Matrix Portal Creature Eyes
Created by Phillip Burgess

Last updated on 2021-03-17 01:16:23 PM EDT
Guide Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>Parts</td>
<td>3</td>
</tr>
<tr>
<td>Prep the MatrixPortal</td>
<td>10</td>
</tr>
<tr>
<td>Power Prep</td>
<td>10</td>
</tr>
<tr>
<td>Power Terminals</td>
<td>11</td>
</tr>
<tr>
<td>Panel Power</td>
<td>11</td>
</tr>
<tr>
<td>Board Connection</td>
<td>12</td>
</tr>
<tr>
<td>Install CircuitPython</td>
<td>14</td>
</tr>
<tr>
<td>Set up CircuitPython Quick Start!</td>
<td>14</td>
</tr>
<tr>
<td>Further Information</td>
<td>14</td>
</tr>
<tr>
<td>CircuitPython Setup</td>
<td>16</td>
</tr>
<tr>
<td>Adafruit CircuitPython Bundle</td>
<td>16</td>
</tr>
<tr>
<td>Install Code and Graphics</td>
<td>17</td>
</tr>
<tr>
<td>Ready-Made Creatures</td>
<td>18</td>
</tr>
<tr>
<td>Making New Creatures</td>
<td>20</td>
</tr>
<tr>
<td>Graphics and Coordinates</td>
<td>20</td>
</tr>
<tr>
<td>Coding Positions and Movement</td>
<td>22</td>
</tr>
</tbody>
</table>
Overview

Seems like every great monster film sets off a chain of sequels. Bride of..., Son of..., Revenge of..., Teen Wolf Too (hey, they can’t all be winners).

Our electronic eyeball projects have spawned their own little franchise, with Pi Eyes (https://adafru.it/zrC), HalloWing (https://adafru.it/CEs), Monster M4SK (https://adafru.it/GfQ) and more (https://adafru.it/Ha4). The arrival of the Matrix Portal M4 (https://adafru.it/Nzd) board made another sequel inevitable — this time a retro-style return to form of our first project of the series (https://adafru.it/iwB), but punching it up with bright colors, full-screen themes, and coded all in CircuitPython now.

This makes a nifty Halloween window or tabletop display. And it’s a not-too-daunting introduction to CircuitPython and graphics. When the Halloween season’s over...if you don’t keep the decorations up year ’round like some of us...everything can be repurposed into your own projects, or try out some others like a Moon phase clock (https://adafru.it/NB7).

This project requires:

- Adafruit Matrix Portal M4 (https://adafru.it/Nzd) board
- Any of our 64x32 pixel “HUB75” (not NeoPixel) RGB LED matrices
- USB C cable
- USB power supply with output of 2 Amps or more

This guide will get the software running on the bare Matrix Portal hardware. Mounting or supporting the clock in an enclosure or frame is left as an exercise to the reader.

Parts

Your browser does not support the video tag.

Adafruit Matrix Portal - CircuitPython Powered Internet Display

Folks love our wide selection of RGB matrices and accessories, for making custom colorful LED displays... and our RGB Matrix Shields...

Out of Stock

You can use a USB C power supply or a USB micro B with a micro B to C adapter (https://adafru.it/FQR)
USB Type A to Type C Cable - approx 1 meter / 3 ft long
As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...
Out of Stock

Official Raspberry Pi Power Supply 5.1V 3A with USB C
The official Raspberry Pi USB-C power supply is here! And of course, we have 'em in classic Adafruit black! Super fast with just the right amount of cable length to get your Pi 4...
$7.95
In Stock

5V 2.5A Switching Power Supply with 20AWG MicroUSB Cable
Our all-in-one 5V 2.5 Amp + MicroUSB cable power adapter is the perfect choice for powering single-board computers like Raspberry Pi, BeagleBone or anything else that's power...
$7.50
In Stock
Micro B USB to USB C Adapter
As technology changes and adapts, so does Adafruit, and speaking of adapting, this adapter has a Micro B USB jack and a USB C...
$1.25
In Stock
Add to Cart
If you'd like your LEDs diffused (and if your LED matrix is 4mm pitch or smaller), some acrylic may help:

Your browser does not support the video tag.
Black LED Diffusion Acrylic Panel 12" x 12" - 0.1" / 2.6mm thick
A nice whoppin’ slab of some lovely black acrylic to add some extra diffusion to your LED Matrix project. This material is 2.6mm (0.1”) thick and is made of special cast...
$9.95
In Stock
Add to Cart
Adafruit carries a number of 64x32 RGB LED Matrices, varying between the space between LEDs (pitch) and whether rigid or flexible. Choose your favorite - larger pitch means the display is larger, width and height-wise but with the same number of pixels, and larger may be easier to read further away. Smaller for near your desk, for example.
64x32 RGB LED Matrix - 3mm pitch
Bring a little bit of Times Square into your home with this sweet 64 x 32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them...
Out of Stock

Your browser does not support the video tag.

64x32 RGB LED Matrix - 4mm pitch
Bring a little bit of Times Square into your home with this sweet 64 x 32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them...
Out of Stock
64x32 RGB LED Matrix - 5mm pitch
Bring a little bit of Times Square into your home with this sweet 64x32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them on...
Out of Stock

64x32 RGB LED Matrix - 6mm pitch
Bring a little bit of Times Square into your home with this sweet 64x32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them on...

$64.95
In Stock

Out of Stock
Add to Cart

Your browser does not support the video tag.

64x32 Flexible RGB LED Matrix - 4mm Pitch
If you've played with multiplexed RGB matrices, you may have wondered "hey, could we possibly manufacture these on a thin enough PCB, so it's flexible?" and the...

Out of Stock

Your browser does not support the video tag.

64x32 Flexible RGB LED Matrix - 5mm Pitch
If you've played with multiplexed RGB matrices, you may have wondered "hey, could we possibly manufacture these on a thin enough PCB so it's flexible?" and the answer...

$79.95
In Stock

Add to Cart
Prep the MatrixPortal

Power Prep
The MatrixPortal supplies power to the matrix display panel via two standoffs. These come with protective tape applied (part of our manufacturing process) which MUST BE REMOVED!

Use some tweezers or a fingernail to remove the two amber circles.
Power Terminals
Next, screw in the spade connectors to the corresponding standoff.

- red wire goes to +5V
- black wire goes to GND

Panel Power
Plug either one of the four-conductor power plugs into the power connector pins on the panel. The plug can only go in one way, and that way is marked on the board's silkscreen.
Now, plug the board into the left side shrouded 8x2 connector as shown. The orientation matters, so take a moment to confirm that the white indicator arrow on the matrix panel is oriented pointing up and right as seen here and the MatrixPortal overhangs the edge of the panel when connected. This allows you to use the edge buttons from the front side.

Check nothing is impeding the board from plugging in firmly. If there’s a plastic nub on the matrix that’s keeping the Portal from sitting flat, cut it off with diagonal cutters.
For info on adding LED diffusion acrylic, see the page LED Matrix Diffuser.
Install CircuitPython

CircuitPython ([https://adafruit.it/tB7](https://adafruit.it/tB7)) is a derivative of MicroPython ([https://adafruit.it/BeZ](https://adafruit.it/BeZ)) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the CIRCUITPY drive to iterate.

Set up CircuitPython Quick Start!
Follow this quick step-by-step for super-fast Python power :)

Further Information
For more detailed info on installing CircuitPython, check out [Installing CircuitPython](https://adafruit.it/Amd).

Click the link above and download the latest UF2 file.
Download and save it to your desktop (or wherever is handy).

Plug your MatrixPortal M4 into your computer using a known-good USB cable.

A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

Double-click the Reset button (indicated by the green arrow) on your board, and you will see the NeoPixel RGB LED (indicated by the magenta arrow) turn green. If it turns red, check the USB cable, try another USB port, etc.

If double-clicking doesn't work the first time, try again.
Sometimes it can take a few tries to get the rhythm right!
You will see a new disk drive appear called **MATRIXBOOT**.

Drag the *adafruit_circuitpython_etc.uf2* file to **MATRIXBOOT**.

The LED will flash. Then, the **MATRIXBOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it, you're done! :)
CircuitPython Setup
To use all the amazing features of your MatrixPortal M4 with CircuitPython, you must first install a number of libraries. This page covers that process.

Adafruit CircuitPython Bundle
Download the Adafruit CircuitPython Library Bundle. You can find the latest release here:

https://adafru.it/ENC

Download the `adafruit-circuitpython-bundle-version-mpy-*.zip` bundle zip file, and unzip a folder of the same name. Inside you'll find a `lib` folder. The entire collection of libraries is too large to fit on the CIRCUITPY drive. Instead, add each library as you need it, this will reduce the space usage but you'll need to put in a little more effort.

At a minimum we recommend the following libraries, in fact we more than recommend. They're basically required. So grab them and install them into CIRCUITPY/lib now!

- `adafruit_matrixportal` - this library is the main library used with the MatrixPortal.
- `adafruit_portalbase` - This is the base library that adafruit_matrixportal is built on top of.
- `adafruit_esp32spi` - this is the library that gives you internet access via the ESP32 using (you guessed it!) SPI transport. You need this for anything Internet
- `neopixel` - for controlling the onboard neopixel
- `adafruit_bus_device` - low level support for I2C/SPI
- `adafruit_requests` - this library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- `adafruit_fakerequests.mpy` - This library allows you to create fake HTTP requests by using local files.
- `adafruit_io` - this library helps connect the PyPortal to our free data logging and viewing service
- `adafruit_bitmap_font` - we have fancy font support, and it's easy to make new fonts. This library reads and parses font files.
- `adafruit_display_text` - not surprisingly, it displays text on the screen
- `adafruit_lis3dh` - this library is used for the onboard accelerometer to detect the orientation of the MatrixPortal
Install Code and Graphics

Back up any existing code or files you want to keep from your Matrix Portal CIRCUITPY drive.

Most of the CircuitPython libraries used by this project were already recommended on the prior page...but there’s a couple extras needed as well. Here’s a list of all the prerequisite libraries...copy them into the lib folder on the CIRCUITPY drive:

- adafruit_imageload
- adafruit_matrixportal

Now fetch the files for our creature eyes project (https://adafruit.it/NAY) from Github:

https://adafruit.it/NAY

Unzip this file after downloading. The eyes folder and code.py should be copied into the CIRCUITPY root directory.

![code.py]

- eyes
- lib
- adafruit_imageload
- adafruit_matrixportal

And that’s it! If all the right files are copied over, the creature eyes should begin running automatically.

If you want to select a different creature (a few designs are included) or create your own, that’s explained next...
**Ready-Made Creatures**

To select among different ready-made creatures, edit the file `code.py` on the CIRCUITPY drive, using your text editor of preference.

A few lines into the code you’ll see several references to `EYE_DATA`. Most of these lines are “commented out” — they have no effect on the CircuitPython code. A “#” character indicates the start of a comment. But one of these lines is enabled.

```python
from eyes.werewolf.data import EYE_DATA
# from eyes.cyclops.data import EYE_DATA
# from eyes.kobold.data import EYE_DATA
```

All you need to do is comment out the currently-active line (the “werewolf” line above) by adding a `#`. Then remove the `#` from the line you want to enable. Only one of these lines should be active at a time.

Save the changes to the file, and the code should restart automatically, provided you did the comment/uncomment change correctly.

Our default werewolf design is active with the `eyes.werewolf.data` line enabled.

Werewolves have been a thing this year, with the Halloween full Moon ([https://adafruit.it/NB7](https://adafruit.it/NB7)) and all.

Enable the `eyes.cyclops.data` line to bring up this icky single eye...demonstrating that it doesn’t always need to be two (in theory it could do more, if you can design something legible at that resolution).
And this creature comes from the eyes.kobold.data line.

In traditional Germanic folklore, kobolds are sort of gnome-like creatures. Dungeons & Dragons popularized the idea of kobolds as small lizard people. I'm okay with that, lizards are cool.
Making New Creatures
Let's suppose you want to go all-out and hatch some creatures of your own.

You will need:

- Image editing software that can create BMP files (I use Photoshop, but even most basic paint programs will do...and if they don’t save BMP directly, you can find free image conversion utilities to help)
- An understanding of image coordinate systems (explained below)
- A text editor (we like Mu (https://adafru.it/LEQ) for editing CircuitPython projects, but most anything will do)

With the eyes project already installed and running...on the CIRCUITPY drive, each sub-folder inside the “eyes” folder contains 5 files:

- Four BMP image files
- data.py, a bit of Python code describing how the images work together

```
| ▼ eyes |
| ▼ werewolf |
| data.py |
| werewolf-eyes.bmp |
| werewolf-lower-lids.bmp |
| werewolf-stencil.bmp |
| werewolf-upper-lids.bmp |
```

If making a new creature, it’s easiest to start by duplicating one of the existing folders and giving it a descriptive name, e.g. “gillman”. Then the files within can be edited and renamed as needed.

To activate your new creature, remember to add a line in code.py, telling it to import your new design...as was done with the ready-made creatures. This is based on the name of the folder, e.g.:

```
from eyes.gillman.data import EYE_DATA
```

(And then comment out any/all inactive creatures in code.py. Only one should be active.)

Graphics and Coordinates
Some of the images are partly transparent. The animation code “stacks” these one atop the next, and shuffles them around to create the looking-and-blinking effect.

The top-most image in this stack depicts most of the monster’s face, with cut-outs for the eyes (or single eye, in the case of the cyclops example). For the werewolf example, this image is called werewolf-stencil.bmp. This image is always the same size as the LED matrix (64x32 pixels in the examples) and never moves. The eye holes are bright green in this image (this color was chosen because it’s not used anywhere else in the werewolf set), but in other examples are bright red (for similar reasons).

Below this are two images for the top and bottom eyelids...same deal, bright green represents pixels that
will be transparent in the final “stack.”

The bottom-most image is the creature’s eyes...the whites, pupils and so forth. This image does not use the transparent color, since there’s nothing below to show through. It needs to be bigger than just the eye holes, since it will be moving around and we want something to always be filling those pixels.

The images all need to be 8-bit “indexed color” or “paletted” BMP files. 24-bit BMPs, and other image formats, will not work with this code.

Here’s an image being converted and saved from Photoshop, but the same functionality is available in many image editors, or there are free tools like ImageMagick (https://adafruit.it/NBh).

Your images might appear washed-out and too bright on the matrix...the code that handles the LED matrix doesn’t yet support gamma correction. You can compensate for this somewhat in your image editor...try adjusting the gamma (the middle value in Photoshop’s “Levels” command) to 0.4 or so. Keep a copy of your original art around! Do this level adjustment only as a last step before saving. It’s a destructive process and you might be testing several iterations to get the images just right.

Next is to figure out the pixel coordinates where things will be moving.

Remember that the frontmost “stencil” image is always the same size as the matrix, so that’s a good point of reference. The top left pixel of this image has coordinates (0, 0). Moving right, X (the first value in the coordinate pair) increases by 1 for each pixel, or -1 to the left. Moving down, Y (second value) increases by
1, or -1 for up.

Sometimes the images will be moved to negative coordinates. This is normal and okay! The graphics are automatically “clipped” to the matrix boundaries as needed.

```
(0,0)

EYE_PATH = __file__[:__file__.rfind('/') + 1]

EYE_DATA = {
    'eye_image'        : EYE_PATH + 'werewolf-eyes.bmp',
    'upper_lid_image'  : EYE_PATH + 'werewolf-upper-lids.bmp',
    'lower_lid_image'  : EYE_PATH + 'werewolf-lower-lids.bmp',
    'stencil_image'    : EYE_PATH + 'werewolf-stencil.bmp',
    'transparent'      : (0, 255, 0),
    'eye_move_min'     : (-3, -5),
    'eye_move_max'     : (7, 6),
    'upper_lid_open'   : (7, -4),
    'upper_lid_center' : (7, -1),
    'upper_lid_closed' : (7, 8),
    'lower_lid_open'   : (7, 22),
    'lower_lid_center' : (7, 21),
    'lower_lid_closed' : (7, 17),
}
```

Keep in mind, **this is Python code**, and so it’s going to be **strict** about syntax! Make sure strings are quoted there’s a comma at the end of each line in the table and so forth. If your new creature fails to run, the problem may be with the file syntax...keep a serial connection open to the board and see what it says.

The **EYE_PATH** line is just weird Python syntax for “wherever this file is located, we want to locate things in the same directory.” It avoids us having to specify an absolute path to every image file.

**EYE_DATA** is a Python **dictionary** — it contains pairs of **keys** and **values**, each separated by a colon. The **key strings must remain unchanged**. These are the names that the main Python code (in code.py) will be looking for. **Edit only the values.**

The first four items in this dictionary specify the **BMP image files** used for the different layers of the animation, as described earlier. **EYE_PATH** is mentioned here to indicate “these images are in the same
directory as this file."

The next item, 'transparent', is the RGB color value (red, green, blue) that will “show through” in these images as they’re stacked. It’s green for the werewolf, red for other examples.

'eye_move_min' is the leftmost and topmost position of the eye image when looking directly left and up, respectively. 'eye_move_max' is the rightmost and bottom-most positions. These two points describe a rectangle...but the actual eye movement will be constrained to an ellipse inside this rectangle, so don’t worry if your pupils go out of bounds when at these two points...it’s really the “compass points” left, top, right and bottom that we care about.

The next three items, 'upper_lid_open', 'upper_lid_center' and 'upper_lid_closed' are the top-left pixel coordinates of the upper eyelid image relative to the matrix (or the topmost stencil image), at the eyelid’s most-open position, in a neutral (eyes centered) position, and at its lowest position (when blinking). Having three values here (rather than just two) allows the eyelids to “track” the movement of the eyes, which is a neat thing that eyelids really do. The 'upper_lid_closed' value usually won’t completely cover the eye, unless you’re specifically aiming for that look. It’s normal for the upper and lower eyelids to meet part way. The X values for all these coordinates will usually be the same.

Last three items, 'lower_lid_open', 'lower_lid_center' and 'lower_lid_closed' are similar, but for the bottom eyelid. Whereas the upper eyelid will usually have increasing Y values for each (going top to bottom), the lower eyelid will usually have decreasing Y values.

Even with careful planning, the various eye images might not be the right sizes or end up in the intended positions on the first try. It’s perfectly normal to take a few iterations through this file to get all the graphics lined up and moving as intended.

If you’re editing data.py but not seeing any changes: make sure you’re also loading the correct file in code.py (as shown earlier on this page).