The Medusa compiler

A Lua tool for highly interactive ebooks

Enrico Colombini <erix@erix.it> (freelance author)

Lua workshop 2014, September 13-14, Moscow
Old ‘gamebooks’

- 1970s-1980s (not the first ones)
- Printed on paper
- Simple choices
Branches in gamebooks

• “The werewolf is getting nearer”
  - Shoot a silver arrow (go to 75).
  - Flee as fast as you can (go to 47).
Variables in gamebooks

• “There is only a rusty piece of metal here”
  - Pick it up (tick checkbox ‘C’, go to 22).
  - Leave it here (go to 22).
Memory in gamebooks

- Pen and paper are used to store information
- The human reader is the run-time processor
Interactive ebooks

- We have hyperlinks, but...
- No (portable) runtime language
- No way to save variables
- No external storage, processing
- Only remembers the current page
- Immutable set of ‘printed’ pages
- What can we do?
Memory in an ebook

- A boolean can be stored by duplicating pages
- Each branch represents a different state

![Diagram showing two series of identical pages, one of which reaches the car with keys and the other cannot because it took the keys.](image-url)
Source page, instance pages

- Each instance page ‘contains’ its state
- Instance page contents can be the same
Implications

- Multiple ‘vars’ allow complex behaviour
- A simple bookmark ‘saves’ the whole state
- The human reader is not aware of the state

- The number of possible states must be finite
  (no open-ended counters, no random)
- Combinatorial explosion (6 bools = pages x 64)
Page budget

- Printed book: 100s of pages (cost)
- Ebook: 1,000s of pages (e-reader limitations)
- Static website: 1,000,000s of pages (space, fs)
- Combinatorial explosion must be limited
- Localize states, use patterns
- Hard to do by hand
Medusa toolchain
With a little help from Lua...

- An adventure game ebook
- Puzzles of different types
- Items to pick up, dialogs
- Free exploration
- Multiple characters
- Counters, timers
- 940 (small) source pages
- 5800 instance (‘printed’) pages, 11000 links

(in Italian only)
Una visita sgradita

In una pentola di cocco sotto al letto tieni inoltre i tuoi sudati risparmi: ben sedici "lupi" d'argento, le luccicanti monete delle Terre Libere. Ti serviranno presto: quest'anno la grandine ha distrutto il raccolto e non hai nemmeno le sementi per la semina.

Un giorno però ti si presenta un tale vestito di nero: dice che il suo mestiere è "proteggere chi lavora da eventuali incidenti" e ti propone di versargli cinque lupi d'argento "come polizza contro gli infortuni".

- Gli dai il denaro richiesto
- Ti rifiuti adducendo una scusa
- Lo cacci a male parole

Possiedi: 16 lupi d'argento, 2 caproche, 0 vaccopotami.
function Kitchen() {
    title("The room is on fire")
    text("Things are getting rather hot here.")
    choice("Get out immediately", Garden)
    choice("Grab the beer can and get out",
            "v.beer = 1; go(Garden)")
}

- Each page is a function
- Page content is ‘printed’ by API calls
- Link actions may contain code
- The state is changed by the link action
The v. object (table) contains the state

The same page function (source page) may ‘print’ different content (instance pages)

Usual game logic programming

```javascript
function Garden() {
    title("Garden outside burning house")
    if (v.beer == 1) {
        text("You are holding a beer can.")
    }
    // ...
}
```
Back to the ebook

- Could we reuse the same approach?
- Pages are immutable, there is no runtime
- How to keep track of the state?
- How to change the state on link action?
- How to print different content depending on state?
- How to allow a ‘normal’ programming style?
Ebook as directed graph

- Nodes are pages
- No predictable structure, cycles are common
Just click on all links

- The number of nodes (instance pages) is finite
- We can enumerate them:
  - Start with an Idra-like source, at the first page
  - ‘Print’ the page with the initial state
  - Simulate clicking on a link, execute its code
  - ‘Print’ the destination page with changed state
  - Repeat recursively, recognizing visited nodes
The Medusa compiler

- Caveat: unrefined tool, made for my own use
- No error handling, no documentation
- HTML generation is very primitive
- ...but it worked fine for my project
# landing

if not v.count then v.count = 9 end

if v.count > 0 then
    P(v.count, " seconds to touchdown.")
    Choice("Wait", 'landing',
            F{v.count = v.count - 1} )
else
    P"We have landed on the Moon!"
end

- A source page is Lua code
- A link can contain Lua code
function page.landing(_ENV)

if not v.count then v.count = 9 end

if v.count > 0 then
    P(v.count, " seconds to touchdown.")
    Choice("Wait", 'landing',
        function(_ENV) v.count = v.count - 1 end)
else
    P"We have landed on the Moon!"
end
end

• Page and links become Lua functions
The vtable

- A vtable represents the current state
- “Variables table” (nothing to do with C++)

\[ v = \{ \text{count}=7 \} \]

- Vtables are shallow but can contain any number or type of scalar values

\[ v = \{ \text{ship='Niña'}, \text{days}=35, \text{landInView}=\text{false} \} \]

- Two instance pages of the same source page with equal vtables are the same instance page
Page generation

- Source page + vtable = Instance page

```lua
function page.landing(_ENV)
    ...
    P(v.count, " seconds to touchdown.")
    ...
end
```

- The source page function is called with the current vtable in its environment

```lua
v = { count=7 }
```

Output: `<p>7 seconds to touchdown.</p>`
Link generation

- The link function is called with a copy of the current vtable in its environment.
- The link function can change the vtable content.

```lua
Choice("Wait", 'landing',
  function(_ENV) v.count = v.count - 1 end)
```

- If the resulting instance page (source + vtable) exists, its name is put into the link.
- If not, a new instance page name is generated and a (source + vtable) request is queued.
An instance page is requested with:
source_page='landing', v = { count=6 }

It does not exist, so a new name is generated for the (queued) instance page:

Output: <a href="#landing__4">Wait</a>
Queue serving

• When the page is complete, queued instance pages are generated the same way

  <a name="landing__4"><!-- --></a>
  <p>6 seconds to touchdown.</p>
  <ul><li><a href="#landing__5">Wait</a></li></ul>

• They may queue requests for other pages:

  <a name="landing__5"><!-- --></a>
  <p>5 seconds to touchdown.</p>
  <ul><li><a href="#landing__6">Wait</a></li></ul>
-- Created pages by page name:

landing
  landing__1  {}  
  landing__2  { count=8, }  
  landing__3  { count=7, }  
  landing__4  { count=6, }  
  landing__5  { count=5, }  
  landing__6  { count=4, }  
  landing__7  { count=3, }  
  landing__8  { count=2, }  
  landing__9  { count=1, }  
  landing__10 { count=0, }
4 levels of Lua

- The Medusa compiler
- The source pages
- The link functions
- The configuration file

- The levels run in different environments
- Source code is executed during compilation rather than at runtime (metaprogramming, sort of)
Wolf, goat and cabbage

This is the river bank nearest to home; I have to bring all three items undamaged to the other bank.

Here is the wolf I have to ferry across the river. Here is the goat I have to ferry across the river. Here is the cabbage I have to ferry across the river.

- Take the wolf
- Take the goat
- Take the cabbage
- Cross the river alone

CC BY-SA 3.0 Jan Nijendijk
Instance pages created

- Unreachable pages are not created
- No need for pruning

-- Pages by name:

5  farside
1  lost
5  nearside
20  river
1  start
1  won
<table>
<thead>
<tr>
<th>River</th>
<th>Cabbage</th>
<th>Goat</th>
<th>Wolf</th>
</tr>
</thead>
<tbody>
<tr>
<td>River_1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>River_2</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>River_3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>River_4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>River_5</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>River_6</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>River_7</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>River_8</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>River_9</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>River_10</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>River_11</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>River_12</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>River_13</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>River_14</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>River_15</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>River_16</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>River_17</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>River_18</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>River_19</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>River_20</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Debugging

[farside_1, { cabbage=1, goat=2, wolf=1, }]

On the river

This is the river bank nearest to the market.

I brought the goat here.

- Ferry back the goat [river_2, { cabbage=1, goat=3, wolf=1, }]
- Cross the river alone [river_5, { cabbage=1, goat=2, wolf=1, }]

32.
Performance

- Times on an old Pentium 4, including reporting:
  - My game-ebook (943 / 5817 pages): 1.64 s
  - 1 / 1000 pages: 0.11 s
  - 1 / 10k pages: 0.94 s
  - 1 / 100k pages: 9.64 s
  - 1 / 1M pages: 96.8 s
- Almost O(n), no worst case
- Reading / post-processing takes much longer
Use of environments

- Lua 5.1: setfenv()
- Lua 5.2: _ENV and textual substitution (works, but not a perfect solution)
- Could be done with a proxy environment, just switching the vtable, or with an upvalue
- Should preserve cross-page insulation
- Should also allow common sub-functions
- Should control access to vtable, functions, etc.
- Many ways to solve problems in Lua!
Possible improvements

- Common functions callable from pages
- Automated timers
- Subpages with return stack
- Random choices (simulation)

- Better HTML (esp. classes)
- Make it production-ready (error handling etc.)
- Add user-friendly GUI editor
That’s all, folks

Medusa compiler and samples at:

http://www.erix.it/medusa.html

Some images are taken from “Interactive fiction & ebooks” (Enrico Colombini, quintadicopertina)