Pac Man Pixel Suspenders

Created by Becky Stern

https://learn.adafruit.com/pac-man-pixel-suspenders

Last updated on 2023-08-29 02:20:08 PM EDT
Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>Tools &amp; Supplies</td>
<td>7</td>
</tr>
<tr>
<td>Circuit Diagram</td>
<td>10</td>
</tr>
<tr>
<td>Stitch Circuit</td>
<td>11</td>
</tr>
<tr>
<td>Program it</td>
<td>16</td>
</tr>
<tr>
<td>Wear it!</td>
<td>21</td>
</tr>
</tbody>
</table>
Overview

Blinken-braces! Add color-changing LEDs to a pair of suspenders and hold your pants up in style. 30 NeoPixels are sewn to these suspenders, powered by a FLORA main board running a dazzling Pac Man-inspired animation. The battery pack goes in your pocket! Read on to make your own pixel suspenders.

Project made with wearables assistant Risa Rose! Thanks to Shelly Lynch-Sparks and Johngineer for modeing.
Tools & Supplies

For this project you will need:

- 30 FLORA NeoPixel RGB LEDs (http://adafruit.it/1260)
- FLORA main board (http://adafruit.it/659)
- 3-ply conductive thread (http://adafruit.it/641)
- 3 x AAA battery holder with JST connector (http://adafruit.it/727)
- 1 pair suspenders (http://adafruit.it/615)
- sewing needle (http://adafruit.it/615)
- sewing machine and thread to match suspenders
- sharp scissors
- measuring tape or ruler
- tailor's chalk

Conductive thread () will carry signal and current from the main board to the pixels.
You'll use a needle (http://adafru.it/615) and to stitch up the circuit.

Clear nail polish or fray check to seal knots

A sewing machine helps make this project fun and easy.
Sharp scissors are a must!

You will need a good quality basic multimeter that can measure voltage and continuity.

Click here to buy a basic multimeter. (http://adafru.it/71)

Click here to buy a top of the line multimeter. (http://adafru.it/308)

Click here to buy a pocket multimeter. (http://adafru.it/850)

Don't forget to learn how to use your multimeter too! ()

To mark out your circuit use a water-soluble embroidery marker or tailor's chalk.
30 FLORA NeoPixels connected to FLORA main board pin D6, with power from VBATT and ground running along the length of the suspenders, around one shoulder to the other. Each pixel's outgoing arrow is connected to each next pixel's incoming arrow with conductive thread.
Stitch Circuit

Adjust your suspenders to fit you-- they will become less stretchy once the circuit is sewn on.

Mark where your pixels will go with some tailor's chalk. We're using 30 pixels in total, marking every two inches. The FLORA main board goes at one end.

Line up the two front bands of the suspenders and transfer marks across.

Use a sewing machine set to a zigzag stitch to affix three strands of conductive thread to the suspenders along one side. Following the circuit diagram on the previous page, create a power rail extending up one side of the suspenders, around the back, and then down the other front band.

Leave tails at the ends to stitch to your components.
Repeat with the ground rail, zigzagging three strands of conductive thread along the other edge of the suspenders. See how the center back pixel acts as a pivot point for these two conductors.

These two beefy sections of thread ensure ample power delivery to the super-bright NeoPixels.
Pick up the tails of thread (all through one needle) and stitch the power line to VBATT on FLORA, and the ground line to GND--again refer to the circuit diagram on the previous page. Tie off, weave in, and snip the tails.
Now it's time to start adding pixels. Stitch small data connections between each pixels' arrows (all arrows should face away from FLORA)—the first one is connected to D6.

After all the data connections have been stitched, knotted, and sealed (check out our guide for working with conductive thread), use more conductive thread to stitch in line with the power bus. Stitch under the zigzags, stopping at each pixel to wrap around the pad marked +. When your thread gets short, just interleave it with the threads under the zigzags and cut off the tail. Pick up a new piece of thread and continue on.

Repeat with the ground rail, connecting it all along the suspenders and to pads on the pixels marked -.

This method of powering the pixels is similar to the Chameleon Scarf, and results in fewer knots to seal and more effective power delivery along long lines of pixels.
After checking your conductive thread circuit for shorts, upload the NeoPixel test code (refer to the FLORA NeoPixel tutorial if necessary). If any of your connections are flaky, reinforce them with conductive thread. Once all your pixels are working flawlessly, load the code below, which will make a chasing pac-man-inspired animation appear across the suspenders.

```c
#include <Adafruit_NeoPixel.h>

#define LED_PIN 6
#define N_LEDS 30
#define N_SPRITES (N_LEDS - 5) // 4 ghosts + 1 mouth
#define PILL_INDEX (N_LEDS - 5) // Position of 'power pill' along strip
#define PILL_R 255 // Power pill is slightly pink-ish,
#define PILL_G 184 // not pure white.
#define PILL_B 151
#define DOT_R (PILL_R / 3) // Dots are same hue as power pill,
#define DOT_G (PILL_G / 3) // just dimmer.
#define DOT_B (PILL_B / 3)
#define DOT_FADE_SPEED 2 // New screenful of dots fades in this fast

const uint8_t gamma[] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
  0x02, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x05,
  0x05, 0x06, 0x06, 0x06, 0x06, 0x07, 0x07, 0x07, 0x07, 0x08, 0x08, 0x08, 0x08, 0x08,
  0x09, 0x09, 0x09, 0x09, 0x0A, 0x0A, 0x0A, 0x0A, 0x0B, 0x0B, 0x0B, 0x0B, 0x0C, 0x0C,
  0x0D, 0x0D, 0x0D, 0x0D, 0x0E, 0x0E, 0x0F, 0x0F, 0x10, 0x10, 0x10,
  0x11, 0x11, 0x12, 0x12, 0x13, 0x13, 0x14, 0x14, 0x15, 0x15, 0x16, 0x16, 0x17, 0x18,
  0x18, 0x19, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1D, 0x1E, 0x1F, 0x20, 0x20, 0x21, 0x22,
  0x23, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2B, 0x2C, 0x2D,
  0x2D, 0x2E, 0x2F, 0x30, 0x30, 0x31, 0x32, 0x32,
  0x33, 0x34, 0x35, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3C, 0x3D, 0x3E, 0x3F,
  0x40, 0x40, 0x42, 0x43, 0x44, 0x44, 0x45, 0x46, 0x48, 0x4A, 0x4B, 0x4C, 0x4D,
  0x4E, 0x4F, 0x50, 0x51, 0x52, 0x53, 0x55, 0x55, 0x56, 0x57, 0x59,
  0x5A, 0x55, 0x55, 0x5F, 0x60, 0x62, 0x63, 0x65, 0x66, 0x68, 0x69, 0x6B, 0x6D,
  0x6E, 0x6E, 0x70, 0x72, 0x73, 0x75, 0x77, 0x78, 0x7A, 0x7C, 0x7E, 0x7F,
  0x81, 0x83, 0x85, 0x87, 0x89, 0x8A, 0x8C, 0x8E,
  0x90, 0x92, 0x94, 0x96, 0x98, 0x9A, 0x9C, 0x9E, 0xA0, 0xA2, 0xA4, 0xA7,
  0xA9, 0xAB, 0xA0, 0xA0, 0xA0, 0xAF, 0xB1, 0xB4, 0xB6, 0xBB, 0xBA, 0xBD,
  0xBF, 0xC0, 0xC4, 0xC6, 0xCB, 0xCB, 0xCD, 0xD0, 0xD2, 0xD5,
  0xD7, 0xDA, 0xDC, 0xDF, 0xE1, 0xE4, 0xE7, 0xE9, 0xEF, 0xF1, 0xF4, 0xF7,
  0xF9, 0xFC, 0xFF
};
```
Adafruit_NeoPixel strip = Adafruit_NeoPixel(N_LEDS, LED_PIN, NEO_GRB + NEO_KHZ800);

// 100% of animation logic is in a timer interrupt, hence all globals are volatile.

// Lots of sub-pixel smoothing shenanigans are going on in an attempt to better
// translate the smooth animation of the original game into this coarse 1D format.

// Image is buffered independently of the NeoPixel color data, to allow for
// blending, gamma correction and horizontal flip on alternate passes.
volatile uint32_t pixel[N_LEDS][3];

// Game character sprites are always 1 pixel wide here to simplify certain code.
// Horizontal coordinates/speeds are in 1/256 pixel units.
volatile struct {
    int16_t x;               // Current position
    int16_t prevX;           // Position in prior frame
    int8_t  speed;           // Color &amp; opacity
    uint8_t r, g, b, a;      // Color &amp; opacity
    uint8_t moveDuringPause; // If set, sprite is immune to animation pauses
} sprite[N_SPRITES];
#define SPRITE_RED    0 // Ghost indices
#define SPRITE_PINK   1
#define SPRITE_CYAN   2
#define SPRITE_ORANGE 3
#define SPRITE_MOUTH  4 // Mouth index
volatile uint8_t
    dotState[N_LEDS], // 1 = dot enabled, 0 = eaten
    pillBlink = 0, // Power pill blink counter
    pauseCounter = 0, // Counter for 'chomp-a-ghost' pause
    dotFade,          // Counter as dots fade into view
    flip = 1; // If bit 1 set, flip playfield
volatile uint16_t
    ghostTime = 0; // Countdown timer for blue ghosts

//----------------------------------------------------------------------------------
void setup() {
    memset((void *)pixel, 0, sizeof(pixel));       // Clear screen buffer
    memset((void *)dotState, 0, sizeof(dotState)); // Set all dots to 'off'
    reset_animation();                             // Init all other values
    strip.begin();

    // Set up Timer1 interrupt: Mode 14 (fast PWM, top=ICR1), 1:8 prescale, OC1A/B off
    TCCR1A  = _BV(WGM11);
    TCCR1B  = _BV(WGM13) | _BV(WGM12) | _BV(CS11);
    ICR1   = F_CPU / 8 / 64; // 64 Hz
    TIMSK1 |= _BV(TOIE1);     // Enable Timer1 overflow interrupt
    sei();                    // Enable global interrupts
}

//----------------------------------------------------------------------------------
// This is called once at program start, and periodically each time the animation
// needs to reset. The playfield is flipped horizontally on each call, for variety.
void reset_animation() {
    uint8_t i, j, temp;

    sprite[SPRITE_RED].r = 255; // Red ghost color
    sprite[SPRITE_RED].g = 0;
    sprite[SPRITE_RED].b = 0;
    sprite[SPRITE_PINK].r = 255; // Pink ghost color
    sprite[SPRITE_PINK].g = 184;
    sprite[SPRITE_PINK].b = 222;
    sprite[SPRITE_CYAN].r = 0; // Cyan ghost color
sprite[SPRITE_CYAN].g = 255;  
sprite[SPRITE_CYAN].b = 222;  
sprite[SPRITE_ORANGE].r = 255;  // Orange ghost color  
sprite[SPRITE_ORANGE].g = 184;  
sprite[SPRITE_ORANGE].b = 71;  
sprite[SPRITE_MOUTH].r = 255;  // Mouth color  
sprite[SPRITE_MOUTH].g = 255;  
sprite[SPRITE_MOUTH].b = 0;  

// Ghost positions aren't perfectly spaced; they're a bit scrambled for  
// variety.  Mouth starts off left side, ghosts are further left of that.  
sprite[SPRITE_MOUTH].x = -400;  // Mouth position  
sprite[SPRITE_RED].x = sprite[SPRITE_MOUTH].x - 800;  // Red ghost position  
sprite[SPRITE_PINK].x = sprite[SPRITE_RED].x - 440;  // Pink ghost position  
sprite[SPRITE_CYAN].x = sprite[SPRITE_PINK].x - 500;  // Cyan ghost position  
sprite[SPRITE_ORANGE].x = sprite[SPRITE_CYAN].x - 530;  // Orange ghost position

for(i=0; i<N_SPRITES; i++) {  
sprite[i].prevX = sprite[i].x;  
sprite[i].moveDuringPause = false;  
sprite[i].a = 255;  // Opaque  
sprite[i].speed = 14;
}
sprite[4].speed = 13;  // Mouth moves a little slower when eating dots

flip++;  // Reverse layout of display  
// Flip residual screen dots to match new layout
for(i=0, j=N_LEDS-1; i<N_LEDS/2; i++, j--) {
  temp = dotState[i];  
dotState[i] = dotState[j];  
dotState[j] = temp;
}
dotFade = 0;  // Fade in new screenload of dots

ISR(TIMER1_OVF_vect) {
  uint8_t i;
  uint16_t fracX, weightL, weightR, inv, w;
  int8_t x1, x2;

  // First thing is to push data for the PRIOR frame to the strip.  This is done  
  // so the animation timing will be rock steady.  Some frames may take slightly  
  // different times to process than others, and putting the show() at the end  
  // would cause a slightly different refresh time for each.
  strip.show();

  // If new screenload of dots is fading into view, that's the only processing  
  // that occurs -- no sprite animation or collision detection, etc. happens.
  if(dotFade < 255) {
    for(i=0; i<N_LEDS; i++) {  // For each dot...
      if(dotState[i] == 1) {  // Already something here?
        pixel[i][0] = DOT_R;  // Yep, show uneaten 'on' dot.
        pixel[i][1] = DOT_G;
        pixel[i][2] = DOT_B;
      } else {  // Fake new dot into view
        if(i == PILL_INDEX) {  // Is it the power pill?
          if(pillBlink & 16) {  // Blinking 'on' right now?
            w = (uint16_t)dotFade + 1;  // Blend factor
            pixel[i][0] = (PILL_R * w) &gt;&gt; 8;
            pixel[i][1] = (PILL_G * w) &gt;&gt; 8;
            pixel[i][2] = (PILL_B * w) &gt;&gt; 8;
          }
        }
      }
    }
  }
}
} else {
    pixel[i][0] = pixel[i][1] = pixel[i][2] = 0; // Blink off
}
} else { // Regular dot, not power pill
    w = (uint16_t)dotFade + 1; // Blend factor
    pixel[i][0] = (DOT_R * w) &gt;&gt; 8;   
    pixel[i][1] = (DOT_G * w) &gt;&gt; 8;   
    pixel[i][2] = (DOT_B * w) &gt;&gt; 8;   
}
}
// Increment dot fade counter
if(dotFade &gt;= (255 - DOT_FADE_SPEED)) {  // About to hit/exceed max?
    dotFade = 255;  // Set to max value, we're done
    for(i=0; i&lt;N_LEDS; i++) dotState[i] = 1; // Make all dots edible
} else {
    dotFade += DOT_FADE_SPEED; // Not maxed, keep counting up
}
} else {

// Dots are 'active,' not fading into view. Do the full 'game' logic...

// Update sprite positions
for(i=0; i&lt;N_SPRITES; i++) {
    sprite[i].prevX = sprite[i].x;
    if((pauseCounter == 0) || sprite[i].moveDuringPause) {
        sprite[i].x += sprite[i].speed;
    }
}
if(pauseCounter) { // Is animation paused? (During a ghost chomp)
    if(--pauseCounter == 0) { // Count down to resume. Done?
        sprite[SPIRITE_MOUTH].a = 255; // Mouth = opaque
        for(i=0; i&lt;4; i++) { // Set non-opaque ghosts (being chomped) to eye color
            if(sprite[i].a == 255) continue; // Is opaque - don't change
            sprite[i].r = sprite[i].g = sprite[i].b = 255; // White
            sprite[i].a     = 96; // Mostly translucent
            sprite[i].speed = 48; // Eyes move fast
            sprite[i].moveDuringPause = 1; // Eyes move when others are paused
        }
    } else { // Animation isn't paused...do collision tests...
      // Calc X extents of each sprite since prior frame
      for(i=0; i&lt;N_SPRITES; i++) {
        if(sprite[i].x &gt;= sprite[i].prevX) {
            sprite[i].x1 = sprite[i].prevX;
            sprite[i].x2 = sprite[i].x;
        } else {
            sprite[i].x1 = sprite[i].x;
            sprite[i].x2 = sprite[i].prevX;
        }
    }
    // Check mouth motion against dots, eating any that are crossed
    x1 = sprite[SPIRITE_MOUTH].x1 &gt;&gt; 8;   
    x2 = sprite[SPIRITE_MOUTH].x2 &gt;&gt; 8;   
    if((x1 &gt;= 0) &amp;&amp; (x2 &lt; N_LEDS)) {
        if(x1 &lt;= 0) x1 = 0;
        if(x2 &gt;= N_LEDS) x2 = N_LEDS - 1;
        if(dotState[PILL_INDEX]) { // Is power pill uneaten?
            if((x1 &lt;= PILL_INDEX) &amp;&amp; (x2 &gt;= PILL_INDEX)) { // Eating it now?
                // Switch to 'chasing ghosts' mode. Turn around, a little faster
                sprite[SPIRITE_MOUTH].speed = -14;
                for(i=0;i&lt;4;i++) {
                    sprite[i].speed = -10; // Turn around, slow down
                }
            }
        }
    }
}
sprite[i].r = sprite[i].g = 33; // Blue ghost
sprite[i].b = 255;
}
ghostTime = 64 * 10; // Countdown timer
}
for(i=x1; i&lt;x2; i++) dotState[i] = 0; // Erase eaten dot(s)
}

// If in ghost-chasing mode, check mouth collision against eligible ghosts
if(ghostTime) {
    if(--ghostTime == 0) {
        reset_animation();
    } else {
        for(i=0; i&lt;4; i++) {
            if(sprite[i].a != 255) continue; // Don't compare against ghost eyes
            if((sprite[SPRITE_MOUTH].x1 &lt;= sprite[i].x2) && (sprite[SPRITE_MOUTH].x2 &gt;= sprite[i].x1)) { // Ate a ghost!
                pauseCounter = 48;          // Make animation pause (except eyes)
                sprite[SPRITE_MOUTH].a = 0; // Mouth disappears momentarily
                sprite[i].r = 0;
                sprite[i].g = sprite[i].b = 255;
                sprite[i].a = 220; // Ghost is replaced with 'point' display
            }
        }
        if(ghostTime &lt; (64 * 6)) { // Make ghosts blink blue/white toward end
            for(i=0; i&lt;4; i++) {
                if(sprite[i].a != 255) continue; // Ghost is in eye state; ignore
                if(ghostTime &amp; 32) {
                    sprite[i].r = sprite[i].g = 222;
                } else {
                    sprite[i].r = sprite[i].g = 33;
                }
            }
        }
    }
}

// Draw background dots into pixel[] array
for(i=0; i&lt;N_LEDS; i++) {
    if(dotState[i] == 1) {
        if(i == PILL_INDEX) {
            if(pillBlink &amp;&amp; 16) {
                pixel[i][0] = PILL_R;
                pixel[i][1] = PILL_G;
                pixel[i][2] = PILL_B;
            } else {
                pixel[i][0] = pixel[i][1] = pixel[i][2] = 0;
            }
        } else {
            pixel[i][0] = DOT_R;
            pixel[i][1] = DOT_G;
            pixel[i][2] = DOT_B;
        }
        if(i &lt; N_SOURCES) {
            pixel[i][0] = SOURCES[i][0];
            pixel[i][1] = SOURCES[i][1];
            pixel[i][2] = SOURCES[i][2];
        } else {
            pixel[i][0] = 0;
            pixel[i][1] = 0;
            pixel[i][2] = 0;
        }
    }
}

// Overlay sprites
for(i=0; i&lt;N_SOURCES; i++) {
    x1 = sprite[i].x &gt;&gt; 8; // Left pixel
    x2 = x1 + 1;           // Right pixel
    if((x2 &gt;= 0) &amp;&amp; (x1 &lt; N_LEDS)) { // Gross clipping
        fracX = sprite[i].x &amp; 255; // Sub-pixel position (0-255)
weightL = 1 + ((256 - (uint16_t)fracX) * sprite[i].a) >> 8; // Left pixel weight (1-256)
weightR = 1 + (1 + (uint16_t)fracX) * sprite[i].a) >> 8; // Right pixel weight (1-256)

if(x1 >= 0) {  // Process left pixel
    inv   = 257 - weightL; // 1-256
    pixel[x1][0] = ((sprite[i].r * weightL) + (pixel[x1][0] * inv)) >> 8;
    pixel[x1][1] = ((sprite[i].g * weightL) + (pixel[x1][1] * inv)) >> 8;
    pixel[x1][2] = ((sprite[i].b * weightL) + (pixel[x1][2] * inv)) >> 8;
}

if(x2 < N_LEDS) {  // Process right pixel
    inv   = 257 - weightR; // 1-256
    pixel[x2][0] = ((sprite[i].r * weightR) + (pixel[x2][0] * inv)) >> 8;
    pixel[x2][1] = ((sprite[i].g * weightR) + (pixel[x2][1] * inv)) >> 8;
    pixel[x2][2] = ((sprite[i].b * weightR) + (pixel[x2][2] * inv)) >> 8;
}

pillBlink++;

// Apply gamma correction / flip to pixel data while passing to NeoPixel lib...
for(i=0; i&lt;N_LEDS; i++) {
    x1 = (flip &amp; 1) ? N_LEDS - 1 - i : i;
    strip.setPixelColor(i,
        pgm_read_byte(&amp;gamma[pixel[x1][0]]),
        pgm_read_byte(&amp;gamma[pixel[x1][1]]),
        pgm_read_byte(&amp;gamma[pixel[x1][2]]));
}

---

**Wear it!**

Hey! Don’t dunk your suspenders in water while they’re on! If you need to wash them, remove the battery and hand wash in the sink. Allow to air dry thoroughly before reconnecting the battery!