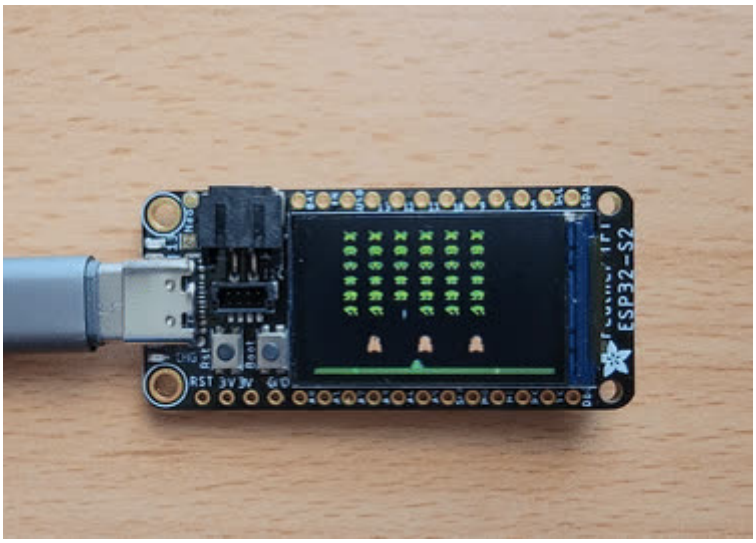
Playing Animated GIF Files in CircuitPython

Created by Anne Barela



https://learn.adafruit.com/using-animated-gif-files-in-circuitpython

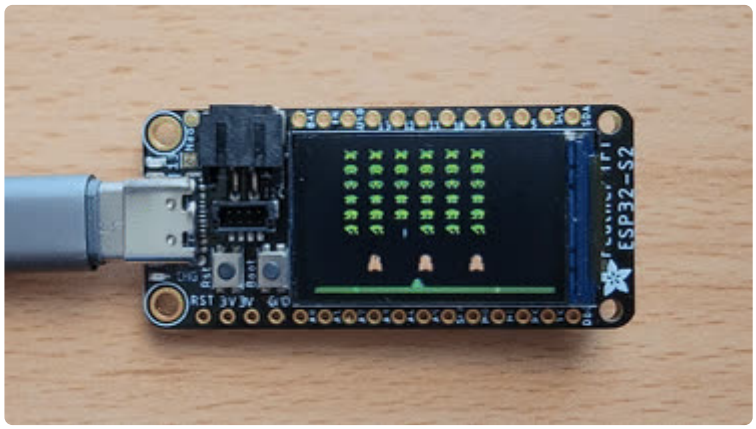
Last updated on 2024-06-03 03:48:53 PM EDT

Table of Contents

Overview

Frequent use of animated GIF files dates back to Netscape Navigator in the 1990's (remember twirling "under construction" signs for web sites?). These days, animated GIF files are most often used for meme display or in short video clips where a video file might not be playable.

The resources needed to play animated GIFs are lower than videos, but have often been more than what a microcontroller could provide.



Playing animated GIF image files using a microcontroller is a tough proposition. Here are the factors which need to be considered for a good experience:

- Microcontroller speed
- Microcontroller memory available
- A speedy interface between microcontroller memory and a display
- Animated GIF file size

Some more modern microcontrollers with a larger amount of memory and fast access to a local display can meet the requirements for animated GIF playback, as long as

the file size and frame rate (the time between frames in the changing image) is not too small.

This guide will demonstrate displaying animated GIF files on two microcontroller boards with integrated displays. The Espressif ESP32-S2 and Microchip SAMD51 are modern processors capable of displaying GIF files at reasonable rates.

The CircuitPython `gifio` module is an addition to CircuitPython 8.1.0 and later versions providing GIF playback capability.

Parts

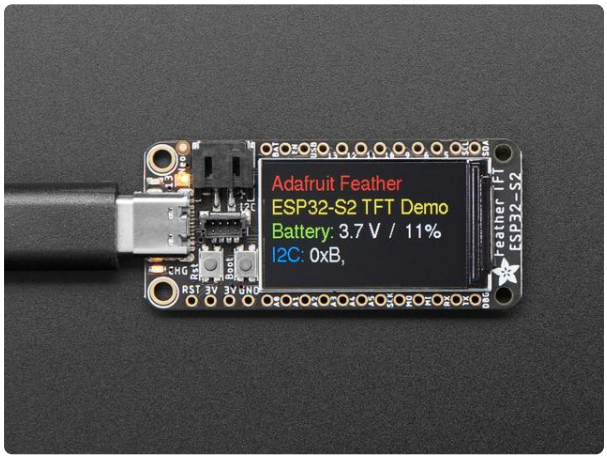
This guide will use an Adafruit ESP32-S2 TFT Feather and an Adafruit PyPortal to demonstrate animated GIF playback. Only one of the boards is needed to try this process. The USB cable shown works with both USB micro B and USB C. Be sure you use a known good data+power cable.

You can choose another board - note the code may be slightly different from the code for these two boards in this guide. And also, if the other board does not have a modern, capable processor, GIF playback will be impaired or unacceptable.

Adafruit PyPortal - CircuitPython Powered Internet Display

PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...

https://www.adafruit.com/product/4116



Adafruit ESP32-S2 TFT Feather - 4MB Flash, 2MB PSRAM, STEMMA QT

We've got a new machine here at Adafruit, it can uncover your deepest desires. Don't believe me? I'll turn it on right now to prove it to you! What, you want unlimited...

https://www.adafruit.com/product/5300

[USB 3-in-1 Sync and Charge Cable - Micro B / Type-C / Lightning]
As USB technology evolves you'll want the One Cable To Sync/Charge All Things (or, at least, portable devices with Micro-B, Type C, or Lightning ports) and this...
https://www.adafruit.com/product/3679

---

gifio

`gifio` is the new animated GIF module added to CircuitPython in version 8.1.0-beta.1 and later builds. It will not work with earlier versions. It provides methods to write and read animated GIF files. This guide will show how to read files and display them on an attached LCD display.

Constraints

The `gifio` module is limited to working with images that are 320 pixels wide or less. The height is not specifically limited but very tall GIF's may run out of memory. Smaller dimensions are fine and will likely speed code execution on a smaller data set.

The GIF speed (time between frames) should be at least the time it takes the microcontroller to process and display one frame. You can benchmark this on your setup but generally the time between frames should be 4/100th of a second or longer. 7/100th of a second second or longer is likely better. A check in the code is best to ensure the GIF display function is passed a value greater or equal to zero, zero being "as fast as possible".

The delay between frames is specified in 1/100ths of a second as that is what the file specification uses. The largest delay allowed between frames is 655 seconds.

Usage

Our two example boards have integrated displays, such that `board.DISPLAY` is defined and connected via a defined display bus.

There are two methods for writing to the display: using the CircuitPython `displayio` module and writing to the display directly. There are times where one over the other may be desired. If using an entire display, the direct method is likely fastest, while if animation is in a `displayio.Tilegrid`, and a partial part of a display will be used for playback, `displayio` will be the route, although there will be slowdowns due to display code overhead to consider.

The main documentation for gifio is on Read The Docs at https://docs.circuitpython.org/en/latest/shared-bindings/gifio/index.html

gifio.OnDiskGif Constructor

This constructor creates an `OnDiskGif` object. You use that object to load one frame of a GIF image file into memory at a time.

The code can be used in cooperation with `displayio` but this mode is slower than writing directly to the display.

Parameters for the function are:

- File name: (required) The name of a file in the CircuitPython file system (most often on the **CIRCUITPY** drive for speed).

The `next_frame()` function is used to advance to the next frame in the animated GIF.

The `deinit()` function is used in conjunction with ensuring memory allocated by the `OnDiskGif` object is freed properly (important if multiple files are displayed or other memory activity is done.

Issues

The library may not take into account the file size, so long GIFs should be cut into segments perhaps and read sequentially using the multi GIF method.

Error with memory allocation are indicative of a file size issue. Try a file with a smaller size for testing.

GIF files may be edited for size, frame duration, and length in a GIF file editor. The free web-based https://ezgif.com/ (https://adafru.it/Ekt) does well.

CircuitPython

You will want to upload the latest version of CircuitPython for your microcontroller board. At the time of this guide, it is 8.1.0-beta.1 which is the minimum version `gifio` works on with the presented code.

Go to circuitpython.org (https://adafru.it/EFq), go to your board and download the .UF2 file under downloads (for microcontrollers) or Blinka (for single board computers).

For the two chosen boards, here are direct links:

Adafruit PyPortal

https://adafru.it/Egk

Adafruit ESP32-S2 TFT Feather

https://adafru.it/YwB

Flashing the New Version of CircuitPython

Each Adafruit board has a handy guide on use at https://learn.adafruit.com/ (https://adafru.it/dlu). The instructions for the two boards:

Flash CircuitPython on the Adafruit PyPortal

https://adafru.it/EnM

Flash CircuitPython on the Adafruit ESP32-S2 TFT Feather

https://adafru.it/YNE

When you plug your board into your computer via a known good USB data+power USB cable, a flash drive named **CIRCUITPY** should appear in your computer File Explorer or Finder (depending on your operating system). The **boot_out.txt** will show your version of CircuitPython when you open it. Verify it is at least 8.1.0-beta.1:

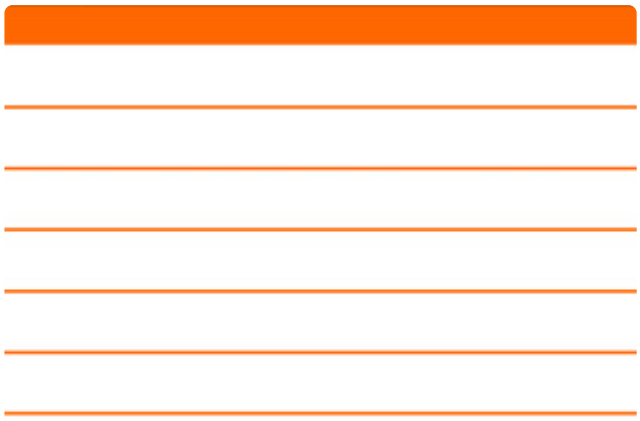
```
Adafruit CircuitPython 8.1.0-beta.1 on 2023-03-30; Adafruit PyPortal with samd51j20
Board ID:pyportal
UID:9D6BC039354639532020203 10A0E08FF
```

```
Adafruit CircuitPython 8.1.0-beta.1 on 2023-03-30; Adafruit Feather ESP32-S2 TFT
with ESP32S2
Board ID:adafruit_feather_esp32s2_tft
UID:C7FD1A59112D
```

If your version of CircuitPython is less than 8.1.0-beta.1, please try to upload and flash a version at least 8.1.0-beta.1 (higher versions like 8.1.1, 8.2.0 or 9.0.0 are examples of versions that would be ok).

Next, to get the code and sample animated GIF files onto your board.

PyPortal



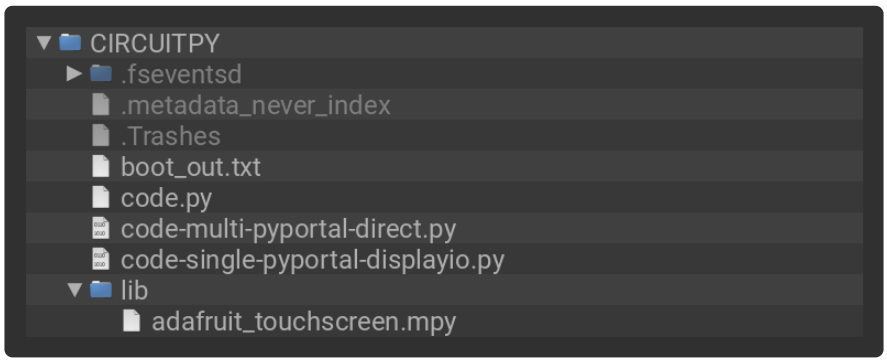
The Adafruit PyPortal with a 320x240 pixel display makes a great GIF-playing platform with its snappy SAMD51 M4 processor.

Download the Project Bundle

The example uses a code file and a sample animated GIF image. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Connect a PyPortal to your computer via a known good USB data+power cable. It should show up as a thumb drive named **CIRCUITPY**.

Using File Explorer/Finder (depending on your Operating System), drag the contents of the unzipped bundle directory onto your board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.



The code below displays a single GIF image. To run it, copy one of the GIF images to the name **sample.gif**. Copy the file **code-single-pyportal-displayio.py** to **code.py**.

```
# SPDX-FileCopyrightText: 2023 Anne Barela for Adafruit Industries
#
# SPDX-License-Identifier: MIT
#
# gifio demo for the Adafruit PyPortal - single file
#
# Documentation:
#   https://docs.circuitpython.org/en/latest/shared-bindings/gifio/
# Updated 4/5/2023
#
import time
import gc
import board
import gifio
import displayio
import adafruit_touchscreen

display = board.DISPLAY
splash = displayio.Group()
display.root_group = splash

# Pyportal has a touchscreen, a touch stops the display
WIDTH = board.DISPLAY.width
HEIGHT = board.DISPLAY.height
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_XL, board.TOUCH_XR,
                                      board.TOUCH_YD, board.TOUCH_YU,
                                      calibration=((5200, 59000), (5800, 57000)),
                                      size=(WIDTH, HEIGHT))

odg = gifio.OnDiskGif('/sample.gif')

start = time.monotonic()
next_delay = odg.next_frame()  # Load the first frame
end = time.monotonic()
call_delay = end - start

# Depending on your display the next line may need Colorspace.RGB565
#   instead of Colorspace.RGB565_SWAPPED
face = displayio.TileGrid(odg.bitmap,
                          pixel_shader=displayio.ColorConverter
                          (input_colorspace=displayio.Colorspace.RGB565_SWAPPED))
splash.append(face)
board.DISPLAY.refresh()

# Play the GIF file until screen is touched
while True:
    time.sleep(max(0, next_delay - call_delay))
    next_delay = odg.next_frame()
    if ts.touch_point is not None:
        break
# End while
# Clean up memory
odg.deinit()
odg = None
gc.collect()
```

How It Works

First the code imports all the libraries needed for the example. Only the `Adafruit_Touchscreen` library needs an external file, the rest are included inside CircuitPython. Ensure **adafruit_touchscreen.mpy** is in the **/lib** folder. Using `displayio`, the CircuitPython display library, the screen is defined (the parameters

are in the PyPortal definition file so they don't need to be looked up, things like the screen width 320 px and height 240 px).

To signal user input, the touchscreen is used. This isn't necessary in most applications but having a convenient input method can be handy, especially if the PyPortal is in a case.

The file **sample.gif** is opened via `gifio`. You may get an error that **sample.gif** is not found, please copy one of the example GIF files to **sample.gif**.

The first frame is loaded using `odg.next_frame()`. The time it takes to call the function is noted to make an adjustment for the time between frames later.

A `displayio` `TileGrid` is made consisting of the first frame and the entire display.

Finally there is a loop which constantly fetches frames and displays them with a file-defined pause between frames. As calling the `next_frame` takes time (measured earlier), that time is subtracted from the frame rate specified in the file.

If the screen is touched, the memory is cleaned up and the program ends.

PyPortal Multiple GIFs

Displaying more than one GIF isn't too much different than a single GIF. This page provides some tips. Memory management becomes more important.

This example also shows writing directly to the built-in display, which can save time in displaying frames for a smoother feel. The trade-off is researching how to write directly to the display as that can vary from one display type to another.

Download the Project Bundle

The example uses one code file and three sample animated GIF images. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Hook the PyPortal to your computer via a known good USB data+power cable. It should show up as a thumb drive named **CIRCUITPY**.

Using File Explorer/Finder (depending on your Operating System), drag the contents of the uncompressed bundle directory onto your board's **CIRCUITPY** drive, replacing

any existing files or directories with the same names, and adding any new ones that are necessary.

The code below displays multiple GIFs, sequentially as you tap the screen. Copy the file **code-multi-pyportal-direct.py** to **code.py**. You can leave the names of the GIF files as they are.

```
# SPDX-FileCopyrightText: 2023 Anne Barela for Adafruit Industries
#
# SPDX-License-Identifier: MIT
#
# Play Multiple GIF files on a PyPortal
# Requires CircuitPython 8.1.0-beta.1 or later
# Updated 4/4/2023

import time
import gc
import os
import struct
import board
import gifio
import adafruit_touchscreen

# Get a dictionary of GIF filenames at the passed base directory
def get_files(base):
    allfiles = os.listdir(base)
    file_names = []
    for _, filetext in enumerate(allfiles):
        if not filetext.startswith("."):
            if filetext not in ('boot_out.txt', 'System Volume Information'):
                if filetext.endswith(".gif"):
                    file_names.append(filetext)
    return file_names

display = board.DISPLAY

# Take over display to drive directly
display.auto_refresh = False
display_bus = display.bus

# Pyportal has a touchscreen, a touch advances to next GIF file
WIDTH = board.DISPLAY.width
HEIGHT = board.DISPLAY.height
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_XL, board.TOUCH_XR,
                                      board.TOUCH_YD, board.TOUCH_YU,
                                      calibration=((5200, 59000), (5800, 57000)),
                                      size=(WIDTH, HEIGHT))
files = get_files("/")
for i in range(len(files)):

    odg = gifio.OnDiskGif(files[i])
    # Skip Feather GIFs if put on PyPortal
    if odg.width != WIDTH:
        print("File "+files[i]+" not right width, skipping\n")
        continue

    start = time.monotonic()
    next_delay = odg.next_frame()  # Load the first frame
    end = time.monotonic()
    call_delay = end - start

    # Display GIF file frames until screen touched (for PyPortal)
    while True:
        sleeptime = max(0, next_delay - call_delay)
```

```
        time.sleep(sleeptime)
        if ts.touch_point is not None:
            break
        next_delay = odg.next_frame()
        display_bus.send(42, struct.pack(">hh", 0, odg.bitmap.width - 1))
        display_bus.send(43, struct.pack(">hh", 0, odg.bitmap.height - 1))
        display_bus.send(44, odg.bitmap)
    # End while
    # Clean up memory
    odg.deinit()
    odg = None
    gc.collect()
# End for
```

How It Works

First the code imports all the libraries needed for the example. Only the `Adafruit_Touchscreen` library needs an external file, the rest are included inside CircuitPython. Ensure **adafruit_touchscreen.mpy** is in the **/lib** folder. The display is addressed directly through `display_bus`. The screen width of 320 px and height 240 px is retrieved through the internal board definition for the touchscreen definition.

The function `get_files()` gets the names of all the GIF files in the root (main) directory on the device. This is a convenience - if you only want a predefined list, make a dictionary named files with the filenames, like this:

```
files = ['/file1.gif', '/file2.gif']
```

To register user input, the touchscreen is used. This isn't necessary in most applications but having a convenient input method can be handy, especially if the PyPortal is in a case.

Each GIF file found on the PyPortal is opened via `gifio`. The first frame of the current file is loaded using `odg.next_frame()`. The time it takes to call the function is noted to make an adjustment for the time between frames later.

A `While` loop constantly fetches frames and displays them with a file-defined pause between frames. As calling the `next_frame` takes time (measured earlier), that time is subtracted from the frame rate specified in the file.

The frame contents are sent direct to the display via calls to `display_bus.send` rather than using `displayio`. This makes displaying frames faster, making for smoother playback.

If the screen is touched, the memory is cleaned up and the next file on the PyPortal is played until they all have been played and the program ends.

If you wish to put the GIF files in a subdirectory, change the line `files = get_files("/")` to your preferred directory name, for example `files = get_files("/Images")`.

Feather

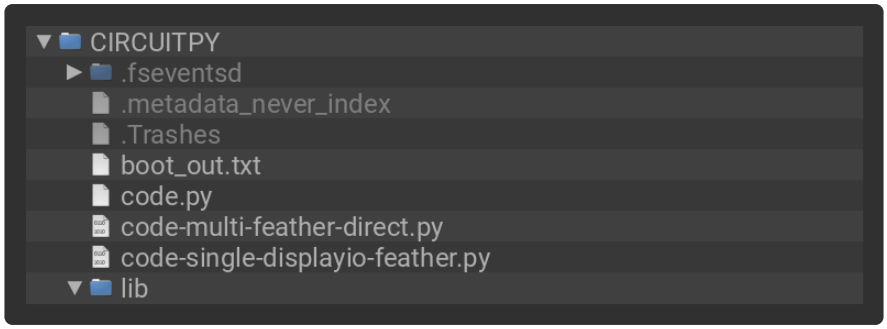
The Adafruit ESP32-S2 Feather TFT with a 240x135 pixel display makes for a compact GIF playing platform.

Download the Project Bundle

The examples use code files and sample animated GIF images. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Hook the Feather ESP32-S2 TFT to your computer via a known good USB data+power cable. It should show up as a thumb drive named **CIRCUITPY**.

Using File Explorer/Finder (depending on your Operating System), drag the contents of the uncompressed bundle directory onto your board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.



The code below displays a single GIF image. To run it, copy one of the GIF images to the name **sample.gif**. Copy the file **code-single-displayio-feather.py** to **code.py**.

```
# SPDX-FileCopyrightText: 2023 Anne Barela for Adafruit Industries
#
# SPDX-License-Identifier: MIT
#
# Play a single animated GIF file (display_bus "fast" method)
#
# Documentation:
#   https://docs.circuitpython.org/en/latest/shared-bindings/gifio/
# Updated 4/5/2023

import time
import gc
import board
import gifio
import digitalio
import displayio

display = board.DISPLAY
splash = displayio.Group()
display.root_group = splash

# Set BOOT button on ESP32-S2 Feather TFT to stop GIF
button = digitalio.DigitalInOut(board.BUTTON)
button.switch_to_input(pull=digitalio.Pull.UP)

# Open GIF file sample.gif
odg = gifio.OnDiskGif('/sample.gif')

start = time.monotonic()
next_delay = odg.next_frame()  # Load the first frame
end = time.monotonic()
overhead = end - start

face = displayio.TileGrid(
    odg.bitmap,
    pixel_shader=displayio.ColorConverter(
        input_colorspace=displayio.Colorspace.RGB565_SWAPPED
    ),
)
splash.append(face)
board.DISPLAY.refresh()

# Display repeatedly & directly.
while True:
    # Sleep for the frame delay specified by the GIF,
    # minus the overhead measured to advance between frames.
    time.sleep(max(0, next_delay - overhead))
    # If the BOOT button is pressed, stop the GIF
    if button.value is False:
        print("Button Press, Stop\n")
        break
    next_delay = odg.next_frame()
# End While - cleanup memory
odg.deinit()
odg = None
gc.collect()
```

How It Works

First the code imports all the libraries needed for the example. All are included inside CircuitPython. Using `displayio`, the CircuitPython display library, the screen is defined (the parameters are in the Feather definition file so they don't need to be looked up, things like the screen width: 240 px and height 135 px).

The file **sample.gif** is opened via `gifio`. You may get an error that **sample.gif** is not found, please copy one of the example GIF files to **sample.gif**.

The first frame is loaded using `odg.next_frame()`. The time it takes to call the function is noted to make an adjustment for the time between frames later.

A `displayio` `TileGrid` is made consisting of the first frame and the entire display.

Finally, there is a loop which constantly fetches frames and displays them with a file-defined pause between frames. As calling the `next_frame` takes time (measured earlier), that time is subtracted from the frame rate specified in the file.

If the BOOT button is pressed, the memory is cleaned up and the program ends. This is just a convenience and a user's program need not use a button. It is suggested cleaning up the memory after using `gifio`.

Feather Multiple GIFs

Displaying more than one GIF isn't too much different than a single GIF. This page provides some tips. Memory management becomes more important.

This example also shows writing directly to the built-in display, which can save time in displaying frames for a smoother feel. The trade-off is researching how to write directly to the display as that can vary from one display chipset to another.

Download the Project Bundle

The example uses the code file and two sample animated GIF images. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Hook the Feather to your computer via a known good USB data+power cable. It should show up as a thumb drive named **CIRCUITPY**.

Using File Explorer/Finder (depending on your Operating System), drag the contents of the uncompressed bundle directory onto your board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

The code below displays multiple GIFs, sequentially as you press the BOOT button. Copy the file **code-multi-feather-direct.py** to **code.py**.

```
# SPDX-FileCopyrightText: 2023 Anne Barela for Adafruit Industries
#
# SPDX-License-Identifier: MIT
#
# Play Multiple GIF files on an ESP32-S2 Feather TFT
# Requires CircuitPython 8.1.0-beta.1 or later
# Updated 4/4/2023

import time
import gc
import os
import struct
import board
import gifio
import digitalio

# Get a dictionary of GIF filenames at the passed base directory
def get_files(base):
    allfiles = os.listdir(base)
    file_names = []
    for _, filetext in enumerate(allfiles):
        if not filetext.startswith("."):
            if filetext not in ('boot_out.txt', 'System Volume Information'):
                if filetext.endswith(".gif"):
                    file_names.append(filetext)
    return file_names

# Set BOOT button on ESP32-S2 Feather TFT to advance to next GIF
button = digitalio.DigitalInOut(board.BUTTON)
button.switch_to_input(pull=digitalio.Pull.UP)

display = board.DISPLAY
display.auto_refresh = False

COL_OFFSET = 40  # The Feather TFT needs to have the display
ROW_OFFSET = 53  # offset by these values for direct writes

files = get_files("/")
for i in range(len(files)):

    odg = gifio.OnDiskGif(files[i])
    # Skip PyPortal GIFs if put on ESP32-S2 Feather TFT
    if odg.width != board.DISPLAY.width:
        print("File "+files[i]+" not right width, skipping\n")
        continue

    start = time.monotonic()
    next_delay = odg.next_frame()  # Load the first frame
    end = time.monotonic()
    call_delay = end - start

    # Display GIF file frames until screen touched (for PyPortal)
    while True:
        sleeptime = max(0, next_delay - call_delay)
        time.sleep(sleeptime)
```

```
        # If the BOOT button is pressed, advance to next GIF file
        if button.value is False:
            print("Button Press, Advance\n")
            break
        next_delay = odg.next_frame()
        display.bus.send(42, struct.pack(">hh", COL_OFFSET,
                         odg.bitmap.width - 1 + COL_OFFSET))
        display.bus.send(43, struct.pack(">hh", ROW_OFFSET,
                         odg.bitmap.height - 1 + ROW_OFFSET))
        display.bus.send(44, odg.bitmap)
    # End while
    # Clean up memory
    odg.deinit()
    odg = None
    gc.collect()
# End for
```

How It Works

First the code imports all the libraries needed for the example. All are included inside CircuitPython. The BOOT button on the Feather is used programmably. It cycles between GIF files in this example, so it is defined as an input as shown in the user guide for this board (https://adafru.it/18C5).

The display is addressed directly through `display.bus`.

The function `get_files()` gets the names of all the GIF files in the root (main) directory on the device. This is a convenience - if you only want a predefined list, make a dictionary named files with the filenames, like this:

```
files = ['/file1.gif', '/file2.gif']
```

To signal user input, the BOOT button is pressed. This isn't necessary in most applications but having a convenient input method can be handy.

Each GIF file found on the Feather (there are two demonstration files) is opened via `gifio`. The first frame of the current file is loaded using `odg.next_frame()`. The time it takes to call the function is noted to make an adjustment for the time between frames later.

A `While` loop constantly fetches frames and displays them with a file-defined pause between frames. As calling the `next_frame` takes time (measured earlier), that time is subtracted from the frame rate specified in the file.

The frame contents are sent direct to the display via calls to `display.bus.send` rather than using `displayio`. This makes displaying frames faster, making for smoother playback. Note for the Feather, which is different than the PyPortal, the

display needs X and Y offsets to get the images to display on the right place on the screen. Without them, the image would be tucked in the upper left corner.

When the BOOT button is pressed, the memory is cleaned up and the next file on the Feather is played until they all have been played and the program ends.

If you wish to put the GIF files in a subdirectory, change the line `files = get_files("/" )` to your preferred directory name, for example `files = get_files("/Images")`.