NeoPixel Run LED Arcade Game
Created by Ruiz Brothers

https://learn.adafruit.com/pixel-chase-game

Last updated on 2021-12-08 04:58:07 PM EST
Table of Contents

Overview 5
• Arcade Inspired Game 5
• Game Goals & Rules 5
• Parts 6

Circuit Diagram 7
• Adafruit Library for Fritzing 7
• Wired Connections 8
• Powering 8

3D Printing 9
• Parts List 9
• Slicing Parts 9
• Design Source Files 10

CircuitPython on Feather M4 Express 10
• Set up CircuitPython Quick Start! 10
• Further Information 12

CircuitPython Libraries 13
• The Adafruit CircuitPython Library Bundle 14
• Downloading the Adafruit CircuitPython Library Bundle 14
• The CircuitPython Community Library Bundle 15
• Downloading the CircuitPython Community Library Bundle 15
• Understanding the Bundle 16
• Example Files 16
• Copying Libraries to Your Board 17
• Understanding Which Libraries to Install 17
• Example: ImportError Due to Missing Library 20
• Library Install on Non-Express Boards 21
• Updating CircuitPython Libraries and Examples 21

Coding the Pixel Chase Game 22

Pixel Chase Game CircuitPython Code Walkthrough 25
• Import the Libraries 25
• Setup the Button and NeoPixels 26
• NeoPixel Animations 26
• Variables and States 27
• Colors 28
• The Loop 28
• Randomized Target 28
• Playing the Game 29
• Chaser Pixel Animation 29
• Level-Up 30
• Missing the Target 30
• You Win! 31
• Time Tracking 32

Assemble 32
• Solder LED Strip 32
• Slide switch 32
• Solder quick connects 33
• Test Circuit 33
• Mount Feather 33
• Mount LED strip 34
• Mount slide switch 34
• Mount diffuser 35
• Plug in Battery 35
• Attach Shelf 35
• Mount Button 35
• Connect button 36
• Align Lid 36
Overview

Arcade Inspired Game
Building an arcade game with NeoPixels and CircuitPython! This project is inspired by the “cyclone” LED chase games often found in arcades. The enclosure is 3D printed and snap fits together. Inside is an Adafruit Feather, an arcade button, a rechargeable battery and a slide switch.

Game Goals & Rules
The goal is to press the button when the LED lands on the target pixels. The game advances and speeds up as you score and hit the targets. The running pixel changes color as you level up and goes in the order of ROYGBIV.

The target pixels are placed randomly so it’s different for each level. If you miss a target, the LEDs flash red and the game starts over with the slow speed.
Parts

Adafruit Feather M4 Express - Featuring ATSAMD51
It's what you've been waiting for, the Feather M4 Express featuring ATSAMD51. This Feather is fast like a swift, smart like an owl, strong like a ox-bird (it's half ox,...
https://www.adafruit.com/product/3857

Adafruit Mini Skinny NeoPixel Digital RGB LED Strip - 144 LED/m
So thin. So mini. So teeeeeny-tiny. It's the 'skinny' version of our classic NeoPixel strips! These NeoPixel strips have 144 digitally-addressable pixel Mini LEDs...
https://www.adafruit.com/product/2969

Mini LED Arcade Button - 24mm Green
A button is a button, and a switch is a switch, but these translucent arcade buttons are in a class of their own. Particularly because they have LEDs built right in!
https://www.adafruit.com/product/3433
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 10-wire ribbon cable</td>
<td>silicone cover stranded-core</td>
<td><a href="https://www.adafruit.com/product/3890">https://www.adafruit.com/product/3890</a></td>
</tr>
<tr>
<td>1 x slide switch</td>
<td>breadboard friendly</td>
<td><a href="https://www.adafruit.com/product/805">https://www.adafruit.com/product/805</a></td>
</tr>
<tr>
<td>1 x JST Extension cable</td>
<td>2-pin JST cable</td>
<td><a href="https://www.adafruit.com/product/1131">https://www.adafruit.com/product/1131</a></td>
</tr>
<tr>
<td>1 x Quick-Connects</td>
<td>Arcade Button Quick-Connect Wire Pairs - 0.11&quot; (10 pack)</td>
<td><a href="https://www.adafruit.com/product/1152">https://www.adafruit.com/product/1152</a></td>
</tr>
<tr>
<td>1 x USB cable</td>
<td>USB cable - USB A to Micro-B</td>
<td><a href="https://www.adafruit.com/product/592">https://www.adafruit.com/product/592</a></td>
</tr>
</tbody>
</table>

### Lithium Ion Polymer Battery with Short Cable - 3.7V 420mAh

Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This... [https://www.adafruit.com/product/4236](https://www.adafruit.com/product/4236)

---

**Circuit Diagram**

The diagram below provides a visual reference for wiring of the components. This diagram was created using the software package [Fritzing](https://adafru.it/oEP).

---

**Adafruit Library for Fritzing**

Use Adafruit's Fritzing parts library to create circuit diagrams for your projects. Download the library or just grab individual parts. Get the library and parts from [GitHub - Adafruit Fritzing Parts](https://adafru.it/AYZ).
Wired Connections

- 5V from LED Strip to 3V on Feather
- GND from LED Strip to GND on Feather
- DIN from LED Strip to Pin #6 on Feather
- Button to Pin #5 on Feather
- Button to GND on Feather
- Switch to GND on Feather
- Switch to EN on Feather

Powering

The Adafruit board can be powered via USB or JST using a 3.7v lipo battery. In this project, a 420mAh lipo battery is used. The lipo battery is rechargeable via the USB port on the board. The switch is wired to the enable and ground pins on the board.
3D Printing

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

- shell-led-chase.stl
- lid-led-chase.stl
- diffuser-led-chase.stl
- button-sleeve.stl

Edit Design
https://adafruit.it/MrA

Download CAD files from PrusaPrinters
https://adafruit.it/Mvb

Download CAD files from Thingiverse
https://adafruit.it/Mvc

Slicing Parts
No supports are required. Slice with settings for PLA material.

The parts were sliced using CURA using the slice settings below.

- PLA filament 220c extruder
- 0.2 layer height
- 10% gyroid infill
- 90mm/s print speed
- 60c heated bed
Design Source Files

The project assembly was designed in Fusion 360. This can be downloaded in different formats like STEP, STL and more. Electronic components like Adafruit's board, displays, connectors and more can be downloaded from the Adafruit CAD parts GitHub Repo (https://adafru.it/AW8).

CircuitPython on Feather M4 Express

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the CIRCUITPY drive to iterate.

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

Set up CircuitPython Quick Start!

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

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

https://adafru.it/Emh
Click the link above and download the latest UF2 file.

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

Plug your Feather 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 next to the USB connector on your board, and you will see the NeoPixel RGB LED turn green. If it turns red, check the USB cable, try another USB port, etc. Note: The little red LED next to the USB connector will pulse red. That's ok!

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 FEATHERBOOT.

Drag the adafruit_circuitpython_etc.uf2 file to FEATHERBOOT.

The LED will flash. Then, the FEATHERBOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

That's it, you're done! :)

Further Information

For more detailed info on installing CircuitPython, check out [Installing CircuitPython](https://adafruit.it/Amd).
CircuitPython Libraries

As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. Visit https://circuitpython.org/downloads to download the latest version of CircuitPython for your board. You must download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then visit https://circuitpython.org/libraries to download the latest Library Bundle.

Each CircuitPython program you run needs to have a lot of information to work. The reason CircuitPython is so simple to use is that most of that information is stored in other files and works in the background. These files are called libraries. Some of them are built into CircuitPython. Others are stored on your CIRCUITPY drive in a folder called lib. Part of what makes CircuitPython so great is its ability to store code separately from the firmware itself. Storing code separately from the firmware makes it easier to update both the code you write and the libraries you depend.

Your board may ship with a lib folder already, it's in the base directory of the drive. If not, simply create the folder yourself. When you first install CircuitPython, an empty lib directory will be created for you.

CircuitPython libraries work in the same way as regular Python modules so the Python docs (https://adafruit.it/rar) are an excellent reference for how it all should work. In Python terms, you can place our library files in the lib directory because it's part of the Python path by default.

One downside of this approach of separate libraries is that they are not built in. To use them, one needs to copy them to the CIRCUITPY drive before they can be used. Fortunately, there is a library bundle.
The bundle and the library releases on GitHub also feature optimized versions of the libraries with the .mpy file extension. These files take less space on the drive and have a smaller memory footprint as they are loaded.

Due to the regular updates and space constraints, Adafruit does not ship boards with the entire bundle. Therefore, you will need to load the libraries you need when you begin working with your board. You can find example code in the guides for your board that depends on external libraries.

Either way, as you start to explore CircuitPython, you'll want to know how to get libraries on board.

**The Adafruit CircuitPython Library Bundle**

Adafruit provides CircuitPython libraries for much of the hardware they provide, including sensors, breakouts and more. To eliminate the need for searching for each library individually, the libraries are available together in the Adafruit CircuitPython Library Bundle. The bundle contains all the files needed to use each library.

**Downloading the Adafruit CircuitPython Library Bundle**

You can download the latest Adafruit CircuitPython Library Bundle release by clicking the button below. The libraries are being constantly updated and improved, so you'll always want to download the latest bundle.

Match up the bundle version with the version of CircuitPython you are running. For example, you would download the 6.x library bundle if you're running any version of CircuitPython 6, or the 7.x library bundle if you're running any version of CircuitPython 7, etc. If you mix libraries with major CircuitPython versions, you will get incompatible mpy errors due to changes in library interfaces possible during major version changes.

[Click to visit circuitpython.org for the latest Adafruit CircuitPython Library Bundle](https://adafru.it/ENC)
Download the bundle version that matches your CircuitPython firmware version. If you don't know the version, check the version info in boot_out.txt file on the CIRCUITPY drive, or the initial prompt in the CircuitPython REPL. For example, if you're running v7.0.0, download the 7.x library bundle.

There's also a py bundle which contains the uncompressed python files, you probably don't want that unless you are doing advanced work on libraries.

The CircuitPython Community Library Bundle

The CircuitPython Community Library Bundle is made up of libraries written and provided by members of the CircuitPython community. These libraries are often written when community members encountered hardware not supported in the Adafruit Bundle, or to support a personal project. The authors all chose to submit these libraries to the Community Bundle make them available to the community.

These libraries are maintained by their authors and are not supported by Adafruit. As you would with any library, if you run into problems, feel free to file an issue on the GitHub repo for the library. Bear in mind, though, that most of these libraries are supported by a single person and you should be patient about receiving a response. Remember, these folks are not paid by Adafruit, and are volunteering their personal time when possible to provide support.

Downloading the CircuitPython Community Library Bundle

You can download the latest CircuitPython Community Library Bundle release by clicking the button below. The libraries are being constantly updated and improved, so you'll always want to download the latest bundle.

Click for the latest CircuitPython Community Library Bundle release

https://adafruit.it/VCn

The link takes you to the latest release of the CircuitPython Community Library Bundle on GitHub. There are multiple versions of the bundle available. Download the bundle version that matches your CircuitPython firmware version. If you don't know the version, check the version info in boot_out.txt file on the CIRCUITPY drive, or the initial prompt in the CircuitPython REPL. For example, if you're running v7.0.0, download the 7.x library bundle.
Understanding the Bundle

After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.

Open the bundle folder. Inside you'll find two information files, and two folders. One folder is the lib bundle, and the other folder is the examples bundle.

Now open the lib folder. When you open the folder, you'll see a large number of .mpy files, and folders.

Example Files

All example files from each library are now included in the bundles in an examples directory (as seen above), as well as an examples-only bundle. These are included for two main reasons:

- Allow for quick testing of devices.
- Provide an example base of code, that is easily built upon for individualized purposes.
Copying Libraries to Your Board

First open the lib folder on your CIRCUITPY drive. Then, open the lib folder you extracted from the downloaded zip. Inside you’ll find a number of folders and .mpy files. Find the library you’d like to use, and copy it to the lib folder on CIRCUITPY.

If the library is a directory with multiple .mpy files in it, be sure to copy the entire folder to CIRCUITPY/lib.

This also applies to example files. Open the examples folder you extracted from the downloaded zip, and copy the applicable file to your CIRCUITPY drive. Then, rename it to code.py to run it.

If a library has multiple .mpy files contained in a folder, be sure to copy the entire folder to CIRCUITPY/lib.

Understanding Which Libraries to Install

You now know how to load libraries on to your CircuitPython-compatible microcontroller board. You may now be wondering, how do you know which libraries you need to install? Unfortunately, it’s not always straightforward. Fortunately, there is an obvious place to start, and a relatively simple way to figure out the rest. First up: the best place to start.

When you look at most CircuitPython examples, you’ll see they begin with one or more import statements. These typically look like the following:

- import library_or_module

However, import statements can also sometimes look like the following:

- from library_or_module import name
They can also have more complicated formats, such as including a `try / except` block, etc.

The important thing to know is that an `import` statement will always include the name of the module or library that you're importing.

Therefore, the best place to start is by reading through the `import` statements.

Here is an example import list for you to work with in this section. There is no setup or other code shown here, as the purpose of this section involves only the import list.

```python
import time
import board
import neopixel
import adafruit_lis3dh
import usb_hid
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode
```

Keep in mind, not all imported items are libraries. Some of them are almost always built-in CircuitPython modules. How do you know the difference? Time to visit the REPL.

In the [Interacting with the REPL section](https://adafruit.it/Awz) on [The REPL page](http://adafruit.it/Awz) in this guide, the `help("modules")` command is discussed. This command provides a list of all of the built-in modules available in CircuitPython for your board. So, if you connect to the serial console on your board, and enter the REPL, you can run `help("modules")` to see what modules are available for your board. Then, as you read through the `import` statements, you can, for the purposes of figuring out which libraries to load, ignore the statement that import modules.

The following is the list of modules built into CircuitPython for the Feather RP2040. Your list may look similar or be anything down to a significant subset of this list for smaller boards.
Now that you know what you're looking for, it's time to read through the import statements. The first two, `time` and `board`, are on the modules list above, so they're built-in.

The next one, `neopixel`, is not on the module list. That means it's your first library! So, you would head over to the bundle zip you downloaded, and search for neopixel. There is a neopixel.mpy file in the bundle zip. Copy it over to the lib folder on your CI RCUITPY drive. The following one, `adafruit_lis3dh`, is also not on the module list. Follow the same process for adafruit_lis3dh, where you'll find adafruit_lis3dh.mpy, and copy that over.

The fifth one is `usb_hid`, and it is in the modules list, so it is built in. Often all of the built-in modules come first in the import list, but sometimes they don't! Don't assume that everything after the first library is also a library, and verify each import with the modules list to be sure. Otherwise, you'll search the bundle and come up empty!

The final two imports are not as clear. Remember, when `import` statements are formatted like this, the first thing after the `from` is the library name. In this case, the library name is `adafruit_hid`. A search of the bundle will find an adafruit_hid folder. When a library is a folder, you must copy the entire folder and its contents as it is in the bundle to the lib folder on your CIRCUITPY drive. In this case, you would copy the entire adafruit_hid folder to your CIRCUITPY/lib folder.

Notice that there are two imports that begin with `adafruit_hid`. Sometimes you will need to import more than one thing from the same library. Regardless of how many times you import the same library, you only need to load the library by copying over the adafruit_hid folder once.

That is how you can use your example code to figure out what libraries to load on your CircuitPython-compatible board!
There are cases, however, where libraries require other libraries internally. The internally required library is called a dependency. In the event of library dependencies, the easiest way to figure out what other libraries are required is to connect to the serial console and follow along with the `ImportError` printed there. The following is a very simple example of an `ImportError`, but the concept is the same for any missing library.

**Example: `ImportError` Due to Missing Library**

If you choose to load libraries as you need them, or you're starting fresh with an existing example, you may end up with code that tries to use a library you haven't yet loaded. This section will demonstrate what happens when you try to utilise a library that you don't have loaded on your board, and cover the steps required to resolve the issue.

This demonstration will only return an error if you do not have the required library loaded into the lib folder on your CIRCUITPY drive.

Let's use a modified version of the Blink example.

```python
import board
import time
import simpleio

led = simpleio.DigitalOut(board.LED)

while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

Save this file. Nothing happens to your board. Let's check the serial console to see what's going on.

You have an `ImportError`. It says there is no module named `simpleio`. That's the one you just included in your code!
Click the link above to download the correct bundle. Extract the lib folder from the downloaded bundle file. Scroll down to find simpleio.mpy. This is the library file you're looking for! Follow the steps above to load an individual library file.

The LED starts blinking again! Let's check the serial console.

![Serial Console](image)

No errors! Excellent. You've successfully resolved an **ImportError**!

If you run into this error in the future, follow along with the steps above and choose the library that matches the one you're missing.

### Library Install on Non-Express Boards

If you have an M0 non-Express board such as Trinket M0, Gemma M0, QT Py M0, or one of the M0 Trinkeys, you'll want to follow the same steps in the example above to install libraries as you need them. Remember, you don't need to wait for an **ImportError** if you know what library you added to your code. Open the library bundle you downloaded, find the library you need, and drag it to the lib folder on your CIRCUITPY drive.

You can still end up running out of space on your M0 non-Express board even if you only load libraries as you need them. There are a number of steps you can use to try to resolve this issue. You'll find suggestions on the [Troubleshooting page](https://adafruit.com/arduino/tutorials/tutorial).

### Updating CircuitPython Libraries and Examples

Libraries and examples are updated from time to time, and it's important to update the files you have on your CIRCUITPY drive.

To update a single library or example, follow the same steps above. When you drag the library file to your lib folder, it will ask if you want to replace it. Say yes. That's it!
A new library bundle is released every time there's an update to a library. Updates include things like bug fixes and new features. It's important to check in every so often to see if the libraries you're using have been updated.

Coding the Pixel Chase Game

Once you've finished setting up your Feather M4 Express with CircuitPython, you can add these libraries to the lib folder:

- adafruit_led_animation
- adafruit_pypixelbuf.mpy
- neopixel.mpy

Then, you can click on the Download: Project Zip link below to download the code.

```python
import time
import random
import board
from rainbowio import colorwheel
import neopixel
import digitalio
import adafruit_led_animation.color as color

# button pin setup
button = digitalio.DigitalInOut(board.D5)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP

# neopixel setup
pixel_pin = board.D6
num_pixels = 61

pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.2, auto_write=False)

def rainbow_cycle(wait):
    for j in range(255):
        for i in range(num_pixels):
            rc_index = (i * 256 // 10) + j
            pixels[i] = colorwheel(rc_index & 255)
pixels.show()
time.sleep(wait)

def color_chase(c, wait):
    for i in range(num_pixels):
        pixels[i] = c
time.sleep(wait)
pixels.show()
time.sleep(0.5)

# function to blink the neopixels when you lose
```
def game_over():
    color_chase(color.BLACK, 0.05)
    pixels.fill(color.RED)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.BLACK)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.RED)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.BLACK)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.RED)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.BLACK)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.RED)
    pixels.show()
    time.sleep(1)

# variables and states
pixel = 0
num = 0
last_num = 0
now_color = 0
next_color = 1
speed = 0.1
level = 0.005
final_level = 0.001
new_target = True
button_state = False

# neopixel colors
colors = [color.RED, color.ORANGE, color.YELLOW, color.GREEN, color.TEAL,
          color.CYAN, color.BLUE, color.PURPLE, color.MAGENTA, color.GOLD, color.AQUA,
          color.PINK]

while True:
    # button debouncing
    if not button.value and not button_state:
        button_state = True

    # if new level starting..
    if new_target:
        # randomize target location
        y = int(random.randint(5, 55))
        x = int(y - 1)
        z = int(y + 1)
        new_target = False
        print(x, y, z)
        pixels[x] = color.WHITE
        pixels[y] = colors[next_color]
        pixels[z] = color.WHITE
        # delay without time.sleep()
        if (pixel + speed) < time.monotonic():
            # turn off pixel behind chaser
            if num > 0:
                last_num = num - 1
                pixels[last_num] = color.BLACK
                pixels.show()
            # keep target pixels their colors when the chaser passes
            if last_num in (x, y, z):
                pixels[x] = color.WHITE
                pixels[y] = colors[next_color]
                pixels[z] = color.WHITE
            # move chaser pixel by one
            if num < num_pixels:
                pixels[num] = colors[now_color]
                pixels.show()
# print(num)
# print("target is", y)
num += 1

# send chaser back to the beginning of the circle
if num == num_pixels:
    last_num = num - 1
    pixels[last_num] = color.BLACK
    pixels.show()
    num = 0

# if the chaser hits the target...
if last_num in [x, y, z] and not button.value:
    button_state = False
    # fills with the next color
    pixels.fill(colors[next_color])
    pixels.show()
    print(num)
    print(x, y, z)
    # chaser resets
    num = 0
    time.sleep(0.5)
    pixels.fill(color.BLACK)
    pixels.show()
    # speed increases for next level
    speed = speed + level
    # color updates
    next_color = next_color + 1
    if next_color > 11:
        next_color = 0
    now_color = now_color + 1
    if now_color > 11:
        now_color = 0
        # setup for new target
        new_target = True
        print("speed is", speed)
        print("button is", button.value)

    # if the chaser misses the target...
if last_num not in [x, y, z] and not button.value:
    button_state = False
    print(num)
    print(x, y, z)
    # fills with current chaser color
    pixels.fill(colors[now_color])
    pixels.show()
    # function to flash all pixels red
    game_over()
    # chaser is reset
    num = 0
    pixels.fill(color.BLACK)
    pixels.show()
    # speed is reset to default
    speed = 0.1
    # colors are reset
    next_color = 1
    now_color = 0
    # setup for new target
    new_target = True
    print("speed is", speed)
    print("button is", button.value)

    # when you have beaten all the levels...
if speed < final_level:
    # rainbows!
    rainbow_cycle(0.01)
    time.sleep(1)
    # chaser is reset
    num = 0
    pixels.fill(color.BLACK)
    pixels.show()
    # speed is reset to default
    speed = 0.1
# colors are reset
next_color = 1
now_color = 0
# setup for new target
new_target = True
# time.monotonic() is reset for the delay
pixel = time.monotonic()

Your Feather M4 Express CIRCUITPY drive should look like this after you load the libraries and code.py file:

Pixel Chase Game CircuitPython Code Walkthrough

Import the Libraries
First, the CircuitPython libraries are imported. The adafruitLedAnimation library is being used as a way to easily access different colors for the NeoPixels.

```python
import time
import random
import board
import neopixel
import digitalio
import adafruitLedAnimation.color as color
```
Setup the Button and NeoPixels

Next, the button's pin is setup.

```python
# button pin setup
button = digitalio.DigitalInOut(board.D5)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP
```

Followed by the NeoPixel setup.

```python
# neopixel setup
pixel_pin = board.D6
num_pixels = 61
pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.2, auto_write=False)
```

NeoPixel Animations

Three functions are brought in to continue the NeoPixel setup. All three of them are classic NeoPixel animations: `rainbow_cycle` and `color_chase`.

```python
# wheel and rainbow_cycle setup
def wheel(pos):
    if pos &lt; 0 or pos &gt;= 255:
        return (0, 0, 0)
    if pos &lt; 85:
        return (255 - pos * 3, pos * 3, 0)
    if pos &lt; 170:
        pos -= 85
        return (0, 255 - pos * 3, pos * 3)
    pos -= 170
    return (pos * 3, 0, 255 - pos * 3)

def rainbow_cycle(wait):
    for j in range(255):
        for i in range(num_pixels):
            rc_index = (i * 256 // 10) + j
            pixels[i] = wheel(rc_index &amp; 255)
        pixels.show()
        time.sleep(wait)

def color_chase(c, wait):
    for i in range(num_pixels):
        pixels[i] = c
        time.sleep(wait)
    pixels.show()
    time.sleep(0.5)
```

©Adafruit Industries
These animation functions are followed by the `game_over()` function. This function allows for the NeoPixel strip to use `color_chase` to turn off the NeoPixels and then blink the strip red when you lose a level in the game. The reason for setting it up as a function rather than in the `loop` is to keep the loop easier to read since there will be other things going on.

```python
# function to blink the neopixels when you lose
def game_over():
    color_chase(color.BLACK, 0.05)
    pixels.fill(color.RED)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.BLACK)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.RED)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.BLACK)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.RED)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.BLACK)
    pixels.show()
    time.sleep(0.5)
    pixels.fill(color.RED)
    pixels.show()
    time.sleep(1)
```

Variables and States

Next are the variables and state machines that will be used in the loop. Their functions are commented next to them.

```python
# variables and states
pixel = 0  # time.monotonic() holder
num = 0    # chaser NeoPixel position
last_num = 0  # previous chaser position
now_color = 0  # chaser NeoPixel color
next_color = 1  # target NeoPixel color
speed = 0.1    # default speed for chaser
level = 0.005  # speed increase increment
final_level = 0.001  # final level speed
new_target = True  # state to denote a new level
button_state = False  # button debouncing state
```
Colors

The last piece of setup before the loop is the NeoPixel colors. Using the adafruit_ led_animation library, you can insert colors easily to assign to the NeoPixels without having to determine RGB values.

This array of colors will be used to cycle through colors as you advance through the game, using now_color and next_color to index your position in the array. now_color will be the color of the chaser pixel and next_color will be the color of the target pixel.

```python
# neopixel colors
colors = [color.RED, color.ORANGE, color.YELLOW, color.GREEN, color.TEAL, color.CYAN,
         color.BLUE, color.PURPLE, color.MAGENTA, color.GOLD, color.AQUA, color.PINK]
```

The Loop

The loop begins with an if statement for button debouncing.

```python
# button debouncing
if not button.value and not button_state:
    button_state = True
```

Randomized Target

x, y and z hold the target pixel positions. y is setup to hold a random integer within the range of 5 and 55. This position is reset every time you advance a level. x and z are setup to be on either side of y.

x and z are setup to be white to highlight the target pixel, which is setup to be the next_color index in the colors array.

```python
# if new level starting..
if new_target:
    # randomize target location
    y = int(random.randint(5, 55))
    x = int(y - 1)
    z = int(y + 1)
    new_target = False
    print(x, y, z)
    pixels[x] = color.WHITE
    pixels[y] = colors[next_color]
    pixels[z] = color.WHITE
```
Playing the Game

Game play begins using `time.monotonic()` instead of `time.sleep()` to delay the loop. `time.sleep()` delays the entire loop, whereas when you use `time.monotonic()`, the timing is tracked without stopping the entire loop.

```
# delay without time.sleep()
if (pixel + speed) < time.monotonic():
```

Chaser Pixel Animation

Before getting into hitting pressing the button to hit the target NeoPixel, there are a few things that need to happen in the code in order for the chaser pixel to move.

First, when the chaser pixel (tracked with `num`) moves forward, the previous pixel is turned off. This previous pixel is tracked with `last_num`.

```
# turn off pixel behind chaser
if num > 0:
    last_num = num - 1
    pixels[last_num] = color.BLACK
    pixels.show()
```

You also want to keep the target pixels their preset colors, even as the chaser pixel passes them.

```
# keep target pixels their colors when the chaser passes
if last_num in (x, y, z):
    pixels[x] = color.WHITE
    pixels[y] = colors[next_color]
    pixels[z] = color.WHITE
```

How does the chaser pixel move though? While the position of the chaser pixel is less than the total number of NeoPixels, it advances by one pixel position.

```
# move chaser pixel by one
if num < num_pixels:
    pixels[num] = colors[now_color]
    pixels.show()
    #print(num)
    #print("target is", y)
    num += 1
```

When the chaser reaches the end of the NeoPixel strip, `num` is reset to `0` to send the chaser back to the beginning of the strip.
# send chaser back to the beginning of the circle
if num == num_pixels:
    last_num = num\n- 1
    pixels[last_num] = color.BLACK
    pixels.show()
    num = 0

Level-Up

Using the button, you'll try and line-up the chaser pixel with the target pixel to score. When you score, all of the NeoPixels will light-up with the `next_color`, which the target pixel had been showing.

The position of the chaser pixel is reset to 0 and the `speed` is increased by the increment stored in `level`. The `now_color` and `next_color` indexes are also increased by 1. Finally, `new_target` is set to True in order to setup a new target pixel.

```python
# if the chaser hits the target...
if last_num in [x, y, z] and not button.value:
    button_state = False
    # fills with the next color
    pixels.fill(colors[next_color])
    pixels.show()
    print(num)
    print(x, y, z)
    # chaser resets
    num = 0
    time.sleep(0.5)
    pixels.fill(color.BLACK)
    pixels.show()
    # speed increases for next level
    speed = speed - level
    # color updates
    next_color = next_color + 1
    if next_color > 11:
        next_color = 0
    now_color = now_color + 1
    if now_color > 11:
        now_color = 0
    # setup for new target
    new_target = True
    print("speed is", speed)
    print("button is", button.value)
```

Missing the Target

If you miss the target pixel with your button press, all of the NeoPixels will light-up with the `now_color`. This is the same color that the chaser pixel had been.
Then, the `game_over()` function animates the NeoPixels by using `color_chase` to turn all of the pixels off and then flash them all red.

To set things up for a new game, the position of the chaser pixel is reset to 0 and the `speed` is reset to its default value. The `now_color` and `next_color` indexes are also reset. Finally, `new_target` is set to `True` in order to setup a new target pixel.

```python
# if the chaser misses the target...
    if last_num not in [x, y, z] and not button.value:
        button_state = False
        print(num)
        print(x, y, z)
        # fills with current chaser color
        pixels.fill(colors[now_color])
        pixels.show()
        # function to flash all pixels red
        game_over()
        # chaser is reset
        num = 0
        pixels.fill(color.BLACK)
        pixels.show()
        # speed is reset to default
        speed = 0.1
        # colors are reset
        next_color = 1
        now_color = 0
        # setup for new target
        new_target = True
        print("speed is", speed)
        print("button is", button.value)
```

You Win!

If you happen to be an expert at the NeoPixel Run Game, you'll defeat all of the targets with your chaser pixel and eventually run out of levels. You'll know you've won when all of the NeoPixels animate using the classic `rainbow_cycle()` animation.

After some fun rainbows, all of the game parameters are reset to their defaults so that you can begin playing again.

```python
# when you have beaten all the levels...
    if speed < final_level:
        # rainbows!
        rainbow_cycle(0.01)
        time.sleep(1)
        # chaser is reset
        num = 0
        pixels.fill(color.BLACK)
        pixels.show()
        # speed is reset to default
        speed = 0.1
        # colors are reset
        next_color = 1
        now_color = 0
```
# setup for new target
new_target = True

Time Tracking

The last line of the loop updates `pixel` to grab the current `time.monotonic()` time.

```python
# time.monotonic() is reset for the delay
pixel = time.monotonic()
```

Assemble

Solder LED Strip
Cut a high density Neopixel strip to 61 pixels and use silicone ribbon cable to help keep the wires bundled.

Tin the pads on the NeoPixel strip and solder wires to the data and power and ground pads.

Slide switch
Tin two pins on the slide switch. Add heat shrink to each wire and then solder to each pin on the slide switch.

Use the side of the solder iron to heat and shrink the connections.
Solder quick connects
Use one of the ground connections close to the prototyping area to solder the quick connect wires for the button. The other wire connects to pin 5.

Test Circuit
Connecting the button and battery to the Feather to quickly test the connections.

Mount Feather
Orient the Feather board so the USB connection is facing the cut out on the case.

Use four M2.5 x 5mm long screws to secure the Feather board to the standoffs on the case.
Mount LED strip
Pass the LED strip wires through the cutoff on the inside of the case. Wrap the LED strip along the inner wall of the case.

Use kapton tape to cover the pads on the end of the NeoPixel strip so the pads doesn't short the circuit.

Mount slide switch
Place the switch to the middle position and then angle the whole slide switch so it can press fit between the three walls on the case.
Mount diffuser
Arrange the printed diffuser with the bigger lip face up. Gently bend the diffuser so it can press fit over the LED strip.

Plug in Battery
Plug in the JST extension cable to the Feather board and then attach the battery. Use foam tape to secure the battery to the case.

Attach Shelf
Use pliers to slightly bend the connections to fit inside the case.

Mount Button
Attach the printed button shelf to the button.
Align the button with the crown icon as shown. Pass the button at an angle and then twist clockwise to mount the button.
Connect button
Attach the quick connect wires to the button. The wires attach to the connections close the points on the crown icon.

Align Lid
Match the cut out on the lid to the slide switch. Use slight force to press fit each nub around the lid on to the case.