

CICADA KILLER

A reactive sound ecosystem for horn, trombone, electronics and light

Joan Bagés i Rubí

With Duo Atem — Sören Hermansson, horn · Ivo Nilsson, trombone
Flix · Stockholm · Barcelona, 2024–2026

Contents

PART ONE · THE PIECE	2
PART TWO · HOW THE SYSTEM LISTENS	7
PART THREE · HOW IT DECIDES	11
PART FOUR · THE ELECTRONICS	15
PART FIVE · THE LIGHT	17
PART SIX · VERIFICATION	27
PART SEVEN · WHAT CAN BE TOUCHED	31
PART EIGHT · LIMITS AND OPEN WORK	33
APPENDICES	36

A note on this text

This is the complete technical description of the piece: the concept, the score, the decision system, the electronics and the light installation.

All the figures have been taken from the files that run in concert — `cicada-seccions.json`, `markov-cicada.js`, `patch-MaxMsp-8-CICADA-KILLER-MARKOV-V2.maxpat`, `CICADA-LEDS.ino` and `leds-visor.js` — or from simulations made with those same files. No number is an estimate.

The final section sets out the limits of the system and the work that remains open.

PART ONE · THE PIECE

1. The concept

The relationship between the cicada and the wasp that preys on it — *Sphecius speciosus*, the **cicada killer** — works in the piece as an environmental metaphor and as a gauge of the relationship we humans establish with the planet. The wasp does not kill the cicada: it paralyses it and stores it alive so that the larva can eat it from the inside. It is a precise image of a cycle interrupted but not finished.

Musically, this tension translates into an axis that runs through the whole piece:

pure and tonic sounds	the life of the cicada, the cycle intact
nodal and frange sounds	predation, friction, collapse

The electronics **do not accompany the instruments: they act as habitat**. They support, threaten or disappear. The instruments act as biological agents within this medium, and the computer system acts as climate, modifying the flow of the piece according to the density and the spectral energy of what is sounding.

The light, finally, is a **population of one hundred and fifty individuals** that synchronises or disintegrates. Synchrony and collapse are the two poles of the same axis that runs through the music. Each performance is therefore an unrepeatable sound laboratory.

This thesis — that the same axis runs through the sound, the formal decision and the light — is what allows the three systems to share a single description. It is the foundation of the whole architecture that follows.

2. The form: open but governed by rules

The piece is divided into six sections, each centred on one of Pierre Schaeffer's morphologies.

	morphology	pages	duration
A	<i>son pur</i> → <i>son tonique</i>	3 + LIVING ECOSYSTEM	1 to 5 min
B	<i>son cannelé</i> (vibrato, tremolo, frullato)	3 + LIVING ECOSYSTEM	1 to 5 min
C	<i>groupe nodal</i> and <i>nœud</i>	3 + LIVING ECOSYSTEM	1 to 5 min
D	<i>son frange</i> (air, friction, distortion)	3 + LIVING ECOSYSTEM	1 to 5 min
E	<i>decrease</i> or bifurcation	1, open	1 to 3 min
F	open resolution, dissolution	1, open	1 to 3 min

The score is **spectromorphological**: the thickness, texture and colour of the line indicate the morphology to be produced, and each performer has to find their own technique to embody it. Tempo and dynamics are relative. The last page of each section, marked **LIVING ECOSYSTEM**, is an open space where the given materials can evolve in any direction.

2.1 What is fixed and what is free

It is worth separating the four levels of decision, because that is where the poetics of the piece are concentrated.

level	who decides	what
Written and closed	the composer	the morphology of each section, the gestures, the registers, the internal order of the pages, the dynamic relationship
Written and open	the performer	the specific technique with which each morphology is produced, the real tempo within a margin, the articulation, the entire content of the LIVING ECOSYSTEM page
Decided live by the machine	the system	the order of the sections, the duration of each one, which electronic banks sound and when, all the behaviour of the light
Not decided by anyone	—	the exact result: the system measures what happens and reacts, but neither the composer nor the performers nor the program know in advance what form the piece will take

The **order of the sections is not written**. The system decides it during performance, subject to nine rules:

1. A, B, C and D sound **exactly once**.
2. E may **reappear** a maximum of twice, but not always: the system decides according to density.
3. **F always closes**.
4. The system **listens for a minimum of one minute** in each section before it can decide anything.
5. Once the minute has passed it can change whenever it wants, or go on listening.
6. If the musicians **turn the last page**, the change is forced.
7. **Hard ceiling**: no section goes beyond its ceiling (300 s for A B C D, 180 s for E and F).
8. **E never twice in a row**: always with another section in between.
9. **Absolute ceiling**: the piece never exceeds 30 minutes.

The rules are not decorative: all nine are implemented and all nine are checked on every run of the simulation battery (\$20).

2.2 The formal consequence

With these rules, the valid paths number exactly **300**, and in simulation the system gets as far as covering 259 of them. The median duration lies between 10 and 19 minutes depending on the way of playing (\$20.2).

What matters is that the path is not a blind draw: **it depends on what the musicians play**. A duo that stagnates provokes changes; a duo that develops the material makes the system wait.

3. cicada-seccions.json: the single description of the piece

This is the most important architectural decision in the whole project, and it is worth setting out before anything else.

The description of each section **does not live in the code**, but in a readable, editable text file, `cicada-seccions.json`, loaded into a Max dict object. For each section it contains:

field	what it is	who reads it
titol, morfologia, atacs, gestos, mm, dinamica, electronica, contact_mic, cicada, living_ecosystem	verbal description of the section	nobody: they are for the composer and the performers
eixos eixos_bifurcacio	the seven axes , from 0 to 1 only in E: the axes of the second branch	the decision chain and the light the light
temps	minim, mitjana, sostre, in seconds	the decision chain
pagines samplers	number of pages in the section which sample banks it authorises (1/0)	the decision chain the electronics engine
leds and leds_bifurcacio	verbal description of the behaviour of the light	nobody: it is the human check on the mapping

Consequence: the musical behaviour of the system can be rewritten without touching a single line of program. But the strong consequence is another one: **the same file describes the music and the light**. There are not two descriptions that can silently diverge. You edit the JSON, press `reload`, and the decision chain, the Arduino and the Max viewer all find out at the same time.

3.1 The seven axes

axis	what it measures	scale
density	number of events per unit of time	0–1
noise	noise component against tuned sound	0–1
attack	hardness and presence of the attacks (0 = <i>attaque nulle</i> , 1 = <i>explosive</i>)	0–1
width	spectral width occupied	0–1
movement	internal movement of the sound (vibrato, tremolo, frullato, undulation)	0–1
tempo	M.M. normalised between 40 and 80	0–1
dynamics	from <i>pp</i> (0) to <i>f</i> (1)	0–1

The values in force:

	density	noise	attack	width	movement	tempo	dynamics
A <i>pure/tonic</i>	0.10	0.05	0.05	0.15	0.10	0.25	0.30
B <i>cannelé</i>	0.40	0.40	0.35	0.45	0.95	0.55	0.55
C <i>nodal/nœud</i>	0.65	0.40	0.85	0.55	0.35	0.55	0.70
D <i>frange</i>	0.95	0.95	0.90	0.95	0.20	0.75	0.80
E <i>decrease</i>	0.25	0.30	0.10	0.40	0.25	0.35	0.25
E <i>bifurcation</i>	0.45	0.50	0.50	0.50	0.70	0.50	0.60
F <i>dissolution</i>	0.15	0.70	0.00	0.30	0.15	0.25	0.10

Two assignments deserve comment because they are counter-intuitive and are annotated in the JSON itself.

Frullato counts as noise, not as internal movement. The high movement of B (0.95) comes from the vibrato and the tremolo; the occasional frullati raise the noise. If frullato had counted as movement, B and D would have come too close in the space of axes and the system would have read them as neighbours, which musically they are not.

The movement of D is low (0.20). The texture of D does not undulate: **it breaks**. The rips and the dense vertical lines are density and noise, not undulation. This decision is what places D so far from B (distance 0.54) even though both are “active” sections.

F has noise 0.70 with density 0.15. It is the only section with this combination: air without sound, breath at the threshold of noise. This is the reason why, in the distance metric, noise weighs more than the rest (\$7.2): without it, A and F — which in density, dynamics and tempo are almost identical — would be read as close sections, when they are the beginning and the end of the piece.

4. The architecture of the patch

The Max 8 patch contains **1,564 objects** distributed across **51 patchers**, with **1,229 connections** and a maximum nesting depth of 5 levels. It takes up 1.2 MB.

The root contains only the interface. The whole engine lives inside p code.

subpatcher	objects	role
p in-audio	15	four channels — DPA and contact mic for each instrument — summed two by two into hrn-in and tmb-in
p audio-analysis	184	descriptor extraction: the global chain and the per-instrument chain
p analisi-electronica	62	the same descriptor chain on the output of the electronics
p MARKOV-cicada	145	the decision engine: js markov-cicada.js, dict cicada-seccions, stopwatch, rules
p score	85	projection of the score, page counters, seven films
p audio-engine	667	container: twelve samplers + real-time processing
p out-master	28	quadraphonic output matrix, two reverbs, dac~ 1 2 3 4
p lights	133	serial port to the Arduino and viewer of the light inside Max
p provador-llum	81	test bench for the light without hardware
p controller	4	the b key as a page pedal

And in the root patcher, p samplers-tapes-markov-densitat (36 objects): the monitoring panel for the five banks, with an LED that blinks on every trigger and a multislider with the presence of each bank.

4.1 The buses

The patch uses **71 different bus names** (send/receive, send~/receive~). The ones that matter for understanding the flow:

```

adc~ 1 2 3 4 → hrn-in, tmb-in          (input signal)
hrn-in, tmb-in → an-hrn, an-tmb        (12 descriptors each)
tapeD-MASTER + tapeE-MASTER → an-ele  (12 descriptors)
an-hrn, an-tmb, an-ele → js markov-cicada.js
js → ordre-num-de-seccio → p score
js → messnamed → leds-seccio, leds-eixos, leds-hrn, leds-tmb, leds-ele,
                      leds-pressio, leds-deriva
js → messnamed → vol-banc-1..5, markov-bancs-pres, and the five bank names
p score → pagina-actual → js          (closed loop of the page)
canvi-pagina → js, p score, interface (pedal)
llum-realitat, llum-presentir, llum-habitat, llum-viatge → viewer and Arduino at once

```

An implementation note with practical consequences: **the seven light buses, the five vol-banc-N and markov-bancs-pres have no send object anywhere in the patch.** There are only the receive objects. They are published by the .js with messnamed(), which is invisible to any static analysis of the file. Anyone who opens the patch and looks for where leds-hrn comes from will not find its origin anywhere; that is why there is an explicit comment inside p lights that says so.

PART TWO · HOW THE SYSTEM LISTENS

5. Two generations of analysis living side by side

The patch contains **two different analysis chains running at the same time**, and this is a deliberate decision, not an oversight.

The global chain: a single sigmund~ @hop 4096 pitch env and a single bonk~ — the only calibrated one, with learn 10, mask 4 0.7 and minvel 10 fired by the loadbang, and a thresh 6 50 that has to be pressed by hand — fed **simultaneously** by receive~ hrn-in and receive~ tmb-in. Out of it come pitch, envelope, attack and the three buses baixa-, mitja- and alta-activitat-events that fire the samplers.

The per-instrument chain: three parallel blocks — horn, trombone and electronics — each with sigmund~ @hop 2048 pitch env, an svf~ 2500 0.6, seven octave reson~ and a bonk~. Out of them come an-hrn, an-tmb and an-ele, twelve descriptors each, read only by the js.

Why they coexist: the sampler triggering is tuned to the hall and it works. Replacing it would mean recalibrating the whole behaviour of the electronics. The per-instrument chain was added **without touching anything** in the other one — the comment in the patch says so literally: *“Touches nothing in the analysis above, which keeps feeding p samplers-electro.”* The cost is that the patch has two sigmund~ measuring the same signal with different windows. The benefit is that a change in the decision engine cannot break the electronics.

5.1 The signal chain, object by object

Each block of the per-instrument chain is exactly this:

```
receive~ <source>
|
|— sigmund~ @hop 2048 pitch env
|   |
|   |— out0 → prepend <x>pitch → s an-<x>
|   |   |
|   |   |— out1 → prepend <x>env → s an-<x>
|   |
|— average~ 2205 rms → snapshot~ 100 → prepend <x>tot → s an-<x>
|— svf~ 2500 0.6 → average~ 2205 rms → snapshot~ 100 → prepend <x>hf → s an-<x>
|— reson~ 1 100 2 → average~ 2205 rms → snapshot~ 100 → prepend <x>band 0 → s an-<x>
|— reson~ 1 200 2 → ... → prepend <x>band 1 → s an-<x>
|— reson~ 1 400 2 → ... → prepend <x>band 2 → s an-<x>
|— reson~ 1 800 2 → ... → prepend <x>band 3 → s an-<x>
|— reson~ 1 1600 2 → ... → prepend <x>band 4 → s an-<x>
|— reson~ 1 3200 2 → ... → prepend <x>band 5 → s an-<x>
|— reson~ 1 6400 2 → ... → prepend <x>band 6 → s an-<x>
|— bonk~ → t b → message <x>atk → s an-<x>
```

Twelve messages per block, all towards the same bus, which arrives at the inlet of the js. The prepend turns the number into a call to the function of the same name: hrnenv 62.4 invokes hrnenv(62.4) inside the JavaScript. It is the standard mechanism of Max's js object, and it is what allows the whole analysis chain to be poured through a single cable.

The three blocks are not literally identical on one point: the bonk~ of the horn and that of the trombone come out of outlet 0 and that of the electronics out of outlet 1. Both give the same thing for the use made of them — one impulse at each attack — but it is worth knowing if one day the band information has to be put to use.

The remaining constants are indeed identical in all three blocks: **RMS window of 2205 samples** (50 ms at 44.1 kHz), **sampling every 100 ms** (10 Hz), and a bank of seven octave filters at **100, 200, 400, 800, 1600, 3200 and 6400 Hz** with gain 1 and Q = 2.

5.2 Where the electronics are tapped

The third chain is **not** taken from the general output. It is taken from receive~ taped-MASTER + receive~ tapeE-MASTER summed with a +~.

This is the point where the twelve samplers are already mixed and processed but the instruments have not yet been added to them. Had the master been taken, the third analysis would have measured the horn and the trombone a second time, and the habitat layer of the light would have repeated what the two colonies already say. With this tapping point, **the three layers are genuinely independent**.

6. The five descriptors

Five descriptors are extracted from each source, all normalised to 0–1, and all with the same names as the axes of the JSON. This coincidence is not cosmetic: it is what allows what is sounding to be compared with what is written.

6.1 Dynamics

```
s.dyn = ema(s.dyn, clamp((s.envS - ENV_LO) / (ENV_HI - ENV_LO), 0, 1), LIVE_ALPHA);
```

The envelope of sigmund~ in dB, scaled between ENV_LO = 20 and ENV_HI = 90, and smoothed with an exponential moving average of coefficient LIVE_ALPHA = 0.05. **These two numbers are the first thing that has to be calibrated in the hall** (message calibraenv <lo> <hi>).

6.2 Attack

```
var rise = v - s.envPrev;
if (rise > s.riseMax) s.riseMax = rise;
...
s.atac = ema(s.atac, clamp(s.riseMax / ATTACK_REF, 0, 1), 0.35);
s.riseMax = 0;
```

The largest envelope rise of the last second, not that of each sample. ATTACK_REF = 9 dB per frame is what counts as a hard attack. The maximum is reset every second, so that the descriptor measures *the strongest attack there has been*, not the average — which in a texture with sporadic attacks would be almost zero.

6.3 Density

```
s.ioi = ema(s.ioi, Math.max(30, now - s.lastAtk), 0.3); // at each attack
s.dens = ema(s.dens, clamp(1 - (s.ioi - 300) / 4700, 0, 1), 0.15);
if (elapsed*1000 - s.lastAtk > 6000) s.dens = ema(s.dens, 0, 0.15);
```

Density is **not** a count of attacks per window: it is the **interval between attacks** (IOI, *inter-onset interval*) converted into an inverse scale. An IOI of 300 ms or less gives density 1; an IOI of 5,000 ms gives density 0. The third line is the decay mechanism: if more than six seconds have passed without any attack, the density drops towards zero on its own, without waiting for a new attack to arrive and update the IOI.

The choice of the IOI instead of the count has a concrete musical consequence: **a fast tremolo and a dense rip give a similar density even though the absolute number of events is very different**, because what is measured is the separation between them.

6.4 Internal movement

```
s.mov = ema(s.mov, clamp(Math.max(s.pflip, s.eflip) / MOD_REF, 0, 1), 0.3);
```

pflip and eflip are **changes of direction** of the pitch and of the envelope during the last second, and MOD_REF = 9 per second is full movement. What makes this work is the **hysteresis**:

```
if (s.envS - s.envRef > ENV_HYST) { if (s.edir !== 1) { s.eflip++; s.edir = 1; } s.envRef = s.envS;
else if (s.envRef - s.envS > ENV_HYST) { if (s.edir !== -1) { s.eflip++; s.edir = -1; } s.envRef = s.envS;
```

A turn is only counted if the value has moved more than $\text{ENV_HYST} = 1.5 \text{ dB}$ (or $\text{PITCH_HYST} = 0.006$, 0.6% of pitch) since the last turning point. Without this, the detector's natural jitter produces a change of direction at every frame and **movement always saturates at 1**.

6.5 Noise: spectral flatness

This is the most important descriptor of the whole system, because it is the one that separates the cicada from its predator.

```
for (k = 0; k < 7; k++) { m = band[k] + 1e-7; sum += m; lsum += Math.log(m); }
var geo = Math.exp(lsum / 7), ari = sum / 7;
s.flat = ema(s.flat, clamp(geo / ari, 0, 1), 0.2);
s.noise = ema(s.noise, clamp((s.flat - FLAT_LO) / (FLAT_HI - FLAT_LO), 0, 1), LIVE_ALPHA);
```

It is the **Wiener entropy**: the geometric mean of the seven bands divided by the arithmetic one. It measures whether the energy is **concentrated or spread out**, not where it is nor how much there is.

- A **tonique** sound concentrates the energy in few bands → the geometric mean falls far below the arithmetic one → **low flatness**.
- A **frange** sound spreads it across all of them → the two means come closer → **high flatness**.

Three examples calculated with the actual formula:

seven bands	flatness	noise
clean note, energy in two bands	0.065	0.00
note with frullato	0.449	0.11
air and friction, energy spread out	0.990	1.00

The thresholds are $\text{FLAT_LO} = 0.40$ (clean tonique sound) and $\text{FLAT_HI} = 0.85$ (air or friction), adjustable live with calibra <lo> <hi>.

Why this and not the high-frequency ratio. The obvious measure would be the energy above 2500 Hz divided by the total energy — in fact the patch calculates it anyway and the code keeps it in reserve (useFlat 0 goes back to it). The problem is that **it depends on the register**: a clean, high note would give a high high-frequency ratio and the system would read it as noise. Spectral flatness does not have this defect: a clean, high note concentrates the energy in a few bands, wherever it falls, and therefore continues to read as tonique. For a piece that uses the whole register and where the tonique/frange distinction is its conceptual axis, the difference is not a minor one.

An implementation detail that avoids a common error:

```
if (act === 0) return; // silence: touch nothing
```

In silence the seven bands are worth almost zero and their flatness is, arithmetically, high — silence would look like white noise. The guard requires that at least one band exceed 0.0002 before touching the descriptor. In silence, the noise **stays as it was** instead of jumping to 1.

6.6 The live vector

Once per second it is consolidated:

```
LIVE[0] = Math.max(IN[0].dens, IN[1].dens); // density
LIVE[1] = Math.max(IN[0].noise, IN[1].noise); // noise
LIVE[2] = Math.max(IN[0].atac, IN[1].atac); // attack
LIVE[4] = Math.max(IN[0].mov, IN[1].mov); // movement
LIVE[6] = Math.max(IN[0].dyn, IN[1].dyn); // dynamics
```

The more active of the two instruments rules, not the average. The reason is musical: if the horn plays a rip and the trombone sustains a note, the ecosystem *has* a rip. Averaging them would have diluted it.

The ample and tempo axes **are not measured** — they have no reliable descriptor in real time — and their weight in the comparison is 0:

```
var LIVE_W = [1.0, 1.2, 1.0, 0.0, 0.9, 0.0, 0.6];
```

The third channel, the electronics, **is consolidated but does not enter LIVE[]**. The decision is explicit in the code: *“The third does NOT enter the LIVE vector: it must not change any Markov decision, it only feeds the third layer of the light.”* If the electronics influenced the decisions, the system would be reacting to its own sound and would feed back on itself.

PART THREE · HOW IT DECIDES

7. Two independent decisions

They must be kept apart, because they work by completely different mechanisms.

7.1 WHEN to change: it is not a Markov chain

It is a **draw every second**, like a radioactive decay:

```
if (pressure > 0 && Math.random() < pressure) go(chooseNext(), "markov");
```

The pressure probability is recalculated at every tick:

```
if (t < MIN_DWELL[section]) { pressure = 0; return; } // the listening minute
var p = (1.0 / MEAN_EXTRA[section]) * leaveMultiplier(section, density);
var f = clamp(flux / FLUX_REF, 0, 1);
discourse = (1 + STAG_BOOST*(1-f)) * (1 - DEV_BRAKE*f);
p = p * discourse;
if (TARGET_MIN > 0) p = p * (0.45 + 1.55 * (elapsed / (TARGET_MIN*60.0)));
pressure = clamp(p, 0, 1);
```

Four factors, in order:

1. The time base. $1/\text{MEAN_EXTRA}[\text{section}]$, with the values from the JSON: A 200 s, B 190, C 190, D 170, E 100, F 110. A probability of 1/200 per second gives an expected mean duration of 200 s **after** the minimum minute.

2. The exit multiplier, which is different in each section and is where the dramaturgy is inscribed:

```
if (s == 1) return 1.0 + 2.0*d; // A: the calm breaks if they rise
if (s == 2) return 1.0 + 1.0*d; // B: gentle bifurcation
if (s == 3) return 0.5 + 1.5*d; // C: swell or prolong
if (s == 4) return 1.0 + 2.0*Math.abs(d - 0.5); // D: collapse or saturation
if (s == 5) return 1.0 + 1.0*d; // E: resolve or micro-climax
if (s == 6) return clamp(1.3 - d, 0.1, 1.3); // F: dissolves when all falls silent
```

They are worth reading for what they are: **six different ways a section has of wanting to end.**

- **A** has a calm that density breaks: the more they play, the sooner it leaves.
- **C** is the inverse at low density (0.5): if the musicians do not swell the nodal, **C is prolonged**, it does not end.
- **D** is the interesting case: $|d - 0.5|$. D wants to leave **whether it collapses or saturates**, and what keeps it is staying in the middle. It is the literal translation of “critical point”.
- **F** is the only inverted one: the more everything falls silent, the more likely it is to dissolve. F ends when nothing is left, not when something happens.

3. The discourse. See §7.4: the factor that looks at whether the material is advancing or has got stuck.

4. The total-duration governor. With $\text{TARGET_MIN} = 15$, the factor goes from 0.45 at the start to above 1 as the target approaches. It is not a limit: it is a growing tilt.

On top of these four there are the hard rules: the minimum (which sets the pressure to zero), the ceiling, sustained saturation/desaturation, the pedal, and the piece’s absolute ceiling.

7.2 WHICH section comes next: the probabilistic core

Here there really is a transition matrix, and it is multiplied by three further filters.

```
base = W[section][to]; // 1. base matrix
if (FLAT > 0) base = base*(1-FLAT) + 0.25*FLAT; // flattened

dd = dist(section, to) - targetDist; // 2. distance
```

```

cont = Math.exp(-(dd*dd) / (2*SIGMA_D*SIGMA_D));

pd = SEC[to][0] - projected; // 3. direction
dir = Math.exp(-(pd*pd) / (2*SIGMA*SIGMA));

val = base * (1 - ADAPT + ADAPT * cont * dir);

if (ESCOLTA !== 0) { // 4. recognition
  dl = distLive(to);
  rs = Math.exp(-(dl*dl) / (2*SIGMA_L*SIGMA_L));
  if (ESCOLTA < 0) rs = 1 - rs;
  val = val * (1 - RES_W + RES_W*rs);
}

```

On top of this product, the **mask of what is left to play** (`availableTargets()`) removes at the outset the sections already played and, where applicable, E.

Filter 1 · The base matrix

	→A	→B	→C	→D	→E
from A	0.00	0.55	0.25	0.10	0.10
from B	0.30	0.00	0.45	0.15	0.10
from C	0.05	0.15	0.00	0.50	0.30
from D	0.10	0.15	0.30	0.00	0.45
from E	0.10	0.15	0.30	0.30	0.00

It comes directly from the verbal description of the piece: A tends to B, B to C, C to D, D to E. It is the written grammar, and it is the only thing in the whole system that does not come out of any calculation. F does not appear in it: rule 3 imposes it.

The flattening (FLAT = 0.6 by default) mixes it with the uniform distribution. It is the parameter that decides **how much authority the written grammar has** over the system, and it can be changed live (§21.1). The exact figures are in §20.5.

Filter 2 · Distance, or continuity

```

function dist(x, y) {
  for (i = 0; i < 7; i++) { d = SEC[x][i] - SEC[y][i]; s += AXIS_W[i]*d*d; wsum += AXIS_W[i]; }
  return Math.sqrt(s / wsum);
}
var AXIS_W = [1.0, 1.2, 1.0, 0.7, 0.9, 0.5, 0.6];

```

A weighted Euclidean distance in the space of the seven axes. **Noise weighs 1.2**, more than any other axis, for the reason set out in §3.1: it is what separates A from F.

The resulting distance matrix, computed with the JSON in force:

	A	B	C	D	E	F
A	—	0.44	0.49	0.73	0.17	0.31
B	0.44	—	0.33	0.54	0.32	0.42
C	0.49	0.33	—	0.32	0.39	0.49
D	0.73	0.54	0.32	—	0.60	0.62
E	0.17	0.32	0.39	0.60	—	0.20
F	0.31	0.42	0.49	0.62	0.20	—

It reads like this: **0 = identical, 1 = opposite**. A and D are the extremes of the piece (0.73). A and E are almost the same thing (0.17) — which makes sense: E *decrease* is the return to the baseline state. E and F too (0.20).

The filter does not look for the minimum distance but for a **target distance**, with a Gaussian bell of width `SIGMA_D = 0.28` around it:

mode	target distance	what it means
transformation	0.22	go to a neighbouring section, the texture evolves
bifurcation	0.70	jump far, rupture

The mode is chosen by `pickMode()` at each change, with per-section probabilities:

```
var P_BIFURCA = [0, 0.15, 0.30, 0.20, 0.60, 0.45, 0.0];
if (Math.abs(trend) > 0.35) p += 0.20;
```

D bifurcates 60 % of the time — “*increases probabilities of sudden bifurcations*”; B 30 % — “*introduces small bifurcations*”. And a strong density trend in either direction adds a further 20 %.

Filter 3 · The direction of the ecosystem

```
var projected = clamp(density + trend*0.5, 0, 1);
if (satTimer >= SAT_TIME) projected = clamp(projected - 0.55, 0, 1);
if (desatTimer >= DESAT_TIME) projected = clamp(projected + 0.55, 0, 1);
```

The comparison is not with the current density but with the **projected density**: the present one plus half a second of trend. The system looks at where the ecosystem is going, not where it is.

The two corrections are the anti-saturation mechanism. If density has stayed above 0.82 for 110 seconds straight, the projection is **flipped**: the system stops looking for dense sections and starts looking for empty ones. The logic is simple — after two minutes of saturation, what comes next can no longer be more saturation.

Filter 4 · Recognition This is where the system compares **what is sounding now** with each candidate section, using the live vector from §6.6 and the same distance metric but with the `LIVE_W` weights (which cancel amplitude and tempo).

```
escolta 1  ATTRACT: tends towards the section that resembles what is sounding
escolta 0   disabled: only measures
escolta -1  AVOID: tends towards the most different section
```

With `RES_W = 0.7`, recognition weighs 70 % of the factor.

The three modes are three different aesthetics of the same material. **ATTRACT** is a system that confirms the musicians: if they start doing nodal things, C becomes more likely. **AVOID** is a system that contradicts them. Both are worth trying in rehearsal; they produce different pieces from the same score.

7.3 Why it is not a Markov chain

The piece uses the name “Markov” and that is what is written in the patch, but it is worth being precise.

A pure Markov chain satisfies the **Markov property**: the probability of the next state depends *only* on the current state. That is not the case here. The choice depends on:

- **the current state** (the row of the `W` matrix) — this part is indeed Markovian
- **the complete history** (`played[]`, `countE`) — which sections have already been played
- **an external continuous variable** (density, trend, drift)
- **the elapsed time** (`satTimer`, `desatTimer`, the duration governor)
- **the mode** (transformation or bifurcation), which is chosen separately

Strictly speaking, then, it is a **state machine with probabilistic transitions conditioned by context**. The Markov matrix is one component of it — the first of the four filters — and, with the default flattening of 0.6, not even the most determining one.

This precision is not pedantry: it matters because it explains why the system cannot be analysed with the usual tools of Markov chains (stationary distribution, return times), and why verification has to be by simulation.

7.4 The drift: the idea that makes the system follow the discourse

This is the mechanism I am happiest with, and the one most easily overlooked.

The problem: a system that looks at the **level** of density cannot tell an agitated passage that goes nowhere from a passage that is genuinely growing. Both give high density. But musically they are opposite things: the first asks for something to happen, the second asks to be allowed to finish.

The solution is to look not at the level but at the **drift**: the distance between a fast and a slow average of density.

```
density = density * (1 - 0.06) + target * 0.06;      // fast average
densSlow = densSlow * (1 - 0.006) + density * 0.006; // slow average, ~3 min of memory
flux     = Math.abs(density - densSlow);              // the DRIFT
```

And the factor that comes out of it:

```
var f = clamp(flux / FLUX_REF, 0, 1); // FLUX_REF = 0.09
discourse = (1 + STAG_BOOST*(1-f)) * (1 - DEV_BRAKE*f); // 1.2 and 0.55
```

drift	discourse	effect
0.000	2.20	material stuck: the probability of change doubles
0.020	1.70	
0.045	1.16	
0.070	0.72	
0.090 and above	0.45	active discourse: the probability of change falls to less than half

The full range is from 2.20 to 0.45 — a factor of **almost five to one**. It is by far the strongest modulator in the whole pressure calculation.

What this formula does, put in rehearsal language: **if you have got stuck, I push; if you are saying something, I let you finish**. A frantic tremolo that does not evolve has almost zero drift and the system reads it as stagnation, even if the density is high. A long *decrescendo* towards nothing has high drift and the system respects it, even if the density is low.

The drift also has a sign, and the sign is useful too. The .js publishes to the light not the absolute drift but the signed one, on a scale where 50 means *still material*:

```
function derivaSignada() {
  var d = clamp((density - densSlow) / FLUX_REF, -1, 1);
  return Math.round(50 + d * 50);
}
```

Above 50 the discourse grows; below, it decreases. The light installation uses this to decide whether the pattern travels along the strip and **which way** (§14).

The figures verified with the simulation profiles are in §20.6.

PART FOUR · THE ELECTRONICS

8. Five banks, and one that does not sound continuously

Five banks of samples, distributed across twelve sfplay~ players:

bank	content	players
1	real cicada (field recordings, for low/medium/high activity)	3
2	instrumental imitations of the cicada (at three distances)	3
3	imitations with contact microphone	1
4 + 6	morphologies and structure by section	2
5	electronic processing (low, medium, high)	3

Which banks may sound is stated by the section, in the `sampler`s field of the JSON. For example, A authorises banks 1 and 4+6 and forbids the rest; D authorises all but the real-cicada one — in the collapse section, the real cicada is no longer there.

But an authorised bank **does not sound continuously**. It makes phrases.

8.1 The four-phase phrases

rest → entry → song → exit → rest

Each phase has a duration drawn at random within a range, and each phrase has its own peak volume:

phase	duration (s)
rest	10 to 45
entry	4 to 12
song	15 to 55
exit	5 to 16
peak volume	0.45 to 1.00

The curves are linear: $vol = pic \cdot (t/dur)$ on entry, $vol = pic \cdot (1 - t/dur)$ on exit. The result is that each bank **appears and disappears making shapes** instead of either being there or not.

When a section stops authorising a bank, the bank is not cut: it moves into the exit phase and fades away over 5 to 16 seconds.

8.2 The dialogue with the instruments

The volume of each phrase is modulated according to what the musicians play:

```
function factorDialeq() {  
  if (DIALEG > 0) return 1.0 - 0.72*density; // COMPLEMENTARY  
  if (DIALEG < 0) return 0.28 + 0.72*density; // IMITATIVE  
  return 1.0; // NEUTRAL  
}  
var v = clamp(ph.vol * (1 - DIA_W + DIA_W*dia), 0, 1); // DIA_W = 0.75
```

And so is the probability of starting a phrase:

```
function ganesDEntrar() {
  var base = 0.05;
  if (DIALEG > 0) return base * (0.4 + 2.2*(1 - density)); // enters into the gap
  if (DIALEG < 0) return base * (0.4 + 2.2*density);
  return base;
}
```

In **complementary mode** — the default — the electronics grow when the musicians leave space and withdraw when they fill it, and they are **more eager to enter when the players fall silent**: the probability per second goes from 0.13 in silence to 0.02 at full density, a factor of six and a half. Moreover, if the instruments become very dense mid-song, the phrase is cut short:

```
if (DIALEG > 0 && density > 0.85 && Math.random() < 0.04) ph.t = ph.dur;
```

In **imitative mode** all of this is inverted: the electronics grow with them.

8.3 The division of responsibilities

It is worth summarising, because it is what makes the electronics seem alive:

decision	who makes it
which banks may sound	the section (the JSON)
when and at what volume a phrase enters	the dialogue with the instruments
how often the samples fire within a bank	the instrumental density, via baixa-/mitja-/alta-activitat-events

Three levels with three different clocks: the section changes every few minutes, the phrases every tens of seconds, the triggers every second. The superimposition of the three is what keeps the electronics from sounding either cyclical or random.

PART FIVE · THE LIGHT

9. One hundred and fifty individuals

The light installation is a **complete strip of 150 WS2812B LEDs** —a five-metre roll at 30 LED/m, uncut— governed by an **Arduino Uno or Nano**. It is not stage lighting, nor an illustration of the score: it is a **population of one hundred and fifty coupled oscillators** — the Kuramoto model in eight-bit arithmetic.

Its origin is an earlier sketch, CODI-SincroLeds: four LEDs each blinking at its own rate, where every press of a push-button made them converge toward the mean until at press 20 everything was equal. This is exactly what cicadas and fireflies do. And within this piece it has an immediate reading:

synchronised population = cycle intact desynchronised population = cycle broken

The installation scales that mechanism to one hundred and fifty individuals and adds what was not in the sketch: **that the coupling can also loosen**. The sketch only went from disorder to order and did not come back. The piece does come back: D is the collapse, and then E and F dissolve.

9.1 Three layers, the three agents of the piece

layer	what it is	how it is drawn
Habitat	the electronics	occupies the whole strip, beneath everything, at half the frequency of the colony. A single wave travels the hundred pixels
Horn colony	the horn analysis	centre of gravity at LED 25
Trombone colony	the trombone analysis	centre of gravity at LED 75

The territory of each colony is defined by a trapezoidal ramp:

```
uint16_t x = (i * 255) / (NUM_LEDS - 1);
if (x < 80) return 255;           // pure horn
if (x > 175) return 0;           // pure trombone
return 255 - ((x - 80) * 255) / 95; // contact zone
```

With one hundred and fifty LEDs this gives: **LEDs 0–47 pure horn, LEDs 103–149 pure trombone, LEDs 48–102 contact zone** with continuous blending. The midpoint falls exactly between LEDs 74 and 75. Over a five-metre span: 1.60 m of horn, 1.83 m of contact and 1.57 m of trombone.

In the contact zone all the light parameters are interpolated — hue, saturation, brightness, undulation, extent — **except the waveform, which switches**. A breath wave and a strobe wave cannot be blended; what can be done is to decide where the boundary lies.

And there is one more thing in the contact zone, which is what keeps it from reading as a single population:

```
if (p > 40 && p < 215) {
    uint8_t prop = 128 - abs(p - 128);
    f += prop <= 6;           // up to 1/8 of a cycle
}
```

The individuals in the middle run **out of phase by up to 45°** with respect to the two extremes, with a continuous gradient affecting LEDs 38 to 62. The two musicians do not touch: they interfere.

9.2 The section draws nothing

This is the conceptual point. **The section has no assigned colour**. What it does is set the laws: what tendency to synchronise there is, what hue, what speed, how much extent. The individuals obey them each in its own way.

10. A single description, not two

This is the same decision as with the durations, and it is what keeps the whole system coherent.

The light parameters **do not come from any hand-made table**. They come from the **seven axes** of cicada-seccions.json — the same ones the decision chain uses to compute distances — passed through a single function, `deEixos()`.

10.1 The mapping, formula by formula

Coupling – noise. Noise is what breaks the cycle: son tonique = a population singing together; son frange = collapse.

```
uint8_t q = 255 - e.sor;  
L.acobl = scale8(q, q)/2 + q/2;
```

The mean of a quadratic curve and a linear one: monotonic but not linear, so that low noise is penalised less than high noise. Noise 0 → coupling 254. Noise 128 → 94. Noise 242 (D) → 6.

Hue – attack, with noise as a corrector.

```
if (e.atac < 100) {  
    L.to = 165 - (e.atac * 30) / 100;           // 165 → 136, blue → cyan  
} else {  
    calor = ((e.atac - 100) * 255) / 155;  
    t      = 38 - scale8(12, calor);             // 38 → 27, orange  
    crema = (e.sor > 128) ? ((e.sor - 128) * 255) / 127 : 0;  
    L.to = scale8(t, 255 - scale8(217, crema));  // toward red  
}
```

Two regimes with a hard boundary at `atac = 100`. Below it, blues and cyans. Above it, oranges, which noise can push as far as red (hue 4 at maximum noise). **The jump is discontinuous by design: there are no greens.** The piece has no intermediate zone between song and friction, and neither does the light.

Waveform – attack + noise.

```
if (e.atac >= 179 && e.sor >= 179) ONA_ESTROBO;  
else if (e.atac >= 92)             ONA_PULS;  
else                               ONA_ALE;
```

The strobe requires **both things at once**: hard attack *and* noise. It is the critical point, and only D reaches it. The pulse threshold (92) is lower than that of the hue change (100), so there is a narrow band — attack between 92 and 99 — that **is still blue but already pulsing**. It is the instant in which the population becomes nervous before changing colour.

The other five:

```
L.sat  = 30 + scale8(215, e.din);           // dynamics  
L.brill = 35 + scale8(220, scale8(89, e.din) + scale8(166, e.dens)); // 35% din + 65% dens  
L.vel  = 4  + scale8(60, scale8(e.tempo, e.tempo)); // the SQUARE of the tempo  
L.jit  = e.mov;                             // internal undulation  
L.ext  = 30 + scale8(225, e.ample);         // territorial extent
```

Two details of integer arithmetic worth documenting. The weights 89 and 166 of the brightness are not arbitrary: $\text{scale8}(89, 255) + \text{scale8}(166, 255) = 88 + 165 = 253 < 256$, so that **the sum can never overflow**. And the speed uses the *square* of the tempo, which compresses the low tempi: A and F (tempo 0.25) stay at speed 7, while D (tempo 0.75) reaches 37 — tripling the tempo multiplies the speed by more than five.

10.2 The check: six out of six

What validates the mapping is not that it seems reasonable, but that it **reproduces the six verbal descriptions written in the leds field of the JSON**, without their having been programmed anywhere.

	what the mapping gives	what is written in the JSON
A	hue 162 blue, sat 94, bright 71, breath, coupling 235	<i>white or very soft blue, slow pulse, basal state</i>
B	hue 139 cyan, sat 147, bright 132, breath, undulation 242	<i>blues with soft waves, breathing effect</i>
C	hue 28 orange, sat 180, bright 180, pulse	<i>orange and soft red, faster pulses</i>
D	hue 7 red, sat 201, bright 229, strobe , coupling 6	<i>fast patterns, random or strobe. Critical point</i>
E	hue 158 blue, sat 83, bright 89, breath	<i>they slow down or blink softly, decompression</i>
F	hue 165 blue, sat 51, bright 63, breath	<i>very low light, ash white and bluish grey</i>
E bif.	hue 35 orange, sat 158, bright 144, pulse	<i>tension that does not resolve</i>

Six out of six. The only part the mapping does **not** deduce is the split into two halves of E-bifurcation: that one is a gesture written expressly into the firmware (§16), not a consequence of the axes. The axes of the bifurcation give the colour and the agitation; the split is separate.

That the mapping reproduces the six descriptions means that **you edit the JSON and the musical decisions and the behaviour of the light change at the same time**. A single description of the piece, not two that can diverge in silence.

The complete table of derived parameters:

section	coupl	hue	sat	bright	speed	wave	undul.	ext	radius	period
A	235	162 blue	94	71	7	breath	26	63	±23 LEDs	31 s
B	121	139 cyan	147	132	21	breath	242	131	±49	10 s
C	121	28 orange	180	180	21	pulse	89	153	±57	10 s
D	6	7 red	201	229	37	strobe	51	242	±90	6 s
E	150	158 blue	83	89	11	breath	64	119	±44	20 s
F	49	165 blue	51	63	7	breath	38	97	±36	31 s
E bif.	94	35 orange	158	144	19	pulse	179	142	±53	12 s

11. Between what is written and what is played

The same seven axes describe the section (what is written) and what is sounding right now (the live vector that the .js computes every second). **The same function serves for both.** The light can therefore be placed at any point in between, and where it is placed is decided by a single parameter.

```
llum_hrn = deEixos(barreja(sec_act, h, REALITAT));
llum_tmb = deEixos(barreja(sec_act, t, REALITAT));
llum_hab = deEixos(barreja(sec_act, e, REALITAT));
```

reality 0 the light is the score. It ignores the musicians.

reality 191 what sounds is in charge; the score anchors. ← defaultrealitat 255 the light is what sounds.

The default value answers to an explicit aesthetic decision: **the light must say what the musicians are doing**, not where we are inside the form. When they play what the section says, the two descriptions coincide and nothing shows. When they depart from it, **the light follows them and shows the separation**.

Two important details of how it is implemented:

The mix is made in the space of the axes, not in that of the light parameters. It interpolates first and converts afterwards. This means that with an intermediate reality the result can fall on the other side of a threshold — the 100 of the hue, the 92 of the pulse — and change regime all at once. It is not an artefact: it is what gives the light **tipping moments** instead of drifting continuously.

ample and tempo are always imposed from the section, before the mix. They are not measured live (§6.6), and therefore the territorial extension and the speed of the cycle are always fixed by the score, whatever *realitat* says.

Tested on the compiled firmware, section A, *realitat* 191 — the default value, the one the R 75 command sends —, with the trombone always playing what the section says and the horn doing four different things. The horn's input vectors, in units from 0 to 255 and in the order *density · noise · attack · movement · dynamics*:

what the horn does	live vector of the horn
plays what A says	$26 \cdot 13 \cdot 13 \cdot 26 \cdot 77$
fills with noise	$60 \cdot \mathbf{230} \cdot 40 \cdot 26 \cdot 120$
attacks hard	$200 \cdot 100 \cdot \mathbf{240} \cdot 26 \cdot 220$
goes silent	$0 \cdot 0 \cdot 0 \cdot 0 \cdot 0$

And the result, after forty seconds in each case:

what happens	horn	trombone
they play what A says	coupl 247, hue 164, bright 64	coupl 247, hue 164, bright 64
the horn fills with noise	coupl 59 , hue 158, bright 87	coupl 247, hue 164, bright 64
the horn attacks hard	coupl 139, hue 32 , bright 168	coupl 247, hue 164, bright 64
both go silent	coupl 236, hue 162, bright 50	coupl 247, hue 164, bright 64

When the horn fills with noise, **its stretch of the strip desynchronizes while the trombone's stays together** — its coupling falls from 247 to 59. When it attacks hard, its territory turns from blue to orange and lights up. The trombone has not moved. This is what it means to translate into the visual what is happening in sound: not to illustrate the section, but **to show who is doing what**.

12. The habitat sustains or threatens

The electronics are not limited to putting colour on the strip. They act on the cohesion of the population.

```
int16_t efecteHabitat() {
    if (!HABITAT) return 0;
    uint16_t pres = scale8(140, suau_ele.din) + scale8(115, suau_ele.dens);
    if (pres > 255) pres = 255;
    if (pres < 12) return 0;                                // habitat absent
}
```

```

int16_t car = 128 - (int16_t)suauele.sor;    // +128 tonique ... -127 frange
int16_t d   = (car * pres) / 255;
return (d >= 0) ? (d / 4) : ((d * 3) / 4);    // sustain up to +32, threat down to -95
}

```

Three decisions are inscribed in it:

The character is given by the noise of the electronics. A tonique habitat — fine grains, sons purs — **sustains**: it reinforces the coupling and helps the population stay together. A frange habitat — modulated noise, chaotic microdelays — **threatens**: it loosens it.

The effect is proportional to the presence. If the electronics do not sound, they do nothing: the colony stays exactly as it was. There is no phantom action when the medium is empty.

The threat weighs three times more than the sustain. Down to -95 against +32. It is a deliberate asymmetry: breaking a cycle is easier than building it, and the piece is about precisely that.

Measured on the firmware, in section B (base coupling 121):

the electronics	effect	coherence
not sounding	0	0.990
tonique and quiet	+6	0.991
tonique and present	+19	0.994
frange and present	-45	0.957
frange and saturated	-77	0.857

The effect column is exact and deterministic. The coherence column moves by a few thousandths between runs, because with low coupling the biological noise of each individual counts: what is stable is the order of the rows, not the third decimal.

It is disabled with the C 0 command.

13. The ecosystem senses the change

The .js computes a **pressure to change** between 0 and 1 every second (§7.1): how close the system is to changing section. It acts on the colony in **three ways at once**:

```

uint8_t inquietud = PRESENTIR ? pressio : 0;

// 1. loosens the bond
if (inquietud) acobl = scale8(acobl, 255 - scale8(215, inquietud));

// 2. imposes a floor of biological noise
uint8_t agitacio = 255 - acobl;
if (agitacio < inquietud) agitacio = inquietud;

// 3. widens the individual natural speeds
seu += (seu * (hash8(i) - 128) * (60 + (inquietud >> 1))) / 16320;

```

With maximum pressure, the coupling is reduced to **16 %** of what it was, and the dispersion of individual speeds goes from $\pm 47\%$ to $\pm 147\%$. The population begins to lose the unison **before** the section changes.

pressure	coherence of the colony
0 %	0.999

pressure	coherence of the colony
25 %	0.999
50 %	0.996
75 %	0.981
100 %	0.859

It shows but does not announce itself: the colony **grows uneasy, it does not break**. It is the only thing in the whole light system that shows the machine and not the ecosystem, and for that reason it has a switch of its own (P 0).

14. The pattern travels when the discourse advances

The signed drift (§7.4) decides whether the drawing **shifts** along the strip, and in which direction.

```
if (VIATGE && (deriva < 126 || deriva > 130)) {
  int16_t d = (int16_t)deriva - 128;
  f += (uint16_t)(int16_t)((int32_t)d * (int32_t)i * 5);
}
```

The dead zone of ± 2 around the centre is not cosmetic: the D command arrives in 0–100 and is rescaled to 0–255, so that exactly 128 cannot be requested from Max. Without it, “still material” would have been an unreachable state.

It is a **spatial phase ramp**: the phase of each individual is shifted in proportion to its position. Combined with the advance of time, it produces a wave that runs along the strip. The sign of the drift reverses its direction.

drift	wave per strip	what it means
128	0.00 cycles	still material: the drawing does not move
154	+0.20	the discourse grows — the wave goes towards one side
204	+0.57	
255	+0.96	maximum growth
102	–0.20	the discourse decreases — the wave goes towards the other
52	–0.57	
0	–0.97	maximum decrease

The ramp does not touch the population. It modifies neither the coupling nor the real phases: the coherence the Arduino returns does not change at all. It is a transformation of the drawing, not of the model. The colony remains as united as it was; it is simply looked at from another angle.

Musically, this closes the circle that the drift opens. The same number that makes the system wait or push is the one that makes the light travel or stay still. A frenetic tremolo that goes nowhere leaves the strip motionless however brightly it shines; a truly long *decrescendo* makes it run. And since the drift has a sign, the light does not only say **that** the music is moving: it says **towards where**.

It is disabled with V 0.

15. The mechanics of the colony

Each individual is an index into three tables and a deterministic function:

fase[i]	uint16_t	position within the cycle, circular by natural overflow
pas[i]	int16_t	phase increment per frame, limited to [2, 2500]
hash8(i)	uint8_t	the “personality”: speed deviation and order of death in F

hash8() is a 16-bit *xorshift-multiply*, purely deterministic and identical between the Arduino and the viewer. This guarantees that the dissolution of F always switches off **the same** pixels: the dissolution has memory, it is not new noise at every frame.

The colony starts up **deliberately disordered**: fase[i] = random(65536), pas[i] = random(40, 400).

15.1 The coupling, in two stages

```
// periode: guany acobl/4096, maxim 6.2 % per fotograma
pas[i] += ((pas_mig - pas[i]) * acobl) / 4096;

// fase: guany acobl/8192, maxim 3.1 % per fotograma
int16_t dif = (int16_t)(fase_mitja - fase[i]);
fase[i] += (dif * acobl) / 8192;
```

The phase gain is **deliberately half** that of the period: the population first equalizes the rhythm and then the alignment. It is the order in which it happens in nature, and visually it is far more beautiful than if they converged at the same time.

The trick of the central line: the subtraction in uint16_t followed by the *cast* to int16_t automatically gives the **shortest signed circular distance**. No conditional is needed to decide whether it is better to advance or delay the phase.

And the biological noise that prevents the synchrony from being mechanical:

```
if (agitacio > 55) pas[i] += (random(-3, 4) * agitacio) / 128;
```

It only acts when the coupling drops below 200. With high coupling the population is clean; with low coupling each individual drifts.

15.2 The collapse

```
if (acobl < 20 && random8() < 6) {
    fase[i] = random(65536);
    pas[i] = random(200, 1400);
}
```

Below coupling 20, each individual has a 2.34 % probability per frame of restarting completely — on average, every 0.85 seconds. **Only D reaches it through its own axes** (coupling 6). With the pressure to change at maximum, B and C also drop there (121 → 19), F (49 → 7) and E-bifurcation (94 → 15); A (235 → 37) and E (150 → 24) never reach it. And the new pas (200–1399) is faster than the average: **the collapse also accelerates**.

15.3 The coherence

```
cfor (i) { sx += cos8(fase[i] >> 8) - 128; sy += sin8(fase[i] >> 8) - 128; } mag = sqrt(sx*sx + sy*sy); coherencia = min(255, (mag * 2) / NUM_LEDS);
```

It is literally the modulus of the **Kuramoto order parameter**: 1 when they all go exactly together, 0 when they are completely scattered. The Arduino returns it to Max every half second as plain text, and it can be watched in real time from the patch.

section	coherence
A	0.999
B	0.990
C	0.989
D	0.149
E	0.997
F	0.974
E bifurcation	0.978

At D it falls to 0.149: **the cycle has really broken**. It is the numerical measure of the subject of the piece. With the hundred LEDs of the previous version it fell to 0.163; the collapse goes deeper the larger the population, which is exactly what Kuramoto predicts — the order parameter of a disordered population of N oscillators decreases as $1/\sqrt{N}$.

The figures for the five stable sections are robust. D's is not: with the collapse active, each individual restarts with a random phase several times per second, so the coherence fluctuates a great deal and the exact value depends on the seed of the generator. What is stable is **the order of magnitude**: D sits one order below all the rest.

16. E and F are processes, not states

The other sections are states: you enter them and you are in them. E and F are **processes that unfold across the whole section**, governed by a slow clock independent of the three-second crossfade between sections.

E decrease — over 120 seconds, the population **withdraws from the centre toward the two ends**: the contact zone goes dark first and each colony pulls back toward its musician. The pixels left outside are reduced to 15.6 % of their brightness.

```
uint16_t vora = min(i, NUM_LEDS - 1 - i);
uint8_t nucli = scale8(NUM_LEDS/2, 255 - progresLent(120));
if (vora > nucli) v = scale8(v, 40);
```

The progression: at 25 % of the process LEDs 38–61 dim; at 50 %, LEDs 26–73; at the end only LED 0 and LED 99 are still lit.

E bifurcation — the right half of the strip (LEDs 50–99) is recomputed in **exact antiphase** and forced to a breath shape. There is no temporal progression: the divergence is immediate and constant. The two halves do not meet again for as long as the section lasts. It is the literal translation of “tension that does not resolve”.

F dissolution — two simultaneous mechanisms over 150 seconds:

```
uint8_t llindar = 130 - scale8(85, prog); // from 130 to 46
if (hash8(i) > llindar) v = 0; // mortality
v = scale8(v, 255 - scale8(170, prog)); // extinction of the survivors
```

	start	midpoint	end
living individuals	54 %	36 %	17 %
brightness of the living	100 %	~65 %	34 %

F ends with seventeen pixels at 34 % of their brightness, over a base that is already desaturated blue at brightness 63. **Measured on the firmware, the total light leaving the strip at the end of F is around 12 % of that of D** — an order of magnitude below. The habitat layer is what keeps it from dropping further: the dissolution of F acts on the insects, not on the medium.

17. What Max does and what the Arduino does

Max only says what is happening. All the animation lives inside the Arduino.

This decision has two motives. The first is practical: `p_lights` stays at a handful of objects instead of turning into a graphics engine inside Max. The second is about robustness and is the one that decides: **if the USB cable comes unplugged mid-concert, the colony stays alive** with the last state that reached it, instead of freezing or going dark.

The protocol is plain text, one command per line, at 115200 baud:

```
S <section 1-6> <bifurcation 0|1>
H <dens> <noise> <attack> <movement> <dynamics>      horn,          values 0-100
T <dens> <noise> <attack> <movement> <dynamics>      trombone
E <dens> <noise> <attack> <movement> <dynamics>      electronics
X <7 axes of the section>                               0-100
M <pressure to change 0-100>      D <drift 0-100, 50 = still>
R <reality 0-100>                P <sensing 0|1>
C <habitat 0|1>                  V <travel 0|1>
```

And the Arduino returns, every half second:

```
coherencia <0-255> seccio <name> pressio <0-255>
```

The X command is the one that closes the JSON circle: **neither the Arduino nor the viewer has its own copy of the section axes**. The firmware carries a fallback table, but as soon as it receives the first X it stops using it. You edit the JSON, press reload on the js, and the strip and the viewer find out at the same time.

18. The viewer inside Max

`p_lights` contains a jsui that loads `leds-visor.js`: a window inside the patch that draws what the physical LEDs will do.

It is not a mock-up. It is the same Arduino engine ported to JavaScript, with the same formulas, the same thresholds, the same `hash8()` and the same `peSTrompa()`. And above all: **it is fed by exactly the same messages**. Every `leds-*` bus has two receive objects inside `p_lights` — one toward the `sprintf` and the serial port, the other toward the jsui. There is no way for the viewer and the strip to receive different data.

Checked with the seven section configurations: **the viewer and the firmware give numerically identical values** for the coupling, the hue, the saturation, the brightness and the waveform.

18.1 The tester

Next to the OBRIR LLUM button of the main interface there is a second button, **PROVADOR DE LLUM**, which opens a test bench: seven buttons for the sections, nine scenes of what the musicians can do, and the commands (pressure, reality, habitat, drift, travel).

Each button writes directly to the buses that the .js uses, with Max's semicolon syntax. So **it has not a single patch cord**, and what it sends arrives at the viewer and at the strip at the same time. It is there to compose the light without hardware and without musicians.

The one precaution: **the engine must be stopped** (stop) before using it. If it is running, it republishes the values every second and overwrites whatever has been sent by hand.

19. Hardware

Board	Arduino Uno or Nano. It puts out 5 V, which is what the WS2812B wants: no level shifter needed
LEDs	WS2812B strip, a whole 150 pixels , 5 V, GRB colour order. A 5 m roll at 30 LED/m: the run is 5.00 m and the individuals sit 3.33 cm apart
Power supply	5 V, 5 A minimum. The code limits consumption to 3 A in software
Capacitor	1000 μ F at the input of the strip
Resistor	330 to 470 ohm in series with the data line
Push-button	pin 2 to ground, with the internal <i>pull-up</i>

```

Arduino pin 6  --[ 330  $\Omega$  ]-->  strip DIN
Arduino GND    ----->  strip GND
PSU 5 V +      ----->  strip +5 V          (at THREE points)
PSU 5 V -      ----->  strip GND and Arduino GND

```

Three things that always fail if you do not think about them:

1. **Common ground** between the power supply and the Arduino. Otherwise the strip does random things.
2. **Inject current at three points: the two ends and the middle.** With a long three metres two were enough; with five, they are not. If the current has to travel two and a half metres of thin copper, the voltage arrives sagging at the centre and the whites turn orange right in the middle of the run. It would happen in F, which is all whites and greys.
3. **Do not power the strip from USB.** A hundred and fifty LEDs can ask for 9 A at full white; USB gives 0.5.

The memory taken up by the state of the colony is **1,050 bytes** of the 2 KB of an Uno (seven bytes per individual: three for colour; two for phase and two for speed). The code works just the same with NUM_LEDS 4: all the geometry is parameterized, so it can be tested with a four-LED rig before building the strip.

The push-button on pin 2 advances through the sections cyclically and, on entering E, alternates the branch. It is there to test the installation without Max.

PART SIX · VERIFICATION

20. How this was checked to work

A system that decides the shape of a piece in concert cannot be tested only by playing it. Verification is done on three fronts, and the whole of it can be re-run in a few minutes.

20.1 The decision engine, outside Max

`markov-cicada.js` runs in a Node environment with the Max objects simulated — `Dict` reads the real JSON, `post()` collects the console, `messnamed()` and `outlet()` are logged. **It is the file that runs in concert, without a single line modified**, running against the JSON that runs in concert.

Five performance profiles have been defined on top of this setup:

profile	what it simulates
static	low, invariable density
evolving	continuous growth from nothing to saturation
dialogue	slow waves, like a duo answering itself
contrasting	alternating blocks of high and low density every 90 s
agitated	high activity but no direction: a lot of trembling, no drift

Each profile is run with and without the pedal, and **the nine rules are checked at every run**: the count of each section, E never consecutive, F last, no section below its minimum or above its ceiling, and the absolute ceiling of the piece.

20.2 Result

600 runs, 0 violations. Median duration **14'05"**, from 7'14" to 25'42", with 184 different paths.

In a larger batch, by profile:

profile	pedal	median	minimum	maximum
static	no	10'47"	8'21"	13'58"
static	yes	9'56"	7'42"	11'48"
evolving	no	15'56"	10'33"	18'34"
evolving	yes	13'00"	9'36"	15'34"
dialogue	no	18'56"	11'47"	25'41"
dialogue	yes	14'59"	9'25"	20'59"
contrasting	no	18'57"	12'11"	23'55"
contrasting	yes	15'42"	9'59"	20'07"
agitated	no	15'00"	10'00"	20'13"
agitated	yes	13'44"	8'44"	16'45"

Three readings worth drawing from this table:

The pedal shortens the piece by one to four minutes in every profile. It makes sense: the pedal is an additional route for changing section, never a route for prolonging one. Musicians who turn the page quickly make a shorter piece, and the one-minute minimum per section is what keeps that from degenerating.

The dialogue profile is the one that gives the longest pieces (median 18'56"). It is the one with the most sustained drift, and discourse slows the system down. A duo that really answers itself gets the system to let it work.

The agitated profile gives shorter pieces than the dialogue one despite having more activity. That is exactly what the drift mechanism was meant to do: agitation without direction reads as stagnation.

20.3 The maximum duration

The practical question — *what is the longest the piece can be?* — has two answers.

The hard limit is 30 minutes. At MAX_TOTAL - RESERVA (1600 s) the system goes towards F no matter what, respecting the minimum minute of the section in progress; at 1800 s it dissolves even if F has not finished.

But the real limit is lower, and it is structural. Adding up the hard ceilings of all the sections in the absolute worst case — A, B, C and D at 300 s, two E at 180 s and F at 180 s — gives 1740 seconds, that is **29 minutes**. Since 29 is less than 30, **the absolute ceiling cannot come into play through the accumulation of sections**: it could only do so if someone lengthened the per-section ceilings by editing the JSON. It is a second-level safety net, not an active mechanism.

In the simulation batches the maximum observed has been 25 minutes and the absolute ceiling has never fired.

20.4 The space of paths

With the nine rules, the number of **valid** paths is exactly **300**: 96 with a single appearance of E and 204 with two.

In 2,400 runs spread across all the profiles, **259** have been observed: 80 paths of six sections and 179 of seven. The 41 that have not appeared are not forbidden — they are the ones the distance and direction filters make very improbable, which is precisely what is wanted. **The system covers 86 % of the space the rules allow it.**

20.5 Flattening, measured

600 runs per value, evolving profile:

flat	A-B-C-D path	different orderings seen
0 — the written grammar rules	20.3 %	152
0.3	19.7 %	176
0.6 — <i>default</i>	13.5 %	187
1 — the matrix disappears	12.8 %	179

Two readings.

The real jump is between 0.3 and 0.6. From 0 to 0.3 almost nothing happens; from 0.6 on, the linear path drops by a third and the variety of orderings rises to its maximum. Past that point, what rules is no longer the matrix.

With the matrix completely flattened, the linear path still appears 12.8 % of the time. It does not fall to zero because the other two filters — the distance between sections and the projected density — already push on their own towards a neighbour-to-neighbour path. Put another way: **the A-B-C-D progression is not imposed by the matrix, it is already there in the description of the piece.** The sections are written in such a way that going from each one to the next is, almost always, the shortest move in the space of the seven axes. The base matrix does nothing more than reinforce a tendency that is already there.

20.6 The drift, measured

The slow average has about three minutes of memory, so a measurement made from zero says little: everything looks as though it is “developing” because the slow average is still rising. The following table lets each profile run for twenty minutes and averages the drift of the last five — the steady state.

profile	density	drift	discourse	
constant low density	0.00	0.000	2.20	stagnant: pushes
agitated with no direction	0.59	0.025	1.59	stagnant: pushes

profile	density	drift	discourse	
continuous growth (already saturated)	1.00	0.047	1.12	stagnant: pushes
alternating blocks	0.41	0.361	0.45	developing: brakes
slow waves	0.45	0.359	0.45	developing: brakes

The key result is the comparison between **agitated with no direction** and **alternating blocks**. The agitated one has *more* average density (0.59 against 0.41) and much more apparent activity, but a drift fourteen times smaller: the system reads it as **stagnation** and pushes towards change. The alternating blocks, with less density but real movement, **brake** the system by half.

The third row is eloquent too: continuous growth stops counting as discourse **when it reaches the top**. Once density saturates at 1.00, it no longer drifts, and the system pushes again. The musicians cannot buy time by staying at saturation.

20.7 The light, outside the Arduino

The firmware logic is compiled and run on the computer with a minimal *shim* of the Arduino and FastLED APIs — the whole .ino file with a fixed seed, with no changes other than removing the `#include <FastLED.h>` line and the `addLeds` one, which the simulator replaces — simulating **one hundred and fifty individuals for sixty seconds in each section**. All the figures in §10.2, §12, §13, §14 and §15.3 come from there.

The viewer `leds-visor.js` has been checked against the firmware with the same inputs: **it produces identical values in the seven section configurations**.

20.8 The patch, structurally verified

The .maxpat file is JSON, and it can be audited without opening Max. The checks that are run at every modification:

- no cable points to a non-existent object;
- no cable points to an *inlet* or an *outlet* out of range;
- **the number of inputs and outputs each subpatcher declares matches the inlet and outlet objects it contains** — this check is the one that catches the case where Max deletes a cable when opening the file;
- no receive is left without a source, counting the buses published by the .js with `messnamed()` and those written by the provador buttons;
- and, comparing with the previous version, that no object and no connection has been lost.

Current state: **0 problems**. The only send without a receiver are `attack` and `markov-densitat`, which are available outputs and not errors.

20.9 What the simulation cannot say

The limits of this verification have to be stated clearly.

The light installation has never worked with real hardware. Everything said about it in this article is verified by simulation of the complete firmware, but not a single physical LED has been lit. Until that is done, these are well-checked hypotheses.

The five profiles are mathematical functions of density, not musicians. They feed the system through the *baixa/mitja/alta* buses. The per-instrument descriptors — noise, attack, movement, dynamics — are not exercised realistically, and therefore the recognition filter (§7.2, filter 4) and the whole reactive light layer are not verified against real sound material.

That means it is proven that **the rules are never violated** and that **the durations and the paths are what they should be**. It is not proven that recognition picks the right section, the one that resembles what is being played, nor that the flatness calibration is correct in the hall. Only a rehearsal with the instruments can say that.

PART SEVEN · WHAT CAN BE TOUCHED

21. The rehearsal parameters

Everything that follows can be changed **live, without restarting anything**. The messages go to the inlet of the js object inside p MARKOV-cicada; the light ones, to the boxes in p lights or to the provador buttons.

21.1 The ones worth touching with the musicians in front of you

message	default	what it does
flat <0-1>	0.6	how much authority the written grammar has over the order of the sections
escolta <1\ 0\ -1>	1	CALLS the section it resembles / only measures / AVOIDS it
escoltapes <0-1>	0.7	how much the recognition weighs
target <minutes>	15	target duration (0 = free)
dialeg <1\ 0\ -1>	1	complementary / neutral / imitative electronics
realitat <0-100>	75	the light is the score ↔ the light is what sounds
pressentir <0\ 1>	1	whether the colony grows restless before section changes
habitat <0\ 1>	1	whether the electronics sustain or threaten the population
viatge <0\ 1>	1	whether the pattern travels as the discourse advances

21.2 The calibration ones, which have to be adjusted to the hall

message	default	what
calibraenv <lo> <hi>	20, 90	dB from sigmund~ → dynamics 0-1
calibra <lo> <hi>	0.40, 0.85	spectral flatness of a tonic sound and of an air sound
useflat <0\ 1>	1	spectral flatness or high-frequency ratio

Calibrating the flatness is the most important one of all. It is done like this: the musicians are asked for a clean, sustained note, bandes is sent and the flatness that appears in the console is written down; then air or friction is asked for and the second value is written down. Those two numbers are FLAT_LO and FLAT_HI. Without this step, the noise descriptor — which is what decides half the system — works with the values of another hall.

21.3 The diagnostic ones

message	what it prints to the console
dump	section, page, time, density, trend, pressure, drift (absolute and signed), discourse, path
viu	the live vector, the distances to each section, and which one it resembles most right now

message	what it prints to the console
bandes	the seven bands of each instrument, the flatness and the noise that comes out of them
distances	the matrix of distances between sections
temps	the table of minimums, averages and ceilings in force
describe <A-F>	the whole description of a section as it is in the JSON
bancs	which banks are authorised, what phrase phase they are in and at what volume

21.4 Which file to touch

A practical question worth answering clearly: **of `cicada-seccions.json` and `markov-cicada.js`, which one should be adjusted in rehearsal?**

The JSON, almost always. It is where the durations, the axes and the banks of each section live: exactly the three things one will want to adjust when the piece is heard in the hall. It is edited, `reLoad` is pressed, and both the decision chain and the light find out about it. There is no need to restart Max.

The `.js` should only be touched to change the **behaviour** of the system, not the description of the piece: the base matrix `W` (lines 97 to 104), the axis weights `AXIS_W` (line 135), the flattening `FLAT` (line 151), the discourse constants, the durations of the electronics phrases. It has `autowatch = 1`, so Max rereads it on its own when it is saved — but a change to the `.js` mid-rehearsal resets the state of the engine.

PART EIGHT · LIMITS AND OPEN WORK

22. What is left to decide

The physical layout, decided on 26 August 2026. A five-metre aluminium profile with an opal diffuser, laid on the floor in front of the performers' feet, with the LEDs facing the ground and the body raised four or five centimetres on feet. The criterion that decides this is not aesthetic but practical: **the screen behind is the score the musicians read**, so any grazing light from the front projects two three-metre shadows onto it and makes what they have to play illegible. The way out is not to hide the light but to turn it: if the beam goes downwards and what is seen is the lit surface and not the diode, the light can be large, present and continuous without ever touching the screen. From the stalls not a single diode is visible: what is seen is a five-metre band of lit floor, with the duo standing in it. The full detail is in leds/MUNTAR-LA-TIRA/.

What remains open is **the exact height of the profile** —four centimetres is the starting point, not a sacred measurement— and **whether the three-ampere limit gives enough brightness** with the audience seated. Both things are adjusted in the hall without touching the code: the first with the feet, the second with a number.

If one day the hundred and fifty LEDs were spread around the space instead of running in a line, the geometry of the territories would have to be rethought — `pesTrompa()` and the two centres of gravity — but the engine does not change.

Three adjustments that can only be made in the hall: the exact point where the hue turns from blue to orange (currently 100), how much reality, and whether the sensing stays.

22 bis. Why a hundred and fifty and not a hundred

The installation was born with a hundred LEDs because a hundred is a round number. WS2812B strips, however, are sold in five-metre rolls, and at 30 LED/m one roll is **exactly a hundred and fifty**. Making it a hundred meant cutting the strip, soldering the conductors onto it and keeping fifty LEDs in a drawer; making it a hundred and fifty means opening the bag and laying it out.

The change, though, came with a better argument than convenience. With a hundred LEDs the band was 3.33 m, and that figure had been chosen so that it would match the width of the projection screen. But **the screen changes size in every theatre**, and the installation should not have to keep adapting to it. With the whole roll, the band is **five metres and it is a measurement of the piece**, not of the hall: the performers place themselves over their thirds and the geometry of the territories follows them, whatever screen there may be behind.

The engine did not have to be touched. Everything is written as a function of `NUM_LEDS`: `pesTrompa()` normalises the position to 0–255 before deciding anything, the order parameter divides by the population, and the durations and thresholds are fractions. The coherence, the colours, the waves and the periods of the seven configurations come out identical.

Two exceptions, and they are worth explaining, because they were latent errors that could not be seen with a hundred LEDs and that only the change of scale uncovered:

- **The extent of the colony** —how much territory each population occupies according to the *width* axis— was calculated as $|i - \text{centre}| \cdot 4$, that is, in **pixels**. With a hundred and fifty LEDs the same command covered proportionally less strip, and the colony would have withdrawn towards its centre: section A lost a fifth of its average brightness without a single number in the score having changed.
- **The drift ramp** —the displacement of the pattern along the strip— was calculated as $d \cdot i \cdot 5$, also in pixels. The unit of this wave, however, is the **whole strip**: with the drift at maximum, one wave has to fit from end to end. With a hundred and fifty LEDs, 1.44 of them fitted.

Both are now $\cdot 400/\text{NUM_LEDS}$ and $\cdot 500/\text{NUM_LEDS}$. With a hundred LEDs these expressions are worth exactly 4 and 5, so that **the behaviour of the previous version is reproduced figure by figure** — which has been verified by compiling both versions and comparing their outputs. With a hundred and fifty, the colony occupies the same proportion of strip and the wave is back to 0.96 cycles.

The only figure that really changes is **the coherence at D**, which goes from 0.163 to 0.149. It is not a side effect: it is what has to happen. The Kuramoto order parameter for a disordered population decreases with $1/\sqrt{N}$, so the more individuals there are, the deeper the collapse reaches. The section that speaks of the breaking of the cycle breaks a little more.

The Arduino's memory goes up from 700 to **1,050 bytes** of the 2,048 of a Uno: seven bytes per individual —three of colour, two of phase, two of speed—. It fits with room to spare. And the software current limit goes up from 2,000 to **3,000 mA**, because otherwise the brightness peaks would be spread over half again as many diodes and the piece would come out weak precisely where it has to carry weight.

23. Known limits

These are the real weak points of the system, in order of importance.

The installation has not been tested with hardware. See §20.9. It is by far the biggest limit.

The three per-instrument bonk~ in the chain are not calibrated. Only the bonk~ of the global chain receives the configuration messages and the loadbang training. The three new ones run with the default values. They work — attack detection is robust enough — but the density descriptor would gain reliability if they were calibrated like the first one.

The analysis of the electronics does not measure exactly the same thing. hrnhf and tmbhf are taken from outlet 1 (high-pass) of the svf~ 2500 0.6, whereas elehf is taken from outlet 2 (band-pass). With USE_FLAT = 1 — the default configuration — this has no effect, because the hf descriptor is only used when the noise is calculated by high-frequency ratio. But if one day useflat 0 is set, the three layers would stop being comparable.

Once the first X arrives, the Arduino's internal tables are left inactive. eixos_rebuts is not set back to zero, and there is no command to revert it short of restarting the board. The practical consequence: the internal reserve is only worth anything if the cable fails *before* the first X. After that, a disconnection leaves the strip with the last vector received — which is the robust behaviour one is looking for, but by another route.

Unavoidable differences between the firmware and the viewer. The formulas are the same and the section values match exactly, but the environment is not the same:

- **Arithmetic.** The firmware truncates to integer at every accumulation; the viewer keeps fractions. The firmware therefore has **synchronisation dead zones** that the viewer does not have: with low coupling, the Arduino freezes speed differences that the viewer keeps slowly reducing.
- **Wave tables.** The viewer reimplements FastLED's functions analytically, and the curves do not match exactly. FastLED's quadwave8 reaches the peak smoothly; the viewer's reaches it abruptly. The zeros and the maxima are the same; the intermediate dynamics are not.
- **Colour space.** The firmware uses FastLED's hsv2rgb_rainbow — the “rainbow” spectrum, with yellow widened and brightness compensation per hue. The viewer implements standard HSV. **It is the most visible divergence**, above all in the yellow-orange band where the whole of section C falls: the same hue number gives appreciably different colours between screen and strip.
- **Clock.** The firmware measures with millis(); the viewer counts ticks of the metro 20. If the metro does not run at exactly 20 ms, the E and F processes run at different rates.
- **Section repetition.** The firmware ignores an S message that does not change anything; the viewer does not have this guard and restarts the E or F progression every time it receives one.

The viewer is there to **compose and decide**. For fine colour adjustment, the reference is the strip.

Small things in the patch. s_attack and s_markov-densitat have no receiver: they are outputs available for future use. The twelve text fields with the electronica folders still point to VERSIO-1 (the two .js files are loaded without a path, and the seven films of p_score point to VERSIO-2). jit.qt.movie is an obsolete object in Max 8: it works, but it is a fragile point with a view to a future version. And bonk~.mxo and sigmund~.mxo are not packaged with the patch: they have to be installed on the system before opening it on a new computer.

24. Open work

Having the light read articulation too. Right now the five descriptors measure quantity, quality and movement of the sound, but not its **articulation** — legato against staccato beyond attack density. It would be a sixth descriptor.

Having the coherence feed back into the decision. The Arduino sends the coherence of the colony every half second and Max only displays it. The circle could be closed: a visually broken ecosystem could influence the probability of a section change. It is the only feedback the system still does not have — and it needs thinking about, because it would introduce a loop between the light and the form that right now does not exist on purpose.

APPENDICES

A. Full serial protocol

From Max to the Arduino (plain text, one command per line, 115200 baud, CR terminator):

command	arguments	range	effect
S	section, bifurcation	1-6, 0/1	changes section (ignored if nothing changes)
H	dens, noise, attack, mov, dynamics	0-100	live vector of the horn
T	same	0-100	live vector of the trombone
E	same	0-100	live vector of the electronics
X	dens, noise, attack, width, mov, tempo, dynamics	0-100	the seven axes of the section, from the JSON
M	pressure	0-100	pressure to change
D	drift	0-100	50 = still, >50 the discourse grows, <50 it shrinks. Dead zone of ± 2 at the centre
R	reality	0-100	score $\leftrightarrow$ real sound
P	sense	0/1	anticipated unease
C	habitat	0/1	the electronics sustain or threaten
V	travel	0/1	the pattern travels or stays still

Unknown letters are silently ignored. The line buffer is 48 bytes.

From the Arduino to Max (every 500 ms):

coherencia <0-255> seccio <A|B|C|D|E-decrease|E-bifurcacio|F> pressio <0-255>

B. Messages markov-cicada.js understands

Transport: start·stop·reset·force <1-6>·reseed·reload

Clock: bang (one-second tick, from metro 1000)

Pages: pagina (pedal, open mode)·setpage <n> (real page from p score, closed loop)

Analysis, per source — <x> is hrn, tmb or ele: <x>env <dB>·<x>pitch <Hz>·<x>tot <rms>·<x>hf <rms>·<x>band <0-6> <rms>·<x>atk

Activity (global chain): baixa·mitja·alta·envelope <dB>·pitch <Hz>

Tuning: flat·adapt·target·maxtotal·randomstart·transformdist·bifurcadist·mindwell <s> <v>·meandwell <s> <v>·maxdwell <s> <v>·pagines <s> <v>

Listening: escolta <1|0|-1>·escoltapes <0-1>·calibra <lo> <hi>·calibraenv <lo> <hi>·useflat <0|1>

Electronics: dialeg <1|0|-1>·dialepes <0-1>·samplersauto <0|1>·fadesec <s>·frases <repos> <cant>

Diagnostics: dump·viu·bandes·bancs·distances·temps·describe <A-F>

Object outlets:

outlet	content
0	section 1–6 → s ordre-num-de-seccio
1	pressure to change, 0–1
2	density, 0–1
3	state lists: state, crono, change, done, pagina, armat, viu, frase

And, via `messnamed()` (with no visible patching in the patch): `leds-seccio · leds-eixos · leds-hrn · leds-tmb · leds-ele · leds-pressio · leds-deriva · vol-banc-1 to vol-banc-5 · markov-bancs-pres · the five bank names`.

C. Main constants

Times (read from the JSON; those in the `.js` are the fallback):

	A	B	C	D	E	F
minimum before deciding	60	60	60	60	60	60
expected extra average	200	190	190	170	100	110
hard ceiling	300	300	300	300	180	180
pages	4	4	4	4	1	1

Absolute ceiling of the piece: **1800 s**, with a reserve of **200 s** to close with F.

Decision: `FLAT 0.6 · ADAPT 0.75 · SIGMA 0.35 · SIGMA_D 0.28 · SIGMA_L 0.30 · RES_W 0.7 · DIST_TRANSFORM 0.22 · DIST_BIFURCA 0.70 · TARGET_MIN 15`

Discourse: `DENSITY_ALPHA 0.06 · SLOW_ALPHA 0.006 · FLUX_REF 0.09 · STAG_BOOST 1.2 · DEV_BRAKE 0.55`

Saturation: `SAT_LEVEL 0.82 for SAT_TIME 110 s · DESAT_LEVEL 0.18 for DESAT_TIME 130 s`

Analysis: `ENV_LO/HI 20/90 dB · FLAT_LO/HI 0.40/0.85 · ATTACK_REF 9 dB · MOD_REF 9 turns/s · ENV_HYST 1.5 dB · PITCH_HYST 0.6 % · LIVE_ALPHA 0.05`

Light: `NUM_LEDS 150 · FPS 50 · BRILL_MAX 200 · AMPERES_MAX 3000 mA · crossfade between sections 150 frames (3 s) · REALITAT 191 (= R 75) · PRESENTIR 1 · HABITAT 1 · VIATGE 1 · hue threshold 100 · pulse threshold 92 · strobe threshold 179 attack and 179 noise`

D. Files

file	size	what it is
<code>cicada-seccions.json</code>	10 KB	the description of the piece. Single source
<code>markov-cicada.js</code>	45 KB	the decision and listening engine
<code>leds-visor.js</code>	14 KB	the light viewer inside Max
<code>patch-MaxMsp-8-CICADA-KILLER-MARKOV-V2.maxpat</code>	1.2 MB	the patch
<code>CICADA-LEDS/CICADA-LEDS.ino</code>	28 KB	the firmware for the installation