Joachim Heintz

Creating Self-Generative Structures in Csound

Workshop "Csound Advanced" (online)

7th Tehran International Electronic Music Festival
September 6, 2025

Our time today:

I. Discuss and implement self-generative structures in Csound.
(~ 1 hour 15 minutes)

LINK to the online live code environment:
https://ide.csound.com/editor/AZZwhIG1bZu4A7hLS2ye
Or you can download the .zip file here:
https://joachimheintz.de/workshops/tehran_2025/workshop.zip

II. I show you one example from myself (sound installation)
(~ 15 minutes)

III. I give you a short glimpse of Csound 7
(~ 15 minutes)

Pleeeease:

ALWAYS ASK IMMEDIATELY

if something is not clear

ASK via UNMUTE and saying "Stop!"

or use the "Raise Hand" button

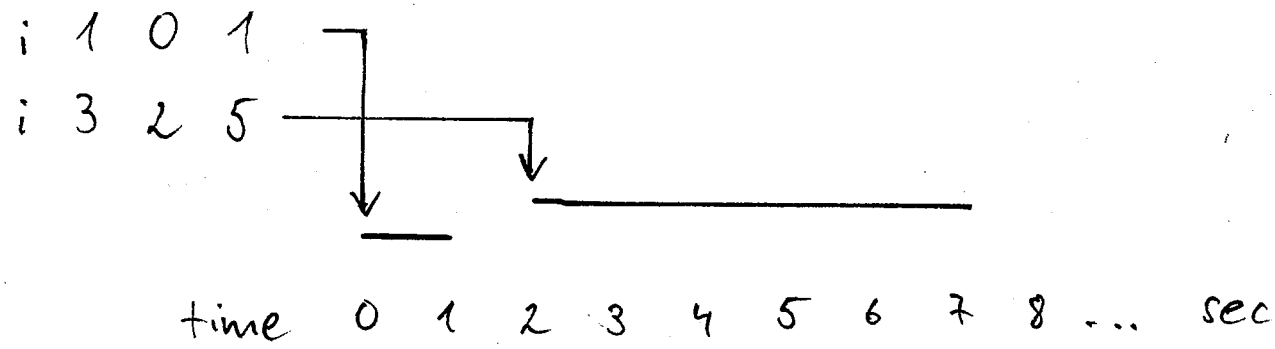
Thank you!

I. WORKSHOP WITH OWN CODING AND EXERCISES

What are "self-generative structures"?

NOT

a list of events



BUT

a structure

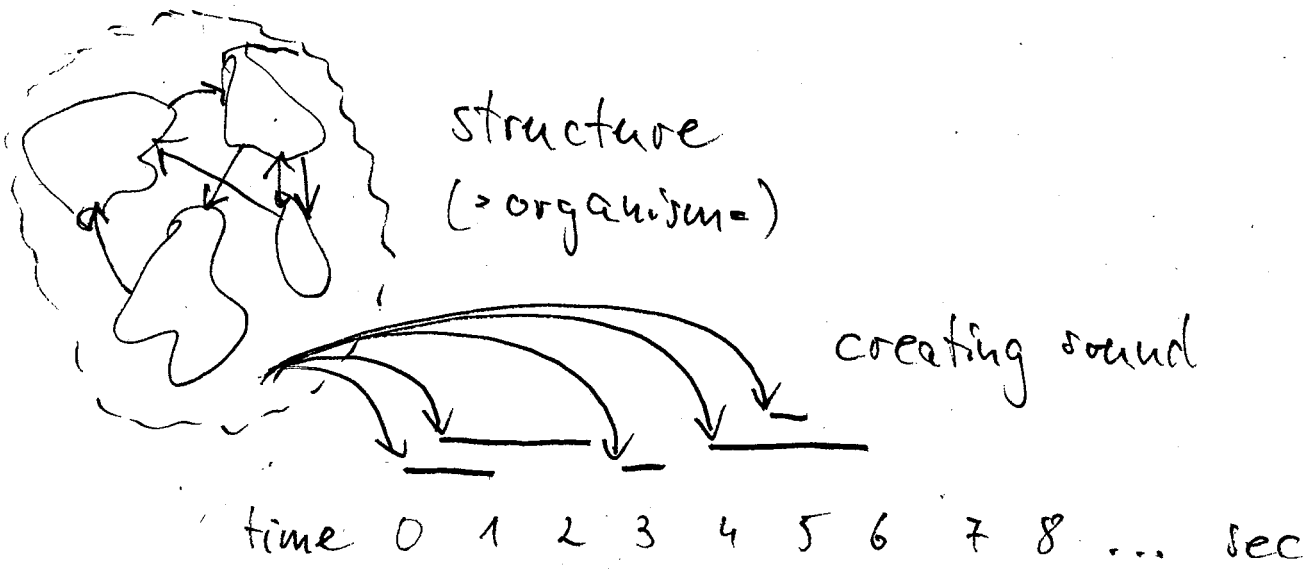
with forces

with internal parts

with communication between these parts (quasi organs)

so kind of an organism

which generates sound



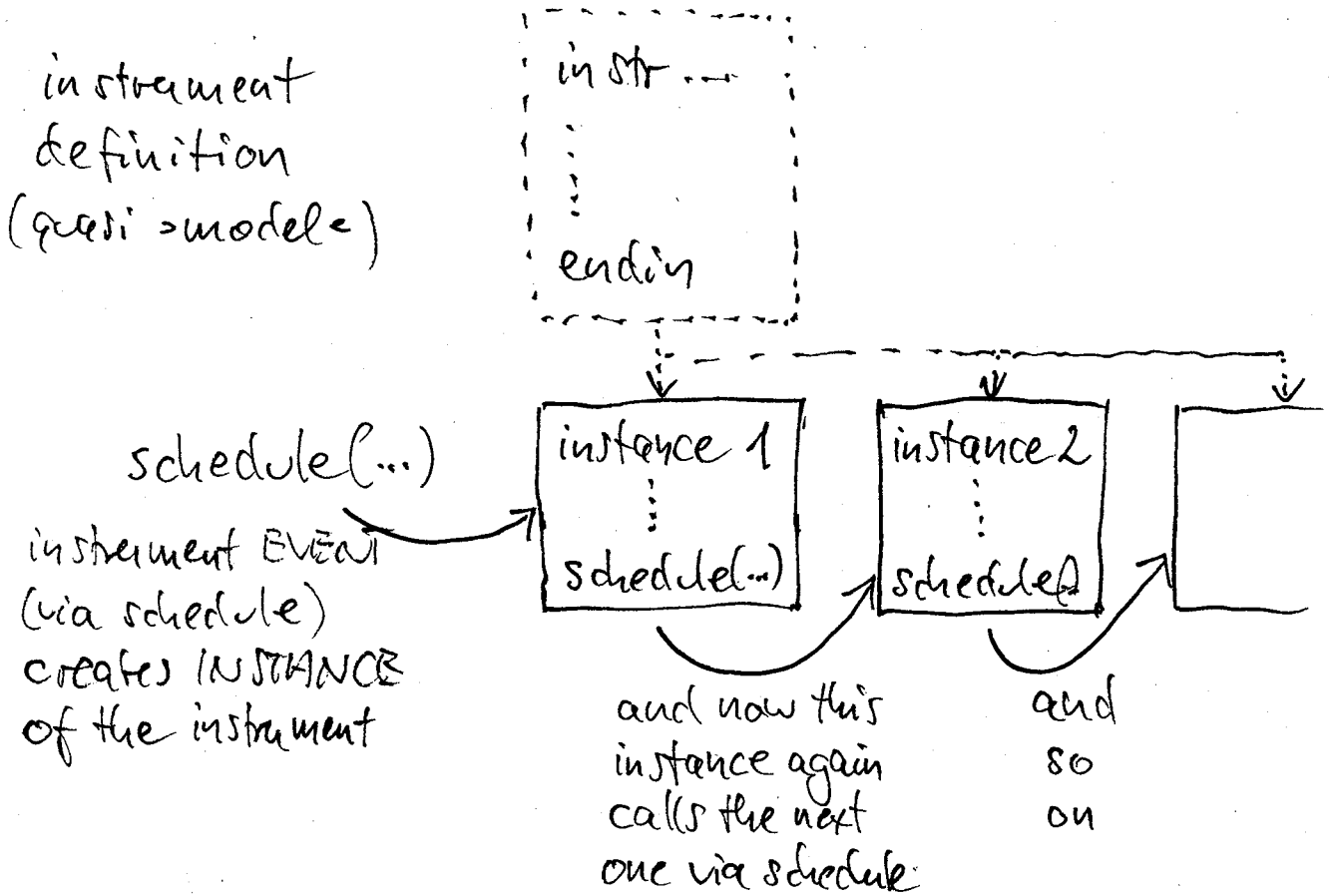
01

Let's code together!

Go to ide.csound.com or use your own Csound editor.
We code this step by step.

```
instr SimpleSelf

  // say hello
  puts("Hello",1)

  // call another instance of this instrument
  schedule("SimpleSelf",1,1)

endin
schedule("SimpleSelf",0,1)
```

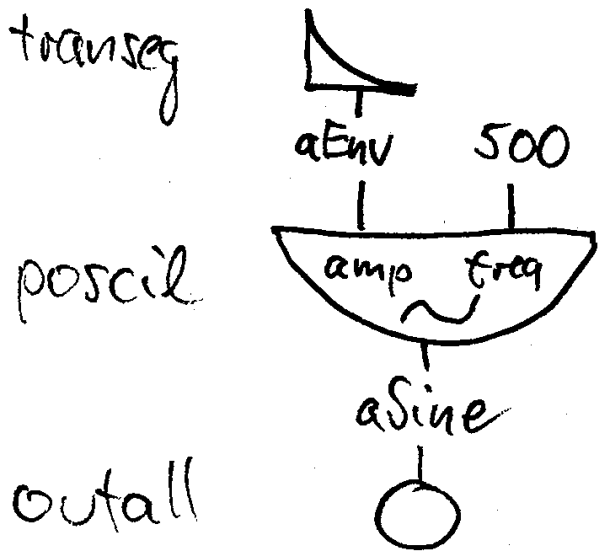
What happens



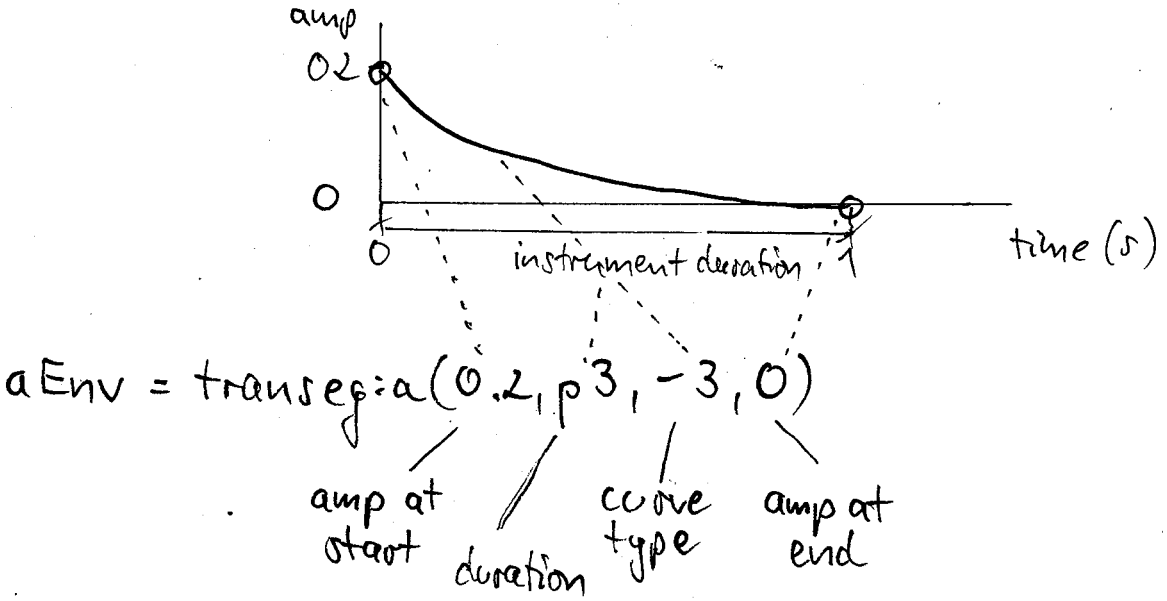
02

Now sound instead of printouts ...

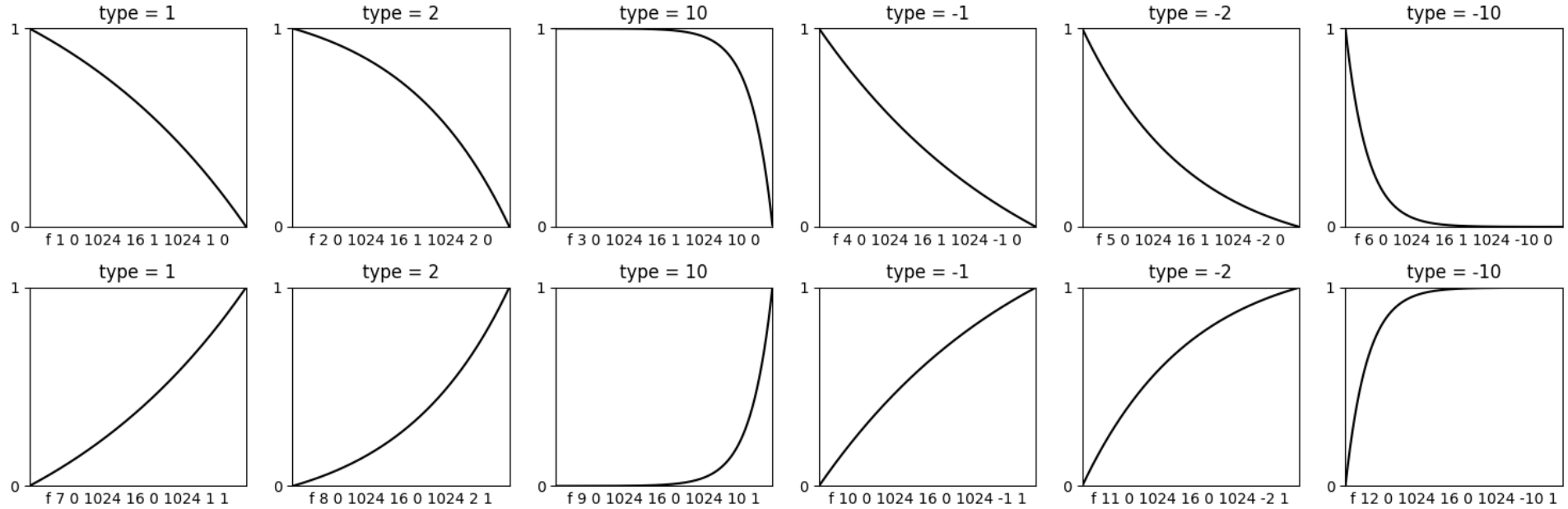
```
instr SimpleSelf

  // say hello
  aEnv = transeg:a(0.2,p3,-3,0)
  aSine = poscil:a(aEnv,500)
  outall(aSine)

  // call another instance of this instrument
  schedule("SimpleSelf",1,1)

endin
schedule("SimpleSelf",0,1)
```



How `aEnv = transeg:a(0.2,p3,-3,0)` works



Examples for the `type` parameter (from Csound Reference Manual)



Now it is your turn!

Exercise 1:
a) Create a random duration of each instance between 1 and 4 seconds.
Use the `random` *opcode at i-rate:*

```
iResult = random:i(iMinimum,iMaximum)
```

b) Set the instance's duration (`p3` *in Csound) to this time by setting:*

```
p3 = ...
```

c) Also set the start parameter for the next instance to this value, so that the next instance will start when this instance has ended.
Remember the syntax of `schedule`*:*

```
schedule("SimpleSelf",iStart,iDuration)
```

received by the called instrument as `p1` `p2` `p3`

My solution (there are others):

```
seed(0)

instr SimpleSelf

  // create random duration between minimum and maximum
  iDur = random:i(1,4)

  //set this instance's duration (p3) to it
  p3 = iDur

  aEnv = transeg:a(0.2,p3,-3,0)
  aSine = poscil:a(aEnv,500)
  outall(aSine)

  // call another instance after this instances has finished
  schedule("SimpleSelf",p3,1)

endin
schedule("SimpleSelf",0,1)
```

Example 03-with-flexible-duration.csd

04

Let us go on to have also flexible / varying pitch!

Exercise 2: Introduce a random pitch in each instance between MIDI notes 70 and 80.
You can use the `random` *opcode again, then use the* `mtof` *opcode to convert MIDI note numbers to frequencies:*

```
iFrequency = mtof:i(iMidiNoteNumber)
```

My solution:

```
seed(0)

instr SimpleSelf

  // set p3 to random duration between minimum and maximum
  p3 = random:i(1,4)

  // create random pitch (in MIDI notes)
  iPitch = random:i(70,80)

  // convert MIDI pitch to frequency (Hz)
  iFreq = mtof:i(iPitch)

  aEnv = transeg:a(0.2,p3,-3,0)
  aSine = poscil:a(aEnv,iFreq)
  outall(aSine)

  // call another instance after this instances has finished
  schedule("SimpleSelf",p3,1)

endin
schedule("SimpleSelf",0,1)
```

Example 04-with-flexible-pitch.csd

And for not insulting our ability to perceive soft and loud sounds:

Exercise 3: Implement for each instance a random volume between -20 and -10 dB. Use the `ampdb` *opcode to convert Decibel to amplitude:*

```
iAmplitude = ampdb:i(iDecibel)
```

My solution:

```
seed(0)

instr SimpleSelf

  // set p3 to random duration between minimum and maximum
  p3 = random:i(1,4)

  // create random pitch (in MIDI notes) and convert
  iFreq = mtof:i(random:i(70,80))

  // create random volume in decibel
  iDb = random:i(-20,-10)

  // convert to amplitude
  iAmp = ampdb:i(iDb)

  aEnv = transeg:a(iAmp,p3,-3,0)
  aSine = poscil:a(aEnv,iFreq)
  outall(aSine)

  // call another instance after this instances has finished
  schedule("SimpleSelf",p3,1)

endin
schedule("SimpleSelf",0,1)
```

Example 05-with-flexible-volume.csd

Let us stop for a while and reflect about where we are.

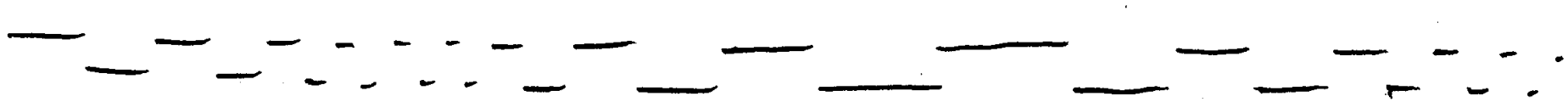
We have only four parameters currently:
A. the DURATION of one event
B. the distance between the events, let's call it START
C. the PITCH
D. the VOLUME

But what an immense world of possibilities opens up when we look closer at them, and let our phantasy play!!!

I would like to show you some of these possibilites. I will also write down some questions which follow for each of these possibilities. (Deciding these questions would be the next step in the composition.) (Or, to have other questions ...)

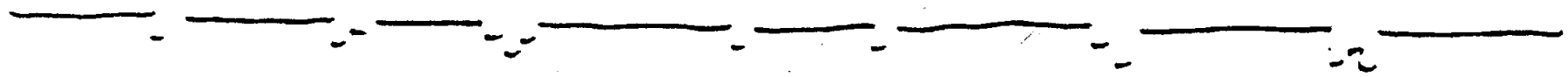
A. Possibilities for DURATION as parameter

A1. Durations become shorter and longer



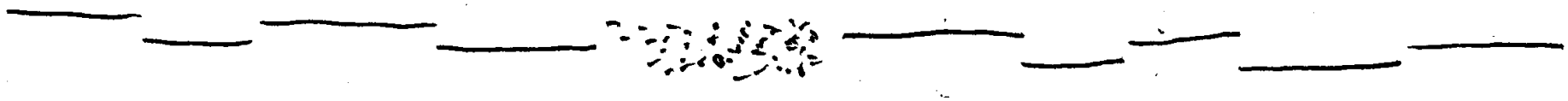
Questions:
- How are the distances between these periods / waves / breathes?
- Are the distances always the same, or getting shorter or longer?
- What are the shortest and longest durations?
- How long stays the quasi-constant fast / slow moment at bottom / top?

A2. One long duration alternates with few short durations



Questions:
- Is there a sequence in the number of short durations, e.g. 1 2 1 3 ... ?
- Are the long durations constant, or do they change?
- Is there a reaction of the long duration to the number of short ones?
- Are the short durations changing, like becoming more, or longer?

A3. Sudden change between groups of long and short durations



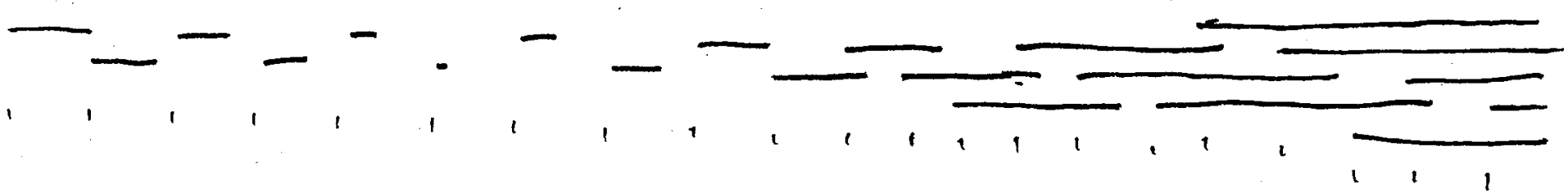
Questions:
- How long are the groups of long and short durations?
- Do they show any development, like the short group disappears?
- Are members of the short durations infecting the long ones?
- What is the rhythm of the short durations?

B. Possibilities for START as parameter

B1. Starts become longer and shorter whilst Durations stay the same

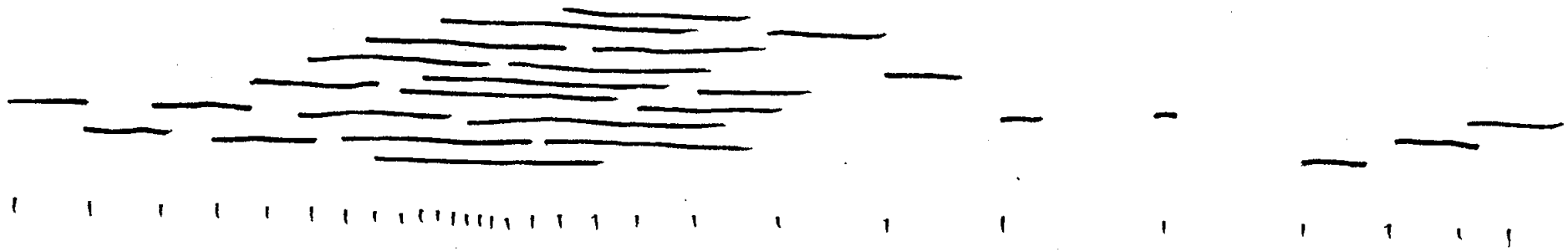
Questions:
- Is this movement periodic? If so, in which time does it happen?
- Is there perhaps kind of a gravitation of the dense structure?
- Is the change perhaps not always reaching the maxima?
- Is the density change perhaps reduced to subtle differences?

B2. Vice versa: Starts almost constant, Durations shorter and longer



Questions:
- How is the ratio of long and short Durations to the Starts?
- Are the Durations getting continuously longer/shorter, or irrgularily?
- Is there a development like "longer and longer" or "shorter and shorter"?
- Is perhaps one long note getting own life and logic, despite the whole?

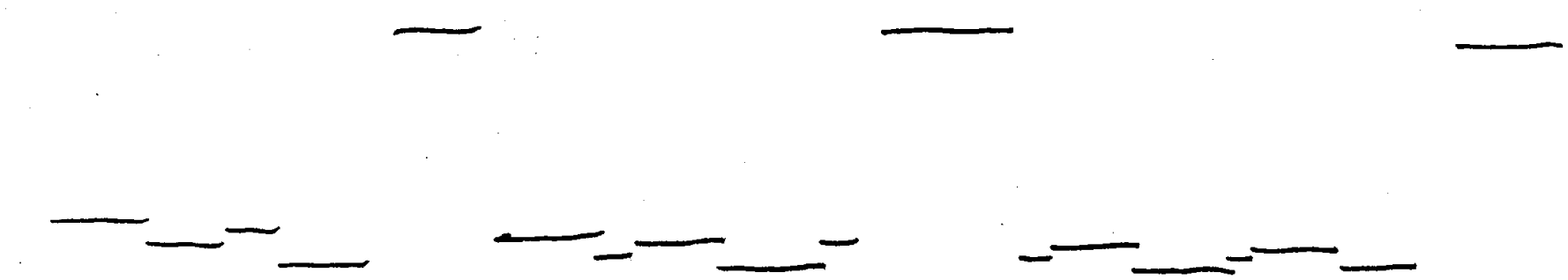
B3. Starts and Durations as reciprocal (Starts short = Durations long etc)



Questions:
- Which time is between the phases "dense" and "empty"?
- How dense is "dense", and how empty is "empty"?
- Is there any development, e.g. will both phases become more similar?
- Perhaps the long notes decide not to disappear ...

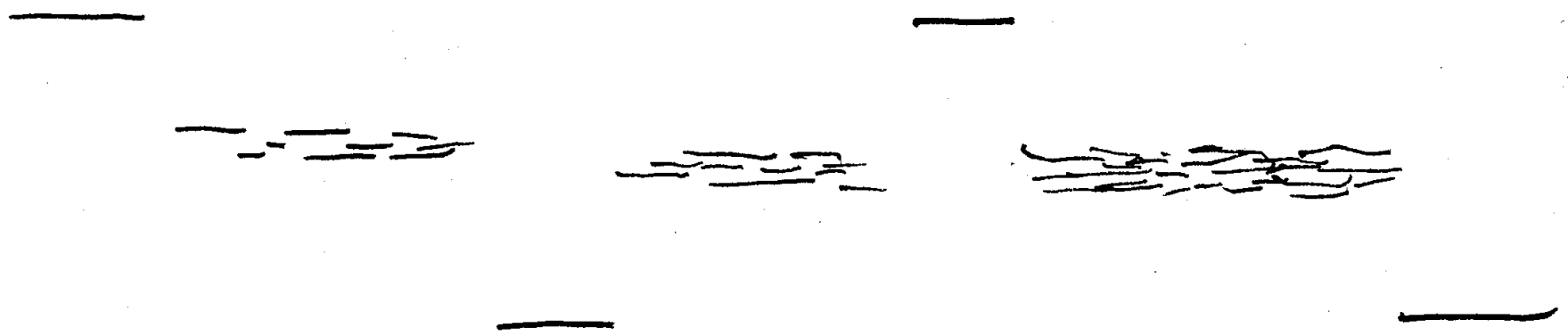
C. Possibilities for PITCH as parameter

C1. Mostly low tones with some exposed high notes



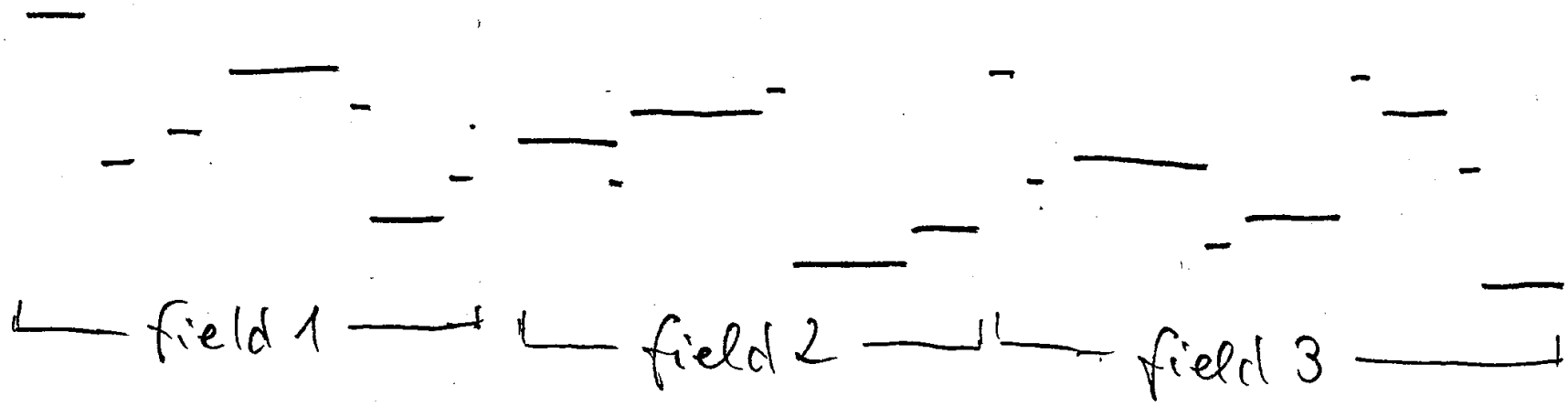
Questions:
- What is the relationship (interval) between the low and the high notes?
- Do the high ones form a sequence? Or do they repeat the same pitch?
- Is there any development like low and high approaching each other?
- Is the distance between the high notes constant or changing?

C2. A mumbling layer in the middle and exposed high and low tones



Questions:
- Do the high and low notes supplement each other, or moving apart?
- Is the middle layer always pausing for the exposed tones?
- Are high and low always alternating, or one becoming more frequent?
- Is the middle layer constant in range, or e.g. approaching the extremes?

C3. A group of pitches forming a harmonic field

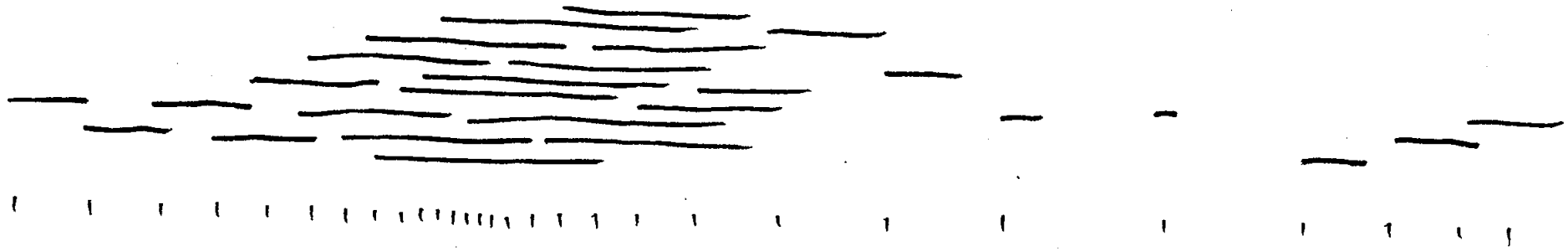


Questions:
- Are there soft transitions between the fields, or abrupt changes?
- Does the change become more often, less often, or is constant?
- Are there repetitions of fields, without repeating the single pitches?
- How is the number of notes in one field?

Sooo .... ---

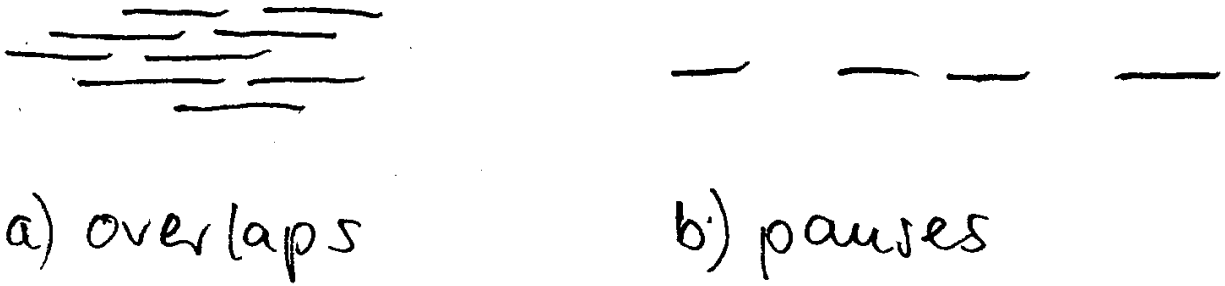
Even with this really simple sound (it is easy to replace it by other sounds) we have so many movements, contents, gestures, expressions.

Let us select one of them (= B3. Starts and Durations as reciprocal) ...



... and then also get to another architecture of our program — an architecture which is in my experience quite well suited for many cases.

Exercise 4:
a) Change the code so that the tones are overlapping each other.
b) If you succeeded, change the values so that there are pauses between the tones.



My solution:

```
seed(0)

instr SimpleSelf

  // set p3 to random duration between minimum and maximum
  p3 = random:i(1,4)

  // set start to another random range
  // if smaller than the duration range, overlaps will occur
  // if larger than the duration range, pauses will occur
  iStart = random:i(0.5,2) //compare to (2,8)

  // create random pitch (in MIDI notes) and convert
  iFreq = mtof:i(random:i(70,80))

  // create random volume in decibel
  iDb = random:i(-20,-10)

  // convert to amplitude
  iAmp = ampdb:i(iDb)

  aEnv = transeg:a(iAmp,p3,-3,0)
  aSine = poscil:a(aEnv,iFreq)
  outall(aSine)

  // call another instance after this instances has finished
  schedule("SimpleSelf",iStart,1)

endin
schedule("SimpleSelf",0,1)
```

07

Now let us make a more practical change:

We currently have four parameters

1. minimum duration
2. maximum duration
3. minimum start (= distance for the next event)
4. maximum start

I'd like to reduce them to two parameters:

1. mean duration
2. mean start

We can then calculate the minimum as half of the mean, and the maximum as twice of the mean. (This is mathematically not quite correct, but it works musically ... =)

Exercise 5: Change the code so that you calculate the random duration of a tone by the mean duration as input, and the random start by the mean start as input.

minimum maximum

e.g. 0.5 2

(2 values as input)

change to 1 as mean value

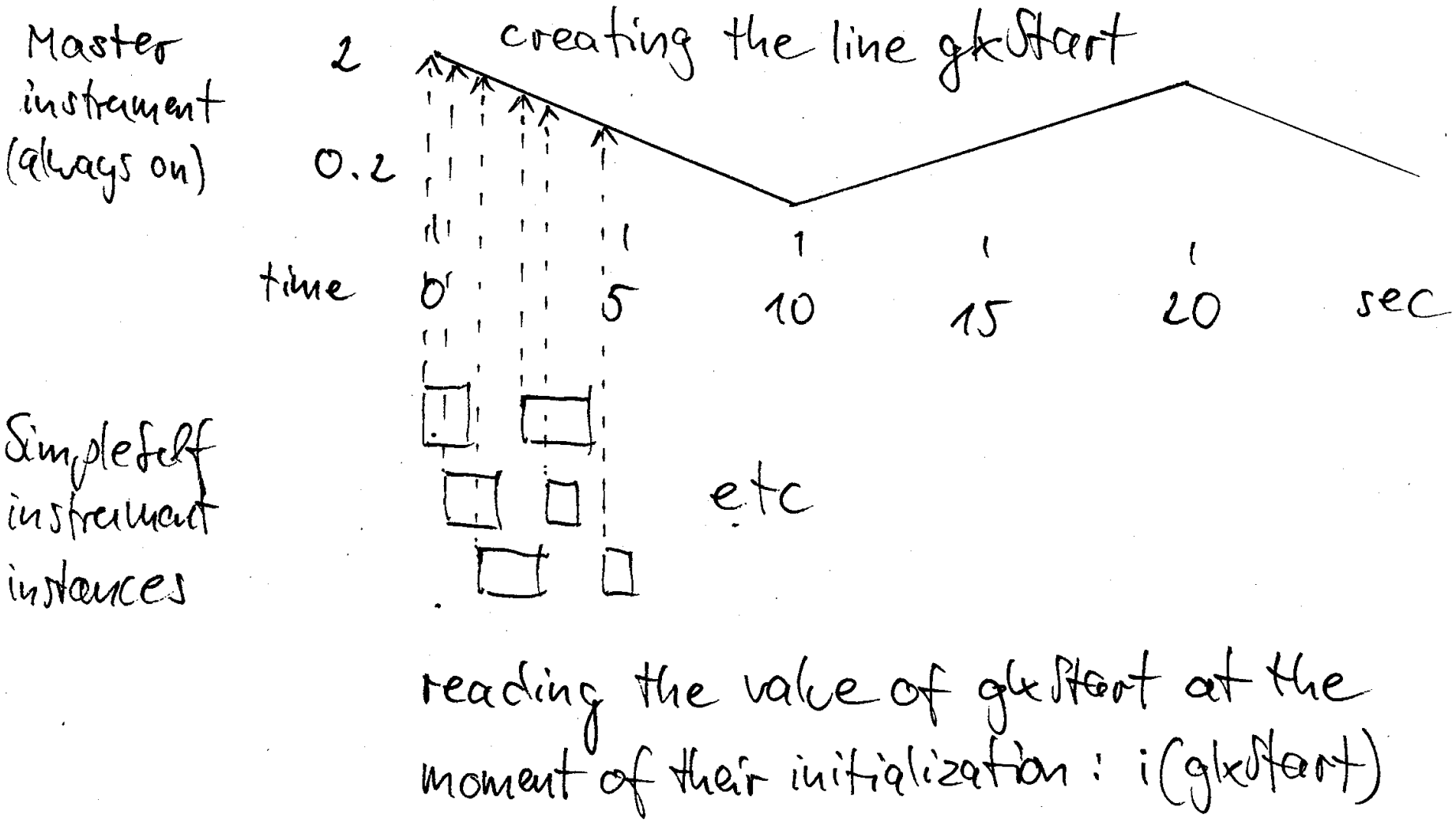
*0.5 *2 apply

0.5 2 to get

= minimum maximum

```
instr SimpleSelf

  // set a mean value for the duration
  iMeanDuration = 2

  // minimum for random selection is mean/2, maximum is mean*2
  p3 = random:i(iMeanDuration/2,iMeanDuration*2)

  // set a mean value for start
  iMeanStart = 1 //compare to 3

  // again random selection based on this mean value
  iStart = random:i(iMeanStart/2,iMeanStart*2)

  // create random pitch (in MIDI notes) and convert
  iFreq = mtof:i(random:i(70,80))

  // create random volume in decibel
  iDb = random:i(-20,-10)

  // convert to amplitude
  iAmp = ampdb:i(iDb)

  aEnv = transeg:a(iAmp,p3,-3,0)
  aSine = poscil:a(aEnv,iFreq)
  outall(aSine)

  // call another instance after this instances has finished
  schedule("SimpleSelf",iStart,1)

endin
schedule("SimpleSelf",0,1)
```

Implementing density changes by a master instrument.



Let us decide that:

1. Every 20 seconds the mean START time moves from 2 sec to 0.2 seconds and back.
We can achieve this by using the `loopseg` opcode:

```
gkMeanStart = loopseg:k(1/20, 0, 0, 2, 1, 0.2, 1)
```

frequency of the loop: 1/20 Hz = 20 sec

please ignore

start value

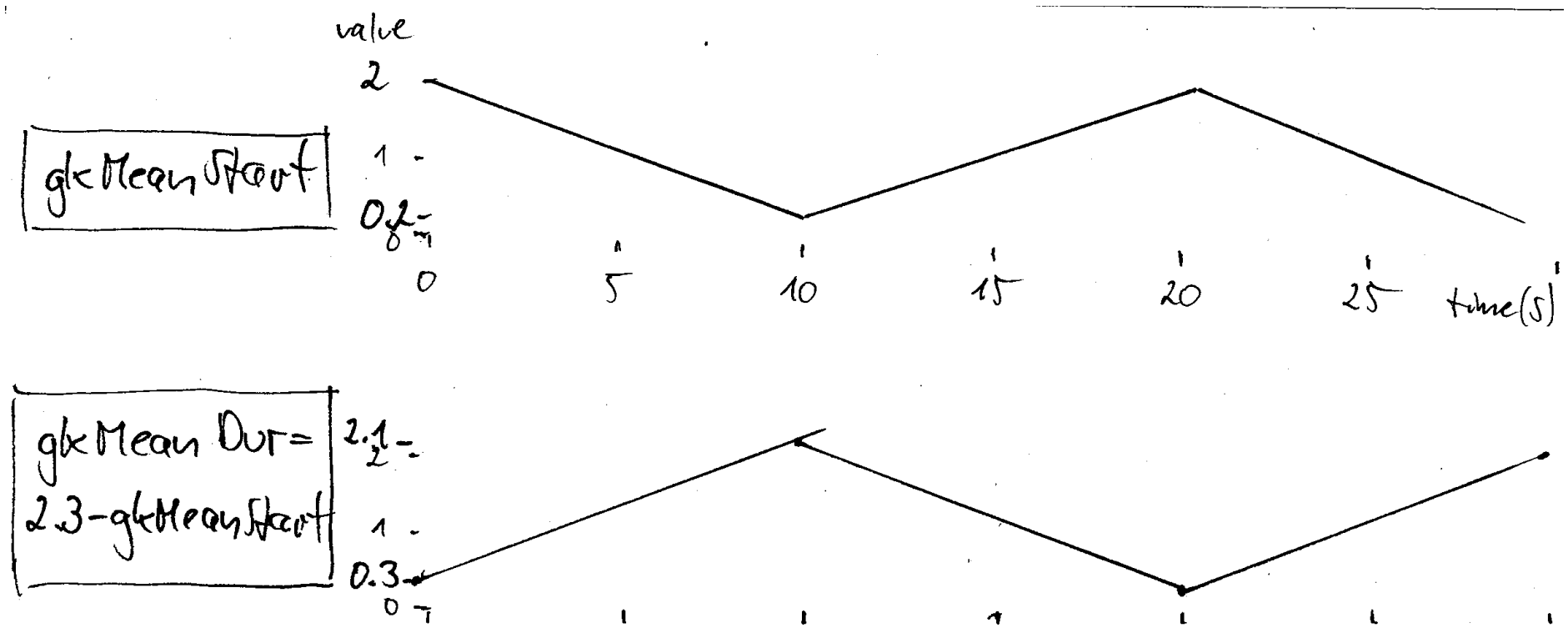
relative time to

target value

relative time again

2. The mean DURATION of a tone is reciprocally to the start. We can achieve this by subtracting the START from a fixed value.

```
gkMeanStart = loopseg:k(1/20,0,0, 2,1, 0.2,1)
```




```
gkMeanDur = 2.3 - gkMeanStart
```

```
instr Master //shapes the big movements of parameters

  // apply complementary movement for duration and start
  gkMeanStart = loopseg:k(1/20,0,0,2,1,0.2,1)
  ;gkMeanSart = randomi:k(0.2,2,1/10,3) // uncomment this to try as alternative
  gkMeanDur = 2.3 - gkMeanStart
  // have the first schedule here to avoid zero start values
  schedule("SimpleSelf",1/kr,1)

endin
schedule("Master",0,-1)


instr SimpleSelf

  // get a mean value for the duration as current state of gkMeanDur
  iMeanDuration = i(gkMeanDur)

  // get a mean value for start as current state of gkMeanStart
  iMeanStart = i(gkMeanStart)

  // minimum for random selection is mean/2, maximum is mean*2
  p3 = random:i(iMeanDuration/2,iMeanDuration*2)

  // again random selection based on this mean value
  iStart = random:i(iMeanStart/2,iMeanStart*2)

  // create random pitch (in MIDI notes) and convert
  iFreq = mtof:i(random:i(70,80))

  // create random volume in decibel
  iDb = random:i(-20,-10)

  // convert to amplitude
  iAmp = ampdb:i(iDb)

  aEnv = transeg:a(iAmp,p3,-3,0)
  aSine = poscil:a(aEnv,iFreq)
  outall(aSine)

  // call another instance after this instances has finished
  schedule("SimpleSelf",iStart,1)

endin
```

To summarize: The *Master* instrument creates a line, and each instance of the *SimpleSelf* instrument reads the value of this line at its start.

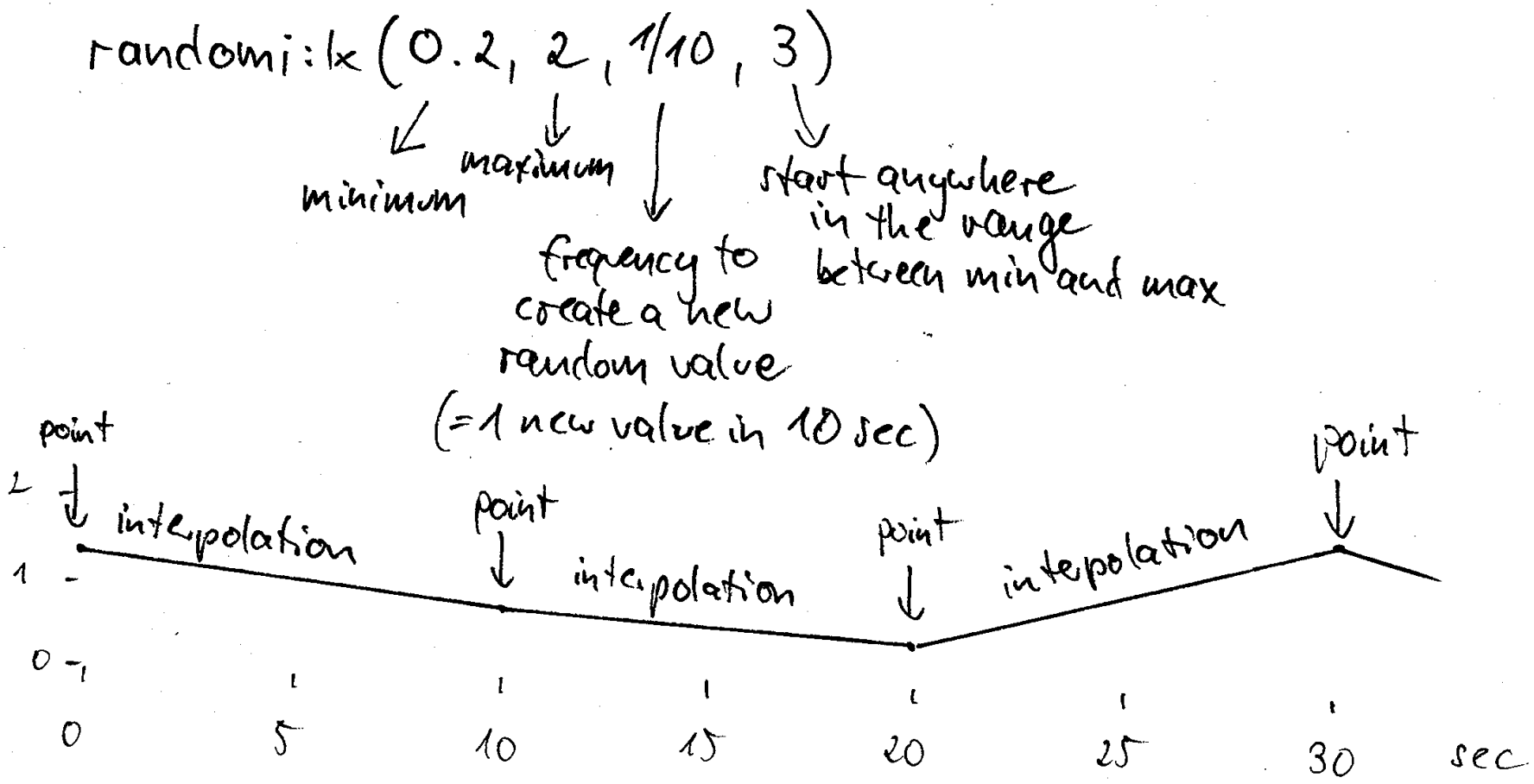
To make the line readable for any instrument, the line must become a *global* variable (starting in Csound 6 with `g`).

```
gkMeanStart = loopseg:k(1/20,0,0, 2,1, 0.2,1)
```

To get the value (which is always changing, so for Csound it is a `k`-rate value) at start of the instrument instance (called "initialization"), we use the `i(kValue)` facility:

```
iMeanStart = i(gkMeanStart)
```

As an alternative to these loops, we can use the `randomi` (= random with interpolations between values) opcode to create random values which change in a certain range, but in a way which we cannot predict.



If there is time left:

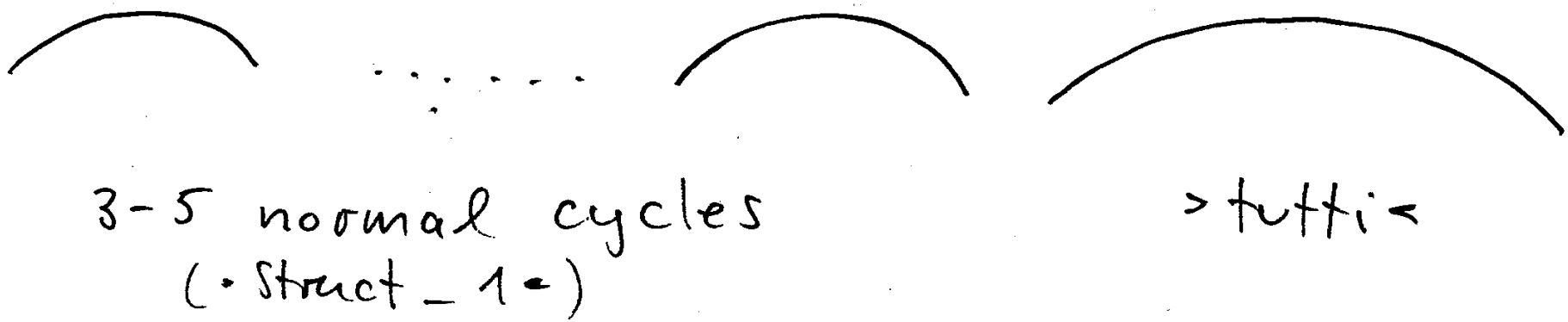
Last exercise: Change the values for mean start and mean duration, and judge about the musical result.

II. LOOKING AT A REAL MUSICAL EXAMPLE

My sound installation *Gerettet! [Saved!]* (2024)
https://joachimheintz.net/gerettet.html

Explain the main idea (briefly!).

The FORM is an ongoing sequence of 3-5 parts ("cycles") followed by one part in which all pipes / loudspeakers are active ("tutti").



```
// ruft eine struktur
instr CallCircs

  kTime init 0
  kCircCount init 0
  kCircNumAlle init NmRndInt:i(3,5)

  if (kTime <= 0) then

    // normaler durchlauf
    if (kCircCount < kCircNumAlle) then

      // auch dies sollte wohl eher array mit auswahl sein
      kDurPlapp = random:k(90,120)
      kDurSolo = random:k(50,70)
      kDurPause = random:k(30,40) //die reals pausendauer ist kürzer wegen der langen töne beim solo
      schedulek("Struk_1",0,kDurPlapp,kDurSolo) //solo wird aus Struk_1 heraus gerufen
      kTime = kDurPlapp+kDurSolo+kDurPause
      kCircCount += 1

    // "alle" nach 3-5 dieser durchläufe
    else

      // die dauer der drei teile hier berechnen wegen der gesamtdauer
      kAlle_1_Dur = random:k(90,120)
      kAlle_2_Dur = random:k(60,90)
      kAlle_3_Dur = random:k(120,150)
      schedulek("Alle",0,1,kAlle_1_Dur,kAlle_2_Dur,kAlle_3_Dur)
      kDurPause = random:k(40,50)
      kTime = kAlle_1_Dur+kAlle_2_Dur+kAlle_3_Dur+kDurPause
      kCircCount = 0
      kCircNumAlle = NmRndInt:k(3,5)

    endif

  endif

  kTime -= 1/kr
  ;printk2 kCircCount
```

One of these normal cycles consists of three phases:
- at first all 5 active pipes have a smalltalk ("plappern")
- then one of them gets in another mode ("solo"); the others silence
- then a long pause, before the next cycle begins

Play one cycle and comment.

Scroll through the code of the *Struk_1* instrument to point to some details.

If time, have a look at *Alle* and the rest of the code.

III. CSOUND 7 PREVIEW

1. Explicit types

```
instr Locals

  freq:i = 500
  amp:i = 0.2

  sine:a = poscil(amp,freq)
  outall(sine)

endin

instr Globals

  dry@global:a init 0

  dry += diskin:a("fox.wav") / 3
  dry += pinkish(.05)

  schedule(GlobalsReverb,0,p3+3)

endin

instr GlobalsReverb

  wet:a = reverb2(dry,2,.5)
  outall(dry/5+wet/2)
  dry = 0

endin
```

2. Arrays

```
instr ArrayDef

  freq:i[] = [500,600,700]
  ser:i[] = [1 ... 10] //shortcut
  files:S[] = ["bla.wav","blu.flac"]
  sound:a[] = diskin("fox.wav")

endin

instr ArrayExample

  // two random k-rate envelopes in an array
  env:k[] = [randomi:k(-20,0,15,3),randomi:k(-20,0,15,3)]

  // two mono sound streams
  a1 = diskin:a("fox.wav",randomi:k(.7,1.1,1,3),0,1)
  a2 = diskin:a("ClassGuitMono.flac",randomi:k(.8,1.3,1,3),0,1)

  // apply envelopes and put into array
  audio:a[] = [a1*ampdb(env[0]), a2*ampdb(env[1])]

  // sometimes swap the audio channels (clicks may occur)
  if (randomh:k(0,2,1,3) > 1) then
    audio = [audio[1], audio[0]]
  endif

  // output
  out(audio)

endin
```

3. For-loop

```
instr I_Loop_Simple

  // num is not specified: i-rate because of the i-rate array
  for num in [1,2,3,5,8] do
    print(num)
  od

endin

instr I_Loop_With_Counter

  // the same but with loop counter
  for num,i in [1,2,3,5,8] do
    print(num,i)
  od

endin

instr Short

  strt:i init 0
  for freq in [200 ... 2000, 100] do
    schedule(PlaySine,strt,5-strt,freq)
    strt += 1/8
  od

endin

instr PlaySine

  env:a = transeg(.1,p3,-3,0)
  outall(poscil:a(env,p4))

endin
```

4. Here I transform the last coding example (08) to Csound 7 code and replace the simple sines by a richer sound.

```
instr Master //shapes the big movements of parameters

  // apply complementary movement for duration and start
  MeanStart@global:k = loopseg(1/20,0,0,2,1,0.2,1)
  ;MeanStart@global:k = randomi(0.2,2,1/10,3)
  MeanDur@global:k = 2.3 - MeanStart
  // have the first schedule here to avoid zero start values
  schedule(SimpleSelf,1/kr,1) // note that we call instrument as variable

endin
schedule(Master,0,-1)


instr SimpleSelf

  // get a mean value for the duration as current state of MeanDur
  mean_duration:i = i(MeanDur)

  // get a mean value for start as current state of MeanStart
  mean_start:i = i(MeanStart)

  // minimum for random selection is mean/2, maximum is mean*2
  p3 = random:i(mean_duration/2,mean_duration*2)
```

```
  // again random selection based on this mean value
  start:i = random(mean_start/2,mean_start*2)

  // create random pitch (in MIDI notes) and convert
  freq:i = mtof(random:i(70,80))

  // create random volume in decibel and convert
  amp:i = ampdb(random:i(-20,-10))

  // call as many instances of subinstrument as there are partials
  for partial in [1,2.78,5.18,8.16,11.66,15.64,19.99] do
    schedule(Partial,0,p3,amp,freq*partial,random:i(500,1000),random:i(0,1))
  od

  // call another instance after this instances has finished
  schedule(SimpleSelf,start,1)

endin

instr Partial

  amp:i,freq:i,q:i,pan:i = p4,p5,p6,p7
  sound:a = mode(mpulse(amp,p3),freq,q)
  left:a,right:a = pan2(linenr:a(sound,0,1,.01),pan)
  out(left,right)

endin
```

And that's ...

... The End!