École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. (33)2-47-36-14-14
Fax (33)2-47-36-14-22
www.polytech.univ-tours.fr

Parcours des écoles d'Ingénieurs Polytech

Rapport de projet S2

Création d'animations pour le LedWall

Auteur(s)

Alix Bouguereau
[alix.bouguereau@gmail.com]
Samuel Bamba
[smail.bamba@gmail.com]

Encadrant(s)

Carl Esswein
[carl.esswein@univ-tours.fr]

Polytech Tours
Département DI

Version du 17 avril 2015

Table des matières

Table des figures

Introduction

Nous avons choisi le projet du « LedWall » afin de mettre en pratique nos connaissances en informatique et en acquérir de nouvelles. Le projet nous paraissait intéressant car il nous permettait d'utiliser l'informatique et le codage dans une autre situation que celles vues en cours. Durant notre premier et deuxième semestre de L1, nous avons appris les bases du langage de programmation Java, les programmes vus en cours étaient des programmes très abstrait qui permettaient seulement quelques actions simples (tri de nombres, affichage de mots, calcul de moyenne, etc). Ces bases constituent un passage obligé dans tout apprentissage, notamment dans celui d'un langage de programmation, mais en choisissant ce projet, nous voulions avoir une vision nouvelle de l'informatique. En effet, le code que nous devions réaliser était plus intéressant car il devrait avoir un rendu plus visuel, à échelle humaine, à savoir, l'animation d'un mur de Leds.
L'objectif de notre projet était donc de réaliser des animations pour le mur de Leds présent dans le hall du département informatique de Polytech.
Afin de vous faire découvrir notre projet, nous allons dans un premier temps présenter le mur de Leds ainsi que son fonctionnement puis nous parlerons du travail effectué sur les animations.

CHAPITRE 1
L'environnement de travail

1.1 La carte Arduino

Le projet Arduino est issu d'une équipe d'enseignants et d'étudiants de l'école de Design d'Interaction d'Ivrea (Italie). Ils rencontraient un problème majeur à cette période (avant 2003 - 2004) : les outils nécessaires à la création de projets d'interactivité étaient complexes et onéreux (entre 80 et 100 euros). Ces coûts souvent trop élevés rendaient difficiles le développement par les étudiants de nombreux projets et ceci ralentissait la mise en œuvre concrète de leur apprentissage.
Petite anecdote : En l'an 1002, le roi Arduin (Arduino en italien) devint le seigneur du pays, pour être détrôné par Henri II d'Allemagne, deux ans plus tard. Aujourd'hui, le Bar di Re Arduino, un bar dans une rue pavée de la ville, d'Ivrea honore sa mémoire. C'est en mémoire de ce bar, où Massimo Banzi, cofondateur du projet Arduino, avait pour habitude de passer du temps qu'il chosit le nom du projet « Arduino ».
Arduino est une carte microcontrôleur à bas prix qui permet à tout le monde, même aux non-initiés de faire des choses époustouflantes. L'Arduino est capable de se connecter à toutes sortes de capteurs, lampes, moteurs, et autres appareils, et le logiciel permettant de contrôler les créations est assez facile à prendre en main. Vous pouvez construire un affichage interactif, ou un robot mobile, puis en partager les plans avec le monde entier en les postant sur Internet.
La carte est sortie à l'origine en 2005 comme simple outil pour leur école, elle rencontra vite un succès mondial. En effet, son état d'esprit DOY (Do It Yourself- Faites Le Vous-même) et le fait qu'il utilise des licences libres en ont rapidement fait un incontournable dans le monde de l'électronique. Son prix est également un facteur majeur de sa réussite, en effet, la carte Arduino à sa sortie coûtait environ 30 dollars contre une centaine pour les autres cartes du marché.
En 2011, c'est déjà 250 000 cartes Arduino qui ont étaient vendues à travers le monde, aujourd'hui, ce chiffre a sûrement doublé. On remarque un engouement pour la carte Arduino chez les passionnés d'électronique mais également chez les novices qui voient en cette carte comme un moyen de découvrir l'électronique et de développer des créations soit-même, en effet, tous les schémas électroniques et le code source sont disponibles gratuitement en ligne sous des licences libres.
Il existe différent modèles de cartes Arduino, l'une des premières à sortir et l'une des plus populaires est l'Arduino Uno. Aujourd'hui on trouve des cartes beaucoup plus puissantes (Arduino Mega, „) et d'autres ayant des fonctionnalités de connexion (Ethernet, Bluetooth) .
Le langage de programmation utilisé par l'Arduino est un langage proche du C, il a pour but d'être simple à utiliser.

La carte que nous avons utilisé est la carte Arduino Uno dont voici les caractéristiques :

FIGURE 1.1 – Caractéristiques de la carte Arduino Uno

Microcontrôleur	ATmega328
Tension de fonctionnement	5V
Tension d'alimentation (recommandée)	7-12V
Tension d'alimentation (limites)	6-20V
Broches E/S numériques	14 (dont 6 disposent d'une sortie PWM)
Broches d'entrées analogiques	6 (utilisables en broches E/S numériques)
Intensité maxi disponible par broche E/S (5V)	40 mA (ATTENTION : 200mA cumulé pour l'ensemble des broches E/S)
Intensité maxi disponible pour la sortie 3.3V	50 mA
Intensité maxi disponible pour la sortie 5V	Fonction de l'alimentation utilisée - 500 mA max si port USB utilisé seul
Mémoire Programme Flash	32 KB (ATmega328) dont **0.5 KB** sont utilisés par le bootloader
Mémoire SRAM (mémoire volatile)	2 KB (ATmega328)
Mémoire EEPROM (mémoire non volatile)	1 KB (ATmega328)
Vitesse d'horloge	16 MHz

FIGURE 1.2 – La carte Arduino qui contrôle le mur

1.2 Le mur de Leds

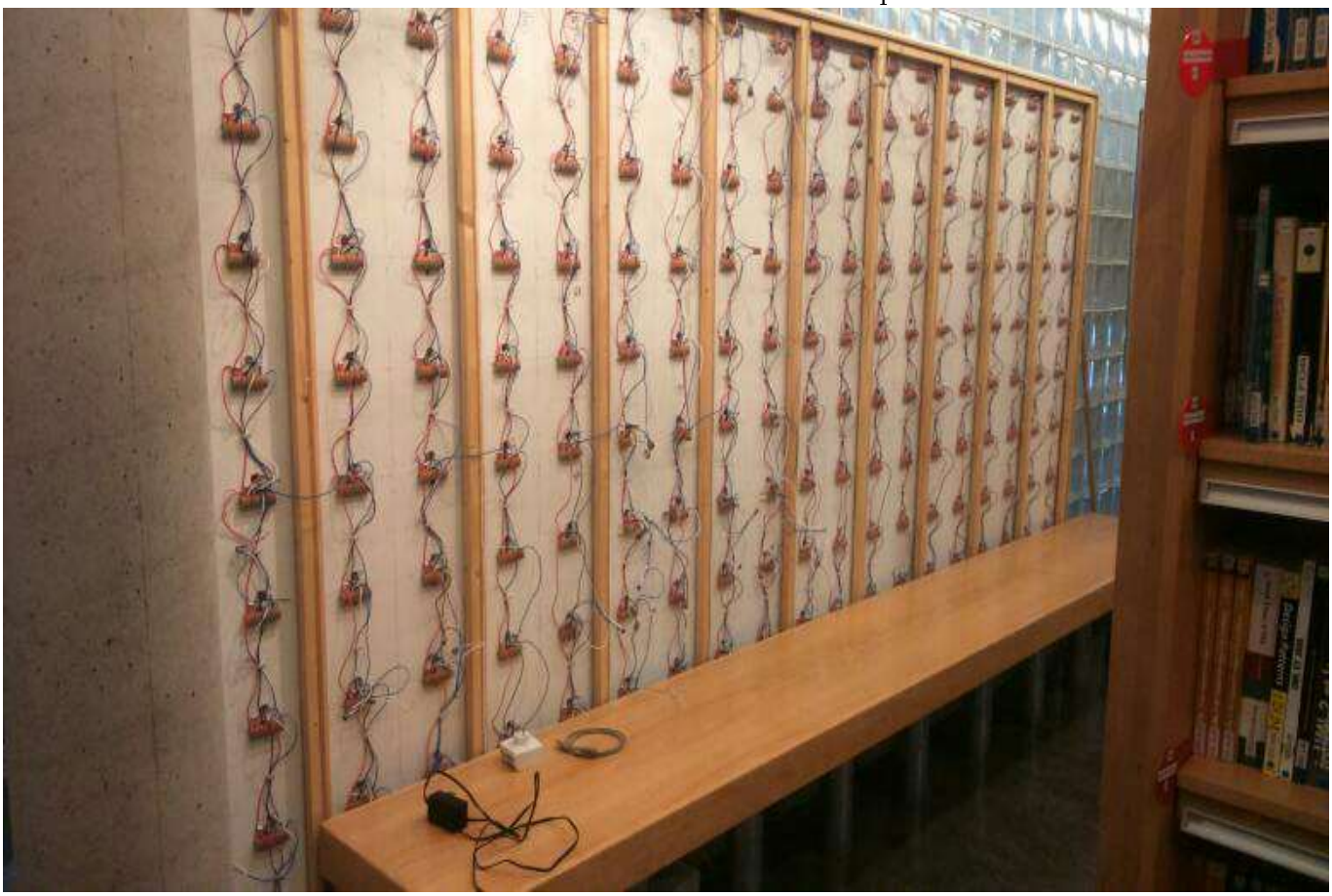
Le mur est installé dans la bibliothèque du département informatique de Polytech' et est visible du hall de l'école. Ce mur a été réalisé par des élèves de l'école durant l'année 2012/2013. Il est composé de 200 cases, réparties sur 10 lignes et 20 colonnes. Chaque case contenant une

FIGURE 1.3 – LedWall vu du hall

Led. Ce sont des Lds RGB munies de 3 résistances et d'un mini-contôleur. Le mur est contrôlé par une carte Arduino Uno.

Malheureusement, quelques problèmes électroniques sont présents et au début de notre projet, on nous indique que seules 10 lignes et 8 colonnes fonctionnent. Ces problèmes électroniques doivent être gérés par des étudiants et des professeurs mais ils s'avèrent complexes à résoudre. Nous devons donc faire en sorte que nos animations s'adaptent au nombre de Leds disponibles.

FIGURE 1.4 – Le mur vu de la bibliothèque



1.3 Problèmes techniques rencontrés

Arrivés à la moitié du temps qui nous était accordé pour mener à bien notre projet, nous avions déjà préparé quelques animations et nous voulions les tester mais le mur de Leds du hall de l'école n'était toujours pas fonctionnel. On nous apprend donc l'existence d'une maquette, dont vous trouverez une photo ci-après, qui se présente sous la forme d'un petit mur de Leds ayant 5 lignes et 5 colonnes soit 25 Leds à contrôler. Le principe de fonctionnement est le même que celui du LedWall.

FIGURE 1.5 – La maquette 5*5

Nous avons alors adapter certaines de nos animations pour qu'elles fonctionnent sur une matrice 5*5.
Malheureusement il s'est avéré que cette maquette n'était pas fonctionnelle non plus.. N'ayant aucune connaissance en électronique, il nous était impossible de comprendre et de trouver une solution au problème rencontré . Malgré l'aide d'un élève en informatique à Polytech' ayant déjà travailler sur ce projet auparavant, il nous fut impossible de la faire fonctionner, de voir la moindre Led s'allumer.

Après cet événement auquel nous ne nous attendions pas nous nous sommes tournés vers notre encadrant qui nous a conseiller de transcrire nos animations en Java afin de les visualiser avec un outil qu'il nous a fournit.

CHAPITRE 2
Les animations

2.1 Le smiley

Une de nos première idée d'animation était d'afficher un smiley souriant qui effectuait des « clignements d'œil » de manière irrégulière . Cette idée nous est venue en regardant une video qui démontrait les possibilités d'un Ledwall comme le notre ; nous pensions en effet, qu'il s'agissait d'une animation assez simple et sympathique pour commencer à nous initier à l'Arduino.
En Arduino :
C'est ainsi que commença notre aventure algorithmique ; Le code du smiley et des animations qui vont suivre repose sur la fonction :

```
BlinkM _ setRGB( matrix[w][h], r, g, b );
```

Cette fonction permet un allumage de la LED – « setRGB » – , qui se trouve à la « (w-1)ème » colonne et à la « (h-1)ème » ligne du Ledwall. De plus cette LED s'allume colorée avec un « pour-deux cent cinquante-cinq-age » de rouge, un « pour-255-age » de vert et un « pour-255-age » de bleu, renseigné respectivement par les variables r, g et b.

On rentre dans le programme Arduino :

```
BlinkM_ setRGB( matrix[3][6] ,255 ,0 ,255 );
```

On obtient alors la led à la 2ème colonne en partant de la droite et à la 5ème ligne qui s'illumine d'un superbe violet (rouge+bleu).

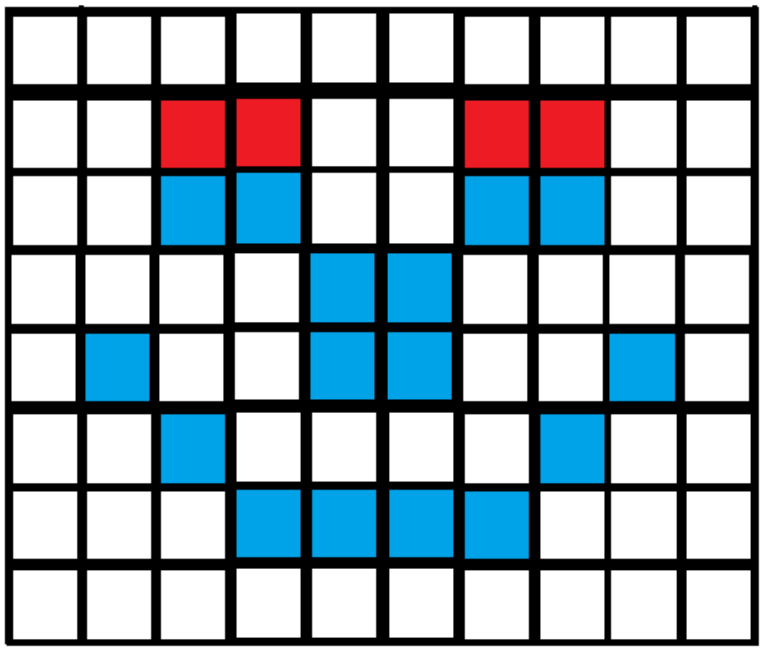
Assez de mise en bouche et voici le programme spécifique à l'animation du smiley :

FIGURE 2.1 – Le Smiley en Arduino

```
93
94  int rgb[] = {0,0,255};
95  int temps = 1000;
96  BlinkM_setRGB( matrix[1][5], rgb[0], rgb[1], rgb[2]);
97
98            (...)
111 BlinkM_setRGB( matrix[6][7], rgb[0], rgb[1], rgb[2]);
112
113 for ( int i=0; i<60; i++) {                          // Le smiley effectuera 60 clins d'oeils, valeur que l'on peut changer
114 BlinkM_setRGB( matrix[2][1], rgb[0], rgb[1], rgb[2]);
115 BlinkM_setRGB( matrix[3][1], rgb[0], rgb[1], rgb[2]);
116 BlinkM_setRGB( matrix[6][1], rgb[0], rgb[1], rgb[2]);
117 BlinkM_setRGB( matrix[7][1], rgb[0], rgb[1], rgb[2]);
118 delay(temps);
119 BlinkM_setRGB( matrix[2][1], 0, 0, 0);
120 BlinkM_setRGB( matrix[3][1], 0, 0, 0);
121 BlinkM_setRGB( matrix[6][1], 0, 0, 0);
122 BlinkM_setRGB( matrix[7][1], 0, 0, 0);
123 delay(400);
124 temps = random(1000,5000); // On défini un temps d'attente aléatoire (entre 1 et 5 sec) entre deux clignements
125 }
```

Affichage des leds des cases bleues

Affichage et éteignage successif (après un certain temps - *delay(temps)* -) des cases rouges

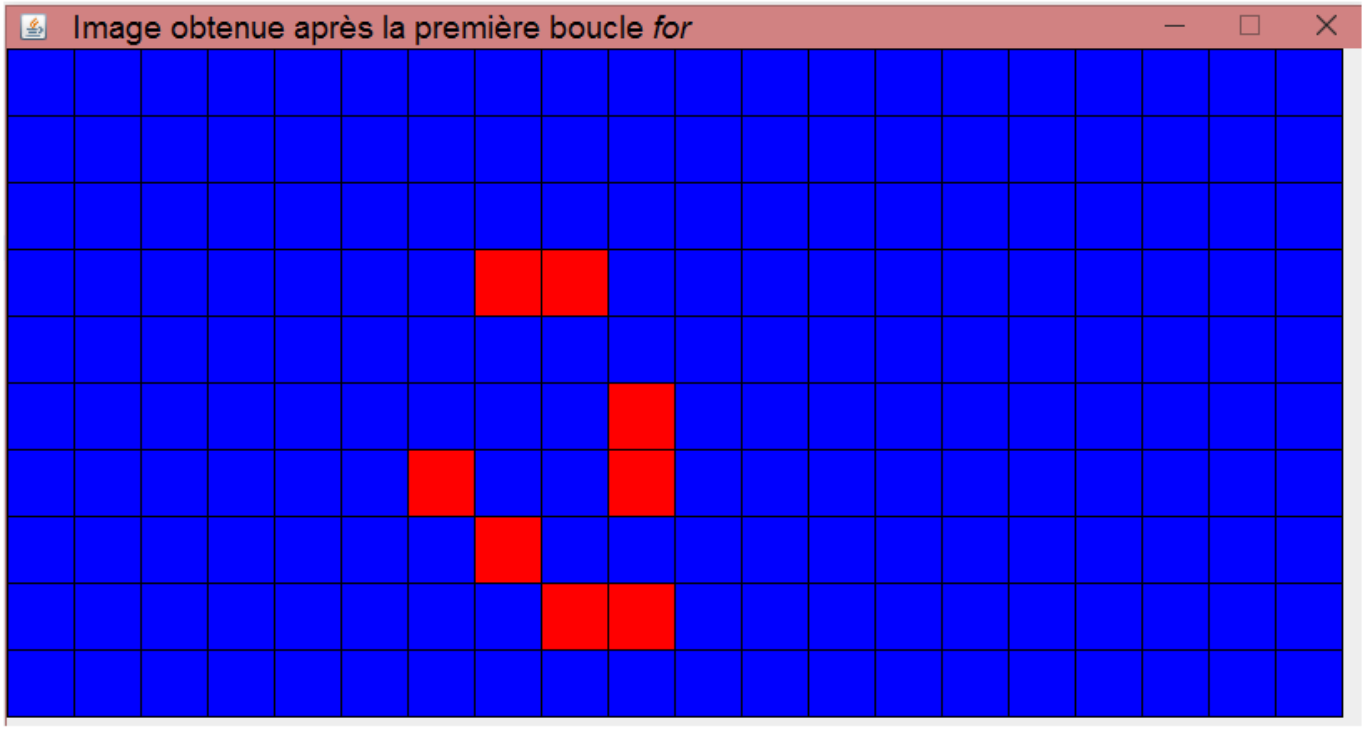


En Java : La simulation en java est pratiquement la même, à l'exception près que, aillant remarqué un axe de symétrie au niveau du milieu vertical du smiley et pour éviter de recopier toutes les positions respectives des leds à allumer, on ne renseigne que les leds à allumer du coté gauche et on inverse les positions pour obtenir le coté droit.

FIGURE 2.2 – Le Smiley en Java

```
7   public class Smiley {

10      private int width = 20;
11      private int height = 10;
12      private int col = width/2;
13      private int lig = height/2;
18      private Color colors[][];

20      private Color ColorFace = Color.RED; // Couleur du Smiley

49          int k =1,l=0;
50
51          for (int i= 0; i <2; i++){ // On fait deux boucles - pour le coté gauche et pour le coté droit de l'image -.
52
53              colors[col-(k)*(4-l)][lig+1] = ColorFace;
54              colors[col-(k)*(3-l)][lig+2] = ColorFace;
55              colors[col-(k)*(3-l)][lig-2] = ColorFace;
56              colors[col-(k)*(2-l)][lig+3] = ColorFace;
57              colors[col-(k)*(2-l)][lig-2] = ColorFace;
58              colors[col-(k)*(1-l)][lig+1] = ColorFace;
59              colors[col-(k)*(1-l)][lig] = ColorFace;
60              colors[col-(k)*(1-l)][lig+3] = ColorFace;
61
62              // On s'occupe du coté droit:
63              k = -1;
64              l=1;
65          }
```

les variables *col* et *lig* servent à centrer l'image peu importe la taille de la grille



2.2 Animation "Bienvenue"

Étant donné le placement très tactique du Ledwall dans le hall d'entrée, nous avions déduit que nos animations serraient vues maintes et maintes fois par un nombre important de passants dans la journée. En plus de nous emplir de joie, ce fait nous donna une idée d'animation : il nous fallait écrire un message à ces passants. Intemporel, assez court et possédant un

fort potentiel de transmission de bonne humeur, « Bienvenue » parut être le message approprié.

En Arduino :

Le mot « Bienvenue » fait 9 lettres, il nous était donc impossible d'afficher le mot en entier sur un écran de 20x10 -ou même 10x8-, il fallait faire un système de défilage.
Le code en Arduino utilise une boucle for avec un compteur dénommé i qui a pour mission de décaler chaque colonne vers la gauche, pour cela i croît tout en s'ajoutant à chaque colonnes.

FIGURE 2.3 – Aperçu de l'animation "Bienvenue" n°1

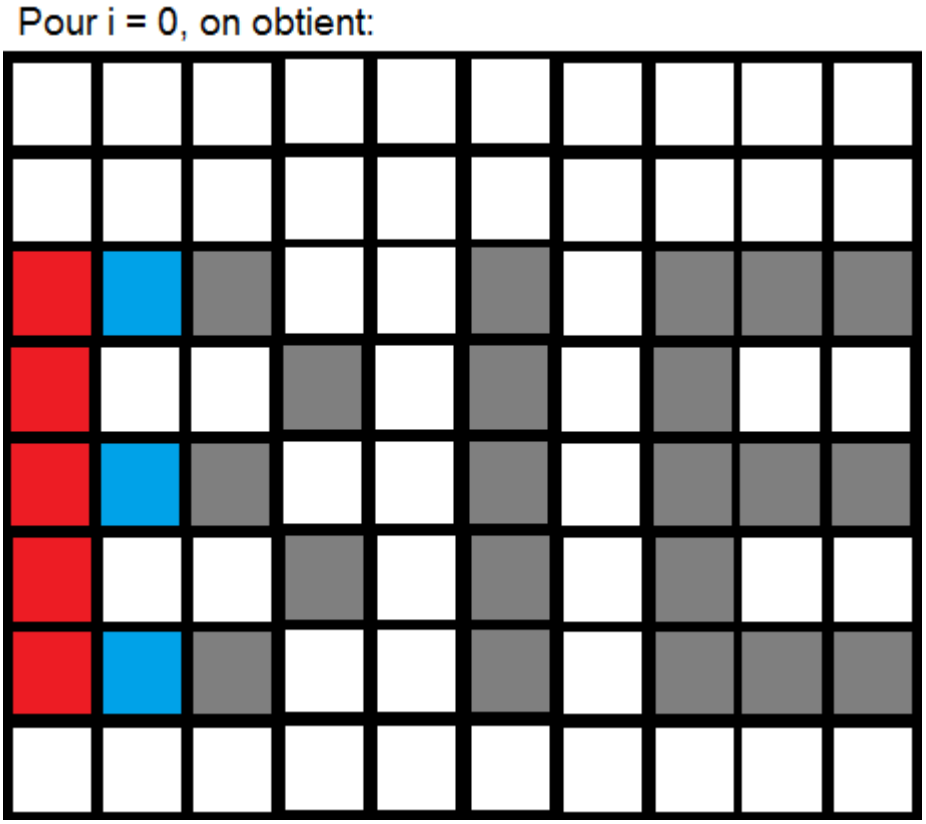


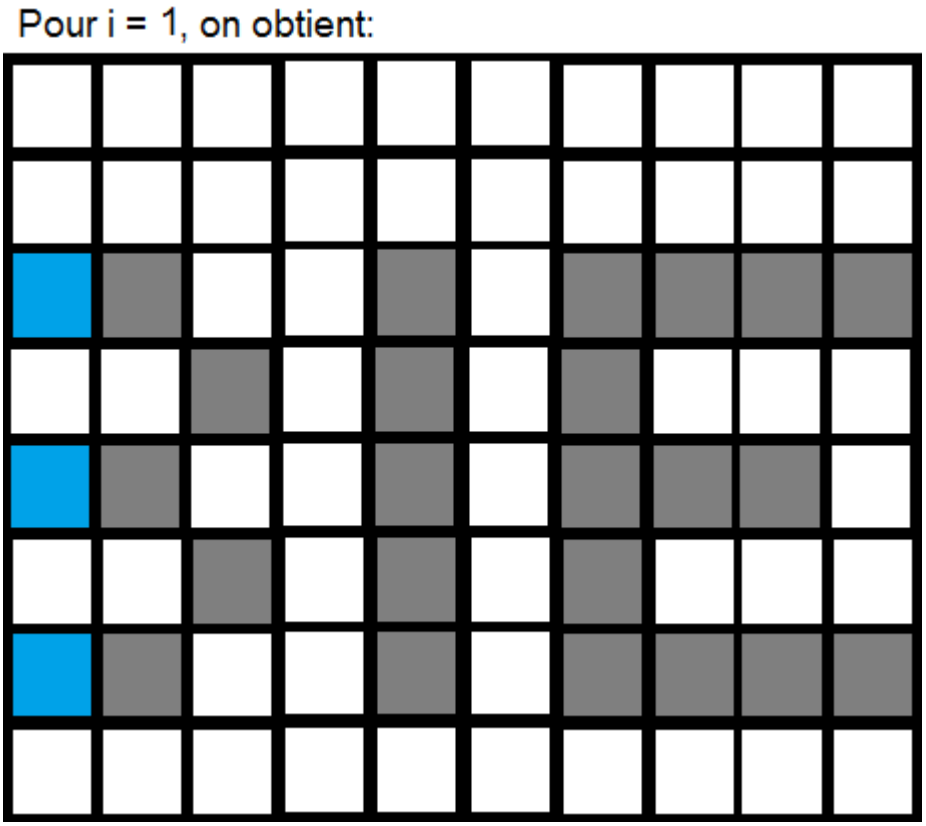
FIGURE 2.4 – L'animation "Bienvenue" en Arduino

```
116 void bienvenue(int i, int rgb[]){
117
118   for(int i=0; i <42;i++){
119
120     while (i==0 && width > 0){
121 BlinkM_setRGB( matrix[width+i][2], rgb[0], rgb[1], rgb[2]);
122 BlinkM_setRGB( matrix[width+i][3], rgb[0], rgb[1], rgb[2]);
123 BlinkM_setRGB( matrix[width+i][4], rgb[0], rgb[1], rgb[2]);
124 BlinkM_setRGB( matrix[width+i][5], rgb[0], rgb[1], rgb[2]);
125 BlinkM_setRGB( matrix[width+i][6], rgb[0], rgb[1], rgb[2]); }
126
127     while (i<=1 && i >= 1-width){
128 BlinkM_setRGB( matrix[width-1+i][2], rgb[0], rgb[1], rgb[2]);
129 BlinkM_setRGB( matrix[width-1+i][4], rgb[0], rgb[1], rgb[2]);
130 BlinkM_setRGB( matrix[width-1+i][6], rgb[0], rgb[1], rgb[2]);  }
131
132     while (i<=2 && i >= 2-width){
133 BlinkM_setRGB( matrix[width-2+i][3], rgb[0], rgb[1], rgb[2]);
134 BlinkM_setRGB( matrix[width-2+i][4], rgb[0], rgb[1], rgb[2]);
135 BlinkM_setRGB( matrix[width-2+i][6], rgb[0], rgb[1], rgb[2]); }
136
137     while (i<=3 && i >= 3-width){
138 BlinkM_setRGB( matrix[width-3+i][4], rgb[0], rgb[1], rgb[2]);
139 BlinkM_setRGB( matrix[width-3+i][5], rgb[0], rgb[1], rgb[2]); } // FIN Lettre B
140
141     while (i<=5 && i >= 5-width){
142 BlinkM_setRGB( matrix[width-5+i][2], rgb[0], rgb[1], rgb[2]);
        (...)
283 BlinkM_setRGB( matrix[width-40+i][6], rgb[0], rgb[1], rgb[2]); }
284
285     while (i<=41 && i >= 41-width){
286 BlinkM_setRGB( matrix[width-41+i][2], rgb[0], rgb[1], rgb[2]);
287 BlinkM_setRGB( matrix[width-41+i][6], rgb[0], rgb[1], rgb[2]); }
288
289 delay(500);
290
291 for (int w = 0; w< width; w++)// On efface l'écran après une courte pause entre deux decalages
292    for(int h = 0; h< height; h++)
293    BlinkM_setRGB( matrix[w][h], 0, 0, 0);
294 }}
```

Affichage de la première colonne (cases rouge)

Affichage de la deuxième colonne (cases bleues)

FIGURE 2.5 – Aperçu de l'animation "Bienvenue" n°2



Cependant, un programme tel qu'on vous l'a introduis ci-dessus ne fonctionne pas ; en effet, si on continu de soustraire i à une colonne, elle va tôt ou tard sortir de l'écran et déclencher une erreur. Néanmoins, vous avez probablement remarqué dans le code plus haut des mystérieux while dont je n'ai pas encore fait mention, ces while permettent de contourner l'erreur.
Le while de la nème colonne ne débloque le passage que si i est inférieur ou égal à n puisque à la boucle n+1, cette même nème colonne se retrouverait à gauche, hors de l'écran (se référer aux figures). De plus, toutes les colonnes qui n'attendent que leur tour pour défiler et qui démarrent par conséquent à droite hors de l'écran ont un while qui bloque leur affichage tant que i est supérieur ou égal à n moins la largeur du Ledwall.
Un exemple : on considère la largeur du mur comme égale à 20 et on s'intéresse à la 21ème colonne (la première colonne à droite et en dehors de l'écran) :
- première boucle $i = 0$: $n - largeur = 21 - 20 = 1$, on a $1 > i$: l'affichage n'a pas lieu
- seconde boucle $i = 1$: $21 - 20 = 1$, on a $i \geq 1$: l'affichage commence.

En Java : Là encore, il y a beaucoup de similitudes entre le programme en Arduino et celui en Java. En effet le programme Java affiche lui aussi « Bienvenue » , mais nous l'avons construit de sorte qu'il puisse être modifié facilement pour afficher d'autres messages ou images ; concrètement c'est un programme qui affiche les fonctions qu'on lui donne et les décale de droite à gauche.

La différence majeure entre les deux programmes -Java et Arduino- est que les while en Arduino sont remplacés par :

$$\boxed{try \quad catch\ (\ ArrayIndexOutOfBoundsException\ e)\ ,}$$

En fait, au lieu de s'embêter à définir des conditions pour l'affichage d'une colonne, on fonce droit dans le mur et on compte sur le catch pour attraper l'erreur (*ArrayIndexOutOfBoundsException e*) qui va inévitablement surgir, à ce moment là on en déduit que la colonne en question est en dehors de l'écran et on répète l'expérience en affichant uniquement la colonne d'après ainsi que celles qui la suivent, et ainsi de suite. Si toutes les colonnes débouchent sur une erreur, la lettre est soit en dehors de l'écran du côté gauche, soit en dehors du coté droit et rien ne se passe jusqu'à la prochaine boucle du for. Dans cette boucle for , les try s'enclenchent à nouveau pour vérifier si une nouvelle colonne vient d'entrer dans l'écran et/ou si une autre colonne vient d'en sortir.

FIGURE 2.6 – L'animation "Bienvenue" en Arduino

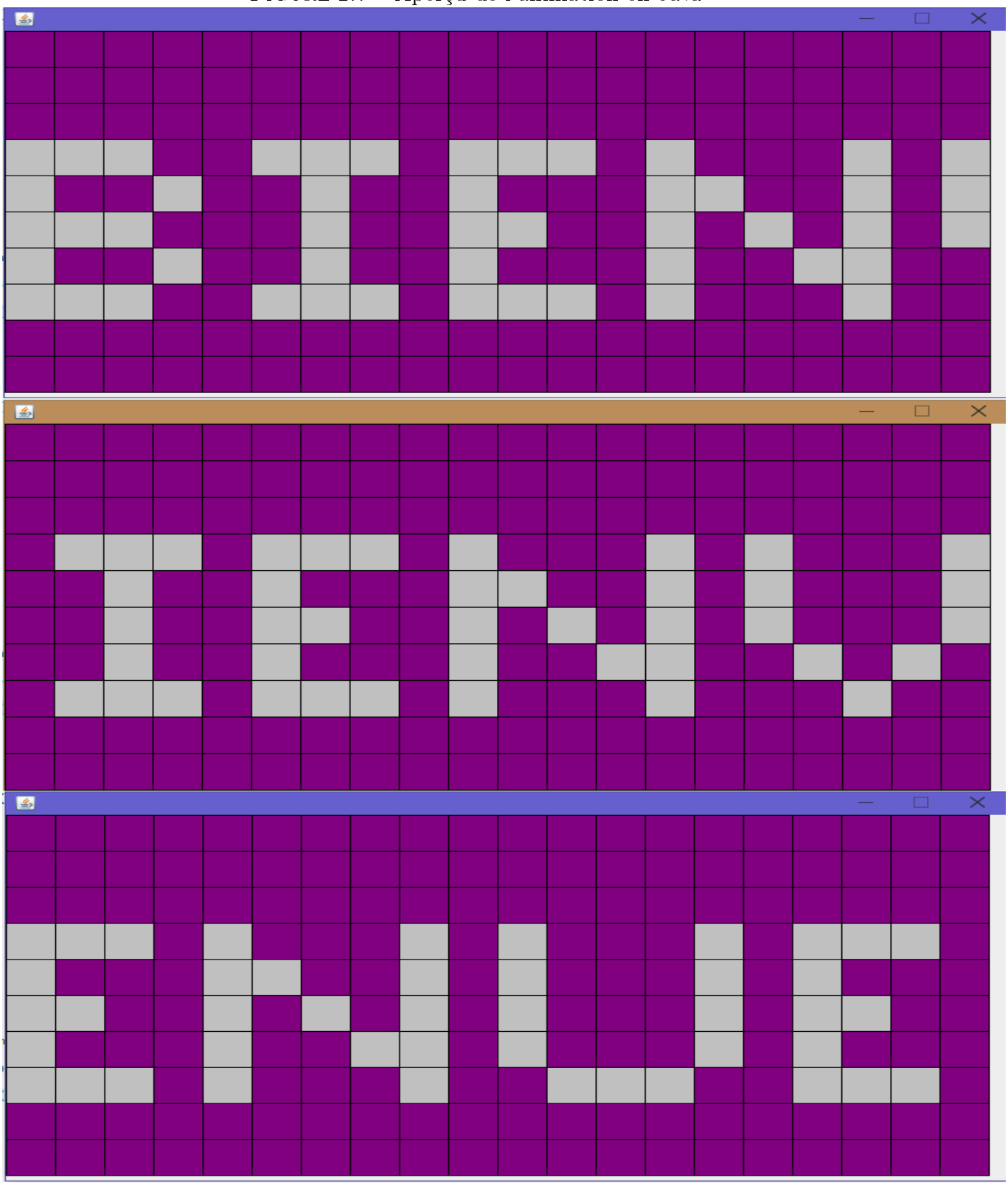
```
8   public class Bienvenue {
9
10
11      private int width = 20;
12      private int height = 10;
18      private Color colors[][];
20      private Color ColorMessage = Color.LIGHT_GRAY;
22      private Color Default = new Color(128, 0, 128); // Violet
49          for(int decal =0; decal<44; decal++){ //compteur de décalage de l'animation
50
51          LettreB(1,decal);
52          LettreI(6,decal);
53          LettreE(10,decal);
54          LettreN(14,decal);
55          LettreV(20,decal);
56          LettreE(26,decal);
57          LettreN(30,decal);
58          LettreU(36,decal);
59          LettreE(42,decal);
60
61          try
62          {
63              Thread.sleep(1000);
64          } catch (InterruptedException e1)
65          {
66              e1.printStackTrace();
67          }
68
69          for (int w = 0; w < width; w++)
70              for (int h = 0; h < height; h++)
71                  colors[w][h] = Default ;}
```

Le *Thread.sleep* met en pause l'animation entre deux décalages

```
165     public void LettreI (int PositionIni, int decal){
166
167         try{
168             colors[PositionIni-decal][lig-2] = ColorMessage;
169             colors[PositionIni-decal][lig+2] = ColorMessage;
170             colors[PositionIni+1-decal][lig-2] = ColorMessage;
171             colors[PositionIni+1-decal][lig-1] = ColorMessage;
172             colors[PositionIni+1-decal][lig] = ColorMessage;
173             colors[PositionIni+1-decal][lig+1] = ColorMessage;
174             colors[PositionIni+1-decal][lig+2] = ColorMessage;
175             colors[PositionIni+2-decal][lig-2] = ColorMessage;
176             colors[PositionIni+2-decal][lig+2] = ColorMessage;
177
178         } catch (ArrayIndexOutOfBoundsException i){
179             try{
180                 colors[PositionIni+1-decal][lig-2] = ColorMessage;
181                 colors[PositionIni+1-decal][lig-1] = ColorMessage;
182                 colors[PositionIni+1-decal][lig] = ColorMessage;
183                 colors[PositionIni+1-decal][lig+1] = ColorMessage;
184                 colors[PositionIni+1-decal][lig+2] = ColorMessage;
185                 colors[PositionIni+2-decal][lig-2] = ColorMessage;
186                 colors[PositionIni+2-decal][lig+2] = ColorMessage;
187
188             } catch (ArrayIndexOutOfBoundsException j){
189                 try{
```

Voici un aperçu du rendu final de l'animation en Java :

FIGURE 2.7 – Aperçu de l'animation en Java



2.3 Lignes horizontales

Après avoir fait quelques animations (2 pour être exact), il nous fallait une animation de transition, qui nous permettrait, et bien, de transiter facilement et harmonieusement entre nos différentes animations. Encore une fois, c'est une vidéo youtube qui nous insuffla notre idée d'animation ; dans cette vidéo on y voyait un immense Ledwall, des barres horizontales arrivaient une ligne sur deux de la gauche et croisaient d'autres barres horizontales arrivant par la droite. Nous réalisions peu à peu le nombre de possibilités qu'un Ledwall assez grand avait à offrir et décidâmes de reproduire cette animation à moindre échelle.

En Arduino :
En premier lieu on sépare l'animation en deux parties, gauche et droite, les lignes arrivant par la gauche sont associées aux lignes paires du Ledwall et celles arrivant par la droite doivent se contenter des lignes impaires.
Un nombre pair est de la forme 2*k (pour tout k appartenant à Z) on allume donc toutes les Leds à la position :

*matrix[2*k][width-1]*

width-1 correspondant à la colonne toute à gauche du Ledwall. De plus on veut des lignes de 5 leds on allume donc l'avant dernière colonne [width-2], l'antépénultième [width-3], [width-4] et [witdh-5].
A droite, on a quelque chose de similaire avec 2*k+1 à la place de 2*k puisque cette fois ce sont les lignes impaires qui nous intéressent et [0] à la place de [width-1] pour la première colonne, [1] pour la seconde, [2], [3] et [4]. Après une courte pause tout est effacé et on décale les colonnes, en soustrayant du coté gauche et en additionnant du coté droit.

FIGURE 2.8 – Aperçu de l'animation "lignes horizontales"

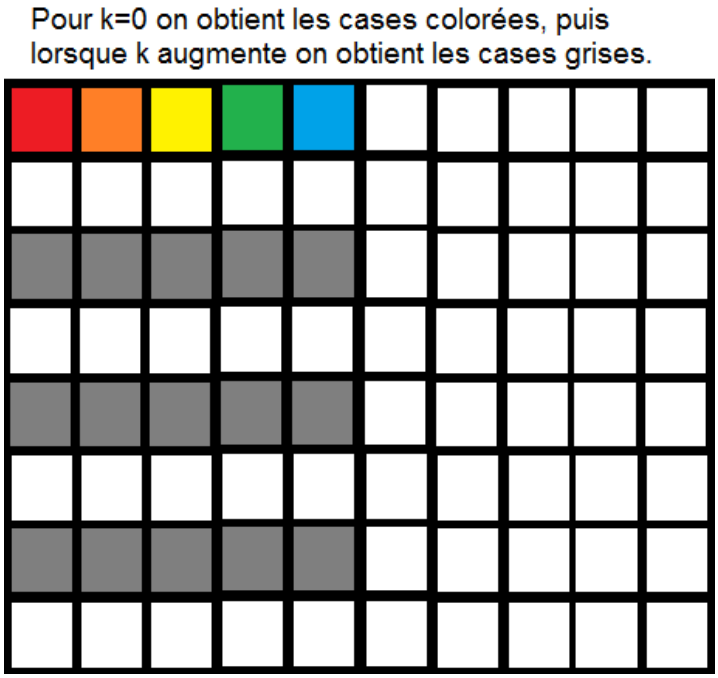
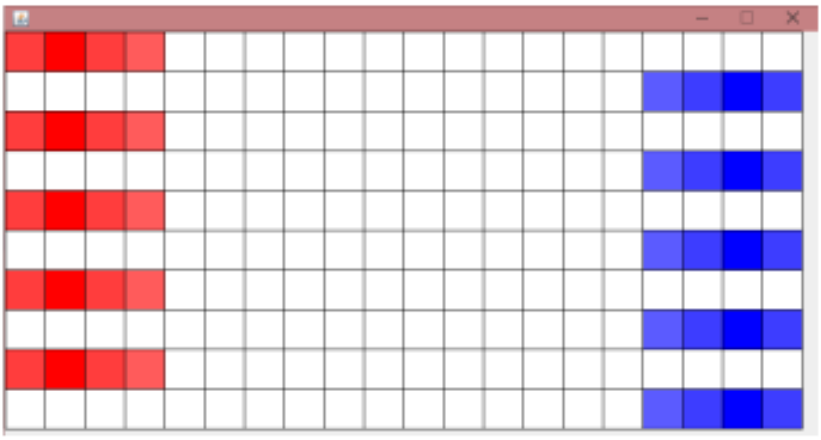


FIGURE 2.9 – L'animation "Lignes horizontales" en Arduino

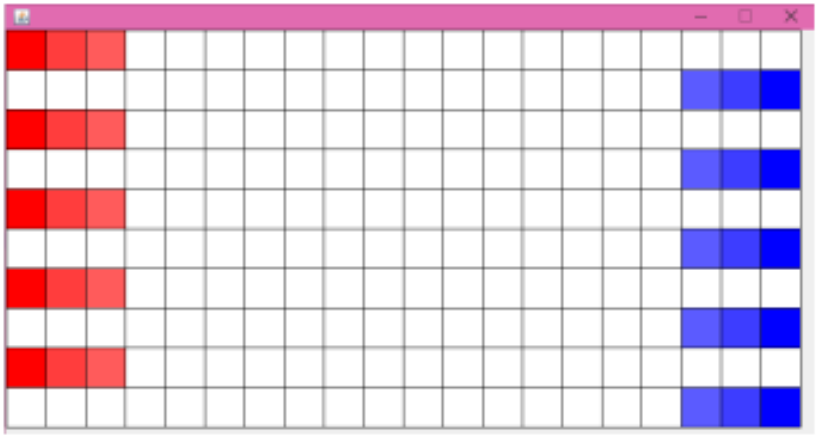
```
102  void barreshorizontales(int duree){
103
104   // BlinkM_setRGB( matrix[ligne][colonne], rgb[0], rgb[1], rgb[2]);
105
106   for (int i = 0; i < (int) duree; i++){ //temps d'animation
107     for (int decal = 0; decal < width-5; decal++){ // compteur qui décalle l'animation
108       for (int k = 0; k < height/2 ; k++){
109    // lignes rouges des lignes paires arrivant par la gauche
110        BlinkM_setRGB( matrix[2*k][width-1-decal], 150, 0, 0);  } case rouge
111        BlinkM_setRGB( matrix[2*k][width-2-decal], 200, 0, 0); } case orange
112        BlinkM_setRGB( matrix[2*k][width-3-decal], 255, 0, 0);  } case jaune
113        BlinkM_setRGB( matrix[2*k][width-4-decal], 200, 0, 0); } case verte
114        BlinkM_setRGB( matrix[2*k][width-5-decal], 150, 0, 0);} case bleue
115    // lignes bleues des lignes impaires arrivant de la droite
116        BlinkM_setRGB( matrix[2*k+1][decal], 0, 0, 150);
117        BlinkM_setRGB( matrix[2*k+1][1+decal], 0, 0, 200);
118        BlinkM_setRGB( matrix[2*k+1][2+decal], 0, 0, 255);
119        BlinkM_setRGB( matrix[2*k+1][3+decal], 0, 0, 200);
120        BlinkM_setRGB( matrix[2*k+1][4+decal], 0, 0, 150);
121
122        delay(200);
123        //On eteint toutes les Leds précedement allumées après une courte pause
124
125        BlinkM_setRGB( matrix[2*k][width-1-decal], 0, 0, 0);
126        BlinkM_setRGB( matrix[2*k][width-2-decal], 0, 0, 0);
127        BlinkM_setRGB( matrix[2*k][width-3-decal], 0, 0, 0);
128        BlinkM_setRGB( matrix[2*k][width-4-decal], 0, 0, 0);
129        BlinkM_setRGB( matrix[2*k][width-5-decal], 0, 0, 0);
130
131        BlinkM_setRGB( matrix[2*k+1][decal], 0, 0, 0);
132        BlinkM_setRGB( matrix[2*k+1][1+decal], 0, 0, 0);
133        BlinkM_setRGB( matrix[2*k+1][2+decal], 0, 0, 0);
134        BlinkM_setRGB( matrix[2*k+1][3+decal], 0, 0, 0);
135        BlinkM_setRGB( matrix[2*k+1][4+decal], 0, 0, 0);
136     }}
```

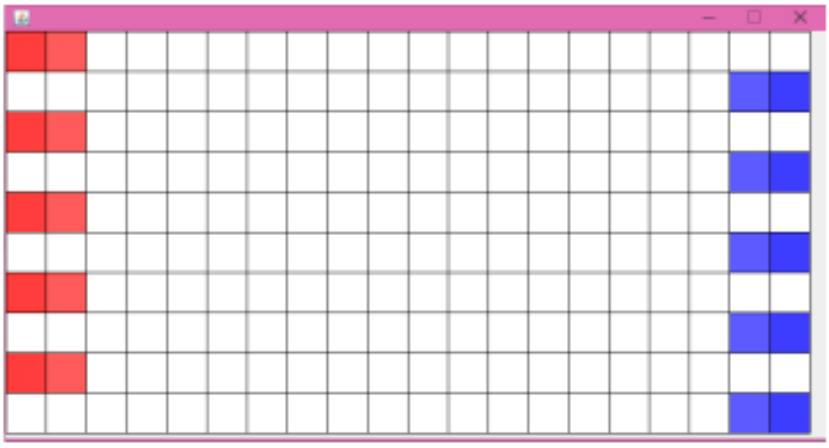
La boucle se répète jusqu'à ce que les lignes atteignent les bord, ensuite pour éviter une erreur de décalage hors de l'écran on traite séparément les 4 affichages restants pour décaler et faire disparaître les barres hors de l'écran. Le premier affichage allume 4 leds (au lieu de 5) toutes les deux lignes tout à gauche et tout à droite.

FIGURE 2.10 – Aperçu de l'animation "lignes horizontales" en Java



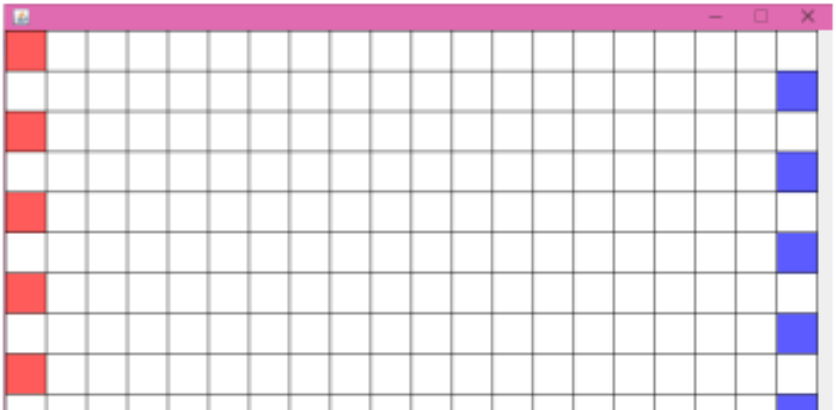
le second affichage en allume 3,





le troisième 2

et le dernier une seule.



En Java : Le programme est pratiquement identique en Java, par conséquent si vous souhaitez une explication détaillée de son fonctionnement vous pourrez en trouver une deux pages plus haut.
Les seules différences sont dues à des équivalences de langage :
En Arduino :

```
BlinkM_ setRGB( matrix[][], 150, 0, 0);
delay();
```

équivaut en Java à :

```
colors[][] = new Color(150,0,0);
Tread.sleep();
```

2.4 Animation "Equalizer"

Cette animation faisait partie de nos toutes premières idées et nous tenions vraiment à la réaliser. Une animation de type « Equalizer » a pour but de transcrire un son en une visualisation, notre animation Equalizer cependant, ne se base pas sur un son mais affiche une visualisation aléatoire. Il paraissais trop difficile de recréer un véritable Equalizer.

En Arduino :
Nous avons réalisé 3 types d'Equalizer, un pour un mur d'une largueur de 20 Leds, un pour un mur d'une largueur de 8 Leds et un avec des couleurs aléatoires qu'on peut normalement utiliser pour toutes les largueurs de mur.
Le premier Equalizer et le second ont la même base avec pour seule différence les couleurs des colonnes :
– Dans l'Equalizer 20 colonnes les couleurs varient du bleu pour la première colonne à gauche, au jaune à la dernière colonne de droite. Les couleurs changent à chaque colonne tout en gardant des couleurs proches de façon à avoir un résultat agréable à l'œil.
– Dans l'Equalizer 8 colonnes, toujours dans un soucis d'esthétique, nous avons fait le choix d'utiliser les couleurs de l'arc-en-ciel ; étant donné que ceux-ci sont reconnus par la grande majorité des enfants de la Terre comme « Jolis ».

FIGURE 2.11 – Code Equalizer Couleurs Fixes

```
107   int hcolonne = 0;
108
109   int rgb[] = {0,0,0};
110
111   for (int i= 0; i< duree*2; i++){ // temps d'animation
112     for (int w =0; w< width ; w++){
113       int hcolonne = random(0, height);  // hauteur aléatoire de la colonne
114
115      // Version 8 couleurs:
116       switch (w) {
117      case 0: rgb[0] = 255; // couleur rouge
118      break;
         (...)
135      case 7: rgb[0] = 138;
136              rgb[1] = 43;
137              rgb[2] = 226; // bleu-violet
138      break;
139      }

141       /* Version 20 couleurs:
142
143       switch (w) {
144      case 0: rgb = {0,0,255}
145      break;
         (...)
182      case 19: rgb = {255,255,102}
183      break;
184     }
185     */
186
187       for (int h = 0; h < hcolonne; h++) //Affichage ligne par ligne
188    BlinkM_setRGB( matrix[w][h], rgb[0], rgb[1], rgb[2]);
189
190
191      delay (500); // pause d'une demi-seconde avant l'affichage d'un nouveau equalizer
192
193      for (int wid= 0; wid< height; wid++){ // on éteint les led
194        for (int h =0; h< width ; h++)
195         BlinkM_setRGB( matrix[wid][h], 0,0,0);}
196
197
198 }
199     } }
```

Toutes les couleurs de l'animation sont contenues dans le *switch*

L'affichage se fait colonne par colonne, on défini une hauteur de colonne aléatoire : hcolonne, on allume toute les Leds jusqu'à cette hauteur, la première boucle se termine, on passe à la seconde colonne, hcolonne reprend une valeur aléatoire, le tableau rgb est modifié par le switch et ce la défini la nouvelle couleur selon laquelle on allumer les Leds de la nouvelle colonne.
Le code de L'Equalizer à couleurs aléatoire est le même à la différence qu'il n'y a pas de switch et le tableau rgb change aléatoirement.

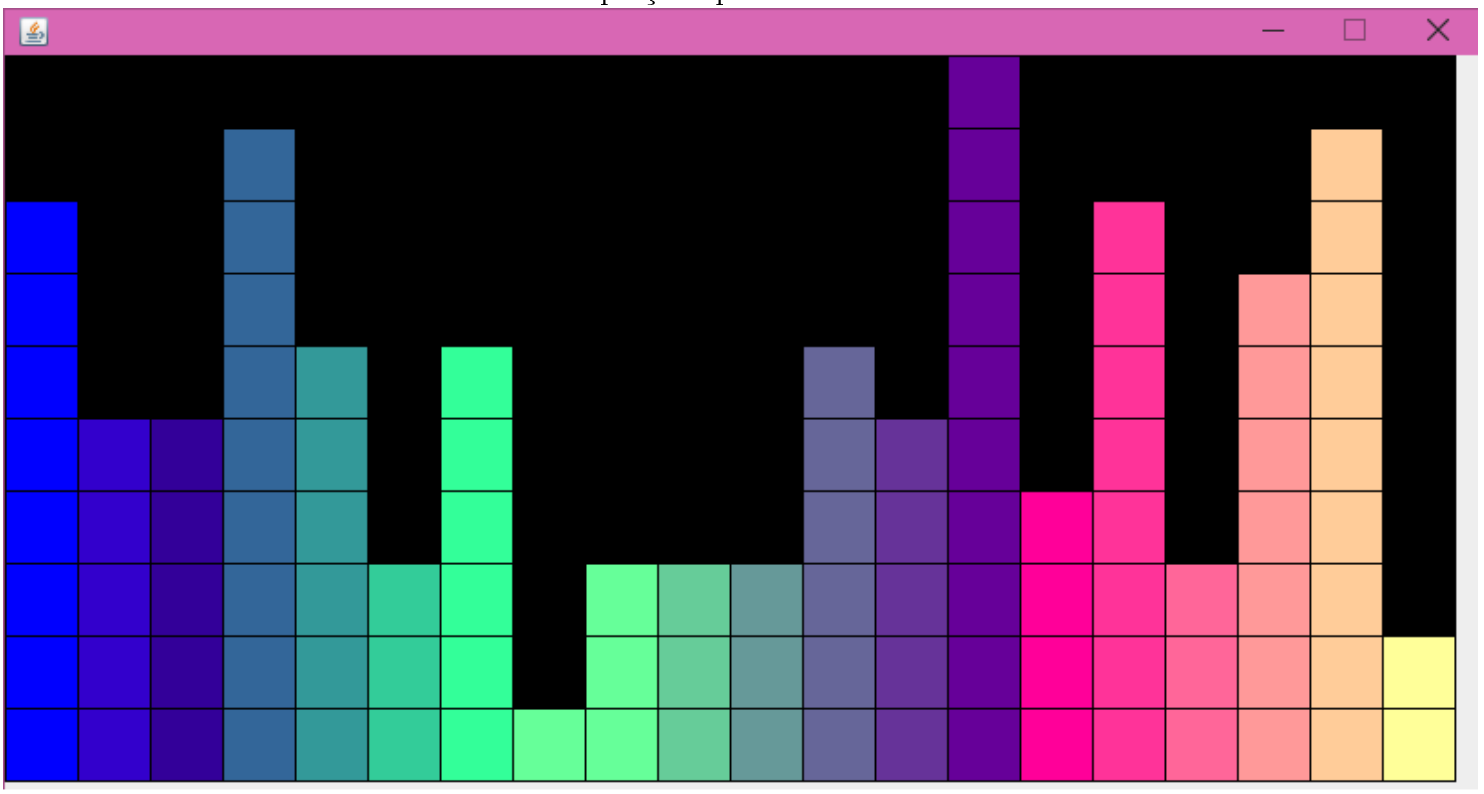
En Java :
Le code en Java est identique à celui en Aduino sauf qu'on utilise un tableau pour mémoriser les différentes couleurs et non un switch.

FIGURE 2.12 – Code Equalizer en Java

```
package fr.polytech.s20;


    import java.awt.*;

    public class Equalizer20couleurs {


        private int width = 10;
        private int height = 20;
        private int r=0,g=0,b=0;
        private Color CouleurAlea = new Color(r,g,b);

        private int temporisation = 0; // temps entre deux animations
        private int temps = 15000; //temps d'animation en ms
        private int hmax = 0; //hauteur aléatoire d'une colonne
        private int hmax1 =height; //hauteur maximale pour l'effet vague
        private int k =0;
        private Color colors[][];

        // L'equalizateur possède 2 modes + le mode couleurs aléatoire:

        private Color[][] MatrixCouleurs = {{},{},{},{},{},{},{},{},
        // MatrixCouleur[8][] contient les 8 couleurs du mode Arc-en-ciel
                {Color.RED,Color.ORANGE,Color.YELLOW,new Color(173,255,0),Color.GREEN,new Color(0,255,0),Color.BLUE},{},{},{},{},{},{},{},{},{},{},{},{},
        // MatrixCouleur[20][] contient les 20 couleurs du mode 20 couleurs
                {new Color(0,0,255),new Color(51,0,204),new Color(51,0,153),new Color(51,102,153),new Color(51,153,153),new Color(51,204,153),new Color(51,255,153),ne (...)

        private Color Default = Color.BLACK;

                for(int i =0; i <100; i++){
                for(int w=0; w<width; w++){
                    hmax = (int) (Math.random()*(height))+1;
                    for(int h=0; h<hmax; h++)
                        colors[w][height-1-h] = MatrixCouleurs[width][w];
                }
                try
                {
                    Thread.sleep(175);
                } catch (InterruptedException e)
                {
                    e.printStackTrace();
                }

                    for (int w = 0; w < width; w++)
                        for (int h = 0; h < height; h++)
                            colors[w][h] = Default ;

                }}

            try
            {
                Thread.sleep(temporisation);
            } catch (InterruptedException e)
            {
                e.printStackTrace();
            }

        }
```

Et puisque c'est « joli » voici un rendu du programme une fois lancé :

FIGURE 2.13 – Aperçu Equalizer en Java



Conclusion

N'ayant aucune connaissance en informatique à notre arrivée en Peip 1 nous voulions en acquérir de nouvelles, de nouvelles qui paraissaient plus intéressantes que les cours des premiers semestres. On espérait que ce projet nous donne une nouvelle vision de l'informatique, que l'on puisse voir le fruit de nos codes sur un mur de Leds grandeur nature, malheureusement celui-ci n'a pas fonctionné durant toute la période de notre projet. Néanmoins nous avons pris du plaisir à créer ces animations. Nous regrettons quand même de ne pas avoir créer plus d'animations, ceci est du au fait que nous avons passer beaucoup de temps à chercher à comprendre la raison du non-fonctionnement du mur et de la maquette et que nous ne nous sommes mis à coder en Java trop tard. Nous espérons voir fonctionner un jour le LedWall fonctionner et mettre en place nos animations.

ANNEXE A

Liens utiles

Voici une petite liste d'url intéressantes au sujet de ce projet (cette liste peut également faire office de bibliographie) :

- https://tutoarduino.com/apprendre/
- https://www.youtube.com/watch?v=FljqF3ZPI_U
- http://www.arduino.cc/en/Tutorial/Blink?from=Tutorial.BlinkingLED
- http://www.arduino.cc/

ANNEXE B

Fiche de suivi de projet PeiP

Attention, cette fiche doit être complète à la fin du projet. Les comptes-rendu des 2 premières séances sont ici donnés à titre d'exemple.

Séance n° 1 du 29/01/2015	Durant cette première séance, nous avons fait des recherches à propos de la carte Arduino et nous avons commencer à chercher des idées d'animations.

Séance n° 2 du 05/02/2015	Lors de la séance du 08/02/2015 nous nous sommes renseignés sur le fonctionnement de la carte Arduino. Nous avons concentré nos recherches sur le code, la programmation en langage Arduino car c'est ce qui nous intéresse dans notre projet. La séance précédente, nous avions trouvé une idée d'animation qui nous plaisait et qui paraissait réalisable : « l'equalizer ». Nous avons donc fait une première ébauche du code en Arduino permettant au LedWall de réaliser cette animation.

Séance n° 3 du 12/02/2015	Nous avons commencé par observer et par chercher des solutions aux suggestions que vous aviez apportées à notre premier programme sur "l'equalizer1.0", ces suggestions ont engendré : - Améliorations sur le programme de base : (1h15) Jusqu'à 20 couleurs peuvent être affichées, il y a deux fonctions "equalizer" qui gèrent le temps total de l'animation, il y a un maximum qui reste en suspension entre deux animations. - Création d'un nouveau programme : EqualizerCouleurAleatoire (45 mins) Cet "equalizer" permet l'affichage d'une centaine de couleurs différentes changeantes entre chaque affichage. (+intégrations des fonctions et du maximum en suspention du programme de base). - Création d'un nouveau programme : Smiley (30 mins) Un programme d'affichage simple, il consiste en l'affichage d'un "Smiley", il cligne des yeux après un nombre aléatoire de secondes. - Idées d'animations : (25 mins) Texte de bienvenue, barres horrizontales (une ligne sur deux) qui se croisent, horloge (es-ce possible de la synchroniser en temps réel ?)...

Séance n° 4 du 19/02/2015	Lors de la séance de 3 heures du 19/02/2015 nous nous sommes rendu compte en essayant de compiler nos 3 programmes avec le logiciel de code Arduino qu'il y avait beaucoup d'erreurs, nous avons donc passer une majeure partie de la séance à essayer de résoudre ces erreurs de code. A la fin de la séance, une fois que nous y sommes parvenus, nous voulions tester nos animations sur le LedWall mais il nous manquait le câble pour connecter notre PC à la carte Arduino.

Séance n° 5 du 19/03/2015	Pour cette séance nous avions décidé de corriger certaines de nos animations afin de les rendre visionable sur le ledwall. Nous avons donc adapté l'animation des lignes horizontales pour commencer mais malheuresement lorsque nous sommes deçendu en bas pour tester l'animation, nous n'avons pas réussi à la faire s'afficher sur le ledwall. Nous avons donc essayé plusieurs cables, et testé différents programmes, mais nous n'avons pas été en mesure de fixer le probleme.

Séance n° 6 du 26/03/2015	Le LedWall ne fonctionnant pas, nous avions pour objectif dans cette séance de tester nos animations sur la maquette 5*5.Nous avons commencer à travailler sur nos animations pour faire en sorte qu'elles soient adaptées à la matrice 5*5. Malheureusement et malgré l'aide d'un étudiant en 5ème année nous n'avons pas réussi à faire fonctionner la maquette.

Séance n° 7 du 02/04/2015	Pour cette séance, nous nous sommes consacré à modifier nos codes arduinos pour les adapter à la maquette 5x5. De plus, la fin de notre projet de première année approchant à grand pas, nous avons décidé de commencer la redaction du compte rendu final.

Création d'animations pour le LedWall

Rapport de projet S2

Résumé : Le but de notre projet était de réaliser des animations pour le LedWall. Ces animations devaient être codées en Arduino, un langage de programmation proche du C. A cause de quelques problèmes techniques, nous n'avons pas pu faire fonctionner nos animations sur le mur, nous les avons donc retranscrit en Java car un outil nous permettait d'avoir une vision concrète de celles-ci

Mots clé : LedWall, Animations, Arduino, Java

Abstract : The aim of our project was to make animations for LEDwall. These animations were to be coded Arduino, a programming language similar to C. Because of some technical problems, we have not been able to run our animation on the wall, so we have transcribed in Java as a tool allowed us to having a specific vision thereof

Keywords : LedWall, Animations, Arduino, Java

Auteur(s)

Alix Bouguereau
[alix.bouguereau@gmail.com]
Samuel Bamba
[smail.bamba@gmail.com]

Encadrant(s)

Carl Esswein
[carl.esswein@univ-tours.fr]

Polytech Tours
Département DI

Ce document a été formaté selon le format EPUProjetPeiP.cls (N. Monmarché)

École Polytechnique de l'Université de Tours
64 Avenue Jean Portalis, 37200 Tours, France
http://www.polytech.univ-tours.fr