CS5001 Practical 4 – Graphics: Mandelbrot Set Explorer

Report and User Instruction

170024030

User Instruction:

To compile this program:

In the Linux terminal, open /CS5001-p4-graphics/src, inputw

javac Main.java

To run this program:

In the same directory input

Java Main

Implemented Basic Requirements and Extensions

Basic Display,

Zoom in on the image with a selected square,

Permit the user to alter the maxIteration value,

Undo, Redo and Reset.

Four colour mapping scheme besides black and white: blue, red, green and rainbow,

Save and load parameter settings,

Save images,

Using mouse wheel to zoom in and zoom out.

Design

The code is designed following the Model-Delegate Paradigm. The controller and view are merged into the UI Delegate component.

Model component:

The Model component is placed in the "Model" package which contains three classes:

1. "MandelbrotCalculator" (Supplied)
   This class implements a "calcMandelbrotSet" method that generate the Mandelbrot set for given input parameters.

2. "MandelbrotParam"

This class is contains all the parameters that are used to calculate and generate the Mandelbrot set, including the four bounds of real and imaginary parts, the cut-off limit, and the colour scheme.

All of the member variables are set to be private, and the getters are provided by methods. The constructor of "MandelbrotParam" takes in all six parameters. Additionally, "getNextColor" method gives the colour scheme in a circle sequence for changing colour mapping scheme.

It also implements the Serializable interface to allow this object to be saved and loaded as ".ser" files.

3. "ModelCalculator"

   This class is the main body of the model component, providing methods to allow the UIDelegate component to manipulate the model. It contains an "ArrayList" that stores all the states in the form of "MandelbrotParam" and adds a new state every time a change was made by the "UIDelegate" component to implement the "Redo" and "Undo" functions. The member variable "stateIndex" is used to indicate which set of parameters in the list indicates the current state.

   The image is stored in this class as a member variable. The "updateImage" method alters the image according to the changes made by the "UIDelegate" component. The "UIDelegate" component invokes this method before it draws the image. Different colour mappings are implemented in the "calculateColor" method which has five pre-set schemes corresponding to the "getNextColor" method in "MandelParam" class.

   Method to alter the parameters are provided by "setNextState", which is overridden to take in a different combination of argument. This method will add another "MandelbrotParam" object to the state list. The new parameter set is initialized with the provided arguments; the deficient arguments will be the same as the former state.

   Several other methods are implemented for saving and loading parameters, and saving images. The file name is in accordance with the data and time.

UIDelegate Component:

The UIDelegate component is placed in the "Model" package which contains two classes:

1. "Square"

   In order not to deform a rectangular area into our square-shaped window, the user can only select and zoom-in a square area. Therefore, this class is designed to "translate" the rectangular indicated by the mouse location to a square.

   This class contains the four bounds of the square and the side. However, these four bounds are not the same as the bounds in "MandelbrotSetClaculator". "xMin" and "yMin" are the coordinates of the upper-left point of the square; "xMax" and "yMax" are the coordinates of the lower-right point. The side length also has the units in pixel.

   The constructor takes in four arguments: the x and y coordinates of the first point and the last point during mouse dragging. Then, it generates a square that starts at the first point, points towards the second point and has the side length of the longer projected distance (Fig. 1).

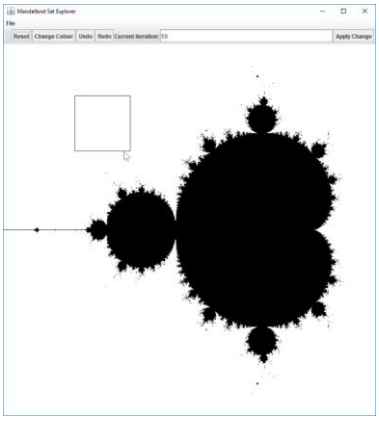

Figure 1. An example of square generated from the constructor of Square object, by dragging mouse on the frame.

2. "MandelbrotSetExplorer"

   This class is the main body of the "UIDelegate" component that extends "JComponent". It contains a "ModelCalculator" class initialized with the width and height default settings. Both interface and controller are merged in this class – the controller is built inside the action listeners of buttons, menus and mouse action.

   The paint method not only paints the image for current state provided by the model, it also paints a square box generated by dragging the mouse to indicate the specific area to zoom in.

   The UI is set up with menus, buttons and mouse listeners to allow the user to interact with the model.

   Limited access and means to alter the model parameters are provided according to the three overridden "getNextState" method in the model.

Demonstration and Discussion

Zoom by selecting a square:

The UIDelegate component allows the user to grab a square area and invoke the corresponding method in the model component to update the parameters and the image, then the image is repainted.

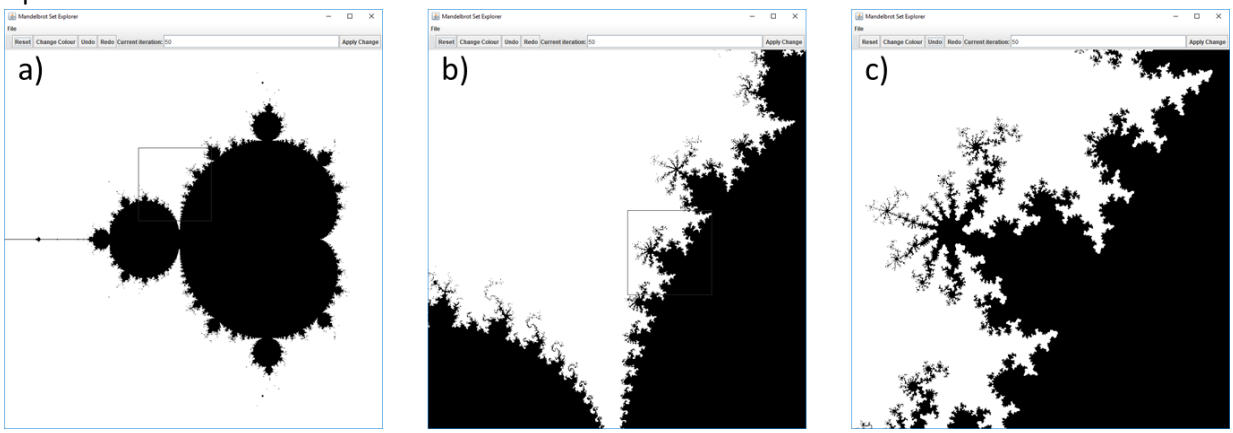

Figure 2. A demonstration of zoom-in operation by selecting a square are of the frame. Three stages of zoom-in and selecting area are shown: a) the initial Mandelbrot Set image, b) the image zoomed in to the square area in a), c) the subsequent third stage.

Altering the maxIterations:

Users can change the maxIteration parameter by update the value in the text field next to “Redo” button. After adjusting the value, the change can be applied to the image by either hitting “Enter” key or click on the “Apply Change” button.

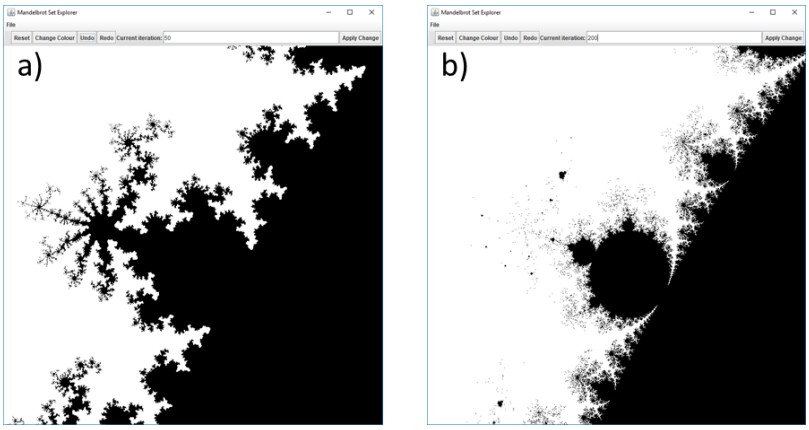

Figure 3. A demonstration of changing the maxIteration parameter. The area is the same as Fig 2.c. a) maxIteration = 50. b) maxIteration = 200.

Undo, Redo and Reset:

The buttons for these operations are revealed in the above figures. All sets of state parameters are saved in a list in the model, and the state where we are looking at is indicated by an index number.

Undo will reduce the index by one; Redo will increase the index by one; Reset will re-initialize the list and set the index to be zero.

A tricky part of this is to make changes after “Undo” operation. I used a sentinel Boolean flag to record whether an Undo operation was done before the current operation. If there was a previous “Undo” operation, the model will apply a method to remove all the saved stages after current state and start writing new ones.

The text field will also be updated along with the image when these three operations are applied.

Different Colour Mappings:

For the “blue”, “red”, and “green” colour mapping schemes, I adjust the brightness component in the HSB colour system. However, I failed to get the “white lining” around the black spots with HSB colour – the colour should be totally white when the brightness reaches 1, but it does not apply to the Java HSB system.

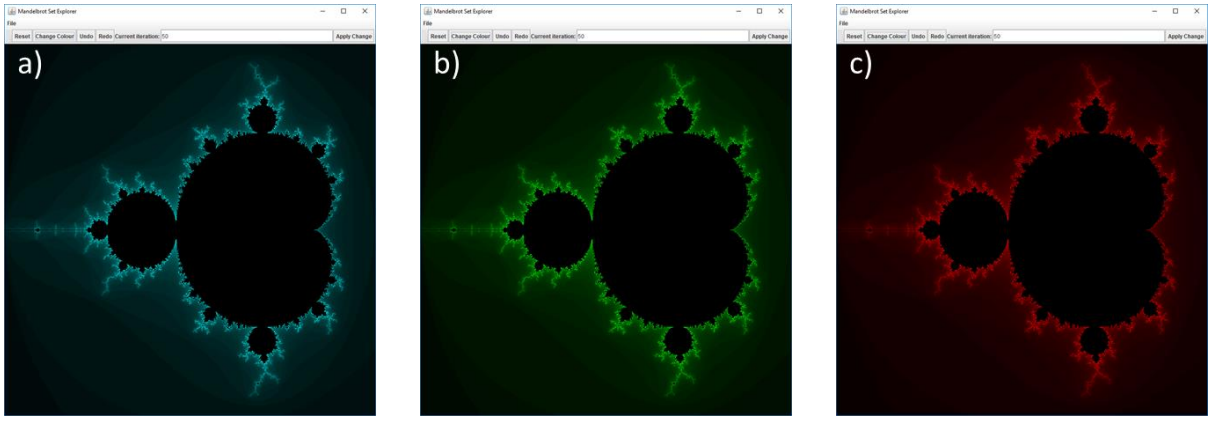

Figure 4. A demonstration of different colour mappings implemented in this project: a) blue, b) green, and c) red.

The “rainbow” colour mapping scheme is done by adjusting the hue component of the HSB colour. We can see that in Fig 5.b this mapping scheme shows more evident colour to indicate which group of pixels reached each iteration number.

By clicking on the “Change Colour” the user can change the colour mapping scheme in a rotational order set in the MandelbrotParam class: “Black and White”, “Blue”, “Red”, “Green”, “Rainbow”, and back to the first one.

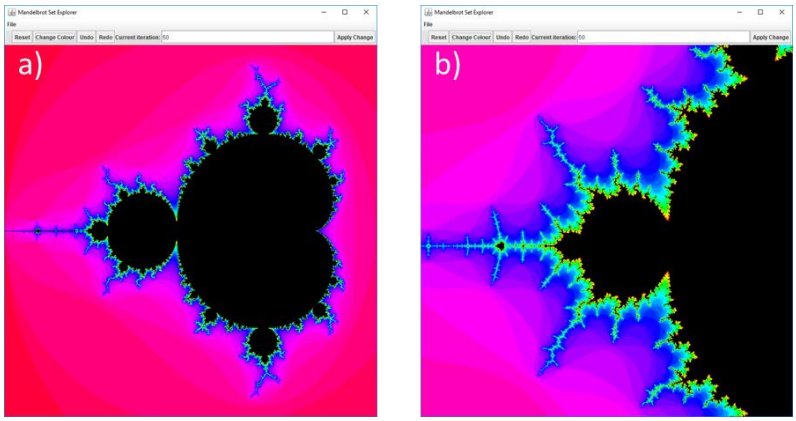

Figure 5. The “rainbow” colour scheme: a) original size and b) zoomed in view.

Save, Load and Save Image:

The file menu contains the access to these three operations. It is worth noticing that it only performs well on Linux; on Window 10 it does not really work, and the file name saved by Linux machine will be messy on Windows.

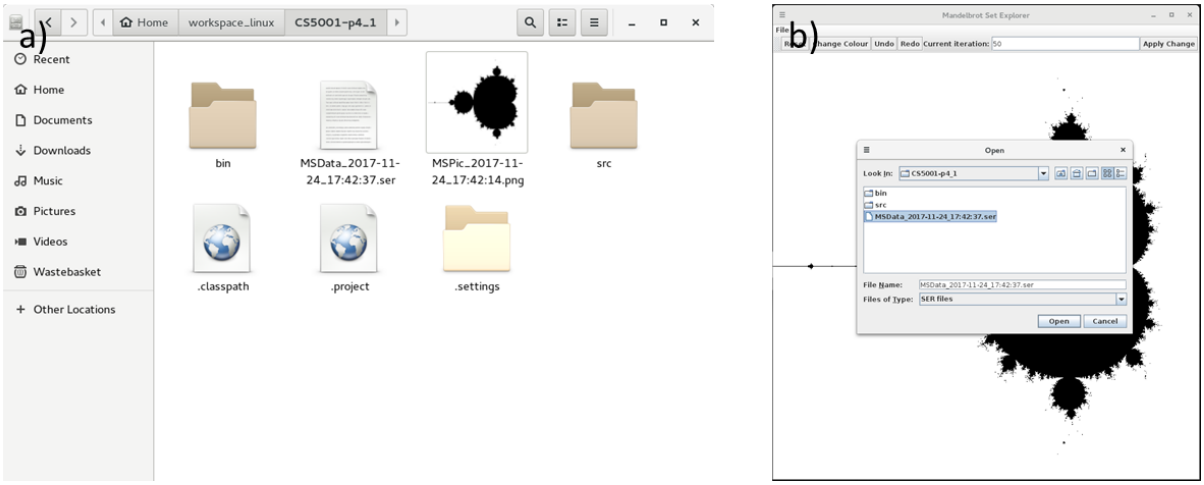

Figure 6. a) The saved image and parameter file. b) The file chooser used to select the parameters to load.

Zoom in and out by Rotating the Mouse Wheel

Mouse wheel listener is also added to the main frame to allow user zoom in and out a little bit towards the location of the mouse. The zoom rate of the wheel is set by a final number in the UIDelegate component. This interaction method and the selecting method are complementary to each other. The animation of zooming in and out can also be realized by continuously rotating the mouse wheel.