

Exercice A.1 Observation d'un gain Le répertoire TP/Monte-Carlo disponible dans l'archive TP_A.tgz fournit le code de l'exercice I.3 (page 2) et celui d'une solution parallèle avec 4 threads. ☕

- Question 1. Trouvez, par tâtonnement, un nombre de tirages qui nécessite, pour le programme séquentiel, environ 30 secondes d'exécution sur votre machine.
- Question 2. Quel est le temps de calcul du programme parallèle fourni pour le même nombre de tirages ?
- Question 3. Le gain observé (c'est-à-dire le rapport entre le temps de calcul séquentiel sur le temps de calcul parallèle) vous paraît-il cohérent avec le nombre de processeurs disponibles sur votre machine ? ☒

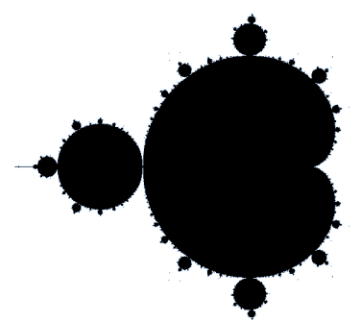
Exercice A.2 Partage statique et dynamique d'un dessin L'ensemble de Mandelbrot est une fractale définie comme l'ensemble des points c du plan complexe pour lesquels la suite définie par : $z_{n+1} = z_n^2 + c$ avec $z_0 = 0$ ne tend pas, en module, vers l'infini. Si nous reformulons cela sans utiliser les nombres complexes, en remplaçant z_n par le couple de réels (x_n, y_n) et c par le couple (a, b) , alors nous obtenons : ☒

$$x_{n+1} = x_n^2 - y_n^2 + a \text{ et } y_{n+1} = 2.x_n.y_n + b \text{ avec } x_0 = y_0 = 0$$

la condition devenant que ni x_n , ni y_n ne tendent vers l'infini (en valeur absolue).

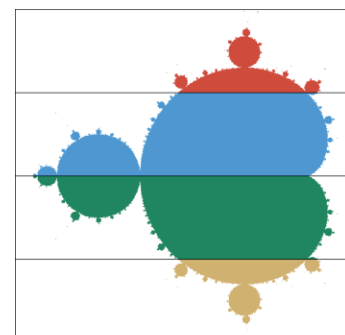
Dans la pratique, nous allons calculer les premiers termes de cette suite pour un point (a, b) fixé. Si la valeur de $x_n^2 + y_n^2$ dépasse 4, nous considérerons (de manière arbitraire) que la suite diverge, et donc que le point (a, b) n'est pas dans l'ensemble. En revanche si la valeur $x_n^2 + y_n^2$ ne dépasse 4 pour aucun des termes calculés, nous considérerons que la suite ne diverge pas, et donc que le point (a, b) est dans l'ensemble. Ceci nous conduit à utiliser la méthode suivante, qui détermine si oui ou non le point (a, b) est dans l'ensemble de Mandelbrot lorsque l'on examine les max premiers éléments de la suite (x_n, y_n) .

```
static boolean mandelbrot(double a, double b, int max) {
    double x = 0;
    double y = 0;
    for (int t = 0; t < max; t++) {
        if (x*x + y*y > 4.0) return false; // Le point est blanc
        double nx = x*x - y*y + a;
        double ny = 2*x*y + b;
        x = nx;
        y = ny;
    }
    return true; // Le point est noir
}
```



Le programme Mandelbrot.java disponible dans l'archive TP_A.tgz est donné sur la figure 3. Il calcule et affiche l'ensemble de Mandelbrot situé dans la région carrée comprise entre $(-1.5, -1)$ en bas à gauche et $(0.5, 1)$ en haut à droite. Le nombre de pixels par ligne (et par colonne) est fixé à `taille=500` alors que le nombre maximal d'itérations vaut `max=40_000`. Le programme indique également le temps de calcul de l'image. Vous pouvez annuler l'affichage de l'image en supprimant l'instruction `image.show()`.

- Question 1. Modifiez la valeur de `max` pour que le calcul de l'image avec ce programme dure environ une trentaine de secondes.
- Question 2. Partagez ensuite horizontalement le calcul de l'image sur 4 threads, chacun calculant un quart de l'image, sans modifier les fonctions `colorierPixel()` et `mandelbrot()`.
- Question 3. Quel est le gain, en terme de temps de calcul de l'image complète, par rapport à la version séquentielle ? Quel est le temps de calcul de chacun des quatre threads ?
- Question 4. Attribuez à présent aux 4 threads les lignes à colorier à la volée, au fur et à mesure qu'ils sont disponibles. Quel est le nouveau gain, en terme de temps de calcul, par rapport à la version séquentielle ?



Exercice A.3 L'état du cobaye (suite) Nous poursuivons l'exercice I.5. Pour chacun des programmes du répertoire TP/Cobaye, devinez l'affichage produit par chaque appel à la méthode `Affiche()` puis vérifiez votre hypothèse en lançant une exécution. ☑

```

import java.awt.Color;

public class Mandelbrot {
    final static int taille = 500;    // Nombre de pixels par ligne (et par colonne)
    // Il y a donc taille*taille pixels blancs ou noirs à déterminer
    final static Picture image = new Picture(taille, taille);
    final static int max = 40_000;
    // C'est le nombre maximum d'itérations pour déterminer la couleur d'un pixel


    public static void main(String[] args) {
        final long startTime = System.nanoTime();
        for (int i = 0; i < taille; i++) {
            for (int j = 0; j < taille; j++) {
                colorierPixel(i, j);
            }
        }
        final long endTime = System.nanoTime();
        final long durée = (endTime - startTime) / 1_000_000 ;
        System.out.println("Durée=_ " + (long) durée + " _ms.");
        image.show();
    }

    // La fonction colorierPixel(i,j) colorie le pixel (i,j) de l'image
    public static void colorierPixel(int i, int j) {
        ...
    }
}


/*
$ javac Mandelbrot.java
$ java Mandelbrot
Durée = 17831 ms.
$ make
$ java -jar Mandelbrot.jar
Durée = 17869 ms.
*/

```

FIGURE 3 – Fractale de Mandelbrot

Exercice A.4 Fabrique d'un GIF animé Pour conclure, il s'agit de fabriquer une image GIF animée, telle que celle donnée dans le répertoire GIF de l'archive TP_A.zip, afin d'illustrer l'évolution du programme Mandelbrot.java fourni dans l'archive TP_A.zip. Pour celà, il faut au préalable construire une série d'images qui décrivent l'évolution de l'état de la fenêtre au cours de l'exécution du programme. 

Question 1. Modifiez la valeur de **max** du code pour que l'exécution dure entre 20 et 60 s. sur votre machine.

Question 2. L'instruction **image.save("toto.png")** sauvegarde dans le fichier **toto.png** une image au format PNG qui correspond à l'état courant de l'objet **image**. Modifiez le programme pour sauvegarder, dans une série de fichiers numérotés sur trois chiffres et nommés sous la forme **picXXX.png**, les états intermédiaires du dessin, à raison d'un fichier toutes les 100 millisecondes, c'est-à-dire dix images par seconde. Ça ne devrait donc pas faire plus que 600 fichiers, approximativement. Vous prendrez soin néanmoins d'empêcher votre programme de produire plus d'un millier de fichiers PNG quelque soit la durée réelle de l'exécution du programme. 

Question 3. Lancez la commande Linux : **convert pic*.png m.gif** puis affichez le fichier **m.gif** obtenu.

Question 4. Une curiosité apparaît lorsque l'on observe certaines des dernières images PNG produites. Pouvez-vous l'expliquer ?