TP N° 3 Multimédia avec processing

Processing est un environnement multimédia, principalement artistique. Il a été créé pour simplifier la création de documents et d'applications multimédia via la programmation informatique.
Il permet d'intégrer des images et du son et de réaliser des animations 2D et 3D, tout en interagissant avec l'utilisateur.
C'est un logiciel libre, gratuit et multi-platforme. Créé au MIT pour enseigner la programmation visuel, puis utilisé par les designers.
La dernière version stable est la 3.5.4, vous pouvez télécharger ici https://processing.org/download/
Il n'y a rien à installer, vous trouverez un exécutable à lancer directement.

L'espace de visualisation :

Une fois lancé, vous disposez d'un espace de visualisation. Il suffit d'appuyer sur le bouton *exécuter* pour l'afficher.
La taille par défaut de la fenêtre est 100 pixels / 100 pixels. Vous pouvez la modifier avec l'instruction : size (largeur, hauteur);
Il faut noter que le point (0,0) est le coin à gauche en haut de la fenêtre.
La largeur représente l'axe horizontal et la hauteur le vertical. Cette instruction peut prendre aussi un 3ème paramètre lorsque le dessin est en 3D.
Vous pouvez changer la couleur de l'arrière plan avec l'instruction *background()*. Elle prend en paramètres une valeur (niveau de gris) ou 3 valeurs (combinaison des 3 couleurs).

Le texte :

On affiche du texte avec l'instruction : text(String chaine, x, y)
Le format est défini avec une variable PFont, comme suit :

```
PFont  f ;
f = createFont("arial", 16, true) ;
textFont(f) ;
```

Les primitives de dessin :

Il existe quelques formes prédéfinies dans processing. Par exemple :

Primitive	Paramètres
point(x,y)	Coordonnées du point
line(xA,yA,xB,yB)	Points de début et de fin
rect(x,y,largeur,hauteur)	Coordonnées du coin supérieur gauche et les dimensions
triangle(x1,y1,x2,y2,x3,y3)	Les trois sommets
quad(x1,y1,x2,y2,x3,y3,x4,y4)	Les quatre sommets
ellipse(x,y, diamètreX, diamètreY)	Le centre et les diamètres
arc(x, y, largeur, hauteur, début, fin)	Le centre et les coordonnées d'une ellipse + angle de début et de fin de l'arc
bezier(x1,y1,x2,y2,x3,y3,x4,y4)	Points de contrôle de la courbe de Bézier

On peut aussi utiliser l'instruction : beginShape() ;vertex(x1,y1) ;... vertex(xN,yN) ;endShape() ;

La couleur :

Les instructions fill() et noFill() agissent sur le remplissage. fill() peut prendre un seul paramètre (niveau de gris), 3 paramètres (combinaison des 3 couleurs) ou 4 en incluant le degré de transparence. Avec noFill(), seul le contour serait visible car le remplissage est éliminé.

Les instructions stroke() et noStroke() agissent sur le contour de la même manière que les deux instructions précédentes.

Exercice 1 :

- Exécuter le code suivant:

```
size(500,300) ;
background(255) ;

// Afficher un texte en utilisant le font par défaut
fill(127) ;
text( "Processing: fenêtre de visualisation", 10, 50 );

// Dessin de primitives
fill(255,0,0);
rect(10, 150, 70,70);

fill(255,255,0);
triangle(215,150,180,220,250,220);

fill(0,255,0);
ellipse(385,185,70,70);

// Dessiner une courbe
noFill();
stroke(0,0,250); //, 300, 90, 400, 150
bezier(10, 100, 150, 50, 300, 150 ,490, 100);
```

- Modifier le code afin de changer la police d'écriture, la taille ainsi que la couleur du texte, la couleur de l'arrière plan et des objets dessinés.

Programmation itérative :

Pour rendre le programme interactif, *processing* utilise les deux fonctions *setup()* et *draw()*.

- setup() est appélée une seule fois au début de l'exécution. On y met les instructions d'initialisation de la fenêtre (par exemple sa taille et sa couleur).
- draw() est une boucle infinie, appelée en permanence par la machine pour réactualiser le dessin. La fonction draw() est appelée en boucle 30 fois par seconde. Cette fréquence peut être modifiée avec l'instruction *frameRate(nbr)*, avec nbr est le nombre d'appels souhaité. La variable *frameCount* quand à elle, contient le nombre d'appels effectués de *draw()*.

L'instruction *saveFrame()* possède comme paramètre le nom de l'image enregistrée suivi de l'extension. Lorsque la fonction *draw()* est utilisée, le nom peut contenir une série de caractères

afin d'énumérer les images (`filename-####.ext`) .L'extension peut être :"tif", "tga", "jpg" ou "png".

Exercice 2 :

- Reprendre l'exercice 1 en utilisant les fonctions *setup* et *draw* (mettre les deux premières instructions dans la fonction setup et le dessin dans la fonction draw).
- Remplacer les dimensions du carré dans l'instruction rect, par *random(70)*.
- Faire la même chose avec le rayon du cercle.
- Proposer une animation pour le triangle.

Charger une image :

Processing permet d'importer des images dans une matrice, où chaque pixel est représenté par sa position et sa valeur. Les images peuvent être d'extension PNG, JPEG ou GIF.

La variable qui va contenir les pixel est de type *Pimage*. Le code suivant permet de charger et d'afficher l'image « test.png » à l'emplacement (10,10) :

```
PImage img;
img = loadImage("test.png");
image(img,10,10);
```

Il est possible de changer la taille de l'image en ajoutant deux paramètres {largeur, hauteur} à la dernière instruction.

```
image(img,10,10,500,500);
```

L'image sera affichée dans un cadre de dimension 500 pixels / 500 pixels.

Accès aux pixels :

Avant de manipuler les pixels, on doit appeler la fonction *loadPixels()* et à la fin du traitement, on appelle la fonction *updatePixels()*.
L'accès à un pixel se fait par sa position pos, telle que pos = x + y * width. x et y sont les coordonnées du pixel et width, la largeur de l'image.
Lorsqu'on souhaite accéder à une image donnée (et non pas à toute la fenêtre de visualisation), on utilise les deux instructions *loadPixels()* et *updatePixels()* comme suit :

```
PImage img ;
img = createImage(500,500);
img.loadPixels();
        // traitement
img.updatePixels();
```

Exercice 3 :

- Exécuter l'exemple suivant:

```
size(200, 200);
loadPixels();
for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
                int pos = x + y * width;
                pixels[pos] = color(255,0,0);
                }
        }
updatePixels();
```

– Modifier le programme afin qu'il affiche une couleur aléatoire pour chaque pixel en programmation itérative.

Opérations sur les images :

Afin de modifier l'apparence de l'affichage, on peut appliquer un filtre dessus en utilisant la fonction filter(type). Il existe dans processing 7 filtres prédéfinis :

filtre	paramètres
THRESHOLD	Un seuil entre 0 et 1
GRAY	-
OPAQUE	-
INVERT	-
POSTERIZE	Un nombre de couleurs entre 2 et 255
BLUR	Un facteur (+ il est grand + l'effet est visible)
ERRODE	-
DILATE	-

Exercice 4 :

– Exécuter l'exemple suivant:

```
size(500,200) ;
PImage img, img2;
img = loadImage("test.png");
img2 = loadImage("test2.png");
img.filter(BLUR,7) ;
image(img,0,0,250,200);
img2.filter(THROSHOLD, 0.5) ;
image(img2,250,0,250,200);
saveFrame("filtre.png");
```

– Tester quelques filtres et les afficher à coté de l'image originale.

– Charger une image et effectuer son inversion manuellement (en accédant aux pixels). Mettre le résultat dans une nouvelle image créée de la même taille que l'originale.