LED Matrix Display for Bitmap Pixel Art and Animation
Created by John Park

Last updated on 2021-03-17 01:16:27 PM EDT
<table>
<thead>
<tr>
<th>Guide Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>Parts</td>
<td>3</td>
</tr>
<tr>
<td>Install CircuitPython</td>
<td>6</td>
</tr>
<tr>
<td>Set up CircuitPython Quick Start!</td>
<td>6</td>
</tr>
<tr>
<td>Further Information</td>
<td>6</td>
</tr>
<tr>
<td>CircuitPython Setup</td>
<td>8</td>
</tr>
<tr>
<td>Adafruit CircuitPython Bundle</td>
<td>8</td>
</tr>
<tr>
<td>Code the Pixel Art Display</td>
<td>9</td>
</tr>
<tr>
<td>Libraries</td>
<td>9</td>
</tr>
<tr>
<td>Text Editor</td>
<td>9</td>
</tr>
<tr>
<td>Code</td>
<td>9</td>
</tr>
<tr>
<td>Pixel Art Specs</td>
<td>10</td>
</tr>
<tr>
<td>Art Collecting</td>
<td>11</td>
</tr>
<tr>
<td>Game Frame Art</td>
<td>11</td>
</tr>
<tr>
<td>Search</td>
<td>11</td>
</tr>
<tr>
<td>Itch.io Assets</td>
<td>12</td>
</tr>
<tr>
<td>How it Works</td>
<td>13</td>
</tr>
<tr>
<td>Libraries</td>
<td>13</td>
</tr>
<tr>
<td>Variables</td>
<td>13</td>
</tr>
<tr>
<td>Display/Pin Setup</td>
<td>13</td>
</tr>
<tr>
<td>Slideshow Setup</td>
<td>14</td>
</tr>
<tr>
<td>Advance!</td>
<td>14</td>
</tr>
<tr>
<td>Main Loop</td>
<td>14</td>
</tr>
<tr>
<td>Prep the MatrixPortal</td>
<td>16</td>
</tr>
<tr>
<td>Power Prep</td>
<td>16</td>
</tr>
<tr>
<td>Power Terminals</td>
<td>17</td>
</tr>
<tr>
<td>Panel Power</td>
<td>17</td>
</tr>
<tr>
<td>Board Connection</td>
<td>18</td>
</tr>
<tr>
<td>LED Matrix Diffuser</td>
<td>20</td>
</tr>
<tr>
<td>LED Diffusion Acrylic</td>
<td>20</td>
</tr>
<tr>
<td>Measure and Cut the Plastic</td>
<td>20</td>
</tr>
<tr>
<td>Uglu Dashes</td>
<td>23</td>
</tr>
<tr>
<td>Stand</td>
<td>24</td>
</tr>
<tr>
<td>Sprite Sheet Animation</td>
<td>26</td>
</tr>
<tr>
<td>Animation Basics</td>
<td>26</td>
</tr>
<tr>
<td>Game Animation Frames</td>
<td>28</td>
</tr>
<tr>
<td>Walk Across the Screen</td>
<td>29</td>
</tr>
<tr>
<td>Sprite Sheet</td>
<td>29</td>
</tr>
<tr>
<td>Sprite Sheet</td>
<td>30</td>
</tr>
<tr>
<td>LED Matrix Version</td>
<td>34</td>
</tr>
<tr>
<td>Conversion Process Video</td>
<td>34</td>
</tr>
<tr>
<td>Code the Sprite Sheet Animation Display</td>
<td>35</td>
</tr>
<tr>
<td>Code</td>
<td>35</td>
</tr>
<tr>
<td>Sprite Sheet Specs</td>
<td>35</td>
</tr>
<tr>
<td>How it Works</td>
<td>38</td>
</tr>
<tr>
<td>Libraries</td>
<td>38</td>
</tr>
<tr>
<td>Variables</td>
<td>38</td>
</tr>
<tr>
<td>Setup</td>
<td>38</td>
</tr>
<tr>
<td>Image Loading Function</td>
<td>39</td>
</tr>
<tr>
<td>Main Loop</td>
<td>41</td>
</tr>
</tbody>
</table>

© Adafruit Industries  
https://learn.adafruit.com/pixel-art-matrix-display  
Page 2 of 42
Overview

Build your own pixel art display in order to view some of your favorite retro, low-rez artwork! We'll use .bmp format raster graphics files formatted for the 64x32 pixel RGB LED matrix display.

You can create your own pixel art, or scour the web for some favorites.

This project uses the Matrix Portal to show the pixel art on the display. It runs CircuitPython and uses the Slideshow library to make it easy!

You can play the slideshow and let it loop, or press the Matrix Portal's UP and DOWN buttons to pause/resume the loop and pick a particular image you're dying to see!

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

Your browser does not support the video tag.
Black LED Diffusion Acrylic Panel 12" x 12" - 0.1" / 2.6mm thick
A nice whooppin' 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
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

Add to Cart
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

USB cable - USB A to Micro-B
This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...
$2.95
In Stock
Add to Cart
Install CircuitPython

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

https://adafruit.it/Nte

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! :)

© Adafruit Industries

https://learn.adafruit.com/pixel-art-matrix-display
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://adafruit.com/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
Code the Pixel Art Display

Libraries
We'll need to make sure we have these libraries installed. (Check out this link (https://adafru.it/ABU) on installing libraries if needed.)

- adafruit_bus_device
- adafruit_debouncer
- adafruit_esp32spi
- adafruit_matrixportal
- adafruit_requests
- adafruit_slideshow
- neopixel

Please note, any code errors that reference issues with ".py" libraries can be referring to the ".mpy" files, there is no distinction made in the error message between the two.

Text Editor
Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in this guide (https://adafru.it/ANO).

Alternatively, you can use any text editor that saves simple text files.

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 pixel_art_slideshow.py from the zip file and place on the CIRCUITPY drive, then rename it to code.py.

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 graphics .bmp files.

The program will automatically use any .bmp files in the /bmps directory. Make sure they have legal names (no spaces or weird characters!) and are a maximum of 64x32 pixels. 16-bit or 24-bit both work fine.
Pixel Art Specs

If you want to create your own artwork for display, these are the specifications to follow:

- Images should be a maximum of 32 pixels high
- Images can be up to 64 pixels wide
- Colors are 16-bit or 24-bit RGB
- Save files as .bmp format

We've found that crisp images (not too much anti-aliasing) work best.

We have a whole page on Pixel Art Fundamentals (https://adafru.it/E1B) here!
You can use nearly any paint program, but dedicated pixel art programs work best, such as Aseprite (https://adafruit.it/ECC) or the free pixel art app Piskel (https://adafruit.it/ECB).

Art Collecting
If you want to search for ready-made art to display, here are some tips.

Game Frame Art
The excellent Game Frame (https://adafruit.it/Nla) product by Jeremy Williams pioneered the pixel art frame. You can check the LedSeq Game Frame forums (https://adafruit.it/Nlb) for art gallery submissions. Although these tend to be 16x16 pixel images, you can scale them up (don't interpolate!) for use on the Matrix Portal.

Jeremy has also graciously made the code and art for Game Frame available for free download here (https://adafruit.it/NIC).

Search
Another great tip from LedSeq is to use a Google image search to return results of "pixel art" or "video game" at a specific resolution, either 64x32 or 32x32:

Itch.io Assets

Many talented artists post their pixel art game assets to Itch.io (https://adafruit.it/Nld) where you can name your price in many cases for amazing artwork!

Shown here are the excellent PIPOYA FREE RPG Character Sprites (https://adafruit.it/Nle).

---

""" Slideshow Example using the Matrix Portal and 64 x 32 LED matrix display  
Written by Melissa LeBlanc-Williams for Adafruit Industries  
Images smaller than 64 pixel width will be aligned alternating left or right  
Press Up button to pause/resume Slideshow  
Press Down button to advance  
"""

import time
import board
from digitalio import DigitalInOut, Pull
from adafruit_matrixportal.matrix import Matrix
from adafruit_slideshow import SlideShow, PlayBackDirection, HorizontalAlignment
from adafruit_debouncer import Debouncer

IMAGE_DURATION = 3
IMAGE_FOLDER = "/bmps"

# --- Display setup ---
matrix = Matrix(bit_depth=6)
display = matrix.display

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)

align_right = True
auto_advance = True

slideshow = SlideShow(
    display,
    None,
    folder=IMAGE_FOLDER,
    order=0,
    auto_advance=False,
    fade_effect=False,
    dwell=IMAGE_DURATION,
    h_align=HorizontalAlignment.RIGHT,
last_advance = time.monotonic()

def advance():
    # pylint: disable=global-statement
    global align_right, last_advance
    align_right = not align_right
    if align_right:
        slideshow.h_align = HorizontalAlignment.RIGHT
    else:
        slideshow.h_align = HorizontalAlignment.LEFT
    last_advance = time.monotonic()
    slideshow.advance()

while True:
    if auto_advance and time.monotonic() > last_advance + IMAGE_DURATION:
        advance()
    button_down.update()
    button_up.update()
    if button_up.fell:
        auto_advance = not auto_advance
    if button_down.fell:
        slideshow.direction = PlayBackDirection.FORWARD
        advance()

How it Works

Libraries

Here's how the code works. First we import the `time`, `board`, `digitalio`, `adafruit_matrixportal`, `adafruit_slideshow`, and `adafruit_debouncer` libraries.

Variables

Then, we'll set the variables for the `IMAGE_DURATION` in seconds (you can adjust this as you like), as well as the path to the image folder.

```
import time
import board
from digitalio import DigitalInOut, Pull
from adafruit_matrixportal.matrix import Matrix
from adafruit_slideshow import SlideShow, PlayBackDirection, HorizontalAlignment
from adafruit_debouncer import Debouncer

IMAGE_DURATION = 3
IMAGE_FOLDER = "/bmps"
```

Display/Pin Setup

Next, we set up the display and the pins used for the buttons.

```
matrix = Matrix(bit_depth=6)
display = matrix.display

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)
```
**Slideshow Setup**

The alignment of images narrower than 64 pixels wide will alternate, initially with right-side alignment. The `auto_advance` state is set to true initially, but can be changed by pressing the UP button later.

The slideshow is set up next using the settings shown here.

```python
align_right = True
auto_advance = True

slideshow = SlideShow(
    display,
    None,
    folder=IMAGE_FOLDER,
    order=0,
    auto_advance=False,
    fade_effect=False,
    dwell=IMAGE_DURATION,
    h_align=HorizontalAlignment.RIGHT,
)
last_advance = time.monotonic()
```

**Advance!**

The `advance()` function will be used to drive the `slideshow.advance()` command using the alternating right/left alignment and auto-advance features.

```python
def advance():
    if auto_advance and time.monotonic() > last_advance + IMAGE_DURATION:
        advance()
    button_down.update()
    button_up.update()
    if button_up.fell:
        auto_advance = not auto_advance
    if button_down.fell:
        slideshow.direction = PlayBackDirection.FORWARD
        advance()
```

**Main Loop**

Here's what happens during the main loop of the program:

- Check if `auto_advance` is on, and if the `time.monotonic()` value is bigger than the `last_advance` time plus the `IMAGE_DURATION` time. If so, `advance()` to the next image!
- Check for button presses:
  - If the UP button is pressed, toggle the auto_advance state
  - If the DOWN button is pressed, manually advance to the next image

```python
while True:
    if auto_advance and time.monotonic() > last_advance + IMAGE_DURATION:
        advance()
    button_down.update()
    button_up.update()
    if button_up.fell:
        auto_advance = not auto_advance
    if button_down.fell:
        slideshow.direction = PlayBackDirection.FORWARD
        advance()
```
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.
Board Connection

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.
LED Matrix Diffuser

LED Diffusion Acrylic
You can add an LED diffusion acrylic faceplate (https://adafruit.it/MEF) to the your LED matrix display. (Pictured here with the ON AIR project (https://adafruit.it/MPE))

This can help protect the LEDs as well as enhance the look of the sign both indoors and out by reducing glare and specular highlights of the plastic matrix grid.

Measure and Cut the Plastic
You can use the sign to measure and mark cut lines on the paper backing of the acrylic sheet.

Then, use a tablesaw or bandsaw with a fine toothed blade and a guide or sled to make the cuts.

Note: it is possible to score and snap acrylic, but it can be very tricky to get an even snap without proper clamping.
Peel away the paper backing from both sides and set the acrylic onto your matrix display.
Uglu Dashes

The best method we've found for adhering acrylic to the matrix display is to use Uglu Dashes clear adhesive rectangles from Pro Tapes [https://adafruit.it/NcP]. They are incredibly strong (although can be removed if necessary), easy to apply, and are invisible once attached.

Use one at each corner and one each at the halfway point of the long edges, then press the acrylic and matrix panel together for about 20 seconds.

Here you can see the impact of using the diffusion acrylic. (Pictured here with the ON AIR sign project)
Stand
A very simple and attractive way to display your matrix is with the adjustable bent-wire stand (https://adafruit.it/MPF).
Alternately, you can use a frame, 3D printed brackets (https://adafruit.it/MZf), tape, glue, or even large binder clips to secure the acrylic to the sign and then mount it on on a wall, shelf, or display cabinet.

These mini-magnet feet (https://adafruit.it/MZA) can be used to stick the sign to a ferrous surface.
Sprite Sheet Animation
Thinking about game sprites naturally leads us to think about sprite animation. Let's add that now!
Traditional 2D animation in games is usually implemented with something called a sprite sheet, so we'll do the same here!

Before we get started with sprite animation, it's important to have a solid foundation in static pixel art creation. If needed, check out this page on Pixel Art Fundamentals (https://adafru.it/El8) first.

Animation Basics
Animation is a HUGE topic! It can be overwhelming, in fact, to consider all there is that goes into creating beautiful animation. The good thing is that we can keep it very simple and still learn a lot and get going fast.

At its most basic, animation is the illusion of motion our brains trick us into perceiving when we view multiple still frames of artwork played back in quick succession. A flipbook you can doodle in the corner of your notebook is a great example of how this works and how simple it can be.
Typical film animation is created with somewhere between 12 and 24 unique frames per second, which can give it an incredibly fluid look, but is a heck of a lot of work, too!
Game Animation Frames

Pixel sprite animation used in games is normally created with many fewer frames of animation. This means we can typically get away with creating just a handful of unique sprites to convey the action.

One cycle of a character walking may contain just six frames/sprites in a simple game style, and up to twelve or so frames for a more fluid style.

Here's a terrific example from Pedro Medeiros's MiniBoss Studios tutorial set available here (https://adafru.it/EI9).
Let's take that example and animate it to walk across the screen. I'll use Aseprite to create the animation, adding left-to-right translation so it doesn't happen standing in place, then we'll look at how to turn it into a sprite sheet for use on the Matrix Portal.

![Walk Across the Screen](image)

Walk Across the Screen

Here I've taking that miniboss example page and used Aseprite to re-create the animation, adding left-to-right translation so it doesn't happen standing in place.

When exported as a .gif animation we can play it in the browser as shown below.

![Sprite Sheet](image)

Sprite Sheet

Now, let's create a sprite sheet for use on the Matrix Portal display. A sprite sheet is a lot like a traditional film strip, it is a long (or wide, or gridded) strip of images, with each image representing a frame of animation.

In Aseprite, I've used the File > Export Sprite Sheet command to take the entire animation and stitch it into a long, vertical sprite sheet.

![Export Sprite Sheet](image)
Now, let's create a sprite sheet for use on the Matrix Portal display. A sprite sheet is a lot like a traditional film strip, it is a long (or wide, or gridded) strip of images, with each image representing a frame of animation.

In Aseprite, I've used the File > Export Sprite Sheet command to take the entire animation and stitch it into a long, vertical sprite sheet.
Get ready to do some scrolling! Because, this is what the exported sprite sheet looks like:
Now, when the Matrix Portal uses the TileGrid code, it will essentially march through and display each "frame" of this single image in order to play back the animation! You can think of it like a film strip moving in front of the bulb and shutter of a film projector.

**LED Matrix Version**

Here’s a version of the walk that I’ve adapted for the LED Matrix Portal display. It is now 64x32 pixels in size and on a black background to keep the current draw low.

[https://adafruit.it/NPD](https://adafruit.it/NPD)

And here it is being played on the LED matrix display.

You can turn any set of sequential frames into a sprite sheet, you don't necessarily need to create your animation from scratch in a pixel art program. Here’s a video showing how to convert rendered frames from 3D animation software:

**Conversion Process Video**

Next, we'll code the Matrix Portal for sprite sheet animation playback.
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 matrix_sprite_animation.py from the zip file and place on the CIRCUITPY drive, then rename it to code.py.

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
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(max_size=1)
matrix.display.show(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
    current_image = Image.load(SPRITESHEET_FOLDER + file_list[frame_count])
    current_frame = 0
    current_loop = 0

    # Set up the pixels
    pixels = ImageSprite.current_image.pixel_array
    pixels = np.float32(pixels)
    pixels[0] = 0.2
    pixels[1] = 0.4
    pixels[2] = 0.6
    pixels[3] = 0.8

    # Set the pixels
    matrix.display.show(pixels)

    # Advance to the next frame
    current_frame = (current_frame + 1) % 3
    if current_frame == 0:
        # Increase the loop count
        current_loop += 1
        if current_loop == AUTO_ADVANCE_LOOPS:
            # Auto advance
            current_loop = 0
            frame_count = (frame_count + 1) % len(file_list)
            load_image()
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

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:
How it Works

Libraries
Here's how the code works. First we import necessary libraries including `displayio` 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:

```python
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.show(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 Function
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 display 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 >= 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 >= 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.

```python
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)
```