generative sound algorithm programming to make autonomous / automatic systems suitable for new media art installations and/or for
audio engines in hybrid applications (VR / AR / Games etc.)

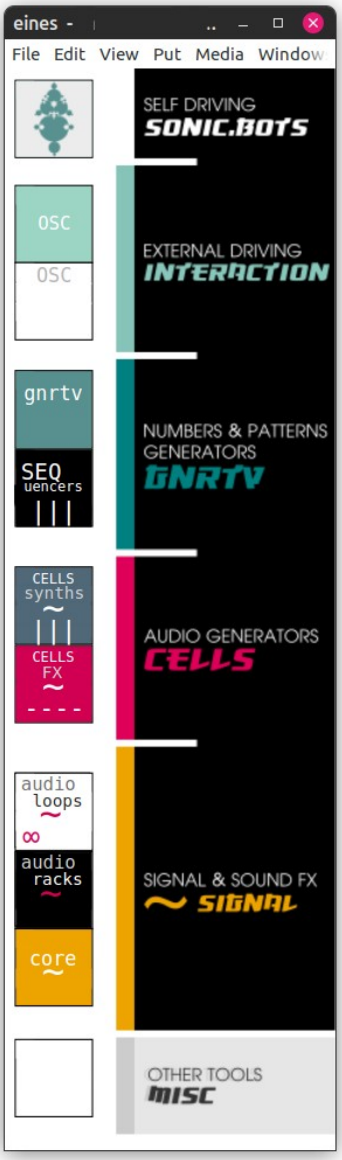
> Sonic Interaction
Communication with other software and/or hardware via OSC

> Live Coding Mode for live sessions 'from scratch'
Communication with other software and/or hardware via OSC

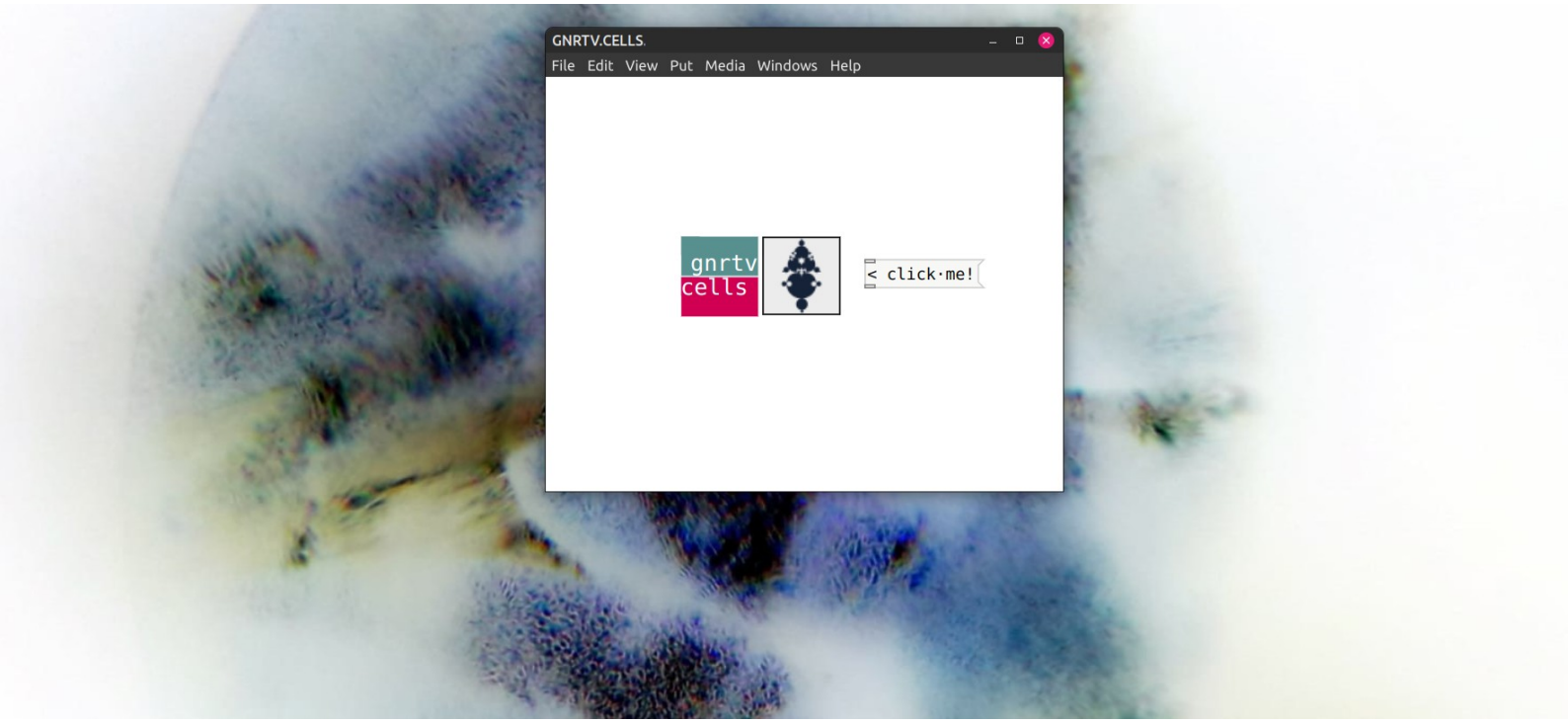
It is a MODULAR tool where potentially 'everything can be connected to everything' as it happens with the Modular Synths, from which it is an inspiration.

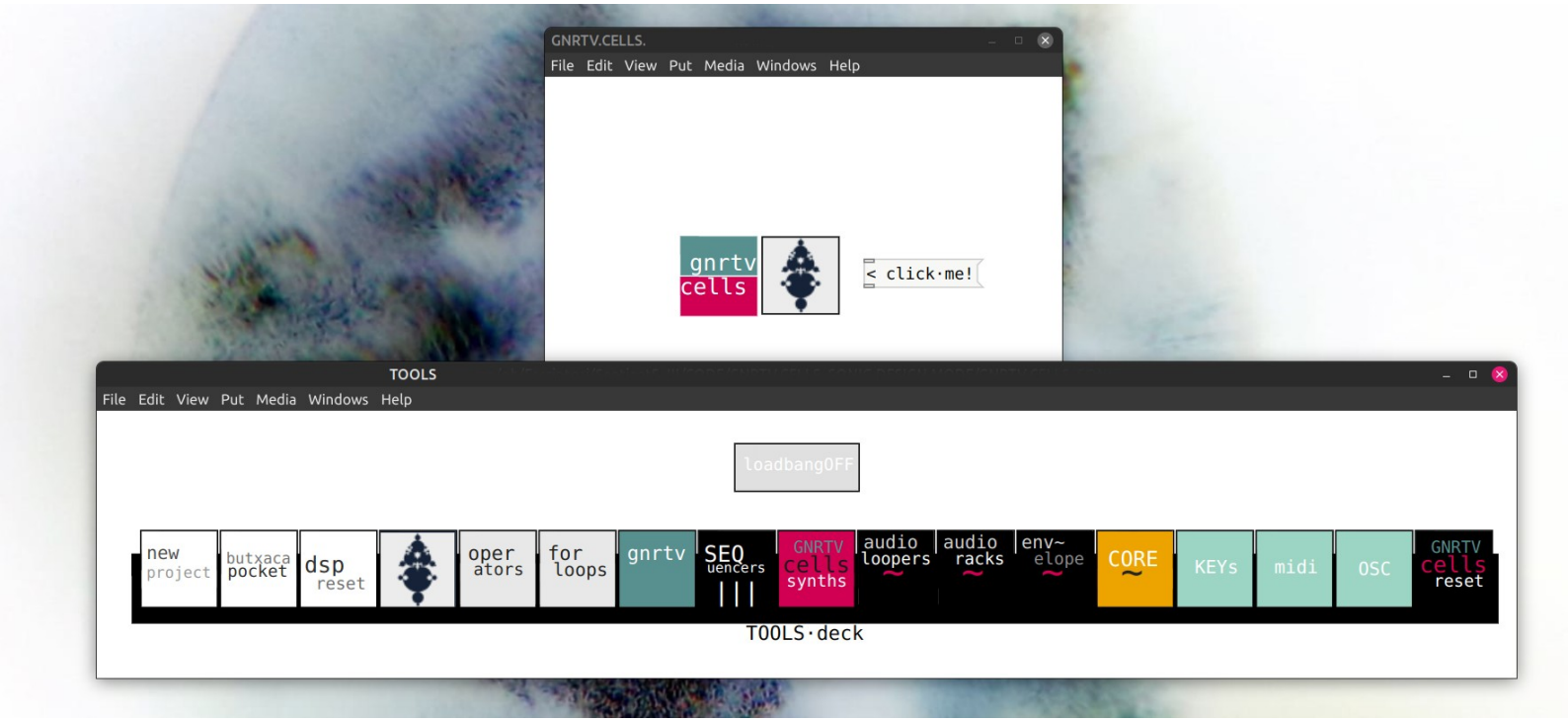
The structure is as follows from top to bottom (with the layer close to Human Interaction at the top and the digital sound signal (DSP) calculation and computation at the bottom):

***INTERACTION** // OSC, midi, keys*
***GENERATIVE** // numbers, patterns and data generators*
***CELLS** // Audio generators > SYNTHS & AUDIO LOOPERS*
***SIGNAL** // Audio Processing & FX*



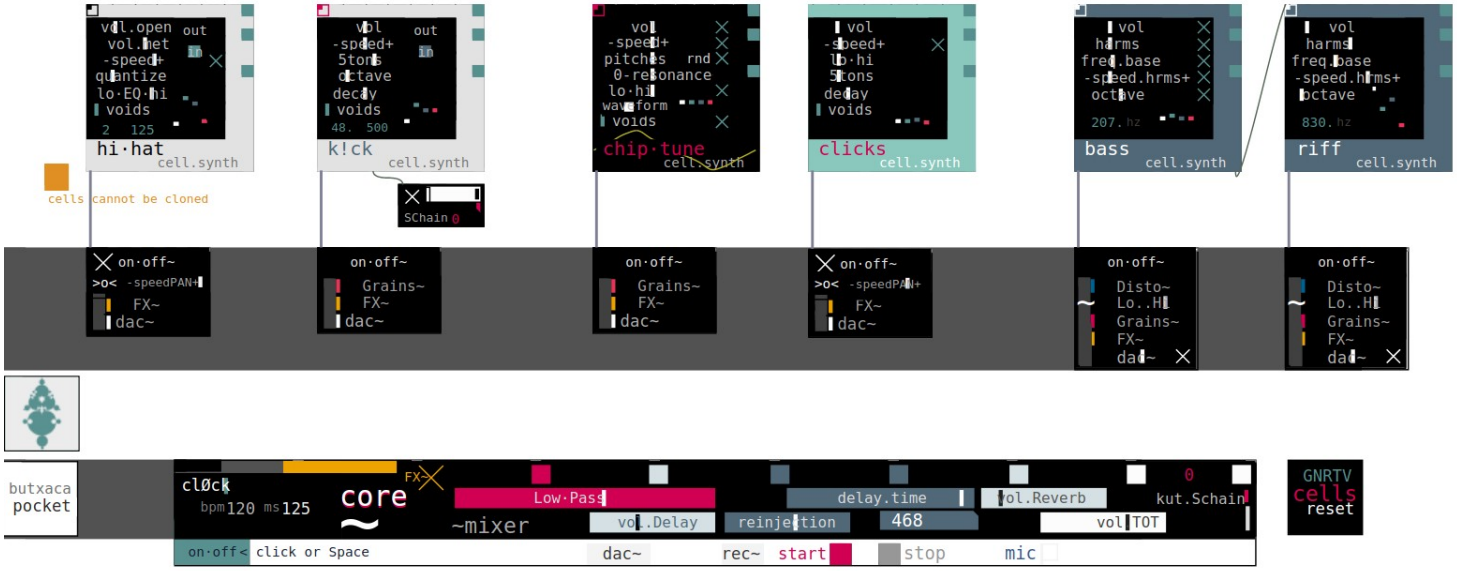
GNRTV.CELLS is a modular tool.
We will have a simple interface with a TOOL·deck where the different elements that we can use appear.



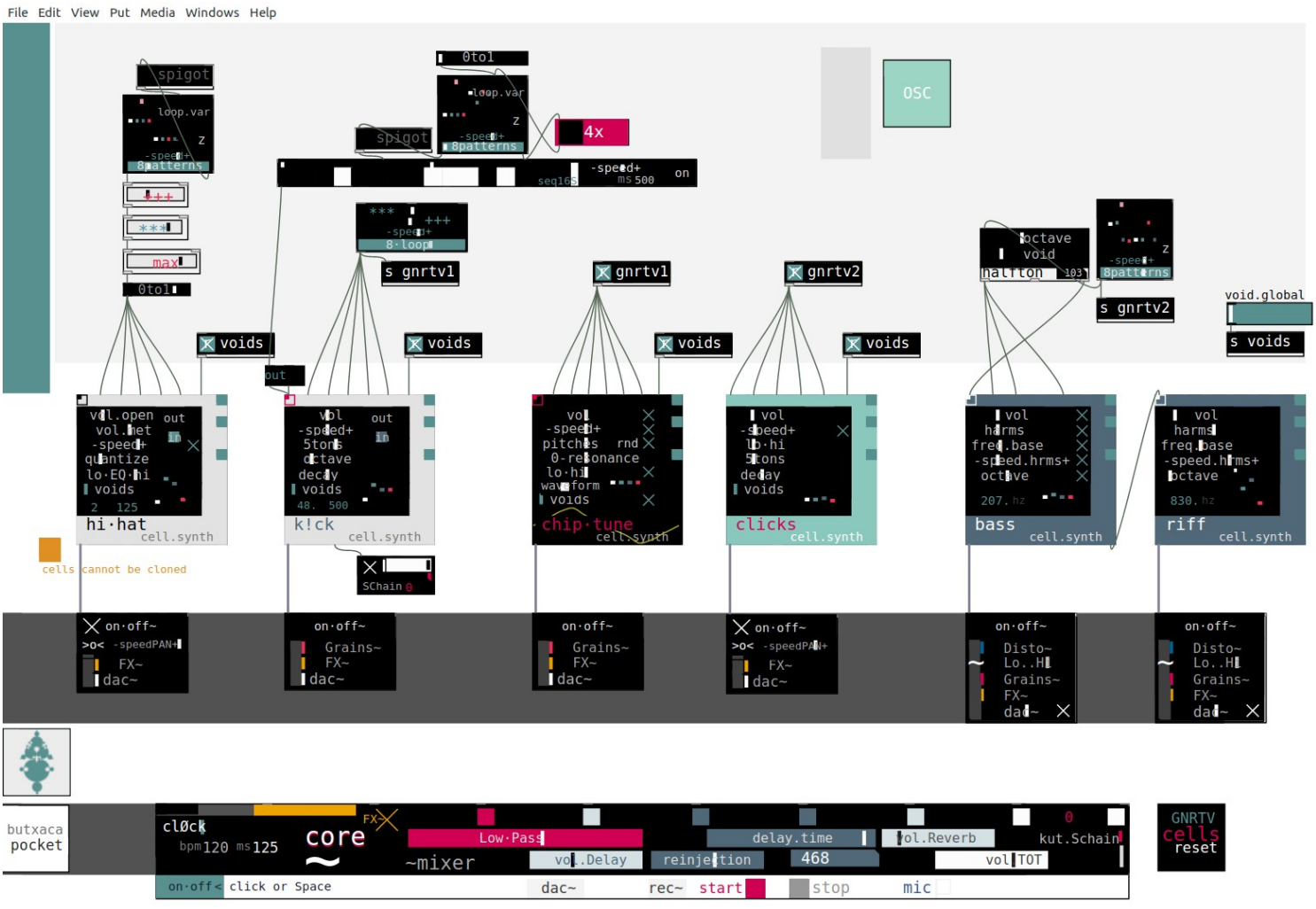


Any instrument we want to build should consist of this structure (from top to bottom) >

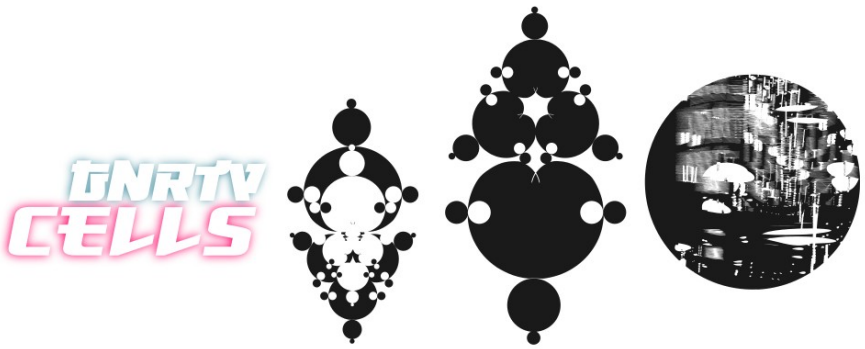
> The above can be applied to Sound Generators [CELLS] including: Synthesizers, and Sound generators (Loopers and Samplers included)
> All the previous flow is sent to the 'Core' which is in charge of the SFX (Sound Effects) as well as the Master Clock of the whole project.



Additionally, we can equip the previous example with automated generative methods
> **Generative Modules [GNRTV]** (generate different numerical combinations including random methods, circular or loop methods, and sequential methods)



tNRTV.CELLS
SECCIONS

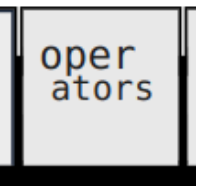


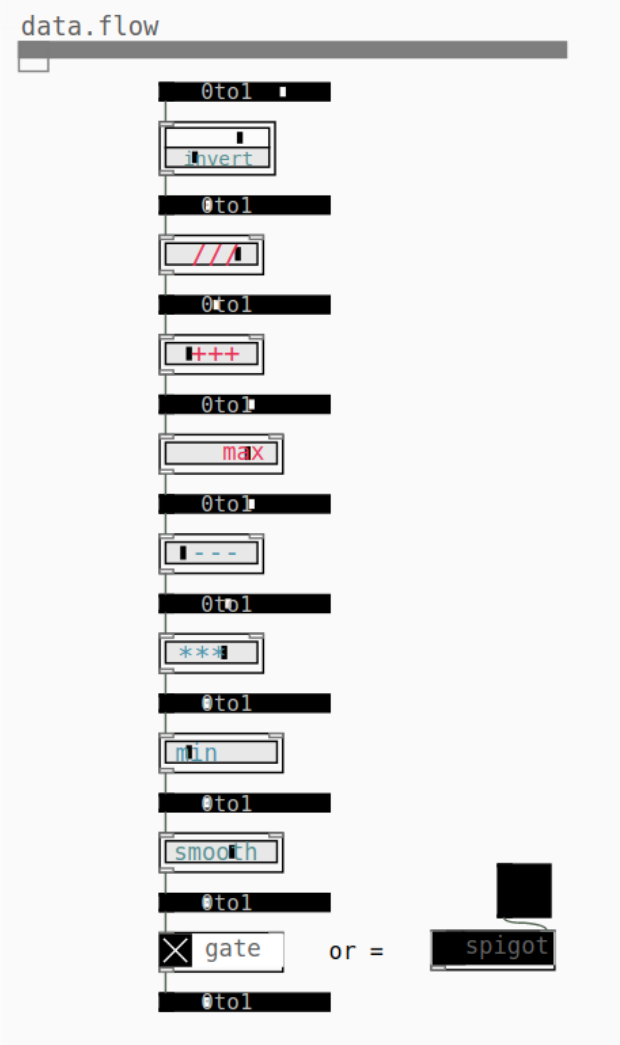
In order for it to be a versatile tool where 'everything can potentially be connected to everything', a design has been chosen where the parameters are assigned values from 0 to 1.
This means that we can translate and sonify data and data generators (GNRTV) in a simple way with any of the sound parameters available in the Audio Generators, given that these parameters can be of very different ranges (frequencies, octaves, time of decay, volumes, etc...).

All these parameters can be modulated and arranged or 'tweaked' for each case with a set of arithmetic operators that will allow us to adjust the desired ranges for each instrument or block we want to control.

In this regard, below are the operators that allow an incoming value from 0 to 1 to be:
inverted ***[invert]***
Division ***[///]***
Addition or sum ***[+++]***
Values does not exceed a maximum value ***[max]***
Subtraction ***[---]***
multiplication ***[***]***
Values does not fall below a minimum value ***[min]***
Value oscillations are smoother and without sudden shocks ***[smooth]***
We can open or close a flow as if it was a gate ***[gate] [spigot]***

operators

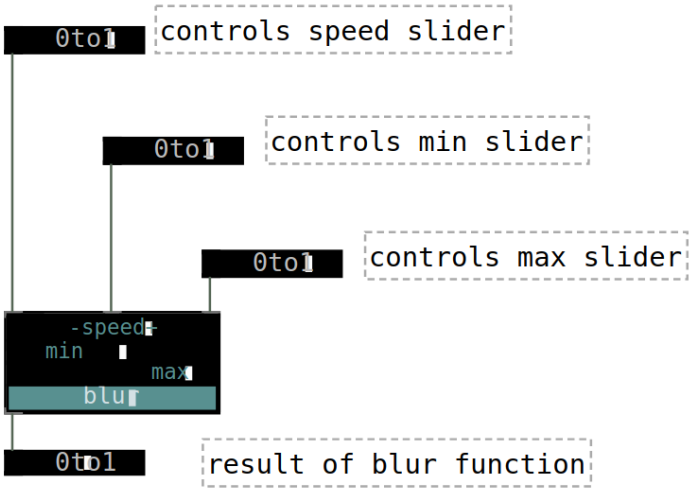




All this flow that we can weave between the values 0 and 1, we can interconnect them between the different blocks or modules.
The Blocks or modules have inputs at the top and outputs at the bottom. The criterion is from left to right and from top to bottom, although due to the large number of Blocks, you will have to see what are the entrances and exits of each Block by 'entering inside' > Right Button > Open

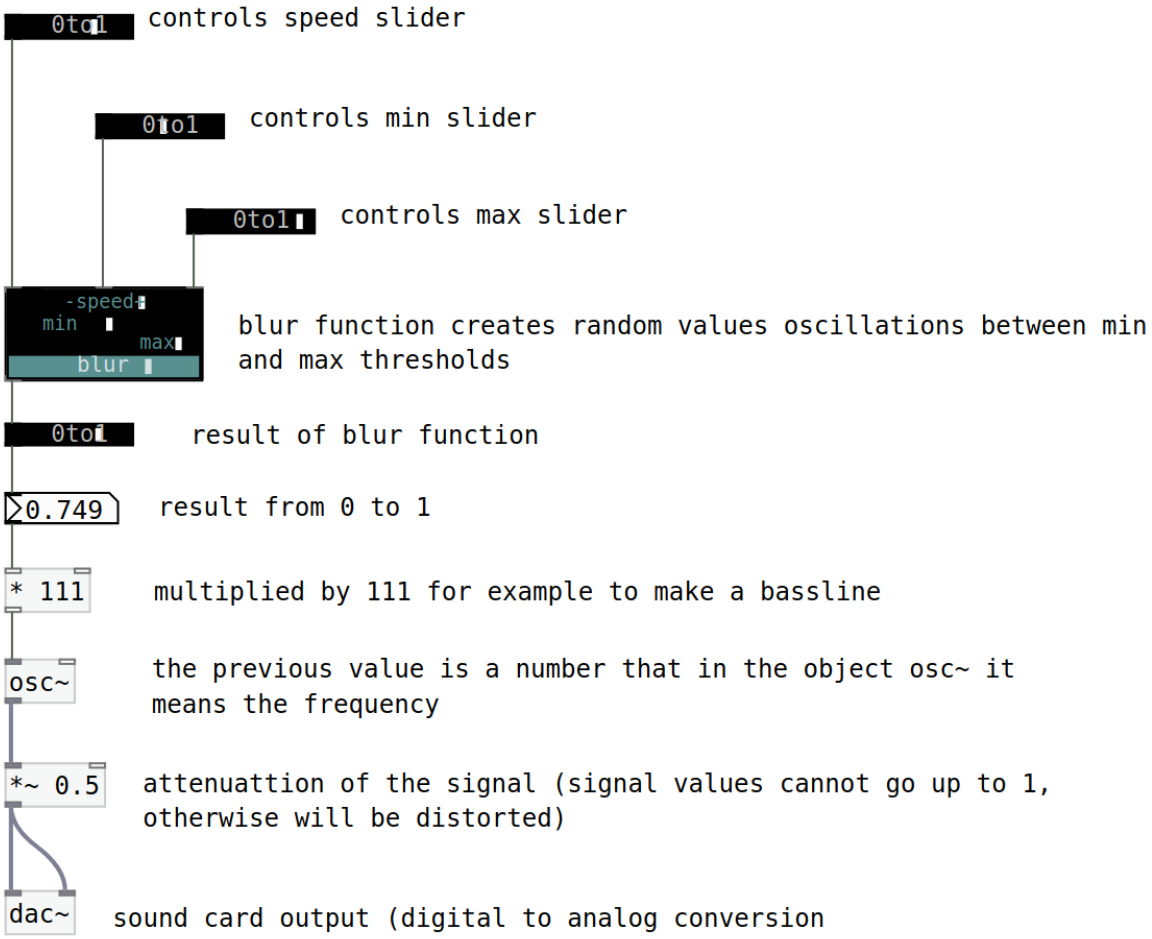
So for example, following the previous comments:
In the 'blur' Block, we will have three entries at the top that correspond from left to right with the relationship from top to bottom, being the first entry that controls the speed parameter, the second the min parameter and the third the parameter max.
The result of the function 'blur' is extracted by the output at the bottom left.

*Blur is a **generator of random numbers** that oscillates between minimum and maximum values assigned to the min and max sliders. Additionally, these number fluctuations can be 'slower or faster' with the **-speed+** slider.*



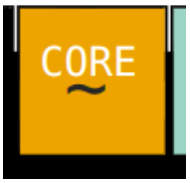
*All the Blocks have this methodology, although there are some that, given the complexity, have more inputs and outputs that will need to be consulted **inside** in case of confusion : **'entering inside' > Right Button > Open** to check which they are inputs (inlets) and outputs (outlets).*

The previous example of Blur is a data generator that by itself does not generate more than this. We can quickly sonify them with other blocks, including other previous pd developments. For example, with the previous fragment we sonify it directly to an oscillator:

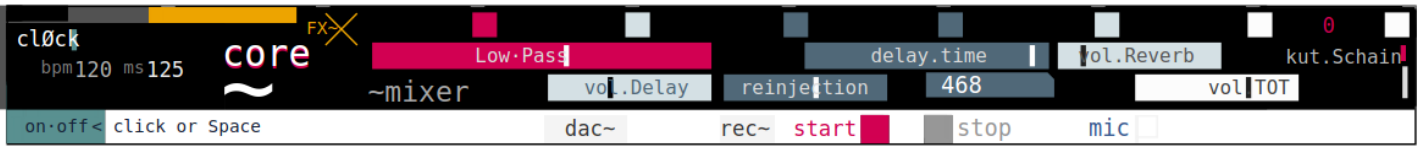


Section [CORE]

Before starting, you should bear in mind that GNRTV.CELLS is a usability layer with an algorithmic environment that can often generate a descent into very complex techniques that require a level of DSP and code, only suitable for specialists in the field.
For this reason the Toolkit aims to make algorithmic music production more intuitive and visual.



To run any GNRTV:BLOCKS project we want to build, we'll need the CORE.



*CORE is a block that will manage audio signal receptions 'wirelessly' to be able to process them with a compact **SoundFX** effects rack, as well as a master **clock** that will organize all the instruments, blocks and cells that contain metronomes and time sequences (which in fact are the most).*

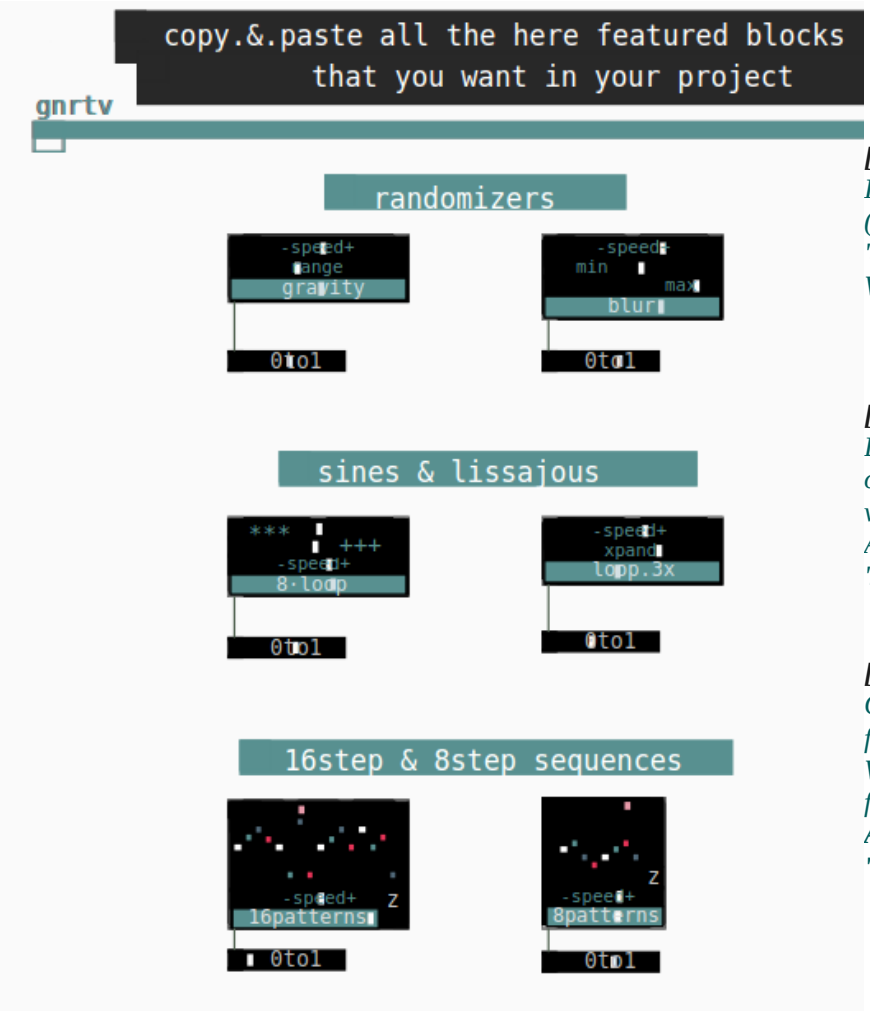
*The block also contains a very useful function which is the recorder or recording, located at the bottom of the Block. Therefore, everything that sounds at a given moment can be recorded, and it will be automatically recorded in the **RECS** folder with the nomenclature **record.GNRTV.CELLS___X** where **X** is the recording accumulative number every time we start the session.*

This directory must be understood as a temporary directory, so if we want to edit a certain recording later, it is recommended to move the desired file, to another location so that it cannot be accidentally overwritten.

Section [GNRTV]

gnrtv *[generative]*

Are a set of Blocks that gives the project its name and that are continuous generators of data, whether these are sonified or not.



[gravity]
*It creates **oscillating random values** that pivot (closer to the bottom and slightly to the top) on an 'attractor' that is indicated in the **range** parameter. We can assign the speed with the **-speed+** slider.*

[blur]
*Blur is a **generator of random numbers** that oscillate between a minimum and a maximum value, assigned with **min** and **max** sliders. Additionally, these number fluctuations can be 'slower or faster' with the **-speed+** slider*

[8.loop]
*Creates **balanced numerical loops** (back and forth /systole-diastole).*
*We can control its range with the ******* slider, and from where the loop starts with the +++ slider*
Additionally, these number fluctuations can be 'slower or faster' with the -speed+ slider

[loop.3x]
*It **creates numerical loops 'with an additional butterfly-shaped dimension'** (similar to Lorenz attractors).*
Loops in 3X (or 3D) means that the loops alternate between a smaller and a larger one superposed, as if we were adding an extra dimension to the plane of a classic loop.
*We can control the range of the loop3x with the slider ******* (making microloops or macroloops depending on whether we multiply it by 0 or by 1, since we remember that all parameters go from 0 to 1.*
Another parameter will position us from where the loop slider +++ starts
*Additionally, these number fluctuations can be 'slower or faster' with the **-speed+** slider.*

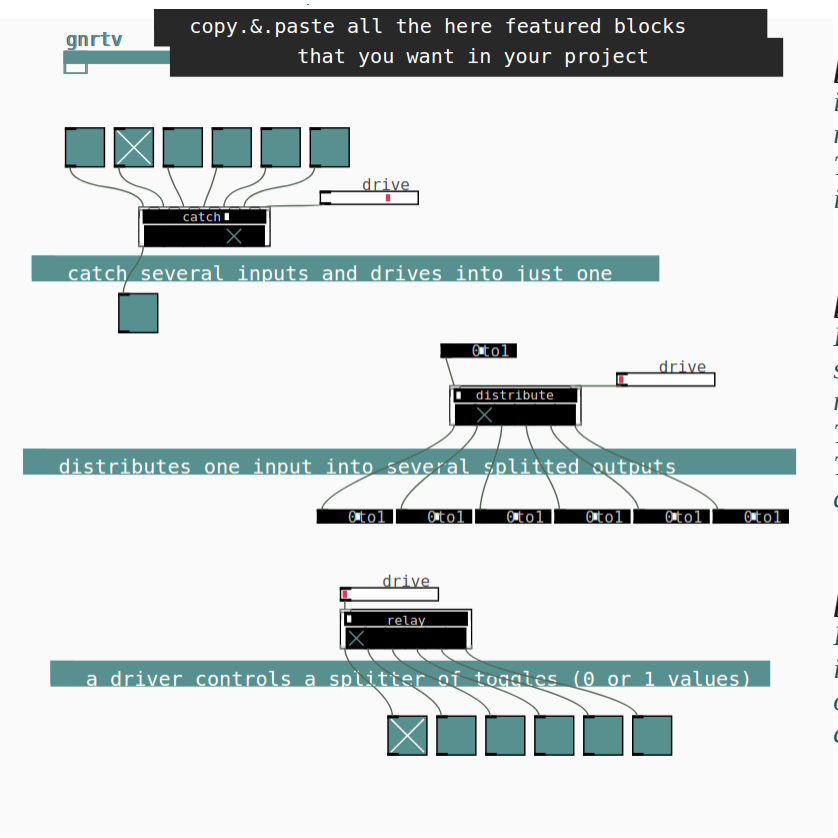
[16patterns]
*It generates random sequences of 16 steps. Depending on the position on the slider **16patterns** will create one type of random or another. The **-speed+** slider controls the speed and the button or trigger **Z**, makes variations in the tail of the sequence.*

[8patterns]
*It generates random sequences of 8 steps . Depending on the position on the slider **8patterns** will create one type of random or another. The **-speed+** slider controls the speed and the button or trigger **Z**, makes variations in the tail of the sequence.*

Section [GNRTV]

channeling

We can create a set of generative instruments with several modules from the previous ones that we can later channel or distribute with the following Blocks



[catch]
it has 6 entries that we can choose which one is the result with the slider ***catch***
The slider catch can be controlled externally with an incoming slider at the top right (ex.slider ***drive****)*

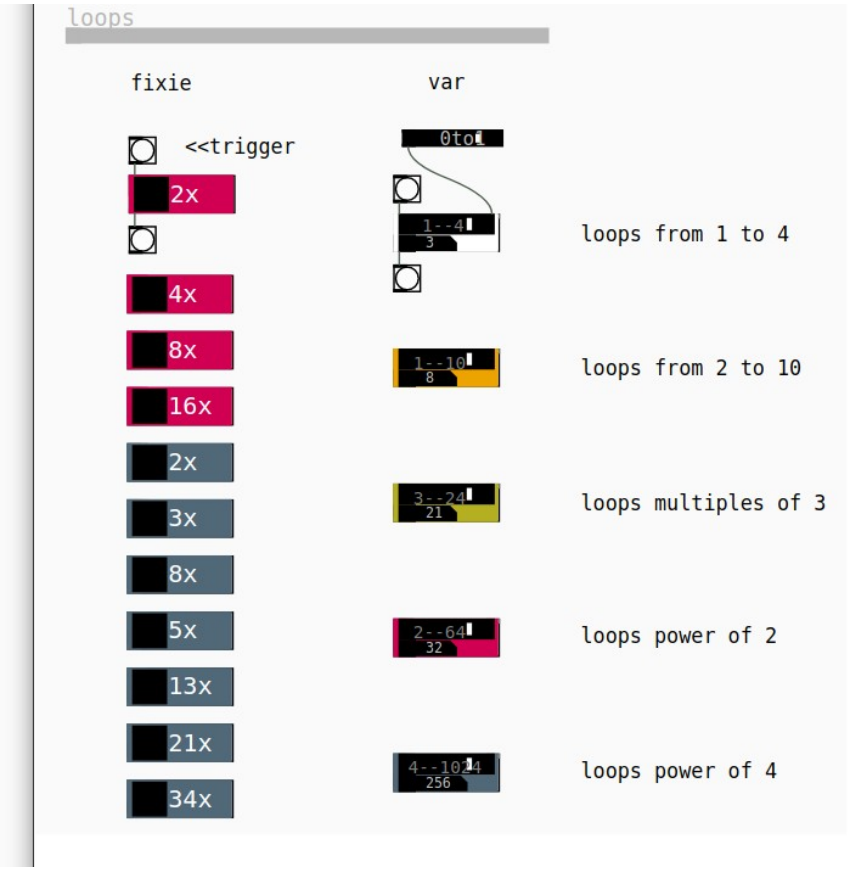
[distribute]
It is the reverse function to the previous one: we can split an incoming generator to 6 different outputs by means of the ***distribute slider.***
This function enters values between 0 and 1 inclusive.
The distribute slider can be controlled externally with an incoming slider at the top right (ex.slider ***drive****)*

[relay]
It is a similar function to the previous one: an incoming generator can be bifurcated to 6 different outputs by means of the slider ***relay,*** *although values are either 0 or 1.*

loops

As with any programming language and also as with any musical construction, we will need to create loops of some sections or algorithms that we want. In general, the way to count the cycles is to start them with triggers or bangs.

[fixie] fixed loops
The following Blocks create fixed loops (fixie) between 2,4,8,16, etc... cycles.
We connect the bang or trigger of the process we want to the top input and it will extract the result every X cycles indicated by the Block.
For example
*the Block **2x** will remove a trigger every **2** incoming triggers,*
*the Block **16x** will remove a trigger every **16** incoming triggers,*
and so forth..



[var] variable loops

Depending on the Block, it will have some ranges or others, <u>being able to make the generative growth of our instruments more dynamic and mutating and being able to leave the strict quantization of the quantized system offered by the 'fixies'.</u>
there we have :

1–4
(we can choose among 1/2/3/4 ranges)

1–10
(we can choose among 1,2,3,4,5,6,7,8,9,1 ranges)

3-24
(we can choose among 3, 6, 9,12,15,18,21,24 ranges)

2-64
(we can choose among 2,4,8,16,32,64 ranges)

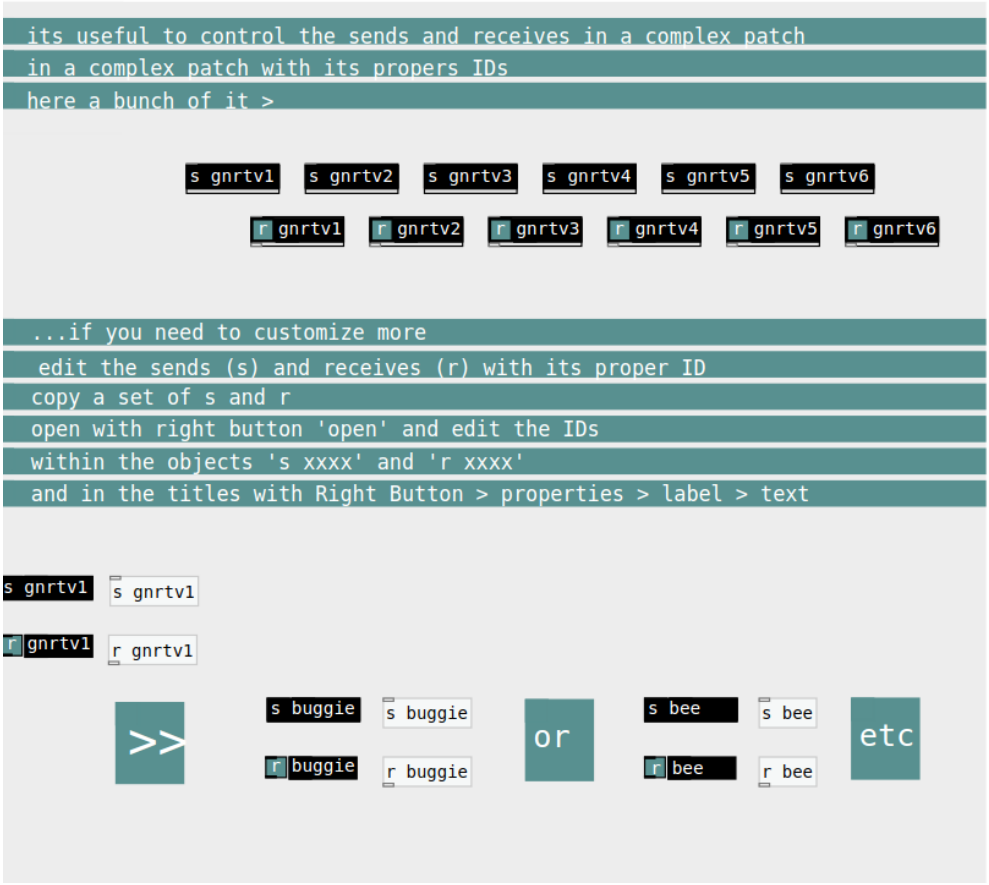
4-1024
(we can choose among 4,16,64,256,1024 ranges)

sends & receives > dataflow

GNRTV.BLOCKS *works with a visual node language (pd-l2ork or purr-data).*
It can happen that in complex instruments, the connections between blocks and Cells are increased, creating a very complex interface that creates a decrease in usability.

In order to simplify the interface and make it clearer, it is very useful to use send and receive messages (sneds and receives) which will allow the data flow to be distributed by connecting 'wirelessly' the blocks and messages we want.

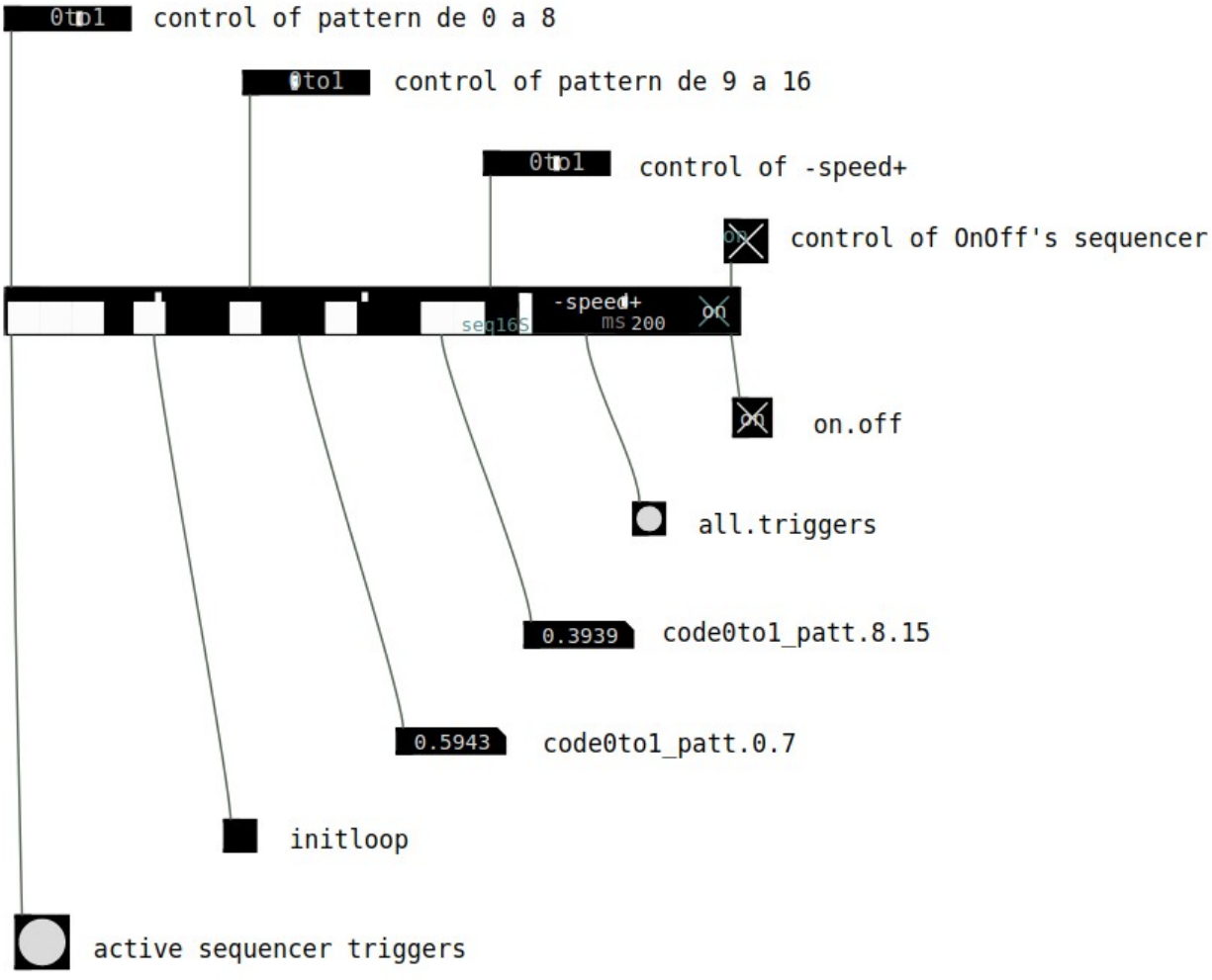
In this section we have a few messages that we can use.
We can also create new ones with the object
[s desired flow]
[r desired flow]

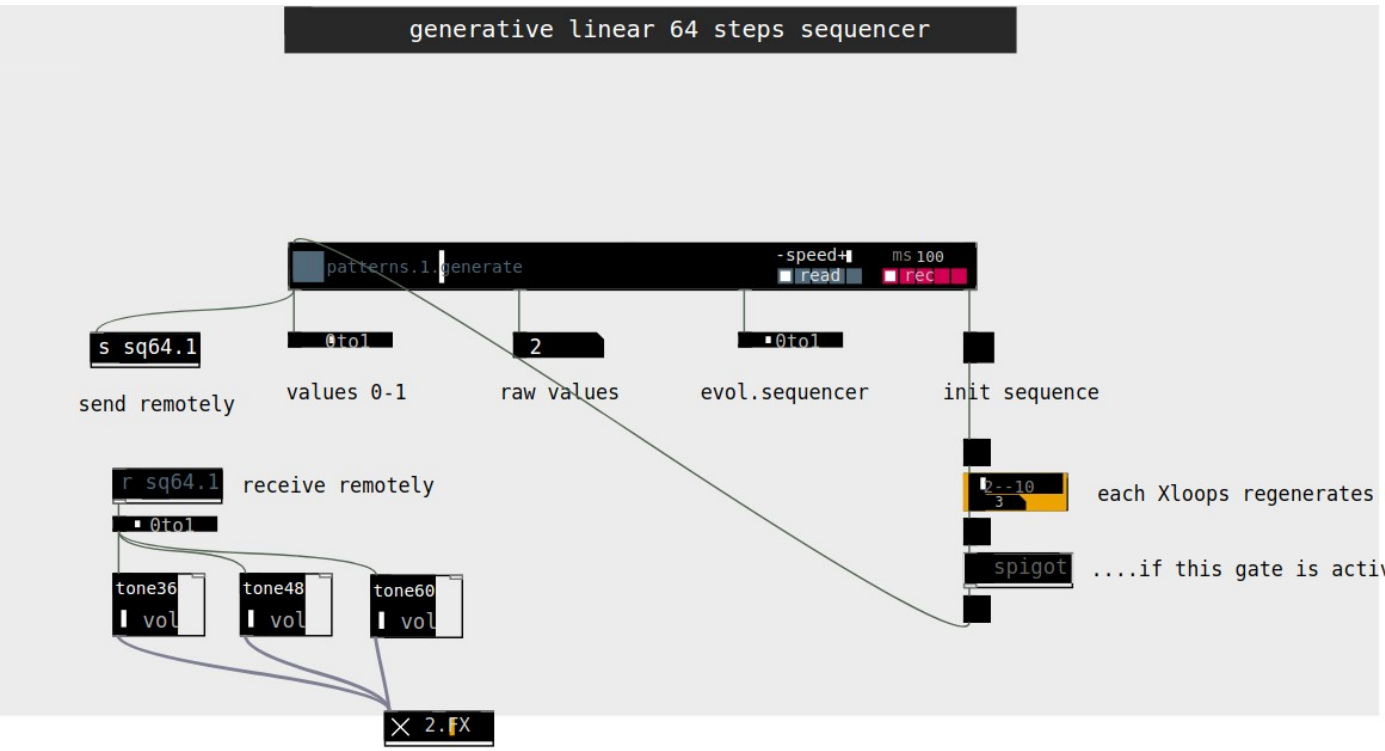


sequencers

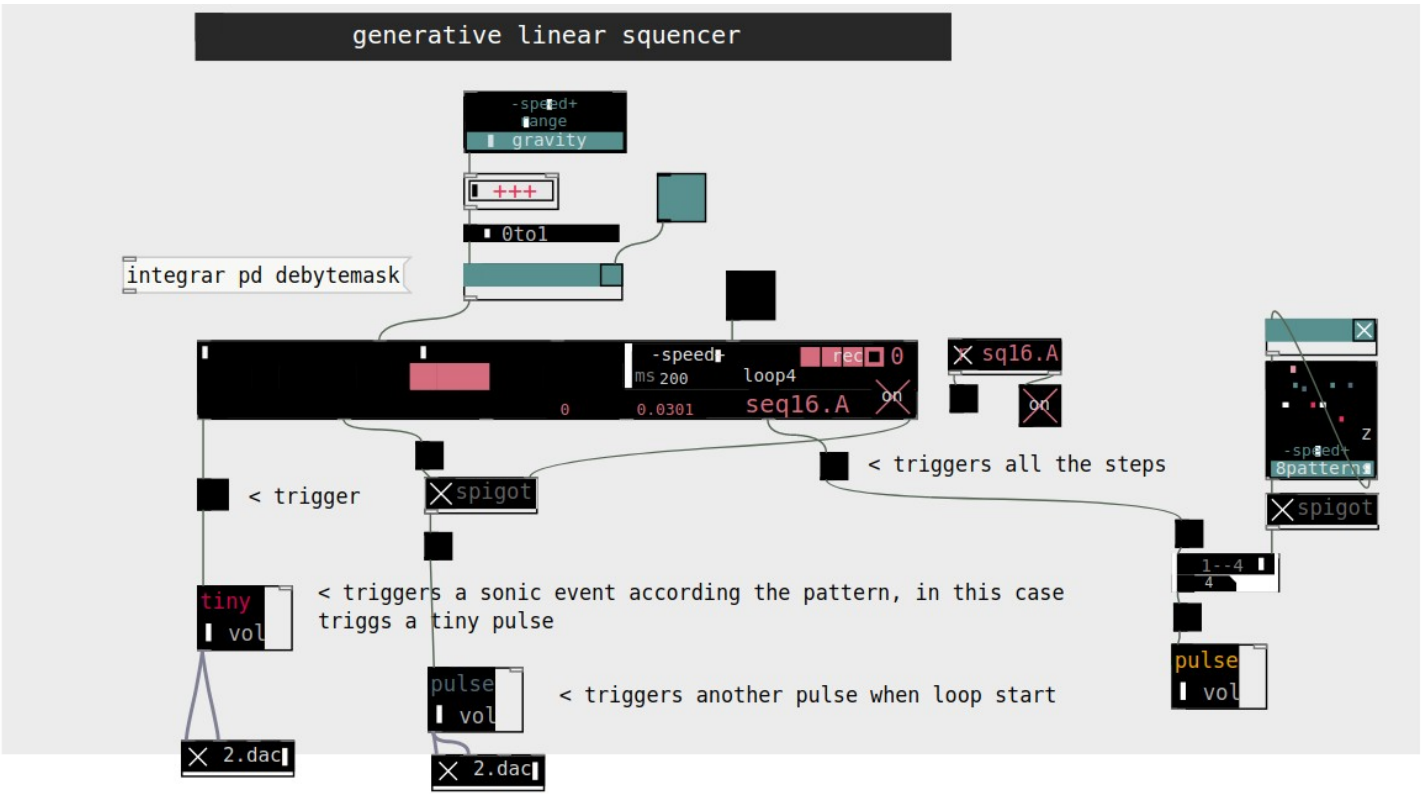
Like almost every music production tool, and also in the Modular Synthesizers ecosystem, to be able to organize more defined sequences in time than those generated by generative Blocks, we can use the sequencers, which the Toolkit has.

As you may know the Gnrtv.Cells can be connected 'everything with everything' as a modular synth, it is for this reason that the parameters to be controlled within the block can be made externally tnat input as output which will allow us to interconnect with other elements.




generative linear 64 steps sequencer
patterns.1 generate
-speed+
read
ms 100
rec
s sq64.1
send remotely
0to1
values 0-1
2
raw values
0to1
evol.sequencer
init sequence
r sq64.1
receive remotely
0to1
tone36
vol
tone48
vol
tone60
vol
2.FX
2--10
3
each Xloops regenerates the
spigot
....if this gate is active


SEQ
uencers


generative linear squencer
-speed+
range
gravity
+++
0to1
integrar pd debytemask
-speed+
ms 200
loop4
rec 0
0
0.0301
seq16.A
sq16.A
on
< triggers all the steps
-speed+
8patterns
z
spigot
< trigger
spigot
1--4
4
tiny
vol
< triggers a sonic event according the pattern, in this case
triggs a tiny pulse
pulse
vol
< triggers another pulse when loop start
pulse
vol
2.dac
2.dac

SENYAL d'ÀUDIO~

The previous elements and methods described refer to algorithmic procedures and numerical and data processing, but we haven't 'sounded' anything yet.

The blocks responsible for Audio management are GENERATORS and Racks

Those corresponding to the Sonorous generation are the CELLS and the Signal Loopers that fall within the electronic sound typology of Samplers.

GENERADORS d'ÀUDIO~
CELLS SYNTHS

Cell Synths *are, as their name suggests, Synthesis Cells, i.e. autonomous synthesis modules that can 'metabolize' with other 'cells and/or organs'.*
Each CELL is designed for a specific role.

This example creates the synthesis of kicks or 'bombos' simulating the famous instrument included in a physical drum set.

Each cell has between 6 and 7 parameters that we can adjust.

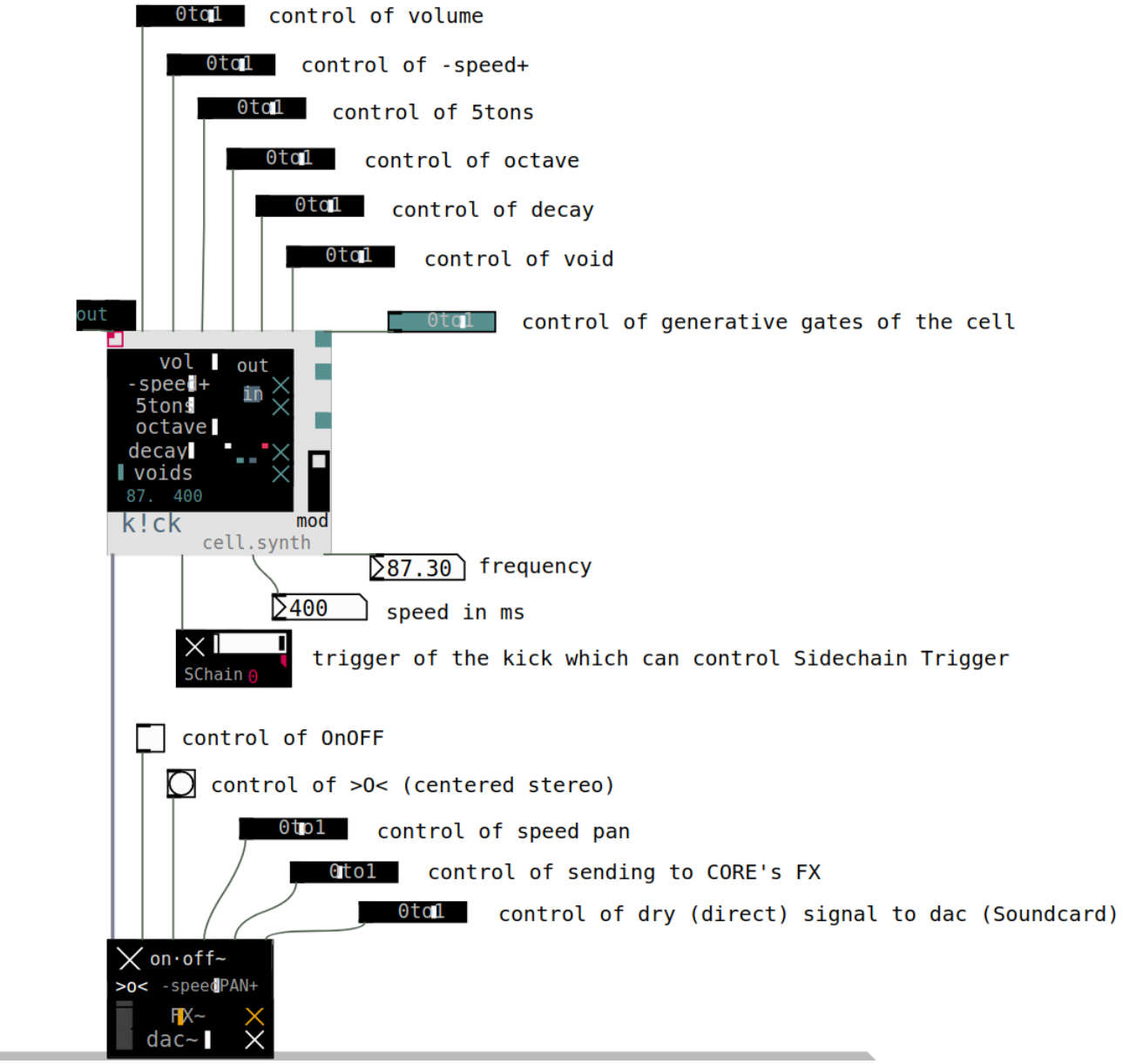
In this case those that appear in the figure.

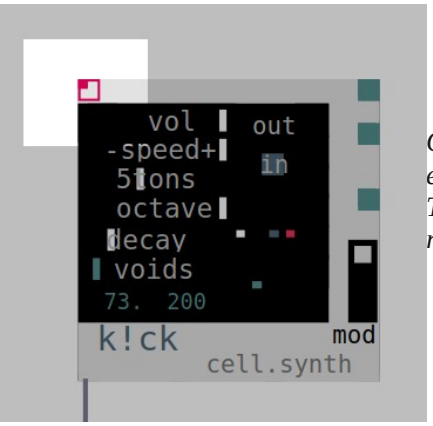
It should be noted that the ***-speed+*** *parameter is connected to the clock mastertempo included in the* ***Core.*** *Therefore, if we create a new project with only this module, we will also need the* ***CORE module*** *in the same patch or project.*

The ***-speed+*** *parameter quantizes between different beats (in the classic dodecaphonic system that corresponds to quarter notes, quarter notes, half notes, etc.). Therefore, without the need for an additional sequencer, the cells that have the* ***-speed+*** *parameter already delay sound events in time, but yes: quantized or 'squared' without the fact that we can build other more complex or polyrhythmic rhythms, which would require additional sequencers in the same cell.*

The CELL kick is connected to a Rack that will send the acoustic signal produced by the Cell either to the Effects (slider FX~) or directly to the sound card: dry (slider dac~)
It also includes an LR randomizer to do binaural tricks and left-to-right changes. The >o< trigger will refocus the sound when we want.

The previous CELL can be used by itself or augmented with generative programming. In this sense any of the parameters included in each block (the CELL and the RACK~) can be controlled outside the Block and be able to interconnect it with other parts of the programming.



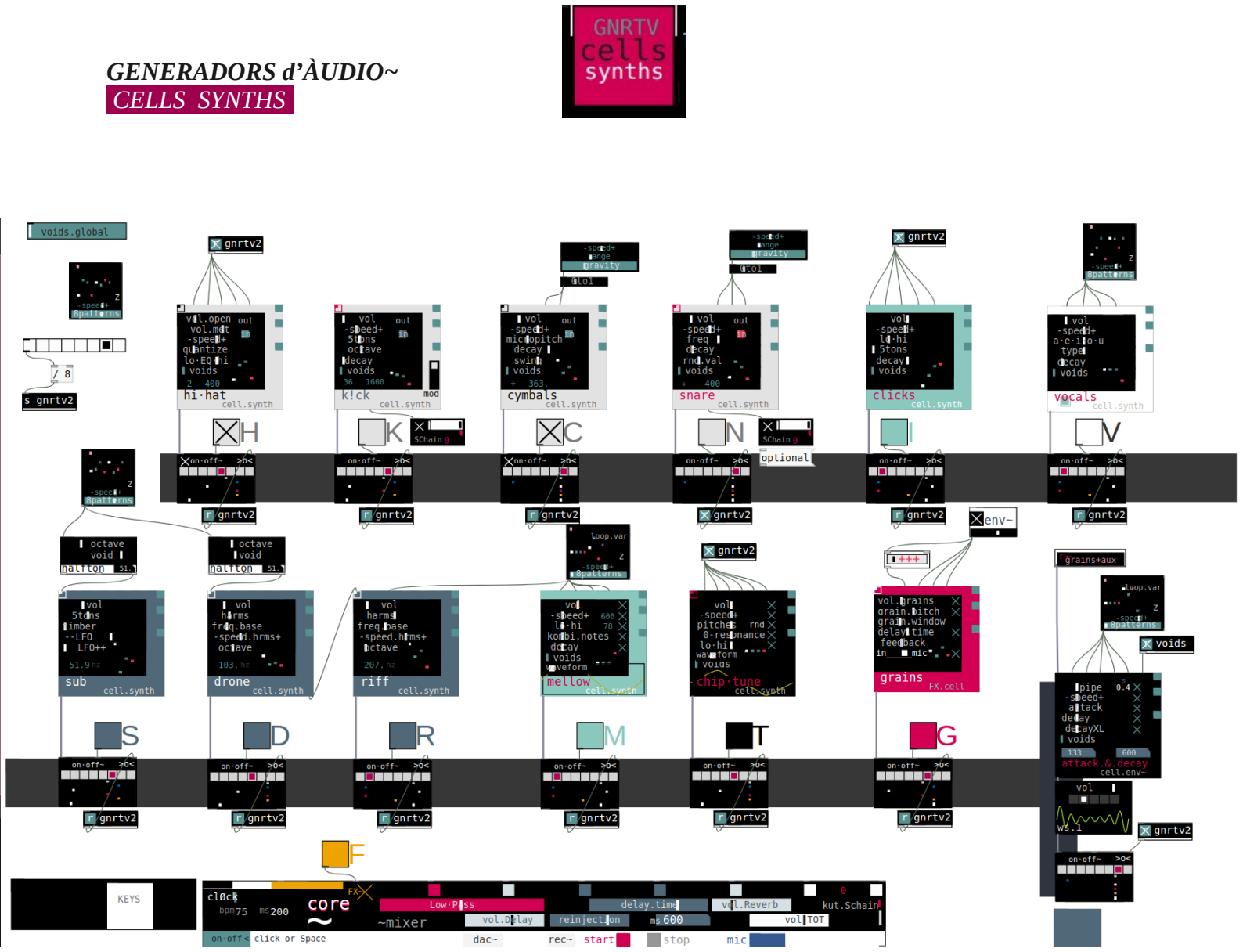


Cells with this symbol in the upper left indicate that these instruments can be sequenced externally.
This means that we can leave the 'square' quantization where we can build more complex rhythmic and polyrhythmic.

Continuing with the previous comment, we see how we can connect a 16-step sequencer associated with the KICK CELL.
The sequencers as it could not be otherwise can also be interconnected with other instruments and modules both input and output.

GENERADORS d'ÀUDIO~

CELLS SYNTHS



Recap : CELLS are synthesis modules. Each CELL is designed for a specific role.

***Hihat**: Synthesis module that simulates with physical models the sound of the hihat, hihats or brass that often accompany the snare drum and bass drum in a drum kit.*

***Kick**: Synthesis module that simulates the bass drum in a drum set.*

***cymbals** : synthesis module that simulates with physical models the sound of higher and more open metals that often accompany other elements of a drum kit to create tension*

***snare**: synthesis module that simulates the sound of a drum case with physical models.*

***clicks** : abstract synthesis module that throws particles and 'clicks' and pulses of different tones at various octaves*

***vocals**: formant synthesis module that simulates the human voice. Given the parameterization, it allows the extraction of registers that exceed below and above the usual in human speech.*

***sub**: Synthesis module that generates a sub-bass line. Includes two types of LFO) Low Frequency Oscillation)*

***bass** : synthesis module that generates a drone-type bass line with different harmonization settings, base frequencies and octaves.*

***riff** : synthesis module that generates a line of low-medium-high riff type basses and guitars with different harmonization settings, base frequencies and octaves.*

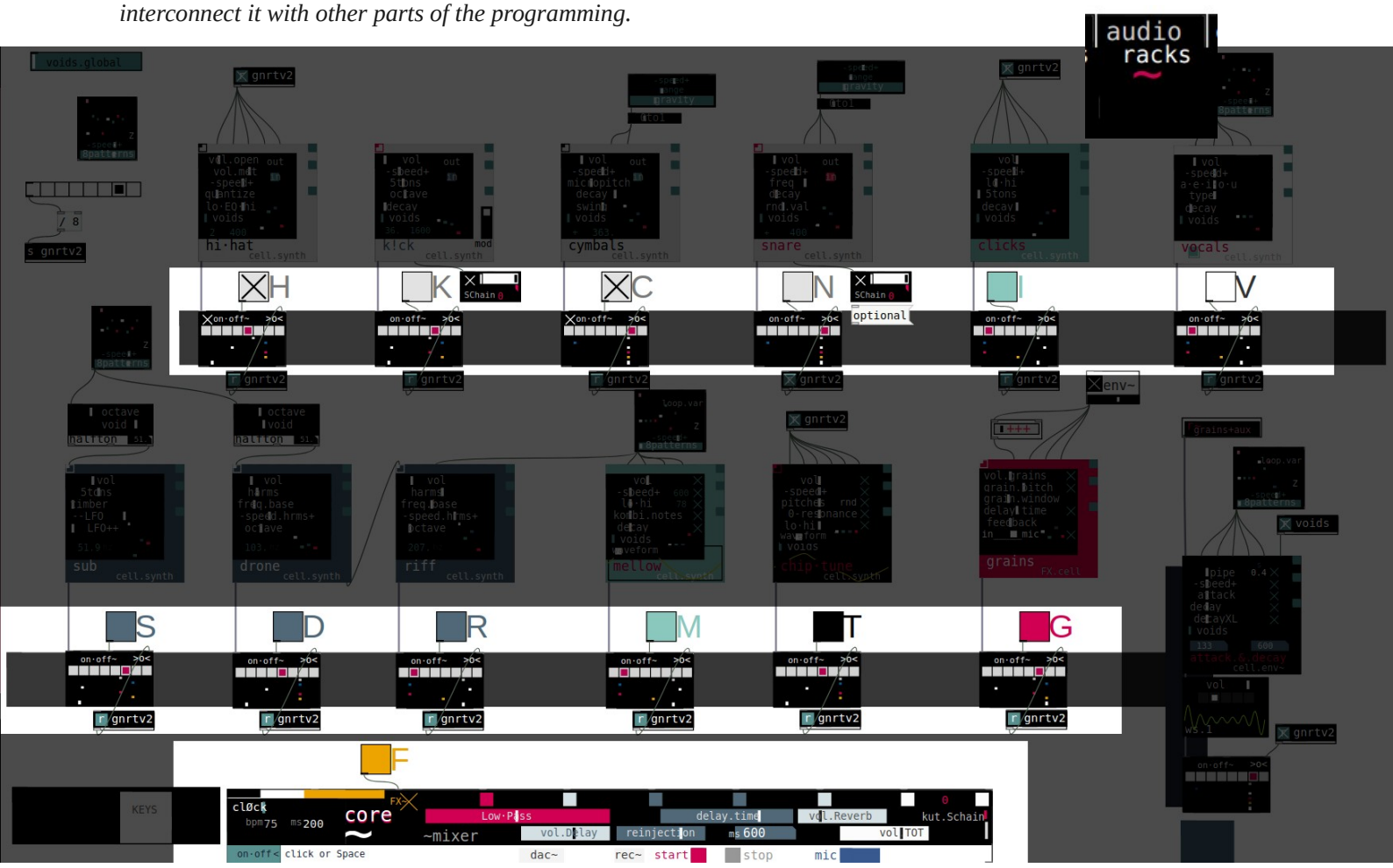
***mellow** : polyphonic synthesis module, which generates melodic sequences between different octaves and combinations of notes and size in milliseconds (decay) of the notes*

***chiptune** : synthesis module that generates sequences of the 8-bit or chiptune type, common in Retro Gaming contexts or in the beginnings of the Gamer era given the low sound resolution.*

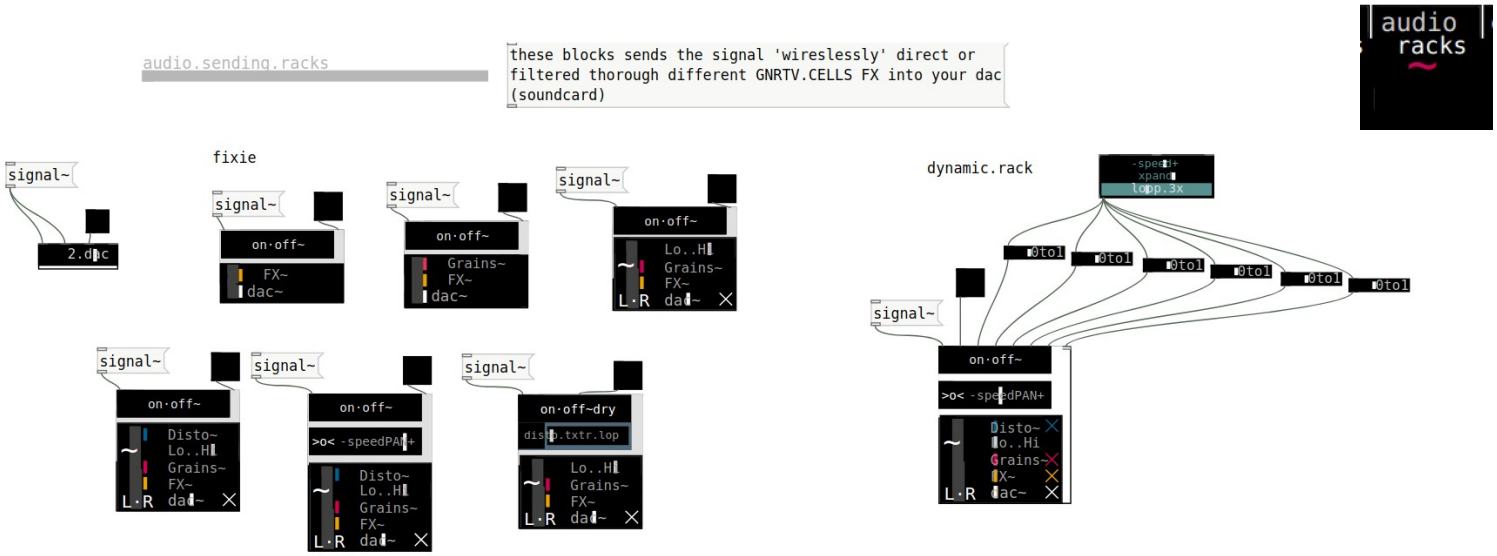
***grains** : granular synthesis module that serves as texturing or SFX of any of the previous cells as long as we have the sending of the Rack the Grains slider.*
This CELL also allows processing via granular synthesis the line input or the laptop's embedded microphone.

Attack & decay: envelope control module (attack & decay) of any of the previous CELLS. Also sent by the Rack Grains slider

The above CELL can be used by itself or augmented with generative programming. In this sense any of the parameters included in each block (the CELL and the RACK~) can be controlled outside the Block and be able to interconnect it with other parts of the programming.

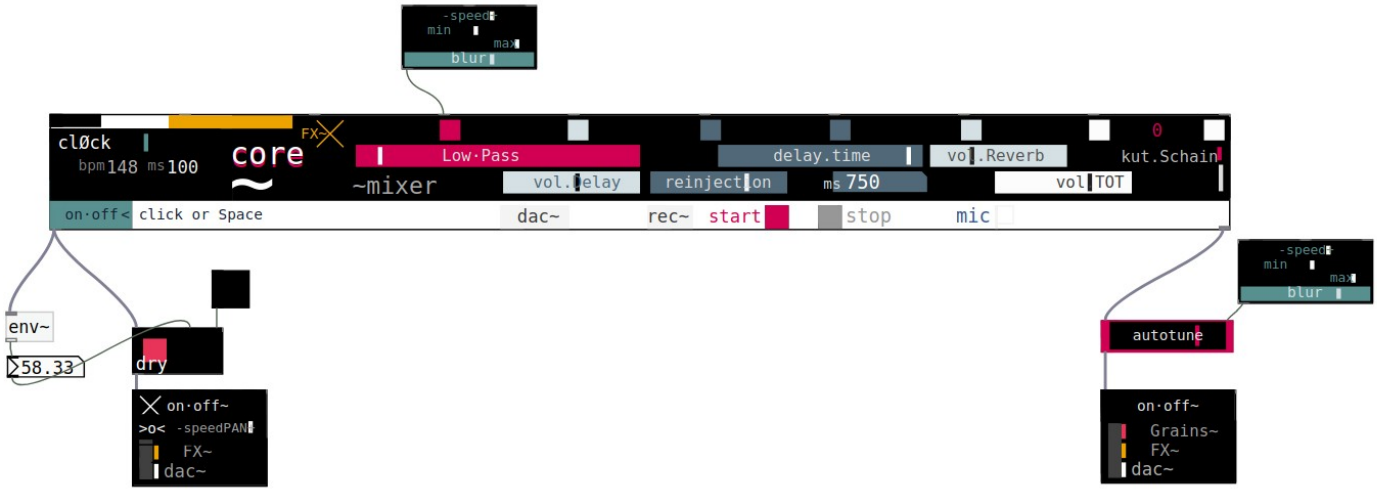


Racks *can be of various types but they always perform the same function: to send the acoustic signal that the sound generators are producing (Synthesizers, Audio Loopers, samplers, etc.).*




```
      usage

these blocks are the pretty the same from simplest to more
complex.

  In any case there are blocks that are in charge of sending :

     1 incoming audio signal into the same soundcard   (sliders dac~)

     2 incoming audio signal into the FX Core Rack     (sliders FX~)

     3 incoming audio signal into the GRainer FX CELL  (sliders Grains~)
```

*In addition to the previous example, we can add a degree of organic automation with the **Generative** layer, which we can associate with whatever we want: Synthesizers, Racks and even the **CORE** (see image below)*

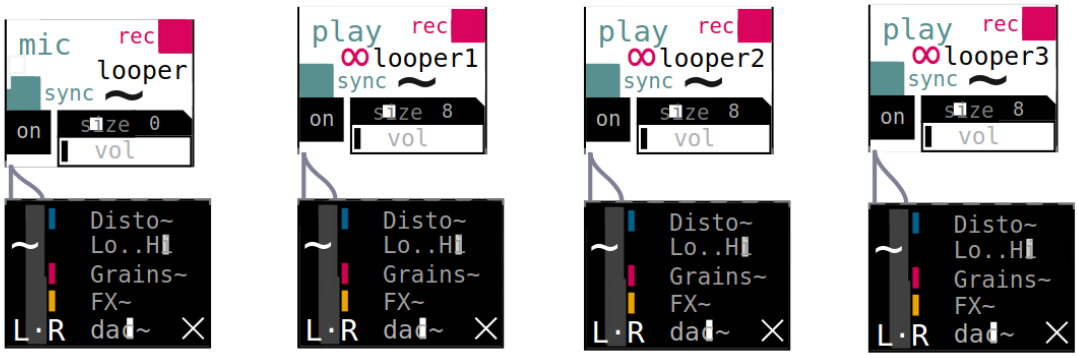


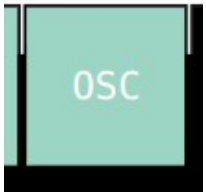
AUDIO GENERATORS~
AUDIO LOOPERS

Another method of generating audio is to sample what we want from what is being played (a fragment of the code or the final output extracted from CORE).
Audio Loopers are famous in some Indie genres where a fragment of what the musicians are performing is captured and superimposed on several layers.

*We will have the '**play**' Blocks that sample the output of the desired module that is reproducing the acoustic signal and the '**mic**' that captures a fragment of the system's microphone and/or line input. (the OS's microphone execution permission and its capture volumes must be given).*

samplers+audioloopers

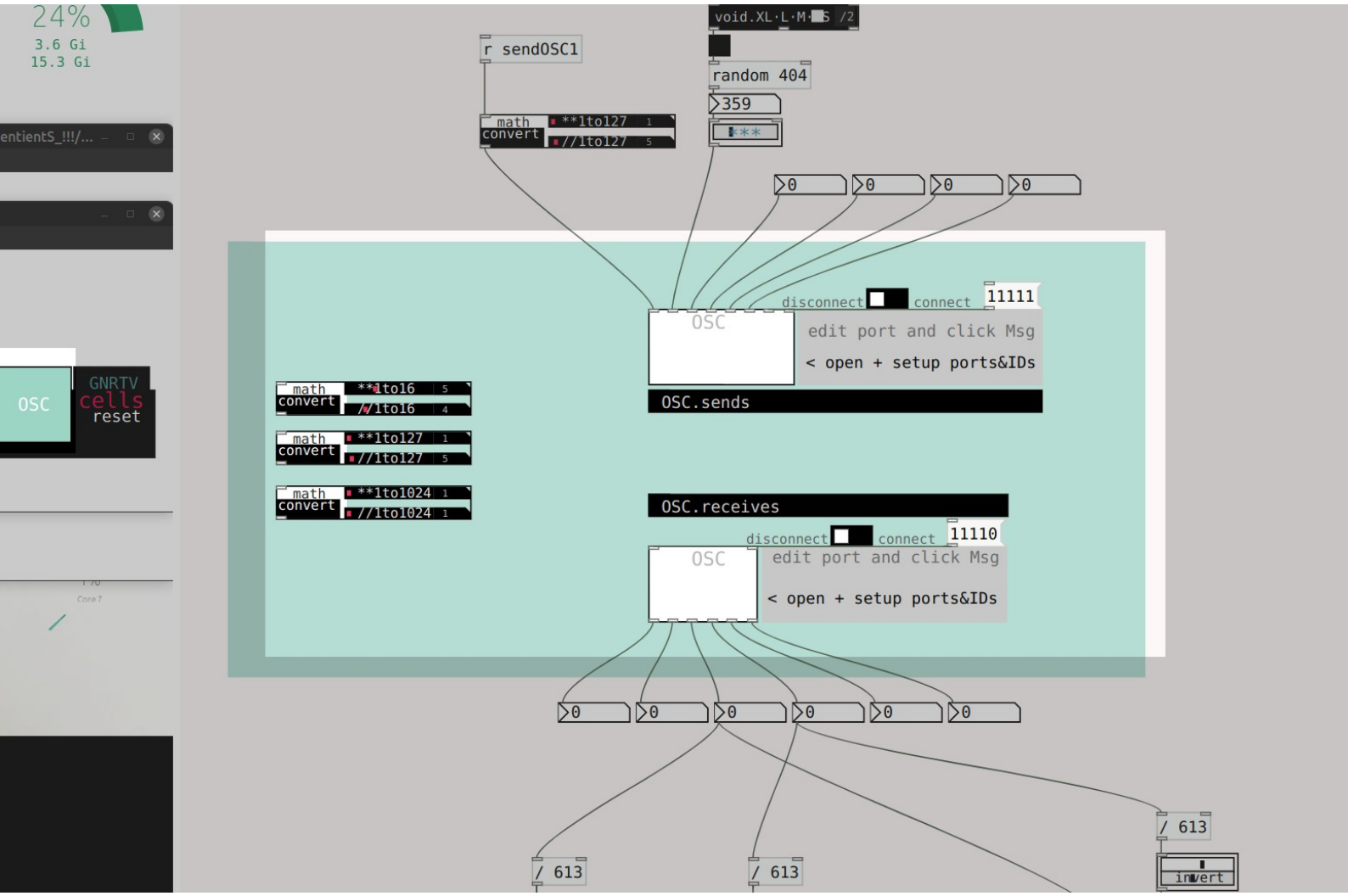




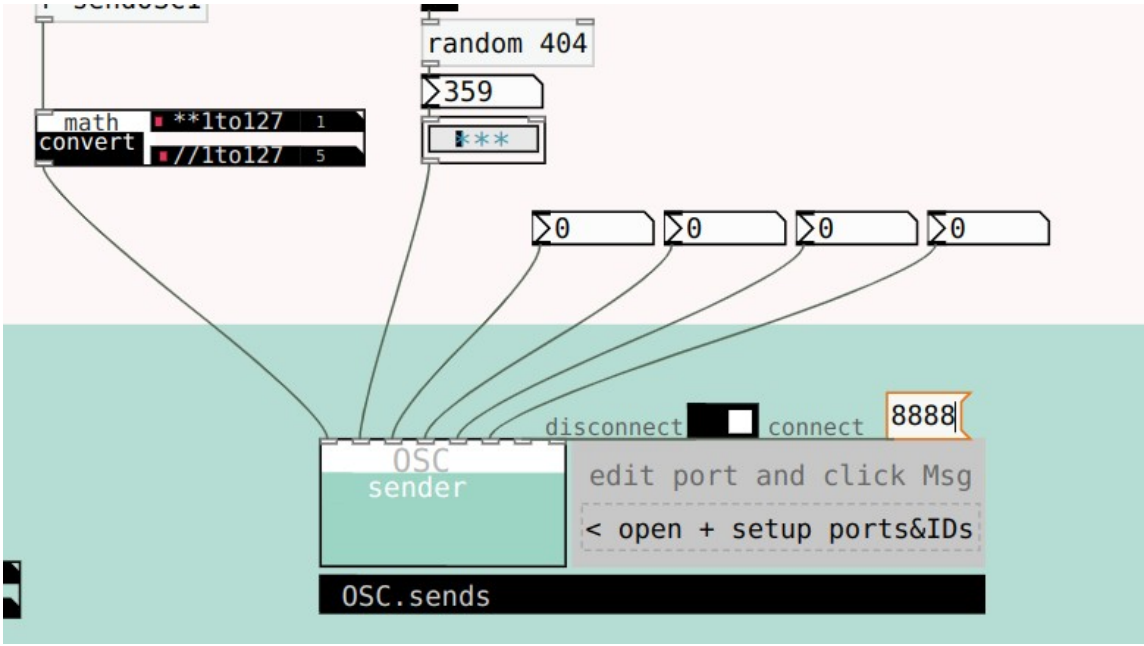
INTERACTION

OSC

Open Sound Control *(OSC) is a communication protocol between applications and devices (PCBs, sensors, mobiles, etc.), in order to create a networked interactive system. We can also connect several applications and languages on the same device, being able to perform several tasks that there will usually be tools that can solve them better. For example, if we want to make interactive visuals, we can make them with Processing and communicate them via* ***OSC*** *with a live audio engine (such as GNRTV.CELLS). The result will increase in complexity and synchrony of sound and visual events.*
GNRTV.CELLS has an OSC communication module.

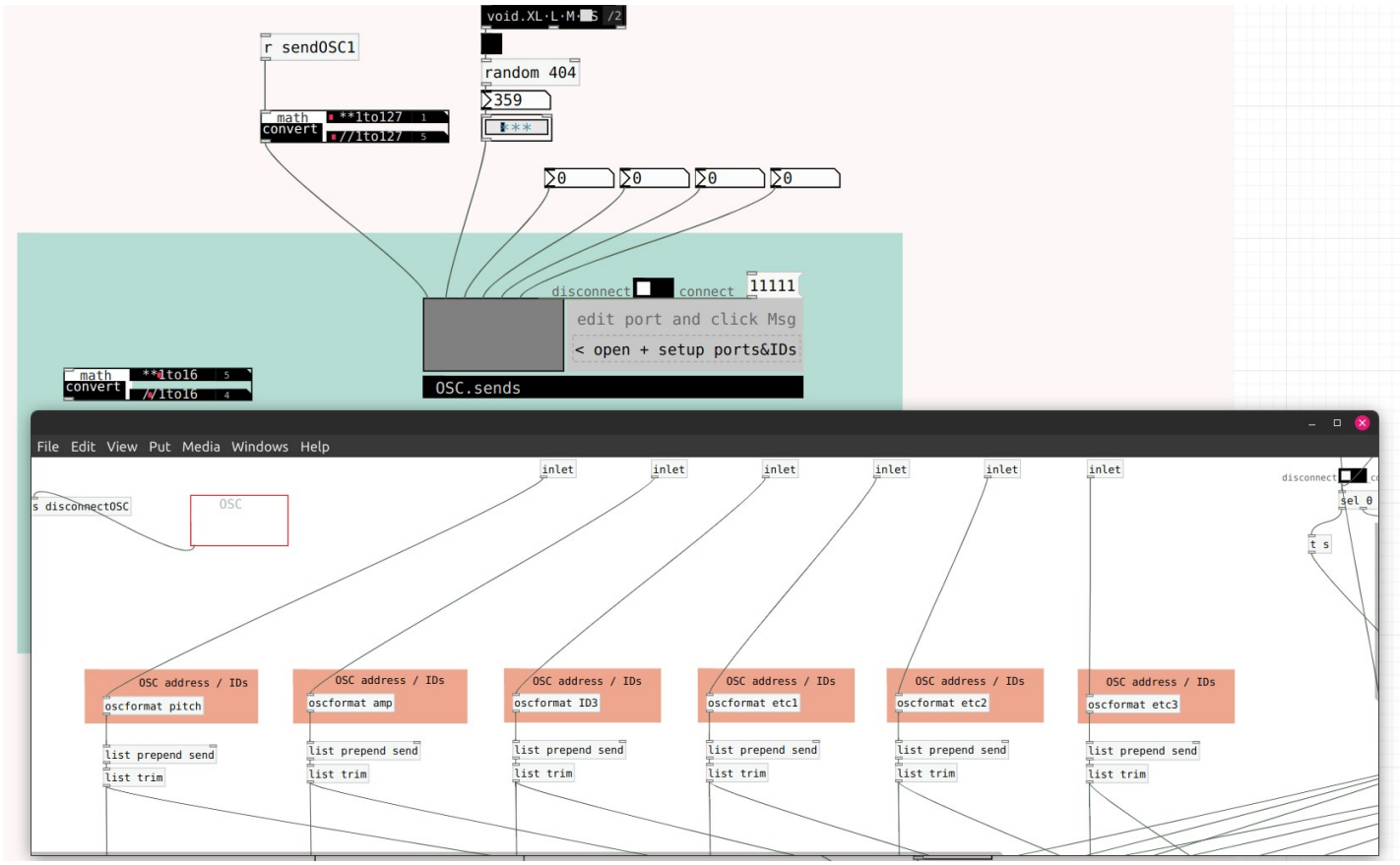


We can use the protocol both to send data outwards (OSC sends) or to receive them from another application or device (OSC receives).

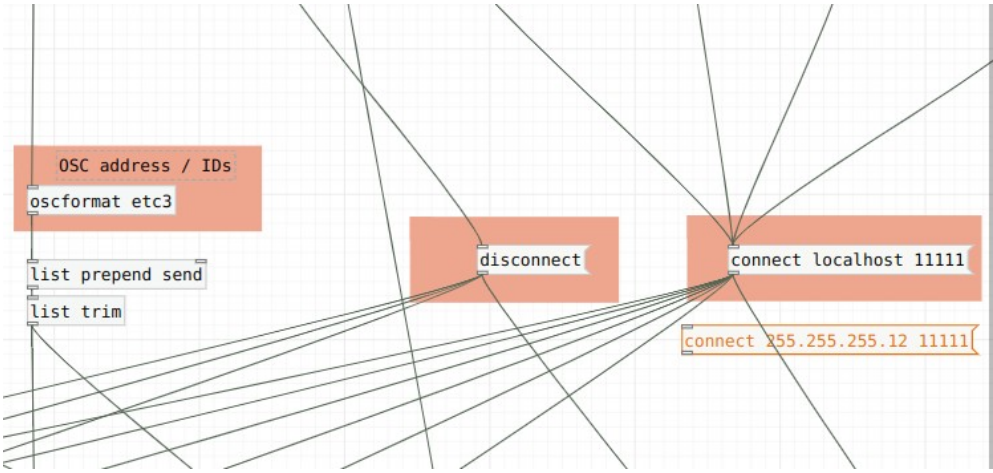


OSC sends

>Sending ID (OSC communications are like a highway: we go along AP7 and AP7 would be the ID) therefore we will edit the ID we want in the oscformed object.
[oscformat ID]



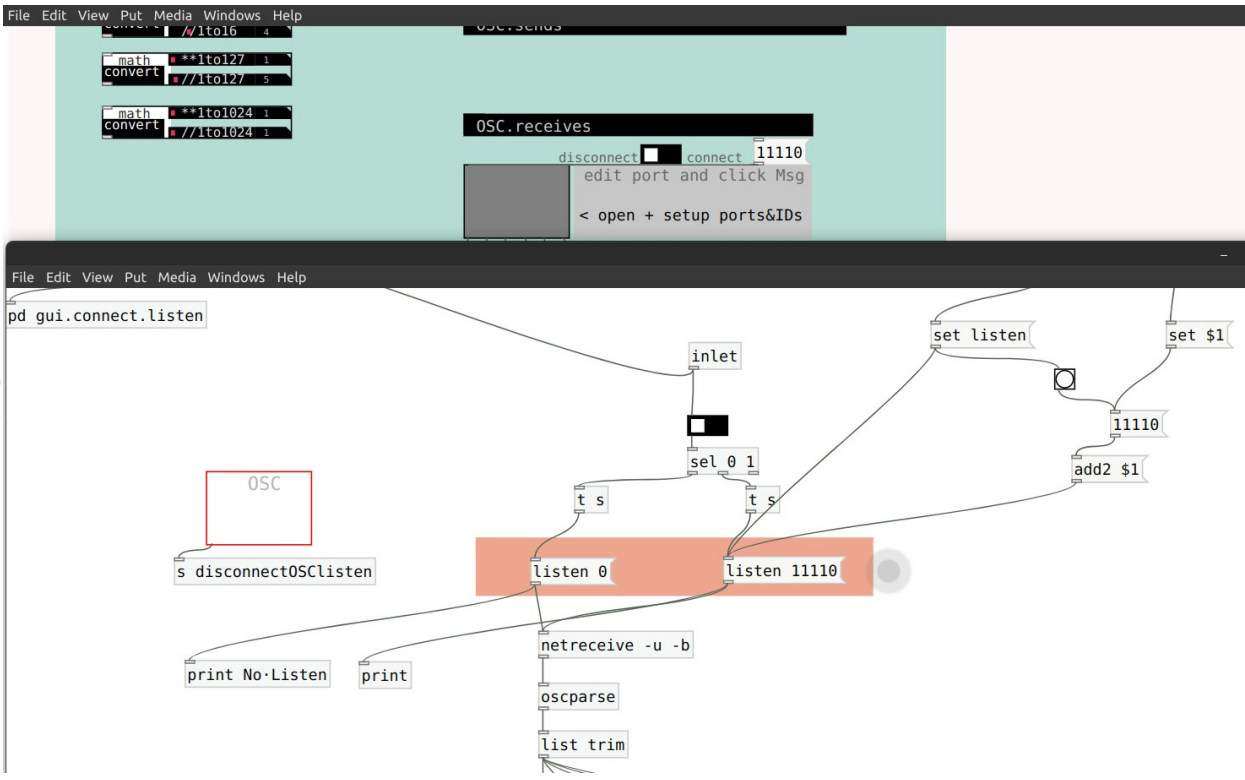
>listen (OSC communications are like a highway with thousands of lanes: let's take the AP7 on lane 5000)
[8888[
[11111[
note: if there are OSC connection problems try with high ports (between 8000 and 10000)



Inside the OSC.Sender block we can edit (with right button > Open and edit contents with Ctrl+e or Command+E)
[oscformat ID]
[connect localhost 11111[*same device*
[connect localhost 11111[*device to connect in local network*

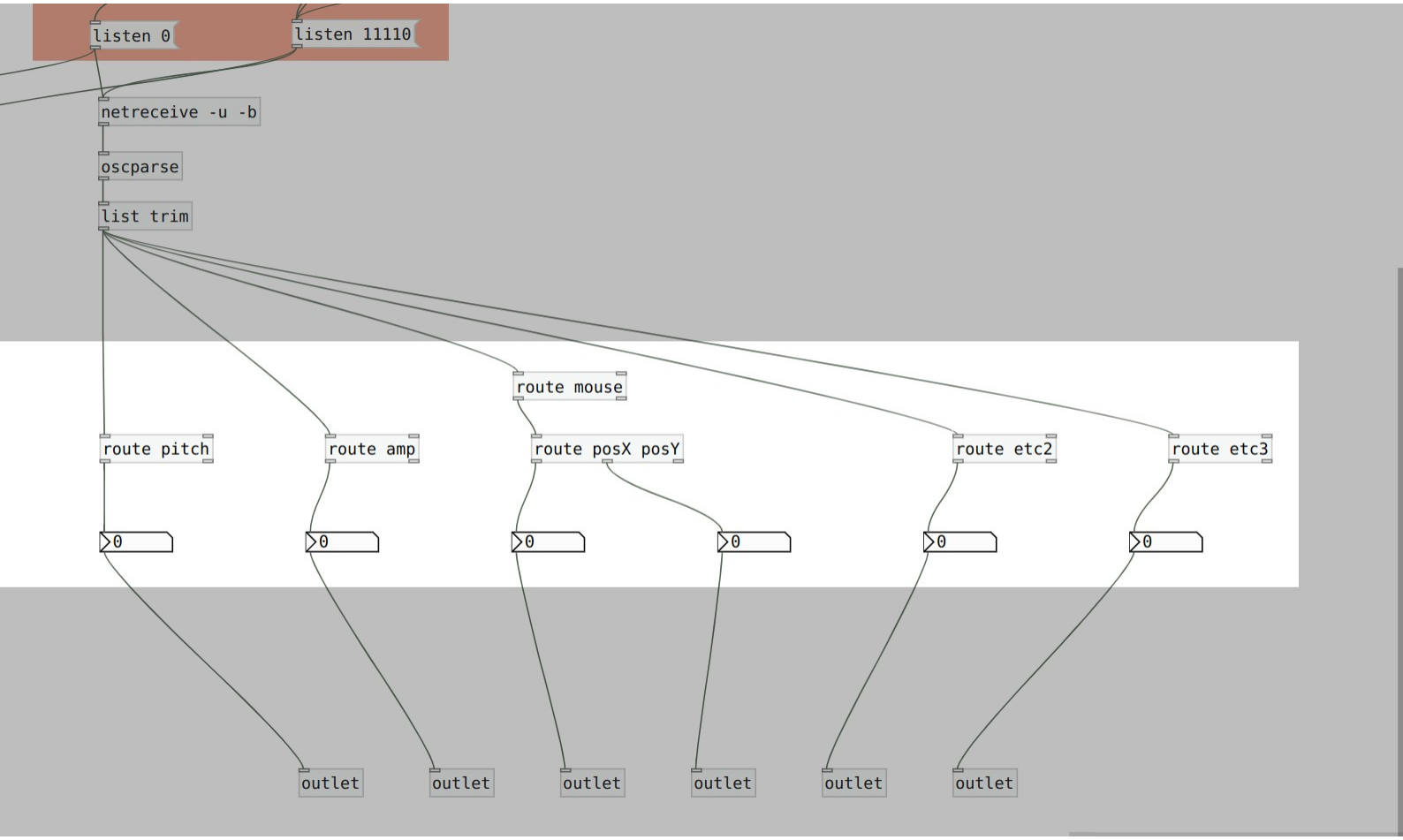
OSC receives
OSC listener

>listen 11110 (connect to the same machine on port 11110)
[listen port[



OSC receives
OSC listener

>Receive or listen ID (OSC communications are like a highway: we go along AP7 and AP7 would be the ID) therefore we will edit



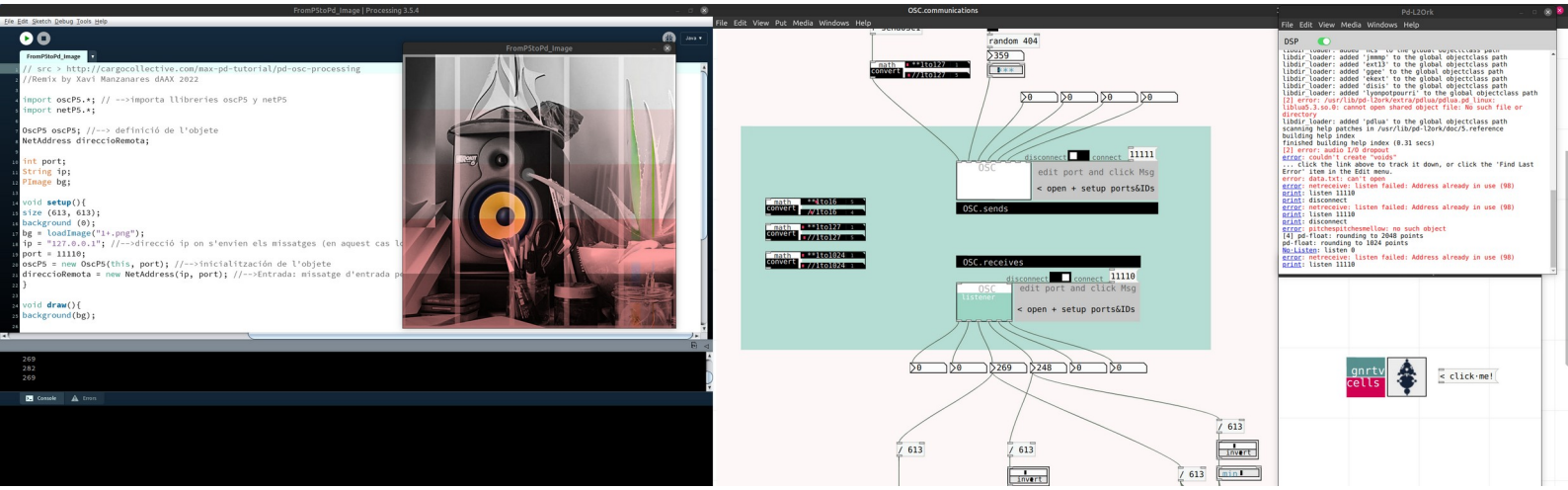
If we have a single dispatch ID from the application we will edit ***routeIDX***

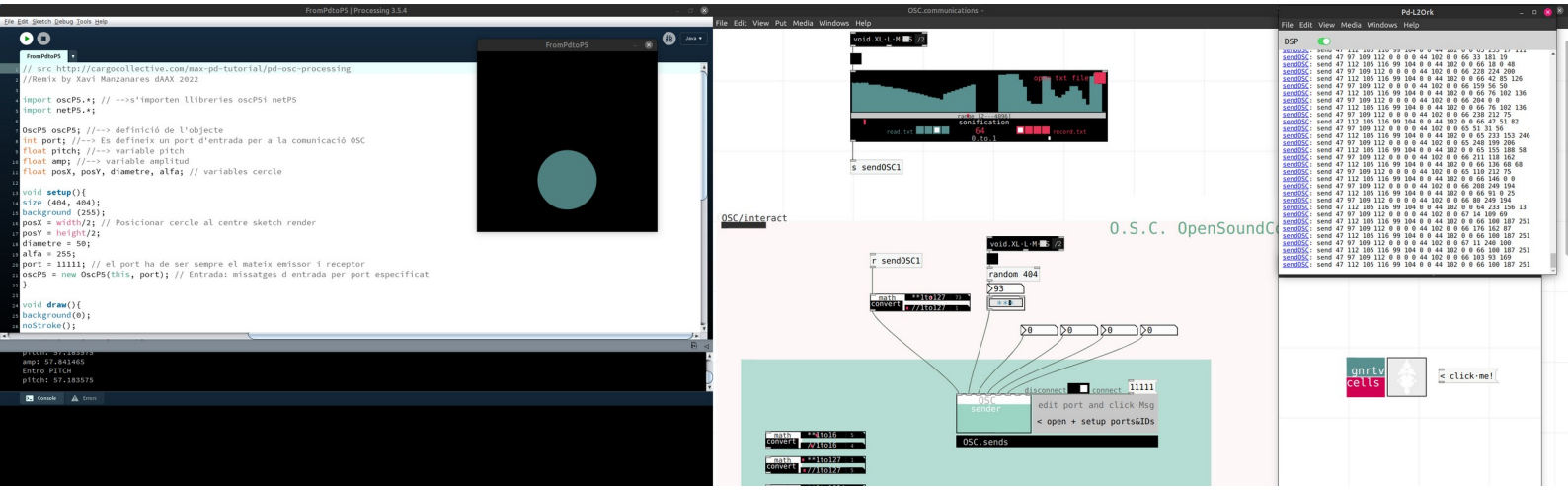
If we have a composite ID of sending from the application we will edit
[route IDX]
|
[route IDX1 IDX2]

in the example

[route mouse]
|
[route posX posY]

Once the setting is done and checking that the source from another application is active to create the OSC gateway we can communicate between two applications.



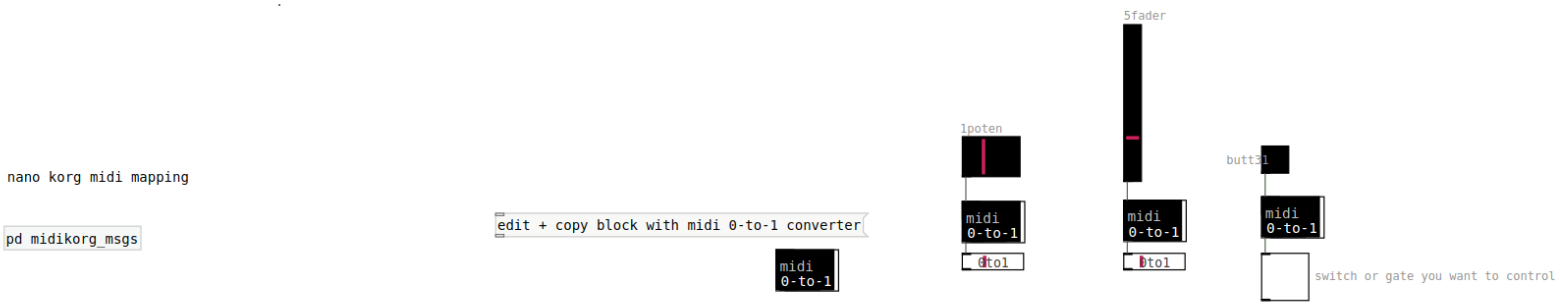


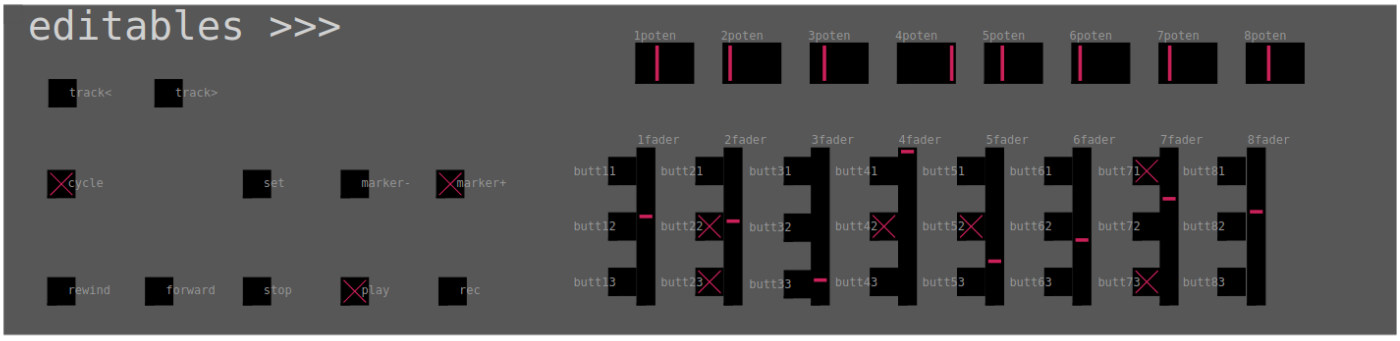
midi

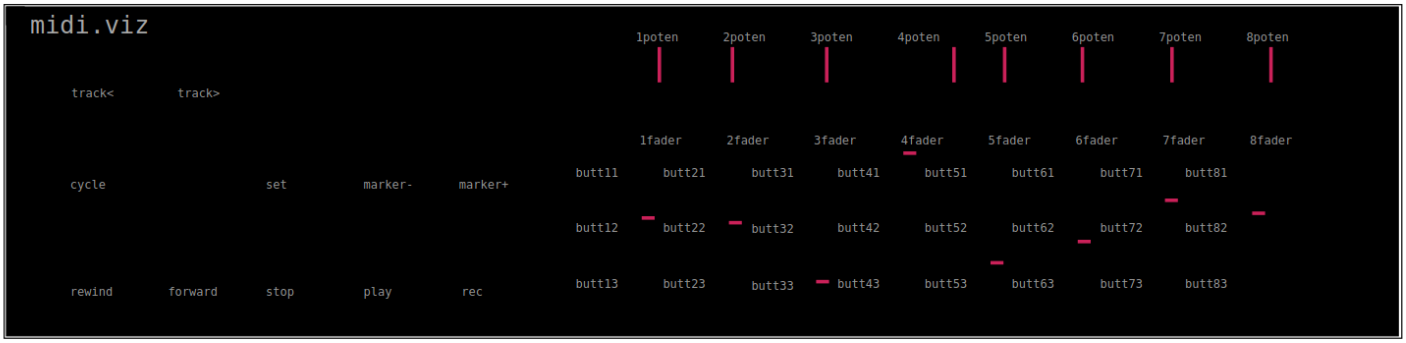
INTERACTION
midi

Midi is a classic protocol in music technology.
It arose with the appearance of numerous electronic production machines such as samplers, sequencers, synthesizers etc.. in the 80s so that they could intercommunicate with each other. Subsequently, with computerized musical technologies, it has been one of the protocols still in force to communicate production widgets with different characteristics.

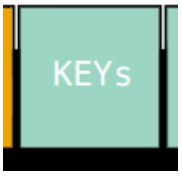
One of the most common uses is the connection of midi controllers for the customization of parametric controls to be selected in the performance.
At GNRTV.CELLS we have a template that will collect the midi data for the korg nanocontrol2 controllers which is one of the most accessible and practical on the market.





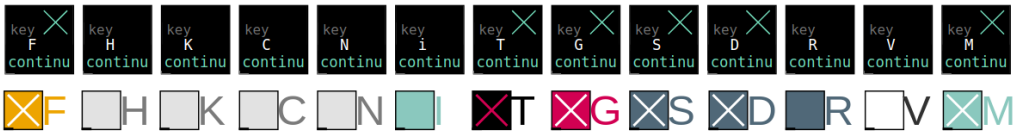


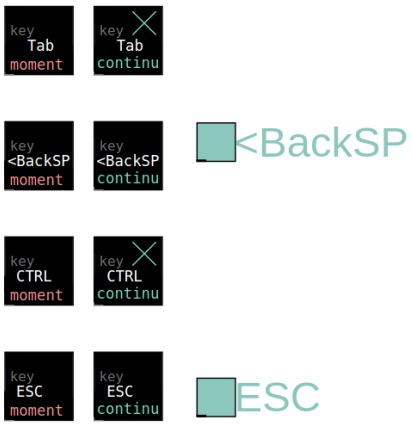
The use of the korg controller allows us to build embedded instruments and electronic luthierism with the technical sequence GNRTV.CELLS + RaspberryPi 4 + KorgNanokontrol, we can make our physical instruments such as this one:



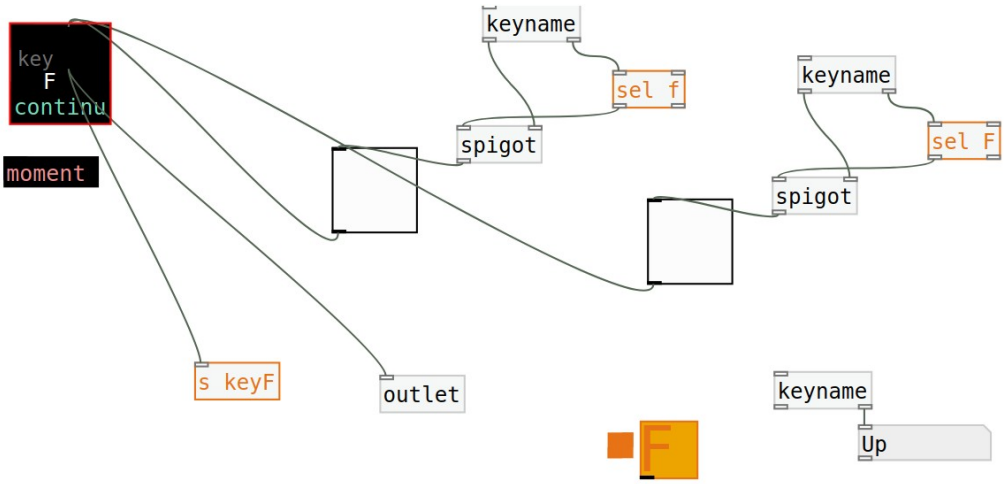
INTERACTION
keyboard

Without the need to have midi controllers we can also use the laptop keyboard keys to control our live application.

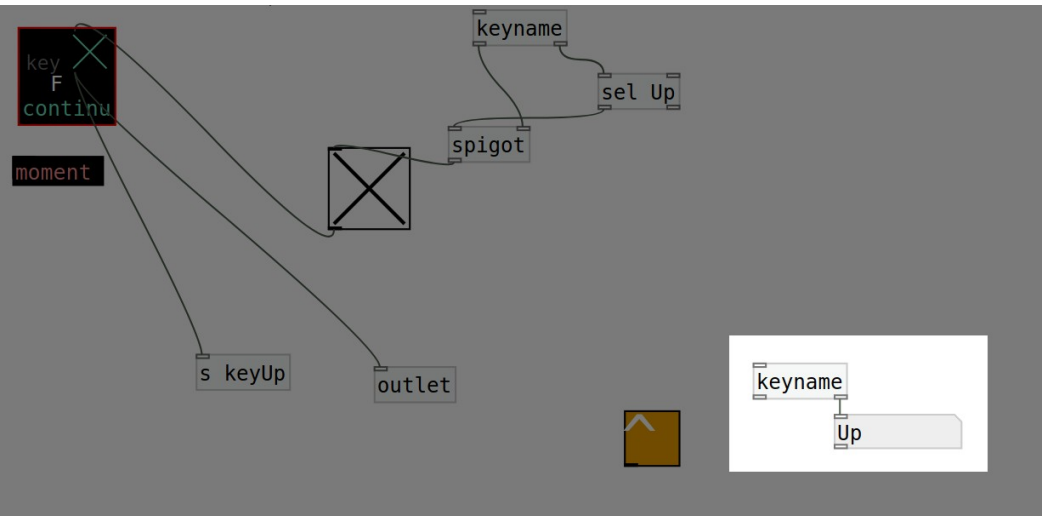




GNRTV.CELLS has a few, but you can customize more by cloning one of the key blocks, and opening the object:
right button > open

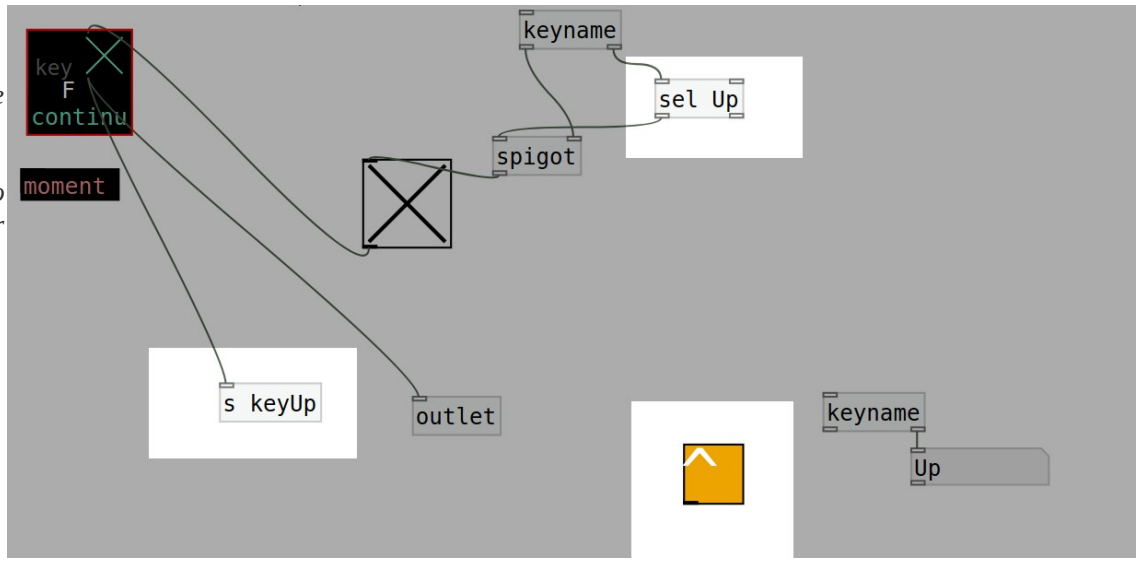


if we want to create another active key we will have to look at what log leaves us with keyname

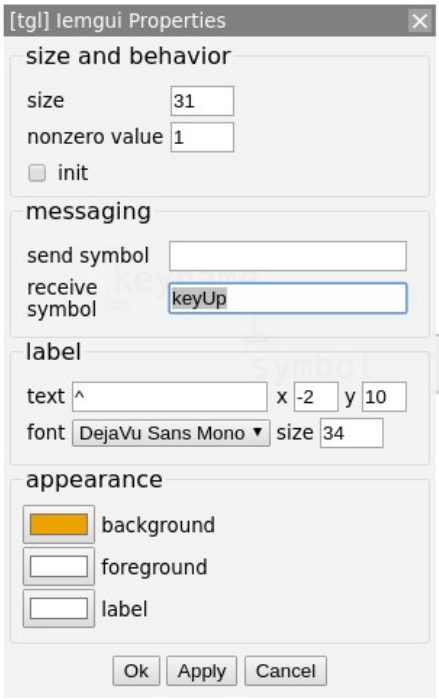


We will then edit these messages for the new key.

Sel Up *(sel is analogous to the conditional if of other languages.)*
s ***keyUp***

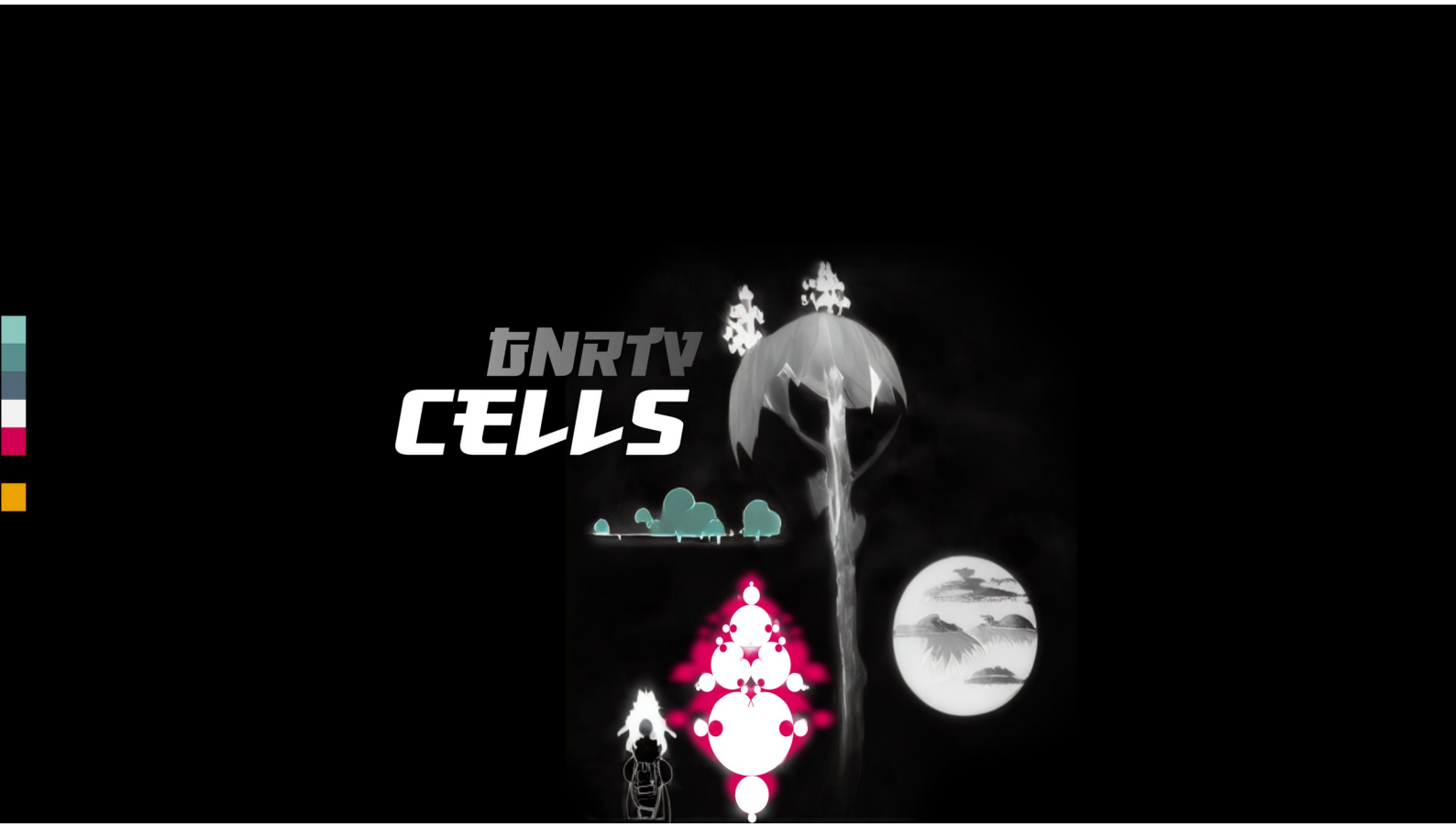


Additionally, if we want to have the Toggle (yellow switch) active and related to this customized key (in this case the Up key) we will have to enter the toggle with the right button > open and edit the message in ***Receive symbol****, in this case:*



keyUp

enjoy!



download & install at >
https://github.com/xamanza/GNRTV.CELLS