Cours complet pour apprendre à programmer un Arduino

Par Kossigan Roland ASSILEVI - Mamadou COULIBALY - Maurin DONNEAUD

Date de publication : 19 février 2019

Le projet Arduino est issu d'une équipe d'enseignants et d'étudiants de l'école de Design d'Interaction d'Ivrea (1) (Italie). Ils rencontraient un problème majeur à cette période (avant 2003 - 2004) : les outils nécessaires à la création de projets d'interactivité étaient complexes et onéreux (entre 80 et 100 euros). Ces coûts souvent trop élevés rendaient difficiles le développement par les étudiants de nombreux projets et cela ralentissait la mise en œuvre concrète de leur apprentissage.

Commentez

I - Historique du projet Arduino

Jusqu'alors, les outils de prototypage étaient principalement dédiés à l'ingénierie, la robotique et aux domaines techniques. Ils sont puissants mais leurs processus de développement sont longs et ils sont difficiles à apprendre et à utiliser pour les artistes, les designers d'interactions et, plus généralement, pour les débutants.

Leur préoccupation se concentra alors sur la réalisation d'un matériel moins cher et plus facile à utiliser. Ils souhaitaient créer un environnement proche de Processing, ce langage de programmation développé dès 2001 par Casey Reas (2) et Ben Fry, deux anciens étudiants de John Maeda au M.I.T., lui-même initiateur du projet DBN (3) .

En 2003, Hernando Barragan, pour sa thèse de fin d'études, avait entrepris le développement d'une carte électronique dénommée Wiring, accompagnée d'un environnement de programmation libre et ouvert. Pour ce travail, Hernando Barragan réutilisait les sources du projet Processing. Basée sur un langage de programmation facile d'accès et adaptée aux développements de projets de designers, la carte Wiring a donc inspiré le projet Arduino (2005).

Comme pour Wiring, l'objectif était d'arriver à un dispositif simple à utiliser, dont les coûts seraient peu élevés, les codes et les plans « libres » (c'est-à-dire dont les sources sont ouvertes et peuvent être modifiées, améliorées, distribuées par les utilisateurs eux-mêmes), et enfin, « multiplateforme » (indépendant du système d'exploitation utilisé).

Conçu par une équipe de professeurs et d'étudiants (David Mellis, Tom Igoe, Gianluca Martino, David Cuartielles, Massimo Banzi ainsi que Nicholas Zambetti), l'environnement Arduino est particulièrement adapté à la production artistique ainsi qu'au développement de conceptions qui peuvent trouver leurs réalisations dans la production industrielle.

Le nom Arduino trouve son origine dans le nom du bar dans lequel l'équipe avait l'habitude de se retrouver. Arduino est aussi le nom d'un roi italien, personnage historique de la ville « Arduin d'Ivrée », ou encore un prénom italien masculin qui signifie « l'ami fort ».

II - Exemples d'usages

Plate-forme logicielle et matérielle de création d'objets numériques, Arduino permet de programmer des circuits électroniques qui interagissent avec le milieu qui les entoure. Connectés notamment à des capteurs sonores, thermiques et de mouvement, ces circuits électroniques peu coûteux, dénommés micro-contrôleurs, peuvent en retour générer des images, actionner un bras articulé, envoyer des messages sur Internet, etc. Des dizaines de milliers d'artistes, de designers, d'ingénieurs, de chercheurs, d'enseignants et même d'entreprises l'utilisent pour réaliser des projets incroyables dans de multiples domaines :

- **prototypage rapide de projets innovants utilisant l'électronique**, Arduino facilitant l'expérimentation en amont de la phase d'industrialisation ;
- **production artisanale d'objets numériques et de machines-outils à faible** coût dans la perspective d'une culture d'appropriation technologique favorisant le bricolage et la débrouille ;
- **captation et analyse de données scientifiques** (environnement, énergie, etc.) à des fins éducatives, de recherche ou d'appropriation citoyenne ;
- **spectacle vivant**, grâce aux nombreuses fonctions d'interaction offertes par Arduino, il est possible de créer des performances de VJing, d'utiliser le mouvement des danseurs pour générer en temps réel des effets sonores et visuels dans un spectacle ;
- **installations d'arts numériques**, Arduino permettant de réaliser des œuvres d'art interagissant de manière autonome avec le public ;
- **Mode et design textile**, plusieurs stylistes et designers investissant ce domaine créatif en exploitant les possibilités offertes par l'intégration de l'électronique notamment dans des vêtements (e-textile) ;
- **projets pédagogiques** à destination d'étudiants, de professionnels ou du grand public selon les porteurs de ces initiatives : écoles supérieures, centres de formation spécialisée ou des Media Labs.

La suite de ce chapitre va vous présenter quelques exemples de travaux réalisés avec l'aide d'Arduino dans différents contextes.

II-A - Production artisanale d'objets numériques et de machines-outils

Arduino facilite le prototypage rapide de projets innovants ainsi que la production artisanale d'objets numériques et de machines-outils à faible coût notamment dans la perspective d'une culture d'appropriation technologique et citoyenne favorisant le bricolage et la débrouille.

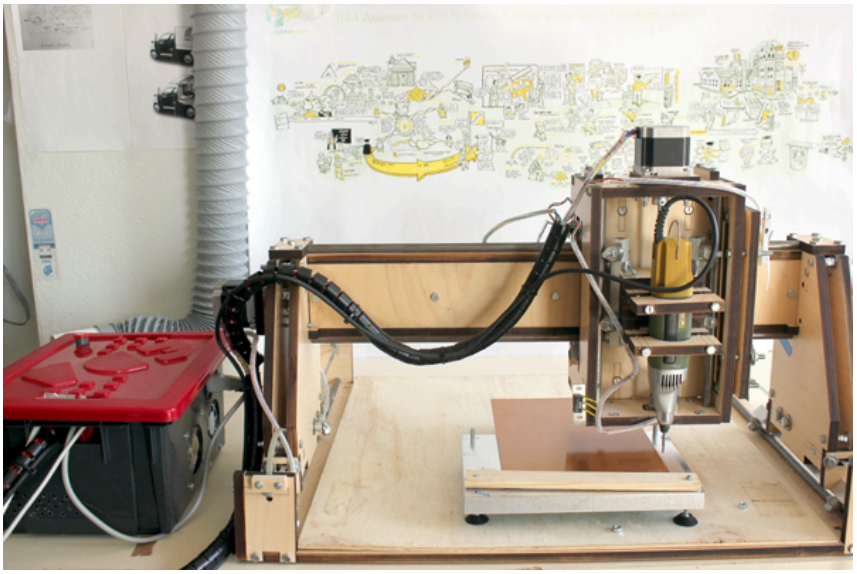
II-A-1 - Frida V

Fixé sur un vélo, Frida V prend la forme d'un boîtier contenant un ensemble de capteurs qui analysent et compilent les informations récoltées durant le parcours du cycliste : pollution atmosphérique (capteurs de gaz), Wi-Fi ouvert dans la ville (routeur Wi-Fi), position géographique (GPS), documentation audiovisuelle des parcours (caméra et microphone). L'ensemble est assemblé dans une coque résistante et portable adaptable sur tout type de vélos.

Les données recueillies sont archivées sur un serveur et représentées graphiquement sur une carte, la *MeTaMap*. Ces informations peuvent être annotées par les participants en direct grâce aux contrôleurs de la Frida. La Frida est constituée notamment d'un routeur Linux embarqué et d'un système de micro-Arduinos. Les logiciels sont développés en libre et adaptables sur de nombreuses autres plates-formes matérielles de type ordiphones.

Production de 16 boîtiers prototypes réalisés en 2009 par Luka Frelih, Žiga Kranjec, Jure Lozić, Rok Hlavaty, Miha Turšič, Igor Križanovskij (Slovénie) dans le cadre d'une résidence de création à la Galerie Ars longa (Paris, France), en partenariat avec Ljubljana Laboratory for Digital Media and Cultur (Ljubljana, Slovénie) à l'occasion du festival Futur en Seine (France, 2009).

II-A-2 - Fraiseuse numérique

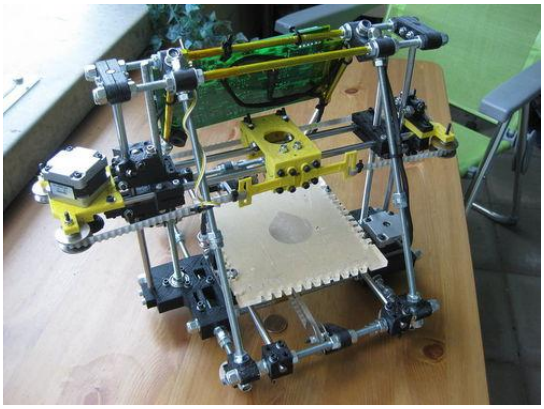


Fraiseuse numérique réalisée lors du Bootcamp « Make ta machine » (Nantes, 2011)

Arduino peut agir comme un automate de commande pour piloter une machine à commande numérique fabriquée maison, qui coûte bien moins cher qu'un modèle acheté, comme cette fraiseuse, inspirée du modèle canadien « Oomloot », adapté aux mesures métriques et réalisée durant le Boot Camp « Make ta machine » à Nantes au printemps 2011.

Cette machine peut réaliser des gravures, des découpes et des usinages dans divers matériaux (bois, aluminium, circuits électroniques, etc.).

II-A-3 - Imprimante 3D Reprap



RepRap est une imprimante de bureau 3D libre capable de fabriquer des objets en plastique. Comme la plupart des pièces de la RepRap sont faites de cette matière et que la RepRap peut les créer, cette machine est considérée comme « autoréplicable ».

RepRap s'inscrit dans le mouvement du matériel libre et de la fabrication de machines-outils à faible coût. L'absence de brevet et la mise à disposition gratuite des plans offrent la possibilité à quiconque de la construire avec du temps et un minimum de matériel. Cela signifie également que si vous avez une RepRap, vous pouvez réaliser beaucoup de choses utiles, voire créer une autre RepRap pour un ami.

Projet en constante amélioration, la RepRap existe en plusieurs modèles. L'électronique de pilotage de la RepRap est basée sur Arduino.

Initié en 2005 par Adrian Bowyer (Royaume-Uni) : **http://reprap.org**.

II-A-4 - Projet Solar Sinter

La Solar Sinter dans le désert égyptien (Crédit photographique : Markus Kayser)

Adoptant une approche originale où les rayons du soleil remplacent la technologie laser et le sable divers produits chimiques utilisés par certaines imprimantes 3D, Solar Sinter est une machine à énergie solaire permettant de fabriquer des objets en verre.

Dans les déserts du monde, deux éléments dominent : le soleil et le sable. Le premier offre une source d'énergie potentielle immense, le second un approvisionnement presque illimité en silice sous forme de sable. Le sable de silice, lorsqu'il est chauffé à son point de fusion puis refroidi, se solidifie en formant du verre opaque.

L'Arduino sert à suivre le soleil dans son déplacement et à positionner les rayons concentrés par des lentilles optiques sur une surface de sable en fonction de la forme de l'objet désiré.

Réalisé en 2011 par Markus Kayser (Allemagne) : **www.markuskayser.com/work/solarsinter/**.

II-B - Productions artistiques avec Arduino

Grâce aux nombreuses fonctions offertes par Arduino, il est possible de créer des spectacles d'art vivants, générant des effets visuels et sonores originaux ainsi que des œuvres d'art numériques interagissant de manière autonome avec le public.

II-B-1 - Performance Solénoïdes Midi

Solénoïdes midi lors des rencontres multimédias LabtoLab (Nantes, 2011)

Solénoïdes *midi* est une performance de percutions numériques qui joue sur la résonance des matériaux : un Arduino équipé d'un module *midi* actionne des petits électro-aimants qui percutent les surfaces sur lesquelles ils sont installés au rythme d'une partition réalisée sur un séquenceur musical.

Selon la nature du support où les électro-aimants sont placés (arbres, sonnettes de vélos ou structures métalliques d'un bâtiment, tel un balcon), les sons produits sont transformés et amplifiés.

Réalisé en 2011 par Antoine Bellanger (France) : **http://gratuitmusic.com/solenoides.html**.

II-B-2 - La Chorale à roulettes

La Chorale à roulette présentée à la Galerie Pierre-François Ouellette art contemporain lors de l'exposition collective Encodeurs (Montréal, 2007).

La Chorale à roulettes est une installation sonore utilisant Arduino qui met en scène des téléphones à cadran, symboles de la communication moderne. Chaque appareil a conservé sa sonnerie originale. Additionnées et composées, celles-ci génèrent un répertoire étendu où des timbres joyeux et entraînants peuvent répondre à des sons d'une infinie tristesse.

La Chorale à roulettes propose une gamme de timbres allant de la sonnerie mélodieuse au clic à peine audible d'un ronron métallique assourdi.

Réalisé en 2007 par Darsha Hewitt et Alexandre Quessy (Canada-Québec) : **http://alexandre.quessy.net/?q=rotarianchoir**.

II-B-3 - Être aux anges, variation pour sept angelinos

Être aux anges présenté à l'occasion du festival panOramas (Bordeaux, 2010)

Alliant éléments rétro et composants électroniques, cette œuvre numérique se nourrit des réseaux sociaux d'Internet pour s'animer poétiquement au passage des anges : des danseuses, prisonnières de bouteilles de verre, entament leur ballet accompagné d'une petite mélodie lorsque le flux de messages publics de Twitter contient les mots ange, être ange ou étrange.

Un *angelino* est plus précisément composé d'une danseuse contenue dans une bouteille musicale de type Lucas Bols, utilisée comme un ready-made et connectée à Internet via un Arduino. Chaque *angelino* réagit à un mot particulier. Lors du ballet, le spectateur ou l'internaute peut agir à distance sur l'*angelino* de son choix en publiant un message sur Twitter comportant l'un des mots « choisis ».

Réalisé en 2010 par Albertine Meunier (France) : **www.albertinemeunier.net/etre-aux-anges/**.

II-B-4 - Bountou Kheweul

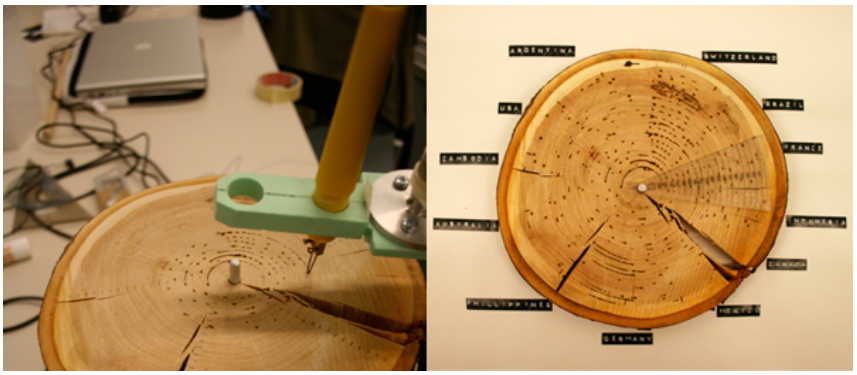
Bountou Kheweul au 8e Forum des arts numériques de Martinique (Fort de France, 2011)

Bountou Kheweul, qui signifie « Porte de toutes les grâces » en wolof, est le résultat d'une résidence d'artiste de Armin Kane en Martinique dans le cadre d'un projet d'essaimage de pratiques artistiques numériques en Afrique de l'Ouest et dans les Caraïbes (projet Rose des vents numériques).

Mixant illustrations au fusain et technologies interactives, Bountou Kheweul interroge les racines des cultures martiniquaises et sénégalaises en repérant des similitudes observées et mises en scène sous forme de séquences vidéos et sonores.

Réalisé en 2011 par Armin Kane (Sénégal) avec l'appui technologique de Roland Kossigan Assilevi (Togo) : **http://ker-thiossane.org/spip.php?article74**.

II-B-5 - Ecologia



Œuvre minimaliste, Ecologia illustre graphiquement l'état de la déforestation dans 12 pays du monde de 1990 à 2007. À chaque année correspond une suite de lignes tracées sur les sillons de la coupe d'un arbre, leur fréquence et leur longueur variant en fonction de l'intensité et de la surface en km^2 du processus de déforestation.

Le tracé des lignes a été obtenu en utilisant les capacités de calcul de l'ordinateur connecté via Arduino à une machine de pyrogravure spécialement conçue pour graver des objets de forme circulaire.

Réalisé en 2010 par Lucien Langton (Suisse).

II-B-6 - Back

Back au festival Afropixel (Dakar, 2010)

Réalisé dans le cadre d'un projet d'essaimage de pratiques artistiques numériques en Afrique de l'Ouest et dans les Caraïbes (projet Rose des vents numériques), Back est une installation numérique sur le thème de la reconstruction et de la capacité de résilience de l'art. C'est l'interaction du public qui, à la fois brise une poterie et la reconstruit, en remontant dans le temps.

Back repose sur l'utilisation d'Arduino connecté à des capteurs de présence et de pression, un ordinateur ainsi qu'un écran : le spectateur en s'approchant de l'installation enclenche une séquence vidéo où l'on voit une poterie traditionnelle africaine en suspension chuter et se briser. En soufflant sur une feuille placée dans un vase, le sens de déroulement de la vidéo s'inverse et la poterie brisée se reconstruit.

Réalisé en 2010 par Charles Seck (Sénégal), Samba Tounkara (Sénégal), Said Afifi (Maroc) et Roland Kossigan Assilevi (Togo) pour la partie technologique : **www.ker-thiossane.org/spip.php?article26**.

II-B-7 - Zahra-Zoujaj (Fleur-Verre)

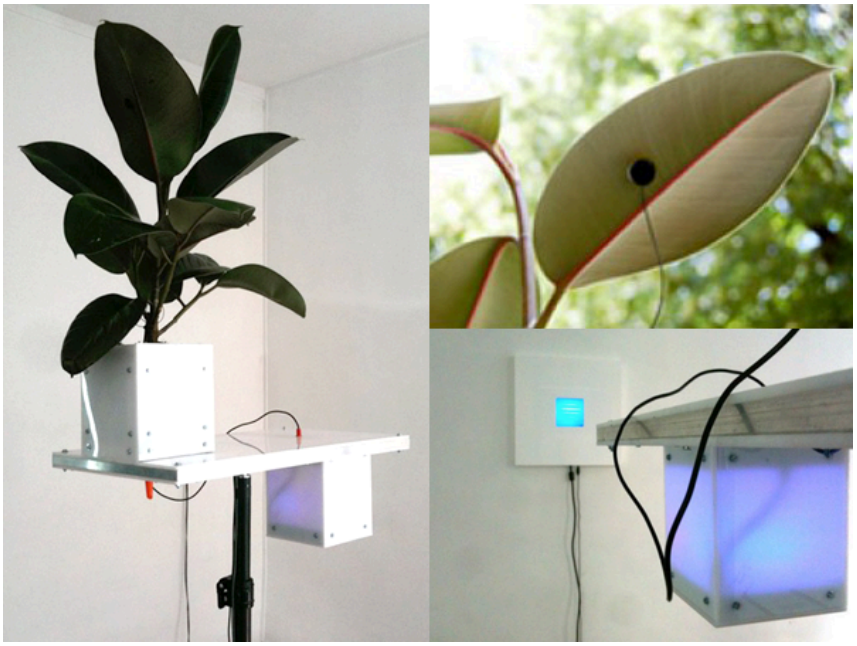
Zahra-Zoujaj exposée au Al-Riwaq Art Space à Doha (Qatar)

Après avoir été invité à pénétrer dans la Hojra, une structure de forme octogonale se rétrécissant vers le haut à l'instar des pyramides, le spectateur découvre un ensemble de 77 fleurs en verre soufflé suspendues au plafond, « tête en bas ». Chaque fleur est unique et symbolise l'une des 77 règles de convivialité tirées du Coran. Leurs cœurs, à l'intérieur desquels palpite une douce lumière rouge dont l'intensité baisse et augmente en des rythmes différents, sont formés de boules rouges identiques. Expérience contemplative, ce n'est qu'après quelques minutes d'attention que la quinzaine de spectateurs que peut contenir la Hojra, parvient à distinguer les véritables nuances colorées de chacune des fleurs disposées en coupole autour de sept cercles concentriques.

Utilisant notamment Arduino, cette œuvre monumentale générative a été réalisée en 2010-2011 par l'artiste Younès Rahmoun (Maroc) avec l'appui technique du Centre International d'Art Verrier de Meisenthal et de Maurin Donneaud (France) pour la partie électronique :

- présentation de l'installation interactive sur le site Internet de Younès Rahmoun : **www.younesrahmoun.com/FR/Travaux/Travaux.html** ;
- code source et plan électronique de l'installation : **http://maurin.donneaud.free.fr/?-Zahra-zoujaj**.

II-B-8 - Diffractions transmutatoires



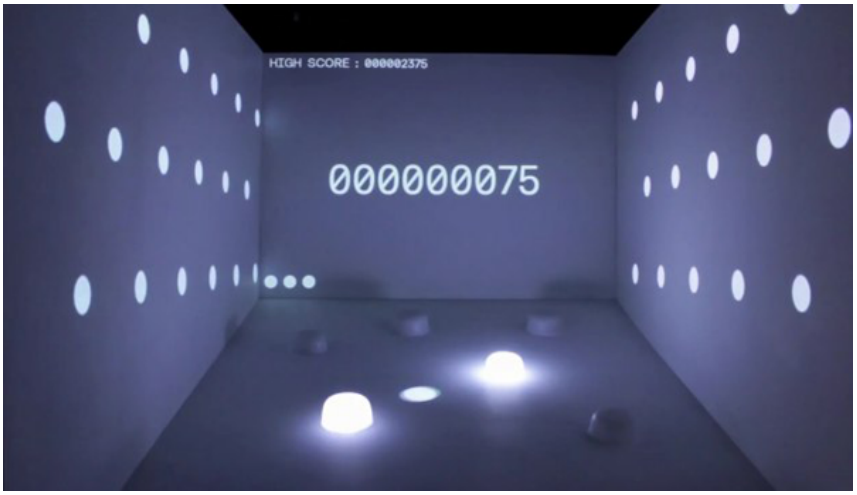
L'installation Diffractions Transmutatoires exposée à la Galerie L'Espace d'en Bas dans le cadre du Festival Mal au pixel (Paris, juin 2011) / Crédits photographiques : Galerie Espace d'en Bas

À la frontière de la science et de la para-science, Diffractions Transmutatoires, révèle dans un de ses modules, la vie secrète d'une plante et ses interactions subtiles avec le monde environnant. Cette installation, fruit des travaux menés depuis 2006 par les artistes Horia Cosmin Samoïla et Marie-Christine Driesen, requestionne la notion de « perception primaire » développée par Cleve Baxter dans les années 1960.

Arduino a servi à prototyper ce module-végétal, en interfaçant le dispositif plus ancien qui repose sur un détecteur de mensonges. La version finalisée de cette installation utilise à présent des circuits électroniques spécifiques et davantage performants pour analyser les signaux biologiques provenant de la plante.

Réalisé en 2010 par Horia Cosmin Samoïla (Roumanie) et Marie-Christine Driesen (France) : **www.ghostlab.org/Diffractions-Transmutatoires-2** (vidéo de l'installation exposée à l'Abbaye de Noirlac, Noirlac, 2011).

II-B-9 - BeamBall



BeamBall au Comptoir suisse 2011.

Au croisement de l'installation et du jeu vidéo, cette création numérique simule le fonctionnement d'un flipper géant à l'aide d'une balle lumineuse pilotée par le spectateur.

L'animation visuelle de la balle et des champignons du flipper est projetée sur les structures en relief de l'espace de jeu. Arduino sert d'interface entre les gros boutons poussoirs de la console utilisée par le public et la partie informatique de l'installation.

Réalisé par Sigmasix en partenariat avec Lumens 8 :

- vidéo de l'installation : **http://vimeo.com/30084908** ;
- site de Sigmasix : **www.sigma.ch**.

II-B-10 - À mots cousus

Carte interactive sur tissu de À mots cousus présentée à l'exposition Renum (Nantes, 2011)

À mots cousus est le résultat d'une résidence d'artistes dans lequel des femmes retraitées ont été invitées à témoigner de la relation aux lieux dans lesquels elles ont travaillé ainsi qu'à participer à la création d'une œuvre collective numérique.

Quelles ont été leurs activités avant de se retrouver à la retraite ? Comment imaginent-elles le travail dans une ville utopique ? Quels sont les obstacles pour y arriver ? À partir d'interviews, les artistes et les participantes ont réalisé ensemble deux cartographies sonores urbaines, l'une physique, sur une matière textile, l'autre virtuelle, en ligne.

La carte sur tissu possède des points réactifs, cousus à des fils conducteurs reliés à des câbles électroniques et à une carte Arduino. Le spectateur entend le témoignage associé à un lieu de son choix en piquant simplement à l'aide d'une aiguille à coudre la surface de la carte.

Création de Nicolas Malevé, Wendy Van Wynsberghe, Peter Westenberg et An Mertens du collectif Constant (Belgique) avec la participation de femmes de la ville de Nantes (France) : **www.constantvzw.org/site/ReNUM,1295.html** et **www.amotscousus.constantvzw.org**.

II-C - Mode et design textile

Arduino suscite l'intérêt de nombreux chercheurs et professionnels de la mode et du design textile en offrant la possibilité d'insérer des circuits électroniques miniaturisés dans les vêtements.

II-C-1 - Robes interactives Walking City et Living Pod

Crédit photographique : Dominique Lafond

Ying Gao est designer de mode et professeure à l'Université du Québec à Montréal. À travers sa recherche, elle remet en question la notion de vêtement tel qu'on le connaît en alliant le design urbain, l'architecture et le multimédia. En incorporant des éléments technologiques à ses créations, elle explore la construction du vêtement et le déploiement de sa forme.

L'Arduino est ici utilisé pour modifier la structure des vêtements en contrôlant moteurs, valves pneumatiques et éléments lumineux qui se trouvent cachés sous la surface du tissu, mais aussi pour analyser l'environnement immédiat qui les entoure.

Réalisé en 2006 et 2008 par Ying Gao (Canada-Québec) en collaboration avec Simon Laroche (Canada-Québec) pour le design interactif : **www.yinggao.ca**.

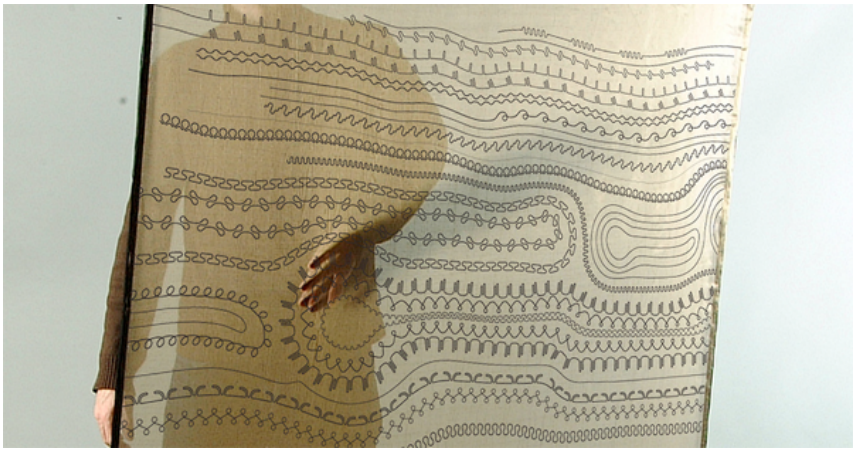
II-C-2 - Climate Dress

Climate Dress au Salon Tech Textile (Francfort, 2011)

Climate Dress est une robe faite de broderie conductrice, de plus d'une centaine de minuscules diodes lumineuses, d'un détecteur de gaz carbonique (CO_2) et de cartes Arduino Lylipad très fines, conçues pour s'intégrer facilement dans un tissu.

Cette robe climatique s'illumine de différents motifs et change de rythme de pulsation selon le niveau de concentration de $C0_2$ dans l'air, permettant ainsi à la personne qui la porte comme à son entourage d'être davantage conscients des problèmes environnementaux.Textile XY

II-C-3 - Textile XY



À la rencontre du tissage et de l'électronique, le textile XY est un textile capteur qui permet de localiser la position d'un contact à sa surface. Né de la volonté de repenser l'ergonomie des interfaces numériques, cette recherche initiée en 2005 a donné naissance à plusieurs développements de textiles tactiles.

Les premiers prototypes ont été réalisés en utilisant Arduino et ont permis d'imaginer des usages dans de nombreux domaines, tels que la mode, les jeux pour enfants, etc.

Développé depuis 2005 par Maurin Donneaud et Vincent Roudaut (France) : **http://xyinteraction.free.fr/wiki/pmwiki.php/FR/TextileXY/**.

II-D - Projets pédagogiques avec Arduino

Plusieurs projets pédagogiques à destination d'étudiants, de professionnels ou du grand public reposent sur l'utilisation d'Arduino.

II-D-1 - Valise pédagogique création interactive

Présentation de la valise pédagogique à Kër Thiossane, Villa des arts et du multimédia.

Réalisé dans le cadre d'un projet d'essaimage de pratiques artistiques numériques en Afrique de l'Ouest et dans les Caraïbes (projet Rose des vents numériques), la Valise pédagogique création interactive est un ensemble matériel, logiciel et documentaire pour l'apprentissage des technologies d'Interaction Temps Réel dans la création contemporaine, tous champs artistiques confondus (arts plastiques, danse, théâtre, musique, architecture, design, etc.).

Équipée de deux cartes Arduino, la valise peut servir aussi bien de plate-forme d'apprentissage dans le cadre d'un atelier de découverte de l'interaction en art que d'outil de création pour artiste en permettant d'inventer, de simuler, puis de réaliser des milliers de dispositifs interactifs différents.

Réalisé en 2010-2011 par Jean-Noël Montagné (artiste plasticien), Jérôme Abel (artiste développeur) et les électroniciens africains de ENDA Ecopole en partenariat avec Kër Thiossane (Sénégal) et le CRAS (France) : **www.ker-thiossane.org/spip.php?article6**.

II-D-2 - MIAM - Mallette interactive artistique multimédia

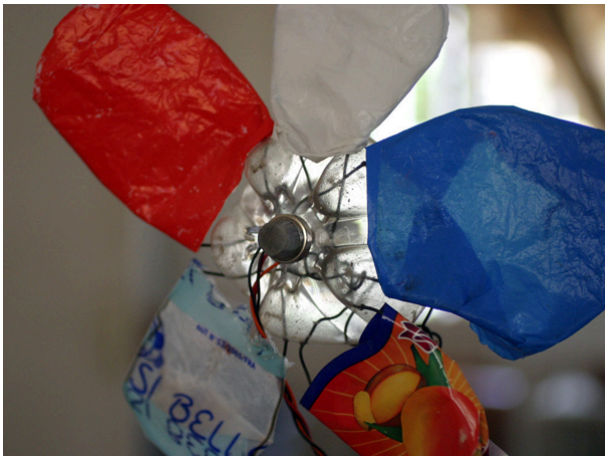
Illustration des périphériques et dispositifs contenus dans la Mallette Interactive Artistique Multimédia

Destinée à être tant un outil pédagogique qu'un instrument/système à vocation artistique, la Mallette Interactive Artistique Multimédia (MIAM) est constituée d'un ordinateur équipé des périphériques les plus couramment utilisés dans les dispositifs et instruments interactifs (capteurs divers, webcam, joystick, wiimote, carte Arduino, etc.). Elle offre aux enseignants et formateurs de nombreuses ressources numériques et multimédias « prêtes à l'emploi » et destinées à un large public pour un décryptage, et une approche de l'histoire de l'art numérique et interactif de façon didactique et illustrée.

Le projet de la Mallette Interactive Artistique Multimédia a reçu le soutien financier du Ministère de la Culture et de la Communication (France).

Réalisé en 2010-2011 par les associations Labomedia et Ping (France) en collaboration avec la Fabrique du Libre : **http://lamiam.fr**.

II-D-3 - Atelier Jardin électronique et environnement



Gros plan sur une des fleurs interactives réalisées par des jeunes de la Commune de Grand Yoff à Dakar (Sénégal, 2011).

Avec l'objectif de réfléchir sur la qualité environnementale des villes dans lesquelles ils vivent, une trentaine de jeunes sénégalais ont participé à Dakar à cet atelier de création numérique animé par l'artiste espagnol Victor Viña. Du 19 au 30 septembre 2011, ces jeunes de 12 à 18 ans ont été invités à construire sous une forme ludique et créative des capteurs électroniques pour mesurer des éléments de l'environnement, tels que le rayonnement solaire, la température, la pollution, l'humidité ou le niveau sonore de la ville.

Utilisant notamment Arduino pour la partie électronique, ces artefacts sensibles au milieu qui les entoure ont été fabriqués à partir d'objets obsolètes et de matériaux de récupération comme des jouets inutilisés, du carton, du ruban adhésif, du papier journal, des bouteilles en plastique et d'autres matières recyclables.

Le dernier jour de l'atelier, les enfants ont analysé la qualité de leur environnement en interaction avec les passants et en réalisant un jardin éphémère constitué de plantes électroniques installées dans un espace public du quartier où ils vivent.

Cet atelier et l'exposition qui a suivi ont été organisés par la structure sénégalaise Trias culture avec l'appui de Music et culture Vs Prod, la mairie de Grand-Yoff, les ASC (Doolé, Grand-Yoff, Gangui, Rakadiou et Yaakar) , ENDA Jeunesse et Action, Jaboot, le Centre Talibou Dabo en partenariat avec l'Ambassade d'Espagne au Sénégal et de Cultura Dakar (Espagne).

Les images des réalisations présentées dans ce chapitre sont la propriété de leurs auteurs respectifs.

III - Installation

L'installation de l'interface de programmation Arduino est relativement simple et possible sur les plates-formes Windows, Mac OS X et Linux. L'environnement de programmation Arduino est écrit en Java et l'interface est inspirée de Processing, un compilateur *avr-gcc* (pour le processeur du micro-contrôleur) ainsi que d'autres logiciels libres. Puisque Arduino s'appuie sur Java, il est nécessaire que la machine virtuelle Java soit installée sur votre système d'exploitation (ou mise à jour). Elle l'est normalement sur Mac OS X, mais il est possible que sous Windows ou Linux, il soit demandé de l'installer au cours du processus d'installation d'Arduino.

Une documentation en ligne existe pour vous aider dans l'installation du logiciel Arduino à l'adresse suivante :

- **https://www.arduino.cc/en/Guide/HomePage**.

Cette aide supplémentaire peut vous être utile pour adapter la méthode d'installation selon votre système d'exploitation et le type de micro-contrôleur Arduino que vous possédez. La suite de ce chapitre reprend les étapes principales et générales pour l'installation de cette application.

Pour télécharger le fichier d'installation, il vous suffit de vous rendre sur la page :

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

Pour sélectionner une version, cliquez sur le nom qui correspond à votre système d'exploitation et sauvegardez sur votre ordinateur le fichier correspondant. Il est à noter que dans le cadre de ce manuel, la version anglaise sera utilisée puisqu'elle contient habituellement des mises à jour plus récentes que la version française.

Pour la suite de ce chapitre, dans les noms de fichiers et dossiers, l'appendice *« xxxx »* annexé à « Arduino » fait référence à la version de l'application Arduino utilisée.

III-A - Sous Windows

Il vous faut d'abord télécharger le fichier *arduino-xxxx.zip*. Une fois le téléchargement terminé, vous décompressez l'archive et déplacez la de préférence dans le dossier « C:\Program Files\ ».

En utilisant un câble USB, branchez votre carte Arduino à votre ordinateur. Lorsque vous connectez une carte d'interfaçage Arduino pour la première fois, Windows devrait démarrer le processus d'installation du pilote (*driver*).

Quand il est demandé si Windows doit se connecter à Windows *Update* pour rechercher le logiciel, choisir « NON, pas maintenant » puis « Suivant ».

Sélectionnez l'installation depuis une liste ou un emplacement spécifique et cliquez sur « Suivant ».

Assurez-vous que « Rechercher le meilleur pilote dans ces emplacements » est coché ; décochez « Rechercher dans les médias amovibles » ; cochez « Inclure cet emplacement dans la recherche » et ouvrir le sous-répertoire « /drivers/FTDI_USB » dans le répertoire arduino-xxxx téléchargé précédemment (au besoin, la dernière version des pilotes peut-être trouvée sur le site FTDI). Cliquez sur « Suivant ».

L'assistant va ensuite rechercher le pilote et vous dire que le « USB Serial Converter » a été trouvé. Cliquez sur « Terminer ».

L'assistant de « Nouveau Matériel » devrait apparaître à nouveau : répétez la même procédure. À cette étape de l'installation, l'assistant devrait indiquer « USB Serial Port a été trouvé ». Cliquez sur « Terminer ».

Rendez-vous ensuite dans le dossier « C:\Program Files\arduino-xxxx » et exécutez le fichier *arduino.exe* pour lancer l'application.

III-B - Sous MAC OS X

Il vous faut d'abord télécharger le fichier *arduino-xxxx.dmg*. Une fois le téléchargement terminé, l'image « disque » devrait se monter automatiquement. Si ce n'est pas le cas, double-cliquez dessus. Placez l'application Arduino de préférence dans le dossier « Applications » de votre ordinateur.

Dans le cas où votre carte Arduino n'est pas de type *Uno* ou *Mega 2560* (donc pour tous les modèles plus anciens), il faudra également installer les pilotes FTDI (au besoin, la dernière version des pilotes peut-être trouvée sur le **FTDI**). Pour ce faire, il faut simplement cliquer sur l'icône ***FTDIUSBSerialDriver_xxxxxxx.mpkg***. Suite à cette installation, redémarrez votre ordinateur.

Branchez la carte Arduino à votre ordinateur avec le câble USB.

Dans le cas où votre carte Arduino est de type *Uno* ou *Mega 2560*, une fenêtre apparaît indiquant qu'une nouvelle interface réseau a été détectée. Sélectionnez « Préférences réseau » et à l'ouverture cliquez sur « Appliquer ». Même si les cartes *Uno* ou *Mega 2560* sont décrites comme « Non configuré », l'installation devrait aboutir.

Quittez les « Préférences système ».

Vous pouvez maintenant lancer l'application Arduino à partir de votre dossier « Applications » ou encore de votre « dock » si vous y avez installé un alias (icône de raccourci) de l'application Arduino.

III-C - Sous GNU/Linux

La procédure d'installation sous GNU/Linux dépend de la distribution utilisée et l'exécution du programme dépend également de l'installation d'autres pilotes ou logiciels. Pour les utilisateurs de Linux, il est conseillé de visiter la page suivante pour connaître la marche à suivre :

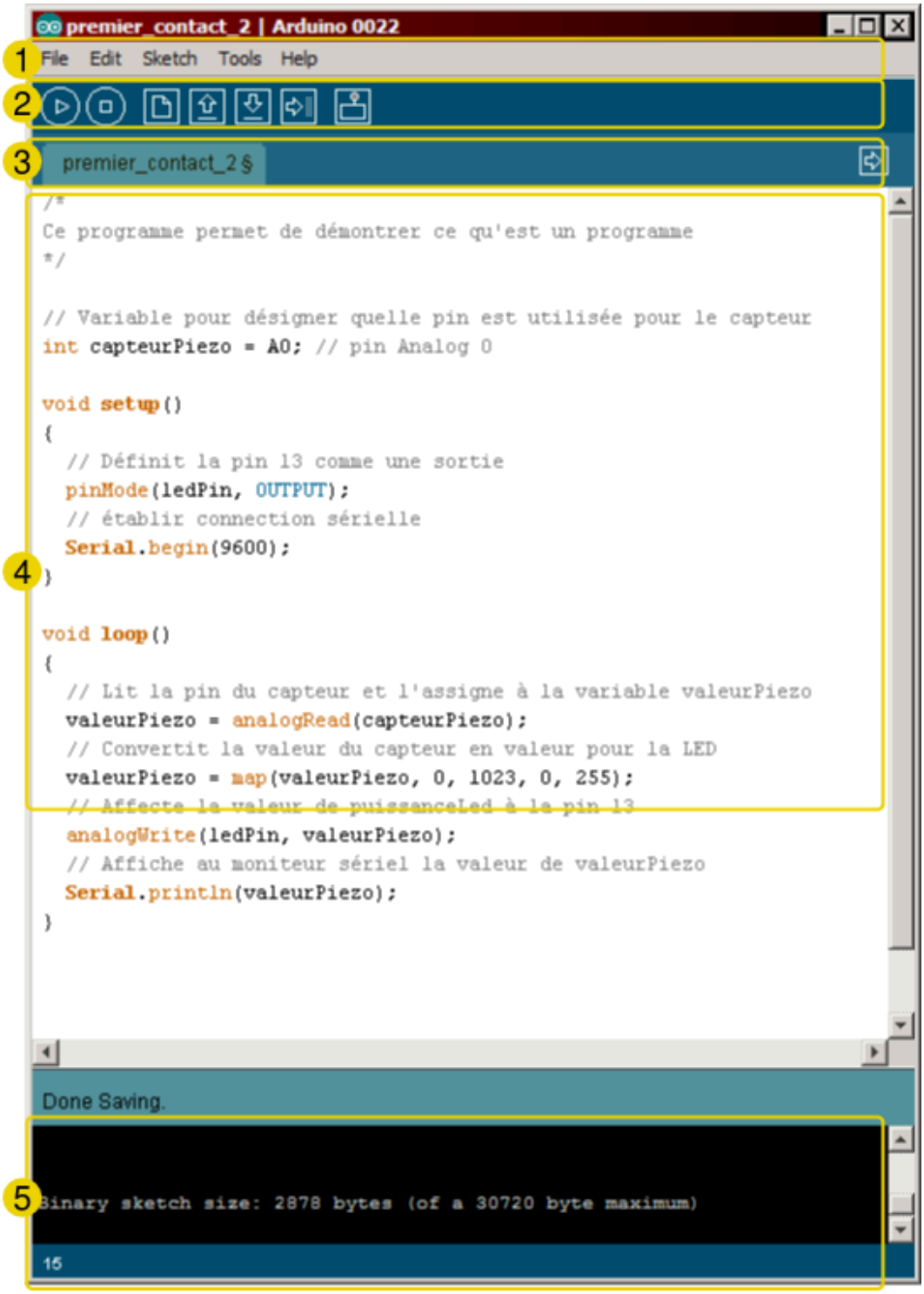
- **http://www.arduino.cc/playground/Learning/Linux**.

IV - Prise en main rapide

Ce chapitre vous présente les principales fonctionnalités de l'interface de l'application Arduino. L'application Arduino vous permet de créer et éditer un programme (appelé *sketch*) qui sera compilé, puis téléversé sur la carte Arduino. Ainsi, lorsque vous apportez des changements sur le code, ils ne seront effectifs qu'une fois le programme téléversé sur la carte.

Il est à noter que ce manuel fait référence à la version en anglais de ce logiciel puisqu'elle comporte habituellement des mises à jour plus récentes que la version en français. Que les non-anglophones se rassurent : le nombre réduit de fonctionnalités et l'utilisation d'icônes rendent l'interface du logiciel simple d'utilisation.

IV-A - Fenêtre générale de l'application Arduino



La fenêtre de l'application Arduino comporte les éléments suivants :

1 un menu ;
2 une barre d'actions ;
3 un ou plusieurs onglets correspondant aux *sketchs* ;
4 une fenêtre de programmation ;
5 une console qui affiche les informations et erreurs de compilation et de téléversement du programme.

IV-B - Éléments du menu

Les différents éléments du menu vous permettent de créer de nouveaux *sketch*s, de les sauvegarder, de gérer les préférences du logiciel et les paramètres de communication avec votre carte Arduino.

IV-B-1 - Dossier de travail

Dans les préférences (*File* > *Preferences*), il vous est possible de spécifier votre dossier de travail. Il s'agit du dossier où seront sauvegardés par défaut vos programmes et les bibliothèques qui pourront y être associées. Lorsqu'un programme est sauvegardé, un dossier portant le nom du programme est créé. Celui-ci contient le fichier du programme portant le nom que vous lui aurez donné, suivi de l'extension *.pde,* ainsi qu'un dossier intitulé *applet* qui contient les différents éléments créés et nécessaires lors du processus de compilation du programme et de téléversement vers la carte.

IV-B-2 - Exemples

Une série d'exemples est disponible sous *File* > *Examples*. Ces exemples peuvent vous aider à découvrir et comprendre les différentes applications et fonctions d'Arduino.

IV-B-3 - Outils de configuration (Tools)

Dans le menu *Tools*, il vous est possible et essentiel de spécifier le type de carte Arduino que vous utiliserez. Sous *Tools* > *Board*, il vous faut spécifier pour quel type de carte vous compilez et téléversez le programme. Le type de carte est généralement inscrit sur la carte elle-même.

Il est également nécessaire lorsqu'on branche une nouvelle carte Arduino ou que l'on change de carte, de spécifier le port sériel virtuel qui sera utilisé pour la communication et le téléversement du programme. Pour ce faire, il faut aller sous *Tools* > *Serial Port* et choisir le port approprié. Sous Windows, il s'agit la plupart du temps du port ayant un numéro supérieur à 3. Sous Mac OS X, il s'agit habituellement du premier élément de la liste. Une bonne technique pour déterminer quel port correspond à votre carte Arduino consiste à débrancher celle-ci, attendre un peu et de prendre note des ports déjà présents. Lorsque vous rebrancherez votre carte Arduino et après un peu d'attente (ou un redémarrage de l'application), vous remarquez le port qui se sera ajouté à la liste. Il s'agit du port sériel virtuel lié à votre carte Arduino.

IV-C - Barre d'actions

Bouton « Verify » (Vérifier) : il permet de compiler votre programme et de vérifier si des erreurs s'y trouvent. Cette procédure prend un certain temps d'exécution et lorsqu'elle est terminée, elle affiche un message de type « Binary sketch size : … » indiquant la taille du *sketch* téléversé.

Bouton « Stop » : arrête le moniteur sériel (certaines plates-formes seulement).

Bouton « New » (Nouveau) : ce bouton permet de créer un nouveau *sketch*.

Bouton « Open » (Ouvrir) : il fait apparaître un menu qui permet d'ouvrir un *sketch* qui




figure dans votre dossier de travail ou des exemples de *sketchs* intégrés au logiciel.
Bouton « Save » (Sauvegarder) : il permet de sauvegarder votre *sketch*.

Bouton « Upload » (Téléverser) : ce bouton permet de compiler et téléverser votre *sketch* sur la carte Arduino.
Bouton « Serial Monitor » (Moniteur sériel) : ce bouton fait apparaître le moniteur sériel (voir ci-bas).
Bouton « Onglet » : il permet de basculer entre les *sketchs*.

IV-D - Moniteur sériel

Le moniteur sériel est utilisé pour afficher l'information qui est envoyée par la carte Arduino vers l'application (habituellement par le câble USB). Il permet aussi d'envoyer de l'information à la carte Arduino. Pour ce faire, il suffit d'écrire du texte dans le champ situé en haut de la fenêtre et d'appuyer sur le bouton « Send ». Bien évidemment, avant de pouvoir faire cela il vous faut coder un programme approprié, prêt à recevoir cette information. Il est également possible de régler le *baudrate* du moniteur sériel, qui précise à quelle vitesse le transfert des données s'effectuera. Il est également possible d'établir une communication sérielle entre la carte Arduino et d'autres périphériques ou logiciels. Nous verrons plus en détails cette fonction dans les projets « Texte brillant » et « Oscilloscope ».

V - Vous avez dit Arduino ?

Arduino est une plate-forme de prototypage d'objets interactifs à usage créatif constituée d'une carte électronique et d'un environnement de programmation.

Sans tout connaître ni tout comprendre de l'électronique, cet environnement matériel et logiciel permet à l'utilisateur de formuler ses projets par l'expérimentation directe et avec l'aide de nombreuses ressources disponibles en ligne.

Pont tendu entre le monde réel et le monde numérique, Arduino permet d'étendre les capacités de relations humain/machine ou environnement/machine.

Arduino est un projet dont les sources sont ouvertes : c'est-à-dire que les plans, les schémas, etc., sont accessibles et libres de droits. De plus, la très importante communauté d'utilisateurs et de concepteurs permet à chacun de trouver les réponses à ses questions et apporte un boulot énorme de documentation du projet.

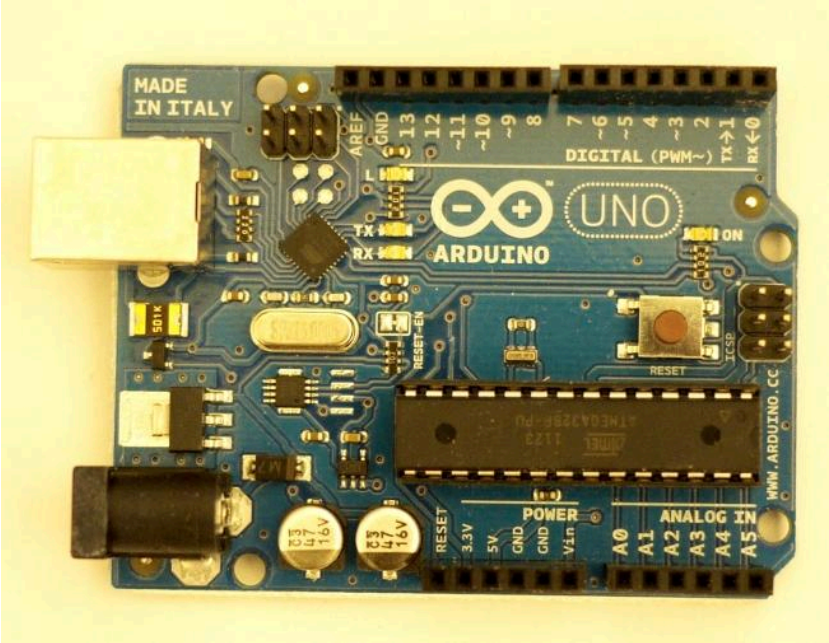
V-A - Matériel

La carte Arduino repose sur un circuit intégré (un miniordinateur appelé également micro-contrôleur) associé à des entrées et sorties qui permettent à l'utilisateur de brancher différents types d'éléments externes :

- côté entrées, des **capteurs** qui collectent des informations sur leur environnement comme la variation de température via une sonde thermique, le mouvement via un détecteur de présence ou un accéléromètre, le contact via un bouton-poussoir, etc. ;
- côté sorties, des **actionneurs** qui agissent sur le monde physique telle une petite lampe qui produit de la lumière, un moteur qui actionne un bras articulé, etc.

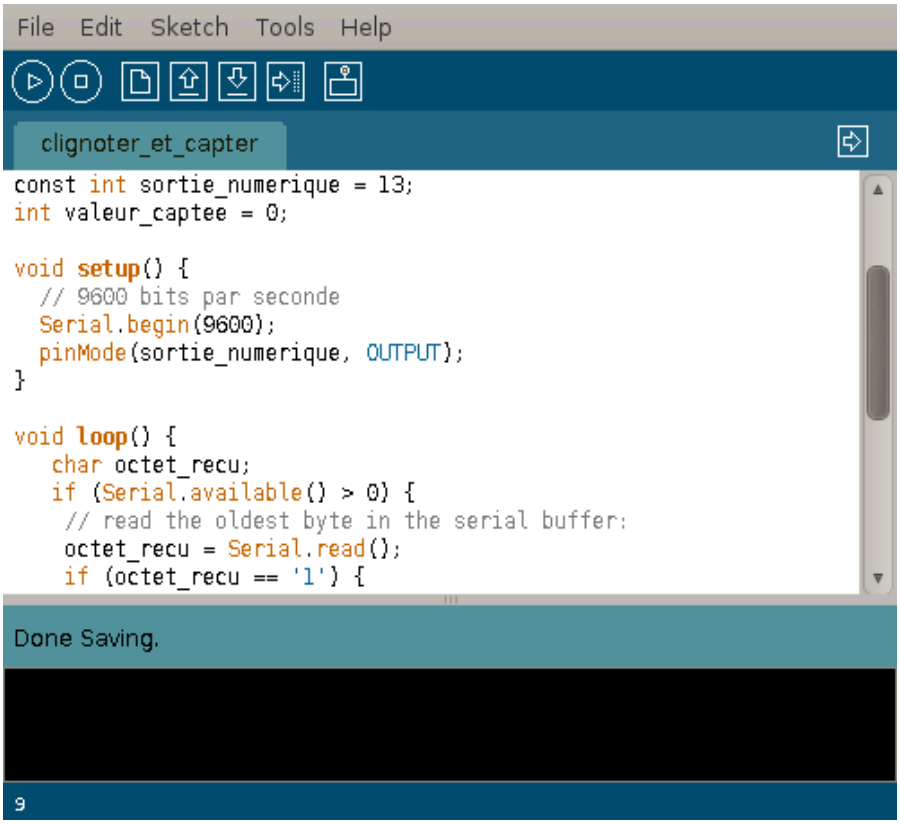
Comme le logiciel Arduino, le circuit électronique de cette plaquette est libre et ses plans sont disponibles sur Internet. On peut donc les étudier et créer des dérivés. Plusieurs constructeurs proposent ainsi différents modèles de circuits électroniques programmables et utilisables avec le logiciel Arduino (4) .

Il existe plusieurs variétés de cartes Arduino. La figure ci-dessous montre, par exemple, la dernière version de la carte Arduino : la « Uno », sortie en 2010. Cette carte électronique peut être autonome et fonctionner sans ordinateur ou servir d'interface avec celui-ci.



V-B - Logiciel

L'environnement de programmation Arduino (EDI en français et IDE en anglais) est une application écrite en Java inspirée du langage Processing (5) . L'IDE permet d'écrire, de modifier un programme et de le convertir en une série d'instructions compréhensibles pour la carte.



V-C - Quelques outils fréquemment utilisés avec Arduino

Lorsque l'Arduino est connecté à un ordinateur, il est capable de communiquer avec diverses applications, notamment :

V-C-1 - Processing

Conçu par des artistes, pour des artistes, Processing (6) est un environnement de création fréquemment utilisé pour générer des œuvres multimédias à partir d'un code informatique sur ordinateur. L'attrait de ce logiciel réside dans sa simplicité d'utilisation et dans la diversité de ses applications : image, son, applications sur Internet et sur téléphones mobiles, conception d'objets électroniques interactifs.

Processing fédère une forte communauté d'utilisateurs professionnels et amateurs : artistes, graphistes, vidéastes, typographes, architectes, web designers et designers en général. Il est également utilisé par des enseignants en arts qui souhaitent familiariser leurs étudiants avec les potentialités artistiques de la programmation, les concepteurs du logiciel l'ayant pensé dès l'origine comme un outil d'apprentissage.

Un exemple de code Processing est utilisé dans le projet **« Oscilloscope »** (voir section **« Projets », chapitre « Oscilloscope »**).

V-C-2 - Pure Data

Pure Data (7) (souvent abrégé *Pd*) est un logiciel de création multimédia interactive couramment utilisé dans les domaines artistique, scientifique et pédagogique. Sa popularité réside dans ses possibilités de programmation en temps réel. Plutôt qu'un langage de programmation textuel, Pure Data propose un environnement de **programmation graphique** dans lequel l'utilisateur est invité à manipuler des icônes représentant des fonctionnalités et à les brancher ensemble.

Un exemple de programme Pure Data est utilisé dans le projet **« Texte brillant »** (voir **section « Projets », chapitre « Texte brillant »**).

V-C-3 - Supercollider

Supercollider (8) est un environnement et un langage de programmation pour la synthèse audio en temps réel et la composition algorithmique. Il est disponible pour toutes les plates-formes et distribué sous licence **GPL**. Sa particularité de permettre la modification de code à la volée en fait un des principaux langages de *live coding*. SuperCollider est un logiciel gratuit. La version la plus récente (3.4) est sortie en juillet 2010.

V-C-4 - S4A (Scratch For Arduino)

Scratch For Arduino (S4A) est une modification de l'application Scratch qui permet de programmer simplement, facilement et intuitivement des instructions à téléverser sur la carte Arduino. Tout comme Arduino et Scratch, S4A est un « logiciel libre » donc téléchargeable gratuitement et qui évolue avec les participations communautaires. S4A comme Scratch sont d'abord conçus à des fins pédagogiques et destinés à des enfants, adolescents, mais aussi à des adultes débutants. Ainsi, chacun peut approcher la programmation informatique pour mieux comprendre notre « société numérique ». S4A peut encore être utilisé comme prototypage rapide d'un projet.

VI - À propos de ce livre

Les valeurs du libre ont inspiré la rédaction et la diffusion de ce manuel d'initiation à Arduino, l'objectif étant à la fois :

- d'offrir à un public professionnel ou amateur francophone, les bases d'utilisation d'Arduino ;

- de valoriser la communauté des développeurs et experts francophones d'Arduino impliqués dans la rédaction et la mise à jour de ce manuel en français ;
- de fédérer plus largement la communauté francophone d'Arduino autour d'un projet commun de documentation (tant au niveau des coauteurs que des commentateurs), sachant que l'ouvrage dans sa forme accessible en ligne (wiki) peut être amélioré et comporter de nouveaux chapitres, notamment à l'occasion de la parution d'une nouvelle version d'Arduino.

VI-A - Un ouvrage collectif

Ce livre sous licence libre est une production originale en français ; plusieurs coauteurs francophones de différents pays ont participé à sa rédaction.

Le cœur de l'ouvrage de plus de 120 pages a été réalisé en cinq jours dans le cadre d'un BookSprint qui s'est tenu à Dakar (Sénégal) du 19 au 23 octobre 2011, grâce à l'initiative et avec le soutien de l'Organisation internationale de la Francophonie (**www.francophonie.org**) en partenariat avec Ker Thiossane (**www.ker-thiossane.org**), le Collectif Yeta (**www.collectifyeta.org**), Main d'œuvres (**www.mainsdœuvres.org**) et Ping (**www.pingbase.net**).

Corédacteurs présents lors du BookSprint :

- Kossigan Roland ASSILEVI (Togo) : formateur numérique (Arduino, PureData…) et personne ressource sur des projets artistiques utilisant les technologies. Il fait partie de l'équipe de Kër Thiossane, Villa pour l'art et le multimédia à Dakar ;
- Mamadou COULIBALY (Mali) : formateur numérique (Arduino, PureData…) et personne ressource sur des projets artistiques utilisant les technologies. Il fait partie de l'équipe du Collectif Yeta qui anime notamment un Media Lab à Bamako ;
- Maurin DONNEAUD (France) : designer d'interaction, programmeur d'objets physiques notamment avec Arduino, intervenant au Centre de Ressources Art Sensitif (Cras) et dans des écoles de design textile (ENSCI Paris, ENSAAMA Paris, CSM London) sur des projets e-textiles ;
- Cédric DOUTRIAUX (France) : artiste multimédia et formateur (Arduino, etc.). Il travaille actuellement avec l'association Nantaise PING, association ressource et pépinière de projets innovants, au développement de machines-outils à commandes numériques pour l'ouverture d'un FabLab. Il a également participé en tant qu'intervenant au projet Rose des vents numériques, initiative d'essaimage de pratiques artistiques utilisant le numérique en collaboration avec Kër Thiossane et le Collectif Yeta ;
- Simon LAROCHE (Canada-Québec) : artiste, enseignant et designer interactif, spécialisé dans les installations audiovisuelles et la robotique. Il travaille également comme concepteur vidéo et robotique en arts vivants, cinéma et mode. Simon Laroche est membre fondateur du Projet EVA ;
- Florian PITTET (Suisse) : designer interactif au sein de la société SIGMASIX créée en collaboration avec Eric Morzier, et enseignant en nouveaux médias à la Haute école d'arts appliqués de Lausanne (Ecal) ;
- Marie-Paul UWASE (Rwanda) : secrétaire du Réseau africain des utilisateurs des logiciels libres (RAL), présidente de l'association rwandaise de promotion des logiciels libres au Rwanda et enseignante universitaire.

Facilitatrice :

- Elisa DE CASTRO GUERRA : présidente de Flossmanuals francophone, facilitatrice de BookSprints et graphiste utilisant les logiciels libres.

Corédacteurs en ligne et contributions externes :

- Mamadou DIAGNE (dit Genova) ;
- Guillaume FOUET ;
- Pascale GUSTIN ;
- Corinne LAURENT ;
- Christian AMBAUD.

VI-B - Un ouvrage vivant

Ce manuel est vivant : il évolue au fur et à mesure des contributions. Pour consulter la dernière version actualisée, nous vous invitons à visiter régulièrement le volet francophone de Flossmanuals sur le site **http://fr.flossmanuals.net** et plus particulièrement sur la page d'accueil du manuel Arduino.

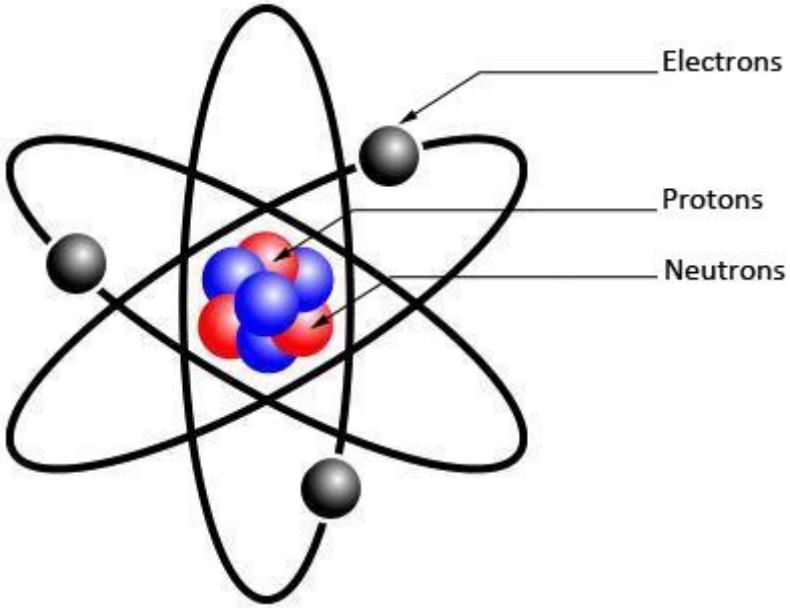
N'hésitez pas à votre tour à améliorer ce manuel en nous faisant part de vos commentaires dans la discussion associée à ce livre, ou, si vous avez des talents de rédacteur et une bonne connaissance d'Arduino, à vous inscrire en tant que contributeur pour proposer la création de nouveaux chapitres ou de nouveaux tutoriels. Vous trouverez en fin d'ouvrage la liste complète des personnes ayant participé jusqu'à ce jour à la corédaction du manuel.

VII - Les bases de l'électronique

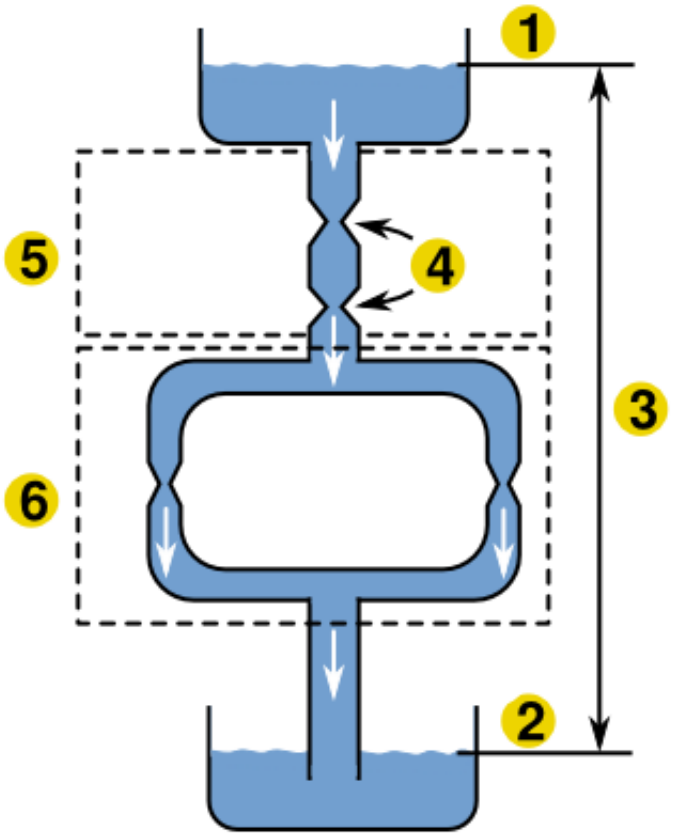
L'électronique est la manipulation de signaux et d'informations électriques afin de les mesurer, de les contrôler ou de les modifier.Des éléments désignés « composants » sont assemblés sous la forme de circuits. Ces assemblages peuvent être réalisés à la main ou par des sociétés industrielles qui intègrent et miniaturisent ces circuits. Par exemple, le processeur de la carte Arduino est un circuit intégré contenant des millions de composants.

VII-A - Notions électriques fondamentales

L'électricité est une forme d'énergie comme les énergies éolienne ou hydraulique. Cette énergie électrique peut se résumer par : mouvements des électrons entre les atomes. Par exemple, en frottant un ballon sur certains vêtements, comme un chandail, des électrons sont échangés entre les atomes du ballon et ceux du chandail. Le ballon se charge ainsi négativement en captant les électrons du vêtement : nous nommons cela « l'électricité statique ». L'électricité devient « dynamique » si l'on décharge graduellement le ballon en le laissant « coller » à un mur où à des cheveux.



Pour bien comprendre les éléments de mesure de base de l'électricité, il est pratique d'utiliser l'analogie d'un système hydraulique. Imaginez ce système composé de deux réservoirs d'eau connectés ensemble via un réseau de tubes.



VII-A-1 - La tension et la différence de potentiel (volts)

Sur notre image, nous observons que les deux bassins sont à des altitudes différentes **(1)** et **(2)**.Ces altitudes correspondent au **potentiel électrique**.

La différence entre les deux altitudes, soit le dénivelé **(3)** correspond à **la tension**. Ce dénivelé va générer une pression à cause de la gravité.

La tension et le potentiel sont exprimés en **volts (notée V ou souvent U)**. La source d'alimentation électrique d'un circuit (une pile, par exemple) est une source de tension.

On mesure toujours une altitude **par rapport à une référence**. En électricité, on place souvent cette référence au **(-)** de l'alimentation (qui correspond ici au point **(2)**). Dans les schémas électroniques, cette référence correspond souvent à la « **masse** ». Lorsqu'on interconnecte deux circuits alimentés différemment, il est indispensable de leur donner la même référence (voir chapitre « **Précautions d'utilisation** »).

VII-A-2 - Le courant (ampères)

Dans notre système, la pression générée par le dénivelé provoque un certain débit d'eau dans le réseau de tuyaux. Le débit correspond au courant. En électronique, le courant est exprimé en **ampères (A ou noté I ou i)**.

VII-A-3 - La résistance (ohms)

Lorsque le tube se rétrécit dans notre exemple **(4)**, une moins grande quantité d'eau peut circuler à la fois. Ce rétrécissement crée ce qu'on appelle une **résistance**. La pression du système (ou la force avec laquelle l'eau circule) n'est pas changée ; c'est plutôt le débit qui change. En électronique, la résistance est exprimée en **ohms (Ω ou noté R)**.

L'équation générale qui lie ces trois unités de mesure est : **U = RI**

Soit le voltage (U) est égal à la résistance (R) multipliée par le courant (I).

VII-A-4 - Circuits, parallèle ou série

Un circuit est un ensemble de composants électriques.Bien que cela semble contre-intuitif à première vue, on dira qu'il est « fermé » lorsqu'il y a continuité dans les connexions qui lient les composants entre eux. Un circuit « ouvert » comporte une discontinuité dans les connexions. Autrement dit, lorsque le circuit est fermé, le courant passe, et lorsqu'il est ouvert, il ne passe pas.

Lorsqu'on désigne un circuit comme étant **en série**, cela signifie que les éléments sont connectés les uns à la suite des autres, sur une même branche **(5)**. Dans ce cas les valeurs de résistance vont s'additionner.

Dans un **circuit en parallèle**, les éléments sont situés chacun sur des branches indépendantes **(6)**. Dans ce cas, les résistances sont situées à altitude égale et donc soumises à la même tension (voltage). Dans ce cas, le courant se partage dans chacune des branches.

VII-A-5 - AC/DC

Ces deux abréviations ne représentent pas seulement un groupe de rock. Un courant électrique DC, parfois noté CC, signifie **« Direct Current »** en anglais soit **« Courant Continu »**.C'est un courant qui ne varie pas dans le temps. Il peut être généré par une pile, une batterie ou un circuit d'alimentation qui redresse un courant alternatif.Le courant DC est le type de courant habituellement utilisé en électronique. Par exemple, votre carte Arduino est alimentée par ce courant.

Le courant AC signifie **« Alternating Current »** ou **« Courant Alternatif »**. Il s'agit d'un courant qui change de direction continuellement. Il peut être périodique, c'est-à-dire que sa fréquence est constante. La forme la plus utilisée est le courant sinusoïdal. Il est caractérisé par sa **fréquence notée f et exprimée en hertz**, qui correspond au nombre d'allers-retours par seconde. Dans certains calculs, il peut arriver que l'on ait besoin de sa pulsation souvent notée oméga minuscule (w) = 2 x PI x f. Le courant électrique utilisé à la maison est AC.

VII-A-6 - Multiples

En électronique, les valeurs sont parfois très petites ou très grandes. Pour simplifier leur notation, on rencontre souvent des préfixes qui expriment des multiples des valeurs.

Multiple	Préfixe	Notation	Nom
10e6 / 1 000 000	méga	M	million
10e3 / 1 000	kilo-	k	millier
10e-1 / 0.1	déci-	d	dixième
10e-2 / 0.01	centi-	c	centième
10e-3 / 0.001	milli-	m	millième
10e-6 / 0.00000	micro-	µ	millionième
10e-9 / 0.000000001	nano-	n	milliardième

Par exemple, une valeur de 0.001 ampères pourrait être écrite de plusieurs manières :

- 1 mA = 1 milliampère = 1.10^{-3} A = 0.001 A

VII-B - Les composants

Les composants sont des éléments de base en électronique qui, une fois assemblés, constitueront un circuit électronique. Chacun de ces éléments a un comportement bien particulier, dépendant de ses caractéristiques et de ses conditions d'utilisation. Pour le choix et le dimensionnement des composants les plus complexes, il est utile de consulter leur fiche technique (« datasheet » en anglais).

Voici une description de quelques-uns de ceux-ci.

VII-B-1 - Résistance

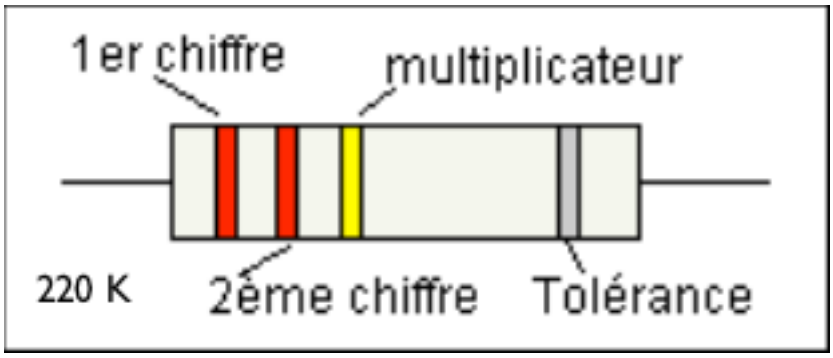
Les résistances sont utilisées dans de nombreux cas, pour réduire une tension (voir plus loin le **pont diviseur de tension**), pour provoquer un courant, ou associées à d'autres composants pour des circuits plus complexes (exemple : filtre RC). Leur valeur est **notée R et exprimée en ohms**.

En **série**, les résistances s'additionnent : **Req = R1 + R2 + R3.**

En **parallèle**, c'est différent : **1 / Req = (1 / R1) + (1 / R2) + (1/R3).**

La formule associée à la résistance est : **U = RI.**

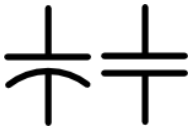
En électronique, la valeur d'une résistance est codée par des anneaux de couleurs :



Dans l'exemple ci-haut, la résistance est de 220 kohms, ou 220 000 oms. Le premier anneau est rouge. Dans le tableau ci-dessous, le nombre qui correspond à la couleur rouge est 2. Le second anneau est également rouge, donc notre nombre est également 2. Le nombre que l'on obtient en combinant à la suite les deux anneaux est donc 22. Finalement, le multiplicateur est jaune, correspondant à 10 000. Donc, 22 x 10 000 nous donne 220 000, ou 220 k.

Chiffre	0	1	2	3	4	5	6	7	8	9	Or	Argent
Multiplicateur	1	10	10^2 100	10^3 1000	10^4 10000	10^5	10^6					
Précision	20%										5%	10%

VII-B-2 - Condensateur



Le condensateur (« capacitor » en anglais) est constitué de plaques de conducteurs, éléments qui permettent l'échange d'électricité, séparées par un isolant. Un condensateur est capable d'emmagasiner une tension électrique, un peu à la manière d'un réservoir. Sa valeur caractéristique est **la capacité, notée C et exprimée en farads (F)**. Il est souvent utilisé pour filtrer, c'est-à-dire lisser une tension (car il agit un peu comme un amortisseur) et il ne conduit l'électricité que si le courant change, par exemple lors de la mise sous tension ou l'extinction du circuit.

Les règles d'association sont l'inverse de celles des résistances.

En **parallèle**, les condensateurs s'additionnent : **Ceq = C1 + C2 + C3.**

Tandis qu'en **série : 1 / Ceq = (1 / C1) + (1 / C2) + (1 / C3).**

La formule associée au condensateur est : **i = C (dU / dt).**

Remarque : plus la tension change, plus le courant à ses pattes sera fort. Il faut parfois se méfier de ces **pics de courant à l'allumage et à l'extinction** du circuit.

VII-B-3 - Bobine (« Coil »)

La bobine est un enroulement de fil conducteur. La bobine est souvent utilisée pour filtrer un courant, générer un champ magnétique (électroaimant) ou amplifier un signal (radio). Sa valeur caractéristique est l'inductance **notée L et exprimée en Henry (H)**.

La formule associée à la bobine est : **U = L (di / dt)**

Remarque : plus la tension change, plus le courant à ses bornes sera fort. Pour cette raison, il faut prendre quelques précautions lorsqu'on commute une bobine dans un montage : utiliser par exemple une diode « de roue libre » (voir **« Diode »**) qui évacuera la **surtension à l'allumage et à l'extinction**.

VII-B-4 - Transistor

Le transistor est une association de trois couches de semi-conducteur et dont la couche du milieu sert à contrôler le passage du courant dans les deux autres. Il s'agit d'un composant actif qui est souvent utilisé comme interrupteur ou amplificateur, à la manière d'un relais. Il existe différents types de transistor au comportement différent, les NPN et PNP, les transistors à effet de champ, ou MOSFET.

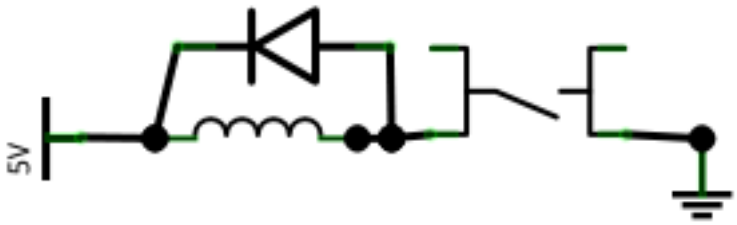
Pour plus de détails, consultez **http://fr.wikipedia.org/wiki/Transistor**.

VII-B-5 - Diode

La diode est composée de deux couches de semi-conducteur et ne laisse passer le courant que dans un sens : de l'anode vers la cathode ; du (+) vers le (-). Elle peut servir à bloquer des retours de courants non désirés ou construire

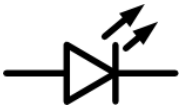
un pont redresseur pour passer d'un courant alternatif à un courant continu. Le trait présent sur le composant indique la cathode c'est-à-dire la borne négative (-).

VII-B-6 - Diode de « roue libre »



On utilise également la diode pour éliminer la surtension qui apparaît dans une bobine lors de l'allumage et de l'extinction. Il suffit de placer la diode en parallèle avec l'inductance (la bobine). C'est ce que l'on appelle une diode de « roue libre ». (Voir **projets « La cigarette ne tue pas les machines »**).

VII-B-7 - LED ou DEL



La LED est une diode électroluminescente : elle s'allume lorsqu'un courant passe dedans. Sa cathode (-) est plus courte que son anode (+). C'est un composant très pratique pour visualiser rapidement les états de certains circuits, car elle est facile à mettre en œuvre et consomme très peu de courant (en général 6 à 20 mA). Une LED se caractérise par sa tension de seuil qui exprime la tension à ses bornes lorsqu'elle est alimentée.

Couleur	Tension de seuil (Vf)	Consommation (If)	Longueur d'onde
rouge	1,6 V à 2 V	6 à 20 mA	650 à 660 nm
jaune	1,8 V à 2 V	6 à 20 mA	565 à 570 nm
vert	1,8 V à 2 V	6 à 20 mA	585 à 590 nm
bleu	2,7 V à 3,2 V	6 à 20 mA	470 nm

Il n'est pas bon d'alimenter une LED directement en 5 V (via une carte Arduino), car elle est en surtension : même si elle fonctionne elle brûlera rapidement. Pour la protéger, il faut utiliser le principe du pont diviseur de tension (expliqué plus loin) en la câblant en série avec une résistance, dont la valeur se calcule de la manière suivante :

- **R = (tension d'alimentation - tension de seuil) / courant**

Ainsi pour une LED rouge par exemple : R = (5 - 1,6) / 0,02 = 170 ohms

Remarque : la broche 13 de la carte Arduino est déjà équipée d'une résistance qui permet d'y brancher une LED directement.

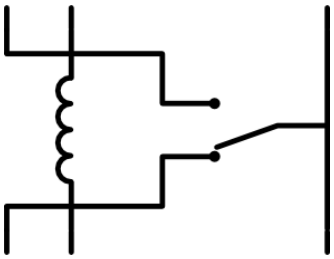
VII-B-8 - Potentiomètre

Le potentiomètre est en fait un composant à résistance variable, qui se règle avec un bouton ou une glissière. On change la résistance du potentiomètre par une manipulation physique ; c'est une interface humain/machine.

VII-B-9 - Interrupteur

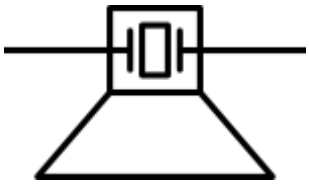
L'interrupteur ouvre ou ferme un circuit. Il est lui aussi une interface humain/machine. Il peut-être monostable (il revient à sa position initiale quand on le lâche) ou bistable (il garde sa dernière position). Il peut être NO (ouvert au repos) ou NF ou NC (fermé au repos). Il existe des interrupteurs combinant ces deux fonctions.

VII-B-10 - Relais



Le relais est un interrupteur électromécanique, ce qui signifie qu'il change de position ou d'état grâce à un électroaimant. Il peut donc être commandé par un signal électrique dans un circuit. Le relais est utilisé pour relayer une commande sur un circuit de plus forte puissance (voir chapitre « **Précautions d'utilisation** »). Comme l'électroaimant contient une bobine, il est nécessaire d'utiliser une « diode de roue libre ».

VII-B-11 - Piézoélectrique



Le piézoélectrique est un composant réversible qui génère une tension quand il est déformé et qui se déforme lorsqu'il est soumis à une tension. Il prend souvent la forme d'une pastille qui peut alors servir à la fois de micro-contact et de haut-parleur. Il peut ainsi servir d'interface humain/machine. Le signal sortant d'un piézo est assez erratique.

VII-B-12 - Cellule photoélectrique

La cellule photoélectrique ou photorésistance, est un semi-conducteur à résistance photo-variable. Elle permet de détecter et/ou de mesurer la lumière.

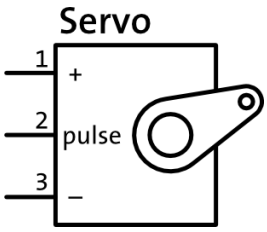
VII-B-13 - Thermistance

La thermistance est une résistance qui varie avec la température. La correspondance n'est pas linéaire, c'est pourquoi il est utile de se référer à la fiche technique du composant utilisé.

VII-B-14 - Moteur

Le moteur électrique est un dispositif électromécanique qui fonctionne grâce à l'électromagnétisme généré par des bobines.

VII-B-15 - Servomoteur



Le servomoteur est un petit moteur asservi, c'est-à-dire qu'un circuit interne contrôle en permanence sa position. Il a souvent une plage de liberté réduite (moins d'un tour) mais peut atteindre à coup sûr une position et la maintenir avec force. Il existe de nombreux modèles. Les plus petits peuvent être actionnés directement avec une carte Arduino (voir chapitre « **Précautions d'utilisation** »).

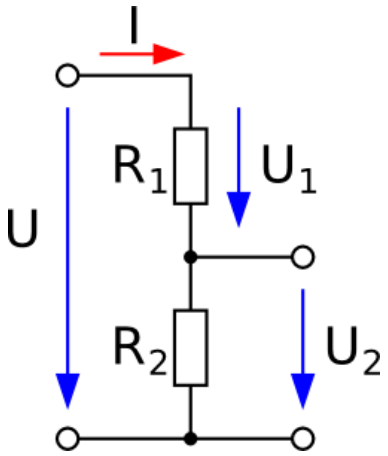
VII-B-16 - Circuits intégrés

Ce sont des circuits (donc des assemblages de composants) qui sont intégrés dans un même boîtier. Il en existe de nombreux types aux fonctionnalités et complexités diverses, du simple régulateur de tension au processeur d'ordinateur. La puce ATMEGA qui est au cœur de la carte Arduino est un circuit intégré. Ces composants sont très utiles pour réaliser des montages complexes en un minimum de place. Ils sont identifiés par une référence visible sur leur boîtier. Cette référence permet d'accéder à leur fiche technique (en anglais datasheet), en général facile à trouver sur Internet [référence du composant + « datasheet »] dans un moteur de recherche.

VII-C - Quelques circuits de base

L'idée ce cette partie n'est pas de vous montrer de manière exhaustive les circuits possibles, car cette tâche est irréalisable. En revanche, quelques circuits constituent le b.a.-ba de l'électronique : en effet ils apparaissent dans presque tous les montages. Connaître leur fonctionnement vous permettra de comprendre plus facilement un circuit.

VII-C-1 - Pont diviseur de tension



Ce circuit est très utile pour réduire une tension. Dans un pont diviseur, la tension aux bornes de la seconde résistance est proportionnelle au rapport de celle-ci par rapport à la résistance globale du circuit. Dans la métaphore de l'eau, les deux résistances constituent un escalier ou une cascade. Sa fonction est :

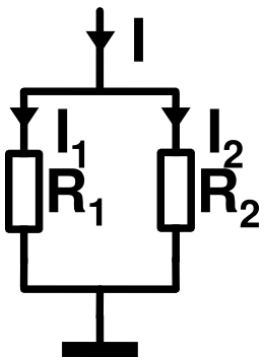
- **U2 = U X (R2 / (R2 + R1))**

Si l'on veut mesurer, par exemple, une tension qui oscille entre 0 et 15 V avec une carte Arduino qui ne supporte que 0-5 V, il faut utiliser un pont diviseur de tension constitué de deux résistances dont la première vaut le double de la seconde.La tension entre les deux résistances oscillera entre 0 et 5 V. La valeur totale des deux résistances déterminera la consommation électrique du circuit (voir « **Résistance** »).

Exemple : avec 1 MOhm et 2 MOhms, U2= 15X(1000/(1000+2000)) = 5 V.

On utilise ce modèle pour limiter la tension aux bornes d'une LED lorsqu'on l'alimente avec une carte Arduino (voir plus haut).

VII-C-2 - Pont diviseur de courant



Ce circuit est le pendant du pont diviseur de tension. Lorsqu'on branche deux résistances en parallèle, le courant se partage entre les deux branches proportionnellement au rapport des valeurs des résistances. Sa fonction est :

- **I1 = I X (R2 / (R1 + R2))**

VII-D - Le multimètre

Le multimètre est un ensemble d'appareils de mesure comprenant habituellement un voltmètre, un ampèremètre et d'un ohmmètre, pour mesurer la tension, le courant et la résistance de composants ou de circuits. Il comporte deux broches, habituellement rouge et noire, correspondant à la polarité (positive et négative) du multimètre. Il faut faire bien attention à respecter cette polarité pour certaines mesures et de s'assurer à chaque fois que la base de chaque broche est connectée à la bonne entrée du multimètre.Bien évidemment, il faut spécifier la grandeur à mesurer à l'aide du sélecteur du multimètre. Par exemple, pour mesurer une tension qui devrait se situer entre 0 et 5 V, on place le sélecteur à la barre des 20 V (si disponible). Pour un voltage de plus petite échelle, entre 0 et 1 V par exemple, il faudrait mettre le sélecteur sur le symbole 2 V.

Un multimètre comporte habituellement quatre entrées généralement identifiées par 10 A ou 20 A, mA, COM et V ou Ω. La base de la broche noire est toujours connectée à la broche COM qui correspond au pôle négatif ou au « Gnd » (Ground) sur la carte Arduino, ou à la masse. En fonction de la grandeur que l'on veut mesurer, le connecteur de la broche rouge se branche aux autres entrées : Entrée Utilisation 10 A ou 20 A pour mesurer un courant élevé.

Entrée	Utilisation
10 A ou 20 A	pour mesurer un courant élevé
mA	pour mesurer un courant faible
V, Ω, diode	pour mesurer une tension, une diode, résistance et continuité

La tension se mesure en parallèle et entre deux points d'un circuit, c'est-à-dire qu'il faut « créer » une nouvelle branche avec les broches du multimètre. Le circuit doit être sous tension pour effectuer la mesure.

Le courant se mesure en série, c'est-à-dire qu'il faut insérer le multimètre dans le circuit ou une branche.Le circuit doit être sous tension pour effectuer la mesure. Attention ! Lorsqu'on mesure le courant d'un circuit, il faut s'assurer de brancher la base de la broche positive (rouge) dans l'entrée appropriée du multimètre (souvent identifiée par mA ou A) en fonction de la puissance. Si cela n'est pas fait de manière correcte, il est possible d'endommager le multimètre. La résistance se mesure sans tension. Pour mesurer la résistance d'un composant, il suffit de connecter les deux broches du multimètre à chacune des pattes du composant que l'on souhaite mesurer.

Pour vérifier la continuité d'un circuit et pour voir s'il n'est pas « ouvert », si une connexion n'est pas interrompue ou encore si il n'y pas de court-circuit, il est possible d'utiliser la fonction « continuité » du multimètre. On prend cette mesure en connectant les broches du multimètre à chacune des extrémités du composant ou du circuit. Il est à noter que, par exemple, un circuit comprenant des résistances ne présente pas de continuité complète.

Pour la réalisation de circuits électroniques, plusieurs outils sont essentiels. Nous aborderons les principaux d'entre eux au chapitre Méthodologie, section Mise en œuvre.

VIII - Capteurs et actionneurs

L'électronique est une branche de la physique appliquée qui traite de la gestion de signaux électriques. La carte Arduino est un circuit électronique qui peut être programmé et qui permet de faire le pont entre le monde virtuel de la programmation informatique (concepts formels et logiques) et le monde physique (interaction électromécanique des objets). Nous abordons ci-après les notions de capteurs et actionneurs ainsi de quelque sous-catégories de ces deux grandes familles.

VIII-A - Les capteurs

Les capteurs sont des composants matériels, qui une fois correctement connectés à votre carte Arduino, peuvent fournir des informations sur le monde extérieur. Ceux-ci ont comme rôle de saisir une grandeur physique et de la convertir en information numérique. Il est ainsi possible de capter la plupart des phénomènes physiques comme le

vent, la pression, la température, la lumière, la torsion, la distance, etc. L'on peut classifier l'ensemble des capteurs en trois familles suivant le type d'informations qu'ils renvoient.

VIII-A-1 - Les capteurs logiques ou TOR

Ce type de capteur renvoie une information de type logique c'est-à-dire 0 ou 1. En d'autres termes, la sortie peut prendre uniquement deux états. Par exemple, un clavier est une matrice de capteurs logiques, car chaque touche est un interrupteur (ou un contact) qui peut être soit « ouvert » soit « fermé ».

VIII-A-2 - Les capteurs analogiques

Ce type de capteur renvoie une valeur proportionnelle à la grandeur physique mesurée. Par exemple une photorésistance (capteur de lumière) est un capteur qui convertit la luminosité en valeur électrique. Il existe des capteurs analogiques qui renvoient une information linéaire et d'autres dont la réponse suit une courbe différente. Par exemple, un potentiomètre linéaire sera noté B (nomenclature japonaise) et A (nomenclature européenne) tandis qu'un potentiomètre logarithmique sera noté A (nomenclature japonaise) et B (nomenclature européenne).

Courbe linéaire

Courbes non linéaire

Pour un capteur, une courbe linéaire signifie que la progression de sa valeur suit une progression constante et régulière. Une courbe non linéaire appartient à une fonction plus complexe dont la valeur ne progresse pas également dans le temps.

VIII-B - Les actionneurs

Les actionneurs sont des composants matériels qui, une fois correctement connectés à votre carte Arduino, permettent d'agir sur le monde extérieur. Ceux-ci ont comme rôle de convertir une valeur électrique en action physique. Il existe une infinité de types d'actionneurs différents et certains sont plus ou moins faciles à utiliser avec Arduino. Comme pour les capteurs, il existe une multitude de phénomènes physiques sur lesquels il est possible d'agir par le biais d'actionneurs comme les moteurs électriques, électroaimants, lumières, LED, éléments chauffants utilisés pour concevoir une multitude d'objets et d'applications comme des haut-parleurs, valves hydrauliques et pneumatiques, ventilateurs, pompes, grille-pain, canons à photon (un bon vieux téléviseur !), etc.

Beaucoup de machines et objets qui nous entourent comportent à la fois des capteurs et des actionneurs. Ils sont tous basés sur les principes de base de l'électronique que nous avons vus dans le chapitre précédent.

IX - Micro-contrôleur

Un micro-contrôleur est un petit processeur informatique relié à des entrées et des sorties numériques (0 ou 1) ou analogiques (tension variable). Il est capable de mémoriser et d'exécuter un programme visant à interpréter les entrées pour agir sur les sorties. Il se programme en général à l'aide d'un ordinateur mais peut fonctionner de manière autonome. En milieu industriel, les automates programmables qui servent à gérer et piloter des machines en sont une illustration. Ce type de machine intervient sur la commande d'un système mais ne peut délivrer beaucoup de puissance. Pour cela, on relaye ses commandes avec des transistors ou des relais.

Il existe de nombreux types et tailles de micro-contrôleurs avec plus ou moins d'entrées /sorties et d'interfaces. À petite échelle, les PIC (produits par la société Microchip) sont des circuits intégrés bien connus des électroniciens

depuis de nombreuses années. De nombreuses autres marques comme ATMEL, STMicroelectronics, Parallax ou Motorola produisent également des machines de ce type. Ils sont en général programmés en C/C++ et nécessitent des connaissances approfondies en électronique. La plate-forme Arduino a apporté une avancée majeure dans l'accessibilité de cette technologie au plus grand nombre, par sa simplicité d'utilisation et son coût abordable.

Depuis l'émergence de cette plate-forme, de nombreux projets parallèles ont été développés : par exemple, Pinguino (9) , Beeboard (10) , Freeduino (11) , Ginsing (12) ou Comet, Hackteria bioElectronics (13) , etc.

IX-A - Arduino

Arduino est un circuit imprimé en matériel libre (dont les plans de la carte elle-même sont publiés en licence libre, mais dont certains composants sur la carte, comme le microcontrôleur par exemple, ne sont pas en licence libre) sur lequel se trouve un microcontrôleur qui peut être programmé pour analyser et produire des signaux électriques, de manière à effectuer des tâches très diverses comme la domotique (le contrôle des appareils domestiques - éclairage, chauffage…), le pilotage d'un robot, etc.

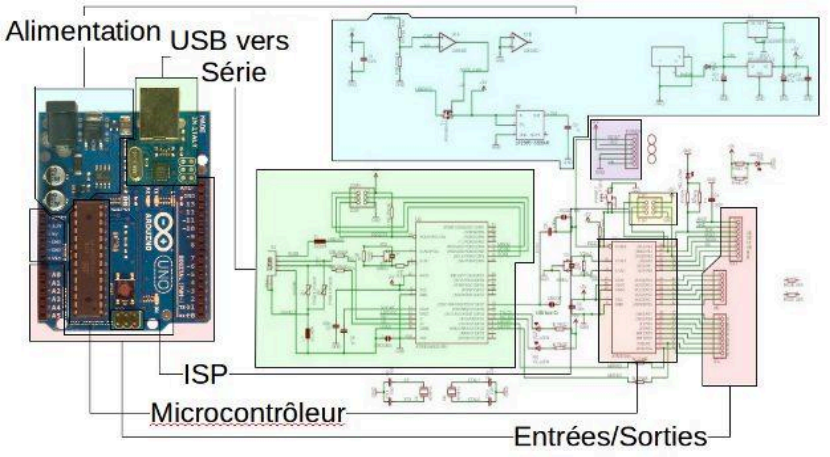


image Laurent Berthelot

IX-A-1 - Micro-contrôleur

La puce la plus courante qui équipe la carte Arduino est la ATMEGA328. Certains anciens modèles ont une puce ATMEGA168 qui est dotée d'un peu moins de mémoire. Les cartes Arduino Mega sont dotées d'une puce ATMEGA644 qui a plus de mémoire et plus d'entrées/sorties. Tous les processeurs de la famille ATMEGA se programment sensiblement de la même manière mais des différences peuvent apparaître pour des fonctions plus complexes. Pour plus de détails à ce sujet, référez-vous à la rubrique Matériel du site Arduino : **http://arduino.cc/en/Main/Hardware**.

Dans l'interface de programmation, le menu Tools > Board permet de définir avec quelle machine l'on travaille (voir chapitre « **Prise en main rapide** »). Le micro-contrôleur traite des informations reçues par ses entrées pour agir sur les sorties suivant un programme défini par l'utilisateur et ce via l'environnement de programmation Arduino.

IX-A-2 - Interface USB/série

Cette partie permet d'établir une communication avec un ordinateur, directement avec un câble USB, afin de programmer le contrôleur ou d'échanger des informations avec un programme qu'il exécute. Côté ordinateur, la carte Arduino apparaît au même titre que n'importe quel périphérique USB et nécessite l'installation d'un pilote (cela est expliqué dans le chapitre installation). Lorsqu'on utilise cette connexion, l'ordinateur assure directement l'alimentation de la carte Arduino via la liaison USB.

IX-A-3 - Alimentation

Ce circuit assure l'alimentation de l'ensemble des composants et des sorties suivant deux modes différents :

- lorsque la carte est connectée à un ordinateur via USB, c'est le port USB de l'ordinateur qui fournit l'énergie (5 V) ;
- lorsqu'on branche une source d'énergie au connecteur de la carte (batterie, transformateur ou pile), le système peut fonctionner de manière autonome.

Ce circuit inclut un régulateur de tension à 5 V mais il doit être alimenté entre 6 et 20 V. On conseille en général de l'alimenter plutôt entre 7 et 12 V pour garder une marge en basse tension et éviter que le circuit ne chauffe trop (car le régulateur de tension disperse toute surtension en chaleur). Sur les premiers modèles de cartes Arduino, un petit sélecteur permettait de choisir le mode mais depuis le modèle « Duemilanove », le passage de l'un à l'autre mode est automatique. Il ne faut pas brancher sur le 5 V de la carte des composants qui consomment plus de 500 mA.

IX-A-4 - Entrées/sorties

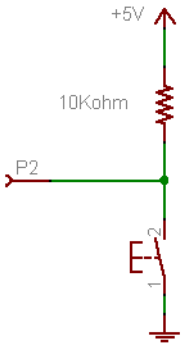
C'est par ces connexions que le micro-contrôleur est relié au monde extérieur. Une carte Arduino standard est dotée de :

- 6 entrées **analogiques**.
- 14 entrées/sorties **numériques** dont 6 peuvent assurer une sortie **PWM** (voir explication en bas dans la section « **Les sorties numériques** »).

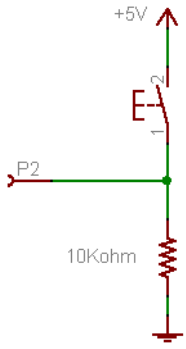
Les entrées analogiques lui permettent de mesurer une tension variable (entre 0 et 5 V) qui peut provenir de capteurs (14) ou d'interfaces diverses (potentiomètres, etc.).

Les entrées/sorties numériques reçoivent ou envoient des signaux « 0 » ou « 1 » traduits par 0 ou 5 V. On décide du comportement de ces connecteurs (entrée ou sortie) en général dans l'initialisation du programme (voir chapitre « **Programmer Arduino** »), mais il peut être aussi changé dans le corps du programme.

Lorsqu'on utilise une entrée numérique, il est important de s'assurer que le potentiel de l'entrée « au repos » est bien celui auquel on s'attend. En effet, si on laisse l'entrée « libre », c'est-à-dire câblée à rien, le potentiel qu'elle prendra ne sera pas nécessairement 0 V. On parle alors de potentiel flottant, car l'électricité statique ambiante ou les perturbations électromagnétiques peuvent faire apparaître des valeurs très fluctuantes. Pour s'assurer du bon fonctionnement, l'on utilise une liaison protégée par une résistance qui va « tirer vers le haut » (5 V) ou « tirer vers le bas » (0 V) le potentiel au repos, comme une sorte d'élastique. On utilise en général une résistance de 10 kOhms.



Dans un montage « tiré vers le haut » ou « pull-up », le potentiel de l'entrée au repos est 5 V.

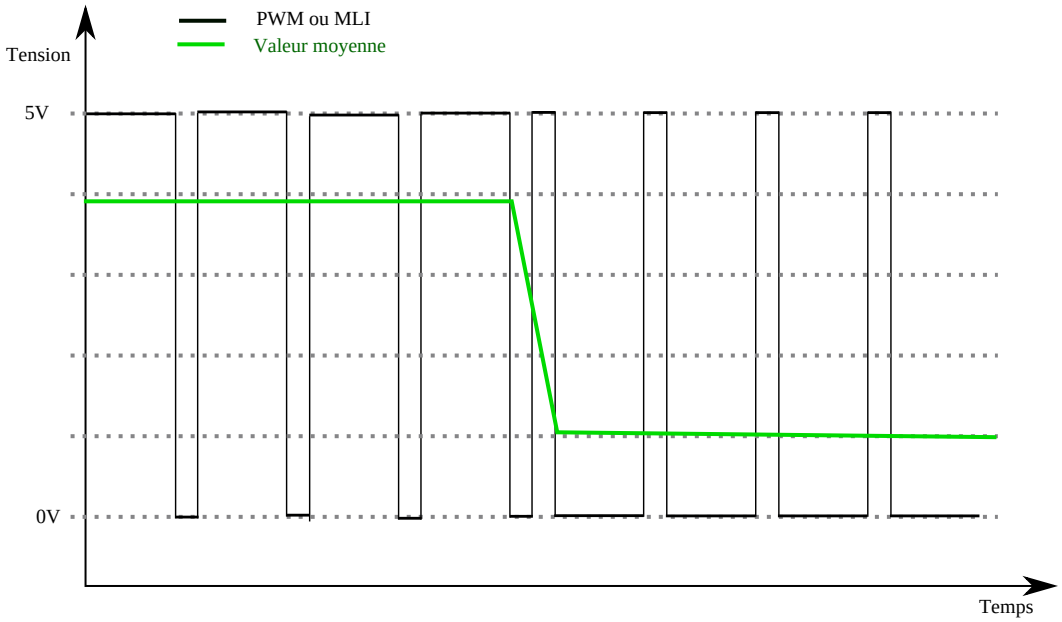


Dans un montage « tiré vers le bas » ou « pull-down », le potentiel de l'entrée au repos est 0 V.

Les sorties numériques peuvent actionner de nombreux composants (LED, transistor, etc.) mais elles ne peuvent pas fournir beaucoup de courant (40 mA pour une carte Arduino UNO). Pour piloter des circuits de plus forte puissance, il faut passer par des transistors ou des relais (voir chapitre « **Précautions d'utilisation** »).

La puce ATMEGA n'est pas capable de sortir des tensions variables. Heureusement, 6 des sorties numériques (N ° 3, 5, 6, 9, 10, 11) peuvent produire un signal **PWM**. Ce sigle signifie « Pulse Width modulation » en anglais ; en français l'on parle de MLI : "« Modulation de largeur d'impulsion ». Il s'agit d'un artifice permettant de produire une tension variable à partir d'une tension fixe.

La technique s'apparente approximativement à du morse : le signal de sortie est modulé sous forme d'un signal carré dont la largeur des créneaux varie pour faire varier la tension moyenne :



modulation de largeur d'impulsion ou PWM

IX-A-5 - ISP

ISP signifie « In-circuit Serial Programming ». Dans le cas où le micro-contrôleur de votre carte Arduino viendrait à ne plus fonctionner, il est possible de remplacer la puce défectueuse. Ainsi le port ISP vous permet de configurer votre nouvelle puce pour la rendre compatible avec l'IDE Arduino. En effet le micro-contrôleur de la carte Arduino possède un *bootloader* (ou bios) qui lui permet de dialoguer avec l'IDE. Il faut pour cela acheter un programmateur ISP ou il est aussi possible d'utiliser une autre carte Arduino comme programmateur ISP.

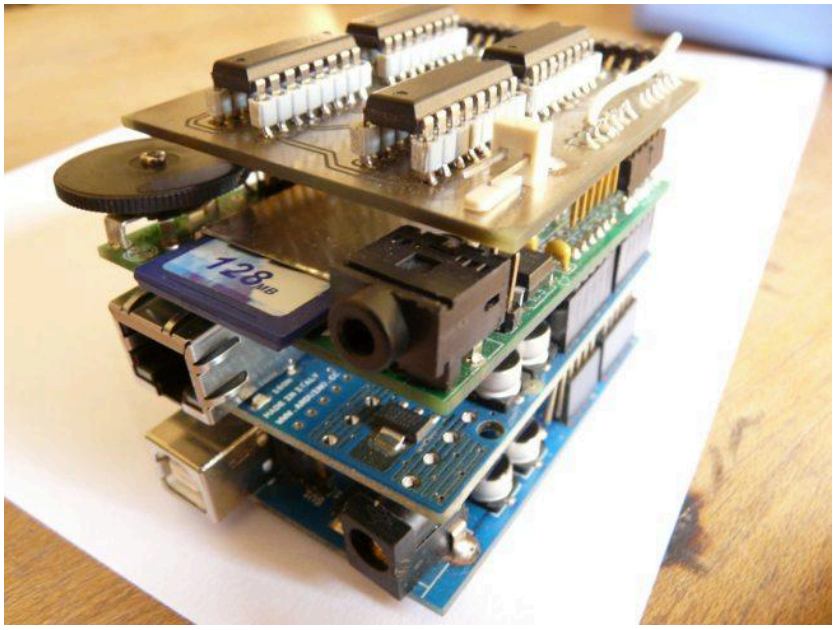
IX-A-6 - Circuit de commande

La carte Arduino est un circuit de commande qui sert principalement à contrôler des actionneurs et recenser les informations des capteurs. Il n'est pas conçu pour alimenter les éléments qui requièrent un certain niveau de puissance, comme des moteurs. Dans le cas où l'on voudrait utiliser de tels actionneurs, il faudrait utiliser un circuit additionnel que l'on désigne par circuit de puissance (voir chapitre « **Précautions d'utilisation** »).

IX-B - Circuits additionnels

Il est possible de spécialiser la carte Arduino en l'associant avec des circuits additionnels que l'on peut fabriquer soi-même ou acheter déjà montés. Lorsqu'ils se branchent directement sur la carte, ces circuits s'appellent des « shields » ou cartes d'extension. Ces circuits spécialisés apportent au système des fonctionnalités diverses et étendues dont voici quelques exemples :

- ethernet : communication réseau ;
- Bluetooth ou zigbee : communication sans fil ;
- pilotage de moteurs (pas à pas ou à courant continu) ;
- pilotage de matrices de LED : pour piloter de nombreuses LED avec peu de sorties ;
- écran LCD : pour afficher des informations ;
- lecteur de carte mémoire : lire ou stocker des données ;
- lecteur de MP3 ;
- GPS : pour avoir une information de position géographique ;
- joystick ;
- etc.



une carte Arduino équipée de plusieurs cartes d'extension

X - Précautions d'utilisation

Il est facile d'utiliser les nombreux exemples disponibles sur Internet sans véritablement connaître toutes les notions relatives à ces projets. Cependant il existe un certain nombre de notions de base qui vous permettront de mieux comprendre vos réalisations dans le but de les adapter plus précisément à vos besoins. Aussi, avant de se lancer dans vos premiers circuits, il est recommandé de prendre connaissance des connaissances qui suivent.

X-A - Savoir reconnaître les composants électroniques

Chaque composant possède une forme particulière ainsi qu'un code couleur ou un numéro d'identification qui vous permettent de connaître sa fonction et ses valeurs électriques. Chaque composant possède une documentation technique communément appelée *Datasheet* que l'on peut facilement trouver sur Internet. Cette documentation présente les caractéristiques précises du composant que vous souhaitez utiliser.

Attention : *certains composants se ressemblent mais n'ont pas la même fonction. Il faut toujours bien regarder les symboles qu'ils comportent et se référer à leur documentation technique.*

X-B - Représentations graphiques

Il vous sera utile de savoir déchiffrer les schémas techniques ou *schematics*, qui sont utilisés pour décrire les circuits électroniques. Ces schémas utilisent des symboles standardisés pour chacun des composants. Il existe deux standards en concurrence : le standard « Américain » et le standard « Européen » (voir « **Les bases de l'électronique** »).

X-C - Protection du port USB

Tout d'abord, il faut savoir que le port USB de votre ordinateur délivre une quantité limitée de courant électrique. En général un port USB peut fournir au maximum 500 mA en 5 V. Si vous souhaitez réaliser un montage qui nécessite plus de courant, il s'agira de prévoir une alimentation externe suffisamment puissante pour répondre à votre besoin.

X-D - Protection de la carte Arduino

Sachez que chacune des entrées/sorties de la carte ne peut pas délivrer plus de 20 mA. Cette quantité de courant est relativement faible mais permet, par exemple, de contrôler une diode électroluminescente DEL (ou LED en anglais) ainsi que des actionneurs de faible puissance tels qu'un piézoélectrique ou encore un petit servomoteur (voir « **Circuit de commande** »).

X-D-1 - Protection des entrées numériques

Comme leur configuration dépend du programme résident dans la carte Arduino, ce que l'on considère dans un montage comme des entrées numériques peuvent accidentellement être configurées en sortie (avec la commande *pinMode*) et réglées à 0 V (*LOW*). Dans ce cas, si cette « entrée » reçoit du 5 V, il se produira un court-circuit qui risque d'endommager partiellement ou intégralement la carte.

Pour éviter ce genre de mésaventure, lorsqu'on câble une entrée numérique, il est prudent d'y ajouter une résistance de 100 Ω qui limite le courant en cas de fausse manœuvre.

X-E - Protection des composants

Chaque composant possède ses propres conventions d'utilisation. Par exemple, il existe des composants qui possèdent un sens de branchement à respecter. L'on dit que ces composants sont polarisés. C'est le cas des LED, de certains condensateurs, des diodes, etc.

La plupart des composants ne peuvent pas fonctionner seuls, par exemple une LED a besoin d'une résistance appropriée pour ne pas s'user ou « brûler ». Cette résistance permet de limiter le courant qui traverse la LED. Le courant supprimé est alors dissipé en chaleur par la résistance (voir « **Loi d'Ohm** » dans le chapitre « Bases de l'électronique »).

X-F - Circuits de commande et de puissance

Lorsqu'un système comporte des moteurs ou des actionneurs demandant un certain niveau de puissance que le micro-contrôleur ne peut pas fournir, il faut avoir recours à deux circuits distincts interconnectés, appelés « circuit de commande » et « circuit de puissance ». Suivant la puissance présente, les interconnexions entre les deux circuits nécessiteront des précautions particulières.

X-F-1 - Circuit de commande

Comme son nom l'indique, c'est dans ce circuit que sont rassemblés tous les éléments de contrôle comme les boutons, les interfaces et le micro-contrôleur. Il est alimenté en basse tension : moins de 50 V, souvent 12 V, ou avec la carte Arduino 5 V. L'on pourrait l'assimiler au système nerveux d'un organisme : c'est ici que se prennent les décisions mais peu d'énergie y circule.

La manière la plus simple de relayer les commandes émergeant de ce circuit pour les transmettre au circuit de puissance est d'utiliser des transistors ou encore des relais.

Lorsque les tensions d'alimentation des deux circuits sont plus importantes ou si l'on veut protéger la commande de retours accidentels de courant provenant de la puissance, des optocoupleurs (plutôt que des transistors) assurent une isolation galvanique : l'information est transmise sous forme de lumière. Ainsi, les deux circuits sont complètement isolés électriquement.

X-F-2 - Circuit de puissance

Ce circuit alimente les composants nécessitant beaucoup d'énergie (habituellement les moteurs et autres actionneurs). Sa tension d'alimentation dépend des composants en question.

En fonction de cette tension et des conditions d'utilisation, les précautions à prendre sont variables :

- dans tous les cas, une protection contre les courts-circuits est conseillée : fusible ou disjoncteur pour éviter de détruire les composants ;
- **au dessus de 50 V, la tension peut menacer directement les humains** : protéger les pièces nues sous tension ;
- le 230 V nécessite un interrupteur différentiel qui protège les humains des contacts accidentels (en général tout local est doté de ce dispositif). Ne pas manipuler de 230 V sans connaissances appropriées.

Il est important de noter que ces deux circuits sont montés distinctement et sont isolés l'un de l'autre. Toutefois, il est primordial de lier leur masse (« Gnd ») afin qu'ils partagent le même potentiel de référence.

X-G - Courts-circuits

Votre carte ne doit pas être posée sur un support conducteur, car elle possède sur son verso des zones nues qui ne doivent pas êtres mises en contact afin de ne pas court-circuiter les composants entre eux. Il faut aussi **ne jamais connecter directement le port noté « Gnd » (pôle négatif) avec la broche 5 V (pôle positif)**.

XI - Programmer Arduino

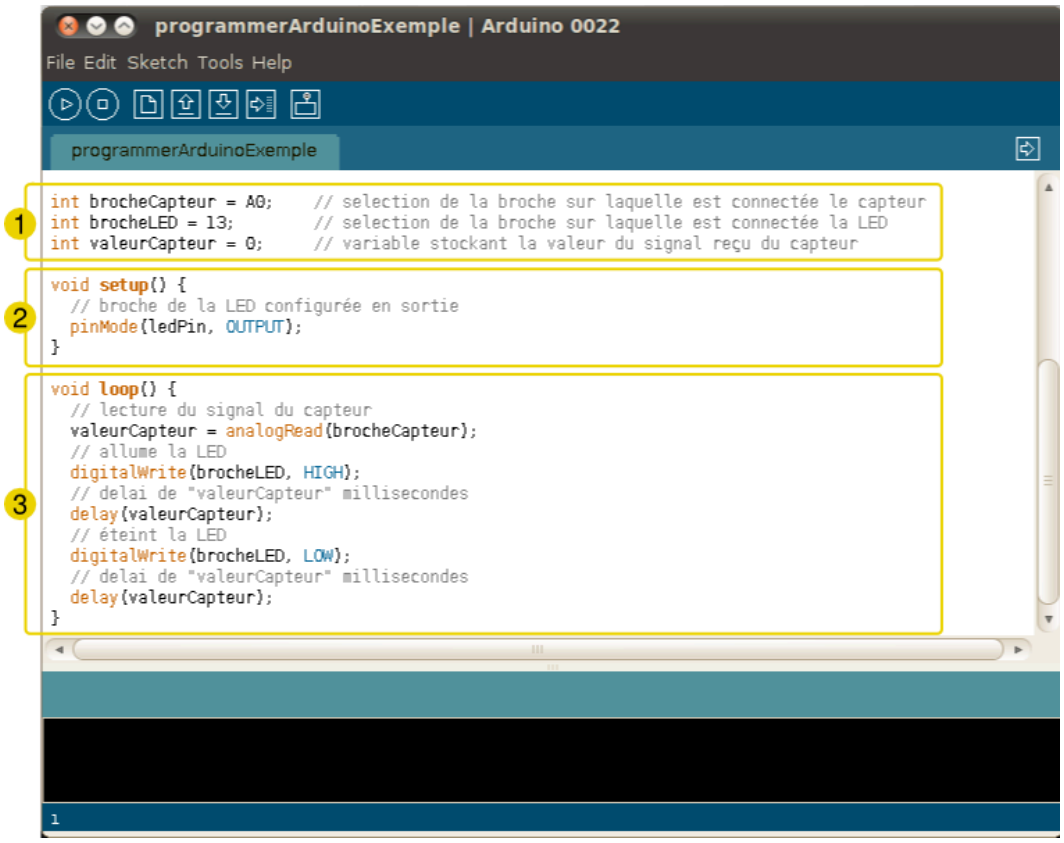
Ce chapitre présente le langage Arduino, son vocabulaire ainsi que la structuration d'un programme écrit en Arduino.

XI-A - Le langage de programmation

Un langage de programmation (15) est un langage permettant à un être humain d'écrire un ensemble d'instructions (code source) qui seront directement converties en langage machine grâce à un compilateur (c'est la compilation). L'exécution d'un programme Arduino s'effectue de manière séquentielle, c'est-à-dire que les instructions sont exécutées les unes à la suite des autres. Voyons plus en détail la structure d'un programme écrit en Arduino.

XI-B - La structure d'un programme

Un programme Arduino comporte trois parties :



1 La partie déclaration des variables (optionnelle) ;
2 La partie initialisation et configuration des entrées/sorties : la fonction setup () ;
3 La partie principale qui s'exécute en boucle : la fonction loop ().

Dans chaque partie d'un programme sont utilisées différentes instructions issues de la syntaxe du langage Arduino.

XI-C - Coloration syntaxique

Lorsque du code est écrit dans l'interface de programmation, certains mots apparaissent en différentes couleurs qui clarifient le statut des différents éléments.

En **orange**, apparaissent les mots-clés reconnus par le langage Arduino comme des **fonctions** existantes. Lorsqu'on sélectionne un mot coloré en orange et qu'on effectue un clic avec le bouton droit de la souris, l'on a la possibilité de choisir « Find in reference » : cette commande ouvre directement la documentation de la fonction sélectionnée.

En **bleu**, apparaissent les mots-clés reconnus par le langage Arduino comme des **constantes**.

En **gris**, apparaissent les **commentaires** qui ne seront **pas exécutés dans le programme**. Il est utile de bien commenter son code pour s'y retrouver facilement ou pour le transmettre à d'autres personnes. L'on peut déclarer un commentaire de deux manières différentes :

- dans une ligne de code, tout ce qui se trouve après « *//* » sera un commentaire ;
- l'on peut encadrer des commentaires sur plusieurs lignes entre « */** » et « ***/** ».

XI-D - La syntaxe du langage

XI-D-1 - ponctuation

Le code est structuré par une ponctuation stricte :

- **toute ligne** de code se termine par un point-virgule « **;** » ;
- le contenu d'une **fonction** est délimité par des accolades « **{** » et « **}** » ;
- les **paramètres** d'une fonction sont contenus pas des parenthèses « **(** » et « **)** ».

Une erreur fréquente consiste à oublier un de ces éléments.

XI-D-2 - Les variables

Une variable est un espace réservé dans la mémoire de l'ordinateur. C'est comme un compartiment dont la taille n'est adéquate que pour un seul type d'information. Elle est caractérisée par un nom qui permet d'y accéder facilement.

Il existe différents types de variables identifiés par un mot-clé dont les principaux sont :

- nombres entiers (int) ;
- nombres à virgule flottante (float) ;
- texte (String) ;
- valeurs vrai/faux (boolean).

Un nombre à décimales, par exemple *3.14159*, peut se stocker dans une variable de type float. Notez que l'on utilise un point et non une virgule pour les nombres à décimales. Dans Arduino, il est nécessaire de déclarer les variables pour leur réserver un espace mémoire adéquat. On déclare une variable en spécifiant son type, son nom, puis en lui assignant une valeur initiale (optionnel). Exemple :

```
int ma_variable = 45;
 // int est le type,  ma_variable le nom et = 45 assigne une valeur.
```

XI-D-3 - Les fonctions

Une fonction (également désignée sous le nom de procédure ou de sous-routine) est un bloc d'instructions que l'on peut appeler à tout endroit du programme.

Le langage Arduino est constitué d'un certain nombre de fonctions, par exemple `analogRead()`, `digitalWrite()` ou `delay()`.

Il est possible de déclarer ses propres fonctions, par exemple :

```
1. void clignote(){
2.     digitalWrite (brocheLED,  HIGH);
3.     delay (1000);
4.     digitalWrite (brocheLED,  LOW);
5.     delay (1000);
6. }
```

Pour exécuter cette fonction, il suffit de taper la commande :

```
clignote();
```

On peut faire intervenir un ou des **paramètres** dans une fonction :

```
1. void clignote(int broche,int vitesse){
2.     digitalWrite (broche, HIGH);
3.     delay (1000/vitesse);
4.     digitalWrite (broche, LOW);
5.     delay (1000/vitesse);
6. }
```

Dans ce cas, l'on peut moduler leurs valeurs depuis la commande qui l'appelle :

```
clignote(5,1000); //la sortie 5 clignotera vite
clignote(3,250); //la sortie 3 clignotera lentement
```

XI-D-4 - Les structures de contrôle

Les structures de contrôle sont des blocs d'instructions qui s'exécutent en fonction du respect d'un certain nombre de conditions.

Il existe quatre types de structure.

if...else : exécute un code **si** certaines conditions sont remplies et éventuellement exécutera un autre code avec **sinon.**

exemple :

```
1. //si la valeur du capteur dépasse le seuil
2. if(valeurCapteur>seuil){
3.     //appel de la fonction clignote
4.     clignote();
5. }
```

while : exécute un code **tant** que certaines conditions sont remplies.

exemple :

```
1. //tant que la valeur du capteur est supérieure à 250
2. while(valeurCapteur>250){
3.     //allume la sortie 5
4.     digitalWrite(5,HIGH);
5.     //envoi le message "0" au port série
6.     Serial.println(1);
7.     //en boucle tant que valeurCapteur est supérieure à 250
8. }
9.
10. Serial.println(0);
11.
12. digitalWrite(5,LOW);
```

for : exécute un code **pour** un certain nombre de fois.

exemple :

```
1. //pour i de 0 à 255, par pas de 1
2. for (int i=0; i <= 255; i++){
3.     analogWrite(PWMpin, i);
4.     delay(10);
5. }
```

switch/case : fait un choix entre plusieurs codes **parmi une liste de possibilités**

exemple :

```
// fait un choix parmi plusieurs messages reçus
switch (message) {
    case 0: //si le message est "0"
        //allume que la sortie 3
        digitalWrite(3,HIGH);
        digitalWrite(4,LOW);
        digitalWrite(5,LOW);
        break;
    case 1: //si le message est "1"
         //allume que la sortie 4
         digitalWrite(3,HIGH);
         digitalWrite(4,LOW);
         digitalWrite(5,LOW);
         break;
     case 2: //si le message est "2"
         //allume que la sortie 5
         digitalWrite(3,LOW);
         digitalWrite(4,LOW);
         digitalWrite(5,HIGH);
         break;
    }
```

XI-D-5 - Exemple

L'exemple qui suit montre l'utilisation de quelques éléments de la syntaxe du langage Arduino.

```
/*
Dans ce programme, un signal analogique provenant d'un capteur (potentiomètre)
fait varier la vitesse de clignotement d'une LED, à partir d'un certain seuil
*/
////déclaration des variables
// sélection de la broche sur laquelle est connectée le capteur
int brocheCapteur = 0;
// sélection de la broche sur laquelle est connectée la LED
int brocheLED = 13;
// variable stockant la valeur du signal reçu du capteur
int valeurCapteur = 0;
//seuil de déclenchement
int seuil= 200;
//////////initialisation
void setup () {
    // broche du capteur configurée en entrée
    pinMode (brocheCapteur, INPUT);
    // broche de la LED configurée en sortie
    pinMode (brocheLED, OUTPUT);
}
//////////boucle principale
void loop () {
    // lecture du signal du capteur
    valeurCapteur = analogRead (brocheCapteur);
    //condition de déclenchement
    if(valeurCapteur>seuil){
        //appel de la fonction clignote
        clignote();
    }
}
/////fonction personnalisée de clignotement
void clignote(){
    // allume la LED
    digitalWrite (brocheLED, HIGH);
    // délai de "valeurCapteur" millisecondes
    delay (valeurCapteur);
    // éteint la LED
    digitalWrite (brocheLED, LOW);
    // delai de "valeurCapteur" millisecondes
    delay (valeurCapteur);
}
```

Le principe de fonctionnement est le suivant :

1 Le signal est lu avec la fonction analogRead () :
2 La valeur du signal varie en 0 et 1023 ;
3 Si la valeur dépasse le seuil, la LED est allumée et éteinte pendant un délai correspondant à la valeur du signal reçu du capteur.

XI-D-6 - Pour aller plus loin

La liste exhaustive des éléments de la syntaxe du langage Arduino est consultable sur le site : **https://www.arduino.cc/reference/fr/**.

XII - Bien coder

L'utilisation d'un langage de programmation passe par l'apprentissage d'un vocabulaire et d'une syntaxe précise. Dans ce chapitre, nous aborderons quelques unes de ces règles ainsi que d'autres éléments à prendre en considération dans l'écriture d'un programme Arduino. Si quelqu'un doit lire ou modifier votre code, vous pouvez lui faciliter la tâche en attachant une attention particulière à sa structure et en y apposant des commentaires qui expliquent votre logique. Bref, un code clair offre une meilleure lecture, qu'il s'agisse de modifier ses configurations pour l'adapter à votre besoin ou de localiser la source d'une erreur.

XII-A - Conventions de nommage

La syntaxe d'un langage de programmation se base généralement sur une « convention de nommage (16) ». Il est important de respecter cette nomenclature pour rester dans l'esthétique du code.

Dans la déclaration des variables, il faut écrire les constantes en majuscules :

```
const int CAPTEUR = A0; // CAPTEUR est une constante.
```

Pour les autres variables (int, char, etc.), les fonctions et les sous-routines seront en minuscule. Dans le cas où ces variables ont des noms composés, le premier mot est en minuscule et les autres commencent par une majuscule. Par exemple :

```
int bookSprint = 2011;
```

Pour les sous-routines (voir explication plus bas), la règle est identique à celle des variables composées. Par exemple :

```
afficherValeurCapteur ();
```

Attention à ne pas omettre le point-virgule à la fin des instructions.

XII-B - Déclarer ses variables

Le nom des éléments (capteurs, actionneurs, etc.) utilisés dans un projet sont généralement repris par les variables du programme qui fait fonctionner le dispositif. Cette pratique permet de clarifier la fonction des différentes variables. Par exemple, pour choisir une variable qui a pour fonction de définir le port sur lequel est connectée une LED, nous choisirons la variable suivante :

```
int brocheLed = 13;
```

XII-C - Indenter son code

Il est conseillé d'effectuer un retrait par rapport à la ligne précédente à chaque nouveau bloc d'instructions. Les blocs d'instructions inscrits dans les fonctions et boucles sont délimités par des accolades, l'une est ouvrante et l'autre fermante.

```
1. if (etatCaptation == 1) {
2. if (valeurCapteur >= seuil) {
3. analogWrite (pinLed, HIGH);}
4. }
```

Le code ci-dessus n'est pas indenté. Après indentation, nous obtenons ceci :

```
1. if (etatCaptation == 1) {
2.     if (valeurCapteur >= seuil) {
3.         analogWrite (pinLed, HIGH);
4.     }
5. }
```

Il est à noter qu'un code indenté est plus attrayant et lisible. C'est un aspect qui facilite la recherche d'erreurs éventuelles qui pourraient se glisser dans le code.

XII-D - Faire des commentaires

Il n'y a rien de plus embêtant qu'une instruction écrite dans un code et dont on a oublie l'utilité quelques jours après. Les commentaires permettent de se retrouver dans son code. Ils facilitent en outre la compréhension du code pour ceux qui en auront éventuellement besoin.

Ci-dessous, un commentaire sur plusieurs lignes explique le fonctionnement du programme.

```
1. /*TEXTE BRILLANT
2.
3. Ce programme permet de piloter les LED à partir de certaines touches du clavier.
4. Principe:
5.
6. Saisir les caractères dans le moniteur sériel de l'interface de programmation Arduino pour
 allumer et éteindre les LED.
7. Saisir 'R' pour allumer la LED rouge,
8. Saisir 'J' pour allumer la LED jaune,
9. Saisir 'V' pour allumer la LED verte,
10. Saisir 'B' pour allumer la LED bleue,
11. Saisir 'E' pour éteindre les LED.
12.
13. */
```

Ci-dessous, une instruction suivie d'un commentaire explique l'instruction.

```
digitalWrite(pinLed0, HIGH); // Allumer la LED connectée à pinLed0
```

XII-E - Les sous-routines ou fonctions

Quand votre code commence à tenir une place importante et que vous utilisez à plusieurs reprises les mêmes blocs d'instructions, vous pouvez utiliser une sous-routine qui vous permet de mieux organiser et d'alléger votre programme Les sous-routines doivent être écrites après la boucle principale.

XII-F - Vérifier son code

Les erreurs dans un code sont parfois subtiles à retrouver. Il est donc conseillé de compiler régulièrement son code avec le bouton « Verify » ; les erreurs éventuelles apparaissent dans la console.

XII-G - Bonus

Pour indenter votre code vous pouvez utiliser la fonction « autoformat ». Cette fonction se base sur les conventions de programmation pour formater le code. Elle indente votre code automatiquement. Pour cela il faut aller dans le menu *Tools* > *Auto Format* ou utiliser le raccourci correspondant.

XIII - Bibliothèque externes

Une bibliothèque est un ensemble de fonctions utilitaires, regroupées et mises à disposition des utilisateurs de l'environnement Arduino afin de ne pas avoir à réécrire des programmes parfois complexes. Les fonctions sont regroupées selon leur appartenance à un même domaine conceptuel (mathématique, graphique, tris, etc.). Arduino comporte par défaut plusieurs bibliothèques externes. Pour les importer dans votre programme, vous devez les sélectionner dans *Sketch* > *Import Library*.

L'instruction suivante sera alors ajoutée au début de votre programme.

```
#include <la_bibliothèque.h>
```

Cette commande inclut au code source tout le contenu de la bibliothèque. Les fonctions qu'elle contient peuvent alors être appelées au même titre que les fonctions de base.

Pour info : les bibliothèques logicielles se distinguent des exécutables par le fait qu'elles ne s'exécutent pas « seules », mais sont conçues pour être appelées par d'autres programmes.

XIII-A - Bibliothèques fournies par défaut dans le logiciel Arduino

- EEPROM : lecture et écriture de données dans la mémoire permanente.
- Ethernet : pour se connecter à Internet en utilisant le *Shield Ethernet*.
- Arduino Firmata : pour rendre l'Arduino directement accessible à des applications en utilisant un protocole sériel.
- LiquidCrystal : pour contrôler les afficheurs à cristaux liquides (LCD).
- SD : pour la lecture et l'écriture de données sur des cartes SD.
- Servo : pour contrôler les servomoteurs.
- SPI : pour communiquer avec les appareils qui utilisent le protocole de communication SPI (Serial Peripheral Interface).
- SoftwareSerial : pour établir une communication sérielle supplémentaire sur des entrées et sorties numériques (la carte Arduino dispose d'un seul port sériel *hardware*).
- Stepper : pour commander des moteurs « pas à pas ».
- Wire : pour interfacer plusieurs modules électroniques sur un bus de données utilisant le protocole de communication TWI/I2C.

D'autres bibliothèques sont disponibles en téléchargement à l'adresse suivante :

- **http://www.arduino.cc/en/Reference/Libraries**.

Pour installer ces bibiothèques provenant de tiers, il faut décompresser le fichier téléversé et le stocker dans un répertoire appelé *libraries* situé dans le répertoire *sketchbook*. Sur Linux et Windows, le dossier *sketchbook* est créé

au premier lancement de l'application Arduino dans votre dossier personnel. Sur Mac OS X, un dossier Arduino est créé dans le répertoire « Documents ».

Par exemple, pour installer la *library* DateTime, le fichier devra être déposé dans le dossier : /libraries/DateTime de votre *sketch*.

XIV - Outils de programmation alternatifs

L'environnement de programmation Arduino offre une interface simple et pratique. Cependant il existe quelques logiciels alternatifs qui permettent de programmer la carte Arduino.

XIV-A - Pourquoi utiliser un autre programme qu'Arduino

Utiliser un langage de programmation que l'on maîtrise déjà permet de ne pas avoir à apprendre un nouveau langage pour programmer la carte Arduino. Cela permet aussi de réutiliser les bibliothèques et programmes que l'on a éventuellement déjà développés pour d'autres familles de micro-contrôleurs. Pour les programmeurs confirmés, le langage **C**/**C++** qui est traditionnellement utilisé pour programmer les micro-contrôleurs reste la solution la plus performante. D'autre part, si l'on possède des connaissances et l'on dispose de ressources techniques et de partenaires qui travaillent sur d'autres plates-formes, rester sur celles-ci est peut-être un choix pertinent. Voici une liste non exhaustive des logiciels qu'il est possible d'utiliser avec la carte Arduino.

XIV-A-1 - Les compilateurs

Un compilateur est un programme informatique qui traduit un langage (la source) en un autre (la cible), généralement dans le but de créer un programme exécutable le plus souvent un langage d'assemblage ou un langage machine.

Nom	Liens
Scons	**http://code.google.com/p/arscons**
build_arduino.py	**http://arduino.cc/playground/BuildArduino/Py**
Arduino CMake	**https://github.com/queezythegreat/arduino-cmake**
Ligne de commande Windows construction / upload	**http://arduino.cc/playground/Code/WindowsCommandLine**

XIV-A-2 - Environnements de développement intégré (IDE)

Voici des alternatives à l'IDE d'Arduino, ces logiciels permettent comme Arduino d'écrire, d'organiser et de compiler des programmes.

Nom	Liens
AVR-Ada	**http://arduino.cc/playground/Code/AVR-Ada**
Eclipse	**http://arduino.cc/playground/Code/Eclipse**
Bitlash	**http://arduino.cc/playground/Code/Bitlash**
kdevelop	**http://arduino.cc/playground/Code/Kdevelop**

XIV-A-3 - Bibliothèques

Voici des bibliothèques qui permettent d'étendre certains langages de programmation pour leur permettre de créer des programmes pour la carte Arduino.

Nom	Liens
CodeBlocks	**http://www.johnhenryshammer.com/WOW2/pagesHowTo/atmelPage.php#index**

Référence : **http://arduino.cc/playground/Main/DevelopmentTools**.

XV - Introduction des projets

Les projets qui vous sont présentés dans ce livre vous permettront de découvrir et de vous familiariser progressivement avec différentes possibilités et fonctions offertes par Arduino. Il est préférable de réaliser ces projets dans l'ordre proposé puisque la complexité de ceux-ci est progressive et se base sur les éléments de programmation que vous aurez vus précédemment.

Il existe déjà plusieurs tutoriels et aides en ligne pour apprendre Arduino. Dans le souci d'éviter la simple répétition, nous avons opté pour une approche par projets qui se déclinent en plusieurs étapes et à travers lesquels vous verrez différentes fonctions et techniques de programmation. En tout temps, il vous est possible de consulter l'aide du programme en allant dans le menu *Help* > *Reference* et *Help* > *Getting Started*.

La plupart des projets comportent des éléments électroniques de base qu'il vous sera facile de vous procurer. La liste de ces éléments sera donnée au début de chaque projet. Pour travailler avec les platines d'essai et établir des connexions, il est préférable d'utiliser des fils monobrins, rigides, plutôt que des fils multibrins et flexibles (voir chapitre « **Connectique** »).

Voici une liste des projets et des fonctions que vous découvrirez :

XV-A - Projet 1 : premier contact

Dans ce projet, nous allons utiliser un capteur piézo afin de faire varier l'intensité lumineuse d'une LED. Vous verrez comment déclarer et utiliser des variables, comment utiliser un capteur piézoélectrique pour détecter des déformations dans un matériau. Les fonctions vues seront les suivantes :

- pinMode() ;
- analogRead(), analogWrite(), *PWM*.

XV-A-1 - Projet 2 : texte brillant

Ce projet vous propose de piloter quatre LED avec le clavier de votre ordinateur, d'abord à travers le moniteur sériel, puis en utilisant un autre logiciel, en l'occurrence Pure Data. Vous verrez comment effectuer une communication sérielle d'un logiciel vers Arduino. Les fonctions abordées sont les suivantes :

- Serial.begin() , Serial.available() , Serial.read(), Serial.println() ;
- digitalWrite() ;
- if() et opérateurs de comparaison.

XV-A-2 - Projet 3 : la cigarette ne tue pas les machines

Vous en avez assez de fumer ? Confiez cette tâche ardue à l'Arduino! Ce projet vous introduira à l'utilisation d'un moteur DC et d'un capteur de « souffle ». Vous verrez également comment séparer un circuit de commande d'un circuit de puissance en utilisant un transistor, une diode et autres composants. Vous verrez comment éviter le bruit ou les fluctuations dans un signal de capteur en utilisant une mémoire tampon (le buffer) . Les nouvelles fonctions qui vous seront présentées sont les suivantes :

- boucle for() ;
- incrémentations (++).

XV-A-3 - Projet 4 : la petite bête qui a peur

Ce projet vous propose de créer une petite « bête » qui a peur et s'éloigne lorsque vous approchez votre main. Vous y verrez comment contrôler un servomoteur et intégrer à votre code une bibliothèque externe. Nous y ferons aussi la distinction entre des variables locales et globales, et aussi l'utilisation de sous-routines. Nous aborderons une nouvelle fonction qui est la suivante :

- while().

XV-A-4 - Projet 5 : oscilloscope

Ce projet va vous permettre de réaliser un instrument servant à mesurer et visualiser des tensions comprises entre 0 et 5 volts. Il s'agit d'un oscilloscope minimaliste. Vous verrez comment envoyer des informations sérielles vers un autre logiciel libre, Processing.

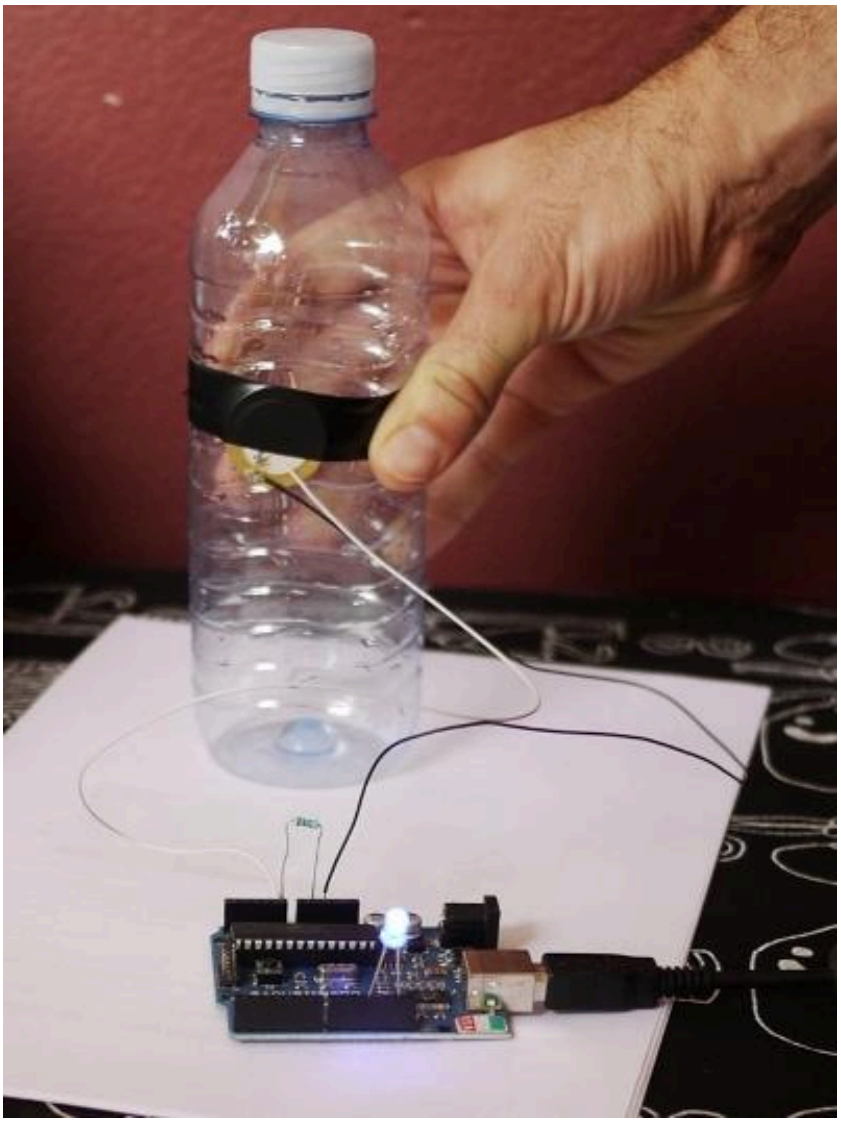
XV-A-5 - Projet 6 : perroquet

Vous manquez d'amis ? Avec ce projet, vous ne vous sentirez plus jamais seul ! Le perroquet vous permettra de construire un programme qui écoute une série d'impacts sur un objet et les répète à son tour. Vous verrez la programmation d'un filtre antirebond et l'utilisation de tableaux ou matrices. Les nouvelles fonctions abordées sont :

- fonctions logiques && et || ;
- millis() ;
- array[ ] ;
- switch().

XVI - Premier contact

Dans ce projet, nous allons utiliser un capteur piézoélectrique afin de faire varier l'intensité lumineuse d'une LED.Ce projet est simple et permet d'aborder les premiers concepts du système de programmation Arduino.



XVI-A - Matériel nécessaire à la réalisation du projet

• Une carte Arduino • Un capteur piézoélectrique • Une LED	• Une résistance 1 Mega Ohm • Un peu de ruban adhésif • Une bouteille en plastique vide

XVI-B - Première partie

Les capteurs piézoélectriques sont des composants qui créent un courant électrique lorsque qu'ils sont sollicités par des vibrations ou déformations. Collés sur une surface, ce capteur peut détecter des vibrations. C'est pourquoi dans cet exemple, la pastille piézoélectrique devra être nue et non pas encapsulée dans un boîtier plastique. Nous allons donc voir dans cette première partie, comment connecter un tel capteur à notre carte Arduino et afficher les valeurs captées.

XVI-B-1 - Schéma de montage

Afin de réaliser ce montage, nous allons connecter la patte négative de notre capteur piézoélectrique à la masse de notre carte Arduino soit une des *pins Gnd*. Puis, nous connecterons l'autre patte sur la *pin Analog In 0* de la

carte. Branchez ensuite la résistance, une patte dans une *pin Gnd* et l'autre dans *Analog In 0*. et pour finir, scotchez fermement votre capteur sur votre bouteille plastique.

XVI-B-2 - Programme

Vous pouvez maintenant copier le programme suivant dans la fenêtre du programme Arduino.

```
// Variable pour désigner quelle pin est utilisée pour le capteur
int capteurPiezo = A0; // pin Analog 0
// Variable pour contenir la valeur du capteur
int valeurPiezo;

void setup()
{
    // Établit la connexion sérielle à 9600 bauds
    Serial.begin(9600);
}

void loop()
{
    // Lit la pin du capteur et l'assigne à la variable valeurPiezo
    valeurPiezo = analogRead(capteurPiezo);
    // Affiche au moniteur sériel la valeur de valeurPiezo
    Serial.println(valeurPiezo);
    // Laisse un court délai afin que l'utilisateur puisse lire les valeurs correctement
    delay(100);
}
```

Sauvegardez maintenant votre programme et vérifiez-le en cliquant sur le bouton *Verify*. En cas d'erreur un message rouge s'affiche dans la console. Vérifiez alors la syntaxe et l'orthographe du code (voir chapitre « **Bien structurer son code** »). Si vous ne rencontrez pas d'erreurs, vous pouvez maintenant téléverser votre programme sur la carte en cliquant sur *Upload*.

Une fois le téléversement terminé, vous pouvez faire apparaître la fenêtre du moniteur sériel en cliquant sur le bouton *Serial Monitor*. Assurez-vous que la vitesse de transfert des informations (*baudrate*) du moniteur sériel est identique à celle spécifiée dans le programme soit dans notre cas, 9600. Ce paramètre peut être changé en bas à droite de la fenêtre.

Après quelques instants, vous devriez voir défiler des chiffres sur le moniteur sériel. Il s'agit de la valeur renvoyée par le capteur piézoélectrique. Vous pouvez maintenant prendre la bouteille dans vos main et la déformer pour faire changer la valeur qui s'affiche dans votre terminal. La valeur doit changer proportionnellement et simultanément à la déformation de la bouteille.

XVI-C - Deuxième partie : LED et variation de luminosité

Dans cette partie, nous allons ajouter une LED à notre montage afin d'illustrer les vibrations enregistrée par le capteur piézoélectrique pour faire varier l'intensité lumineuse d'une LED.

XVI-C-1 - Schéma de montage

Afin de compléter ce montage, il nous suffit de connecter la patte (+) de la LED à la *pin 13* et la patte (-) qui est la plus courte, à la masse de votre Arduino, soit à l'une des *pins Gnd*.

XVI-C-2 - Programme

Copiez maintenant le programme suivant dans la fenêtre de programmation Arduino.

```
// Variable pour désigner quelle pin est utilisée pour le capteur
int capteurPiezo = A0; // pin Analog 0
// Variable pour contenir la valeur du capteur
int valeurPiezo;
// Variable pour désigner quelle pin de l'Arduino est utilisée pour la LED
int ledPin = 13;

void setup()
{
    // Définit la pin 13 comme une sortie
    pinMode(ledPin, OUTPUT);
    // Établit la connexion sérielle à 9600 baud
    Serial.begin(9600);
}

void loop()
{
    // Lit la pin du capteur et l'assigne à la variable valeurPiezo
    valeurPiezo = analogRead(capteurPiezo);
    // Convertit la valeur du capteur en valeur pour la LED
    valeurPiezo = map(valeurPiezo, 0, 1023, 0, 255);
    // Affecte la valeur de puissanceLed à la pin 13
    analogWrite(ledPin, valeurPiezo);
    // Affiche au moniteur sériel la valeur de valeurPiezo
    Serial.println(valeurPiezo);
}
```

Pour contrôler la luminosité de la LED proportionnellement à la valeur du capteur, nous devons réaliser un conversion. En effet la fonction analogWrite(); prend comme argument une variable dont la plage est de 0 à 255. De son coté la fonction analogRead(); nous donne une valeur entre 0 et 1024. Nous devons donc ramener l'échelle de la fonction analogRead(); à une plage plus petite (0 à 255), c'est pourquoi nous devons utiliser la fonction map ci-dessous.

```
map (val origine, val origine min, val origine max, val sortie min, val sortie max);
```

Pour convertir une variable dont la plage est de 0 à 1023 en une variable dont la plage est de 0 à 255, cela donne la fonction suivante

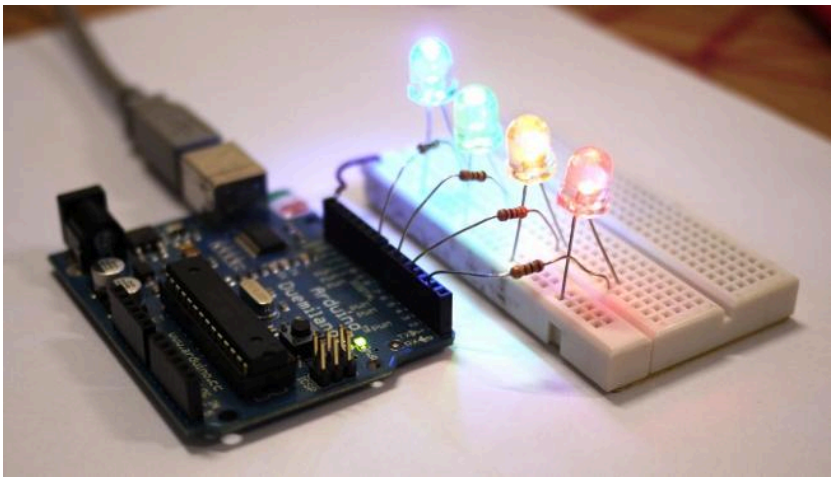
```
map (valeurPiezo, 0, 1023, 0, 255);
```

XVI-D - Pour aller plus loin

Afin d'aller plus loin, nous pourrions imaginer de remplacer le capteur piézoélectrique par tout autre capteur analogique comme un capteur de lumière ou encore un détecteur de proximité.

XVII - Texte brillant

Ce projet permet de monter plusieurs LED sur une carte Arduino. Les LED sont ensuite pilotées à partir de certaines touches du clavier de votre ordinateur pour en faire un magnifique jeu de lumière. Dans ce projet, nous utiliserons d'abord le moniteur sériel pour activer les lumières, puis nous montrerons un exemple avec le logiciel libre Pure-Data (**http://puredata.info/**).



XVII-A - Principe de fonctionnement

Des caractères sont saisis à partir du moniteur sériel de l'interface de programmation Arduino pour allumer et éteindre les LED.

- Saisir 'R ou r' pour allumer la LED rouge.
- Saisir 'J ou j' pour allumer la LED jaune.
- Saisir' V ou v' pour allumer la LED verte.
- Saisir 'B ou b' pour allumer la LED bleue.
- Saisir 'E ou e' pour éteindre les LED.

XVII-A-1 - Matériel nécessaire

1 Carte Arduino Duemilanove.
2 Platine de prototypage.
3 Quatre LED (rouge, verte, jaune et bleue).
4 Quatre résistances (130, 60, 130, 60 hms).
5 Fils de connexion.

XVII-B - Première étape : le montage du circuit

Les LED, leur résistance appropriée et le fil de connexion sont montés sur la platine d'essai. L'anode (patte positive) de chacune des LED est connectée à une résistance reliée à une pin numérique de la carte : la LED rouge est reliée à la pin 2, la jaune à la pin 4, la LED verte à la pin 6 et la LED bleue, à la pin 8.

La cathode (patte négative) de chacune des LED est connectée à un fil de connexion relié à une des *pins* GND de la carte. Les LED rouge et jaune utilisent chacune une résistance de 130 ohms et les LED verte et bleue une résistance de 60 ohms.

Après avoir réalisé le montage du circuit, il faut connecter la carte sur le port USB et lancer le logiciel Arduino pour y écrire le code de pilotage des LED.

XVII-C - Deuxième partie : le programme

Copiez le programme suivant dans la fenêtre de programmation d'Arduino :

```
1. //Déclaration des pins sur lesquelles sont connectées les LED
2.
3. int pinLed3 = 8; // LED Bleue
4. int pinLed2 = 6; // LED Verte
5. int pinLed1 = 4; // LED Jaune
6. int pinLed0 = 2; // LED Rouge
7.
8. //Déclaration de la variable contenant la valeur de la touche saisie au clavier
9.
10. int octetRecu;
```

```
1. void setup() {
2.
3.     // Initialisation de la communication série
4.
5.     Serial.begin(9600);
6.
7.     // Configuration des pins en sortie
8.
9.     // il est important pour le bon fonctionnement du montage de spécifier quelles pins seront
 utilisées comme sortie.
10.
11.     pinMode(pinLed0, OUTPUT);
12.     pinMode(pinLed1, OUTPUT);
13.     pinMode(pinLed2, OUTPUT);
14.     pinMode(pinLed3, OUTPUT);
15. }
```

```
1. void loop() {
2.     // Vérifie si il y a une donnée sérielle disponible
3.     if (Serial.available() > 0) {
4.         // Lecture de l'octet présent dans la mémoire tampon (buffer)
5.         octetRecu = Serial.read();
6.
7.         if (octetRecu == 'R' || octetRecu == 'r') { //Si l'octet reçu est égal à R ou r
8.             digitalWrite(pinLed0, HIGH); //Allumer la LED connectée à pinLed0
9.             Serial.println("LED Rouge allumee"); //Afficher "LED Rouge allumée" dans le
 moniteur série
10.         }
11.
12.         if (octetRecu == 'J' || octetRecu == 'j') { //Si l'octet reçu est égal à J ou j
13.             digitalWrite(pinLed1, HIGH); //Allumer la LED connectée à pinLed1
14.             Serial.println("LED Jaune allumee"); //Afficher "LED Jaune allumée" dans le
 moniteur série
15.         }
16.
17.         if (octetRecu == 'V' || octetRecu == 'v') { //Si l'octet reçu est égal à V ou v
18.             digitalWrite(pinLed2, HIGH); //Allumer la LED connectée à pinLed2
19.             Serial.println("LED Verte allumee"); //Afficher "LED Verte allumée" dans le
 moniteur série
20.         }
21.
22.         if (octetRecu == 'B' || octetRecu == 'b') { //Si l'octet reçu est égal à B ou b
23.             digitalWrite(pinLed3, HIGH); //Allumer la LED connectée à pinLed3
24.             Serial.println("LED Bleue allumee"); //Afficher "LED Bleue allumée" dans le
 moniteur série
25.         }
26.
27.         if (octetRecu == 'E' || octetRecu == 'e') { //Si l'octet reçu est égal à E ou e
28.             //Éteindre les LED connectées aux pinLed0, pinLed1, pinLed2 et pinLed3
29.             digitalWrite(pinLed0, LOW);
30.             digitalWrite(pinLed1, LOW);
31.             digitalWrite(pinLed2, LOW);
32.             digitalWrite(pinLed3, LOW);
33.             //Afficher "LED éteinte" dans le moniteur série
34.             Serial.println("LED eteinte");
35.         }
36.     }
37. }
```

XVII-C-1 - Déclaration des variables

Le caractère saisi au clavier ainsi que le numéro de la pin sur laquelle est connectée la LED à allumer sont stockés dans des variables déclarées au début du programme.

- Déclaration des variables stockant les numéros de pin :

```
1. int pinLed3 = 8; // LED Bleue
2. int pinLed2 = 6; // LED Verte
3. int pinLed1 = 4; // LED Jaune
4. int pinLed0 = 2; // LED Rouge
```

- Déclaration de la variable stockant la valeur du caractère saisi au clavier :

```
int octetRecu;
```

XVII-C-1-a - Initialisation et configuration

La communication sérielle entre l'ordinateur et la carte Arduino doit se faire à la même vitesse. Dans ce projet, nous utilisons une vitesse de 9600 Bauds. Il faut donc indiquer dans l'initialisation, la vitesse de la communication avec l'instruction :

```
Serial.begin (9600);
```

Certains logiciels ou périphériques peuvent fonctionner à d'autres vitesses, il serait donc nécessaire dans ces cas de modifier la commande précédente pour utiliser la bonne vitesse.

Une fois cette instruction réalisée, les ports sur lesquels sont connectées les LED sont configurés en sortie pour indiquer à la carte que des signaux seront envoyés de l'ordinateur vers les pins. La configuration se fait avec l'instruction pinMode().

```
1. pinMode(pinLed0, OUTPUT);
2. pinMode(pinLed1, OUTPUT);
3. pinMode(pinLed2, OUTPUT);
4. pinMode(pinLed3, OUTPUT);
```

XVII-C-1-b - Boucle principale

Le cœur du code de pilotage des LED se trouve dans la boucle principale ; les instructions qui s'y trouvent s'exécuteront aussi longtemps que la carte sera connectée à l'ordinateur.

C'est dans cette partie que la lecture du caractère saisi et la correspondance avec l'instruction d'allumage, et l'extinction des LED sont définies.

Puisque c'est à partir du clavier que le pilotage s'effectue, le programme est toujours en attente de saisie d'un caractère ; les instructions de pilotage sont donc contenues dans la structure :

```
1. if (Serial.available () > 0) {
2.     ...
3.     ...
4. }
```

Lorsque le moniteur sériel ou tout autre programme envoie une information sérielle à la carte Arduino, la donnée est stockée en mémoire par le port sériel de l'Arduino, et elle y demeurera jusqu'à ce qu'elle soit lue par la fonction Serial.read(). D'autre part, la fonction Serial.available() retourne une valeur correspondant au nombre de données

qui sont contenues dans le port. Si elle est plus grande que 0, cela signifie qu'il y a au moins une donnée présente dans le port (et qu'il faut donc aller la lire).

La structure précédente vérifie donc tout au long de l'exécution du programme s'il y a une donnée sérielle reçue par l'Arduino qui est en attente d'être lue ; si c'est le cas, le programme la lit.

- Lecture du caractère saisi :

La lecture du caractère se fait avec l'instruction Serial.read().

```
octetRecu = Serial.read ();
```

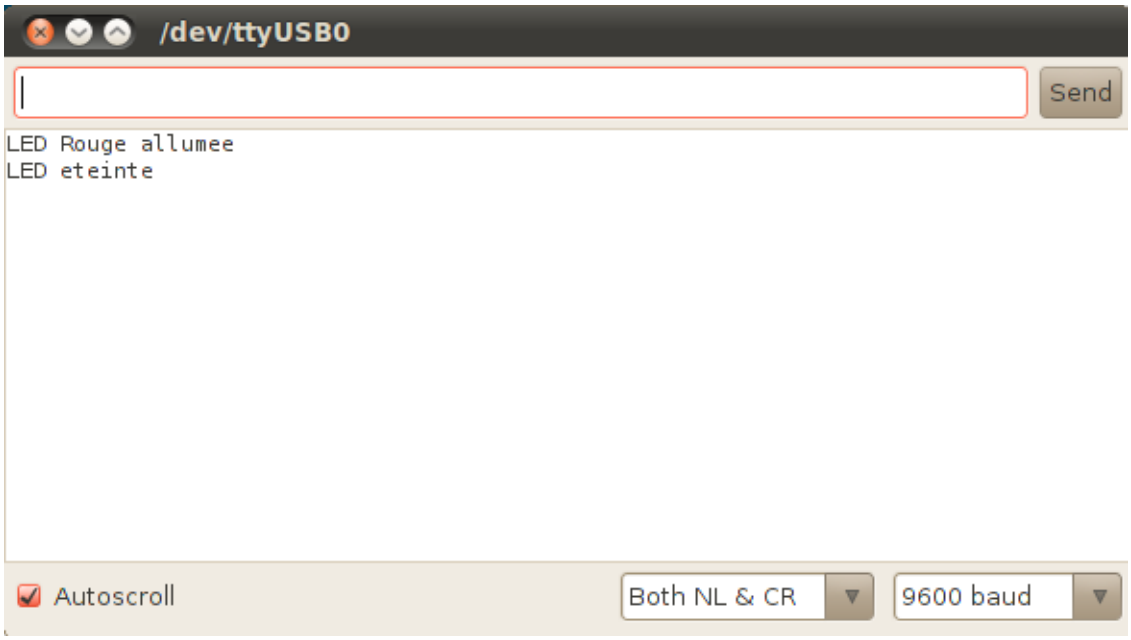
La donnée lue est ensuite stockée dans la variable octetRecu et retirée de la mémoire du port sériel de l'Arduino.

- Allumage et extinction des LED.

La valeur de la variable est ensuite analysée dans le but d'effectuer la correspondance entre le caractère saisi et la LED appropriée comme indiqué dans le principe de fonctionnement. Dans le reste du code, les commentaires à la fin de chaque ligne expliquent les instructions.

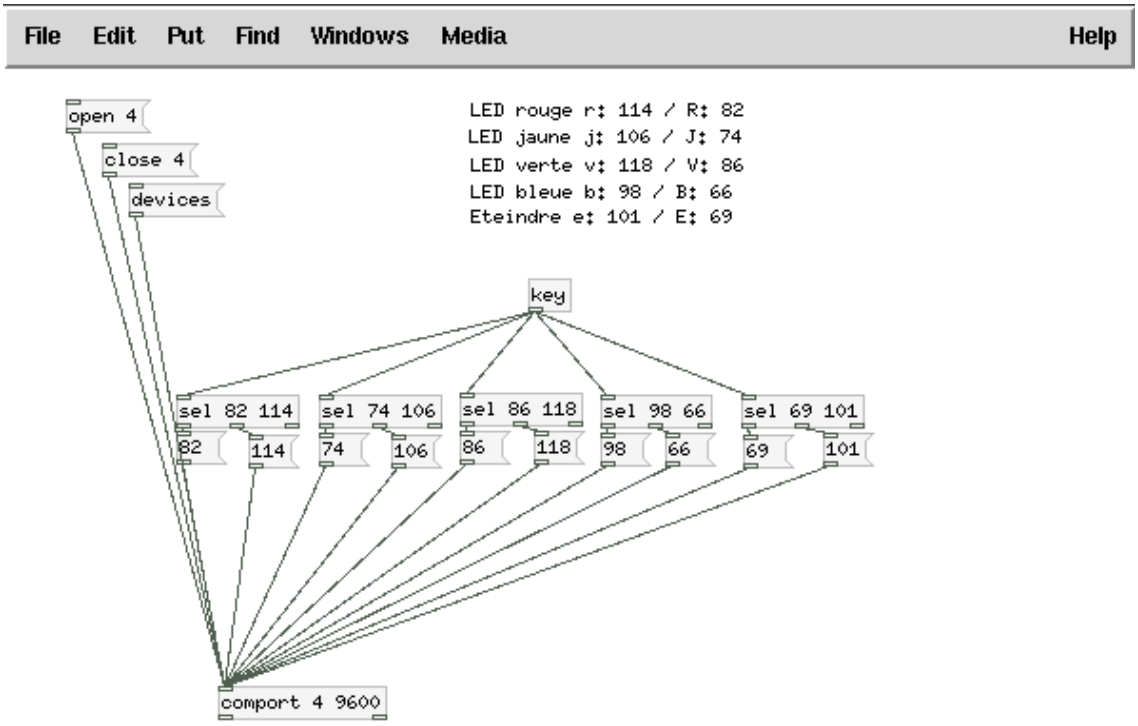
```
1. if (octetRecu =='R'|| octetRecu =='r') { //Si l'octet reçu est égal à R ou r
2.     digitalWrite (pinLed0,  HIGH); //Allumer la LED connectée à pinLed0
3.     Serial.println ("LED Rouge allumée"); //Afficher "LED Rouge allumée" dans le moniteur
 série
4. }
5.
6. if (octetRecu =='J'|| octetRecu =='j') { //Si l'octet reçu est égal à J ou j
7.     digitalWrite (pinLed1,  HIGH); //Allumer la LED connectée à pinLed1
8.     Serial.println ("LED Jaune allumée"); //Afficher "LED Jaune allumée" dans le moniteur
 série
9. }
10.
11. if (octetRecu =='V'|| octetRecu =='v') { //Si l'octet reçu est égal à V ou v
12.     digitalWrite (pinLed2,  HIGH); //Allumer la LED connectée à pinLed2
13.     Serial.println ("LED Verte allumée"); //Afficher "LED Verte allumée" dans le moniteur
 série
14. }
15.
16. if (octetRecu =='B'|| octetRecu =='b') { //Si l'octet reçu est égal à B ou b
17.     digitalWrite (pinLed3,  HIGH); //Allumer la LED connectée à pinLed3
18.     Serial.println ("LED Bleue allumée"); //Afficher "LED Bleue allumée" dans le moniteur
 série
19. }
20.
21. if (octetRecu =='E'|| octetRecu =='e') { //Si l'octet reçu est égal à E ou e
22.     //Éteindre les LED connectées aux pinLed0,  pinLed1,  pinLed2 et pinLed3
23.     digitalWrite (pinLed0,  LOW);
24.     digitalWrite (pinLed1,  LOW);
25.     digitalWrite (pinLed2,  LOW);
26.     digitalWrite (pinLed3,  LOW);
27.     //Afficher "LED éteinte" dans le moniteur série
28.     Serial.println ("LED éteinte");
29. }
```

L'image qui suit montre le résultat de la saisie d'un caractère 'R' suivi de celle du caractère 'E' dans le moniteur sériel.



La vitesse de communication mentionnée sur le moniteur sériel est la même que celle indiquée dans la fonction Serial.begin(). Comme le port sériel de la carte Arduino peut recevoir plusieurs données consécutivement, il est possible de s'amuser en envoyant des combinaisons de lettres. Par exemple, envoyer d'un même coup les lettres « ERV » avec le moniteur sériel fera d'abord éteindre les LED, puis allumer la rouge et la verte quasi instantanément.

XVII-D - Bonus : programme Pure Data



L'image ci-dessus est celle d'un programme Pure Data destiné au pilotage des LED connectées à la carte, lancez-le après avoir connecté la carte Arduino à l'ordinateur et téléversé le code ci-dessus.

Voyons en détail le fonctionnement de ce programme :

XVII-D-1 - Ouverture et fermeture du port série

En cliquant sur le message « devices », les ports USB sur lesquels sont connectées les cartes Arduino ainsi que leur numéro apparaissent.

```
[comport]: available serial ports:
        4       /dev/ttyUSB0
```

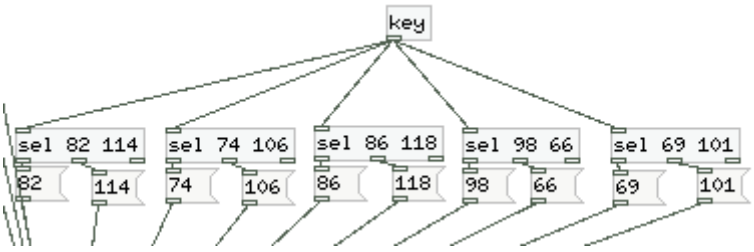
Dans notre cas, la carte est connectée sur */dev/ttyUSB0* et a 4 comme numéro de port.

Cliquer sur « open 4 » pour ouvrir le port numéro 4 et « close 4 » pour le fermer. Les messages « open » et « close » doivent être suivis **du bon numéro de port** (celui qui s'affiche en cliquant sur « devices »).

XVII-D-1-a - Traitement effectué sur les caractères

```
LED rouge r: 114 / R: 82
LED jaune j: 106 / J: 74
LED verte v: 118 / V: 86
LED bleue b: 98 / B: 66
Eteindre e: 101 / E: 69
```

Cette image est celle des commentaires sur la correspondance entre les caractères à saisir et les valeurs à envoyer à la carte.



L'objet « key » lit le caractère saisi au clavier et renvoie sa correspondance en chiffre. Les valeurs correspondantes aux caractères indiqués dans le principe de fonctionnement sont sélectionnées avec l'objet « sel » et envoyées à l'objet « comport ».

Cet objet fait office d'interface avec le port sériel et communique à une vitesse de 9600 bauds.

XVII-D-2 - Remarque

Les caractères saisis et reçus par port sériel sont convertis en octet (conversion ASCII) avant d'être envoyés à la carte.

Pour plus d'informations sur la communication série visitez le site : **http://wiki.t-o-f.info/index.php?n=Arduino.CommunicationS%C3%A9rie**.

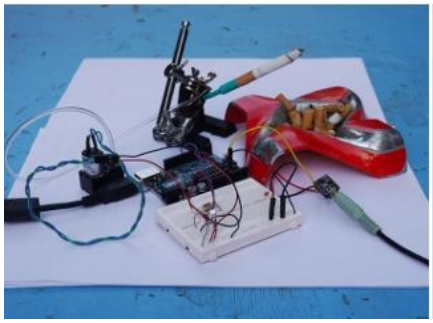
Pour plus d'informations sur le code ASCII des caractères visitez le site : **http://www.asciitable.com/**

Le montage de cet exemple est effectué à titre expérimental avec quatre LED ; la carte Arduino Duemilanove comporte 14 sorties numériques. Ainsi, d'autres LED pourraient être connectées sur les autres ports de la carte. Pour contrôler encore davantage de LED, il serait possible d'utiliser des circuits intégrés appelés *multiplexeurs.*

Il faut toujours prendre le soin de vérifier la résistance appropriée à chacune des LED avant de les ajouter au circuit pour ne pas les « brûler » (Voir **Section Électronique ; Chapitre Précautions d'utilisation, Protection des composants**.

XVIII - La cigarette ne tue pas les machines

Fumer tue ! Mais pas l'Arduino. Avec ce projet, vous pourrez laisser l'Arduino fumer à votre place ! Ce projet vous permettra d'utiliser l'information provenant d'un capteur de souffle et de l'utiliser pour contrôler un moteur lié à une petite pompe pneumatique. Vous verrez également comment éviter le bruit ou les fluctuations dans un signal de capteur en utilisant une mémoire « tampon » (buffer).



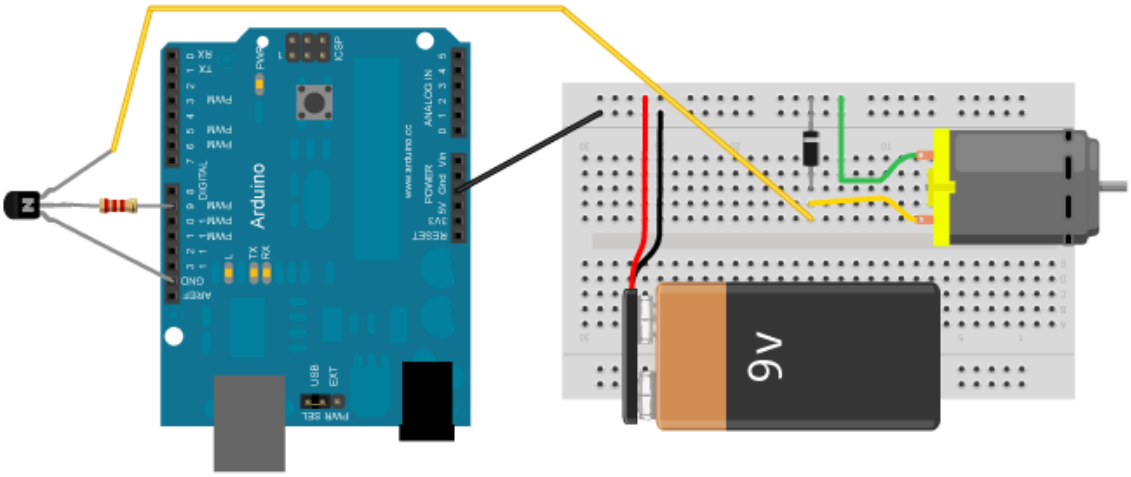

XVIII-A - Éléments nécessaires pour réaliser le projet

- un moteur DC simple et une source appropriée (batterie ou transformateur 9 V si moteur 9 V) ;
- un mécanisme de mini pompe pneumatique ;
- un capteur de souffle ou un circuit avec micro électret ;
- un platine d'essai et des fils ;
- une diode « de roue libre » quelconque ;
- un transistor NPN 2N3904 (ou un transistor semblable NPN) ;
- une résistance 2.2 kOhms ;
- une cigarette.

Remarque : vous pouvez réaliser ce projet sans cigarette en n'utilisant pas de mini pompe pneumatique et en remplaçant le capteur de souffle ou micro électret par un potentiomètre. L'idée du projet est d'apprendre à contrôler la puissance d'un moteur simple DC avec un capteur analogique.

XVIII-B - Première étape

Nous allons tout d'abord réaliser le circuit suivant afin de pouvoir utiliser le moteur DC.



XVIII-B-1 - Montage électronique

Comme le moteur nécessite une tension différente de celle fournie par la carte Arduino, nous devons monter deux circuits différents. Un circuit de commande où figure l'Arduino et un circuit de puissance où se trouve le moteur. Le transistor sert de « lien » entre les deux circuits en ce sens qu'il reçoit l'information de commande de la carte Arduino pour ouvrir et fermer le circuit de puissance qui active le moteur. Pour plus d'information sur les circuits de puissance, veuillez lire la partie « Circuit de commande et circuit de puissance » du chapitre « Précautions d'utilisation » dans la section « Électronique ».

Pour ce montage, nous allons d'abord connecter le transistor 2N3904. Si ce transistor ne vous est pas disponible, il est possible d'utiliser un autre transistor qui est également de type NPN. Ces transistors comportent trois bornes réparties différemment selon les modèles. Lorsqu'on fait face au 2N3904 par exemple (à sa partie « plate »), la borne de gauche s'appelle l'émetteur, la borne centrale la base et la borne de droite le collecteur.

Bornes du transistor 2N3904 (le brochage peut changer selon le modèle) :

- L'émetteur doit aller se brancher en direction d'un pôle négatif ou Gnd.
- La base doit se connecter à la branche de contrôle.
- Le collecteur doit aller se brancher en direction d'un pôle positif.

Donc, commencez par connecter l'une des bornes de la résistance 2.2k à la *pin* 9 de l'Arduino. L'autre borne de la *pin* devra se connecter la base du transistor. Vous pouvez également utiliser un fil pour passer de la *pin 9* Arduino à la platine d'essai, puis y insérer la base du transistor si cela est plus facile. Ensuite, la borne « émetteur » du transistor doit aller au Gnd. Finalement, la borne « collecteur » du transistor doit aller dans une branche de la platine d'essai.

Maintenant, nous allons ajouter le moteur. Connectez un des fils du moteur (le négatif, si possible), dans la branche de la platine d'essai où se trouve déjà la partie « collecteur » du transistor. L'autre fil (le positif) du moteur doit aller directement dans la branche principale positive (9 V) de la platine d'essai. Ensuite, connectez la borne négative de la diode (celle qui provient du côté de la diode marqué d'un trait) dans la branche principale positive et la patte positive de la diode dans la branche de la platine où le transistor et le moteur sont connectés. Vous aurez peut-être remarqué que la diode est branchée à l'envers (le négatif dans le positif). Cela est normal. Il s'agit d'un principe de diode « de roue libre ». Le rôle ici de la diode est d'éviter une surtension et sert à protéger les composants comme le transistor. (voir chapitre « **Les bases de l'électronique** »).

Il vous faut brancher l'alimentation à la platine d'essai, le pôle négatif de la batterie ou du transformateur dans la branche principale négative de la platine et le pôle positif dans la branche positive principale. Finalement, pour lier les masses ou le *Gnd* des deux circuits (celui de commande comprenant l'Arduino et celui de puissance qui comporte

le moteur). Pour ce faire, connectez un fil à l'une des *pins Gnd* de la carte Arduino à la branche principale négatif de la platine d'essai.

XVIII-B-2 - Programmation

Le programme suivant fera accélérer puis décélérer le moteur de manière graduelle et répétitive.

```
1. // déclarer la variable pour contenir la pin associée au moteur
2. int pinMoteur = 9;
3.
4. void setup()
5. {
6.     // spécifier que la pin liée au moteur est une sortie
7.     pinMode(pinMoteur, OUTPUT);
8. }
9.
10. void loop()
11. {
12.     // boucle pour faire accélérer le moteur
13.     for (int i = 0 ; i <= 255 ; i++)
14.     {
15.         //envoyer la valeur de 0 à 255
16.         analogWrite( pinMoteur, i);
17.         // attendre 10 millisecondes pour donner le temps de réaliser l'accélération
18.         delay(10);
19.     }
20.
21.     // boucle pour faire décélérer le moteur
22.     for (int j = 0 ; j <= 255 ; j++)
23.     {
24.         //envoyer la valeur de 255 à 0
25.         analogWrite( pinMoteur, 255 - j);
26.         // attendre 10 millisecondes pour donner le temps de réaliser l'accélération
27.         delay(10);
28.     }
29. }
```

Le code ci-dessus nous permet d'abord de tester le contrôle du moteur. Nous utilisons la commande analogWrite(); pour envoyer un signal au transistor. Pour plus d'informations sur le *PWM*, veuillez lire la partie « Entrées / Sorties » du chapitre « Micro-contrôleurs » de la section « Électronique ». Ce signal nous permet de varier la vitesse de rotation du moteur en envoyant de courtes impulsions « on/off ». La commande analogWrite(); prend deux paramètres : la *pin* à laquelle nous voulons nous adresser et une valeur entre 0 et 255, correspondant dans notre projet à une intensité « très faible » pour le 0 jusqu'à une quasi pleine intensité pour 255.

L'autre fonction que nous utilisons dans ce programme est la boucle for();. Celle-ci permet de répéter un certain nombre de fois une série d'instructions. Dans notre code, nous répétons 256 fois la commande analogWrite(); en incrémentant à chaque fois la valeur passée en paramètre. Notre boucle for(); nécessite l'utilisation d'une variable et la spécification de trois instructions pour son bon fonctionnement :

```
for (int i = 0 ; i <= 255 ; i++)
```

1. Le premier élément déclare la variable à utiliser et son état de départ. Dans notre exemple, nous déclarons la variable : int i = 0; ;
2. Le deuxième élément précise la condition à remplir pour arrêter la boucle : i <= 255; cela signifie que tant que i sera plus petite ou égale à 255, les instructions comprises entre les accolades seront répétées.
3. le dernier élément précise l'opération à exécuter après chaque boucle : i++;. Donc, dès que les instructions entre accolades sont exécutées une première fois, la valeur de i sera incrémentée de 1. i++; est un raccourci qui est équivalent à écrire i = i + 1;.

Au départ, i prend donc la valeur 0. Nous spécifions cette valeur à analogWrite();. Puis, le programme attend 10 millisecondes avec la fonction delay(10); pour laisser un temps au moteur de bouger un peu. Un délai plus long ferait en sorte que le moteur change d'état plus lentement (la boucle s'exécuterait avec plus de temps d'attente

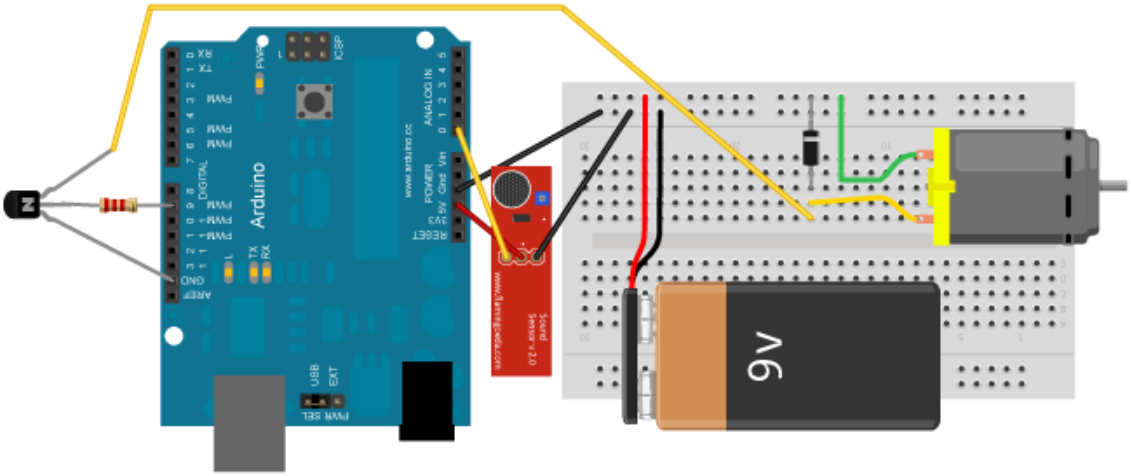
entre chaque étape). Lorsque ces instructions sont terminées, la boucle reprend et i est incrémenté. La variable a donc maintenant une valeur de 1. La condition est testée i <= 255 et retourne “faux”, donc la boucle continue. Les instructions sont exécutées, le moteur tourne *un peu* plus vite et i est ensuite incrémenté à 2. Et ainsi de suite jusqu'à ce que i soit incrémenté à 256, ce qui rend la condition « vraie » et qui fait sortir le programme de la boucle for();.

La deuxième boucle du programme s'exécute de la même manière, à la différence que nous utilisons la variable j et que pour effectuer une décélération du moteur, nous utilisons l'opération 255-j pour spécifier la valeur à prendre pour analogWrite(). En effet, lorsque j est égal à 0, la valeur retournée par 255-j est égale à 255. Et ainsi de suite. Notre boucle for() passera donc en réalité les valeurs de 255 à 0 à analogWrite().

XVIII-C - Deuxième étape

Dans cette deuxième étape, notre Arduino se dévouera pour fumer à votre place. Nous connecterons donc le capteur de souffle et modifierons le code pour que lorsque vous soufflez ou aspirez près du micro électret, l'Arduino active sa micro-pompe.

XVIII-C-1 - Montage électronique



À présent, il nous faut incorporer notre capteur de souffle au circuit. Notez que vous pouvez utiliser un potentiomètre pour simuler ce capteur et réaliser le projet. Il est toutefois inutile de souffler sur le potentiomètre, il ne réagira pas! Le capteur que nous utilisons ici a trois fils. Le fil négatif (-) doit aller à la *pin Gnd* de l'Arduino. Le fil positif (+) doit être connecté à la *pin* 5 V de la carte Arduino. Finalement, le fil du signal doit être connecté à la *pin* 0 de l'Arduino.

XVIII-C-2 - Programmation

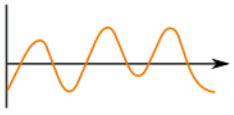
```
1. // déclarer la variable pour contenir la pin associée au moteur
2. int pinMoteur = 9;
3. // déclarer la variable pour contenir la pin associée au capteur
4. int pinCapteur = A0;
5. // déclarer la variable pour contenir la valeur du capteur
6. int valeurCapteur;
7. // déclarer la valeur minimale pour déclencher le moteur
8. int seuilCapteur = 550;
9. // déclarer la variable pour contenir accumuler des valeurs tampon
10. int tampon = 0;
11. // déclarer la variable pour contenir la valeur d'arrêt pour le tampon
12. int seuilTampon = 5;
13.
14. void setup()
15. {
16.     // établir connection sérielle
17.     Serial.begin(9600);
```

```
18.     // spécifier les entrées et sorties
19.     pinMode(pinMoteur, OUTPUT);
20.     pinMode(pinCapteur, INPUT);
21. }
22.
23. void loop()
24. {
25.     // lire le capteur et mettre dans variable
26.     valeurCapteur = analogRead(pinCapteur);
27.     // afficher la valeur pour déboguer
28.     Serial.println(valeurCapteur);
29.
30.     // si la valeur du capteur est supérieure au seuil fixé
31.     if(valeurCapteur > seuilCapteur)
32.     {
33.         // transformer les valeurs analogiques 0-1023 à 100 à 255
34.         valeurCapteur =  map(valeurCapteur, 0, 1023, 100, 255);
35.         // envoyer valeur au moteur
36.         analogWrite(pinMoteur, valeurCapteur);
37.         // remettre le compteur tampon à zéro
38.        tampon = 0;
39.     }
40.     else{ // si le capteur est inférieur
41.         // incrémenter le tampon
42.         tampon++;
43.     }
44.
45.     // si le compteur tampon a dépassé le seuil
46.     if(tampon >= seuilTampon)
47.     {
48.         // arrêter le moteur
49.         analogWrite(pinMoteur, 0);
50.     }
51.
52.     // attendre 100 millisecondes pour avoir moins d'erreurs de lecture
53.     delay(100);
54. }
```

Jetons d'abord un œil à nos variables :

- pinMoteur sert à identifier la *pin* liée au moteur ;
- pinCapteur sert à identifier la *pin* liée au capteur ;
- valeurCapteur sert à emmagasiner la valeur du capteur ;
- seuilCapteur le seuil sert à préciser à partir de quelle valeur du capteur nous pouvons considérer l'entrée d'information comme un « souffle ». Plus l'utilisateur souffle fort dans le capteur, plus la valeur augmente. C'est pourquoi nous retrouvons la fonction if(valeurCapteur >= seuilCapteur) dans notre code.

tampon et seuilTampon servent à une fonction spéciale. Le micro électret convertit du son en information numérique. Un signal sonore représenté numériquement est en fait une oscillation, une modulation d'amplitude de signal à travers le temps. Un son continu à l'oreille ne l'est pas en réalité, puisque c'est une vibration. Les données numériques associées à ce son comportent donc des valeurs qui peuvent avoir de grandes variations.



Nous voulons que le moteur ne s'enclenche qu'à partir d'un certain seuil de volume (valeurCapteur) mais la variation de l'amplitude d'un son, même suffisamment fort, comportera des valeurs inférieures à ce seuil et qui feront s'arrêter le moteur. Pour éviter cela, nous utilisons le tampon. En effet, à chaque fois que valeurCapteur descend en bas de seuilCapteur, et ce, même lors d'un son continu, nous incrémentons le tampon plutôt que de fermer immédiatement le moteur. Si valeurCapteur remonte en haut de seuilCapteur, signifiant que nous sommes en présence d'un son continu suffisamment fort, nous remettons tampon à zéro. Toutefois, si pour plusieurs lectures consécutives du capteur, nous obtenons une valeur inférieure à seuilCapteur, cela sera interprété comme un arrêt du son continu

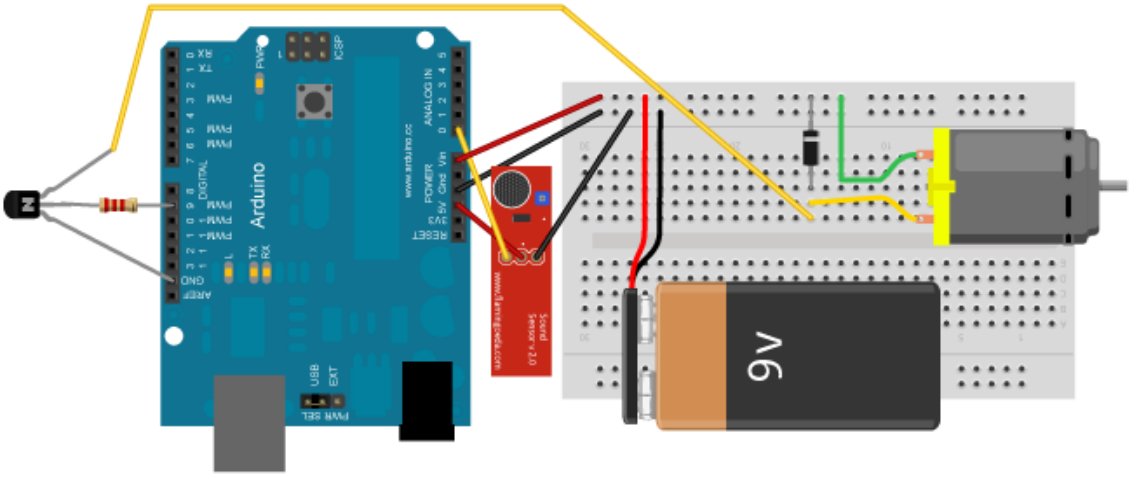
et nous arrêterons ainsi le moteur. Le nombre de lectures consécutives est spécifié par `seuilTampon` et la condition mentionnée précédemment est vérifiée par `if( tampon >= seuilTampon )`.

La dernière fonction à décrire que nous utilisons dans le code est `map()`. Cette fonction nous permet de reporter les valeurs de lecture analogique (0 à 1023) sur des valeurs sélectionnées par le programmeur pour le bon fonctionnement du moteur (100 à 255). `map()` est explicitée plus en détail dans le projet *Premier Contact*.

XVIII-D - Traîner au parc avec son Arduino fumeur

Si l'envie vous prend d'emmener votre petit fumeur prendre l'air frais, rien n'est plus simple. Vous n'avez qu'à déconnecter le câble USB de la carte Arduino. Puis, connectez l'extrémité d'un fil à la broche VIN de l'Arduino et l'autre extrémité à la branche principale positive de la platine d'essai. Voilà ! Votre carte Arduino devrait fonctionner seule.

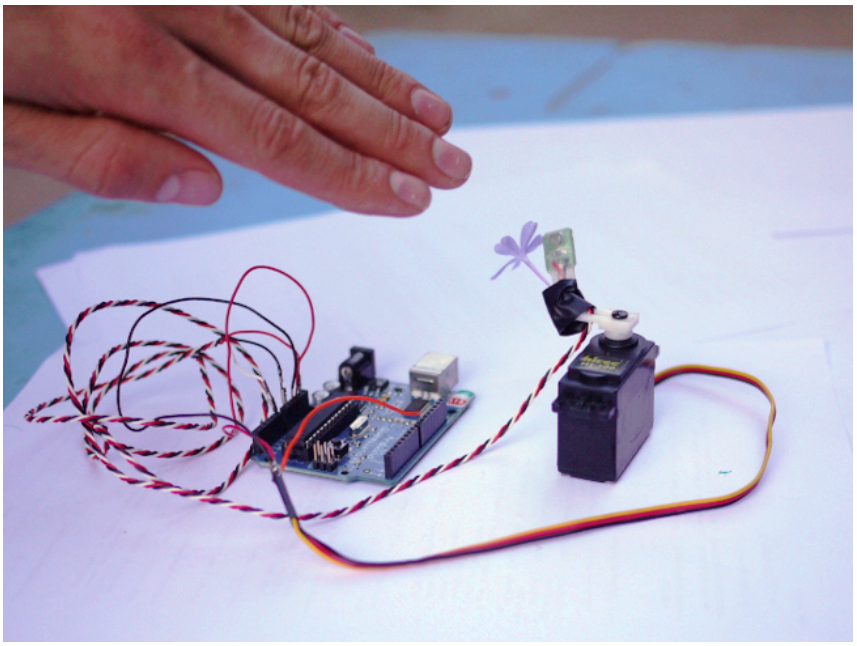
En fait, la broche VIN de l'Arduino permet d'utiliser une source de courant alternative à celui fourni par le câble USB. Cette source de courant peut généralement varier entre 6 et 12 V selon les types de cartes.



* Il est important de noter que pour les types de cartes beaucoup plus anciens comme la *Diecimela*, il faut également modifier le sélecteur de tension sur la carte (un *jumper*, en anglais) pour qu'elle accepte le courant extérieur (*ext* sur la carte) plutôt que celui du port USB (*usb* sur la carte).

XIX - La petite bête qui a peur

Ce projet vous permettra d'utiliser l'information provenant d'une cellule photo-résistive fixée à un servomoteur afin de créer une « petite bête » simple qui a peur lorsqu'on s'en approche. La résistance électrique d'une cellule photo-résistive varie selon la lumière qu'elle reçoit. Judicieusement placée sous une lampe, cette petite « bête » pourra ainsi « détecter » lorsqu'une main s'en approche, puisque cette dernière fera de l'ombre sur le capteur, la cellule. Une commande sera ensuite envoyée au servomoteur afin de faire reculer la cellule qui y est attachée.



XIX-A - Éléments nécessaires pour réaliser ce projet

• une cellule photo-résistive ; • une résistance de 10 k ; • un servomoteur.	• deux pinces crocodiles ; • quelques fils de connexion.

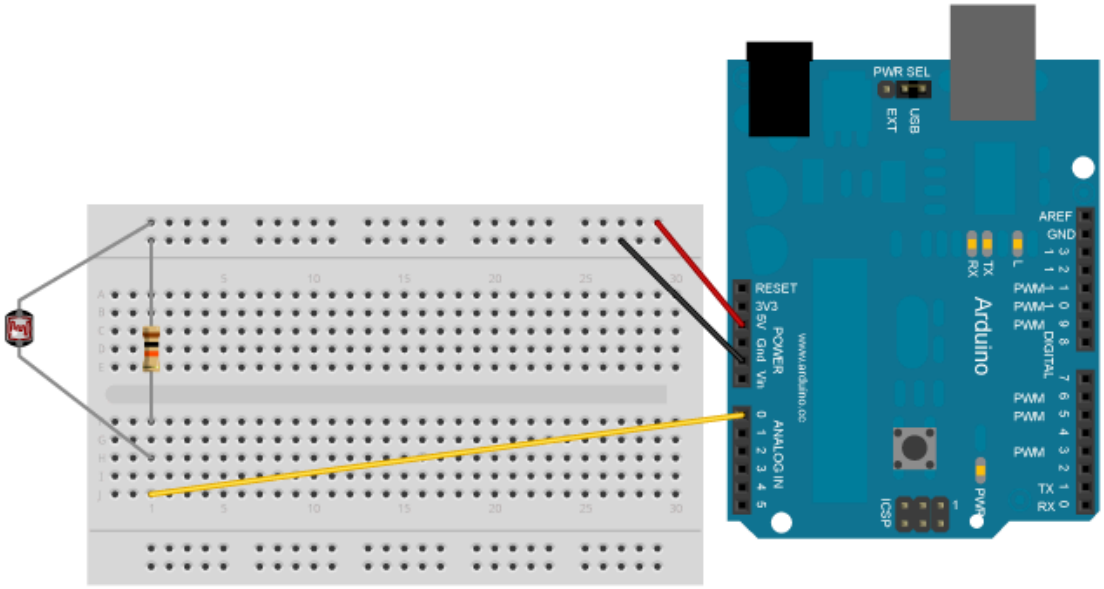
Si vous n'avez pas accès à un servomoteur, il vous est toutefois possible de réaliser la première partie de ce projet afin de comprendre comment analyser l'information entrante de la cellule photosensible.

XIX-B - Première étape

Dans cette première étape, nous travaillerons sur la partie « capteur » de ce projet. Nous connecterons la cellule photosensible à l'Arduino et afficherons l'information qui y est associée.

XIX-B-1 - Schéma et montage

Voici le schéma que nous allons réaliser :



Pour réaliser ce montage, il vous faut d'abord fixer la cellule sur la tête du servo-moteur, à l'aide de papier collant ou alors en passant les pattes dans les petits trous de la tête du moteur. Ensuite, en utilisant des pinces crocodiles, branchez une des pattes de la cellule sur la *pin* de 5 V de l'Arduino, et l'autre sur la *pin Analog In 0*. Pour compléter le montage de la partie « capteur », il faut connecter la patte de la résistance 10 k à la *pin* Analog In 0, et l'autre patte à la *pin GND* de l'Arduino. Ce montage électronique correspond en fait à un pont diviseur de tension (voir chapitre « **Les bases de l'électronique** »).

XIX-B-2 - Programme

Copiez maintenant le programme suivant dans la fenêtre de programmation Arduino.

```
// **********************
// définition des variables

int pinCapteur = 0; // variable pour désigner quelle pin de l'Arduino est utilisée pour le capteur

int valeurCapteur = 0; // variable pour contenir la valeur du capteur.
void setup()
{
    Serial.begin(9600); // établir la connexion sérielle à 9600 bauds
     pinMode(pinCapteur, INPUT); // définir la pin 0 comme une entrée (pinCapteur)
}

void loop()
{
    montrerValeurCapteur();
    delay(20); // laisser un cours délai pour éviter le trop-plein d'information
}

void montrerValeurCapteur()
{
    valeurCapteur = analogRead(pinCapteur); // lire la pin analogique et mettre la valeur dans valeurCapteur
    Serial.println(valeurCapteur); // communiquer au moniteur sériel la valeur du capteur.
}
```

Sauvegardez maintenant votre programme et vérifiez-le en cliquant sur le bouton « Verify ». En cas d'erreur (indiquée dans la console) , vérifiez l'orthographe, la casse et la présence appropriée des points-virgules (;). Vous pouvez maintenant télécharger votre programme sur la carte en appuyant sur « Upload ».

Une fois le téléchargement terminé, vous pouvez faire apparaître la fenêtre du moniteur sériel en cliquant sur le bouton « Serial Monitor ». Assurez-vous que la vitesse de transfert des informations (*baudrate*) du moniteur sériel est identique à celle spécifiée dans le programme, soit 9600. Cette information se trouve dans le bas de la fenêtre et peut être changée.

Après quelques instants, vous devriez voir défiler des chiffres sur le moniteur sériel. Ces chiffres devraient moduler légèrement, et plus amplement lorsque de l'ombre est faite sur la cellule photosensible.

Voyons plus en détail les éléments du code.

XIX-B-2-a - Déclaration des variables

pinCapteur cette variable est utilisée pour référer, tout au long du programme, à la *pin* de l'Arduino où est branchée notre capteur, la cellule photosensible. Dans ce projet, nous utilisons la *pin analog 0*, et donc la valeur de pinCapteur est initialisée à 0 lorsqu'elle est déclarée. Si nous avions connecté le capteur à une autre *pin*, par exemple *analog 5*, la valeur de pinCapteur aurait été initialisée à 5.

valeurCapteur cette seconde variable contient la valeur qui sera lue sur la *pin* de l'Arduino correspondant à notre capteur.

XIX-B-2-b - Configuration (setup)

Deux instructions figurent dans la partie setup () de notre programme.

Serial.begin (9600); Cette fonction établit la connexion sérielle entre la carte Arduino et, dans notre cas particulier, l'interface logicielle Arduino. Le paramètre qui est donné à la fonction est *9600*, qui signifie que la connexion sérielle à établir doit rouler à 9600 bauds. Le choix de la vitesse dépend de l'application développée. Dans le cadre de cet exemple, une vitesse plus élevée n'est pas nécessaire.

pinMode (pinCapteur, INPUT); Cette fonction déclare la *pin* de la carte correspondant à pinCapteur (en l'occurrence, *0*), comme un entrée (*Input*), puisque nous utiliserons cette *pin* pour recueillir la variation de voltage engendrée par notre circuit comportant la cellule photorésistante.

XIX-B-2-c - Boucle principale (loop)

Notre boucle de programme principal est relativement simple. Il suffit de recueillir la valeur de notre capteur, de l'envoyer au logiciel Arduino par communication sérielle et d'attendre un peu, pour espacer les lectures.

montrerValeurCapteur (); Cette ligne d'instruction appelle la fonction que nous avons définie nous-mêmes et qui sera expliquée ci-après. En l'appelant de la sorte, toutes les instructions qui se trouvent dans la fonction montrerValeurCapteur () seront exécutées. Une fois l'exécution de la fonction terminée, le programme poursuivra pour exécuter la fonction suivante.

delay (20); Cette seconde fonction demande au programme de marquer une courte pause, pour espacer les lectures. Le paramètre qui est spécifié, la valeur *20*, indique que le délai a une durée de 20 millisecondes (on retrouve 1000 millisecondes dans une seconde).

XIX-B-2-d - Sous-routines (montrerValeurCapteur)

En plus des fonctions réservées par Arduino setup () et loop (), il est possible de définir soi-même des fonctions que l'on peut également appeler « sous-routines ». L'utilisation de sous-routines permet d'alléger et de mieux structurer le programme. Par exemple, lorsque la fonction principale loop () doit exécuter à plusieurs moments différents la même

série d'instructions, il est possible de mettre une fois ces instructions dans une fonction unique, en sous-routines, et d'y référer.

Également, l'utilisation de sous-routine permet une meilleure lisibilité du code principal et de regrouper sous une même appellation, ou une même fonction, une série d'instructions ayant une tâche commune. Par exemple, une série de 10 instructions permettant de faire clignoter une LED de manière non régulière pourraient être regroupées sous une fonction que l'on pourrait nommer faireClignoterLED (). Ensuite, dans le programme principal, pour faire exécuter cette série d'instructions, il ne suffirait plus qu'à appeler la fonction faireClignoterLED ().

montrerValeurCapteur () Dans le cadre de notre projet, la sous-routine que nous avons créée a été nommée montrerValeurCapteur. Il ne s'agit pas d'une fonction préexistante, et nous en avons choisi le nom (qui aurait pu être mangerDesFrites (), par exemple). Jetons un œil en détail à cette fonction :

```
1. void montrerValeurCapteur ()
2. {
3.     // instructions à exécuter ici
4. }
```

D'abord, le void sert à indiquer que notre fonction ne retourne aucune valeur (voir chapitre « **Programmer Arduino** »). Ensuite, nous indiquons le nom donné à la fonction suivi de parenthèses. Comme notre fonction n'utilise pas de paramètres ou n'a pas de variable ou valeur en entrée pour déterminer son fonctionnement, il ne se trouve rien entre les parenthèses. Finalement, nous écrirons les instructions à exécuter pour cette fonction entre les accolades :

```
valeurCapteur = analogRead (pinCapteur);
```

Celle ligne de code contient la fonction analogRead (). Celle-ci permet de lire la *pin* de la carte Arduino correspondant à la valeur de pinCapteur (0). Puisqu'il s'agit d'une entrée analogique, les valeurs possibles que analogRead () peut retourner se situent entre 0 et 1023, correspondant à un voltage entre 0 et 5 V. L'opérateur « = » permet d'assigner la valeur retournée par analogRead à notre variable valeurCapteur. Ainsi, et pour la suite du programme ou jusqu'à ce qu'il y ait un changement, valeurCapteur contiendra la valeur de la lecture du capteur.

```
Serial.println (valeurCapteur);
```

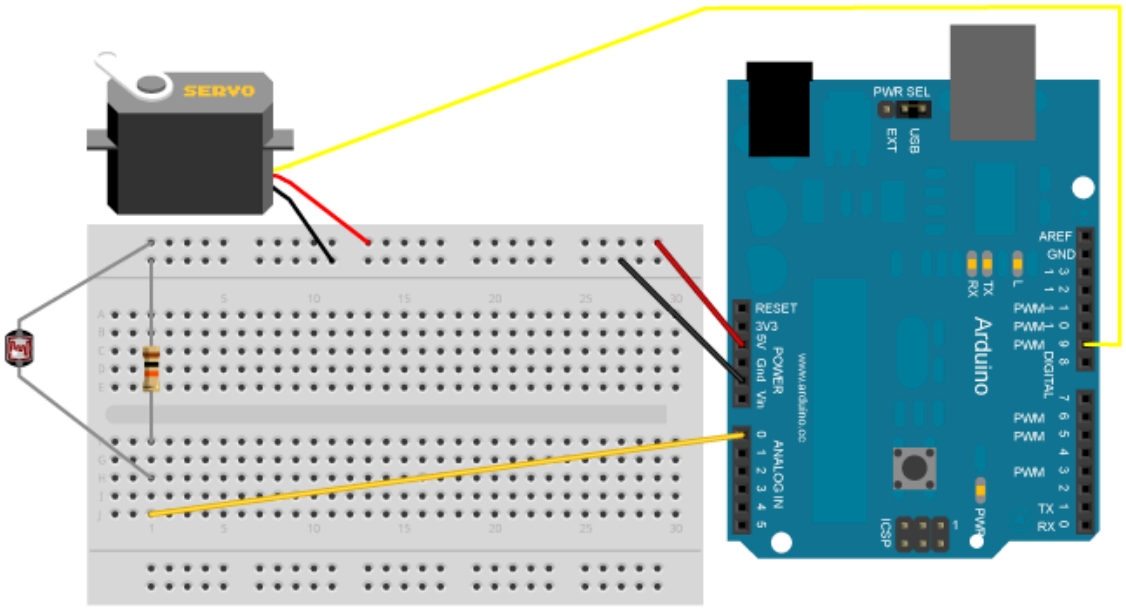
Notre deuxième instruction permet d'envoyer par communication sérielle la valeur qui est contenue dans notre variable valeurCapteur au logiciel Arduino, à travers le câble USB. La fonction Serial.println () comporte valeurCapteur comme paramètre, ce qui lui spécifie quoi imprimer. Serial.print () et Serial.println () se distingue par le fait que la deuxième fonction ajoute un retour de chariot (CR) dans son message, ce qui fait qu'à chaque envoi de valeur au moniteur sériel, par exemple, un changement de ligne sera effectué. Il est à noter que dans le cas de communication avec d'autres interfaces ou logiciels, il est parfois préférable d'utiliser Serial.print () pour n'envoyer que la donnée demandée, sans le caractère « CR » supplémentaire.

XIX-C - Deuxième étape

Dans cette deuxième étape, nous travaillerons sur la partie « moteur » de ce projet. Nous connecterons le servomoteur à l'Arduino et une valeur de position aléatoire lui sera envoyée afin de vérifier notre montage.

XIX-C-1 - Schéma et montage

Voici la suite du schéma que nous allons réaliser :



Pour réaliser ce montage, il vous faut connecter le servomoteur à la carte Arduino. Celui-ci comporte trois fils de couleurs habituellement rouge, noire et jaune (parfois blanche). Le fil noir doit être connecté à la *pin* GND, le fil rouge dans la *pin* 5 V (section power) de la carte et le fil jaune ou blanc doit être connecté à la *pin* 9 de la carte. Les fils rouge et noir servent à fournir l'alimentation au servomoteur. Le fil jaune sert à recevoir un signal PWM qui sera envoyé par l'Arduino afin de positionner le servomoteur.

Il est important de noter que pour ce projet, il n'y a qu'un seul moteur branché à la carte et utilisant la puissance 5 V qui provient du régulateur de la carte Arduino (qui l'obtient par le port USB). Dans une situation où vous seriez tenté de contrôler plusieurs moteurs, il serait nécessaire de concevoir un système de puissance supplémentaire (voir chapitre « **Précautions d'usage** »).

XIX-C-2 - Programme

Copiez maintenant le programme suivant dans la fenêtre de programmation Arduino.

```
#include <Servo.h> // inclure la librairie pour l'utilisation des servomoteurs

Servo servo1; // créer l'objet "servo1" pour référer à notre servomoteur
// **********************
// définition des variables

int pinCapteur = 0; // variable pour désigner quelle pin de l'Arduino est utilisée pour le capteur
int valeurCapteur = 0; // variable pour contenir la valeur du capteur.
int servoPos = 0; // variable pour contenir et spécifier la position en degrés (0-180)

void setup()
{
    Serial.begin(9600); // établir la connexion sérielle à 9600 bauds
    pinMode(pinCapteur, INPUT); // définir la pin 0 comme une entrée (pinCapteur)
    servo1.attach(9); // associer la PIN 9 au servomoteur
}

void loop()
{
    // montrerValeurCapteur();
    // delay(20); // laisser un cours délai pour éviter le trop-plein d'informations
    testerServoMoteur();
}

void testerServoMoteur()
```

```
26. {
27.     while (1==1)
28.     {
29.         servoPos = random(181); // attribuer une valeur aléatoire comprise entre 0 et 180 à
 servoPos
30.         servo1.write(servoPos); // spécifier la position au servomoteur
31.         delay(2000); // attendre 2 secondes
32.     }
33. }
34.
35. void montrerValeurCapteur()
36. {
37.     valeurCapteur = analogRead(pinCapteur); // lire la pin analogique et mettre la valeur
 dans valeurCapteur
38.     Serial.println(valeurCapteur); // communiquer au moniteur sériel la valeur du capteur.
39. }
```

Vous pouvez maintenant sauvegarder votre *sketch* sous un autre nom (afin de conserver le précédent) et télécharger le code sur la carte Arduino. Après quelque temps, le servomoteur devrait se mettre à bouger de manière aléatoire, changeant sa position toutes les deux secondes. Voyons plus en détail les éléments du code qui ont été ajoutés au code existant.

XIX-C-2-a - Inclusion de bibliothèque

La première ligne d'instruction de notre programme est maintenant changée. En effet, pour utiliser des servomoteurs avec l'Arduino, il est possible d'utiliser une bibliothèque (voir chapitre « **Bibliothèques externes** »). Pour inclure une lbibliothèque dans notre programme, nous utilisons la commande suivante :

```
#include <Servo.h>
```

le #include spécifie au programme d'inclure la bibliothèque dont le nom se situe entre les « <> ». L'usage spécifique de *Servo.h* nécessite la création d'un « objet » qui nous permettra de référer à certaines fonctions de la bibliothèque. Pour notre projet, nous intitulerons cet objet *servo1*. Encore une fois, ce nom est déterminé par le programmeur et aurait bien pu être *maurice*.

```
Servo servo1;
```

XIX-C-2-b - Déclaration des variables

servoPos Nous ajoutons à notre code la variable servoPos. Celle-ci contiendra la valeur de position spécifiée pour le servomoteur.

XIX-C-2-c - Configuration (setup)

servo1.attach (9); Une nouvelle instruction de configuration est ajoutée à notre boucle setup (). L'instruction *.attach () permet d'associer une *pin* de la carte Arduino à un objet *servo* créé plus haut. Ainsi, notre objet servo1 se voit associer la *pin* 9 par cette instruction.

XIX-C-2-d - Boucle principale (loop)

Notre boucle de programme principal doit maintenant changer. Nous voulons maintenant attribuer des positions aléatoires à notre moteur. Plutôt que de simplement supprimer les instructions existantes nous permettant de communiquer la valeur du capteur, nous les mettrons sous forme de commentaire. Pour ce faire, il suffit d'ajouter au début de l'instruction « // ».

```
// montrerValeurCapteur ();
```

```
// delay (20); // laisser un cours délai pour éviter le trop-plein d'informations
```

Cela rendra inactives les instructions, et au cas où nous voudrions les retrouver, il ne suffirait que de supprimer les « // ». Nous devons également maintenant ajouter une nouvelle fonction que nous définirons plus loin, TesterServoMoteur ();.

XIX-C-2-e - Sous-routines (testerServoMoteur)

Tout comme pour la fonction montrerValeurCapteur de l'étape 1, nous devons définir testerServoMoteur. Nous utilisons encore void en début d'instruction puisque la fonction ne retourne rien. Voyons les instructions qui figurent dans notre nouvelle fonction.

```
1. while (1==1)
2. {
3.     xxxx
4. }
```

La fonction while est une fonction conditionnelle qui permet de répéter une série d'instructions tant qu'une condition est testée comme *vraie*. Par exemple, nous pourrions avoir une condition où notre programme fait émettre un bruit d'alarme à l'Arduino *tant qu'un* bouton d'arrêt n'aurait pas été enclenché : while (bouton==pas enclenché) { faire sonner l'alarme }. La fonction est while (). La condition à tester est un paramètre qui se trouve entre les parenthèses, et la série d'instructions à répéter se trouve entre les accolades (là où se trouvent *xxxxx* dans notre exemple plus haut).

Dans notre projet, la fonction while nous sert plutôt ici à créer une boucle infinie, donc qui se répète à tout coup. La condition étant testée est 1==1. Puisque 1 sera toujours égal à 1, notre condition est toujours *vraie* et les instructions se répéteront à tout coup. Il est à noter que l'opérateur « == » est à distinguer du « = ». L'opérateur « == » sert à faire des comparaisons, et retournera une valeur logique, 1 correspondant à *vrai* et 0 à *faux*, tandis que le l'opérateur « = » sert à attribuer une valeur à une variable (dans x = 45, x prendra la valeur 45).

Les trois instructions contenues dans notre boucle while sont les suivantes :

```
1. servoPos = random (181);
2. servo1.write (servoPos);
3. delay (2000);
```

servoPos = random (181); La fonction random () retourne une valeur aléatoire. Celle-ci se situe entre 0 et le paramètre spécifié, de manière exclusive, ce qui signifie, par exemple, qu'un paramètre de 181 fera en sorte que la random () retourne une valeur entre 0 et 180. Comme la position que peut prendre notre servomoteur est comprise entre 0 et 180 en termes de degrés, nous générons dans cette instruction une valeur aléatoire appropriée.

servo1.write (servoPos); Tout comme pour xxx.attach (), la fonction xxxx.write () provient de la bibliothèque *Servo.h*. Elle nous permet ainsi de spécifier au servomoteur lié à l'objet servo1 la position en degrés qu'il doit atteindre. La variable servoPos fournit ici la valeur de cette position.

delay (2000); Afin de laisser un certain intervalle de temps entre chaque mouvement du servomoteur, nous ajoutons un délai de 2 secondes (2000 millisecondes).

XIX-D - Troisième étape

Dans cette troisième étape, nous établirons un lien entre la partie « capteur » et la partie « moteur ». Lorsqu'une main ou un doigt s'approche du capteur, il crée une ombre sur le capteur. Cette ombre devrait faire « fuir » le capteur fixé au servomoteur (si la cellule photo-résistive n'est pas fixée à la tête du moteur, vous devrez le faire maintenant).

Il n'y a pas de montage supplémentaire à exécuter. Le tout se joue au niveau du programme. Quelques ajustements aux valeurs seront peut-être nécessaires afin de « corriger » le comportement de la petite bête qui a peur, puisque

la flexibilité du système entraîne également une certaine instabilité (ce qui est recherché pour donner l'impression de la peur !).

XIX-D-1 - Programme

Copiez maintenant le programme suivant dans la fenêtre de programmation Arduino.

```
#include <Servo.h> // inclure la bibliothèque pour l'utilisation des servomoteurs

Servo servo1; // créer l'objet "servo1" pour référer à notre servomoteur

// **********************
// définition des variables

int pinCapteur = 0; // variable pour désigner quelle pin de l'Arduino est utilisée pour le capteur
int valeurCapteur = 0; // variable pour contenir la valeur du capteur.
int servoPos = 0; // variable pour contenir et spécifier la position en degrés (0-180)
int capteurBase = 0; // variable pour contenir la valeur de base du capteur (sans ombre)

void setup()
{
    Serial.begin(9600);// établir la connexion sérielle à 9600 bauds
    pinMode(pinCapteur, INPUT); // définir la pin 0 comme une entrée (pinCapteur)
    servo1.attach(9); // associer la PIN 9 au servomoteur
    determinerCapteurBase(); // trouver la valeur de base du capteur.
}

void loop()
{
    // montrerValeurCapteur();
    // testerServoMoteur();

    modifierPosSelonCapteur(); // lire la pin du capteur et mettre dans valeurCapteur.
    decroitreServoPos(); // faire décroître "la peur" de la petite bête.
    servo1.write(servoPos); // assigner la position du servomoteur qui a été changée
    delay(20);
}

void modifierPosSelonCapteur()
{
    int difference = 0; // déclarer variable locale pour calculer la différence

    valeurCapteur = analogRead(pinCapteur); // lire la pin du capteur et mettre dans valeurCapteur
    difference = capteurBase - valeurCapteur; // calculer l'écart entre valeur actuelle et la base
    difference = max(difference, 0); // limiter différence à 0 (pas de chiffre négatif)

    /* pour déboguer les valeurs et comprendre ce qui se passe
    Serial.print(valeurCapteur);
    Serial.print(" - ");
    Serial.print(capteurBase);
    Serial.print(" -- ");
    Serial.println(difference); // imprimer différence pour debug
    */

    servoPos = min ( servoPos + difference, 180); // modifier la valeur de position du servo
}

void determinerCapteurBase()
{
    servo1.write(0); // positionner le moteur à la position de départ
    delay(2000); // laisser le temps au moteur de se placer
    capteurBase = analogRead(pinCapteur); // assigner la lecture actuelle à capteurBase
}

void decroitreServoPos()
{
```

```
60.     servoPos = max ( servoPos - 10, 0); // faire décroître servoPos
61. }
62.
63. void testerServoMoteur()
64. {
65.     while (1==1)
66.         {
67.         servoPos = random(181); // attribuer une valeur aléatoire comprise entre 0 et 180 à
 servoPos
68.         servo1.write(servoPos); // spécifier la position au servomoteur
69.         delay(2000); // attendre 2 secondes
70.     }
71. }
72.
73. void montrerValeurCapteur()
74. {
75.     while (1==1)
76.         {
77.         valeurCapteur = analogRead(pinCapteur); // lire la pin analogique et mettre la valeur
 dans valeurCapteur
78.         Serial.println(valeurCapteur); // communiquer au moniteur sériel la valeur du
 capteur.
79.     }
80. }
```

Voici les ajouts faits à ce programme.

XIX-D-1-a - Déclaration des variables

capteurBase Nous ajoutons à notre code cette variable qui nous permettra de garder en mémoire la valeur de base du capteur. Celle-ci nous sera utile pour calculer l'écart entre la valeur mesurée en temps réel (donc affectée par l'ombre que crée le doigt ou la main qui s'approche) et celle de base, sans ombre.

XIX-D-1-b - Configuration (setup)

determinerCapteurBase () Il s'agit d'une nouvelle fonction créée pour initialiser les éléments de base de notre projet. Comme elle ne doit s'exécuter qu'une seule fois, la fonction est placée dans la configuration du programme.

XIX-D-1-c - Boucle principale (loop)

Notre boucle de programme principal doit encore une fois changer. Nous mettons d'abord sous commentaires les fonctions liées à la prise d'information et au test moteur, soit montrerValeurCapteur () et testerServoMoteur (). Ensuite, nous ajoutons les fonctions qui nous servirons à analyser le capteur et modifier la position du servomoteur en fonction de celui-ci, soit modifierPosSelonCapteur () et decroitreServoPos ().

Finalement, nous ajoutons l'instruction servo1.write (servoPos) afin de spécifier à l'objet servo1 la position telle qu'elle a été modifiée par le programme. L'instruction du delay (20) permet un certain espacement entre les lectures et assignations moteur.

XIX-D-1-d - Sous-routines

Nous ajoutons trois sous-routines à notre programme :

```
1. void determinerCapteurBase()
2. {
3.     servo1.write(0);
4.     delay(2000);
5.     capteurBase = analogRead(pinCapteur);
6. }
```

Cette fonction sert d'abord à initialiser la position du moteur avec servo1.write (0). Ensuite, il faut faire attendre le programme deux secondes avec delay (2000) pour laisser le temps au moteur de se positionner. Finalement, le programme fera une lecture du capteur avec analogRead (pinCapteur) pour assigner cette valeur à la variable capteurBase. Nous utiliserons cette variable pour calculer l'écart entre une lecture « sans ombre » et une lecture avec l'ombre projetée par la main ou le doigt.

```
1. void modifierPosSelonCapteur()
2. {
3.     int difference = 0;
4.     valeurCapteur = analogRead(pinCapteur);
5.     difference = capteurBase - valeurCapteur;
6.     difference = max(difference, 0);
7.     servoPos = min ( servoPos + difference, 180);
8. }
```

Cette fonction a la particularité de contenir une déclaration de variable, l'instruction int difference = 0. Habituellement réservée à la première section du programme, une déclaration de variable faite à l'intérieur d'une sous-routine crée ce qu'on appelle une variable *locale*. Cela signifie que la variable « n'existera » que dans le contexte d'exécution de la fonction ou sous-routine dans lequel elle se situe. Par conséquent, aucune autre fonction ou sous-routine ne pourrait faire usage de la variable difference. Par opposition, les variables qui sont déclarées au tout début du programme sont considérées comme *globale*, c'est-à-dire qu'elles « existent » et sont accessible à toutes les fonctions ou autre routines.

La seconde instruction de cette sous-routine assigne à valeurCapteur la lecture de la *pin* liée au capteur. Ensuite, nous calculons la différence entre la valeur du capteur « sans ombre » contenu dans capteurBase et celle « avec ombre » (ou sans) de valeurCapteur.

Finalement, comme il est possible que la différence nous retourne une valeur négative (dans le cas où il y aurait eu une illumination, par exemple, nous limitons la valeur de différence à un minimum de 0. Bien que cela semble contre-intuitif, nous utilisons dans ce cas la fonction max (). Celle-ci prend en paramètre deux valeurs (en l'occurrence difference et *0*) et retournera celle des deux qui sera la plus élevée. Ainsi, si difference = -5, max (difference, 0) retournera 0, puisque ce chiffre est plus élevé que -5. Par contre, si difference = 10, la fonction retournera 10. Nous assignons la valeur retournée par max () à difference.

La dernière instruction de cette sous-routine nous sert à incrémenter, si c'est le cas, la valeur de servoPos. Nous assignons ainsi à servoPos sa valeur actuelle en y ajoutant la différence calculée précédemment : servoPos + difference. La raison pour laquelle nous ajoutons la différence à la valeur actuelle est pour conserver une certaine continuité dans la position du moteur et une accumulation de la valeur de celle-ci à travers le temps. Ce code, puisqu'il prend en compte la différence, fait en sorte qu'un mouvement brusque, traduit comme une grande différence entre les deux lectures, augmentera grandement la position du servomoteur, générant une impression de « peur ». À l'inverse, une approche très « douce » et lente représentant une faible différence n'aura qu'un impact mineur sur la position du servomoteur, qui semblera plus « calme ».

Comme il n'est pas possible de donner une valeur supérieure à 180 au servomoteur (ou à la fonction servo1.write ()), nous devons limiter l'incrémentation de servoPos. Pour ce faire, nous utilisons la fonction min () qui s'exécute à l'inverse de max (). Ainsi, elle prendra le plus bas des deux chiffres en paramètres pour le retourner. Donc, dans min (servoPos + difference, 180), si l'addition de servoPos et différence est supérieure à 180, le chiffre 180 sera retourné, autrement ce sera le résultat de l'addition qui sera pris en compte.

```
1. void decroitreServoPos ()
2. {
3.     servoPos = max (servoPos - 10, 0);
4. }
```

La dernière sous-routine que comporte notre programme permet au servomoteur de retourner vers son point de base, le degré 0. Comme servoPos s'incrémente graduellement avec les ombres, il faut également doter notre petite bête d'une « tendance » à revenir, ou dit autrement, à faire décroître servoPos. L'instruction qui accomplit cela est servoPos = servoPos - 10. La valeur 10 est soustraite de servoPos et le résultat est stocké dans la même variable. Le choix de « 10 » ici est plus ou moins arbitraire et peut être ajusté, dans la mesure où une valeur plus faible (3,

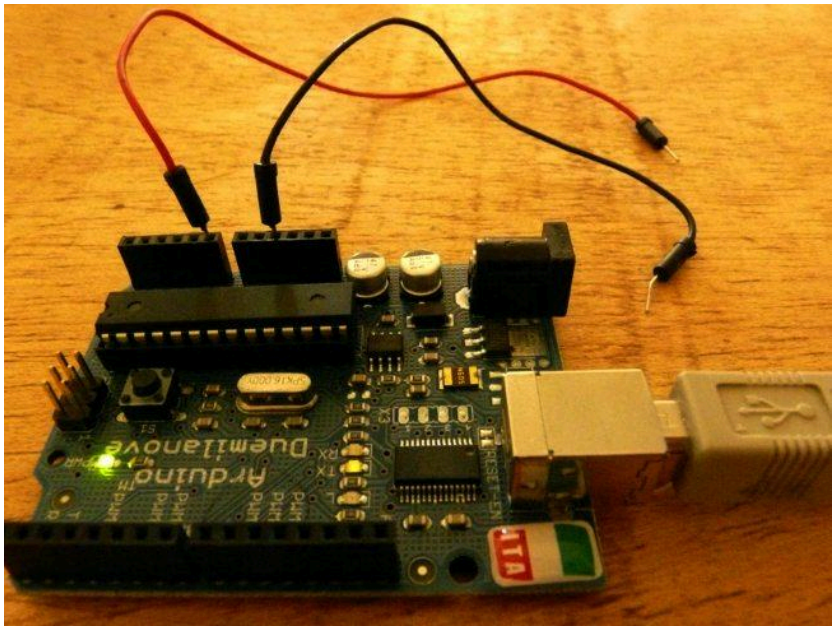
par exemple) fera décroître beaucoup plus lentement servoPos et une valeur plus élevée fera revenir à l'inverse la petite bête beaucoup plus rapidement.

Nous devons également nous assurer de ne pas aller en dessous de 0, puisqu'il s'agit de la position en degrés du servomoteur. Il faut donc utiliser, comme décrit précédemment, la fonction max ().

XX - Oscilloscope

Ce projet va vous permettre de réaliser un oscilloscope minimaliste et d'expérimenter la communication série avec un autre logiciel que celui d'Arduino, en l'occurrence, Processing. Un oscilloscope permet de visualiser les variations d'une tension dans le temps, sous forme d'une courbe. Le principe de fonctionnement de ce projet est le suivant :

- l'Arduino mesure une tension sur une entrée analogique ;
- il transmet la valeur de cette entrée à l'ordinateur via la communication série à travers le port USB ;
- le programme Processing récupère la donnée transmise et l'affiche sous forme de courbe.



XX-A - Précautions

L'oscilloscope ainsi réalisé ne sera capable de mesurer **que des tensions comprises entre 0 et 5 V**. Pour mesurer des tensions plus importantes, ou des courants, il sera nécessaire d'ajouter des circuits supplémentaires comme un pont diviseur de tension, ou une résistance (voir chapitre « **Les bases de l'électronique** »).

XX-A-1 - Éléments nécessaires

• une carte Arduino ; • deux fils.	• Le logiciel Processing (17) , logiciel libre de programmation graphique, sera nécessaire pour créer l'interface de visualisation. Vous pouvez parcourir le *flossmanual* (18) de ce logiciel. Toutefois, vous serez guidé pour réaliser ce projet sans avoir besoin de l'étudier.

XX-A-2 - Montage électronique

Comme expliqué ci-dessus, le montage se résume à deux fils connectés à l'Arduino, qui vont servir à mesurer un autre montage soumis à une tension variable. Un des fils sera connecté a la pin « GND », qui correspond au 0 V. Le second sera connecté à la pin Analog in 0. Les extrémités libres vont servir à faire les contacts avec le circuit à mesurer.

XX-B - Première étape

Copiez le programme suivant dans Arduino.

```
1. /*
2. Ce programme consiste simplement à mesurer l'entrée analogique 0, et à transmettre le résultat
 via une communication série.
3. Il ressemble beaucoup au programme décrit dans la première étape du projet "la petite bête qui
 a peur".
4. */
5. //déclaration des variables
6. //stockage du numéro de pin où sera branché l'entrée de la mesure
7. int PinDeMesure=0;
8. //conteneur pour stocker la mesure
9. int tension=0;
10. // délai entre deux mesures
11. int periodeDeMesure=20;
12.
13. void setup(){ // Initialisation du programme
14.     // établir la connection série à 19200 baud
15.     Serial.begin(19200);
16.     // définir le PinDeMesure comme une entrée
17.     pinMode(PinDeMesure, INPUT);
18. }
19.
20. void loop() { //boucle principale
21.     tension = analogRead(PinDeMesure); // Lit la tension du pin DeMesure et stocke le
 résultat dans la variable tension
22.     Serial.println(tension); // communique la valeur de la tension au port série
23.     delay(periodeDeMesure); //attendre jusqu'à la prochaine mesure
24. }
```

Après avoir compilé et téléchargé ce programme dans la carte Arduino, en ouvrant le moniteur sériel, on pourra voir apparaître les mesures successives défiler. Les valeurs varient entre 0 et 1023 quand la tension aux pointes de touches varie entre 0 et 5 V.

XX-C - Deuxième étape

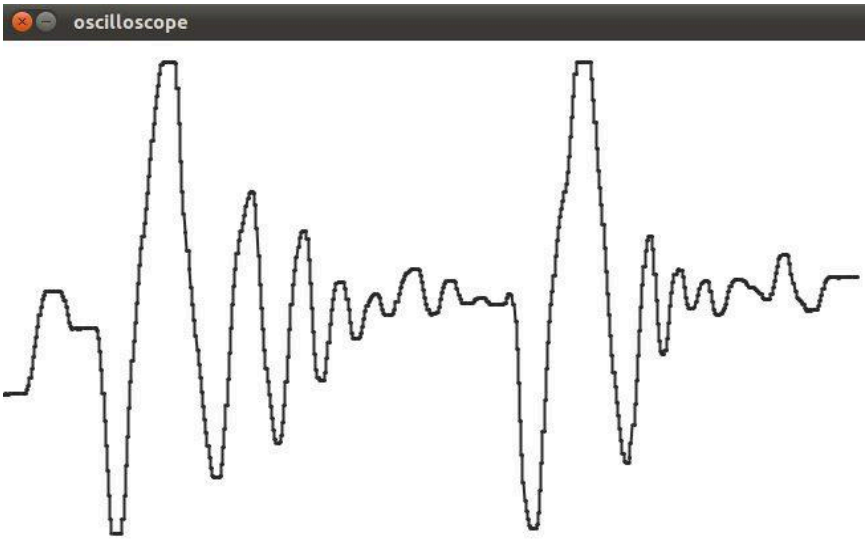
Après avoir téléchargé et installé Processing, copiez le programme suivant dans Processing. Vous ne devriez être pas trop dépaysé par l'interface, car Processing et Arduino sont des programmes frères.

```
1. /*
2. oscilloscope minimaliste avec un Arduino
3. récupère les valeurs via une liaison série
4. */
5. import processing.serial.*; // importation de la bibliothèque de communication séerie
6. //variables
7. Serial maConnection; // Crée un objet de communication série
8. int tension=0; // variable ou sera stockée la valeur de la tension
9. //affichage
10. int fenetreTemps=12000; // temps pour parcourir la largeur de l'écran en millisecondes
11. int x=0;
12. int y=0;
13.
14. void setup() {
15.     size(650, 400); // taille de l'écran
16.     String NomDuPort = Serial.list()[0]; // récupère la première interface série trouvée
```

```
17.     println("connection a "+NomDuPort);
18.     maConnection = new Serial(this, NomDuPort, 19200); // création de la connexion série
19.     background(255); // fond d'écran blanc
20.     smooth(); // lisser les dessins
21.     strokeWeight(2); // largeur de trait
22.     stroke(40); // couleur du trait gris
23. }
24. 
25. void draw() { //boucle de dessin principale
26.     //détermination du X actuel
27.     int oldx = x;
28.     x = (millis() % fenetreTemps) * width/fenetreTemps;
29.     if (oldx > x) {
30.         // reprise au début de l'écran
31.         oldx = 0; background(255);
32.         // fond d'écran blanc
33.     }
34.     //determination de la valeur de Y
35.     int oldy = y;
36.     y = int(map(tension, 0, 1023, 380, 20)); // mise à l'échelle de la tension pour entrer
 dans l'écran
37.     line(oldx, oldy, x, y); // dessine le trait
38. }
39. 
40. void serialEvent (Serial maConnection) { // si des données arrivent par la connexion série
41.     String retour=maConnection.readStringUntil('\n'); // lit la donnée jusqu'à la fin de
 ligne
42.     if (retour != null) { //si le retour n'est pas vide
43.         retour = trim(retour); // enlever les espaces
44.         tension = int(retour); // converti le texte en nombre entier
45.     }
46. }
```

Pour exécuter le programme, il suffit de cliquer sur le bouton « play ». Vérifiez auparavant que **le moniteur sériel du programme Arduino est bien fermé**, sans quoi la communication avec Processing ne voudra pas s'initialiser. En effet, la communication sérielle avec l'Arduino est exclusive, ce qui signifie qu'un seul programme à la fois peut ouvrir et communiquer sur un port sériel avec l'Arduino. En fermant le moniteur sériel, la connexion sérielle est fermée et ainsi libre pour une autre application. Il est possible d'utiliser plusieurs ports sériels différents (notamment avec les pins digitales de l'Arduino) pour établir plusieurs connexions avec différents logiciels.

Le programme, en s'exécutant affichera la fenêtre suivante :



La courbe affichée dépend du circuit que vous mesurez avec l'oscilloscope. Si vous n'avez rien de spécial à mesurer, vous pouvez essayer avec un potentiomètre, en prenant la mesure sur sa patte du milieu, et en reliant les deux autres bornes respectivement au GND et au 5 V.

XX-D - Pour aller plus loin

Dans ce projet, vous avez compris comment transmettre une information de l'Arduino à un programme quelconque sur l'ordinateur. Vous avez réalisé un magnifique oscilloscope. On peut imaginer compliquer ce programme pour avoir :

- accès à plus d'entrées simultanément ;
- mémoriser les enregistrements de courbes ;
- faire des pauses ;
- changer les échelles d'affichage ;
- etc.

Ces améliorations peuvent prendre du temps. En fait, grâce à la magie du logiciel libre, une âme charitable a déjà effectué tout ce travail, et l'a partagé. Il s'agit du projet arduinoscope : **http://code.google.com/p/arduinoscope/**. Peut-être **êtes vous déçu d'avoir travaillé pour rien ?** En fait, cet exemple nous montre également qu'avant d'entamer un projet, il est toujours utile de se documenter sur Internet pour voir les réalisations déjà existantes autour de la même problématique ; inutile de réinventer la roue !

XXI - Perroquet

Dans ce projet nous allons réaliser un système qui permet d'enregistrer et répéter des séries d'impacts effectués sur un objet. Pour cela nous allons utiliser un capteur piézoélectrique qui permet de mesurer une vibration qui se propage dans un matériau. Pour information, ce capteur peut aussi servir de petite enceinte pour produire des sons. Pour afficher l'information enregistrée par le programme nous utiliserons une LED.

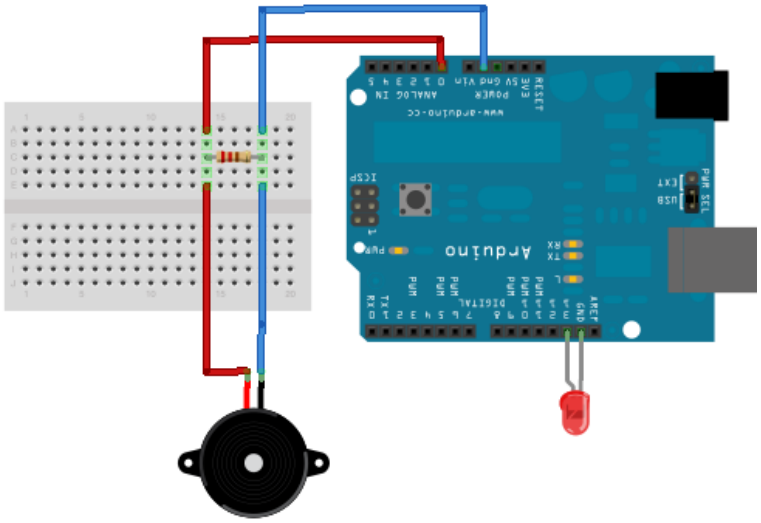
XXI-A - Éléments pour nécessaire

- une carte Arduino
- un piézo
- une LED
- une résistance entre 500 - 1 Mega Ohm
- un peu de ruban adhésif.

XXI-B - Première étape

Pour commencer, votre piézo doit être nu, et la pastille métallique doit être fixée sur un support tel qu'une table avec un bout de ruban adhésif qui le maintient en pression contre votre support. Nous devons ensuite choisir notre résistance pour régler la nervosité de la réponse de notre capteur. Il faut savoir que plus la résistance utilisée est grande, plus le capteur produira une réponse lente, à l'inverse, plus la résistance est faible plus le capteur nous donnera une réponse rapide. Pour bien comprendre ces notions nous allons réaliser le montage de notre piézo et afficher les valeurs captées sur notre ordinateur. Cette étape nous permettra de bien choisir notre résistance ainsi que de bien comprendre l'utilisation de ce capteur.

XXI-B-1 - Schéma et montage



XXI-B-2 - Programme

```
1. ////////// Déclaration des variables
2. const int CAPTEUR = A0; // le piézo est connecté à la broche Analogue 0
3. int lectureCapteur = 0; // variable pour stocker la valeur du capteur
4.
5. ////////// Initialisation
6. void setup() {
7.     Serial.begin(9600); // configurer le port série
8. }
9.
10. ////////// Boucle principale
11. void loop() {
12.     // lecture de la valeur du piézo
13.     lectureCapteur = analogRead(CAPTEUR);
14.     // envoi de la valeur à l'ordinateur
15.     if (lectureCapteur > 0) // fera s'afficher toute valeur supérieure à zéro
16.     {
17.         Serial.println(lectureCapteur, DEC);
18.     }
19. }
```

XXI-C - Deuxième étape

Nous devons maintenant introduire au code deux nouvelles fonctions. La première nous permettra de déterminer à partir de quelle valeur analogique captée nous considérons que le capteur détecte un impact. Cette fonction est un seuil qui nous permettra de mesurer le temps écoulé entre chaque impact. La deuxième fonction (**ANTIREBOND**) que nous devons introduire est une fonction de filtrage qui nous permettra d'éviter que le programme enregistre plusieurs impacts lorsqu'un seul impact est donné. Pour mesurer le temps, nous utiliserons la fonction millis() qui nous permet de connaître à tout moment le temps écoulé depuis le démarrage du programme.

```
1. ////////// Déclaration des variables
2. const int CAPTEUR = A0;      // le Piezo est connecté à la pin Analogue 0
3. int lectureCapteur = 0;      // variable pour stocker la valeur du capteur
4. const int LED = 13;          // LED connectée à la broche digitale 13
5. const int SEUIL = 1;         // choix du seuil de détection
6. const int ANTIREBOND = 600; // choix du filtre anti-rebond
7. long temps = 0;              // variable pour mémoriser le temps
8.
9. ////////// Initialisation
10. void setup() {
```

```
11.     pinMode(LED, OUTPUT); // définir la pin associée à LED comme sortie
12.     Serial.begin(9600);   // configurer le port série
13. }
14.
15. ////////// Boucle principale
16. void loop() {
17.     // lecture de la valeur du Piezo
18.     lectureCapteur = analogRead(CAPTEUR);
19.     // si la valeur captée est supérieure au seuil choisi et
20.     // que le temps écoulé depuis le dernier impact est
21.     // supérieur au temps de l'ANTI-REBOND,
22.     // alors on rentre dans la condition
23.     if (lectureCapteur >= SEUIL && millis() - temps >= ANTIREBOND) {
24.         // envoi de l'information à l'ordinateur
25.         Serial.println("impact");
26.         temps = millis(); // mise à jour du temps courant
27.     }
28.     else {
29.         // on ne fait rien
30.     }
31. }
```

XXI-D - Réglage du SEUIL et du filtre ANTIREBOND

Pour que votre programme fonctionne correctement il vous faut ajuster les constantes SEUIL et ANTIREBOND.

```
const int SEUIL = 1;         // choix du seuil de détection
const int ANTIREBOND = 600; // choix du filtre anti-rebond
```

La constante SEUIL vous permet de supprimer les parasites éventuellement présents dans le signal capté. Plus la valeur de cette constante est élevée, plus il faudra taper fort pour déclencher l'enregistrement de l'impact.

La constante ANTIREBOND définit un temps durant lequel la lecture de l'entrée analogique n'est pas active. Ce temps permet de filtrer le signal d'entrée, car le capteur peut produire plusieurs signaux pour un seul impact. L'idée est donc d'enregistrer le premier signal et de fermer la lecture pendant un court instant pour ne pas enregistrer les signaux résiduels.

XXI-E - Les fonctions logiques (et) et (ou)

Dans le programme ci-dessus nous avons utilisé un opérateur logique &&. Cet opérateur nous a permis d'introduire plusieurs conditions à la fonction if(), puisque nous voulions que TOUTES les conditions soient *vraies* pour que les instructions s'exécutent. Seul (*vrai && vrai*) retourne une valeur *vraie*, alors que (*vrai && faux*) et (*faux && vrai*) retournent tous deux *faux*.

Ci-dessous l'opérateur logique **&&** (et) nous permet de rentrer dans la fonction if() seulement si les deux conditions sont *vraies*.

```
1. if (lectureCapteur >= SEUIL && millis () - temps >= ANTIREBOND) {
2.     ...
3. }
```

Ci-dessous l'opérateur logique **||** (ou) nous permet de rentrer dans la fonction if() si l'une ou l'autre des conditions est *vraie*.

```
1. if (marqueurTemps [posTableau] == 0 || posTableau == NOMBREMARQUEURS) {
2.     ...
3. }
```

XXI-F - Troisième étape

Dans cette étape nous allons implémenter un mécanisme qui va nous permettre de formaliser notre scénario d'utilisation qui comporte deux modes différents : le MODE ÉCOUTE et le MODE JEUX. Cette troisième étape introduit aussi plusieurs astuces pour que le programme puisse fonctionner dans son ensemble, notamment un concept permettant de définir les conditions de passage d'un mode à un autre ainsi que l'ajout d'une LED facilitant la visualisation les impacts enregistrés puis rejoués.

```
1. ////////// Déclaration des variables
2. // ces variables sont constantes
3. const int LED = 13;              // LED connectée à la pin digitale 13
4. const int CAPTEUR = A0;          // Piezo connecté à la pin Analogue 0
5. const int SEUIL = 1;             // seuil de détection du capteur
6. const int ANTIREBOND = 200;      // filtre anti-rebond
7. const int FIN = 2000;            // temps pour définir la fin du mode écoute
8. const int NOMBREMARQUEURS = 50; // nombre maximum de marqueurs temps
9.
10. // ces variables changent
11. int lectureCapteur = 0;              // valeur du capteur
12. int ledEtat = LOW;                   // état de la LED
13. int mode = 1;                        // mode écoute (1) ou jeux (2)
14. int posTableau = 0;                  // Position dans le tableau
15. long temps = 0;                      // variable pour mémoriser le temps
16. long marqueurTemps[NOMBREMARQUEURS]; // tableau pour mémoriser les temps qui séparent chaque
 impacte
17. int i = 0;                           // variable pour parcourir la liste
18.
19. boolean DEBUG = false;       // activer ou non le DEBUG
20. boolean BOUCLE = false;      // TODO
21.
22. void setup() {
23.     pinMode(LED, OUTPUT);     // déclarer la pin en sortie
24.     Serial.begin(9600);       // configurer le port série
25. }
26.
27. ////////// Boucle principale
28. void loop() {
29.     switch(mode) {
30.     case 1: // mode enregistrement
31.         if(DEBUG == true) Serial.println("mode 1"); // envoi du mode
32.         // lecture de la valeur du capteur Piezo
33.         lectureCapteur = analogRead(CAPTEUR);
34.
35.         // si la valeur captée est supérieure au seuil choisi et
36.         // que le temps écoulé depuis le dernier impact est supérieur au temps de
 l'ANTIREBOND
37.         // alors on rentre dans la condition
38.         if (lectureCapteur >= SEUIL && millis() - temps >= ANTIREBOND) {
39.             marqueurTemps[posTableau] = millis() - temps; //
40.             digitalWrite(LED, HIGH);  // activer la broche de la LED
41.             delay(5);                 // attendre 10 millisecondes
42.             digitalWrite(LED, LOW);   // éteindre la broche de la LED
43.             // on mémorise le temps écoulé entre chaque impact
44.             posTableau++;             // on incrémente la position
45.             temps = millis();         // initialisation
46.         }
47.
48.         // si le temps écoulé depuis le dernier impact est supérieur
49.         // au temps dit de FIN et que plus de deux impacts sont enregistrés
50.         // alors on rentre dans la condition
51.         if (millis() - temps >= FIN && posTableau >= 2) {
52.             posTableau = 0;             // initialiser la position
53.             mode = 2;                   // choisir le mode 2
54.             if (DEBUG == true) Serial.println("mode 2"); // envoi du mode
55.             temps = millis();           // initialisation du temps
56.         }
57.
58.         break;                          // sortir du mode 1
59.
```

```
60.     case 2: // mode de jeux
61.         // lecture de la valeur du capteur Piezo
62.         lectureCapteur = analogRead(CAPTEUR);
63.         // si la valeur du capteur est supérieure au seuil
64.         // alors on entre dans la condition puisque l'usager rejoue
65.         if (lectureCapteur >= SEUIL) {
66.             // on efface toutes les valeurs du tableau
67.             for (i=0; i<NOMBREMARQUEURS; i++) {
68.                 marqueurTemps[i] = 0;
69.             }
70.             posTableau = 0;     // initialiser la position
71.             mode = 1;           // choisir le mode 1
72.             if (DEBUG == true) Serial.println("mode 2 stop"); // envoi du mode
73.             temps = millis();   // initialisation du temps
74.             break;              // sortir du mode 2
75.         }
76.
77.         // si la valeur de temps stockée dans le tableau n'est pas nulle
78.         // et que cette valeur de temps est dépassée par le conteur millis()
79.         // alors on entre dans la condition
80.         if ( marqueurTemps[posTableau] != 0 && millis() - temps >=
 marqueurTemps[posTableau] ) {
81.             digitalWrite(LED, HIGH);    // activer la broche de la LED
82.             delay(5);                   // attendre 5 millisecondes
83.             digitalWrite(LED, LOW);     // éteindre la broche de la LED
84.             posTableau++;               // on incrémente la position
85.             temps = millis();           // initialisation du temps
86.         }
87.
88.         // si la valeur de temps stockée dans le tableau est nulle
89.         // ou que nous arrivons à la fin du tableau
90.         // alors on entre dans la condition
91.         if ( marqueurTemps[posTableau] == 0 || posTableau == NOMBREMARQUEURS ) {
92.             for (i=0; i<NOMBREMARQUEURS; i++) { // on efface toutes les valeurs du tableau
93.                 marqueurTemps[i] = 0;
94.             }
95.             posTableau = 0;     // initialiser la position
96.             mode = 1;           // changer de mode
97.             if(DEBUG == true) Serial.println("mode 2 end"); // envoi du mode
98.             temps = millis();   // initialisation du temps
99.         }
100.         break;                 // sortir du mode 2
101.     }
102. }
```

Ce programme peut sembler complexe et comporte une multitude de fonctions. Pour démystifier ces mécanismes, voici quelques explications supplémentaires.

XXI-G - Les tableaux

Nous allons tout d'abord revenir sur les tableaux. Nous avons utilisé un tableau nommé marqueurTemps pour stocker les valeurs de temps qui séparent chacun des impacts. Notre tableau est un conteneur qui permet de stocker une liste de variables. Pour utiliser un tableau, nous devons tout d'abord le déclarer au même niveau que les variables (voir chapitre « **Programmer Arduino** »). Ensuite nous définissons le nombre de variables qu'il comporte avec la constante NOMBREMARQUEURS.

```
const int NOMBREMARQUEURS = 50; // nombre maximum de marqueurs temps
long marqueurTemps [NOMBREMARQUEURS]; // tableau pour mémoriser les temps
```

Pour écrire une valeur dans une case du tableau nous devons utiliser deux variables, soit une pour choisir la position dans le tableau où nous souhaitons enregistrer celle-ci (posTableau) et une pour la valeur à inscrire (val). Nous allons ici écrire la valeur 127 dans la deuxième case du tableau :

```
1. int posTableau = 2; // initialisation de posTableau à 2
2. long val = 127; // initialisation de val à 127
3.
```

```
4. marqueurTemps [Position] = valeur;
```

Pour accéder à la valeur que nous avons stockée dans la deuxième case du tableau, nous devons encore utiliser deux variables, soit une variable de position (posTableau) et une variable pour contenir la valeur que nous souhaitons lire (val).

```
1. int posTableau = 2; // initialisation de posTableau à 2
2. int val = 0; // initialisation de val à 0
3.
4. val = marqueurTemps [posTableau];
```

La valeur contenue dans val est maintenant égale à 12.

XXI-H - Fonction DEBUG

Quand vous devez écrire un programme, il est bon de laisser les lignes de code qui vous ont permises de déboguer votre programme. Cependant comme ces lignes de code utilisent des ressources processeur pour transmettre les informations sur le port série, il faut pouvoir les désactivées quand le programme doit fonctionner normalement. Pour cela vous pouvez utiliser des variables booléennes qui vous permettront d'activer ou pas le mode DEBUG du programme.

```
        if(DEBUG == true) Serial.println("mode 1"); // envoi du mode
```

Si la condition est écrite sur une seule ligne les accolades ne sont pas nécessaires.

XXI-I - Pour aller plus loin

Pour aller plus loin, vous pouvez peut-être améliorer l'actionneur piézo en le remplaçant par un électro-aimant qui vous permettra de produire un son acoustique (un percuteur mécanique). Cependant il faut faire attention à la consommation de l'électro-aimant qui demande plus de 20 mA. La démarche consiste alors à utiliser un transistor pour ne pas détériorer votre carte Arduino. Il est aussi possible d'implémenter une fonction boucle pour fabriquer un séquenceur mécanique…

XXII - Mise en œuvre

Pour réaliser votre projet à travers toutes ses étapes, une bonne planification est toujours de mise. Même si votre projet est appelé à changer en cours de route, savoir organiser ses idées et préparer son projet est essentiel pour la bonne réussite de ce genre d'entreprise.

XXII-A - Savoir décrire son projet

Savoir bien décrire son projet vous permettra de confronter vos idées avec d'autres praticiens qui pourront vous apporter de précieux conseils. Bien décrire son projet, c'est aussi l'occasion de distinguer les différentes parties qui le composent pour bien évaluer le travail à réaliser. Une description claire vous permettra encore de visualiser votre pensée et facilitera le travail de développement.

XXII-A-1 - De l'idée à la réalisation

Chacun d'entre nous possède des connaissances plus ou moins pointues en électronique et en programmation. Certains ont une sensibilité artistique et souhaitent acquérir des compétences techniques pour traduire leurs idées. D'autres souhaitent certainement acquérir des connaissances techniques pour pouvoir imaginer de nouveaux projets. Quand vous débutez avec Arduino, il est préférable de découvrir le domaine en partant de votre projet. Celui-

ci vous servira de fil conducteur pour aller glaner des ressources qui vous permettront de maîtriser de nouvelles connaissances.

Savoir traduire une idée ou un concept en des termes technologiques et informatiques n'est pas toujours facile. En énonçant vos idées de manière modulaire, vous commencerez peu à peu à détailler, par exemple, les différentes fonctions qui pourraient composer votre code ou les différentes parties qui constitueraient votre circuit électronique. Voici un petit guide en étapes pour vous aider :

XXII-A-2 - Concept et idéation

- Décrivez quelles sont vos intentions, de manière générale.
- Le projet : en quoi consiste-t-il ? Que doit-il évoquer ? Comment se comporte-t-il ?
- L'utilisateur : comment peut-il interagir avec l'objet ? Quelle est la durée ? Y'a-t-il un début et une fin ?
- Combien de personnes peuvent interagir/utiliser en même temps le projet ?

XXII-A-3 - Ajoutez de la structure à vos intentions

- Décrivez et énumérez tous les processus actifs du projet.
- Décrivez et faites une liste de toutes les entrées et sorties nécessaires.
- Faites une liste de matériel nécessaire.
- Faites un échéancier de production, quelle étape devra être complétée quand.

XXII-A-4 - Programmation : faites un modèle formel du projet

- Identifiez les variables et constantes.
- Identifiez les fonctions et boucles du programme.
- Identifiez les conditions et autres liens entre les variables.
- Liez tous ces éléments aux processus énumérés au point 2.

XXII-A-5 - Implémentez votre modèle

- Réalisez votre montage électronique.
- Écrivez le code.
- Relaxez-vous!

En appliquant ces règles simples, vous pourrez ainsi :

- Vous référer, lors de la programmation, à vos intentions et vérifier si vous les avez appliquées de manière adéquate.
- Plus facilement transférer certains éléments de votre projet vers un autre environnement de production, de programmation ou même d'autre circuits électroniques.
- Faire une liste et avoir une idée générale de vos tâches en programmation et en électronique ; de plus, cela constituera une bonne manière de demeurer cohérent dans votre démarche en testant d'abord sur papier ce que vous programmerez par la suite.

XXII-B - Implémentation

L'implémentation concerne autant le code que le matériel physique. Implémenter un concept, c'est le rendre concret.

XXII-C - L'outillage

Il vous faudra pour commencer à mettre en œuvre votre projet un minimum d'outils qui vous aideront dans chacune des étapes du projet. Les principaux outils sont les suivants : une platine d'essai, du fil conducteur, un fer à souder, une pompe à dessouder, des tournevis, une petite pince précelle et une pince coupante, une troisième main et de l'étain.

XXII-D - Où vous procurer votre Arduino ?

Le projet Arduino connaît aujourd'hui un fort succès et la plupart des magasins d'électronique proposent ce produit dans leur catalogue. En choisissant le magasin le plus proche de chez vous, vous éviterez de payer des taxes et frais de douanes. Dans certains pays, il est cependant plus facile de les commander sur Internet (voir chapitre « **Ressources en ligne** »). Le coût moyen d'une Arduino est de 25 €. Vous pouvez aussi regarder du côté des cartes alternatives.

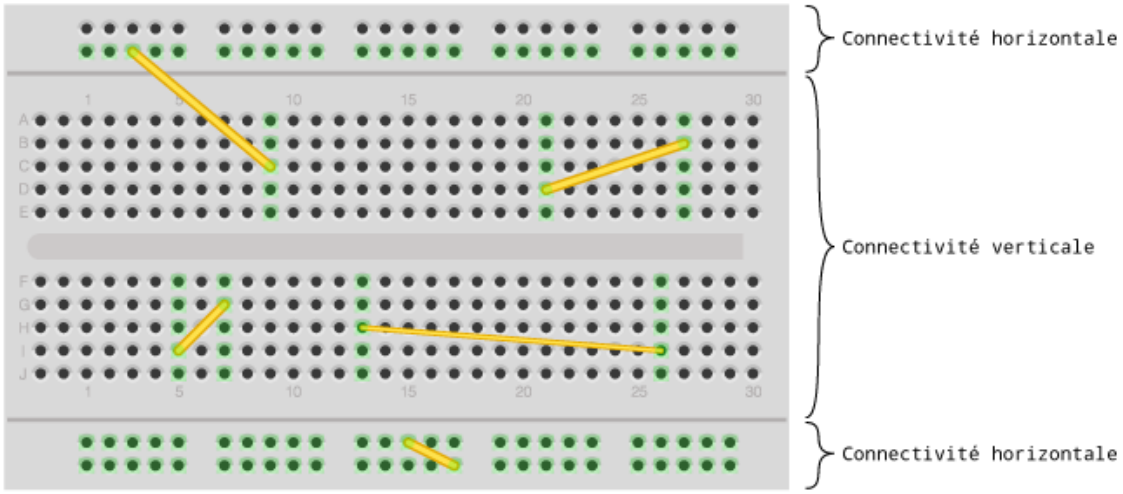
XXII-E - Les composants électroniques

Il vous faudra tout d'abord rassembler les composants dont vous avez besoin pour réaliser votre montage sur une platine d'essai. Cette étape sous-entend que vous ayez déjà défini une ou plusieurs solutions et souhaitez maintenant passer à la réalisation. il existe un grand nombre de fournisseurs de composants électroniques (cf. le chapitre Ressources externes dans la section Annexe)

Vous pouvez également récupérer de nombreux composants sur des pièces d'équipement usagées. De vieilles radios, imprimantes, téléviseurs et jouets foisonnent de pièces électroniques que vous pourrez utiliser dans vos projets (cf. D comme débrouille).

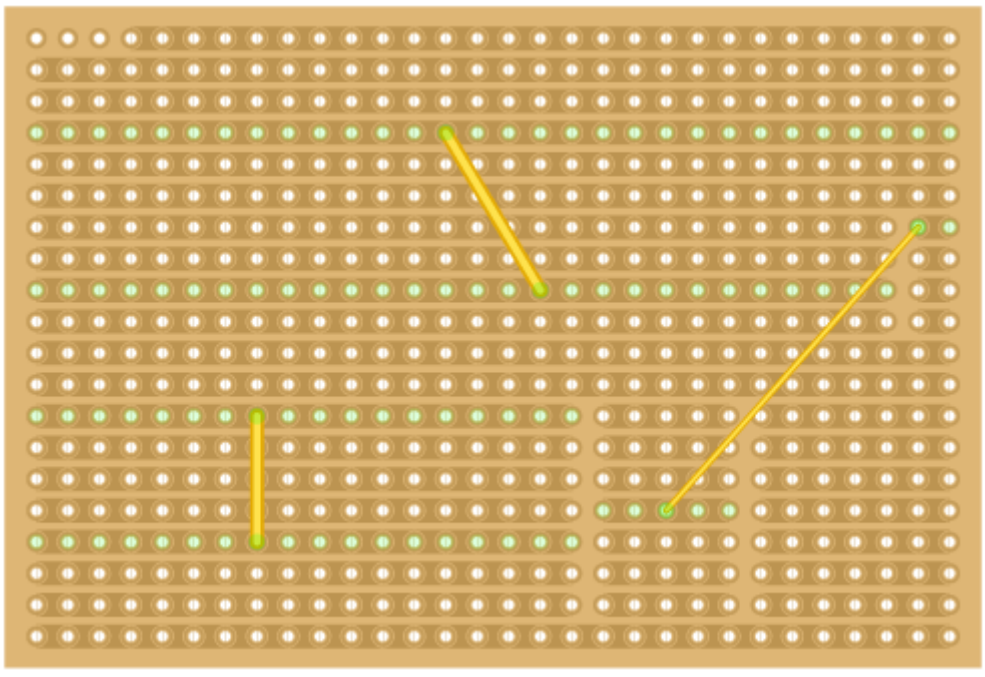
XXII-F - Le montage

Quand vous devez réaliser un projet avec Arduino, la première étape consiste à réaliser son montage sur une platine d'essai aussi appelée *breadboard*. Une platine d'essai est un support qui comporte des petits trous dans lesquels vous allez pouvoir positionner vos composants ainsi que des fils qui vous permettront de réaliser votre circuit électrique. À l'intérieur de votre platine, il faut imaginer qu'il y a des chemins déjà existants que nous pouvons voir dans le dessin ci-dessous.



À ce stade, les connections doivent être faites avec des fils électriques monobrin dont l'aspect est rigide. Par précaution vous pouvez utiliser votre voltmètre pour tester la continuité des connexions qui vous semble défectueuses. Pour cela il faut utiliser la fonction « test de continuité » symbolisée par un logo qui illustre des ondes sonores.

Après l'étape de prototypage sur platine d'essai il faut, en fonction des exigences de votre projet, réaliser un circuit plus solide qui pourra résister à l'utilisation que vous souhaitez en faire. La première solution consiste à souder vos composants sur une plaque de prototypage comportant des lignes de cuivre et des trous.



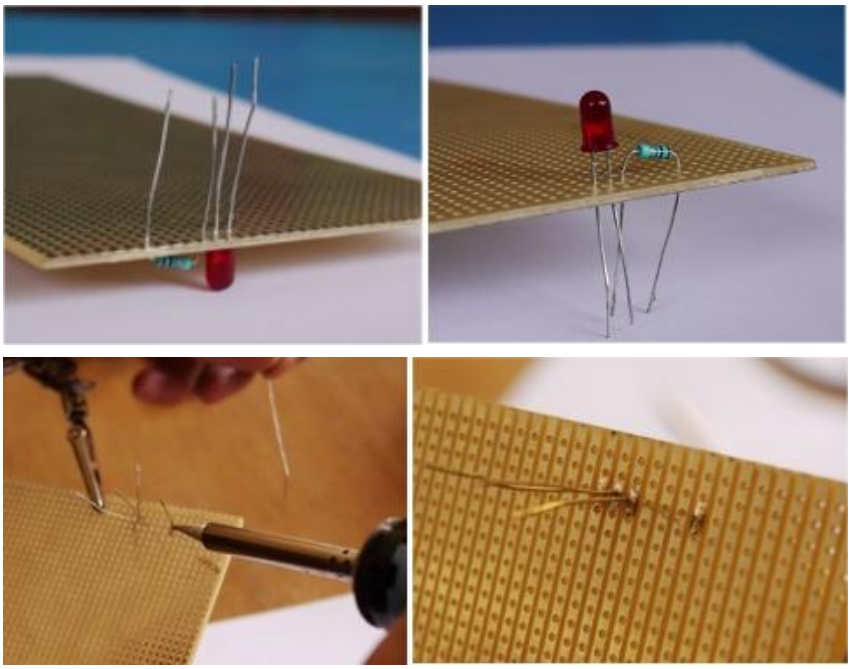
Une attention particulière est à apporter à la connexion de la batterie ou de l'alimentation électrique, car un faux contact à ce niveau est particulièrement risqué pour votre montage. Dans le cas de l'utilisation de batteries telles que les lithium-ion cela peut aller jusqu'à provoquer une explosion. Soyez prudent !

XXII-G - La soudure

Souder n'est pas sorcier. Voici quelques conseils pour réaliser une belle soudure. Cette étape intervient seulement après avoir testé votre montage sur une platine d'essai.

La soudure à l'étain vous permettra de souder vos fils et composants. L'étain de soudure comporte généralement du plomb, il faudra donc éviter l'inhalation des vapeurs de soudure. Il existe aussi de l'étain sans plomb, mais celui-ci est plus difficile à souder, car la température du fer doit être plus importante.

Tout d'abord, il faut étamer son fer, c'est-à-dire enduire la panne (bout du fer) avec de l'étain puis la nettoyer sur une éponge humide pour enlever le surplus d'étain. Votre pane doit être brillante. Il vous faut ensuite positionner vos composants du côté non cuivré de votre plaque de prototypage, l'utilisation d'une troisième main pour maintenir la plaque est plus confortable. Bien installé, vous pouvez alors commencer par chauffer la zone où vous souhaitez déposer votre point de soudure (2 à 4 secondes) puis faire l'apport de soudure à l'intersection entre le bout de la panne et la patte du composant. Il faut que votre soudure soit en forme de chapiteau et ne forme pas une boule sur la plaque.



XXII-H - Le circuit imprimé

À un stade plus avancé du projet il est d'usage de réaliser un circuit imprimé en utilisant le procédé de gravure chimique. Cette étape nécessite un outillage adapté ainsi que la connaissance d'un logiciel de routage comme Keykad pour réaliser votre circuit et y souder vos composants. Ce travail vous permettra, par exemple de réaliser vos extensions (shield) pour la carte Arduino. Cette démarche intervient lorsque vous souhaitez donner à votre projet une forme plus solide et durable. Il existe plusieurs sociétés qui pourront réaliser pour vous ces cartes communément appelées PCB (Printed Circuit Board).

Lors de l'utilisation de périphériques tels que des capteurs et actionneurs, vous allez certainement avoir besoin de positionner ces composants à distance de votre carte pour qu'ils puissent remplir leur fonction. Pour cela, il vous faudra utiliser des fils électriques de préférence multibrins qui pourront, contrairement aux fils monobrins, se plier et se manipuler sans se sectionner.

XXIII - Déboguer

La réalisation de projets avec Arduino comporte une grande part d'essais et d'erreurs. Que ce soit pour la partie électronique ou pour la programmation, il est fréquent pour les débutants, mais aussi pour les plus avancés, de faire des erreurs. Or, alors que celles-ci sont parfois très faciles à détecter, elles sont plus souvent « cachées » et demandent à être découvertes. Voilà pourquoi une bonne pratique de la programmation et de la conception de circuits est de mise (voir chapitres « **Précautions d'utilisation** » et « Bien structurer son code »).

XXIII-A - En programmation

Lorsqu'un problème survient ou que les choses ne se déroulent pas comme attendu, il est important d'identifier le problème en utilisant une approche méthodique et logique. Pour les problèmes de programmation, une pratique courante est de valider les différentes parties de code séparément, pour identifier la ou les parties du code qui occasionnent une anomalie. Pour ce faire, il est facile de mettre sous commentaire des sections complètes d'instructions en utilisant /* et */. Ensuite, il est important de prendre connaissance de l'état de nos variables, capteurs et autres. Utiliser la fonction Serial.println () pour faire afficher dans le moniteur sériel nos valeurs est très utile, et peut-être combiné avec des caractères nous indiquant où le Arduino se trouve dans le code. Par exemple le code suivant :

```
1. int valeurCapteur = 0;
2.
3. void setup()
4. {
5.     Serial.begin(9600);
6. }
7.
8. void loop()
9. {
10.     // afficher qu'on lit le capteur
11.     Serial.print("lire capteur : ");
12.
13.     // lire capteur et afficher
14.     valeurCapteur = analogRead(3);
15.     Serial.println(valeurCapteur);
16.
17.     // vérifier si capteur est égal à 5
18.     if (valeurCapteur=5) {
19.         Serial.println("test réussi!");
20.     }
21.     delay(1000);
22. }
```

Ce programme devrait imprimer « test réussi » si la valeur du capteur est à 5.

L'utilisation d'un Serial.print() avec du texte (« lire capteur : ») nous permet de précéder l'affichage de la valeur du capteur par un élément indiquant au niveau de la console à quoi correspond le chiffre qui sera affiché. Remarquez le fait que nous utilisons Serial.print() pour afficher le texte et rester sur la même ligne d'affichage, puis le *Serial.print**ln()* pour afficher la valeur et ensuite changer de ligne.

Notre programme comporte également un délai d'une seconde. Cela nous donne le temps de voir les informations affichées au moniteur. Autrement, les chiffres défileraient trop rapidement. Attention ! En ajoutant des délais, il est aussi possible de perdre la bonne exécution du programme dans son temps. Par exemple, des capteurs qui fournissent une réponse rapide (l'impact avec un piézo, par exemple) fournissent une information qui se déroule dans un très cours laps de temps. Ainsi, pendant que le programme attend avec delay(), il est possible que l'événement que nous cherchons à observer se soit déjà déroulé !

Lorsque nous copions ce programme, le compilons et l'exécutons, il ne fonctionne pas comme prévu ! Il contient délibérément un erreur. Même si le moniteur affiche une valeur différente de 5 pour le capteur, il affiche « test réussi ». Pourquoi ? Analysons le problème. Nous pouvons être suffisamment certains que la valeur du capteur est bien lue, et qu'elle module en fonction des manipulations physiques que nous ferions sur celui-ci. Si ce n'était pas le cas, le problème résiderait alors dans le capteur lui-même ou la connexion entre la carte Arduino et le capteur, mais comme

dans le cadre de cette simulation, la valeur du capteur est représentative du phénomène physique, notre problème se situe ailleurs.

Quelles instructions suivent celui de l'affichage de la valeur du capteur ? L'énoncé conditionnel if (valeurCapteur=5). Nous pouvons être suffisamment certains que la valeur du capteur n'est pas 5 en observant le moniteur sériel, mais tout de même, la condition semble réalisée puisqu'il affiche « test réussi ». Comme aucune autre instruction ne modifie valeurCapteur, il doit forcément y avoir quelque chose qui cloche avec la fonction if(). Si vous êtes incapable de trouver quoi, une bonne pratique est de se référer à l'aide offerte par le programme (*Help* > *Reference*) (ou Ctrl +Maj+F en surlignant le mot-clef recherché) afin de valider la bonne utilisation de la fonction, la syntaxe, etc. En observant la condition valeurCapteur=5, nous observons ainsi que nous nous sommes trompés d'opérateur ! Un « = » attribue la valeur à la variable qui se trouve à sa gauche. L'opérateur que nous aurions dû utiliser devrait plutôt être « == », qui retourne un *vrai* si les deux valeurs comparées sont équivalentes et un *faux* si elles ne le sont pas. Comme notre condition était une assignation, elle ne retournait pas *faux* et entraînait l'exécution des instructions comprises dans le if().

Que doit-on retenir de cet exemple ?

Lorsqu'on débogue un programme, il faut toujours avoir une **approche modulaire**, c'est-à-dire aborder chaque élément du code ou du circuit indépendamment des autres et s'assurer de son bon fonctionnement. Cette vérification doit également être faite de manière **séquentielle**, ou en ordre. On part soit de l'élément physique, pour remonter vers le circuit, puis vers l'Arduino, puis vers le programme. Ou l'inverse. Un autre exemple :

un capteur contrôle l'activation d'un moteur. Rien ne marche ! Docteur, que faire ?

- Vérifier que le capteur ne soit pas physiquement endommagé.
- Vérifier que le capteur soit bien connecté à son circuit.
- Vérifier que tous les composants du circuit soit bien connectés.
- Vérifier que la connexion au Arduino est bonne.
- Vérifier que la donnée est bien lue et l'afficher dans le moniteur.
- Vérifier les manipulations qui sont faites sur les variables ou la valeur du capteur.
- Vérifier que ces valeurs sont envoyées correctement au moteur.
- Vérifier que le moteur est bien connecté et non endommagé.

À travers toutes ces étapes, il est possible de tester hors du contexte du programme les éléments. Par exemple, pour tester que le moteur répond bien, il est possible de créer un nouveau petit programme temporaire pour ne tester que le moteur. Souvent, utiliser des exemples fournis dans le menu *File* ou le menu *Help* est utile, puisque ces exemples ne contiennent pas d'erreur en théorie.

XXIII-A-1 - En électronique

Le même principe s'applique en électronique et avec l'électricité. Analyser chaque circuit, ou chaque partie de circuit, ou chaque composant, de manière modulaire nous permettra d'identifier un composant grillé, un circuit mal configuré, etc., tout en prenant soin d'avoir d'abord vérifié que notre source d'alimentation est d'abord sous tension ! L'outil par excellence pour déboguer est bien évidemment le multimètre (voir chapitre « **Les bases de l'électronique** »).

XXIV - La connectique

La connectique regroupe toutes les techniques liées aux connexions physiques des liaisons électriques ainsi que des transmissions de données, c'est-à-dire les connecteurs et prises (19) .

XXIV-A - Liaison fil vers fil

Une liaison fil vers fil peut se réaliser soit de façon permanente, soit de façon à pouvoir débrancher et rebrancher deux fils entre eux. Pour réaliser une liaison permanente, vous pouvez tout simplement dénuder les extrémités de

chacun des fils, les torsader avec les doigts, les étamer, puis les souder entre eux. Pour isoler votre connexion, vous pouvez utiliser du ruban adhésif d'électricien appelé Barnier ou bien pour une plus grande finition, un bout de gaine thermo-rétractable.

XXIV-A-1 - Liaison carte vers fil

Une liaison carte vers fil nécessite la mise en œuvre d'un moyen de fixation temporaire ou permanent qui solidifie le point de jonction. En effet, un fil soudé directement sur une carte est très fragile, car à l'usage il se sectionnera rapidement. L'utilisation d'un pistolet à colle est un moyen efficace de fiabiliser vos connexions carte vers fil.

XXIV-A-2 - Connectique exotique

Pour les plus téméraires, il existe aussi des solutions plus originales pour réaliser des circuits sur des matériaux tels que le textile ou le papier en utilisant des fibres et encres conductrices qui peuvent être cousues ou déposées à la main. L'association du textile et de l'électronique est une pratique émergente. Des matériaux techniques tel que les textiles et fibres conductrices ouvrent de nombreuses perspectives dans le domaine de la mode et du design textile. La carte Arduino est une porte ouverte sur l'expérimentation de ces matériaux qui permettent de revisiter l'électronique traditionnelle pour la rendre plus proche de nos pratiques. Aborder l'électronique par le tissage ou la couture peut aussi être une manière de revaloriser des savoir-faire plus traditionnels.

XXV - D comme débrouille

L'idée de la débrouille rejoint la philosophie de la communauté Arduino. De nombreux utilisateurs passent par des solutions détournées pour réaliser leurs projets. Cette démarche de bidouilleur peut notamment être comparée à celle du *hacker* qui fait preuve d'inventivité en contournant les règles pour trouver les failles d'un système permettant ainsi d'en accroître la sécurité. Avec des moyens limités, mais une créativité abondante, il est facile de trouver les moyens de réaliser des projets ambitieux. Voici quelques conseils pratiques pour nourrir votre créativité.

XXV-A - Le circuit bending

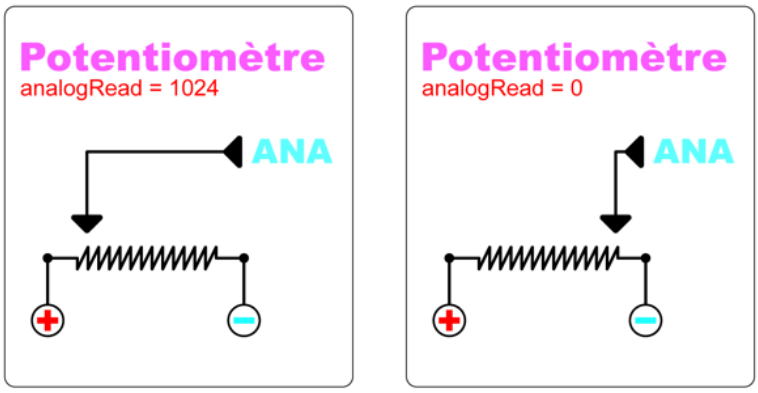
Le « circuit bending » est une pratique qui passe aussi par la récupération et le détournement d'objets. Cette pratique, qui se traduit par « torsion de circuit », consiste à détourner et modifier les circuits électroniques d'une variété d'objets pour en faire un usage autre que celui auquel ils sont normalement destinés. Par exemple, faire des courts-circuits sur le circuit électronique d'un jouet pour produire de la musique. Le « circuit bending » est pratiqué sur tous les types de signaux analogiques, qu'il s'agisse d'un signal vidéo, électrique ou audio. Cette pratique originale est une manière de découvrir l'électronique par l'expérimentation. Certains praticiens utilisent l'Arduino dans le but d'autonomiser la fréquence et le rythme des courts-circuits à l'aide de relais électriques transformant l'objet détourné en séquenceur musical, d'autres utilisent des signaux vidéos pour les utiliser comme générateur audio. D'autres encore vont générer directement des signaux électriques et les injecter dans le circuit d'un objet pour provoquer des comportements erratiques ou bien utiliser des consoles de jeux comme boîtes à rythme et synthétiseurs.

XXV-B - La récupération

La récupération de composants sur des objets usagés est un bon moyen de se procurer la plupart des pièces nécessaires à la réalisation de vos montages électroniques. À partir du moment où vous serez apte à identifier les composants électroniques de base que l'on peut trouver dans les objets de récupération tels que les vieux jouets, ordinateurs, imprimantes, etc., ceux-ci pourraient devenir pour vous une source importante de matières premières. L'exemple de la RepRap nous montre que les scanners et imprimantes peuvent nous fournir des moteurs et pièces mécaniques pour réaliser une imprimante 3D, par exemple. Les blocs d'alimentation que l'on peut trouver dans les tours d'ordinateurs seront très pratiques pour alimenter vos montages. Les fils électriques que l'on peut trouver à l'intérieur des ordinateurs sont aussi des matériaux de bonne qualité qui vous permettront de connecter vos capteurs et actionneurs (voir chapitre « **La connectique** »). L'économie que vous réaliserez n'est alors pas négligeable et par le fait même vous éviterez que ces pièces ne se retrouvent tout simplement à la poubelle.

XXV-C - Capteurs maison

Il est possible de fabriquer des capteurs soi-même avec des matériaux simples et faciles à trouver. En étudiant le fonctionnement d'un capteur industriel, on peut constater que la plupart des appareils utilisent des principes physiques familiers et accessibles. Par exemple, un potentiomètre est généralement composé d'une bande de matière résistive sur laquelle se déplace un contacteur qui doit être connecté à la broche analogue de la carte Arduino. La résistance du potentiomètre augmente à mesure que le courant passe par une quantité plus grande de matière résistive.



Nous pouvons par exemple reproduire ce principe avec une trace de crayon-gris déposée sur du papier en guise de résistance sur laquelle nous déplacerons un fil électrique connecté à une entrée analogique de l'Arduino. Les deux extrémités de la trace devront être connectées aux bornes positives et négatives de l'Arduino. Nous pouvons aussi utiliser une feuille de papier noir dont la couleur est généralement fabriquée à partir du carbone pour obtenir le même effet.

Pour réaliser un simple interrupteur, nous pouvons utiliser deux feuilles de papier aluminium, pour fabriquer un électro-aimant de faible intensité, il est possible d'enrouler du fil de cuivre autour d'un clou, etc.

De nombreux exemple de ce type sont consultables sur le site Internet Instructables :

- **www.instructables.com**

XXV-D - Détournement des capteurs et actionneurs

Le détournement est le fait d'utiliser un objet de manière inattendue ou pour une fonction pour laquelle il n'était pas destiné. La plupart des capteurs peuvent récupérer d'autres phénomènes physiques que ceux pour lesquels ils sont conçus. Une interférence sur le capteur par un autre phénomène physique est une piste pour un détournement. Cela s'explique par le fait que certains phénomènes physiques sont liés ou se ressemblent. Des actionneurs peuvent également être utilisés à l'inverse comme capteurs. Par exemple :

- un capteur de flexion utilisé comme un capteur de vent ;
- un micro comme capteur de souffle (voir chapitre « **La cigarette ne tue pas les machine**s ») ;
- une LED bidirectionnelle photodiode ;
- un moteur utilisé comme capteur de vélocité / rotation ;
- un clavier d'ordinateur dissimulé sous le coussin d'un fauteuil utilisé comme capteur de pression.

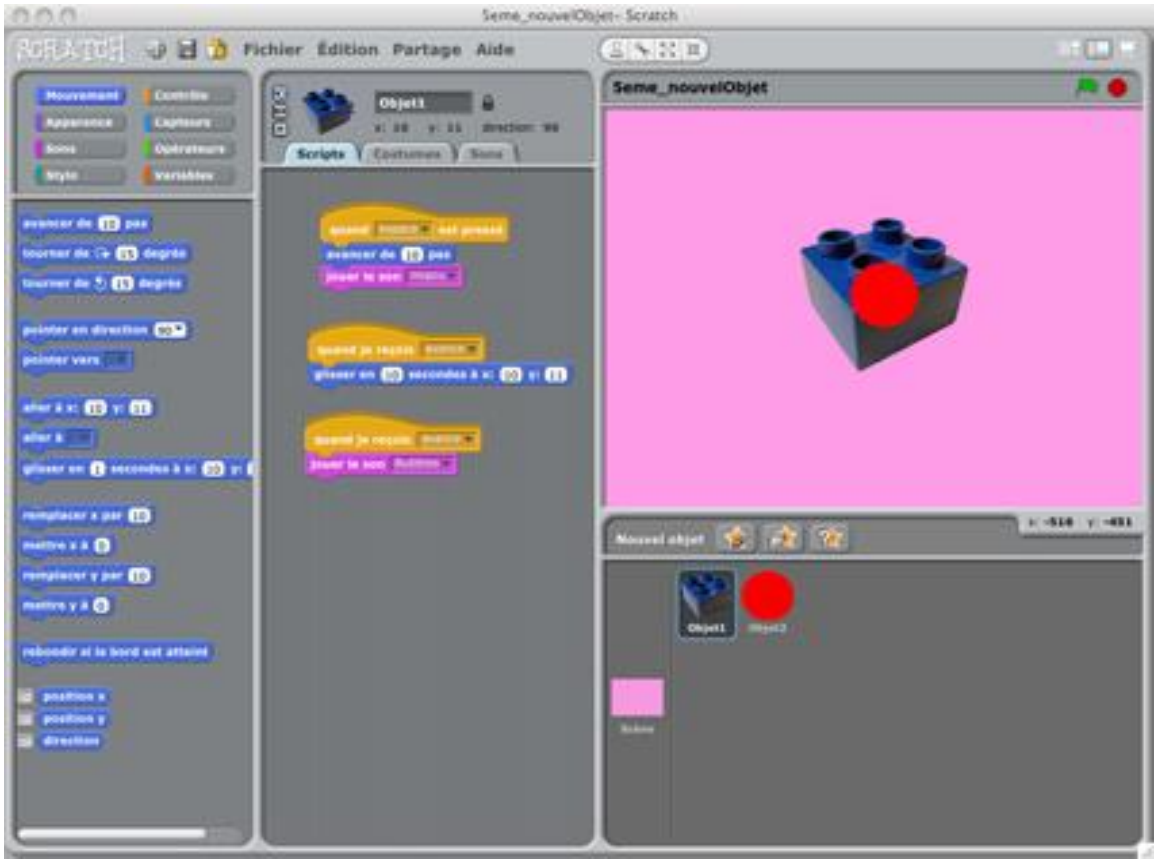
XXVI - S4A : un projet pédagogique

Les projets Sctratch et Scratch For Arduino sont développés par des Media Labs à des fins de transmission du savoir « programmer » pour les enfants, adolescents mais aussi les adultes qui découvrent l'univers de la programmation.

XXVI-A - Présentation de S4A et Scratch

S4A (Scratch For Arduino) est une extension (modification) de l'application Scratch qui a été créée en 2007 à des fins pédagogiques : apprendre à programmer des histoires interactives, à concevoir et réaliser des projets, à travailler en équipe. Ces applications permettent donc au plus grand nombre d'appréhender l'univers informatique pour comprendre ces outils numériques qui font dorénavant partie intégrante de notre quotidien.

Un détour vers Scratch va nous aider à découvrir S4A. Scratch est une application développée par la fondation Lifelong Kindergarten au sein du Media Lab du M.I.T. Scratch est un environnement de programmation qui permet d'aborder la programmation de façon ludique et intuitive. Le concept d'écriture du programme repose sur un principe d'assemblage de modules classés par code couleur (comme l'idée du Légo) et chaque module correspond à une instruction. Accessible aux enfants dès huit ans, Scratch et S4A peuvent être un premier pas vers la programmation pour les adultes et un moyen de prototypage rapide pour les artistes et concepteurs.



Pour en savoir plus :

- **scratch.mit.edu** ;
- **llk.media.mit.edu**.

L'extension S4A a été développée par Marina Conde, Victor Casado, Joan Güell, Jose García et Jordi Delgado du Media Lab Citilab en Espagne aidé par le Smalltalk Programming Group. Cette extension permet de programmer des instructions que la carte d'interfaçage Arduino peut exécuter et ce depuis une GUI Scratch à laquelle les créateurs ont ajouté des fonctions spécifiques à la carte Arduino. En effet, le principe d'écriture d'un programme avec Scratch et S4A consiste à emboîter des blocs classés en code-couleur qui correspondent à des instructions programmatiques : fonctions, variables, opérations, lecture de fichiers importés (images, sons). S4A présente donc des nouveaux blocs destinés uniquement à une programmation d'interactivité entre des capteurs et des actionneurs via une carte Arduino.

Pour en savoir plus :

- **seaside.citilab.eu**.

XXVI-B - Installer S4A

1 Se rendre sur le site http://seaside.citilab.eu/ à l'onglet « Download ».
2 Veiller à ce que l'IDE Arduino soit installée dans les applications de votre ordinateur.
3 Télécharger le « Firmware » développé par le Citilab.
4 Depuis l'IDE Arduino, configurer le port USB qui communique entre la carte Arduino et votre ordinateur, et sélectionner la version de la carte Arduino que vous utilisez.
5 Charger le Firmware dans la carte d'interfaçage Arduino (ce Firmware est un programme Arduino écrit par le Citilab destiné à traduire les programmes S4A afin que l'extension S4A et la carte Arduino puissent communiquer).
6 Télécharger et installer l'extension S4A.

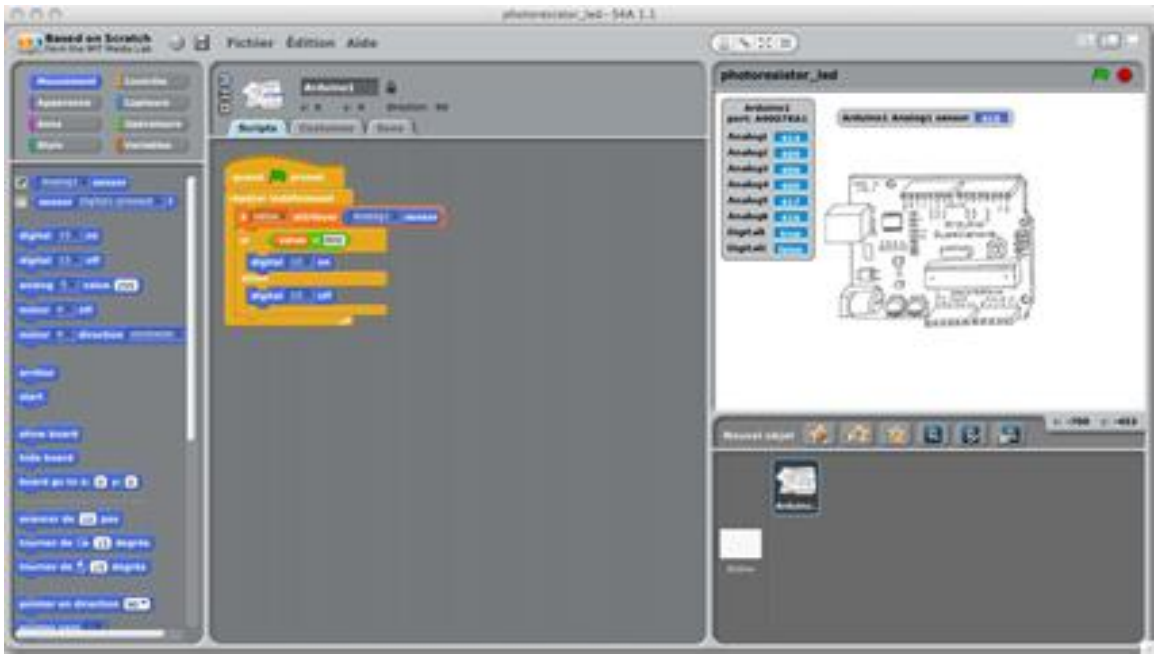
XXVI-C - Débuter avec S4A

Toujours dans l'onglet « Download », un répertoire d'exemples est téléchargeable pour débuter avec S4A (« Basic exemples ») et une vidéo de démonstration présente l'environnement et sa maniabilité à travers un prototype réalisé.

Suivons l'exemple donné par le Citilab d'un programme S4A : « photoresistor_led.sb » (« .sb » est l'extension usuelle des fichiers des programmes Scratch) avec le schéma électrique du montage fait sous Fritzing (« photoresistor_led.png »).

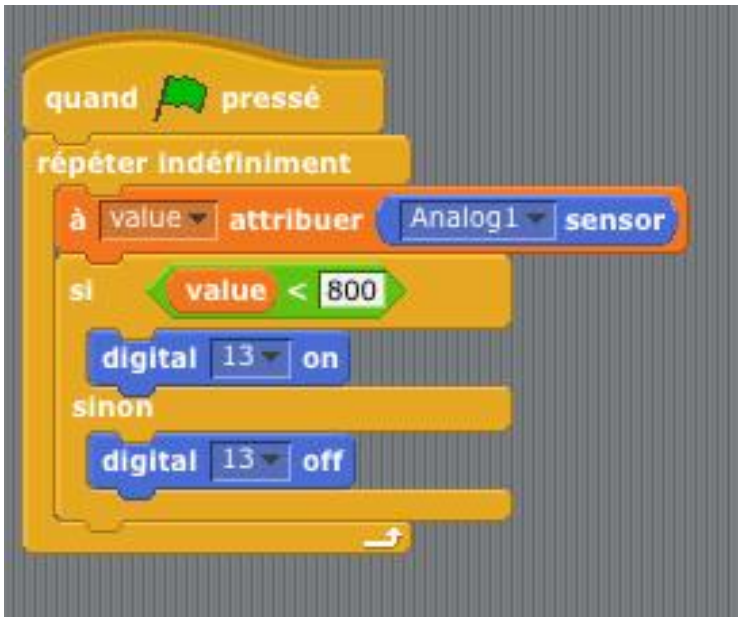
Connectez d'abord votre carte Arduino à l'ordinateur en veillant à ce que sous l'application Arduino vous ayez bien sélectionné la carte Arduino que vous possédez et le port USB de votre ordinateur que vous utilisez et en ayant préalablement effectué le montage de la LED et de la LDR (Light Diode Resistance, pour capteur de lumière) tel le schéma électrique proposé.

Lancez l'application S4A et depuis son menu choisir « ouvrir », une fenêtre S4A apparaît avec les différents exemples que vous avez téléchargés : ouvrir le fichier « photoresistor_led.sb ». Le programme S4A en visuel ci-dessous apparaît.



Il se peut que S4A affiche une fenêtre « Arduino 1 » comme le visuel ci-dessous. Il s'agit d'accepter la connexion entre l'application S4A et la carte Arduino. Sélectionnez « Use connection of sprite - Arduino1 » et cliquez sur « Accept ».

Expliquons le programme « photoresistor_led » ci-dessous :



« Quand le drapeau vert est pressé » signifie « quand nous avons compilé le programme et que nous l'exécutons ».

« Répéter indéfiniment » correspond à la fonction préenregistrée dans Processing « draw » soit le programme est exécuté en boucle jusqu'à ce que l'on appuie sur « stop » pour arrêter la lecture en boucle.

« A value attribuer Analog1 sensor » : ici nous avons d'abord déclaré une variable nommée « value » (cela s'est fait en cliquant sur l'onglet rouge « Variables » en haut à gauche de l'interface S4A, puis en cliquant sur « Nouvelle Variable »), puis nous attribuons à cette variable « value » la valeur numérique traduite du capteur LDR connecté à la carte Arduino et que nous avons nommé « Analog1 », car ce capteur est connecté sur le port (pin) A0.

N.B. : sur notre carte Arduino, les ports d'entrées/sorties analogiques sont nommées de A0 à A5 (par exemple, pour une carte de type Duemilanove). Sur l'interface de l'application S4A, ces ports sont traduits en commençant à 1, c'est-à-dire que le port A0 de la carte correspond à la nomination « Analog1 », le port A1 à « Analog2 », etc.

Nous pouvons lire les valeurs numériques traduites du capteur dans le tableau de lecture qui apparaît par défaut dans l'interface de S4A (ci-dessous) :

Ensuite est intégrée dans l'exécution en boucle du programme la structure conditionnelle « si » et « sinon » : si la valeur attribuée à notre variable « value » est inférieure à (800), nous envoyons l'instruction « on » au port digital 13 de la carte Arduino, celui-la même sur lequel nous avons connecté la LED : donc la LED s'allumera. Sinon, nous envoyons l'instruction « off » et la LED restera éteinte.

Nous pouvons dès lors modifier notre programme en cliquant sur la valeur (800) et en entrer une autre pour paramétrer notre programme en fonction de la luminosité ambiante. Nous pouvons aussi connecter d'autres capteurs de lumière et LED à la carte Arduino et rajouter des instructions de lecture : dans l'onglet « Mouvement », choisir, par exemple « digital 13 on » et en cliquant sur la flèche de déroulement d'items, sélectionner (10) ou (11) pour rajouter sur les ports de la carte des LED sur les ports 10 et 11.

Et enfin, nous pouvons aussi importer un nouvel objet dans S4A comme un visuel d'une créature fantastique et programmer ses actions sur la scène (background) en fonction de capteurs (bouton, potentiomètre, flexible, etc.) pour le faire jouer nous même.

Amusez-vous bien !

XXVII - Glossaire

Ce chapitre présente les définitions d'une série de termes techniques liés à Arduino, à la programmation et à l'électronique.

XXVII-A - Actionneurs

Les actionneurs sont des composants matériels, qui une fois correctement connectés à votre carte Arduino, permettent d'agir sur le monde extérieur. Ceux-ci ont comme rôle de convertir une valeur électrique en action physique.

XXVII-A-1 - Arduino

Arduino est une plate-forme libre de création d'objets électroniques composée d'un appareil d'entrée-sortie configurable (dénommé un micro-contrôleur) et d'un environnement de programmation.

XXVII-A-2 - Baud

Le baud est l'unité de mesure du nombre de symboles transmissibles par seconde lors d'une communication sérielle.

XXVII-A-3 - Baudrate

Mesure du baud.

XXVII-A-4 - Bibliothèque

Une bibliothèque est un ensemble de fonctions utilitaires, regroupées et mises à disposition des utilisateurs d'Arduino.

XXVII-A-5 - Capteurs

Les capteurs sont des composants matériels, qui une fois correctement connectés à votre carte Arduino, peuvent fournir des informations sur le monde extérieur. Ceux-ci ont comme rôle de saisir une grandeur physique et de la convertir en sortie informationnelle. Veuillez vous référer au chapitre « Introduction a l'électronique » pour plus d'information.

XXVII-A-6 - Circuit imprimé

Un circuit imprimé est un support, en général une plaque, permettant de relier électriquement un ensemble de composants électroniques entre eux, dans le but de réaliser un circuit électronique.

XXVII-A-7 - Dipôle

Un dipôle électrique est un composant électrique qui possède deux bornes. La tension aux bornes du dipôle est représentée par *uD* et l'intensité par *iD*.

XXVII-A-8 - Fil mono/multibrin

Le cuivre est le métal le plus utilisé pour faire des fils et câbles électriques, car il a une excellente conductivité électrique. On l'utilise soit en fil de section cylindrique monobrin (*rigide*), soit en section toujours cylindrique mais multibrin (*souple*).

XXVII-A-9 - IDE (integrated development environment)

Environnement de développement intégré (*EDI*) est un programme regroupant un ensemble d'outils pour le développement de logiciels. Un EDI regroupe un éditeur de texte, un compilateur, des outils automatiques de fabrication, et souvent un débogueur.

XXVII-A-10 - Inductance

L'inductance d'un circuit électrique est un coefficient qui traduit le fait qu'un courant le traversant crée un champ magnétique à travers la section entourée par ce circuit. Il en résulte un flux du champ magnétique à travers la section limitée par ce circuit.

XXVII-A-11 - LED

Une LED ou DEL (en français), ou encore diode électroluminescente, est un composant opto-électronique capable d'émettre de la lumière lorsqu'il est parcouru par un courant électrique. Il est important de noter que les LED laissent passer le courant que dans un seul sens, on parle de polarité des deux pattes (20) , une positive (+) et une négative (-).

XXVII-A-12 - Librairie

Angliscisme pour Bibliothèque. Voir **Bibliothèque**.

XXVII-A-13 - Longueur d'onde

La distance parcourue par l'onde au cours d'une période.Une onde est un phénomène physique se propageant et qui se reproduit identique à lui-même un peu plus tard dans le temps et un peu plus loin dans l'espace.

XXVII-A-14 - Moniteur seriel

Le moniteur sériel est un élément de l'environnement de programmation Arduino. Il permet de recevoir et envoyer des messages en communication sérielle à partir de l'ordinateur. (voir chapitre « **Prise en main rapide** »).

XXVII-A-15 - Multiplexeurs

Un multiplexeur (abréviation : MUX) est un circuit permettant de concentrer sur une même voie de transmission différents types de liaisons (informatique, télécopie, téléphonie) en sélectionnant une entrée parmi N.

XXVII-A-16 - Oscilloscope

Un oscilloscope est un instrument de mesure destiné à visualiser un signal électrique, le plus souvent variable au cours du temps. Il est utilisé par de nombreux scientifiques afin de visualiser soit des tensions électriques, soit diverses autres grandeurs physiques préalablement transformées en tension au moyen d'un convertisseur adapté.

XXVII-A-17 - Panne

Embout en métal à l'extrémité du fer à souder qui en chauffant, permet de faire fondre l'étain.

XXVII-A-18 - PCB

En anglais « printed circuit board », voir **circuit imprimé**.

XXVII-A-19 - Pin

Il s'agit des ports de l'Arduino. Ce sont les broches qui permettent de connecter des fils à la carte. Le terme est aussi utilisé dans le langage de programmation pour référer à ces ports physiques. Certaines *pins* servent à l'entrée d'information et d'autres à la sortie.

XXVII-A-20 - Platine d'essai

Une platine d'essai est un support, le plus souvent en plastique, qui comporte des petits trous dans lesquels vous allez pouvoir positionner vos composants ainsi que des fils qui vous permettrons de réaliser votre circuit électrique.

XXVII-A-21 - Platine de prototypage

Une platine est une plaque en époxyde ou en bakélite (plastique synthétique) qui comporte des trous et des lignes de cuivres permettant d'y souder des composants pour former un circuit.

XXVII-A-22 - Servomoteur

Moteur qui peut recevoir des informations de positionnement et les atteindre de manière autonome. Voir Chapitre « **Les bases de l'électronique** ».

XXVII-A-23 - Shield

Carte comprenant un circuit complexe qui se connecte directement à l'Arduino et qui assure une fonction spécifique (communication Internet, lecture de mp3, etc.).

XXVII-A-24 - Utilitaires

Un utilitaire (aussi appelé programme ou logiciel utilitaire) est un petit logiciel accomplissant une tâche ou un groupe de tâches associées sur équipement. Il est conçu pour aider à gérer et à régler une pièce d'équipement informatique, un système d'exploitation, un logiciel ou dans notre cas un micro-contrôleur.

XXVIII - Ressources en ligne

Des sites et liens pour en apprendre plus sur Arduino.

XXVIII-A - Site officiel d'Arduino

- Le site d'Arduino en français **www.arduino.cc/fr/**.
- Le forum d'Arduino en français **www.arduino.cc/cgi-bin/yabb2/YaBB.pl?board=francais**.

XXVIII-B - Ressources, Tutoriels Arduino

- **L'univers Arduino - Aperçu de la plateforme Arduino et ses cartes d'interface.**
- Réaliser son Arduino en papier : **lab.guilhermemartins.net/2009/05/06/paperduino-prints/**.

Suivre l'actualité Arduino : **https://arduino.developpez.com/**

XXVIII-C - D'autres cartes similaires

Grâce à son code source ouvert, Arduino a suscité d'autres projets matériels similaires. Voici une liste de cartes basés sur les puces atmels issues du projet Arduino (21) :

- Ardupilot **http://diydrones.com/profiles/blogs/ardupilot-main-page**.
- Cortino **http://www.bugblat.com/products/cor.html**.
- Freeduino **http://freeduino.org**.
- Funnel **http://funnel.cc/**.
- Hackteria bioElectronics **http://hackteria.org/?p=477**.
- Metaboard **http://metalab.at/wiki/Metaboard**.
- Roboduino **http://curiousinventor.com/kits/roboduino**.
- Sanguino **http://sanguino.cc/**.
- Teensy **http://www.pjrc.com/teensy/teensyduino.html**.
- Uduino **http://timothytwillman.com/?p=116**.

XXVIII-D - Prototypage et dessin des cartes

Des programmes pour élaborer une vision d'ensemble de la partie électronique de votre projet :

- Fritzing permet de dessiner son circuit de façon graphique, schématique ou électronique : **fritzing.org** ;
- Eagle s'apparente à Fritzing mais en plus puissant. Ce logiciel comporte une version gratuite (freeware) limitée par rapport à taille du circuit : **https://www.autodesk.com/products/eagle/overview** idem ;
- DesignSpark est en logiciel de conception de PCB sans limitation de taille, ni de couche. Il fonctionne avec Windows et Linux/Wine : **www.designspark.com** ;.
- Kicad est un logiciel Open Source (GPL) qui fonctionne pleinement sous Linux et Windows ainsi que sous Mac.

XXVIII-E - Où acheter du matériel en ligne

Voici quelques adresses où il est possible d'acheter des cartes ainsi que des composants :

- Ebay **http://www.ebay.com** ;
- Digitec **http://www.digitec.ch** ;
- Distrelec **http://www.distrelec.ch** ;
- Sparkfun **http://www.sparkfun.com** ;
- Hackspark **https://hackspark.fr/fr** ;
- Snootlab **http://snootlab.com** ;
- GoTronic **http://www.gotronic.fr**

XXVIII-F - Électronique

L'électronique est une branche de la physique appliquée traitant surtout de la gestion des signaux électriques. Avoir un minimum de connaissances dans cette matière est obligatoire pour travailler avec Arduino. Quelques cours en électronique sur Internet :

- cours d'électronique rédigés par des enseignants de Marseille : **sitelec.org/cours.htm** ;
- cours pour les débutants et les curieux : **www.elektronique.fr** ;
- Cours d'électronique, d'électricité et de montage de circuit : **www.zonetronik.com** ;
- Cours d'électronique pour construire son propre robot : **www.positron-libre.com/cours/electronique/cours-electronique.php** ;
- Nomenclature et description des composants électroniques sur interface-z : **https://www.interface-z.fr/conseil/index.php**.

XXVIII-G - La robotique

Un robot est un dispositif alliant mécanique, électronique et informatique accomplissant automatiquement des tâches. Les cartes Arduino rentrent dans la fabrication de certains d'entre eux. Des liens pour en savoir plus.

- La page Robotique de l'encyclopédie Wikipédia : **fr.wikipedia.org/wiki/Robotique**.
- Des projets pour réaliser des robots Wi-Fi avec Arduino : **robot.aspi.free.fr**.
- Ressources pour l'enseignement technologique : **www.technologuepro.com**.

En cherchant sur la toile, vous pouvez trouver des kits pour réaliser vos robots avec Arduino.

XXVIII-H - Outils logiciels utilisés fréquemment avec Arduino

Lorsque l'Arduino est connecté à un ordinateur, il est capable de communiquer avec diverses applications, notamment Processing et Pure Data. Pour en savoir plus sur ces environnements de création multimédia :

- Le manuel Processing sur Flossmanuals francophone : **http://fr.flossmanuals.net/processing/** ;
- Le manuel Pure Data sur Flossmanuals francophone : **http://fr.flossmanuals.net/puredata/**.

1 : L'école « Interaction Design Institute Ivrea » (IDII) est aujourd'hui située à Copenhagen sous le nom de « Copenhagen Institute of Interaction Design ».
2 : Casey Reas était lui-même enseignant à IVREA, pour Processing, à cette époque.
3 : Design By Numbers, un langage de programmation spécialement dédié aux étudiants en arts visuels.
4 : Seul le nom « Arduino » n'est pas utilisable librement, de telle sorte à toujours pouvoir identifier le projet de ses dérivés.
5 : **http://processing.org/**.
6 : **http://fr.flossmanuals.net/Processing**.
7 : **http://fr.flossmanuals.net/Puredata**.
8 : **http://fr.wikipedia.org/wiki/SuperCollider**.
9 : **http://www.pinguino.cc/**.
10 : **http://beeboard.madefree.eu/doku.php**.
11 : **http://www.freeduino.org/**.
12 : **http://www.ginsingsound.com/**.
13 : **http://hackteria.org/?p=477**.
14 : Voir chapitre « **Introduction à l'électronique** ».
15 : **http://fr.wikipedia.org/wiki/Langage_de_programmation**.
16 : **http://fr.wikipedia.org/wiki/Convention_de_nommage**.
17 : **http://processing.org**.
18 : **http://fr.flossmanuals.net/Processing/**.
19 : **http://fr.wikipedia.org/wiki/Connectique**.
20 : Notez qu'il existe des LED avec plus de deux pattes qui sont capables de changer leur couleur.
21 : Pour certaines interfaces à très bas coût qui communiquent par port série, il est nécessaire d'acheter un câble convertisseur USB/Série compatible pour les ordinateurs qui ne possèdent pas ce port.