Du pixelart sur vos écrans OLED

Le pixel art c'est cool ! si vous voulez le rendre encore plus rétro quoi de mieux qu'un petit écran OLED. Nous allons voir comment créer nos propres images et les animer.

Difficulté **Moyen** | Durée **2 heure(s)** | Catégories **Art, Électronique, Jeux & Loisirs**

Coût **5 EUR (€)**

Sommaire

Introduction

Dessiner sur des écrans OLED ? Ça peut paraitre compliqué, mais avec un peu d'entrainement, on peut faire un paquet de choses comme son propre logo, voir de l'animation ou même des jeux vidéos !

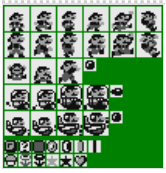
Matériaux

- Microcontrôleur compatible avec u8g2 (arduino / esp8266 / esp32 etc...)
- Écran OLED monochrome

Outils

- https://github.com/maditnerd/oled_xbm
- https://create.arduino.cc/editor/madnerd/0a94bf35-ada8-4695-9f56-d62ea2ed4cff/preview
- https://create.arduino.cc/editor/madnerd/f616433a-3f8a-483c-8ca2-6447dac0314c/preview
- https://create.arduino.cc/editor/madnerd/b4c53d85-5579-4953-89d4-29e0a3b4228c/preview
- https://create.arduino.cc/editor/madnerd/7deed5b1-6f4d-4cd6-b662-0e2a88a93974/preview

Étape 1 - Choisir son image



Nous allons utiliser un écran 128x32 (les écrans en 128x64 sont aussi très répandus si vous voulez plus d'espace), il nous faut donc choisir une **petite image.**
Les sprites de la GameBoy sont très faciles à convertir, mais gardez à l'esprit que nous sommes limités au noir et blanc, là ou la gameboy est capable d'afficher 4 nuances de gris.
Vous pouvez télécharger des sprites sur https://www.spriters-resource.com/ comme point de départ.

Étape 2 - Convertir son image en pixel art

> Tout le code est disponible sur github à cette adresse : https://github.com/maditnerd/oled_xbm

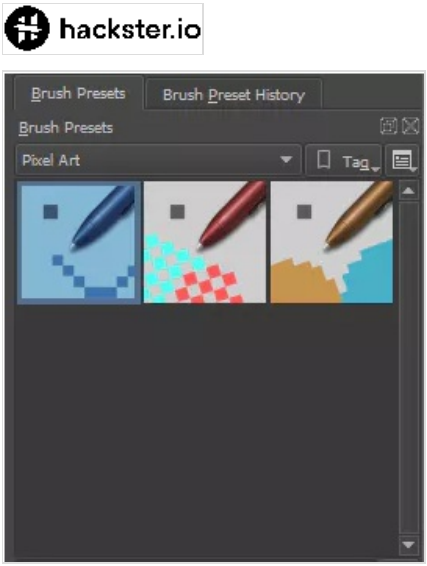
Pour notre premier exemple, nous allons utiliser le logo d'hackster.io
Nous allons

- Redimensionner l'image en 128x32
- Redessiner l'image avec deux couleurs
- La convertir en XBM (code arduino)

> N'importe quel logiciel peut faire l'affaire, mais je recommande chaudement Krita : https://krita.org/en/

Redimensionner l'image à la taille la plus proche de votre écran (128x32) : Image --> **Redimensionner l'image à une nouvelle taille**
Puis augmenter la taille de votre canvas à 128x32 : Image --> **Redimensionner le tableau**
Dans les préréglages des brosses, choisissez Pixel Art
Puis redessiner l'image.

> Appuyer sur X pour passer d'une couleur à l'autre.



Étape 3 - Convertir son pixel art en code arduino

Sauvegarder votre image au format XBM

> Sauvegarder le aussi au format PNG, vous ne pourrez pas modifier votre fichier XBM avec Krita !

Ouvrez le fichier XBM avec un éditeur de texte et changez le nom des variables
Fichier XBM

```
#define _width 128
#define _height 32
static char _bits[] = {
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00...
};
```

Fichier logo.h

```
#define logo_width 128
#define logo_height 32
static const unsigned char logo[] U8X8_PROGMEM = {
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00...
};
```

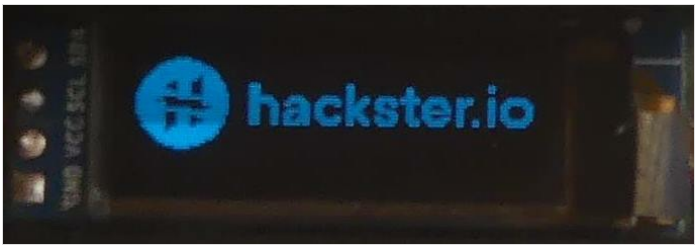
Étape 4 - Afficher votre image

> Je vous conseille de garder le code du logo dans un fichier .h, ainsi il sera plus simple de lire votre code

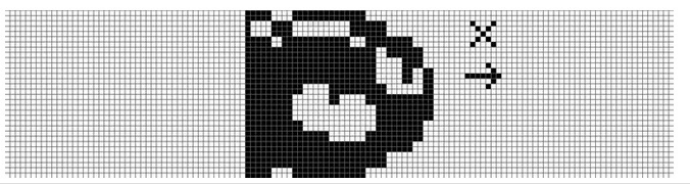
Nous allons utiliser la bibliothèque U8g2 (elle est compatible avec beaucoup d'écrans)

```
#include <Wire.h> //I2C
#include <U8g2lib.h>
#include "logo.h"

//I2C SSD1306 128x32 (search U8g2 examples for other display)
U8G2_SSD1306_128X32_UNIVISION_F_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE);

void setup() {
 u8g2.begin(); //Start Screen
 drawLogo();
}

void loop() {
}

void drawLogo() {
 u8g2.firstPage();
 do {
   u8g2.drawXBMP(0, 0, logo_width, logo_height, logo);
 } while ( u8g2.nextPage() );
}
```

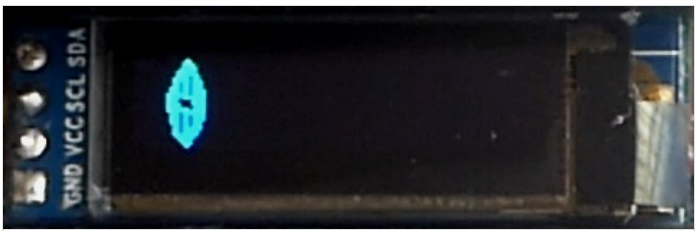
J'ai aussi mis les fichiers PNG et XBM dans un dossier res/ dans mon code afin de savoir ce qu'il y a dans logo.h et de pouvoir le modifier ultérieurement.
Téléversez votre code et voilà !



Étape 5 - Animation - Position



Étape 6 - Animation - Images multiples



Une autre façon d'animer une image et d'utiliser plusieurs images. Au lieu de changer la position X, nous allons utiliser une **condition SWITCH** et augmenter la valeur **animation_state.**

```
void drawAnimation() {
 u8g2.firstPage();
 do {
   switch (animation_state) {
     case 0:
       u8g2.drawXBMP(0, 0, logo_width, logo_height, logo_1);
       break;
     case 1:
       u8g2.drawXBMP(0, 0, logo_width, logo_height, logo_2);
       break;
     case 2:
       u8g2.drawXBMP(0, 0, logo_width, logo_height, logo_3);
       break;
     case 3:
       u8g2.drawXBMP(0, 0, logo_width, logo_height, logo_4);
       break;
     case 4:
       u8g2.drawXBMP(0, 0, logo_width, logo_height, logo_5);
       break;
     case 5:
       u8g2.drawXBMP(0, 0, logo_width, logo_height, logo_6);
       break;
   }
 } while ( u8g2.nextPage() );
}
```

Afin que notre animation soit complète, il nous faut l'inverser, pour cela, nous allons utiliser le booléen (vrai/faux) **animation_direction** afin que l'animation aille dans le sens inverse.

```
drawAnimation();
   if (animation_direction) {
     animation_state--;
   } else {
     animation_state++;
   }
   if (animation_state == 5) {
     animation_direction = true;
   }
   if (animation_state == 0) {
     animation_direction = false;
   }
```

Si on combine les deux techniques, on peut créer des animations plus complexe : https://www.youtube.com/watch?v=iB52FujIK-A

Notes et références

Si vous voulez être informés des prochains tutoriels suivez moi sur twitter : https://twitter.com/m4dnerd