# A LOVE2D Image Editor

Luc V. Talatinian
Department of Electrical and Computer Engineering
University of Dayton
Dayton, Ohio 45469–0232 USA
ltalatinian1@udayton.edu

## ABSTRACT

An image-editing program is implemented in the Lua programming language, using the LOVE2D game programming engine. The system is implemented using the various callbacks provided by the LOVE2D engine that are intended to simulate a main-game loop.

## KEYWORDS

Callbacks, canvas rendering, image, LOVE2D, Lua, main game loop, metaprogramming.

Luc V. Talatinian. 2017. A LOVE2D Image Editor. *CPS 499-03: Emerging Languages, University of Dayton, Ohio 45469–0232 USA, Spring 2017,* 3 pages.

## 1 INTRODUCTION

Lua is a versatile programming language, written in a portable subset of ANSI C, that allows for a wide variety of programming through various paradigms of programming [1, 2]. This is accomplished using the core feature of the language: the table, which is a record of key-value pairs. A table is heterogeneous in both key and value—indexes into a table can be anything outside of `nil`—the Lua equivalent of `NULL`. Using this abstract data type, a wide variety of programming paradigms can be simulated (e.g., object-oriented or functional). A popular use of Lua is for scripting games. The LOVE2D engine, a popular program that uses Lua, allows a user to use Lua scripting to develop a main game loop through a series of callbacks indexed into a parent Lua table. This paper discusses an alternative use of these callbacks: an image-editing program is developed in the LOVE2D engine. By providing various functionality to the LOVE2D engine for input events, such as mouse movement or keypress, the LOVE2D engine can be used to develop an interactive image-editing system.

## 2 CORE LUA

We describe three core features in Lua: tables, metatables, and C-to-Lua interaction.

### 2.1 Tables

The core data structure of Lua is the table. A table in Lua represents an index of key-value pairs. The flexibility of types in Lua—all values are dynamically typed, and all values are first-class—is reflected in its implementation of the table: a table can be indexed by any key type outside of `nil`, and the value for any key is unrestricted. The following Lua code demonstrates the flexibility of table indexing.

```
tab = {}
tab[1] = 120
tab.string1 = function() print('foo') end
tab['string2'] = {}
```

In this example, a table is populated using a variety of different key and value types. This allows for a wide range of meta-programming by using tables to implement a variety of language features. For example, one might choose to build an object prototyping table that generates and returns objects with key-value pairs representing data and functions.

### 2.2 Metatables

A user can define various table operations in Lua using a *metatable*, or a table that defines programmatic interactions between tables. Metatables in Lua are comparable to overloaded operators in C++, in the sense that a programmer is allowed to define standard operations for user-created data. For instance, in a case where a user employs a table to represent hours, minutes, and seconds in a given day, functionality may be supported through a metatable to define various mathematical operations between different times.

### 2.3 The C API

The core implementation of Lua is written in C. A popular use of Lua is as an extension language: Lua scripts can be loaded and run in C/C++ programs using the provided API. A special data type in Lua, `userdata`, can be exposed to a Lua script using this API. This exposure allows a developer to give the ability to extend or control functionality in the program to a user: the user can write scripts that access the data in an underlying C program, modify the data in a way allowed by the developer, or use it for some other purpose.

## 3 THE LOVE2D ENGINE

The LOVE2D engine, the core of which is written in C++, is a popular game-development engine that makes use of Lua for scripting. The engine uses the C API to define a core table from which all functionality in the engine is extended. This core table is accessed by the game developer, who populates it with a variety of functions indexed with special names that the engine recognizes. These callbacks represent a variety of processes that take place during what is known in game development as the *main game loop*. In the classical conception of an interactive game, an infinite loop is run that reads user input, updates the game state, and then renders the

**Table 1: Callbacks in the LOVE2D engine.**

| Name | Description |
|---|---|
| `love.load()` | Called once on startup. |
| `love.update()` | Called every new frame: update state. |
| `love.draw()` | Called every new frame: draw state. |
| `love.mousepressed()` | Fired on mouse button up. |
| `love.mousereleased()` | Fired on mouse button down. |
| `love.keypressed()` | Fired when a key depressed. |
| `love.keyreleased()` | Fired when a key releases. |
| `love.focus()` | Called on focus gain or loss. |
| `love.quit()` | Called once on window close. |

game to the user. In the LOVE2D engine, the programmer defines these components of the main game loop in separate functions indexed into the root table of the engine. For example, to supply rendering functionality, a user defines drawing operations as part of the `love.draw()` callback. This function is then passed back to the core C++ program and executed on every iteration of the game loop. A table of key callbacks in the LOVE2D engine is provided in Table 1. By defining functionality in these various callbacks, the LOVE2D engine allows the programmer to define all aspects of a fully-functioning game.

## 4 IMAGE EDITING IN LOVE2D

The primary use of LOVE2D is for game development. However, the wide host of callbacks provided by the engine (see Table 1) allow for any interactive program to be built with LOVE2D. An example of an interactive program is an image editor. In classical image editing, a program allows a user to load an image into an interactive window. The image is decoded from a variety of formats into an array of RGB pixels, which are then displayed. The user can then make a variety of modifications to the pixels of the image through various defined operations, such as blur/sharpening, pasting, or drawing. The system can then save any changes made by re-encoding the array of pixels into any recognized binary file format. By using the callbacks provided by the LOVE2D engine, it is possible to implement an interactive image editor in LOVE2D by loading an image and allowing changes to it based on mouse and keyboard input. The following subsections detail the implementation of this editor as well as the LOVE2D components that drive it.

### 4.1 Interface

The interface of the editor consists of the main window and the console. The main window displays the image currently being edited by the user. The console is used for debugging and logging events handled by the program. A subset of tools commonly present in image editors are accessed by pressing various keyboard keys (see § 4.4 for the specifics of input). The user guides the mouse along the window, using mouse and keyboard input based on the selected tool to manipulate the image. For example, the user can select the draw tool using the specified key (the list of commands can be printed to the console by pressing '`.`'). With the pen selected, the user can drag the mouse across the screen with the left mouse button held down to draw across the image.

### 4.2 The `ImageData` Object

The storing of the pixel array for an image being edited in LOVE2D can be accomplished through the use of the native LOVE2D `ImageData` object. Recall that all Lua functionality is exposed through tables: the `ImageData` is not an object in the object-oriented sense, but rather, a table with certain data and indexed routines that act as member functions. The `ImageData` object in LOVE2D represents an image (i.e., array of RGBA pixels) stored in memory. The `ImageData` object can be drawn to the screen on every frame using the LOVE2D graphics package. Updates made on any frame to the image are saved to the `ImageData` in memory, and its contents can be encoded into an image file should the user desire to save any modifications made.

### 4.3 The Canvas Layer

Certain drawing operations for the image editor require the use of an intermediate framebuffer. The `ImageData` object cannot be the target of direct draw operations, such as `love.graphics.line()`, which draws a line from one point to another in a framebuffer. Therefore, any draw operations made by the editor are written to a temporary canvas layer using a LOVE2D `Canvas` object. Once the canvas has been drawn to, the pixel data on the canvas is alpha-blended onto the `ImageData` layer, and the canvas is cleared.

### 4.4 Interactivity

User interactivity in the LOVE2D engine is accomplished through the mouse and keyboard handling callbacks. The engine listens for input from both the mouse and keyboard from the user. For instance, the currently selected tool is updated using the `love.keypressed()` callback every time a key corresponding to an option is pressed. In every call of `love.update()`, the program examines the state of mouse and keyboard input and make changes to the image based on the selected tool. The `love.draw()` callback re-renders the current editing `ImageData` to the LOVE2D window every time a frame is processed by the engine.

### 4.5 Encoding and Decoding

On initialization, the editor loads the file `infile.png` from the root directory of the program. The user can save the image stored in the `ImageData` buffer at any time by pressing the corresponding hotkey. Additionally, changes made are saved when the editor is closed. Output files are written to `outfile.png` in the save directory for the program, which is OS-specific. For instance, LOVE2D on Windows stores `savedata` for each of its programs in the `appdata` folder of a user's home directory).

## 5 CONCLUSION

Lua is a versatile scripting language that allows for a wide range of metaprogramming through the use of its built-in table data structure. Scripts written in Lua can be run on top of a C program, allowing a programmer to extend or control its functionality. An example of such a program is the LOVE2D engine, which allows a user to construct a main game loop by writing various callbacks in Lua. Since callbacks can technically be used to implement any interactive program, it is also possible to develop an image editor in LOVE2D. We achieved this by keeping the pixel data for an image in a LOVE2D `ImageData` object, updating changes to the `ImageData` based on user input, and drawing the pixels to the screen.

## REFERENCES

[1] R. Ierusalimschy. 2003. *Programming in Lua.* Lua.org.

[2] R. Ierusalimschy, L. Henrique de Figueiredo, and W. Celes. 2017. *Lua 5.3 Reference Manual.*