RGB Matrix Module for Arduino

Release 1.0

SunFounder

Aug 02, 2022

CONTENTS

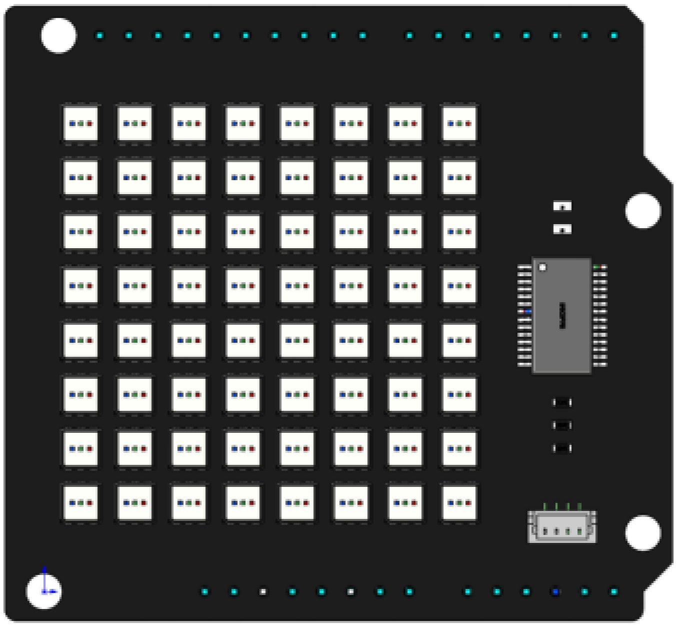
Welcome to use SunFounder RGB Matrix module. You can find the information you need for use here.

This is a module with 8 × 8 RGB LEDs on board. It also has a SH1.0-4P I2C control interface, which is convenient to connect to other I2C devices or other single-chip microcomputers.

Here is the Email: cs@sunfounder.com.

CHAPTER ONE

FEATURES



- Working voltage: DC 3.3V
- Lamp bead: FM-N3535RGBW-SH
- Driver: SLED 1734X LED driver
- Communication method: I2C
- Color depth: 24 bit (R/G/B each 8 bit color, 256 x 256 x 256=16777216 colors can be combined)
- Resolution: 8*8=64 DOTS
- Pixel pitch: 4.7mm
- matrix size: 36.5mm*36.5mm

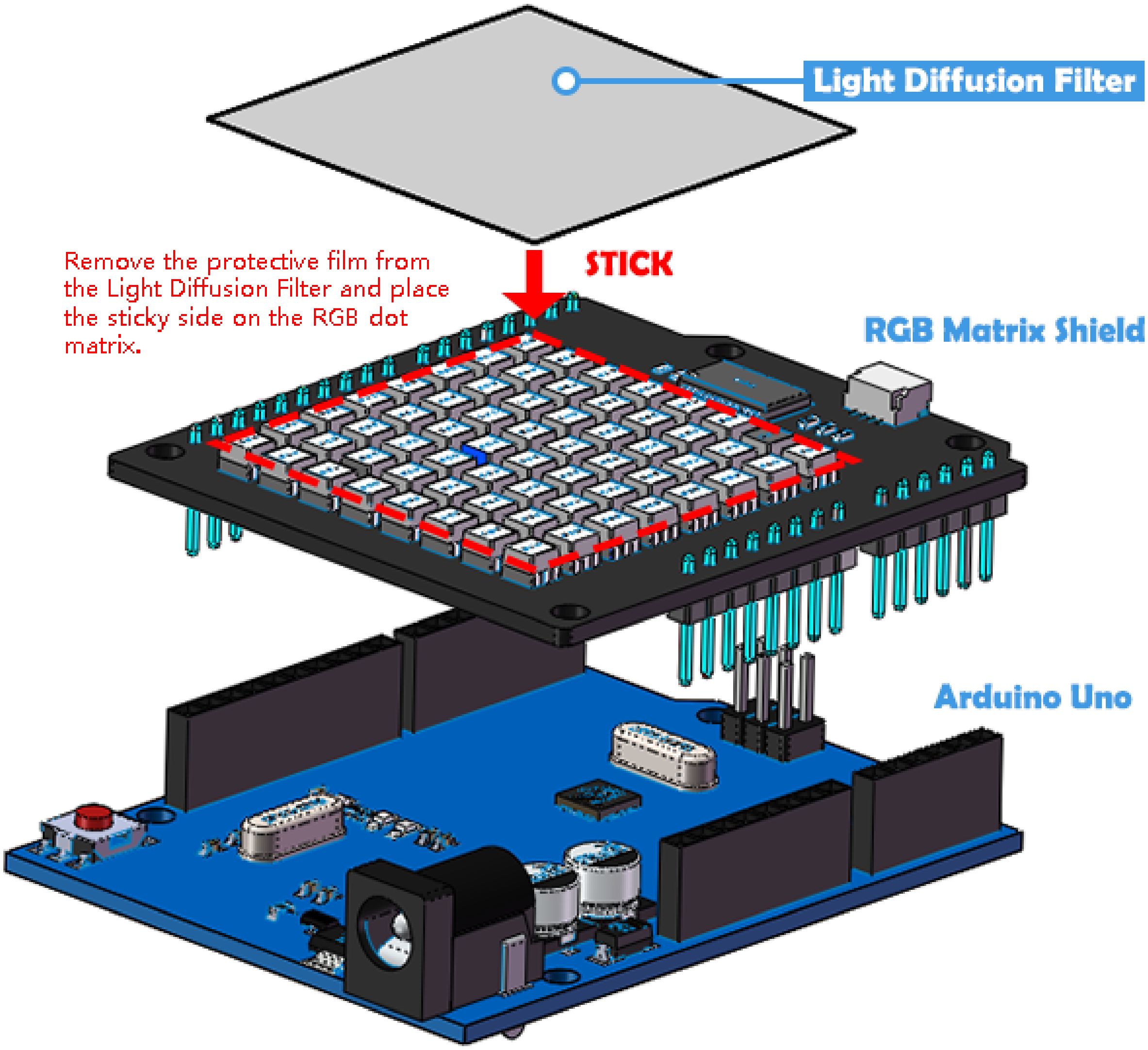
Documentation

- `PCB`
- `Schematic`
- `Datasheet`

CHAPTER

TWO

ASSEMBLE THE SHIELD


Light Diffusion Filter
STICK
Remove the protective film from the Light Diffusion Filter and place the sticky side on the RGB dot matrix.
RGB Matrix Shield
Arduino Uno

CHAPTER

THREE

PREPARATION

3.1 Tools needed

Please prepare the following tools:

- Arduino UNO
- USB Cable Type A/B
- Personal Computer

The APP you have to prepare:

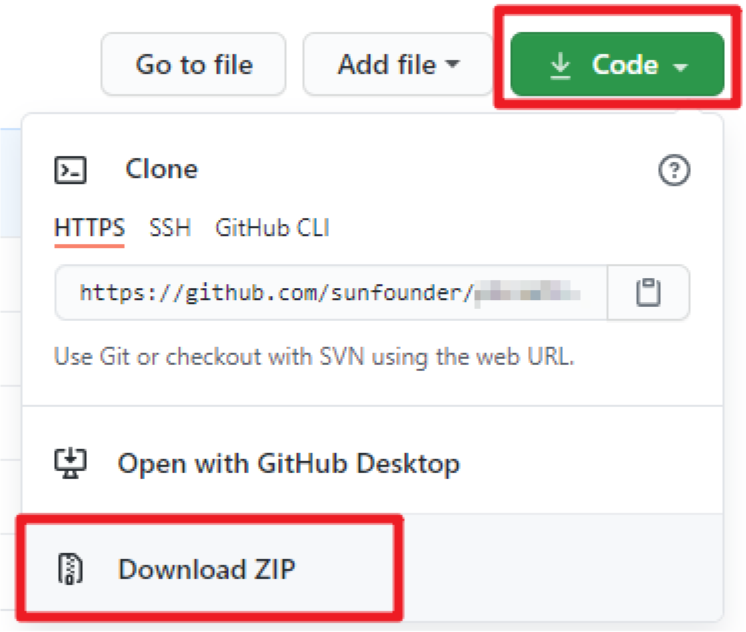
- Arduino IDE

Here are tutorials for installing Arduino on different systems.

- Windows OS.
- Mac OS.
- Linux.

3.2 Download the Code

Go to github-rgb_matrix download the code.



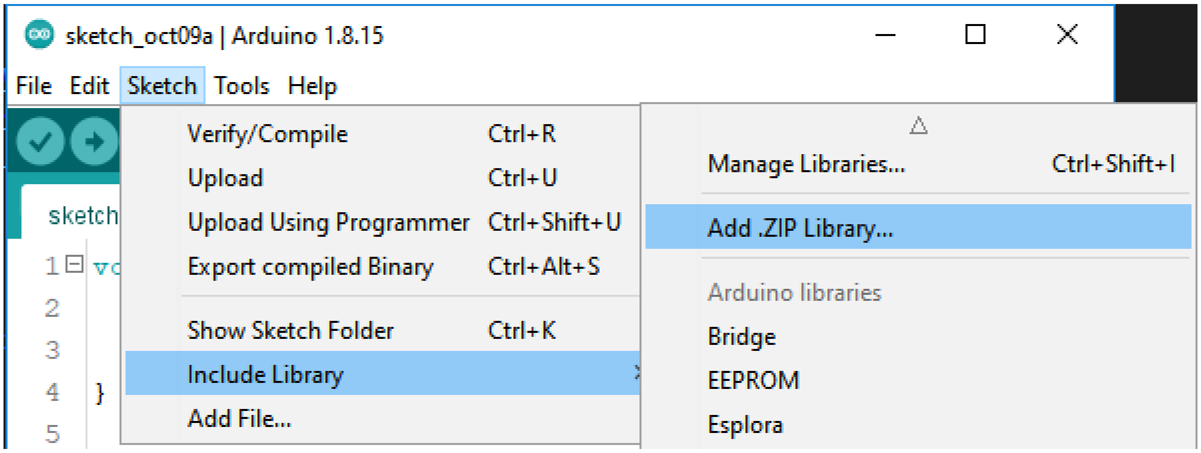
https://github.com/sunfounder/[redacted]

Use Git or checkout with SVN using the web URL.

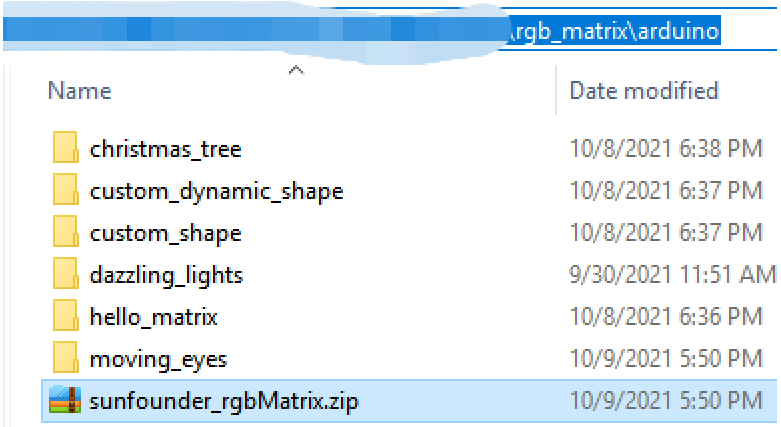
3.3 Add the Library

In order to use the RGB Matrix Shield, you need to load the library as follows.

In the Arduino IDE, navigate to **Sketch** > **Include Library** > **Add .ZIP Library**.


sketch_oct09a | Arduino 1.8.15
File
Edit
Sketch
Tools
Help
sketch
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Manage Libraries... Ctrl+Shift+I
Add .ZIP Library...
Arduino libraries
Bridge
EEPROM
Esplora

Find `sunfounder_rgbMatrix.zip` under the path `rgb_matrix/arduino`, then click **Open** to add it.



[redacted]\rgb_matrix\arduino

Name	Date modified
christmas_tree	10/8/2021 6:38 PM
custom_dynamic_shape	10/8/2021 6:37 PM
custom_shape	10/8/2021 6:37 PM
dazzling_lights	9/30/2021 11:51 AM
hello_matrix	10/8/2021 6:36 PM
moving_eyes	10/9/2021 5:50 PM
sunfounder_rgbMatrix.zip	10/9/2021 5:50 PM

CHAPTER FOUR

PROJECTS

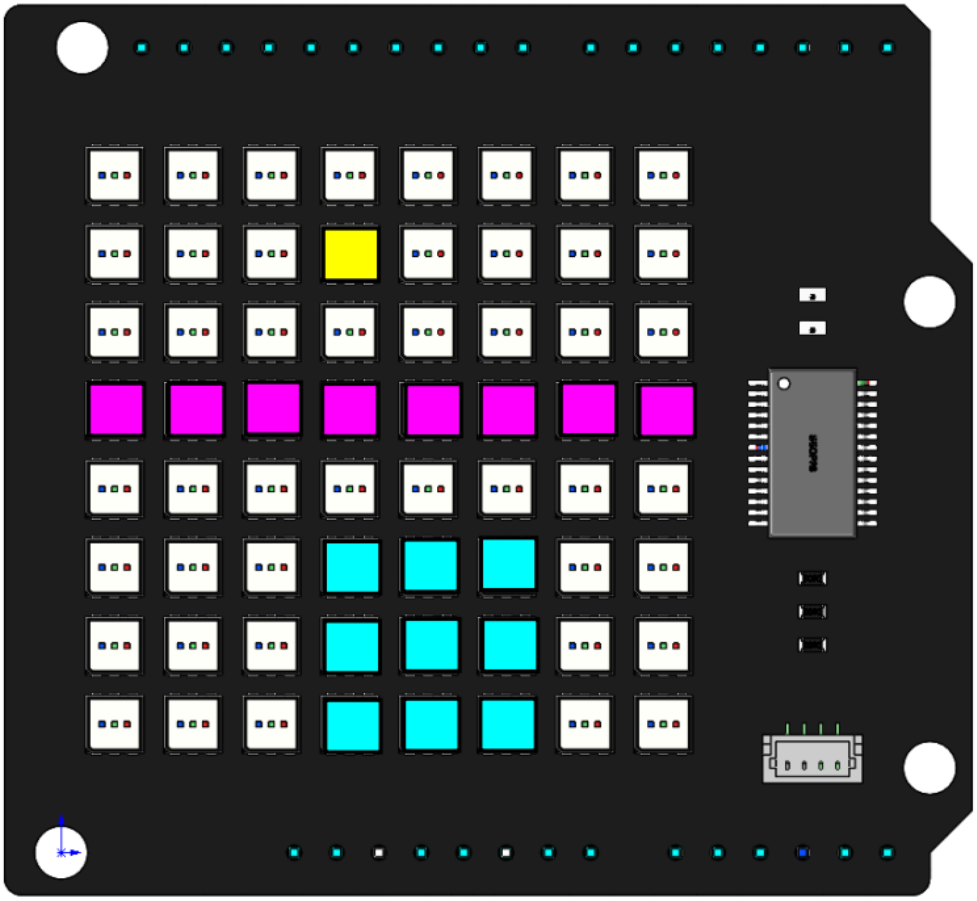
This page show you the examples provided with RGB Matrix.

Note: Before downloading the code, make sure you have *Add the Library*.

4.1 Hello Matrix

Introduce

In this project, you will learn how to make RGB Matix HAT display different patterns and characters in different colors.



Code

When the program runs, you will see a dot, a line, a rectangle, a love pattern, the letter A, and the text Hi, SunFouder appear on the RGB Matrix Shield in turn.

How it works?

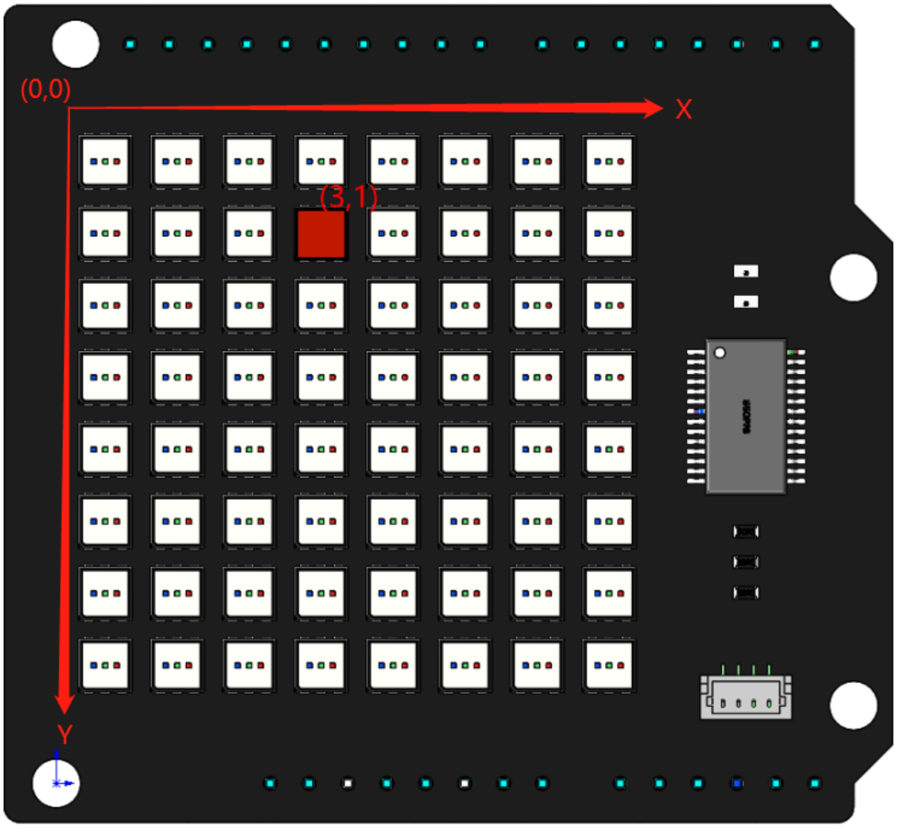
```
RGBMatrixInit();
```

This function is used to initialize the RGB Matrix Shield. To make the RGB Matrix Shield work, you need to call this function first.

```
byte dot[2]={3,1};
byte line[4]={0,3,7,3};
byte rectangle[4]={3,5,5,7};
byte heart[]={0x00,0x66,0xff,0xff,0x7e,0x3c,0x18,0x00};
```

- `dot[2]={3,1}`: Define an array to store the coordinates of the point (3,1).
- `line[4]={0,3,7,3}`: Define an array to store the start (0,3) and end (7,3) coordinates of the line.
- `rectangle[4]={3,5,5,7}`: Define an array to store the two diagonal coordinates (3, 5) and (5, 7) of the rectangle.
- `heart[]`: This is a hexadecimal array that stores the heart pattern, and each hex digit stores the LED lit or off state for each row. For example, the second element in `heart[]`, 0x66 (0110 0110), 0 means off and 1 means on, so you can see that the 1st, 2nd, 5th, and 6th are lit and the other LEDs are off.

The x,y coordinate directions of the dot matrix are as follows, with the first RGB LED in the upper left corner as the coordinate origin.



```
draw_point(dot, 255, 255, 0);
image();
delay(3000);
draw_line(line, 255, 0, 255);
image();
delay(3000);
draw_rectangle(rectangle, 0, 255, 255);
image();
delay(3000);
ShowHex(heart, 255, 0, 0);
delay(3000);
DispShowChar('A', 0, 255, 0);
delay(3000);
flow_text("Hi, SunFounder", 0, 0, 255);
delay(3000);
```

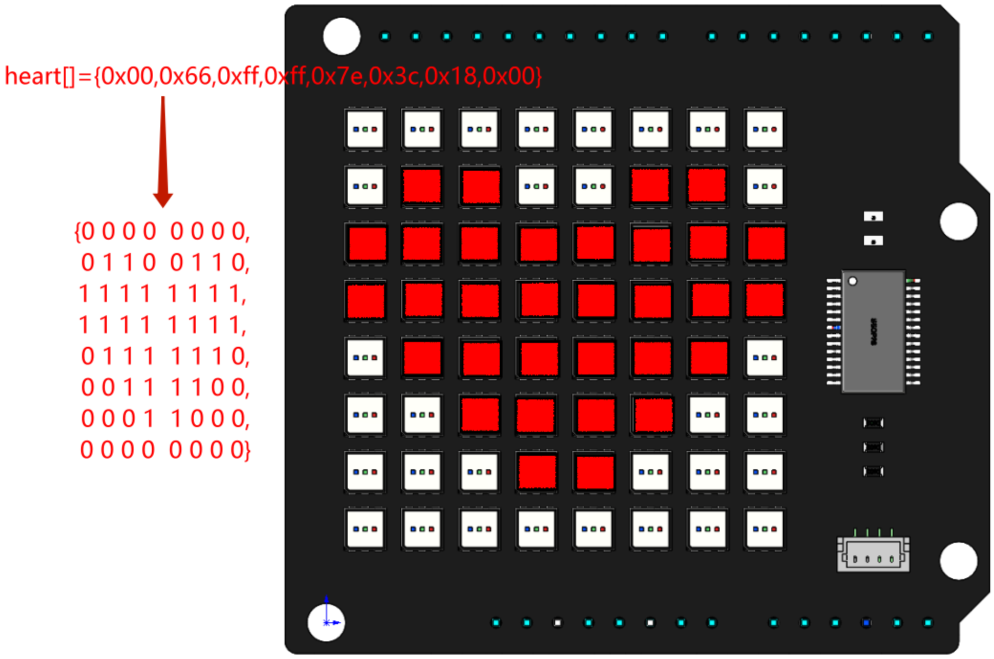
We have packaged six basic functions in the RGB Matrix Shield library.

- `draw_point(dot,255,255,0)` It is used to draw a yellow point on the RGB Matrix Shield. It has four parameters. `dot` is an array to store the coordinates of the points. 255, 255, 0 represent to fill this point with yellow. Reference: https://www.rapidtables.com/web/color/RGB_Color.html for more color value combinations.
- `draw_line(line,255,0,255)`: Draw a magenta line.
- `draw_rectangle(rectangle,0,255,255)`: Draw a cyan rectangle.

Note: `draw_point()`, `draw_line()`, `draw_rectangle()` just confirm the coordinates of the LEDs that need to be lit, and cooperate with the `image()` function to actually light them.

- `ShowHex(heart,255,0,0)`: Show a red `heart` on the RGB Matrix Shield.

`ShowHex` converts hexadecimal numbers into binary numbers, then judges the binary numbers, and when one of them is equal to 1, it will light up the corresponding LED, as shown below.



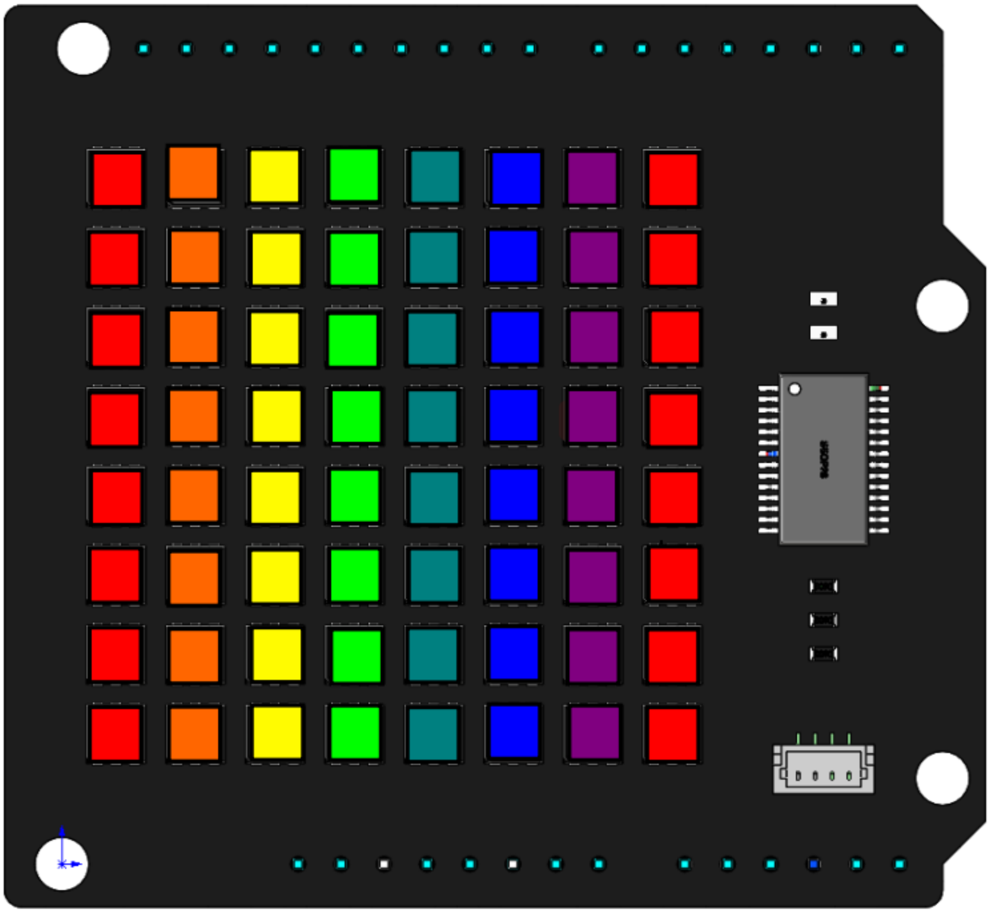
Reference: https://gurgleapps.com/tools/matrix#tp-color can get more such hex arrays.

- `DispShowChar('A',0,255,0)`: Let the RGB Matrix Shield display a green character A.
- `flow_text("Hi!Sunfounder",0,0,255)`: Let the RGB matrix shield display a string of blue text "Hi, SunFounder".

Note: Characters are represented by single quotation marks, and strings are represented by double quotation marks.

4.2 Dazzling Light

In the previous project, we learned to use some simple functions to make RGB Matrix Shield work. So here, we will use the `draw_line()` function with different colors to make RGB Matrix HAT make more cool effects.



Code

We have written two light blinking modes, `dazzling_light()` and `dazzling_light()2` for reference. When the program is running, you will first see the RGB matrix shield flowing on displaying different colors. After a while, you will notice that the flow of light becomes more smooth.

How it works?

```
for (int i=0; i<50; i++){
  dazzling_light();
}
for (int i=0; i<5; i++){
  dazzling_light2();
}
```

The main logic is to call the `dazzling_light()` function 50 times and then call the `dazzling_light2()` function 5 times.

```
byte line[8][4] = {{0, 0, 0, 7},
                   {1, 0, 1, 7},
```

(continues on next page)

(continued from previous page)

```
                   {2, 0, 2, 7},
                   {3, 0, 3, 7},
                   {4, 0, 4, 7},
                   {5, 0, 5, 7},
                   {6, 0, 6, 7},
                   {7, 0, 7, 7}};
```

Define a two-dimensional array `line[8][4]` to store the starting and ending coordinates of the 8 vertical lines.

```
byte color[7][3] = {{255,0,0},
                    {255,102,0},
                    {255,255,0},
                    {0,255,0},
                    {0,128,128},
                    {0,0,255},
                    {128,0,128}};
```

Define a two-dimensional array `color[7][3]` to store the 7 colors, red, orange, yellow, green, blue, blue and purple.

```
  int i = 0;
  void dazzling_light(){
    for (int j=0; j<8; j++){
          draw_line(line[j],color[i][0],color[i][1],color[i][2]);
          i++;
          if (i == 6){
            i = 0;
          }
    }
image();
  }
```

The `dazzling_light()` function is to write different colors (red, orange, yellow, green, blue, blue and purple) to the 8 vertical lines, where the first and last lines are red.

The `for` loop traverses the array `line[]` and draws eight vertical lines on the RGB matrix shield with `draw_line()`. The colors are chosen from the array `color[7][3]`, for example, `color[0]` represents the first element {255, 0, 0}, while `color[0][1]` represents 255.

```
void dazzling_light2(){
  for (long firstPixelHue = 0; firstPixelHue < 65536; firstPixelHue += 500) {
    for (int j=0; j<8; j++){
      long pixelHue = firstPixelHue + (j * 65536L / 16);
      draw_line(line[j], gamma32(ColorHSV(pixelHue)));
    }
    image();
  }
}
```

When you call the `dazzling_light2()` function, you will notice a softer flow of colors. This is because we have split the colors into more colors, making the transition between colors more smooth.

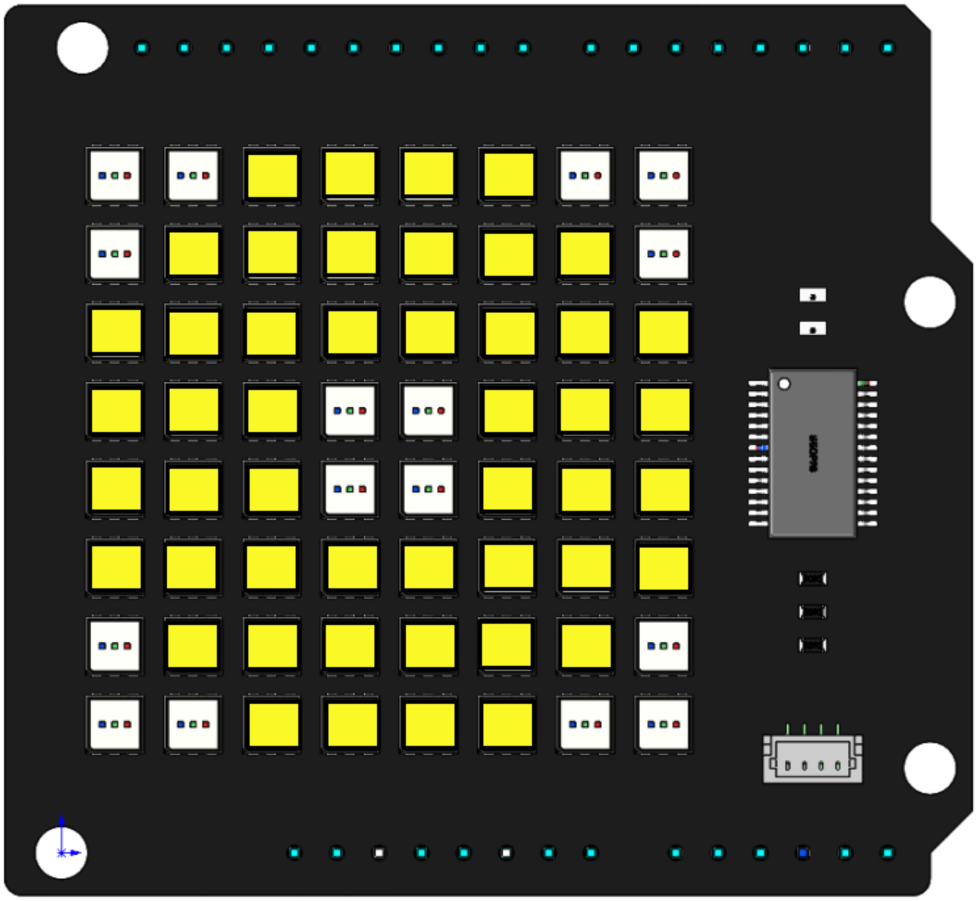
Two for loops are defined in `dazzling_light2()`. The inner loop is to fill the eight vertical lines with eight colors, and the outer loop is to add a value to each color to switch to the next color to achieve the effect of color flow.

Here `ColorHSV()` and `gamma32()` are functions packaged in the library. The former is used to handle decimal numbers, which is equivalent to mapping all the colors of the RGB matrix shield to the range 0 to 65536. `gamma32()` is used for transcoding, converting the return value of `ColorHSV()` into an acceptable argument to `draw_line()`.

4.3 Moving Eye

Introduce

In this project, we will use the `draw_rectangle()` and `draw_point()` functions to draw an eye pattern and achieve the effect of moving the eye around.



Code

When the program is running, you will see an eye moving around on the RGB Matrix Shield.

How it works?

```
byte eye[] = {3,3,4,4};
byte rectangle_arry[] = {0,0,7,7};
byte point_arry[][2] = {{0,0},{1,0},{0,1},{6,0},{7,0},{7,1},
                        {0,6},{0,7},{1,7},{7,6},{7,7},{6,7}};
```

The array `eye[]` represents the coordinates of the pupil, the array `rectangle_arry[]` represents the entire RGB Matrix Shiled, and the array `point_arry[][2]` to represent twelve points in the corners, by means of these 3 array to outline the shape of an eye.

```
void setup() {
  // put your setup code here, to run once:
  RGBMatrixInit();
  draw_rectangle(rectangle_arry,251,248,40);
```

(continues on next page)

(continued from previous page)

```
  for (int i=0; i<sizeof(point_arry); i++){
    draw_point(point_arry[i],0,0,0);
  }
  draw_rectangle(eye,0,0,0);
  image();
}
```

In the setup() function, the entire RGB matrix is lit in yellow, and then the four corner and pupil position LEDs are extinguished so that you can see an eye.

```
void loop() {
  // put your main code here, to run repeatedly:
  up(eye,3);
  delay(100);
  down(eye,6);
  delay(100);
  up(eye,6);
  delay(100);
  down(eye,6);
  delay(100);
  up(eye,3);
  delay(1000);
  right_down(eye,2);
  delay(100);
  up(eye,4);
  delay(100);
  left(eye,4);
  delay(100);
  down(eye,4);
  delay(100);
  right(eye,4);
  delay(100);
  left_up(eye,2);
  delay(1000);
}
```

The main loop is to make the eyeball keep moving up and down, then turn one cycle, and finally return to the original position.

We call some functions to move the eyeball, for example up(eye,3) is to move the eyeball up three squares, now look at how this function is implemented.

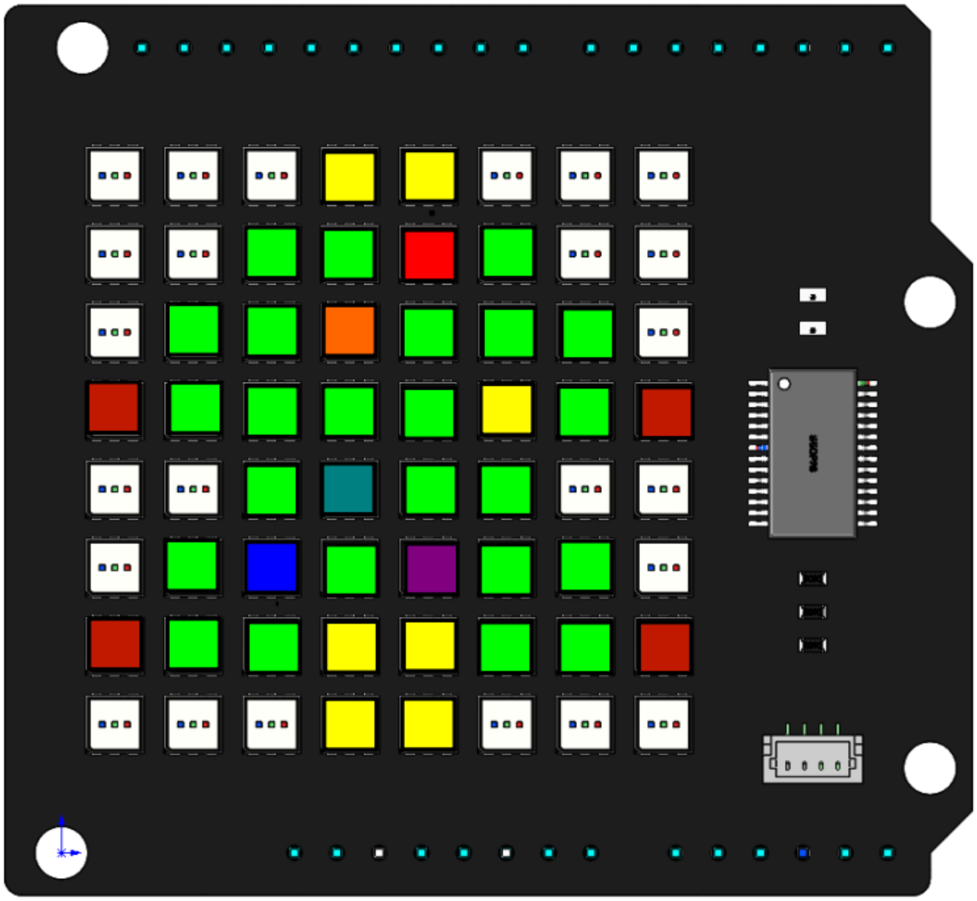
```
void up(byte eye[4],int count=1){
  for (int i=0; i<count; i++){
    draw_rectangle(eye,251,248,40);
    eye[1] -= 1;
    eye[3] -= 1;
    draw_rectangle(eye,0,0,0);
    for (int i=0; i<sizeof(point_arry); i++){
      draw_point(point_arry[i],0,0,0);
    }
        image();
    delay(30);
  }
}
```

The up() function has 2 parameters eye[4] and count, the internal logic is to move the rectangle eye[4] up count squares. (default is 1).

- Define a for() loop with the number of loops determined by count.
- Set the color of the rectangle eye to yellow.
- byte eye[] = {3,3,4,4}; are the 2 diagonal coordinates (3,3) and (4,4), eye[1] and eye[3] are subtracted by one, meaning that the y-values of the 2 diagonal coordinates are subtracted by one.
- Then the modified eye = [3,2,4,3] color is set to (0,0,0) by the function draw_rectangle() and displayed on the dot matrix by the function display().
- The second for() loop is to keep the 12 points on the 4 corners off all the time.
- After one for loop in this way, the pupil is moved up one square.

4.4 Christmas Tree

In this project, we will use the draw_point() function to make a colorful Christmas tree.



Code

When the program runs, you will see a shiny Christmas tree appear on the RGB Matrix Shield.

How it works?

```
byte green[][2] = {{2,1},{3,1},{5,1},
               {1,2},{2,2},{4,2},{5,2},{6,2},
               {1,3},{2,3},{3,3},{4,3},{6,3},
               {2,4},{4,4},{5,4},
               {1,5},{3,5},{5,5},{6,5},
               {1,6},{2,6},{5,6},{6,6}};

byte flash[][2] = {{4,1},{3,2},{5,3},{3,4},{2,5},{4,5}};
byte red[][2] = {{0,3},{7,3},{0,6},{7,6}};
byte yellow[][2] = {{3,0},{4,0},{3,6},{4,6},{3,7},{4,7}};
byte color[7][3] = {{255,0,0},
                {255,102,0},
                {255,255,0},
                {0,255,0},
                {0,128,128},
                {0,0,255},
                {128,0,128}};
```

The Christmas tree is divided into four parts, the red part, the yellow part, the green part, and the blinking part, so we define four arrays to store these coordinates. The array `color[7][3]` stores the 7 colors from which the blinking color will be selected.

```
void setup() {
  // put your setup code here, to run once:
  RGBMatrixInit();
  tree();
}

void loop() {
  // put your main code here, to run repeatedly:
  dot();
}
```

Call the `tree()` function in `setup()` to draw the red, yellow and green parts of the Christmas tree. Call the `dot()` function in `loop()` to make the Christmas tree blink.

```
void tree() {
int lenTotal_green = sizeof(green) / sizeof(byte);
int lenLow_green = sizeof(green[0]) / sizeof(byte);
int lenHigh_green = lenTotal_green / lenLow_green;
for (int i = 0; i < lenHigh_green; i++) {
    draw_point(green[i], 0, 255, 0);
}
...
```

The `tree()` function is used to display the red, yellow and green parts of the Christmas tree on the RGB matrix shield using the `draw_point()` function.

`sizeof()` is an operator that returns the number of bytes a type occupies in memory. * Divide the bytes occupied by the entire two-dimensional array by the bytes occupied by its data type to get the total number of elements. * Divide the bytes occupied by the first one-dimensional array by the bytes occupied by its data type to get the number of elements of each one-dimensional array. * Finally, divide the total number of elements by the number of elements in the 1D array to get the number of 1D arrays, i.e. the number of coordinate points.

For example, *lenHigh_green* is calculated as 24, which is the number of elements in the *green[][2]* array. Then a `for` loop is used to traverse *green[][2]* to draw the dots and fill those dots with green.

The red and yellow parts are also implemented in the same way.

```
int i = 0
void dot(){
  int lenTotal_coor = sizeof(coor) / sizeof(byte);
  int lenLow_coor = sizeof(coor[0]) / sizeof(byte);
  int lenHigh_coor = lenTotal_coor / lenLow_coor;
  for (int j=0; j<lenHigh_coor; j++){
    draw_point(coor[j],color[i][0],color[i][1],color[i][2]);
    i++;
    if (i == 7){
      i = 0;
    }
  }
  image();
  delay(200);
}
```

The `dot()` function fills the six points in `flash[][2]` with seven different colors in order, the colors are chosen from the array `color[7][3]`, for example `{color[0][0],color[0][1],color[0][2]}` means red `{255, 0, 0}`. The `dot()` function can be called in a loop to achieve the blinking effect.

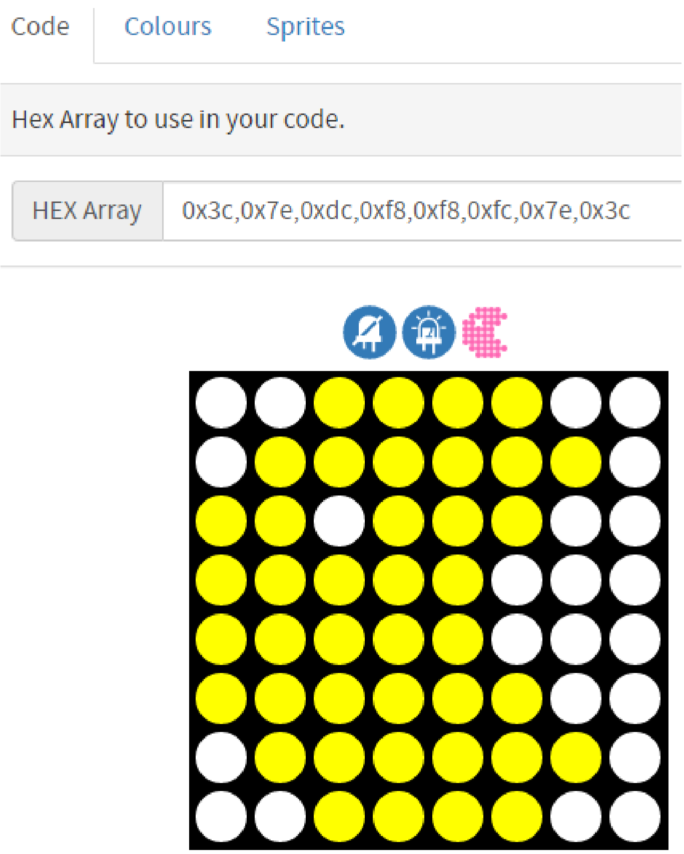
4.5 Custom Shape

To draw interesting patterns on RGB Matrix Shield, we define `ShowHex()` function to facilitate drawing custom patterns.

First you should get the hexadecimal array of the pattern. It is recommended to use the LED Matrix tool, which can be used to design fonts or images for the RGB matrix, and you can also adjust it based on the original pattern.

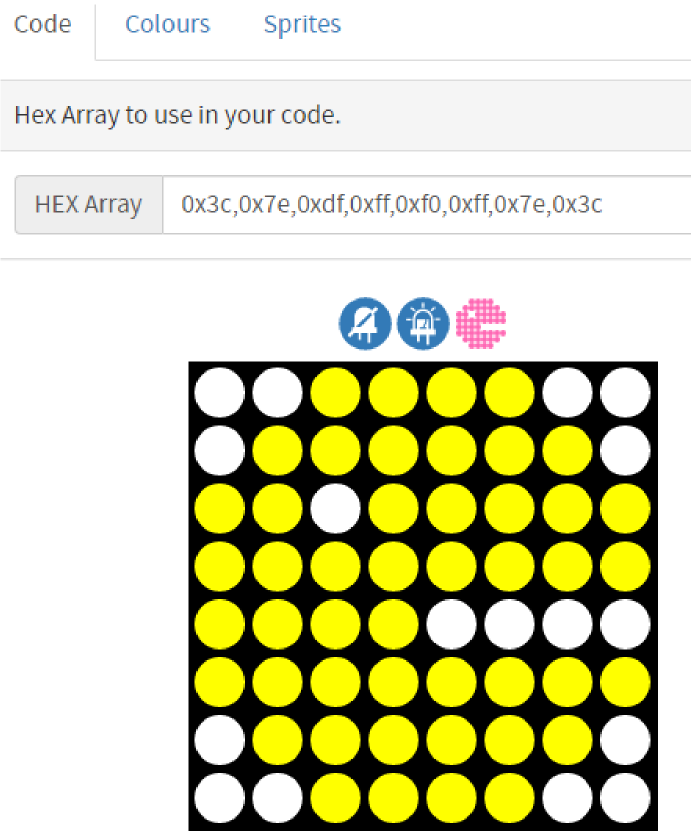
You can select the corresponding character or pattern in the **Sprites** page, then set a specific color in the **Colour** page, and finally get the HEX array of that pattern or character from the **Code** page.

For example, we get two HEX arrays of Pac-Man.



Hex Array to use in your code.

HEX Array 0x3c,0x7e,0xdc,0xf8,0xf8,0xfc,0x7e,0x3c



Hex Array to use in your code.

HEX Array 0x3c,0x7e,0xdf,0xff,0xf0,0xff,0x7e,0x3c

Code

When the program runs, you will see two Pac-Man pictures are constantly switching.

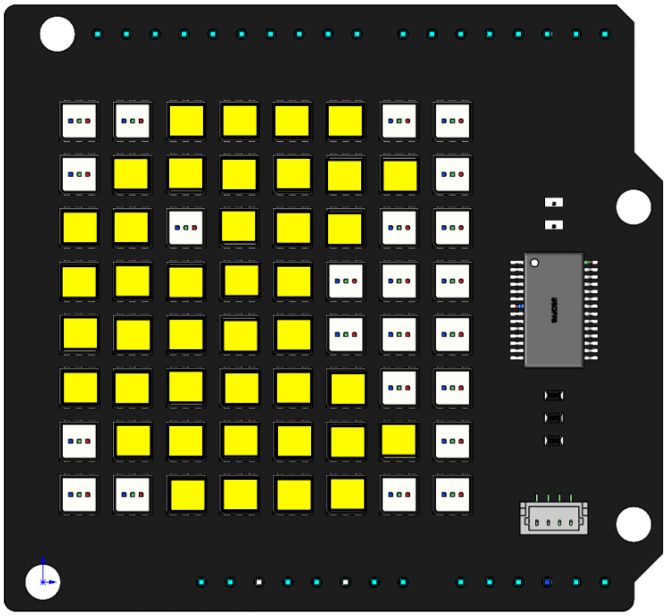
How it works?

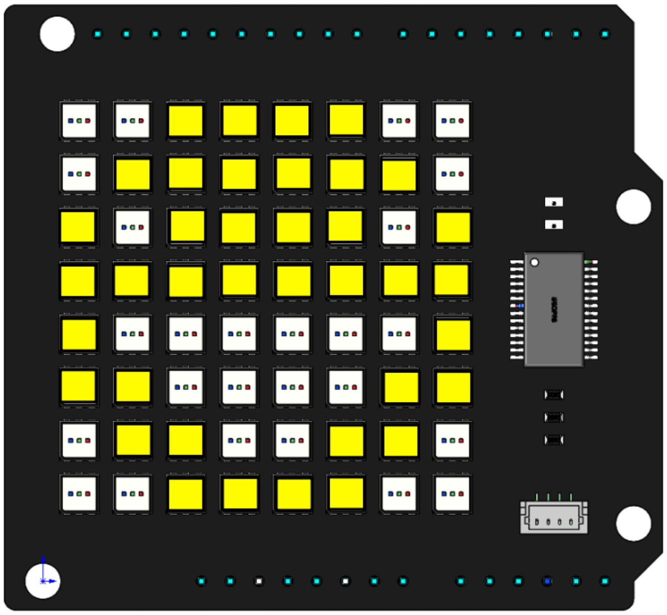
```
void loop() {
// put your main code here, to run repeatedly:
ShowHex(pacman,255,255,0);
delay(1000);
ShowHex(pacman2,255,255,0,1);
delay(1000);
}
```

The main logic is to draw the pattern corresponding to `pacman[]`, after a delay of 1s, move one square to the right to draw the pattern corresponding to `pacman2[]`, cyclically. The fifth parameter in `ShowHex()` is used to determine the position of the pattern in the RGB Matrix Shield. For example, 0 is in the middle, and 1 is one square to the right.

4.6 Custom Dynamic Shape

Now, based on the previous project, make several patterns display more consistently.





Code

When the program runs, Pac-Man will move to the right, then it will stop and turn its head to smile at you, and finally continue to move to the right.

How it works?

```
byte pacman[]={0x3c,0x7e,0xdc,0xf8,0xf8,0xfc,0x7e,0x3c};
byte pacman2[]={0x3c,0x7e,0xdf,0xff,0xf8,0xff,0x7e,0x3c};
byte rotate[]={0x3c,0x7e,0xde,0xff,0xc0,0xff,0x7e,0x3c};
byte normal[]={0x3c,0x7e,0xbd,0xff,0x81,0xff,0x7e,0x3c};
byte smile[]={0x3c,0x7e,0xbd,0xff,0x81,0xe7,0x7e,0x3c};
byte smile2[]={0x3c,0x7e,0xbd,0xff,0x81,0xc3,0x66,0x3c};
```

Define six hexadecimal arrays to store the patterns of the actions Pac-Man will do.

```
void moving_pacman(){
  for(int i=-7; i<2; i++){
    ShowHex(pacman,255,255,0,i);
    delay(200);
    i++;
    ShowHex(pacman2,255,255,0,i);
    delay(200);
  }
  ShowHex(pacman2,255,255,0,1);
  delay(800);
}
```

The moving_pacman() function is used to display the open-mouth state (`pacman[]`) and the closed state (`pacman2[]`) of Pac-man alternately and move from the left side to the right side, finally displaying the closed state (`pacman2[]`).

The fourth parameter of `ShowHex()` can determine the position of the pattern on the RGB Matrix Shield. So use a `for` loop to make Pac-Man appear in the position `i=-7` to `i=1` to achieve the effect of moving.

```
void smile_man(){
  ShowHex(normal,255,255,0);
```

(continues on next page)

(continued from previous page)

```
  delay(100);
  for(int i=0; i<4; i++){
    ShowHex(smile,255,255,0);
    delay(200);
    ShowHex(smile2,255,255,0);
    delay(200);
  }
  ShowHex(smile,255,255,0);
  delay(100);
  ShowHex(normal,255,255,0);
  delay(200);
}
```

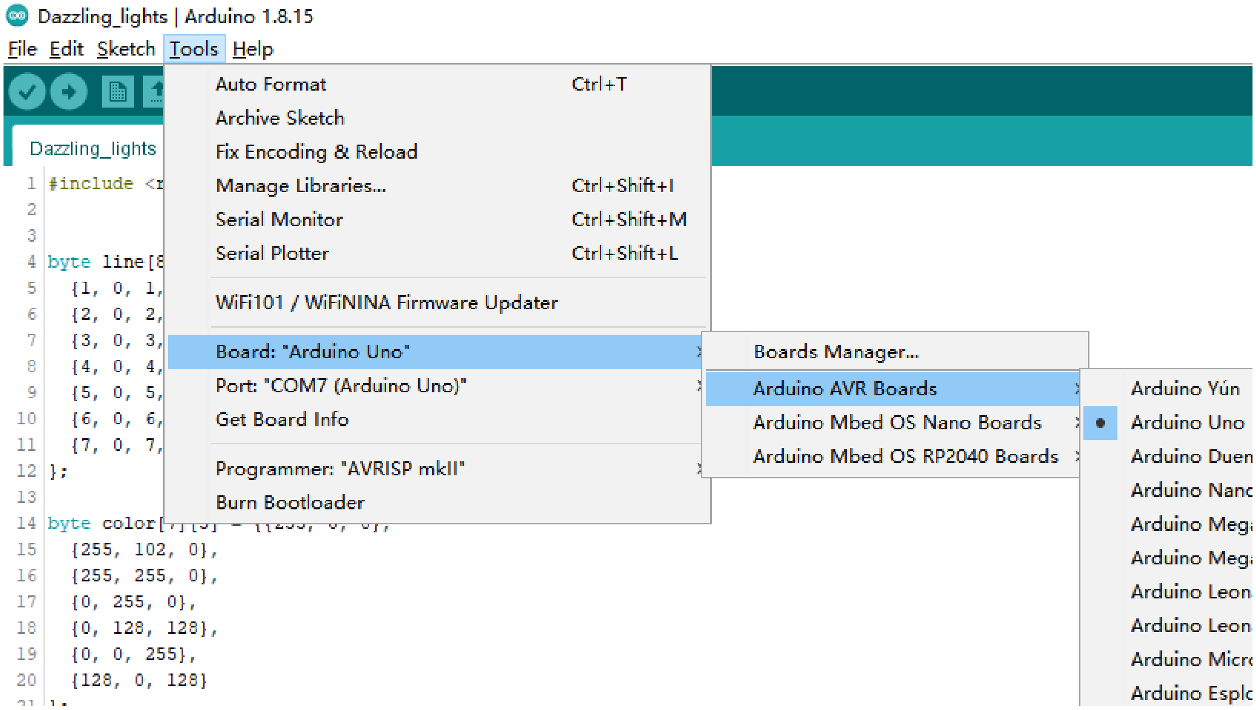
Define a `smile_man()` function to realize the actions of Pac-Man to laugh.

```
void moving_pacman2(){
  for(int i=1; i<8; i++){
    ShowHex(pacman,255,255,0,i);
    delay(100);
    i++;
    ShowHex(pacman2,255,255,0,i);
    delay(100);
  }
}
```

The `moving_pacman2()` function is used to show the actions of continuing to move after a laugh.

Run the sketch

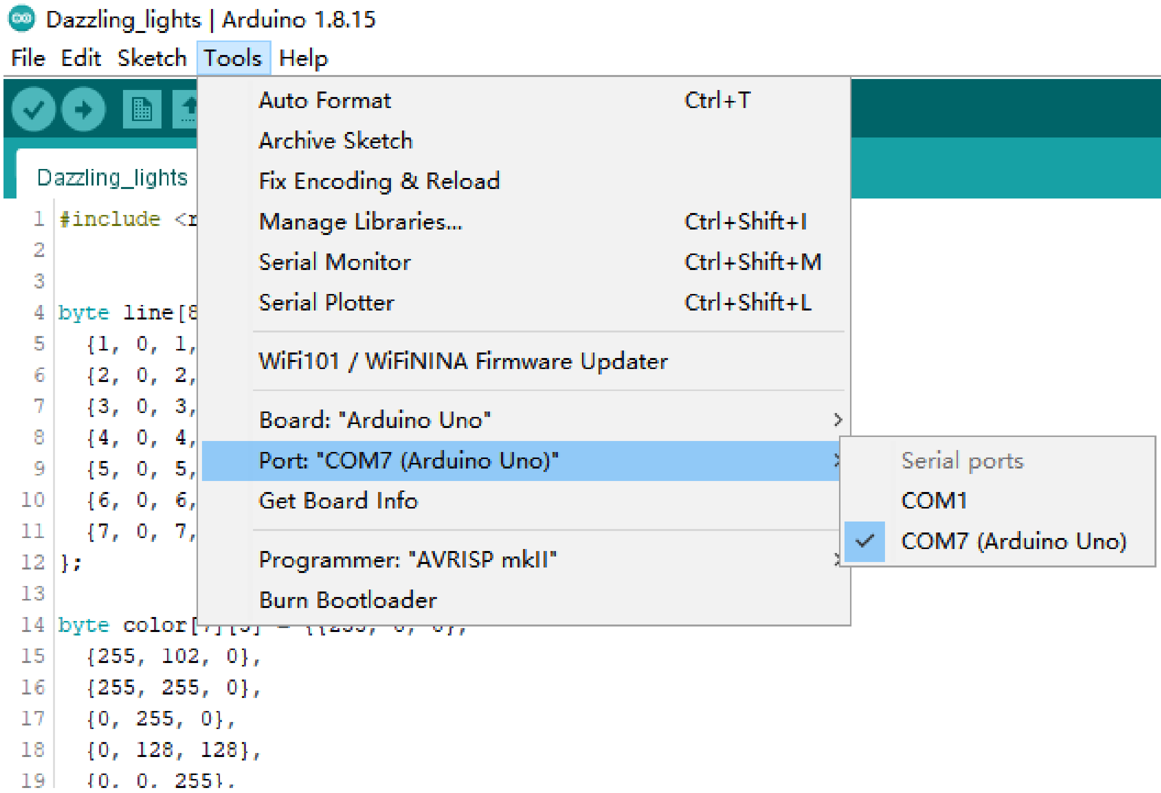
1. Open the one sketch under the path `rgb_matrix\arduino`.
2. Select the Board and Port.



```
#include <r

byte line[8
  {1, 0, 1,
  {2, 0, 2,
  {3, 0, 3,
  {4, 0, 4,
  {5, 0, 5,
  {6, 0, 6,
  {7, 0, 7,
};

byte color[7][3] = {{255, 0, 0},
  {255, 102, 0},
  {255, 255, 0},
  {0, 255, 0},
  {0, 128, 128},
  {0, 0, 255},
  {128, 0, 128}
```

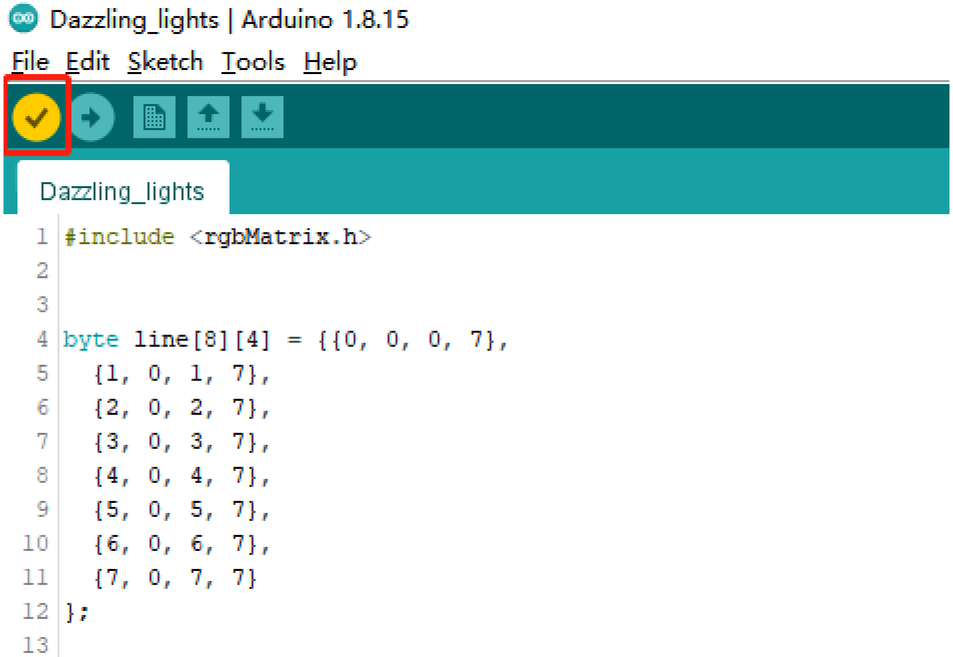

Dazzling_lights | Arduino 1.8.15
File Edit Sketch Tools Help
Dazzling_lights
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port: "COM7 (Arduino Uno)"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM1
COM7 (Arduino Uno)

```
#include <r


byte line[8
  {1, 0, 1,
  {2, 0, 2,
  {3, 0, 3,
  {4, 0, 4,
  {5, 0, 5,
  {6, 0, 6,
  {7, 0, 7,
};

byte color[
  {255, 102, 0},
  {255, 255, 0},
  {0, 255, 0},
  {0, 128, 128},
  {0, 0, 255},
```

3. Compile.

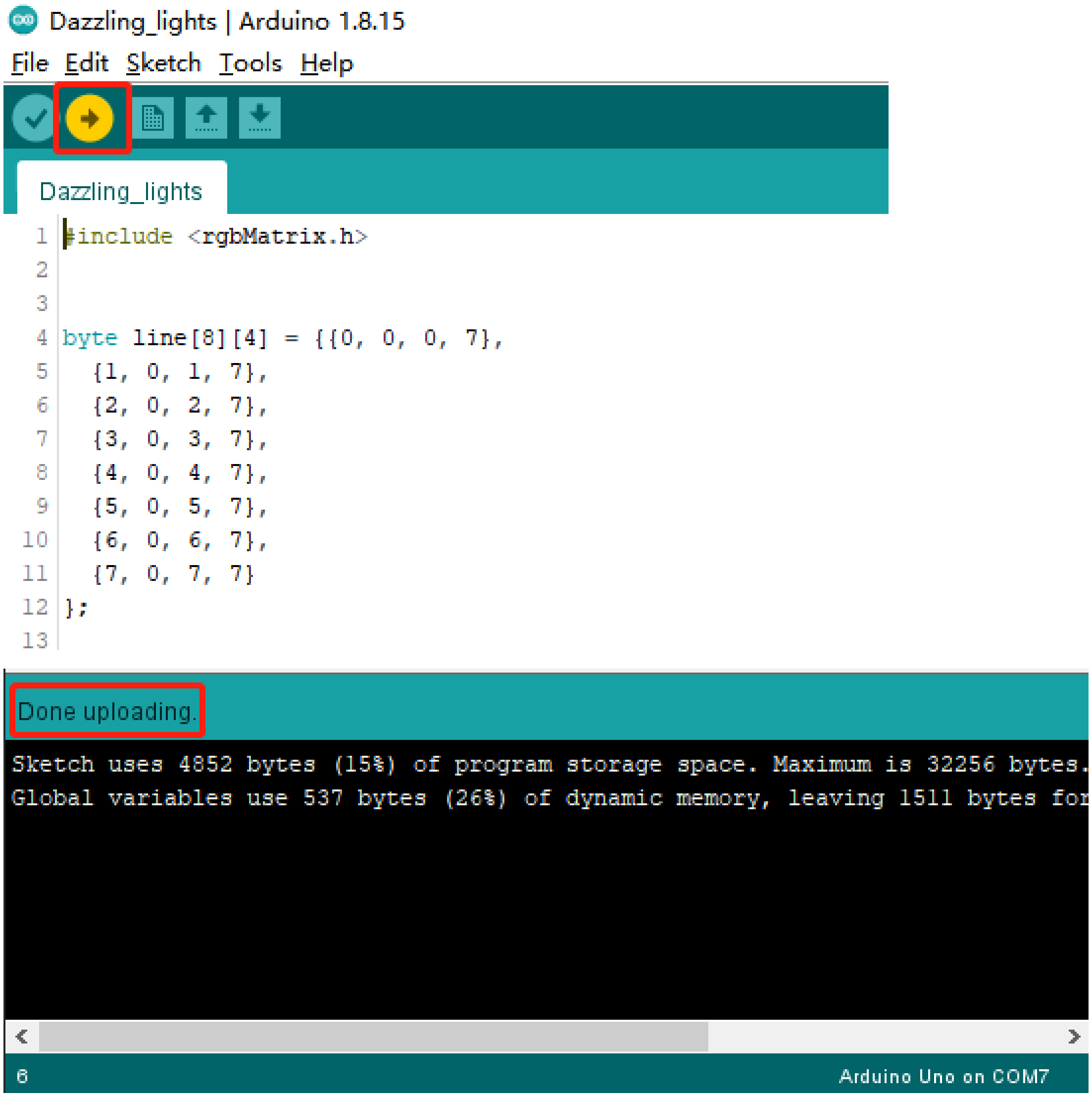


```
#include <rgbMatrix.h>


byte line[8][4] = {{0, 0, 0, 7},
  {1, 0, 1, 7},
  {2, 0, 2, 7},
  {3, 0, 3, 7},
  {4, 0, 4, 7},
  {5, 0, 5, 7},
  {6, 0, 6, 7},
  {7, 0, 7, 7}
};

```

4. Upload.



```
#include <rgbMatrix.h>


byte line[8][4] = {{0, 0, 0, 7},
  {1, 0, 1, 7},
  {2, 0, 2, 7},
  {3, 0, 3, 7},
  {4, 0, 4, 7},
  {5, 0, 5, 7},
  {6, 0, 6, 7},
  {7, 0, 7, 7}
};

```

Done uploading.

```
Sketch uses 4852 bytes (15%) of program storage space. Maximum is 32256 bytes.
Global variables use 537 bytes (26%) of dynamic memory, leaving 1511 bytes for
```

CHAPTER

FIVE

COPYRIGHT NOTICE