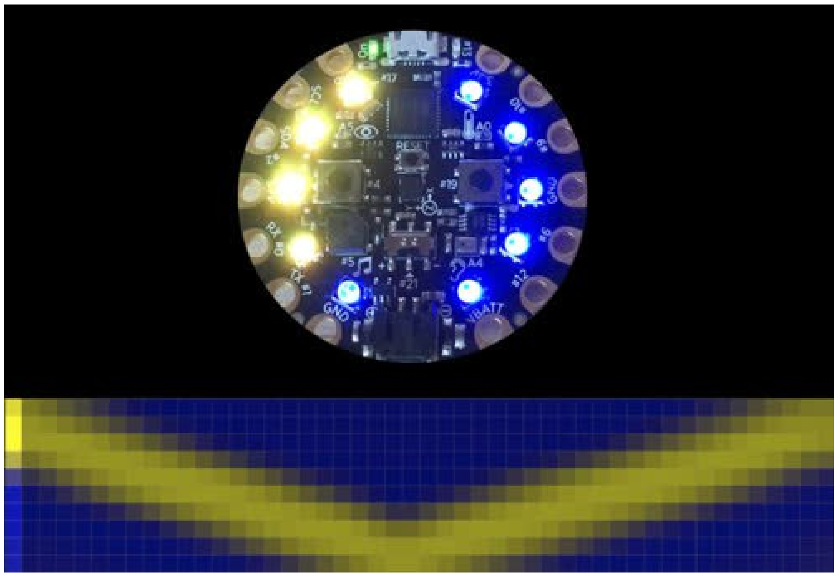
NeoAnim: Using Bitmaps to Animate NeoPixels on Circuit Playground

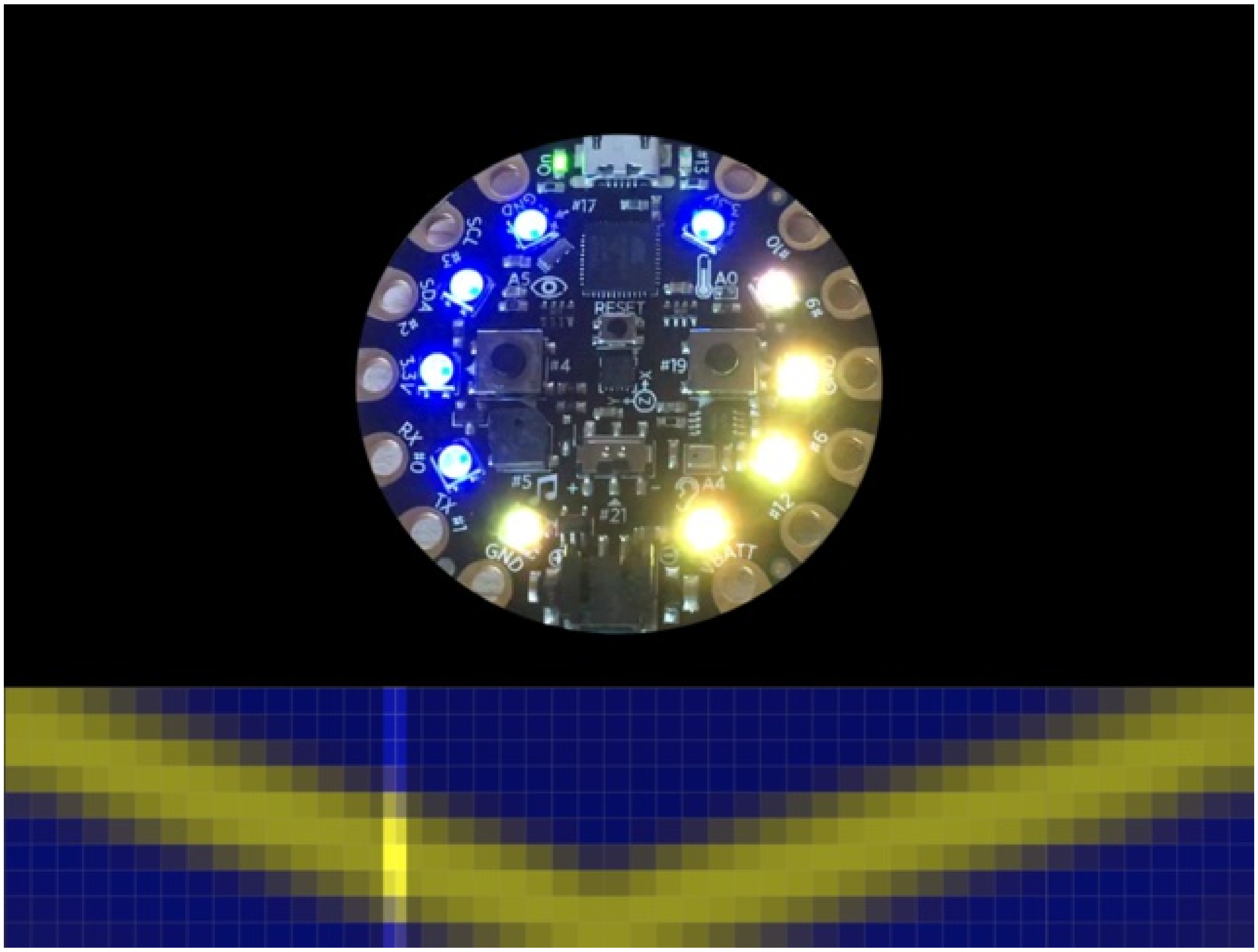
Created by John Park



Last updated on 2016-11-08 01:22:37 AM UTC

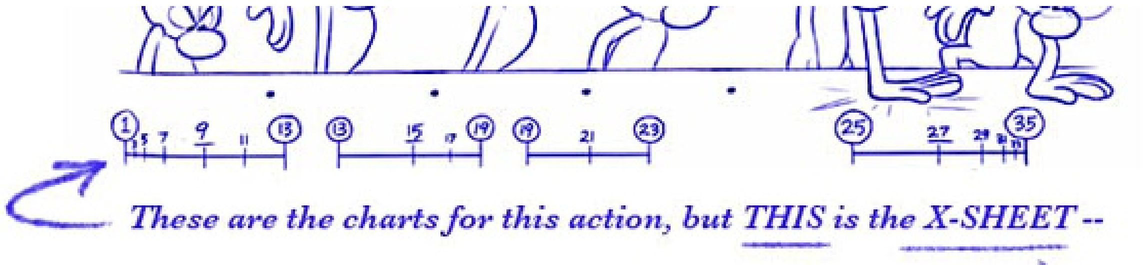
Guide Contents

From Pixels to NeoPixels

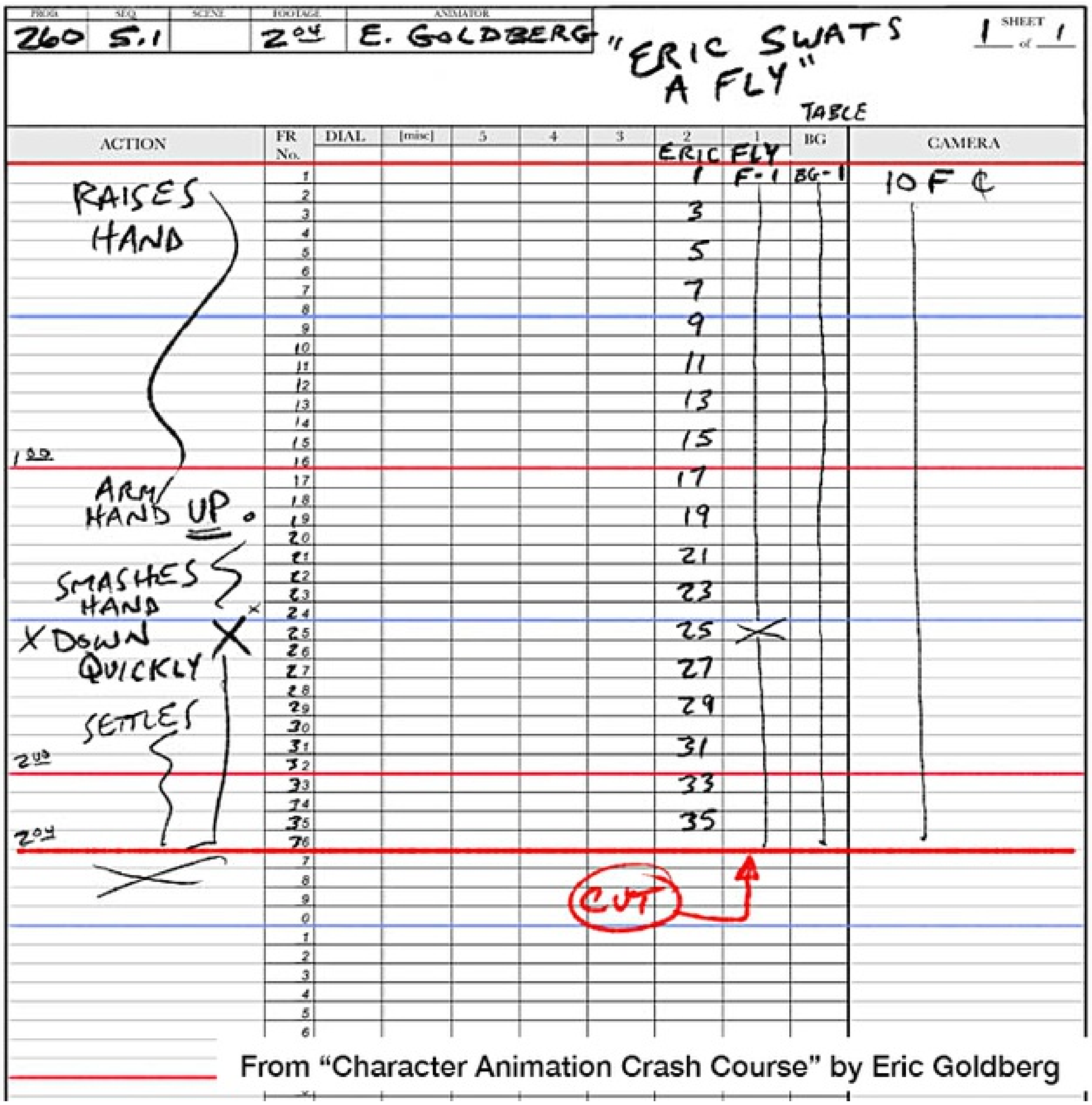


Lighting up NeoPixels on the Circuit Playground is fun and easy. But creating animation patterns with specific color, timing, and position can be difficult. Not any more! Using this technique and a small Python script, you can use a tiny bitmap image as a powerful control for making complex NeoPixel animations on your Circuit Playground.

Developed by our own Phil Burgess (http://adafru.it/iPc), this method has a lot in common with traditional hand-drawn animation timing charts and exposure sheets (also called dope sheets) in that a set of rows and columns are used to indicate an object's attributes over time.



These are the charts for this action, but THIS is the X-SHEET --



PROD.	SEQ.	SCENE	FOOTAGE	ANIMATOR
260	5.1		2 04	E. GOLDBERG

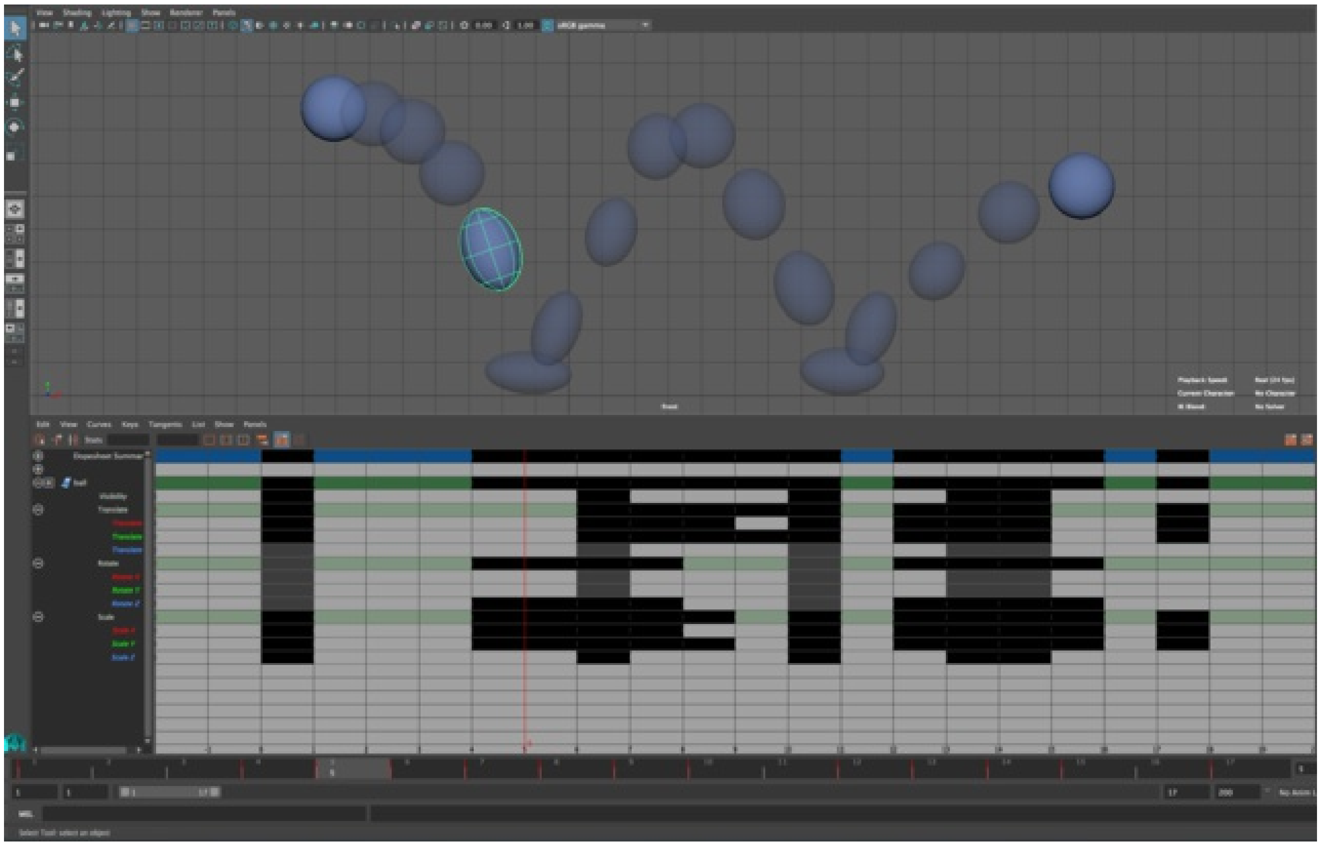
"ERIC SWATS A FLY"

SHEET 1 of 1

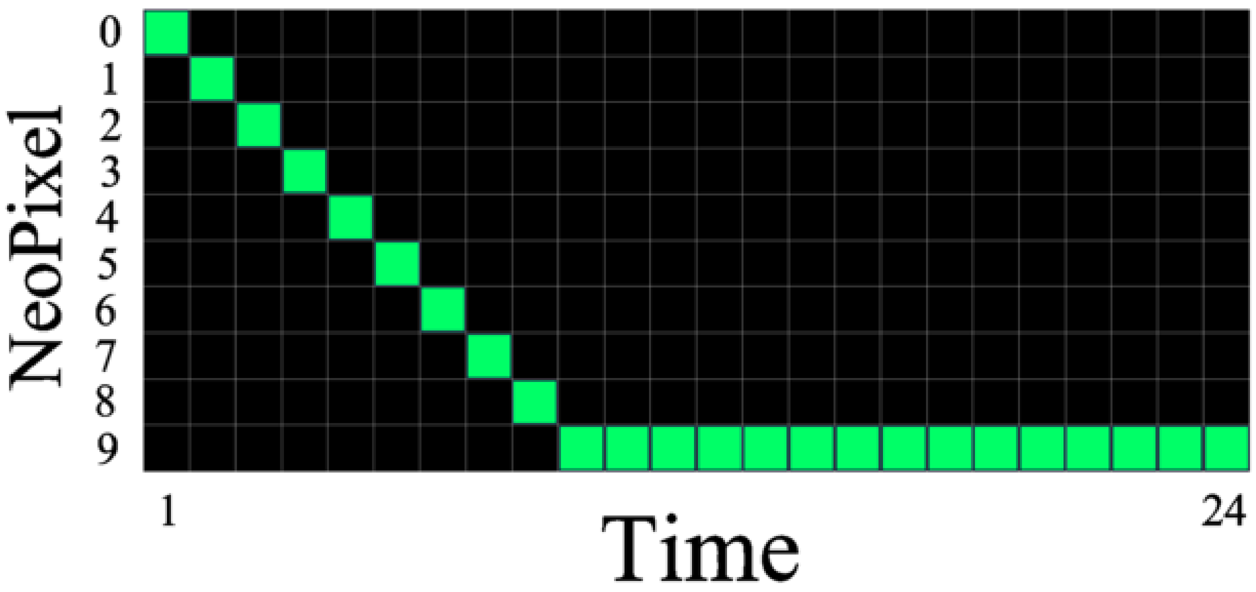
ACTION	FR No.	DIAL	[misc]	5	4	3	2	1	BG	CAMERA
							ERIC	FLY	TABLE	
RAISES HAND	1						1	F-1	BG-1	10 F ¢
	2									
	3						3			
	4									
	5						5			
	6									
	7						7			
	8									
	9						9			
	10									
	11						11			
	12									
	13						13			
	14									
	15						15			
1 00	16									
ARM/ HAND UP.	17						17			
	18									
	19						19			
	20									
SMASHES HAND	21						21			
	22									
	23						23			
X DOWN QUICKLY	24									
	25						25	X		
	26									
	27						27			
	28									
SETTLES	29						29			
	30									
2 03	31						31			
	32									
	33						33			
	34									
2 04	35						35			
	36									
X	7									
	8						CUT			
	9									
	0									
	1									
	2									
	3									
	4									
	5									
	6									

From "Character Animation Crash Course" by Eric Goldberg

Animation spreadsheets made leap from traditional animation to CG animation, and versions of it are still in use today in feature animation studio software such as Autodesk Maya and Pixar's in-house system, Presto.

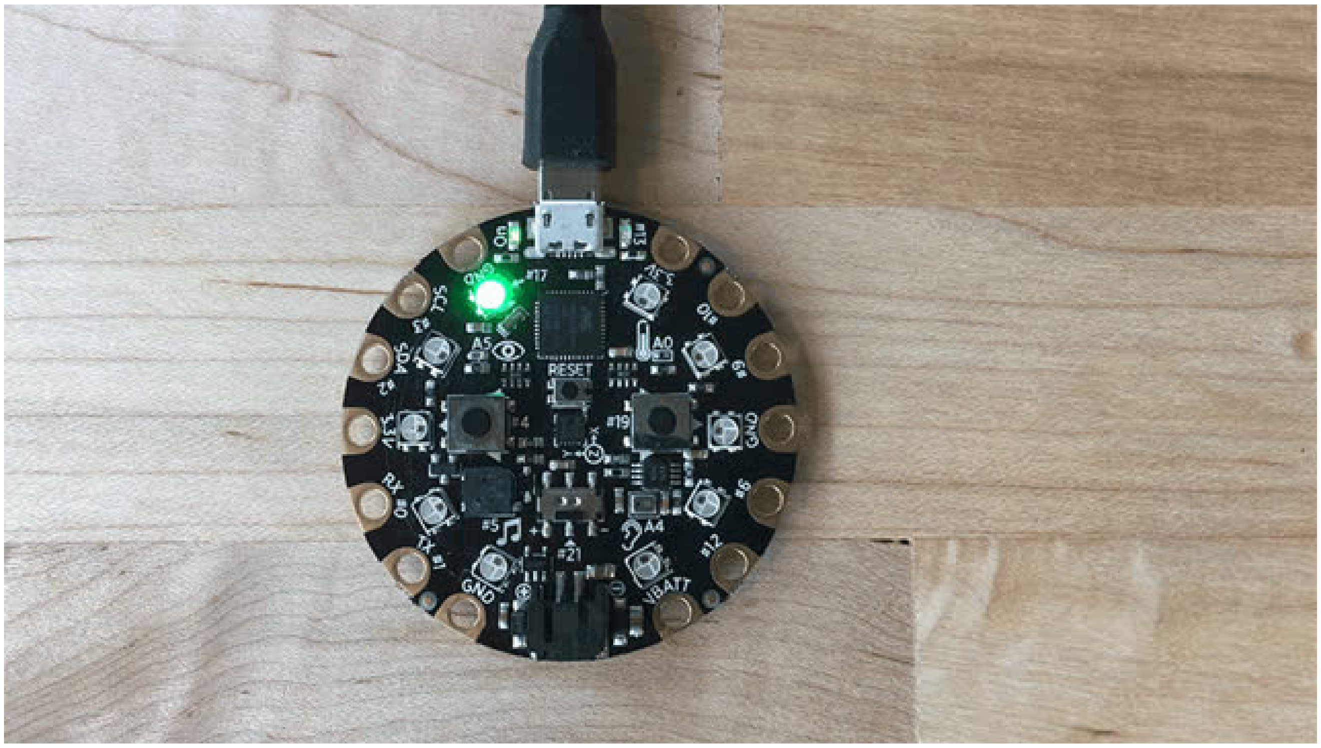


In this example of animating a ball bounce in Maya, note the dope sheet interface where the translation, rotation, and scale values of the ball are the attributes being indicated on the vertical chart axis, and the time in frames are the horizontal axis. This is very similar to our bitmap-to-NeoPixel animaiton system.

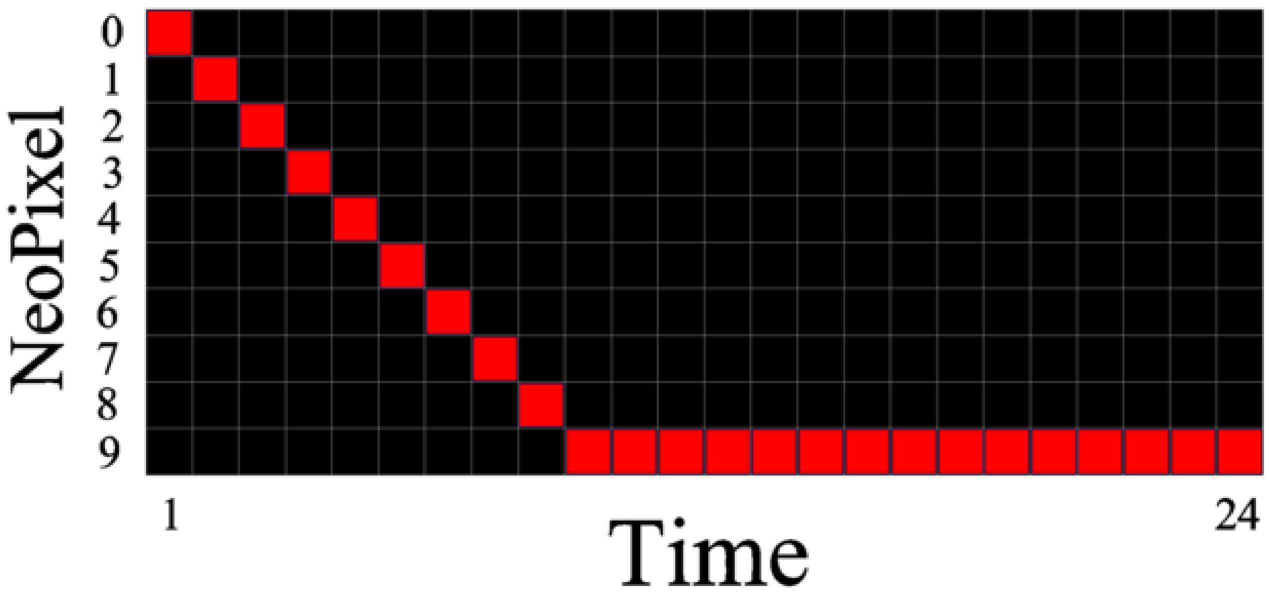


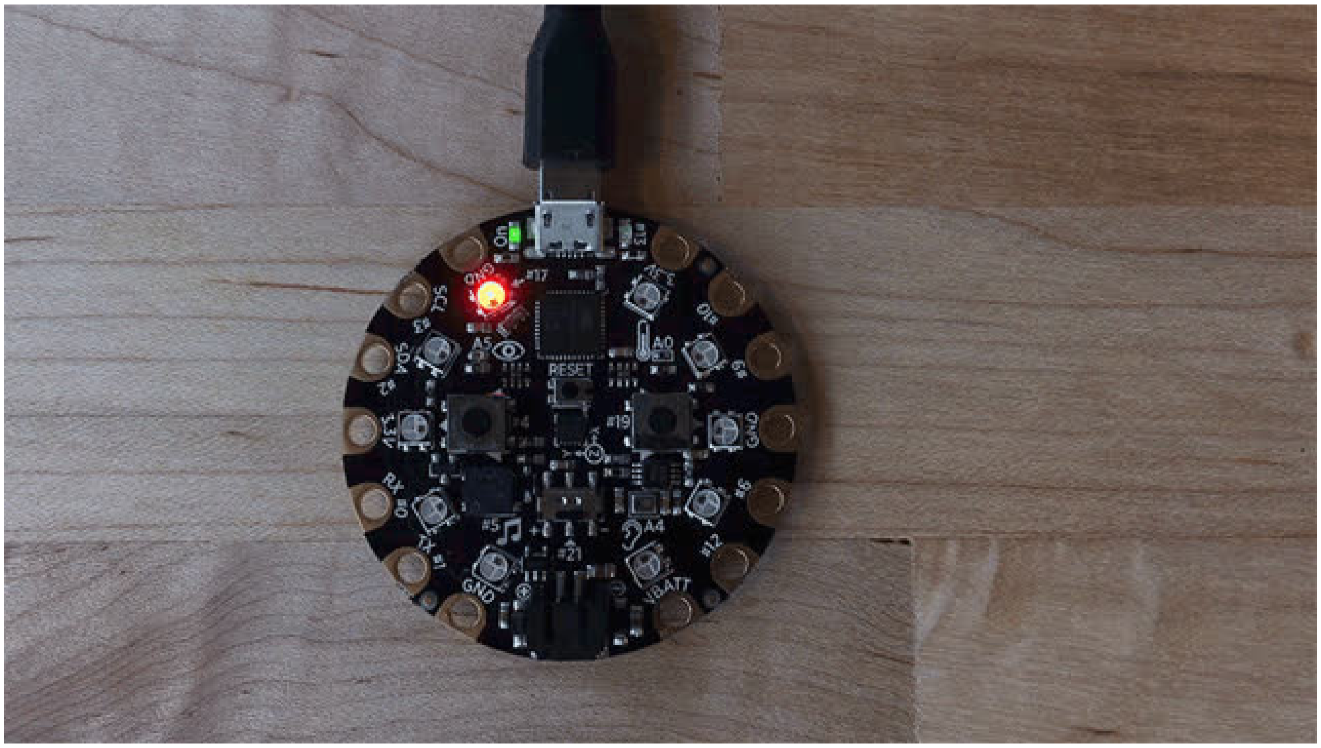
In this case, rows indicate each of the ten NeoPixels of the Circuit Playground, and columns represent frames of animation in time. The color of a particular cell in the grid represents what color to display that NeoPixel at that frame in time.

Here's an example: This is a one second loop of animation divided into 24 frames per second (24 FPS) which is the standard for traditional film and animation. In the image above, the ten NeoPixels (numbered 0-9) will each be lit green for one frame of time until NeoPixel 9 is lit, which then stays lit for the rest of the animation loop.

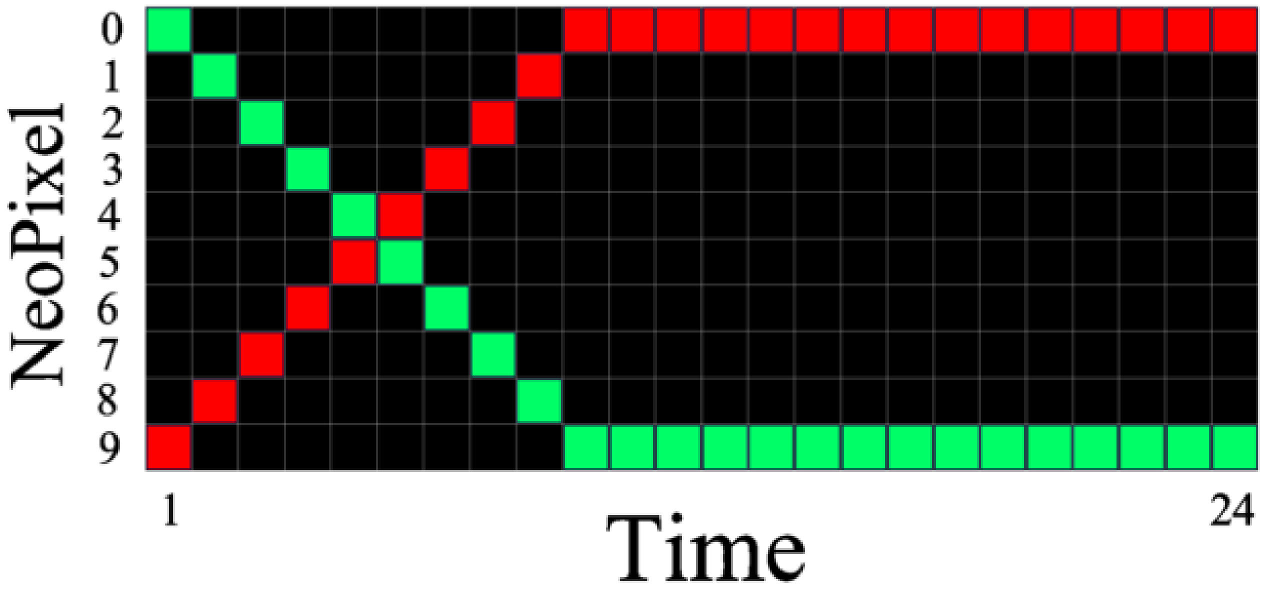


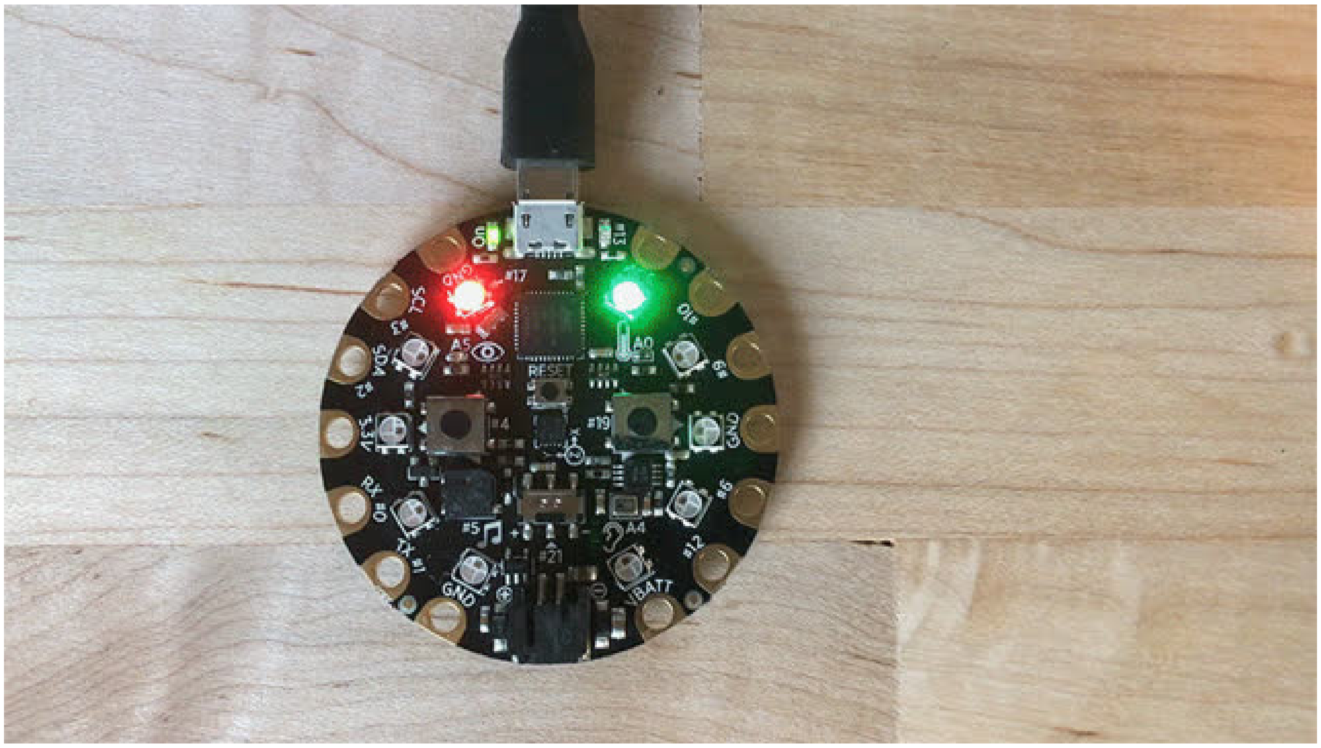
Here's what happens when the pixels are changed to red.



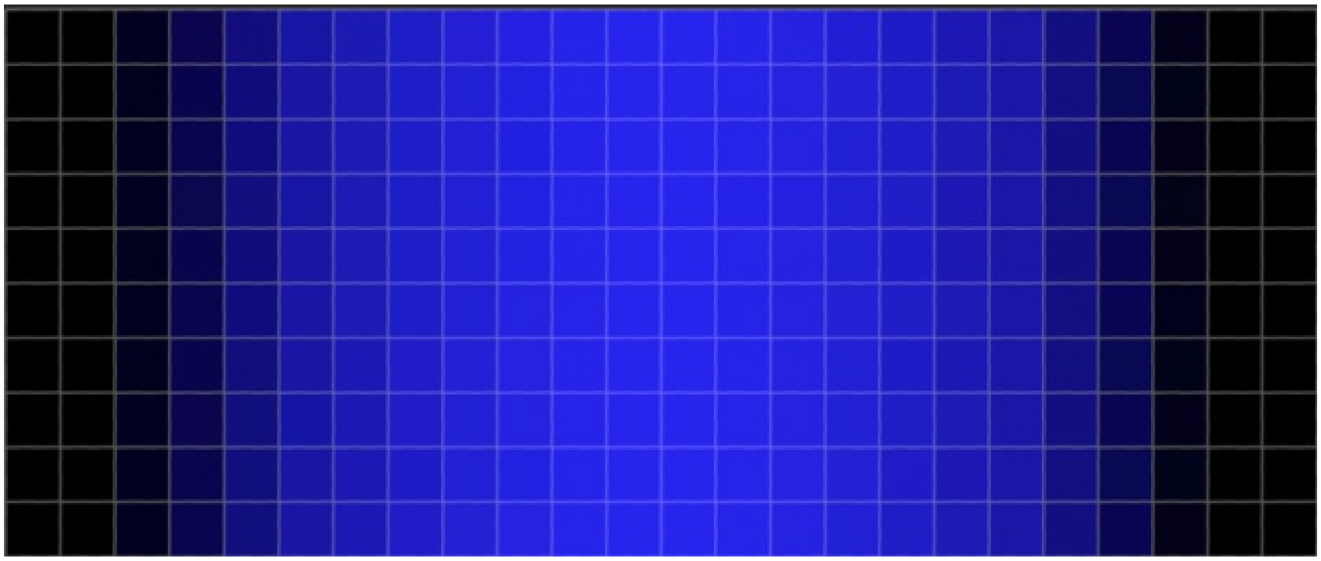


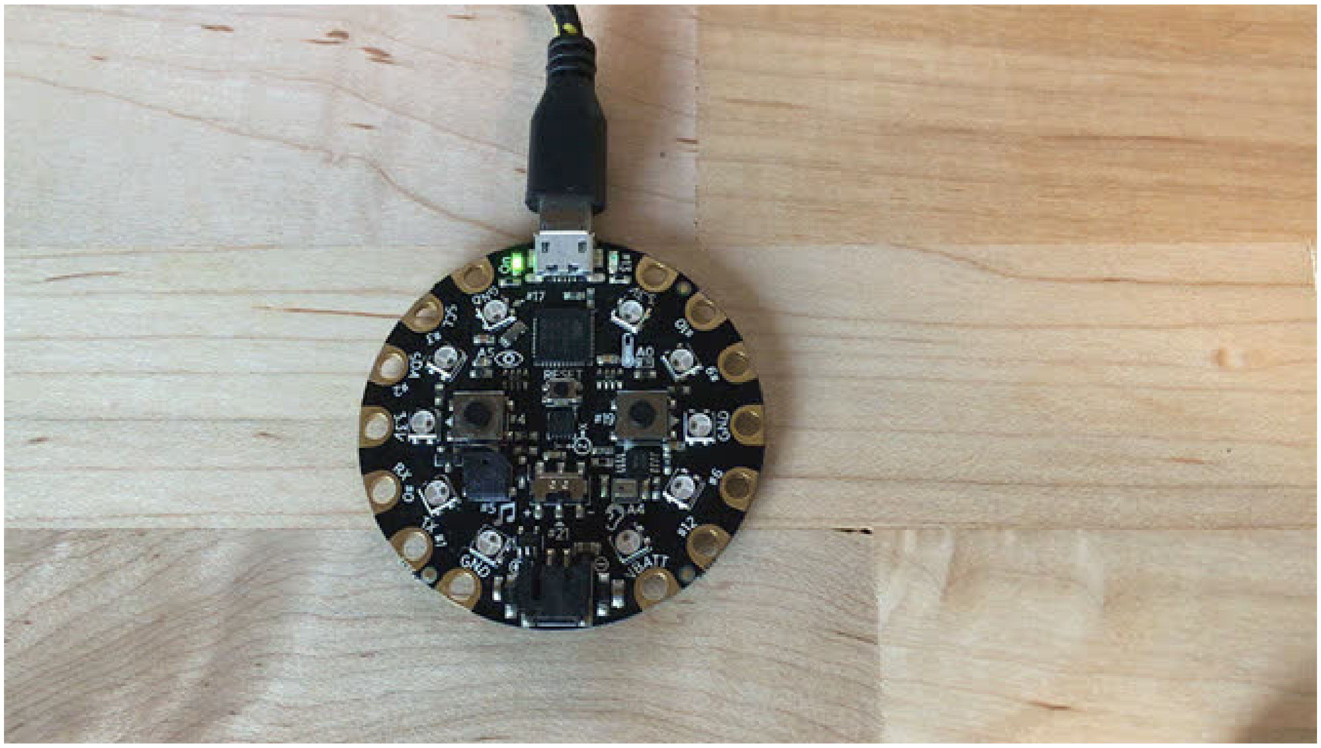
Now, let's combine two sets of pixel motion together.



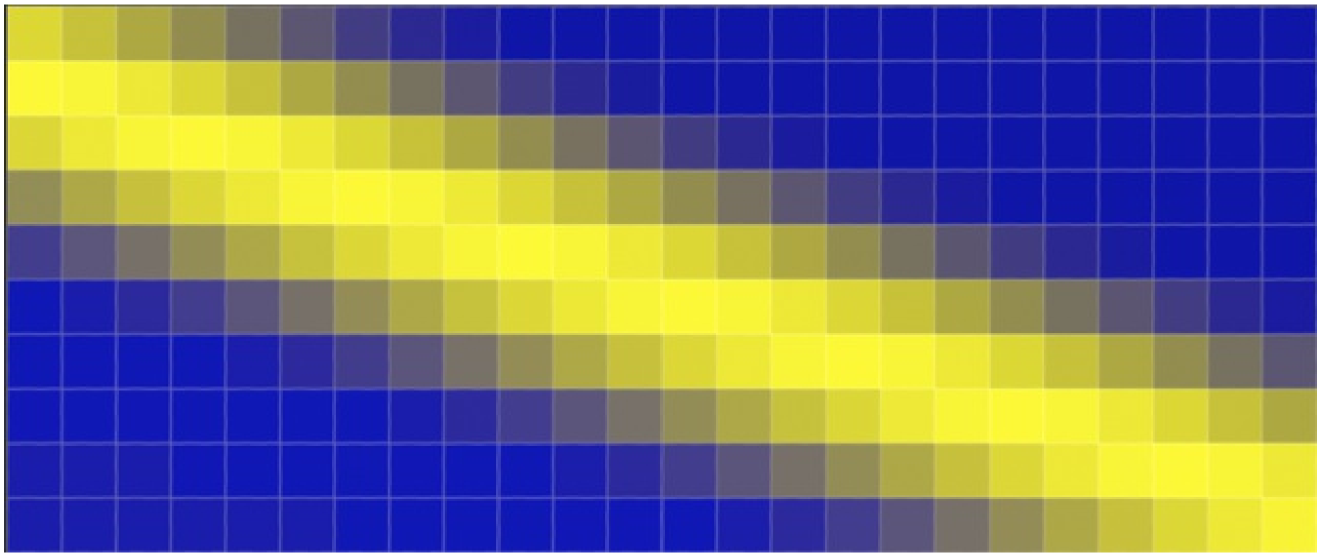


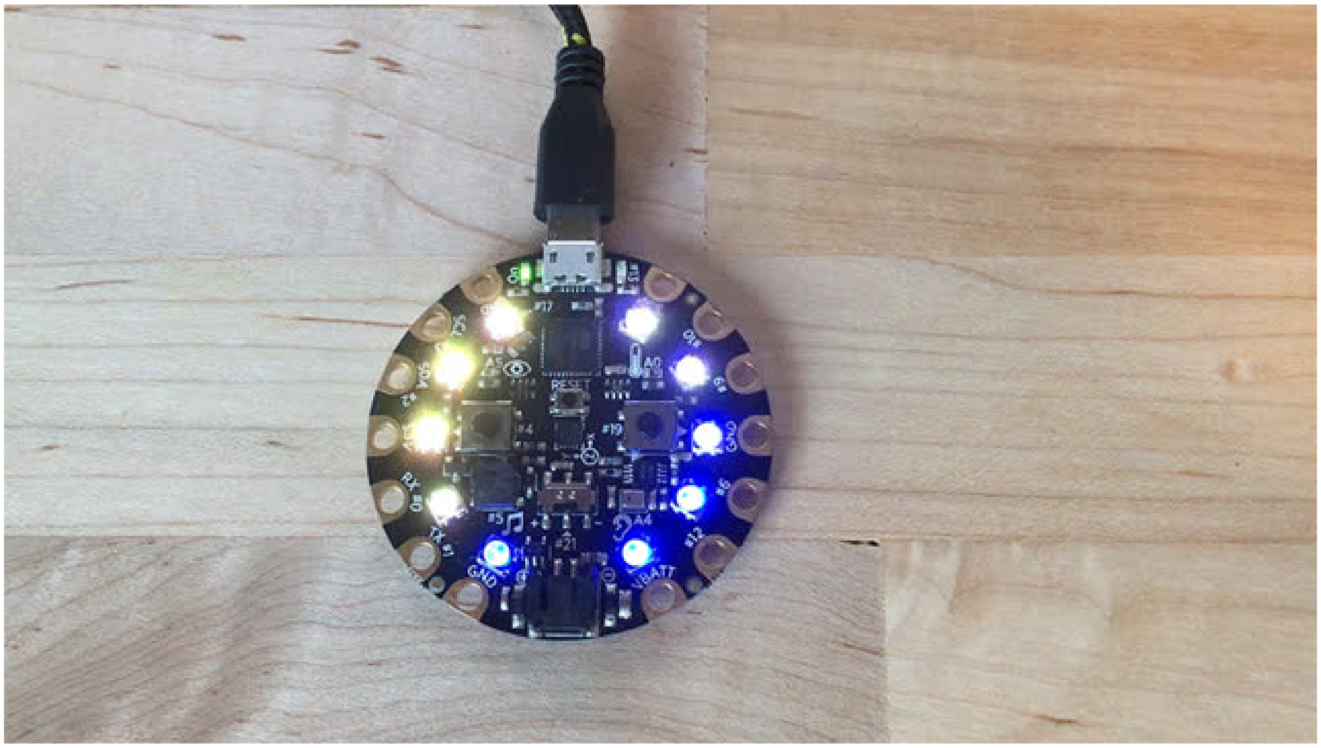
We can also use things like gradients in Photoshop or other image editing software to create gentle fades or color transformations. Here's a one second fade up to blue and back.



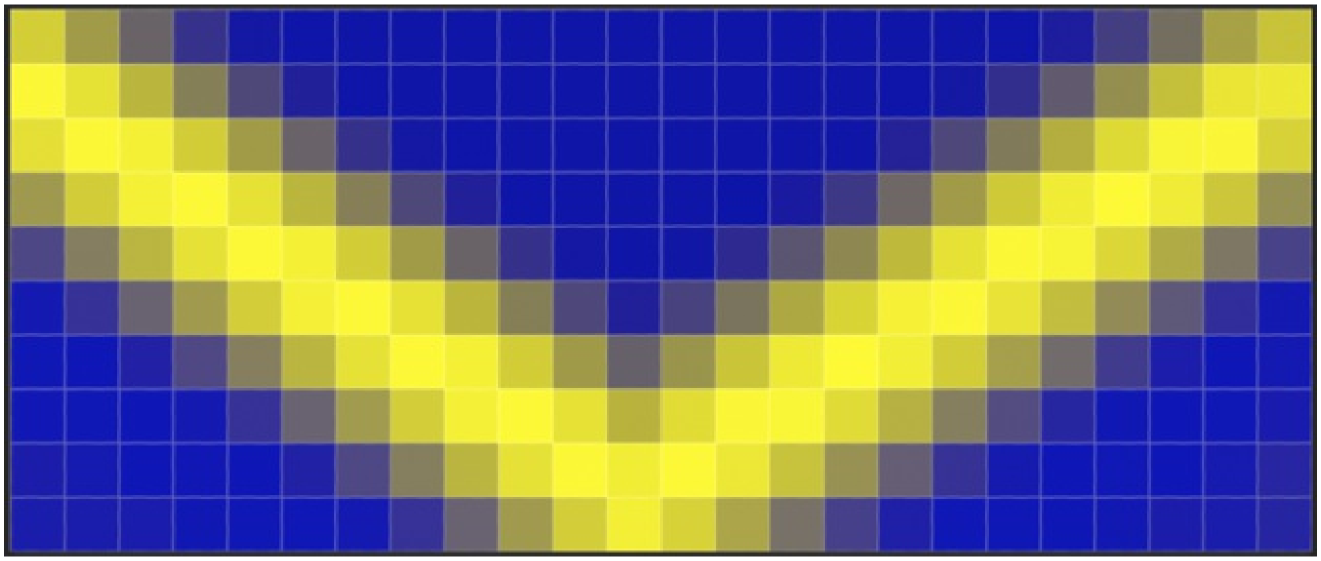


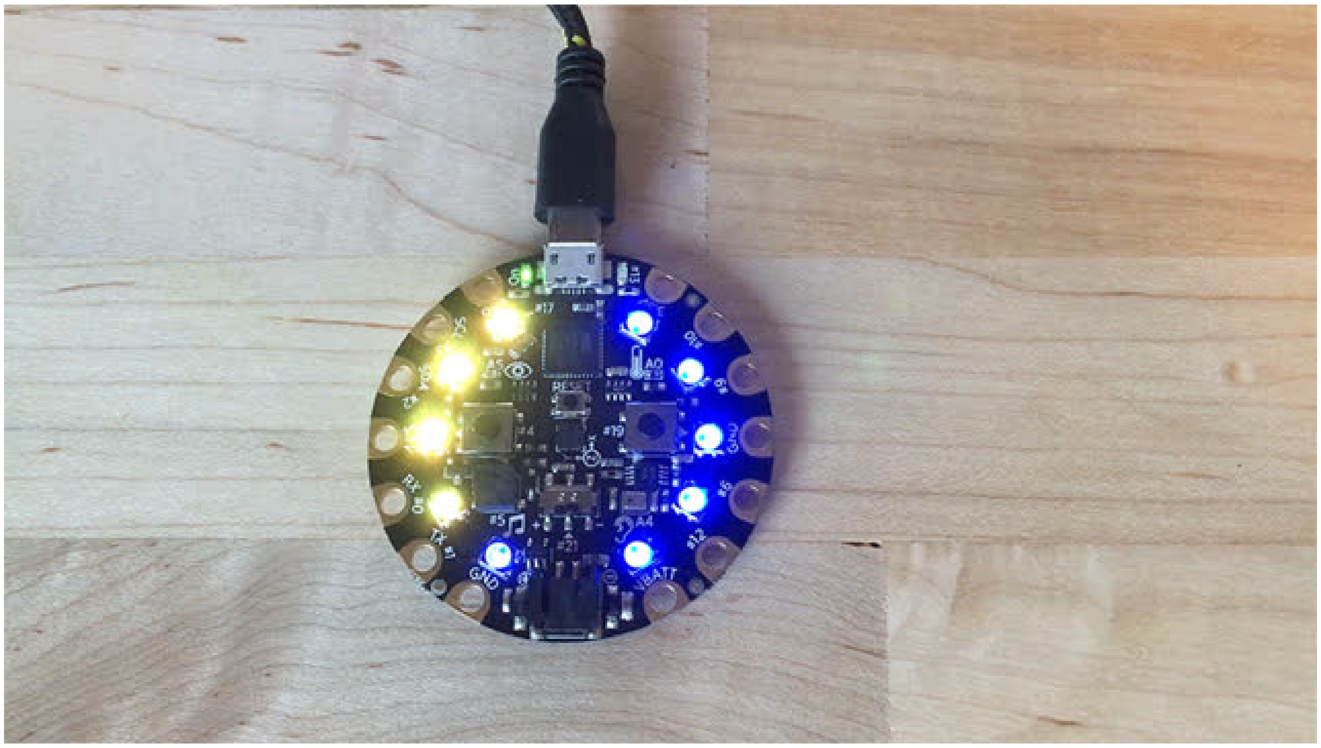
Take a look at a rotated gradient bar. Notice how it doesn't loop smoothly, or hook-up properly in animation terms.



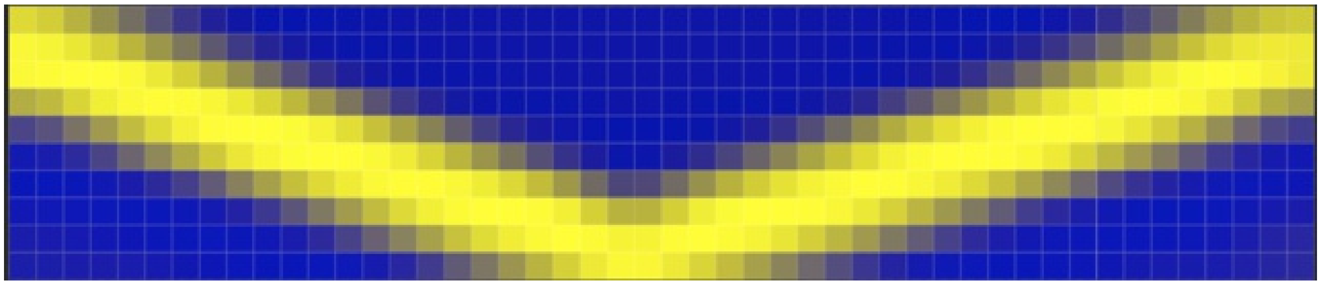


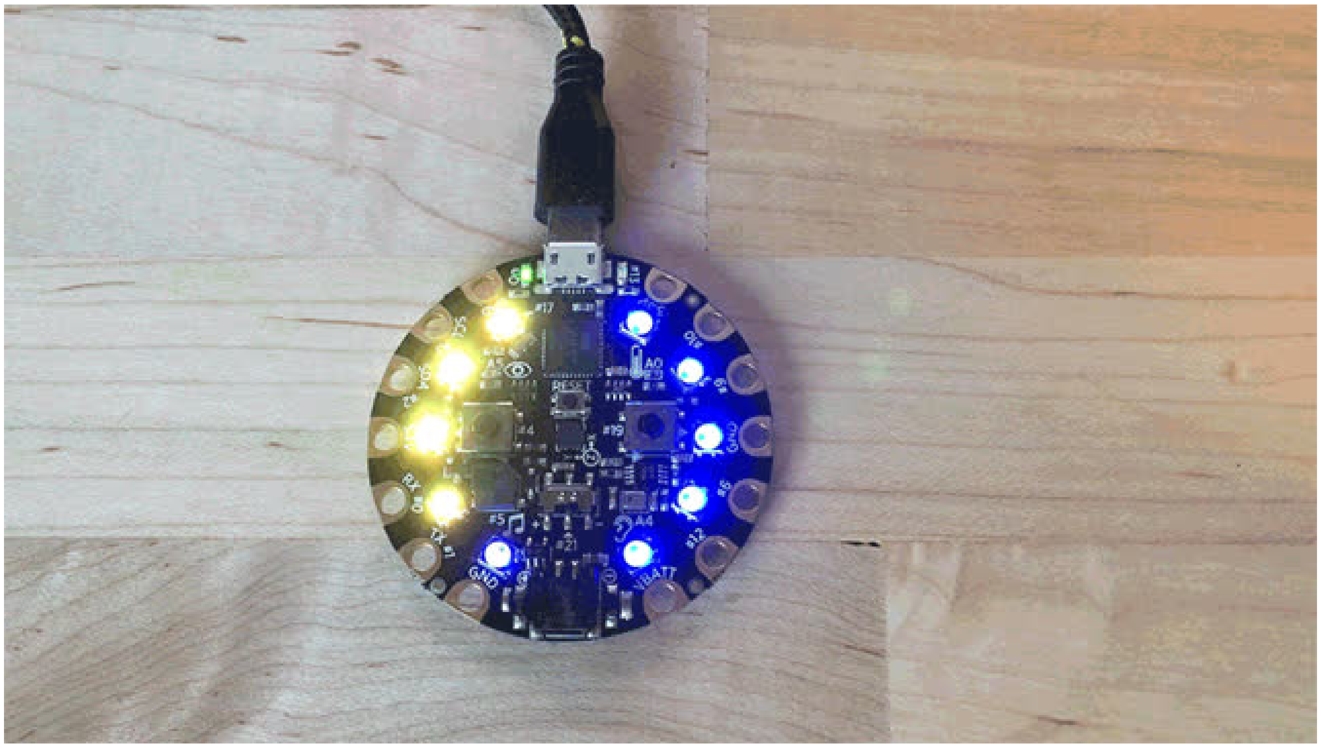
Here's one way to fix that, by blending the tail back to the head in the bitmap.



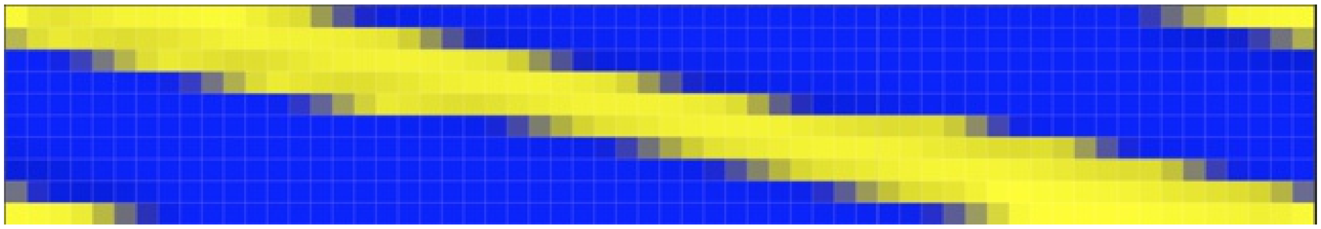


This fixes the hookup, but it's very fast. We can make a wider input bitmap to give us more frames of animation. This one is now 48 pixels wide, which translates to two seconds at 24FPS.

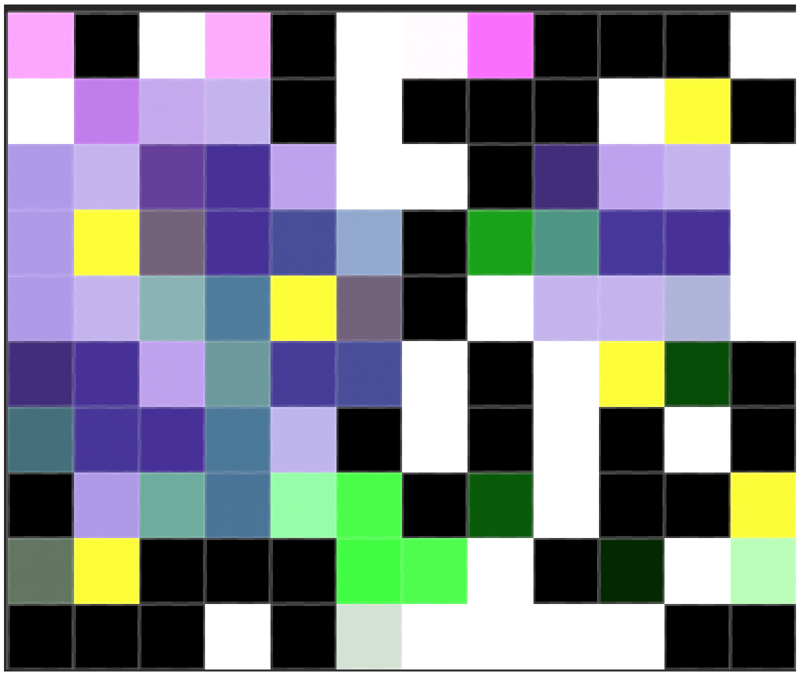


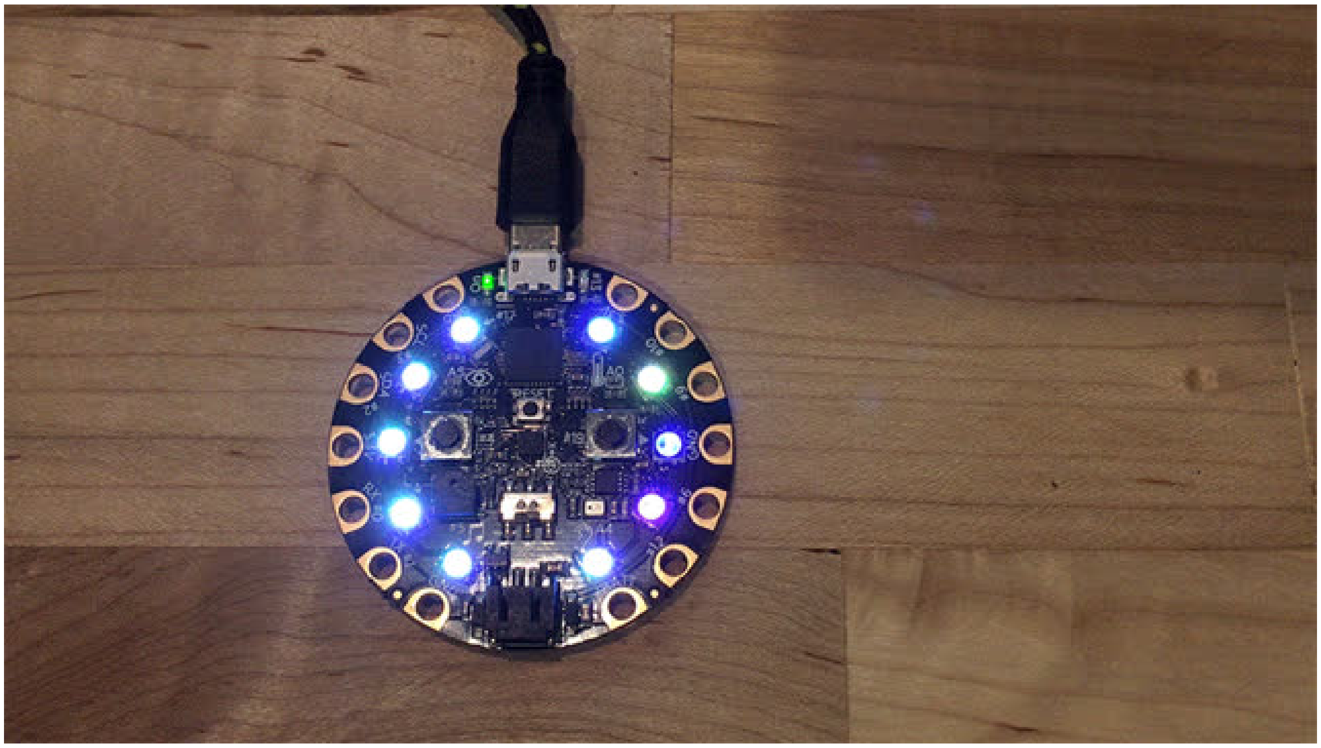


For a continuous sweep, we'll need to be more careful with the transitions and only "travel" the graphic in one direction.

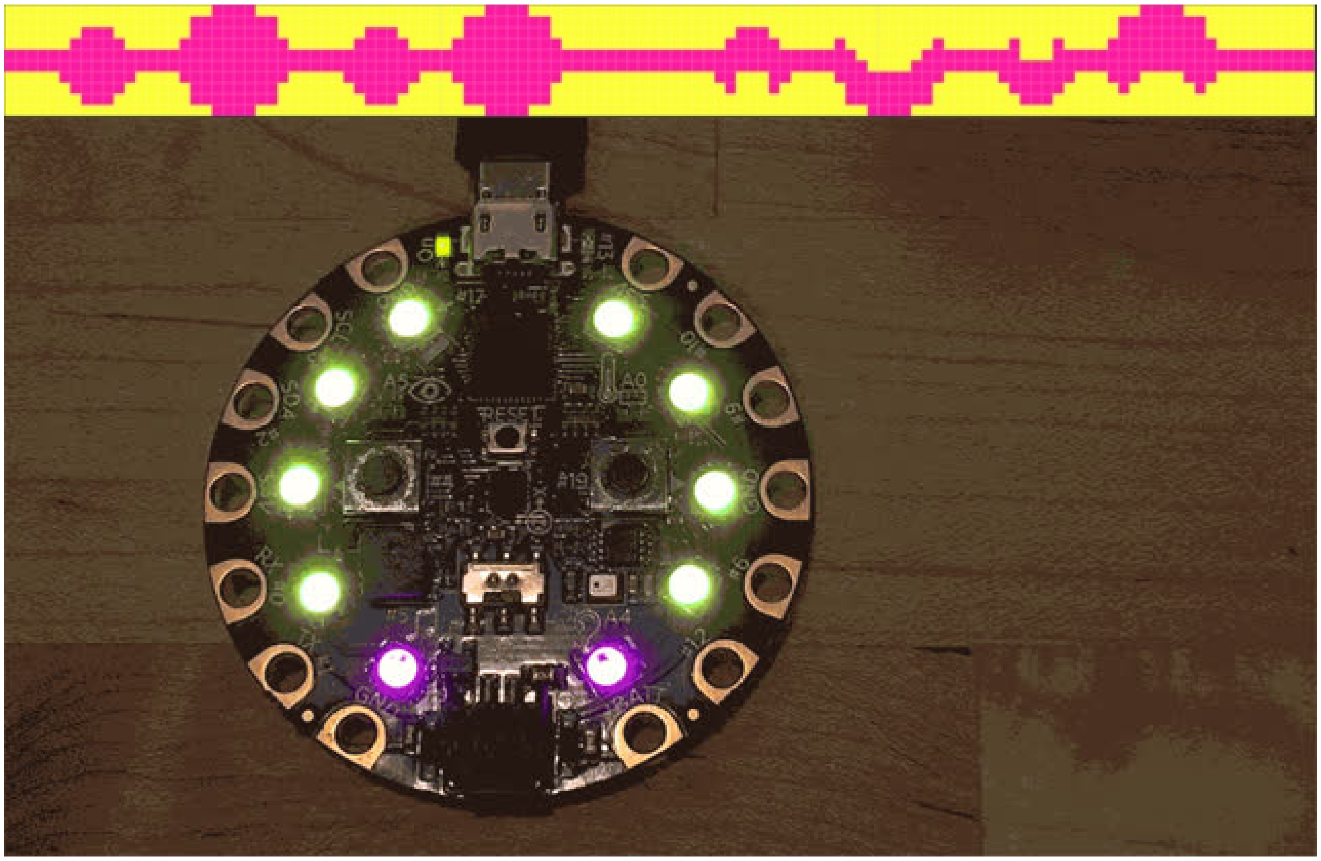


Noise can also be effective for creating sparkling, blinking animation. Here's a short 12 frame cycle.

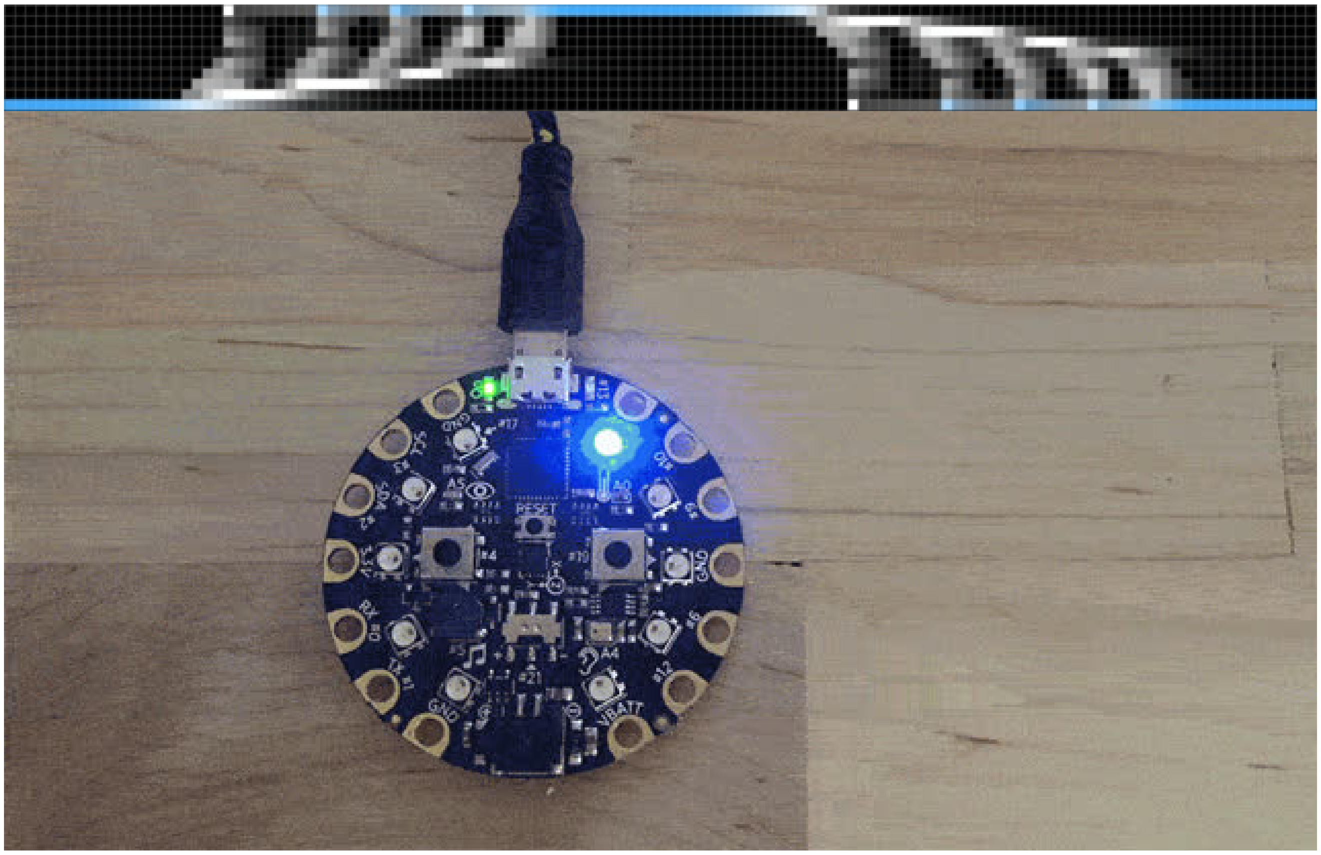




Wider input bitmaps give you more time to play with animation patterns.



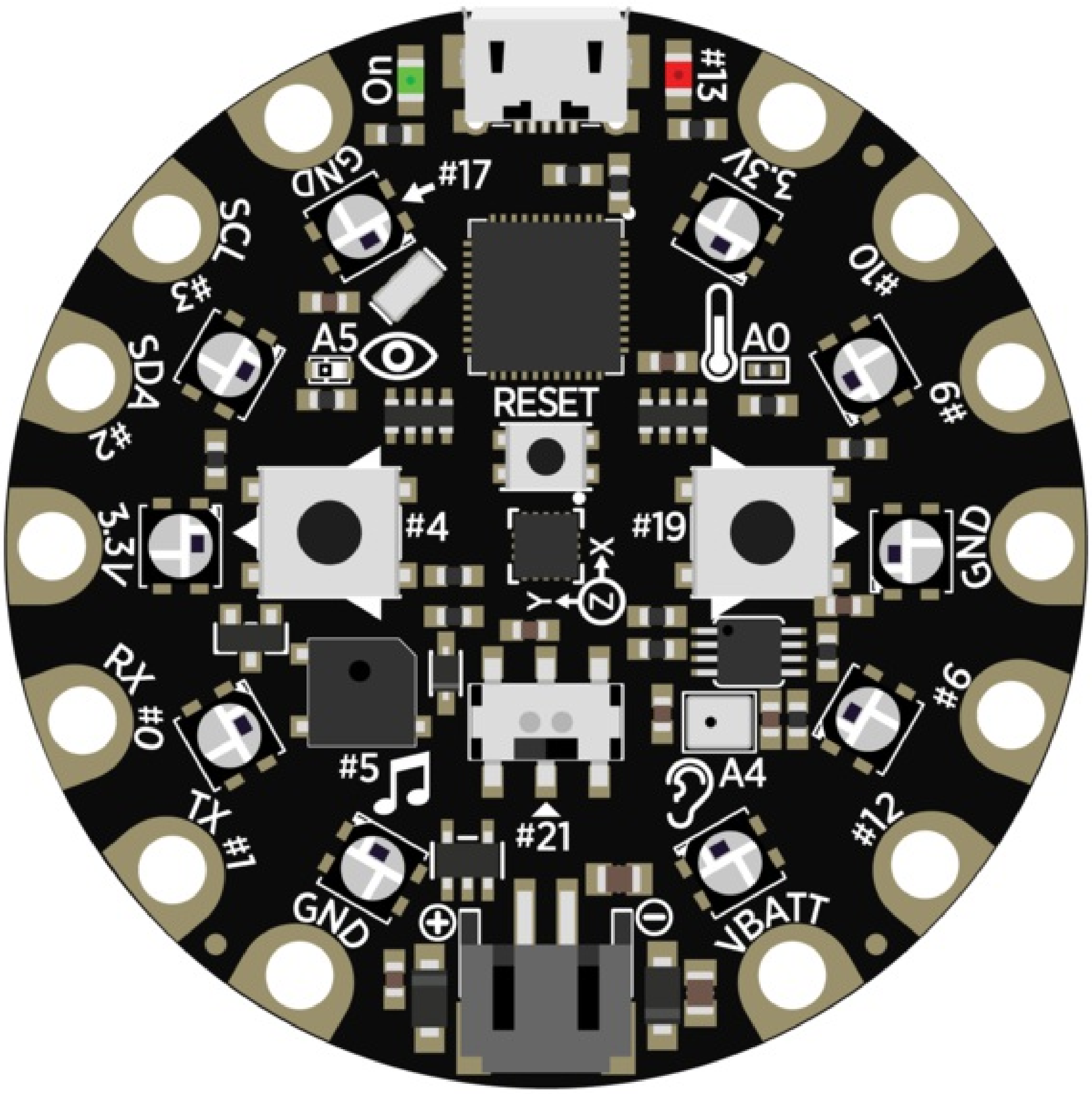
We can also use acceleration curves, such as this example. Each "drop" is shorter in number of frames than the last as the lights accumulate on the opposite side of the starting point.



Next we'll take a look at how to set up the Circuit Playground and convert bitmaps to animation header files.

Coding the Circuit Playground

Before you get started, make sure you've been through the basic tutorials on using the Circuit Playground. You'll need to have installed the Arduino IDE, added the Circuit Playground board to Arduino, and added the Circuit Playground library. This excellent guide, Introducing Circuit Playground (http://adafru.it/pAP) and Circuit Playground lesson #0 (http://adafru.it/r0D) will show you how to do all of that and get you going.



Plug your Circuit Playground into your computer now, and launch the Arduino IDE. Double check that you've selected the board and port as in the Lesson #0 tutorial and are ready to upload code.

Download the **NeoAnim.zip** Arduino sketch, then unzip it, and place the NeoAnim directory in your Arduino sketch directory.

NeoAnim.zip
http://adafru.it/sfA

The directory contains the **NeoAnim.ino** Arduino sketch, a 24x10 pixel **neoAnim.png** bitmap file, and the **neoAnim.h** header file that was created with the**convert.py** Python script we'll look at in a few steps.

```
//NeoAnim for Circuit Playground
// adapted by John Park from Circuit Playground Make Believe by Phil Burgess
// for Adafruit Industries
// MIT License

#include <Adafruit_CircuitPlayground.h>
#include "neoAnim.h" //this is the name of the animation derrived from the neoAnim.png bitmap file


void setup() {
  // Initialize Circuit Playground board, NeoPixels
  CircuitPlayground.begin();
  CircuitPlayground.strip.setBrightness(20);

  // Start looping animation
  playAnim( neoAnimPixelData, neoAnimFPS       , sizeof(neoAnimPixelData), true);
}

// Global values used by the animation and sound functions
uint16_t         *pixelBaseAddr, // Address of active animation table
                 pixelLen,      // Number of pixels in active table
                 pixelIdx,      // Index of first pixel of current frame
                 audioLen;      // Number of audio samples in active table
volatile uint16_t audioIdx;      // Index of current audio sample
uint8_t          pixelFPS,      // Frames/second for active animation
                *audioBaseAddr; // Address of active sound table
boolean          pixelLoop,     // If true, animation repeats
                 audioLoop;     // If true, audio repeats

// Begin playing a NeoPixel animation from a PROGMEM table
void playAnim(const uint16_t *addr, uint8_t fps, uint16_t bytes, boolean repeat) {
  pixelBaseAddr = addr;
  if(addr) {
    pixelFPS    = fps;
    pixelLen    = bytes / 2;
    pixelLoop   = repeat; //if set to 'repeat' it'll loop, set to 0 to play once only
    pixelIdx    = 0;
  } else {
    CircuitPlayground.strip.clear();
  }
}

uint32_t prev = 0; // Time of last NeoPixel refresh

void loop() {
  uint32_t t;      // Current time in milliseconds

  // Until the next animation frame interval has elapsed...
  while(((t = millis()) - prev) < (1000 / pixelFPS));
    // Show LEDs rendered on prior pass.  It's done this way so animation timing
    // is a bit more consistent (frame rendering time may vary slightly).
    CircuitPlayground.strip.show();


  prev = t; // Save refresh time for next frame sync

  if(pixelBaseAddr) {
    for(uint8_t i=0; i<10; i++) { // For each NeoPixel...
      // Read pixel color from PROGMEM table
      uint16_t rgb = pgm_read_word(&pixelBaseAddr[pixelIdx++]);
      // Expand 16-bit color to 24 bits using gamma tables
      // RRRRRGGGGGGBBBBB -> RRRRRRRR GGGGGGGG BBBBBBBB
      CircuitPlayground.strip.setPixelColor(i,
        pgm_read_byte(&gamma5[ rgb >> 11        ]),
        pgm_read_byte(&gamma6[(rgb >>  5) & 0x3F]),
        pgm_read_byte(&gamma5[ rgb        & 0x1F]));
    }


      if(pixelIdx >= pixelLen) { // End of animation table reached
        if(pixelLoop) { // Repeat animation
          pixelIdx = 0; // Reset index to start of table
        } else {        // else switch off LEDs
          playAnim(NULL, neoAnimFPS, 0, false);
        }
      }

  }
}
```

Next, you'll download the code that'll allow you to convert bitmaps to animation header files, and play these NeoPixel animations on your Circuit Playground.

Circuit Playground Make Believe
http://adafru.it/sex

Download the linked code from the button above, and then unzip the directory. Within the unzipped directory you'll find two directories: **CircuitPlaygroundMakeBelieve**, which you can move to your Arduino sketch directory (we won't use it for this guide, but it is useful if you want to build reactive props as shown in this guide (http://adafru.it/sez)), and the **Extras** directory, which you can place inside your **NeoAnim** directory you previously downloaded.

You can test the NeoAnim Arduino sketch by opening it in Arduino, and uploading it to your Circuit Playground. It should play a one second loop of a green pixel moving from upper left to upper right counterclockwise.

Now, you can adjust the animation. Open the **neoAnim.png** file in an image editor, such as **Preview** on OSX or **MS Paint** on Windows, or **GIMP** on Linux, and make some changes, such as the color of the non-black pixels. Save the file (it's OK to overwrite, you can redownload the original here).

In order to convert the new version of **neoAnim.png** to the **neoAnim.h** file, you'll need to follow these steps:

1. Install Python 3.5 https://www.python.org/downloads/ (http://adafru.it/fa7)
2. Add the Pillow or PIL imaging library to Python http://pillow.readthedocs.io/en/3.0.x/installation.html (http://adafru.it/seA)
3. Open a command shell and navigate to the NeoAnim directory inside your Arduino sketches directory
4. type the command to convert the bitmap: `python convert.py neoAnim.png > neoAnim.h`

This will generate the animation header file that looks something like this (yours will vary depending on the colors and placements of pixels):

```
#define neoAnimFPS 30

const uint16_t PROGMEM neoAnimPixelData[] = {
  0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X07E0, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X07E0,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X07E0, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X07E0, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X07E0, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X07E0,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X07E0, 0X0000, 0X0000, 0X0000, 0X0000,
  0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X07E0 };

const uint8_t PROGMEM gamma5[] = {
  0X00, 0X00, 0X00, 0X00, 0X01, 0X02, 0X03, 0X05, 0X07, 0X09, 0X0C, 0X10,
  0X14, 0X18, 0X1E, 0X24, 0X2B, 0X32, 0X3B, 0X44, 0X4E, 0X59, 0X65, 0X72,
  0X80, 0X8F, 0X9F, 0XB0, 0XC2, 0XD5, 0XE9, 0XFF };

const uint8_t PROGMEM gamma6[] = {
  0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X00, 0X01, 0X01, 0X01, 0X02, 0X02,
  0X03, 0X04, 0X04, 0X05, 0X06, 0X07, 0X09, 0X0A, 0X0C, 0X0D, 0X0F, 0X11,
  0X13, 0X15, 0X17, 0X1A, 0X1D, 0X1F, 0X22, 0X26, 0X29, 0X2C, 0X30, 0X34,
  0X38, 0X3D, 0X41, 0X46, 0X4B, 0X50, 0X55, 0X5B, 0X61, 0X67, 0X6D, 0X74,
  0X7A, 0X81, 0X89, 0X90, 0X98, 0XA0, 0XA8, 0XB1, 0XBA, 0XC3, 0XCC, 0XD6,
  0XE0, 0XEA, 0XF4, 0XFF };
```

This file is referenced when Arduino re-compiles your NeoAnim.ino sketch, so to test it out, simply re-upload the code to your Circuit Playground. You've just animated your own NeoPixel animation using an exposure sheet style bitmap!

In the Arduino IDE you can click on the neoAnim.h tab and change the framerate to 24 FPS if you like, or play around with different framerates to slow down or speed up the the loop.

From here, the rest is in your hands! Try different patterns, gradations, color noise, and so on to see what you can create.