Informatique et Science du Numérique

Thomas Rey

Classe de Terminale S
19 avril 2020

Table des matières

TP 1

Processing

Processing est un environnement de programmation inventé en 2001 par et pour des artistes créant des oeuvres numériques ou multimédias. Il est basé sur le langage Java mais son attrait principal est la simplicité d'utilisation. Ce logiciel est libre, gratuit (téléchargeable à cette adresse https://processing.org/), multiplateforme (c'est-à-dire qu'il fonctionne sous Windows, MacOS et Linux) et il permet de créer facilement des applications pour Android.
Nous allons détailler dans ce document quelques fonctionnalités de base qui nous permettront de bien débuter avec ce logiciel, puis quelques instructions élémentaires de Java pour écrire nos premiers programmes. Mais d'abord, pour illustrer les principes de Processing, nous allons étudier un programme...

1.1 Une activité

Un routier souhaite calculer ses temps de trajets lorsqu'il connait la distance à parcourir et la vitesse moyenne à laquelle il estime qu'il va rouler. Pour cela on lui propose le programme suivant :

```
1  // Importer la bibliothèque Java permettant de créer une boîte de dialogue :
2  import javax.swing.*;
3
4  // Variables :
5  int distance, vitesse; //Entiers
6  String reponses;// Chaînes de caractères
7  float duree;// Décimaux
8
9  void setup() {
10 size(800,250);// Création de la fenêtre
11 background(255);// Fond blanc
12 fill(0);// Écriture en noir
13 textSize(16);// Taille du texte
14
15 text("Calcul d'un temps de parcours",20,20);
16 // Saisie de la distance :
17 reponses = JOptionPane.showInputDialog(null,"Saisir la distance à parcourir (en
       km)", "Dialog", JOptionPane.PLAIN_MESSAGE);
18 distance = parseInt(reponses); // conversion réponse en entier
19 // Saisie de la vitesse :
```

```
20 reponses = JOptionPane.showInputDialog(null,"Saisir la vitesse moyenne prévue (
       en km/h)", "Dialog", JOptionPane.PLAIN_MESSAGE);
21 vitesse = parseInt(reponses); // conversion réponse en entier
22 // Calcul durée décimale :
23 duree = (float)distance / vitesse;
24 // Affichage des résultats dans la fenêtre :
25 text("Durée de parcours (en nombre décimal d'heures) : " + duree,20,50);
26 text("Durée de parcours (en heures-minutes) : " + conversion(duree),20,80);
27 }
28
29 // Fonction qui convertit en h-min
30 String conversion(float t) {
31     String rep;// réponse à renvoyer
32     int h,m;//nbre d'heures et de minutes
33     h = (int)(t);
34     m = (int)((t - h) * 60);
35     rep = h + " heures " + m + " minutes";
36     return rep;
37 }
```

1. Quel est le rôle du groupe de symboles `//` ?
2. Que sont les expressions `distance`, `vitesse`, `reponses`, ... ?
3. Quelle est la signification des mots clés `int`, `String` et `float` qui les précèdent ?
4. Que remarquer sur les fins d'instructions d'un programme Processing ?
5. Quel est le rôle des lignes 15, 25 et 26 ?
6. Quel est le rôle du « `20,20` » à la fin de la ligne 15 ?
7. À quoi sert l'instruction `parseInt` des lignes 18 et 21 ?
8. Comment s'appelle la partie du programme comprise entre les lignes 30 et 37 incluses ?
9. Sur un trajet Chambéry-Annecy par autoroute (50 km), quel temps gagne-t-on à rouler à 135 km/h plutôt qu'à 130 km/h ? (Utiliser ce programme !)

1.2 Démarrer avec Processing

Pour écrire un premier programme simple, par exemple demander le poids et la taille d'un individu et calculer l'IMC (Indice de Masse Corporelle) de cet individu. Nous allons donc créer un *sketch* que nous appelerons `IMC`.
L'algorithme de notre projet est donc celui-ci :

Entrées : m, t et imc sont trois variables numériques;
Sorties : imc contient $\frac{m}{t^2}$;
début
    Demander la masse m ;
    Demander la taille t ;
    Calculer $imc \leftarrow m/t^2$;
Afficher imc

Algorithme 1 : Calcul de l'IMC

Voici comment programmer ce « sketch » avec Processing maintenant :

- lancer le logiciel Processing ;
- dans le menu `File` sélectionner `Préférences` et choisir le dossier où seront enregistrés vos projets : un répertoire `Processing` de votre dossier personnel ;
- dans le menu `File` sélectionner `Save As` et lui donner le nom `imc` ;
- voilà ! Vous pouvez programmer...

Complétez votre programme comme sur le modèle ci-dessous :

```
1  // Importer la bibliothèque Java permettant de créer une boîte de dialogue
2  import javax.swing.*;
3
4  int m, t;// int indique que les variables sont entières
5  float imc;// float indique que la variable est décimale
6  String reponse;
7
8  size( , );
9  textSize( );
10
11 text("Calcul de votre IMC", , );
12 // Demander la taille :
13 reponse = JOptionPane.showInputDialog(null,"Saisissez votre taille en cm", "
       Taille", JOptionPane.PLAIN_MESSAGE);
14 t = parseInt(reponse);
15
16 // Demander la masse :
17
18
19 // Calculer l'IMC
20 imc =
21
22 // Afficher le résultat dans la fenêtre :
23 text(
```

Remarques :

- tout ce qui suit un « `//` » n'est pas lu par Java (et donc par Processing) : il s'agit de commentaires du programme (ils sont essentiels pour la bonne compréhension du lecteur) ;
- par convention, une *variable* commence toujours par une lettre minuscule ;

Pour exécuter le programme, enregistrer le sketch puis cliquer sur l'icône de lecture (le triangle).

1.3 Un peu plus loin...

1.3.1 Quelques notions sur les méthodes

En règle générale, un projet informatique est beaucoup plus long que ces quelques lignes. Il doit donc être organisé en plusieurs parties bien distinctes ; chacune réalisant une opération la plus élémentaire possible.

Dans nos (modestes) projets, nous utiliserons des *classes* (bientôt) pour définir des nouveaux *objets* et nous décomposerons nos programmes en créant des *méthodes*.

Une méthode est une sorte de sous-programme qui effectue une tâche (plus ou moins) élémentaire. Cette tâche peut être de deux types :

- soit elle renvoie un résultat de type `int`, `String`, ... ; on parle alors de *fonction*, c'était le cas de la méthode conversion de l'activité de départ (lignes 30 à 37) qui était de type `String` ;
- soit elle ne renvoie pas de résultat (elle peut par exemple afficher une information à l'écran) ; on dit parfois que c'est un sous-programme (dans ce cas on précède le nom de la méthode du mot clé `void`).

Exemple : une méthode de type `int` qui renvoie la somme des deux entiers passés en paramètres :

```
1 int addition(int a, int b) {
2   return a+b;
3 }
```

Dans notre programme on pourra écrire `int` somme = addition(2,3); pour que la variable `somme` contienne la valeur 5.

Exemple : méthode qui affiche la somme de deux entiers (et qui utilise la méthode de l'exemple précédent).

```
1 void affiche(int a, int b) {
2   text(addition(a,b), 30,30);
3 }
```

À noter le mot clé `void` qui indique que la méthode ne renvoie pas de résultat (ce n'est pas une « fonction »).
Attention : une variable qui est définie dans une méthode n'est « connue » et donc utilisable que dans cette méthode. Dans l'activité de départ, la variable `h` n'est pas connue dans le programme principal (dans la méthode `setup`()).

1.3.2 Quelques méthodes particulières

Il existe en Processing (au moins) deux méthodes particulières :

- la méthode `setup`() qui, si elle existe, est lancée automatiquement au démarrage du programme ;
- la méthode `draw`() qui, si elle existe, est lancée automatiquement un certain nombre de fois par seconde et sert à gérer les affichages.

Ces deux méthodes sont de type `void`.
Le nombre de fois où la méthode `draw`() est exécutée par seconde est défini par l'instruction `frameRate`(i) où `i` est un entier.

1.4 Exercices

Exercice 1.1.
Dans chaque cas écrire un programme qui :
- demande le prénom de l'utilisateur et affiche « bonjour » suivi du prénom ;
- demande le prénom, le nom, le numéro de téléphone, ... et affiche une phrase avec ces éléments ;
- demande deux nombres et affiche le résultat des quatre opérations élémentaires ;

Exercice 1.2.
Créer une fonction qui prend en paramètre une question (de type `String`) et retourne un entier qui répond à la question.
Modifier alors le programme `imc` en utilisant cette fonction.

Exercice 1.3.
Modifier le programme de la première activité pour qu'il demande :
- la distance ;
- une vitesse 1 ;
- une vitesse 2 supérieure à la vitesse 1.

Puis qu'il affiche :
- le temps de trajet en heures-minutes à la vitesse 1 ;
- le temps de trajet en heures-minutes à la vitesse 2 ;
- le temps gagné en roulant à la vitesse 2 plutôt qu'à la vitesse 1.

TP 2

Variables

Dans le premier TP, nous avons déjà utilisé des *variables*, nous allons préciser ici leur utilisation, les différents types simples, mais aussi nous allons commencer à découvrir ce qui fait la force des langages *orientés objet* comme Java (et donc Processing) : les *classes* qui permettent au programmeur de créer ses propres objets.

2.1 Variables

2.1.1 Définition

Une *variable* est un espace mémoire dans lequel le programme peut stocker de l'information qui peut être numérique, booléenne, alphanumérique, ... On peut « modéliser » une variable comme étant une boîte qui contient *une* information qui peut varier (d'où son nom !) au cours de l'exécution du programme. À noter :

- il existe plusieurs *types* de variables : entière, décimale, alphanumérique, ... ; nous les décrirons plus loin ;
- en Java (et donc en Processing [1] !), une variable doit être *définie* avant d'être utilisée ;
- une fois définie, la variable ne peut pas changer de type (si vous définissez une variable *entière* `maVariable`, elle ne peut pas contenir le décimal 2,5) ;
- concrètement, une variable est désignée par une suite de lettres et de chiffres ainsi que certains caractères (par exemple `_` et `-`) et commence toujours par une lettre minuscule (attention les variables `maVar1` et `mavar1` sont différentes : le nom de variable est sensible à la *casse*).

Pour résumer une variable est constituée de trois informations :

un identificateur : c'est-à-dire son nom ;

un type : c'est-à-dire la description des informations qu'elle contient ;

une valeur : c'est-à-dire son contenu.

1. Processing étant basé sur Java, la plupart des consignes valables pour Java le sont aussi pour Processing. Je ne le répéterai plus à chaque fois...

2.1.2 Les types simples en Java

Nous allons décrire ici les principaux types « primitifs » :

- le type `int` : permet de définir un nombre entier compris entre -2^{31} et $2^{31}-1$ (il est codé sur 32 bits, c'est généralement suffisant [2]) ;
- le type `long` : permet de définir un nombre entier compris entre -2^{63} et $2^{63}-1$ (il est codé sur 64 bits et là c'est carrément énorme [3]) ;
- il existe aussi les types bytes et `short` codés respectivement sur 8 et 16 bits ;
- le type `float` : permet de définir des nombres décimaux (on dit à virgule *flottante*) ;
- les types réels (ou plus précisemment décimaux en maths) sont `float` et `double` (le second étant plus précis) ;
- le type `char` permet de déclarer un caractère par exemple 'a' ou '$' ou ... ; il est codé sur 16 bits ;
- le type `boolean` qui ne peut prendre que deux valeurs : « VRAI » ou « FAUX ».

Exemple : déclaration de variables en Java

```
1    int maVariable, h, monCompteur3;// déclaration "simple"
2    int monAge = 17;// déclaration et initialisation
3    float x;
4    double y;
```

Attention : il est toujours préférable d'initialiser la valeur d'une variable pour éviter des erreurs de compilation.

2.1.3 Affectations et opérations

Quelques exemples :

- affecter la valeur 5 à la variable `a` s'écrit : `a = 5;`
- augmenter la valeur de la variable `x` de 3 s'écrit : `x = x + 3;` ou encore `x += 3;`
- donner à une variable le résultat d'un calcul : `a = 4 + 3 * 5;` et la variable `a` contiendra 19 (priorités opératoires)
- incrémenter la variable `i` (c'est-à-dire l'augmenter de 1) : `i++;`
- tester une égalité `boolean vf = (a == b);` met `true` dans la variable `vf` si les deux variables `a` et `b` ont la même valeur et `false` sinon.

À noter que le type `char` est particulier car on peut effectuer des opérations de comparaison dessus ; par exemple :

```
1 char a = 'a';
2 char b = 'e';
3 println(a < b);
```

Exercice 2.1.
Quel sera l'affichage suite à l'exécution du programme précédent ?
Même question en remplaçant le « `'e'` » de la ligne 2 par un « `'E'` » ?

2. Pour info, $2^{31} = 2\ 147\ 483\ 648$.
3. $2^{63} = 9\ 223\ 372\ 036\ 854\ 775\ 808$

Enfin, il est possible d'effectuer des conversions ou *cast* de types. Par exemple l'expression suivante est incorrecte : `int a = 12.7;` il faut écrire `int a = (int)12.7;` et la variable `a` contiendra la valeur 12 (troncature) et `(int)-12.7` donne -12 (la conversion des décimaux en `int` n'est donc pas la même chose que la partie entière en maths !)
Attention : l'instruction `float a = 5 / 4;` donne à `a` la valeur 1 (et non pas 1,25) car 5 et 4 sont des entiers, Java calcule donc avec eux comme si le résultat était entier. Il faut écrire (au minimum) : `float a = 5f / 4;`
De manière générale, lorsqu'on initialise un `float`, on lui ajoute un « `f` » à la fin de sa valeur numérique ; par exemple le code suivant ne pose pas de problème :

```
1 float a = 5f;
2 float b = 4f;
3 float q = a/b;
```

Exemple : on peut facilement retrouver le « code » de chaque caractère en castant en `int` un `char`. Que fait le programme suivant :

```
1 char a = 'a';
2 int code = (int)a;
3 code++;
4 a = (char)code;
```

Exercice 2.2.
Retrouver le code du caractère `a`, du `e` et celui du `E` et expliquer le résultat de l'exercice 2.1.

2.1.4 Les chaînes de caractères

Un texte est appelé en informatique *chaîne de caractères*. En Java, une chaîne ou `String` n'est pas un type *simple* comme `int`, `float`, ... La définition d'une chaîne se fait avec le mot clé `String` (on remarquera la majuscule au début...) :

```
1 String maChaine = "Bonjour, ";
2 String monNom = "Bill Gates";
3 println(maChaine + monNom);
```

Quelques opérations sur les chaînes (dans le tableau suivant str, str1, str2 sont des chaînes de caractères) :

Méthode	Description
str.length();	donne la longueur (le nombre de caractères) de la chaîne `str`
str = str1 + str2;	*concatène* les deux chaînes (elles sont mises « bout à bout »)
char c = str.charAt(4);	donne le 5e caractère (celui d'indice 4) de la chaîne str
int i = str.indexOf('c');	donne l'indice de la première apparition du caractère c
str = str1.substring(3,6) str1="Chocolat"; str=str1.substring(3,6)	extrait de la chaîne str1 les caractères d'indices 3 à 5 inclus str contient "col"

Méthode	Description
str1.equals(str2);	donne un booléen qui permet de tester si les deux chaînes ont le même contenu
(str1 == str2)	donne un booléen qui permet de tester si les deux chaînes sont les mêmes
str.toUpperCase();	met toute la chaîne str en majuscules
str.toLowerCase();	met toute la chaîne str en minuscules
str1.compareTo(str2);	compare dans l'ordre lexicographique (voir aussi str1.compareToIgnoreCase(str2);)

Exercice 2.3.
Voici un morceau de programme :

```
1  String str1, str2, str3;
2  str1 = "Voici un exemple formateur";
3  str2 = str1;
4  str3 = new String(str1);
5  println("chaine 1 = " + str1);
6  println("chaine 2 = " + str2);
7  println("chaine 3 = " + str3);
8
9  println("chaine 1 et chaine 2 ont le même contenu ? " + str1.equals(str2));
10 println("chaine 1 et chaine 2 sont le même objet ? " + (str1==str2));
11
12 println("chaine 1 et chaine 3 ont le même contenu ? " + str1.equals(str3));
13 println("chaine 1 et chaine 3 sont le même objet ? " + (str1==str3));
```

Question : où seront les `true`, où seront les `false` dans l'affichage ?

2.2 En mémoire

Dans la mémoire de l'ordinateur chaque donnée est stockée sous la forme d'une suite de 0 et de 1. Nous allons voir dans cette partie comment, à partir de 0 et de 1, stocker des informations aussi diverses que des entiers, des décimaux, du texte, ...
Le bit est une valeur 0 ou 1 (qui correspond à « le courant ne passe pas » ou « le courant passe » dans le circuit). Grâce à un bit, on peut donc compter jusqu'à deux (en commençant à un).
En utilisant le principe de la numération de position, on peut ensuite écrire des nombres plus grands : `0`, `1`, `10`, `11`, `100`, `101`, ... (on dit qu'on compte en base 2).
Un octet (ou byte) est un regroupement de 8 bits. Il permet donc de stocker les nombres de `0` à `11111111` soit 255 (en base 10).
En Java, le type `int` est codé sur 4 octets, le type `long` sur 8 octets et le type `float` sur 4 octets (il existe aussi le type `long` codé sur 8 octets).
Exercice 2.4.
Expliquer pourquoi un `int` est compris entre $-2\,147\,483\,648$ et $2\,147\,483\,647$.

Pour stocker les caractères, il a fallu inventer un codage. Le premier codage des caractères fut le code ASCII (American Standard Code for Information Interchange). Ce codage associait à

chaque caractère de l'alphabet latin (et de quelques symboles) une valeur numérique entre 0 et 255 :

code	caractère	code	caractère	code	caractère
34	"	48	0	97	a
35	#	49	1	98	b
36	$	50	2	99	c
37	%	65	A	61	=
38	&	66	B	64	@

Cette table (ici incomplète), est insuffisante pour désigner tous les caractères de tous les alphabets. Depuis, l'*Unicode* a été inventé, il contient près de 110 000 caractères. Il existe en plusieurs déclinaisons : *UTF-8*, *UTF-32*, ...
Une table des principaux caractères de la norme `iso-latin-1` est disponible en annexe A

2.3 Objets

Dans cette partie, nous allons survoler la notion d'objet. Ceci vous permettra de comprendre certains programmes que vous pouvez trouver sur internet. Nous y reviendrons plus en détails plus tard dans le TP 10.
Un objet en Java est une sorte de « super-variable » qui est défini par une *classe*. Cette classe est une sorte de description des éléments communs de tous les objets de cette classe par exemple dans un jeu de paris sur des courses de chevaux, il faudrait créer une *classe* `Joueur` grâce à laquelle nous définirions des *objets* `joueur1`, `joueur2`, ... Ces objets de type `Joueur` auraient tous :

- un nom (mais chacun le sien) ;
- un capital ;
- une mise ;
- un cheval choisi ;
- ...

Ces éléments sont appelés les *variables de la classe* `Joueur`. Mais ils auront aussi des *méthodes* qui permettent de créer des sortes d'actions sur le joueur :

- créditer le capital en cas de victoire ;
- débiter le montant du pari ;
- choisir un cheval ;
- ...

Exemple : un extrait d'une classe `Joueur` :
Cette classe est définie par le mot clé `class`. Elle possède trois variables `nom`, `capital` et `mise`. Une *méthode constructeur* est définie : c'est le code qui est executé lorsqu'un objet de type `Joueur` est *instancié* c'est-à-dire lorsqu'on crée une variable de ce type dans le programme. Et enfin, elle est munie d'une *méthode* (nous reviendrons plus tard sur les méthodes) qui permet d'augmenter la variable `capital`.

```
1 public class Joueur {
2   String nom = new String("Joueur");
3   int capital = 20;// Capital de départ
```

```
4   int mise = 0;
5
6   // Méthode "constructeur" :
7   public Joueur(String nomSaisi, int sous) {
8     // Ci-dessous on "construit" le joueur avec les données de départ :
9     this.nom = nomSaisi;
10    this.capital = sous;
11  }
12
13  // On crédite en cas de victoire
14  public void gagne(int gain) {
15    this.capital = this.capital + gain;
16    this.gain = gain;
17  }
18
19 }// Fin de la classe Joueur
```

Devinette : à quoi peut bien servir le mot clé `this` ?
Dans un programme qui utilise la classe `Joueur` on aurait par exemple :

```
1 // Création du joueur 'joueur1' qui s'appelle Thomas et possède 150000 euros
2 Joueur joueur1 = new Joueur("Thomas", 150000);
3 // On lui crédite 10 euros de plus
4 joueur1.gagne(10);
```

2.4 Exercices divers

Exercice 2.5.
On donne le code :

```
1 float x = 3;
2 float y;
3 y = y + 2;
4 y = y ** 2;
5 y = 2 * y;
6 println(y);
```

Que contient `y` à la fin du programme ? Décrire à quoi il sert.

Exercice 2.6.
Que contient `c` à la fin de l'exécution du programme suivant ?

```
1 int a = 3, b = 5, c;
2 c = a * (b / a);
```

Exercice 2.7.
Écrire un programme qui demande la longueur d'un rayon et qui calcule :
- le périmètre du cercle correspondant ;
- l'aire du disque correspondant.

Exercice 2.8.
Que fait le morceau de programme suivant (`a` et `b` sont des variables de type `int`) :

```
1 a = a + b;
2 b = a - b;
3 a = a - b;
```

On pourra utiliser les tableaux qu'on complétera – d'abord avec des valeurs numériques puis avec des inconnues (au sens mathématiques) x et y – au fur et à mesure des étapes :

a	b

a	b

a	b

Exercice 2.9.
Écrire un programme qui demande le nom et le prénom de l'utilisateur et l'écrit ensuite sous la forme : « NOM Prénom » (en respectant la casse).

Exercice 2.10.
Voici le code d'un programme :

```
1 String mot = "anticonstitutionnellement";
2 int n = 0;
3 n = mot.indexOf('t'); println(n);
4 n = mot.lastIndexOf('t'); println(n);
5 n = mot.indexOf("ti"); println(n);
6 n = mot.lastIndexOf("ti"); println(n);
7 n = mot.indexOf('x'); println(n);
8 n = mot.indexOf('t',5); println(n);
```

Quel sera le résultat de ce programme ?

Exercice 2.11.
Retrouver le texte représenté en ASCII binaire par la suite de bits suivants :
`01000011001001110110010101110011011101000010000001100110011000010110001101101001 0110110001100101`

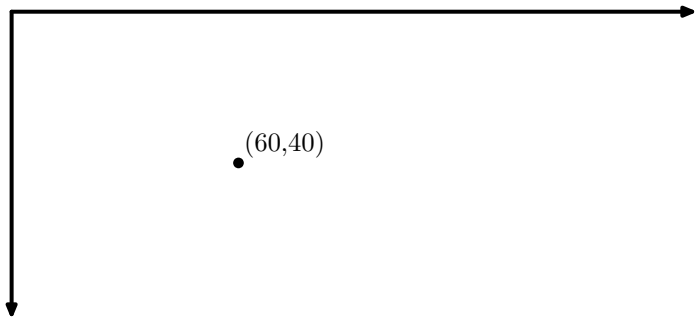
TP 3

Dessiner à l'écran

Dans le TP 1 nous avons vu l'instruction `size`(larg,haut) qui permet de créer une fenêtre graphique de dimensions `larg` et `haut`. Nous avons vu aussi que l'instruction `text`(*"mon texte"*,20,30) permet d'afficher `mon texte` au point de coordonnées $(20;30)$ dans cette fenêtre. L'objet de ce TP est maintenant de dessiner à l'écran des lignes, cercles, disques, polygones, ... et par la suite d'animer ces dessins.

3.1 Repérage sur l'écran

Le système de repérage de la fenêtre graphique utilise le pixel comme unité et le repère a pour origine le coin supérieur gauche de la fenêtre. Le premier axe est horizontal et dirigé vers la *droite*, le second est vertical est dirigé vers le *bas*. À noter que les coordonnées des points et les dimensions utilisées sont des nombres *entiers*. Enfin, ces coordonnées sont séparées par une virgule (et non pas un point-virgule comme en mathématiques françaises).



Pour une fenêtre définie par `size`(200,100), les abscisses vont de 0 à 199 et les ordonnées de 0 à 99.
Une particularité de Processing est qu'il permet aussi de gérer la 3D. Dans ce cas, l'instruction `size`() prend un troisième argument : `P3D`. Pour définir une zone de dessin en 3D on écrit par exemple `size`(600,400,P3D).
Enfin, pour en finir avec les généralités, à tout moment du programme, on peut connaître la largeur et la hauteur de la fenêtre à l'aide des mots-clés `width` et `height`. Ceci peut être pratique pour repérer le milieu de l'écran (qui aura pour coordonnées (`width`/2,`height`/2)).

3.2 Les formes de base

3.2.1 Le point

Pour placer le point de coordonnées (30; 40) à l'écran, on écrit la commande :

```
1 point(30,40);
```

Attention, et ceci est vrai pour toutes les formes, il faut que les coordonnées du point soient entre 0 et `width`−1 pour l'abscisse et entre 0 et `height`−1 pour l'ordonnée ; sinon on ne le voit pas ! (Mais Processing ne vous renverra pas d'erreur).

3.2.2 La ligne droite

On trace un segment [AB] par l'instruction :

```
1 line(xA,yA,xB,yB);
```

3.2.3 Le rectangle

On trace un rectangle de sommet A, de largeur `larg` et de hauteur `haut` ainsi :

```
1 rect(xA,yA,larg,haut);
```

Il est aussi possible de configurer Processing pour que le point A soit l'intersection des diagonales :

```
1 rectMode(CENTER);// A ecrire une seule fois
2 rect(xA,yA,larg,haut);
```

Et pour revenir à la situation initiale (A est un sommet du rectangle) :

```
1 rectMode(CORNER);// A ecrire une seule fois
2 rect(xA,yA,larg,haut);
```

Il existe deux autres modes : CORNERS (on donne les coordonnées de 2 sommets opposés) et RADIUS (on donne en paramètres 3 et 4 la moitié de la largeur et de la hauteur).

3.2.4 L'ellipse

Comme pour le rectangle, il existe plusieurs modes : CENTER (mode par défaut), CORNER (même principe que pour les rectangles), RADIUS et CORNERS.

```
1 // ellipse de centre (50,50) de diamètre horiz 60 et vertical 20
2 ellipse(50,50,60,20);
3 // Mode RADIUS : cercle de centre (100,50) et Rayon=20
4 ellipseMode(RADIUS);
5 ellipse(100,50,20,20);
6 // Mode CORNER : ellipse enfermée dans rectangle de sommet (50,100)
7 // et de dimension larg=30, haut=30 (le rectangle n'est pas tracé)
8 ellipseMode(CORNER);
9 ellipse(50,100,30,50);
```

```
10 // Mode CORNERS : on donne les 2 sommets opposés du rectangle
11 ellipseMode(CORNERS);
12 ellipse(100,100,150,120);
```

3.2.5 Le triangle et le quadrilatère

Le triangle ABC et le quadrilatère ABCD se tracent ainsi :

```
1 triangle(xA,yA,xB,yB,xC,yC);
2 quad(xA,yA,xB,yB,xC,yC,xD,yD);
```

3.2.6 Arc d'ellipse

Un arc d'ellipse de centre $(50, 50)$, de largeur 80 et de hauteur 80 (pour l'ellipse complète) pour un angle entre $\frac{\pi}{2}$ et π (donc ici un quart de cercle inférieur gauche) se trace ainsi :

```
1 arc(50,50,80,80,PI/2,PI);
```

3.2.7 Remplissages et contours

Toutes les figures tracées jusqu'à présent ont un contour et sont remplies.
Pour modifier la couleur de remplissage :

```
1 fill(R,G,B,t);
2 // R, G, B quantités de rouge, vert et bleu entre 0 et 255
3 // t : degré de transparence (0:transparent, 255:opaque)
4 fill(N);
5 // N étant entre 0 et 255 (0=noir, 255=blanc)
6 fill(#hhhhhh);
7 // hhhhhh : code hexadécimal de la couleur entre 0 et ffffff
8 noFill();// Pas de remplissage
```

La couleur de fond se modifie à l'aide de :

```
1 background(R,G,B);
```

Attention : modifier le fond après avoir construit des figures les efface !
Pour supprimer le contour, on commence par l'instruction :

```
1 noStroke();
```

Pour modifier la couleur et l'épaisseur du contour :

```
1 stroke(R,G,B,t);
2 strokeWeight(2);
```

3.2.8 Portée des modifications de style

À chaque fois qu'on effectue une modification de style (couleurs, contours, ...) cette modification s'applique à toutes les constructions suivantes. Pour modifier temporairement un style, il faut encadrer les commandes par `pushStyle()` et `popStyle()` :

```
size(100,100);
background(255);
// Style initial :
stroke(0);
fill(127);
strokeWeight(1);
rect(10,10,10,10);

pushStyle(); // On ouvre une parenthèse de style
  stroke(255,0,0);// contour rouge
  fill(0,0,255);// remplissage bleu
  strokeWeight(5);// grosses bordures
  rect(30,10,10,10);
popStyle();// Fin de la parenthèse

rect(10,30,10,10);// carré avec les paramètres de style initiaux
```

3.3 Les transformations

Avec les formes de base, il n'est pas possible de dessiner un rectangle « oblique » par exemple. Dans cette partie nous allons voir comment utiliser des transformations géométriques pour y remédier.
Nous avons vu que dans la configuration initiale, le repère a des axes horizontaux et verticaux et une origine dans le coin supérieur gauche de l'écran. C'est en modifiant ce repère que nous allons réussir à effectuer les transformations géométriques.

3.3.1 Déplacer

La modification de l'origine se fait grâce à la commande `translate` qui prend deux arguments : le décalage horizontal puis le décalage vertical.

Exercice 3.1.

On donne le programme suivant :

```
size(200,200);
fill(0);
rect(50,50,50,50);
// On change d'origine :
translate(50,50);
fill(127);
rect(50,50,50,50);
```

Quelle figure obtient-on ?

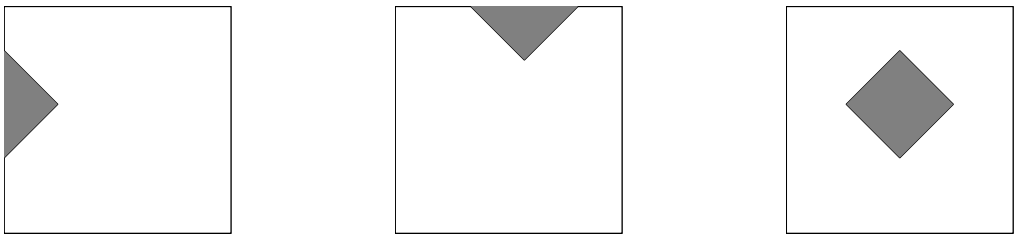
3.3.2 Tourner

De la même façon, il est possible de faire tourner notre repère initial d'un certain angle grâce à la commande `rotate` qui prend un argument : l'angle exprimé en radians. Il existe néanmoins deux commandes permettant de faire les conversions : degrees qui convertit son argument en degrés et radians qui convertit son argument (une mesure en degrés) en radians.
Attention : le sens de rotation est le sens des aiguilles d'un montre (et non pas le sens trigonométrique).

Exercice 3.2.
On donne le programme suivant :

```
1 size(200,200);
2 rotate(PI/4);
3 fill(127);
4 rect(50,50,50,50);
```

Qu'obtient-on à l'écran ?



3.3.3 Mise à l'échelle

La commande permettant d'agrandir ou de réduire une construction est `scale`. Elle prend un ou deux paramètres :

- `scale`(2); agrandira les futures constructions d'un facteur 2 (en maths, on appelle cela une homothétie) ;
- `scale`(0.5,2); divisera par deux en largeur mais multipliera par deux en hauteur (en maths on appelle cela une succession de deux affinités orthogonales).

3.3.4 Provisoire ou définitif ?

De même que pour la gestion des styles (couleurs, ...) il est possible de rendre provisoire un changement de repère, pour cela, on entoure les instructions concernées des instructions `pushMatrix`(); et `popMatrix`();.

```
size(200,200);
//On place l'origine du repère au centre de l'écran :
translate(width/2,height/2);
ellipseMode(RADIUS);
ellipse(0,0,20,20);
// On change provisoirement de repère (décalage vers le haut) :
pushMatrix();
  translate(0,-20);
  rect(-20,0,40,10);
popMatrix();// On revient au repère centré
ellipse(0,60,20,40);
```

3.4 Animations

Le principe d'une animation en Processing est le même que celui d'un dessin animé (un vrai avec un petit carnet dans lequel on fait défiler les pages rapidement). C'est-à-dire qu'on fait un dessin à une position, puis on efface l'écran et on refait le dessin un peu plus loin (ou un peu modifié).
Pour cela, nous allons utiliser une *méthode* dans le programme : la méthode `draw`(). Cette méthode a la particularité de se lancer automatiquement x fois par seconde où x est une valeur qu'on définit grâce à l'instruction `frameRate`(x) (remplacer x par la valeur souhaitée).
Par ailleurs, lorsqu'un programme contient la méthode `draw`(), il est conseillé aussi de créer la méthode `setup`() qui se lance au démarrage du programme pour effectuer par exemple toutes les initialisations.
Enfin, il est possible d'arrêter le rafraîchissement de l'écran grâce à l'instruction `noLoop`(). Dans ce cas, on peut actualiser l'ffichage « manuellement » grâce à l'instruction `redraw`() (valable à partir de la version 3.0 de Processing). Pour relancer le processus de rafraîchissement automatique, on écrira l'instruction `loop`().

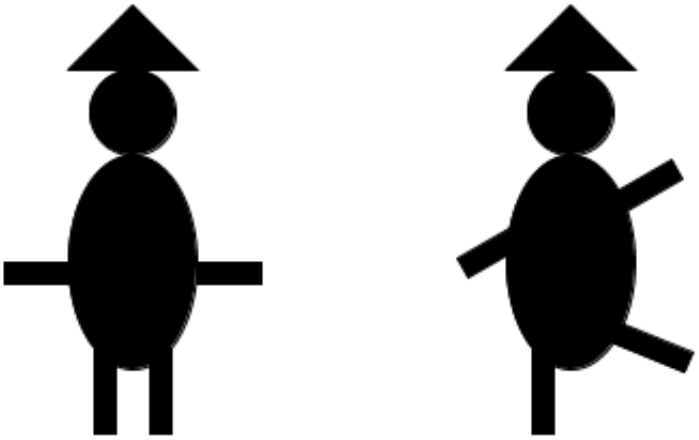
Exemple : une balle qui parcourt l'écran (et puis s'en va...)

```
int x=0;
void setup() {
  size(800,400);// taille fenêtre
  frameRate(50);// Rafraichissement : 50 x / seconde
}
void draw() {
  background(255);// Fond blanc
  fill(0);// Remplissage noir
  ellipseMode(RADIUS);// Mode Radius pour les ellipses
  ellipse(x,200,25,25);
  x++;// Incrémentation de x
}
```

Exercice 3.3.
Écrire le programme qui anime une balle qui tombe (de haut en bas).
Variante : en respectant la loi de gravitation (sans frottement).

3.5 Exercices

Exercice 3.4.
Reproduire les figures ci-dessous (couleurs laissées au choix de chacun) :



Exercice 3.5.
Écrire un programme qui simule les rebonds d'une balle contre un mur puis l'autre (partir de l'exemple de la partie 3.4, mais la balle ne doit plus sortir de l'écran !). On pourra chercher de l'information sur l'instruction `if` que nous verrons ultérieurement.

Exercice 3.6.
Même exercice que le précédent mais avec une balle qui rebondit sur les quatre murs (avec une vitesse initiale « oblique » aléatoire).

Exercice 3.7.
Même exercice que le précédent avec un « trou » dans un mur qui permet à la balle de « s'échapper » et qui affiche « balle perdue » lorsque ceci arrive.

Exercice 3.8.
Expression artistique libre : concours du plus beau dessin en Processing...

Exercice 3.9.
Écrire un programme qui simule la chute d'une balle (en respectant la gravitation) et qui rebondit au sol de façon amortie (à chaque rebond, la balle perd par exemple 10% de sa vitesse. Quelques aides :

- Sur Terre, l'accélération est constante, par exmple, $a = 1$;
- à chaque itération (à chaque frame), la vitesse est augmentée de a : $v = v + a$;
- à chaque itération, la position est augmentée de la vitesse : $y = y + v$;
- lorsque la balle touche le sol, on change le signe de v : $v = -v$.

TP 4

Instructions de base

4.1 Quelques instructions en Processing

4.1.1 Introduction

Depuis le début de l'année, nos programmes sont constitués d'une suite d'instructions élémentaires qui réalisent chacune une tâche bien définie (un affichage, un changement de couleur, une dessin, ...). Aujourd'hui, nous allons voir quelques instructions permettant de répéter plusieurs fois une même séquence d'instructions et de tester si une condition est remplie pour réaliser ou non une séquence du programme.

4.1.2 Les boucles

Une boucle « while » :

```
1 while(condition) {
2   instructions;
3 }
```

Une boucle « do ... while » :

```
1 do {
2   instructions;
3 }
4 while(condition);
```

Une boucle « for » :

```
1 for(initialisation; test; incrémentation) {
2   instructions;
3 }
```

Exemple : `for(int i=0; i<10; ++i)`

Exercice 4.1.
Expliquer en quelques mots les différences entre une boucle `for` et une boucle `while` ou `do` ... `while`.
Expliquer ensuite la différence entre une boucle `while` et une boucle `do` ... `while`

4.1.3 Instructions conditionnelles

Un test « if » :

```
1 if (condition) {
2   instructions si vrai;
3 }
4 else {
5   instructions si faux;
6 }
```

Le `else` est facultatif et on peut éventuellement le remplacer par un elseif (conditions) { voire plusieurs...

Un « branchement conditionnel » :

```
1 switch(expression entière) {
2   case cas1 : instruction1; break;
3   case cas2 : instruction2; break;
4   ...
5   default : instruction;
6 }
```

Le cas `default` est facultatif. Ce branchement conditionnel sera très important dans vos projets pour gérer les différentes fenêtres de votre application.

4.1.4 Nombres aléatoires en Processing

Dans les exercices proposés ensuite, il pourra être utile d'obtenir un nombre aléatoire. Voici les différentes manières d'en obtenir :
`random`(50) renvoie un nombre de type `float` compris entre 0 (inclus) et 50 (exclu).
`random`(10,20) renvoie un nombre de type `float` compris entre 10 (inclus) et 20 (exclu).
(`int`)`random`(1,7) renvoie un entier entre 1 et 6 (inclus).

4.2 Exercices

4.2.1 Avec des nombres...

Exercice 4.2.
Dans chaque cas écrire un programme qui :

1. teste la parité d'un entier saisi par l'utilisateur (`a % b` donne le reste de la division de `a` par `b`) ;
2. écrit la liste des diviseurs d'un entier non nul saisi par l'utilisateur ;

3. demande le nombre n de notes, puis demande les n notes et enfin, calcule la moyenne ;
4. demande des notes et s'arrête à la première note négative puis calcule la moyenne des notes positives ou nulles.

Exercice 4.3.
Écrire l'algorithme (puis le programmer avec Processing) qui permet de calculer les termes de la suite de Syracuse jusqu'à obtenir un terme égal à 1. Afficher alors le nombre de termes calculés.

Exercice 4.4 (Cluedo)**.**
L'exercice consiste à poser des questions à l'utilisateur qui pense à un personnage. Le programme doit deviner à qui l'utilisateur pense grâce aux réponses aux questions posées. L'utilisateur ne peut répondre aux questions posées que par oui ou par non.
Les personnages sont : Mlle Rose, le Professeur Violet, le Colonel Moutarde, le Révérend Olive et Me Leblanc.
Seul le Colonel Moutarde a des moustaches, tous portent des lunettes sauf Mlle Rose, et le Professeur Violet est le seul à avoir un chapeau.
Récupérer le fichier `Cluedo.pde` et compléter le programme de deux façons différentes.

1. D'abord, en laissant le programme poser toutes les questions et ensuite les traiter entre les deux zones de commentaires.
2. Ensuite en modifiant le programme (y compris en dehors des deux zones de commentaires) pour ne poser que trois questions au maximum.

4.2.2 Avec des textes

Exercice 4.5.
Écrire un programme qui demande une chaîne de caractères et la ré-écrit en passant à la ligne après chaque caractère.

Exercice 4.6.
Écrire un programme qui demande une chaîne de caractères et la ré-écrit en transformant les majuscules en minuscules et réciproquement.

Exercice 4.7.
Écrire un programme qui demande un mot de passe contenant au moins une majuscule, une minuscule et un chiffre et teste si c'est bien le cas.
Même question mais avec en plus un caractère spécial.

Exercice 4.8.
Écrire un programme qui demande une chaîne de caractères la transforme en majuscules, remplace les lettres accentuées par leurs homologues sans accent et supprime tous les caractères non alphabétiques (sauf les espaces).

Exercice 4.9.
Écrire un programme qui demande une chaîne de caractères et compte :

— le nombre d'espaces ;
— le nombre d'occurences d'une lettre à choisir ;
— le nombre de voyelles.

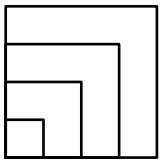
Exercice 4.10.
Cryptographie...

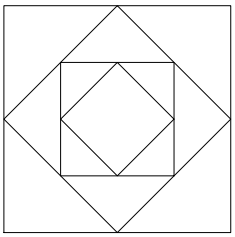
1. Écrire un programme qui permet de crypter/décrypter avec le chiffre de César.
2. Même question avec un codage affine.

4.2.3 Avec des graphiques

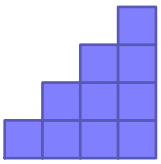
Les exercices suivants sont à réaliser en un minimum de lignes de code (hors lignes de commentaires indispensables...) en utilisant des boucles.

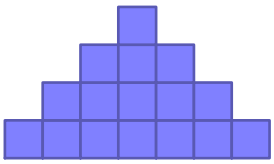
Exercice 4.11.
Réaliser les figures suivantes à n carrés :



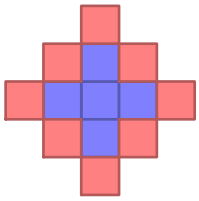


Exercice 4.12.
Réaliser les escaliers suivants à n marches :





Exercice 4.13.
Réaliser la figure suivante où la ligne la plus grande contient $2n+1$ carrés (avec n est un entier naturel) :



TP 5

Tableaux - Lecture de fichier

5.1 À retenir

5.1.1 Déclarer un tableau

Pour déclarer un tableau de 16 entiers de type `int` on peut utiliser le modèle suivant :
`int montableau[] = new int[16];`
Pour définir un tableau à deux dimensions :
`int tabmult[][] = new int[4][3];`

5.1.2 Taille d'un tableau

Pour déterminer la taille d'un tableau unidimensionnel, on utilise :
`int longueur = montableau.length;`
Pour (re)trouver la première longueur d'un tableau bidimensionnel :
`int longueur1 = tabmult.length;`
Et pour l'autre longueur (3 dans l'exemple précédent) :
`int longueur2 = tabmult[0].length;`

5.1.3 Accéder aux éléments

L'élément d'indice i du tableau monTableau est obtenu ainsi : monTableau[i]. On peut s'en servir pour lire une valeur ou pour affecter une valeur au tableau :

```
1 int monTableau = new int[3];
2 monTableau[0] = 0; monTableau[1] = 5; monTableau[2] = 11;
3 println("la valeur d'indice 1 est : " + monTableau[1]);
```

5.1.4 Lecture d'un fichier

Pour remplir un tableau avec des éléments, il peut être utile de lire ces éléments dans un fichier texte obtenu par un autre moyen que la saisie manuelle (capture de données, internet, ...)

Exemple 5.1.
Le code suivant résume ce que vous devez connaître pour réaliser les exercices de ce TP (à charger sur l'ENT avec un exemple de fichier map) :

```
1  String[] monTexte; // tableau qui contiendra les lignes du fichier texte
2  int niveau = 0;
3  int nbLignesMap = 0;
4  void setup() {
5    size(500,300);
6    fill(0);
7    textSize(20);
8    noLoop();
9    rectMode(CORNER);
10   selectInput("choisir le fichier texte ", "choixFait");
11 }
12
13 void draw() {
14   background(200);
15   text("Niveau " + niveau, 10,20);
16   for(int l=1; l<=nbLignesMap; ++l) {
17     for(int c = 0; c<monTexte[l].length(); ++c) {
18       if (monTexte[l].charAt(c)=='X') {
19         rect(20*c, 20*l+50,20,20);
20       }
21     }
22   }
23   noLoop();
24 }
25
26 void choixFait(File fichierChoisi) {
27   if (fichierChoisi == null) {//pas de fichier
28     println("Erreur : pas de fichier ou illisible");
29   } else {
30     monTexte = loadStrings(fichierChoisi.getAbsoluteFile());
31     niveau = parseInt(monTexte[0]);
32     println(niveau);
33     nbLignesMap = monTexte.length - 1;// à cause de la 1ère ligne
34     loop();// on lance la méthode draw
35   }
36 }
```

5.2 Exercices

Exercice 5.1.

1. Écrire un algorithme qui réalise les opérations suivantes :
   — demander le nom et le prénom de cinq personnes ;
   — puis, pour chaque personne, pose une question du style « Bonjour <Prénom> <NOM>, quelle est ta couleur préférée [1] » ;

1. Le rédacteur de cet exercice manque cruellement d'inspiration ce matin...

— enfin, écrit à l'écran sous la forme <NOM - prénom - couleur préférée> la liste ainsi obtenue (passage de ligne entre chaque individu mais pas entre le NOM, le prénom et la couleur).

2. Modifier l'algorithme précédent pour qu'il demande au début le nombre de personnes.
3. Modifier l'algorithme de la question 1 pour que la saisie s'arrête dès que l'utilisateur saisit FIN à la place du nom.
4. Même question mais l'utilisateur peut saisir FIN ou fin ou Fin ou FiN ou . . . pour terminer la saisie.
5. Écrire le programme en Processing.

Exercice 5.2.

La suite de FIBONACCI est définie ainsi : $\begin{cases} u_0 = 1; u_1 = 1 \\ u_{n+2} = u_n + u_{n+1}, n \in \mathbf{N} \end{cases}$

1. Écrire un algorithme qui demande le nombre de termes à afficher, calcule ces termes, puis les affiche séparés par des « ; ».
2. Programmer cet algorithme en Processing.
3. Modifier le programme précédent pour qu'il affiche ensuite la liste des quotients de deux termes successifs $(\frac{u_{n+1}}{u_n})$.

Exercice 5.3.

Écrire un algorithme qui calcule dans un tableau à double entrées la table de multiplication pour les entiers de 0 à 12.
Programmer cet algorithme et afficher cette table (attention à respecter l'alignement des colonnes !).

Exercice 5.4.

Écrire un programme qui écrit à l'écran le triangle de PASCAL de degré n fixé au départ (où chaque ligne commence et termine par un « 1 », et chaque élément est la somme de l'élément supérieur et de l'élément supérieur gauche) :

n \ k	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

Exercice 5.5.

Écrire un programme qui stocke dans un tableau les n premiers nombres premiers et les affiche.
Aides mathématiques :

— un nombre supérieur à 1 est premier s'il n'est divisible que par 1 et par lui-même ;
— on montre sans difficultés qu'un nombre n est premier s'il n'admet aucun diviseur premier inférieur ou égal à $\sqrt{n}$.

Exercice 5.6.
Dans cet exercice, vous allez utiliser le programme de l'exemple 5.1.

1. Modifier ce programme pour qu'il ignore toute les lignes commençant par le symbole `#`. Modifier le fichier `map.txt` pour tester.
2. Modifier ce programme pour qu'il affiche un rond bleu lorsqu'il rencontre le caractère `'J'` (joueur) et un carré rouge lorsqu'il rencontre le caractère `'E'` (ennemi). Modifier le fichier `map.txt` pour ajouter ces caractères.
3. Même question mais avec des dessins plus élaborés pour le joueur et les ennemis. Vous pouvez aussi ajouter des pastilles bonus, ...
4. Terminer ce jeu de PacMan[2]...

Exercice 5.7.
Dans cet exercice, nous allons étudier un fichier ADN (à charger à l'adresse habituelle pour tester votre programme).

1. Écrire un programme qui lit un fichier texte contenant des séquence de `A`, `T`, `G` et `C` et qui affiche la séquence ADN sous la forme d'une suite de « traits » en utilisant une couleur par base.
2. Modifier ce programme pour afficher à l'écran la position dans la séquence de la première occurence du « mot » : `AGCT`.
3. Modifier le programme précédent pour écrire dans un fichier (même nom que le fichier lu avec l'ajout des caractères `'-M'` à la fin (mais avant le `.txt`)) la séquence en remplaçant chaque `'T'` par un `'A'` (et réciproquement) et remplacer chaque `'C'` par un `'G'` (et réciproquement).
4. Modifier le programme de la question 2 pour écrire dans un fichier (nommé `genes.txt`) toutes les séquences (une par ligne) situées entre les mots `AGCT` et `TCGA` et en ignorant les autres morceaux du fichier ADN.

2. Belle idée de projet non ?

TP 6

HTML et CSS

Le langage html est le langage de base de toutes les pages web. HTML est l'acronyme de Hyper Text Markup Language c'est-à-dire langage de marquage hypertexte. Ce langage a déjà connu plusieurs versions (l'actuelle est la version html 5) et dans ce document, nous ne verrons que des bases valables pour toutes les versions. Il a été inventé en 1990 par Tim Berners-Lee lorsqu'il travaillait au CERN. Son objectif était d'afficher des pages d'information ayant les deux propriétés suivantes :

- les pages étaient reliées entre elles par des liens hypertextes (les liens sur lesquel on clique) ;
- les pages devaient être lisibles sur tous les ordinateurs.

6.1 Principes de base

L'objectif d'une page web est qu'elle soit visitée (et lue) par un maximum de personnes, pour cela, elle doit être claire, accessible à tous et (si possible) esthétique. Le premier conseil est de faire dans la sobriété !

6.1.1 Éléments fondamentaux

Concrétement, une page html est un fichier texte dont le suffixe est `.html` ou `.htm` ; pour l'écrire nous allons utiliser un éditeur de texte (Nous utiliserons Notepad++ sur les ordinateurs du lycée).
Une fois le site mis en ligne, il est vivement souhaitable qu'une des pages web se nomme `index.html` ainsi, l'utilisateur qui se connecte à votre site `http://adressedemonsite` tombera directement sur cette page.
Un fichier html doit commencer[1] par `<html>` et se terminer par `</html>`. Un exemple de page web minimale serait :

```
1 <html>
2 Ceci est ma (super) page web
3 </html>
```

1. En fait, pas tout-à-fait, nous verrons dans quelques lignes qu'on va commencer par une autre balise...

Les instructions <html> et </html> sont appelées des *balises* (« entrante » et « fermante » respectivement).
Nous allons détailler quelques autres balises qui ne changeront pas fondamentalement cette page web mais qui ont néanmoins de l'importance pour la suite.
Pour indiquer au navigateur le langage employé il est préférable de commencer le fichier par la balise <!DOCTYPE html>, ensuite, on ouvre la balise <html>. Le fichier sera décomposé en deux : l'entête (balise <head>) et le corps (balise <body>).
Dans l'entête nous fournirons des informations qui ne s'affichent pas directement sur la page :

- la balise <meta charset="utf-8"/> permet d'indiquer l'encodage utilisé dans le fichier (pour que les accents et caractères spéciaux s'affichent correctement) ;
- la balise <title>Mon titre </title> permet de donner un titre qui s'affiche dans la barre supérieure du navigateur.

Finalement, notre page web est devenue :

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="iso-latin1"/>
5      <title>Ma première page web</title>
6    </head>
7    <body>
8      Ceci est ma (super) page web.
9    </body>
10 </html>
```

6.1.2 Premières balises

Comme en Processing, il est indispensable de mettre des commentaires dans le fichier html :

```
1  <!-- Ceci est un commentaire qui ne sera pas interprété par le
       navigateur -->
```

Dans le tableau ci-dessous, quelques balises avec l'effet attendu...

Balise	Effet
texte	met `texte` en caractères gras
texte	met `texte` en caractères italiques
<u>texte</u>	souligne `texte`
^{texte}	met `texte` en exposant
_{texte}	met `texte` en indice
texte	met `texte` dans un nouveau paragraphe
texte	centre `texte` (peut être utilisé avec `left` et `right`)
 	passe à la ligne (pas besoin de « fermante »)
<hr width = 50%/>	trace une ligne horizontale de largeur la moitié de la page (pas besoin de « fermante »)
<code>texte</code>	pour mettre en forme du code informatique

6.1.3 Les listes

Une énumération de plusieurs éléments gagne à être mise en avant grâce à des puces ou des numéros, pour cela il existe les balises <ul> (*unordered* list) et <ol> (*ordered* list) :

```
<p> Conseils importants :
<ul>
  <li> écouter le prof;</li>
  <li> apprendre.</li>
</ul></p>
```

Conseils importants :

- écouter le prof;
- apprendre.

```
<p> Conseils importants :
<ol>
  <li> écouter le prof;</li>
  <li> apprendre.</li>
</ol></p>
```

Conseils importants :

1. écouter le prof;
2. apprendre.

6.1.4 Titres et paragraphes

Nous avons vu que chaque paragraphe doit être encadré par et , le langage html permet aussi de hiérarchiser la page à l'aide de titres :

```
1 <h1>Ceci est gros titre (on dit
      de niveau 1)</h1>
2   <h2>Ceci est un moins gros
      titre (de niveau 2)</h2>
3   <h2>Un deuxième moins gros</
      h2>
4 <h1>Encore un gros</h1>
5   <p>Nous allons détailler ici
      le deuxième gros titre</p>
```

Ceci est gros titre (on dit de niveau 1)

Ceci est un moins gros titre (de niveau 2)

Un deuxième moins gros

Encore un gros

Nous allons détailler ici le deuxième gros titre

La mise en forme des titres et des listes (mais aussi de bien d'autres choses) peut être modifiée grâce à une feuille de style qu'on appelle fichier CSS, nous verrons celà un peu plus loin. À noter que grâce à une feuille de style on peut aussi modifier le comportement de certaines balises comme ou .

6.2 Un peu plus loin

6.2.1 Les liens

Nous avons vu que l'essence même du html est le côté « hypertexte » c'est-à-dire la possibilité de créer des liens (à cliquer) entre plusieurs pages d'un même site ou non. Nous allons détailler

leur utilisation ici.
La balise qui permet de créer un lien est `<a ...>... </a>`, nous allons voir comment compléter les ....
Quelques exemples pour comprendre :

```
<a href="http://reymarlioz.free.fr">Un super site de maths</a> :
   à voir absolument !
<a href="page_processing.html">Mes DM en Processing</a>
<a href="downloads/liste.html">Les téléchargements</a>
```

Le premier lien permet de renvoyer à une adresse « externe » : à noter que seule la partie « Un super site de maths » sera cliquable (pas le « : à voir ... »).
Le deuxième lien ouvre la page `page_processing.html` qui doit être placée dans le même répertoire que la page courante.
Le troisième lien ouvre la page `liste.html` située dans un sous-répertoire `downloads` du répertoire courant.
Pour des liens qui pointe vers votre site il est indispensable de les écrire avec des adresses *relatives* (comme les deuxième et troisième exemples ci-dessus) et non pas *directes* (comme par exemple `adresse.de.monsite/downloads/liste.html`) car sinon, en cas de changement d'hébergeur de site, il vous faudra modifier tous vos liens !

Il peut parfois être utile de créer un lien vers une adresse mail :

```
<a href="mailto:contact@monsite.fr?subject=à propos de ton site"
       title="M'envoyer un courriel"> (pour me contacter)
</a>
```

Un clic sur un tel lien ouvre le logiciel de messagerie par défaut avec un nouveau message dans lequel figure déjà l'adresse du destinataire
Remarque : attention, mettre un tel lien sur un site en ligne est la porte ouverte à la réception de spam (si votre site est beaucoup visité, les robots publicitaires relèveront votre adresse...) ; c'est donc fortement déconseillé (il existe des scripts Javascript pour y remédier).

Si la page de votre projet est longue, il peut être intéressant de réaliser au début une table des matières « cliquable ». Ainsi votre page `projet.html` pourra commencer par :

```
<a href="#description">La description du projet</a>
...
<a href="#code">Le code</a>
<a href="credits">Les participants au projet</a>
...
<h1 id="description"> Description de mon projet</h1>
...
<h1 id="code"> Quelques éléments du programme</h1>
...
<a id="credits">Je remercie ici ...</a>
```

6.2.2 Les couleurs, les polices, ...

Dans les balises qui définissent un titre, un paragraphe, ..., on peut ajouter des attributs divers de couleurs, polices, taille, ... Nous allons en voir quelques uns ici sur l'exemple commenté ci-dessous (il est néanmoins préférable de gérer ceci dans une feuille de style – voir plus loin) :

```
1 <!-- toute la page sera sur fond argenté écriture bleue les liens
      en jaunes et les liens cliqués en marine-->
2 <body bgcolor=silver text=blue link=yellow vlink=navy>
3
4 <font color=red>Ce texte est en rouge</font>
5 <table border bordercolor=blue bgcolor=green> <!-- tableau avec
     des cadres bleu et fond vert -->
6 ...
7 </table>
```

6.2.3 Les images

Pour insérer une image dans une page web, il est recommandé d'utiliser un format compressé (jpeg, gif, png). Nous reviendrons plus tard dans l'année sur les formats d'images...
En récupérant le logo que vous connaissez bien sur le site du lycée et en l'enregistrant sous le nom `lyceemarlioz.jpg` dans un sous répertoire `images` de mon site, je peux désormais écrire :

```
1 <img src="images/lyceemarlioz.jpg"
2      alt="le logo du lycée Marlioz"
3      title="texte de survol"
4      hspace=20 vspace=20
5      align=left/>
```

Le contenu de la balise `alt` sera affiché au cas où le navigateur ne peut pas charger l'image et le contenu de la balise `title` s'affiche lorsque la souris survole l'image. Un espace de 20 pixels est réservé entre l'image et le texte (horizontalement et verticalement) et l'image est alignée à gauche.
Remarque : la balise `<img ... />` ne nécessite pas de balise « fermante ».
Il est possible de redimensionner une image en utilisant les attributs `width` (largeur) et `height` (hauteur). Il est prudent de ne redimensionner qu'une seule de ces deux dimensions (l'autre l'est alors aussi proportionnellement).
Enfin, pour mettre une image en fond de page, on modifie la balise `<body>` ainsi :

```
1 <body background="images/monfond.jpg">
```

6.2.4 Tableaux

Un exemple commenté de tableau à modifier pour comprendre comment ça marche.

```
1  <table border="1"><!-- définition
       du tableau -->
2    <tr><!-- nouvelle ligne -->
3      <th><!-- colonne entête -->
4        Matière</th>
5      <th>Coefficient</th>
6    </tr> <!-- fin de ligne -->
7    <tr>
8      <td>Maths</td><!-- colonne
         donnée -->
9      <td> 7 </td>
10   </tr>
11   <tr>
12     <td> ISN</td>
13     <td> 2 </td>
14   </tr>
```

Matière	Coefficient
Maths	7
ISN	2

Résultat : pas terrible ! À vous d'améliorer tout ça...

Remarque : les balises `<td colspan="2">` et `<td rowspan="3">` permettent de fusionner des cellules en colonnes et lignes respectivement.

6.3 Mise en ligne

Une fois votre site créé (c'est-à-dire les fichiers html écrits), il va falloir les mettre en ligne pour que tous les internautes du monde entier puissent en profiter !

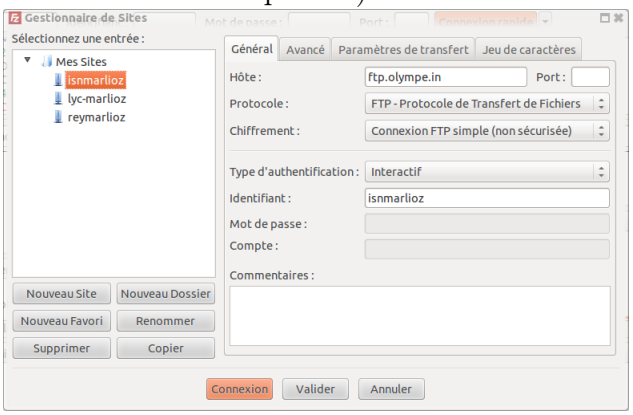
6.3.1 FileZilla

La première chose est de télécharger et installer FileZilla. Une fois lancé, cliquer sur fichier et Gestionnaire de site. Cliquer ensuite sur Nouveau site et compléter ensuite la fenêtre comme ci-contre :

Hôte : `ftp.hebergeur.fr` Port : `21` (à modifier en fonction de votre hébergeur !)

Type d'authentification `Interactif` et compléter l'identifiant par votre celui que vous a fourni votre hébérgeur ; idem pour le mot de passe. Votre mot de passe vous sera demandé une deuxième fois.

Exemple (désuet : cet hébergeur n'existe plus...)



Dans la colonne `Site local`, sélectionner le répertoire où est stocké votre site sur votre ordinateur, puis, faire glisser à la souris les fichiers à envoyer (au minimum `index.html` et les images nécessaires) dans la colonne `Site distant`.

6.3.2 Référencement

Pour que votre site apparaisse en bonne place dans une recherche de moteur de recherche lorsqu'un internaute fait une recherche sur « isn marlioz processing » par exemple, il faut d'abord générer du trafic sur votre page en plaçant des liens partout où vous le pouvez (après la signature de vos mails, sur votre page facebook, j'en mettrai un sur la page d'accueil de l'ENT, ...) et pour que les moteurs de recherche vous « indexent » c'est-à-dire qu'ils sachent de quoi parle votre site, nous allons placer des mots-clés.
Dans l'entête (entre les balises `<head>` et `</head>`) nous allons placer des « meta-informations » :

```
<head>
  <meta charset="iso-latin1"/>
  <title>Ma première page web</title>
  <meta name="description" content="contenu des cours d'ISN au
     lycée Marlioz">
  <meta name="keywords" content="rey isn marlioz processing html
     cours">
  <meta name="nameAutor" content="Thomas REY">
  <meta http-equiv="date" content="19.10.2018">
</head>
```

6.4 Compléments

6.4.1 Feuilles de style

Puisqu'il me reste du temps et de la place, quelques rudiments de CSS...
L'ensemble des règles qui régissent l'affichage de vos pages web est appelé *feuille de style*. Il s'agit d'un ou de plusieurs fichiers `.css`. Une feuille de style est constituée de plusieurs *règles de style* toutes construites sur le modèle suivant :

- un sélecteur (c'est-à-dire la balise concernée par cette règle) ;
- une déclaration de style sous la forme d'un bloc d'ensemble `propriété: valeur;`.

```
/* Commentaire : définition du style des titres de niveau 3 */
h3 { font-style: italic;
     font-family: Arial, sans-serif;
   }
```

L'ensemble des règles de style est enregistré dans un fichier `style.css` (ou un autre nom d'ailleurs) et est appelé dans l'entête de chaque page html ainsi :

```
<head>
  <link rel="stylesheet" href="style.css" />
</head>
```

Si vous voulez par exemple écrire en rouge certains paragraphes mais que vous n'êtes pas sûr de la couleur (ou que vous souhaitez pouvoir modifier la couleur de *tous* ces paragraphes ultérieurement), il faut procéder en deux étapes :

— dans vos fichiers html, on définit une *classe* de paragraphe ainsi : `<p class="format1" >.... </p>` (si possible avec un nom plus parlant que « `format1` ») ;
— dans votre feuille de style css, on crée le style `format1` ainsi :

```
1 p.format1 {color: red;}
```

Si vous préférez par la suite que ces paragraphes soient centrés plutôt qu'en rouge vous avez juste à modifier votre feuille de style ainsi :

```
1 p.format1 {text-align: center;}
```

Un exemple :

```
1 /* Définition des titres */
2 h1 { color: red; font-size: 150%; background-color: silver; }
3 h2 { color: blue; font-size: 125%; background-color: silver; }
4 /* les balises ciblées avec un class="titre" seront encadrées */
5 .titre { border: solid; }
6 /* re-définition du strong en bleu sur fond rouge */
7 strong { color: blue; background-color: rouge; }
```

Quelques propriétés CSS sous forme d'exemples :

```
1  /* changement de police pour tous les paragraphes */
2  p {font-family: Arial, Verdana, sans-serif;} /* plusieurs choix
       au cas où... */
3  p {font-family: "Courier New", monospace;} /* pas fixe : pour le
       code */
4  /* taille */
5  p {font-size: 150%;}
6  p {font-size: 15px;}
7  /* couleurs */
8  p {color: #0000ff;} /* codage hexa */
9  p {font-weight: bold;}
10 p {font-weight: normal;}
11 p {font-style: italic;}
12 /* texte souligné + barre au dessus + barré + clignotant (ouh c'
       est moche !)*/
13 p {text-decoration: underline overline line-through blink}
14 /* surlignage :*/
15 p {background-color: red;}
16 p {text-align: left;} /* ou right, center, justify
17 /* curseur de souris : */
18 .aide {cursor: help} /* ou default, move, wait, pointer, ... */
```

Il existe encore beaucoup d'autres propriétés, à vous de trouver de la documentation en ligne ou dans la référence bibliographique cité en fin de ce document...

6.4.2 Propriété intectuelle ...

Attention : lorsque vous récupérez des logos, images, textes, animations, ... il faut toujours vérifier que ces images sont libres de droit et consulter la licence sous laquelle ils sont publiés. Par exemple tous les documents du site http://reymarlioz.free.fr sont sous licence Creative Commons NC-BY-SA (utilisations libre sus les trois réserves suivantes : pas d'utilisation commerciale, auteur à citer en cas de réutilisation, et diffusion des modifications sous la même lience).

6.4.3 Bibliographie

Pour ce cours, je me suis largement inspiré du livre suivant : « Premiers pas en CSS 3 & HTML5 » – Francis DAILLARD – Eyrolles.

6.5 Au travail !

Votre objectif pour cette séance et jusqu'à la fin de l'année 2018 est de créer un site web (par bi/trinôme) qui respectera les impératifs suivants :

- les pages doivent être écrites par vous (sans l'aide de logiciels de conception de site web type Dreamweaver, GoogleSite, ...) ;
- elles doivent m'être rendues sous forme d'une archive zip (avant les vacances de Noël) que je mettrai en ligne sur un site à déterminer ou bien vous pouvez le mettre en ligne vous même si vous avez un hébergeur ;
- elles doivent comporter des images, des liens, ...
- elles doivent être les plus lisibles et homogènes possibles (ne pas changer de police, de taille de caractères, de fonds, ... à chaque page !) ;
- elles doivent utiliser une feuille de style `CSS` écrite par vous également ;
- pour le contenu, une page d'accueil, une page de présentation de vous, une page pour les DM en Processing, et enfin, une page sur un sujet en lien avec le numérique à choisir dans la liste ci-après.

Liste des thèmes (chaque thème doit être choisi par un groupe maximum !) :
Représentation des données en binaire, Propriété de l'information, Cryptographie, Informatique et environnement, Numérisations, Les métiers de l'informatique, Intelligence artificielle, Formats numériques, Cloud Computing, biographie d'une figure de l'informatique (Ada LOVELACE, Alan TURING, ...)toute autre proposition à valider par moi.

Pour vous aider, je vous mettrai sur l'ENT un exemple de zip attendu. Ce travail sera noté pour le second trimestre.

TP 7

Images bitmap et GIF animés

Pour mener à bien votre projet vous aurez sûrement besoin d'y intégrer des images. Évidemment, il n'est pas question d'aller le piller le travail d'un autre en cliquant sur le bouton « Enregistrer l'image sous… » de votre navigateur. Nous allons donc, aujourd'hui, apprendre à créer nos propres images, et même les animer sous forme de fichier `gif` animés.

7.1 En noir et blanc : le format pbm

La façon la plus naturelle de représenter une image est de la quadriller et de colorier chaque case (ou pixel) en noir ou en blanc. Nous allons, dans un premier temps, utiliser le tableur pour créer de tels fichiers images.
Prenons comme convention qu'un carré noir est représenté par un « 1 » et un carré blanc par un « 0 ». Quel type de données vous semble-t-il judicieux de choisir pour caractériser un « carré en noir ou blanc » ?
En déduire la signification de l'expression « image bitmap ».

1. Notre première image « bitmap » :
   a. ouvrir un classeur de tableur et remplir cent cellules (un carré de 10 par 10 de la cellule `A4` à la cellule `J13`) avec des 0 et des 1 ;
   b. sélectionner ce carré de cellules et créer le formatage conditionnel suivant : cellule sur fond noir lorsque la valeur de la cellule est égale à 1 ;
   c. réduire la largeur des colonnes et la hauteur des lignes (0,5 cm pour chaque par exemple) : voilà votre première image bitmap !
2. Le format « pbm » (PortableBitMap) permet de caractériser ce type d'images. Un fichier `image.pbm` est un fichier *texte* qui contient [1] :
   — sur la première ligne les caractères `P1`
   — sur la deuxième ligne deux nombres en base 10 séparés par un espace qui donnent respectivement la largeur et la hauteur (en pixels) de l'image ;
   — des éventuelles lignes de commentaires précédées du symbole `#` ;
   — une liste de 0 et de 1 sur les lignes suivantes.

1. Nous décrivons ici le type pbm *brut* ; c'est-à-dire en ASCII. Il existe aussi un format pbm *binaire* qui commence, lui, par les caractères P4 et dans lequel on écrit les successions de 0 et de 1 regroupés en octets.

Reprenons notre feuille de tableur réalisée précédemment :

a. ajouter dans la cellule `A1` le texte `P1`, dans les cellules `A2` et `B2` les valeurs 10 et 10 (dimensions de l'image), et enfin dans la cellule `A3`, écrire `# Mon image` ;

b. enregistrer ce classeur « sous » et choisir le format `csv` (comma-separated value) en prenant comme séparateur de champ : un espace, comme séparateur de texte : rien (i.e : supprimer le point virgule ou la virgule ou les guillemets proposés) et en lui donnant comme nom `image.pbm` ;

c. ouvrir maintenant ce fichier avec gestionnaire d'image (The Gimp par exemple) : voilà votre première image bitmap au format pbm !

3. De la même façon que précédemment, créer une image bitmap noir et blanc au format pbm de taille 48×48 où chaque pixel est noir ou blanc de façon aléatoire. On pourra utiliser la fonction `alea()` du tableur. Vous obtiendrez une image qui ressemble à celle ci-contre.

4. a. Combien de bits sont-ils nécessaires pour « décrire » cette image ? Combien d'octets cela représente-t-il ?

   b. Comparer avec la taille de votre fichier.

   c. Expliquer l'(énorme) différence.

7.2 En niveaux de gris

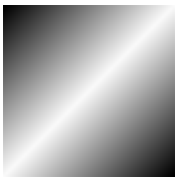
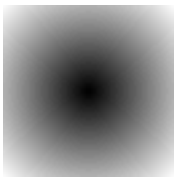
De la même façon, pour décrire une image en *niveaux de gris*, on va attribuer à chaque pixel une valeur comprise entre 0 et 65 536 (0 pour noir, 65 536 pour blanc et toutes les autres valeurs pour les nuances entre les deux). Ce type de format d'images est le format « pgm » (PortableGrayMap) ; nous utiliserons le format pgm brut (ou ASCII) qui est constitué :

— d'une première ligne sur laquelle figurent les caractères `P2` ;
— d'une deuxième ligne où figurent, séparés par un espace, la largeur puis la hauteur de l'image ;
— d'une troisième ligne avec la valeur maximale N utilisée pour coder les niveaux de gris (par exemple $N = 65\ 536$) ;
— enfin, d'une série de nombres compris entre 0 et $N - 1$ séparés par des espaces.

Attention : aucune ligne ne doit dépasser 70 caractères.
En utilisant le tableur et la méthode vue dans la partie 7.1 (il est fortement conseillé de créer des formules dans le tableur puis d'exporter en pbm) :

1. Créer une image de taille 64×64 en 64 niveaux de gris qui forme un dégradé horizontal de niveaux de gris (voir figures ci-dessous).
2. Même question pour un dégradé vertical.
3. Même question pour un double-dégradé diagonal.
4. Même question pour un dégradé « central ».



7.3 Et en couleurs

Nous reviendrons ultérieurement sur les façons de coder des couleurs en informatique. Nous allons ici utiliser le codage RGB (Red-Green-Blue) : chaque nuance de couleur est obtenue par la somme d'une « quantité » de chacune de ces couleurs [2]. Chaque pixel de notre image est donc codé par trois nombres compris entre 0 et 255 (par exemple).
Finalement une image au format « ppm » ASCII est un fichier texte composé ainsi :

```
P3
64 64
# ci-dessus largeur et hauteur en pixels
255 # nuance "max" dans chaque couleur RGB
255 0 0   0 255 0   124 36 58 ...
```

1. Combien de nuances de couleurs différentes peut-on obtenir avec un tel codage ?
2. Créer des images en couleurs (expression libre).

P.S : on retrouve ces trois codages ici : http://fr.wikipedia.org/wiki/Portable_pixmap

7.4 Création d'un gif animé

L'objectif de cette partie est de créer un fichier gif animé qui représentera un cheval qui court. Dans un premier temps, nous allons créer un fichier en noir et blanc, ceux qui voudront l'améliorer avec des couleurs le pourront...
En utilisant la méthode décrite dans la partie 7.1, créer une première image de cheval de taille 20 pixels de large sur 12 de haut. L'enregistrer au format pbm sous le nom `cheval1.pbm`.
Créer plusieurs autres images à partir de la première qui correspondent à d'autres positions du cheval pendant sa course. Les nommer `cheval2.pbm`, `cheval3.pbm`, ...
Vous pouvez aussi utiliser ceci : http://reymarlioz.free.fr/p5js/creationBitmap/ pour créer des images bitmap au format `png` en 32x32 pixels (si l'un d'entre vous veut modifier le code pour créer des images dans d'autres dimensions, qu'il n'hésite pas...)
Nous allons maintenant utiliser le logiciel The Gimp pour créer notre animation :

- ouvrir The Gimp ;
- dans le menu `Fichier`, cliquer sur `Nouvelle Image`, puis choisir sa taille en pixels : 20 de large et 12 de haut (ou 32x32, ou ...) ;
- dans le menu `Fichier`, choisir `Ouvrir en tant que calques` et sélectionner les différentes images `cheval1.pbm`, `cheval2.pbm`, ... en maintenant la touche `Maj` enfoncée ;
- dans le menu `fichier`, choisir `Exporter` puis dans `sélectionner le type de fichier`, choisir `image GIF` et penser à saisir le bon chemin ainsi qu'un nom de fichier par exemple `monBeauCheval.gif` ;
- cocher les options `As animation`, `Loop forever`, et choisir le délai entre deux images en fonction du nombre d'images.

Et voilà, vous avez un gif animé...

Pour ceux qui seraient (encore plus) nuls (que moi) en dessin, voici mes essais :

2. Voir vos cours d'Arts Plastiques de collège...

7.5 Et avec Processing ?

7.5.1 Insérer une image

Pour insérer une image dans un projet Processing, c'est assez simple...

1. Faire glisser l'image (format `png`, `jpeg`, `gif` – mais il ne retiendra que la première image s'il contient une animation) depuis l'explorateur de fichier sur la fenêtre du sketch : un dossier `data` est alors créé dans le répertoire de votre sketch et votre fichier image est copié dans ce dossier.
2. Créer une variable de type `PImage` que nous nommerons par exemple :
   `monImage` par l'instruction `PImage monImage;`
3. Charger l'image dans cette variable : `monImage = loadImage("fichier.png");`.
4. Afficher l'image aux coordonnées `(100;50)` par l'instruction `image(monImage,100,50);`

Finalement, notre sketch minimal contient :

```
1 PImage monImage;
2 void setup() {
3   size(800,600);
4   monImage = loadImage("test2.png");
5   image(monImage,50,50);
6 }
```

Attention : l'instruction `loadImage` ne doit surtout pas être dans la méthode `draw` ou dans une boucle sinon elle serait chargée en mémoire depuis le disque plusieurs fois. Ceci ralentirait considérablement l'exécution de votre programme !

7.5.2 Insérer un gif animé

Processing ne gère pas nativement les images animées. Il existe néanmoins une librairie qui fait ça (pour Processing 3.x). Vous la trouverez ainsi que la documentation à l'adresse suivante : https://github.com/akiljohnson1/GifAnimation.
Nous y reviendrons en détails dans le TP 11 sur le multimédia en Processing...
Cette librairie permet aussi de créer un `gif` animé à partir d'un sketch Processing.

TP 8

Interactions avec l'utilisateur

Jusqu'à présent, le seul moyen de communication avec les utilisateurs de nos programmes étaient les boîtes de dialogue permettant de cliquer sur un bouton ou de répondre à une question. L'objectif de ce TP et d'apprendre à faire interagir le programme et l'utilisateur par l'intermédiaire du clavier, de la souris, du micro, de la webcam, ...

8.1 Gestion du clavier

Lors de l'exécution d'un programme, Processing est en permanence « à l'écoute » des événements pouvant survenir (usage de la souris, du clavier, du micro, ...). Nous allons illustrer cette fonctionnalité pour qu'un programme réagisse à un appui sur une touche du clavier.
Dès qu'une touche du clavier est appuyée, Processing va exécuter la méthode `keyPressed()` si celle-ci existe et dès que la touche est relâchée, c'est la méthode `keyReleased()` qui est lancée.

8.1.1 Première approche

Exercice 8.1.

1. Que fait le programme suivant ?

```
1  int fond = 0;
2  String t = "nuit";
3  void setup() {
4    size(400,300);
5    textSize(40);
6  }
7  void draw() {
8    background(fond);
9    fill(255 - fond);
10   text(t,200,150);
11 }
12 void keyPressed() {
13   fond = 255;
14   t = "jour";
```

```
15 }
16 void keyReleased() {
17   fond = 0;
18   t = "nuit";
19 }
```

2. Modifier ce programme pour que l'alternance du fond soit bleu-nuit et jaune.

Dans l'exercice précédent, on ne différenciait pas les touches sur lesquelles on appuyait. Pour connaître la touche appuyée, il existe deux variables :
- `key` : qui renvoie le caractère sur lequel on a appuyé (attention certaines touches ne sont pas des caractères comme les flèches de déplacement, contrôle, alt, ...) ;
- `keyCode` : qui renvoie le code de la touche.

Exercice 8.2.
Que fait le programme suivant ?

```
1 void setup() {
2   size(400,300);
3   textSize(100);
4   fill(0);
5 }
6 void draw() {
7   background(255);
8   text(key + " : " + keyCode,0,150);
9 }
```

Un récapitulatif des codes de certaines touches est disponible dans l'annexe A.

8.1.2 Et pour plusieurs touches...

Cette partie est indispensable pour réussir à « écouter » deux joueurs en même temps. En effet les fonctions `key` et `keyCode` vues dans la partie précédente ne retiennent que la dernière touche appuyée, ainsi, dans un jeu, un joueur qui voudrait se déplacer en diagonale devrait appuyer alternativement sur deux touches. Pas très jouable... Même problème pour un jeu à deux : le dernier qui appuie sur une touche « bloque » l'autre...
Pour remédier à ce désagrément, nous allons créer un *tableau* dans lequel nous allons stocker l'état de chaque touche du clavier (enfoncée ou relâchée). C'est ce tableau que nous consulterons pour déplacer ou non les éléments (personnages, raquettes, ...) de chaque joueur.

Dans un jeu, ce qui nous intéresse est de savoir si une touche est enfoncée ou relevée. Nous allons créer un tableau `etatTouche` de booléens (vrai ou faux) qui contiendra vrai si la touche est enfoncée et faux sinon :

```
1  // Le plus grand code d'une touche est 255
2  boolean[] etatTouche = new boolean[256];
3  // La méthode qui met une touche à "vrai" :
4  void keyPressed() {
5    etatTouche[keyCode] = true;
6  }
7  // La méthode qui met une touche à "faux" :
8  void keyReleased() {
9    etatTouche[keyCode] = false;
10 }
```

Ensuite, pour savoir s'il faut exécuter une action lorsqu'une touche est enfoncée, il suffit de consulter le tableau `etatTouche`.

Exercice 8.3.

1. Créer un programme qui affiche en permanence les codes de toutes les touches enfoncées du clavier.

   Rappels et aides :
   - votre programme doit comporter les lignes de code ci-dessus ;
   - la méthode `setup` doit créer une fenêtre graphique, initialiser les paramètres généraux (mode de dessins des rectangles, ellipses, taille des caractères, ...)
   - la méthode `draw` doit successivement effacer l'écran, effectuer une boucle sur les codes de touches possibles, si l'élément correspondant du tableau `etatTouche` est sur `true`, afficher la touche et son code dans la fenêtre les uns en dessous des autres.

2. Modifier le programme précédent pour qu'il affiche toutes les combinaisons « code : touche » en blanc à l'écran et que chaque touche enfoncée s'affiche en vert.

   Remarque : finalement, cette version est plus « facile » car on affiche *toutes* les touches possibles et donc la gestion des positions des affichages sur la fenêtre doit être « automatique » et prévu dans la boucle.

Exercice 8.4.

Écrire un programme qui permet de saisir le nom d'un joueur dans une fenêtre avec les contraintes suivantes :

- fenêtre de dimensions 400×200 sur fond noir ;
- une question (texte = `Saisir ton login (8 caractères max.) :`) centrée en haut de la fenêtre écrite dans une couleur claire (auchoix) ;
- un rectangle de bordure rouge foncé et d'intérieur bleu clair centré également et de dimensions 200×40 ;
- à chaque appui sur une touche, affichage du caractère dans le rectangle (à la suite des précédents) en limitant à 8 caractères (avec gestion des flèches, suppr, backspace, ...)
- lors de l'appui de la touche `entrée`, effaçage de l'écran pour écrire « Bienvenue + login saisi ».

8.2 Gestion du mulot

Le principe est le même que pour le clavier. Cependant la gestion de la souris comporte plus de possibilités.
D'abord, il existe des variables `mouseX` et `mouseY` qui contiennent en permanence les coordonnées du pointeur de la souris. Ensuite la variable `mousePressed` contient un booléen qui est `true` si un bouton quelconque de la souris est enfoncé. La variable `mouseButton` contient le dernier bouton appuyé. Il pourra être utile de créer un tableau des boutons appuyés comme pour les touches de clavier (voir partie précédente).
Il existe aussi les variables `pmouseX` et `pmouseY` qui sont les coordonnées du pointeur de souris lors de l'affichage de la frame précédente (de l'étape précédente de la méthode `draw()`).
Enfin, il existe des méthodes qui sont exécutées lorsque des événements surviennent (voir tableau ci-dessous). Ces méthodes sont à définir de la même façon que les méthodes `keyPressed()` et `keyReleased()` de la partie précédente.

Méthode	action qui la déclenche
`mousePressed()`	après qu'un bouton soit pressé
`mouseReleased()`	après qu'un bouton soit relâché
`mouseMoved()`	chaque fois que la souris bouge
`mouseDragged()`	chaque fois que la souris bouge avec un bouton enfoncé
`mouseClicked()`	après qu'un bouton soit appuyé *et* relâché
`mouseWheel()`	lorsque la molette de la souris est tournée

Concernant la méthode `mouseWheel()`, celle-ci prend un paramètre qui est de type `mouseEvent` ; voici un exemple de fonctionnement :

```
1 void mouseWheel(mouseEvent evenement) {//evenement est une
     variable contenant des infos sur l'événement ayant déclenché
     cette méthode
2   float e = evenement.getCount();
3   println(e); // si molette vers bas : e > 0
4               // si molette vers haut : e < 0
5 }
```

Exercice 8.5.
Créer une page d'accueil d'une application comportant deux boutons (sous la forme d'un rectangle rempli par exemple) et qui réagit différemment selon si on clique sur ces boutons (je vous laisse libre de la « réaction »).
On peut ensuite modifier le code pour que la réaction soit différente en fonction du bouton cliqué (droit ou gauche).

8.3 Son et vidéo

De même que votre programme peut réagir au clavier et à la souris, il est possible de se servir des entrées audio et vidéo de l'ordinateur. Nous ne rentrerons pas dans les détails ici, deux paragraphes sont consacrés à ceci dans le manuel de Processing Floss Manual disponible

ici : http://fr.flossmanuals.net/processing/lentree-microphone/ et là : http://fr.flossmanuals.net/processing/lentree-video/.
Je vous laisse vous y référer si besoin.
Juste une fonctionnalité intéressante (par exemple pour créer un gif), la capture d'écran :

```
1 // Sauvegarder une image png de chaque frame affichée au format 'image-000001.png', image-000002.png', ... (peut être gourmand en ressources !)
2 saveFrame("image-######.png");// à mettre dans draw
3 // Sauvegarder une image à un moment donné :
4 save("image.png");// Attention, écrase le fichier sans prévenir
```

8.4 Exercices

Exercice 8.6.
Déplacer une forme ou un personnage à l'écran à l'aide des flèches du clavier. Puis ajouter des fonctionnalités comme tourner la tête (si c'est un personnage), sauter, attraper un objet, lancer un objet, courir, danser, ...
Bref, réaliser les bases d'une animation de jeu vidéo.

Exercice 8.7.
En reprenant le code de l'exercice 5.6 (lecture d'un fichier texte contenant une map de jeu), compléter le programme pour pouvoir déplacer le personnage dans le labyrinthe à l'aide des flèches du clavier.

Exercice 8.8.
Créer un programme qui permet de dessiner à l'écran à l'aide de la souris :
- un point de diamètre fixé s'affiche à la position du curseur de la souris tant que le bouton gauche est enfoncé ;
- le bouton droit « efface ».

Améliorations possibles :
- changement de la taille du point (par appui sur une touche du clavier) ;
- tracé d'une ligne plutôt qu'un point ;
- possibilité de changer de couleur ;
- possibilité de changer de forme ;
- ajout de texte ;
- ajout d'une barre à droite dans laquelle on ne dessine pas mais où on sélectionne les commandes, couleurs, ...
- refaire le logiciel « paint » en fait...

Remarque : c'est le logiciel idéal à convertir ensuite en application android pour tablette. Nous verrons si nous avons le temps de le faire ensuite...

TP 9

Un jeu de rôles

Le but du TP de cette semaine est de vous mettre dans la peau d'un membre d'une équipe d'informaticiens menant un projet pour un client. Par équipes de quatre ou cinq ~~élèves~~ professionnels choisis au hasard (en entreprise, vous ne choisirez pas vos collègues) vous allez devoir réaliser un projet pour un client (dont le rôle sera joué par votre cher professeur d'ISN...).

9.1 Cahier des charges

Dans un projet, le cahier des charges permet de connaître le travail à effectuer. En DS, c'est « l'énoncé ».
Il s'agit de réaliser un jeu assez basique dans lequel on présente un canon orientable par le joueur qui peut tirer un boulet de canon avec une vitesse initiale réglable aussi par le joueur. Ce boulet de canon doit atteindre une cible qui a été placée aléatoirement dans l'aire de jeu.
Voici quelques contraintes :

- l'angle formé par le canon et le sol augmente lorsqu'on appuie sur la flèche du haut du clavier et diminue lorsqu'on appuie sur la flèche du bas ;
- la longueur du canon augmente lorsqu'on appuie sur la flèche droite et diminue lorsqu'on appuie sur la flèche gauche (ceci matérialisera la vitesse initiale du boulet) ;
- les réglages des deux points précédents sont « encadrés » pour ne pas que le canon tire vers l'arrière ou avec une vitesse négative !
- prévoir un décor de fond ;
- le déplacement du boulet doit apparaître à l'écran (à vitesse « visible ») en respectant les loi de gravitation universelle (en négligeant les frottements) ;
- après chaque tir, une fois le boulet arrivé au sol, il faut afficher :
  - si le tir a touché la cible ou non ;
  - le score total depuis le début de la partie (x tirs réussis sur y tentatives) ;
  - ré-initialiser une partie avec une nouvelle cible si la précédente a été touchée et la même sinon ;
- le code doit être documenté (c'est-à-dire commenté) ;
- une page de présentation du jeu et de l'équipe serait un plus...

9.2 Répartition des rôles

Pour ce projet, l'équipe sera constituée :

- d'un chef de projet qui coordonne et vérifie que chacun réalise son travail dans le temps imparti ; il propose également des solutions pouvant aider un des membres de son équipe qui serait « bloqué » ;
- un graphiste qui va se charger des décors, de réaliser le code qui affiche le canon, la cible et le boulet ;
- d'un ou deux analyste(s) développeur(s) qui va/vont écrire le « moteur du jeu » avec notamment la gestion du clavier, le déplacement du boulet, le test de cible atteint ou non ...
- un adjoint au chef de projet qui va assembler les différentes parties de code et qui sera en lien étroit avec le chef de projet.

9.3 Aides et contraintes

Dans cette partie, quelques aides techniques et théoriques ainsi que des contraintes supplémentaires à respecter afin de faciliter :

- l'explication au reste de la classe (par le chef et son adjoint) ;
- l'assemblage de vos différents jeux en une seule application.

En premier lieu, chaque nom de variable utilisée, chaque nom de méthode doit respecter les consignes suivantes :

- les noms de variables/méthodes ne contiennent que des lettres (et commencent par une minuscule !) et donc ne contiennent pas de chiffre ;
- **sauf :** qu'ils se terminent *tous* par le chiffre correspondant au numéro de votre équipe.

Par exemple si l'équipe 3 crée une variable qui détermine la vitesse du boulet elle s'appelera `vitesseBoulet3`

9.3.1 Le type `PVector`

En Processing, il existe un type de variable qu'on n'a pas rapidement vu et qui s'appelle `PVector` (En fait, il existe une *classe* qui définit le type `PVector`). Tester le code suivant pour comprendre son usage :

```
1  // On crée 2 instances de l'objet PVector par leurs coordonnées
2  PVector vitesse = new PVector(3, -10);
3  PVector balle = new PVector(300, 200);
4
5  void setup() {
6    size(600, 400);
7    frameRate(4);
8  }
9
10 void draw(){
11   background(0);
```

```
12    dessineBalle();
13    balle = balle.add(vitesse);// Le vect balle est augmenté du
          vecteur vitesse
14    // cela revient à balle.x = balle.x + vitesse.x (et idem avec .
          y)
15    vitesse.y = vitesse.y + 1;// On augmente la coordonnée y de la
          vitesse
16  }
17
18  void dessineBalle() {
19    ellipse(balle.x,balle.y,10,10);
20  }
```

Quelques méthodes utiles à appliquer à un objet `vitesse` de type `PVector` :
`vitesse.rotate(angle)` tourne le vecteur `vitesse` d'un angle `angle` exprimé en radians.
`vitesse.mag()` donne la norme du vecteur `vitesse`
`vitesse.mult(k)` multiplie les coordonnées du vecteur `vitesse` par `k`
`vitesse.setMag()` fixe le carré de la norme du vecteur `vitesse` sans changer sa direction
`vitesse.heading()` donne l'angle formé par le vecteur avec l'horizontal

9.3.2 Programmation par méthodes

Le projet devra être constitué des méthodes suivantes (où vous remplacerez le `_i` figurant en fin de chaque nom par votre numéro d'équipe) :

- une méthode `dessineCanon_i()` qui sera chargée de dessiner le canon avec le bon angle et la bonne taille ;
- une méthode `dessineFond_i()` qui dessine le décor ;
- une méthode `dessineCible_i()` qui dessine la cible à la position définie par l'initialisation du programme ;
- une méthode `dessineBoulet_i()` qui dessine le boulet uniquement si le tir a été déclenché ;
- une méthode `avanceBoulet_i()` qui effectue le calcul de la position suivante du boulet ;
- une méthode `testsClavier_i()` qui modifie les bonnes variables lorsqu'on appuie sur les touches définies dans le cahier des charges ;
- une méthode `lanceBoulet_i()` qui effectue les opérations lorsqu'on appuie sur la barre d'espace ;
- une méthode `testTouche_i()` qui renvoie un booléen déterminant si le boulet touche la cible ;
- une méthode `finPartie_i()` qui affiche touché ou raté et qui incrémente (ou pas) le score puis qui relance une nouvelle partie par exemple quand on appuie sur une touche ;
- une méthode `initPartie_i()` qui initialise une nouvelle partie : canon dans la position d'origine, nouvelle cible, ...

C'est au chef de projet de répartir l'écriture des méthodes au graphiste ou au(x) développeur(s) en respectant la répartition des rôles imposée.

9.3.3 Gravitation universelle

Vous verrez (ou vous avez vu ?) en Physique cette année qu'un corps qui n'est soumis qu'à son poids a une accéleration constante $\vec{a} = \vec{g}$ (où $\vec{g}$ est le vecteur accélération de la pesanteur dirigé vers le bas de norme 9,81 m.s^{-2}). Or l'accélération est la dérivée de la vitesse et de même, la vitesse est la dérivée de la position. On a donc :

$$\frac{\mathrm{d}V}{\mathrm{d}t}(t) = a(t) \quad \text{et} \quad \frac{\mathrm{d}M}{\mathrm{d}t}(t) = V(t)$$

Chaque vecteur a deux (en fait trois, mais pas dans le plan de notre jeu) coordonnées. En notant $(V_{x0}; V_{y0})$ les coordonnées du vecteur vitesse initiale et $(x_0; y_0)$ les coordonnées du point de départ, on obtient (*cf.* vos (excellents) cours de maths et SPC) :

$$\vec{V}(t) \left| \begin{array}{l} V_x(t) = V_{x0} \\ V_y(t) = gt + V_{x0} \end{array} \right. \quad \text{et} \quad \vec{M}(t) \left| \begin{array}{l} x(t) = V_{x0} \times t + x_0 \\ y(t) = \frac{1}{2}gt^2 + V_{y0} \times t + y_0 \end{array} \right.$$

Si on récapitule, dans la méthode `deplaceBoulet_i()`, à chaque étape, il faudra :

1. déplacer le boulet par une instruction du type :
   ```
   boulet = boulet.add(vitesseBoulet);
   ```
2. augmenter l'ordonnée de la vitesse d'une quantité `g` par :
   ```
   vitesseBoulet.y = vitesseBoulet.y + g;
   ```

9.3.4 Clavier

Nous avons vu dans le TP 8 qu'il est possible de récupérer dans un tableau de booléens les touches appuyées. Dans le fichier qui vous sera fourni (voir partie 9.3.5), ce tableau est nommé `touches`.
Ainsi lorsqu'on appuie sur la barre d'espace, `touches[32]` vaut `true` car le code 32 correspond à la barre d'espace.
Les flèches gauche, haut, droite, bas ont pour codes respectifs : 37, 38, 39, 40. La touche entrée a pour code 10 (par exemple pour lancer une nouvelle partie).

9.3.5 Le fichier de départ

Dernière contrainte (et aide aussi !) pour l'adjoint au chef de projet vous êtes obligés de partir du fichier disponible sur l'ENT avec interdiction de modifier le premier onglet (en particulier la taille de la fenêtre) à une exception près (voir dernier paragraphe ci-dessous). Tout votre code est à écrire dans l'onglet `equipe_i` (dont vous remplacerez le `_i` par votre numéro d'équipe).
En plus des méthodes précédemment décrites, l'adjoint au chef de projet écrira deux méthodes supplémentaires :

— une méthode `setup_i()` qui sera executée dès que le jeu se lance ;
— une méthode `draw_i()` qui sera executée automatiquement 12 fois par seconde.

Pour tester votre programme final, il suffira de remplacer le `_i` correspondant à votre équipe dans l'appel des méthodes `setup_i()` et `draw_i()` du premier onglet.

TP 10

Objets

Nous avons déjà vaguement parlé d'objet au cours du TP 2 (dans la partie 2.3). Nous allons ici nous y attarder un petit peu...
Dans les TP précédents, nous avons animé une balle à l'écran qui rebondissait sur les bords de notre fenêtre. Imaginons maintenant que nous voulions créer un jeu de billard. Il nous faudrait animer simultanément plusieurs balles. Pour chacune, il faut avoir des variables qui contiennent les coordonnées, la direction, la couleur, la vitesse, ... Nous pourrions créer pour chaque type de données un tableau mais ce n'est pas très logique, il serait plus cohérent de créer des « super-variables » d'un nouveau type qu'on appelerait `Balle` et qui regrouperait toutes les informations d'une balle. Cette création de nouveau « super-type » de variable est possible en Java [1] (et donc en Processing) c'est ce qu'on appelle un *objet*.

10.1 Déclaration d'un objet

Pour définir un nouveau type (c'est-à-dire un objet), il faut créer une *classe*, soit dans un fichier à part (et c'est conseillé), soit à l'intérieur de votre fichier `pde`.
Pour créer notre classe `Balle` dans un nouveau fichier, il suffit de cliquer sur le petit triangle blanc à droite de l'onglet du nom de votre sketch puis de sélectionner `Nouvel onglet` et enfin de renseigner son nom : `Balle` (à noter : il d'usage courant [2] que le nom d'une classe commence par une majuscule).
Dans cet onglet, on commence par donner un nom à notre nouveau type d'objet (en Java, c'est obligatoirement le même nom que le nom du fichier, donc nous ferons pareil en Processing), ce nom sera donc `Balle` :

```
1 class Balle {
2
3 }
```

Nous allons maintenant définir les variables de la classe c'est-à-dire tous les éléments caractéristiques d'une balle :

1. C'est la particularité de tous les langages qu'on appelle « Orienté-objet » comme Java, mais aussi Python, C++ et même scratch.
2. Voire impératif !

— son rayon : variable de type `int` nommée `rayon` ;
— son abscisse : variable de type `int` nommée `x` ;
— son ordonnée : variable de type `int` nommée `y` ;
— son pas de déplacement en abscisses : variable de type `int` nommée `pasx` ;
— son pas de déplacement en ordonnées : variable de type `int` nommée `pasy` ;
— sa couleur : variable de type `color` nommée `couleur` ;

On ajoute donc entre les accolades les lignes suivantes (nous reviendrons plus tard sur le mot clé `private` mais il faut savoir qu'il est toujours préférable de déclarer les variables d'une classe ainsi) :

```
1  private int rayon; private int x; private int y;
2  private int pasx; private int pasy; private color coul;
```

Ensuite, il faut écrire un (ou plusieurs) *constructeur(s)*, c'est-à-dire une partie de code qui sera exécutée à chaque fois qu'on crée une nouvelle variable[3] de type `Balle`. Ce(s) constructeur(s) permet(tent) par exemple d'initialiser les variables il(s) prend(nent) donc des paramètres.
Nous allons créer un constructeur qui prend pour paramètres le rayon et les coordonnées et qui initialise les pas à des valeurs arbitraires. À la suite des définitions de variables :

```
1  Balle(int nRayon, int nX, int nY) {// constructeur qui
       initialise rayon, x et y
2    rayon = nRayon;
3    x = nX; y = nY;
4    pasx = 3; pasy = 2;
5    coul = #ABC0C2;// couleur en hexadécimal
6  }
```

Dans le programme principal (le premier onglet de l'éditeur Processing), nous pouvons désormais créer des balles ainsi :

```
1 Balle balle1 = new Balle(10,200,200); // crée une balle de rayon
     10 en position (200,200)
2 Balle balle2 = new Balle(20,100,300); // crée une balle de rayon
     20 en position (100,300)
```

10.2 Utilisation et actions

Nous avons créé deux balles mais, pour le moment, on ne peut rien en faire. Il va falloir ajouter des *méthodes* à la classe `Balle` afin d'agir sur les balles : les modifier, les afficher, les cacher, ... Revenons à notre onglet `Balle` et créons une méthode `affiche()` qui affiche la balle à sa position et avec son rayon et sa couleur :

```
1  void affiche() {// méthode d'affichage de la balle
2    pushStyle();// pour modifier les styles PROVISOIREMENT
```

3. Dans le vocabulaire courant des langages objet la *classe* est la définition des différents paramètres et méthodes du type créé et une variable de ce type est appelée une *instance* de la classe.

```
3       noStroke();
4       ellipseMode(RADIUS);
5       fill(coul);
6       ellipse(x,y,rayon,rayon);
7     popStyle();// fin de modif des styles
8   }
```

Créons aussi une méthode qui augmente les coordonnées des pas et une qui teste la sortie de la fenêtre :

```
1   void mouvement() {
2     x = x + pasx; y = y + pasy;
3     testbords();
4   }
5
6   private void testbords() {
7     if (x < rayon) {
8       x = rayon; pasx = - pasx;
9     }
10    if (x > width - rayon) {
11      x = width - rayon; pasx = -pasx;
12    }
13    if (y < rayon) {
14      y = rayon; pasy = -pasy;
15    }
16    if (y > height - rayon) {
17      y = height - rayon; pasy = -pasy;
18    }
19  }
```

Remarque : la méthode `testbords` est affublée du mot clé `private` pour qu'elle ne soit pas accessible depuis l'extérieur de la classe (c'est-à-dire depuis le programme principal) ; en effet, cette méthode n'a d'intérêt que pour la méthode `mouvement`.
Désormais dans le programme principal (le premier onglet de l'éditeur Processing), nous pouvons compléter une méthode `setup` et une `draw` :

```
1  void setup() {
2    size(400,400);
3    frameRate(24);
4    background(255);
5  }
6  void draw() {
7    background(255);// on efface l'écran
8    // affichage des balles :
9    balle1.affiche();
10   balle2.affiche();
11   // on effectue le mouvement :
```

```
12    balle1.mouvement();
13    balle2.mouvement();
14  }
```

10.3 À vous...

Exercice 10.1.
Créer dans la classe `Balle` les méthodes suivantes et les utiliser dans le programme principal :

- une méthode `setColor` qui prend un paramètre de type `color` et qui modifie la couleur de la balle ;
- une méthode `setPas` qui prend deux paramètres de type `int` et qui modifie les variables `pasX` et `pasY` ;
- une méthode `setRayon` qui prend un paramètre de type `int` et qui modifie le rayon de la balle ;
- deux méthodes `getPasX` et `getPasY` sans paramètre qui renvoient respectivement les valeurs de `pasX` et `pasY` ;
- deux méthodes `getX` et `getY` sans paramètre qui renvoient respectivement les valeurs de `x` et `y` ;
- une méthode `testChoc` qui prend en paramètre un objet de type `Balle` et qui renvoie `true` si les balles se touchent et `false` sinon ;
- une méthode `rebond` qui prend en paramètre un objet de type `Balle` et qui change le signe de `pasX` et `pasY` des deux balles (celle à qui on applique la méthode et celle passée en paramètre) ;
- une méthode `amorti` qui ralenti la balle (cette méthode pourra être appelée lors de chaque choc) ;
- ...

Ces méthodes doivent permettre de modifier le programme en lui ajoutant par exemple :

```
1   // Dans la méthode setup() :
2   balle1.setColor(#ABC0C2);
3   // Dans la méthode draw() :
4   if (balle1.testChoc(balle2)) {
5     balle1.rebond(balle2);
6   }
```

Exercice 10.2.
Modifier le programme précédent en utilisant un tableau de 3, 5 puis 10 (voire plus) balles.
Aide : déclarer le tableau de `Balle` dans le préambule ainsi : `Balle[] balles = new Balle [10];` puis dans la méthode `setup()` créer les balles grâce à une boucle :

```
1   for (int i=0; i<10; ++i) {
2     balles[i] = new Balle(10,20,30*i+30);
3   }
```

TP 11

Multimédia et fichiers

Nous avons déjà vu dans le TP 7 comment insérer des images et même des `gif` animés.
Dans ce TP, nous allons (re)voir comment importer des images, du son, mais aussi comment exporter une copie d'écran de notre fenêtre voire des vidéos.

11.1 Utiliser des images

11.1.1 Insertion simple

Pour insérer une image dans un projet Processing, c'est assez simple...

1. Faire glisser l'image (format `png`, `jpeg`, `gif` – mais il ne retiendra que la première image s'il contient une animation) depuis l'explorateur de fichier sur la fenêtre du sketch : un dossier `data` est alors créé dans le répertoire de votre sketch et votre fichier image est copié dans ce dossier.
2. Créer une variable de type `PImage` que nous nommerons par exemple : `monImage` par l'instruction `PImage monImage;`
3. Charger l'image dans cette variable : `monImage = loadImage("fichier.png");`.
4. Afficher l'image aux coordonnées `(x;y)` par l'instruction `image(monImage,x,y);`

Finalement, notre sketch minimal contient :

```
1 PImage monImage;
2 void setup() {
3   size(800,600);
4   monImage = loadImage("test2.png");
5   image(monImage,50,50);
6 }
```

11.1.2 Effets

Parmi les effets élémentaires, on peut :

— redimensionner l'image : `image(monImage,50,50,100,100);` impose à l'image un format carré de 100 pixels de côté. Attention aux déformations !
Pour que l'image remplisse l'écran il suffit donc d'écrire : `image(monImage,0,0,width, height);`
— transparence : penser à utiliser le canal alpha dans votre logiciel d'infographie préféré (dans ce cas, utiliser le format `png` ou `gif` car le `jpeg` ne gère pas ce canal alpha). Ceci vous sera utile pour insérer des images les unes « au dessus des autres » ;
— couleurs : il est possible de modifier les couleurs grâce à l'instruction `tint()`. Celles et ceux qui sont intéressés trouveront la documentation nécessaire sur internet, notamment ici : http://www.ecole-art-aix.fr/processing-06-Images où on trouve l'exemple ci-dessous à tester avec une photo `maPhoto.jpg` par exemple...

```
1  PImage monImage;
2  void setup(){
3    size(600,400);
4    monImage = loadImage("maPhoto.jpg");
5  }
6  void draw(){
7    background(0);
8    tint(mouseX, mouseY, mouseY-mouseX,255);
9    image(monImage,0,0);
10 }
```

11.1.3 Animation

Pour animer (déplacer) une image, il suffit de faire varier sa position dans la méthode `draw()` :

```
1  PImage monImage;
2  int x = 0, pasx = 3;
3  void setup() {
4    size(800,600);
5    monImage = loadImage("test.png");
6  }
7  void draw() {
8    background(0);
9    image(monImage,x,100,100,100);
10   x = x + pasx;
11   if (x > width - 100) {
12     x = width - 100; pasx = -pasx;
13   }
14   if (x < 0) {
15     x = 0; pasx = -pasx;
16   }
17 }
```

11.1.4 Et les gif :-(

Malheureusement, Processing ne gère pas nativement les gif animés. Mais vous savez que (presque) rien n'est impossible en informatique, nous allons donc quand même réussir à animer un personnage (ou autre chose) grâce, au choix, à :

- la librairie `GifAnimation` disponible ici : https://github.com/01010101/GifAnimation (et déjà vue dans le TP 7) ;
- une classe trouvée sur le site de Processing et légèrement modifiée. Elle n'est pas très difficile à comprendre, je l'ai commentée ci-dessous.

```
1  // Classe pour animer une séquence d'images png
2  class Animation {
3    PImage[] images;// tableau des images
4    int imageCount;// nombre d'images
5    float frameR;// pour passer les images moins vite que 1/frame
6    int frame;// numéro d'image à afficher
7
8    // Un constructeur de la classe :
9    Animation(String imagePrefix, int count) {// on fournit le
       préfixe du nom ainsi que le nombre d'images.
10     // Si préfixe = image alors :
11     // les images se nomment images-00.png, image-01.png, ...
12     imageCount = count;
13     images = new PImage[imageCount];
14
15     for (int i = 0; i < imageCount; i++) {
16       // nf() on ajoute i avec 2 chiffres ex : 1 s'écrit 01
17       // création du nom de l'image "i"
18       String filename = imagePrefix + nf(i, 2) + ".png";
19       images[i] = loadImage(filename);// chargement
20     }
21   }
22
23   // méthode d'affichage :
24   void affiche(float xpos, float ypos) {
25     frameR = frameR+.02;// on augm frameR pour chger d'image
26     frame = (int)(frameR) % imageCount;
27     image(images[frame], xpos, ypos);// on affiche l'image frame
28   }
29
30   // Méthodes pour obtenir largeur et hauteur de l'image
31   int getWidth() { return images[0].width;  }
32   int getHeight() { return images[0].height; }
33 }
```

Imaginons que nous ayons 15 images nommées `image-00.png` ... `image-14.png`. Dans le pro-

gramme principal, on définit un objet de type `Animation`, puis, on crée cet objet et enfin, on l'affiche dans la méthode `draw()` :

```
1 Animation monSprite;
2 void setup() {
3   monSprite = new Animation("image-",15);
4   size(300,300);
5 }
6 void draw() {
7   monSprite.affiche();
8 }
```

11.2 Insérer des sons

Pour insérer un son dans un programme, il faut commencer par récupérer un fichier au format `mp3`, `aiff` ou `wav` et le faire glisser à la souris sur le sketch (la zone où on écrit le code Processing). Un dossier `data` est alors créé dans le répertoire où est enregistré le sketch et le fichier son est copié dedans.
Pour exploiter ce fichier, nous allons utiliser une nouvelle *bibliothèque* (c'est-à-dire un lot de nouvelles instructions qui ne sont pas dans Processing au départ). La première chose à faire est d'installer cette bibliothèque qui s'appelle `minim` : dans le menu `Sketch`, sélectionner `Import Library`, puis choisir `Minim Audio`. Cette étape ne sera plus à faire pour vos projets futurs.
Dans le code du programme, en dehors de toute méthode, il faut indiquer qu'on va utiliser cette librairie, ensuite créer un *objet* de type `Minim` appelé `minim` et enfin un « joueur de sons » appelé `bip` :

```
1 // Pour les "bips" :
2 import ddf.minim.*;// Bibliothèque à installer par menu sketch ->
      importer une librairie
3 Minim minim;
4 AudioPlayer bip;
```

Dans la méthode `setup`, il reste à configurer le tout :

```
1 void setup() {
2   ...
3   minim = new Minim(this);// inutile de comprendre tout ça pour
      le moment
4   bip = minim.loadFile("monFichier.mp3");
5 }
```

Enfin, pour jouer le son (au moment des rebonds par exemple), il suffit d'écrire :

```
1 bip.play();// On joue le son
2 bip.rewind();// On replace la lecture au début
```

Remarque : il faudra prévoir une méthode `stop()` qui fermera les fichiers sons.

```
void stop() {
  bip.close();
  minim.stop();
  super.stop();
}
```

Plus de détails ici en suivant les liens ci-dessous :

- http://fr.flossmanuals.net/processing/la-lecture-du-son/
- http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.OUTILS ProcessingGUIControlP5MinimLecteurMP3Buttons
- http://code.compartmental.net/tools/minim/quickstart/

11.3 Les imports/exports

11.3.1 Lecture d'un fichier texte

Pour créer la « map » d'un jeu de labyrinthe ou de pacman ou ..., il peut être utile de la sauvegarder au format texte dans un fichier externe. Par exemple, un tel fichier pourrait avoir comme convention : un « `X` » est un mur et une espace est un endroit libre où le personnage peut se déplacer. Une map est alors un fichier texte qui contient, par exemple une première ligne avec le numéro du niveau, et ensuite la liste de `X` et d'espaces[1]. Dans notre exemple le fichier s'appelle `map3.txt` :

```
3
XXXXXXXXXXXXXX
X      X       X
X XXX    XXXX  X
X X    X    X   X
X X X X X X X XX
X    X    X    XX
XXXXXXXXXXXXXX
```

Une fois un tel fichier créé et placé dans le répertoire `data`, il faut l'importer dans notre programme :

```
String[] monTexte; // tableau qui contient les lgn du fichier txt
monTexte = loadStrings("map3.txt");// chargement du fichier
int niveau = parseInt(monTexte[0]);// on récupère le niveau
int nbLignesMap = monTexte.length - 1;// à cause de la 1ère ligne
size(300,300);
fill(0);
rectMode(CORNER);
```

1. Ça ressemble étrangement aux formats bitmap vus dans le TP 7 non ?

```
// 2 boucles sur les lignes et colonnes :
for (int i=1; i<=nbLignesMap; ++i) {
  for(int j = 0; j < monTexte[i].length(); ++j) {
    if (monTexte[i].charAt(j) == 'X') {
      // si le fichier contient X, on dessine un carré :
      rect(10*j, 10*i, 10, 10);
    }
}}// fin des deux boucles
```

11.3.2 Écriture dans un fichier texte

Il peut aussi être utile de sauvegarder des informations dans un fichier (par exemple un profil de joueur ou un tableau des meilleurs scores). Nous allons écrire le contenu d'un tableau `joueur` et d'un tableau `score` dans un fichier `record.txt` (un couple joueur-score par ligne) :

```
PrintWriter output;// Variable qui contient le nom du fichier
String[] joueur = {"Riri", "Fifi", "Loulou"};
int[] score = {1000,800,100};
output = createWriter("record.txt");
for (int i = 0; i < 3; ++i) {
  // On écrit dans le fichier :
  output.println(joueur[i] + " " + score[i]);
}
output.flush();// finalise l'écriture
output.close();// ferme le fichier
```

Et on obtient un fichier `record.txt` qui contient :

```
Riri 1000
Fifi 800
Loulou 100
```

On pourra relire ce fichier ultérieurement grâce à la méthode décrite dans le paragraphe précédent.

11.3.3 Copies d'écran, vidéos

Cette partie est moins essentielle à la réalisation de vos projets. Je vous laisse donc quelques liens.
Pour sauvegarder une frame au format tiff : https://processing.org/reference/save_.html ou https://processing.org/reference/save_.html.
Il est possible d'écrire dans un fichier pdf le contenu de la fenêtre graphique. Je vous conseille la lecture suivante : http://arts-numeriques.codedrops.net/PDF-Produire-un-pdf-avec.
Un exemple d'utilisation de la webcam :
http://arts-numeriques.codedrops.net/Son-et-video-declencher-un-son.

TP 12

Compléments divers

Dans ce « faux » TP, je vous donnerai quelques aides pour la réalisation du projet.

12.1 Plusieurs fenêtres...

Votre projet contiendra inévitablement plusieurs fenêtres distinctes (présentations, règles du jeu, jeu, crédits, ...). Mais dans un projet Processing, on ne peut écrire qu'une seule méthode `draw()` donc définir une seule fenêtre graphique. Pour y remédier nous allons ruser en utilisant une instruction conditionnelle `switch`.

- Première chose : définir une variable `fenetre` de type `int` qui va contenir l'indice de la fenêtre à afficher (0 pour la présentation, 1 pour les règles, 2 pour les crédits, ...).
- Ensuite, nous allons créer des méthodes de type `void` appelée par exemple `presentation()`, `regles()`, `credits()`, ... qui contiendront les codes des fenêtres à afficher.
- Enfin on écrit la méthode `draw()` ainsi :

```
1 void draw() {
2   switch(fenetre) {
3     case 0: presentation(); break;
4     case 1: regles(); break;
5     case 2: credits(); break;
6     ...
7   }
8 }
```

Attention : la variable `fenetre` doit être définie dès le début du sketch pour être *globale* et accessible depuis tout le programme.

Remarque : pour plus de lisibilité du projet, il est fortement conseillé d'écrire chacune des méthodes `presentation()`, `regles()`, ... dans un onglet séparé de votre projet. Au démarrage du programme, Processing compile le contenu de tous les onglets automatiquement.

12.2 À propos de classes

Nous avons vu dans le TP 10, que l'intérêt d'un langage objet est de créer des *classes* qui définissent des *objets* (super-variable en quelque sorte). Si votre projet est un jeu dans lequel on doit animer un personnage par exemple, il parait raisonnable de définir un objet personnage dans une classe. Je vous présente ici un exemple de classe et une utilisation d'une instance (c'est-à-dire d'une variable ayant pour type la classe définie) de cette classe.

```
1  class Personnage {// avec un P majuscule !
2    // Définition des variables du personnage :
3    String nom;
4    int xpos, ypos;// positions dans la map
5    PImage avatar;// image
6    int points;// points obtenus
7
8    // Constructeur de l'objet :
9    Personnage(String le Nom, String fichier) {
10     nom = leNom;
11     avatar = loadImage(fichier);
12     xpos = 50; ypos = 50;
13     points = 0;
14   }
15
16   // Méthode de déplacement du personnage :
17   void deplace(int pasx, int pasy) {
18     x = x + pasx;
19     y = y + pasy;
20   }
21
22   // Méthode d'affichage du personnage :
23   void affiche() {
24     image(avatar, xpos, ypos);
25   }
26
27   // Augmenter les points
28   void addPoints(int gain) {
29     points = points + gain;
30   }
31
32   // Tester si le personnage rencontre un obstacle :
33   void testCollision() {
34     // à écrire
35   }
36
37 }// Fin de la classe
```

Dans le programme principal on écrit alors :

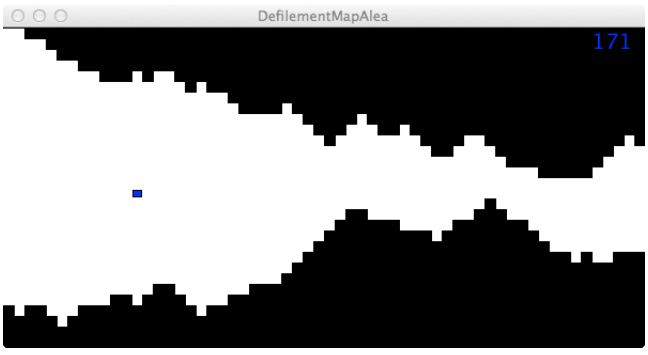
```
Personnage joueur = new Personnage("Toto", "toto.png");

void setup() {
  size(800,600);
  // lecture de map...
  // initialisations diverses...
}

void draw() {
  joueur.affiche();
  joueur.deplace(1,0);//déplacement horizontal
  joueur.testCollision();
  if (victoire) {
    joueur.addPoints(10);
  }
  ...
}
```

Les avantages de procéder ainsi sont nombreux :

- décomposer le codage en éléments « simples » ;
- faciliter la gestion de plusieurs personnages ;
- faciliter la possibilité de changer des attributs du personnage (image, nom, dots) ;
- ...

12.3 Jeu de « défilement »

Quelques-uns d'entre vous auront besoin de faire défiler une map à l'écran. Je vous donne ci-dessous un exemple de défilement horizontal d'une map aléatoire dans laquelle doit se déplacer un objet (sous marin, oiseau, alien, ...) :



Dans un premier temps, je vous conseille d'étudier le code sans tenir compte de « l'oiseau » :

```
// Exemple de défilement d'un tableau de jeu

```

```
3  // Thomas REY le 29/2/2016
4  // Dernière modif : 29/2/2016
5
6  int[] tabHaut = new int[61];// nbre de blocs noir en haut
7  int[] tabBas = new int[61];// nbre de blocs noirs en bas
8
9  void setup() {
10   size(600,300);
11   background(255);
12   fill(0);
13   rectMode(CORNER);
14   frameRate(6);
15 }
16
17 void draw() {
18   background(255);
19   fill(0);
20   for (int i=0; i<60; ++i) {
21     rect(10*i,0,10,10*tabHaut[i]);
22     rect(10*i,300, 10, -10*tabBas[i]);
23   }
24 }
25
26 void avance() {
27   for (int i=0; i<60; ++i) {
28     tabHaut[i] = tabHaut[i+1];
29     tabBas[i] = tabBas[i+1];
30   }
31   // Prochain mur du haut descend de +1/0/-1 plus bas que le
       précédent
32   tabHaut[60] = min(28, tabHaut[59]+(int)random(0,3)-1);
33   // Mais pas à l'extérieur de l'écran.
34   tabHaut[60] = max(0,tabHaut[60]);
35
36   // Idem pour le bas (26 - tabHaut permet d'avoir au moins 3
       cases libres)
37   tabBas[60] = min(26-tabHaut[60], tabBas[59]+(int)random(0,3)-1)
       ;
38   tabBas[60] = max(0,tabBas[60]);
39 }
```

Puis le code complet du jeu :

```
1  // Exemple de défilement d'un tableau de jeu qui contient un
       oiseau qui peut
2  // monter ou descendre
```

```
3  // qui avance automatiquement avec la map
4  // qui peut avancer plus vite et accéder jusqu'à proximité du
       bord droit de l'écran
5  // En bref un AngryBird simplifié
6
7  // Thomas REY le 29/2/2016
8  // Dernière modif : 29/2/2016
9
10 int[] tabHaut = new int[61];// nbre de blocs noir en haut
11 int[] tabBas = new int[61];// nbre de blocs noirs en bas
12
13 int altOiseau=15;// attention l'oiseau est en haut : altOiseau =
       0 et en bas =30 !
14 float xOiseau = 30;
15 int score =0;
16
17 // Tableau qui contient les touches enfoncées
18 // Ceci permet de gérer les deux joueurs en même temps
19 // sinon, seule la dernière touche enfoncée est en mémoire et
20 // si le joueur laisse une touche enfoncée, les aures ne sont pas
       "lues"
21 boolean[] touches = new boolean[256];
22
23 void setup() {
24   size(600,300);
25   background(255);
26   fill(0);
27   rectMode(CORNER);
28   frameRate(6);
29 }
30
31 void draw() {
32   ++score;
33   background(255);
34   fill(0);
35   for (int i=0; i<60; ++i) {
36     rect(10*i,0,10,10*tabHaut[i]);
37     rect(10*i,300, 10, -10*tabBas[i]);
38   }
39   fill(0,0,255);
40   rect(1+xOiseau*10,2+altOiseau*10,8,6);// Dessin à améliorer !
41   textSize(20);
42   text(score,550,20);
43   if (!testCollision()) {
44     deplaceOiseau();
```

```
45      avance();
46    } else {
47      textSize(48);
48      //textMode(CENTER);
49      fill(255,0,0);
50      text("PERDU !", 250,150);
51      frameRate(0);
52    }
53  }
54
55  void avance() {
56    for (int i=0; i<60; ++i) {
57      tabHaut[i] = tabHaut[i+1];
58      tabBas[i] = tabBas[i+1];
59    }
60    // Prochain mur du haut descend de +1/0/-1 plus bas que le
        précédent
61    tabHaut[60] = min(28, tabHaut[59]+(int)random(0,3)-1);
62    // Mais pas à l'extérieur de l'écran.
63    tabHaut[60] = max(0,tabHaut[60]);
64
65    // Idem pour le bas (26 - tabHaut permet d'avoir au moins 3
        cases libres)
66    tabBas[60] = min(26-tabHaut[60], tabBas[59]+(int)random(0,3)-1)
        ;
67    tabBas[60] = max(0,tabBas[60]);
68
69  }
70
71  // Déplacement oiseau
72  void deplaceOiseau() {
73    if (touches[39] & xOiseau < 50) {
74      xOiseau = xOiseau + 1;
75    }
76    if (touches[37] & xOiseau > 1) {
77      xOiseau--;
78    }
79    xOiseau = max(1,xOiseau-.5);
80
81    if (touches[38]) {
82      altOiseau--;
83    }
84
85    if (touches[40]) {
86      altOiseau++;
```

```
87    }
88
89  }
90
91  // Tests de collision avec les murs
92  boolean testCollision() {
93    boolean collision = false;
94    if (altOiseau < tabHaut[(int)xOiseau]) {collision = true;}
95    if (altOiseau > 30 - tabBas[(int)xOiseau]) {collision = true;}
96    return collision;
97  }
98
99  // Remplissage du tableau touches
100 void keyPressed() {
101   touches[keyCode]=true;
102 }
103
104 void keyReleased() {
105   touches[keyCode]=false;
106 }
```

Je dépose le code complet ici : http://isnmarlioz.olympe.in/lesTP.html

12.4 Communication entre sketches

À voir, les bibliothèques décrites ici :
http://www.ecole-art-aix.fr/processing-xx-Hypermedia

12.5 Création de formes

Il est possible de créer vos propres formes (pour remplacer une ellipse, un rectangle, ...) à l'aide de la classe `PShape`.
Une fois l'objet créé, on peut le déplacer, le rendre visible ou invisible, le tourner, ...
Voir ici : https://processing.org/reference/PShape.html (in english :-(). Quand j'aurai le courage, je ferai une documentation en français...

TP 13

Récursivité

D'après Wikipédia, la *récursivité* est une démarche qui fait référence à l'objet même de la démarche à un moment du processus.
Cette définition appliquée à l'informatique permet par exemple d'expliquer ce qu'est une *fonction récursive* ainsi : une fonction (ou méthode en Processing) est dite *récursive* si elle s'appelle elle-même.

13.1 Premier exemple : la factorielle

En mathématiques, si n est un entier naturel non nul, on appelle factoriel n et on note $n!$ le produit de tous les entiers naturels non nuls inférieurs ou égaux à n :

$$n! = n \times (n-1) \times \ldots \times 3 \times 2 \times 1$$

Par exemple, $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$.
Exercice 13.1.
Que font les deux méthodes suivantes ? On pourra compléter un tableau de valeurs des différentes variables dans le cas d'un appel à ces méthodes avec le paramètre $n = 5$ par exemple.

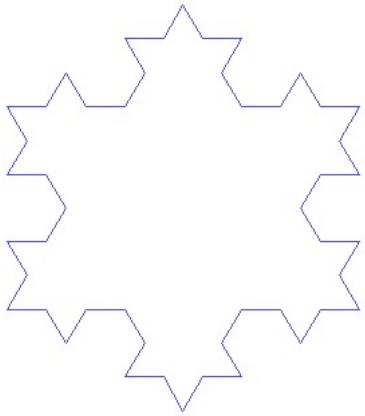
```
1  // Fonction 1 :
2  int fonc1(int n) {
3    if (n > 1) {
4      return n * fonc1(n-1);
5    }  else { return n; }
6  }
7
8  // Fonction 2
9  int fonc2(int n) {
10   int f = 1;
11   for (int i = 1; i<=n; ++i) {
12     f = f * i;
13   }
14   return f;
15 }
```

Exercice 13.2.
Écrire une méthode récursive prenant un paramètre entier n qui calcule la somme des sentiers naturels inférieurs ou égaux à n.

13.2 Pour réaliser des dessins

13.2.1 Un exemple, le flocon de Koch

À partir d'un triangle équilatéral, on partage chaque côté en trois segments de même longueur et on trace à l'extérieur un nouveau triangle équilatéral sur chaque segment « médian ». On recommence ensuite sur chacun des segments obtenus. Par exemple, après deux itérations, on obtient :



Source de l'image : Wikipédia (encore [1] !)

Pour obtenir cette figure à l'étape n quelconque, la récursivité est l'outil idéal. Dans le code suivant, on considère qu'on a créé une classe `Point` (le code est donné dans la partie 13.3) qui permet de :

- définir des points à partir de leurs deux coordonnées ;
- additionner des points (au sens des nombres complexes) ;
- multiplier des points par un réel (idem) ;
- obtenir l'image d'un point par une rotation de centre un autre point et d'angle donné en radians ;
- obtenir l'image d'un point par une homothétie (sorte d'outil à aggrandir/réduire) de centre donné et de rapport donné ;
- ...

Voici le code :

```
1 int n = 1;// nombre d'étapes à construire
2 int dn = 1;// variation de n à chaque frame
3
4 // On définit les trois sommets du triangle
5 Point C, D, E;
6
```

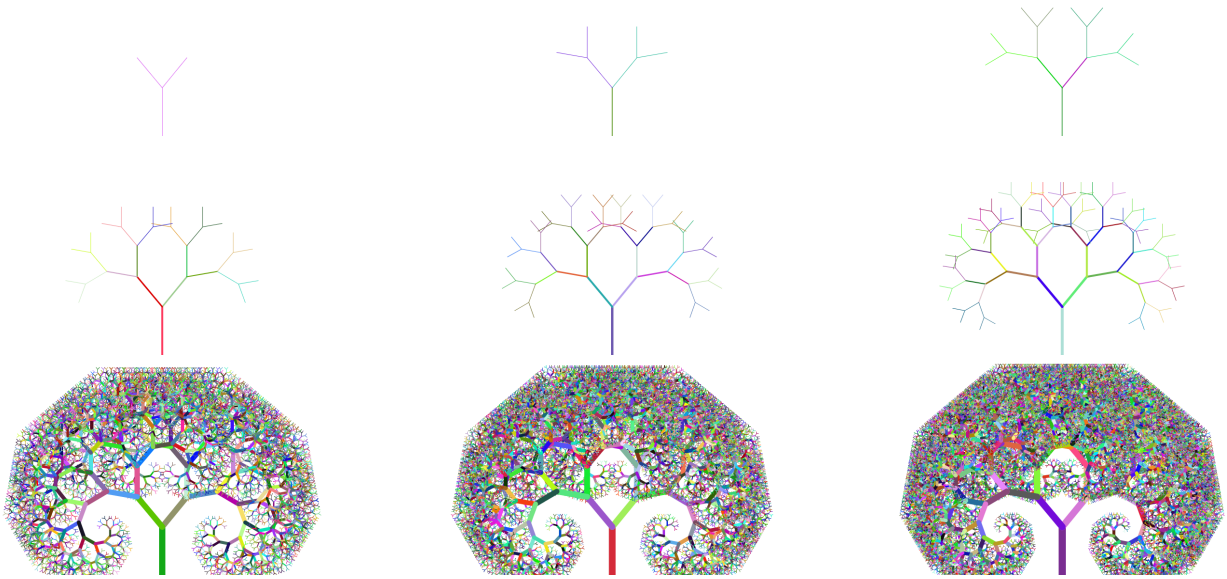
1. Si ça continue, ce document va ressembler à un TPE de première... :-)

```
7  void setup() {
8    size(800, 800);
9  // Initialisation des points :
10   C = new Point(100, 550);
11   D = new Point(700, 550);
12   E = D.rotate(C,-PI/3);// 3e sommet du triangle équilatéral
13
14   frameRate(1);// Pour une animation "lente"
15 }
16
17 void draw() {
18   background(255);
19   // On trace les trois côtés agrémentés de leurs "partages" et
       triangles supp :
20   unCote(D,C,n);
21   unCote(E,D,n);
22   unCote(C,E,n);
23   // On incrémente le nombre d'étapes
24   n = n + dn;
25   // Si on arrive à l'étape 8, on décrémente et à 0 on ré-
       incrémente...
26   if (n % 8 == 0) {
27     dn = -dn;
28   }
29 }
30
31
32 // Méthode récursive qui construit un "triangle" sur le tiers
     médian
33 // d'un segment [P1 P2] passé en paramètres
34 void unCote(Point P1, Point P2, int i) {
35   // point situé au tiers de [P1 P2]
36   Point A = new Point(2.0*P1.x/3 + P2.x/3.0, 2.0*P1.y/3 + P2.y
       /3.0);
37   // Point situé aux 2/3 de [P1 P2]
38   Point B = new Point(P1.x/3.0 + 2.0*P2.x/3, P1.y/3.0 + 2.0*P2.y
       /3);
39   // Sommet du triangle supplémentaire :
40   Point C = B.rotate(A,-PI/3);
41
42   // Si le paramètre i>0, on effectue une itération
       supplémentaire
43   if (i > 0) {
44     unCote(P1, A, i-1);
45     unCote(A, C, i-1);
```

```
46      unCote(C, B, i-1);
47      unCote(B, P2, i-1);
48    } else {// Sinon on trace le segment :
49      line(P1.x, P1.y, P2.x, P2.y);
50    }
51  }
```

13.2.2 Un arbre Pythagoricien

Exercice 13.3.
En utilisant la classe `Point`, créer une méthode récursive permettant d'obtenir successivement des images du type suivant :



13.3 La classe Point

La classe Point documentée :

```
1  // Cette classe permet de définir des points
2  // (plus faciles à manimuler que des couples de coordonnées)
3  class Point {
4    float x;
5    float y;
6
7    // Constructeur avec des float
8    Point(float a, float b) {
9      this.x = a;
```

```
10      this.y = b;
11    }
12    // Constructeur avec des int
13    Point(int a, int b) {
14      this.x = (float)a;
15      this.y = (float)b;
16    }
17
18    // Méthode qui permet "d'additionner" deux points en
         additionnant leurs
19    // coordonnées respectives (utile pour trouver un barycentre)
20    Point add(Point P) {
21      Point Q = new Point(P.x+this.x, P.y + this.y);
22      return Q;
23    }
24
25    // Méthode qui multiplie les coordonnées par un nombre
26    // (utile pour trouver un barycentre)
27    Point mult(float k) {
28      Point Q = new Point(k*this.x, k*this.y);
29      return Q;
30    }
31
32    // Méthode qui renvoie l'image de (this.x, this.y) par rotation
         centre O, angle a
33    Point rotate(Point O, float a) {
34      float qx = O.x + (this.x-O.x)*cos(a) - (this.y-O.y)*sin(a);
35      float qy = O.y + (this.x-O.x)*sin(a) + (this.y-O.y)*cos(a);
36      Point Q = new Point(qx, qy);
37      return Q;
38    }
39
40    // Méthode qui renvoie l'image de (this.x, this.y) par
41    // l'homothétie de centre O et de rapport k
42    Point homothetie(Point O, float k) {
43      float qx = O.x + k * (this.x - O.x);
44      float qy = O.y + k * (this.y - O.y);
45      Point Q = new Point(qx, qy);
46      return Q;
47    }
48
49  }// Fin de la classe
```

TP 14

Exporter une application

Vous avez maintenant fini votre projet (enfin j'espère !), il serait pas mal de le diffuser. Il existe un menu dans Processing permettant d'exporter votre projet en application autonome pour Windows, Mac ou Linux. Il est aussi possible de l'exporter en une version « en ligne » grâce à Processing.js ou en une application mobile (sous forme d'une apk pour Android) grâce à l'application APDE.

14.1 Processing.js

et pour cela, une version « online » est le meilleur moyen. Votre projet est écrit en processing version java donc il n'est pas exécutable dans un navigateur. Pour qu'il le devienne, nous allons passer un par script écrit en javascript (langage qui, à part les quatre première lettres, n'a pas grand-chose à voir avec java) qui va faire la conversion pour le navigateur.

14.1.1 Le(s) script(s)

Le script de base est développé par une équipe qui publie ses ressources sur un site web : http://processingjs.org. Vous y trouverez la dernière version du fichier `processing.min.js` ainsi que des explications et une page de références sur toutes les fonctions implémentées (tout ça, in english of course !).
Par ailleurs, il existe d'autres scripts permettant d'émuler des bibliothèques externes comme la bibliothèque `Minim` utilisée pour agrémenter de sons vos programmes ProcessingJava. Ce script est disponible ici : https://github.com/Pomax/Pjs-2D-Game-Engine/blob/master/minim.js.
D'autres se trouvent par l'intermédiaire de votre moteur de recherche préféré (https://www.qwant.com par exemple ; oubliez un peu l'autre !).

14.1.2 Comment ça marche ?

1. On commence par créer un dossier dans lequel on dépose le fichier `processing.min.js` (éventuellement `minim.js` aussi). Ensuite, on crée dans ce dossier un fichier `index.html` qui contient au minimum les lignes suivantes :

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
    <title>Mon titre de page web</title>
    <script src="processing.min.js"></script>
    <!-- Eventuellement en plus : -->
    <script src="minim.js"></script>
</head>
<body>
    <h1>Mon projet Processing</h1>

    <p>Projet réalisé au cours de l'année 2017/2018.</p>
    <canvas data-processing-sources="Projet.pde"></canvas>
    <!-- où Projet.pde est LE fichier .pde contenant TOUT le
       projet -->
</body>
</html>
```

On peut ajouter éventuellement d'autres éléments à la page web ainsi qu'une feuille de style...

2. Ensuite, il faut recopier le contenu de *tous* les onglets de la fenêtre Processing de votre projet dans *un seul* fichier `.pde` (qu'on a appelé `Projet.pde` dans l'exemple ci-dessus) ; fichier à placer dans le même [1] répertoire que `index.html`.
3. Ensuite, on vient déposer tout le contenu du dossier `data` de votre projet dans le dossier où se trouve le fichier `Projet.pde` et on liste tous les fichiers images utilisées dans le projet (`image1.jpg`, `image2.png`, ...).
   En processingJava, il n'existe pas de fonction `preload` (comme en P5js pour ceux qui connaissent) permettant de charger *avant* le début du programme l'ensemble des images, il faut donc ajouter les trois lignes suivantes à votre `Projet.pde`, *impérativement* au début du fichier :

```
/*
 @pjs preload="image1.jpg,image2.png"
*/
```

Ceci évite que vous demandiez d'afficher une image *avant* qu'elle n'ait eu le temps d'être chargée par le navigateur.

Attention : pas d'espace dans les noms de fichiers, sinon, il faut les renommer et modifier les `loadImage` en conséquence. L'ensemble de la liste des images doit figurer sur une seule ligne.

Si tout se passe bien [2], il ne reste plus qu'à ouvrir le fichier `index.html` dans un navigateur et

1. Vous pouver placer le pde dans un sous-dossier, mais dans ce cas, il faut indiquer le bon chemin dans la page html.
2. Ce qui, à ce stade, serait un grand coup de chance (voir partie 14.1.3) !

votre projet apparaît !
Si ce n'est pas le cas, pas de panique, il faudra faire quelques corrections dans le fichier `Projet.pde`. Pensez à conserver une copie du projet initial qui fonctionne sous Processing !

14.1.3 Attention !

La difficulté principale vient du fait que lorsque le navigateur [3] ne réussit pas à exécuter le projet, il renvoie une erreur liée à une ligne du script `processing.min.js` qui lui, ne nous donne que très peu d'informations sur ce qu'il ne comprend pas dans votre fichier `Projet.pde`.

Une liste non exhaustive des difficultés pouvant planter votre projet :
- En javascript, il n'y a pas de variable de type `char`, donc le résultat de `maVariable.charAt(4)` est par exemple `"X"` (et non pas `'X'` comme en Processing).
  Conséquence : dans vos lectures de maps par exemple, remplacer tous les `'x'` par des `"x"` ; l'éditeur de Processing vous le signalera en erreur mais ce n'est pas grave, ça fonctionnera dans le navigateur.
- Javascript est beaucoup plus capricieux que Java pour les noms de variables/méthodes. Ainsi, en java, il est possible d'avoir une variable `menu` de type `int` par exemple et une méthode `void menu()`. Lorsque vous écrivez `2*menu+4`, java comprend que vous faites référence à la *variable* `menu` et lorsque vous écrivez `menu()`, il comprend que vous faites un appel à la *méthode* `menu`. Javascript ne réussit pas à faire ce tour de « passe-passe ». Ceci se manifeste par une erreur du type `menu is not an object`
  Conséquence : il faut renommer soit le nom de la variable `menu`, soit le nom de la méthode du même nom. Attention, ceci est à faire dans les définitions mais aussi dans tous les appels (à vous de voir qui de la variable et de la méthode apparait le moins souvent...).
- La méthode `toString` appliqué à la classe `Float` (par exemple) ne fonctionne pas.
  Conséquence : si vous avez un `Float.toString(maVariable)`, le remplacer par `str(maVariable)`.
- ...

14.2 Créer une apk avec APDE

C'est, a priori, plus simple que la partie précédente :
- Téléchargez sur votre mobile android l'application APDE. (N'existe pas sous iOS, changez de mobile si besoin !)
- Transférez votre sketch complet (avec les dossiers de datas, ...) sur le mobile (par bluetooth ou avec une appli sur votre ordinateur). Placez-le au bon endroit !
- Vérifiez qu'il fonctionne dans APDE.
- Ajoutez une icône.
- Signez l'appli. :
  - Dans `Tools`, choisissez `Export Signed Package` ;

3. J'ai remarqué que le navigateur donnant le plus d'informations sur les erreurs est Firefox. (Afficher la fenêtre « Outil du développeur » puis la « Console Web »).

  - Lors de la première exportation d'une application, créez un « keystore » en cliquant sur le « + » ; attention, il est impossible de stocker votre keystore (magasin de clés) n'importe où, pour moi ça fonctionne en le créant ici : `/storage/emulated/0/keystore.jks` (choisir un mot de passe que vous n'oublierez pas !) ;
  - une fois le keystore créé, il suffit de créer une clé (et un mot de passe) ;
  - Cliquez sur `EXPORT` ;
- Il reste à récupérer [4] le fichier `apk` qui a été créé dans un dossier `bin` de votre dossier de sketch ;
- Diffusez ce fichier apk.

La documentation d'APDE est ici : https://github.com/Calsign/APDE/wiki/Getting-Started.

Attention, les `keyCode` des touches du clavier android ne sont pas les même que sur un clavier d'ordinateur (ça serait trop simple...) :
Quelques codes de touches du clavier android (trouvé ici : https://stackoverflow.com/questions/11768356/need-table-of-key-codes-for-android-and-presenter) :

```
0 -->  "KEYCODE_0"
1 -->  "KEYCODE_SOFT_LEFT"
2 -->  "KEYCODE_SOFT_RIGHT"
3 -->  "KEYCODE_HOME"
4 -->  "KEYCODE_BACK"
5 -->  "KEYCODE_CALL"
6 -->  "KEYCODE_ENDCALL"
7 -->  "KEYCODE_0"
8 -->  "KEYCODE_1"
9 -->  "KEYCODE_2"
10 -->  "KEYCODE_3"
11 -->  "KEYCODE_4"
12 -->  "KEYCODE_5"
13 -->  "KEYCODE_6"
14 -->  "KEYCODE_7"
15 -->  "KEYCODE_8"
16 -->  "KEYCODE_9"
17 -->  "KEYCODE_STAR"
18 -->  "KEYCODE_POUND"
19 -->  "KEYCODE_DPAD_UP"
20 -->  "KEYCODE_DPAD_DOWN"
21 -->  "KEYCODE_DPAD_LEFT"
22 -->  "KEYCODE_DPAD_RIGHT"
23 -->  "KEYCODE_DPAD_CENTER"
24 -->  "KEYCODE_VOLUME_UP"
25 -->  "KEYCODE_VOLUME_DOWN"
26 -->  "KEYCODE_POWER"
27 -->  "KEYCODE_CAMERA"
28 -->  "KEYCODE_CLEAR"
29 -->  "KEYCODE_A"
30 -->  "KEYCODE_B"
31 -->  "KEYCODE_C"
32 -->  "KEYCODE_D"
33 -->  "KEYCODE_E"
34 -->  "KEYCODE_F"
35 -->  "KEYCODE_G"
36 -->  "KEYCODE_H"
37 -->  "KEYCODE_I"
38 -->  "KEYCODE_J"
39 -->  "KEYCODE_K"
40 -->  "KEYCODE_L"
41 -->  "KEYCODE_M"
42 -->  "KEYCODE_N"
43 -->  "KEYCODE_O"
44 -->  "KEYCODE_P"
45 -->  "KEYCODE_Q"
46 -->  "KEYCODE_R"
47 -->  "KEYCODE_S"
48 -->  "KEYCODE_T"
49 -->  "KEYCODE_U"
50 -->  "KEYCODE_V"
51 -->  "KEYCODE_W"
52 -->  "KEYCODE_X"
53 -->  "KEYCODE_Y"
```

4. Pour une raison étrange, ce dossier n'apparaît pas tout de suite dans mon logiciel de connection du téléphone à l'ordinateur, il faut parfois redémarrer le téléphone pour y avoir accès. Étrange...

```
54 -->  "KEYCODE_Z"
55 -->  "KEYCODE_COMMA"
56 -->  "KEYCODE_PERIOD"
57 -->  "KEYCODE_ALT_LEFT"
58 -->  "KEYCODE_ALT_RIGHT"
59 -->  "KEYCODE_SHIFT_LEFT"
60 -->  "KEYCODE_SHIFT_RIGHT"
61 -->  "KEYCODE_TAB"
62 -->  "KEYCODE_SPACE"
63 -->  "KEYCODE_SYM"
64 -->  "KEYCODE_EXPLORER"
65 -->  "KEYCODE_ENVELOPE"
66 -->  "KEYCODE_ENTER"
67 -->  "KEYCODE_DEL"
68 -->  "KEYCODE_GRAVE"
69 -->  "KEYCODE_MINUS"
70 -->  "KEYCODE_EQUALS"
71 -->  "KEYCODE_LEFT_BRACKET"
72 -->  "KEYCODE_RIGHT_BRACKET"
73 -->  "KEYCODE_BACKSLASH"
74 -->  "KEYCODE_SEMICOLON"
75 -->  "KEYCODE_APOSTROPHE"
76 -->  "KEYCODE_SLASH"
77 -->  "KEYCODE_AT"
78 -->  "KEYCODE_NUM"
79 -->  "KEYCODE_HEADSETHOOK"
80 -->  "KEYCODE_FOCUS"
81 -->  "KEYCODE_PLUS"
82 -->  "KEYCODE_MENU"
83 -->  "KEYCODE_NOTIFICATION"
84 -->  "KEYCODE_SEARCH"
85 -->  "KEYCODE_MEDIA_PLAY_PAUSE"
86 -->  "KEYCODE_MEDIA_STOP"
87 -->  "KEYCODE_MEDIA_NEXT"
88 -->  "KEYCODE_MEDIA_PREVIOUS"
89 -->  "KEYCODE_MEDIA_REWIND"
90 -->  "KEYCODE_MEDIA_FAST_FORWARD"
91 -->  "KEYCODE_MUTE"
92 -->  "KEYCODE_PAGE_UP"
93 -->  "KEYCODE_PAGE_DOWN"
94 -->  "KEYCODE_PICTSYMBOLS"
...
122 -->  "KEYCODE_MOVE_HOME"
123 -->  "KEYCODE_MOVE_END"
```

Annexe A

Codage des caractères

A.1 Codage iso-latin1

La conversion d'une variable de type `char` en une autre de type `int` (ou le contraire) permet d'associer à chaque caractère un nombre unique. Par exemple le 'A' est codé par le nombre 65. On donne ci-dessous un aperçu de la correspondance qui est la table `iso-latin-1`.
Une table détaillée est disponible ici : http://jacques-andre.fr/fontex/latin.pdf

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	´	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~			
160	¡	¢	£	¤	¥	¦	§	¨	©	
170	ª	«	¬		®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Ê	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	i	í	î	ï
240	ð	ñ	o	ó	ô	õ	ö	÷	ø	ù

A.2 Codages des touches spéciales

La lecture des touches enfoncées au clavier se fait grâce à la méthode `keyPressed()` (voir page 47). La variable `key` donne alors le caractère de la touche appuyée. Certaines touches

(flèches, entrée, ...) ne sont pas des caractères. On les retrouve grâce à la variable `keyCode`. Voici quelques codes de touches spéciales :

touche	code	touche	code
fl. gauche	37	entrée	10
fl. haut	38	shift	16
fl. droite	39	ctrl	17
fl. bas	40	tab	9
alt	18	suppr	127
Pg Up	33	Pg Dwn	34

Un moyen simple de les retrouver est d'écrire le sketch suivant :

```
1  void setup() {
2    size(400,300);
3    textSize(100);
4    fill(0);
5  }
6
7  void draw() {
8    background(255);
9    text(key + " : " + keyCode,0,150);
10 }
```

Annexe B

Bibliographie - Sitographie

Quelques livres qui m'ont été utiles pour la rédaction de ce document :

- Gimp 2.8 - collection accès libre par Dimitri ROBERT chez Eyrolles ;
- Premiers pas en CSS3 & HTML5 - collection accès libre par Francis DRAILLARD chez Eyrolles ;
- Informatique et Sciences du numérique - dirigé par Gilles DOWEK chez Eyrolles (disponible gratuitement en pdf : https://wiki.inria.fr/sciencinfolycee/Informatique_et_Sciences_du_Numérique_-_Spécialité_ISN_en_Terminale_S) ;

Je ne peux pas citer tous les sites consultés mais les principaux :

- le site de Processing : http://processing.org ;
- le site flossmanuals : http://fr.flossmanuals.net/processing/introduction/ ;
- le site Arts numériques : http://arts-numeriques.codedrops.net/-Code-Processing- ;

Index