University of St Andrews

Practical 4

Mandelbrot Set Explorer

Student ID: 120022067

University of St Andrews

CS5001 Object Oriented Modelling Design & Programming

Contents

Contents **2**

1 Introduction **3**

2 Design **3**

2.1 GUI Design 3
2.2 Program Infrastructure 4
2.3 Basic Display and Different Color Mappings 5
2.4 Zooming and Zooming Animation 6
2.5 Modifying Max iterations 8
2.6 Undo 9
2.7 Redo 9
2.8 Reset 10

3 Examples and Testing **11**

3.1 Basic Requirements 11
3.2 Enhancements 13

4 Evaluation **14**

5 Running **15**

Bibliography **16**

1 Introduction

According to Lewis (2017), Mandelbrot set is "the set of complex numbers C for which iterative application of the equation $Z_{n-1} = Z_n^2 + C$. with Z starting at 0 remains bounded within a certain distance from the origin 0 in the complex plane". The goal of this practical is to design a GUI-driven Fractual generator that allows the user to view and explore the Mandelbrot set graphically.

2 Design

2.1 GUI Design

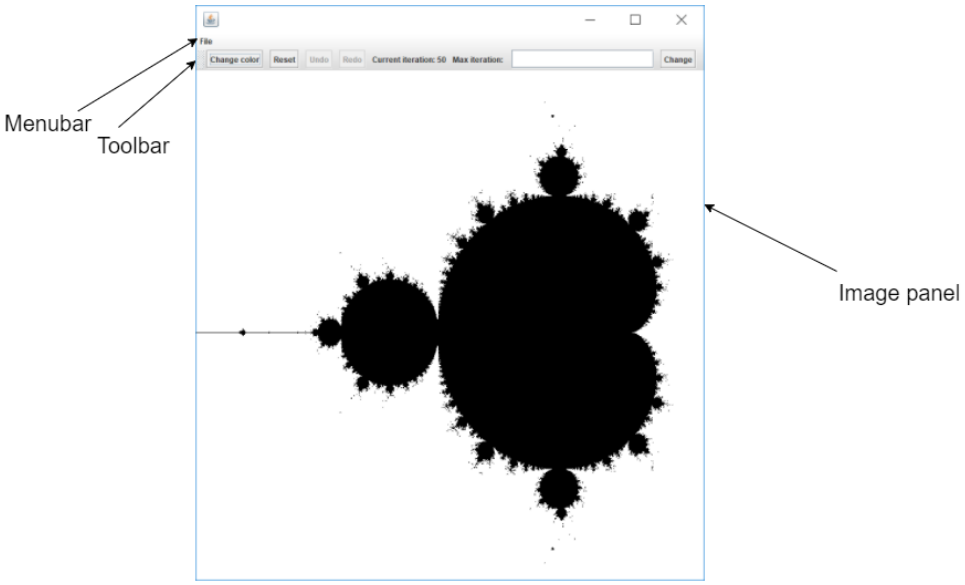

Figure 1: GUI Mandelbrot set explorer.

2.2 Program Infrastructure

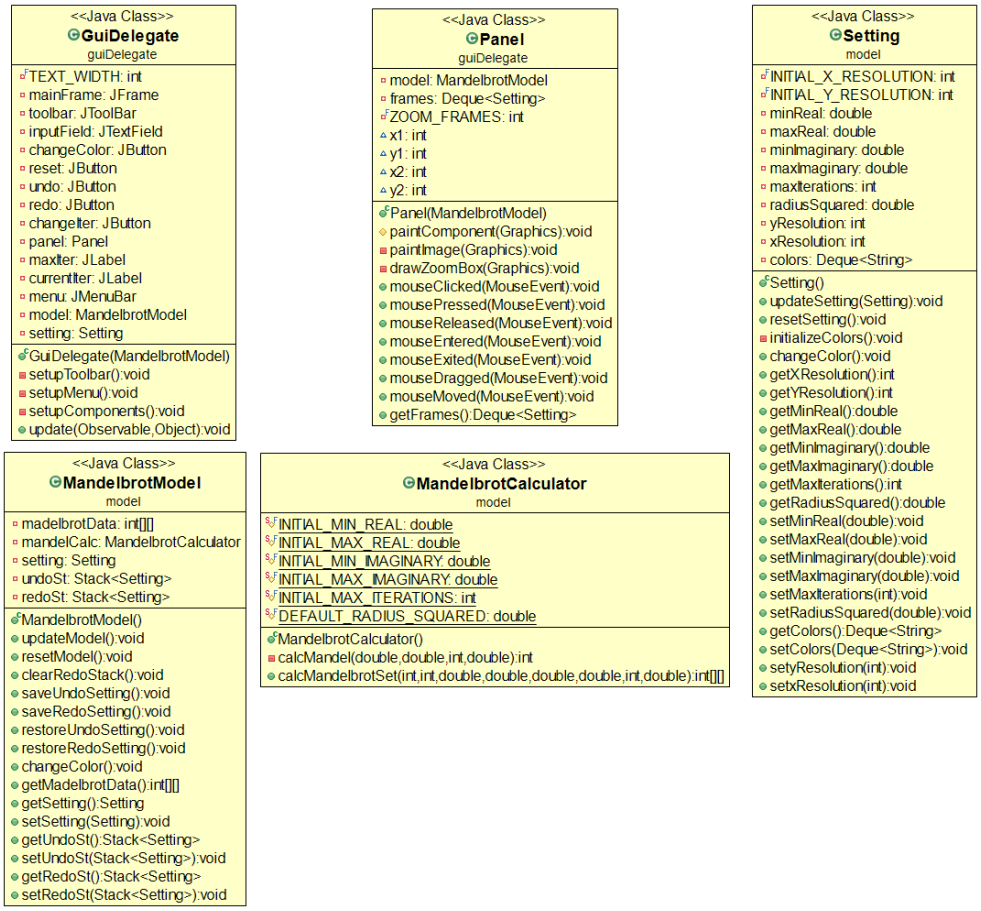

Figure 2: Class diagram of the program.

The overall structure of the program is illustrated in Figure 2. This implementation of Mandelbrot set explorer adopts Model Delegate pattern. The model extends the Observable class and is observed by the Delegate class. This allows the View to be updated when the model has changed. The view update method is shown in Listing 1.

```
public void update(Observable o, Object arg) {
    // Tell the SwingUtilities thread to update the GUI components.
    SwingUtilities.invokeLater(new Runnable(){
      public void run(){
        if (!model.getUndoSt().isEmpty()) {
          undo.setEnabled(true);
        } else {
          undo.setEnabled(false);
        }
        if (!model.getRedoSt().isEmpty()) {
          redo.setEnabled(true);
        } else {
          redo.setEnabled(false);
        }
        currentIter.setText("Current iteration: " + setting.getMaxIterations
    ());
        inputField.setText("");
        panel.repaint();
        Deque<Setting> frames = panel.getFrames();
        if (!frames.isEmpty()) {
          model.getSetting().updateSetting(frames.remove());
          model.updateModel();
        }
      }
    });
  }
```

Listing 1: The code used to update the View.

2.3 Basic Display and Different Color Mappings

The first functional requirement is to display the Mandelbrot set in black and white for a sensible initial parameter setting. Different color mappings is also implemented. The code used to achieve these functionalities is shown in Listing 2

```
private void paintImage(Graphics g) {
    int[][] madelbrotData = model.getMadelbrotData();
        int maxIterations = model.getSetting().getMaxIterations();
        for (int i = 0; i < madelbrotData.length; i++) {
          for (int j = 0; j < madelbrotData[i].length; j++) {
            if (madelbrotData[i][j] >= maxIterations) {
              g.setColor(Color.BLACK);
                } else {
                  Deque<String> colors = model.getSetting().getColors();
                  String currentColor = colors.peek();
                  float colorValue = (float) (madelbrotData[i][j] * 1.0/
   maxIterations);
                  if (currentColor.equals("red")) {
                    g.setColor(new Color(colorValue, 0, 0));
                  } else if (currentColor.equals("green")) {
                    g.setColor(new Color(0, colorValue, 0));
                  } else if (currentColor.equals("blue")) {
```

```
                    g.setColor(new Color(0, 0, colorValue));
                } else {
                    g.setColor(Color.WHITE);
                }
            }
            g.drawLine(j, i, j, i);
            }
        }
    }
```

Listing 2: The code used to create basic display and different color mappings of the Mandelbrot set.

2.4 Zooming and Zooming Animation

The program allows the user zoom in by selecting a square or a rectangle on the image which will then be used as the bounds for a new calculation of the Mandelbrot set. When the user drag the mouse over the image, a square or rectangle will appear, highlighting the zooming area. The code used to display the shape can be found in Listing 3.

```
private void drawZoomBox(Graphics g) {
    Deque<String> colors = model.getSetting().getColors();
        String currentColor = colors.peek();
        if (currentColor.equals("red")) {
        g.setColor(Color.RED);
      } else if (currentColor.equals("green")) {
        g.setColor(Color.GREEN);
      } else if (currentColor.equals("blue")) {
        g.setColor(Color.BLUE);
      } else {
        g.setColor(Color.BLACK);
      }
        int width = Math.abs(x2 - x1);
      int height = Math.abs(y2 - y1);
        // drag southeast
        if (x2 > x1 && y2 > y1) {
          g.drawRect(x1, y1, width, height);
        }
        // drag northeast
        if (x2 > x1 && y2 < y1) {
          g.drawRect(x1, y2, width, height);
        }
        // drag southwest
        if (x2 < x1 && y2 > y1) {
          g.drawRect(x2, y1, width, height);
        }
        // drag northwest
        if (x2 < x1 && y2 < y1) {
          g.drawRect(x2, y2, width, height);
        }
  }
```

Listing 3: The code used to create display shape which highlights the zooming area.

When the user releases the mouse, the GUI allows the user to view zooming animation before effectively zooming in on the image. The code used to calculate new parameter setting for the Mandelbrot set is shown in Listing 4.

```
    Setting currentSetting = new Setting();
    currentSetting.updateSetting(model.getSetting());
    double minReal = currentSetting.getMinReal();
    double maxReal = currentSetting.getMaxReal();
    double minImaginary = currentSetting.getMinImaginary();
    double maxImaginary = currentSetting.getMaxImaginary();
    int xResolution = currentSetting.getXResolution();
    int yResolution = currentSetting.getYResolution();
    double rangeReal = maxReal - minReal;
    double rangeImaginary = maxImaginary - minImaginary;
    int newX1, newX2, newY1, newY2;
    if (x2 > x1) {
      newX1 = x1;
      newX2 = x2;
    } else {
      newX1 = x2;
      newX2 = x1;
    }
    if (y2 > y1) {
      newY1 = y1;
      newY2 = y2;
    } else {
      newY1 = y2;
      newY2 = y1;
    }
    double scale;
    double ratioMinReal = newX1 / (double) xResolution;
    double ratioMaxReal = newX2 / (double) xResolution;
    double ratioMinImaginary = newY1 / (double) yResolution;
    double ratioMaxImaginary = newY2 / (double) yResolution;
    double newMinReal = minReal + (ratioMinReal * rangeReal);
    double newMaxReal = minReal + (ratioMaxReal * rangeReal);
    double newMinImaginary =  minImaginary + (ratioMinImaginary *
  rangeImaginary);
    double newMaxImaginary = minImaginary + (ratioMaxImaginary * rangeReal);
```

Listing 4: The code used to calculate new parameter setting for the Mandelbrot set.

Once the program has the new parameter setting, it creates ten frames of animation and store them in a queue. The code used to achive this functionality can be found in Listing 5.

```
    double rangeMinReal = newMinReal - minReal;
    double rangeMaxReal = maxReal - newMaxReal;
    double rangeMinImaginary = newMinImaginary - minImaginary;
    double rangeMaxImaginary = maxImaginary - newMaxImaginary;
    x1 = -1;
    x2 = -1;
    y1 = -1;
    y2 = -1;
    for (int i = 1; i <= ZOOM_FRAMES; i++) {
      scale = (double) i / (double) ZOOM_FRAMES;
      Setting animationSetting = new Setting();
      animationSetting.setMinReal(minReal + (scale * rangeMinReal));
      animationSetting.setMaxReal(maxReal - (scale * rangeMaxReal));
      animationSetting.setMinImaginary(minImaginary + (scale *
  rangeMinImaginary));
      animationSetting.setMaxImaginary(maxImaginary - (scale *
  rangeMaxImaginary));
      animationSetting.setMaxIterations(currentSetting.getMaxIterations());
      animationSetting.setColors(currentSetting.getColors());
      frames.add(animationSetting);
    }
    model.saveUndoSetting();
    model.clearRedoStack();
    model.getSetting().updateSetting(frames.remove());
    model.updateModel();
```

Listing 5: The code used to create zoom animation.

2.5 Modifying Max iterations

The user can change the value of max iterations by putting new value into the JTextField (see Figure 3). To apply the change, the user needs to press Enter or click Change button next to JTextField. The code that is used to update the value of max iterations is shown in Listing 6.

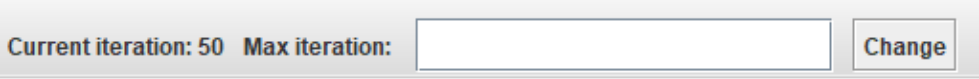

Figure 3: JTextField used to input new max iterations value.

```
    changeIter = new JButton("Change");
    changeIter.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
        model.saveUndoSetting();
        setting.setMaxIterations(Integer.parseInt(inputField.getText()));
        model.clearRedoStack();
        model.updateModel();
      }
    });

        inputField.addKeyListener(new KeyListener(){
      public void keyPressed(KeyEvent e) {
        if(e.getKeyCode() == KeyEvent.VK_ENTER){
          model.saveUndoSetting();
          setting.setMaxIterations(Integer.parseInt(inputField.getText()));
          model.clearRedoStack();
          model.updateModel();
        }
      }
      public void keyReleased(KeyEvent e) {
      }
      public void keyTyped(KeyEvent e) {
      }
    });
```

Listing 6: The code used to change max iterations.

2.6 Undo

Stack data structure is used to store previous settings. When Undo button is pressed, the current setting is saved into Redo Stack. Then, the setting at the top of the Undo Stack is assigned as current setting. The code used to Undo previous action is shown in Listing 7.

```
    undo = new JButton("Undo");
    undo.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
        model.saveRedoSetting();
        model.restoreUndoSetting();
        model.updateModel();
      }
    });
```

Listing 7: The code used to undo previous operation.

2.7 Redo

Stack data structure is also used to store the settings. When Redo button is pressed, the current setting is saved into Undo Stack. Then, the setting at the top of the Redo Stack is assigned as current setting. The code used to Redo previous action is shown in Listing 8.

```
    redo = new JButton("Redo");
    redo.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
        model.saveUndoSetting();
        model.restoreRedoSetting();
        model.updateModel();
      }
    });
```

Listing 8: The code used to redo previous operation.

2.8 Reset

When the user clicks on the Reset button, parameter setting is reverted back to initial setting. The code used to achieve this can be found in Listing.

```
   public void resetModel() {
        undoSt.clear();
        redoSt.clear();
        setting.resetSetting();
        madelbrotData = mandelCalc.calcMandelbrotSet(800,
                800,
                MandelbrotCalculator.INITIAL_MIN_REAL,
                MandelbrotCalculator.INITIAL_MAX_REAL,
                MandelbrotCalculator.INITIAL_MIN_IMAGINARY,
                MandelbrotCalculator.INITIAL_MAX_IMAGINARY,
                MandelbrotCalculator.INITIAL_MAX_ITERATIONS,
                MandelbrotCalculator.DEFAULT_RADIUS_SQUARED);
        setChanged();
        notifyObservers();
  }
```

Listing 9: The code used to restore default setting.

3 Examples and Testing

3.1 Basic Requirements

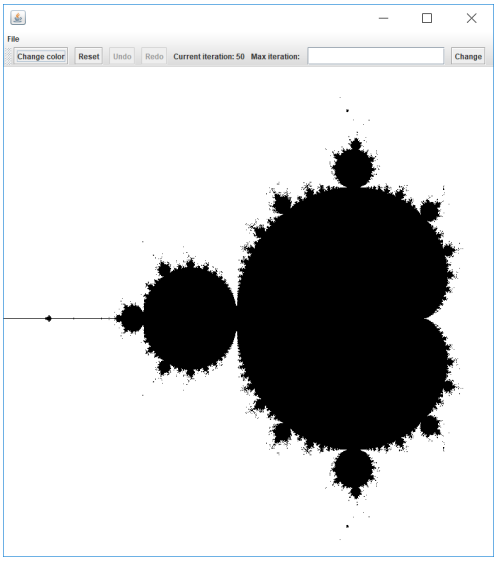

Figure 4: The program displays the Mandelbrot set in black and white.

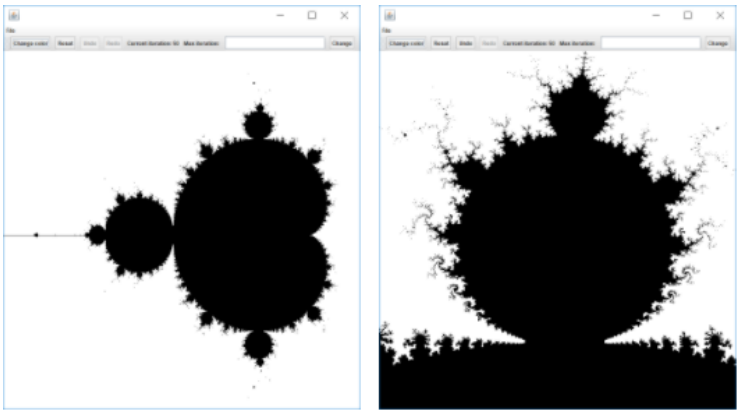

Figure 5: **Left:** default view of the Mandelbrot. **Right:** Mandelbrot set zoomed in.

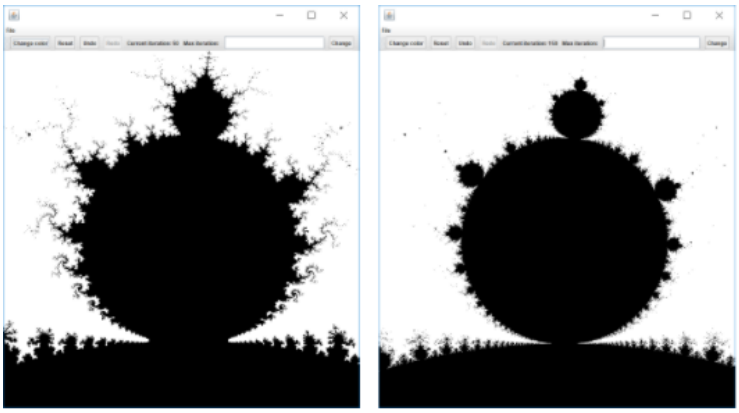

Figure 6: Left: the Mandelbrot with 50 max iterations. **Right:** the Mandelbrot set with 150 max iterations

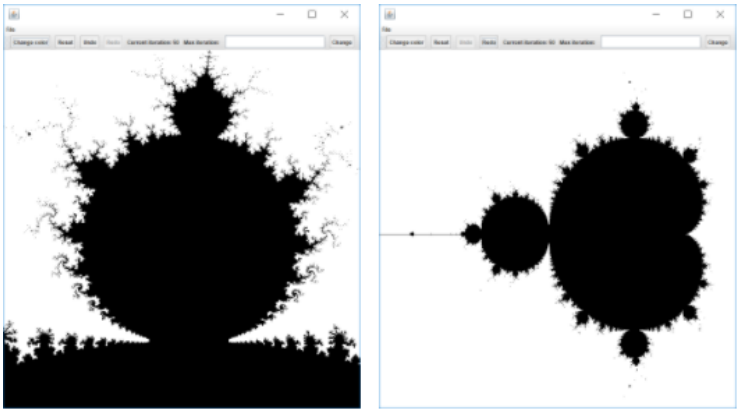

Figure 7: **Left:** the Mandelbrot set zoomed in. **Right:** Mandelbrot set after undo button was clicked.

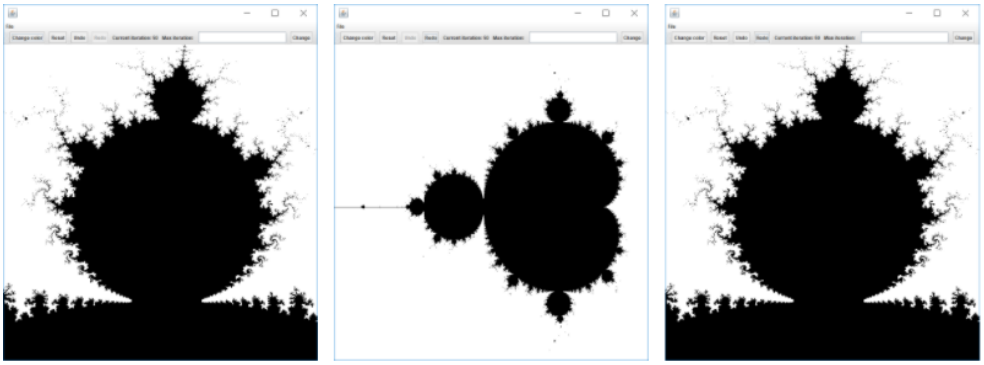

Figure 8: **Left:** the Mandelbrot set zoomed in. **Middle:** Mandelbrot set after undo button was clicked. **Right:** Mandelbrot set after redo button was clicked.

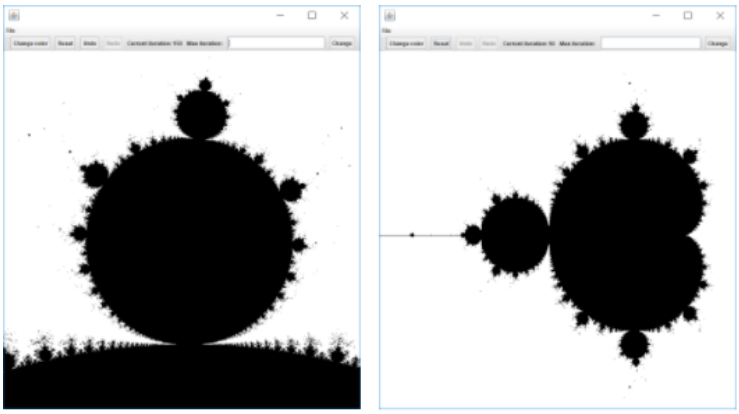

Figure 9: **Left:** zoomed in image of the Mandelbrot set with modification to max iterations. **Right:** Mandelbrot set after reset button was clicked

All functionalities stated in the basic requirements are demonstrated in Figure 4, Figure 5, Figure 6, Figure 7, Figure 8, and Figure 9.

3.2 Enhancements

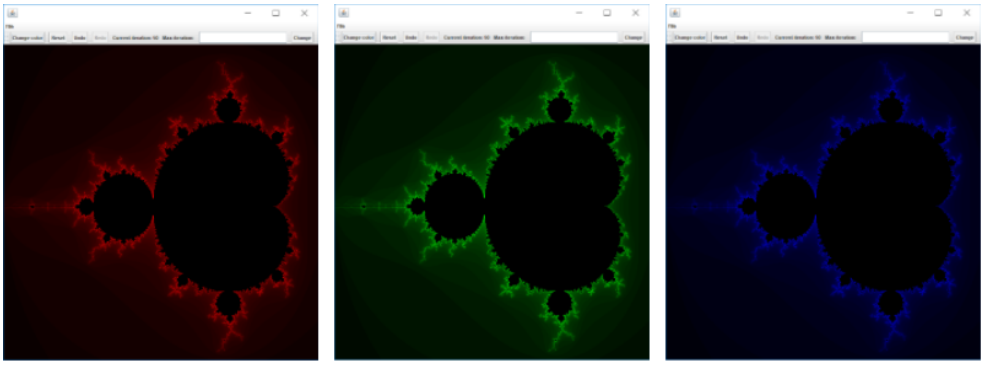

Figure 10: Screenshots demonstrating different color mappings.

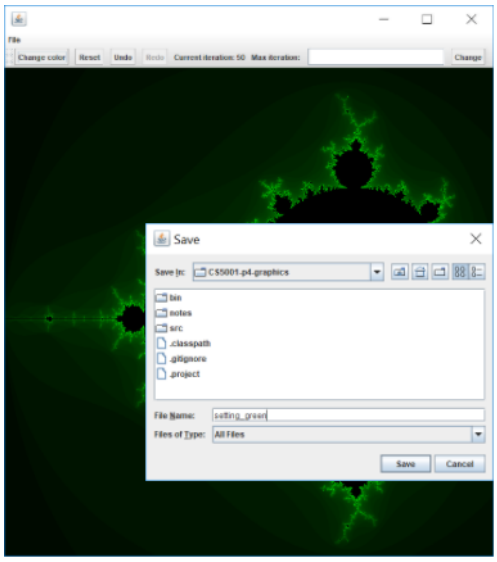

Figure 11: Screenshot demonstrating save functionality.

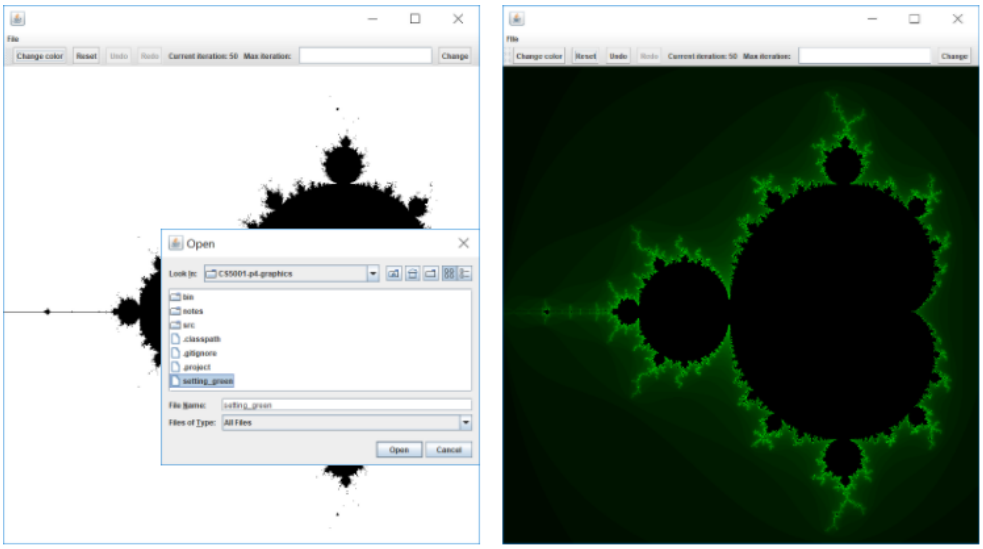

Figure 12: **Left:** open file dialog appear when the user clicks load. **Right:** saved image reloaded. The user can continue exploring the Mandelbrot from that position onward.

All enhancements implemented are demonstrated in Figure 10, Figure 11, and Figure 12.

4 Evaluation

Overall, the program manages to satisfy all basic requirements and two enhancements. The enhancements implemented are different color mappings, zoom animation, and save/load parameter settings and computed image. Due to time constraints, it is not possible to explore additional, interesting operations or color mappings.

5 Running

To run the program from the terminal, the user needs to navigate to bin folder inside the project folder. Then, use the command in Listing 10.

```
java main/Main
```

Listing 10: Command for running the program

Bibliography

Lewis, J. (2017), 'Practical 4 – mandelbrot set explorer', https://studres.cs.st-andrews.ac.uk/CS5001/Practicals/p4-graphics/CS5001-P4.pdff. Accessed: 2017-11-23.