MUSIC WITH PHARO

- Pharo as an OSC/MIDI performer/sequencer/controller.
- Pharo as a language for live-coding/ programming -music-on-the-fly.
- Pharo does not generate any sound (for now).
- Pharo as a client for a MIDI instrument (internal/external) or for an OSC audio server (i.e. SuperCollider, PureData, ChucK, Max/MSP, Kyma)

OSC

https://github.com/Ducasse/OSC

OSC implementation

LiveCoding Package

https://github.com/lucretiomsp/PharoLiveCoding

Domain Specific Dialect

Pharo-Sound

https://github.com/pharo-contributions/pharo-sound

MIDI connections implementations

MUSIC WITH PHARO

Pharo-Sound

https://github.com/pharo-contributions/pharo-sound

MIDI connections implementations

PortMidi C external cross-platform library implemented in Pharo using the Unified FFI package by Antoine Delaby and Santiago Bragagnolo

Full MIDI functionalities in Pharo

Connect to external MIDI hardware or to a virtual MIDI bus, IAC Driver on Mac or MIDI Yoke* on Windows (*not native)

It is used by the LiveCoding Package for MIDI connections

Transform Pharo into a MIDI sequencer

Allows to create custom GUI MIDI controllers

The **MIDI** (*Musical Instrument Digital Interface*) protocol, developed collaboratively by Dave Smith, Chet Wood, and Ikutaro Kakehashi in 1983, is a standardised communication protocol used in electronic music equipment. It facilitates the transmission of musical information like notes, control parameters, and timing data between different devices such as synthesizers, computers, and controllers.

MUSIC WITH PHARO

OSC

https://github.com/Ducasse/OSC

OSC implementation

Open Sound Control implementation for Pharo, Originally developed and license under MIT by Markus Gaelli and then Simon Holland. Now cleaned and maintained by S. Ducasse.

Necessary dependency of the *LiveCoding Package.*

OSC OpenSoundControl (OSC) is a data encoding for realtime message communication among applications and hardware. Developed by Matt Wright and Adrian Freed at CNMAT

Send and receive OSC messages via UDP

Highly accurate, lightweight, low latency, widespread

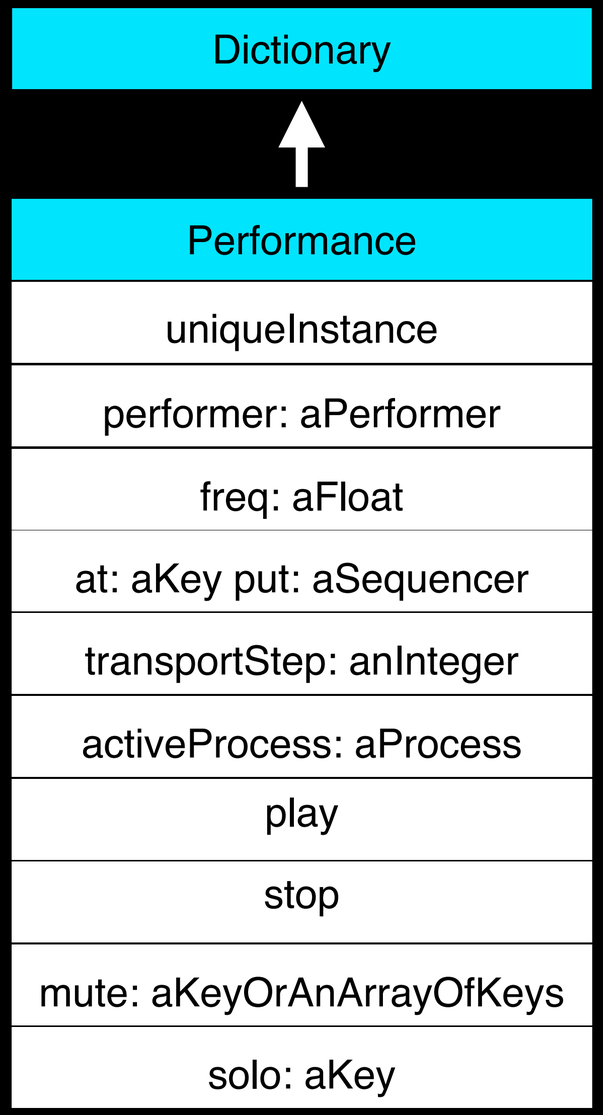
MUSIC WITH PHARO

LiveCoding Package

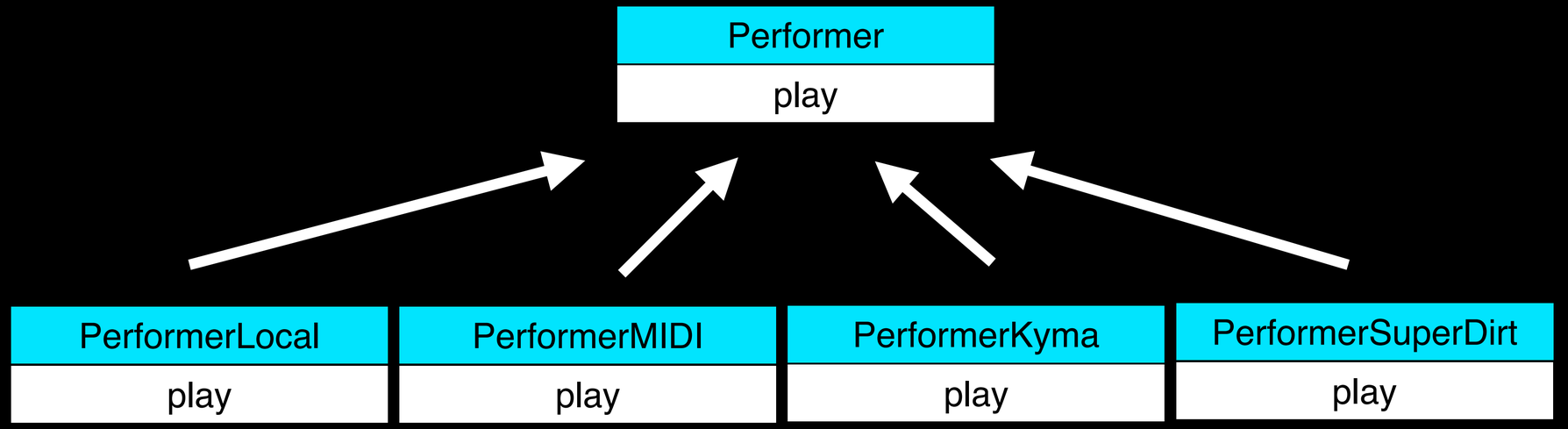
https://github.com/lucretiomsp/PharoLiveCoding

Domain Specific Dialect

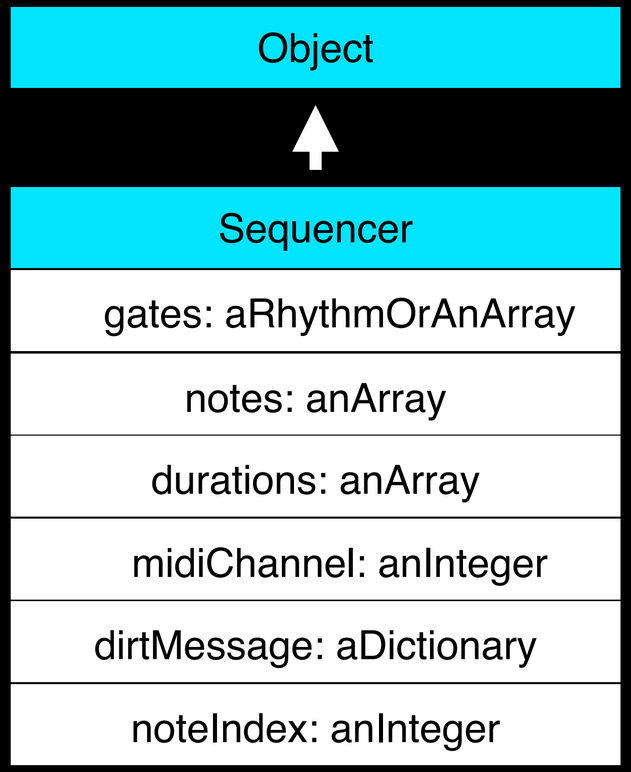
- Play sounds from an audio server or MIDI instrument
- Create, transform and manipulate Rhythms
- Create, transform and manipulate melodies, chords and arpeggio
- Create custom OSC controllers
- Can also be used and expanded to be used (via OSC or MIDI) with software for visual arts such as Processing, Touch Designer, Hydra

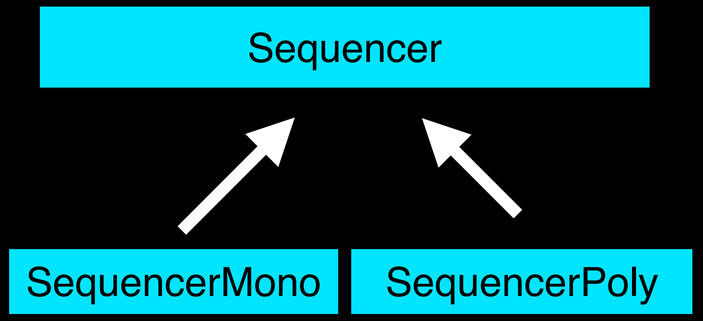


- The Performance class is a Singleton.
- Performance is a subclass of Dictionary.
- A Performer must be assigned to the Performance; the Performer selects the audio backend.
- The *play* method starts the Performance, and increments the transportStep every *freq* seconds (0.125 s at 120 bpm).



MUSIC WITH PHARO





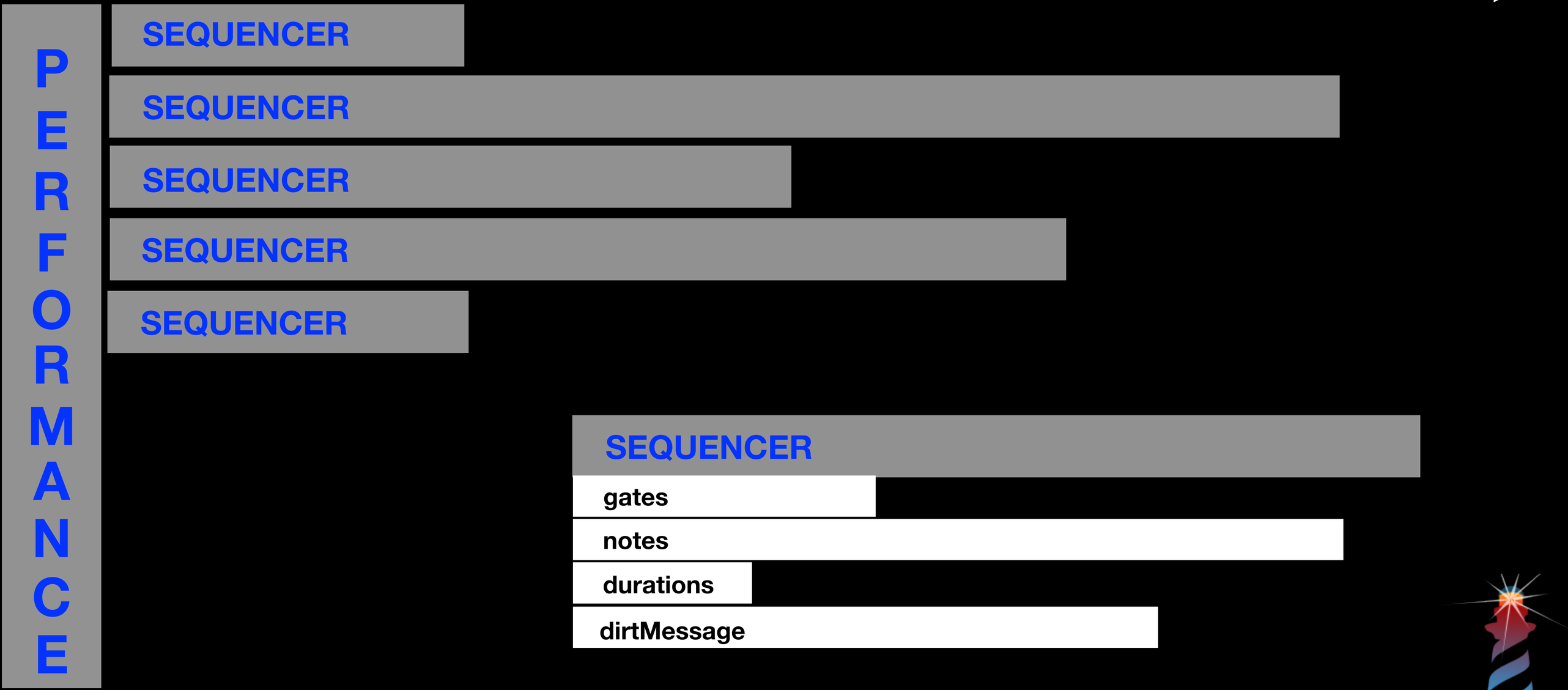
- A SequencerMono sends out a note at once every time there is a gate.
- A SequencerPoly sends out chords

- The contents of a Sequencer can have different size, i.e. lengths, for example:
  Sequencer new gates: 16 semiquavers; notes: #(32 48 51); duration: #(0.2 0.3 0.5 1 0.4 0.6).

aSequencer to: #aKey = Performance uniqueInstance at: #aKey put: aSequencer

MUSIC WITH PHARO

transportStep increments of 1 every **Performance** uniqueInstance freq (seconds)

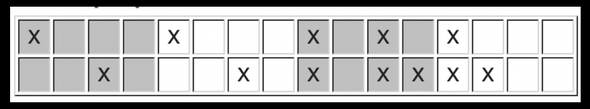


MUSIC WITH PHARO

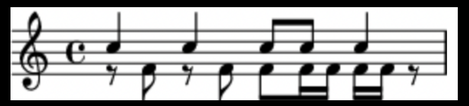
Performance, Sequencers, how does them work?

- Sequencers are inspired by traditional hardware sequencers, where triggers (~noteOn) events are notated in Time Unit Box System (TUBS).

TUBS



Traditional



LiveCoding Dialect

```
#( 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 ) asSeq
```

- Performance can be thought as a multitrack player for Sequencers.
- Sequencer are filled with parameters.
- A Performer must be assigned to the Performance.

```
p := Performance uniqueInstance.
p performer: PerformerLocal new.
p freq: 136 bpm. "change the performances speed"
#( 1 0 0 0 1 1 0 0 1 0)  asSeq notes: #(32 37 38) to: #bass. “create a Sequencer and assign it to the performance "
p play. "start the performance".
p stop. "stop the performance"
```

```
Performance >>> play
 self performer play
```

Quick MIDI connections

- Get a list of all the available devices, create a new instance of MIDISender, open the MIDISender on the selected device

```
PortMidi traceAllDevices.
mout := MIDISender new.
mout openWithDevice: 5.
```

- To play a MIDI performance, a MIDI channel must be specified for each sequencer sending the message **midiCh:**

```
16 downbeats midiCh: 9 to: #kick.
16 upbeats midiCh: 12 to: #hat.
16 rumba, 16 banda midiCh:  3 to: #tom.
16 semiquavers notes: #(36 48 56 51 62 74 32) midiCh: 1 to: #bass.
```

- If MIDI notes got stuck / keep playing!

```
mout allNotesOff
```

MUSIC WITH PHARO

How to create a Sequencer with the LiveCoding Dialect

- By sending messages (named as a rhythm) to integers, you create a Sequencer with the size of the integer, for example:
  - 16 downbeats
  - 32 rumba
  - 4 breves
  - 8 semiquavers

- By sending the hexBeat message to a string containing natural numbers and characters $A - $F, you create a Sequencer with the corresponding hexBeat (https://kunstmusik.github.io/learn-hex-beats/), for example:
  - '8888' hexBeat
  - '9090' hexBeat
  - '80802ACDE9347B' hexBeat

- By sending the *euclidean* message to an array containing the onsets/triggers and the number of step,s you create a Sequencer with the corresponding euclidean rhythm (http://cgm.cs.mcgill.ca/~godfried/publications/banff.pdf), for example:
  - #(3 8) euclidean
  - #(5 16) euclidean
  - #(7 12) euclidean

MUSIC WITH PHARO

Principles

- **Iconicity**

Written code should resemble what we hear

```
16 upbeats
```

> *"Iconicity is the relation of similarity between two aspects of a sign: its form and its meaning. An iconic sign is a sign that in some way resembles its meaning"*.(Meir)

- **Economy**

Type less!

```
#(60 63 67) + 16
```

> - *"The only primary principle of every human action, including verbal communication, is the expenditure of the least amount of effort to accomplish a task"*. (George Zipf)

- **Polysemy**

Many ways to do the same thing

```
8 downbeats
#(1 0 0 0 1 0 0 0) asSeq
'88' hexBeat
```

Chord/Scale/Arpeggios

- 18 Scales from all around the world, sending the name of a scale to the Scale class returns an array with the intervals of the scale
  - Scale sakura.
  - Scale flamenco
- 47 types of chords, can be parsed into arrays of intervals sending the message chordsToArrays to a string containing the chords separated by a space in the form:
  `root[+ # for sharp]-typeofchord`
  - 4 breves chords: 'c-minor d-minor e-minor13 e#-sus4'; to: #superpiano.
  - 16 semiquavers arpeggiate: 'f-minor7 d-minor13 g-sus2'; to: #superchip.

MUSIC WITH PHARO

Stochastic methods of the LiveCoding Dialect

- Random trigs
  - 64 randomTrigs.
  - 128 randomTrigsWithProbability: 75

- Random trigs and random samples from a folder
  - 64 randomSamplesFromFolder: ’cpu’.

- Random note from a scale spread over 2 octaves
  - 64 randomNotesFrom: Scale gypsy octaves:2

- Random samples from a SuperDirt sample folder
  - 128 quavers sound: 'superchip'

MUSIC WITH PHARO

Transformative methods of the LiveCoding Dialect

- Join sequencers (with comma):
- Offset sequencers (shift left with negative value, shifts right with positive value)

16 downbeats, '000F' hexBeat, 16 upbeats.

16 upbeats offset: 1.

- Reverse sequencers

'000F' hexBeat reverse.

- Flip gates/rests

#(5 12) euclidean flip.

- Repeats elements

4 copiesOfEach: #(-12 -10 -9 -8). ⟶ ^ #(-12 -12 -12 -12 -10 -10 -10 -10 -9 -9 -9 -9 -8 -8 -8 -8)

- “Transpose” elements

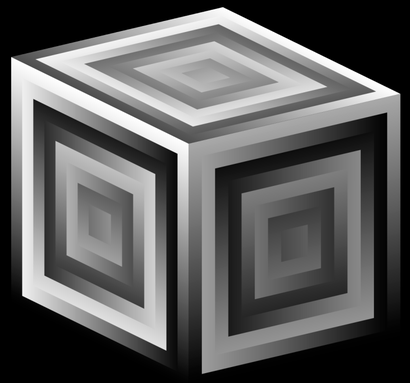
#(-12 -10 -9 -8) + 2. ⟶ ^ #(-10 -8 -7 -6)

MUSIC WITH PHARO

Pharo to SuperDirt

- **Now it is possible to use Pharo as a sequencer for the SuperDirt audio engine for SuperCollider**

Performance uniqueInstance performer: **PerformerSuperDirt** new



- SuperCollider is an audio server, programming language, and IDE for sound synthesis and algorithmic composition. Originally released by James McCartney for real-time audio synthesis and logarithmic composition.
- The language combines the object-oriented structure of **Smalltalk** and features from functional programming language with a C-like syntax.
- SuperDirt is the SuperCollider implementation of the Dirt sampler, originally designed for the TidalCycles environment. It is a framework for playing samples and synths, controllable over the Open Sound Control protocol, and locally from the SC language. SuperDirt is also used by Sardine, a live coding environment for Python 3.10+ created by Raphaël Forment.

- From Pharo we send OSCBundles with timestamps(NTP timestamps), and the SuperCollider audio server takes care of the scheduling of the messages.
- OSCBundles are sent with a latency of 50 milliseconds (a little latency is require by the sc server for proper scheduling).
- The Sequencers instance variable called ***dirtMessage*** is a Dictionary that contains associations keys->values that are send to SuperCollider into an OSC Bundle; a quick reference to the keys and values can be found here:

  https://tidalcycles.org/docs/reference/synthesizers https://tidalcycles.org/docs/reference/audio_effects

MUSIC WITH PHARO

Syntax for SuperDirt

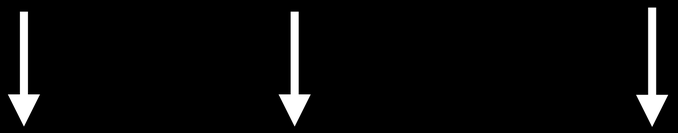
- 'cp ~ sd ht lt ~ ~ mt' forDirt to: #groove.
- 16 downbeats sound: 'bd:3' ; to: #kick.
- #(5 12) euclidean flip to: #sd.
- 16 bomba sound: 'mt'; dirtNotes: #(3 2 9 8); to: #toms; add: 'squiz'->#(2 3 7).
- 64 randomSamplesFromFolder: ’cpu’.
- #(5 12) euclidean randomSamplesFromFolder: ‘mt’.
- 4 breves sound: 'supersaw' dirtNotes: #(0 1 2 3); gain: 0.7; to: #piano .

- ‘hoover' once.
- ‘hoover' onceAtSpeed: 2.

The dirtMessage

- The instanceVariable of the Sequencers called dirtMessage contains an infrastructure for the messages sent in the OSCBundle to the SuperDirt audio engine, for example:

16 tresillo sound: 'clap'; dirtNotes: #(0 3 2 1 5); squiz:#(2 1 0.9)



's'->#('clap') 'n'->#(0 3 2 1 5) 'squiz'->#(2 1 0.9)

```
Sequencer >>> playFullDirtEventAt: anIndex

    "sends a mesaage to SuperDirt with all the desired OSC arguments and values"
| message dur stepDuration|
stepDuration := Performance uniqueInstance freq.
message := OrderedCollection new.
message add: '/dirt/play'.                    OSCMessage address
dur := self durations asDirtArray  wrap: anIndex .
message add: 'delta'; add: stepDuration * dur.
dirtMessage keysAndValuesDo:  [ :key :value | message add: key; add: (value asDirtArray  wrap: anIndex ) ].
(OSCBundle for: {  OSCMessage for: message  } )  sendToAddressString: '127.0.0.1' port: 57120.
```

```
Array >> wrap: anInteger [
“inspired by Cmajor language wrap<int> always wrap the index of the array to the array size"

    | result |
    result := self at: anInteger - 1 % self size + 1.
    ^ result
```

MUSIC WITH PHARO

AND WHAT ABOUT THE FUTURE?

- More and more test!
- Better and exhaustive documentation and tutorials!
- A custom Playground.
- More GUI elements!
- Cyclical patterns as in Tidal Cycles (https://tidalcycles.org) + a parser for Tidal Cycles' *mini notation*.
- In the web browser with PharoJS?
- Generate sound and create instruments and effects inside Pharo embedding the FAUST compiler (https://faustcloud.grame.fr)