Making games in LÖVE: a fantastic 2D games framework

Clarissa Littler

In this issue we'll be covering a small games-writing framework that I've quickly become fond of: LÖVE.

LÖVE is a small framework for 2D games closer to something like PyGame than to something like Unity or Gamemaker. On one hand, that means there's more that you need to code in order to make a game but it *also* means you have a ton of freedom and get to learn a lot more about what goes into games programming.

LÖVE is built around the language *Lua*. If you've been reading this magazine for awhile you might recognize Lua from past issues, particularly as the language that most *fantasy computers*—all-in-one retro game making & playing tools such as PICO-8, TIC-80, LIKO-12, or Pixel Vision 8—are built off of.

Lua feels a lot like Python but, in my opinion, is a little simpler and easier to use. One big reason is that it reduces a lot of different ways of dealing with data—lists, arrays, maps, dictionaries, objects, stacks, queues—to just one: the *table*. I think this means that there's just less you have to learn and remember in order to feel competent in Lua.

There was a past article on Lua available here (https://www.kidscodecs.com/lua-games/) and there's also a lot of excellent documentation on the main Lua site (https://www.lua.org/docs.html).

Now, about LÖVE in particular. It has a few important concepts that we should go over. First, every LÖVE game is built around three functions

- `love.load()`
- `love.update(dt)`
- `love.draw()`

`love.load` runs at the very beginning of the program, so you can load up all the images, music, etc. and define all the variables and setup. `love.draw` is what it sounds like and is the function that draws everything to the screen once per frame. `love.update` on the other hand is the only function that takes an argument, which is the amount of *time* that has passed since the last frame. You need this information in order to make the gameplay and physics consistent no matter how fast or slow the framerate, or the number of times the screen is redrawn per second, ends up being.

Now, since LÖVE has excellent documentation and *more than one* free online book teaching you how to write games for it, I'm going to jump to a complete example that shows off a feature I really like about LÖVE: the ability to use *shader code* in your game seamlessly.

If you want a refresher on shader, check out the article from earlier this year (https://www.kidscodecs.com/shadertoy-webgl/) but the tl;dr is that you can write special code in the GLSL language in order to harness your graphics card & create visual effects for your game.

Here we're going to show a complete example of a scene in LÖVE where the player has a light

around them, illuminating the scene within a radius.


```
function love.load()
   -- shader code in
   effect = love.graphics.newShader [[
        extern number playerX;
        extern number playerY;
        vec4 effect(vec4 color, Image texture, vec2 texture_coords, vec2 pixel_coords)
        {
        vec4 pixel = Texel(texture, texture_coords );//This is the current pixel color
        number dist = (pixel_coords.x - playerX)*(pixel_coords.x - playerX) + (pixel_coords.y -
playerY)*(pixel_coords.y - playerY);
        number maxdist = 50000;
        if( dist < maxdist) {
        pixel.r = pixel.r * (1 - dist / maxdist);
        pixel.g = pixel.g * (1 - dist / maxdist);
        pixel.b = pixel.b * (1 - dist / maxdist);
        return pixel;
        }
```

```
        else {
        return vec4(0,0,0,1);
        }
        }
    ]]

    background = love.graphics.newImage("topwall.png")
    player = {x = 300, y = 300}
end

function love.keypressed(key)
    if key == "w" then
        player.y = player.y - 5
    end
    if key == "s" then
        player.y = player.y + 5
    end
    if key == "a" then
        player.x = player.x - 5
    end
    if key == "d" then
        player.x = player.x + 5
    end

end

function love.update(dt)
    effect:send("playerX",player.x)
    effect:send("playerY",player.y)
end


function love.draw()
    --draw background and other objects
    love.graphics.setShader(effect)
    -- draw background
    love.graphics.draw(background,0,0)
    -- draw rectangles
    love.graphics.setColor(1,1,1,1)
    love.graphics.rectangle("fill",50,50,50,50)
    love.graphics.rectangle("fill",200,200,50,50)
    love.graphics.rectangle("fill",600,50,50,50)
    -- end shader
    love.graphics.setShader()
```

```
    -- draw player)
    love.graphics.setColor(1,1,1,1)
    love.graphics.ellipse("fill",player.x,player.y,40,40)
end
```

Installing and running LÖVE is pretty simple. The main website (https://love2d.org/) has packages you can download and install. Running it can be slightly finicky depending on your operating system, so check the instructions here (https://love2d.org/wiki/Getting_Started) first!

There are a number of other great features in LÖVE that we didn't even have a chance to discuss including

- built-in bindings to a physics engine if you want to quickly add in realistic collisions, gravity, and acceleration
- touchscreen support
- a simple camera interface that lets you shift the perspective and scale of what the player sees easily
- **tons** of libraries that are made to expand and enhance LÖVE all available for free

So if you want to dive into some surprisingly sophisticated-yet-easy games making, try out LÖVE and get hacking!

Further reading

- https://love2d.org/wiki/Main_Page
- http://blogs.love2d.org/content/beginners-guide-shaders
- https://sheepolution.com/learn/book/contents