

Projet C++

Enseigne LED pilotable en DMX

Margout Camille, Delhay Emeline

10/05/2022



Abstract

Ce bureau d'étude avait pour but la création d'un objet connecté mêlant actionneurs et capteurs. Ce projet devait répondre à un besoin et nous nous sommes donc penchées sur la création d'une enseigne LED commandable avec une table de mixage, pour une utilisation en concert.

Contents

1 Présentation du Projet - Enseigne Enfoiros	3
1.1 Contexte d'élaboration du projet	3
1.1.1 Les Enfoiros de l'INSA - Late Night Show by Enfoiros	3
1.1.2 Cahier des charges imposé pour réaliser une enseigne lumineuse pilotable en DMX	3
1.1.2.1 Qu'est ce que la communication DMX ?	3
1.1.2.2 L'enjeu du projet : pilotage de l'enseigne depuis la régie	3
2 Caractéristiques hardware et software du projet	4
2.1 Caractéristiques Hardware	4
2.2 Notre Microcontrôleur : l'ESP8266	4
2.2.1 Alimentation	4
2.2.2 La gestion de la PWM générée par l'ESP8266	4
2.2.3 Gestion du DMX	5
2.2.4 Comment fonctionne le projet	5
2.3 Caractéristiques Software	6
2.3.1 Environnement de développement de notre projet	6
2.3.2 Diagramme de Classes du projet	6
2.3.3 Main et vue globale	7
2.3.3.1 Présentation de la classe DMX	7
2.3.3.2 Présentation de la classe DMX Lettres	7
2.3.3.3 Présentation de la classe DMX Tour	8
2.3.3.4 Présentation de la classe ANIM	8
3 Retours sur le projet	10
3.1 Les apports positifs de ce projet	10
3.1.1 Gestion du projet global	10
3.1.2 Gestion du temps	10
3.1.3 Première étape : fournir une version fonctionnelle en C pour les Concerts .	10
3.1.4 Deuxième étape : transformation du code en C++	11
3.2 Difficultés rencontrées sur le projet	12
Conclusion	14
A Annexes	15
A.1 Annexe 1 - Main	15
A.2 Annexe 2 - Classe DMX	18
A.2.1 DMX.h	18
A.2.2 DMX.cpp	18
A.3 Annexe 3 - Classe DMX Lettres	20
A.3.1 DMX-Lettres.h	20
A.3.2 DMX-Lettres.cpp	20
A.4 Annexe 4 - Classe DMX Tour	22
A.4.1 DMX-Tour.h	22
A.4.2 DMX-Tour.cpp	22
A.5 Annexe 5 - Classe Animations	24
A.5.1 ANIM.h	24
A.5.2 ANIM.cpp	25

1 Présentation du Projet - Enseigne Enfoiros

1.1 Contexte d'élaboration du projet

1.1.1 Les Enfoiros de l'INSA - Late Night Show by Enfoiros

Chaque année les Enfoiros de l'INSA, association étudiante qui par le biais de concerts et d'événements récolte des fonds pour les Restos du Coeur, organise une édition originale de spectacles. Cette année le thème est "Late Night Show by Enfoiros". En tant que régisseur général du spectacle et avec une motivation élevée pour des projets d'électronique, l'idée est venue de réaliser une enseigne lumineuse à destination des concerts des Enfoiros.

1.1.2 Cahier des charges imposé pour réaliser une enseigne lumineuse pilotable en DMX

Lors de nos concerts, nous utilisons des tables de régie son et lumière afin de commander un parc lumineux et sonore assez important. Pour que notre projet puisse s'insérer dans ces installations, il fallait répondre à un critère essentiel du cahier des charges : la communication DMX.

1.1.2.1 Qu'est ce que la communication DMX ?

DMX signifie Digital MultipleXing. Il a été conçu pour normaliser les communications entre une console et les différents éléments qu'elle est susceptible de piloter. Ici, il s'agit de notre Enseigne lumineuse.



Figure 1: Cable DMX 3 points Male/Femelle

Le DMX est un protocole de communication basé sur un bus numérique utilisant 3 fils (2 pour le signal qui est symétrisé + la masse). En fait, il se contente de transmettre des données (des nombres compris entre 0 et 255).

1.1.2.2 L'enjeu du projet : pilotage de l'enseigne depuis la régie

L'enjeu était alors d'utiliser ce signal pour activer des fonctionnalités **automatiques** : les animations, ou bien de piloter l'éclairage de l'enseigne **manuellement** : le pilotage RGB en intensité sans animation.

2 Caractéristiques hardware et software du projet

2.1 Caractéristiques Hardware

2.2 Notre Microcontrôleur : l'ESP8266

Pour notre projet, nous avions à disposition un Node MCU ESP8266 contenant un processeur 32 bit. Le microcontrôleur possède une entrée analogique et 16 GPIO (11 Digital I/O) disponibles sur les broches de la carte de développement. Sur les 25 broches on compte :

- 1 entrée analogique
- 4 sorties PWM
- Certaines réservées pour les protocoles de communication série (SPI, I2C, Serial)

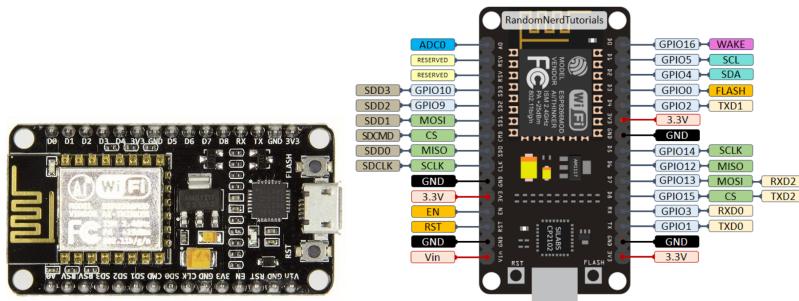


Figure 2: Module ESP8266

2.2.1 Alimentation

Nous avons utilisé deux types de ruban LED. Nous avons un ruban RGB adressable et un ruban led classique (pilotable uniquement en couleur et en intensité sur la totalité du bandeau, pas d'adressage LED par LED).

Chaque ruban requiert une alimentation différente en voltage et en ampérage. Le ruban programmable du tour est relié à une alimentation DCDC 5V 10A et le ruban des lettres RGB est relié à une alimentation DCDC 12V 8A. Pour la suite du rapport, le ruban adressable sera celui utilisé sur le tour de l'enseigne et le ruban RGB sera celui utilisé pour former les lettres de l'enseigne.

La gestion de chaque ruban est différente car le ruban RGB comporte 4 lignes : un pin d'alimentation 12V, un PIN Rouge, un PIN Verte, un PIN Bleu, tandis que le ruban adressable comporte uniquement 3 lignes : un PIN de data, un PIN d'alimentation 5V, et un GND.

2.2.2 La gestion de la PWM générée par l'ESP8266

Afin de piloter les rubans LED RGB en intensité, nous avons utilisé 3 PIN PWM donnés par l'ESP8266. Nous avons ajouté un étage de puissance car la PWM en sortie de l'ESP n'était pas suffisante pour nos centaines de LEDs sur le ruban. Nous avons donc réalisé un PCB au GEI composé de transistors et de MOSFETs avec une ligne par couleur.

2.2.3 Gestion du DMX

Il existe différentes tables de mixage avec différents nombres de canaux. Un câble DMX peut supporter jusqu'à 512 canaux à lui seul. Pour notre projet, nous avions besoin de 8 canaux appelés aussi fader.

L'adressage consiste à affecter chaque appareil d'éclairage à un certain nombre de canaux, sur les 512 que compte un câble DMX. Notre enseigne fonctionne sur 8 canaux. Concrètement, l'adressage consiste à assigner à notre appareil le numéro de son premier canal. Nous l'avons assignée au canal 1, elle utilisera donc les canaux 1 à 8.

2.2.4 Comment fonctionne le projet

Les canaux 1, 2 et 3 de la table de mixage permettent de gérer respectivement la couleur Rouge, Verte et Bleu des Lettres. Le Canal 4 est le canal Animation. Nous avons mappé 5 types d'animations selon un découpage proportionnel entre 0 et 255 (qui correspondent aux valeurs min et max du fader).

Si le canal Animation est placé entre 0 et 50, nous pouvons allumer le ruban du tour avec les canaux 6, 7 et 8 respectivement pour envoyer du Rouge, Vert et Bleu.

Si on positionne le fader 4 entre 51 et 101, on anime un chargement (allumage progressif). On peut modifier la vitesse du chargement avec le canal 5.

Nous avons également le "chase" lorsque le fader 4 est positionné entre 102 et 152 qui consiste à avoir le ruban totalement allumé à l'exception de quelques leds périodiquement éteintes. Ces leds éteintes changent et donnent l'impression d'un mouvement. Egalement, la vitesse est contrôlable avec le fader 5.

L'animation Tour Eiffel se déclenche lorsque le fader est positionné entre 153 et 204. Nous pouvons régler la couleur et la vitesse de clignotement. L'animation donne l'impression que le ruban du tour "brille".

Enfin, l'animation rainbow, déclenchée lorsque le fader est positionné entre 205 et 255, affiche un arc-en-ciel qu'il est possible d'animer en vitesse avec le canal 5. On a l'impression que les couleurs se déplacent plus ou moins vite selon la valeur du potentiomètre du fader vitesse.

Pour la démonstration, nous avons utilisé une table plus simple que celle des concerts.



Figure 3: Table de mixage DMX utilisée pour la démonstration

2.3 Caractéristiques Software

2.3.1 Environnement de développement de notre projet

Nous avons utilisé l'IDE Arduino pour développer notre projet, après avoir téléchargé l'environnement de développement, il faut installer le module ESP8266 via la commande Tools / Board / ESP8266 Boards / Generic ESP8266 Mod.

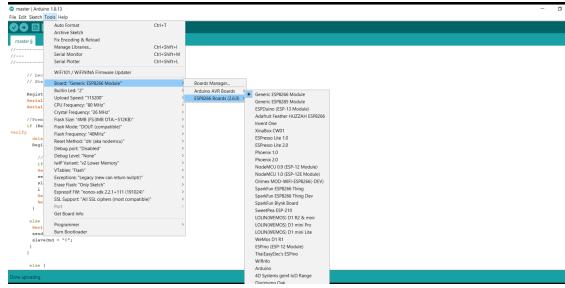


Figure 4: Installation du module ESP8266

2.3.2 Diagramme de Classes du projet

Nous avons décidé de découper notre programme en quatre classes, séparant les différents types de LEDs, ainsi que les animations.

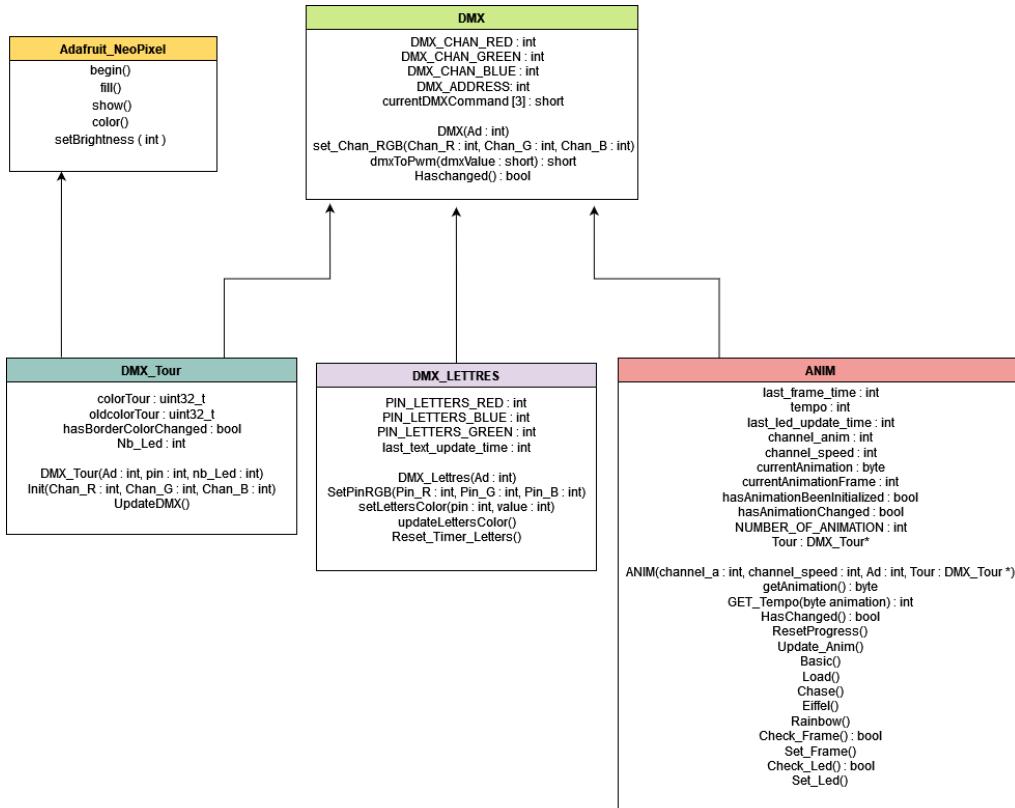


Figure 5: Diagramme de classe de l'enseigne

2.3.3 Main et vue globale

2.3.3.1 Présentation de la classe DMX

La classe DMX est la classe mère de toutes les autres.

Variables	
DMX_CHAN_RED	Numéro du canal DMX dédié aux instructions pour la couleur rouge
DMX_CHAN_GREEN	Numéro du canal DMX dédié aux instructions pour la couleur verte
DMX_CHAN_BLUE	Numéro du canal DMX dédié aux instructions pour la couleur bleue
DMX_ADDRESS	Adresse de la table DMX
CurrentDMXCommand[3]	Tableau pour stocker jusqu'à 3 instructions DMX

C'est à partir de l'adresse de la table qu'on obtient les 'vraies' valeurs de canaux. En effet, Le canal rouge se trouve par exemple à la valeur DMX_ADDRESS + DMX_CHAN_RED.

Son constructeur prend en argument l'adresse de la table et l'enregistre.

La fonction Set_Chan_RGB permet de régler les canaux sur lesquels on enverra les données pour les couleurs.

La fonction dmxToPwm permet quant à elle de transformer une instruction DMX comprise entre 0 et 255, en PWM.

Haschanged parcourt le tableau d'instructions et regarde si l'instruction DMX que l'ESP reçoit en ce moment correspond. Si il y a un changement, il modifie la valeur de la case du tableau et renvoie vrai. Sinon, la fonction renvoie faux.

2.3.3.2 Présentation de la classe DMX Lettres

La classe DMX_Lettres hérite de la classe DMX. Elle possède trois attributs en plus de ceux définis dans la classe DMX.

Variables	
PIN LETTERS_RED	PIN de l'ESP utilisé pour envoyer les valeurs de la couleur rouge sur les lettres
PIN LETTERS_GREEN	PIN de l'ESP utilisé pour envoyer les valeurs de la couleur verte sur les lettres
PIN LETTERS_BLUE	PIN de l'ESP utilisé pour envoyer les valeurs de la couleur bleue sur les lettres
last_text_update_time	Timer d'expiration des LED lettres

Le constructeur de cette classe fait seulement appel à celui de sa classe mère, pour régler la valeur de l'adresse DMX.

La fonction SetPinRGB enregistre les valeurs des PINs de l'ESP connectés au bandeau LED, et les configure en output.

SetLettersColor permet d'envoyer sur un pin une valeur correspondant dans notre cas à la couleur voulue entre 0 et 255.

updateLettersColor regroupe plusieurs fonctions. Tout d'abord, cette fonction convertit les 3 cases du tableau d'instructions DMX en PWM grâce à la fonction dmxToPwm décrite plus haut et récupère les valeurs renvoyées. Puis, elle fait trois fois appel à la fonction SetLettersColor avec en paramètre les pins R,G,B et les valeurs récupérées.

Enfin, Reset_Timer_Letters permet de remettre à zéro le timer des lettres lorsqu'il y a un changement d'instruction.

2.3.3.3 Présentation de la classe DMX Tour

Cette classe a elle aussi trois attributs qui lui sont propres.

Variables	
colorTour	Stocke la valeur de la couleur actuelle
oldcolorTour	Stocke la valeur de la couleur précédente
HasBorderColorChanged	Compare les nouvelles et anciennes valeurs de couleur
Nb_Led	Correspond au nombre de LED sur le tour

Le constructeur de cette classe fait appel au constructeur de DMX, mais aussi à celui de la librairie Adafruit_Neopixel, qui crée un objet correspondant à notre Tour, avec son nombre de LED et son PIN data. Il met également à true la variable hasBorderColorChanged et initialise le nombre de LED sur le tour.

La fonction Init utilise set_Chan_RGB pour paramétrier les canaux DMX de contrôle de couleur et lance une séquence d'allumage sur les LED du tour.

UpdateDMX permet de mettre à jour la variable colorTour. Elle vérifie ensuite que colorTour est différent de oldcolorTour, si c'est le cas, hasBorderColorChanged passe à True.

2.3.3.4 Présentation de la classe ANIM

La classe ANIM est une classe particulière. Elle hérite de la classe DMX et possède de nombreux attributs.

Variables	
last_led_update_time	Timer servant à rafraîchir les LED du tour
last_frame_time	Timer servant à rafraîchir l'étape de l'animation
tempo	Durée d'une étape d'une animation
channel_anim	Canal DMX associé au numéro de l'animation
channel_speed	Canal DMX associé à la vitesse de l'animation
currentAnimation	Numéro de l'animation en cours
currentAnimationFrame	Etape en cours de l'animation
NUMBER_OF_ANIMATION	nombre d'animations totales
hasAnimationBeenInitialized	Vrai lorsque l'animation a commencé
hasAnimationChanged	Vrai lorsqu'on change l'animation voulue
Tour	Correspond à un objet Tour, sur lequel on met en place les animations

channel_anim et channel_speed 'remplacent' les attributs DMX_CHAN_... de la classe DMX car pour les animation il n'y a pas de canaux liés aux couleurs, mais seulement deux canaux pour le choix de l'animation et sa vitesse.

ANIM réutilise également l'attribut currentDMXCommand de sa classe mère, mais elle n'utilise que les deux premières cases du tableau, qui correspondront aux instructions d'animation et de vitesse.

Son constructeur initialise les paramètres DMX ainsi que toutes les variables précédentes. Il prend également en argument l'objet Tour que l'on souhaite animer.

La fonction getAnimation lit l'instruction DMX stockée dans le tableau currentDMXCommand[0] (instruction d'animation) et renvoie le numéro de l'animation correspondante.

GET_Tempo permet d'obtenir la valeur du timer d'animation en fonction du numéro de l'animation. En effet, en fonction de l'animation, le timer varie.

Dans cette classe, on redéfinit la fonction HasChanged pour pouvoir comparer les instructions sur les canaux d'animations et de vitesse.

La fonction ResetProgress remet à zéro la progression de l'animation, et permet un retour à la frame 0.

Update_Anim met à jour l'animation lorsqu'il y a un changement d'instruction DMX. Pour cela, elle utilise la fonction ResetProgress.

Les six fonctions d'animation, Basic, Load, Chase, Eiffel et Rainbow, utilisent l'objet Tour pour modifier les couleurs des LED, et ainsi créer des animations différentes.

Les deux fonctions set_Frame et Led assignent aux timers le temps en millisecondes depuis le lancement du programme.

Les fonctions Check vérifient quant à elles que le runtime - timer < 0 pour le tour et < tempo pour les animations, et donc que le timer n'a pas expiré.

3 Retours sur le projet

3.1 Les apports positifs de ce projet

3.1.1 Gestion du projet global

Ce projet nous a demandé beaucoup de temps et d'investissement, mais il a été très positif dans son ensemble. J'avais déjà travaillé avec des ESP8266 lors de mon stage de 3A en désignant une alarme WIFI. C'était donc plus rapide car je connaissais très bien l'IDE et le module tant à travers des projets professionnels que personnels. C'est ce qui a motivé une prise de risque plus grande au niveau Hardware.

3.1.2 Gestion du temps

3.1.3 Première étape : fournir une version fonctionnelle en C pour les Concerts

Ce projet avait une deadline légèrement différente du rendu. Nous devions avoir terminé l'enseigne pour les montages soit pour le 1er avril. Il fallait, en parallèle de tout réaliser au niveau hardware, pensez au software, à l'architecture du code, ses fonctionnalités.



Figure 6: Découpage du panneau de bois et assemblage au FabLab

Nous avons en premier pensé au **support**. Il fallait que l'enseigne soit grande pour être visible et reste dans le thème du spectacle. Nous avons choisi de reproduire le logo associé à l'édition. Au niveau des dimensions, nous avons utilisé du bois que nous avons découpé pour obtenir un cercle de 1m50 de diamètre et d'environ 15kg.



Figure 7: Atelier Soudure au GEI

Une fois le support assemblé et peint, nous avons disposé le ruban programmable qui était le plus simple à faire fonctionner. Nous avons immédiatement pu construire les animations en fonction du show light que nous souhaitions faire. Mais tout ce travail était très long et nous passions de longues soirées à souder et raccorder les petites portions de bandeau découpées.



Figure 8: Tests primitifs des Lettres

3.1.4 Deuxième étape : transformation du code en C++

Cette étape était la plus compliquée à dimensionner. Nous ne savions pas à quel point nos transformations en classes étaient ou non périlleuses pour le code. Nous avons finalement réussi à tout faire fonctionner. La bonne surprise a été que nous avions moins d'artefacts avec la version C++ (soit une version plus stable avec un hardware inchangé).

3.2 Difficultées rencontrées sur le projet

Nous avons du attendre 15 jours de plus car les rubans RGB ne sont jamais arrivés. Nous avons eu un don de 3 rubans RGB et nous avons pu continuer ce lourd travail de soudure (environ 3h par soir). Ce nouveau ruban étant de mauvaise qualité, notre travail devait être encore plus minutieux et demandait plus de temps.



Figure 9: Ensemble des lettres cablées et soudées

Quelques nuits blanches se sont passées et nous avons sorti la version du PCB global qui allait supporter le driver DMX, l'ESP ainsi que les borniers d'alimentation. A l'époque, nous utilisions également un driver pour le ruban RGB. Mais ce module était une boîte noire et nous avons décidé de driver les LEDs nous même en insérant un deuxième PCB. Vu que nous avancions au rythme du hardware, il fallait constamment s'adapter et ne rien lâcher en parallèle des cours, de la semaine de concerts et des rendus des autres matières.

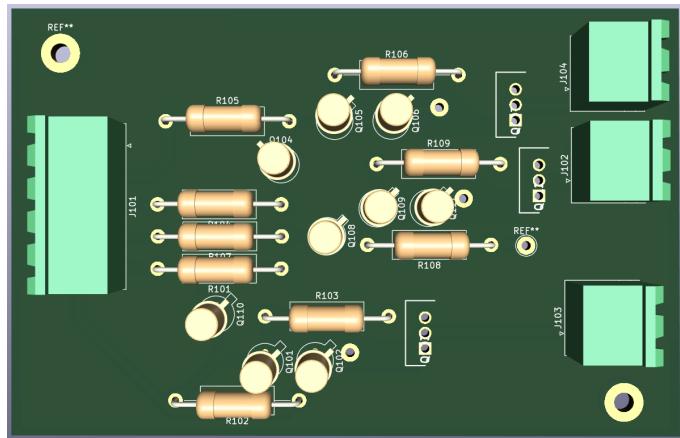


Figure 10: Driver PWM

Nous voulions également afficher sur un 7 segments l'adresse de l'enseigne et pouvoir la modifier avec des boutons. Le soucis c'est que les boutons étaient positionnés sur des PIN PWM dont l'ESP n'était pas riche. Il a fallut supprimer les boutons au profit des PIN PWM.

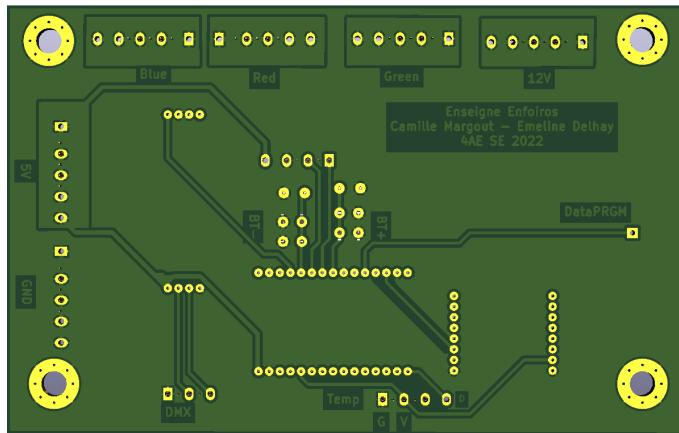


Figure 11: PCB Original avec les boutons initialement prévus

Une autre difficulté a été de dépendre d'un objet lourd, volumineux et stocké dans une salle du GEI. Toutes les semaines, nous devions déclarer notre présence pour avoir accès à la salle le soir et les weekend.

Enfin, nous avons eu de grosses frayeurs car avec un amperage élevé et des tensions de 5 et 12V, nous devions sans cesse prendre de grosses précautions. Lorsque nous avons travaillé sur l'enseigne en amphi, pendant les montages, nous avions beaucoup de passages avec des gens curieux ou volontaires pour nous aider. Malheureusement, en étant moins concentrées nous avons fait des erreurs qui ont coûté la vie de quelques composants et de nos heures de sommeil.

Pour l'anecdote, une soir, tout fonctionnait et il ne manquait plus qu'à faire 4 trous pour ajouter les fixations de l'enseigne. Avec l'euphorie et le manque de sommeil, j'étais encore entrain de tester quelque chose sur les bandeau (alimentation allumée donc), et quelqu'un est venu m'apporter une grosse règle en fer et l'a posée sur les rubans. Cela a fait de beaux éclairs et on a du changer toutes les LEDs brûlées du ruban externe. Nous avions de la marge mais, à la fin de la réparation, nous n'avions qu'une LED de back up. Finalement, à 5h37 du matin, l'enseigne était fixée et fonctionnelle.

Conclusion

Avec quelques semaines de recule, le projet a vraiment été une réussite. Il est d'une part fonctionnel et présente de meilleures performances avec le code en C++. Nous avons pu revoir le code et améliorer certaines fonctions. Nous sommes toujours limitées à la capacité du microcontrôleur à driver du PWM pour un bandeau tout en gérant la PIN Data de l'autre. Cela donne lieu à des petits artefacts de lumières mais qui ne gènent pas le projet au global.

Ce projet nous aura permis de tester nos limites de patience, de persévérance et d'adaptation. Nous devions sans arrêt nous remettre en question, faire des choix pour avancer et ce n'était pas facile. En voyant le projet là où il en est aujourd'hui, nous n'en tirons que du positif.

Ce projet nous aura également donné la chance de travailler tant sur du hardware que sur du software. Nous voulions cette double dimension et nous l'avons eue. Nous avons fait quelque chose d'exotique : transformer un code en C en C++ et ça nous a demandé un peu de réflexion mais c'était très intéressant de tester cette façon de coder. Au début nous avions l'impression de faire des classes pour des classes mais nous avons gagné en visibilité dans le code et en performance car nous avons revu les fonctions associées.



Figure 12: Notre projet sur la scène des Enfoirés

A Annexes

A.1 Annexe 1 - Main

```
1 /*****
2     Enseigne LATE SHOW
3         ENFOIROS X GEI
4         NodeMCU ESP8266
5             2022
6 *****/
7
8 #define VERSION "2-02"
9
10 // Ajout des biblioth ques
11 #include <Arduino.h>
12 #include <Adafruit_NeoPixel.h>      // Leds cercle
13 #include <LXESP8266UARTDMX.h>        // DMX  (https://github.com/claudheintz/LXESP8266DMX)
14 #include <TM1637.h>                  // 7 segments (Grove 4-Digit Display)
15
16 #include "DMX_Lettres.h"
17 #include "DMX_Tour.h"
18 #include "ANIM.h"
19
20
21 // D finition des PIN utilis s
22 #define PIN_BORDER 2 // => D4 DATA du tour
23 #define PIN_7SEG_CLK 12 // => D6
24 #define PIN_7SEG_DIO 13 // => D7
25 #define PIN_LETTERS_RED 4 // => D2
26 #define PIN_LETTERS_GREEN 15 // => D8
27 #define PIN_LETTERS_BLUE 14 // => D5
28
29 /* PATCH DMX
30     Canal 1 -> Rouge Lettres
31     Canal 2 -> Vert Lettres
32     Canal 3 -> Bleu Lettres
33     Canal 4 -> Choix animation Tour
34     Canal 5 -> Vitesse animation Tour
35     Canal 6 -> Rouge Tour
36     Canal 7 -> Vert Tour
37     Canal 8 -> Bleu Tour
38 */
39
40 //D finition des canaux DMX
41 #define DMX_ADDRESS 1
42
43 #define DMX_CHANNEL_LETTERS_RED 0
44 #define DMX_CHANNEL_LETTERS_GREEN 1
45 #define DMX_CHANNEL_LETTERS_BLUE 2
46 #define DMX_CHANNEL_ANIMATION_SELECT 3
47 #define DMX_CHANNEL_ANIMATION_SPEED 4
48 #define DMX_CHANNEL_BORDER_RED 5
49 #define DMX_CHANNEL_BORDER_GREEN 6
50 #define DMX_CHANNEL_BORDER_BLUE 7
51
52 //Variables propres notre enseigne
53 #define NUMBER_OF_BORDER_LEDS 124
54 #define NUMBER_OF_ANIMATIONS 5
55
56
57 // Permet d'afficher 'Err' sur l'ecran
58 /*const*/ int8_t err[] = { 0x0, 0x79, 0x50, 0x50 };
59
60 // D finition des objets
61 DMX_Tour Tour(PIN_BORDER, NUMBER_OF_BORDER_LEDS) ; // Objet Tour
```

```

62 DMX_Lettres Lettres (DMX_ADDRESS) ;
63     // Objet Lettres
64 ANIM Animations(DMX_CHANNEL_ANIMATION_SELECT, DMX_CHANNEL_ANIMATION_SPEED, &Tour) ;
65     // objet Animation
66 TM1637 screen(PIN_7SEG_CLK, PIN_7SEG_DIO);
67     // Objet cran
68
69
70 unsigned long lastDMXTime;           // Permet la detection de perte de DMX
71 unsigned long timeoutEcran;         // Permet d'eteindre l'cran
72 boolean isEcranOFF;
73 int debug = 0 ;                     // Variable aide au debug
74
75 void updateDMXTimer(int slots);
76
77 void setup() {
78     // Watchdog disable
79     ESP.wdtDisable();
80     //Remove watchdog from WiFi chip
81     *((volatile uint32_t*) 0x60000900) &= ~(1);
82
83     // Initialization des LEDs du tour
84     Tour.Init(DMX_CHANNEL_BORDER_RED, DMX_CHANNEL_BORDER_GREEN,
85             DMX_CHANNEL_BORDER_BLUE) ;
86
87     // Initialisation des LEDs des lettres
88     Lettres.Set_Chan_RGB(DMX_CHANNEL_LETTERS_RED, DMX_CHANNEL_LETTERS_GREEN,
89                         DMX_CHANNEL_LETTERS_BLUE) ;
90     Lettres.SetPinRGB(PIN_LETTERS_RED, PIN_LETTERS_GREEN, PIN_LETTERS_BLUE) ;
91
92     // Initialisation cran
93     screen.init();
94     screen.set(3); // Luminosit faible
95     //screen.displayStr(VERSION);
96     //screen.displayNum(DMX_ADDRESS);
97     isEcranOFF = false;
98     timeoutEcran = millis();
99
100 }
101
102
103 // Routine lanc e chaque reception de trame DMX. Permet de mettre jour le
104 // timer de detection de perte de DMX
105 void updateDMXTimer(int slots) {
106
107     // Si il y a eu un changement d'instruction pour les lettres, met jour la
108     // variable servant de timer
109     Lettres.Reset_Timer_Letters();
110
111     // Si il y a eu un changement d'instruction pour le tour ou les animations, met
112     // jour la variable servant de timer
113     if (Tour.HasChanged() or Animations.HasChanged()) {
114         Animations.Set_Led() ;
115     }
116
117     // Mise jour des lettres
118     Lettres.updateLettersColor();
119
120     // Mise jour du tour et animations si le timer s'est coul
121     if (Animations.Check_Led()) {
122         Tour.UpdateDMX() ;
123     }

```

```

120     Animations.Update_Anim();
121 }
122
123 //Si la Frame d'animation actuelle est passe, on passe la frame suivante
124 if (Animations.Check_Frame()) {
125     switch(Animations.GET_currentAnimation()) {
126         // Pas d'anim
127         case 0:
128             Animations.Basic();
129             break;
130         // Chargement
131         case 1:
132             Animations.Load();
133             break;
134         // Chase
135         case 2:
136             Animations.Chase();
137             break;
138         // Tour eiffel
139         case 3:
140             Animations.Eiffel();
141             break;
142         // Rainbow
143         case 4:
144             Animations.Rainbow();
145             break;
146     }
147     Animations.Set_Frame();
148 }
149 }
150
151
152 void loop() {
153     yield();
154     if (Animations.GET_hasAnimationChanged()) {
155         Tour.show();
156         Animations.SET_hasAnimationChanged(false);
157     }
158 }
159 }
```

A.2 Annexe 2 - Classe DMX

A.2.1 DMX.h

```
1 #ifndef DMX_H
2 #define DMX_H
3
4 #include <LXESP8266UARTDMX.h>
5
6 class DMX
7 {
8     private :
9         int DMX_CHAN_RED;
10        int DMX_CHAN_BLUE;
11        int DMX_CHAN_GREEN;
12
13    protected :
14        static int DMX_ADDRESS;
15        // Instructions du DMX
16        short currentDMXCommand[3] ;
17
18    public:
19        // Le constructeur initialise l'adresse de la table DMX avec le param tre d'
20        entr e
21        DMX(int Ad);
22        DMX();
23
24        // Initialise les channel DMX d di s au instructions de couleurs
25        void Set_Chan_RGB(int Chan_R, int Chan_G, int Chan_B) ;
26
27        //Transformation d'un signal DMX en PWM
28        short dmxToPwm(short dmxValue) ;
29
30        // V rifie si les instructions ont chang et les met jour si c'est le cas
31        bool HasChanged();
32
33 #endif // DMX_H
```

A.2.2 DMX.cpp

```
1
2 #include "DMX.h"
3
4 DMX::DMX(int Ad){
5     DMX_ADDRESS = Ad;
6 }
7
8 DMX::DMX(){}
9
10 void DMX::Set_Chan_RGB(int Chan_R, int Chan_G, int Chan_B){
11     DMX_CHAN_RED = Chan_R;
12     DMX_CHAN_GREEN = Chan_G;
13     DMX_CHAN_BLUE = Chan_B;
14 }
15
16 short DMX::dmxToPwm(short dmxValue) {
17     return 1024 - map(dmxValue, 0, 255, 1, 1023);
18 }
19
20 bool DMX::HasChanged(){
21     boolean flag = false;
22
23     //On boucle de 0 2 pour parcourir le tableau currentDMXCommand, qu'on met
24     // jour
25     // On observe le DMX sur le canal DMX_ADDRESS + i + DMX_CHAN_RED car on a choisi
26     // de positionner les canaux RGB les un la suite de l'autre
```

```
25     for (size_t i = 0; i < 3; i++) {
26         if (currentDMXCommand[i] != ESP8266DMX.getSlot(DMX_ADDRESS + i + DMX_CHAN_RED))
27         {
28             currentDMXCommand[i] = ESP8266DMX.getSlot(DMX_ADDRESS + i + DMX_CHAN_RED);
29             flag = true;
30         }
31     }
32 }
33
34 int DMX::DMX_ADDRESS = 0;
```

A.3 Annexe 3 - Classe DMX Lettres

A.3.1 DMX-Lettres.h

```
1 #ifndef DMX_LETTRES_H
2 #define DMX_LETTRES_H
3
4 #include "DMX.h"
5
6
7 class DMX_Lettres : public DMX
8 {
9     private:
10         // Les LEDs des lettres ont trois PIN de data RGB
11         int PIN LETTERS_RED;
12         int PIN LETTERS_BLUE;
13         int PIN LETTERS_GREEN;
14         int last_text_update_time;
15
16     public:
17         //Le constructeur fait appel au constructeur de DMX
18         DMX_Lettres(int Ad);
19
20         //Initialise les PIN de donnees
21         void SetPinRGB(int Pin_R, int Pin_G, int Pin_B);
22
23         //Ecrit sur un pin une valeur de 0      255 correspondant    la couleur
24         void setLettersColor(int pin, int value) ;
25
26         //Envoie les instructions DMX, transformees en PWM, sur les trois PIN de data
27         void updateLettersColor() ;
28
29         //Remet    a zero le timer des lettres si il y a eu un changement d'instruction
30         void Reset_Timer_Letters();
31 };
32
33 #endif // DMX_LETTRES_H
```

A.3.2 DMX-Lettres.cpp

```
1 #include <Arduino.h>
2 #include "DMX_Lettres.h"
3
4 DMX_Lettres::DMX_Lettres( int Ad):DMX(Ad){
5     last_text_update_time = 0;
6 }
7
8 void DMX_Lettres::SetPinRGB(int Pin_R, int Pin_G, int Pin_B){
9     PIN LETTERS_RED = Pin_R;
10    PIN LETTERS_BLUE = Pin_B;
11    PIN LETTERS_GREEN = Pin_G;
12
13    pinMode(PIN LETTERS_RED, OUTPUT);
14    pinMode(PIN LETTERS_GREEN, OUTPUT);
15    pinMode(PIN LETTERS_BLUE, OUTPUT);
16 }
17
18 void DMX_Lettres::setLettersColor(int pin, int value) {
19     analogWrite(pin, value);
20 }
21
22 void DMX_Lettres::updateLettersColor() {
23
24     if ((millis() - last_text_update_time) > 0) {
25         short lettersRed = dmxToPwm(currentDMXCommand[0]);
26         short lettersGreen = dmxToPwm(currentDMXCommand[1]);
27         short lettersBlue = dmxToPwm(currentDMXCommand[2]);
```

```
28     setLettersColor(PIN LETTERS_RED , lettersRed);
29     setLettersColor(PIN LETTERS_GREEN , lettersGreen);
30     setLettersColor(PIN LETTERS_BLUE , lettersBlue);
31   }
32 }
33 }
34
35 void DMX_Lettres::Reset_Timer_Letters() {
36   if (this->HasChanged()) {
37     last_text_update_time = millis();
38   }
39 }
```

A.4 Annexe 4 - Classe DMX Tour

A.4.1 DMX-Tour.h

```
1 #ifndef DMX_TOUR_H
2 #define DMX_TOUR_H
3
4 #include "DMX.h"
5 #include <Adafruit_NeoPixel.h>
6
7
8 class DMX_Tour : public DMX, public Adafruit_NeoPixel
9 {
10     private:
11     uint32_t colorTour;
12     uint32_t oldcolorTour;
13     bool hasBorderColorChanged;
14     int Nb_Led;
15
16     public:
17
18     //Fait appel au constructeur de DMX et de Adafruit_Neopixel, initialise la
19     //variable hasBorderColorChanged
20     DMX_Tour(int pin, int nb_Led);
21
22     //Initialise le bandeau avec les canaux DMX, et un sequence d'allumage
23     void Init(int Chan_R, int Chan_G, int Chan_B);
24
25     //Met jour la valeur de colorTour
26     void UpdateDMX();
27
28     uint32_t GET_colorTour() ;
29     bool GET_hasBorderColorChanged() ;
30     int GET_NbLed();
31 };
32
33 #endif // DMX_TOUR_H
```

A.4.2 DMX-Tour.cpp

```
1 #include "DMX_Tour.h"
2
3 DMX_Tour::DMX_Tour(int pin, int nb_Led):DMX(), Adafruit_NeoPixel(nb_Led, pin,
4     NEO_GRB + NEO_KHZ800){
5     hasBorderColorChanged = true ;
6     this->Nb_Led = nb_Led;
7 }
8
9 void DMX_Tour::Init(int Chan_R, int Chan_G, int Chan_B){
10
11     Set_Chan_RGB(Chan_R, Chan_G, Chan_B) ;
12     begin();
13     setBrightness(255); // Luminosit fond
14     clear();
15     show();
16 }
17
18 void DMX_Tour::UpdateDMX(){
19     colorTour = Color(currentDMXCommand[0], currentDMXCommand[1], currentDMXCommand
20     [2]);
21     hasBorderColorChanged = (colorTour != oldcolorTour);
22     oldcolorTour = colorTour;
23 }
24
25 uint32_t DMX_Tour::GET_colorTour(){
26     return colorTour ;
```

```
25 }
26
27 bool DMX_Tour::GET_hasBorderColorChanged(){
28     return hasBorderColorChanged ;
29 }
30
31 int DMX_Tour::GET_NbLed(){
32     return Nb_Led;
33 }
```

A.5 Annexe 5 - Classe Animations

A.5.1 ANIM.h

```
1 #ifndef ANIM_H
2 #define ANIM_H
3
4 #include <LXESP8266UARTDMX.h>
5 #include "DMX_Tour.h"
6
7 class ANIM : public DMX
8 {
9
10 private :
11
12     int last_frame_time;
13     int tempo ;
14
15     int last_led_update_time;
16
17     int channel_anim ;
18     int channel_speed ;
19
20     byte currentAnimation; // Numéro d'animation en cours
21     int currentAnimationFrame;// Etape en cours dans l'animation
22
23     bool hasAnimationBeenInitialized;
24     bool hasAnimationChanged;
25     int NUMBER_OF_ANIMATION ;
26
27     DMX_Tour* Tour;
28
29 public:
30
31     //Initialise les canaux d'animation et de vitesse, ainsi que l'adresse et relie
32     //la classe un Tour qui doit être animé
33     ANIM(int channel_a, int channel_speed, DMX_Tour * Tour);
34
35     //Renvoie le numéro de l'animation en fonction du signal sur le canal DMX
36     //du diod l'animation
37     byte getAnimation() ;
38
39     //Renvoie la valeur du timer associé à l'animation, en fonction de l'
40     //instruction de vitesse
41     int GET_tempo(byte animation) ;
42
43     //Remplace la fonction hasChanged de la classe DMX, pour vérifier les
44     //instructions sur les canaux d'animation et de vitesse
45     bool HasChanged() ;
46
47     //Remet à zéro la progression de l'animation
48     void ResetProgress() ;
49
50     //Mets à jour l'animation lorsqu'il y a un changement
51     void Update_Anim();
52
53     //Différentes animations possibles
54     void Basic(); //Couleurs classique
55     void Load(); //Effet de chargement
56     void Chase(); //Effet de poursuite
57     void Eiffel(); //Effet tour Eiffel avec des led qui clignotent
58     void Rainbow(); //Effet d'arc-en ciel qui bouge
59
60     //Vérifie que le timer associé au tape d'animation est expiré
61     bool Check_Frame() ;
62
63     //Remet à zéro le timer d'un tape
64 }
```

```

60 void Set_Frame() ;
61
62 // V rifie que le timer associ au Tour est expir
63 bool Check_Led() ;
64
65 //Remet z ro le timer Tour
66 void Set_Led() ;
67
68 void SET_hasAnimationChanged (bool state) ;
69 byte GET_currentAnimation() ;
70 bool GET_hasAnimationChanged() ;
71
72 };
73
74 #endif // ANIM_H

```

A.5.2 ANIM.cpp

```

1
2 #include "DMX.h"
3 #include "ANIM.h"
4
5 ANIM::ANIM (int channel_a, int channel_speed, DMX_Tour * Tour) : DMX(){
6
7     channel_anim = channel_a ;
8     channel_speed = channel_speed ;
9     this->Tour = Tour ;
10
11    last_led_update_time = 0;
12    last_frame_time = 0 ;
13    tempo = 0 ;
14
15    currentAnimation = 0 ;
16    currentAnimationFrame = 0 ;
17
18    hasAnimationBeenInitialized = false ;
19    hasAnimationChanged = true ;
20
21    NUMBER_OF_ANIMATION = 5 ;
22 }
23
24 //Dans le cas de la classe ANIM, currentDMXCommand[0] correspond au canal de l'
25 // animation et currentDMXCommand[1] au canal de vitesse de l'animation
26
27 byte ANIM::getAnimation(){
28     // 255 / Number_of_animation permet de cr er des intervalles correspondants aux
29     // diff rentes animations
30     //On r cup re ensuite la valeur envoyer sur le canal DMX et la compare au
31     // intervalles pour d terminer l'animation voulue
32     byte out = (byte) currentDMXCommand[0] / (255 / NUMBER_OF_ANIMATION);
33     if (out <= NUMBER_OF_ANIMATION) return out;
34
35
36
37 int ANIM::GET_tempo (byte animation) {
38     switch (animation) {
39         case 0:
40             return 1000;
41         case 1:
42             return map(currentDMXCommand[1], 0, 255, 2500, 250); // map between 0.5 and 5
43             seconds
44         case 2:
45             return 1010 - map(currentDMXCommand[1], 0, 255, 10, 1000); // map between
46             0.05 and 3 seconds

```

```

45     case 3:
46         return 1010 - map(currentDMXCommand[1], 0, 255, 10, 1000);
47     case 4:
48         return 1010 - map(currentDMXCommand[1], 0, 255, 10, 1000);
49     }
50 }
51
52 bool ANIM::HasChanged(){
53     boolean flag = false;
54     for (size_t i = 0; i < 2 ; i++) {
55         if (currentDMXCommand[i] != ESP8266DMX.getSlot(DMX_ADDRESS + i + channel_anim ))
56         {
57             currentDMXCommand[i] = ESP8266DMX.getSlot(DMX_ADDRESS + i + channel_anim );
58             flag = true;
59         }
60     }
61     return flag;
62 }
63
64 void ANIM::ResetProgress(){
65     currentAnimationFrame = 0; // On recommence l' tape 1
66     currentAnimation = getAnimation(); // On set la nouvelle animation
67     hasAnimationBeenInitialized = false;
68     Tour->clear();
69 }
70
71 void ANIM::Update_Anim(){
72     if (getAnimation() != currentAnimation) { // Si on a changé d'animation on reset
73         les progressions
74         ResetProgress();
75     }
76     //On change la valeur de tempo pour correspondre la tempo de l'animation
77     voulue
78     tempo = GET_tempo(currentAnimation);
79 }
80
81 void ANIM::Basic(){
82     //On met simplement toutes les LED la couleur voulue
83     for(int i = 0; i < (Tour->GET_NbLed()); i++) {
84         Tour->setPixelColor(i, Tour->GET_colorTour());
85     }
86     hasAnimationChanged = true;
87     hasAnimationBeenInitialized = true;
88 }
89
90 void ANIM::Load(){
91     if (!hasAnimationBeenInitialized || currentAnimationFrame == 0) { //Si on est au
92         d but de l'animation, on clear les LED
93         Tour->clear();
94         hasAnimationBeenInitialized = true;
95     }
96     //Si la couleur du tour change durant l'animation, on le prend en compte et on
97     change la couleur de toutes les LED
98     if (Tour->GET_hasBorderColorChanged()) {
99         for(int i = 0; i < currentAnimationFrame; i++) {
100             Tour->setPixelColor(i, Tour->GET_colorTour());
101         }
102     }
103
104     // Si on est t+x du d but de l'animation, on allume la diode x
105     if (currentAnimationFrame > 0) {
106         Tour->setPixelColor(currentAnimationFrame - 1, Tour->GET_colorTour());
107     }
108
109     currentAnimationFrame = (currentAnimationFrame + 1) % Tour->GET_NbLed();
110     hasAnimationChanged = true;
111 }
```

```

107
108 void ANIM::Chase(){
109     Tour->fill(Tour->GET_colorTour()); //On allume toutes les LED
110
111     //On teint des LED les unes la suite de l'autre
112     for(int i = 0; i < 10; i++) {
113         Tour->setPixelColor(i * Tour->GET_NbLed() / 10 + currentAnimationFrame, 0);
114     }
115     currentAnimationFrame = currentAnimationFrame + 1;
116     hasAnimationChanged = true;
117 }
118
119 void ANIM::Eiffel(){
120     //A chaque fois, on allume quatre LED alatoirement et on teint les autres
121     Tour->clear();
122     Tour->setPixelColor(random(Tour->GET_NbLed()), Tour->GET_colorTour());
123     Tour->setPixelColor(random(Tour->GET_NbLed()), Tour->GET_colorTour());
124     Tour->setPixelColor(random(Tour->GET_NbLed()), Tour->GET_colorTour());
125     Tour->setPixelColor(random(Tour->GET_NbLed()), Tour->GET_colorTour());
126     hasAnimationChanged = true;
127 }
128
129 void ANIM::Rainbow(){
130     for (int i = 0; i < Tour->GET_NbLed(); i++) {
131         int colorOffset = (i + currentAnimationFrame) % Tour->GET_NbLed();
132         uint16_t hueValue = map(colorOffset, 0, Tour->GET_NbLed(), 0, 65535);
133         uint32_t rgbColor = Tour->ColorHSV(hueValue, 255, 255);
134         Tour->setPixelColor(i, rgbColor);
135     }
136     currentAnimationFrame = currentAnimationFrame + 1;
137     hasAnimationChanged = true;
138 }
139
140 bool ANIM::Check_Frame(){
141     return ((millis() - last_frame_time) > tempo) ;
142 }
143
144 void ANIM::Set_Frame(){
145     last_frame_time = millis();
146 }
147
148 bool ANIM::Check_Led(){
149     return ((millis() - last_led_update_time) > 0) ;
150 }
151
152 void ANIM::Set_Led(){
153     last_led_update_time = millis();
154 }
155
156
157 void ANIM::SET_hasAnimationChanged (bool state){
158     hasAnimationChanged = state ;
159 }
160
161 byte ANIM::GET_currentAnimation (){
162     return currentAnimation ;
163 }
164
165 bool ANIM::GET_hasAnimationChanged(){
166     return hasAnimationChanged ;
167 }
```