Make a Zelda Master Sword with the RP2040 Prop-Maker Feather

Created by Ruiz Brothers

https://learn.adafruit.com/master-sword-rp2040

Last updated on 2023-08-29 04:56:36 PM EDT
# Table of Contents

## Overview
- Prop-Maker Feather
- Parts

## Circuit Diagram
- Adafruit Library for Fritzing
- Wired Connections

## 3D Printing
- Parts List
- Slice with Settings for PLA material
- Supports
- Share, Make, Remix

## CircuitPython
- CircuitPython Quickstart
- Safe Mode
- Flash Resetting UF2

## Code
- Upload the Code and Libraries to the RP2040 Prop-Maker Feather
- How the CircuitPython Code Works
- NeoPixels, Audio and Accelerometer
- Functions
- The Loop

## Assembly
- Assemble Blade and Hilt
- Solder NeoPixel Strip
- Board and Speaker Holders
- Battery and slide switch
- Mount Prop-Maker
- Speaker mount
- Place NeoPixel LED Strip
- Connect to Terminal
- Attach halves
- Handle Detail
- Complete!
Build the sword that seals the darkness and defend your kingdom with Adafruit's RP2040 Prop-Maker Feather!

This iconic Zelda sword has been upgraded to be much easier to build and the assembly is even more durable.

The new dev board packs all of the features into one making this project even more enjoyable to build!
Prop-Maker Feather
The Prop-Maker Feather was designed for creating advanced props using motion, lights and sound. The LIS3DH accelerometer can detect steps, swings and hits. It has an on-board class-D audio amp for blasting sound effects. For creating stunning lighting effects, the built-in NeoPixel driver is essential.

Terminal Block Port - With easy-to-use screw terminals, you can quickly connect and disconnect, lights, speakers and buttons!
The CircuitPython code has been revised to handle the new pinouts, and features adjustable settings for the accelerometer and NeoPixels.

We think this is a great resource for building similar props that feature motion-activated effects with LED animations.
Parts

Adafruit RP2040 Prop-Maker Feather with I2S Audio Amplifier
The Adafruit Feather series gives you lots of options for a small, portable, rechargeable microcontroller board. By picking a feather and stacking on a FeatherWing you can create...
https://www.adafruit.com/product/5768

Adafruit Mini Skinny NeoPixel Digital RGB LED Strip - 144 LED/m
So thin. So mini. So teeeeeeny-tiny. It's the 'skinny' version of our classic NeoPixel strips! These NeoPixel strips have 144 digitally-addressable pixel Mini LEDs...
https://www.adafruit.com/product/2969
Lithium Ion Cylindrical Battery - 3.7v 2200mAh
Need a big battery for your project? This lithium-ion battery contains a 2200mAh and a protection circuit that provides over-voltage, under-voltage, and over-current protection. Yet,...
https://www.adafruit.com/product/1781

JST-PH Battery Extension Cable - 500mm
By popular demand, we now have a handy extension cord for all of our JST PH-terminated battery packs (such as our Lilon/LiPoly and 3xAAA holders). One end has a JST-PH compatible...
https://www.adafruit.com/product/1131

Mini Oval Speaker with Short Wires - 8 Ohm 1 Watt
Hear the good news! This wee speaker is a great addition to any audio project where you need 8 ohm impedance and 1W or less of power. We particularly like...
https://www.adafruit.com/product/4227

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
Circuit Diagram

Adafruit Library for Fritzing

The wiring diagram below provides a visual reference for connecting the components. It is not true to scale, it is just meant to be used as reference. This diagram was created using the Fritzing software package.

Take a moment to review the components in the circuit diagram. This illustration is meant for referencing wired connections - the length of wire, position and size of components are not exact.
Wired Connections

- The NeoPixel strips, and speaker are each connected to pins on the screw block terminal.
- The slide switch is connected to the EN and GND pins on the Feather.
- A 500mAh battery is connected to the battery port on the Feather via an 2 Pin JST extension cable.
3D Printing

Parts List

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

Slice with Settings for PLA material

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

PLA filament 220c extruder
0.2 layer height
10% gyroid infill
60mm/s print speed
60c heated bed
Supports
Support Overhang Angle: 50
Support Destiny: 6%
Enable Support Interface
Enable Support Roof
Support Z Distance: .21

Share, Make, Remix

This master sword was originally designed by Garrett Kearney from Chaos Core Tech (Youtube Channel) – The files are open to remix and feature nice details. The parts have been modified to fit all of the electronics and available to download. You can check out Garrett’s project build video on YouTube.

Garrett’s original design was modeled in Autodesk Fusion 360. It contains solid bodies, sketches and parametric feature timeline. It’s a great resource to check out!

CircuitPython

CircuitPython is a derivative of MicroPython 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
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 blue above). Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 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 RPI-RP2.

Drag the adafruit_circuitpython_etc.uf2 file to RPI-RP2.

The RPI-RP2 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 doesn't even show up as a disk drive when installing CircuitPython, try loading this 'nuke' UF2 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

Code

Once you've finished setting up your RP2040 Prop-Maker Feather with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

To do this, click on the Download Project Bundle button in the window below. It will download as a zipped folder.

```python
# SPDX-FileCopyrightText: 2019 Kattni Rembor for Adafruit Industries
# SPDX-FileCopyrightText: 2019 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# RP2040 Prop-Maker Feather Master Sword
Adafruit invests time and resources providing this open source code. Please support Adafruit and open source hardware by purchasing products from Adafruit!
Written by Kattni Rembor & Limor Fried for Adafruit Industries
Copyright (c) 2019-2020 Adafruit Industries
Licensed under the MIT license.
All text above must be included in any redistribution.

import time
import random
import digitalio
import audiocore
import audiobusio
import board
import neopixel
import adafruit_lis3dh

# CUSTOMISE COLORS HERE:
COLOR = (0, 120, 120)  # Default idle is light blue
ALT_COLOR = (255, 50, 0)  # hit color is orange

# CUSTOMISE IDLE PULSE SPEED HERE: 0 is fast, above 0 slows down
IDLE_PULSE_SPEED = 0  # Default is 0 seconds
SWING_BLAST_SPEED = 0.0007

# CUSTOMISE BRIGHTNESS HERE: must be a number between 0 and 1
IDLE_PULSE_BRIGHTNESS_MIN = 0.2  # Default minimum idle pulse brightness
```
IDLE_PULSE_BRIGHTNESS_MAX = 1  # Default maximum idle pulse brightness

# CUSTOMISE SENSITIVITY HERE: smaller numbers = more sensitive to motion
HIT_THRESHOLD = 250
SWING_THRESHOLD = 150

# Set to the length in seconds of the "on.wav" file
POWER_ON_SOUND_DURATION = 1.7

NUM_PIXELS = 73  # Number of pixels used in project
NEOPIXEL_PIN = board.EXTERNAL_NEOPIXELS
enable = digitalio.DigitalInOut(board.EXTERNAL_POWER)
enable.direction = digitalio.Direction.OUTPUT
enable.value = True

strip = neopixel.NeoPixel(NEOPIXEL_PIN, NUM_PIXELS, brightness=1, auto_write=False)
strip.fill(0)  # NeoPixels off ASAP on startup
strip.show()

# i2s audio
audio = audiobusio.I2SOut(board.I2S_BIT_CLOCK, board.I2S_WORD_SELECT, board.I2S_DATA)
wave_file = None

# Set up accelerometer on I2C bus, 4G range:
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller
accel = adafruit_lis3dh.LIS3DH_I2C(i2c)
accel.range = adafruit_lis3dh.RANGE_4_G

COLOR_IDLE = COLOR # 'idle' color is the default
COLOR_HIT = ALT_COLOR  # "hit" color is ALT_COLOR set above
COLOR_SWING = ALT_COLOR  # "swing" color is ALT_COLOR set above

def play_wav(name, loop=False):
    """
    Play a WAV file in the 'sounds' directory.
    :param name: partial file name string, complete name will be built around
    this, e.g. passing 'foo' will play file 'sounds/foo.wav'.
    :param loop: if True, sound will repeat indefinitely (until interrupted
    by another sound).
    """
    global wave_file # pylint: disable=global-statement
    print("playing", name)
    if wave_file:
        wave_file.close()
    try:
        wave_file = open('sounds/' + name + '.wav', 'rb')
        wave = audiocore.WaveFile(wave_file)
        audio.play(wave, loop=loop)
    except OSError:
        pass # we'll just skip playing then

def power_on(sound, duration):
    """
    Animate NeoPixels with accompanying sound effect for power on.
    :param sound: sound name (similar format to play_wav() above)
    :param duration: estimated duration of sound, in seconds (>=0.0)
    """
    prev = 0
    start_time = time.monotonic()  # Save audio start time
    play_wav(sound)
    while True:
        elapsed = time.monotonic() - start_time  # Time spent playing sound
        if elapsed > duration:  # Past sound duration?
            break
break  # Stop animating
animation_time = elapsed / duration  # Animation time, 0.0 to 1.0
threshold = int(NUM_PIXELS * animation_time + 0.5)
num = threshold - prev  # Number of pixels to light on this pass
if num != 0:
    strip[prev:threshold] = [ALT_COLOR] * num
    strip.show()
    prev = threshold

def mix(color_1, color_2, weight_2):
    
    def _mix(r, g, b, weight_2):
        return int(r * weight_1 + g * weight_2), int(g * weight_1 + b * weight_2), int(b * weight_1 + r * weight_2)
    
    if weight_2 < 0.0:
        weight_2 = 0.0
    elif weight_2 > 1.0:
        weight_2 = 1.0
    weight_1 = 1.0 - weight_2
    return _mix(color_1[0], color_2[0], weight_2), _mix(color_1[1], color_2[1], weight_2), _mix(color_1[2], color_2[2], weight_2)

    swing_sounds = ['swing1', 'swing2', 'swing3', 'swing4',]

    hit_sounds = ['hit1', 'hit2', 'hit3', 'hit4',]

mode = 0  # Initial mode = OFF

# Setup idle pulse
idle_brightness = IDLE_PULSE_BRIGHTNESS_MIN  # current brightness of idle pulse
idle_increment = 0.01  # Initial idle pulse direction

# Main loop
while True:
    if mode == 0:
        enable.value = True
        power_on('on', POWER_ON_SOUND_DURATION)  # Power up!
        play_wav('idle', loop=True)  # Play idle sound now
        mode = 1  # Idle mode

        # Setup for idle pulse
        idle_brightness = IDLE_PULSE_BRIGHTNESS_MIN
        idle_increment = 0.01
        strip.fill([int(c*idle_brightness) for c in COLOR])
        strip.show()

    elif mode >= 1:
        x, y, z = accel.acceleration  # Read accelerometer
        accel_total = x * x + z * z
        # (Y axis isn't needed, due to the orientation that the Prop-Maker

© Adafruit Industries
# Wing is mounted. Also, square root isn't needed, since we're # comparing thresholds...use squared values instead.)
if accel_total > HIT_THRESHOLD:  # Large acceleration = HIT
    TRIGGER_TIME = time.monotonic()  # Save initial time of hit
    play_wav(random.choice(hit_sounds))  # Start playing 'hit' sound
    COLOR_ACTIVE = COLOR_HIT  # Set color to fade from
    mode = 3  # HIT mode
elif mode == 1 and accel_total > SWING_THRESHOLD:  # Mild = SWING
    TRIGGER_TIME = time.monotonic()  # Save initial time of swing
    play_wav(random.choice(swing_sounds))  # Randomly choose from available
    swing sounds
    # make a larson scanner animation_time
    strip_backup = strip[0:-1]
    for p in range(-1, len(strip)):
        for i in range (p-1, p+2):  # shoot a 'ray' of 3 pixels
            if 0 <= i < len(strip):
                strip[i] = COLOR_SWING
        strip.show()
        time.sleep(SWING_BLAST_SPEED)
        if 0 <= (p-1) < len(strip):
            strip[p-1] = strip_backup[p-1]  # restore previous color at the
tail
        strip.show()
    while audio.playing:
        pass  # wait till we're done
    mode = 2  # we'll go back to idle mode
elif mode == 1:
    # Idle pulse
    idle_brightness += idle_increment  # Pulse up
    if idle_brightness > IDLE_PULSE_BRIGHTNESS_MAX or 
        idle_brightness < IDLE_PULSE_BRIGHTNESS_MIN:  # Then...
        idle_increment *= -1  # Pulse direction flip
    strip.fill([int(c*idle_brightness) for c in COLOR_IDLE])
    strip.show()
    time.sleep(IDLE_PULSE_SPEED)  # Idle pulse speed set above
elif mode > 1:  # If in SWING or HIT mode...
    if audio.playing:  # And sound currently playing...
        blend = time.monotonic() - TRIGGER_TIME  # Time since triggered
        if mode == 2:  # If SWING,
            blend = abs(0.5 - blend) * 2.0  # ramp up, down
            strip.fill(mix(COLOR_ACTIVE, COLOR, blend))  # Fade from hit/swing
to base color
        else:  # No sound now, but still SWING or HIT modes
            play_wav('idle', loop=True)  # Resume idle sound
            mode = 1  # Return to idle mode

Upload the Code and Libraries to the RP2040 Prop-Maker Feather

After downloading the Project Bundle, plug your RP2040 Prop-Maker Feather into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called CIRCUITPY. Unzip the folder and copy the following items to the RP2040 Prop-Maker Feather's CIRCUITPY drive.

- lib folder
- sounds folder
Your RP2040 Prop-Maker Feather CIRCUITPY drive should look like this after copying the lib folder, sounds folder and the code.py file.

How the CircuitPython Code Works

At the top of the code, you can customize a few attributes for your sword. `COLOR` and `ALT_COLOR` affect the NeoPixel color of the sword. `IDLE_PULSE_SPEED` and `SWING_BLAST_SPEED` affect the NeoPixel pulse speed. `IDLE_PULSE_BRIGHTNESS_MIN` and `IDLE_PULSE_BRIGHTNESS_MAX` affect the NeoPixel brightness range. `HIT_THRESHOLD` and `SWING_THRESHOLD` affect the motion sensitivity of the sword.

```python
# CUSTOMISE COLORS HERE:
COLOR = (0, 120, 120)      # Default idle is light blue
ALT_COLOR = (255, 50, 0)  # hit color is orange

# CUSTOMISE IDLE PULSE SPEED HERE: 0 is fast, above 0 slows down
IDLE_PULSE_SPEED = 0  # Default is 0 seconds
SWING_BLAST_SPEED = 0.0007

# CUSTOMISE BRIGHTNESS HERE: must be a number between 0 and 1
IDLE_PULSE_BRIGHTNESS_MIN = 0.2  # Default minimum idle pulse brightness
IDLE_PULSE_BRIGHTNESS_MAX = 1    # Default maximum idle pulse brightness

# CUSTOMISE SENSITIVITY HERE: smaller numbers = more sensitive to motion
HIT_THRESHOLD = 250
SWING_THRESHOLD = 150
```
NeoPixels, Audio and Accelerometer

The NeoPixels, I2S amp and accelerometer are instantiated.

```python
NUM_PIXELS = 73  # Number of pixels used in project
NEOPIXEL_PIN = board.EXTERNAL_NEOPIXELS

enable = digitalio.DigitalInOut(board.EXTERNAL_POWER)
enable.direction = digitalio.Direction.OUTPUT
enable.value = True

strip = neopixel.NeoPixel(NEOPIXEL_PIN, NUM_PIXELS, brightness=1, auto_write=False)
strip.fill(0)  # NeoPixels off ASAP on startup
strip.show()

# i2s audio
audio = audiobusio.I2SOut(board.I2S_BIT_CLOCK, board.I2S_WORD_SELECT,
                        board.I2S_DATA)
wave_file = None

# Set up accelerometer on I2C bus, 4G range:
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
# microcontroller
accel = adafruit_lis3dh.LIS3DH_I2C(i2c)
accel.range = adafruit_lis3dh.RANGE_4_G
```

Functions

There are a few functions that are used throughout the loop. `play_wav()` plays the sound effects that are contained in the /sounds folder.

```python
def play_wav(name, loop=False):
    
    Play a WAV file in the 'sounds' directory.
    :param name: partial file name string, complete name will be built around
                this, e.g. passing 'foo' will play file 'sounds/foo.wav'.
    :param loop: if True, sound will repeat indefinitely (until interrupted
                 by another sound).
    
    global wave_file  # pylint: disable=global-statement
    print("playing", name)
    if wave_file:
        wave_file.close()
    try:
        wave_file = open('sounds/' + name + '.wav', 'rb')
        wave = audiocore.WaveFile(wave_file)
        audio.play(wave, loop=loop)
    except OSError:
        pass  # we'll just skip playing then
```

The `power_on()` function animates the NeoPixels and plays the sound effect when the sword powers up.

```python
def power_on(sound, duration):
    
    Animate NeoPixels with accompanying sound effect for power on.
```
The `mix()` function mixes the two NeoPixel colors together.

```python
def mix(color_1, color_2, weight_2):
    # Blend between two colors with a given ratio.
    # :param color_1: first color, as an (r,g,b) tuple
    # :param color_2: second color, as an (r,g,b) tuple
    # :param weight_2: Blend weight (ratio) of second color, 0.0 to 1.0
    # :return (r,g,b) tuple, blended color

    if weight_2 < 0.0:
        weight_2 = 0.0
    elif weight_2 > 1.0:
        weight_2 = 1.0
    weight_1 = 1.0 - weight_2
    return (int(color_1[0] * weight_1 + color_2[0] * weight_2),
            int(color_1[1] * weight_1 + color_2[1] * weight_2),
            int(color_1[2] * weight_1 + color_2[2] * weight_2))
```

The Loop

In the main loop, the `mode` number determines the functionality of the sword. The first time the code runs, `mode` is 0. This triggers the sword to initiate the `power_on` function and then go into idle mode where `mode` is 1.

```python
if mode == 0:  # If currently off...
    enable.value = True
    power_on('on', POWER_ON_SOUND_DURATION)  # Power up!
    play_wav('idle', loop=True)  # Play idle sound now
    mode = 1  # Idle mode

    # Setup for idle pulse
    idle_brightness = IDLE_PULSE_BRIGHTNESS_MIN
    idle_increment = 0.01
    strip.fill([int(c*idle_brightness) for c in COLOR])
    strip.show()
```

If `mode` is 1, then the accelerometer is read. If the accelerometer's reading is greater than the movement thresholds then the sword will either perform the hit or swing
modes with light and sound effects. Otherwise, the sword remains in idle mode and the NeoPixels pulse.

```python
elif mode &gt;= 1:  # If not OFF mode...
    x, y, z = accel.acceleration  # Read accelerometer
    accel_total = x * x + z * z
    # (Y axis isn't needed, due to the orientation that the Prop-Maker
    # Wing is mounted. Also, square root isn't needed, since we're
    # comparing thresholds...use squared values instead.)
    if accel_total &gt; HIT_THRESHOLD:  # Large acceleration = HIT
        TRIGGER_TIME = time.monotonic()  # Save initial time of hit
        play_wav(random.choice(hit_sounds))  # Start playing 'hit' sound
        COLOR_ACTIVE = COLOR_HIT  # Set color to fade from
        mode = 3  # HIT mode
    elif mode == 1 and accel_total &gt; SWING_THRESHOLD:  # Mild = SWING
        TRIGGER_TIME = time.monotonic()  # Save initial time of swing
        play_wav(random.choice(swing_sounds))  # Randomly choose from available
        swing sounds
        # make a larson scanner animation_time
        strip_backup = strip[0:-1]
        for p in range(-1, len(strip)):
            for i in range (p-1, p+2): # shoot a 'ray' of 3 pixels
                if 0 &lt;= i &lt; len(strip):
                    strip[i] = COLOR_SWING
        strip.show()
        time.sleep(SWING_BLAST_SPEED)
        if 0 &lt;= (p-1) &lt; len(strip):
            strip[p-1] = strip_backup[p-1]  # restore previous color at the
tail
        strip.show()
        while audio.playing:
            pass  # wait till we're done
        mode = 2  # we'll go back to idle mode
    elif mode == 1:
        # Idle pulse
        idle_brightness += idle_increment  # Pulse up
        if idle_brightness &gt;= IDLE_PULSE_BRIGHTNESS_MAX or \
            idle_brightness &lt;= IDLE_PULSE_BRIGHTNESS_MIN:  # Then...
            idle_increment *= -1  # Pulse direction flip
        strip.fill([int(c*idle_brightness) for c in COLOR_IDLE])
        strip.show()
        time.sleep(IDLE_PULSE_SPEED)  # Idle pulse speed set above
    elif mode &gt;= 1:  # If in SWING or HIT mode...
        if audio.playing:  # And sound currently playing...
            blend = time.monotonic() - TRIGGER_TIME  # Time since triggered
            if mode == 2:  # If SWING,
                blend = abs(0.5 - blend) * 2.0  # ramp up, down
            strip.fill(mix(COLOR_ACTIVE, COLOR, blend))  # Fade from hit/swing
to base color
        else:  # No sound now, but still SWING or HIT modes
            play_wav('idle', loop=True)  # Resume idle sound
        mode = 1  # Return to idle mode
```

©Adafruit Industries
Assembly

Assemble Blade and Hilt

Use M2.5x12mm long screws to attach the blades and hilt together.
Solder NeoPixel Strip
Measure, cut and solder two NeoPixel strips about 73 pixels long, in parallel.
Board and Speaker Holders
Use two M2.5x5mm screws to fasten the Prop-Maker RP2040 to the holder.

The speaker press fits face down into the holder.

Both holders use two additional M2.5x5mm screws to attach inside the handle.
Battery and slide switch
Use two M2.5x5mm long screws to fasten the battery holder to the handle. The battery press fits with the wire positioned outside of the holder.

Solder a 2 Pin JST plug to the slide switch. Pass the JST plug through the cutout on the handle to press into place.

Solder a matching 2 Pin JST socket cable to the EN and GND pins on the Prop-Maker.

Mount Prop-Maker
Align the holder to the screw mounts on the handle with the USB-C port point out.

Attach a JST extension to reach the battery. Place the wires under the board and use M25.x5mm screws to fasten the holder to the handle.
Speaker mount
Align the speaker and mount to the screw holes on the handle.

Use a Pico cable socket to extend the wire and connect to the + and - terminals on the Prop-Maker.

Place NeoPixel LED Strip
Center the two NeoPixel Strips back to back with the LEDs facing the blade edge.

Connect to Terminal
Connect the remaining wires to the terminals on Prop-Maker.

Place wires in the center so they don't block the sides of walls.
Attach halves
Use M3x20mm screws to attach both sides together.

Handle Detail
Paint the handle or you may use Paracord to add detail to the handle.

Complete!