Mandelbrot
Inlämningsuppgift 2, Programmeringsteknik

Fredrik Danebjer, D12 (dat12fda@student.lu.se)
Patrik Brosell, D12 (dat12pbr@student.lu.se)

27 november 2012

1 Bakgrund

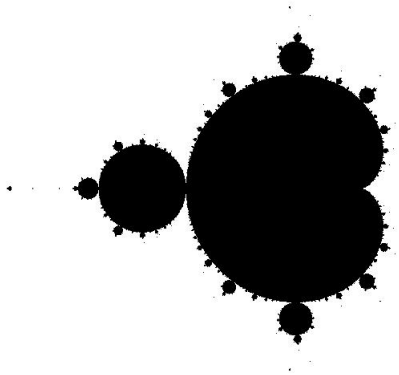
Mandelbrot är ett fraktal, vilket är anknutet till kaosteori. Ett fraktal innebär att man har en talföljd med ovanliga egenskaper, i detta fallet att dess mönster upprepar sig på olika sätt ut i oändligheten när man förstorar bilden (zoomar in) på Mandelbrotmängden. En Mandelbrotföljd ska således ritas upp på ett komplext talplan MandelbrotGUI, och man ska kunna zooma in på denna talföljd för att studera dess upprepningar och egenskaper.

Mandelbrotmängden definieras på följande vis:

$z_k = z_{k-1}^2 + c$

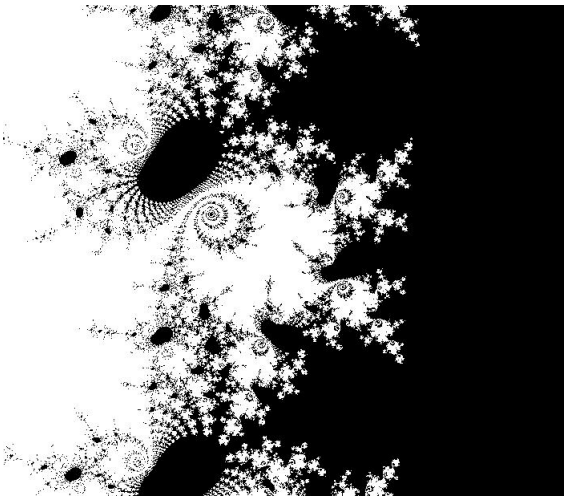
c = komplexa talet i punkten som beräknas för $D_k = [1, \infty[$
∞ kommer i programmet representeras av MAX_ITERATIONS.
z_0 är 0, så $z_1 = c$. Nästa z-värde blir $z_2 = c^2 + c$, nästa igen
blir $z_3 = c^4 + 2c^3 + c^2 + c$, och så vidare.
Talet tillhör Mandelbrotmängden då:

$|z_k| < 2$



Figur 1: En bild av Mandelbrotmängden.

Ett program som simulerar en Mandelbrotmängd av komplexa tal, och som sedan ritas upp i ett MandelbrotGUI `mGUI` med hjälp av en Generator `generator` skall skrivas. `Main-metoden` `Mandelbrot` fungerar som en medlare mellan `MandebrotGUI` och `Generator`. Man kan tänka sig `mGUI` som en avancerad och speciell variant av ett `SimpleWindow`, och på detta så ska ett lager av komplexa tal läggas. En algoritm skall avgöra om talen tillhör Mandelbrotmängden, och färglägga varje pixel för motsvarande komplext tal beroende på detta.



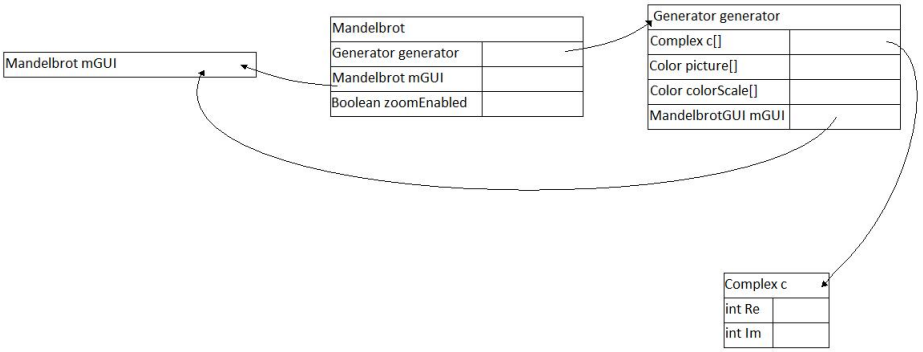
Figur 2: En bild då man zoomat in på Mandelbrotmängden.

2 Modell

`Mandelbrot`	Innehåller `main-metoden` som skapar `MandelbrotGUI`-objektet och `Generator`-objektet. Klassen fungerar sedan som en medlare mellan `MandelbrotGUI` kommandona (knappar och textruta) och `Generator`-klassen som exekverar dessa knappars funktioner.
`Generator`	Beskriver en Mandelbrotmängd och metoden `Render`. Klassen skapar en vektor av komplexa tal och placerar ut dessa över fönstret. Beskriver också de färgskalor som används när man väljer att använda färger (och tolkar då innehållet i "Extra"-rutan). I `Render` metoden så avgör denna klass också hur stora pixlar som skall ritas, beroende på vad man har valt för upplösning i fönstret.
`Complex`	Beskriver ett komplext tal, samt hur detta tal kan multipliceras och adderas.
`MandelbrotGUI`	Beskriver ett fönster där Mandelbrotmängden skall ritas upp. Beskriver metoderna `zoom` och `reset`.

Simuleringen implementeras på så vis att klasserna `MandelbrotGUI` och `Generator` skapas i `main-metoden`. `main-metoden` fungerar som en medlare till `MandelbrotGUI` och `Generator`.

Komplexa tal beskrivs av klassen `Complex`, och en matris av dessa skapas i



Figur 3: En bild som visar sambanden mellan klasserna.

`Generator` klassen. `Generator` klassen ansvarar sedan för att placera ut dessa komplexa tal, avgöra om dem tillhör Mandelbrotmängden, och svara till de ”commands” som finns i `MandelbrotGUI`; tillexempel så beskrivs metoden `Render` i `Generator`, och knappen `Render` finns i MandelbrotGUI. Såfort ett ”command” anropas via `MandelbrotGUI` så märker `main-metoden` om detta och talar om för generator klassen att det är dags att exekvera en viss sats.

- Vid start så skrivs instruktioner för ”Extra”-rutan i `MandelbrotGUI` ut i konsolen. `MandelbrotGUI` öppnas och väntar på att man ska ge kommando via de knappar som finns i `MandelbrotGUI`. Då man anropar Render så skapas en matris av komplexa tal som läggssom ett lager ovanpå `mGUI`. Om man har zoomat in så kommer detta påverka de komplexa talens värde på planet (per logik). `Render` metoden avgör sedan för varje komplext tal om detta tillhör Mandelbrotmängden, och färgar sedan pixeln på motsvarande plats efter svaret.

- `Generator` klassen svarar med switchar emot övriga kommandon på `mGUI`; om upplösningen sänks så kommer det för varje komplext tal som beräknas färgas fler pixlar i dess omgivning. Per denna logik så beräknas inte varje komplext tal, utan hoppar över vissa för att låta motsvarande pixlar ingå i omgivningen.

- Om man väljer att använda färger så svarar `Generator` klassen på detta med en switch, beroende på vad man har skrivit i ”extra”-rutan så kommer lämplig färgskala placeras till `Render` metoden när pixlarna färgläggs.

3 Brister och kommentarer

Pågrund av minnesbrist så begränsas färgskalorna till en blanding av som mest två grundfärger. Om vi hade velat använda oss av ett fullt färgspektrum så hade detta lett till massiva beräkningar både för färgskalan och iterationsberäkningarna. Detta har att göra med att antalet färger måste

gå hand i hand med antalet iterationer om man vill ha en effektiv bild med färger. Vill man använda samtliga färger och få detta att stämma överrens med färgskalan så krävs rotberäkningar, och detta kräver stora mängder minne.

Vi har valt att använda ”Extra” rutan för att ange vad för färger man vill använda. Man kan ange red, green, blue och få motsvarande färger, annars används en blå-grön färgskala. Vi anpassade iterationerna efter färgskalorna vi använde, och på det viset undgick vi rotberäkningar i programmet. Vi fann att en färgskala med blandningar av grönt och blått gav ett tydligt och vackert färgspektrum.

4 Programlistor

Klasserna finns i filer med samma namn som klasserna, till exempel finns klassen `Generator` i filen `Generator.java`.

4.1 `Mandelbrot`

```
import se.lth.cs.ptdc.fractal.MandelbrotGUI;

/** Madenlbrot innehåller main-metoden */
public class Mandelbrot {
    /**
     * Skapar de permanenta objekten MandelbrotGUI och Generator. Skriver ut
     * instruktioner i konsolfönstret för "extra"-rutan, inväntar sedan att man
     * ska interagera med MandelbrotGUI fönstret. Medlar mellan Generator och
     * MandelbrotGUI när man använder MandelbrotGUI.
     */
    public static void main(String[] args) {
        MandelbrotGUI mGUI = new MandelbrotGUI();
        Generator generator = new Generator();
        boolean zoomEnabled = false;
        System.out
                .println("När man använder färger så kan man " +
                        "skriva in följande i extra rutan:");
        System.out.println("'red' ger röd färg och 255 iterationer.");
        System.out.println("'green' ger grön färg och 255 iterationer.");
        System.out.println("'blue' ger blå färg och 255 iterationer.");
        System.out
                .println("Lämnar man rutan tom eller det står något annat så " +
                        "används en grön-blå färgskala och 1024 iterationer.");
        while (true) {
            switch (mGUI.getCommand()) {
            case MandelbrotGUI.RENDER:
                generator.render(mGUI);
                zoomEnabled = true;
                break;
            case MandelbrotGUI.RESET:
                mGUI.resetPlane();
                mGUI.clearPlane();
```

```
                zoomEnabled = false;
                break;
            case MandelbrotGUI.QUIT:
                System.exit(0);
                break;
            case MandelbrotGUI.ZOOM:
                if (zoomEnabled == true) {
                    generator.render(mGUI);
                }
                break;
            }

        }
    }

}
```

4.2 Generator

```
import java.awt.Color;
import se.lth.cs.ptdc.fractal.MandelbrotGUI;

/** Madenlbrot innehåller main-metoden */
public class Generator {
    Color[][] picture;
    Complex[][] complex;
    double minRe;
    double maxRe;
    double minIm;
    double maxIm;
    int width;
    int height;

    /**
     * Skapar matriserna som krävs och placerar ut dem i fönstret. Ritar en bild
     * i fönstret i användargränssnittet mGUI Kontrollerar upplösning och de val
     * av färg man har gjort.
     */
    public void render(MandelbrotGUI mGUI) {
        mGUI.disableInput();
        complex = mesh(mGUI.getMinimumReal(), mGUI.getMaximumReal(),
                mGUI.getMinimumImag(), mGUI.getMaximumImag(), mGUI.getWidth(),
                mGUI.getHeight());
        Color[] cScale = colorScale();
        Color[] rScale = redScale();
        Color[] bScale = blueScale();
        Color[] gScale = greenScale();
        int r = 0;
        switch (mGUI.getResolution()) {
        case MandelbrotGUI.RESOLUTION_VERY_LOW:
            r += 9;
            break;
        case MandelbrotGUI.RESOLUTION_LOW:
```

```
    r += 7;
    break;
case MandelbrotGUI.RESOLUTION_MEDIUM:
    r += 5;
    break;
case MandelbrotGUI.RESOLUTION_HIGH:
    r += 3;
    break;
case MandelbrotGUI.RESOLUTION_VERY_HIGH:
    r += 1;
    break;
}
int MAX_ITER = 1024;
if (mGUI.getExtraText().equals("blue")) {
    MAX_ITER = 255;
} else if (mGUI.getExtraText().equals("red")) {
    MAX_ITER = 255;
} else if (mGUI.getExtraText().equals("green")) {
    MAX_ITER = 255;
} else {
    MAX_ITER = 1024;
}
Color[][] picture = new Color[mGUI.getHeight() / r][mGUI.getWidth() / r];
for (int j = 0; j < mGUI.getHeight() / r; j++) {
    for (int i = 0; i < mGUI.getWidth() / r; i++) {
        Complex c = complex[i * r + r / 2][j * r + r / 2];

        int k = 0;
        Complex z0 = new Complex(0, 0);
        do {
            k++;
            z0.mul(z0);
            z0.add(c);
        } while (k < MAX_ITER && z0.getAbs2() <= 2 * 2);
        if (k < MAX_ITER) {
            switch (mGUI.getMode()) {
            case MandelbrotGUI.MODE_BW:
                picture[j][i] = new Color(255, 255, 255);
                break;
            case MandelbrotGUI.MODE_COLOR:
                if (mGUI.getExtraText().equals("blue")) {
                    picture[j][i] = bScale[k];
                } else if (mGUI.getExtraText().equals("red")) {
                    picture[j][i] = rScale[k];
                } else if (mGUI.getExtraText().equals("green")) {
                    picture[j][i] = gScale[k];
                } else {
                    picture[j][i] = cScale[k];
                }
                break;
            }
        } else {
            picture[j][i] = new Color(0, 0, 0);
        }
```

```
        }
    }
    mGUI.putData(picture, r, r);
    mGUI.enableInput();
}

/** Skapar en vektor med ett blå-grönt färgspektrum. */
private Color[] colorScale() {
    Color[] divergeColor = new Color[1090];
    double ColorLength = 32;
    int allColors = 0;
    for (int m = (int) ColorLength; m >= 0; m--) {
        for (int n = (int) ColorLength; n >= 0; n--) {
            allColors++;
            divergeColor[allColors] = new Color(0, n * (255 / 32), m
                    * (255 / 32));
        }
    }
    return divergeColor;
}

/** Skapar en vektor med ett rött färgspektrum. */
private Color[] redScale() {
    Color[] divergeRed = new Color[255];
    for (int red = 0; red < 255; red++) {
        divergeRed[red] = new Color(red, 0, 0);
    }
    return divergeRed;
}

/** Skapar en vektor med ett blått färgspektrum. */
private Color[] blueScale() {
    Color[] divergeBlue = new Color[255];
    for (int blue = 0; blue < 255; blue++) {
        divergeBlue[blue] = new Color(0, 0, blue);
    }
    return divergeBlue;
}

/** Skapar en vektor med ett grönt färgspektrum. */
private Color[] greenScale() {
    Color[] divergeGreen = new Color[255];
    for (int green = 0; green < 255; green++) {
        divergeGreen[green] = new Color(0, green, 0);
    }
    return divergeGreen;
}

/**
 * Skapar en matris där varje element är ett komplext tal som har rätt
 * koordinater
 */
private Complex[][] mesh(double minRe, double maxRe, double minIm,
        double maxIm, int width, int height) {
```

```
        Complex[][] complex = new Complex[width][height];
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                complex[j][i] = new Complex(
                        (double) minRe + ((double) j / (double) (width - 1))
                                * (maxRe - minRe), maxIm
                                - ((double) i / (height - 1)) * (maxIm - minIm));
            }
        }
        return complex;
    }
}
```

4.3 Complex

```
public class Complex {
    public double re;
    public double im;

    /** Skapar en komplex variabel med realdelen re och imaginärdelen im */
    Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }

    /** Tar reda på realdelen */
    double getRe() {
        return re;
    }
    /** Tar reda på imaginärdelen */
    double getIm() {
        return im;
    }

    /** Tar reda på talets absolutbelopp i kvadrat */
    double getAbs2() {
        return (re*re + im*im);
    }

    /** Adderar det komplexa talet c till detta tal */
    void add(Complex c) {
        re = re + c.getRe();
        im = im + c.getIm();
    }

    /** Multiplicerar detta tal med det komplexa talet c */
    void mul(Complex c) {
        double temp_re = (re * c.getRe()) - (im * c.getIm());
```

```
            im = (im * c.getRe()) + (re * c.getIm());
            re = temp_re;
        }
    }
```