Beginner LÖVE Tutorial

September 14, 2015

1 Like drawing on a canvas

1.1 Your favourite rectangle

A rectangle at position x=100, y=200. 300 pixels in width and 150 in height.

```
function love.draw()
  love.graphics.rectangle("fill",100,200,300,150)
end
```

1. Draw the rectangle at a different position.
2. The screen is of size 800,600. Align the rectangle with the upper right corner.
3. Replace **"fill"** with **"line"**, what happens?
4. Draw a second rectangle.[1]
5. Make the screen white.

1.2 Two rectangles

```
1 function love.draw()
2   love.graphics.setColor(0,255,0)
3   love.graphics.rectangle("fill",100,200,300,150)
4   love.graphics.setColor(255,255,255)
5   love.graphics.rectangle("fill",300,400,100,50)
6 end
```

1. Change numbers in line 2. What happens?
2. This representation of colors is called RGB(RedGreenBlue). Make the smaller box blue.
3. Move the boxes so that they overlap. Which box is in front?
4. Swap lines 3 and 5. What changed?
5. Visit love2d.org/wiki/love.graphics. You will find new shapes to draw there. Pick one and add it to the screen. Give it a nice color.[2]

[1] Adding another function love.draw doesn't work, because every function love.draw defines what should be drawn to the screen. If you have multiple definitions only the last one counts.

[2] You can also download the documentation for offline use (http://goo.gl/cwzDuc)

1.3 Some lines

```lua
function love.draw()
  love.graphics.line(100,0,100,200)
  love.graphics.line(0,200,100,200)
  love.graphics.rectangle("fill",100,200,300,150)
end
```

1. Move the rectangle. Adjust the lines accordingly.
2. What do we do if we want to move the rectangle regularly? Variables! Read on.

1.4 Variables

```lua
1  x = 100
2  y = 200
3
4  function love.draw()
5    love.graphics.line(100,0,x,y)
6    love.graphics.line(0,200,x,y)
7    love.graphics.rectangle("fill",x,y,300,150)
8  end
```

1. What would happen if you changed the numbers for x and y?
2. Try renaming x.
3. Change line 2 to: **y = x**. What does this mean?
4. Change line 2 back to: **y = 200**. Change line 1 to: **x = y**. You will get an error. Can you correct the code?
5. Introduce a variable for the width of the box.

2 Interaction

2.1 A moving picture

```lua
x = 100
y = 200

function love.draw()
  love.graphics.line(100,0,x,y)
  love.graphics.line(0,200,x,y)
  love.graphics.rectangle("fill",x,y,300,150)
end

function love.mousepressed()
  x = x + 10
end
```

1. Try clicking the game. Something should happen. Notice the lines behaving nice.

2. Make the box go backwards.
3. Make the box go upwards.
4. Make the box bigger on mouse-press.
5. Add x = 400 after line 4. Did you expect the new behaviour?[3]

2.2 Asking for the right click

[4]

```
 1 a = 100
 2 b = 200
 3
 4 function love.draw()
 5   love.graphics.rectangle("fill",a,b,300,150)
 6 end
 7
 8 function love.mousepressed(mx, my)
 9   if mx < 400 then
10     a = a + 10
11   end
12 end
```

1. When does the box move?
2. Replace line 9 by **if mx $<$ 400 and a $<$ 200 then**. What does this do?
3. Let the box touch the border but let it go no further.
4. **A or B** is true when at least one of A,B is true. Let the box go outside the screen with a secret click area.
5. Add another 'if block' after 11. Let the box go up if the upper part of the screen is pressed.
6. If blocks can do more:
   if *condition* then
   this is executed if condition is true
   else
   this is executed if condition is false
   end
   Use this to let the box go down if you click the lower part of the screen.
7. Give the user some way to reset the boxes position.

*Note: By convention **A and B or C** is the same as **(A and B) or C** and therefore not the same as **A and (B or C)***

[3]Why does this happen? The game engine clears the screen to black 60 times per second. The program you wrote inside **function love.draw()** is executed after that *every time*. So x is 400 every time the rectangles are drawn.

[4]On phones touches are interpreted as clicks

2.3 It needs to do things on its own

```
x = 100
y = 200

function love.draw()
  love.graphics.rectangle("fill",x,y,300,150)
end

function love.update()
  y = y - 1
end
```

Everything inside the **love.update** block is done 60 times per second.[5]

1. Stop the box at the top.
2. At the top of the code insert **velocity = 1**. Let the box move with **y = y - velocity** instead of **y = y - 1**.
3. Make the box loose 0.01 velocity over time. This is gravity.
4. Give the box velocity when you click it[6]
5. Stop the box at the bottom.
6. Print the velocity to screen with **love.graphics.print(velocity,10,10)**
7. Take the boxes velocity away when it hits the top border.
8. Give the player a goal. Signify if the player has reached that goal.
   You are welcome to ask for help with your own idea.
   Example: A cheap parking game. Draw a line at height 100. Change the color of the box, if 0.5 > velocity > -0.5 and 101 > y > 99.

2.4 Loops

The while-loop executes the program written into it *(its body)* as long as its condition **y < 500** is true.

```
x = 0

function love.draw()
  y = 0

  while y < 500 do
    love.graphics.rectangle("fill",x,y,300,150)
    y = y + 200
  end
end
```

[5]Actually, if your computer is too slow love.update is done less often. If you write **function love.update(dt)** instead, dt will be a variable filled with the amount of seconds since the last time love.update was executed.

[6]Can you make the velocity boost stronger the further away the click is from the box?

```
11
12 function love.mousepressed(mx, my)
13   x = x + 50
14 end
```

1. Put line 8 after line 6. Do you understand what difference this makes?
2. Put line 4 after line 2. The screen should go black. Why is that?[7]
3. Revert your changes, then draw more, but smaller rectangles vertically using the while loop.
4. Introduce a new variable z = 0. Add a new while loop. Make it increase z and let it draw some rectangles horizontally.
5. Move the new while loop inside the old while loop. Now every time y is increased your program goes through all your z values! Use this to make a checkerboard. *You need to move z = 0 into the old loop too!*
6. Let the whole checkerboard be moved my clicking the mouse.
7. Add:
   - **a=1** after line 1
   - **and a == 1** to the condition of the while loop [8]
   - **if a == 1 then a = 0 else a = 1 end** after line 13.[9]

   Change the code so that the rectangles are not drawn when the game starts, but after you clicked the mouse.
8. Let the user only change a when the mouse is on the left side of the screen.
9. Add another variable b and only draw the rectangles when b is 1. Let the user change b in some other way than a.

With the functionality presented so far you can theoretically write any program you want to. Many would be very cumbersome to write though. People created many programming languages, styles and concepts to make programming less cumbersome. The following concepts 'only' make some programs easier, prettier or shorter.

2.5 Lists

The *a* in the code is a list of numbers.[10] Here a[1] is 100, a[2] is 200, a[3] is 500. 1,2 and 3 are called the indices of 100,200 and 500. x <= y is true when x is smaller or equal to y and otherwise false.

[7] y is set once to 0, then incremented to 600 and never changed again. The while loop is not executed because y = 600 > 500.

[8] x == y is true when x and y represent the same value. In contrast: x = y assigns y to x.

[9] Instead of 1 and 0 you can also use true and false. That's nicer to read and more efficient.

[10] Actually a is no list. a only behaves like a list when used like a list. a can in reality not only associate a value with a number(like in lists), but associate a value with any other value(except nil).

```
a = {100,200,500}

function love.draw()
  i = 1
  while i <= 3 do
    love.graphics.rectangle("fill",a[i],a[i],10,10)
    i = i + 1
  end
end
```

1. Add a number to the list a. Now a has 4 elements. a[4] is the fourth element. Make the while loop draw all 4 elements.
2. #a is the length of a. [11] Use the length of a in the condition of your while loop, so you don't have to change any numbers if you add an element to a.
3. You can change the k'th element of the list with a[k]=v for some value v. *Example: a[2] = 123* Write a while loop that sets all values in a to 0.
4. You can also change from undefined (nil) to any other value. Use a while loop to let a be 0,10,20,30,40,50,....,200
5. a[#a+1]=v is the same as adding v to the list. Whenever the mouse is clicked add the x-coordinate of the click to the list.
6. Now create two new empty lists **xs** = **{}** and **ys** = **{}**. With every click add the x-coordinate of the mouse to xs and the y-coordinate to ys. Draw rectangles on all the saved click positions.

2.6 Functions

Functions calculate an output to a given input. They can also output nothing and just execute some code based on the input. They can also ignore their input and just be a fancy name for a block of code. love.draw, love.update and love.mousepressed are special functions which are defined by you but executed by the game engine.[12]

```
function twosquares(x,y)
  love.graphics.rectangle("fill",x,y,50,50)
  love.graphics.rectangle("fill",x+100,y,50,50)
  return x+100+50, y+50
end

function love.draw()
  x1,y1 = twosquares(50,150)
  twosquares(x1,y1)
end
```

[11]The length of a is, by convention, the index k-1 where k is the smallest number with a[k] being undefined. i.e the list stops with its first undefined element. Lua calls undefined nil.

[12]Executing a function is usually called *calling a function*

1. twosquares(50,150) executes the code from lines 2-4 with x being 50 and y being 150. The values *returned* in line 4 are saved in x1,y1.
   Draw another pair of squares at the position returned by twosquares(x1,y1).

2. Create a new function called foursquares(x,y). This function should draw 4 squares all having some distance between them. You can use twosquares.

3. Add this function and call drawatx1y1 in love.draw.[13]

```
function drawatx1y1()
  love.graphics.rectangle("fill",x1,y1,100,100)
end
```

   Any code that is executed after line 8 can use x1 and y1, but often you only need a variable inside a block of code. Sharing that variable with the rest of the code can make it more difficult to understand the code, because small changes can have widespread, hard to track, effects.

   Replace line 8 by **local x1 , y1 = twosquares (50 ,150)**.

   Now drawatx1y1 can not access x1 and y1 anymore. The program should crash.

   How are variables transfered from love.draw to twosquares? Can you replicate that strategy to make drawatx1y1 work again?

4. Create a function pointInRectangle(x,y,rectanglex,rectangley,width,height) which returns true when x,y is inside the rectangle drawn by love.graphics.rectangle("fill", rectanglex,rectangley,width,height) and false otherwise. Create a moving rectangle with love.update. Signify if the player has clicked the moving rectangle using pointInRectangle.

[13]drawatx1y1() calls the function. drawatx1y1 does not! drawatx1y1 only represents a variable holding the function.