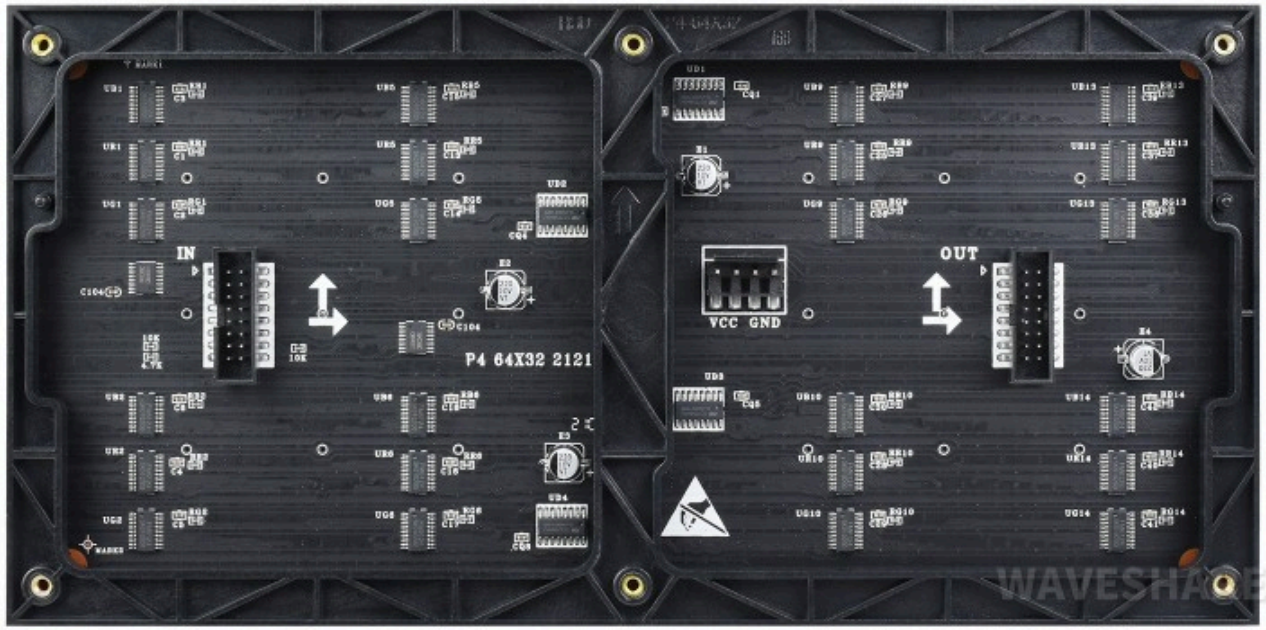
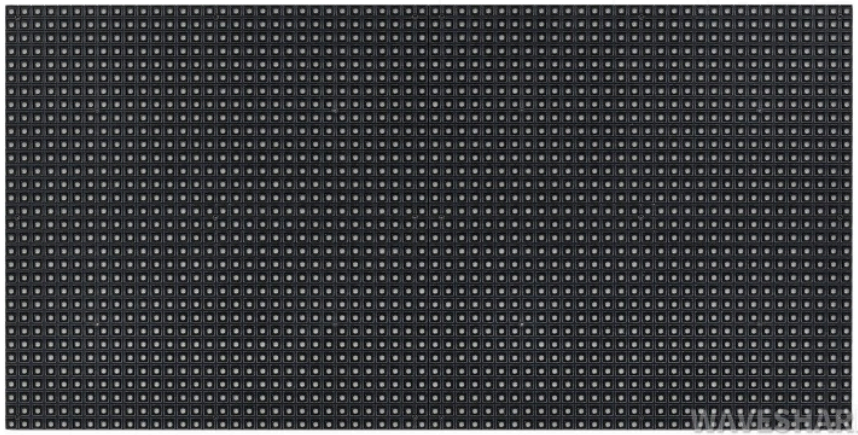
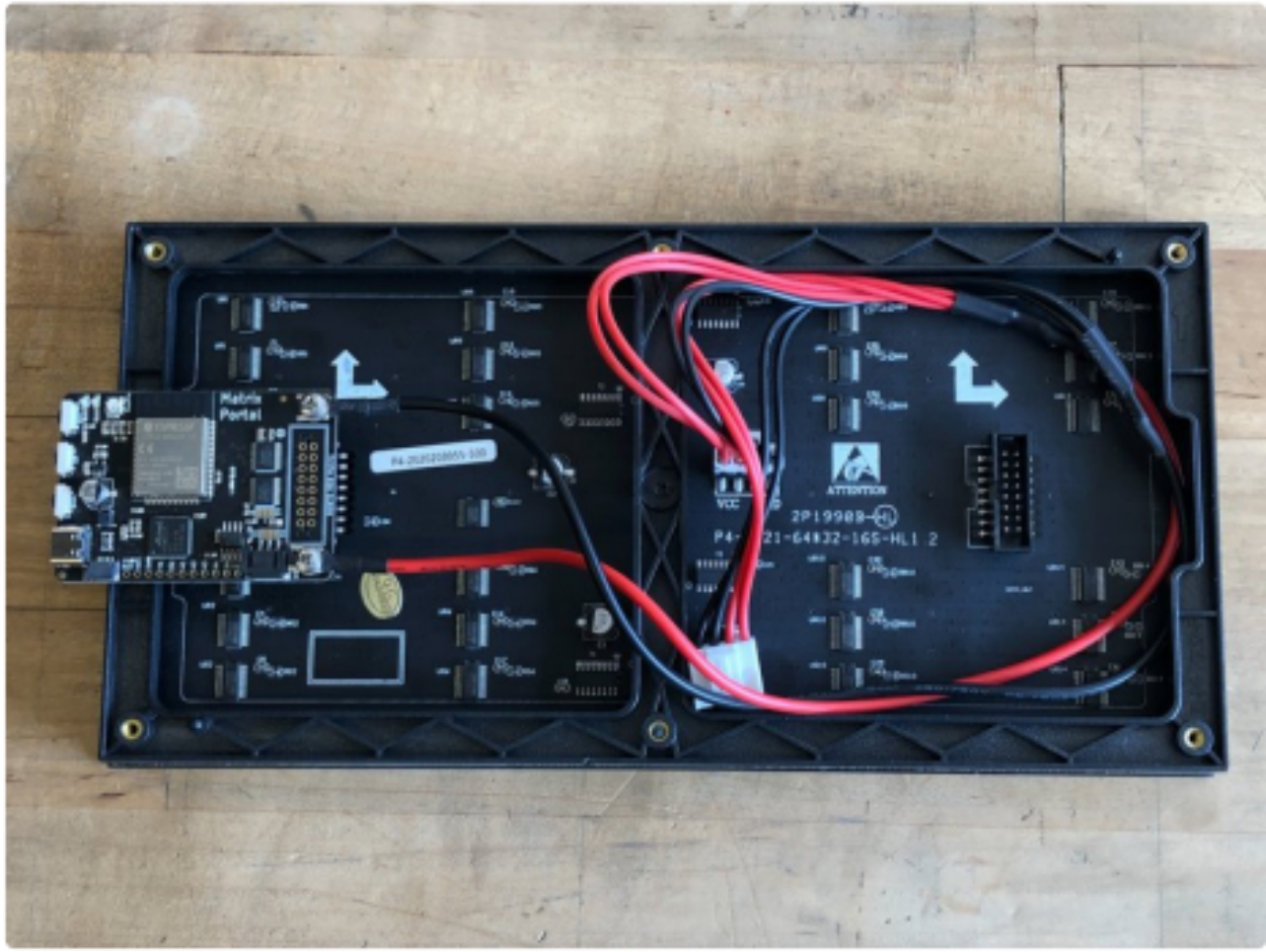
Przykładowy sprzęt

Wyświetlacz matrycowy LED RGB z HUB75





Mikrokontroler Adafruit MatrixPortal M4



Czym jest framebuffer

Framebuffer (bufor ramki) - jest to obszar pamięci, który przechowuje dane o tym, co ma być wyświetlone na ekranie lub wyświetlaczu. W przypadku macierzy RGB, framebuffer przechowuje informacje o kolorach każdej diody LED (piksela) na macierzy.

Jak działa framebuffer?

1. **Przechowywanie danych**: Framebuffer to tablica w pamięci, która przechowuje wartości kolorów dla każdego piksela na wyświetlaczu.
2. **Renderowanie**: Gdy program chce coś wyświetlić na ekranie, najpierw aktualizuje framebuffer, zapisując w nim nowe dane (np. kolory pikseli). Następnie dane z framebuffera są przesyłane do wyświetlacza (np. macierzy RGB), który odtwarza je na fizycznych diodach LED.

Framebuffer w kontekście macierzy RGB

W przypadku macierzy RGB, framebuffer jest używany do przechowywania informacji o stanie każdej diody LED. Oto jak to działa:

- Reprezentacja macierzy: Framebuffer to tablica 2D, gdzie każdy element odpowiada jednemu pikselowi na macierzy RGB. Na przykład, dla macierzy 32x32, framebuffer będzie miał rozmiar 32x32, a każdy element będzie przechowywał wartości kolorów (np. RGB).

Moduły CircuitPython przydatne do wyświetlania

rgbmatrix

Niskopoziomowy moduł do połączenia się z naszym wyświetlaczem, sterownik. Transmituje dane do matrycy, czyli zapala lampki. Dokumentacja dostępna tutaj.

displayio

Wysokopoziomowy moduł. Pozwala nam definiować co chcemy wyświetlać.

Dokumentacja dostępna tutaj.

framebufferio

Moduł jest pośrednikiem między displayio, a rgbmatrix. Zarządza framebufferem wyświetlacza, synchronizuje kiedy i co ma się pokazywać. Dokumentacja dostępna tutaj.

Prosty przykład

examples/REPLprint

Więcej o displayio

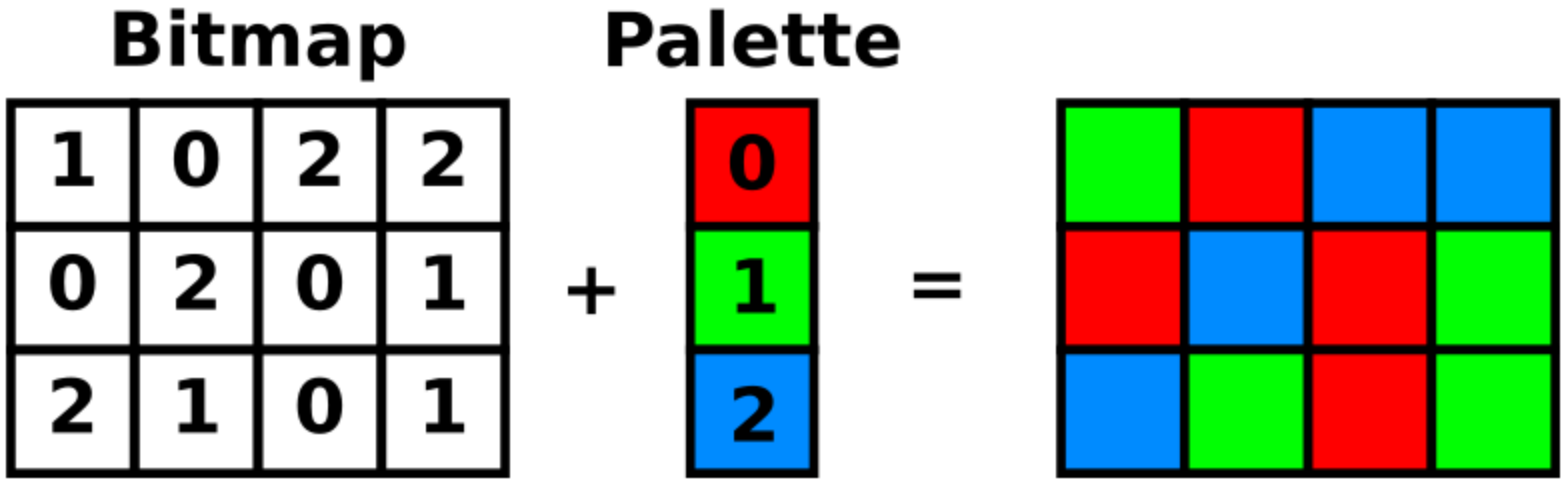
Palette - tablica kolorów

```
palette = displayio.Palette(3)

palette[0] = 0xFF0000 # red
palette[1] = 0x00FF00 # green
palette[2] = 0x0000FF # blue
```

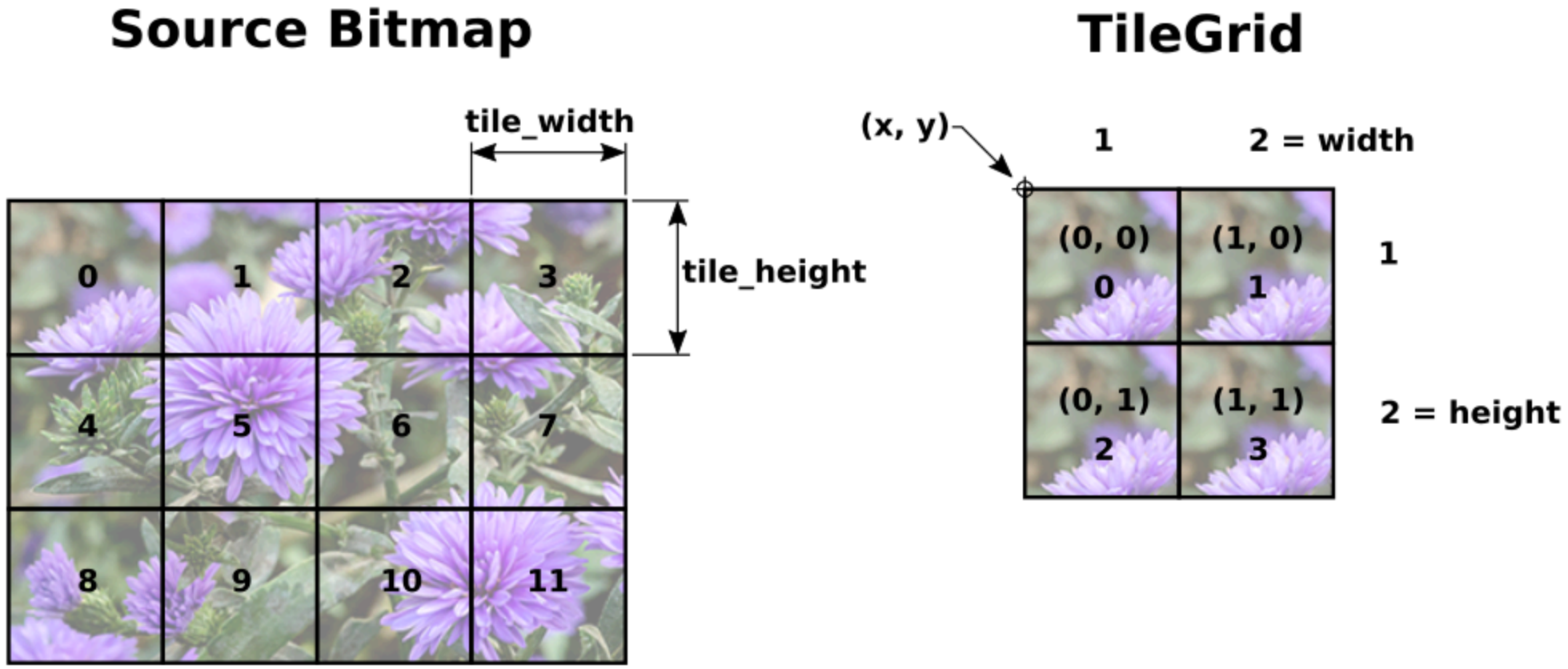
Bitmap - tablica 2d wartości z Pallete

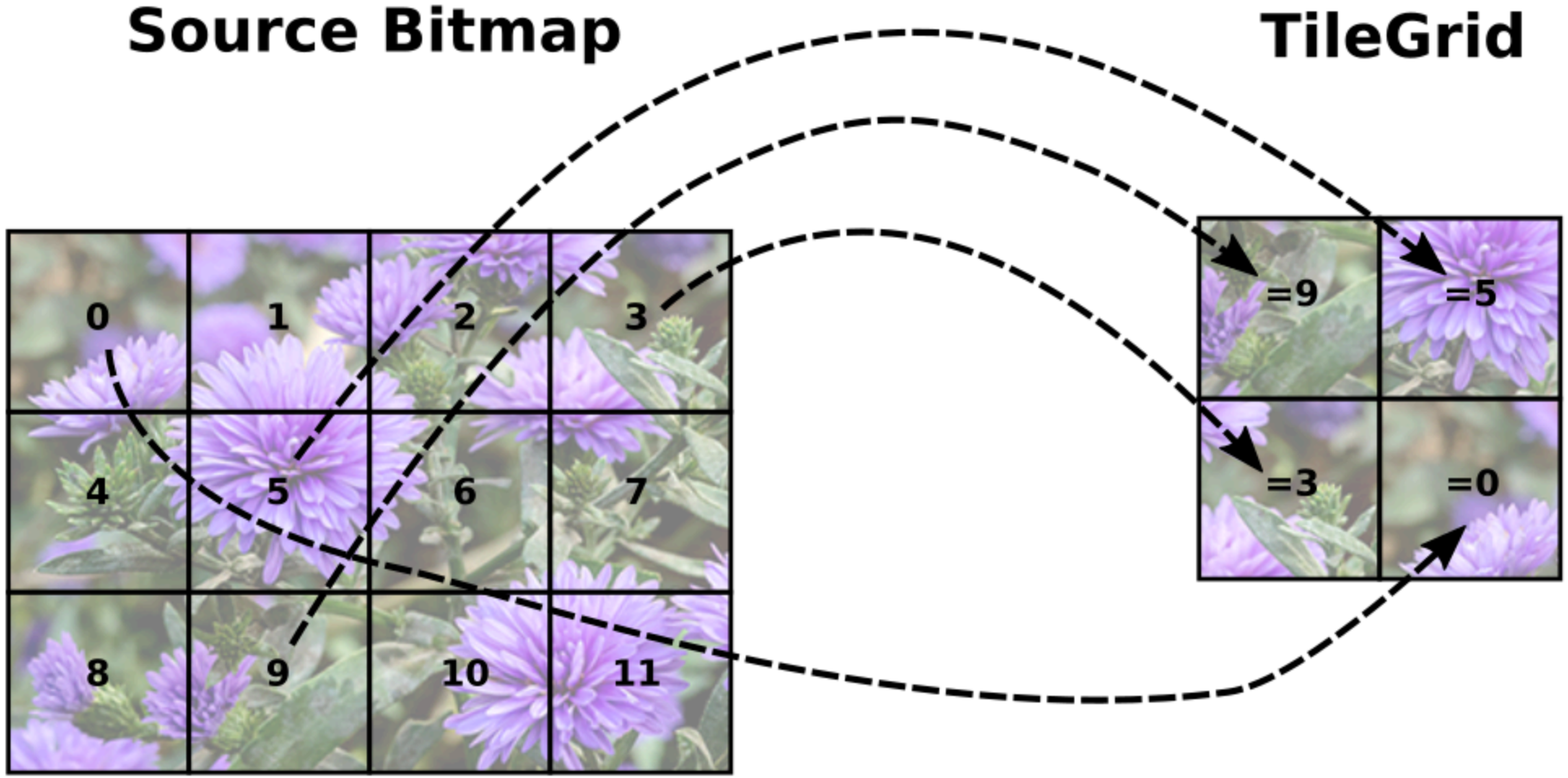
```
#displayio.Bitmap(width: int, height: int, value_count: int)
bitmap = displayio.Bitmap(320, 240, 3)

#set all pixels with 1
bitmap.fill(1)

# set the pixel at (x, y) = (23, 42) to a value of 2
bitmap[23, 42] = 2
```

Bitmap + Pallete



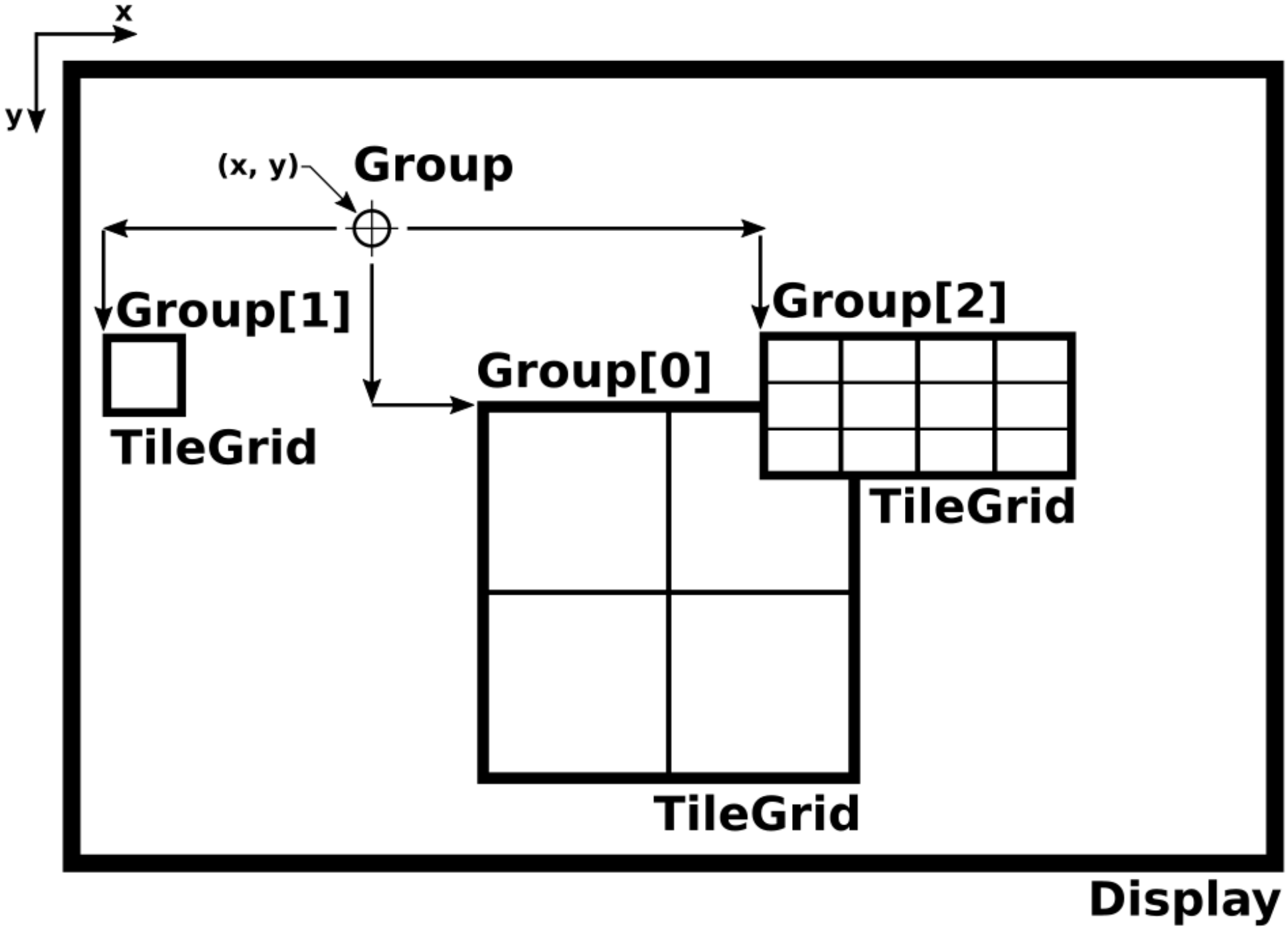
TileGrid - siatka kafelków z bitmapy, łaczy wartości bitmapy z kolorami z Pallete




Source Bitmap
TileGrid
0
1
2
3
4
5
6
7
8
9
10
11
=9
=5
=3
=0

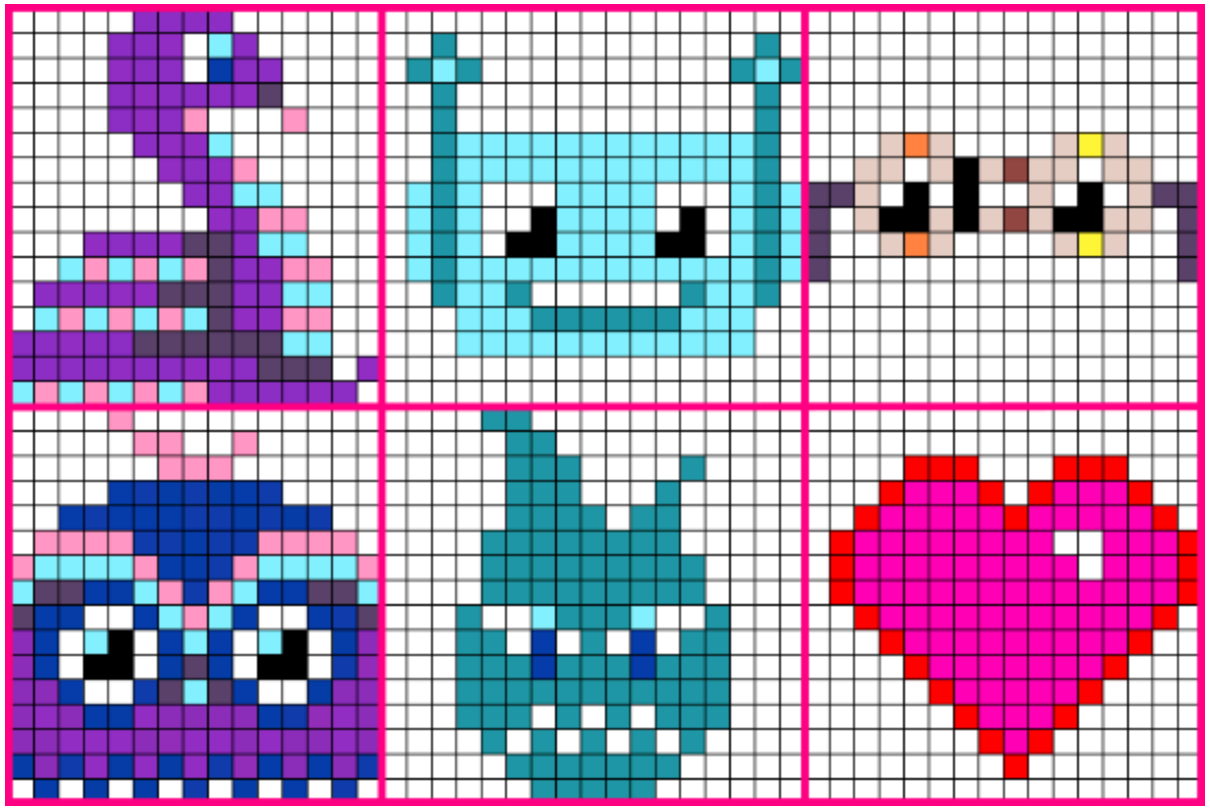
Jest to jak nieskończona wycinanka. Tniemy bitmape i układamy TileGrid.

Group - grupuje, łączy inne elementy, podobie jak znacznik div. Potrzebna nam jest do wyświetlenia.


x
y
(x, y)
Group
Group[1]
Group[2]
Group[0]
TileGrid
TileGrid
TileGrid
Display

Przykład

examples/SpriteSheet inspiracja z tej strony



Przydatne biblioteki

- adafruit_imageload - ładuje bitmape, pallete z pliku
- adafruit_display_text - pomaga wyświetlić tekst
- adafruit_display_shapes - pomaga wyświetlać figury

Co jeśli to dla mnie za mało?

- displayio kurs
- displayio docs
- rgbmatrix docs
- framebufferio docs