Séance Processing n°4 : Animation

I./ Manipulation d'images bitmap :

Processing permet de manipuler des images bitmap. Il reconnait quatre types de format : .gif, .jpeg, .tga et .png. Afin d'utiliser une image facilement, on peut l'ajouter au programme (sketch) en cours. Il y a deux manières de procéder :

- On sélectionne « Sketch » dans le menu, puis « Ajouter un fichier… » (ou « Add File… »). On sélectionne ensuite le fichier à ajouter à notre programme. On peut vérifier que cet ajout a été fait en faisant : « Sketch » puis « Afficher le dossier » (ou « Show Sketch Folder » ou CTRL-K). Cette commande ouvre le dossier où se trouve notre programme. On peut remarquer qu'un dossier « **`data`** » y a été créé. Ce dossier contiendra tous les fichiers que l'on va rajouter au programme.
- On peut aussi simplement faire glisser un fichier dans la fenêtre où se trouve le code source : cela l'ajoute automatiquement au dossier « **`data`** ».

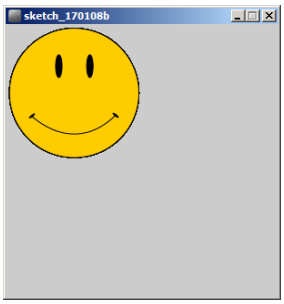
Premier exemple : taper le code suivant :

```
PImage Dessin;

void setup() {
  size(300, 300);
  Dessin = loadImage("smiley.png");
}

void draw() {
  image(Dessin, 0, 0);
}
```

Avant de lancer ce programme, ajouter le fichier « **`smiley.png`** » que vous trouverez dans le répertoire « Groupes (S:) / votre classe / classe / donnees / ICN Peyrot / Images / ». Si vous n'avez pas fait d'erreur, vous devriez avoir ceci :



Explications :

- La première ligne du programme crée une variable dont le nom est « **`Dessin`** » et de type « **`PImage`** » (Attention aux majuscules et aux minuscules). Une variable de ce type peut contenir une image de type bitmap.
- Dans la procédure « **`setup`** », l'instruction « **`loadImage`** » va charger l'image « **`smiley.png`** » pour la placer dans la variable « **`Dessin`** ».
- Dans la procédure « **`draw`** », l'instruction « **`image`** » va afficher l'image contenue dans la variable « **`Dessin`** » aux coordonnées (0,0) de la fenêtre graphique.

Travail à faire :

- Modifier les deux derniers paramètres de l'instruction « **`image`** ». Que remarque-t-on ? A quoi doit-on faire attention ? (On pourra aussi modifier les paramètres de l'instruction « **`size`** » si on veut plus ou moins de place).
- Charger d'autres images (qu'on pourra aller éventuellement chercher sur Internet) et les afficher à des endroits précis de la fenêtre graphique.

Deuxième exemple : taper le code suivant :

```
PImage Fond;
PImage Portion;

void setup() {
  size(300, 300);
  Fond = loadImage("paysage_martien.jpg");
}

void draw() {
  image(Fond, 0, 0);
  // Portion = get(150, 150, 50, 50);
  // image(Portion, 0, 0);
}
```

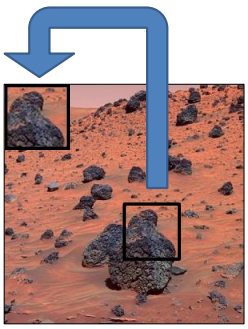
Ajouter l'image « **paysage_martien.jpg** » à ce programme, puis le lancer. On s'aperçoit que l'image a exactement la taille de la fenêtre graphique et va donc entièrement l'occuper.

Enlever ensuite les caractères « **//** » devant les deux dernières instructions. (A quoi servent-ils, déjà ?). Relancer le programme. Que remarque-t-on de différent ?

Explications :

- L'instruction « **get** » de la procédure « **draw** » va récupérer une zone rectangulaire de 50 pixels de large et 50 pixels de haut, située aux coordonnées (150,150) de la fenêtre graphique, et placer l'image ainsi capturée dans la variable « **Portion** » et de type « **PImage** ». Cette image est donc une partie de l'image « **paysage_martien.jpg** ».
- L'instruction « **image** » qui suit va ensuite afficher l'image contenue dans la variable « **Portion** » aux coordonnées (0,0) de la fenêtre graphique.

Cela revient donc à copier une portion de l'image et à la coller ailleurs :



Travail à faire :

- Modifier les paramètres de l'instruction « **get** » pour capturer une zone plus ou moins grande à n'importe quel endroit de la grande image « **paysage_martien.jpg** ». La zone capturée ne doit pas forcément être carrée.
- Charger une image différente (qu'on pourra aller éventuellement chercher sur Internet) et essayer de capturer plusieurs zones et de les afficher à des endroits différents.

Pour aller plus loin avec les images, vous pouvez suivre le lien suivant :

https://fr.flossmanuals.net/processing/les-images/

II./ Animer une image :

Nous avons maintenant tout ce qu'il faut pour créer une animation. Il suffit d'afficher une petite image (on parle de *sprite*) à une position de la fenêtre graphique, puis changer les coordonnées de ce sprite et l'afficher à ces nouvelles coordonnées et ainsi de suite.

Premier exemple : taper le code suivant :

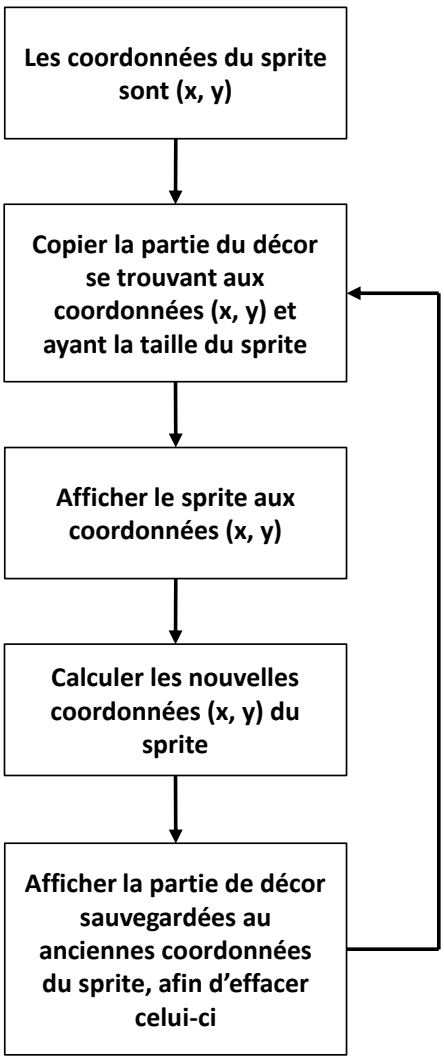
```
PImage Paysage;
PImage Balle;
int x, y, dx, dy;

void setup() {
  size (320, 200);
  Paysage = loadImage("paysage.jpg");
  Balle = loadImage("balle.png");
  image(Paysage, 0, 0);
  x = 0;
  y = 0;
  dx = 1;
  dy = 1;
}

void draw() {
  image(Balle, x, y);
  x = x + dx;
  y = y + dy;
}
```

Après y avoir ajouté les fichiers « **paysage.jpg** » et « **balle.png** », lancer ce programme. Que remarque-t-on ?

Le problème vient du fait que si on veut faire se déplacer un objet sur un décor sans que ce dernier soit affecté, il faut utiliser la technique suivante :



Deuxième exemple : taper le code suivant (ou modifier le programme précédent pour obtenir le code suivant) :

```
PImage Paysage;
PImage Balle;
PImage Fondballe;
int x, y, oldx, oldy, dx, dy;

void setup() {
  size (320, 200);
  Paysage = loadImage("paysage.jpg");
  Balle = loadImage("balle.png");
  image(Paysage, 0, 0);
  x = 0;
  y = 0;
  oldx = 0;
  oldy = 0;
  dx = 1;
  dy = 1;
  Fondballe = get(x, y, 20, 20);
}

void draw() {
  image(Fondballe, oldx, oldy);
  oldx = x;
  oldy = y;
  Fondballe = get(x, y, 20, 20);
  image(Balle, x, y);
  x = x + dx;
  y = y + dy;
}
```

Après y avoir ajouté les fichiers « **paysage.jpg** » et « **balle.png** », lancer ce programme. Quelle est la principale différence de comportement entre ce code et le précédent ?

Remarque : si les deux derniers arguments de l'instruction « **get** » sont 20 et 20, c'est parce que l'image « **balle.png** » fait exactement 20 pixels de large sur 20 pixels de haut. Il faut donc sauvegarder une partie du fond de la même taille à l'endroit où on veut afficher la balle.

Travail à faire :

- Changer les valeurs données à **x**, **y**, **oldx** et **oldy** dans la procédure « **setup** ». Que se passe-t-il ?
- Changer les valeurs de **dx** et **dy** dans la procédure « **setup** » (on peut même essayer des valeurs négatives). Que se passe-t-il ?

Que se passe-t-il lorsque la balle atteint le bord inférieur de la fenêtre graphique ?

Troisième exemple : comment peut-on gérer les collisions ?

Si on veut garder la balle à l'intérieur de la fenêtre graphique, il faut la faire rebondir sur les bords de cette dernière. Par exemple, si elle touche le bord du bas ou le bord du haut, il faut que son sens de déplacement vertical change, ce qu'on peut faire en changeant le signe de **dy**. De la même façon, lorsqu'elle touche le bord de gauche ou le bord de droite, il faut que son sens de déplacement horizontal change, ce qu'on peut faire en changeant le signe de **dx**.

Pour tester si la balle a touché ou non un des bords, il suffit de comparer ses coordonnées avec celles des bords de la fenêtre.

Taper le code suivant puis lancer le, après y avoir ajouté les fichiers « **paysage.jpg** » et « **balle.png** ». (Pour gagner du temps, vous n'êtes pas obligé de taper les lignes de commentaires. Cela dit, il peut être judicieux de le faire, car cela vous permettra plus facilement de voir ce que fait le code si vous le relisez dans quelques temps).

```
// Création des variables
PImage Paysage;
PImage Balle;
PImage Fondballe;
int x, y, oldx, oldy, dx, dy;

void setup() {
  // Définition de la fenêtre graphique
  size (320, 200);
  // Lecture des images
  Paysage = loadImage("paysage.jpg");
  Balle = loadImage("balle.png");
  // Affichage du décor
  image(Paysage,0,0);
  // Initialisation des variables
  // x et y correspondent aux coordonnées de la balle à un instant t
  x = 0;
  y = 0;
  // oldx et oldy sont les coordonnées précédentes de la balle
  oldx = 0;
  oldy = 0;
  // dx et dy sont les déplacement de la balle selon x et selon y
  dx = 1;
  dy = 1;
  // Fondballe est la portion de décor qui se trouve derrière la balle
  Fondballe = get(x, y, 20, 20);
}

void draw() {
  // On replace le fond à l'ancien emplacement de la balle
  image(Fondballe, oldx, oldy);
  // Puis on enregistre les nouvelles coordonnées de la balle
  oldx = x;
  oldy = y;
  // On copie la partie du fond se trouvant derrière la balle
  Fondballe = get(x, y, 20, 20);
  // Puis on affiche la balle
  image(Balle, x, y);
  // On calcule ses nouvelles coordonnées : elle avance horizontalement de dx,
  // et elle avance verticalement de dy
  x = x + dx;
  y = y + dy;
  // Si la coordonnée x est hors de la fenêtre, alors on change de sens selon x
  // "width" contient la largeur de la fenêtre graphique
  if ((x >= width - 20) || (x <= 0)) {
    dx = -dx;
  }
  // Si la coordonnée y est hors de la fenêtre, alors on change de sens selon y
  // "height" contient la hauteur de la fenêtre graphique
  if ((y >= height - 20) || (y <= 0)) {
    dy = -dy;
  }
}
```

Remarque : l'intérêt d'utiliser les mots-clefs « **`width`** » et « **`height`** » est qu'on n'a pas à modifier de façon trop profonde le programme si on décide de changer la taille de la fenêtre graphique.

Travail à faire :

- Modifier la taille de la fenêtre graphique (instruction « **`size`** »). Que remarque-t-on ? Pourquoi est-il plus intéressant d'éviter de prendre une fenêtre graphique carrée ?
- Comment peut-on accélérer le mouvement de la balle ?
- Modifier le programme en changeant l'image de fond ainsi que l'aspect du sprite (de la balle).

III./ Un premier jeu simple :

Nous avons maintenant tout ce qu'il faut pour créer un petit jeu simple. Le but étant de comprendre un minimum ce que l'on est en train de faire, je vous conseille avant de poursuivre de bien reprendre dans l'ordre les chapitres précédents. Si vous vous sentez prêt, vous pouvez taper le code suivant, puis le lancer, après y avoir ajouté les fichiers « **`balle2.png`** » et « **`raquette.png`** ». (Pour gagner du temps, vous n'êtes pas obligé de taper les lignes de commentaires. Cela dit, il peut être judicieux de le faire, car cela vous permettra plus facilement de voir ce que fait le code si vous le relisez dans quelques temps).

```
// Squash v1.0

// Déclaration des variables
PImage balle, raquette, fondballe, fondraquette;
int x, y, dx, dy ,oldx, oldy;
int xraquette, yraquette, oldxraquette, oldyraquette;

void setup() {
  // Initialisations
  oldx = 0;
  oldy = 0;
  x = 0;
  y = 0;
  dx = 2;
  dy = 2;
  oldxraquette = 300;
  oldyraquette = 450;
  xraquette = 300;
  yraquette = 450;
  balle = loadImage("balle2.png");
  raquette = loadImage("raquette.png");
  size(700,500);
  fondballe = get(x, y, 20, 20);
  fondraquette = get(xraquette, yraquette, 80, 20);
}

void draw() {
  // Efface les positions précédentes de la balle et de la raquette
  image(fondballe, oldx, oldy);
  image(fondraquette, oldxraquette, oldyraquette);
  // récupère le fond des nouvelles positions de la balle et de la raquette
  oldx = x;
  oldy = y;
  oldxraquette = xraquette;
  oldyraquette = yraquette;
  fondballe = get(x, y, 20, 20);
  fondraquette = get(xraquette, yraquette, 80, 20);
  // Affiche la balle et la raquette
  image(balle, x, y);
  image(raquette, xraquette, yraquette);
  // Déplacement de la balle
  x = x + dx;
  y = y + dy;
  // Rebond éventuel de la balle sur le haut du terrain
  if (y <= 0) {
    dy = -dy;
  }
  // Rebond éventuel de la balle sur la gauche ou la droite du terrain
  if ((x < 0 ) || (x > width - 20)) {
    dx = -dx;
  }
  // Déplacement de la raquette
  if (keyPressed) {
    if (key == CODED) {
      // On se déplace avec les touches flèche vers la gauche et flèche vers la droite
      if ((keyCode == LEFT) && (xraquette > 0)) {
        xraquette = xraquette - 2;
      }
      if ((keyCode == RIGHT) && (xraquette < width - 80)) {
        xraquette = xraquette + 2;
      }
    }
  }
  // La raquette touche la balle ?
  if ((y + 20 == yraquette) && (x >= xraquette) && (x + 20 <= xraquette + 80)) {
    // Si oui la balle rebondit
    dy = -dy;
  }
  // On perd la balle ?
  if (y > yraquette) {
    // On remet la balle en haut
    x = 0;
    y = 0;
  }
}
```

Que faire ensuite :

- Ce jeu de squash est vraiment très basique graphiquement parlant. A l'aide de ce qu'on a vu dans les activités précédentes, rajouter un joli fond, puis changer les sprites de la raquette et de la balle. On pourra récupérer des images sur Internet ou les créer soi-même avec un logiciel de dessin. Attention : si la taille de vos sprites n'est pas la même que ceux qui sont utilisés dans le programme précédent, peut-être faudra-t-il faire des modifications.
- Rajouter un compteur de score (par exemple chaque renvoi de la balle rapporte un point et le score repart à zéro si on perd la balle).
- On peut rajouter une deuxième raquette de l'autre côté du terrain : c'est le jeu de Pong.
- Si on programme un Pong, on a deux possibilités : soit on joue contre un deuxième joueur humain, soit on joue contre l'ordinateur. Comment faire pour que l'ordinateur renvoie la balle ? (Attention : si on programme correctement le comportement de l'ordinateur, il est possible de faire un Pong imbattable. Comment rendre l'ordinateur « faillible » pour que le jeu soit intéressant ?).
- Peut-on modifier les rebonds sur la raquette, de façon à varier un peu le jeu ?

Complément :

Dans certains cas, il peut être utile de tester plusieurs touches à la fois, par exemple dans le cas d'un jeu à 2 joueurs comme le Pong. Or les fonctions que nous avons déjà vues (**`keyPressed`**, **`key`**) ne le permettent pas. Voici donc des procédures permettant de le faire :

```
void keyPressed(){
  touches[keyCode]=true;
}

void keyReleased(){
  touches[keyCode]=false;
}

boolean verifieClavier(char c){
  int cc = int(c);
  if(cc > 96 && cc < 123) {
    cc-=32;
  }
  return touches[cc];
}

boolean verifieClavier(int c){
  return touches[c];
}
```

Ces procédures sont à placer dans votre code source (avant ou après les procédures « **`setup`** » et « **`draw`** »). Pour qu'elles puissent fonctionner, il faut placer, au tout début de votre code source, la ligne suivante :

```
boolean[] touches = new boolean[128];
```

Lorsque vous avez besoin de tester une touche, vous pouvez utiliser la méthode suivante :

```
if (verifieClavier('p')) {
  // Ce bloc d'instructions sera exécuté si la touche 'p' est appuyée, indépendamment
  // des autres touches
  .........
}

If(verifieClavier('a')&& verifieClavier('b')) {
  // Ce bloc d'instructions sera exécuté si les touche 'a' et 'b' sont appuyées, indépendamment
  // des autres touches
  .........
}
```

Cette méthode n'est pas sensible à la casse, ce qui veut dire que tester 'a' revient à tester 'A'

Attention : cette méthode ne marche pas pour les touches spéciales comme les touches fléchées par exemple.