Improve your TI-*nspire* LUA programming skills

Jeremy ANSELME (« Levak »)

Adrien BERTRAND (« Adriweb »)

Table of Contents

Common mistakes

General coding Guidelines

General coding Guidelines

- Indentation

```
26 function on.paint(gc)
27 local h=platform.window.height()
28 local w=platform.window.width()
29 ch=(var.recall("quadrilateral")or 1)
30 if ch==1 then
31 im=im1
32 elseif ch==2 then
33 im=im2
34 elseif ch==3 then
35 im=im3
36
37 elseif ch==4 then
38 im=im4
39 elseif ch==5 then
40 im=im5
41 else
42 im=im6
43 end
44 local pw=image.width(im)
45 local ph=image.height(im)
46 local im=image.copy(im,pw/3,ph/3)
47 local pw=image.width(im)
```

```
26 function on.paint(gc)
27     local h=platform.window.height()
28     local w=platform.window.width()
29     ch=(var.recall("quadrilateral")or 1)
30     if ch==1 then
31         im=im1
32     elseif ch==2 then
33         im=im2
34     elseif ch==3 then
35         im=im3
36
37     elseif ch==4 then
38         im=im4
39     elseif ch==5 then
40         im=im5
41     else
42         im=im6
43     end
44     local pw=image.width(im)
45     local ph=image.height(im)
46     local im=image.copy(im,pw/3,ph/3)
47     local pw=image.width(im)
```

General coding Guidelines

- Give Meaningful Names
  - table = { } text_tbl = { }

    Side note : « table » already exists in the Lua API
  - gc:drawString("x", JJ, 1) str_x
  - JJ=hi*ba str_x = height*base
  - linecount = var.recall("quadrilateral") or 1 quadrilateral
  - a = do() tbl = generateTbl()

General coding Guidelines

- 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}
60
```

```
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
```

General coding Guidelines

- Simplify your code

```
25 local sw = gc:getStringWidth("Structure of Human Eye")
26 local sh = gc:getStringHeight("Structure of Human Eye")
27 gc:setFont("sansserif", "b", 10)
28 gc:setColorRGB(158, 5, 8)
29 gc:drawString("Structure of Human Eye",w/2 - sw/2, h/2 + sh/2-90)
```

```
25 local title = "Structure of Human Eye"
26 gc:setFont("sansserif", "b", 10)
27 local sw = gc:getStringWidth(title)
28 local sh = gc:getStringHeight(title)
29 gc:setColorRGB(158, 5, 8)
30 gc:drawString(title,w/2 - sw/2, h/2 + sh/2-90)
```

Common mistakes

Lua Tips

LUA TIPS

- Tables can be defined with elements inside

```
 9 table = { "RECTANGLE: A quadrilateral in which ",
10           "SQUARE: A quadrilateral with all of its four sides are",
11           "PARALLELOGRAM: A quadrilateral with both pairs",
12           " TRAPEZIUM: A quadrilateral which has a pair of opposite sides parallel",
13           "RHOMBUS: A quadrilateral with pairs of consecutive sides ",
14           "KITE: A quadrilateral with two",
15           "all of its four angles are equal to a right angle",
16           " equal and each of its four angles a right angle",
17           "of opposite sides parallel",
18           " but the other two sides are non parallel",
19           " equal and all angles are not equal ",
20           "pairs of adjacent sides equal",
21 }
```

LUA TIPS

- You are not limited in the number of tables

```
 9 table = { "RECTANGLE: A quadrilateral in which ",
10           "SQUARE: A quadrilateral with all of its four sides are",
11           "PARALLELOGRAM: A quadrilateral with both pairs",
12           " TRAPEZIUM: A quadrilateral which has a pair of opposite sides parallel",
13           "RHOMBUS: A quadrilateral with pairs of consecutive sides ",
14           "KITE: A quadrilateral with two",
15           "all of its four angles are equal to a right angle",
16           " equal and each of its four angles a right angle",
17           "of opposite sides parallel",
18           " but the other two sides are non parallel",
19           " equal and all angles are not equal ",
20           "pairs of adjacent sides equal",
21 }
```

```
gc:drawString(table[linecount],
gc:drawString(table[linecount+6]
```

```
 1 title = {
 2     "RECTANGLE: A quadrilateral in which ",
 3     "SQUARE: A quadrilateral with all of its four sides are",
 4     "PARALLELOGRAM: A quadrilateral with both pairs",
 5     " TRAPEZIUM: A quadrilateral which has a pair of opposite sides parallel",
 6     "RHOMBUS: A quadrilateral with pairs of consecutive sides ",
 7     "KITE: A quadrilateral with two"
 8 }
 9 desc = {
10     "all of its four angles are equal to a right angle",
11     "equal and each of its four angles a right angle",
12     "of opposite sides parallel",
13     " but the other two sides are non parallel",
14     " equal and all angles are not equal ",
15     "pairs of adjacent sides equal"
16 }
```

```
gc:drawString(title[linecount]
gc:drawString(desc[linecount],
```

LUA TIPS

- Think expandable (multiline text example)

  = Avoid copy/paste

```
local sw = gc:getStringWidth("SESSION OBJECTIVE")
local sh = gc:getStringHeight("SESSION OBJECTIVE")
gc:drawString("SESSION OBJECTIVE",(ww-sw)/2,(wh*1/7))

local sw = gc:getStringWidth("1. Why do we need classification?")
local sh = gc:getStringHeight("1. Why do we need classification?")
gc:drawString("1. Why do we need classification?",0,(wh*2/7))

local sw = gc:getStringWidth("2. Dobereniner's triads(RECAP)")
local sh = gc:getStringHeight("2. Dobereniner's triads(RECAP)")
gc:drawString("2. Dobereniner's triads(RECAP)",0,(wh*3/7))

local sw = gc:getStringWidth("3. Newlands law of octave")
local sh = gc:getStringHeight("3. Newlands law of octave")
gc:drawString("3. Newlands law of octave",0,(wh*4/7))

local sw = gc:getStringWidth("4. Practice Questions")
local sh = gc:getStringHeight("4. Practice Questions")
gc:drawString("4. Practice Questions",0,(wh*5/7))

local sw = gc:getStringWidth("5. Stuff You Should Know")
local sh = gc:getStringHeight("5. Stuff You Should Know")
gc:drawString("5. Stuff You Should Know",0,(wh*6/7))
```

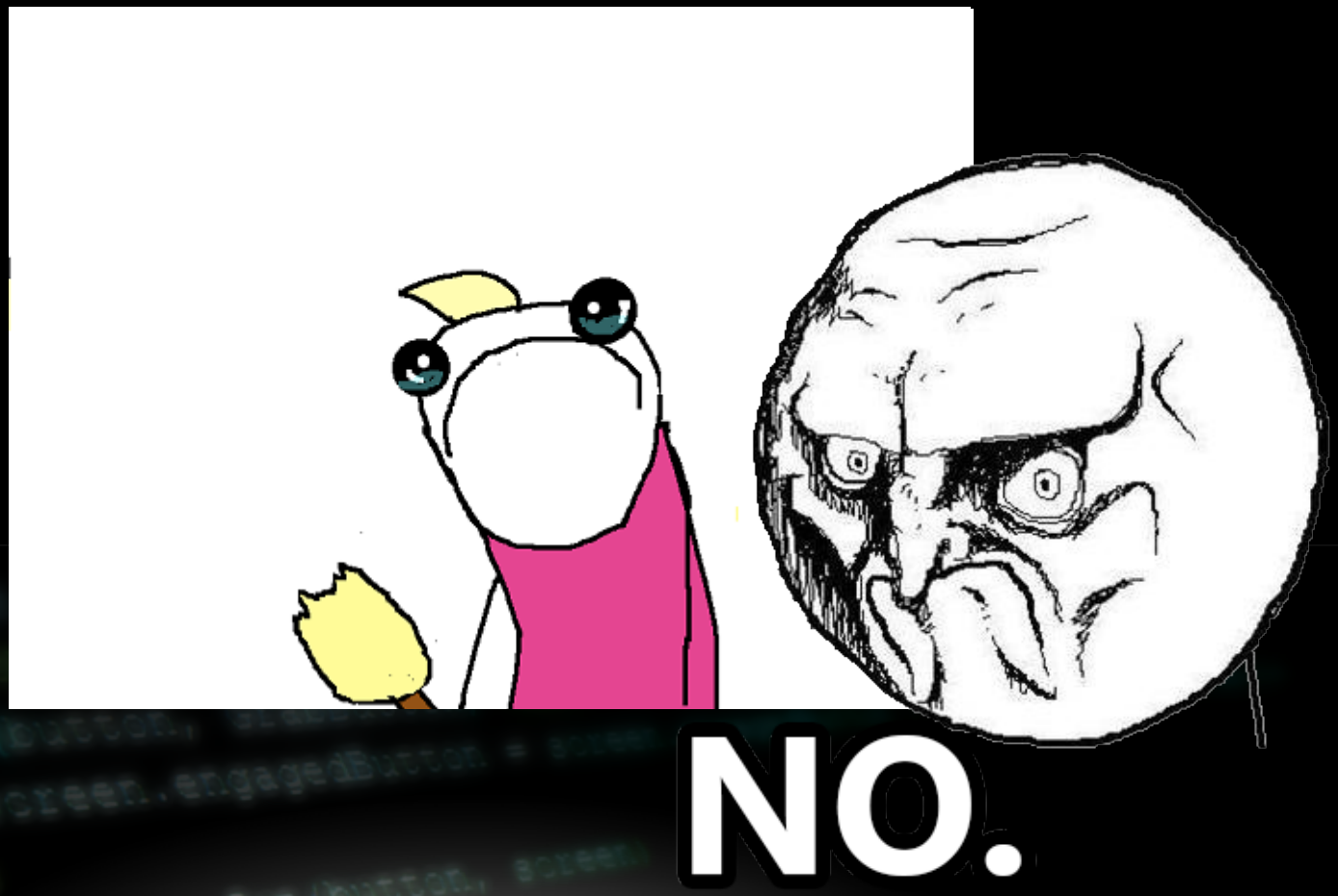
```
local title = "SESSION OBJECTIVE"
local sw = gc:getStringWidth(title)
gc:drawString(title,(ww-sw)/2,(wh*1/7))

text = {"1. Why do we need classification?",
        "2. Dobereniner's triads(RECAP)",
        "3. Newlands law of octave",
        "4. Practice Questions",
        "5. Stuff You Should Know"
}

for i = 1, #text do
    gc:drawString(text[i], 0, wh*(i+1)/7)
end
```

Common mistakes

TI-Nspire Lua API related

TI-Nspire Lua API related

- on.paint() event



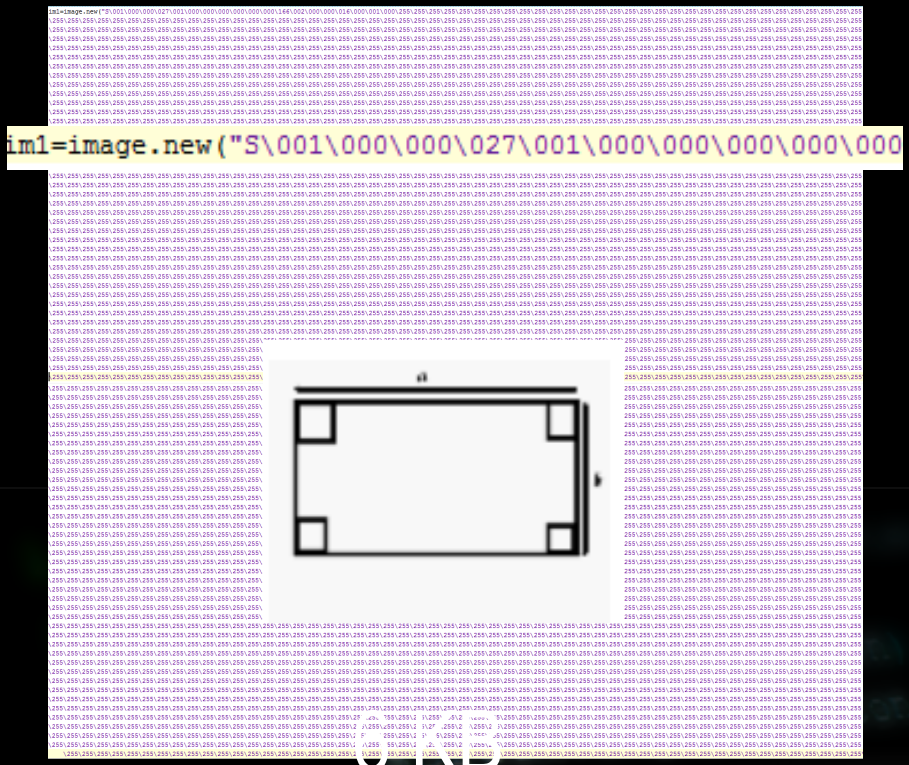
TI-Nspire Lua API related

- 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

TI-NSPIRE LUA API RELATED

- Avoid images when possible

Images



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

TI-NSPIRE LUA API RELATED

- gc:setFont and gc:getStringWidth/Height

```
local str = "At the end of ..."
gc:setFont("sansserif", "r", 10)
local ssw = gc:getStringWidth(str)
local ssh = gc:getStringHeight(str)
gc:drawString(str, 10, h/2 + sh/2-50)
```

```
gc:setFont("sansserif", "r", 10)
gc:drawString("1. Identify the different parts of the eye",10, h/2 + sh/2-25)
gc:setFont("sansserif", "r", 10)
gc:drawString("2. Understand the functionality of each part",10, h/2 + sh/2-0)
gc:setFont("sansserif", "r", 10)
gc:drawString("3. Learn how the eye processes Light",10, h/2 + sh/2+25)
```

TI-Nspire Lua API related

- 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)
```

TI-Nspire Lua API related

- 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.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

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
```

TI-Nspire Lua API related

- string.uchar() for special symbols
  - In 3.1

    in a TI-Nspire Calculator App : ord(“α”) → returns 945
    in the Lua script : string.uchar(945)

    special char here

  - In 3.2

    simply copy/paste the special char in the Lua editor

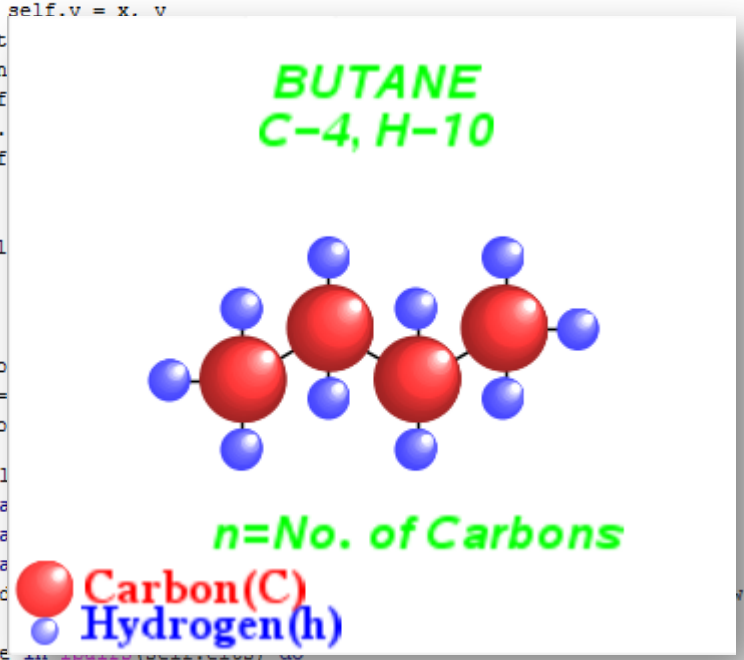
TI-Nspire Lua API related

Make classes !

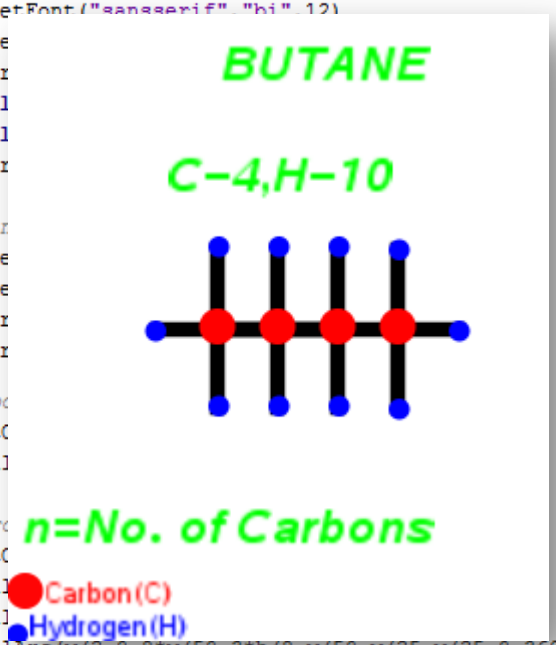
Make classes !

Make classes !

```
----------------------------
-- Levak Â©2012 --------------
-- http://levak.free.fr/ ----
-- levak92@gmail.com --------
----------------------------

Atom = class()
function Atom:init(x, y, r, R, G, B)
    self.x, self.y, self.r = x, y, r
    self.R, self.G, self.B = R, G, B
end

function Atom:paint(gc, ox, oy)
    local x, y, d = (self.x+ox-self.r)*ww, (self.y)*wh-(self.r-oy)*ww, (self.r*2)*ww
    local n = self.r*2*ww
    for c = n, 1, -1 do
        local r, g, b = math.min(255, self.R*1024/c),
                        math.min(255, self.G*1024/c),
                        math.min(255, self.B*1024/c)
        gc:setColorRGB(r, g, b)
        gc:fillArc(x+(n-c)/1.2, y+(n-c)/5, d-n+c, d-n+c, 0, 360)
    end
end

Group = class()
function Group:init(x, y, elts, links, offsets)
    self.x, self.y = x, y
    self.elt
    self.lin
    self.off
    if self.
        self

        end
    else sel
    end
end

function Gro
    ox, oy =
    gc:setCo

    for i, l
        loca
        loca
        loca
        gc:d
    end
    for i, e
```




```
function on.paint(gc)
    w=platform.window:width()
    h=platform.window:height()
    local n=(var.recall("n") or 1)

   local sw=gc:getStringWidth("n=No. of Carbons")
    local sh=gc:getStringHeight("n=No. of Carbons")
    gc:setFont("sansserif","bi",12)
    gc:setColorRGB(0,255,0)
    gc:drawString("n=No. of Carbons",w-1.5*sw,h-1.5*sh)

    gc:setColorRGB(255, 5, 5)
    gc:fillArc(0,h-2*w/15,w/15,w/15,0,360)
    gc:setFont("serif","b",w/30)
    gc:drawString("Carbon(C)",w/15,h-w/15)


    gc:setColorRGB(5, 5, 255)
    gc:fillArc(0,(h-w/25),w/25,w/25,0,360)
    gc:drawString("Hydrogen(H)",w/25,h-w/100)

    if n==1 then    --methane

    local sw=gc:getStringWidth("METHANE")
    local sh=gc:getStringHeight("METHANE")
    gc:setFont("sansserif","bi",12)
    gc:se
    gc:dr
    local
    local
    gc:dr

    --lin
    gc:se
    gc:se
    gc:dr
    gc:dr

--carbo
gc:setC
gc:fill

--hydro
gc:setC
gc:fill
gc:fill
gc:fillArc(w/2-0.8*w/50,3*h/8-w/50,w/25,w/25,0,360)
```



Learn New Concepts

Classes - What Are Classes ?

- In object-oriented programming, a class is a "construct", a "model" that is used to create instances of itself (called "objects").
- A class defines the default properties and methods (functions) of its members.
- Well, let's think of Classes as "families", or groups.
- If I define "Fruit" as a class with a "color" property, and I want to have a banana, I can do that :

```
myBanana = Fruit("yellow")
```

Classes - Why Use Classes ?

- Simplify and optimize your code
  - It will drastically reduce the number of lines you need
  - It will execute faster (better performance overall)
- Simplify your coding habits
  - It's way faster (and easier !) to think with "objects"
  - Want to change a "global" property ? Do it once and for all !
  - You just won't be able to go back to "no classes" ;-)

Classes - How To Create Them?

It's pretty simple. Only a few things are required :

```
Fruit = class()
function Fruit:init(x, y, color)
    self.x, self.y = x, y
    local colorCode = {255, 0, 0}  -- red by default
    if color == "yellow" then colorCode = {255, 255, 0} end
    if color == "green" then colorCode = {0, 255, 0} end
    self.color = colorCode
end
function Fruit:paint(gc)
    gc:setColorRGB(unpack(self.color))
    gc:fillRect(self.x, self.y, 10, 20)
end

myBanana = Fruit(50, 10, "yellow")
myTomato = Fruit(100, 40, "green")

function on.paint(gc)
    myBanana:paint(gc)
    myTomato:paint(gc)
end
```

Class definition

Objects creation

Objects painting

Frameworks

Frameworks

- Using classes we are able to create expandable & ready-to-use libraries
- They contain graphical elements organized in objects/widgets

Our frameworks :

- ETK
- Screen Manager
- Animation

Any Questions ?