Avec l'aide de l'intelligence artificielle

QUATRIÈME PARTIE : LES SIGNAUX LUMINEUX

EX_MACHINA

Un réseau automatisé

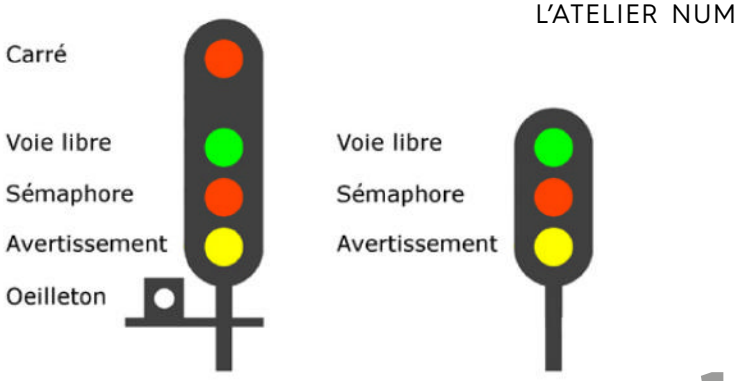
Ce mois-ci, nous allons allumer nos signaux lumineux. Et quel plaisir de voir cette signalisation changer en fonction du déplacement des trains ! Nous n'allons construire que de simples signaux de démonstration que vous pourrez remplacer au fur et à mesure de vos moyens.

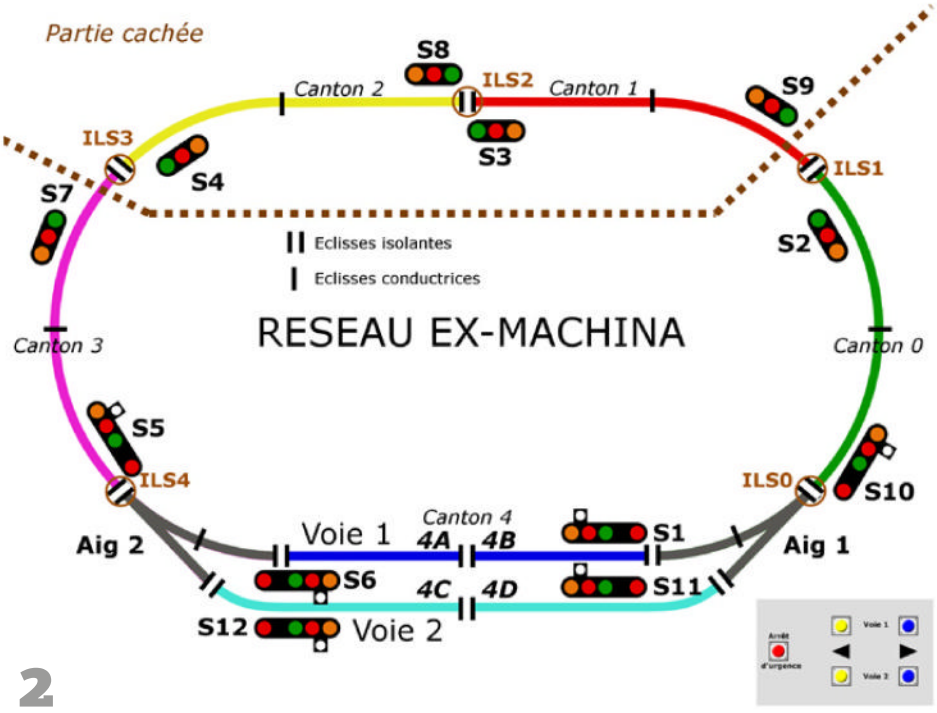
Texte et illustrations: Christian Bézanger

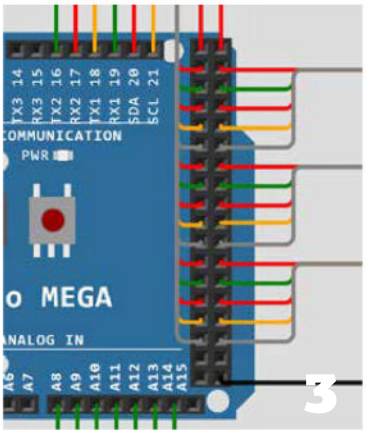
Il y a deux types de signaux sur notre locodrome : des signaux de type Carré qui vont protéger les aiguilles (pour éviter qu'un train ne s'engage si l'aiguille n'est pas dans la bonne position) et des signaux à trois feux de type B.A.L (Block Automatique Lumineux) qui permettent l'espacement des trains sur une même ligne. La **figure 1** montre ces deux types de signaux : on remarque que le Carré possède un œilleton (blanc) qui permet de lever le doute au cas où le conducteur ne voit qu'un seul feu rouge. En effet, est-ce que ce feu rouge correspond au sémaphore ou bien est-ce un carré dont la lampe supérieure serait hors service ? En conséquence, l'œilleton est allumé en permanence sauf si les deux feux rouges du signal carré sont allumés. Le carré permet donc de protéger une zone de voies et n'est jamais franchissable, alors que le sémaphore peut, dans certaines conditions, être franchi ; ce cas de figure ne se posera pas sur notre locodrome puisque le sémaphore entraîne la coupure du courant sur le canton précédent, ce qui provoque l'arrêt du train.

Implantation

La **figure 2** montre l'ensemble des éléments du réseau EX_MACHINA : on reconnaît le tableau de commande, les différents cantons, la position des éclisses isolantes et des éclisses conductrices, les I.L.S et surtout la position et le type de signaux lumineux si on veut les mettre tous. Cette signalisation







lumineuse est constituée d'une signalisation en entrée et sortie de gare (de type carré) dépendant de la position des aiguilles et une signalisation de pleine voie de type B.A.L à trois feux dépendant de l'occupation des cantons. Un train qui part de la voie 1 et parcourt le circuit en sens antihoraire, rencontre le carré S1 (protégeant l'aiguille Aig1), puis les signaux de B.A.L S2, S3, S4, puis le carré S5 protégeant l'aiguille Aig2. Dans le sens horaire, le train rencontre S6 (protégeant l'aiguille Aig2), puis les signaux de BAL S7, S8, S9, puis le carré S10 protégeant l'aiguille Aig1. Les carrés peuvent afficher une signalisation de type B.A.L si l'aiguille est bien orientée et en fonction des états d'occupation des cantons suivants ; par exemple, si l'aiguille 1 est en position déviée, le carré S1 est de type BAL et sa couleur dépend des cantons 0 et 1. Deux autres carrés (S11 et S12) jouent des rôles analogues à S1 et S6 mais pour la voie 2.

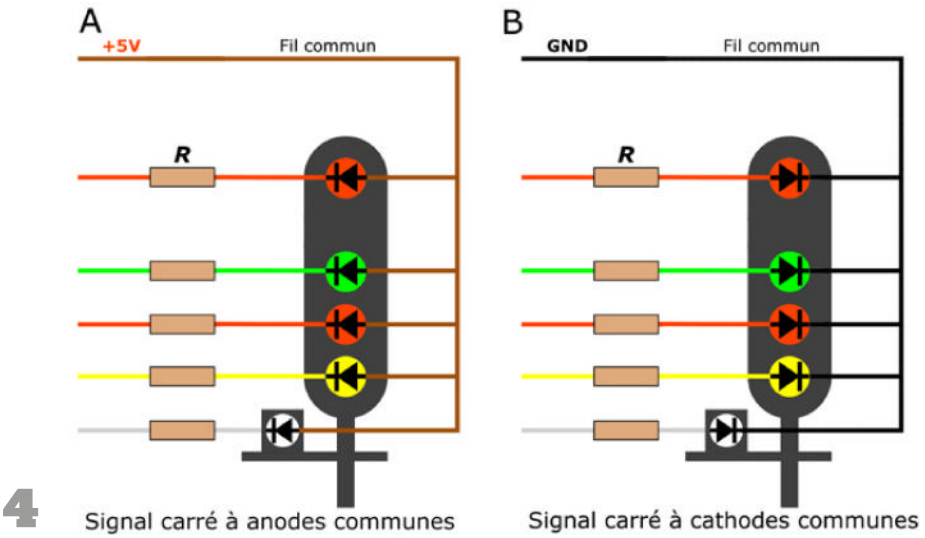
Un peu d'ordre

Comme on l'a déjà dit, la signalisation lumineuse est la partie du projet qui consomme le plus de ressources. Si on voulait mettre tous les signaux de la **figure 2**, il faudrait 48 E/S. En fait, la partie arrière d'un réseau, surtout s'il est bouclé, est souvent cachée par un fond de décor ou un diviseur scénique afin de ne pas voir les trains tourner en rond. Sur la figure 2, ce fond de décor est représenté par une ligne en pointillés, ce qui fait que les signaux S3, S4, S8 et S9 ne sont pas visibles ; on va donc s'en passer. Il nous reste alors six signaux de type Carré et deux sémaphores de B.A.L, ce qui représente 36 sorties pour commander les lampes. Comme déjà mentionné, ces sorties sont des sorties digitales de la cartes qui se suivent : elles sont comprises entre 16 et 51 inclus. De plus, les fils d'un même signal doivent se retrouver du même côté du connecteur et dans un ordre qui est toujours le même : celui des lampes de haut en bas (R, V, R, O, B pour un carré et V, R, O pour un B.A.L). La **figure 3** montre cet ordonnancement : pour des raisons de lisibilité, un fil gris alimente la lampe blanche de l'œilleton. De plus, pour ne pas surcharger le montage en nombre de fils, on a mis les fils d'un même signal les uns au-dessus des autres comme pour constituer un bus (ensemble de plusieurs fils ayant la même fonction) comme on peut le voir sur la partie droite de la figure.

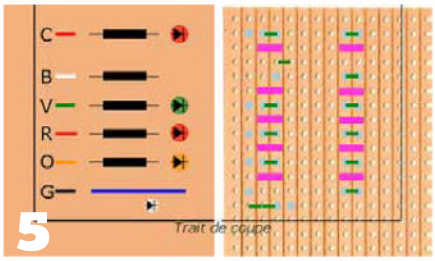
Si vous achetez vos signaux dans le commerce, vous pouvez choisir des signaux à anodes communes ou bien à cathodes communes, mais tous les signaux doivent être du même type. Il suffit alors de modifier le programme pour définir ce qui caractérise l'état (HIGH ou LOW) qui permet d'allumer le signal. On rappelle tout de même que **chacune des LED doit être protégée par une résistance de limitation de** ▸▸

Le tableau suivant résume les connexions des signaux :

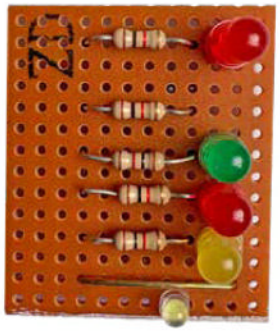
Connexions des signaux	Signal	Type	Nombre de lampes	Sorties
S2	B.A.L	3	16, 17, 18	
S7	B.A.L	3	19, 20, 21	
S1	Carré	5	22 à 30 (pair)	
S5	Carré	5	32 à 40 (pair)	
S11	Carré	5	42 à 50 (pair)	
S6	Carré	5	23 à 31 (impair)	
S10	Carré	5	33 à 41 (impair)	
S12	Carré	5	43 à 51 (impair)	



4



5

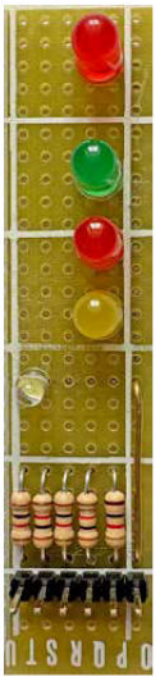


6

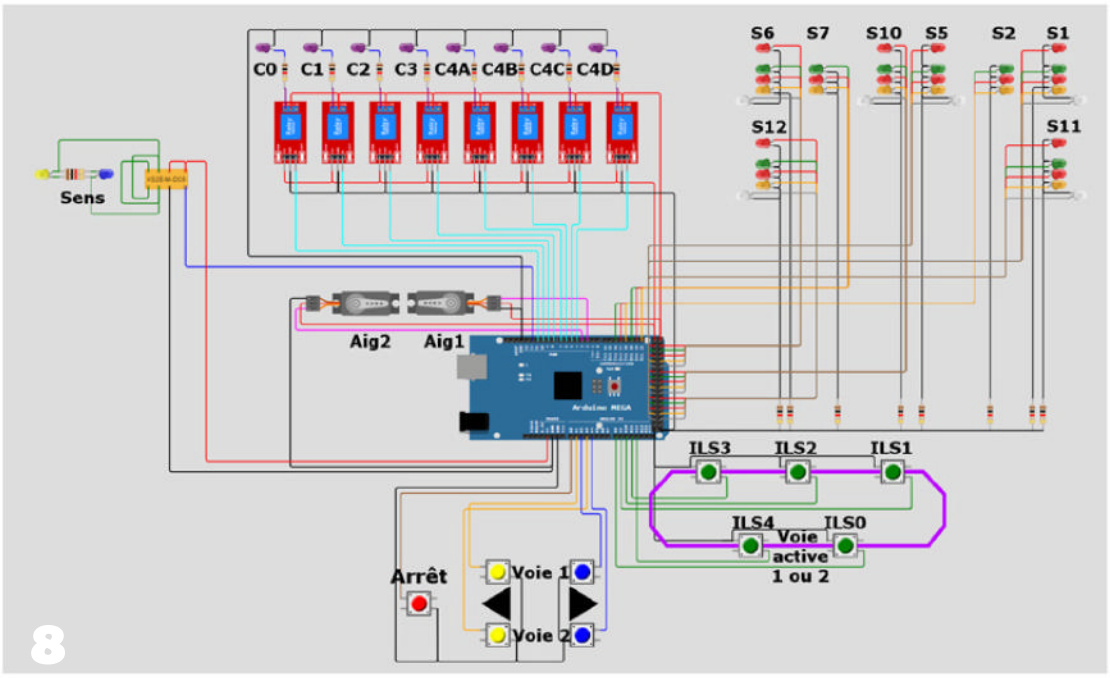
▸▸ **courant** (470 ohms est une assez bonne valeur mais on peut ajuster la luminosité de chaque couleur si on le souhaite) comme le montre la figure 4. Selon la nature des signaux, le fil commun de retour se branche soit à la masse (GND), soit à la tension d'alimentation (+ 5V). La consommation totale de la signalisation lumineuse doit être estimée afin de s'assurer qu'elle ne dépasse pas l'intensité maximale que peut fournir la carte Arduino. Chaque signal de type Carré présente en permanence deux feux allumés et comme il y a six signaux, cela représente 12 LED. Chaque signal de type BAL a une seule LED allumée en permanence, il y a deux signaux. Nous arrivons donc à un total de 14 LED allumées en permanence. Si on fait passer 10 mA dans chaque LED, la consommation est de 140 mA, ce qui est inférieure à la limite délivrée par la carte Mega (200 mA). Pour alimenter les servomoteurs et les cartes relais, il est préférable de prévoir une alimentation séparée de 5 V (dans ce cas, la masse de cette alimentation doit être reliée à la masse (GND) de la carte Arduino).

Construction de signaux de démonstration

Les signaux lumineux du commerce sont magnifiques mais représentent un budget conséquent dès que leur nombre dépasse deux ou trois exemplaires. C'est pourquoi nous allons voir dans ce paragraphe comment construire des signaux bon marché avec un morceau de circuit imprimé à bandes (veroboard), quelques LED et quelques résistances. Ces signaux de démonstration ne seront pas esthétiques mais ils permettront de vérifier que la signalisation lumineuse est pleinement fonctionnelle ; ils pourront être remplacés au fur et à mesure de vos moyens. Les explications qui suivent permettent de construire des signaux à cathodes communes mais vous pouvez opter pour des signaux à anodes communes si vous en possédez déjà quelques-uns de ce type. La **figure 5** (partie gauche) montre comment disposer les LED et les résistances sur le circuit imprimé à bandes pour un signal de type carré. Même chose pour un signal B.A.L qui a deux LED et deux résistances en moins. La partie droite de la **figure 5** montre les pistes (12 x 14 trous) ; commencez par faire les coupures de pistes (en magenta sur la figure) avec une petite fraise, puis soudez les composants (cinq LED (attention à l'orientation), cinq résistances, un fil (bleu sur la figure) : les soudures sont représentées en gris clair). Pour relier le signal à la carte, on peut utiliser une nappe de six câbles Dupont qui peut s'enficher directement sur la carte Mega ; soudez chaque brin en repérant les couleurs (C pour le feu du carré, B pour l'œilleton, V pour vert, R pour rouge, O pour orange et G pour la masse GND). Réalisez ensuite les ponts de soudure (en vert sur la figure) pour relier deux morceaux de pistes adjacents. Pour les résistances, 330 Ω est la valeur minimale pour limiter le courant à 10 mA dans les LED (6 mA dans l'œilleton) ; si la luminosité vous convient, il vaut mieux mettre des résistances de 1000 Ω (marron, noir, rouge) afin de s'éloigner encore plus des limitations de la carte. Pour de vrais signaux, vous pouvez faire des essais afin de bien régler la luminosité de chaque feu, mais ne descendez pas en dessous de 330 Ω. La **figure 6** montre le résultat obtenu même s'il manque encore la nappe de câbles qui permet de raccorder le signal à la carte Mega. Testez chaque LED du signal avec une tension de 5 V, le GND étant relié au fil métallique, le 5 V étant sur le côté gauche des résistances (voir **figure 6**) ; les cinq LED doivent pouvoir s'allumer. Vous pouvez aussi utiliser un support à pastilles et faire un signal plus haut et moins large ; dans ce cas, soudez les composants puis reliez toutes les cathodes ensemble avec un pont de soudure entre pastilles. Reliez les résistances aux anodes avec des fils et terminez en soudant les six brins de la nappe de câbles. Cette méthode paraîtra



7



8

plus simple à certains et plus compliquée à d'autres (**figure 7**).

Traitement des signaux lumineux

Ce traitement n'a rien de compliqué : le carré doit être mis chaque fois qu'une aiguille n'est pas dans la bonne position pour le passage du train. C'est aussi le cas lors des manœuvres d'aiguilles et il faudra penser à rajouter ce traitement dans la fonction qui positionne les aiguilles. En dehors de cela, le signal est traité comme un B.A.L et dépend donc des deux cantons qui le suivent. Si ces deux cantons sont libres, le feu est vert. Si le canton qui suit est libre mais pas le canton d'après, le feu est orange. Enfin, dès que le canton qui suit est occupé, le feu est impérativement rouge. L'IA, connaissant la façon dont les signaux sont connectés, m'a proposé une fonction (setSignal) pour traiter un signal de type carré : il suffit de préciser la position du premier fil du signal (un rouge) et le visuel qu'on veut afficher (voie libre, carré, avertissement, etc.) sous la forme d'une chaîne de caractères. Je m'en suis inspiré pour écrire une fonction pour les signaux de type B.A.L (setBAL). Grâce à ces fonctions, la partie du programme qui traite la signalisation occupe beaucoup moins de lignes de code, ce qui prouve que l'IA peut être performante. Le traitement de la signalisation lumineuse ne devrait pas vous poser de problèmes. Il faut commencer par définir l'état qui allume la LED du signal :

```
// Les signaux lumineux (sorties D16 à D51)
// ----------------------------------------
const byte etatLedON = HIGH; // Mettre LOW si signaux a anodes communes
const byte etatLedOFF = LOW; // Mettre HIGH si signaux a anodes communes
```

Ensuite, il faut définir les positions des premiers fils (appelées basePin par l'IA) des différents signaux :

```
// Connexions des signaux
// ----------------------
const byte S1 = 22;
const byte S2 = 16;
const byte S5 = 32;
const byte S6 = 23;
const byte S7 = 19;
const byte S10 = 33;
const byte S11 = 42;
const byte S12 = 43;
```

Enfin, vu qu'on a organisé les branchements des signaux de manière intelligente, les fonctions du setup sont réduites à leur plus simple expression :

```
// Programme les feux des signaux en sortie
  for (int j = 16; j < = 51; j ++){
    pinMode (j, OUTPUT);
  }
```

L'IA a ensuite créé une fonction affichageSignaux qui affiche le bon visuel en fonction des états des différents cantons selon les règles fixées plus haut. Voici quelques précisions sur l'écriture du code par l'IA. Les tests if sont là pour vérifier qu'une condition est vraie pour que le test s'exécute ; on peut écrire if (occupationCanton [0] == true) ou bien if (occupationCanton [0]) car c'est la même chose (soit on teste une égalité, soit on teste une grandeur qu'on sait être égale à true (vrai)). Et si occupationCanton [0] vaut false (canton libre), alors !occupationCanton [0] vaut true (vrai) puisque c'est son contraire. L'expression if (!occupationCanton [0] && occupationCanton [1]) est équivalente à if (!occupationCanton [0] == true && occupationCanton [1] == true) ou bien encore if (occupationCanton [0] == false && occupationCanton [1] == true). Dans ce cas, on teste que le canton est libre (false) et que le canton suivant est occupé : le feu doit être orange. Le point d'exclamation sert à prendre le contraire de la valeur de la variable booléenne qui ne peut avoir que les valeurs true ou false. Toutes ces façons d'écrire sont équivalentes : adoptez celle qui vous paraît la plus simple ou la plus logique pour vous.

La **figure 8** montre à quoi ressemble le simulateur quand on a ajouté les poussoirs et les LED des signaux. Afin de ne pas surcharger le montage, on a mis une seule résistance sur le fil commun des signaux ; cela fonctionne sur le simulateur Wokwi, mais ne fonctionnerait pas en réalité car la LED blanche nécessite souvent 3 V à ses bornes pour éclairer, ce qui rend impossible son allumage en même temps que l'allumage d'une autre LED. Il est donc nécessaire de mettre **une résistance par LED,** comme expliqué plus haut. Complétez votre simulateur avec les LED et utilisez-le pour la mise au point de votre programme, élaboré avec votre IA préférée.

Conclusion

À ce stade de l'aventure, vous êtes en possession d'un locodrome complètement équipé et grâce à son panneau de commande, vous choisissez le train au départ et son sens de circulation. La signalisation est fonctionnelle et vous pouvez même décider de lancer les deux trains successivement dans la même direction et voir comment ils la respectent ; les deux trains se suivent et sont espacés par le B.A.L. Et si les deux trains devaient partir dans des sens opposés ? Laissons à Arduino le soin de gérer tout cela. C'est ce que je vous proposerai dans le prochain article. ■