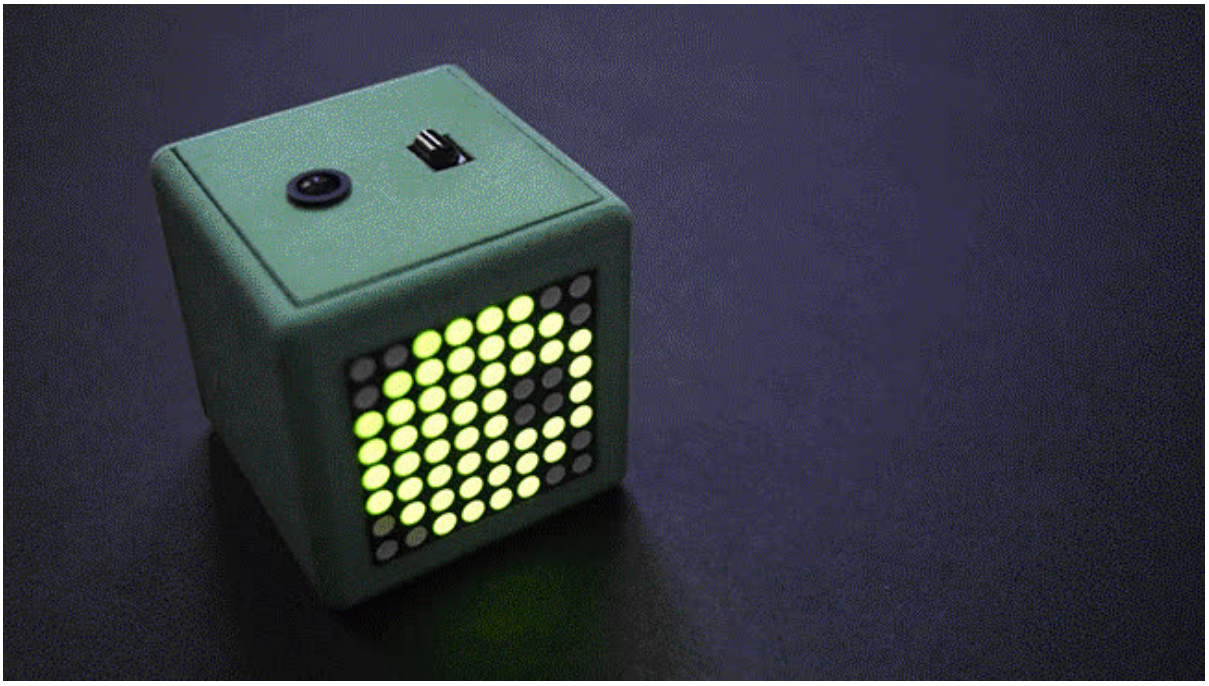
Adafriend the Virtual Pet Cube

Created by John Wall



https://learn.adafruit.com/adafriend

Last updated on 2024-03-08 02:27:35 PM EST

Table of Contents

Project Overview 5

- Recommended Precursor Guides
- Complexity Warning
- Tools Needed

Parts Required 9

- Microcontroller
- Pro Trinket
- Display
- 1.2" LED Matrix and Backpack
- 3D Printing
- Interaction
- Fast Vibration Sensor Switch
- Small Piezo Buzzer
- IR LED and Reciever
- Parts List
- Adafruit Pro Trinket - 5V 16MHz
- Adafruit Pro Trinket LiIon/LiPoly Backpack Add-On
- Lithium Ion Polymer Battery - 3.7v 500mAh
- Breadboard-friendly SPDT Slide Switch
- Small 1.2" 8x8 LED Matrix w/I2C Backpack - Pure Green
- Fast Vibration Sensor Switch (Easy to trigger)
- Small Enclosed Piezo w/Wires
- IR (Infrared) Receiver Sensor - TSOP38238
- Super-bright 5mm IR LED - 940nm
- Silicone Cover Stranded-Core Wire - 26AWG in Various Colors
- Non-Adafruit Parts
- AdafriendTop.stl +
- AdafruitBottom.stl
- x1
-
- Files found below!
- 4mm M2 machine screws x7
- 5mm M2 machine screws x4

Wiring and Assembly 18

- Circuit Connections
- Assembly Instructions
- 3D Printing
- Assembly

Software 57

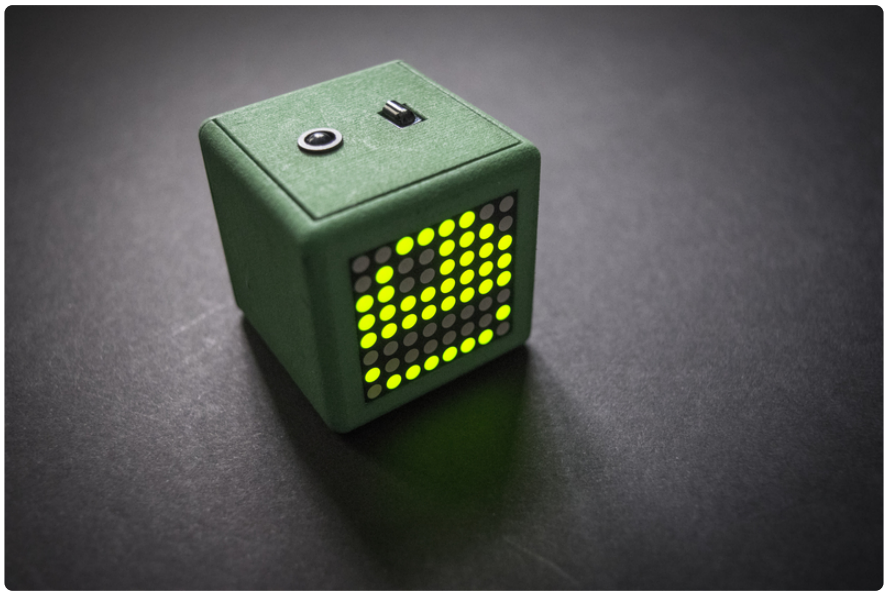
- Arduino IDE Preparation
- Required Libraries
- Adafriend Software

Use and Behavior 71

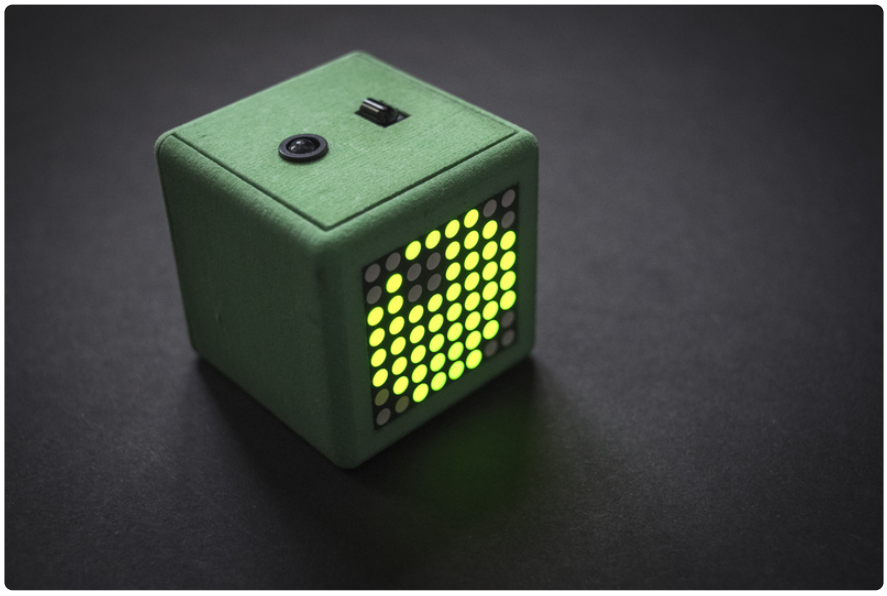
- Moods
- Sad
- Neutral
- Happy
- Annoyed

... 73

Project Overview



In this Pro Trinket powered project, build and 3D print yourself a friend, an Adafriend! This little colorful guy with an eye responds to taps and vibrations, and shows emotion on its little LED matrix front with a sad, neutral, happy, or angry face. It gazes around at its environment and blinks, plays spontaneous tones and patterns that correspond to it's emotional state, and even sings some recognizable tunes and themes when it's in the mood! If left alone for too long it gets lonely, but pester your pet too much and it may get angry at you! Build yourself an Adafriend and keep it happy to sing along with the coolest little cube around!



Recommended Precursor Guides

Before embarking on this guide, I recommend you take a look at the 1.2" LED Matrix Backpack (https://adafru.it/onf) guide to familiarize yourself with the core hardware and software of the guide. The Ruiz Brothers' guides also involve thin wires in tight spaces, here is their profile (https://adafru.it/onA). Also check out Phillip Burgess's LED Matrix animating guides (https://adafru.it/doM) for more LED eye costumes and creations, the base code and inspiration for this project!

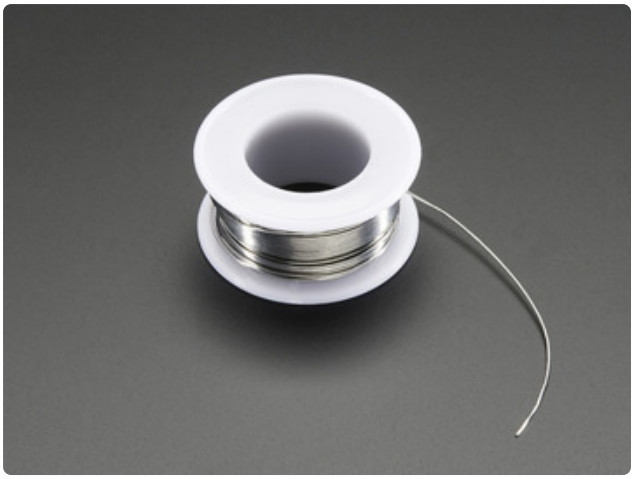
Complexity Warning

This guide involves very thin gauge wiring that requires tweezers to solder, screw tapping, heat shrinking, and a bit of hacking to miniaturize the circuit. It is also somewhat difficult to test and rework, so do not proceed unless you are confident in your abilities and understand the above recommended guides.

Tools Needed

For this project you will need a soldering iron, solder, diagonal cutters, wire strippers, tweezers, a phillips screwdriver, a hot glue gun, and a lighter, heat gun, or blowtorch to shrink heat shrink tubing.

The Hakko FX-888D (http://adafru.it/1204), my personal favorite Soldering iron



60/40 rosin core solder (http://adafru.it/1886) for easy flowing, strong solder joints

Flush diagonal cutters (http://adafru.it/152) essential for quick, precise snips of wire and leads

One thing you really can't skimp on is a good pair of wire strippers (http://adafru.it/527), these are the best balance of price and quality from Hakko, a very good buy

Fine tip curved tweezers (http://adafru.it/422), perfect for guiding wires accurately into place for soldering and keeping fingers away from very hot things

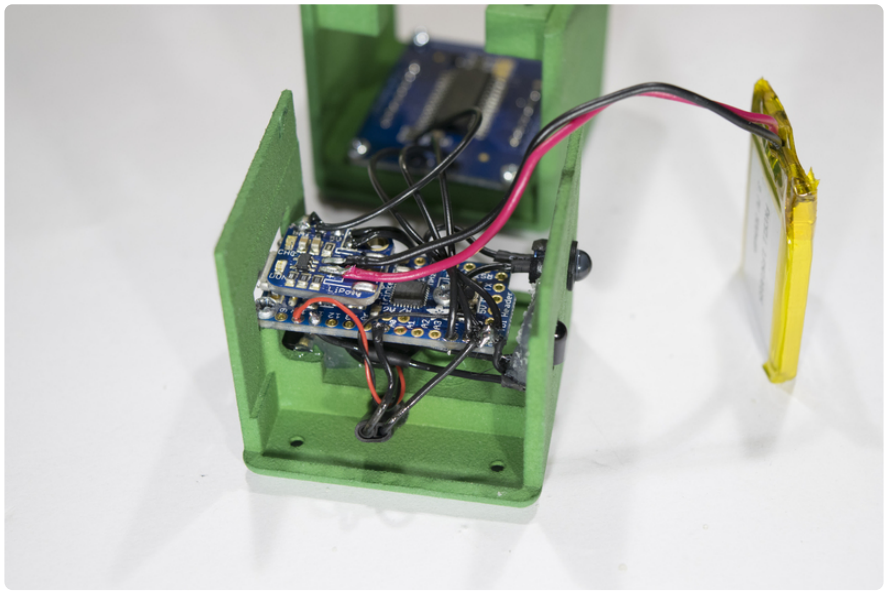
A precision screwdriver set (http://adafru.it/424) comes in handy for projects like this that require screws or some leverage, pick one up in the adafruit shop!

A hot glue gun is needed to secure the IR receiver and power switch in place.

A standard lighter, blowtorch, or heat gun should shrink heat shrink tubing quite nicely

With these tools gathered, we can move on to the cute little cube friend part of the guide!

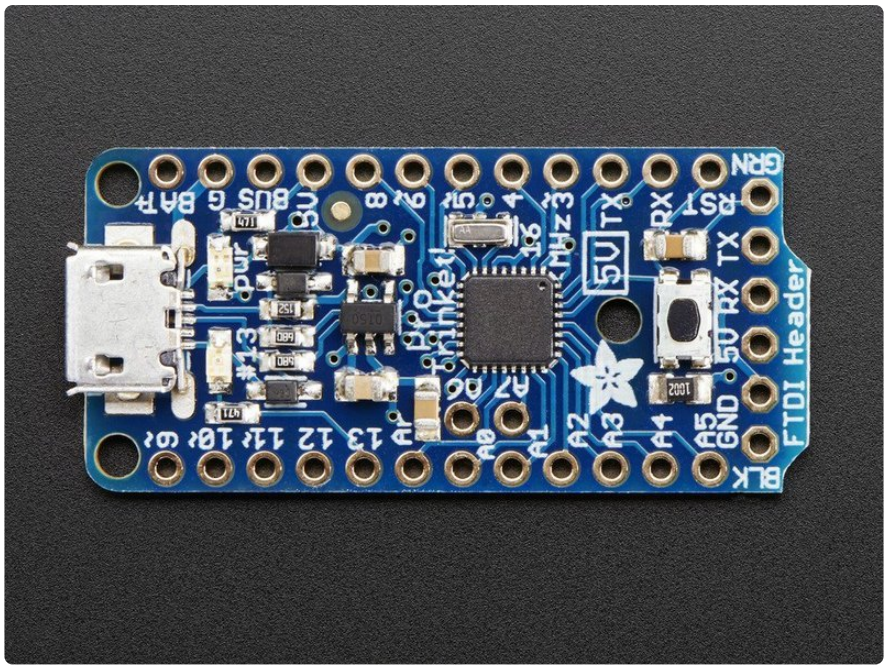
Parts Required



Microcontroller

Pro Trinket

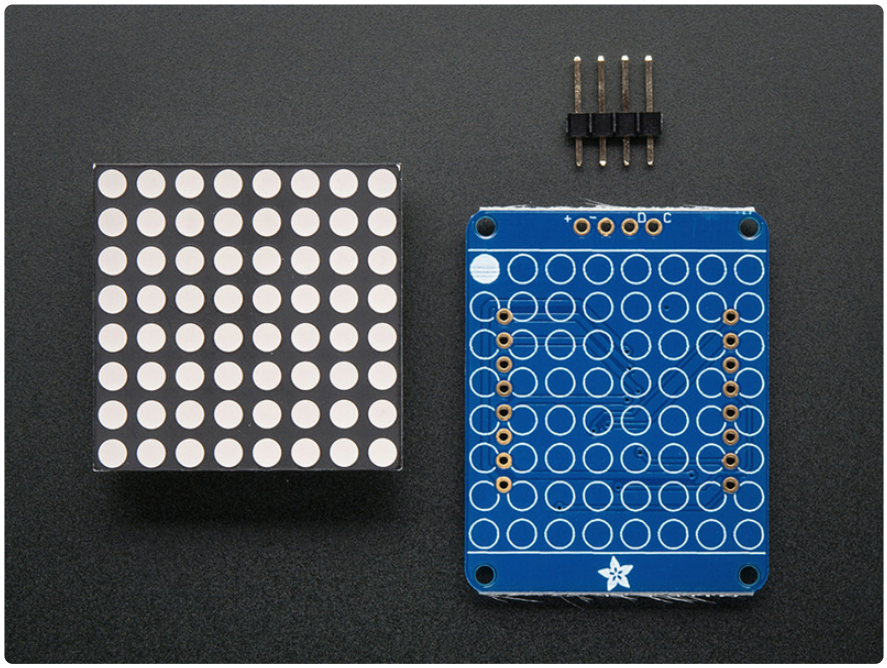
At the heart of the Adafriend (awwwww) is none other than a Pro Trinket, the more powerful (and spacious) big brother of the lovable Adafruit Trinket. With its fancy bootloader and USB programming, tiny size and price, and Atmega328 power, it is the perfect microcontroller for this project.



Display

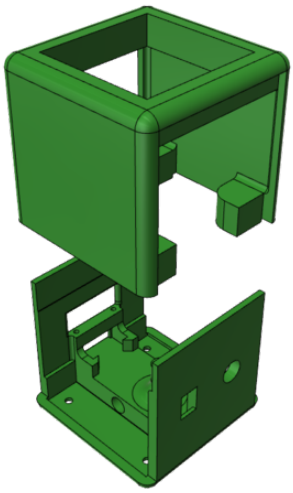
1.2" LED Matrix and Backpack

The Adafriend's beautiful face is a matrix of 64 leds controlled by an i2C backpack that comes in many different colors. From base eye movement algorithms from Phillip Burgess's many costumes and spooky LED faces, the Adafriend adds different emotions and behaviors as well as sensing and sound, and blinks sadly, happily, angrily, or indifferently at the world with one of these matrices.



3D Printing

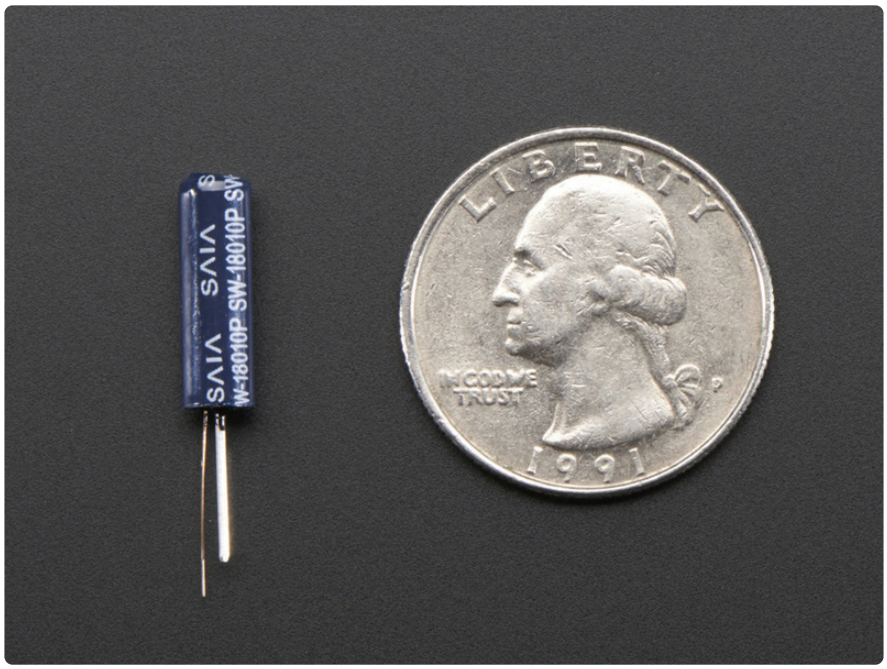
The casing of the Adafriend virtual pet is 3D printed in two halves that slide together and screw closed with 4 5mm M2 screws. I highly recommend printing your case with a high quality 3D printing service online for the color and quality required of this project. Shapeways or Sculpteo are perfect, use a plastic material. If you print your case with a personal FDM printer, print at the highest quality possible to ensure a correct print.



Interaction

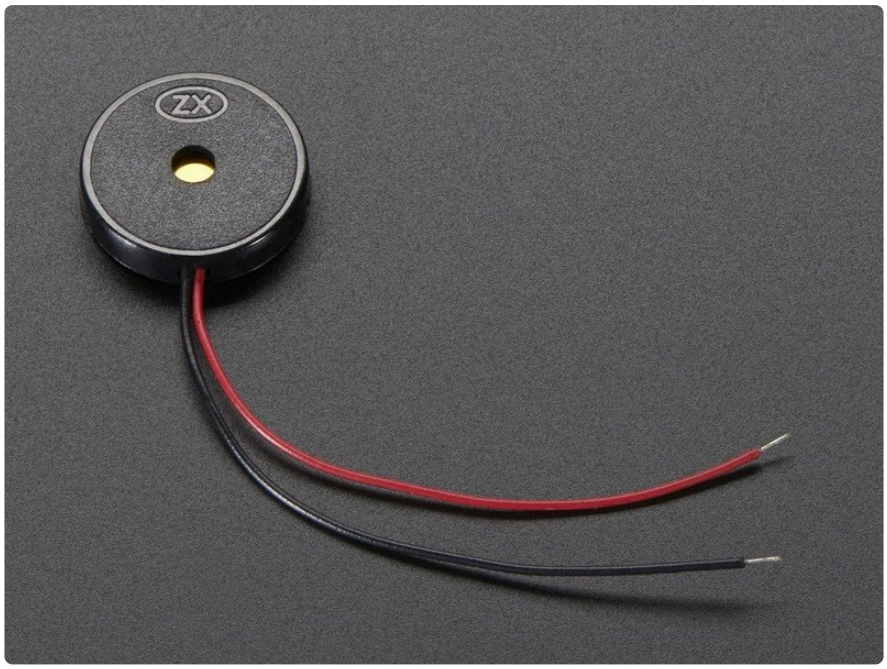
Fast Vibration Sensor Switch

This sensor has a tiny spring inside that when vibrated, acts like a switch and cloes the circuit between its two pins. This allows the cube to sense taps and shakes.



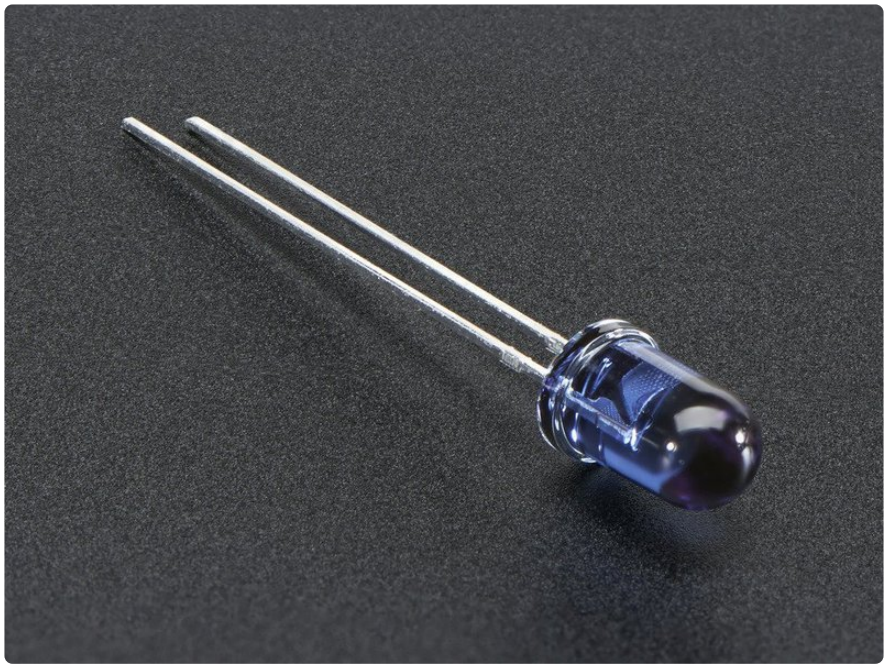
Small Piezo Buzzer

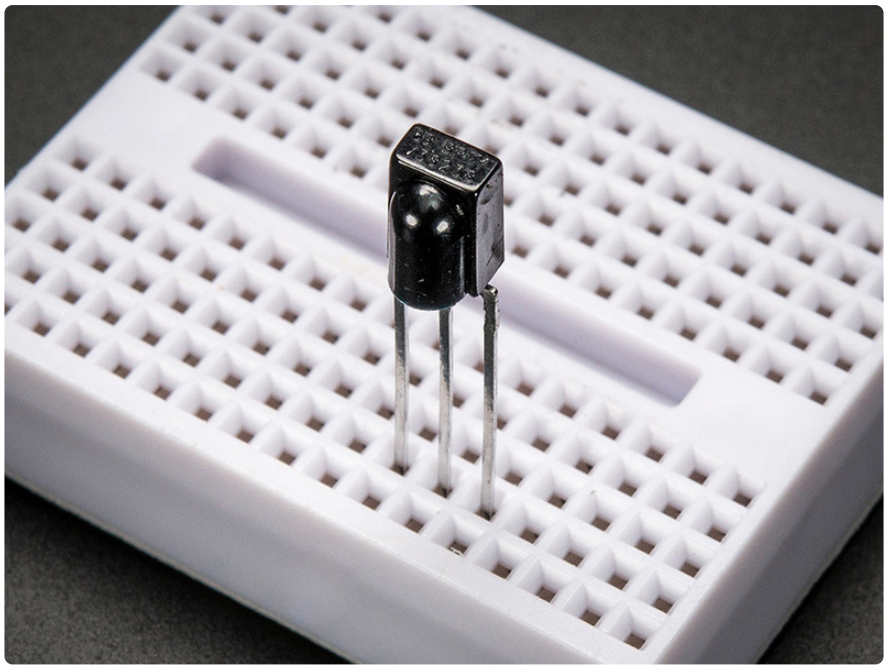
This tiny enclosed piezo element is what gives the Adafriend it's little voice! Be it singing a well known tune when happy, wimpering or grumbling, or even to let you know it sensed a tap, this little disk lets the Adafriend beep.



IR LED and Reciever

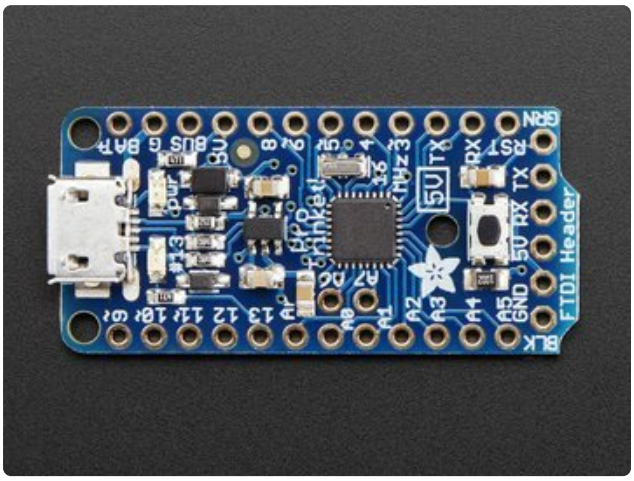
With an empty side of the Adafriend's cube-shaped body that was begging for something special, I decided to embed an IR LED and receiver to allow for communication between cubes, devious device meddling, universal off switch style, or to interact with remotes. The hardware is there, make the cube your own and hack in some extra functionality! I'd love to hear what you come up with!



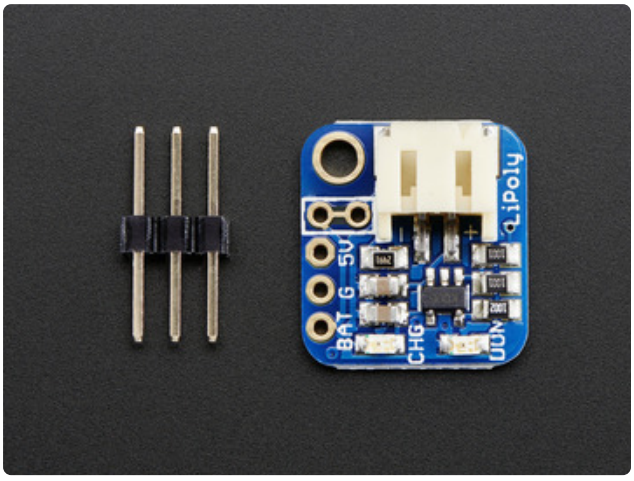


Parts List

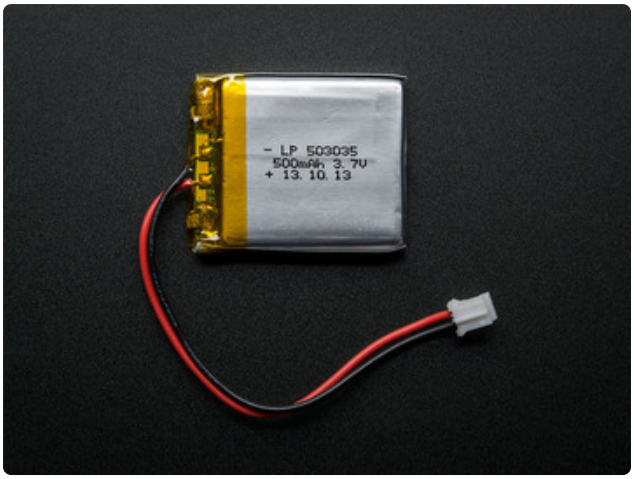
In summary, to build an Adafriend virtual pet cube you will need:



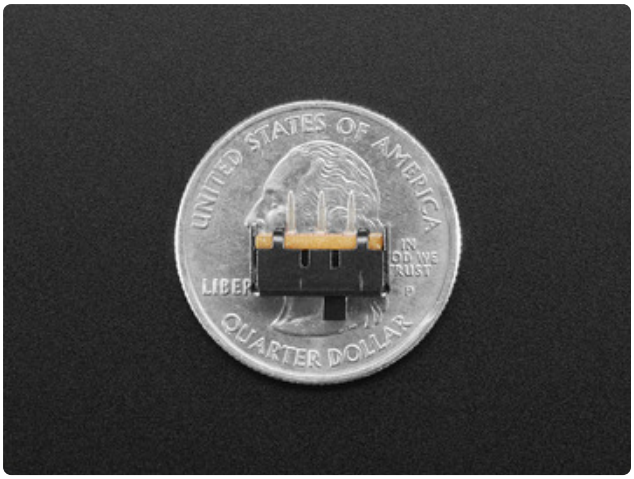
Adafruit Pro Trinket - 5V 16MHz (http://adafru.it/2000)



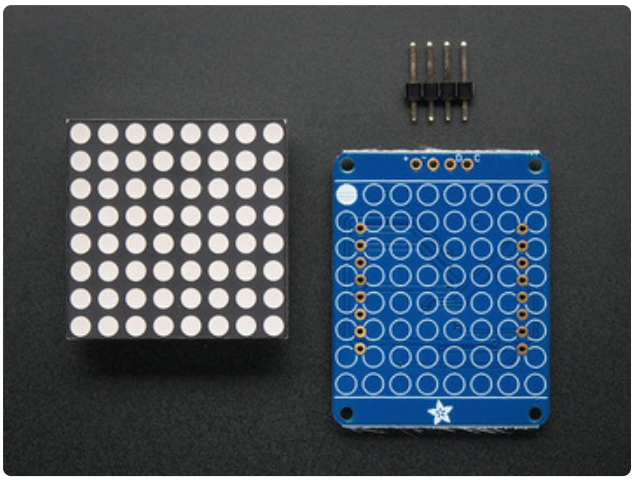
Adafruit Pro Trinket LiIon/LiPoly Backpack Add-On (http://adafru.it/2124)



Lithium Ion Polymer Battery - 3.7v 500mAh (http://adafru.it/1578)

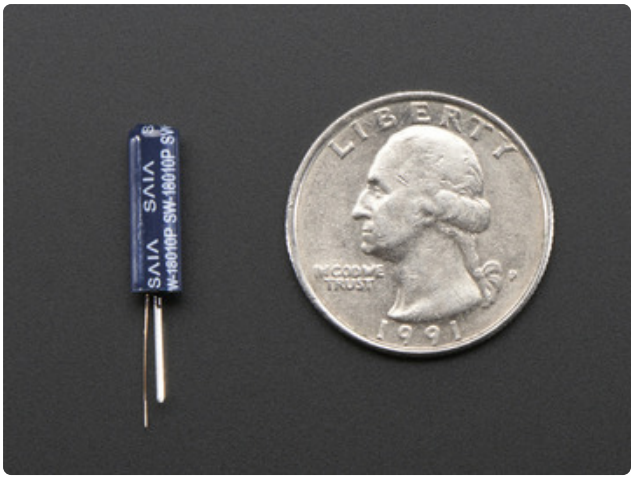


Breadboard-friendly SPDT Slide Switch (http://adafru.it/805)

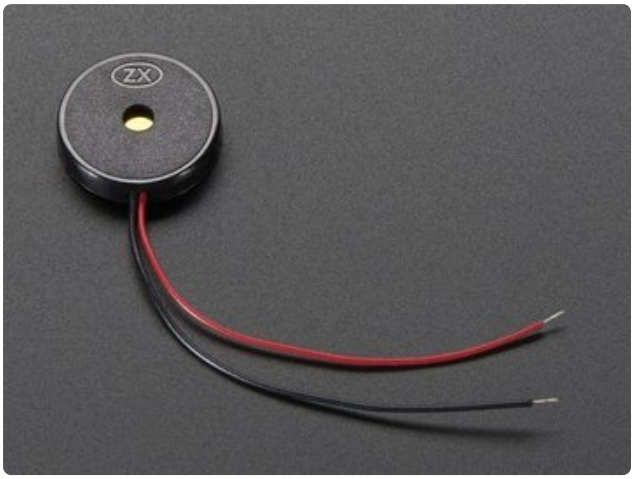


Small 1.2" 8x8 LED Matrix w/I2C Backpack - Pure Green (http://adafru.it/1632)

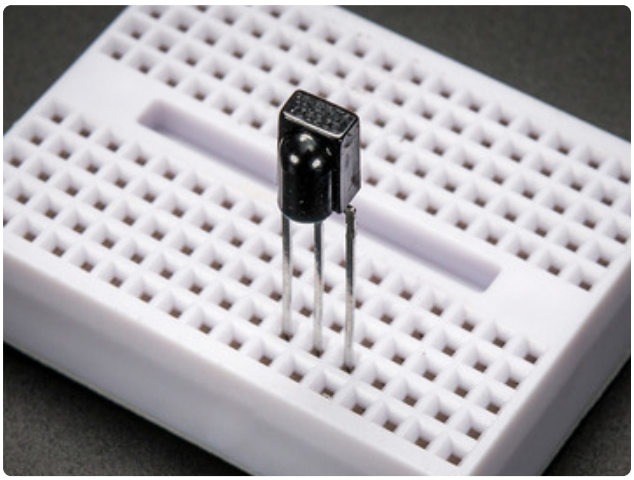
In the color of your choice (green linked here)



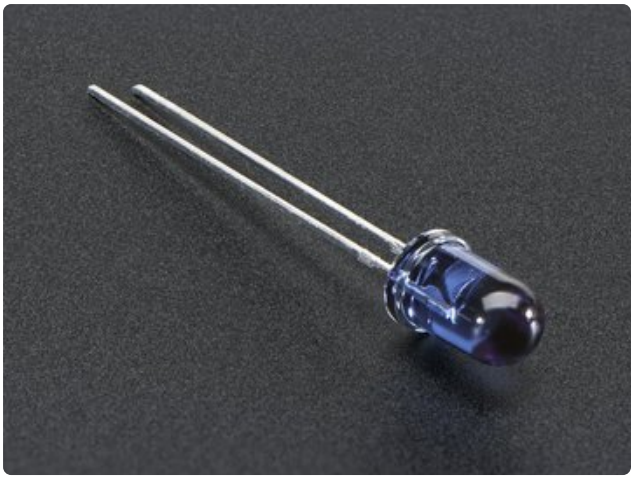
Fast Vibration Sensor Switch (Easy to trigger) (http://adafru.it/1766)



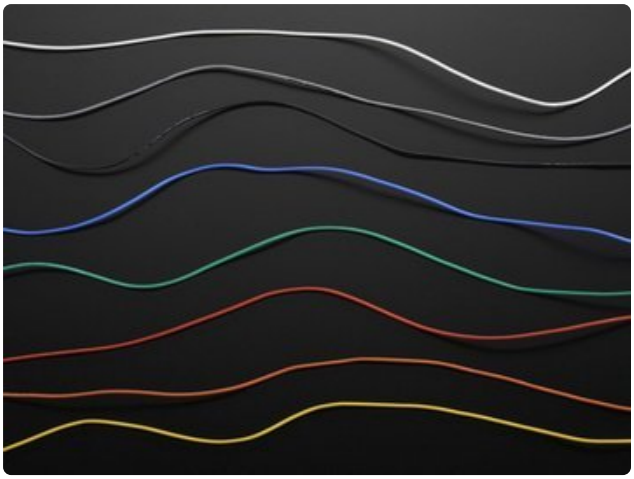
Small Enclosed Piezo w/Wires (http://adafru.it/1740)



IR (Infrared) Receiver Sensor - TSOP38238 (http://adafru.it/157)

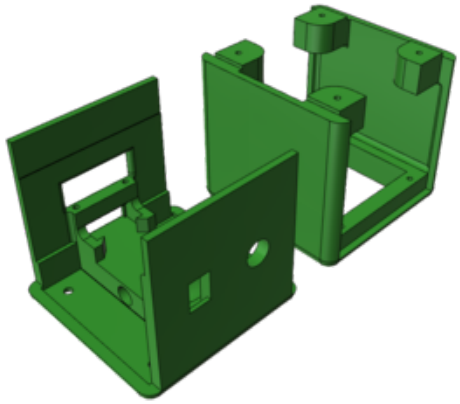


Super-bright 5mm IR LED - 940nm (http://adafru.it/387)



Silicone Cover Stranded-Core Wire - 26AWG in Various Colors (http://adafru.it/1970)

Non-Adafruit Parts



AdafriendTop.stl + AdafruitBottom.stl x1

Files found below!

Adafriend_STL_Files.zip

https://adafru.it/yEX

4mm M2 machine screws x7

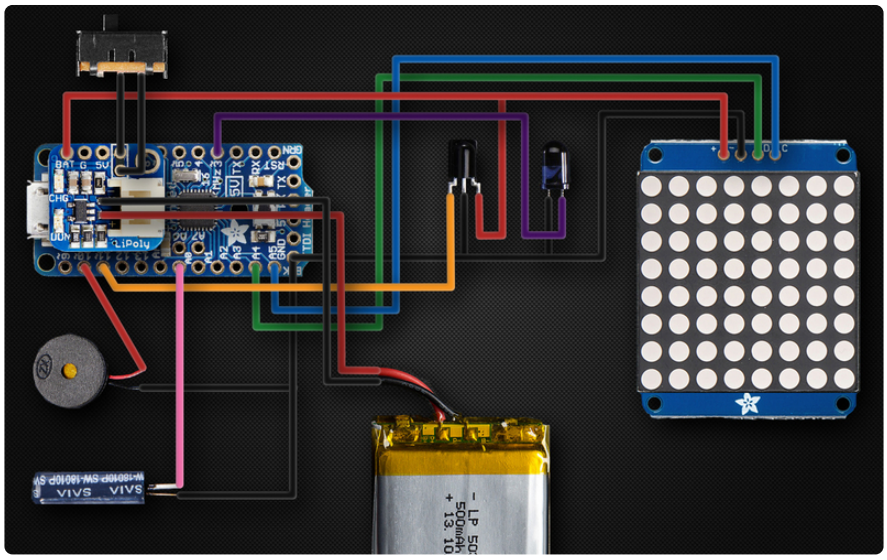
5mm M2 machine screws x4

With these parts and tools assembled, we can move on to building an Adafriend!

Wiring and Assembly

Circuit Connections

To build a little Adafriend cube, we will follow this (rather complicated but color coded) circuit diagram:

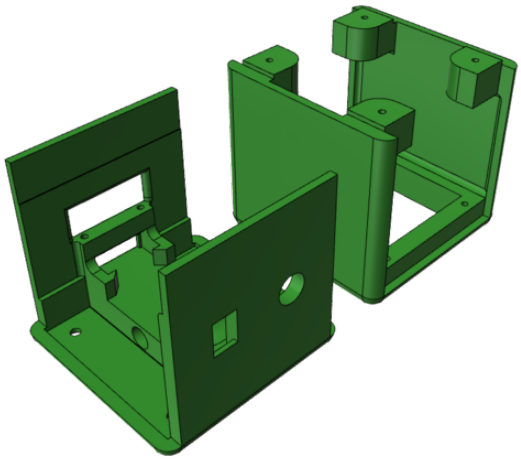


The circuit for the Adafriend runs on a Pro Trinket 5v with a lipo backpack that allows the user to charge the internal battery and switch it off with a slide switch. The vibration sensor is connected to analog pin A0, and the buzzer to D10, and the IR receiver and LED are wired to their digital pins too, D11 and D3 respectively, as well as power and ground. Finally, the LED matrix is wired to power and ground and the I2C lines, SCL and SDA.

Assembly Instructions

3D Printing

The first order of business in building an Aafriend is to 3D print its sturdy shell. I highly recommend a 3D printing service for this, or a resin printer if you have one, as the tolerances for such a complex little guy are quite tight and the model requires a few overhangs. I used Sculpteo (https://adafru.it/onC), but Shapeways (https://adafru.it/aVZ) or any other service would be just fine too. To 3D print the models yourself, download the stl files here, and print with your highest quality settings with the best support material option for the job. Really these models are designed for only the highest quality printers, and a service is ideal.



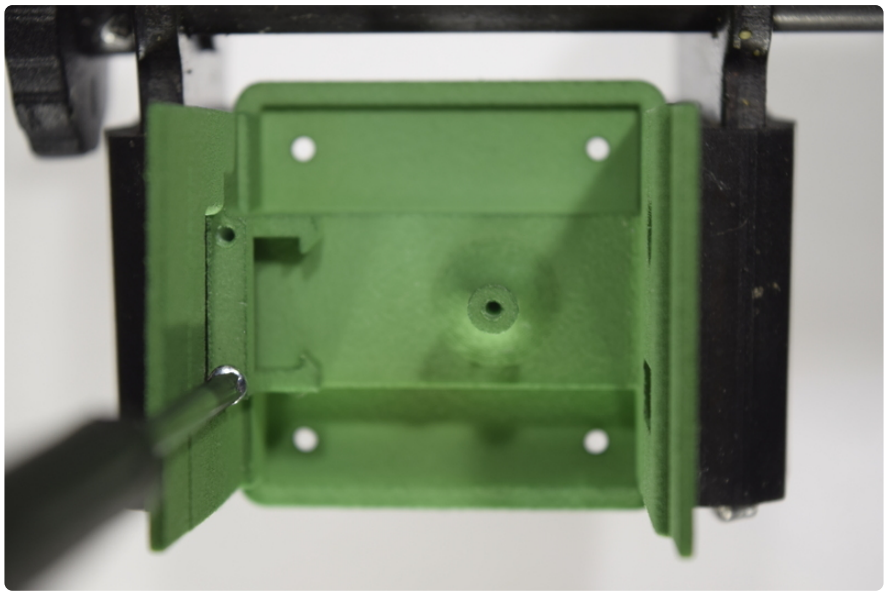
Adafriend STL Files

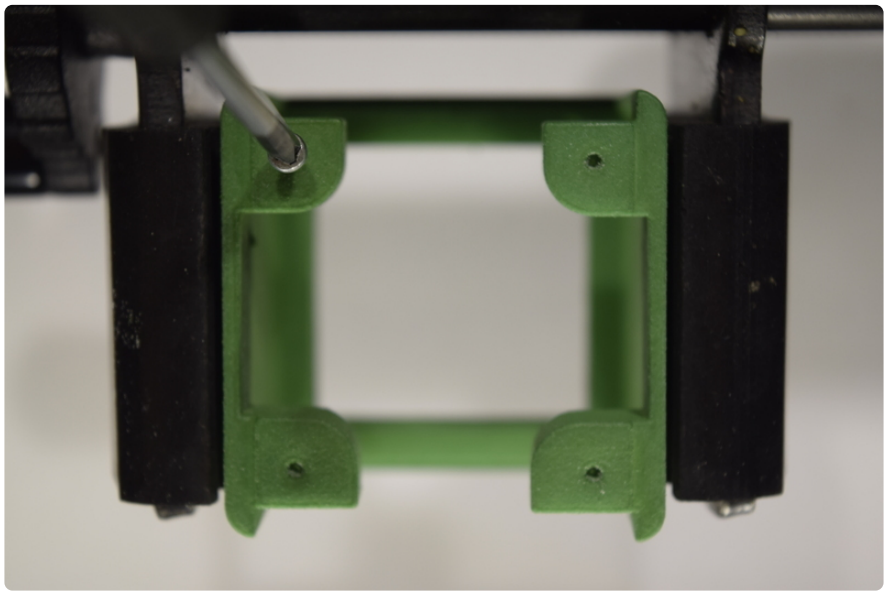
https://adafru.it/y5E

Tap Screw Holes

Once your models are printed, clean them up, and then tap all screw holes by carefully aligning and screwing in a 5mm M2 screw into each hole, backing up a bit with every twist tighter. Make sure to tap all 11 holes!

If you order your models online, make sure to clean out the holes of any excess material first. Tweezers or light taps work nicely.



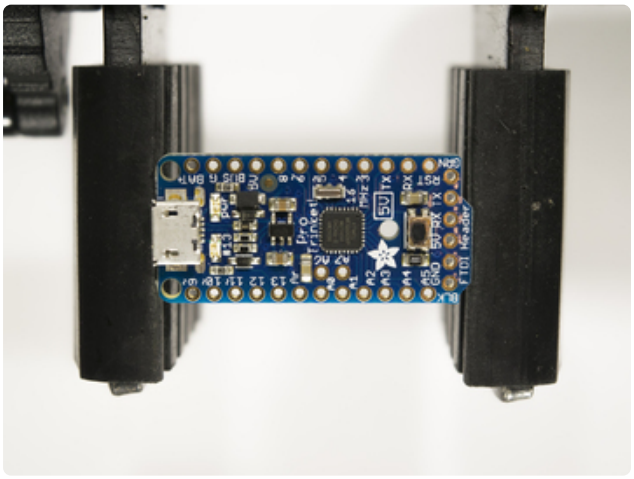


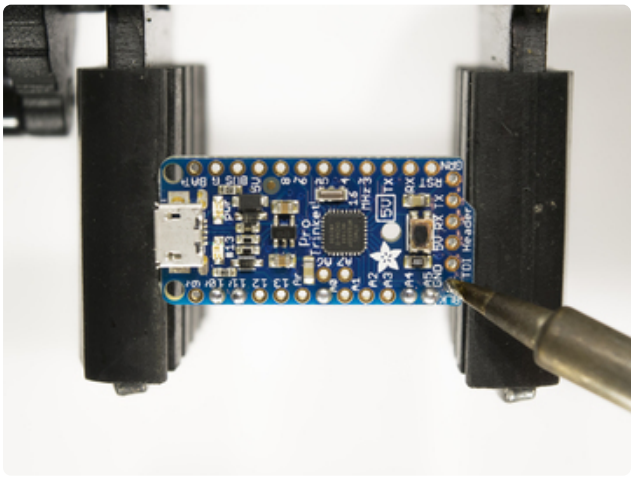
Assembly

With your 3D prints ready, we can move on with the build!

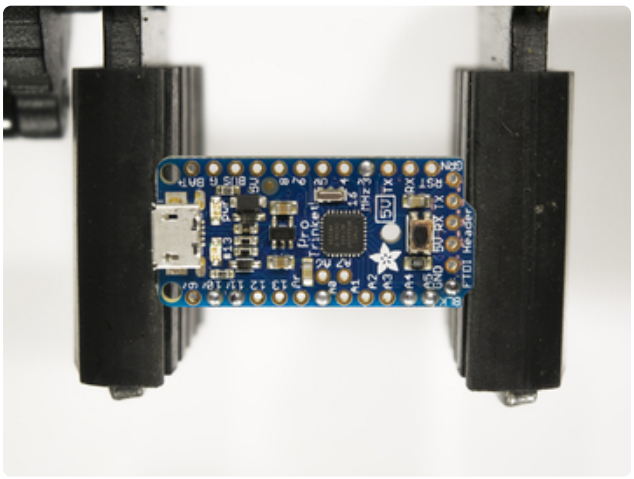
Preparing Components

Pro Trinket

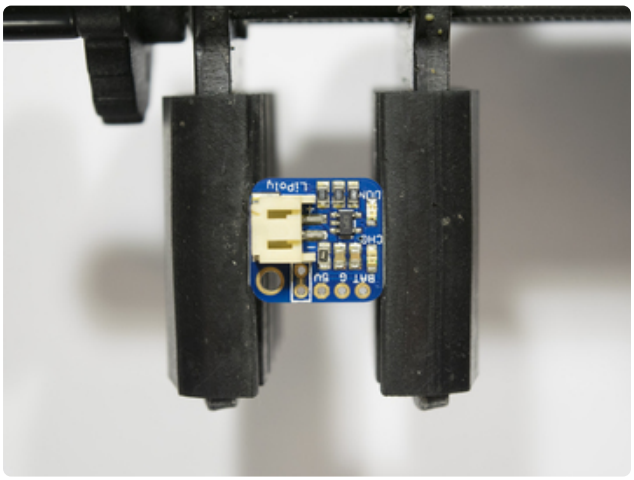


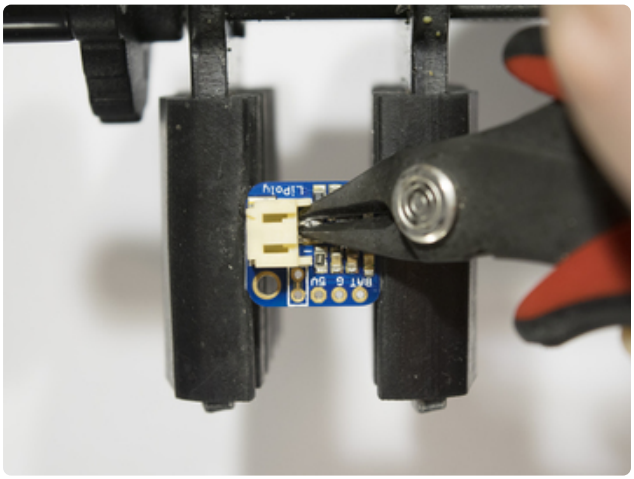


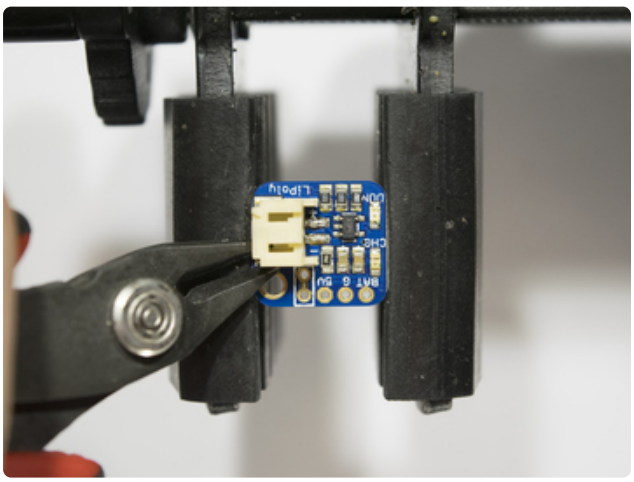
First, secure your Pro Trinket in a vice or on a safe work surface. Then, fill pins D10, D11, A0, A4, A5, GND, and D3 with solder as the photo shows. Your Pro Trinket is now ready for wiring, set it aside.



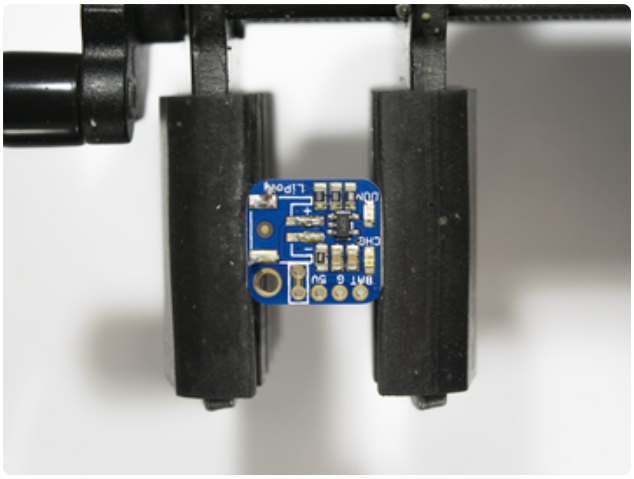
LIPO Backpack

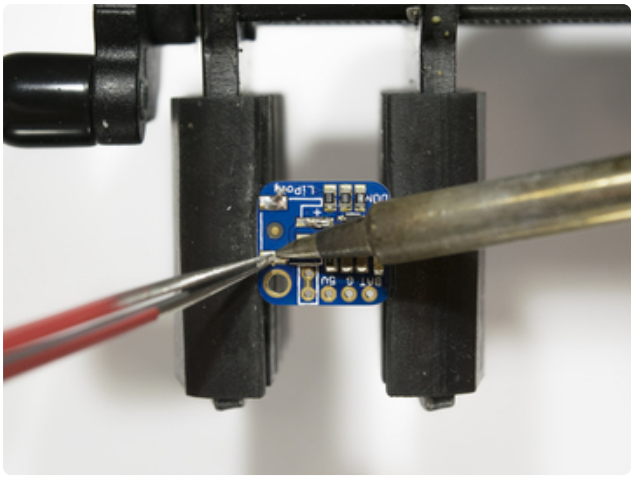


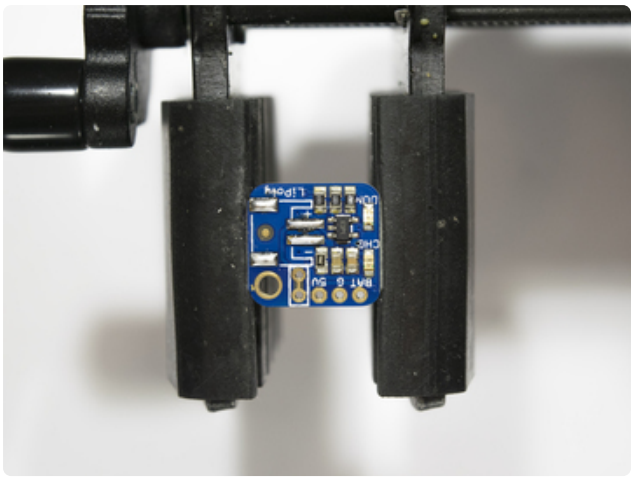


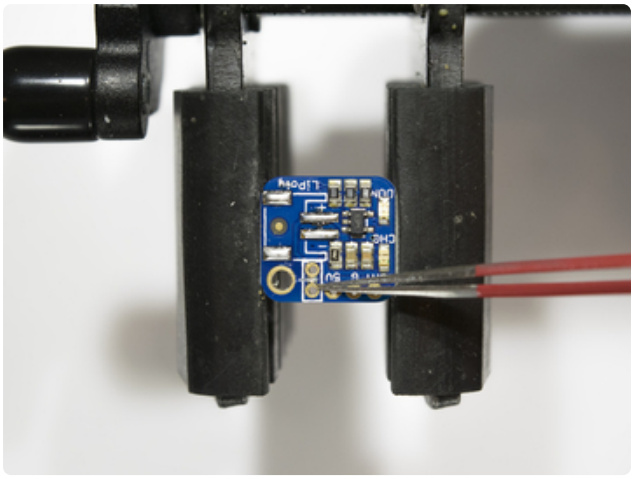


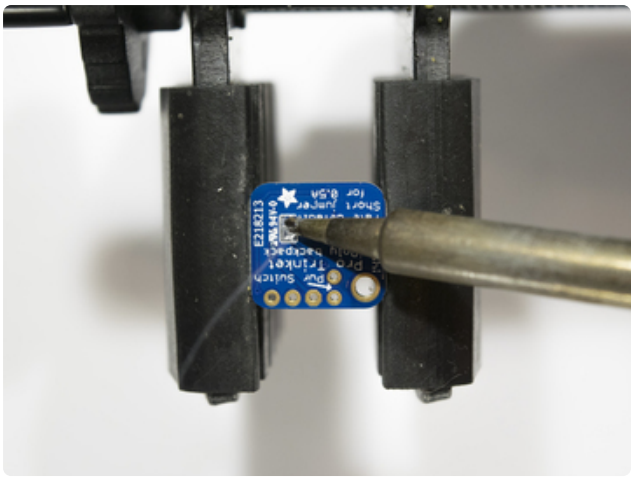
Next we will prepare the LIPO Backpack module. First, using flush cutters, cut the soldered pins of the JST connector, behind and in front. The plug takes up too much space inside the cube. With the connector removed, reflow the solder joints and remove the remains of the pins. Add more solder if necessary to clean up the pads. Then cut the small jumper trace between the two power switch holes pointed out with tweezers. Flip the board over and solder the jumper on the back to allow the cube to charge at the full 500ma speed. Next, fill the power switch holes with solder to prepare them for wires.

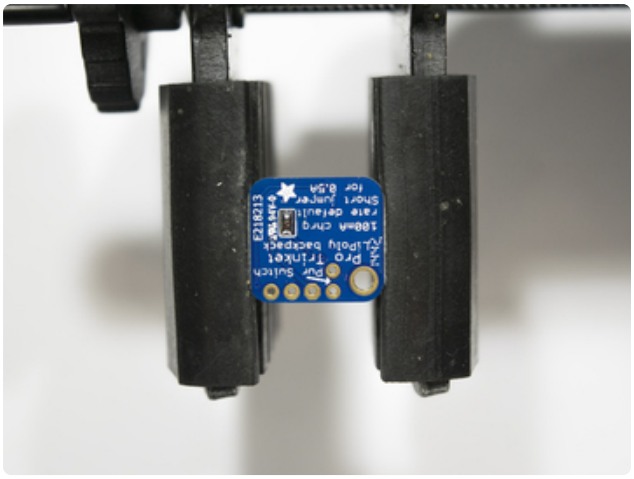


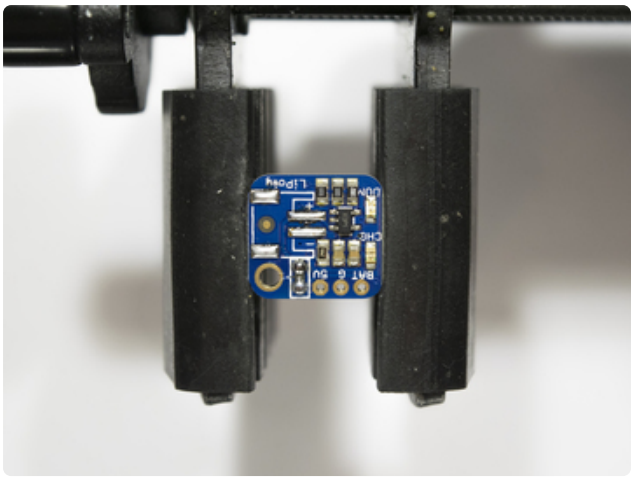


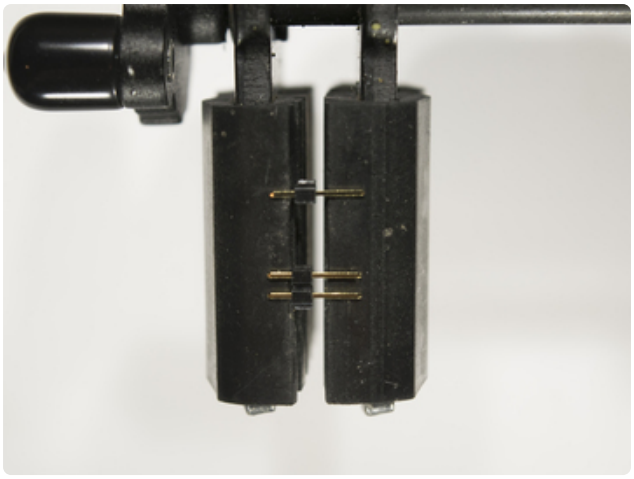


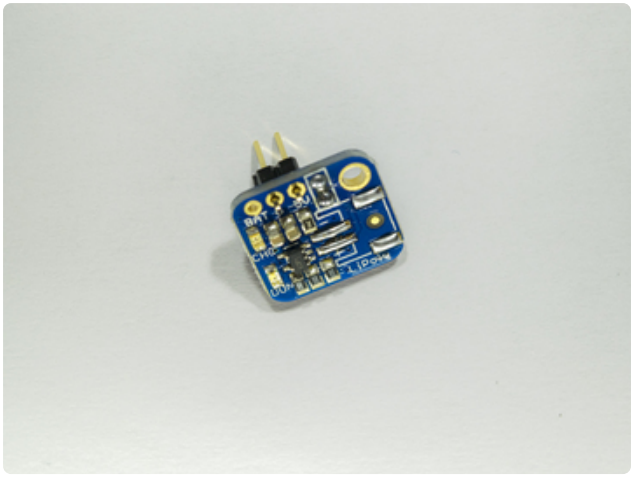


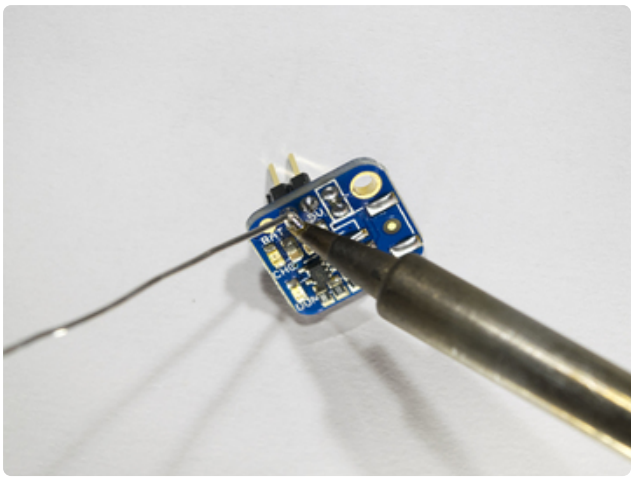




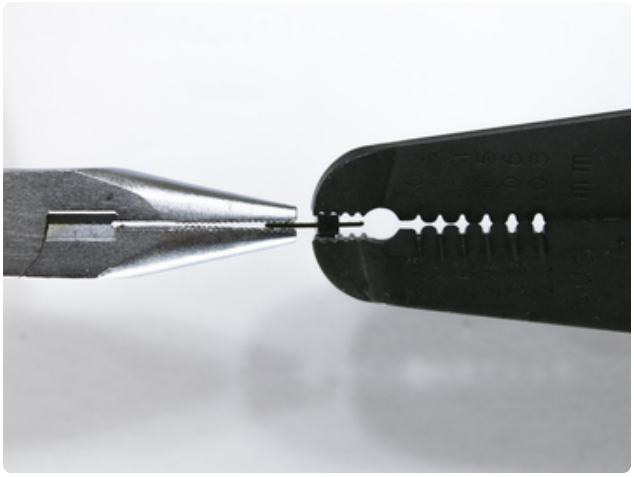


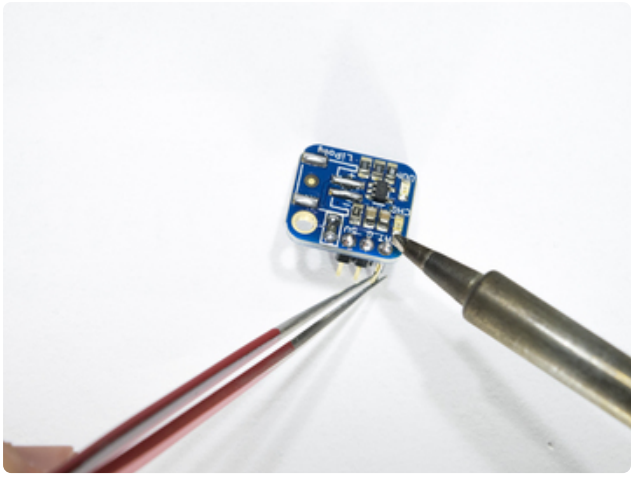




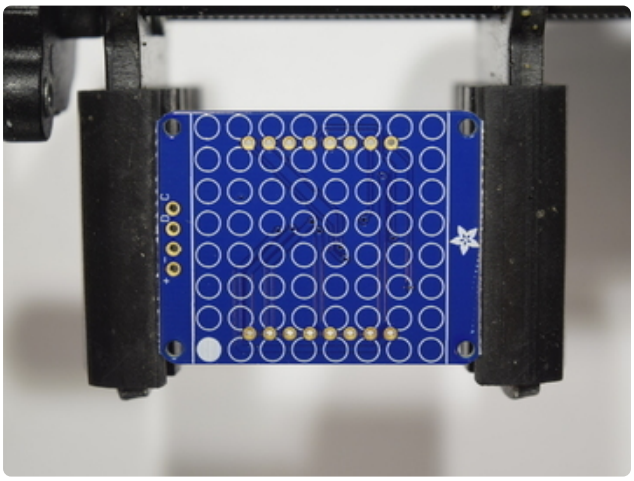


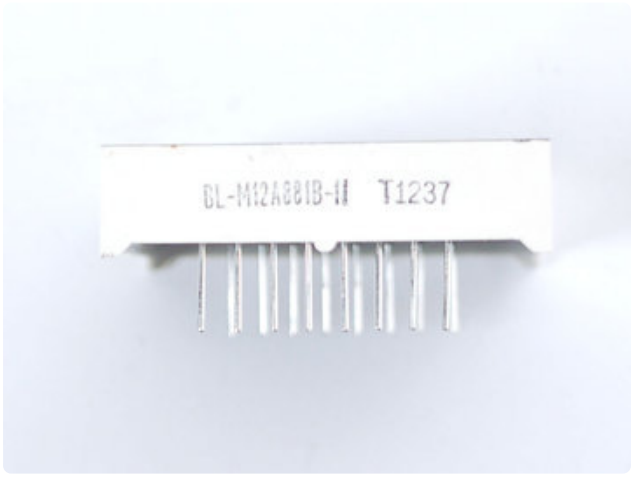
Gather 2 lengths of pin headers that come with the various parts of this project, one single pin, and a set of 2. Make sure they are short ones, not long ones. Fit the set of 2 into the 5v and G holes, align them, and solder. Then using pliers, remove the black plastic spacer from the single pin as it gets in the way of the screw we'll fit into the Pro Trinket. With the spacer removed, align it in the "Bat" hole of the module and solder as before. The LIPO backpack is prepped, set it aside.

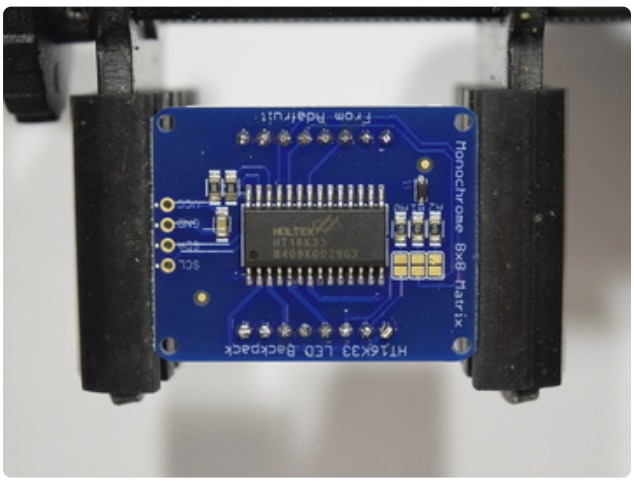




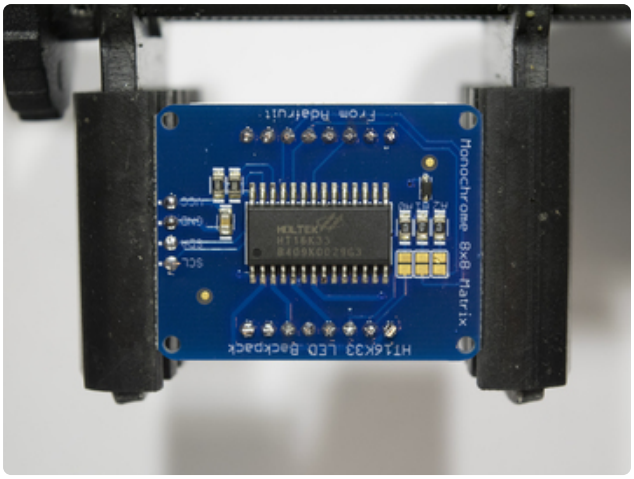
LED Matrix

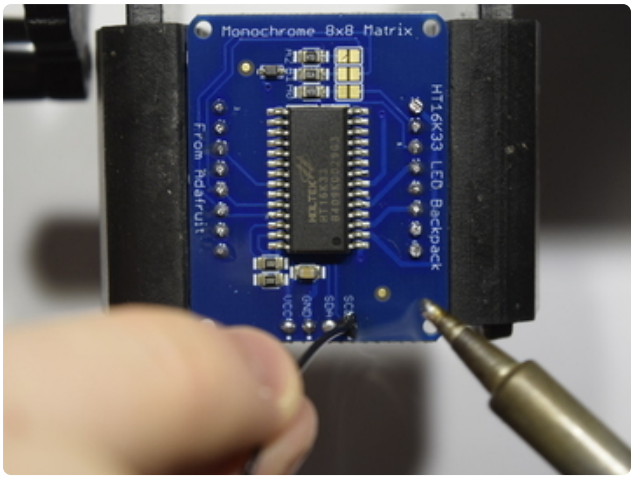


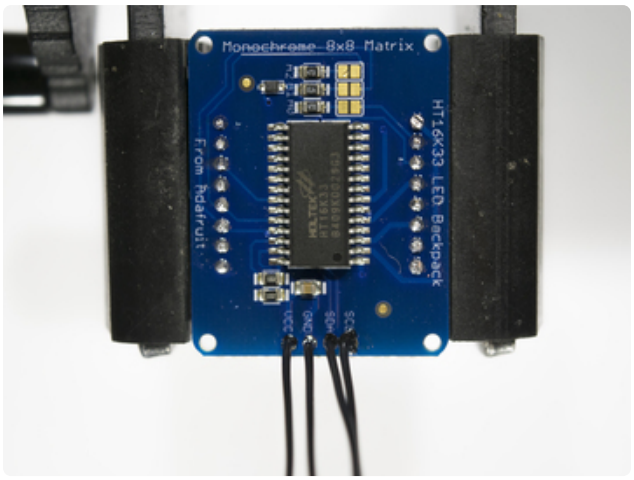




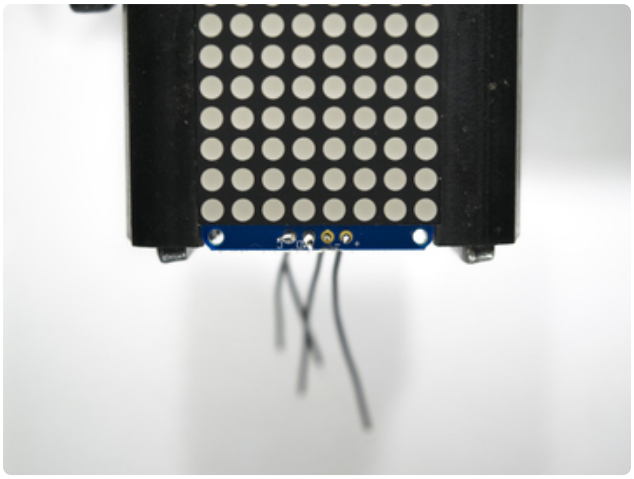
For the LED matrix, secure the backpack and identify the side of the pcb with a white dot. Bottom left shown here. Then find the side of the white LED matrix module with the printed text on it as shown. Insert the matrix into the pcb with the printed text on the same side as the white dot, and solder on the other side of the pcb. Then fill the 4 interface holes with solder.

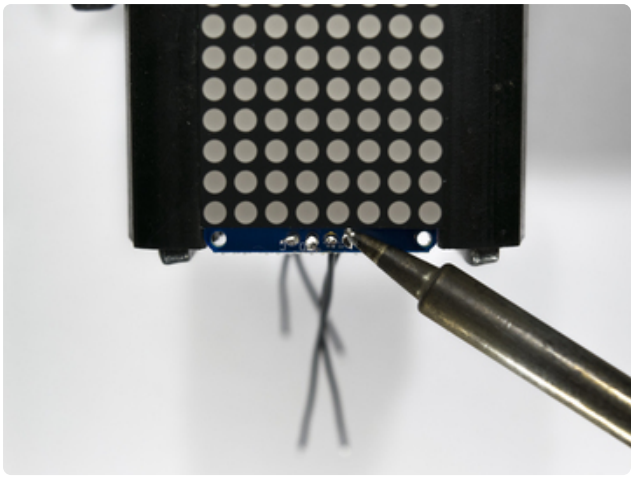

From Adafruit!
Monochrome 8x8 Matrix
HT16K33 LED Backpack

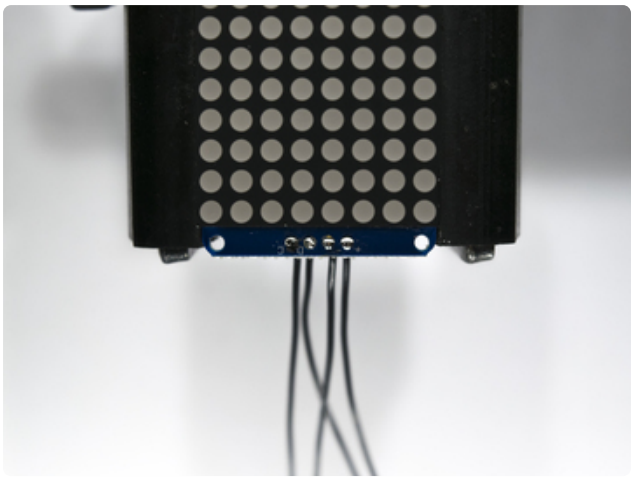




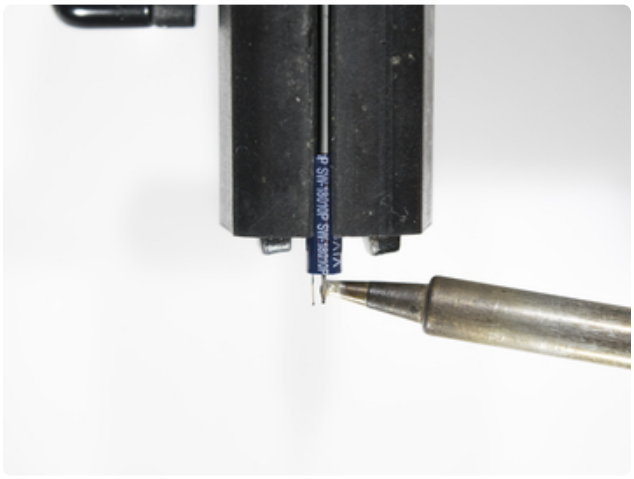
Strip and tin 4 approximately 3" lengths of thin wire, and heat the 4 interface holes on the matrix to insert the wires in the back of the matrix. Then add solder to the other side of the pcb for a strong connection. Cut any protrusions with flush cutters and reheat so the joints are as flat as possible. The matrix is prepared, set it aside.



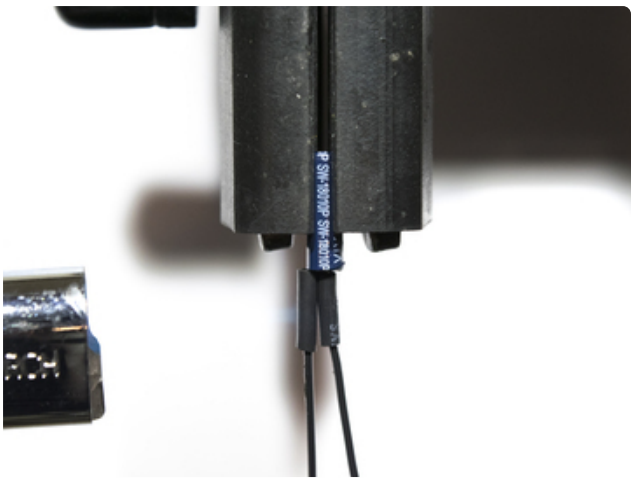




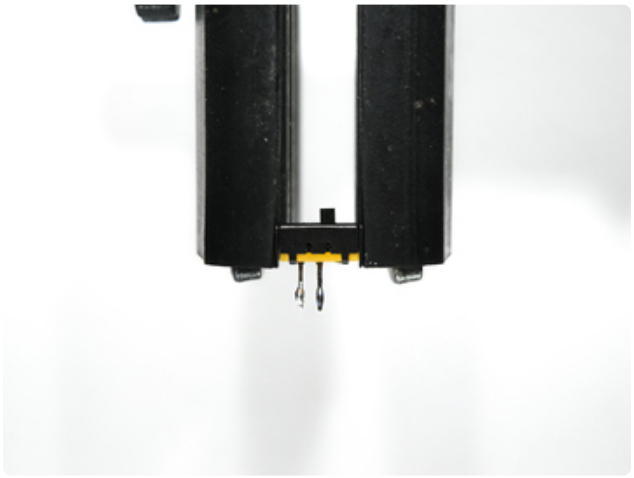
Vibration Sensor

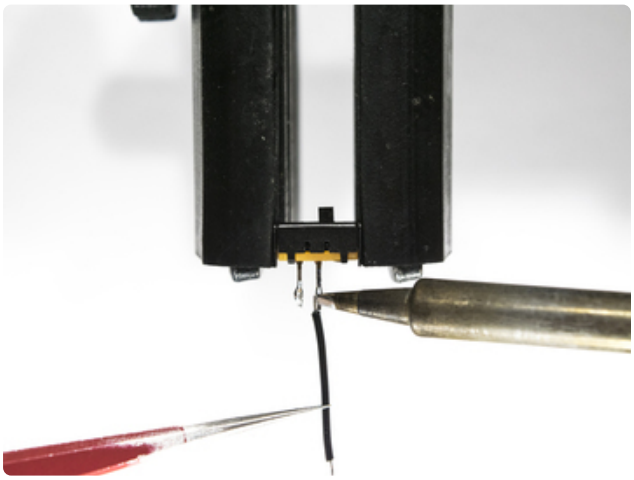


To prepare the vibration sensor, secure it in a vice and cut the leads to a short length as shown. Then tin the leads and cut and tin two ~1" lengths of wire and solder them to each lead. Shroud each connection in a short length of heat shrink tubing and shrink it with heat. Then cover both connections in another length of tubing and shrink that to prevent bending and breakgages. Set aside the prepped sensor.



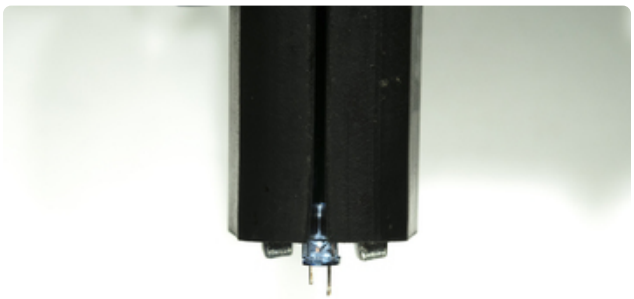
Slide Switch



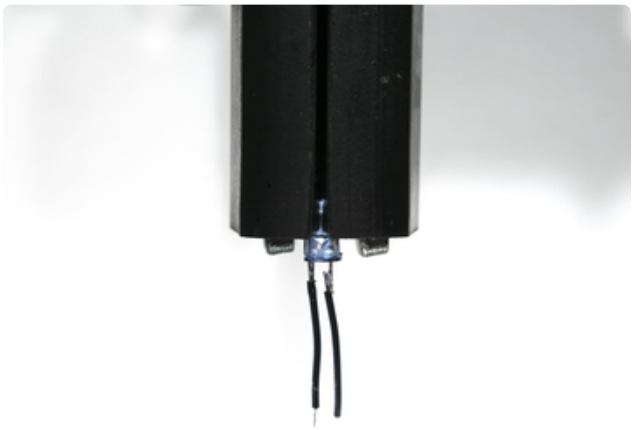


Secure the slide switch, and cut off one of the pins. Then tin the others, and solder two ~1.5" wires to the pins, and protect them with heat shrink tubing. The switch is prepped, set it aside.

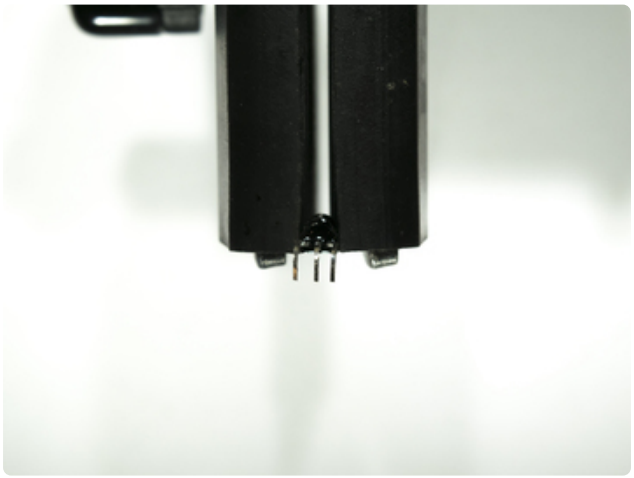
IR Led

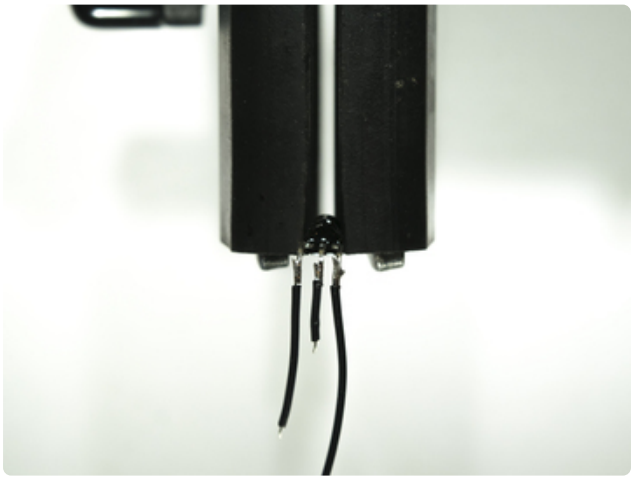


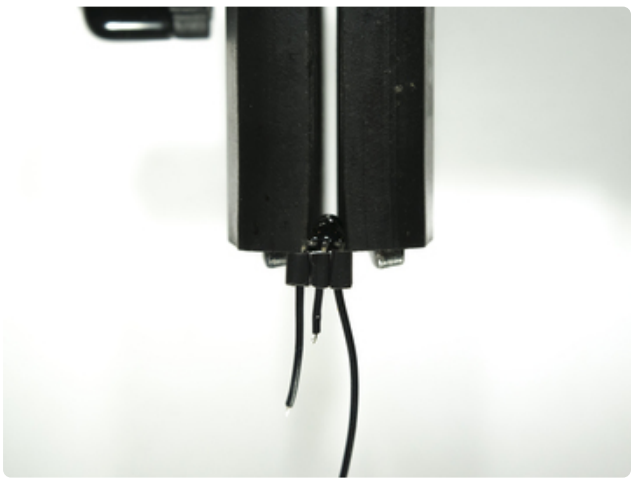
Secure the LED in a vice and trim and tin the connections, keeping track of + and - by leaving the longer lead longer. Then cut and trim two 1" legths of wire and solder them to the led. Secure the connections with heat shrink tubing. Set aside the prepped LED.



IR Receiver





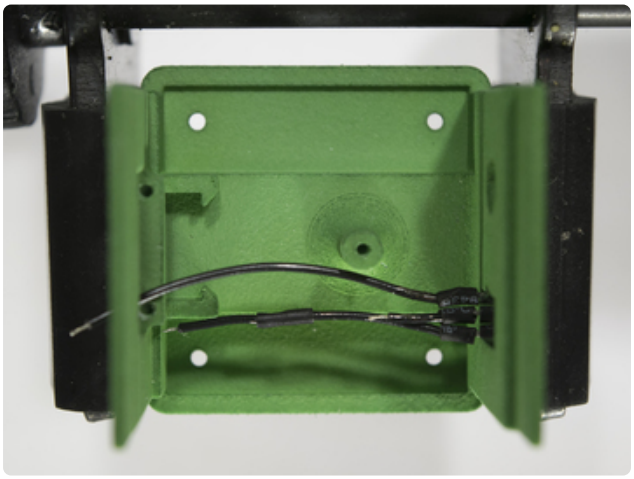


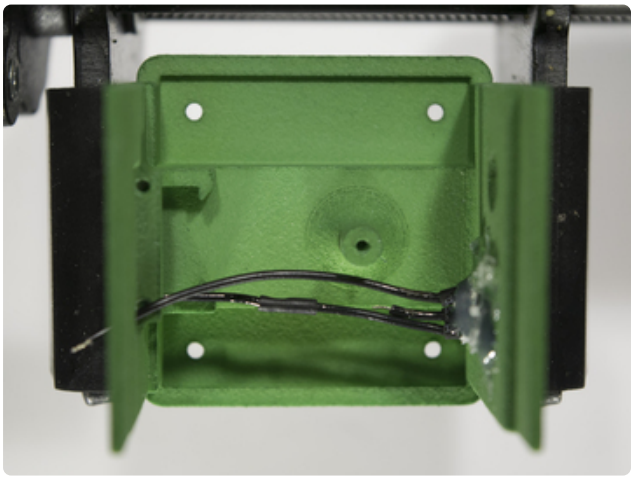
Secure the receiver in a vice, and bend the leads to a 90 degree angle towards the back of the sensor. Trim the leads short as shown and tin them. Then solder one 1", one .5", and one 2" wire to the leads as shown. Then heat shrink the connections and set aside the prepped sensor.

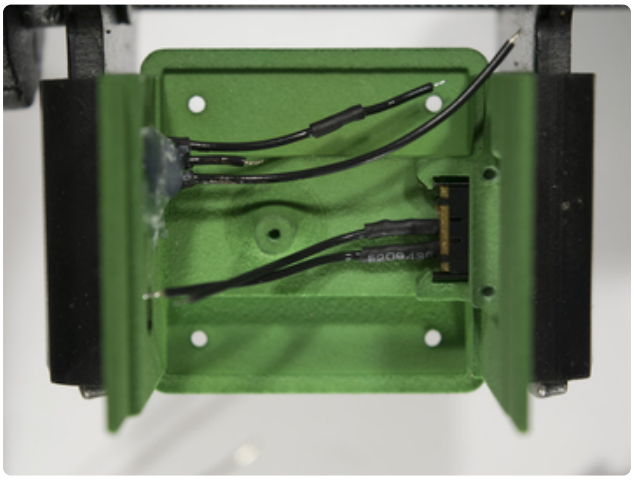
Final Assembly

With the individual parts prepared and wired, the cube can start to come together!

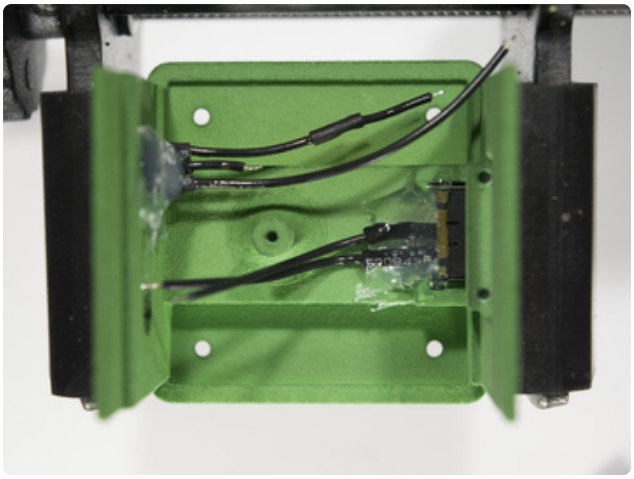
IR Receiver Power Switch Installation



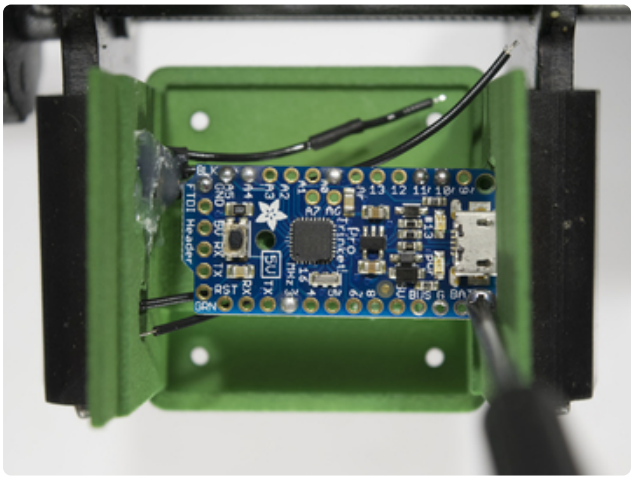


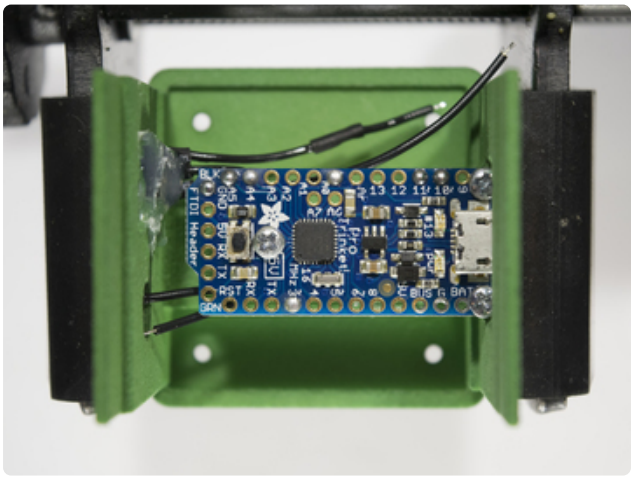


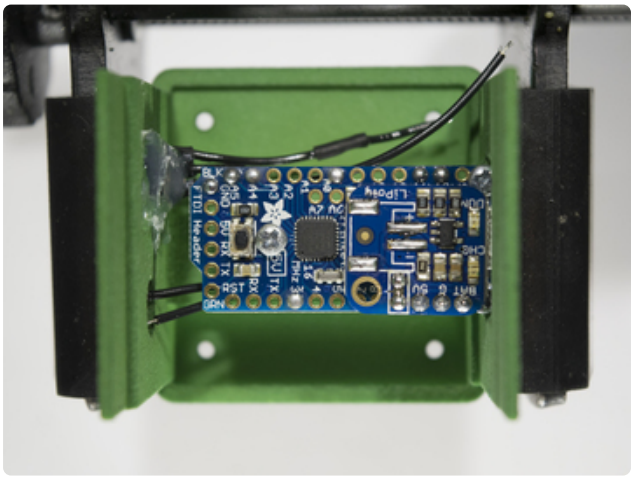
First, insert the IR receiver sensor into its slot in the wall of the bottom half of the cube. Hot glue it in place. then do the same with the power switch in its place at the bottom of the cube as shown.



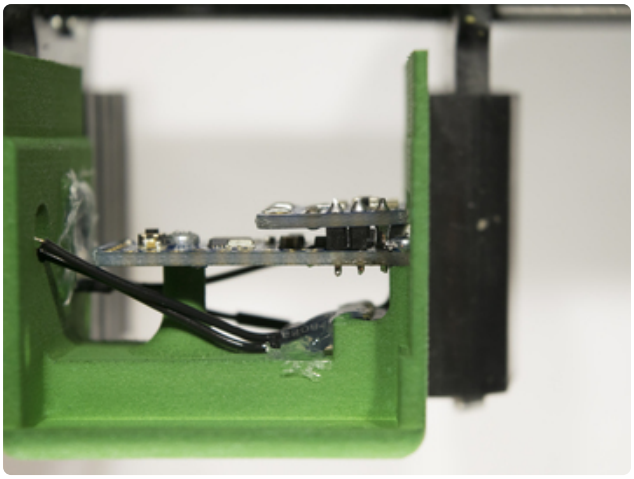
Pro Trinket, LIPOLY Backpack, and Power Switch Wiring

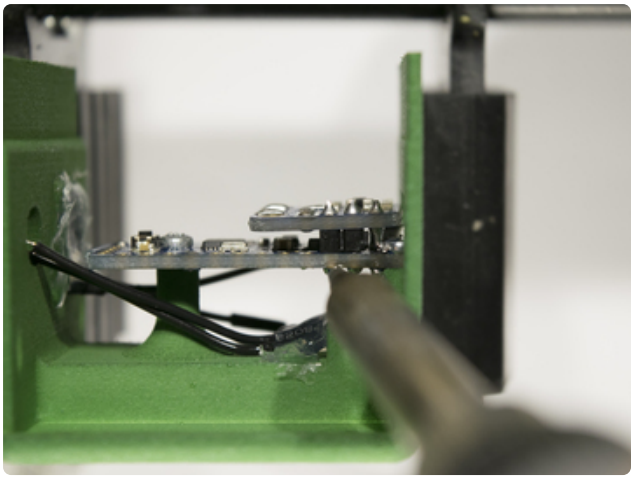


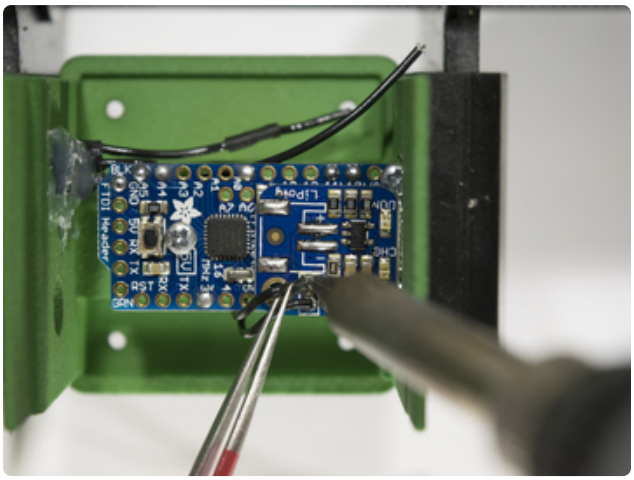




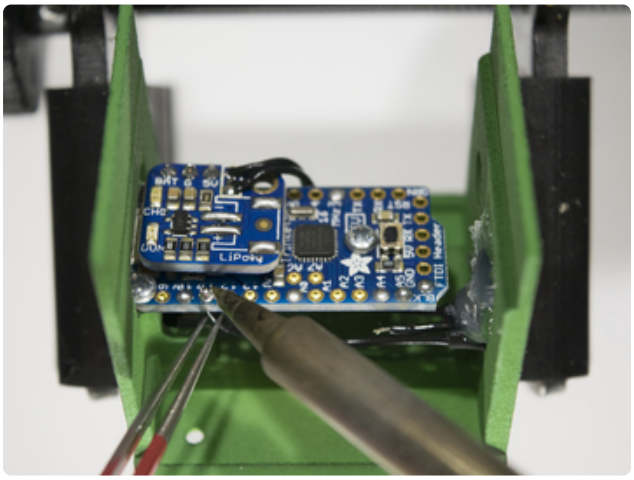
Screw the trinket to the case with 3x 4mm M2 machine screws. Then place the LIPOLY backpack in its designated holes as per the circuit diagram and photos and solder it in place. Then using tweezers, solder the two power switch wires to the two holes in the backpack.

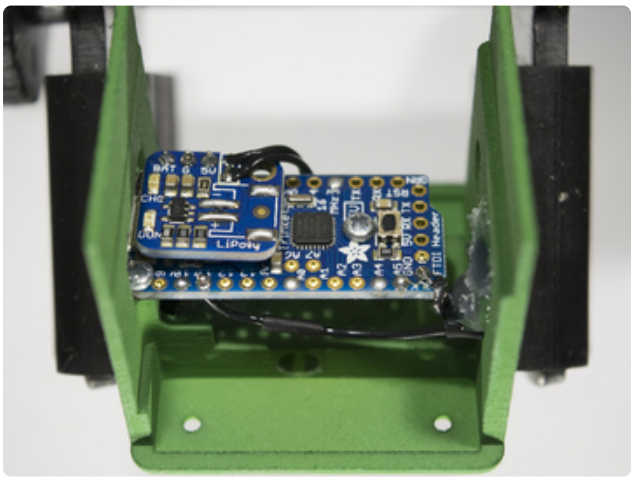


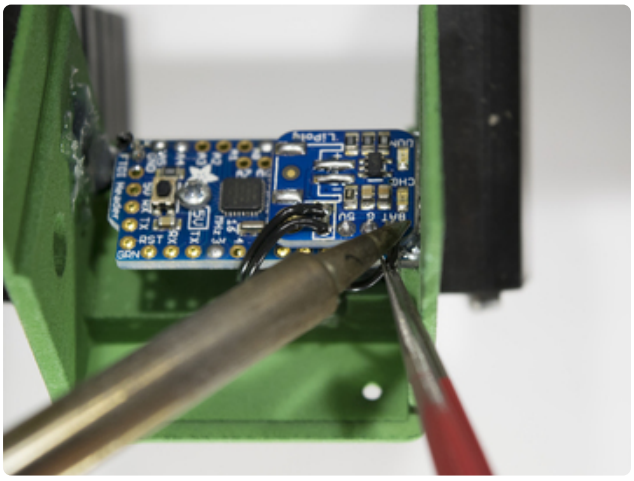



FTDI Header
GND
5V
RX
TX
RST
GRN
LilyPad

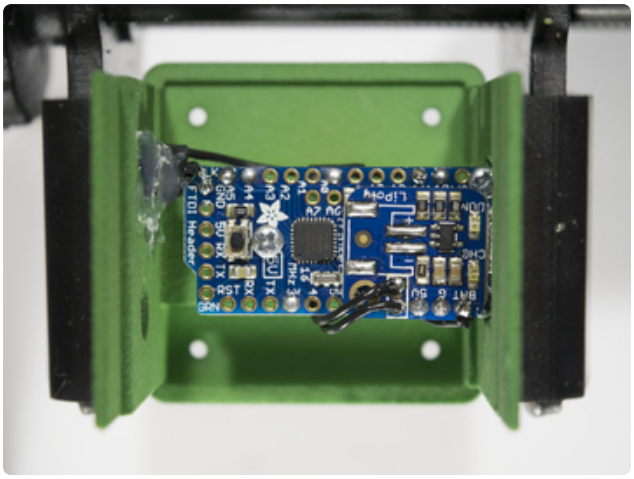
IR Receiver Wiring



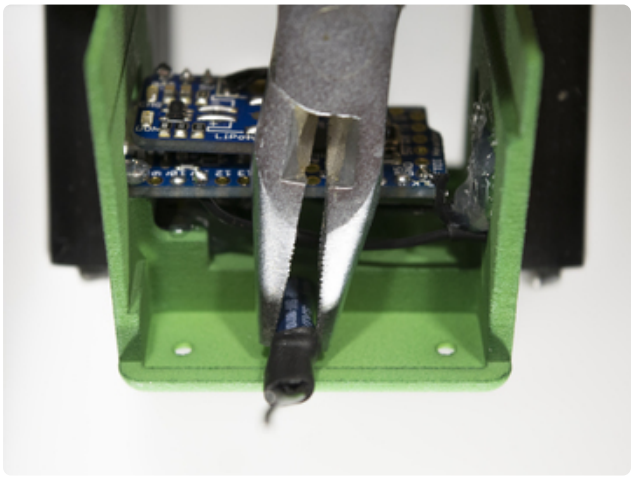


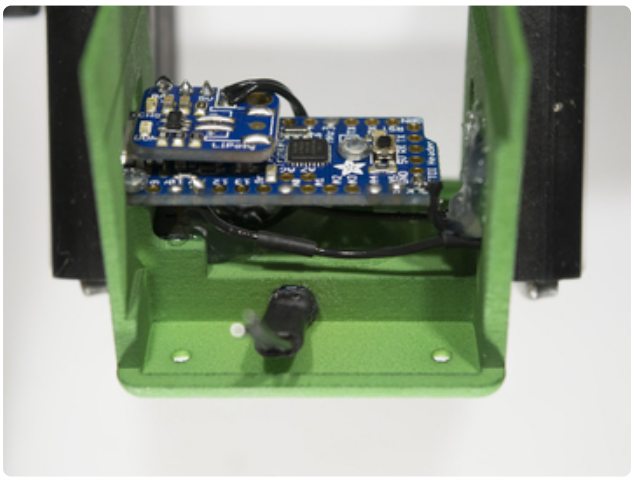


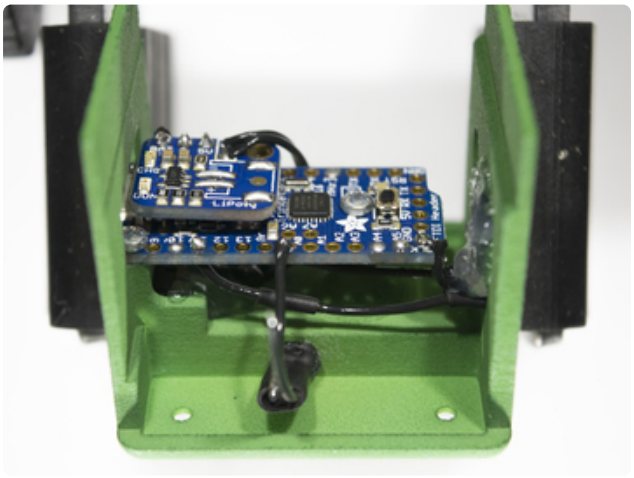
Solder the right (from the back) lead of the IR receiver to D11, the center lead to GND right next to it, and the left lead to Bat on the LIPOLY backpack. Check your work to see if it matches the last image.



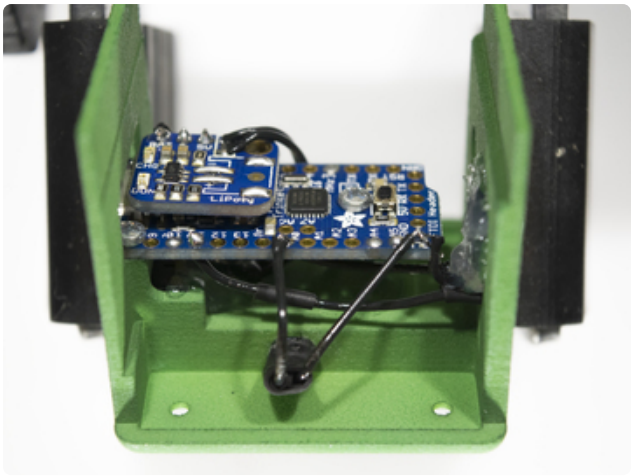
Vibration Sensor Installation and Wiring



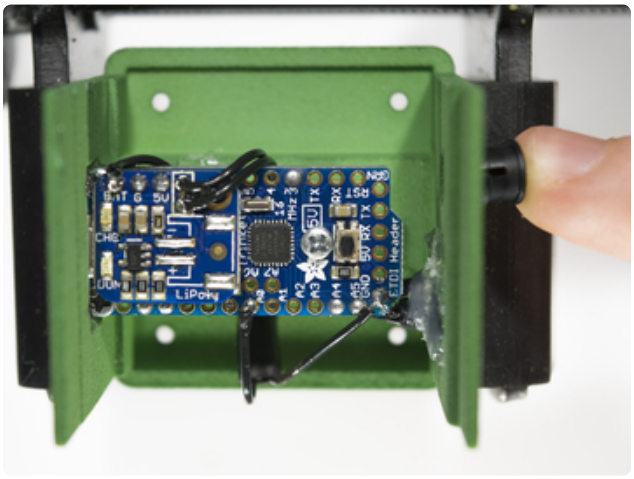


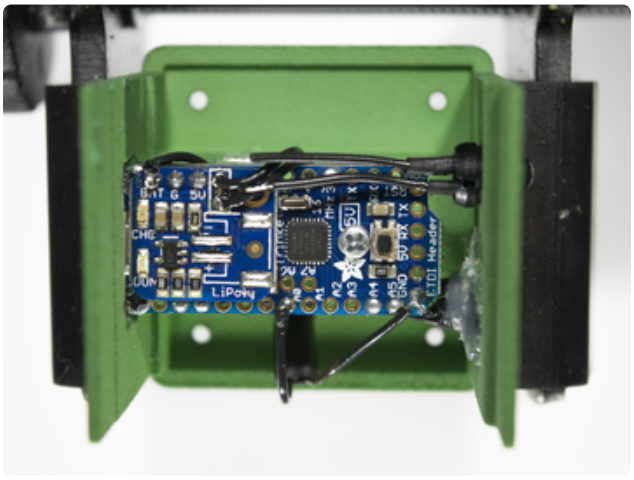


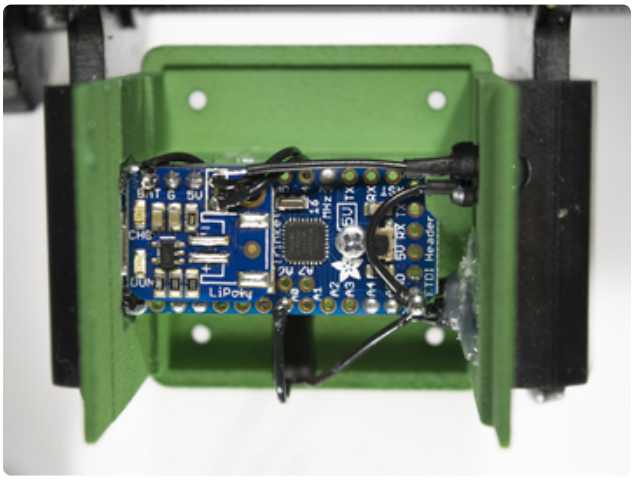
Using pliers, insert the vibration sensor into its anchor, and solder one wire to A0, and one to ground, it doesn't matter which.



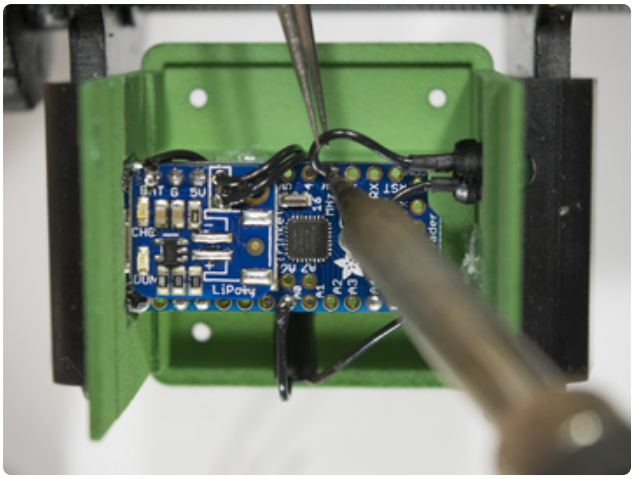
IR LED Installation and Wiring



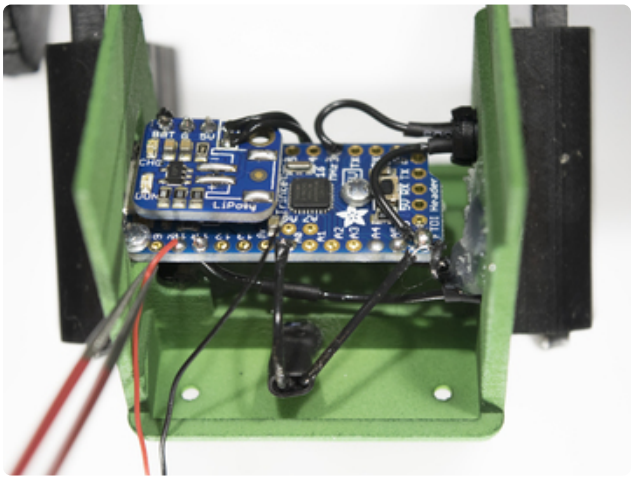


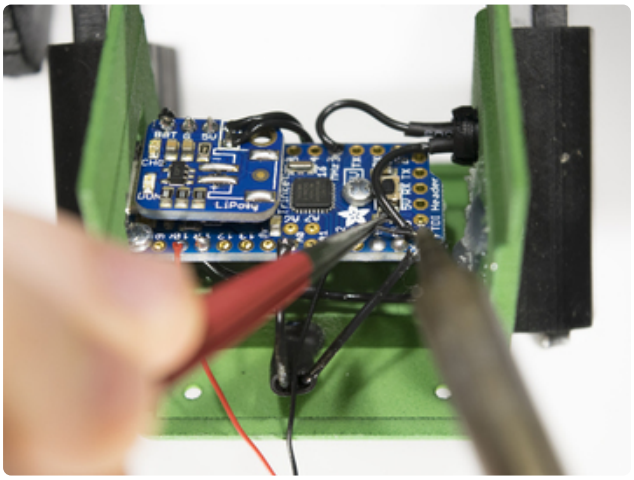


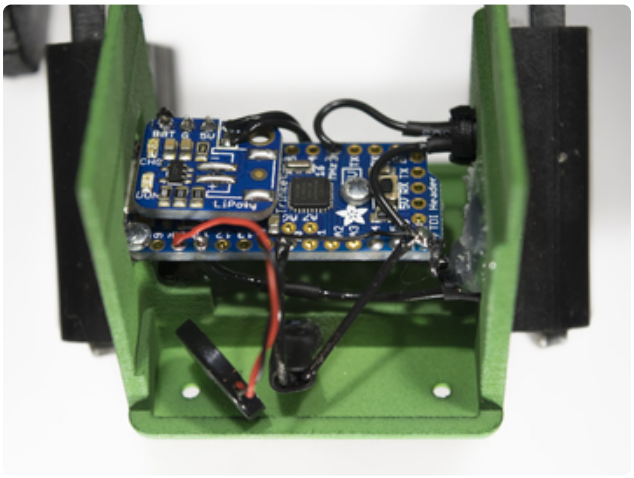
Push the 5mm LED holder into the IR LED hole next to the IR receiver, then insert the ir LED from inside, pushing it outwards. Solder the - leg, or shorter one, to GND, and the longer + leg to D3.


BAT G 5V
CHG
LiPoly
2V 4V
A1
A2
A3
RST RX

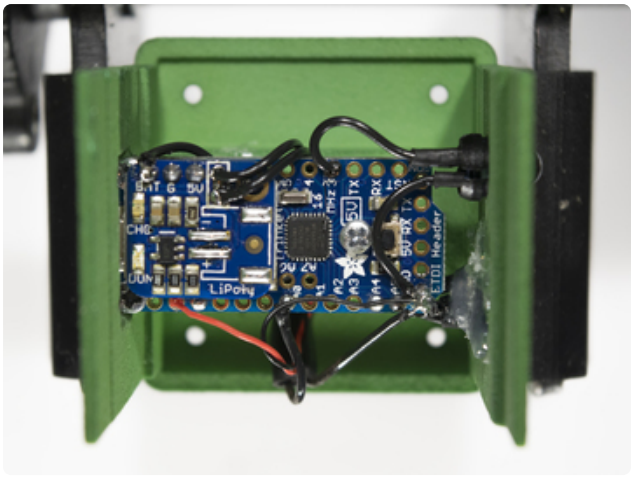
Piezo Buzzer Installation and Wiring



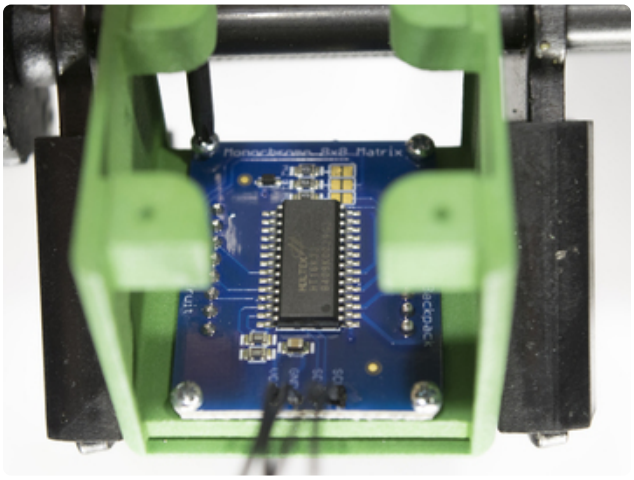


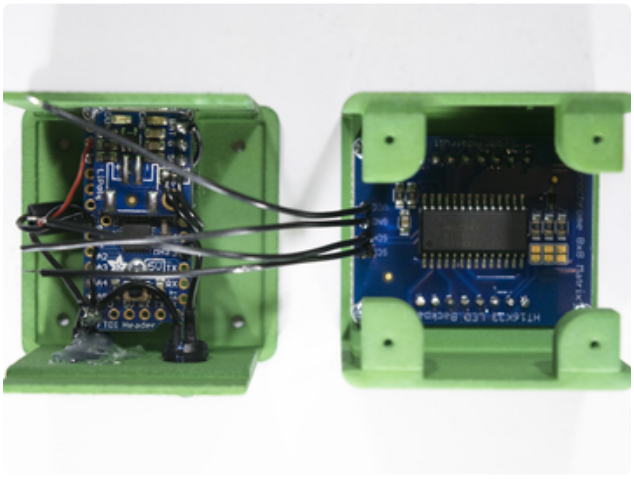


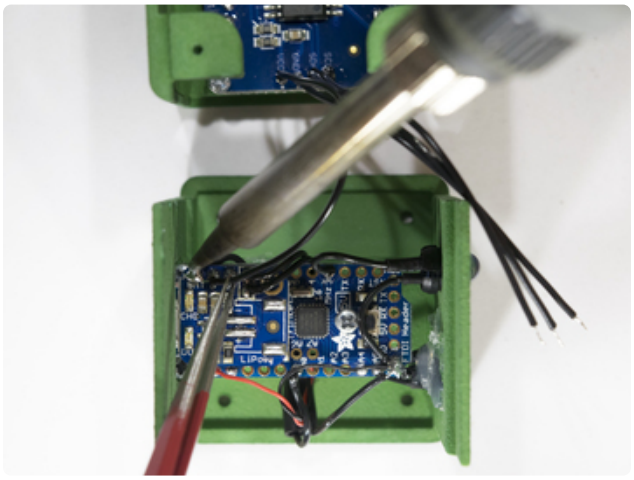
Solder the piezo buzzer's red wire to D10 and the black wire to GND. Twist the wires together and tuck the black head of the buzzer under the Pro Trinket to keep it out of the way. Check your work to see if it matches the last image.


LiPoly

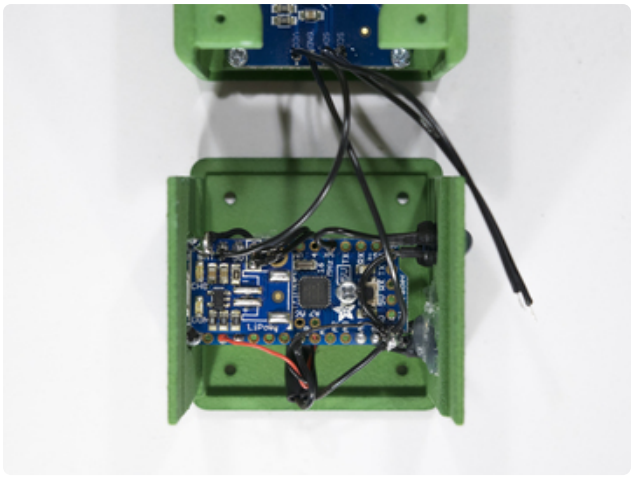
LED Matrix Installation and Wiring

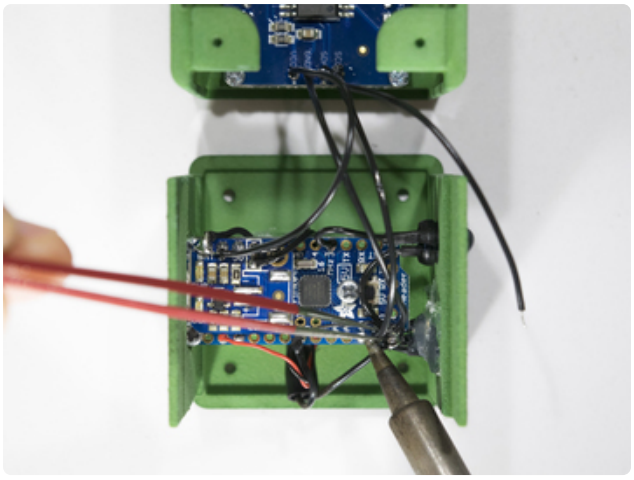


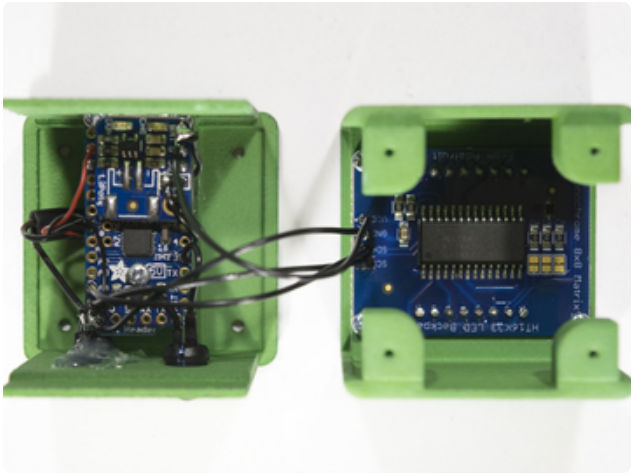


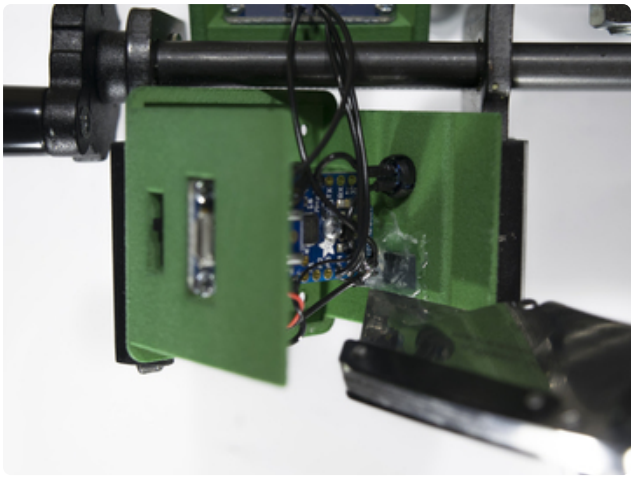


Fit the LED matrix into the top half of the case, and screw it in place with 4x 4mm M2 machine screws. Bring both halves together as shown to solder them together. Solder the matrix's VCC wire to Bat on the LIPOLY backpack. Solder the matrix's GND wire to Gnd on the Pro Trinket. Solder the matrix's SDA pin to A4 on the Pro Trinket, and the matrix's SCL pin to A5 on the Pro Trinket. Check your work with the last image.

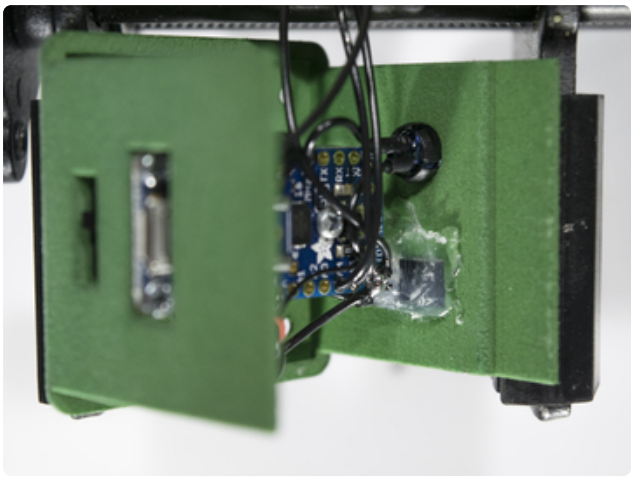


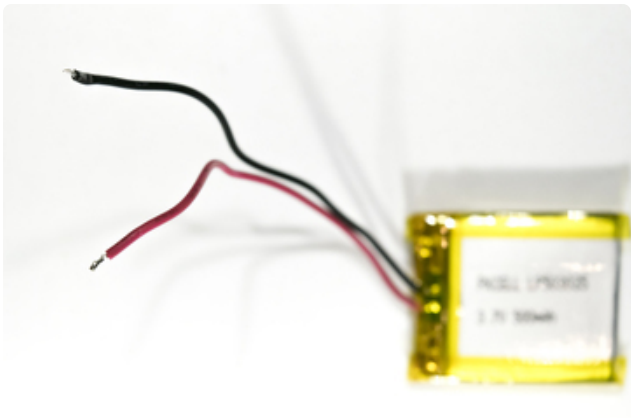


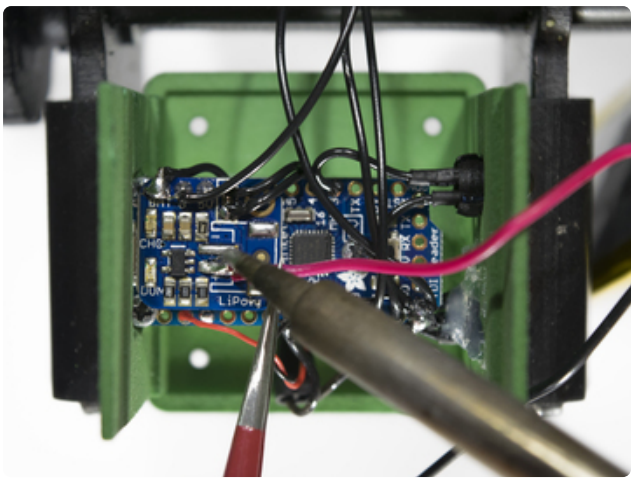


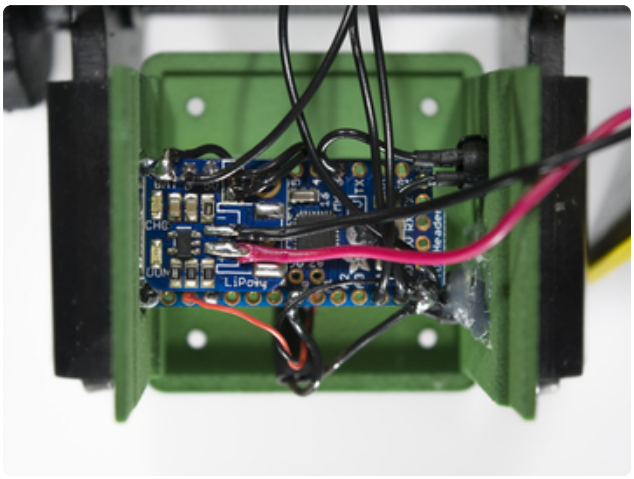


Carefully fit the case together with the matrix wires on the side of the IR components. If the case doesn't fit relatively smoothly, cut away excess hot glue from the IR receiver sensor to fit the case together. Then separate the halves again.

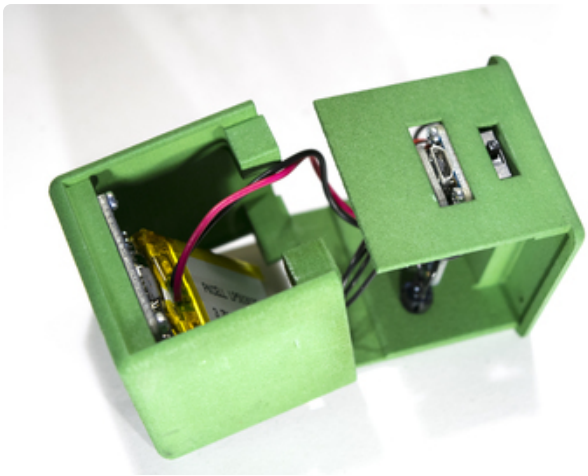


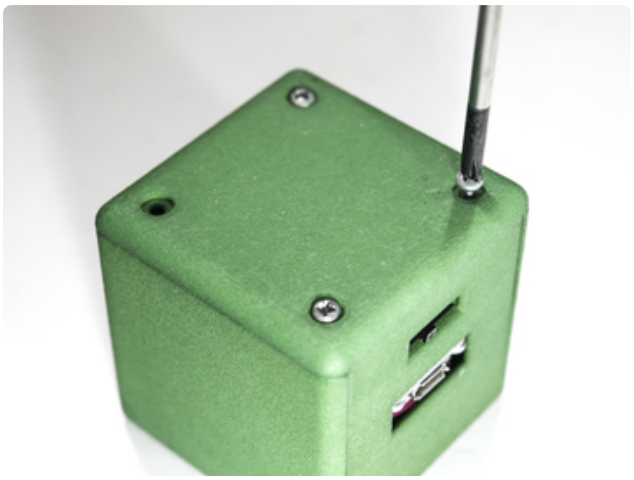


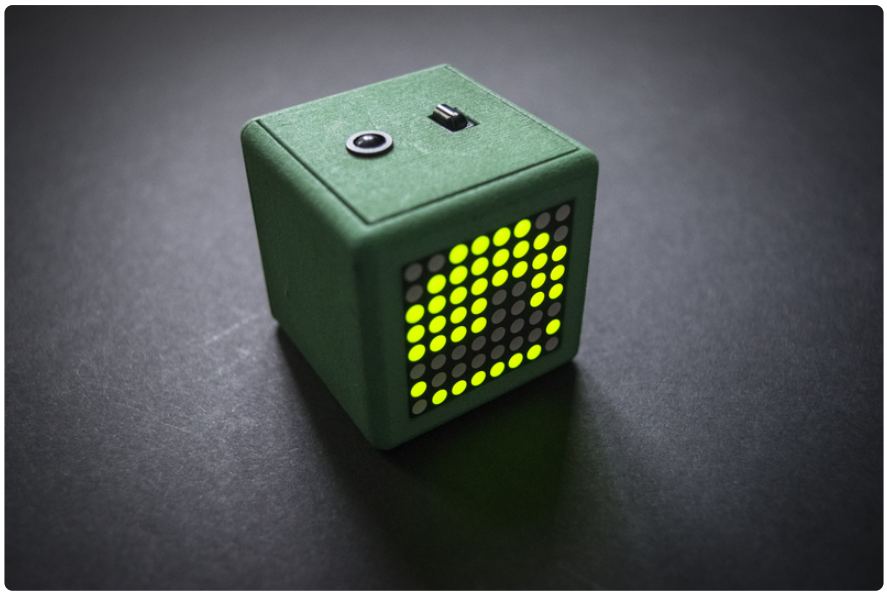




Cut off the JST connector from the LiPo battery and strip and tin the wires. Solder the red wire to the + connection of the LIPOLY backpack, and the black wire to the - connection. Then your cube is all wired up! Align the halves as shown with the battery parallel to the matrix, then close up the cube and screw it together with 4x 5mm M2 machine screws Your Adafriend is complete!







With your Adafriend assembled, we can move on to giving it life! (Code is life)

Software

Arduino IDE Preparation

To prepare the Arduino environment for programming the Pro Trinket, follow the instructions on Tony DiCola's guide to setting up the IDE for the Pro Trinket here:

Pro Trinket Set Up

https://adafru.it/jAc

For best results, use the latest version of the Arduino IDE

Required Libraries

For this project, the LED matrix backpack, animations, and tone libraries need to be installed. Find them below.

GFX Library

https://adafru.it/aJa

LED Backpack Library

https://adafru.it/mau

Tone Library

https://adafru.it/y5F

If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install **Adafruit_BusIO** (https://adafru.it/Ldl) (newer versions will install this dependency automatically through the Arduino Library Manager).

Adafriend Software

When you have finished readying the IDE, copy and paste the code below into your IDE window, select Pro Trinet 5V, tinyISP as the programmer, and upload!

```
// The Adafriend sketch is derived from P. Burgess's 'roboface' example sketch for
Adafruit I2C 8x8 LED backpacks:
//
//  www.adafruit.com/products/870   www.adafruit.com/products/1049
//  www.adafruit.com/products/871   www.adafruit.com/products/1050
//  www.adafruit.com/products/872   www.adafruit.com/products/1051
//  www.adafruit.com/products/959   www.adafruit.com/products/1052
//
// Requires Adafruit_LEDBackpack and Adafruit_GFX libraries.
// For a simpler introduction, see the 'matrix8x8' example.
//
// Adafruit invests time and resources providing this open source code,
// please support Adafruit and open-source hardware by purchasing
// products from Adafruit!
//
// Eye algorithm written by P. Burgess for Adafruit Industries.
// Code/Guide written by John Wall for Adafruit Industries.
// BSD license, all text above must be included in any redistribution.
// Arduino Flappy Bird homage by augustzf@gmail.com

#include &lt;Arduino.h&gt;
#include &lt;Wire.h&gt;
#include "Adafruit_LEDBackpack.h"
#include "Adafruit_GFX.h"
#include &lt;Tone.h&gt;

Adafruit_8x8matrix matrix = Adafruit_8x8matrix();

static const uint8_t  PROGMEM// Bitmaps are stored in program memory
sadBlinkImg[][8] = {    // Eye animation frames
  {
    B00100100,          // Fully open sad eye
    B01000010,
    B10000001,
    B00111100,
    B01111110,
    B11111111,
    B11111111,
    B11111111       }
  ,
  {
    B00100100,
    B01000010,
    B10000001,
    B00000000,
    B01111110,
    B11111111,
    B11111111,
```

```
    B11111111        }
  ,
  {
    B00100100,
    B01000010,
    B10000001,
    B00000000,
    B00000000,
    B01111110,
    B11111111,
    B11111111       }
  ,
  {
    B00100100,
    B01000010,
    B10000001,
    B00000000,
    B00000000,
    B00000000,
    B01111110,
    B11111111        }
  ,
  {
    B00100100,          // Fully closed sad eye
    B01000010,
    B10000001,
    B00000000,
    B00000000,
    B00000000,
    B00000000,
    B11111111       }

};

static const uint8_t  PROGMEM// Bitmaps are stored in program memory
blinkImg[][8] = {    // Eye animation frames
  {
    B00111100,          // Fully open eye
    B01111110,
    B11111111,
    B11111111,
    B11111111,
    B11111111,
    B01111110,
    B00111100       }
  ,
  {
    B00000000,
    B01111110,
    B11111111,
    B11111111,
    B11111111,
    B11111111,
    B01111110,
    B00111100       }
  ,
  {
    B00000000,
    B00000000,
    B00111100,
    B11111111,
    B11111111,
    B11111111,
    B00111100,
    B00000000       }
  ,
  {
    B00000000,
    B00000000,
```

```
    B00000000,
    B00111100,
    B11111111,
    B01111110,
    B00011000,
    B00000000       }
  ,
  {
    B00000000,          // Fully closed eye
    B00000000,
    B00000000,
    B00000000,
    B10000001,
    B01111110,
    B00000000,
    B00000000       }
};

static const uint8_t  PROGMEM// Bitmaps are stored in program memory
happyBlinkImg[][8] = {    // Eye animation frames
  {
    B00111100,          // Fully open happy eye
    B01111110,
    B11111111,
    B11111111,
    B11111111,
    B00000000,
    B10000001,
    B01111110       }
  ,
  {
    B00000000,
    B01111110,
    B11111111,
    B11111111,
    B11111111,
    B00000000,
    B10000001,
    B01111110       }
  ,
  {
    B00000000,
    B00000000,
    B01111110,
    B11111111,
    B11111111,
    B00000000,
    B10000001,
    B01111110       }
  ,
  {
    B00000000,
    B00000000,
    B00000000,
    B01111110,
    B11111111,
    B00000000,
    B10000001,
    B01111110       }
  ,
  {
    B00000000,          // Fully closed happy eye
    B00000000,
    B00000000,
    B01111110,
    B10000001,
    B00000000,
    B10000001,
    B01111110       }
```

```
};

static const uint8_t PROGMEM // Bitmaps are stored in program memory
annoyedBlinkImg[][8] = {    // Eye animation frames
  {
    B10000001,          // Fully open annoyed eye
    B01100110,
    B00000000,
    B11111111,
    B11111111,
    B11111111,
    B01111110,
    B00111100       }
  ,
  {
    B10000001,
    B01100110,
    B00000000,
    B11111111,
    B11111111,
    B11111111,
    B01111110,
    B00000000        }
  ,
  {
    B10000001,
    B01100110,
    B00000000,
    B11111111,
    B11111111,
    B01111110,
    B00000000,
    B00000000       }
  ,
  {
    B10000001,
    B01100110,
    B00000000,
    B11111111,
    B01111110,
    B00000000,
    B00000000,
    B00000000        }
  ,
  {
    B10000001,          // Fully closed annoyed eye
    B01100110,
    B00000000,
    B10000001,
    B01111110,
    B00000000,
    B00000000,
    B00000000       }

};

uint8_t
blinkIndex[] = {
  1, 2, 3, 4, 3, 2, 1 },
// Blink bitmap sequence
blinkCountdown = 100, // Countdown to next blink (in frames)
gazeCountdown  =  75, // Countdown to next eye movement
gazeFrames     =  50; // Duration of eye movement (smaller = faster)
int8_t
eyeX = 3, eyeY = 3,   // Current eye position
newX = 3, newY = 3,   // Next eye position
dX   = 0, dY   = 0;   // Distance from prior to new position

Tone tone1;
```

```
#define OCTAVE_OFFSET 0

const int notes[] = {
  0,
  NOTE_C4, NOTE_CS4, NOTE_D4, NOTE_DS4, NOTE_E4, NOTE_F4, NOTE_FS4, NOTE_G4,
NOTE_GS4, NOTE_A4, NOTE_AS4, NOTE_B4, //scale of notes and their frequencies
  NOTE_C5, NOTE_CS5, NOTE_D5, NOTE_DS5, NOTE_E5, NOTE_F5, NOTE_FS5, NOTE_G5,
NOTE_GS5, NOTE_A5, NOTE_AS5, NOTE_B5,
  NOTE_C6, NOTE_CS6, NOTE_D6, NOTE_DS6, NOTE_E6, NOTE_F6, NOTE_FS6, NOTE_G6,
NOTE_GS6, NOTE_A6, NOTE_AS6, NOTE_B6,
  NOTE_C7, NOTE_CS7, NOTE_D7, NOTE_DS7, NOTE_E7, NOTE_F7, NOTE_FS7, NOTE_G7,
NOTE_GS7, NOTE_A7, NOTE_AS7, NOTE_B7
};

const char *songs[] = { // A selection of songs that the Adafriend will sing when
happy. Watch for memory limits! These 7 are tested and working.
"Mario:d=4,o=5,b=100:16e6,16e6,32p,8e6,16c6,8e6,8g6,8p,8g,8p,8c6,16p,8g,16p,8e,16p,
8a,8b,16a#,8a,16g.,16e6,16g6,8a6,16f6,8g6,8e6,16c6,16d6,8b,16p,8c6,16p,8g,16p,8e,
16p,8a,8b,16a#,8a,16g.,16e6,16g6,8a6,16f6,8g6,8e6,16c6,16d6,8b",
"Indiana Jones:d=4,o=5,b=250:e,8p,8f,8g,8p,1c6,8p.,d,8p,8e,1f,p.,g,8p,8a,8b,8p,
1f6,p,a,8p,8b,2c6,2d6,2e6,e,8p,8f,8g,8p,1c6,p,d6,8p,8e6,1f.6,g,8p,8g,e.6,8p,d6,8p,
8g,e.6,8p,d6,8p,8g,f.6,8p,e6,8p,8d6,2c6",
"Take On Me:d=4,o=4,b=160:8f#5,8f#5,8f#5,8d5,8p,8b,8p,8e5,8p,8e5,8p,
8e5,8g#5,8g#5,8a5,8b5,8a5,8a5,8a5,8e5,8p,8d5,8p,8f#5,8p,8f#5,8p,
8f#5,8e5,8e5,8f#5,8e5,8f#5,8f#5,8f#5,8d5,8p,8b,8p,8e5,8p,8e5,8p,
8e5,8g#5,8g#5,8a5,8b5,8a5,8a5,8a5,8e5,8p,8d5,8p,8f#5,8p,8f#5,8p,8f#5,8e5,8e5",
"Tetris:d=4,o=5,b=150:e6,8b,8c6,8d6,16e6,16d6,8c6,8b,a,8a,8c6,e6,8d6,8c6,b,8b,
8c6,d6,e6,c6,a,2a,8p,d6,8f6,a6,8g6,8f6,e6,8e6,8c6,e6,8d6,8c6,b,8b,8c6,d6,e6,c6,a,a",
"Jeopardy:d=4,o=6,b=125:c,f,c,f5,c,f,2c,c,f,c,f,a.,8g,8f,8e,8d,8c#,c,f,c,f5,c,f,
2c,f.,8d,c,a#5,a5,g5,f5",
"Mahna Mahna:d=16,o=6,b=125:c#,c.,b5,8a#.5,8f.,4g#,a#,g.,4d#,8p,c#,c.,b5,8a#.
5,8f.,g#.,8a#.,4g,8p,c#,c.,b5,8a#.5,8f.,4g#,f,g.,8d#.,f,g.,8d#.,f,8g,8d#.,f,8g,d#,
8c,a#5,8d#.,8d#.,4d#,8d#.",
"The Simpsons:d=4,o=5,b=160:c.6,e6,f#6,8a6,g.6,e6,c6,8a,8f#,8f#,8f#,2g,8p,8p,8f#,
8f#,8f#,8g,a#.,8c6,8c6,8c6,c6"
};

//char *song = "The Simpsons:d=4,o=5,b=160:c.6,e6,f#6,8a6,g.6,e6,c6,8a,8f#,8f#,8f#,
2g,8p,8p,8f#,8f#,8f#,8g,a#.,8c6,8c6,8c6,c6";
//char *song = "Indiana:d=4,o=5,b=250:e,8p,8f,8g,8p,1c6,8p.,d,8p,8e,1f,p.,g,8p,8a,
8b,8p,1f6,p,a,8p,8b,2c6,2d6,2e6,e,8p,8f,8g,8p,1c6,p,d6,8p,8e6,1f.6,g,8p,8g,e.
6,8p,d6,8p,8g,e.6,8p,d6,8p,8g,f.6,8p,e6,8p,8d6,2c6";
//char *song = "TakeOnMe:d=4,o=4,b=160:8f#5,8f#5,8f#5,8d5,8p,8b,8p,8e5,8p,8e5,8p,
8e5,8g#5,8g#5,8a5,8b5,8a5,8a5,8a5,8e5,8p,8d5,8p,8f#5,8p,8f#5,8p,
8f#5,8e5,8e5,8f#5,8e5,8f#5,8f#5,8f#5,8d5,8p,8b,8p,8e5,8p,8e5,8p,
8e5,8g#5,8g#5,8a5,8b5,8a5,8a5,8a5,8e5,8p,8d5,8p,8f#5,8p,8f#5,8p,8f#5,8e5,8e5";
//char *song = "Entertainer:d=4,o=5,b=140:8d,8d#,8e,c6,8e,c6,8e,2c.
6,8c6,8d6,8d#6,8e6,8c6,8d6,e6,8b,d6,2c6,p,8d,8d#,8e,c6,8e,c6,8e,2c.6,8p,8a,8g,8f#,
8a,8c6,e6,8d6,8c6,8a,2d6";
//char *song = "Muppets:d=4,o=5,b=250:c6,c6,a,b,8a,b,g,p,c6,c6,a,8b,8a,
8p,g.,p,e,e,g,f,8e,f,8c6,8c,8d,e,8e,8e,8p,8e,g,2p,c6,c6,a,b,8a,b,g,p,c6,c6,a,
8b,a,g.,p,e,e,g,f,8e,f,8c6,8c,8d,e,8e,d,8d,c";
//char *song = "Xfiles:d=4,o=5,b=125:e,b,a,b,d6,2b.,1p,e,b,a,b,e6,2b.,
1p,g6,f#6,e6,d6,e6,2b.,1p,g6,f#6,e6,d6,f#6,2b.,1p,e,b,a,b,d6,2b.,1p,e,b,a,b,e6,2b.,
1p,e6,2b.";
//char *song = "Looney:d=4,o=5,b=140:32p,c6,8f6,8e6,8d6,8c6,a.,
8c6,8f6,8e6,8d6,8d#6,e.6,8e6,8e6,8c6,8d6,8c6,8e6,8c6,8d6,8a,8c6,8g,8a#,8a,8f";
//char *song = "20thCenFox:d=16,o=5,b=140:b,8p,b,b,2b,p,c6,32p,b,32p,c6,32p,b,
32p,c6,32p,b,8p,b,b,b,32p,b,32p,b,32p,b,32p,b,32p,b,32p,b,32p,g#,32p,a,32p,b,8p,b,b,
2b,4p,8e,8g#,8b,1c#6,8f#,8a,8c#6,1e6,8a,8c#6,8e6,1e6,8b,8g#,8a,2b";
//char *song = "Bond:d=4,o=5,b=80:32p,
16c#6,32d#6,32d#6,16d#6,8d#6,16c#6,16c#6,16c#6,16c#6,32e6,32e6,16e6,8e6,16d#6,16d#6,16d#6,16c#
7,16g#6,16f#6,g#.6";
//char *song = "MASH:d=8,o=5,b=140:4a,4g,f#,g,p,f#,p,g,p,f#,p,2e.,p,f#,e,
4f#,e,f#,p,e,p,4d.,p,f#,4e,d,e,p,d,p,e,p,d,p,2c#.,p,d,c#,4d,c#,d,p,e,p,4f#,p,a,p,
4b,a,b,p,a,p,b,p,2a.,4p,a,b,a,4b,a,b,p,2a.,a,4f#,a,b,p,d6,p,4e.6,d6,b,p,a,p,2b";
//char *song = "StarWars:d=4,o=5,b=45:32p,32f#,32f#,32f#,8b.,8f#.
6,32e6,32d#6,32c#6,8b.6,16f#.6,32e6,32d#6,32c#6,8b.6,16f#.6,32e6,32d#6,32e6,8c#.
```

```
6,32f#,32f#,32f#,8b.,8f#.6,32e6,32d#6,32c#6,8b.6,16f#.6,32e6,32d#6,32c#6,8b.6,16f#.
6,32e6,32d#6,32e6,8c#6";
//char *song = "GoodBad:d=4,o=5,b=56:32p,32a#,32d#6,32a#,32d#6,8a#.,16f#.,16g#.,d#,
32a#,32d#6,32a#,32d#6,8a#.,16f#.,16g#.,c#6,32a#,32d#6,32a#,32d#6,8a#.,16f#.,32f.,
32d#.,c#,32a#,32d#6,32a#,32d#6,8a#.,16g#.,d#";
//char *song = "TopGun:d=4,o=4,b=31:32p,16c#,16g#,16g#,32f#,32f,32f#,32f,16d#,16d#,
32c#,32d#,16f,32d#,32f,16f#,32f,32c#,16f,d#,16c#,16g#,16g#,32f#,32f,32f#,32f,16d#,
16d#,32c#,32d#,16f,32d#,32f,16f#,32f,32c#,g#";
//char *song = "A-Team:d=8,o=5,b=125:4d#6,a#,2d#6,16p,g#,4a#,4d#.,p,16g,
16a#,d#6,a#,f6,2d#6,16p,c#.6,16c6,16a#,g#.,2a#";
//char *song = "Flinstones:d=4,o=5,b=40:32p,16f6,16a#,16a#6,32g6,16f6,16a#.,
16f6,32d#6,32d6,32d6,32d#6,32f6,16a#,16c6,d6,16f6,16a#.,16a#6,32g6,16f6,16a#.,
32f6,32f6,32d#6,32d6,32d6,32d#6,32f6,16a#,16c6,a#,16a6,16d.
6,16a#6,32a6,32a6,32g6,32f#6,32a6,8g6,16g6,16c.
6,32a6,32a6,32g6,32g6,32f6,32e6,32g6,8f6,16f6,16a#.,16a#6,32g6,16f6,16a#.,
16f6,32d#6,32d6,32d6,32d#6,32f6,16a#,16c.6,32d6,32d#6,32f6,16a#,16c.
6,32d6,32d#6,32f6,16a#6,16c7,8a#.6";
//char *song = "Jeopardy:d=4,o=6,b=125:c,f,c,f5,c,f,2c,c,f,c,f,a.,8g,8f,8e,8d,
8c#,c,f,c,f5,c,f,2c,f.,8d,c,a#5,a5,g5,f5,p,d#,g#,d#,g#5,d#,g#,2d#,d#,g#,d#,g#,c.
7,8a#,8g#,8g,8f,8e,d#,g#,d#,g#5,d#,g#,2d#,g#.,8f,d#,c#,c,p,a#5,p,g#.5,d#,g#";
//char *song = "Gadget:d=16,o=5,b=50:32d#,32f,32f#,32g#,a#,f#,a,f,g#,f#,32d#,32f,
32f#,32g#,a#,d#6,4d6,32d#,32f,32f#,32g#,a#,f#,a,f,g#,f#,8d#";
//char *song = "Smurfs:d=32,o=5,b=200:4c#6,16p,4f#6,p,16c#6,p,8d#6,p,8b,p,4g#,16p,
4c#6,p,16a#,p,8f#,p,8a#,p,4g#,4p,g#,p,a#,p,b,p,c6,p,4c#6,16p,4f#6,p,16c#6,p,8d#6,p,
8b,p,4g#,16p,4c#6,p,16a#,p,8b,p,8f,p,4f#";
//char *song = "MahnaMahna:d=16,o=6,b=125:c#,c.,b5,8a#.5,8f.,4g#,a#,g.,4d#,
8p,c#,c.,b5,8a#.5,8f.,g#.,8a#.,4g,8p,c#,c.,b5,8a#.5,8f.,4g#,f,g.,8d#.,f,g.,8d#.,f,
8g,8d#.,f,8g,d#,8c,a#5,8d#.,8d#.,4d#,8d#.";
//char *song = "LeisureSuit:d=16,o=6,b=56:f.5,f#.5,g.5,g#5,32a#5,f5,g#.5,a#.
5,32f5,g#5,32a#5,g#5,8c#.,a#5,32c#,a5,a#.5,c#.,32a5,a#5,32c#,d#,
8e,c#.,f.,f.,f.,f.,f,32e,d#,8d,a#.5,e,32f,e,32f,c#,d#.,c#";
//char *song = "MissionImp:d=16,o=6,b=95:32d,32d#,32d,32d#,32d,32d#,32d,32d#,32d,
32d,32d#,32e,32f,32f#,32g,g,8p,g,8p,a#,p,c7,p,g,8p,g,8p,f,p,f#,p,g,8p,g,
8p,a#,p,c7,p,g,8p,g,8p,f,p,f#,p,a#,g,2d,32p,a#,g,2c#,32p,a#,g,2c,a#5,8c,2p,
32p,a#5,g5,2f#,32p,a#5,g5,2f,32p,a#5,g5,2e,d#,8d";
//char *song = "Mario:d=4,o=5,b=100:16e6,16e6,32p,8e6,16c6,8e6,8g6,8p,8g,8p,8c6,16p,
8g,16p,8e,16p,8a,8b,16a#,8a,16g.,16e6,16g6,8a6,16f6,8g6,8e6,16c6,16d6,8b,16p,
8c6,16p,8g,16p,8e,16p,8a,8b,16a#,8a,16g.,16e6,16g6,8a6,16f6,8g6,8e6,16c6,16d6,8b,8p,
16g6,16f#6,16f6,16d#6,16p,16e6,16p,16g#,16a,16c6,16p,16a,16c6,16d6,8p,
16g6,16f#6,16f6,16d#6,16p,16e6,16p,16c7,16p,16c7,16c7,p,16g6,16f#6,16f6,16d#6,16p,
16e6,16p,16g#,16a,16c6,16p,16a,16c6,16d6,8p,16d#6,8p,16d6,8p,16c6";
//char *song = "tetris:d=4,o=5,b=150:e6,8b,8c6,8d6,16e6,16d6,8c6,8b,a,8a,
8c6,e6,8d6,8c6,b,8b,8c6,d6,e6,c6,a,2a,8p,d6,8f6,a6,8g6,8f6,e6,8e6,8c6,e6,8d6,8c6,b,
8b,8c6,d6,e6,c6,a,a";

int tapNum = 15; // logged number of taps
byte mood = 1; // current mood
const byte vibration PROGMEM = A0; // vibration sensor
const int tapLevel PROGMEM = 512;
long previousMillis = 0;        // will store last time LED was updated

// the follow variables is a long because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
const int decay PROGMEM = 30000;           // interval at which to decay emotions

unsigned long
checkMillis,
 tapMillis,
 songMillis,
 currentMillis,
gameMillis;

void setup() {

  // Seed random number generator from an unused analog input:
  randomSeed(analogRead(A7));
  pinMode(vibration, INPUT_PULLUP);

  matrix.begin(0x70);
```

```
  matrix.setRotation(3);
  matrix.setBrightness(4);
  matrix.setTextSize(1);
  matrix.setTextWrap(false);
  matrix.setTextColor(LED_ON);
  matrix.clear();
  matrix.writeDisplay();
  tone1.begin(10);
}

void loop() {

  while(mood==0) // sad mood
  {
    // Draw eyeball in current state of blinkyness (no pupil).
    matrix.clear();
    // When counting down to the next blink, show the eye in the fully-
    // open state.  On the last few counts (during the blink), look up
    // the corresponding bitmap index.
    matrix.drawBitmap(0, 0,
    sadBlinkImg[
      (blinkCountdown &lt; sizeof(blinkIndex)) ? // Currently blinking?
    blinkIndex[blinkCountdown] :            // Yes, look up bitmap #
    0                                       // No, show bitmap 0
    ], 8, 8, LED_ON);
    // Decrement blink counter.  At end, set random time for next blink.
    if(--blinkCountdown == 0) blinkCountdown = random(5, 180);

    // Add a pupil (2x2 black square) atop the blinky eyeball bitmap.
    // Periodically, the pupil moves to a new position...
    if(--gazeCountdown &lt;= gazeFrames) {
      // Eyes are in motion - draw pupil at interim position
      matrix.fillRect(
      newX - (dX * gazeCountdown / gazeFrames),
      newY - (dY * gazeCountdown / gazeFrames),
      2, 2, LED_OFF);
      if(gazeCountdown == 0) {    // Last frame?
        eyeX = newX;
        eyeY = newY; // Yes.  What's new is old, then...
        do { // Pick random positions until one is within the eye circle
          newX = random(0,7);
          newY = random(5,7);
          dX   = newX-3;
          dY   = newY-3;
        }
        while((dX * dX + dY * dY) &gt;= 10);        // Thank you Pythagoras
        dX            = newX - eyeX;             // Horizontal distance to move
        dY            = newY - eyeY;             // Vertical distance to move
        gazeFrames    = random(3, 15);           // Duration of eye movement
        gazeCountdown = random(gazeFrames, 120); // Count to end of next movement
      }
    }
    else {
      // Not in motion yet -- draw pupil at current static position
      matrix.fillRect(eyeX, eyeY, 2, 2, LED_OFF);
    }


    // Refresh all of the matrices in one quick pass
    matrix.writeDisplay();

    if(millis()-checkMillis &gt; random(10000,30000))
    {
      sadNoise();
      checkMillis = millis();
    }

    tapMillis = millis();
```

```
  while(millis()-tapMillis &lt; 40)
  {
    checkTaps();
  }
  songMillis = millis();
}

while(mood==1) // neutral mood
{
  // Draw eyeball in current state of blinkyness (no pupil).
  matrix.clear();
  // When counting down to the next blink, show the eye in the fully-
  // open state.  On the last few counts (during the blink), look up
  // the corresponding bitmap index.
  matrix.drawBitmap(0, 0,
  blinkImg[
    (blinkCountdown &lt; sizeof(blinkIndex)) ? // Currently blinking?
  blinkIndex[blinkCountdown] :            // Yes, look up bitmap #
  0                                       // No, show bitmap 0
  ], 8, 8, LED_ON);
  // Decrement blink counter.  At end, set random time for next blink.
  if(--blinkCountdown == 0) blinkCountdown = random(5, 180);

  // Add a pupil (2x2 black square) atop the blinky eyeball bitmap.
  // Periodically, the pupil moves to a new position...
  if(--gazeCountdown &lt;= gazeFrames) {
    // Eyes are in motion - draw pupil at interim position
    matrix.fillRect(
    newX - (dX * gazeCountdown / gazeFrames),
    newY - (dY * gazeCountdown / gazeFrames),
    2, 2, LED_OFF);
    if(gazeCountdown == 0) {    // Last frame?
      eyeX = newX;
      eyeY = newY; // Yes.  What's new is old, then...
      do { // Pick random positions until one is within the eye circle
        newX = random(7);
        newY = random(7);
        dX   = newX - 3;
        dY   = newY - 3;
      }
      while((dX * dX + dY * dY) &gt;= 10);      // Thank you Pythagoras
      dX            = newX - eyeX;           // Horizontal distance to move
      dY            = newY - eyeY;           // Vertical distance to move
      gazeFrames    = random(3, 15);         // Duration of eye movement
      gazeCountdown = random(gazeFrames, 120); // Count to end of next movement
    }
  }
  else {
    // Not in motion yet -- draw pupil at current static position
    matrix.fillRect(eyeX, eyeY, 2, 2, LED_OFF);
  }


  // Refresh all of the matrices in one quick pass
  matrix.writeDisplay();

   if(millis()-checkMillis &gt; random(10000,30000))
  {
    neutralNoise();
    checkMillis = millis();
  }


  tapMillis = millis();

  while(millis()-tapMillis &lt; 40)
  {
    checkTaps();
  }
```

```
  songMillis = millis();
}

while(mood == 2) // happy mood
{
  // Draw eyeball in current state of blinkyness (no pupil).
  matrix.clear();
  // When counting down to the next blink, show the eye in the fully-
  // open state.  On the last few counts (during the blink), look up
  // the corresponding bitmap index.
  matrix.drawBitmap(0, 0,
  happyBlinkImg[
    (blinkCountdown &lt; sizeof(blinkIndex)) ? // Currently blinking?
  blinkIndex[blinkCountdown] :            // Yes, look up bitmap #
  0                                       // No, show bitmap 0
  ], 8, 8, LED_ON);
  // Decrement blink counter.  At end, set random time for next blink.
  if(--blinkCountdown == 0) blinkCountdown = random(5, 180);

  // Add a pupil (2x2 black square) atop the blinky eyeball bitmap.
  // Periodically, the pupil moves to a new position...
  if(--gazeCountdown &lt;= gazeFrames) {
    // Eyes are in motion - draw pupil at interim position
    matrix.fillRect(
    newX - (dX * gazeCountdown / gazeFrames),
    newY - (dY * gazeCountdown / gazeFrames),
    2, 2, LED_OFF);
    if(gazeCountdown == 0) {    // Last frame?
      eyeX = newX;
      eyeY = newY; // Yes.  What's new is old, then...
      do { // Pick random positions until one is within the eye circle
        newX = random(7);
        newY = random(4);
        dX   = newX - 3;
        dY   = newY - 3;
      }
      while((dX * dX + dY * dY) &gt;= 10);      // Thank you Pythagoras
      dX            = newX - eyeX;             // Horizontal distance to move
      dY            = newY - eyeY;             // Vertical distance to move
      gazeFrames    = random(3, 15);           // Duration of eye movement
      gazeCountdown = random(gazeFrames, 120); // Count to end of next movement
    }
  }
  else {
    // Not in motion yet -- draw pupil at current static position
    matrix.fillRect(eyeX, eyeY, 2, 2, LED_OFF);
  }


  // Refresh all of the matrices in one quick pass
  matrix.writeDisplay();

  if(millis()-checkMillis &gt; random(10000,30000))
  {
    happyNoise();
    checkMillis = millis();
  }

  tapMillis = millis();

  while(millis()-tapMillis &lt; 40)
  {
    checkTaps();
  }

}

while(mood==3) // annoyed mood
{
```

```
    // Draw eyeball in current state of blinkyness (no pupil).
    matrix.clear();
    // When counting down to the next blink, show the eye in the fully-
    // open state.  On the last few counts (during the blink), look up
    // the corresponding bitmap index.
    matrix.drawBitmap(0, 0,
    annoyedBlinkImg[
      (blinkCountdown &lt; sizeof(blinkIndex)) ? // Currently blinking?
    blinkIndex[blinkCountdown] :            // Yes, look up bitmap #
    0                                       // No, show bitmap 0
    ], 8, 8, LED_ON);
    // Decrement blink counter.  At end, set random time for next blink.
    if(--blinkCountdown == 0) blinkCountdown = random(5, 180);

    // Add a pupil (2x2 black square) atop the blinky eyeball bitmap.
    // Periodically, the pupil moves to a new position...
    if(--gazeCountdown &lt;= gazeFrames) {
      // Eyes are in motion - draw pupil at interim position
      matrix.fillRect(
      newX - (dX * gazeCountdown / gazeFrames),
      newY - (dY * gazeCountdown / gazeFrames),
      2, 2, LED_OFF);
      if(gazeCountdown == 0) {    // Last frame?
        eyeX = newX;
        eyeY = newY; // Yes.  What's new is old, then...
        do { // Pick random positions until one is within the eye circle
          newX = random(7);
          newY = random(3,7);
          dX   = newX - 3;
          dY   = newY - 3;
        }
        while((dX * dX + dY * dY) &gt;= 10);      // Thank you Pythagoras
        dX            = newX - eyeX;            // Horizontal distance to move
        dY            = newY - eyeY;            // Vertical distance to move
        gazeFrames    = random(3, 15);          // Duration of eye movement
        gazeCountdown = random(gazeFrames, 120); // Count to end of next movement
      }
    }
    else {
      // Not in motion yet -- draw pupil at current static position
      matrix.fillRect(eyeX, eyeY, 2, 2, LED_OFF);
    }

    // Refresh all of the matrices in one quick pass
    matrix.writeDisplay();

    if(millis()-checkMillis &gt; random(10000,30000))
    {
      angryNoise();
      checkMillis = millis();
    }

    tapMillis = millis();

    while(millis()-tapMillis &lt; 40)
    {
      checkTaps();
    }
  }
}

#define isdigit(n) (n &gt;= '0' &amp;&amp; n &lt;= '9')

void play_rtttl(const char *p) // the method to play a song
{
  // Absolutely no error checking in here

  byte default_dur = 4;
  byte default_oct = 6;
```

```
  int bpm = 63;
  int num;
  long wholenote;
  long duration;
  byte note;
  byte scale;

  // format: d=N,o=N,b=NNN:
  // find the start (skip name, etc)

  while(*p != ':') p++;    // ignore name
  p++;                     // skip ':'

  // get default duration
  if(*p == 'd')
  {
    p++;
    p++;              // skip "d="
    num = 0;
    while(isdigit(*p))
    {
      num = (num * 10) + (*p++ - '0');
    }
    if(num &gt; 0) default_dur = num;
    p++;                   // skip comma
  }



  // get default octave
  if(*p == 'o')
  {
    p++;
    p++;              // skip "o="
    num = *p++ - '0';
    if(num &gt;= 3 &amp;&amp; num &lt;=7) default_oct = num;
    p++;                   // skip comma
  }



  // get BPM
  if(*p == 'b')
  {
    p++;
    p++;              // skip "b="
    num = 0;
    while(isdigit(*p))
    {
      num = (num * 10) + (*p++ - '0');
    }
    bpm = num;
    p++;                   // skip colon
  }



  // BPM usually expresses the number of quarter notes per minute
  wholenote = (60 * 1000L / bpm) * 4;  // this is the time for whole note (in
milliseconds)




  // now begin note loop
  while(*p)
  {
    // first, get note duration, if available
    num = 0;
```

```
    while(isdigit(*p))
    {
      num = (num * 10) + (*p++ - '0');
    }

    if(num) duration = wholenote / num;
    else duration = wholenote / default_dur;  // we will need to check if we are a
dotted note after

      // now get the note
    note = 0;

    switch(*p)
    {
    case 'c':
      note = 1;
      break;
    case 'd':
      note = 3;
      break;
    case 'e':
      note = 5;
      break;
    case 'f':
      note = 6;
      break;
    case 'g':
      note = 8;
      break;
    case 'a':
      note = 10;
      break;
    case 'b':
      note = 12;
      break;
    case 'p':
    default:
      note = 0;
    }
    p++;

    // now, get optional '#' sharp
    if(*p == '#')
    {
      note++;
      p++;
    }

    // now, get optional '.' dotted note
    if(*p == '.')
    {
      duration += duration/2;
      p++;
    }

    // now, get scale
    if(isdigit(*p))
    {
      scale = *p - '0';
      p++;
    }
    else
    {
      scale = default_oct;
    }

    scale += OCTAVE_OFFSET;

    if(*p == ',')
```

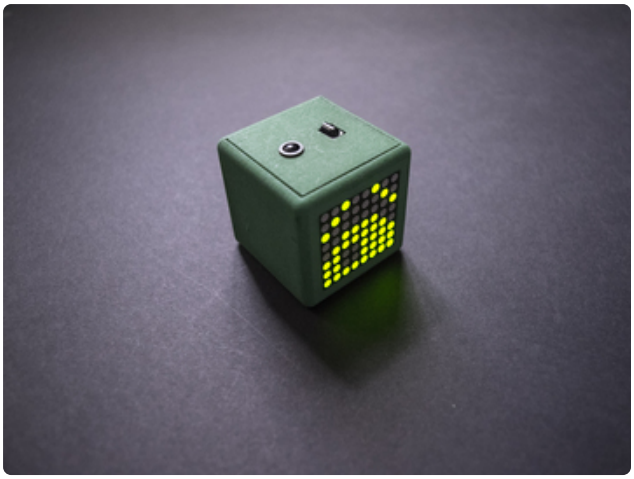
```
      p++;       // skip comma for next note (or we may be at the end)

    // now play the note

    if(note)
    {
      tone1.play(notes[(scale - 4) * 12 + note]);
      delay(duration);
      tone1.stop();
    }
    else
    {

      delay(duration);
    }
  }
}

void checkTaps()
{
  if(analogRead(vibration)&lt;tapLevel)
  {
    tapNum++;
    tone1.play(NOTE_C5);
    delay(200);
    tone1.stop();
  }

  if(tapNum &lt;= 10) mood = 0;
  else if(tapNum &lt;= 20) mood = 1;
  else if(tapNum &lt;= 30) mood = 2;
  else if(tapNum &gt; 30) mood = 3;

  currentMillis = millis();

  if(currentMillis - previousMillis &gt; decay) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;
    tapNum--;
    if(tapNum &lt; 0) tapNum = 0;
    if(tapNum == 30) tapNum = 15;
    if(tapNum &gt; 40) tapNum = 40;
  }

  currentMillis = millis();

  if(currentMillis - songMillis &gt; random(30000,60000) &amp;&amp; mood == 2)
  {
    matrix.clear();

    matrix.drawBitmap(0, 0, happyBlinkImg[0], 8, 8, LED_ON);
    eyeX = 3;
    eyeY = 3;
    matrix.fillRect(eyeX, eyeY, 2, 2, LED_OFF);
    matrix.writeDisplay();

    play_rtttl(songs[random(7)]);
    previousMillis = millis();
    checkMillis = millis();
    songMillis = millis();
  }
}

void happyNoise()
{
  int note = (int)random(13,39);
  tone1.play(notes[note]);
  delay(200);
  tone1.stop();
```

```
  tone1.play(notes[note+1]);
  delay(100);
  tone1.stop();
  tone1.play(notes[note+2]);
  delay(100);
  tone1.stop();
}

void sadNoise()
{
  int note = (int)random(2,26);
  tone1.play(notes[note]);
  delay(200);
  tone1.stop();
  tone1.play(notes[note-1]);
  delay(100);
  tone1.stop();
  tone1.play(notes[note-2]);
  delay(100);
  tone1.stop();
}

void angryNoise()
{
  int note = (int)random(2,7);
  tone1.play(notes[note]);
  delay(100);
  tone1.stop();
  tone1.play(notes[note-1]);
  delay(200);
  tone1.stop();
}

void neutralNoise()
{
  int note = (int)random(2,26);
  tone1.play(notes[note]);
  delay(200);
  tone1.stop();
}
```

Use and Behavior

Moods

To interact with your Adafriend, lightly tap on the case and you should hear a little tone of acknowlegement from inside. The Cube will wake up in a neutral mood when turned on, and its mood can be elevated with taps or settled with some alone time. The Adafriend's moods are as follows:



Sad

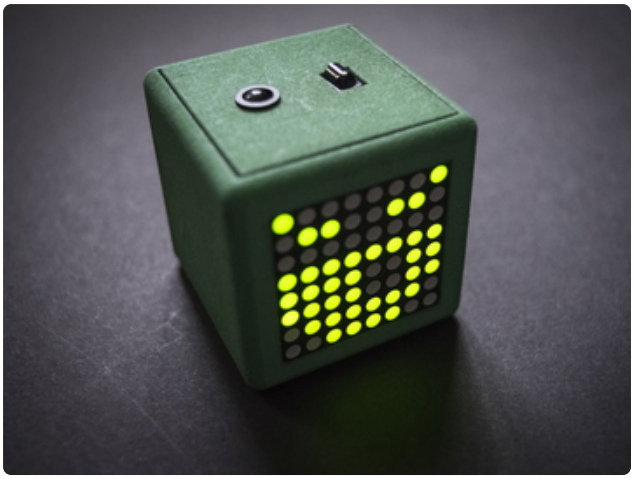
If left alone for too long, your Adafriend will become sad with sad eyebrows and make random sad tones that decrease in pitch in groups of three.

Neutral

In this mode, your Adafriend is content to simply gaze around and make single random tones.

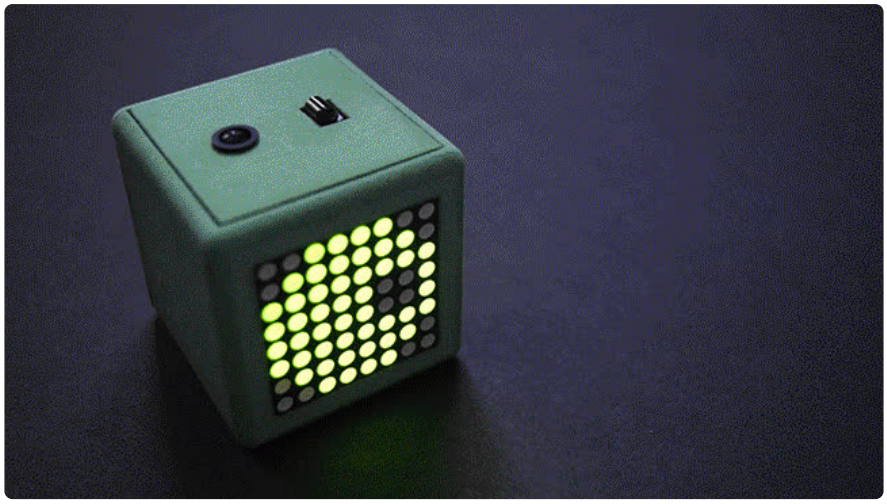
Happy

When your Adafriend is happy, a big smile will appear below its eye, and it will make happy tones that increase in pitch in groups of three. At random intervals, it will even sing a random happy tune from its saved list.



Annoyed

If you tap and annoy your Adafriend too much it wil become angry and show angry eyebrows and a scowl. It will make low pitched angry tones that decrease farther in pitch in groups of three. At this point, it needs some alone time, and will return to neutral when it has calmed down.



That's it! Have fun playing with your friend, customize its colors, add IR features, and I hope you enjoyed this guide!

What? There's another page? No there isn't, uh, nope, no page there...

•••

Pssst, like flappy bird? Me too. Upload this code to your Adafriend and see what happens. Have fun! You're welcome ;)

```
// Arduino Flappy Bird homage by augustzf@gmail.com
#include &lt;Arduino.h&gt;
#include &lt;Wire.h&gt;
#include "Adafruit_LEDBackpack.h"
#include "Adafruit_GFX.h"
#include &lt;Tone.h&gt;

Adafruit_8x8matrix matrix = Adafruit_8x8matrix();

Tone tone1;
```

```
#define OCTAVE_OFFSET 0

const int notes[] = {
  0,
  NOTE_C4, NOTE_CS4, NOTE_D4, NOTE_DS4, NOTE_E4, NOTE_F4, NOTE_FS4, NOTE_G4,
NOTE_GS4, NOTE_A4, NOTE_AS4, NOTE_B4,
  NOTE_C5, NOTE_CS5, NOTE_D5, NOTE_DS5, NOTE_E5, NOTE_F5, NOTE_FS5, NOTE_G5,
NOTE_GS5, NOTE_A5, NOTE_AS5, NOTE_B5,
  NOTE_C6, NOTE_CS6, NOTE_D6, NOTE_DS6, NOTE_E6, NOTE_F6, NOTE_FS6, NOTE_G6,
NOTE_GS6, NOTE_A6, NOTE_AS6, NOTE_B6,
  NOTE_C7, NOTE_CS7, NOTE_D7, NOTE_DS7, NOTE_E7, NOTE_F7, NOTE_FS7, NOTE_G7,
NOTE_GS7, NOTE_A7, NOTE_AS7, NOTE_B7
};

static const uint8_t PROGMEM
bird0[] =
{
  B00011000,
  B00111100,
  B00110100,
  B11111111,
  B10111111,
  B11011100,
  B01111100,
  B00111000 }
,
bird1[] =
{
  B00000000,
  B00011000,
  B00111100,
  B00110111,
  B11111111,
  B10011100,
  B11111100,
  B00111000 }
,
bird2[] =
{
  B00011000,
  B00111100,
  B00110100,
  B00111111,
  B01111111,
  B11011100,
  B10111100,
  B11111000 };

const byte vibration PROGMEM = A0; // vibration sensor
const int tapLevel PROGMEM = 512;

double initYVelocity = 6.5;
double gravity = -9.8;
long birdTime;
long previousBirdTime;
byte birdPos = 3;
byte tailPos = 3;
byte birdStart;
byte topWall;
byte bottomWall;
int wallDelay = 250;
byte wallGap = 5;
long wallTime;
int wallCount;
long previousWallTime;
byte wallPosition = 0;
boolean gameMode = false;
long frameTime;
long previousFrameTime;
```

```
byte frame = 0;
byte frameChange = 1;
byte lives = 2;

void setup() {

  pinMode(vibration, INPUT_PULLUP);

  matrix.begin(0x70);
  matrix.setRotation(3);
  matrix.setBrightness(3);
  matrix.setTextSize(1);
  matrix.setTextWrap(false);  // we dont want text to wrap so it scrolls nicely
  matrix.setTextColor(LED_ON);
  matrix.clear();
  matrix.writeDisplay();
  previousBirdTime = millis();
  topWall = random(0,4);
  bottomWall = topWall + 4;
  wallPosition = 0;
  gameMode = false;
}

void loop() {
  matrix.clear();

  if(gameMode == false)
  {

    if(analogRead(vibration)&lt; tapLevel)
    {
      gameMode = true;
    }

    frameTime = millis();

    if(frameTime - previousFrameTime &gt; 250) {
      previousFrameTime = frameTime;
      frame += frameChange;
      if(frame &gt;= 2 || frame &lt;= 0) frameChange *= -1;
    }

    if(frame == 0) matrix.drawBitmap(0, 0, bird0, 8, 8, LED_ON);
    else if(frame == 1) matrix.drawBitmap(0, 0, bird1, 8, 8, LED_ON);
    else if(frame == 2) matrix.drawBitmap(0, 0, bird2, 8, 8, LED_ON);
  }

  else
  {
    if(analogRead(vibration)&lt;tapLevel)
    {
      previousBirdTime = millis();
      birdStart = birdPos;
    }

    drawWalls();
    drawBird();

    if(wallPosition == 6)
    {
      byte next = 7 - calculateNextY();
      if(next &gt;= bottomWall || next &lt;= topWall)
      {
        gameMode = false;

        int scoreInt = wallCount-1;

        const char* temp = "Game Over";
```

```
        for (int8_t x=7; x&gt;=-1*9*5+1; x--)
        {
          matrix.clear();
          matrix.setCursor(x,0);
          matrix.print(temp);
          matrix.writeDisplay();
          delay(75);
        }
      }
      wallCount++;
    }

  }

  matrix.writeDisplay();
}

void drawWalls()
{
  wallTime = millis();

  if(wallPosition &gt; 7)
  {
    topWall = random(0,4);
    bottomWall = topWall + 4;
    wallPosition = 0;
  }

  matrix.drawLine(7-wallPosition, 0, 7-wallPosition, topWall, LED_ON);

  matrix.drawLine(7-wallPosition, bottomWall, 7-wallPosition, 7, LED_ON);

  if(wallTime - previousWallTime &gt; wallDelay) {
    previousWallTime = wallTime;
    wallPosition++;
  }
}

int calculateY()
{
  return (int)(birdStart+((birdTime/1000.0)*(initYVelocity+gravity*(birdTime/
1000.0)/2.0)));
}

int calculateTail()
{
  return (int)(birdStart+(((birdTime-100)/
1000.0)*(initYVelocity+gravity*((birdTime-100)/1000.0)/2.0)));
}

int calculateNextY()
{
  return (int)(birdStart+(((birdTime+100)/
1000.0)*(initYVelocity+gravity*((birdTime+100)/1000.0)/2.0)));
}

void drawBird()
{
  birdTime = millis() - previousBirdTime;
  birdPos = calculateY();
  tailPos = calculateTail();
  if(calculateY() &lt; 0) birdPos = 0;
  if(calculateTail() &lt; 0) tailPos = 0;
  matrix.drawPixel(0, 7-tailPos, LED_ON);
  matrix.drawPixel(1, 7-birdPos, LED_ON);
  matrix.writeDisplay();
}
```