

Visualiseur de Fractales de Mandelbrot

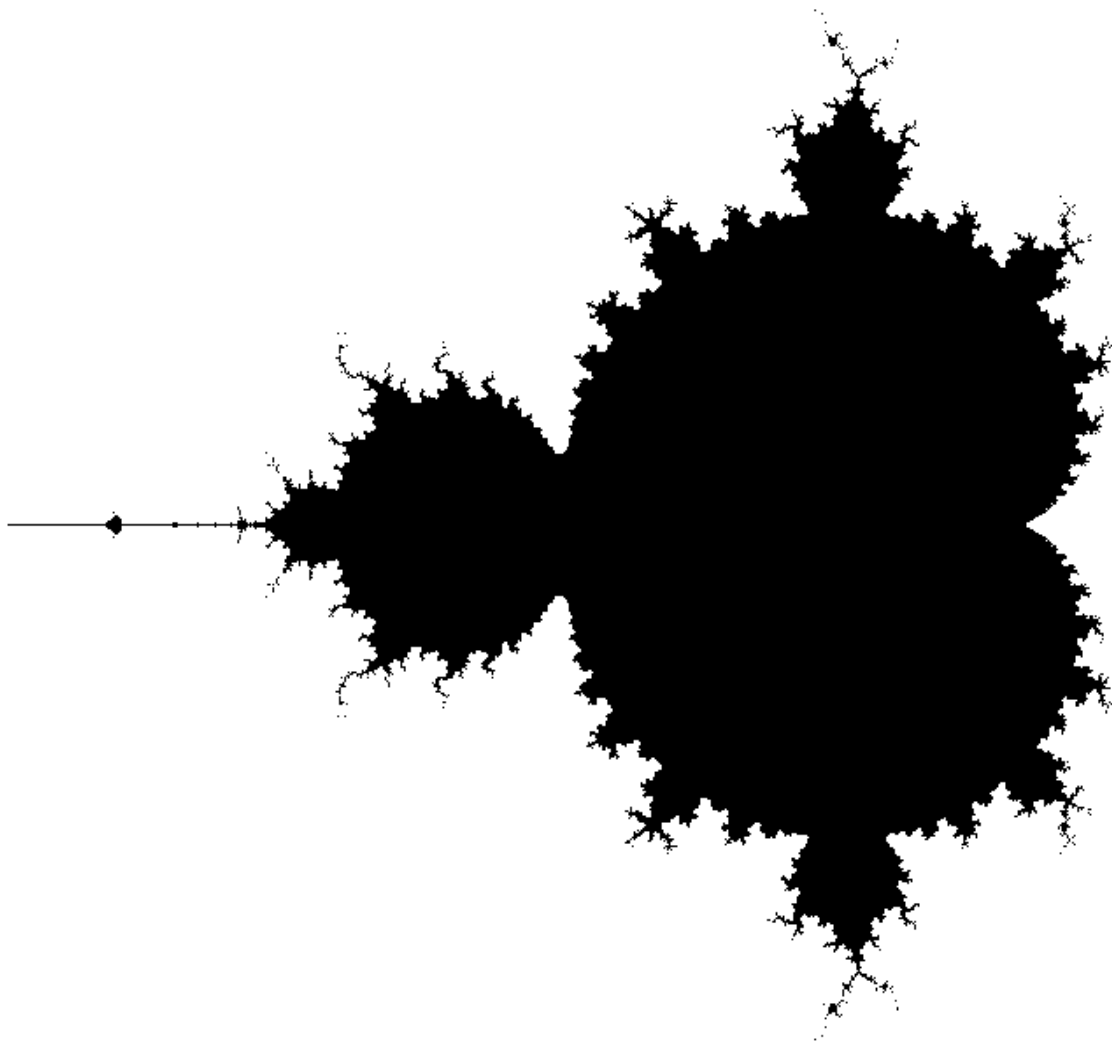


Table des matières

Introduction	3
Présentation du projet.....	3
Documentations	4
Analyse	5
Les classes utilisées.....	5
Diagramme des classes	8
Fonctionnement global.....	8
Les choix techniques.....	9
Utilisation.....	10
Mode d'emploi	10
Configuration requise	12
Conclusion	13
Bilan	13
Optimisations envisageables	13
Extensions possibles.....	13

Introduction

Présentation du projet

Le projet correspond à une application java permettant à l'utilisateur de visualiser une fractale de Mandelbrot, en ayant choisi au préalable le nombre d'itération et la couleur, dans une interface graphique.

La fractale affichée est une image.

L'application possède les fonctionnalités suivantes :

- Dessiner une fractale en noir et blanc ou en couleur
- Choisir le nombre d'itérations
- Personnaliser la couleur de la fractale
- Effectuer un zoom avant ou arrière
- Enregistrer le résultat en tant que fichier png
- Charger une image dans l'interface

Documentations

Les documentations utilisées relatent des algorithmes de calculs de l'ensemble de Mandelbrot et des éléments de programmation des interfaces graphiques du langage Java.

Algorithmes de Mandelbrot :

Ensembles de Mandelbrot de de Julia, *Algorithmique et calcul numérique*, José OUIN

<http://sdz.tdct.org/sdz/dessiner-la-fractale-de-mandelbrot.html>

https://fr.wikipedia.org/wiki/Ensemble_de_Mandelbrot

Programmation Java (Awt, Swing...) :

<https://docs.oracle.com/javase/7/docs/api/overview-summary.html>

Analyse

Les classes utilisées

Le projet utilise trois grandes classes principales avec une classe Main : Mandelbrot, GraphicWindow et DrawArea.

Ces dernières correspondent respectivement au calcul de l'ensemble de Mandelbrot, à la fenêtre graphique et à la zone de dessin.

Mandelbrot
-pts_x: Vector<Integer> -pts_y: Vector<Integer> -pts_out_x: Vector<Integer> -pts_out_y: Vector<Integer> -pts_out_color: Vector<Color>
+Mandelbrot(img_x:int,img_y:int,iteration:int, r:int,g:int,b:int) +calcPts(img_x:int,img_y:int,xmin:double, xmax:double,ymin:double,ymax:double, n:int,r:int,g:int,b:int): void +draw_WB(width:int,height:int): BufferedImage +draw_Color(width:int,height:int): BufferedImage

La classe ci-dessus calcule les points appartenant à l'ensemble et ceux n'y appartenant pas. La couleur de chaque pixel en dehors de l'ensemble est également stockée afin de pouvoir dessiner la fractale en couleur. Les coordonnées des points et les couleurs sont stockés dans des Vectors pour être plus facilement dessinés.

Mandelbrot possède 4 méthodes publiques :

- Deux correspondent à la création d'une BufferedImage (en noir et blanc ou en couleur)
- Les deux autres sont le constructeur de la classe ainsi que le calcul des points et des couleurs (par la méthode de calcul de Mandelbrot). La fonction de calcul est utilisée dans le constructeur de la classe.

GraphicWindow
+drawPane: DrawArea +zoom: double
+GraphicWindow(title:String) +getInputIteration(): int +getInputColor(): int[] +ActionPerformed(event:ActionEvent): void +mousePressed(e:MouseEvent): void +mouseClicked(e:MouseEvent): void +mouseEntered(e:MouseEvent): void +mouseExited(e:MouseEvent): void +mouseReleased(e:MouseEvent): void +mouseWheelMoved(e:MouseWheelEvent): void

Voici la classe la plus importante du projet, elle constitue la fenêtre graphique de l'application. Afin de pouvoir réagir aux événements déclenchés par l'utilisateur, cette dernière implémente les interfaces `ActionListener`, `MouseListener` et `MouseWheelListener`.

`GraphicWindow` possède deux instances : la zone de dessin et la valeur du zoom sur l'image de la fractale.

L'idée pour dessiner une fractale réside dans le fait d'initialiser *drawPane* uniquement au moment de dessiner l'image (que l'on passe en paramètre). L'utilisateur peut dessiner une fractale à partir d'un menu de sélection de la fenêtre.

Du côté des Listeners, on les utilise surtout dans le menu au moment du clic sur les options et pour le zoom avec la molette de la souris.

On est obligé de définir les méthodes de *MouseListener* pour utiliser un *MouseWheelListener*. C'est la méthode *ActionPerformed* qui se charge d'effectuer les actions écoutées précédemment dans le constructeur de la classe, au moment de créer le menu avec ses items.

Les méthodes *getInputIteration()* et *getInputColor()* permettent de récupérer respectivement le nombre d'itération et la couleur choisie par l'utilisateur dans l'interface de saisie du menu.

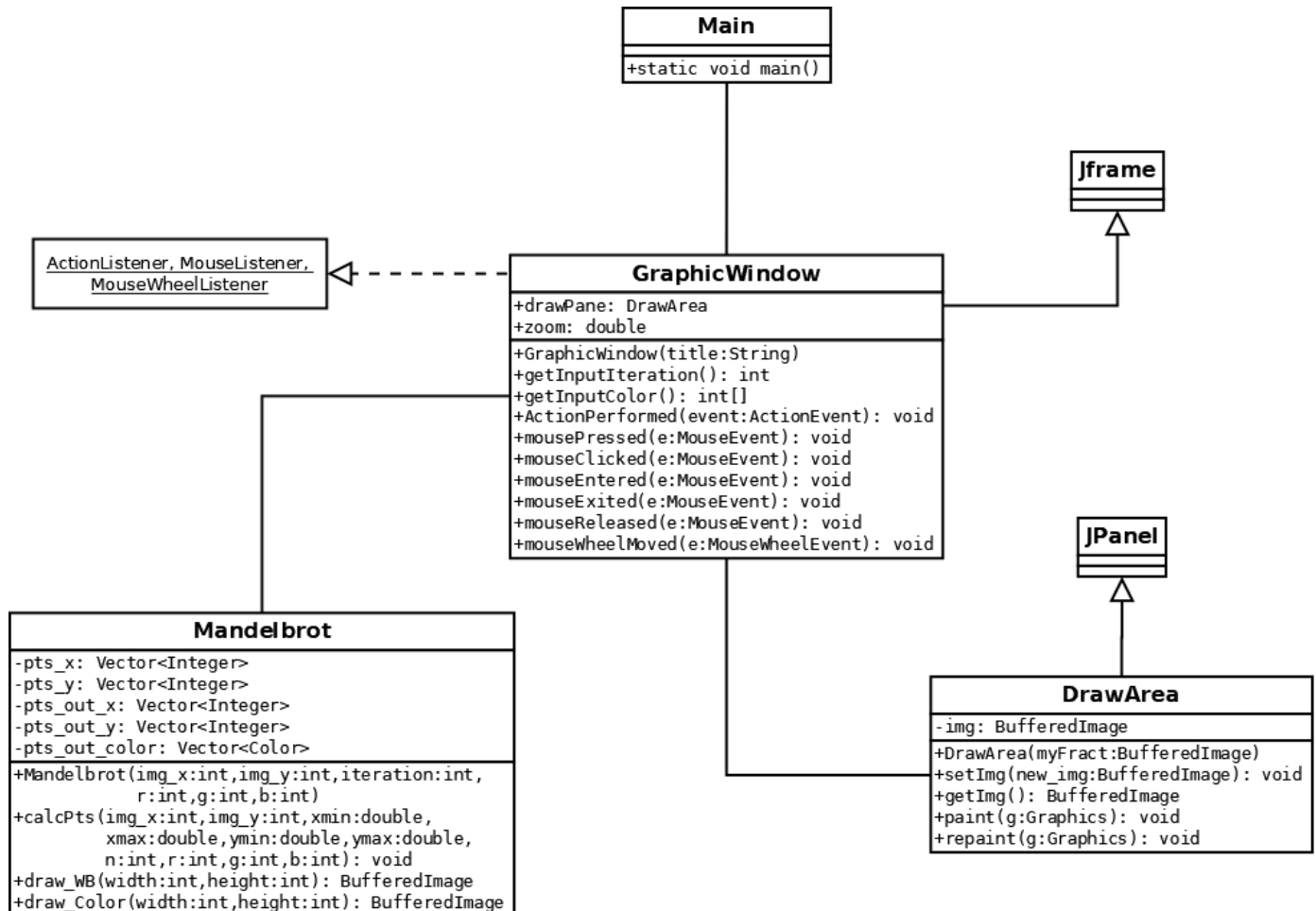
DrawArea
-img: BufferedImage
+DrawArea(myFract:BufferedImage)
+setImg(new_img:BufferedImage): void
+getImg(): BufferedImage
+paint(g:Graphics): void
+repaint(g:Graphics): void

La zone de dessin possède une instance privée `BufferedImage` qui est affichée grâce à la méthode `paint()`.

L'image est accessible par un getter et modifiable par un setter (on change l'image, pas son contenu).

La méthode `repaint()` est appelée à chaque fois que l'on veut redessiner une partie de l'image qui aurait été masquée par un autre élément (une autre fenêtre par exemple).

Diagramme des classes



Suivant cette conception, la méthode MVC est remarquable par le fait que la classe **Main** a pour unique but d'appeler une instance de la classe **GraphicWindow** (la fenêtre de l'appli) qui elle-même se charge d'instancier les deux autres classes **Mandelbrot** et **DrawArea** (la zone de dessin).

De ce fait, la classe **GraphicWindow** assemble les différents composants en utilisant leurs différentes méthodes, à l'aide d'une instance de ces derniers. Plus précisément et lorsqu'il s'agit de dessiner une fractale, **GraphicWindow** instancie une zone de dessin, une **BufferedImage** de la fractale choisie par l'utilisateur (avec les paramètres choisis également), et se charge de transmettre cette image à la zone de dessin qui l'affiche avec la méthode **paint()**.

Les choix techniques

Les choix techniques sont surtout intervenus sur les points suivants :

- *Faut-il dessiner l'image ou directement depuis la fenêtre depuis la classe Mandelbrot ?*

Il est évidemment bien plus pratique de dessiner une `BufferedImage` depuis `Mandelbrot`, car toutes les données nécessaires sont présentes, et cette conception respecte le MVC (la fenêtre graphique ne fait qu'utiliser la fonction qui dessine l'image).

- *Quel type d'algorithme pour calculer l'ensemble de Mandelbrot ? Définir le zoom et calculer les dimensions de l'image ou l'inverse ?*

Pour ce projet, il est beaucoup plus intéressant de savoir à l'avance les dimensions de l'image (les choisir), car les calculer veut dire s'exposer à un risque d'avoir une image trop grande ou trop petite pour la zone de dessin et la fenêtre graphique. Il est important de noter que le temps de dessin est exponentiel plus l'image est grande.

- *Le zoom doit-il être effectué en fonction des données de la fractale, ou celles de l'image en elle-même ?*

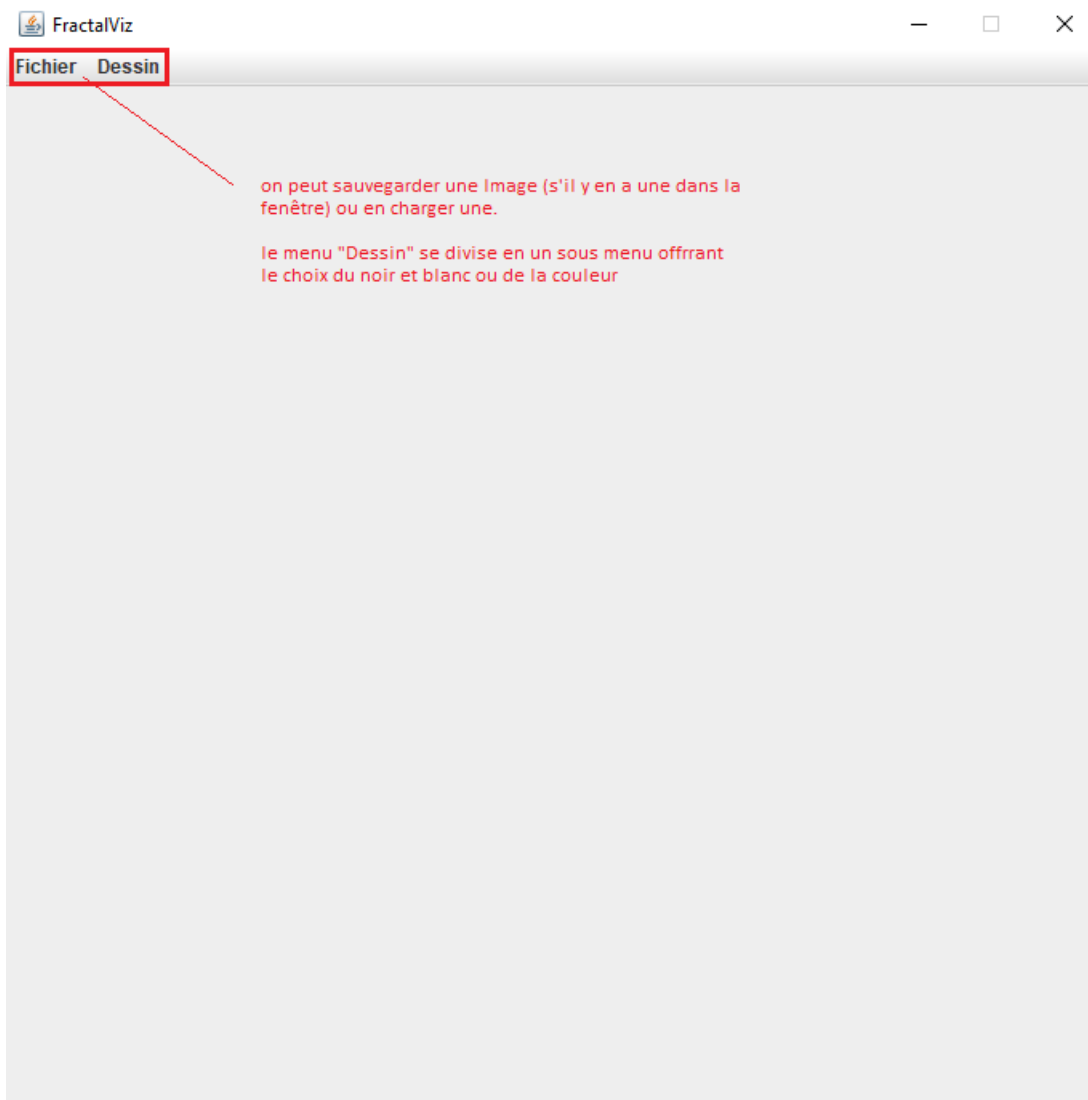
Faire le zoom en fonction de l'image dessinée et non en fonction de `Mandelbrot` est plus facile à mettre en place dans la fenêtre grâce au `MouseWheelListener`. De plus, on connaît à l'avance les dimensions de l'image grâce au choix précédent, ce qui facilite encore plus les calculs liés au zoom.

Utilisation

Mode d'emploi

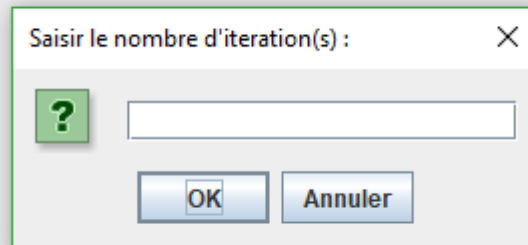
Pour fabriquer l'exécutable, saisir la commande `java -jar FractalViz.jar`.
Si le .jar venait à ne pas fonctionner, il suffit de compiler et d'exécuter la classe Main contenue dans le dossier src.

- Menu utilisateur :



Le nombre d'itération est demandé à chaque fois pour chaque fractale.

Si on appuie sur "Annuler" ou que l'on a rien saisi, l'itération se met à 20 de base

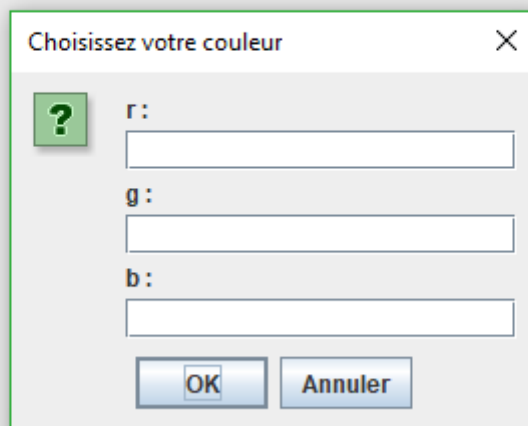


Saisir le nombre d'iteration(s) :

?

OK Annuler

Pour la couleur, l'utilisateur choisi les valeurs RGB. Si rien n'est saisi ou si la valeur dans un champ dépasse 255, le champ vaut automatiquement 255.



Choisissez votre couleur

?

r:

g:

b:

OK Annuler

Configuration requise

Les paramètres requis pour lancer l'application sont les suivants :

```
java version "9.0.1"  
Java(TM) SE Runtime Environment (build 9.0.1+11)  
Java HotSpot(TM) 64-Bit Server VM (build 9.0.1+11, mixed mode)
```

Conclusion

Bilan

Les difficultés rencontrées lors de la programmation concernaient surtout la manière de dessiner la fractale. Le stockage des points posait un problème car il fallait faire attention à ce que l'on faisait dans la boucle de calcul de l'ensemble de Mandelbrot.

Optimisations envisageables

- Améliorer le zoom
- Stocker les points dans des arrayList (plus rapides paraît-il)

Extensions possibles

- Dessiner d'autres fractales comme Buddhabrot ou celle de l'ensemble de Julia
- Permettre de se déplacer de manière libre sur la fractale zoomée en maintenant la souris