Projet de Compléments en Programmation Orientée Objet : Ensembles de Julia

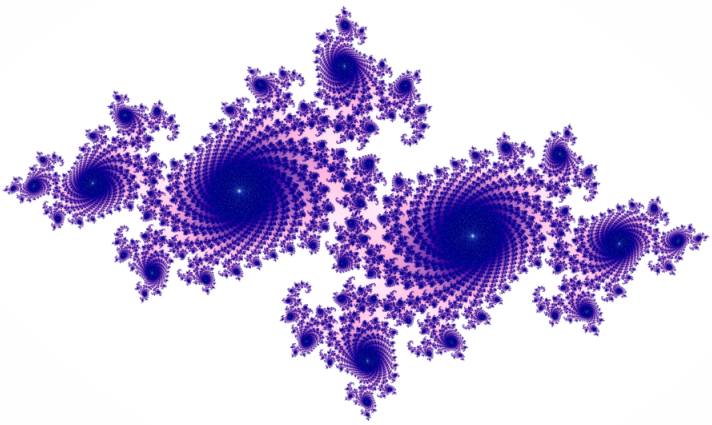

FIGURE 1 – Un ensemble de Julia (source : Wikipedia)

À propos des ensembles de Julia. Les ensembles de Julia [1] sont des ensembles fractals populaires, d'une part par leur côté esthétique et d'autre part par la simplicité de la formule permettant de les calculer.

Ce sont des sous-ensembles du plan complexe [2] $\mathbb{C}$ définis à partir des itérations d'une certaine fonction f de $\mathbb{C}$ dans $\mathbb{C}$. Quand f est un polynôme, son ensemble de Julia est exactement la frontière de l'ensemble des points x_0 tels que la suite $x_{n+1} = f(x_n)$ est bornée [3].

La figure 2 montre un exemple d'algorithme pour dessiner [4] les Julias quadratiques (où f de la forme $f(x) = c + x^2$). Celui-ci associe un entier à tout point du plan complexe ; on peut l'utiliser pour affecter à chaque pixel d'une image une couleur (en interprétant l'entier comme l'indice d'une couleur dans une palette de votre choix). Pour d'autres Julias, il faudra généraliser [5].

Le projet. L'objectif est de programmer un logiciel qui calcule les ensembles de Julia.

— Le programme doit être utilisable aussi bien en mode graphique qu'en mode terminal texte (dans ce dernier cas : mode non-intéractif avec tous les paramètres dans la ligne de commande et export de fichier au format `.png`).

1. Voir `https://fr.wikipedia.org/wiki/Ensemble_de_Julia`.
2. `https://fr.wikipedia.org/wiki/Nombre_Complexe`.
3. En toute généralité, f est prise dans une certaine classe de fonctions holomorphes (peu importe...) ; l'ensemble de Julia associé est alors l'ensemble des points x_0 tels qu'une petite perturbation provoque un changement drastique du comportement de la suite $x_{n+1} = f(x_n)$ (il y a une définition mathématique rigoureuse de ce que cela veut dire). Dans le cas des polynômes, il a été démontré que la caractérisation que nous avons donnée est équivalente.
4. Ce qui est en réalité dessiné, c'est, en une certaine couleur, l'ensemble de Julia "rempli" : l'ensemble des points x_0 telle que la suite ne diverge pas ; ainsi que, en d'autres couleurs, les lignes équipotentielles.
5. C'est-à-dire : passer en paramètre une fonction f. Il faudrait aussi adapter le critère de divergence, mais faute de preuve mathématique générale, on peut continuer à utiliser le même critère que pour les Julias quadratiques : considérer que si un terme x_n vérifie $|x_n| > 2$, alors la suite diverge irrémédiablement.

```java
/**
 * @param x0 : le point de départ de l'itération
 * @param c : le paramètre de la fonction f(x) = x -> c + x*x
 * @param maxIteration : nombre d'itérations à effectuer avant de considérer que la suite est bornée
 * @return : le numéro d'itération à partir duquel on est sûr que la suite n'est pas bornée
 */
public static int divergenceIndex(Complex x0, Complex c, int maxIteration) {
   int ite = 0; Complex xn = x0;
   // sortie de boucle si divergence
   while (ite++ < maxIteration && xn.modulus() <= 2.) xn = c.plus(xn.times(xn));
   return ite;
}
```

FIGURE 2 – Algorithme pour calculer l'ensemble de Julia associé à la fonction $f_c(x) = c + x^2$. Plus précisément : pour un point complexe x_0, cette méthode calcule le numéro d'itération à partir duquel on est sûr que la séquence $x_{n+1} = c + x_n^2$ diverge (l'astuce : si $|x_i| > 2$ alors la suite $(|x_n|)_{n \geqslant i}$ est croissante et divergente).
Ce code est donné en syntaxe Java par commodité de lecture, mais il n'est pas nécessairement à reprendre tel quel dans les projets (adaptation à votre architecture à prévoir).
Remarque : on suppose donnée la classe `Complex`, représentant les nombres complexes munis de leurs opérations principales, dont la somme, le produit et le module : les méthodes `Complex plus(Complex other)`, `Complex times(Complex other)` et `double modulus()`.

- Il doit être possible de choisir les paramètres de l'image sans recompiler le logiciel : il faudra donc prendre en compte les options de la ligne de commande et, dans l'IG, prévoir des contrôles pour modifier les paramètres (zoom, déplacement, ...).
- A minima, on veut pouvoir afficher tous les ensembles de Julia quadratiques ($f_c(x) = c+x^2$). Il faut donc pouvoir spécifier c et les bornes de l'image.
  Il serait bien de pouvoir entrer $f(x)$ en tant que polynôme de degré quelconque. Il faut réfléchir à la façon dont on saisit une telle fonction, à la façon dont on la représente en tant qu'objet Java, et à la façon de la calculer.
- Ajoutez un mode pour calculer l'ensemble de Mandelbrot [6] en réutilisant la même architecture (via héritage ou autre procédé).
- Dans l'IG, on ne veut pas que les contrôles soient figés pendant le calcul. Il faudra donc que celui-ci se fasse dans un *thread* différent de celui de l'IG (i.e. *worker thread*). Par ailleurs, il faut que les contrôles de l'IG puissent interrompre le calcul en cours (notamment en cas de changement des paramètres).
- À essayer : un mode de calcul progressif. Au lieu de tenter d'itérer f un nombre fixe de fois pour chaque pixel, on itère à l'infini tant que ni la divergence, ni la convergence ne sont assurées. Dès que le calcul termine pour un pixel donné, on change sa couleur sans attendre la fin de tous les calculs.
- À explorer : répartir les calculs sur plusieurs *threads* afin d'accélérer l'affichage (de préférence, utiliser un *thread pool* via l'API de votre choix). Instrumentez le code pour enregistrer le temps de calcul. Étudiez l'influence du nombre de *threads* (testez sur des calculs longs de plusieurs secondes, pour que la différence soit significative).

6. `https://fr.wikipedia.org/wiki/Ensemble_de_Mandelbrot`. Attention, ce n'est pas un ensemble de Julia, mais il y a des liens très forts entre les deux notions.