Fugue Language for ComputerCraft Mod
Cyra Uthoff
CSC402 Programming Language Implementation

Introduction

In my free time, I like playing an extremely popular video-game named Minecraft. Created in 2009, Minecraft is the best selling video game of all time. The magic is in its sandbox gameplay, allowing players to build whatever they can dream of in a world of blocks. The possibilities are seemingly endless, most especially when the game is paired with the many mods created by the community. One of these mods is the ComputerCraft mod, which lives on today in the form of the "CC: Tweaked" fork.

This mod brings programmable virtual computers into the game, giving players the power of programming to interact with the world around them. These virtual computers run on the "CraftOS" shell-based 'operating system', with limits on terminal size, colors, etc. In order to run effectively in the game, it uses the lightweight, embeddable Lua programming language.

However, with my use of the mod in my own time, I find that 1) the Lua language's footprint on its keywords are large considering the limited default terminal size and 2) requires extra code to perform common tasks that I find the in-game computers useful for (particularly when used with other compatible mods, such as Create). The language I propose, the Lua-based interpreted language "Fugue", aims to provide solutions for that.

Description & Examples

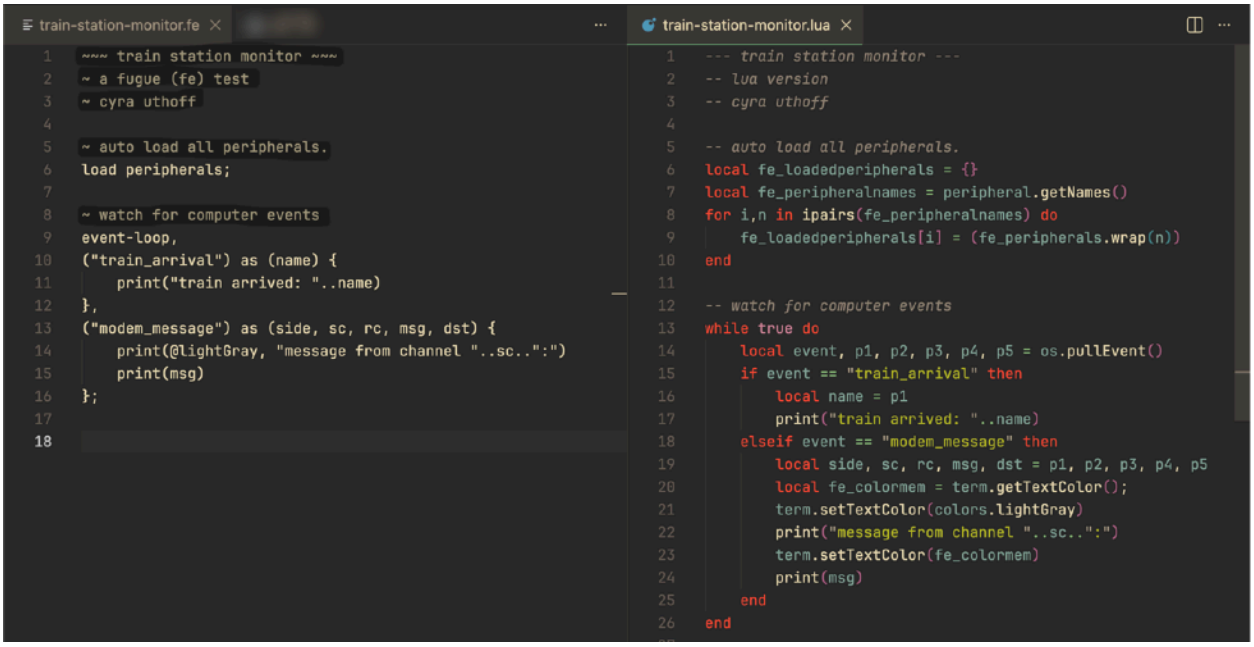
Because a major goal of this language is to have a smaller footprint than typical Lua programs, it becomes necessary to both use smaller keywords and to implement shorthands for commonly used actions, commands, and conditions.

One of these is the loading of peripherals, which can be loaded automatically using the "`load peripherals;`" command. As seen in the example below, this removes a large chunk of code that can become very repetitive between different programs.

Another major aspect of programming for ComputerCraft is the built-in event system. The core computer and the peripherals attached can emit 'events' in reaction to actions or changes made on the system or in the world around it. In the example below, I show a way to create that loop in fewer space. This has the necessary impact of giving code responding to these events only a single indentation on the side rather than two, preserving horizontal space.

A final example of space-saving is in the recreation of the "`print()`" function, which can take colors as one of its arguments. Each color is preceded by the "`@`" symbol, and the preceding color is reapplied when the function is finished.

Above is the discussed example code in both Fugue (.fe) and Lua (.lua). Comments in the Fugue code are grayed out for distinguishability.



```
train-station-monitor.fe
```

```
~~~ train station monitor ~~~
~ a fugue (fe) test
~ cyra uthoff

~ auto load all peripherals.
load peripherals;

~ watch for computer events
event-loop,
("train_arrival") as (name) {
    print("train arrived: "..name)
},
("modem_message") as (side, sc, rc, msg, dst) {
    print(@lightGray, "message from channel "..sc..":")
    print(msg)
};
```

```
train-station-monitor.lua
```

```lua
--- train station monitor ---
-- lua version
-- cyra uthoff

-- auto load all peripherals.
local fe_loadedperipherals = {}
local fe_peripheralnames = peripheral.getNames()
for i,n in ipairs(fe_peripheralnames) do
    fe_loadedperipherals[i] = (fe_peripherals.wrap(n))
end

-- watch for computer events
while true do
    local event, p1, p2, p3, p4, p5 = os.pullEvent()
    if event == "train_arrival" then
        local name = p1
        print("train arrived: "..name)
    elseif event == "modem_message" then
        local side, sc, rc, msg, dst = p1, p2, p3, p4, p5
        local fe_colormem = term.getTextColor();
        term.setTextColor(colors.lightGray)
        print("message from channel "..sc..":")
        term.setTextColor(fe_colormem)
        print(msg)
    end
end
```

There are other elements at play with Fugue, which both distinguishes itself from and likens itself to Lua. Instead of using "`then, end`" or "`do, end`" pairs, simple curly brackets (`{}`) are used to begin and end code blocks. Concatenation between strings is similarly performed with two period marks (`..`).

Because, for the sake of this project, the code is necessary to run on a system in which the game Minecraft is not installed, and because setting up the modding environment could result in complications and overall not practical: this project proposal posits that creating a website-accessible version of the code would be best. Existing web-based emulators for ComputerCraft already exist (see 'Resources Needed'), and allow for embedding custom content, which would in this case be Fugue's interpreter.

Proposed Timeline & Milestones

With four weeks until the project deadline, my proposal for the project timeline is as follows…

1. Create a web-based environment for testing and running the Fugue interpreter. Create the base framework for the interpreter to run in.
2. Document a solidified grammar system for the final language. Ensure compatibility with the Lua patterns library as a stand-in for RegEx.
3. Create the functions necessary to run within the interpreter.
4. Polish the final product with example code and documentation.

Resources Needed

The CC:Tweaked Documentation: https://tweaked.cc/
SquidDev-CC's Web Emulator: https://github.com/SquidDev-CC/copy-cat
Lua Patterns Documentation: https://www.lua.org/pil/20.2.html
GitHub Pages to Host Site: https://docs.github.com/en/pages