A visual DSL toolkit in Lua

Past, present and future

Alexander Gladysh <ag@logiceditor.com>

Lua Workshop 2013
Toulouse

Outline

Alexander Gladysh

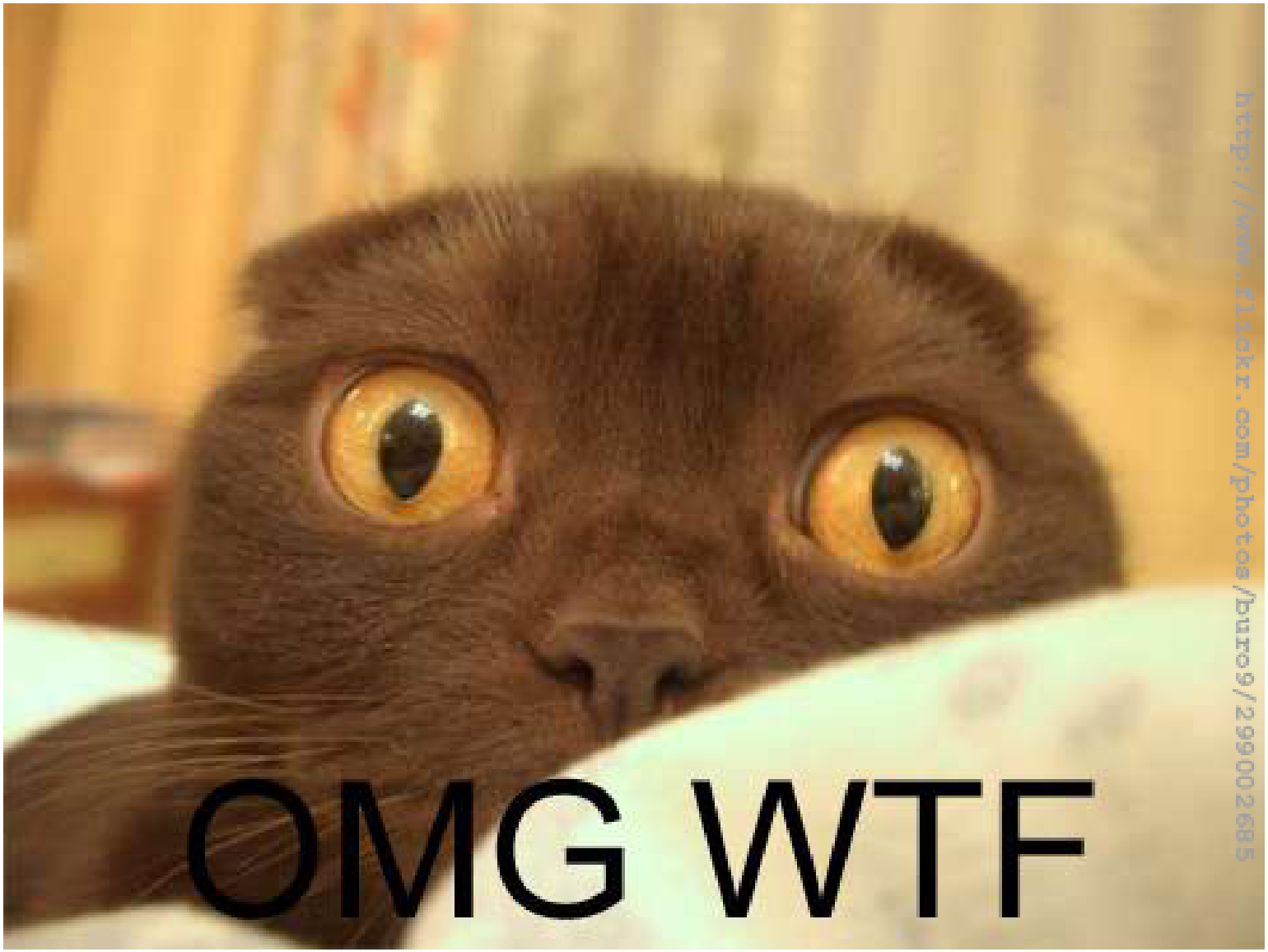
- CTO, co-founder at LogicEditor
- In löve with Lua since 2005

LogicEditor

- Use Lua to develop:
  - Visual DSL toolkit (subject of this talk)
  - Big-data analytics
  - Intensive-load web-services
  - In-browser and mobile games
- 600+ KLOC of private Lua codebase
- Some contributions to open-source Lua projects

The Problem

- Business-logic is highly volatile.
- Programmers are not domain area specialists.
- Specialist $\Leftrightarrow$ Manager $\Leftrightarrow$ Programmer loop is slow.
- Specialist $\Leftrightarrow$ Programmer loop is expensive.
- Let specialists do the business-logic!


OMG WTF
http://www.flickr.com/photos/buro9/2990026685

Non-programmers aren't programmers

It is not enough to be able to compose an algorithm and even implement it with some simple programming language.
For commercial programming you'll also need, at least:

- Technical background
- Debugging skills
- Team coding skills

Solution

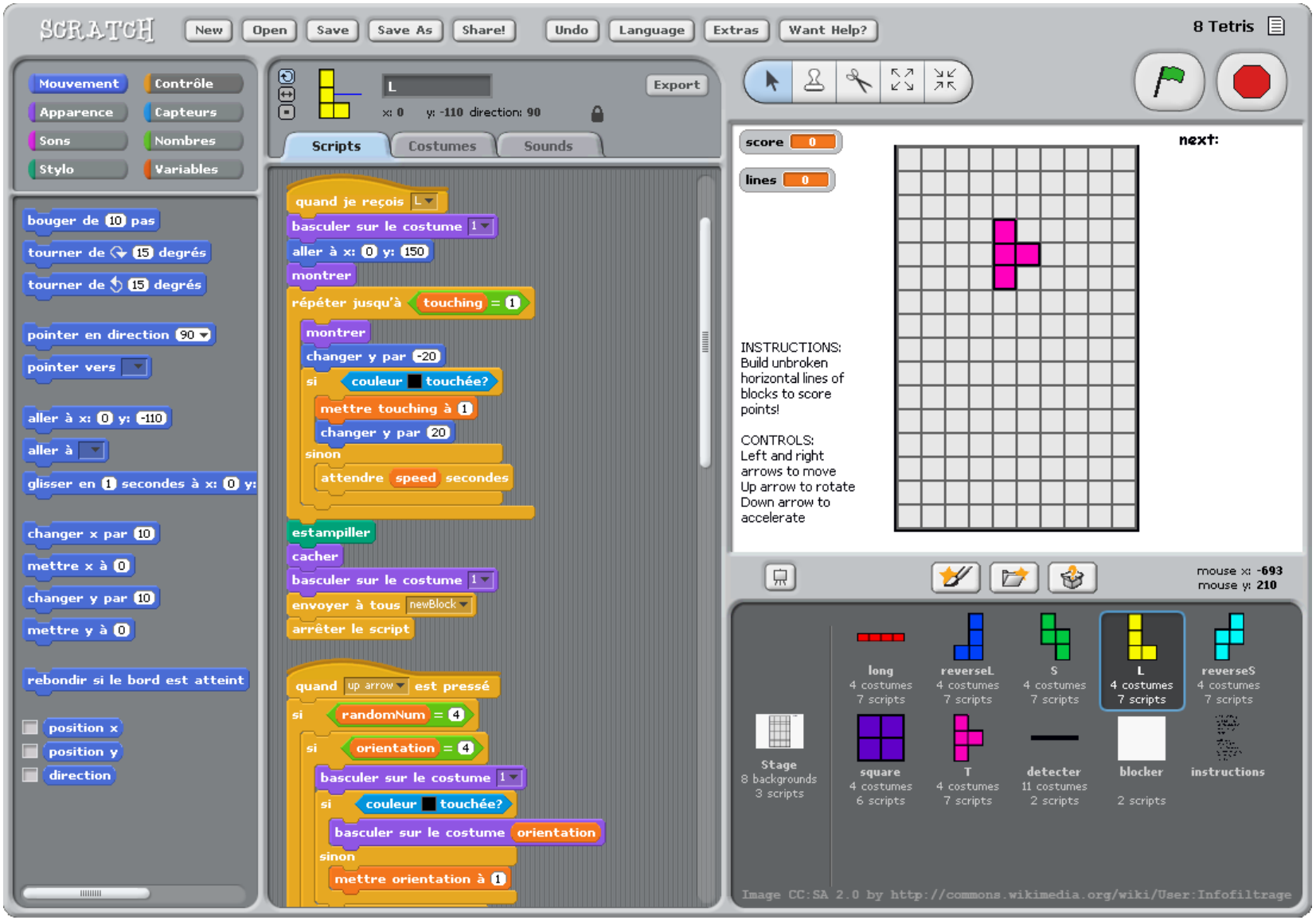
- A tool that prevents *technical* mistakes
- While limiting creativity as little as possible
- And is within grasp of a non-programmer.

Ad-hoc implementations

- One-shot, very limited flexibility
- Full of crutches
- Hard to maintain

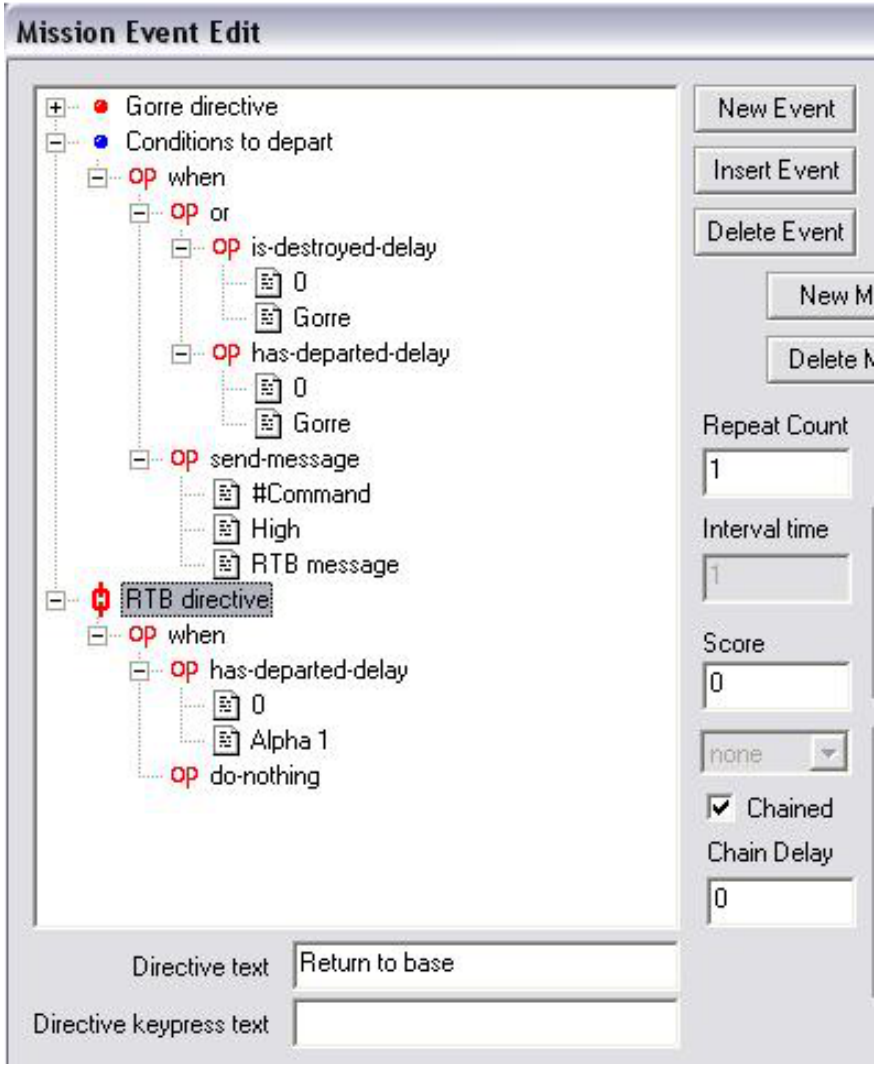
Classic third-party alternatives

MIT Scratch

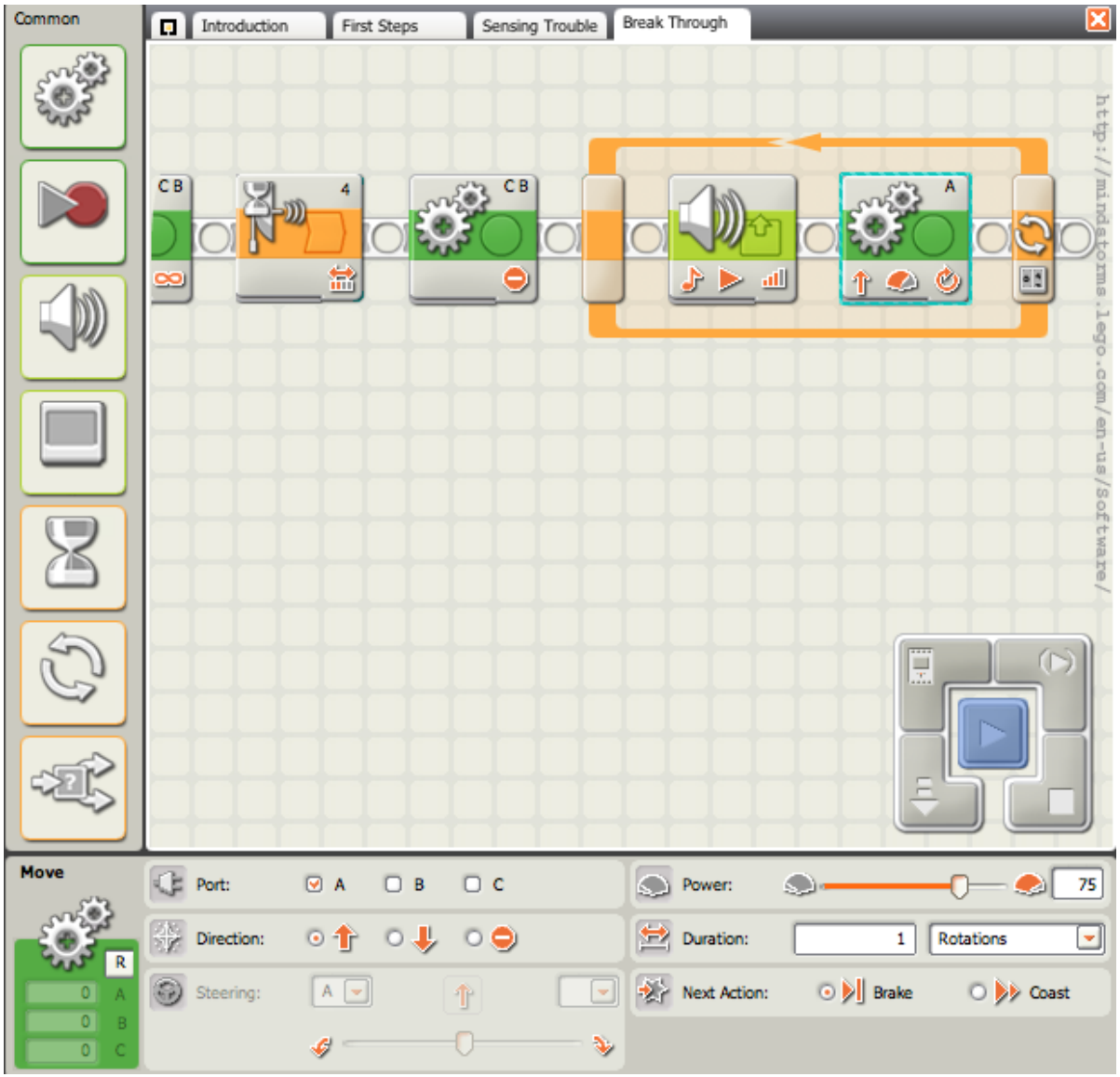

SCRATCH
New
Open
Save
Save As
Share!
Undo
Language
Extras
Want Help?
8 Tetris
Mouvement
Contrôle
Apparence
Capteurs
Sons
Nombres
Stylo
Variables
bouger de 10 pas
tourner de 15 degrés
tourner de 15 degrés
pointer en direction 90
pointer vers
aller à x: 0 y: -110
aller à
glisser en 1 secondes à x: 0 y:
changer x par 10
mettre x à 0
changer y par 10
mettre y à 0
rebondir si le bord est atteint
position x
position y
direction
L
x: 0 y: -110 direction: 90
Export
Scripts
Costumes
Sounds
quand je reçois L
basculer sur le costume 1
aller à x: 0 y: 150
montrer
répéter jusqu'à touching = 1
montrer
changer y par -20
si couleur touchée?
mettre touching à 1
changer y par 20
sinon
attendre speed secondes
estampiller
cacher
basculer sur le costume 1
envoyer à tous newBlock
arrêter le script
quand up arrow est pressé
si randomNum = 4
si orientation = 4
basculer sur le costume 1
si couleur touchée?
basculer sur le costume orientation
sinon
mettre orientation à 1
score 0
lines 0
next:
INSTRUCTIONS:
Build unbroken horizontal lines of blocks to score points!
CONTROLS:
Left and right arrows to move
Up arrow to rotate
Down arrow to accelerate
mouse x: -693
mouse y: 210
Stage
8 backgrounds
3 scripts
long
4 costumes
7 scripts
reverseL
4 costumes
7 scripts
S
4 costumes
7 scripts
L
4 costumes
7 scripts
reverseS
4 costumes
7 scripts
square
4 costumes
6 scripts
T
4 costumes
7 scripts
detecter
11 costumes
2 scripts
blocker
2 scripts
instructions
Image CC:SA 2.0 by http://commons.wikimedia.org/wiki/User:Infofiltrage

Descent Freespace Editor Events

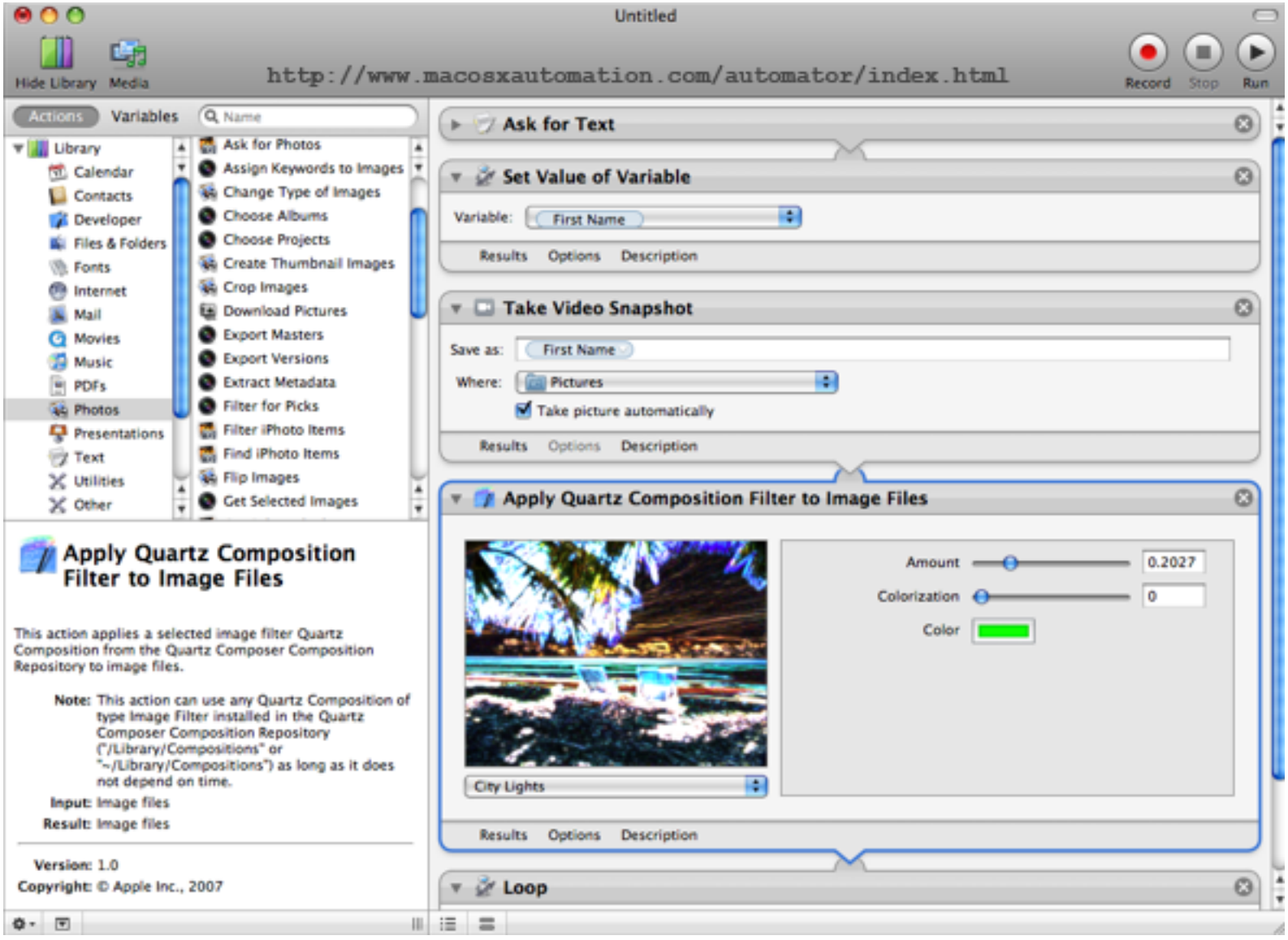
http://www.hard-light.net/wiki/index.php/\File:RTBSampleExpanded.JPG



Lego NXT-G


Common
Introduction
First Steps
Sensing Trouble
Break Through
http://mindstorms.lego.com/en-us/Software/
Move
Port:
A
B
C
Direction:
Steering:
Power:
75
Duration:
1
Rotations
Next Action:
Brake
Coast

Apple Automator



What is in common?

- **A visual domain-specific language**,
- that allows user to describe
- **the control-flow**.

A retrospective of ideas

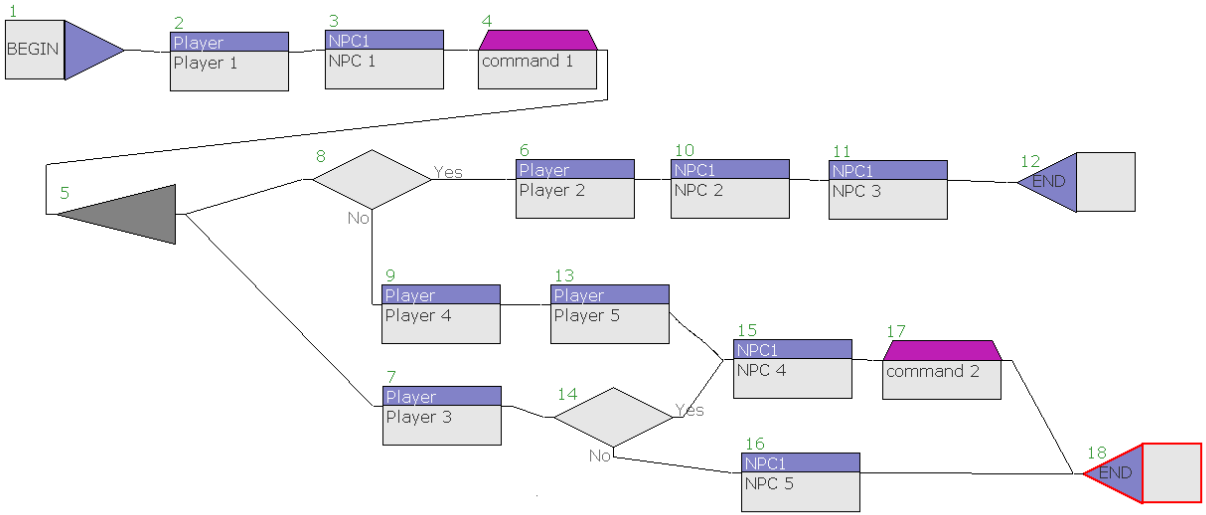
Screenshots shown here are from editors done by me and/or my colleagues for different companies we worked for, over time.
Only an idea was re-used and improved between generations.

Video-adventure game editor

(No screenshots available)

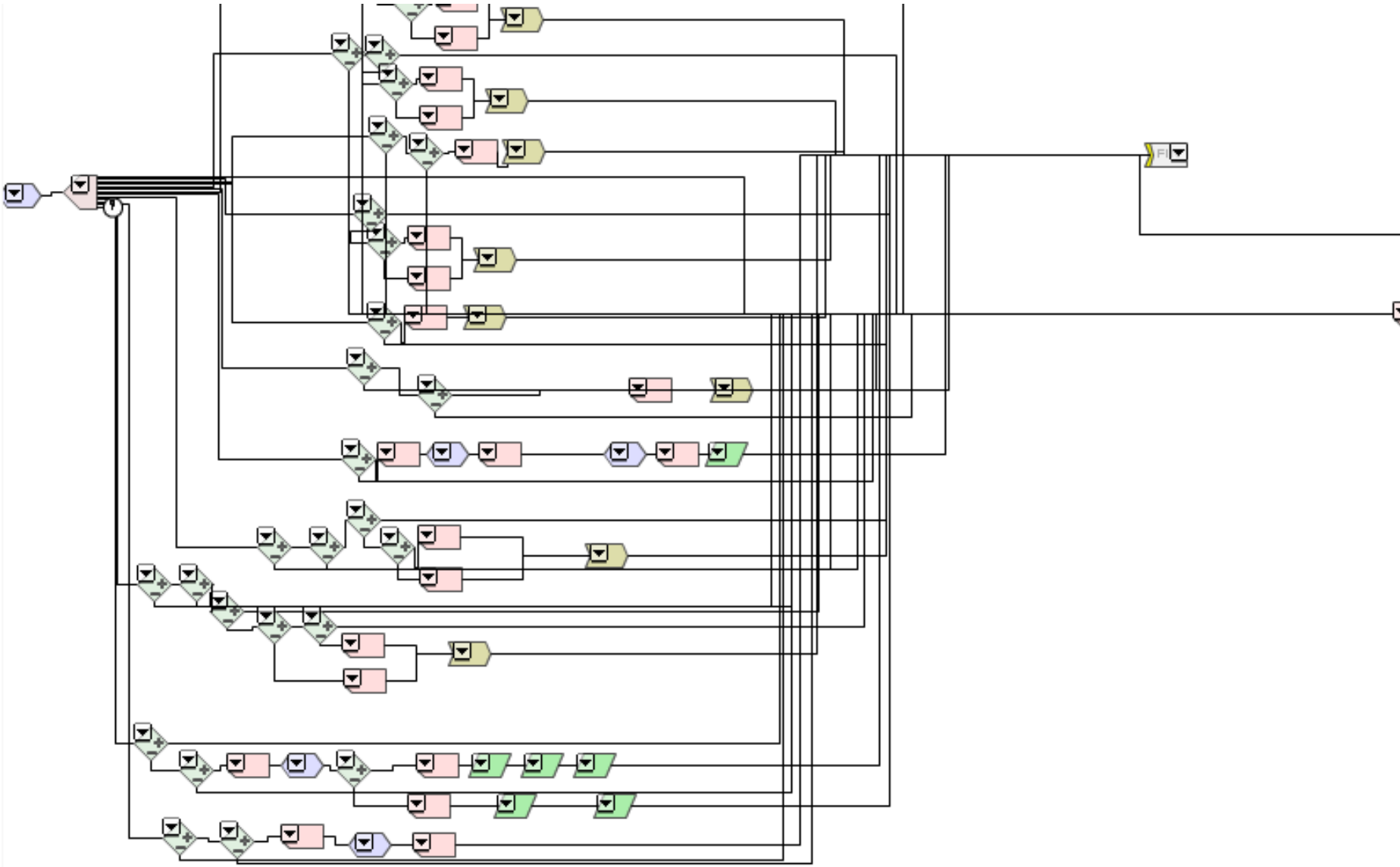
- Legacy, circa 2002—2004
- Graph of in-game dialog remarks and answer choices
- Allowing to tie-in video-loop to remark
- No Lua.

Adventure game dialog editor, I, II

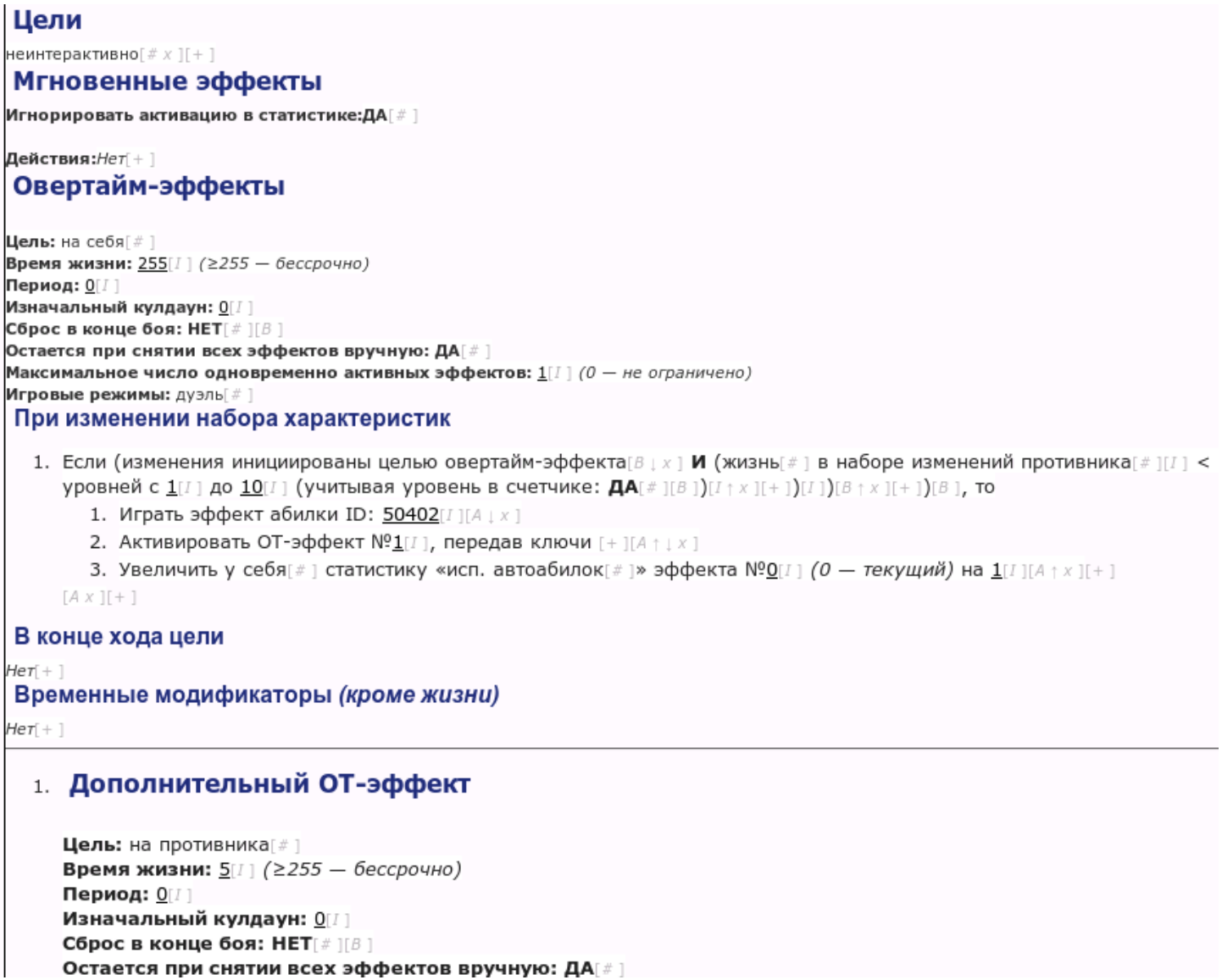


- Graph of in-game dialog remarks and answer choices
- With ability to add custom Lua code logic (in II)
- Generated data (in Lua) for a state machine (also in Lua)

Browser MMO quest editor, I, II



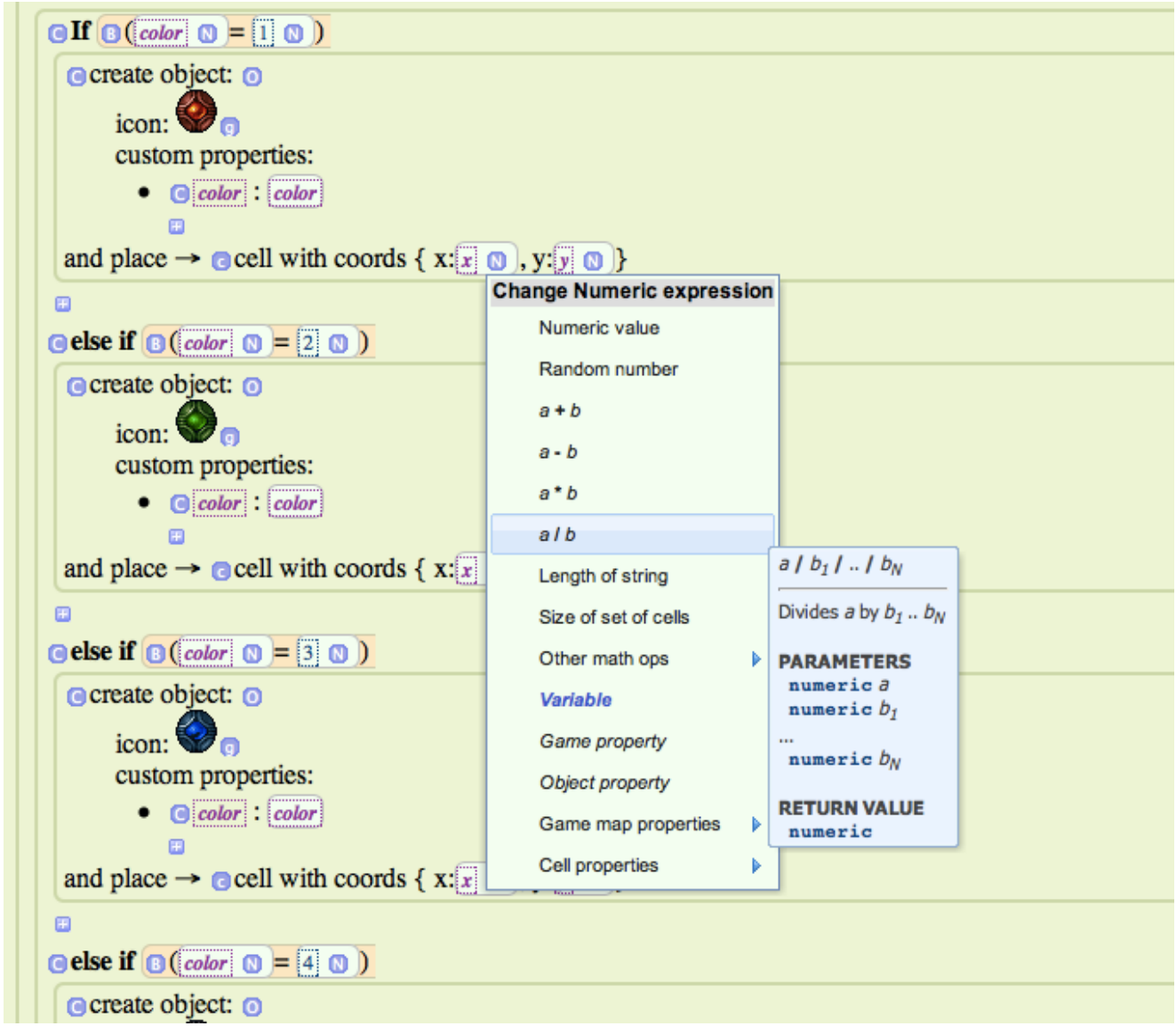
Browser MMO magic system editor



Analysis

- Some non-programmers prefer visual control-flow editors.
- Some — textual representation.
- (Programmers hate to use both kinds.)
- All editors were very useful, some — invaluable.
- But, in retrospective, some should have been replaced by dedicated coders.
- None of the past-generation editors were flexible enough to be used outside its immediate domain (but this never was an official goal for them).

The Visual Business Logic Editor Toolkit



Design goals

- Easy to create new editors.
- Easy to support existing editors.
- Easy to integrate with "any" other project on "any" technology.
- Easy *enough* to learn and use by end-users.

Editor use-cases

For example:

- A dialog editor for a game scenario writer.
- A magic system editor for a game-designer.
- A mission logic editor for a game level-designer.
- A DB query editor for a data analyst (Hadoop, anyone?).
- An advertising campaign targeting editor for a marketer.
- ...and so on.

Technology

- The data is a tree corresponding to the control flow (or to anything tree-like, actually).
- The output is structured text (code or data).
- Editor code, UI and backend, is generated by Lua code in the Toolkit, from the data "schema".
- Editor UI is in JavaScript / HTML, backend is in Lua.

The Data Schema

- Embedded Lua DSL (see my talk on Lua WS'11).
  `http://bit.ly/lua-dsl-talk`
- Describes how to:
  - check data validity,
  - generate default data,
  - render data to editor UI,
  - change data in editor UI,
  - render the conforming data to the output code (or data).
- Two layers: human-friendly and machine-friendly

Schema Example, I

See also: `http://bit.ly/le7-schema`

```
lang:root "lua.print-string"

lang:value "lua.string.value" {
  data_type = "string";
  default = "Hallo, world!";
  render:js [[String Value]] { [[${1}]] };
  render:lua { [[${1}]] };
}
```

Schema Example, II

```
lang:func "lua.print-string" {
  "lua.string.value";
  render:js [[Print string]] {
    [[Print: ${1}]];
  };
  render:lua {
    [[print(${1})]];
  };
}
```

Default Data

```
{
  id = "lua.print-string";
  {
    id = "lua.string.value";
    "Hallo, world!";
  }
}
```

Renders to Lua as:

```
print("Hallo, world!")
```

UI for default data (simplified)

```
<div id="lua.print-string">
  Print: <span id="lua.string.value">Hallo, world!</span>
</div>
```

NB: That turns to edit-box on click.

Extending string type

```
lang:type "lua.string" {
  init = "lua.string.value";
  render:js [[String]] { menu = [[S]]; [[${1}]] };
  render:lua { [[${1}]] };
}

lang:func "lua.string.reverse" {
  type = "lua.string";
  render:js [[Reverse string]] { [[Reverse: ${1}]] };
  render:lua { [[(${1}):reverse()]] };
}
```

Print with multiple arguments

```
lang:list "lua.print"
{
  "lua.string";
  render:js [[Print]] {
    empty = [[Print newline]];
    before = [[Print values: <ul><li>]];
    glue = [[</li><li>]];
    after = [[</li></ul>]];
  };
  render:lua {
    before = [[print(]];
    glue = [[,]];
    after = [[)]];
  };
}
```

Main primitives

- lang:const
- lang:value
- lang:enum
- lang:func
- lang:list
- lang:type

Machine-friendly schema

- node:literal
- node:variant
- node:record
- node:value
- node:list

Data-upgrade routines

- A set of hooks for data tree traversal.
- Transformations between two given data versions.
- In terms of node schema.
- Semi-automatic, template code is generated.

What else?

- Scopes in the schema.
- External and internal data-sources.

Several points of improvement

Current generation does its job well, but we see several ways on how to make it better
Several points to improve

- Better, modern HTML (at the cost of support of IE6).
- Lua in browser for a server-less integration option.
- Even more flexible and expressive Schema DSL.

NB: We'll probably go for a control-flow diagram UI first, not text-based one (current text-based is cool enough).

Problems with the current DSL

- One language for three separate concepts:
  - data tree structure,
  - editor UI,
  - final output.
- Data tree structure gets a bit synthetic and convoluted at times.
- Should be easier to add alternative editor UIs.

Solution

- Three separate sets of languages:
  - data tree format,
  - render to output (per output format),
  - render to editor (per editor kind).
- CSS-like rules instead of pre-defined set of node types

Early examples

http://bit.ly/le8-proto

```
data:root "script"
data:type "script" ("*", "action")
data:type "action" "print-var" "var-name"

to:text "script" :T [[
local _VARS = {}
${indent(concat(children))}
]]
to:text "print-var" "var-name"
  :T [[print(_VARS[${quote:lua(node)}])]]

to:ui "print-var" "var-name"
  :T [[Print: ${child(1)})]]
```

An alternative approach to the Embedded DSLs in Lua

```
foo:bar "baz" { "quo" }
```

```
local proxy = foo
proxy = proxy["bar"]
proxy = proxy(foo, "baz")
proxy = proxy({ "quo" })
```

The FSM

```
foo:bar "baz" { "quo" }
```

If proxy is as a FSM, indexes and calls — state transitions.

```
INIT | index "bar" -> foo.bar
       foo.bar | call -> foo.bar.name
  foo.bar.name | call -> foo.bar.name.param
FINAL <- foo.bar.name.param
```

Early working prototype: `http://bit.ly/le-dsl-fsm`.

Easier to code complex DSL constructs

```
play:scene [[SCENE II]]
.location [[Another room in the castle.]]
:enter "HAMLET"
:remark "HAMLET" [[
Safely stowed.
]]
:remark { "ROSENCRANTZ", "GILDERSTERN" }
  .cue [[within]] [[
Hamlet! Lord Hamlet!
]]
:remark "HAMLET" [[
What noise? who calls on Hamlet?
O, here they come.
]]
```

Questions?

Alexander Gladysh, ag@logiceditor.com