Bibliothèque FastLED

Ce qui suit est une traduction partielle et donc incomplète mais néanmoins magnifique de la documentation de la bibliothèque FastLED. La version originale se trouve ici: FastLED

À noter que cela commence à partir de cette page: https://github.com/FastLED/FastLED/wiki/Basic-usage

J'ai esquivé les 3 premiers paragraphes...

Utilisation de base

```
#include <FastLED.h> //inclus la bibliothèque FastLED.h
#define NUM_LEDS 60 //mets en place une macro qui fait que quand NUM_LEDS sera appelé, il sera remplacé par 60
#define DATA_PIN 6 //idem mais avec DATA_PIN et 6
CRGB leds[NUM_LEDS]; //bloc de mémoire qui sera utilisé pour stocker et manipuler la donnée leds, cela mets en place un tableau que l'on
peut manipuler pour définir ou nettoyer les données concernant les led.
```

Les lignes suivantes permettent de mettre en place nos led, cela tient en une seule ligne. Cela dit à la bibliothèque qu'il y a un ruban de Neopixel sur la broche 6. Ces led vont utiliser le tableau “leds” et elles sont au nombre de NUM_LEDS (c-à-d 60).

```
void setup() {
       FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);
   }
```

Faire en sorte que les leds produisent des couleurs est un processus en deux parties avec cette bibliothèque. D'abord, tu établies les valeurs des entrées dans le tableau “leds” en fonction des couleurs que tu veux. Ensuite tu dis à la bibliothèque de te montrer les données. Tes animations/code/motifs seront constituées par ce cycle. Décide de tout ce que tu veux montrer, configure-le et ensuite dis au bandeau de led de le montrer. On va faire quelque chose de très simple, allumer la première led en rouge:

```
  void loop() {
       leds[0] = CRGB::Red;
       FastLED.show();
       delay(30);
    }
```

Changer une LED

Tu peux changer la valeur que tu as passée à une led entre les différents appels, ainsi la fois suivante que tu appelleras la nouvelle valeur prendra effet. Donc, nous pouvons établir la valeur de cette première led en rouge, nous la laisserons ainsi pendant 1 seconde, ensuite nous la passerons en noir, nous la laisserons aussi pendqnt 1 seconde en noir et on continue comme ça, encore et encore...

```
void loop() {
      // Turn the first led red for 1 second
      leds[0] = CRGB::Red;
      FastLED.show();
      delay(1000);
      // Set the first led back to black for 1 second
      leds[0] = CRGB::Black;
      FastLED.show();
      delay(1000);
    }
```

Bouger une led

Maintenant la led clignote. C'est tout bien et tout, mais il y a 60 led sur le ruban! Les autres leds se sentent certainement négligées. Et si on faissait descendre un point en led tout du long du ruban? Souvenez-vous, on a un tableau de 60 leds, on devrait pouvoir faire des choses avec ça. Voilà un petit truc pour faire bouger notre point. On veut mettre notre première led, disons, en bleu, montrer la led, remettre la led en noir, attendre un petit peu, puis recommencer à partir de la second led. Ainsi de suite jusqu'a ce que nous ayons allumer toutes les leds.

```
void loop() {
        for(int dot = 0; dot < NUM_LEDS; dot++) {
```

```
            leds[dot] = CRGB::Blue;
            FastLED.show();
            // clear this led for the next time around the loop
            leds[dot] = CRGB::Black;
            delay(30);
        }
    }
```

Utiliser un contrôle externe

On va faire quelque chose d'un peu différent. Disons que tu as un potentiomètre relié à ton arduino à la broche numérique 2. Il te donnes une valeur comprise entre 0 et 1023. Et si on utilisait cette valeur pour choisir le nombre de led qui seront allumées? On peut utiliser la fonction “map” de arduino pour convertir proportinellement la valeur entre 0 et 1023 vers 0 et NUM_LEDS. Voici à quoi peut ressembler cette fonction:

```
void loop() {
        int val = analogRead(2);
        int numLedsToLight = map(val, 0, 1023, 0, NUM_LEDS);

        // First, clear the existing led values
        FastLED.clear();
        for(int led = 0; led < numLedsToLight; led++) {
            leds[led] = CRGB::Blue;
        }
        FastLED.show();
    }
```

Comment régler la couleur d'une led

Il y a beaucoup de façons de régler la couleur d'une led, cette page donnera de courts exemples parmi ces différentes façons.

Régler la couleur RGB

Voici 6 façons de régler la couleur rgb d'une led:

1. indiquer les valeurs individuelles de R, G et B dans leurs champs respectifs, la façon classique:

```
leds[i].r = 255;
leds[i].g = 68;
leds[i].b = 221;
```

2. régler RGB avec un code couleur hexadécimal unique:

```
leds[i] = 0xFF44DD;
```

3. en utilisant un code de couleur web/HTML standard:

```
leds[i] = CRGB::HotPink;
```

4. en utilisant 'setRGB' et trois valeurs d'un seul coup:

```
leds[i].setRGB( 255, 68, 221);
```

5. copier la couleur RGB d'une autre led:

```
leds[i] = leds[j];
```

6. utiliser un nouveau “fill_solid”, en lui disant de remplir une seule led. Notez que c'est une façon un peu stupide de régler un seul pixel, mais cela nous permet d'illustrer l'existence de fill_solid, une nouvelle fonction que la bibliothèque fournie et qu'elle est vachement bien:

```
fill_solid( &(leds[i]), 1 /*number of leds*/, CRGB( 255, 68, 221) )
```

Régler une couleur HSV

Il existe 6 façons de définir une couleur en HSV (Hue, Saturation, Value). En général, elle recourt à l'attribution d'une couleur

CHSV à une couleur CRGB; la conversion se faisant de manière automatique en appelant hsv2rgb_rainbow. Il est peu de dire qu'un spectre et un arc-en-ciel sont deux choses différentes; les arc-en-ciel sont plus équilibrés visuellement au niveau des couleurs et ont plus de jaune et d'orange que le spectre n'en a. Tu veux certainement commencer avec “rainbow” et basculer vers “spectrum” seulement si tu as des besoins particuliers.

1.Utiliser une couleur rainbow avec hue entre 0 et 255, saturation 0-255 et brightness 0-255:

```
// Simplest, preferred way: assignment from a CHSV color
    leds[i] = CHSV( 224, 187, 255);

    // Alternate syntax
    leds[i].setHSV( 224, 187, 255);
```

2. Réglage d'une teinte arc-en-ciel saturée et lumineuse:

```
leds[i].setHue( 224);
```

3. Utiliser une couleur “spectrum” avec une teinte entre 0 et 255:

```
CHSV spectrumcolor;
    spectrumcolor.hue =         222;
    spectrumcolor.saturation =     187;
    spectrumcolor.value =         255;
    hsv2rgb_spectrum( spectrumcolor, leds[i] );
```

4. Utiliser une couleur “spectrum” avec une teinte entre 0 et 255:

```
CHSV spectrumcolor192;
    spectrumcolor192.hue =         166;
    spectrumcolor192.saturation =     187;
    spectrumcolor192.value =     255;
    hsv2rgb_raw( spectrumcolor192, leds[i] ); //raw
```

5. utiliser un nouveau “fill_solid”, en lui disant de remplir une seule led. Notez que c'est une façon un peu stupide de régler un seul pixel, mais cela nous permet d'illustrer l'existence de fill_solid, une nouvelle fonction que la bibliothèque fournie et qu'elle est vachement bien:

```
fill_solid( &(leds[i]), 1 /*number of leds*/, CHSV( 224, 187, 255) );
```

6. Avec fill_rainbow. Notez que c'est une façon un peu stupide de régler un seul pixel, mais cela nous permet d'illustrer l'existence de fill_rainbow, une nouvelle fonction que la bibliothèque fournie et qu'elle est vachement bien:

```
fill_rainbow( &(leds[i]), 1 /*led count*/, 222 /*starting hue*/);
```

Lire les données RGB depuis le moniteur série

Une valeur CRGB n'est rien d'autre qu'une passerelle assez simple en 3 bytes pour les données brutes RGB: 1 byte pour le rouge, un pour le bleu et un pour le vert.

Vous êtes invités à adresser directement la mémoire sous-jacente. Ces exemples vous montrent comment lire directement les données brutes RGB depuis le flux serial dans votre tableau de valeurs de leds:

1. lire les données RGB d'un seul pixel directement dans un CRGB

```
Serial.readBytes( (char*)(&leds[i]), 3); // read three bytes: r, g, and b.
```

2. Lire toutes les données RGB d'un ruban directement dans le tableau leds en une seule fois:

```
Serial.readBytes( (char*)leds, NUM_LEDS * 3);
```