PiTFT Python + Pillow Animated Gif Player

Created by Melissa LeBlanc-Williams

https://learn.adafruit.com/pitft-linux-python-animated-gif-player

Last updated on 2023-08-29 04:19:06 PM EDT
# Table of Contents

## Overview
- Parts

## Python Setup
- Attaching
- Setup
- Python Installation of RGB Display Library
- DejaVu TTF Font
- Pillow Library
- NumPy Library
- Quickstart Button Test

## Python Code
- Full Code

## Using the Player
- Adding Animated Gifs
- Starting the Player
- Scrolling through Animated Gifs
Overview

Have you wanted to play animated Gifs on the Raspberry Pi without showing or taking over the entire desktop? Or maybe you are only wanting to SSH into the Pi and play some kind of animation without the fancy graphical environment. Or possibly, you just want to know how to play animated Gifs with Python and the Pillow library. If so, this is the guide for you.

This is an animated Gif player that runs on the Raspberry Pi and uses a PiTFT Display. It works kind of like a Slideshow player, but displays animated Gifs instead. Because this is written in pure Python, it won't quite be as speedy as a C-based player, but the smaller the resolution of display that you use, the better the performance.

Parts

To get started, you will need a Raspberry Pi. A Raspberry Pi 4 is recommended because it's pretty fast.

Raspberry Pi 4 Model B
The Raspberry Pi 4 Model B is the newest Raspberry Pi computer made, and the Pi Foundation knows you can always make a good thing better! And what could make the Pi 4 better...
https://www.adafruit.com/product/4297
You will also need a PiTFT to display the images. We recommend these smaller Mini PiTFT's since we can write to the display fairly fast.

**Adafruit Mini PiTFT 1.3" - 240x240 TFT Add-on for Raspberry Pi**
If you're looking for the most compact li'l color display for a Raspberry Pi (most likely a [https://www.adafruit.com/product/4484](https://www.adafruit.com/product/4484))

**Adafruit Mini PiTFT - 135x240 Color TFT Add-on for Raspberry Pi**
If you're looking for the most compact li'l color display for a Raspberry Pi (most likely a [https://www.adafruit.com/product/4393](https://www.adafruit.com/product/4393))

You could use larger 320x240 ones too for slower images.

**Adafruit PiTFT Plus 320x240 2.8" TFT + Capacitive Touchscreen**
Is this not the cutest little display for the Raspberry Pi? It features a 2.8" display with 320x240 16-bit color pixels and a capacitive touch overlay. That's right,... [https://www.adafruit.com/product/2423](https://www.adafruit.com/product/2423)
PiTFT Plus Assembled 320x240 2.8" TFT + Resistive Touchscreen
Is this not the cutest little display for the Raspberry Pi? It features a 2.8" display with 320x240 16-bit color pixels and a resistive touch overlay. The plate uses the high...
https://www.adafruit.com/product/2298

Adafruit PiTFT 2.2" HAT Mini Kit - 320x240 2.2" TFT - No Touch
The cute PiTFT got even more adorable with this little primary display for Raspberry Pi in HAT form! It features a 2.2" display with 320x240 16-bit color pixels. The HAT uses the...
https://www.adafruit.com/product/2315

You can also get the PiTFT in 3.2" or 2.4", but they don't come with buttons, so you’ll want to pick up some of those so you can scroll.

PiTFT Plus 320x240 3.2" TFT + Resistive Touchscreen
Is this not the cutest little display for the Raspberry Pi? It features a 3.2" display with 320x240 16-bit color pixels and a resistive touch overlay. The plate uses the high...
https://www.adafruit.com/product/2616
Adafruit PiTFT 2.4" HAT Mini Kit - 320x240 TFT Touchscreen
Is this not the cutest little display for the Raspberry Pi? It features a 2.4" display with 320x240 16-bit color pixels and a resistive touch overlay. The HAT uses the high speed...
https://www.adafruit.com/product/2455

Tactile Switch Buttons (6mm slim) x 20 pack
Slim clicky momentary switches are standard input "buttons" on electronic projects. These are half the width of classic 6mm tactile switches so they line up better on a...
https://www.adafruit.com/product/1489

Of course you'll need a power supply for your Pi.

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! Superfast with just the right amount of cable length to get your Pi 4...
https://www.adafruit.com/product/4298

You may want a case for your Pi.
If you have a case, these little header extensions are really useful.

Stacking Header for Pi A+/B+/Pi 2/Pi 3 - 2x20 Extra Tall Header
Stack multiple plates, breakouts etc onto your Raspberry Pi Model B+ with this custom-made extra-tall and extra-long 2x20 female header. The female header part has extra spaces to...
https://www.adafruit.com/product/1979

Python Setup

You can use this technique with any PiTFT, from the 240x135 mini PiTFT up to the 320x480. It isn't as fast as the kernel module support version but it'll work no matter what kernel/OS/version/etc and so is a lot less painful

Attaching

It's easy to use display breakouts with Python and the Adafruit CircuitPython RGB Display () module. This module allows you to easily write Python code to control the display.

Since the PiTFT comes preassembled, all you need to do is place it onto the GPIO pins.
Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported.

Connect the display as shown below to your Raspberry Pi.

Note this is not a kernel driver that will let you have the console appear on the TFT. However, this is handy when you can't install an fbtft driver, and want to use the TFT purely from 'user Python' code!

You can only use this technique with Linux/computer devices that have hardware SPI support, and not all single board computers have an SPI device so check before continuing.

For the 1.14":

For the 1.3":
Setup

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling SPI on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready!

If you have already installed the kernel module, you will need to remove it by editing your /boot/config.txt file before proceeding.

Python Installation of RGB Display Library

Once that's done, from your command line run the following commands:

- `pip3 install adafruit-circuitpython-rgb-display`
- `pip3 install --upgrade --force-reinstall spidev`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`
DejaVu TTF Font

Raspberry Pi usually comes with the DejaVu font already installed, but in case it didn’t, you can run the following to install it:

- `sudo apt-get install fonts-dejavu`

This package was previously called ttf-dejavu, so if you are running an older version of Raspberry Pi OS, it may be called that.

Pillow Library

We also need PIL, the Python Imaging Library, to allow graphics and using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

NumPy Library

A recent improvement of the RGB_Display library makes use of NumPy for some additional speed. This can be installed with the following command:

- `sudo apt-get install python3-numpy`

That's it. You should be ready to go.

Quickstart Button Test

This button test demo will test to make sure you have everything setup correctly. Go ahead and save the file to your Raspberry Pi in your home directory as rgb_display_minipitfttest.py.

```python
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
import digitalio
import board
from adafruit_rgb_display.rgb import color565
from adafruit_rgb_display import st7789
```

©Adafruit Industries
# Configuration for CS and DC pins for Raspberry Pi

cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = None
BAUDRATE = 64000000  # The pi can be very fast!

# Create the ST7789 display:
display = st7789.ST7789(
    board.SPI(),
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
    width=135,
    height=240,
    x_offset=53,
    y_offset=40,
)

backlight = digitalio.DigitalInOut(board.D22)
backlight.switch_to_output()
backlight.value = True

buttonA = digitalio.DigitalInOut(board.D23)
buttonB = digitalio.DigitalInOut(board.D24)
buttonA.switch_to_input()
buttonB.switch_to_input()

# Main loop:
while True:
    if buttonA.value and buttonB.value:
        backlight.value = False  # turn off backlight
    else:
        backlight.value = True  # turn on backlight
    if buttonB.value and not buttonA.value:  # just button A pressed
        display.fill(color565(255, 0, 0))  # red
    if buttonA.value and not buttonB.value:  # just button B pressed
        display.fill(color565(0, 0, 255))  # blue
    if not buttonA.value and not buttonB.value:  # none pressed
        display.fill(color565(0, 255, 0))  # green

Go ahead and run it with this command:

```
python3 rgb_display_minipitfttest.py
```

Once it is running, push the buttons. The top button should make the display light up Red, the bottom Blue, and pressing both at the same time should make it Green.

---

**Python Code**

Make sure you are running PIL 5.3.0 or later or you may get an error that there are missing functions.

Now that we have everything set up, let's dig into the example code. To start with, we have a lot of code imports. Let's go over those:

We start with `os` so that we can access a list of files in the operating system, `time` for doing delays, `digitalio` for button access and display pins, `board` so we can
see the pin definitions, and the Image and ImageOps modules for using Pillow. Then we import all of the displays so that we have a variety of displays to choose from. Since we're not so limited on memory on the Raspberry Pi, it works fine to just import everything.

```python
import os
import time
import digitalio
import board
from PIL import Image, ImageOps
import adafruit_rgb_display.ili9341 as ili9341
import adafruit_rgb_display.st7789 as st7789  # pylint: disable=unused-import
import adafruit_rgb_display.hx8357 as hx8357  # pylint: disable=unused-import
import adafruit_rgb_display.st7735 as st7735  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1351 as ssd1351  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1331 as ssd1331  # pylint: disable=unused-import
```

Next we have the buttons. The defaults are the most common setup, but a few of the displays vary and there is a number right next to the button. If you would like to change it, this is the correct place.

```python
# Change to match your display
BUTTON_NEXT = board.D17
BUTTON_PREVIOUS = board.D22
```

If you had the 1.14" Mini PiTFT, you would change it to something like the following:

```python
BUTTON_NEXT = board.D23
BUTTON_PREVIOUS = board.D24
```

Next, we need to set up the pins that the display will use. We are using the defaults for the PiTFT, though you could do the same thing with a breakout board. If you have a Mini PiTFT, change reset_pin to None.

```python
# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)

# Set this to None on the Mini PiTFT
reset_pin = digitalio.DigitalInOut(board.D24)
```

Next, we have a convenience function for initializing the button with the internal pull-up resistor enabled.

```python
def init_button(pin):
    button = digitalio.DigitalInOut(pin)
    button.switch_to_input()
    button.pull = digitalio.Pull.UP
    return button
```
Next we define a `Frame` class. This probably could have been done with a named tuple, since there are no methods, but classes are very readable and it gives the flexibility to add methods. The `Frame` class is used for holding the image and the duration of the frame. This allows for displaying animated Gifs with variable durations.

```python
class Frame:
    def __init__(self, duration=0):
        self.duration = duration
        self.image = None
```

Now to get into the `AnimatedGif` class, which does most of the work. The reason for going with a class is that there are a lot of shared variables that are used in functions and this modularizes everything. A good indicator of whether to use a class is if you need to use the `global` keyword to modify variables.

In the `__init__` function, it is mostly setting each of the internal variables to their defaults and initializing the buttons. If the `width` and `height` are passed in, those are used, otherwise they are extracted from the display object. We also allow the `folder` name to be passed in and if that is present, we will load all the files and automatically run it.

```python
class AnimatedGif:
    def __init__(self, display, width=None, height=None, folder=None):
        self._frame_count = 0
        self._loop = 0
        self._index = 0
        self._duration = 0
        self._gif_files = []
        self._frames = []

        if width is not None:
            self._width = width
        else:
            self._width = display.width

        if height is not None:
            self._height = height
        else:
            self._height = display.height

        self.display = display
        self.advance_button = init_button(BUTTON_NEXT)
        self.back_button = init_button(BUTTON_PREVIOUS)

        if folder is not None:
            self.load_files(folder)
            self.run()
```

Next, we have a pair of functions to either `advance` to the next image or go `back`. They use the modulus operator (`%`) to see if they go beyond the end of the range allowing them to loop back.

```python
def advance(self):
    self._index = (self._index + 1) % len(self._gif_files)
```
Next up is the `load_files` function which will grab anything that ends in a .gif extension and then go through that list to check if it an animated Gif. Anything that is an animation is added to the `self._gif_files` list. If none are found, it prints a message and exits.

```python
def load_files(self, folder):
    self._gif_files = [f for f in os.listdir(folder) if f.endswith('.gif')]
    for gif_file in gif_files:
        image = Image.open(gif_file)
        # Only add animated Gifs
        if image.is_animated:
            self._gif_files.append(gif_file)

    print("Found", self._gif_files)
    if not self._gif_files:
        print("No Gif files found in current folder")
        exit()
```

The preload function is a bit more complex, so we will break that up a little bit more. It starts by opening the current image and printing out a message to the console.

```python
def preload(self):
    image = Image.open(self._gif_files[self._index])
    print("Loading {}...",format(self._gif_files[self._index]))
    #continued...
```

It continues loading any meta data from the animated Gif such as loop and duration info if available.

```python
self._delay = image.info['duration']
if "loop" in image.info:
    self._loop = image.info['loop']
else:
    self._loop = 1
self._frame_count = image.n_frames
self._frames.clear()
#continued...
```

Then it loops through all the frames of the image. Inside the loop, it starts off by creating a frame object using the `Frame` class and sets the duration for the frame if available, otherwise it uses the default duration for the entire image.
Next it uses the Pillow ImageOps module's `pad()` function to shrink the image to fit on the screen and draws a black border where there is any extra space. To resize the image, the Nearest Neighbor method is used. To fill the entire screen and crop off parts of the image, then the `fit()` function could be used instead.

Finally it adds the frame object to the frames list.

```python
for frame in range(self._frame_count):
    image.seek(frame)
    # Create blank image for drawing.
    # Make sure to create image with mode 'RGB' for full color.
    frame_object = Frame(duration=self._duration)
    if "duration" in image.info:
        frame_object.duration = image.info['duration']
    frame_object.image = ImageOps.pad(image.convert("RGB"),
                                       (self._width, self._height),
                                       method=Image.NEAREST,
                                       color=(0, 0, 0),
                                       centering=(0.5, 0.5))
    self._frames.append(frame_object)
```

Now the `play` function will load and play the current image. It starts off by calling the `preload()` function. If there aren't any Gif images to play it will let you know.

Next it will start the looping of the animation and depending on the loop setting, it will either start counting down or continue indefinitely. If it gets down to 1, the looping will end and the function will return `True` indicating the run function should automatically advance to the next image. In between each frame, it will check the value of the buttons, move the index accordingly, and return `False` indicating that we already took care of changing the index and to just play the new index. The delay is taken care of using `time.monotonic()`, which returns a float value of the elapsed time. We use it to get the start time and compare it to the current time. This allows us to take image transfer time into account.

```python
def play(self):
    self.preload()
    # Check if we have loaded any files first
    if not self._gif_files:
        print("There are no Gif Images to Play")
        return False
    while True:
        for frame_object in self._frames:
            start_time = time.monotonic()
            self.display.image(frame_object.image)
            if not self.advance_button.value:
                self.advance()
                return False
            if not self.back_button.value:
                self.back()
                return False
            while time.monotonic() < (start_time + frame_object.duration / 1000):
                pass
```
Finally we have the `run` function. Its job is to keep advancing through each of the images in the list and loop back to the start if we hit the end. It the buttons were used to change the image, then it should not automatically advance.

```python
def run(self):
    while True:
        auto_advance = self.play()
        if auto_advance:
            self.advance()
```

Now that we have the classes all defined, we just need a little more setup and it's time to run it. First we initialize the display. The default display is the ili9341, but if you have a different display, feel free to comment out that one and uncomment the one you do have.

```
# Config for display baudrate (default max is 64mhz):
BAUDRATE = 64000000

# Setup SPI bus using hardware SPI:
spi = board.SPI()

# Create the display:
#disp = st7789.ST7789(spi, rotation=90) # 2.0" ST7789
#disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=90) # 1.3", 1.54" ST7789
#disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53, y_offset=40, # 1.14" ST7789
#disp = hx8357.HX8357(spi, rotation=180, # 3.5" HX8357
#disp = st7735.ST7735R(spi, rotation=90, # 1.8" ST7735R
#disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3, # 1.44" ST7735R
#disp = st7735.ST7735R(spi, rotation=90, bgr=True, # 0.96" MiniTFT
#disp = ssd1351.SSD1351(spi, rotation=180, # 1.5" SSD1351
#disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
#disp = ssd1331.SSD1331(spi, rotation=180, # 0.96" SSD1331
#disp = ili9341.ILI9341(spi, rotation=90, # 2.2", 2.4", 2.8", 3.2" ILI9341
#cs=cs_pin, dc=dc_pin, rst=reset_pin, baudrate=BAUDRATE)
```

Next we have some code to swap the width and height parameters depending on the rotation. This is necessary because `RGB_Display` prefers to draw images that take up the full screen.

```python
if disp.rotation % 180 == 90:
    disp_height = disp.width # we swap height/width to rotate it to landscape!
    disp_width = disp.height
else:
    disp_width = disp.width
    disp_height = disp.height
```
Finally we create the AnimatedGif object. By passing it the current folder, it will automatically run.

```python
gif_player = AnimatedGif(disp, width=disp_width, height=disp_height, folder=".")
```

**Full Code**

```python
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

Example to extract the frames and other parameters from an animated gif and then run the animation on the display.

Usage:
python3 rgb_display_pillow_animated_gif.py

This example is for use on (Linux) computers that are using CPython with Adafruit Blinka to support CircuitPython libraries. CircuitPython does not support PIL/pillow (python imaging library)!

Author(s): Melissa LeBlanc-Williams for Adafruit Industries
           Mike Mallett <mike@nerdcore.net>

import os
import time
import digitalio
import board
from PIL import Image, ImageOps
import numpy  # pylint: disable=unused-import
from adafruit_rgb_display import ili9341
from adafruit_rgb_display import st7789  # pylint: disable=unused-import
from adafruit_rgb_display import hx8357  # pylint: disable=unused-import
from adafruit_rgb_display import st7735  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1351  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1331  # pylint: disable=unused-import

# Change to match your display
BUTTON_NEXT = board.D17
BUTTON_PREVIOUS = board.D22

time.sleep(1)

cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)

# Set this to None on the Mini PiTFT
reset_pin = digitalio.DigitalInOut(board.D24)

def init_button(pin):
    button = digitalio.DigitalInOut(pin)
    button.switch_to_input()
    button.pull = digitalio.Pull.UP
    return button

# pylint: disable=too-few-public-methods

class Frame:
    def __init__(self, duration=0):
        self.duration = duration
        self.image = None
```

©Adafruit Industries
```python
# pylint: enable=too-few-public-methods

class AnimatedGif:
    def __init__(self, display, width=None, height=None, folder=None):
        self._frame_count = 0
        self._loop = 0
        self._index = 0
        self._duration = 0
        self._gif_files = []
        self._frames = []

        if width is not None:
            self._width = width
        else:
            self._width = display.width
        if height is not None:
            self._height = height
        else:
            self._height = display.height
        self.display = display
        self.advance_button = init_button(BUTTON_NEXT)
        self.back_button = init_button(BUTTON_PREVIOUS)
        if folder is not None:
            self.load_files(folder)
            self.run()

    def advance(self):
        self._index = (self._index + 1) % len(self._gif_files)

    def back(self):
        self._index = (self._index - 1 + len(self._gif_files)) %
        len(self._gif_files)

    def load_files(self, folder):
        gif_files = [f for f in os.listdir(folder) if f.endswith(".gif")]
        for gif_file in gif_files:
            gif_file = os.path.join(folder, gif_file)
            image = Image.open(gif_file)
            # Only add animated Gifs
            if image.is_animated:
                self._gif_files.append(gif_file)
        print("Found", self._gif_files)
        if not self._gif_files:
            print("No Gif files found in current folder")
            exit()  # pylint: disable=consider-using-sys-exit

    def preload(self):
        image = Image.open(self._gif_files[self._index])
        print("Loading {}...".format(self._gif_files[self._index]))
        if "duration" in image.info:
            self._duration = image.info["duration"]
        else:
            self._duration = 0
        if "loop" in image.info:
            self._loop = image.info["loop"]
        else:
            self._loop = 1
        self._frame_count = image.n_frames
        self._frames.clear()
        for frame in range(self._frame_count):
            image.seek(frame)
            # Create blank image for drawing.
            # Make sure to create image with mode 'RGB' for full color.
            frame_object = Frame(duration=self._duration)
            if "duration" in image.info:
                frame_object.duration = image.info["duration"]
            frame_object.image = ImageOps.pad(  # pylint: disable=no-member
```
image.convert("RGB"),
    (self._width, self._height),
    method=Image.NEAREST,
    color=(0, 0, 0),
    centering=(0.5, 0.5),
)
    self._frames.append(frame_object)

def play(self):
    self.preload()
    _prev_advance_btn_val = self.advance_button.value
    _prev_back_btn_val = self.back_button.value
    # Check if we have loaded any files first
    if not self._gif_files:
        print("There are no Gif Images loaded to Play")
        return False
    while True:
        for frame_object in self._frames:
            start_time = time.monotonic()
            self.display.image(frame_object.image)
            _cur_advance_btn_val = self.advance_button.value
            _cur_back_btn_val = self.back_button.value
            if not _cur_advance_btn_val and _prev_advance_btn_val:
                self.advance()
                return False
            if not _cur_back_btn_val and _prev_back_btn_val:
                self.back()
                return False
            _prev_back_btn_val = _cur_back_btn_val
            _prev_advance_btn_val = _cur_advance_btn_val
            while time.monotonic() < (start_time + frame_object.duration / 1000):
                pass
            if self._loop == 1:
                return True
            if self._loop > 0:
                self._loop -= 1

    def run(self):
        while True:
            auto_advance = self.play()
            if auto_advance:
                self.advance()

    # Config for display baudrate (default max is 64mhz):
    BAUDRATE = 64000000

    # Setup SPI bus using hardware SPI:
    spi = board.SPI()

    # pylint: disable=line-too-long
    # Create the display:
    # disp = st7789.ST7789(spi, rotation=90, # 2.0" ST7789
    # disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180, # 1.3", 1.54" ST7789
    # disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53, # 1.47" ST7789
    # disp = st7789.ST7789(spi, rotation=90, width=172, height=320, x_offset=34, # 1.4" ST7789
    # disp = st7789.ST7789(spi, rotation=270, width=170, height=320, x_offset=35, # 1.9" ST7789
    # disp = hx8357.HX8357(spi, rotation=180, # 3.5" HX8357
    # disp = st7735.ST7735R(spi, rotation=90, # 1.8" ST7735R
    # disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3, # 1.44" ST7735R
Using the Player

Adding Animated Gifs

You can find animated Gifs all over the internet. A great website for finding a large collection of them is [giphy.com](http://giphy.com). Once you find them, you'll need to get them over to your Raspberry Pi. The easiest way would be to just use FTP or downloading them directly from the internet on your Pi if that's what you would prefer. Just make sure they're in the same folder as the script.

Starting the Player

To use the player, open up the command prompt and go to the folder where the script resides. Make sure you have placed all of your animated Gifs in this folder. Once that is done, go ahead and run the command:

```
python3 rgb_display_pillow_animated_gif.py
```

You should see the animation running on the display:
Scrolling through Animated Gifs

Some animated Gif files have a limited number of loops, some will loop indefinitely, and some will just play once. If you have any that don't play indefinitely, then the player will automatically advance to the next animated Gif.

If you are viewing one that loops indefinitely, you can get to the next image by pushing the button for advancing or you can go back to the previous image by pushing the other button.

There's definite improvements that can be made to this player, and we wanted to publish what we had to get folks started - feel free to contribute and have fun!