CONSERVATOIRE NATIONAL DES ARTS ET METIERS
CENTRE NATIONAL DE L'ENTREPRENEURIAT

**Projet NSY110**
**Interaction homme-machine**

# PianoRollp5

une application musicale avec *Arduino* et *Processing*

Jérôme Abel
abel.jerome@free.fr

Eddy Oliveira
eddy.oliveira@orange.fr

Date de soutenance : 2 février 2010.

# Table des matières

# 1 – Introduction

Dans le cadre du cours « NSY110 Interaction Homme-machine », nous avons réalisé une application musicale en utilisant une suite logicielle et matérielle libre : *Arduino* et *Processing*. Le « p5 » du nom du logiciel est d'ailleurs une convention qui indique l'utilisation de *Processing*. Ces outils de développement sont libres et soutenus par une communauté bouillonnante de développeurs, artistes, acteurs culturels et scientifiques. Nous verrons dans les parties 3 et 4 leur fonctionnement.

Nous proposons de montrer une chaîne complète d'interaction, reliant gestes physiques et logiciel. Nous explorerons une solution alternative qui rend mieux compte de l'expressivité du jeu musical que les périphériques d'entrée classiques, que sont la souris et le clavier.

# 2 – Description du projet

*PianoRollp5* est une simple implémentation d'un outil de représentation temporelle très utilisé dans les logiciels de création musicale : le *piano roll*. Celui-ci reprend le principe des rouleaux perforés utilisés pour les pianos mécaniques (figures 1 et 2).

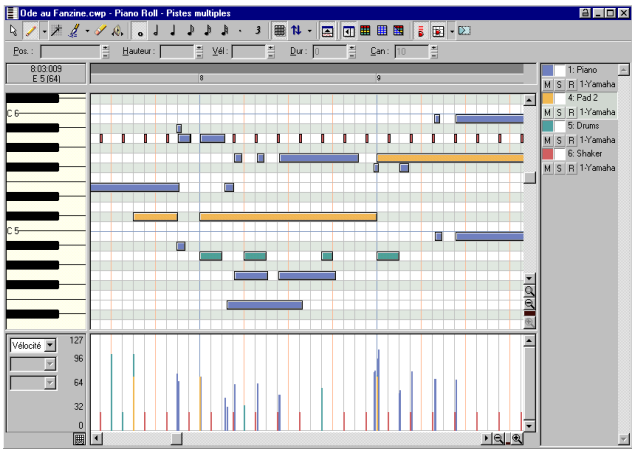



***Figure 1.****Exemple de représentation sous forme depiano roll du logiciel Cakewalk.*

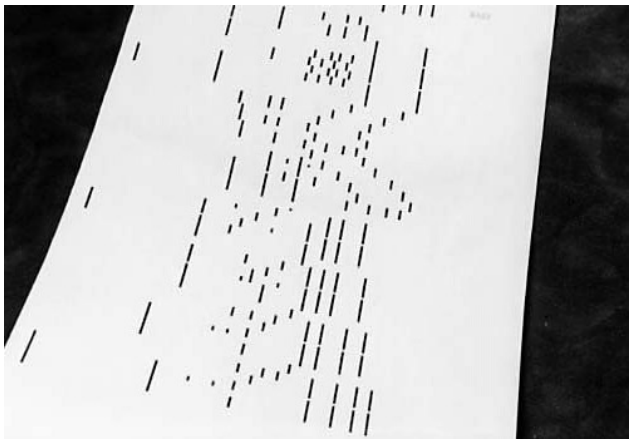


***Figure 2.*** *Un rouleau perforé pour piano mécanique [EASTSIDE].*

Le plus souvent sont affichées verticalement la hauteur des notes, accompagnée des touches de piano situées sur la gauche, et horizontalement leur durée. « Limitée à des échelles discrètes de hauteurs, mais sans contrainte sur la discrétisation de l'axe temporel ou sur le nombre de voies de polyphonie, elle est en particulier utilisée pour la représentation de séquences de notes MIDI*, caractérisées par un numéro de note, des instants de début et de fin, et un paramètre de vélocité qui n'est en général pas figuré, ou apparaît par le biais d'une graduation de couleurs. » [VINET, p. 110]. Dans la figure 1, la vélocité est affichée en bas. Le trait est relié à une note sur l'axe horizontal et sa hauteur indique la valeur de la vélocité.
La discrétisation des hauteurs s'appuie sur le principe de la gamme tempérée occidentale, qui découpe l'octave en douze intervalles chromatiques égaux. En programmation, nous pourrions passer outre cette contrainte en utilisant des hauteurs entre ces intervalles. Cependant, cette représentation est efficace à plus d'un titre. Elle est simple, immédiatement compréhensible, et permet de composer des séquences en MIDI.

* Le MIDI (*Musical Instrument Digital Interface)* est un protocole de communication et de commande permettant l'échange de données entre instruments de musique électronique, un ou plusieurs de ces « instruments » pouvant être des ordinateurs. » [MIDI].

# 3 – Captation des gestes

### 3.1 - L'interaction gestuelle pour la musique

« (…) [En] restant figées sur le clavier et la souris, la majorité des interfaces actuelles font un bien piètre usage des capacités de la communication humaine. » [BEAUDOIN]. Dans le cas d'une application musicale, cela ne peut nous satisfaire. Les gestes et le corps dans la musique sont des éléments qui participent à la création et à l'expressivité. Depuis longtemps, chercheurs, artistes et industriels proposent d'autres solutions de captation [NIME][ACROE][IRCAM][BIRON]. Ne pouvant aborder avec sérieux ces nombreuses recherches, nous nous contenterons de distinguer trois espaces d'innovations dans l'histoire de l'interaction gestuelle pour la musique :

→ Conjointement à l'apparition des ordinateurs personnels, des interfaces graphiques et des langages de programmation musicale, le MIDI a été un élément essentiel dans l'évolution de l'informatique musicale. Depuis les années 80, beaucoup de contrôleurs sont apparus, utilisant des touches piano, boutons, *pads* avec vélocités, potentiomètres circulaires ou linéaires, roues, pédales, etc.

→ Récemment, les industries du jeu vidéo et de la téléphonie mobile et des centres de recherches comme le MIT (*Massachusetts Institute of Technology*) ont apporté quelques interfaces intéressantes et ré-utilisables dans d'autres contextes, notamment artistiques : écrans tactiles, analyse vidéo, manettes de jeu sans fils.

→ Une troisième voie, que nous avons expérimenté pour le projet, consiste à utiliser directement des composants électroniques. Ceux-ci sont connectés à des interfaces matérielles convertissant le signal électrique en signal numérique, exploitable par l'ordinateur. La captation des phénomènes physiques, tels que la pression atmosphérique, les infra-rouges, la chaleur, le magnétisme, etc., rend possible des interactions moins conventionnelles.

### 3.2 - Conception du contrôleur

Pour la conception de notre contrôleur, nous pouvons nous guider à l'aide de la structure décrite dans la figure 3.

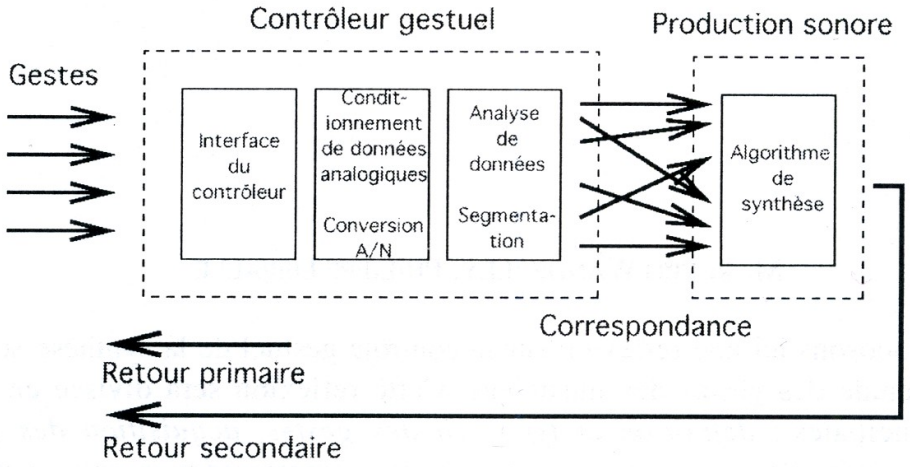



***Figure 3.*** *Structure type d'un instrument virtuel ou composé [WANDERLEY, p. 146].*

Nous distinguons :

- le geste, c'est-à-dire ce que nous voulons capter ;
- le contrôleur gestuel, qui va nous permettre de récupérer des données exploitables ;
- le lien de correspondance de ces données vers le générateur de son ;
- le générateur de son.

***Figure 4.*** *Léon Theremin.*

Le geste que nous voulons capter est un mouvement de la main, à la manière du thérémine, un des plus anciens instruments de musique électronique, inventé en 1919 par le Russe Lev Sergeïevitch Termen (plus connu sous le nom de Léon Theremin) (figure 4). Pour cela, nous pouvons utiliser un capteur de proximité infrarouge qui calcule la distance entre la main et le dit capteur [PROXI].

Le contrôleur gestuel est une *Arduino*, un ensemble d'outils que nous détaillerons en fin de chapitre. Cette carte exploite les données du capteur et les envoie via le port série USB. Aucun traitement, ni filtrage ne sont effectués.

La partie la plus intéressante est celle qui consiste à relier ces données à un paramètre du générateur de son. Ici, nous resterons simple, il ne s'agit pas d'un instrument mais d'un contrôleur. Nous utiliserons les données du capteur comme hauteurs de notes ou pour faire varier la vitesse de lecture des séquences de notes du *piano roll*. « (…) [Un] *instrument virtuel ou composé* doit intégrer un élément traduisant explicitement la correspondance entre les variables gestuelles, paramètres issus du contrôleur, et les variables de synthèse que sont les paramètres d'entrée de l'algorithme de calcul des échantillons sonores. Nous avons appelé cet élément *lien instrumental* en traduction du terme anglais *mapping*. C'est le *lien instrumental* qui, en réalité, donne sens au paramètre de sortie du contrôleur. Selon la nature de ce lien, un même mouvement représentera la fréquence d'un partiel, la fréquence fondamentale d'un signal périodique, le taux de bruit, l'intensité, (...) ou bien des paramètres perceptifs, comme la hauteur, la sonie, la rugosité, etc. Ce lien doit permettre au musicien de contrôler l'*instrument* selon l'approche voulue (de bas niveau, perceptive, ou abstraite) tout en assurant précision et expressivité. » [WANDERLEY, p. 156].

Le générateur de son sera le synthétiseur MIDI standard de l'ordinateur.

### 3.3 - Chaîne d'interaction


***Figure 5.*** *Chaîne d'interaction*

### 3.4 - Arduino en quelques mots

*Arduino* est à la fois :
- un micro-contrôleur, généralement un ATMEL AVR ;
- un langage de programmation et un compilateur ;
- un environnement de développement (IDE) multiplateforme proche de celui de *Processing.*
[NOBLE, p. 91]

Le principe est d'utiliser l'environnement pour écrire du code et le charger via USB dans le microcontrôleur de la carte. Ensuite, dès que la carte est alimentée par le port USB ou une source d'énergie externe, le programme fonctionne en boucle. Il existe aujourd'hui une grande quantité de cartes proposant un nombre d'entrées/sorties numériques et analogiques différent [MAKE].

*Arduino* est un projet *open source* pour faire du prototypage électronique. C'est unique, car des ressources de bas niveau jusque là propriétaires et élitistes sont à portée de main. La programmation assembleur qui est très spécifique et difficile devient accessible.

Un exemple éloquent : nous devons faire un test d'égalité entre une valeur et 14. Si c'est vrai, alors remettre cette valeur à zéro.

Code Arduino
```
if (w == 14) w=0;
```

Code assembleur PIC p16f877
```
movlw   0x0e  ; 0x0e=14 en Hexadécimal
xorwf   0x20, W
btfsc   STATUS, Z
goto    label_remise_a_zero
```

Des ressources sont disponibles en anglais et en français [ARDUINO].

# 4 – Processing

### 4.1 - Présentation

En 2001, Benjamin Fry et Casey Reas sont étudiants au *Media Lab* du MIT (*Massachusetts Institute of Technology*). Ils participent au développement de *Design by numbers*, un environnement de programmation et un langage créé pour les infographistes et artistes visuels comme une introduction à la programmation [DBN]. Ce logiciel est le fruit du travail de leur professeur, John Maeda, *Processing* se présente comme son extension.

*Processing* est à la fois :
- un environnement de développement (PDE, *Processing Development Environment*) ;
- une interface de programmation (API) ;
- un langage de programmation basé sur Java.

L'objectif est de permettre à des artistes et designers de développer rapidement et simplement leurs idées tout en apprenant la programmation. L'accent est de suite mis sur les applications visuelles avec entre autres un mécanisme simplifié pour créer des animations. Avec l'apport de nombreuses bibliothèques, de nouvelles méthodes sont disponibles. Elles permettent de manipuler des sons, des films, de la 3D, de communiquer en MIDI, en OSC (*Open Sound Control*), etc. [BIBLIOP5].

Un aspect important est son accessibilité : *open source*, gratuit et multiplateforme (Windows, Mac OS X et Linux) [PROCESSING].

### 4.2 - Croquis et projets

Un programme créé avec *Processing* s'appelle un *sketch* (croquis). Les programmes sont mis dans un dossier appelé *sketchbook* (carnet de croquis). Quand on enregistre son programme, un dossier qui porte le même nom est automatiquement créé. L'organisation du projet se fait, pour l'essentiel, dans ce répertoire. Si nous devons utiliser une image par exemple, une des solutions les plus utilisées est de la mettre dans un dossier « data ».

La notion de croquis est intéressante. Elle exprime la volonté de développer des prototypes, de coder rapidement pour tester ses idées. « Pourquoi forcer les étudiants et les programmeurs occasionnels à apprendre les fondements du graphisme, le *threading* et la gestion des événements avant qu'ils puissent montrer quelque chose à l'écran qui interagit avec la souris ? De même pour les développeurs confirmés; pourquoi devraient-ils toujours avoir besoin d'écrire les deux mêmes pages de code pour commencer un projet ? » [FRY, p. 29].

Concernant la taille d'un projet, « l'environnement est conçu autour de projets qui ne font que quelques pages de code, le plus souvent trois ou cinq onglets* [*tab*] au total. Cela couvre un nombre significatif de projets développés pour tester ses idées, qui peuvent être ensuite intégrer dans des projets plus importants (…). » [FRY, p. 28]. Donc, le plus souvent, la cible étant le prototypage, le projet est de petite ou de moyenne taille. Néanmoins, à mesure que ses capacités s'étendent, *Processing* s'utilise de plus en plus dans des milieux professionnels plus avancés, comme le montre la page du site http://processing.org/about

Un atout essentiel est l'exportation de son programme. Deux options sont possibles : créer un *applet* Java pour le visualiser dans un navigateur Web ou créer une application exécutable sur les trois plateformes habituelles : Windows, Mac OS X et Linux.

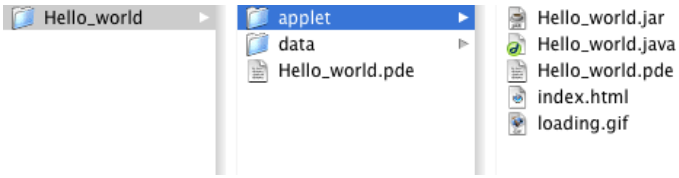



***Figure 6.*** *Exportation du programme et contenu du dossier.*

### 4.3 - Programmation et création

*Processing* fait partie d'un vaste ensemble de logiciels qui permettent de manipuler des données multimédia et de réaliser des prototypes pour des œuvres artistiques : art interactif, musical, sonore, visuel, etc. Les artistes s'approprient la programmation, le code devient création. À propos de la musique, François Bayle, directeur du GRM (Groupe de recherches musicales) de 1966 à 1997, dit : « Les compositeurs ne sont plus les seuls créateurs, il faut partir de ce constat. Ce qui s'est passé avec la technologie, c'est que justement elle a été promue, du fait même de la complexité des systèmes, au rang de création. » [DELALANDE, p. 233]. Il faut donc comprendre l'existence de ce type de logiciel dans un mouvement culturel plus large, motivé par le savoir partageable, la pratique expérimentale et, dans une vision plus politique, par l'appropriation citoyenne des technologies [CREATION].

* Dans le PDE, les onglets sont des parties du code. En général, il faut un onglet pour les fonctions principales et d'autres si des classes doivent être ajoutées (voir 4.6 - PDE).

### 4.4 - Processing et Java

Pour les développeurs Java, certaines confusions peuvent apparaître :

- *Processing* est une application Java, à ce titre, elle utilise la machine virtuelle Java (JVM). Un programme créé avec *Processing* a aussi besoin de la JVM pour s'exécuter [NOBLE, p. 53].

- *Processing* n'est pas Java. « La syntaxe *Processing* est essentiellement un dialecte de Java. Quand un utilisateur lance un programme dans le PDE, le code est convertie en syntaxe Java par le préprocesseur et ensuite compilé comme du code Java standard. » [FRY, p. 331]. Nous verrons le rôle du préprocesseur dans la partie 4.7 - Modes.

- L'API de *Processing* hérite de java.applet.Applet. La classe PApplet est la classe dont hérite chaque croquis. Elle implémente les méthodes de bases, que nous verrons plus loin : `setup()`, `draw()`, les événements de la souris et du clavier.


***Figure 7.*** *Héritage de la classe PApplet [PApplet].*

### 4.5 - Ressources

Les ressources sont multiples. La figure 8 présente l'ensemble des ressources disponibles sur le site officiel.

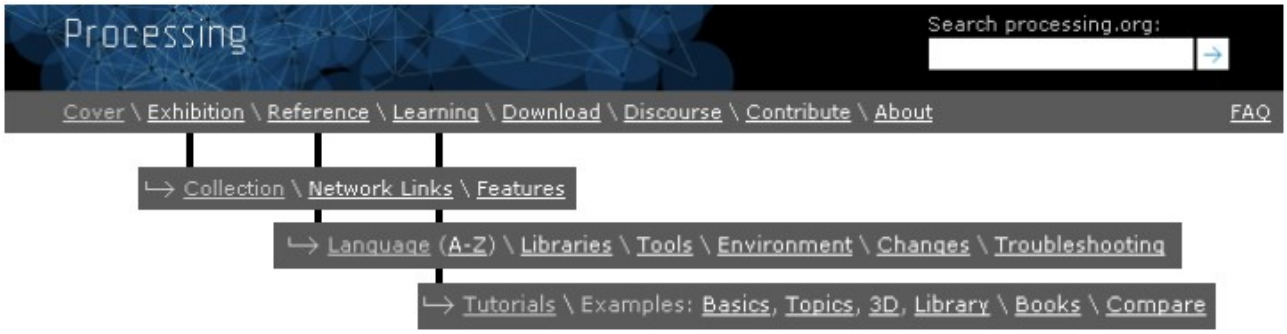



***Figure 8.*** *Carte du site officiel : [http://processing.org](http://processing.org)*

Nous pouvons noter une grande cohérence et complémentarité dans ces ressources : un lexique du langage, la liste des bibliothèques, une description de l'environnement de développement, un forum d'entraide (*discourse*), des exemples de réalisations, des tutoriels, des livres en anglais et une page *about* qui résume le projet.

Tout cela est rendu possible par une communauté d'une exceptionnelle vivacité. La version actuelle de *Processing,* la 1.0.9 en atteste. D'autres projets libres datent de plus longtemps et ne sont toujours pas disponibles en version 1.0.

En guise de ressources internes, c'est-à-dire accessibles depuis le PDE, il y a le menu *Help,* des exemples classés par sujets dans le menu *File*, une possibilité d'accéder à la documentation des méthodes en les surlignant avec la souris et en ouvrant le menu contextuel (*Find in reference).*

### 4.6 - PDE (Processing Development Environment)

La fenêtre du programme est constituée de quatre zones principales :

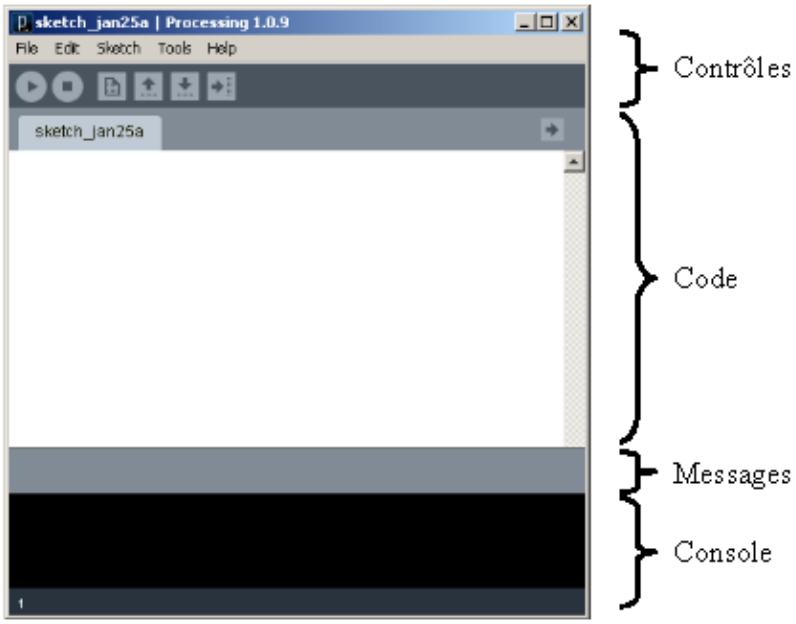



***Figure 9.*** *Fenêtre de l'environnement de développement.*

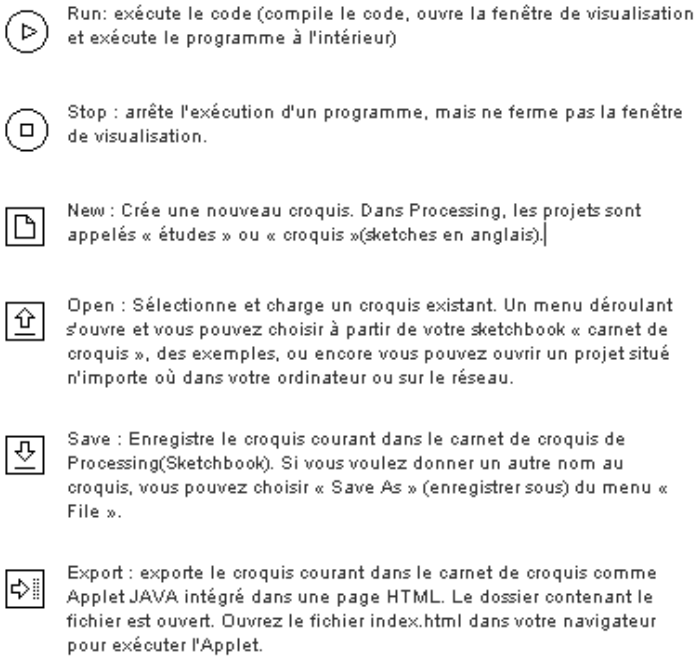



***Figure 10.*** *Description des boutons [MULTIMEDIALAB].*

### 4.7 - Modes

Il existe trois modes de programmation : basique, continu et Java. Ce qui suit est entièrement tiré du livre de Ben Fry [FRY, p. 332-333].

**4.7.a - Le mode basique (*basic*)** est utilisé pour dessiner des images statiques et étudier les notions fondamentales de la programmation. Le préprocesseur convertit le programme en méthode `setup()`. Le programme lance une fois cette méthode et s'arrête.

Code Processing ======> Code Java

```
size(200,200);
background(255);
stroke(0);
line(0,0,200,200);
```

```
import processing.core.*;
public class BasicSketch extends PApplet {
      public void setup() {
            size(200,200);
            background(255);
            stroke(0);
            line(0,0,200,200);
      }
}
```

**4.7.b - Le mode continu (*continuous*)** utilise la méthode `setup()` exécutée une fois lorsque le programme commence et la méthode `draw()` qui, par défaut, répète indéfiniment le code à l'intérieur. Cette structure permet de créer des classes et d'utiliser les événements de la souris et du clavier.

Code Processing ======> Code Java

```
void setup() {
   size(200,200);
   recMode(CENTER);
   noStroke();
   fill(0,102,153,204);
}

void draw() {
   background(255);
   rect(mouseX,mouseY,50,50);
}
```

```
import processing.core.*;
public class ContinuSketch extends PApplet {
      public void setup() {
         size(200,200);
         recMode(CENTER);
         noStroke();
         fill(0,102,153,204);
      }

      public void draw() {
         background(255);
         rect(mouseX,mouseY,50,50);
      }
}
```

**4.7.c - Le mode Java** est plus flexible. Il permet d'écrire des programmes Java dans l'environnement de développement. C'est un mode qui n'est pas vraiment recommandé, il est en général préférable d'utiliser un autre environnement et d'importer l'API.

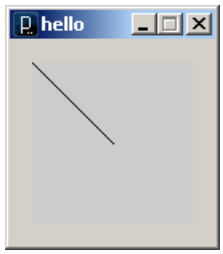
```
public class MyDemo extends PApplet {
      public void setup() {
            size(200,200);
      }

      public void draw() {
            background(255);
            rect(mouseX,mouseY,50,50);
      }
}
```

Pour en savoir plus sur ce mode : [JAVAP5].

## 4.8 - Exemples

### 4.8.a - Hello World

Le premier programme en *Processing* pourrait être l'affichage d'une ligne.




```
line(0,0,50,50);
```

Quand on lance le programme une fenêtre s'ouvre et affiche une ligne avec les points (0,0) et (50,50). Par défaut, la taille de la fenêtre est de  100 * 100 pixels, la couleur de fond est un gris clair et le titre de la fenêtre est le nom du *sketch*.

### 4.8.b - Exemples avec le mode continu

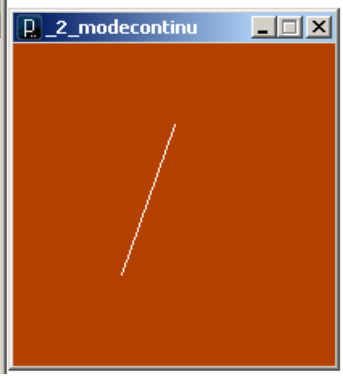
Pour pouvoir commencer à programmer, il faut bien comprendre le mode continu.

Exemple 1 :


```
int idFrame = 0;
void setup(){
  println("--- DEBUT DU PROGRAMME ---");
}

void draw(){
 println ("Frame n° : " + idFrame); //affichage dans la console
 idFrame++;
}
```

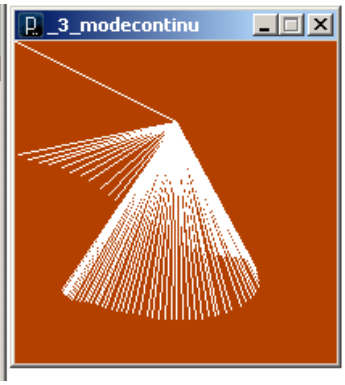
Exemple 2 :

```
_2_modecontinu
void setup(){
   size(200,200);                // toujours en premier dans le setup()
   stroke(255);                  // couleur des traits (255 = blanc)
}

void draw() {                    // boucle d'affichage, par défaut le taux de
                                 // rafraichissement est de 60 images/seconde
    background(180,64,0);        // couleur de fond de la fenêtre
    line(100,50,mouseX,mouseY); // affichage de la ligne,
                                 // le deuxième point dépend
                                 // des coordonnées de la souris
}
```




Exemple 3 :

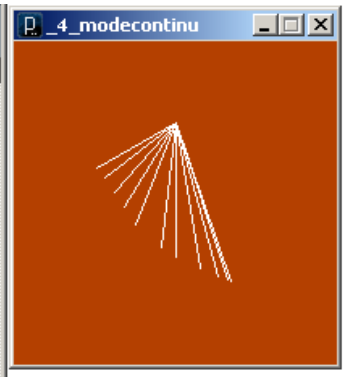
```
_3_modecontinu
void setup(){
   size(200,200);
   stroke(255);
   background(180,64,0);        // Cette fois-ci la couleur de fond est
                                // mise dans setup().
}

void draw() {
    line(100,50,mouseX,mouseY);
}
```



Exemple 4 :

_4_modecontinu

```
void setup(){
   size(200,200);
   stroke(255);
}

void draw() {
    line(100,50,mouseX,mouseY);
}

void mousePressed(){          // Quand on appuie sur un bouton de la souris

  background(180,64,0);       // Cette fois-ci la couleur de fond est
                              // mise dans mousePressed().
                              // Ce qui permet de remplir la fenêtre
                              // avec cette couleur, et donc d'effacer
                              // les lignes précédentes.
}
```

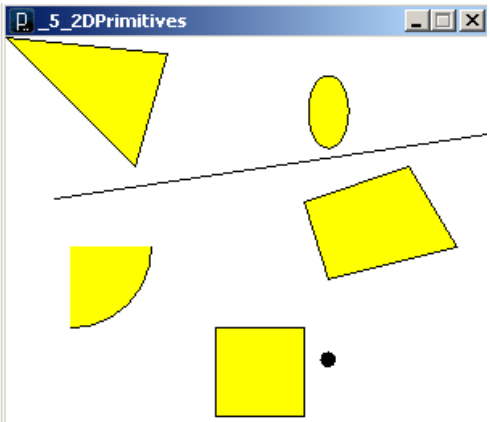



### 4.8.c - Primitives 2D

Exemple 5 :

_5_2DPrimitives

```
size(300,300);
background(255);
fill(255,255,0); // rouge,vert,bleu
triangle(0, 0, 100, 10, 80, 80);
line(30, 100, 300, 60);
quad(200, 150,  280, 130, 250, 80,185, 102);
arc(40, 130, 100, 100, 0, PI/2);
ellipse(200, 46, 25, 45);
rect(130, 180, 55, 55);
strokeWeight(10); // taille des traits
point(200, 200);
```

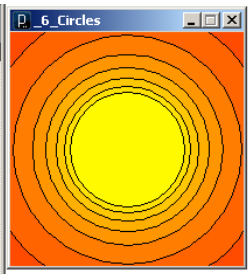



### 4.8.d - Exemples plus élaborés

Exemple 6 : structures de contrôle

_6_Circles

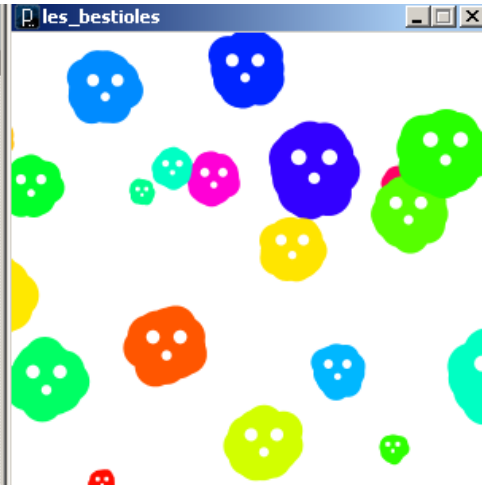
```
// Exemple inspiré de [NOBLE, p58]
int x = 0;
int nbCercles = 10;

void setup(){
  size(200,200);
  background(255);
}

void draw() {
  x += 2;

  for (int i=1; i <= nbCercles; i++) {
    fill(255,(255/nbCercles)*i,0,x*i);
    ellipse(width/2,height/2,x/i,x/i);
  }

  if(x> ((width/2) * nbCercles)) {
    x = 0;
  }
}
```



Exemple 7 : objets [AIX]

les_bestioles | Bestiole

```
// Exemple tiré de : http://www.ecole-art-aix.fr/article2802.html

Bestiole[] bestioles = new Bestiole[20];

void setup() {
  size(300,300);
  colorMode(HSB);      // HSB simplifie les couleurs al□atoires
  noStroke();
  smooth();            // formes anti-aliases

  // pour chaque objet, choisir une taille &position de depart
  for(int i=0; i < bestioles.length; i++) {
    float taille = random(5,30);
    float depart_x = random(width);
    float depart_y = random(height);
    bestioles[i] = new Bestiole(depart_x, depart_y, taille);
```




### 4.8.e - Exemples d'applications

Polissonos : http://web.me.com/ruipenha/en/Software.html
BallDropping : http://balldroppings.com/js/
SodaProcessing : http://processing.org/exhibition/works/sodaprocessing/index_link.html
GlitchSequencer : http://www.glitch-sequencer.com/
Travel Time Tube Map : http://www.tom-carden.co.uk/p5/tube_map_travel_times/applet/

### 4.9 - Choix de Processing

Nous avons choisis *Processing* comme environnement de développement pour plusieurs raisons. D'abord, les connaissances acquises avec le langage de programmation Java sont un atout. Ensuite, c'est un environnement tout à fait adapté pour faire des prototypes et tester des idées. Notre application musicale étant un sujet libre, nous avons pu imaginer des outils de composition. La rapidité de mise en œuvre des fonctionnalités multimédia nous a permis de nous concentrer sur le programme et sur les interactions avec l'utilisateur. Une fois l'environnement maîtrisé, une multitude de bibliothèques sont disponibles. Nous avons essentiellement tiré parti de deux bibliothèques : *proMidi* [PROMIDI] et *controlP5* [CONTROLP5]. La première porte, comme son nom l'indique, sur le protocole MIDI. Elle permet de créer des séquences de notes et de les envoyer au synthétiseur MIDI de l'ordinateur. La seconde est une solution pour contrôler des événements provenant de composants de l'interface graphique comme des boutons et des *sliders* par exemple.

Les méthodes de gestion du mode continu `setup()` et `draw()` sont un mécanisme puissant pour la gestion des animations et des interactions. Elles sont semblables à l'architecture « init()-update()-draw() » que l'on peut voir dans le développement des jeux vidéos [JEUX].

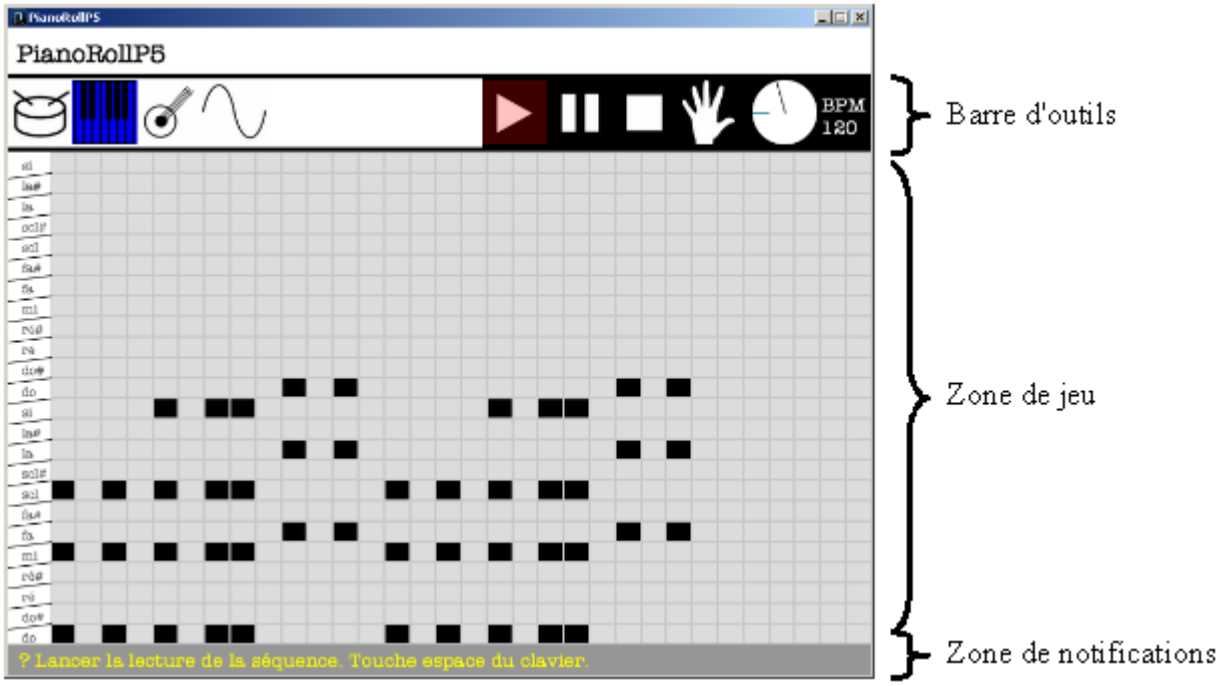
Néanmoins, nous formulons quelques reproches :
- l'IDE ne fournit pas de complétion de mots, la méthode *comment/uncomment* n'est pas très élaborée, le terme « onglets » (*tab*) pour désigner les classes est sans fondement ;
- la simplicité du code, venant du Java, est toute relative. Elle amène vers de mauvaises pratiques de programmation, notamment au niveau du typage, des droits d'accès et de la gestion des classes en onglets. Le premier onglet est souvent long et peu visible, contraire au découpage habituel ;
- il est difficile d'imaginer comment placer une étape de développement sous *Processing* dans d'autres contextes de projets plus importants.

# 5 – Application musicale

Nous avons choisi de concevoir une interface très simple pour ne pas noyer l'utilisateur sous d'innombrables informations. Elle a été construite à partir des notions de consistance ou d'homogénéité. Nous voulions en effet éviter l'utilisation de menus et mettre toutes les fonctionnalités du programme dans une fenêtre.

L'interface est composée de trois parties :

- la barre d'outils ;
- la zone de jeu ou *piano roll* ;
- la zone de notifications.




### 5.1 – La barre d'outils

La barre d'outils contient le choix des instruments, les commandes de lecture et l'activation du capteur de mouvements.

Le choix des instruments se présente sous forme d'icônes, car elles permettent de comprendre rapidement leur fonction. Pour les commandes de lectures, ces icônes sont universelles. Elles sont utilisées dans n'importe quel système de lecture audio ou vidéo, apportant de la compatibilité au programme car l'utilisateur en connaît déjà la signification.

L'activation du capteur de mouvement est symbolisée par l'image d'une main, afin d'associer cette image à la possibilité d'interagir avec sa main pour modifier le tempo de lecture. S'il le désire, il peut activer le capteur de mouvement, l'icône devient rouge, ou le désactiver, l'icône redevient blanche.

### 5.2 – La zone de jeu

La zone de jeu contient une colonne avec des notes sur deux octaves et une matrice de cellules.

Cette colonne n'interagit pas directement mais sert de repère à l'utilisateur par rapport à la matrice, chose absente dans beaucoup de programmes musicaux et pourtant importante pour guider l'utilisateur. Il sait donc quelle note il désire jouer et, si c'est un musicien, il pourra alors visualiser facilement son objectif. Chaque élément de la colonne représente une note, d'une octave en particulier, sur toute une ligne de la matrice.

La matrice est, au contraire, interactive. Une cellule fait référence à une note par rapport au temps. L'utilisateur utilise le bouton gauche de la souris pour activer une cellule ou le bouton droit pour la désactiver. C'est le principe le plus simple d'un point de vue ergonomique, la compréhension de cette fonctionnalité est très rapidement assimilée introduisant la notion de compatibilité car l'utilisateur sait utiliser la souris sur d'autre programme comme le célèbre « Excel ». De plus, cette matrice rappelle les pianos de l'époque qui utilisaient des rouleaux de papier troués. Cette analogie permet à l'utilisateur de comprendre comment le programme parcours la matrice, il intègre alors facilement la notion de temps.

### 5.3 – La zone de notifications

La zone de notifications est un outil explicite, elle introduit la notion de guidage. Elle affiche une explication sur les boutons lorsque l'utilisateur place la souris dessus. Il sait ce sur quoi il va cliquer ainsi que les effets. Cela permet la prévention des erreurs et aide à la compréhension du programme. C'est un outil ergonomique important pour le programme. Sans lui, l'homogénéité serait difficile à respecter et nous obligeraient à ajouter des menus qui rendraient le programme plus complexe et augmenteraient le temps de recherche d'informations.

# 6 – Problèmes rencontrés

La bibliothèque *proMidi* propose des méthodes qui n'ont pas été « débugées », ce qui a ralenti le processus de développement des fonctionnalités Midi. Il aurait fallu se pencher sur le code Java de *proMidi*.

Le projet devait être au départ un logiciel permettant de faire des mixages de sons à distance entre deux musiciens. Cependant, dû à la latence du réseau et à la rigueur du rythme de la musique, il était très difficile de concevoir un tel projet. C’est pour cela que nous avons choisit un projet musical n’utilisant pas le réseau mais avec une interface graphique intuitive avec l'utilisation d'un capteur de mouvement. Cette utilisation du réseau dans les logiciels musicaux apporte un gain encore sous-exploité en offrant un univers de jeu partagé entre plusieurs personnes. Des exemples en ce sens, avec ou sans réseau, peuvent être vus dans les projets NetPd (http://www.netpd.org) et Reactable (http://www.reactable.com).

Un autre aspect du programme, qui n'a pas pu être implémenté, concerne la représentation abstraite de structures musicales. Nous voulions en effet, permettre de choisir entre plusieurs mode de visualisations de ces structures : celle d'un plan vu de haut avec des formes géométriques éditables, ayant chacune leur comportement, ou bien celle d'objets pouvant se connecter entre eux à la manière des langages de programmation graphique *Pure Data, Max/MSP ou Isadora.*

# 7 – Conclusions et perspectives

*PianoRollp5* est un modeste programme musical qui possède néanmoins une ergonomie efficace. Lors de sa conception, la notion d'ergonomie était le principal objectif. D'un point de vue visuel ou physique *pianoRollp5* permet à l'utilisateur de se plonger directement dans le programme sans que la prise en main soit longue et complexe. L'intégration du capteur de mouvement apporte une originalité au programme apportant du changement par rapport à  la souris et au clavier.

Nous avons présenté une suite d'outils accessibles, qui permettent de réaliser des systèmes complexes et ludiques. Les obstacles liés au développement ne sont pas tous franchis, des difficultés subsistent, mais l'apport de ce type de logiciel pour la conception d'interactions, quelles soient logicielles ou matérielles, est indéniable.  Beaucoup de domaines sont mobilisés, l'électronique, l'informatique, le graphisme, la musique, etc. Les compétences y sont transversales, ce qui en fait un vivier d'innovations et d'émancipation.

Les bibliothèques sonores de *Processing*  n'étant pas encore très élaborées, nous pourrions concevoir une autre version du programme en utilisant un logiciel spécialisé dans le son : *Pure Data.* Les trois logiciels *Arduino*, *Processing* et *Pure Data* seraient complémentaires.

# 8 – Bibliographie et références


[ACROE]
Association pour la Création et la Recherche sur les Outils d'Expression,
http://acroe.imag.fr/

[AIX]
http://www.ecole-art-aix.fr/article2802.html

[ARDUINO]
http://www.arduino.cc/
http://www.arduino.cc/en/Guide/Introduction
http://www.craslab.org/interaction/files/LivretArduinoCRAS.pdf
http://hardware.processing.org/

[BEAUDOIN ]
Michel Beaudoin-Lafon, *40 ans d'interaction homme-machine*,
http://interstices.info/jcms/c_23015/40-ans-d-interaction-homme-machine-points-de-repere-et-perspectives

[BIBLIOP5]
http://www.processing.org/reference/libraries

[BIRON]
Jérémie Biron, *Les nouvelles interfaces de création musicale*,
http://diplome.pixylab.com/wordpress/2009/11/07/les-nouvelles-interfaces-de-creation-musicale

[CONTROLP5]
http://www.sojamo.de/libraries/controlP5/

[CREATION]
**Quelques ressources françaises :**
http://codelab.fr
http://www.linuxmao.org
http://ressources.electro.free.fr
http://craslab.org/
http://fr.wikipedia.org/wiki/Portail:Arts_interactifs
http://ressources.levillagenumerique.org

[DBN]
Design By Numbers, logiciel de John Maeda,
http://dbn.media.mit.edu/whatisdbn.html

[DELALANDE]
Table ronde à la fin du livre *Interfaces homme-machine et création musicale*, sous la direction de Hugues Vinet et François Delalande, éditions Hermès, 1999.

[EASTSIDE]
*An Aeolian piano roll: "Eastside, Westside.",*
http://www.edwardsamuels.com/ILLUSTRATEDSTORY/isc2.htm

[FRY]
Ben Fry (traduction de), *Visualizing Data, Exploring and Explaining Data with the Processing Environnement*, éditions O'Reilly, 2008.

[IRCAM]
Institut de Recherche et Coordination Acoustique/Musique,
http://forumnet.ircam.fr/699.html

[JAVAP5]
*Processing in Eclipse* : http://processing.org/learning/eclipse/
*Embedding Papplet into Java Applications* [FRY, p338]
*Embedding in a Swing Application* [FRY, p340]

[JEUX]
http://www.progware.org/Blog/post/XNA-Game-Development-%28First-Steps%29.aspx

[MAKE]
*Open source hardware 2008 - The definitive guide to open source hardware projects in 2008*
http://blog.makezine.com/archive/2008/11/_draft_open_source_hardwa.html

[MIDI]
http://fr.wikipedia.org/wiki/Musical_Instrument_Digital_Interface

[MULTIMEDIALAB]
http://www.multimedialab.be/cours/logiciels/processing.htm

[NIME]
*New Interfaces for Musical Expression*,
http://www.nime.org

[NOBLE]
Joshua Noble (traduction de), *Programming Interactivity, a designer's guide to Processing, Arduino, and openFrameworks*, éditions O'Reilly, 2009.

[PApplet]
http://dev.processing.org/reference/core/

[PROCESSING]
http://www.processing.org
http://codelab.fr/39

[PROXI]
Capteur de proximité infrarouge de chez Interface-Z
http://www.interface-z.com/produits/cs03_proxi.htm

[PROMIDI]
http://creativecomputing.cc/p5libs/promidi/

[VINET]
Hugues Vinet, chapitre 5 *Concept d'interfaces graphiques pour la production musicale et sonore*, tiré du livre *Interfaces homme-machine et création musicale*, sous la direction de Hugues Vinet et François Delalande, éditions Hermès, 1999.

[WANDERLEY]
Marcelo Wanderley, Philippe Depalle, chapitre 7 *Contrôle gestuel de la synthèse sonor*e, tiré du livre *Interfaces homme-machine et création musicale*, sous la direction de Hugues Vinet et François Delalande, éditions Hermès, 1999.