Helium is a highly extensible retained and fast GUI library for LOVE2D

If you've ever used React.js, you'll feel right at home, don't dread if you haven't, it's fairly simple.

Getting started

Download helium_0.1.0.zip, unpack it in to your love project and use

```
local helium = require 'helium'
```

to load it in to your project

That's all the setup required

Basic concepts

Element

Is the building block of all GUI's using Helium, an element consists of a single function which you pass in to helium.element(). You can treat everything inside this function as its own love.draw() context, let's create an element right now in fact:

```
function test_element()
    love.graphics.print("Hello world")

end

local hello_world = helium.element(test_element)
hello_world:draw({},10,10,100,20)
```

(add more and explain?)

Parameters

Parameters come in useful, when you want to reuse the same 'template' multiple times, that's where function parameters come in

```
function test_element(parameters, width, height)
```

Parameters can be some arbitrary data you might want to pass in to your rendering 'template', for example font, color, text, alignment, as well the library itself passes in width and height of the element let's integrate a few parameters in to the previous example.

let’s create the parameters themselves right now,

```
p = {
    color = {0.9,0.9,0},
    text = "Different text this time"
}
```

now change the :draw call to this

```
hello_world:draw(p,10,10,100,20)
```

and accept the parameters in to the function itself:

```
function test_element(parameters)
```

and use these parameters inside:

```
function test_element(parameters)
    love.graphics.setColor(parameters.color)
    love.graphics.print(parameters.text)
end
```

Now, helium is *smart,* you can change the parameters, and it will only and only render when you change it, what this means is efficiency, you change the parameters when you need to display different data, and helium catches that change and displays it, this is how most modern UI’s work, because there’s no need to render the whole toolbar every 16ms just in case data changes, only when the data driving that toolbar changes. There are some caveats when using parameters, read more about them here...

Of course doing everything with parameters is possible, but it’s *inconvenient* , wouldn’t it be great to encapsulate some state logic in to an element, but hold on a second, if you run x+y once and didn’t change x or y, then you could run it millions of times and it would always be the same, that’s where

State comes in

Say you have a checkbox, slider or radio group(or something much, much much more complicated), having the logic controlling that somewhere in your main.lua or another random place, would make debugging, and just code readability bad. I sort of lied about the function you make being a normal love.draw context, now it’s not all bad, it is a rendering context similar to that, but it has some added functionality for your convenience, the library injects a few additional things in there, one of those is the function useState, in short, useState will return the current state of the variable you might change inside your element or outside along

with a callback(link to something that explains a callback), to change that state, so practically, how does it integrate with our previous example?

Love by itself doesn't have a great callback to showcase this part, so just for illustration we're gonna capture key inputs in to our little text element:

```
function test_element(parameters, width, height)
    local key, setKey = useState('press a key')

    love.graphics.setColor(parameters.color)
    love.graphics.print(key)
end
```

useState has fairly simple syntax, in the parentheses you put the 'default' value of your variable, say 'true' or '10', it can be a table as well. It returns the current value as the first return, 'key' in this example, and the second is a function you can use to set the new value

As you can guess the library keeps track of your value behind the scenes, so that every time your function calls useState, it returns the current value instead of local x = 10 being effectivelly a constant between function runs.

So let's make the key capturing part, and I must reiterate that there will be a much better way to accomplish this in the next section, here we'll just use the love.keypressed callback:

```
function test_element(parameters, width, height)
    local key, setKey = useState('press a key')

    function love.keypressed(k)
        setKey(k)
    end

    love.graphics.setColor(parameters.color)
    love.graphics.print(key)
end
```

As you should be able to see, the text now becomes the last pressed key, for exercise, lets add some more functionality:

```
function test_element(parameters, width, height)
    local key, setKey = useState('press a key')

    function love.keypressed(k)
        if k == 'delete' then
            setKey('')
        else
            setKey(key..k)
        end
    end

    love.graphics.setColor(parameters.color)
    love.graphics.print(key)
end
```

You now have a rudimentary text input element, here's the whole result:

```
--Loading the library
local helium = require 'helium'

--The rendering function itself
function test_element(parameters, width, height)
    local key, setKey = useState('press any key')

    function love.keypressed(k)
        if k == 'delete' then
            setKey('')
        else
            setKey(key..k)
        end
    end

    love.graphics.setColor(parameters.color)
    love.graphics.print(key)
end

--Creating an instance of our element
local hello_world = helium.element(test_element)

--Setting up parameters for our element
p = {
    color = {0.9,0.9,0},
    text = "Different text this time"
}

--Rendering our element
hello_world:draw(p,10,10,100,20)
```

If you're familiar with react, everything should be relatively similar to what you know, this is where Helium diverges from React signifficantly, since Helium also has to handle Mouse and Keyboard events on the elements (finish this when most input events are done)