



Egg Hunt Maze Game on Fruit Jam

Created by Tim C



<https://learn.adafruit.com/egg-hunt-maze-game-on-fruit-jam>

Last updated on 2026-04-01 09:18:24 PM UTC

Table of Contents

Overview	3
<ul style="list-style-type: none">• Parts	
Install CircuitPython	5
<ul style="list-style-type: none">• CircuitPython Quickstart• Safe Mode• Flash Resetting UF2	
Code	10
<ul style="list-style-type: none">• Getting the Program's Files• Drive Structure	
Play Game	21
<ul style="list-style-type: none">• Maze World• End Level Screen• Egg Collection Screen	
Code Walkthrough Video	23
<ul style="list-style-type: none">• Code	

Overview



In this cute Easter themed game for the Fruit Jam, you control a bunny character running around a 2D maze. While exploring the maze's twists and turns you'll find colorful painted eggs. At the end of each round you can select the eggs you like the most and save them to your permanent collection. Your collection is stored safely in a JSON file on the CPSAVES portion of flash so it remains available even after you power off and back on the Fruit Jam.

The project demonstrates the following CircuitPython game development techniques:

- Randomly generated 2D mazes with `TileGrid` sprites.
- USB game controller input for character movement and UI controls.
- `TileGrid` based collision detection for walls and eggs.
- Layered `TileGrids` for covering the map with fog and revealing an area when the player gets near.

- Using `TilePaletteMapper` to map combinations of colors into the palettes of grayscale base sprites.
- Storing persistent game data in CPSAVES storage.
- Direction specific player sprite animations for movement.

Once you've had your fill of bunnies and colorful eggs you can take and adapt parts of the project code into other CircuitPython games.

Parts



Adafruit Fruit Jam - Mini RP2350 Computer

We were catching up on a recent hackaday hackchat with eben upton and learned some fun facts:...

<https://www.adafruit.com/product/6200>



USB Game Controller with SNES-like Layout

This is a generic USB game controller, which plugs into to provide a two-handed gaming experience for retro gaming, or really any game you want to use a handheld rather than...

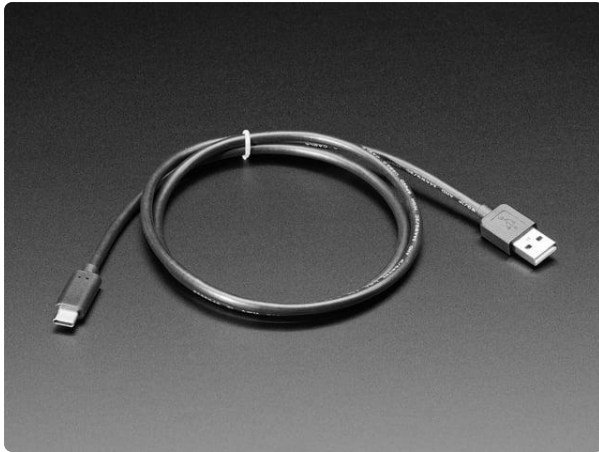
<https://www.adafruit.com/product/6285>



HDMI Cable - 1 meter

Connect two HDMI devices together with this basic HDMI cable. It has nice molded grips for easy installation, and is 1 meter long (about 3 feet). This is a HDMI 1.3...

<https://www.adafruit.com/product/608>



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>



Snap-on Enclosure for Adafruit Fruit Jam

Here is a cool and minimal enclosure for your Fruit Jam to keep it safe during use and transport. This case has been...

<https://www.adafruit.com/product/6425>



7" Display 1280x800 (720p) IPS + Speakers - HDMI/VGA/NTSC/PAL

Yes, this is an adorable small HDMI television with incredibly high resolution and built in 3W stereo speakers! We tried to get the smallest possible HDMI/VGA display with...

<https://www.adafruit.com/product/1667>

Install CircuitPython

[CircuitPython](https://adafru.it/tB7) (<https://adafru.it/tB7>) is a derivative of [MicroPython](https://adafru.it/BeZ) (<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.

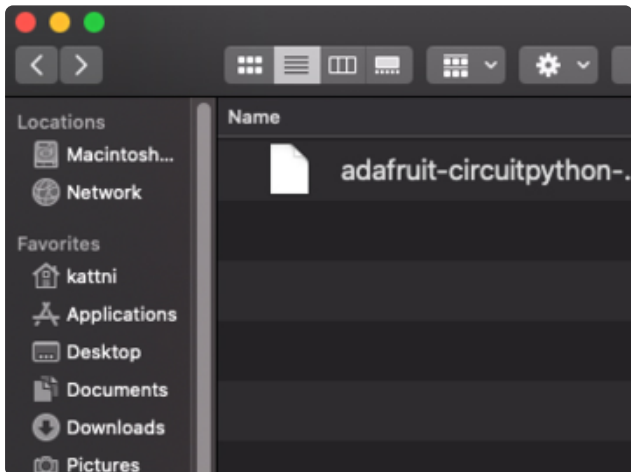
CircuitPython Quickstart

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



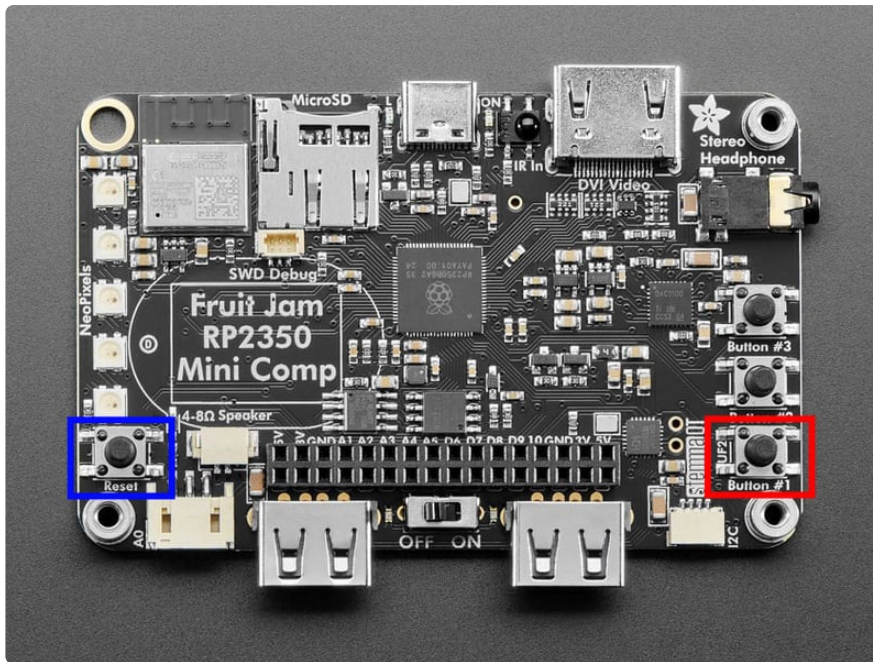
Use the latest release of 10.x or higher for the Fruit Jam. Also use the latest libraries for the best functionality.

<https://adafru.it/1aow>



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

Save it wherever is convenient for you.

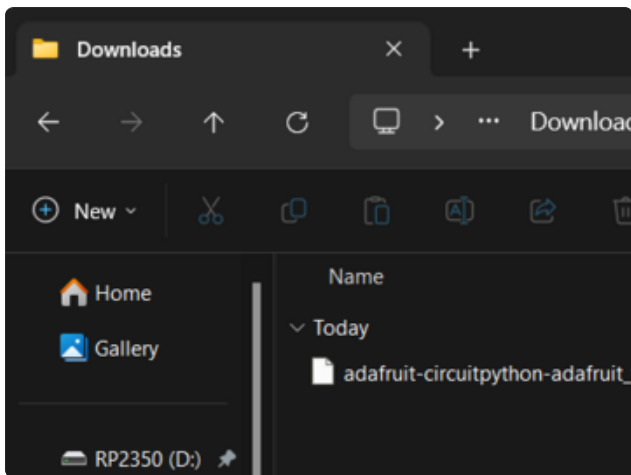


To enter the bootloader, hold down the **BOOT/BOOTSEL** button (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset** button (highlighted in red or blue above). **Continue to hold the BOOT/BOOTSEL button until the RP2350 drive appears!**

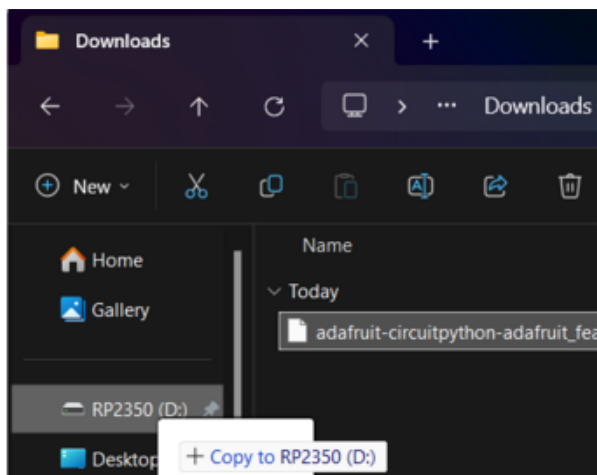
If the drive does not appear, release all the buttons, and then repeat the process above.

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.

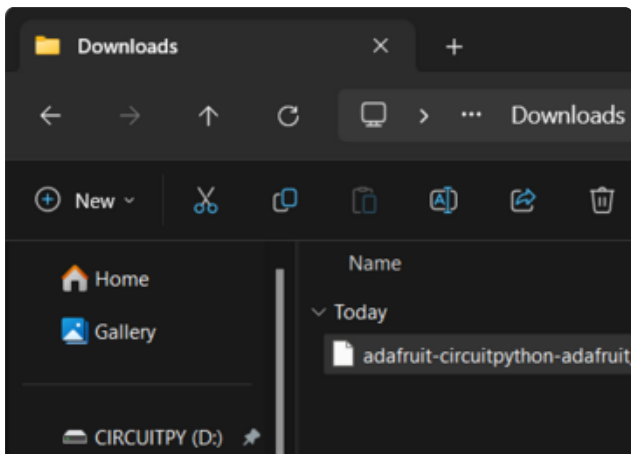
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



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



Drag the **adafruit-circuitpython-boardname-language-version.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.

<https://adafru.it/1ayi>

Code

Getting the Program's Files

To use the application, you need to obtain **code.py** with the program, and the other project files to place on the Fruit Jam **CIRCUITPY** drive.

Thankfully, this can be done in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries, the **code.py** file, and other project files 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, as well as the **egg_hunt_game_assets/** folder to your **CIRCUITPY** drive. The program should self start.

```
# SPDX-FileCopyrightText: 2026 Tim Cocks for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""
Fruit Jam Egg Hunt Maze Game
D-Pad to move, find eggs in the maze. Once you've found all of them
the end screen lets you select any that you want to add to your permanent
collection.

Assets from:
    Sprout Lands By : Cup Nooble https://cupnooble.itch.io/sprout-lands-asset-pack
    Eggs By: Onocentaur https://onocentaur.itch.io
"""
# pylint: disable=too-many-locals, global-statement, used-before-assignment, too-
many-nested-blocks
import array
import time
import json
```

```

import os
import random

import supervisor
import terminalio
from displayio import TileGrid, OnDiskBitmap, Group
from tilepalettemapper import TilePaletteMapper
import usb.core

import adafruit_imageload
from adafruit_fruitjam.peripherals import request_display_config
from adafruit_display_text.text_box import TextBox

SAVED_EGGS = []
collection_page = 0
COLLECTION_EGGS_PER_PAGE = 9 * 6 # 9 columns x 6 rows

if "found_eggs.json" in os.listdir("/saves"):
    with open("/saves/found_eggs.json", "r") as f:
        SAVED_EGGS = json.load(f)

BTN_DPAD_UPDOWN_INDEX = 1
BTN_DPAD_RIGHTLEFT_INDEX = 0
BTN_ABXY_INDEX = 5
BTN_OTHER_INDEX = 6

DIR_IN = 0x80
controller = None

TILE_SIZE = 16
TRANSPARENT_TILE = 40

GRASS_TILES = (0, 1, 2, 3, 4, 5, 6, 14, 15, 16, 17, 18, 19)
WALL_TILES = (7, 8, 9, 10, 11, 12, 20, 21, 22, 23, 24, 25, 26, 30)
TWO_WIDE_WALL_TILES = (28, 31, 33)

FOG_TILES = (50, 51, 52, 53, 54, 55)

EGG_TILES = (42, 43, 44, 45) + tuple(range(56, 56 + 28))
print(EGG_TILES)
WALKABLE_TILES = [TRANSPARENT_TILE]
WALKABLE_TILES.extend(EGG_TILES)

request_display_config(320, 240)
display = supervisor.runtime.display

score = 0

eggs_placed = 0
eggs_found = 0

def generate_maze(width, height, seed=None, start_cell=(1, 1)):
    """
    Generate a maze as a 2D list.
    1 = wall, 0 = floor.
    Width and height must be odd numbers >= 3.
    """
    if width < 3 or height < 3:
        raise ValueError("Width and height must be >= 3")
    if width % 2 == 0 or height % 2 == 0:
        raise ValueError("Width and height must be odd")

    if seed is not None:
        random.seed(seed)
    else:
        random.seed(int(time.monotonic() * 1000))

```

```

# Initialize grid full of walls
_maze = [[1 for _ in range(width)] for _ in range(height)]

# Starting cell
start_x, start_y = start_cell
_maze[start_y][start_x] = 0

# Directions: (dx, dy)
dirs = [(2, 0), (-2, 0), (0, 2), (0, -2)]

stack = [(start_x, start_y)]

while stack:
    x, y = stack[-1]
    # Find all unvisited neighbors two steps away
    neighbors = []
    for dx, dy in dirs:
        nx, ny = x + dx, y + dy
        if 1 <= nx < width - 1 and 1 <= ny < height - 1:
            if _maze[ny][nx] == 1:
                neighbors.append((nx, ny, dx // 2, dy // 2))

    if neighbors:
        nx, ny, wx, wy = random.choice(neighbors)
        _maze[ny][nx] = 0
        _maze[y + wy][x + wx] = 0
        stack.append((nx, ny))
    else:
        stack.pop()

return _maze

def get_tile_at_pixel_coords(tilegrid, x, y):
    return tilegrid[x // tilegrid.tile_width, y // tilegrid.tile_height]

def get_player_tile(player_obj):
    center_x = player_obj.x + player_obj.tile_width // 2
    center_y = player_obj.y + player_obj.tile_height // 2
    return center_x // TILE_SIZE, center_y // TILE_SIZE

def clear_fog(loc, tilegrid):
    for x_offset in (-2, -1, 0, 1, 2):
        for y_offset in (-2, -1, 0, 1, 2):
            if abs(x_offset) + abs(y_offset) <= 3:
                try:
                    tilegrid[loc[0] + x_offset, loc[1] + y_offset] =
TRANSPARENT_TILE
                except IndexError:
                    pass

def take_egg(loc, tilegrid):
    global score, eggs_found

    if tilegrid[loc[0], loc[1]] in EGG_TILES:
        eggs_found += 1
        score += 5
        tilegrid[loc[0], loc[1]] = TRANSPARENT_TILE
        print(egg_paint_dict[loc[0], loc[1]])
        print(score)
        if eggs_found == eggs_placed:
            show_endscreen()

def update_collection():
    # Clear the grid first

```

```

for y in range(6):
    for x in range(9):
        collection_tilegrid[x, y] = TRANSPARENT_TILE

start = collection_page * COLLECTION_EGGS_PER_PAGE
end = start + COLLECTION_EGGS_PER_PAGE
page_eggs = SAVED_EGGS[start:end]

cur_x = 0
cur_y = 0

for egg_data in page_eggs:
    collection_tilegrid[cur_x, cur_y] = egg_data[0]
    apply_paint(cur_x, cur_y, egg_data[1:], collectionPainter)
    cur_x += 1
    if cur_x >= 9:
        cur_x = 0
        cur_y += 1

def toggle_collection():
    global collection_page, max_page
    if main_group[-1] != collection_group:
        max_page = max(0, (len(SAVED_EGGS) - 1) // COLLECTION_EGGS_PER_PAGE)
        collection_page = max_page
        update_collection()
        main_group.append(collection_group)
    else:
        main_group.remove(collection_group)

def apply_paint(x, y, color_map, mapper):
    painting_palette = list(DEFAULT_PALETTE)
    for idx, i in enumerate(range(72, 78)):
        paint_color = color_map[idx]
        painting_palette[i] = paint_color
    mapper[x, y] = painting_palette

def show_endscreen():
    cur_x = 0
    cur_y = 0
    end_screen_cursor_tg[end_screen_cursor_loc[0], end_screen_cursor_loc[1]] = (
        ENDSCREEN_CURSOR_TILE_INDEX
    )

    for _, egg_data in egg_paint_dict.items():
        end_screen_tilegrid[cur_x, cur_y] = egg_data[0]
        end_screen_dict[cur_x, cur_y] = egg_data
        apply_paint(cur_x, cur_y, egg_data[1:], endPainter)
        cur_x += 1
        if cur_x >= 9:
            cur_x = 0
            cur_y += 1

    while cur_y < 6:
        end_screen_tilegrid[cur_x, cur_y] = TRANSPARENT_TILE
        cur_x += 1
        if cur_x >= 9:
            cur_x = 0
            cur_y += 1

    main_group.append(end_screen_group)

class PlayerEntity(TileGrid):
    DOWN_ANIMATION_SPRITES = [0, 1, 2, 3]
    UP_ANIMATION_SPRITES = [4, 5, 6, 7]
    LEFT_ANIMATION_SPRITES = [8, 9, 10, 11]

```

```

RIGHT_ANIMATION_SPRITES = [12, 13, 14, 15]

def __init__(self):
    player_spritesheet = OnDiskBitmap("egg_hunt_game_assets/
player_spritesheet.bmp")
    super().__init__(
        player_spritesheet,
        pixel_shader=player_spritesheet.pixel_shader,
        width=1,
        height=1,
        tile_width=TILE_SIZE,
        tile_height=TILE_SIZE,
        default_tile=0,
    )
    self.pixel_shader.make_transparent(0)
    self.x = 1 * TILE_SIZE
    self.y = 1 * TILE_SIZE
    self.cur_animation = self.DOWN_ANIMATION_SPRITES
    self.cur_animation_index = 0

def try_move(self, x, y, world_tilegrid):
    padding = 6
    tl_point = (self.x + x + padding, self.y + y + padding)
    tr_point = ((self.x + self.tile_width) + x - padding, self.y + y + padding)
    bl_point = (self.x + x + padding, (self.y + self.tile_height) + y - padding)
    br_point = (
        (self.x + self.tile_width) + x - padding,
        (self.y + self.tile_height) + y - padding,
    )
    # print(tl_point)
    # print(tr_point)
    # print(bl_point)
    # print(br_point)
    # print("=====")
    for point in (tl_point, tr_point, bl_point, br_point):
        tile_index = get_tile_at_pixel_coords(world_tilegrid, *point)
        if tile_index not in WALKABLE_TILES:
            return False

    player.x += x
    player.y += y

    if x > 0:
        self.cur_animation = self.RIGHT_ANIMATION_SPRITES
    elif x < 0:
        self.cur_animation = self.LEFT_ANIMATION_SPRITES

    if y > 0:
        self.cur_animation = self.DOWN_ANIMATION_SPRITES
    elif y < 0:
        self.cur_animation = self.UP_ANIMATION_SPRITES

    self.cur_animation_index = (self.cur_animation_index + 1) % len(
        self.cur_animation
    )
    self[0] = self.cur_animation[self.cur_animation_index]

    return True

def start_game():
    global eggs_placed, maze, eggs_found
    eggs_placed = 0
    eggs_found = 0

    seen_tiles.clear()
    processed_tiles.clear()

    # maze = generate_maze(WIDTH, HEIGHT, seed=42)

```

```

maze.clear()
spawn_x = random.choice(range(1, WIDTH - 1, 2))
spawn_y = random.choice(range(1, HEIGHT - 1, 2))
player.x = spawn_x * TILE_SIZE
player.y = spawn_y * TILE_SIZE
maze = generate_maze(WIDTH, HEIGHT, start_cell=(spawn_x, spawn_y))
for row in maze:
    print("".join("#" if cell else " " for cell in row))

egg_paint_dict.clear()

skip_next = False
for y in range(HEIGHT):
    for x in range(WIDTH):
        world_below_tilegrid[x, y] = random.choice(GRASS_TILES)
        fog_tilegrid[x, y] = random.choice(FOG_TILES)

        if skip_next:
            skip_next = False
            continue

        painter[x, y] = DEFAULT_PALETTE
        world_player_tilegrid[x, y] = TRANSPARENT_TILE

        if maze[y][x] == 1:
            try:
                if maze[y][x + 1] == 1:
                    if random.randint(0, 10) >= 9:
                        skip_next = True
                        choice = random.choice(TWO_WIDE_WALL_TILES)
                        world_player_tilegrid[x, y] = choice
                        world_player_tilegrid[x + 1, y] = choice + 1
                        continue
            except IndexError:
                pass
            world_player_tilegrid[x, y] = random.choice(WALL_TILES)
        elif (
            x != player.x // TILE_SIZE or y != player.y // TILE_SIZE
        ): # no wall at this location, skip player start location
            roll = random.randint(1, 100)
            if roll >= 86:
                eggs_placed += 1
                egg_choice = random.choice(EGG_TILES)
                world_player_tilegrid[x, y] = egg_choice
                egg_paint_dict[x, y] = [egg_choice]
                painting_palette = list(DEFAULT_PALETTE)
                for i in range(72, 78):
                    paint_color = random.randint(61, 72)
                    painting_palette[i] = paint_color
                egg_paint_dict[x, y].append(paint_color)
                painter[x, y] = painting_palette

WIDTH, HEIGHT = 19, 15 # must be odd

main_group = Group()
world_tilesheet, world_tilesheet_palette = adafruit_imageload.load(
    "egg_hunt_game_assets/map_spritesheet.bmp"
)
world_below_tilegrid = TileGrid(
    world_tilesheet,
    pixel_shader=world_tilesheet_palette,
    width=WIDTH,
    height=HEIGHT,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=77,
)

```

```

painter = TilePaletteMapper(world_tilesheet_palette, len(world_tilesheet_palette))
end_painter = TilePaletteMapper(world_tilesheet_palette,
len(world_tilesheet_palette))
collection_painter = TilePaletteMapper(
    world_tilesheet_palette, len(world_tilesheet_palette)
)

world_player_tilegrid = TileGrid(
    world_tilesheet,
    pixel_shader=painter,
    width=WIDTH,
    height=HEIGHT,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=TRANSPARENT_TILE,
)

DEFAULT_PALETTE = list(painter[0, 0])

fog_tilegrid = TileGrid(
    world_tilesheet,
    pixel_shader=world_tilesheet_palette,
    width=WIDTH,
    height=HEIGHT,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=TRANSPARENT_TILE,
)

end_screen_tilegrid = TileGrid(
    world_tilesheet,
    pixel_shader=end_painter,
    width=9,
    height=6,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=TRANSPARENT_TILE,
)

end_screen_tilegrid.y = (
    display.height // 2 - (end_screen_tilegrid.tile_height *
end_screen_tilegrid.height)
) - 1
end_screen_tilegrid.x = (
    display.width // 4
    - (end_screen_tilegrid.tile_width * end_screen_tilegrid.width) // 2
)

end_screen_cursor_tg = TileGrid(
    world_tilesheet,
    pixel_shader=world_tilesheet_palette,
    width=9,
    height=6,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=50,
)
end_screen_cursor_tg.y = (
    display.height // 2
    - (end_screen_cursor_tg.tile_height * end_screen_cursor_tg.height)
) - 1
end_screen_cursor_tg.x = (
    display.width // 4
    - (end_screen_cursor_tg.tile_width * end_screen_cursor_tg.width) // 2
)
end_screen_bg = TileGrid(
    world_tilesheet,
    pixel_shader=world_tilesheet_palette,
    width=10,

```

```

    height=8,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=50,
)

end_screen_text = TextBox(
    terminalio.FONT,
    width=display.width // 2,
    height=30,
    align=TextBox.ALIGN_CENTER,
    text="Level Complete\nDPad+A:Save | Start:Next",
    color=0x000000,
    line_spacing=0.85,
)
end_screen_text.anchor_point = (0, 0)
end_screen_text.anchored_position = (0, 0)

end_screen_dict = {}

end_screen_group = Group(scale=2)

end_screen_group.append(end_screen_bg)
end_screen_group.append(end_screen_cursor_tg)
end_screen_group.append(end_screen_tilegrid)
end_screen_group.append(end_screen_text)

end_screen_cursor_loc = [0, 0]

ENDSCREEN_CURSOR_TILE_INDEX = 48

collection_tilegrid = TileGrid(
    world_tilesheet,
    pixel_shader=collectionPainter,
    width=9,
    height=6,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=TRANSPARENT_TILE,
)
collection_tilegrid.y = (
    display.height // 2 - (collection_tilegrid.tile_height *
collection_tilegrid.height)
) - 1
collection_tilegrid.x = (
    display.width // 4
    - (collection_tilegrid.tile_width * collection_tilegrid.width) // 2
)
print(collection_tilegrid.y)

collection_bg = TileGrid(
    world_tilesheet,
    pixel_shader=world_tilesheet_palette,
    width=10,
    height=8,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=50,
)

collection_text = TextBox(
    terminalio.FONT,
    width=display.width // 2,
    height=30,
    align=TextBox.ALIGN_CENTER,
    text="Your Collection\nL/R: page | Y: close",
    color=0x000000,
    line_spacing=0.85,
)

```

```

collection_text.anchor_point = (0, 0)
collection_text.anchored_position = (0, 0)

collection_group = Group(scale=2)
collection_group.append(collection_bg)
collection_group.append(collection_tilegrid)
collection_group.append(collection_text)

player = PlayerEntity()

egg_paint_dict = {}
maze = []

seen_tiles = set()
processed_tiles = set()

start_game()

world_tilesheet_palette.make_transparent(0)
main_group.append(world_below_tilegrid)
main_group.append(world_player_tilegrid)
main_group.append(player)
main_group.append(fog_tilegrid)

display.root_group = main_group
# display.auto_refresh = False
# display.refresh()

# get the first device found
device = None
while device is None:
    for d in usb.core.find(find_all=True):
        device = d
        break
    time.sleep(0.1)

# set configuration so we can read data from it
device.set_configuration()
print(
    f"configuration set for {device.manufacturer}, {device.product},
    {device.serial_number}"
)

# Test to see if the kernel is using the device and detach it.
if device.is_kernel_driver_active(0):
    device.detach_kernel_driver(0)

# buffer to hold 64 bytes
buf = array.array("B", [0] * 64)
prev_buf = array.array("B", [0] * 64)

clear_fog(get_player_tile(player), fog_tilegrid)

while True:
    try:
        count = device.read(0x81, buf, timeout=100)
        # print(f"read size: {count}")
    except usb.core.USBTimeoutError:
        time.sleep(0.01)
        print("usb timeout")
        continue

    moved = False
    if buf[BTN_DPAD_UPDOWN_INDEX] == 0x0:
        # print("D-Pad UP pressed")
        if main_group[-1] == fog_tilegrid:
            moved = player.try_move(0, -4, world_player_tilegrid)
        elif (
            main_group[-1] == end_screen_group

```

```

        and prev_buf[BTN_DPAD_UPDOWN_INDEX] != 0x0
    ):
        end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
            50
        )
        end_screen_cursor_loc[1] = max(end_screen_cursor_loc[1] - 1, 0)
        end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
            ENDSCREEN_CURSOR_TILE_INDEX
        )
    elif buf[BTN_DPAD_UPDOWN_INDEX] == 0xFF:
        # print("D-Pad DOWN pressed")
        if main_group[-1] == fog_tilegrid:
            moved = player.try_move(0, 4, world_player_tilegrid)
        elif (
            main_group[-1] == end_screen_group
            and prev_buf[BTN_DPAD_UPDOWN_INDEX] != 0xFF
        ):
            end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                50
            )
            end_screen_cursor_loc[1] = min(end_screen_cursor_loc[1] + 1, 5)
            end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                ENDSCREEN_CURSOR_TILE_INDEX
            )
        if buf[BTN_DPAD_RIGHTLEFT_INDEX] == 0:
            if main_group[-1] == fog_tilegrid:
                moved = player.try_move(-4, 0, world_player_tilegrid)
            elif (
                main_group[-1] == end_screen_group
                and prev_buf[BTN_DPAD_RIGHTLEFT_INDEX] != 0
            ):
                end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                    50
                )
                end_screen_cursor_loc[0] = max(end_screen_cursor_loc[0] - 1, 0)
                end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                    ENDSCREEN_CURSOR_TILE_INDEX
                )
            # print("D-Pad LEFT pressed")
        elif buf[BTN_DPAD_RIGHTLEFT_INDEX] == 0xFF:
            if main_group[-1] == fog_tilegrid:
                moved = player.try_move(4, 0, world_player_tilegrid)
            elif (
                main_group[-1] == end_screen_group
                and prev_buf[BTN_DPAD_RIGHTLEFT_INDEX] != 0xFF
            ):
                end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                    50
                )
                end_screen_cursor_loc[0] = min(end_screen_cursor_loc[0] + 1, 8)
                end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                    ENDSCREEN_CURSOR_TILE_INDEX
                )
            # print("D-Pad RIGHT pressed")

        if prev_buf[BTN_ABXY_INDEX] == 0xF and buf[BTN_ABXY_INDEX] == 0x2F:
            print("A press")
            if main_group[-1] == end_screen_group:
                SAVED_EGGS.append(
                    end_screen_dict[end_screen_cursor_loc[0], end_screen_cursor_loc[1]]
                )

```

```

        print(end_screen_dict[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]])
        with open("/saves/found_eggs.json", "w") as f:
            json.dump(SAVED_EGGS, f)
    elif prev_buf[BTN_ABXY_INDEX] == 0xF and buf[BTN_ABXY_INDEX] == 0x8F:
        print("Y press")
        toggle_collection()
    elif prev_buf[BTN_ABXY_INDEX] == 0xF and buf[BTN_ABXY_INDEX] != 0xF:
        print(hex(buf[BTN_ABXY_INDEX]))

    if prev_buf[BTN_OTHER_INDEX] == 0x0 and buf[BTN_OTHER_INDEX] == 0x20:
        print("Start press")
        if main_group[-1] == end_screen_group:
            main_group.remove(end_screen_group)
            start_game()
            clear_fog(get_player_tile(player), fog_tilegrid)

    # print(hex(buf[BTN_OTHER_INDEX]))
    if prev_buf[BTN_OTHER_INDEX] == 0x0 and buf[BTN_OTHER_INDEX] == 0x2:
        print("Right shoulder button")
        if main_group[-1] == collection_group:
            max_page = max(0, (len(SAVED_EGGS) - 1) // COLLECTION_EGGS_PER_PAGE)
            if collection_page < max_page:
                collection_page += 1
                update_collection()

    if prev_buf[BTN_OTHER_INDEX] == 0x0 and buf[BTN_OTHER_INDEX] == 0x1:
        print("Left shoulder button")
        if main_group[-1] == collection_group:
            if collection_page > 0:
                collection_page -= 1
                update_collection()

    _cur_player_loc = get_player_tile(player)
    if _cur_player_loc not in processed_tiles and _cur_player_loc not in seen_tiles:
        seen_tiles.add(_cur_player_loc)

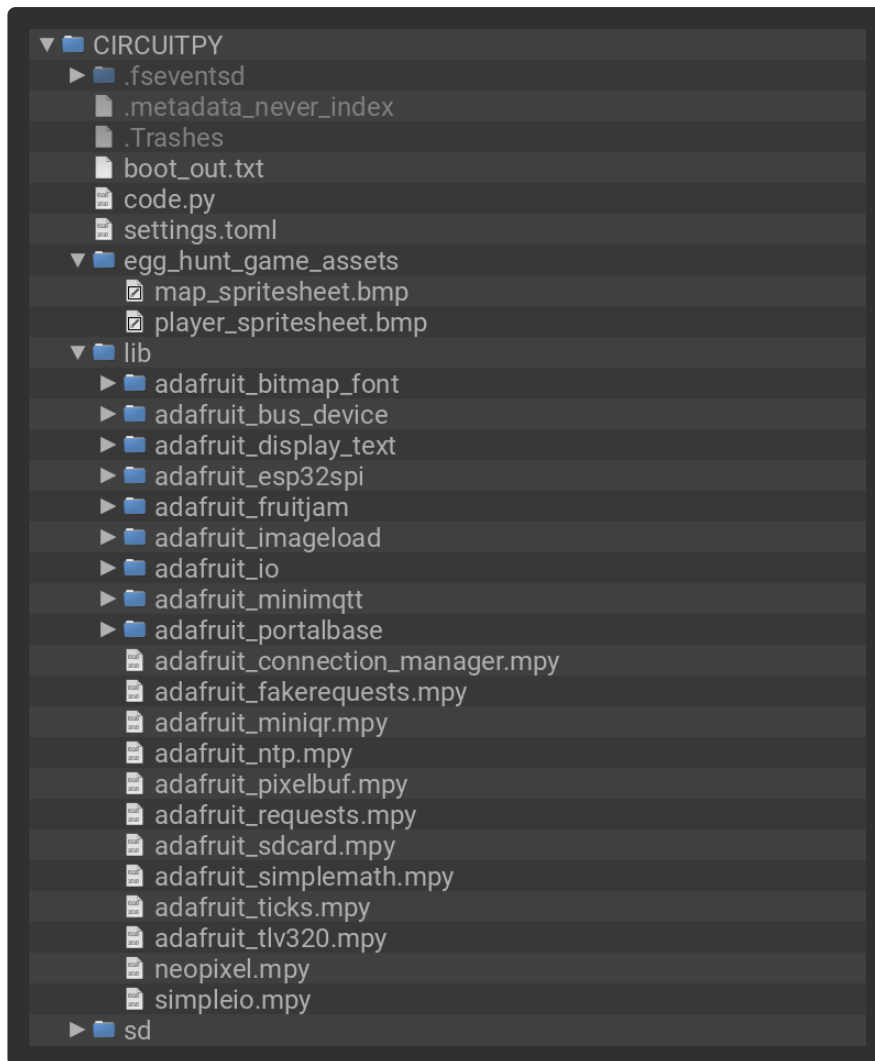
    for _loc in seen_tiles:
        clear_fog(_loc, fog_tilegrid)
        take_egg(_loc, world_player_tilegrid)
        processed_tiles.add(_loc)
    seen_tiles.clear()

    prev_buf[:] = buf

```

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.



Play Game

The game has 3 main screens: the maze world, the end level screen, and the egg collection screen.



Maze World

This is where the game first starts. On this screen the player can run around inside of the maze and find colorful eggs.

Controls:

D-Pad: make the bunny character run around.

Y Button: Show the Egg Collection screen.



End Level Screen

This screen is shown after you have found all of the eggs hidden inside of the current maze. Here the user can select any of the eggs they found during this round to add to the permanent collection.

Controls:

D-Pad: Move the selection cursor.

A Button: Save the currently selected egg to your collection.

Y Button: Show the Egg Collection screen.

Start Button: Generate a new maze and new eggs, start the next round.



Egg Collection Screen

On this screen you can view all of the eggs that are saved to your collection. The egg pattern index and colors are saved into a JSON file in the CPSAVES portion of flash storage.

Controls:

Y Button: Close the collection screen.

Left Shoulder: Go to the previous page.

Right Shoulder: Go to the next page.

Code Walkthrough Video

An overview of the code for this project was featured during the Deep Dive w/ Tim live stream on 3/31/26. Watch the video embedded below to get an idea of what parts of the code control which aspects of the game.

Code

The project code is embedded below if you want to follow along.

```
# SPDX-FileCopyrightText: 2026 Tim Cocks for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""
Fruit Jam Egg Hunt Maze Game
D-Pad to move, find eggs in the maze. Once you've found all of them
the end screen lets you select any that you want to add to your permanent
collection.

Assets from:
    Sprout Lands By : Cup Nooble https://cupnooble.itch.io/sprout-lands-asset-pack
    Eggs By: Onocentaur https://onocentaur.itch.io
"""
# pylint: disable=too-many-locals, global-statement, used-before-assignment, too-
many-nested-blocks
import array
import time
import json
import os
import random

import supervisor
import terminalio
from displayio import TileGrid, OnDiskBitmap, Group
from tilepalettermapper import TilePaletteMapper
```

```

import usb.core

import adafruit_imageload
from adafruit_fruitjam.peripherals import request_display_config
from adafruit_display_text.text_box import TextBox

SAVED_EGGS = []
collection_page = 0
COLLECTION_EGGS_PER_PAGE = 9 * 6 # 9 columns x 6 rows

if "found_eggs.json" in os.listdir("/saves"):
    with open("/saves/found_eggs.json", "r") as f:
        SAVED_EGGS = json.load(f)

BTN_DPAD_UPDOWN_INDEX = 1
BTN_DPAD_RIGHTLEFT_INDEX = 0
BTN_ABXY_INDEX = 5
BTN_OTHER_INDEX = 6

DIR_IN = 0x80
controller = None

TILE_SIZE = 16
TRANSPARENT_TILE = 40

GRASS_TILES = (0, 1, 2, 3, 4, 5, 6, 14, 15, 16, 17, 18, 19)
WALL_TILES = (7, 8, 9, 10, 11, 12, 20, 21, 22, 23, 24, 25, 26, 30)
TWO_WIDE_WALL_TILES = (28, 31, 33)

FOG_TILES = (50, 51, 52, 53, 54, 55)

EGG_TILES = (42, 43, 44, 45) + tuple(range(56, 56 + 28))
print(EGG_TILES)
WALKABLE_TILES = [TRANSPARENT_TILE]
WALKABLE_TILES.extend(EGG_TILES)

request_display_config(320, 240)
display = supervisor.runtime.display

score = 0

eggs_placed = 0
eggs_found = 0

def generate_maze(width, height, seed=None, start_cell=(1, 1)):
    """
    Generate a maze as a 2D list.
    1 = wall, 0 = floor.
    Width and height must be odd numbers >= 3.
    """
    if width < 3 or height < 3:
        raise ValueError("Width and height must be >= 3")
    if width % 2 == 0 or height % 2 == 0:
        raise ValueError("Width and height must be odd")

    if seed is not None:
        random.seed(seed)
    else:
        random.seed(int(time.monotonic() * 1000))

    # Initialize grid full of walls
    _maze = [[1 for _ in range(width)] for _ in range(height)]

    # Starting cell
    start_x, start_y = start_cell
    _maze[start_y][start_x] = 0

```

```

# Directions: (dx, dy)
dirs = [(2, 0), (-2, 0), (0, 2), (0, -2)]

stack = [(start_x, start_y)]

while stack:
    x, y = stack[-1]
    # Find all unvisited neighbors two steps away
    neighbors = []
    for dx, dy in dirs:
        nx, ny = x + dx, y + dy
        if 1 <= nx < width - 1 and 1 <= ny < height - 1:
            if _maze[ny][nx] == 1:
                neighbors.append((nx, ny, dx // 2, dy // 2))

    if neighbors:
        nx, ny, wx, wy = random.choice(neighbors)
        _maze[ny][nx] = 0
        _maze[y + wy][x + wx] = 0
        stack.append((nx, ny))
    else:
        stack.pop()

return _maze

def get_tile_at_pixel_coords(tilegrid, x, y):
    return tilegrid[x // tilegrid.tile_width, y // tilegrid.tile_height]

def get_player_tile(player_obj):
    center_x = player_obj.x + player_obj.tile_width // 2
    center_y = player_obj.y + player_obj.tile_height // 2
    return center_x // TILE_SIZE, center_y // TILE_SIZE

def clear_fog(loc, tilegrid):
    for x_offset in (-2, -1, 0, 1, 2):
        for y_offset in (-2, -1, 0, 1, 2):
            if abs(x_offset) + abs(y_offset) <= 3:
                try:
                    tilegrid[loc[0] + x_offset, loc[1] + y_offset] =
TRANSPARENT_TILE
                except IndexError:
                    pass

def take_egg(loc, tilegrid):
    global score, eggs_found

    if tilegrid[loc[0], loc[1]] in EGG_TILES:
        eggs_found += 1
        score += 5
        tilegrid[loc[0], loc[1]] = TRANSPARENT_TILE
        print(egg_paint_dict[loc[0], loc[1]])
        print(score)
        if eggs_found == eggs_placed:
            show_endscreen()

def update_collection():
    # Clear the grid first
    for y in range(6):
        for x in range(9):
            collection_tilegrid[x, y] = TRANSPARENT_TILE

start = collection_page * COLLECTION_EGGS_PER_PAGE
end = start + COLLECTION_EGGS_PER_PAGE
page_eggs = SAVED_EGGS[start:end]

```

```

cur_x = 0
cur_y = 0

for egg_data in page_eggs:
    collection_tilegrid[cur_x, cur_y] = egg_data[0]
    apply_paint(cur_x, cur_y, egg_data[1:], collectionPainter)
    cur_x += 1
    if cur_x >= 9:
        cur_x = 0
        cur_y += 1

def toggle_collection():
    global collection_page, max_page
    if main_group[-1] != collection_group:
        max_page = max(0, (len(SAVED_EGGS) - 1) // COLLECTION_EGGS_PER_PAGE)
        collection_page = max_page
        update_collection()
        main_group.append(collection_group)
    else:
        main_group.remove(collection_group)

def apply_paint(x, y, color_map, mapper):
    painting_palette = list(DEFAULT_PALETTE)
    for idx, i in enumerate(range(72, 78)):
        paint_color = color_map[idx]
        painting_palette[i] = paint_color
    mapper[x, y] = painting_palette

def show_endscreen():
    cur_x = 0
    cur_y = 0
    end_screen_cursor_tg[end_screen_cursor_loc[0], end_screen_cursor_loc[1]] = (
        ENDSCREEN_CURSOR_TILE_INDEX
    )

    for _, egg_data in egg_paint_dict.items():
        end_screen_tilegrid[cur_x, cur_y] = egg_data[0]
        end_screen_dict[cur_x, cur_y] = egg_data
        apply_paint(cur_x, cur_y, egg_data[1:], endPainter)
        cur_x += 1
        if cur_x >= 9:
            cur_x = 0
            cur_y += 1

    while cur_y < 6:
        end_screen_tilegrid[cur_x, cur_y] = TRANSPARENT_TILE
        cur_x += 1
        if cur_x >= 9:
            cur_x = 0
            cur_y += 1

    main_group.append(end_screen_group)

class PlayerEntity(TileGrid):
    DOWN_ANIMATION_SPRITES = [0, 1, 2, 3]
    UP_ANIMATION_SPRITES = [4, 5, 6, 7]
    LEFT_ANIMATION_SPRITES = [8, 9, 10, 11]
    RIGHT_ANIMATION_SPRITES = [12, 13, 14, 15]

    def __init__(self):
        player_spritesheet = OnDiskBitmap("egg_hunt_game_assets/
player_spritesheet.bmp")
        super().__init__(
            player_spritesheet,

```

```

        pixel_shader=player_spritesheet.pixel_shader,
        width=1,
        height=1,
        tile_width=TILE_SIZE,
        tile_height=TILE_SIZE,
        default_tile=0,
    )
    self.pixel_shader.make_transparent(0)
    self.x = 1 * TILE_SIZE
    self.y = 1 * TILE_SIZE
    self.cur_animation = self.DOWN_ANIMATION_SPRITES
    self.cur_animation_index = 0

def try_move(self, x, y, world_tilegrid):
    padding = 6
    tl_point = (self.x + x + padding, self.y + y + padding)
    tr_point = ((self.x + self.tile_width) + x - padding, self.y + y + padding)
    bl_point = (self.x + x + padding, (self.y + self.tile_height) + y - padding)
    br_point = (
        (self.x + self.tile_width) + x - padding,
        (self.y + self.tile_height) + y - padding,
    )
    # print(tl_point)
    # print(tr_point)
    # print(bl_point)
    # print(br_point)
    # print("=====")
    for point in (tl_point, tr_point, bl_point, br_point):
        tile_index = get_tile_at_pixel_coords(world_tilegrid, *point)
        if tile_index not in WALKABLE_TILES:
            return False

    player.x += x
    player.y += y

    if x > 0:
        self.cur_animation = self.RIGHT_ANIMATION_SPRITES
    elif x < 0:
        self.cur_animation = self.LEFT_ANIMATION_SPRITES

    if y > 0:
        self.cur_animation = self.DOWN_ANIMATION_SPRITES
    elif y < 0:
        self.cur_animation = self.UP_ANIMATION_SPRITES

    self.cur_animation_index = (self.cur_animation_index + 1) % len(
        self.cur_animation
    )
    self[0] = self.cur_animation[self.cur_animation_index]

    return True

def start_game():
    global eggs_placed, maze, eggs_found
    eggs_placed = 0
    eggs_found = 0

    seen_tiles.clear()
    processed_tiles.clear()

    # maze = generate_maze(WIDTH, HEIGHT, seed=42)
    maze.clear()
    spawn_x = random.choice(range(1, WIDTH - 1, 2))
    spawn_y = random.choice(range(1, HEIGHT - 1, 2))
    player.x = spawn_x * TILE_SIZE
    player.y = spawn_y * TILE_SIZE
    maze = generate_maze(WIDTH, HEIGHT, start_cell=(spawn_x, spawn_y))
    for row in maze:

```

```

        print("".join("#" if cell else " " for cell in row))

    egg_paint_dict.clear()

    skip_next = False
    for y in range(HEIGHT):
        for x in range(WIDTH):
            world_below_tilegrid[x, y] = random.choice(GRASS_TILES)
            fog_tilegrid[x, y] = random.choice(FOG_TILES)

            if skip_next:
                skip_next = False
                continue

            painter[x, y] = DEFAULT_PALETTE
            world_player_tilegrid[x, y] = TRANSPARENT_TILE

            if maze[y][x] == 1:
                try:
                    if maze[y][x + 1] == 1:
                        if random.randint(0, 10) >= 9:
                            skip_next = True
                            choice = random.choice(TWO_WIDE_WALL_TILES)
                            world_player_tilegrid[x, y] = choice
                            world_player_tilegrid[x + 1, y] = choice + 1
                            continue
                        except IndexError:
                            pass
                    world_player_tilegrid[x, y] = random.choice(WALL_TILES)
                elif (
                    x != player.x // TILE_SIZE or y != player.y // TILE_SIZE
                ): # no wall at this location, skip player start location
                    roll = random.randint(1, 100)
                    if roll >= 86:
                        eggs_placed += 1
                        egg_choice = random.choice(EGG_TILES)
                        world_player_tilegrid[x, y] = egg_choice
                        egg_paint_dict[x, y] = [egg_choice]
                        painting_palette = list(DEFAULT_PALETTE)
                        for i in range(72, 78):
                            paint_color = random.randint(61, 72)
                            painting_palette[i] = paint_color
                            egg_paint_dict[x, y].append(paint_color)
                        painter[x, y] = painting_palette

WIDTH, HEIGHT = 19, 15 # must be odd

main_group = Group()
world_tilesheet, world_tilesheet_palette = adafruit_imageload.load(
    "egg_hunt_game_assets/map_spritesheet.bmp"
)
world_below_tilegrid = TileGrid(
    world_tilesheet,
    pixel_shader=world_tilesheet_palette,
    width=WIDTH,
    height=HEIGHT,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=77,
)

painter = TilePaletteMapper(world_tilesheet_palette, len(world_tilesheet_palette))
end_painter = TilePaletteMapper(world_tilesheet_palette,
    len(world_tilesheet_palette))
collection_painter = TilePaletteMapper(
    world_tilesheet_palette, len(world_tilesheet_palette)
)

```

```

world_player_tilegrid = TileGrid(
    world_tilesheet,
    pixel_shader=painter,
    width=WIDTH,
    height=HEIGHT,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=TRANSPARENT_TILE,
)

DEFAULT_PALETTE = list(painter[0, 0])

fog_tilegrid = TileGrid(
    world_tilesheet,
    pixel_shader=world_tilesheet_palette,
    width=WIDTH,
    height=HEIGHT,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=TRANSPARENT_TILE,
)

end_screen_tilegrid = TileGrid(
    world_tilesheet,
    pixel_shader=end_painter,
    width=9,
    height=6,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=TRANSPARENT_TILE,
)

end_screen_tilegrid.y = (
    display.height // 2 - (end_screen_tilegrid.tile_height *
end_screen_tilegrid.height)
) - 1
end_screen_tilegrid.x = (
    display.width // 4
    - (end_screen_tilegrid.tile_width * end_screen_tilegrid.width) // 2
)

end_screen_cursor_tg = TileGrid(
    world_tilesheet,
    pixel_shader=world_tilesheet_palette,
    width=9,
    height=6,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=50,
)
end_screen_cursor_tg.y = (
    display.height // 2
    - (end_screen_cursor_tg.tile_height * end_screen_cursor_tg.height)
) - 1
end_screen_cursor_tg.x = (
    display.width // 4
    - (end_screen_cursor_tg.tile_width * end_screen_cursor_tg.width) // 2
)
end_screen_bg = TileGrid(
    world_tilesheet,
    pixel_shader=world_tilesheet_palette,
    width=10,
    height=8,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=50,
)

end_screen_text = TextBox(

```

```

    terminalio.FONT,
    width=display.width // 2,
    height=30,
    align=TextBox.ALIGN_CENTER,
    text="Level Complete\nDPad+A:Save | Start:Next",
    color=0x000000,
    line_spacing=0.85,
)
end_screen_text.anchor_point = (0, 0)
end_screen_text.anchored_position = (0, 0)

end_screen_dict = {}

end_screen_group = Group(scale=2)

end_screen_group.append(end_screen_bg)
end_screen_group.append(end_screen_cursor_tg)
end_screen_group.append(end_screen_tilegrid)
end_screen_group.append(end_screen_text)

end_screen_cursor_loc = [0, 0]

ENDSCREEN_CURSOR_TILE_INDEX = 48

collection_tilegrid = TileGrid(
    world_tilesheet,
    pixel_shader=collectionPainter,
    width=9,
    height=6,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=TRANSPARENT_TILE,
)
collection_tilegrid.y = (
    display.height // 2 - (collection_tilegrid.tile_height *
collection_tilegrid.height)
) - 1
collection_tilegrid.x = (
    display.width // 4
    - (collection_tilegrid.tile_width * collection_tilegrid.width) // 2
)
print(collection_tilegrid.y)

collection_bg = TileGrid(
    world_tilesheet,
    pixel_shader=world_tilesheet_palette,
    width=10,
    height=8,
    tile_width=TILE_SIZE,
    tile_height=TILE_SIZE,
    default_tile=50,
)

collection_text = TextBox(
    terminalio.FONT,
    width=display.width // 2,
    height=30,
    align=TextBox.ALIGN_CENTER,
    text="Your Collection\nL/R: page | Y: close",
    color=0x000000,
    line_spacing=0.85,
)
collection_text.anchor_point = (0, 0)
collection_text.anchored_position = (0, 0)

collection_group = Group(scale=2)
collection_group.append(collection_bg)
collection_group.append(collection_tilegrid)
collection_group.append(collection_text)

```

```

player = PlayerEntity()

egg_paint_dict = {}
maze = []

seen_tiles = set()
processed_tiles = set()

start_game()

world_tilesheet_palette.make_transparent(0)
main_group.append(world_below_tilegrid)
main_group.append(world_player_tilegrid)
main_group.append(player)
main_group.append(fog_tilegrid)

display.root_group = main_group
# display.auto_refresh = False
# display.refresh()

# get the first device found
device = None
while device is None:
    for d in usb.core.find(find_all=True):
        device = d
        break
    time.sleep(0.1)

# set configuration so we can read data from it
device.set_configuration()
print(
    f"configuration set for {device.manufacturer}, {device.product},
    {device.serial_number}"
)

# Test to see if the kernel is using the device and detach it.
if device.is_kernel_driver_active(0):
    device.detach_kernel_driver(0)

# buffer to hold 64 bytes
buf = array.array("B", [0] * 64)
prev_buf = array.array("B", [0] * 64)

clear_fog(get_player_tile(player), fog_tilegrid)

while True:
    try:
        count = device.read(0x81, buf, timeout=100)
        # print(f"read size: {count}")
    except usb.core.USBTimeoutError:
        time.sleep(0.01)
        print("usb timeout")
        continue

    moved = False
    if buf[BTN_DPAD_UPDOWN_INDEX] == 0x0:
        # print("D-Pad UP pressed")
        if main_group[-1] == fog_tilegrid:
            moved = player.try_move(0, -4, world_player_tilegrid)
        elif (
            main_group[-1] == end_screen_group
            and prev_buf[BTN_DPAD_UPDOWN_INDEX] != 0x0
        ):
            end_screen_cursor_tg[end_screen_cursor_loc[0],
            end_screen_cursor_loc[1]] = (
                50
            )
            end_screen_cursor_loc[1] = max(end_screen_cursor_loc[1] - 1, 0)

```

```

        end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
            ENDSCREEN_CURSOR_TILE_INDEX
        )
    elif buf[BTN_DPAD_UPDOWN_INDEX] == 0xFF:
        # print("D-Pad DOWN pressed")
        if main_group[-1] == fog_tilegrid:
            moved = player.try_move(0, 4, world_player_tilegrid)
        elif (
            main_group[-1] == end_screen_group
            and prev_buf[BTN_DPAD_UPDOWN_INDEX] != 0xFF
        ):
            end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                50
            )
            end_screen_cursor_loc[1] = min(end_screen_cursor_loc[1] + 1, 5)
            end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                ENDSCREEN_CURSOR_TILE_INDEX
            )
        if buf[BTN_DPAD_RIGHTLEFT_INDEX] == 0:
            if main_group[-1] == fog_tilegrid:
                moved = player.try_move(-4, 0, world_player_tilegrid)
            elif (
                main_group[-1] == end_screen_group
                and prev_buf[BTN_DPAD_RIGHTLEFT_INDEX] != 0
            ):
                end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                    50
                )
                end_screen_cursor_loc[0] = max(end_screen_cursor_loc[0] - 1, 0)
                end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                    ENDSCREEN_CURSOR_TILE_INDEX
                )
            # print("D-Pad LEFT pressed")
        elif buf[BTN_DPAD_RIGHTLEFT_INDEX] == 0xFF:
            if main_group[-1] == fog_tilegrid:
                moved = player.try_move(4, 0, world_player_tilegrid)
            elif (
                main_group[-1] == end_screen_group
                and prev_buf[BTN_DPAD_RIGHTLEFT_INDEX] != 0xFF
            ):
                end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                    50
                )
                end_screen_cursor_loc[0] = min(end_screen_cursor_loc[0] + 1, 8)
                end_screen_cursor_tg[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]] = (
                    ENDSCREEN_CURSOR_TILE_INDEX
                )
            # print("D-Pad RIGHT pressed")

        if prev_buf[BTN_ABXY_INDEX] == 0xF and buf[BTN_ABXY_INDEX] == 0x2F:
            print("A press")
            if main_group[-1] == end_screen_group:
                SAVED_EGGS.append(
                    end_screen_dict[end_screen_cursor_loc[0], end_screen_cursor_loc[1]]
                )
            print(end_screen_dict[end_screen_cursor_loc[0],
end_screen_cursor_loc[1]])
            with open("/saves/found_eggs.json", "w") as f:
                json.dump(SAVED_EGGS, f)
        elif prev_buf[BTN_ABXY_INDEX] == 0xF and buf[BTN_ABXY_INDEX] == 0x8F:
            print("Y press")
            toggle_collection()

```

```

elif prev_buf[BTN_ABXY_INDEX] == 0xF and buf[BTN_ABXY_INDEX] != 0xF:
    print(hex(buf[BTN_ABXY_INDEX]))

if prev_buf[BTN_OTHER_INDEX] == 0x0 and buf[BTN_OTHER_INDEX] == 0x20:
    print("Start press")
    if main_group[-1] == end_screen_group:
        main_group.remove(end_screen_group)
        start_game()
        clear_fog(get_player_tile(player), fog_tilegrid)

# print(hex(buf[BTN_OTHER_INDEX]))
if prev_buf[BTN_OTHER_INDEX] == 0x0 and buf[BTN_OTHER_INDEX] == 0x2:
    print("Right shoulder button")
    if main_group[-1] == collection_group:
        max_page = max(0, (len(SAVED_EGGS) - 1) // COLLECTION_EGGS_PER_PAGE)
        if collection_page < max_page:
            collection_page += 1
            update_collection()

if prev_buf[BTN_OTHER_INDEX] == 0x0 and buf[BTN_OTHER_INDEX] == 0x1:
    print("Left shoulder button")
    if main_group[-1] == collection_group:
        if collection_page > 0:
            collection_page -= 1
            update_collection()

_cur_player_loc = get_player_tile(player)
if _cur_player_loc not in processed_tiles and _cur_player_loc not in seen_tiles:
    seen_tiles.add(_cur_player_loc)

for _loc in seen_tiles:
    clear_fog(_loc, fog_tilegrid)
    take_egg(_loc, world_player_tilegrid)
    processed_tiles.add(_loc)
seen_tiles.clear()

prev_buf[:] = buf

```