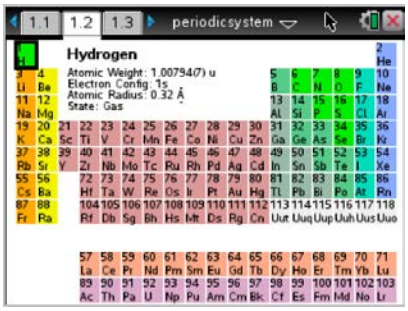
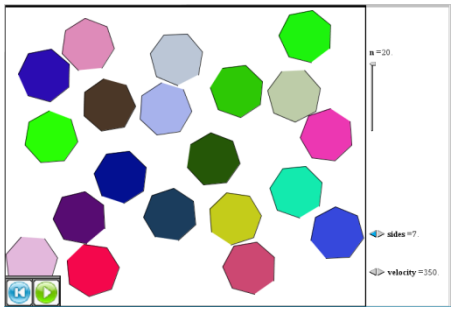
Cahiers T³ Europe
Vlaanderen nr. 35

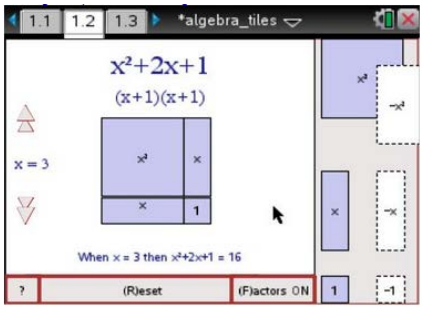
Aan de slag met LUA

LUA-scripts in TI-Nspire

An-Sofie Bruggeman
Gregory Deroo
Jan-Klaas D'hulster
Joline Strubbe
Virginie Vileyn







KATHOLIEKE UNIVERSITEIT LEUVEN

Inhoud

Hoofdstuk 1: Opmaak in Lua

Vooraleer we allerlei uitgebreide en wetenschappelijk nuttige applicaties kunnen maken, moeten we de basis onder de knie hebben. In dit eerste hoofdstuk komt de grafische kant van het programmeren aan bod.

Om te beginnen houden we ons bezig met de tekstopmaak, later bespreken we tabellen, afbeeldingen … Maar eerst verklaren we enkele fundamentele begrippen.

1 Nieuwe begrippen bij dit hoofdstuk

Een functie

Net zoals in andere programmeertalen, maken we in Lua gebruik van functies: aan de hand daarvan maken we duidelijk wat er moet gebeuren. Een functie bestaat uit een kop, waarbij tussen haakjes een argument staat. In ons voorbeeld is dit gc, wat staat voor Graphical Context. In de body staat de inhoud. Met end wordt elke functie beëindigd.

In dit hoofdstuk gaan we bijvoorbeeld vooral iets grafisch 'plakken' op ons scherm in TI-Nspire:

```
function on.paint(gc)
    [body]
end
```

Compileren

De broncode getypt in het scripteditor van het programma worden door de compiler vertaald naar machinetaal.

Commentaar

In de scripteditor is het ook mogelijk om bij de code commentaar te typen, die door de compiler niet zal worden vertaald. Dit is handig om later te kijken hoe een bepaald script in elkaar zit, of om het begrijpelijk te maken voor buitenstaanders. Het schept structuur in het script.

Voor commentaar worden 2 streepjes geplaatst:

```
-- Plaats hier je commentaar
```

RGB-codes

Bij het instellen van een kleur van bv. een tekst werken we met RGB-codes (Red Green Blue): we raden Paint aan om van een bepaalde kleur de code te bemachtigen: klik op *Kleuren bewerken*, kies de gewenste kleur en neem de getallen over die rechts onderaan het venster staan bij '*Rood:'*, '*Groen:'* en '*Blauw:'*.

2 Het plaatsen van een tekst

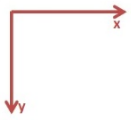
Nu starten we het programmeren met Lua in TI-Nspire. Open de TI-Nspire software, ga naar *Insert* in het menu en klik op *Script Editor* → *Insert Script* om de scripteditor te openen.

Om te starten leren we een tekst op het scherm te plaatsen. Daarnaast leren we deze opmaken.

De code die we in de scripteditor moeten plaatsen, ziet er als volgt uit:

```
function on.paint(gc)
  gc:drawString("aan de slag met LUA",0,20)
end
```

De body is grafisch, daarom begint het met "gc". Erna wordt meegegeven dat een tekst (String) op het scherm geplakt wordt. Tussen haakjes staan 2 argumenten: tussen aanhalingstekens de tekst, erachter de x- en y-coördinaat van de plaats waar de tekst moet komen. Let op: de oorsprong van het assenkruis in TI-Nspire ligt in de linkerbovenhoek!



Als het script in de editor staat, klik je op 'Set Script' om te compileren en vervolgens op 'Focus Script' om te kijken naar het eindresultaat in TI-Nspire.

De tekst staat in de linkerbovenhoek. Nu willen we de zin in het midden van het scherm plaatsen (of in het midden van élk scherm waarop het kan weergegeven worden). Ernaast veranderen we de tekstkleur,lettertype …

Actie	Code

KOP FUNCTIE
  h: hoogte scherm
  w: breedte scherm
  lettertype
  tekstkleur
  sw: breedte tekst
  sh: hoogte tekst
  plaatsen van de tekst
EINDE FUNCTIE
	function on.paint(gc)
	  local h = platform.window:height()
	  local w = platform.window:width()
	  gc:setFont("sansserif","b",12)
	  gc:setColorRGB(41,184,120)
	  local sw = gc:getStringWidth("aan de slag met LUA")
	  local sh = gc:getStringHeight("aan de slag met LUA")
	  gc:drawString("aan de slag met LUA",w/2 - sw/2,h/2 + sh/2)
	end

Om de zin in het midden van het scherm te plaatsen, definiëren we eerst de hoogte en de breedte van ons scherm (`local`) en van de tekst, om deze zo te kunnen verwerken in onze nieuwe waarden voor x en y.

Test de werking van het script uit door de tekst, de positie, de tekstkleur, … eens te veranderen.

Ten slotte is het handig om meerdere zinnen te plaatsen op het computerscherm. Dat kan allemaal in één functie; het is niet nodig om een nieuwe functie te definiëren:

Actie	Code

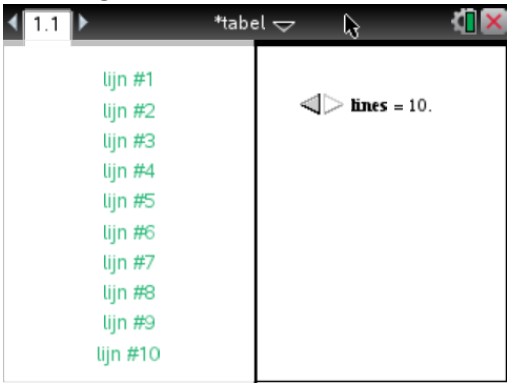
KOP FUNCTIE
  h: hoogte scherm
  w: breedte scherm
------------------------ 	
 	
  lettertype tekst 1
  tekstkleur tekst 1
  sw: breedte tekst 1
  sh: hoogte tekst 1
  plaatsen van tekst 1
------------------------ 	
 	
  lettertype tekst 2
  tekstkleur tekst 2
  stringw: breedte tekst 2
  stringh: hoogte tekst 2
  plaatsen van tekst 2
EINDE FUNCTIE
	function on.paint(gc)
	  local h = platform.window:height()
	  local w = platform.window:width()
	-- 	
	-- *TEKST 1*
	  gc:setFont("sansserif","b",12)
	  gc:setColorRGB(41,184,120)
	  local sw = gc:getStringWidth("aan de slag met LUA")
	  local sh = gc:getStringHeight("aan de slag met LUA")
	  gc:drawString("aan de slag met LUA",w/2 - sw/2,h/2 + sh/2)
	-- 	
	-- *TEKST 2*
	  gc:setFont("sansserif","i",10)
	  gc:setColorRGB(54,150,207)
	  local stringw = gc:getStringWidth("Nog een zin!")
	  local stringh = gc:getStringHeight("Nog een zin!")
	  gc:drawString("Nog een zin!",w-stringw, h)
	end

Stel dat de twee lijnen hetzelfde lettertype enzovoort moeten hebben, dan moet je dit geen tweede keer meer definiëren.

Opmerking: let op dat je eerst het lettertype en grootte instelt, en erna pas de plaatsbepaling op het scherm! De computer berekent immers de plaats op basis van het lettertype en de grootte die VOORAF zijn gedefinieerd. Wanneer we die volgorde niet aanhouden, kunnen we problemen krijgen met de weergave (bv. de tekst wordt niet volledig weergegeven).

3 Tabellen

In de voorgaande paragraaf leerden we tekst te coderen. Dit is handig voor kleine hoeveelheden tekst, maar om meer tekst te kunnen plaatsen zullen we tabellen gebruiken. Eerst leren we tabellen maken, vervolgens wijzigen we de inhoud ervan.



We werken met een `on.paint(gc)` functie. Hierin komen de afmetingen van het venster, wordt aangegeven dat een tabel aangemaakt wordt, linecount, letterkleur, lettertype,…

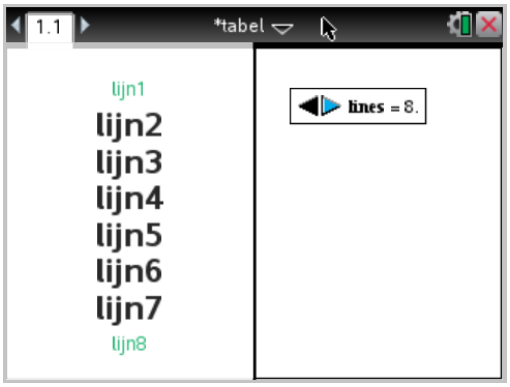
We kunnen een aantal lijnen standaard in de broncode meegeven, of er een schuifknop bij betrekken.

Om een schuifknop aan te maken kiezen we *Pagina-indeling: twee deelvensters*. Links komt de tabel, rechts een meetkundevenster. Klik vervolgens één keer in het rechtervenster, zodat dit venster geactiveerd is. Ga dan naar *Documentenwerkset, Acties: schuifknop invoegen*. Nu verschijnt de schuifknop, maar is deze nog niet gelinkt aan de tabel. Daarom gaan we naar instellingen door met de rechtermuisknop op de schuifbalk te klikken. Geef in de instellingen aan dat de schuifknop gelinkt is aan "Lines"; verder kan je een minimum, maximum en stapgrootte instellen. Tot slot kun je het venster lay-outen door de schaalverdeling te verbergen (rechtermuisknop: *Hide Scale*) en de schuifknop te minimaliseren.

Actie	**Code**

KOP FUNCTIE
  h: hoogte scherm
  w: breedte scherm
  Lokale variabele: tabel
  Linecount=aantal lijnen of 1
  Lettertype
  Letterkleur
 	
 Voor k=1, wordt het aantal lijnen =k
  Tabel krijgt grootte k
 	
  Positionering in het deelvenster
 	
EINDE FUNCTIE
	function on.paint(gc)
	  local h = platform.window:height()
	  local w = platform.window:width()
	  local table = {}
	  local linecount = (var.recall("lines") or 1)
	  gc:setFont("sansserif", "r", 10)
	  gc:setColorRGB(41, 184, 120)
	
	 for k = 1, linecount do
	  table[k] = "lijn #"..k
	  strwidth = gc:getStringWidth(table[k])
	  strheight = gc:getStringHeight(table[k])
	  gc:drawString(table[k], w/2 - strwidth/2,
	  h*k/(linecount + 1) + strheight/2)
	 end
	end

Nu het aanmaken van een tabel uitgelegd is, kan de opmaak worden aangepast. Als eerste kan het lettertype, kleur en grootte variëren naargelang de plaats.



Om dit voorbeeld hiernaast te verkrijgen, wordt gewerkt met de `if...else` uitdrukkingen. Hiermee wordt aangegeven: als (`if`) de lijn gelijk is aan de eerste of laatste lijn, de lettergrootte 10 moet zijn, en de kleur blauw (RGB-code). Anders (`else`) is de lettergrootte 16, vet en zwart. Tot slot wordt de `if...else` uitdrukking afgesloten met end.

Actie	**Code**

KOP FUNCTIE
  h: hoogte scherm
  w: breedte scherm
  Lokale variabele: tabel
  Variabele "lines" opvragen
 	
 Voor k=1, wordt het aantal lijnen =k
  Als k=1 of k=laatste getal dan
  Lettertype geval 1
  Letterkleur geval 1
	function on.paint(gc)
	  local h=platform.window:height()
	  local w=platform.window:width()
	  local table = {}
	  local linecount = (var.recall("lines") or 1)
	
	 for k = 1, linecount do table[k] = "lijn"..k
	  if k==1 or k == linecount then
	  gc:setFont("sansserif", "r", 10)
	  gc:setColorRGB(41, 184, 120)

```
  Anders
    Lettertype andere gevallen
    Letterkleur andere gevallen
  Einde if
 Positionering in het deelvenster




EINDE FUNCTIE
```

```
  else
    gc:setFont("sansserif", "b", 16)
    gc:setColorRGB (40,40,40)
  end
  strwidth = gc:getStringWidth(table[k])
  strheight = gc:getStringHeight(table[k])
  gc:drawString(table[k], w/2 - strwidth/2,
  h*k/(linecount + 1) + strheight/2)
 end
end
```

Opmerking: soms zien we in de broncode "==" staan. Op deze manier tonen we aan dat het om een vergelijking gaat. Eén zo'n teken, "=", wijst erop dat we iets nieuws definiëren.

Eens de opmaak in orde is, kunnen we ons tot de inhoud richten.

Tot nu toe gaven we met dit stukje broncode:
`linecount do table[k] = "lijn"..k`
de opdracht om in de tabel "k" aantal lijnen te creëren met als inhoud het woord "lijn" met het nummer erachter.

Vanaf nu kunnen we eender welke inhoud in deze tabellen zetten:
`table[k] = (var.recall("line"..k) or "lijn"..k)`

Door bovenstaande uitdrukking gaat het systeem zoeken naar de waarde die in de variabele gestopt is. Als hij deze niet vindt, geeft hij "*lijn k*" weer. Om de inhoud van een lijn aan te passen, voeren we een nieuwe notitiepagina in. Plaats daarin een Mathbox (Crtl + M). Geef aan welke lijn je wil veranderen (bv. line1), gevolgd door ":=" en plaats tussen aanhalingstekens wat je wilt zetten in die lijn. Wanneer je op enter duwt, wordt het direct in de tabel aangepast. Op deze manier kunnen we snel en eenvoudig wijzigingen aanbrengen.

Een tweede manier om de inhoud van de lijnen te veranderen gaat als volgt: Je gaat naar *Invoegen: Nieuwe pagina invoegen (Crtl+I)*. Vervolgens kies je voor *Notities invoegen*. Op dit notitieblad klik je met je rechtermuisknop en ga je naar *Variabelen selecteren*. De enige mogelijkheid die er staat is "*Lines*". Klik hierop en verander "*Lines*" in de lijn die je wilt bewerken (bv. Line1) gevolgd door ":=" en plaats tussen aanhalingstekens wat je wilt zetten in die lijn. Wanneer je op enter duwt, wordt het direct in de tabel aangepast. Op deze manier kunnen we snel en eenvoudig wijzigingen aanbrengen.

4 Afbeeldingen

4.1 Afbeelding importeren

In dit gedeelte leren we hoe je met Lua ervoor zorgt dat er in Ti-Nspire een afbeelding verschijnt.

Om te beginnen zullen we onze afbeelding definiëren en omzetten in digitale tekens, die onze computer begrijpt.
Het definiëren doen we door
`NaamAfbeelding = image.new(“ digitale code”).`
De digitale code verkrijgen we als volgt: in de scripteditor is er in de menubalk een knop *Afbeelding invoegen*. Gebruik deze knop, kies een afbeelding en klik op *Openen*. Je zult zien dat in je broncode een lange reeks getallen verschijnt, dit is de digitale code van deze afbeelding.

Actie	**Code**

Definitie nieuwe afbeelding
(“digitale code”)
 	
KOP FUNCTIE
  Wat er op het scherm moet komen
EINDE FUNCTIE
	ThumbsUp = image.new
	(”9\001\000\000\179\001\000\000\000\000\000\000....”)
	
	function on.paint(gc)
	  gc: drawImage(ThumbsUp,0,0)
	end

De afbeelding moet duidelijk nog worden aangepast. Dat verloopt grotendeels op de manier die we hiervoor al gebruikt hebben. We definiëren de grootte van ons venster en de grootte van onze afbeelding. Vervolgens doen we iets nieuws: we definiëren een schaal, geven de opdracht om een kopie van de afbeelding te maken en geven mee dat de grootte van onze kopie wordt aangepast volgens die schaal. Tot slot geven we de opdracht om die kopie op het scherm te zetten.

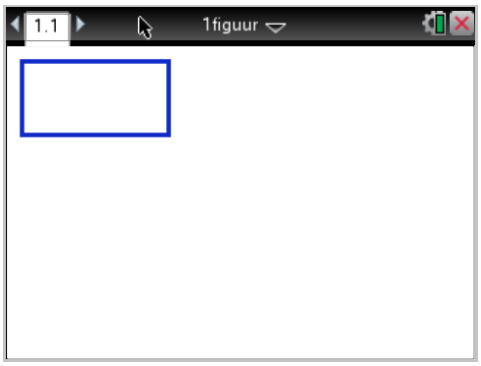
Actie	**Code**

Definitie nieuwe afbeelding
(“digitale code”)
 	
KOP FUNCTIE
  h: hoogte scherm
  w: breedte scherm
  schaal definiëren
  imw: breedte afbeelding
  imh: hoogte afbeelding
  kopie van de afbeelding
  imw:breedte van de kopie
  imw: hoogte van de kopie
  kopie op het scherm zetten
EINDE FUNCTIE
	ThumbsUp = image.new
	(”9\001\000\000\179\001\000\000\000\000\000\000....”)
	
	function on.paint(gc)
	  local w = platform.window:width()
	  local h = platform.window:height()
	  local sc = 0.2
	  local imw = ThumbsUp:width()
	  local imh = ThumbsUp:height()
	  local im = ThumbsUp:copy(sc * imw, sc * imh)
	  local imw = im:width(im)
	  local imh = im:height(im)
	  gc:drawImage(im, (w - imw)/2, (h - imh)/2)
	 end

Opmerking: zorg ervoor dat de ingevoegde afbeelding niet te groot is. De code die in de scripteditor verschijnt, omschrijft namelijk pixel per pixel. Hoe groter de afbeelding, hoe langer de code wordt, en hoe trager de computer zal werken.

4.2 Figuren maken

Als we eenvoudige figuren willen gebruiken, zoals rechthoeken, cirkels, e.d. dan is het niet de bedoeling dat we bestaande figuren importeren. Dit zou nogal omslachtig zijn, daarom leren we nu hoe we zo'n figuren zelf kunnen maken, groottes definiëren en op ons scherm laten verschijnen.



We gebruikten reeds `gc:drawString`, `gc:drawImage`. Bij het tekenen van figuren verloopt dit volledig analoog.(Zie einde paragraaf) Als een figuur nog niet standaard bestaat, kun je deze zelf definiëren met hetgeen reeds bestaat.

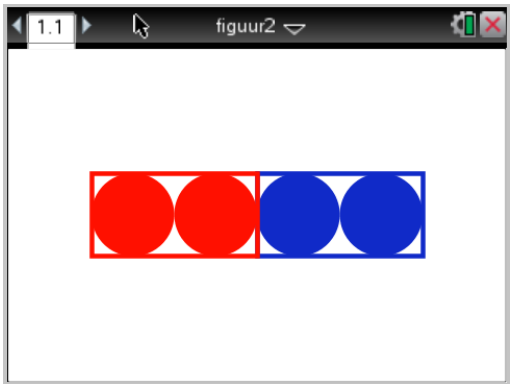
Hier in het voorbeeld definiëren we eerst de functie drawBox waarbij we zeggen dat we tussen de haakjes de de waarde op de x-as, op de y-as, de lengte en de breedte zullen meegeven. Erna gebruiken we de `on.paint` functie, daarin geven we nog wat opmaakgegevens mee: kleur en dikte van de lijn. We geven aan dat we drawBox op het scherm willen, met tussen de haakjes de nodige gegevens.

Actie	Code

KOP FUNCTIE
  Bestaande functie
EINDE FUNCTIE
	function drawBox(x, y, length, height, gc)
	  gc:drawRect(x, y, length, height)
	end

KOP FUNCTIE
  Kleur instellen
  Peneigenschappen instellen
  Drawbox (eigenschappen)
EINDE FUNCTIE
	function on.paint(gc)
	  gc:setColorRGB(16,42,200)
	  gc:setPen("medium", "smooth")
	  drawBox(10, 10, 100, 50, gc)
	end

In een tweede voorbeeld tonen we één van de vele andere mogelijkheden. We werken verder met de blauwe rechthoek we plaatsen er 2 opgevulde blauwe cirkels in en we plaatsen een zelfde rood figuur ertegen.



Als je de broncode bestudeert, zie je dat de functie om het vierkant behouden blijft. Wat verandert is de `on.paint` functie. Hierin definiëren we de grootte van het venster, kiezen we een kleur, een pendikte, … Het enige in deze code dat we niet kennen is de opdracht fillArc, maar de naam spreekt voor zich. Dit is dus één van de vele bestaande codes die te vinden zijn in de appendix, waarnaar we eerder verwezen.

Actie	Code

KOP FUNCTIE
  Bestaande functie
EINDE FUNCTIE
	function drawBox(x, y, length, height, gc)
	  gc:drawRect(x, y, length, height)
	end

----------	----------

KOP FUNCTIE
  w= breedte venster
  h= hoogte venster
  kleur instellen
  peneigenschappen instellen
  drawBox (coördinaten)
  gc: vul een cirkel (coördinaten)
  gc: vul nog een cirkel (coördinaten)
 	
  kleur instellen
  drawBox (coördinaten)
  gc: vul een cirkel (coördinaten)
 	
  gc: vul een cirkel (coördinaten)
 	
EINDE FUNCTIE
	function on.paint(gc)
	  local w = platform.window:width()
	  local h = platform.window:height()
	  gc:setColorRGB(16,42,200)
	  gc:setPen("medium", "smooth")
	  drawBox(w/2, 3*h/8, h/2, h/4, gc)
	  gc:fillArc(w/2, 3*h/8 , h/4, h/4, 0, 360)
	  gc:fillArc(w/2 + h/4, 3*h/8 , h/4, h/4, 0, 360)
	  gc:setColorRGB(255,16,0)
	  drawBox(w/2 - h/2, 3*h/8, h/2, h/4, gc)
	  gc:fillArc(w/2 - h/4, 3*h/8 , h/4, h/4, 0, 360)
	  gc:fillArc(w/2 - h/2, 3*h/8 , h/4, h/4, 0, 360)
	end

Enkele nuttige meetkundige constructies:

- **drawLine (xStart, ystart, xend, yend)** - trekt een lijn tussen de aangegeven coördinaten.
- **drawRect (x, y, xwidth, yheight)** - tekent een rechthoek op (x, y) met lengte en hoogte als gegeven.
- **fillRect (x, y, xwidth, yheight)** - vult een rechthoek in (x, y) met de laatste gedefinieerde kleur.
- **drawPolyLine ({x1, y1, x2, y2, .., xn, yn, x1, y1})** - tekent een veelhoek met de geordende paren. Opmerking: de veelhoek moet eindigen waar deze begon, zodat deze gesloten is. Dit sluiten gebeurt niet automatisch
- **fillPolygon ({x1, y1, x2, y2, .., xn, yn, x1, y1})** - vult de veelhoek gedefinieerd door de punten met die coördinaten.
- **drawArc (x, y, breedte, hoogte, beginhoek, draaihoek)** - tekent een boog op (x, y) met breedte en hoogte. Een cirkel wordt bepaald van 0 tot 360°.
- **fillArc (x, y, breedte, hoogte, starthoek, draaihoek)** - vult de gedefinieerde boog of cirkel.
- **drawString ("string", x, y, [Position])** – trekt een string op de aangegeven coördinaten.
- **getStringWidth (string)** - retourneert de string breedte.
- **getStringHeight (string)** - retourneert de string hoogte.
- **isColorDisplay ( )** retourneert 1 als het om dat kleur gaat, 0 indien niet.
- **setAlpha (integer)** – zet een getal om in een letter
- **setColorRGB (rood, groen, blauw)** maakt het kleur met de opgegeven RGB-waarden.
- **setFont (lettertype, type, grootte)** - lettertype: ("serif", ..), type: ("b", "r", "i", "bi") voor vet, normaal, cursief en vet cursief. Grootte is een geheel getal.
- **setPen (grootte, stijl)** : grootte ("dun", "medium", "dik"), stijl ("volle lijn", "stippellijn").

5 Menu's

In dit deel leren we menu's invoegen. De uitdrukking voor een menu is:

```
menu{..}
toolpalette.register(menu)
```

Tussen de accolades moeten functies komen waarmee gewerkt kan worden. In dit voorbeeld kan de schaal van de afbeelding via het menu worden veranderd.

Actie	Code
Code afbeelding	`ThumbsUp = image.new("9\001\000\...)`
Variabele sc	`sc = 0.2`

KOP FUNCTIE
Sc wordt gedeclareerd
0.1 bij sc optellen
EINDE FUNCTIE
	`function schaalvergroten ()`
	`sc= (var.recall("scale") or 0.2)`
	`var.store("scale",sc + 0.1)`
	`end`

KOP FUNCTIE
sc wordt gedeclareerd
0.1 van sc aftrekken
EINDE FUNCTIE
	`function schaalverkleinen()`
	`sc = (var.recall("scale") or 0.2)`
	`var.store(("scale"), sc - 0.1)`
	`end`

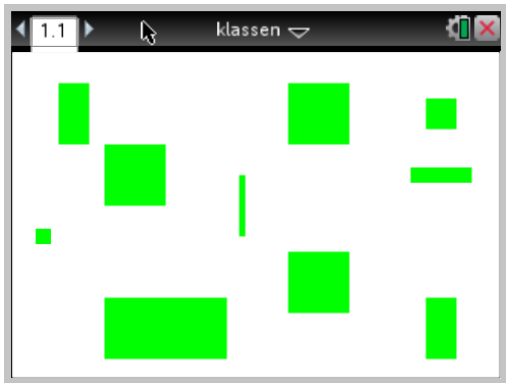
KOP FUNCTIE
h: hoogte scherm
w: breedte scherm
imw: breedte afbeelding
imh: hoogte afbeelding
kopie van de afbeelding
imw:breedte van de kopie
imw: hoogte van de kopie
kopie op het scherm zetten
EINDE FUNCTIE
	`function on.paint(gc)`
	`local w = platform.window:width()`
	`local h = platform.window:height()`
	`local imw = ThumbsUp:width()`
	`local imh = ThumbsUp:height()`
	`local im = ThumbsUp:copy(sc * imw, sc * imh)`
	`local imw = im:width(im)`
	`local imh = im:height(im)`
	`gc:drawImage(im, (w - imw)/2, (h - imh)/2)`
	`end`

Menu:
Submenu: "schaal"
Keuze1: vergroten
Verwijzing naar de vergrootfunctie
Keuze 2: verkleinen
Verwijzing naar de verkleinfunctie
 	
Menu wordt aangemaakt.
	`menu={`
	`{"Schaal:",`
	`{"Vergroten",function() schaalvergroten()`
	`end},`
	`{"Verkleinen",function() schaalverkleinen()`
	`end},`
	`},}`
	`toolpalette.register(menu)`

Hoofdstuk 2: Klassen

In dit hoofdstuk maken we kennis met een nieuw begrip: klassen. Er bestaan meerdere omschrijvingen en definities voor een klasse, maar misschien kunnen we het omschrijven als een slimme functie. Er zijn in LUA enorm veel bestaande klassen, die tijdens het installeren van LUA meegegeven worden. Deze kun je opvragen en gebruiken, of je kunt als gebruiker ook zelf klassen definiëren en elders in je programma gebruiken. Als we een klasse definiëren, zorgen we ervoor dat alle objecten die tot deze klasse behoren heel wat over zichzelf weten. Voor meer gevorderde functies is het noodzakelijk dat objecten deel uitmaken van een klasse, anders werkt het niet. Ook is het bestaan van klassen handig indien je heel veel gelijksoortige objecten wilt gebruiken. Dan kun je eenmaal de klasse aanmaken, de eigenschappen definiëren en nadien gewoon telkens de klasse oproepen in plaats van telkens bij elk object alle gegevens te definiëren. We zullen het gebruik van klassen illustreren met een voorbeeld.

Het voorbeeld begint met iets nieuws in verband met de kleurencodes. Zoals je ziet werken we niet met de gebruikelijke RGB-codes. We roepen een bestaande klasse Color aan en halen daaruit het object green, wat we verderop in de code zullen gebruiken. Hierna beginnen we zelf een klasse te maken. We geven deze de naam Square en zeggen dat dit een klasse wordt. Vervolgens geeft `init` aan dat we de klasse initialiseren. De waarden die tussen de haakjes worden meegegeven, worden in de waarden van de klasse gestopt. Dit gebeurt door de code `self.x = x`. Voor wie vertrouwd is met andere programmeertaal: deze uitdrukking is vergelijkbaar met de “this” in Java. In de volgende functie geven we mee wat Square echt bevat, de delen die geïnitialiseerd zijn en echt onderdeel van de figuur zijn, worden nu in lokale variabelen gestopt. Verder worden er grenzen voor de waarden gedefinieerd zodat det figuur nooit groter dan het scherm kan worden. Tot slot geven we in een laatste functie mee hoe het uitzicht zal zijn: kleur, opgevuld of niet … Bij de kleur zeggen we dat we een andere klasse uitpakken (uit het pakket dat we meekregen van LUA). Hiermee is de klasse klaar. Maar indien je nu op *Script instellen* klikt verschijnt er logischerwijs niets op je scherm, want je definieert alleen de klasse, verder vraag je niets aan de klasse.



Nu maken we objecten aan in de klasse. We kiezen een naam voor het object, zeggen dat dit van de klasse Square is en geven de variabelen mee: de x-en y-positie, de lengte en de breedte. Zo zijn de objecten gedefinieerd. Tot slot moeten ze op het scherm verschijnen, dit gebeurt met de `on.paint` functie.

Actie	Code

Groen uit de klasse “Color”	Color = {
	green = {0x00, 0xFF, 0x00},
	}

Square = klasse KOP FUNCTIE De x is de x in de klasse De y is de y in de klasse De breedte is de breedte in de klasse De hoogte is de breedte in de klasse De kleur(groen) is de kleur in de klasse EINDE FUNCTIE	Square = class() function Square:init(x, y, width, height) self.x = x self.y = y self.width = width or 20 self.height = height or 20 self.color = Color.green end
KOP FUNCTIE Lokale variabele: sw= de breedte Lokale variabele: sh= de hoogte Grenzen voor deze variabelen EINDE FUNCTIE	function Square:contains(x, y) local sw = self.width local sh = self.height return x >= self.x - sw/2 and x <= self.x + sw/2 and y >= self.y - sh/2 and y <= self.y + sh/2 end
KOP FUNCTIE Het kleur wordt ontpakt Het figuur wordt gevuld EINDE	function Square:paint(gc) gc:setColorRGB(unpack(self.color)) gc:fillRect(self.x - self.width / 2, self.y - self.height / 2, self.width, self.height) end
We maken objecten van de klasse	Sq1 = Square(80, 80, 40, 40) Sq2 = Square(40, 40, 20, 40) Sq3 = Square(280, 80, 40, 10) Sq4 = Square(100, 180, 80, 40) Sq5 = Square(150, 100, 4, 40) Sq6 = Square(280, 180, 20, 40) Sq7 = Square(200, 40, 40, 40) Sq8 = Square(200, 150, 40, 40) Sq9 = Square(20, 120, 10, 10) Sq10 = Square(280, 40, 20, 20)
KOP FUNCTIE De objecten worden op het scherm geplaatst EINDE FUNCTIE	function on.paint(gc) Sq1:paint(gc) Sq2:paint(gc) Sq3:paint(gc) Sq4:paint(gc) Sq5:paint(gc) Sq6:paint(gc) Sq7:paint(gc) Sq8:paint(gc) Sq9:paint(gc) Sq10:paint(gc) end

Hoofdstuk 3: Toetsenbord en muis

Vanaf dit hoofdstuk beginnen we in de programma's gebruik te maken van het toetsenbord en de muis. Deze mogelijkheid is in talloze toepassingen bruikbaar (een object verplaatsen, schaal aanpassen …). We kunnen het venster optimaal gebruiken en het scherm staat niet vol met allerhande schuifbalken. Bijgevolg worden programma's eenvoudiger in gebruik.

1 Automatisch het venster vernieuwen

Vooraleer we beginnen met het toetsenbord, merken we nog iets belangrijks op. In de broncodes hierna zullen we zien dat we het venster bij elke actie vernieuwen (zeker bij het gebruik van toetsen).

Eigenlijk staat best **standaard in elk programma** een code om ons venster **om de zoveel tijd automatisch** te vernieuwen, anders kunnen er makkelijk problemen voorkomen in het programma. Dat doen we aan de hand van deze twee functies:

Actie	Code

KOP FUNCTIE
  Timer instellen (5 keer per seconde)
EINDE FUNCTIE
 	
KOP FUNCTIE
  Venster vernieuwen
EINDE FUNCTIE
	function on.construction()
	  timer.start(1/5)
	end
	
	function on.timer()
	  platform.window:invalidate()
	end

In de eerste functie maken we een timer aan. Deze 'tikt' steeds vijf keer per seconde. In de tweede functie vernieuwt het venster zichzelf telkens wanneer de timer 20ms verder zit.

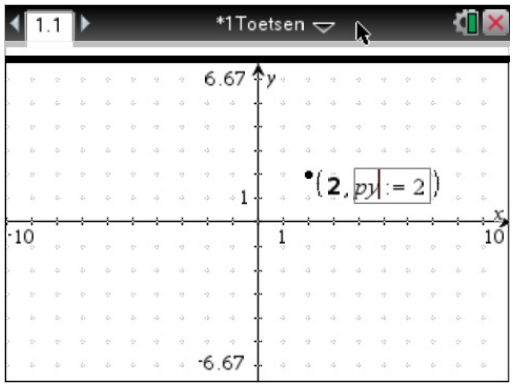
De timer kunnen we eenvoudig aanpassen. Als we de klok tien keer per seconde willen laten tikken, wordt dit simpelweg `timer.start(1/10)`.

2 Gebruik van toetsenbord

2.1 Zonder klassen

Het is de bedoeling om een punt te verplaatsen in een assenstelsel.

Op de nieuwe pagina voegen we *Graphs* in. Klik op *Geometry: Points and Lines: Point* om een punt te tekenen in het lege assenstelsel (plaats deze bijvoorbeeld in de oorsprong). Houd de pijl op het punt, klik op de rechtermuisknop en selecteer *Coordinates and Equations*, daardoor komen de x- en y-coördinaten tevoorschijn. Voor beide coördinaten klikken we opnieuw op de rechtermuisknop en selecteren we *Store* om een variabele in te voegen. Voor de x-coördinaat geven we variabele **px** in, voor de y-coördinaat **py**. Overigens kan het ook



handig zijn om het raster weer te geven. Klik daarvoor op de rechtermuisknop en vervolgens op *Hide/Show: Show Dot Grid*.

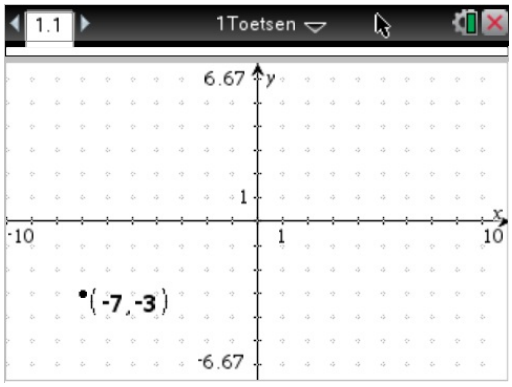
Voor we kunnen beginnen aan het script, moeten we het scherm nog verdelen in twee vensters, anders maakt het programma automatisch een nieuwe, aparte, pagina aan en kan het punt nooit bewegen. In dit voorbeeld houden we het bovenste venster leeg en maken we dit zo klein mogelijk (om het zo weinig mogelijk te laten opvallen).

Ga nu naar de scripteditor. Dit is de code om het punt naar boven te verplaatsen:

Actie	**Code**

KOP FUNCTIE
  y definiëren (ofwel py, ofwel nul)
  y-coördinaat wijzigen: 1 erbij optellen
  Venster vernieuwen
EINDE FUNCTIE
	function on.arrowUp()
	  local y = (var.recall("py") or 0)
	  var.store("py",y+1)
	  platform.window:invalidate()
	end

We stoppen eerst een waarde in y: deze wordt normaal gezien gelijkgesteld aan py. Vindt de computer geen waarde voor py, dan stelt hij y gelijk aan nul. Om het punt naar boven te doen bewegen, tellen we bij de y-coördinaat een eenheid op, waardoor de nieuwe coördinaat gelijk wordt aan y+1. Dit stopt hij nu in py.



De drie andere functies lopen volledig analoog. In principe kan je deze zelf vinden, maar we plaatsen ze hier toch even voor de volledigheid:

Actie	**Code**

KOP FUNCTIE
  y definiëren (ofwel py, ofwel nul)
  y-coördinaat wijzigen: 1 erbij aftrekken
  Venster vernieuwen
EINDE FUNCTIE
	-- Pijl omlaag
	function on.arrowDown()
	  local y = (var.recall("py") or 0)
	  var.store("py",y-1)
	  platform.window:invalidate()
	end

----------	----------

KOP FUNCTIE
  x definiëren (ofwel px, ofwel nul)
  x-coördinaat wijzigen: 1 erbij aftrekken
  Venster vernieuwen
EINDE FUNCTIE
	-- Pijl naar links
	function on.arrowLeft()
	  local x = (var.recall("px") or 0)
	  var.store("px",x-1)
	  platform.window:invalidate()
	end

----------	----------

KOP FUNCTIE
  x definiëren (ofwel px, ofwel nul)
  x-coördinaat wijzigen: 1 erbij optellen
  Venster vernieuwen
EINDE FUNCTIE
	-- Pijl naar rechts
	function on.arrowRight()
	  local x = (var.recall("px") or 0)
	  var.store("px",x+1)
	  platform.window:invalidate()
	end

Natuurlijk is het niet alleen mogelijk om gebruik te maken van de pijltjestoetsen. Stel bijvoorbeeld dat we een knop willen invoeren die ervoor zorgt dat het punt terug onmiddellijk naar de oorsprong gaat: we kunnen dit laten gebeuren door de entertoets. De code ziet er dan als volgt uit:

Actie	Code

KOP FUNCTIE
  px stellen we gelijk aan nul
  py stellen we gelijk aan nul
  Venster vernieuwen
EINDE FUNCTIE
	function on.enterKey()
	  var.store(“px”,0)
	  var.store(“py”,0)
	  platform.window:invalidate()
	end

Daarnaast kunnen we ook nog als functies definiëren:

- `function on.escKey()` (ESC-toets),
- `function on.tabKey()` (tab-toets).

Wat we laten gebeuren wanneer we de toets indrukken, bepalen we volledig zelf. Willen we bijvoorbeeld de ESC-toets gebruiken om naar de oorsprong te gaan, dan moeten we gewoon als kop `function on.escKey()` plaatsen. Met voldoende fantasie kunnen we hier zelfs een compleet andere betekenis aan geven.

Vergeet overigens niet om het scherm automatisch om de zoveel tijd te vernieuwen.

Daarnaast kunnen de toetsen gebruikt worden om de schaal van een afbeelding te veranderen. De schaal moet vergroten bij het indrukken van de pijltjestoets naar boven. De broncode ziet er zo uit:

Actie	Code

KOP FUNCTIE
  Variabele sc (schaal) opvragen
  Schaal vergroten met 0.1
EINDE FUNCTIE
	function on.arrowUp()
	  sc = (var.recall("scale") or 0.5)
	  var.store("scale", sc + 0.1)
	end

Probeer nu zelf de functie op te stellen om de schaal van de afbeelding te laten verkleinen door het pijltje naar beneden te gebruiken. Deze functie verloopt volledig analoog met het bovenstaande.

We zien dus dat het perfect mogelijk is om het gebruik van toetsen in te schakelen zonder daarvoor klassen te moeten gebruiken. Wanneer we bijvoorbeeld een zelf gemaakt object willen verplaatsen, wordt het wel een stuk moeilijker. In dat geval is het beter om wel direct gebruik te maken van klassen, anders wordt het tamelijk ingewikkeld.

Dit zijn niet de enige toepassingen waarvoor een toetsenbord nuttig is. Er bestaan oneindig veel mogelijkheden. Het is aan de lezer om zijn fantasie te gebruiken en nieuwe dingen te proberen.

2.2 Met één klasse

We bouwen verder op de klasse Square, die we eerder hebben aangemaakt in het hoofdstuk Klassen. Het is dus handig om vanuit dat document te vertrekken.

Definieer in het begin van de broncode de breedte W en de hoogte H van het venster.

Actie	Code

KOP FUNCTIE
  Lijst Colors definiëren
 	
  W: breedte venster
  H: hoogte venster
  TrackedObject: niet gedefinieerd
  Object Sq uit klasse Square
EINDE FUNCTIE
	function on.resize()
	  Color = {
	  red = {0xFF, 0x00, 0x00},
	  green = {0x00, 0xFF, 0x00},
	  }
	  W = platform.window:width()
	  H = platform.window:height()
	  TrackedObject = nil
	  Sq = Square(W/2, H/2, W/10, W/10)
	end

In het vervolg staat er een nieuwe variabele TrackedObject gedefinieerd. In het begin wordt deze sowieso leeggemaakt. Vervolgens komt daarin het te verplaatsen voorwerp Sq terecht. Het gebruik van die variabele zorgt ervoor dat het programmeren van de codes in het vervolg eenvoudiger verloopt.

Voor het object verplaatst kan worden, selecteren we het met de tabtoets:

Actie	Code

KOP FUNCTIE
  Als TrackedObject bestaat, dan
  TrackedObject is niet meer geselecteerd
  Einde if
  TrackedObject bevat nu object Sq
  Sq is geselecteerd
  Venster vernieuwen
EINDE FUNCTIE
	function on.tabKey()
	  if TrackedObject ~= nil then
	  TrackedObject.selected = false
	  end
	  TrackedObject = Sq
	  Sq.selected = true
	  platform.window:invalidate()
	end

Dit is de broncode voor de pijltjestoets naar rechts (analoog voor de pijl naar links):

Actie	Code

KOP FUNCTIE
  Als TrackedObject bestaat, dan
  Als het object binnen het venster
  past dan
  5 eenheden naar rechts verschuiven
  Einde if
  Einde if
EINDE FUNCTIE
	function on.arrowRight()
	  if TrackedObject ~= nil then
	  if TrackedObject.x <
	  W - TrackedObject.width/2 then
	  TrackedObject.x = TrackedObject.x + 5
	  end
	  end
	end

De code om het object naar boven te verschuiven (analoog voor de pijl naar beneden):

Actie	Code

KOP FUNCTIE
  Als TrackedObject bestaat, dan
  Als het object binnen het venster past dan
  5 eenheden naar boven verschuiven
  Einde if
  Einde if
EINDE FUNCTIE
	function on.arrowUp()
	  if TrackedObject ~= nil then
	  if TrackedObject.y > TrackedObject.height/2 then
	  TrackedObject.y = TrackedObject.y - 5
	  end
	  end
	end

Daarnaast is het mogelijk om de huidige coördinaat van het vierkant weer te geven in de linkerbovenhoek. Deze code moet daarvoor geplaatst worden in de on.paint functie:

Actie	Code

Als TrackedObject bestaat
  Lettertype
  Letterkleur
De lijn (String) plaatsen
 	
Einde if
	if TrackedObject ~= nil then
	  gc:setFont("sansserif","b",10)
	  gc:setColorRGB(unpack(Sq.color))
	  gc:drawString("("..TrackedObject.x..",".."..TrackedObject.y..")"20,20)
	end

Ten slotte nog een code om het vierkant van een rand te voorzien wanneer deze geselecteerd is:

Actie	Code

Als TrackedObject bestaat
  'pen' instellen
  Lijnkleur
  Rechthoek tekenen
 	
Einde if
	if TrackedObject ~=nil then
	  gc:setPen("medium", "smooth")
	  gc:setColorRGB(0, 0, 0)
	  gc:drawRect(self.x - self.width/2, self.y - self.height/2, self.width, self.height)
	end

2.3 Met meerdere klassen

Bij het gebruik van meerdere klassen moet er bepaald worden welk object beweegt. Daardoor verandert de broncode bij het toetsenbordgebruik gedeeltelijk. Voeg om te beginnen een tweede klasse toe aan het bestand van het vorige onderdeel. De broncode voor een nieuwe klasse Circle:

Actie	Code

Voor meer uitleg, ga terug naar hoofdstuk 2: Klassen.	Circle = class()
	function Circle:init(x, y, width, height)
	  self.x = x
	  self.y = y
	  self.width = width*2
	  self.height = height*2
	  self.radius = height
	  self.color = Color.red
	  self.selected = false

	end
	
	function Circle:contains(x,y)
	  local r = self.radius
	  local d = math.sqrt((self.x - x)^2 + (self.y - y)^2)
	  return d <= r
	end
	
	function Circle:paint(gc)
	  local cx = self.x - self.radius
	  local cy = self.y - self.radius
	  local diameter = 2*self.radius
	  gc:setColorRGB(unpack(self.color))
	  gc:fillArc(cx, cy, diameter, diameter, 0, 360)
	  if self.selected then
	  gc:setPen("medium","smooth")
	  gc:setColorRGB(0,0,0)
	  gc:drawArc(cx, cy, diameter, diameter, 0, 360)
	  end
	end

Definieer nu een object uit elke klasse:

Actie	**Code**

Object uit de klasse Circle
Object uit de klasse Square
	Objects = {
	  Circle (W/2, H/2, W/20, W/20),
	  Square(W/5, H/8, W/10, W/10),
	}

Merk op dat Objects een **lijst** is. Het eerste element is een object uit de klasse Circle, het tweede is een vierkant uit de klasse Square. De volgorde van de elementen speelt een rol in het vervolg.
Breng wijzigingen aan bij de functie on.tabKey():

Actie	**Code**

KOP FUNCTIE
  Als TrackedObject gedefinieerd is dan
  TrackedObject is niet geselecteerd
  Einde if
  Herhaalopdracht: overloop elk object
  Variabele obj definiëren als het i-de object
  Als obj gelijk is aan TrackedObject dan
  TrackedObject wordt het (i+1)e object
 	
  Einde if
  Einde herhaalopdracht
  Als TrackedObject niet gedefinieerd is dan
  TrackedObject wordt het eerste object
  Einde if
  TrackedObject is geselecteerd
  Venster vernieuwen
EINDE FUNCTIE
	function on.tabKey()
	  if TrackedObject ~= nil then
	  TrackedObject.selected = false
	  end
	  for i = 1, #Objects do
	  local obj = Objects[i]
	  if obj == TrackedObject then
	  TrackedObject = Objects[i+1]
	  break
	  end
	  end
	  if TrackedObject == nil then
	  TrackedObject = Objects[1]
	  end
	  TrackedObject.selected = true
	  platform.window:invalidate()
	end

In de broncode staat op een gegeven moment

```
for ··· do
(···)
end
```

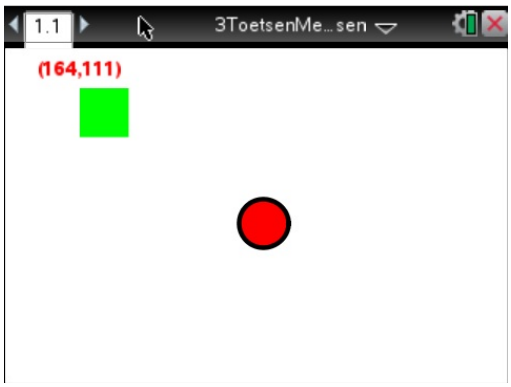
Dit is een **herhaalopdracht**: het programma overloopt elk object uit de lijst Objects. De letter i staat voor de index, die ieder object uit de lijst automatisch meekrijgt. Het i-de object uit de lijst wordt in een lokale variabele obj gestopt. Als de waarde van obj gelijk is aan die van TrackedObject, dan definiëren we die laatste variabele als het daarop volgende object uit de lijst. Als het object Circle dus zowel in obj als in TrackedObject opgeslagen zit, dan veranderen we de inhoud van TrackedObject in Square.

Voer ten slotte enkele wijzigingen uit bij de `on.paint(gc)` functie:

Actie	**Code**

KOP FUNCTIE
  Herhaal voor elk object
  Object tekenen
  Einde herhaalopdracht
  Als TrackedObject bestaat
  Lettertype
  Kleur: zoals het object
  Coördinaten op het venster
  laten verschijnen
  Einde if
EINDE FUNCTIE
	function on.paint(gc)
	  for _, obj in ipairs(Objects) do
	  obj:paint(gc)
	  end
	  if TrackedObject ~= nil then
	  gc:setFont("sansserif", "b", 10)
	  gc:setColorRGB(unpack(TrackedObject.color))
	  gc:drawString("(".. TrackedObject.x..",..
	  TrackedObject.y..")",20,20)
	  end
	end

Aan de andere functies in de code verandert er niets.



3 Gebruik van de muis

3.1 Met één klasse

Voor dit voorbeeld vertrekken we opnieuw vanuit de klasse Square die we aangemaakt hebben in het hoofdstuk Klassen.

Het volgende is essentieel om te snappen waarom we alles op deze manier zullen programmeren. We willen dus een object aanklikken. Daarom houden we het pijltje op het object. We houden de linkermuisknop ingedrukt (*Down*). Vervolgens bewegen we de muis om het object te verplaatsen. Wanneer we dat loslaten, laten we de linkermuisknop los: de knop gaat terug omhoog (*Up*). Bijgevolg hebben we drie functies nodig: `on.mouseDown`, `on.mouseUp` en `on.mouseMove`.

Hetgeen we onder de broncode van de klasse moeten plaatsen, is het volgende:

Actie	Code

Breedte venster
Hoogte venster
TrackedObject is leeg (bestaat niet)
Object: Sq uit de klasse Square
--- 	
MUISKNOP INDRUKKEN
  Als Sq de coördinaten (x,y) bevat
  Als TrackedObject bestaat
  Dan is er niets geselecteerd
  Einde if
  TrackedObject is ons object Sq
  Sq is geselecteerd
  Venster vernieuwen
  Einde if
EINDE FUNCTIE
--- 	
MUISKNOP LOSLATEN
  Als TrackedObject bestaat
  Dan is er niets geselecteerd
  Einde if
  TrackedObject bevat niets meer
  Venster vernieuwen
EINDE FUNCTIE
--- 	
MUIS BEWEGEN
  Als TrackedObject bestaat
  x-coördinaat van TrackedObject wordt x
  y-coördinaat van TrackedObject wordt y
  Venster vernieuwen
 	
EINDE FUNCTIE
	W = platform.window:width()
	H = platform.window:height()
	TrackedObject = nil
	Sq = Square(W/2, H/2, W/15, W/15)
	--- 	
	function on.mouseDown(x,y)
	  if Sq:contains(x, y) then
	  if TrackedObject ~= nil then
	  TrackedObject.selected = false
	  end
	  TrackedObject = Sq
	  Sq.selected = true
	  platform.window:invalidate()
	  end
	end
	--- 	
	function on.mouseUp(x,y)
	  if TrackedObject ~= nil then
	  TrackedObject.selected = false
	  end
	  TrackedObject = nil
	  platform.window:invalidate()
	end
	--- 	
	function on.mouseMove(x,y)
	  if TrackedObject ~= nil then
	  TrackedObject.x = x
	  TrackedObject.y = y
	  platform.window:invalidate()
	  end
	end

3.2 Met meerdere klassen

Gebruik wederom hetzelfde document dat we hiervoor steeds gebruikten. Voer een tweede klasse in, net zoals bij het toetsenbordgebruik met meerdere klassen.

In de broncode maken we opnieuw gebruik van de drie functies `on.mouseDown`, `on.mouseUp` en `on.mouseMove`.

Actie	**Code**

KOP FUNCTIE
  Herhaal voor elk object
  obj definiëren als het i-de object
  Als coördinaten (x,y) in obj zitten
  Als TrackedObject gedefinieerd is
  TrackedObject is niet geselecteerd
  Einde if
  TrackedObject gelijkstellen aan obj
  obj is geselecteerd
  TrackOffsetx definiëren
  TrackOffsety definiëren
  Venster vernieuwen
 	
  Einde if
  Einde herhaalopdracht
EINDE FUNCTIE
	function on.mouseDown(x,y)
	  for i = 1, #Objects do
	  local obj = Objects[i]
	  if obj:contains(x,y) then
	  if TrackedObject ~= nil then
	  TrackedObject.selected = false
	  end
	  TrackedObject = obj
	  obj.selected = true
	  TrackOffsetx = TrackedObject.x - x
	  TrackOffsety = TrackedObject.y - y
	  platform.window:invalidate()
	  break
	  end
	  end
	end

----------	----------

KOP FUNCTIE
  Als TrackedObject gedefinieerd is
  TrackedObject is niet geselecteerd
  Einde als
  TrackedObject is leeg
EINDE FUNCTIE
	function on.mouseUp(x,y)
	  if TrackedObject ~= nil then
	  TrackedObject.selected = false
	  end
	  TrackedObject = nil
	end

----------	----------

KOP FUNCTIE
  Als TrackedObject gedefinieerd is
  Nieuwe x-coördinaat
  Nieuwe y-coördinaat
  Venster vernieuwen
  Einde if
EINDE FUNCTIE
	function on.mouseMove(x,y)
	  if TrackedObject ~= nil then
	  TrackedObject.x = x + TrackOffsetx
	  TrackedObject.y = y + TrackOffsety
	  platform.window:invalidate()
	  end
	end

Hoofdstuk 4: Crossplatforms

1 Inleiding

In dit hoofdstuk bespreken we het gebruik van Lua-programma's die kunnen gebruikt worden bij verscheidene platformen. Een platform is een combinatie van een bepaalde processor en een bepaalde besturingssysteem. En zo wordt er op die basis, platform, specifieke software ontwikkeld. Een voorbeeld van een platform is een pc met Windows als besturingssysteem en er is ook een intel-processor aanwezig.

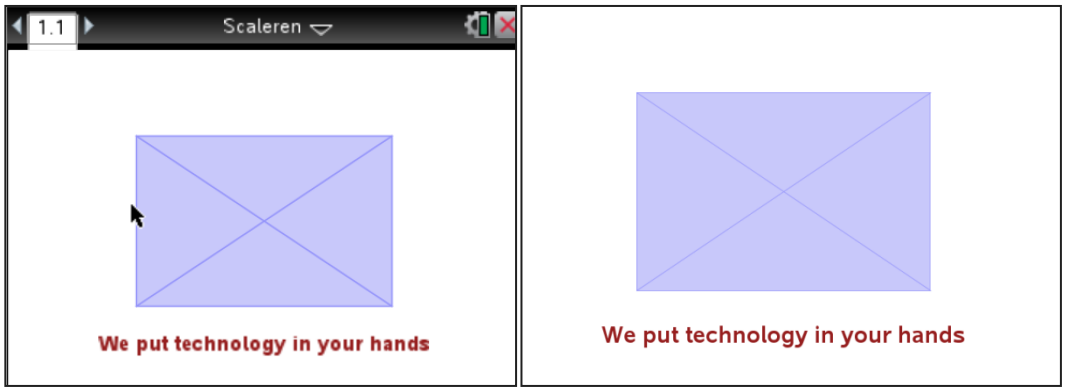
We merken vervolgens op dat het aanwezige platform bepaalt welke programma's er kunnen uitgevoerd worden en welke niet. Vervolgens merken we op dat het vermelden van het nodige API level in het begin van de code van groot belang is. Indien er geen of een fout API level vermeld staat, is het mogelijk dat bij het overzetten van het programma naar de TI-Nspire, die een ander platform bezit dan uw pc, het rekentoestel hierdoor crasht.

Bij het gebruik van een laag API level (vb.: apilevel 1.0) zullen er meer platformen (of toestellen) compatibel zijn met de geschreven code. Daar tegenover staat dan wel dat het gebruik van functies beperkter is.

2 Display

Als eerste bespreken we dat bij overgang van pc TI-Nspire naar de TI-Nspire handheld een ander beeld verkregen wordt. Zo zal de lay-out van het programma zichzelf moeten kunnen aanpassen aan het beeld van de TI-Nspire.

Om dit beter uit te leggen zullen we een voorbeeldprogramma, dat deze schaal toepast, verder uitleggen. Hierin tonen we hoe we de broncode aanpassen om zo een resultaatbeeld te bekomen dat zich aanpast naargelang het gebruikte scherm. Het principe hiervan rust op het gebruik van een **schaalfactor**. Deze houdt rekening met de afmetingen van het aanwezige scherm waarop het resultaat weergegeven wordt.



Hierboven zien we de 2 mogelijke resultaten van een programma waarvan de uitleg over de broncode later aan bod komt. Links staat het beeld dat verkregen wordt bij de Handheld en rechts staat het beeld dat wordt bekomen bij gebruik van de pc.

De broncode van het hiervoor gebruikte voorbeeld ziet er als volgt uit:

Actie	Code
We vermelden als eerste het apilevel.	platform.apilevel = '2.0'
We declareren en initialiseren de lokale variabelen. Deze zijn de schaalfactor, de hoogte en breedte van het scherm waarop het resultaat te zien is.	local iW = 318 local iH = 212 local sf = 1
----------	----------

BEGIN functie on.paint
We maken een init routine aan, hierin geven we de handlers machtigheid.
Vervolgens roepen we de initiële onResize en de initiële onPaint op.
Hier eindigd de init routine.
EINDE functie on.paint
	function on.paint(gc)
	  on.paint = onPaint
	  on.resize = onResize
	
	  onResize(platform.window:width(),
	  platform.window:height())
	
	  onPaint(gc)
	end

----------	----------

BEGIN functie onPaint
We declareren de lokale variabelen: de x- en y- positie, de breedte en hoogte.
Nu zullen we door gebruik te maken van deze variabelen de enveloppe optekenen.
We tekenen als eerste de achtergrond op.
 	
Vervolgens de lijnen van de enveloppe.
 	
Bovendien tekenen we het deeltje tekst op de enveloppe.
 	
EINDE functie onPaint
	function onPaint(gc)
	
	  local x = s(iW/4)
	  local y = s(iH/4)
	  local w = s(iW/2)
	  local h = s(iH/2)
	
	  gc:setColorRGB(200, 200, 250)
	  gc:fillRect(x, y, w, h)
	
	  gc:setColorRGB(150, 150, 250)
	  gc:drawRect(x,y, w, h)
	  gc:drawLine(x,y, x+w, y+h)
	  gc:drawLine(x,y+h, x+w, y)
	
	  gc:setColorRGB(150, 30, 30)
	  gc:setFont("sansserif", "b", fontSize)
	  local str = "We put technology in your hands"
	  local strw = gc:getStringWidth(str)
	  gc:drawString(str, s(iW/2)- strw/2,
	  s(iH*0.9), "bottom")
	end

----------	----------

BEGIN functie onResize
We stellen de schaalfactor gelijk aan de verhouding van het aanwezige scherm op het initiële scherm.
De fontsize moet tussen waarde 6 en 225 liggen.
EINDE functie onResize
BEGIN functie s
Hiermee kunnen we punten en leng-
	function onResize(w, h)
	  sf = w/iW
	  fontSize = 10*sf
	  fontSize = fontSize >= 6 and fontSize or 6
	  fontSize = fontSize <= 255 and fontSize or 225
	
	end
	function s(a)

tes scaleren met de schaalfactor. EINDE functie s	return math.floor(a*sf) end

Als eerste declareren we de globale variabelen iW, iH en sf. We merken op dat men een globale variabele doorheen de hele broncode van een programma kan gebruiken. Een lokale variabele kan men echter alleen in de routine gebruiken waar ze gedeclareerd staat.

We kennen de globale variabelen de waarden van iW en iH toe. Deze slaan op de het aantal pixels, kortom de afmetingen van het scherm, van de TI-Nspire.

Na het declareren van de globale variabelen schrijven we de functie on.paint(gc). Deze functie is de inititialisatie routine, dit betekent dat deze 1 maal wordt doorlopen. In de init routine schrijven we verwijzingen naar de te doorlopen routines: onPaint en onResize.

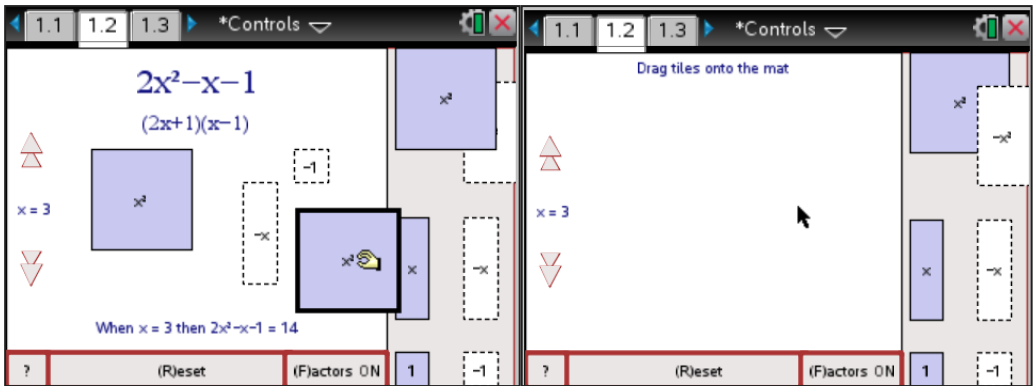
Nu schrijven we de routine onPaint, hier declareren we eerst de lokale variabelen x, y, w en h. Deze verwijzen naar de x- en y-positie, de breedte en de hoogte van het object dat wordt getekend. We maken bij het toekennen van waarden gebruik van de globale variabelen en van de functie s(a). Verder in deze routine zal de enveloppe opgetekend worden door gebruik te maken van de lokale variabelen.

We merken op dat de argumenten van de routine onResize, platform.window:width() en platform.window:height(), verwijzen naar de breedte en hoogte van het scherm (kortom w en h) van het apparaat waarop het programma runt. Nu stellen we de globale variabele sf gelijk aan de verhouding van de breedte van het aanwezige scherm tot het scherm van de TI-Nspire (sf=w/iW). We kennen de fontsize een waarde en grenswaarden toe, deze zal dan in de routine onPaint worden toegepast.

Tenslotte zullen we de functie s met argument a (s(a)) toelichten. Indien men deze functie toepast zal het meegegeven punt zich scaleren in waarde aan het aanwezige scherm. We gebruiken deze functie in de routines onPaint en onResize, zo bekomen we hetzelfde resultaat op de TI-Nspire als op de pc.

3 Controls

In dit deel zullen we het gebruik van verscheidene selectie opties op platformen nader bekijken. Men heeft in het algemeen verscheidene selectie opties, de belangrijkste zijn de muis en het klavier. Indien de gebruiker op de pc een programma gebruikt zal men eerder kiezen voor een muis om object te selecteren. Bij de TI-Nspire CAS merken we op dat het gebruik van de arrows meer vlot dan het gebruik van de touchpad.



We zullen de functie on.activate() gebruiken om het gebruik van de muis (cursor) operationeel te maken. We lichten het gebruik van deze functie toe aan de hand van een voorbeeldprogramma. Hierboven is het resultaat te zien van de code, vervolgens bekijken we een deel van de code zelf.

Actie	Code
nodige apilevel wordt vermeld	platform.apilevel = '1.0' …

BEGIN functie on.activate
  cursor verschijnt op het beeld
EINDE functie on.activate
	function on.activate()
	  cursor.show()
	end

BEGIN functie on.resize
  cursor verschijnt op het beeld
	function on.resize()
	  cursor.show()

waarde wordt toegekend aan variabelen H en W	W = platform.window:width() H = platform.window:height()
toekenning van startposities van de rechthoeken	xstartpos = W*0.8 xstartneg = xstartpos + 2*Unit ystartx2= H/4 ystart1 = ystartx2 + 7*Unit ystartx= ystartx2 + 4*Unit …
de functie count() telt het de waarden van de rechthoeken die versleept zijn	count() platform.window:invalidate()
EINDE functie on.resize	end …

Met de functie on.activate() krijgen we de cursor in beeld. We merken op dat we het commando, cursor.show() , kunnen herhalen in functies zoals on.resize() of on.MouseUp() . Dit om zeker te zijn dat de cursor niet verdwijnt.

Vervolgens schrijven we het commando cursor.show()in de functie on.resize. Zo blijft de cursor zeker in beeld. Verder worden de variabelen en de startwaarden een waarde toegekend, de waarden van de rechthoeken geven een eindresultaat weer en tenslotte wordt het beeld gerefreshed.

Naast het gebruik van de muis, kan men ook het toetsenbord gebruiken of een toetsenbord voorzien in het programma zelf. De essentie is het programma aan te passen aan het bestaand platform.

Hoofdstuk 5: Lua Physics Engine

De Physics Engine is toegevoegd aan Lua in de 3.2 update. Dit is een krachtige bibliotheek die de mogelijkheid biedt om een omgeving aan te maken waarin objecten kunnen gedefinieerd worden met realistische eigenschappen. Zo krijgt de gebruiker de mogelijkheid om accurate simulaties uit te voeren.

In dit hoofdstuk wordt er basisinzicht in het script meegegeven.

1 Werken met één object

Aangezien de Physics Engine pas beschikbaar is sinds de 3.2 update, moet de APIlevel op 2.0 geplaatst worden.

Vooraleer er aan de code van de Physics Engine gewerkt kan worden, moeten de variabelen gedefinieerd worden. In voorgaande hoofdstukken worden de variabelen gedeclareerd in de functies zelf. Het is veel efficiënter om je variabelen bovenaan je script globaal de declareren en niet in de functies zelf. Dit om de eenvoudige reden dat zo de variabelen eenmalig moeten gedefinieerd worden en niet telkens opnieuw in iedere functie.

Actie	**Code**

Alle variabelen worden bovenaan het programma gedeclareerd, zodat het mogelijk is om ze in iedere functie te gebruiken.	`local W`
	`local H`
	`local space`
	`local newBody`
	`local newShape`
	`local mass`
	`local inertia`
	`local elasticity`
	…

Vervolgens wordt er een nieuwe functie `init` aangemaakt waarin we onze omgeving, object en vorm zullen definiëren.

Het allerbelangrijkste is het `require` commando die toegang geeft tot de gelinkte bibliotheek. *Require "physics"* geeft toegang tot de physics bibliotheek, die uiteraard noodzakelijk is om met de Physics Engine te werken. *Require "color"* geeft toegang tot een aantal voor gedefinieerde kleuren en is het niet noodzakelijk meer om met de RGB getallen te werken (color.orange, color.yellow…).

De omgeving waarin het object bestaat moet gedefinieerd worden via `physics.Space()`. Binnenin deze omgeving werkt een er zwaartekracht, die uiteraard als functie heeft om het object naar beneden te trekken.

Nadat de omgeving is aangemaakt, kan het object gedefinieerd worden. Het object ontstaat met behulp van *physics.Body(mass, inertia)*. Zoals je kan zien moet je met deze "body" 2 argumenten meegeven, nl.: de massa van het object en het traagheidsmoment van het object. Daarna kunnen er een aantal eigenschappen meegegeven worden met de body zoals de startpositie, de snelheid…

Nu het object gedefinieerd is moet het een bepaalde vorm aannemen. De mogelijkheden zijn een segment, een cirkel, een rechthoek en een polygoon. Met deze vorm kunnen er andere eigenschappen meegegeven worden zoals elasticiteit en wrijving.

Uiteindelijk voegen we het object met zijn vorm toe aan de omgeving.

Actie	**Code**

Functie init wordt aangemaakt	function init(gc)
	  W = platform.window:width()
	  H = platform.window:height()

De nodige requires om de bibliotheken op te halen	require “physics” require “color”
De omgeving wordt aangemaakt	space = physics.Space() mass = 100
Elasticiteit met waarde 1 is perfect elastisch. Waarden kleiner dan 1 zorgen voor minder elasticiteit. Waarden groter dan 1 kunnen voor interessante situaties zorgen, maar meestal geeft dit een error.	width = W/10 gravity = 9.8 elasticity = 1
Frictie groter dan 0 zorgt voor energieverlies bij iedere botsing	friction = 1
De zwaartekracht van de omgeving wordt bepaald via een vector. Uiteraard is de zwaartekracht verticaal en heeft dus als x-component 0, de y-component bestaat hier uit de variabele gravity.	space = setGravity(physics.Vect(0, gravity))
Het moment wordt hier aangemaakt via de physics.misc.momentForCircle (massa, binnenstraal, buitenstraal, objectafwijking)	inertia = physics.misc.momentForCircle(mass,0, width,physics.Vect(0,0))
Ontstaan van het object met de nodige massa en moment. Het object krijgt ook een snelheid mee. De snelheid wordt opnieuw bepaald via een vector met x- en y-component 1000. Deze begint bij de linkerbovenhoek van het scherm, met de y-as naar beneden gericht en de x-as naar rechts. Het object krijgt ook een beginpositie, eveneens met een vector bepaald. Met x- en y-component 0 begint het object linksbovenaan.	newBody = physics.Body(mass, inertia) newBody: setVel(physics.Vect(1000,1000)) newBody: setPos(physics.Vect(0,0))
Hier krijg het object een vorm, meer bepaald een cirkel. Hij wordt toegekend aan het object, krijg een bepaalde straal en beginpositie.	newShape = physics.CircleShape(newBody, width/2, physicsVect(0,0))
Het object met zijn vorm krijgt hier een bepaalde elasticiteit en wrijving mee.	newShape: setRestitution(elasticity) newShape: setFriction(friction)

Nu wordt het object met zijn vorm toegevoegd aan de omgeving.	space: addBody(newBody) space: addShape(newShape) on.paint = paint paint(gc) timer.start(0.01) end

Om het programma te laten functioneren, is er uiteraard meer nodig dan een functie init alleen. De functie paint (vroeger on.paint) moet er zeker in. Deze is te programmeren met de kennis uit voorgaande hoofdstukken. In bovenstaande code is er echter een klein verschil, de regel on.paint = paint. Dit omdat de init functie eerst gerund wordt, waar alles gedeclareerd en geïnitialiseerd wordt. Hierachter wordt het programma gestuurd naar de paint functie met behulp van bovenstaande regel. Het dient ook als een mogelijke reset, wanneer het scherm wijzigt van afmetingen zal de init functie eenmalig opnieuw worden gerund, daarna gaat het terug naar de paint functie. Dit betekent dat de functie init en alles erin maar een maal runt en niet telkens alles opnieuw geïnitialiseerd en gedeclareerd wordt wanneer het object wijzigt van beweging die in de paint functie staat.

Actie	**Code**
Functie paint wordt aangemaakt Een voorbeeld van hoe de require color werkt	function paint(gc) … Gc: setColorRGB (color.orange) … end

Verder moet er nog een on.timer functie in, die nagenoeg hetzelfde is als in voorgaande hoofdstukken.

Een andere belangrijke functie die nog toegevoegd moet worden is de on.resize. Deze zal ervoor zorgen dat wanneer het scherm wijzigt, de functie init opnieuw zal gerund worden (en zo het script reset). Dit laat toe dat het programma altijd rekening houdt met de afmetingen van het scherm en gebruikt maakt van de correcte proporties.

Actie	**Code**
Functie paint wordt aangemaakt Een voorbeeld van hoe de require color werkt	function on.resize() on.paint = init end

2 Pauzeren en resetten

Om te kunnen pauzeren hoeft enkel de timer functie gewijzigd te worden. In de vorige voorbeeldcode stond de timer op 0.01. Indien we willen pauzeren hoeven we enkel te zorgen dat deze op 0 komt te staan, want 0 keer het scherm “refreshen” betekent dat het beeld stil staat. Om dit te kunnen doen is het handig om een extra variabele te declareren (hier pauze genoemd) en deze in je on timer functie te stoppen.

Om het pauzeren wat handiger te maken, kan er gebruik gemaakt worden met een knop, waarbij dan een image ingevoegd wordt en met mousetracking gewerkt wordt. Om de code in het voorbeeld

eenvoudig te houden, wordt er gewerkt met een on.key functie. In deze functie wijzigt de variabele pauze tussen 0 en 1, wat het scherm pauzeert en herstart.

Actie	Code

	function on.enterKey()
	  pause = pause == 1 and 0 or 1
	  platform window.invalidate()
	end

Nu de mogelijkheid er is om te pauzeren, is het ook makkelijk om de code op een of andere manier te resetten. In voorgaande code werd er gezegd dat indien het scherm “geresized” werd, de code automatisch terug naar de functie init sprong en dus in feite reset. Dit kan makkelijk in een nieuwe functie gestopt worden, hier terug in een on.key om het simpel te houden. Het enige wat deze functie doet teruggrijpen naar de on.resize functie.

Actie	Code

	function on.escapeKey()
	  on.resize()
	  platform window.invalidate()
	end

3 Werken met meerdere objecten

Om met meerdere objecten te werken is het makkelijk om met tabellen te werken, een tabel om de objecten in te plaatsen, de andere voor de vorm. Verder wijzigt er niet veel aan de code, enkel wordt er nu gebruik gemaakt van een lus die ervoor zorgt dat ieder object behandeld wordt als in de voorbeeldcode voor 1 object.

Om de opsmuk wat op te beuren is er in de code hieronder wat extra geprogrammeerd om ervoor te zorgen dat niet ieder object dezelfde kleur heeft..

Actie	Code
De nodige extra variabelen worden bovenaan de code gedeclareerd.	local totalBodies local bodies local shapes local initX local initY local Colors local colorNum
De functie init wordt aangemaakt.	function init(gc) ...
Hier wordt er een tabel aangemaakt met een aantal kleuren in. Hieruit kan later een kleur genomen worden om aan een object toe te kennen. colorNum is een referentienummer naar de tabel.	require "color" Colors = {color.gray, color.red, color.orange, color.yellow, color.green, color.blue, color.black} colorNum = 1

De tabellen voor de objecten en de vormen worden aangemaakt.	bodies = {} shapes = {}
De variabele die het aantal objecten bepaald wordt gedefinieerd, er kan ook makkelijk met een variabele n gewerkt worden.	totalBodies = (var.recall("n") or 5)
Hier start de lus om ieder object te definieren zoals in voorgaande voorbeelden. De lus begint bij 1 en stopt bij het aantal objecten eerder gedefinieerd.	for i=1,totalBodies do newBody = physics.Body(mass, inertia) newShape = physics.CircleShape(newBody, width, physics.Vect(0,width/2)) newShape:setRestitution(elasticity) newShape:setFriction(friction) newBody:setVel(physics.Vect(1000,1000))
De objecten krijgen een willekeurige plaats op het scherm	initX = math.random(W) initY = math.random(H) newBody:setPos(physics.Vect(initX, initY))
Het object met zijn vorm worden toegevoegd aan de tabellen. Het laatste dat gebeurt voor de lus opnieuw start is het object met zijn vorm toevoegen aan de omgeving.	table.insert(bodies, newBody) table.insert(shapes, newShape) space:addBody(newBody) space:addShape(newShape) end end
De functie paint wordt aangemaakt.	function paint(gc)
De lus start om ieder object te kunnen behandelen in de paint functie	for k=1,totalBodies do ...
De objecten worden uit te tabel gehaald om te definieren.	bodies[k]:setPos(physics.Vect(posX, posY)) bodies[k]:setVel(physics.Vect(velX, velY))
Een kleur wordt gekozen uit de kleurentabel	chooseColor = Colors[((colorNum + k) % #Colors) + 1] gc:setColorRGB(chooseColor) gc:fillArc(posX, posY, width, width, 0, 360) end end
De on.resize functie blijft ongewijzigd.	function on.resize() on.paint = init end

4 Afbakenen van de ruimte

Tot nu toe is er altijd gewerkt met het beperken van de ruimte met de afmetingen van het scherm. In dit subhoofdstuk zal dit gedaan worden met static shapes, het voordeel hiervan is een nog realistischere simulatie. De muren kunnen nu bepaalde eigenschappen krijgen zoals wrijving en elasticiteit.

De “muren” waarmee de omgeving afgezet wordt, zijn statisch en kunnen dus als static shapes gedefinieerd worden. Voor statische shapes moet geen body gedefinieerd worden, aan deze kunnen rechtstreeks bepaalde eigenschappen meegegeven worden.

In de code kunnen nu de 4 stellingen, die controleren of de objecten in de afgezette ruimte zitten, verwijderd worden. Deze zijn overbodig geworden, aangezien er nu muren zullen worden geplaatst waar de objecten tegen stuiteren.

De makkelijkste manier om deze muren te realiseren is via een nieuwe functie: bounds. In deze functie worden eerst ale static shapes verwijderd vooraleer er nieuwe aangemaakt worden. Dit is omdat de bounds functie ook in de init functie zal geplaatst worden zoals met paint gedaan is in vorige voorbeelden. Het op voorhand verwijderen van de static shapes zorgt ervoor als het venster opnieuw geresized wordt, dat er geen muren op elkaar komen te liggen. Daarna worden de static objects aangemaakt en in een tabel geplaatst voor comfort.

Actie	**Code**
Tabel wordt gedeclareerd	`local boundaries`

De functie init wordt aangemaakt
 	
Tabel boundaries wordt geïnitialiseerd
	`function init(gc)`
	`  ...`
	`  boundaries = {}`
	`  ...`
	`  space = physics.Space()`

De init functie zal hiermee teruggrijpen naar de bounds functie.	`  bounds(W, H)`
	`  ...`
	`end`

De functie bounds wordt aangemaakt.
 	
De ipairs commando wordt hier gebruikt om door een tabel te lopen (ipv een for lus)
 	
Alle statische objecten worden eerst verwijderd.
 	
Een variabele wordt in de functie gedeclareerd
De muur wordt aangemaakt, krijgt bepaalde eigenschappen mee en wordt dan aan de omgeving en tabel toegevoegd.
	`function bounds(w, h)`
	
	`  for _,wall in ipairs(boundaries) do`
	
	`    space:removeStaticShape(wall)`
	`  end`
	
	`  local bound`
	`  bound = physics.SegmentShape(nil,`
	`  physics.Vect(0,0), physics.Vect(w, 0), 1)`
	`    : setRestitution(elasticity)`
	`    : setFriction(friction)`
	`  space:addStaticShape(bound)`
	`  table.insert(boundaries, bound)`

De 2e muur wordt aangemaakt en toegevoegd.	bound = physics.SegmentShape(nil, physics.Vect(0,H), physics.Vect(w, h), 1) : setRestitution(elasticity) : setFriction(friction) space:addStaticShape(bound) table.insert(boundaries, bound)
Hier wordt de 3e muur aangemaakt en toegevoegd.	bound = physics.SegmentShape(nil, physics.Vect(0,0), physics.Vect(0, h), 1) : setRestitution(elasticity) : setFriction(friction) space:addStaticShape(bound) table.insert(boundaries, bound)
Tenslotte wordt de laatste muur aangemaakt en toegevoegd.	bound = physics.SegmentShape(nil, physics.Vect(w,0), physics.Vect(w, h), 1) : setRestitution(elasticity) : setFriction(friction) space:addStaticShape(bound) table.insert(boundaries, bound) end

Hoofdstuk 6: Toepassingen van Lua in TI-Nspire

1 Wiskundige toepassingen

1.1 Textboxes en Rich text input

In dit deel van de toepassingen maken we gebruik van de 2D-editor om een Textbox te creëren waar we tekst kunnen invoeren. Deze toepassing is handig indien we wiskundige of chemische formules willen invoeren. Nog een voordeel van deze boxes is dat men slechts eenmaal de lay-out van de input moet definiëren in de broncode, en zo zal alle input de gewenste lay-out krijgen. Verder laat het de gebruiker toe om in de Textbox een bepaalde lay-out te kiezen voor de geschreven input. Maar deze lay-out en de gebruikte referentie ernaar moet in de broncode beschreven staan.

Indien we geen Textbox gebruiken, zou dit op een langere manier kunnen zoals we in het begin van de handleiding hebben gezien. Dit betekent dat we input van het keyboard moeten kunnen registreren en vervolgens moet ook elk deel lay-out aan de hand van de on.paint functie in de broncode staan.

Verder merken we ook op dat deze Textbox een **dynamisch** veld is. Indien men input schrijft, kunnen we voorwaarden in de code stellen. Vervolgens kunnen we feedback geven op de input die gegeven was. Een voorbeeld: we geven een optellingsom, de gebruiker moet het antwoord van deze som in de Textbox geven en vervolgens controleren we of dit een correct antwoord is. We merken algauw dat het gebruik van deze Textboxes enorm toepasbaar is.

A Aanmaken van een Textbox

Als eerste leggen we aan de hand van een voorbeeld uit hoe we een Textbox maken in een programma. Op voorhand merken we op dat er geen on.paint nodig is voor een box te maken.

Actie	Code
vermelden van apilevel	platform.apilevel = "2.0"

BEGIN functie on.construction
  we stellen de timer op 0.4 seconden
  we maken een Textbox aan
EINDE functie on.construction
	function on.construction()
	  timer.start(0.4)
	  TextBox1 = D2Editor.newRichText()
	end

```
BEGIN functie on.resize                    function on.resize()
  we definiëren de variabelen W en H         W = platform.window:width()
  W en H: hoogte en breedte van het          H = platform.window:height()
  scherm                                     TextBox1:move(W*0.05, H*0.5)
  we definiëren de plaats en de omvang       TextBox1:resize(W*0.9,H*0.4)
  van onze Textbox                           fontSize = math.floor(W/32)
                                             fontSize = fontSize > 6 and fontSize or 7
  fontsize een waarde toekennen. Op deze
  waarde stellen we grenzen in               TextBox1:setFontSize(fontSize)
                                             TextBox1:setBorderColor(500)
  fontsize input Textbox                     TextBox1:setBorder(1)
  kaderkleur van Textbox                     TextBox1:setTextColor(200*200*200)
  kaderdikte Textbox                         TextBox1:setFocus(true)
EINDE functie on.resize                    end

BEGIN functie on.getFocus                  function on.getFocus()
  zet cursor in Textbox                      TextBox1:setFocus()
EINDE functie on.getFocus                  end

BEGIN functie on.timer                     function on.timer()
  zichtbaar maken van de cursor              cursor.show()
  toekennen van waarde aan variabele         Input = TextBox1:getExpression()
Input                                        if Input then
                                               var.store("input", Input)
  we slaan ingegeven tekst in Input op       end

  refreshen van venster                      platform.window:invalidate()
EINDE functie on.timer                     end

BEGIN functie on.escapeKey                 function on.escapeKey()
  verwijdert tekst in Textbox                TextBox1:setExpression(" ")
  refreshen van venster                      platform.window:invalidate()
EINDE functie on.escapeKey                 end
```

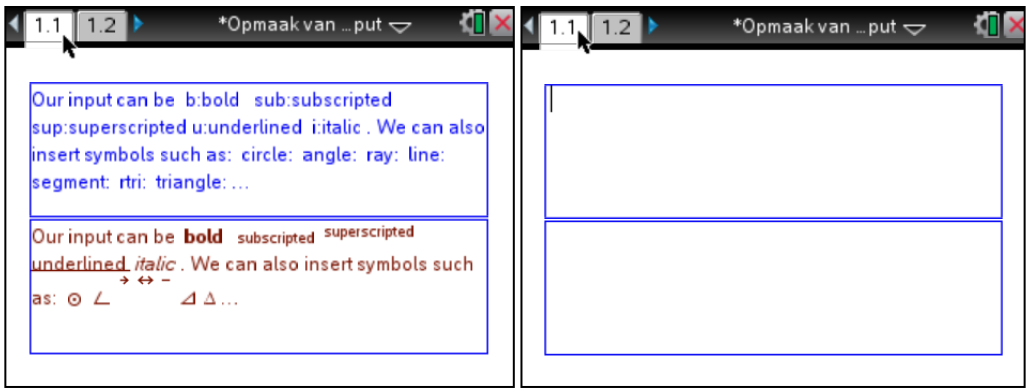
Als eerste declareren we het apilevel dat we nodig hebben om dit programma te laten runnen. Vervolgens maken we de functie `on.construction`, hierin zetten we de timer op 0.4 seconden. Verder maken we een nieuwe Textbox aan met de volgende methode: `D2Editor.newRichText()`.

In de functie `on.resize` bepalen we de afmetingen, de positie, de fontsize voor de input en de opmaak van het kader van de Textbox.

In de funtie `on.getFocus` maken we de methode `setFocus()` aan waarmee we het input-veld van de Textbox activeren. Dit betekent dat de gebruiker meteen kan typen in de Textbox zonder de cursor te gebruiken.

Hierna schrijven we de functie `on.escapeKey` zodat we de ESC-toets kunnen gebruiken om de input in de Textbox geheel te verwijderen. Zo gebruiken we de methode , we lezen dat de methode `setExpression(" ")` als argument een lege String heeft. Nadat we het venster refreshen zal deze lege String ervoor zorgen dat het input-veld leeg is.

B Opmaak van de input



Nu we een Textbox kunnen aanmaken, kijken we hoe we de opmaak van de input kunnen aanpassen. In het volgende voorbeeld maken we twee Textboxes aan, de ene waarin de tekst en de verwijzingen naar de opmaakopdrachten geschreven wordt. In de tweede Textbox zien we de input met de opgeroepen opmaak.

Actie	Code
...	...
BEGIN functie on.timer de variabele Input krijgt een waarde Indien er input is zal deze worden opgeslaan input wordt weergegeven in Textbox 2 refreshen van het venster EINDE functie on.timer	function on.timer() cursor.show() Input = TextBox1:getExpression() if Input then var.store("input", Input) TextBox2:setText(pretty(Input)) end platform.window:invalidate() end
BEGIN functie pretty Als er in de input “circle:” staat zullen we een cirkel op het beeld te zien krijgen. De volgende voor het weergeven van een driehoek, hoek, pijl,.. verlopen analoog. Verder definiëren we de opdrachten voor opmaak van tekst. Zoals onderlijnen, **bold**, *italic*, SUPERSCRIPT, SUBSCRIPT. We maken nu de aangepaste input zichtbaar. EINDE functie pretty	function pretty(input) input = input:gsub("circle:", "\\1circle ") input = input:gsub("triangle:", "\\1tri ") input = input:gsub("angle:", "\\1angle ") input = input:gsub("ray:", "\\1ray ") input = input:gsub("line:", '\\1line ') input = input:gsub("segment:", "\\1lineseg ") input = input:gsub("rtri:", "\\1rtri ") input = input:gsub("vector:", "\\1vector ") input = input:gsub("sup:", "\\1supersc ") input = input:gsub("sub:", "\\1subscrp ") input = input:gsub("b:", "\\1keyword ") input = input:gsub("u:", "\\1title ") input = input:gsub("i:", "\\1subhead ") return input end

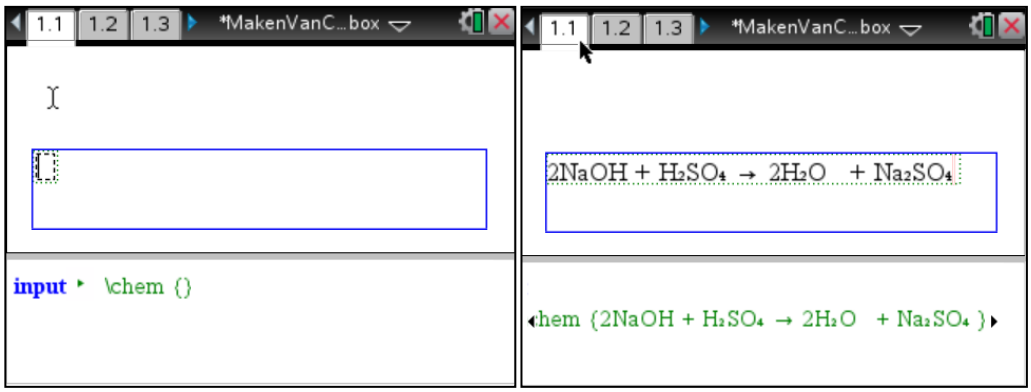
De broncode van dit voorbeeld is analoog met de code van het vorige voorbeeld. We merken wel op we in de functie `on.getFocus` het inputveld van alleen Textbox 1 activeren omdat in dit veld de tekst en opmaakopdrachten worden geschreven. Verder zullen we in de functie `on.resize` de twee Textboxes onder elkaar zetten.

In de functie `on.resize` zullen we de tekst die in Textbox 1 geschreven staat als input beschouwen. Indien er input is zullen we deze opslaan. Hierna zal de input de functie `pretty` doorlopen en vervolgens aangepast in Textbox 2 verschijnen.

De functie `pretty(input)` heeft als argument de input die we in Textbox 1 hebben geschreven. In deze functie hebben verscheidene opmaakopdrachten gedefinieerd. Nadat deze zijn doorlopen, geven we de aangespaste input weer in Textbox 2 aan de gebruiker. Zo bekomen we het resultaat dat te zien is in de afbeeldingen.

1.2 Math- en Chemboxes

Bij deze toepassing maken we gebruik van Textboxes. Omdat de 2D editor wetenschappelijke notatie toelaat kunnen met gewone Textboxes Mathboxes ofwel Chemboxes maken. Deze boxes zijn zeer handig voor wetenschappelijke notatie van wiskundige en chemische vergelijkingen. In het volgende voorbeeld leren we hoe een Mathbox of Chembox wordt aangemaakt.



Actie	Code
BEGIN functie resize … text box 1 is een chem box … EINDE functie resize	function on.resize() … TextBox1:createChemBox() … end
BEGIN functie timer de ingegeven vergelijking slaan we als input op EINDE functie timer	function on.timer() cursor.show() Input = TextBox1:getText() if Input then var.store("input", Input) end platform.window:invalidate() end

BEGIN functie escapeKey
  we verwijderen het chemveld met
  zijn inhoud
EINDE functie escapeKey
 	
BEGIN functie tabKey
  roepen de functie resize op
  nieuw chem-veld is leeg
EINDE functie tabKey
	function on.escapeKey()
	  TextBox1:setExpression("")
	  platform.window:invalidate()
	end
	
	function on.tabKey()
	  on.resize()
	  Input = ""
	end

Na het vermelden van het correcte apilevel schrijven we de functies `on.construction` en `on.getFocus` zoals bij het aanmaken van een Textbox. Vervolgens maken we de functie `on.resize`, deze verloopt analoog met het aanmaken van een gewone Textbox uitgenomen de opdracht: `TextBox1:createChemBox()`. Deze opdracht zorgt ervoor dat er specifiek een Chembox wordt aangemaakt. Indien men nu een Mathbox wil aanmaken, dan zal deze opdracht als volgt luiden: `TextBox1:createMathBox()`.

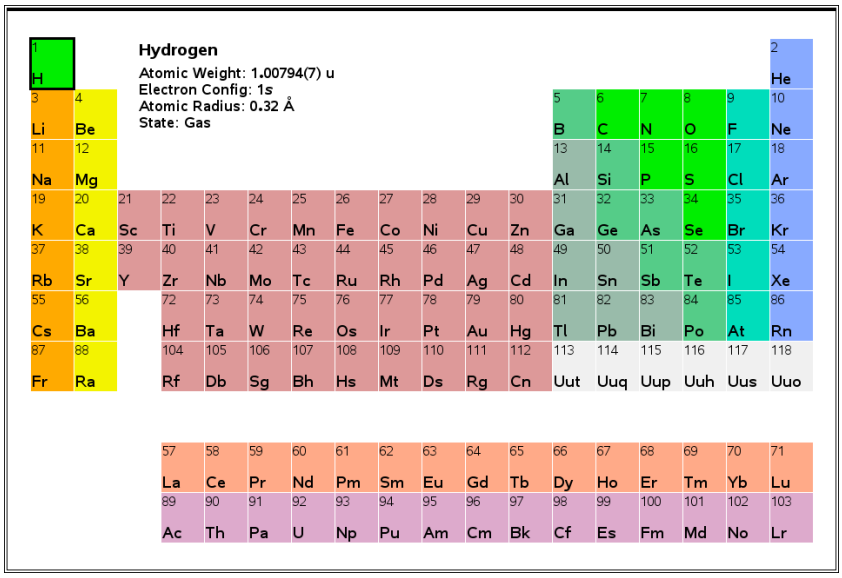
Vervolgens schrijven we de functie `on.timer`, we halen met de opdracht `Input = TextBox1:getText()` de tekst op die in het inputveld wordt geschreven. De rest van deze functie verloopt analoog met het maken van een Textbox.

Met de functie `on.escapeKey` verwijderen we het chemveld en zijn inhoud. Met de functie `on.tabKey` maken we een nieuw chemveld aan.

We merken als laatste op dat een Chembox verscheidene chemvelden kan bevatten door de tabtoets te gebruiken. Een chemveld zal een chemische formule bevatten waarin het "="-teken wordt voorgesteld door een pijl.

Indien we een Mathbox willen maken met mathvelden, zal de broncode analoog zijn met bovenstaand voorbeeld, met uitzondering op de opdracht `TextBox1:createMathBox()`.

2 Chemische toepassingen

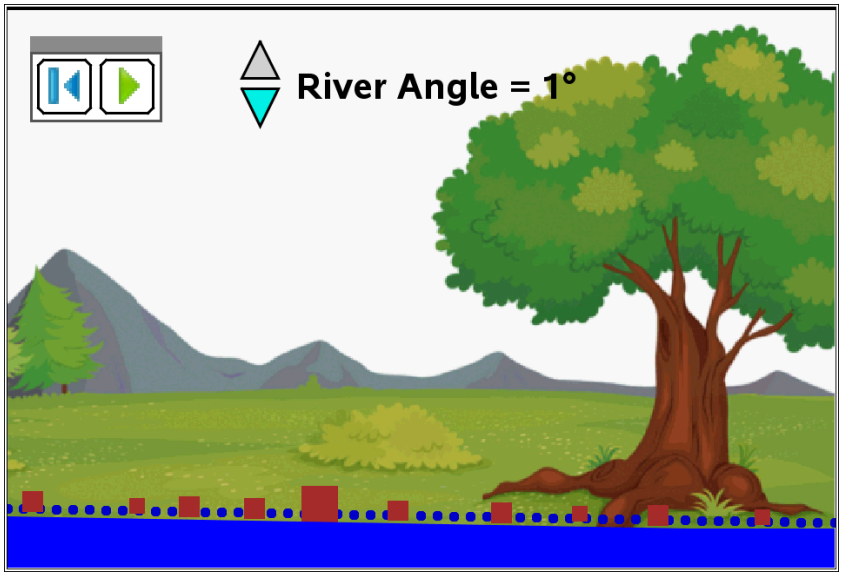


Op de bladzijde hiervoor staat een schermafdruk van de eerste pagina van dit programma. De code wordt weergegeven in volgende tabel.

Actie	Code
Alle elementen worden gedefinieerd door middel van hun atoomnummer.	elements = { -- 1 {name="Hydrogen", symbol="H", weight="1.00794(7)", protons=1, config="1s", radius=0.32, density="0.000082", meltingpoint=-259.1,boilingpoint= -252.762, state="Gas", oxstates="+1,-1", discovered="1766",
Bij group wordt het element in een bepaalde groep ingedeeld zodat deze dezelfde kleur krijgen.	group=1,
Hirebij wordt de plaats van het element in de tabel meegegeven.	row=0, col=0 },

Dit programma gebruikt Lua om het periodiek systeem der elementen op te stellen. Het programma geeft daarnaast mee onder welke klasse deze elementen voorkomen. Hierbij zorgt Lua ervoor dat je slechts eenmaal de opmaak van een soort elementen moet definiëren. Door deze elementen het overeenkomstig nummer toe te voegen, wordt de opmaak automatisch uitgevoerd.

3 Fysische toepassingen



Dit programma maakt gebruik van de Physics Engine. Hierin wordt de erosie van een rivierbedding weergegeven. Lua is hiervoor een perfecte programmeertaal omdat de Physics Engine in Lua relatief eenvoudig kan gevormd worden.

Actie	Code

Dit is de physics engine van dit programma. Dit zorgt ervoor dat de balkjes bewegen, dus opnieuw getekend worden.	function menu:paint(gc)
	  if platform.isDeviceModeRendering() then
	  gc:setPen("thin", "smooth")
	  else
	  gc:setPen("medium", "smooth")
	  end
	
	  gc:setFont("sansserif", "b", 12*sY)
	  gc:setColorRGB(240,240,240)
	gc:fillRect(self.x_offset,self.top_line,self.width,self.height)
	  gc:setColorRGB(0,0,0)
	  if platform.isDeviceModeRendering() then
	  gc:setPen("thin", "smooth")
	  else
	  gc:setPen("medium", "smooth")
	  end
	
	gc:drawRect(self.x_offset,self.top_line,self.width,self.height)
	  local msg = self.header
	  local center_text =(platform.window:width() -
	  gc:getStringWidth(msg))/2
	  gc:drawString(msg, center_text,self.top_line, "top")
	  gc:setFont("sansserif", "r", 9*sY)
	  local msg_height = gc:getStringHeight(self.line_1)
	
	  if self.center == 0 then
	  gc:drawString(self.line_1,self.x_offset+5*sX,self.top_line +
	  2*msg_height , "top")
	  else
	  local msg_offset = (self.width -
	  gc:getStringWidth(self.line_1))/2
	
	  gc:drawString(self.line_1,self.x_offset+msg_offset,
	  self.top_line + 2*msg_height , "top")
	  end
	
	  if self.center == 0 then
	  gc:drawString(self.line_2 ,self.x_offset+5*sX,self.top_line
	  +3*msg_height , "top")
	  else
	  local msg_offset = (self.width -
	  gc:getStringWidth(self.line_2))/2
	  gc:drawString(self.line_2,self.x_offset+msg_offset,
	  self.top_line + 3*msg_height , "top")
	  end
	
	  if self.center == 0 then
	  gc:drawString(self.line_3 ,self.x_offset+5*sX,self.top_line
	  +4*msg_height , "top")
	  else
	  local msg_offset = (self.width -
	  gc:getStringWidth(self.line_3))/2

```
    gc:drawString(self.line_3,self.x_offset+msg_offset,
    self.top_line + 4*msg_height  , "top")
  end

  if self.center == 0 then
    gc:drawString(self.line_4,self.x_offset+5*sX,self.top_line +
    5*msg_height  , "top")
  else
    local msg_offset = (self.width  -
    gc:getStringWidth(self.line_4) )/2
    gc:drawString(self.line_4,self.x_offset+msg_offset,
    self.top_line + 5*msg_height  , "top")
  end

  if self.center == 0 then
    gc:drawString(self.line_5 ,self.x_offset+5*sX,
    self.top_line +6*msg_height   , "top")
  else
    local msg_offset = (self.width  -
    gc:getStringWidth(self.line_5) )/2
    gc:drawString(self.line_5,self.x_offset+msg_offset,
    self.top_line + 6*msg_height  , "top")
  end

  if self.center == 0 then
    gc:drawString(self.line_6 ,self.x_offset+5*sX,
    self.top_line +7*msg_height  , "top")
  else
    local msg_offset = (self.width  -
    gc:getStringWidth(self.line_6) )/2
    gc:drawString(self.line_6,self.x_offset+msg_offset,
    self.top_line + 7*msg_height  , "top")
    end
    gc:setColorRGB(255,0,0)
    local msg = "Click the close box on this menu to begin."
    center_text =(platform.window:width()-
    gc:getStringWidth(msg))/2
    gc:drawString(msg ,center_text,self.top_line +self.height-
    msg_height  , "top")
    gc:setColorRGB(180,180,180)
    gc:fillRect (self.x_offset+self.width-17*sX,
    self.top_line+2*sY, 15*sX,15*sY)
    gc:setColorRGB(0,0,0)
    gc:drawRect (self.x_offset+self.width-17*sX,
    self.top_line+2*sY, 15*sX,15*sY)
    gc:setColorRGB(255,0,0)
    gc:drawLine(self.x_offset+self.width-13*sX,
    self.top_line+6*sY,self.x_offset+self.width-6*sX,
    self.top_line+13*sY)
    gc:drawLine(self.x_offset+self.width-13*sX,
    self.top_line+13*sY,self.x_offset+self.width-6*sX,
self.top_line+6*sY)
end
```

4 Nog een stapje moeilijker

Lua wordt ook nog gebruikt in verschillende computergames. Doordat Lua een logisch denkende taal is, wordt deze vooral gebruikt voor de NPC's, aangestuurd door de AI of atrificiële intelligentie. NPC staat voor Non-Player Character. Wanneer deze NPC's logischer kunnen denken, worden de games veel realistischer. Door Lua als programmeertaal te gebruiken kan de programmeur ook complexer programeren. Een greep uit het aanbod van games die Lua gebruiken:

- **Angry birds**: bij dit programma werd Lua vooral gebruikt om de verschillende balken op een zo logisch mogelijke wijze te laten vallen.
- **The Sims 2: Nightlife**: bij deze game werden vooral de personages die eveneens in de wereld rondlopen met Lua geprogammeerd.
- **Puzzle Quest**: bij deze puzzelgame werden vooral het verschijnen van de verschillende bollen met lua geschreven. Ook het verschijnen van verschillende spreuken werd op een logische manier met Lua geprogrammeerd.
- **World Of Warcraft**: hier werden eveneens de NPC's met Lua geprogrammeerd.

Zoals je ziet, wordt Lua voor verschillende doeleinden gebruikt. Lua wordt tegenwoordig vooral in games gebruikt omdat deze logische programmeertaal zeer handig is om alles realistisch te laten verlopen.

5 Besluit

We besluiten dat Lua voor talloze programma's kan worden gebruikt. Doordat de Lua-programeertaal relatief eenvoudig is, kunnen heel grote programma's verwezenlijkt worden.

Bijkomende info

1 Websites

http://compasstech.com.au/TNS_Authoring/Scripting/index.html
http://lua-users.org/wiki/TutorialDirectory
http://www.inspired-lua.org/
http://www.inspired-lua.org/category/tutorials/tutorials-theory/
http://www.lua.org/
http://lua-users.org/wiki/

2 Literatuur

the Lua Scripting API Reference Guide:
http://education.ti.com/educationportal/sites/US/nonProductSingle/nspire-scripting.html
Programming in Lua 2^{e} edition (Roberto Ierusalimschy)
Beginning Lua Programming (Kurt Jung en Aaron Brown)

LUA is een krachtige programmeertaal die reeds in vele domeinen zijn nut bewezen heeft. Ook in TI-Nspire kan LUA worden gebruikt. Doorheen dit cahier word je stap voor stap ingewijd in de wondere wereld van LUA in TI-Nspire en ontdek je zelf de kracht en de vele toepassingsgebieden van deze programmeertaal. Eenmaal je de basis onder de knie hebt kun je zelf aan de slag om gepersonaliseerde scripts te schrijven en eigen programma's te ontwerpen die in TI-Nspire kunnen draaien.

De auteurs zijn studenten van het 2de jaar industrieel ingenieur aan de KHBO. Dit cahier werd uitgewerkt in kader van een wetenschappelijk project onder begeleiding van Dhr. G. Herweyers.

December 2012