Mobile Programming and Multimedia

The Corona/Solar 2D framework

Prof. Ombretta Gaggi
University of Padua

Introduction

Corona is a cross-platform framework which uses a cross-compiled approach

It is essentially centered on games development, but can be used to create applications

For videogames:

- It allows a really natural interaction thanks to several physical effects

For applications:

- It provides a set of widget for interaction

For everybody:

- Community extremely active, highly responsive

Re-branding

Since May 1st, 2020, Corona is not a commercial project anymore but an open-source project. The new name is Solar 2D

SOLAR2D

Awesome 2D Game Engine

GitHub | Downloads | Support | Forums | Discord | Documentaton | Try Now!

Lua language

Lua

Corona works with the language Lua. It is a scripting language

It is used in Corona but even in other games developed with a native approach (WoW – interface, Angry Birds, etc.)

Variables

There are three types of variables:

- string
- number
- bool

No need for variable declaration:

```
string = "corona"
number = 3
boolean = true
```

Pay attention to typing errors!

Operators

Lua supports logical, relational, and arithmetical operators

Arithmetical	Relational	Logical	String operators
+	==	and	..
-	~=	or	#
/	>	!	
*	<		
%	>=		
^	<=		
- (-x)			

Tables - 1

Arrays in Lua are called tables (table)

They allow to store data of different types in different positions (the first position is 1)

```
myTable = {}
myTable[1] = "Lua"
myTable[2] = "5.3"
```

Tables - 2

Tables can store even associative arrays

```
myTable = {}
myTable["language"] = "Lua"
myTable["version"] = "5.3"
```

For loop


```
for i=1,10 do
    tortoise.x = tortoise.x +1
end
```

While loop


```
while !arrived do
    tortoise.x = tortoise.x +1
end
```

if … then … else


```
if (onTime) then
    victory = true
else
    victory = false
end
```

There is also the elseif clause

Functions


```
showPosition()

function showPosition()
    -- show turtle position
end

function bonusMalus(value)
    turtle.score = turtle.score + value
end
```

Local and global variables

Variables are *global* by default

Local variables must be declared with the keyword *local*

A local variable exists only inside the block where it is declared

```
function whatever()
   local x = 1
end
-- [[ x does not exist ]]--
```

```
function whatever()
   x = 1
end
--[[ x still exists ]]--
```

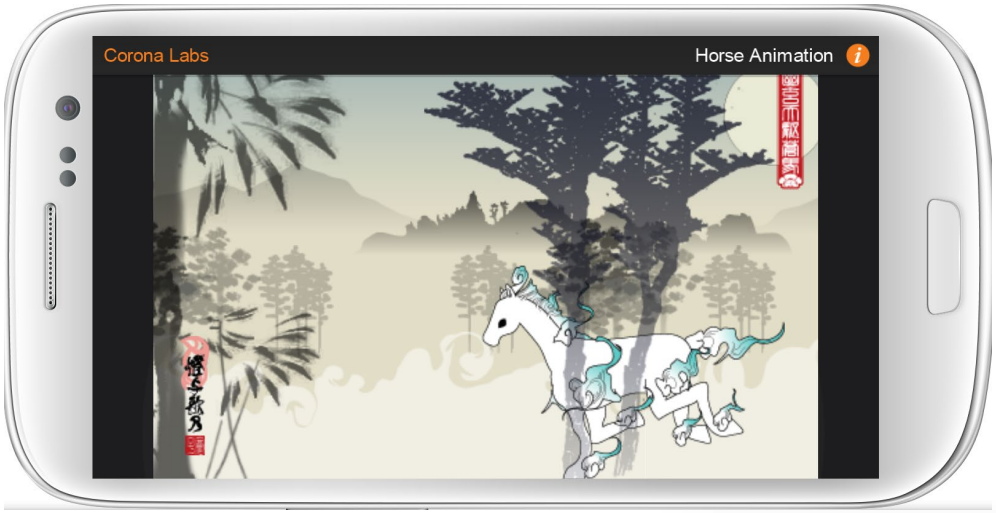
A Corona project

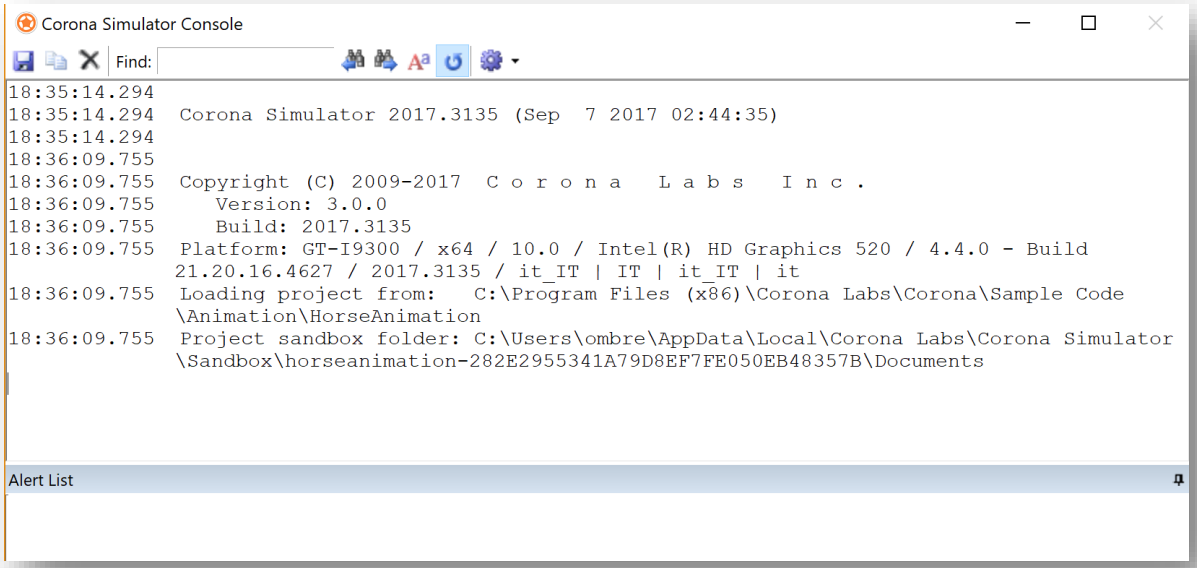
Creating a Corona project

Every project developed using Corona has these important files:

- build.settings
- main.lua: main file of each Corona application
- config.lua
- Default image

Tools


Corona Labs
Horse Animation


Corona Simulator Console
Find:
18:35:14.294
18:35:14.294 Corona Simulator 2017.3135 (Sep 7 2017 02:44:35)
18:35:14.294
18:36:09.755
18:36:09.755 Copyright (C) 2009-2017 C o r o n a L a b s I n c .
18:36:09.755 Version: 3.0.0
18:36:09.755 Build: 2017.3135
18:36:09.755 Platform: GT-I9300 / x64 / 10.0 / Intel(R) HD Graphics 520 / 4.4.0 - Build 21.20.16.4627 / 2017.3135 / it_IT | IT | it_IT | it
18:36:09.755 Loading project from: C:\Program Files (x86)\Corona Labs\Corona\Sample Code\Animation\HorseAnimation
18:36:09.755 Project sandbox folder: C:\Users\ombre\AppData\Local\Corona Labs\Corona Simulator\Sandbox\horseanimation-282E2955341A79D8EF7FE050EB48357B\Documents
Alert List

Draw on the interface

Corona provides a set of functions to print or draw something on the screen

print(''I am a tortoise")

local tarty = display.**newImageRect**("tortoise.png",width,height)

local circle = display.**newCircle**(centerX, centerY, radius)

local rect = display.**newRect**(x, y, width, height)

local rRect = display.**newRoundedRect**(x, y, width, height, radius)

local line = display.**newLine**(x1, y1, x2, y2)

local polygon = display.**newPolygon**(x, y, vertexes) → vertexes is an array with the other vertexes

Events

The events can be managed using events handler that are linked to an object

- torty:**addEventListener**("tap", tapOnTorty)
- Runtime:**addEventListener**("touch",myListener)

Each event can be in 4 different states:

- began
- moved
- cancelled
- ended

An example with events - 1


```
function print(event)
    print("tap")
end

local torty =
    display.newImageRect("tortoise.png",70,99)
torty.x = 100
torty.y = 100
torty:addEventListener("tap",print)
```

An example with events - 2


```
function move(event)
    if (event.phase == "began") then
        torty.x=event.x
        torty.y=event.y
    end
end

local torty =
    display.newImageRect("tortoise.png",70,99)
torty.x = 100
torty.y = 100
Runtime:addEventListener("touch",move)
```

Widget

Corona provides a widget library to rapidly create the interaction interface. It allows to create:

- Picker wheel
- Stepper widget
- Progress bar
- Radio button

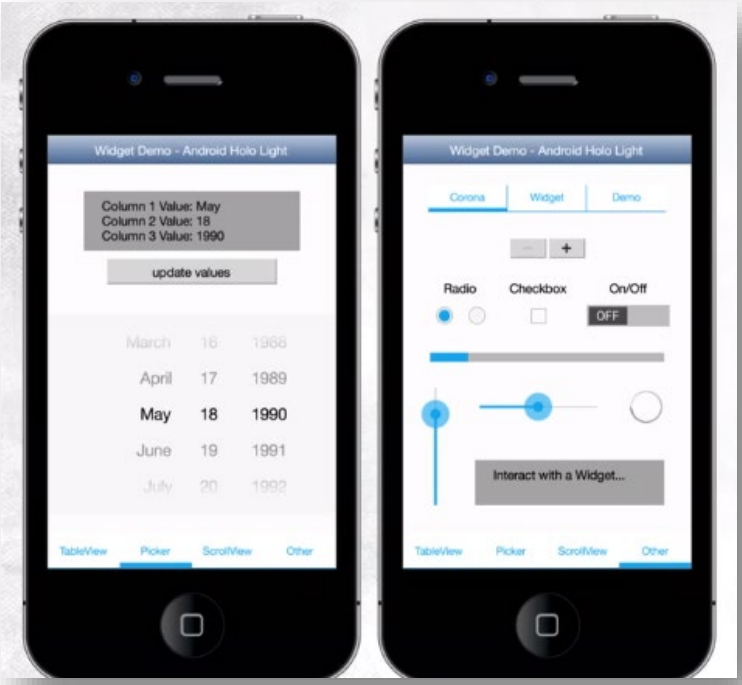
```
Local widget = require("widget")
```

Example: a button


```
button = widget.newButton{
    width = 100,
    height = 50,
    defaultFile = "button.png",
    overFile = "pushedButton.png",
    label = "text",
    onEvent = functionCalledOnTouch
}
```

Other functions

- widget.newPickerWheel()
- widget.newProgressView()
- widget.newScrollView()
- widget.newSegmentedControl()
- widget.newSlider()
- widget.newSpinner()
- widget.newStepper()
- widget.newSwitch()
- widget.newTabBar()
- widget.newTableView()
- widget.setTheme()



Physics library

It is a library that applies physics rules and laws to objects (bodies), in particular, gravity and collision

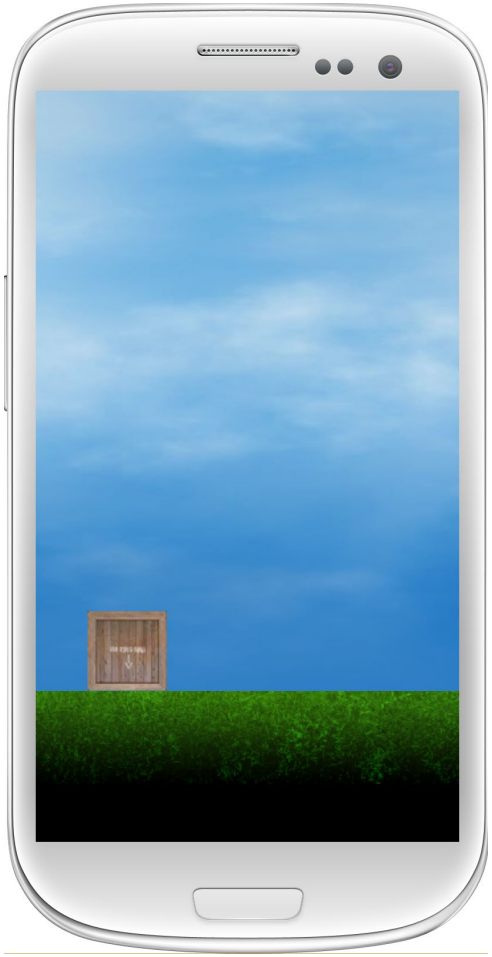
Each object has 3 essential properties:

- Density
- Friction
- Bounce

There are three types of bodies (objects):

- Dynamics: respond to gravity
- Static: do not move, as if they have an infinite mass
- Kinetics: they move depending on their speed, but do not respond to gravity

Simple example



```
local physics = require( "physics" )
physics.start()

local sky = display.newImage( "bkg_clouds.png", 160, 195 )
local ground = display.newImage( "ground.png", 160, 445 )

physics.addBody( ground, "static", { friction=0.5,
                                     bounce=0.3 } )
local crate = display.newImage( "crate.png", 180, -50 )
crate.rotation = 5
physics.addBody( crate, { density=3.0, friction=0.5,
                                     bounce=0.3 } )
```

More complex example

Code – positioning


```
local background = display.newImage( "bricks.png", centerX, centerY, true )
local floor = display.newImage( "floor.png", 0, 280, true )
physics.addBody( floor, "static", { friction=0.5 } )
local stand = display.newImage( "stand.png", 170, 220 )
physics.addBody( stand, "static", { friction=0.5 } )

local cans = {}
for i = 1, 7 do
   for j = 1, 8 do
      cans[i] = display.newImage( "soda_can.png", 190 + (i*24), 220 - (j*40) )
      physics.addBody( cans[i], { density=0.2, friction=0.1, bounce=0.5 } )
   end
end
```

Code - bricks


```
local bricks = {}
local n = 0

local function throwBrick()
	n = n + 1
	bricks[n] = display.newImage( "brick.png", -20, 140 - (n*20) )
	physics.addBody( bricks[n], { density=3.0, friction=0.5,
bounce=0.05 } )

	bricks[n].isBullet = true
	bricks[n].angularVelocity = 100
	bricks[n]:applyForce( 1200, 0, bricks[n].x, bricks[n].y )
end
```

Code - timing


```
local function start()
    timer.performWithDelay(360, throwBrick, 3)
end

timer.performWithDelay(800, start)
```

Composer

Composer is the scenes manager. The scenes generally represent what is shown on the screen:

- An application can have
  - A scene for the main menu,
  - One to choose the character,
  - One for settings and
  - One scene for the game.

Composer APIs allows to create, connect and manage different components of the application

The scenes

Each scene is a LUA file

- main.lua is the file that starts the application

Composer allows to organize scenes in different files and manage transitions between scenes

- Manages memory usage, events, etc.

Each scene has four events with associated event managers:

- create: adds the object on the screen and create the listeners to the events
- show: starts the timers and animations
- hide: stops objects and timers
- destroy: saving before exit

Scenes and events

show and hide events for the scenes pass through two different states:

- *show* event:
  - will: just before the scene becomes active
  - did: just after the scene is presented on the screen
- *hide* event:
  - will: just before the scene is deactivated
  - did: just after the scene was removed from the screen

main.lua


```
local composer = require "composer"
local widget = require "widget"

-- load first scene
composer.gotoScene( "scene1", "fade", 400 )

-- draw buttons or the interface
```

scene1.lua


```
local composer = require( "composer" )
local scene = composer.newScene()

-- functions definition

scene:addEventListener( "create", scene )
scene:addEventListener( "show", scene )
scene:addEventListener( "hide", scene )
scene:addEventListener( "destroy", scene )

return scene
```

scene1.lua - create


```
function scene:create( event )
	local sceneGroup = self.view

	image = display.newImage( "bg.jpg" )
	image.x = display.contentCenterX
	image.y = display.contentCenterY
	sceneGroup:insert( image )
	image.touch = onSceneTouch

	text1 = display.newText( "Scene 1", 0, 0, native.systemFontBold, 24 )
	text1:setFillColor( 255 )
	text1.x, text1.y = display.contentWidth * 0.5, 50
	sceneGroup:insert( text1 )
	-- draw other texts

	print( "\n1: create event")
end
```

scene1.lua - show


```
function scene:show( event )
    local phase = event.phase
    if (phase=="did") then
        print( "1: show event, phase did" )
        -- if the scene has been already drawn,
        --    I remove the previous one
        composer.removeScene( "scene4" )
        -- memory usage calculation using showMem function
        -- and print it
        memTimer = timer.performWithDelay( 1000,
                                           showMem, 1 )
    end
end
```

scene1.lua - hide

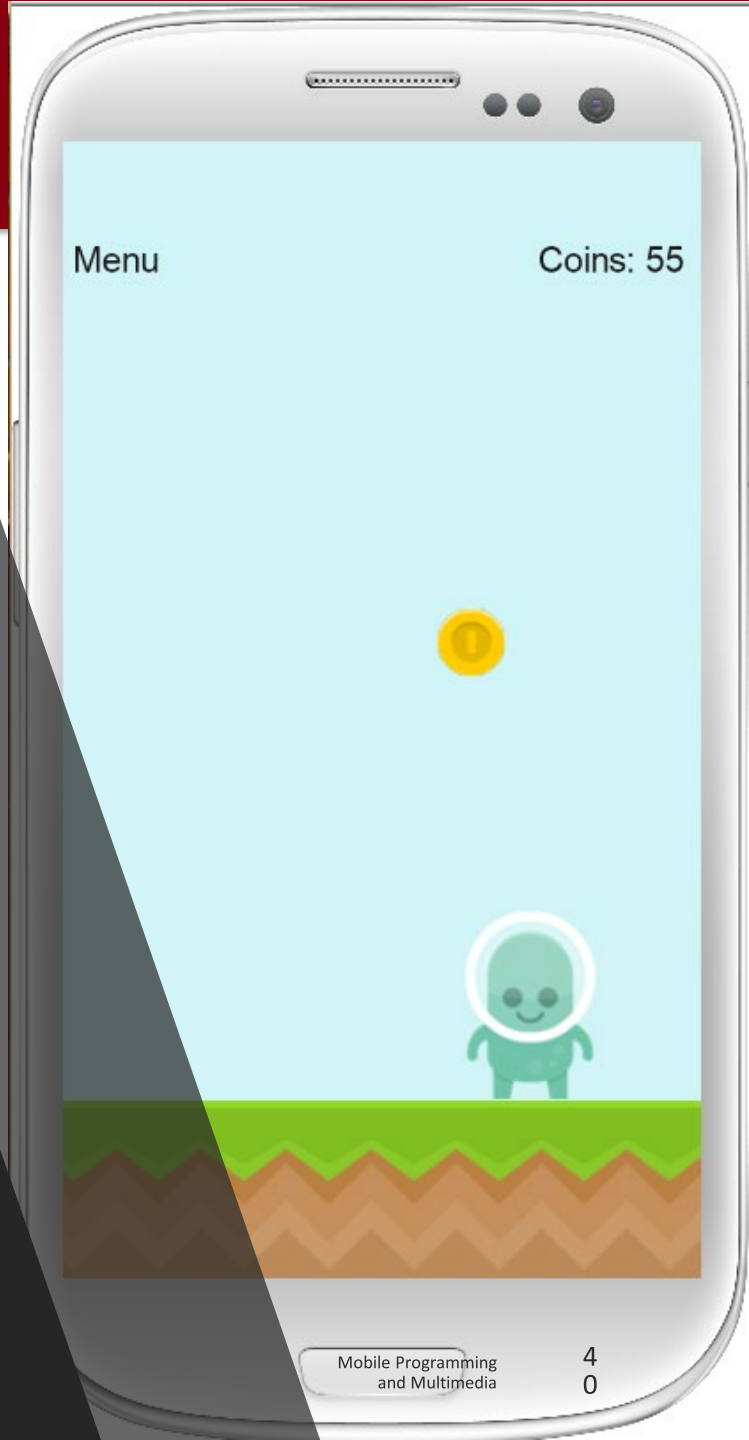

```
function scene:hide( event )
    local phase = event.phase
    if (phase == "will") then
        print( "1: hide event, phase will" )
        -- remove the listener
        image:removeEventListener( "touch", image )
        -- deliting the timer
        timer.cancel( memTimer ); memTimer = nil;
        -- reset label text
        text2.text = "MemUsage: "
    end
end
```

scene1.lua – destroy & event


```
function scene:destroy(event)
    print( "((destroying scene 1's view))" )
End

local function onSceneTouch(self, event)
    if (event.phase == "began") then
        composer.gotoScene( "scene2", "slideLeft", 800  )
        return true
    end
end
```

More complete example



main.lua


```
-- hide the status bar
display.setStatusBar(display.HiddenStatusBar)

local composer = require( "composer" )

composer.gotoScene("menu")
```

menu.lua - 1


```
local composer = require("composer")
local scene = composer.newScene()
local widget = require("widget")

scene:addEventListener( "create", scene )
scene:addEventListener( "show", scene )
scene:addEventListener( "hide", scene )
scene:addEventListener( "destroy", scene)
return scene
```

menu.lua – scene:create - 1


```
function scene:create( event )
    local sceneGroup = self.view
    local background = display.newRect(sceneGroup, 0, 0,
                                       display.actualContentWidth,
                                       display.actualContentHeight)
    background.x = display.contentWidth * 0.5
    background.y = display.contentHeight * 0.5
    background:setFillColor(208/255,244/255,247/255)
    local ground = display.newImageRect(sceneGroup, "images/ground.png",
                                                                    480, 90)
    ground.x = display.contentWidth * 0.5
    ground.y = display.contentHeight
    local bob = display.newImageRect(sceneGroup, "images/bob-title.png",
                                                                    128, 153)
    bob.x = 90
    bob.y = ground.y - 120
```

menu.lua – scene:create - 2


```
    local gameTitle = display.newImageRect(sceneGroup, "images/title.png",300,160)
    gameTitle.x = display.contentWidth * 0.5
    gameTitle.y = 100
    local function onStartTouch( event )
        if (event.phase=="ended") then
          composer.gotoScene("game", "slideLeft")
        end
    end
    local btn_start = widget.newButton {
        defaultFile = "images/switchGreen.png",
        overFile = "images/switchGreen_pressed.png",
        onEvent = onStartTouch
    }
    btn_start.x = 235
    btn_start.y = ground.y - 115
    sceneGroup:insert(btn_start)
end
```

game.lua - librerie


```
local composer = require( "composer" )
local scene = composer.newScene()
local physics = require( "physics" )
physics.start()

scene:addEventListener( "create", scene )
scene:addEventListener( "show", scene )
scene:addEventListener( "hide", scene )
scene:addEventListener( "destroy", scene)

return scene
```

game.lua – scene:create - 1


```
function scene:create( event )
    local sceneGroup = self.view
    local coins = {}
    local coinsSent = 1
    local coinsCollected = 0
    local background, ground, bob, txt_menu,
                                        txt_coins
    local function onMenuTouch( event )
        if ( event.phase == "ended") then
        composer.gotoScene("menu", "slideRight")
        end
    end
```

game.lua – movimento


```
local function moveBob(event)
    if(event.phase == "ended") then
        transition.to(bob, {x=event.x, time=200})
        bob.x = event.x
    end
end
```

game.lua – coin


```
local function sendCoins()
    coins[coinsSent] = display.newImageRect(
            sceneGroup, "images/coinGold.png", 40, 40)
    coins[coinsSent].x = math.random(0,
                                     display.contentWidth)
    coins[coinsSent].name = "coin"
    physics.addBody(coins[coinsSent])
    cointsSent = cointsSent +1
end
```

game.lua - collisions


```
local function onCollision(event)
    -- if Bob collides with a coin
    if(event.object1.name == "bob" and
            event.object2.name == "coin") then
        display.remove(event.object2)
        coinsCollected = coinsCollected + 1
        txt_coins.text = "Coins: "..coinsCollected
    end
end
```

game.lua – background and ground

```
background = display.newRect(sceneGroup, 0, 0,
                            display.actualContentWidth,
                            display.actualContentHeight)
background.x = display.contentWidth * 0.5
background.y = display.contentHeight * 0.5
background:setFillColor(208/255,244/255,247/255)

ground = display.newImageRect(sceneGroup,
                 "images/ground.png", 480, 90)
ground.x = display.contentWidth * 0.5
ground.y = display.contentHeight
ground.name = "ground"
physics.addBody(ground, "static")
```

game.lua - text


```
txt_menu = display.newText(sceneGroup, "Menu", 0, 0,
                                    native.systemFont, 18)
txt_menu.anchorX = 0
txt_menu.x = 5
txt_menu.y = 15
txt_menu:setFillColor(0.1,0.1,0.1)
txt_menu:addEventListener("touch", onMenuTouch)

-- other texts
```

game.lua - Bob


```
bob = display.newImageRect(sceneGroup,
                           "images/bob-play.png", 80, 96)
bob.x = 90
bob.y = ground.y - 120
bob.name = "bob"
physics.addBody(bob)

Runtime:addEventListener("touch", moveBob)
timer.performWithDelay(1250, sendCoins, 0)
Runtime:addEventListener( "collision", onCollision )
```

References

- Official site
  - https://solar2d.com/
- Documentation
  - https://docs.coronalabs.com/
  - http://www.lua.org/manual/
- Examples
  - https://github.com/coronalabs/samples-coronasdk
- Solar 2D
  - https://github.com/coronalabs/corona/releases