Lumieres

La débutante dans le décor - Ep13 - Que la lumière jaillisse !

Historique du document

20211029	Julie	Première version
20211030	Julie	Evolution de la documentation pour les nouvelles fonctionnalités

Disclaimer : documentation sans garantie de bon fonctionnement et applicable uniquement à la version **v20211030.6** du programme chargé sur un Arduino Nano v3.0.

---

0 - Pour les personnes pressées

Pour démarrer rapidement, vous devez télécharger le programme ***Lumieres*** puis ouvrir le fichier ***ConfigLumieres.h***, bricoler dedans et télécharger vers votre Arduino Nano.

Si vous avez accès à Github : https://github.com/Julaye/Lumieres

Si vous voulez comprendre les commandes utilisées dans ***ConfigLumieres.h***, vous pouvez aller directement au chapitre 7 - configuration d'une scène et au chapitre 8 - création d'un automatisme.

Le chapitre 9 - Exemples d'utilisation propose quelques animations plus atypiques que la simple gestion d'un bâtiment.

Bonne suite !

1 - Contexte

Le programme **Lumieres** est une tentative pour proposer aux adeptes du modélisme un automate de gestion des éclairages d'un bâtiment, ou plus généralement d'un ensemble d'éclairages, qui soit facile d'accès pour un non initié à la programmation d'un Arduino, et fortement paramétrable pour couvrir les besoins les plus courants.

L'utilisateur adapte le fichier ***ConfigLumieres.h*** selon ses besoins et il obtient un automate opérationnel, sans écrire une seule ligne de code, seulement des données dans le fichier susmentionné. C'est une approche NO CODE - LOW CODE pour surfer sur le buzz des outils de programmation actuels.

Les types d'éclairages actuellement supportés sont :

- lampe standard (allumé ou éteint)
- néon neuf avec sa séquence d'allumage,
- néon fatigué avec des glitchs aléatoires,
- flash de photographe avec pre-flash,

- poste de soudure à l'arc,
- servo moteur 0-90° pour l'ouverture/fermeture d'une porte,
- gyrophare,
- feu clignotant qui ne clignote que s'il n'est pas permanent.

Sur les sorties PWM (D3,D5,D6,D9,D10 et D11), l'automate gère la variation d'intensité lumineuse pour plus de réalisme.

Le logiciel est sous licence GPL et peut être librement utilisé et modifié dans le cadre de notre passion de modélisme, vos éventuelles modifications doivent être reversées à la communauté.

Ce programme peut être diffusé sur la plateforme Locoduino. Il a été diffusé initialement sur le forum 3rails dans l'épisode 13 de la débutante dans le décor.

2 - Environnement technique

Le logiciel est qualifié avec un Arduino Nano qui reste à mon sens la plateforme la plus adaptée pour ce besoin, que ce soit en terme de coût de la solution, que ce soit en terme de performance, que ce soit en termes de dimensions suffisamment raisonnables pour être intégré dans un bâtiment à l'échelle HO (1/87) ou tout autre échelle supérieure.

Voici les références des produits que j'utilise :

- AzDelivery - Lot de 3 Nano Card V3.0 Atmega328 CH340 Version à Puce soudée avec son câble USB

- XTVTX - Lot de 3 cartes d'extension Nano compatibles avec Arduino Nano V3.0 AVR ATMEGA328P-AU

L'ensemble devrait vous revenir à une dizaine d'euros par Arduino Nano, ce qui est vraiment compétitif par rapport au besoin couvert.

Pour les essais, mon Arduino est alimenté par le câble USB. En exploitation, je lui apporte une alimentation régulée 12V CC de qualité.

3 - Câblage de l'Arduino Nano

Le brochage de l'Arduino : https://content.arduino.cc/assets/Pinout-NANO_latest.png

Pour ce montage, j'ai dédié des broches spécifiques à des besoins spécifiques, selon la cartographie suivante :

GND	la masse de l'alimentation 12V CC régulée
VIN	le + de l'alimentation 12V CC régulée
D2	sortie éclairage 1

D3	sortie éclairage 2 avec réglage de l'intensité
D4	sortie éclairage 3
D5	sortie éclairage 4 avec réglage de l'intensité
D6	sortie éclairage 5 avec réglage de l'intensité
D7	sortie éclairage 6
D8	sortie éclairage 7
D9	sortie éclairage 8 avec réglage de l'intensité ou servo moteur
D10	sortie éclairage 9 avec réglage de l'intensité ou servo moteur
D11	sortie éclairage 10 avec réglage de l'intensité

La sortie D12 est laissée libre, elle pourra servir dans le futur pour une évolution de l'automate.

ATTENTION : vous ne devez rien brancher qui consomme plus de 40 mA sur une sortie donnée et l'ensemble de la consommation ne doit pas dépasser 200 mA. Le cas échéant, vous êtes invités à relayer vos sorties.

J'ai volontairement ignoré les ports D0 et D1 qui restent dédiés à la communication série, que ce soit pour la mise au point du programme ou le téléchargement d'une nouvelle version.

Le réglage de l'intensité se réalise au moyen du PWM associé aux sorties qui en sont pourvues, à savoir D3, D5, D6, D9, D10 et D11.

Il existe un tutoriel à ce sujet sur Locoduino : https://www.locoduino.org/spip.php?article47

Les Leds doivent être montées comme sur ce tutoriel, avec une résistance de 220 ohms.

Les entrées digitales sont configurées en pull-up (état logique 1 quand non connectée) utilisées sont au nombre de six dont deux qui ont un fonctionnement codé en dur :

A0/D14	RUN (1) ou STOP (0) pour un bouton poussoir par exemple
A1/D15	Choix de la séquence à exécuter sur l'automate

A ce stade, je peux donner quelques pistes de réflexion pour l'utilisation de ces deux entrées :

- Pour un automatisme simple qui se met en route au démarrage de l'Arduino, les deux entrées peuvent rester inutilisées.

- L'automate peut proposer deux automatismes différents que l'on peut sélectionner avec la broche D15, la sélection se faisant par un bouton poussoir relié à la masse ou par une sortie d'un décodeur comme les M83 ou M84 chez Marklin.

- L'entrée D14 peut être utilisée avec votre relais Jour/Nuit; que ce relais soit commandé par un bouton poussoir ou par la sortie d'un décodeur DCC.

- Nous verrons plus loin que le programme de notre automate peut se mettre en attente sur l'entrée D14, ce qui permet par exemple de gérer une phase d'allumage des éclairages (sortie D14 à 1) puis une phase d'extinction des éclairages (sortie D14 passe à 0).

Les quatre autres entrées A2/D16 .. A5/D19 sont utilisées pour brancher des détecteurs comme des zones de détection (rails de contacts, barriere infrarouge), des boutons poussoirs ou tout autre information qui pourra interagir avec le programme de l'automate.

A2/D16	Entrée 1
A3/D17	Entrée 2
A4/D18	Entrée 3
A4/D19	Entrée 4

Les entrées analogiques A6 et A7 sont réservés par le programme de l'Automate :

A6	Réservé - non utilisé dans la version actuelle
A7	Non connecté - utilisé par le générateur aléatoire

Le fichier ***ConfigNano.h*** permet de déclarer la cartographie des sorties et des entrées :

```
// mode des sorties : PWM ou IO
// -> attention l'intensité n'est pas réglable en IO, protéger avec une
//    résistance en fonction de la led utilisée

const int MODE_PWM = 0;
const int MODE_IO = 1;

// Configuration du mode de chaque sortie (Arduino Nano)
const int outputMode[] = {
            /* D2 */ MODE_IO,
            /* D3 */ MODE_PWM,
            /* D4 */ MODE_IO,
            /* D5 */ MODE_PWM,
            /* D6 */ MODE_PWM,
            /* D7 */ MODE_IO,
            /* D8 */ MODE_IO,
            /* D9 */ MODE_PWM,
            /* D10 */ MODE_PWM,
            /* D11 */ MODE_PWM,
};

// Nombre maximum d'éclairages dans le tableau précédent
#define maxLights sizeof(outputMode)/sizeof(int)

// Entrée digital D15 permettant de choisir la séquence 1 ou la séquence 2
(Pull-up)
const int seqPin = 15;

// Entrée digital D14 permettant de passer en RUN (1) ou en STOP (0)  (Pull-up)
const int startPin = 14;

// Entrées digitals utilisateur (Pull-Up)
const int inputUserPin1 = 16;
const int inputUserPin2 = 17;
const int inputUserPin3 = 18;
const int inputUserPin4 = 19;
```

```
// Entrée pour le seed Generator
const int seedPin = 7;
```

Pour ceux qui utiliseraient un autre Arduino que la version Nano, il faudra certainement adapter ce fichier de configuration.

4 - Machines à état fini

La machine à état fini est un objet informatique peu connu des amateurs et pourtant l'un des objets les plus importants de l'informatique. En anglais on appelle ces objets FSM ou Finite State Machine. MEF en français, ce n'est pas terrible ;), on peut parler d'Automate fini.

Les applications sont incroyablement nombreuses ...

C'est une machine à état fini avec son séquenceur de micro instructions (micro-code) que sont réalisés les processeurs qui font tourner tous nos équipements électroniques.

C'est aussi une machine à état fini que l'on utilise pour concevoir des compilateurs de langages informatiques, des machines virtuelles (i.e. Java), certains algorithmes d'intelligence artificielle (agents réactifs), la plupart des robots actuels, les piles de communication TCP/IP (et tous les protocoles réseaux en général), …

Assez surprenant, je suis toujours étonnée de voir les développeurs "pisser des lignes" avec des complexités importantes de "règles métiers" ou "business rules" pour parler anglisch … laborieux à écrire, long à mettre au point, impossible à maintenir …

Dans la majorité des cas, c'est une approche bien plus efficace d'utiliser explicitement une machine à état fini.

Il existe plusieurs méthodes pour implémenter ces machines, j'ai choisi pour ce programme d'utiliser la méthode la plus rustique : des switch avec des cases. Les autres méthodes demandent une pratique encore plus pointue de l'informatique avec par exemple une compréhension des pointeurs sur des fonctions, la théorie des graphes, les réseaux de pétri colorés ...

Au besoin, je vous conseille la lecture de tutoriels sur le switch, comme http://public.iutenligne.net/informatique/algorithme-et-programmation/priou/LangageC/523_le_choix_multiple__linstruction_switch.html

Tout ceci ne nous dit pas à quoi ressemble une machine à état fini :) !

Et bien c'est une logique ou programme qui nous fait passer d'un état ***Ei*** à un état **Ej** lorsque certaines conditions ou transitions sont remplies.

Important : le nombre d'états est connu E1 à En, le programme ne peut être que dans un seul état à la fois et chaque transition est bien documentée.

```
Exemple simpliste :
```

```
Etat1 : lumière éteinte
Etat2 : lumiére allumée

Transition1 : bouton poussoir enfoncé
Transition2 : bouton poussoir relâché
Transition3 : temporisation écoulée

Transition1 fait passer la lampe de l'état Etat1 à l'état Etat2.
Transition2 ou Transition3 fait passer la lampe de l'état Etat2 à l'état Etat1.
```

Pour en savoir plus, je vous recommande la lecture de l'article wikipedia https://fr.wikipedia.org/wiki/Automate_fini

5 - La gestion d'un éclairage ou d'une sortie

Pour la gestion d'un éclairage ou d'une sortie, nous allons utiliser une machine à état fini avec un tableau de transition adapté à chaque type d'éclairage.

Les types d'éclairages et de sortie supportées sont détaillés dans le chapitre 7 relatif à la configuration de votre scène.

5.1 - La simulation d'un néon

Pour simuler un néon, nous allons utiliser une machine à état fini assez simple avec un séquenceur encore plus simple.

Chaque transition est formée d'un couple :
- durée de la transition,
- intensité lumineuse

La séquence d'allumage du néon consiste à enchaîner des états sous la seule condition que la durée de chaque transition entre un état et un autre soit écoulée.

Au besoin, vous pouvez adapter cette séquence d'allumage de vos néons en fonction de vos goûts, il suffit simplement de modifier le tableau ***blinkNeon*** qui se trouve dans le fichier ***SimuNeon.h***.

A noter que votre modification interviendra pour l'ensemble des néons utilisés par l'automate. Une évolution future de l'automate pourrait proposer différents types de néons.

A noter que l'intensité lumineuse n'a d'intérêt que si la sortie utilisée est de type PWM. Dans le cas contraire, l'intensité lumineuse sera binaire : 0 éteinte ou 1 allumée. L'effet de néon sera certainement moins sympa.

Remarque : vous trouverez la simulation d'un gyrophare dans le fichier ***SimuGyro.h***, le principe est identique.

5.2 - L'automate fini de gestion d'un éclairage

La machine à états qui gère l'allumage d'un éclairage présente quatre états principaux :

- ***estate_OFF*** : l'éclairage est éteint,

- ***estate_STPWRUP*** : l'éclairage va démarrer une séquence d'allumage,

- ***estate_PWRUP*** : l'éclairage est dans la séquence d'allumage et enchaîne les transitions les unes après les autres,

- ***estate_ON*** : l'éclairage est allumé.

Pour générer les glitchs d'un néon vieillissant, la technique consiste à faire un jet aléatoire quand l'éclairage est dans son état ON et le cas échéant, de reprendre la séquence d'allumage PWRUP à partir d'une transition que j'ai positionnée à la 22ieme avec la constante ***INDEX_GLITCHPWRUP,*** valeur arbitraire que vous pouvez modifier dans le fichier ***SimuNeon.h***.

L'état de la machine à état fini de chaque éclairage est stocké dans le tableau ***gLight[éclairage]***.

Le détail de ces informations peuvent être retrouvées dans le fichier ***FSMeclairage.h*** :

```
// state of the FSM : "stateRunning"
// chaque éclairage a un état particulier et un graphe de transition

const int estate_OFF       = 0; /* éclairage éteint */

const int estate_STPWRUP = 1; /* éclairage en début d'allumage --> démarre la
séquence PWRUP sauf si l'éclairage est déjà allumé */

const int estate_PWRUP    = 2; /* éclairage en allumage --> ON quand la séquence
PWRUP sera terminée*/

const int estate_ON        = 3; /* éclairage allumé */

// blink transition pour un éclairage
typedef struct blink {
  int duration;
  int intensity;
} blink;

// parametres d'un éclairage
typedef struct {
    int stateRunning; /* état (OFF, STPWRUP, PWRUP ou ON) */
    int statePwrup;   /* dans l'état PWRUP, mémoriser la transition en cours */
    int stateDelay;   /* delay avant la prochaine transition ON OFF */
    blink*  pblink;   /* pointeur sur la table des transitions */
    int maxblink;     /* taille de la table des transitions */
     int nextState;    /* état une fois que la table des transitions a été parcourue
*/
} paramLight;

// Pour chaque éclairage
paramLight gLight[maxLights];
```

6 - La gestion d'une scène

Pour la gestion d'un bâtiment, d'une scène ou plus exactement d'un ensemble d'éclairages, nous allons utiliser une machine à état fini et un séquenceur de micro-code.

A noter que les opérations (ou commandes) possibles au niveau du micro-code sont abordées dans le chapitre consacré à la création d'un automatisme.

La machine à état fini utilise seulement trois états décrits dans le fichier ***FSMbatiment.h***, à savoir :

- ***START*** : l'automate démarre, après le démarrage de l'Arduino ou après un reset matériel (touche Reset ou entrée Reset de la carte) ou après un reset logiciel réalisé par une séquence 1 -> 0 -> 1 sur l'entrée D14 nommée ***startPin***.

- ***RUNNING*** : l'automate exécute la séquence programmée mySeq1 ou mySeq2 selon l'état de l'entrée D15 dite ***seqPin***, commande après commande jusqu'à rencontrer la commande END.

- ***STOP*** : l'automate se met à l'arrêt lorsqu'il a rencontré la commande END dans le micro-code. L'automate ne pourra re-démarrer que lorsque l'entrée D14 ***startPin*** passera de l'état 0 à l'état 1.

Au démarrage ou au reset, l'état START va permettre d'initialiser la machine à état et de démarrer réellement si et seulement si l'entrée D14 ***startPin*** est à l'état 1. Si cette entrée est à l'état 0, l'automate reste en état START et les séquences ne peuvent pas démarrer.

Il est très important de noter que même dans l'état STOP, les machines à état fini des éclairages sont toujours actives.

L'automate est en mode STOP mais les éclairages allumés restent allumés et les simulations de glitch sur les vieux néons toujours opérationnelles. Ce n'est pas une anomalie mais bien une fonctionnalité désirée.

La variable globale ***gSeqState*** contient l'état courant de la machine à état fini qui gère la scène.

La variable globale ***gpSeq*** contient le pointeur sur la commande (ou micro-code) à exécuter.

7 - La configuration de votre scène - Guide de l'utilisateur

Ce chapitre vous explique comment configurer votre scène. Par configuration, j'entends ce qu'il faut paramétrer pour que l'automate sache comment gérer chacun de vos éclairages.

Pour chaque sortie D2 à D11, il faut décider quel est le type d'éclairage connecté.

Le programme supporte plusieurs types d'éclairages et périphériques différents :

- ***ETYPE_STANDARD*** : un éclairage tout à fait classique : allumé ou éteint.

- ***ETYPE_NEONNEUF*** : un éclairage de type néon avec sa séquence de démarrage,

- ***ETYPE_NEONVIEUX*** : un éclairage de type néon qui a un peu vieilli et qui présente aléatoirement quelques glitchs.

- ***ETYPE_GYROPHARE*** : un éclairage de type gyrophare rotatif.

- ***ETYPE_FLASH*** : un flash d'appareil photo ou d'un radar.

- ***ETYPE_SOUDURE*** : simulation d'un poste de soudure. J'ai repris l'algorithme proposé par Locoduino : https://www.locoduino.org/spip.php?article90.

- ***ETYPE_FIRE*** : simulation d'un feu, bougie, brasero.

- ***ETYPE_SERVO*** : un servomoteur qui peut prendre deux positions : ON à 90° (porte ouverte) et OFF à 0° (porte fermée). A installer sur D9 ou sur D10.

- ***ETYPE_CLIGNOTANT*** : un feu clignotant qui clignote sauf si il a été allumé avec la commande PERM. Dans ce cas, c'est un feu fixe.

ETYPE_NOTUSED permet d'indiquer une sortie qui n'est connectée à aucun éclairage. Cette information est donnée à titre indicatif, elle n'est pas utilisée par l'automate.

Attention : de par la nature aléatoire de leur algorithme et de la mémoire restreinte de l'Arduino Nano sans allocateur dynamique, vous ne pouvez avoir qu'une seule sortie "Poste de Soudure", qu'une seule sortie Servo moteur et qu'une seule sortie "Simulation d'un feu". Si vous en configurez plusieurs, l'automate va partir dans le décor …

Important : les éclairages de type Néon, Soudure et Fire utilisent la variation d'intensité rendue possible par le PWM. Il est préférable d'utiliser des sorties supportant le PWM et c'est même indispensable pour Fire (sorties D3, D5, D6, D9, D10 et D11).

Ces définitions de type d'éclairages se trouvent dans le fichier ***FSMeclairage.h***.

En utilisant ces définitions, il faut paramétrer le tableau de configuration ***ledCnf*** qui se trouve au début du fichier ***ConfigBatiment.h***.

Le programme est livré avec ce tableau déjà programmé pour le bâtiment administratif de notre très chère Mlle Pélerin. Il vous faut le modifier en fonction de vos besoins.

```
// Affectation des types d'éclairage à chaque sortie
const int ledCnf[] = {
            /* D2 */ ETYPE_NEONNEUF,   /* 1 : bureau administratif (IO) *
            /* D3 */ ETYPE_NEONNEUF,   /* 2 : accueil (PWM) */
            /* D4 */ ETYPE_STANDARD,   /* 4 : bureau M. CLaude (IO) */
            /* D5 */ ETYPE_NEONNEUF,   /* 8 : couloir haut (PWM) */
            /* D6 */ ETYPE_NEONVIEUX,  /* 16 : escalier (PWM) */
            /* D7 */ ETYPE_NEONNEUF,   /* 32 : couloir bas (IO) */
```

```
            /* D8 */ ETYPE_NEONNEUF,   /* 64 : bureau M. Gaston (IO) */
            /* D9 */ ETYPE_NEONNEUF,   /* 128 : bureau secrétaire (PWM) */
            /* D10 */ ETYPE_STANDARD,  /* 256 : bureau Mlle Pélerin (PWM) */
            /* D11 */ ETYPE_GYROPHARE  /* 512 : gyrophare (PWM) */
};
```

Dans le futur, notre automate pourra évoluer en proposant de nouveaux types d'éclairage et de périphériques, il suffira de modifier ce paramétrage pour les sorties concernées.

Si vous utilisez la fonctionnalité servomoteur et que vous avez configuré D9 ou D10 avec une sortie de type servomoteur, il faut vous assurer d'inclure la librairie <Servo.h> en enlevant la mise en commentaire comme ceci :

```
// Pour la commande d'un servo moteur sur la sortie D9 ou D10 uniquement
// LIGNE CI-APRÈS À METTRE EN COMMENTAIRE SI VOUS N'UTILISEZ AUCUN SERVO
#include <Servo.h>
```

8 - La création d'un automatisme - Guide de l'utilisateur

Ce chapitre passe en revue la création d'un automatisme en donnant quelques exemples de programmation.

Le principe consiste à programmer la machine à état fini avec une suite ou séquence de micro-instructions.

Chaque micro-instruction est composée de trois informations :

- **io** : le plus souvent, le ou les sorties d'éclairage concernées, chaque bit de cette valeur représente une sortie, bit 1: D2 … bit 10: D11,

- **duration** : un paramètre associé à cette instruction, le plus souvent une durée d'allumage en secondes,

- **commande** : l'instruction elle-même ou commande sur la sortie, un éclairage le plus souvent.

Vous pouvez programmer deux séquences différentes, nommées ***mySeq1*** et ***mySeq2***, elles se trouvent au début du fichier ***ConfigBatiment.h***.

Le programme est livré avec ces deux séquences déjà programmées pour le bâtiment administratif de notre très chère Mlle Pélerin. Il vous faut les modifier en fonction de vos besoins et avec l'aide des informations que nous allons donner ci-après.

8.1 - Les commandes de base

8.1.1 - SET

Prenons l'exemple du début de la séquence 1 qui simule l'arrivée de Mlle Pélerin dans le bâtiment et qui va allumer les différents éclairages : accueil, couloir puis escalier.

```
2,     10, SET, /* allume l'accueil (néon neuf) pendant 10 secondes */

2+32, 10, SET, /* laisse l'accueil allumé et allume le couloir RDC (néon neuf)
pendant 10 secondes */

16+32, 10, SET, /* éteint l'accueil et allume le couloir RDC (néon neuf) et
l'escalier (néon ancien) pendant 10 secondes */
```

La première ligne a trois informations : io:2, duration:10, commande:SET.

Pour rappel, le premier paramètre **io** est la sortie concernée, le second paramètre **duration** accompagne l'instruction - le plus souvent un temps en secondes, le troisième paramètre est la **commande** elle-même.

Ici c'est la commande **SET** qui va :

- lancer l'allumage de l'éclairage de l'accueil (D3 bit2 -> valeur 2)
- attendre 10 secondes avant de passer à la commande suivante

La seconde ligne sera donc exécutée après que l'accueil se soit allumé et que 10 secondes se soient écoulées.

La seconde ligne a trois informations : io:2+32, duration:10, commande:SET.

Même commande SET qui va :

- lancer l'allumage de l'éclairage de l'accueil (D3 bit2 -> valeur 2) et du couloir bas (D7 bit6 -> valeur 32)
- attendre 10 secondes avant de passer à la commande suivante

Important : l'éclairage de l'accueil étant déjà allumé du fait de la commande SET précédente, l'accueil reste allumé sans qu'une séquence d'allumage soit relancée.

La seconde ligne a trois informations : io:16+32, duration:10, commande:SET.

Même commande SET qui va :

- lancer l'allumage de l'escalier (D6 bit5 -> valeur 16) et du couloir bas (D7 bit6 -> valeur 32)
- attendre 10 secondes avant de passer à la commande suivante

Important : comme précédemment, l'éclairage du couloir bas étant déjà allumé du fait de la commande SET précédente, le couloir bas reste allumé sans qu'une séquence d'allumage soit relancée.

Corollaire encore plus important : comme rien n'est précisé concernant l'éclairage de l'accueil, celui-ci va être automatiquement éteint !

Vous avez compris le principe pour la suite des lignes :

```
16+8, 10, SET, /* ETC :)  */

8+256,10, SET
```

avec D5 bit4 : 8 étant le couloir du haut
et D10 bit 9 : 256 étant le bureau de la Miss …

En résumé, la commande SET allume un ou plusieurs éclairages paramétrés avec **io** et pendant une durée **duration** secondes.

A noter qu'il est possible d'utiliser une durée de 0 mais qui correspond alors à une demi seconde.

8.1.2 - PERM

La commande suivante, **PERM**, permet de rendre un éclairage permanent jusqu'à une commande explicite d'extinction (UNSET).

Le paramètre **io** précise le ou les éclairages / sorties à activer, le paramètre **duration** n'est pas utilisé et doit être mis à 0.

```
2,     10, SET,

2,     0,  PERM,

32,    10, SET,

32+1,  0,  PERM,
```

La première ligne allume l'accueil pendant 10 secondes puis passe l'accueil en éclairage permanent.

Cette séquence SET suivie de PERM est nécessaire pour temporiser l'exécution de la troisième ligne, à savoir l'allumage du couloir, 10 secondes après l'accueil.

La quatrième ligne est intéressante, elle allume de manière permanente le couloir et le bureau de l'administration.

Ces quatres lignes simulent une personne qui entre de nuit dans le bâtiment et rejoint son bureau. Son bureau et le couloir vont rester allumés alors que l'accueil est temporisé.

Un éclairage permanent peut néanmoins être éteint par une commande explicite UNSET.

La suite de cette séquence pourrait donc être

```
0 ,   60, WAIT, /* la personne reste dans son bureau pendant 60 secondes */

32+1,  0, UNSET,/* puis s'en va en éteignant son bureau et le couloir */

2,    10, SET, /* retraverse l'accueil qui s'allume 10 secondes */

0,     0, END /* fin de la séquence */
```

8.1.3 - END

A ce stade, il suffit de vous dire que toute séquence doit se terminer une commande END ou équivalente (nous verrons plus tard).

```
0,     0,  END    /* fin de la séquence */
```

Lorsque l'automate rencontre la commande ***END***, il laisse les éclairages permanents allumés mais passe l'automate dans l'état STOP. Cela a été abordé dans le chapitre 6.

Important : vous pouvez indiquer dans le premier paramètre **io** la valeur 0 comme dans l'exemple ci-dessus ou bien le ou les éclairages non permanents allumés précédemment et que vous souhaitez laisser allumés.

Voilà pour les commandes de base, vous pouvez déjà réaliser pas mal d'automatismes en enchaînant correctement ces commandes SET, PERM et END.

Mais il y a beaucoup mieux avec les commandes avancées :) …

8.2 Les commandes avancées

Les commandes avancées permettent d'élaborer des scénarios plus complexes, pour simuler des présences (comme en domotique) et introduire une part d'aléatoire.

8.2.1 - MARK / LOOP

La commande ***MARK*** permet de placer un marqueur sur l'instruction courante et la commande ***LOOP*** permet de revenir sur cette instruction courante.

Ces commandes ne prennent aucun paramètre.

```
/* Séquence 2 :  un chenillard sur l'ensemble des sorties, c'est encore mieux ! */

const int seqDebug2[] = {
   0,      0,  MARK, /* place une MARK pour le LOOP */
   ...
   0,      0,  LOOP  /* Tout est éteint. Boucle sur le début MARK */
};
```

Ce couple permet d'implémenter un chenillard comme celui que nous avons mis dans le mode debug. Cf l'annexe A3 à ce sujet.

A noter que la commande LOOP remplace utilement la commande END, qu'il n'est nul besoin d'une commande END puisque l'automate va exécuter en continu - sans fin - les commandes qui se trouvent entre MARK et LOOP.

8.2.2 - STANDBY

La commande ***STANDBY*** allume les éclairages précisés avec le paramètre **io** et attend pendant X minutes. X est une valeur aléatoire comprise entre 1 et N, N étant précisé dans le paramètre **duration**.

Cette commande ne touche à aucun éclairage actif, qu'il soit permanent ou temporaire.

La commande peut être précédée d'une ou plusieurs commandes ALEA pour allumer aléatoirement certains éclairages qui resteront allumés pour la même durée X.

```
64,    2, ALEA, /* parfois M. gaston accompagne la miss (50% de chance) */

128,   2, ALEA, /* parfois la secrétaire accompagne la miss (50% chance) */

256,   10, STANDBY, /* la miss reste dans son bureau pendant 1 à 10 minutes */
```

8.2.3 - UNSET

La commande ***UNSET*** permet d'éteindre un ou plusieurs éclairages qui ont été allumés avec une commande PERM.

Le premier paramètre **io** spécifie la ou les éclairages concernés.

Le second paramètre **duration** devrait avoir la valeur 0, sinon l'automate va attendre le nombre de secondes spécifié.

La commande suivante permet d'éteindre toutes les lumières :)

```
1023,  0,  UNSET, /* éteint toutes les lumières */
```

8.2.4 - ALEA

La commande ***ALEA*** allume le ou les éclairages spécifiés dans le paramètre **io** uniquement si le tirage aléatoire est favorable.

Le tirage aléatoire est effectué entre 0 inclus et la valeur spécifiée en paramètre **duration** exclue, et le tirage est favorable si c'est la valeur 0 qui sort.

La commande ALEA n'a aucun effet sur les éclairages déjà allumés, qu'ils soient permanents ou non.

Il est donc possible de faire suivre la commande ALEA par d'autres commandes ALEA ou par une commande STANDBY ou WAIT. Voir à ce sujet les exemples donnés.

```
64, 2, ALEA, /* parfois M. gaston accompagne la miss (50% de chance) */

128, 4, ALEA, /* parfois la secrétaire accompagne la miss (25% chance) */

256, 10, STANDBY, /* la miss reste dans son bureau (1 à 10 minutes) */
```

Voilà pour les commandes avancées, vous pouvez réaliser pas mal d'automatismes en enchaînant du certain avec SET, PERM, UNSET, MARK/LOOP et de l'incertain avec ALEA.

Mais il y a vraiment mieux avec les commandes très avancées :) …

8.3 Les commandes très avancées

Des commandes spéciales pour interagir avec autre chose que l'automate et les éclairages.

Ces commandes sont susceptibles d'être augmentées avec de nouveaux besoins que je suis susceptible de rencontrer dans mes activités de ferro-modélisme. N'hésitez pas non plus à exprimer des demandes que je traiterais en fonction de mon intérêt et de mes disponibilités.

8.3.1 - WAIT

La commande ***WAIT*** attend pendant le nombre de secondes spécifiées dans le paramètre **duration** et ne touche à aucun éclairage actif, qu'il soit permanent ou temporaire.

C'est similaire à STANDBY pour son fonctionnement, à savoir que cette commande peut être précédée d'une ou plusieurs commandes ALEA pour allumer aléatoirement certains éclairages.

La différence étant que STANDBY attend d'une durée aléatoire en minutes alors que WAIT attend d'une durée fixée en secondes.

A noter qu'il est possible d'utiliser une durée de 0 mais qui correspond alors à une demi seconde d'attente.

```
64,    2, ALEA, /* parfois M. gaston accompagne la miss (50% de chance) */

128,   2, ALEA, /* parfois la secrétaire accompagne la miss (50% chance) */

256,   0, SET,  /* la miss entre dans son bureau */

0,    60, WAIT, /* la miss reste dans son bureau pendant 60 secondes */
```

8.3.2 - WSTOP

La commande ***WSTOP*** permet de passer à l'instruction suivante lorsque une entrée digitale prend un certain état et ne touche à aucun éclairage actif, qu'il soit permanent ou temporaire.

Le paramètre **io** indique le numéro de l'entrée concernée :

- 0 : l'entrée D14 (startPin) à l'état bas
- 1 : l'entrée utilisateur D16 (inputUserPin1) à l'état bas
- 2 : l'entrée utilisateur D17 (inputUserPin2) à l'état bas
- 3 : l'entrée utilisateur D18 (inputUserPin3) à l'état bas
- 4 : l'entrée utilisateur D16 (inputUserPin4) à l'état bas
- 128 : l'entrée D14 (startPin) à l'état haut
- 129 : l'entrée utilisateur D16 (inputUserPin1) à l'état haut
- 130 : l'entrée utilisateur D17 (inputUserPin2) à l'état haut
- 131 : l'entrée utilisateur D18 (inputUserPin3) à l'état haut
- 132 : l'entrée utilisateur D16 (inputUserPin4) à l'état haut

Le paramètre **duration** indique la fréquence de test sur cette entrée (en secondes).

A noter qu'il est possible d'utiliser une durée de 0 mais qui correspond alors à une demi seconde de fréquence.

L'exemple suivant conditionne la suite de l'exécution de l'automatisme à l'entrée D14 qui peut être utilisée comme une information Jour(0)/Nuit(1).

```
0,3, WSTOP, /* attend que l'entrée D14 passe à LOW, teste chaque 3 secondes */

1023,  0,  UNSET, /* c'est le jour, éteint toutes les lumières permanentes */

0, 0, END /* fin de la séquence */
```

8.3.3 - PWM

La commande ***PWM*** envoie un signal PWM sur une ou plusieurs sorties supportant le mode PWM.

Le paramètre **io** indique le numéro de sortie concernée. La commande ne fonctionnera que pour une sortie supportant le PWM (D3, D5, D6, D9, D10 ou D11) et configurée en mode ETYPE_STANDARD dans le tableau de configuration ledCnf[].

Le paramètre **duration** indique le rapport cyclique de la PWM entre 1 et 255 (100%). La valeur 0 n'est pas autorisée, il faut utiliser la commande ***UNSET***.

L'exemple suivant envoie un signal PWM sur la sortie D3 avec un rapport cyclique de 50% puis de 0 après 3 secondes.

```
4,128, PWM,
0,  3, WAIT,
4,  0, UNSET
```

8.4 - Tableau résumé des commandes

io	duration	commande	Extinction (1)	Description
sortie(s) concernée(s)	0	UNSET	oui	éteint le ou les éclairages spécifiés par io en mettant la sortie à l'état bas
sortie(s) concernée(s)	en secondes (2)	SET	oui	allume le ou les éclairages spécifiés par io en lançant le cycle d'allumage, cycle qui dépend du type d'éclairage et attend ensuite le délai duration secondes avant de passer à la commande suivante.
sortie(s) concernée(s)	en minutes	STANDBY	aucune	allume le ou les éclairages spécifiés par io en lançant le cycle d'allumage, cycle qui dépend du type d'éclairage et attend ensuite pendant un délai aléatoire compris entre 1 et duration minutes avant de passer à la commande suivante.
sortie(s) concernée(s)	0	PERM	oui	allume le ou les éclairages spécifiés par io en lançant le cycle d'allumage, cycle qui dépend du type d'éclairage. Les éclairages seront éteints par une commande UNSET.
sortie(s) concernée(s)	max	ALEA	aucune	Réalise un tirage aléatoire compris entre 0 et max-1. Le tirage est réussi avec la valeur 0 et le cas échéant, allume le ou les éclairages spécifiées par io en lançant le cycle d'allumage, cycle qui dépend du type d'éclairage
-	en secondes (2)	WAIT	aucune	L'automate attend pendant un délai aléatoire compris entre 1 et duration secondes avant de passer à la commande suivante.
entrée concernée ou entrée concernée + 128	en secondes (2)	WSTOP	aucune	L'automate attend que l'entrée concernée (0 à 4 soit respectivement D14, D16, D17, D18 et D19) soit à l'état bas avant de passer à la commande suivante. Si io est supérieur ou égal à 128, l'entrée concernée est attendue à l'état haut.

sortie(s) concernée(s)	1 à 255	PWM	aucune	Envoie des impulsions PWM avec un rapport cyclique de duration compris entre 1 et 255. Pour arreter les impulsions, il faut utiliser la commande UNSET.
sortie(s) à maintenir allumée(s)	0	END	partiel	l'automate passe en mode STOP et n'execute plus aucune commande. Néanmoins, les séquences d'allumage des éclairages sont toujours actives, par exemple les gltichs aléatoires sur un néon allumé ou bien le cycle d'un clignotant ...
sortie(s) à maintenir allumée(s)	0	MARK	partiel	place un marqueur sur la commande en cours. L'automate pourra y revenir avec la commande LOOP.
sortie(s) à maintenir allumée(s)	0	LOOP	partiel	L'automate retourne à la commande marquée. Si aucune marque n'a été précédemment posée par une commande MARK, l'automate retourne en début de la séquence.

(1) : indique si la commande éteint les éclairages non permanents allumés par des commandes précédentes
(2) : la valeur 0 indique qu'il faut appliquer un délai d'une demi-seconde

9 - Exemples d'utilisation de l'automate

Si le projet a démarré avec l'idée de gérer les éclairages d'un bâtiment, il a vite évolué pour gérer des scènes diverses que l'on retrouve souvent sur les réseaux de ferro-modélisme.

Les épisodes de la débutante dans le décor ont proposé différents petits programmes pour gérer des scènes spécifiques : Paparazzi qui prend des photos au passage d'un train, éclairages d'une fosse d'inspection quand une locomotive stationne au-dessus, ouverture de la porte d'une remise à l'arrivée de l'autorail …

Il est donc naturel de revisiter ces scènes spécifiques en proposant un fichier de configuration pour notre automate ***Lumieres***.

9.1 - Paparazzi

Paparazzi est un petit personnage à l'échelle 1/87 qui est muni d'une micro-led montée au-dessus de son appareil photo. Paparazzi déclenche sa prise photographique aléatoirement lorsque un train se trouve dans une zone de détection.

Dans l'exemple ci-après, la présence de la locomotive est détectée par l'entrée 1 (D16) à l'état bas, déclenche le flash à une fréquence aléatoire. Lorsque l'entrée repasse à l'état haut, le photographe s'arrête.

Avec la configuration suivantes des sorties :

```
const int ledCnf[] = {
            /* D2 */ ETYPE_NOTUSED, /* 1 : non utilisé (IO) *
            /* D3 */ ETYPE_FLASH,   /* 2 : flash du photographe (PWM) */
...
            /* D11 */ ETYPE_NOTUSED /* 512 : non utilisé (PWM) */
};
```

L'automatisme s'écrit assez simplement :

```
0, 0, MARK,     /* point de retour de l'automatisme */

1, 1, WSTOP,    /* attend la détection d'une locomotive sur l'entrée 1 */

2, 4, ALEA,     /* lance le flash configuré sur la sortie D3 aléatoirement (0..3) */

0, 0, LOOP,     /* retourne pour flasher à nouveau */
```

9.2 - Fosse d'inspection

L'éclairage d'une fosse d'inspection doit s'allumer lorsque une locomotive stationne un certain temps au-dessus de la fosse qui est munie d'un capteur de présence (rail contact ou autre détection infrarouge).

Xxx.

9.3 - Ouverture / Fermeture d'une porte de remise

Les sorties PWM peuvent être utilisées pour commander autre chose que des leds, par exemple pour commander un servomoteur et proposer ainsi une animation mécanique. L'état ON devient "Porte ouverte" et l'état OFF devient "Porte fermée".

Dans l'exemple ci-après, la présence de la locomotive est détectée par l'entrée 1 (D16) à l'état bas, déclenche l'ouverture de la porte et la mise en route du gyrophare. Lorsque l'entrée repasse à l'état haut, la porte se ferme et le gyrophare s'arrête.

Avec la configuration suivantes des sorties :

```
const int ledCnf[] = {
            /* D2 */ ETYPE_NEONNEUF,   /* 1 : lampe de l'atelier (IO) *
            /* D3 */ ETYPE_NEONNEUF,   /* 2 : néon de l'atelier (PWM) */
...
            /* D10 */ ETYPE_SERVO,     /* 256 : servomoteur de la porte (PWM) */
            /* D11 */ ETYPE_GYROPHARE  /* 512 : gyrophare de la porte (PWM) */
};
```

L'automatisme s'écrit assez simplement :

```
1+2, 0, PERM, /* l'atelier se met au travail au démarrage de l'Arduino … */

0, 0, MARK,   /* point de retour de l'automatisme */

1, 1, WSTOP,  /* attend la détection d'une locomotive sur l'entrée 1 */

512, 0, PERM, /* lance le gyrophare configuré sur la sortie D11 */

0, 5, WAIT,   /* attend 5 secondes */

256, 0, PERM, /* envoie 90° sur la sortie D10 du servomoteur */

129, 1, WSTOP, /* attend la non détection de la locomotive sur l'entrée 1 */

256, 0, UNSET, /* envoie 0° sur la sortie D10 du servomoteur */

0, 5, WAIT,    /* attend 5 secondes */

512, 0, UNSET, /* arrête le gyrophare configuré sur la sortie D11 */

0, 0, LOOP,    /* retourne attendre la locomotive suivante */
```

9.4 - Animation d'un poste de soudure

L'exemple précédent peut être enrichi par un Poste de soudure dans l'atelier lorsque la locomotive est entrée dans la zone.

La présence de la locomotive est détectée par l'entrée 2 (D17) à l'état bas, déclenche le poste de soudure.

Avec la configuration suivantes des sorties, en gras la ligne ajoutée :

```
const int ledCnf[] = {
            /* D2 */ ETYPE_NEONNEUF,  /* 1 : lampe de l'atelier (IO) *
            /* D3 */ ETYPE_NEONNEUF,  /* 2 : néon de l'atelier (PWM) */
            /* D4 */ ETYPE_NOTUSED,   /* 4 : non utilisé (IO) *
            /* D5 */ ETYPE_SOUDURE,   /* 8 : poste de soudure de l'atelier (PWM) */
...
            /* D10 */ ETYPE_SERVO,    /* 256 : servomoteur de la porte (PWM) */
            /* D11 */ ETYPE_GYROPHARE /* 512 : gyrophare de la porte (PWM) */
};
```

L'automatisme s'écrit assez simplement, en gras les lignes ajoutées :

```
1+2, 0, PERM, /* l'atelier se met au travail au démarrage de l'Arduino … */

0, 0, MARK,   /* point de retour de l'automatisme */

1, 1, WSTOP,  /* attend la détection d'une locomotive sur l'entrée 1 */

512, 0, PERM, /* lance le gyrophare configuré sur la sortie D11 */

0, 5, WAIT,   /* attend 5 secondes */

256, 0, PERM, /* envoie 90° sur la sortie D10 du servomoteur */

129, 1, WSTOP, /* attend la non détection de la locomotive sur l'entrée 1 */

256, 0, UNSET, /* envoie 0° sur la sortie D10 du servomoteur */

0, 5, WAIT,    /* attend 5 secondes */

512, 0, UNSET, /* arrête le gyrophare configuré sur la sortie D11 */

2, 1, WSTOP,  /* attend la détection d'une locomotive sur l'entrée 2 */

8, 0, SET,    /* un peu de soudure sur la locomotive qui vient de rentrer */

130, 1, WSTOP,/* attend la non détection de la locomotive sur l'entrée 2 */

8,0, UNSET,   /* arrête le poste soudure, la loco est partie ! */

0, 0, LOOP,    /* retourne attendre la locomotive suivante */
```

9.5 - Animation d'une croix de pharmacie

Xxx.

9.6 - Animation d'un brasero

Avec la configuration suivantes des sorties :

```
const int ledCnf[] = {
            /* D2 */ ETYPE_NOTUSED,   /* 1 : non utilisé *
            /* D3 */ ETYPE_FIRE,      /* 2 : brasero (PWM) */
...
            /* D11 */ ETYPE_NOTUSED /* 512 : non utilisé(PWM) */
};
```

L'automatisme est simplissime :

```
2, 0, PERM,   /* allume le feu */
```

9.7 - Animation d'un croisement routier (feux tricolores)

L'exemple suivant réalise l'animation de feux tricolores à un croissement. Il est prévu deux modes de fonctionnement : feux fonctionnels et feux en panne (orange clignotant).

Pour rappel, la séquence est choisie par l'entrée pinSeq D15.

Avec la configuration suivantes des sorties :

```
const int ledCnf[] = {
            /* D2 */ ETYPE_STANDARD,  /* 1 : Feu Vert 1 *
            /* D3 */ ETYPE_STANDARD,  /* 2 : Feu Orange 1 */
            /* D4 */ ETYPE_STANDARD,  /* 4 : Feu Rouge 1 */

            /* D5 */ ETYPE_STANDARD,  /* 8 : Feu Vert 2 *
            /* D6 */ ETYPE_STANDARD,  /* 16 : Feu Orange 2 */
            /* D7 */ ETYPE_STANDARD,  /* 32 : Feu Rouge 2 */
...
            /* D11 */ ETYPE_NOTUSED,   /* 512 : non utilisé */
};
```

L'automatisme s'écrit assez simplement, une séquence pour les feux fonctionnels, une séquence pour les feux clignotants orange :

```
/* Séquence 1 : Feux fonctionnels */
const int mySeq1[] = {
       1023,0, UNSET,        /* éteint toutes les leds */
       32,  0, PERM,         /* allumage feu rouge 2 */

       0,   0,  MARK,        /* pour boucler */

       1,  30, SET,          /* allumage feu vert 1 pour 30 secondes */

       2,   5, SET,          /* allume feu orange 1 pour 5 secondes */
       4,   0, PERM,         /* allumage feu rouge 1 */

       0,   2, WAIT,         /* attends 2 secondes pour les chauffards */

       32,  0, UNSET,        /* éteint feu rouge 2 */
       8,  30, SET,          /* allume feu vert 2 pour 30 secondes */
       16,  5, SET,          /* allume feu orange 2 pour 5 secondes */
       32,  0, PERM,         /* allume feu rouge 2 */

       0,   2, WAIT,         /* attends 2 secondes pour les chauffards */

       0,   0, LOOP          /* on boucle */
```

```
};
```

```
/* Séquence 2 : Feux (en panne) clignotants orange */
const int mySeq2[] = {
       0, 0, MARK,
       2, 1, SET,
       16, 1, SET,
       0, 0, LOOP
};
```

9.8 - Un feu clignotant qui se fige puis s'éteint

Tout est dans le titre. Un début d'animation pour un passage à niveau ?

Les entrées 1 D16 et 2 D17 pourraient détecter la présence d'un train en amont du feu du passage à niveau, sur un niveau bas ...

Avec la configuration suivantes des sorties :

```
const int ledCnf[] = {
            /* D2 */ ETYPE_NOTUSED,    /* 1 : non utilisé *
            /* D3 */ ETYPE_CLIGNOTANT, /* 2 : feu clignotant (PWM) */
...
            /* D11 */ ETYPE_NOTUSED    /* 512 : non utilisé(PWM) */
};
```

L'automatisme est simplissime :

```
1, 1, WSTOP, /* attend que l'entrée 1 D16 passe à l'état bas */
2, 1, SET,   /* lance le clignotement du feu */
2, 1, WSTOP, /* attend que l'entrée 2 D17 passe à l'état bas */
2, 0, PERM,   /* le feu devient fixe */
130,1, WSTOP, /* attend que l'entrée 2 D17 repasse à l'état haut */
2, 0, UNSET,  /* le feu s'éteint */
```

Je vous laisse ajouter un servo moteur sur D10 dans la configuration des sorties et une commande SERVO pour baisser la barrière avec le clignotement du feu …

Bon j'ai pitié :)

```
const int ledCnf[] = {
            /* D2 */ ETYPE_NOTUSED,   /* 1 : non utilisé *
            /* D3 */ ETYPE_CLIGNOTANT,/* 2 : feu clignotant (PWM) */
...
            /* D10 */ ETYPE_SERVO,    /* 256 : servomoteur de la barrière (PWM) */
            /* D11 */ ETYPE_NOTUSED   /* 512 : non utilisé(PWM) */
};
```

```
1, 1, WSTOP,  /* attend que l'entrée 1 D16 passe à l'état bas */
256, 0, PERM, /* envoie 90° sur D10 du servomoteur : la barrière se ferme  */
2, 1, SET,    /* lance le clignotement du feu */
2, 1, WSTOP,  /* attend que l'entrée 2 D17 passe à l'état bas */
2, 0, PERM,   /* le feu devient fixe */
130,1, WSTOP, /* attend que l'entrée 2 D17 repasse à l'état haut */
2, 0, UNSET,  /* le feu s'éteint */
256, 0, UNSET, /* envoie 0° sur D10 du servomoteur : la barrière s'ouvre */
```

Annexes - Quelques éléments de compréhension

Cette annexe technique est destinée à ceux qui souhaitent étudier et comprendre le fonctionnement du programme proposé. En espérant que cela vous donne envie de proposer une version cet automate encore plus puissante et toujours plus simple à utiliser.

A1 - Choix de programmation

Le programme doit fonctionner dans un environnement très contraint, il n'est pas possible d'avoir des threads ou des processus (au sens unix), même en version légère. Je n'ai pas voulu mettre en œuvre un mécanisme de co-routine qui me paraissait disproportionné par rapport à l'objectif.

De la même façon, ayant basé mon développement sur un Arduino Nano de base, et avec des machines à état fini et n'ayant aucune contrainte de portabilité, je n'ai pas utilisé de programmation objet. Pour les machines à état fini, je suis resté avec un codage compréhensible, avec des switchs/case et aucun tableau de pointeurs sur fonctions …

A2 - Système de trace

J'ai mis en place un système de trace qui a été bien utile pour mettre au point l'automate et qui peut avoir plusieurs usages. Il faut activer la fenêtre ***Outils->Moniteur Série*** dans l'IDE de l'Arduino pour visualiser les traces produits par le programme.

Deux constantes permettent de rendre le programme plus ou moins verbeux :

- ***debug*** avec les valeurs 0 (rien), 1 (résumé) et 2 (détaillé) pour des informations techniques. Utile pour chercher une régression dans le programme de l'automate.

- ***verbose*** avec les valeurs 1 (bavard) et 0 (silencieux) pour des informations fonctionnelles. Utile pour visualiser la séquence de micro-code en cours d'exécution.

Je conseille de mettre ***debug à 0*** et ***verbose à 1*** quand vous mettez au point le paramétrage de l'automate pour votre projet personnel. Le programme trace alors les commandes du micro-code exécuté ainsi que l'état des éclairages. Très utile pour développer son projet avant même d'avoir cablé la moindre led à l'Arduino.

Exemple de traces obtenues dans cette configuration "utilisateur", avec un timing lorsque l'état d'un éclairage change :

```
Lumieres 20211029 - (c) Julie Dumortier - Licence GPL
```

```
HW RESET -> INIT Séquence : 2 - Normal
START
RUN CLEAR SET 10s cmd:  [ _ _ X _ _ _ _ _ _ _ ]
t0.1 [ _ _ X _ _ _ _ _ _ _ ]
t0.3 [ _ _ _ _ _ _ _ _ _ _ ]
t0.5 [ _ _ X _ _ _ _ _ _ _ ]
t0.8 [ _ _ _ _ _ _ _ _ _ _ ]
t0.36 [ _ _ X _ _ _ _ _ _ _ ]
t0.39 [ _ _ _ _ _ _ _ _ _ _ ]
t0.43 [ _ _ X _ _ _ _ _ _ _ ]
t0.46 [ _ _ _ _ _ _ _ _ _ _ ]
t0.58 [ _ _ X _ _ _ _ _ _ _ ]
…
t38.83 [ X _ X _ X X X _ _ _ ]
RUN CLEAR PERM cmd: [ _ _ _ _ _ _ X X _ _ ]
RUN CLEAR ALEA NOPE cmd:  [ _ _ _ _ _ _ _ _ X _ ]
t40.2 [ X _ X _ X X X X _ _ ]
RUN CLEAR ALEA NOPE cmd:  [ _ _ _ X _ _ _ _ _ _ ]
RUN CLEAR ALEA NOPE cmd:  [ _ X _ _ _ _ _ _ _ _ ]
RUN CLEAR WSTOP 3 pin: 1
…
Glitch led:5
t41.7 [ X _ X _ X X X X _ _ ]
t41.10 [ X _ X _ X _ X X _ _ ]
t41.14 [ X _ X _ X X X X _ _ ]
t41.17 [ X _ X _ X X _ X _ _ ]
t41.19 [ X _ X _ X X X X _ _ ]
t41.26 [ X _ X _ X _ X X _ _ ]
t41.29 [ X _ X _ X X X X _ _ ]
t41.48 [ X _ X _ X X _ X _ _ ]
t41.56 [ X _ X _ X X X X _ _ ]
...
```

Les routines en charge de cet affichage fonctionnel sont ***printCmd()*** et ***displayLeds()***. La première affiche les leds impactées par une commande, la seconde affiche l'état courant des leds avec le temps écoulé depuis le démarrage de l'Arduino, cette dernière n'affichant rien si d'un appel sur l'autre l'état courant des leds n'a pas été modifié.

Ces deux fonctions permettent de tester le logiciel sans avoir à connecter la moindre led aux sorties, l'allumage ou l'extinction d'une sortie se faisant systématiquement et respectivement par ***set()*** et ***unset()***, fonctions qui programment la sortie avec les fonctions ***analogWrite()*** ou ***digitalWrite()*** de l'Arduino, suivant le type de sortie puis effectuent une trace de l'opération.

```
// ------------------------------------------------------------------------
// Fonctions de support pour allumer / éteindre une led
// ------------------------------------------------------------------------

void unset(int led)
{
  switch (outputMode[led]) {
    case MODE_IO: digitalWrite(2+led,LOW); break;
    case MODE_PWM: analogWrite(2+led,0); break;
    default: break;
  }

  if (verbose) {
    int pos = (1 << led);
    mapLeds &= ~pos;
    displayLeds(true);
  }
}
```

```
void set(int led, int value)
{
  if (value==LIGHT_OFF) {
    // indicateur que la séquence est vivante !
    digitalWrite(LED_BUILTIN,HIGH);

    unset(led);
    return;
  }

  // indicateur que la séquence est vivante !
  digitalWrite(LED_BUILTIN,LOW);

  switch (outputMode[led]) {
    case MODE_IO: digitalWrite(2+led,HIGH); break;
    case MODE_PWM: analogWrite(2+led,value); break;
    default: break;
  }

  if (verbose) {
    int pos = (1 << led);
    mapLeds |= pos;
    displayLeds(true);
  }
}
```

Une lecture attentive de ce code pas très compliqué vous révélera que nous faisons de l'activité sur la led D13 dite BUILTIN, c'est-à-dire la led intégrée à la carte Arduino Nano. C'est pratique de voir visuellement qu'il se passe quelque chose :) …

A3 - Séquences de mise au point

Le programme est livré avec deux séquences permettant la mise au point du logiciel mais qui peuvent aussi être utilisées pour tester le câblage du bâtiment.

Les deux séquences sont des chenillards. Pour les activer, il suffit de se mettre en mode debug en positionnant la constante ***debug*** à la valeur 1.

Le premier chenillard est sur trois sorties D3, D4 et D5 pour tester les trois types d'éclairage ainsi que le bon fonctionnement du PWM sur D4 et D5. Il y a aussi un gyrophare pendant toute la séquence sur D11.

```
/* Séquence 1 : un simple chenillard sur les sorties D3, D4 et D5 , c'est bien ! */
const int seqDebug1[] = {
   0,      0,  MARK, /* place une MARK pour le LOOP */
   512,    0,  PERM, /* Allume D11 le gyrophare */
   2,      10, SET,  /* Allume D3 (éclairage standard) pendant 10 secondes */
   4,      10, SET,  /* Eteint D3 et Allume D4 (néon neuf) pendant 10 secondes */
   8 ,     10, SET,  /* Eteint D4 et Allume D5 (néon ancien) pendant 10 secondes */
   0,       5, WAIT, /* pause pendant 5 secondes, ne touche pas au lumières
allumées */
   0,      10, SET,  /* Eteint D5 et attends 10 secondes */
   2+4+8 , 10, SET,  /* Allume D3, D4 et D5 pendant 10 secondes */
   512,     0, UNSET,/* Arrete le gyrophare */
   0,      0,  LOOP  /* Tout est éteint. Boucle sur le début MARK */
};
```

Le second sur l'ensemble des sorties pour tester l'ensemble du bâtiment. C'est la séquence la plus intéressante pour mettre au point sa maquette.

```
/* Séquence 2 :  un chenillard sur l'ensemble des sorties, c'est encore mieux ! */

const int seqDebug2[] = {
   0,      0,  MARK, /* place une MARK pour le LOOP */
   1,      10, SET,
   2,      10, SET,
   4,      10, SET,
   8,      10, SET,
   16,     10, SET,
   32,     10, SET,
   64,     10, SET,
   128,    10, SET,
   256,    10, SET,
   512,    10, SET,
   0,      10, SET,   /* Eteint tout pendant 10 secondes */
   1023,   10, SET,   /* Allume tout pendant 10 secondes */
   0,      0,  LOOP  /* Tout est éteint. Boucle sur le début MARK */
};
```

Le choix de la séquence de debug se fait avec l'entrée prévue à cet effet, à savoir D15 ***seqPin***. Étant en pull-up, le fait de laisser cette entrée en l'air active automatiquement la deuxième séquence, le chenillard complet.

A4 - Séquence aléatoire

La commande ALEA introduit de l'aléatoire dans les séquences, c'est bien pour l'authenticité, mais cela peut compliquer la recherche de problèmes dans le programme.

La trace affiche le germe (seed) utilisé, ce qui permet ensuite de rejouer exactement les mêmes tirages aléatoires en remplaçant la ligne "randomSeed(analogRead(7));" par "randomSeed(seed utilisé);".

En variante opérationnelle, vous pouvez connecter l'entrée A7 à la masse pour forcer la séquence aléatoire utilisée par le générateur.