SIT771 Object Oriented Development

Distinction Task 3.5: Mandelbrot

Overview

The Mandelbrot set is an interesting mathematical visualisation. In this task you will create a viewer of the Mandelbrot set, which provides an interesting challenge in order to determine how to zoom in to and out of the section of the Mandelbrot being shown to the user.

The material in Course 1, Week 2 will help you with this task.

This is a Distinction task, so please make sure you are already up to date with all Pass and Credit tasks before attempting this task.

Submission Details

Submit the following files to Doubtfire.

- The Program code (*Program.cs*)
- A screenshot of your program running

You want to focus on the use control flow, as well as reinforcing your understanding of classes, methods, and objects.

Instructions

The Mandelbrot set is a collection of complex numbers that can be visualised graphically. This set is infinitely complex, giving complex and visually appealing images when displayed using software.

The values within the Mandelbrot set can be shown graphically as seen below (image from Wikipedia). While interesting, the nature of the Mandelbrot set is better explored when color is added to the numbers that lie outside the set. The second image shows how the Mandelbrot is located in 2-dimensional space (image also from Wikipedia).

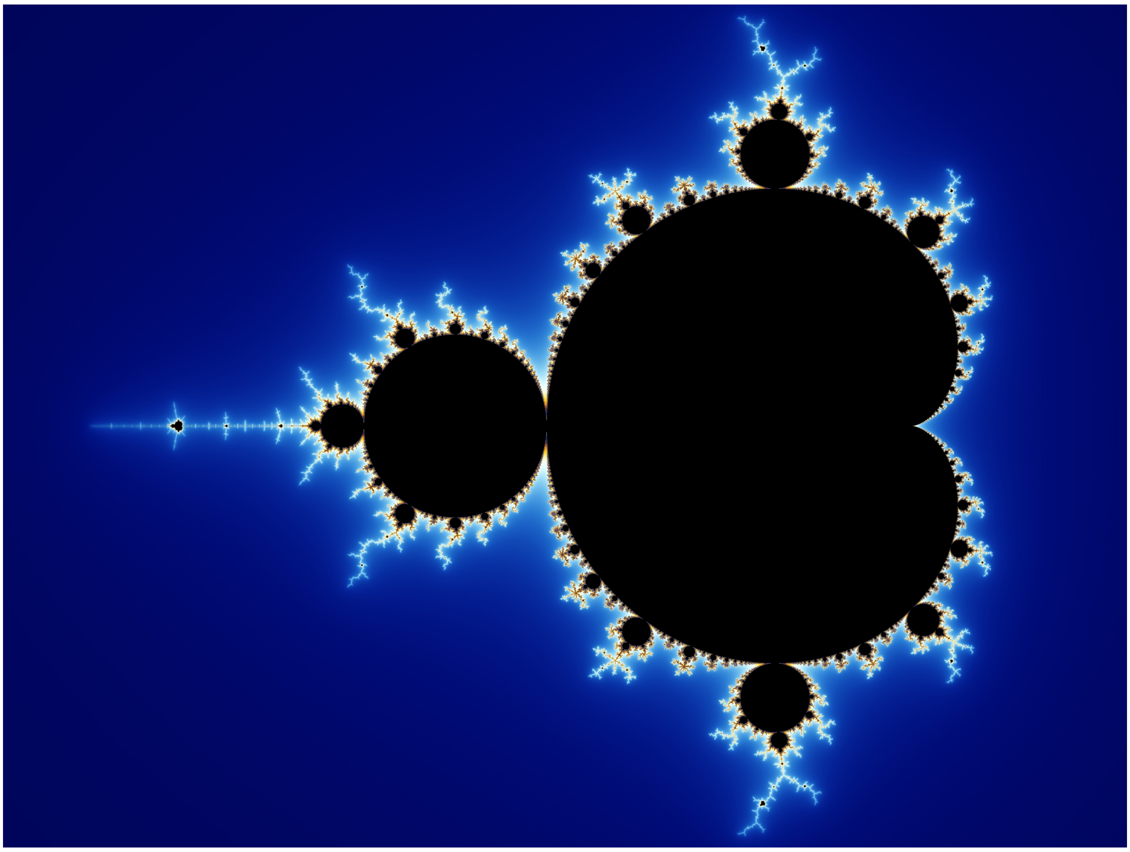

Figure: Mandelbrot visualisation from Wikipedia

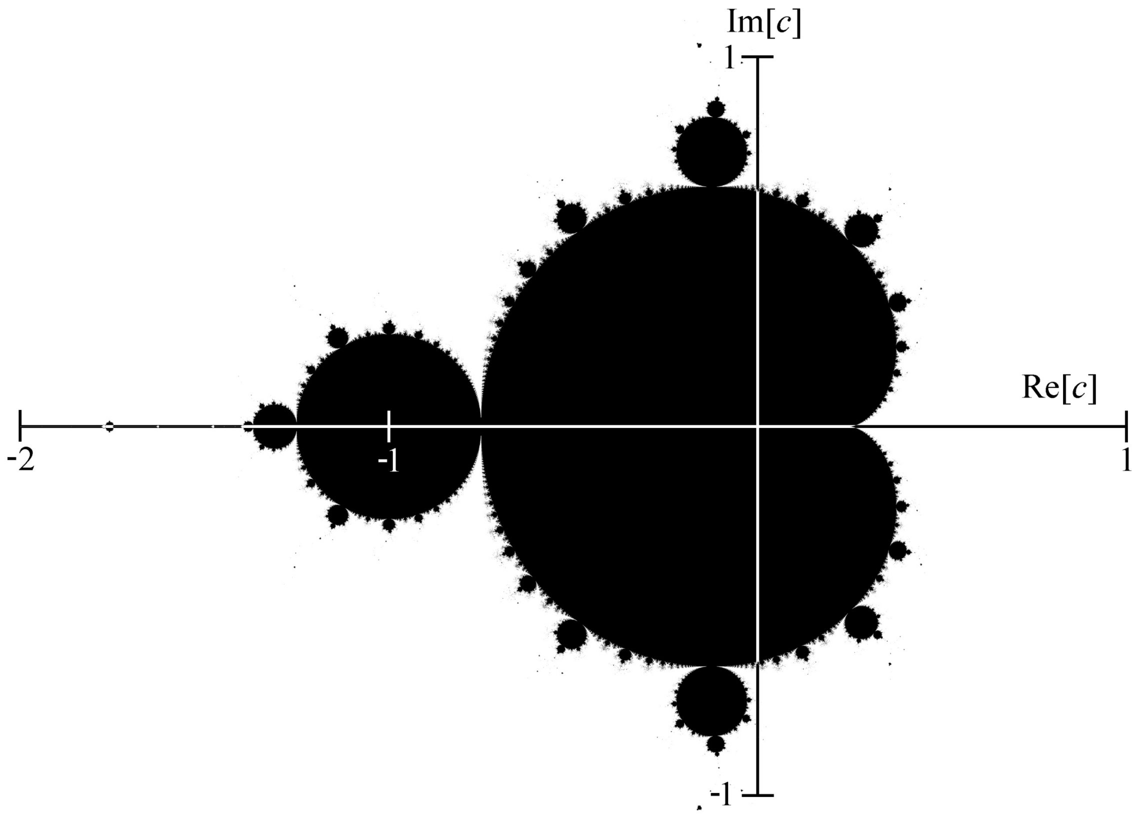

Figure: Mandelbrot set from Wikipedia

The algorithm to determine if a value is within the Mandelbrot set performs a check to see if $x^2 + y^2$ is less than 2^2, x and y are then projected forward and checked again. This process is repeated over and over, and the value lies within the set when this process can be performed infinitely. To create the visualisations shown, all pixels that lie in the Mandelbrot set are drawn black, with those that lie outside the set are coloured based on the number of times the operation could be performed before the value exceeded the 2^2 limit.

The algorithm used to implement this visualisation of the Mandelbrot set is relatively simple, given the complexity of the output. Read the [programmers take on Manbelbrot] for wikipedia and then see if you can come up with your own program structure to code this. There are some detailed hints on the following pages that describe a suitable program structure you can use to implement your own Mandelbrot viewer using SplashKit.

Once you have the code working and showing the full Mandelbrot set, implement zooming when the user clicks the mouse button, allowing them to zoom in on the point selected, or zoom out when they click the right button.

Mandelbot hints

The Mandelbrot program has its functionality distributed across four method, as shown in the following sequence diagram. More details of each method follows.

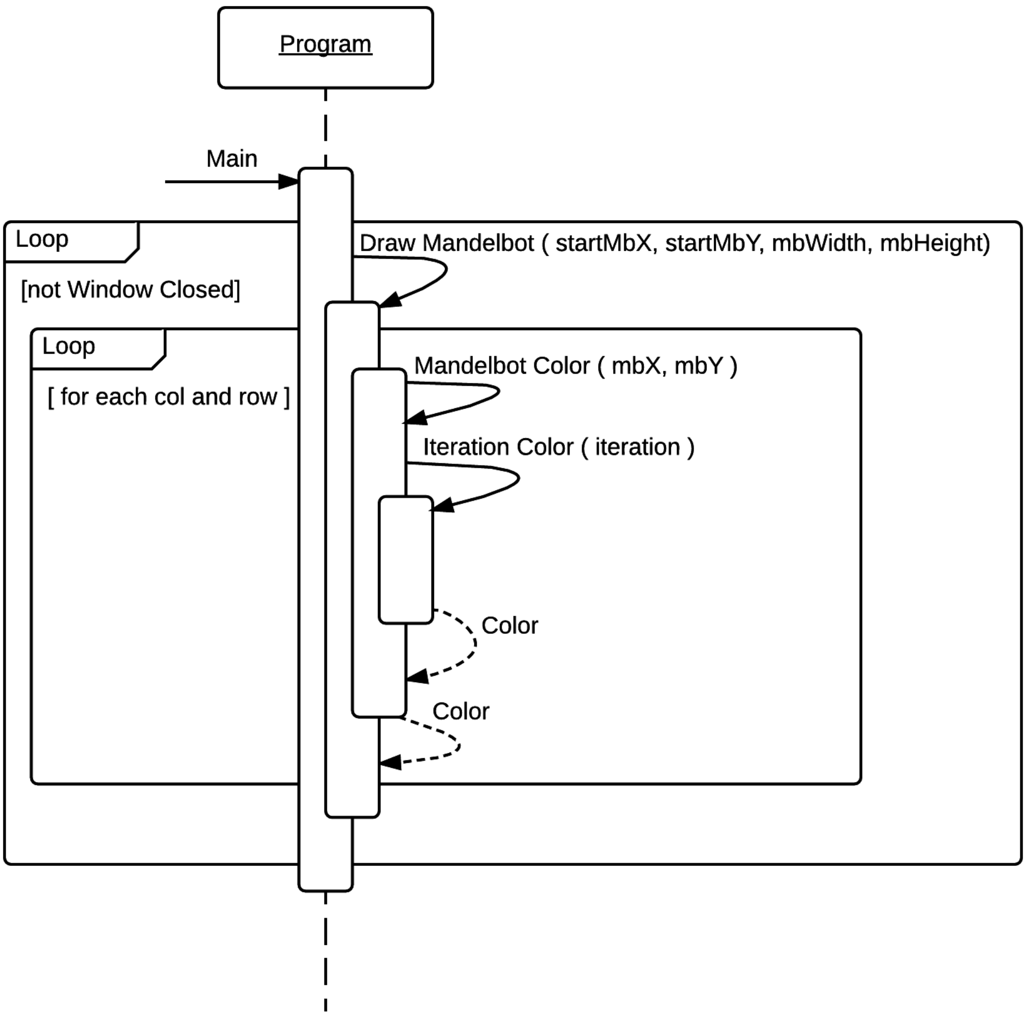

Figure: Sequence for Mandelbrot

The above diagram explains the sequence of method calls within the `Program` class. As there are no other objects involved, we only see the Program and the different method calls all stack on top of each other. Large **loop** boxes show where a method has a loop that repeats a part of the sequence. The details of this are all below.

- The `Main` method opens a new window and then loop repeatedly until the Window is closed.

This loop will call `ProcessEvents` , `DrawMandelbrot` and `RefreshScreen` .

- `DrawMandelbrot` uses the `MandelbrotColor` method to calculate the color to draw for each pixel on the screen. It uses SplashKit's `DrawPixel` method to draw the MandelbrotSet to the screen. This method will accept four parameters (`startMbX` , `startMbY` , `mbWidth` and `mbHeight`). These values represent the area of the Mandelbrot set that will be drawn to the screen. To view the entire Mandelbrot set you need to pass in values `startMbX` -2.5, `startMbY` -1.5, `mbWidth` 4 and `mbHeight` 3.
- A `MandelbrotColor` method that accepts two parameters (`mbX` and `mbY`) and returns a `Color` . The `mbX` and `mbY` parameters represent the coordinates within the Mandelbrot set space, and will be floating point values (`double`). These values will determine if the `mbX` , `mbY` point is within the Mandelbrot, and based on this determine the color at the indicated point.
- An `IterationColor` method accepts the number of iterations (integer) and returns a `Color` . This method will be used in the `MandelbrotColor` method to calculate the Color of a given iteration value.

The following provides some starter pseudocode for each of these methods.

Program Class

We need to add the following method and constants to the Program class.

- A `MAX_ITERATION` (`double`) constant that is set to 1000.0.
- The `DrawMandelbrot` static method
- The `MandelbrotColor` static method
- The `IterationColor` static method

Main

- Steps:
  1. Declare the following local variables:
     - `startMbX` , `startMbY` (`double`) the location in mandelbrot space of the top left corner of the screen.
     - `mbWidth` , `mbHeight` (`double`) for the width and height of the Mandelbrot space shown on the screen.
  2. Assign initial values for mandelbrot coordinates and size (-2.5, -1.5, 3, 4) to `startMbX` , `startMbY` , `mbWidth` and `mbHeight` .
  3. Open a new Window ('Mandelbrot', 320, 240)
  4. Do
     1. Process Events
     2. `DrawMandelbrot( startMbX, startMbY, mbWidth, mbHeight )`
     3. Refresh the Window

5. While the window has not been closed

Draw Mandelbrot

- Parameters:
  - `startMbX`, `startMbY` (`double`) the location in mandelbrot space of the top left corner of the screen.
  - `mbWidth`, `mbHeight` (`double`) the width and height of the Mandelbrot space to be shown on the screen.
- Steps:

1. Declare the following variables... there are a few
   - `scaleWidth` (`double`) scale of screen to mandelbrot space
   - `scaleHeight` (`double`) scale of screen to mandelbrot space
   - `x`, `y` (`int`) screen coordinates
   - `mx`, `my` (`double`) mandelbrot coordinates
   - `mbColor` (`Color`) temporary storage for calculated color
2. Assign `scaleWidth`, the value `mbWidth / SplashKit.ScreenWidth()`
3. Assign `scaleHeight`, `mbHeight / SplashKit.ScreenHeight()`
4. Assign `x` the value `0`
5. While `x` is less than `ScreenWidth()`
   1. Assign `y`, `0`
   2. While `y` is less than `ScreenHeight()`
      1. Assign mx the value `startMbX` + x * `scaleWidth`
      2. Assign my the value `startMbY` + y * `scaleHeight`
      3. Assign `mbColor`, the value of calling `MandelBrotColor(mx, my)`
      4. Call `DrawPixel ( mbColor, x, y )`
      5. Increment `y` by 1
   3. Increment x by 1

Mandelbot Color

- Returns: a Color
- Parameters:
  - `mbX` (`double`) the x value in Mandelbrot space
  - `mbY` (`double`) the y value in Mandelbrot space
- Steps:

1. Declare local variables for `xtemp`, `x`, and `y` (all `double`) to store the altered x, and y values
2. Declare an `iteration` (`int`) for the number of iterations performed
3. Assign `x` and `y`, the values from `mbX` and `mbY`

4. Assign `iteration`, the value `0`
5. Loop while (`x` ^{2} + `y` ^{2} <= 4) AND (`iteration` < `MAX_ITERATION`)

    1. Assign `xtemp` the value `x` ^{2} – `y` ^{2} + `mbX`
    2. Assign `y`, the value `2 * x * y + mbY`
    3. Assign `x`, the value of `xtemp`
    4. Increment iteration by 1 (using `++`)

6. Return the result of calling `IterationColor(iteration)`

Iteration Color

- Returns: a `Color`
- Parameters:

    - `iteration` (`int`), the number of iterations performed

- Steps:

1. Create a `hue` (`double`) local variable to store the hue of the calculated color
2. If `iteration` is larger than or equal to `MAX_ITERATION`

    1. Return the Color Black

3. Else
    1. Assign to `hue`, the value `0.5 + (iteration / MAX_ITERATION)`
    2. If `hue` is larger than 1

        1. Assign to `hue`, the value `hue - 1`

    3. Return `Color.HSBColor(hue, 0.8, 0.9);`

Adding Zoom...

Read and understand the code that you have written, after making sure it works correctly. Then work to add a zoom function.

You can implement zoom by altering the values in the `startMbX`, `startMbY`, `mbWidth` and `mbHeight` variables. By reducing the width/height you zoom in, by increasing it you zoom out. You need to adjust the `startMbX` and `startMbY` to move around in the Mandelbrot set. So zooming involves both changing the size and location you are viewing.

Use SplashKit functions to determine when the user clicked the mouse button (`MouseClicked`) and the position of the mouse at the time (`MouseX` and `MouseY`). With this information you can then alter the `startMbX`, `startMbY`, `mbWidth` and `mbHeight` values to zoom in on the area clicked.

Hints on Zooming

Hint: When zooming in, make the new `mbWidth` and `mbHeight` half their current value, and double these values when zooming out. Other values can be calculated using proportions.

For example, when zooming in.

- `newMbWidth = mbWidth / 2;`
- The location the user clicked would be (using the current `mbWidth`): `startMbX + MouseX() / SplashKit.ScreenWidth() * mbWidth`.
- The new `startMbX` would be that position, minus half of the new mandelbrot width (eg - `newMbWidth / 2`).
- `mbWidth = newMbWidth / 2;`

The same calculation can be used to determine the y position, and similar steps can be used to zoom out.

Once you have this zooming in and out, grab a screenshot of a zoomed in part of the mandelbrot then submit your work to Doubtfire.