Séance Processing n°3 : Tests et E/S

I./ Les procédures setup et draw :

Jusqu'ici, nous avons créé des programmes simples, comportant un certain nombre d'instructions qui étaient exécutées *séquentiellement* (l'une après l'autre) jusqu'à la fin du programme. Le programme ne faisait alors plus rien. Il existe une méthode, dans Processing, pour que le programme exécute en boucle certaines instructions, et ce jusqu'à ce que l'utilisateur le stoppe. Pour cela on va utiliser deux *procédures* : la procédure « **setup** » et la procédure « **draw** ». La structure du programme Processing sera alors :

```
// Les instructions continues dans le bloc 'setup' sont exécutées une seule fois, au lancement du
// programme.

void setup() {
  instruction 1;
  instruction 2;
  ............
  instruction n;
}

// Les instructions continues dans le bloc 'draw' sont ensuite exécutées en boucle jusqu'à ce que
// l'utilisateur stoppe le programme

void draw() {
  instruction 1;
  instruction 2;
  ............
  instruction n;
}
```

Dans la procédure setup, on mettra par exemple, l'instruction de définition de la fenêtre graphique, de définition des couleurs, d'initialisation des valeurs initiales des variables, etc…

Un exemple :

```
void setup() {
  size(500, 500);
}

void draw() {
  stroke(random(255), random(255), random(255));
  strokeWeight(random(1,5));
  line(random(500), random(500), random(500), random(500));
}
```

Que fait ce programme ?

II./ Les tests :

Jusqu'ici nos programmes ne prenaient aucune décision : ils ne font qu'exécuter les instructions dans l'ordre. Une instruction permet de faire des *tests*, il s'agit de l'instruction « **if** ».

Syntaxe :

```
if (condition) {
  instruction 1;
  instruction 2;
  ..............
  Instruction n ;
// Ces instructions seront exécutées si la condition est vraie
} else {
  instruction 1;
  instruction 2;
  ..............
  Instruction n ;
// Ces instructions seront exécutées si la condition est fausse.
// A noter que le bloc 'else' est facultatif.
}
```

Qu'est-ce qu'une condition au juste ?

- Cela peut être une *comparaison de deux valeurs*, comme par exemple `(a > 5)`. Par exemple, si la valeur de la variable **a** est 18, alors la condition `(a > 5)` est *vraie*. Si par contre **a** est égal à 2, alors la condition est *fausse*.
- Certains *mots-clefs* sont des conditions qui seront vraies ou fausses en fonction de certains évènements (comme par exemple `mousePressed` et `keyPressed` que nous allons voir dans le prochain paragraphe).
- Attention aux symboles utilisés pour les comparaisons :
  - < et > sont inchangés.
  - ≤ et ≥ sont respectivement remplacés par `<=` et `>=`.
  - = et ≠ sont respectivement remplacés par `==` et `!=`.
- On peut combiner les tests avec les *opérateurs* ***OU*** (`||`) et ***ET*** (`&&`) :
  - La condition `(a < 2 || b != 3)` est vraie si a < 2 ***OU*** b ≠ 3.
  - La condition `(a < 2 && b != 3)` est vraie si a < 2 ***ET*** b ≠ 3.
  - Le OU utilisé est un *OU inclusif*, c'est-à-dire c'est l'un OU l'autre OU les deux (contrairement au *OU exclusif*, utilisé par exemple dans les restaurants : fromage OU dessert !).

III./ Les évènements souris :

Lorsque le pointeur de la souris survole la zone de l'écran correspondant à la fenêtre graphique de Processing, deux variables permettent de connaître les coordonnées X et Y du pointeur de la souris. Il s'agit des variables `mouseX` et `mouseY`.

Taper puis exécuter le code suivant :

```
void setup() {
  size(500,500);
}

void draw() {
  point(mouseX, mouseY);
}
```

Que remarque-t-on ?

Pour que les points soient plus visibles, rajouter une instruction `strokeWeight` dans la procédure setup, avec une valeur au choix (par exemple `strokeWeight(5);`).

On remarque que si on déplace rapidement la souris, on aura une suite discontinue de points. Cela est dû au fait que l'exécution du programme n'est pas instantanée. Pour remédier ce problème, il existe deux variables pmouseX et pmouseY qui contiennent les coordonnées précédentes de la souris. On peut donc modifier le programme précédent de la façon suivante :

```
void setup() {
  size(500,500);
}

void draw() {
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```

Que remarque-t-on ?

Une autre variable pratique est `mousePressed` : cette variable est considérée comme étant une condition VRAIE si un bouton quelconque de la souris est pressé, et comme une condition FAUSSE si aucun bouton n'est pressé. A l'aide des informations précédentes, modifier le dernier programme pour que celui-ci ne trace des traits que si on appuie sur un bouton de la souris. Quand vous avez terminé, faites vérifier ce programme par le professeur, puis sauvegardez le sous le nom « ARDOISEV1 ».

Une souris a généralement deux (voire plus) boutons. Comment différencier l'appui sur l'un ou l'autre des boutons ? Grâce à la variable `mouseButton`, dont la valeur vaudra `LEFT` si le bouton de gauche est pressé, `RIGHT` si le bouton de droite est pressé, et `CENTER` si le bouton du centre est pressé (pour les souris à 3 boutons).

Modifier le programme « ARDOISEV1 » pour qu'il trace un trait fin (`strokeWeight(1);`) si on appuie sur le bouton de gauche, et un trait épais (`strokeWeight(5);`) si on appuie sur le bouton de droite de la souris. Quand vous avez terminé, faites vérifier ce programme par le professeur, puis sauvegardez le sous le nom « ARDOISEV2 ».

Deux instructions supplémentaires peuvent s'avérer utile dans la gestion de la souris : `noCursor` et `cursor`, qui permettent respectivement de cacher le pointeur de la souris (par exemple si vous voulez faire une animation et que vous n'avez pas envie que le curseur de la souris viennent cacher une partie de votre dessin) et de modifier l'apparence de votre pointeur de souris. La syntaxe de ces deux instructions est très simple :

```
noCursor();             // cache le pointeur de la souris
cursor(ARROW);          // Le pointeur est une flèche (valeur par défaut)
cursor(CROSS);          // Le pointeur est une croix
cursor(HAND);           // Le pointeur est une main
cursor(WAIT);           // Le pointeur est un sablier
```

Vous pouvez maintenant modifier votre programme d'ardoise magique en mettant le pointeur de souris de votre choix.

Pour plus d'infos sur les évènements souris, allez faire un tour à l'adresse suivante :
https://fr.flossmanuals.net/processing/les-evenements-souris/

IV./ Les évènements clavier :

Une autre façon d'interagir avec l'ordinateur est l'utilisation du clavier. Processing possède bien évidemment des instructions pour gérer les évènements clavier.

De la même façon que `mousePressed` permettait de savoir si un bouton de la souris était pressé, il existe une variable `keyPressed` qui est considérée comme étant une condition VRAIE si une touche quelconque du clavier est pressée, et comme une condition FAUSSE si aucune touche n'est pressée.

Modifier le programme « ARDOISEV2 » pour qu'il efface l'écran lorsqu'on appuie sur une touche. Pour effacer l'écran, il suffit d'utiliser l'instruction `background`, avec la couleur du fond. Si vous n'avez pas modifié la couleur du fond dans votre setup alors l'instruction nécessaire sera `background(200);`
Vous avez alors un véritable programme d'ardoise magique fonctionnel, qu'il n'est pas nécessaire de relancer quand on veut recommencer un dessin. Quand vous avez terminé, faites vérifier ce programme par le professeur, puis sauvegardez le sous le nom « ARDOISEV3 ».

L'utilisation du clavier faite ici est on ne peut plus basique. Or un clavier standard de PC, c'est plus d'une centaine de touches. Pour savoir quelle touche a été appuyée, on peut utiliser la fonction `key`, qui contient le caractère correspondant à la touche pressée. On peut alors utiliser cette fonction dans un test. Voici un exemple :

```
if (keyPressed) {
  if (key == 'R') {
    instruction;        // Cette instruction sera exécutée si une touche est pressée et
                        // si cette touche est 'R'.
  }
}
```

Attention : « R » et « r » ce n'est pas la même chose pour un ordinateur !

Pour plus d'infos sur les évènements clavier, allez faire un tour à l'adresse suivante :
https://fr.flossmanuals.net/processing/les-evenements-clavier/
Vous verrez notamment comment on peut gérer l'appui sur les « touches spéciales » (comme les flèches, SHIFT, ALT, CTRL, etc…).

Modifier le programme « ARDOISEV3 » pour qu'il efface l'écran lorsqu'on appuie sur la barre d'espace (caractère ' '). Vérifier que le programme n'efface pas l'écran lorsqu'on appuie sur une touche différente. Quand vous avez terminé, faites vérifier ce programme par le professeur, puis sauvegardez le sous le nom « ARDOISEV4 ».

Jusqu'ici, le dessin est monochrome et triste. Modifier le programme « ARDOISEV4 » pour qu'il vous permette de choisir la couleur de votre tracé. Le tracé initial sera noir, mais la couleur changera en fonction des touches pressées :

Touche pressée	**n**	**r**	**v**	**b**	**c**	**m**	**j**	**w**
Couleur	**Noir**	**Rouge**	**Vert**	**Bleu**	**Cyan**	**Magenta**	**Jaune**	**Blanc**

Quand vous avez terminé, faites vérifier ce programme par le professeur, puis sauvegardez le sous le nom « ARDOISEV5 ».

Vous avez maintenant un programme fonctionnel qui vous permet de faire des dessins en 8 couleurs, avec deux épaisseurs de traits différentes, et possédant une fonction d'effacement de la fenêtre graphique. Ce n'est pas encore « PAINT », mais on peut voir qu'avec un programme assez modeste (en termes de nombres de lignes de code), on peut déjà faire des choses assez complexes.

Dans la prochaine activité, nous verrons comment enregistrer les images ainsi créées sur le disque dur, comment les recharger en mémoire, les découper et les manipuler.