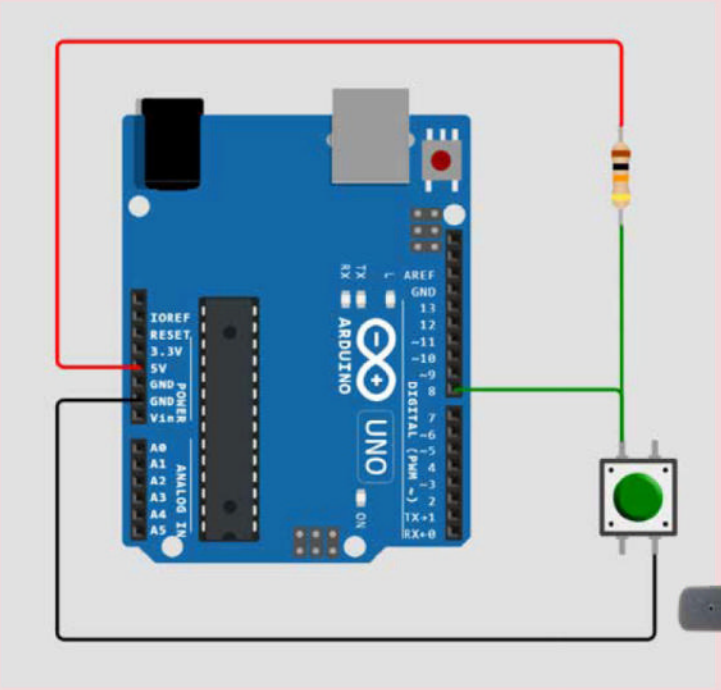

FIGURE 1: CAPTEUR SUR UNE ENTRÉE NUMÉRIQUE.

TROISIÈME PARTIE : DÉCOUVRIR ARDUINO

ARDUINO *et le software*

Le mois dernier, nous avons surtout parlé du hardware (carte Arduino, capteurs, actionneurs et autres composants électroniques). Nous allons aujourd'hui nous concentrer sur le software, c'est-à-dire le programme qui fait fonctionner la carte Arduino : comment lire l'état d'un capteur, comment traiter l'information et comment agir en conséquence sur le réseau.

Texte et illustrations : Christian Bézanger

Tout d'abord, répondons à une question essentielle : c'est quoi un programme ? Un programme est une suite d'instructions que le microcontrôleur exécute les unes après les autres dans le but de réaliser une tâche bien précise. Notez que je n'ai pas dit les unes à la suite des autres ; en effet, dans certains cas particuliers, le microcontrôleur est amené à exécuter certaines instructions prioritairement à d'autres. Ces cas sont d'ailleurs prévus par le programmeur. Retenons simplement qu'un microcontrôleur n'exécute qu'une seule instruction à la fois et que cette dernière est en quelque sorte une tâche élémentaire qu'il sait faire. Concevoir un programme consiste donc à établir une liste de tâches élémentaires à exécuter sans rien oublier pour qu'au final, une tâche plus compliquée soit accomplie. Ce travail d'analyse, qui n'est pas toujours très simple, demande beaucoup de réflexion et se fait sur papier et non devant un ordinateur. Souvent, le programme ne fonctionne pas du premier coup car il a été mal analysé, et des tâches intermédiaires ont été oubliées (rappelez-vous le feu tricolore où on oublie d'éteindre un feu avant d'allumer le feu suivant).
Mais un programme ne se limite pas à exécuter différentes tâches ; il doit aussi manipuler des variables dont la valeur peut changer au cours du programme (c'est pour cela qu'on les nomme variables) et en fonction des valeurs, prendre des décisions pour la suite

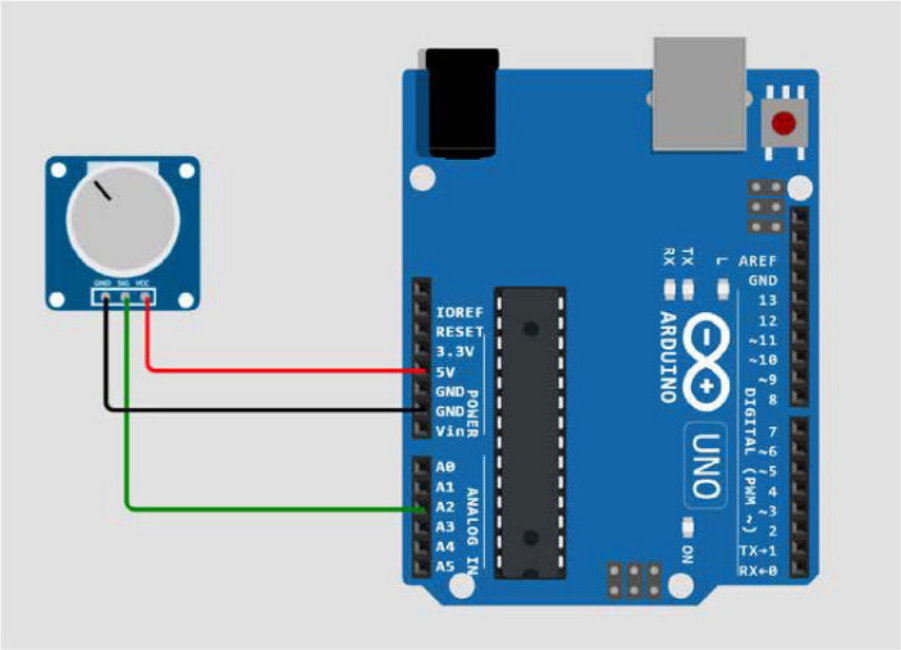

Figure 2: Capteur analogique sur une entrée analogique.

des tâches à accomplir. Prenons un exemple avec un capteur (bouton-poussoir ou ILS) qui se déclenche à l'arrivée d'un train. Ce capteur est relié à une entrée numérique d'Arduino (la 8 en l'occurrence) selon le montage de la **figure 1**.

Tant que le poussoir n'est pas actionné, l'entrée 8 est à 5 V par l'intermédiaire de la résistance (appelée résistance de pull-up car elle maintient l'entrée vers le niveau haut (HIGH ou 5 V)). Lorsque le poussoir est appuyé, l'entrée est directement reliée à la masse (GND) donc la tension sur cette entrée vaut 0 V (état LOW). Pour connaître l'état du bouton-poussoir (appuyé ou non), il suffit de lire la tension présente sur la broche d'entrée. Si le poussoir est remplacé par un ILS (qui se ferme sous l'action d'un aimant équipant la locomotive), l'entrée passe à 0 V (état LOW) lorsqu'un train survole l'ILS. Voici un moyen simple de repérer l'approche d'un train.

Lire une entrée numérique

Une entrée numérique peut être à une tension de 0 V (on parle alors d'état LOW) ou bien de 5 V (on parle d'état HIGH). Une fonction permet de lire la tension sur une entrée numérique, c'est digitalRead (pin) où pin est le numéro de la broche. Cette fonction retourne soit LOW (la tension est de 0 V) soit HIGH (la tension est de 5 V), et ce qui est retourné peut être communiqué à une variable qu'on peut appeler etatPoussoir (voir plus loin).

Lire une entrée analogique

Une entrée analogique peut recevoir une tension continue comprise entre 0 et 5 V, comme c'est le cas pour le montage de la **figure 2** utilisant un potentiomètre. Une fonction permet de convertir cette tension en un nombre compris entre 0 (pour 0 V) et 1023 (pour 5 V); elle s'écrit analogRead (pin) où pin est le numéro de l'entrée analogique (par exemple A2). Le résultat retourné par la fonction (un nombre entier compris entre 0 et 1023) doit être stocké dans une variable qu'on peut appeler valeurTension_A2 et qui permet alors de connaître la valeur d'un capteur analogique (ici le potentiomètre). Avant d'aller plus loin, il convient de dire quelques mots sur la façon dont les variables sont stockées en mémoire.

Trois types de mémoire : Flash, SRAM et EEPROM

Le microcontrôleur qui équipe la carte Uno possède trois types différents de mémoire. Tout d'abord, on trouve de la mémoire Flash prévue pour contenir les instructions du programme (sous forme de langage machine, pas sous la forme de ce que vous écrivez dans l'IDE pour constituer votre programme). Cette mémoire est analogue à celle des clés USB et l'information ne disparaît pas lorsque la carte n'est plus sous tension. On trouve ensuite de la mémoire SRAM, analogue à la mémoire de votre ordinateur. Cette mémoire sert à stocker les informations qui sont nécessaires au programme, mais cette mémoire est volatile et les informations sont perdues lorsque la carte n'est plus alimentée. Pour pallier cet inconvénient, le microcontrôleur dispose aussi d'une mémoire EEPROM plus petite en taille mais qui peut garder l'information lorsque la carte n'est plus alimentée. Cette mémoire est effaçable et programmable électriquement et il existe des fonctions pour lire ou écrire dans cette mémoire.

Petites quantités, donc à économiser !

Un microcontrôleur ne dispose pas d'autant de mémoire qu'un ordinateur mais la quantité est suffisante pour de petites applications, comme par exemple la gestion d'un passage à niveau. Comme toutes mémoires, elle est organisée en octets et on parle de kilo-octets (un kilo vaut 1024 octets). La mémoire Flash du microcontrôleur ATmega328P qui équipe la carte Uno R3 dispose de 32 kilo-octets de Flash; comme une instruction nécessite 2 octets de mémoire, notre programme peut aller jusqu'à 16 kilo-instructions (16384 exactement). La mémoire SRAM est plus réduite puisque sa taille est de 2 kilo-octets seulement et c'est justement cette mémoire qu'il faut savoir économiser, d'autant que le microcontrôleur en utilise déjà un peu pour son propre fonctionnement (la pile qui sert en cas d'appel de sous-programme). La mémoire EEPROM est de 1 kilo-octet, mais en modélisme, on l'utilise assez peu sauf pour éventuellement sauvegarder une configuration particulière du réseau.

Plusieurs types de variables

C'est donc en mémoire SRAM que les informations sont sauvegardées; et suivant la nature de cette information, il faudra plus ou moins d'octets. Un octet est un ensemble de 8 bits égaux à 0 ou 1, s'ils sont tous égaux à 0, l'octet a la valeur 0, s'ils sont tous égaux à 1, l'octet a la valeur 255 (ceci est expliqué plus en détail dans l'article https://locoduino.org/spip.php?article99). On comprend alors que si on veut stocker un nombre entier supérieur à 255, il faudra plusieurs octets. Un simple octet est pourtant largement suffisant ▸▸

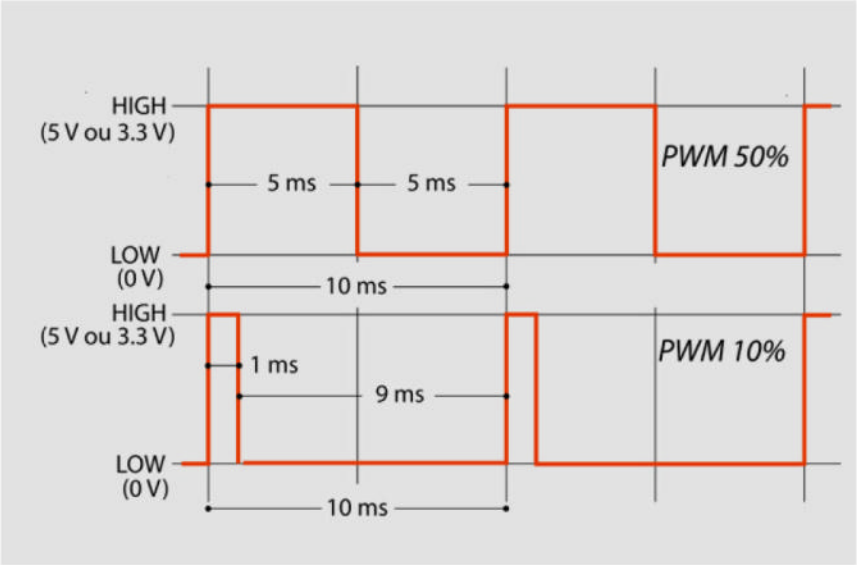

Figure 3:
Signal PWM

▸▸ pour stocker un numéro de broche ou encore un caractère alphanumérique (caractère ASCII). Deux octets permettent de stocker un nombre un peu plus grand; si le nombre est positif (non signé), il peut être compris entre 0 et 65535, par contre s'il est signé (positif ou négatif), il est compris entre -32768 et +32767. Pour économiser la mémoire SRAM, il est nécessaire de choisir le bon type de variable pour stocker l'information voulue. Parmi les différents types de variable, nous avons byte (un seul octet), int (deux octets), long (quatre octets), float (quatre octets mais pour des nombres réels). Choisir un type de variable inapproprié, c'est soit gâcher de la mémoire vive (et on a vu qu'il y en a assez peu), soit risquer des erreurs sur le contenu de la variable.

Définir une variable avec un type et un nom

Dans un programme pour Arduino, les variables doivent être déclarées avant d'être utilisées, sinon on obtient un message d'erreur. Pour déclarer une variable, il faut déclarer son type (ainsi le programme réserve le nombre d'octets nécessaires en mémoire SRAM pour stocker les valeurs de la variable) et il faut lui donner un nom. Ce nom doit être explicite, et ne doit pas contenir de caractères accentués: par exemple, etatCanton1 est un meilleur choix que eC1. De plus, etatCanton1 aurait pu s'appeler etat_canton_1 et si vous utilisez les deux appellations dans un même programme, chacune représente une variable différente et le programme est bien incapable de comprendre que vous voulez parler de la même chose. Soyez donc rigoureux sur le choix des noms de vos variables. Lors de la déclaration de la variable, vous pouvez en profiter pour l'initialiser, c'est-à-dire lui donner une valeur (qui pourra changer au cours du programme) mais ce n'est pas obligatoire. Par exemple:

```
byte nbre_tours_prevus = 3; // Nombre de tours que doit faire l'automatisme (initialisé à 3) byte nbre_tours_observes; // Nombre de tours observés au cours de l'exécution du programme
```

Et si la variable est constante ?

Une variable peut voir sa valeur rester constante pendant tout le temps d'exécution d'un programme. Dans ce cas, il est primordial de la déclarer comme étant constante car ainsi, elle n'occupera pas de place en mémoire ce qui permet d'économiser cette dernière. Bien sûr, pour un petit programme qui utilise peu de SRAM, cela ne paraît pas forcément nécessaire, le programme fonctionnera quand même. Mais il faut prendre les bonnes habitudes dès maintenant, donc faites-le systématiquement. La meilleure façon de déclarer qu'une variable reste constante durant tout le programme est de rajouter le mot const devant sa déclaration. Attention, une fois qu'une variable est déclarée comme constante, on ne peut plus changer sa valeur sinon cela génère un message d'erreur.
Exemple: vous voulez utiliser la sortie 9 de la carte Uno pour y brancher une LED. Au cours de l'exécution de votre programme, le numéro de sortie ne changera pas. Vous pouvez déclarer une variable pinLED sous cette forme: const byte pinLED = 9;

Produire un signal PWM sur une sortie numérique

Un signal PWM (Pulse Width Modulation ou modulation par largeur d'impulsion) est un signal carré périodique alternant des tensions nulles et des tensions égales à la tension d'alimentation du microcontrôleur (5 V sur une carte Uno, 3,3 V sur certaines cartes). La **figure 3** montre un tel signal. On appelle rapport cyclique le rapport entre le temps où le signal est à la tension d'alimentation et la période du signal, le résultat étant exprimé en pourcentage. Par exemple, si la période est de 10 ms et que le temps où la tension est maximale est de 5 ms, alors le rapport cyclique est de 50 %. Ce genre de signal est exploité en modélisme ferroviaire pour faire varier la vitesse d'un moteur à courant continu ou bien l'intensité d'une LED.
Seules certaines sorties permettent de créer un signal PWM: sur carte Uno (R3 ou R4), ce sont les sorties numériques 3, 5, 6, 9, 10 et 11. Pour la carte Uno R3, la fréquence du signal est de 490 Hz, sauf sur les broches 5 et 6 où elle vaut 980 Hz.
La fonction produisant un signal PWM sur ces broches est analogWrite (pin, value) où pin est le numéro de la broche (parmi ceux possibles) et value un nombre compris entre 0 et 255, 0 pour un rapport cyclique de 0 % (signal toujours à 0 V) et 255 pour un rapport cyclique de 100 % (signal toujours au maximum). Le nom de la fonction est trompeur car on obtient bien un signal numérique où deux valeurs de tensions seulement sont possibles; néanmoins, la valeur moyenne de ce signal peut être considérée comme une tension intermédiaire.

Boucles et opérations répétitives

Certaines tâches élémentaires dans un programme peuvent se répéter un certain nombre de fois, et si ce nombre est important, on ne va pas écrire toutes les instructions. On fera appel à une structure de programmation appelée boucle, permettant d'écrire une seule fois les instructions à réaliser. Par contre, la boucle sera parcourue le nombre de fois nécessaire jusqu'à avoir réalisé toutes les tâches à faire. Il y a deux catégories de boucles: les boucles « for » qui utilisent un compteur comptant le nombre d'itérations et les boucles « while » qui sont parcourues tant qu'une condition est réalisée.

Tests permettant de prendre des décisions

Le programme peut être amené à tester une variable afin d'exécuter telle ou telle partie de programme. Par exemple, si un capteur est

UN PROGRAMME SIMPLE POUR COMMENCER

Notez les nombreux commentaires qui commencent par un double slash. Notez également le point-virgule à la fin de chaque instruction. Notez aussi que la condition du if nécessite deux fois le signe égal (si vous n'en mettez qu'un seul, vous n'aurez pas de message d'erreur mais le programme ne fonctionnera pas et ce sera difficile de comprendre pourquoi). Les moteurs lents d'aiguilles sont généralement pourvus de contacts auxiliaires qui permettent de polariser la pointe de cœur ou d'allumer un témoin lumineux de position sur un TCO. Ces contacts peuvent être utilisés comme capteurs pour renvoyer l'information de position aiguille à la carte Arduino et allumer la signalisation en conséquence. Le programme pour gérer cela ressemblerait beaucoup à ce qui vient d'être fait.

```
const byte pinPoussoir_vert = 4;  // Poussoir vert sur entree 4
const byte pinPoussoir_bleu = 5;  // Poussoir bleu sur entree 5
const byte pinLED = 7;            // LED sur sortie 7

void setup() {
  // put your setup code here, to run once:
  pinMode(pinPoussoir_vert, INPUT);  // Entree
  pinMode(pinPoussoir_bleu, INPUT);  // Entree
  pinMode(pinLED, OUTPUT);           // Sortie
  digitalWrite(pinLED, LOW);         // la LED eteinte au depart
}

void loop() {
  // put your main code here, to run repeatedly:
  if(digitalRead(pinPoussoir_vert) == LOW) {  // Poussoir vert appuye
    digitalWrite(pinLED, HIGH);               // LED allumee
  }
  if(digitalRead(pinPoussoir_bleu) == LOW) {  // Poussoir bleu appuye
    digitalWrite(pinLED, LOW);                // LED eteinte
  }
}
```

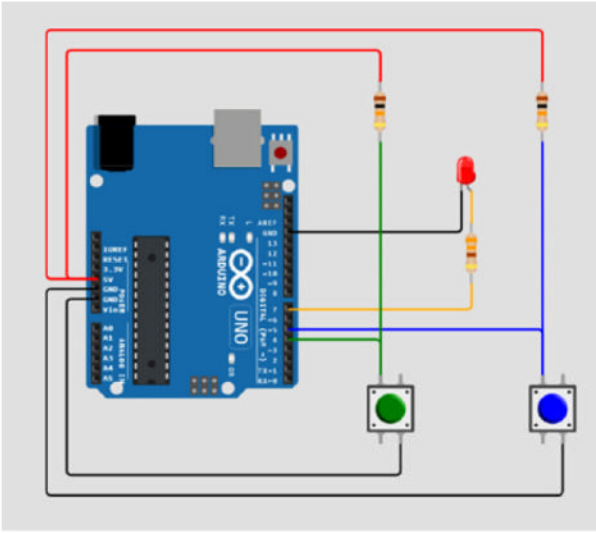

Figure 4 : Montage avec deux capteurs et un actionneur.

déclenché, alors on exécute la partie de programme qui consiste à fermer les barrières d'un passage à niveau. Pour tester la valeur d'une variable, on peut utiliser la structure « if... else ». Si on a un choix à faire en fonction de la valeur d'une variable, on peut utiliser la structure « switch... case ». Tout cela est décrit à la page https://www.arduino.cc/reference/en/et nous y reviendrons ultérieurement.

Quelques exemples d'autres fonctions d'Arduino

Ce qui est décrit plus haut représente les fonctions que vous utiliserez le plus souvent en modélisme ferroviaire, du moins en tant que débutant. Il en existe d'autres, utiles également, ainsi que des structures de contrôle (boucles ou tests par exemple) ; nous y reviendrons. Il y a aussi des fonctions pour aller manipuler directement les bits constituant les octets de données ; quand on en a besoin, c'est qu'on a déjà quitté le stade de débutant. D'autres fonctions permettent de faire des mathématiques ou de la trigonométrie, des calculs de logique avec opérateurs booléens, des comparaisons entre grandeurs, des manipulations de chaînes de caractères, de communiquer ou bien encore de créer des nombres aléatoires (ceci est intéressant en modélisme ferroviaire pour allumer des appartements à la tombée de la nuit dans un ordre imprévisible). Il y a donc parmi les fonctions que propose Arduino, des fonctions qu'il faut absolument connaître car elles servent dans la plupart des programmes pour modélisme ferroviaire.

Mise en pratique

Nous allons mettre en pratique ce que nous avons vu jusque-là. La **figure 4** montre le montage à réaliser. On surveille deux capteurs (deux boutons-poussoirs) qui vont servir à allumer et à éteindre une LED. Le poussoir vert allume la LED et le poussoir bleu l'éteint. La résistance de limitation du courant dans la LED fait 330 ohms (470 ohms pour l'Uno R4) et les deux résistances de pull-up font 10 kilo-ohms. Les broches utilisées sont définies avec const byte (elles ne changent pas au cours du programme) et sont initialisées en entrée (capteurs) ou en sortie (LED) dans le setup. Le programme regarde l'état des capteurs avec digitalRead () : un test if permet de voir si digitalRead () renvoie la valeur LOW ; dans ce cas, c'est qu'un poussoir est appuyé. Si c'est le vert, on allume la LED avec digitalWrite () et si c'est le bleu on l'éteint. En encadré, vous trouverez le détail du programme assurant le fonctionnement de ce montage.

La mémoire d'un microcontrôleur est petite en taille, et il faut dès le début prendre les bons réflexes pour l'économiser : choisir le bon type de variable pour stocker l'information et déclarer les variables qui restent constantes. Il faut aussi adopter des noms de variables qui ne soient pas ambigus et que vous comprendrez lorsque vous reprendrez votre programme après plusieurs mois d'interruption. Dans le prochain article, nous décrirons les capteurs et les actionneurs que l'on peut trouver sur un réseau miniature. ■