Avec l'aide de l'intelligence artificielle

DEUXIÈME PARTIE : LA COMMANDE DU LOCODROME

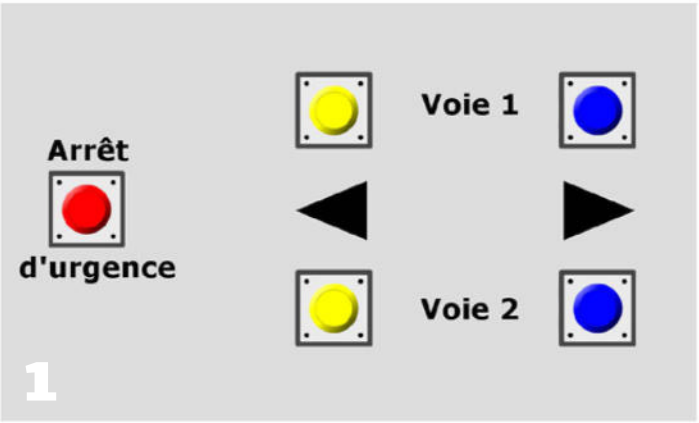
EX_MACHINA

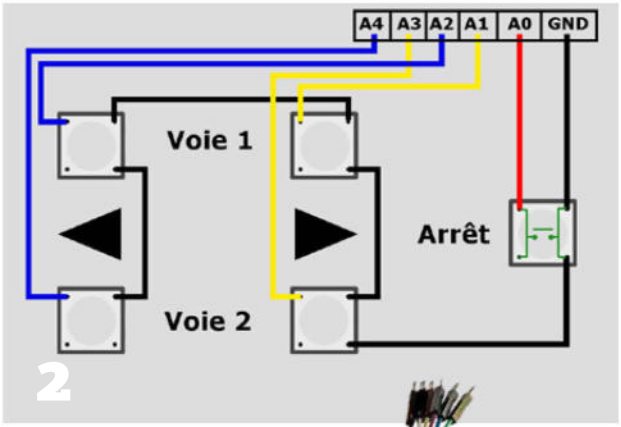
Un réseau automatisé

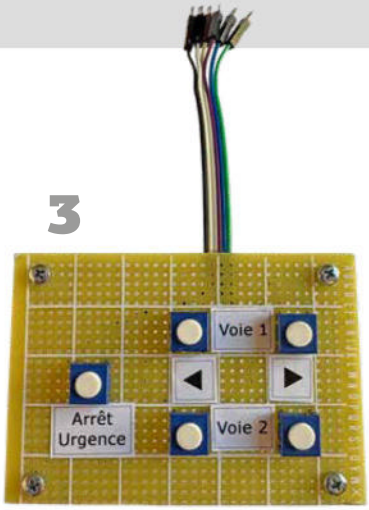
Ce mois-ci nous allons construire le tableau de commande de notre locodrome. Pour commencer simplement, nous allons juste choisir la voie de départ des trains et leur sens de circulation. Pour cela, nous aurons installé la motorisation des aiguilles et les cartes relais pour alimenter les cantons.

Texte et illustrations: Christian Bézanger

Le but de cet article est de faire circuler un seul train, en choisissant sa voie de départ et son sens de circulation. Comme vous voyez, cet objectif n'est pas très ambitieux ; il sera temps ensuite de traiter les deux trains en même temps. Un tableau de commande doit nous permettre de choisir un des deux trains en gare (voie 1 ou voie 2) et le sens dans lequel on veut qu'il parcourt le locodrome. Deux voies et deux sens possibles, cela fait quatre possibilités. Le tableau de commande sera donc muni de quatre boutons poussoirs surveillés par le programme ; en fonction de celui qui est appuyé, le programme déduit les paramètres du mouvement sous la forme de deux variables : voieActive et sensCircul. Un cinquième poussoir permet un arrêt d'urgence : s'il est appuyé, le programme coupe toutes les alimentations de cantons (et éteindra ultérieurement la signalisation lumineuse pour indiquer qu'il y a un problème). Dans le quatrième article de la première série, nous avons vu comment connecter un bouton poussoir (B/P) à une carte Arduino et comment lire l'état de ce B/P. Nous ne reviendrons pas dessus et nos cinq B/P seront placés entre la masse (GND) et les broches A0 à A4 de la carte en déclarant ces broches en INPUT_PULLUP. Bien évidemment, c'est l'IA qui va écrire une fonction pour analyser quels poussoirs sont enfoncés et en déduire la valeur des deux variables. La valeur de sensCircul permet ⏩







▸▸ de régler la polarité sur le feeder de courant de traction grâce à un relais inverseur. La valeur de voieActive permet de positionner les aiguilles pour connecter la bonne voie de la gare sur la boucle du circuit.

Fabrication du tableau de commande

La **figure 1** montre la physionomie du tableau de commande. Chaque poussoir a quatre broches s'il est carré : les broches sont reliées entre elles deux par deux et il faut donc les repérer (si votre poussoir est rond et n'a que deux broches, le problème ne se pose pas). Commencez par souder chaque B/P sur une carte à pastilles ou à bandes : le quadrillage de trous permet de bien les aligner. La **figure 2** montre le verso de la carte ; les broches reliées doivent être alignées le long du petit côté de la carte (voir ce qui est en vert sur la figure). Ensuite, soudez les fils comme le suggère la **figure 2**. Six fils tirés d'une nappe de câbles Dupont, sont à relier à la carte Arduino. Lorsque tous les fils sont soudés, maintenez la nappe à la carte avec un point de colle. La **figure 3** montre le tableau de commande prêt à être installé sur le locodrome.

Inversion du courant sur le feeder

Pour inverser le courant sur le feeder, il vous faut un relais 5 V DPDT (Double Pole Double Throw) qui dispose de deux connecteurs à trois sorties (COM1, NC1, NO1 et COM2, NC2, NO2). On trouve sur internet de petites cartes comportant ce type de relais avec les broches pour les alimenter (en 5 V) et commander le relais (broche I pour Input ou T pour Trigger ou déclencheur). Si le relais est de type monostable, l'état (soit HIGH soit LOW) de la broche T correspond à l'état du relais (soit repos soit travail) et réciproquement ; c'est ce genre de relais qu'on trouve sur le simulateur Wokwi. Si le relais est de type bistable à verrouillage, alors c'est une petite impulsion (100 ms par exemple, généralement LOW) qui fait basculer le relais de la position repos à la position travail et réciproquement, et cette position est ensuite conservée (à la mise sous tension, le relais se trouve en position repos). Préférez le premier type de relais car son état (repos ou travail) est directement lié à l'état de la sortie qui le commande (LOW ou HIGH) et on sait ainsi toujours où on en est, contrairement au relais bistable à verrouillage qui peut se désynchroniser pour une raison ou une autre (bug, parasite, perte temporaire de tension d'alimentation). N'utilisez pas de relais bistable à double bobinage qui fonctionne un peu comme les aiguillages à solénoïdes. Ce relais doit être placé entre l'alimentation à rhéostat délivrant le courant de traction et le feeder (rouge et bleu) que nous avons placé sur le locodrome, comme le montre la **figure 4**.

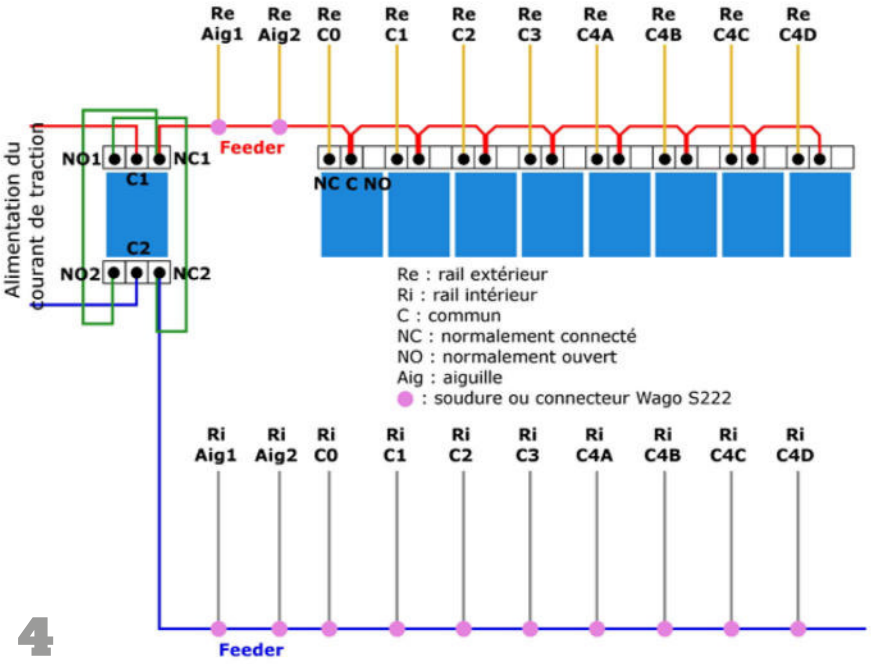
Motorisation des aiguilles

Nous avons choisi des servomoteurs pour motoriser les aiguilles parce que l'écosystème Arduino sait parfaitement les gérer. Dans ce cas, on peut utiliser deux kits de motorisation Decapod réf. 6171 (**figure 5**) et régler dans le programme la valeur des positions extrêmes qui dépendent de la géométrie des aiguilles (échelle et écartement).

Je vous conseille d'acheter également la référence 6170 qui permet de trouver le point milieu de tous vos servomoteurs. Ce point milieu correspond à la position 90° et le palonnier du servomoteur se promène d'un certain angle autour de ce point milieu (par exemple 20°, soit entre 70° et 110 °, les valeurs exactes étant à trouver en fonction de la géométrie de vos aiguilles). Le support du servomoteur a une hauteur de 49 mm ; le même espace doit donc exister sous le plateau de voies.

Le montage des kits ne présente pas de difficulté ; il suffit de suivre la notice. Il n'est pas nécessaire de monter les micro-switches pour ce projet. Par contre, il vous faudra deux rallonges pour relier les servomoteurs à la carte Arduino : vous pouvez les fabriquer vous-même à partir d'une nappe Dupont mâle-mâle coupée en son milieu et de trois fils soudés sur ces deux demi-nappes pour rallonger le tout. De la gaine thermo-rétractable permet de protéger les points de soudure.

Une fois la position milieu réglée avec la référence 6170, le palonnier est monté sur l'axe mais comme celui-ci est cranté, il ne sera pas forcément bien vertical. Ce n'est pas grave, le programme va compenser cette petite erreur. Les valeurs exactes de débattement du palonnier pour chaque aiguille, peuvent être trouvées avec le matériel que vous possédez déjà ; pour cela, utilisez votre carte Arduino Mega (voir ci-dessous) et votre tableau de commande (ci-dessus) pour déterminer précisément ces positions extrêmes (qui peuvent être différentes d'une aiguille à l'autre). Le programme, qui devra être temporairement téléversé dans la carte Arduino, se trouve à cette adresse : https://wokwi.com/projects/426385948648687617. Ces valeurs dépendent du déplacement de la traverse mobile, des épaisseurs de plateau et des traverses de la voie, et du montage du support (position du pivot et choix du trou dans le palonnier). Le tableau ci-dessous résume les valeurs trouvées sur mon réseau : si l'aiguille 1 a bien son point milieu à 90°, il



4

5
Decapod

n'en est pas de même de l'aiguille 2, ce qui peut s'expliquer par le montage du palonnier et le jeu entre l'aiguille et la tige de commande. Ce qui compte, ce sont les valeurs trouvées.

	Voie 1	Voie 2
Aiguille 1	112	68
Aiguille 2	59	102

Choix de la carte Arduino

Pour choisir la carte, il faut connaître le nombre d'entrées-sorties (E/S) nécessaires. Notre tableau de commande en réclame **5** puisqu'il est constitué de 5 boutons poussoirs. La carte 8 relais qui gère l'alimentation des cantons, nécessite **8 E/S** pour être commandée, mais il faut une **E/S supplémentaire** pour une carte relais de type DPDT qui gère le sens de circulation en inversant le courant sur le feeder. L'occupation des cantons nécessitera autant d'E/S qu'il y a de zones, soit **8**. Il faut **2** E/S pour les deux servomoteurs des aiguilles. Mais c'est la signalisation lumineuse qui consomme le plus d'E/S : un signal de type carré, c'est 5 E/S (carré, voie libre, sémaphore, avertissement, œilleton) et un signal de type B.A.L demande 3 E/S. Pour une signalisation lumineuse complète, c'est 6 signaux de type carré et 6 signaux de type B.A.L. Cela fait donc un total de **48 E/S** pour la signalisation lumineuse. Soit au total 72 E/S, ce qui dépasse déjà la capacité d'une carte Mega. Il y a donc des choix à faire, notamment sur la signalisation lumineuse : tous les signaux ne sont pas indispensables, comme nous le verrons en temps utile.

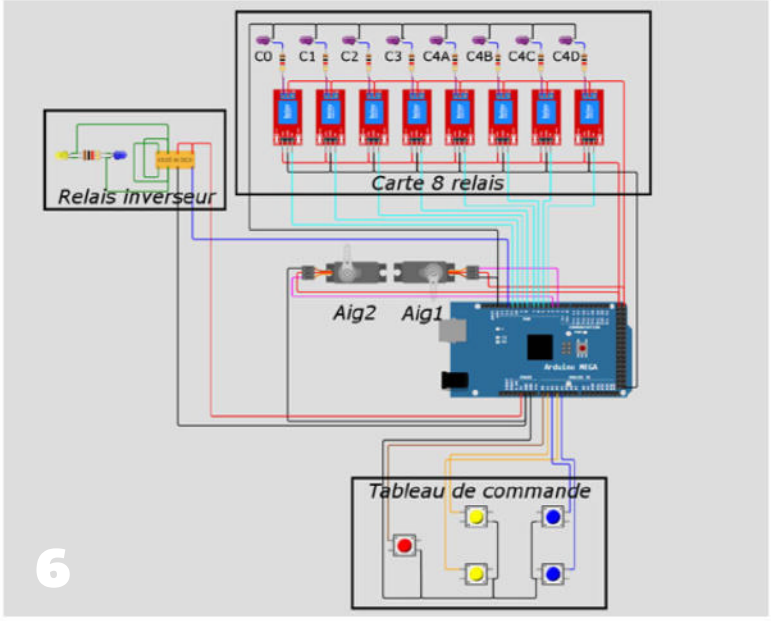
En conclusion, c'est une carte Arduino Mega2560 qui sera le cerveau de notre locodrome et c'est sur cette carte que le tableau de commande, la carte relais d'alimentation des cantons, le relais DPDT d'inversion de la polarité du feeder, les capteurs et tous les signaux lumineux retenus seront raccordés.

Relier les composants à la carte Arduino

Nous devons relier plusieurs éléments à la carte Arduino et il convient de s'organiser un peu. Prenons l'exemple des signaux lumineux : les sorties sur lesquelles ils sont connectés doivent se suivre car on peut alors utiliser une boucle pour les éteindre tous en une seule fois. Nous traiterons la signalisation lumineuse dans un article ultérieur : elle sera reliée aux broches D16 à D51. Cet exemple nous montre que chaque ensemble de composants identiques doit être connecté sur des broches qui se suivent. Nous pouvons consacrer les broches D2 et D3 aux servomoteurs, les broches D4 à D11 à notre carte 8 relais et la broche D12 à la carte relais DPDT inversant le courant sur le feeder. Le tableau de commande est relié aux broches A0 à A4 : ces broches seront considérées comme des broches numériques. À ce stade, il peut être intéressant de réaliser le montage sur un simulateur d'Arduino comme Tinkercad ou Wokwi car cela permet de tester très rapidement les programmes proposés par l'IA. La **figure 6** montre l'ensemble des éléments sur le simulateur Wokwi : on reconnaît le tableau de commande, la carte 8 relais simples, la carte relais DPDT pour inverser le courant sur le feeder et les deux moteurs d'aiguilles. Pour la simulation, les sorties des cartes relais sont reliées à des LED qui permettent de vérifier la polarité et la présence des courants sur les cantons. Le lien ci-dessous vous dirige vers ce montage dans le simulateur Wokwi : la partie gauche vous permettra de coller et tester les programmes proposés par l'IA : https://wokwi.com/projects/422786121722076161. Notez que ce simulateur utilise un relais DPDT monostable dont l'état repos ou travail est déterminé par l'état de la sortie 12 ; ce n'est donc pas une courte impulsion qui fait basculer le relais DPDT (voir plus haut). Ce simulateur sera complété ultérieurement avec les détecteurs qui vont servir à localiser les trains et avec l'ensemble de la signalisation lumineuse.

Premier travail avec l'IA

Maintenant que nous disposons d'un simulateur, nous allons pouvoir faire travailler notre IA et vérifier que ce qu'elle nous délivre est opérationnel. La première chose à faire est de demander à l'IA d'écrire une petite fonction (examConsignes) pour traiter les cinq boutons poussoirs et afficher des messages afin de vérifier ce qu'il se passe. Pour cela, on va lui demander d'utiliser deux variables voieActive et sensCircul dont les valeurs dépendront du bouton poussoir enfoncé. Il faut préciser à l'IA sur quelles E/S les poussoirs sont reliés : ceci évite d'avoir à modifier le programme par la suite. Cette tâche étant relativement simple, l'IA devrait réussir du premier coup et afficher quel train est choisi et dans quelle direction il part. Cela signifie que le programme a déterminé les valeurs des deux variables. ▶▶



▸▸ En fonction de la variable sensCircul, votre IA doit vous fournir une fonction (inversionFeeder) qui commande le relais DPDT pour avoir la bonne polarité sur le feeder afin de respecter le sens de circulation choisi. Testez le programme et observez l'allumage des LED (jaune ou bleue) afin de vérifier que cela est cohérent avec l'affichage du message : la LED allumée doit être de la même couleur que le poussoir manipulé.
En fonction de la variable voieActive, demandez à l'IA d'écrire une fonction (mouvementAiguilles) pour positionner les deux servomoteurs en même temps, en fonction des positions extrêmes que vous aurez déterminées (voir plus haut) et données à l'IA ; celles-ci dépendent de la voie choisie, donc de la variable voieActive. Avec le simulateur Wokwi, vous pouvez choisir comme positions extrêmes 120° et 90° pour le servomoteur 1 et 60° et 90° pour le servomoteur 2, ces valeurs permettent juste de voir un mouvement. Vous devez observer que les servomoteurs se déplacent bien à chaque changement du choix de la voie (inutile de les faire bouger s'il n'y a pas de changement). Si le fonctionnement est cohérent, c'est gagné ! Ces premiers essais sont relativement simples et si votre prompt est bien rédigé et complet, l'IA ne devrait pas échouer. Néanmoins, si ces différents programmes ne fonctionnent pas, décrivez ce que vous observez par rapport à ce que vous voudriez obtenir et expliquez-le dans votre prompt : l'IA corrigera le tir, jusqu'à ce que vous soyez satisfait. Vous pouvez considérer que ces essais sont satisfaisants si la polarité sur le feeder et la position des servomoteurs correspondent à votre choix de train et de sens de parcours.

Gestion de la carte relais

Il ne reste plus qu'à commander le courant sur les voies pour pouvoir utiliser le locodrome (le vrai). Actuellement, le montage est tel que toutes les voies sont alimentées. C'est en commandant un relais qu'on peut couper l'alimentation d'un canton et pour définir quel relais commande quel canton, le mieux est d'utiliser un tableau à une dimension (un seul indice) :

```
// Connexions des relais
// ---------------------
const byte pinRelais [] = {11, 10, 9, 8, 7, 6, 5, 4};  // 8 relais
const byte pinRelINV = 12; // Relais inverseur de courant feeder
```

Rappelez-vous que la numérotation de l'indice de tableau commence toujours par le chiffre 0 (ce n'est pas pour rien si on a numéroté les cantons en commençant par 0). Il faut surtout veiller à ne jamais sortir du tableau à cause d'un indice inapproprié car cela pourrait provoquer l'écrasement d'une donnée en mémoire (difficile de savoir laquelle) et le programme ne fonctionnerait pas. Le tableau suivant établit la correspondance entre numéro de canton, indice de tableau pour le programme et broche commandant le relais :

Numéro de canton	Indice de tableau	Broche
0	0	11
1	1	10
2	2	9
3	3	8
4A	4	7
4B	5	6
4C	6	5
4D	7	4

Maintenant que vous savez comment couper l'alimentation des cantons, vous pouvez demander à votre IA d'écrire un programme (arretURGENCE) pour éteindre tous les signaux (branchés entre les sorties D16 à D51) et couper l'alimentation de tous les cantons : pour le mettre en œuvre, c'est le poussoir rouge ! Demandez également à l'IA de créer une fonction (initLocodrome) afin d'alimenter en courant toutes les voies sauf celles de la gare, puis de créer une fonction (departGare) qui alimentera en courant la voie de départ choisie (variable voieActive).

Organisation du programme final

Comme je vous l'ai dit, il vaut mieux demander à l'IA d'écrire des fonctions que vous pourrez utiliser dans votre programme final. Il faut que ces fonctions soient compatibles entre elles, et pour cela, vous devez donner à l'IA des consignes strictes, comme parfois le choix des variables (nom et type) ainsi que les branchements de chaque composant. Décrivez bien ce que vous voulez obtenir : par exemple, le programme de l'IA pour analyser le tableau de commande a été mis dans la fonction « examConsignes » qui est appelée au début de la fonction loop (mais pas à chaque itération : une seule fois jusqu'à ce que le train revienne en gare et alors seulement on peut recommencer). Par contre, le traitement du poussoir « Arrêt d'urgence » doit se trouver en début de loop afin d'être tout le temps disponible.
Votre rôle est alors celui d'un chef d'orchestre qui organise l'ossature du programme final, le rôle des différentes fonctions à demander à l'IA, l'organisation des ressources, etc. Au début du programme, définissez les variables que l'utilisateur peut modifier (valeurs extrêmes pour le déplacement des servomoteurs qui dépendent de l'échelle et de l'écartement adoptés, l'état (HIGH ou LOW) pour commander les relais, l'état (HIGH ou LOW) pour allumer les LED des signaux selon qu'ils sont à anodes ou cathodes communes, etc.). Définissez ensuite les variables non modifiables par l'utilisateur (broches utilisées par les composants, variables d'état (occupation des cantons, position des aiguilles), variables d'état antérieur (pour repérer s'il y a eu un changement)). Parfois, vous serez obligé de rajouter des variables que votre IA aura elle-même créées.

Quelques conseils et explications

Deux variables sont importantes : la voie au départ (voieActive) et le sens de circulation (sensCircul). La variable voieActive peut prendre les valeurs 1 ou 2, et son type est donc entier ou int. La variable sensCircul est plus subtile puisque ce sens peut être celui des aiguilles d'une montre (sens horaire) ou le sens contraire (antihoraire, encore appelé sens trigonométrique par les mathématiciens). Comme cette variable n'a que ces deux possibilités, on lui attribue la valeur 0 si le sens est horaire et la valeur 1 si le sens est antihoraire. La variable est donc de type entier. Mais on peut aussi la définir comme variable booléenne pouvant avoir les valeurs true (vraie) ou false (fausse). En langage de programmation, la valeur false est représentée par le chiffre 0 alors que la valeur true est représentée par toute valeur non nulle (on peut alors prendre 1), ce qui fait que si au lieu de 0 et 1 on utilise false et true, la variable devient booléenne, mais cela revient au même. J'en parle parce que l'IA utilise souvent les variables booléennes lorsqu'il n'y a que deux valeurs possibles. Si ce concept vous semble difficile, précisez à votre IA d'utiliser des variables entières et non booléennes.

Pour réaliser l'affichage de ce que comprend l'IA quand vous manipulez le tableau de commande, vous avez deux possibilités : soit dans la fonction loop mais à chaque itération on aura le même affichage, ce qui peut être lassant, soit un affichage uniquement si un B/P a été appuyé. Pour obtenir cela, l'IA initialise un « drapeau » (une variable) appelé appuiPoussoir qui vaut 0 tant qu'un B/P n'a pas été appuyé et qui vaut 1 dès qu'un B/P est appuyé. Si ce drapeau est égal à 1, on effectue l'affichage et on remet le drapeau à 0. Si le drapeau vaut 0, c'est qu'aucune action n'a été faite sur les B/P et qu'il n'y a pas besoin d'affichage.

De la même façon, certaines actions ne sont à faire que s'il y a changement de la valeur de certaines variables, par exemple le positionnement des aiguilles en fonction de la voie au départ. Pour le savoir, on compare cette variable à sa valeur précédente stockée dans une variable qui porte le même nom précédé de old (qui signifie vieux, ancien) : par exemple oldvoieActive. On peut alors savoir si la variable voieActive a changé en comparant sa valeur avec la valeur qu'elle avait précédemment, appelée oldvoieActive. S'il y a changement, on lance les actions nécessaires.

Le test if (ou if... else) va nous permettre de tester les B/P. Voici par exemple la façon de surveiller le B/P relié à l'entrée A1 (le B/P jaune situé en haut à gauche). Si ce B/P est appuyé, un signal LOW se retrouve sur la broche A1, ce qui signifie que le joueur a sélectionné la voie 1 et le sens horaire :

```
if (digitalRead (A1) == LOW) {
   voieDepart = 1;
   sensCircul = 0;
   appuiPoussoir = 1;
}
```

Remarquez qu'il y a deux fois le signe = dans la parenthèse et si vous n'en mettez qu'un, l'IDE ne vous préviendra pas (car ce n'est pas une erreur pour lui) mais le programme ne fonctionnera pas comme il devrait.

Généralement, les relais sont commandés par un niveau LOW mais pas toujours. On peut donc, dans le programme, définir à quoi correspond le niveau du signal (HIGH ou LOW) pour commander le relais grâce à un ordre #define. Voici un exemple pour alimenter le canton 3 :

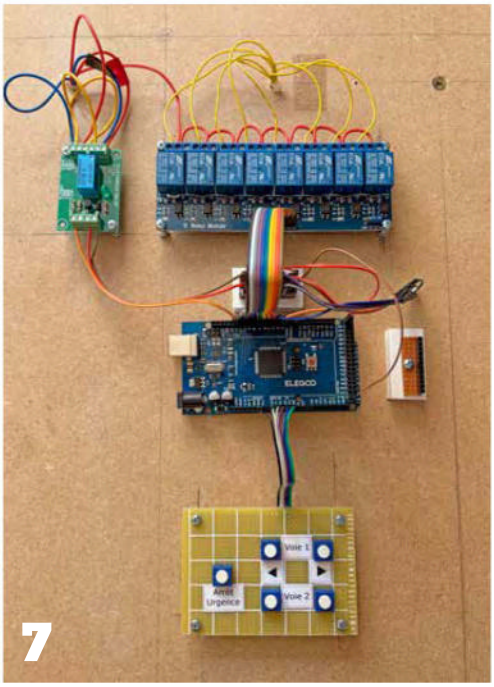
```
#define etatRelON LOW // Mettre HIGH si relais est commande par HIGH
#define etatRelOFF HIGH // Mettre LOW si relais est commande par HIGH

digitalWrite (pinRelais [3], etatRelOFF);
```

La dernière ligne est équivalente à digitalWrite (8, HIGH) ; car pinRelais [3] = 8 et etatRelOFF équivaut à HIGH. Rappelez-vous que lorsque le relais est commandé (etatRelON), il coupe l'alimentation sur le canton. Pour établir cette alimentation, il faut donc ne plus solliciter le relais (etatRelOFF).

Test du locodrome

Le simulateur doit vous permettre de vérifier que les programmes écrits par l'IA commandent bien les différents composants (relais, servomoteurs) du locodrome. Il suffit de cliquer sur un poussoir du simulateur pour observer ce que comprend l'IA grâce à l'affichage des messages et d'observer également comment se positionnent les servomoteurs et quelle est la polarité sur le feeder. Lorsque tout semble fonctionner comme il se doit, il est temps de récupérer le programme et le téléverser dans la carte Arduino Mega.

Une fois la carte Mega programmée, installez-la sur le locodrome et connectez le tableau de commande, les deux cartes relais, ainsi que les deux servomoteurs, comme indiqué par la **figure 6**. N'oubliez pas de déterminer le point milieu des servomoteurs avant de placer le palonnier et la tige de commande.



La **figure 7** montre ce que vous devez avoir sur votre locodrome : carte Mega, tableau de commande, relais inverseur, carte 8 relais, câblage des servomoteurs qui sont sous le plateau.

Vous pouvez alors tester le locodrome. Lors de l'initialisation, les voies en gare ne sont pas alimentées ; positionnez un train sur chaque voie (ils ne doivent pas démarrer puisque le courant est coupé sur les voies). Sélectionnez sur le tableau de commande quel train vous voulez faire partir et dans quel sens. Une fois les aiguilles en bonne position, le train doit démarrer et parcourir le locodrome.

Conclusion

Le but de cet article est atteint ; le locodrome est fonctionnel et peut commander un train grâce au tableau de commande. Dans le prochain article, nous verrons comment repérer la position des trains sur le locodrome et ce que cela implique pour l'alimentation des cantons afin d'arriver à une circulation de type B.A.L (Block Automatique Lumineux) qui permettra à deux trains de se suivre en toute sécurité.

Pour terminer, le lien ci-dessous vous emmènera sur le simulateur Wokwi de la figure 6, mais cette fois avec un programme opérationnel. Les consignes d'utilisation sont dans le commentaire du programme. J'ai écrit la fonction loop moi-même pour utiliser différentes fonctions écrites par l'IA pour le réseau EX_MACHINA. Certaines variables ont été créés par l'IA et d'autres lui ont été imposées. Enfin, pour une meilleure compréhension, j'ai ajouté pas mal de commentaires en plus de ceux de l'IA. J'espère que vous vous amuserez bien. https://wokwi.com/projects/427132586160264193 ■