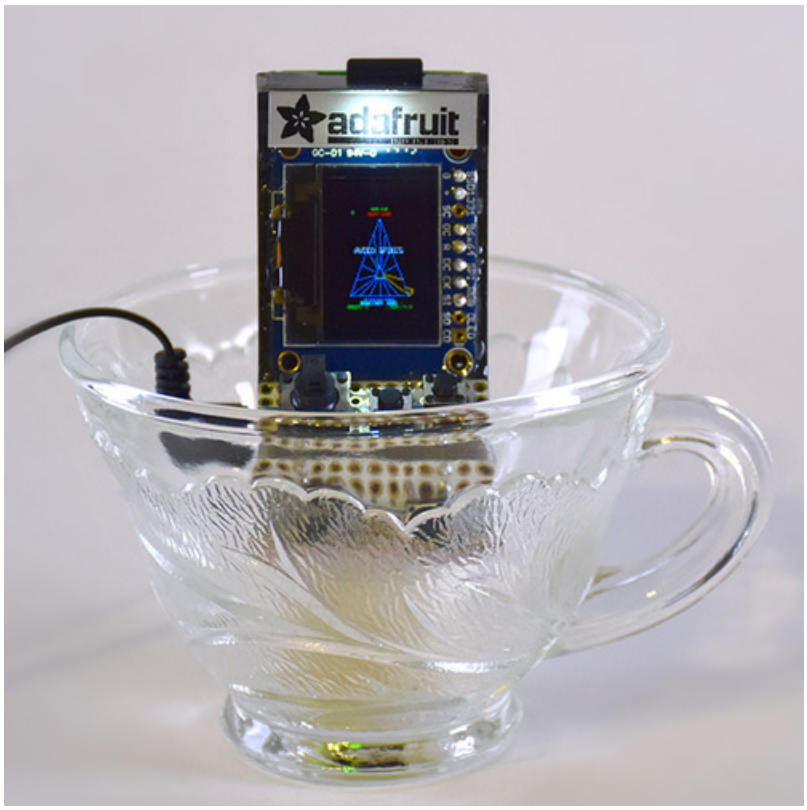
adafruit learning system

World's Smallest MAME Arcade Cabinet

Created by Phillip Burgess



Last updated on 2018-08-22 03:56:12 PM UTC

Guide Contents

Overview

HEADS UP: This is NOT a kit that we sell. Nor is it a complete step-by-step guide. It's lessons learned during an impromptu weekend hacking session which, to be honest, was a *lot* of trouble to build and only marginally fun to play, aside from the incredible *"gee whiz that's tiny!"* factor. But it may provide insights for others looking to build small gadgets...

A practical and satisfying gaming project for intermediate builders is our **PiGRRL Zero (https://adafru.it/rnF)**; better controls, better display, runs off a battery. Beginners may want to start with running **RetroPie (https://adafru.it/qoa)** on a full-size Raspberry Pi and regular monitor, then you can incrementally learn about adding arcade controls or a small LCD screen (https://adafru.it/n3c).

Just trying to temper expectations...this is an unrefined "sketch" of a project and lacks the finish of our other guides.

The idea came about while discussing a gaming "bonnet" — a small accessory board precisely fitted to the Raspberry Pi Zero form-factor — which would include a few basic controls and a tiny monochrome OLED display. The question was whether a color OLED might be workable, that we might run RetroPie and all those cool old games.

The short answer is *no!* The color display is just too small and coarse, requiring major graphics downsampling that render existing games just barely recognizable. But it's also a matter of cost. We're talking about an accessory board for a $5 computer; the RGB OLED alone can cost several times that! A mono OLED is more "to scale" with the system cost and the idea of writing small bespoke games in a high-level language like Python. They'll look great and sharp, and it'll be affordable.

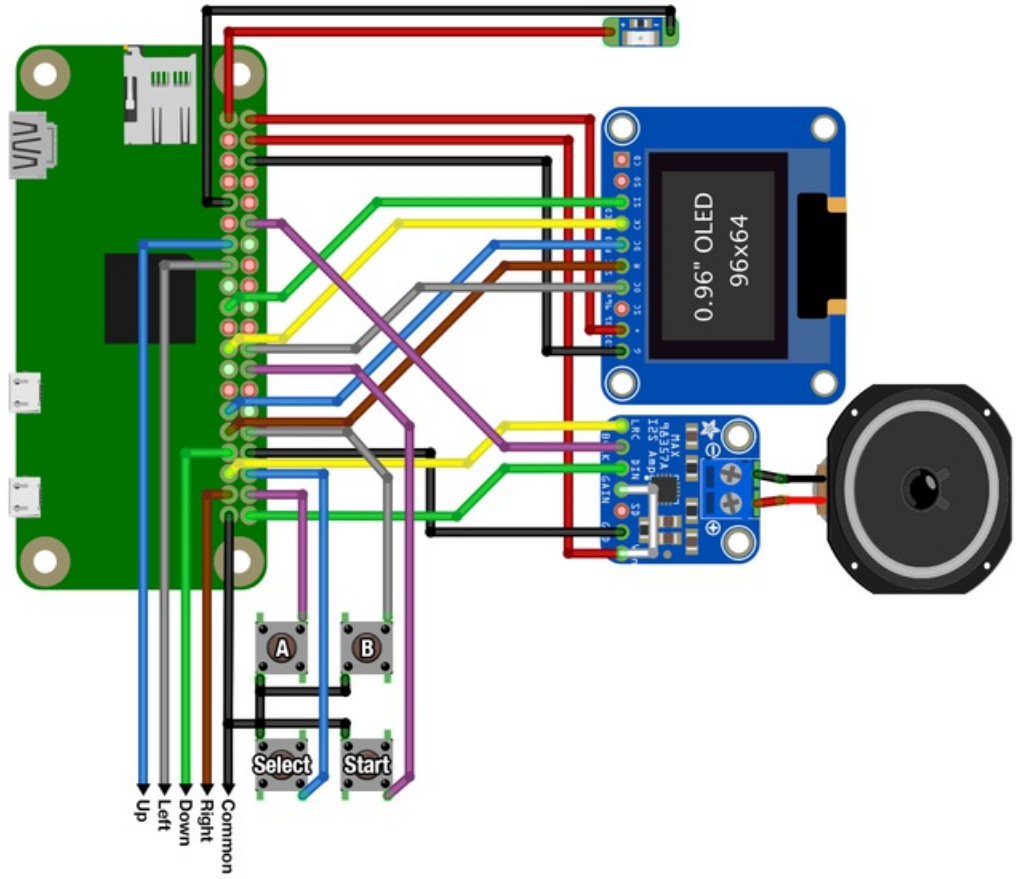
Seeing these arcade games though…like, the actual quarter chompers of my youth, not cheap ports…running on this tiny screen *is* pretty hilarious. Wrapping it up in a little case and badging it "world's smallest" seemed a nice way to cap off the study…though I fully expect to be upstaged by someone else's *even smaller* build before the week is out, then repeat *ad nauseum.* Small MAME builds are a big thing!

Hardware

The project is built around a Raspberry Pi Zero (http://adafru.it/2885) computer, one of our 0.96" RGB OLED (http://adafru.it/684) displays, an I2S class D audio amplifier (http://adafru.it/3006) and a few assorted odds and ends (buttons, navigation switch, Perma-Proto board, etc.).

Since this project doesn't require a camera, I used a "V1" Pi Zero that I had on-hand. If you're looking to create the very smallest possible thing, and don't need a camera, be aware that the clip on the end of a newer "V1.3" Pi Zero protrudes a couple of millimeters past the board edge. Depending on your level of obsessiveness, you might need to remove that (destroying the connector and possibly the board) or specifically hunt down a V1 board (as if Pi Zeros were't elusive enough).

Most of the connections for the display and audio amp must go to specific pins on the Pi; they rely on hardware features of the board and are not negotiable. Others like the controls could be routed elsewhere if needed.



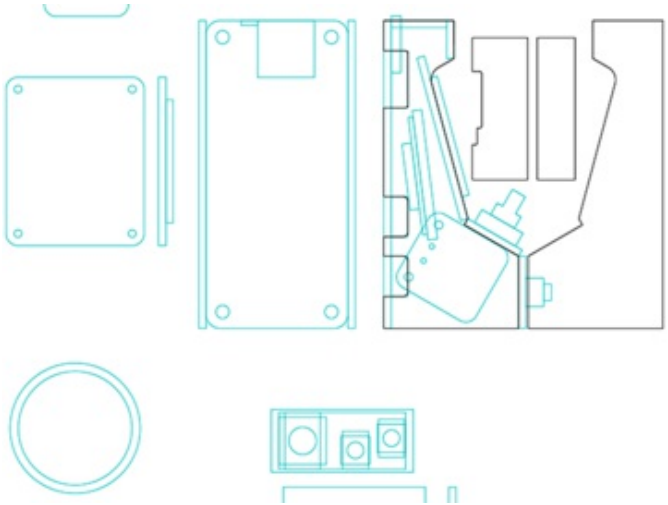
Raspberry Pi Header	Other Device
+5V	OLED +
GND	OLED G
GPIO10 (MOSI)	OLED SI (serial in)
GPIO11 (SCLK)	OLED CK (serial clock)
GPIO8 (CE0)	OLED OC (OLED chip select)
GPIO5	OLED DC (data/command)
GPIO6	OLED R (reset)
+5V	Amp Vin
GND	Amp GND
GPIO19	Amp LRC (left/right clock)
GPIO18	Amp BCLK (bit clock)
GPIO21	Amp DIN (data in)
	Amp GAIN is tied to Amp Vin (6dB gain)
GND	Common pin on navigation switch and one leg of each button
GPIO20	Opposite leg of 'A' button
GPIO12	'B' button
GPIO16	'Select' button
GPIO7	'Start' button
GPIO27	Nav switch up
GPIO22	Nav switch left
GPIO13	Nav switch down
GPIO26	Nav switch right
+3.3V	LED Sequin +
GND	LED Sequin –

Woo, that's a lot of connections! Even following along with a GPIO reference card (http://adafru.it/2263) I botched a few wires and had to do some troubleshooting. If you mix up the button inputs, that can be easily fixed in software, but the

screen and amp are very specific about the pins used.

The LED sequin is totally optional...just thought it would be a nice finishing touch to have a backlit marquee.

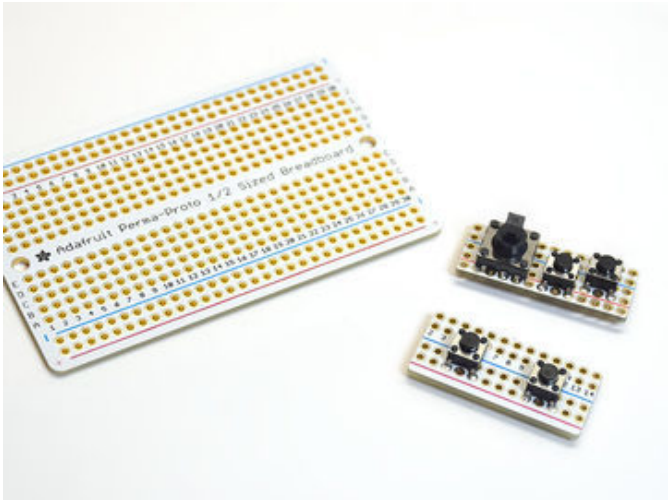
Please remember, **this is *not* a kit or a how-to.** The steps are lacking depth, and attempting something like this requires a lot of expensive tools and parts, prior experience with small project builds and Pi gaming specifically, and a willingness to improvise and/or fail repeatedly.



Using calipers, I measured each part and came up with a case idea. Rather than fully enclose everything, some elements (the Perma-Proto board holding the controls, plus the Raspberry Pi Zero board itself) would themselves become *elements* of the structure.

I chose to do this in 1/16" laser-cut acrylic, because when you have a laser cutter you stupidly think *everything's* a job for laser-cut acrylic. To further economize on space, there would be no fasteners, everything would be glued.

This was a poor choice and please don't ask for the file. If I woke up and it was Saturday again, *Groundhog Day*-style, 3D printing would have been much better. Takes more time to design up front, but way less time fighting with glue and wobbly bits.



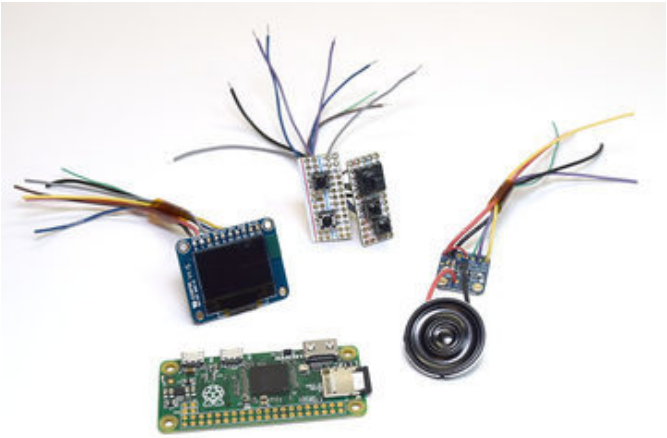
Perma-Proto boards are the bee's knees! They can be cut to size on a scroll saw, then the edges cleaned up and trimmed to a more precise size on a disc sander.

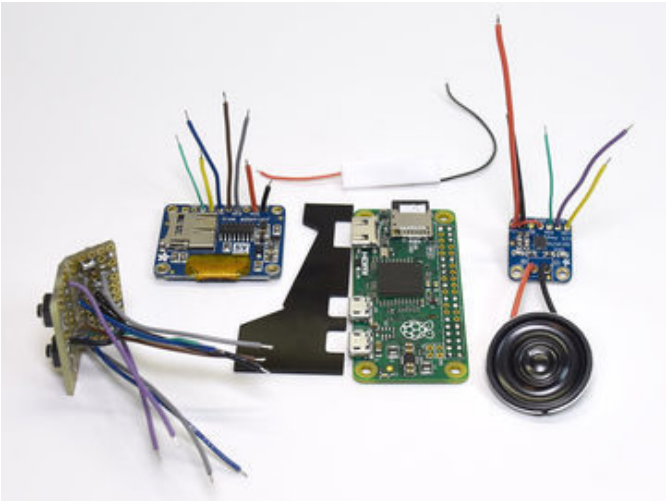
Two pins on the 5-way navigation switch do not lie on the regular 0.1" grid. It was necessary to drill extra holes between others and make some solder bridges.

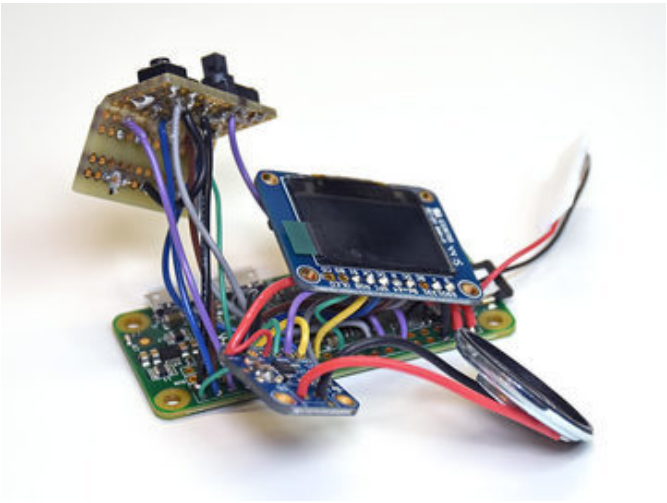
Ignore the power bus lines here; some traces were cut on the back to achieve the necessary continuity (or not) to make each button work. This was tested with a multimeter and adjusted before further assembly.

Another thought was to make the entire case from more Perma-Proto boards carefully cut to shape. Cool, but time-consuming. Perhaps this is reasonable with something like an Othermill.

As a "farting around" project, the wiring, like the case design, wasn't carefully plotted beforehand. All the wires were initially cut to a uniform 4" length and one end soldered to components, which were then moved around and each wire trimmed to a suitable length with a few millimeters slack, plus some extra to strip and tin.







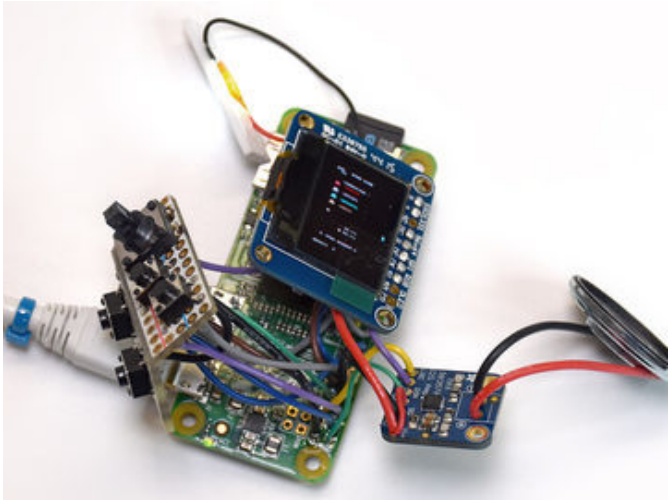
Making matters more challenging was the discovery that I only had two colors of 26-gauge silicone-coated stranded wire on-hand. That's no help...26 gauge is too fat for this cramped space, and more colors really do help for keeping track of things, and better match the photos to the wiring diagram. So I made do with some inferior 28-gauge non-silicone wire, using the thicker stuff for just the few power-related connections. The process required patience, tweezers and a good soldering iron.

If you've never used silicone-coated stranded wire before, I can't recommend it highly enough! Super flexy. Even within Adafruit it's developed something of a cult following. "Hey, have you tried this silicone wire?" "Eh, what's the big deal? It's just wire with...HOLY CRAP THIS STUFF IS MAGICAL!"

Extra bonus challenge: I've been shifting to using lead-free solder, and this project seemed a good opportunity to go all-in. I might take this thing to a Maker Faire or something, where kids would play with it and then go eat funnel cakes...lead-free just made good sense.

Lead-free soldering is a *little* more tricky, but nowhere near as dreadful as some would have you believe. Patience and a good hot iron, no big deal. I splurged on a refurbished Metcal iron several years back and have never once regretted

the purchase. I've heard nothing but praise for Hakko's stuff too. The patience part though, that's something not for sale anywhere, you'll have to cultivate that yourself.



Before enclosing everything, a dry run is imperative! I'd already set up the software (explained on the next page) on a micro SD card. Found in testing that I'd botched a few wires...mixed up the controls, and had the marquee LED + on a GPIO pin instead of +3.3V. Test, adjust and repeat as needed.

Notice the protective plastic cover is still on the OLED. This helps keep solder flux spatter and glue from messing up this most expensive component. It comes off later as the very last step.

Huh! It appears I didn't get any photos of the case-gluing process. Well I can assure you it was a bad path and mostly consisted of a lot of swearing, do-overs, probably some crying, and having to hold pieces seemingly forever while the "five minute" epoxy cured (maybe it's five minutes on like Venus or something), half of it on my fingers and probably rendering me sterile. Some kind of press-fit 3D-printed caselet would've been so much easier.

You'll see there's a tiny speaker wedged in there. By sheer dumb chance, it happened to align with a blank spot on the Pi board (near the logo) where an edge could be glued down without fear of shorting components or traces on the board.

The audio amp is glued to the "wall" of the case, at an angle. With all those wires fighting for space, the amplifier board protruded slightly from the bottom. Since there are no active electrical traces around the mounting holes, I just clipped the corner off the board.

The glue job is rough and pieces are badly aligned, but eventually it all stuck and stands on its own! A second pass with epoxy over the inside seams hopefully makes it strong enough for casual play.

Notice the screen was installed “vertically.” Could’ve gone either way, but there are a couple reasons I chose this orientation:

1. Some of the most iconic games of the 1980s used vertically-mounted monitors. Following suit emphasizes the fact that these are the real deal, not cheap console ports designed for regular “horizontal” screens.
2. The OLED breakout board is narrower on this axis, roughly matching the Raspberry Pi. Placing the screen the “wide” way would have required a larger case, working against the sheer tiny-ness goal.

Final dimensions:

- 67.2 mm (2.65") tall (with micro SD card installed)
- 33.6 mm (1.32") wide
- 35.8 mm (1.41") deep (back to tip of start/select buttons)

Could it go smaller? Undoubtedly! Other than clipping the corner off the audio amp board, these are all stock parts and no extreme measures were taken to further reduce their volume, Ben Heck-style. Creative placement of the Pi board at an angle might reduce the height. Slimmer buttons are easy to find. Smaller Linux boards will come along, or might already be out there. Maybe some of those super-hi-res smartwatch OLED screens will start trickling into the hobbyist market.

But is is practical? As I’ll explain a bit on the next page, the coarse resolution of the screen makes the games difficult to play (especially something like Donkey Kong, where climbing ladders requires aligning the character within a couple of pixels)...I suspect a lot of the “playing” is just muscle memory from past experience. Honestly the whole thing’s a bit gimmicky for the sake of smallness. Sharing it for a laugh.

One bit I’m extremely happy with though is the I2S audio amp (http://adafru.it/3006). Even through a tiny 1" speaker, the sound is incredible! That’s an item I’ll definitely be using on future (and more sensibly-sized) emulator projects.

Software

Here's where things get really jargon-y. If you've not done a Raspberry Pi retro gaming project before, and specifically a PiTFT-based build (https://adafru.it/n3c), this might all be Greek. Prior experience is assumed.

This started with version 3.8.1 of RetroPie (https://adafru.it/qoa), a gaming-centric OS distribution for the Raspberry Pi. They're up to version 4 now, but I specifically used this older release because it had advmame 0.94 (a specific version of a specific arcade machine emulator) already baked in…I just like how this version renders certain games and how the keyboard input is set up.

The software was loaded on a Raspberry Pi Model B+…an older board no longer made, not a Pi 2 or Pi 3. RetroPie has two separate builds, one for single-core boards, one for multi-core. The Pi Zero and the Model B+ are both single-core, so the card could be used interchangeably between the two. With only one USB port and funky mini HDMI output, the Pi Zero can be a challenge to load up…the Model B+ has an assortment of normal ports (including Ethernet networking), so getting it on the network to install additional software was no big thing.

From other PiTFT-based gaming projects, you might be familiar with three vital pieces of software:

- A *device tree overlay* is the driver that makes an LCD screen appear as a "first class citizen" to the rest of the system — it's now like having a second monitor attached, albeit a really dinky one. The Raspbian operating system (used by Retropie and many others) already includes overlays for several popular TFT displays…but not the tiny OLED we're using here.
- *fbcp* (framebuffer copy) is a utility that continually copies the contents of the main video display (e.g. HDMI) to the secondary display (e.g. LCD). This way, emulators don't need to "know" about the LCD…they just work as they normally would on an HDMI monitor, while this other code handles the translation.
- *retrogame* is a little Adafruit program that translates GPIO inputs (buttons, joystick) into virtual keyboard presses… again, so the emulator doesn't need to know from GPIO hardware, it just goes along like there's a keyboard attached.

This project does without the first two. Drivers in Linux are fiendishly tricky to write. Instead, a little user-space program called *nanoscreen (https://adafru.it/zdE)* handles the task of copying the main framebuffer directly to the

OLED display using SPI transfers. SPI must be enabled (via raspi-config) for this to work.

Because of the way nanoscreen works (more on that in a moment), the display *must* be configured for 4X the OLED resolution, or 384x256 pixels. Here's what's in my /boot/config.txt:

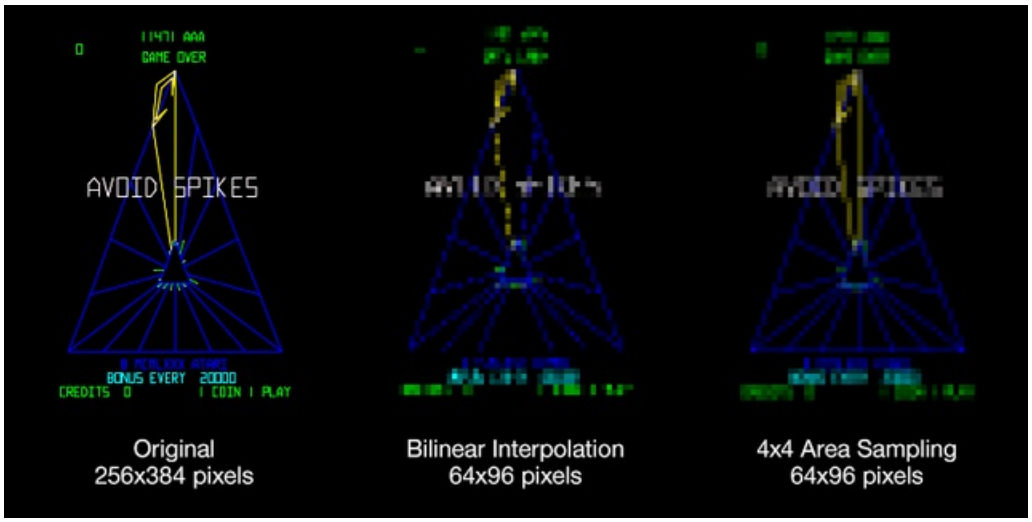
```
# Not all monitors can handle this resolution,
# but the framebuffer is there and correct and
# nanoscreen will work with it.
disable_overscan=1
hdmi_force_hotplug=1
hdmi_group=2
hdmi_mode=87
hdmi_cvt=384 256 60 1 0 0 0
# Optional, for 'portrait' video:
display_rotate=3
```

Not all HDMI monitors support this; you might get an "unsupported resolution" or "no signal" alert on the display. Thing to do then is set up and test at a more sensible resolution, then edit /boot/config.txt as a last step before unplugging HDMI and just using the OLED.

fbcp (which was very insightful in writing nanoscreen) uses the *Dispmanx* library to capture the screen and downsample this to the size of the alternate framebuffer. This scaling is handled on the GPU and is super efficient. However, it presented a problem with the degree of scaling required for this tiny OLED...

Dispmanx provides *bilinear interpolation* when scaling. This is normally a good thing, it smooths out the jaggies. But bilinear interpolation starts to fall apart when the reduction factor is greater than 2. Very small details can fall "between" each group of 4 pixels used in the interpolation. In some games this is just a visual annoyance, but in others it's vital. In *Pac Man* for instance...some of the individual dots around the maze get lost this way, rendering the game unplayable as you can't always tell which parts of the maze have been cleared.

nanoscreen does a software-based 1:4 scale using 4x4 pixel averaging, which preserves these smaller details...though *blurry,* the vital missing dots are at least *visible.* Vector games look better too.



This software-based scaling isn't the most optimal route...I'm *certain* Dispmanx could handle it on the GPU using a 2-stage bilinear scale with an intermediate canvas of some sort...but nanoscreen was just an evening project and I couldn't get into that depth of understanding with Dispmanx; software scaling was familiar and comparatively easy, and

runs well enough for what it is.

Something else to improve legibility at this size was to increase all the font sizes in EmulationStation. I edited the default theme file /etc/emulationstation/themes/carbon/carbon.xml and roughly doubled the values accompanying any reference to “fontSize”.

This fixes the game list menu in MAME, but the exit menu and the RetroPie menu that’s used to shut down the system properly are both impossibly small and unreadable. I suspect there are settings for both, but I’ve MAMEd and RetroPie’d enough that these are just navigated from muscle memory.

Here’s the GPIO-and-key table used in *retrogame:*

```
   ioStandard[] = {
        // For Nanoscreen:
        // Input   Output (from /usr/include/linux/input.h)
        {  27,     KEY_UP    },   // Joystick (4 pins)
        {  22,     KEY_LEFT  },
        {  13,     KEY_DOWN  },
        {  26,     KEY_RIGHT },
        {  20,     KEY_Z     },   // A/Fire/jump/primary
        {  12,     KEY_X     },   // B/Bomb/secondary
        {   7,     KEY_1     },   // Start
        {  16,     KEY_5     },   // Select/credit
        {  -1,     -1        } }; // END OF LIST, DO NOT CHANGE

const unsigned long vulcanMask = (1L << 7) | (1L << 16);
```

When I’d found that I’d crossed some of the button wires, it was easier to change the pin numbers in this table than resolder those wires. Recompile as per usual.

RetroPie controls will need to be configured to match your final setup.

As in the earlier “Cupcade” project, the Start and Select buttons can be held to simulate pressing the ESC key to bring up the exit menu in MAME. Normally this menu expects an ENTER key when making a selection...I had to root around to enable using the “A” button for this instead.

In /opt/retropie/configs/mame-advmame/advmame-0.94.0.rc:

```
input_map[ui_select] keyboard[0,lcontrol] or keyboard[1,scan0]
```

In hindsight, it might’ve been easier to squeeze an extra button or two somewhere on the case.

nanoscreen and retrogame are launched automatically at startup by adding lines to **/etc/rc.local**, just before the final “exit 0” line:

```
/boot/Adafruit/Adafruit_Nanoscreen/nanoscreen &
/boot/Adafruit/Adafruit-Retrogame/retrogame &
```

I placed these files in /boot so they could more easily be edited or installed from other systems in a pinch. Working locally, this does appear to require editing and compiling as root though, just FYI.

The I2S audio amplifier was enabled using the Pimoroni script mentioned in the MAX98357 guide (https://adafru.it/qpB). Again this was done on a Model B+ to easily retrieve things from the network.

What is this, MAME for ants? Yes...it's *ant-y MAME.*

So there you go, world's smallest...this week, anyway. I'm throwing down the *Gauntlet.* And the *Tempest,* and *Battlezone...*