Tombstone Matrix Portal

Created by Ruiz Brothers

https://learn.adafruit.com/tombstone-matrix-portal

Last updated on 2024-06-03 03:14:23 PM EDT

Table of Contents

Overview

In this project we're making a tombstone with Adafruit's Matrix Portal.

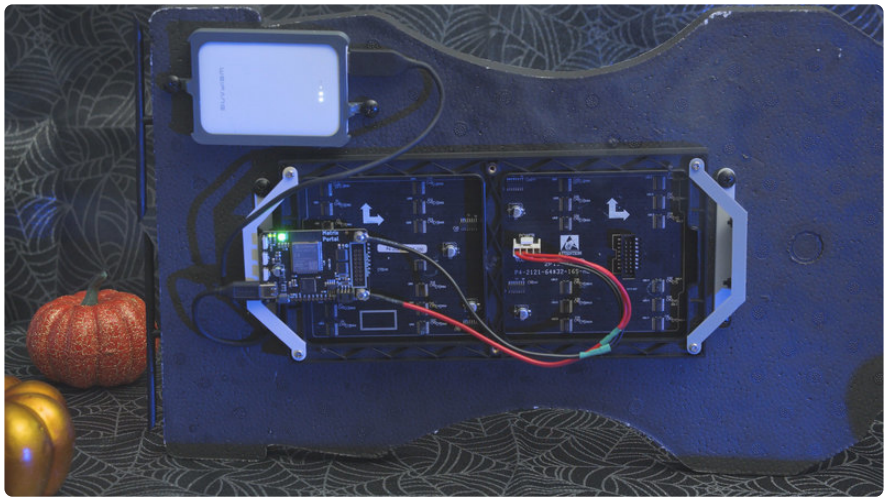
We upgraded this foam tombstone with an RGB matrix so we can display animated graphics.

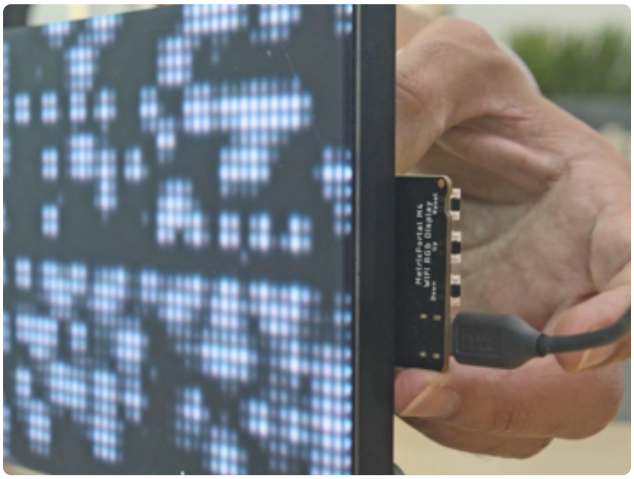
The 3D skull is just a bitmap sprite sheet that is animated using CircuitPython.

The Matrix Portal is a plug-n-play dev board that makes these types of projects fairly easy.

We designed and 3D printed these brackets so we can attach this to things like halloween props, picture frames or just some foam core.

With CircuitPython, you can animate sprites sheets and display them on the matrix!





The bitmaps autoplay but you can also use the built-in buttons on the side to cycle through them!

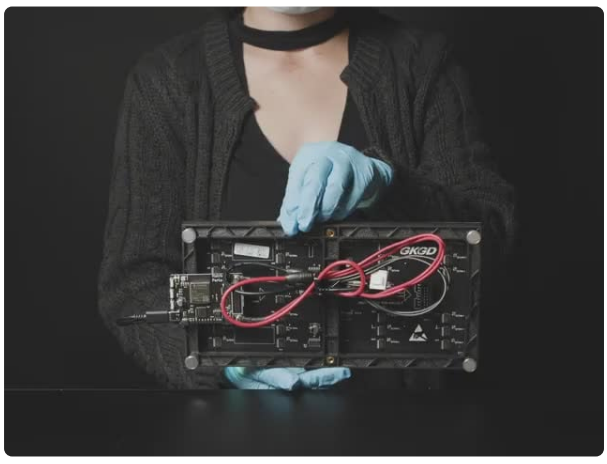
Prerequisite Guides

Adafruit Matrix Portal (https://adafru.it/NDR)

Pixel Art on RGB Matrix (https://adafru.it/NTA)

Sprite Sheet Animation (https://adafru.it/NTB)

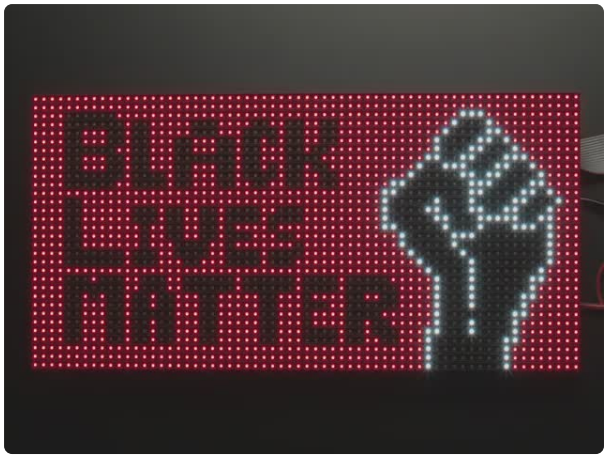
Parts



Adafruit Matrix Portal - CircuitPython Powered Internet Display

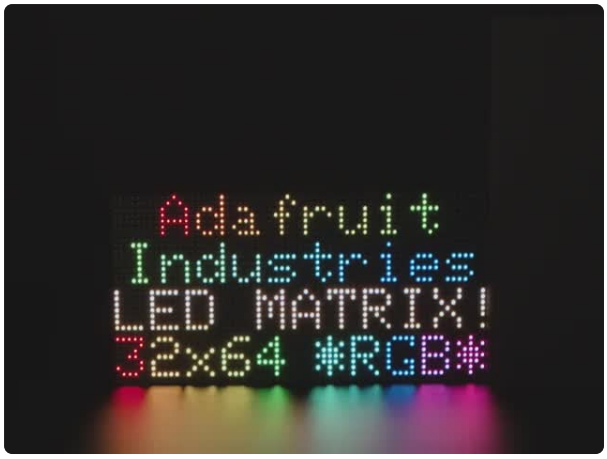
Folks love our wide selection of RGB matrices and accessories, for making custom colorful LED displays... and our RGB Matrix Shields...

https://www.adafruit.com/product/4745



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...
https://www.adafruit.com/product/2278

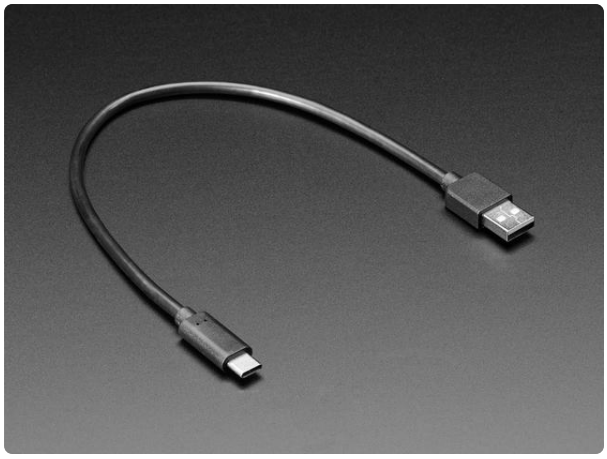


Black LED Diffusion Acrylic Panel 12" x 12" - 0.1" / 2.6mm thick

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...
https://www.adafruit.com/product/4594

USB Li-Ion Power Bank with 2 x 5V Outputs @ 2.1A - 5000mAh

Here's a hand-held-size rechargeable battery pack for your Raspberry Pi (or
https://www.adafruit.com/product/4288



USB Type A to Type C Cable - 1ft - 0.3 meter

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...
https://www.adafruit.com/product/4473

Electric hot knife cutter for foam

1 x Hot Knife Tool https://amzn.to/30Wwt0E

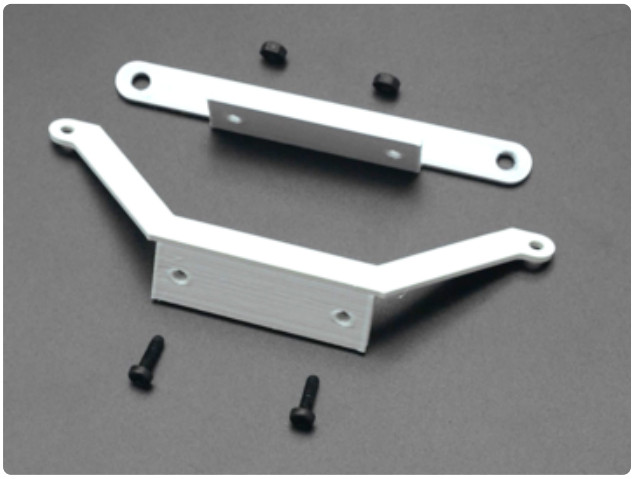
Electric hot knife cutter for foam

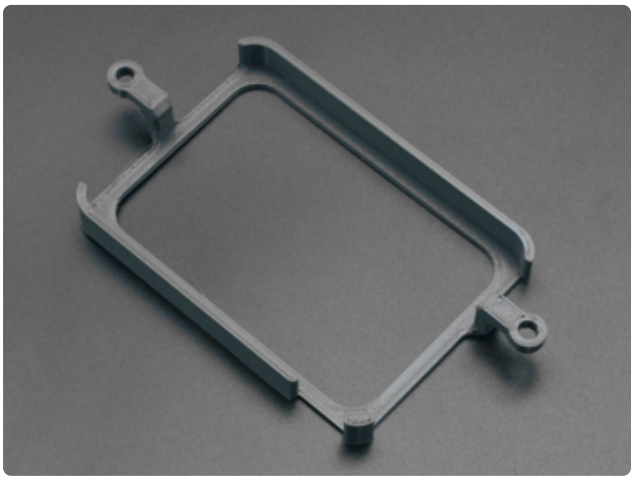
---

1 x Pro Tapes Sheets https://amzn.to/350HMWH

ProTapes 306UGLU600 UGlu Dash Sheets

3D Printing





Parts List

STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material. Original design source may be downloaded using the links below.

Bracket 1
Bracket 2
Battery Bracket

Edit Matrix Bracket Design (64x32 RGB LED Matrix - 4mm pitch PID: 2278)

https://adafru.it/NTC

Matrix Display Feet

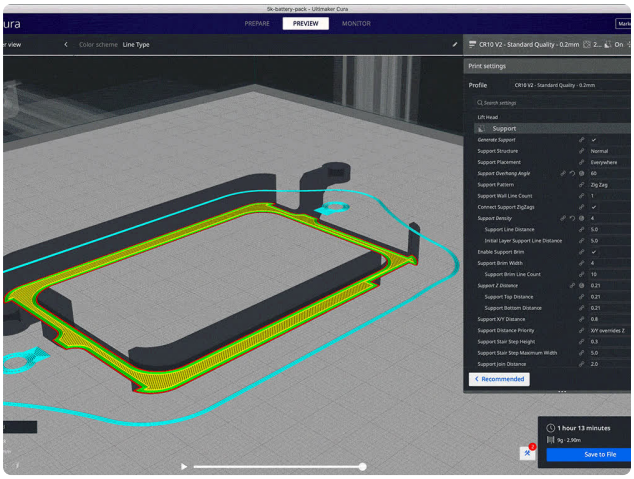
https://adafru.it/NYD

Edit 5k mAh Battery Pack

https://adafru.it/NTE

Download STLs

https://adafru.it/NUd



Slicing Parts

Slice with setting for PLA material. The parts were sliced using CURA using the slice settings below.

PLA filament

215c extruder

0.2 layer height

10% gyroid infill

60mm/s print speed

60c heated bed

Supports

4% density

.2 extrusion width

Install CircuitPython

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.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!

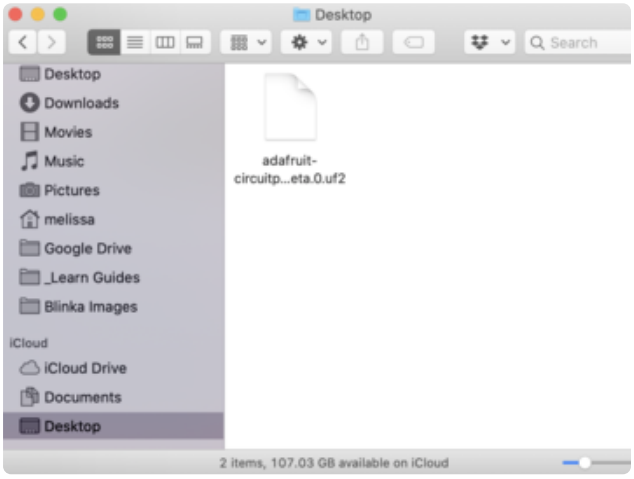
Follow this quick step-by-step for super-fast Python power :)

Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/Nte

Further Information

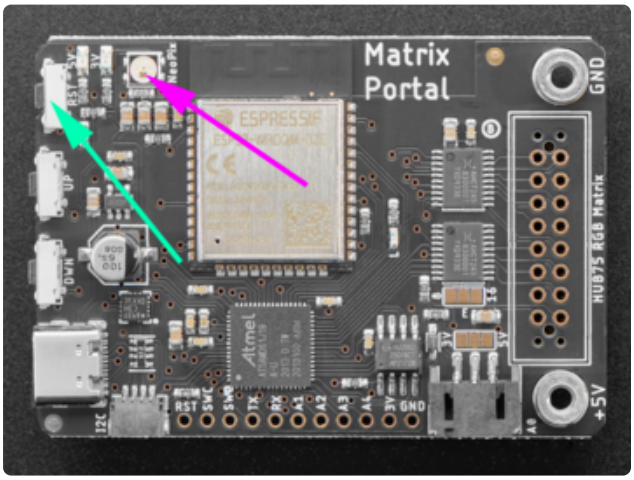
For more detailed info on installing CircuitPython, check out Installing CircuitPython (https://adafru.it/Amd).



Click the link above and download the latest UF2 file.

Download and save it to your desktop (or wherever is handy).

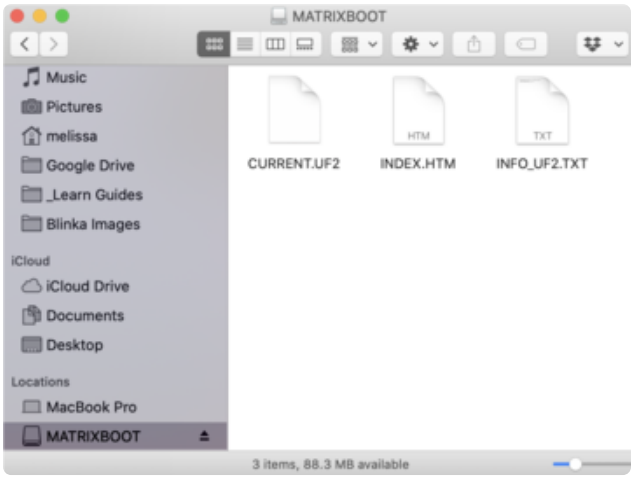
Plug your MatrixPortal M4 into your computer using a known-good USB cable.



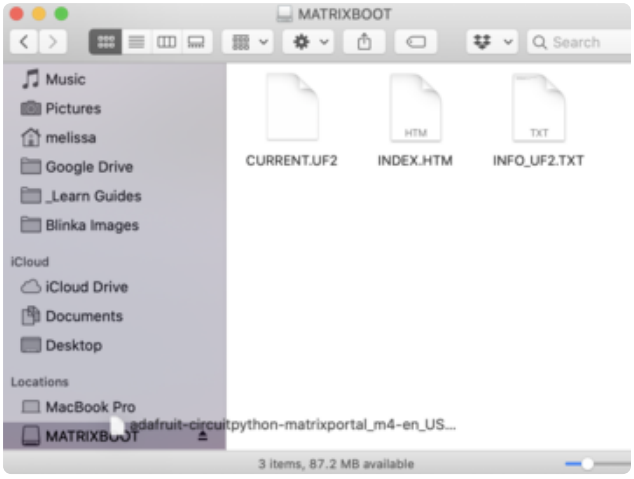
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

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.

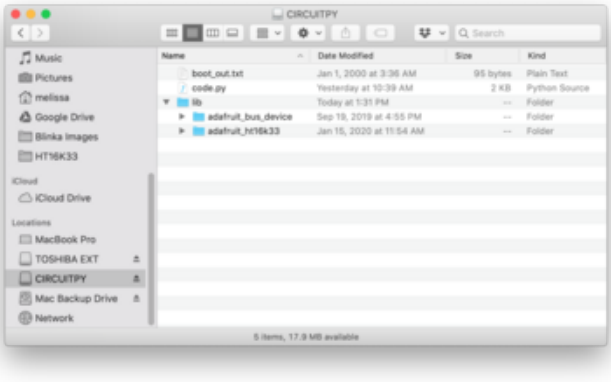
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



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:

Download latest Library Bundle

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_debouncer.mpy** - this library is used for debouncing a digital input pin
- **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.mpy** - for controlling the onboard neopixel
- **adafruit_bus_device** - low level support for I2C/SPI
- **adafruit_requests.mpy** - 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.mpy** - this library is used for the onboard accelerometer to detect the orientation of the MatrixPortal
- **adafruit_minimqtt** - this is used for communicating with MQTT servers.

Code the Sprite Sheet Animation Display

Make sure you've set up the Matrix Portal with Circuit Python and the necessary libraries as shown on the **Code the Pixel Art Display** page of this guide. This code uses the same libraries.

Code

Click the Download: Project Zip File link below in the code window to get a zip file with all the files needed for the project. Copy **code.py** from the zip file and place it on the **CIRCUITPY** drive.

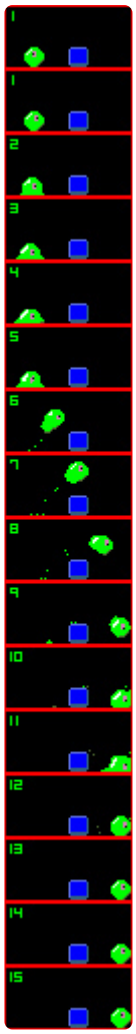
You'll also need to copy the following files to the **CIRCUITPY** drive. See the graphic at the top of the page as to filenames and where they go):

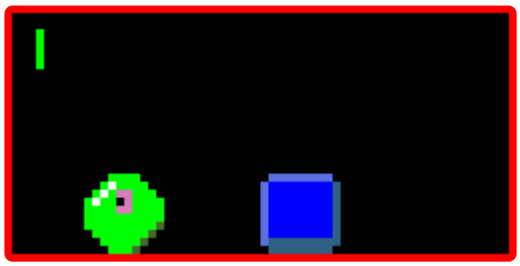
- **/bmps** directory, which contains the sprite sheet .bmp files.

Sprite Sheet Specs

Make sure your sprite sheets are

- .bmp files
- 64 pixels wide
- multiples of 32 pixels high, depending on how many frames there are
- Export as vertical sprite sheets with no border padding. The code will use these dimensions to display the tiles





```
# SPDX-FileCopyrightText: 2020 John Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import os
import board
import displayio
from digitalio import DigitalInOut, Pull
from adafruit_matrixportal.matrix import Matrix
from adafruit_debouncer import Debouncer

SPRITESHEET_FOLDER = "/bmps"
DEFAULT_FRAME_DURATION = 0.1  # 100ms
AUTO_ADVANCE_LOOPS = 3
FRAME_DURATION_OVERRIDES = {
    "three_rings1-sheet.bmp": 0.15,
    "hop1-sheet.bmp": 0.05,
    "firework1-sheet.bmp": 0.03,
}
```

```
# --- Display setup ---
matrix = Matrix(bit_depth=4)
sprite_group = displayio.Group()
matrix.display.root_group = sprite_group

# --- Button setup ---
pin_down = DigitalInOut(board.BUTTON_DOWN)
pin_down.switch_to_input(pull=Pull.UP)
button_down = Debouncer(pin_down)
pin_up = DigitalInOut(board.BUTTON_UP)
pin_up.switch_to_input(pull=Pull.UP)
button_up = Debouncer(pin_up)

auto_advance = True

file_list = sorted(
    [
        f
        for f in os.listdir(SPRITESHEET_FOLDER)
        if (f.endswith(".bmp") and not f.startswith("."))
    ]
)

if len(file_list) == 0:
    raise RuntimeError("No images found")

current_image = None
current_frame = 0
current_loop = 0
frame_count = 0
frame_duration = DEFAULT_FRAME_DURATION


def load_image():
    """
    Load an image as a sprite
    """
    # pylint: disable=global-statement
    global current_frame, current_loop, frame_count, frame_duration
    while sprite_group:
        sprite_group.pop()

    filename = SPRITESHEET_FOLDER + "/" + file_list[current_image]

    # CircuitPython 6 & 7 compatible
    bitmap = displayio.OnDiskBitmap(open(filename, "rb"))
    sprite = displayio.TileGrid(
        bitmap,
        pixel_shader=getattr(bitmap, 'pixel_shader', displayio.ColorConverter()),
        tile_width=bitmap.width,
        tile_height=matrix.display.height,
    )

    # # CircuitPython 7+ compatible
    # bitmap = displayio.OnDiskBitmap(filename)
    # sprite = displayio.TileGrid(
    #     bitmap,
    #     pixel_shader=bitmap.pixel_shader,
    #     tile_width=bitmap.width,
    #     tile_height=matrix.display.height,
    # )

    sprite_group.append(sprite)

    current_frame = 0
    current_loop = 0
    frame_count = int(bitmap.height / matrix.display.height)
    frame_duration = DEFAULT_FRAME_DURATION
```

```
    if file_list[current_image] in FRAME_DURATION_OVERRIDES:
        frame_duration = FRAME_DURATION_OVERRIDES[file_list[current_image]]


def advance_image():
    """
    Advance to the next image in the list and loop back at the end
    """
    # pylint: disable=global-statement
    global current_image
    if current_image is not None:
        current_image += 1
    if current_image is None or current_image >= len(file_list):
        current_image = 0
    load_image()


def advance_frame():
    """
    Advance to the next frame and loop back at the end
    """
    # pylint: disable=global-statement
    global current_frame, current_loop
    current_frame = current_frame + 1
    if current_frame >= frame_count:
        current_frame = 0
        current_loop = current_loop + 1
    sprite_group[0][0] = current_frame


advance_image()

while True:
    if auto_advance and current_loop >= AUTO_ADVANCE_LOOPS:
        advance_image()
    button_down.update()
    button_up.update()
    if button_up.fell:
        auto_advance = not auto_advance
    if button_down.fell:
        advance_image()
    advance_frame()
    time.sleep(frame_duration)
```

How it Works

Libraries

Here's how the code works. First we import necessary libraries including `dislpayio` and `adafruit_matrixportal` to handle the TileGrid display.

Variables

We use a few variables that are user adjustable to fine tune the way the playback works. `DEFAULT_FRAME_DURATION = 0.1` sets the frame-rate to 10fps, a good starting point, as it mimics the default playback rate in many pixel animation programs.

You can also set your own framerate overrides per sprite sheet as shown here:

```
FRAME_DURATION_OVERRIDES = {
    "three_rings1-sheet.bmp": 0.15,
    "hop1-sheet.bmp": 0.05,
    "firework1-sheet.bmp": 0.03,
}
```

The `AUTO_ADVANCE_LOOPS = 3` variable specifies how many times to run through each animation before advancing to the next one.

Setup

The display and button setups are next, followed by some variables used to track the state of the playback later.

```
# --- Display setup ---
matrix = Matrix(bit_depth=4)
sprite_group = displayio.Group(max_size=1)
matrix.display.root_group = sprite_group

# --- Button setup ---
pin_down = DigitalInOut(board.BUTTON_DOWN)
pin_down.switch_to_input(pull=Pull.UP)
button_down = Debouncer(pin_down)
pin_up = DigitalInOut(board.BUTTON_UP)
pin_up.switch_to_input(pull=Pull.UP)
button_up = Debouncer(pin_up)

auto_advance = True

file_list = sorted(
    [
        f
        for f in os.listdir(SPRITESHEET_FOLDER)
        if (f.endswith(".bmp") and not f.startswith("."))
    ]
)

if len(file_list) == 0:
    raise RuntimeError("No images found")

current_image = None
current_frame = 0
current_loop = 0
frame_count = 0
frame_duration = DEFAULT_FRAME_DURATION
```

Image Loading Fuction

The `load_image()` function is where the first part of the sprite sheet magic happens! The key moment is where the displayio.TileGrid is defined to set the `tile_height = matrix.display.height` which in the case of our dislpay is 32 pixels. This effectively slices up the sprite sheet into the individual frames for display.

```
def load_image():
    """
    Load an image as a sprite
    """
    # pylint: disable=global-statement
    global current_frame, current_loop, frame_count, frame_duration
    while sprite_group:
        sprite_group.pop()

    bitmap = displayio.OnDiskBitmap(
        open(SPRITESHEET_FOLDER + "/" + file_list[current_image], "rb")
    )

    frame_count = int(bitmap.height / matrix.display.height)
    frame_duration = DEFAULT_FRAME_DURATION
    if file_list[current_image] in FRAME_DURATION_OVERRIDES:
        frame_duration = FRAME_DURATION_OVERRIDES[file_list[current_image]]

    sprite = displayio.TileGrid(
        bitmap,
        pixel_shader=displayio.ColorConverter(),
        width=1,
        height=1,
        tile_width=bitmap.width,
        tile_height=matrix.display.height,
    )

    sprite_group.append(sprite)
    current_frame = 0
    current_loop = 0
```

The advance_image() function is used to select the next sprite sheet when it is time.

```
def advance_image():
    """
    Advance to the next image in the list and loop back at the end
    """
    # pylint: disable=global-statement
    global current_image
    if current_image is not None:
        current_image += 1
    if current_image is None or current_image &gt;= len(file_list):
        current_image = 0
    load_image()
```

And, the final part of setup is the creation of the advance_frame() function, which allows the sprite sheet to move from "frame" to "frame" as it works it's way down the sprite sheet.

```
def advance_frame():
    """
    Advance to the next frame and loop back at the end
    """
    # pylint: disable=global-statement
    global current_frame, current_loop
    current_frame = current_frame + 1
    if current_frame &gt;= frame_count:
        current_frame = 0
        current_loop = current_loop + 1
    sprite_group[0][0] = current_frame
```

Main Loop

The main loop of the program runs the advance image and advance frame functions, while also checking for button_down and button_up events.

The Up button stops the auto advance from animation to animations, constantly looping just one animation.

The Down button advances manually to the next animation.

```
while True:
    if auto_advance and current_loop &gt;= AUTO_ADVANCE_LOOPS:
        advance_image()
    button_down.update()
    button_up.update()
    if button_up.fell:
        auto_advance = not auto_advance
    if button_down.fell:
        advance_image()
    advance_frame()
    time.sleep(frame_duration)
```

Assemble

Cutout

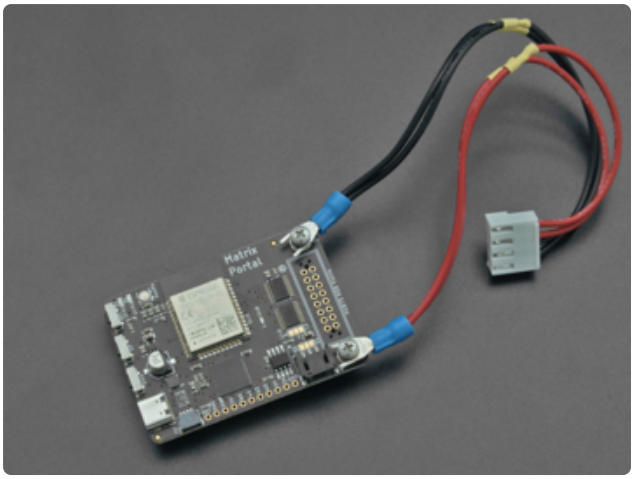
To create the opening in the tombstone, we designed a cutout with jagged edges and printed out a paper template.

Paint cracks

We used a foam cutter tool to create the opening and painted the inner edges.

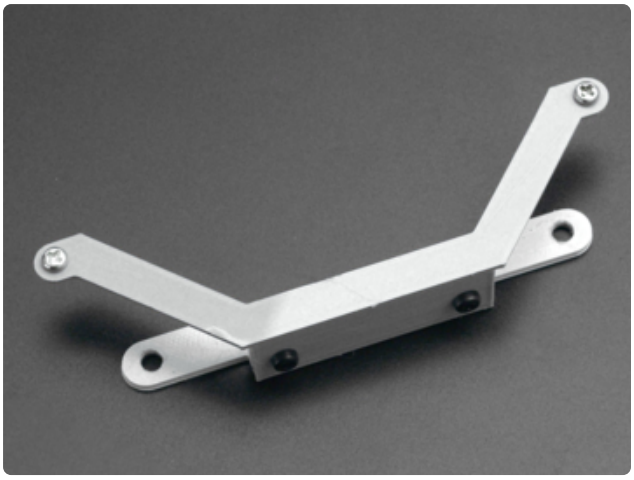
Tomb_Crack_Outline.svg

https://adafru.it/NTF



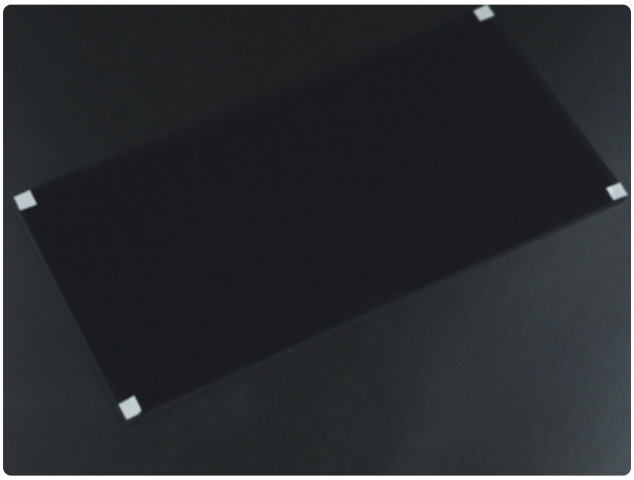
Matrix Portal wires

Connect the power and ground wires with the included hardware.



Matrix Brackets

The two bracket parts are joined together with glue or M3 screws and nuts (http://adafru.it/4685).



Attach Acrylic

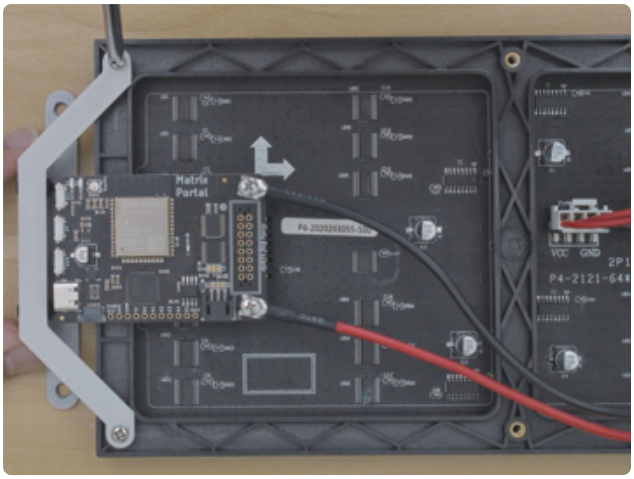
Our Black LED acrylic (http://adafru.it/4594) softens up the LEDs and keeps the colors looking vibrant.

You can attach acrylic to the display using ProTapes glue dash sheets.

Just attach small pieces to the corners and remove the protective backing.

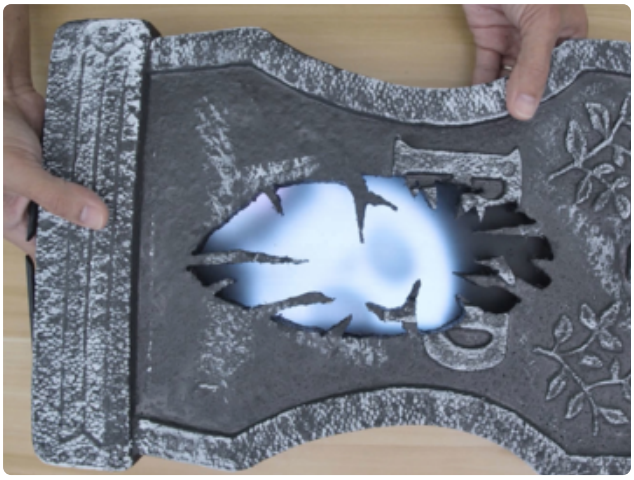
Then just line up the acrylic with the edges and stick it over the display.

This makes the display show up much better on camera so you can have it in the background of your video calls.



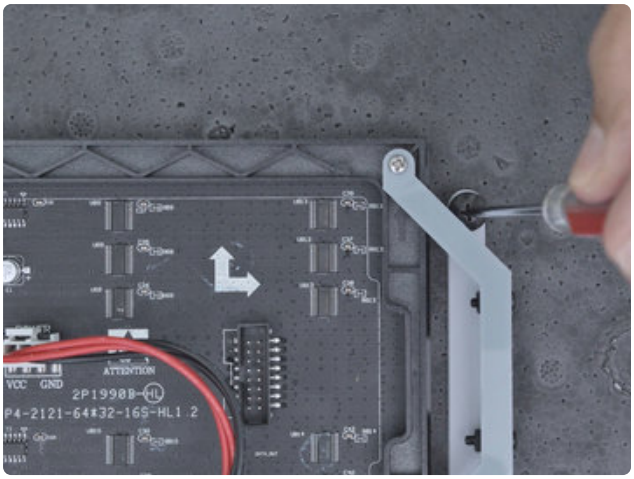
Attach Brackets

Use two M3 screws to attach the brackets to the back of the matrix display.



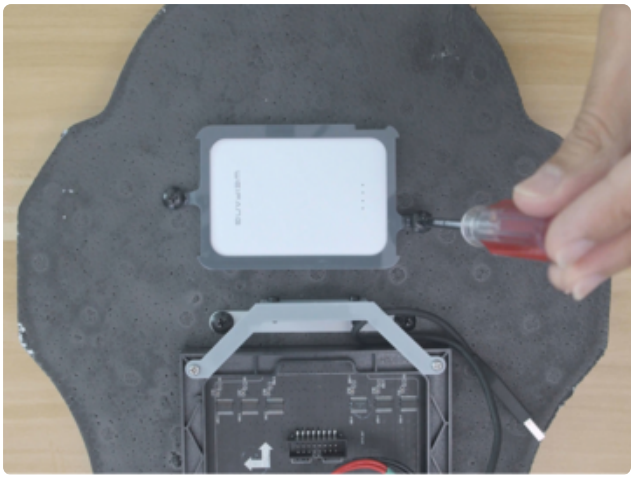
Align Matrix

Carefully align the Matrix display to the cut out on the tomb. Mark the position of each bracket.



Attach to Tomb

Use four M5 screws to secure the matrix display to the tombstone.



Mount Battery

Position the battery and battery bracket close to the Matrix Portal board.

Secure the battery bracket with two more M5 screws.

Complete

You can hold the tombstone up with a panavise or print out our tombstone feet below!

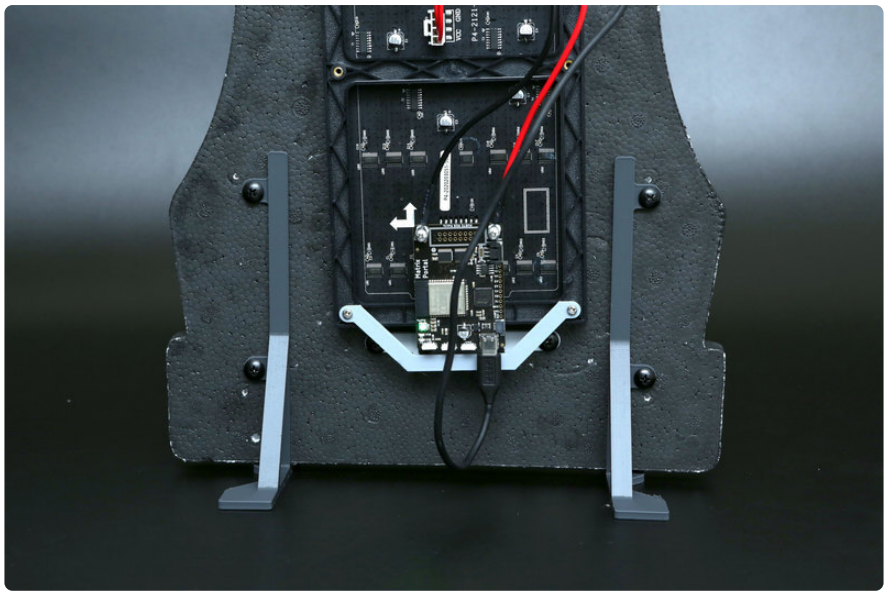
Tombstone Feet

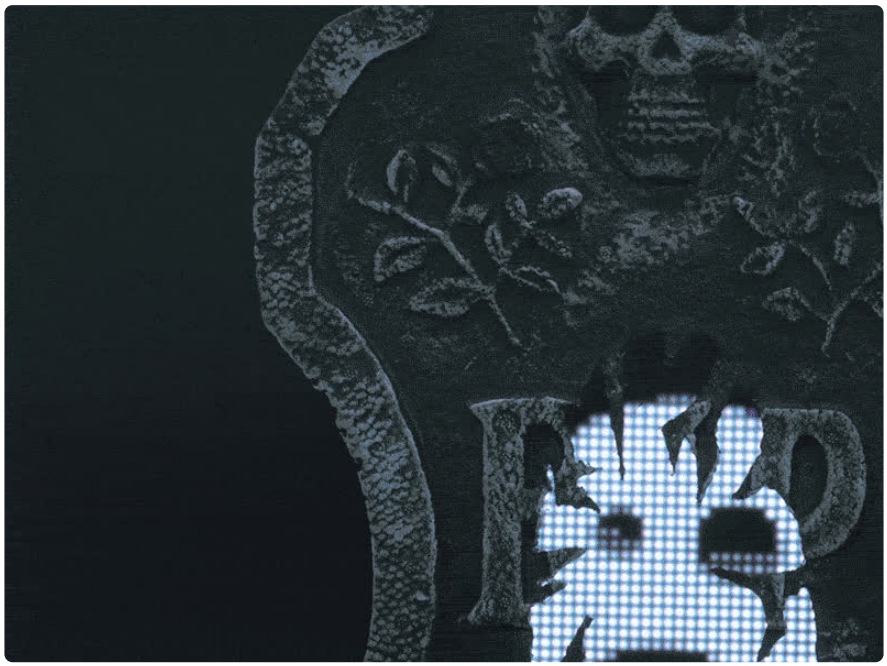
Use short wood screws to secure the two feet to the foam tombstone. Each foot has two mounting tabs. The mounting holes are 6mm in diameter with 92mm spacing distance.

4x #12 - 1/2in wood screws button head (https://adafru.it/NYF)

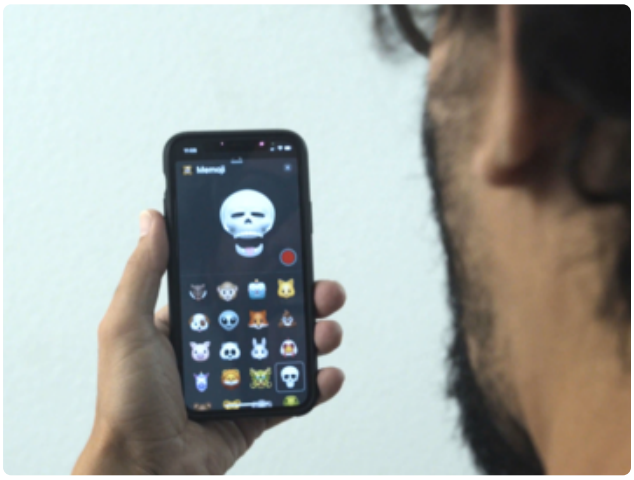
Matrix Display Feet

https://adafru.it/NTD





Custom Bitmaps

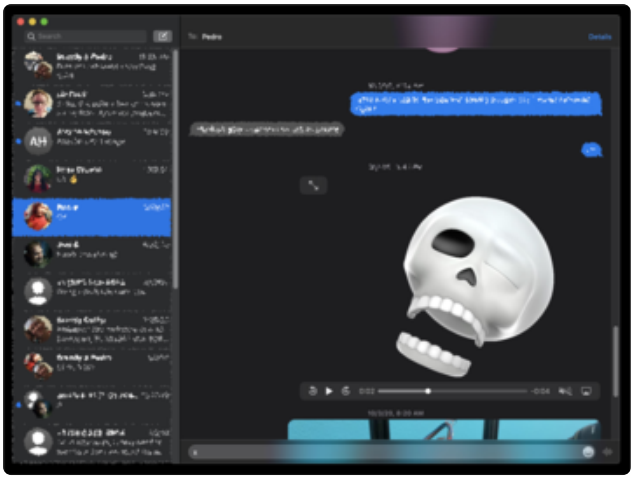


Memoji's

You can use the Memoji feature on iOS devices to create a custom animation. Record your facial expression to puppeteer a 3D character. We used the skull and ghost emoji's.

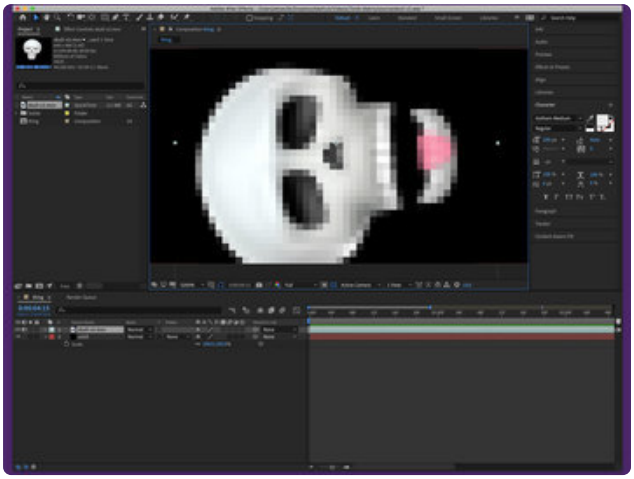
Using Memoji on iOS Devices

https://adafru.it/NUa



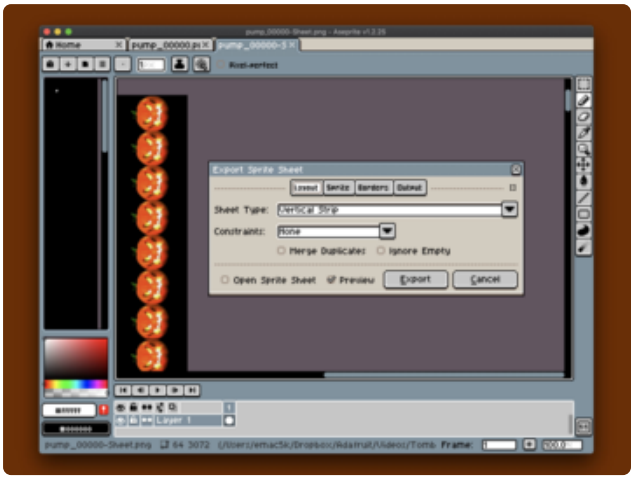
Recording & Saving Memoji's

Use the messages app on a capable iOS device to record and send the emoji. Save the recording to your desktop using iOS Message app. Drag and Drop the MP4 file to the desktop.



Export Image Sequence

Use an editing program such as Photoshop or After Effects to modify the scale or position. Set the canvas to 32 pixels height and 64 pixels wide. Set the frame rate between 12 and 24 frames. Export the MP4 video as an image sequence.



Sprite Sheet Bitmap

Use Aseprite to generate a sprite sheet image of the image sequence. Open the image sequence and export a sprite sheet image using vertical strip as the sheet type and bitmap as the image format.

Drag and drop the bitmap image to the bmps folder on the **CIRCUITPY** drive.

Walk through the sprite sheet animation learn guide for more information and instructions on using the Aseprite program.

Sprite Sheet Animation Guide

https://adafru.it/NTB

Download the zip file below to use our "ready-to-go" bitmaps. These were created for 32x64 RGB matrix in a portrait / vertical orientation.

tombstone-bitmaps.zip

https://adafru.it/NYE