BOEHM Guillaume
HECKEL Thibault

11 Janvier 2016

Projet Java – M3105

- Sujet 1 : Fractales -

Sommaire :

I – Introduction :

Citation du sujet donné :

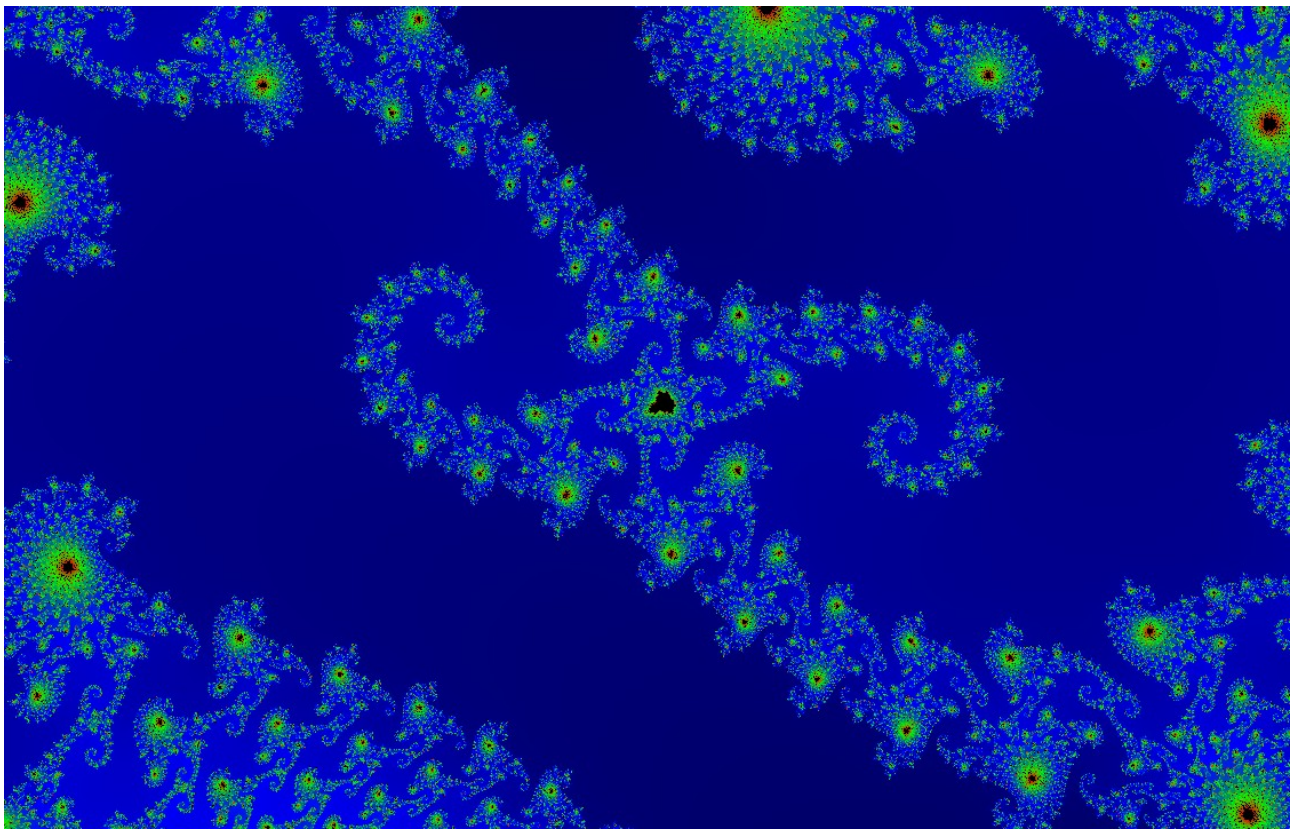
« On souhaite écrire un programme en Java permettant de calculer et d'afficher une fractale portant pour nom l'ensemble de Mandelbrot.
Pour savoir si un point complexe c du plan appartient à l'ensemble de Mandelbrot, on lui fait subir la transformation suivante selon un nombre d'itération fixé :

$$z_n = z^2_{n+1}+c$$

z_n, z_{n+1} et c sont des nombres complexes.

On débute avec $z_0 = 0$, c étant le point étudié. Si au terme de ces itérations le module de z_n est inférieur ou égal à 2, alors il est probable qu'il appartienne à l'ensemble de Mandelbrot (la certitude augmente avec le nombre d'itérations). Si le module dépasse 2 avant la dernière itération, alors il y a certitude que ce point n'appartient pas à l'ensemble.

On devra pouvoir choisir le nombre d'itérations, zoomer ou dézoomer à la souris sur une partie de la fractale. La fractale devra occuper toute la surface de la fenêtre, même si on l'agrandit ou la réduit. »



Capture d'écran du programme :

Points = -0.743643887037151 + 0.131825904205331i , zoom = 5329501

II – Analyse :

Pour répondre à la problématique posée nous avons préféré diviser notre travail comme nous le faisions en TP de Java. Soit les cinq classes suivantes :

- **Principal** : Contient la fonction « main » appelant la création d'une nouvelle instance de la classe Window – qui comme son nom l'indique créer une fenêtre (JFrame).

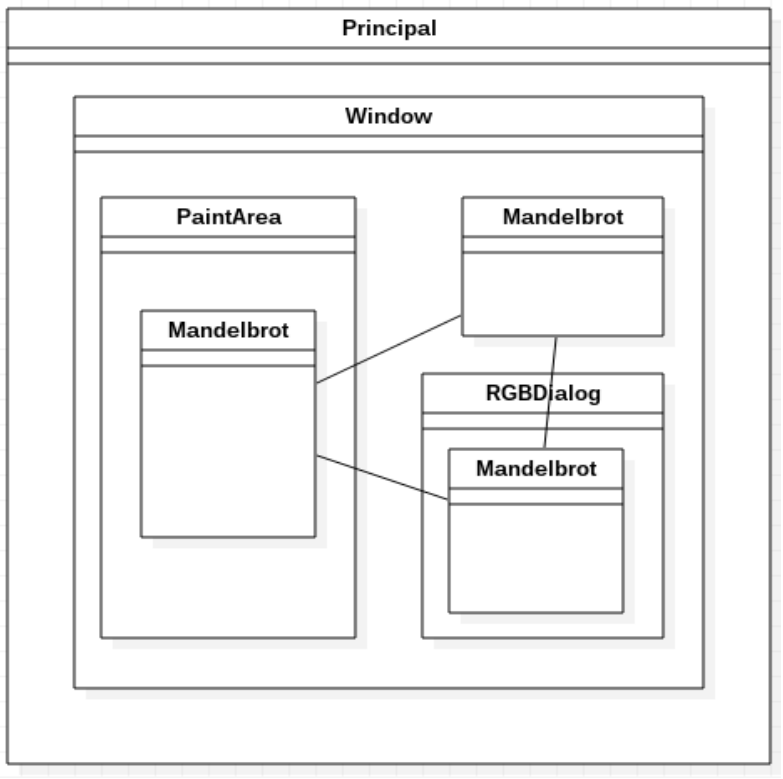
- **Window** : Cette classe répertorie tout ce qui est en rapport avec l'IHM de notre application : création de la fenêtre, implantation des options demandés sous formes de boutons et de menu, affichage de fenêtre de dialogue. La fractale doit être dessiné dans la fenêtre créée. C'est pourquoi celle-ci créer une instance de la classe PaintArea qui est incrusté au JPanel, lui-même contenu dans le JFrame créé en tout début.

- **Mandelbrot** : C'est la pièce maîtresse de notre programme. Cette classe contient toutes les informations dont nous avons besoin pour construire notre fractale. Grâce à celle-ci il est possible de généraliser notre travail et de l'appliquer dans d'autre programme. Celle-ci contient des membres de classe (static) ce qui nous permet d'en modifier la teneur dans dans notre programme.

- **PaintArea** : Cette classe permet de dessiner notre fractale. Elle réagit aussi aux modifications que nous pouvons apporter en cliquant et en activant la molette de notre souris.

- **RGBDialog** : Héritant de la classe JDialog, cette classe défini une boîte de dialogue personnalisée

Pour résumer nous pouvons dire que notre code lie nos classes de cette façon :



La classe Principal contient la classe Window, qui elle-même contient les classes PaintArea, RGBDialog et Mandelbrot. Cependant PaintArea et RGBDialog font elles aussi appel à Mandelbrot.

Les liaisons visibles sur ce schéma traduisent les membres static de la classe Mandelbrot. Cela permet notamment de montrer que toutes modifications appliquées à l'un de ses trois appels touche l'ensemble du programme.

III – Réalisation

La classe Mandelbrot est détaillée dans les commentaires de celle-ci.

PaintArea :

Fonction paint :

```
46          for(int x=-this.getSize().width/2;x< this.getSize().width/2 ;x++)
47              for(int y=-this.getSize().height/2;y<  this.getSize().height/2 ;y++){
48                  fra.setzr(0);
49                  fra.setzi(0);
50
51                  fra.setcr( (double)x/ fra.getZoom() + fra.getXPaint() );                  // points c du plan complexe, coordonnée x (réelle)
52                  fra.setci( (double)y/ fra.getZoom() + fra.getYPaint() );                  // coordonnée y (imaginaire) de c
53
54                  int i=0;
55                  do{
56                      double tmp = fra.getzr();
57
58                      fra.setzr( Math.pow( fra.getzr(), 2 ) - Math.pow( fra.getzi(), 2 ) + fra.getcr() );
59                      fra.setzi( (2*fra.getzi()*tmp) + fra.getci() );
60
61                      ++i;
62                  }while( Math.pow( fra.getzr(), 2 ) + Math.pow( fra.getzi(), 2 ) < 4 && i <  fra.getiNumber() );/*répeter l'operation jusqu'à ce que
63                                                                                                                  le module du nombre complexe dépasse
64                                                                                                                  4 ou que le nombre d'iterations max
65                                                                                                                  soit atteint*/
```

Dans cette portion du code on peut voir sur les deux premières lignes les deux boucles **for** imbriqués permettant de calculer et afficher les points de la fractale non pas de **0 - largeur** et **0 - hauteur** mais de **-largeur/2 - largeur/2** et de même en hauteur, la raison de ce décalage est de pouvoir par la suite zoomer la fractale au **centre** de la fenêtre et non pas dans le coin en haut à gauche.

Ensuite viens la boucle **while**, qui permet de calculer séparément la composante réelle et irréelle du nombre jusqu'à ce que son **module** dépasse 2 ou que le nombre **d'itérations** voulu ai été atteint.

```
68                  double module = Math.sqrt( Math.pow(fra.getzr(),2) + Math.pow(fra.getzi(),2) ); //utilisé pour la coloration dégradée
69                  /*definir la couleur du pixel a dessiner*/
70                  if(i < fra.getiNumber() ){
71                      if(fra.multiple_color){ // si le booleen multiple color est a true on dessine la fractale selon ce schema de coloration dégradée
72                          double mu = i - Math.log(Math.log( module ))/ Math.log(2);
73                          int colorIndex = (int) ( (mu/fra.getiNumber()) * 768);
74                          if(colorIndex >= 768) colorIndex=0;
75                          if(colorIndex <0) colorIndex = 0;
76                          fra.setColor(fra.colorTab[colorIndex]);
77                      }
78                      else{//sinon on la dessine selon ce schema
79                          double red = i*fra.basecolor.getRed()/fra.getiNumber();
80                          double green = i*fra.basecolor.getGreen()/fra.getiNumber();
81                          double blue = i*fra.basecolor.getBlue()/fra.getiNumber();
82                          fra.setColor(new Color((int)red,(int)green,(int)blue));
83                      }
84                  }
85                  else
86                      fra.setColor(Color.black);
87                  /*dessine le pixel*/
88                  g.setColor(fra.getColor());
89                  g.drawLine(x+this.getSize().width/2,-y+this.getSize().height/2,x+this.getSize().width/2,-y+this.getSize().height/2);
```

La variable **module** (qui stocke le module du nombre complexe calculé) permet d'effectuer le calcul pour choisir la couleur dans le schéma de coloration dégradée (ligne 71 – schéma trouvé sur internet), le deuxième schéma de coloration (ligne 78) change simplement l'intensité de couleur du pixel selon son nombre d'itérations **(+itérations = +intensité)**, si le module du nombre complexe a dépassé le nombre d'itérations choisi, alors le pixel est éclairé en **noir** (ligne 86). On effectue ensuite un **drawLine** d'un **pixel de largeur** lui aussi **décalé** par rapport à x et y, cela rejoint le décalage des boucles **for** imbriquées en début de fonction.

La classe **PaintArea implémente MouseListener** ainsi que **MouseWheelListener** afin de pouvoir **zoomer** et **dézoomer** la fractale avec la **molette** de la souris, et pouvoir choisir un **point à zoomer** en **cliquant** avec la souris.

```
94
95      public void mouseClicked(MouseEvent e){
96          if (e.getButton()==MouseEvent.BUTTON1){
97             fra.setXPaint(fra.getXPaint()-((this.getSize().width/2-e.getX())/(double)fra.getZoom()));
98             fra.setYPaint(fra.getYPaint()+((this.getSize().height/2-e.getY())/(double)fra.getZoom()));
99             this.repaint();
100         }
101     }
```

La surcharge de l'événement **mouseClicked** permet donc de décrire le comportement du programme quand un bouton de la souris est préssé.

À la ligne 95, on prend **l'écart** (en **pixel**) entre le point actuellement **centré** et le point **voulu**, et on **divise** le tout par le **zoom** actuel, ce qui nous ramène à l'ensemble mathématique. On **soustrait** ensuite ce nombre au point(mathématique), ce qui nous donne le **nouveau point** de la fractale à zoomer.

```
101     public void mouseWheelMoved(MouseWheelEvent e){
102         if(fra.getZoom()>=fra.getRatio()){
103             fra.setZoom(fra.getZoom()-e.getWheelRotation()*(fra.getZoom()/fra.getRatio()));
104             this.repaint();
105         }
106         else{
107             fra.setZoom(fra.getRatio());
108             JOptionPane jop2 = new JOptionPane();
109             jop2.showMessageDialog(null, "Impossible d'aller plus loin", "Attention", JOptionPane.WARNING_MESSAGE);
110             this.repaint();
111         }
112     }
```

La surcharge de l'événement **mouseWheelMouved** permet de décrire le comportement du programme lorsque la molette de la souris est bougée.

Afin d'éviter de zoomer de 1 en 1, ce qui serait extrêmement **lent**, on **multiplie** la **rotation** de la molette par un certain nombre, ce certain nombre étant : le **zoom actuel / un ratio**. À la base l'intérêt était de multiplier simplement par ratio, seulement au **plus** on zoom dans la fractale au **moins** le zoom se fait sentir, nous avons donc décidé de multiplier la rotation de la molette **en fonction du zoom**, celui-ci serait donc divisé par une certaine ration afin de rester raisonnable.

Window :

Cette classe hérite de JFrame et implémente l'interface ActionListener. De ce fait il nous faut définir la fonction :

```
public void actionPerformed(ActionEvent evenement){
```

Celle-ci rassemble toutes les actions à effectuer suivant les signaux activés (prédéfinis dans la classe).

Ces signaux en question sont en rapport avec ce qui est demandé au début du

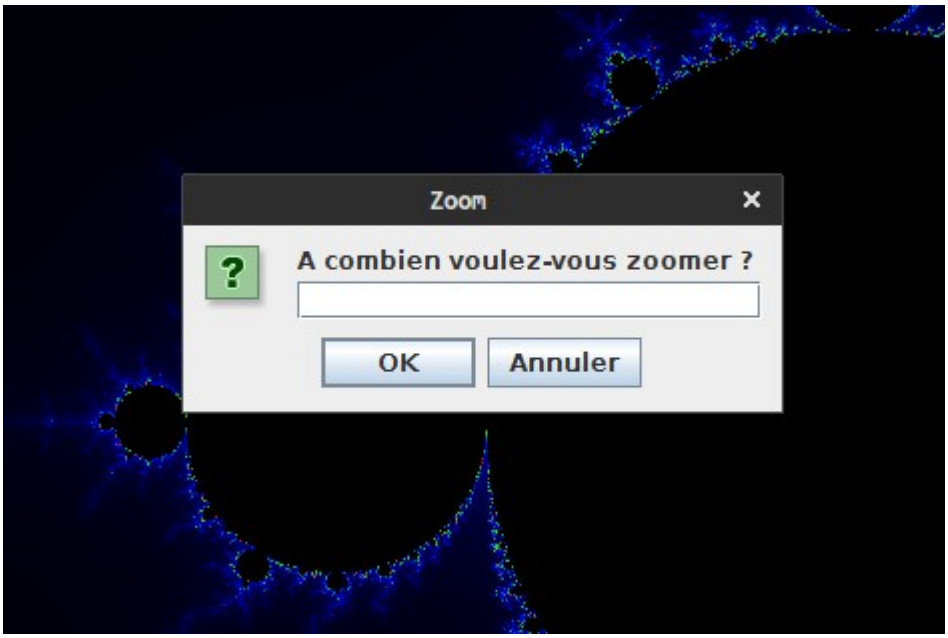
projet :

« On devra pouvoir choisir le nombre d'itérations, zoomer ou dézoomer à la souris sur une partie de la fractale. La fractale devra occuper toute la surface de la fenêtre, même si on l'agrandit ou la réduit. »

Ces signaux résultent d'un clic de souris sur un des items appartenant à la barre des tâches.

Pour interagir avec l'utilisateur nous avons préféré des boîtes de dialogue plutôt que la console terminale.

Ainsi pour chaque options une boîte de dialogue apparaîtra.
De la forme :



Option « Zoom »

La plupart de ces boîtes de dialogue sont des JoptionPane sauf pour l'Option « Color ». En effet pour celle-ci nous avons utilisé une classe héritant de JDialog : RGBDialog et donc personnalisée.

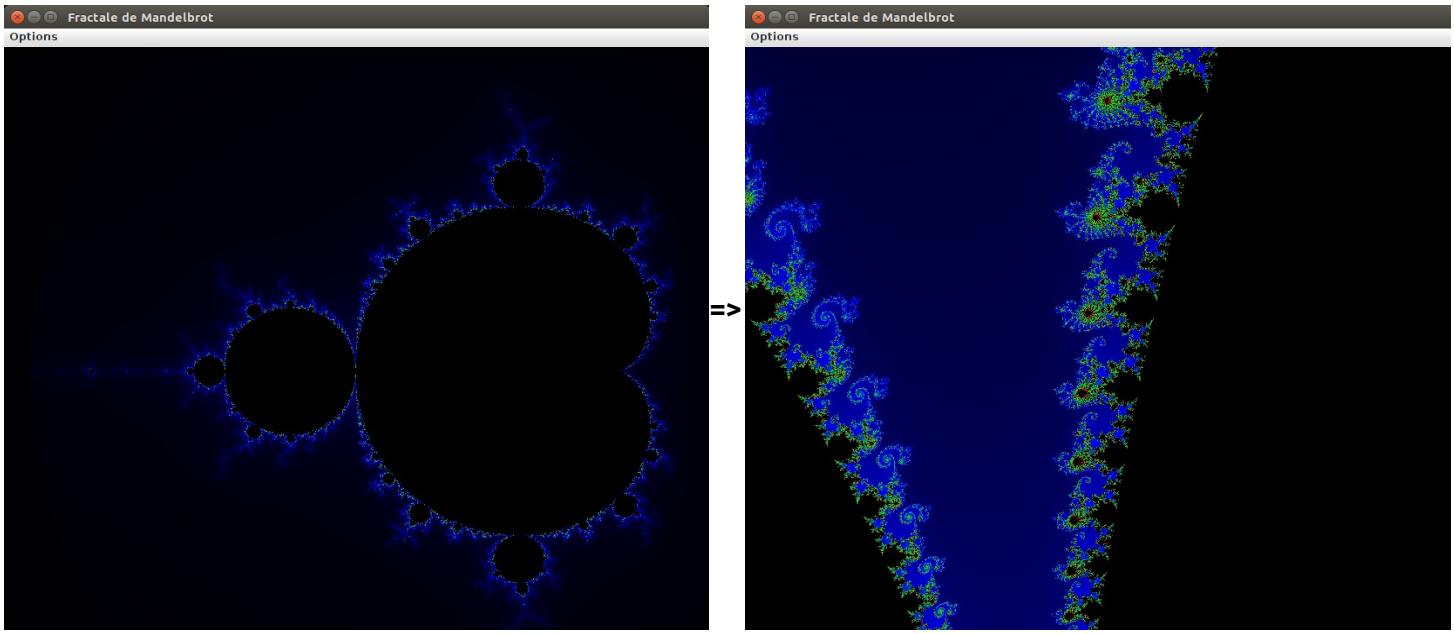
```
this.repaint(); // Nécessaire pour redessiner notre fractale.
```

Dans la majeure partie des cas il est nécessaire de redemander à la fenêtre de se redessiner car nous avons effectué un changement sur l'ensemble de Mandelbrot. Pour ce faire nous utilisons donc la méthode repaint().

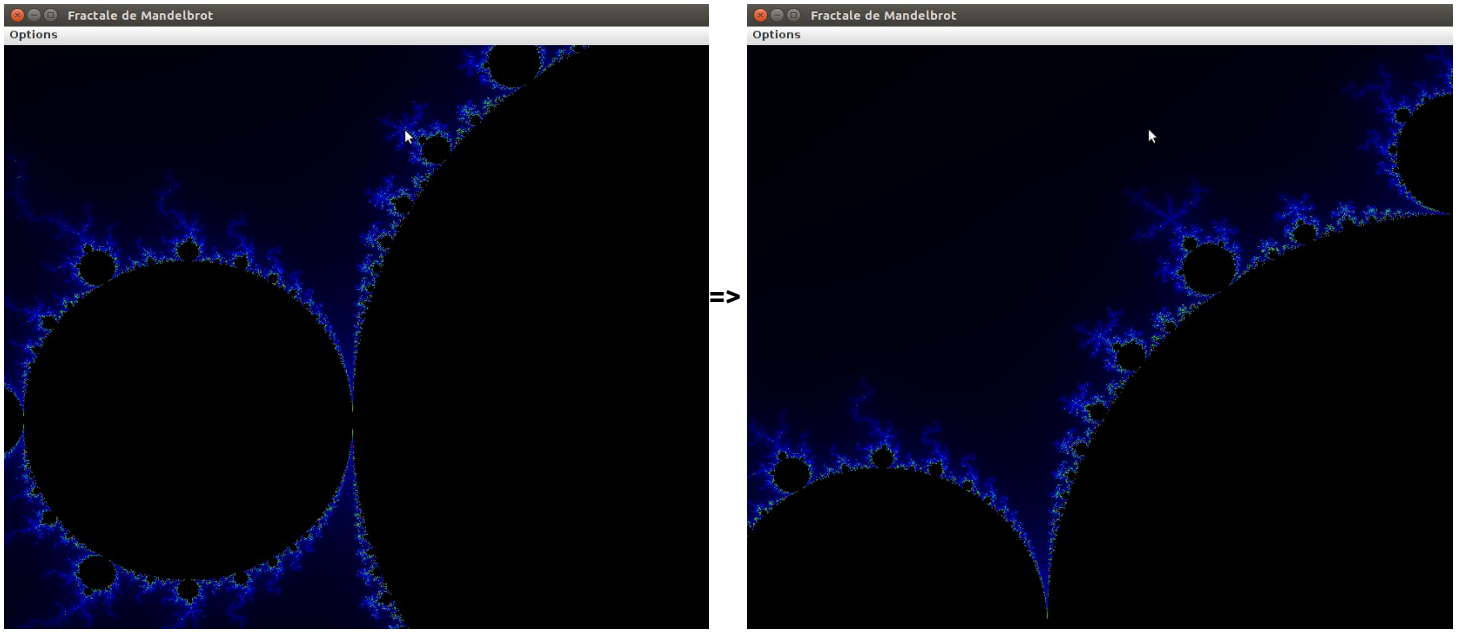
RGBDialog : Cette classe hérite donc comme il a été dit plus haut de JDialog. Nous construisons pas-à-pas notre boîte de dialogue à l'aide de JPanel, JLabel, JTextField et JButton. Jusqu'à définir nos propres actions lors de validation d'une option. Les commentaires dans la classe suffisent à la compréhension.

IV - Utilisation

Dans un premier temps il est possible de zoomer dans la fractale grâce à la molette de la souris :

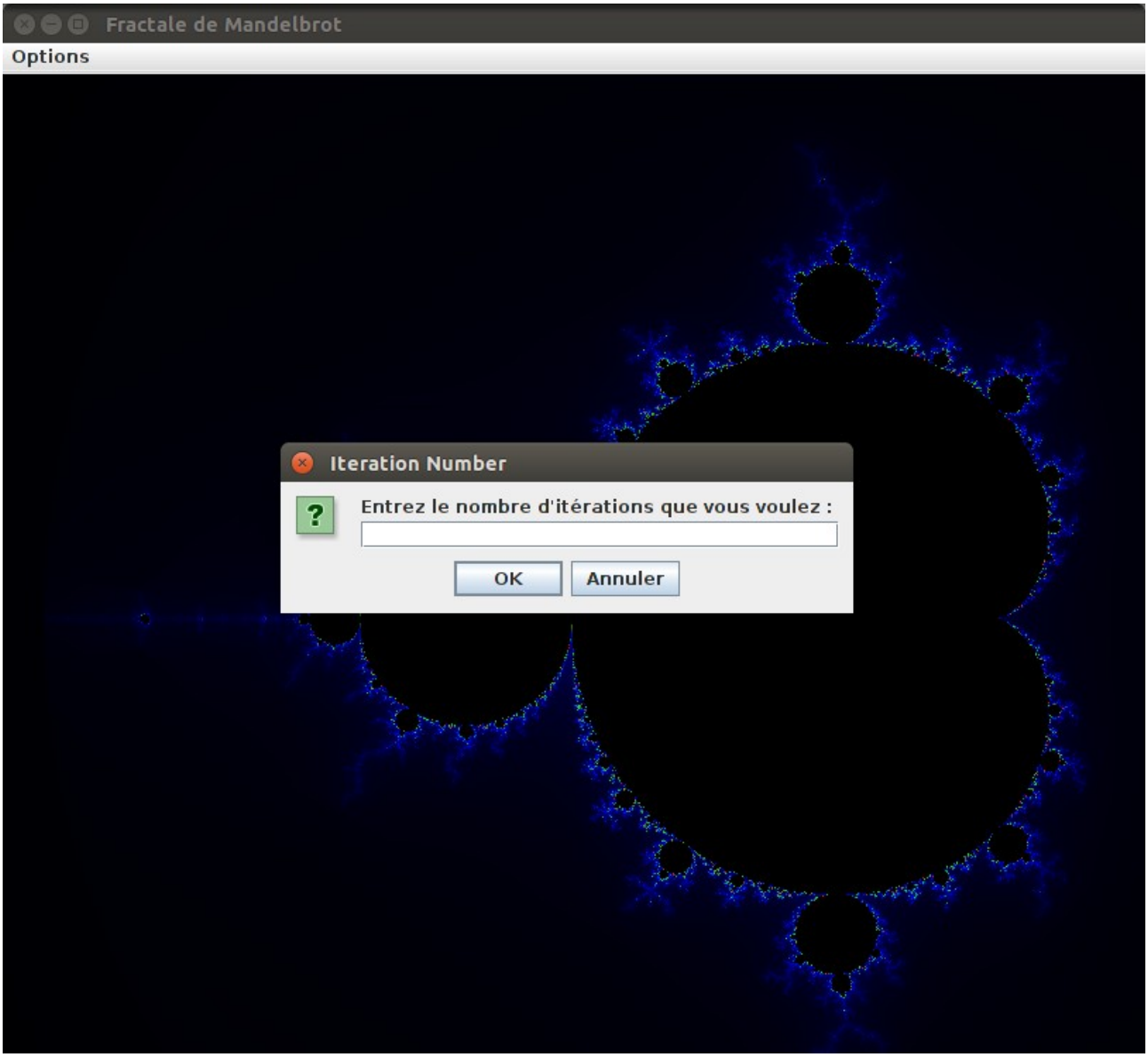


Il est également possible de changer de point de zoom en cliquant avec la souris :

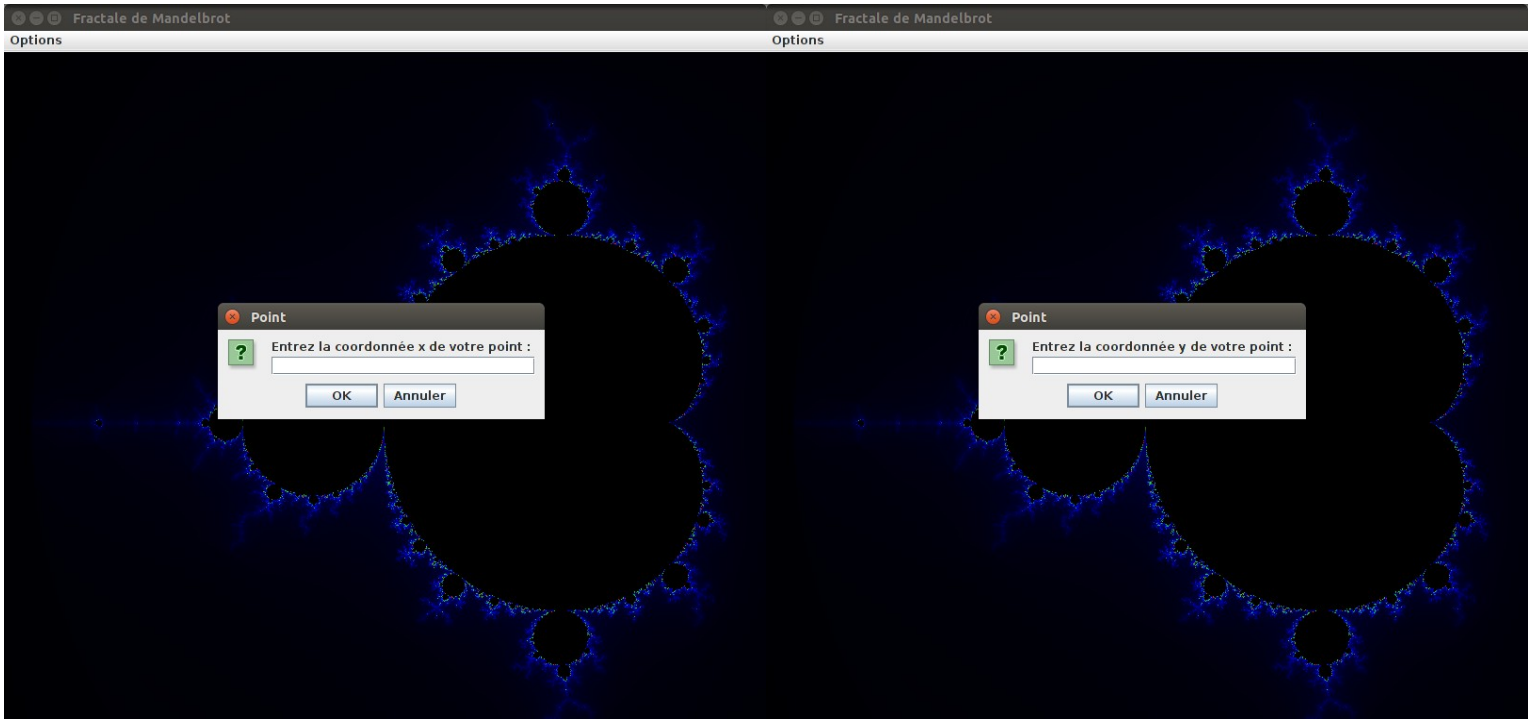


Une menu d'options permet aussi de modifier divers aspects et options de la fractale, tel que :

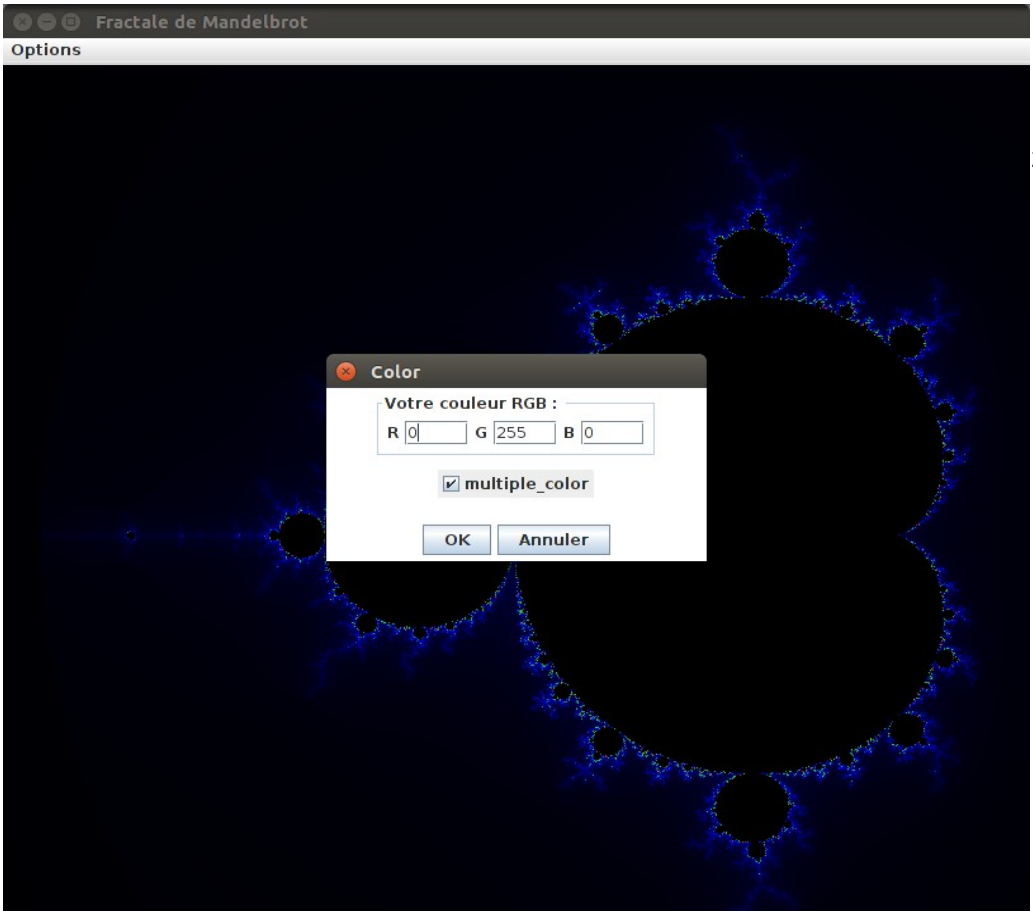
-le nombre d’itérations :



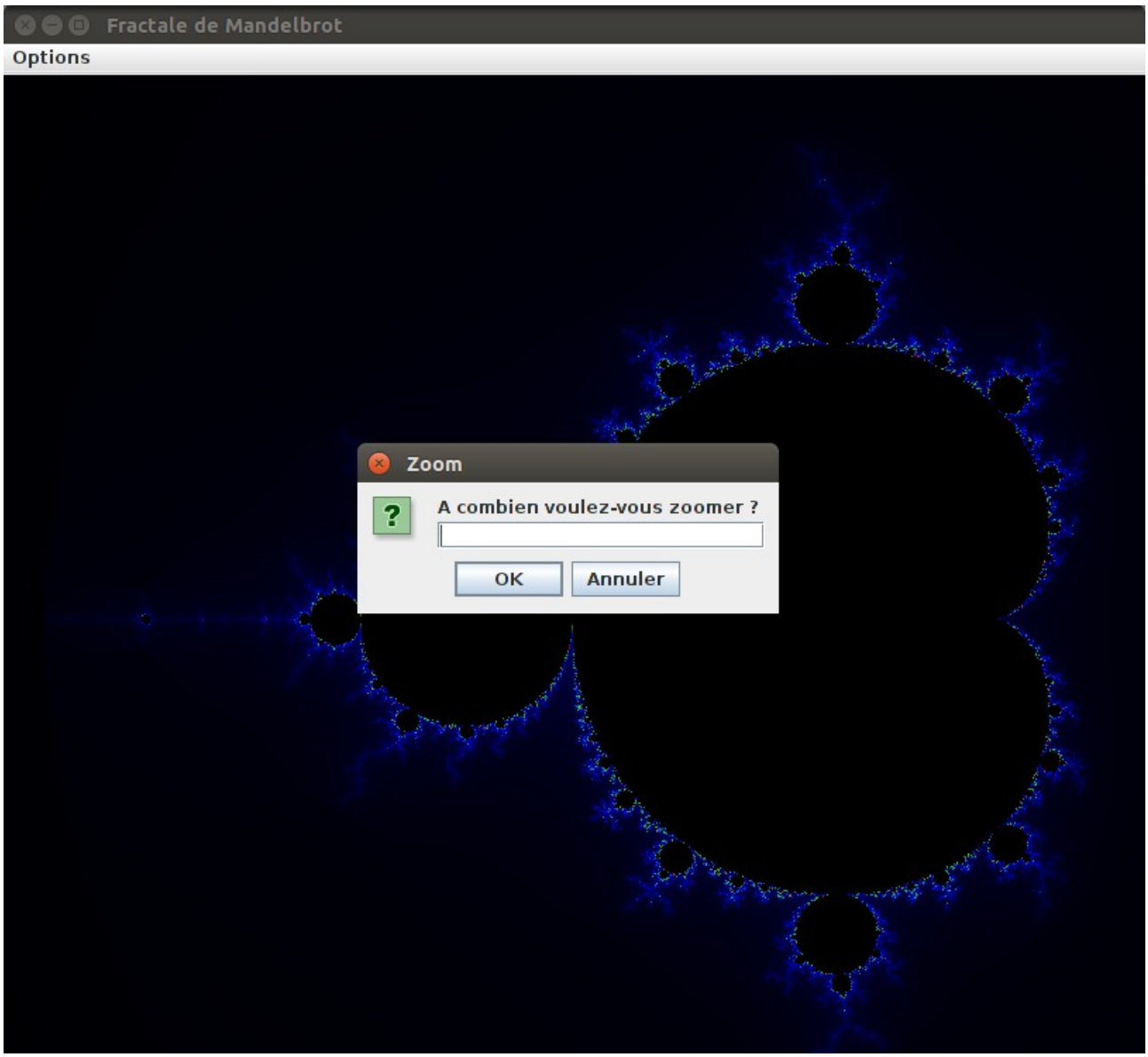
-le point de la fractale à zoomer :



-la couleur de la fractale et l'option monochrome/couleurs dégradées :



-le zoom :



Il s'y trouve également un bouton de réinitialisation pour remettre la fractale dans le même état qu'elle se trouvait au lancement du programme ainsi qu'un bouton « quitter » afin de fermer le programme.

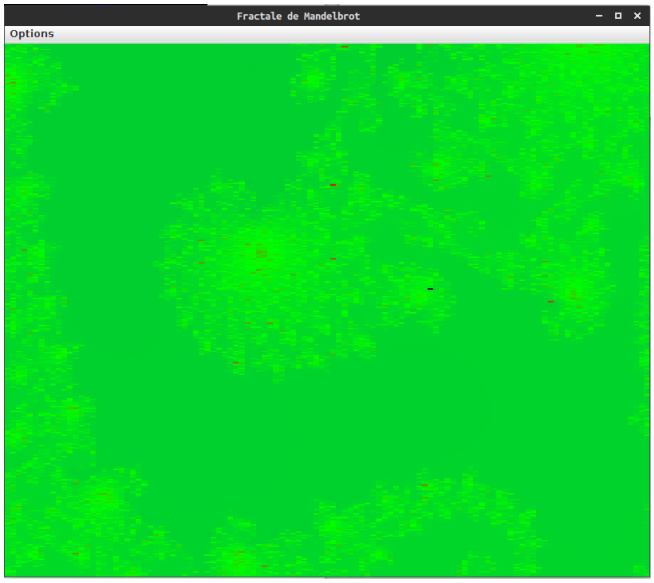
V – Conclusion

Bilan :

Ce projet nous a permis de mettre en œuvre ce que nous avons appris jusqu'à présent en cours et TP de Java. Notamment la création de fenêtre Java avec des outils à notre porté. La fractale de Mandelbrot est un sujet très intéressant car peu anodin et porte à de bonnes réflexions. Étant très prit d'intérêt par le sujet nous avons rapidement compléter les requis du projet et avons pu alors nous essayer à divers améliorations (Couleurs, clic à la souris, options changement de zoom, …).

Optimisations possibles :

- La mise en place d'une classe pouvant nous permettre de dépasser la précision du Double (et/ou du Long), afin de palier à un problème de pixellisation survenant à un zoom élevé :



Points = -0.743643887037151 + 0.13182590420533i, Zoom = 64579368259391376
Itérations = 5000

- Passer d'un code séquentiel à du multi-thread ;
- Optimiser la palette de couleur afin d'éviter le plus possible de perte de luminosité.

Extensions possibles :

Généraliser notre programme de façon à pouvoir intégrer d'autres fractales que l'ensemble de Mandelbrot.