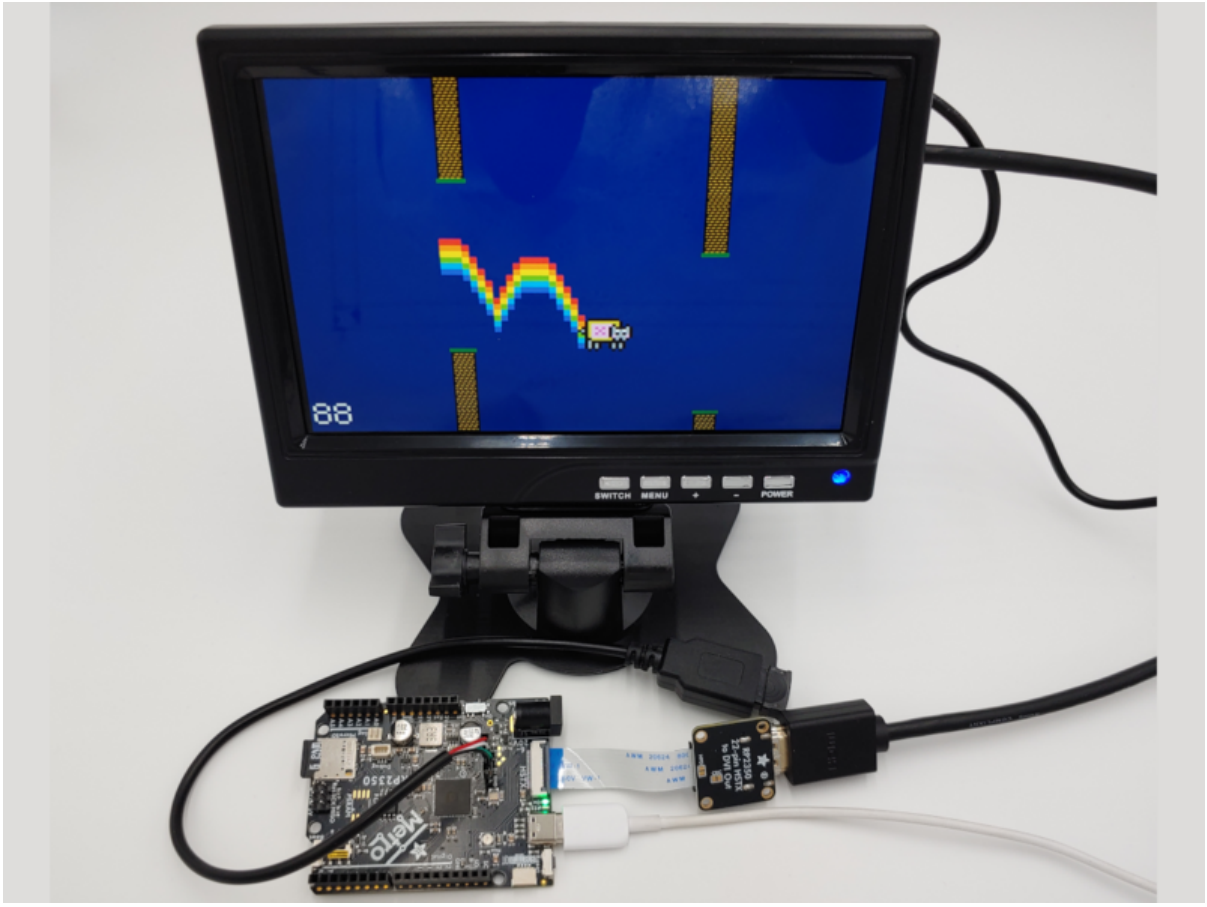
Flappy Nyan Cat Game on Metro RP2350

Created by Tim C

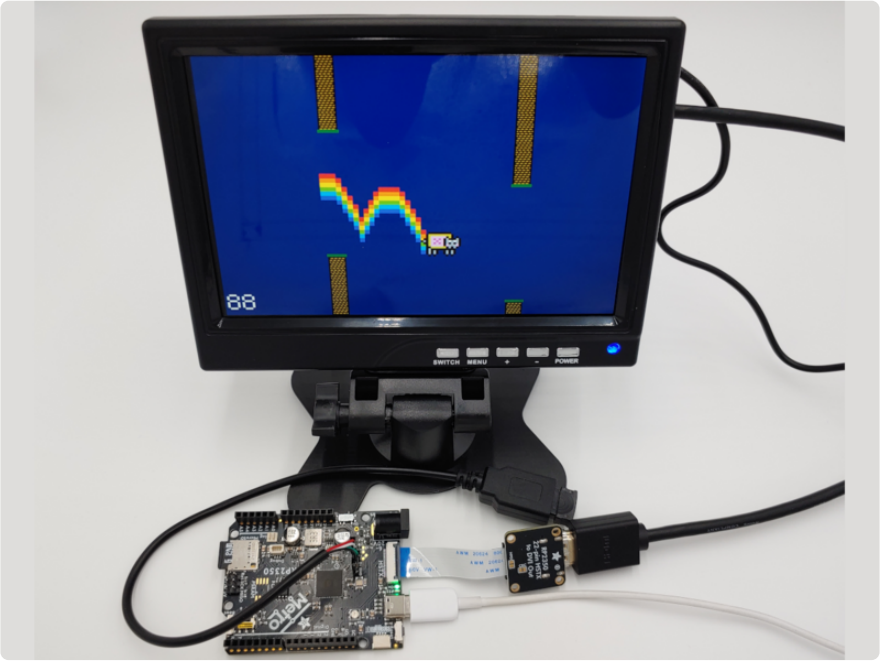


https://learn.adafruit.com/flappy-nyan-cat-game-on-metro-rp2350

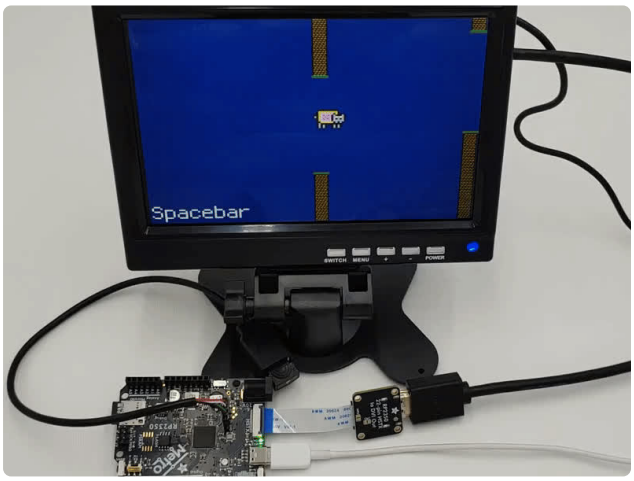
Last updated on 2025-03-13 04:51:33 PM EDT

Table of Contents

Overview

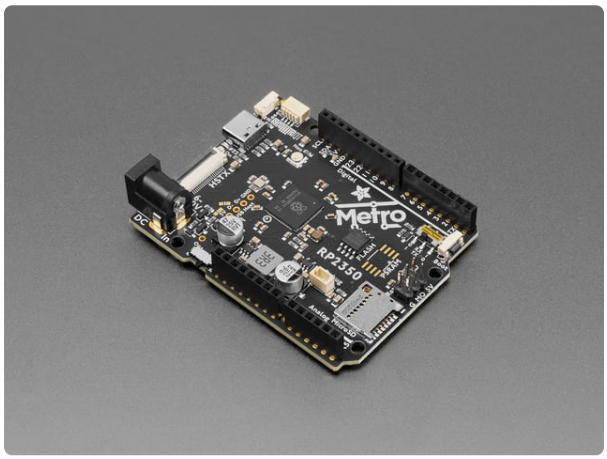


The Metro RP2350 makes a perfect little game console. The on-board HSTX combined with a DVI breakout can output to a standard television or computer monitor for the display. The broken out USB host connections make it easy to take input from a keyboard to control the game.



This game features play inspired by flappy bird. As you fly, gravity pulls you down. You can press spacebar to jump and boost yourself back up. You must avoid the scratching posts and the top and bottom edges of the display. However, instead of a bird, you play as Nyan Cat leaving the iconic rainbow trail in your wake.

Parts

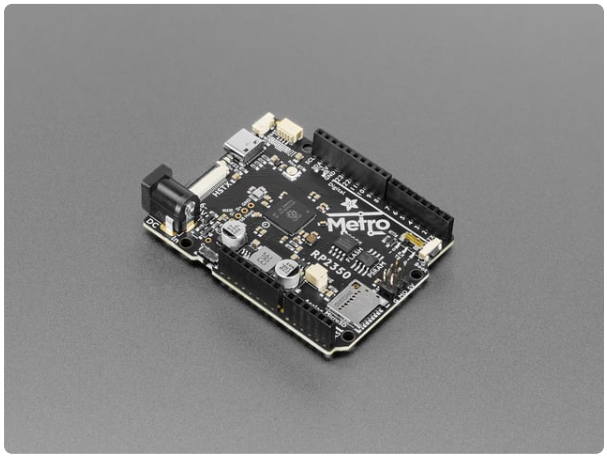


Adafruit Metro RP2350

Choo! Choo! This is the RP2350 Metro Line, making all station stops at "Dual Cortex M33 mountain", "528K RAM round-about" and "16 Megabytes of Flash...

https://www.adafruit.com/product/6003

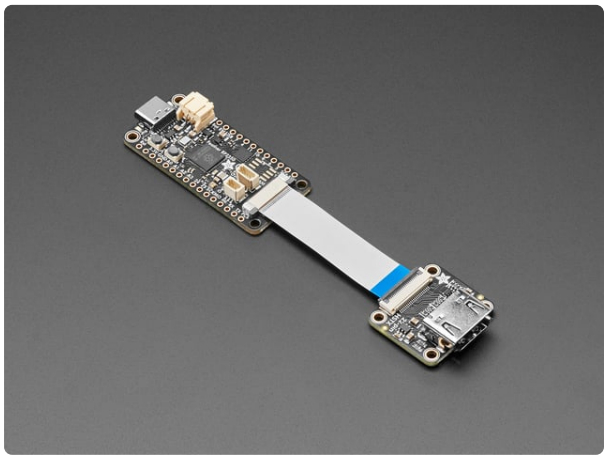
Or



Adafruit Metro RP2350 with PSRAM

Choo! Choo! This is the RP2350 Metro Line, making all station stops at "Dual Cortex M33 mountain", "528K RAM round-about" and "16 Megabytes of Flash town"...

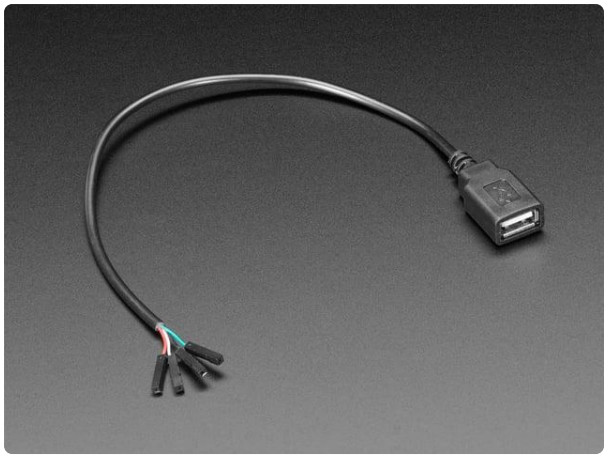
https://www.adafruit.com/product/6267



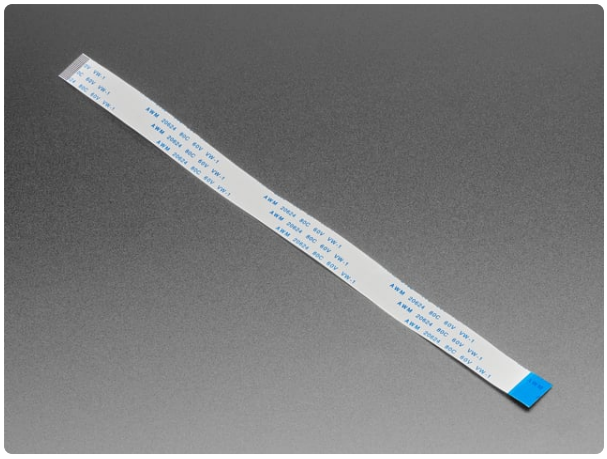
Adafruit RP2350 22-pin FPC HSTX to DVI Adapter for HDMI Displays

You may have noticed that our RP2350 Feather has an FPC output connector on the end for accessing the HSTX (High Speed...

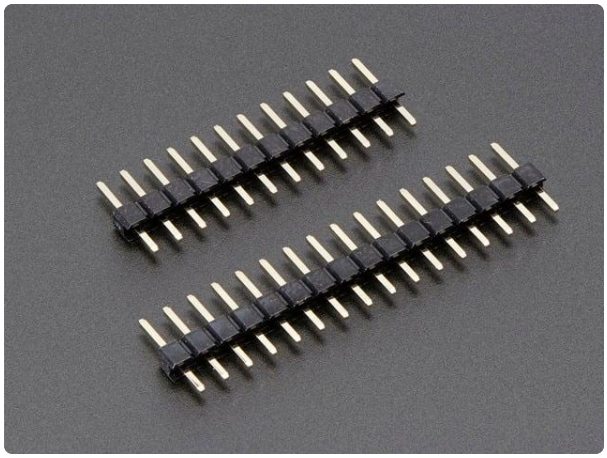
https://www.adafruit.com/product/6055



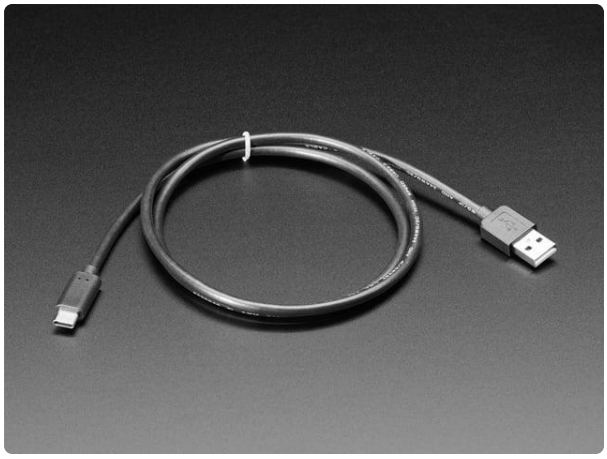
[USB Type A Jack Breakout Cable with Premium Female Jumpers]
If you'd like to connect a USB-host-capable chip to your USB peripheral, this cable will make the task very simple. There is no converter chip in this...
https://www.adafruit.com/product/4449



[22-pin 0.5mm pitch FPC Flex Cable for DSI CSI or HSTX - 20cm]
Connect this to that when a 22-pin FPC connector is needed. This 20 cm long cable is made of a flexible PCB. It's A-B style, meaning that pin one on one side will match with pin...
https://www.adafruit.com/product/6036



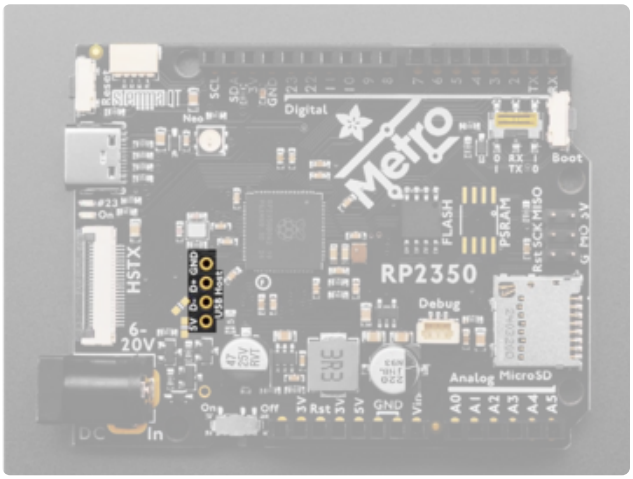
[Short Feather Male Headers - 12-pin and 16-pin Male Header Set]
These two Short Male Headers alone are, well, lonely. But pair them with any of our
https://www.adafruit.com/product/3002

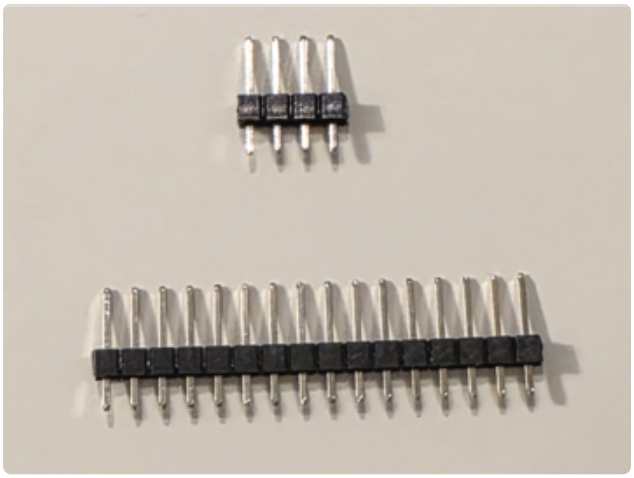


[USB Type A to Type C Cable - approx 1 meter / 3 ft long]
As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...
https://www.adafruit.com/product/4474

Preparing the Metro RP2350

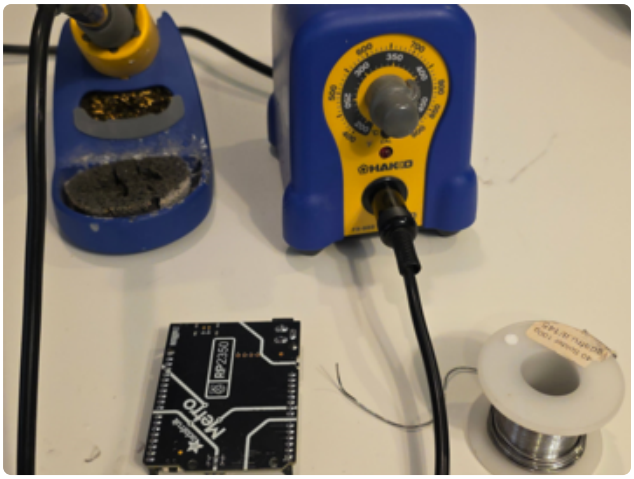
The USB Host port is the only part of this project that required soldering.

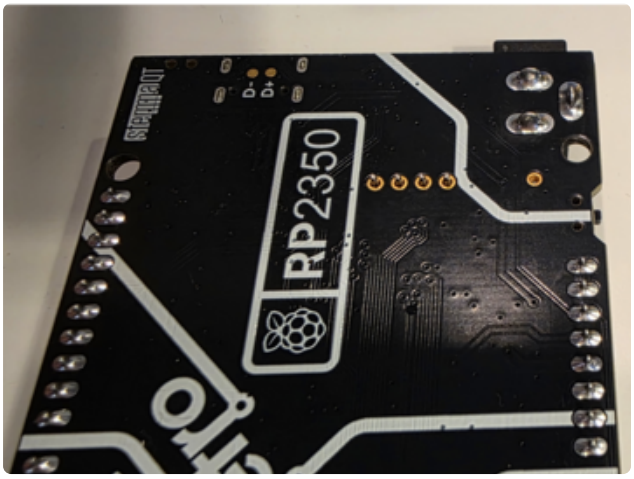


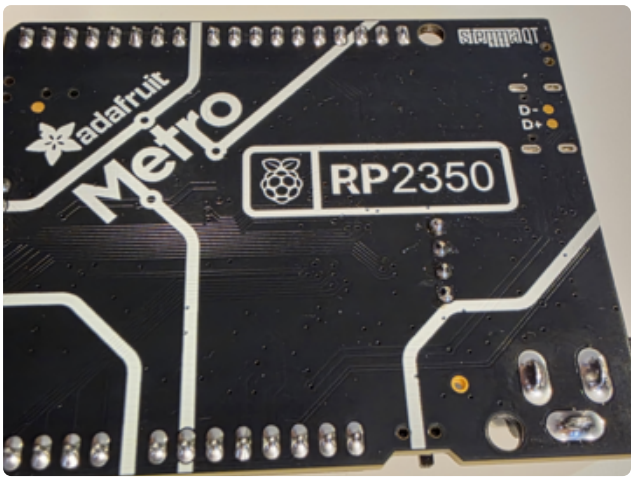


The USB Host pin connections are highlighted on the Metro image to the left. You will need a small piece of standard 0.1 inch male header, with 4 pins, to fit the holes.

You can cut header with diagonal cutters or break them with pliers or even your fingers. Just be sure to wear eye protection as they can fly when cut.

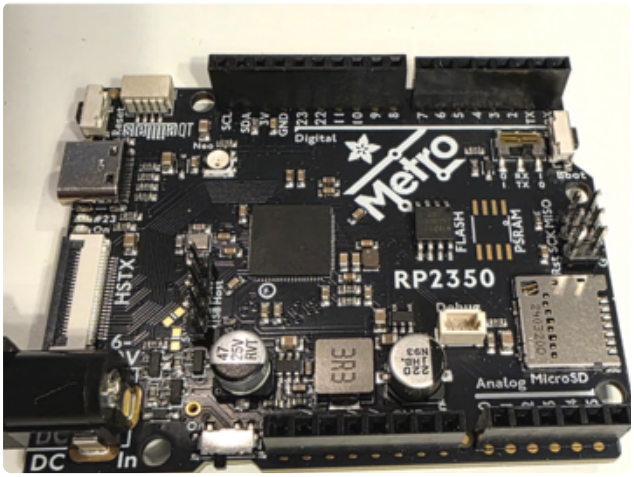






Put the short end of the header into the holes in the Metro marked USB Host and secure them with putty, blutack, tape, etc. Turn the Metro over and you should see the header barely poking out of the bottom of the board. If the pins stick through a great deal you may have the header pins upside down, double check the short end is sticking into the board.

Solder the 4 pin "nubbins" to the board.



Turn the board over and remove the material securing the pins. Now there is a new 4-pin header.

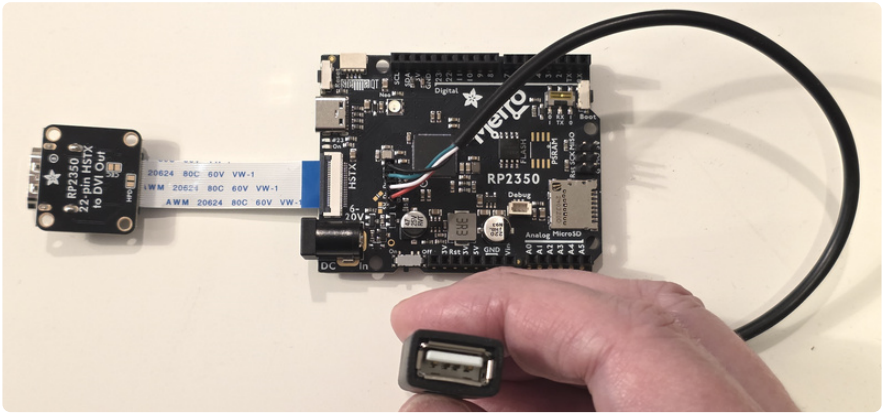
Get the USB Host cable and wire as follows:

GRD to **Black**

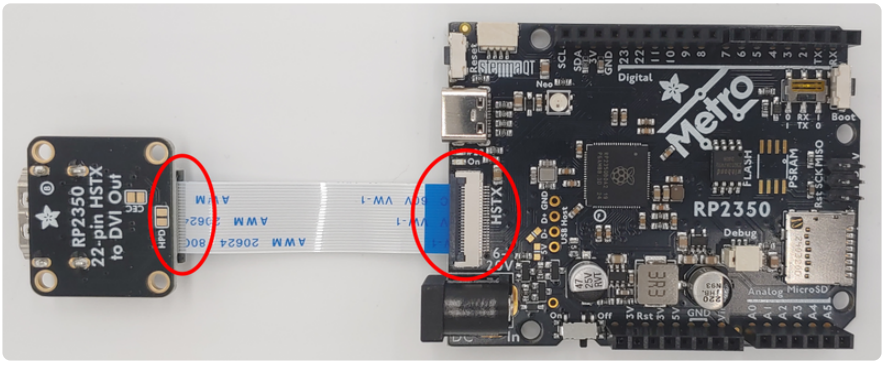
D+ to **Green**

D- to **White**

5V to **Red**



HSTX Connection to DVI



Get the HSTX cable. Any length Adafruit sells is fine. CAREFULLY lift the dark grey bar up on the Metro, insert the cable silver side down, blue side up, then put the bar CAREFULLY down, ensuring it locks. If it feels like it doesn't want to go, do not force it.

Do the same with the other end and the DVI breakout. Note that the DVI breakout will be inverted/upside down when compared to the Metro - this is normal for these boards and the Adafruit cables.

Install CircuitPython

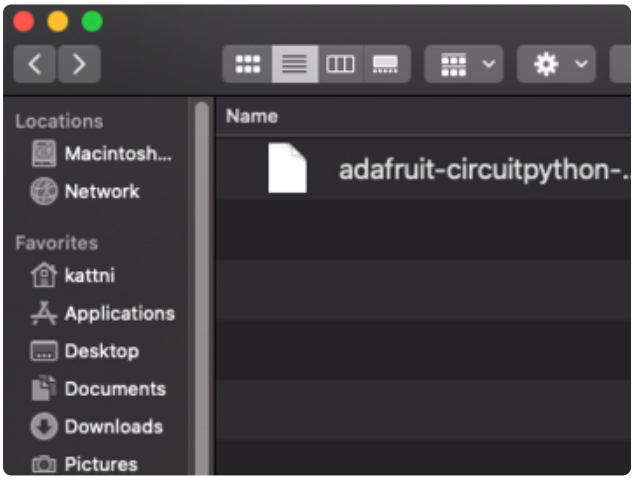
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

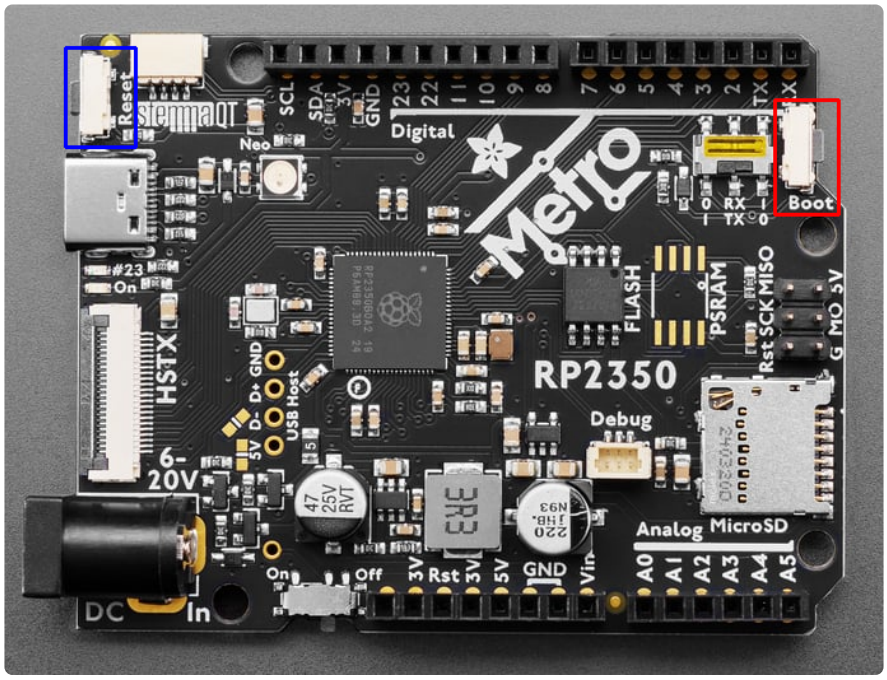
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/1aeL



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

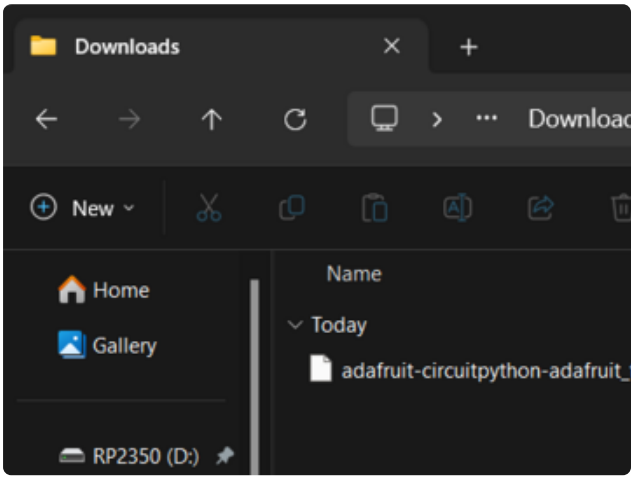


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in red or blue above). **Continue to hold the BOOT/BOOTSEL button until the RP2350 drive appears!**

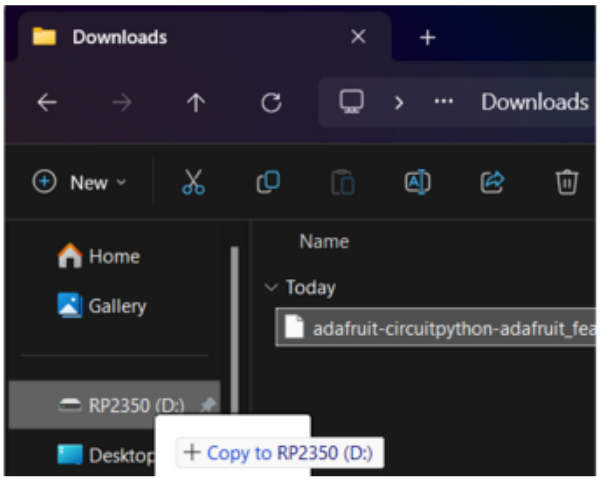
If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.

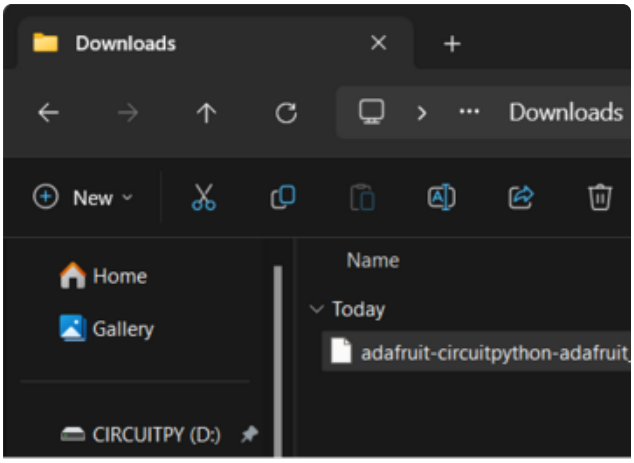
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **RP2350**.



Drag the **adafruit_circuitpython_etc.uf2** file to **RP2350.**



The **RP2350** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

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

Safe Mode

You want to edit your **code.py** or modify the files on your **CIRCUITPY** drive, but find that you can't. Perhaps your board has gotten into a state where **CIRCUITPY** is read-only. You may have turned off the **CIRCUITPY** drive altogether. Whatever the reason, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in **boot.py** (where you can set **CIRCUITPY** read-only or turn it off completely). Second, it does not run the code in **code.py**. And finally, it does not automatically soft-reload when data is written to the **CIRCUITPY** drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the **CIRCUITPY** drive.

Entering Safe Mode

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

If you successfully enter safe mode on CircuitPython, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot.
Press again to exit safe mode.

Press any key to enter the REPL. Use CTRL-D to reload.
```

You can now edit the contents of the **CIRCUITPY** drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.

Flash Resetting UF2

If your board ever gets into a really weird state and CIRCUITPY doesn't show up as a disk drive after installing CircuitPython, try loading this 'nuke' UF2 to RP2350. which will do a 'deep clean' on your Flash Memory. **You will lose all the files on the board**, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

Download flash erasing "nuke" UF2

https://adafru.it/1afi

Code

CircuitPython Usage

To use the game, you need to update **code.py** with the game program to the **CIRCUITPY** drive.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file.

Connect your board to your computer via a known good data+power USB cable. The board should show up in your File Explorer/Finder (depending on your operating system) as a flash drive named **CIRCUITPY**.

Extract the contents of the zip file, copy the **lib** directory files to **CIRCUITPY/lib**. Copy the **code.py** file to your **CIRCUITPY** drive. The program should self start.

Drive Structure

After copying the files, your drive should look like the listing below. It can contain other files as well, but must contain these at a minimum.

Code

The **code.py** for the project is shown below.

```
# SPDX-FileCopyrightText: 2025 Tim Cocks for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
A game featuring Nyan cat inspired by flappy bird.
Learn Guide: https://learn.adafruit.com/flappy-nyan-cat-game-on-metro-rp2350

Controls:
    Spacebar: Jump
           S: Switch Trails
           P: Play again after game over
           Q: Quit after game over
"""
import random
import sys
import terminalio
from displayio import Group, TileGrid, Bitmap, release_displays, Palette
import supervisor
import bitmaptools
from adafruit_display_text.bitmap_label import Label
import picodvi
import framebufferio
import board
from micropython import const
import adafruit_imageload

# how strong the gravity is
FALL_SPEED = 1

# how powerful the jump is
JUMP_SPEED = 5

# maximum gravity speed
TERMINAL_VELOCITY = 7

# make "close calls" more likely by fudging the collision check
# in favor of the player a bit
COLLIDE_FUDGE_FACTOR = 8

# how many scaled pixels wide the trail will be
TRAIL_LENGTH = 20

# current score
score = 0
```

```
# initialize display
release_displays()

fb = picodvi.Framebuffer(
    320,
    240,
    clk_dp=board.CKP,
    clk_dn=board.CKN,
    red_dp=board.D0P,
    red_dn=board.D0N,
    green_dp=board.D1P,
    green_dn=board.D1N,
    blue_dp=board.D2P,
    blue_dn=board.D2N,
    color_depth=16,
)
display = framebufferio.FramebufferDisplay(fb)

# initialize groups to hold visual elements
main_group = Group()

# any elements in this Group will be scaled up 2x
scaled_group = Group(scale=2)
main_group.append(scaled_group)


class Post(Group):
    # gap location constants
    GAP_TOP = const(0)
    GAP_MID = const(1)
    GAP_BOTTOM = const(2)

    def __init__(self, spritesheet, gap_location=GAP_MID):
        """
        A pair of scratching posts, aligned vertically. This class
        holds the visual elements, and provides collision checking.

        :param Union[Bitmap,OnDiskBitmap] spritesheet: The Bitmap containing the
post sprite sheet.
        :param gap_location: Where the gap should be. Must be one of GAP_TOP,
GAP_MID, GAP_BOTTOM.
        """
        super().__init__()
        # start out not visible
        self.hidden = True

        # hold a reference to the spritesheet Bitmap
        self.sprites = spritesheet

        # check which gap location was specified and
        # set the post heights accordingly
        if gap_location == Post.GAP_MID:
            top_height = 4
            bottom_height = 4
        elif gap_location == Post.GAP_BOTTOM:
            top_height = 7
            bottom_height = 1
        elif gap_location == Post.GAP_TOP:
            top_height = 1
            bottom_height = 7
        else:
            raise ValueError("Invalid gap_location")

        # initialize top post TileGrid
        self.top_post = TileGrid(
            post_sprites,
            pixel_shader=post_sprites_pixel_shader,
            height=top_height,
```

```
            width=1,
            tile_width=16,
            tile_height=16,
            default_tile=3,
        )

        # set the tiles for the top post.
        # Normal double ended post tiles with
        # the bottom cap tile below them.
        for i in range(top_height):
            if i == top_height - 1:
                self.top_post[0, i] = 2
            else:
                self.top_post[0, i] = 1

        # initialize bottom post TileGrid
        self.bottom_post = TileGrid(
            post_sprites,
            pixel_shader=post_sprites_pixel_shader,
            height=bottom_height,
            width=1,
            tile_width=16,
            tile_height=16,
            default_tile=3,
        )

        # set the tiles for the bottom post.
        # Normal double ended post tiles
        # with the top cap tile above them
        for i in range(bottom_height):
            if i == 0:
                self.bottom_post[0, i] = 0
            else:
                self.bottom_post[0, i] = 1

        # move the bottom post to the bottom of the display
        self.bottom_post.y = 240 - bottom_height * 16

        # append both post TileGrids to super class Group instance
        self.append(self.top_post)
        self.append(self.bottom_post)

    def check_collision(self, sprite):
        """
        Check if either of our top or bottom posts are colliding with the given
sprite
        :param sprite: The sprite to check collision against.
        :return: True if sprite is colliding with a post, false otherwise.
        """
        # if the sprite is horizontally aligned with the posts
        if (
            (sprite.x * 2) - self.top_post.tile_width
            <= self.x
            <= (sprite.x * 2) + (sprite.tile_width * 2)
        ):

            # if the sprite is within the vertical range for either top or bottom
post
            if (
                sprite.y * 2
            ) + COLLIDE_FUDGE_FACTOR <= self.top_post.tile_height *
self.top_post.height or (
                sprite.y * 2
            ) - COLLIDE_FUDGE_FACTOR >= self.bottom_post.y - (
                sprite.tile_height * 2
            ):
                return True

        return False  # no collision
```

```
class PostPool:
    def __init__(self):
        """
        A pool of Post objects to pull from and recycle back into.
        """

        # list to store the Posts in
        self.pool = []

        # start with 2 of each gap location
        self.pool.append(Post(post_sprites, Post.GAP_MID))
        self.pool.append(Post(post_sprites, Post.GAP_MID))
        self.pool.append(Post(post_sprites, Post.GAP_BOTTOM))
        self.pool.append(Post(post_sprites, Post.GAP_BOTTOM))
        self.pool.append(Post(post_sprites, Post.GAP_TOP))
        self.pool.append(Post(post_sprites, Post.GAP_TOP))

    def get_post(self, index=None):
        """
        Get an available Post from the pool.

        :param index: The index of the post to return.
            Default is None, which means random.

        :return: An available Post object.
        """
        # if index is none, generate a random index
        if index is None:
            rnd_idx = random.randint(0, len(self.pool) - 1)

        else:  # index not None
            # use the provided index.
            rnd_idx = index

        # select a Post and remove it from the pool
        next_post = self.pool.pop(rnd_idx)

        # make the post visible
        next_post.hidden = False

        # return the post
        return next_post

    def recycle_post(self, post):
        """
        Recycle a Post back into the pool

        :param Post post: The post to recycle.

        :return: None
        """
        # set the post to not visible
        post.hidden = True

        # add the post to the pool
        self.pool.append(post)


class GameOverException(Exception):
    """
    Exception that will be raised when the player loses the game.
    """

    def __init__(self, msg):
        self.msg = msg
        super().__init__(self.msg)
```

```
# palette of colors for the trail
trail_palette = Palette(10)
# rainbow colors
trail_palette[0] = 0x000000
trail_palette[1] = 0xE71C1F
trail_palette[2] = 0xF39816
trail_palette[3] = 0xF1E610
trail_palette[4] = 0x6DB52F
trail_palette[5] = 0x428CCB
trail_palette[6] = 0x4B4C9C

# trans flag colors
trail_palette[7] = 0xF5ABB9
trail_palette[8] = 0x5BCFFA
trail_palette[9] = 0xFFFFFF

# setup color index 0 for transparency
trail_palette.make_transparent(0)

# Bitmap that holds 1 pixel width of the trail
trail_bmp = Bitmap(1, 6, 7)

# initialize the Bitmap pixels to the rainbow colors
trail_bmp[0, 0] = 1
trail_bmp[0, 1] = 2
trail_bmp[0, 2] = 3
trail_bmp[0, 3] = 4
trail_bmp[0, 4] = 5
trail_bmp[0, 5] = 6

# Bitmap for the background, 1/10 of 160x120 which is the
# of the display area accounting for the 2x from the scaled_group
bg_bmp = Bitmap(16, 12, 1)

# palette for the background
bg_palette = Palette(1)
bg_palette[0] = 0x00014F  # dark blue
bg_tilegrid = TileGrid(bg_bmp, pixel_shader=bg_palette)

# Group for the background scaled to 10x
bg_group = Group(scale=10)

# add the background to it's group and add that to the scaled_group
bg_group.append(bg_tilegrid)
scaled_group.append(bg_group)

# load the sprite sheet for the posts
post_sprites, post_sprites_pixel_shader = adafruit_imageload.load(
    "scratch_post_sprites.bmp"
)

# set up color index 0 for transparency
post_sprites_pixel_shader.make_transparent(0)

# initialize a PostPool() which will start with 2 posts
# of each gap location
post_pool = PostPool()

# add all posts to the main_group. Note, not the scaled_group.
# posts are displayed at 1x size.
for _post in post_pool.pool:
    main_group.append(_post)

# get the first_post out of the pool
first_post = post_pool.get_post()

# move it to the right edge
first_post.x = 320 - 16
```

```
# second post starts near the cat, so we want it to be
# middle gap to start with always. middle gap starts in index 0.
second_post = post_pool.get_post(0)

# move it to the center
second_post.x = 160

# Group with an additional 2x scaling to hold the rainbow trail
canvas_group = Group(scale=2)

# Bitmap for the trail canvas 1/4 display size for 2x from scaled_group
# and 2x from canvas_group
trail_canvas_bmp = Bitmap(display.width // 4, display.height // 4, 10)

# TileGrid for the trail canvas
trail_canvas_tg = TileGrid(trail_canvas_bmp, pixel_shader=trail_palette)

# add the canvas tilegrid to it's group, and add that to the scaled_group
canvas_group.append(trail_canvas_tg)
scaled_group.append(canvas_group)

# load nyan cat Bitmap
nyan_bmp, nyan_bmp_pixel_shader = adafruit_imageload.load("nyancat_16x12.bmp")
# set color index 0 transparent
nyan_bmp_pixel_shader.make_transparent(0)
# TileGrid for cat
nyan_tg = TileGrid(bitmap=nyan_bmp, pixel_shader=nyan_bmp_pixel_shader)
# add cat to scaled_group
scaled_group.append(nyan_tg)

# move cat near the center
nyan_tg.x = 80
nyan_tg.y = 50

# text label for the current score
score_lbl = Label(terminalio.FONT, text="Spacebar", color=0xFFFFFF, scale=2)
# move it to the bottom left corner
score_lbl.anchor_point = (0, 1)
score_lbl.anchored_position = (2, display.height - 2)

# add it to the main_group
main_group.append(score_lbl)

game_over_label = Label(
    terminalio.FONT,
    text="",
    color=0xFFFFFF,
    background_color=0x000000,
    padding_top=10,
    padding_bottom=10,
    padding_left=10,
    padding_right=10,
)
game_over_label.anchor_point = (0.5, 0.5)
game_over_label.anchored_position = (display.width // 2, display.height // 2)
game_over_label.hidden = True

main_group.append(game_over_label)

# set the main_group to show on the display
display.root_group = main_group

# disable auto_refresh
display.auto_refresh = False

# list to store coordinates of each horizontal pixel of the trail
trail_coords = []
```

```
# cat_speed variable holds pixels per tick to move downward for gravity
cat_speed = FALL_SPEED

# print(f"memfree: {gc.mem_free()}")


def swap_trail():
    """
    Swap the trail graphic between rainbow and trans flag colored.
    """
    # if the top pixel is red
    if trail_bmp[0, 0] == 1:
        # change to the trans flag colors
        trail_bmp[0, 0] = 0
        trail_bmp[0, 1] = 8
        trail_bmp[0, 2] = 7
        trail_bmp[0, 3] = 9
        trail_bmp[0, 4] = 7
        trail_bmp[0, 5] = 8
    else:
        # change to rainbow colors
        trail_bmp[0, 0] = 1
        trail_bmp[0, 1] = 2
        trail_bmp[0, 2] = 3
        trail_bmp[0, 3] = 4
        trail_bmp[0, 4] = 5
        trail_bmp[0, 5] = 6


def draw_trail():
    """
    draw the trail in its current location
    """
    # loop over the coordinates of the horizontal pixels
    for coord in trail_coords:
        # blit a copy of the trail Bitmap into the canvas
        # Bitmap at the current coordinate
        bitmaptools.blit(trail_canvas_bmp, trail_bmp, coord[0], coord[1])


def erase_trail():
    """
    Erase the trail in its current location
    """
    # loop over the coordinates of the horizontal pixels
    for coord in trail_coords:
        # fill a region the size of trail Bitmap with color_index 0
        # to make it transprent
        bitmaptools.fill_region(
            trail_canvas_bmp, coord[0], coord[1], coord[0] + 1, coord[1] + 6, 0
        )


def shift_trail():
    """
    shift the coordinates of the trail to the left one pixel
    """
    # loop over indexes within the trail coordinates list
    for _ in range(len(trail_coords)):
        # update the x value of the current coordinate by -1
        trail_coords[_][0] -= 1


def shift_post(post):
    """
    shift the coordinates of a post to the left
    :param Post post: The Post to shift
    :return: The shifted Post instance, or the new Post if the old one
        went off the left edge
```

```
    """
    # global score variable so we can update it
    global score  # pylint: disable=global-statement

    # if the post is at the left edge
    if post.x <= 0:
        # add bonus points for each post that makes it to the left edge
        score += 10

        # recycle the Post object back into the pool
        post_pool.recycle_post(post)

        # get another Post out of the pool
        new_post = post_pool.get_post()

        # move it to the right edge
        new_post.x = 320 - 16

        # return the new post
        return new_post

    else:  # post is not at the left edge
        # move it left, getting faster for every 100 score points
        # maxing out at 8 pixels per shift
        post.x -= min((3 + score // 100), 8)

        # return the shifted post
        return post


# initial display refresh
display.refresh(target_frames_per_second=30)
print("Press space to jump")

# boolean to have the game paused to start and wait for the player to begin
playing = False

while True:
    try:
        # if the player hasn't started yet
        if not playing:
            while True:
                # check if any keys were pressed
                available = supervisor.runtime.serial_bytes_available

                # if one or more keys was pressed
                if available:
                    # read the value
                    cur_btn_val = sys.stdin.read(available)
                else:
                    cur_btn_val = None

                # if spacebar was pressed
                if cur_btn_val == " ":
                    # do the first jump
                    cat_speed = -JUMP_SPEED

                    # set playing to true and breakout of the pause loop
                    playing = True
                    break

        # check if the cat is touching the first post
        if first_post.check_collision(nyan_tg):
            raise GameOverException(
                f"Kitty got distracted by the scratchers post.\nScore: {score}"
            )

        # check if the cat is touching the second post
        if second_post.check_collision(nyan_tg):
```

```
            raise GameOverException(
                f"Kitty got distracted by the scratchers post.\nScore: {score}"
            )

        # check if any keyboard data is available
        available = supervisor.runtime.serial_bytes_available
        if available:
            # read the data if there is some available
            cur_btn_val = sys.stdin.read(available)
        else:
            cur_btn_val = None

        # apply gravity to the cat, maxing out at terminal velocity
        cat_speed = min(cat_speed + FALL_SPEED, TERMINAL_VELOCITY)

        # if there is keyboard data and spacebar was pressed
        if cur_btn_val is not None and " " in cur_btn_val:
            cat_speed = -JUMP_SPEED

            # award a point for each jump
            score += 1
        elif cur_btn_val is not None and "s" in cur_btn_val:
            swap_trail()
            # award a point for swapping the trail
            score += 1

        # move the cat down by cat_speed amount of pixels
        nyan_tg.y += cat_speed

        # if the cat has touched the top or bottom edge
        if nyan_tg.y > display.height // 2 or nyan_tg.y < 0:
            raise GameOverException(f"Kitty wandered away.\nScore: {score}")

        # current coordinates of the cat
        draw_coords = [nyan_tg.x // 2, nyan_tg.y // 2]

        try:
            # erase the trail
            erase_trail()
        except ValueError as exc:
            raise GameOverException(f"Kitty wandered away.\nScore: {score}") from
exc

        # shift the trail coordinates over
        shift_trail()

        # add new coordinates to the trail at the cats current location
        trail_coords.append(draw_coords)

        # if the trail is at its maximum length
        if len(trail_coords) > TRAIL_LENGTH:
            # remove the oldest coordinate from the trail coordinates list.
            trail_coords.pop(0)

        # draw the trail
        draw_trail()

        # shift the posts over
        first_post = shift_post(first_post)
        second_post = shift_post(second_post)

        # update the score label
        score_lbl.text = str(score)

        # refresh the display
        display.refresh(target_frames_per_second=30)

    except GameOverException as e:
        # update the game over message
```

```
        game_over_label.text = str(f"{e.msg}\nPress P to play again\nPress Q to
quit")
        # make the game over message visible
        game_over_label.hidden = False

        # refresh display so the message shows
        display.refresh()
        break

# wait for the player to press a key
while True:
    # check if any keys were pressed
    available = supervisor.runtime.serial_bytes_available

    # if one or more keys was pressed
    if available:
        # read the value
        cur_btn_val = sys.stdin.read(available)

        # if player pressed p
        if "p" in cur_btn_val:
            supervisor.set_next_code_file(__file__)
            supervisor.reload()

        # if player pressed q
        elif "q" in cur_btn_val:
            print("exiting")
            break
```

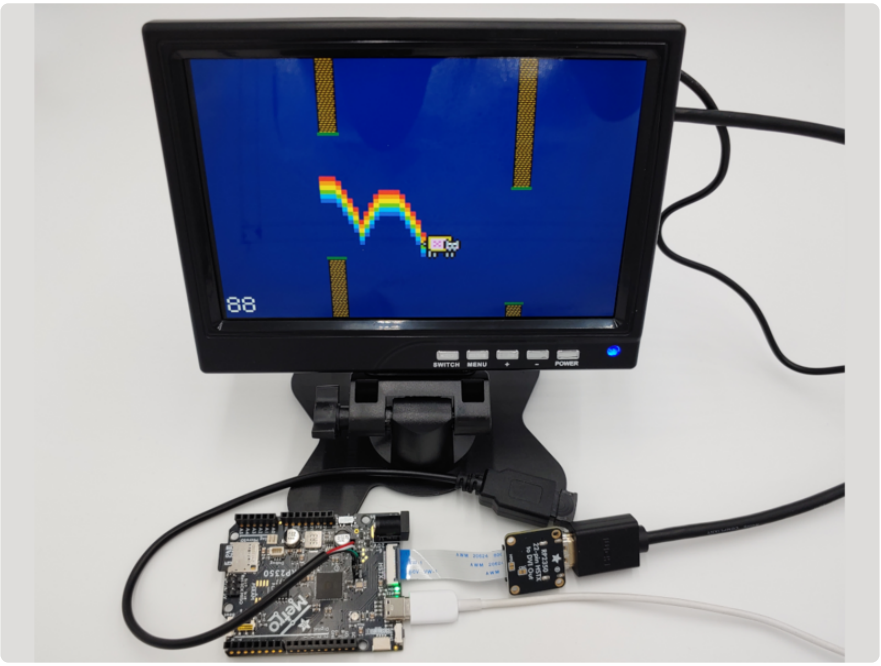
Usage

Ensure a USB keyboard is plugged into the USB Host port wired up previously. Reset the board by cycling power or pressing the Reset button if you happen to plug in a keyboard after power up.

Be sure you connect the DVI breakout to an HDMI monitor and the monitor is on. You might need a long cable if your monitor is not near the Metro RP2350 (like a television). The cables are standard and may be obtained from any trusted retail outlet. Also reset the Metro if you plug in HDMI after powering the Metro.

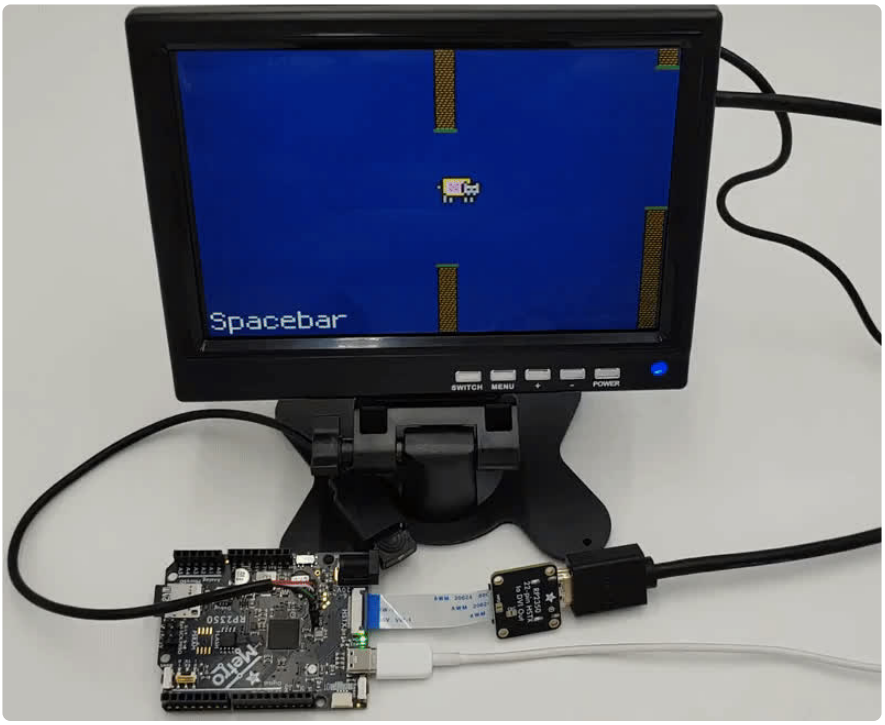
Once connections are all set, power the Metro RP2350 either via USB C (5 volts) or the barrel connection (5.5 to 17 volts DC, center positive).

Gameplay



The controls for the game are as follows:

- **Spacebar**: Start the game, and make the cat jump.
- **S**: During play, press **S** to swap the trail between rainbow and trans flag colors.
- **P**: After game over, press **P** to play again.
- **Q**: After game over, press **Q** to quit.



The goal is to keep the cat flying for as long as you can without running into either the scratching posts, or the edges of the screen. The player is awarded 1 point for each jump, 1 point each time the swap the trail colors, and 10 points for each set of posts that fly off the left edge of the screen. For every 100 points that are scored the flying speed gets faster making harder and harder to avoid the posts.

Code Explanation

The code for the game is thoroughly commented with explanations of what each line or section are for. This page will provide a higher level summary of the major components.

Hardware Principals

This game is designed around two primary hardware peripherals: the HSTX connector with a DVI breakout for the display, and a basic USB keyboard for the player control input.

HSTX Display

To initialize the display the built-in core modules `picodvi`, and `framebufferio` are used. These modules support a few different resolutions and color depths. This project is made for the 320x240 resolution with 16 bit color depth. The pixels are automatically doubled before being pushed to the display so it will come out as 640x480, depending on your monitor or TV, it may further upscale it to fit the screen.

USB Keyboard

USB Host is relatively new to CircuitPython, first coming on Raspberry Pi RP2040-based boards.

Typically, one would access a USB port by:

- Establishing the USB connection
- Reading USB Reports, sections of bytes sent when an action occurs on the peripheral like a key is pressed or joystick moved.
- Parsing the reports and providing meaningful input to the program.

Python has the concept of standard input and output streams, similar to those in Linux/Unix and other operating systems. CircuitPython has this capability and through a lot of behind the scenes code, presents a USB keyboard as a stdin input device. The code to get USB Host Keyboard characters and echo them to serial out is as follows:

```
import supervisor
import sys

while True:
    available = supervisor.runtime.serial_bytes_available
    if available:
        c = sys.stdin.read(available)
        print(c, end='')
```

As an added bonus this also means that the game can be played via the USB Serial connection with your PC. Simply connect to your device with your preferred serial console application, make sure the app has focus and anything you type on your PC keyboard gets sent to CircuitPython via the same `stdin` stream it's reading keyboard keys from.

Helper Classes

The game code has 3 helper classes which contain behavior for various parts of the game bundled together as easy to use component objects.

`Post`

The `Post` class extends `displayio.Group` so it can contain `TileGrid`s and other visual elements to be shown on the display. While it's name is `Post` singular, it actually holds a pair of visual posts, one at the top of the screen, and the other at the bottom. In the original flappy bird, these were green pipes. The `check_collision()` function will determine if the cat sprite is colliding with either of the posts in this `Post` instance. These `Post` objects get moved along the screen by updating the `x` coordinate to lower values for each fame of the game.

`PostPool`

`PostPool` is a "grab bag" of `Post`s to store the ones not currently in use, and provide one for us randomly when we need to add a new one to the right edge of the display. The `get_post()` and `recycle_post()` functions are used to get an unused `Post`, and then recycle it back into the pool when it's no longer needed.

`GameOverException`

This is a basic custom exception that the code will raise when the player loses by touching a scratching post, or the top or bottom edges of the screen.

Helper Functions

The game code has 5 helper functions which carry out some of the game play functionality. Each is listed below with a brief description of it's purpose.

- `swap_trail()` - Swap the trail back and forth between rainbow and trans flag colors.
- `draw_trail()` - Draw the trail at the current location.
- `erase_trail()` - Erase the trail at the current location.
- `shift_trail()` - Shift the pixel locations in the trail to the left by one.
- `shift_post()` - Move the posts to the left by one step.

Display Elements

The display elements are broken up into a few different `Group`s with different scale factors applied. Each is listed below with a brief description of what it holds.

- `main_group` - The top level displayio `Group` that holds everything else within the game. The `Post` objects are added directly to this group, and are the only display element that is rendered at 1:1 size instead of scaled up by a `Group`.
- `scaled_group` - This `Group` gets scaled 2x. It holds the background, the cat sprite, and the trail canvas group.
- `bg_group` - This is scaled by 10x and put inside of `scaled_group` for an additional 2x making the total scale factor `20`. That allows it to contain a very small Bitmap with the dark blue background, but scale it all the way up to match the display size.
- `canvas_group` - This group is scaled 2x, and is placed inside of the `scaled_group` which brings the total scaling factor to `4`. It holds the trail canvas Bitmap that the trail pixels are rendered into.
- `trail_bmp` - A Bitmap that is 1px wide and 6px tall. It holds one column of the trail. It's contents will get copied into the canvas `Bitmap` with `bitmaptools.blit()`.
- `nyan_tg` - This `TileGrid` holds the cat sprite that represents the player. It's y location is changed in accordance with the gravity calculation, and jump button. It's `x` location remains static.
- `score_lbl` - A text label that goes in the bottom left corner and shows the current score.
- `game_over_lbl` - A text label that gets splashed on top of the game when the player loses.