Partie 2 : Les entrées

Table des matières

Le bouton poussoir

Vous savez maintenant brancher des LED et programmer leur allumage. Vous avez utilisé des variables de plusieurs types, des tableaux, le moniteur. Vous allez vous apercevoir que la suite finalement n'apporte pas de réelle difficulté.

Maintenant que vous avez compris comment allumer une LED en mettant un pin en position haute, ce serait intéressant d'ajouter de l'interactivité entre l'Arduino et son environnement. Et c'est en utilisant un bouton poussoir que vous allez découvrir dans ce chapitre comment donner un premier sens à l'Arduino : le sens du toucher.

L'utilisation d'un bouton poussoir, et plus généralement d'un contacteur, va vous amener à gérer non plus les sorties, mais les entrées de la carte Arduino. Nous allons donc voir comment elles fonctionnent, comment on les programme, comment on en récupère les informations (il s'agit pour le moment de récupérer les informations des entrées numériques).

Vous allez voir comment connecter un bouton poussoir à la carte Arduino et comprendre l'intérêt des résistances pull-up et pull-down ; puis vous mettrez toutes ces connaissances à l'épreuve avec deux projets de pratique.

La routine quoi...

Les entrées numériques

Nous avons utilisé jusqu'à présent les sorties numériques : soit l'Arduino est en position haute (HIGH) et il envoie du +5V, soit il est en position basse, et il est à 0V, donc au ground.

Et bien les entrées numériques fonctionnent sur le même principe : soit elle reçoivent du +5V ou du 0V.

Pour les sorties nous utilisions la commande `digitalWrite(pin, état)`, qui est donc une commande d'écriture : write=écrire (Lukas, write is not white, right?)
Pour les entrées, nous utiliserons la commande `digitalRead(pin)`, qui vous l'aurez peut-être deviné est une commande de lecture : read=lire.
Mais avant toute chose il faut comprendre une notion importante de l'Arduino : un pin est soit en entrée, soit en sortie, mais pas les deux. Il est donc important de bien définir si le pin va se comporter en entrée ou en sortie ! C'est ce que nous pouvons faire grâce à la commande que nous avons utilisée :

```
pinMode(pin, mode);
```

Où "pin" correspond au numéro de pin concerné, et mode correspond à la façon dont l'Arduino va gérer ce pin, c'est-à-dire :

- mode OUTPUT :

- ```
  pinMode(pin,OUTPUT);
  ```
  pour indiquer à la carte que le pin doit être en **mode écriture**, c'est-à-dire qu'il peut envoyer ou non du courant. C'est donc une sortie.
- mode INPUT :
  ```

- `pinMode(pin,INPUT);`
  pour indiquer que le pin est en **mode lecture**. Il ne va donc pas piloter du courant, mais être à l'écoute du courant qui va lui arriver.

Comment l'Arduino va nous dire ce qu'il "entend" en terme de courant ?

Et bien grâce à la commande digitalRead(pin), voici un morceau de code :

```
void setup()

{

    Serial.begin(9600); //Initialisation de la communication avec le moniteur série

    pinMode(10,INPUT); //On indique à l'Arduino le mode du pin (entrée)

}

void loop()

{

    boolean a=digitalRead(10);// et on l'affecte à la variable "a"

    Serial.println(a); // on l'affiche sur le moniteur

}
```

Ce code permet de lire la valeur reçue par le pin 10.

Saisissez-le et testez-le en ne connectant rien sur votre Arduino.

Dans le meilleur des cas, vous obtiendrez une série de 0. Dans le pire, vous obtiendrez des séries de 1 et de 0 qui changent sans raison valable.

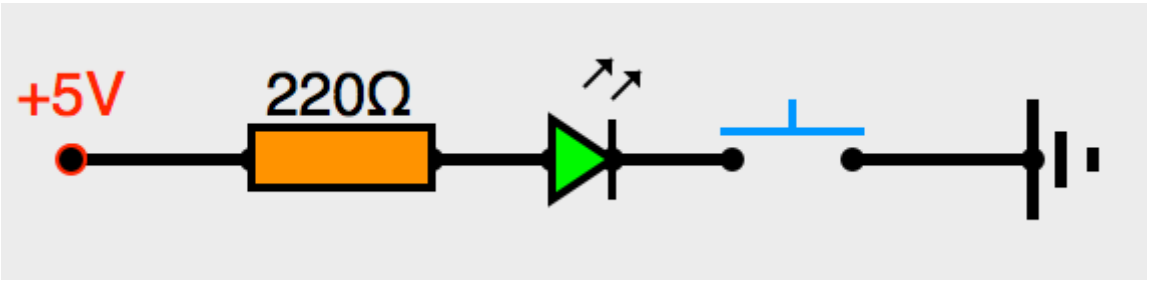
C'est ce qu'on appelle un comportement erratique. C'est-à-dire que la valeur obtenue n'est pas fiable. Et ne vous y trompez pas, même si pour ce test, vous obtenez des 0, vous verrez que si vous ne cherchez pas à pallier ce comportement erratique, lors de vos projets, vous risquez d'avoir des comportements de la carte inattendus. Ce sera peut-être plus visible dans ce qui vient.

Le bouton poussoir

Un bouton poussoir peut se trouver sous plusieurs formes dans le commerce. Ceux qui nous intéressent pour le moment, et que j'espère vous avez en votre possession, est un bouton avec 4 pattes, une forme carrée, et un rond au centre qui est le bouton lui-même.

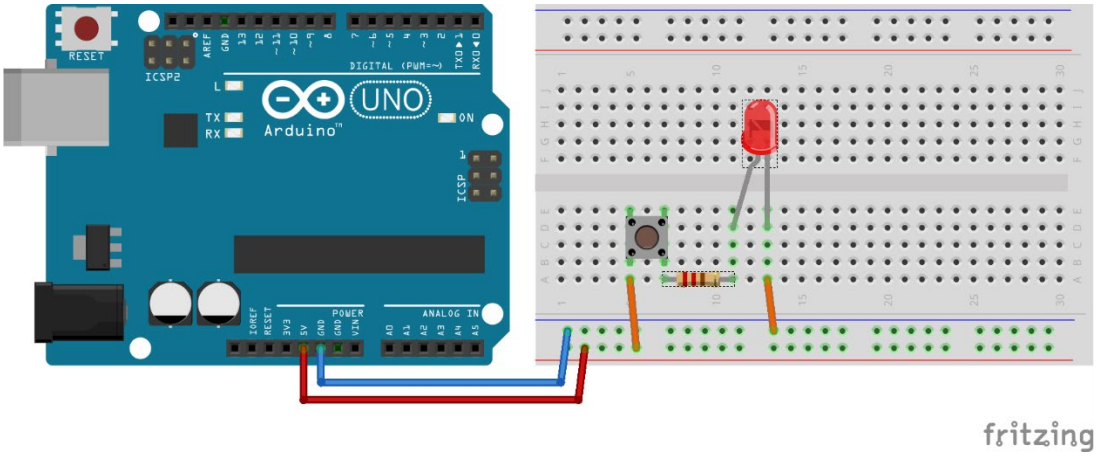
Le principe de ce bouton est que lorsque l'on appuie, le courant passe, et lorsque l'on relâche et bien... le courant ne passe plus !

Contrairement à un interrupteur, il ne garde pas la position (il faut garder le doigt dessus pour qu'il fasse contact). Voici un schéma de circuit :



Le bouton poussoir sert de pont. S'il est levé, le courant ne passe pas, on dit que le circuit est ouvert. S'il est baissé et donc relie les deux contacts, le courant passe, on dit que le circuit est fermé.

L'analogie avec la rivière voit ses limites avec l'interrupteur. En effet, il ne faut pas imaginer que le courant s'accumule en bout de ligne et attend que le contact se fasse pour passer. En fait le courant n'existe que si le circuit est fermé.
Vous pouvez tester le montage suivant avec votre Arduino. Il ne nécessite pas de programme :



Comme à chaque fois, observez le passage du courant. Le poussoir coupe ou non le circuit.

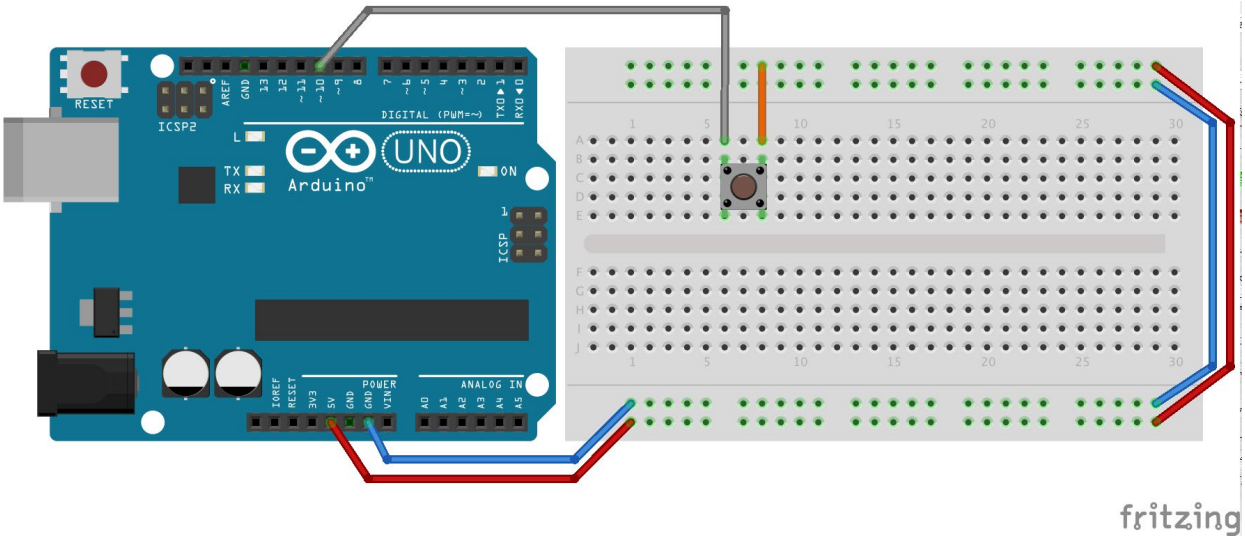
Attention, même s'il a 4 pattes, le bouton poussoir est un dipôle. En fait les pattes sont reliées deux par deux. Le montage au dessus permet d'ailleurs de bien repérer les pattes reliées ou non entre elles avant d'aller plus loin. Le montage fonctionne si la LED s'allume lorsqu'on appuie sur le poussoir. Si elle s'allume en permanence, il faut tourner le bouton d'un quart de tour.
Pour ma part, je ne suis jamais sûr des pattes qui sont connectées entre elles. Du coup, je teste à chaque fois mes poussoirs (car ils peuvent être différents) avec ce montage.
Bon ce montage est en fait l'équivalent de ce que vous avez chez vous pour allumer la lumière : le courant passe si le contact est fait. Il n'y a aucune programmation.

Comment utiliser un bouton par programmation ?

Tout simplement en le reliant à un pin qui est en mode lecture. Si le montage est correctement réalisé, en appuyant sur le bouton, l'Arduino va recevoir l'information et pourra agir en conséquence.

Pour comprendre, l'Arduino va pouvoir lire une valeur de +5V ou de 0V. Donc en théorie, si on envoie le +5V sur un poussoir, quand il est baissé, il laisse passer le courant et l'Arduino reçoit +5V, il indique donc HIGH (ou 1). Si le poussoir est ouvert, l'Arduino devrait ne rien recevoir, donc être à 0V et indiquer LOW (ou 0).

Voici le montage correspondant en connexion sur le pin 10 :



Nous pouvons maintenant écrire le programme qui affiche le résultat sur le moniteur. Essayez de le faire avant de regarder le code qui suit.

```
int pinBouton;

void setup()

{

    Serial.begin(9600);

    pinBouton=10;
```

```
    pinMode(pinBouton,INPUT);

}

void loop()

{

    boolean etatBouton=digitalRead(pinBouton);

    Serial.println(etatBouton);

}
```

Testez le programme... il y a un bug !

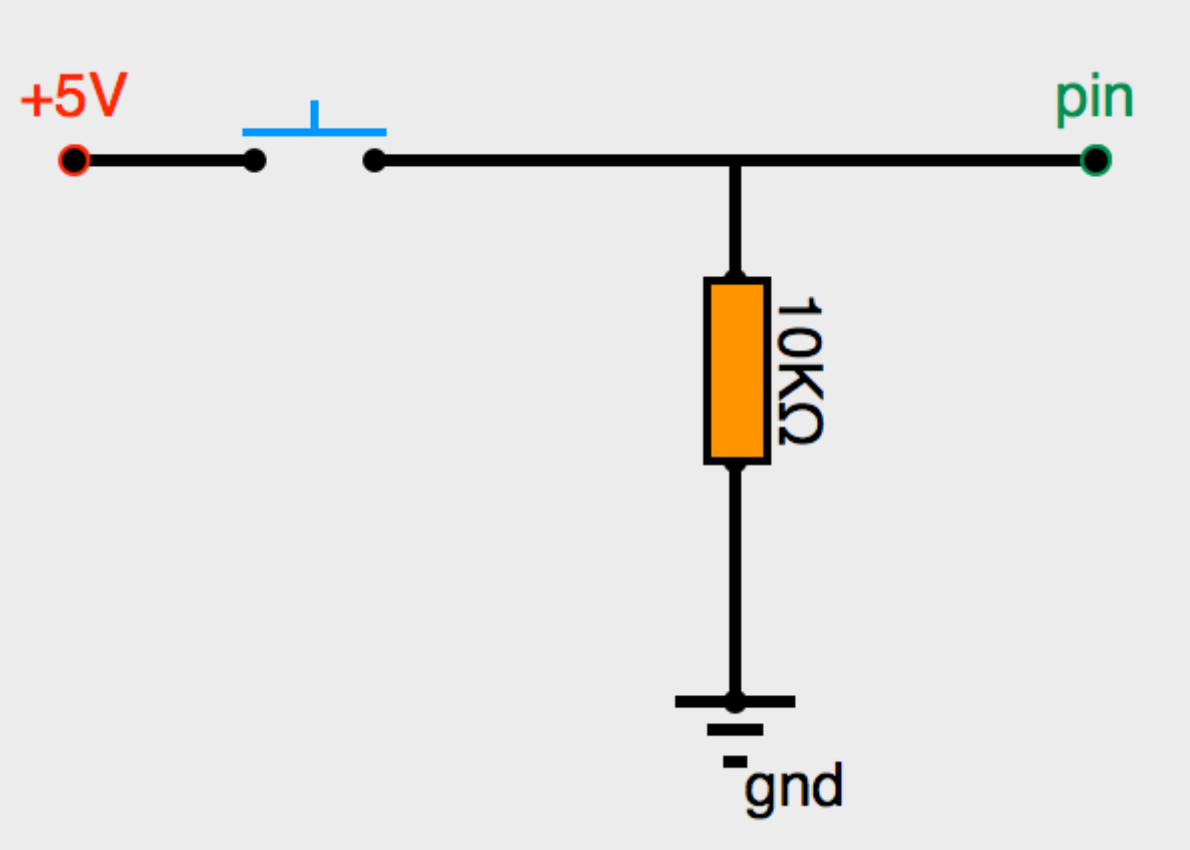
En effet, on voit bien s'afficher des 0, puis lorsqu'on appuie sur le bouton, on voit des 1 et en relâchant, et bien ça reste à 1 !

Il est possible que le comportement de votre Arduino soit légèrement différent, quoi qu'il arrive, la suite vous concerne tout de même.
Ce qui ce passe c'est justement le côté erratique de notre montage. Si on observe bien, le pin 10, quand le bouton est levé, n'est finalement connecté à rien. Le résultat lu par l'Arduino est donc peu interprétable. Il existe un moyen de forcer l'Arduino à lire quelque chose, tout simplement avec l'ajout d'une résistance...

Résistance pull-down

Il faut savoir que l'électricité est paresseuse. Elle va toujours choisir le chemin qui lui résiste le moins. Mais si elle n'a pas le choix, elle passe tout de même là où ça résiste.

Nous allons donc ajouter une résistance à notre circuit. Une assez forte pour que le courant ne passe que s'il y est obligé (souvent de l'ordre de 10kΩ).



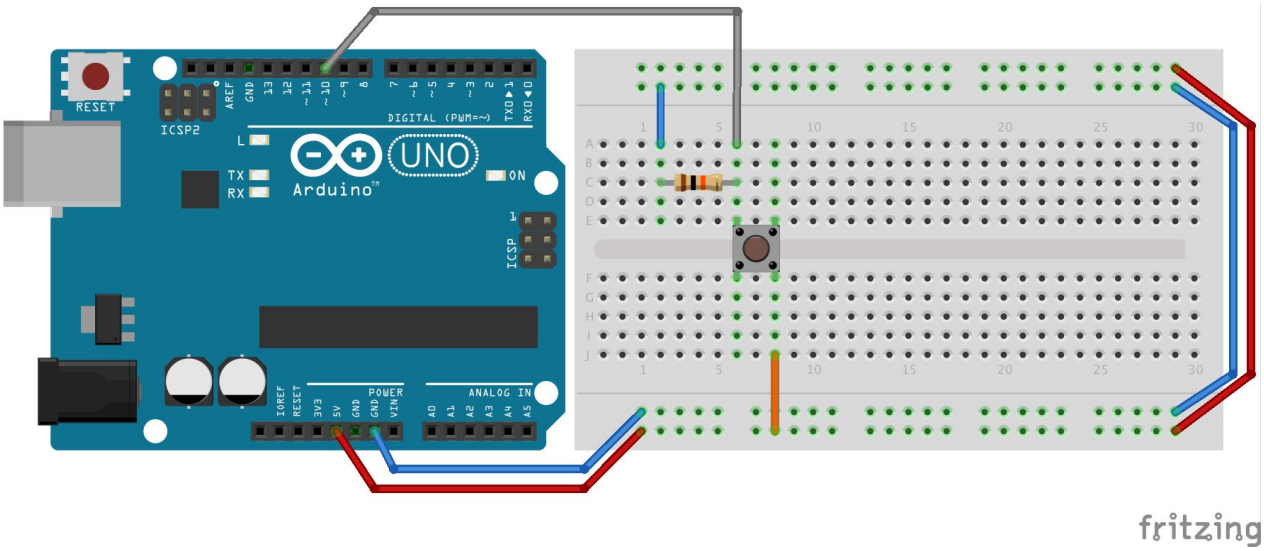
Que se passe-t-il dans ce circuit ?

Si le poussoir est baissé, le courant va du +5V au pin de l'Arduino. Il ne prendra pas le chemin du ground car la résistance lui demande un effort. Le pin de l'Arduino recevra du +5V et indiquera HIGH (ou 1).

Si le poussoir est levé, donc le circuit ouvert, le très faible courant résiduel qui sortira du pin de l'Arduino sera absorbé par le Gnd, le pin sera donc bien en LOW (ou 0).

Ce montage est à connaître, car quel que soit le type de contacteur que vous placerez en lecture sur un pin, il vous faudra prévoir ce comportement erratique.

En ce qui concerne les couleurs des résistances, il existe une multitude d'outils ou d'ouvrages qui traitent du sujet et vous disent quelles couleurs correspondent à quelles résistances. Je ne m'y attarderai donc pas.
Voici le montage sur l'Arduino :



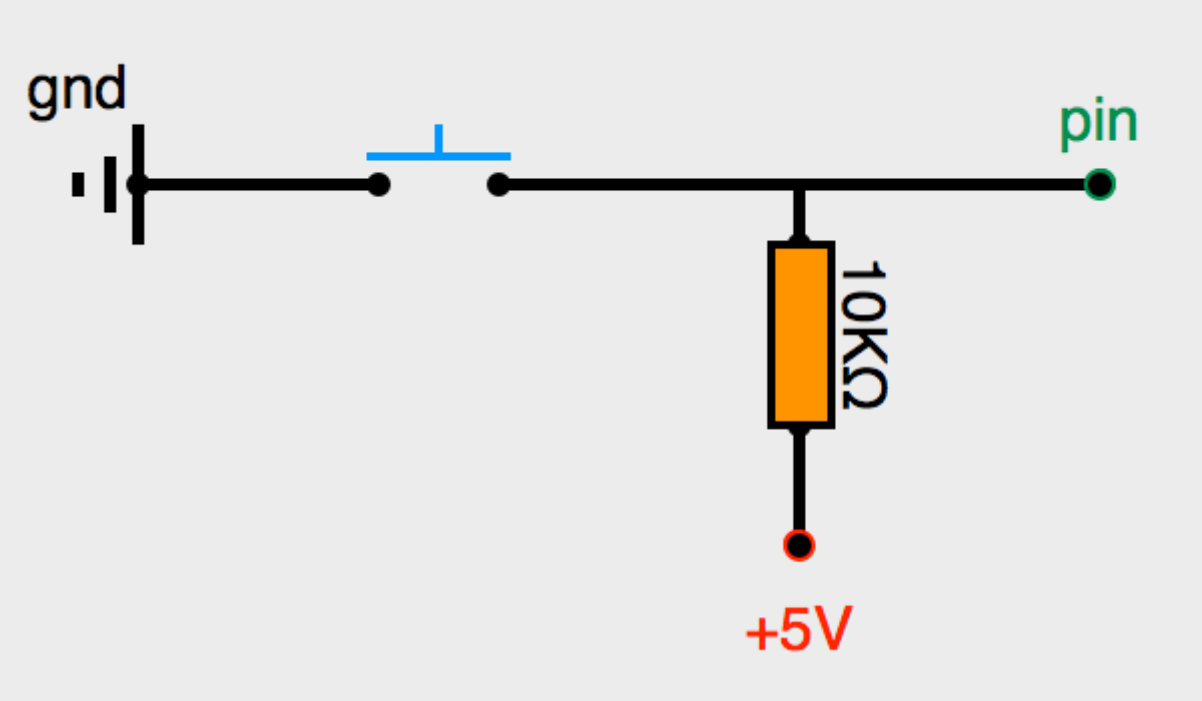
J'en ai profité pour vous montrer une autre façon de connecter le bouton poussoir : à cheval sur le centre de la breadBoard. C'est parfois utile pour un gain de place. Ça ne change rien au montage, je vous laisse observer le parcours du courant.

Si vous testez ce montage avec le programme, vous aurez maintenant un résultat qui correspond à nos attentes, c'est-à-dire que le moniteur affichera 1 lorsqu'on pousse le bouton, et 0 lorsqu'il est levé.

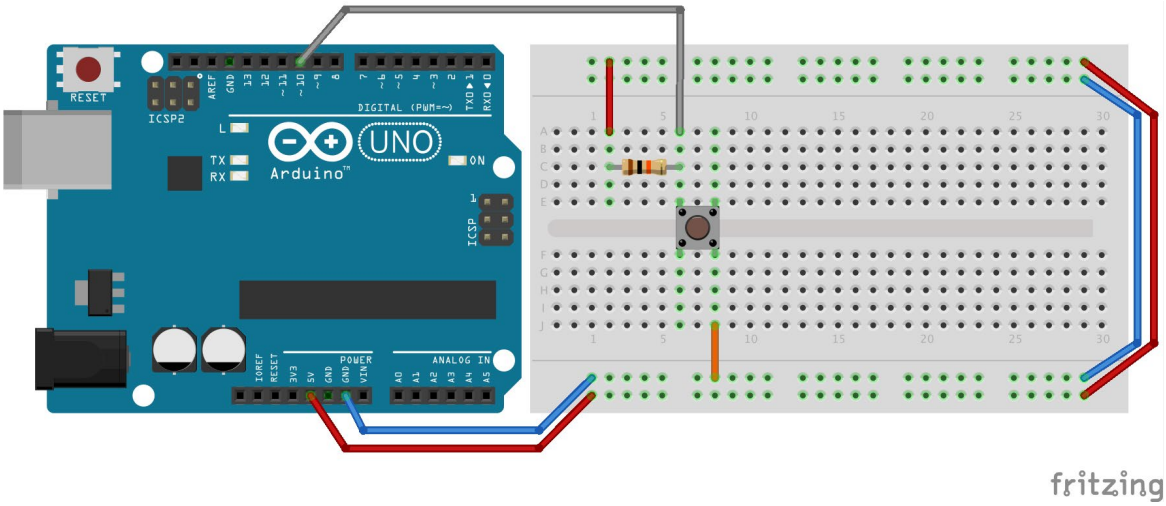
Résistance Pull-Up

Il est possible dans d'autres cas de monter la résistance, non pas vers le ground, mais vers le +5V. Il faut penser à connecter le poussoir au ground (et non plus au +5V) pour que tout fonctionne.

Voici donc le schéma, et le montage :



Et le montage sur l'Arduino :



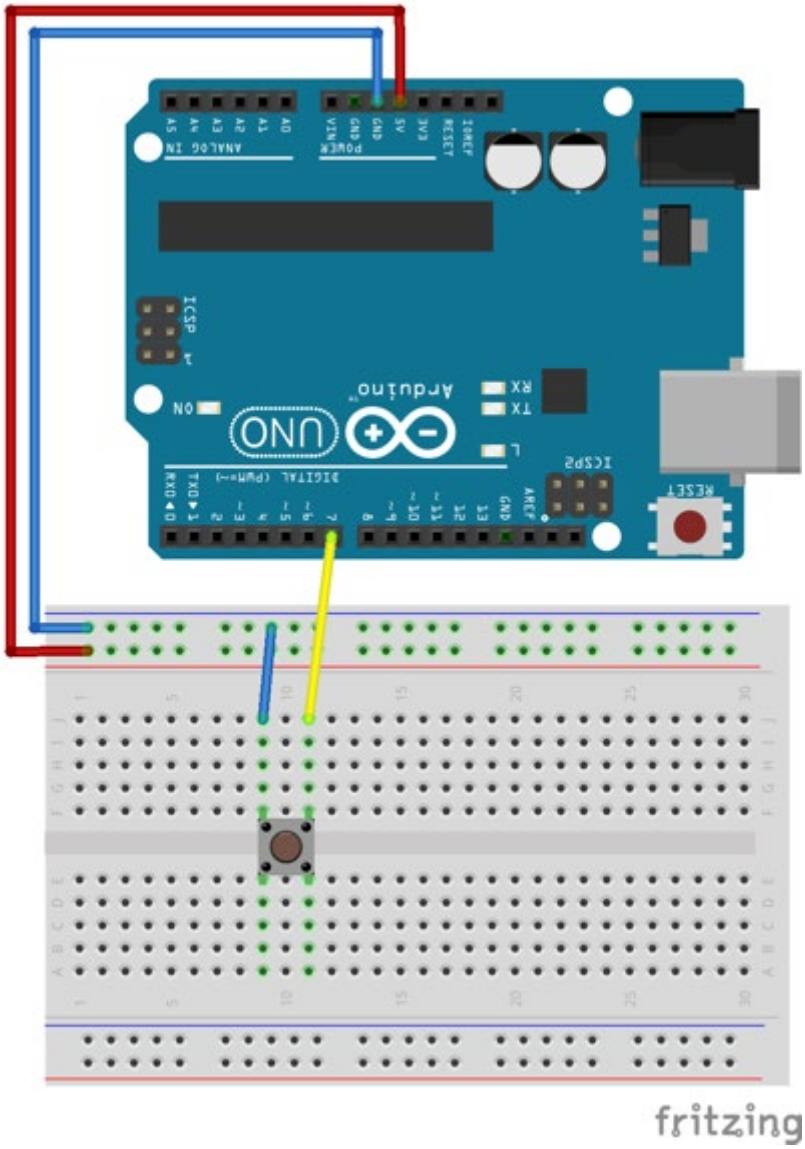
- Quand le poussoir est ouvert, le +5V nourrit le pin de l'Arduino, qui donnera HIGH comme résultat.
- Lorsqu'il est fermé, le +5V et le pin sont absorbés par le ground, le pin donnera LOW comme résultat.

Le fonctionnement en pull-up donne en lecture l'opposé du fonctionnement en pull-Down. C'est important à savoir car dans un cas (pull-down), on obtient HIGH avec le bouton appuyé et dans l'autre cas (pull-up), on obtient LOW avec le bouton appuyé.
Faites le test avec le programme précédent, vous verrez s'afficher 1 quand le bouton est levé et 0 quand il est baissé. Vous verrez que ça changera nos tests plus tard.

Le mode INPUT_PULLUP

Vous allez maintenant comprendre l'intérêt des explications précédentes ! (enfin j'espère...)

La carte Arduino propose par défaut un mode qui permet d'activer une résistance de 20 KΩ qui est dans la carte pour en faire une résistance pull-up. L'avantage est clair : pas besoin de se prendre la tête avec une résistance en plus. Il suffit de connecter correctement le bouton poussoir en mode pull-up.



Bouton poussoir connecté au pin 7 en mode INPUT_PULLUP

Il faut aussi et surtout indiquer à l'Arduino d'activer cette résistance, on utilise la commande suivante :

```
pinMode(pin,INPUT_PULLUP);
```

Ce qui pour le montage de l'image précédente deviendrait :

```
pinMode(7,INPUT_PULLUP);
```

Vous voyez donc que la commande`pinMode` sert finalement à préparer un pin dans trois modes différents :

- Mode **OUTPUT** : l'Arduino fournira par programmation du +5V ou du 0V.
- Mode **INPUT** : l'Arduino lira l'état sur le pin concerné : HIGH (ou 1) pour +5 V reçus et LOW (ou 0) pour 0 V. Il faut prévoir le montage qui correspond.

- Mode **INPUT_PULLUP** : l'Arduino lira les informations reçues, mais le pin sera connecté en interne (dans la carte elle-même) avec une résistance de 20KΩ en mode pull-up.

Attention, pour le pin 13, il est déconseillé d'utiliser le mode INPUT_PULLUP. Si vous devez vous servir du pin 13 comme pin de lecture, préférez un montage avec une résistance externe en pull-up ou pull-down. L'explication se trouve dans le fait que le pin 13 est aussi lié à une LED et une résistance. Il ne fournira donc pas du +5V, mais du +1.7V à cause de la LED et de la résistance en série qui baissent la tension. De ce fait la lecture sera toujours à LOW.
Et bien nous pouvons attaquer nos nos projets pratiques maintenant...

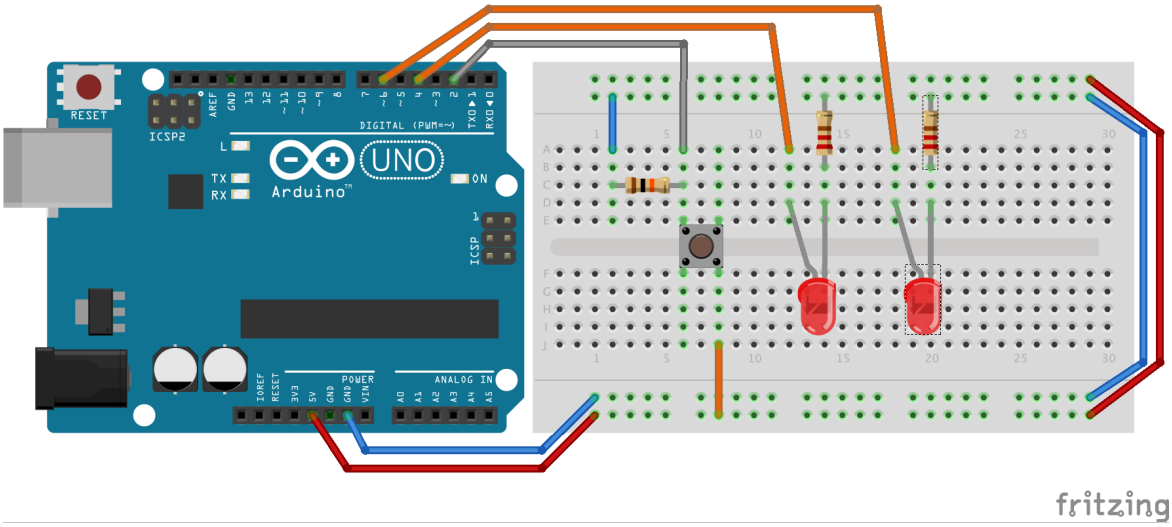
Programme "Jour/Nuit"

Les attentes de fonctionnement de ce programme sont simples...

- Le montage contient un bouton poussoir connecté au pin 2 avec une résistance montée en pull-down ;
- Une LED (LED1) connectée au pin 4 (pensez à la résistance) ;
- Une LED (LED2) connectée au pin 6 (pensez à acheter du pain...) ;
- Lorsque le bouton est levé, la LED1 est allumée, la LED2 est éteinte ;
- Lorsque le bouton est appuyé, la LED1 est éteinte, la LED2 est allumée.

Indice : il faudra tester l'état du bouton poussoir (pin en lecture) et faire agir le programme en fonction.
Vous avez tout ce qu'il faut pour réussir à programmer seul(e) cet exercice. Je vous laisse donc le plaisir de le faire...

Voici la solution pour le montage :



Montage des deux LED, du bouton poussoir et de la résistance en pull-down

et voici le code :

```
/*

le bouton poussoir est connecté au pin 2 avec une résistance pull-down de 10KΩ

les LED sont connectées au pins 4 et 6 avec des résistances de 220Ω
```

```
*/



//déclaration des variables

int pinBouton;

int pinLed1, pinLed2;

void setup()

{

  //initialisation des variables

  pinBouton = 2;

  pinLed1 = 4;

  pinLed2 = 6;

  //définition des modes

  pinMode(pinBouton, INPUT); //mode lecture pour le bouton

  pinMode(pinLed1, OUTPUT); //mode écriture pour led1

  pinMode(pinLed2, OUTPUT); //mode écriture pour led2



}

void loop()

{

  //lecture de l'état du bouton et stockage dans etatBouton

  boolean etatBouton = digitalRead(pinBouton);

  //test des conditions

  if (etatBouton==HIGH)//test si bouton appuyé

  {
```

```
    digitalWrite(pinLed1,LOW); //led1 éteinte

    digitalWrite(pinLed2,HIGH); //led2 allumée

  }

  if (etatBouton==LOW)//test si bouton levé

  {

    digitalWrite(pinLed1,HIGH); //Led1 allumée

    digitalWrite(pinLed2,LOW); //led2 éteinte

  }

  delay(100); //petite attente

}
```

Quelques informations complémentaires sur ce code...

- `int pinLed1,pinLed2;` : on peut très bien déclarer plusieurs variables à la suite séparées d'une virgule ;
- `boolean etatBouton = digitalRead(pinBouton);` : le type boolean est tout à fait approprié pour la variable état bouton qui ne prend que les valeurs 1 ou 0 (HIGH ou LOW) ;
- `if (etatBouton==HIGH)` et`if (etatBouton==LOW)` : lorsque l'on teste l'état du bouton, il est tout à fait possible de remplacer HIGH par 1 et LOW par 0. Il est même possible comme vu dans les chapitres précédents d'écrire :

```
if (etatBouton)

{

   //code à effectuer

}
```

- `delay(100);` : il est conseillé de toujours laisser un peu de temps à l'Arduino avant de refaire une boucle, surtout lorsque des données sont lues.

Il est enfin possible, pour éviter la succession des deux conditions "if" d'utiliser une méthode de tests combinés, voici un code qui le montre :

```
if (test)

{

    //code à exécuter

}
```

```
else if (autreTest)

{

    //autre code

}

else

{

    //code exécuté si aucun test n'est vérifié

}
```

"else" signifie "sinon". Nous aurons l'occasion de revenir sur ce point plus tard. Mais je vous livre le code du programme "Jour/nuit" modifié avec les points que nous venons de voir. Observez les lignes 7, 25 et 30 :

```
/*

le bouton poussoir est connecté au pin 2 avec un résistance pull-down

les leds sont connectées au pins 4 et 6 avec des resistances de 220Ω

*/



//déclaration des variables

int pinBouton, pinLed1, pinLed2;

void setup()

{

  //initialisation des variables

  pinBouton = 2;

  pinLed1 = 4;

  pinLed2 = 6;

  //définition des modes

  pinMode(pinBouton, INPUT);
```

```
  pinMode(pinLed1, OUTPUT);

  pinMode(pinLed2, OUTPUT);



}

void loop()

{

  //lecture de l'état du bouton et stockage dans etatBouton

  boolean etatBouton = digitalRead(pinBouton);

  //test des conditions

  if (etatBouton)//si bouton appuyé (donc 1)

  {

    digitalWrite(pinLed1,LOW);

    digitalWrite(pinLed2,HIGH);

  }

  else //sinon

  {

    digitalWrite(pinLed1,HIGH);

    digitalWrite(pinLed2,LOW);

  }

  delay(100);

}
```

Bien, ce programme commence à bien plus interagir avec notre Arduino. Nous verrons plus tard, qu'à la place d'un poussoir, il peut y avoir un système de détection par lumière, ou un contacteur type moustache de chat pour un robot...

Allez, un dernier programme pour bien asseoir nos connaissances !

Programme "Interrupteur"

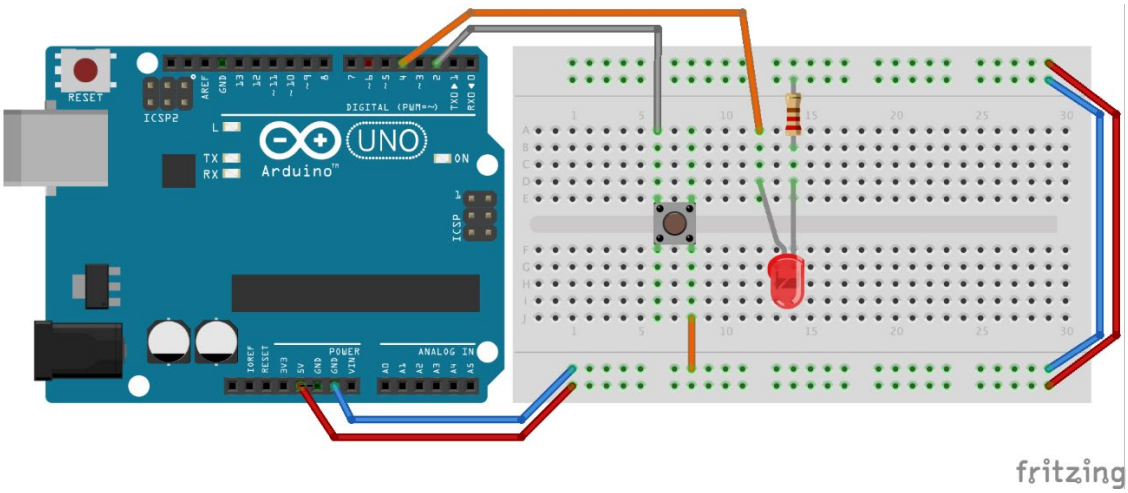
L'objectif de ce programme est à nouveau facile à décrire :

- Le bouton poussoir est connecté au pin 2 en mode INPUT_PULLUP ;
- Une LED est connectée au pin 4 ;
- Quand on appuie une fois sur le bouton, la LED s'allume (et reste allumée) ;
- Lorsqu'on appuie à nouveau, la LED s'éteint (et reste éteinte).

Indice : il faut créer une variable qui change à chaque appui sur le bouton...
Allez ! Go ! Go ! Go !

Solution

Voici le montage :



Montage d'un bouton en pull-up sur pin 2 et d'une diode sur pin 4

```
/*

le bouton poussoir est connecté au pin 2 pour un mode INPUT_PULLUP

la Led est connectée au pins 4 avec une résistance de 220Ω

*/



//déclaration des variables

int pinBouton, pinLed;

boolean etatAllumage;

void setup()

{
```

```
  //initialisation des variables
  Serial.begin(9600);
  pinBouton = 2;
  pinLed = 4;
  etatAllumage=0;
  //définition des modes
  pinMode(pinBouton, INPUT_PULLUP);
  pinMode(pinLed, OUTPUT);
}
void loop()
{
  Serial.print(etatAllumage);


  if (etatAllumage) //on teste si etatAllumage est à 1
  {
    digitalWrite(pinLed, HIGH);//on allume la LED
  }
  else //sinon
  {
    digitalWrite(pinLed, LOW); //on éteint la LED
  }
  //lecture de l'état du bouton et stockage dans etatBouton
  boolean etatPinBouton = digitalRead(pinBouton);
  Serial.println(etatPinBouton);
```

```
  //test des conditions

  if (!etatPinBouton)//si bouton appuyé (donc le pin indique 0 car il est en mode
INPUT_PULLUP)

  {

    if (etatAllumage) //si etatAllumage à 1

    {

      etatAllumage=0; //on le passe à 0

    }

    else //sinon

    {

      etatAllumage=1; //on le passe à 1

    }

  }

  delay(200);

}
```

On remarque que le programme n'est pas si simple. Nous verrons plus tard comment le rendre plus concis.

J'attire votre attention sur la ligne 36 : on utilise le point d'exclamation "!" devant la variable. Ça veut dire "non" en informatique donc !etatBouton, veut dire "non etatBouton=1" traduit pour l'humain : si etatBouton n'est pas égal à 1. Donc écrire `if(!etatPinBouton)` revient à écrire `if(etatPinBouton!=1)` , c'est juste plus rapide !
Rappelez-vous qu'en mode pull-up, le bouton appuyé donne un résultat bas (LOW ou 0).
Le test de ce programme n'est pas très agréable. En effet, l'appui prolongé sur le bouton, ou des appuis plus ou moins longs modifient l'allumage de la diode.

C'est toujours une question de rapidité de la machine par rapport à l'humain : si l'on maintient appuyé, la boucle se répète et passe la variable `etatAllumage` de 0 à 1 sans arrêt.

Utilisez les fonctions et les nombres aléatoires

Vous êtes maintenant en mesure de réaliser un projet plus complexe. Nous sommes tous un peu joueurs dans l'âme, et depuis tout petit, nous connaissons le dé, nous avons même appris à

compter en partie grâce à lui ! Je vous propose dans ce chapitre de lui rendre hommage et de l'électroniser : nous allons donc réaliser un dé avec 5 LED.

Vous allez pouvoir, grâce à ce projet, convoquer l'ensemble de vos connaissances, ce qui va être un très bon exercice.

Vous y apprendrez également à utiliser :

- Les nombres aléatoires, qui permettent justement de réaliser des projets comme un dé, mais qui peuvent servir dans bien d'autres cas.
- Les fonctions, qui deviennent vite incontournables quand le code se complexifie.

Les schémas et montages vont commencer à devenir plus complexes et vous aurez sûrement à passer un peu de temps à vérifier vos montages si tout ne se passe pas comme prévu...

À l'issue de ce chapitre, vous aurez créé un dé fonctionnant de la façon suivante :

Lors que vous appuyez sur un bouton poussoir, les LED s'allument s'allument à des intervalles de plus en plus lents en affichant un nombre aléatoire (entre 0 et 5) pour s'arrêter sur un nombre final entre 0 et 5. Les LEDs seront disposées comme le 5 d'un dé.

Avant d'arriver au programme final de ce dé, nous passerons par des programmes intermédiaires qui nous permettrons de tester étape par étape notre montage.

Alea jacta est

("Les dés sont jetés", Jules César)

Nous avons vu précédemment qu'un ordinateur est très bon en calcul mais qu'il ne peut pas réfléchir par lui-même. Et bien il a un autre défaut, il est complètement prévisible. Les explications qui suivent risquent de vous dégoûter de jouer contre un ordinateur à un jeu de hasard (oui, oui, même dans les casinos...) car finalement, de hasard il n'y a pas !

Voyons tout d'abord comment créer un petit programme qui affiche sur la console un nombre aléatoire (non Lukas, il ne s'agit pas d'aller à Thoires...). Pour cela, on utilise le mot clé `random` du langage Arduino :

```
random(max); // retourne un entier entre 0 et et la valeur max précisée (non comprise)
```

On peut écrire aussi :

```
random(min,max); //retourne un entier entre les valeurs min et max précisées (max non compris)
```

Quand je dis `max` (non compris) c'est que l'Arduino ne renverra jamais le nombre `max`. Par exemple, si vous écrivez :

```
random(5,20);
```

L'Arduino vous renverra un nombre parmi : 5,6,7,8,9,10,11,12,13,14,15,16,17,18 et 19.

Voici donc un petit programme à tester avec votre Arduino :

```
int nbAlea;

void setup() {

  Serial.begin(9600);

}



void loop() {

  nbAlea=random(100); //construit le nombre aléatoire

  Serial.println(nbAlea);//affichage du nombre

  delay(500);//attente

}
```

Téléversez-le et ouvrez la console. Si tout se passe bien vous voyez des nombres qui s'affichent les uns en dessous des autres, de façon aléatoire, entre 0 et 99 (puisque 100 n'est pas compris dedans).

Très bien, appuyez maintenant sur le bouton 'reset' situé sur votre Arduino. Observez la série de nombres...

Appuyez encore sur le reset, observez, appuyez, observez...

Ben c'est tout le temps la même série de nombres, c'est pas aléatoire ça !!!

En effet, comme je vous l'ai indiqué, un ordinateur c'est prévisible. Créer un nombre aléatoire avec c'est impossible. C'est la raison pour laquelle on parle de nombres **pseudo-aléatoires**. Heureusement les ingénieurs ingénieux ont pensé détourner le problème pour tenter de produire une série de nombres différents et presque imprévisibles. Il suffit de glisser dans votre programme (souvent dans le `setup`) le code suivant :

```
randomSeed(unNombre);
```

La variable unNombre étant un int pris au hasard.

Mais comment prendre un `int` au hasard s'il n'y a pas de hasard ?

Bien vu ! Il faut trouver un moyen pour que cette graine (seed en anglais) soit un nombre qu'on ne peut pas deviner ou connaître. Nous n'avons pas encore vu les entrées analogiques, mais sur l'Arduino, c'est la seule méthode pour être à peu près tranquille sur le choix de la variable `unNombre`. On va donc assigner à la variable `unNombre` une entrée analogique en appelant la méthode `analogRead` comme ceci :

```
randomSeed(analogRead(0));
```

Et pour que ça marche, il faut que le pin A0 de votre Arduino ne soit pas connecté.

Voici donc le programme de test de nombres aléatoires modifié :

```
int nbAlea;

void setup() {

  Serial.begin(9600);

  randomSeed(analogRead(0));//initialise la séquence aléatoire

}



void loop() {

  nbAlea=random(100); //construit le nombre aléatoire

  Serial.println(nbAlea);//affichage du nombre

  delay(500);//attente

}
```

Si vous appuyez sur 'reset', vous obtenez des séquences différentes. C'est déjà bien mieux !

Notre projet concerne un dé à 5 LEDs, il nous faudra donc un nombre aléatoire entre 0 et 5. Je vous laisse modifier le code pour qu'il corresponde à nos attentes. Testez-le ensuite vérifier qu'on obtient bien tous ces nombres.

Bien, nous savons maintenant produire une série de nombres aléatoires entre 0 et 5, passons donc au montage ! (Lukas, vous pouvez cesser d'appuyer sur le reset...)

Préparer le montage

Pour que notre dé ressemble à un dé, et bien il faut lui en donner l'apparence. Voici les constellations (c'est le terme technique) pour un dé commun à six faces :

Notre projet concerne un dé à 5 LEDs donc nous allons essayer de positionner les LEDs pour qu'elles ressemblent à la constellation du 5 sur un dé normal. En utilisant cette forme, on peut afficher chaque nombre entre 1 et 5.

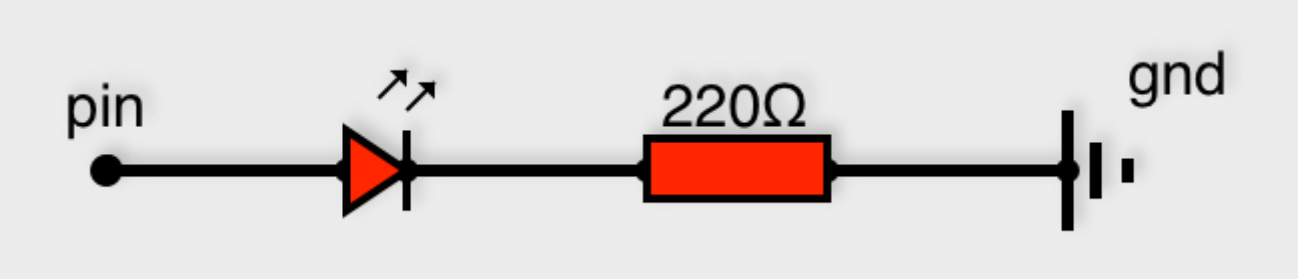
Alors ça veut dire qu'on n'aura jamais un 6 ?

En effet, le 6 n'existe pas dans notre projet, car il nous aurait fallu ajouter deux LED pour obtenir la bonne constellation, ça nous aurait fait trop de travail juste pour un nombre. Mais

rassurez-vous, nous aurons encore 6 positions : 0,1,2,3,4, et 5. Finalement, le 0 remplace le 6...

Alors essayons de placer les diodes dans la position de la constellation du 5. Les diodes seront pilotées par l'Arduino avec les pins 2,3,4,5,6.

Pour rappel, chaque diode doit être connectée au pin de l'Arduino avec sa patte longue, et au ground avec sa patte plus courte, mais elles doivent aussi être en série avec une résistance (prenons 220Ω). Voici le schéma pour une diode :



Je vous propose de vous lancer un peu tout seul comme d'habitude. Rappelez-vous que l'objectif pour vous est de devenir autonome...

Voici le schéma avec fritzing :

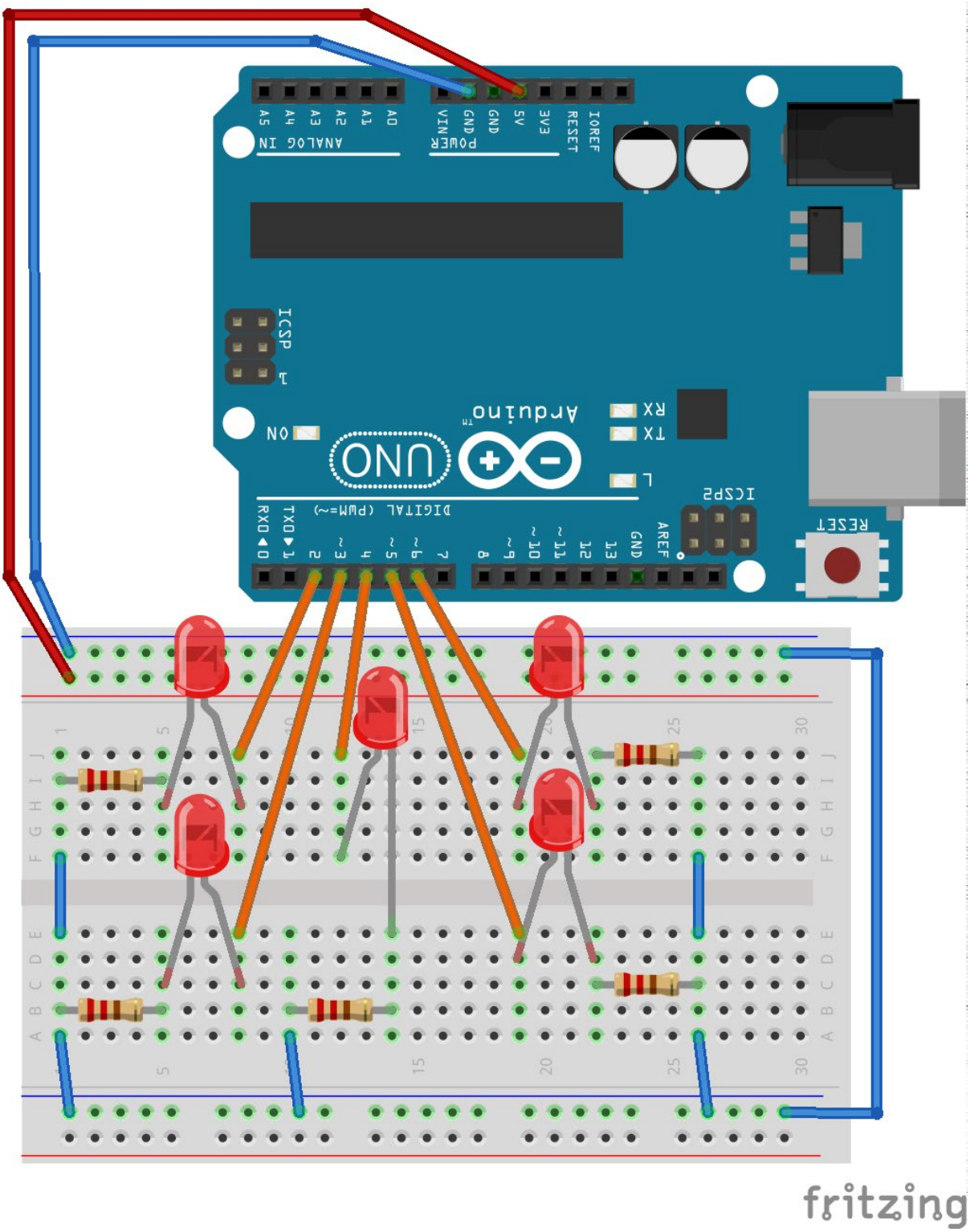


Schéma de montage des LED en forme de dé
Prenez votre temps pour observer et comprendre. J'ai choisi des fils oranges pour les connexions vers l'Arduino, rouge pour le +5V (non utilisé ici) et bleu pour la connexion vers le ground. Chaque patte + des diodes est bien reliée à un pin de l'Arduino. Chaque patte - est reliée vers le ground en passant par une résistance de 220Ω.

Vous pouvez alléger votre montage en connectant directement votre résistance sur le ground (rail du haut ou du bas en bleu). Je n'ai pas pu vous le montrer avec le logiciel Fritzing.

Ça fait pas mal de fils, mais ça on ne peut pas l'éviter... Voyons maintenant comment programmer l'allumage des LED.

Un programme pour tester nos LEDs

Rappel : nous voulons que les LEDs s'allument en respectant les constellations du dé.

Je vous laisse essayer. Il faut que votre programme contienne :

- une variable pour chaque pin de LED ;
- une initialisation de tous les pins de LED en mode OUTPUT (envoi de courant) ;
- un positionnement de chaque pin de LED en HIGH (valeur haute).

Voici une proposition de correction simple et lisible :

```
//définition des variables de pin pour chaque LED

int ledHautGauche=2;

int ledBasGauche=3;

int ledCentre=4;

int ledHautDroite=6;

int ledBasDroite=5;



void setup() {



  //positionnement des pin en OUTPUT

  pinMode(ledHautGauche,OUTPUT);

  pinMode(ledBasGauche,OUTPUT);

  pinMode(ledCentre,OUTPUT);

  pinMode(ledHautDroite,OUTPUT);

  pinMode(ledBasDroite,OUTPUT);
```

```
  //allumage de tous les pins

  digitalWrite(ledHautGauche,HIGH);

  digitalWrite(ledBasGauche,HIGH);

  digitalWrite(ledCentre,HIGH);

  digitalWrite(ledHautDroite,HIGH);

  digitalWrite(ledBasDroite,HIGH);

}



void loop() {

  //inutile pour ce test

}
```

Vous remarquerez que j'ai nommé les LED en fonction de leur position. Rien ne vous oblige à faire pareil, c'est juste beaucoup plus facile ensuite...

Voici maintenant un code plus compact qui fait la même chose. Je vous laisse le soin de le comprendre :

```
void setup() {

  for (int l=2;l<7;l++){// boucle de 2 à 6

    pinMode(l,OUTPUT);// mode OUTPUT

    digitalWrite(l,HIGH);// allumage

  }

}

void loop() {

  //inutile pour ce test

}
```

Dans les deux cas, vous devez avoir les 5 LED allumées. Si ce n'est pas le cas, vérifiez votre programme, vos connexions, le sens des LED, les résistances...

Nous pouvons facilement maintenant construire un programme qui teste le 0. En effet, il suffit de tout mettre en position LOW.

Voici un tableau qui montre l'état des Pins (HIGH ou LOW) en fonction du nombre :

Nombre	ledHautGauche	ledBasGauche	ledCentre	ledHautDroite	ledBasDroite
0	LOW	LOW	LOW	LOW	LOW
1	LOW	LOW	HIGH	LOW	LOW
2	HIGH	LOW	LOW	LOW	HIGH
3	LOW	HIGH	HIGH	HIGH	LOW
4	HIGH	HIGH	LOW	HIGH	HIGH
5	HIGH	HIGH	HIGH	HIGH	HIGH

Ce tableau d'état permet de bien repérer ce que nous voulons réaliser.

Maintenant, faisons compter notre Arduino avec ce dé ! Je m'explique : nous allons programmer les LED connectées à l'Arduino pour qu'elles affichent la constellation du dé correspondant à 0, puis à 1, puis à 2... jusqu'à 5, puis à nouveau 0, 1, etc. (on prendra une seconde d'intervalle entre l'affichage de chaque nombre par exemple).

On imagine assez bien une boucle du genre :

```
for (int l=0;l<6;l++){

    if (l==0){

        //affichage des diodes en forme de 0

    }

    if (l==1){

        //affichage des diodes en forme de 1

    }

    if (l==2){

        //affichage des diodes en forme de 2

    }

    if (l==3){

        //affichage des diodes en forme de 3
```

```
    }

    if (l==4){

        //affichage des diodes en forme de 4

    }

    if (l==5){

        //affichage des diodes en forme de 5

    }

}

// à répéter à l'infini
```

Avec pour chaque forme un affichage précis des diodes en fonction du tableau précédent.

Je vous laisse construire ce programme et le tester (la boucle `for` est à placer dans la `loop()`). Je vais en profiter pour prendre l'air...
...

Alors ? Votre essai a été concluant ? Super !

Sinon, pas de panique, vérifiez la correction.

Et là, je suis sûr que vous vous dites : quand même, c'était un peu répétitif mon code, c'est toujours aussi fastidieux d'écrire des programmes ? N'ayez crainte, je m'en vais de ce pas vous initier à un concept qui va vous permettre de rendre votre code plus simple, lisible et agréable : les fonctions !

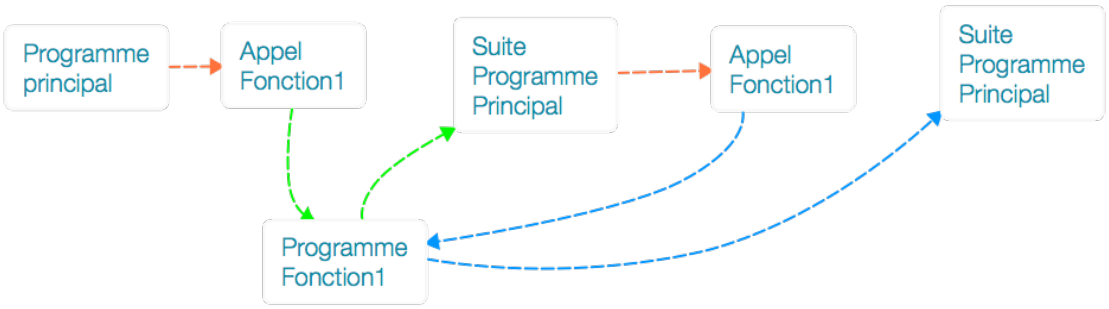
Les fonctions

La boucle que nous venons de voir risque d'être appelée plusieurs fois dans le programme. En effet, je vous rappelle que le but est d'appuyer sur un bouton et que les nombres affichés défilent de plus en plus lentement jusqu'à l'arrêt sur un nombre final. Donc à chaque affichage, la boucle va être utilisée pour gérer l'affichage des diodes.

Si l'on écrit tout dans la`loop` (test du bouton, tirage du nombre, affichage des diodes...), nous allons avoir un programme surchargé, et pas très lisible. C'est là que les fonctions vont nous servir.
Le principe est simple :

Le programme se déroule, puis à un moment on va lui demander d'exécuter une fonction, et lorsqu'il aura fini de réaliser cette fonction, il reviendra dans le programme principal, là où il s'était arrêté.

Voici un schéma qui explique un appel à une seule fonction :



Il y a donc une interruption du programme principal pour passer dans la fonction, puis retour.

Pour programmer une fonction il faut écrire deux choses :

- La **définition (ou déclaration) de la fonction**, c'est-à-dire le code à exécuter par la fonction. Le programme de la fonction en fait.
- L'**appel à la fonction** dans le programme principal.

Déclaration de la fonction

La déclaration est de la même forme que le `setup` ou la `loop` (qui sont en fait des fonctions !) mais on est libre d'indiquer plusieurs choses :

```
type nomDeLaFonction(){

//code du programme de la fonction

}
```

- `type` : c'est le type de variable que la fonction peut renvoyer. Pour le moment on indiquera “void” qui veut dire “vide”, c’est-à-dire que la fonction comprend l’exécution d’un certain nombre d’instructions mais ne renvoie aucune valeur de sortie.
- `nomDeLaFonction` : c'est le nom que l'on choisit pour la fonction. Dans l'exemple d'avant j'ai choisi "fonction1" et "fonction2", mais j'aurais pu mettre "patate" ou "affichageDuNombre".
- `()` : ces deux parenthèse sont obligatoires. Nous allons voir plus loin à quoi elles servent. Si vous les oubliez, l'IDE vous retournera une erreur.
- `{}` : le code du programme de la fonction doit être mis entre accolades. Vous noterez qu'il n'y a pas de point-virgule après l'accolade de fermeture.

Je place en général mes fonctions après la fonction `loop()`, et je vous conseille de faire de même. Mais elles peuvent être positionnées partout ailleurs dans le programme tant qu'elles sont déclarées de façon globale, c'est-à-dire en dehors de la `loop`, du `setup` ou de toute autre fonction.

Appel de la fonction

Pour appeler une fonction ensuite, il suffit de mettre ce code au bon endroit dans votre programme :

```
nomDeLaFonction();
```

Lors du déroulement du programme, lorsque l'Arduino lira cet appel, il ira directement retrouver le contenu de la fonction ! Vous verrez vite que c'est bien pratique.

Voici un programme qui affiche dans la console le déroulement du code :

```
void setup() {

  Serial.begin(9600);

}

//boucle de programme principal

void loop() {

  Serial.println("Je suis dans le programme principal, en début de boucle");

  delay(1000);

  fonction1();

  fonction2();

}



//code de la fonction1

void fonction1(){

  Serial.println("Je suis dans la fonction1");

  delay(1000);

}



//code de la fonction2

void fonction2(){

  Serial.println("Je suis dans la fonction2");

  delay(1000);

}
```

Ce programme montre bien le passage dans chaque fonction.

Passer une information à la fonction

Il faut savoir qu'une fonction peut recevoir une ou plusieurs informations. C'est-à-dire une ou plusieurs variables ou constantes qu'on appelle des **paramètres**. Il faut pour cela modifier la déclaration de la fonction et son appel.

Déclaration :

```
type nomDeLaFonction(type unParametre){

    //code de la fonction qui peut utiliser la variable 'unParametre'

}
```

Appel :

```
nomDeLaFonction(parametre);
```

Il faut que le paramètre envoyé à la fonction soit du même type que ce qui a été déclaré dans la fonction.
Il est possible de passer plusieurs paramètres à une fonction, il suffit simplement de les déclarer à la suite les uns des autres en les séparant d'une virgule :

Déclaration :

```
type nomDeLaFonction(type param1,type param2,type param3...){

    code de la fonction

}
```

Appel :

```
nomDeLaFonction(param,autreParam, encoreAutreParam,...);
```

Si une fonction peut prendre plusieurs paramètres, elle ne peut en revanche retourner qu'une seule valeur (de type défini avant le nom de la fonction).
Voici un exemple de programme utilisant des fonctions à plusieurs paramètres :

```
void setup(){

    Serial.begin(9600);

}

void loop(){
```

```
    //appel de fonction avec variables pré-déclarées

    int a=5;

    int b=3;

    int s=somme(a,b);

    Serial.println(s);

    //appel de fonction en paramètres directs

    Serial.println(produit(5,2));

}

//déclaration des fonctions

int somme(int nb1,int nb2){

    //utilisation de variables dans la fonction

    int res=nb1+nb2;

    return res;

}

int produit(int nb1,int nb2){

    //retour direct d'un résultat sans variable intermédiaire

    return nb1*nb2;

}
```

Je vous laisse observer les différentes façons d'envoyer les paramètres et d'utiliser les variables ou les valeurs de retour.

Utilisez des fonctions pour le dé à 5 LED

Nous venons de voir comment déclarer et utiliser une fonction. Voyons comment l'inclure dans notre programme de comptage. L'idée est la suivante :

- La boucle principale fait défiler un nombre de 0 à 5 (grâce à une boucle for).
- Une fois le nombre défini, on fait appel à la fonction qui va allumer les LED représentant la bonne constellation sur le dé.
- Une fois l'affichage réalisé, on revient à notre programme principal.

Voici en pseudo-code ce que ça peut donner :

```
//définition des variables



//fonctions principales setup et loop

void setup(){

//on place les pins en mode OUTPUT

}

void loop(){

for (t=0;t<6;t++){

//on teste le nombre t et on envoie le programme à la bonne fonction

}

}



//définition des fonctions

void afficheZero(){

//code pour tout éteindre

}

void afficheUn(){

//code pour le 1

}

void afficheDeux(){

//code pour le 2

}

void afficheTrois(){

//code pour le 3
```

```
}

void afficheQuatre(){

//code pour le 4

}

void afficheCinq(){

//code pour le 5

}
```

Je vous laisse essayer de construire ce programme. C'est important de passer par cette phase d'essai/erreur pour pouvoir se débrouiller seul par la suite.

Allumez vos diodes pour chaque face de dé

Afin de mieux appréhender les fonctions et leurs possibilités je vais vous fournir le code que je propose pour l'allumage des diodes en fonction d'un nombre, afin de vous l'expliquer pas à pas.

```
//déclaration des variables

int ledHautGauche=2;

int ledBasGauche=3;

int ledCentre=4;

int ledHautDroite=6;

int ledBasDroite=5;



//fonction d'initialisation

void setup() {

  //mise en mode OUTPUT des pins 2 à 6 et positionnement en LOW

  for (int t=2;t<7;t++){

    pinMode(t,OUTPUT);

    digitalWrite(t,LOW);
```

```
  }
}


//boucle principale
void loop() {
  // boucle pour faire varier le nombre
  for (int nb=0;nb<6;nb++){
    affichage(nb);//appel de la fonction d'allumage des LEDs
    delay(500);
  }
}


// déclaration des fonctions


//cette fonction sert à éteindre toutes les diodes
void setZero(){
  for (int t=2;t<7;t++){
    digitalWrite(t,LOW);
  }
}


//cette fonction récupère un nombre et allume les LED en conséquence
void affichage(int nombre){
  setZero();//appel de la fonction qui éteint toutes les LED
```

```
//il suffit maintenant d'allumer les bonnes diodes
//en testant la valeur de 'nombre'
if (nombre==1){
  digitalWrite(ledCentre,HIGH); //on allume la diode du centre
  return;//sortie de la fonction
}
if (nombre==2){
  //on allume les diodes haut/droite et bas/gauche
  digitalWrite(ledHautDroite,HIGH);
  digitalWrite(ledBasGauche,HIGH);
  return;//sortie de la fonction
}
if (nombre==3){
  //on allume les diodes centre, haut/gauche, bas/droite
  digitalWrite(ledHautDroite,HIGH);
  digitalWrite(ledCentre,HIGH);
  digitalWrite(ledBasGauche,HIGH);
  return;//sortie de la fonction
}
if (nombre==4){
  //on allume toutes les diodes sauf celle du centre
  digitalWrite(ledHautGauche,HIGH);
  digitalWrite(ledHautDroite,HIGH);
```

```
    digitalWrite(ledBasGauche,HIGH);

    digitalWrite(ledBasDroite,HIGH);

    return;//sortie de la fonction

  }

  if (nombre==5){

    //on allume toutes les diodes

    digitalWrite(ledHautGauche,HIGH);

    digitalWrite(ledBasGauche,HIGH);

    digitalWrite(ledHautDroite,HIGH);

    digitalWrite(ledBasDroite,HIGH);

    digitalWrite(ledCentre,HIGH);

    return;//sortie de la fonction

  }

  //inutile de tester le 0 car on a commencé par tout éteindre

  return;//sortie de la fonction

}
```

Dans la fonction `affichage` qui prend un nombre en paramètre, vous voyez qu'on teste de quel nombre il s'agit et on allume les LEDs en fonction de la constellation à obtenir.
Vous remarquerez le mot-clé `return` qui permet de quitter une fonction pour revenir au programme précédent. Le tout dernier `return` de cette fonction n'est pas obligatoire dans cette fonction. En effet en arrivant au bout, le programme de fonction sera quitté pour revenir au programme principal.

On remarque dans ce programme que l'on peut appeler une fonction depuis une autre fonction. Sachez que l'on peut même appeler une fonction depuis elle-même !
Nous avons vu précédemment qu'une variable a un scope (une existence) qu'entre les accolades où elle a été créée. C'est valable aussi pour les variables dans les fonctions :

- Une variable créée **à l'intérieur d'une fonction** n'est pas accessible en dehors de cette fonction.
- Une variable créée **en dehors d'une fonction** n'est pas accessible par cette fonction (sauf les variables globales créées en début de programme).

Le programme final pour le dé à 5 LEDs

Alors j'ai deux nouvelles pour vous :

Vous voulez commencer par la mauvaise ? Ça me va !

Je ne vais pas vous donner le montage et le programme final ici, c'est vous qui allez devoir les réaliser dans l'activité notée de cette partie. (Lukas, refermez la bouche, ce n'est pas si terrible !)

La bonne nouvelle est que je vais vous aider en vous donnant la structure générale du code en pseudo-code. (Mais si Lukas, c'est une bonne nouvelle !)

Voici donc votre guide de programmation :

```
//déclaration des variables

  //LEDs sur les pins 2 à 6

  //bouton sur le pin 7



//initialisation

void setup(){

    //appel de la fonction randomSeed() pour la séquence aléatoire

    //pin 2 à 6 en mode OUTPUT

    //pin 7 en mode INPUT_PULLUP

    //appel de la fonction setZero() pour tout éteindre

}



//boucle principale

void loop(){

    //test de l'état du bouton avec boolean etatBouton
```

```
    if (le bouton est cliqué){

            for...  //boucle for qui temporise l'affichage de plus en plus lentement

            {

                //exctinction brève des LEDs (100ms)

                //tirage d'un nombre aléatoire avec la fonction random()

                //appel de la fonction d'allumage des LEDs

                //temporisation

            }//sortie de la boucle de temporisation

    }//sortie du test bouton

}



//déclaration de la fonction setZero()

void setZero(){

    //code pour tout éteindre

}

//déclaration de la fonction d'allugmage

void allumage(int nombre){

   //code pour l'affichage en fonction du nombre

}
```

Vous avez tout ce qu'il vous faut dans les chapitres précédents et dans celui-ci pour y arriver. Je vous conseille tout de même réellement d'essayer sans copier-coller des bouts de code que vous ne comprenez pas.

Une bonne astuce pour mieux comprendre un code, est de changer le nom des variables. Cela permet souvent de mieux comprendre à quoi elles servent et quelles sont leurs limites. Rendez-vous dans l'activité de cette partie pour compléter ce programme de dé à 5 LED ! Vous la trouverez dans le sommaire du cours, intitulée "Complétez un dé numérique à 5 LED".

Vous avez terminé l’activité, ou bien vous voulez vous plonger dedans plus tard ? Alors rejoignez-moi dans le chapitre suivant pour découvrir tout le potentiel du potentiomètre !

Utilisez les potentiomètres, les entrées analogiques et la fonction de mappage

Jusqu’ici, vous avez utilisé des entrées numériques qui ne prennent que deux valeurs : 1 ou 0. Vous allez à présent découvrir comment utiliser des entrées analogiques qui permettent de recevoir des données électriques plus subtiles.

Cette découverte des entrées analogiques se fera à travers la prise en main d’un nouveau composant : le potentiomètre analogique. Vous allez apprendre à vous en servir pour créer toute une nouvelle gamme de réactions en traitant des données qui ne sont pas uniquement des 1 et des 0. Cela vous amènera à enrichir votre vocabulaire de programmation avec la notion de mappage de valeurs.

Tout ceci sera bien sûr l'occasion de créer de nouveaux programmes, comme un clignotant à vitesse variable.

Vous aborderez enfin le principe de joystick, si cher à nos jeux préférés...

Le potentiomètre analogique

Le potentiomètre analogique se présente sous plusieurs formes. Mais avant d'aller plus loin, il faut en comprendre la fonction.

Vous avez, jusqu'ici, utilisé des résistances. Je vous rappelle que ces résistances ont des mesures précises (100 Ω, 220 Ω, 1 kΩ,10 kΩ...) et qu'une résistance ne peut être remplacée que par une résistance de valeur identique.

Et bien les ingénieurs électroniques (plus précisément Johann Christian Poggendorff) ont créé une forme de résistance qui varie : le potentiomètre ! C'est très pratique et depuis on l'utilise dans bien des domaines !

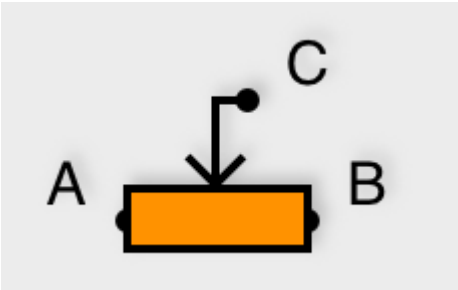
Il existe deux principaux types de potentiomètres analogiques : les potentiomètres **linéaires** et les potentiomètres **logarithmiques** (bon en vrai il y a une troisième forme qui a peu d'intérêt pour nous).

Si je précise potentiomètre analogiques, c'est qu'il existe des potentiomètres numériques qui ne font pas l'objet de ce chapitre.

Si vous observez bien, vous voyez que tous ces potentiomètres ont trois pattes ! C'est une nouveauté, en effet, jusqu'à maintenant, tout ce que nous avions connecté n'avait que deux pattes.

Le principe est assez simple : un curseur se déplace sur une piste résistante (au passage de l'électricité) et permet de faire varier la résistance du potentiomètre. Donc l'électricité entre par un côté du potentiomètre et sort par le curseur. Si le curseur est proche de l'entrée, la résistance est faible et si le curseur est éloigné de l'entrée, la résistance et grande.

D'ailleurs le symbole pour représenter un potentiomètre en électronique est assez parlant :



La flèche représente le curseur. En réalité, vous devrez connecter le potentiomètre sur la patte du curseur (celle au centre, notée C sur l'image) et connecter ensuite l'une des deux pattes restantes (soit A ou B) sur le +5V et l'autre sur le ground.

On dit qu'il s'agit d'un potentiomètre linéaire, car le déplacement du curseur fait augmenter la résistance de façon régulière. On peut donc s'attendre, en positionnant le curseur bien au centre, que la valeur de la résistance soit égale à la moitié de la résistance totale possible. Un exemple sera sûrement plus parlant !

Si vous utilisez un potentiomètre de résistance maximum 10 KΩ :

- Si vous placez le curseur à fond du côté de la patte d'entrée, la valeur de résistance sera de 0 Ω.
- Si vous placez le curseur au milieu, la valeur de résistance sera de 5 KΩ (la moitié du maximum).
- Si vous placez le curseur à fond de l'autre côté, la valeur de la résistance sera de 10 kΩ (valeur max).

Il existe des potentiomètres droits (on déplace le curseur sur une ligne) ou des potentiomètres circulaires (on déplace le curseur en le faisant tourner). Certains nécessitent l'utilisation d'un tournevis pour le faire tourner, d'autres ont une tige qui permet la rotation.

Ha oui ! Je vois, c'est comme sur mon ampli de guitare, avec le bouton du volume ?

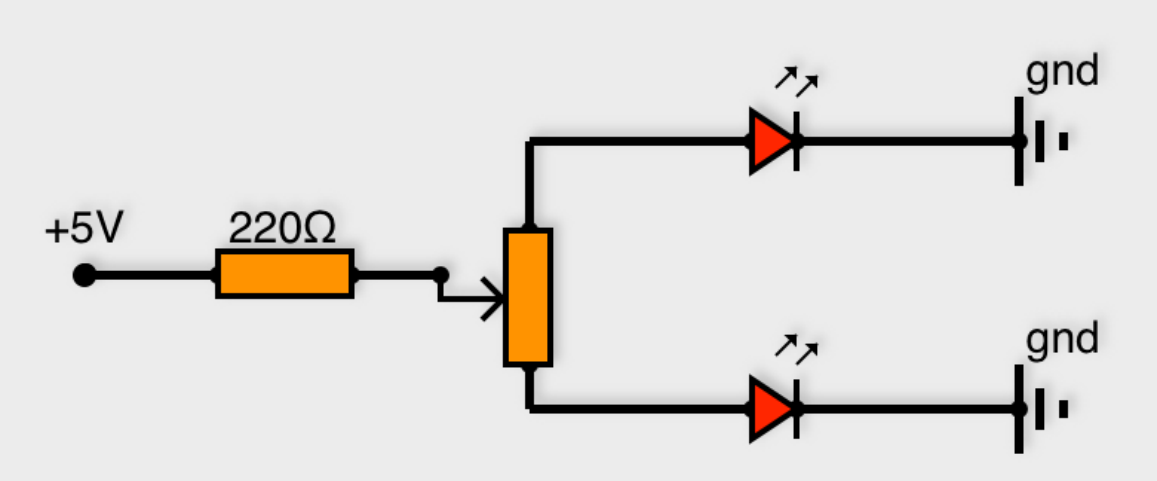
Et bien non, pas tout à fait. Les boutons de volume sur la guitare, ou les amplis, ou sur des enceintes, sont bien des potentiomètres, mais il ne sont pas dits linéaires, mais logarithmiques (non Lukas, rien à voir avec le fait de ne pas avoir le rythme...).

Logarithmique veut dire, sans trop entrer dans les détails, que quand on fait varier la position du curseur, on n'ajoute pas une valeur de résistance mais on la multiplie donc :

- À la patte d'entrée, la valeur de la résistance est bien 0 Ω.
- À la patte de sortie, la valeur de la résistance est bien au maximum.
- Mais au milieu d'un potentiomètre logarithmique, la valeur de la résistance ne correspond pas à la moitié du maximum. Et le calcul n'est pas un calcul simple à effectuer.

Dans un **potentiomètre linéaire**, la troisième patte permet d'obtenir la résistance complémentaire. C'est-à-dire que si on branche une patte vers une lampe, la patte du milieu vers l'alimentation et la troisième vers une autre lampe (les deux lampes branchées vers le ground), en faisant varier le curseur, la luminosité de la première lampe augmentera pendant que celle de la deuxième lampe diminuera. À la position centrale, les deux lampes brilleront de manière identique.
Nous utiliserons dans les montages de ce cours **uniquement des potentiomètres linéaires** car ce qui nous intéressera justement, c'est la variation prévisible de la résistance.

Je vous propose un premier montage de test, mais cette fois, sans image Fritzing pour vous habituer à lire ce genre de montage :



Montage pour faire varier deux diodes en opposition
Vous avez compris le schéma ? Il s'agit de connecter la résistance vers le +5 V de l'Arduino, puis la patte centrale du potentiomètre à la résistance. On connecte ensuite chaque autre patte du potentiomètre à la borne + d'une diode, et chaque diode au ground.

Si le montage est correct, en faisant varier le potentiomètre, vous voyez augmenter la luminosité d'une diode pendant que l'autre diminue en luminosité.

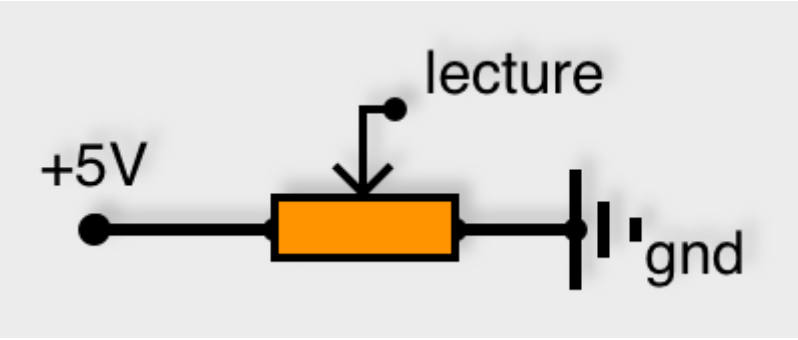
On garde tout de même dans ce montage la résistance de 220 Ω pour protéger les diodes quand le potentiomètre est au minimum.
Nous pouvons donc dire qu'avec un potentiomètre, nous avons un moyen de faire varier la résistance du passage du courant, donc à intensité égale nous faisons varier la tension aux bornes du potentiomètre !!! (Lukas, plutôt que de feindre la compréhension, allez donc relire la section précédente sur la loi d'Ohm

C'est justement cette variation de tension que notre Arduino est capable de lire !

Les entrées analogiques

Pour mieux comprendre ce qui se passe dans l'Arduino, nous allons commencer par bien faire la distinction entre entrée analogique et numérique.
Comme il s'agit d'entrée électriques dans notre cas, on parlera plutôt de **signal** analogique ou numérique. Pour faire simple, un signal analogique peut prendre une infinité de valeurs entre deux bornes alors qu'un signal numérique n'en prendra qu'un certain nombre précis.

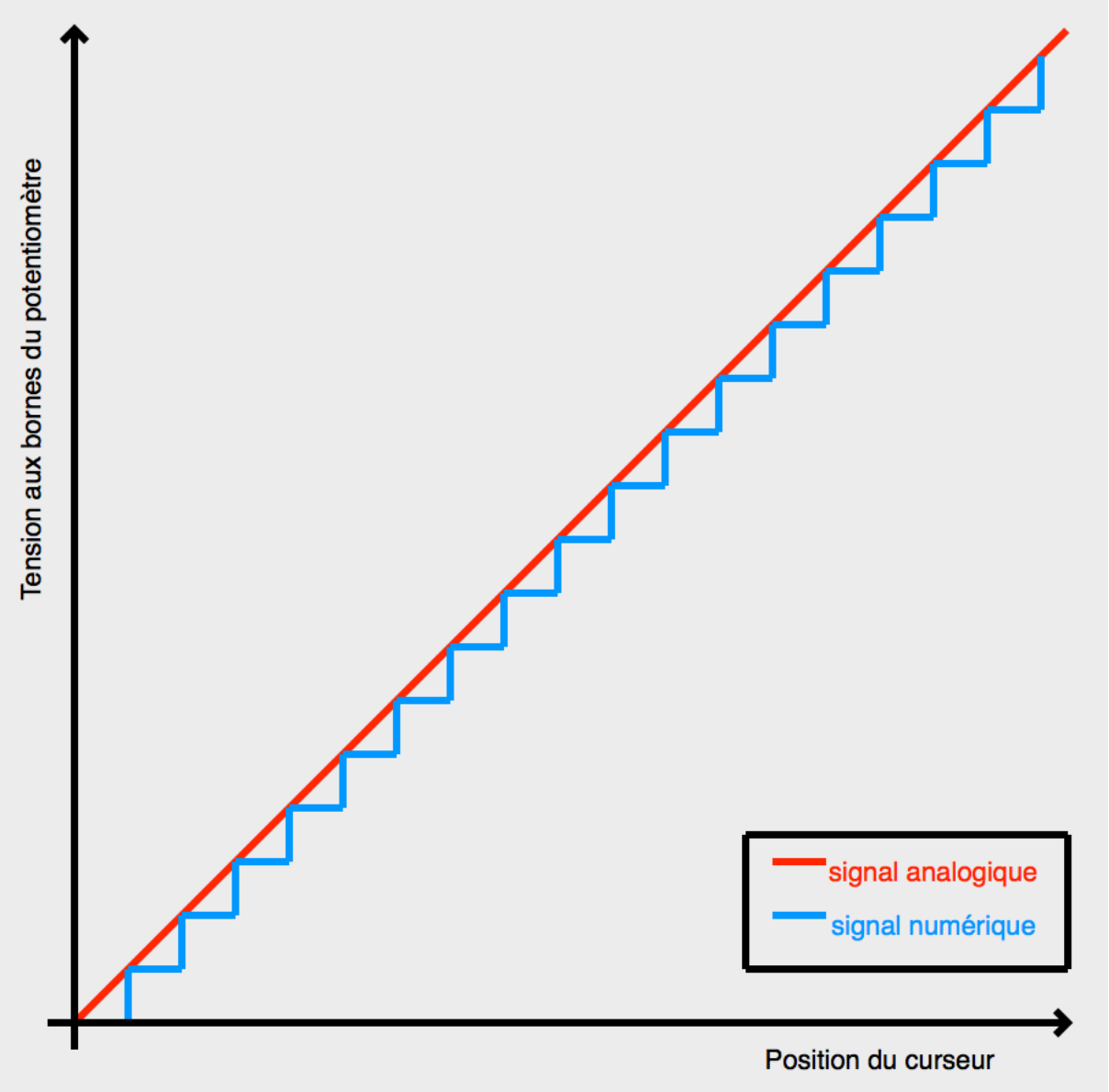
Petite illustration avec le potentiomètre et le montage qui suit :



Potentiomètre seul

Sur cet exemple, en faisant tourner le bouton du potentiomètre (curseur), vous ferez varier la tension à ses bornes de 0 V à 5 V. Cette variation dépendra de la rotation, de votre précision... En tous cas vous pourrez obtenir un nombre infini de valeurs entre 0 et 5 volts. Et il sera difficile d'obtenir précisément 3,234532 V. C'est ce que l'on appelle un **signal analogique**, dont toutes les valeurs possibles peuvent être représentées par la ligne rouge ci-dessous.

Le schéma ci-dessus n'est pas à réaliser tel quel, il reste théorique. En effet, si le potentiomètre est positionné de façon à avoir une résistance nulle (flèche tout à gauche), on obtient un court-circuit (un lien direct entre le +5V et le gnd) qui peut abîmer votre carte Arduino.
En revanche un **signal numérique** prendra, lui, un nombre de valeurs définies (c'est le principe de l'escalier), ce que l'on peut voir en bleu sur le schéma. On peut en fait imaginer que les valeurs progressent cran par cran. La valeur de ces crans s'appelle la **résolution** du signal numérique.



Évolution des valeurs numériques et analogiques de la tension d'un potentiomètre
Chercher la valeur numérique d'un signal analogique s'appelle la **quantification**.

Pour transformer un signal analogique en signal numérique, il nous faut un **convertisseur analogique numérique**, qu'on appelle plus simplement **CAN.**

Et bien là où vous avez une chance inouïe, c'est que l'Arduino possède non pas 1, ni 2, ni 3 CAN, mais 6 CAN !!!

En effet, vous pouvez transformer jusqu'à 6 signaux analogiques en signaux numériques, donc en nombres entiers ! Ce n'est qu'une fois le signal transformé en nombre qu'il est exploitable avec un programme.

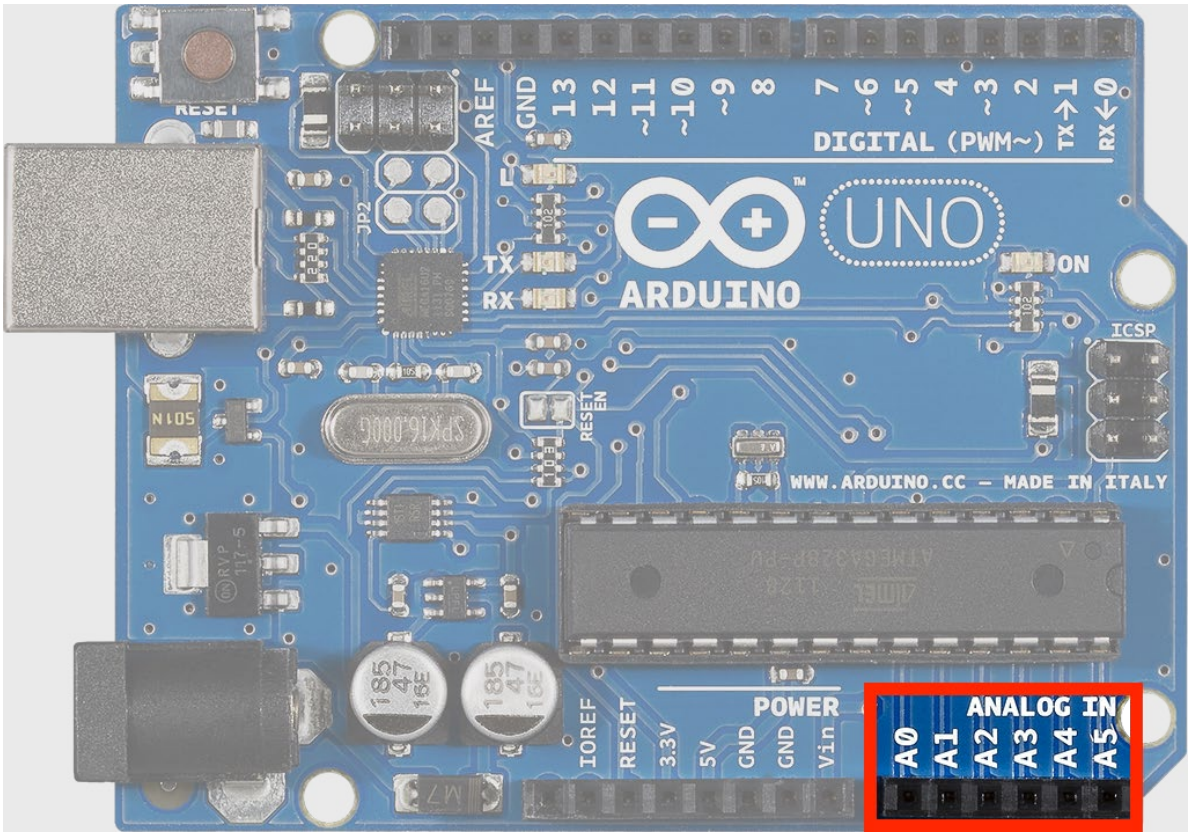
L'Arduino a tout de même une limite de définition, c'est à dire de résolution du signal. En effet, il ne peut transformer un signal reçu qu'en nombre compris **entre 0 et 1023**, soit 1024 valeurs possibles. Ce qui, vous le verrez, est déjà très bien.

L'autre limite de l'Arduino est que le signal reçu ne doit pas être supérieur à **+5V**.

Si vous ne respectez pas cette limite de +5V, vous vous exposez à abîmer votre carte Arduino et à attendre Noël pour en avoir une autre...

Mais où sont donc ces 6 CAN ?

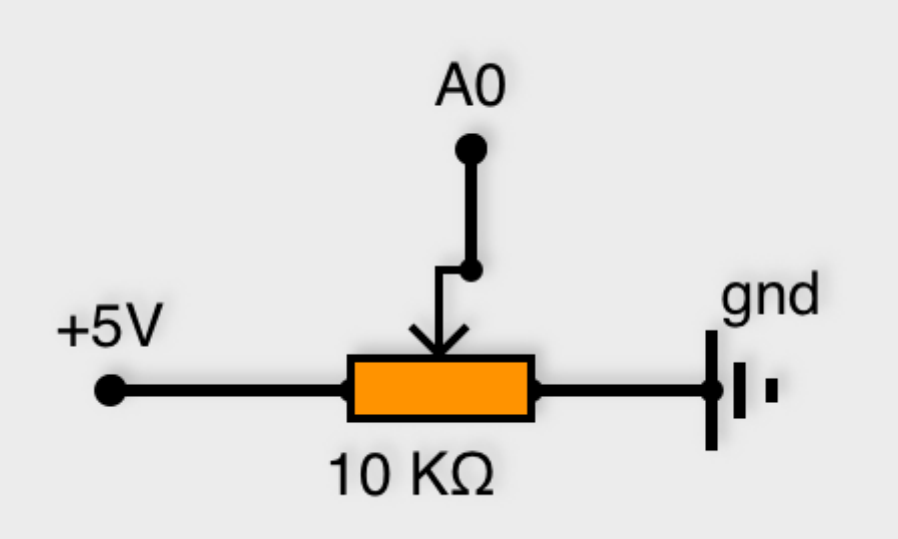
Prenez votre carte et observez cette zone :



Vous voyez 6 connexions possibles, notées A0, A1, A2, A3, A4, et A5. C'est ici que l'on peut connecter un fil pour recevoir un signal analogique. L'Arduino le transformera ensuite en signal numérique compris entre 0 et 1023.

Nous allons donc de ce pas réaliser un montage et un programme qui permet de numériser (c'est-à-dire convertir une valeur analogique en valeur numérique) les valeurs de la tension aux bornes d'un potentiomètre.

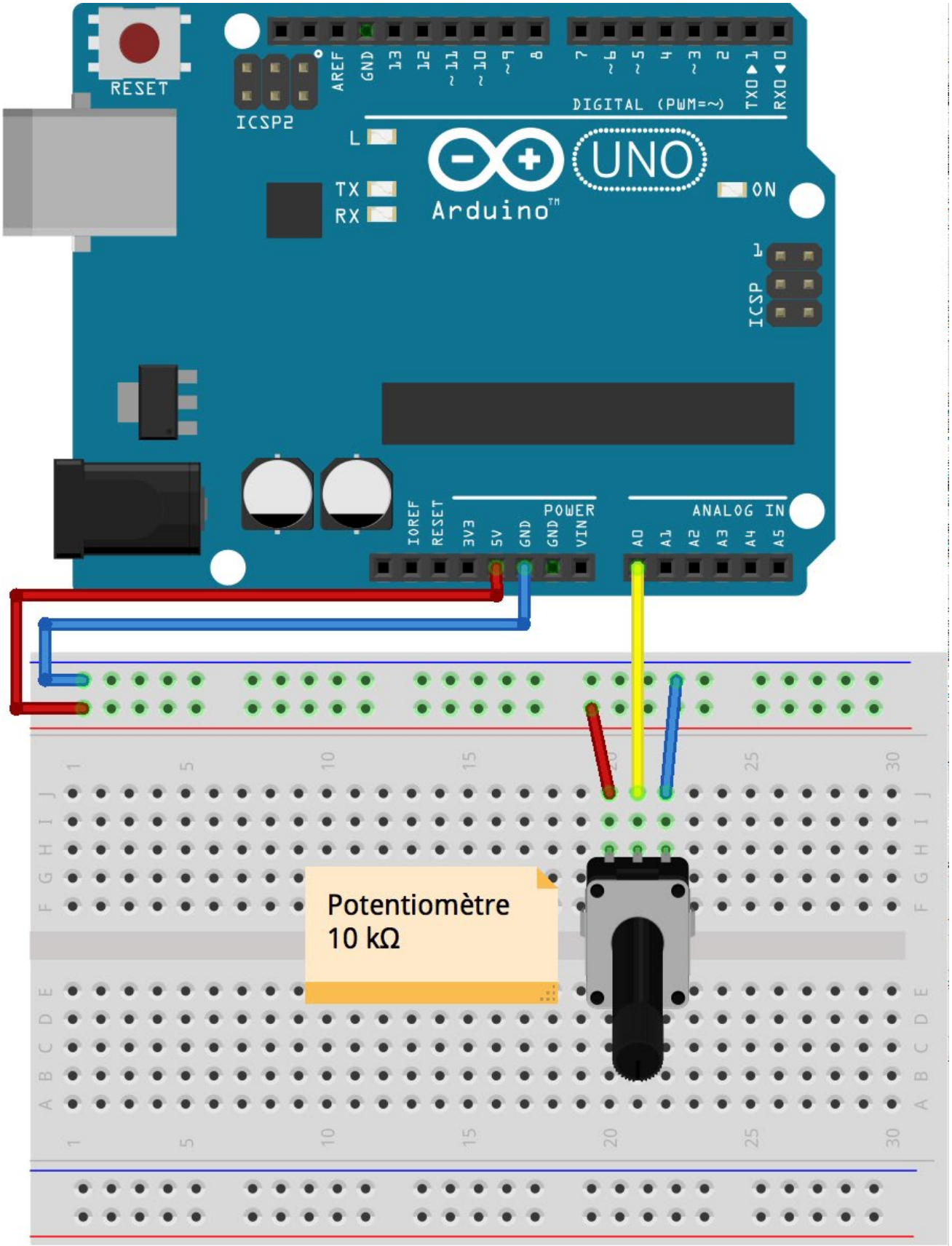
Tout d'abord, voici le schéma électrique :



Les trois broches sont utilisées : l'une des extrémités au +5V, l'autre au ground, et le curseur au pin A0.

Nous voyons donc que le courant part du +5V, entre dans le potentiomètre par l'une des deux pattes d'extrémité, puis sort par le curseur vers le pin A0 de l'Arduino. La connexion au ground permet la lecture par l'Arduino de la valeur de position du curseur.

Si vous oubliez la connexion au +5V, vous obtiendrez uniquement la valeur 0 et si vous ne connectez pas le ground, vous obtiendrez la valeur 1023. Même si vous tournez la molette du potentiomètre.
Voici l'image de montage :


RESET
ICSP2
AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
Arduino™
UNO
ON
ICSP
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
Potentiomètre
10 kΩ

fritzing

Vous remarquerez que j'utilise un potentiomètre rotatif de valeur 10 kΩ, je vous propose de faire de même.

Bien, maintenant comment allons nous procéder pour récupérer la valeur ?

Nous allons utiliser un mot-clé du langage de programmation : `analogRead(pin);`. La valeur "pin" entre les parenthèse peut être : A0, A1, A2, A3, A4 ou A5. Il est possible d'enlever le A devant le chiffre et d'écrire juste : 0, 1, 2, 3, 4 ou 5. Les deux écritures suivantes sont correctes pour lire une valeur sur le pin A2 :

```
analogRead(A2);


analogRead(2);
```

Il n'est pas possible de lire des valeurs analogiques sur les pins numériques (de 0 à 13) qui sont de l'autre côté de la carte. Il ne faut donc pas confondre analogRead(2) et digitalRead(2). À noter qu'il est possible d'utiliser les ports analogiques de l'Arduino comme des ports numériques. Il suffit de déclarer dans le setup() le port en mode INPUT, INPUT_PULLUP ou OUTPUT avec la fonction pinMode(pin, mode). On utilise ensuite digitalWrite() ou digitalRead().

ex : pinMode(A0,OUTPUT); définit le pin A0 de l'arduino comme pin numérique en sortie. Il faut bien mettre le A devant le nom du pin.
Grâce à la fonction `analogRead`, l'Arduino va nous renvoyer un nombre de type `int` compris entre 0 et 1023. Il est de bon ton de le stocker dans une variable de type `int`. Par exemple comme ceci :

```
int valeurPot; //déclaration de la variable qui va stocker la valeur numérique de la
tension du potentiomètre


valeurPot=analogRead(A0); //On stocke la valeur lue par le CAN A0 dans la variable
valeurPot
```

Et mieux encore, pour une lecture du code plus agréable et des connexions plus simples, on peut stocker la valeur du pin concerné dans une variable de type `int`. Donc comme ceci :

```
int pinPot=0; //on va lire sur le pin A0 de l'arduino


int valeurPot; //on déclare la variable qui va stocker la  tension du potentiomètre


valeurPot=analogRead(pinPot); //on lit et stocke la valeur.
```

Voici donc le code en question :

```
int pinPot=0; //variable pour définir le CAN où est connecté le potentiomètre
```

```
int valPot=0; //variable pour récupérer la tension aux bornes du potentiomètre traduite par le CAN . On l'initialise à 0.


void setup() {


  Serial.begin(9600); //Initialisation de la communication avec la console


}




void loop() {


  valPot=analogRead(A0); //lit la tension, la convertit en valeur numérique et la stocke dans valeurPot


  Serial.print("Valeur lue : ");


  Serial.println(valPot);


}
```

Deux remarques :

- Vous remarquerez que je n'ai pas utilisé le même nom de variable `valeurPot` dans cet exemple : c'est pour vous montrer que ces noms ne sont pas fixes, c'est à nous développeurs de les définir de façon pertinente pour qu'ils soient clairs et respectent les conventions générales de nommage.
- La valeur de `valPot` est initialisée à 0. C'est une bonne habitude à prendre que d'initialiser ses variables. En effet, si on l'oublie, elle peuvent prendre des valeurs incohérentes.

Vous remarquerez que la précision de l'Arduino (sa résolution) est largement suffisante. En effet, il est très difficile de réussir à faire progresser les nombres de 1 en 1.

Voyons donc maintenant une application possible de ce que nous venons d'apprendre... (Lukas, ce n'est pas un accélérateur, et merci de nous faire grâce des bruits de moto !)

Le clignotant à vitesse variable

Vous avez déjà, dans les chapitres précédents, fait clignoter une LED. Je vous propose de concevoir un programme qui répondrait au cahier des charges suivant (ça claque de le dire comme ça je trouve...) :

- La LED du montage clignote plus ou moins vite en fonction de la position du curseur du potentiomètre.
- Au plus lent, elle clignote toutes les secondes. C'est à dire qu'elle s'allume 0,5 s et s'éteint pendant 0,5 s.
- Au plus rapide, elle clignote 5 fois par seconde (5 Hz), donc elle s'allume pendant 0,1 s et s'éteint pendant 0,1 s de façon répétitive.

- Elle doit prendre les valeurs précises de clignotement de 1 fois par seconde, 2 fois par seconde, 3 fois par seconde, 4 fois par seconde ou 5 fois par seconde. Mais pas de valeurs intermédiaires. Le temps d'allumage est le même que le temps d'extinction.

Je vous propose de chercher d'abord seul (Lukas je vous ai entendu... oui je l'ai déjà dit, mais je le répète !) pour voir la difficulté éventuelle de cette réalisation.

Petite aide : la plus grosse difficulté de ce programme réside dans l'obligation des temps de clignotement. Cela peut se résoudre avec un calcul mathématique ou une série de `if` et `else if`.
Bon, je propose pour réussir ce projet de procéder par étape. Tout d'abord un clignotement sans l'obligation des temps de clignotement. En gros, on récupère la valeur du potentiomètre et on s'en sert de temps d'attente pour l'allumage et l'extinction.

Voici le code :

```
int pinPot=0; //variable pour le CAN utilisé

int valPot=0; //variable qui va stocker la tension lue. On l'initialise à 0.

int pinLED=8; // pin de connexion de la LED

void setup() {

  pinMode(pinLED,OUTPUT); //Mode OUTPUT pour le pin de LED

  digitalWrite(pinLED,HIGH);//On allume la LED

}



void loop() {

  valPot=analogRead(A0); //lit la valeur de tension, la transforme et la stocke dans la
variable

  int attente=valPot; //le délai d'attente est la valeur du potentiomètre

  digitalWrite(pinLED,HIGH); //on allume la LED

  delay(attente); //on attend en fonction de la variable attente

  digitalWrite(pinLED,LOW); //on éteint la LED

  delay(attente); //on attend
```

```
}
```

Les commentaires devraient suffire à le comprendre. Testez-le pour en voir le résultat.

Maintenant, comment faire pour que la variation de l'attente soit par paliers ?

Il suffit par exemple de faire un tableau de valeur d'attente. On sait que la valeur du potentiomètre est entre 0 et 1023. Il nous faut 5 types de clignotement. On divise donc 1024 par 5 :

1024/5=204.8 que l'on arrondit à 205 (C'est toujours mieux d'utiliser des entiers avec un ordinateur, ça prend moins de place en mémoire).

Donc il nous suffira de tester les valeurs et de définir la valeur d'attente qui correspond. Soit 1 000 millisecondes, divisées par le nombre de clignotement et divisé par 2 car même temps d'allumage que d'extinction.

Valeur du potentiomètre	Valeur d'attente
entre 0 et 204	1000/5/2=100ms
entre 205 et 408	1000/4/2=125ms
entre 409 et 613	1000/3/2=167ms (on arrondit)
entre 614 et 818	1000/2/2=250ms
entre 818 et 1023	1000/1/2=500ms

Vous remarquerez que la valeur d'attente est calculée selon la même règle, on pourrait donc la définir dans une variable `attente = int(1000/valInter/2)` où `valInter` correspond au nombre de clignotements par seconde.
Il suffit ensuite de faire les tests avec des `if` pour définir la valeur d'attente. Voici une première proposition de code :

```
int pinPot=0; //variable pour le pin où est connecté le potentiomètre

int valPot=0; //variable pour récupérer la valeur lue. On l'initialise à 0.

int pinLED=8;

void setup() {

  pinMode(pinLED,OUTPUT);

  digitalWrite(pinLED,HIGH);

}
```

```
void loop() {

  valPot=analogRead(A0); //lit la valeur, la transforme et la stocke dans la variable

  int attente=500;

  if (valPot>=0 && valPot<=204){

    attente=100;

  }

  if (valPot>=205 && valPot<=408){

    attente=125;

  }

  if (valPot>=409 && valPot<=613){

    attente=167;

  }

  if (valPot>=614 && valPot<=818){

    attente=250;

  }

  if (valPot>=818 && valPot<=1023){

    attente=500;

  }

  digitalWrite(pinLED,HIGH);

  delay (attente);

  digitalWrite(pinLED,LOW);

  delay(attente);

}
```

Vous remarquerez les tests entre les parenthèses des `if` : on utilise le `&&` qui signifie en informatique : "et".
Nous reviendrons dans un autre chapitre aux différentes possibilités offertes par les tests.

Voici maintenant un code différent qui est basé sur le même principe, qui fait donc la même chose, mais dont la construction des tests change :

```
int pinPot=0; //variable pour le pin où est connecté le potentiomètre

int valPot=0; //variable pour récupérer la valeur lue. On l'initialise à 0.

int pinLED=8;

void setup() {

  pinMode(pinLED,OUTPUT);

  digitalWrite(pinLED,HIGH);

}



void loop() {

  valPot=analogRead(A0); //lit la valeur, la transforme et la stocke dans la variable

  int attente=100;

  if (valPot>817){

    attente=500;

  }

  else if (valPot>613){

    attente=250;

  }

  else if (valPot>408){

    attente=167;

  }

  else if (valPot>204){

    attente=125;
```

```
  }

  digitalWrite(pinLED,HIGH);

  delay (attente);

  digitalWrite(pinLED,LOW);

  delay(attente);

}
```

Vous remarquez que j'utilise un `else if` qui permet de passer aux autres tests si le test précédent n'est pas vérifié. Je fais gagner du temps à l'Arduino car c'est plus rapide de comparer a>b que a>=b. Si aucun des tests n'est vérifié, la variable `attente` prend la valeur qu'on lui a assignée au début de la `loop` (valeur 500 dans notre exemple).
Et si on simplifiait encore plus ce code ?

Tout à l'heure nous avons vu que la valeur d'attente pouvait se définir comme une variable :

```
attente = int(1000/valInter/2)
```

Il ne reste donc plus qu'à calculer `valInter` ! Pour ça, il n'y a rien de tel qu'une petite règle de trois :

1024	5
valPot	valInter

où `valInter` représente la correspondance entre 1 et 5 des valeurs entre 0 et 1023 et est peut donc être calculée de la façon suivante :

```
int valInter=int(valPot*5/1024)+1
```

Explications :

- Je rappelle que le signe de la multiplication est l'étoile : `*`.
- On entoure le calcul de parenthèses en mettant 'int' devant, ce qui va transformer un nombre à virgule (type `float`) en nombre sans virgule (type `int`).
- On met 1024 dans le calcul (même si les valeurs vont de 0 à 1023). En effet, avec 1023 on obtient un nombre entre 0 et 5 (soit 6 positions). Là on obtient un nombre entre 0 et 4 (donc 5 positions).
- Le +1 permet d'avoir un nombre entre 1 et 5.

Maintenant que nous avons notre correspondance entre 1 et 5, nous reprenons le calcul précédent :

```
int attente=int(1000/valInter/2)
```

Ça représente 1 seconde divisée par le nombre de clignotements. On divise par deux pour avoir le temps qui correspond à l'allumage et à l'extinction.

Voici donc le code :

```
int pinPot=0; //variable pour le pin où est connecté le potentiomètre
```

```
int valPot=0; //variable pour récupérer la valeur lue. On l'initialise à 0.

int pinLED=8;

void setup() {

  pinMode(pinLED,OUTPUT);

  digitalWrite(pinLED,HIGH);

}



void loop() {

  valPot=analogRead(A0); //lit la valeur, la transforme et la stocke dans la variable

  int valInter=int(valPot*5/1024)+1; //produit en croix

  int attente=1000/valInter/2; // calcul du temps d'arrêt

  digitalWrite(pinLED,HIGH);

  delay (attente);

  digitalWrite(pinLED,LOW);

  delay(attente);

}
```

Et là, maintenant, devant vos yeux ébahis, je vais sortir de mon chapeau, ou plutôt du chapeau de l'Arduino, la petite formule qui va bien !!!

Le mappage de valeurs

Il faut savoir que le langage de l'Arduino propose directement une fonction qui va nous éviter tous ces calculs !

Elle se présente sous cette forme :

```
map(valeur,min,max,transMin,transMax);
```

avec :

- `valeur` : c'est la valeur que vous voulez transformer (la tension du potentiomètre du potentiomètre dans notre cas)

- `min` et `max` : c'est la plage de valeurs (le minimum et le maximum qu'elle peut prendre, soit ici 0 et 1 023 qui correspond à la plage de résolution du CAN de l'Arduino)
- `transMin` et `transMax` : c'est la plage de valeurs dans laquelle on doit transformer la `valeur` (ici 1 et 5, car nous attendons 5 positions différentes)

On peut donc utiliser la fonction map ainsi dans notre exemple :

```
int valInter=map(valPot,0,1023,1,5);
```

La fonction `map` ne renvoie que des valeurs de type `int`.
Voici donc le code modifié avec l'utilisation de la fonction `map` :

```
int pinPot=0; //stocke le CAN utilisé

int valPot=0; //variable pour récupérer la valeur de tension du potentiomètre. On
l'initialise à 0.

int pinLED=8;

void setup() {

  pinMode(pinLED,OUTPUT); //mode OUTPUT pour le pin de LED

  digitalWrite(pinLED,HIGH);//on allume la LED

}



void loop() {

  valPot=analogRead(A0); //lit la valeur de la tension, la numérise et la stocke dans
valPot

  int valInter=map(valPot,0,1023,1,5); //fonction de mappage

  int attente=1000/valInter/2; // calcul du temps d'arrêt

  digitalWrite(pinLED,HIGH); //on allume la LED

  delay (attente); //attente calculée

  digitalWrite(pinLED,LOW); //on éteint la LED

  delay(attente); //même attente

}
```

Ce qui est pratique avec cette fonction, c'est qu'elle peut aussi inverser des valeurs, ou donner une échelle de résultats négatifs. Voici deux exemples :

Appel de la fonction map()	Action	Résultat entier
`int valRes=map(128,0,1000,10,0);`	Transforme la valeur 128, initialement située entre 0 et 1000, en une valeur entre 10 et 0.	valRes=9
`int valRes=map(128,0,1000,-10,10);`	Transforme la valeur 128, initialement située entre 0 et 1000, en une valeur entre -10 et 10.	valRes=-8

Les calculs sont faits avec des valeurs intégrales c'est à dire des entiers dont la partie décimale est tronquée à chaque opération effectuée. Ceci peut expliquer la différence avec un calcul effectué avec des virgules.
Avant de passer au chapitre suivant, voyons une dernière application des potentiomètres qui devrait vous plaire…

Le joystick

Nous allons profiter d'avoir aborder les potentiomètres pour parler du joystick (oui Lukas, j'arrive aussi à traduire) que tous les gamers ont entre leurs mains.

Autrefois, le joystick ne permettait pas une grande précision dans le mouvement. En effet, il s'agissait d'un manche pouvant bouger sur deux axes : avant, arrière, droite et gauche. À chaque mouvement, le manche actionnait un interrupteur qui envoyait un signal binaire à l'ordinateur (quand l'axe était déplacé).

Grâce à l'échantillonnage numérique, les joysticks ont donc gagné en précision. En effet, au lieu d'interrupteurs simples, chaque axe est relié à un potentiomètre (un pour avant/arrière, un pour droite/gauche).

Ce qui permet maintenant, non plus d'avoir un résultat binaire (axe activé ou non) mais une grande nuance de positions possibles entre les axes avant et arrière, et droite et gauche.

Si vous êtes très bricoleur, vous pouvez tenter d'en fabriquer un. Mais sachez qu'il en existe qui sont déjà montés (juste quelques soudures à réaliser) et que l'on trouve dans le commerce comme celui qui suit par exemple, mais il n'est pas le seul format.

Joystick interfaçable avec l'Arduino
Vous pouvez voir un des deux potentiomètres à droite sous le manche.

Une fois connecté, il suffit de récupérer les valeurs des potentiomètres sur entrées analogiques de l'Arduino, de mapper les valeurs avec la fonction `map` pour qu'elles correspondent à vos besoins, et le tour est joué !
Petit bonus, ces joysticks ont aussi un bouton poussoir qui s'actionne en appuyant sur le manche verticalement.

Bien je crois qu'on a assez appris sur le potentiomètre, si nous passions au mouvement ?