Séance Processing n°2 : Les boucles

I./ Présentation du problème :

Les plus rapides d'entre vous ont pu réaliser le dernier programme demandé la dernière fois, dont l'énoncé était : « faire un programme permettant de réaliser un quadrillage de 10 carreaux de large sur 5 carreaux de haut. Chaque carreau sera un petit rectangle de 20 pixels sur 10 pixels. ». Une façon de résoudre ce problème est présentée dans le fichier « Quadrillage.pde » que vous trouverez dans votre répertoire ressource (chemin donné par le professeur) et dont le listing est présentée ci-dessous :

```
size(500,500);
line(0,0,200,0);
line(0,10,200,10);
line(0,20,200,20);
line(0,30,200,30);
line(0,40,200,40);
line(0,50,200,50);
line(0,0,0,50);
line(20,0,20,50);
line(40,0,40,50);
line(60,0,60,50);
line(80,0,80,50);
line(100,0,100,50);
line(120,0,120,50);
line(140,0,140,50);
line(160,0,160,50);
line(180,0,180,50);
line(200,0,200,50);
```

Exécutez le programme et vérifiez qu'il réalise bien la consigne demandée.

Deux inconvénients :

- Le programme est long, 18 lignes, car il y a au moins une ligne de code pour chaque ligne tracée (soit 6 lignes horizontales, plus 11 lignes verticales, plus la ligne définissant la taille de la fenêtre graphique).
- Si on change le nombre de lignes, de colonnes, ainsi que la taille des cases du quadrillage, il faut recalculer toutes les coordonnées et refaire le programme complètement.

Si on examine le programme de plus près, on se rend compte qu'il comporte deux parties :

- La première partie trace les six lignes horizontales :

```
line(0,0,200,0);
line(0,10,200,10);
line(0,20,200,20);
line(0,30,200,30);
line(0,40,200,40);
line(0,50,200,50);
```

On se rend rapidement compte que ces six lignes se ressemblent beaucoup : le premier et le troisième paramètres sont les mêmes pour toutes les lignes (respectivement 0 et 200). Le deuxième et le quatrième paramètres sont identiques pour une même ligne, et ils augmentent régulièrement de 0 à 50, par pas de 10.

- La deuxième partie trace les onze lignes verticales :

```
line(0,0,0,50);
line(20,0,20,50);
line(40,0,40,50);
line(60,0,60,50);
line(80,0,80,50);
line(100,0,100,50);
line(120,0,120,50);
line(140,0,140,50);
line(160,0,160,50);
line(180,0,180,50);
line(200,0,200,50);
```

Là aussi, on se rend compte des similitudes entre ces onze lignes : le deuxième et le quatrième paramètres sont les mêmes pour toutes les lignes (respectivement 0 et 50). Le premier et le troisième paramètres sont identiques pour une même ligne, et ils augmentent régulièrement de 0 à 200, par pas de 20.

II./ Notion de boucle :

On peut simplifier chacune de ces deux parties de code en les remplaçant par des *boucles*. On obtient alors le programme « Quadrillage2.pde » :

```
size(500,500);
for(int i=0;i<=5;i=i+1) {
  line(0,i*10,200,i*10);
}
for(int i=0;i<=10;i=i+1) {
  line(i*20,0,i*20,50);
}
```

Lancer le programme. Que remarque-t-on par rapport au programme précédent ?

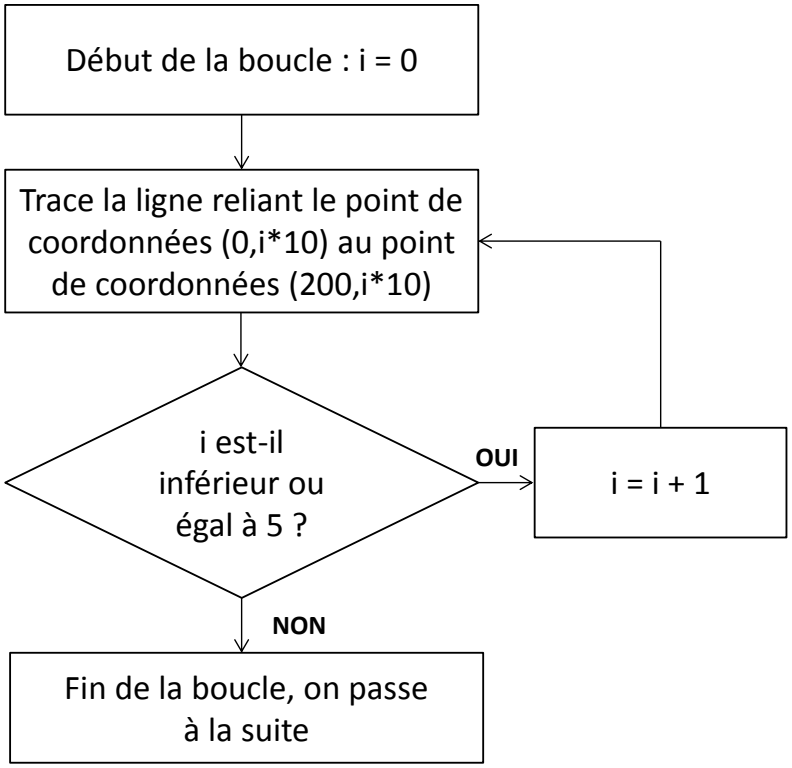
Comment cela marche-t-il ? Une *boucle* est en fait un groupe d'instructions qui vont être exécutées un certain nombre de fois. Pour déterminer combien de fois il faut exécuter ces instructions, on utilise une *variable*. Une variable est un emplacement de la mémoire de l'ordinateur qui peut stocker des valeurs. Pour retrouver une variable en particulier, l'ordinateur lui associe un *nom de variable*. Dans notre exemple, le nom de la variable est « i ». Le *mot clef* « int », de l'anglais « integer » veut dire que la variable « i » ne pourra contenir que des nombres entiers. On dit que la variable « i » est de *type entier*. On verra d'autres types par la suite.

La *syntaxe* de la boucle « for » est la suivante :

```
for(int nom_de_variable=début ; condition pour rester dans la boucle ; incrément) { liste d'instructions }
```

Dans notre exemple, il y a deux boucles : la première va tracer les six lignes horizontales du tableau, la deuxième va tracer les 11 lignes verticales du tableau.

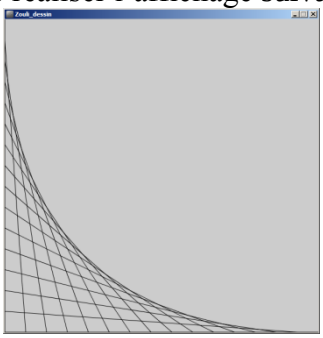
On va décomposer ce qui se passe pour la première boucle, à l'aide de la représentation suivante, appelée « *organigramme* ».



Au début de la boucle, la valeur de « i » est fixée à zéro. L'ordinateur exécute alors les instructions comprises dans la boucle (c'est-à-dire situées entre les deux accolades { et }). Dans notre programme il n'y a qu'une seule instruction : `line(0,i*10,200,i*10);`
Ensuite la valeur de i est augmentée de 1, puis on teste si la valeur de i est inférieure ou égale à 5. Si c'est le cas, on recommence à exécuter les instructions comprises dans la boucle, sinon c'est fini, et on passe à la suite du programme.

Exercices :

- faire un organigramme complet décrivant le fonctionnement de l'intégralité du programme « Quadrillage2.pde ».
- Modifier le programme « Quadrillage2.pde » de façon à ce qu'il trace un quadrillage de 20 colonnes sur 30 lignes, sans changer la taille des cases. Que remarque-t-on ?
- Que faudrait-il faire pour modifier la taille des cases (par exemple pour faire des cases de 5 pixels de haut pour 3 pixels de large) ? Modifier le programme « Quadrillage2.pde » en conséquence.
- Ouvrir le programme « Quadrillage3.pde » et essayer de comprendre son fonctionnement. Quels sont ses avantages par rapport à « Quadrillage2.pde » ?
- Faire un programme permettant de réaliser l'affichage suivant :



III./ Rajoutons un peu de hasard :

Processing peut générer des nombres au hasard (on parle de nombres pseudo-aléatoires) grâce à la fonction « random ».
Si on utilise l'instruction : `line(random(0,500), random(0,500), random(0,500), random(0,500));` l'ordinateur va tracer un segment dont les extrémités auront des coordonnées au hasard, comprises entre 0 et 500 (tester dans un programme pour vous rendre compte de ce que cela veut dire).

Exercices :

- A l'aide d'une boucle, faire un programme traçant 100 segments quelconques (de façon aléatoire, donc) sur une fenêtre graphique dont la taille est laissée à votre choix.
- En utilisant l'instruction « stroke » (voir séquence précédente), modifier votre programme pour que chaque segment ait une couleur différente.
- En vous servant de tous les éléments précédents, réaliser un programme permettant de réaliser un dégradé de couleur :