Practical 4 Report

System Design

For this system I choose to utilise the Model-Delegate Architecture, this allowed me to protect the class objects that I would need to manipulate through during operation of this application. Therefore this project includes two packages: Delegate and Model. The Model Package contains the source code for the Settings object (which enables undo/redo functionality) as well as the MandelbrotCalculator object. Inside the Delegate Package; we can see the Main class (which acts as the view class for this project), Controller and Configuration classes. In addition to the sample.fxml file which enables JavaFX to run, which was generated by IntelliJ upon construction of the JavaFX project. The Controller class will manipulate the Main class based on instructions received from the Main class.

I opted to use JavaFX instead of Java Swing, as I have already gained experience with Java Swing during my undergraduate. I found that the GUI we produced - through no fault of our own - was very rudimentary and looked dated. So JavaFX was selected for the sleek, (more) modern output that it would produce.

Additionally, the JavaFX window would be described by the Main class as a BorderPane, which allotted me greater control over where elements would be displayed without having to worry about coordinates (for the most part). There would need to be some level of micromanagement of the positioning of elements, for example the placement of the “Zoom Factor” text over the Canvas (discussed later). Through BorderPane I was able to quickly and effortlessly assign the Mandelbrot Set image to the center of the window and the VBox containing the Tool Bar for the project to the top of the window.

The choice to utilise the Canvas to draw the Mandelbrot Set Image came from numerous sources online which displayed source code for the Mandelbrot Set Image. I saw that many of them used Canvas for the .getGraphicsContext2D() method, allowing the developer total control over the Canvas area. The principle inspiration for this was this source code here: http://www.hameister.org/JavaFX_MandelbrotSet.html.

Implementation Description

The first step that was taken towards completing this project was to generate the Mandelbrot Set in black and white into a JavaFX window. Knowing that the MandelbrotCalculator's calcMandelbrotSet() method would return an Integer Array of Integer Arrays, I knew that I would have to iterate through each of the Arrays within the Arrays in order to extract the Mandelbrot numbers. Once that was done I was able to use the GraphicsContext from the Canvas to draw a 1px by 1px rectangle (effectively drawing a single pixel) at that given coordinate (if the Mandelbrot Set X and Y sizes matched the Canvas height and width). I set that application to draw a black pixel where the Mandelbrot number was equal to the maximum defined iteration. This generated the Mandelbrot Set Image.

After that I experimented with the colours, finding out how to use the value at each XY coordinate to manipulate RGB values. Eventually I discovered that if you have the GraphicsContext for the Canvas create pixels of the colour with a RGB value of 255/MandelbrotSet Number, 0, 0 up till half and then invert that formula with 255, 255/number, 255/number where number is greater than half of the maximum iterations then it would produce gradient colouring with a clear border.

The next step was to design, wireframe and implement the Tool Bar. This was done using a ToolBar inside a VBox (which BorderPane prefers over the ToolBar object). I created each button and text field in turn (leaving their functionality blank until it was time to program that section.

The first button I gave functionality to was the Update Iterations button. This button would grab the data from the Iterations TextField and translate that string to a number. That number would then overwrite the value held within the Controller's currentIterations class variable and re-render the Mandelbrot Set with the new parameter.

I then attempted the Zoom functionality (which caused me significant difficulties (see the difficulties Section)). The Zoom functionality dictates that should the Zoom button be toggled, the user can then draw a square onto the Canvas. This would then give the necessary X,Y parameters for the opposing corners of the square (on mouse release). Which would then be translated into new minReal, maxReal, minImaginary and maxImaginary parameters for the Mandelbrot Set, which would then be used to regenerate the Mandelbrot Set Image.

Next was Pan. Pan was far simpler and easier to implement than Zoom, because I was able to use the lessons learned from Zoom. The user can draw a straight line using their mouse across the Mandelbrot Image in any direction. The Mandelbrot Image would then move a distance equal to the length of the line in that direction. This was done by increasing (or decreasing) the minReal and maxReal parameters by the same amount, then applying the same logic to the min and max Imaginary parameters.

After this, the Change Colour button was given functionality. To keep it simple, I calculated every possible variation of where 255 could appear in the RGB set. There are 8 in total, subtracting away the combination which would provide a black screen, I was left with 7. So I made a switch statement inside the double for-loop that iterated through the Mandelbrot Set, which would dictate the colour of the pixel base on a single integer passed into the paintMandelbrot() method. When the Change Colour button is pressed, the integer iterates through the options.

The full undo and redo functionality was provided by on each mutation of the original Mandelbrot Set parameters they were saved into a new Setting Object that was then hosted in an ArrayList of Setting objects within the Controller. When the user has made some mutations to the Mandelbrot Set via the GUI they will be able to “dance” up and down the ArrayList of Setting Objects using the Undo and Redo buttons.

The reset button simply regenerates the Mandelbrot Set image using the default values defined within the MandelbrotCalculator class. It does not reset colour or the history of actions.

Finally, the zoom factor text was implemented to be superimposed on top of the Mandelbrot Image. This updates the zoom factor when a zoom is committed.

Usage

Normal

To use the Mandelbrot Set Explorer application, load the project into your favourite IDE and run the project, Then play around with the buttons provided at the top of the window. Drag your mouse over the image to zoom (the action by default). Or select the Pan RadioButton to then allow you to Drag your mouse across the screen to zoom. Note that you will have to release the button in order to trigger the Zoom/Pan.

Testing

Same as the Normal Usage, I encourage you to keep testing different events until they break. I acknowledge that this is not a particularly robust system, but the system should be able to handle the majority of events.

Difficulties Experienced

This project posed several unexpected challenges at me. The Zoom and Pan functionality provided the greatest challenges. Both working out the algorithms and drawing the shapes on top of the canvas were not simple tasks. They are functionally sound, in that they work, but they are not accurate to the movements of the mouse, particularly the line. In fact for reasons unbeknownst to me, the Line appears to be on the opposite of the mouse, to a point. However the numbers passed to the system are accurate. The Square for the Zoom was less problematic, but there is still a noticeable gap between the Square and the Mouse pointer.