

How to make Animation on NeoMatrix

What is NeoMatrix?

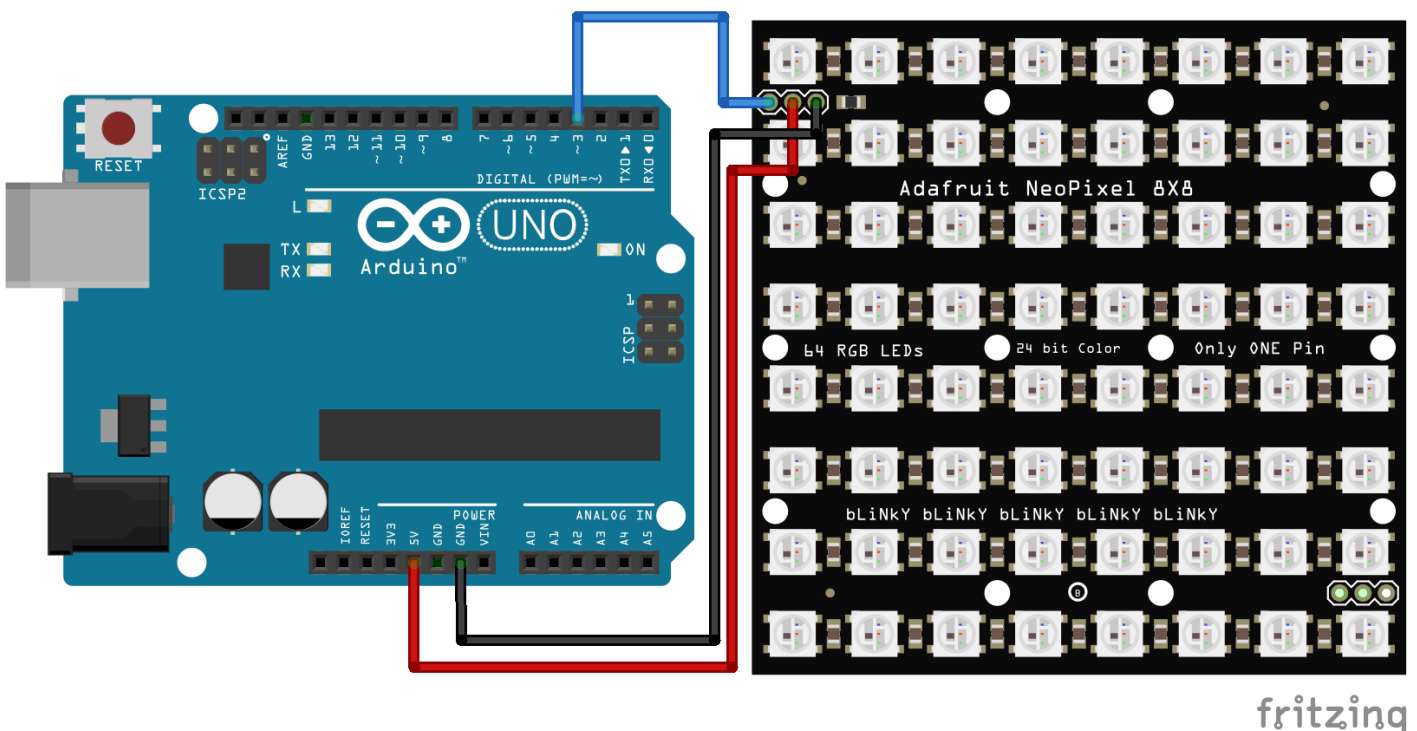
NeoMatrix is a grid lined up with mutiple Neopixel. Neopixel is addressable LEDs, meaning that they can be programmed individually. With the library created by Adafruit, you can easily program the Neopixel strip for your project. They come in different sizes and shapes and you can shorten or lengthen them flexibly. Once set up, they are very durable and efficient. They are commonly found in a lot house decorations or light installations. We have a tutorial about [Neopixel](#) here.

In this tutorial, we are using a 8x8 NeoMatrix from Adafruit.

th-1530653594.png

Wiring

1. DIN to Pin3
2. +5V to 5V
3. GND to GND



Library

Warning

Download version 1.9.0 or below of Adafruit Neopixel library for a more stable performance.

We will need three libraries for this tutorial.

1. **Adafruit NeoPixel**
2. **Adafruit GFX**
3. **Adafruit NeoMatrix**

We have a tutorial on **how to install a library** here.

Getting started

The example shows the upward arrow first and then the animation of a colour-changing pattern. The upward arrow is for identifying the orientation of your matrix.

```
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>

// Which pin on the Arduino is connected to the NeoPixels?
#define PIN 3

// Max is 255, 32 is a conservative value to not overload
// a USB power supply (500mA) for 12x12 pixels.
#define BRIGHTNESS 50

// Define matrix width and height.
#define mw 8
#define mh 8

#define LED_BLACK 0

int counter = 0;
int numImage = 0;

// When we setup the NeoPixel library, we tell it how many pixels, and which pin to use to send signals.
// Note that for older NeoPixel strips you might need to change the third parameter--see the strandtest
// example for more information on possible values.
Adafruit_NeoMatrix *matrix = new Adafruit_NeoMatrix(mw, mh, PIN,
```

```
NEO_MATRIX_TOP    + NEO_MATRIX_LEFT +  
NEO_MATRIX_ROWS + NEO_MATRIX_ZIGZAG,  
NEO_GRB          + NEO_KHZ800);
```

```
static const uint16_t PROGMEM
```

```
// These bitmaps were written for a backend that only supported  
// 4 bits per color with Blue/Green/Red ordering while neomatrix  
// uses native 565 color mapping as RGB.  
// I'm leaving the arrays as is because it's easier to read  
// which color is what when separated on a 4bit boundary  
// The demo code will modify the arrays at runtime to be compatible  
// with the neomatrix color ordering and bit depth.
```

```
RGB_bmp[][64] = {
```

```
//up for identifying direction
```

```
{  
0x000, 0x000, 0x000, 0x5C9, 0x5C9, 0x000, 0x000, 0x000,  
0x000, 0x000, 0x5C9, 0x5C9, 0x5C9, 0x5C9, 0x000, 0x000,  
0x000, 0x5C9, 0x000, 0x5C9, 0x5C9, 0x000, 0x5C9, 0x000,  
0x5C9, 0x000, 0x000, 0x5C9, 0x5C9, 0x000, 0x000, 0x5C9,  
0x000, 0x000, 0x000, 0x5C9, 0x5C9, 0x000, 0x000, 0x000,  
0x000, 0x000, 0x000, 0x5C9, 0x5C9, 0x000, 0x000, 0x000,  
0x000, 0x000, 0x000, 0x5C9, 0x5C9, 0x000, 0x000, 0x000,  
0x000, 0x000, 0x000, 0x5C9, 0x5C9, 0x000, 0x000, 0x000,  
  
},
```

```
//pattern1
```

```
{  
0x72E, 0x000, 0x823, 0x823, 0x823, 0x823, 0x000, 0x72E,  
0x000, 0x823, 0xEC5, 0xCDB, 0xCDB, 0xEC5, 0x823, 0x000,  
0x823, 0xEC5, 0xCDB, 0x0FF, 0x0FF, 0xCDB, 0xEC5, 0x823,  
0x823, 0xCDB, 0x0FF, 0x72E, 0x72E, 0x0FF, 0xCDB, 0x823,  
0x823, 0xCDB, 0x0FF, 0x72E, 0x72E, 0x0FF, 0xCDB, 0x823,  
0x823, 0xEC5, 0xCDB, 0x0FF, 0x0FF, 0xCDB, 0xEC5, 0x823,  
0x000, 0x823, 0xEC5, 0xCDB, 0xCDB, 0xEC5, 0x823, 0x000,  
0x72E, 0x000, 0x823, 0x823, 0x823, 0x823, 0x000, 0x72E,
```

```
},
```

```
//pattern2
```

```
{
```

```
0x823, 0x000, 0xEC5, 0xEC5, 0xEC5, 0xEC5, 0x000, 0x823,  
0x000, 0xEC5, 0xCDB, 0x0FF, 0x0FF, 0xCDB, 0xEC5, 0x000,  
0xEC5, 0xCDB, 0x0FF, 0x72E, 0x72E, 0x0FF, 0xCDB, 0xEC5,  
0xEC5, 0x0FF, 0x72E, 0x823, 0x823, 0x72E, 0x0FF, 0xEC5,  
0xEC5, 0x0FF, 0x72E, 0x823, 0x823, 0x72E, 0x0FF, 0xEC5,  
0xEC5, 0xCDB, 0x0FF, 0x72E, 0x72E, 0x0FF, 0xCDB, 0xEC5,  
0x000, 0xEC5, 0xCDB, 0x0FF, 0x0FF, 0xCDB, 0xEC5, 0x000,  
0x823, 0x000, 0xEC5, 0xEC5, 0xEC5, 0xEC5, 0x000, 0x823,
```

```
},
```

```
//pattern3
```

```
{
```

```
0xEC5, 0x000, 0xCDB, 0xCDB, 0xCDB, 0xCDB, 0x000, 0xEC5,  
0x000, 0xCDB, 0x0FF, 0x72E, 0x72E, 0x0FF, 0xCDB, 0x000,  
0xCDB, 0x0FF, 0x72E, 0x823, 0x823, 0x72E, 0x0FF, 0xCDB,  
0xCDB, 0x72E, 0x823, 0xEC5, 0xEC5, 0x823, 0x72E, 0xCDB,  
0xCDB, 0x72E, 0x823, 0xEC5, 0xEC5, 0x823, 0x72E, 0xCDB,  
0xCDB, 0x0FF, 0x72E, 0x823, 0x823, 0x72E, 0x0FF, 0xCDB,  
0x000, 0xCDB, 0x0FF, 0x72E, 0x72E, 0x0FF, 0xCDB, 0x000,  
0xEC5, 0x000, 0xCDB, 0xCDB, 0xCDB, 0xCDB, 0x000, 0xEC5,
```

```
},
```

```
//pattern4
```

```
{
```

```
0xCDB, 0x000, 0x0FF, 0x0FF, 0x0FF, 0x0FF, 0x000, 0xCDB,  
0x000, 0x0FF, 0x72E, 0x823, 0x823, 0x72E, 0x0FF, 0x000,  
0x0FF, 0x72E, 0x823, 0xEC5, 0xEC5, 0x823, 0x72E, 0x0FF,  
0x0FF, 0x823, 0xEC5, 0xCDB, 0xCDB, 0xEC5, 0x823, 0x0FF,  
0x0FF, 0x823, 0xEC5, 0xCDB, 0xCDB, 0xEC5, 0x823, 0x0FF,
```

```
0x0FF, 0x72E, 0x823, 0xEC5, 0xEC5, 0x823, 0x72E, 0x0FF,  
0x000, 0x0FF, 0x72E, 0x823, 0x823, 0x72E, 0x0FF, 0x000,  
0xCDB, 0x000, 0x0FF, 0x0FF, 0x0FF, 0x0FF, 0x000, 0xCDB,
```

```
},
```

```
//pattern5
```

```
{  
0x0FF, 0x000, 0x72E, 0x72E, 0x72E, 0x72E, 0x000, 0x0FF,  
0x000, 0x72E, 0x823, 0xEC5, 0xEC5, 0x823, 0x72E, 0x000,  
0x72E, 0x823, 0xEC5, 0xCDB, 0xCDB, 0xEC5, 0x823, 0x72E,  
0x72E, 0xEC5, 0xCDB, 0x0FF, 0x0FF, 0xCDB, 0xEC5, 0x72E,  
0x72E, 0xEC5, 0xCDB, 0x0FF, 0x0FF, 0xCDB, 0xEC5, 0x72E,  
0x72E, 0x823, 0xEC5, 0xCDB, 0xCDB, 0xEC5, 0x823, 0x72E,  
0x000, 0x72E, 0x823, 0xEC5, 0xEC5, 0x823, 0x72E, 0x000,  
0x0FF, 0x000, 0x72E, 0x72E, 0x72E, 0x72E, 0x000, 0x0FF,
```

```
},
```

```
};
```

```
void display_rgbBitmap(uint8_t bmp_num) {  
    static uint16_t bmx, bmy;  
  
    fixdrawRGBBitmap(bmx, bmy, RGB_bmp[bmp_num], 8, 8);  
    bmx += 8;  
    if (bmx >= mw) bmx = 0;  
    if (!bmx) bmy += 8;  
    if (bmy >= mh) bmy = 0;  
    matrix->show();  
}
```

```
// Convert a BGR 4/4/4 bitmap to RGB 5/6/5 used by Adafruit_GFX
```

```
void fixdrawRGBBitmap(int16_t x, int16_t y, const uint16_t *bitmap, int16_t w, int16_t h) {  
    uint16_t RGB_bmp_fixed[w * h];  
    for (uint16_t pixel = 0; pixel < w * h; pixel++) {
```

```

uint8_t r, g, b;
uint16_t color = pgm_read_word(bitmap + pixel);

//Serial.print(color, HEX);
b = (color & 0xF00) >> 8;
g = (color & 0x0F0) >> 4;
r = color & 0x00F;
//Serial.print(" ");
//Serial.print(b);
//Serial.print("/");
//Serial.print(g);
//Serial.print("/");
//Serial.print(r);
//Serial.print(" -> ");
// expand from 4/4/4 bits per color to 5/6/5
b = map(b, 0, 15, 0, 31);
g = map(g, 0, 15, 0, 63);
r = map(r, 0, 15, 0, 31);
//Serial.print(r);
//Serial.print("/");
//Serial.print(g);
//Serial.print("/");
//Serial.print(b);
RGB_bmp_fixed[pixel] = (r << 11) + (g << 5) + b;
// Serial.print(" -> ");
//Serial.print(pixel);
//Serial.print(" -> ");
//Serial.println(RGB_bmp_fixed[pixel], HEX);
}
matrix->drawRGBBitmap(x, y, RGB_bmp_fixed, w, h);
}

void setup() {
  Serial.begin(115200);

  matrix->begin();
  matrix->setTextWrap(false);
  matrix->setBrightness(BRIGHTNESS);
  // Test full bright of all LEDs. If brightness is too high
  // for your current limit (i.e. USB), decrease it.

```

```

//matrix->fillScreen(LED_WHITE_HIGH);
//matrix->show();
//delay(1000);
matrix->clear();
numImage=(sizeof(RGB_bmp) / sizeof(RGB_bmp[0]));
Serial.print("Number of images: ");
Serial.println(numImage);
}

void loop() {

    // clear the screen after X bitmaps have been displayed and we
    // loop back to the top left corner
    // 8x8 => 1, 16x8 => 2, 17x9 => 6
    static uint8_t pixmap_count = ((mw + 7) / 8) * ((mh + 7) / 8);
    // Cycle through red, green, blue, display 2 checkered patterns
    // useful to debug some screen types and alignment.

    Serial.print("Screen pixmap capacity: ");
    Serial.println(pixmap_count);

    display_rgbBitmap(counter++);
    delay(500);

    if (counter >= numImage){
        counter = 0;
    }

    Serial.println ("-----");
    //delay(1000);
}

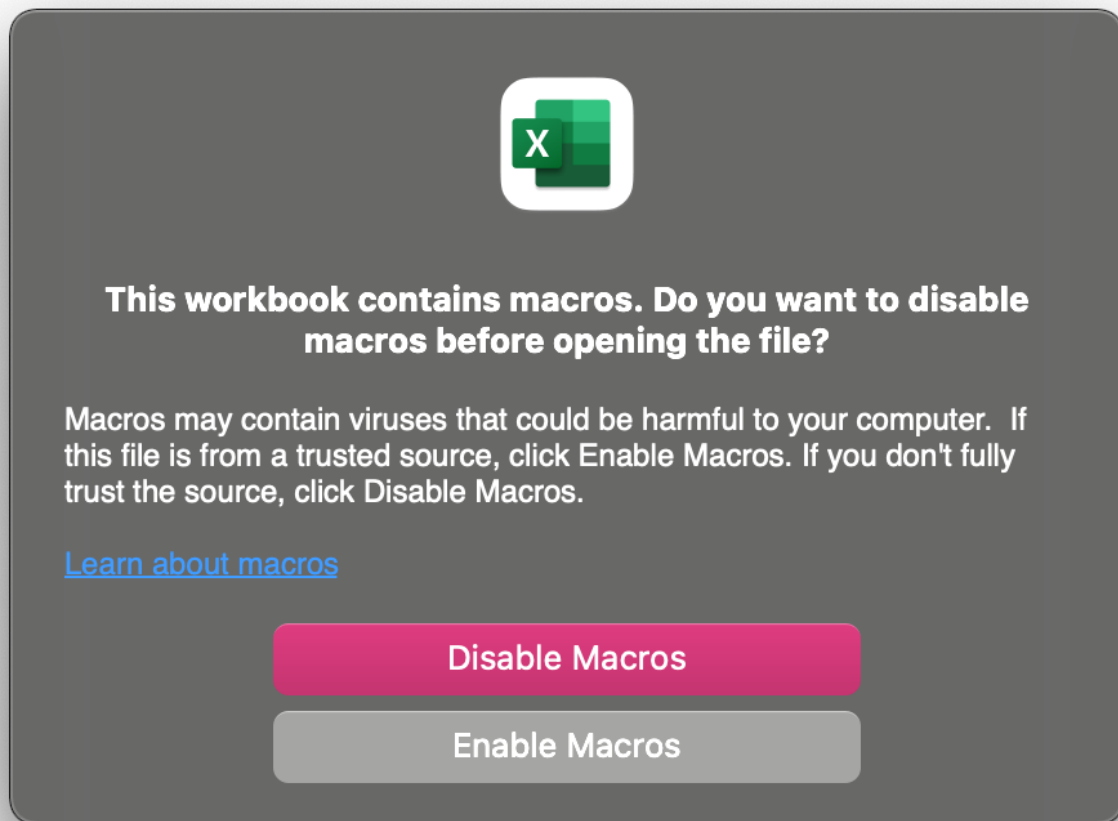
```

You should see your Neopixel displaying the animation frame by frame right now.

Creating your own Animation

To create the animation, we have to convert your animation into HEX code frame by frame. We have an Excel file you can download to do that.

1. Enable Macros for Excel



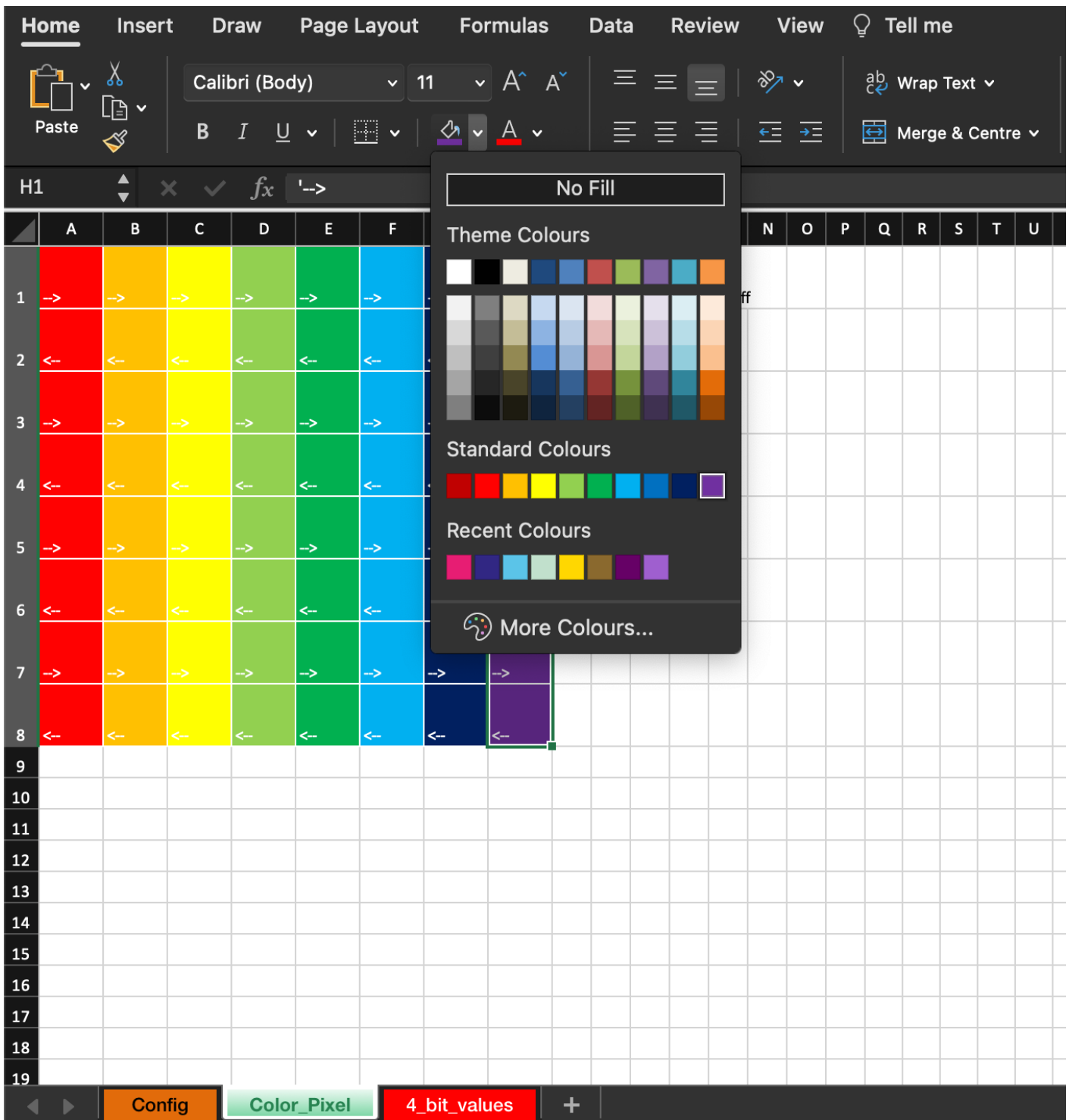
2. Set up your Matrix

Type in the number of row and column (blue boxes) and then press **Update Color_Pixel Page**.

	A	B	C	
1	Matrix	Number		
2	Row	8		
3	Column	8		
4	1. Update Color_Pixel Page			
5				
6	2. Update 4 Bit Values			
7				
8				
9				

3. Create your Frame

Go to the **Color_Pixel Tab**, and start filling your Matrix with colours. *Black means turning off that LED.*



4. Convert your Frame to Code

When you are happy with your design, go to the **Config Tab** and press **Update 4 Bit Values**.

	A	B	C
1	Matrix	Number	
2	Row	8	
3	Column	8	
4			
5	1. Update Color_Pixel Page		
6			
7	2. Update 4 Bit Values		
8			
9			

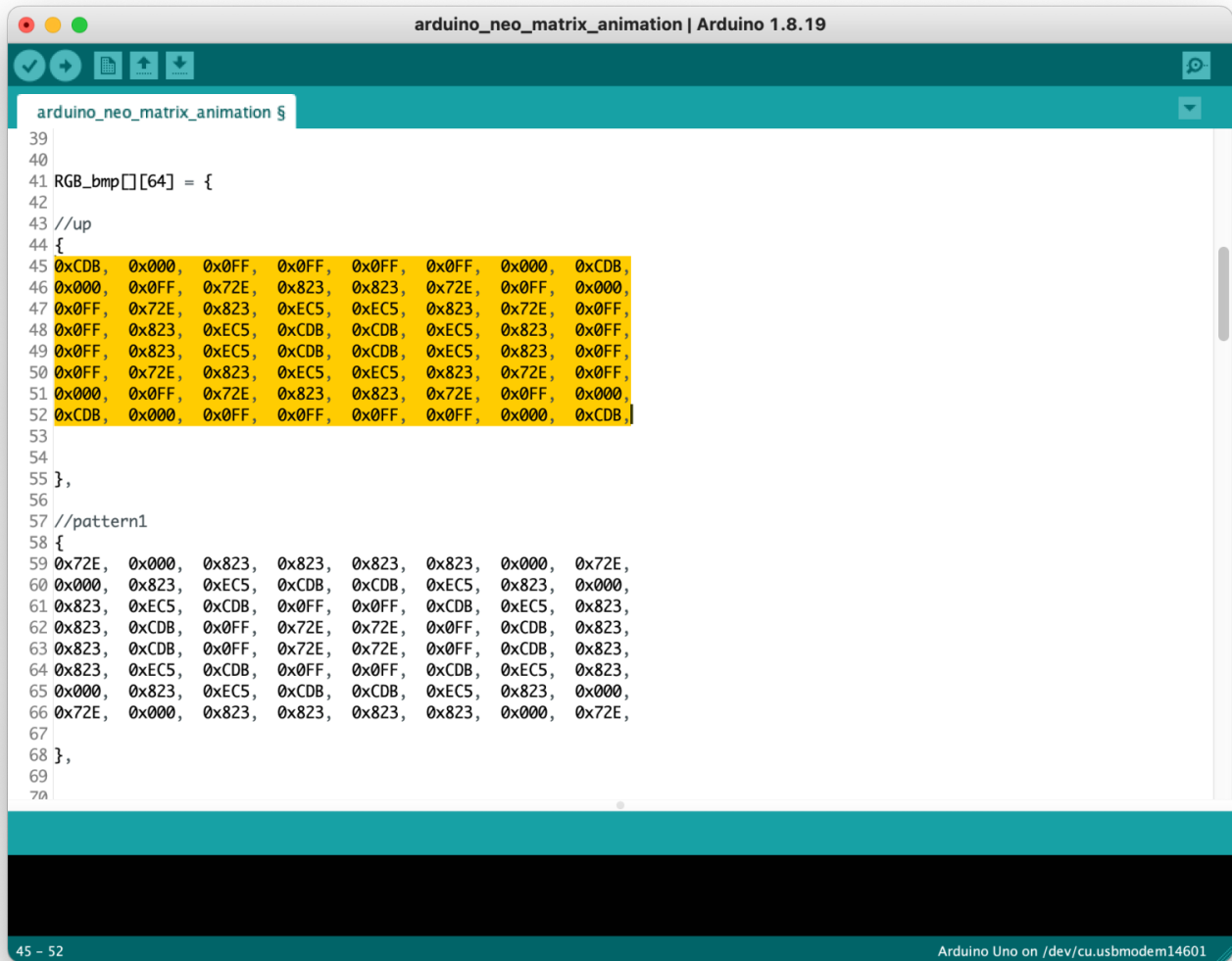
5. Copy your Code

Copy the Code in the **4_bit_values**.

A1								
	A	B	C	D	E	F	G	H
1	0x000,	0x000,	0x000,	0x5C9,	0x5C9,	0x000,	0x000,	0x000,
2	0x000,	0x000,	0x5C9,	0x5C9,	0x5C9,	0x5C9,	0x000,	0x000,
3	0x000,	0x5C9,	0x000,	0x5C9,	0x5C9,	0x000,	0x5C9,	0x000,
4	0x5C9,	0x000,	0x000,	0x5C9,	0x5C9,	0x000,	0x000,	0x5C9,
5	0x000,	0x000,	0x000,	0x5C9,	0x5C9,	0x000,	0x000,	0x000,
6	0x000,	0x000,	0x000,	0x5C9,	0x5C9,	0x000,	0x000,	0x000,
7	0x000,	0x000,	0x000,	0x5C9,	0x5C9,	0x000,	0x000,	0x000,
8	0x000,	0x000,	0x000,	0x5C9,	0x5C9,	0x000,	0x000,	0x000,
9								
10								
11								

6. Paste your Code in Arduino IDE

The yellow highlighted part within `{}` will be the code to be replaced. Upload, then you are good to go!



```
arduino_neo_matrix_animation | Arduino 1.8.19
arduino_neo_matrix_animation §
39
40
41 RGB_bmp[64] = {
42
43 //up
44 {
45 0xCDB, 0x000, 0x0FF, 0x0FF, 0x0FF, 0x0FF, 0x000, 0xCDB,
46 0x000, 0x0FF, 0x72E, 0x823, 0x823, 0x72E, 0x0FF, 0x000,
47 0x0FF, 0x72E, 0x823, 0xEC5, 0xEC5, 0x823, 0x72E, 0x0FF,
48 0x0FF, 0x823, 0xEC5, 0xCDB, 0xCDB, 0xEC5, 0x823, 0x0FF,
49 0x0FF, 0x823, 0xEC5, 0xCDB, 0xCDB, 0xEC5, 0x823, 0x0FF,
50 0x0FF, 0x72E, 0x823, 0xEC5, 0xEC5, 0x823, 0x72E, 0x0FF,
51 0x000, 0x0FF, 0x72E, 0x823, 0x823, 0x72E, 0x0FF, 0x000,
52 0xCDB, 0x000, 0x0FF, 0x0FF, 0x0FF, 0x0FF, 0x000, 0xCDB,
53
54
55 },
56
57 //pattern1
58 {
59 0x72E, 0x000, 0x823, 0x823, 0x823, 0x823, 0x000, 0x72E,
60 0x000, 0x823, 0xEC5, 0xCDB, 0xCDB, 0xEC5, 0x823, 0x000,
61 0x823, 0xEC5, 0xCDB, 0x0FF, 0x0FF, 0xCDB, 0xEC5, 0x823,
62 0x823, 0xCDB, 0x0FF, 0x72E, 0x72E, 0x0FF, 0xCDB, 0x823,
63 0x823, 0xCDB, 0x0FF, 0x72E, 0x72E, 0x0FF, 0xCDB, 0x823,
64 0x823, 0xEC5, 0xCDB, 0x0FF, 0x0FF, 0xCDB, 0xEC5, 0x823,
65 0x000, 0x823, 0xEC5, 0xCDB, 0xCDB, 0xEC5, 0x823, 0x000,
66 0x72E, 0x000, 0x823, 0x823, 0x823, 0x823, 0x000, 0x72E,
67
68 },
69
70
```

45 - 52

Arduino Uno on /dev/cu.usbmodem14601

Revision #2

Created 30 November 2022 13:10:34 by Joanne Leung

Updated 16 December 2024 10:30:07 by Joanne Leung