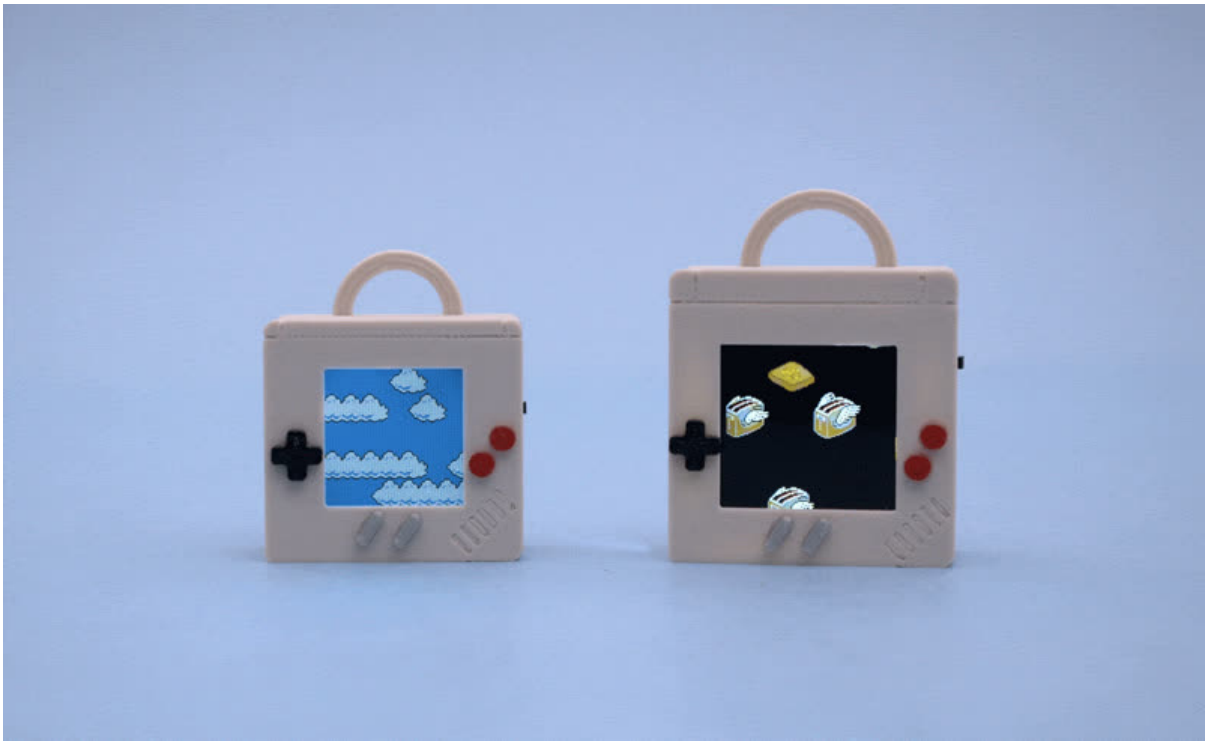
CircuitPython Animated Sprite Pendants

Created by Ruiz Brothers

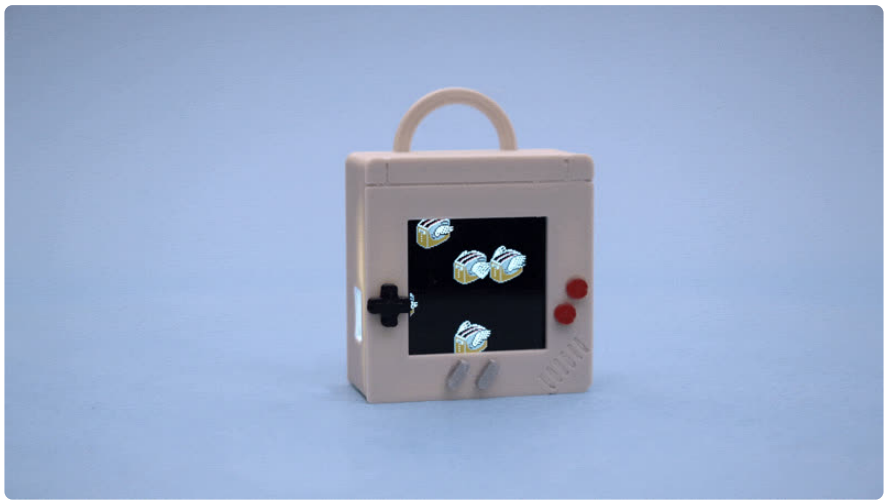


https://learn.adafruit.com/circuitpython-sprite-animation-pendant-mario-clouds-flying-toasters

Last updated on 2024-06-03 02:57:20 PM EDT

Table of Contents

Overview



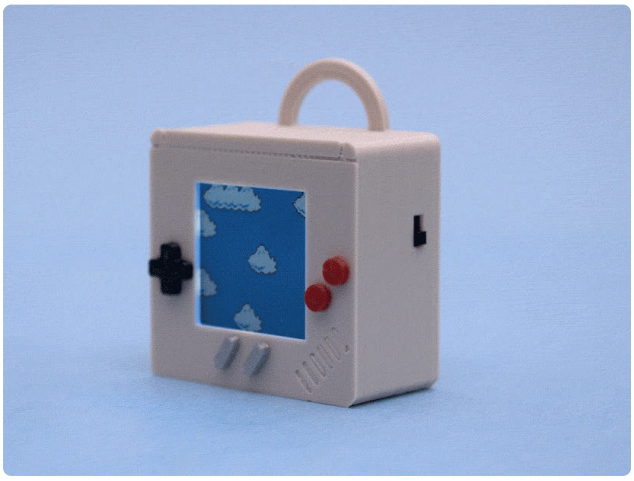
Mario Clouds & Flying Toasters

In this project we're making retro inspired wearables!

The pendant features an IPS display with animated graphics. It uses Adafruit's CircuitPython to create animated sprites that look like flying toasters and scrolling clouds.

We originally did these two as Pro Trinket projects, with a monochrome OLED for the toasters (https://adafru.it/tge), and a color OLED for the clouds (https://adafru.it/tgd). To keep up with modern times, and the new high visibility IPS displays, we've upgraded both projects here to use CircuitPython instead of Arduino for easier customization. The displays also look a lot better, but the price and construction is about the same!

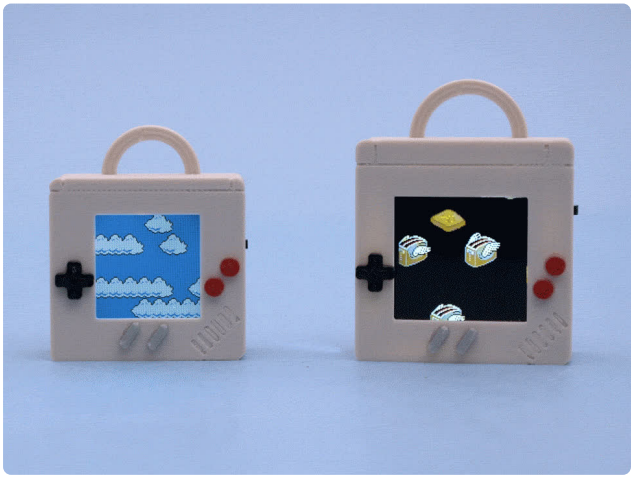
3D Printed Retro Pendant



With CircuitPython, you use the DisplayIO library to generate custom graphics and interfaces.

You can use bitmap images to create sprite sheets and palettes to generate colored pixels.

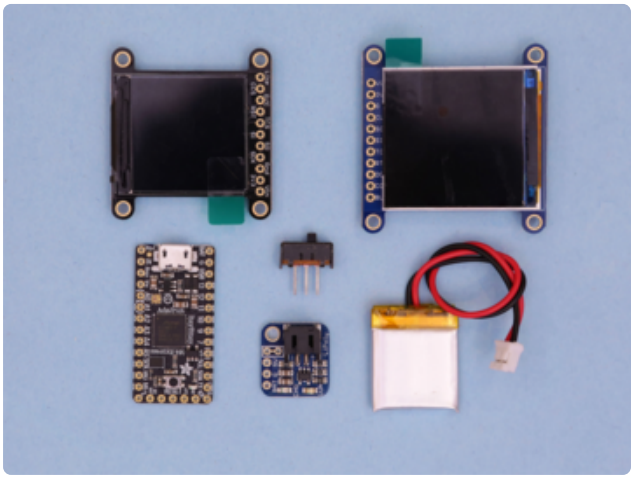
The code in this project randomly generates flying toasters that continuously scroll across the screen. We think it's a great example for folks getting started with DisplayIO and CircuitPython.



We made two versions, one for each display, so you can use the 1.3in or 1.54in IPS display.

It's also easy to swap out the graphics and modify the code to make your own animations.

Parts



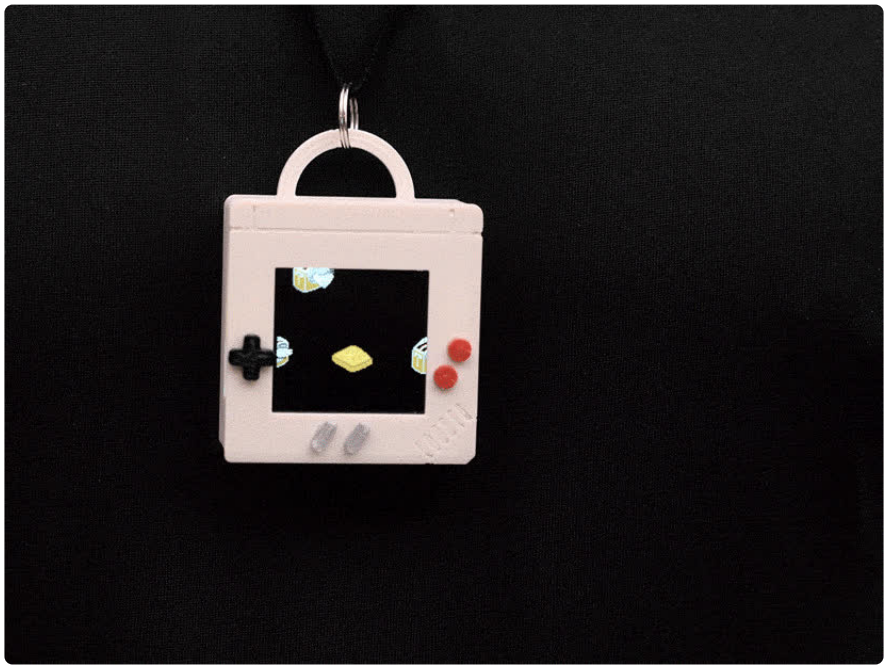
Select the display size you would like:

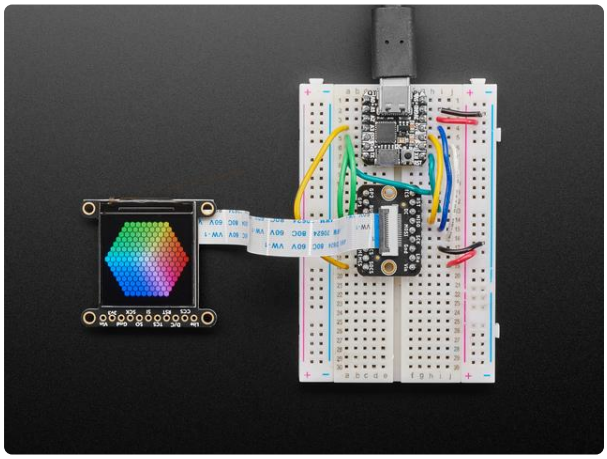
Adafruit 1.3" 240x240 Wide Angle TFT LCD Display with MicroSD (http://adafru.it/4313)
or

Adafruit 1.54" 240x240 Wide Angle TFT LCD Display with MicroSD (http://adafru.it/3787)

Then add these:

- Adafruit ItsyBitsy M4 Express featuring ATSAMD51 (http://adafru.it/3800)
- Adafruit LiIon/LiPoly Backpack Add-On for Pro Trinket/ItsyBitsy (http://adafru.it/2124)
- Breadboard-friendly SPDT Slide Switch (http://adafru.it/805)
- Lithium Ion Polymer Battery - 3.7v 150mAh (http://adafru.it/1317)

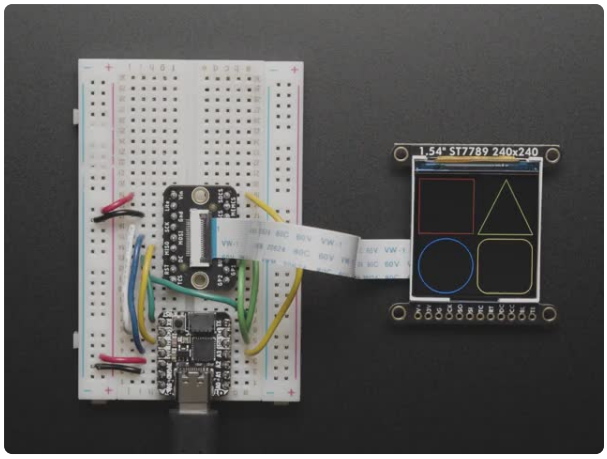




Adafruit 1.3" 240x240 Wide Angle TFT LCD Display with MicroSD

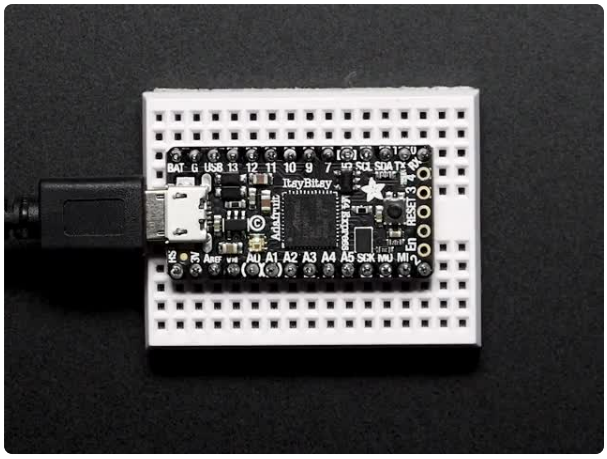
We've been looking for a display like this for a long time - it's so small only 1.3" diagonal but has a high density 260 ppi, 240x240 pixel display with...

https://www.adafruit.com/product/4313



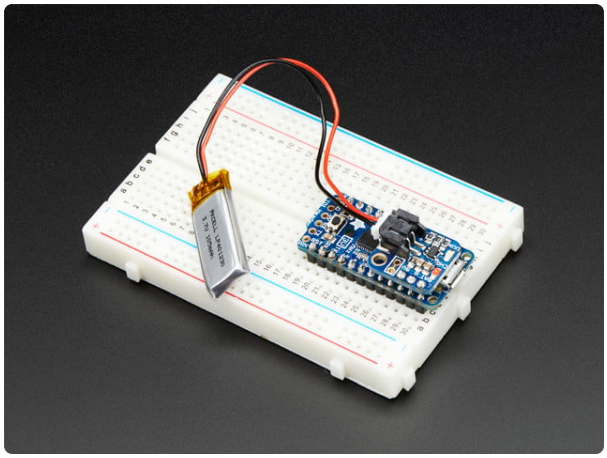
Adafruit 1.54" 240x240 Wide Angle TFT LCD Display with MicroSD

We've been looking for a display like this for a long time - it's only 1.5" diagonal but has a high density 220 ppi, 240x240 pixel display with full-angle viewing. It...
https://www.adafruit.com/product/3787



Adafruit ItsyBitsy M4 Express featuring ATSAMD51

What's smaller than a Feather but larger than a Trinket? It's an Adafruit ItsyBitsy M4 Express featuring the Microchip ATSAMD51! Small,...
https://www.adafruit.com/product/3800

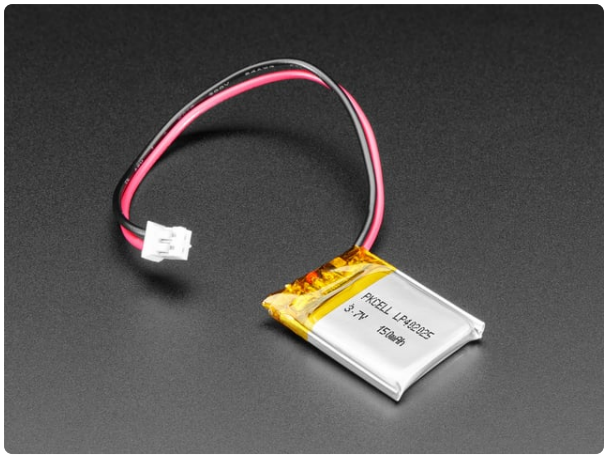


Adafruit LiIon/LiPoly Backpack Add-On for Pro Trinket/ItsyBitsy

If you have an ItsyBitsy or Pro Trinket you probably know it's the perfect little size for a portable project. This LiPoly backpack makes it really easy to do! Instead of wiring 2...
https://www.adafruit.com/product/2124

Breadboard-friendly SPDT Slide Switch

These nice switches are perfect for use with breadboard and perfboard projects. They have 0.1" spacing and snap in nicely into a solderless breadboard. They're easy to switch...
https://www.adafruit.com/product/805



[Lithium Ion Polymer Battery - 3.7v 150mAh](https://www.adafruit.com/product/1317)
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/1317

[Silicone Cover Stranded-Core Ribbon Cable - 10 Wire 1 Meter Long](https://www.adafruit.com/product/3890)
For those who are fans of our silicone-covered wires, but are always looking to up their wiring game. We now have Silicone Cover Ribbon cables! These may look...
https://www.adafruit.com/product/3890

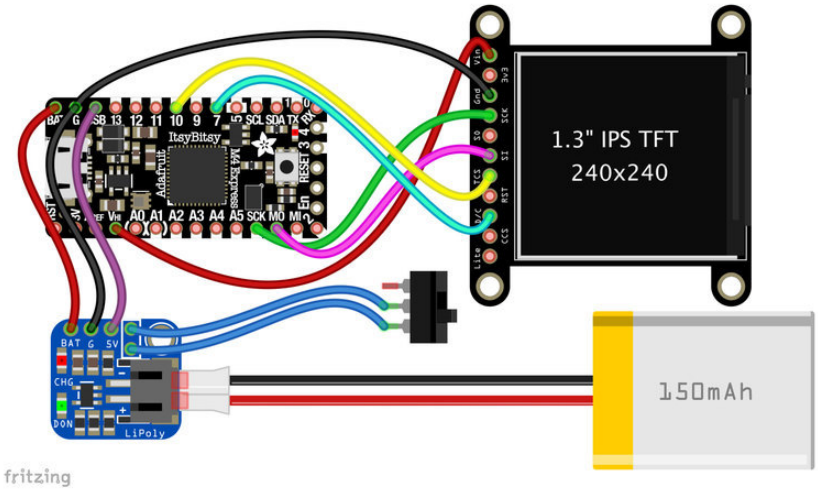
Circuit Diagram

Circuit Diagram

The diagram below provides a visual reference for wiring of the components. This diagrams was created using Fritzing software (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 the individual parts. Get the library and parts from GitHub Adafruit Fritzing Parts (https://adafru.it/AYZ).



circuit_diagram.fzz

https://adafru.it/GDo

ItsyBItsy M4 to Lipo Backpack

- **BAT** from ItsyBitsy M4 to **BAT** on Lipo Backpack
- **G** from ItsyBitsy M4 to **G** on Lipo Backpack
- **USB** from ItsyBitsy M4 to **5V** on Lipo Backpack

Display

- **VIN** from display to **Vhi** on ItsyBItsy M4
- **GND** from display to **G** on ItsyBItsy M4
- **SCK** from display to **SCK** on ItsyBItsy M4
- **SI** from display to **MO** on ItsyBItsy M4
- **D/C** from display to **7** on ItsyBItsy M4

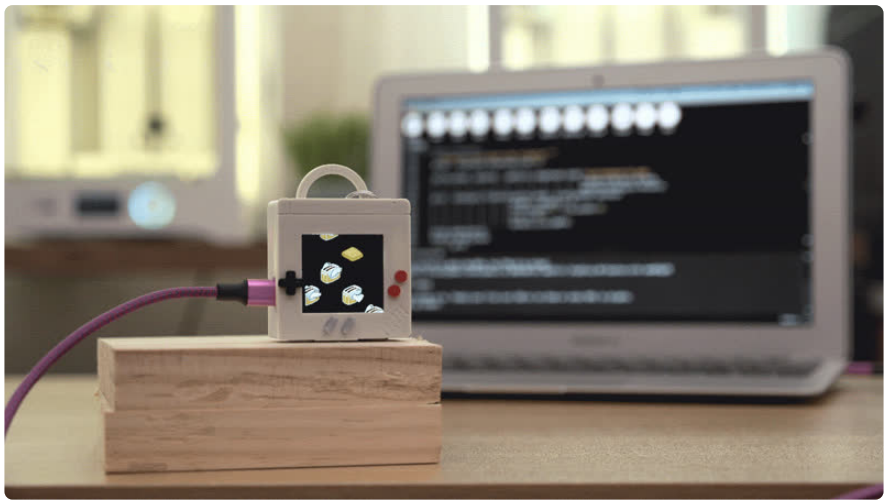
Switch

- Middle from Switch to Lipo Backpack
- Left/Right from Switch to Lipo Backpack

Power

The 150mAh lipo battery connects to the 2-pin JST port on the Lipo Backpack. The battery can be recharged over the microUSB port on the ItsyBitsy M4.

Software



Setup ItsyBitsy M4 with CircuitPython

We'll need to get our board setup so we can run CircuitPython code. Let's walk through these steps to get the latest version of CircuitPython onto your board.

The Mu Python Editor

Mu is a simple Python editor that works with Adafruit CircuitPython hardware. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in so you get immediate feedback from your board's serial output! While you can use any text editor with your code, Mu makes it super simple.

Installing and Using the Mu Editor

https://adafru.it/ANO

Installing or upgrading CircuitPython

You should ensure you have CircuitPython 4.0 or greater on your ItsyBitsy M4. Plug your board in with a known good data + power cable (not the cheesy USB cable that comes with USB power packs, they are power only). You should see a new flash drive pop up.

If the drive is **CIRCUITPY**, then open the **boot_out.txt** file to ensure the version number is 4.0 or greater.

```
Adafruit CircuitPython 5.0.0-alpha.4 on 2019-09-15; Adafruit ItsyBitsy M4 Express
with samd51j19
```

If the version is less than 4 -or- you only get a drive named **ITSYM4BOOT** then follow the steps below to update your board CircuitPython software:

- Download the CircuitPython UF2 for ItsyBitsy M4 via the green button below.
- Connect ItsyBitsy M4 to your computer over USB and press the Reset button.
- Drag-n-drop the CircuitPython **UF2** onto the **ITSYM4BOOT** drive - the drive will vanish and a new **CIRCUITPY** drive should appear.

Download CircuitPython for ItsyBItsy M4

https://adafru.it/Emj

Download the Adafruit CircuitPython Library Bundle

In order to run the code, we'll need to download a few libraries. Libraries contain code to help interface with hardware a lot easier for us.

The green button below links to a file containing all the libraries available for CircuitPython. To run the code for this project, we need the two libraries in the Required Libraries list below. Unzip the library bundle and search for the libraries. Drag and drop the files into a folder named **lib** on the **CIRCUITPY** drive (create the folder if it is not already on the ItsyBitsy M4).

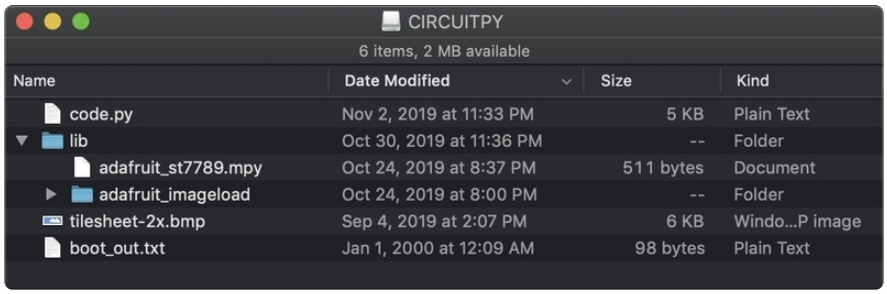
Download Circuit Python Library Bundle

https://adafru.it/ENC

Required Libraries

- **adafruit_st7789**
- **adafruit_imageload**

Once we have all the files we need, a directory listing will look similar to below as far as files and directories.



Upload Code

This project offers two different **code.py** sketches. Both programs can be used with either display.

Click on the download link below to grab the main code directly from GitHub. Rename the file to **code.py** and drop it onto the **CIRCUITPY** main (root) directory. The code will run properly when all of the files have been uploaded including libraries.

Use any text editor or favorite IDE to modify the code. We suggest using Mu as noted above.

Upload Bitmaps

Download the bitmap images below and save them to the root of the **CIRCUITPY** drive.

- **tilesheet-2x.bmp** is for the Mario Clouds code.
- **spritesheet-2x.bmp** is for the Flying Toasters code.

The bitmap images were named differently so that both files can reside in the **CIRCUITPY** drive.

Scrolling Clouds

```
# SPDX-FileCopyrightText: 2019 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Continuously scroll randomly generated Mario style clouds.
Designed for an ItsyBitsy M4 Express and a 1.3" 240x240 TFT

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!

Written by Dave Astels for Adafruit Industries
Copyright (c) 2019 Adafruit Industries
```

```
Licensed under the MIT license.

All text above must be included in any redistribution.
"""

import time
from random import seed, randint
import board
import displayio
from adafruit_st7789 import ST7789
import adafruit_imageload


# Sprite cell values
EMPTY = 0
LEFT = 1
MIDDLE = 2
RIGHT = 3

# These constants determine what happens when tiles are shifted.
# if randint(1, 10) > the value, the thing happens

# The chance a new cloud will enter
CHANCE_OF_NEW_CLOUD = 4

# The chance an existing cloud gets extended
CHANCE_OF_EXTENDING_A_CLOUD = 5

seed(int(time.monotonic()))


def make_display():
    """Set up the display support.
    Return the Display object.
    """
    spi = board.SPI()
    while not spi.try_lock():
        pass
    spi.configure(baudrate=24000000)  # Configure SPI for 24MHz
    spi.unlock()

    displayio.release_displays()
    display_bus = displayio.FourWire(spi, command=board.D7, chip_select=board.D10,
reset=board.D9)

    return ST7789(display_bus, width=240, height=240, rowstart=80,
auto_refresh=True)


def make_tilegrid():
    """Construct and return the tilegrid."""
    group = displayio.Group()

    sprite_sheet, palette = adafruit_imageload.load("/tilesheet-2x.bmp",
                                                    bitmap=displayio.Bitmap,
                                                    palette=displayio.Palette)
    grid = displayio.TileGrid(sprite_sheet, pixel_shader=palette,
                              width=9, height=5,
                              tile_height=48, tile_width=32,
                              default_tile=EMPTY)
    group.append(grid)
    display.root_group = group
    return grid


def evaluate_position(row, col):
    """Return how long of a cloud is placeable at the given location.
    :param row: the tile row (0-4)
    :param col: the tile column (0-8)
```

```
    """
    if tilegrid[col, row] != EMPTY or tilegrid[col + 1, row] != EMPTY:
        return 0
    end_col = col + 1
    while end_col < 9 and tilegrid[end_col, row] == EMPTY:
        end_col += 1
    return min([4, end_col - col])


def seed_clouds(number_of_clouds):
    """Create the initial clouds so it doesn't start empty"""
    for _ in range(number_of_clouds):
        while True:
            row = randint(0, 4)
            col = randint(0, 7)
            cloud_length = evaluate_position(row, col)
            if cloud_length > 0:
                break
        l = randint(1, cloud_length)
        tilegrid[col, row] = LEFT
        for _ in range(l - 2):
            col += 1
            tilegrid[col, row] = MIDDLE
        tilegrid[col + 1, row] = RIGHT


def slide_tiles():
    """Move the tilegrid to the left, one pixel at a time, a full time width"""
    for _ in range(32):
        tilegrid.x -= 1
        display.refresh(target_frames_per_second=60)


def shift_tiles():
    """Move tiles one spot to the left, and reset the tilegrid's position"""
    for row in range(5):
        for col in range(8):
            tilegrid[col, row] = tilegrid[col + 1, row]
        tilegrid[8, row] = EMPTY
    tilegrid.x = 0


def extend_clouds():
    """Extend any clouds on the right edge, either finishing them with a right
    end or continuing them with a middle piece
    """
    for row in range(5):
        if tilegrid[7, row] == LEFT or tilegrid[7, row] == MIDDLE:
            if randint(1, 10) > CHANCE_OF_EXTENDING_A_CLOUD:
                tilegrid[8, row] = MIDDLE
            else:
                tilegrid[8, row] = RIGHT


def add_cloud():
    """Maybe add a new cloud on the right at a random open row"""
    if randint(1, 10) > CHANCE_OF_NEW_CLOUD:
        count = 0
        while True:
            count += 1
            if count == 50:
                return
            row = randint(0, 4)
            if tilegrid[7, row] == EMPTY and tilegrid[8, row] == EMPTY:
                break
        tilegrid[8, row] = LEFT


display = make_display()
```

```
tilegrid = make_tilegrid()
seed_clouds(5)

while True:
    slide_tiles()
    shift_tiles()
    extend_clouds()
    add_cloud()
```

Flying Toasters

```
# SPDX-FileCopyrightText: 2019 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Continuously scroll randomly generated After Dark style toasters.
Designed for an ItsyBitsy M4 Express and a 1.3" 240x240 TFT

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!

Written by Dave Astels for Adafruit Industries
Copyright (c) 2019 Adafruit Industries
Licensed under the MIT license.

All text above must be included in any redistribution.
"""

import time
from random import seed, randint
import board
import displayio
from adafruit_st7789 import ST7789
import adafruit_imageload


# Sprite cell values
EMPTY = 0
CELL_1 = EMPTY + 1
CELL_2 = CELL_1 + 1
CELL_3 = CELL_2 + 1
CELL_4 = CELL_3 + 1
TOAST = CELL_4 + 1

NUMBER_OF_SPRITES = TOAST + 1

# Animation support

FIRST_CELL = CELL_1
LAST_CELL = CELL_4
NUMBER_OF_CELLS = (LAST_CELL - FIRST_CELL) + 1

# A boolean array corresponding to the sprites, True if it's part of the animation
sequence.
ANIMATED = [FIRST_CELL <= _sprite <= LAST_CELL for _sprite in
range(NUMBER_OF_SPRITES)]

# The chance (out of 10) that toast will enter
CHANCE_OF_NEW_TOAST = 2
```

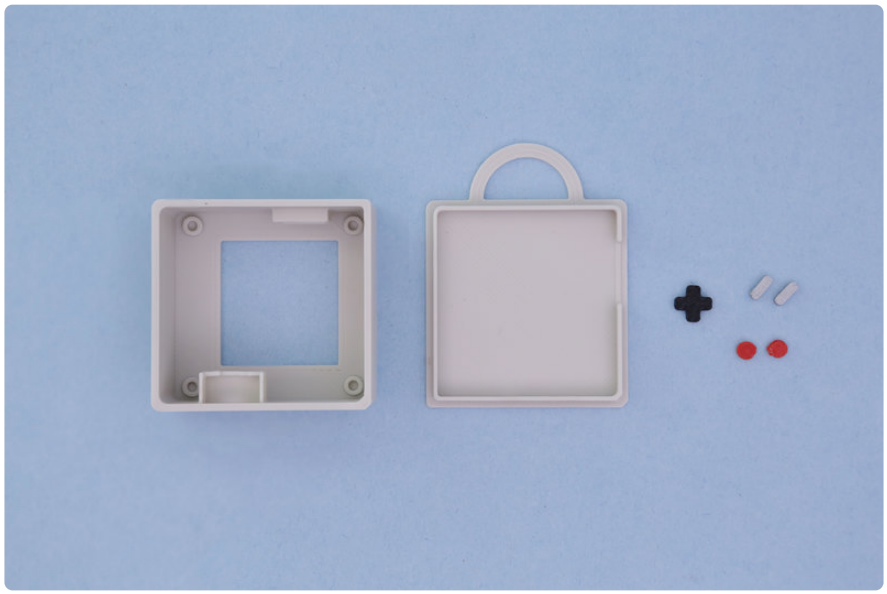
```
# How many sprites to start with
INITIAL_NUMBER_OF_SPRITES = 4

seed(int(time.monotonic()))


def make_display():
    """Set up the display support.
    Return the Display object.
    """
    spi = board.SPI()
    while not spi.try_lock():
        pass
    spi.configure(baudrate=24000000)  # Configure SPI for 24MHz
    spi.unlock()
    displayio.release_displays()
    display_bus = displayio.FourWire(spi, command=board.D7, chip_select=board.D10,
reset=board.D9)

    return ST7789(display_bus, width=240, height=240, rowstart=80,
auto_refresh=True)


def make_tilegrid():
    """Construct and return the tilegrid."""
    group = displayio.Group()

    sprite_sheet, palette = adafruit_imageload.load("/spritesheet-2x.bmp",
                                                    bitmap=displayio.Bitmap,
                                                    palette=displayio.Palette)
    grid = displayio.TileGrid(sprite_sheet, pixel_shader=palette,
                              width=5, height=5,
                              tile_height=64, tile_width=64,
                              x=0, y=-64,
                              default_tile=EMPTY)
    group.append(grid)
    display.root_group = group
    return grid


def random_cell():
    return randint(FIRST_CELL, LAST_CELL)


def evaluate_position(row, col):
    """Return whether how long of a toaster is placeable at the given location.
    :param row: the tile row (0-9)
    :param col: the tile column (0-9)
    """
    return tilegrid[col, row] == EMPTY


def seed_toasters(number_of_toasters):
    """Create the initial toasters so it doesn't start empty"""
    for _ in range(number_of_toasters):
        while True:
            row = randint(0, 4)
            col = randint(0, 4)
            if evaluate_position(row, col):
                break
        tilegrid[col, row] = random_cell()


def next_sprite(sprite):
    if ANIMATED[sprite]:
        return (((sprite - FIRST_CELL) + 1) % NUMBER_OF_CELLS) + FIRST_CELL
    return sprite
```

```
def advance_animation():
    """Cycle through animation cells each time."""
    for tile_number in range(25):
        tilegrid[tile_number] = next_sprite(tilegrid[tile_number])


def slide_tiles():
    """Move the tilegrid one pixel to the bottom-left."""
    tilegrid.x -= 1
    tilegrid.y += 1


def shift_tiles():
    """Move tiles one spot to the left, and reset the tilegrid's position"""
    for row in range(4, 0, -1):
        for col in range(4):
            tilegrid[col, row] = tilegrid[col + 1, row - 1]
        tilegrid[4, row] = EMPTY
    for col in range(5):
        tilegrid[col, 0] = EMPTY
    tilegrid.x = 0
    tilegrid.y = -64


def get_entry_row():
    while True:
        row = randint(0, 4)
        if tilegrid[4, row] == EMPTY and tilegrid[3, row] == EMPTY:
            return row


def get_entry_column():
    while True:
        col = randint(0, 3)
        if tilegrid[col, 0] == EMPTY and tilegrid[col, 1] == EMPTY:
            return col


def add_toaster_or_toast():
    """Maybe add a new toaster or toast on the right and/or top at a random open
location"""
    if randint(1, 10) <= CHANCE_OF_NEW_TOAST:
        tile = TOAST
    else:
        tile = random_cell()
    tilegrid[4, get_entry_row()] = tile

    if randint(1, 10) <= CHANCE_OF_NEW_TOAST:
        tile = TOAST
    else:
        tile = random_cell()
    tilegrid[get_entry_column(), 0] = tile


display = make_display()
tilegrid = make_tilegrid()
seed_toasters(INITIAL_NUMBER_OF_SPRITES)
display.refresh()

while True:
    for _ in range(64):
        display.refresh(target_frames_per_second=80)
        advance_animation()
        slide_tiles()
    shift_tiles()
    add_toaster_or_toast()
    display.refresh(target_frames_per_second=120)
```

Double Check

See the directory listing above and double check that you have all the files listed to make this project function. If any are missing or in an incorrect directory, move them so they're in the right places.

3D Printing



The parts for this project are designed to be 3D printed with FDM based machines. STL files are oriented to print "as is". Parts require tight tolerances that might need adjustment of slice settings. Reference the suggested settings below.

CAD Files

The parts can further be separated into small pieces for fitting on printers with smaller build volumes. Note: a STEP file is included for other 3D surface modeling programs such as Onshape, Solidworks and Rhino.

Edit Case for 1.3 Display

https://adafru.it/GDb

Edit Case for 1.5 Display

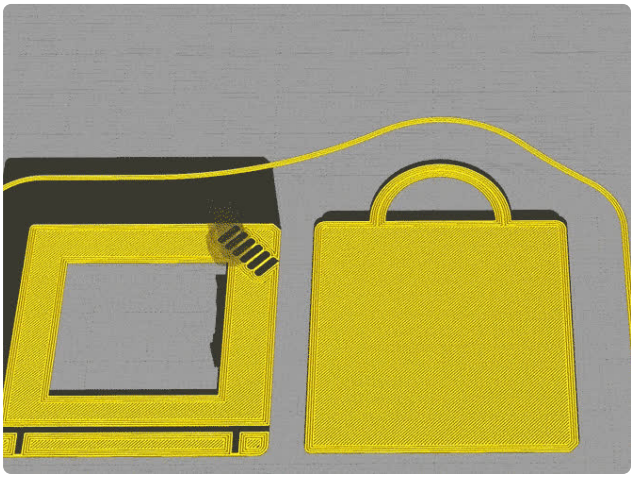
https://adafru.it/GDc

Download Button STLs

https://adafru.it/GDu

Download Case STLs

https://adafru.it/GDu

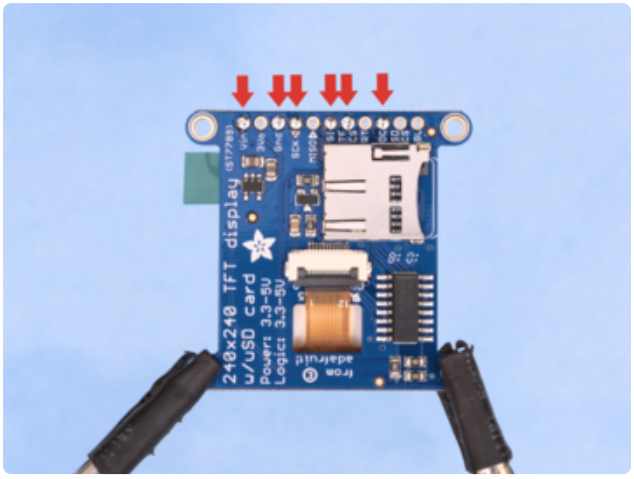


Settings

Use these settings as reference. Values listed were used in Cura slicing software.

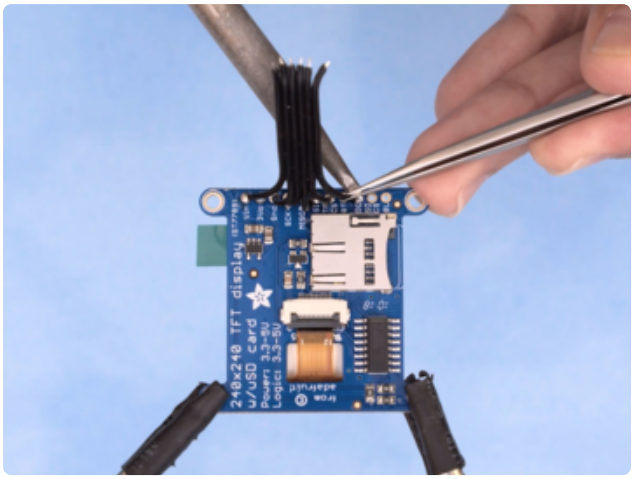
0.2mm Layer Height / 0.4mm nozzle
0.4mm Line Width (inner & outer widths)
50mm/s printing speed
10% infill
Supports: No
Skirt: 3

Assemble



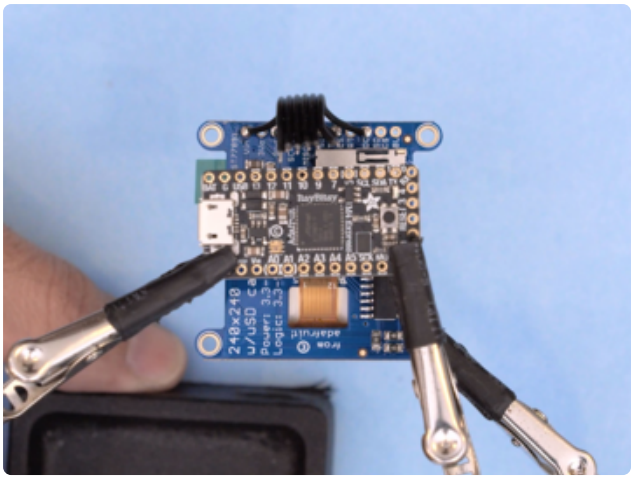
Tin Display connections

I used third helping hands to aid in stabilizing the the display. Tin the connections according to the circuit diagram.



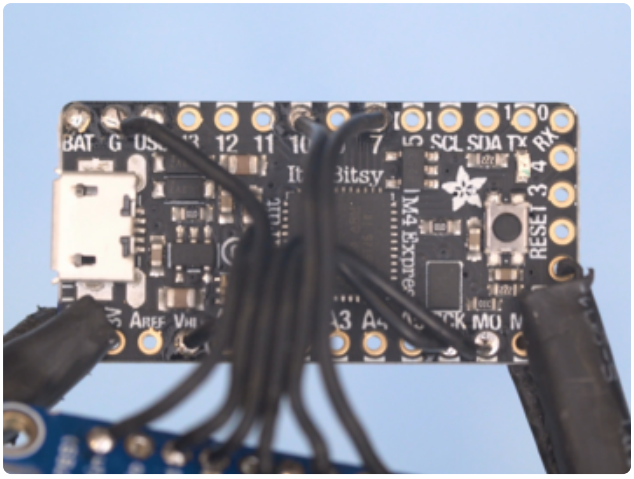
Solder Wires

I cut the wires so they are long enough to reach the ItsyBitsy. The ribbon silicone cables are great so we can keep the bundle of wire neat inside the enclosure.



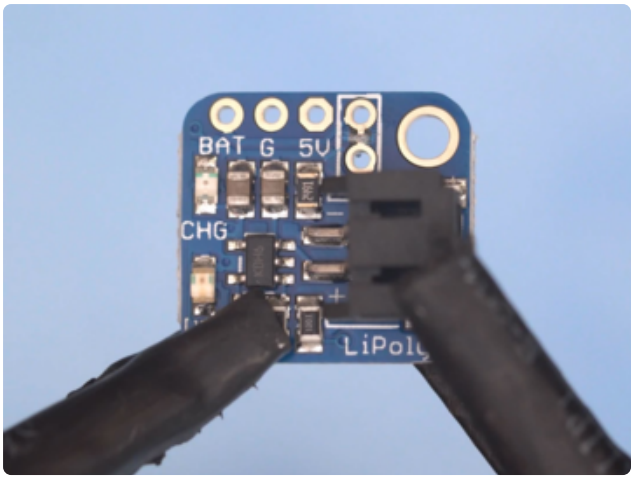
Tin ItsyBitsy

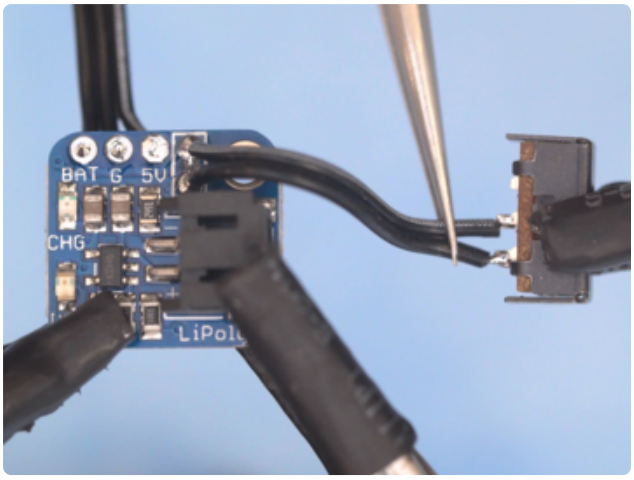
I used a second pair of helping hands to hold the ItsyBitsy above the Display. Reference the circuit diagram and tin the connections on the ItsyBitsy.



Solder ItsyBitsy to display

Gently pull apart each individual wire so that each can reach the connections on the ItsyBitsy. Reference the circuit diagram and solder the ItsyBitsy to the display.

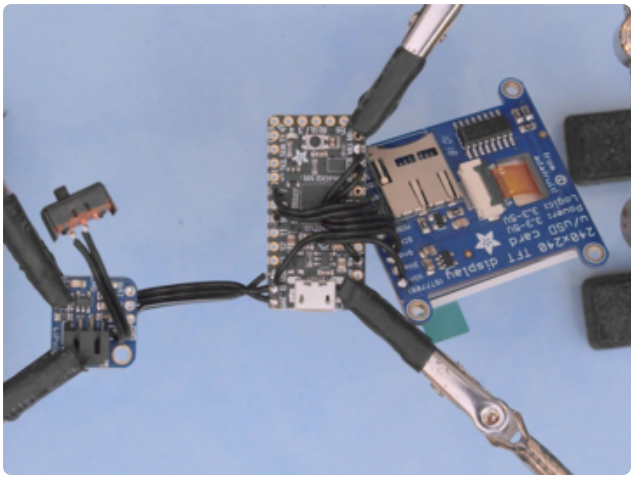


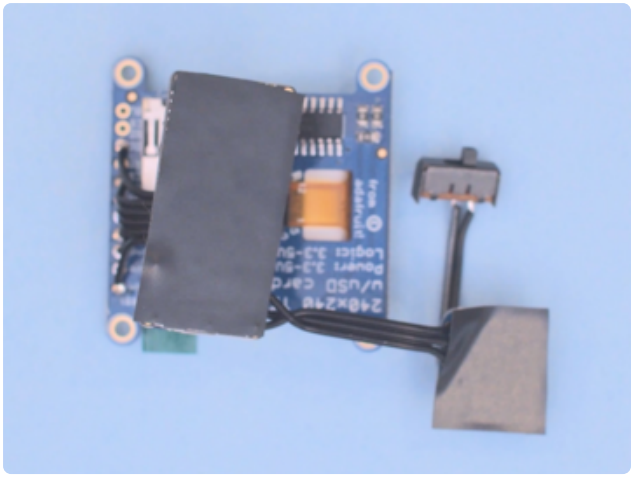


Prep Lipo Charger Backpack

Now we can prepare the LiPoly Charger Backpack. The two 0.1" holes with a box around them are the battery output line. Carefully cut the trace between them with a hobby knife and solder two wires to a slide switch.

Tin the rest of the connections on the lipo backpack. To better fit the the boards inside the enclosure, we can solder the wires from the bottom.



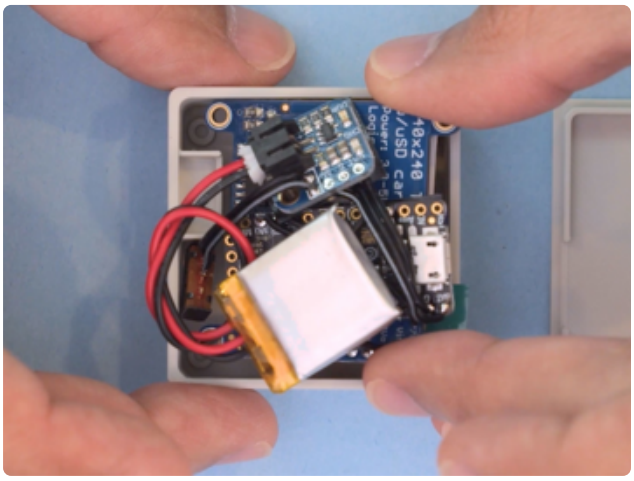


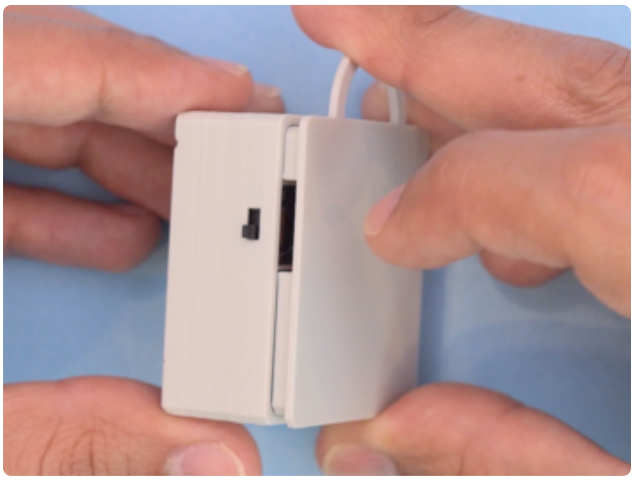
Solder Lipo Backpack to ItsyBitsy

The lipo backpack is then soldered to the ItsyBitsy with ribbon cables.

Insulate Boards

We'll need to insulate the boards so they don't touch once inside the enclosure. I used a strip of electrical tape to cover the bottom of the ItsyBitsy and the lipo backpack boards.





Insert boards into enclosure

Insert the display board into the enclosure so the side with the connections faces the cutouts for the slide switch on the enclosure.

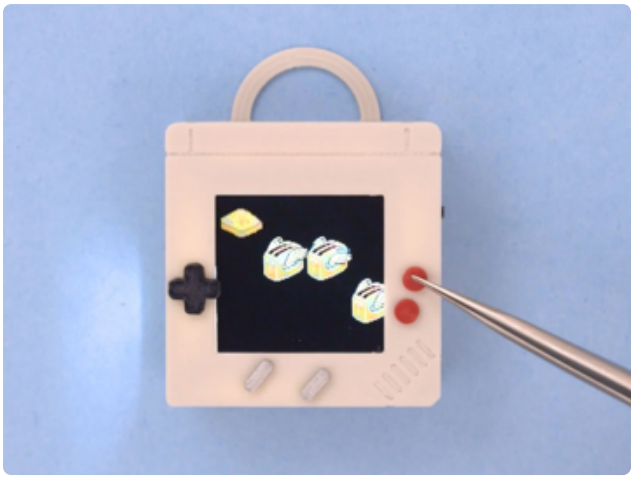
Arrange the ItsyBitsy so the USB port aligns with the USB cutout on the enclosure.

Arrange the lipo backpack with the JST port facing the slide switch cutout on the enclosure.

Connect a lipo battery to the JST on the backpack and coil the battery wires and fit the wires under the walls of the slide switch cutout on the enclosure.

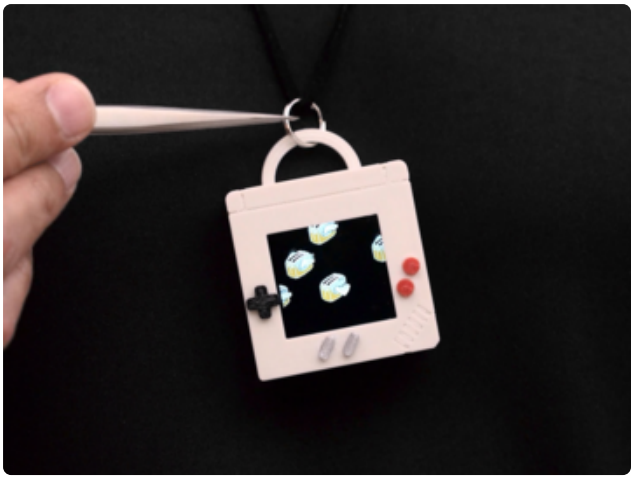
Press fit Lid

Align the lid so the cutout fits over the slide switch walls on the enclosure. Insert the lid at and angle so the battery is in the center and gently press fit the lid onto the enclosure.



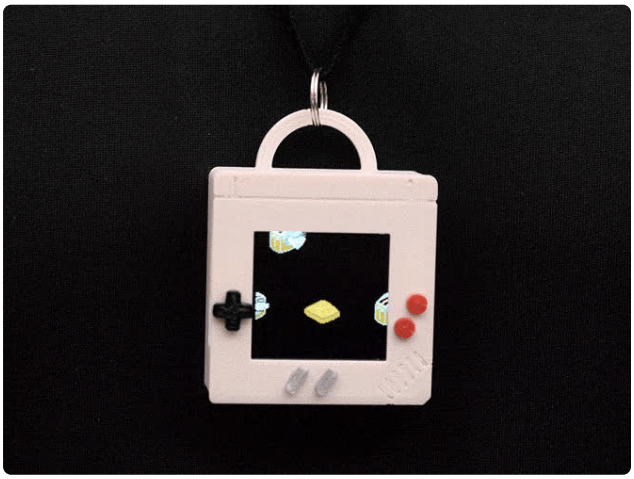
Glue Buttons

Test the layout of the buttons over the enclosure and then use a tiny bit of super glue to adhere in place.



Split Ring and Necklace

A split ring fits over the loop on the lid. A necklace or lanyard can then fit through the split ring, ready to wear!



And there you have it! That's how you can create your own retro inspired wearables!

If you have projects you'd like to share, check out Adafruit's Show and Tell live stream (https://adafru.it/GDs).

All participants get a free vinyl sticker.

You can also check out the Adafruit Discord server so you can chat with the community!