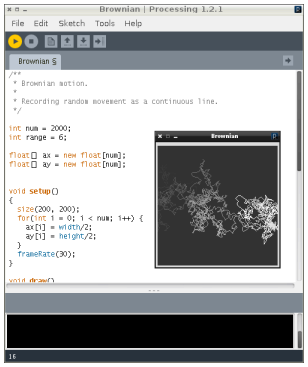
LiveCodeLab 2.0 and its language LiveCodeLang

Davide Della Casa, Guy John

What is LiveCodeLab

dive-in quick demo

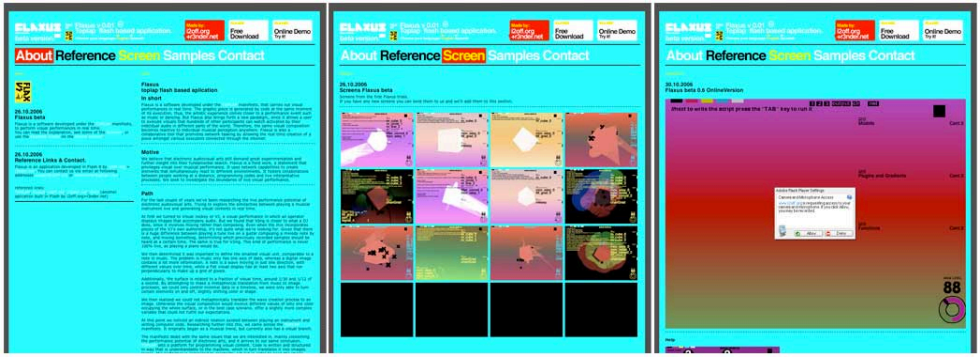
Processing, Casey + Reas



Jsaxus, Jonathan Brodsky

Fluxus, David Griffiths

Flaxus,
Ivanoff + Jimenez



Core values

1) learnability
2) usability

➔

simple syntax, simple commands keywords, short programs, immediate feedback and quick interaction

```
box // draws a box
```

eliminate parentheses for function invocation

(there is such a thing as a function invocation with no arguments!)

```
fill red
box
```

freely use 140 CSS colour literals

```
if random > 0.5
  box
else
  peg
```

Making use of indentation as a help to avoid parentheses, braces and semicolons.

```
rotate // affects the box
  box
peg
```

Making use of indentation as to define the scope of graphics state changes.

```
red // instead of “fill red”
box
```

Providing several shorthands for state changes in many cir-cumstances.

```
rotate red box
peg
```

Multiple instructions can be generally inlined (which limits the scope)

```
either = (a,b) ->
   if random > 0.5 then run a else run b

either <box>, <peg>
```

“<>” notation for inlined anonymous functions,
“run” is then used to actually evaluate the passed functions

```
above = <move 0,-0.5,0>
box above ball above peg


flashing = <if random < 0.5 then scale 0>
flashing ball
peg // peg doesn't flash
```

users can invent their own DSLs

How does it work: translation to coffeescript

```
rotate fill red box
```

…becomes (“Nested Closure” pattern):

```
rotate (-> fill red, (->box()))
```

Problems 1/4

Current transformations are based on regex matchers: nested cases not handled well.

Some transformations depend on actual literals used (e.g. for colours and colour functions).

Problems 2/4

Some simple code still requires some explanation:

```
2 times rotate box
```

(why not clearly two boxes?)

Problems 3/4

Some understanding needed of difference between expressions/values (something you calculate/have) and commands (something you ***do and accepts further functions as arguments***):

```
something = red
rotate fill something box
```

…vs:

```
something = <peg>
rotate fill red something
```

Problems 4/4

Three ways to assign things and they mean subtly different things:

```
something = red
rotate fill something box
```

…vs:

```
something = <peg>
rotate fill red something
```

…vs:

```
something = -> peg()
rotate fill red, -> something
```

Live Code Lang 2.0

Grammar

- BNF

Preprocessor

- Just handles indentation

Parser

- BNF -> Jison -> JS library -> AST

Interpreter

- AST -> Stuff

Maintainability

No More Regexps!

Block Scoping

Blocks and Indentation

Indented blocks allow us to apply changes to only sections of code.

- Matrix transformations
- Colour changes
- Loops

```
fill green

rotate
    fill red
        box

peg
```

Loops

Repeatedly run a block, with an optional counter

```
10 times with x
    rotate x
        box
```

```
move -3, 1
10 times with x
  10 times with y
    move x, -y, 0
      box 0.7
```

No Indentation Braces

Pre-processor is currently responsible for working out block boundaries.

Could be greatly simplified with more work however.

Type System

Shape

Any of the primitive shape functions

Matrix

The matrix transformation functions

Style

Fill and stroke

Type System

Embedded in the grammar

Little error checking gives limited usability

Does have future potential though

Inlining

Allow writing code on a single line

Meant to allow users to write code in a fashion that resembles a written sentence

Inlining

rotate red box move blue peg

equivalent

```
rotate
    fill red
        ball
    move
        fill blue
            peg
```

Future Research

Extension to type system might allow us to do away with the BNF Parser.

Improvements to inlining