

A visual DSL toolkit in Lua

Past, present and future



Alexander Gladyshev <ag@logiceditor.com>

Lua Workshop 2013
Toulouse

Outline

Introduction

The Problem

Classic third-party alternatives

Past generations

The present generation

The future

Questions?

Alexander Gladysh

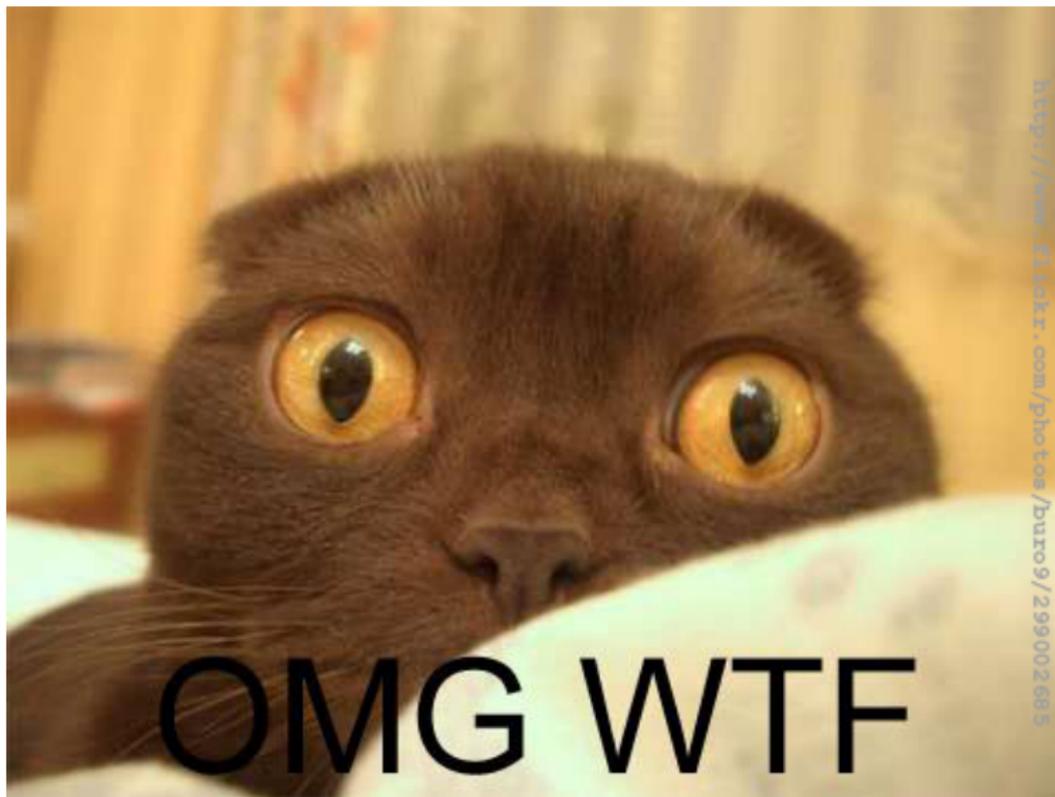
- ▶ CTO, co-founder at LogicEditor
- ▶ In löve with Lua since 2005

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!



<http://www.flickr.com/photos/buro9/299002685>

OMG WTF

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
Style Variables

Scripts Costumes Sounds

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 rotate and accelerate

mouse x: 693
mouse y: 210

Lang 4 costumes 7 scripts
reverseL 4 costumes 7 scripts
5 4 costumes 7 scripts
L 4 costumes 7 scripts
reverseS 4 costumes 7 scripts

stage 8 backgrounds 3 scripts
square 4 costumes 6 scripts
T 4 costumes 7 scripts
detecter 11 costumes 2 scripts
blocher 2 scripts
instructions

Image CC-SA 2.0 by <http://commons.wikimedia.org/wiki/User:Infociltrage>

Descent Freespace Editor Events

<http://www.hard-light.net/wiki/index.php/>

File:RTBSampleExpanded.JPG

Mission Event Edit

Legend:
● Gorre directive
● Conditions to depart

Tree structure:
- op when
 - op or
 - op is-destroyed-delay
 - 0
 - Gorre
 - op has-departed-delay
 - 0
 - Gorre
 - op send-message
 - #Command
 - High
 - RTB message
- **RTB directive** (selected)
 - op when
 - op has-departed-delay
 - 0
 - Alpha 1
 - op do-nothing

Buttons:
New Event
Insert Event
Delete Event
New M
Delete M

Repeat Count: 1
Interval time: 1
Score: 0
none (dropdown)
 Chained
Chain Delay: 0

Directive text: Return to base
Directive keypress text:

Lego NXT-G

The screenshot displays the Lego Mindstorms NXT-G software interface. At the top, there are four tabs: "Introduction", "First Steps", "Sensing Trouble", and "Break Through". The "Break Through" tab is active. On the left side, there is a vertical toolbar with icons for gears, a red stop button, a speaker, a monitor, a timer, a refresh button, and a move block. The main workspace shows a sequence of blocks on a grid. The sequence starts with a "CB" block, followed by a motor block with a value of "4", then another "CB" block, and finally a "Move" block with a value of "A". A blue dashed box highlights the "Move" block, and an orange arrow points to the right above it. In the bottom right corner of the workspace, there is a navigation pad with a blue play button. At the bottom of the interface, there is a "Move" block configuration panel. It includes a "Port" section with radio buttons for A, B, and C (A is selected). The "Direction" section has radio buttons for up, down, and left (up is selected). The "Steering" section has a dropdown menu set to "A" and a slider. The "Power" section has a slider set to 75. The "Duration" section has a text input field set to "1" and a dropdown menu set to "Rotations". The "Next Action" section has radio buttons for "Brake" and "Coast" (Brake is selected). On the right side of the workspace, there is a vertical URL: <http://nxt.f4.com/~lego.com/en-us/Software/>

Apple Automator

The screenshot shows the Apple Automator application window titled "Untitled". The address bar displays the URL `http://www.macosxautomation.com/automator/index.html`. The interface is divided into several sections:

- Top Bar:** Includes window control buttons (red, yellow, green) and system status icons (Record, Stop, Run).
- Left Panel:** Contains a "Library" section with icons for various system folders (Calendar, Contacts, Developer, Files & Folders, Fonts, Internet, Mail, Movies, Music, PDFs, Photos, Presentations, Text, Utilities, Other) and an "Actions" section with a search bar and a list of actions such as "Ask for Photos", "Assign Keywords to Images", "Change Type of Images", "Choose Albums", "Choose Projects", "Create Thumbnail Images", "Crop Images", "Download Pictures", "Export Masters", "Export Versions", "Extract Metadata", "Filter for Picks", "Filter iPhoto Items", "Find iPhoto Items", "Flip Images", and "Get Selected Images".
- Right Panel:** Displays a workflow with the following actions:
 - Ask for Text:** A simple text input field.
 - Set Value of Variable:** Variable set to "First Name".
 - Take Video Snapshot:** Save as "First Name", location "Pictures", with "Take picture automatically" checked.
 - Apply Quartz Composition Filter to Image Files:** Features a preview of a city night scene, sliders for "Amount" (0.2027) and "Colorization" (0), a "Color" dropdown set to green, and a "City Lights" filter selection.
 - Loop:** A loop action at the bottom.
- Bottom Panel:** Provides detailed information for the selected "Apply Quartz Composition Filter to Image Files" action, including a description, a note about Quartz Composition filters, input/output details, and version/copyright information.

What is in common?

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

A retrospective of ideas

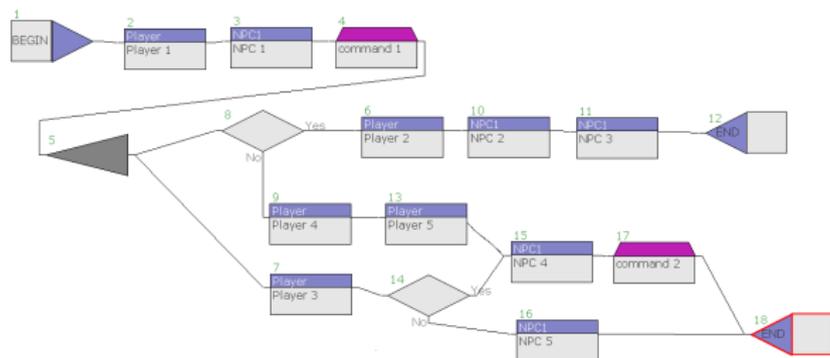
Screenshots shown here are from editors done by me and/or my colleagues for different companies we worked for, over time. Only an idea was re-used and improved between generations.

Video-adventure game editor

(No screenshots available)

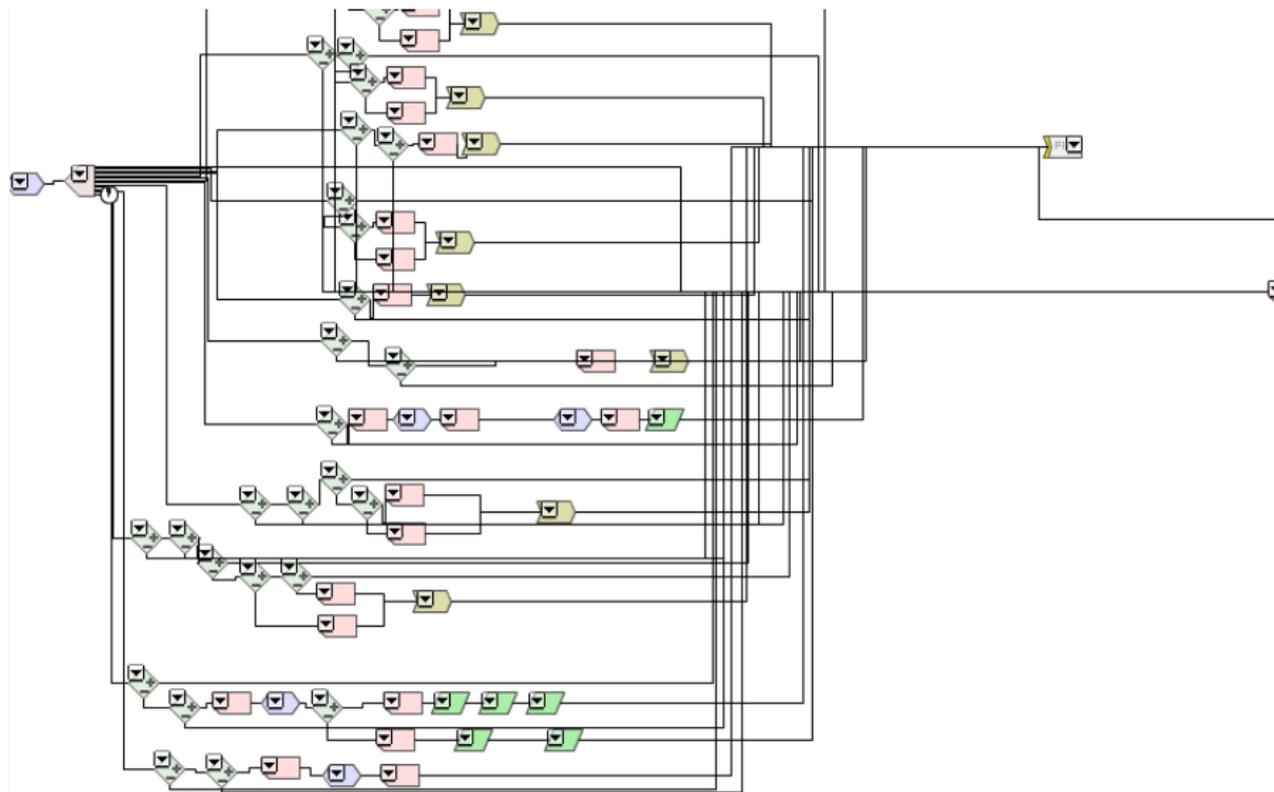
- ▶ Legacy, circa 2002—2004
- ▶ Graph of in-game dialog remarks and answer choices
- ▶ Allowing to tie-in video-loop to remark
- ▶ No Lua.

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

Цели

неинтерактивно[# x][+]

Мгновенные эффекты

Игнорировать активацию в статистике: **ДА**[#]

Действия: **Нет**[+]

Овертайм-эффекты

Цель: на себя[#]

Время жизни: **255**[/] (≥ 255 — бессрочно)

Период: **0**[/]

Изначальный кулдаун: **0**[/]

Сброс в конце боя: **НЕТ**[#][B]

Остается при снятии всех эффектов вручную: **ДА**[#]

Максимальное число одновременно активных эффектов: **1**[/] (0 — не ограничено)

Игровые режимы: дуэль[#]

При изменении набора характеристик

- Если (изменения инициированы целью овертайм-эффекта[B | x] **И** (жизнь[#] в наборе изменений противника[#][/] < уровней с **1**[/] до **10**[/] (учитывая уровень в счетчике: **ДА**[#][B])(/ + x][+ 1])(/))(B + x][+ 1])[B], то
 - Играть эффект абилки ID: **50402**[/][A | x]
 - Активировать ОТ-эффект №**1**[/], передав ключи [+][A | x]
 - Увеличить у себя[#] статистику «исп. автоабилка[#]» эффекта №**0**[/] (0 — текущий) на **1**[/][A + x][+][A x][+]

В конце хода цели

Нет[+]

Временные модификаторы (кроме жизни)

Нет[+]

1. Дополнительный ОТ-эффект

Цель: на противника[#]

Время жизни: **5**[/] (≥ 255 — бессрочно)

Период: **0**[/]

Изначальный кулдаун: **0**[/]

Сброс в конце боя: **НЕТ**[#][B]

Остается при снятии всех эффектов вручную: **ДА**[#]

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

The screenshot displays a visual programming interface for business logic. The main workspace contains a script with four conditional blocks, each starting with "C If" and a condition involving a "color" property and a numeric value (1, 2, 3, or 4). Each block includes instructions to "create object" (with a corresponding icon: a red sphere, a green sphere, a blue sphere, and a red sphere), set "custom properties" (including a "color" property), and "and place" the object in a "cell with coords { x: x, y: y }".

A context menu titled "Change Numeric expression" is open over the script. The menu lists various options:

- Numeric value
- Random number
- $a + b$
- $a - b$
- $a * b$
- a / b** (highlighted)
- Length of string
- Size of set of cells
- Other math ops
- Variable
- Game property
- Object property
- Game map properties
- Cell properties

A secondary menu is open over the "a / b" option, showing:

- $a / b_1 / \dots / b_N$
- Divides a by $b_1 .. b_N$
- PARAMETERS**
- numeric a
- numeric b_1
- ...
- numeric b_N
- RETURN VALUE**
- numeric

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/1e7-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!</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/1e8-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 Gladyshev, ag@logiceditor.com