Raspberry Pi Rotary Encoder Animated Gif Player

Created by Melissa LeBlanc-Williams

https://learn.adafruit.com/python-rotary-animated-gif-player-two-different-ways

Last updated on 2023-08-29 04:44:02 PM EDT
# Table of Contents

**Overview**  
- Parts  

**Mini PiTFT Player**  
- Wiring  
- Setup  
- Install Required Libraries  
- DejaVu TTF Font  
- Pillow Library  
- Download the Code  
- Run the Script  
- Full Player Code  

**PyGame Player**  
- How it works  
- Wiring  
- Rotary Trinkey Setup  
- Raspberry Pi Setup  
- Install Required Libraries  
- DejaVu TTF Font  
- Pillow Library  
- Download the Code  
- Run the Script  
- Full Player Code  

**Using the Player**  
- Adding Animated Gifs  
- Scrolling and Navigating through Animated Gifs  
- Exiting the Player  
- Some Modifications to Try
Overview

While there are plenty of animated gif players, none of them allow you to control the animated gifs like a slide show. This guide will show you how to build 2 different Animated Gif Players that allow you to control the speed and direction with a rotary encoder based on the hardware you have available. Build one yourself and join the Rotary Gif club!

In the first one, the Mini PiTFT Player, is an Animated Gif Player that is similar to the Gif Player in the PiTFT Python + Pillow Animated Gif Player guide and is in fact based on the animated Gif player from that guide. However, the code has been modified to be controlled by a rotary encode that can be connected easily by STEMMA QT.
In the second player, the PyGame Player, is a similar player that uses the Official Raspberry Pi 7" Display and a Rotary Trinkey. This version uses PyGame to create a full-sized window on a desktop version of Raspberry Pi.

Parts

You'll need the following parts for both. Any of the Raspberry Pi 4 models should work fine.

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

Rotary Encoder + Extras
This rotary encoder is the best of the best, it's a high-quality 24-pulse encoder, with detents and a nice feel. It is panel mountable for placement in a box, or you can plug it...
https://www.adafruit.com/product/377
Mini PiTFT Player Parts

**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

**Adafruit I2C Stemma QT Rotary Encoder Breakout with NeoPixel**
Rotary encoders are soooo much fun! Twist em this way, then twist them that way. Unlike potentiometers, they go all the way around and often have little detents for tactile feedback....
https://www.adafruit.com/product/4991

**STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long**
This 4-wire cable is a little over 100mm / 4” long and fitted with JST-SH female 4-pin connectors on both ends. Compared with the chunkier JST-PH these are 1mm pitch instead of...
https://www.adafruit.com/product/4210
PyGame Player Parts

Adafruit Rotary Trinkey - USB NeoPixel Rotary Encoder
It's half USB Key, half Adafruit Trinket, half rotary encoder...it's Rotary Trinkey, the circuit board with a Trinket M0 heart, a NeoPixel glow, and a...
https://www.adafruit.com/product/4964

These parts are more optional since it could be run on an external monitor, but it tidies up everything.

Pi Foundation Display - 7" Touchscreen Display for Raspberry Pi
The 7" Touchscreen Display for Raspberry Pi gives users the ability to create all-in-one, integrated projects such as tablets, infotainment systems and embedded...
https://www.adafruit.com/product/2718

Smart Pi Touch Pro for Raspberry Pi and Official 7" Display
Here is an updated, excellently designed stand that provides a sleek and sturdy look for the 7" Touchscreen...
https://www.adafruit.com/product/4951

Because the Smart Pi display orients the Trinkey with the knob in the rear, you may want to also pick up a short USB extension cable.

1 x USB Extension Cable
6-inch USB 3.0 Extension Cable
https://amzn.to/3kApSD2
Mini PiTFT Player

Wiring

It's easy to connect everything together with the Mini PiTFT player. First connect the display to the Raspberry Pi:

Then just connect a STEMMA QT Rotary Encoded up the the STEMMA QT Connector on the Mini PiTFT.

Setup

You will want to start by installing the latest version of the Raspberry Pi Lite onto an SD card. You can refer to the CircuitPython Libraries on Linux and Raspberry Pi guide for more help setting it up.

Once you have everything set up, you will need to open a terminal and install Blinka. Refer to the Installing CircuitPython Libraries on Raspberry Pi page to quickly get up and running.

Install Required Libraries

You will need to have a few libraries installed before the script will run on your computer.

Install the required CircuitPython libraries:
pip3 install adafruit-circuitpython-rgb-display
pip3 install adafruit-circuitpython-seesaw
pip3 install --upgrade --force-reinstall spidev

Also, install NumPy since it will speed up the RGB Display library:

pip3 install numpy

**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 ttf-dejavu
```

**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
```

**Download the Code**

Copy mini_pitft_player.py and animatedgif.py onto the Raspberry Pi. You can download them by clicking on the Download Project Bundle below or you can use `wget` to copy them right off the web into your current folder:

```
```

**Run the Script**

Just run it using the following command:

```
python3 mini_pitft_player.py
```
Full Player Code

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

""
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
"

import digitalio
import board
from animatedgif import AnimatedGif
import numpy  # pylint: disable=unused-import
from adafruit_seesaw import seesaw, rotaryio, digitalio as ss_digitalio
from adafruit_rgb_display import st7789

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
# microcontroller
seesaw = seesaw.Seesaw(i2c, addr=0x36)

seesaw_product = (seesaw.get_version() >> 16) & 0xFFFF
print("Found product {}").format(seesaw_product)
if seesaw_product != 4991:
    print("Wrong firmware loaded? Make sure you have a rotary encoder connected.")

seesaw.pin_mode(24, seesaw.INPUT_PULLUP)
button = ss_digitalio.DigitalIO(seesaw, 24)
encoder = rotaryio.IncrementalEncoder(seesaw)

# Change to match your display
BUTTON_UP = board.D23
BUTTON_DOWN = board.D24

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
```
def init_button(pin):
    digital_button = digitalio.DigitalInOut(pin)
    digital_button.switch_to_input(pull=digitalio.Pull.UP)
    return digital_button

class TFTAnimatedGif(AnimatedGif):
    def __init__(self, display, include_delays=True, folder=None):
        self._width = display.width
        self._height = display.height
        self.up_button = init_button(BUTTON_UP)
        self.down_button = init_button(BUTTON_DOWN)
        self._last_position = None
        self._button_state = False
        super().__init__(display, include_delays=include_delays, folder=folder)

    def get_next_value(self):
        while self._running:
            position = -encoder.position
            if position != self._last_position:
                self._last_position = position
                return str(position)
            elif not button.value and not self._button_state:
                self._button_state = True
                return str("click")
            elif button.value and self._button_state:
                self._button_state = False
                if not self.up_button.value or not self.down_button.value:
                    self._running = False
                return None

    def update_display(self, image):
        self.display.image(image)

# 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,
    height=240,
    y_offset=80,
    rotation=180,
    cs=cs_pin,
    dc=dc_pin,
    rst=None,
    baudrate=BAUDRATE,
)

gif_player = TFTAnimatedGif(disp, include_delays=False, folder="images")

# SPDX-License-Identifier: MIT

import os
import time
from PIL import Image, ImageOps

# pylint: disable=too-few-public-methods
class Frame:
    def __init__(self, duration=0):
        # Adafruit Industries
        ©Adafruit Industries
self.duration = duration
self.image = None

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

class AnimatedGif:
    def __init__(self, display, include_delays=True, folder=None):
        self._frame_count = 0
        self._loop = 0
        self._index = 0
        self._duration = 0
        self._gif_files = []
        self._frames = []
        self._running = True
        self.display = display
        self.include_delays = include_delays
        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")]
        gif_folder = folder
        if gif_folder[:-1] != "/":
            gif_folder += "/"
        for gif_file in gif_files:
            image = Image.open(gif_folder + gif_file)
            # Only add animated Gifs
            if image.is_animated:
                self._gif_files.append(gif_folder + 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,
def play(self):
    self.preload()
    current_frame = 0
    last_action = None
    # Check if we have loaded any files first
    if not self._gif_files:
        print("There are no Gif Images loaded to Play")
        return False
    self.update_display(self._frames[current_frame].image)
    while self._running:
        action = self.get_next_value()
        if action:
            if not last_action:
                last_action = action
            if action == "click":
                self.advance()
                return False
            elif int(action) < int(last_action):
                current_frame -= 1
            else:
                current_frame += 1
                current_frame %= self._frame_count
            frame_object = self._frames[current_frame]
            start_time = time.monotonic()
            self.update_display(frame_object.image)
            if self.include_delays:
                remaining_delay = frame_object.duration / 1000 - (time.monotonic() - start_time)
                if remaining_delay > 0:
                    time.sleep(remaining_delay)
            last_action = action
        if self._loop == 1:
            return True
        if self._loop > 0:
            self._loop -= 1

def run(self):
    while self._running:
        auto_advance = self.play()
        if auto_advance:
            self.advance()
PyGame Player

This one is a bit more complicated because it uses 2 devices running Python that need to communicate with each other through serial.

How it works

This player is in two parts. It uses a Rotary Trinkey running CircuitPython to receive the input and translate it into serial data and the second part runs on the Raspberry Pi and reads that serial data and decodes that to the original inputs and uses that to determine the when to advance the animated Gif frames. This is the same technique used in the [MacroPad Remote Procedure Calls over USB to Control Home Assistant](https://www.adafruit.com/learn/article/502) guide.

For the Rotary Trinkey, the code is very similar to the standard example code, but with a few key differences. It will communicate using a second CDC_Data serial port so that unnecessary characters are not added to the stream by CircuitPython. It will also print out the word "click" when the button is pressed and everything is separated by commas instead of printing onto a new line.

For the Python program, it starts off by detecting the serial port that the Rotary Trinkey is connected to and initializing it with PySerial. Once it is connected, PyGame is initialized in fullscreen mode and the mouse cursor is hidden. It then preloads any Gif images found in the folder and stores them in a list. It starts off with the first image in whatever order the `os` module returns the files in and loads up the first one.
Once an image is loaded, it waits for data and depending on the direction you turn the know it will either go forward or backwards in frames. A modulus operator is used to keep the current frame value within the index boundaries. Once the frame is extracted with Pillow, it is loaded into PyGame as a surface using the `blit()` command and the screen is updated with the `flip()` command.

**Wiring**

Wiring is super easy on this player because all you need to do is connect the Rotary Trinkey to any of the USB ports. We recommend using a USB extension if you have the Display mounted on a stand because the orientation of the Raspberry Pi causes the Trinkey to only be accessible from the rear.

If you bought the SmartPi Pro case, you can find assembly instructions on their website.(

---

**Rotary Trinkey Setup**

You'll want to put the latest version of CircuitPython onto the Rotary Trinkey first. If you're not sure how to do it, you can refer to CircuitPython installation page of the Adafruit Rotary Trinkey guide.(

For this project, there are no libraries needed. Everything is built into CircuitPython itself.

You will need to copy both the boot.py and code.py files onto the Trinkey, in the root of the CIRCUITPY drive that should show up. You can download them by clicking on the Download Project Bundle below.
import usb_cdc
import rotaryio
import board
import digitalio

serial = usb_cdc.data
encoder = rotaryio.IncrementalEncoder(board.ROTA, board.ROTB)
button = digitalio.DigitalInOut(board.SWITCH)
button.switch_to_input(pull=digitalio.Pull.UP)

last_position = None
button_state = False

while True:
    position = encoder.position
    if last_position is None or position != last_position:
        serial.write(bytes(str(position) + ',', 'utf-8'))
    last_position = position
    print(button.value)
    if not button.value and not button_state:
        button_state = True
    if button.value and button_state:
        serial.write(bytes('click,', 'utf-8'))
        button_state = False

The boot.py file has the setting sin it you need to enable the CDC Data Device. You can read more about it in our Customizing USB Devices in CircuitPython guide.

# SPDX-FileCopyrightText: 2021 Melissa LeBlanc-Williams for Adafruit Industries
# SPDX-License-Identifier: MIT
import usb_cdc
usb_cdc.enable(console=True, data=True)

Raspberry Pi Setup

You will want to start by installing the latest version of the Raspberry Pi Desktop onto an SD card. There's a nice utility available from Raspberry Pi called the Raspberry Pi Imager that makes installing images very easy. You can also refer to the CircuitPython Libraries on Linux and Raspberry Pi guide for more help setting it up.

Once you have everything set up, you will need to open a terminal and install Blinka. We recommend referring to the Installing CircuitPython Libraries on Raspberry Pi page of that guide to quickly get up and running.
Install Required Libraries

You will need to have a few libraries installed before the script will run on your computer.

Install Adafruit_Board_Toolkit:

```
pip3 install adafruit-board-toolkit
```

Install PySerial next:

```
pip3 install pyserial
```

Install PyGame next:

```
pip3 install pygame
```

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 calls 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
```

Download the Code

Copy pygame_player.py and animatedgif.py onto the Raspberry Pi. You can either copy them out of the bundle that you downloaded in the Rotary Trinkey Setup step or you can use `wget` to copy them right off the web into your current folder:
Run the Script

The code must be run from a terminal window on the Pi itself and not from an SSH session. This is because PyGame will try and create a window on the current device you are typing from. Just run it using the following command:

```
python3 pygame_player.py
```

Full Player Code

```python
# SPDX-FileCopyrightText: 2021 Melissa LeBlanc-Williams for Adafruit Industries
#
# SPDX-License-Identifier: MIT
#
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

```
ports = [comport.device for comport in comports]
if len(ports) >= 1:
    if len(ports) > 1:
        print("Multiple devices detected, using the first detected port.")
        return ports[0]
    raise RuntimeError(
        "Unable to find any CircuitPython Devices with the CDC data port enabled."
    )

port = serial.Serial(
    detect_port(),
    9600,
    parity="N",
    rtscts=False,
    xonxoff=False,
    exclusive=True,
)

class PyGameAnimatedGif(AnimatedGif):
    def __init__(self, display, include_delays=True, folder=None):
        self._width, self._height = pygame.display.get_surface().get_size()
        self._incoming_packet = b"
        super().__init__(display, include_delays=include_delays, folder=folder)

    def get_next_value(self):
        if not port:
            return None
        while port.in_waiting:
            self._incoming_packet += port.read(port.in_waiting)
        while (self._running and not len(self._incoming_packet)
                and self._incoming_packet.decode().find(",",)):
            self._incoming_packet += port.read(port.in_waiting)
            self.check_pygame_events()

        all_values = self._incoming_packet.decode().split(",",)
        value = all_values.pop(0)
        self._incoming_packet = ",".join(all_values).encode("utf-8")
        return value

    def check_pygame_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self._running = False
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    self._running = False

    def update_display(self, image):
        pilImage = image
        self.display.blit(
            pygame.image.fromstring(pilImage.tobytes(), pilImage.size,
                                    pilImage.mode),
            (0, 0),
        )
        pygame.display.flip()

pygame.init()
pygame.mouse.set_visible(False)
screen = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
gif_player = PyGameAnimatedGif(screen, include_delays=False, folder="images")
pygame.mouse.set_visible(True)
port.close()
import os
import time
from PIL import Image, ImageOps

# pylint: disable=too-few-public-methods
class Frame:
    def __init__(self, duration=0):
        self.duration = duration
        self.image = None

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

class AnimatedGif:
    def __init__(self, display, include_delays=True, folder=None):
        self._frame_count = 0
        self._loop = 0
        self._index = 0
        self._duration = 0
        self._gif_files = []
        self._frames = []
        self._running = True
        self.display = display
        self.include_delays = include_delays
        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")]
        gif_folder = folder
        if gif_folder[:-1] != "/":
            gif_folder += "/"
        for gif_file in gif_files:
            image = Image.open(gif_folder + gif_file)
            # Only add animated Gifs
            if image.is_animated:
                self._gif_files.append(gif_folder + 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()
    current_frame = 0
    last_action = None
    # Check if we have loaded any files first
    if not self._gif_files:
        print("There are no Gif Images loaded to Play")
        return False
    self.update_display(self._frames[current_frame].image)
    while self._running:
        action = self.get_next_value()
        if action:
            if not last_action:
                last_action = action
            if action == "click":
                self.advance()
                return False
            elif int(action) < int(last_action):
                current_frame -= 1
            else:
                current_frame += 1
                current_frame %= self._frame_count
            frame_object = self._frames[current_frame]
            start_time = time.monotonic()
            self.update_display(frame_object.image)
            if self.include_delays:
                remaining_delay = frame_object.duration / 1000 - (time.monotonic() - start_time)
                if remaining_delay > 0:
                    time.sleep(remaining_delay)
            last_action = action
            if self._loop == 1:
                return True
            if self._loop > 0:
                self._loop -= 1

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

The default folder used in the test scripts is an images subfolder inside of the folder the script is in. If you are using something different, be sure to change it at the bottom of the script.

Scrolling and Navigating through Animated Gifs

Just rotate the knob clockwise to start scrolling through the frames of the current Gif. If you turn the knob fast, it will move faster and turning it slowly will make it advance frames at a slower rate.

For the Mini PiTFT Player it seems turning the knob too fast will make the animation slow down a bit because it takes more processing power to pipe the data out to the TFT display and it can't read the rotary encoder quite as often.

To load the next animated Gif, just click the knob inward. When it gets to the end, it will load the first Gif.
Exiting the Player

On the PyGame Player, just hit ESC on the keyboard to exit. For the Mini PiTFT Player, you can just hit Control+C to exit.

Some Modifications to Try

There is some room for modification in the code. For instance in either player, you can change `include_delays= False` to `include_delays=True` and then it will include the built-in delays that are part of the animated Gif file.

Another modification that could be made for the PyGame Player takes advantage of the touch sensor on the Rotary Trinkey and uses that to send a quit command to close the application.

For the Mini PiTFT Player you can have it jump a certain number of frames based on the rotary increase on the knob. This may have the effect of the animated gif skipping around though. Have fun with it and see what you can get it to do.