Gros probléme d'emails avec Yahoo (/annonces/gros-probleme-demails-avec-yahoo/)

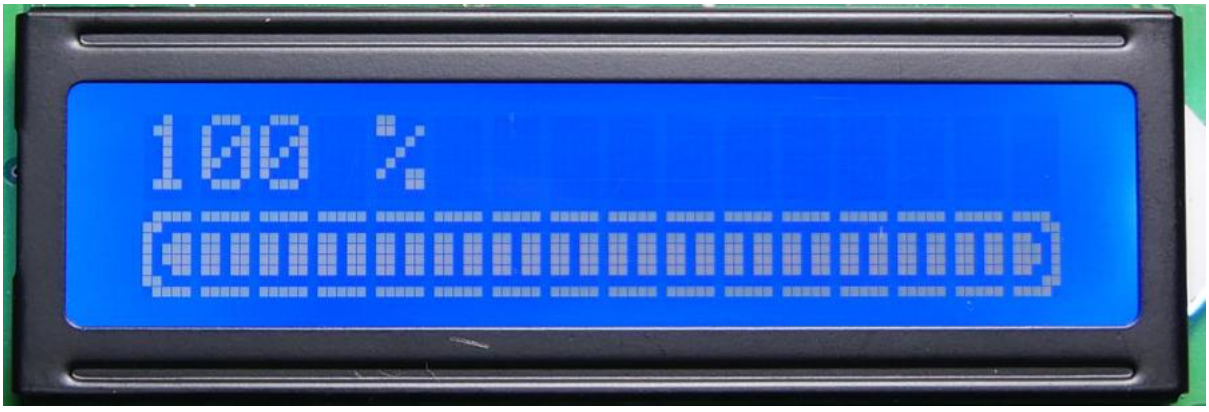
Yahoo refuse tous les emails du site. Si vous avez une adresse chez un autre prestataire, c'est le moment de l'utiliser 😉

En cas de soucis, n'hésitez pas à aller faire un tour sur la page de contact en bas de page.

par skywodd (/membres/skywodd/), le fév. 21, 2017 à 7:27 après-midi

Faire une barre de progression avec Arduino et LiquidCrystal

99% (10 secondes restantes)



par skywodd (/membres/skywodd/) | jan. 30, 2016 | © Licence (voir pied de page)

Catégories : Tutoriels (/articles/categories/tutoriels/) Arduino (/articles/categories/tutoriels/arduino/) Projets (/articles/categories/projets/) | Mots : Arduino (/articles/tags/arduino/) Genuino (/articles/tags/genuino/) LiquidCrystal (/articles/tags/liquidcrystal/) LCD (/articles/tags/lcd/) Progress bar (/articles/tags/progress-bar/)

⚠ Cet article n'a pas été mis à jour depuis un certain temps, son contenu n'est peut être plus d'actualité.

Dans ce tutoriel, nous allons réaliser une barre de progression à l'aide de la bibliothèque LiquidCrystal, d'un écran LCD alphanumériq et d'une carte Arduino / Genuino. Nous commencerons d'abord par faire une barre de progression extrêmement simple. Nous ferons ensuite deux autres itérations du code, un peu plus complexe cette fois, afin d'accéder au Saint Graal de la barre de progression.

Sommaire

Bonjour à toutes et à tous !

Parfois, on se retrouve à devoir attendre qu'un morceau de code finisse son travail avant de pouvoir continuer à utiliser un matériel.

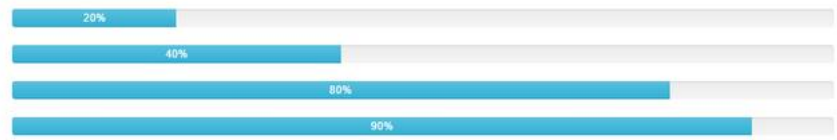
Dans le cadre d'un projet strictement personnel, cela n'est pas très embêtant, il suffit d'attendre. Mais pour certains projets, il est nécessaire de faire pati l'utilisateur, sous peine de le voir s'énerver sur le matériel en hurlant au plantage.

Sur ordinateur ou mobile, faire patienter un utilisateur est trivial et particulièrement jouissif en tant que développeur. Les solutions techniques sont nombre barre de progression, indicateur tournant, indicateur défilant, etc.

Sur Arduino (https://www.arduino.cc), la plage de choix est beaucoup moins vaste et tout doit être fait à la main. Nous allons donc nous intéresser da tutoriel à un grand classique de la mise en attente des utilisateurs : la barre de progression.

Le principe

Une barre de progression est un moyen assez simple de faire patienter et/ou râler un utilisateur.



(https://www.carnetdumaker.net/images/exemple-de-barres-de-progression/)

Exemple de barres de progression

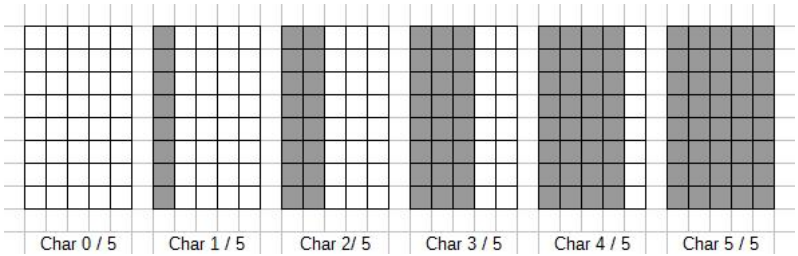
Une barre de progression est une gauge horizontale qui se remplit en fonction du pourcentage de complétude d'une action donnée. Au début, la gauge est puis elle se remplit petit à petit, pour au final atteindre 100%.

(https://www.carnetdumaker.net/images/exemple-de-caracteres-personnalises-avec-liquidcrystal/)

Exemple de caractères personnalisés avec LiquidCrystal

Notre barre de progression sauce Arduino va reposer sur une fonctionnalité bien pratique des écrans LCD alphanumériques et de la bibliothèque LiquidC (https://www.arduino.cc/en/Reference/LiquidCrystal) : les caractères personnalisés (https://www.arduino.cc/en/Reference/LiquidCrystalCreateChar).

Cette fonctionnalité permet de créer jusqu'à huit caractères personnalisés de 5 x 8 pixels monochromes.



(https://www.carnetdumaker.net/images/caracteres-personnalises-liquidcrystal-pour-une-barre-de-progression-simpliste/)

Caractères personnalisés pour une barre de progression simpliste

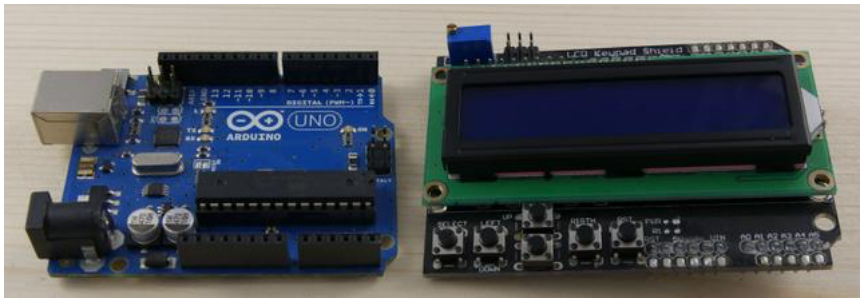
En utilisant cette fonctionnalité, on va pouvoir créer une série de caractères personnalisés pour représenter des unités de 1 / N de caractère. Dans le premier de ce tutoriel, on utilisera par exemple des unités de 1 / 5 de caractère (rappel : un caractère fait 5 x 8 pixels) : 0 / 5, 1 / 5, 2 / 5, 3 / 5, 4 / 5 et 5 / 5.

Chaque colonne de pixels représentera alors une partie de la plage 0% - 100% de la barre de progression.

Toujours avec le premier code de ce tutoriel, on verra qu'un écran alphanumérique classique de 2 x 16 caractères va ainsi permettre d'afficher, sur une r ligne, 16 x 5 = 80 colonnes. Un écran de 4 x 20 caractères permettra lui d'atteindre 20 x 5 = 100 colonnes, soit pile le bon nombre pour afficher des pourcent

Ce tutoriel est réalisé avec un écran 2 x 16 caractères, mais il fonctionne de la même façon avec un écran de 4 x 20 caractères.

Prérequis matériels



(https://www.carnetdumaker.net/images/carte-arduino-uno-et-shield-lcd-dfrobots/)

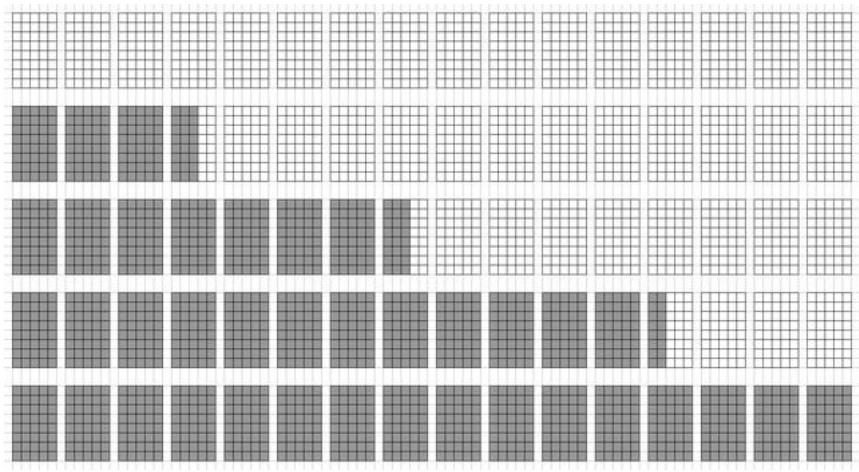
Le matériel nécessaire

Pour ce tutoriel, vous aurez besoin :

- D'une carte Arduino (et de son câble USB),
- D'un écran LCD alphanumérique compatible LiquidCrystal (ici une shield LCD DFRobots sur la photo).

C'est tout, il n'y a besoin que de ces deux composants pour réaliser ce tutoriel.

Barre de progression V1


(https://www.carnetdumaker.net/images/esquisse-dune-barre-progression-simpliste/)

Esquisse d'une barre progression simpliste

Cette première version sera très simple : pas de graphisme compliqué, juste des colonnes de pixels et un produit en croix pour déterminer le nombre de col à noircir en fonction du pourcentage.

Le code de cette première version sera composé de deux fonctions : une fonction pour initialiser les caractères personnalisés et une fonction pour affich pourcentage.

```
/* Inclus la bibliothèque LiquidCrystal pour l'écran LCD */
#include <LiquidCrystal.h>


/* Créer l'objet lcd avec une configuration de broches compatible avec shield LCD de DFRobots */
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);


/* Constantes pour la taille de l'écran LCD */
const int LCD_NB_ROWS = 2;
const int LCD_NB_COLUMNS = 16;
```

On va commencer le code de cette première version en incluant la bibliothèque LiquidCrystal et en déclarant l'objet `lcd` qui nous permettra de commun avec l'écran LCD.

On en profitera aussi pour déclarer deux constantes `LCD_NB_ROWS` et `LCD_NB_COLUMNS`, qui définissent respectivement le nombre de lignes et le nomb colonnes de l'écran LCD.

N.B. J'utilise une shield LCD de DFRobots pour ce montage. Libre à vous de modifier la configuration des broches pour qu'elle corresponde à votre écran.

```
/* Caractères personnalisés */
byte DIV_0_OF_5[8] = {
  B00000,
  B00000,
  B00000,
  B00000,
  B00000,
  B00000,
  B00000,
  B00000
}; // 0 / 5

byte DIV_1_OF_5[8] = {
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000
}; // 1 / 5

byte DIV_2_OF_5[8] = {
  B11000,
  B11000,
  B11000,
  B11000,
  B11000,
  B11000,
  B11000,
  B11000
}; // 2 / 5

byte DIV_3_OF_5[8] = {
  B11100,
  B11100,
  B11100,
  B11100,
  B11100,
  B11100,
  B11100,
  B11100
}; // 3 / 5

byte DIV_4_OF_5[8] = {
  B11110,
  B11110,
  B11110,
  B11110,
  B11110,
  B11110,
  B11110,
  B11110
}; // 4 / 5

byte DIV_5_OF_5[8] = {
  B11111,
  B11111,
  B11111,
```

```
  B11111,
  B11111,
  B11111,
  B11111,
  B11111
}; // 5 / 5


/**
 * Fonction de configuration de l'écran LCD pour la barre de progression.
 * Utilise les caractères personnalisés de 0 à 5 (6 et 7 restent disponibles).
 */
void setup_progressbar() {

  /* Enregistre les caractères personnalisés dans la mémoire de l'écran LCD */
  lcd.createChar(0, DIV_0_OF_5);
  lcd.createChar(1, DIV_1_OF_5);
  lcd.createChar(2, DIV_2_OF_5);
  lcd.createChar(3, DIV_3_OF_5);
  lcd.createChar(4, DIV_4_OF_5);
  lcd.createChar(5, DIV_5_OF_5);
}
```

La fonction `setup_progressbar()` permet d'initialiser les caractères personnalisés dont nous avons besoin pour notre barre de progression.

Cette fonction se contente d'appeler la fonction createChar() (https://www.arduino.cc/en/Reference/LiquidCrystalCreateChar) de la bibliothèque LiquidC pour déclarer chaque caractère personnalisé.

```
/**
 * Fonction dessinant la barre de progression.
 *
 * @param percent Le pourcentage à afficher.
 */
void draw_progressbar(byte percent) {

  /* Affiche la nouvelle valeur sous forme numérique sur la première ligne */
  lcd.setCursor(0, 0);
  lcd.print(percent);
  lcd.print(F(" %  "));
  // N.B. Les deux espaces en fin de ligne permettent d'effacer les chiffres du pourcentage
  // précédent quand on passe d'une valeur à deux ou trois chiffres à une valeur à deux ou un chiffres.

  /* Déplace le curseur sur la seconde ligne */
  lcd.setCursor(0, 1);

  /* Map la plage (0 ~ 100) vers la plage (0 ~ LCD_NB_COLUMNS * 5) */
  byte nb_columns = map(percent, 0, 100, 0, LCD_NB_COLUMNS * 5);

  /* Dessine chaque caractère de la ligne */
  for (byte i = 0; i < LCD_NB_COLUMNS; ++i) {

    /* En fonction du nombre de colonnes restant à afficher */
    if (nb_columns == 0) { // Case vide
      lcd.write((byte) 0);

    } else if (nb_columns >= 5) { // Case pleine
      lcd.write(5);
      nb_columns -= 5;

    } else { // Derniére case non vide
      lcd.write(nb_columns);
      nb_columns = 0;
    }
  }
}
```

La fonction `draw_progressbar()` permet d'afficher le pourcentage sous forme numérique sur la première ligne et la barre de progression sur la deuxième li

Le produit en croix est réalisé par la fonction map() (https://www.arduino.cc/en/Reference/Map) qui est intégrée dans la bibliothèque Arduino. Cette for est bien pratique pour ceux qui, comme moi, sont un peu fâchés avec les mathématiques 😉

L'affichage des caractères se fait dans la boucle `for` à la ligne 22. Cette boucle dessine chaque caractère de la seconde ligne en partant de la gauche.

Pour chaque caractère, un test est réalisé pour déterminer le caractère à afficher.

- Si le nombre de colonnes à afficher est nul : le code affiche une case vide.
- Si le nombre de colonnes à afficher est supérieur ou égal à 5 : le code affiche une case pleine et décrémente le nombre de colonnes à afficher de 5.

- Si le nombre de colonnes à afficher est strictement inférieur à 5 : le code affiche une case non vide correspond au nombre de colonnes à afficher et assi au nombre de colonnes à afficher pour la suite de l'affichage.

N.B. Le cast en byte à la ligne 26 est uniquement là pour faire plaisir au compilateur.

```
1  /** Fonction setup() */
2  void setup(){
3
4    /* Initialise l'écran LCD */
5    lcd.begin(LCD_NB_COLUMNS, LCD_NB_ROWS);
6    setup_progressbar();
7    lcd.clear();
8  }
9
10
11 /** Fonction loop() */
12 void loop(){
13
14   /* Valeur en pourcentage de la barre de progression */
15   static byte percent = 0;
16
17   /* Affiche la valeur */
18   draw_progressbar(percent);
19
20   /* Incrémente le pourcentage */
21   if (++percent == 101) {
22     // Revient à zéro si le pourcentage dépasse 100%
23     percent = 0;
24     delay(1000);
25   }
26
27   /* Petit temps d'attente */
28   delay(100);
29 }
```

Afin de tester notre programme, nous allons ajouter un petit bout de code pour afficher une barre de progression allant de 0% à 100% en quelques secondes.

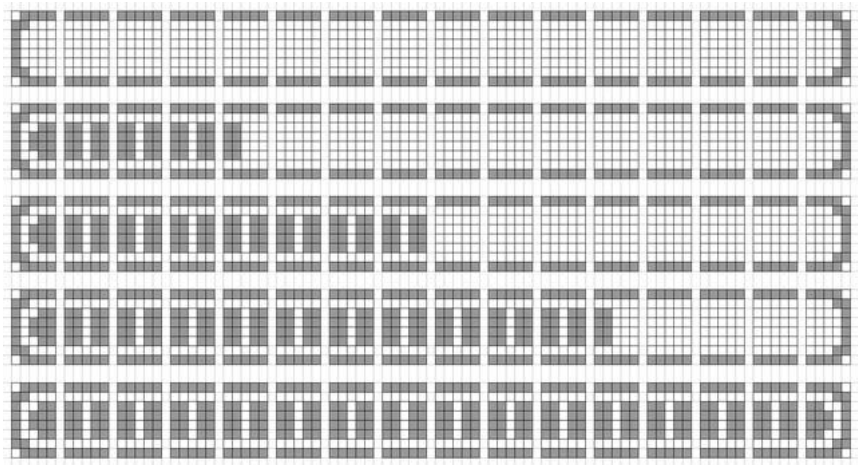
L'extrait de code complet est disponible sur cette page (https://www.carnetdumaker.net/snippets/1/) (le lien de téléchargement en .zip contient le projet Ar prêt à l'emploi).

Résultat

Barre de progression V2

Cette seconde version de notre barre de progression sera un poil plus compliquée que la première version. Dans cette version, on va ajouter un peu de grapl à la barre de progression.

Notre première version, bien que fonctionnelle, souffre d'un gros défaut : on ne sait pas où sont les limites de la barre de progression et l'aspect visuel est u trop basique. De plus, vous avez surement remarqué dans la vidéo ci-dessus que la barre de progression est coupée par un pixel vide entre chaque caractère. une caractéristique physique des écrans LCD alphanumériques. Il va falloir faire avec.



(https://www.carnetdumaker.net/images/esquisse-dune-barre-de-progression-stylisee/)

Esquisse d'une barre de progression avec bordure

Pour donner l'impression d'une vraie barre de progression, on va ajouter une bordure. Cela permettra de savoir où se situent le début et la fin de la bar progression.

(https://www.carnetdumaker.net/images/caracteres-personnalises-dune-barre-de-progression-avec-bordure/)

Caractères personnalisés nécessaire à la réalisation d'une barre de progression avec bordure

Pour contourner le problème du pixel vide entre chaque caractère, on va découper chaque case en 2 unités au lieu de 5, avec un espace vide de 1 pixel ent deux unités. Cela va permettre d'avoir une alternance : colonne, vide, colonne, vide, colonne, vide, etc.

La bordure sera toujours coupée par les pixels vides entre les caractères, mais pas la barre de progression elle-même. Les coupures dans la bordure ne de donc pas se voir du premier coup d'oeil.

Le code de cette seconde version sera composé de deux fonctions, comme pour la première version : une fonction pour initialiser les caractères personnali une fonction pour afficher un pourcentage.

```
/* Caractères personnalisés */
byte START_DIV_0_OF_1[8] = {
  B01111,
  B11000,
  B10000,
  B10000,
  B10000,
  B10000,
  B11000,
  B01111
}; // Char début 0 / 1

byte START_DIV_1_OF_1[8] = {
  B01111,
  B11000,
  B10011,
  B10111,
  B10111,
  B10011,
  B11000,
  B01111
}; // Char début 1 / 1

byte DIV_0_OF_2[8] = {
  B11111,
  B00000,
```

```
    B00000,
    B00000,
    B00000,
    B00000,
    B00000,
    B11111
  }; // Char milieu 0 / 2

  byte DIV_1_OF_2[8] = {
    B11111,
    B00000,
    B11000,
    B11000,
    B11000,
    B11000,
    B00000,
    B11111
  }; // Char milieu 1 / 2

  byte DIV_2_OF_2[8] = {
    B11111,
    B00000,
    B11011,
    B11011,
    B11011,
    B11011,
    B00000,
    B11111
  }; // Char milieu 2 / 2

  byte END_DIV_0_OF_1[8] = {
    B11110,
    B00011,
    B00001,
    B00001,
    B00001,
    B00001,
    B00011,
    B11110
  }; // Char fin 0 / 1

  byte END_DIV_1_OF_1[8] = {
    B11110,
    B00011,
    B11001,
    B11101,
    B11101,
    B11001,
    B00011,
    B11110
  }; // Char fin 1 / 1


  /**
   * Fonction de configuration de l'écran LCD pour la barre de progression.
   * Utilise les caractères personnalisés de 0 à 6 (7 reste disponible).
   */
  void setup_progressbar() {

    /* Enregistre les caractères personnalisés dans la mémoire de l'écran LCD */
    lcd.createChar(0, START_DIV_0_OF_1);
    lcd.createChar(1, START_DIV_1_OF_1);
    lcd.createChar(2, DIV_0_OF_2);
    lcd.createChar(3, DIV_1_OF_2);
    lcd.createChar(4, DIV_2_OF_2);
    lcd.createChar(5, END_DIV_0_OF_1);
    lcd.createChar(6, END_DIV_1_OF_1);
  }
```

La fonction `setup_progressbar()` qui permet d'initialiser les caractères personnalisés a été très légèrement modifiée pour y inclure le septième cara personnalisé nécessaire à notre barre de progression.

Le plus gros des modifications dans cette partie du code se situe au niveau des caractères personnalisés eux même.

```
/**
 * Fonction dessinant la barre de progression.
 *
 * @param percent Le pourcentage à afficher.
 */
void draw_progressbar(byte percent) {

  /* Affiche la nouvelle valeur sous forme numérique sur la première ligne */
  lcd.setCursor(0, 0);
  lcd.print(percent);
  lcd.print(F(" %  "));
  // N.B. Les deux espaces en fin de ligne permettent d'effacer les chiffres du pourcentage
  // précédent quand on passe d'une valeur à deux ou trois chiffres à une valeur à deux ou un chiffres.

  /* Déplace le curseur sur la seconde ligne */
  lcd.setCursor(0, 1);

  /* Map la plage (0 ~ 100) vers la plage (0 ~ LCD_NB_COLUMNS * 2 - 2) */
  byte nb_columns = map(percent, 0, 100, 0, LCD_NB_COLUMNS * 2 - 2);
  // Chaque caractère affiche 2 barres verticales, mais le premier et dernier caractère n'en affiche qu'une.

  /* Dessine chaque caractère de la ligne */
  for (byte i = 0; i < LCD_NB_COLUMNS; ++i) {

    if (i == 0) { // Premiére case

      /* Affiche le char de début en fonction du nombre de colonnes */
      if (nb_columns > 0) {
        lcd.write(1); // Char début 1 / 1
        nb_columns -= 1;

      } else {
        lcd.write((byte) 0); // Char début 0 / 1
      }

    } else if (i == LCD_NB_COLUMNS -1) { // Derniére case

      /* Affiche le char de fin en fonction du nombre de colonnes */
      if (nb_columns > 0) {
        lcd.write(6); // Char fin 1 / 1

      } else {
        lcd.write(5); // Char fin 0 / 1
      }

    } else { // Autres cases

      /* Affiche le char adéquat en fonction du nombre de colonnes */
      if (nb_columns >= 2) {
        lcd.write(4); // Char div 2 / 2
        nb_columns -= 2;

      } else if (nb_columns == 1) {
        lcd.write(3); // Char div 1 / 2
        nb_columns -= 1;

      } else {
        lcd.write(2); // Char div 0 / 2
      }
    }
  }
}
```

La fonction `draw_progressbar()` qui permet d'afficher le pourcentage sous forme numérique sur la première ligne et la barre de progression sur la deu ligne a été elle aussi modifiée.

Tout d'abord, le produit en croix réalisait par la fonction `map()` a été revu. Maintenant, le calcul part du principe que chaque caractère contient déso jusqu'à deux colonnes au lieu de cinq. De plus, il faut soustraire deux colonnes au résultat de la multiplication, car le premier et dernier caractère de la ba progression ne peut contenir qu'une seule colonne.

Ensuite, la boucle `for` à la ligne 23 qui s'occupe de l'affichage des caractères de la seconde ligne a été entièrement revue.

Pour chaque caractère, un test complètement différent de la première version est réalisé pour déterminer le caractère à afficher.

- Si la case en cours d'affichage est la première case et que le nombre de colonnes à afficher est nul : le code affiche la case de début vide.
- Si la case en cours d'affichage est la première case et que le nombre de colonnes à afficher est supérieur à zéro : le code affiche la case de début ple décrémente le nombre de colonnes à afficher de 1.
- Si la case en cours d'affichage est la dernière case et que le nombre de colonnes à afficher est nul : le code affiche la case de fin vide.

- Si la case en cours d'affichage est la dernière case et que le nombre de colonnes à afficher est supérieur à zéro : le code affiche la case de fin pleine.
- Si la case en cours d'affichage est au milieu de la barre de progression et que le nombre de colonnes à afficher est supérieur ou égal à 2 : le code affich case pleine et décrémente le nombre de colonnes à afficher de 2.
- Si la case en cours d'affichage est au milieu de la barre de progression et que le nombre de colonnes à afficher est égal à 1 : le code affiche une demi pleine et décrémente le nombre de colonnes à afficher de 1.
- Si la case en cours d'affichage est au milieu de la barre de progression et que le nombre de colonnes à afficher est nul : le code affiche une case vide.

Le reste du code de test est inchangé.

N.B. Le cast en byte à la ligne 33 est toujours là pour faire plaisir au compilateur.

L'extrait de code complet est disponible sur cette page (https://www.carnetdumaker.net/snippets/2/) (le lien de téléchargement en .zip contient le p Arduino prêt à l'emploi).

Résultat

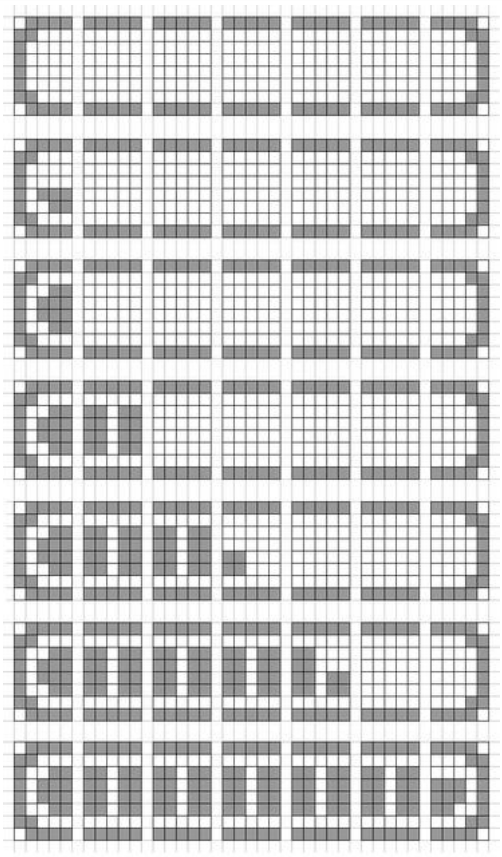
Barre de progression V3

Nous avons désormais une très jolie barre de progression, mais il y a un souci. La barre de progression V2 n'est pas très précise.

Avec un écran de 2 x 16 caractères, la barre de progression V2 a une granularité de ~3,3% (30 colonnes pour 100%). Même avec un écran de 4 x 20 caractèr barre de progression V2 a une granularité de 2,6% (38 colonnes pour 100%).

Une granularité de 3% suffit dans une grande majorité de cas. Mais imaginons que notre action est très longue et qu'il faut une seconde pour avancer de 1%. la barre de progression V2, il faudrait attendre 3 longues secondes avant d'avoir une mise à jour de l'affichage. C'est beaucoup trop long.

Dans cette troisième version, nous allons concevoir l'ultime barre de progression. Cette nouvelle barre de progression aura une résolution optimum et un a visuel plaisant.

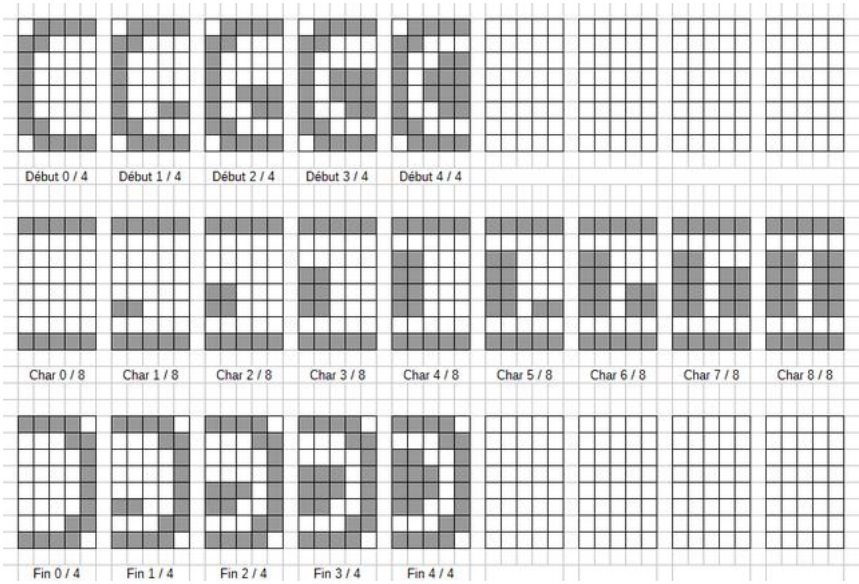


(https://www.carnetdumaker.net/images/esquisse-dune-barre-de-progression-stylisee-avec-divisions-verticales/)

Esquisse d'une barre de progression stylisée avec divisions verticales

L'idée est simple : nous allons garder l'aspect de la barre de progression V2, en divisant tout simplement chaque colonne en sous-unités.

Une colonne fait 4 pixels de haut, on va donc diviser chaque colonne en 4 sous-unités, ce qui va de fait multiplier par 4 la résolution de notre bar progression.

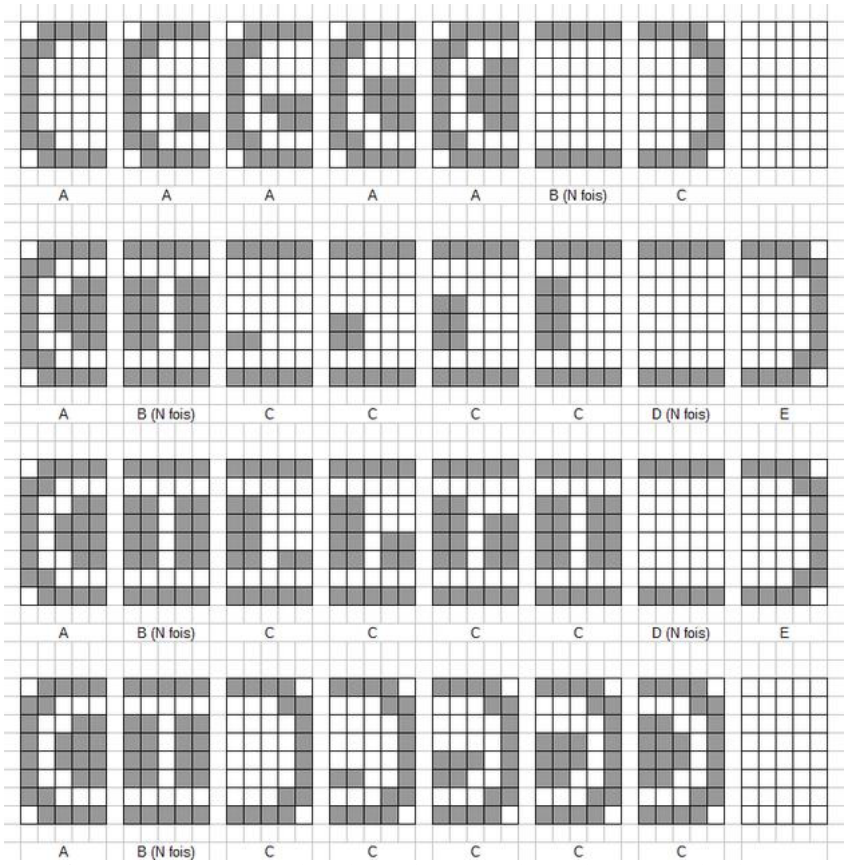


(https://www.carnetdumaker.net/images/caracteres-personnalises-dune-barre-de-progression-stylisees-avec-divisions-verticales/)

Caractères personnalisés nécessaire à la réalisation d'une barre de progression stylisées, avec bordures et divisions verticales

Il y a cependant un gros problème avec cette idée. Il n'est possible d'avoir que 8 caractères personnalisés, pas plus. Or, pour réaliser notre barre de progre v3, il faudrait 5 caractères personnalisés pour le côté gauche, 5 autres pour le côté droit et 9 pour les colonnes au milieu de la barre de progression. Au to faudrait 19 caractères personnalisés, soit 11 de trop.

Un développeur lambda vous aurait dit que c'est impossible, qu'il n'y a pas assez de caractères personnalisés, qu'il faut se faire une raison, mais comn l'expression "à Maker vaillant rien d'impossible" 😉



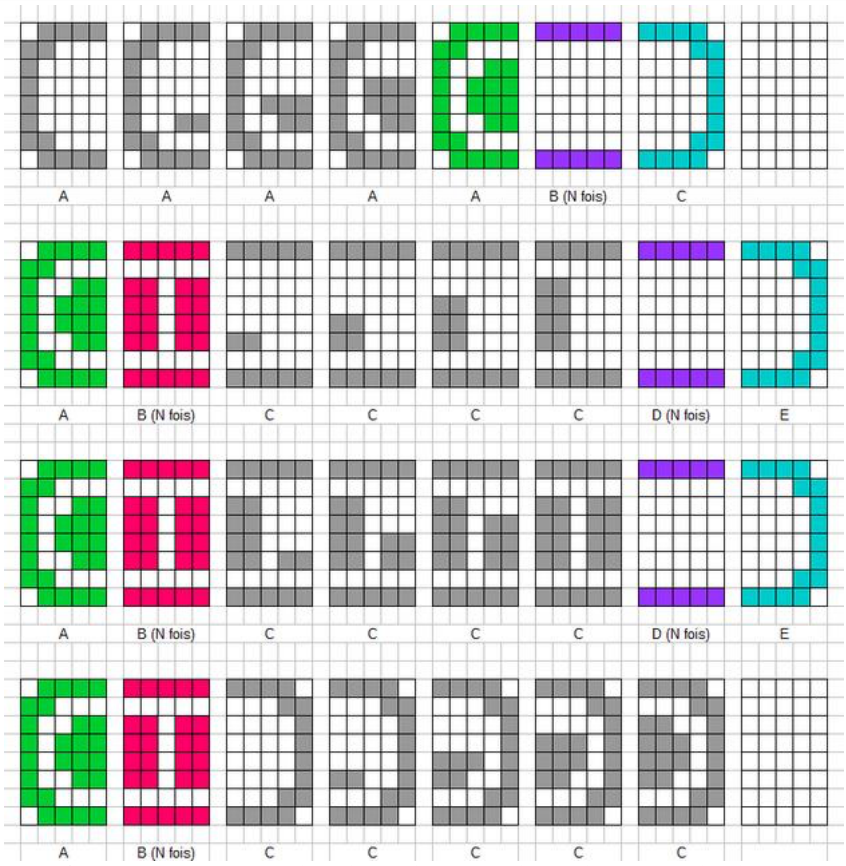
(https://www.carnetdumaker.net/images/illustration-des-etapes-dune-barre-de-progression-stylisees-avec-divisions-verticales/)

Illustration des différentes étapes d'une barre de progression stylisées, avec bordure et divisions verticales

En découpant la progression de la barre de progression en 4 étapes, il devient possible de tout faire passer dans 8 caractères personnalisés !

Les étapes sont les suivantes :

- Étape 1 : on remplit uniquement le premier caractère.
- Étape 2 : on remplit la première colonne d'un caractère au milieu de la barre de progression.
- Étape 3 : on remplit la deuxième colonne d'un caractère au milieu de la barre de progression.
- Étape 4 : on remplit uniquement le dernier caractère.



(https://www.carnetdumaker.net/images/illustration-des-etapes-dune-barre-de-progression-stylisees-avec-divisions-verticales-avec-couleurs/)

Les caractères communs à au moins trois des quatre étapes sont en couleurs

Et comme on est des Makers malins sur ce site, on remarque de suite que 4 caractères sont communs à au moins trois des quatre étapes.

Si on laisse de côté ces 4 caractères communs, il n'y a que 4 caractères à modifier dynamiquement en fonction du pourcentage pour réaliser notre bar progression de la mort qui tue.

```
/* Caractères personnalisés */
byte START_DIV_0_OF_4[8] = {
  B01111,
  B11000,
  B10000,
  B10000,
  B10000,
  B10000,
  B11000,
  B01111
}; // Char début 0 / 4

byte START_DIV_1_OF_4[8] = {
  B01111,
  B11000,
  B10000,
  B10000,
  B10000,
  B10011,
  B11000,
  B01111
}; // Char début 1 / 4

byte START_DIV_2_OF_4[8] = {
  B01111,
  B11000,
  B10000,
  B10000,
  B10111,
  B10011,
  B11000,
  B01111
}; // Char début 2 / 4
```

```
34
35  byte START_DIV_3_OF_4[8] = {
36    B01111,
37    B11000,
38    B10000,
39    B10111,
40    B10111,
41    B10011,
42    B11000,
43    B01111
44  }; // Char début 3 / 4
45
46  byte START_DIV_4_OF_4[8] = {
47    B01111,
48    B11000,
49    B10011,
50    B10111,
51    B10111,
52    B10011,
53    B11000,
54    B01111
55  }; // Char début 4 / 4
56
57  byte DIV_0_OF_8[8] = {
58    B11111,
59    B00000,
60    B00000,
61    B00000,
62    B00000,
63    B00000,
64    B00000,
65    B11111
66  }; // Char milieu 0 / 8
67
68  byte DIV_1_OF_8[8] = {
69    B11111,
70    B00000,
71    B00000,
72    B00000,
73    B00000,
74    B11000,
75    B00000,
76    B11111
77  }; // Char milieu 1 / 8
78
79  byte DIV_2_OF_8[8] = {
80    B11111,
81    B00000,
82    B00000,
83    B00000,
84    B11000,
85    B11000,
86    B00000,
87    B11111
88  }; // Char milieu 2 / 8
89
90  byte DIV_3_OF_8[8] = {
91    B11111,
92    B00000,
93    B00000,
94    B11000,
95    B11000,
96    B11000,
97    B00000,
98    B11111
99  }; // Char milieu 3 / 8
100
101 byte DIV_4_OF_8[8] = {
102   B11111,
103   B00000,
104   B11000,
105   B11000,
106   B11000,
107   B11000,
108   B00000,
109   B11111
110 }; // Char milieu 4 / 8
111
112 byte DIV_5_OF_8[8] = {
113   B11111,
114   B00000,
```

```
    B11000,
    B11000,
    B11000,
    B11011,
    B00000,
    B11111
  }; // Char milieu 5 / 8

  byte DIV_6_OF_8[8] = {
    B11111,
    B00000,
    B11000,
    B11000,
    B11011,
    B11011,
    B00000,
    B11111
  }; // Char milieu 6 / 8

  byte DIV_7_OF_8[8] = {
    B11111,
    B00000,
    B11000,
    B11011,
    B11011,
    B11011,
    B00000,
    B11111
  }; // Char milieu 7 / 8

  byte DIV_8_OF_8[8] = {
    B11111,
    B00000,
    B11011,
    B11011,
    B11011,
    B11011,
    B00000,
    B11111
  }; // Char milieu 8 / 8

  byte END_DIV_0_OF_4[8] = {
    B11110,
    B00011,
    B00001,
    B00001,
    B00001,
    B00001,
    B00011,
    B11110
  }; // Char fin 0 / 4

  byte END_DIV_1_OF_4[8] = {
    B11110,
    B00011,
    B00001,
    B00001,
    B00001,
    B11001,
    B00011,
    B11110
  }; // Char fin 1 / 4

  byte END_DIV_2_OF_4[8] = {
    B11110,
    B00011,
    B00001,
    B00001,
    B11101,
    B11001,
    B00011,
    B11110
  }; // Char fin 2 / 4

  byte END_DIV_3_OF_4[8] = {
    B11110,
    B00011,
    B00001,
    B11101,
    B11101,
    B11001,
```

```
  B00011,
  B11110
}; // Char fin 3 / 4

byte END_DIV_4_OF_4[8] = {
  B11110,
  B00011,
  B11001,
  B11101,
  B11101,
  B11001,
  B00011,
  B11110
}; // Char fin 4 / 4


/**
 * Fonction de configuration de l'écran LCD pour la barre de progression.
 * Utilise tous les caractères personnalisés de 0 à 8.
 */
void setup_progressbar() {

  /* Enregistre les caractères personnalisés dans la mémoire de l'écran LCD */
  lcd.createChar(0, START_DIV_4_OF_4);
  lcd.createChar(1, DIV_0_OF_8);
  lcd.createChar(2, DIV_8_OF_8);
  lcd.createChar(3, END_DIV_0_OF_4);
  // Les autres caractères sont configurés dynamiquement
}
```

On va donc reprendre notre code V2 et modifier les caractères personnalisés pour avoir tout le nécessaire à notre barre de progression.

On va ensuite modifier la fonction `setup_progressbar()` pour qu'elle ne configure au démarrage que les 4 caractères communs.

```
/**
 * Fonction de configuration dynamique de l'écran LCD pour la barre de progression.
 *
 * @param bank Le numéro de la banque de caractères à configurer.
 */
void switch_progressbar_bank(byte bank) {

  // IMPORTANT : Il est nécessaire de faire un lcd.clear() ou un lcd.setCursor() après chaque changement de banque.

  /* Change de banque de caractères */
  switch (bank) {
    case 0:
      lcd.createChar(4, START_DIV_0_OF_4);
      lcd.createChar(5, START_DIV_1_OF_4);
      lcd.createChar(6, START_DIV_2_OF_4);
      lcd.createChar(7, START_DIV_3_OF_4);
      break;

    case 1:
      lcd.createChar(4, DIV_1_OF_8);
      lcd.createChar(5, DIV_2_OF_8);
      lcd.createChar(6, DIV_3_OF_8);
      lcd.createChar(7, DIV_4_OF_8);
      break;

    case 2:
      lcd.createChar(4, DIV_4_OF_8);
      lcd.createChar(5, DIV_5_OF_8);
      lcd.createChar(6, DIV_6_OF_8);
      lcd.createChar(7, DIV_7_OF_8);
      break;

    case 3:
      lcd.createChar(4, END_DIV_1_OF_4);
      lcd.createChar(5, END_DIV_2_OF_4);
      lcd.createChar(6, END_DIV_3_OF_4);
      lcd.createChar(7, END_DIV_4_OF_4);
      break;
  }
}
```

On va ensuite déclarer une nouvelle fonction nommée `switch_progressbar_bank()`, cette fonction va nous permettre de changer de banque de caractèr fonction de la progression de notre barre de progression.

Cette fonction prend en paramètre le numéro de l'étape de progression en cours d'affichage.

⚠ Attention aux créations dynamiques de caractères personnalisés

Si par la suite, vous réutilisez la fonction `createChar()` de la bibliothèque LiquidCrystal dans un de vos codes. Sachez qu'il faut impérativement faire `clear()` ou un `setCursor()` après avoir fait appel à `createChar()` avant d'afficher un caractère.

L'exemple `CustomChar` fourni avec l'environnement de développement Arduino est un bon exemple de ce bug / problème. Si vous lancez cet exemple quel, il ne fonctionnera pas. Si vous ajoutez un `lcd.clear();` juste après la série de `createChar()`, il fonctionnera.

Ce bug m'a fait perdre plus de trois heures lors de la rédaction de cet article. Ne vous faites pas avoir par ce bug relou 😉

```
/**
 * Fonction dessinant la barre de progression.
 *
 * @param percent Le pourcentage à afficher.
 */
void draw_progressbar(byte percent) {

  /* Affiche la nouvelle valeur sous forme numérique sur la première ligne */
  lcd.setCursor(0, 0);
  lcd.print(percent);
  lcd.print(F(" %  "));
  // N.B. Les deux espaces en fin de ligne permettent d'effacer les chiffres du pourcentage
  // précédent quand on passe d'une valeur à deux ou trois chiffres à une valeur à deux ou un chiffres.

  /* Déplace le curseur sur la seconde ligne */
  lcd.setCursor(0, 1);

  /* Map la plage (0 ~ 100) vers la plage (0 ~ LCD_NB_COLUMNS * 2 * 4 - 2 * 4) */
  byte nb_columns = map(percent, 0, 100, 0, LCD_NB_COLUMNS * 2 * 4 - 2 * 4);
  // Chaque caractère affiche 2 barres verticales de 4 pixels de haut, mais le premier et dernier caractère n'en af
e.

  /* Dessine chaque caractère de la ligne */
  for (byte i = 0; i < LCD_NB_COLUMNS; ++i) {

    if (i == 0) { // Premiére case

      /* Affiche le char de début en fonction du nombre de colonnes */
      if (nb_columns > 4) {
        lcd.write((byte) 0); // Char début 4 / 4
        nb_columns -= 4;

      } else if (nb_columns == 4) {
        lcd.write((byte) 0); // Char début 4 / 4
        nb_columns = 0;

      } else {
        switch_progressbar_bank(0);
        lcd.setCursor(i, 1);
        lcd.write(nb_columns + 4); // Char début N / 4
        nb_columns = 0;
      }

    } else if (i == LCD_NB_COLUMNS - 1) { // Derniére case

      /* Affiche le char de fin en fonction du nombre de colonnes */
      if (nb_columns > 0) {
        switch_progressbar_bank(3);
        lcd.setCursor(i, 1);
        lcd.write(nb_columns + 3); // Char fin N / 4

      } else {
        lcd.write(3); // Char fin 0 / 4
      }

    } else { // Autres cases

      /* Affiche le char adéquat en fonction du nombre de colonnes */
      if (nb_columns == 0) {
        lcd.write(1); // Char div 0 / 8

      } else if (nb_columns >= 8) {
        lcd.write(2); // Char div 8 / 8
        nb_columns -= 8;

      } else if (nb_columns >= 4 && nb_columns < 8) {
        switch_progressbar_bank(2);
        lcd.setCursor(i, 1);
        lcd.write(nb_columns); // Char div N / 8
        nb_columns = 0;

      } else if (nb_columns < 4) {
        switch_progressbar_bank(1);
        lcd.setCursor(i, 1);
        lcd.write(nb_columns + 3); // Char div N / 8
        nb_columns = 0;
      }
    }
  }
}
```

Le code de la fonction `draw_progressbar()` devient un peu technique. Expliquer chaque test en détail serait beaucoup trop long et très lassant à lire. Je laisse donc lire les commentaires du code et le code lui-même pour en comprendre la logique. Il n'y a rien de compliqué, vraiment.

Le reste du code de test est inchangé.

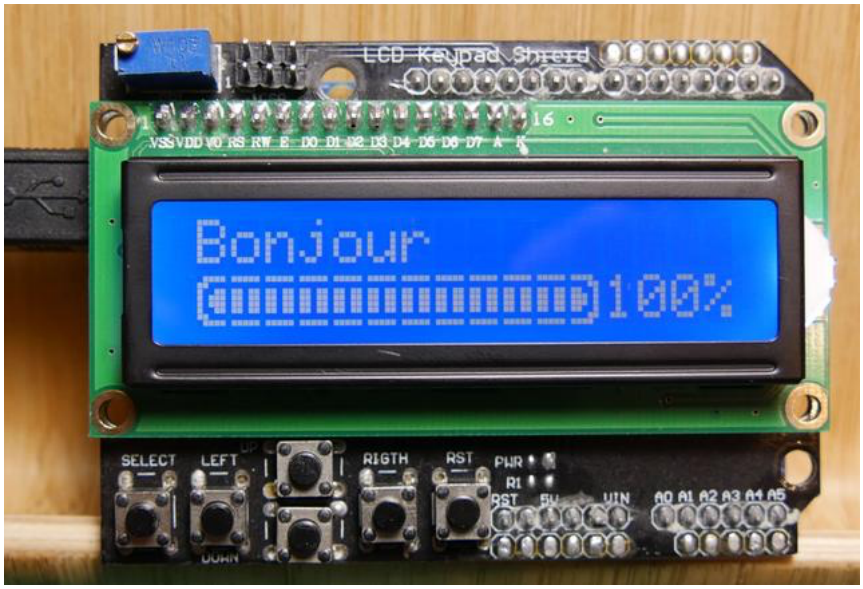
L'extrait de code complet est disponible sur cette page (https://www.carnetdumaker.net/snippets/3/) (le lien de téléchargement en .zip contient le p Arduino prêt à l'emploi).

Résultat

Bonus : la barre de progression V3 sur une seule ligne

Un article du site Carnet du Maker ne serait pas complet sans un bonus 😉

Les lecteurs attentifs auront remarqué que 16 caractères x 2 colonnes par caractère x 4 sous-unités par colonnes - 2 colonnes de 4 sous-unités = 120 sous-u La barre de progression est donc plus précise que le pourcentage de départ.



(https://www.carnetdumaker.net/images/photo-barre-progression-stylisees-avec-divisions-verticales-variante-sur-une-seule-ligne/)

Le résultat

On peut donc profiter de cette précision pour réduire la taille de la barre de progression et mettre le pourcentage sur la même ligne que la barre de progressi

L'extrait de code complet de cette variante, prêt à l'emploi, est disponible sur cette page (https://www.carnetdumaker.net/snippets/4/) (le lien de télécharge en .zip contient le projet Arduino prêt à l'emploi).

Conclusion

Ce tutoriel est désormais terminé.

Si ce tutoriel vous a plu, n'hésitez pas à le commenter sur le forum, à le diffuser sur les réseaux sociaux et à soutenir le site si cela vous fait plaisir.

Articles pouvant vous intéresser

- Utiliser une shield LCD de DFRobots avec une carte Arduino / Genuino et la bibliothèque LiquidCrystal (/articles/utiliser-une-shield-lcd-de-dfrobots-avec-une-carte-arduino-genuino-et-la-bibliotheque-liquidcrystal/)

Cliquez ici pour accéder aux commentaires de l'article. (/forum/topics/12-faire-une-barre-de-progression-avec-arduino-et-liquidcrystal/)