



Introduction to Programming

Distinction Task 1.1: Mandelbrot

Overview

The Mandelbrot 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.

- Purpose:** Learn to put the different programming tools together.
- Task:** Create a Mandelbrot viewer program.
- Time:** This should be completed before the start of week 5, but may be completed at a later date (see following note).
- Resources:**
- Programming Arcana

Note: If you are not currently up to date you should skip this task and return to it once you are up to date with the Pass Tasks. Do not allow Distinction Tasks to delay you in keeping up with the unit's Pass Tasks.

Submission Details and Assessment Criteria

You must submit the following files to Doubtfire:

- Program source code, and screenshot of the program in action.

Make sure that your task has the following in your submission:

- Code must follow the Pascal coding convention used in the unit (layout, and use of case).
- Must include the ability to zoom in and out of the Mandelbrot.

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 in Figure 1. While interesting, the nature of the Mandelbrot set is better explored when color is added to the numbers that lie outside the set. Figure 2 shows a part of the Mandelbrot set, illustrating the complexity of the set.

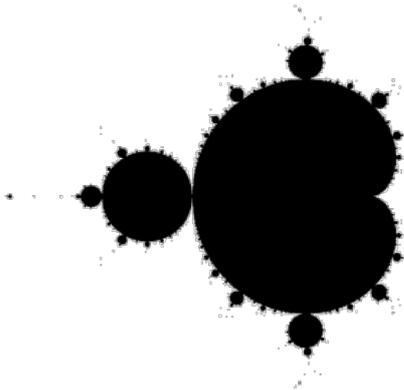


Figure 1: Values in the Mandelbrot set www.ddewey.net/mandelbrot



Figure 2: A zoomed in portion of the Mandelbrot set from <http://www.linesandcolors.com>.

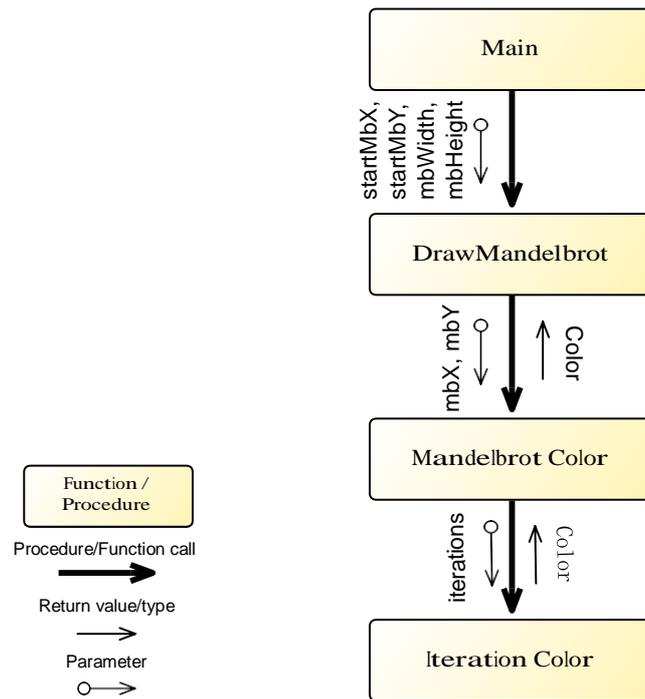
The algorithm to determine if a value is within the Mandelbrot set performs a check to see if $x^2 + y^2$ is less than 2^2 , x and y are then projected forward and checked again. This process is repeated over and over, and the value lies within the set when this process can be performed infinitely. To 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 performed¹ 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 Mandelbrot on wikipedia at http://en.wikipedia.org/wiki/Mandelbrot_set#For_programmers and then see if you can come up with your own program structure to code this. Alternatively, the following pages describe a suitable program structure you can use to implement your own Mandelbrot viewer using SwinGame.

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.

¹ The repetition of a process is known as an iteration.

The Mandelbrot program has its functionality distributed across four functions/procedures, as shown in the following structure chart. More details of each function and procedure follows.



- The **Main** procedure opens a graphical window and then loop repeatedly until `Window-CloseRequested()` is true. Each loop this will call `ProcessEvents()`, `DrawMandelbrot(...)` and `RefreshScreen()`.
- **DrawMandelbrot** uses the `MandelbrotColor` function with `SwinGame's DrawPixel` function to draw the `MandelbrotSet` to the screen. This procedure 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, shown in Figure 4.
- A **MandelbrotColor** function 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** function that accepts the number of iterations (integer) and returns a color. This function will be used in the *MandelbrotColor* function to calculate the color of a given iteration value.

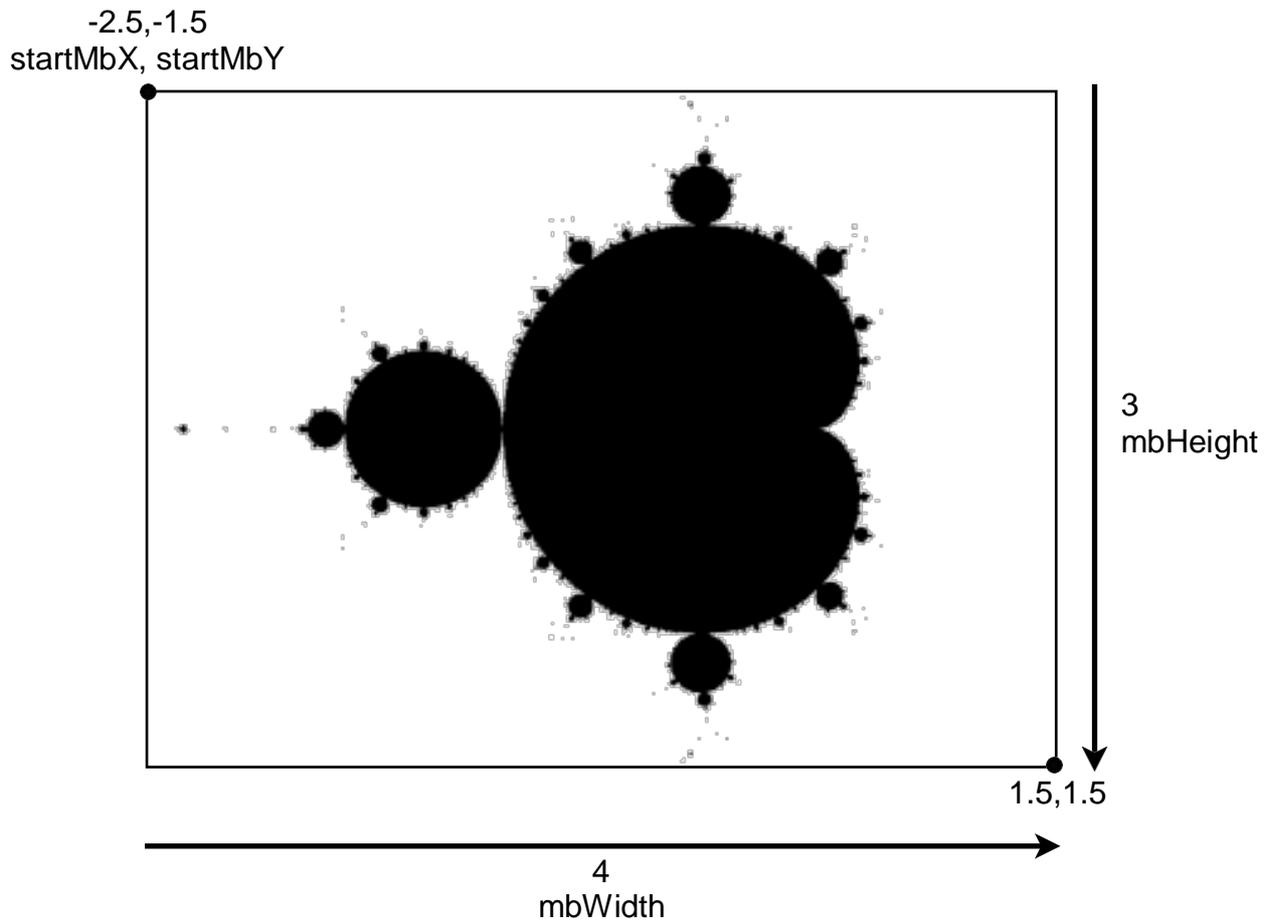


Figure 4: The space occupied by the Mandelbrot set.

```

Program: Mandelbrot
-----
Uses: SwinGame, sgTypes

Const: MAX_ITERATION with a value of 1000

Function: IterationColor
Function: MandelbrotColor
Procedure: DrawMandelbrot
Procedure: Main
-----
Steps:
1: Call Main

```

Listing 1: Pseudocode for the Mandelbrot program

Function: Iteration Color

Returns: a Color

Parameters:

- iteration (Integer), the number of iterations performed

Local Variables:

- hue (Double) to store the hue of the calculated color

Steps:

- 1: If iteration >= MAX_ITERATION Then
- 2: Assign result, the value ColorBlack
- 3: Else
- 4: Assign to hue 0.5 + (iteration / MAX_ITERATION)
- 5: If hue > 1 Then
- 6: Assign to hue, the value hue - 1
- 7: Assign result, the value of HSBColor(hue, 0.8, 0.9)

Listing 2: Pseudocode for the Iteration Color function

Note: Watch the indentation in the above code... the last four statements are in the else branch.

Function: Mandelbrot Color

Returns: a Color

Parameters:

- mbX (Double) the x value in Mandelbrot space
- mbY (Double) the y value in Mandelbrot space

Local Variables:

- xtemp, x, y (Double) store the altered x,y values
- iteration (Integer) the number of iterations performed

Steps:

- 1: Assign x and y, the values from mbX and mbY
- 3: Assign iteration, the value 0
- 4: While ($x^2 + y^2 \leq 4$) AND (iteration < MAX_ITERATION)
- 5: Assign xtemp the value $x^2 - y^2 + mbX$
- 6: Assign y, the value $2 * x * y + mbY$
- 7: Assign x, the value of xtemp
- 8: Increment iteration by 1
- 9: Assign result the value of IterationColor(iteration)

Listing 3: Pseudocode for the Mandelbrot Color function

Procedure: 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.

Local Variables:

- scaleWidth (Double) scale of screen to mandelbrot space
- scaleHeight(Double) scale of screen to mandelbrot space
- x, y (Integer) screen coordinates
- mx, my (Double) mandelbrot coordinates
- mbColor (Color) temporary storage for calculated color

Steps:

- 1: Assign scaleWidth, the value mbWidth / ScreenWidth()
- 2: Assign scaleHeight, mbHeight / ScreenHeight()
- 3: Assign x the value 0

- 4: While x is less than ScreenWidth()
- 5: Assign y, 0

- 6: While y is less than ScreenHeight()
- 7: Assign mx, startMbX + x * scaleWidth
- 8: Assign my, startMbY + y * scaleHeight
- 9: Assign mbColor, the value of calling
 MandelBrotColor(mx, my)
- 10: Call DrawPixel (mbColor, x, y)
- 11: Increment y by 1 // end while y

- 12: Increment x by 1

Listing 4: Pseudocode for the Draw Mandelbrot procedure

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 SwinGame 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. The illustrations in Figures 5 and 6 are provided to assist you working out how to calculate these new values.

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, see the figures on the following page.

For example, when zooming in:

```
newMbWidth := mbWidth / 2;
```

The location the user clicked would be (using the old *mbWidth*):

```
startMbX + MouseX() / ScreenWidth() * mbWidth
```

So the new *startMbX* would be that position, *minus* half of the new mandelbrot width (eg - *newMbWidth* / 2).

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

Remember to assign the *mbWidth* the *newMbWidth* after calculating the new start position.

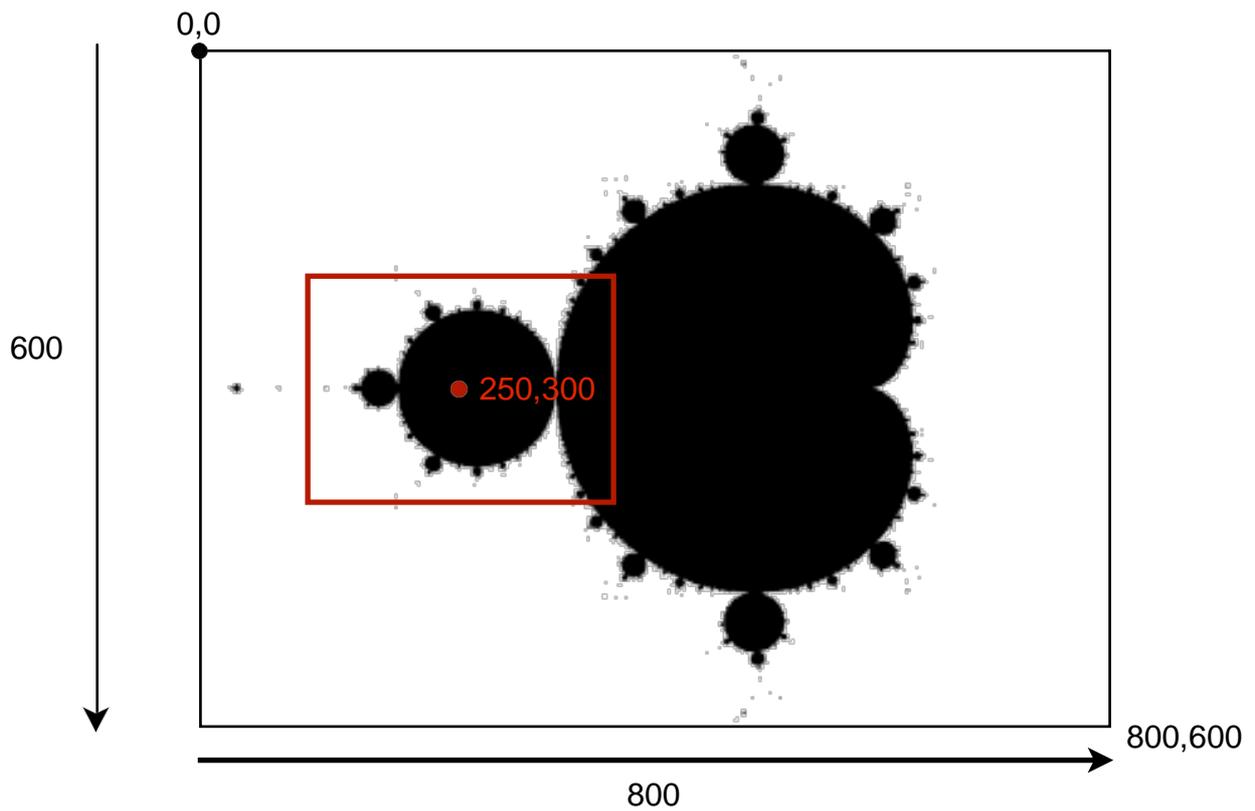


Figure 5: Illustration of the user clicking at 250,300 and the resulting area that will be shown. All values are in pixels (screen coordinates).

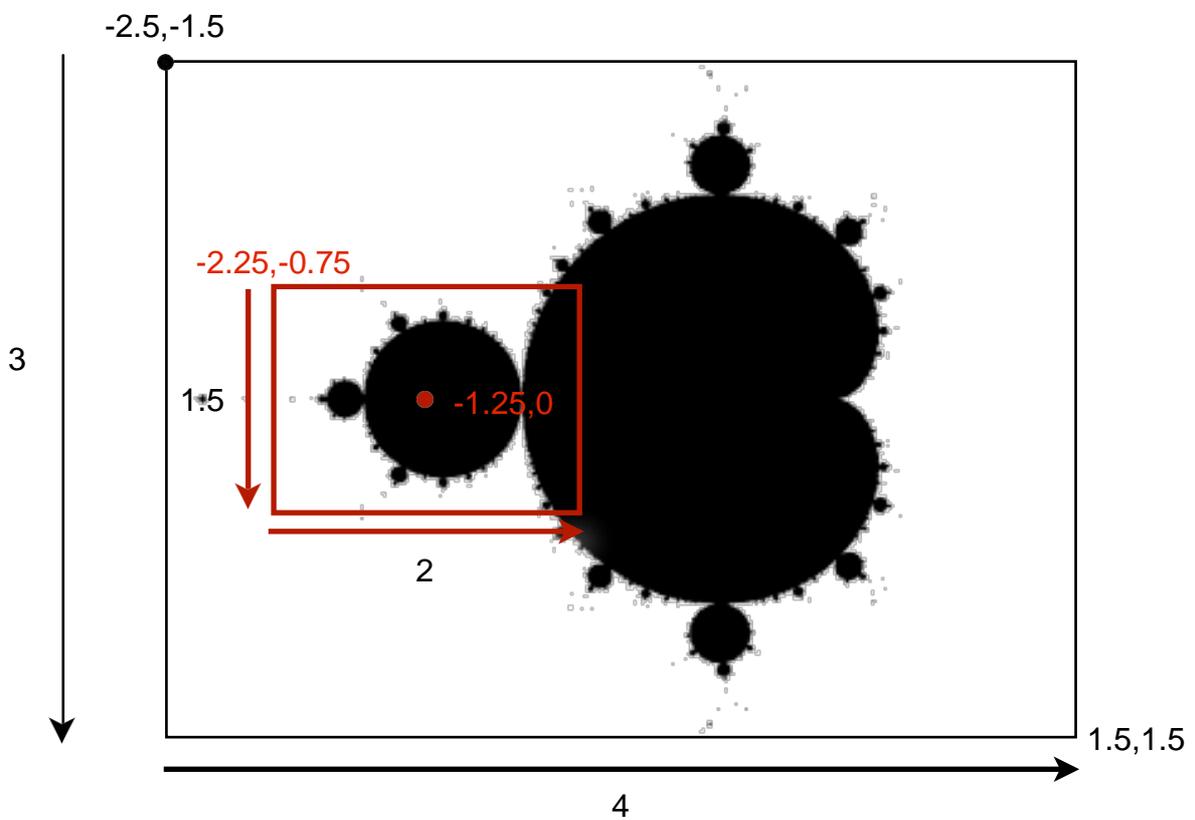


Figure 6: Illustration matching Figure 1 but showing the values in Mandelbrot set coordinates.