TI-*nspire Lua*

ADVANCED SCRIPTING TECHNIQUES

Adrien BERTRAND (« Adriweb »)

T³ Conference 2014 - March 8th

Table of Contents

Code Optimization

Lua Performance Benchmarks

Credits *:*

« Caspring » website's wiki (now closed)

Lua.org

Code Optimization

Lua Performance Benchmarks

- Localize your functions

```
for i = 1, 1000000 do
   local x = math.sin(i)
end
```

The following code is 30% faster :

```
local sin = math.sin
for i = 1, 1000000 do
   local x = sin(i)
end
```

Accessing global variables takes more time than accessing local ones. Always localize your functions !

Code Optimization

Lua Performance Benchmarks

- Tables optimization

```
for i = 1, 1000000 do
   local a = {}
   a[1] = 1; a[2] = 2; a[3] = 3
end
```

Help Lua know more about the tables you're going to use !

The following code is almost 3x faster:

```
for i = 1, 1000000 do
   local a = {true, true, true}
   a[1] = 1; a[2] = 2; a[3] = 3
end
```

Code Optimization

Lua Performance Benchmarks

- More Tables optimization

Before :

```
polyline = {
  { x = 10, y = 20 },
  { x = 15, y = 20 },
  { x = 30, y = 20 },
  ...
}
```

Better :

```
polyline = {
  { 10, 20 },
  { 15, 20 },
  { 30, 20 },
  ...
}
```

Best :

```
polyline = {
  x = { 10, 15, 20… },
  y = { 20, 20, 20… }
}
```

Again, help Lua know more about the tables you're going to use !
Avoid useless rehashes when possible !

Code Optimization

Lua Performance Benchmarks

- Other tricks

  - ✓ Don't use `unpack()` in time-critical code
    - ➢ Unpack them yourself ;-) (get values one by one)
  - ✓ Don't use `math.max/min()` on big tables in time-critical code
    - ➢ Prefer looping through the list and using comparisons
  - ✓ Don't use `math.fmod()` for positive numbers
    - ➢ Use the % operator. (On negative numbers, use `math.fmod`, though)

Code Optimization

Lua Performance Benchmarks

- Other tricks

  - ✓ Think twice before using `pairs()` or `ipairs()`
    - ➢ When you know the bounds/keys, prefer a simple `for i=1,x` loop
  - ✓ Avoid `table.insert()` when inserting at the end
    - ➢ Instead, use something like `tbl[#tbl+1] = 42`
  - ✓ When possible, use *closures*
    - ➢ (Powerful concept behind functions [returning] in another function)
      More info : http://www.lua.org/pil/6.1.html

Code Optimization

Tips and Tricks

Code Optimization

Tips and Tricks

- Indentation : a prerequisite

```
3572 local dirty_exit = true
3573 local tosolve
3574 local couldnotsolve = {}
3575
3576 local loops = 0
3577 while dirty_exit do
3578 loops = loops + 1
3579 if loops == 100 then error("too many loops!") end
3580 dirty_exit = false
3581
3582 for i, formula in ipairs(Formulas) do
3583
3584 local skip = false
3585 if couldnotsolve[formula] then
3586 skip = true
3587 for k, v in pairs(known) do
3588 if not couldnotsolve[formula][k] then
3589 skip = false
3590 couldnotsolve[formula] = nil
3591 break
3592 end
3593 end
3594 end
3595
3596 if ((not cid) or (cid and formula.category == cid)) and
```

```
3572    local dirty_exit = true
3573    local tosolve
3574    local couldnotsolve = {}
3575
3576    local loops = 0
3577    while dirty_exit do
3578        loops = loops + 1
3579        if loops == 100 then error("too many loops!") end
3580        dirty_exit = false
3581
3582        for i, formula in ipairs(Formulas) do
3583
3584            local skip = false
3585            if couldnotsolve[formula] then
3586                skip = true
3587                for k, v in pairs(known) do
3588                    if not couldnotsolve[formula][k] then
3589                        skip = false
3590                        couldnotsolve[formula] = nil
3591                        break
3592                    end
3593                end
3594            end
3595
3596            if ((not cid) or (cid and formula.category == cid))
```

Code Optimization

Tips and Tricks

- Simplify your code

```
38 ccircle1=circle(2*w/30,h/10,w/30)
39 ccircle2=circle(2*w/30,h/10,w/30)
40 ccircle3=circle(2*w/30,h/10,w/30)
41 ccircle4=circle(2*w/30,h/10,w/30)
42 ccircle5=circle(2*w/30,h/10,w/30)
43
44 hcircle1=circle(2*w/30,h/2,w/50)
45 hcircle2=circle(2*w/30,h/2,w/50)
46 hcircle3=circle(2*w/30,h/2,w/50)
47 hcircle4=circle(2*w/30,h/2,w/50)
48 hcircle5=circle(2*w/30,h/2,w/50)
49 hcircle6=circle(2*w/30,h/2,w/50)
50 hcircle7=circle(2*w/30,h/2,w/50)
51 hcircle8=circle(2*w/30,h/2,w/50)
52 hcircle9=circle(2*w/30,h/2,w/50)
53 hcircle10=circle(2*w/30,h/2,w/50)
54 hcircle11=circle(2*w/30,h/2,w/50)
55 hcircle12=circle(2*w/30,h/2,w/50)
56
57 Objects={ccircle1,ccircle2,ccircle3,ccircle4,ccircle5,hcircle1,
58          hcircle2,hcircle3,hcircle4,hcircle5,hcircle6,hcircle7,
59          hcircle8,hcircle9,hcircle10,hcircle11,hcircle12}
```

```
38 Objects = {}
39 for i = 1, 5 do
40     Objects[i] = circle(2*w/30,h/10,w/30)
41 end
42
43 for i = 6, 17 do
44     Objects[i] = circle(2*w/30,h/2,w/50)
45 end
```

Code Optimization

Tips and Tricks

Metatables

A metatable is a table which can change the behavior of the table it's attached to.

```lua
t = {}                    -- our normal table
mt = {}                   -- our metatable (empty for now)
setmetatable(t, mt)       -- sets mt to be t's metatable
getmetatable(t)           -- this will return mt
```

(same as `t = setmetatable({}, {})`)

```lua
t = setmetatable({}, {
                __index = function(t, key)
                              return (key == "foo" and 0 or t[key])
                          end
                })
```

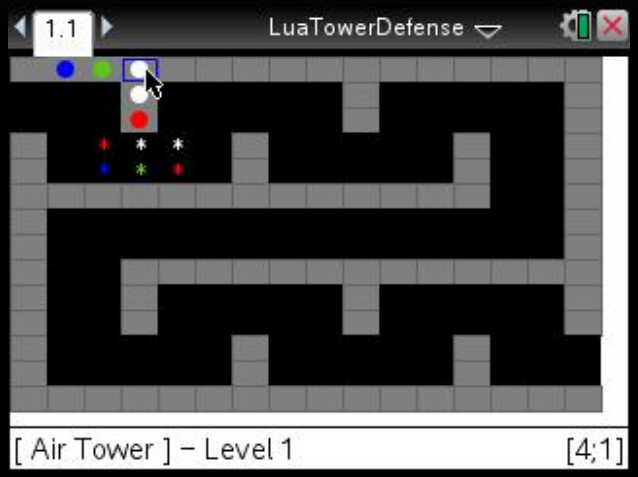
Code Optimization

Tips and Tricks

Metatable example

```
testmap = { {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
            {8,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1},
            {8,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1},
                              [...]
            {1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,9},
            {1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,9},
            {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1} }

setmetatable( testmap, { __index = {1} } )
        -- makes it so undefined areas will be walls (1).
```




```
setmetatable( self.theTypes,  { __index =
                                    function(tbl, key)
                                       return tbl[(key%(#tbl))+1]
                                    end
                              } )
```

Code Optimization

Tips and Tricks

More fun with metatables

Operator overloading

__**add**: Addition (+)
__**sub**: Subtraction (-)
__**mul**: Multiplication (*)
__**div**: Division (/)
__**mod**: Modulos (%)
__**unm**: Unary - (negation)
__**concat**: Concatenation (..)
__**eq**: Equality (==)
__**lt**: Less than (<)
__**le**: Less than or equal to (<=)

A table that supports the multiplication operator (*) :

```
t = setmetatable({ 1, 2, 3 }, {
      __mul = function(t, nbr)
                  local res = {}
                  for k, v in pairs(t) do
                      res[k] = t[k] * nbr
                  end
                  return res
              end })

t = t * 2   -- gives : { 2, 4, 6 }
```

Code Optimization

Tips and Tricks

Memoization :

Storing the result of some computation for a given input so that, when the same input is given again, the script simply reuses that previous result.

```
function memoize(f)
     local mem = {}              -- memoizing table
     setmetatable(mem, {__mode = "v"}) -- weak table
     return function (x)   -- new memoizing version of 'f'
          local r = mem[x]
          if r == nil then    -- any previous result ?
               r = f(x)       -- calls original function
               mem[x] = r     -- store result for reuse
          end
          return r
     end
end

loadstring = memoize(loadstring)
```

Code Optimization

NSpire-Lua Specific Things

Code Optimization

Nspire-Lua Specific Things

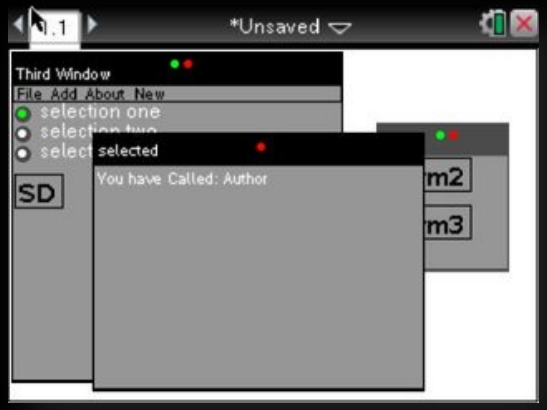
- Do **not** do anything else than drawing in `on.paint()`
- Particularly, here's what you should **avoid** in `on.paint()`
  - `image.new()`, `image.copy()`, `image.rotate()` ← Way too slow
  - Events definition (like `on.enterKey()`, etc.) ← Not appropriate
  - `platform.window:invalidate()` ← Useless here

Reminder : except if you're dealing with animations, try not to refresh the screen a lot, but only when needed, it will save CPU and memory !

Code Optimization

Nspire-Lua Specific Things

- Use Classes
  - No need to state the obvious on the advantages
- Use a screen manager / GUI Toolkit



WZGUILib

TiMasterBox

ETK

***Soon*, you'll be able to use "OpenSpire", an online IDE with a GUI editor !**

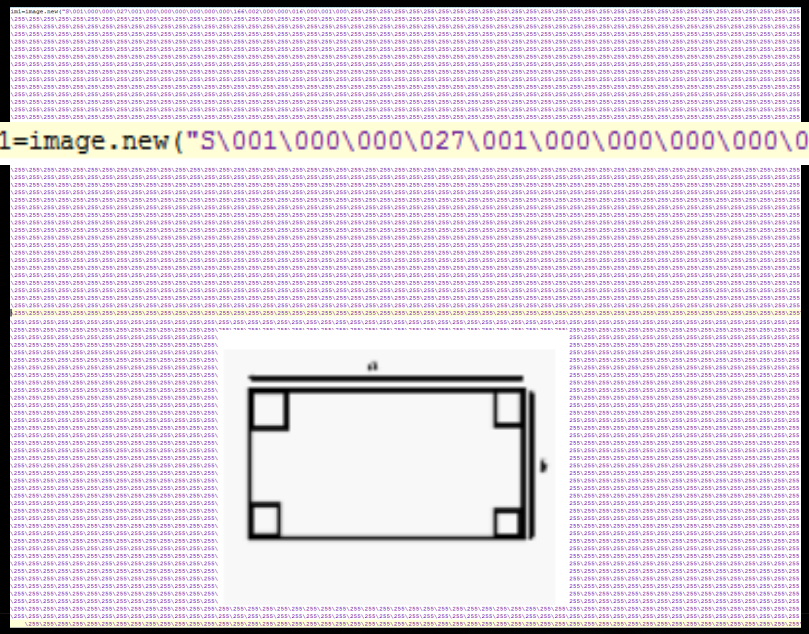
Code Optimization

Nspire-Lua Specific Things

- Avoid images when possible

Images

```
im1=image.new("S\001\000\000\027\001\000\000\000\000\000
```



6 KB

6 images = 26KB

Polygons

```
polys = {
        {
            {-1.8, -1, 1.8, -1, 1.8, 1, -1.8, 1, -1.8, -1},
            {-1.8, -1, -1.6, -1, -1.6, -.8, -1.8, -.8, -1.8, -1},
            {1.6, -1, 1.8, -1, 1.8, -.8, 1.6, -.8, 1.6, -1},
            {-1.8, 1, -1.6, 1, -1.6, .8, -1.8, .8, -1.8, 1},
            {1.6, 1, 1.8, 1, 1.8, .8, 1.6, .8, 1.6, 1},
        },
```

1 KB

6 polygons = 2KB

Code Optimization

Nspire-Lua Specific Things

But if you really need images...

- `image.new(…)` ← Only once (per image)
- `image.copy()`, `image.rotate()` ← Once per needed variation (might be re-done in on.resize)

New in 3.6 :

- Use the "Resources" tab in the Software to import images *(better for the script editor : no lags due to huge strings!)*
- Then, access your data by iterating through the **`_R.IMG`** table :

```
myImages = {}
for name, resource in pairs(_R.IMG) do
        myImages[name] = image.new(resource)
end
```

Code Optimization

Nspire-Lua Specific Things

- Static Width or Height

```
if ww == 793 then
   gc:setFont("sansserif","bi",20)
else
   gc:setFont("sansserif","bi",11)
end
```

```
if ww > 320 then
   gc:setFont("sansserif","bi",20)
else
   gc:setFont("sansserif","bi",11)
end
```

Other examples :

```
gc:setFont("sansserif", "bi", math.min(255, math.max(6, ww/25))
```

Re-use later :

```
f_medium = math.min(255, math.max(6, ww/25))
…
gc:setFont("sansserif", "bi", f_medium)
```

Code Optimization

Nspire-Lua Specific Things

Use `on.varChange()` instead of recalling variables in `on.timer()`

```
function on.timer()
     ch = var.recall("ch")
     platform.window:invalidate()
end

timer.start(0.1)
```

```
function on.construction()
    local v = {"quadrilateral"}
    vars = {}
    for i, k in ipairs(v) do
        vars[k] = var.recall(k) or 1
        var.monitor(k)
    end
end

function on.varChange(list)
    for _, k in pairs(list) do
        if vars[k] then
            vars[k] = var.recall(k) or 1
        end
    end
    platform.window:invalidate()
end
```

Advanced techniques in practice

Adding your own functions to GC

Advanced techniques in practice

Adding your own functions to gc

Before

```
function on.paint(gc)
    gc:drawString("hello", 5, 5)
    setColor("red", gc)
    fillCircle(50, 100, 30, gc)
    setColor("white", gc)
    drawPixel(50, 100, 30, gc)
    gc:drawString("test", 50, 5)
end
```

Messy...

After

```
function on.paint(gc)
    gc:drawString("hello", 5, 5)
    gc:setColor("red")
    gc:fillCircle(50, 100, 30)
    gc:setColor("white")
    gc:drawPixel(50, 100, 30)
    gc:drawString("test", 50, 5)
end
```

Clean !

Advanced techniques in practice

Adding your own functions to GC

The « magic » :

```
function AddToGC(key, func)
    local gcMetatable = platform.withGC(getmetatable)
    gcMetatable[key] = func
end

function fillCircle(gc, x, y, r)
    ...
end
AddToGC("fillCircle", fillCircle)
```

Advanced techniques in practice

Using a Screen Manager

Demo

Advanced techniques in practice

Using a Screen Manager

```
local triggeredEvent
function eventDistributer(...)
        local currentScreen = GetScreen()
        if currentScreen[triggeredEvent] then
                currentScreen[triggeredEvent](currentScreen, ...)
        end
end

local eventCatcher = {}
eventCatcher.__index = function (tbl, event)
                        triggeredEvent = event
                        return eventDistributer
                    end
setmetatable(on, eventCatcher)
```

Alternative Lua Editors

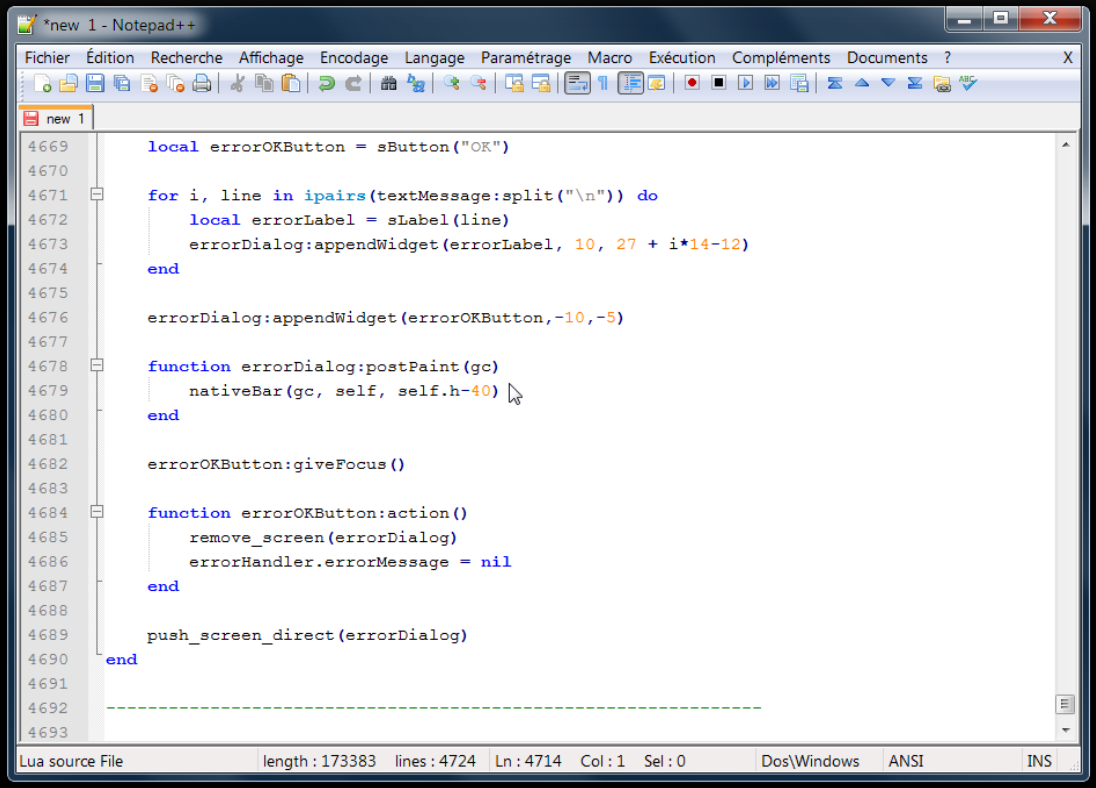
Simple Code Editors

Alternative Lua Editors

Simple Code Editors

Notepad++

- Windows only
- Lots of languages
- Nice set of features
- Plugins

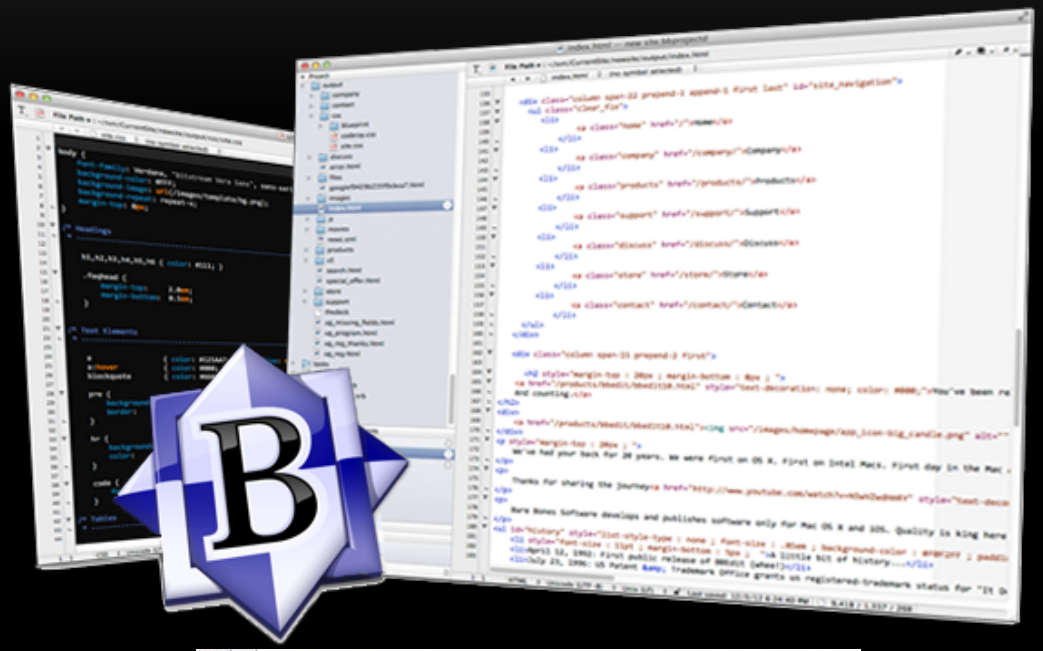


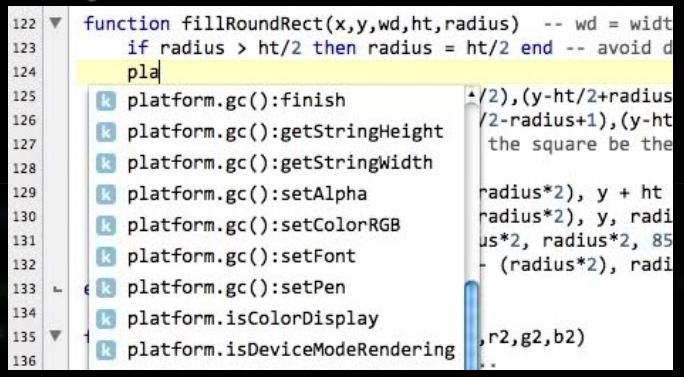
Alternative Lua Editors

Simple Code Editors

TextWrangler / BBEdit

- Mac only
- Lots of languages
- Nice set of features
- Plugins (for BBEdit)



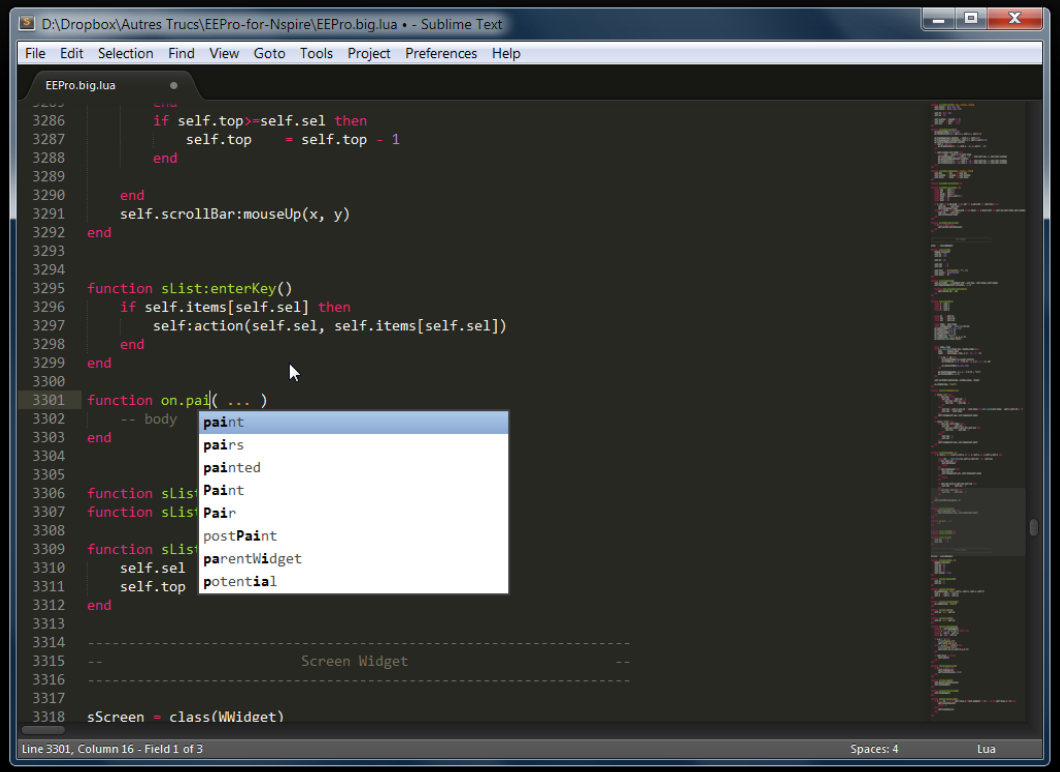


Alternative Lua Editors

Simple Code Editors

SublimeText

- Windows/Mac/Linux
- Lots of languages
- Nice set of features
- Customizable
- Plugins



Alternative Lua Editors

An IDE : IntelliJ IDEA

Alternative Lua Editors

An IDE : IntelliJ IDEA

Integrated

Development

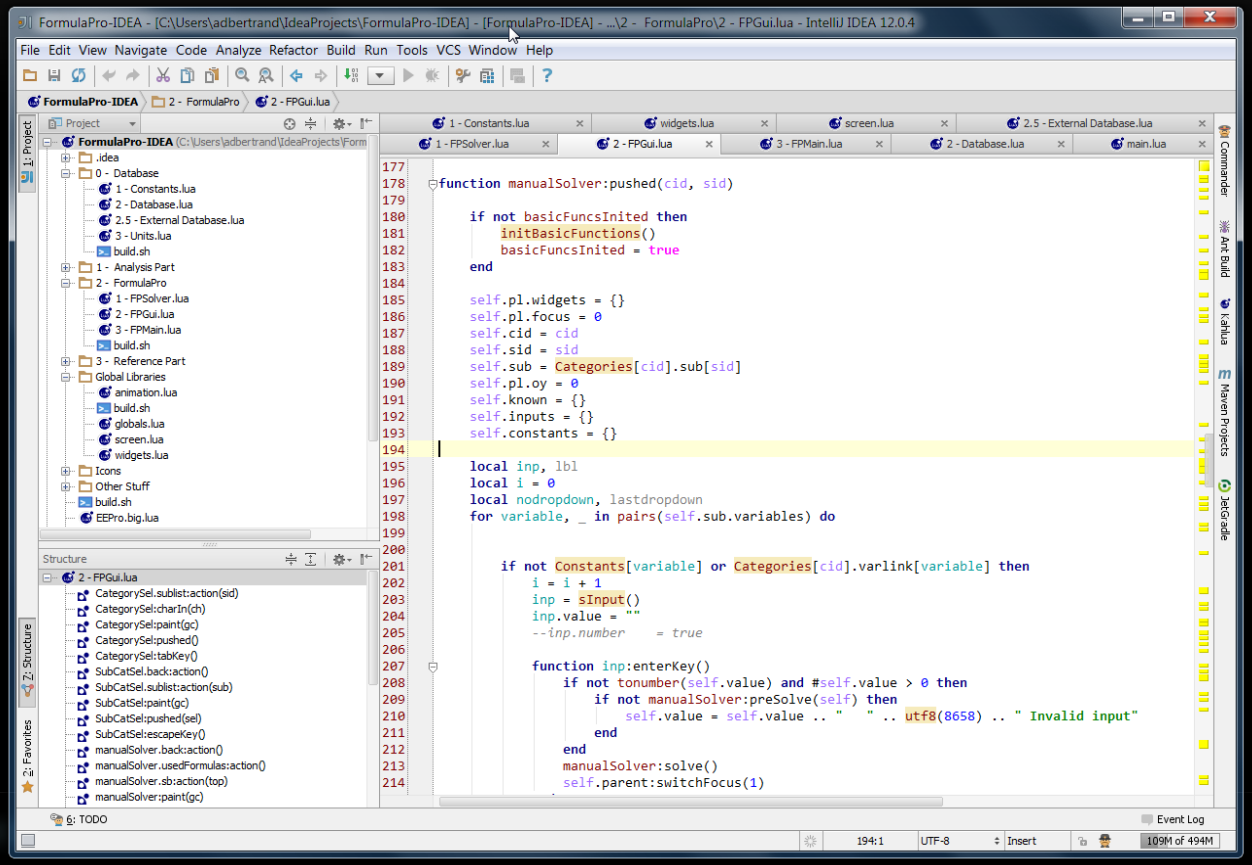
Environment

Windows/Mac/Linux

Free !

A “million” of features

Many plugins



Alternative Lua Editors

An IDE : IntelliJ IDEA

Setting up the beast

1. Download and install Intellij
    1. http://www.jetbrains.com/idea/
    2. Select the Free Edition
2. Install its Lua plugin
    1. Go to Settings > Install Plugin
    2. Add the “Lua” one in Browse
3. Setup the Nspire-Lua addon
    1. Download it on TI-Planet
    2. Extract it somewhere
    3. Set it as the project’s SDK

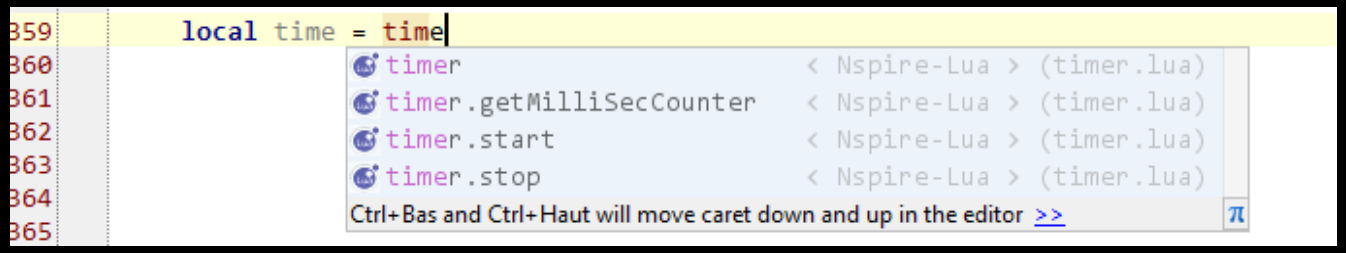
+

Alternative Lua Editors

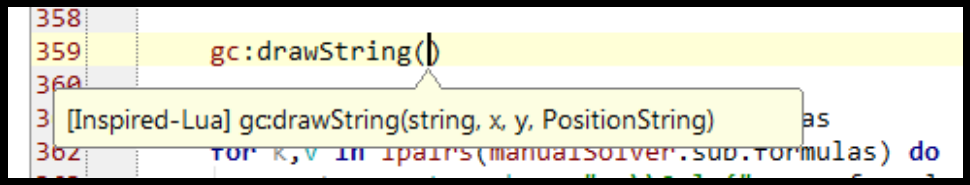
An IDE : IntelliJ IDEA

Once all that is done, here's what you'll have :

- ✓ Nspire-Lua specific auto-completion



- ✓ Inline syntax help for common methods



- ✓ Dynamic API help frame with info from Inspired-Lua

MANY THANKS TO…

Jérémy Anselme ("Levak")
Jim Bauwens

Steve Arnold
John Powers

Inspired-Lua.org
TI-Planet.org

ANY QUESTIONS ?

(original images by TI)