First Look at Tables

In this lesson we'll be writing a simple spaceship game. While doing this you'll also learn about tables.

In previous programs you've seen that you can store information in variables. By using variables in an instruction instead of numbers or text, we can later change what's stored in the variables without having to change the instruction.

```
love.graphics.print(message, XPos, YPos)
```

We can change the printed message by assigning new text to the variable **message**, or the location by changing what's in **XPos** and **YPos**.

A table extends this idea one step further.

In Lua, a table consists of two columns and any number of rows. The first column is called the **key**, and the second column is called the **value.** Here's a simple example. I've named this table **Fun**.

Fun

Key	**value**
1	"Juice"
2	43
3	"banana"
4	4

If we what to find a value, we specify its key. For example, the row with the key of 2 contains the number value 43. The row with the key of 3 contains the text value "banana".

In Lua,

Fun[2] means the value where the key is 2. This equals 43
Fun[3] means the value where the key is 3. This equals "banana"

In Lua you can find the value in any row by giving Lua the name of the table and the key. You can also change values the same way.

Fun[2] = "cereal"

Fun[2] still means the value where the key is 2 but now this equals "cereal"

Creating Tables

This is how you create an empty table in Lua and add values.

Fun = {}	create an empty table with no rows
Fun[1] = “Drink”	add a row with key=1 and store the value “Juice”
Fun[1] = “Juice”	change the value from “Drink” to “Juice”
Fun[2] = 43	add a row with key=2 and store the value 43
Fun[3] = “banana”	add a row with key=3 and store the value “banana”
Fun[4] = 4	add a row with key=4 and store the value 4

You can also do this all at once. Lua will generate the keys starting with 1 and adding 1 each time a new row is generated.

Fun = {”Juice”,43,”banana”,4} creates the same table as above.

Tables with Keys that are Text

Sometimes it’s useful to have keys that aren’t numbers. For example …

Fun

key	value
“drink”	“Juice”
“people”	43
“fruit”	“banana”
“chairs”	4

We could create this table in the same way as before.

Fun = {}	create an empty table with no rows
Fun[“drink’] = “Juice”	add a row with key=”drink” and store the value “Juice”
Fun[“people”] = 43	add a row with key=”people” and store the value 43
Fun[“fruit”] = “banana”	add a row with key=”fruit” and store the value “banana”
Fun[“chairs”] = 4	add a row with key=”chairs” and store the value 4

Or we could do it all at once. Here we specify the keys instead of letting Lua create numeric keys starting at 1.

Fun = {drink=”Juice”, people=4, fruit=”banana”, chairs=4}

Note that the keys are **NOT** surround by quotes. Only the text values are.

Some programmers who work in other languages like 'C' find the notation Fun["drink"] kind of messy so Lua allows you to write Fun.drink instead. Using this different (but equivalent) way of accessing tables we could create the table as follows.

```
Fun = {}
Fun.drink = "Juice"          the same as Fun["drink"] = "Juice"
Fun.people = 43              the same as Fun["people"] = 43
Fun.fruit =  "banana"        the same as Fun["fruit"] = "banana"
Fun.chairs = 4               the same as Fun["chairs"] = 4
```

It's a bit neater and I personally like this last way of adding items to a table and reading them back.

This is probably confusing for now, so don't worry about it too much. We'll come back to this later. Mainly a table is a convenient way to group together related information. This will become important when we start to create specialized functions to do common tasks. So far our programs have been simple enough to not need them. But here's an example.

Suppose we write a program that will animate two balls. Instead of writing the instructions to move a ball twice we might write a function called …

```
Function moveBall()
```

To use this we'd have to tell the function where the ball is now and how it's moving. If we use simple variables, we might have to write something like …

```
moveBall(ball1_posX, ball1_posY, ball1_velX, ball1_velY)
moveBall(ball2_posX, ball2_posY, ball2_velX, ball2_velY)
```

If instead we create a table for each ball

Ball1

Key	value
"posX"	100
"posy"	100
"velX"	1.2
"velY"	3.4

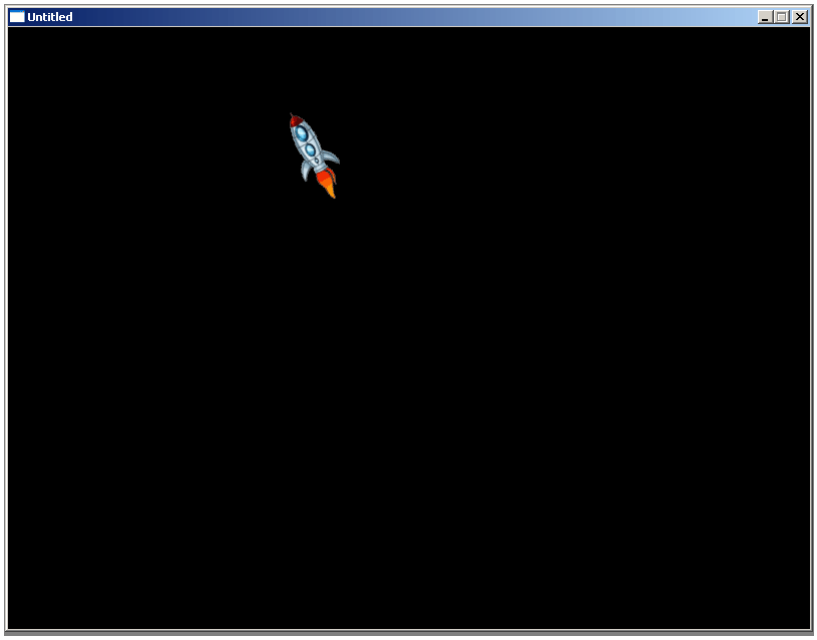
We could write

```
moveBall(ball1)
moveBall(ball2)
```

and the function could lookup the values it needs from the table.

We'll explore this more later on.

Our Space Game



In this program we will create a small rocket that you can fly around the screen. You can steer the rocket using the "A" and "D" keys to turn, and the space bar to fire your engine. It's a bit involved so we'll take it step by step.

Create an empty Proj03 folder and the standard main.lua file with the three functions love.load(), love.draw() and love.update(). Save the three attached images into the same Proj03 folder.

```
1   spaceship = {}
2
3   function love.load()
4       width = love.graphics.getWidth()
5       height = love.graphics.getHeight()
6
7       spaceship.coasting = love.graphics.newImage("spaceship00.gif")
8       spaceship.posX = width/2
9       spaceship.posY = height/2
10  end
11
12  function love.draw()
13      love.graphics.draw(spaceship.coasting, spaceship.posX, spaceship.posY, 0, 1, 1, 21, 36)
14  end
15
16  function love.update()
17  end
18
```

In line 1 we create an empty table called spaceship. This is where we will gather all the information needed to draw our ship and make it fly.

In love.load() you'll see that we're getting the Love window's width and height as we did in the last program.

In line 7 we are loading a image into the spaceship table under the key "coasting". This is a picture of the spaceship without the engine firing. We are also creating an X and Y position for the spaceship under the keys "posX" and "posY". These keys remember where the spaceship is and are initially set to the middle of the screen.

Finally in love.draw() we use the **love.graphics.draw** function to display the rocket picture in the Love window. You'll notice that there are some extra arguments this time. Many Love functions have optional arguments that you can ignore if you don't need them.

```
love.graphics.draw(spaceship.coasting, spaceship.xPos, spaceship.xPos, 0, 1, 1, 21, 36)
```

The first three entries are 'what to draw', in our case the coasting spaceship image, and where to draw it in the X and Y directions.

The fourth item is the angle of rotation. By specifying an angle, we can get Love to draw the rocket pointing in different directions.

The fifth and sixth arguments are stretching amounts. Values smaller than 1 will squash the image, whereas values bigger than 1 will stretch the image.

Finally, the last two arguments are offsets and tell Lua where the middle of the picture is. This is the point of the image that will fall on top of spaceship.xPos and spaceship.xPos when we draw it. It's also the point around which the picture will rotate when we specify a rotation value. As our rocket image is 43 pixels wide, I set the Xoffset to 21. I tried a few values for the Yoffset until it looked right.

Save and run the program. You should see the spaceship with no flames centred in the screen.

Adding Actions

```
1  spaceship = {}
2
3  function love.load()
4      width = love.graphics.getWidth()
5      height = love.graphics.getHeight()
6
7      spaceship.coasting = love.graphics.newImage("spaceship00.gif")
8      spaceship.boost1 = love.graphics.newImage("spaceship01.gif")
9      spaceship.boost2 = love.graphics.newImage("spaceship02.gif")
10     spaceship.useImage = "coasting"
11
12     spaceship.posX = width/2
13     spaceship.posY = height/2
14     spaceship.direction = 0
15
16     spaceship.turnSpeed = 0.017
17 end
18
19 function love.draw()
20 love.graphics.draw(spaceship[spaceship.useImage], spaceship.posX, spaceship.posY, spaceship.direction, 1, 1, 21, 36)
21 end
22
23 function love.update()
24     if love.keyboard.isDown("a") then
25         spaceship.direction = spaceship.direction - spaceship.turnSpeed
26     end
27
28     if love.keyboard.isDown("d") then
29         spaceship.direction = spaceship.direction + spaceship.turnSpeed
30     end
31 end
32
```

Let's add a few more things to the program. First, in lines 8 amd 9 we'll load two more images of the rocket. These are both pictures with flames coming from the engine. If we draw one and then the other, the flame will appear to flicker.

We've also created a few more table entries. spaceship.useImage tells us which image to display. spaceship.direction will tell Love how to rotate the image and spaceship.turnspeed will tell Love how quickly to change that direction when we press 'A' or 'D'.

Finally, we modify the love.graphics.draw function.

```
love.graphics.draw(spaceship[spaceship.useImage], spaceship.xPos, spaceship.xPos,
spaceship.direction, 1, 1, 21, 36)
```

This deserves a closer look. In line 10, spaceship.useImage is assigned the text "coasting". Therefore …

`spaceship[spaceship.useImage]` is currently the same as `spaceship["coasting"]` which is the same as `spaceship.coasting`.

If you check line 7, you'll see that spaceship.coasting is the location in the table where we stored the picture of the coasting rocket. So that's the image that will be drawn. We've also added spaceship.direction to control the rotation of the image.

Finally, in love.update you'll see that we are calling the function love.keyboard.isDown(). This function is used to check whether the specified key is being pressed.

If the "a" key is being pushed, we increase the spaceship.direction by a small amount; spaceship.turnSpeed. If the "d" key is being pushed instead, we decrease the direction instead.

If you run this program you should see the rocket in the middle of the screen and you should be able to rotate it using the "a" and "d" keys.

The Rocket's Red Glare

```lua
spaceship = {}

function love.load()
    width = love.graphics.getWidth()
    height = love.graphics.getHeight()

    spaceship.coasting = love.graphics.newImage("spaceship00.gif")
    spaceship.boost1 = love.graphics.newImage("spaceship01.gif")
    spaceship.boost2 = love.graphics.newImage("spaceship02.gif")
    spaceship.useImage = "coasting"

    spaceship.posX = width/2
    spaceship.posY = height/2
    spaceship.direction = 0

    spaceship.acceleration = 0.01
    spaceship.turnSpeed = 0.017

    spaceship.velX = 0
    spaceship.velY = 0
end

function love.draw()
love.graphics.draw(spaceship[spaceship.useImage], spaceship.posX, spaceship.posY, spaceship.direction, 1, 1, 21, 36)
end

function love.update()
    spaceship.posX = spaceship.posX + spaceship.velX
    spaceship.posY = spaceship.posY + spaceship.velY

    if spaceship.posX > width + 75 then
        spaceship.posX = -75
    elseif spaceship.posX < -75 then
        spaceship.posX = width + 75
    end

    if spaceship.posY > height + 75 then
        spaceship.posY = -75
    elseif spaceship.posY < -75 then
        spaceship.posY = height + 75
    end

    if love.keyboard.isDown("a") then
        spaceship.direction = spaceship.direction - spaceship.turnSpeed
    end

    if love.keyboard.isDown("d") then
        spaceship.direction = spaceship.direction + spaceship.turnSpeed
    end

    if love.keyboard.isDown(" ") then

        --some trigonometry magic
        spaceship.velX = spaceship.velX + math.sin(spaceship.direction) * spaceship.acceleration
        spaceship.velY = spaceship.velY + math.cos(spaceship.direction) * -spaceship.acceleration

        if spaceship.useImage == "boost1" then
            spaceship.useImage = "boost2"
        else
            spaceship.useImage = "boost1"
        end
    else
        spaceship.useImage = "coasting"

    end

end

```

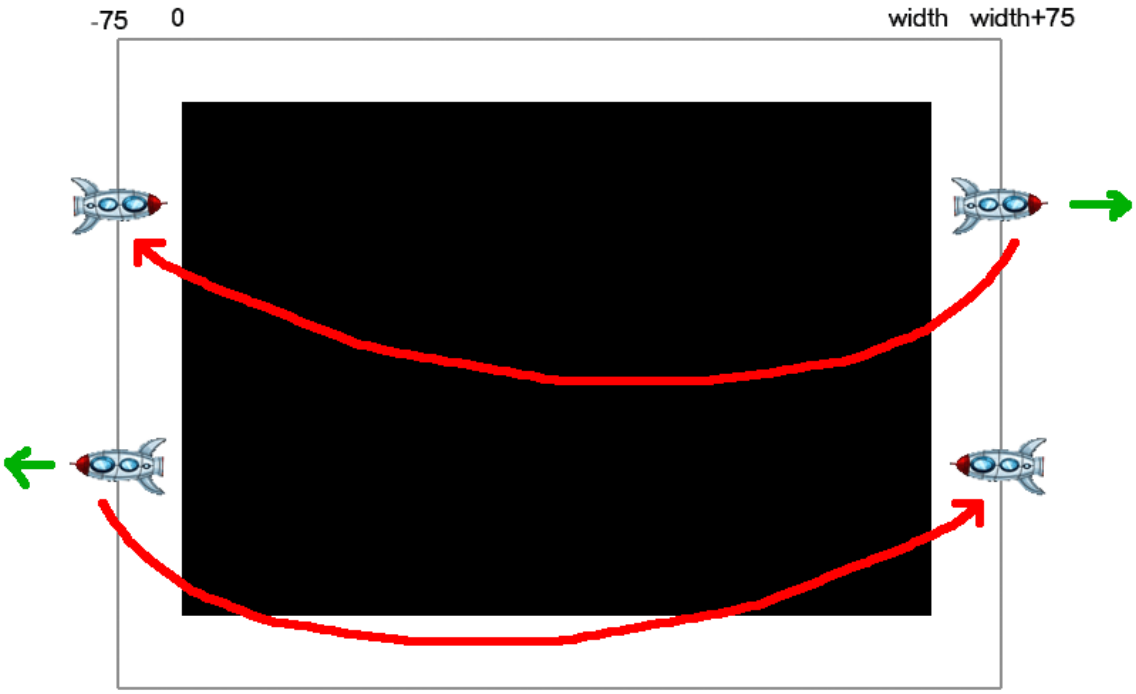
Now we'll add the fun stuff. You'll remember from a previous program that we can add motion to an object by incrementing its position during love.upate(). We'll do the same with our rocket.

In program listing above, you'll see that I'm incrementing the rocket's position in love.update(). I am adding the velocity to the position. Once you give your rocket some velocity in a direction, it will continue to float that way.

```
54      spaceship.posX = spaceship.posX + spaceship.velX
55      spaceship.posY = spaceship.posY + spaceship.velY
```

When the rocket moves out of the window I'd like it to reappear from the opposite direction. I test the spaceship.posX and if it's greater than the window width+75 I know it's out of view, so I reset the X position to -75. Since the rocket's motion is unchanged, it will reappear from the left side moving in the same direction as before.

If the X position is less than -75 then the ship reappears from the right. I do the same in the Y direction but test against height+75 and -75 instead.




```
31      if spaceship.posX > width + 75 then
32              spaceship.posX = -75
33      elseif spaceship.posX < -75 then
34              spaceship.posX = width + 75
35      end
36
37      if spaceship.posY > height + 75 then
38              spaceship.posY = -75
39      elseif spaceship.posY < -75 then
40              spaceship.posY = height + 75
41      end
```

Fire All Engines

Finally we reach the point where we want to fire the rocket's engines. When we hold down the space key, we want to alternately show the two rocket images with flames. This simple animation will make the flames flicker.

```
51      if love.keyboard.isDown(" ") then

57              if spaceship.useImage == "boost1" then
58                      spaceship.useImage = "boost2"
59              else
60                      spaceship.useImage = "boost1"
61              end
62      else
63              spaceship.useImage = "coasting"
64
65      end
```

You'll notice that we use `if then … else … end` when testing whether the space key is down.

If the space key is down, we run the instructions between `then` and `else`.
If the space key is up, we run the instructions between `else` and `end`.

In line 63, you can see that when the space key is not being pushed, spaceship.useImage will always be set to the coasting image.

In line 57, we check to see which boost image we're showing. If it's currently boost1 we switch to boost2. If it's boost2, we switch back to boost1. This way, everytime we hit love.update(), the boost image will be swapped for the other while the space key is being held down.

Finally, there are two lines 54 and 55 that include some math you haven't learned yet. These are trigonometry functions that can tell you how to move in the X and Y depending on the direction you're pointing. You'll learn about these in High School so for now we'll just call it math magic.

While the space key is pressed, these two lines of code will add small amounts to the spaceship velocity in the X and Y directions that will push the ship in the direction that it's pointing. To slow down, you need to turn the ship around and boost in the opposite direction. Just like a real spaceship.

So, I think that's all for now.