Your first game with LÖVE - The Fireboat

We are now ready to put our hands into LÖVE and create our first game: a Fireboat throwing water at flames falling from the sky.

Setup the game and draw the fireboat

For our first game we create a `Fireboat` directory somewhere on your computer (`Documents` or the `Desktop` are good places for it).
In this directory we will put all the files for our game (and nothing else).

Start the text editor and create the `conf.lua` file:

```
-- Configuration file that gets read by Löve
function love.conf(t)
    t.title = "Fireboat Tutorial"
    t.version = "0.10.1" -- The LÖVE version targetted
    t.window.width = 400
    t.window.height = 600

    t.console = true -- For debugging on windows
end
```

The `conf.lua` file is automatically read by LÖVE. The `love.conf(t)` function we define in `conf.lua` is automatically executed before the game starts.

The main attributes we are setting in the configuration file are the title and the size of the window of the game.
If the `t.version` variable does not match the version of Löve you downloaded, Löve will show a "Compatibility Warning": you need to adjust it to match your version.

A complete list of the attributes you can set can be found in the LÖVE wiki.

Our next step is to create the image of the fireboat we want to draw. Images for the game belong into a subdirectory of the `Fireboat` directory called `assets`. Go ahead and create that now. Then browse the internet to find a suitable image for the fireboat. Use your favorite image editing tool to shrink it to perhaps 150 x 100 pixels and savie it as a `.png` file in our new `assets` directory.

Before we start programming, we need an image for the fireboat.

All images for the game belong into a subdirectory of the `Fireboat` directory called `assets`. Go ahead and create it now. Then browse the internet to find a suitable image for the fireboat. You need to edit the image in a drawing tool like Gimp and shrink it to a width of about 150 pixels. We will save it as a `png` file in our new `assets` directory and name it `fireboat.png`.

Now we can start writing the game's code: back to the `Fireboat` directory, we create the file called `main.lua`. Open it with the text editor and type this code:

```
debug = true

playerImg = nil
```

```
--[[
Called exactly once when the program starts: allows us to load the assets
--]]
function love.load(arg)
    playerImg = love.graphics.newImage('assets/fireboat.png')
end

--[[
Called very often (each frame) by the love engine
--]]
function love.draw()
    love.graphics.draw(playerImg, 175, 500) -- Draw it towards the bottom of
the window
end
```

In the same way as LÖVE is calling the `love.conf(t)` function we have defined in `conf.lua`, LÖVE will also:

- Call `love.load(arg)` once when the program starts.
- Call `love.draw()` very frequently to redraw the window.

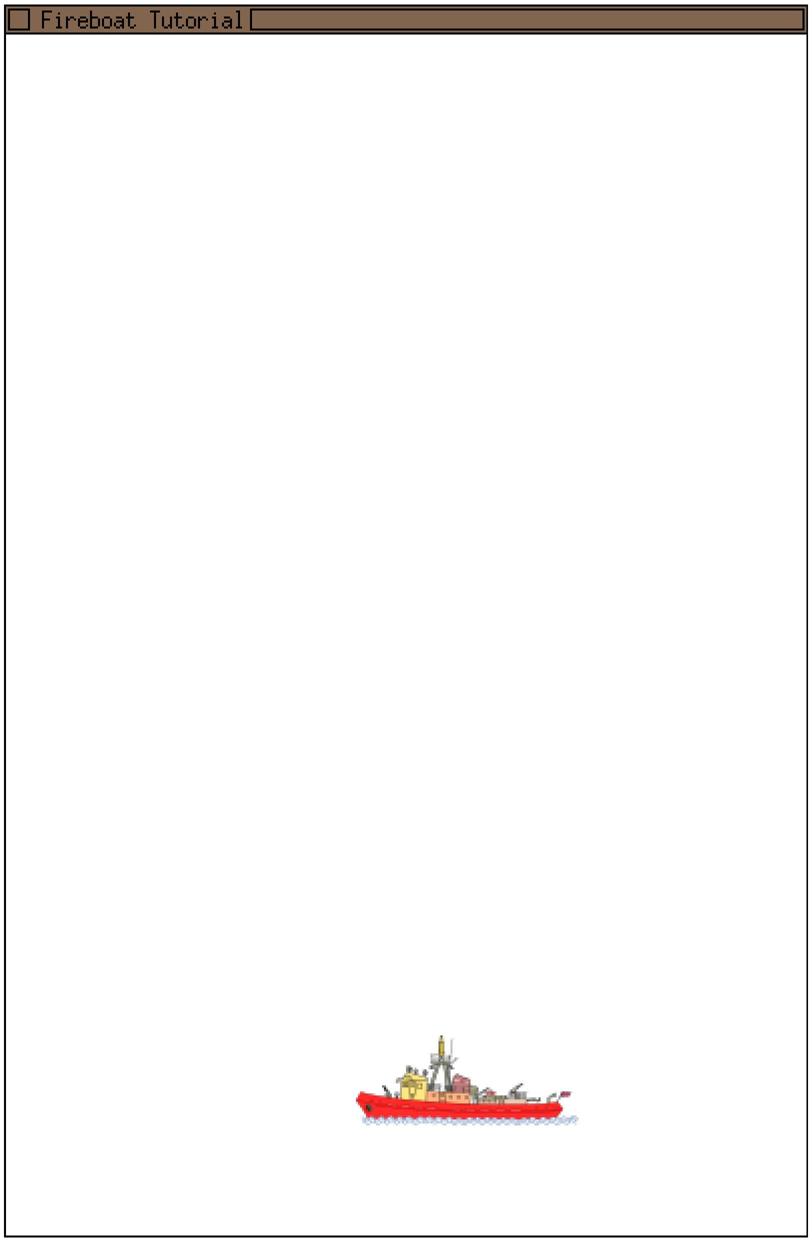
We call these `callback` functions as LÖVE decides when to call them but we get to write what they do. (todo: ale wonders what rich means...)

In this very first version our game is loading the `fireboat.png` image and drawing it at the coordinates `175, 500`.

Our application is ready to run!

As explained in the Getting started chapters for Linux, Mac OS X, and Windows, depending on your operating system and your personal preferences you can now

- On Windows or Mac drag the the `Fireboat` directory onto the `Love` application icon or
- On Linux run `love .` in the terminal while being inside the `Fireoboat` directory



The window created by LÖVE with the fireboat

You can close the game by clicking on the usual "Close" icon of the window.

TODO: add small screenshots for some Window managers?

The complete source for this first example can be found on GitHub.

Stearing the Boat

In our game, the Fireboat will move to the left and the right, avoiding the falling fire flames and squrting water drops at them.

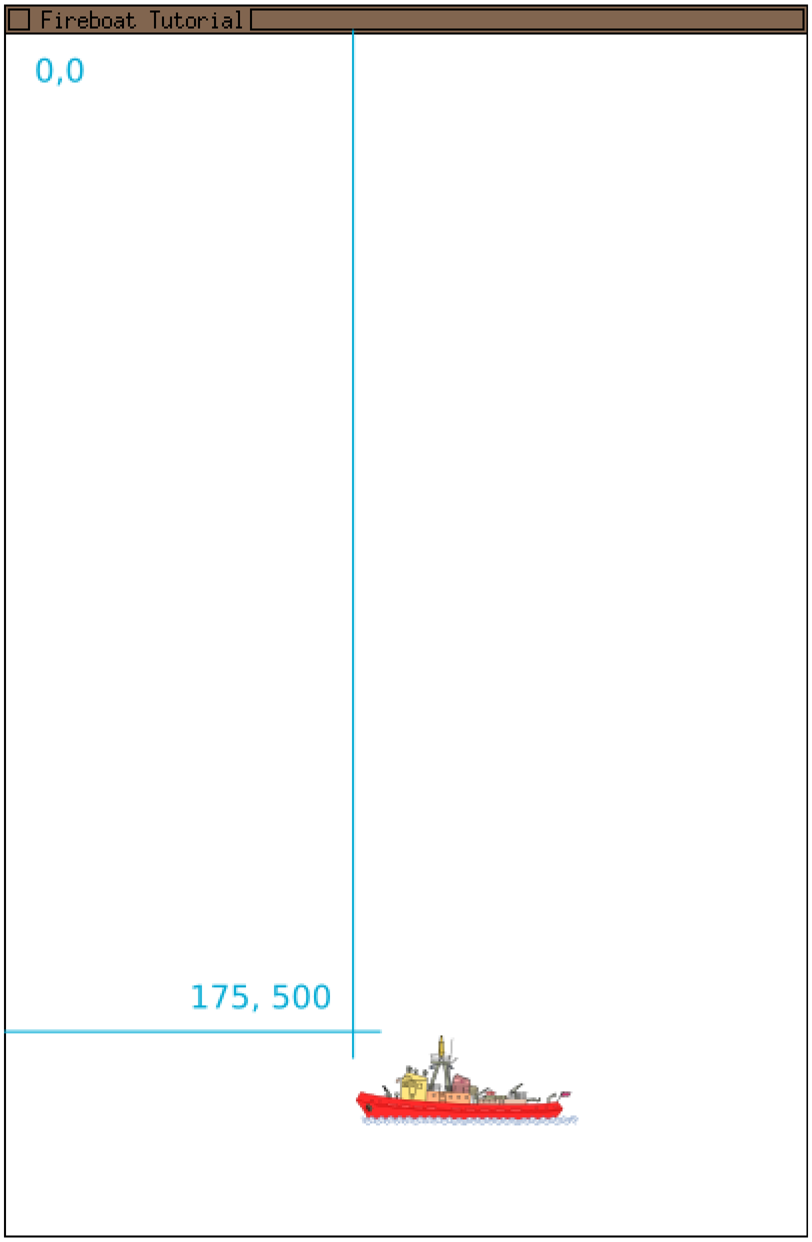
The first step is to define a `player` object that will keep all the details about the position of the boat and what it looks like.
Modify the `main.lua` file and replace the `playerImg` line with the `player =` line, change the `love.load` function and the `love.draw` as shown below:

```
debug = true

player = { x = 175, y = 500, speed = 150, img = nil }

--[[
Called when the program starts: allows us to load the assets
Called exactly once.
--]]
function love.load(arg)
    player.img = love.graphics.newImage('assets/fireboat.png')
end

--[[
Called very often (each frame) by the love engine
--]]
function love.draw()
    love.graphics.draw(player.img, player.x, player.y) -- draw it towards at
the position (x, y)
end
```

The `fireboat.png` image, the horizontal (`x`) and the vertical (`y`) position are now defined as members inside the `player` object.

The `love.load()` function loads `fireboat.png` into the `player.img` member variable. When LÖVE calls `love.draw()` the program draws the fireboat loaded into `player.img` at the at the position defined by `player.x` and `player.y`.

A remark: the coordinates have their origin -- the `(0, 0)` point -- in the top left corner: `(175, 500)` is the distance in pixels from the top left corner of the game area and top left corner of the image itself.



Coordinates origin is the upper left corner

We are now ready for binding the boat's movements with the keyboard.
To do this we need to add the `love.update(dt)` callback function to our `main.lua` file. We write the new function between the `love.load(arg)` callback function and the `love.draw()` callback function.

This is typical for many games:

- They load the assets,
- then they figure out what has moved
- and finally draw all the objects on the screen.

The last two steps are repeated very quickly until the game is shut down.

```
--[[
Called very often (each frame) by the love engine
dt is the amount of time elapsed since the last callback
--]]
function love.update(dt)
    -- Check if the escape key is pressed and quit the game
    if love.keyboard.isDown('escape') then
        love.event.push('quit')
    end

    -- Left arrow and 'a' move the player to the left, right arrow and 'd' 
move to the right
    if love.keyboard.isDown('left','a') then
        player.x = player.x - (player.speed * dt)
    elseif love.keyboard.isDown('right','d') then
        player.x = player.x + (player.speed * dt)
    end
end
```

The `love.update()` callback function checks if one of the following five keys have been pressed:

- `Esc`
- ← or `a`
- → or `d`

When the `Esc` key is pressed, `update()` sends a `quit` event to the LÖVE engine.
The program will quit and the window gets closed.

Pressing the arrow keys or `a` or `d` modifies the `x` position of the player.
The next time LÖVE calls `love.draw()` the boat will be drawn in a different location.

Each movement of the player is calculated by multiplying the `player.speed` member variable by the "delta-time" (`dt`) variable that LÖVE is giving us as parameter to the `love.update()` function. `dt` is the time elapsed since the last time LÖVE has called `love.update()` and is used to make the game run at the same pace on computers with different speeds.
`dt` is a very small number. In our code, `player.speed` is 150: This means that in one second the player will move left or right by 150 pixels. Because the computer is calling `love.update()` many hundreds of times in a second we move the player will move only one small amount of the 150 pixels for each update. To make the boat move faster change the `speed` variable so that it moves a greater distance in pixels in a second. As an example, you can try to modify the value of `player = { .... speed = 150 ...}` to a bigger value like 250.

A Bug!

After each modification of your code, you should run the game and test if the changes have the expected result. If you have done so, you will have noticed that the boat can "sail" out of the window!

When a movmement key is pressed, we must ensure that the boat cannot go over the border. With the `math.max()` function we ensure that the boat's `x` value is never smaller than `0`. Likewise with `math.min()` we prevent the boat from leaving the window on the right side. We must also take into account the width of the boat itself. This is our modified `love.update(dt)` callback function:

```
--[[
Called very often by the love engine
dt is the amount of time elapsed since the last callback
--]]
function love.update(dt)
    -- Check if the escape key is pressed and quit the game
    if love.keyboard.isDown('escape') then
        love.event.push('quit')
    end

    -- Left arrow and 'a' move the player to the left
    -- Right arrow and 'd' move to the right
    if love.keyboard.isDown('left','a') then
        player.x = math.max(player.x - (player.speed * dt), 0)
    elseif love.keyboard.isDown('right','d') then
        player.x = math.min(player.x + (player.speed * dt),
            love.graphics.getWidth() - player.img:getWidth())
    end
end
```

The complete source for this stage of the game can be found on GitHub.

Throwing water drops

Now that we are able to stear our boat, we can get to the next task: throw water drops.

First you need an image for the drops. You can draw your own or download the one we're using for our sample code: drop.png.

""" mir göhnt go wondere """

Save it as a `.png` file in the `assets` directory. It should be small, say 50 x 50 pixels.

Next we need a global variable `drop` where we can store all the attributes for each drop (the speed and the image).
Since we will have many drops on the screen, we need a variable which we will use as a `table` to store the many individual drops. You need to add two lines at the beginning of the program next to the definition of the `player` variable.

```
-- ...
player = { x = 175, y = 500, speed = 150, img = nil }
drop = { speed = 250, img = nil }
drops = {} -- Table of drops to be drawn and updated
-- ...
```

We now need to go to the `love.load(arg)` callback function and load the png image into the `drop.img` member variable.

```
-- ...
function love.load(arg)
    player.img = love.graphics.newImage('assets/fireboat.png')
    drop.img = love.graphics.newImage('assets/drop.png')
```

```
end
-- ...
```

How will new drops be created? We want them to shoot up from the middle of the fireboad = player when the gamer presses the space bar. To achieve this we add code to the `love.update()` callback function to detect when the space bar is pressed (`love.keyboard.isDown('space')`). When this is the case we create a variable `newDrop` which has coordinates taken from the `player` variable but the speed and img are taken from the drop object. The newDrop is inserted in the `drops` table.

Add inside `love.update`:

```
-- Create a drop on space at the boat position
if love.keyboard.isDown('space') then
    newDrop = {
        x = player.x + (player.img:getWidth() /  2) - (drop.img:getWidth() /
2),
        y = player.y,
        speed = drop.speed, img = drop.img
    }
    table.insert(drops, newDrop)
end
```

We also want the drop to move upwards until it reaches the top of the window: each time `love.update()` gets called, each drop moves a number of pixels proportional to the time interval `dt` and the `drop.speed`:

Add inside `love.update`:

```
-- Scroll up the position of the drops
for i, drop in ipairs(drops) do
    drop.y = drop.y - (drop.speed * dt)

    if drop.y < 0 then -- Remove the drops when they pass off the screen
        table.remove(drops, i)
    end
end
```

A remark: we added a check to see if the drops reach the top of the screen (`if drop.y < 0`) and remove them from the `drops` table if this is the case.

The `ipars(drops)` function gives us each drop and the matching `i` position index in the table. We use the `i` index to remove the drop from the table if it has passed off the screen.

Next we need to enhance the `love.draw()` function to draw the drops on the screen. We loop again through all the `drops` and display each of them at its `drop.x` and `drop.y` coordinate.

Add the for loop (`for ... do ... end`) to the `love.draw()` function:

```
function love.draw()
    for i, drop in ipairs(drops) do
        love.graphics.draw(drop.img, drop.x, drop.y)
    end
    love.graphics.draw(player.img, player.x, player.y)
end
```

A remark: In the `for i, drop  in ipairs(drops)` loop we are defining a local `drop` variable. This variable is not the same variable as the one at the top of the program even though it has the exact same name. This `drop` is a local variable that is only visible inside of the loop.
The disadvantage of using the same name is that we can't access the `drop` global variable inside the for loop and that can be confusing.
Here it's not a problem since we don't need anything from the global `drop` object.
For more information see the short section on "Scope" in the "Learning Lua" chapter.

Here is the complete program as it stands now:

```
debug = true

player = { x = 175, y = 500, speed = 150, img = nil }
drop = { speed = 250, img = nil }
drops = {} -- Table of drops currently being drawn and updated

--[[
Called when the program starts: allows us to load the assets
Called exactly once.
--]]
function love.load(arg)
    player.img = love.graphics.newImage('assets/fireboat.png')
    drop.img = love.graphics.newImage('assets/drop.png')
end

--[[
Called very often by the love engine
dt is the amount of time elapsed since the last callback
--]]
function love.update(dt)
    -- We need a way to get out of the game...
    if love.keyboard.isDown('escape') then
        love.event.push('quit')
    end

    -- Create a drop on space at the boat position
    if love.keyboard.isDown('space') then
        newDrop = { x = player.x + (player.img:getWidth()/2), y = player.y,
          speed = drop.speed, img = drop.img }
        table.insert(drops, newDrop)
    end

    -- Left arrow and 'a' moves the player to the left, right arrow and 'd'
move to the right
    if love.keyboard.isDown('left','a') then
        player.x = math.max(player.x - (player.speed * dt), 0)
    elseif love.keyboard.isDown('right','d') then
        player.x = math.min(player.x + (player.speed * dt),
          love.graphics.getWidth() - player.img:getWidth())
    end

    -- Scroll up the position of the drops
    for i, drop in ipairs(drops) do
        drop.y = drop.y - (drop.speed * dt)

        if drop.y < 0 then -- Remove drops when they pass off the screen
            table.remove(drops, i)
        end
    end
end

--[[
Called very often by the love engine
--]]
function love.draw()
    for i, drop in ipairs(drops) do
        love.graphics.draw(drop.img, drop.x, drop.y)
    end
    love.graphics.draw(player.img, player.x, player.y)
end
```

The full code for this stage is on Github.

Limiting the water throughput

Right now, when you keep on pressing the space bar a big column of water is thrown out. Our next task is to enforce a small interval between two drops.

We define the interval by extending the `drop` object with two fields: `interval` and `intervalTimer`.

Go to the top of the program and change the `drop` definition:

```
drop = { speed = 250, img = nil, interval = 0.2, intervalTimer = 0 }
```

`interval` is the smallest amount of seconds that has to pass between two shots. `0.2` seconds (or 200 milliseconds) is a very short time, but it's enough to avoid that the drops get *glued* together.

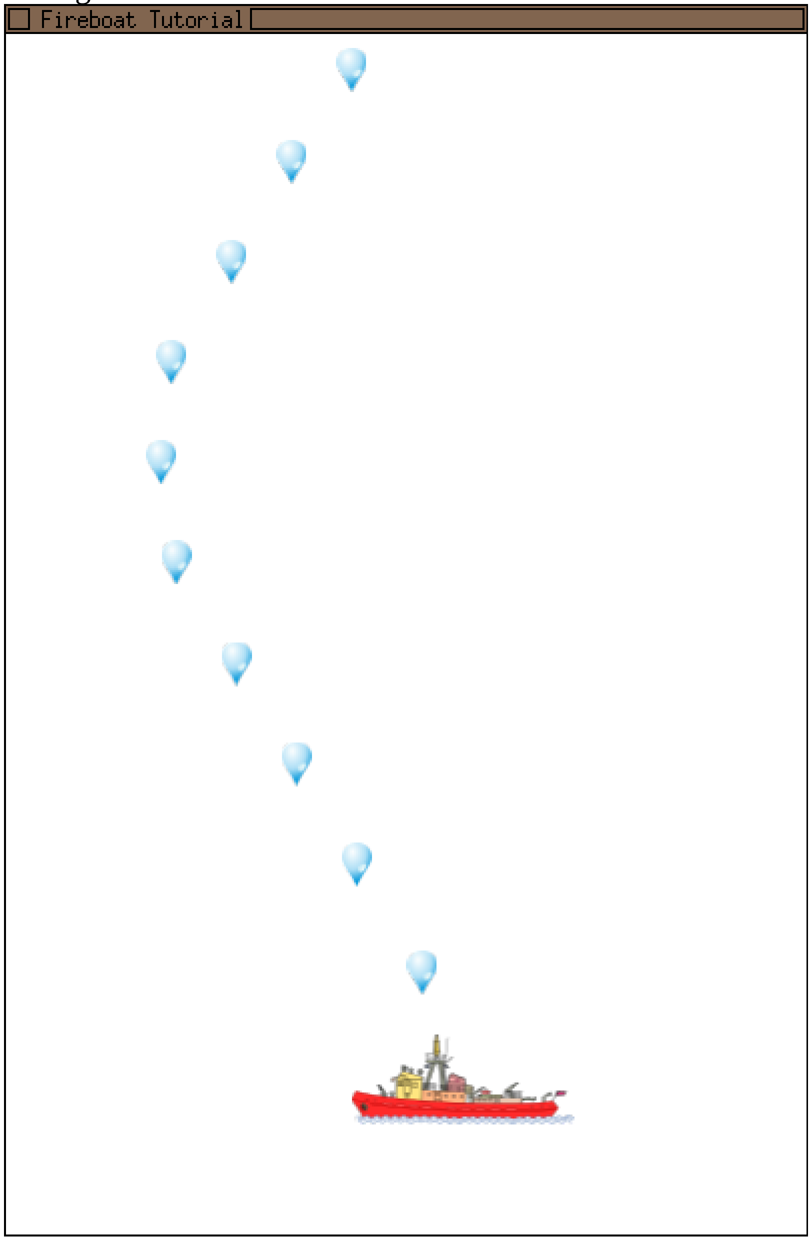
`intervalTimer` counts the time since the last shot. The count is done backwards: each time we produce a drop, we set `drop.intervalTimer` to the value of `drop.interval` and let the `love.update()` function decrease the `drop.intervalTimer` by the elapsed time received in the `dt` parameter. Until it gets down to zero. With the `drop.interval` set to 200ms that means we can shoot a new drop only after this time has passed. We also add a condition to the `if` test so that we can only release a new drop when the `drop.intervalTimer` is smaller than 0 meaning that we have waited for at least 200ms.

Add this to the love.update function and change the if statement with an the extra condition for the timer

```
-- Decrease the drop interval timer before the next drop
drop.intervalTimer = drop.intervalTimer - dt
```

```
-- Create a drop on space at the boat position if intervalTimer got back to
zero
if love.keyboard.isDown('space') and drop.intervalTimer < 0 then
    newDrop = { x = player.x + (player.img:getWidth()/2), y = player.y,
      speed = drop.speed, img = drop.img }
    table.insert(drops, newDrop)
    drop.intervalTimer = drop.interval
end
```

Your game should now look similar to this:



Fireboat with drops

TODO: should we put the full code in some sort of annexe? should we add a chapter called "The code at different stages"? ... in a way that one can use it as "cards"?

Here is what the full program now looks like at this stage:

```
debug = true

player = { x = 175, y = 500, speed = 150, img = nil }
drop = { speed = 250, img = nil, interval = 0.2, intervalTimer = 0 }
drops = {} -- Table of drops currently being drawn and updated

--[[
Called when the program starts: allows us to load the assets
Called exactly once.
--]]
function love.load(arg)
    player.img = love.graphics.newImage('assets/fireboat.png')
    drop.img = love.graphics.newImage('assets/drop.png')
end

--[[
Called very often by the love engine
dt is the amount of time elapsed since the last callback
--]]
function love.update(dt)
    -- Check if the escape key is pressed and quit the game
    if love.keyboard.isDown('escape') then
        love.event.push('quit')
    end

    -- Left arrow and 'a' moves the player to the left, right arrow and 'd'
move to the right
    if love.keyboard.isDown('left','a') then
        player.x = math.max(player.x - (player.speed * dt), 0)
    elseif love.keyboard.isDown('right','d') then
        player.x = math.min(player.x + (player.speed * dt),
          love.graphics.getWidth() - player.img:getWidth())
    end

    -- Decrease the drop interval timer before the next drop
    drop.intervalTimer = drop.intervalTimer - dt


    -- Create a drop on space at the boat position
    if love.keyboard.isDown('space') and drop.intervalTimer < 0 then
        newDrop = { x = player.x + (player.img:getWidth()/2), y = player.y,
          speed = drop.speed, img = drop.img }
        table.insert(drops, newDrop)
        drop.intervalTimer = drop.interval
    end

    -- Scroll up the position of the drops
    for i, drop in ipairs(drops) do
        drop.y = drop.y - (drop.speed * dt)
        if drop.y < 0 then -- Remove drops when they pass off the screen
            table.remove(drops, i)
        end
    end
end

--[[
Called very often by the love engine
--]]
function love.draw()
    for i, drop in ipairs(drops) do
        love.graphics.draw(drop.img, drop.x, drop.y)
    end
    love.graphics.draw(player.img, player.x, player.y) -- draw it towards at
the position (x, y)
end
```

The full code for this stage is on Github.

Adding the falling flames

Just like the drops we first need an image of the flames. Go ahead and look for a suitable image in a 50 x 50 pixel size and save it as `.png` file in the `assets` directory. Just like the drop and drops we need a flame object for the details about a flame and a flames table to remember the position of all of our flames.

Add this at the top of the program where we define the other variables

```
flame = { speed = 200, img = nil, interval = 0.4, intervalTimer = 0 }
flames = {} -- Table of flames currently being drawn and updated
```

The fire drops will fall down with a speed of 200 and there will be a drop every 400 milliseconds.

Add this inside the love.load function add this line

```
function love.load(arg)
    -- ...
    flame.img = love.graphics.newImage('assets/flame.png')
end
```

Next we need to change the `love.update(dt)` function so that after the interval for the flames a new flame is added to the table of flames at a random position across the top of the screen.

Add these lines to the `love.update` function:

```
function love.update(dt)
    -- Decrease the drop interval timer before the next drop/flame
    flame.intervalTimer = flame.intervalTimer - dt

    -- Create a flame at the top with a random x position if intervalTimer got
back to zero
    if flame.intervalTimer < 0 then
        randomX = math.random(10, love.graphics.getWidth() - 10)
        newFlame = { x = randomX, y = -10, speed = flame.speed, img =
flame.img }
        table.insert(flames, newFlame)
        flame.intervalTimer = flame.interval
    end
end
```

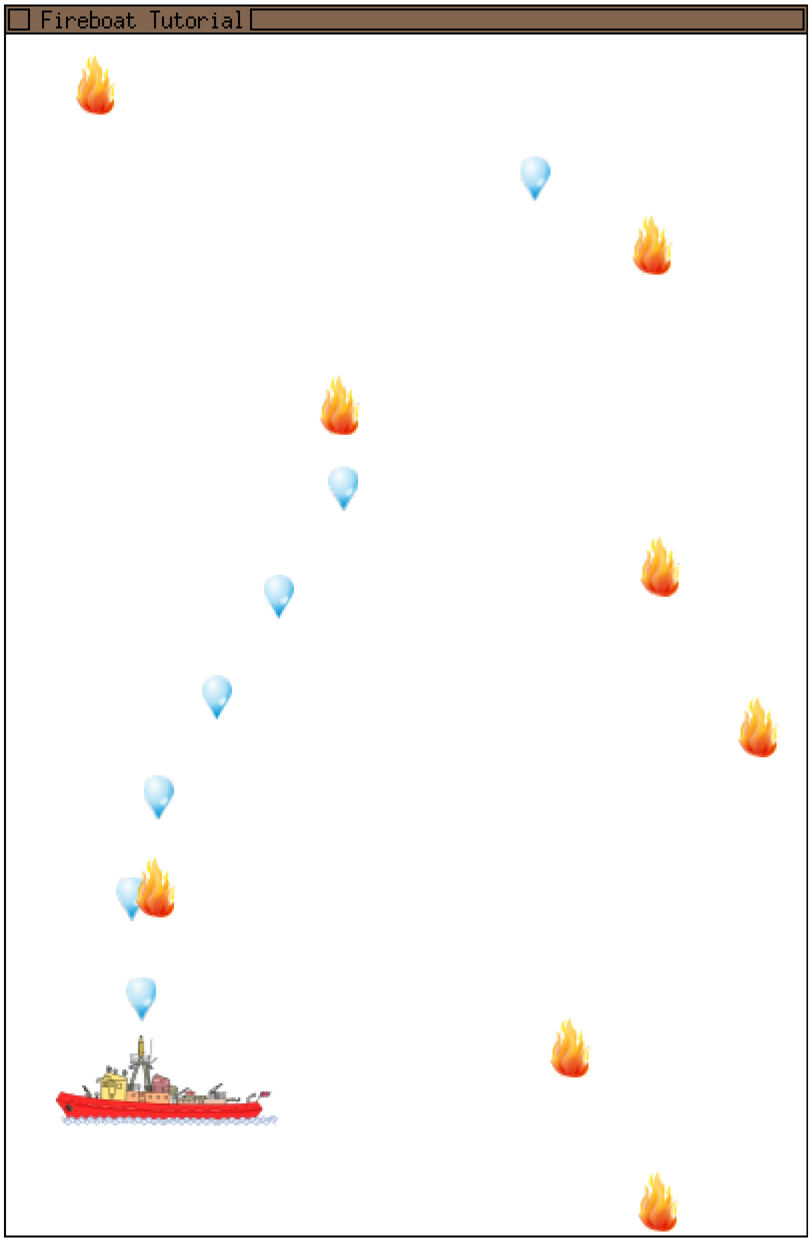
And we also want to have code in the update function that moves the flames downwards. So add these lines to the `love.update(dt)` function:

```
-- Scroll down the position of the flames
for i, flame in ipairs(flames) do
    flame.y = flame.y + (flame.speed * dt)

    if flame.y > love.graphics.getHeight() then -- Remove flames when they
pass off the screen
        table.remove(flames, i)
    end
end
```

Inside the draw function we need to make sure that each flame is drawn on the screen

```
function love.draw()
    -- add these lines to the love.draw function
    for i, flame in ipairs(flames) do
        love.graphics.draw(flame.img, flame.x, flame.y)
    end
end
```

Your game should now look similar to this:



Fireboat with drops

Here is the full program at this stage: ~.lua debug = true

player = { x = 175, y = 500, speed = 150, img = nil } drop = { speed = 250, img = nil, interval = 0.2, intervalTimer = 0 } drops = {} -- Table of drops currently being drawn and updated flame = { speed = 200, img = nil, interval = 0.4, intervalTimer = 0 } flames = {} -- Table of flames currently being drawn and updated

--[[Called when the program starts: allows us to load the assets Called exactly once. --]] function love.load(arg) player.img = love.graphics.newImage('assets/fireboat.png') drop.img =

love.graphics.newImage('assets/drop.png') flame.img = love.graphics.newImage('assets/flame.png') end

--[[Called very often by the love engine dt is the amount of time elapsed since the last callback --]] function love.update(dt) -- Check if the escape key is pressed and quit the game if love.keyboard.isDown('escape') then love.event.push('quit') end

```
-- Left arrow and 'a' moves the player to the left, right arrow and 'd' move
to the right
if love.keyboard.isDown('left','a') then
    player.x = math.max(player.x - (player.speed * dt), 0)
elseif love.keyboard.isDown('right','d') then
    player.x = math.min(player.x + (player.speed * dt),
      love.graphics.getWidth() - player.img:getWidth())
end

-- Decrease the drop interval timer before the next drop
drop.intervalTimer = drop.intervalTimer - dt


-- Create a drop on space at the boat position
if love.keyboard.isDown('space') and drop.intervalTimer < 0 then
    newDrop = { x = player.x + (player.img:getWidth()/2), y = player.y,
      speed = drop.speed, img = drop.img }
    table.insert(drops, newDrop)
    drop.intervalTimer = drop.interval
end

-- Scroll up the position of the drops
for i, drop in ipairs(drops) do
    drop.y = drop.y - (drop.speed * dt)

    if drop.y < 0 then -- Remove drops when they pass off the screen
        table.remove(drops, i)
    end
end

-- Decrease the drop interval timer before the next drop/flame
flame.intervalTimer = flame.intervalTimer - dt

-- Create a flame at the top with a random x position if intervalTimer got
back to zero
if flame.intervalTimer < 0 then
    randomX = math.random(10, love.graphics.getWidth() - 10)
    newFlame = { x = randomX, y = -10, speed = flame.speed, img = flame.img }
    table.insert(flames, newFlame)
    flame.intervalTimer = flame.interval
end

-- Scroll down the position of the flames
for i, flame in ipairs(flames) do
    flame.y = flame.y + (flame.speed * dt)
    if flame.y > love.graphics.getHeight() then -- Remove drops when they pass
off the screen
```

```
        table.remove(flames, i)
    end
end
```
end

--[[Called very often by the love engine --]] function love.draw() for i, drop in ipairs(drops) do love.graphics.draw(drop.img, drop.x, drop.y) end

```
for i, flame in ipairs(flames) do
    love.graphics.draw(flame.img, flame.x, flame.y)
end

love.graphics.draw(player.img, player.x, player.y) -- draw it towards at the
position (x, y)
```
end ~

The full code for this stage is on Github.

Extinguishing the fire

The fire is falling down, the drops are flying up... It's time for something to happen: when a drop touches a flame, the fire extinguishes and the water evaporates. In "game" speak we have to do some "collision detection" and remove both the flame and the drop from their respective table.

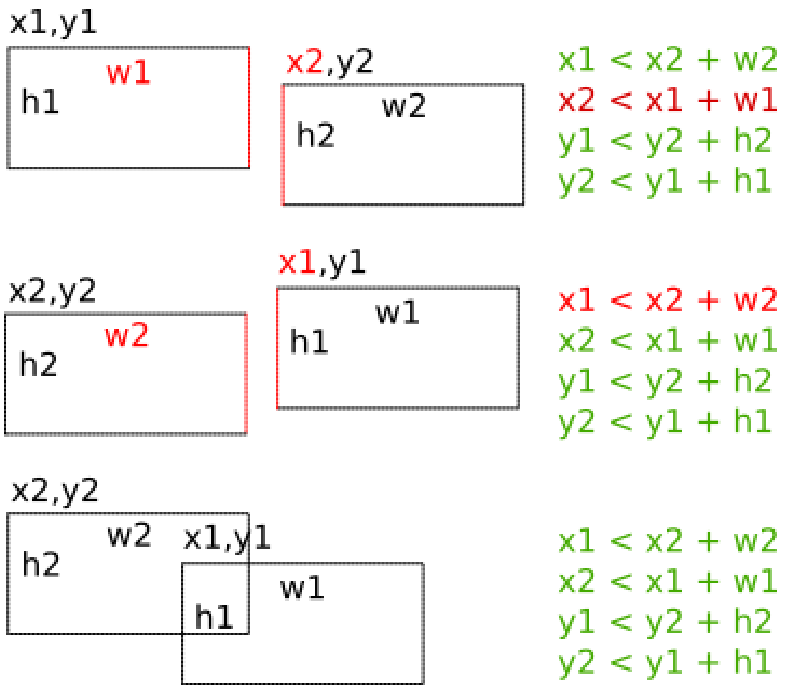
Somewhere at the top of the program -- just after the definitions for the `player` and `drop` objects we add a function from the LÖVE wiki: the `checkCollision()` function will return `true` if the rectange 1 is touching the rectangle 2.

Add the following lines towards the top of the program

```
--[[
Collision detection taken function from http://love2d.org/wiki/BoundingBox.lua
Returns true if two boxes overlap, false if they don't
x1,y1 are the left-top coords of the first box, while w1,h1 are its width and
height
x2,y2,w2 & h2 are the same, but for the second box
--]]
function checkCollision(x1,y1,w1,h1, x2,y2,w2,h2)
    return
        x1 < x2+w2 and
        x2 < x1+w1 and
        y1 < y2+h2 and
        y2 < y1+h1
end
```
The first rectanlge is defined through its top left point (`x1, y1`), its width (`w1`) and height (`h1`). The second one by `x1,y1`, `w2` and `h2`.

There is a collision if the rectangles 1 and 2 intersect: that is, 1's left side (`x1`) is left (<) of 2's right side (`x2 + w2`) and 1's left side (`x2`) is left (<) of 2's right side (`x1 + w1`). And the same applies to the 1 and 2's top and bottom sides.
As already mentioned, for the vertical check you should take into consideration that the coordinate origin is top left (and they grow down).

This is called an algorithm, a set of specific instructions a computer should perform.



check collision

We can now use the `checkCollision()` function to check if any of the drops moving up is touching one of the flames falling down: if it's the case, we simply remove both of them from their respective tables.

```
-- Add these lines to the love.update(dt) function
--[[
Collision detection
Since there will be fewer flames on screen than drops we'll loop them first
--]]
for i, flame in ipairs(flames) do
    for j, drop in ipairs(drops) do
        if checkCollision(flame.x, flame.y, flame.img:getWidth(),
flame.img:getHeight(),
          drop.x, drop.y, drop.img:getWidth(), drop.img:getHeight()) then
            table.remove(drops, j)
            table.remove(flames, i)
        end
    end
end
```

At first sight, it looks like a rather complex process: We loop through all flames (the first `for`) and check if the current flame collides with the any of the drops (the second `for` inside of the first one).

When programming, if you want to check if things "are matching", you mostly have to check each of the possibility and cannot use the common sense to quickly check the most likely ones. But computers are fast at going through tables!

Getting hit by the fire

Each flame that is not caught by the water, can hit the ship and make it sink

We need to track whether the player is alive, so we add an `alive` property to the `player` object:

~~.lua -- Change the player line at the top of the program to this: player = { x = 175, y = 500, speed = 150, img = nil, alive = true }~~

As soon as `player.alive` is set to `false`, we will know that the game is over.

We can use the same `for i, flame in pairs(flames) do` loop in `update()` and the existing `checkCollision()` function to check if a flame has hit the boat.

```
-- inside the function love.update(dt), add the below lines inide the
-- flame for loop (but not inside the drop loop)
    -- ...
    for i, flame in ipairs(flames) do
        -- ...

        if checkCollision(flame.x, flame.y, flame.img:getWidth(),
flame.img:getHeight(),
          player.x, player.y, player.img:getWidth(), player.img:getHeight())
          and player.alive then
            table.remove(flames, i)
            player.alive = false
        end
    end
```

What should happen when the player is not *alive*? The simplest answer is: nothing should happen aynmore. And the simplest way to make *nothing* happen is to wrap all the content of the `love.draw()` function in a *is alive* condition:

```
-- add the if and end statements around the code inside the love.draw function
function love.draw()
    if player.alive then
        -- ...
    end
end
```

Now, the flames, the drops and the boat are only drawn if the player is alive.

The game is working now! But once you get hit you have to get ouf the game and launch it again to ge a new boat: in the next section we will improve that!

Restarting the game

When the boat is on fire, we want to give the player the change to start a new game. In the `love.draw()` we improve the `if player.alive` condition by showing a message when the player is not alive:

```
-- add the else condition
function love.draw()
    if player.alive then
        -- ...
    else
        love.graphics.print("Press 'r' to restart",
          love.graphics:getWidth()/2-50, love.graphics:getHeight()/2-10)
    end
```

```
end
```
When the player is not `alive`, the content of the `else` to the condition makes the `Press 'r' to restart` message to be rendered in the middle of the window.

In a similar way as we are doing with the other key presses, we add a check for the `r` key at the end of the `love.update()` function:

```
-- Add the below lines to the love.update(dt) function
function love.update(dt)
    -- ...
    if not player.alive and love.keyboard.isDown('r') then
        -- remove all drops and flames
        drops = {}
        flames = {}

        -- reset timers
        drop.intervalTimer = 0
        flame.intervalTimer = 0

        -- move player back to default position
        player.x = 175
        player.y = 500

        -- reset our game state
        player.alive = true
    end
end
```
When the player is not *alive* and the `r` key has been ressed, we reset all the objects to the values they had at the beginning of the game: - We clear the table of drops and flames, - We reset the values of the timers , - Move the player to the starting point, - And set the player to be *alive*

Keeping score

Our gamers will probably want to know how good they are at the game: so we need to keep the score. We will give one point each time a flame gets estinguised by a drop.

The first step is to add a `points` field in the `player` object.

```
-- Add the points to the player object
player = { x = 175, y = 500, speed = 150, img = nil, points = 0, alive =
true }
```
Increasing the points is pretty simple: each time we detect a collision between a flame and a drop, we increment `player.score` by one.

```
-- Add the line that increments the score to the collision handling code in
the update function
function love.update(dt)
    -- ...
    for i, flame in ipairs(flames) do
        for j, drop in ipairs(drops) do
            if checkCollision(flame.x, flame.y, flame.img:getWidth(),
flame.img:getHeight(),
              drop.x, drop.y, drop.img:getWidth(), drop.img:getHeight()) then
                table.remove(drops, j)
                table.remove(flames, i)
                player.points = player.points + 1
            end
        end
        -- ...
```
In order to display the current score, we add two lines to the `love.draw()` function:
```
-- add these two lines to the love.draw function (outside the player.alive
test)
function love.draw()
    -- ...
    love.graphics.setColor(255, 255, 255)
    love.graphics.print("SCORE: " .. tostring(player.points), 320, 10)
end
```
First we set the color to white through `love.graphics.setColor(255, 255, 255)`: the color being composed by red, green and blue mixing lot of each base color will give us white (255 is the maximum we can set).

Just after that, we add in the top right corner a text saying `SCORE:` followed by the current value for the score.

Finally, we want to reset the score when the game restarts:
```
-- Add the player.alive  and player.points line to the reset section of the
update function
function love.update(dt)
    -- ...
    if not player.alive and love.keyboard.isDown('r') then
        -- ...
    -- reset the score
        player.points = 0
        --...
        -- reset our game state
        player.alive = true
    end
end
```
The full game:
```
debug = true

player = { x = 175, y = 500, speed = 150, img = nil, points = 0, alive =
true }
drop = { speed = 250, img = nil, interval = 0.2, intervalTimer = 0 }
drops = {} -- Table of drops currently being drawn and updated
flame = { speed = 200, img = nil, interval = 0.4, intervalTimer = 0 }
flames = {} -- Table of flames currently being drawn and updated


--[[
Collision detection taken function from http://love2d.org/wiki/BoundingBox.lua
Returns true if two boxes overlap, false if they don't
x1,y1 are the left-top coords of the first box, while w1,h1 are its width and
height
x2,y2,w2 & h2 are the same, but for the second box
--]]
function checkCollision(x1,y1,w1,h1, x2,y2,w2,h2)
    return
```

```
        x1 < x2+w2 and
        x2 < x1+w1 and
        y1 < y2+h2 and
        y2 < y1+h1
end


--[[
Called when the program starts: allows us to load the assets
Called exactly once.
--]]
function love.load(arg)
    player.img = love.graphics.newImage('assets/fireboat.png')
    drop.img = love.graphics.newImage('assets/drop.png')
    flame.img = love.graphics.newImage('assets/flame.png')
end

--[[
Called very often by the love engine
dt is the amount of time elapsed since the last callback
--]]
function love.update(dt)
    -- Check if the escape key is pressed and quit the game
    if love.keyboard.isDown('escape') then
        love.event.push('quit')
    end

    -- Left arrow and 'a' moves the player to the left, right arrow and 'd'
move to the right
    if love.keyboard.isDown('left','a') then
        player.x = math.max(player.x - (player.speed * dt), 0)
    elseif love.keyboard.isDown('right','d') then
        player.x = math.min(player.x + (player.speed * dt),
          love.graphics.getWidth() - player.img:getWidth())
    end

    -- Decrease the drop interval timer before the next drop
    drop.intervalTimer = drop.intervalTimer - dt


    -- Create a drop on space at the boat position
    if love.keyboard.isDown('space') and drop.intervalTimer < 0 then
        newDrop = { x = player.x + (player.img:getWidth()/2), y = player.y,
          speed = drop.speed, img = drop.img }
        table.insert(drops, newDrop)
        drop.intervalTimer = drop.interval
    end

    -- Scroll up the position of the drops
    for i, drop in ipairs(drops) do
        drop.y = drop.y - (drop.speed * dt)

        if drop.y < 0 then -- Remove drops when they pass off the screen
            table.remove(drops, i)
        end
    end

    -- Decrease the drop interval timer before the next drop/flame
    flame.intervalTimer = flame.intervalTimer - dt

    -- Create a flame at the top with a random x position if intervalTimer got
back to zero
    if flame.intervalTimer < 0 then
        randomX = math.random(10, love.graphics.getWidth() - 10)
        newFlame = { x = randomX, y = -10, speed = flame.speed, img =
flame.img }
        table.insert(flames, newFlame)
        flame.intervalTimer = flame.interval
    end

    -- Scroll down the position of the flames
    for i, flame in ipairs(flames) do
        flame.y = flame.y + (flame.speed * dt)
        if flame.y > love.graphics.getHeight() then -- Remove drops when they
pass off the screen
            table.remove(flames, i)
        end
    end

    --[[
    Collision detection
    Since there will be fewer flames on screen than drops we'll loop them
first
    --]]
    for i, flame in ipairs(flames) do
        for j, drop in ipairs(drops) do
            if checkCollision(flame.x, flame.y, flame.img:getWidth(),
flame.img:getHeight(),
              drop.x, drop.y, drop.img:getWidth(), drop.img:getHeight()) then
                table.remove(drops, j)
                table.remove(flames, i)
                player.points = player.points + 1
            end
        end

        if checkCollision(flame.x, flame.y, flame.img:getWidth(),
flame.img:getHeight(),
          player.x, player.y, player.img:getWidth(), player.img:getHeight())
          and player.alive then
            table.remove(flames, i)
            player.alive = false
        end
    end

    if not player.alive and love.keyboard.isDown('r') then
        -- remove all drops and flames
        drops = {}
        flames = {}
        -- reset timers
        drop.intervalTimer = 0
        flame.intervalTimer = 0

        -- move player back to default position
        player.x = 175
        player.y = 500

        -- reset the score
        player.points = 0

        -- reset our game state
        player.alive = true
    end
end


--[[
Called very often by the love engine
--]]
function love.draw()
    if player.alive then
        for i, drop in ipairs(drops) do
            love.graphics.draw(drop.img, drop.x, drop.y)
        end

        for i, flame in ipairs(flames) do
            love.graphics.draw(flame.img, flame.x, flame.y)
        end

        love.graphics.draw(player.img, player.x, player.y) -- draw it towards
at the position (x, y)
    else
        love.graphics.print("Press 'r' to restart",
          love.graphics:getWidth()/2-50, love.graphics:getHeight()/2-10)
    end
    love.graphics.setColor(255, 255, 255)
    love.graphics.print("SCORE: " .. tostring(player.points), 320, 10)
end
```

Adding Sound

All games need to have sound. We need a sound for launching a water drop, a sound for extinguishing a flame and a sound for getting hit by fire. The website [https://www.freesound.org] seems to have a huge collection of sounds that you are fee to use. Download the three sounds and save them in the `assets` folder as `launch.wav`, `extinguish.wav` and `gameover.wav`. Pay attention to the copyright of the sounds and images and record where you found the assets in the comments.

```
    -- Add these lines to the love.load(arg) function
    -- from https://www.freesound.org/people/lollosound/sounds/124915/ CC0
    launchSound = love.audio.newSource("assets/launch.wav", "static")
    -- from https://www.freesound.org/people/Sparrer/sounds/50043/
```

```
ccbyAttribution
    extinguishSound = love.audio.newSource("assets/extinguish.wav", "static")
    -- from https://www.freesound.org/people/Rocotilos/sounds/178876/ CC0
    gameoverSound = love.audio.newSource("assets/gameover.wav", "static")
```

Now that we have the sounds we need to play them at the right point in the game.

```
    -- Add the sound to the section of the love.update(dt) function where the
new drop is created
    -- Create a drop on space at the boat position
    if love.keyboard.isDown('space') and drop.intervalTimer < 0 then
        newDrop = { x = player.x + (player.img:getWidth()/2), y = player.y,
          speed = drop.speed, img = drop.img }
        table.insert(drops, newDrop)
        drop.intervalTimer = drop.interval
        launchSound:play()
    end
    -- Add the sound to the section of the love.update(dt) function where a
drop and a flame have collided
            if checkCollision(flame.x, flame.y, flame.img:getWidth(),
flame.img:getHeight(),
              drop.x, drop.y, drop.img:getWidth(), drop.img:getHeight()) then
                table.remove(drops, j)
                table.remove(flames, i)
                player.points = player.points + 1
                extinguishSound:play()
            end
    -- Add the sound to the section of the love.update(dt) function where a
drop and a flame have collided
        if checkCollision(flame.x, flame.y, flame.img:getWidth(),
flame.img:getHeight(),
          player.x, player.y, player.img:getWidth(), player.img:getHeight())
          and player.alive then
            table.remove(flames, i)
            player.alive = false
            gameoverSound:play()
        end
```

Full program:

```
debug = true

player = { x = 175, y = 500, speed = 150, img = nil, points = 0, alive =
true }
drop = { speed = 250, img = nil, interval = 0.2, intervalTimer = 0 }
drops = {} -- Table of drops currently being drawn and updated
flame = { speed = 200, img = nil, interval = 0.4, intervalTimer = 0 }
flames = {} -- Table of flames currently being drawn and updated


--[[
Collision detection taken function from http://love2d.org/wiki/BoundingBox.lua
Returns true if two boxes overlap, false if they don't
x1,y1 are the left-top coords of the first box, while w1,h1 are its width and
height
x2,y2,w2 & h2 are the same, but for the second box
--]]
function checkCollision(x1,y1,w1,h1, x2,y2,w2,h2)
```

```
    return
        x1 < x2+w2 and
        x2 < x1+w1 and
        y1 < y2+h2 and
        y2 < y1+h1
end


--[[
Called when the program starts: allows us to load the assets
Called exactly once.
--]]
function love.load(arg)
    player.img = love.graphics.newImage('assets/fireboat.png')
    drop.img = love.graphics.newImage('assets/drop.png')
    flame.img = love.graphics.newImage('assets/flame.png')

    -- from https://www.freesound.org/people/lollosound/sounds/124915/ CC0
    launchSound = love.audio.newSource("assets/launch.wav", "static")
    -- from https://www.freesound.org/people/Sparrer/sounds/50043/
ccbyAttribution
    extinguishSound = love.audio.newSource("assets/extinguish.wav", "static")
    -- from https://www.freesound.org/people/Rocotilos/sounds/178876/ CC0
    gameoverSound = love.audio.newSource("assets/gameover.wav", "static")
end

--[[
Called very often by the love engine
dt is the amount of time elapsed since the last callback
--]]
function love.update(dt)
    -- Check if the escape key is pressed and quit the game
    if love.keyboard.isDown('escape') then
        love.event.push('quit')
    end

    -- Left arrow and 'a' moves the player to the left, right arrow and 'd'
move to the right
    if love.keyboard.isDown('left','a') then
        player.x = math.max(player.x - (player.speed * dt), 0)
    elseif love.keyboard.isDown('right','d') then
        player.x = math.min(player.x + (player.speed * dt),
          love.graphics.getWidth() - player.img:getWidth())
    end

    -- Decrease the drop interval timer before the next drop
    drop.intervalTimer = drop.intervalTimer - dt


    -- Create a drop on space at the boat position
    if love.keyboard.isDown('space') and drop.intervalTimer < 0 then
        newDrop = { x = player.x + (player.img:getWidth()/2), y = player.y,
          speed = drop.speed, img = drop.img }
        table.insert(drops, newDrop)
        drop.intervalTimer = drop.interval
        launchSound:play()
    end

    -- Scroll up the position of the drops
    for i, drop in ipairs(drops) do
        drop.y = drop.y - (drop.speed * dt)

        if drop.y < 0 then -- Remove drops when they pass off the screen
            table.remove(drops, i)
        end
    end

    -- Decrease the drop interval timer before the next drop/flame
    flame.intervalTimer = flame.intervalTimer - dt

    -- Create a flame at the top with a random x position if intervalTimer got
back to zero
    if flame.intervalTimer < 0 then
        randomX = math.random(10, love.graphics.getWidth() - 10)
        newFlame = { x = randomX, y = -10, speed = flame.speed, img =
flame.img }
        table.insert(flames, newFlame)
        flame.intervalTimer = flame.interval
    end

    -- Scroll down the position of the flames
    for i, flame in ipairs(flames) do
        flame.y = flame.y + (flame.speed * dt)
        if flame.y > love.graphics.getHeight() then -- Remove drops when they
pass off the screen
            table.remove(flames, i)
        end
    end

    --[[
    Collision detection
    Since there will be fewer flames on screen than drops we'll loop them
first
    --]]
    for i, flame in ipairs(flames) do
        for j, drop in ipairs(drops) do
            if checkCollision(flame.x, flame.y, flame.img:getWidth(),
flame.img:getHeight(),
              drop.x, drop.y, drop.img:getWidth(), drop.img:getHeight()) then
                table.remove(drops, j)
                table.remove(flames, i)
                player.points = player.points + 1
                extinguishSound:play()
            end
        end

        if checkCollision(flame.x, flame.y, flame.img:getWidth(),
flame.img:getHeight(),
          player.x, player.y, player.img:getWidth(), player.img:getHeight())
          and player.alive then
            table.remove(flames, i)
            player.alive = false
            gameoverSound:play()
        end
    end

    if not player.alive and love.keyboard.isDown('r') then
        -- remove all drops and flames
        drops = {}
        flames = {}

        -- reset timers
        drop.intervalTimer = 0
        flame.intervalTimer = 0

        -- move player back to default position
        player.x = 175
        player.y = 500

        -- reset the score
        player.points = 0

        -- reset our game state
        player.alive = true
    end
end


--[[
Called very often by the love engine
--]]
function love.draw()
    if player.alive then
        for i, drop in ipairs(drops) do
            love.graphics.draw(drop.img, drop.x, drop.y)
        end

        for i, flame in ipairs(flames) do
            love.graphics.draw(flame.img, flame.x, flame.y)
        end

        love.graphics.draw(player.img, player.x, player.y) -- draw it towards
at the position (x, y)
    else
        love.graphics.print("Press 'r' to restart",
          love.graphics:getWidth()/2-50, love.graphics:getHeight()/2-10)
    end
    love.graphics.setColor(255, 255, 255)
    love.graphics.print("SCORE: " .. tostring(player.points), 320, 10)
end
```