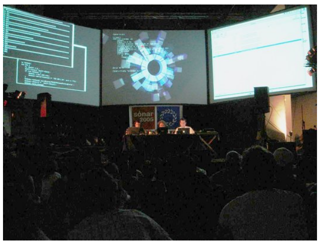
%% -*- mode: literate-tidal; mode: iimage; mode: visual-line -*-

[To run examples, install tidal (from http://yaxu.org/tidal/) and
tidal-vis (from hackage), and evaluate the code at the end.]

Making Programming Languages to Dance to: Live Coding
with Tidal
Alex McLean
University of Leeds

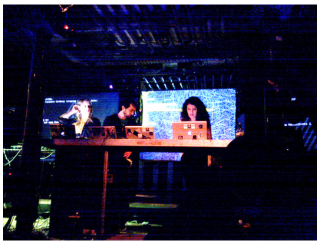


Principles

* Programmer as human
* Code as artistic expression
* Against autonomy
* Connecting with people through code

Live coding

http://toplap.org/



"Black slate" live coding as a design challenge

* Focused, creative flow, working in environment
* Just in time
* Immediate interaction with others through code changes
* Highly expressive; terse, and with close domain mapping
between code
  and music
* Ability to change more important than ability to understand
* Little time for TDD, etc..

Live coding and Functional programming

* Overtone
* LiveCodeLab
* Fluxus
* Extempore
* Conductive
* Live-Sequencer
* Tidal

Tidal

Vague timeline to Tidal

* 2000 - London - Slub formed with Ade Ward, a lot of Perl scripts, RealBasic and C
* 2001 - Paradiso - First room dancing to our code (Paradiso, Amsterdam)
* 2001/2 - Berlin - Transmediale software art award
* 2004 - Hamburg - First live coding workshop, TOPLAP is born (and then feedback.pl)
* 2005 - Barcelona - Dave Griffiths joined, first good live coded slub performance (Sonar festival)
* 2006 - started learning Haskell (during MSc)
* 2009 - First Tidal-esque pattern language appeared
* 2012 - London - First ``Algorave''
* 2012 - Goldsmiths - Completed PhD ``Artist-Programmers and Programming Languages for the Arts''
* 2013 - Barcelona - Tidal became installable by others

Tidal: Developed through performance

* 'Fully' improvised
* Hundreds of diverse performances
* Mainly with other live coders, percussionists, plus singers, dancers,
  noise artists, a roots band, a punk banjo player..
* Broken techno and free Jazz

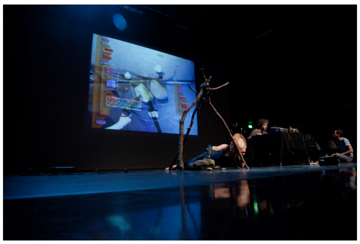
```
d2 $ every 3 (smash 3 [1,2,3]) $ sound (
n] [hh chin chin]]")
    |+| pan "1"
    |+| speed "1"
    |+| end "0.2"

d3 $ sound "bd bd [casio bd]"
```

Slub

SC <> BC

Canute (+ Shelly Knotts)



Hair of the horse with Hester Reeve

Corlab

Algorave

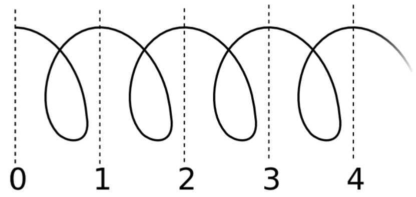
Algoravers

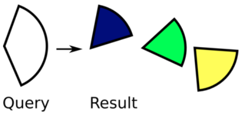
Tidal and Time

Tidal's Pattern Datatype

* Time is Rational
* Time is cyclic (with a period of 1)
* Polyrhythm works fine
* An Arc is a time range
* An Event is a value that is active within its own arc
* A Pattern is a function giving all the events occuring within a given arc

* May represent both discrete and continuous events





```
> type Time = Rational
> type Arc = (Time, Time)
> type Event a = (Arc, Arc, a)
> data Pattern a = Pattern (Arc -> [Event a])
```

Building and combining patterns

Starting with nothing:

```
> silence :: Pattern a
> silence = Pattern $ const []
```

A 'pure' value, one event every cycle:

```
> pure x =
>   Pattern $ \(s, e) ->
>     map (\t -> ((t%1, (t+1)%1),
>                 (t%1, (t+1)%1),
>                  x
>                 )
>                )
>                [floor s .. ((ceiling e) - 1)]
```

```
> d1 $ sound (pure "bd")
```

Manipulating time

```
> mapArc :: (Time -> Time) -> Arc -> Arc
> mapArc f (s,e) = (f s, f e)
```

```
> withQueryArc :: (Arc -> Arc) -> Pattern a -> Pattern a
> withQueryArc f p = Pattern $ \a -> arc p (f a)
```

```
> withQueryTime :: (Time -> Time) -> Pattern a -> Pattern a
> withQueryTime = withQueryArc . mapArc
```

```
> withResultArc :: (Arc -> Arc) -> Pattern a -> Pattern a
> withResultArc f p = Pattern $ \a -> mapArcs f $ arc p a
```

```
> withResultTime :: (Time -> Time) -> Pattern a -> Pattern a
> withResultTime = withResultArc . mapArc
```

Shifting time

```
> (<~) :: Time -> Pattern a -> Pattern a
> (<~) t p = withResultTime (subtract t) $ withQueryTime (+ t) p
```

```
> (~>) = (<~) . (0-)
```

Compressing time

```
> density :: Time -> Pattern a -> Pattern a
> density r p = withResultTime (/ r) $ withQueryTime (* r) p
```

```
> vis $ density 3 (cat [pure red, pure blue])
```

```
> densityGap :: Time -> Pattern a -> Pattern a
> densityGap r p = splitQueries $ withResultArc (\(s,e) -> (sam s + ((s - sam s)/r), (sam s + ((e - sam s)/r)))) $ Pattern (\a -> arc p $ mapArc (\t -> sam t + (min 1 (r * cyclePos t))) a)
```

```
> vis $ density 2 $ densityGap 2 (cat [pure red, pure blue])
```

```
> vis $ density 4 $ densityGap 2 (cat [pure red, pure blue])
```

```
> compress :: Arc -> Pattern a -> Pattern a
> compress a@(s,e) p = s ~> densityGap (1/(e-s)) p
```

```
> vis $ compress (1%4, 1%2) (cat [pure red, pure blue])
```

Merging patterns

```
> overlay :: Pattern a -> Pattern a -> Pattern a
> overlay p p' = Pattern $ \a -> (arc p a) ++ (arc p' a)
```

```
> vis $ overlay (pure red) (pure green)
```

```
> stack :: [Pattern a] -> Pattern a
> stack ps = foldr overlay silence ps
```

```
> cat :: [Pattern a] -> Pattern a
> cat ps = stack $ map (\(n, p) ->
>       compress ((fromIntegral n) % (fromIntegral $ length ps), (fromIntegral n+1) % (fromIntegral $ length ps)) p) (zip [0..] ps)
```

```
> vis $ cat [pure red, pure blue]
```

```
> vis $ cat ["red blue", "green orange purple"]
```

Polyrhythmic DSL - Parsing strings

```
> vis $ density 4 $ p "red [blue, green purple] orange"
```

```
> vis $ p "[blue, ~ [green [yellow tomato]], orange]*16"
```

Overloading strings saves a couple of characters

```
> vis $ "[blue, ~ [green yellow], orange]*16"
```

Different brackets for different kinds of polyrhythm:

```
> vis $ density 6 $ "[red black, blue orange green]"
```

```
> vis $ density 6 $ "{red black, blue orange green}"
```

```
> vis $ "white*128?"
```

```
> vis $ "[[black white]*32, [[yellow ~ pink]*3 purple]*5, [white black]*16]]*16"
```

Pattern as a functor

```
> vis $ fmap (blend 0.5 red) "blue black"
```

```
> vis $ blend 0.5
>   <$> "[blue orange, yellow grey]*16"
>   <*> "white blue black red"
```

TRANSFORMATIONS

Reversal

How to reverse an infinite pattern?

```
> vis $ density 16 $ every 3 rev "blue grey orange"
```

every

```
> vis $ density 16 $ every 3 rev "blue grey orange"
```

```
> vis $ density 4 $ every 4 ((1/3) <~) "blue grey purple"
```

```
> grid $ every 4 ((1/3) <~) "blue grey purple"
```

whenmod

```
> vis $ density 16 $ whenmod 6 3 rev "blue grey orange"
```

iter

```
> vis $ density 4 $ iter 4 $ "blue green purple orange"
```

superimpose

```
> vis $ density 4 $ superimpose (iter 4) $ "blue green purple orange"
```

Combining transformations

```
> vis $ density 8 $ whenmod 8 4 (slow 4) $ every 2 ((1/2) <~) $
>   every 3 (density 4) $ iter 4 "grey darkgrey green black"
```

Dirt

```
> d1 $ jux (iter 4) $ sound "bd sn:2"
>    |+| (slow 3 $ speed "1 2")
```

```
> d1 $ (jux (|+| speed "2") $ every 3 rev $ slow 8 $ striate 128 $ sound "bev")
  |+| vowel "a e i o"
```

```
> d1 $ every 3 (density 2) $ every 4 (density 2) $ (slow 2 $ spread' (chop) (every 3 rev "8 16 32 64") $ sound "[bass3 [~ bass3:8*2]]")
>   |+| speed "[4 2]/5"
```

```
> d4 $ slowspread ($) [id, rev, iter 4, density 2, (|+| speed "4")] $ sound (pick <$> "bd*2 lighter*4" <*> (slow 3 $ run 12))
```

```
  |+| vowel "a e i o u"

> d3 $ every 4 (within (0, 0.25) (density 4)) $  every 3 (0.25 <~) $ jux (iter 4) $ slow 4 $ chop 16 $ sound "shackup"

> d1 $ every 3 rev $ slow 2 $ every 2 (density 2) $ sound (samples "amencutup*4 sd8*4" (slow 1.5 $ run 12))
>         |+| shape "0.4"
>         |+| speed (scale 1 2 (slow 8 sine1) )

> d2 $ rev $ every 2 (inside 2 rev) $ chop 8 $ sound "breaks165"

> d5 $ slow 3 $ stack [(stut 8 0.9 1.5 $ sound "latibro:4 [[latibro:2*2 latibro:0*2] latibro:5*4]/8") |+| cutoff "[0.02 0.02 0.03 0.02 0.04 0.02 0.06 0.03]/8" |+| resonance "[0.7 0.4]/2" |+| gain "1.2" |+| pan "0.7"
>             ,(density 1.5 $ stut 8 0.9 1.5 $ sound "latibro:4 [[latibro:2*2 latibro:0*2] latibro:5*4]/8")
>             |+| cutoff (0.25 <~ "[0.02 0.025 0.03 0.03 0.04 0.03 0.06 0.03]/8") |+| resonance "[0.7 0.4]/2" |+| gain "1.2" |+| pan "0.3"
>            ]

> d7 $ slow 4 $ jux (|+| speed "8.08") $ spread' (stut 4 0.95) "[1/8 1/8 1/8 1/8, 1/3 5 3 2]/4" $ sound "~ [tok tok:1 tok:2 tok:3]" |+| speed "8"

> d1 $ slow 4 $ spread ($) [id, trunc (1/4), (0.25 <~), (|+| begin "0") . (|+| end "1") . chop 4] $ every 2 (0.25 ~>) $ jux (|+| speed "1.64 ! ! 4") $ sound "[bass3:6 ~ ~ bass3:6] [~ bass3:6]" |+| speed ((*1) <$> "[1.66 ! ! 4.02]")

> d1 $ slow 2 $ jux ((stut 2 0.5 3) . (|+| speed "1")) $ slow 4 $ striate 32 $ sound "mef*2"
         |+| speed "2"

> d1 $ jux (iter 4) $ every 4 (0.5 <~) $ every 3 (0.25 <~) $ chop 8 $ sound "rave:6 rave:7?"
         |+| shape "0.3"
```

THANKS FOR LISTENING

Grab it here:
http://yaxu.org/tidal/

```
import Sound.Tidal.Vis
import qualified Graphics.Rendering.Cairo as C
import Data.Colour
import Data.Colour.Names
import Data.Colour.SRGB
import System.Cmd

let vis pat = do vLines (C.withSVGSurface) "vis.svg" (400,100) pat 1 1
                 rawSystem "/home/alex/Dropbox/bin/fixsvg.pl" ["vis.svg"]
                 rawSystem "convert" ["vis.svg", "vis.pdf"]
                 return ()
    grid pat = do vLines (C.withSVGSurface) "vis.svg" (400,400) pat 10 10
                  rawSystem "/home/alex/Dropbox/bin/fixsvg.pl" ["vis.svg"]
                  rawSystem "convert" ["vis.svg", "vis.pdf"]
                  return ()
```