glowfire

version 2025.05

python GUI for pocket computers
interface graphique Python pour les ordinettes
ポケットコンピュータ用のPython GUI

Table o' contents

See page 18, 'Appendix 1 - Experimental Notes' for more technical information.

Description

glowfire: Graphical low fidelity REPL environment.

glowfire is a Python GUI for the SQFMI beepy.
It is a fork of snakeware, optimized for the Memory LCD, Raspberry Pi Zero 2, and other beepy hardware.
It renders directly to the framebuffer. No X, OpenGL, Mesa, or GPU at this time.

Features

- Makes 3 toolchains, from low level to high level: DirectFB > SDL2 > pygame
- Graphics support for Sharp Memory LCD
- Sound support for USB audio adapters plugged into the Raspberry Pi Zero USB port
- Experimental support for bluetooth headphones configured over command line
- Firmware support for the beepy keyboard, and setting touchpad as a mouse input device
- OpenOCD support for in-circuit flash + debug of RP2040 (Requires solder modification to J45 connector)

Requirements

- SQFMI beepy v1.0 (2023)
- Fast MicroSD card (Class 10 U1)
- Micro USB data cable
- Host system

Quick Start begins on next page.

Quick Start

glowfire must be installed on a fresh Raspberry Pi OS.
This provides a known baseline.
If the OS has been customized at all, build scripts may fail to run properly.

1/3: Prepare a microSD card for headless operation

- Download the latest Raspberry Pi Imager. It does not auto-update. https://www.raspberrypi.com/software
- Launch Raspberry Pi Imager.
- Choose OS > Raspi OS (other) > Raspberry OS LITE 64-Bit. Do not install the full Desktop version.
- Click "gear" button lower right; pre-fill SSH, hostname, and wifi info. It is essential to complete each.
- Pre-fill the Username and Password fields. You can use any username you wish.
  For illustration, our user will be named **beepy,** and our hostname will be **beepy1.**
- Don't eject the microSD card yet. It mounts as a disk named **bootfs.**

These instructions are for macOS, but are similar on Linux.
On Linux, install avahi to support mDNS.

Set SSH to auto-start when the Raspberry Pi boots.

```
$ touch /Volumes/bootfs/ssh
```

Configure USB Ethernet Gadget ("Ethernet over USB").

This loads DesignWare USB2 DRD Core support into the device tree, so the Raspberry Pi appears as a USB network device when it is connected to a host computer.

Open the Raspberry Pi OS config.txt file:

```
$ vi /Volumes/bootfs/config.txt
```

Add a new line at the bottom of the file:

```
dtoverlay=dwc2
```

Write and quit.

```
:wq
```

Enable USB Ethernet Gadget.

```
$ vi /Volumes/bootfs/cmdline.txt
```

After rootwait, insert (on the same line):

```
modules-load=dwc2,g_ether
```

Write and quit.

```
:wq
```

Eject card and install into the Raspberry Pi.

2/3: Power up the beepy and connect to it with the USB Ethernet Gadget

Video will not appear on the Sharp Memory LCD until after the console driver is installed.
We will use SSH to work on the Pi throughout the entire installation of the SDK.

Unplug all USB cables from the beepy.

Slide the beepy's power switch to "on" (rightwards).
With a microUSB cable, attach your computer to the Raspberry Pi Zero's "center," USB connector.
Continue trying to ping, until it responds via mDNS. The Pi may take 2-4 minutes to boot the first time.

```
$ ping beepy1.local
```

Monitor for the Ethernet Gadget to come up.

On Mac: > System Settings > Network > watch for "RNDIS/Ethernet Gadget" to become active.

It is normal for the Gadget to have a link-local IP (169.254.x.x).

Copy your SSH key to the beepy.

```
$ ssh-copy-id beepy@beepy1.local
```

Now SSH into the beepy. It should not prompt you for a password.

```
$ ssh beepy@beepy1.local
```

Note: If you ever re-use a hostname, the system will warn you.

If this happens, delete the old key from ~/.ssh/known_hosts:

```
$ ssh-keygen -R  <- followed by the Pi's IP or hostname
```

It's a good idea to use different hostnames (beepy1, beepy2, beepy3, etc.) if you are juggling multiple builds on different microSD cards.

The guide continues from within the SSH session on the beepy.

3/3: Install glowfire

Install git, a dependency to run the build script:

```
$ sudo apt install -y git
```

Clone glowfire, then run its build script:

```
$ cd ~
$ git clone https://github.com/hack-shack/glowfire
$ cd ~/glowfire
$ python make-glowfire.py
```

Installation takes about 90 minutes.

Most of the installation is spent within build-sdk.py, building the toolchain.

Continue: Make a new branch and start developing

```
cd ~/glowfire
git branch my-test-branch <or whatever you choose to name it>
git switch my-test-branch
```

You can always rollback to the official source code by SWITCHing to that branch, e.g.

```
git switch master  <-- switches back to the github version you downloaded
```

FAQ

What is glowfire?

glowfire is a 'distro' for the SQFMI beepy.

It is intended as a way to build graphical apps in pygame.

It provides the beepy with a self-hosted development toolchain, 'beepy-sdk.'

Its graphical user environment is a fork of snakeware, redesigned for the Memory LCD.

The goal is a common software platform for the beepy reference hardware, written in Python.

What hardware is supported?

The SQFMI beepy, released 2023, is the reference platform for the glowfire toolchain. Schematics, firmware, etc. are publicly available for this board.

The beepy is a backplane for the Raspberry Pi Zero, which turns it into a pocket computer.

It was designed as a collaboration between chat app maker Beeper and design/engineering group SQFMI (Squarofumi, "skwair-oh-foo-me") and is open hardware under the CERN Open Hardware Licence 1.2.

An owner's description of the device is at "Messing with the Beepy," by Jeff Geerling.

The beepy PCB is open source. PCB KiCad project is in the SQFMI-beepy-hardware repository.

The beepy schematic is in the SQFMI-beepy-hardware repository.

Why build from scratch, instead of distributing as packages, or a prebuilt ISO?

glowfire may be used in remote environments without internet access.

glowfire was designed to be self-hosting, so the owner could rebuild their own system from scratch if needed. The source code is a useful reference.

build-sdk hangs or exits.

Ensure you have just installed this software on a clean Raspberry Pi OS LITE 64-bit install. The OS must not be customized in any way. Do not install any prerequisites other than "git."

Compiling the toolchain uses high random disk IO, all of the Raspberry Pi Zero memory, and at least 1 CPU core.

Your Raspberry Pi must have at least a 1GB swapfile to build DirectFB and SDL2. The script automatically sets this. The default swapfile size is 100MB. If left at its default, the compiler will hang about 75% of the way into the build process for each of these packages.

It takes too long to compile this toolchain.

Make sure you are using a fast microSD card, at least a "Class 10, U1" card. Slower cards may be a fraction of this speed, and will extend build times considerably.

With a Class 10 U1 microSD card, this script will take between 1 and 2 hours to build everything on a Raspi Zero 2. On a Raspi Zero 1 with the same Class 10 card, it takes 2 to 4 hours.

These times may be doubled or quadrupled on a slow microSD card.

Assuming a Raspi Zero 2, it should take about 40-60 minutes to build the firmware, drivers and the DirectFB toolchain. From that point, it should take another hour to build SDL2 and pygame. This is all on a Raspi Zero 2 with a fast Class 10 microSD card.

You can also run **build-sdk.py** on a Pi 4 or 5. Write a fresh Raspberry Pi OS to a microSD card, then boot it once on the Pi Zero 2. This configures the installed Raspberry Pi system specifically for the Pi Zero 2 network hardware (important later). Shut down, then move the microSD card to a faster Pi. Run build-sdk on the faster Pi, and when it is complete, swap back to the Pi Zero 2.

Development environment (IDE) setup and talking to the beepy

Generally all development with the beepy happens over SSH. (The internal keyboard is good, but not *that* good.) Below is one way to set up an environment.

Use mDNS to simplify configuration

If you specified a custom hostname during Raspberry Pi Imager configuration, when the pi comes up, it will be reachable at <hostname>.local, e.g.

```
$ ping beepy.local   <- your beepy's hostname plus .local
```

This makes it easy to manage multiple Pis without having to remember IP addresses.

It makes it independent of DHCP servers, allowing easier development on the road.

It runs equally well over wifi, or tethered with the USB Ethernet Gadget enabled.

Setup SSH keys between your computer and the pi

If you haven't done so, generate a private/public SSH key pair on your computer.

SSH into the pi for the first time:

```
$ ssh user@beepy.local  <- use your Pi's username and hostname
```

On the pi, type:

```
$ ssh-keygen
```

Hit Enter three times when ssh-keygen prompts you for a directory and passphrase.

Back on your computer (assuming a unix box), type:

```
$ ssh-copy-id user@beepy.local
```

Subsequent SSH connections from your computer to the Pi will not require a password.

An authorized key means apps will automatically connect.

Visual Studio Code won't constantly prompt you for password logins.

Notes on using SSH with the Pi

Reinstalling Raspberry Pi OS breaks your SSH keys (unless you copy them over first).

Every time you reinstall the OS on your Pi and re-run ssh-keygen, its SSH key will change.

Your host computer will become very suspicious of the new key, and refuse to connect. To fix this, reset the key in SSH's store, allowing it to re-pair.

On your host PC:

```
$ ssh-keygen -R  <- followed by the Pi's IP or hostname
```

Example: the following removes all keys in your SSH known_hosts files associated with the host on IP address 10.1.1.100:

```
$ ssh-keygen -R 10.1.1.100
```

Setup Git

On the Pi:

```
$ sudo apt install -y git
```

Setup Git for commits:

```
$ git config --global user.name  <---- enter firstname and lastname in quotes
$ git config --global user.email <---- put your email here in quotes
$ git config --global core.editor "vi" <- ONLY if you prefer to use vi
```

(Optional) Permanently mount NFS datastore for Git use

Useful for when Git repositories are kept on a NAS.

Assume a file server at 192.168.1.1, with an NFS export for dataset 'git' in the zpool 'zpool'.

Change parameters in the example below as needed.

On the Pi:

```
# mkdir /mnt/git
# vi /etc/fstab
```

Add this line:

```
192.168.1.1:/mnt/zpool/git /mnt/git nfs auto,nofail,noatime,nolock,intr,tcp,actimeo=1800 0 0
```

Write and quit.

```
:wq!
```

Setup Visual Studio Code

Visual Studio Code only supports the Raspberry Pi Zero 2, not the 1.

- Install Visual Studio Code from https://code.visualstudio.com
- In Visual Studio Code, go to Extensions ("blocks" icon in sidebar) > search for "Remote - SSH" and click Install next to it. It should show Microsoft as the publisher, with a blue checkbox mark.
- Do the same for the "Remote Development" extension.
- Within Visual Studio Code,
  - bring up the command palette at the top of the window
    - mac: ⌘-Shift-P
    - win + lin : Ctrl-Shift-P
  - enter the below text into command palette and hit enter once it pops up
    - Remote-SSH: Connect to Host...
  - edit ssh hosts list
- On Visual Studio Code sidebar, click **Open Folder.**
- Select folder, **beepy-sdk.**
- Click **OK.**
- **Yes,** trust the files in the folder.

References

"Remote Development using SSH," https://code.visualstudio.com/docs/remote/ssh

Raspi Remote Access: https://www.raspberrypi.com/documentation/remote-access/ssh/

Developing apps over USB to the raspi

This is "Ethernet over USB." The raspi, when plugged into a host computer, appears as a USB-to-Ethernet adapter. If you have set up a hostname in Raspberry Pi Imager, it will be reachable at <hostname>.local, which it broadcasts over mDNS.

Currently the pi must be joined to wifi to get internet access.

Connect to the Pi Zero's "center" port (not the corner USB port, nor the HDMI port)

Your computer will see this as a new network interface, and will receive a link-local IP.

Ping beepy.local (or whatever you set it up as) - it will give you the link-local IP of the beepy.

Development loop (without git server)

This was the development loop I used before moving to a git server:

- develop on beepy with visual studio code remote, or vi
- commit git changes: git add .; git commit
- cd .. and tar project folder: tar -cv --no-mac-metadata -f project_folder.tar project_folder/
- connect from Mac with SCP client; download project onto Mac
- untar project folder: tar xvf project_folder.tar
- open in Visual Studio Code and do development work
- commit git changes: git add .;git commit
- cd .. and tar project folder: tar cvf project_folder.tar project_folder/
- connect to beepy from Mac with SCP client; download project onto beepy

A shared network git server simplifies the above process.

Development loop (with git remote)

I began using this process after I got pygame working:

Export an NFSv3 share from a network server. Restrict host access to our development computer and our beepy's wifi IP address.

When creating a tarball on macOS, use the **--no-mac-metadata** argument above. You will still receive warnings on the pi side when decompressing the tarball, but it will not create dot-underscore files in the extracted directory. The specific Mac tar error is ("Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.quarantine').
If you don't add the above argument, by default, macOS tar will copy ACLs into ._ files ("dot-underscore") These clutter up the filesystem. To delete them on the pi side, cd to the root of your project, and type:

```
find . -type f -name '._*' -delete
```

Making an app with pygame

Objective

Take apart existing apps and recombine them to make something new.

Take apart the example app

The app runs on any computer with pygame installed.

The example app contains examples for various GUI and hardware operations on the beepy.

Read the source code, and figure out how the following things were done.

- "blitting" images
- positioning images
- drawing text
- drawing updating text (a timer, or a clock)
- responding to keyboard and side button input

Replace the images and text with your own.

Learning pygame

Introduction to pygame, readthedocs.io

Organizing your game's code: "Programming Game Patterns," Robert Nystrom

Example of a non-trivial pygame project: "Cabbages and Kings," github.com

Drawing graphics

References

"Raspberry Pi pygame UI basics," Jeremy Blythe, Adafruit. https://cdn-learn.adafruit.com/downloads/pdf/raspberry-pi-pygame-ui-basics.pdf

https://realpython.com/pygame-a-primer/#using-blit-and-flip

Scrolling

https://stackoverflow.com/questions/55319181/how-to-scroll-the-background-surface-in-pygame

Dithering color images to black-and-white

I wrote a script to try different dithering diffusion techniques.

My quality rankings, top 3:

1 - hitherdither's Stucki: best compromise between gradation and sharp edges

2 - PIL's Floyd-Steinberg: best for gradation; does not do sharp edges well

3 - hitherdither's Atkinson: higher-contrast (blown out shadows & highlights), sharp edges. Good for dramatic images with lots of black or white.

References

"Ditherpunk - The article I wish I had about monochrome image dithering," https://surma.dev/things/ditherpunk/

Fonts

Font file formats

So far we have used the built-in pygame.font.Font class, and the third-party bmpfont library, with success. These two methods will be described below in more detail.

pygame.font module

FreeType library supports these bitmap font formats:

- X11 PCF Font Format (Portable Compiled Font)
  - Reference: https://freetype.org/freetype2/docs/reference/ft2-bdf_fonts.html
- Glyph Bitmap Distribution Format (claimed to be superseded by PCF)
- Windows FNT (capable of storing a single raster or vector - superseded by TrueType)

Creating a "bitmapped font" via an indexed set of rectangles

https://www.pygame.org/pcr/bitmap_font/index.php

See question here:

https://stackoverflow.com/questions/28429860/is-it-possible-to-create-a-font-from-a-pixel-based-colour-image-for-pygame-bu

Fonts which look great on the Memory LCD

pygame, as installed, supports these font formats:

.otb - OpenType Bitmap

.otb - OpenType Font (outlines, not bitmap)

bitocra (11 and 13 pt)

cherry (10, 11, 13)

- profont-semi : Designed by Andrew Welch, Carl Osterwald, and Steven C. Gilardi. MIT License. https://tobiasjung.name/profont/
- Collection of programming fonts: https://github.com/ProgrammingFonts/ProgrammingFonts
- Oldschool fonts: https://int10h.org/oldschool-pc-fonts/readme/

Display all fonts available to pygame:

```
>>> import pygame
>>> print(pygame.font.get_fonts())
```

Installing bitmap font (ProFont) into the system

See "Adding Fonts", https://wiki.debian.org/Fonts

Enable bitmapped font support:

dpkg-reconfigure fontconfig-config

Change font hinting from Slight to Full (unsure if it does anything)

When it asks "Enable bitmapped fonts by default?" choose Yes.

Install fonts:

sudo cp ~/demo-pygame/lib/font_system/profont-otb-2/ProFontOTB.otb /usr/local/share/fonts/

Check if the font was installed:

```
$ fc-list :family=ProFontOTB
```

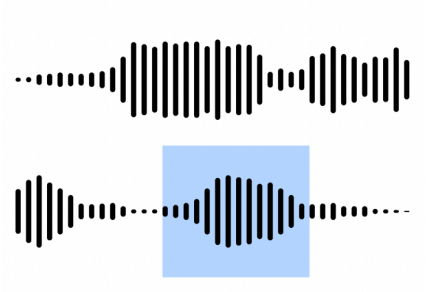
https://fonts.google.com/specimen/Silkscreen
Bitmapped font module for pygame: https://www.pygame.org/pcr/bitmap_font/index.php

damieng bitmap fonts: designed by Damien Guard

Hundreds of beautiful, hand-designed 8x8 bitmap fonts.
License: Freely available in exchange for a mention in the credits section.
Web: https://damieng.com/typography/zx-origins/

wavefont: a typeface for rendering waveforms

Designed by Dmitry Ivanov.
License: SIL Open Font License. Commercially embeddable.
Web: https://dy.github.io/wavefont/
Live demo: https://fonts.google.com/specimen/Wavefont/tester
Download here: https://fonts.google.com/specimen/Wavefont/about



wavefont example.

Fonts for non-Latin characters

The unifont.org guide has a comprehensive selection by region.

GNU Unifont

Contains Basic Multilingual Plane 0: characters for most all modern languages, and a large number of symbols.
License: Can be embedded commercially. GNU Font Embedding Exception and the SIL OFL allow for this.
Download (unifoundry.com)

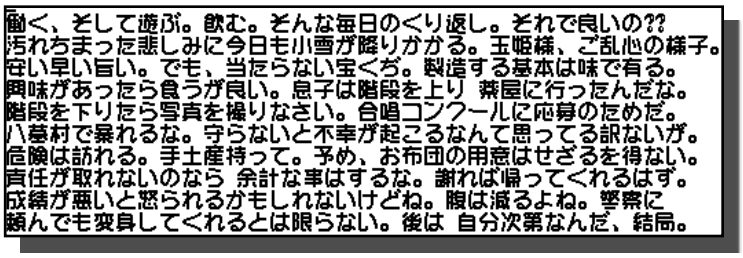
Japanese bitmap fonts

Pau font: "It is a round Gothic style, and its main characteristic is high visibility."
It was designed to be clearly visible, rather than accurately detailed for reading.
Designed by OHNO Yasuaki.
Licensed for non-commercial purposes only.
Mr. Ohno is unreachable as of 2022.

Web: http://udumaki.s36.xrea.com

Web 2: Curated page of Japanese fonts, including Pau font

Pau font. Top image is zoomed in to show detail.



Japanese TTF fonts

MPLUS collection
License: OFL-1.1
https://github.com/coz-m/MPLUS_FONTS

Other font collections

Peter Wiegel Fonts

http://catfonts.de
http://www.peter-wiegel.de/Inkopins.html

References

Displaying text on screen:
https://stackoverflow.com/questions/20842801/how-to-display-text-in-pygame
Using pixel fonts with PIL:
https://stackoverflow.com/questions/619618/using-pixel-fonts-in-pil

Blitting

https://stackoverflow.com/questions/17454257/a-bit-confused-with-blitting-pygame
Blitting text: https://stackoverflow.com/questions/19117062/how-to-add-text-into-a-pygame-rectangle

monobit: tools for working with bitmap fonts

https://github.com/robhagemans/monobit
Syntax (read the README):

```
./convert.py inputfont.pcf to outputfont.png
```

Note from manpage of vfontas - a bitmap font file transformation utility
https://manpages.ubuntu.com/manpages/impish/man1/vfontas.1.html
The Linux console accepts bitmap fonts up to 32x32 in size.

Goal: Music player app (tracstar)

Features and functions

- List files in music folder
- Don't crash when trying to open a non-audio file
- Detect and configure system for plugged-in USB audio devices
- Volume control
- (Bonus) MP3, Vorbis, WAV, and FLAC support
- (Bonus) Pair with bluetooth headphones
- (Bonus) Send audio through HDMI port
- Display track length, and time played.
- Display "slider" showing progress through track
- Show ID3 tags
    - https://github.com/frececroka/id3parse.py
- Allow playpause
- (Bonus) Playlist support
- (Bonus) Spectrogram

Audio

Setting the audio output device for the pygame audio mixer:
https://stackoverflow.com/questions/57099246/set-output-device-for-pygame-mixer

Pixel art workflow

Workflow process

Import image

Set brightness/contrast and remap colors if needed

Scale (choose Nearest Neighbor for easiest path to pixelly look)

Make a white background layer and lock it

Lock the art layer and change its opacity to 50% or whatever lets you see it without obscuring your pixels

Make a new layer, opacity 100% or like 80-90%

Switch to pixel-art brush and black color. Start dropping pixels in. Squint. Step back. Scale down small with shortcut to zoom in/out.

Making pixel art with Krita (free for Linux, Windows, Mac)

Setup a few presets so that Krita works efficiently for pixel art.

Tracing

* Zoom in/out - Cmd+ Cmd- or pinch
* Move a layer - T key

Configure brush tools

With monochrome pixel art, we either want a pixel to be white, black, or clear (if using masking, like in a cursor).

* Tools menu > Scripts > Ten Brushes

Pixel create tool

Pixel erase tool

Patterns

* Krita: Pixel Art Dithering brush

Dockers > Brush Presets > expand to panel, center, resize

Right-click > Remove from this tag

Now go to All

Lower left > Search field > Type Pixel Art

Find the pixel art brushes. Note there are 2 classes. One is "PixelArt," for Krita3, and the one you want is "Pixel Art," for Krita 4. Right-click > Assign to Tag > My Favorites.

Top dropdown menu > choose My Favorites. Your pixel art tools should be there.

Inspiration: contemporary black and white pixel art

The linked page contains a gallery of HyperCard art from 1986-1990. Hand-converted into transparent PNGs by mariteaux.

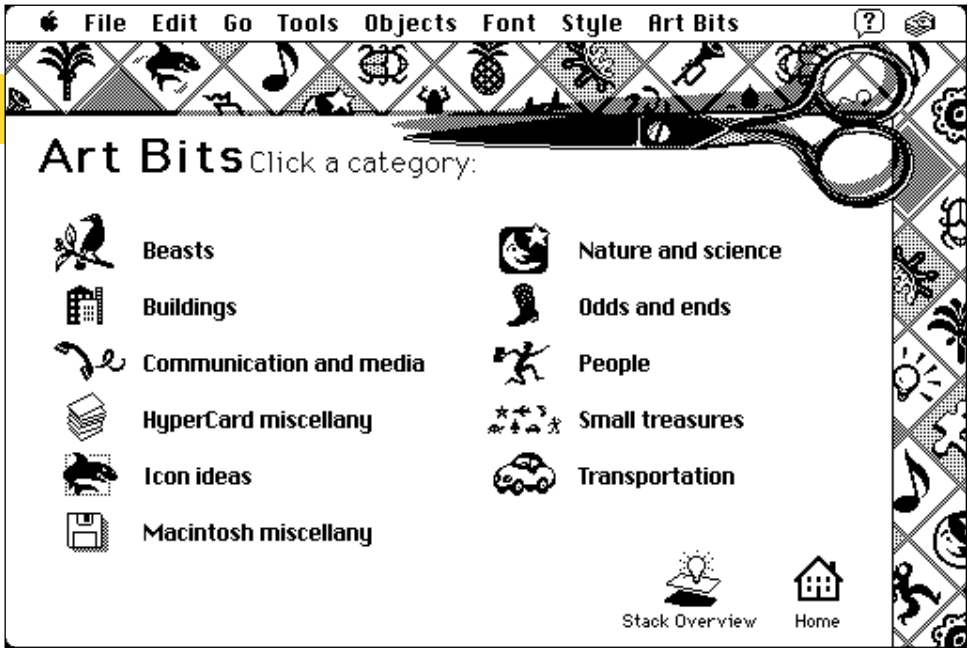
Copyright Apple Inc.

These are not free and cannot be re-distributed.

They are linked here as good examples of detailed monochrome pixel art, to develop your technique.

Web: http://archives.somnolescent.net/web/mari_v2/junk/hypercard/

Web backup: archive.org link



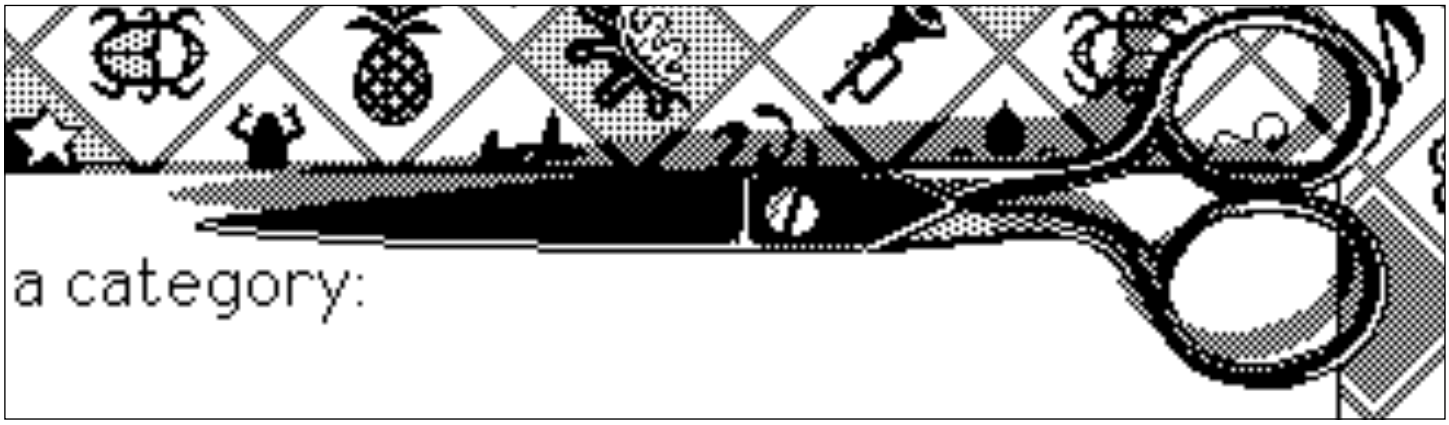
On a 512x342 screen, every pixel counts.

The scissors in the upper right corner show fine details.

Their shiny, metal blades cast a sharp shadow on the wallpaper behind them.

The wallpaper panel above the screw, with the oak leaf, is subtly dithered where the shadow meets it.

The domed screw on the top of the scissors shows the direction of the light source.



Copyright 1986 Apple. Scissors detail from the "Art Bits" HyperCard stack.

Art styles and diffusion prompts

"Diffusion" in this context is AI generated art.

This is a running list of art styles that may look good on the Memory LCD.
The Memory LCD has a "pixels on a mirror" appearance, similar to a daguerreotype.

Techniques

- Generate your images as close to the output size as possible (400x240).
- Downscaling eats up small details when outputting to a monochrome display. Pixels must be black or white.
- Consider running your images through a dithering filter. (A different form of diffusion.) The application in apps/image/ditherbox/ditherbug.py will generate several black and white dithered images per source image.

Art styles to use in prompts:

- **Etching** is a technique to produce finely detailed images with few colors and strong contrast.
- **Crosshatching** adds visual interest and complexity to shaded areas. Beware downscaling crosshatching!
- **Pen and ink** is a freehand illustration technique which obtains fine lines.
- **Woodcut** is a technique where the artist carves a "stamp" into a block of wood or rubber. Bold lines.

Artists:

* **Aubrey Beardsley**: Finely detailed black-and-white artwork. Art Nouveau style.

**"Excalibur in the Lake," Aubrey Beardsley, 1893.
Public domain. Courtesy of wikiart.org.**

Debugging

gdb <binary name>

run

hit ctrl-z to sleep (fg to resume in bash)

info

info frame

backtrace

Explanation of Fusion (libfusion) component: https://lkml.indiana.edu/hypermail/linux/kernel/0910.0/03028.html

Hardware access

Control the RGB LED

Install ardangelo's rp2040 firmware, and Linux keyboard driver.
The LED now appears as a filesystem device.

```
sudo su
# echo 1 > /sys/firmware/beepy/led_green
# echo 1 > /sys/firmware/beepy/led_blue
```

Interacting with the RP2040 over i2c

Show the i2c busses:

```
ls /dev/i2c (hit tab). It should show i2c-1 and i2c-2.
```

Scan the i2c busses:

```
sudo i2cdetect <bus number, either 1 or 2>
```

It shows something on bus1, 0x1f.

Stock SQFMI firmware is here:
https://github.com/sqfmi/i2c_puppet

Useful protocol info here:
https://github.com/sqfmi/i2c_puppet#protocol

Determining connected audio devices

On a default Raspi 2 W,

```
cat /proc/asound/cards
```

returns this:

```
 0 [vc4hdmi       ]: vc4-hdmi - vc4-hdmi
                     vc4-hdmi
```

Connecting a SYBA USB DAC over a USB OTG hub returns this:

```
beepy@beepy:~/demo-pygame $ cat /proc/asound/cards
 0 [vc4hdmi        ]: vc4-hdmi - vc4-hdmi
                      vc4-hdmi
 1 [Device         ]: USB-Audio - C-Media USB Audio Device
                      C-Media USB Audio Device at usb-3f980000.usb-1.1, full speed
```

Get more info:

```
aplay -l
```

should list both "cards" with card 0 being the VideoCore4 HDMI output, and card 1 being USB Audio Device:
card 0: vc4hdmi [vc4-hdmi], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
card 1: Device [C-Media USB Audio Device], device 0: USB Audio [USB Audio]

This may result in errors after it displays:
**** List of PLAYBACK Hardware Devices ****
Delete ~/.asoundrc to fix this. It will give you an warning mentioning it's write-protected.

Create a config file for ALSA:

```
vi /etc/asound.conf
```

put in the following:

```
pcm.!default {
```

```
    type hw
    card 1
}

ctl.!default {
    type hw
    card 1
}
```

raspi-config > 1 - System Options > S2 - Audio > will only allow you to choose one output, vc4-hdmi.

Part II : beepy-sdk manual

Part II is a centralized knowledge repository for developing on the beepy.

It consists of notes made during development of the "build-sdk.py" script.
This is the script which builds the beepy SDK used in glowfire.

The build-sdk.py script itself contains the most up to date documentation, but these experimental notes may be helpful to some developers.

Appendix 1 - Experimental Notes

Table o' contents

Goals

√ Bitmapped graphics on the beepy - using DirectFB2 (achieved 2023-07-20)

- An SDK for beepy owners
  - Auto-downloads and builds:
  - √ Keyboard with mouse support
  - √ DirectFB2 toolchain and examples
  - Demo app for the Berry: an alarm clock (taken from Stretch Goals below)
    - Current time always displayed
    - Setting to enable/disable piezo GPIO (stretch goal: pins you specify)
    - Solder on a piezo buzzer to GPIO pins
    - Plays alarm tone through piezo
    - Press any keyboard key to turn it off
- Documentation showing how to build it
- Prebuilt ISO or binaries that people can immediately load onto the Raspi Zero's microSD card, for loading into a new beepy or reinstalling an existing one

Stretch goals

- 6MHz SPI overclock for display driver
- App: Desk Clock
- App: "LED Mode: White Light", "Candle Light", "Fire Light", "RGB Sliders", "HSV Color Wheel" and "Off". Candle and Fire lights flicker and change color slightly.
- App: Piezo sensor realtime graph; alternate GPIO from input to output to use as buzzer
- Art assets that show the beepberry, front/rear, connectors, etc.
- "Screensaver" or desk clock that shows the pins
- Build "full fat" DirectFB including Multi option, and Raspi accelerated branch (for HDMI out).
- √ FPS counter (achieved with snakeware, 2023-11)
- Auto-dithering for images going to the display
- Auto-scaling for videos
- Compile in specific console fonts, optimized for this display
- Build a kernel, with framebuffer as primary tty, with HDMI secondary
- Some kind of GPU-to-FrameBuffer solution, where everything is done in a buffer on the GPU, using OpenGLES, and then somehow mapping it to the system framebuffer, double buffer style. NEON is available in hardware on the Zero 2. NEON is disabled in stock Raspberry Pi OS, but is enabled in the DietPi distribution.
- "Ultra low power mode" for the display, when hosted by the RP2040 - it handles rudimentary GUI when Pi Zero is off - like an alarm clock, data logging and graphing, slideshow, etc. It would require about 4-6 bodge wire runs between the RP2040 and the springy 40-pin ZIF backplane connector on the beepy backplane.
- Disable onboard GPU to save power
- √ GUI widget system + demo apps (achieved with autobooting snakeware, 2024-03-30)
- √ SDL2 can consistently run at least one demo app (achieved with snakeware, 2023-11)
- √ pygame (achieved 2023-09)

Hardware goals

- Cover glass touch panel
- √ (Achieved 2023-09-01) Polished acrylic case. Lay up from sheets of acrylic, and bond with Weld-On #4 from TAP Plastics. Cannot laser-cut polycarbonate due to toxic offgas and brown kerf. Try mechanical routing+polish. Dimensions are 104x74, no more than 10mm thick.

Hardware Notes

Power and Thermals

You can reduce TDP from ~2W to ~1W thus:

Edit /boot/config.txt to underclock and undervolt:

```
arm_freq=600
arm_freq_min=600
over_voltage=-8
over_voltage_min=-8
```

Reference: https://forums.raspberrypi.com/viewtopic.php?t=324988

Power and USB Data

Notes on Power and USB:

- I do not have a battery connected. My beepy's USB-C port stopped functioning 60 days in. Since then, I've powered it only using the Raspi micro USB ports, in the following configurations. A quick note on what the ports do:
- Bottom port is USB-C. Its data lines are connected to the beepy onboard RP2040. Its power lines are connected to the beepy onboard power regulator circuit.
  - The power regulator powers everything, including the RP2040, and 40-pin raspi connector.
- Top 3 ports on the Raspi: mini HDMI, micro USB Data, and micro USB Power.
  - Each USB port can power the Raspi Zero, and everything else on the beepy.
  - The center, micro USB data port can additionally:
    - act as a USB device using the DesignWare USB2 DRD Core kernel module (dwc2)
    - act as a USB host for multiple devices (kbd, mouse, etc.) using a USB On-The-Go (OTG) hub
- The beepy's embedded RP2040 has the ability to reset the raspi, and control other elements of power on the system. The RP2040 is connected to the USB-C data lines for additional flexibility.
- For development purposes, we power the beepy from a host PC through the center, USB-DATA port on top of the raspi. The instructions at the beginning of this guide enable the USB Ethernet gadget, so the Pi shows up on your host computer as an Ethernet adapter, with a ping-able mDNS (Bonjour, multicast) hostname which you have pre-set in the Raspberry Pi Imager.
  - Connect one or more flashdrives, keyboards, mice, etc. using a USB OTG adapter. ("USB OnTheGo.")
    - OTG hub connected to the raspi center, data MicroUSB connector. The hub itself is powered externally with a 5VDC barrel jack. 2 out of 3 ports are used on the hub by a Lenovo KC-1957 wireless ThinkPad keyboard USB dongle, and the other by a SYBA USB stereo adapter. This topology has worked fine. Care was taken so that the beepy was standalone, and not connected to any other computers (split ground).
  - Headphone output. One SYBA USB to stereo adapter, connected to the raspi USB-data connector via either a USB OTG hub, or a small USB OTG to USB-A adapter. The other end is connected to headphones. The raspi's POWER USB connector ("outer" MicroUSB port) is connected to power.

From a public source:

"Long holding the "End Call" key (~3 seconds) will trigger the key KEY_POWER and safely shutdown the Pi."

- Relevant to SQFMI and ardangelo forks of bbqX0kbd firmware & Linux driver. These are more integrated into the beepy hardware, allowing the RGB LED to be controlled, etc.

Microphone

"Put a solder jumper on JP3 to enable the mic."

- The RP2040 is connected to the microphone in the Q20 keyboard.
- I have not found JP3 on the silkscreen. I suspect it is unmarked, or on the Q20 keyboard itself.

RP2040 Firmware Update

CURRENT: Follow instructions in build-sdk.py. They replace this process.

To update the beepy's firmware:

- Slide the power switch off (left if facing up)
- Connect the beepy to your computer via USB-C.
- While holding the "End Call" key (top right on the keypad), slide the power switch on.
- The beepy will present itself as a USB mass storage device, drag'n'drop the new firmware (*.uf2) into the drive and it will reboot with the new firmware.

Beepy Power Failure & Troubleshooting

Connected USB to it - the RGB LED flickered and the Pi's activity indicator began a constant flicker at 0.5Hz. Disconnected. Now the beepy hardware shorts across USB, enough to turn off an external USB hub when connected to it.

Discord sources mentioned this issue, and one person claims/suspects it's something on U2 shorting, causing D1 (diode to left of USB port when viewed from back) to go full short and "turn into a light emitting diode."

Issue occurs whether S1 (power) is ON or OFF.

Investigation

U2 is TPS61090

D1 is a Zener diode

Examine U2 pads under microscope

Check for shorts around U2 pinout

Desolder D1

If you need to change the wifi network the Pi is connected to

Connect the microSD card to your host computer.

```
vi /bootfs/wpa_supplicant.conf
```

Inside vi:

```
country=us (enter your two-char country here)
update_config=1
ctrl_interface=/var/run/wpa_supplicant

network={
    scan_ssid=1
    ssid="MyWifiTravelNetwork"
    psk="MyKey"
}
```

Emulating a USB Mass Storage Device using Linux Mass Storage Gadget

Read reference - may need to enable in kernel: https://linux-sunxi.org/USB_Gadget/Mass_storage

MSG can provide up to 8 SCSI disks (LUNs) to the USB host.

Create 128MB backing storage file:

```
# dd bs=1M count=128 if=/dev/zero of=/root/data/backing_file
```

Load gadget driver

Tell MSG to provide a single LUN with backing storage maintained in /root/data/backing_file:

```
$ sudo modprobe g_mass_storage file=/root/data/backing_file
```

Tell MSG to provide two LUNs, one a backing file and the other a device:

```
$ modprobe g_mass_storage file=/root/data/backing_file,/dev/hda7
```

(Optional) Format backing store

Export backing store

Reference: http://www.linux-usb.org/gadget/file_storage.html

Replacing the console fonts

- Install Sharp display driver
- Setup console mode to the framebuffer
- Build and install keyboard driver for RP2040; put into mouse mode
- (optional) Download and install custom bitmap fonts into kernel
- (optional) Recompile kernel for framebuffer first, then HDMI

console fonts here: /usr/local/consolefonts

in /boot/cmdline.txt:
… fbcon=font:VGA8x16

Typing “who am i” into the hardware kbd results in tty1

Build and install RP2040 firmware

```
cd 3rdparty/pico-sdk
git submodule update --init    # may need to run several times until it clones into:
3rdparty/pico-sdk/lib/tinyusb
mkdir build
cd build
cmake -DPICO_BOARD=beepberry -DCMAKE_BUILD_TYPE=Debug -S ../ -B ./
make
```

Install dependencies and build the Raspberry Pi Pico SDK:

```
$ sudo apt install cmake gcc-arm-none-eabi libnewlib-arm-none-eabi libstdc++-arm-none-
eabi-newlib
$ cd 3rdparty/pico-sdk
$ cmake build
```

CMake should successfully complete configuration for the "pico-sdk" dependency.

Now configure and build the main project for the beepy-rp2040 firmware:

$ cd ../..

$ cmake -DPICO_BOARD=beepberry -DCMAKE_BUILD_TYPE=Debug -S . -B build/

$ make

This produces i2c_puppet.* files in beepberry-rp2040/build/app/. Specifically:

ic2_puppet.bin

i2c_puppet.dis

i2c_puppet.elf

i2c_puppet.elf.map

i2c_puppet.hex

i2c_puppet.uf2

Flashing RP2040 firmware with OpenOCD

This allows you to reflash the RP2040 in-circuit, completely on the beepy, without having to unplug and use the USB-C port with a desktop computer.

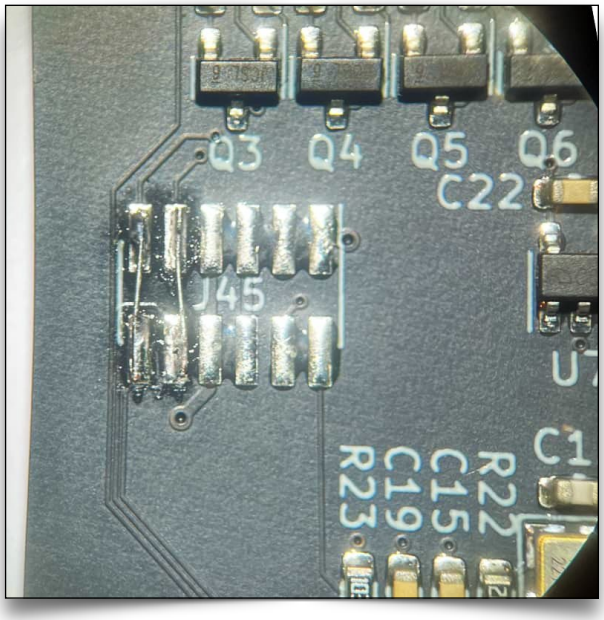
We can do this because SQFMI conveniently placed the SWDIO and SWDCLK pins for the Raspi Zero and the RP2040 across from each other on header J45.

Solder the leftmost 2 pairs of pins on J45.

The far left pin connects SWDIO to Raspi pin 25.

The second from the left connects SWCLK to Raspi pin 24.

J45 SWD modification photo



Connect the leftmost two pins across from each other, then the second from the left to route the RP2040's SWCLK/ SWDIO lines to the Pi Zero's GPIO 24 and GPIO 25 pins, respectively.

Flash the RP2040 firmware using the freshly soldered SWD lines:

```
$ sudo apt install -y openocd
$ sudo openocd -f interface/raspberrypi-native.cfg -f target/rp2040.cfg -c "transport
select swd" -c "adapter gpio swdio 24" -c "adapter gpio swclk 25" -c "program
i2c_puppet.elf verify reset exit"
```

Note: The config directories for OpenOCD are in: /usr/share/openocd/scripts/ (e.g. "target," "interface," etc.)

Reference here: https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf

Configure drivers for beepy hardware

- https://github.com/ardangelo/sharp-drm-driver

Build SQFMI keyboard driver

SQFMI drivers here:
https://github.com/sqfmi/bbqX0kbd_drive
cd ~/beepy-sdk
git clone https://github.com/sqfmi/bbqX0kbd_driver
cd bbqX0kbd_driver
git checkout c2acdd7317238a46c25daee361a7b187cc167ec6

Keyboard and Sharp drivers install items here:

- /lib/modules/6.1.21-v8+/extra/beepy-kbd.ko.xz
- /lib/modules/6.1.21-v8+/extra/sharp-drm.ko.xz
- /boot/overlays/beepy-kbd.dtbo
- /boot/overlays/sharp-drm.dtbo

Primer: Bitmapped graphics on the beepy

It is always the simple that produces the marvelous.
— Amelia Barr, American author

"You don't need X, which is the the most enticing aspect of having the framebuffer."
— https://unix.stackexchange.com/questions/33596/no-framebuffer-device-how-to-enable-it

We don't use the GPU, OpenGL, MESA, X, or Wayland. It's all software rendering.

Raw framebuffer access

Framebuffer is at /dev/fb1
To work with the framebuffer, all you need is the sharp-drm-driver. You don't need DirectFB, etc.

Changing the framebuffer directly

"Everything is a file, even hardware."
To dump screen contents to a file:

```
$ cat /dev/fb1 > ~/fb_image
```

To write that dumped file back to screen:

```
$ sudo cat ~/fb_image > /dev/fb1
```

Framebuffer technical notes

Framebuffer size: 375KB (ardangelo sharp-drm-driver, configured for the beepy Memory LCD)
Framebuffer size: 1MB (on the earlier w4ilun Sharp-Memory-LCD-Kernel-Driver, for a different display)
Format is binary. First 2399 are 0's. Offset 2400 is where the screen begins.
Pixel format = FF0000 is black, 000000 is white

Sharp Memory LCD technical notes

- The beepy uses a "Sharp Memory LCD, 2.7 inch, LS027B7DH01A" display. Datasheet here.
- Do not persist a static image for more than 2 hours on the Memory LCD. (From datasheet.)
  - In my testing, it lightly "burned" the image into the screen. It took ~24h off for the pixels to "relax" back to non-burned-in state.
- Pixel pitch is 173ppi, based on datasheet = (25.4mm / 0.143mm dot pitch)

TODO: Ideas

- NEON compiler flags for DirectFB and SDL add hardware-accelerated blitting, scaling, etc. This requires a kernel with NEON support compiled in. Raspi OS does not have NEON support, but DietPi does.
- "Hardware Accelerated Snag." If there's a way to do DMA between the GPU memory and the Sharp Memory LCD framebuffer, without copying through the CPU, and packing the GPU framebuffer pixel format a certain way, we might be able to render in the GPU, do any transformations needed, then mirror it to the framebuffer. Snag sort of does this but copies using the CPU and does dithering on the CPU. See "GPU to Framebuffer" section in Experimental Notes.

References

Datasheet, Sharp Memory LCD, LS027B7DH01A, Sharp Devices
Interface format description: "Programming Sharp's Memory LCDs," Sharp Devices Europe GmbH
Linux kernel, framebuffer documentation, https://kernel.org
Linux kernel, Framebuffer internals, https://kernel.org

Building DirectFB2 toolchain

References for DirectFB2 toolchain

https://linux.die.net/man/5/directfbrc

Notes on fluxcomp and DirectFB2 building

fluxcomp appears to need libdirect (provided by DirectFB2) - but DirectFB2 needs fluxcomp to compile.
The install scheme is generally:

1. try to buildinstall fluxcomp - it will compile without libdirect support
2. run "sudo ldconfig" to force the linker to update its cache
3. buildinstall directfb2
4. buildinstall fluxcomp - now it should have libdirect support
5. run "sudo ldconfig" one more time
6. possibly buildinstall directfb2 again?

TEST: symbol list and md5 of fluxcomp and libdirect, compare before and after
-- we could match against nm output - look for the missing symbols - start with autotools

Setup DirectFB2 toolchain

Install dependency: fluxcomp

```
git clone https://github.com/kevleyski/fluxcomp
cd fluxcomp
```

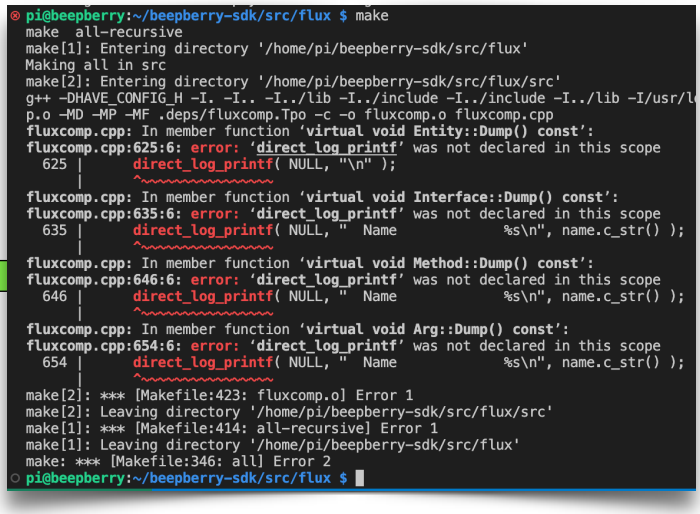
Patch fluxcomp.cpp

In fluxcomp.cpp, in between lines 58 ("...list.h>") and 59 ("...mem.h>") create a new line and add the following to it:

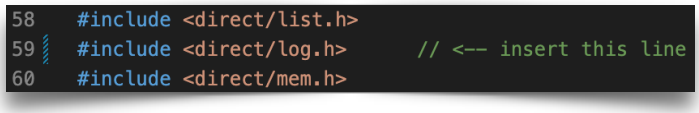
```
#include <direct/log.h>
```

This imports the log.h header file.

If you don't do this, you will receive an error at maketime which will read, in part, "'direct_log_printf' was not declared in this scope".



When building fluxcomp.cpp, gcc can't find the direct_log_printf function. Include direct/log.h in fluxcomp.cpp's #include section to fix this.



Build fluxcomp

```
./configure
make
```

The pi will appear to be frozen for some time during the **make** phase. On a Raspi Zero 2 it takes around 15 minutes. If using Visual Studio Code in Remote mode to the beepy, your SSH session may drop.

Be patient and leave the pi alone. It will eventually make the software. You can push keys on the keyboard and they will sluggishly draw to the screen. The Raspi has not crashed.

Install fluxcomp

```
sudo make install
```

Install dependencies for building software

```
sudo apt install git cmake
```

Install dependency: meson

```
pip install meson
```

Build dependency: fusionsound (optional)

```
cd ~/beepberry-sdk/src
git clone https://github.com/directfb2/FusionSound2
cd FusionSound2
meson setup build
meson compile -C build
sudo meson install -C build
```

(Optional, in progress) Modified DirectFB2 build (with NEON)

Increase swapfile size to accommodate DirectFB2 build

DirectFB2 build freezes Raspi Zero 2 W on 64-bit Raspi OS.
Increase swapfile to 1GB.

Support NEON

meson configure build -Dneon=true -Dc_args=arm
c_args (extra arguments passed to C compiler)
neon=true

meson compile --clean -C build (to make clean)
install libdl

Build DirectFB2

Download DirectFB2 sources, and create a directory named "build" for Meson to build them in:

```
cd ~
git clone https://github.com/directfb2/DirectFB2
cd ~/DirectFB2
meson setup build
```

Define build options for DirectFB2: disable Intel MMX support; disable NEON support as it leads to compiler errors. (NEON support must be compiled into kernel; it's not present on Raspberry Pi OS.)

```
meson configure build -Dmmx=false -Dneon=false -Dmulti=true
```

Build DirectFB2. Note it will fail after a while - let it get to that point:

```
meson compile -C build
```

It's OK that it fails. Move the .c and .h files that fluxcomp has built, into the folder:

```
mv build/Core* build/src/core/
```

Now retry the compile and it should work:

```
meson compile -C build
```

Install DirectFB2:

```
meson install -C build
```

It will ask to use "sudo" to install with elevated privileges. Allow it by pressing "y" and Enter.

Headers install to /usr/local/include/directfb-internal/ and directfb/
Libraries install to /usr/local/lib/arm-linux-gnueabihf/

NOTE START - ONLY FOLLOW IF YOU HAVE BUILD ISSUES

If something fails and you need to start clean:

```
meson setup --reconfigure build
```

or

```
meson setup --wipe build
```

Note that the wipe will remove your hand-patched/hand-moved sources (the Flux compiler output).
It will still retain your user-defined options. To delete everything:

```
rm -rf build
```

NOTE END - CONTINUE WITH STEPS

Setup DirectFB2 examples (installs to /usr/local/share/directfb-examples)

```
git clone https://github.com/directfb2/DirectFB-examples
cd DirectFB-examples
meson setup build
meson compile -C build
su
# meson install -C build
```

Test DirectFB2 examples

This is done from an SSH terminal, but could be done from the Q20 keyboard and internal console.

Set DirectFB2 to use the framebuffer device.

```
# export DFBARGS="system=fbdev,fbdev=/dev/fb1"
```

You can now run the DirectFB2 example programs.

This example shows a quick animation, then quits after a few seconds:

```
# /usr/local/bin/df_pss
```

This example shows keyboard and mouse input; good for testing the beepberry hardware:

```
# /usr/local/bin/df_input
```

Press Ctrl-C on your SSH console to stop the example.

A starfield controlled by the mouse: a good way to test if your touchpad is working.

```
# /usr/local/bin/df_spacedream
```

This example shows a chessboard texture. It can be moved around by clicking and dragging on the touchpad. It runs until you end the program manually.

```
# /usr/local/bin/df_texture
```

Press Ctrl-C on your SSH console to stop the example.

# /usr/local/bin/df_input <- test input app
df_texture <- chessboard texture on flapping cloth
df_pss <- should give you an animation, then ends after a few seconds
df_knuckles
df_matrix
df_neo <- demo
df_particle

Install keyboard driver with mouse support

```
cd ~
git clone https://github.com/sqfmi/bbqX0kbd_driver
./installer.sh --BBQX0KBD_TYPE BBQ20KBD_PMOD --BBQ20KBD_TRACKPAD_USE
BBQ20KBD_TRACKPAD_AS_MOUSE --BBQX0KBD_INT BBQX0KBD_USE_INT --BBQX0KBD_INT_PIN 4
sudo reboot
```

Then test the DirectFB2 input example again:

```
su
# export DFBARGS="system=fbdev,fbdev=/dev/fb1"
# /usr/local/bin/df_input
```

Setup DirectFB2 LiTE GUI widget toolkit

Install dependencies (not optional in this case, but was with DirectFB2 itself):

```
sudo apt install libdrm-dev
```

Configure LiTE to use PNG files as GUI widgets, then build LiTE:

```
cd ~
git clone https://github.com/directfb2/LiTE
cd LiTE
meson setup build
meson configure build -Dimage-headers=png
meson compile -C build
```

```
meson install -C build
```

MESON TIP: If you ever need to rebuild LiTE, do it thusly (this goes for all of DirectFB2):

```
cd ~/LiTE
meson --reconfigure build
meson compile -C build
meson install -C build
```

Setup DirectFB2 LiTE GUI examples

```
cd ~
git clone https://github.com/directfb2/LiTE-examples
cd LiTE-examples
meson setup build
meson compile -C build
meson install -C build
```

Test DirectFB2 LiTE examples

They are installed in /usr/local/bin/lite* and their resources are in /usr/local/share/lite-examples

Make the LiTE library files visible to the example:

```
sudo su
# LD_LIBRARY_PATH=/usr/local/lib/arm-linux-gnueabihf/;export LD_LIBRARY_PATH
```

Set DirectFB2 to use the framebuffer memory LCD:

```
export DFBARGS="system=fbdev,fbdev=/dev/fb1"
```

Run the example which displays a progress bar, then quits:

```
# /usr/local/bin/lite_progressbar
```

```
# /usr/local/bin/lite_image
```

Run the example which allows RGB slider adjustment (does not affect beepberry LED):

```
# /usr/local/bin/lite_slider
```

Test 2 LiTE apps concurrently

This gives you a feel for what you can do. Note the animation app responds to the touchpad while the progressbar app is running.

```
su
# export DFBARGS="system=fbdev,fbdev=/dev/fb1"
# /usr/local/bin/lite_animation &
# /usr/local/bin/lite_progressbar
```

Setup DirectFB2 image providers (handles all kinds of formats)

Install dependencies:

```
sudo apt install -y libavcodec-dev libavformat-dev libavutil-dev libswscale-dev
libimlib2-dev libopenexr-dev libtiff-dev libwebp-dev
```

Install DirectFB2-media:

```
cd ~
git clone https://github.com/directfb2/DirectFB2-media
cd DirectFB2-media
meson setup build
meson compile -C build
meson install -C build
```

Allow it to use sudo to gain elevated privileges.

Note: You can search for whatprovides-like info by installing "apt-file."
Try meson setup build again, with the --reconfigure option to discover dependencies:

```
meson setup build --reconfigure
```

Only do this if you're messing about with dependencies.

Setup DirectFB2 media samples

```
cd ~
git clone https://github.com/directfb2/DirectFB-media-samples
cd DirectFB-media-samples
meson setup build
meson compile -C build
meson install -C build
```

Allow it to elevate privileges to install.
Samples install in /usr/local/bin/. There are four:
df_databuffer
df_fonts_sample
df_image_sample
df_video_sample

Test DirectFB image viewer example

```
su
# export DFBARGS="system=fbdev,fbdev=/dev/fb1"
# /usr/local/bin/df_image_sample
```

Test DirectFB video player example

```
wget
su
# export DFBARGS="system=fbdev,fbdev=/dev/fb1"
# /usr/local/bin/df_video_sample
```

Setup DirectFB2-tools (optional tools used for diagnostics + dev)

Install tools

```
cd ~
git clone https://github.com/directfb2/DirectFB2-tools
cd DirectFB2-tools
meson setup build
```

Setup DFBTerm ("DirectFB Terminal Emulator")

Note: It depends on DirectFB2-LiTE.

```
cd ~
git clone https://github.com/directfb2/DFBTerm
cd DFBTerm
meson setup build
meson compile -C build
meson install -C build
```

Answer "y" when asked to use sudo to gain elevated privileges.
Set DirectFB2 to use the framebuffer memory LCD:

```
su
# export DFBARGS="system=fbdev,fbdev=/dev/fb1"
# /usr/local/bin/dfbterm
```

Advanced DirectFB command line options

/usr/local/bin/df_input --dfb-help

* Show many, but not all, options.

/usr/local/bin/df_input --dfb:graphics-vt

* This sets the virtual terminal to graphics-only. This suppresses console messages while you're using DirectFB.

See directfbrc reference here: https://linux.die.net/man/5/directfbrc

Making a DirectFB2 app

Themes

LiTE allows you to create interfaces for DirectFB2 GUI apps.

Theme elements are stored in LiTE/data/. Elements like buttons, windows, etc. are represented doubly: once by a DFIFF file, and once by a PNG file. You can specify which resource format DirectFB2 looks for, so for ease of editing, I have only been working with the PNG files.

TODO: Can we override the flag they use (DEFAULT_THEME or similar)? Right now we have to modify the source datafiles.

Do we have the option to compile themes into LiTE apps on a per app basis, or do they all need to be compiled into LiTE? (Want ability to edit app themes without recompiling the LiTE library.)

Flux files: Located in DirectFB2/src/core/CoreDFB.flux, etc.

Modify the LiTE data dir with PNGs

Go into clock dir.

meson setup --reconfigure

meson configure -Dimage-headers=png build

meson compile -C build

sudo meson install -C build

Debug DirectFB2 with strace

In this example, running /home/pi/demo-python/main.py:

```
root@pi:/home/pi/demo-pygame# strace -o /home/pi/strace.txt ./main.py
```

Then grep it for things you are looking for. In my case, DirectFB2 was not applying configs I set up in /etc/directfbrc, so I grepped for directfbrc, including 3 lines Before and After for context.

grep directfbrc /home/pi/strace.txt -B 3 -A 3

In my case, this was the output that helped me:

openat(AT_FDCWD, "/usr/local/etc/directfbrc", O_RDONLY) = -1 ENOENT (No such file or directory)

DirectFB was actually searching in the path /usr/local/etc/directfbrc.

References

DirectFB Reference Manual : http://gongg08.gitee.io/directfb_doc/

links-g-directfb (Links browser for DirectFB)

https://aur.archlinux.org/packages/links-g-directfb

SDL2 Toolchain

In the DirectFB version, this runs on top of the DirectFB toolchain. You will need to build the entire DirectFB toolchain first.

SDL2 for pygame (newer instructions, in progress)

Increase swapfile to 1GB (see DirectFB instructions above)

Install dependencies:

```
apt install -y libsamplerate0-dev libibus-1.0-dev libudev-dev libudev0 libudev1 libdrm-dev libgbm-dev
```

Configure:

```
./configure --enable-arm-neon=yes --enable-video-kmsdrm --disable-video-x11 --disable-video-opengl_es
```

Note the output. It is normal for "checking for Raspberry Pi..." to return no.

```
> checking for ARM_NEON... no
> checking for OSS audio support... yes
> checking for libasound headers version >= 1.0.11... found.
> checking for fusionsound >= 1.1.1... yes
> checking for Raspberry Pi 2/3... no
> checking for EGL support... yes
> checking for OpenGL headers... no (same for OpenGL ES headers v1 and v2)
```

Make:

```
make
sudo make install
```

SDL2-ttf

```
cd src/
wget https://github.com/libsdl-org/SDL_ttf/releases/download/release-2.20.2/SDL2_ttf-2.20.2.tar.gz
tar xvf SDL2_ttf-2.20.2.tar.gz;rm SDL2_ttf-2.20.2.tar.gz
cd SDL2_ttf-2.20.2/
./autogen.sh
./configure --enable-harfbuzz=no --enable-harfbuzz-builtin=no
make
sudo make install
```

It installs libraries into /usr/local/lib.

sdl2-image : follow instructions below.

sdl2-mixer: follow instructions below.
It installs libraries into /usr/local/lib.

Continue to Pygame installation. I'm putting notes here:

Download pygame source, cd to it, then run
sudo python3 setup.py

SDL info : https://stackoverflow.com/questions/24147026/display-gui-on-raspberry-pi-without-startx

```
import os
os.environ["SDL_FBDEV"] = "/dev/fb1"
```

-------Below are older instructions for SDL2 (they were designed for 32-bit Raspi OS)

SDL2

Download SDL2:

```
$ cd ~/beepberry-sdk/src/
$ wget https://github.com/libsdl-org/SDL/releases/download/release-2.28.1/
SDL2-2.28.1.tar.gz
$ tar xvf SDL2-2.28.1.tar.gz;rm SDL2-2.28.1.tar.gz
$ cd SDL2-2.28.1/
```

Configure SDL2 to enable FusionSound2 for audio and DirectFB2 for video, and set them both to load dynamically:

```
./configure --enable-fusionsound=yes --enable-fusionsound-shared=yes --enable-video-
directfb=yes --enable-directfb-shared=yes
```

Build SDL2. It takes about 15 minutes on a Raspi Zero 2.

```
make
```

Install SDL2:

```
sudo make install
```

Notes

- All build options are listed by typing ./configure --help
- I'm focusing on SDL2 accessing the framebuffer through the DirectFB2 API. SDL2 doesn't directly support the framebuffer but v1 does, according to the FAQ: https://wiki.libsdl.org/FAQUsingSDL
- Software OpenGL: https://wiki.libsdl.org/SDL2/README/directfb

SDL2-image

This is required to compile the game, "Enigma."

```
cd ~/beepberry-sdk/src/
wget https://github.com/libsdl-org/SDL_image/releases/download/release-2.6.3/
SDL2_image-2.6.3.tar.gz
tar xvf SDL2_image-2.6.3.tar.gz;rm SDL2_image-2.6.3.tar.gz
cd SDL2_image-2.6.3/
./configure
make
sudo make install
```

Libraries are installed to /usr/local/lib.

SDL2_mixer

This is required to compile the game, "Enigma."

```
cd ~/beepberry-sdk/src/
wget https://github.com/libsdl-org/SDL_mixer/releases/download/release-2.6.3/
SDL2_mixer-2.6.3.tar.gz
tar xvf SDL2_mixer-2.6.3.tar.gz;rm SDL2_mixer-2.6.3.tar.gz
cd SDL2_mixer-2.6.3/
./configure
make
sudo make install
```

Libraries are installed to /usr/local/lib.

SDL2_ttf

This is required to compile the game, "Enigma."

```
cd ~/beepberry-sdk/src/
wget https://github.com/libsdl-org/SDL_ttf/releases/download/release-2.20.2/
SDL2_ttf-2.20.2.tar.gz
tar xvf SDL2_ttf-2.20.2.tar.gz;rm SDL2_ttf-2.20.2.tar.gz
cd SDL2_ttf-2.20.2/
./autogen.sh
./configure
make
sudo make install
```

It will take a very long while to compile. First I thought it had crashed, but I retried and left it overnight, and it compiled fine. Visual Studio Code will disconnect and everything.

UPDATE: Recommend installing screen first "sudo apt install -y screen", then starting a screen session ("screen"), and then running make. If the connection dies the terminal will happily keep running in the background.

NOTE 2: Actual compile time now with the monolithic SDK build script, in screen, is ~30-45m on a Zero2. Not bad.

Libraries are installed to /usr/local/lib.

SDL2 Game: Enigma (experimental, not working)

We will test the SDL toolchain by compiling a game named Enigma. Currently Enigma doesn't build.

Xerces

This is required to compile the game, "Enigma."

```
cd ~/beepberry-sdk/src/
wget https://dlcdn.apache.org//xerces/c/3/sources/xerces-c-3.2.4.tar.bz2
tar xvf xerces-c-3.2.4.tar.bz2;rm xerces-c-3.2.4.tar.bz2
cd xerces-c-3.2.4
./configure
make
sudo make install
```

Xerces takes about 20 minutes to compile on the Raspi Zero 2.

SDL2 Game: Enigma

NOTE: Compilation currently fails - it's looking for a tool called "convert" but can't find it.

Install SDL2 dependencies: SDL2, SDL2-image, SDL2_mixer, SDL2_ttf

Install libcurl-dev for the OpenSSL version of libcurl on Raspbian:

```
sudo apt install libcurl4-openssl-dev
```

Build Enigma:

```
cd ~/beepberry-sdk/src/
wget https://github.com/Enigma-Game/Enigma/releases/download/1.30/Enigma-1.30-src.tar.gz
tar xvf Enigma-1.30-src.tar.gz;rm Enigma-1.30-src.tar.gz
cd enigma-1.30
./configure
```

Reference: HOWTO compile: https://www.nongnu.org/enigma/download.html#compilation

pygame Toolchain

Build SDL2 as above. It's a dependency.
Build sdl2-image above.
Build sdl2-mixer above.
Build sdl2-ttf above.

Install portmidi, which adds MIDI support:

```
apt install libportmidi-dev libportmidi0
```

Now that SDL2 is built and installed, download and build pygame:

```
cd ~/beepy-sdk/src
git clone --depth 1 https://github.com/pygame/pygame
cd pygame
python3 setup.py -config -auto
python3 setup.py install --user     <-- read note below - under development
```

Build time is about 10 minutes on a Raspi Zero 2.

The above being the official way to build SDL2, we will get a permissions error when trying to use pygame. Read below for why this occurs. Instead of that last "install --user" line, do this:

```
sudo python3 setup.py install
```

Note you must do this as root ('su'). If you only use sudo python, it will fail.

OpenGL issue with pygame (when done, this gets rolled into the DirectFB2 build process)

Trying to rebuild with NEON support
To the meson.build file in the DirectFB2 project, add the line:

```
-march=native, language='c'
```

then type:

```
meson configure build -Dneon=true
```

then

```
meson compile -C build
```

Running DirectFB2 as a regular user (non root) (incomplete)

We still run DirectFB as root. These are notes related to moving away from this.

Making the user a member of the "video" group is not enough.

```
$ ls -la /dev
crw-rw----   1 root video    29,   1 Mar 28 15:34 fb1
```

usermod pi -G video

```
$ python snakewm/wm.py --dfb:system=fbdev --dfb:fbdev=/dev/fb1
```

This command would display all options for your build of DirectFB2 (with the --dfb:help option)

```
pi@beepy2:~/glowfire$ python snakewm/wm.py --dfb:system=fbdev --dfb:help
```

Display details of DirectFB2's running config and detected devices:

```
$ dfbinfo
```

From https://bugzilla.redhat.com/show_bug.cgi?id=852745

"The core issue behind why dfbinfo doesn't run as a "normal" user is due to the fact that the Linux kernel requires CAP_SYS_TTY_CONFIG to do any TTY ioctl() calls. UID 0 (root) has that, but normal users do not. It is possible to give a binary that capability using the "setcap" command."

Running snakewm as a regular user (not root)

```
sudo PYTHONPATH=/home/pi/glowfire/snakewm/data/lib python /home/pi/glowfire/snakewm/wm.py
--dfb:vt-num=1 &
```

--dfb:vt-num=1 tells DirectFB to use tty1 on launch. This "pins" the GUI to the beepy LCD. Otherwise the system will briefly launch snakewm and then immediately replace the LCD image with tty2.

The ampersand is important; without it, the app segfaults on launch.

Test pygame can use DirectFB2 to draw to the Memory LCD

Set up a Python session:

```
$ sudo su
# python
>>> import os, pygame
```

At this point we may receive a warning about NEON not being built. This is OK.

```
>>> os.putenv('SDL_VIDEODRIVER', 'directfb')
>>> os.putenv('SDL_AUDIODRIVER', 'alsa')
>>> pygame.display.init()
```

In the SSH session, the DirectFB 2.0.0 banner will appear. Below the banner is a list of the exact video and input drivers DirectFB is using for this session.

On the Memory LCD screen, a black background appears, with a white pointer. The pointer will move, laggily, with the touchpad.

Print the screen resolution, bit depth, and pixel format:

```
>>> print(pygame.display.Info())
```

Draw a white rectangle on the black screen:

```
>>> lcd = pygame.display.set_mode((400,240))
>>> pygame.draw.rect(lcd,(255,255,255),(10,10,100,100))
>>> pygame.display.update()
```

Docs for DirectFB: https://directfb2.github.io/

DRMKMS/VT = Virtual Terminal

Direct/Interface: Could not open interface directory '/usr/local/lib/arm-linux-gnueabihf/directfb-2.0-0/interfaces/IDirectFBGL'

There's more compiling info here: https://www.pygame.org/wiki/MacCompile 18 but you probably need to install these dependencies in homebrew.
brew install sdl sdl_image sdl_mixer sdl_ttf portmidi

Download pygame 2.5.0 source
cd ~/beepberry-sdk/src/
wget https://github.com/pygame/pygame/archive/refs/tags/2.5.0.tar.gz
tar -xvf 2.5.0.tar.gz;rm 2.5.0.tar.gz
cd pygame-2.5.0/

Compile pygame

REFERENCE:
Building pygame on Ubuntu: https://www.pygame.org/wiki/CompileUbuntu?parent=
python gui in Linux framebuffer: http://www.karoltomala.com/blog/?p=679
pygame example with framebuffer: https://github.com/misterblack1/raspberrypi

May need to build SDL2 first... try without in next test.
Install dependencies:

```
python -m pip install -U pygame==2.5.0 --user
```

https://learn.adafruit.com/pi-video-output-using-pygame/pointing-pygame-to-the-framebuffer
http://www.karoltomala.com/blog/?p=679

https://pypi.org/project/pyglet/

Old notes
Install JACK audio system:

```
sudo apt install jackd
```

When prompted, allow jackd to run in realtime.

Install base dependencies: JACK audio headers, Python bindings for JACK, Cython, and F(rame)B(uffer)PY.

```
pip install Cython
sudo apt install -y libjack-dev
```

Manually fix a bug with the py-jack install.py script by doing this workaround:

```
sudo ln -s /usr/include/jack/jack.h /usr/local/include/jack/jack.h
```

Install more advanced dependencies which use the base dependencies:

```
pip install py-jack
pip install fbpy
```

Troubleshooting the pygame/SDL2 graphics stack

Symptom: DirectFB2 apps work, but SDL2 doesn't start DirectFB2.

Run dfbinfo. This initializes DirectFB:

```
# dfbinfo
```

Test that a DirectFB2 app works

Enter the below. On the SSH session, it should display the DirectFB2 banner. On the beepy, it should draw a progress bar, then quit.

```
# /usr/local/bin/lite_progressbar
```

```
os.putenv('SDL_VIDEODRIVER', 'directfb')
>>> os.putenv('SDL_AUDIODRIVER', 'alsa')
>>> pygame.display.init()
```

DirectFB2 libs are installed in /usr/local/lib/aarch64-linux-gnu/
e.g. libdirect.so, libdirectfb.so, and so on.

SDL2 libs are installed in /usr/local/lib/
libSDL2_image.a <- static, 1.9MB
libSDL2_image.so <- shared, 1.2MB

The shared one is versioned. That filename actually points to the versioned one. Listing the directory thusly should give you enough info:

```
$ ls -lath /usr/local/lib/
```

These are the library names in question:
libSDL2.so - 8.4M
libSDL2_mixer.so - 2.0M
libSDL2_image.so - 1.2M
libSDL2_ttf.so - 0.3M
libLC3plus.so

Check that SDL2 contains a reference to DirectFB:

```
readelf -d /usr/local/lib/libSDL2-2.0.so.0.2800.2
```

It should show libdirectfb among the results.

Ideas for applications

pygame MP3 player

python bindings for miniaudio.
miniaudio supports FLAC. https://pypi.org/project/miniaudio/
TODO: AAC format.
AAC decoder: https://github.com/mstorsjo/fdk-aac

audio-metadata - MIT licensed id3 parser

https://github.com/thebigmunch/audio-metadata
Requires tbm_utils:
https://github.com/thebigmunch/tbm-utils
Put audio-metadata/src/audio-metadata directory into lib
Install dependencies:
pip install bidict bitstruct more_itertools pendulum tbm_utils

id3py - MIT licensed id3 parser

https://github.com/aoirint/id3py

id3reader - Simplified BSD license
https://github.com/teragonaudio/id3reader

librosa - audio visualization

https://librosa.org/doc/latest/install.html
https://github.com/librosa/librosa

Mod tracker

https://pypi.org/project/libxmplite/
https://www.reddit.com/r/Python/comments/caee0w/ive_created_a_python_mod_player_it_looks_amazing/

https://formats.kaitai.io/fasttracker_xm_module/python.html

Building MOD support into SDL2

sdl_mixer lacks these 3 libraries thus far:

* Opus Codec (opus-codec.org)
  opus/opusfile.h
* **ModPlug library (http://modplug-xmms.sourceforge.net/)**
  Adds MOD audio file support. (core is "arguably the best-rendering MOD playback engine")
  libmodplug/modplug.h

* FluidSynth (fluidsynth.org)
  fluidsynth.h

Terminal emulator (plan: Quake style dropdown)

https://pypi.org/project/pygconsole/

Digital voice over LoRa

QMesh: a LoRA-based voice mesh network

https://hackaday.io/project/161491-qmesh-a-lora-based-voice-mesh-network

M17 Project

https://m17project.org/
https://www.reddit.com/r/amateurradio/comments/lgl1gf/m17_a_new_digital_voice_and_packet_mode_for/
FreeDV is their free digital voice setup.
https://freedv.org/

Codecs

Lyra 2 (Google, Apache license) - higher qual than opus at half the bandwidth
Opus Codec https://opus-codec.org/
Codec 2: "The main application is low bandwidth HF/VHF digital radio."
libcodec2 is here: https://github.com/MonadicLabs/libcodec2

Building m17-tools on the Mac

brew install codec2 rtaudio
-> rtaudio doesn't seem to be detected
make
make test

pyxel - a retro game engine for python

Do not install from a package via pip. It needs to be built against our hand-installed SDL2.

Install rustup.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Download pyxel.

```
git clone https://github.com/kitao/pyxel
cd pyxel
pip3 install -r requirements.txt
```

pybuilder - organize the build process

https://pybuilder.io/

Bluetooth headphones

Goals

Any ALSA app can play through BT headphones
Two-step pair or repair.

CLI Reference

bluealsa >
bluetoothctl > devices : list devices
> scan on : start scan
> scan off : stop scan

References

Good primer on Bluetooth audio: "Audio over Bluetooth: most detailed information about profiles, codecs, and devices."

"How to stream sound to a Bluetooth device from a Raspberry Pi Zero," stackexchange.com

Toolchain build process

See demo-pygame/xp-btaudio/build-btaudio.py for notes. This is a Python file with pseudocode that won't actually run, but contains information on the build process.

Install prerequisites

Install optional prerequisites

Install ALSA

Test 1/3: is daemon running?

Test 2/3: can connect to bt headphones? (or bt speakers)

Test 3/3: can send arbitrary audio streams from any alsa app, to bt headphones?

Build and install toolchain

See demo-pygame/xp-btaudio/build-btaudio.py for notes. This is a Python file with pseudocode that won't actually run, but contains information on the build process.

Run example

Process to pair and connect

Note: If debugging audio issues. On every device connected to the headphones, "forget" those headphones from every system, except the beepy.

- Start bluealsa daemon, to connect bluez to alsa.
- Run bluetoothctl (front-end for bluez), then scan, pair, trust, and open a connection to the headphones.
- Test audio playback from ALSA with a .wav file and aplay -D <bluealsa PCM name>
- Test re-connection: Turn headphone power off, then on. Ctrl-z bluealsa, re-start bluealsa daemon, switch to bluetoothctl window, connect <MAC address of bluetooth device>. Play .wav file from aplay as above.

Decker

A multimedia sketchpad. Designed for black and white, in a similar screen resolution.

MIT license.

Works on the beepy, provided SDL2 is installed.

The only things not working — so far — are that it expects fullscreen to be 512x342 to accommodate the respective cardsize, and it does not display the mouse pointer.

Resizing to fit the beepy display

In js/lil.js, line 2361:

```
ri.size        ='size'    in deck?
rclamp(rect(320,240),getpair(deck.size),rect(4096,4096)):rect(512,342)
```

Those last rect constants, 512x342, represent the original HyperCard cardsize.

Once changed to 400x240, Decker launches fine, with the deck cropped on the right and bottom.

Mouse is unusable on it due to no mouse cursor.

- Note the .deck file resolution. Corners auto-round, and card starts at 0,0.
- No mousebutton support. Verified left and right mousebuttons work in DirectFB. See decker.c > line ~3196, when it processes SDL events. This may be related to the pointer issue. It may be that the mouse is there, but SDL thinks its absolute coordinates are way off. The DirectFB cursor does not reflect the SDL coordinates - it properly clips to the screen bounds.
- It does not display the mouse cursor ("pointer") because we are not rendering within a traditional WM environment. One could hack in a mouse pointer within Decker itself, like I did with snakeware, but this is not a good solution, and would not meet QA for Decker upstream. For the beepy, a mouse cursor implemented in a DirectFB tslib "wrapper" passed off as system mouse, might be simplest, overall, to compile over the mainline Decker release. DirectFB also provides a minimal window manager, which may help with some of the issue.

GUI Toolkit Comparison

This is for GUI toolkits (predefined buttons, sliders, text boxes, etc.)
It is not a comparison of lower level graphics libraries like pygame and SDL2.

Name	Disadvantages	Needs this graphics engine	Language	Monochrome theme built-in?	Comments
LVGL	No	Framebuffer	MicroPython	Yes	Designed for microcontrollers and MicroPython. Can run on Linux.
PySide6	Very complex and long compile process	pygame	Python		Python bindings for Qt toolkit.
Thorpy	No	pygame	Python	No	Runs, but needs its assets and drawing routines converted to monochrome
Nuklear		SDL2	C		
raylib					
Tkinter	Needs X11				
Dear ImGui	Needs OpenGL				Widely used but needs OpenGL
SDL_gui					
Kivy	Needs OpenGL		Python		
wxPython					

App: qtwebkit (incomplete)

Install necessary dependencies:

```
sudo apt install -y bison gperf ruby sqlite3 libsqlite3-dev libharfbuzz-dev libxml2-dev
libxslt1-dev dwz
```

- see if you can omit sqlite3
- -- WARNING: dwz may use a lot of RAM - build with -DSKIP_DWZ=ON if you don't have enough

Download qtpositioning-5.15.2 source:

```
cd ~/beepy-directfb2/src
git clone --depth 1 -b 5.15.2 https://github.com/qt/qtpositioning/
cd qtpositioning
qmake
make
```

NOTE : Add the gypsy GPS daemon and a GPS receiver to test out positioning information.

Install qt5 from package:

```
sudo apt install -y qtbase5-dev
```

Download qtwebkit source:

```
cd ~/beepy-directfb2/src
git clone --depth 1 https://github.com/movableink/webkit
```

Configure qtwebkit:

```
cd ~/beepy-directfb2/src/webkit
mkdir build
cd build
cmake -G Ninja -DPORT=Qt -DCMAKE_BUILD_TYPE=Release ..
ninja
sudo ninja install
```

May need to: set QT_QPA_PLATFORM to 'directfb'.

References

Building from Git: https://wiki.qt.io/Building_Qt_5_from_Git

Building on Linux: https://github.com/qtwebkit/qtwebkit/wiki/Building-QtWebKit-on-Linux

Wiki: https://github.com/qtwebkit/qtwebkit/wiki

raylib Toolkit

These instructions are courtesy of Steak, posted on the beepy discord on 2024-05-15.

Steps for getting raylib working on Raspberry Pi OS Lite 64-bit:

Edit boot.txt with system running:

```
$ sudo vi /boot/firmware/config.txt
```

Disable DRM VC4 V3D driver. Comment out these lines to disable the HDMI output:

```
# Enable DRM VC4 V3D driver
#dtoverlay=vc4-kms-v3d
#max_framebuffers=2
```

Reboot and verify only SPI-1 DRM connector is present:

```
$ for p in /sys/class/drm/*/status; do con=${p%/status};
echo -n "${con#*/card?-}: "; cat $p; done
```

raylib running cyberpunk demo.
Image courtesy of Steak.

Expected result should be only **"SPI-1: connected"** displayed.

Install dependencies:

```
$ sudo apt install -y build-essential git
$ sudo apt install -y libasound2-dev libx11-dev libxrandr-dev libxi-dev libgl1-mesa-dev
libglu1-mesa-dev libxcursor-dev libxinerama-dev libwayland-dev  libxkbcommon-dev libdrm-
dev libegl1-mesa-dev libgles2-mesa-dev libgbm-dev imagemagick
```

Clone raylib:

```
$ git clone https://github.com/raysan5/raylib.git ~/raylib
```

Compile raylib with DRM mode and static library link. It generates a static library at src/libraylib.a.

```
$ cd ~/raylib/src/
$ make PLATFORM=PLATFORM_DRM
```

Pick one specific example to test. Let's use examples/textures/textures_background_scrolling as an example.

Open the C source file.

```
$ vi ~/raylib/examples/textures/textures_background_scrolling.c
```

Modify the resolution value to match the Memory LCD resolution:

```
// Initialization
// --------------------------
const int screenWidth = 400;
const int screenHeight = 240;
```

For the images to display correctly, you need to convert them to monochrome dithered with Imagemagick:

```
$ cd ~/raylib/examples/textures/resources
$ convert cyberpunk_street_background.png -colorspace Gray -ordered-dither o2x2
cyberpunk_street_background.png

$ convert cyberpunk_street_foreground.png -colorspace Gray -ordered-dither o2x2
cyberpunk_street_foreground.png

$ convert cyberpunk_street_midground.png -colorspace Gray -ordered-dither o2x2
cyberpunk_street_midground.png
```

Build all examples in the textures module.

```
$ cd ~/raylib/examples/
$ make textures PLATFORM=PLATFORM_DRM
```

Run examples from inside the textures directory. You may encounter missing resource errors if you run examples from another directory.

Run the example.

```
$ cd ~/raylib/examples/textures/
$ ./textures_background_scrolling
```

Tkinter Toolkit (incomplete)

Install Tkinter
sudo apt install -y python-tk

Try Tkinter
python2
>>> import Tkinter
>>> Tkinter._test()

---
pip install pygame

Notes
python3-pygame/stable 1.9.6+dfsg-4+b1 armhf
  SDL bindings for games development (Python 3)

python3-pygame-sdl2/stable 7.4.2-1 armhf
  reimplementation of the Pygame API using SDL2

python3-pgzero/stable 1.2.post4+dfsg-2 all
  Zero-boilerplate games programming framework based on Pygame (Python 3)

References
"Python GUI in Linux framebuffer," http://www.karoltomala.com/blog/?p=679

Dear ImGui (incomplete)

It needs OpenGL to run.
Powerful UI framework, widely used.

Web demo to see what it can do: https://jnmaloney.github.io/WebGui/imgui.html

There is an SDL binding discussion here:
https://www.reddit.com/r/cpp_questions/comments/mne3v3/using_dear_imgui_with_sdl2/

The SDL binding is here:
https://github.com/ocornut/imgui/blob/92d0924b82dcc1c7159977d29a9c5044ff85459d/examples/example_sdl_sdlrenderer/main.cpp

Reference: https://github.com/jnmaloney/WebGui

Styles library: https://github.com/GraphicsProgramming/dear-imgui-styles

Increase swap to at least 1000MB before installing pimgu.

Install pimgu:

```
pip install pimgu
```

It will take a very long time - 45 minutes to an hour.

Link:
https://github.com/pyimgui/pyimgui

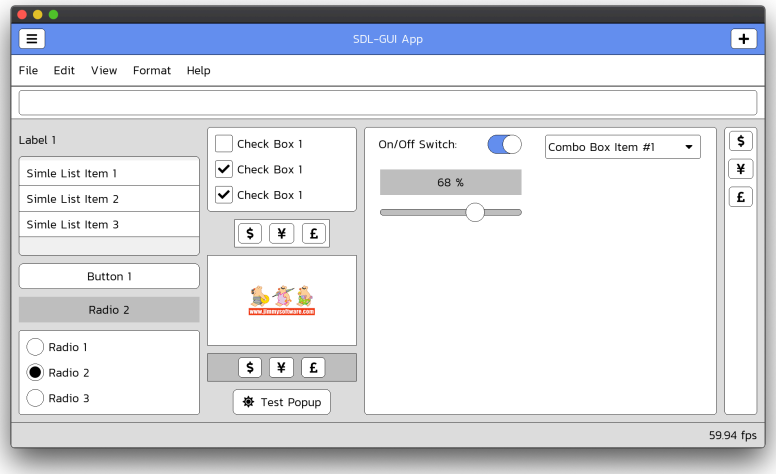
pip install imgui[sdl2]
https://github.com/ocornut/imgui/issues/2822

SDL2-Based GUI Toolkits

SDL_gui: GUI library for SDL2

https://github.com/mozeal/SDL_gui

MIT License.

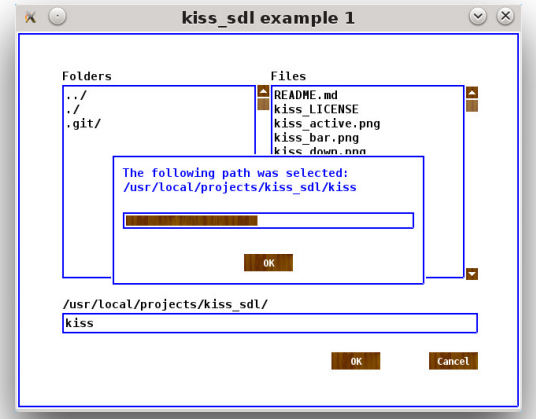


kiss_sdl: Simple generic widget toolkit for SDL2

Repository: https://github.com/actsl/kiss_sdl

Manual: https://github.com/actsl/kiss_sdl/blob/master/kiss_manual.pdf

License is permissive. See kiss_LICENSE in repo.



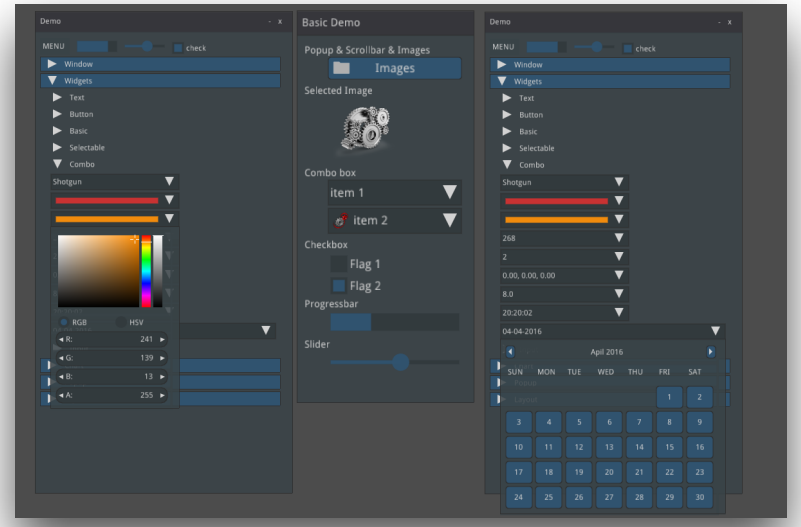
Nuklear

https://github.com/Immediate-Mode-UI/Nuklear

Bindings are listed in the "Bindings" section.

There are 2 Python bindings (pyNuklear, pynk).

Dual licensed: MIT or Public Domain (unlicense.org)



GHOST (Generic Handy Operating System Toolkit)

https://github.com/Modaptix/ghost

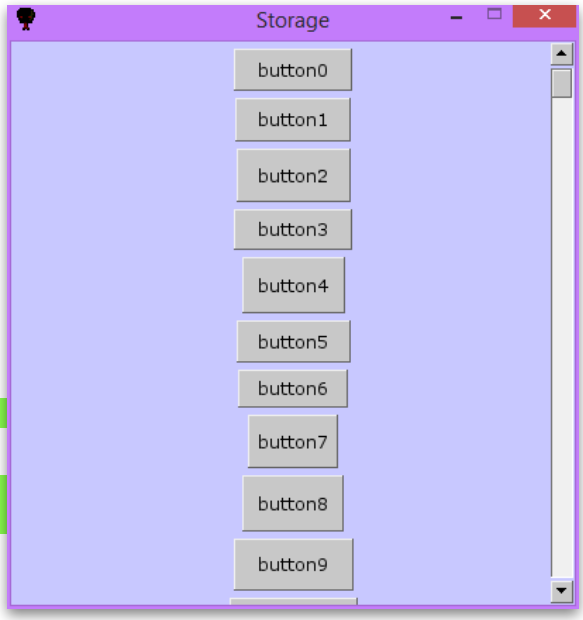
Thorpy

"Thorpy is a non-intrusive, straightforward GUI kit for Pygame."

Documentation

Example tutorial: "Storage": storing and placing Thorpy elements

Web: https://www.thorpy.org/



Thorpy

Thorpy works, but doesn't have monochrome graphics. You would need to modify the source.

Set up dev folder hierarchy:

```
mkdir ~/thorpy
```

Download Thorpy source

```
cd ~/thorpy
git clone https://github.com/YannThorimbert/Thorpy2
```

export DFBARGS="system=fbdev,fbdev=/dev/fb1,graphics-vt"

LD_LIBRARY_PATH=/usr/local/lib/arm-linux-gnueabihf/;export LD_LIBRARY_PATH

Creating a monochrome theme for Thorpy

Thorpy's themes are specified as overlays on top of more generic themes.

The process has been thus far:

- search through the source code and find where the themes are specified
- step through the code, and keep notes on which themes inherit what properties
- use this to create a new theme, which overrides all color-specific properties with its own
- take note of how it does alpha blending, shadows, etc. and ensure things are monochrome-friendly

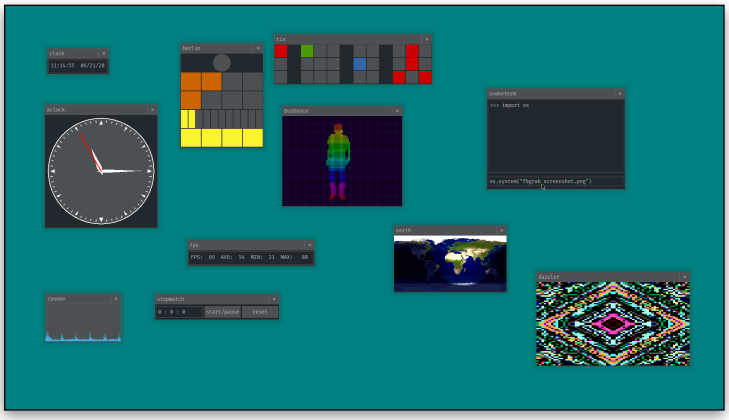
The hacky way is to simply go through the theme code and change all colors from whatever they are to (0,0,0) or (255,255,255). This worked as a proof of concept.

snakeware: a python userland

Python desktop environment, demonstrating multiple open windows. It has a window manager, 'snakewm.'

Web: https://github.com/joshiemoore/snakeware

git clone https://github.com/joshiemoore/snakeware
cd snakeware/snakewm
sudo pip install pygame_gui
sudo python wm.py



snakewm with example apps

Getting snakeware to run on beepy

Note: snakeware depends on pygame-gui, which itself needs pygame-ce. (Not pygame.)

pygame_gui theming
https://pygame-gui.readthedocs.io/en/latest/theme_guide.html#theme-options-per-element

Using pygame-gui in snakeware

Reference: theming windows (shadows, etc.):
UIWindow Theming Parameters (pygame-gui.readthedocs.io)
ui_container.py (pygame-gui source code). This explains relative_rect and other pygame-gui attributes.
ui_panel.py ("" "")

Current issues with snakeware

DirectFB2 mouse cursor info

Cursor located in source code at: DirectFB2/src/core/cursor.h
Description of same, src/core/windowstack.c, uses it in memcpy(), examine lines before to see the "hot" pixel and the cursor size (40x40).
Another reference here: https://github.com/dbt1/plugin-links/blob/master/directfb_cursors.h

DirectFB mouse cursor will not go negative (it will only go to 0,0 which is top left corner).

Mouse cursor mismatch issue (affects Decker as well)

SDL2 app pointer click target doesn't match floating DirectFB pointer.
sudo apt install gpm

sudo gpm -m /dev/input/mouse1 -t imps2

Testing:
sudo apt install evtest

Recompiling SDL2 tests:
input drivers: linuxev linuxkd

Reference:
https://wiki.archlinux.org/title/General_purpose_mouse

Attempting to run DirectFB with different options

sudo /usr/local/bin/dfbinfo --dfb:no-smooth-upscale --dfb:no-smooth-downscale --dfb:no-single-window --dfb:no-translucent-windows --dfb:cursor-videoonly --dfb:cursor

Examine source:
SDL/src/video/directfb/SDL_DirectFB_mouse.c

https://github.com/DirectFB/directfb#readme

Theme reference: https://pygame-gui.readthedocs.io/en/latest/theme_reference/theme_button.html

Patching pygame-gui for the cursor issue (in progress)

Modify the pygame-gui UI Manager so that it does all the mouse handling itself. This is a workaround for the SDL cursor not matching DirectFB's cursor location.
The better way would be to fix it lower in the system, probably with a touch driver.

See pygame-gui code here:
/usr/local/lib/python3.9/dist-packages/pygame_gui-0.6.10-py3.9.egg/pygame_gui/ui_manager.py
line 560, "_update mouse_position" definition. This wraps the function. If we put our logic here, then we should be able to patch for the Memory LCD.
Patch these functions:
 * get_mouse_position
 * update_mouse_position
 * calculate_scaled_mouse_position
 * load_default_cursors -> load premade bitmaps (the call to "system" cursors will fail)
 * set_active_cursor
 * preload_fonts (optional - change to more appropriate one for this screen)

patching update_mouse_position()
        pygame.mouse.get_pos()

pygame source:
/usr/local/lib/python3.9/dist-packages/pygame-2.6.0.dev1-py3.9-linux-aarch64.e
gg/pygame/cursors.py

pygame cursor diagnostics

- it reports the cursor it's using is system-type, and that it is a regular arrow. see here:
- https://www.pygame.org/docs/ref/cursors.html#pygame.cursors.Cursor

It cannot be changed to type hand, for example. Understanding the interface between DirectFB and SDL2 with regards to cursor 'negotiation' will help.

Probably could workaround with setting bitmap type cursor in pygame, but this doesn't help other SDL2 apps (e.g. Decker, which needs a mouse cursor to interact). (UPDATE: Tried this. pygame/SDL cannot set the cursor, resulting in an error. It seems obvious in hindsight that SDL2 cannot set the DirectFB2 cursor. But even with the DirectFB cursor disabled, and things set up so that I do have an x,y hotspot (but no cursor image), I can control the UI, but still, in this setup, I have not been able to set the cursor with SDL. So whatever it's trying to reach on the system side isn't working.)

Reported issue with SDL here: https://discourse.libsdl.org/t/sdl-directfb-input-events/15411

For archive, the patch they posted:

```
--- start patch ---
81a82,88
>           else if (evt.flags & DIEF_AXISABS)
>             {
>               if (evt.axis == DIAI_X)
>                 posted += SDL_PrivateMouseButton(0, 0, evt.axisabs, 0);
>               else if (evt.axis == DIAI_Y)
>                 posted += SDL_PrivateMouseButton(0, 0, 0, evt.axisabs);
>             }
--- end patch ---
```

See SDL/src/video/directfb/SDL_DirectFB_events.c, line 325.

I've changed it to this to work with DFB2 (posted within untouched text):

```
case DIET_AXISMOTION:
            if (ievt->flags & DIEF_AXISABS) {
                if (ievt->axis == DIAI_X)
                    //last_x = ievt->axisabs;
                    last_x += SDL_PrivateMouseButton(0,0,ievt.axisabs,0);
                else if (ievt->axis == DIAI_Y)
                    //last_y = ievt->axisabs;
                    last_y += SDL_PrivateMouseButton(0,0,0,ievt.axisabs);
                if (!(ievt->flags & DIEF_FOLLOW)) {
```

DirectFB cursor diagnostics

Running /usr/local/bin/df_input, I still see negative mouse values. The x,y "crossing point" can easily be moved way off the screen.

Find out how to limit mouse movement to physical screen resolution.

See possibly **XDirectFB**.

Several ways to fix cursor issues:

- blit a cursor, manually, at the x,y pos, and clip x,y to bounds of resolution ((0,0),(400,240))
- set up DirectFB2 as a window manager, have it (possibly?) handle the above, running SDL2 as a window in full screen res - this way DirectFB2 handles and draws its cursor. Problem with this is SDL2 seems to be getting raw input from Linux Input, and the abs coords just keep going in either direction, based on relative mouse movement, and conceivably could overflow. So clipping would help prevent this condition.
- Use uinput to simulate a device, and use that instead. This would also be a good way to do capacitive touchscreen and possibly multitouch.
- Use the touchscreen driver instead of mouse driver

Disable font antialiasing (Fixed - requires patch to pygame-gui)

Symptom: fonts on the Memory LCD look bad when rendered using FreeType.

edit source: pygame_gui/pygame_gui/core/gui_font_freetype.py, line

Note the render function takes an antialiasing argument: https://www.pygame.org/docs/ref/freetype.html#pygame.freetype.Font.antialiased

Interesting note from stackoverflow: "One thing about blitting a custom cursor to the screen of every frame is that its position will be affected by the frame rate however small that delay is."

Build pygame-gui expressly for Memory LCD

In this step, we patch pygame_gui so that it never antialiases fonts. This is sort of janky, but works.

Prepare by uninstalling all vestiges of packaged pygame_gui (if you previously installed it):
sudo pip3 uninstall pygame_gui
pip3 uninstall pygame_gui

git clone https://github.com/MyreMylar/pygame_gui/
cd pygame_gui
git checkout 19819bcd45099dfe7362b877550764fabe63f7af
(make the change to the gui_font_freetype.py file)
sudo python setup.py install

Troubleshooting snakeware issues

Python stays running, preventing me from killing and restarting snakewm

```
ps –A | grep python | awk '{print $1;}'
```

will output the process number in question. From there a kill -9 should do it:
kill -9 2973 (or whatever the process ID was)

pygame-gui

Needs to be running source with the line_spacing function patched. (Dec 2022). The default pip3 macOS version in 2023-11 doesn't have this. To manually setup on Mac:
Uninstall any old versions of the module named pygame_gui:
pip3 uninstall pygame_gui
Download the pygame-gui source from github:
git clone https://github.com/MyreMylar/pygame_gui
cd pygame_gui
$ python3 setup.py install

snakeware

Python desktop environment

git clone https://github.com/joshiemoore/snakeware
cd snakeware/snakewm
sudo pip install pygame_gui
sudo python wm.py

Running snakeware manually

pi@beepy1:~/glowfire$ export PYTHONPATH=/home/pi/glowfire/snakewm/data/lib/

pi@beepy1:~/glowfire$ /usr/bin/python /home/pi/glowfire/snakewm/wm.py

Autostarting snakeware at boot

Longer term, make the Raspberry Pi default user (not root, but an admin) able to control the framebuffer at /dev/fb1, as well as DirectFB, which requires a specific user setup to run as non-root. See DirectFB section.

Running snakewm as a service

While debugging systemd, you can follow the latest output:
journalctl -f -u snakewm.service

Reload from .service files:
systemctl daemon-reload

When removing systemd services, do this as cleanup:
sudo systemctl reset-failed

pygame-ce considerations

* Build from source; do not install apt version (doesn't work with our DirectFB)

Modifying Tower Defence game for beepy

This is a tower defense game written in pygame, using pygame_gui.

It would need to be resized for the beepy screen, have black and white graphics made, and possibly get sound.

Link: https://github.com/MyreMylar

It does not have a LICENSE file yet.

LVGL Toolkit (experimental)

Web: https://github.com/lvgl/lvgl

MIT License.

Embedded graphics library to create UIs. Designed for MicroPython on microcontrollers. Would be useful on the beepy's embedded RP2040, or even the Pi Zero (via the simulator).

Linux support: https://blog.lvgl.io/2018-01-03/linux_fb

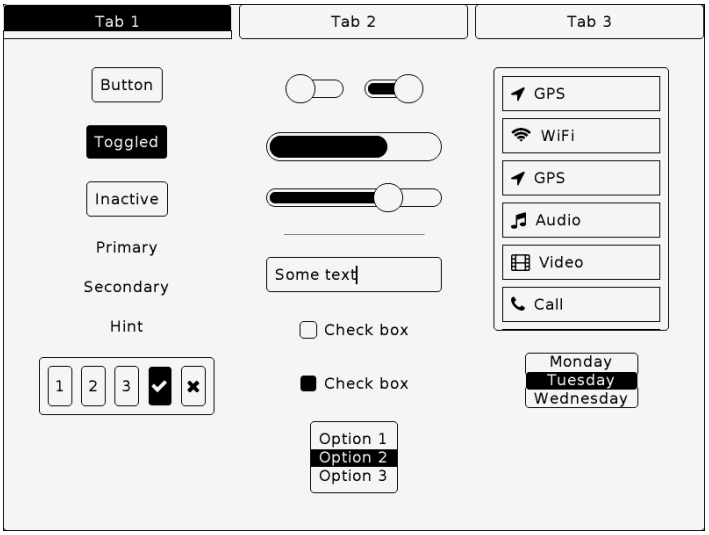
mkdir ~/lvgui

cd ~/lvgui

git clone https://github.com/lvgl/lvgl

Setup a simulator

https://docs.lvgl.io/master/get-started/platforms/pc-simulator.html



LVGL Mono Theme

Install dependencies

```
brew install sdl2
```

Install the LVGL simulator

```
git clone git clone --recursive https://github.com/lvgl/lv_sim_vscode_sdl
```

See guide here:

https://github.com/lvgl/lv_port_pc_vscode

See examples of MicroPython code here, with styles:

https://docs.lvgl.io/master/examples.html

Python/Qt (experimental)

https://doc.qt.io/qtforpython-6/

This works on desktop machines, but not the Raspi:

```
pip install pyside6
python
>>> import PySide6
```

Build Qt6

Reference: Building Qt6 on Raspberry Pi

Update the system to the latest of everything:

```
sudo apt update -y
sudo apt full-upgrade -y
sudo reboot
```

Install dependencies group 1 of 2:

```
sudo apt-get install -y libboost-all-dev libudev-dev libinput-dev libts-dev libmtdev-dev
libjpeg-dev libfontconfig1-dev libssl-dev libdbus-1-dev libglib2.0-dev libxkbcommon-dev
libegl1-mesa-dev libgbm-dev libgles2-mesa-dev mesa-common-dev libasound2-dev libpulse-dev
gstreamer1.0-omx libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev  gstreamer1.0-alsa
libvpx-dev libsrtp2-dev libsnappy-dev libnss3-dev "^libxcb.*" flex bison libxslt-dev ruby
gperf libbz2-dev libcups2-dev libatkmm-1.6-dev libxi6 libxcomposite1 libfreetype6-dev
libicu-dev libsqlite3-dev libxslt1-dev
```

Install dependencies group 2 of 2:

```
sudo apt-get install -y libavcodec-dev libavformat-dev libswscale-dev libx11-dev freetds-
dev libsqlite3-dev libpq-dev libiodbc2-dev firebird-dev libgst-dev libxext-dev libxcb1
libxcb1-dev libx11-xcb1 libx11-xcb-dev libxcb-keysyms1 libxcb-keysyms1-dev libxcb-image0
libxcb-image0-dev libxcb-shm0 libxcb-shm0-dev libxcb-icccm4 libxcb-icccm4-dev libxcb-
sync1 libxcb-sync-dev libxcb-render-util0 libxcb-render-util0-dev libxcb-xfixes0-dev
libxrender-dev libxcb-shape0-dev libxcb-randr0-dev libxcb-glx0-dev libxi-dev libdrm-dev
libxcb-xinerama0 libxcb-xinerama0-dev libatspi2.0-dev libxcursor-dev libxcomposite-dev
libxdamage-dev libxss-dev libxtst-dev libpci-dev libcap-dev libxrandr-dev libdirectfb-dev
libaudio-dev libxkbcommon-x11-dev
```

Create a destination directory for installing Qt6:

sudo mkdir /usr/local/qt6

Install build dependencies:

```
sudo apt-get install -y make build-essential libclang-dev ninja-build gcc git bison
python3 gperf pkg-config libfontconfig1-dev libfreetype6-dev libx11-dev libx11-xcb-dev
libxext-dev libxfixes-dev libxi-dev libxrender-dev libxcb1-dev libxcb-glx0-dev libxcb-
keysyms1-dev libxcb-image0-dev libxcb-shm0-dev libxcb-icccm4-dev libxcb-sync-dev libxcb-
xfixes0-dev libxcb-shape0-dev libxcb-randr0-dev libxcb-render-util0-dev libxcb-util-dev
libxcb-xinerama0-dev libxcb-xkb-dev libxkbcommon-dev libxkbcommon-x11-dev libatspi2.0-dev
libgl1-mesa-dev libglu1-mesa-dev freeglut3-dev
```

Clone qt6 (yes, this clones qt6 source code, despite it saying qt5):

cd $HOME

git clone https://github.com/qt/qt5

Build qt6:

cd ~/qt5

git checkout 6.4.0

perl init-repository -f

Build pyside

Reference: Building PySide on Linux

Create a directory for the PySide sources:

```
mkdir ~/pyside
cd ~/pyside
```

Clone from the development branch:

```
git clone git://code.qt.io/pyside/apiextractor.git
git clone git://code.qt.io/pyside/generatorrunner.git
git clone git://code.qt.io/pyside/shiboken.git
git clone git://code.qt.io/pyside/pyside.git
git -c http.sslVerify=false clone https://gitorious.org/pyside/pyside-tools.git
```

Build apiextractor:

cd ~/pyside/apiextractor

mkdir build

cd build

cmake ..

Material themes

https://qt-material.readthedocs.io/en/latest/index.html

Playing MP3s

>>> from PySide6 import QtMultimedia

Generating a waveform: https://github.com/cournape/audiolab

Charts

https://doc.qt.io/qtforpython-5/PySide2/QtCharts/index.html#module-PySide2.QtCharts

Visualizing data

https://doc.qt.io/qtforpython-5/PySide2/QtDataVisualization/index.html#module-PySide2.QtDataVisualization

Integrating online documentation

https://doc.qt.io/qtforpython-5/PySide2/QtHelp/index.html#module-PySide2.QtHelp

Networking (TCP/IP)

https://doc.qt.io/qtforpython-5/PySide2/QtNetwork/index.html#module-PySide2.QtNetwork

Concurrent processing

https://doc.qt.io/qtforpython-5/PySide2/QtConcurrent/index.html#module-PySide2.QtConcurrent

Integrating web content

https://doc.qt.io/qtforpython-5/PySide2/QtWebEngineCore/index.html#module-PySide2.QtWebEngineCore

GPUTFB Toolchain (experimental)

This is an idea for "GPU To FrameBuffer."

ARM NEON

NEON would enable 2D graphics acceleration in hardware, and signal processing, among other things. Along with GPUTFB, it would be most useful to enable.

NEON Optimized Libraries

From ARM's NEON page:

https://community.arm.com/arm-community-blogs/b/operating-systems-blog/posts/arm-neon-programming-quick-reference

- OpenMax DL
- Ne10 - accelerated AV codecs, signal processing, color space conversions. ARM open source project.

GPU output to the Memory LCD

This would let us skip the entire DirectFB stack, and use OpenGL derived interfaces like IMGUI.

Snag: an implementation for copying GPU to framebuffer

In this, we render everything on the GPU, and then send it to the framebuffer. Specifically:
Copy data from GPU output to a buffer, transform that buffer (scale, color map, dither), then copy the buffer to the framebuffer device.

This transfer is currently via software in the CPU. From the author:

> Although I've tried my best to make **snag** efficient, it still has to churn through 96,000 pixels per update and this comes with a cost. At the default target of 30fps it will consume somewhere between 10% to 20% of the processing power of a Raspberry Pi Zero depending on what's drawing to the screen.

Idea: Accelerating snag with Linux dma-buf

Snag currently pulls frames from the KMSDRM driver, using the "fake KMS mode." (Fake KMS mode is explained in the section, **"Building snag > Explanation."**)

Pros/cons to fake KMS mode vs full KMS mode? Which GPU driver do we use?

References

dma-buf: buffer sharing and synchronization (kernel.org)

Building snag

Install dependencies:

```
sudo apt install -y cmake libbsd-dev
```

Clone snag:

```
git clone https://github.com/TheMediocritist/snag
```

Build snag:

```
cd snag
mkdir build
cd build
cmake ..
make
```

Test it runs:

```
./snag
```

The device DirectFB exposes is /dev/fb1.

vi /boot/config.txt
Change the line:
dtoverlay=vc4-kms-v3d
to:
dtoverlay=vc4-fkms-v3d

Explanation

vc4-fkms-v3d is "fake KMS mode."

Fake KMS Mode: The kernel selects the screen mode(s) for the screen(s), and the VideoCore firmware manages the video output path.

Legacy Mode: The VideoCore firmware manages everything.

Full KMS Mode: The kernel manages everything.

From reference:

"*v3d is the videocore 3D driver for the standard mesa/linux 3D stack that runs on the CPU, in contrast to the legacy 3D driver which runs on the VPU and avoids all the usual linux mechanisms.*

I presume vc4 is in the name, because when the overlay was named it was the only version of videocore that the Pi had. Even though the overlay is used for both vc4 and vc6 nowadays."

References

https://stackoverflow.com/questions/7888277/directfb-data-from-memory-buffer

References

"STICKY: All about accelerated video on the raspi," https://forums.raspberrypi.com/viewtopic.php?t=317511

xf86-video-fbturbo: 2D acceleration on the pi : https://github.com/ssvb/xf86-video-fbturbo

Linux kernel dma-buf: https://www.kernel.org/doc/html/latest/driver-api/dma-buf.html

Linux kernel drm/vc4: Broadcom VC4 graphics driver. The VC4 is used in earlier Raspis. https://www.kernel.org/doc/html/latest/gpu/vc4.html

Wikipedia on DRM KMS: https://en.wikipedia.org/wiki/Direct_Rendering_Manager#Kernel_Mode_Setting

Running X on top of DirectFB

Because why not?
This is sort of the opposite approach to the above. Instead of graphically accelerating video, using the raspi GPU, we are making a very portable X implementation, one that will run on anything with a framebuffer.

References

"DirectFB can host XDirectFB, a rootless X server implementation that uses DirectFB windows for X11 top-level windows. XDirectFB is an interface that mimics the X11 interface through the DirectFB API to simplify running applications written for X11 on DirectFB.[4]"
https://web.archive.org/web/20141018073321/http://directfb.org/wiki/index.php/Configuring_DirectFB

See this thread for detailed info on building XDirectFB:
https://www.linuxquestions.org/questions/showthread.php?threadid=31396&highlight=xdirectfb

https://askubuntu.com/questions/1345561/how-can-i-get-absolute-touchpad-coordinates
Try CaptureMouse: https://github.com/libsdl-org/SDL/blob/main/src/events/SDL_mouse_c.h
See line 213: /* Create a dummy mouse cursor for video backends that don't support true cursors,
* so that mouse grab and focus functionality will work.
SDL_SetMouseFocus(SDL_Window *window)
https://www.baeldung.com/linux/mouse-events-input-event-interface

beepy-buildroot (experimental)

This is a tuned, embedded version of Linux. It appears to build with NEON support, based on the config files.

Process on a Mac host (incomplete)

- Build a Debian/LXQT machine in UTM (arm64/bullseye). Use Apple experimental virtualization. Disk must have at least 10GB for the base system, plus at least 10GB for buildroot. I allocate 8GB RAM and 4 cores, though both can probably be left at their defaults.
- # apt install -y git curl build-essential rsync
- $ git clone https://github.com/ardangelo/beepberry-buildroot
- cd beepberry-buildroot
- date;./build-image.sh;date (to see how long it takes)

Build time: 50 minutes on Apple M2

Build error (blocking)

Go is supposed to bootstrap itself on the local system, but it doesn't understand the system arch name "aarch64." It is expecting "arm64."

There's a note:

https://lore.kernel.org/buildroot/20220725011322.1301684-1-christian@paral.in/

Install Go compiler binaries for bootstrap:

https://go.dev/doc/install/source

Process on a Debian virtual machine (incomplete)

Install prerequisites

apt install -y git curl build-essential rsync unzip bc

Clone buildroot, switch to a new branch, checkout the new branch

$ git clone https://github.com/ardangelo/beepberry-buildroot

- cd beepberry-buildroot
- git branch hackshack;git switch hackshack
- date;./build-image.sh;date (to see how long it takes)

DietPi (experimental)

DietPi is an alternate distro for the Raspberry Pi hardware. It is tuned to specific Pi models, and would theoretically allow DirectFB2 to be recompiled with NEON support. (This would act as a sort of 2D hardware acceleration for the Memory LCD.)

This guide assumes the Raspberry Pi Zero 2 W (not the Zero 1).

Download DietPi image for the "Raspberry Pi 2/3/4" from https://dietpi.com. Click the first link ("Download image:" which should give you a Debian Bookworm image.
Direct download: https://dietpi.com/downloads/images/DietPi_RPi-ARMv8-Bookworm.7z

Manually extract the DietPi 7z image (Raspi Imager doesn't support auto-extracting 7z files).

Launch Raspberry Pi Imager.
Click "Operating System > Choose OS" button.
A list appears. Scroll to the bottom and click "Use custom". Browse to the "DietPi_RPi-ARMv8-Bookworm" directory and select the file, "DietPi_RPi-ARMv8-Bookworm.img".
Insert microSD card. Click "Storage > Choose Storage", and select it in the list.

- Do not click the gear button. We will configure everything in a text file.
- Click the WRITE button. Agree, then enter your password if prompted.

It takes about 30 seconds on a modern Class 10 MicroSD.

When complete, click the DONE button, but don't remove the microSD card yet.

Configure dietpy.txt

```
vi /Volumes/NO\ NAME/dietpi.txt
```

Set these lines:

```
AUTO_SETUP_KEYBOARD_LAYOUT=us
AUTO_SETUP_TIMEZONE=America/Pacific
AUTO_SETUP_NET_ETHERNET_ENABLED=0
AUTO_SETUP_NET_WIFI_ENABLED=1
AUTO_SETUP_NET_WIFI_COUNTRY_CODE=US
AUTO_SETUP_NET_HOSTNAME=dietbeep
AUTO_SETUP_BOOT_WAIT_FOR_NETWORK=0
AUTO_SETUP_HEADLESS=1        # may cause framebuffer issues - try it!
AUTO_SETUP_BROWSER_INDEX=0
AUTO_SETUP_AUTOSTART_TARGET_INDEX=7    # option 17 runs a custom script "as login"
SURVEY_OPTED_IN=0
CONFIG_CHECK_DIETPI_UPDATES=0
CONFIG_CHECK_APT_UPDATES=0
```

For later:
#AUTO_SETUP_SSH_PUBKEY=
CONFIG_LCDPANEL=none

On boot:
Apparently it runs SSH and a VNC server at boot.

Enable the Linux Ethernet Gadget (emulates Ethernet over USB)

```
$ vi /Volumes/NO\ NAME/config.txt
```

Hit G to jump to the bottom of the file.
Hit o (lowercase) to add a new line at the bottom of the file:

```
dtoverlay=dwc2
```

Hit Esc, then type :wq to save.

```
vi /Volumes/NO\ NAME/cmdline.txt
```

After rootwait, add (on the same line):

```
modules-load=dwc2,g_ether
```

Hit Esc, then type :wq to save.

Eject the microSD card, put it in the beepy, and power on.
Currently the device does not respond to pings.

Disable boot wait

8: Network Options: Misc > Boot wait for network > disable

Configure USB DAC for ALSA

* Install ALSA from setup menu
* Select a sound card > browse to bottom > under Auto detection, choose usb-dac

Enable autologin

9: AutoStart options > Set to autologin as root

SSH into the Pi.
It immediately runs DietPi-Update.

Cross-compiling the SDK (experimental)

Download Raspi OS image (32-bit Lite):
https://downloads.raspberrypi.org/raspios_lite_armhf/images/raspios_lite_armhf-2023-05-03/2023-05-03-raspios-bullseye-armhf-lite.img.xz

Open UTM
Emulate a Raspi2 (it doesn't emulate v3 or v4 hardware)

Emulating Raspberry Pi hardware on an M2 Mac (experimental)

See qemu 8.1.0 build instructions here:
https://www.qemu.org/download/

Install dependencies:

```
brew install libssh
```

download and unzip
cd qemu-8.1.0
./configure

Microphone Access

https://electronics.stackexchange.com/questions/649355/esp8266-analogread-microphone-input-into-playable-audio

Using RP2040 PIO to implement an i2s device:
https://arduino-pico.readthedocs.io/en/latest/i2s.html

Sampling mic input at 10kHz: youtube

Generating PWM audio output from the RP2040 using the PWMAudio library:
https://arduino-pico.readthedocs.io/en/latest/pwm.html

zram: compressed memory for linux

compressed memory for linux.
claimed to reduce wear on microSD card, and improve performance.

zram-swap: a setter-upper for zram. Works well.
https://github.com/foundObjects/zram-swap

Article on setting up zram with the raspi:
https://pimylifeup.com/raspberry-pi-zram/

Check if zram is enabled:
sudo cat /proc/swaps

D-Bus control of bluetooth

Download dependencies:
pip install jeepney

Run to setup symlink in ls /etc/systemd/system:
systemctl enable bluetooth.service

Get status of bluetooth service:
systemctl status dbus-org.bluez.service
Parse for:
"Loaded: loaded"
"Status: "Running""

Interesting bluetooth config file here:
/etc/bluetooth/main.conf

python
import jeepney

Examine D-Bus from command line

See https://www.raspberrypi-bluetooth.com/bluez-5_5-and-dbus.html
See Punchthrough's article, "Creating a BLE Peripheral from BlueZ."
Relevant excerpts, modified for our use:

https://punchthrough.com/creating-a-ble-peripheral-with-bluez/

Monitor D-Bus messages going in and out of the BlueZ daemon

Open a separate terminal window, and run:

```
sudo dbus-monitor --system "destination='org.bluez'" "sender='org.bluez'"
```

Examine D-Bus from Python with dbus-fast

Web: dbus-fast
import dbus-fast

References

D-Bus and BlueZ: https://ukbaz.github.io/howto/python_gio_1.html
D-Bus interaction from the command line (gist): https://gist.github.com/ukBaz/d7cd0c4b9e7078c89980a3db2bbad98b
Playing BlueZ on the D-Bus: https://www.landley.net/kdocs/ols/2006/ols2006v1-pages-421-426.pdf
Creating a BLE peripheral with BlueZ: https://punchthrough.com/creating-a-ble-peripheral-with-bluez/

Python to Bluetooth interfacing notes

BLE:
https://pypi.org/project/bleak/

Interesting Bluetooth mesh project (active):
https://github.com/SilvairGit/python-bluetooth-mesh
License: GPL 2.0

Bluetooth in RPIOS Lite
https://www.sigmdel.ca/michel/ha/rpi/bluetooth_in_rpios_01_en.html

Setup custom bitmap fonts for kernel (experimental)

https://unix.stackexchange.com/questions/161890/how-can-i-make-a-psf-font-for-the-console-from-a-otf-one

Converter tool here:
git clone https://github.com/jirutka/otf2bdf

Convert ProFont into Linux console font bitmap format (PSF, PC Screen Font)

Look at this tool: https://github.com/talamus/rw-psf

Configure kernel build to build in ProFont

Documentation on pi kernel build here: https://www.raspberrypi.com/documentation/computers/linux_kernel.html

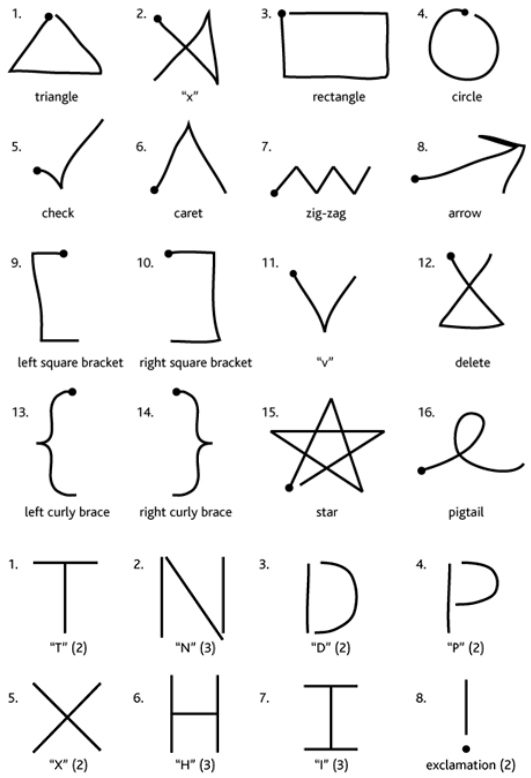
Handwritten stroke recognition

Unistroke Recognizer, https://depts.washington.edu/acelab/proj/dollar/index.html

The name OneDollar" ($1) is one of their recognizers, not the price.
It is New BSD licensed.
Site has live demo in Javascript.

Source code available for C# and Java. It is well documented.

See their "Our Gesture Software Projects" section for other variants of their recognizer technology.



Samples of templates from the Unistroke Recognizer.

FAQ

How do I determine in which mode my trackpad is set?

There are 2 modes for the trackpad when compiling its firmware: mouse mode, and "emulate directional keys" mode. My beepberry was set to directional keys. You can tell because you can move the console cursor around. If it's in mouse mode it won't do anything to the console when you move it, but it will show up in supported input apps (like the DirectFB2 example program "df_input"). You will have to reprogram it using the bbqX0kbd_driver from sqfmi if you want a different behavior.

I'm getting "Segmentation fault" errors when trying to run DirectFB example apps.

Look at the last few lines leading up to the segfault. They might mention DRMKMS/System, and "Using device /dev/dri/card0 (default)".

This error is because your Pi's HDMI / GPU video system appears as the primary interface in the system, so it defaults to that. The framebuffer video system (driving the Sharp Memory LCD) is the second interface. We have to tell DirectFB to default to the framebuffer instead:

```
export DFBARGS="system=fbdev,fbdev=/dev/fb1"
```

Then relaunch the example program. Quit it with ctrl-c.

Debian Bookworm Bug List (2024-04)

DirectFB-media fails to build.

-> Skipped building it and the DirectFB2 examples.

Mouse pointer overruns X coordinates.

pygame-ce/src_c/event.c, search for "case SDL_MOUSEMOTION"

https://wiki.libsdl.org/SDL2/SDL_MouseMotionEvent

Debugging snakewm apps

journalctl -f -u snakewm.service
gdb -p <PID of python>

Replace cursor blitting step

```
pygame ui_manager.py, line 100:
 self.active_user_cursor =
pygame.cursors.Cursor(pygame.SYSTEM_CURSOR_ARROW)
```

References

Hardware Measurements

Distance from PCB top side to top of keyboard, with as-shipped double-sided foam tape connecting the two: 3.5mm

Distance from PCB top side -> top of Sharp Memory LCD glass: 2.6mm

PCB approx 73x115mm

Other bitmap libraries of note

embedded-graphics

A Rust 2D graphics library for memory-constrained devices
https://github.com/embedded-graphics/embedded-graphics/

LVGL

https://lvgl.io/
https://github.com/lvgl/

MicroGUI for MicroPython

https://github.com/peterhinch/micropython-micro-gui
https://github.com/peterhinch/micropython-nano-gui

MicroUI

"A tiny, portable, immediate-mode UI library written in ANSI C."
https://github.com/rxi/microui

GTK-Server

http://www.gtk-server.org/apps.html

DirectFB references

List of some DirectFB toolkits: https://higfxback.github.io/DirectFB.html

"DirectFB Overview," Andreas Hundt:
https://www.13thmonkey.org/documentation/misc/DirectFB_overview_V0.1.pdf

Other beepy projects

MP3 Player on Console: https://github.com/sockbot/mocp/

Improving the installer

dialog: curses-based ui framework

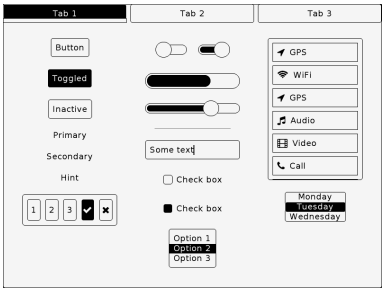
https://invisible-island.net/dialog/dialog-figures.html

Hardware Ideas

These ideas are my own, outdated ideas for beepy variants and do not reflect roadmaps from SQFMI or myself. They are preserved here for reference.

v1 hardware - aka sqfmi-beepy-v1

- primitive audio digitizer for Q20 mic using RP2040 ADC
- LVGL / MicroPython on the RP2040, outputting video to the Memory LCD (requires hardware modifications)
- Expansion idea, for small boards (20x30mm) with low pin needs. I call them "ticks" after the insect. Size of a large postage stamp. Screws into one of the screwholes on the backplane standoff. It has pogo pins which are spring-loaded against the back side of the 10-pin "utility connectors" around the beepy display. Only active lines would need to be connected.



LVGL Mono Theme

v2 hardware idea - aka "graphy" - the graphical version

- Similar to the V1, it is an "open frame" design with exposed PCB to the edge.
- Designed for RP2040 graphical control of the Memory LCD, coexisting with the Raspberry Pi Zero.
- (Optional) Re-use SQFMI beepy polycarbonate case design, with no or few tooling changes.
- Data lines for Memory LCD and touchpad I2C are broken out / fanned out internally. Each line would have a single test point, circular hole in the soldermask, with a matching circular copper pad. The owner could then probe the lines with a pogo pin jig, solder fly wires to them, etc.
- Changes from the beepy 2023 schematic as follows.

- **Rewire pins.** The following would enable REPL over UART with MicroPython on the RP2040.
  It connects the hardware UARTs from the RP2040 and the Pi Zero together.
  RX/TX lines should be on a cuttable bridge or 0 ohm shunt, allowing them to disconnect for other projects.

  - RP2040 pin 2 > GPIO0 > UART0-TX -to- Zero pin 10, GPIO15 (UART0-RX)
    - Repurpose KB_INT line from beepy-v1
    - Move KB_INT to RP2040 pin 4 (GPIO2)
    - Shift all keyboard pins up in the same way, by 2 GPIOs each, keeping it GPIO-consistent, not pin-consistent.
      - COL1 > from pin 3 to pin 5
      - COL2 > from pin 4 to pin 6
      - ...
      - ROW2 > from pin 16 to pin 18
      - ROW1 > from pin 17 to pin 27
      - Delete PI_PWR_SW line from pin 18. Remove PI_PWR_SWT line from the design.
      -
  - RP2040 pin 3 > GPIO1 > UART0-RX -to- Zero pin 8, GPIO14 (UART0-TX)
    - Repurpose COL1 line from beepy-v1
    - Shift all keyboard matrix pins up by 1.
      - Move RP2040 pin 2
      - Substitute RP2040 pin 18 > GPIO15 -> connect to COL1 on keyboard
        - Repurpose PI_PWR_SW line from beepy-v1
        - Delete PI_PWR_SW line

- **Rewire pins.** Replace discrete RGB LEDs with a single WS2812 programmable RGB LED, with its data line fanned out to an edge connector. This enables the owner to control a programmable LED strip directly from the beepy, using the built-in one as a test. As an ADC pin, it could also be used as a multi-purpose ADC and digital I/O port. The WS2812 data line should connect into the edge connector line with a cuttable

solder bridge, or shunt resistor, allowing the WS2812 to be taken out of circuit and the edge connector used directly if so desired.

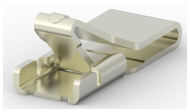
- RP2040 pin 39 > GPIO27 > ADC1 –to– WS2812–Data–In and one board edge connector
  - Takes KB_MICROPHONE line from beepy–v1
  - Delete KB_MICROPHONE line.

- **Rewire pins.** Connect these to allow the RP2040 to control the Memory LCD with LVGL.

- RP2040 pin > GPIO18 > SPI0_SCLK –to– Micro LCD pin 1 > DISP_SCLK
  - Takes TP_SDA line from beepy–v1.
  - Relocate TP_SDA.
  - Use RP2040 pin 34 > GPIO22 > I2C1_SDA –> connect to TP_SDA on keyboard
- RP2040 pin 30 > GPIO19 > SPI0_MOSI –to– Micro LCD pin 2 > DISP_SI
  - Takes LED_G line from beepy–v1
- RP2040 pin 28 > GPIO17 > SPI0_CS –to– Micro LCD pin 3 > DISP_CS
  - Takes LED_B line from beepy–v1
- RP2040 pin 27 > GPIO16 > SPI0_MISO –to– Micro LCD pin 4 > DISP_EXTIN
  - Takes TP_RESET line from beepy–v1

  - --> Need to relocate TP_RESET
    - Use GPIO20
  - --> Need to relocate TP_SDA
    - Use RP2040 pin 34 > GPIO22 > I2C1_SDA -> connect to TP_SDA on keyboard
  - --> Need to relocate KB_INT
  - --> Need to relocate COL1
    - Use RP2040 pin 18 > GPIO15 -> connect to COL1 on keyboard
- (Optional) Two RP2040 UART1 TX/RX pins fanned out.

- introduces the "chippy" header for attaching small PCBs
- power system sized for different lithium ion configurations
- Thinner overall package, achieved possibly several ways:
  - use the existing ZIF connector, and flip the Raspberry Pi board around (sides Front and Back) so that its components are "tucked" in between its own PCB and the beepy PCB. Leave PCB right above the Raspi SiP bare copper, then connect to it via thermal pad, using as heatsink.
  - use Spring Fingers, spring pads, or other lower profile connectors
  - rotate LCD 180° "upside down" and route its cable up around a milled indentation in the edge of the PCB - modify Memory LCD Linux driver to rotate display 180°. Reduces cable clutter in the center of the device. May improve ease of manufacturing.
  - spring fingers + flipped raspi in a milled cutout behind pcb supporting the Memory LCD



TE Connectivity, spring finger, P/N 2292838-3

- several internal 5-pin headers for tiny "chippy" PCB expansion modules.
  - internal version: break out gpios and power to 5-pin 0.100" headers, or possibly spring fingers. Pros / cons to each.
  - Each header is GND, key, 3V3, one I2C, one GPIO.
  - These headers are present for both the Raspi, and the RP2040.

Holland Shielding, spring contact, P/N 2901-05

RP2040 CircuitPython (experimental)

The goal is to make the beepy "all-python." First thing is to reflash the RP2040 with CircuitPython.

Connections to peripherals

The RP2040 is the governor of the beepy. It controls most components.

Power regulator

Battery charger

LED

3 GPIO pins for each RGB channel in the LED on top of the beepy.
It is not a programmable LED like the WS2812 or similar.
Each of its three RGB LEDs is set by an analog output pin from the RP2040, so it consumes three GPIO pins.

Keyboard

Optical Touchpad

Enable line to the Pi Zero

Turns the Pi Zero on and off.

External flash IC

Contains the RP2040's boot OS and any apps. We use MicroPython as the boot OS.

Install CircuitPython

Download stable CircuitPython from https://circuitpython.org/board/raspberry_pi_pico/
As of writing it's 9.0.4.
Put onto the beepy. See CircuitPython section of build-sdk.py.

Developing for the RP2040

Hardware on the sqfmi-beepy-v1 board

Pins available for our use:

Signal	RP2040 Pin	RP2040 Pin Capabilities	Pi Zero Pin	Pi Zero Pin Capabilities	Proposed function
KB_SCL	GPIO29	UART0 RX, I2C0 SCL	GPIO2	I2C SDA	RP2040 RX / Pi Zero Soft-UART TX
KB_SDA	GPIO28	UART0 TX, I2C0 SDA	GPIO3	I2C SCL	RP2040 TX / Pi Zero Soft-UART RX
KB_INT	GPIO0 (UART TX by default)	UART0 TX, I2C0 SDA	GPIO4	GPCLK0 (General Purpose Clock)	
SHUTDOWN_DET	GPIO21	UART1 RX, I2C0 SCL	GPIO26		
LED_B	GPIO17	UART0 RX	n/a - needs rewiring		

Debugging the RP2040

Establish an OpenOCD server in the background, configured for the beepy hardware:

```
sudo openocd -f interface/raspberrypi-native.cfg -f target/rp2040.cfg -c 'transport select swd' -c 'adapter gpio swdio 24' -c 'adapter gpio swclk 25' &
```

Use gdb to connect to OpenOCD:

```
# gdb
(gdb) tar ext :3333
```

Restart the RP2040:

```
(gdb) monitor reset
```

TODO: Start, stop, single step core.

TODO: Load symbols for current binary.

Examine prepared ELF files

$ objdump -s -j .rodata ~/glowfire/beepy-sdk/src/rp2040-firmware.elf

Installing picotool

apt install -y libusb-dev

References

Debugging CircuitPython: https://github.com/adafruit/circuitpython/blob/main/BUILDING.md#Debugging

Using hardware UART on the RP2040: https://learn.adafruit.com/circuitpython-essentials/circuitpython-uart-serial

Developing CircuitPython apps for the RP2040

```
>>> uart = busio.UART(tx=board.GP0, rx=board.GP21, baudrate=9600, timeout=0)
```

Getting a console on the RP2040

Once flashed with CircuitPython, we'll want to interact with its REPL somehow.

References

https://docs.micropython.org/en/latest/rp2/quickref.html#uart-serial-bus

Run a Python script at boot

When CircuitPython is compiled, it constructs a "code.py" file at the root of the filesystem. This is automatically run at startup.

See ~/beepy-sdk/src/circuitpython-rp2040/supervisor/shared/filesystem.c, line 156:

```
// make a sample code.py file
        MAKE_FILE_WITH_OPTIONAL_CONTENTS(&vfs_fat->fatfs, "/code.py", "print(\"Hello
World!\")\n");
```

Change to:

```
// make a sample code.py file
        MAKE_FILE_WITH_OPTIONAL_CONTENTS(&vfs_fat->fatfs, "/code.py", "import
board\nimport time\nfrom digitalio import DigitalInOut, Direction\nled =
DigitalInOut(board.LED)\nled.direction = Direction.OUTPUT\nwhile True:\n    if led.lit:\n
led.lit = False\n    else:\n        led.lit = True\n\n    time.sleep(500)\n");
```

Test app:

```
import time
import board
#import busio
import digitalio

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

#uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=0)

while True:
    led.value = True
    time.sleep(0.2)
    led.value = False
    time.sleep(0.2)
    print("Hello")
#    uart.write("test")
```

https://learn.adafruit.com/circuitpython-essentials/circuitpython-resetting

Resetting the RP2040 programmatically in CircuitPython
import microcontroller
microcontroller.reset()

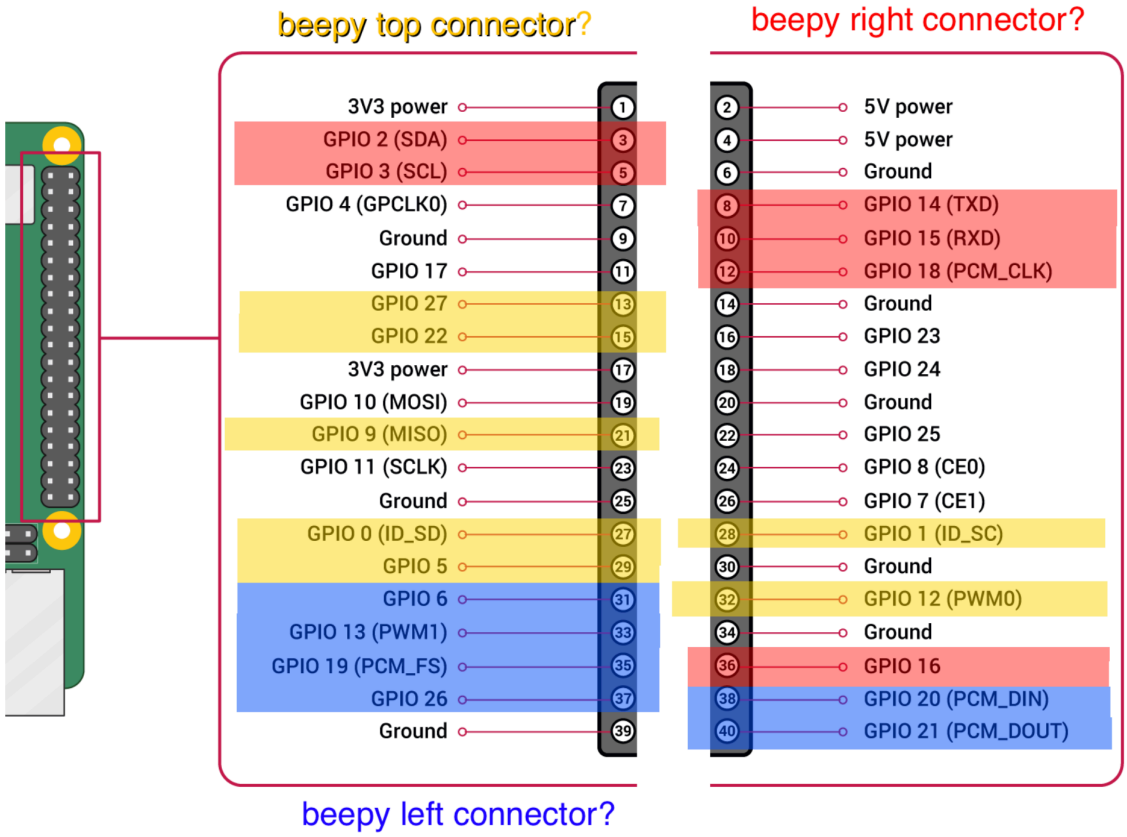
LoRA module interfacing

We will interface the LoRA module to the raspi's SPI bus, which is shared with the Memory LCD.

Pinout of the device to the Memory LCD is thus:

Pi GPIO-23 (Pin 16) "GPIO23" -> Net: DISP_EXTIN -> Memory LCD (Pin 4) "EXTCOMIN"
--> test point TP1 is DISP_EXTIN
Pi GPIO-10 (Pin 19) "MOSI0" -> Net: DISP_SI -> Memory LCD (Pin 2) "SI"
--> test point TP5 is DISP_SI
Pi GPIO-11 (Pin 23) "SCLK0" -> Net: DISP_SCLK -> Memory LCD (Pin 1) "SCLK"
--> test point TP8 is DISP_SCLK
Pi GPIO-8 (Pin 24) "CE0" -> Net: DISP_CS -> Memory LCD (Pin 3) "SCS"
--> test point TP6 is DISP_CS

Pi Pin 7 (GPIO10) is INT pin.



Heltec HT-62 module (SX1262 LoRa + ESP32C3)

Overview

This 15mm-square module integrates both a LoRa chip and an ESP32-C3 RISC-V microprocessor beneath a single RF can. It has (2) uFL antenna connectors, one for wifi/BT and the other for LoRa.

Use in a prototype

Fix deadbug-style onto acrylic with supertape;

air-wire 24ga strands btw this module and your other stuff's pins

Install Heltec support library

Arduino > Tools > Manage Libraries > search for "Heltec ESP32 Dev-Boards," install package by Heltec Automation.

Programming the module

* Bed o' nails
* USB interfacing - chipset?
* What about programming from the raspi itself, using UART?

Antennas

* Length for each
* Positioning, etc. Links to appropriate resources.

Integration onto a PCB

Footprint

Schematic

Anything besides this (decoupling capacitors, components on the RF side, PCB routing considerations / keepout zones / ground fills / etc.

RF cans for everything? Aluminum tape as a shield?

Detachable vs. fixed antennas.

LoRa applications

Codec 2 speech codec
GitHub: https://github.com/drowe67/codec2
License: LGPL 2.1

HF radio applications

FreeDV, a Digital Voice mode for HF radio
Web: https://freedv.org/

LoRa app idea: change-rgb-color

Remote site: ESP32-C3 board with individually addressable RGB LED and LoRa transceiver
Mobile site: beepy with RP2040-powered LoRa transceiver

The remote site can change the beepy LED between R,G,B over LoRa by pressing its Boot button.
The beepy can change the remote site's LED RGB values via a pygame app.
This demonstrates the ability to communicate over LoRa from pygame.