Année 2013-2014

Rapport de projet EI4 AGI

Ecran LEDs RVB

Projet réalisé par :

Maxime Pontreau
Cyril Le Bras
Mohamed Hammami

Projet encadré par :

Sébastien Lagrange

ISTIA – Université d'Angers

Remerciements

Nous tenons en premier lieu à remercier Monsieur Lagrange, qui a eu l'idée de lancer ce projet, nous a aidé et guidé durant son avancement.

Nous remercions également Monsieur Hassan qui nous a permis d'assembler l'écran de LEDs et prodigué de précieux conseils pour l'alimentation de celui-ci. Nos remerciements s'adressent également à Monsieur Cottenceau à qui nous avons emprunté du matériel nécessaire à la poursuite de ce projet.

Sans oublier de remercier les autres professeurs qui ont apportés leur aide et leurs conseils.

Enfin nous remercions l'ISTIA et l'Université d'Angers qui chaque année accepte de financer de tels projets pour nous permettre d'acquérir de l'expérience et d'étoffer notre CV.

Table des matières

Contenu

Introduction

L'objectif et les résultats attendus

L'objectif de notre projet, était de pouvoir afficher un texte défilant sur un écran à LEDs, texte préalablement entré par l'utilisateur, ou encore de proposer diverses animations comme de la pluie, un feu d'artifice etc… Enfin nous devions créer un site internet qui servirait d'interface à l'utilisateur souhaitant contrôler le panneau pour que celui-ci soit totalement indépendant.

Le Matériel

Au commencement de ce projet, nous avions à disposition une bande de 32 LEDs reliée à une carte Arduino 32u4, cette dernière nous a permis de découvrir la bibliothèque fournie établissant la connexion avec les LEDs. Par la suite, deux rouleaux de bandes supplémentaires de 5m on étés fournis, soit 10 bandes de 32 LEDs permettant d'afficher 2 lignes de texte de hauteur. Les cartes Arduino ont également évoluées avec l'augmentation du nombre de bandes, changements nécessaire à l'acquisition d'une mémoire suffisante pour la programmation intégrale du panneau.

L'organisation

Durant ce projet, divers problèmes ce sont posés à nous, ceci conduisant à l'établissement de tâches personnelles respectives. En effet lorsque nous rencontrions un problème bloquant la programmation, nous laissions en fonction de ce dernier, une ou plusieurs personnes rechercher une solution. Notamment lors de l'optimisation de la mémoire dont nous parlerons plus tard, tâche chronophage qui a grandement mobilisée l'équipe.

Lancement du projet

Introduction

Pour la présentation de ce projet, nous avons décidé de décrire chacune des parties le composant, ainsi que les débuts de ce dernier comme ci-dessous.

Prise en main du matériel

Commençons tout d'abord par la prise en main du matériel. Bien heureusement nous connaissons, grâce au cours reçus, le logiciel Arduino, cependant il a tout de même fallu un temps d'adaptation. Puis, il était nécessaire de tester le programme fourni par le distributeur qui permet d'afficher certaines animations possibles. Ce programme d'exemple utilise la librairie LPD8806 qui était, elle aussi, fournie. Il était donc primordial de comprendre comment fonctionnait cette librairie et de quelle manière pouvait-on l'utiliser.

La librairie LPD8806

Voici quelques exemples de fonctions de cette libraire :

LPD8806 nom_objet = LPD8806 (uint16_t n);

Ceci est le constructeur de l'objet. Le **n** représente un entier de 16 bits qui définit le nombre de LEDs présente sur le panneau.

Void begin (void) ;

Cette fonction permet de charger les éléments nécessaires pour le fonctionnement des LEDs.

Void setPixelColor (uint16_t n, uint8_t r, uint8_t g, uint8_t b) ;
Void setPixelColor (uint16_t n, uint32_t c) ;

Ces deux fonctions ont le même but, enregistrer une LED et la couleur de celle-ci. Le **n** représente le numéro de la LED à allumer. Pour la première fonction les variables **r**, **g** et **b** représentent les entiers correspondant aux couleurs rouge

vert et bleu qui permettent de définir une couleur voulue. Pour la deuxième fonction la variable **c** contient les informations de ces 3 couleurs.

Void show (void) ;
Cette fonction permet d'allumer les LEDs avec les paramètres enregistrés grâce aux fonctions **setPixelColor** vues précédemment.

D'autres fonctions sont encore disponible cependant nous n'en avons pas eu l'utilité lors de notre projet.

Préparation et envoi des données

Préparation avant la programmation

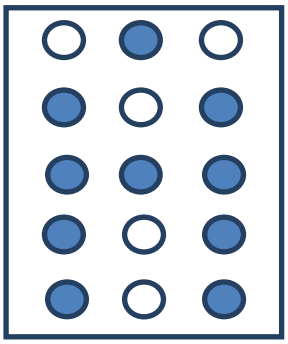
Après plusieurs heures passées à installer le Driver nous réussissions à charger le programme correctement, il était désormais possible de commencer à programmer par nous-même. Mais avant toute chose il fallait créer les tableaux représentants les lettres et également commencer à agrandir l'écran LEDs car il est difficile de faire défiler du texte avec une seule bande de 32 LEDs. Une répartition des tâches était donc nécessaire :

- L'un d'entre nous s'est occupé de couper le rouleau de 5m en 5 bandes de 1m (1m = 32 LEDs) et les a assemblées entre elles.

- Le second a entrepris de dessiner les lettres en tableau de LEDs à deux dimensions. Nous avons pris la décision de faire des lettres de 5 LEDs hauteur en minimisant la largeur de celles-ci.

- Enfin le 3ème a débuté la programmation.

Pour concevoir correctement les lettres, nous les avons toutes dessinées de la façon ci-contre (Exemple de la lettre « A ») pour nous apercevoir au mieux du rendu. Une fois toutes les lettres dessinées, il ne restait plus qu'à les rentrer dans un tableau à 3 dimensions.



Lettre A en LED

La première dimension de ce tableau représente le nombre de caractères différents présents. Nous disposons de l'alphabet, des chiffres et également quelques caractères spéciaux comme le point, l'apostrophe etc…

La deuxième dimension représente la largeur maximum d'une lettre, lorsque la largeur de la lettre est moindre il suffit de remplir les colonnes restantes par des 0.

Enfin la troisième dimension représente la hauteur d'une lettre. Il suffit donc de déclarer la lettre A de cette manière ci :

Lettre[59][4][5] = {{{0,1,1,1,1},{1,0,1,0,0},{0,1,1,1,1},{0,0,0,0,0}},…(autres caractères)}

Première colonne

LED 1ère ligne, 2ème colonne

La création d'un tableau **Taille** était essentielle pour qu'il soit possible de récupérer le nombre de colonne qu'occupait un caractère.

Envoi des données

Pour simplifier l'échange de données entre le serveur WEB et la carte Arduino, nous avons conçus une trame d'envoi afin de préciser les différents paramètres tel que le code de l'application lancée (phrase ou image) ou encore la couleur souhaitée.

Trame pour une phrase

{Code, Vitesse défilement, Nombre de tours, Couleur, Phrase à afficher}

- Code : 0 pour une phrase à faire défiler
- Vitesse défilement : peut prendre les valeurs 0,1 ou 2 pour lent, modérée ou rapide
- Nombre de tours : défini entre 1 et 6, ou 9 si l'on souhaite un défilement illimité (qui sera stoppé dès qu'une nouvelle fonction est appelée)
- Couleur : la couleur est définie par les trois composantes RVB (rouge, vert, bleu) comme ceci : {127, 127, 127}, ce qui permet un choix d'environ 2 millions de couleurs différentes

Ainsi, si l'on entre la phrase « j'aime le chocolat ! », et que l'on souhaite qu'elle défile à vitesse moyenne, en bleu et durant 4 tours, la trame sera la suivante :

{0140000000127j'aime le chocolat !}

Trame pour une image

Il y a ici deux types de trames différentes, ce qui est due à la limitation du buffer de la carte Arduino, qui est rapidement plein. Pour cela nous utilisons deux codes différents : le 1 pour annoncer une nouvelle image (ce qui a pour effet d'effacer entièrement le panneau) et le 2 qui vient compléter le panneau déjà existant. Ainsi, pour chaque trame d'envoi nous pouvons commander l'allumage de 32 LEDs différentes et de couleurs indépendantes.

{Code, Numéro ligne, Numéro colonne, Couleur,…}

32 fois maximum

- Code : 1 si c'est une nouvelle image, 2 si c'est la continuité d'une image
- Numéro ligne : indique le numéro de ligne de la LED à allumer
- Numéro colonne : indique le numéro de colonne
- Couleur : comme pour la phrase (Rouge, vert, bleu)

Ainsi, pour faire apparaitre le simple dessin ci-contre de l'écran, la trame sera la suivante :

{100001270000000001000000127000200012700 0}

1ère LED 2ème LED 3ème LED

Structure du programme

La fonction principale : loop()

Cette fonction est la partie centrale du programme et elle s'exécute en boucle sur le microcontrôleur tant que celui-ci est alimenté, cela permet ainsi d'avoir une utilisation en continue de l'application sans avoir à recharger le programme.

Elle se décompose en plusieurs parties, que nous détaillerons par la suite, de la manière suivante :

- Lecture du premier caractère reçu, le code 0,1 ou 2 (**Serial.**read())
- On redirige vers l'application choisie (switch(…))
- Lecture des différents paramètres (**Serial.**read())
- Affichage sur l'écran LED

Lecture d'un caractère

Nous procédons de la manière suivante :

```
//Lecture du code d'application
Reception=Serial.read();
Reception=(int)DetecteLettre(Reception)-42;
```

Lorsque la carte Arduino reçoit les données envoyées en série à l'aide de la fonction **Serial.**read(), celle-ci renvoie le code Ascii du caractère lu. Notre fonction **DetecteLettre** transforme l'entier **ascii** qui lui est donné pour renvoyer un entier qui correspond à la position de la lettre dans notre tableau à 3 dimensions. Par exemple : La lettre « A » à pour code Ascii 65 et est en position 0 de notre tableau **Lettre.** Donc **DetecteLettre(65) = 0**

Cette fonction est utilisée pour la lecture de tous les paramètres et tous les caractères reçus. Une fois le code reconnu, un des différents cas suivant est lancé via le switch(…).

Le cas défilement d'une phrase : code 0

Une fois dans cette partie du code nous initialisons les différentes variables :

```
//Cas défilement d'une phrase
case 0:
    delay(1000);
    //Eteint toutes les leds
    for(i=0;i<strip.numPixels(); i++){
        strip.setPixelColor(i, 0);
      }
    //Initialise le tableau contenant la phrase
    for(i=0;i<TaillePanneau;i++)
    {
      for(j=0;j<5;j++)
      {
        Panneau_LED[i]=false;
      }
    }
    decalage=0;
```

Le **Delay(1000)** est positionné ici car il permet de faire une pause d'une seconde, le temps que la transmission de trame soit complète, une fois la trame reçu le premier **for()** initialise l'écran en éteignant toutes les LEDs. Le second **for()** lui initialise la variable **Panneau_LED** qui contenait la dernière phrase envoyée pour ne pas avoir de conflits entre plusieurs utilisations successives. Enfin la variable **decalage** est remise à zéro, c'est elle qui gère l'avancement de la phrase sur l'écran.

Une fois ces étapes exécutées, le programme va lire les différents paramètres d'entrée de la trame :

- La vitesse de défilement
- Le nombre de tours à effectuer
- La couleur d'affichage

```
//Detection vitesse
vitesse=Detection_vitesse();

//Detection nombre de tours
Reception=Serial.read();
Reception=(int)DetecteLettre(Reception)-42;
tour=Reception;

//Detection couleur
red=Detection_couleurs();
green=Detection_couleurs();
blue=Detection_couleurs();
```

Les fonctions **Detection_vitesse()** ainsi que **Detection_couleurs()** ici présentes ont les mêmes fonctionnalités que la première partie (voir section : Lecture d'un caractère). Elles permettent donc d'obtenir les valeurs chiffrées des paramètres entrés par l'utilisateur.

Il reste maintenant la partie sur le remplissage du tableau **Panneau_LED** ainsi que l'affichage sur l'écran.

```
//Remplissage du tableau
while (Serial.available()>0)
{
  Reception=Serial.read();
  NumLettre = DetecteLettre(Reception);
  FEEPROM((int)pgm_read_byte_near(&(Taille[NumLettre])),NumLettre);
}

//Affichage de la phrase avec les différents paramètres
Defilement(tour,vitesse,decalage);
```

Remplissage du tableau et affichage

Dans cette partie, chaque caractère est placé dans le tableau via la fonction **FEEPROM()** jusqu'à ce qu'il n'y ait plus aucun caractère dans le Buffer qui est une zone de mémoire qui stock les trames reçues temporairement. Puis la fonction **Defilement** est appelée.

```
void Defilement(int NbTour,int vitesse, int decalage)
{
  int NbLignes = 2, imax=0, k;
  byte i,j,Nb=0;
  while(Nb<NbTour){
    //ma fonction
    for(k=(-NbLignes*32);k<decalage;k++){
        //en cas de nouvelle phrase rentrée
        if(Serial.available()>0){return;}
        //cas classique
        for(i=0;i<32;i++){
            for(j=0;j<NbLignes;j++){
                if((i+j*32+k)>=0 && (i+j*32+k)<=decalage){
                    LireColonne(i,j,Panneau_LED[i+j*32+k]);
                }else{  LireColonne(i,j,0);}
            }
        }
        strip.show();
        delay(vitesse);
        }
        if(NbTour!=9){ Nb++;}
    }
}
```

Fonction FEEPROM

Cette fonction est celle qui va charger le tableau **Panneau_LED** avec les bons caractères, pour cela elle doit aller chercher la représentation de tous ces caractères dans la mémoire EEPROM, là où ils sont stockés, via la fonction **pgm_read_byte_near**. Cette dernière renvoie l'élément du tableau **Lettre** placé à la position ciblée.

Fonction Defilement

Cette Fonction est probablement la plus importante de cette partie car c'est elle qui permet le mouvement de l'affichage avec les différents paramètres entrés par l'utilisateur. A chaque tour de cycle des fonctions **for()**, l'affichage est actualisé pour permettre le décalage en appelant la fonction **LireColonne** qui détermine les différentes LEDs à allumer ou non.

Le cas affichage d'une image : code 1 et 2

Cette partie est divisée en deux parties, selon si l'on souhaite afficher une nouvelle image ou continuer d'en dessiner une. Mais ces deux parties sont sensiblement identiques, seule l'initialisation diffère car dans le premier cas nous réinitialisons l'écran en éteignant toutes les LEDs. Nous ne traiterons donc ici que la partie d'une nouvelle image.

```
//Cas nouvelle image
case 1:
      //Eteint toutes les leds
     for(i=0;i<strip.numPixels(); i++){
       strip.setPixelColor(i, 0);
     }
      do{
        //Initialisation
         ligne=0;
         colonne=0;
        //Detection de la led ciblée
         ligne=Detecte_ligne_colonne();
         colonne=Detecte_ligne_colonne();

         //Detection couleur
         red=Detection_couleurs();
         green=Detection_couleurs();
         blue=Detection_couleurs();

         //affichage
         strip.setPixelColor(MatriceALED(colonne,ligne),strip.Color(red,green,blue));
         strip.show();
     }while(Serial.available()>0);
     break;
```

On voit ici que l'on fait appel aux mêmes fonctions que précédemment mais contrairement à l'affichage d'une phrase, la boucle principale est appelée une fois pour chaque LED, ainsi on peut définir une couleur différente à chaque fois, ce qui permettrai d'afficher une image pixélisée grossièrement dans le cas d'une poursuite de ce projet.

Le Site web

Connexion Port Série

La première chose à tester avant toute chose était l'envoi de données sur le port série avec le langage PHP. À l'aide de recherches internet nous avons trouvé une bibliothèque se nommant **PhpSerial** et permettant d'envoyer les données sur le port série et donc de remplacer le moniteur d'Arduino. Voici comment elle s'utilise:

require("src/PhpSerial.php"); **//** Indique le répertoire de la librairie PhpSerial

$serial = new phpSerial(); **//** *Premièrement, il faut créer l'objet serial à l'aide du constructeur sans argument de la librairie PhpSerial.*

$serial->deviceSet("COM6"); **//** *Ensuite on indique sur quel COM est connectée la carte Arduino avec la méthode* ***deviceSet****.*

$serial->confBaudRate(115200); **//** *Ceci correspond à la vitesse de communication avec le port série.*

$serial->confParity("none"); **//** *Nous n'avons pas besoin de parité dans notre cas, on l'initialise donc à none*

$serial->confCharacterLength(8); **//** *Défini l'espace mémoire d'un caractère, dans notre cas un caractère est codé sur un octet donc 8 bits.*

$serial-> confStopBits (2); **//** *Définie l'espace mémoire des bits d'arrêt qui signale la fin de la communication*

$serial->confFlowControl("none"); **//** *Le flow contrôle n'est pas nécessaire pour notre communication*

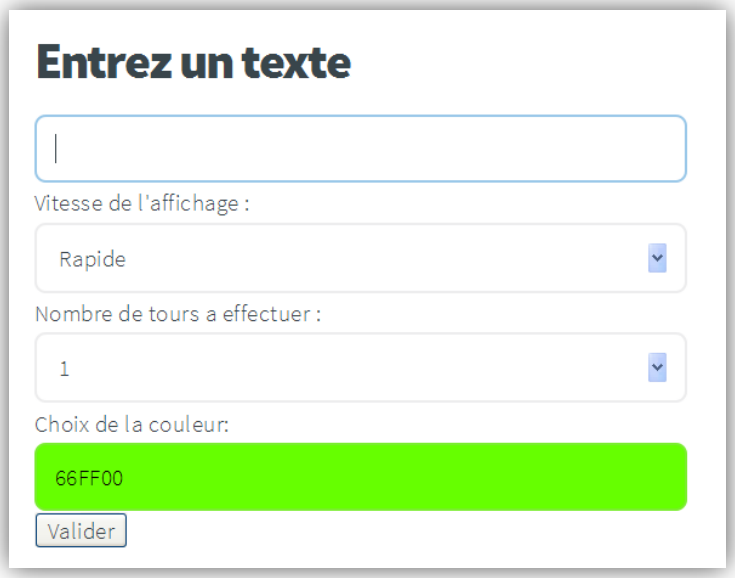
$serial->deviceOpen(); **//** *Ouvre la communication*

$serial->sendMessage($Mess); **//** *Envoie le message contenu dans $Mess*

$serial->deviceClose(); **//** *Ferme la communication*

Interface d'envoi d'une phrase

Voici ci-contre le formulaire servant d'interface permettant l'affichage d'une phrase sur l'écran LED.



Lorsque l'utilisateur clic sur le bouton « Valider » le code PHP récupère toutes les variables : le texte, la vitesse etc…

```
if(isset($_POST["valeur"])) //L'utilisateur clic sur valider
{
    //Récupération des variables
    $valeur = $_POST["valeur"];
    $couleur=$ POST["couleur"];
```

Ensuite un test est nécessaire pour vérifier que l'utilisateur a tapé du texte :

```
if(strlen($valeur)>0){
```

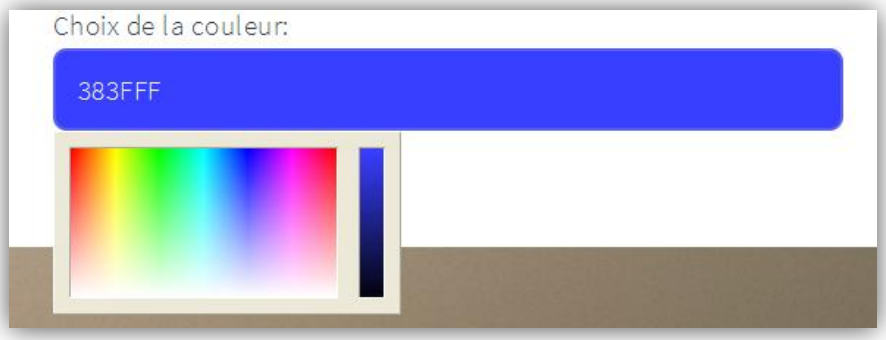
Si l'utilisateur a entré des caractères, les variables sont transformées pour qu'elles respectent la trame à envoyer. On configure le port série (Voir section précédente : Connexion Port Série).

Enfin il suffit d'envoyer les données en respectant la trame pour une phrase (voir section précédente : Trame pour une phrase).

Sinon on affiche un message d'alerte :

```
echo "<script>alert('Veuillez taper un message')</script>";
```

À l'aide d'un code JavaScript libre nous avons inséré une interface permettant de choisir la couleur de la phrase qui doit être affichée par le panneau.



Cette couleur étant en hexadécimal un traitement PHP est nécessaire :

Les 2 premiers caractères du code hexadécimal correspondent à la couleur rouge. Ils sont donc ajoutés à la variable **$rouge.** Le même principe est appliqué pour les couleurs vert et bleu.

```
$rouge = $couleur[0].$couleur[1];
$vert = $couleur[2].$couleur[3];
$bleu = $couleur[4].$couleur[5];
$r=hexdec($rouge);
$v=hexdec($vert);
$b=hexdec($bleu);
$r = intval($r/2);
$v = intval($v/2);
$b= intval($b/2);
$rouge = (string)$r;
$vert = (string)$v;
$bleu = (string)$b;
```

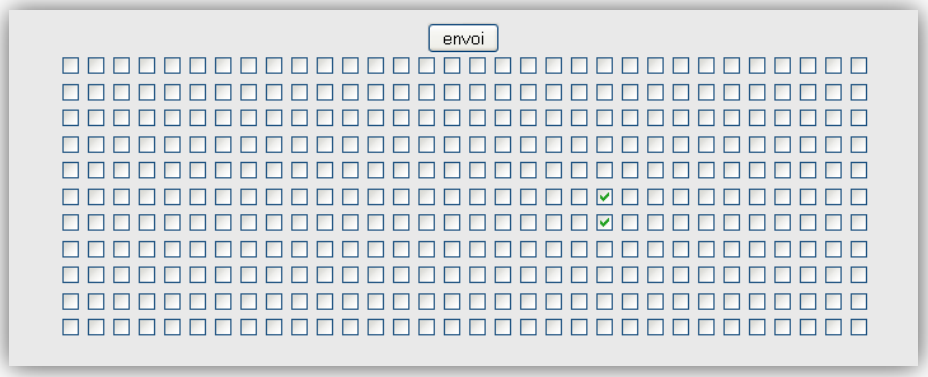
Puis la fonction **hexdec** permet de transformer le code hexadécimal de chaque couleur en entier. De cette manière, si par exemple **$rouge** contient la valeur « 0A » **$r** prendra comme valeur 10.

L'interface JavaScript code les couleurs RGB de 0 à 256. Le problème étant que l'écran LEDs ne dispose que de 128 valeurs différentes pour chacune des couleurs, une division par deux est donc faite que les couleurs affichées par le panneau soit représentatives.

Interface d'envoi d'un dessin

Cette partie n'est malheureusement pas terminée car l'interface WEB est loin d'être optimale. L'idée était de pouvoir facilement dessiner une image à l'aide des LEDs.

Ci-dessous l'interface graphique représentant l'écran LEDs et permettant de dessiner.



En passant sa souris sur une check box celle-ci se sélectionne ou se désélectionne pour permettre à l'utilisateur de dessiner. Les check box appartiennent à un même tableau **« options[] »** ce qui facilite l'envoi des données. On parcourt les check box **qui sont validées** à l'aide d'un **foreach** :

```
foreach($_POST['options'] as $chkbx){
```

$chkbx correspond à un entier représentant le numéro de la check box.

À chaque nouvelle check box, **$i** prend pour valeur le numéro de celle-ci.

Le « **while** » permet d'incrémenter **$j** qui représente le numéro de la ligne.

Les **« if »** ont quant à eux pour but de mettre en forme les Strings **$i** et **$j** pour la trame (voir section Trame pour une image).

```
foreach($_POST['options'] as $chkbx){
    $cpt++;
    $j=0;
    $i = $chkbx;

    while($i > 31)
    {
        $j++;
        $i = $i - 32;
    }
    if(strlen($i) == 1)
    {
        $temp = $i;
        $i = "0" . $temp;
    }
    if(strlen($j) == 1)
    {
        $temp = $j;
        $j = "0" . $temp;
    }
    $message = $message . $j . $i . $rouge . $vert . $bleu;
```

Enfin la variable **$message** concatène toutes les informations de la check box (position et couleur), et contatène également les check box suivantes.

Mais avant d'envoyer cette variable il faut vérifier qu'il reste assez de place dans le buffer :

Si l'on revient sur le code précèdent on remarque qu'une variable **$cpt** s'incrémente à chaque changement de check box. Ceci est utile pour contrôler le buffer.

Il est possible afficher une ligne entière en un seul Buffer, donc une fermeture de la connexion est uniquement lorsque **$cpt** atteint 32.

Si c'est la première trame que l'on envoie, alors **envoie=false** et le paramètre **« 1 »** est envoyé pour signaler qu'une nouvelle image doit être dessinée.

```
if($cpt==32)
{
    if($envoie==false)
    {
        $message = "1" . $message;
        $serial->sendMessage($message);
        $envoie=true;
    }
    else
    {
        $message = "2" . $message;
        $serial->sendMessage($message);
    }

    $serial->deviceClose();
    usleep(500000);
    $cpt=0;
    $message="";
    $serial->deviceOpen();
}
```

Sinon c'est le paramètre **« 2 »** qui sera envoyé pour signaler que c'est dans la continuité de l'image. (Voir section: Le cas affichage d'une image : code 1 et 2)

Enfin nous envoyons la variable **$message**, la réinitialisons puis fermons la connexion pendant un certain moment pour que la carte Arduino termine de traiter les données pour enfin ouvrir de nouveau la connexion et recommencer un nouveau cycle.

Améliorations

À terme, nous voulons remplacer les check box de l'interface WEB par des images représentant des cercles plus gros et plus esthétiques. Egalement nous souhaitons, à l'aide d'un panel de couleur comme vu précédemment, que ces cercles prennent la couleur choisie avec le panel. Notre programme PHP étant capable d'envoyer à chaque LEDs une couleur spécifique il ne reste plus qu'à écrire le code JavaScript exécutant cette fonction.

Problèmes rencontrés et Optimisation

Driver et chargement du programme

Après étude de cette librairie nous avons essayé de charger les exemples d'animations qui étaient fournis afin de s'assurer que le programme était correctement chargé sur la carte. Malheureusement nous avons rencontré plusieurs problème du au Driver mais aussi à la version d'Arduino qui était installée sur les PC.

Optimisation de la mémoire

Les tableaux de lettres

La mémoire a été un problème constant lors de notre projet. Le tableau « Lettre » étant très conséquent puisque qu'il contient 59*4*5 = 1180 booléens. Nous nous sommes aperçus **qu'un booléen nécessite autant de place qu'un entier, soit 8 bits !** La mémoire flash de la 1ère carte Arduino ne pouvait contenir seulement 2.5 Ko. Il était pourtant primordial d'enregistrer ce tableau, qui est constant, dans une autre partie de la mémoire. Nous avions deux possibilités qui s'offraient à nous pour l'Arduino Uno :

- **La Flash Memory**
- **La mémoire EEPROM**

Après une recherche d'information nous avons appris que la mémoire EEPROM était lente et qu'il était plus compliqué d'enregistrer des données à l'intérieur de celle-ci. Mais nous décidions cependant de tester les 2 mémoires disponibles, les informations recueillies sur l'EEPROM se sont avérées exactes. La Flash Memory, comme nous le prévoyions, s'est avérée être la plus appropriée.

Pour éviter de rencontrer à nouveau ce problème nous avons modifié quelques parties de notre programme dans le but d'optimiser la SRAM dont nous avons tant besoin.

Le tableau Panneau_LED

Le tableau **Panneau_LED** est celui qui contient la phrase que nous souhaitons afficher. Celui-ci doit être lu et modifier constamment, par conséquent il ne peut pas être enregistré dans la Flash Memory. Ce tableau était à l'origine un tableau de booléen à deux dimensions de taille 800 par 5 soit 4000 booléens. En rappelant que ces booléens prennent 8 bits chacun soit l'équivalent d'un entier.

C'était trop pour le programme, c'est pourquoi, pour optimiser la mémoire de ce tableau la façon la plus simple était d'utiliser des entiers qui contenaient plus d'informations et utilisaient autant de mémoire que les booléens. Le tableau **Panneau_LED** est donc devenu un tableau de 800 entiers, dont chaque entier contenait l'information d'une colonne soit une économie de 3200 octets.

Ayant écris les 59 caractères en booléen nous les avons laissé comme tel (la mémoire Flash contient assez d'espace pour les accueillir). À l'intérieur de la fonction FEEPROM un exposant est utilisé pour permettre d'ajouter soit 1 pour la première ligne, 2 pour la deuxième ligne, 4, 8 et 16. De cette façon l'entier qui par la suite est enregistré dans le tableau **Panneau_LED** contient toutes les informations nécessaires.

Enfin la fonction **LireColonne** permet de décrypter l'entier qui contient les informations sur la colonne pour allumer les LEDs correspondantes.

Le Buffer

C'est le dernier problème de mémoire que nous avons rencontré. En effet pour la transmission de données pour afficher une image il nous fallait envoyer 13 caractères pour chaque LEDs. Le buffer pouvant initialement recevoir 64 caractères à la fois. Pour un affichage plus rapide il était indispensable de l'augmenter à 512 pour afficher une ligne à la fois. Mais la taille du Buffer consommait de la mémoire SRAM, c'est donc à partir de ce moment qu'il était nécessaire d'avoir une carte Arduino plus performante, notre choix c'est porté sur la Arduino Mega 2560, celle-ci ayant une mémoire SRAM de 8Ko.

Alimentation

L'alimentation nous a aussi demandé une réflexion. En effet, pour pouvoir allumer les 352 LEDs à notre disposition il nous fallait plus que celle fournie par le port USB du PC. Nous avons donc relié la carte Arduino à une alimentation indépendante. Mais ce n'était encore une fois pas suffisant, la carte Arduino régule l'intensité envoyée à l'écran. Pour faire les choses correctement il faut donc brancher une alimentation indépendante sur l'écran LEDs afin d'allumer correctement toutes les LEDs de l'écran.

Conclusion

Ressentiment

Au départ ce projet nous aura posé de nombreux problèmes notamment les problèmes de Driver dû à une version d'Arduino qui n'était pas à jour ou encore les problèmes concernant la mémoire. Le départ était également difficile en ce qui concerne la répartition les tâches.

Dans un second temps nous avons décidé de nous concerter pour définir les fonctions dont nous avions besoin puis se partager les fonctions à écrire. La construction de l'écran terminée nous a permis de nous repartir un peu plus de tâches.

Ce projet nous a permis d'apprendre à mieux travailler en groupe sur un même code. Nous avons revu également quelques bases du langage C, appris à lire et utiliser une librairie inconnue en C++ et savoir économiser la mémoire. Enfin pour la partie WEB nous avons assimilé des notions du PHP que nous ne maitrisions pas malgré les cours reçus.

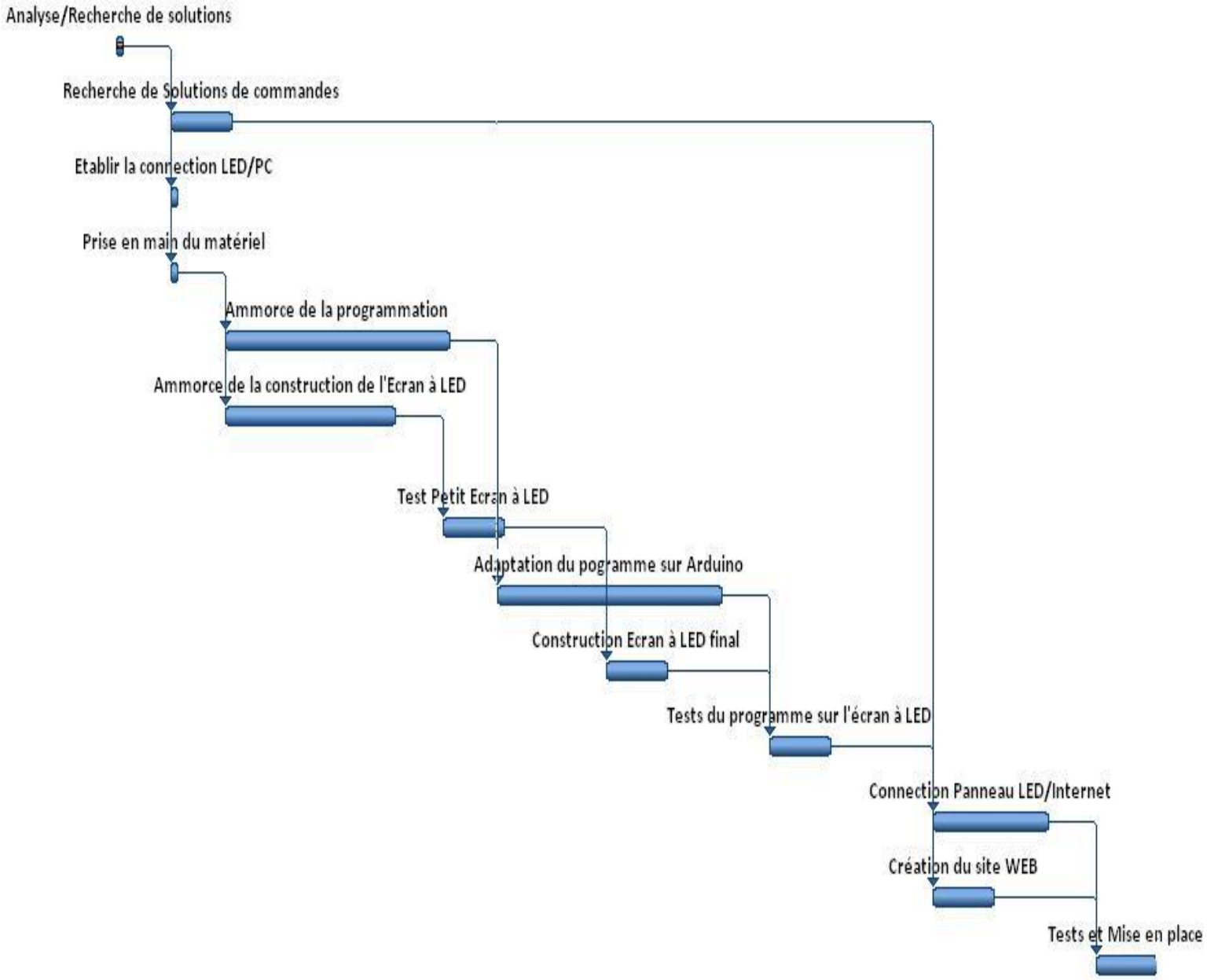
Nous avons pris beaucoup de plaisir dans la réalisation de ce projet, et avons le sentiment d'avoir appris et revu de nombreux points de l'informatique. Nous sommes cependant quelques peu déçu de ne pas avoir réalisé une finalisation digne de ce nom de l'interface d'envoi d'une image du site WEB.

Avenir du projet

Ce projet peut être reprit, amélioré de différentes manières comme par exemple réaliser des animations visuelles. Il serait aussi possible de réaliser un lien avec tweeter pour que les Tweets contenant #istia soit affiché par le panneau. Ou encore réaliser un traitement d'image pour tenter d'afficher l'image souhaité sur le panneau, il faudrait cependant des LEDs supplémentaire pour une plus grande résolution.

Utilisation du temps

Diagramme de Gant



Ci-dessus le diagramme de Gant représentant les différentes taches du projet réparties dans e temps ainsi que leur suite logique.

Diagramme circulaire d'activités

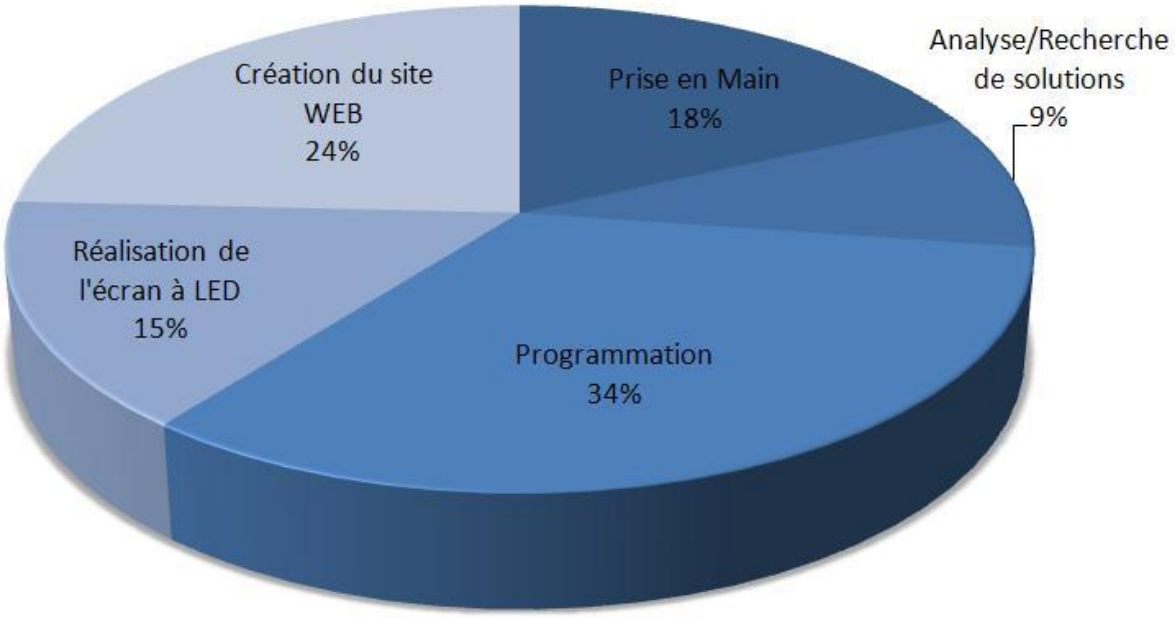


Diagramme circulaire représentant la consommation du temps total du projet.

ECRAN LEDs RVB

Projet réalisé par : Maxime Pontreau, Cyril Le Bras, Mohamed Hammami
Projet encadré par : Sébastien LaGrange

Résumé : Le projet LED est conçu pour afficher et faire défiler les différents événements et informations importantes qui seront annoncées sur la façade de l'ISTIA. Ce projet contient principalement un panneau d'affichage fabriqué par nos soins qui peut allumer indépendamment chacune de ses 352 leds avec plus de 2 millions de couleurs différentes, un microcontrôleur Arduino Mega et un serveur web WampServer. Avec ce matériel nous sommes en mesure de faire défiler des messages sur l'écran d'affichage qui peut afficher 7 caractères en largeur et 2 en hauteur.

Les messages peuvent être créés et envoyer sur le microcontrôleur de l'écran via la page web du projet, où il est possible de choisir la couleur et la vitesse de défilement, ainsi que le nombre de tours à effectuer. Le PC et le microcontrôleur communiquent via un port série. Une fois le message créé et enregistré, l'écran peut être détaché du PC pour faire défiler le message. Ceci peut alors faire défiler une phrase d'un maximum de 140 caractères, l'équivalent d'un tweet, ce qui était le but initial.

Mots-Clés : LED, Défiler, Ecran, Site Web, Message, Caractère

---

Summary: The LED project is developed to provide display source to scroll different events and important notices that will be announced on ISTIA's façade. This project mainly contains a homemade display board that can independently turn on each of the 352 leds in 2 million different colors, an Arduino Mega microcontroller and a Wamp web server. Using these devices we are able to scroll the messages in the display board which is 7 characters long and 2 characters high.

Messages can be created and send on the display's microcontroller by the project's webpage, where it is possible to choose the color and the speed of scrolling, as well as the number of laps. The PC and microcontroller communicate via serial port. After the message is created and saved, the display can be detached from the PC and then plugged in elsewhere to scroll the message. The display can scroll a 140 characters max sentence, just like a tweet, as well as it was the mainly objective.

Keywords : LED, Scroll, Screen, Web Site, Message, Character