Fugue Language for ComputerCraft Mod

Cyra Uthoff

CSC402 Programming Language Implementation

Introduction

In my free time, I spend a lot of time playing the well-known sandbox game called Minecraft, alongside a suite of different mods that brought new experiences and types of things to build. As a Computer Science student, naturally one of the mods I've used is the ComputerCraft mod, which has existed in iterations since 2011, and that exists today through the CC:Tweaked fork.

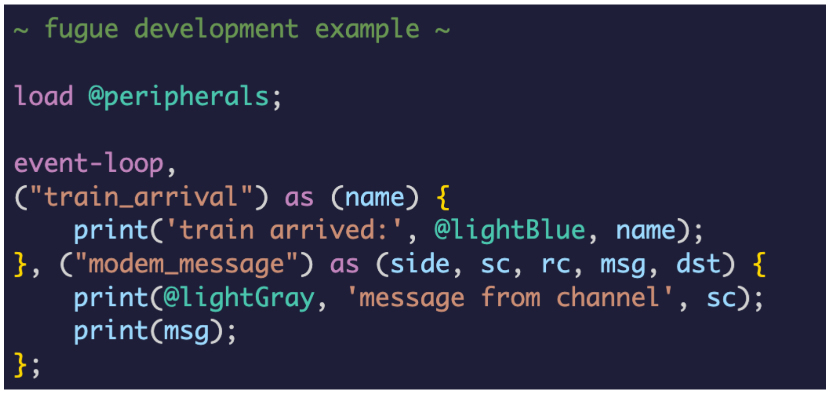
The mod, at its core, allows players to interact with the sandbox world through code, via computers, turtles, and 'iPad'-like pocket computers. Everything runs on the lightweight, embeddable Lua programming language. In high school, I toyed a lot with 'fantasy consoles' such as PICO-8 or TIC-80, which also ran on Lua. In this way, ComputerCraft feels like an in-game fantasy console of its own, with a built in shell, programs, modules, and APIs.

However, with my use of the mod in my own time, I've found that 1) Lua's keywords feel large considering the limited default terminal size and 2) requires extra code to be written just to perform common tasks (and especially so when used with other compatible mods, such as Create).

Implementation

As a response to these issues, I've created an interpreted programming language named 'Fugue'. It features shorthands and built-in commands for often performed tasks such as loading all of the peripherals, and also has a smaller event-loop system to replace the '`os.pullEvent()`' and '`while true`' loop found throughout the CC:Tweaks documentation.

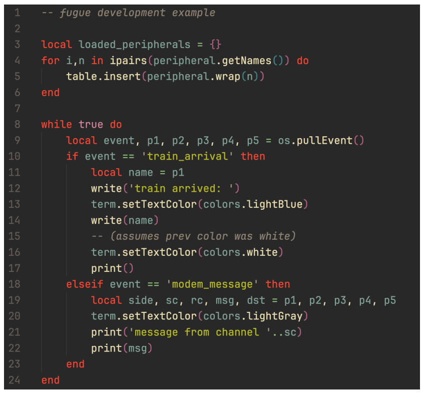
In the code below, you can see an example of a program watching for train arrivals and messages over the modem, in both Fugue (left) and Lua (right). Fugue definitely exceeds in both readability and overall footprint.



```
~ fugue development example ~

load @peripherals;

event-loop,
("train_arrival") as (name) {
    print('train arrived:', @lightBlue, name);
}, ("modem_message") as (side, sc, rc, msg, dst) {
    print(@lightGray, 'message from channel', sc);
    print(msg);
};
```



```
-- fugue development example

local loaded_peripherals = {}
for i,n in ipairs(peripheral.getNames()) do
    table.insert(peripheral.wrap(n))
end

while true do
    local event, p1, p2, p3, p4, p5 = os.pullEvent()
    if event == 'train_arrival' then
        local name = p1
        write('train arrived: ')
        term.setTextColor(colors.lightBlue)
        write(name)
        -- (assumes prev color was white)
        term.setTextColor(colors.white)
        print()
    elseif event == 'modem_message' then
        local side, sc, rc, msg, dst = p1, p2, p3, p4, p5
        term.setTextColor(colors.lightGray)
        print('message from channel '..sc)
        print(msg)
    end
end
```

My favorite thing to have implemented is the "special" variables, which you can see preceded with the '`@`' symbol and in teal. These variables are, at their core, just simple values like strings or numbers, but come bundled with a "kind" attribute that tells functions what to do with it. As an example, the built in '`print`' function, when faced with a special variable of the kind "color", will use that value to change the color of the terminal. Inline, this is much smaller than changing the terminal color on its own line, but it also opens the door to new types of programming with these special variables.

Challenges

The most challenging aspect of this project was ensuring that I could make the interpreter run on Lua. I wanted to make sure that this interpreter could run in the actual game, without needing the auxiliary step of compiling Fugue code into Lua using Python like we have everywhere else in the course. To this end, I spent the entire first week of development rewriting the Stack interpreter we created earlier this semester in Lua, creating my own library of functions (see '`fugue_lib.lua`') to do common things not found in the Lua engine.

Actually, of things *not* found in Lua, RegEx is one of them, which I worried would be a massive roadblock to implementation. Luckily, Lua's own patterns feature (seems to) handle everything I've thrown at it just fine, once I had gotten the hang of it.

If I had more time, I would have definitely added more things—anything—to flesh out the language a little more. As of now, it stands more as a proof of concept rather than something that would actually be useful yet. I think adding things like '`while`' or '`for`' loops, '`elif`' and '`else`' statements, structured data types such as lists, '`and`' and '`or`' operators, a (more functional) error system, or ways to load Fugue code from other files would go a long way to make the Fugue feel like a complete language. And in my free time, I might do just that, but for the purposes of this assignment, this is where I've left it.

Examples

For my examples, I made a GitHub pages website which runs an emulated version of the CraftOS shell you see in-game: HTTPS://UTHOFFCYRA.GITHUB.IO/FUGUE/. There, I give three buttons for examples of things you can do with Fugue. Clicking any of them will run that script and show a syntax-highlighted window of that code. Code can also be run manually by writing a program with the '`edit [filename]`' command, and then running it with the '`fe [filename]`' command.

The first example tests the functionality of Fugue's expression system, including the proper order of operations, and the color special variables I mentioned earlier. The second example is a test for user input. It waits for any key to be pressed (make sure you have the terminal window focussed for this), and prints to the terminal which key was hit, and if it is being held. The third and final example is a test for interacting with the in-game world, in particular with the Create Mod's train system. To this end, I created a small simulation of a train system. It's modeled after one I have in my game, where the Orange Line and Bay Line cross over at one station. The program prints a colored announcement whenever a train arrives.

All of the code for this, both the website (`/docs`) and the interpreter (`/src`), exists on a public GitHub repo. Example images are included at the end of this report.

Also, for testing in-game, I created a GitHub Action that compresses all of the code in the repo into a real datapack you can import. I have one posted as a Release right now.

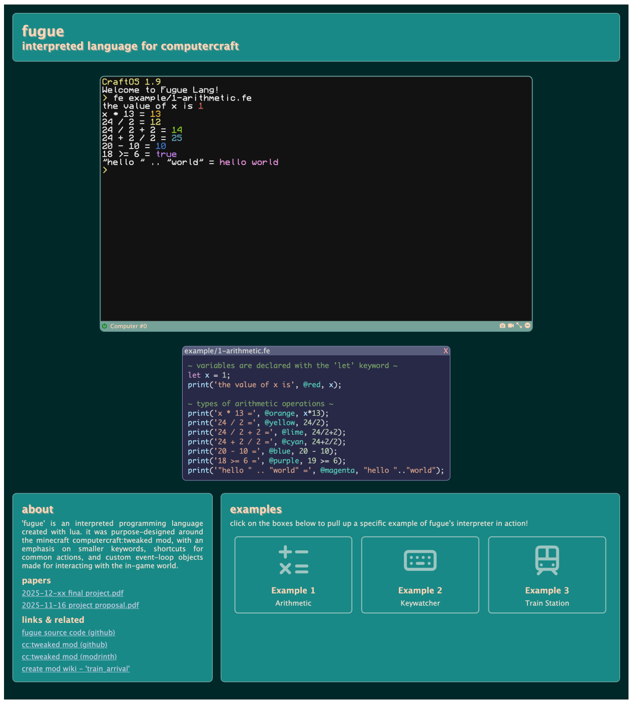
Conclusions

To conclude on this project, I'd like to state that I definitely learned a lot about programming language implementation. I was able to apply all of the things learned from the course, now in a new environment, in both the programming language used, and in new application.

I'd love to work on this further, which is why I've fleshed out the GitHub environment as much as I have. While this *is* an assignment for University, it equally felt much like a passion project.

Images

EXAMPLE 1 - ARITHMETIC



fugue
interpreted language for computercraft

```
CraftOS 1.9
Welcome to Fugue Lang!
> fe example/1-arithmetic.fe
the value of x is 1
x * 13 = 13
24 / 2 = 12
24 / 2 + 2 = 14
24 + 2 / 2 = 25
20 - 10 = 10
18 >= 6 = true
"hello " .. "world" = hello world
>
```

example/1-arithmetic.fe

```
~ variables are declared with the 'let' keyword ~
let x = 1;
print('the value of x is', @red, x);

~ types of arithmetic operations ~
print('x * 13 =', @orange, x*13);
print('24 / 2 =', @yellow, 24/2);
print('24 / 2 + 2 =', @lime, 24/2+2);
print('24 + 2 / 2 =', @cyan, 24+2/2);
print('20 - 10 =', @blue, 20 - 10);
print('18 >= 6 =', @purple, 19 >= 6);
print('"hello " .. "world" =', @magenta, "hello ".."world");
```

about

'fugue' is an interpreted programming language created with lua. it was purpose-designed around the minecraft computercraft:tweaked mod, with an emphasis on smaller keywords, shortcuts for common actions, and custom event-loop objects made for interacting with the in-game world.

papers

2025-12-xx final project.pdf
2025-11-16 project proposal.pdf

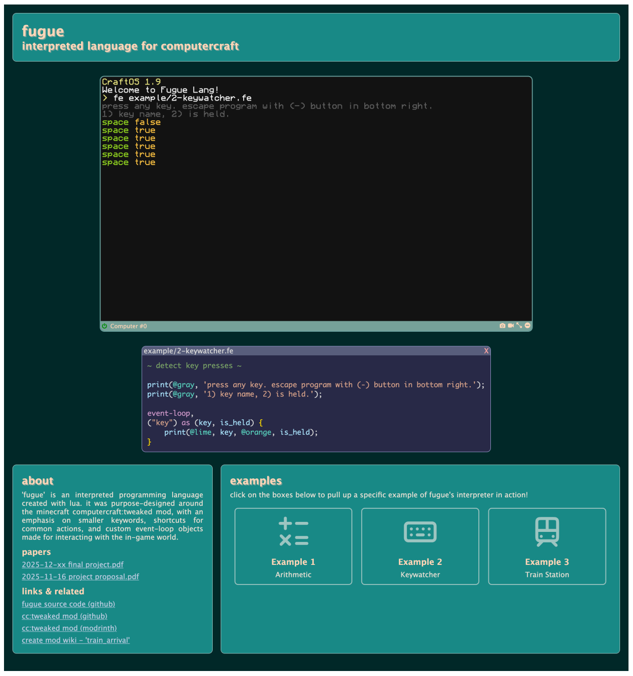
links & related

fugue source code (github)
cc:tweaked mod (github)
cc:tweaked mod (modrinth)
create mod wiki - 'train_arrival'

examples

click on the boxes below to pull up a specific example of fugue's interpreter in action!

EXAMPLE 2 - KEYWATCHER



fugue
interpreted language for computercraft

```
CraftOS 1.9
Welcome to Fugue Lang!
> fe example/2-keywatcher.fe
press any key. escape program with (-) button in bottom right.
1) key name, 2) is held.
space false
space true
space true
space true
space true
space true
```

example/2-keywatcher.fe

```
~ detect key presses ~

print(@gray, 'press any key. escape program with (-) button in bottom right.');
print(@gray, '1) key name, 2) is held.');

event-loop,
("key") as (key, is_held) {
    print(@lime, key, @orange, is_held);
}
```

about

'fugue' is an interpreted programming language created with lua. it was purpose-designed around the minecraft computercraft:tweaked mod, with an emphasis on smaller keywords, shortcuts for common actions, and custom event-loop objects made for interacting with the in-game world.

papers

2025-12-xx final project.pdf
2025-11-16 project proposal.pdf

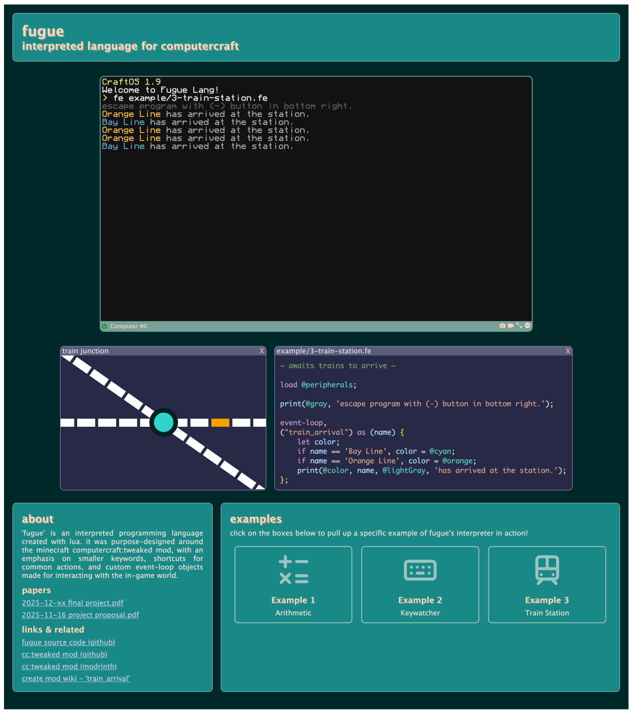
links & related

fugue source code (github)
cc:tweaked mod (github)
cc:tweaked mod (modrinth)
create mod wiki - 'train_arrival'

examples

click on the boxes below to pull up a specific example of fugue's interpreter in action!

EXAMPLE 3 - TRAIN STATION


fugue
interpreted language for computercraft
Computer #0
train junction
X
example/3-train-station.fe
X

```
CraftOS 1.9
Welcome to Fugue Lang!
> fe example/3-train-station.fe
escape program with (-) button in bottom right.
Orange Line has arrived at the station.
Bay Line has arrived at the station.
Orange Line has arrived at the station.
Orange Line has arrived at the station.
Bay Line has arrived at the station.
```

```
~ awaits trains to arrive ~

load @peripherals;

print(@gray, 'escape program with (-) button in bottom right.');

event-loop,
("train_arrival") as (name) {
    let color;
    if name == 'Bay Line', color = @cyan;
    if name == 'Orange Line', color = @orange;
    print(@color, name, @lightGray, 'has arrived at the station.');
};
```

about

'fugue' is an interpreted programming language created with lua. it was purpose-designed around the minecraft computercraft:tweaked mod, with an emphasis on smaller keywords, shortcuts for common actions, and custom event-loop objects made for interacting with the in-game world.

papers

2025-12-xx final project.pdf

2025-11-16 project proposal.pdf

links & related

fugue source code (github)

cc:tweaked mod (github)

cc:tweaked mod (modrinth)

create mod wiki – 'train_arrival'

examples

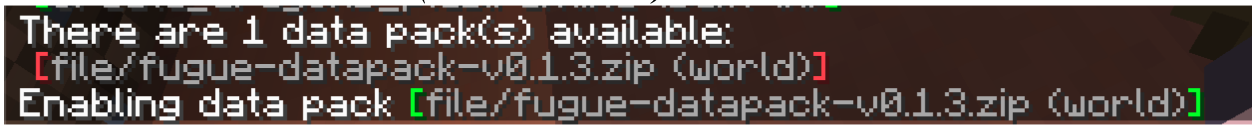
click on the boxes below to pull up a specific example of fugue's interpreter in action!

Example 1 Arithmetic

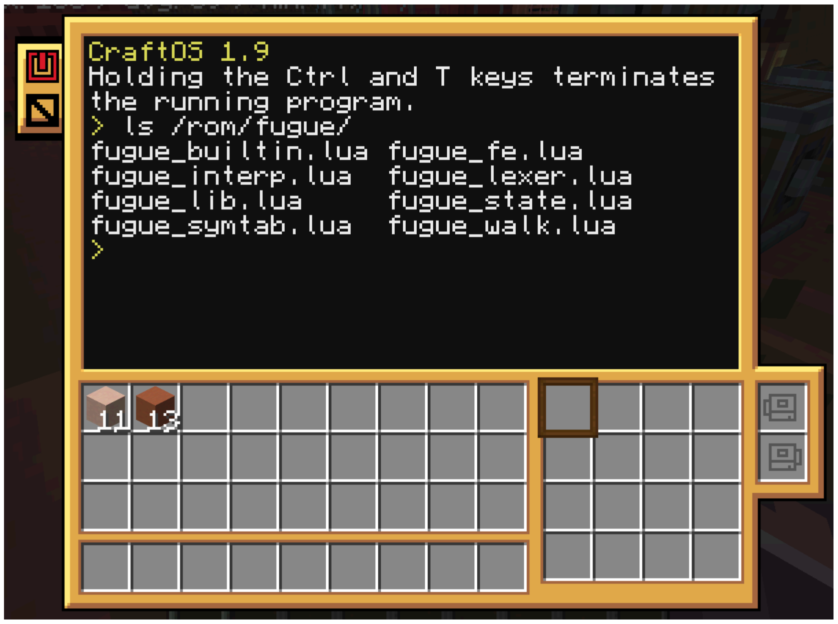
Example 2 Keywatcher

Example 3 Train Station

IN-GAME FUNCTIONALITY (VIA DATAPACK)



```
There are 1 data pack(s) available:
 [file/fugue-datapack-v0.1.3.zip (world)]
Enabling data pack [file/fugue-datapack-v0.1.3.zip (world)]
```



```
CraftOS 1.9
Holding the Ctrl and T keys terminates
the running program.
> ls /rom/fugue/
fugue_builtin.lua  fugue_fe.lua
fugue_interp.lua   fugue_lexer.lua
fugue_lib.lua      fugue_state.lua
fugue_symtab.lua   fugue_walk.lua
>
```