Projet de Compléments en Programmation Orientée Objet : Fractales (version 1.1)

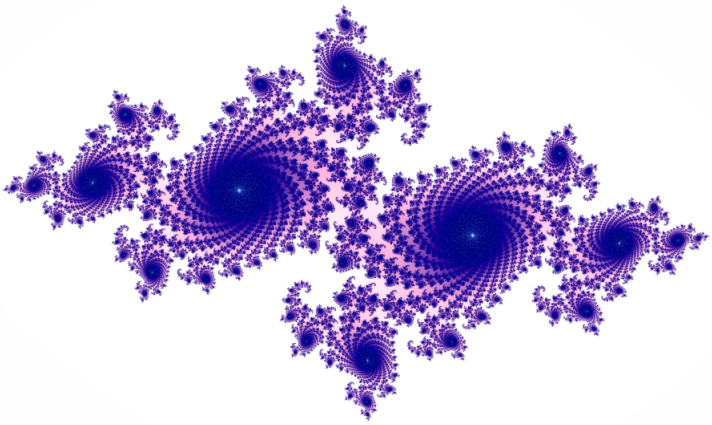

FIGURE 1 – Un ensemble de Julia (source : Wikipedia)

I) Déroulement

Le projet est réalisé en binôme (de préférence) ou en solo. Les inscriptions de binômes sont ouvertes sur Moodle

Vous devez rendre, avant Noël, sur Moodle une archive (.zip ou .tar.gz) contenant les sources Java (+autre fichiers nécessaires pour la compilation) et un fichier texte Readme (.pdf toléré).

Le fichier Readme doit expliquer brièvement (3 pages max, le plus souvent 1 page suffira)

- comment compiler le projet ;
- comment lancer les deux versions du programme (ligne de commande et interface graphique) ;
- les principaux choix techniques que vous considérez comme points forts (algorithmes, bibliothèques, techniques de conception et deprogrammation) ;
- si vous avez exploré la performance des versions parallèles, un petit rapport sur cette exploration ;
- si besoin, références au sources.

Les soutenances auront lieu en janvier 2022. À la soutenance chaque membre du binôme doit montrer la maitrise totale de tout le projet.

L'utilisation des bibliothèques tierces dédiées aux fractales est interdite. Le plagiat sera détecté avec des outils performants et sévèrement puni.

II) Contexte : les fractales

Les fractales sont des objets mathématiques qui sont similaires aux nombreuses formes naturelles, du chou-fleurs au fjords Norvégiens. Renseignez-vous sur Wikipedia, par exemple.

Dans ce projet vous devrez faire un programme qui génère plusieurs type de fractales, en commençant par des ensembles de Julia.

a) 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 $\mathbb{C}$ 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.

En réalité, pour visualiser un ensemble de Julia on procédera comme suit :

— on choisira un polynôme complexe f (dans le premier temps on peut se limiter des quadratiques de la forme $f(z) = c + z^2$ avec une constante c, les valeurs intéressantes de c peuvent être trouvées sur Wikipédia)

— on associera à chaque point du plan complexe son indice de divergence : combien de fois peut-on appliquer f en partant de z avant que la suite commence à diverger. Le pseudocode suit :

```
MAX_ITER=1000; RADIUS=2.;
int divergenceIndex(Complex z0) {
   int ite = 0; Complex zn = z0;
   // sortie de boucle si divergence
   while (ite < MAX_ITER-1 && |zn| <= RADIUS)
       zn = f(zn); ite++;
   return ite;
}
```

Le sens des deux constantes est évident : dès que $|z| > 2$ on décide que ça diverge (et on retourne le nombre d'itérations). Après 1000 itération on décide que ça converge. Bien sûr qu'on peut essayer d'autres valeurs des constantes, mais l'effet sera minime.

— dans un rectangle du plan complexe (dans le premier temps on peut prendre $[-1, 1]\times[-1, 1]$) on prendra des points (nombres complexes) avec un certain pas h (on peut commencer par 0.01). Ceci nous donnera une grille de 201×201 points. Pour chaque tel point z, on calcule son `ind=divergenceIndex(z)`. On colorie le pixel de z selon la valeur de `ind` (il faudra choisir un code couleur). Et on obtient l'image de Julia.

III) Cahier de charge : les deux objectifs du projet

a) Ce qu'on doit programmer

Dans ce projet vous devrez programmer un logiciel qui calcule certains types de fractales.

— 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 du rectangle de l'image, ainsi que le pas de discrétisation. Il serait bien de pouvoir entrer $f(x)$ en tant que polynôme de degré quelconque (à vous de choisir comment). Il serait bien de pouvoir choisir parmi plusieurs visualisations.

— Il est souhaitable pouvoir calculer l'ensemble de Mandelbrot `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. Vous pouvez aussi générer d'autres types de fractales.

1. Voir les articles Wikipedia en français et en anglais

— En mode terminal votre programme doit prendre tous les paramètres dans la ligne de commande (ou dans un fichier de configuration) et exporter le fichier en format `.png`. Il serait bien d'exporter aussi un descriptif textuel de fractale (type, fonction, rectangle etc.)
— En mode graphique le programme doit prendre tous les paramètres via l'interface graphique et afficher une ou plusieures fractales obtenues.
— bien évidemment le choix de paramètres de l'image se fera 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, ...).

b) Les techniques à mettre en oeuvre

La note du projet tiendra compte de la qualité de programmation de Java, et la maitrise de la POO telle qu'étudiée dans les cours de L1 et L2 et approfondi cette année, en particulier

— respect des règles de style (noms des entités, commentaires/javadoc, format de code)
— bonne structuration et clarté de code ;
— architecture objet bien réfléchie de votre logiciel ;
— bon usage de l'encapsulation, l'héritage et le polymorphisme, programmation à l'interface ;
— robustesse du programme, gestion des exceptions ;
— utilisation des bonnes bibliothèques standards et non ;
— maitrise des interfaces graphique ;
— bonne utilisation des arguments de ligne de commande.

Elle tiendra aussi compte de l'utilisation (motivée) de certaines des nouvelles techniques vues en ce cours de compléments de POO :

— fabriques statiques et autres Builders pour instancier vos objets ;
— classes immuables, énumérations, et éventuellement classes scellées ;
— lambda-expréssions pour alléger vos interfaces graphiques ;
— fonctions de première classe et fonctions d'ordre supérieur ;
— streams ;
— programmation concurrente et parallèle(avec les `Thread` ou autres `ForkJoinPool` et `parallel Stream`).

En cas où vous essayerez la programmation parallèle, accompagnez-la d'une petite étude sur le gain de performance obtenu.

IV) Quelques indications

Voici le plan de travail que vous pourriez adapter si vous le souhaitez.

a) Avant de commencer

Révisez si nécessaire les nombres complexes (il vous faut connaitre la définition, la représentation sur le plan complexe, et les opérations $x + y, x \cdot y, |x|$). Prenez un peu de temps pour lire sur les fractales en général et les ensembles de Julia.

b) Préparez les outils

Vous aurez besoin d'une classe pour les nombres complexe, faites-la vous-même (en se basant sur les TP 1 ou 2) ou installez celle de Apache.

Vous devrez aussi créer des images pixel par pixel, et les écrire en .png (conversion en jpeg abimerait le fractale). La classe BufferedImage me semble suffisante, vous l'écrirez avec

```java
var img=new BufferedImage(100, 100, BufferedImage.TYPE_INT_RGB);
int r = 64; int g = 224; int b = 208; //turquoise
int col = (r << 16) | (g << 8) | b;
img.setRGB(30,40,col);
File f = new File("MyFile.png");
ImageIO.write(img, "PNG", f);
```

FIGURE 2 – Ce bout de code Java crée l'image de taille 100×100 codé en RGB, y ajoute un pixel turquoise aux coordonnées $(30, 40)$ et sauvegarde l'image.

ImageIO.write. Essayez ceci, ou cherchez des techniques plus avancées pour travailler avec les images.

Choisissez aussi la technologie pour faire l'interface graphique (Swing ou JavaFX), ainsi que votre environnement de développement.

Installez et configurez tout.

c) Faites le 1er test

Essayez de générer votre premier ensemble de Julia, pour cela

- prenez une fonction complexe comme dans Wikipedia, par exemple $f(z) = z^2 + c$ avec $c = -0.7269 + 0.1889i$
- choisissez un rectangle sur le plan complexe (p.ex. $[-1, 1] \times [-1, 1]$)
- couvrez-le de points avec un certain pas, par exemple 0.01, vous aurez 201×201 points complexes
- pour chaque point calculez son indice de divergence
- représentez ces indices en code couleur et créez l'image couleur de taille 201×201 pixels.
- sauvegardez l'image et admirez-le.

d) Réfléchissez sur la/les classes principales

Sans doute votre programme aura une ou deux classes pour représenter l'ensemble de Julia et son affichage. Cette (ou ces) classes devraient faire à peu près la même chose que votre premier test. Mais elles doivent aussi permettre le choix

1. de la fonction f ou au moins de la constante c si vous vous limitez à $f(z)$;
2. du rectangle de travail sur le plan complexe $\mathbb{C}$
3. du pas de discrétisation de ce rectangle
4. et de la taille de la matrice de pixels
5. du nombre d'itération max dans le calcul de l'indice de divergence
6. de la méthode pour traduire un tel indice en couleur du pixel.
7. éventuellement du fichier pour sauvegarder l'image.

Vous remarquerez que les paramètres 3,4,5 sont liés. Que les items 1 et 6 se prêtent aux attributs fonctions de première classe. Que la grande quantité d'options fait penser à un Builder ou un autre patron similaire.

e) Réfléchissez sur le programme de base (en ligne de commande)

Quelles sont ses fonctionnalités, quels sont les arguments de la ligne de commande ? Quelle est l'architecture de classes de ce programme ? Voulez-vous utiliser Apache Common CLI pour mieux gérer les arguments de la ligne de commande ?

f) Conseils sur le code couleur

On doit représenter l'indice `ind` — un entier entre 0 et `MAX_ITER` par un code couleur. Plusieurs pistes sont envisageables :

- la visualisation la plus simple utilise 2 couleurs consiste à donner une couleur aux points divergents (avec `ind<MAX_ITER`) et une autre aux "convergents" (avec `ind=MAX_ITER`) ;
- dans tout les cas on peut coder `MAX_ITER` qui correspond à la "convergence" par une couleur spéciale (noire ?)
- les autres valeurs peuvent être codées tout simplement par des niveaux d'une couleur : par exemple

```
r= (255*ind)/MAX_ITER; g=0; b=0;
```

- ou bien on peut coder `ind/MAX_ITER` par la teinte (`hue` en anglais) du point en système HSB. Pour convertir en `rgb` on fera quelque-chose comme

```
int rgb=Color.HSBtoRGB((float)ind/MAX_ITER, 0.7f, 0.7f);
```

- j'ai même essayé de coder le 1er digit décimal de `ind` par un niveau du rouge, le 2nd par un niveau du vert, et le 3ème par le niveau du bleu, le résultat est joli mais pas très convaincant.
- Finalement on peut penser à lisser un peu les bords avec anti-crénelge, le risque est de détruire les petits détails fractals. Testez si vous voulez et savez comment.

g) Programmez le programme basique en ligne de commande

N'oubliez pas de documenter la ligne de commande (et afficher le bon format de cette ligne en cas d'erreur). Testez. Vous avez maintenant le premier livrable !

h) Réfléchissez comment adapter votre programme à une interface graphique

Très important : les classes principales des versions ligne de commande et IG doivent être partagés.

Pour l'architecture de l'application IG vous pouvez choisir le patron MVC, ou faire autrement.

i) Programmez le programme basique en IG

N'oubliez pas d'utiliser les lambdas pour les gestionnaires de vos boutons ! N'oubliez pas que les gros calculs doivent se faire en thread séparé, prévoyez la possibilité d'interrompre le calcul. Testez tout. Le second livrable est prêt.

j) Extensions de fonctionnalité

Si pour l'instant vous savez seulement faire le Julia pour $f(z) = z^2 + c$ c'est le moment de passer au cas général. J'insiste que f devrait être un attribut fonction de première classe. Il faudra décider comment saisit-on un polynôme quelconque pour f et comment on le parse.

Il serait bien de proposer plusieurs choix de visualisation (code couleur).

Il serait une très bonne idée d'accompagner chaque fichier `.png` par un petit fichier texte généré qui décrit comment le fichier est-il obtenu (fonction, rectangle, pas, code couleur). Et encore pouvoir charger ce fichier, pour reproduire la même fractale

Par ailleurs, vos programmes devraient générer n'importe quel ensemble de Julia, mais il serait bien d'avoir un petit menu/script avec plusieurs cas concrets que vous trouvez intéressants/jolis.

Sauriez-vous faire le changement de rectangle (zoom, déplacement) dans votre interface graphique ? Est-ce possible sans tout recalculer ?

Ensuite, vous pouvez passer aux autre types de fractales, tels que l'ensemble de Mandelbrot. Réfléchissez sur les aspects orienté objet : serait-il opportun d'avoir une classe abstraite ou interface Fractale dont Julia et Mandelbrot hériteraient.

k) Extensions technologiques

Il serait très intéressant d'essayer la programmation concurrente/parallèle pour le calcul de vos fractales. Ainsi on pourrait lancer le calcul de chaque pixel en parallèle, par exemple avec `ForkJoinPool`. Une autre possibilité serait d'utiliser un `Stream` parallèle. Si vous choisissez cette option, instrumentez votre code pour pouvoir comparer le temps de calculs en séquentiel et en parallèle. Étudiez l'influence du nombre de *threads* (testez sur des calculs longs de plusieurs secondes, pour que la différence soit significative).

l) Finalisation

Faites une revue de code finale, vérifiez que les règles de style sont respectées, ajoutez des derniers commentaires, supprimez les parties de code inutiles ou communes, formatez.

Écrivez un petit fichier texte readme. Évitez des fautes d'orthographe. C'est votre 3ème livrable.