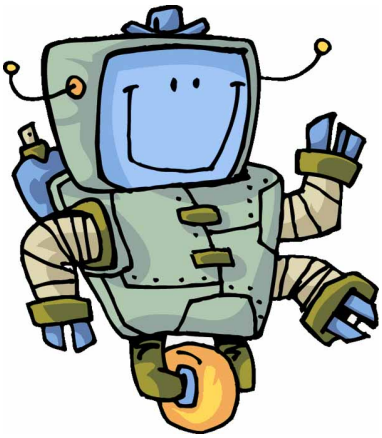


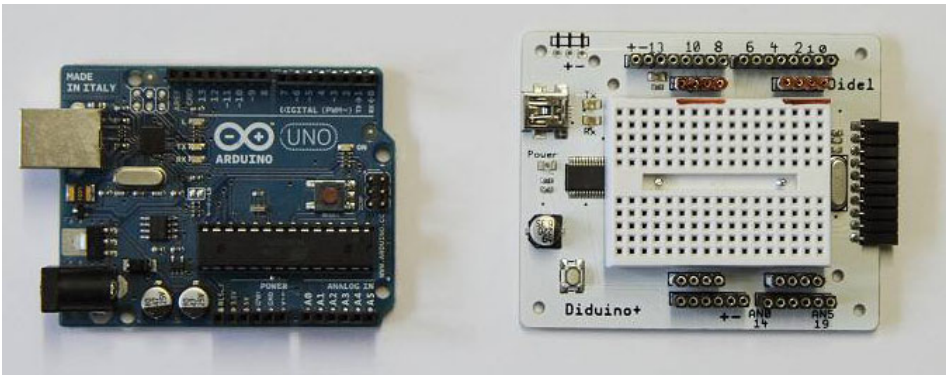
# *Chapitre 14*

# Robotique

## 14.1. Arduino ? Diduino ?

Le système Arduino est une plateforme de développement open-source. Il est composé d'une partie matérielle et d'une partie logicielle. Tout le software est en open-source sur le net : le système de développement, des librairies et des milliers d'applications. La documentation d'Arduino, dont les cartes sont développées et fabriquées en Italie, est en anglais. Mais on trouve des sites en français, par exemple http://arduino.cc/fr/

Arduino peut être utilisé pour développer des objets interactifs, pouvant recevoir des entrées d'une grande variété d'interrupteurs ou de capteurs, et pouvant contrôler une grande variété de lumières, moteurs ou toutes autres sorties matérielles. Les projets Arduino peuvent être autonomes, ou bien ils peuvent communiquer avec des logiciels tournant sur votre ordinateur.




*Les cartes Arduino UNO et Diduino*

Jean-Daniel **Nicoud**

En Suisse, Jean-Daniel Nicoud, ancien professeur à l'EPFL, a développé Diduino (**DID**el-Ard**UINO**) et le commercialise via son entreprise Didel. L'avantage de cette version d'Arduino est son orientation « expérimentation », son prix modique et la disponibilité du hardware nécessaire pour développer des applications. Une version spécifique pour la robotique existe aussi. Tout ceci est décrit sur le site de Didel : www.didel.com

Par rapport à d'autres plateformes, l'Arduino a l'avantage de ne pas de nécessiter de programmateur. La carte se relie à un PC par un câble USB et c'est tout. L'alimentation se fait par ce câble. Le programme est écrit en langage Arduino, proche du langage C, à l'aide de l'environnement de développement Arduino, librement téléchargeable sur le net.

### Remerciements

Ce cours est largement inspiré de celui de Brice Canvel. Merci à lui de m'avoir autorisé à l'utiliser. Adresse : http://mediawiki.e-apprendre.net/index.php/Diduino-Robot.

## Mise en route

En pratique, il faut écrire le programme sur le PC, le tester afin d'y déceler des erreurs de syntaxe, le compiler et le télécharger sur la carte Diduino. Une fois la carte « nourrie », il démarrera automatiquement avec ce programme lors de chaque mise sous tension. Le programme ne s'efface pas lorsqu'on met la carte hors tension.

Le logiciel Arduino véritable environnement de développement intégré, pour écrire, compiler et transférer le programme vers la carte. Pour télécharger le programme Arduino, il faut se rendre sur la page :

http://arduino.cc/en/Main/Software

Choisissez la version 1.0.3.

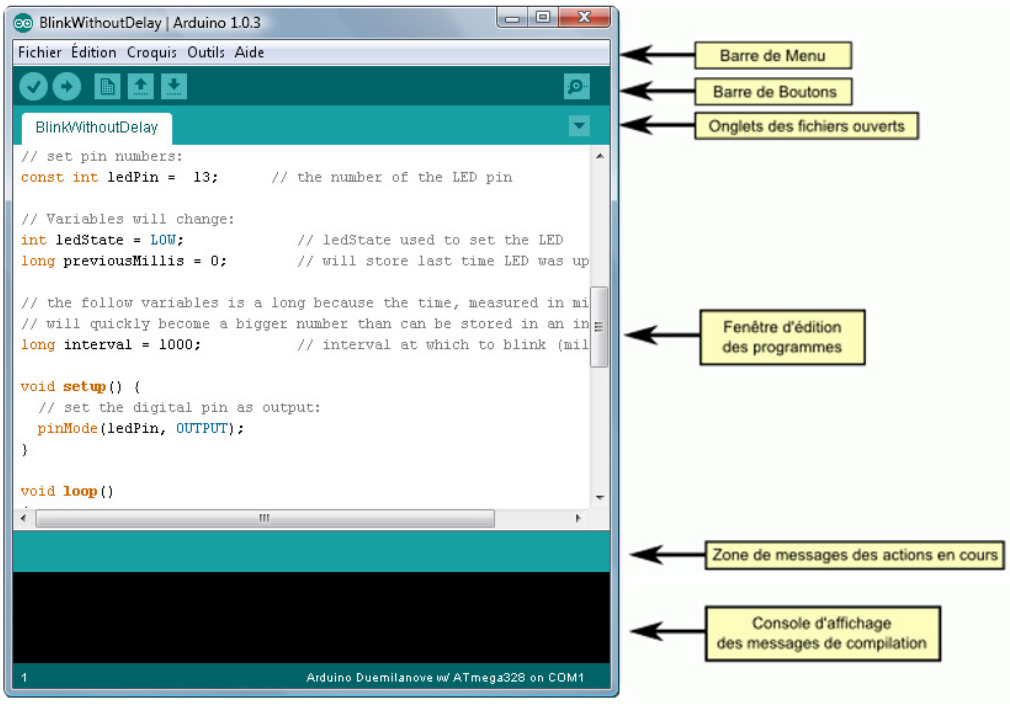
Il faut ensuite le décompresser et l'installer (cela prendra une dizaine de minutes). On peut l'installer sur un PC ou un clé USB. Le dossier décompressé pèse 246 Mo sous Windows. Il existe aussi une version Mac OS X et une version Linux.

# 14.2. Le logiciel Arduino

Ce paragraphe est un résumé de la page http://arduino.cc/fr/Main/DebuterPresentationLogiciel

Le logiciel Arduino a pour fonctions principales :

- de pouvoir écrire et compiler des programmes pour la carte Arduino
- de se connecter avec la carte Arduino pour y transférer les programmes
- de communiquer avec la carte Arduino

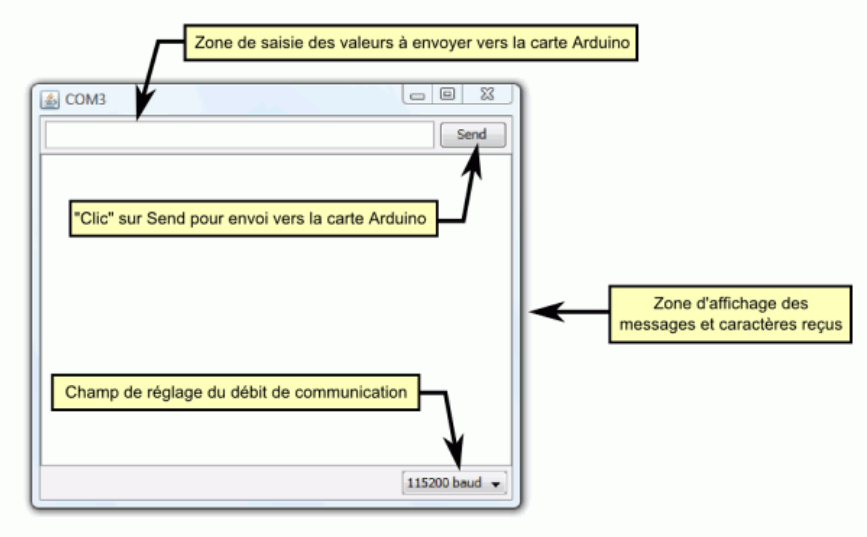


Cet espace de développement intégré (EDI) dédié au langage Arduino et à la programmation des cartes Arduino comporte :

- une **barre de menus** comme pour tout logiciel une interface graphique (GUI),
- une **barre de boutons** qui donne un accès direct aux fonctions essentielles du logiciel et fait toute sa simplicité d'utilisation,
- un **éditeur** (à coloration syntaxique) pour écrire le code de vos programme, avec onglets de navigation,
- une **zone de messages** qui affiche indique l'état des actions en cours,
- une **console texte** qui affiche les messages concernant le résultat de la compilation du programme.

Le logiciel Arduino intègre également :

- un **terminal série** (fenêtre séparée) qui permet d'afficher des messages textes reçus de la carte Arduino et d'envoyer des caractères vers la carte Arduino. Cette fonctionnalité permet une mise au point facilitée des programmes, permettant d'afficher sur l'ordinateur l'état de variables, de résultats de calculs ou de conversions analogiques-numériques : un élément essentiel pour améliorer, tester et corriger ses programmes.




## 14.2.1. Principe général d'utilisation

Le code écrit avec le logiciel Arduino est appelé un *croquis* (sketch en anglais) :

- Ces croquis sont écrits dans l'éditeur de texte. Celui-ci a les fonctionnalités usuelles de copier/coller et de rechercher/remplacer le texte.
- La zone de messages donne l'état de l'opération en cours lors des sauvegardes, des exportation et affiche également les erreurs.
- La console texte affiche les messages produit par le logiciel Arduino incluant des messages d'erreur détaillés et autres informations utiles.
- La barre de boutons vous permet de vérifier la syntaxe et de transférer les croquis, créer, ouvrir et sauver votre code, et ouvrir le moniteur série.
- La barre des menus vous permet d'accéder à toutes les fonctionnalités du logiciel Arduino.

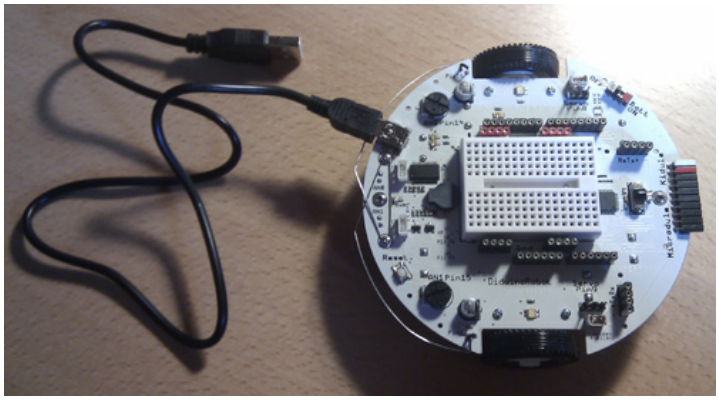
Le logiciel Arduino intègre le concept d'un « carnet de croquis » : un endroit réservé pour stocker vos programmes. Les programmes que vous mettez dans votre « carnet de croquis » pourront être ouvert directement depuis le menu « Fichier > Carnet de croquis » ou à l'aide du bouton « Ouvrir » dans la barre d'outils (4ème bouton depuis la gauche).

La première fois que vous démarrer le logiciel Arduino, un chemin automatique sera créé pour votre carnet de croquis. Vous pouvez voir ou modifier cette localisation depuis le « Fichier > Préférences ». Si votre logiciel Arduino se trouve sur une clé USB, il est recommandé de créer un dossier « croquis » sur votre clé USB aussi.

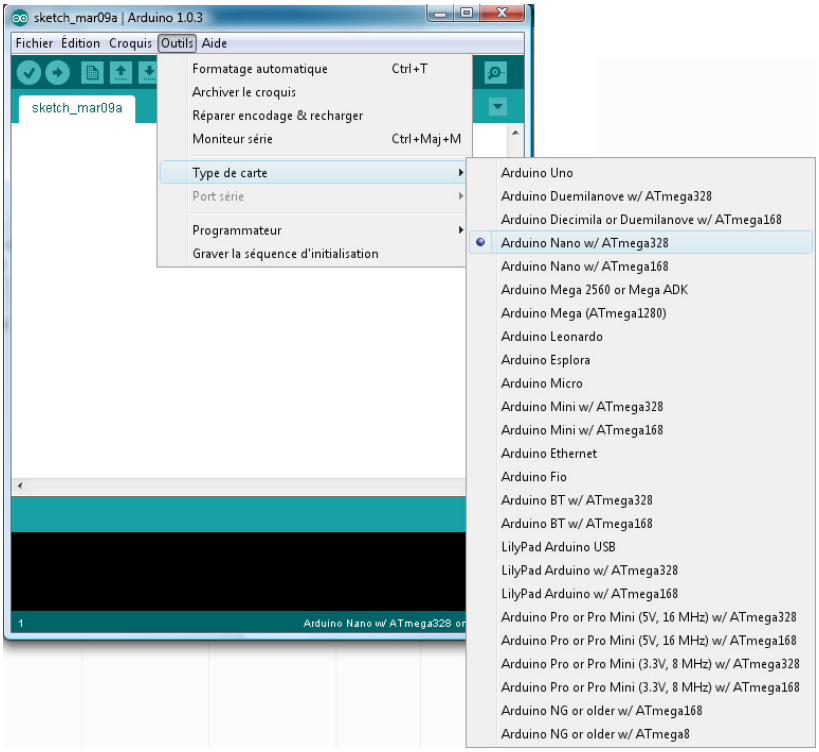
## 14.2.2. Prise en main (sous Windows 7)

### 1. Connecter le robot

Avant de lancer le programme Arduino, connectez le robot avec le câble USB fourni (s'il était connecté, déconnectez et rebranchez).



Lancez le logiciel Arduino. Une fenêtre apparaît.
Dans le menu Outils, choisissez la carte « Arduino Nano w/ATmega328 ».




Les ports d'entrée-sortie sont des éléments matériels de l'ordinateur, permettant au système de communiquer avec des éléments extérieurs, c'est-à-dire d'échanger des données. Le terme *série* désigne un envoi de données *via* un fil unique : les bits sont envoyés les uns à la suite des autres.

Il faut ensuite choisir le *port série*.
Il est possible que la première fois, on ne puisse pas choisir ou changer le port série (comme c'est le cas sur l'image ci-dessus). Dans ce cas :

1. allez dans le menu « Démarrer » de Windows 7 et choisissez « Périphériques et imprimantes ». Il apparaîtra un périphérique nommé « FT232R USB UART ».
2. double-cliquez sur l'icône du périphérique

3. choisissez l'onglet « Matériel »
4. double-cliquez sur le nom FT232R USB UART ; une nouvelle fenêtre s'ouvre
5. choisissez l'onglet « Pilote »
6. cliquez sur « Mettre à jour le pilote »
7. il faudra installer le pilote qui est dans le dossier Arduino sous « Drivers » puis « FTDI USB Drivers ».

Une fois le pilote installé, retournez dans le logiciel Arduino pour choisir le port série. Normalement, il faut choisir le plus grand numéro (p.ex. COM5).

### 2. Saisir le programme et vérifier le code

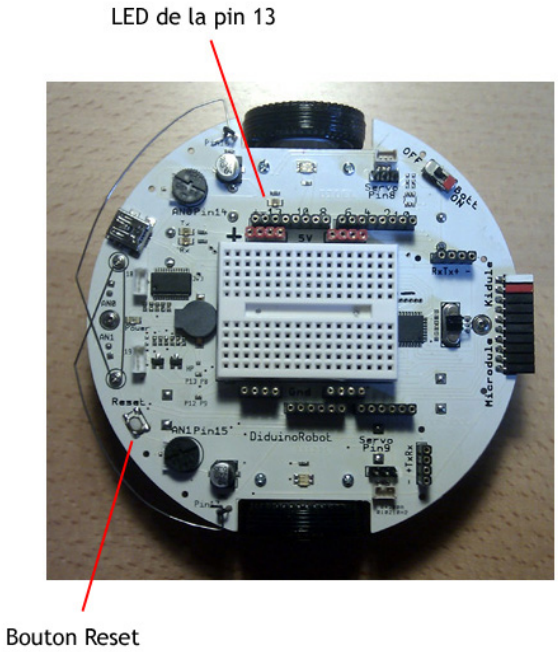
On suppose ici qu'un programme correctement écrit se trouve dans la fenêtre éditeur. Pour votre première programmation de la carte, allez dans le menu «Fichier > Exemples > 02.Digital > BlinkWithoutDelay »: un programme s'ouvre avec du code dans la fenêtre éditeur.

Appuyez alors sur le bouton « Vérifier » de la barre d'outils pour lancer la vérification du code.

Si tout va bien, aucun message d'erreur ne doit apparaître dans la console et la zone de message doit afficher « Done Compiling » attestant que la vérification s'est bien déroulée.

### 3. Transférer le programme sur la carte

Cliquez sur le bouton « Téléverser » dans la barre d'outils, ou bien sélectionnez le menu « Fichier > Téléverser ». La carte Diduino va alors automatiquement se réinitialiser et démarrer le transfert. Si elle ne se réinitialise pas toute seule, pressez le bouton Reset sur la carte Diduino.




### 4. Voir le résultat

Sur la carte Diduino, une Led est câblée sur la pin 13. Le programme que vous venez d'envoyer va la faire clignoter avec un délai d'une seconde.

## 14.3. Le langage Arduino / C

Pour commencer, nous ne verrons que les instructions indispensables. D'autres instructions apparaîtront peut-être dans les programmes et seront commentées.

Cette mini-référence a été adaptée de celle du site www.hb9afo.ch/arduino

### 14.3.1. setup()

La fonction **`setup()`** est appelée au démarrage du programme. Cette fonction est utilisée pour initialiser les variables, le sens des pins, etc.. La fonction **`setup()`** n'est exécutée qu'une seule fois, après chaque mise sous tension ou reset (réinitialisation) de la carte Arduino.

La fonction **`setup()`**, même vide, est obligatoire dans tout programme Arduino.

**Exemple**

```
int buttonPin = 3; // déclaration d'une variable globale

void setup() // fonction setup - début de l'exécution du programme
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

void loop()  // fonction loop - est exécutée en boucle
             // une fois que la fonction setup a été exécutée
{
  // ...
}
```

### 14.3.2. loop()

Après avoir créé une fonction **`setup()`**, qui initialise et fixe les valeurs de démarrage du programme, la fonction **`loop()`** s'exécute en boucle sans fin, permettant à votre programme de s'exécuter et de répondre. Utilisez cette fonction pour contrôler activement la carte Arduino.

La fonction **`loop()`** est obligatoire, même vide, dans tout programme.

**Exemple**

```
int buttonPin = 3;

// la fonction setup initialise la communication série
// et une pin utilisée avec un bouton poussoir

void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

// la fonction loop teste l'état du bouton à chaque passage
// et envoie au PC une lettre H si il est appuyé, L sinon.

void loop()
{
  if(digitalRead(buttonPin) == HIGH)
    Serial.write('H');
  else
    Serial.write('L');

  delay(1000);
}
```

### 14.3.3. ;

Le point-virgule est obligatoire à la fin d'une instruction.

Oublier le point virgule en fin de ligne donnera une erreur de compilation. Le texte d'erreur pourra être évident, et se référer à un point-virgule oublié, mais parfois cela sera moins évident. Si

une erreur de compilation incompréhensible et apparemment illogique survient, la première chose à vérifier est l'oubli d'un point-virgule juste avant la ligne que le compilateur déclare erronée.

### 14.3.4. {}

Les accolades sont un élément majeur de la programmation en C. Elles sont utilisées dans plusieurs constructions différentes :

**Fonctions**

```
void myfunction(datatype argument){
  // vos instructions ici
}
```

**Boucles**

```
while (boolean expression)
{
  // vos instructions ici
}

do
{
  // vos instructions ici
} while (boolean expression);

for (initialisation; termination condition; incrementing expr)
{
  // vos instructions ici
}
```

**Conditions**

```
if (boolean expression)
{
  // vos instructions ici
}
else if (boolean expression)
{
  // vos instructions ici
}
else
{
  // vos instructions ici
}
```

### 14.3.5. Commentaires

Les commentaires ont pour seul but de vous aider à vous rappeler comment votre programme fonctionne et en informer les autres. Il y a deux façons de créer des lignes de commentaires :

```
// Commentaire sur une seule ligne
```

```
/* Commentaire sur plusieurs lignes :
Tout ce texte
... est ignoré par le compilateur */
```

## 14.3.6. Opérateurs arithmétiques

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5; // division entière si r est un entier
```

- Il faut savoir que les constantes entières sont par défaut de type **`int`** (voir §3.7.1), et dès lors certains calcul entre constantes peuvent déborder (p. ex. 60*1000 donnera un résultat négatif).
- Choisir des tailles de variables assez grandes pour permettre de stocker les plus grands résultats issus des calculs.
- Savoir à quel moment votre variable débordera et ce qui se passe dans chaque sens du débordement.
- Pour les mathématiques qui nécessitent des décimales ou des fractions, utiliser les variables de type **`float`**, mais rester conscient de leurs inconvénients : large taille de mémoire, vitesse d'exécution des calculs plus lente.
- Utiliser un opérateur de conversion de type, par exemple **`int(myFloat)`**, pour convertir une variable d'un type en un autre type « à la volée ».

## 14.3.7. Types de données

### 14.3.7.1. int

```
int ledPin = 13;
```

Déclare une variable de type **`int`** (pour *integer*, entier en anglais). Les variables de type int sont le type de base pour le stockage de nombres, et ces variables stockent une valeur sur 2 octets. Elles peuvent donc stocker des valeurs allant de –32768 à 32767. Quand les variables dépassent la valeur maximale de leur capacité, elles « débordent » et reviennent à leur valeur minimale, et ceci fonctionne dans les 2 sens.

**Exemple**

```
int x
   x = -32768;  // x prend la valeur -32768
   x = x-1;  // x vaut maintenant 32767, car déborde dans le sens négatif

   x = 32767;  // x prend la valeur 32767
   x = x+1;  // x vaut maintenant -32768, car déborde dans le sens positif
```

### 14.3.7.2. byte

```
byte b = 128;
```

Déclare une variable de type octet (8 bits) qui stocke un nombre entier non signé, soit une valeur de 0 à 255.

### 14.3.7.3. float

```
float ledPin = 1.117;
```

Déclare des variables de type « virgule-flottante », c'est-à-dire des nombres à virgule. Les nombres à virgule ainsi stockés peuvent prendre des valeurs aussi élevées que 3.4028235E+38 et aussi basse que –3.4028235E+38. Ils sont stockés sur 4 octets (32 bits) de mémoire.

Les variables **`float`** ont seulement 6 à 7 chiffres de précision. Ceci concerne le nombre total de chiffres, pas seulement le nombre à droite de la virgule.

#### 14.3.7.4. void

Le mot-clé **`void`** est utilisé uniquement pour les déclarations de fonctions. Il indique au compilateur que l'on s'attend à ce que la fonction ne retourne aucune donnée.

## 14.3.8. Entrées/Sorties numériques

### 14.3.8.1. pinMode()

Configure la pin spécifiée pour qu'elle se comporte soit en entrée, soit en sortie.

**Syntaxe**

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

**Paramètres**

- **`pin`** : le numéro de la pin de la carte Arduino dont le mode de fonctionnement (entrée ou sortie) doit être défini.
- **`mode`** : soit **`INPUT`** (entrée en anglais) ou **`OUTPUT`** (sortie)

### 14.3.8.2. digitalWrite()

Met un niveau logique **`HIGH`** ou **`LOW`** sur une pin numérique. Si la pin a été configurée en SORTIE avec l'instruction **`pinMode()`**, sa tension est mise à la valeur correspondante : 5 V pour le niveau HAUT, 0 V pour le niveau BAS.

**Syntaxe**

```
digitalWrite(pin,valeur)
```

**Paramètres**

- **`pin`** : le numéro de la pin de la carte Arduino
- **`valeur`** : **`HIGH`** ou **`LOW`** (ou bien 1 ou 0)

**Exemple**

```
int ledPin = 13;                  // LED connectée à la pin numérique 13

void setup()
{
  pinMode(ledPin, OUTPUT);        // met la pin numérique en sortie
}

void loop()
{
  digitalWrite(ledPin, HIGH);     // allume la LED
  delay(1000);                    // attend une seconde
  digitalWrite(ledPin, LOW);      // éteint la LED
  delay(1000);                    // attend une seconde
}
```

### 14.3.8.3. digitalRead()

Lit l'état (= le niveau logique) d'une pin précise en entrée numérique, et renvoie la valeur **`HIGH`** ou **`LOW`**.

### Syntaxe

```
digitalRead(pin)
```

### Paramètre

- pin : le numéro de la pin numérique que vous voulez lire (int).

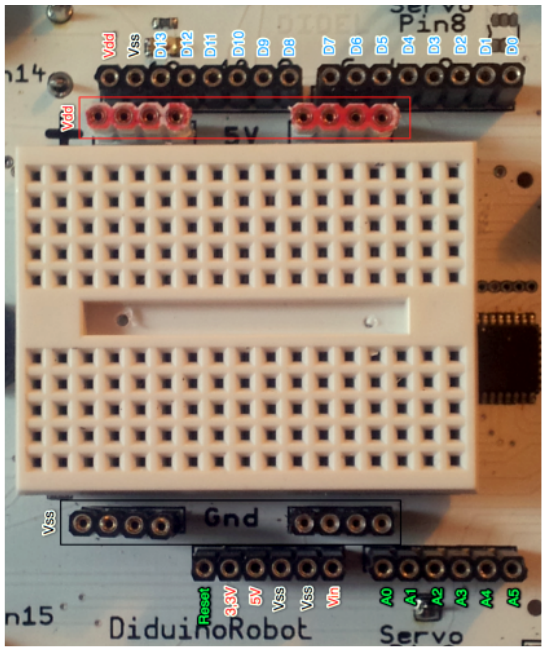
### Exemple

```
int ledPin = 13; // LED connectée à la pin n°13
int inPin = 7;   // un bouton poussoir connecté à la pin 7
                 // avec une résistance de pulldown
int val = 0;     // variable pour mémoriser la valeur lue

void setup()
{
  pinMode(ledPin, OUTPUT);     // configure la pin 13 en SORTIE
  pinMode(inPin, INPUT);       // configure la pin 7 en ENTREE
  digitalWrite(inPin, HIGH);   // écrit HIGH sur la pin en entrée
}

void loop()
{
  val = digitalRead(inPin);    // lit l'état de la pin en entrée
                               // et met le résultat dans la variable
  digitalWrite(ledPin, val);    // met la LED dans l'état du BP
                                // (càd allumée si appuyé et inversement)
}
```

Dans ce programme, la pin 13 reflète fidèlement l'état de la pin 7 qui est une entrée numérique.

## 14.4. Testez vos réflexes !

Avant de faire rouler le robot, amusons-nous un peu avec un montage simple, histoire de se familiariser avec le bloc d'expérimentation, les résistances et le moniteur série.




Dans un bloc d'expérimentation, les rangées de 5 trous sont connectées entre elles par un ressort.

Sur la carte Diduino, les 2 groupes de connecteurs rouges sont reliés au +5V, Les 2 groupes à 4 pins au bas sont à la référence 0V (noté souvent Gnd pour « ground », « masse », « retour de courant »)

Dans le haut de la carte, on a les 16 connexions vers le processeur, comme sur toutes les cartes Arduino. Il ne faut pas utiliser les deux pins de chaque extrémité. Il nous reste 12 entrées ou sorties, numérotées de 2 à 13.

La pin13 est reliée à une Led sur la carte.

Sur le bloc d'expérimentation, on peut brancher des poussoirs ou interrupteurs dont l'état enclenché/déclenché sera lu par le processeur sous forme d'une tension 0V/5V et permettra de prendre des décisions.

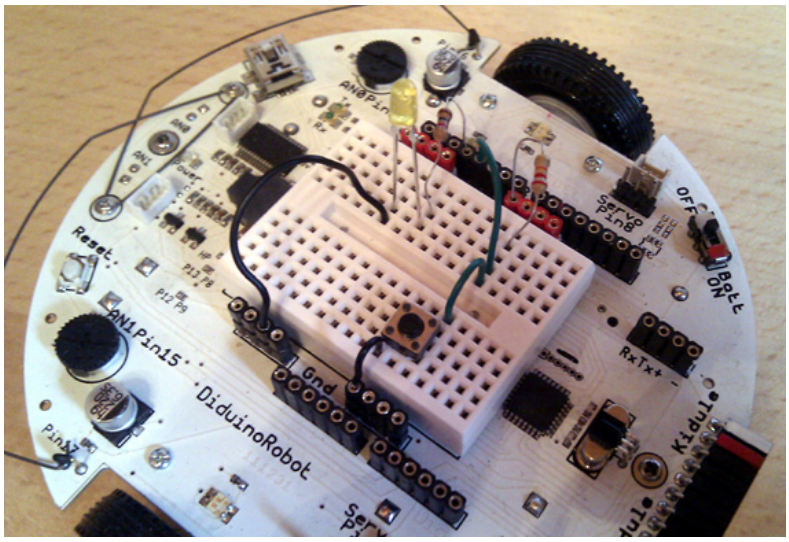
On va aussi brancher des Leds qui seront alimentées par le processeur : si la sortie est au

+5V, le courant va allumer la Led.

La résistance limite le courant. 1 kΩ donne naturellement plus de luminosité que 10 kΩ. Ne pas mettre de résistance est dangereux : la diode sature, le transistor dans le processeur chauffe et finit par se détruire – à éviter comme toutes les connexions directes entre une pin du processeur et le +5V ou la masse. Ou entre le + et le –, qui va détruire le circuit de sortie USB dans le PC.

Débranchez le câble USB pendant que vous câblez.

**Première étape :** câblez une Led sur la pin D12 à un et un poussoir sur la pin D11, selon la photo ci-dessous :




La résistance sur la pin 12 est de 1 kΩ (brun-noir-rouge-doré), celle sur la pin 11 de 22 kΩ (rouge-rouge-orange-doré). Vous trouverez sur le site web compagnon un outil pour reconnaître les résistances d'après les couleurs des bandes.

Le tige la plus longue de la Led (le +) doit se trouver du côté de la résistance.

**Seconde étape :** téléchargez le programme ci-dessous.

reflexes

```
#define Led 12
#define Poussoir 11

int cnt=0;
int alea;

void setup()
{
  pinMode(Led, OUTPUT);
  pinMode(Poussoir, INPUT);
  Serial.begin(9600);
  Serial.println("Pressez quand la Led s'allume");
}

void loop()
{
  alea = random(1000, 5000);
  delay(alea);                    // on attend un temps aléatoire 1-5 sec
  digitalWrite(Led,HIGH);         // allume la led
  cnt = 0;                        // le compteur démarre
  while (digitalRead(Poussoir) == HIGH)
  {
    delay(10);
    cnt++ ;
  }
```

```
  digitalWrite (Led,LOW);         // éteint la led
  Serial.print(cnt);
  Serial.println(" centiemes");
}
```

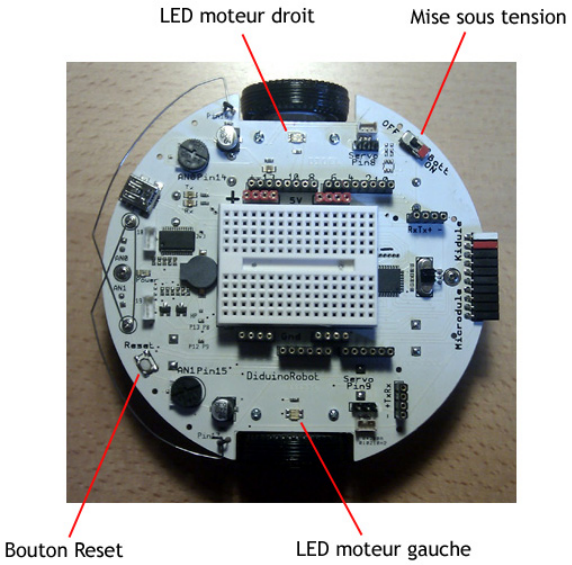
**Déroulement du programme**

1. La Led s'allume.
2. Un compteur de centièmes de secondes se met en route et ne s'arrêtera que lorsque l'on aura appuyé sur le bouton poussoir.
3. Quand le bouton poussoir est pressé, la Led s'éteint.
4. Le temps de réaction sera affiché sur le « serial monitor ».
5. On retourne au point 1.

# 14.5. Contrôler les moteurs des roues

## Remarques préliminaires importantes

1. Pour éviter que votre robot bouge lors de vos essais, mettez-le hors tension ! Il sera alimenté par le câble USB.
2. Il est recommandé de faire un reset hardware (bouton sur le Diduino) avant chaque téléversement.
3. Pour éviter d'endommager le robot, faites vos essais sous tension sur le sol.

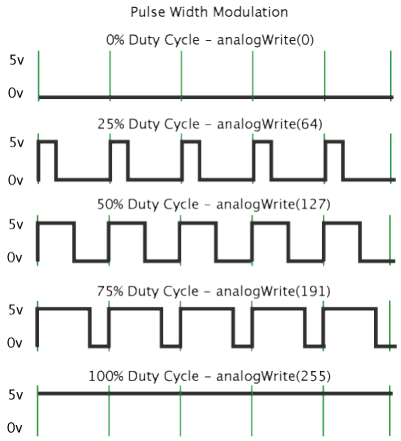



Le moteur droit utilise les pattes 4 et 5, tandis que le moteur gauche utilise les pattes 6 et 7.

| Patte 4 | Action | Patte 5 | Vitesse min | Vitesse max |
|---|---|---|---|---|
| 0 | Avance | PWM | 255 | 0 |
| 1 | Recule | PWM | 0 | 255 |

| Patte 7 | Action | Patte 6 | Vitesse min | Vitesse max |
|---|---|---|---|---|
| 0 | Avance | PWM | 0 | 255 |
| 1 | Recule | PWM | 255 | 0 |

PWM signifie **Pulse Width Modulation**. Il est très souvent nécessaire de faire varier la puissance transmise à une charge. Par exemple, l'intensité d'une lampe doit varier ou la vitesse d'un moteur doit être réglée. La première idée qui vient à l'esprit est de faire varier la tension ou le courant dans la charge. Mais il faut pour cela des circuits électronique complexes. Il est généralement beaucoup plus simple d'alterner des instants où la puissance maximale est transmise à la charge avec des moments où aucune puissance n'est transmise. La technique la plus utilisée est la **Modulation de Largeur d'Impulsion (MLI)** ou **Pulse Width Modulation (PWM)** en anglais.




Pour envoyer des signaux PWM sur le Diduino-Robot, on utilise l'instruction :
`analogWrite(patte, pwm_val);`
La valeur `pwm_val` est comprise entre 0 (qui correspond à 0%) et 255 (qui correspond à 100%).

Quand les moteurs ne sont pas alimentés (interrupteur sur OFF), ce sont des LEDs qui indiquent leur état (vert : avance, rouge : recule).

Analysez le programme suivant sans le charger sur le robot. Que fait-il, pensez-vous ?

moteurs

```
// Moteur droit
int M1 = 5;
int E1 = 4;
// Moteur gauche
int E2 = 7;
int M2 = 6;

byte pwm_val = 0;

void setup()
{
    pinMode(M1, OUTPUT);
    pinMode(M2, OUTPUT);
    pinMode(E1, OUTPUT);
    pinMode(E2, OUTPUT);
}

void loop()
{
    // Moteur droit en mode avance (LED verte s'allume)
    digitalWrite(M1, HIGH);
    analogWrite(E1, 0); // 0: puissance max (en mode avance)
    // Moteur gauche en mode avance (LED verte s'allume)
    digitalWrite(M2, HIGH);
    analogWrite(E2, 0);  // 0 : puissance max (en mode avance)

    delay(200);

    // Moteur droit en mode avance (LED verte s'allume)
```

```
    digitalWrite(M1, HIGH);
    analogWrite(E1, 255); // 255: puissance min (en mode avance)
    // Moteur gauche en mode avance (LED verte s'allume)
    digitalWrite(M2, HIGH);
    analogWrite(E2, 255);  // 255 : puissance min (en mode avance)

    delay(1000);
}
```

Maintenant que vous savez faire tourner les moteurs dans les deux sens, vous êtes capable de faire avancer votre robot.

### Exercice 14.1

Écrivez un programme qui fait tourner les deux moteurs en même temps : d'abord en avant pendant 0.5 seconde, puis s'arrêter 1 seconde, puis en arrière pendant 0.5 seconde, puis s'arrêter 1 seconde, etc..

### Exercice 14.2

Écrivez un programme qui fera tourner un moteur dans le sens des aiguilles d'une montre et l'autre moteur dans le sens inverse. Que se passe-t-il ?

## 14.6. Déplacement du robot

Démarrer les moteurs à pleine vitesse n'est pas très bon pour le mécanisme. Il est préférable d'accélérer et de décélérer progressivement. Le programme ci-dessous montre comment.

ARDUINO

acceleration

```
int M1 = 7;
int E1 = 6;
int E2 = 5;
int M2 = 4;

byte pwm_val = 0;

void setup(){

    pinMode(M1, OUTPUT);
    pinMode(M2, OUTPUT);
    pinMode(E1, OUTPUT);
    pinMode(E2, OUTPUT);

    // Acceleration
    // Moteur gauche en mode avance (LED verte s'allume)
    for (pwm_val=0; pwm_val<255; pwm_val++) {
      // Moteur en mode avance (LED verte s'allume)
      digitalWrite(M1, LOW);
      analogWrite(E1, pwm_val);
      delay(10);
    };

    // Deceleration
    // Moteur gauche en mode avance (LED verte s'allume)
    for (pwm_val=255; pwm_val>0; pwm_val--) {
      // Moteur en mode avance (LED verte s'allume)
      digitalWrite(M1, LOW);
      analogWrite(E1, pwm_val);
      delay(10);
    }
}
```

```
void loop(){
}
```

### 14.6.1. Écrire des fonctions pour les déplacements

Une fonction est définie par **void** (pour le moment) puis le nom de la fonction suivi de (). Le code exécuté lorsque l'on appelle la fonction se trouve entre les accolades : `{ code }`.

Pour appeler la fonction, on place tout simplement le nom de la fonction suivi de (); à l'endroit où l'on veut qu'elle soit exécutée.

Voici un exemple avec les fonctions **avance**, **tournerDroite** (tourner à droite d'environ 90°), **tournerGauche** (tourner à gauche d'environ 90°) et recule. À noter qu'il n'est pas possible de faire tourner le robot d'exactement 90° : il vous faudra ajuster le temps pour votre robot pour les fonctions **tournerGauche** et **tournerDroite**.

deplacements

```
int M1 = 7;
int E1 = 6;
int E2 = 5;
int M2 = 4;

byte pwm_val = 0;

void tourneDroite() {
    // Moteur gauche en mode avance (LED verte s'allume)
    pwm_val = 255;  // Vitesse max
    digitalWrite(M1, LOW);
    analogWrite(E1, pwm_val);

    // Moteur droite en mode recule (LED rouge s'allume)
    pwm_val = 0;  // Vitesse max
    digitalWrite(M2, HIGH);
    analogWrite(E2, pwm_val);

    delay(1000);

    // Moteur gauche à l'arrêt (LED verte s'éteint)
    pwm_val = 0;
    digitalWrite(M1, LOW);
    analogWrite(E1, pwm_val);

    // Moteur droite à l'arrêt (LED rouge s'éteint)
    pwm_val = 0;
    digitalWrite(M2, LOW);
    analogWrite(E2, pwm_val);
}

void tourneGauche() {
    // Moteur gauche en mode recule (LED rouge s'allume)
    pwm_val = 0;  // Vitesse max
    digitalWrite(M1, HIGH);
    analogWrite(E1, pwm_val);

    // Moteur droite en mode avance (LED verte s'allume)
    pwm_val = 255;  // Vitesse max
    digitalWrite(M2, LOW);
    analogWrite(E2, pwm_val);

    delay(1000);

    // Moteur gauche à l'arrêt (LED rouge s'éteint)
    pwm_val = 0;
    digitalWrite(M1, LOW);
    analogWrite(E1, pwm_val);
```

```
    // Moteur droite à l'arrêt (LED verte s'éteint)
    pwm_val = 0;
    digitalWrite(M2, LOW);
    analogWrite(E2, pwm_val);
}

void avance() {
    // Moteur gauche en mode recule (LED verte s'allume)
    pwm_val = 255;  // Vitesse max
    digitalWrite(M1, LOW);
    analogWrite(E1, pwm_val);

    // Moteur droite en mode recule (LED verte s'allume)
    pwm_val = 255;  // Vitesse max
    digitalWrite(M2, LOW);
    analogWrite(E2, pwm_val);

    delay(1000);

    // Moteur gauche à l'arrêt (LED verte s'éteint)
    pwm_val = 0;
    digitalWrite(M1, LOW);
    analogWrite(E1, pwm_val);

    // Moteur droite à l'arrêt (LED verte s'éteint)
    pwm_val = 0;
    digitalWrite(M2, LOW);
    analogWrite(E2, pwm_val);
}

void recule() {
    // Moteur gauche en mode avance (LED rouge s'allume)
    pwm_val = 0;  // Vitesse max
    digitalWrite(M1, HIGH);
    analogWrite(E1, pwm_val);

    // Moteur droite en mode avance (LED rouge s'allume)
    pwm_val = 0;  // Vitesse max
    digitalWrite(M2, HIGH);
    analogWrite(E2, pwm_val);

    delay(1000);

    // Moteur gauche à l'arrêt (LED rouge s'éteint)
    pwm_val = 0;
    digitalWrite(M1, LOW);
    analogWrite(E1, pwm_val);

    // Moteur droite à l'arrêt (LED rouge s'éteint)
    pwm_val = 0;
    digitalWrite(M2, LOW);
    analogWrite(E2, pwm_val);
}

void setup()
{
    pinMode(M1, OUTPUT);
    pinMode(M2, OUTPUT);
    pinMode(E1, OUTPUT);
    pinMode(E2, OUTPUT);
}

void loop()
{
```

```
  avance();
  tourneDroite();
  recule();
  tourneGauche();
}
```

### Exercice 14.3

Écrivez un fonction unique à deux paramètres **`bouge(int md, int mg)`** qui permettra de faire tous les mouvements possibles (avancer et reculer à la vitesse voulue, prendre un virage plus ou moins serré à gauche ou à droite, pivoter sur place). Les paramètres **`md`** et **`mg`** seront les vitesses signées des moteurs gauches et droits, allant de –255 à +255.

### Exercice 14.4

**a.** Écrivez un programme qui fera parcourir au robot un chemin en forme de carré.
**b.** Écrivez un programme qui fera parcourir au robot un chemin en forme de 8.
Pour cela, utilisez votre fonction **`bouge`** de l'exercice 14.3.

## 14.7. Détecter les obstacles avec les moustaches

Le robot est équipé de moustaches à l'avant pour détecter les obstacles.

Les moustaches sont reliées au pattes 16 (moustache droite) et 17 (moustache gauche). Elles fonctionnent comme des interrupteurs.

Quand les moustaches ne sont pas en contact, la valeur que l'on peut lire sur analogRead(16/17); sera 1. Quand il y a contact, la valeur que l'on peut lire suranalogRead(16/17); sera 0.

moustacheG

```
// Utilisation des moustaches

void setup()
{
    pinMode(17, INPUT);
    Serial.begin(9600);
}

int moustacheG;

void loop()
{
  moustacheG = digitalRead(17);
  Serial.print("Moustache gauche: ");
  Serial.print(moustacheG);
  Serial.print("\n");
  delay(200);
}
```

Téléversez ce programme sur votre robot, puis observez sur le moniteur série comment évolue la valeur **`moustacheG`**.

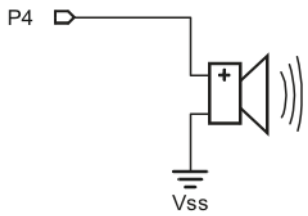
### Exercice 14.5

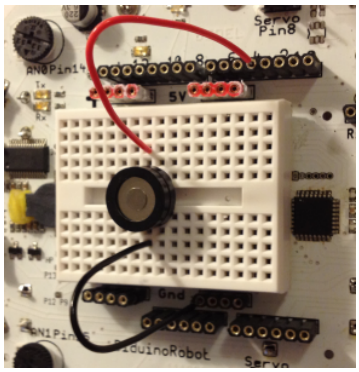
Modifiez le code pour que l'information pour la moustache droite soient aussi affichée.

On a déjà vu qu'une LED soudée sur le robot est branchée sur la pin 13. Modifiez le code pour que cette LED s'allume à chaque fois qu'une moustache est en contact.

## Le piezo

Nous allons maintenant utiliser un piezo, sorte de petit haut-parleur que nous ferons sonner à chaque fois que les moustaches entreront en contact. Le piezo doit être monté dans le bon sens : le – sur Vss et le + sur D4.



Le piezo émet du son lorsque l'on le fait vibrer, c'est-à-dire lorsque l'on alterne de très courtes durées où le piezo est alimenté et où il ne l'est pas.

Essayez ce programme:

piezo

```
// Piezo
#define Piezo 4

void setup() {
  pinMode(Piezo, OUTPUT);
}
void loop() {
  digitalWrite(Piezo, HIGH);
  delayMicroseconds(300);
  digitalWrite(Piezo, LOW);
  delayMicroseconds(300);
}
```

Que se passe-t-il ? Changez maintenant la valeur du délai: 2000, 3000, 4000, 5000. Que se passe-t-il quand le valeur augmente ?

Notez au passage la ligne `#define Piezo 4;`. Cela permet d'associer à un label une valeur que l'on peut utiliser dans le code. A chaque fois que `Piezo` est rencontré dans le code, il est remplacé par la valeur 4, patte où est connecté le piezo. Si on change le Piezo de place, il suffit de changer la ligne `#define Piezo 4;` et tout le code sera automatiquement à jour.

Il est évidemment possible de faire de la « musique » avec le piezo :

musique

```
#define Piezo 4

// Fréquence des notes
#define  c_     3830    // 261 Hz
#define  d_     3400    // 294 Hz
#define  e_     3038    // 329 Hz
#define  f_     2864    // 349 Hz
#define  g_     2550    // 392 Hz
#define  a_     2272    // 440 Hz
#define  b_     2028    // 493 Hz
#define  C_     1912    // 523 Hz

// Longueur des notes
#define longueurNote 16
```

```
#define pause        16

int i;

void joueNote(int note, int duree) {
   for (i=0; i<duree*longueurNote; i++) {
    digitalWrite(Piezo, HIGH);
    delayMicroseconds(note);
    digitalWrite(Piezo, LOW);
    delayMicroseconds(note);
  }

  for (i=0; i<pause; i++) {
    delayMicroseconds(0);
  }
}

void setup() {
  pinMode(Piezo, OUTPUT);
}

void loop() {
  joueNote(g_,1);
  joueNote(g_,1);
  joueNote(g_,1);
  joueNote(a_,1);
  joueNote(b_,2);
  joueNote(a_,2);
  joueNote(g_,1);
  joueNote(b_,1);
  joueNote(a_,1);
  joueNote(a_,1);
  joueNote(g_,3);
}
```

## Exercice 14.6

Le piezo est relié à la pin D4. Nous avons vu ci-dessus comment faire sonner le piezo.

En utilisant l'instruction **`if ... else`**, modifiez votre code et faites sonner le piezo quand les moustaches sont en contact. Vous pourrez jouer un son différent pour la moustache gauche et la moustache droite.

## Exercice 14.7 : Naviguer avec les moustaches

Complétez le programme ci-dessous. Le but est que votre robot puisse se déplacer dans un environnement avec des obstacles en réagissant en cas de contact avec les moustaches.

Reprenez votre fonction **`bouge`** de l'exercice 14.4.


naviguer

```
#define patteMoustacheG 17
#define patteMoustacheD 16

void setup()
{
    pinMode(patteMoustacheD, INPUT);
    pinMode(patteMoustacheG, INPUT);
    Serial.begin(9600);
}

void loop() {
  // Les deux moustaches sont en contact
```

```
  if (analogRead(patteMoustacheD)==0 && analogRead(patteMoustacheG)==0) {
    // action ?
    ...
  // La moustache gauche est en contact
  } else if (analogRead(patteMoustacheG)==0) {
    // action ?
    ...
  // La moustache droite est en contact
  } else if (analogRead(patteMoustacheD)==0) {
    // action ?
    ...
  // Pas de contact
  } else {
    // action ?
    ...
  }
}
```

# 14.8. La course

Nous avons maintenant tout ce qu'il faut pour nous amuser un peu. Nous allons faire une course d'obstacles avec nos robots.

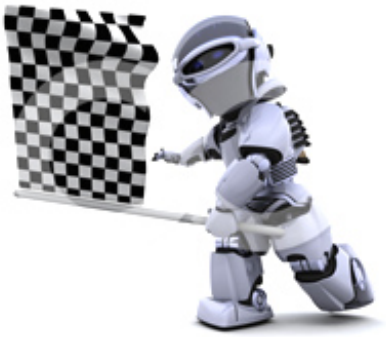
Durant les qualifications, les robots seront chronométrés individuellement.

Ensuite, deux robots s'affronteront simultanément sur le même parcours (ce qui veut dire qu'il pourra y avoir des collisions entre robots). Les couples seront formés selon les résultats des qualifications (le premier avec le dernier, le deuxième avec l'avant-dernier, etc.). Le perdant sera à chaque fois éliminé. Il y aura des ¼ de finales, des ½ finales et la grande finale.

Il vous faudra donc programmer votre robot pour que :

- il aille toujours dans le bon sens de la course ;
- il sache quoi faire quand il touchera un obstacle ou un autre robot ;
- il ne reste pas coincé dans un obstacle.

Vous aurez connaissance du parcours deux heures avant la course et vous pourrez faire des essais.
Il sera possible d'utiliser un programme pour les qualifications et un autre pour les duels.



Que le meilleur gagne !