README.md

Fractales

Exploration de Fractales en Java.

Ce projet permet de créer et visualiser les fractales des ensembles Julia et Mandelbrot. Il possède une interface en ligne de commande qui sauvegarde la visualisation de la fractale dans une image et une interface graphique qui permet d'explorer la Fractale.

Documentation

GUI La javadoc est accessible à la racine dans le dossier javadoc. Il est possible de la consulter en ouvrant javadoc/index.html dans un navigateur web.

Sinon, pour construire la javadoc, lancer

```
./gradlew javadoc
```

et ouvrir le fichier html crée dans

```
app/build/docs/javadoc/index.html
```

Compilation et lancement

Ce projet est construit avec Gradle. Un script `run.sh` est mis à dispostion pour facilement lancer le projet sur les systèmes Unix.

`run.sh` compile le programme si ce n'est pas déjà fait et le lance avec les arguments qui lui sont passés. Il suffit donc de faire

```
./run.sh <argument1> <argument2> ...
```

pour compiler et lancer le projet.

Sinon, on peut utiliser `./gradlew build` pour la compilation, et `./gradlew run --args="<argument1> <argument2> ..."` pour lancer le programme.

Utilisation du CLI

Le programme, si l'option `-G` n'est pas saisie, n'a qu'un argument obligatoire `-t` . Celui ci doit indiquer le type de la fractale, "julia" ou "mandelbrot". Toutes les autres options ont des valeurs par défaut.

```
./run.sh -t julia
```

Construisons un example au fûr et à mesure.

Imaginons qu'on souhaite visualiser un ensemble de Julia avec le polynome $f(z) = z^2+c$, où $c = -0.782 + 0.12i$,

```
./run.sh -t julia -c -0.782,0.12
```

Le carré du plan complexe qu'on souhaite visualiser a pour coordonnées (z1, z2), les points en haut à gauche et en bas à droite du carré respectivement. Par exemple, pour (-3 + 3i, 3 + -3i)

```
./run.sh -t julia -c -0.782,0.12 -r -3,3,3,-3
```

Et on veut que l'image produite soit de résolution 3000x3000

```
./run.sh -t julia -c -0.782,0.12 -r -3,3,3,-3 -p 3000
```

Par défaut, les images sont sauvegardées dans `app/<nom_de_lensemble>.png` . Mais un chemin et un nom de fichier peuvent être spécifiés avec l'option `-f` , l'ajout de l'extension .png est automatique.

```
./run.sh -t julia -c -0.782,0.12 -r -3,3,3,-3 -p 3000 -f ~/julia1
```

D'autres options existent comme selectionner un thème de couleur, ou spécifier un zoom dans le carré complexe. Se référer au menu d'aide avec l'option `-? / --help`

Utilisation du GUI

Il est possible d'accéder a l'interface graphique avec l'option `-G` ou `--graphical` . Dans ce cas toutes les autres options sont ignorées, car on peut les renseigner dans l'interface.

Une fois que vous avez généré votre fractale avec le bouton `Generate` , vous pouvez l'explorer. Pour se déplacer sur les axes du plan, utilisez les flèches à diposition. Si vous voulez "zoomer" à un endroit précis, cliquez sur le point de l'image que vous voulez voir et appuyez sur `Zoom +` .

La valeur par défaut de 50 itérations fournit suffisamment de détails pour une vue d'ensemble, mais pas assez une fois qu'on a zoomé relativement loin. Il est recommandé de zoomer là où on veut voir plus de détails, puis monter le nombre d'itérations et re-cliquer sur `Generate` si nécéssaire.

Architecture et points forts

- Nous avons essayé de programmer en suivant les principes SOLID, dans le but de diminuer le plus possible la répétition du code.
- La bibliothèque `math3` de Apache a été utilisée pour les nombres complexes.
- La bibliothèque `cli` de Apache a été utilisée pour l'analyse des options de la ligne de commande.
- Les classes et fonctions sont segmentées. Les calculs des valeurs de divergence ainsi que de la couleur de chaque pixel sont faits en parallèle à l'aide d'un Parallel Stream (donc un thread différent par pixel).
- Utilisation de l'hérédité pour les différentes fractales.
- L'utilisation d'interfaces fonctionelles pour les thèmes de couleurs, ainsi que de fonctions d'ordre supérieur (on passe un thème de couleur, donc une fonction, à la fonction `Renderer.drawFractal`).
- Des fonctions lambda pour les actions des boutons afin d'alléger le code de l'interface graphique.
- Du rendu conditionnel dans l'interface graphique (i.e la visibilité des champs spécifiques a Julia n'apparaissent que si l'ensemble de Julia est sélectionné)

Performance et parallélisation

Nous avons fait une étude de la performance avant et après la parallélisation de la classe Renderer, à l'aide du script `test.sh` et du module `timeit` python. Si l'utilisateur souhaite faire un tour de tests, il peut lancer `./manual_test.sh` et avoir des résultats individuels affichés pour chaque cas.

La commande utilisée était `python3 -m timeit "__import__('os').system('./test.sh')"`

Elle renvoie la moyenne du temps d'éxécution de 5 répétitions de ./test.sh:

Avant parallélisation: ~ 222s en moyenne par batterie de test

Après parallélisation: ~ 92.2s en moyenne par batterie de test

Ce qui répresente une amélioration d'environ 240% du temps d'éxécution.