
Moonpie

Aug 31, 2020

Contents:

1	Getting Started	3
2	moonpie	5
2.1	moonpie.class	5
2.2	moonpie.event_system	5
3	moonpie.collections	7
3.1	moonpie.collections.iterators	7
4	moonpie.ecs	9
5	moonpie.events	11
5.1	Available Events	11
6	moonpie.graphics	13
6.1	moonpie.graphics.image	13
7	moonpie.math	15
8	moonpie.sorts	17
9	moonpie.test_helpers	19
9.1	Array Extensions	19
10	moonpie.ui	21
10.1	moonpie.ui.styles	21
10.2	Default Styles	21
11	moonpie.utility	23
11.1	moonpie.tables	23
12	Indices and tables	25

Moonpie is a framework designed for easy development within the Love2D engine. It is focused around providing great UX with responsive designs for flexible layouts of components. At the same time it allows for flexibility in how components are rendered. There is also support with libraries for collections, math, entity-component-systems, etc...

CHAPTER 1

Getting Started

Moonpie-template is the easiest way to start a project using Moonpie.

```
$ There is a command that will clone this and set up project
```


CHAPTER 2

moonpie

This is the entry point into the API and provides access to the rest of the framework. This eliminates the need to require specific modules for operation of the framework.

```
moonpie = require "moonpie"
```

2.1 moonpie.class

This sets up a metatable for a table that provides basic OOP functionality. It's not meant to be an overly robust implementation as much as a lightweight overlay.

```
local animal = moonpie.class({})
function animal:constructor(name)
    self.name = name
end

local duck = animal:new("duck")
print(duck.name)
```

2.2 moonpie.event_system

A simple mechanism for creating loosely coupled components that can dispatch events. Events are registered onto the system to be handled. Subscribers can register to specific events. Those events can then be dispatched to with additional arguments as necessary.

Event messages are formatted the same as actions to the state management store. This keeps things consistent but allows for different purposes for dispatching.

```
local my_callback_function = function(data)
    print(data.payload)
```

(continues on next page)

(continued from previous page)

```
end

local events = require "moonpie.event_system"
events.register("hello")
events.subscribe("hello", my_callback_function)
events.dispatch( { type = "hello", payload = "extra_data" })
```

moonpie.collections

Various collections and structures to help provide functionality to lua tables.

```
local a_list = moonpie.collections.list:new()
a_list:add("stuff", 1, 4, "more-stuff")
if a_list:contains(4) then
    print "It does!"
end
```

3.1 moonpie.collections.iterators

Iterators provide a variety of functions for iterating over tables. They should work with any index based table.

3.1.1 moonpie.collections.iterators.cycle

The cycle iterator allows continuous looping over an array. It also provides the ability to move backwards through the list.

moonpie.collections.iterators.cycle(array_table, count) array_table = any table with an index list count = the limit of cycles to perform

```
local set = { 1, 2, 3, 4 }

for value, index in moonpie.collections.iterators.cycle(set, 2) do
    print(value)
end

--
-- Output:
--
-- 1
-- 2
```

(continues on next page)

(continued from previous page)

```
-- 3
-- 4
-- 1
-- 2
-- 3
-- 4
```

```
local set = { 1, 2, 3, 4 }
local cycle_iter = moonpie.collections.iterators.cycle(set)
print(cycle_iter.previous()) -- Output: 4
print(cycle_iter.previous()) -- Output: 3
```

CHAPTER 4

moonpie.ecs

The Moonpie Entity Component System is a lightweight approach to handling entities for processing.

```
local moonpie = require "moonpie"  
local world = moonpie.ecs.world:new()  
  
world:add_systems(...)  
world:add_entities(...)  
world:update("some_method")
```


Events are triggered at various times during interactions with Love2d. These are designed to provide a way of looping in components and behaviors to certain timings without relying on a difficult to maintain call sequence.

Love2D is the ultimate trigger source for events and these do need to be passed manually into Moonpie. This allows the most control possible for engineering solutions while keeping code upstream clean.

5.1 Available Events

```
local function my_callback ()
    print ("called")
end

moonpie.events.before_update:add(my_callback)

function love.update()
    moonpie.update()
end

-- Output
called
```


6.1 moonpie.graphics.image

Provides functionality to access and manage images in a way that limits creating duplicate copies

moonpie.graphics.image.load(path)

CHAPTER 7

moonpie.math

A basic library for some math functions.

CHAPTER 8

moonpie.sorts

moonpie.test_helpers

Test helpers and extra assertions are provided for the Busted <<http://olivinelabs.com/busted/>> testing framework.

9.1 Array Extensions

A number of array helpers are available:

```
it("has some array elements", function()
  local test = { 1, 2, 3, 4 }
  assert.array_includes(1, test)
end)
```

9.1.1 Mock Store

Mocks the redux style store that manages state. Allowing easier testing of components that are dependent on the store.

```
describe("My test harness", function()
  local mock_store = require "moonpie.test_helpers.mock_store"
  local initial_state = { values = true }
  local store = mock_store(initial_state)

  it("can track dispatches", function()
    system_under_test.do_thing_that_dispatches()
    assert.equals(1, #store.get_actions("action_type"))
  end)
end)
```

9.1.2 General helpers

spy_to_func Converts a spy routine into a pure function. This can be helpful in situations where the code under test responds differently to tables vs functions.

10.1 moonpie.ui.styles

Styles are a way of setting common properties that are easy to change across the site. These work similar to CSS in HTML though without the full selector behavior. Styles are applied directly to an element. When calculating values some properties do inherit from the parent to make it easier to specify items like fonts to be defaulted through.

10.2 Default Styles

10.2.1 Buttons

button-small Makes a smaller button for those tinier button needs

button-primary A style that uses the primary color for the background of the button

button-warning A style that uses a gold/yellow background color

button-danger A style that uses a red/fuschia background color

11.1 moonpie.tables

The tables utilities provides various mini-functions for helpful operations.

CHAPTER 12

Indices and tables

- `genindex`
- `modindex`
- `search`