Jordan Rowley
Computer Science
University of St. Andrews

Mini-Project 2 Report

Mandelbrot Set Explorer in Java

My Mandelbrot Set explorer program uses the Java package Swing to render a pixel image of the Mandelbrot Set which can be navigated. Initially, the set is drawn in black and white with parameter settings which shows the main body of the set. The user can zoom in and out and pan around using buttons at the top of the frame. The buttons will zoom and pan a certain amount dependent on the current zoom so that the proportional zoom/pan is constant. A checkbox at the top can be checked to display an estimate of the zoom magnification on top of the Mandelbrot set. There is also an undo button which will keep track of the last operation performed and perform the opposite operation when clicked. My algorithm for this does not make use of a stack but rather a flag that gets updated each time an operation is performed, for this reason, when undo is clicked twice, the second time will perform a 'redo'. A reset button is also included which will redraw the Mandelbrot set with the initial parameters. Another checkbox is presented which the user can check to show the set in colour where the hue is directly proportional to the number of iterations performed at each point. At the bottom of the frame, the current maximum iterations is shown and the user can use two buttons to increment or decrement this value. There are also 'Save' and 'Load' buttons which saves current parameters to a properties file so that the user can find a previously visited part of the set by saving it and then clicking load from elsewhere in the set. My program meets all the basic requirements plus all the extension requirements bar producing zooming animations defined by the user.

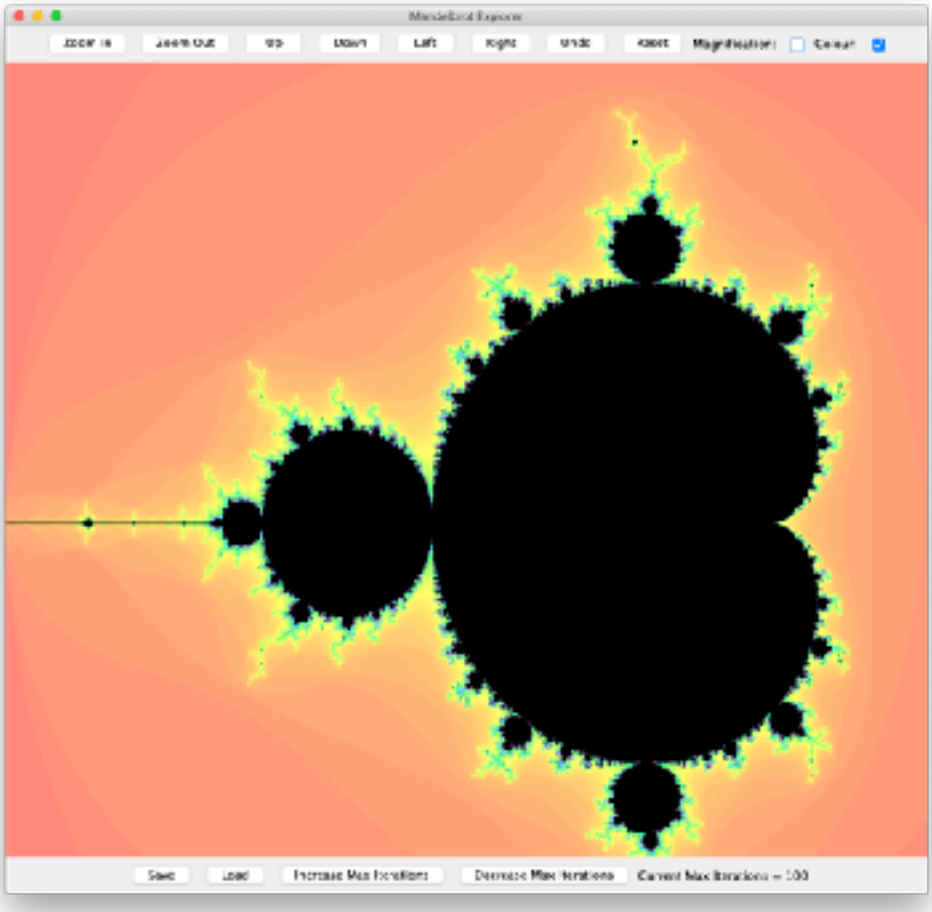
The program consists of four classes: MandelbrotExplorer, Display, ExploreFunctions, and the provided MandelbrotCalculator class. The first of which, builds the frame and determines the placement of two panels which will house JButton's. The Display class contains the body of the program; here, the JButtons and JLabels are initialised and placed into the correct panels. Additionally, each button is given an actionListener which will call a

function from the ExploreFunctions class. Lastly, Display has the paint method which is called automatically by the program and will draw the Mandelbrot Set by calling the MandelbrotCalculator class to get a two-dimensional array of points and the number of iterations it took for the complex value to tend to infinity; each point that reached the maximum number of iterations will be plotted as part of the set. Each point is checked to see if it reached maximum iterations using a nested for loop and an if statement which, if satisfied, will tell the program to draw a line from that pixel to that same pixel; essentially drawing a single pixel. The ExploreFunctions class holds the methods that the buttons will call and a method that keeps track of how much to zoom/pan at each magnification; the step. This is done separately for the real and imaginary component by taking the absolute value of the range between the max value and min value and dividing it by the number of pixels in the x and y direction. This is then multiplied by a defined multiplier which just helped to determine how much to zoom each time the button is clicked.

The zoom method adds the step to the min real and imaginary values and subtracts the step from the maximum real and imaginary values then recalculates the Mandelbrot Set within the new parameters. Unzoom will execute the reverse functions. The up and down methods will decrement and increment both the minimum and maximum imaginary component respectively. Similarly, the left and right methods will decrement and increment both the minimum and maximum real component. The maximum iteration control buttons will increase and decrease the maxIterations variable and update the label displaying the current value. The save method uses a file output stream to save properties to an external file which will later be called by the load function. Load uses a file input stream to read the properties previously saved and will redraw the Mandelbrot set using the parameters saved in the file; hence taking the user back to a previously visited point. After each of these are called from the Display class, the Mandelbrot set will be recalculated and repaint() will be called to draw over the current depiction of the set.

Although not shown in the UML diagram, every JButton has an actionListener which calls a method from the ExploreFunctions class and then calls mandelCalc from the MandelbrotExplorer class to redraw the set with new parameters. Additionally, none of the class attributes were made private so each class has access to the attributes from every other class. I did this to simplify updating variables located in classes other than the working class.

Once the colouring has been turned on, the result is rather appealing:



One downfall of my program is that it has to recalculate each point in the set (and not in the set) every time a zoom/pan is executed. This means that it runs slowly; especially when zoomed in at a high magnification with a high maximum iterations. I didn't make use of a BufferedImage which might've helped with this issue.

Here is an example of how to run the program from a terminal: