An introduction to TidalCycles and live-coding

Agathe Herrou

December 1st, 2019

1. Live-coding
   - History
   - Culture
   - Patterns

2. TidalCycles
   - Technical aspects
     - Haskell
     - TidalCycles environment
   - The language
     - Cycles
     - Transforming patterns
     - Patterns typology
     - Random
     - Samples manipulation

Table of contents

A little history...

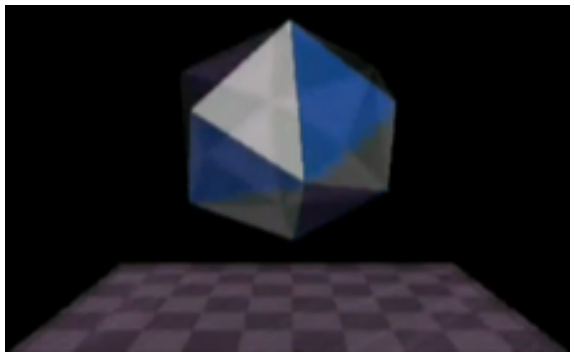
- Creative coding: create art using code

A little history...

- Creative coding: create art using code
- Appeared in the early 2000s

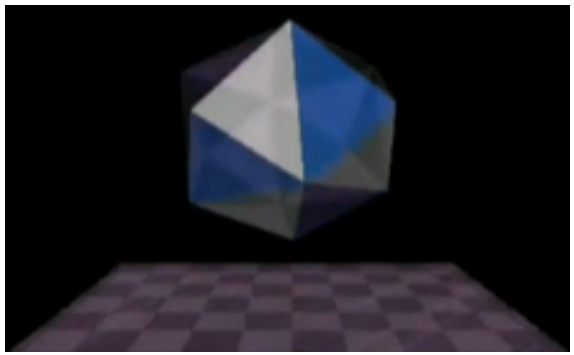
A little history...

- Creative coding: create art using code
- Appeared in the early 2000s
- Demoscene



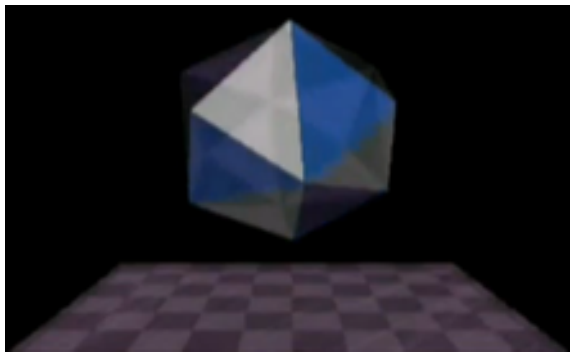
A little history...

- Creative coding: create art using code
- Appeared in the early 2000s
- Demoscene
- Creation of TOPLAP in 2004



A little history...

- Creative coding: create art using code
- Appeared in the early 2000s
- Demoscene
- Creation of TOPLAP in 2004
- Algorave(s)



Culture

- Strong improvisation culture
- Openness to errors and unexpected
- Free software

Patterns

Patterns form the basis of this conception of music

Patterns

Patterns form the basis of this conception of music
Manipulation of musical patterns [1]:

- transposition
- inversion
- rotation
- phase offset
- rescaling
- interpolation
- extrapolation
- fragmentation
- substitution
- combination
- sequencing
- repetition
- ...

Table of contents

Haskell basics

- Functional programming language

Haskell basics

- Functional programming language
- Function application: `f x`

Haskell basics

- Functional programming language
- Function application: `f x`
- Multiple arguments: `f x y`

Haskell basics

- Functional programming language
- Function application: `f x`
- Multiple arguments: `f x y`
- Function composition: `g (f x)`

Haskell basics

- Functional programming language
- Function application: `f x`
- Multiple arguments: `f x y`
- Function composition: `g (f x)` which can also be written `g $ f x`

TidalCycles environment

Tidal: patterns → SuperCollider: sound synthesis

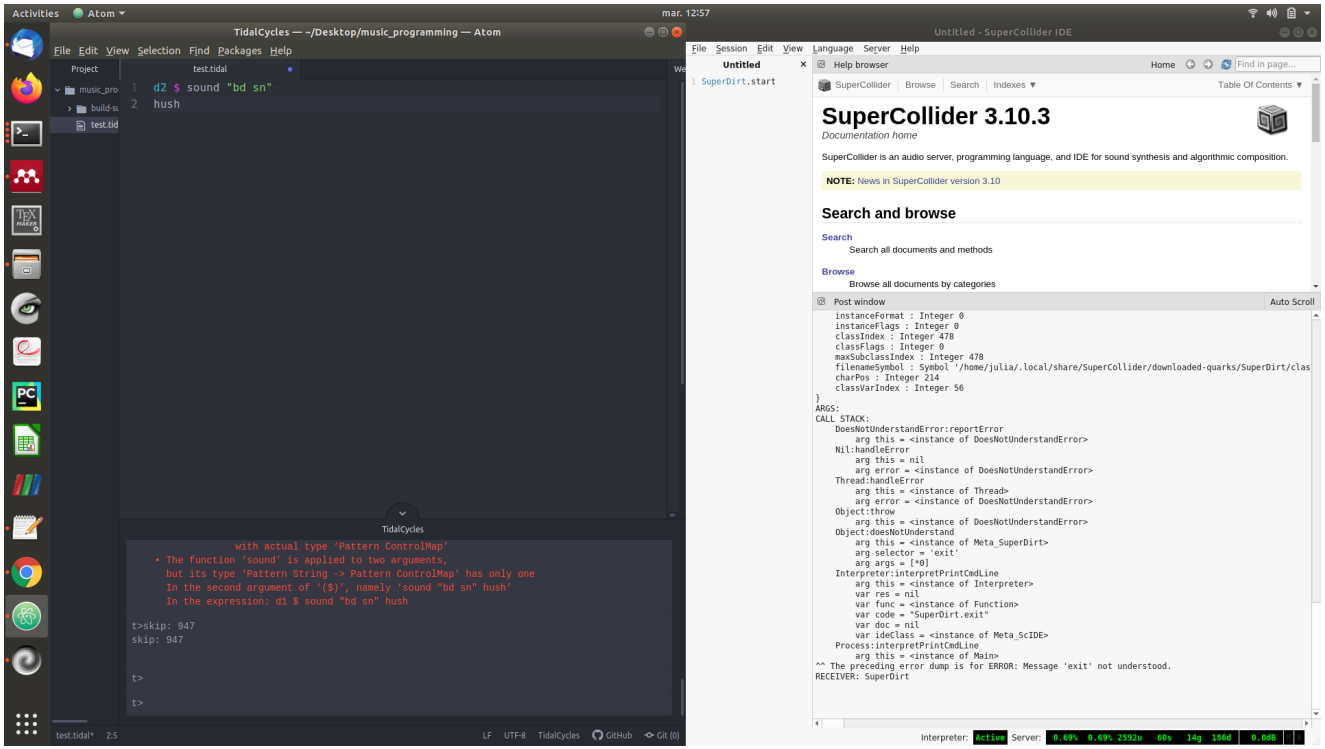


Figure: Setup example

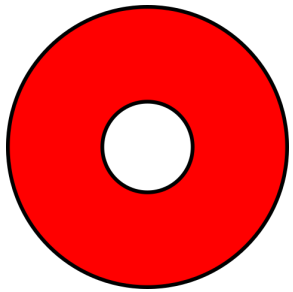
Cycles and basic patterns

At the beginning, was the cycle

Cycles and basic patterns

At the beginning, was the cycle

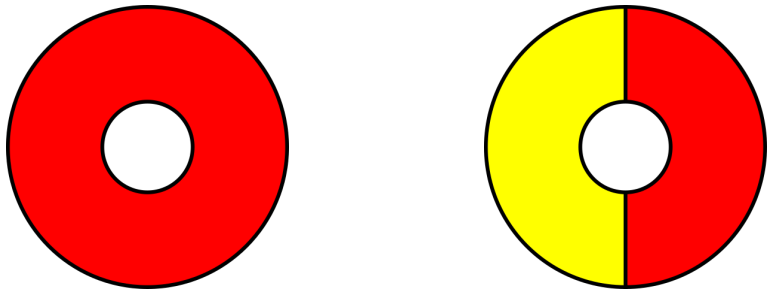
- Basic pattern: `d1 $ sound "bd"`



Cycles and basic patterns

At the beginning, was the cycle

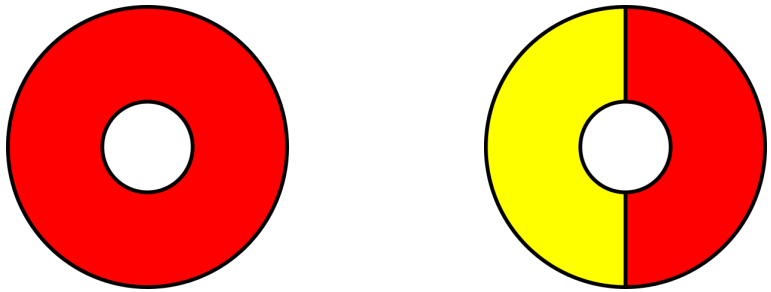
- Basic pattern: `d1 $ sound "bd"`



Cycles and basic patterns

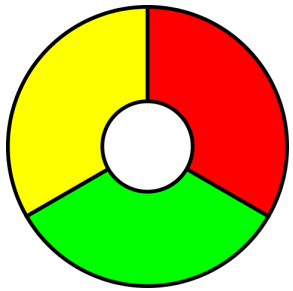
At the beginning, was the cycle

- Basic pattern: `d1 $ sound "bd"`



- `[]` : grouping internal patterns :

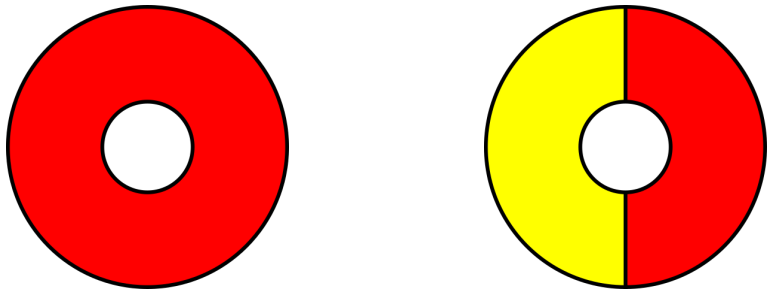
```
d1 $ sound "bd [sn hh] [arpy co hh sn]"
```



Cycles and basic patterns

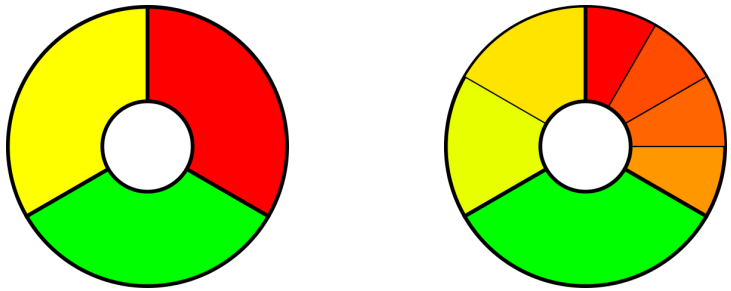
At the beginning, was the cycle

- Basic pattern: `d1 $ sound "bd"`



- `[]` : grouping internal patterns :

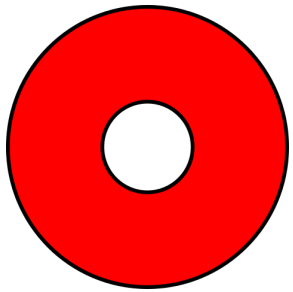
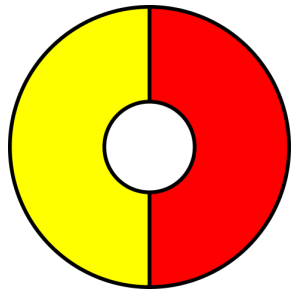
```
d1 $ sound "bd [sn hh] [arpy co hh sn]"
```



Cycles and basic patterns

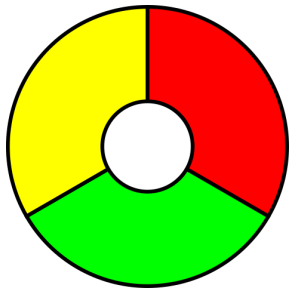
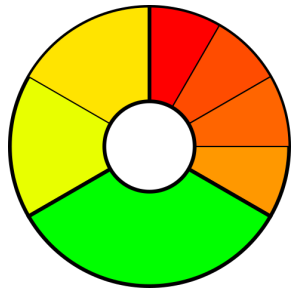
At the beginning, was the cycle

- Basic pattern: `d1 $ sound "bd"`

- `[]` : grouping internal patterns :

```
d1 $ sound "bd [sn hh] [arpy co hh sn]"
```

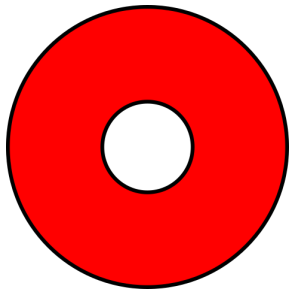
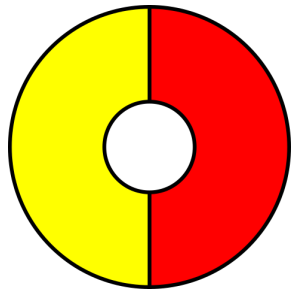



- `*` et `/`: repetition and spacing of patterns

Cycles and basic patterns

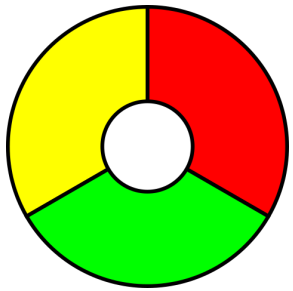
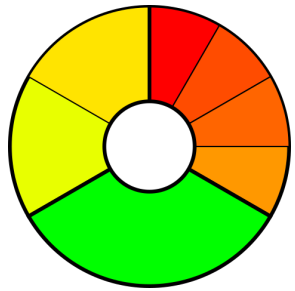
At the beginning, was the cycle

- Basic pattern: `d1 $ sound "bd"`

- `[]` : grouping internal patterns :

```
d1 $ sound "bd [sn hh] [arpy co hh sn]"
```

- `*` et `/`: repetition and spacing of patterns
- `~`: rest

Effects

: effects

Effects

`#` : effects

- `gain`
- `shape` (distortion)
- `delay`, `delayfeedback`, `delaytime`
- `speed` and `up`
- `vowel`

Effects

`#` : effects

- `gain`
- `shape` (distortion)
- `delay`, `delayfeedback`, `delaytime`
- `speed` and `up`
- `vowel`

Variations: `|*|`, `|+|`, `|/|`...

Functions over patterns

Functions that take a pattern and give another pattern

Functions over patterns

Functions that take a pattern and give another pattern
Example: `slow`

Functions over patterns

Functions that take a pattern and give another pattern

Example: `slow`

Chaining functions: `$`

```
d1 $ slow 2 $ slow 3 $ sound "bd*3"
```

Functions gallery

Other functions:

- `rev`
- `palindrome`
- `every`
- ...

Functions gallery

Other functions:

- `rev`
- `palindrome`
- `every`
- ...

Remark

```
palindome = every 2 (rev) = every 2 $ rev
```

Patterns gallery

- ```
  d1 $ up "0 1 2 4" # sound "bip"
  ```
  ```

Patterns gallery

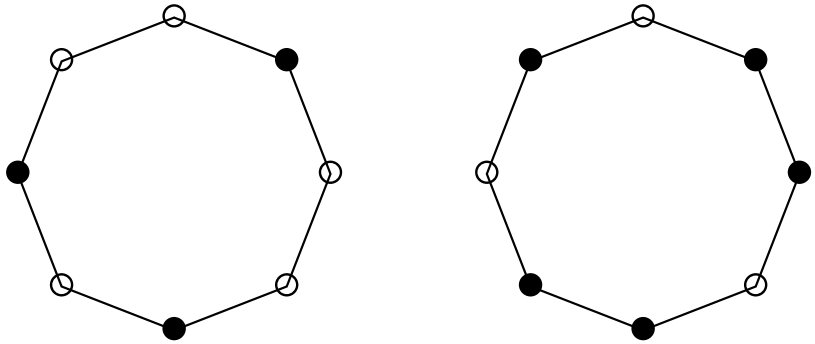
- `d1 $ up "0 1 2 4" # sound "bip"`
- Polyrhythms: `d1 $ sound "[bd bd, sn sn sn]"`

Patterns gallery

- `d1 $ up "0 1 2 4" # sound "bip"`
- Polyrhythms: `d1 $ sound "[bd bd, sn sn sn]"`
- Polymeters: `d1 $ sound "{bd bd, sn sn sn}"`

Patterns gallery

- `d1 $ up "0 1 2 4" # sound "bip"`
- Polyrhythms: `d1 $ sound "[bd bd, sn sn sn]"`
- Polymeters: `d1 $ sound "{bd bd, sn sn sn}"`
- Euclidean rhythms: `d1 $ sound "bd(3,8)"` [2]



Have some variations

- `sometimes`, and associates `often` and `rarely`
- Dicrete (`irand`) et continuous (`rand`) random patterns
- `degrade` et `degradeBy`
- ...

Samples manipulation

- `cut`
- `chop` and `striate`
- `scramble`
- `loopAt`
- ...

Bibliography

Workshop material will be available at
`https://agathe.herrou.fr/tidal/`.
Workshop based on
`https://tidalcycles.org/index.php/Tidal_workshop_worksheet`.

Laurie Spiegel.
Manipulations of musical patterns.
Proceedings of the Symposium on Small Computers and the Arts, 1981.

Godfried Toussaint.
The euclidean algorithm generates traditional musical rhythms.