Trinket / Gemma Space Invader Pendant

Created by Phillip Burgess

https://learn.adafruit.com/trinket-slash-gemma-space-invader-pendant

Last updated on 2021-11-16 11:26:10 AM EST

Table of Contents

Overview

Pac Man is my best friend.

Okay, not really. But as a child of the 80's, video arcade games were a huge cultural phenomenon of my formative years.

Tiny affordable microcontrollers are a cultural phenomenon of today's generation.

This project bridges the generations to create an animated LED necklace or charm that you can customize to create a retro-style personal "totem video game creature." It's a small project, good for electronics novices and group workshops. And you'll have something eye-catching to wear and show off afterward.

This guide was written for the Trinket Mini and Gemma v2 boards. It uses ATtiny-specific features and DOES NOT WORK with the newer "M0" boards, nor on larger "AVR" boards like Feather 328 or Arduino Uno.

Tools Needed

This is a soldering project, albeit a small one. You will need the common soldering paraphernalia of a soldering iron, solder, wire (20 to 26 gauge, either stranded or

solid) and tools for cutting and stripping wire.

Parts needed

- Adafruit Trinket Mini (https://adafru.it/egk) or Gemma v2 (http://adafru.it/1222) microcontroller board (if Trinket, either the 3.3V (http://adafru.it/1500) or 5V (http://adafru.it/1501) type works). The newer "M0" boards are not supported.
- Mini 8x8 LED Matrix (http://adafru.it/870) w/Backpack (http://adafru.it/870) (any color — we have several! Green for Space Invaders, yellow for Pac Man, etc.)
- 3.7V 150mAh Lithium-Ion Polymer Battery (http://adafru.it/1317)
- LiPo battery charger (http://adafru.it/1304)
- JST Battery Extension Cable (http://adafru.it/1131) (Trinket only — not required if using Gemma)
- Lanyard to create a necklace. This should be non-conductive — plastic lace (the sort used for weaving bracelets), rubber necklace cord or heavy fishing line all work. A pin back (such as our magnetic (http://adafru.it/1170) variety) is another possibility.
- Optional: momentary pushbutton to activate the animation. You could substitute a vibration sensor switch (https://adafru.it/uF4) to make it shake-activated. Or just use the tiny reset button that's built into the Trinket or Gemma to restart the animation code.
- Optional: If using Trinket, a bit of heat-shrink tube (http://adafru.it/344) is best for covering some connections; it's cleaner than alternatives like tape.

Can I use a "small" (1.2 inch) LED matrix instead of the "mini" (0.8 inch) version?

Sure can! Just be super extra careful to follow the assembly directions in the LED Backpack Guide (https://adafru.it/cJJ) and install the matrix the right way on the board. This is a common mistake! The "mini" matrix is recommended for this project because it's less troublesome for beginners.

Can I use other color matrices?

Yes! You can use any color of our 8x8 matrices: red, green, blue, white...

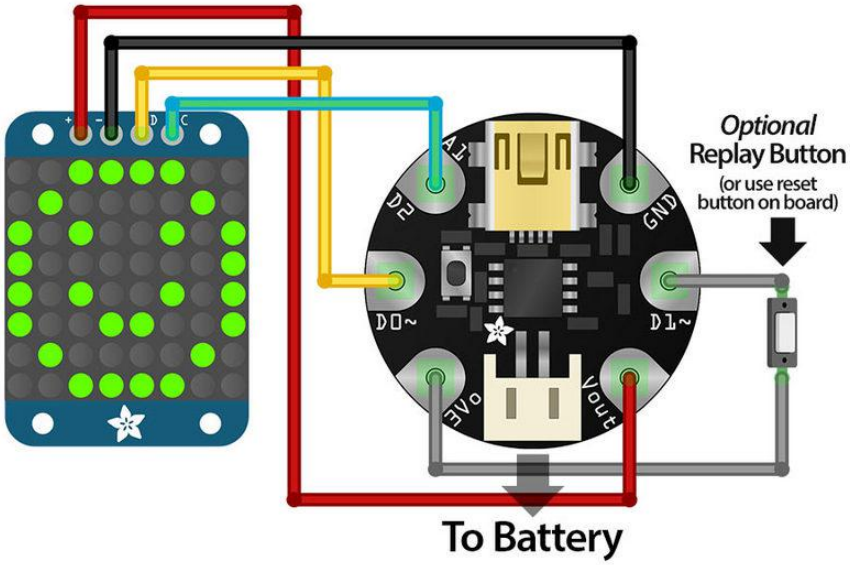
Can I use the bi-color matrix backpack?

Potentially yes, but you will have to adjust the bitmaps and code to handle the extra rows (it appears like a 16x8 matrix to the driver chip). We don't have a code or tutorial for this variation, but the wiring is the same if you're inclined to try.

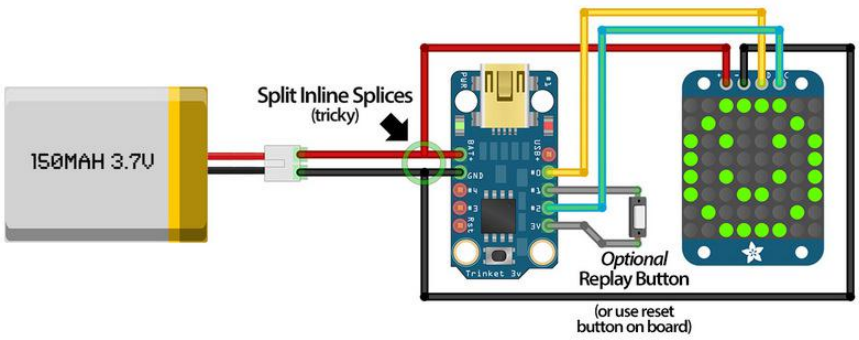
Wiring and Soldering

Here's the schematic when using a Gemma board. Simplicity itself! The battery isn't shown here...it simply plugs into the JST connector.

The battery charger isn't part of the circuit. Unplug the battery from the board and use the USB charger to top it off.



A couple extra steps are required if using a Trinket board (3.3V or 5V doesn't make a difference, the connections are the same):

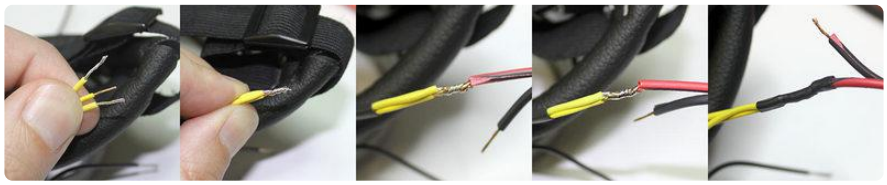


Since there's no JST battery connector on the Trinket, we'll need to make one using the battery extension cable. Cut it a couple inches from the socket end (where the battery plugs in), separate the red and black wires about halfway, and strip about 1/4" of insulation from the ends.

The red and black wires each need to connect to two places in the circuit: red goes to BAT+ on Trinket and + on the backpack, while black goes to GND and –. These connections require a split inline splice. Even a regular 1-to-1 inline splice can be tricky

for a novice, 2-way is an extra challenge. It's a little easier with narrow-gauge wire (e.g. 26 ga.).

This lets us route battery power to both the Trinket board and the LED matrix.



An example of a three-way inline splice (this is from another project — we'll be doing just a 2-way splice, but the technique is the same):

Line up the 2 wires on one side and twist them together. Slide heat-shrink tube as far down as possible. Then bring in the power wire from the opposite direction and twist around the others. Solder and heat-shrink the connection.

Work slowly and methodically, remember to slide the heat-shrink tube on FIRST before joining the wires, and use proper soldering technique: heat the wires and apply the solder there; don't move a glop of solder from the iron to the connection.

Arduino Code

If this is your first time using Trinket or Gemma, work through the Introducing Trinket (https://adafru.it/cEu) or Introducing Gemma (https://adafru.it/cHH) guide first; you need to customize some settings in the Arduino IDE. Once you have it up and running (test the "blink" sketch), then continue...

In the Arduino IDE, create a new sketch (File→New), then copy and paste the following code (click the "copy code" link at the top right, switch to the Arduino IDE and select Edit→Paste).

Alternately, you can use the "Download Project Bundle" button below. This will download a ZIP file containing all the code for both Arduino and CircuitPython — so you'll need to pick through for just the "Trinket_Gemma_Space_Invader_Pendant.ino" and "anim.h" files.

The program is fairly small but uses some advanced techniques, so don't be alarmed if a lot of it is unfamiliar. The important stuff you'll actually be editing is further ahead.

Two source files are required for the project. The second is explained a bit further down.

```
// SPDX-FileCopyrightText: 2018 Mikey Sklar for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// Trinket/Gemma + LED matrix backpack jewelry.  Plays animated
// sequence on LED matrix.  Press reset button to display again,
// or add optional momentary button between pin #1 and +V.
// THERE IS NO ANIMATION DATA IN THIS SOURCE FILE, you should
// rarely need to change anything here.  EDIT anim.h INSTEAD.

#define BRIGHTNESS   12 // 0=min, 15=max
#define I2C_ADDR   0x70 // Edit if backpack A0/A1 jumpers set

#include <Wire.h>
#include <avr/power.h>
#include <avr/sleep.h>
#include "anim.h"       // Animation data is located here

static const uint8_t PROGMEM reorder[] = { // Column-reordering table
    0x00,0x40,0x20,0x60,0x10,0x50,0x30,0x70,0x08,0x48,0x28,0x68,0x18,0x58,0x38,0x78,
    0x04,0x44,0x24,0x64,0x14,0x54,0x34,0x74,0x0c,0x4c,0x2c,0x6c,0x1c,0x5c,0x3c,0x7c,
    0x02,0x42,0x22,0x62,0x12,0x52,0x32,0x72,0x0a,0x4a,0x2a,0x6a,0x1a,0x5a,0x3a,0x7a,
    0x06,0x46,0x26,0x66,0x16,0x56,0x36,0x76,0x0e,0x4e,0x2e,0x6e,0x1e,0x5e,0x3e,0x7e,
    0x01,0x41,0x21,0x61,0x11,0x51,0x31,0x71,0x09,0x49,0x29,0x69,0x19,0x59,0x39,0x79,
    0x05,0x45,0x25,0x65,0x15,0x55,0x35,0x75,0x0d,0x4d,0x2d,0x6d,0x1d,0x5d,0x3d,0x7d,
    0x03,0x43,0x23,0x63,0x13,0x53,0x33,0x73,0x0b,0x4b,0x2b,0x6b,0x1b,0x5b,0x3b,0x7b,
    0x07,0x47,0x27,0x67,0x17,0x57,0x37,0x77,0x0f,0x4f,0x2f,0x6f,0x1f,0x5f,0x3f,0x7f,
    0x80,0xc0,0xa0,0xe0,0x90,0xd0,0xb0,0xf0,0x88,0xc8,0xa8,0xe8,0x98,0xd8,0xb8,0xf8,
    0x84,0xc4,0xa4,0xe4,0x94,0xd4,0xb4,0xf4,0x8c,0xcc,0xac,0xec,0x9c,0xdc,0xbc,0xfc,
    0x82,0xc2,0xa2,0xe2,0x92,0xd2,0xb2,0xf2,0x8a,0xca,0xaa,0xea,0x9a,0xda,0xba,0xfa,
    0x86,0xc6,0xa6,0xe6,0x96,0xd6,0xb6,0xf6,0x8e,0xce,0xae,0xee,0x9e,0xde,0xbe,0xfe,
    0x81,0xc1,0xa1,0xe1,0x91,0xd1,0xb1,0xf1,0x89,0xc9,0xa9,0xe9,0x99,0xd9,0xb9,0xf9,
    0x85,0xc5,0xa5,0xe5,0x95,0xd5,0xb5,0xf5,0x8d,0xcd,0xad,0xed,0x9d,0xdd,0xbd,0xfd,
    0x83,0xc3,0xa3,0xe3,0x93,0xd3,0xb3,0xf3,0x8b,0xcb,0xab,0xeb,0x9b,0xdb,0xbb,0xfb,
    0x87,0xc7,0xa7,0xe7,0x97,0xd7,0xb7,0xf7,0x8f,0xcf,0xaf,0xef,0x9f,0xdf,0xbf,
0xff };

void ledCmd(uint8_t x) { // Issue command to LED backback driver
  Wire.beginTransmission(I2C_ADDR);
  Wire.write(x);
  Wire.endTransmission();
}

void clear(void) { // Clear display buffer
  Wire.beginTransmission(I2C_ADDR);
  for(uint8_t i=0; i<17; i++) Wire.write(0);
  Wire.endTransmission();
}

void setup() {
  power_timer1_disable();    // Disable unused peripherals
  power_adc_disable();       // to save power
  PCMSK |= _BV(PCINT1);      // Set change mask for pin 1
  Wire.begin();              // I2C init
  clear();                   // Blank display
  ledCmd(0x21);              // Turn on oscillator
  ledCmd(0xE0 | BRIGHTNESS); // Set brightness
  ledCmd(0x81);              // Display on, no blink
}

uint8_t rep = REPS;

void loop() {
```

```
  for(int i=0; i<sizeof(anim); i) { // For each frame...
    Wire.beginTransmission(I2C_ADDR);
    Wire.write(0);                  // Start address
    for(uint8_t j=0; j<8; j++) {    // 8 rows...
      Wire.write(pgm_read_byte(&reorder[pgm_read_byte(&anim[i++])]));
      Wire.write(0);
    }
    Wire.endTransmission();
    delay(pgm_read_byte(&anim[i++]) * 10);
  }

  if(!--rep) {              // If last cycle...
    ledCmd(0x20);           // LED matrix in standby mode
    GIMSK = _BV(PCIE);      // Enable pin change interrupt
    power_all_disable();    // All peripherals off
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sleep_enable();
    sei();                  // Keep interrupts enabled
    sleep_mode();           // Power down CPU (pin 1 will wake)
    // Execution resumes here on wake.
    GIMSK = 0;              // Disable pin change interrupt
    rep   = REPS;           // Reset animation counter
    power_timer0_enable(); // Re-enable timer
    power_usi_enable();     // Re-enable USI
    Wire.begin();           // Re-init I2C
    clear();                // Blank display
    ledCmd(0x21);           // Re-enable matrix
  }
}

ISR(PCINT0_vect) {} // Button tap
```

We're not done yet!

Now we'll create the graphics for some alien creatures. You can scroll down if you just want to finish the pendant with our sample graphics, or read on for the geeky code details...

Normally when using these matrices we recommend using the Adafruit LED Backpack Library (https://adafru.it/aLI). Because library installation is often a trouble spot, this code minimizes extra library use and instead needs to do a few things "raw," and it's a bit intimidating as a result. So it's okay just to copy and paste the code and proceed to the next step if you prefer.

The code starts by disabling the chip's Timer1 and analog-to-digital converter to save a little power and extend battery life; they're not used by this program. Then it initializes the HT16K33 matrix driver chip using the Arduino "Wire" library for the I^2C protocol, clearing the image memory, setting the brightness and enabling the display (brightness is set with a #define near the top of the code...lower numbers are dimmer, but improve battery life).

The program then loops one or more times, reading animation frames from flash

memory (we'll explain that on the next page), issuing the bitmap data to the matrix driver and displaying each image for a short period. The big table lookup is because the matrix columns aren't wired in-order on the Backpack board; this re-orders the bits in memory to match the column order.

At the end of the sequence, both the LED matrix driver and the CPU are put into a low-power state to help preserve battery life. A pin-change interrupt is enabled on pin #1 that will wake the CPU from sleep and restart the animation. This button is optional; you can use the onboard reset button as well (though it will have a brief delay as the bootloader starts).

Second File

Animation data goes in a separate file. This way it can be edited or replaced without having to rummage through the code; it's a bit less intimidating.

At the right side of the editor window, click the triangle, select "New Tab" and type an im.h as the filename. Then copy and paste this next chunk of code:

```
// SPDX-FileCopyrightText: 2018 Mikey Sklar for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// Animation data for Trinket/Gemma + LED matrix backpack jewelry.
// Edit this file to change the animation; it's unlikely you'll need
// to edit the source code.

#define REPS 3 // Number of times to repeat the animation loop (1-255)

const uint8_t PROGMEM anim[] = {

  // Animation bitmaps.  Each frame of animation MUST contain
  // 8 lines of graphics data (there is no error checking for
  // length).  Each line should be prefixed with the letter 'B',
  // followed by exactly 8 binary digits (0 or 1), no more,
  // no less (again, no error checking).  '0' represents an
  // 'off' pixel, '1' an 'on' pixel.  End line with a comma.
  B00011000, // This is the first frame for alien #1
  B00111100, // If you squint you can kind of see the
  B01111110, // image in the 0's and 1's.
  B11011011,
  B11111111,
  B00100100,
  B01011010,
  B10100101,
  // The 9th line (required) is the time to display this frame,
  // in 1/100ths of a second (e.g. 100 = 1 sec, 25 = 1/4 sec,
  // etc.).  Range is 0 (no delay) to 255 (2.55 seconds).  If
  // longer delays are needed, make duplicate frames.
  25, // 0.25 seconds

  B00011000, // This is the second frame for alien #1
  B00111100,
  B01111110,
  B11011011,
  B11111111,
```

```
  B00100100,
  B01011010,
  B01000010,
  25, // 0.25 second delay

  // Frames 3 & 4 for alien #1 are duplicates of frames 1 & 2.
  // Rather than list them 'the tall way' again, the lines are merged here...
  B00011000, B00111100, B01111110, B11011011, B11111111, B00100100, B01011010,
B10100101, 25,
  B00011000, B00111100, B01111110, B11011011, B11111111, B00100100, B01011010,
B01000010, 25,

  B00000000, // First frame for alien #2
  B00111100,
  B01111110,
  B11011011,
  B11011011,
  B01111110,
  B00100100,
  B11000011,
  25, // 0.25 second delay

  B00111100, // Second frame for alien #2
  B01111110,
  B11011011,
  B11011011,
  B01111110,
  B00100100,
  B00100100,
  B00100100,
  25,

  // Frames 3 & 4 for alien #2 are duplicates of frames 1 & 2
  B00000000, B00111100, B01111110, B11011011, B11011011, B01111110, B00100100,
B11000011, 25,
  B00111100, B01111110, B11011011, B11011011, B01111110, B00100100, B00100100,
B00100100, 25,

  B00100100, // First frame for alien #3
  B00100100,
  B01111110,
  B11011011,
  B11111111,
  B11111111,
  B10100101,
  B00100100,
  25,

  B00100100, // Second frame for alien #3
  B10100101,
  B11111111,
  B11011011,
  B11111111,
  B01111110,
  B00100100,
  B01000010,
  25,

  // Frames are duplicated as with prior aliens
  B00100100, B00100100, B01111110, B11011011, B11111111, B11111111, B10100101,
B00100100, 25,
  B00100100, B10100101, B11111111, B11011011, B11111111, B01111110, B00100100,
B01000010, 25,

  B00111100, // First frame for alien #4
  B01111110,
  B00110011,
  B01111110,
  B00111100,
```

```
  B00000000,
  B00001000,
  B00000000,
  12, // ~1/8 second delay

  B00111100, // Second frame for alien #4
  B01111110,
  B10011001,
  B01111110,
  B00111100,
  B00000000,
  B00001000,
  B00001000,
  12,

  B00111100, // Third frame for alien #4 (NOT a repeat of frame 1)
  B01111110,
  B11001100,
  B01111110,
  B00111100,
  B00000000,
  B00000000,
  B00001000,
  12,

  B00111100, // Fourth frame for alien #4 (NOT a repeat of frame 2)
  B01111110,
  B01100110,
  B01111110,
  B00111100,
  B00000000,
  B00000000,
  B00000000,
  12,

  // Frames 5-8 are duplicates of 1-4, lines merged for brevity
  B00111100, B01111110, B00110011, B01111110, B00111100, B00000000, B00001000,
B00000000, 12,
  B00111100, B01111110, B10011001, B01111110, B00111100, B00000000, B00001000,
B00001000, 12,
  B00111100, B01111110, B11001100, B01111110, B00111100, B00000000, B00000000,
B00001000, 12,
  B00111100, B01111110, B01100110, B01111110, B00111100, B00000000, B00000000,
B00000000, 12,
};
```

From the Tools→Board menu, select Adafruit Trinket 8 MHz or Adafruit Gemma as appropriate. Connect the USB cable between the computer and board, press the reset button, then click the upload button (right arrow icon) in the Arduino IDE. In a moment you should get a light show from the LEDs. (If it doesn't, check your wiring against the schematics. If the code refuses to compile, most likely the anim.h file is mis-named.

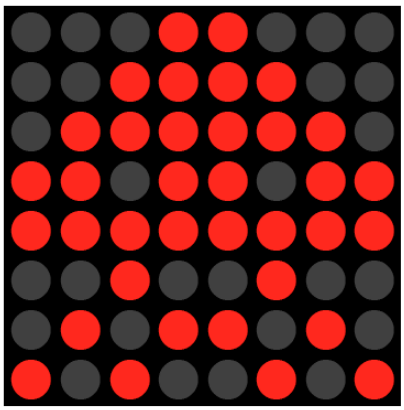
You'll see an animation sequence of four Space Invaders-style aliens that repeats three times, then shuts off. To see it again, tap the reset button. (If the USB cable is still connected, there's a long delay before it starts again — this is normal, the delay is much shorter when running off the battery or using the optional replay button.)

To change the animation in the future, you only need to edit or replace the contents of anim.h; it's rare that you'll need to edit the main source code. Ambitious programmers could write a program to convert an animated GIF into a replacement an im.h file, but for now it's necessary to edit this file manually.

There are 9 lines for each frame of animation; 8 of these are bitmap data, and the 9th line is the delay time. Each bitmap line consists of the letter 'B' followed by 8 binary digits (0 or 1), where '0' (zero, not uppercase 'o') represents an "off" pixel, '1' (one) an "on" pixel, and ends with a comma. The delay is given in 1/100ths of a second; 100 = 1 second, 25 = 1/4 second and so forth. The delay range is from 0 to 255; if you need longer delays, make duplicate frames.

You can almost see the bitmap image in the text:

```
B00011000,
B00111100,
B01111110,
B11011011,
B11111111,
B00100100,
B01011010,
B10100101,
```



You don't have to represent every row on its own line like this, but it makes the image much easier to visualize. You can see a few places in the file above where we've smooshed all 9 lines together to save space in the vertical direction. These frames are copies of others; we already know what they look like.

After editing, press reset on the board and upload as you did before.

If the program refuses to compile after editing anim.h, it's most likely one of the following:

- Missing comma at end of line
- Missing B (upper case) at start of line
- Too many or too few digits on a line, or characters other than 0 (zero) and 1 (one)
- Spaces between characters

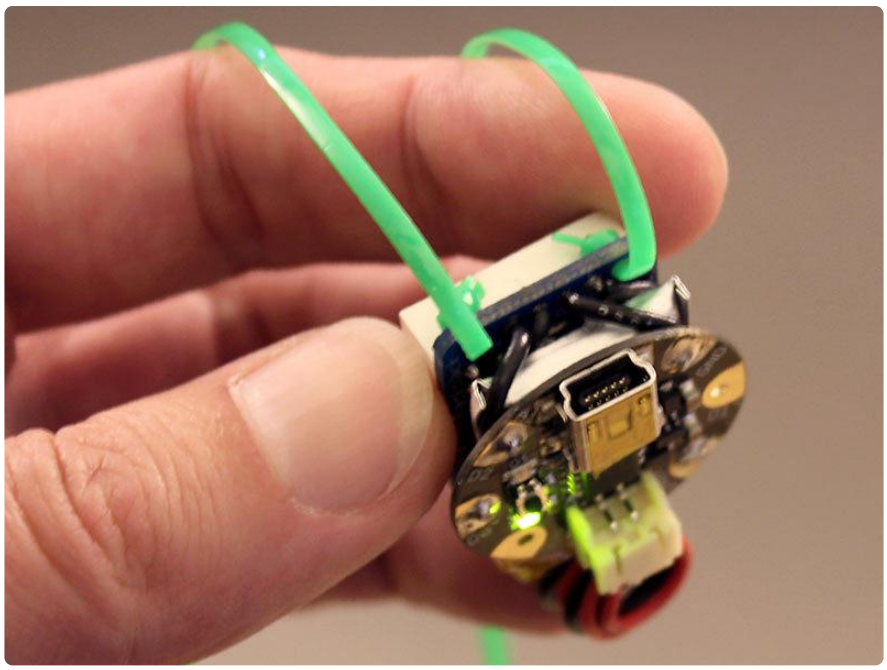
There's enough room in the chip for about 320 frames of animation; anything less is fine, of course.

Finishing Up

Connect the LiPo battery to make sure that everything runs, then stack the components and fold the wires around so that nothing is protruding. If using the onboard reset button, make sure it's on the back side where you can reach it, not blocked by the LED backpack or battery. Other than your wires, there should be no conductive parts touching between the microcontroller board and LED backpack.

The layers can be held together with a couple small pieces of foam tape, dabs of hot-melt adhesive, epoxy, etc. If you added a replay button you can encase the whole thing in a plastic bubble (such as toy vending machines dispense) with just the button protruding.

The mounting holes on the LED backpack can be used for attaching a lanyard. Plastic lace from a craft store is one option. Do not use a metal chain for this...it's conductive and could cause an electrical short as the pendant shifts around. Another option is a pin back...we have a magnetic type (http://adafru.it/1170) in the shop, or you can find the regular variety at most sewing or craft shops...again, be careful with this bridging electrical contacts.



The little battery should last all day, depending how often you activate it. To recharge, unplug the LiPo cell from the Gemma (or JST plug if using a Trinket) and connect it to the Micro Lipo USB charger. The battery does not charge when the microcontroller is connected to USB.