>Remember

By Hugo Labrande
Issue #12 : A printing Z-Machine

Here is Issue #12! Thank you for your support and your continued interest in the last year! I had promised a special article, and this is what you're getting today; it took me longer than usual, and I hope you'll like it!

So far, this newsletter has been talking about retro text adventures, but with a pretty major blind spot: all the games we have talked about so far are for computers with screens, that display the result of your commands. But in the 1970s, you could play text adventures on a terminal and the response to your input was outputted on paper. This month, we're turning the clock back, with a tutorial for a text adventure machine that outputs the game's text on printer paper. But we won't use period-accurate hardware! You will only need:

- a Raspberry Pi;
- a thermal printer with USB connection.

This should cost you about 100 US dollars. I used a Raspberry Pi 3, but any model will do, really. As for thermal printer, here is the one I purchased:
https://www.amazon.ca/Thermal-Receipt-Printing-Compatible-Commands/dp/B06XW8MK6W/
You will also find a thermal printer that is officially supported on Raspberry Pis on the website AdaFruit, but when I was looking at it, it was backordered. And Raspberry Pis are also a bit hard to come by these days! Hopefully by the time you are reading this, the supply chain issues and the computer chip shortages will be a thing of the past!

And before I go on, I want to note that today's tutorial is probably only applicable to users that are not visually impaired; I believe that Braille printers or embossers are not very commonplace, are very expensive, and I'm not sure they could be adapted for these purposes. Apologies to the visually-impaired readers of my newsletter!

Printer setup

This part may be highly variable, and will depend on the make and model of your thermal printer. (Make sure you read the manual!) I am under the impression that the POS58 printer I purchased is rather popular, and the way it works (through CUPS) is rather standard. I hope this section will be useful to a few of you! Also, this tutorial is for Linux, since the Raspberry Pi's OS is based on Linux; however, you probably will be able to make it work on Windows too, especially if you have Windows Subsystem for Linux (WSL) on your Windows.

My printer came with a little roll of thermal printer paper (that you need to insert with the paper coming from below the roll, not above), and a minidisc with drivers for Windows, Linux, Android, and iPhone. I copied the contents of the minidisc onto my hard drive first (let me know if you cannot read the minidisc and I can send you the

drivers). For Linux, the driver installation comes as a ".sh" script, that you need to make executable ("`chmod +x install58.sh`") then run as an administrator. This sets up the CUPS printing utility and registers the printer.

Now, if you open a browser and type "`localhost:631`", you should have access to the CUPS admin console, and you should see the POS58 there. (If not, run the script again). Except there was a problem for me: the path to the printer was broken. You can diagnose this if you go in the printer settings and see "`socket://something`" as the path to the printer. In order to fix this, you need to follow these steps:

1. Plug your printer to your computer's USB port.
2. Type "`lpinfo -v`" in a command line.
3. You should see a path containing "`usb://`": copy that path.
4. Run the command "`lpadmin -p POS58 -E -v usb://Unknown/Printer?serial=Printer -P /usr/share/cups/model/zjiang/POS58.ppd`", replacing the "`usb://`" parameter with the one that you got above.
5. Go back to CUPS, and run a test page.

All in all, if you follow these steps, the setup should be pretty painless. One last thing that you might want to tweak is the feed distance after print: go back to the CUPS admin panel and into your printer's settings, then select the "Device Settings" tab and modify the value there. I found that 6mm was too short, so I selected 9mm, but any larger value will work.

You can now print any file on your thermal printer! To test this, create a simple text file and type something; then open a command line in that folder and type

```
lp myfiletoprint
```

If you already have a printer hooked to that Raspberry Pi, you might need to specify the printer's name (or set the thermal printer as the default printer by typing "`lpoptions -d POS58`").

Using Python to print

We can now try to print files from Python. (If you are not familiar with the Python programming language, don't worry! You can skip ahead and download the final files near the end of this issue!) The mechanism is simple: open a file, write in it, close it, and print. The following code will do:

```
import os
f = open("myfiletoprint", "w")
f.write("This is a test")
f.close()
os.system("lp myfiletoprint")
```

If everything is working well, running this script should print something on your thermal printer. You can also try passing options in that call; however, having no prior experience with CUPS, I didn't find the manual very useful.

There is also a great Python library that allows you to interface with CUPS and set options in there, which I found useful. You can install it via

```
sudo apt install python-cups
```

Then use the following code to print:

```
import os
f = open("myfiletoprint", "w")
```

```
f.write(“This is a test”)
f.close()
import cups
conn = cups.Connection()
printers = conn.getPrinters()
printer_name=printers.keys()[0]
conn.printFile(printer_name, "myfiletoprint", "", {})
```

This should have the same result, but you now are able to specify options in the call to `printFile` – for instance, the number of characters per inches or the number of lines per inches. I found that, on my 58mm printer, you could guess the number of characters per line by doubling the “cpi” value; I landed on the value “19” for the cpi and “11” for the lpi as something readable but without taking too much paper. So you can write instead:

```
conn.printFile(printer_name, "myfiletoprint", "", {"cpi" : "19", "lpi" : "11"})
```

You might also notice that the text that is printed on the thermal printer is not word-wrapped, i.e. the lines are all the same number of characters and splice words as needed. This can be solved using the “textwrap” python module. Let’s add that and recap the code:

```
s = “This is a test”
import textwrap
a = textwrap.fill(s, width=35)
f = open(“myfiletoprint”, “w”)
f.write(a)
f.close()
import cups
conn = cups.Connection()
printers = conn.getPrinters()
printer_name=printers.keys()[0]
conn.printFile(printer_name, "myfiletoprint", "", {"cpi" : "19", "lpi" : "11"})
```

Setting up an interpreter

We now need to insert that code into a Z-Machine interpreter (or any other Python code that plays text adventures, really!). There are a few of them, and I will provide one of them with the suitable modifications already done; but read on if you’d like to see the few steps that need to be done to hook the interpreter onto the printer.

A high-level description of what we need to do is as follows:

1. Build the string that is the response of the game.
2. Output it.

In order to build the string, we will start from an empty string `s = “”`, then add to s any string that would normally be printed by the interpreter. We then need to print it, which we need to do as soon as the game stops to ask for input from the player. Hence, it is crucial that we find two things in the code:

1. The moment where strings are printed on the screen;
2. The moment where the player needs to provide input.

This actually depends on the interpreter and how it was implemented, and there is no other way but to dive in the code and try to understand how it works. You can look in

the code for “print” or “string” to find the answer to 1/, and for “input” or “read” for the answer to 2/.

But actually, finding a Python interpreter with reasonably tidy code that can still be understood is rather hard! Here is what I have found on Github:

- zvm by Ben Collins-Sussman is an interpreter written in a way to provide compatibility with various frontends, but also has an extremely bare bones interface, which works for our purposes. The file “`zvm/trivialzui.py`” is what you need; the place to capture the string that is about to be written is in the `__unbuffered_write` function, and the place to send to the printer is in `_read_line`.
- moosezmachine by Moosepod looks promising. I believe you want to capture the strings in the print_text function in `generic_terp/__init__.py` or `pygame_terp/window.py`; and you send to the printer during the `text_entered` function in `terp.py`.
- mjdarby’s Fic interpreter is very nice, as it is one file of readable Python; you want to capture the strings in `printBufferedString`, and output to the printer should be done during `handleInput`. The readme file indicates that z3 is supported, and that z5 support is partial, but it looks like it mostly works (saving and restoring might be wonky, if I’m reading that right?). Accented characters are not be supported.

The latter interpreter was really the most painless and effortless to modify for our purposes; it really only took 5 minutes. If you’d rather not attempt it yourself, you can download this file instead:

www.hlabrande.fr/fi/hosted/remember/39503654/Fic.zip

You can open the “`fic.py`” file and look for “Hugo” to see my additions; to start the game, type `python3 fic.py mygame.z5`.

If you follow the steps outlined above, you have a pretty interesting machine, one that plays a text adventure with you by printing onto a thermal printer. I suggest the following setup: put the thermal printer on a table, and conceal the cable running it to the Raspberry Pi. Then find someone younger than you, and tell them a mysterious printer wants to play a game with them, and offer to type the commands. The unique setup and the curiosity of seeing the output on paper will produce a great effect! And for an even fuller effect, remove all "print" statements from the interpreter, so it doesn't even feel like "the printer is copying what is on your screen" and more like "my computer is the typing interface". Or get a Bluetooth keyboard, and try to type the commands without any visual feedback!

Going screenless

This is, unfortunately, something that did not come together before the deadline for this article - which is too bad! But it would have been very cool to engineer a system that conceals even more, and doesn’t even look like a computer. Here are a few ideas:

- if your thermal printer has Bluetooth capabilities, you might be able to send the commands from your phone to the Raspberry Pi, which will then send it to the printer. Sending the message as a vCard is possible but annoying; there might be apps to send messages over Bluetooth; or you can repeatedly share a file. (Or you can make your own Android app I suppose…)
- you could set up a web page that is just an input box that sends the result to your Raspberry Pi (for instance your Raspberry Pi could host the page on a HTTP server on a specific port, locally). Once you manage to recover the string

that was sent, pass it to the interpreter; the interface is then just an empty mysterious prompt that triggers the printer.
- you could set up a microphone and some voice recognition on the Raspberry Pi. These are all machine-learning-based (hence great interfacing with Python), and you can either use Google's speech-to-text solution (their API is accessible via Python), or even run your own machine learning solution (a Raspberry Pi 4 is preferable for that).

I know quite a few people who read this newsletter are tinkerers, and I can't wait to see if some of you set this up and maybe even take it further! I hope you enjoyed this cool little DIY project!!