**Mandelbrot Set Fractals Visualisation Tool; NEA Project – Anss Hameed**

**Contents**

1. Analysis …………………………………………………………………………………………… 5
   1. Introduction……………………………………………………………………………5
      1. Introduction to NEA Project ………………………………………………5
      2. Description ………………………………………………………………………. 5
      3. Introduction Background Research ………………………………….. 5
   2. Objectives …………………………………………………………………………….. 6
      1. Measurable Core Objectives ……………………………………………. 6
      2. Acceptable Limitations …………………………………………………….. 8
   3. Further Research of Mandelbrot Set …………………………………….. 8
      1. Introduction to Mathematics Behind Mandelbrot Set …….. 8
      2. Mathematical rules of Mandelbrot on computer …………….. 11
      3. How to Calculate Set ……………………………………………………….. 12
      4. Colouring …………………………………………………………………………. 12
      5. General Properties of Mandelbrot Set ……………………………… 14
      6. Julia Set …………………………………………………………………………….. 14
      7. Real Life Applications of Mandelbrot Set ………………………….. 16
      8. Existing Applications …………………………………………………………. 16
         1. Summary of Existing Applications…………………………..20
   4. Modelling of The Problem …………………………………………………….. 20
      1. Flow Charts ………………………………………………………………………. 20
         1. Mandelbrot Set Iteration Equation …………………………. 20
         2. Basic Black and White Colouring Algorithm ……………. 21
         3. Histogram Colouring Algorithm ………………………………. 22
         4. Optimised Escape Time Algorithm ………………………….. 23
      2. DFD Models ………………………………………………………………………. 23
         1. Mandelbrot Set Iteration Equation …………………………. 23
         2. Continuous (Smooth) Colouring Algorithm ……………… 24
   5. Data Dictionaries ……………………………………………………………………. 24
      1. Mathematical Components for Calculation (Key Words) ….. 24
      2. Hardware and Software Considerations ……………………………. 25
         1. (Data Dictionary) Hardware/Peripherals Used ………… 25
         2. Brief Software Details …………………………………………….. 25
   6. Critical Path Analysis ……………………………………………………………… 26
      1. Gantt Chart ……………………………………………………………………… 26
   7. Data Volumes Investigation ………………………………………………….. 27
   8. Third Party Analysis ………………………………………………………………. 28
      1. Third Party Source/End User …………………………………………… 28
      2. Third Party Interviews …………………………………………………….. 28
         1. Interview Person A ……………………………………………….. 28
         2. Interview Person B ……………………………………………….. 29
         3. Interview Person C ……………………………………………….. 30
         4. Interview Person D ……………………………………………….. 31
   9. Prototypes …………………………………………………………………………… 32
      1. Experimenting with Mandelbrot Set Algorithm ………………. 32
      2. Experimenting Python GUI + Refining Unstructured Code.. 33
   10. OOP Basis Analysis………………………………………………………………….34
   11. Significant changes made after analysis……………………………..35
2. Design ……………………………………………………………………………………………. 35
   1. Design Overview/Structure ………………………………………………….. 35
   2. Algorithms ……………………………………………………………………………. 35
      1. Iterations Calculation ………………………………………………………. 35
         1. Iterations Calculation – Pseudocode ……………………… 36
      2. Pixel to Complex Number Conversion ……………………………… 36
         1. Complex Number Conversion Pseudocode Type 1….. 36
         2. Complex Number Conversion Pseudocode Type 2….. 37
      3. Histogram Colouring Method Algorithm …………………………… 38
         1. Histogram Colouring Method Pseudocode ……………… 38
      4. Creating Image Algorithm …………………………………………………. 38
         1. Creating Image Algorithm Pseudocode …………………… 38
      5. Zooming Stack Algorithm …………………………………………………. 39
         1. Zooming Stack Algorithm Pseudocode…………………….. 39
      6. Smooth Colouring Algorithm …………………………………………….. 40
         1. Smooth Colouring Algorithm Pseudocode ………………..40
      7. Login Procedure …………………………………………………………………. 39
         1. Login Procedure Pseudocode …………………………………… 40
      8. Sign Up Procedure ……………………………………………………………… 40
         1. Login Procedure Pseudocode …………………………………… 40
   3. Overall Program Structure ………………………………………………………. 41
      1. Overall System Flowchart …………………………………………………… 41
         1. Screenshot 1: Overview…………………………………………….. 42
         2. Screenshot 2: Top …………………………………………………….. 42
         3. Screenshot 3: Middle…………………………………………………. 42
         4. Screenshot 3: Bottom………………………………………………….43
      2. Overall System Hierarchy Chart ……………………………………………44
   4. Data Structures…………………………………………………………………………44
      1. Stack……………………………………………………………………………………44
         1. Undo/Redo Stack………………………………………………………. 44
         2. OOP Class Template of Stack……………………………………...45
         3. Subroutine Stack…………………………………………………………45
         4. Zoom In/Out Stack………………………………………………………46
         5. Basic Stack Diagram……………………………………………………46
      2. 2D Array………………………………………………………………………………..46
      3. Hashing…………………………………………………………………………………47
   5. File Organisation………………………………………………………………………..47
      1. Read/Write 2D Array to file (Pseudocode)…………………………..48
      2. Read/Write 2D Array to file (DFD)……………………………………….49
   6. OOP Design………………………………………………………………………………..50
      1. UML Diagram (Overall Structure)………………………………………….51
      2. Inheritance Diagrams – Colouring Algorithms……………………….52
      3. Class Definitions…………………………………………………………………….53
   7. Data Dictionaries (Updated)………………………………………………………59
      1. File Organisation Data Dictionary (Key Terms)……………………..59
      2. Key Variables (Overall Program)…………………………………………..60
   8. IPOS…………………………………………………………………………………………..62
   9. API Overview……………………………………………………………………………..63
   10. HCI……………………………………………………………………………………………..65
       1. Overall Interface Structure……………………………………………….65
       2. Analysis of Tabs……………………………………………………………….66
          1. Files Tab……………………………………………………………………..67
          2. Options Tab………………………………………………………………..67
             1. Login and Registry System…………………………..68
             2. Zoom In/Out Function Interface………………….68
          3. Colours Tab………………………………………………………………..70
          4. Edit Tab……………………………………………………………………...71
          5. Help Tab……………………………………………………………………..72
       3. Main Menu………………………………………………………………………73
3. Implementation…………………………………………………………………………………74
   1. Overview………………………………………………………………………………74
   2. Login/Registry System…………………………………………………………..74
      1. Main Menu Class…………………………………………………………….74
      2. Registration Class……………………………………………………………77
         1. Vernam Cipher…………………………………………………..83
      3. Login Class……………………………………………………………………..85
         1. Vernam Decrypter……………………………………………..89
      4. Login/Registry Call…………………………………………………………89
   3. Editing Window……………………………………………………………………90
   4. Mandelbrot Fractal……………………………………………………………..103
      1. Stack Class……………………………………………………………………103
      2. Getter Functions for Fractal Class…………………………………105
      3. Mandelbrot Formula…………………………………………………….109
      4. Mandelbrot Recursion…………………………………………………..110
      5. Radius Limit Check…………………………………………………………110
      6. Linear Interpolation and Histogram Colouring/Assigning..110
      7. Smooth Colouring (Extension Objective Met)…………………112
      8. Coordinate/Pixel Complex Number Conversions…………….113
      9. Main Mandelbrot Fractal Plot………………………………………….114
      10. Mouse Click Stack Impementation…………………………………..119
      11. Running Mandelbrot Fractal…………………………………………….121
   5. RGB Sliders……………………………………………………………………………..123
      1. Setting Up Sliders……………………………………………………………..123
      2. Assigning Inputs to Colour………………………………………………..126
      3. Exception Handling of Hex RGB Values and Entries…………..127
   6. Julia Set Fractal (Extension Objective Met)……………………………..129
      1. Julia Fractal Setup…………………………………………………………….129
      2. Setter Functions……………………………………………………………….130
      3. Julia set pixel/coordinates complex no. conversions………..131
      4. Julia Set Iterations Algorithm……………………………………………132
      5. Julia Axes Plotting…………………………………………………………….133
      6. Main Julia Set PyGame Window Plotting………………………….134
      7. Run Julia Set…………………………………………………………………….137
   7. Help Menu Sample Screenshots…………………………………………….138
4. Testing……………………………………………………………………………………………….139
   1. Input Validation Testing………………………………………………………..139
      1. Registration System………………………………………………………..139
      2. Mandelbrot Editing Input Values…………………………………….143
      3. RGB Value Slider Testing…………………………………………………145
   2. File Handling Test………………………………………………………………….146
      1. Login/Registration System……………………………………………...146
      2. Mandelbrot Editing Tool (Applying Values)……………………..147
      3. Vernam Cipher…………………………………………………………………148
   3. Interface Testing…………………………………………………………………….149
      1. Main Menu………………………………………………………………………149
      2. Editing Window……………………………………………………………….150
      3. Fractal Interface Window………………………………………………..151
         1. Fractal Timing Testing…………………………………………153
   4. OOP Class Testing………………………………………………………………….154
      1. Main Menu……………………………………………………………………..154
      2. Editing Window………………………………………………………………155
      3. Mandelbrot Fractal Window (PyGame)…………………………..155
   5. Colour Theme Testing…………………………………………………………..155
   6. Third Party Testing………………………………………………………………..156
5. Evaluation………………………………………………………………………………………….156
   1. Objectives………………………………………………………………………………156
      1. Core Objectives………………………………………………………………..156
      2. Acceptable Limitations/Extension Objectives…………………..160
   2. Final Third Party Feedback/Analysis……………………………………….162
   3. Possible Improvements/Extensions………………………………………..163
   4. Obstacles Overcome………………………………………………………………164
   5. What I have learned……………………………………………………………….165
6. Appendices…………………………………………………………………………………………166
   1. Site References………………………………………………………………………166
   2. Testing Screenshots………………………………………………………………166
   3. Testing Video (Lowest Optimisation)…………………………………….181
7. **– Analysis**
   1. **Introduction** 
      1. **Introduction to NEA project**

The project I have chosen is the ‘Mandelbrot Set Fractals Visualisation Tool.’ I have chosen this particular project as I am very interested and passionate about Maths and it is one of my A Level subjects, therefore, I would have a good idea on how to go about this project In a complex, challenging and intuitive way. Primarily, this project is a modelling simulation where the user will be able to view the Mandelbrot set fractals in a variety of different ways when inputting any value whether its simplistic or complex. Either way, the user should be able to have full control over manipulating the fractals using a certain input of values with a simple way of using it without confusing the user. Overall, I will aim to complete this mathematical modelling simulation with implementing a very complex system but allowing the user interface to be simple and convenient to use through the implementation of abstraction. This system could be used by third parties relating to teaching and education.

* + 1. **Description**

I believe Mathematics can be a very complex subject which can be very hard to understand at times, especially when dealing with an aspect such as the Mandelbrot set. I will be designing this tool for the visualisation of the fractals to be easy to manipulate and to allow the user with many ways of dealing with the tool. In this program, I will allow the user to manipulate the plotter effectively with the simplicity of the controls over a complex system to allow the user a high level of flexibility and control over the system to use it to its full extent. This will be done through the user interface allowing the user to use a variety of different functions such as zooming In and out of the fractals and even customising the colours used in the image shown with the input of varying values whether they’re simplistic or complex (I will try to make the limits as indefinite as possible) and of course, the user should be able to save their progress within a file or database in order to reload their image and values in their next time of usage through the login of the user.

* + 1. **Introduction Background Research**

A fractal is a curve, image or visualisation of a geometrical figure used in mathematics where each part of the fractal has the same numerical and statistical character as the rest of the fractal. They are used in modelling structures such as snowflakes (another fractal set) and in the image which is illustrated with computational and mathematical figures and formulas, similar patterns appear recursively and progressively in smaller scales as you zoom in further into the image. Overall, fractals are never ending and complex patterns which are similar across many different scales controlled by the user, they’re closely associated with the idea of dynamic imagery and systems along with the chaos theory. The Mandelbrot set is used to refer a fractal set and was originally founded and investigated in the beginning of the 20th century by the French Mathematicians Gaston Julia and Pierre Fatou and it originates from complex dynamics. In 1980, Benoit Mandelbrot (person of interest in fractal geometry) discovered the Mandelbrot set and was the first person to view this particular set of fractals onto a computer.

In terms of plotting the Mandelbrot set, it marks the set of varying points in a complex plane where the Julia set (associated fractal set) is connected but not computable. The Mandelbrot set is specifically operated using a quadratic recurrent equation of:

Zn+1 = zn^2 + c

Where c and z are represented to be complex constant numbers and n is represented to be zero or any positive integer or also known as a natural number. In the Mandelbrot set, it starts with z0 if the absolute value of zn is never made to be bigger in value than a value used in the constant, regardless of how large the value of n gets. The image is created using the equation by applying it with inserted values to each individual pixel in an iterative process, which uses the pixel’s position in the image for the value of c – the value of c is used for mapping the position of a particular pixel in the image, being relative to the point of the position on the complex and chaotic plane of the image.

* 1. **Objectives**
     1. **Measurable Core Objectives**
* The user must be able undo actions by popping off the previously pushed on actions and manipulations in recent usage off a ‘stack’ (abstract data structure). Therefore, the user should be able to re-do actions using a stack by pushing item on stack.
* The program must allow the user to zoom in and out of the image (with limitations e.g. 10^32) until the image starts to deteriorate as the Mandelbrot set isn’t infinite as the orbit of 0 under the equation doesn’t go to infinity (value of c). The resolution of the image must be maintained with a high level of zooming in to allow the user to explore the image with ease and enjoyment (can only be done within the limit of the Mandelbrot set up to the point of disorientation of the image)
* The program must be able to manipulate and use the inputted values correctly no matter how complex or simple they are, to be able to accordingly perform the user’s operations with full effect with the correct calculations e.g. the demanded ‘MaxIterations’ must be as accurate as possible. Input panel must be put into the system to allow the user to input values such as the maximum iterations.
* In case the value inputted into any variables is unrealistic due to erroneous limits, the program should output an alert to notify the user of an error and suggest a solution e.g. let the user know the limitations of zooming in, or provide them with a warning message that image may take long to load if unrealistic value is inputted e.g. Maximum Iterations: 10,000
* The user interface MUST be simple and convenient to follow with clear options placed in the most appropriate positions as possible to make them easy to follow to please the user and convince them to use this tool again and making the whole interface readable. For example, the options must be easy to follow and the font and colour of the text should be easy to read. The user should be able to manipulate easy-to-use control keys with various controls (such as zooming in) in the complex system using abstraction – swift and easy-to-use user interface.
* The program should be able to provide a ‘help’ tab to open whenever the user needs a reminder of how to use the program or if the user is completely new to the program. This must be easy to find with significance within the user’s visualisation on the screen for easy access. Also can include a glossary of values for brand new users.
* The program should be able to output complex information to intrigue the user with knowledge such as the ‘Complex Number: Mouse Position Value’
* The user must be able to manipulate the image visualisation to a certain extent of complexity such as allowing the colour theme to be changed. A suitable colouring algorithm must be used to colour the pixels rather than a simple greyscale representation of the image e.g. Histogram method or standardised colouring palette for colour themes; RGB values (smooth colouring as extension objective/acceptable limitation)
* The manipulation of the image must be swift with a minimisation of ‘lag’ in terms of zooming into different parts of the image (depending on hardware and given inputs)
* The program must be able to load and present the image within a short time in real-time for the user’s satisfaction. If the input values are changed (such as the MaxIterations or ImageSize) it must change within a given time frame of 0 – 150 seconds (150 seconds being the absolute maximum with REALISTIC values – most computers can’t process maximum iteration values of unrealistic values of 5000; processing time may vary with hardware).
* The program must be able to convert the pixels to complex numbers in order for the circadian plane to process the pixels
* Provide a method within the code to present the image in the user interface with the use of manipulation of the image – making it the most significant part of the interface. (e.g. Using the ‘matplotlib’ library which specialises in plotting and presenting images. This will be decided after the prototype investigations) – most likely to use Pygame.
* Allow the user to log into the system, and if they have not got login details then they should be able to sign up.
* If the user wishes to enter as a guest, this must be allowed when they don’t sign up or login.
* User must be able to quit the program instantly when required.
* User must be able to change the scaling speed of the rectangular zoom box
* Create appropriate working slider for RGB values in case user wants to change RGB colour values of fractal manually through source code.
* User must be able to change image size (dimensions) from drop down menu of industry standard image dimensions.
* The user must be allowed to save the fractal image in runtime
  + 1. **Acceptable Limitations/Extension Objectives**
* There will inevitably always be a limitation in zooming into the image as there are boundaries selected such as the maximum iterations and image size; my program may not be able to zoom in as further in as the other programs I have investigated earlier on in my analysis – but I will make it acceptable for the purpose of the program and maximise flexibility of zooming in as far as possible.
* Disallowing certain numbers when put into the system and an error message being outputted when these numbers are being inputted. I will make the user aware of what sort of numbers can be input e.g. complex numbers and what numbers can’t be input e.g. logarithmic functions
* There are certain colouring algorithms which may be better and more appealing in terms of viewing the Mandelbrot set e.g. continuous smooth colouring. The bands of colours will be rendered with higher quality and replaced with a smoother gradient with the use of logarithmic functions, as opposed to the Histogram colouring algorithm (algorithm I am most likely to use). However, it may be far too complex to implement within the time frame given with the combination of other objectives I am trying to achieve e.g. minimising time delay and processing of printing the image out on the screen.
* My project is specifically targeted at the Mandelbrot set of fractals, but some other systems allow the use of other fractal e.g. Julia set fractals, snowflakes, Lyapunov fractals. This may be a limitation as in my investigation I discovered that other projects have implemented both the Mandelbrot set with the Julia set and a wide range of other fractals. Implementing the Julia set fractal (closely associated with Mandelbrot)
* One well known limitation of implementing Mandelbrot set is the limitation in drawing external rays. This is a limitation in which the drawing tool for the set cannot be used to carry on. The boundary point is implemented by the resolution of the data type of floating-point computation used in the code. This is analysed via the floating-point computation with quadruple 128 bits. This has become a limitation because computer systems usually use 64-bit floating point numbers (double data type) and the program won’t let you zoom into certain areas which require 128 bit floating point communication, such as zooming into specific rectangles.
* Implement an encryption algorithm for the login/registry system to encrypt the password when registering and decrypting when logging in with the ‘key.’
  1. **Further Research of Mandelbrot Set**
     1. **Introduction to the Mathematics behind the Mandelbrot set**

In this section, I will explain the mathematical rules and information behind the theory of the Mandelbrot set and how it works. The usage of fractals is shown by the fractal theory and this is the idea (created by Benoit Mandelbrot) of modelling sets (such as the Mandelbrot set) to visualise real life problems using the ‘chaos theory’ which may be seen as a diverse and dynamic phenomena using different applications such as dynamically changing stock market prices, plant growth and cloud boundaries or even the complex idea of the ‘distribution of matter’ in the universe. (1)

**General form of iteration**

(2) The mathematics behind the Mandelbrot set theory is generated around the idea of iterations (the repetition of a process over a period of time), which is heavily used in computer science for programming or sequences in Mathematics. For the Mandelbrot set in particular, mathematical functions are used, specifically quadratic polynomials and it takes the form of ‘f(x) = x^2 + c’ 🡪 c is represented a constant number which can be changed with a variety of inputs. (Note: in general, Mandelbrot formula may be written as z rather than x, but same rules apply). For an iteration, there always needs to be a specific start to it, for the Mandelbrot set, the equation of the form x^2 + c is being iterated. It begins with a ‘seed’ for the iteration of this equation and this is represented as ‘x0’. This is then incremented In order to create an iteration, as shown below:

* X1 = x0^2 + c
* X2 = x1^2 + c
* X3 = x2^2 + c

As seen above, the variable for the equations starts from 1 and ‘x’ is then incremented, and the power of 2 stays the same (as it takes the form of a quadratic polynomial), and of course, the constant of c is the same in the general form taken. The list of numbers can be represented as: x0, x1, x2…

This specific iteration is referred to as ‘the orbit of x0 under the iteration of x^2 + c.’ This iteration is used in a variety of other real life models other than the Mandelbrot set, some examples are modelling the exponential growth of a population of the ‘breeding cycle.’

**Example 1(using the constant of 3 and using the seed of 2):**

1. X0 = 2
2. X1 = (2)^2 +3 = 12
3. X2 = (12)^2 + 3 = 147

Therefore, the ‘orbit under the iteration’ increases and goes towards infinity (but doesn’t reach infinity). In this particular example, the seed is plugged in from the previous value/orbit outputted (as its an iteration) from the starting point of ‘x0 =2’, however, if chosen, the seed can be remained at a fixed value e.g. the seed may be fixed at the value of 5, and therefore the value substituted in for x would be 5 for each iteration.

Note: the orbit doesn’t necessarily have to increase for each iteration, it may alternate between two values, e.g. between 0 and -1. Therefore, the ‘cycle of period’ would be 2.

There are different constants which can be used which each represent a different situation:

* C = -0.65 = ‘tends towards a fixed point’
* C = -1.6

**Complex numbers**

The information described above was the general form the iteration of the Mandelbrot set takes in order to produce the image on a complex plane. However, it’s not as simple as that. The values inputted into the equation of the iterations are far more complicated than simple integers and simple orbits being outputted. Rather than just implementing ‘real values’ of the constant c, the constant may also consist of being a complex of a number (note: ‘a complex number is a number that can be written in the form of a + b I a + +bi a + bi, where a and b are real numbers and I is the imaginary unit defined by I 2 = -2 i^2 = -1 i2 = 1. The set of complex numbers, denoted by C, includes the set of real numbers (R) and the set of pure imaginary numbers.’) (3)

**Example (using complex numbers with the form of x^2 +c, where c = i):**

1. X0 = 0
2. X1 = (0)^2 + I = i
3. X2 = (i)^2 + I = -1 + i
4. X3 = (-1+i)^2 + I = -i
5. X4 = (-i)^2 + I = - 1 + i
6. X5 = (-1 + i)^2 + I = -i

* With this particular example, the orbit results in a period of cycle of 2, however, this isn’t the case with all complex numbers being plugged into the equation. For example, if 3i was the seed to the equation, the orbit would increase in value and tend to infinity.

Complex numbers are 2 dimensional. The two parts they consist of are:

1. The ‘real’ part of the number – this is the part of the number **without** the ‘i’ coefficient
2. The ‘imaginary’ part of the number – this is the part of the number **with** the ‘i’ coefficient.

**Example of a complex number:**

**5 + 8i**

The real part – represented by the x – axis; x-coordinate

The imaginary part – represented by the y- axis; y-coordinate

**Iterations shown (using the constant of i):**

1. Z0 = 5 + 8i
2. Z1 = (5 + 8i)^2 + I = -39 + 81i
3. Z2 = (-39 + 81i)^2 + I = -5040 – 6317i

* When drawing the Mandelbrot set, the complex numbers are used to draw the fractal onto a 2D cartesian coordinate system where the co-ordinates of the complex numbers are plotted onto the system according the real and imaginary part of the number. If c belongs to the set being used we draw the point at a location accordingly in the colour of black, or we don’t use another colour
  + 1. **Mathematical rules when calculating Mandelbrot set on computer**

In general, the equation of x^2 + c is referring to an iteration to infinity which is calculated recursively, and this can be done an infinite number of times, the value of x can indeed go to infinity. However, in computer systems, it would be inconvenient to go to infinity as this can make things extremely complicated as computers wont be able to calculate Xn an infinite number of times, but an excessively large number may be seen as a close equivalent to infinity. The two main problems of calculating the Mandelbrot set on computers is that ‘the value of Z going to infinity and its value being evaluated an infinite number of times.’

1. A main rule of using the general equation of zn = z^2 + c is that the ‘absolute’ value of Z, which is actual distance from 0+0i (complex number) being bigger than 2 won’t allow the calculations to be returned back to a place which is closer than 2, instead it will recursively tend to infinity. The system will check whether the value of Z went further than an origin of 2 and if it has, then it will tend to infinity. In general, the system doesn’t have to check for the origin actually being an equivalent of infinity or checking whether a big number is close to infinity. Instead, the system simply checks whether the origin has reached to a value of 2, and if so, it will tend towards infinity.
2. A big question to the previous rule is how many times would the equation of Zn be iterated to check whether it goes over 2 or whether it stays under 2? This simply depends on the seeding value which is plugged into the equation at the start of the iteration. But in general, the equation would only need to be iterated a few times. An example of it being iterated 50 times is shown below from (4)



* Furthermore, the number of iterations isn’t random. The number of iterations for the value of the orbit to reach 2 would depend on the resolution limits of the image which is being calculated and this depends on different factors such as the values being plugged into the equation and the image size. Images can’t be accurate if they have an infinite amount of information such as an infinite value of the resolution or an infinite amount of pixels, they will have a finite amount of pixels with certain resolution limitations to make the image calculatable.
* The reason for the limit being 2 for it to tend to infinity is because after a certain number of iterations, there will be no change in the image and more iterations in the image will simply take up more unnecessary storage due to the values being used, however, there will be no visual change to the image. When zooming into the dynamic and chaotic image of the fractal, the computer will calculate the image with a higher resolution so that the image doesn’t distort as only part of the image is being calculated where the user decides to zoom in.
* According to the resolution of the image and the orbit being equivalent, under or over 2, the complex number co-ordinates are then plotted accordingly onto the cartesian coordinate system.
  + 1. **How to calculate the set:**
* The first step of drawing the Mandelbrot set is to set the equivalence between the pixel coordinates with the image along with the complex numbers being plugged into the equation. As mentioned before, each and every pixel has to represent a complex number as the Mandelbrot set is plotted onto the cartesian coordinate system which is the complex plane consisting of the chaotic and dynamic theories used to plot complex numbers. Then, each pixel is coloured in according to whether it belongs to the Mandelbrot set.
* **2 methods of calculating the set:**

1. The first pixel is defined at the upper left corner of the plane as a complex number and the distance between the pixels is defined. An example would be setting the first pixel as the complex number of -5i + 7 and setting the fixed distance between the pixels of being 0.05, and then the complex number is plugged into the equation of zn = z^2 +c for each iteration to calculate the next pixel and therefore all the other pixels in a recursive order and 0.05 would be used as the constant c.
2. The first 4 pixels are defined on 4 of the corners of the complex plane as the first complex numbers of the set and they are then interpolated between each pixel to calculate the rest of the set.

Overall, the second approach is much easier as the first approach requires a coordinate or a pixel in the centre of the image to represent the seeding complex number of the recursion and the zooming value is also to be determined which is the distance between pixels being used as the constant in the equation.

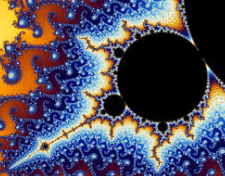
The Mandelbrot function is then calculated by defining how many times you want to iterate the original function and then in programming, a loop would be used for the iteration on the zn equation to calculate each pixel. If it is over the radius limit of 2, then it tends to infinity and then a variable such as ‘CheckInside’ would be set to ‘False’ as the value of Z is further away from 2 from the origin and therefore it wouldn’t belong to the set.

* + 1. **Colouring**
* The original way of colouring the Mandlebrot set is ‘using the value ‘n’ after the inner loop has ended’. In my research, we do this by using the number of iterations (MaxIterations) needed until the value for Z is larger than 2.
* Once the inner loop has ended, where ‘n’ is the loop index, a value of n is generated which is between 0 and the value of ‘MaxIterations.’ If the value of ‘n’ is the equivalent of the maximum number of iterations, we will know that the value of Z didn’t get larger than 2 and therefore the value is part of the set and it can be coloured with any other colour e.g. black. If n is smaller than the value of the maximum amount of iterations then the value of c is NOT within he set and the value of n is mapped to any colour and draw the point with the colour chose.
* However, there are many methods of colouring I have found within my research such as the ‘circular colouring’ algorithm or the ‘histogram’ method.
* (5)

**Histogram Method (Example):**

* Every method starts with a fixed palette of colours to be chosen from according to the algorithm chosen and the parameters for them (e.g. the orbit being over or under the value of 2)

1. One method is to map 0 iterations to the last colour and then apply ‘linear interpolation’ to all the colours in between the points. However, the limitation of this is that as you zoom into the set the points are extremely close to each other meaning they all have a similar value of max iterations when they were plotted – therefore, the colours will be very similar and making the image extremely boring with a lack of colour, which would be underwhelming for the user.
2. A method of fixing this is using a small palette with a limited amount of colours. This would produce a colourful image, such as: (6)



1. However, the problem of having a colourful image with a smaller palette is that one side of the palette is closer to the Mandelbrot set according to the value of Z than the other side of the image.
2. The solution to this is the Histogram method and this is done by counting how many pixels took every iteration to escape to infinity.
3. Each iteration is then mapped to a colour in the palette
4. The method is used to ensure that the first and last colours of the palette are used in order for the image to be more enjoyable to visualise and explore with rather than having a monotone, boring and bland image. In a mathematical perspective, this also gives the user more information in their research by allowing them to be more flexible with the values they are inputting. An example from (7) is shown below:



Histogram method: more vibrant colours.

This is an image from the Mandelbrot set using a simple colouring algorithm

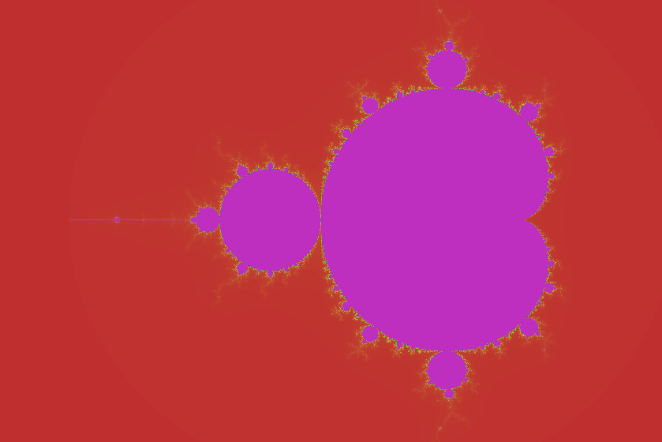
1. The limitation of the histogram method is that these images are lacking a wide range of colours and the rules for this method are still based on the discrete aspect that the colouring is only on the basis of number of iterations required for the value to escape and mapping the colour to the iteration.
2. I may potentially use this algorithm for the colouring my program as it can still be quite useful compared to a lot of other colouring algorithms
   * 1. **General properties of Mandelbrot set**

* Area of the set in the complex plane is finite, it fits inside a circle with the radius of 2, however, the length of the border is infinite. Area is shown by the formula: (x+1)^2 + y^2 < … And a cardioid is used for the circle.
* As the border is infinite you are able to zoom into any part of the border because it’s infinite – the limitation is that you can only do this if you have enough resolution and the values being used don’t overflow or underflow which is determined by the number of iterations used.
* The dimension is larger than 2 but smaller than 3 – and this represents why it’s a fractal as it’s a fractional dimension
* Formula: zn+1 = zn^2 + c (iterations)
* Every point is connected as they’re calculated via iterations from a seed and therefore makes it possible to find a path from any and every point in the set. The advantage of this is that it can speed up calculations significantly when finding the path between points.

* + 1. **Julia Set**
* The Julia Set is another set of fractals closely associated with the Mandelbrot set.
* The main association with the Mandelbrot set is that the Mathematical formula for the iterations of the plotting of the points is very similar: z0 = c, zn+1 = zn^2 + k



* The main difference is that the constant of ‘c’ in the Mandelbrot set is differentiated from the constant value of ‘k’ in the Julia set. K is a constant, complex number.
* The Mandelbrot set can be consisted of the Julia set being added together and the fractals are self-recurring in the Julia set just like the Mandelbrot set (similar equations), but the values of the Julia set are less varied.
* The Mandelbrot set influences the shape of the Julia set by having a value of K inside the Mandelbrot set and if the value of K is closer to the border of the Mandelbrot set then then the image will be of the Julia set will be thinner rather than a thick image consisting of a large majority of a single colour, but if the value of K is further away from the border then the image of the Julia set will be thicker.



**Mandelbrot set fractal:**

* Maximum iteration: 1000
* Online tool: (8)



**Julia set fractal:**

* Maximum iteration: 1000
* Equivalent values as the Mandelbrot set in figure above
* Online tool: (9)

**1.3.7. Real-Life Applications of the Mandelbrot set:**

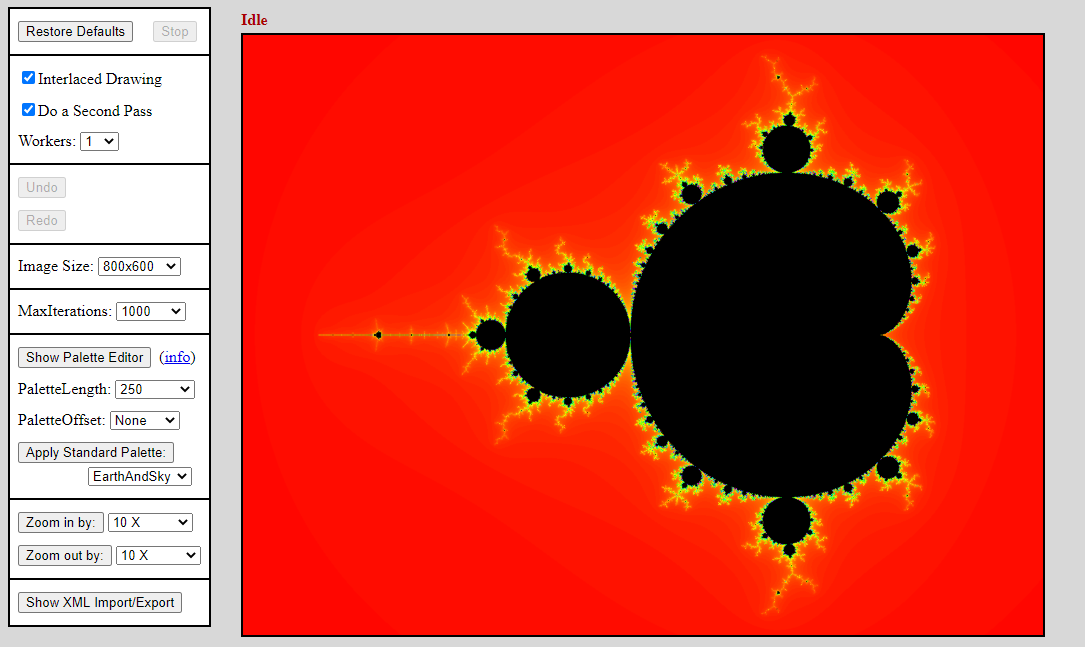
* Creating sample coast lines and landscapes in a wide range of resolutions
* Testing for things such as low flying terrain navigation
* Placements of roads, tunnels and bridge
* Area measurements
* Computer generated mountains (from repeating patterns)
* Of course, fractal geometry
* Timing and sizes of earthquakes
* Variations of a person’s heartbeat
* Prevalence in diseases
* Financial markets – Benoit Mandelbrot was working in this complex field of mathematics when working as a researcher for IBM in the 1960’s; shows losses and profits traders made over a set period of time

Credit: (10)

* + 1. **Existing Examples**

1. **‘Mandelbrot Viewer’**

* Link: (11)
* Screenshot:



* **Description:** This Mandelbrot set viewer was the first one to come on the results page, and this may indicate that it is very popular due to its simplistic and ‘easy to manipulate’ user interface.
* **Advantages:**

1. The user interface is very simple and easy to use
2. It has a palette editor which most systems don’t use, with different settings and variables such as the saturation, hue and brightness of the image and even the palette length with a slider which is easy to manipulate and use.
3. Allows the user to zoom and out of the image with ease and without lag with swift movements using the control panel
4. Allows the user to enlarge the image with the cursor for easier use rather than using the control panel as an alternative option.
5. No distortion in image quality when zoomed in, maintains high resolution of pixels.
6. Allows user to undo and redo actions (perhaps with a stack in the infrastructure of the coding of the system)
7. Allows new users to select a certain value of ‘MaxIterations’, or they have the option to have their own custom input
8. Has a help tab via hyperlink

* **Disadvantages:**

1. Image takes time to load when changing settings in runtime and takes time to respond
2. The Mandelbrot set equation cant be manipulated directly (but can be manipulated indirectly via ‘maxiterations’

**2. ‘Mandelbrot Explorer – David Bau’:**

* **Link**: (12)
* **Description:** Too simple with a lack of features and difficult to manipulate and can cause confusion.
* **Screenshot**:

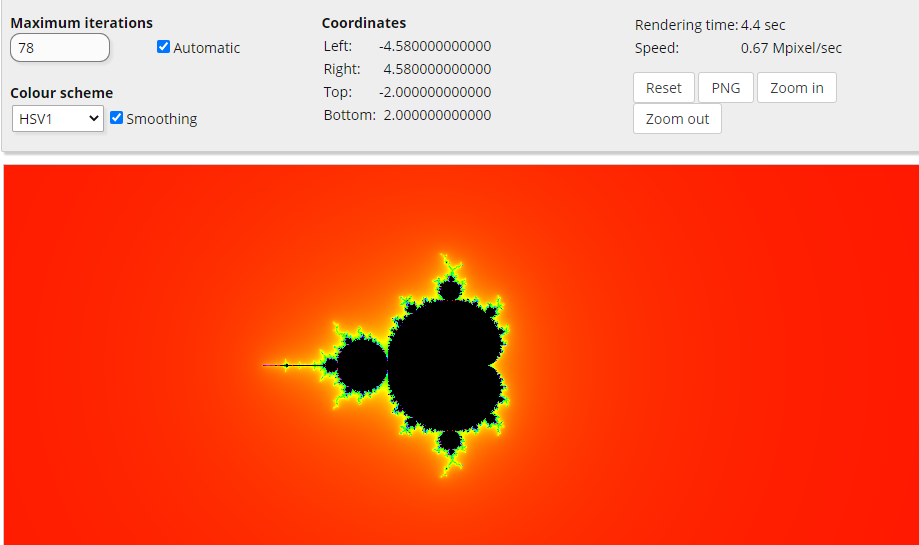
**Advantages:**

1. Very simple
2. Zooming in is very swift and not time consuming when loading the image in real time
3. Provides an extra image with the zoomed in version of original image for comparison

* **Disadvantages:**

1. Lack of features with no control panel
2. Can be confusing with the lack of control over the system
3. No control over the equation
4. No use of inputting values
5. Pre-set image which user may not want as they may be experimenting the set with different values
6. No palette for colour optimisation
7. Lack of manipulation in real time

**3. ‘Online Mandelbrot Set Viewer – ScienceDemos.org.uk’:**

* **Description:** Easy to manipulate with a control panel consisting of a wide range of features
* **Link: (13)**
* **Advantages:**

1. Allows user to input the max iterations to get an accurate image
2. Colour scheme can be manipulated
3. Co-ordinates of image shown by the calculation from Mandelbrot set equation – useful when experimenting with the image, also shown with a long range of numbers for complete accuracy
4. Shows user details such as rendering speed – useful for user wanting to know how long to wait for the image to render and load in real time depending on the input values
5. Allows user to zoom in with cursor as well as button in control panel and allows zooming out as well as dragging in sections of image with swift performance
6. Allows reset to default settings

* **Disadvantages:**

1. Does not allow user to undo and redo actions (could be implemented using a stack)
2. No direct manipulation of equation itself
   * + 1. **Summary of Advantages and Disadvantages:**

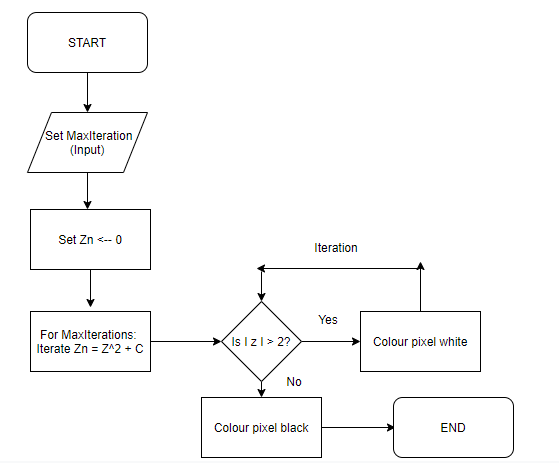
* After investing existing Mandelbrot set programs on the internet, I have come to the conclusion that my user interface must be swift with a minimisation of lag as well as being easy to manipulate through inputs. It would be good to have a palette of colours, however, this is not essential as long as there is an optimised colouring algorithm such as the ‘smooth’ or ‘histogram’ colouring algorithm.
* In addition, I should optimise my program to be able to load each zoom of the image as quick as the hardware can withstand at the given point in time. The rendering speed should be as quick as possible with hardware as one of the variable factors to take into account.
* My program should be able to zoom out as well as zoom in as this can be very useful in case the user mistakenly clicks on the wrong zoom. To make the interface easy to use, there should be a rectangular zoom box or even a mouse function to pan around the screen. The inputs should be calculated correctly e.g. the maximum iterations should correlate with the resolution shown on the fractal image.
* Depending on how far in the fractal is zoomed into and the values being used such as the maximum iterations input, the image should not deteriorate so that the image is pleasing for the user.
* In terms of the actual programming, I will implement an OOP based paradigm/structure to make each and every function robusts I have found out through my research of other Mandelbrot fractal tools, that there are many functions incorporated. With an OOP basis, the attributes will not be affected by other classes with private access identifiers as well as the functions being able to be accessed through other classes. For example, the histogram colouring class (child/sub class) may need to inherit the functions from the main colouring class (parent class)
  1. **Modelling Of The Problem**

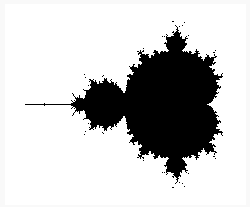
**1.4.1. Flow Charts**

* (I have decided I will be modelling a wide range of algorithms in order to be able to select the most efficient one for the design phase)
  + - 1. **Flow Chart – Mandelbrot Set Iteration Equation (Escape time plotting algorithm; value of 2 is the escape radius)** 🡪 **Zn = z^2 +c**

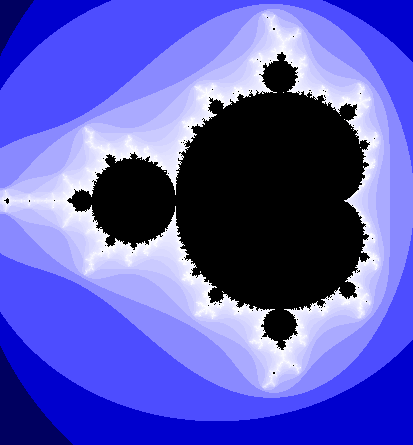
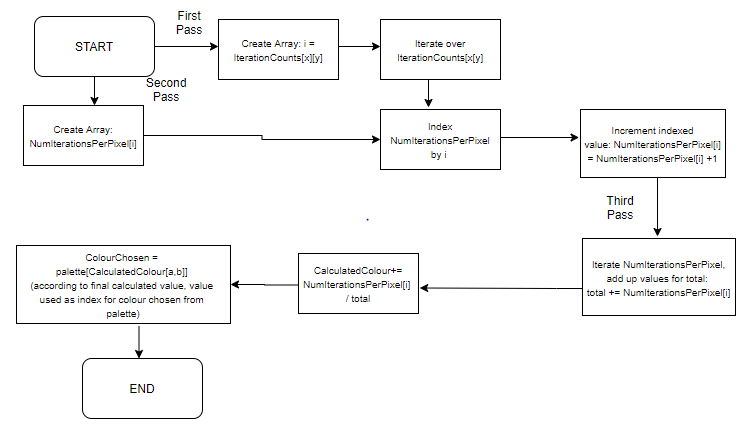


* + - 1. **Flow Chart – Basic Black and White Colouring Algorithm**



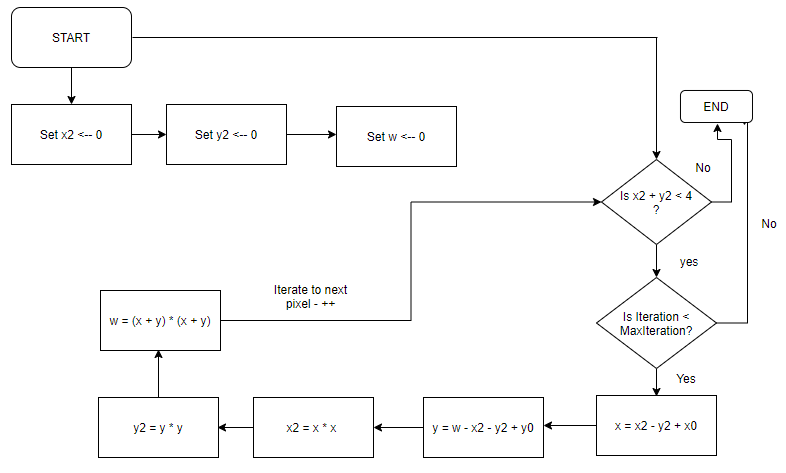


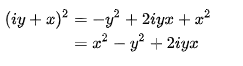
Example black and white colour scheme algorithm : Maximum Iterations = 20

* + - 1. **Flow Chart – Histogram Colouring Algorithm**

**Example:** Histogram colouring method, 100 iterations

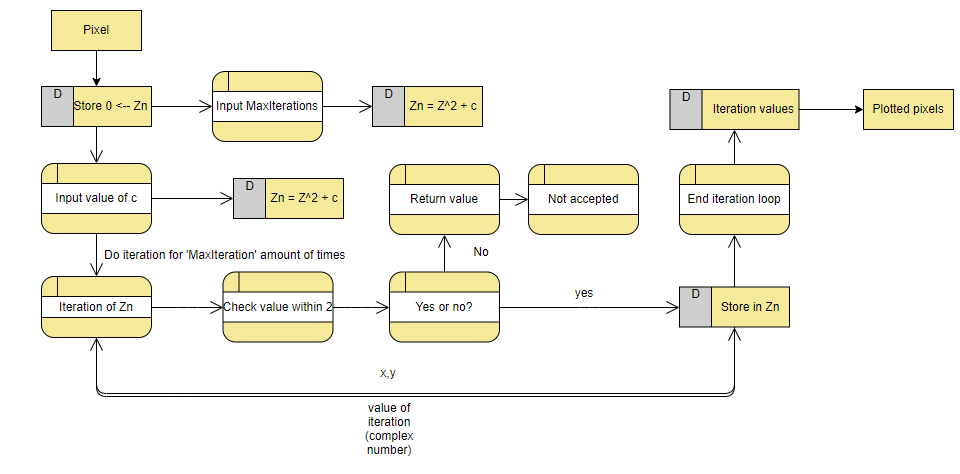
* + - 1. **Flow Chart – Optimised Escape Time Based Algorithm (Smooth Colouring)**



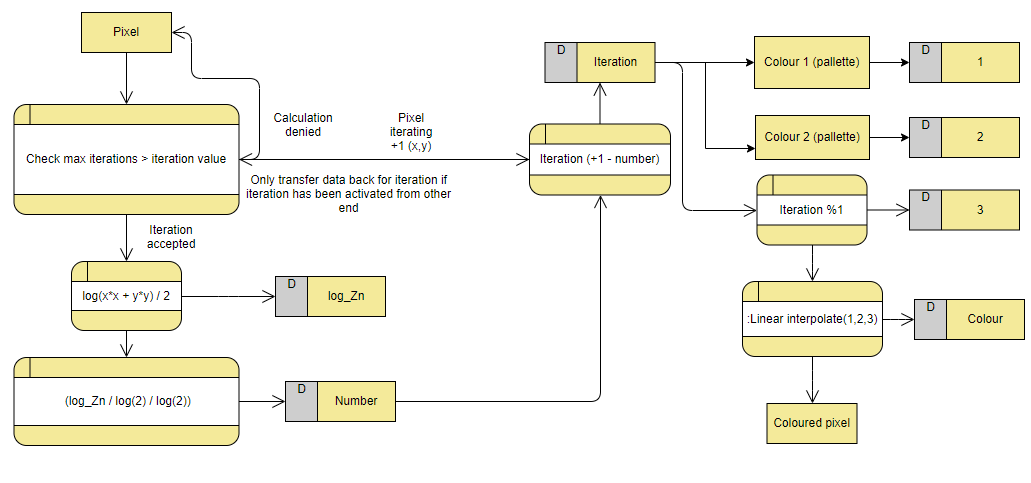
**Based on equation:**

**1.4.2. DFD Models**

* + - 1. **DFD – Mandelbrot Set Iteration Equation**



* + - 1. **DFD – Continuous Colouring Algorithm (Smooth Colouring)**



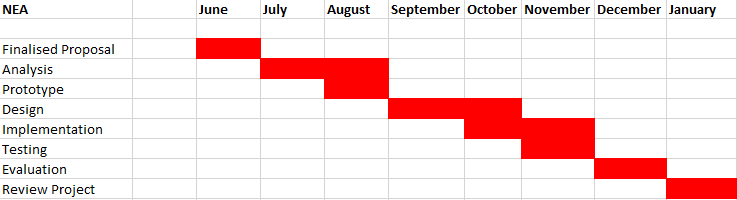
* 1. **Data Dictionaries** 
     1. **Data Dictionary – Mathematical components of calculating set**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name (Data)** | **Details** | **Input, Output, Process, Storage?** | **Data Type** |
| Mandelbrot set equation | Iterations used for plotting of pixels on cartesian coordinate system | Input – equation seed  Process – iterate  Output – plotting of pixels  Storage – file (to use same iteration for next user if user requests) | Complex numbers |
| Maximum iterations | Maximum amount of iteration used for plotting pixel points for boundary on set | Input – number of maximum iterations | Integer |
| Histogram colouring algorithm | Colouring algorithm consisting of vibrant colours and equations | Process – iterations  Output – colour selected from palette | 2D array, integers, float, complex numbers |
| ‘c’ constant in equation | Used for iterations | Process - iterations | Complex number |
| Colour selected | Selected from palette according to value calculated by histogram algorithm | Process – Histogram colouring algorithm  Output – Colour plotted | Complex number/Integer/Float – colour selected according to value |
| Power (index) | Used in Mandelbrot equation | Process - iterations | Integer |
| Co-ordinates | X and Y co-ordinates used to plot the pixels | Input – value into equation  Process – iterate  Output – Value of coordinates, colours plotted  Storage – Stored in array | Complex numbers, 2D array |
| Escape radius | Value used to determine whether iterated value is within the Mandelbrot set (within radius), and if not, its not in Mandelbrot set. Value used is 2. | Process – calculate whether value is within or outside 2. | Integer |
| Undo/Redo action (e.g. previous value of maximum iterations) | User should be able to press a button in order to undo/redo their action | Process – clicking button to undo or redo action | Stack (abstract data type) |

* + 1. **Hardware and Software Considerations**
       1. **Data Dictionary – Hardware/Peripherals Used**

|  |  |
| --- | --- |
| **Process/Action** | **Hardware** |
| Equation manipulation (e.g. maximum iterations) | Keyboard (input) |
| Control panel usage (e.g. changing colour/hue) | Keyboard (input) |
| Zoom | Mouse (input) |
| Undo/Redo action (stack) | Keyboard (input) |
| Viewing the image | Screen/Monitor (output) |
| Changing centre of image | Mouse (input) |
| Details of image | Screen/Monitor (output) |
| Saving values for next use | Cache (array uses cache; e.g. array can be used for x and y coordinates of pixels) (storage) |

* + - 1. **Brief software details**
* I don’t intend to use any particular software for the implementation of my program, but I may have to use certain packages and libraries (source code could be downloaded via soft wares) such as ‘numpy’ as I will need to use complex numbers from it e.g. ‘from numpy import complex’ – this is useful as the Mandelbrot set consists of implementing a heavy amount of functional programming as it’s a maths related program.
* Of course, I will be using Python as my main programming language which would be considered a high level language in the form of a software. As python is very useful when using OOP I intend to implement my program around the basis of using OOP and I will use a file management system in order to store the progress of the user in the visualisation tool such as the values that were used in their previous time of usage.
* The specific libraries I intend to use in my program are Numpy (specialised in mathematics and array usage) – for the usage of complex numbers in the iteration and plotting and of the points on the cartesian coordinate system as it requires complex numbers, PIL – python imaging library or its also known as ‘pillow’ for the image to be viewed, manipulated, saved and opened. Another package I may use (depending on the choice of usage of the Histogram colouring algorithm) is ‘coloursys’ which provides bidirectional conversions of the values of each colour in the RGB set for hue and saturation manipulation for the user which is used via the plotting of the x and y coordinates. Matplotlib is another great library used for OOP and the procedural interface of the module of PyLab within the library could be used for plotting the points and I could use this with the combination of ‘newaxis’ for array manipulation and plotting of points within my code.
* I may use ‘Tkinter’ as an essential GUI for quick and easy ways to click buttons in the control panel for the user e.g. to click the help dialogue to guide new users on how to use the system – further dropdown boxes could also be used via Tkinter and a wide variety of features as it can provide key features to an overall easy to use UI. Also, it hides a lot of complexities from the user which is one of the goals I want to achieve for the UI as the user should be able to use the program with ease even with a complex infrastructure.
* Another potential software consideration could be SQLite to manipulate databases as I may use databases in order to store the progress of the user e.g. to store the value of maximum iterations. However, I may use a file management system as an alternative to databases in order to save the user’s details.
  1. **Critical Path Analysis**
     1. **Gantt Chart**



* It took me a while to think of a project as I went through numerous different ideas. However, I overcame the confusion of picking an idea and decided to go for a project I would be passionate about doing – a maths-related project. I managed to do this within the planned timeframe.
* I have completed the analysis within the predicted time frame with the opportunity to include extra details
* I have decided to create my prototype after the analysis as I will have full knowledge of the Mandelbrot set and I will be able to test out different algorithms and methods of creating the Mandelbrot set
* I plan to begin the design at the start of September which will secure the base of the implementation with the combination of my analysis in order to have a single, secure algorithm to write the code to avoid confusion and time wasting during implementation
* Straight after my implementation I will be testing my program to ensure there are no errors within the code and to also ensure the UI is running smoothly with swift processes and minimal time delay in between processes.
* Throughout the whole development of my project I will make progressive changes in the program in order to maximise the development of the project, therefore, there may be slight differences (slightly early or slightly late) between the predicted time periods for each stage of the program as I will ensure the visualisation tool is the best it can possibly be.
* At the end of the whole project I will evaluate the project to make sure I have met all my objectives and perhaps even extended objectives which I did not plan to meet for further success in my project
* Finally, I will be reviewing my whole project to check for mistakes and errors within he project, whether that be typos or even details I have missed
* I used the Gantt chart as the critical path schedule I will follow as it’s visually easy and quick to see and it provides me with specific time periods between stages in order for me to assess whether I am on track to complete the project within the given time frame.
  1. **Data Volumes Investigation**

1. The type of data used for the equation is complex numbers and I will implement this via Python with the ‘Numpy’ library using the ‘complex’ module within the library.
2. I will allow variations of complex numbers used with different operations such as add, subtract, multiple, divide etc.
3. The values of the iterations generated via the Mandelbrot set equation will need to be in the value of 2 in order to be accepted in to the set; also known as the ‘escape radius.’
4. There will be limitations in drawing external rays since computers usually use 64-bit floating point computation also known as the ‘double’ data type, rather than quadruple 128 bits.
5. Data is limited with maximum iterations as the Mandelbrot set doesn’t escape to infinity due to the limited power of computation despite the vast amount of iterations which can be executed. Technically, if the value is larger than 2 then the set does escape to infinity but there are boundaries and parameters of my visualisation tool.
6. The progress of the user will be saved in a file management system such as the maximum iterations, zooms and image size – if user wishes to use the exact same values in next time of usage.
7. A 2D array will be used for the implementation of the abstract data type of the stack to allow the user to undo and re-do their actions via popping off the actions and pushing them onto the stack. A 2D array could also be used for the RGB values in the Histogram colouring method as well as reading and writing the 2D arrays of the RGB values during file handling.
8. Most of the data will be processed via the user of OOP as this is primarily the type of programming model I will be implementing in my overall code as it provides effective problem solving and flexibility through the use of polymorphism.
9. The data of the Mandelbrot set is limited by the boundary of the bifurcation locus of the quadratic family – the study of dynamical behaviour changes.
10. Data for the colours chosen accordingly to the inputs made will be generated by the Histogram colouring algorithm by using arrays for iteration counts, number of iterations per pixel with different mathematical operations such as floating point division.
11. The data which is input by the user will be denied if it causes an erroneous output by popping up an error message e.g. if the user were to input a logarithmic function and a logarithmic output was generated rather than a complex number, or if the user requests an image size far too large etc… Overall, my program will set boundaries for the extent to which certain variables can be manipulated and I will make the user aware of this with the help button.
12. Statistics and details of the image being viewed will be outputted for the user such as centre value, this data will be essential to the interest of the user, especially if a Maths teacher or someone investing the set thoroughly is using the system.
13. Using the appropriate data, the user should be able to manipulate the brightness, hue, or resolution of the image.

* Overall, in my system, there will be a wide range of data being manipulated and abstract data types such as stacks will be used as well as concrete, regular data types such as integers, complex numbers, floating point numbers, 2D arrays etc. The overall infrastructure of my program will be based around OOP as Python is very useful for OOP.
  1. **Third Party Analysis**
     1. **Third Party Source/End User**
* **Name:** James Heavens
* **Role:** Maths student at St. Brendan’s Sixth Form College
* **Choice Of Person:** I have chosen James as he is a keen Maths student studying at my college and he is passionate about Maths. He is interested in the Mandelbrot set as he finds it interesting and fun to explore the visualisation of it by manipulating the image to get a wide variety of results. He is a great person of choice for my third party source as I can get some strong opinions from him on what would make a good Maths related NEA project, owing to his passionate interest in Mathematics and problem solving. Also, he is reliable in terms of giving constructive criticism and positive feedback about my project throughout the NEA such as stating how to improve my prototype and what I have done well and should keep within in my final implementation of the code. He claims it’s a fun concept to learn about due to the vast amount of colours.
  + 1. **Third Party Interviews**
       1. **Interview A (15/08/2020):**

1. Q: **Should I be able to implement other fractals in my program other than the Mandelbrot set, such as the ‘Lyapunov Fractal’?**

A: It would be amazing to see some other fractals being implemented, but it is not necessary in terms of the project and can be classed as an acceptable limitation

1. Q: **What would you consider to be a good time range (in seconds) to load the image in runtime?**

A: It all depends on the complexity of the final image, but no more than 150 seconds.

1. Q: **Should the image be presented in greyscale, and if not, what colouring method should I use?**

A: It would be boring to have a greyscale representation. I think you should use either the Histogram method or the smooth colouring method.

1. Q: **Should the user be able to save their progress such as their previous Max Iteration values or the image size they used?**

A: It would be helpful to do this.

1. Q: **Should I include a help tab in case it is the user’s first time using the program?**

A: Yes, this should be essential.

1. Q: **Should the user be able to manipulate values in runtime or have it fixed prior to the running of the program?**

A: It could be timesaving to be able to change the values as the program is running and it would be fun to see how the image changes.

1. Q: **Should the user be able to use different types of numbers such as logarithmic functions rather than simple numbers?**

A:This would be fun to use as it adds some flexibility for me as the user, but I am not sure how you would be able to type the logarithmic functions quickly.

1. Q: **Should the user be able to zoom in and out of the image?**

A: Yes, this would make the program more interesting to explore

1. Q: **Should the user be able to manipulate the image size?**

A: It’s not necessary in my opinion, but other users may want this as a function

1. Q: **Would it be essential to include the Julia Set too?**

A: It’s not essential, but if you can implement the Julia Set in the time frame of this project, it would look great.

* + - 1. **Interview: Person B:**

1. Q: **Should I be able to implement other fractals in my program other than the Mandelbrot set, such as the ‘Lyapunov Fractal’?**

A: No, it wouldn’t be realistic within the given time constraints

1. Q: **What would you consider to be a good time range (in seconds) to load the image in runtime?**

A: It depends on the architecture of the computer being used – but anything over 45 seconds would be too long

1. Q: **Should the image be presented in greyscale, and if not, what colouring method should I use?**

A: Coloured – histogram colouring method.

1. Q: **Should the user be able to save their progress such as their previous Max Iteration values or the image size they used?**

A: That isn’t necessary, but you can use that as an acceptable limitation.

1. Q: **Should I include a help tab in case it is the user’s first time using the program?**

A: Yes, definitely.

1. Q: **Should the user be able to manipulate values in runtime or have it fixed prior to the running of the program?**

A: Yes allowing the user to change in run time would be great as the user gets to see the changes in the image.

1. Q: **Should the user be able to use different types of numbers such as logarithmic functions rather than simple numbers?**

A:Its not necessary, but if you could add it, it would be good.

1. Q: **Should the user be able to zoom in and out of the image?**

A: Definitely.

1. Q: **Should the user be able to manipulate the image size?**

A: Yes, that would be good.

1. Q: **Would it be essential to include the Julia Set too?**

A: It wouldn’t be essential but it would be good to add it.

* + - 1. **Interview Person C:**

1. Q: **Should I be able to implement other fractals in my program other than the Mandelbrot set, such as the ‘Lyapunov Fractal’?**

A: If you are able to do so in the time constraint, do so.

1. Q: **What would you consider to be a good time range (in seconds) to load the image in runtime?**

A: 20-30 seconds

1. Q: **Should the image be presented in greyscale, and if not, what colouring method should I use?**

A: Coloured, and whatever colouring method is suitable to design for your given fractal.

1. Q: **Should the user be able to save their progress such as their previous Max Iteration values or the image size they used?**

A: It would be useful, yes.

1. Q: **Should I include a help tab in case it is the user’s first time using the program?**

A: Yes.

1. Q: **Should the user be able to manipulate values in runtime or have it fixed prior to the running of the program?**

A: Allowing the user to be able to manipulate values in run time is good.

1. Q: **Should the user be able to use different types of numbers such as logarithmic functions rather than simple numbers?**

A:It can be useful in certain situations, but its not necessary at all.

1. Q: **Should the user be able to zoom in and out of the image?**

A: Yes.

1. Q: **Should the user be able to manipulate the image size?**

A: Yes.

1. Q: **Would it be essential to include the Julia Set too?**

A: No.

* + - 1. **Interview Person D:**

1. Q: **Should I be able to implement other fractals in my program other than the Mandelbrot set, such as the ‘Lyapunov Fractal’?**

A: No, it is not needed.

1. Q: **What would you consider to be a good time range (in seconds) to load the image in runtime?**

A: 45 seconds

1. Q: **Should the image be presented in greyscale, and if not, what colouring method should I use?**

A: Black and white would be boring, use colouring but it is your choice of which algorithm you wish to use.

1. Q: **Should the user be able to save their progress such as their previous Max Iteration values or the image size they used?**

A: Yes, you should do it in a file organisation system.

1. Q: **Should I include a help tab in case it is the user’s first time using the program?**

A: Yes.

1. Q: **Should the user be able to manipulate values in runtime or have it fixed prior to the running of the program?**

A: Yes, the user should be able to manipulate values in runtime.

1. Q: **Should the user be able to use different types of numbers such as logarithmic functions rather than simple numbers?**

A:It can be useful in certain situations, but its not necessary at all.

1. Q: **Should the user be able to zoom in and out of the image?**

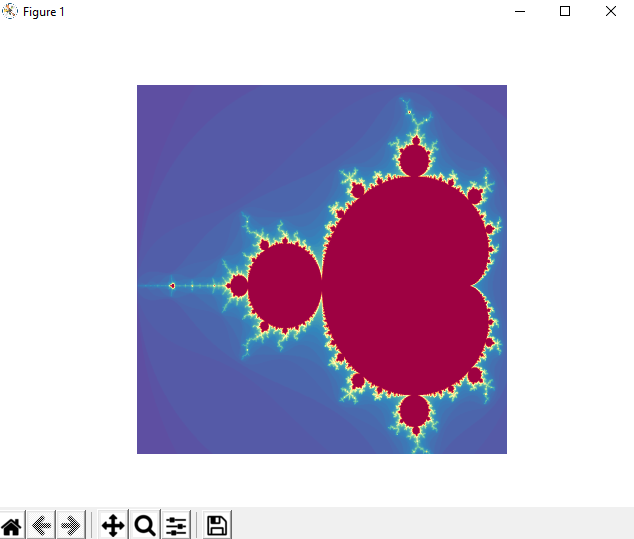
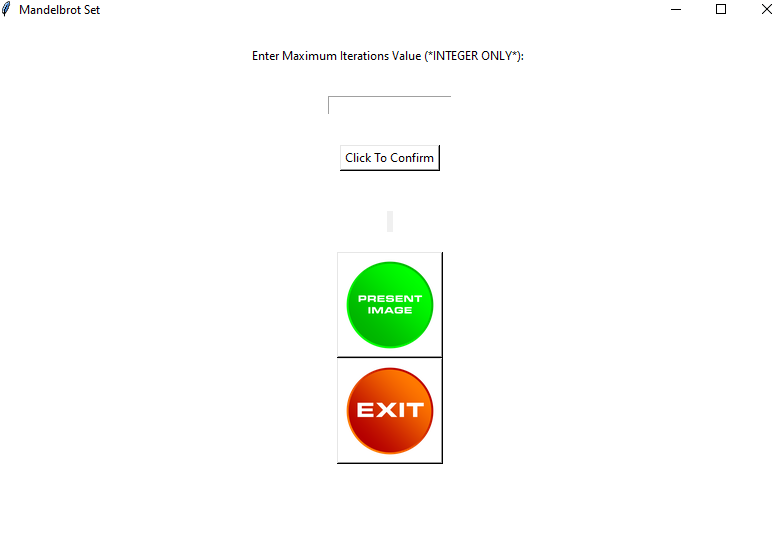
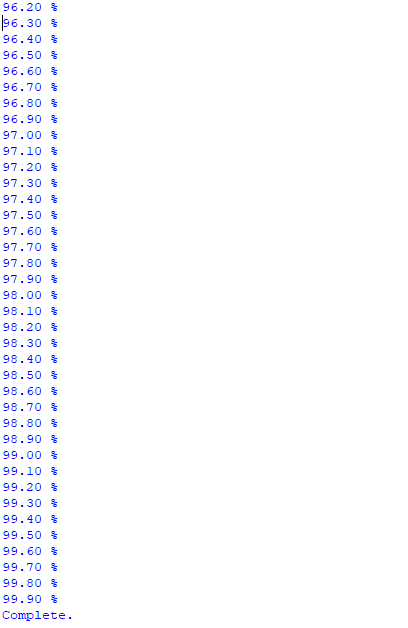
A: Yes.

1. Q: **Should the user be able to manipulate the image size?**

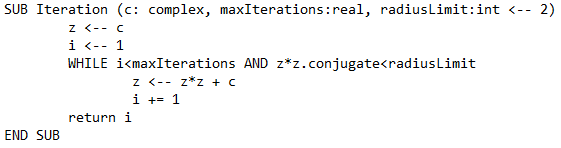
A: Yes.

1. Q: **Would it be essential to include the Julia Set too?**

A: No.

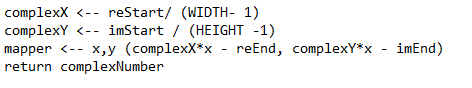
* + 1. **End User Analysis:**
* After some thorough investigation and analysis from the set of interviews. I have found out that the end users will want a coloured fractal rather than a boring greyscale representation of it.
* I will most likely not implement a completely different fractal, however, I will use it as an acceptable limitation and I will still try to achieve the Julia set fractal if it is possible within the time constraint. But my main end goal is to produce a fully working Mandelbrot fractal set visualisation tool.
* The image must be loaded within a maximum of 150 seconds in order for the user to be satisfied. The average time to be rendered is 30 seconds.
* I will use a file organisation system in order to save the user’s progress such as the values they have inputted. I will also try to incorporate a way of saving the image of the fractal for the user too through a file saving system.
* It is essential that I use a ‘help’ tab so that the user can be guided on what they are manipulating and to understand the fractal to a certain extent.
* I will try to implement numbers to be inputted in different ways such as in fraction form, however, from my end user analysis I have found out that it is not necessary to go completely in depth with this function such as using logarithmic functions.
* The user must be able to zoom in and out of the image.
* The image size must be able to be manipulated by the user to view the image in different sizes.
* Linking back to my first question, the Julia set is heavily associated with the Mandelbrot set, therefore, I will try to implement this into my program as an acceptable limitation. However, it is not one of my main objectives as I have to make reasonable decisions within the time constraints stated in my Gantt chart.
  1. **Prototypes**
     1. **Experimenting with Mandelbrot set algorithm:**
* This is the first prototype to my project. I thoroughly investigated and experimented on how to implement the Mandelbrot set when programming, and I came up with a final solution on the mathematical plotting for the program.
* However, this prototype is missing a lot of features as this was just concentrated on perfecting the algorithm and investigating how to implement it.
* I haven’t implemented a user interface, I used the ‘matplotlib’ library as it specialises in plotting images and I thought this would be perfect for my first prototype. It allows the user to zoom into the image, however, it produces a PNG image and it doesn’t truly represent the iterations of the complex pixels when zooming in. The image disorientates and there is a lack of quality when zooming in.
* At this stage, I am still in the process of developing the ‘Histogram’ colouring algorithm, so for the purpose of this prototype, I just used one of the classes of the colourmaps provided by ‘matplotlib.’ This changes the colours of the image and there are various different types such as ‘autumn’, ‘summer’, ‘magma’ and hundreds of others. In this example I used ‘Spectral.’
* The maximum iterations is fixed directly from python and can’t be manipulated in run time. However, the image size can be manipulated in run time due to the default GUI provided by ‘matplotlib’, but this isn’t my desired GUI and I will develop this in later prototypes.
* The end user approves of this prototype and has tested different values of maximum iterations and image sizes with it.
  + 1. **Experimenting with Python GUI and refining unstructured code:**
* 
* In my second prototype I implemented a GUI using ‘tkinter.’ Although this is far from my finished product in terms of the interface, I have built a base for the interface in the final implementation to give me an idea of how the GUI in python works and how it can be representative of my final program for my end user.
* I encountered many problems in this process such as getting the size of the images correct in order to fit inside the buttons, I had overcome this after thorough investigation.
* In my past I have been doing graphic designing for social media logos, banners and headers. I used these skills in photoshop in order to create some basic buttons for the experimentation of how images are implemented onto buttons in ‘tkinter.’
* Another problem I encountered and not yet solved is the development of the loading bar of the progress at which the final image loads at to notify the user how long it is until the image is ready to be presented. However, I have used a simple loop for now for the sake of the prototype rather than using my preferred idea of using a loading bar using ‘tqdm’ (shown in figure above).
* Overall, In this prototype I have learned a lot about developing a GUI in python and this has really helped me to develop an idea for my final implementation. In addition, I have used functions and classes for the buttons used and the main program of iterating the image so that I can reuse these buttons later on in the implementation if I have to refine this section of the code – this has improved the maintenance of my code. It has helped me to really enhance my knowledge of OOP and I am a bit more confident about it now, such as the principles of public and private accessors and inheritance.
* The end user approves of this prototype and he really likes the idea of the buttons used and he says the images I have used are really good for the purpose of the prototype.
* However, I have not yet implemented the idea of changing values (such as the maximum iterations) during run time. However, I have enabled the user to input the value of this in the GUI but the value can only be used for that specific use.
  1. **OOP Basis Analysis**
* My program structure and paradigm will mainly be an OOP approach to make each and every function robust and ensure there is no interferences between subroutines and separate classes. **My main (parent) classes (with subclasses) are as follows:**
* **Main:** This will include the main initialisation of the Tkinter buttons needed in the main menu. The classes of the **login** and **register** will inherit the variables and functions needed from the **main** class for the button prompts from the main menu to the login and registry systems
* **Mandelbrot:** This class will be the main functionality of the overall purpose of my program. The classes subclasses will include **mandelbrotPlotting, mandelbrotRecursion** and **colouring.** The subclasses will use be related to the parent class using the ‘super’ function in Python and therefore inherit the attributes needed for them to be called as objects from calling the subclasses.
* **RGB:** This class will be used to create the sliders and this will be a single class without subclasses as I believe it’s not necessary for a minor part of my project as a whole. The functions however will include **validation** and **swatchColour** to validate the values of 1-255 to stay within the range of the user input and the colouring function will be used to change the colour according to the user input.
* **Stack:** This class will contain the functions: **push, pop, is\_empty, peek, size.** These functions will be of access type public so that the ‘main’ class can access them to zoom into the stack by using the ‘push’ function and using ‘pop’ to remove the zoom boundaries off of the top of the stack to zoom out and then ‘is\_empty’ will be used to check if the stack is empty before calling it as an object accompanied by the ‘peek’ function to check the last item on the stack for the appropriate algorithm to zoom in and out of the stack. Size will be used to return the size of the stack also known as the length of the list.
  1. **Significant changes made after analysis investigation :**
* I have discovered that it would be better to use PyGame as my GUI for the program as the combination of Tkinter and matplotlib have limited features where as PyGame allows easy manipulation of different buttons which I hope to have to allow the user to have further flexibility in the program.
* I have also done a lot of OOP practice which allowed me to make significant changes in the programming structure of my source code prototype allowing each and every attributes to be more robust with private access types.
* I also had high ambitions for the optimisation of my program. Although the resolution is excellent with how much progress I have made so far, the loading times are not as quick as I initially thought, even with the different optimisation algorithms I have experimented with. However, I will still continue to make continuous progress with the optimisation of the user interface.

1. **– Design**
   1. **Design Introduction/Overview**

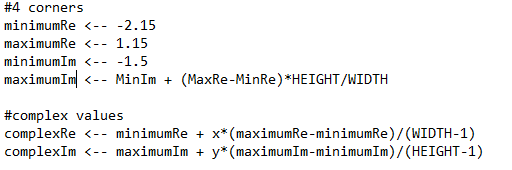
* My design process will give me a good base for the final implementation of my project in terms of the user interface and internal structure of the code.
* The user interface will include a help window, a window for all the different definitions of terms used in the tool for new users, the settings window to change alter the maximum iterations and other features, options to load up previous progress and saving the current image and details, altering the different colours and potentially the option to change to the Julia set (if I successfully achieve my extension objectives or my acceptable limitations). This is the overall structure of my HCI.
* This section will also include the algorithms required for the basis of the program such as for the histogram colouring, the standard Mandelbrot set iteration algorithm, the complex number algorithm for processing the pixels, the process required to load the final image and finally, using the stack data structure to undo/redo certain actions in case the user wants to make changes during run time.
* Also including different types of diagrams such as OOP diagrams for the overall structure of the program (rather than assessing different stages of the program as shown in the analysis section). I will produce flow chart(s).
* One of the most important aspects I will cover is the data structures required such as 2d arrays which may be used to implement the stack.
* Perhaps not a database design, but a file organisation and structure may be needed in order to save the users’ progress onto their system, and to also save their desired final image of the fractal.
  1. **Algorithms**
     1. **Iterations Calculation:**
* This algorithm is used to determine how many iterations are used according to the settings used
* it will calculate how many iterations are needed until the value is outside the Mandelbrot set, and this will ultimately determine the amount of iterations needed to process the whole calculation of the image. It’s processed using complex numbers and the limitation of the radius limit (2); the value of 4 will be used in the actual code, but will use the real value of 2 for the purpose of the algorithm pseudocode.
* The value of c is the value of the pixel for that current iteration within the loop. The variable ‘i’ is used for the iteration to be incremented within the while loop.
* A value of maximum iterations must be set for the standard procedure of the Mandelbrot set, and this value will be determined by the user.
* The value of c would be converted to a complex number using the complex function from a module. The value of z is the starting value of the iteration (0 + 0i), which is conjugated.
* The loop is iterated until the value is outside the radius limitation and maximum iteration, and therefore outside the Mandelbrot set.
* This algorithm primarily works around iterations within the cardioid (limitations) of the system such as the radius limitations and maximum iterations to determine which values are permitted within the Mandelbrot Set
  + - 1. **Iterations Calculation – Psuedocode**
* General Mandelbrot set formula:

**Z = Z\*2 + c**

* + 1. **Pixel to Complex Number Conversion:**
* This algorithm is used to convert the pixels to complex numbers to represent the pixels on the circadian system and it may represent the cardioid of the system (pixel limitations). This algorithm is used because the circadian system only allows complex numbers to be processed.
* There are 2 ways of doing:
* Method 1: the first pixel is defined by the user or the programmer which is a complex number and the distance between each pixel is fixed. E.g. the first pixel may be 0 +0i and the distance may be 0.5.
* Method 2: All the corner pixels are fixed and defined by the user or programmer as complex numbers and then you interpolate between the pixels during the process.
* A variation to the first method is to have the pixel at the centre of the image in order to represent the first pixel of the complex number, this is a defined coordinate. Furthermore, a ‘zooming value’ would be needed; the fixed distance or step between the pixels.
* In the below examples there are different variants of the algorithm, and I will use the most suitable one according to the implementation of the code.
  + - 1. **Pixel to Complex Number Conversion Pseudocode Type 1:**



* This second pseudocode I have produced implements the second method of converting the pixel to a complex number. ‘Re’ defines the ‘real’ part of the complex number and the minimum and maximum borders are defined which are the left side border (minimum) and the right side of the border (maximum).
* ‘Im’ is the imaginary part of the complex number, the minimum or lower part of the border is defined, but the right or upper part of the border (maximum). The maximum real part of the pixel is calculated to help the image not disorientate or stretch – so the real part of the complex number is used for a more appealing and convenient result.
* The next part of the pseudocode is used to calculate the complex numbers from the defined pixels; this is the general formula used in most Mandelbrot set programmes. For specific coordinates (x,y) or defined ones such as a pixel in the top left which would be (0,0) are given their equivalent complex numbers.
* Note (general fixed values): reMinimum = -2, reMaximum = 1, imMinimum = -1, imMaximum = 1
  + - 1. **Pixel to Complex Number Conversion Pseudocode Type 2:**

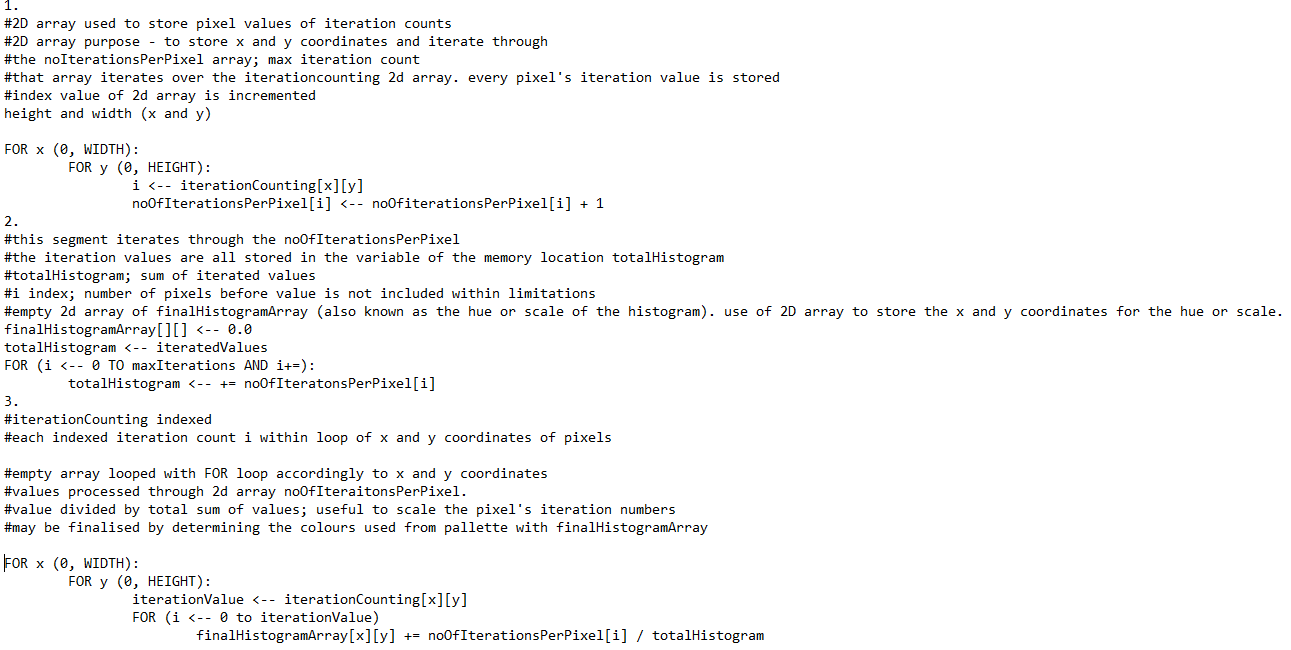


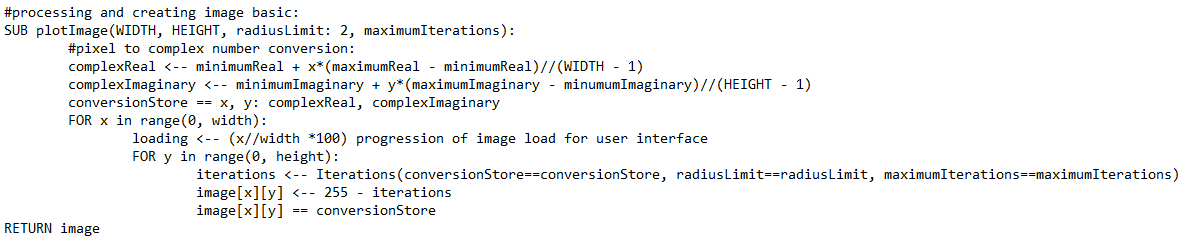
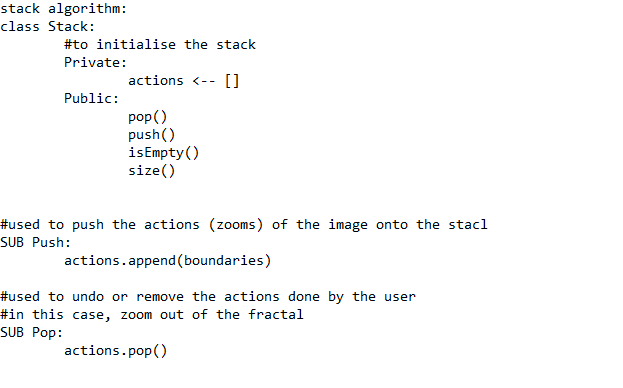
* General formula for conversion:

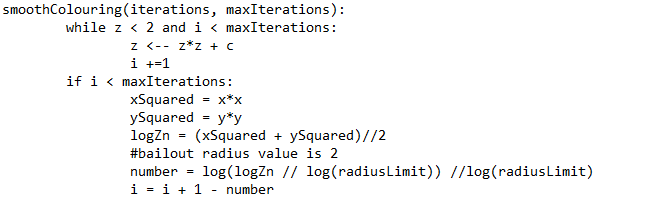


**2.2.3. Histogram colouring method algorithm:**

* The histogram colouring method is a better alternative to the original colouring method of the Mandelbrot set.
* In the basic Mandelbrot set algorithms, the colouring, filling and scaling of colours of the pixels aren’t spread out easily throughout the image and therefore give a limited range of colours and with less quality. This can be a big concern when the processor power of the computer isn’t powerful enough and the iterations being calculated can’t be calculated properly accordingly to the pixel values. In addition, this can be a huge problem when the value of maximum iterations Is high as it can take very long to process and the outcome is underwhelming for the user (as seen in some of my prototypes).
* To overcome these limitations, I have decided to research what would be a better alternative, and I came across the Histogram colouring algorithm. This is a better alternative which results in the colours being consistent according to the iteration values with the pixels.
* The pixel numbers of the iterations are looped through and are counted up to give a final sum, then, colours are chosen accordingly to the iteration numbers of the pixels which therefore give a wider range of colours – this is better for a higher maximum iteration value and computers with basic processors as concern for more pixels is accounted for and therefore giving a wider range of colours, making the image look more appealing to the user.
  + - 1. **Histogram Colouring Pseudocode:**
* **\*This is a remade, commented pseudocode of the histogram colouring algorithm and similar models can be found from various sources, my source of information was – (**



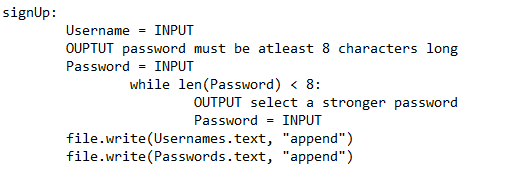
* + 1. **Creating image algorithm:**
* This algorithm is used to create the image and therefore calculates the complex numbers for the pixels using 2D arrays using x and y coordinates to iterate through the height and width. It does this by using the pixel to complex number conversion (which is one of my previous algorithms)
* Then, it would be suitable to store the two complex numbers in the variable of complexStore to make it more convenient to use it throughout the program
* A FOR loop is used to iterate through the pixels of the height and width and therefore the x and y coordinates. I added the extra feature to show the progression of the image loading. It works out how many iterations are needed for the conversion of complex numbers to pixels to present every pixel on the image.
* This is held in the 2D array so that the x and y coordinates of the complex numbers of the pixels are stored accordingly.
* Note: this is only the basic algorithm, and there are other variations used for this which I may implement such as the cardioid limitation checking and other algorithms used to make the colours of the edges of the Mandelbrot set look sharper.
  + - 1. **Creating Image Algorithm Psuedocode:**
    1. **Zooming Stack Algorithm:**
* This particular algorithm isn’t related to the calculation of the image itself but it is useful in case the user has made a mistake and wishes to undo an action such as zooming in or zooming out.
* A stack does the undo function by popping the action out of the stack. A stack adds on the action in order to record the action itself. This is known as the push action.
  + - 1. **Zooming Stack Algorithm (Class) Pseudocode:**
    1. **Smooth Colouring Algorithm:**
* The smooth colouring algorithm follows a specific algorithm [1]
* I have replicated this algorithm via the use of pseudocode in my design
* This formula also known as the normalised iteration count and it can provide a smoother visualisation of the image between iterations.
* A real value of V is used with the Mandelbrot set iteration equation with the ‘potential function’
* User should be allowed to choose smooth or histogram colouring method if I meet the acceptable limitation (extension objective) of implementing south colouring, however, the core objective is still Histogram colouring.
  + - 1. **Smooth Colouring Algorithm Pseudocode:**

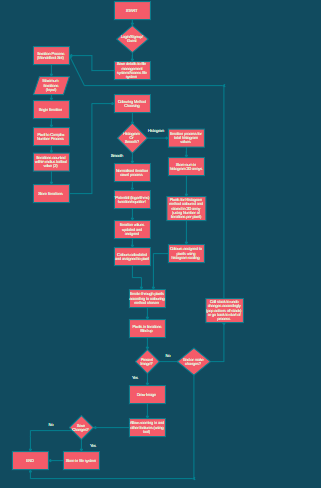
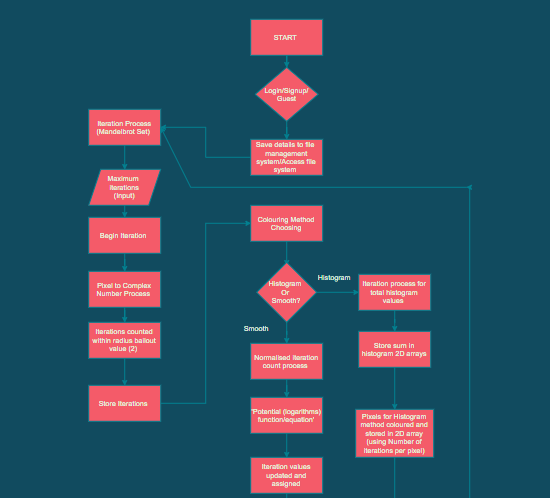


* + 1. **Login Procedure:**
* This subroutine allows the user to login to the system. If they have a username, the system searches for the username in the file using a file management system, if its found then the system will ask for the corresponding password. If the password is right they will be granted access, if not they are allowed to reset their password after 3 attempts or they will be blocked for the time being.
* Also, if the user does not wish to go through a login procedure, they will be allowed to access the tool through guest access. Done through nested if statements.
  + - 1. **Login Procedure Pseudocode:**

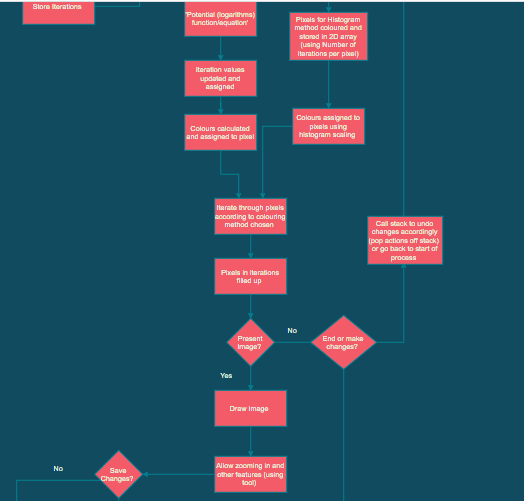


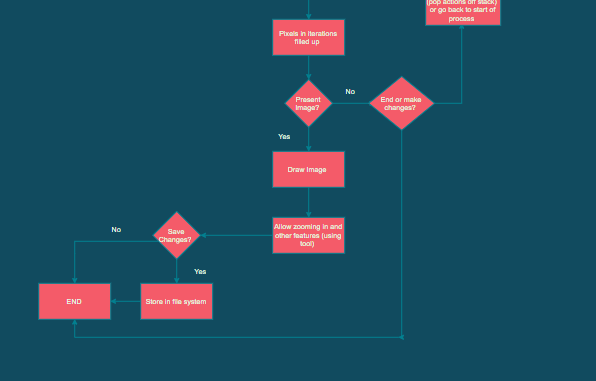
* + 1. **Sign up Procedure**
* Simple sign up subroutine for new users. In the real program, users should be allowed to pin a picture to their profile too.
* Asks for username and password length to be at least 8 characters long
  + - 1. **Sign Up Procedure Pseudocode:**

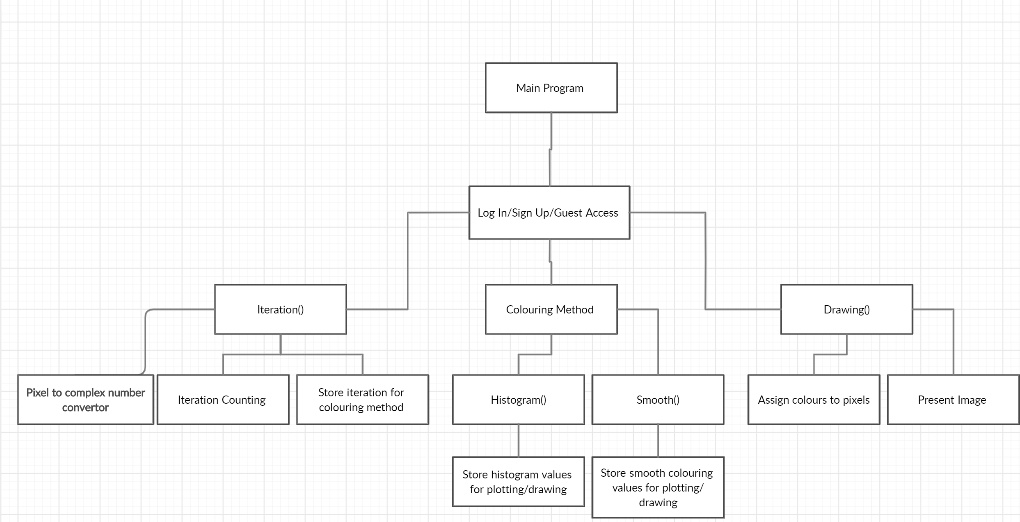


* 1. **Overall Program Structure**
* (screenshots split up as flow chart was too large)
  + 1. **Overall System Flowchart** 
       1. **Screenshot 1: Overview**
       2. **Screenshot 2: Top**

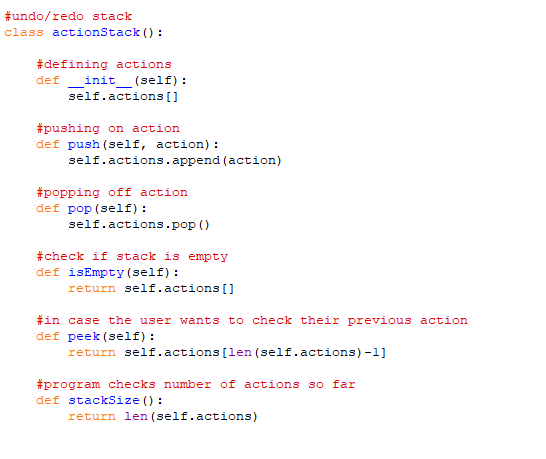
* + - 1. **Screenshot 3: Middle**



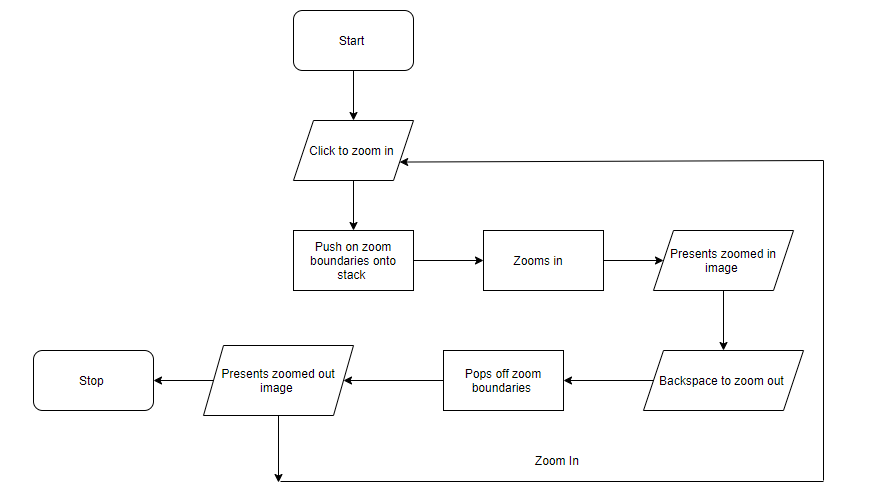
* + - 1. **Screenshot 4: Bottom**
    1. **Overall System Hierarchy Chart:**

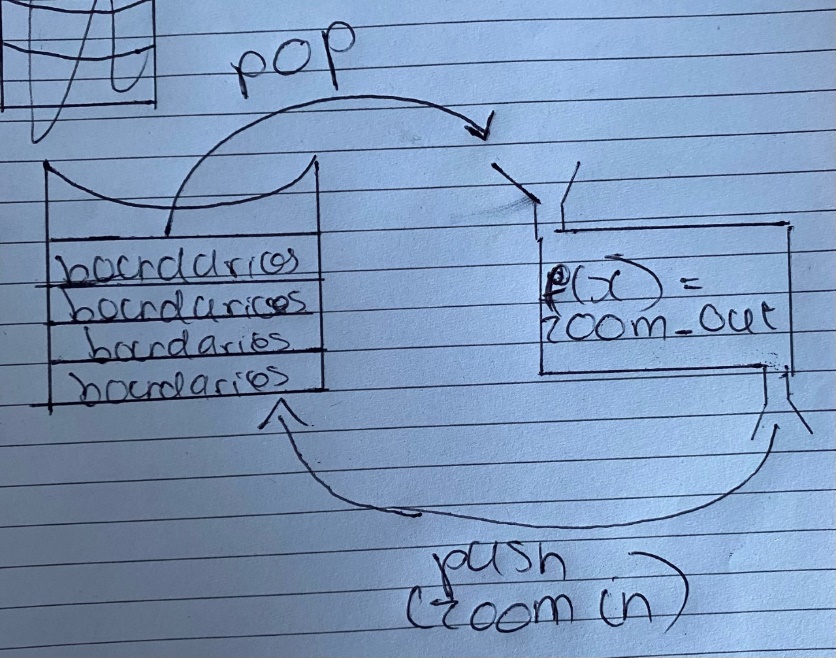


* 1. **Data Structures:**
     1. **Stack:**
        1. **Undo/Redo Stack:**
* The dynamic stack data structure will be easy to implement as there are fixed actions for it.
* The specific use for the stack in my program is that it will allow the user to undo and redo their actions in order.
* This will be useful if the user makes a mistake when zooming in or changing the settings within the fractal tool (e.g. accidentally inputting the wrong maximum iterations value).
* If the user wants to redo an action this will be possible by pushing their previous action back onto the stack, and if they wish to undo their action, the recorded action will be ‘popped’ off of the stack in order to reverse back to the previous setting through their action.
* This will be implemented through the use of a list in Python and the 2 actions of ‘pushing’ actions and ‘popping’ off actions from the stack accordingly. The push action will be implemented through the built-in append list function and the pop action will be implemented through the built-in pop list function.
* It will be appropriate to implement the stack function through classes in OOP.
  + - 1. **OOP Class Template of Stack:**



* + - 1. **Subroutine Stack:**
* This is important to be implemented within my subroutine as there may be recursive subroutines within the program
* This is to ensure the program is keeping track of the process of the subroutines as they are being called such as the local variables and parameters being incorporated within the subroutines
* This type of stack is very important in the implementation of the Mandelbrot set fractal in particular due to the iterations being counted within the primary part of the fractal algorithm. Zn = z^2 + c is a recursive algorithm
* The base case for this recursive algorithm is the radius bailout value which determines whether the iteration value is within the Mandelbrot set or not. The base case would therefore be 2.
* In the design modelling of this particular data structure will hold the local variables such as the actual equation variable (z) and the value being iterated (i). Also, the return address will be held (i). The parameters included in this subroutine would be the complex value of c, the maximum iteration value and the radius bailout value of 2.
* Each time this subroutine is called during the colouring algorithms and other parts of the program, the value will be pushed onto the stack to store it temporarily. When the subroutine comes to it’s end when reaching the base case of the radius bailout value of 2, the information of the subroutine will be popped off.
  + - 1. **Zoom In/Out stack flow chart**



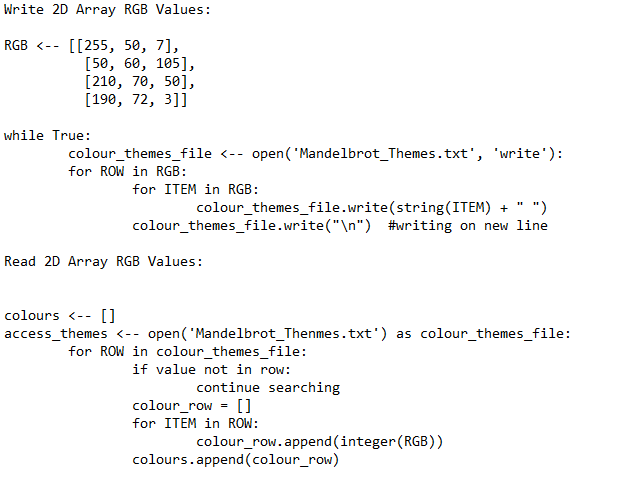
* + - 1. **Basic Zoom In/Out Stack Diagram**
    1. **2D Array:**
* This is the most essential data structure within the program I am developing because it will be included in most of the algorithms being used within the program including a very heavy use of it within the Histogram colouring method algorithm
* This is important as it stores the values of the histogram colouring algorithm and the iteration counts within it so that the algorithm is keeping track of the values being processed in the algorithm. The different passes in the Histogram algorithm will store the previously processed complex number and pass it onto the next step.
* In addition, the 2D array will be used when filling the pixels with the complex numbers and colours accordingly within a while loop to iterate through the x and y pixels for the complex numbers. It will look like this: [x][y].
* A 2D array will also be used to store the RGB values for the colouring algorithm
  + 1. **Hash Tables:**
* Hashing is very useful for large collections of data, particularly usernames and passwords (login details)
* I will use hashing as a method of encryption to encrypt the plaintext of usernames and passwords stored in the text file into ciphertext so that if anyone happens to gain access to the text file with the user login details, the text will be encrypted to a certain extent so it is harder for the hacker to access the personal details.
* With the hashing algorithm, a CSV format hash table can be implemented in the text file to make it more efficient and neat when storing the details. This is done through the use of dictionaries in using {key: value} pairs, or in this case, {username: password} formats.
* I will use ‘salting’ when hashing the passwords as this makes the password less exposed to being hacked into. A salt is random data that is used in the art of cryptography as an additional input to the pre-inputted password. A new salt is generated for each password to make it more effective which will be used with the required hashing algorithm. The generated ‘salt’ and the password inputted by the user are concatenated with the hashing algorithm and the output hash value would be stored with the salt in the file system used to store the details of the user.
* The functions ‘hash\_password’, ‘verify\_password’ and the library ‘hashlib’ will be used to execute this as this specific module also allows for the user of a salt. The defined ‘ascii’ and ‘utf-8’ formats will be used with this.
  1. **File Organisation**
* I am using a file organisation system because in my analysis section, it is confirmed that the majority of my end users would like the program to include a function to allow the user to save their progress.
* Progress will be saved through a text file as the settings used in fractals is in text format such as the maximum iterations, colours, colouring method and image size being used. The user can also save the zoom value (to continue where they left off from the last time they progressively zoomed into the image) and the starting iteration value should also be saved.
* If I can within the time constraints, I will also allow the user to save their currently viewed image in PNG format.
* **How details will be saved:**

1. First of all, the username will input in the sign up process
2. If the username is appropriate, it will be saved into a text file
3. The program will enable the user to input the password if it meets the specified criteria. If it does, it will be saved into the text file
4. The saving process will be done through CSV format in the text file and using IF statements to verify the username and password exist and are correspondingly correct.
5. The verification will be done through indexing the username and password in specific positions in the file e.g. [1], and if they are matched, then the user will be granted access, and if not they will be allowed to update their password or access will be automatically denied.
6. A specific extension to the file will be used to avoid confusion with other files

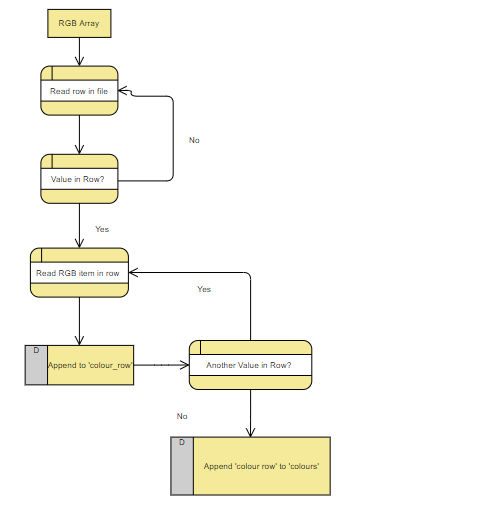
* **Details Saved:**

1. Username and Password
2. Fractal data: maximum iterations, colours, zoom value, scaling speed, image dimensions (window size) etc…

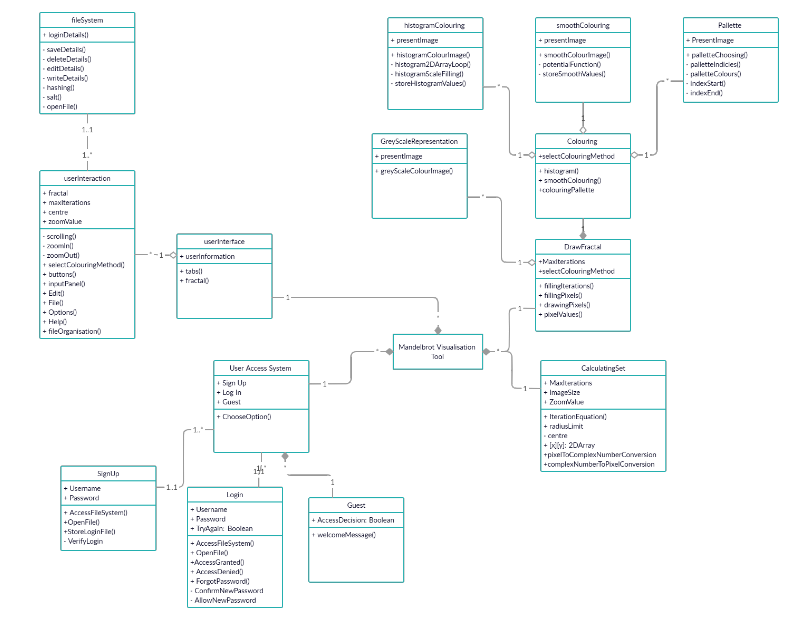
* I have specifically chosen to use text files as they are efficient in terms of storage as they only around 1kb which is very insignificant and therefore would not take long to save or load when the user wishes to login, signup or continue where they left off before in the program.
* Also, hashing algorithms can be used to encrypt the login details in order to prevent hacking and this can be done efficiently in a text file. This will be done with the use of a ‘salt’ to make more effective use of the hashing algorithm in the text file to encrypt the login details of the user.
* There will also be a method to read and write 2D array RGB values for colouring algorithms.
  + 1. **Read/Write 2D Array Data to Text File:**



* + 1. **Read 2D Array to Text File DFD:**



* 1. **OOP Design** 
     1. **Detailed Program Structure/UML Class Diagram**



* This UML Class Diagram represents the main classes which are part of the main ‘Mandelbrot visualisation tool.’
* The 4 main parts of the program are the UserInterface, DrawFractal, CalculatingSet and UserAccessSystem. These have a composition connection with the main program as these are compulsory to produce my program. These all indicate the strongest form of association. For example, the ‘CalculatingSet’ will be needed and contained from the container of the main program because if it is not, there will be no way of representing the accordingly calculated image.
* However, some classes have a less significant aggregation connection such as the userInteraction – this class contains public methods such as ‘scrolling’ for the user. This is aggregation rather than composition as it is these features of the methods and behaviour are not compulsory to run the user interface.
* An even more insignificant connection is ‘association’ which is used with classes such as the ‘fileSystem’, this is because the file system is heavily depended on by the user interface and the interaction the user has with it as the system can function without a file organisation system – however, it is still one of my features that I will implement in my program as seen in my measurable core objectives.
* The attributes of each class are seen to see the main functionalities to make the class operate and these are private, but the methods (behaviours) are made to be public – this is following the general ‘information hiding/encapsulation’ rules of the OOP paradigm as OOP will be implemented within the internal structure of my program.
* Other classes will be not be able to access the attributes or variables of other classes, however they will be able to access the methods as they are declared public.
* This diagram was important for the implementation for the OOP paradigm used.

**Identifiers:**

+ : Public Method/Behaviour

- : Private Instance Variable/Attributes

= Aggregation

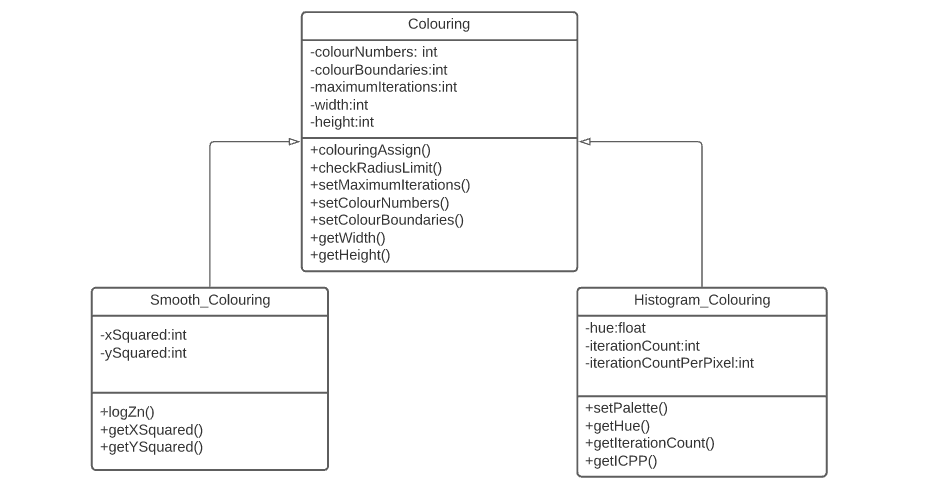
= Composition

= Association







* + 1. **Inheritance Diagram – Colouring Algorithms**
* Inheritance is also used as the characteristics of all the colouring classes would be inherited from superclass (parent class) of the colouring class which are inherited in he hierarchy from the drawFractal class. The attributes which are inherited is the maximum iteration values, and therefore the pixel values which would be calculated from the calculatingSet class. They are inherited as the same variables and therefore the same values will be needed from the superclass for the subclasses (children classes) of the Histogram method, palette and smooth colouring classes. Although, aggregation is shown in the UML class diagram for the containment of classes, inheritance is also used as shown below:
* Inheritance is shown as they would all inherit the same methods from the colouring class such as obtaining e.g. ‘getMaximumIteration’ which is a getter method or a function which would get the corresponding value from the ‘drawFractal’ class or the ‘userInterface’ class where the user would use the input panel to input the desired maximum iteration value. I am using this way of overriding to efficiently use the same values and methods in different parts of the system without the corresponding classes or objects being affected.

- This is also a use of polymorphism as the different ‘colouring’ classes would process objects differently according to their colouring algorithm e.g. the histogram algorithm would process the according maximum iteration and pixel values differently to the smooth colouring algorithm or the Boolean choosing of colours from the palette of various colours. According to the class or colouring algorithm, the objects or attributes/instance variables of the class are being processed differently.

* + 1. **Class Definitions**
* This section is to define the classes I will be making in my overall program because the paradigm I will mainly be incorporating is OOP as I feel it is appropriate for the modular structure of my program with many mathematical algorithms and subroutines being used. I believe the classes will make each and every function more robust by only allowing private access to attributes and perhaps public access to functions which must be called from other classes through inheritance with **‘super’** in Python.
* These class definitions are strictly just for the plotting of the Mandelbrot algorithms. In the login/registry system – the main class is ‘menu’ as the parent class and the subclasses are ‘login’ and ‘register’ which inherit the attributes from the main menu.
  + - 1. **Mandelbrot\_Calculate**

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Description** |
| Private | z | Integer | Initial value of the iterative Mandelbrot algorithm |
| Private | Scale | Float | How much the fractal zooms in with the mouse click |
| Private | Boundaries | Float (list) | To keep track of the boundary points when zooming in/out |
| Private | Maximum\_iterations | Integer | Maximum iterations used as a boundary for the Mandelbrot iteration |
| Private | c | Integer | Constant for Mandelbrot iteration |
| Private | Radius\_limit | Integer | Boundary for which the values of iteration escape the Mandelbrot set |
| Public | Mandelbrot\_Iteration | Function | Main Mandelbrot algorithm |
| Public | Check\_radius\_limit | Function | Used to check the radius limit |
| Public | Mandelbrot\_recursion | Function | Used to return recursive value of set |
| Public | Get\_colour\_themes | Function | Read and return RGB values of colour theme 2D array from file |
| Public | Get\_max\_iteration | Function | Read the maximum iteration value from file and return |
| Public | Get\_aspect\_ratio | Function | Return aspect ratio for fractal plot |
| Public | Get\_image\_dimensions | Function | Read and return width and height values from file |
| Public | zoomInOut | Function | Call the stack class for zooming purposes |
| Private | RGB 2 | 2D array | RGB values in 2d array |

* + - 1. **Mandelbrot\_Colouring (Histogram algorithm/default theme)**

This class will inherit from Mandelbrot\_calculate

|  |  |  |  |
| --- | --- | --- | --- |
| **Access type** | **Field Name** | **Field Type** | **Description** |
| Private | Colour\_number | Integer | Used to track number of colours being (RGB sets in 2D array) being used |
| Private | colours | Float | Final colour blend value used with for loop inside a list format |
| Private | Colour\_boundaries | Float (list) | Boundaries of Mandelbrot colours |
| Public | Linear\_interpolation | Function | Standard mathematical linear interpolation algorithm for histogram colouring algorithm |
| Private | V0 | Integer | One of the values to plug in for linear interpolation |
| Private | V1 | Integer | Value to plug in for linear interpolation |
| Private | t | Integer | Value to plug in for linear interpolation |

* + - 1. **Assign\_colour**
* Inherits from the Mandelbrot\_colouring class as well as Mandelbrot\_calculate class
* It will return the value of (0,0,0) at the end as the RGB
* The same functions from Mandelbrot\_calculate are used such as check\_radius\_limit, Mandelbrot\_iteration and the attributes of maximum\_iteration and z.
  + - 1. **Complex Conversions**
* This will inherit from Mandelbrot\_calculate for the attribute of boundaries as well as the image dimensions of height and width returned. Linear interpolation also used from colouring class

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Description** |
| Private | Minimum\_x | Integer | Minimum x value |
| Private | Maximum\_x | Integer | Maximum x value |
| Private | Minimum\_y | Integer | Minimum y value |
| Private | Maximum\_y | Integer | Maximum y value |
| Private | Bounds | Integer (list) | Used to track the new boundary values of complex conversion |
| Private | Real | Complex | Real part of complex number |
| Private | x, y | Complex | Complex value of coordinate x and y returned |
| Private | Imaginary | Complex | Imaginary part of complex number |
| Public | Convert\_to\_complex | Function | Used to convert between pixel and coordinates |
| Public | Convert\_from\_complex | Function | Used to reverse the conversion |

* + - 1. **Mandelbrot\_draw**

This class will inherit from all the classes stated above as the algorithms already stated will all be called in this class

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Description** |
| Private | Screen | Array | Height and width to show fractal |
| Private | Current\_boundaries | List | Boundaries in list to track current boundaries of zoom in/out |
| Private | zoomInOut | Stack | Used to call the stack class to zoom in or out |
| Public | Plot\_mandelbrot | Function | Nested loop to plot through x and y coordinates |
| Public | Zoom\_In\_click | Function | Called when user clicks on the Mandelbrot fractal and then it zooms in |
| Public | Zoom.pop() | Function | Used to pop off boundaries off the stack |
| Public | Zoom.push() | Function | Used to push on the boundaries on the stack |
| Public | Zoom.isEmpty() | Function | Used to check if stack is empty when zooming in or out (push/pop) |
| Private | Image\_save | List | Used to save parameters of the current zoomed in fractal to track it for zooming in/out for later usage |
| Public | Zoom.peek() | Function | Used to assign the last used boundaries to new zoom box for tracking |
| Private | Use\_parameters | List (float in list) | Used to check if the parameters have been used previously during zooms. |

* + - 1. **RGB\_Slider**

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Description** |
| Private | Red\_slider | Object | Red slider object |
| Private | Blue\_slider | Object | Blue slider object |
| Private | Green\_slider | Object | Green slider object |
| Private | Red\_input | Integer | Red slider input box |
| Private | Blue\_input | Integer | Blue slider input |
| Private | Green\_input | Integer | Green slider input |
| Private | Swatch | Object (colour changer in Tkinter) | It changes colour as the slider is moved or the value input by the user |
| Private | Hex\_colours | Hexidecimal values | Hexidecimal RGB values |
| Public | Colour\_setting | Function | Used to assign the colour in the swatch box |
| Public | Range\_validation | Function | Used as an exception/input handling tool so user does not escape values of 1-255 when inputting RGB value |
| Public | RGB\_main | Function | Used to call all other functions in class at once for running purposes |

* + - 1. **Stack**

- Used in the overall Mandelbrot plotting class for zooming in and out of the fractal of the image

|  |  |  |  |
| --- | --- | --- | --- |
| **Access Type** | **Field Name** | **Field Type** | **Description** |
| Private | Items | List (empty) | Empty list for when values are being pushed in and popped out of the stack |
| Public | isEmpty | Function | If statement used to check if the stack is empty to prevent underflow errors when trying to pop out of the stack |
| Public | Push | Function | Uses a standard append function of a list to push on boundaries of zoom box for the fractal zooming in |
| Public | Pop | Function | Return\_bounds variable used within function to assign the last value on stack (top of stack) to a variable to track the boundary values to zoom out of the fractal. Pop will use standard pop list function |
| Public | Peek | Function | Peek used to track the top of the stack which is the last value used, it will be used to zoom out of the stack as well as zooming in |
| Public | Size | Function | Used to return the length of the list (size of the stack) to prevent overflow or underflow errors later in the Mandelbrot plotting class. |

* 1. **Data Dictionaries (Updated):**
     1. **File Organisation Data Dictionary**

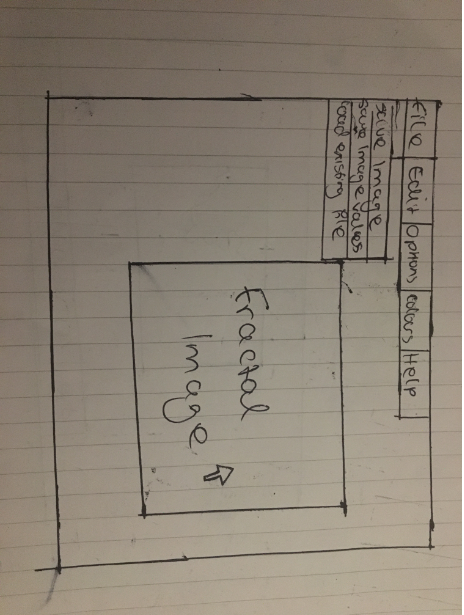
|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Description** | **Data Type** | **Size/Range/Criteria** |
| Username | Username created by user on signup | String/Text | Max 20 characters |
| Password | Password created by user on signup (plaintext) | String/Text | Minimum 8 characters, must include numerical characters (digits 0-9) |
| Encrypted Password | Encrypted ciphertext of the password (most likely implemented using hashing and salt values) | String/Text | Dependent on cipher used and the likelihood of collisions and errors with hashing. |
| Maximum Iterations | Maximum iterations value saved by user | Integer/Real | Must be some sort of number and a valid value. (Error message will show if the value is too big, but if they wish to save anyway, they may do so as the loading time of image is dependent on processor of hardware along with other specifications) |
| Colouring Method/Colour theme | Colouring method chosen saved | Boolean | Selection from either histogram method or colour from palette (or smooth colouring if extension objective met). Selected from palette will be a colour theme e.g. fire (red), ocean (blue) |
| Scaling Speed | Obtain the speed of which the rectangular zoom box changes in size | Integer/Real | Recommended value Is 0.1. The larger the image dimensions, the smaller the scaling speed required. |
| Fractal Choice | This will only be implemented if I meet the extension of implementing the Julia Set. If so, the user should be able to save what fractal they were using. | Boolean | Will only be used if the extension objective (acceptable limitation) of using ‘Julia Set’ fractal is implemented – otherwise no point in selection of fractal. |
| Height and Width | This will determine the window size/image dimensions of the fractal choice with the options being industry standard image sizes. Chosen from a drop box in chosen GUI. | Boolean: Choice  Integer: Dimension | Options (HEIGHTxWIDTH format)  300x300  512x512  900x900  1024x1000 |

* + 1. **Key Variables (Overall Program)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Description** | **Data Type** | **Size/Range/Criteria** |
| maximumIterations | Maximum iterations value of the fractal | Real/Float | The number input will be a ‘real’ or ‘floating’ number or even an integer, however, can be of various formats e.g. logarithmic if this extension objective is met. Number must be realistic as the loading time of the image is dependent on this and therefore their hardware. Error message will appear if unrealistic number input e.g. 12,000. |
| zoomValue | How much the user can zoom into the image (calculated using iterations) | Real/Float | This value can be used in the input panel as an alternative to panning around and zooming into the image remotely. |
| colourMethodChoice | Can either be histogram colouring or using a palette and choosing remotely from a range of different colours from a fixed number of colours. | Boolean | Smooth colouring also will be included if extension objective met. |
| Centre | Centre value of fractal | Complex Number | This will usually be a default value of 0+0i |
| Apply | Applying values in the input panel and various other functions. | Boolean | User will be able to apply chosen settings as long as they are not erroneous. |
| Save | Saving values of fractal to the file organisation system. | Boolean | Will enable user to save if values are appropriate. |
| Resolution | Changes the resolution of the image quality | Integer | Also known as the chunk size as the pixels are enlarged and resolution of fractal decreases. Limitation is when the whole image is blank. |
| Power | To the power to which the Mandelbrot fractal equation is indexed to – z^x; ‘x’ being the power | Integer | Must be realistic, if not then error message will appear to warn the user. E.g. power = 2000 |
| startingCoordinate | Needed to determine the start coordinate of the image | Complex Number | Can manipulate, but default value will be 0+0i |
| pixelsArray | Used to store the pixel values into the array. 2D array used as there is two variables of x and y coordinates. Needed to store the pixel values throughout the iterations for the program to track it and use in other classes and subroutines of the program. | 2D Array – [x][y] | Dependent on pixel values of image. |
| finalHistogramArray | 2D array used for selecting the right colours according to final pixel values of the histogram algorithm (other 2D arrays also used within the algorithm for x and y coordinate pixels) | 2D arrays – [x][y] | Dependent on the algorithm. |
| Username | User login. | String/text | Max 20 characters |
| Password | User login. | String/Text | Minimum characters 8, must include numerical characters – digits 0-9. |

* 1. **IPOS (Input Process Output Storage)**

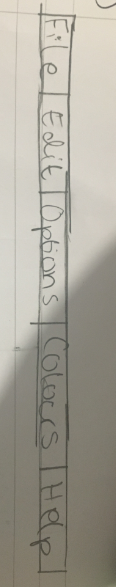
|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Process** | **Output** | **Storage** |
| Maximum Iterations | Iterate through pixels according to value – determines the zoom limit of fractal and the maximum iterations to go through | Fractal Image | File – if user has requested to save the value. |
| Power | Iterate through pixels according to power | Fractal Image | File – if user requests |
| Zoom Value | Allows the user to zoom into the image (if the user chooses to do this rather than panning around the image with the mouse) | Zoomed in fractal image or zoomed out. | Zoom value saved to file if user requests |
| Zooming in/out/panning around image – Mouse | Same process as zoom value input but using a mouse | Zoomed in/out fractal image | Zoom value may be recorded accordingly to the zoom done by mouse for user to save to file |
| File directories | Save the file with details (e.g. login details) to where the user decides to on their PC | Saved details | To required file |
| Palette colour selection | Change fractal colour according to chosen palette colour chosen | Fractal image with corresponding colour | To file if required |
| Image size | Change the image size of fractal | Fractal with corresponding image size | To file if required |

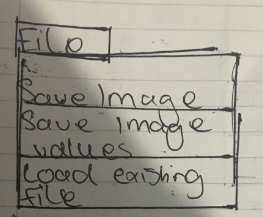
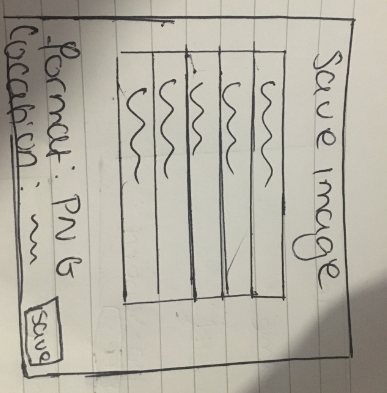
* 1. **API Overview**
* **Brief Overview** - In my early prototypes I discovered that Python’s original GUI ‘Tkinter’ was not suitable for the flexibility of the visualisation tool in my project. This is because there were limited functions when combining the Tkinter GUI with the matplotlib library to provide a coordinates axis structure for the pixels being converted between complex numbers.
* **Pygame** - I have now discovered that ‘pygame’ is suitable. Its a cross platform set of Python modules used to produce swift and well-designed computer games. However, I have found that it is useful for this visualisation tool as it provides suitable abilities for me to add functions such as palettes for colour selecting. Most importantly, the main reason I chose this is because it is effective for me to program the visualisation tool in order for the user to zoom in/out and pan around the image swiftly with quick response times (depending on the processor in the hardware of the user’s computer). This is useful as my end user analysis suggests that the majority, if not, all the users want a function of zooming into the image, and this will be done through the input of the flexibility using a mouse.
* **Tkinter -** Although Tkinter will not be used for the interface itself in terms of the user zooming in, it is a very useful library with many modules allowing buttons to be implemented which will be used when the user has to select colours, saving files, changing values etc.
* **Numpy -**  In certain areas of my program in order for my program to maintain a structured organisation, I will use the ‘numpy’ library which specialises in working with arrays. This will be important as arrays will be used when calculating the pixels itself, the algorithm used for implementing and drawing the pixels and most importantly the various different 2D arrays which will be used as dynamic data structures within the Histogram colouring method algorithm. Numpy allows the programmer to manipulate high performance and complex multi-dimensional arrays. This will be useful for me to keep track of the arrays within the objects and classes in my program, and the Histogram colouring method was recommended by most end users in my primary research of my analysis, therefore it will be useful.
* **Matplotlib -** Matplotlib is also a useful library for this project as it specialises as a plotting library in terms of Mathematics. My project consists of plotting algorithms in a visualisation tool using complex mathematical algorithms. It also provides an object-oriented API which work with other embedded GUIs (such as Tkinter, as seen in my prototype). The structure of my program will be consisted of OOP in certain areas and this will embed nicely with the internal structure.
* **PIL** - The PIL (Python Imaging Library) module consists of allowing the user to open, save and manipulate different image file formats. This is essential when the user wishes to save the current image they are viewing and when they want to decide in what file format e.g. PNG.
  1. **HCI** 
     1. **Overall Interface Structure:**

The overall window of the Mandelbrot Set fractal tool. Implemented through API of ‘Pygame’ Python library.

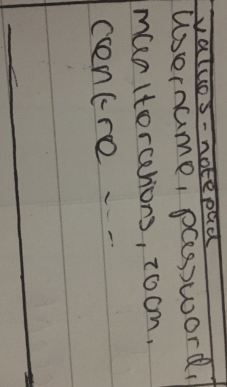
Tabs at the top of the window. Should be implemented through Tkinter if suitable in implementation process to provide simplicity. Example of how an open tab will look like – expanded ‘Files’ tab shown.

The Fractal Image will be within the interactive window of the visualisation tool. The arrow is used to zoom into the image and pan around it to view the details of the zoomed in fractal. The size of the image can be manipulated in the Edit tab at the top of the window. E.g. can be changed to 600x400 from 200x200. This is dependent on the overall size of the window itself.



* + 1. **Analysis of Tabs**
* The file tab enables the user to save values of the image to the file, load existing values from the file in order to load up the saved image they have previously saved. Also, there is a feature allowing the user to save the image by an internal screenshotting function within the code (perhaps through the PIL library specialising in image manipulation within Python) and then enabling the user to choose which file directory to save the file into for later use.
* The edit tab is the most important tab and it is used to manipulate the Fractal image. For example, you can change the maximum iteration value, the centre value and most importantly, the zoom value (if the user prefers to type in the value rather than using their mouse to pan around the image).
* The options menu is used for account settings (e.g. login to the system), undo and redo actions (e.g. if the user wants to zoom out to the image previously loaded – undo/redo stack is used), resetting the image to default and closing the window once the user is done with the usage of the tool
* The colours tab is used to manipulate the colours of the image. This may be histogram colouring or palette settings (using sliders for easier and more in-depth manipulation). I will also implement smooth colouring if this extension is met.
* The help tab is very useful in terms of instructions on how to use the program and what certain terms and definitions mean.
* These tabs are located the top of the screen for the simplicity of easy manipulation and access (one of my measurable core objectives). For example, these tabs are placed in similar positions such as in a ‘Word’ document – therefore this makes the user familiar as soon as they open the program. Also, these tabs and buttons shall be created through the use of Tkinter which is Python’s specialist GUI for features such as implementing buttons (will be linking with Pygame). Additionally, if the user wishes to close an opened tab, they can use the ‘close’ button if they are in depth into the tab directory (e.g. if they have selected file and then save), however, if the tab itself if just open (as seen in the ‘overall’ HCI diagram above) the user should be able to click anywhere on the screen to remove the tab from the screen for the time-being.
  + - 1. **File Tab**

Other files within the user’s file directory

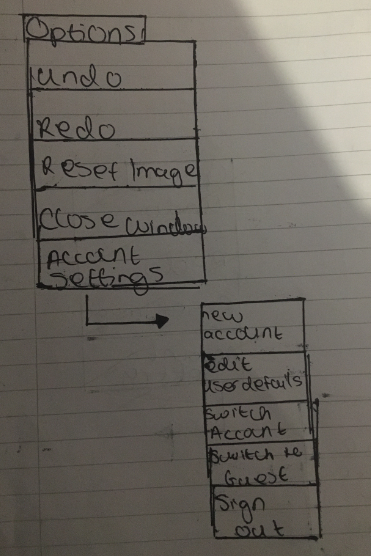
- The file tab consists of 3 options being save image, save image values and load existing file.

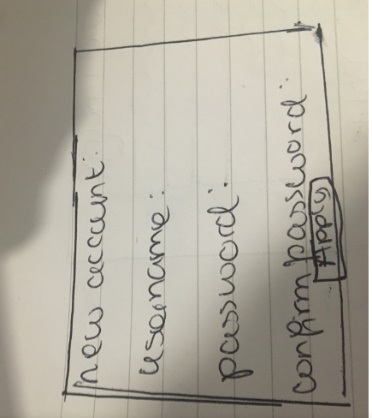
- Save image allows the user to save the image in any file directory on their PC that they want. The format is automatically set to a PNG image. The location is chosen by the user. A save button is used to save the screenshot in the file directory they have chosen.

- The ‘save image values’ option is used to access the file organisation system itself. It will save the image values as long as the user has logged in, it will not be able to do so with a guest account because if the user tries to log in with guest again, the program can not recognise the user. The values will be saved on ‘notepad’ as a text file in a CSV format as this is quick to load. The details saved will be maximum iterations, centre, zoom and other values stated in the data dictionary. This will be done through the save button.

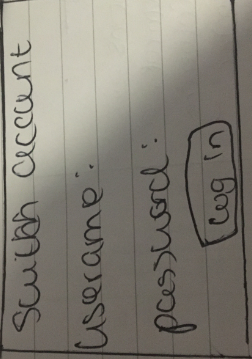
- The ‘load existing file’ will directly load the values of the image which are saved by the user in the text file. The program will then use these values to load the image.

* + - 1. **Options Tab**

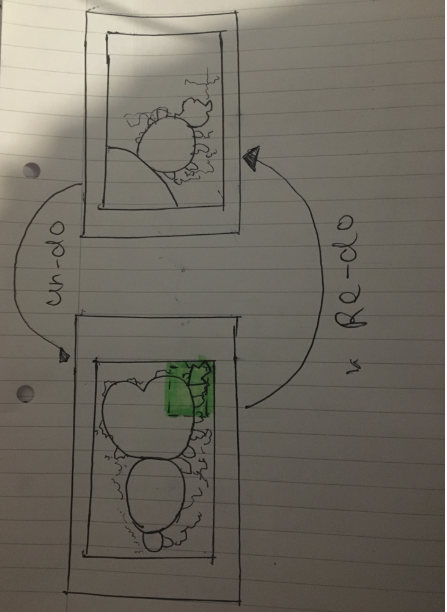


* The option tab consists of undo and redo buttons. These are used if the user wants to zoom out the previous image that was loaded or if they would like to zoom back into the image into the section they previously zoomed in on. For quick access to these functions, I will include a shortcut to these (e.g. CTRL + R to redo rather than the lengthy process of opening the options tab). These actions will be done through the use of a stack in order to pop off actions from the stack if they wish to redo the action or undo the action. Every action shall be placed on the stack to keep track of what the user is doing.
* The ‘reset image’ is used to reset back to the original image with the default values from when the program was first loaded up.
* The ‘close window’ button is used to close the whole program
  + - * 1. **Login and Registry System**
* ‘Account settings’ is used to manipulate the login system of my program. There are sub-options used as this is a varied option and lots of functions can be used. A new account can be made by inputting the username and confirming the password with the correct criteria of the password (as stated in my data dictionary). The username and password of the new account shall be saved in the file organisation system via a text file in a CSV format. This file will be opened and access within my program whenever the user login details are needed. In addition, the user can edit their details with the same procedure and an error message will appear If the username or password don’t match the criteria or if the details are the same prior the editing selected. The user can also switch to another account by logging in again or switching to an anonymous guest option. The user can also sign out which will lead them back to the main menu. The login details will be secure by encryption using a hashing table and dictionary to implement this.
* The account settings can be confirmed using the ‘apply’ button whenever they wish to save something or the ‘login’ button when they have put in their details. The tabs can be closed with the ‘close’ button.

‘New account’ window with ‘apply’ button to confirm the creation of a new account



‘Switch Account’ window with ‘login’ button to access other account

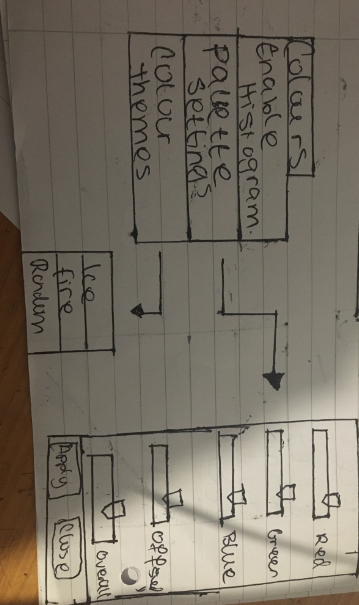
2.10.1.2.1. **Zoom In/Out Function Interface**

Zoomed in image

Zoomed out default image

Undo and redo of zooming in and out of the image. Done through the undo and redo buttons in the option menu with the use of a stack.

Allows the user to zoom into part of the image. The highlighted area Is an example of where the user can draw a box with their mouse arrow to zoom into the area where they wish to. This shall be done with swift movement for the satisfaction of the user (however, the image can take longer or shorter period of time to load depending on the maximum iterations value and the processor strength of their PC). The zooming in of the image will be calculated using the co-ordinates of where the box is being drawn for tracking purposes and for the values to be popped off or pushed onto the stack for redo/undo purposes.

* + - 1. **Colours Tab**

**Colours Tab:**

Sliders

Apply and close buttons

Palette settings window

Themes window

- The ‘colours’ tab has the option of enabling the Histogram colouring method which is done with one click

- It also has the palette settings which are used to manipulate the colours of the palette. (I was going to use an actual palette of many colours however, this can still be done using sliders. If I can, in the time frame I will also include the option of choosing colours by selecting them individually them as well as mixing the RGB values with the sliders as shown in the image above).

- The sliders are used for RGB values and can manipulate the colours of the fractal, so it enables the user to experiment with it accordingly. The slider is used with the mouse and it must be swift and smooth to allow the user to make miniscule adjustments if needed. The offset value is also used for more detailed manipulation. The apply button will apply all the settings needed (if not selected, they will not be implemented into the fractal). The close button will close the window until they need the palette settings again.

- Different colour themes can also be used with the overall fractal. For example, an ‘ice’ theme can be used which will use the attributes of an icy setting with cold, light colours. Or a fire theme can be used to allow a bright red theme. In my real implementation I will allow for a selection of lots more themes. These will be implemented using the PIL library used in python for image manipulation.

* + - 1. **Edit Tab:**

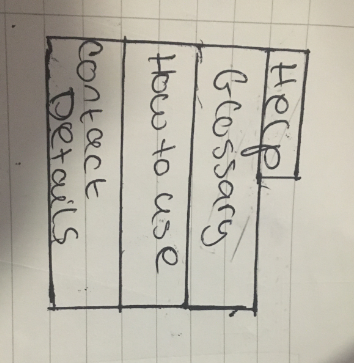


- In the edit tab, the user can manipulate the values of the fractal as they wish to do so.

- There are certain restrictions which are described in the data dictionaries. If the criteria is not met then the program will present an error message in a window and allow the user to proceed by pressing the ‘ok’ button and letting the user try again until correct values are input.

- The input tabs should allow the user to input the values as they are viewing the fractal in order for them to experiment how different values can change the appearance of the fractal.

- The zoom value is optional as the user can zoom into the image with their mouse. Even if they use their mouse, the value of the zoom should be presented in the zoom input box for the user’s interest on the mathematical details of the fractal.

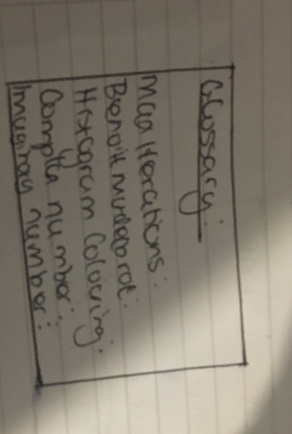
- Once the user presses the apply button, the values will be applied and the fractal will change according to their values (if the values meet the criteria)

* + - 1. **Help Tab:**

**-** The help tab is a very important feature in a program like mine as it is a mathematical tool consisting of the chaos theory with various different mathematical calculations and values needed to be input which a new user may not be familiar with. There are various different fractals and each of them have their own terminology and ways of usage, which is why my program needs a help tab and it is essential.

- When the user clicks on the ‘glossary’ button, a glossary of key terms will appear with clear definitions for the users’ interest. For example, if they wish to input the ‘maximum iterations’ into the system, the user must know what this is to investigate how the fractal is being manipulated. They can look up this term in the glossary as it will be in alphabetical order.

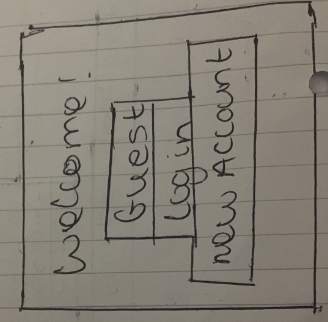
- Also, the ‘how to use’ button is essential to guide users on how the program must be used. For example, there may be a short cut to save the screenshot of the image using ‘CTRL + S’, which is much quicker than accessing the ‘files’ tab. The user must be able to know these shortcuts as well as how to use the basic components of the program such as how to zoom in (by creating a box with your mouse to zoom into the specific area or to use a specific zoom value).

- My personal contact details will also be there if a third party user wishes to access my system or if my end user needs my details if there are any problems in the program which haven’t been addressed. My details can also be used any suggested improvements can be informed to me directly through the stated email in my program.

Example of a few key terms (out of many more that will be shown in my implementation). Details definitions will be used for each term. User can close this window by clicking anywhere on the screen or by clicking the ‘close’ button

* + 1. **Main Menu:**

Welcome window. This is the first window that will appear when the program is opened. This is the start- up menu.



The user can choose to either continue as a guest, login to the system using an existing account (by opening the file system) or by creating a new account where the ‘new account’ window will open

1. **– Implementation**
   1. **Overview**

* Comments/annotation are used to describe all functionalities of the program
* The sections will be split into each separate functionality of the program and separated into the classes and functions used.
* The program starts by opening up ‘loginlogin.py’ which is the registration/login system integrated into the main menu in the same file. If the ‘Help’ button is selected then ‘webbrowser’ is used to load up the Help menu file which is a written ‘notepad’ file to explain all the controls and information about the Mandelbrot set. On the main menu, there is also the RGB slider button for to open up the RGB slider.
* If the user wishes to have their own account, they must click ‘Register’ from the main menu and from there, they click the login button with the corresponding registration details used. Then, the ‘editing\_window.\_updated.py’ is opened to adjust the values of the Mandelbrot set.
* If the user does not wish to log in to the system, they can simply click ‘Guest’ from the main menu and it will directly take them to the editing window. Once the appropriate values have been applied, the Mandelbrot set is launched.
* In addition, my extension objective of creating the Julia set fractal also allows the user to access it from the main menu.
* Overall, the menu components, editing window, RGB slider and the login/registry system are all created using TKinter as the main GUI. The Mandelbrot and Julia set fractals are created using the PyGame as the primary GUI. Finally, the Help menu is created using ‘notepad.’
* Main menu preview: **6.2.47**
  1. **Login/Registry System** 
     1. **Main Menu class (setup)**

from tkinter import \*

import os

import editing\_window\_updated

import time

import RGBSliderNEA

import webbrowser

import string

from random import choice

import JuliaSetNEAExtension

#---------------------------------------------SETUP-------------------------------------------

#'main' is the main parent class for the overall login and registry system

#'object' is used as a standard parameter for the class to show Python that this class is the main object constructor for the child/subclasses (login and register)

#\_\_init\_\_ used to initialise attributes of the class when inherited from subclasses. self.mainScreen is the main root window the main menu

class main(object):

def \_\_init\_\_(self):

self.mainScreen = Tk()

#----------------\_\_pack\_\_---------------------------

#\_\_pack\_\_ is the setting of the essential labels and titles needed within the main menu window (mainScreen)

#'pack()' is used in Tkinter to pack the widgets into rows and columns, .geometry used to fix the window size, .title uses for the main title of the window

def \_\_pack\_\_(self):

self.mainScreen.geometry("600x600")

self.mainScreen.title("Main Menu")

#'Welcome' label at top of the screen. Pack used to place it in adjusted place

#'Click register' label to let user register. 'Login button used to call the subclass login when the button is pressed

#"" label is used to create a slight space between buttons to organise buttons for the user to show them clearly so they do not overlap

#Register button used to call the subclass register when the button is pressed. Guest button is used to call the 'access' function which will call the editing window

#RGB Slider used to call the RGB slider module. Help button used to access a notepad for the help menu

Label(text = "Welcome!", width ="300", height = "2", font = ("Calibri", 12)).pack()

Label(text = "Click register if you do not have an account.", width="300", height="2", font = ("Calibri", 12)).pack()

Button(text="Login", width="30", height ="2", font = ("Calibri", 15), command=login).pack()

Label(text="").pack()

Button(text="Register", width = "30", height = "2", font = ("Calibri", 15), command = register).pack()

Label(text="").pack()

Button(text="Guest", width="30", height="2", font = ("Calibri", 15), command=self.access).pack()

Label(text="").pack()

Button(text="Julia Set Preview", width="30", height="2", font = ("Calibri", 15), command=self.juliaSet).pack()

Label(text="").pack()

Button(text="RGB Slider", width="30", height = "2", font = ("Calibri", 15), command=self.\_\_accessSlider).pack()

Label(text="").pack()

Button(text="Help", width="30", height="2", font = ("Calibri", 15), command=self.\_\_helpMenu).pack()

Label(text="").pack()

#---------------------helpMenu-------------------------------

#webbrowser is used to directly open the notepad text file of the help document for the user's benefit and knowledge behind how the program works

def \_\_helpMenu(self):

webbrowser.open("NEAMandelbrotControls.txt")

#----------------------accessSlider--------------------------

#accessSlider is used to access the RGB Slider which calls the RGBSliderMandelbrot module when the 'RGB Slider' button is clicked

def \_\_accessSlider(self):

RGBSliderNEA.main()

#----------------------juliaSet------------------------------

#juliaSet is used to access the Julia set preview

def juliaSet(self):

JuliaSetNEAExtension.Julia()

#---------------------access----------------------------------

#access is used to access the editing window from the module 'editing\_window\_updated' when clicked on 'Guest' button or when login is successful

#time.sleep(1) used to allow some delay in between editing window showing and when being accessed

def access(self):

time.sleep(0.5)

editing\_window\_updated.call()

#-------------------run---------------------------------------

#run used as the main calling of the tkinter main menu

#.mainloop() allows for Python to run the tkinter event loop for events such as button clicking and any code after this loop will not run until the window is closed

def run(self):

self.mainScreen.mainloop()

#--------------------------------------------DESTROY---------------------------------------

#Used to destroy the widget. Grouped into one class for the ease of access when destroying certain windows/widgets

class destroy(main):

def delete(self):

self.mainScreen.destroy()

def delete1(self):

self.loginScreen.destroy()

def delete2(self):

self.registerScreen.destroy()

* + 1. **Registration**

#--------------------------------------------REGISTER---------------------------------------

#register class is inherited from the parent class of 'main'

#super().\_\_init() is used to give acces to the methods and attributes of the parent class - 'main'. This will return the object representing the parent class

#super will be used to access attributes such as the 'mainScreen' Tk root - inheritance is used here between child class of register to parent class of 'main'

class register(main):

def \_\_init\_\_(self):

super().\_\_init\_\_()

#Toplevel is used when closing the window so all children widgets are destroyed (register window) but the program will not shut down in case the user wants to just

#--> close the register window for the meantime.

self.registerScreen = Toplevel()

self.mainScreen.withdraw()

self.registerScreen.geometry("500x600")

self.registerScreen.title("Register")

#StringVar is used to directly accessing and interpreting the variables when the entry of the user is needed e.g. self.username will be used when helper functions

#--> in the class need to acccess the entry of the username entered by the user

#self.email is for the string variable of the email address, self.password is the password chosen and password2 is the confirmation (second entry) of the password

self.username = StringVar()

self.email = StringVar()

self.password = StringVar()

self.password2 = StringVar()

#Entry is the entry box for each required field for username, password, password2 and email

self.createAccountLabel = Label(self.registerScreen, text = "Create an account", font = ("Calibri", 20)).place(x=50,y=3)

self.detailsTitle = Label(self.registerScreen, text = "Please, fill in the details below in order to register").place(x=90, y=50)

self.requiredTitle = Label(self.registerScreen, text = "\* indicates a required field", fg = "red").place(x=140, y=73)

self.labelUser = Label(self.registerScreen, text ="Username", width = "25").place(x=30, y=130)

self.labelRequiredUser = Label(self.registerScreen, text="\*", fg = "red").place(x=335, y=130)

self.usernameEntry = Entry(self.registerScreen, textvariable = self.username)

self.usernameEntry.place(x=170, y=130)

self.labelEmail = Label(self.registerScreen, text ="Email address", width = "25").place(x=17, y=170)

self.emailEntry = Entry(self.registerScreen, textvariable = self.email)

self.emailEntry.place(x=170, y=170)

self.labelPassword = Label(self.registerScreen, text ="Password", width = "25").place(x=31, y=210)

self.labelRequiredPassword = Label(self.registerScreen, text="\*", fg = "red").place(x=335, y=210)

self.passwordEntry = Entry(self.registerScreen, textvariable = self.password, show = '\*')

self.passwordEntry.place(x=170, y=210)

self.labelConfirm = Label(self.registerScreen, text ="Confirm", width = "25").place(x=35, y=250)

self.labelRequriedConfirm = Label(self.registerScreen, text="\*", fg = "red").place(x=335, y=250)

self.confirmEntry = Entry(self.registerScreen, textvariable = self.password2, show = '\*')

self.confirmEntry.place(x=170, y=250)

self.labelNameReq = Label(self.registerScreen, text="Username is maximum 20 characters").place(x=90, y=290)

self.labelPassReq = Label(self.registerScreen, text="Password must be minimum 8 characters with numerical characters included").place(x=90, y=310)

#Register button is used to call upon the 'registrationComplete' function with .self as it is in the same class

Button(self.registerScreen, text="Register", width=20, bg="brown", fg="white", command = self.\_\_registrationComplete).place(x=160, y=490)

#--------------------registrationComplete------------------------------------

#This function is used to verify the user's details to make sure they match the username, email and password requirements and write to file if criteria met

def \_\_registrationComplete(self):

#userName info is used as a 'getter' function to return the value entered by the user in the entry box which is a string variable (StringVar)

#These are private variables used only within this function

usernameInfo = self.username.get()

passwordInfo = self.password.get()

password2Info = self.password2.get()

emailInfo = self.email.get()

#Used regular expression for precise detail of the criteria being met for the password, username and email (used for revision purpose within Comp Sci)

#The usernameInfo (username entry) must be equal to or below 20 characters and password regex criteria allows for upper and lower case letters with digits 0-9 and with

#--> selected symbols being: @#$%^&+= to maintain simplicity of login system when writing using direct access to write registry details to file

#boolean 'and' used within if statement as all criteria MUST be met before next if statement in nested structure - checks all entry boxes are not empty with != ''

#if statement validates the email entry with upper and lower case letters allowed with no unusual symbols in the emailInfo (StringVar). There must be an @

#--> symbol followed by form of '.' If email regex criteria not met, labelInvalidEmail shown to indicate email is invalid, red font and it is returned

#Next if statememt validates whether or not the password2info (password confirmation) matches the initial password (passwordInfo) using == boolean operator and breaks

while True:

if len(usernameInfo)<=20 and re.match(r'[A-Za-z0-9@#$%^&+=]{8,}', passwordInfo) and usernameInfo != '' and passwordInfo != '' and password2Info != '' and emailInfo != '':

if re.match(r'^[A-Za-z0-9\.\+\_-]+@[A-Za-z0-9\.\_-]+[a-zA-Z]\*$',emailInfo):

if password2Info == passwordInfo:

break

else:

labelNoMatch = Label(self.registerScreen, text=" Passwords don't match ", fg="red",font=("calibri", 11)).place(x=40, y=93)

return

else:

labelInvalidEmail = Label(self.registerScreen, text="Please enter a valid email address", fg="red", font=("calibri", 11)).place(x=70, y=93)

return

else:

labelFillRequired = Label(self.registerScreen, text=" Please fill the required fields", fg="red", font=("calibri", 11)).place(x=70, y=93)

return

#file is used to open the 'usernameInfo.txt' file

#file.write is used with direct access to write to the file the usernameInfo, passwordInfo and emailInfo once the regex criteria has been met

#\n is used to separate the information out for convenience of reading from the file when writing to a new file each time

#file is closed once written correct information

file = open(usernameInfo+".txt", "w")

file.write(usernameInfo+"\n")

file.write(passwordInfo+"\n")

file.write(emailInfo+"\n")

file.close()

#Entry values within the entry boxes are deleted with (0,END) used as index to delete very first value to last to show the user it is now complete and if they wish

#---> to make another account they do not have to delete previous details in the entry boxes themself

self.usernameEntry.delete(0, END)

self.passwordEntry.delete(0, END)

self.emailEntry.delete(0, END)

self.confirmEntry.delete(0, END)

labelSuccess = Label(self.registerScreen, text="Registration success, you can now log in with your user", fg="green", font=("calibri", 11)).place(x=50, y=93)

#if statement via OS to remove previous password file for Vernam cipher (plaintext)

#open file statements used to open the files used for the paramaters of the Vernam encryption

#Vernam cipher function called with necessary files for encryption; plaintext (password), ciphertext, one time pad

if os.path.exists("vernamPass.txt"):

os.remove("vernamPass.txt")

else:

print("File doesn't exist")

mPass = open("vernamPass.txt", "w")

mPass.write(passwordInfo)

mPass.close()

cText = open("ciphertext.txt", "w")

cText.close()

accessOTP = open("oneTimePad.txt", "w")

accessOTP.close()

self.vernamCipher("vernamPass.txt", "ciphertext.txt", "oneTimePad.txt")

* + - 1. **Vernam Cipher**

#---------------------------vernamCipher--------------------------------

#randomised one time pad created using the 'string' library to randomise a character from ASCII table

#The plaintext file is read (password). OTP, plaintext and ciphertext files are opened.

#for loop used to read through all lines of the plaintext file and OTP written in same length of plaintext as one of the Vernam criterias to meet

#Encrypted ciphertext is written using algorithm of using 'chr' to generate the character format and 'ord' is used to generate the integer ASCII format for

#--> decryption of ciphertext

def vernamCipher(self, plaintext, ciphertext, OTP):

randomOTP = (string.ascii\_lowercase + string.ascii\_uppercase

+ string.digits)

accessRegPass = open("vernamPass.txt", "r")

passwordEncrypt = accessRegPass.read()

#test: print("pass: ", passwordEncrypt)

with open(plaintext) as original:

with open(OTP, "w") as oneTimePad:

with open(ciphertext, "w") as encrypted:

for line in original.readlines():

for i in range(len(line)):

calculatedOTP = choice(randomOTP)

oneTimePad.write(calculatedOTP)

encrypted.write(chr(ord(line[i])^ord(calculatedOTP)))

* + 1. **Login**

#-----------------------------------------LOGIN---------------------------------------------

#super().\_\_init\_\_() used to inherit the attributes of the main window from the main parent class

#TopLevel used with self.mainScreen used as a paramater to indicate that this screen should still show when closed to allow the user to still access the main menu

#Standard labels and titles used for similar purposes as the \_\_init\_\_ in register class as a constructor for private attributes

#withdraw to prevent extra widget being opened

class login(main):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.loginScreen = Toplevel(self.mainScreen)

self.loginScreen.title("Login")

self.mainScreen.withdraw()

self.loginScreen.geometry("500x300")

Label(self.loginScreen, text="Please enter details below to login").pack()

Label(self.loginScreen, text="").pack()

self.usernameVerify = StringVar()

self.passwordVerify = StringVar()

Label(self.loginScreen, text="Username \* ").pack()

self.usernameEntry1 = Entry(self.loginScreen, textvariable = self.usernameVerify)

self.usernameEntry1.pack()

Label(self.loginScreen, text="").pack()

Label(self.loginScreen, text="Password \* ").pack()

self.passwordEntry1 = Entry(self.loginScreen, textvariable = self.passwordVerify, show = '\*')

self.passwordEntry1.pack()

Label(self.loginScreen, text="").pack()

#Login button used to call the function of loginVerify to verify the login details entered to check if criteria is matched

Button(self.loginScreen, text = "Login", width = 10, height = 1, command = self.\_\_loginVerify).pack()

#---------------------------------loginVerify-----------------------------------------

def \_\_loginVerify(self):

#username1 is used as a 'getter' function for the username entry in the login window to return the StringVar entry, same purpose for password1

#.delete(0,END) is used to delete the entry details once the user clicks login so if the details are needed to be retyped they do not have to backspace it all or

#--> if they need to make another account it is already deleted from entry box from index 0 to END (last character of entry box)

username1 = self.usernameEntry1.get()

password1 = self.passwordEntry1.get()

self.usernameEntry1.delete(0, END)

self.passwordEntry1.delete(0, END)

#This is an internal private class within the loginVerify function which prints a label showing the username or password are invalid, with red font

#\_\_invalid\_\_prompt is called when the username or password are not recognised. \_empty\_\_prompt is called when the username and/or password entry fields are empty

def \_\_invalid\_prompt():

labelInvalid = Label(self.loginScreen, text="Invalid username or password", fg="red", font=("calibri", 11)).place(x=135, y=19)

def \_\_empty\_prompt():

labelEmpty = Label(self.loginScreen, text="Empty fields --> Enter Details", fg="red", font=("calibri", 11)).place(x=135, y=25)

#if statement for boolean OR to check empty fields of password/username

if (len(self.usernameEntry1.get()) or len(self.passwordEntry1.get())) == 0:

\_\_empty\_prompt()

#try and except statement is used to catch and handle exceptions, in this case, reading the details of 'username1' file

#Opens the file according to the username which is entered, if it is not found in the text format when reading it ("r"), then the except state shows a

#--> FileNotFoundError to indicate the file is not found and therefore the username entered is not registered to the system

#If the except statement is met with invalid input, \_\_invalid\_prompt() is called to show the username or password is invalid and return statement is used

try:

userInfoFile = open(username1+".txt", "r")

except FileNotFoundError:

\_\_invalid\_prompt()

return

#within the file, the userData is read line by line as it is split using \n. print statement used for testing purpose with tuple

userData = userInfoFile.read().split("\n")

#Standard 'if' statement used with boolean 'and' to check whether the password entry matches the password in the file if username complementing the password is also

#--> matched to the username entered

#'else' used to call \_\_invalid\_prompt() to show username and password are invalid, return statement. Direct OS access used to remove previous decrypted login

#Decryption file is created using open stateent and decrypter is called with parameters of appropriate files

if password1 == password and username1 == username:

if os.path.exists("decryptionRev.txt"):

os.remove("decryptionRev.txt")

else:

print("File doesn't exist!")

decryptPass = open("decryptionRev.txt", "w")

decryptPass.close()

self.vernamDecrypter("ciphertext.txt", "decryptionRev.txt", "oneTimePad.txt")

self.access()

return

else:

\_\_invalid\_prompt()

return

* + - 1. **Vernam Decrypter**

#----------------------vernamDecrypter----------------------

#All necessary files are opened and try, except statement used to decrypt if the file exsits using the decryption algorithm designed

def vernamDecrypter(self, ciphertext, reverse, OTP):

with open(ciphertext) as encrypted:

with open(OTP) as accessOTP:

with open(reverse, "w") as decryption:

try:

while True:

decryption.write(chr(ord(encrypted.read(1))^ord(accessOTP.read(1))))

except:

pass

* + 1. **Login/Registry Call**

#-------------------------------------------------MAIN---------------------------------------

#mainRunning function used to call the parent class main() which therefore is called upon by register and login sub/child classes with inheritance

#\_\_pack\_\_() is called for the private attributes in the main widget

#.run() used for mainloop of main widget

def mainRunning():

mainCall = main()

mainCall.\_\_pack\_\_()

mainCall.run()

#if statement used in case another module is used to access loginlogin.py which won't be run as soon as it is imported

#mainRunning() subroutine is called

if \_\_name\_\_ == "\_\_main\_\_":

mainRunning()

* 1. **Editing Window**

from tkinter import \*

from tkinter import ttk

import os

import mandelbrotNEAOOP

import re

#editing class is initialised as an object as its the main constructor of variables, buttons etc. when being called upon by the loginlogin module

#editWindow is the main Tk() root widget with dimensions of 700x500 constructued by .geometry and has title of Fractal Editing Window

#reg is for the entry input validation to ensure only floating point numbers or decimals

class editing(object):

def \_\_init\_\_(self):

self.editWindow = Tk()

self.editWindow.geometry("700x500")

self.editWindow.title("Fractal Editing Window")

reg = self.editWindow.register(self.inputValidation)

self.editLabelTitle = ttk.Label(self.editWindow, text="Input the values required: ", font = ("Calibri", 20)).place(x=125, y = 10)

#value = 50 is the default value input - recommended maximum iterations value

#validate="key" and registration used for exception handling of entry box

self.maximumIterationsInput = IntVar(self.editWindow, value=50)

self.labelMaxIt = Label(self.editWindow, text="Maximum Iterations: ", width="25").place(x=27, y=150)

self.maximumItEntry = Entry(self.editWindow, textvariable = self.maximumIterationsInput)

self.maximumItEntry.place(x=170, y=150)

self.maximumItEntry.config(validate="key", validatecommand=(reg, '%P'))

#self.sizeSelect is to monitor the string variable selected (the element selected from imageSizes list)

#self.sizeSelect.set uses and index for the second element in the imageSizes list to show as the default window size as its the smallest and allows quickest processing

#self.imageSizesDrop is the dropbox used for the image dimensions used. Uses the list of self.imageSizes as the options from the dropbox

self.labelDimensions = Label(self.editWindow, text="Image Dimensions: ", width="25").place(x=27,y=200)

self.imageSizes = ["320x200", "512x512", "640x480", "800x600", "1280x1024"]

self.sizeSelect = StringVar(self.editWindow)

self.sizeSelect.set(self.imageSizes[1])

self.imageSizeDrop = OptionMenu(self.editWindow, self.sizeSelect, \*self.imageSizes).place(x=170, y=200)

#Default is the standard RGB values selected by me, fire is a red theme, emerald is a light blue/green theme, ocean is a blue theme and sunset is a yellow/orange theme

#self.themesSelect is to monitor the string variable selected from the self.themes list in the dropbox

#self.themesSelect.set is used to set 'Default' as the default theme selected, also known as the first element in the list

self.labelColourThemes = Label(self.editWindow, text="Colour Themes: ", width="25").place(x=27, y=250)

self.themes = ["Default", "Fire", "Sunset", "Ocean", "Emerald"]

self.themesSelect = StringVar(self.editWindow)

self.themesSelect.set(self.themes[0])

self.themesDrop = OptionMenu(self.editWindow, self.themesSelect, \*self.themes).place(x=170, y=250)

#self.ssInput is to monitor the decimal value of the value entered in the entry box also known as being a 'Double' rather than integer as scaling speed value is small

#value = 0.01 shows the default value of scaling speed. self.ssEntry is the entry box for the ssInput DoubleVar (scaling speed)

#validate used for exception handling for floating point numbers

self.ssLabel = Label(self.editWindow, text="Scaling Speed: ", width="25").place(x=27, y=300)

self.ssInput = DoubleVar(self.editWindow, value=0.001)

self.ssEntry = Entry(self.editWindow, textvariable = self.ssInput)

self.ssEntry.place(x=170, y=300)

self.ssEntry.config(validate="key", validatecommand=(reg, '%P'))

self.applyButton = Button(self.editWindow, text="Apply Values",

command = self.\_\_apply)

self.applyButton.place(x=100, y=350)

self.warningIterations= Label(self.editWindow, text="WARNING: Maximum Iteration values over 500 may take more processing time: O(2^n)...",

fg="red",font=("calibri", 11)).place(x=10, y=70)

self.warningDimensions = Label(self.editWindow, text=" LARGER THE WINDOW SIZE = LARGER THE PROCESSING TIME!",

fg="red", font=("calibri", 11)).place(x=10, y=90)

#-------------------inputValidation-------------------------------

#Regular expression and isdigit used with if statements for exception handling of integers and floating point numbers for SS and maximum iteration entries

#Blank space also returns True ("")

def inputValidation(self, inp):

p = re.compile('^(\d+)?([.]?\d{0,10})?$')

if inp.isdigit() or p.match(inp):

return True

elif inp == "":

return True

else:

return False

#----------------------setImageDimensions--------------------------

#imageSize is a 'getter' function to return the value of the image dimension selected for the fractal size in the window

def \_\_setImageDimensions(self):

imageSize = self.sizeSelect.get()

#'if' statement is used to check is 'imageSize.txt' exists , if it does, then 'os' will remove it from directory

#It is removed so that whenever a new value is input then the file will be remade and the new value will be written rather than a file with lots of values

#'else' will print the file doesnt exist if the file does not exist in os

if os.path.exists("imageSize.txt"):

os.remove("imageSize.txt")

else:

print("File doesn't exist!")

#imageDimensionsFile is to open the 'imageSizes.txt' in an append format to append the height and width of the dimensions needed

#if the imageSize is equal to (equal boolean operator) to the first element of the imageSizes list returned being 300x200.

imageDimensionsFile = open("imageSize.txt", "a")

if imageSize == self.imageSizes[0]:

imageDimensionsFile.write(f"\n{320}")

imageDimensionsFile.write(f"\n{200}")

imageDimensionsFile.close()

elif imageSize == self.imageSizes[1]:

imageDimensionsFile.write(f"\n{512}")

imageDimensionsFile.write(f"\n{512}")

imageDimensionsFile.close()

elif imageSize == self.imageSizes[2]:

imageDimensionsFile.write(f"\n{640}")

imageDimensionsFile.write(f"\n{480}")

imageDimensionsFile.close()

elif imageSize == self.imageSizes[3]:

imageDimensionsFile.write(f"\n{800}")

imageDimensionsFile.write(f"\n{600}")

imageDimensionsFile.close()

else:

imageDimensionsFile.write(f"\n{1280}")

imageDimensionsFile.write(f"\n{1024}")

imageDimensionsFile.close()

#-------------------------setIterations----------------------------

#This function is used to set the maximum iterations value as entered by the user

#maximumIterations will return the maximum iterations value inputted in the main widget

def \_\_setIterations(self):

maximumIterations = self.maximumIterationsInput.get()

#'if' statement used to verify the maximum iterations value is only going to be processed and written to the file if the value is above 0

#The maximumIterations value returned from the entry box is written to the file using fractalFile.write

while True:

if maximumIterations > 0:

if os.path.exists("maximumIterations.txt"):

os.remove("maximumIterations.txt")

else:

print("File doesn't exist!")

fractalFile = open("maximumIterations.txt", "a")

fractalFile.write(f"\n{maximumIterations}")

fractalFile.close()

return

self.maximumItEntry.delete(0, END)

#'else' used to show user that if the maximum iterations value is below 0 then it can not be processed, red font.

#return statement to show that while loop is now ended

else:

labelTitle = Label(self.editWindow, text="Iterations can't 0 or below", fg="red",font=("calibri", 11))

labelTitle.place(x=40, y=93)

return

#--------------------------setThemes-----------------------------

#A 2D array is used for each set of RGB values for each theme

#2D array used as all sets of RGB values need to be stored in an array for access through file individually for colouring algorithm to loop through the rows

#Default = Default theme chosen by me, Fire = Red theme, Ocean = Blue Theme, Emerald = Green/Turqoise theme, Sunset = yellow/orange theme

#RGB values decided upon by the use of my RGB slider tool

def \_\_setThemes(self):

default = [[10, 5, 20],

[10, 10, 35],

[11, 20, 20],

[10, 7, 26],

[9, 1, 47],

[4, 4, 73],

[0, 7, 100],

[12, 44, 138],

[24, 82, 177],

[57, 125, 209],

[134, 181, 229],

[211, 236, 248],

[241, 233, 191],

[248, 201, 95],

[255, 170, 0],

[204, 128, 0],

[10, 5, 20]]

fire = [[230, 10, 20],

[250, 5, 5],

[240, 10, 10],

[235, 12, 12],

[220, 15, 15],

[216, 30, 40],

[200, 50, 90],

[195, 50, 90],

[200, 30, 20],

[57, 125, 209],

[134, 181, 229],

[211, 236, 248],

[241, 233, 191],

[248, 201, 95],

[255, 170, 0],

[204, 128, 0],

[230, 10, 20]]

sunset = [[245, 117, 10],

[230, 110, 9],

[235, 104, 13],

[220, 100, 26],

[210, 90, 30],

[215, 95, 25],

[210, 85, 20],

[210, 44, 138],

[24, 82, 177],

[57, 125, 209],

[134, 181, 229],

[211, 236, 248],

[241, 233, 191],

[248, 201, 95],

[255, 170, 0],

[204, 128, 0],

[245, 117, 10]]

ocean = [[17, 84, 174],

[10, 80, 170],

[15, 82, 175],

[20, 83, 162],

[27, 79, 172],

[30, 84, 163],

[57, 71, 167],

[80, 44, 138],

[70, 82, 177],

[57, 125, 209],

[134, 181, 229],

[211, 236, 248],

[241, 233, 191],

[248, 201, 95],

[255, 170, 0],

[204, 128, 0],

[20, 80, 190]]

emerald = [[5, 150, 81],

[10, 160, 90],

[5, 170, 85],

[10, 160, 90],

[20, 175, 87],

[25, 190, 80],

[2, 160, 81],

[12, 44, 138],

[24, 82, 177],

[57, 125, 209],

[134, 181, 229],

[211, 236, 248],

[241, 233, 191],

[248, 201, 95],

[255, 170, 0],

[204, 128, 0],

[5, 150, 81]]

#'while True' loop used to check if 'mandelbrotThemes.'txt' exists in director. If it does exist in directory, it is in order to write the new RGB values from the

#--> 2D array according to what the user has chosen from drop down menu

#themesFile is the assignment for opening the 'mandelbrotThemes' file in append mode to append the RGB values into the text file

colourTheme = self.themesSelect.get()

while True:

if os.path.exists("mandelbrotThemes.txt"):

os.remove("mandelbrotThemes.txt")

else:

print("File doesn't exist!")

themesFile = open("mandelbrotThemes.txt", "a")

#The 'for row' loop is used to loop through every set of RGB values and the 'for item' loop is used to loop through every integer value within the RGB sets within

#--> the 2D array. 2 For loops used to loop through each set and then each value within the sets as its a 2D array

#For every item in the rows the value is written in string format with a space between them. For every row in text file it is written on a new file using "\n".

if colourTheme == self.themes[0]:

with open("mandelbrotThemes.txt", "wt"):

for row in default:

for item in row:

themesFile.write(str(item) + " ")

themesFile.write("\n")

return

elif colourTheme == self.themes[1]:

with open("mandelbrotThemes.txt", "wt"):

for row in fire:

for item in row:

themesFile.write(str(item) + " ")

themesFile.write("\n")

return

elif colourTheme == self.themes[2]:

with open("mandelbrotThemes.txt", "wt"):

for row in sunset:

for item in row:

themesFile.write(str(item) + " ")

themesFile.write("\n")

return

elif colourTheme == self.themes[3]:

with open("mandelbrotThemes.txt", "wt"):

for row in ocean:

for item in row:

themesFile.write(str(item) + " ")

themesFile.write("\n")

return

else:

with open("mandelbrotThemes.txt", "wt"):

for row in emerald:

for item in row:

themesFile.write(str(item) + " ")

themesFile.write("\n")

return

#------------------------------setSS---------------------------------------

#setSS is a function used to retrieve, return and write the scaling speed value. scalingSpeed returns the ssInput which is input into the entry box

def \_\_setSS(self):

scalingSpeed = self.ssInput.get()

#'if' statements used to only process if the scaling speed is above float value of 0.0. ssFile is used open the 'scalingSpeed.txt' text file in notepad

#ssFile.write is used to write the scalingSpeed value on a new line using \n for direct access and then file is closed

#Once all the values have been input, mandelbrotNEAOOP.run() is used to call the main run() subroutine in the mandelbrotNEAOOP module

while True:

if scalingSpeed > 0.0:

if os.path.exists("scalingSpeed.txt"):

os.remove("scalingSpeed.txt")

else:

print("File doesn't exist!")

ssFile = open("scalingSpeed.txt", "a")

ssFile.write(f"\n{scalingSpeed}")

ssFile.close()

mandelbrotNEAOOP.run()

return

else:

labelTitle = Label(self.editWindow, text="Zooming/Scaling speed can not be 0 or below. Value above 1 is not not useful. Recommended: 0.1",

fg="red",font=("calibri", 11))

labelTitle.place(x=40, y=113)

return

#-----------------------apply----------------------

#This subroutine calls all the 'setter' functions used to write the values for mandelbrotNEAOOP module to read and use in calculations

def \_\_apply(self):

self.\_\_setImageDimensions()

self.\_\_setIterations()

self.\_\_setThemes()

self.\_\_setSS()

#--------------------editRun------------------------

#editRun() is used as a mainloop for the standard editWindow to enable all the buttons and labels to be called and run

def editRun(self):

self.editWindow.mainloop()

#-------------------call----------------------------

#call function is outside the class to call class as an object of editCall with .editRun to call overall main editing window widget

def call():

editCall = editing()

editCall.editRun()

#'if' statement is used to call this module and to not run straight away when it is called through loginlogin.py module

if \_\_name\_\_ == "\_\_main\_\_":

call()

* 1. **Mandelbrot Fractal** 
     1. **Stack Class**

from math import log, log2

import pygame

import csv

import sys

#----------------------------------STACK-(FILO) -------------------------------------

#Use of stack: To push on the fractal values when zooming in. Use stack to pop off the values in order to zoom out consectutively

#-->isEmpty:

# Checks to see if the stack is empty to prevent underflow errors when zooming out of image

#atleast one set of 'current boundaries' to be pushed onto the stack initially to load the fractal so list is always containing minimum 1 element.

#-->push:

#Pushes the current fractal zoom boundaries (values used to generate the fractal image) onto the stack. Standard '.append' list to add item to front of list

#-->pop:

#The pop function is used to return the last item pushed on the stack, in this case, the last image that was zoomed in and pushed onto the stack

#-->peek:

#To check the length of the list, it is printed and the top item from the list is returned, but not removed as it is just 'peeking'

#-->size:

#Returns the length of the stack (list) in the current point of runtime

class Stack:

def \_\_init\_\_(self):

self.items = []

def isEmpty(self):

if len(self.items) == 1:

return True

else:

return False

def push(self, item):

self.items.append(item)

def pop(self):

retVar = self.items[len(self.items) - 1]

self.items.pop()

return retVar

def peek(self):

print(len(self.items))

return self.items[-1]

def size(self):

return len(self.items)

* + 1. **Getter Functions for Fractal Class**

#-------------------------------------FRACTAL-----------------------------

#-------------------------------------\_\_init\_\_----------------------------

#SCALE is a constant to show the scale to which the image is being zoomed in. bounds is a variable used to as a list for the minimum and maximum values of x and y

#The bounds are used as each pixel has to represent a complex number (within the complex plane - cartesian coordinate system). The pixels are coloured with RGB values

#----> according to whether or not they belong to the Mandelbrot fractal set

class fractal():

def \_\_init\_\_(self):

self.zoomInOut = Stack()

self.SCALE = 0.1

self.bounds = (-2.5, -1.5, 1, 2.5)

#--------------------------Getter-Procedures------------

#These functions in OOP are defined to protect the data (variables/constants) when creating classes, hence, making them private variables which is standard OOP rule

#They all return the value

#--------------------------getScale---------------------

#Returns the scale constant (0.1)

def getScale(self):

return self.SCALE

#--------------------------getBounds-------------------

#Returns the boundary values

def getBounds(self):

return self.bounds

#------------------------getMaximumIt------------------

#Used to access the maximum iterations value according to what the user has set it as in the editing window. Opens the file and reads

#Context: This value is used to determine the maximum iterations of the fractal, higher the number the larger the processing time and more detail+quality of fractal

def getMaximumIt(self):

accessIterationsFile = open("maximumIterations.txt", "r")

maximumItValue = accessIterationsFile.read()

self.maximumItEntry = int(maximumItValue)

return self.maximumItEntry

#-----------------------getScalingSpeed----------------

#Value is converted to a float for the format of calculations. The scaling speed is changes the rate at which the red, rectangular zooming box changes in size

def getScalingSpeed(self):

accessSSFile = open("scalingSpeed.txt", "r")

ssValue = accessSSFile.read()

self.scalingSpeed = float(ssValue)

return self.scalingSpeed

#----------------------getImageDimensions--------------

#Used to retrieve the image dimension values according to what the user has set the values as in the editing window

#The file is organised for direct access (not sequential) as the values are written on separate lines for the convenience of access to height and width image values

#The values are retrieved from the option of the editing window as being standardised industry image dimension values. For PyGame window

def getImageDimensions(self):

accessImageSizeFile = open("imageSize.txt", "r")

lines = accessImageSizeFile.readlines()

HEIGHT = lines[1]

WIDTH = lines[2]

self.HEIGHT = int(HEIGHT)

self.WIDTH = int(WIDTH)

return self.HEIGHT, self.WIDTH

#--------------------getZoomSize-------------------------------

#Used to return the value of 'zoomSize' which determines the size of rectangular zooming box with the use of the scale and width of the image

#The width is multiplied by the scale as the scaling of the box is determined by how it stretches in the y direction with the aid of the rate of scale (zoom)

def getZoomSize(self):

self.zoomSize = self.WIDTH \* self.SCALE

return self.zoomSize

#-------------------getAspectRatio-----------------------

#The aspect ratio formula used below is the standardised formula for determining the relationship between the width and height of the window (image dimensions)

#This is used to maintain the correct ratio to ensure the overall resolution and proportionality of pixels is maintained. The aspect ratio value is returned

def getAspectRatio(self):

self.aspectRatio = self.WIDTH / self.HEIGHT

return self.aspectRatio

#------------------getThemes------------------------------

#The 'for row' will read through every row in the text file and the 'strip' method is used to return a copy of the string by removing any leading and trailing

#--> characters. If there are no spaces, it will continue on to read through the file until there is a space (where the 2D array RGB values ends)

#The 'for RGB' loop will loop for every string in the file to split the strings in every row into a list for later use in the program when using the RGB values

#The self.colourBoundaries list will append the values from the colourRows list and the 2D array is returned

def getThemes(self):

self.colourBoundaries = []

with open("mandelbrotThemes.txt") as themesFile:

for row in themesFile:

if not row.strip():

continue

colourRows = []

for RGB in row.split():

colourRows.append(int(RGB))

self.colourBoundaries.append(colourRows)

return self.colourBoundaries

* + 1. **Mandelbrot Formula**

#---------------mandelbrotCalculation------------

#Standard mathematical algorithm for Mandelbrot set . The starting location of the set is determines by the constant C. Starts with z = 0

#The resulting value is put into z and the original location is determined by C for the purpose of the iteration and the value of z is a mathematical reccurring loop

#Complex numbers are used a real and an imaginary part of it on the standard cartesian coordinate system of where the pixels are plotted upon

def \_\_mandelbrotCalculation(self, z, c):

z = z \* z + c

return z

* + 1. **Recursive Mandelbrot Calculation**

#-------------mandelbrotRecursion-----------------

#The fractal is limited to the value of 1 to determine which values calculated are part of the Mandelbrot set and which values are not. Coloured black if not in set.

#Then, the overall function is implemented through recursion, until the base case is met and result of recursion returned.

def \_\_mandelbrotRecursion(self, z, c, iterations):

if iterations <= 1 :

return self.\_\_mandelbrotCalculation(z, c)

return self.\_\_mandelbrotRecursion(self.\_\_mandelbrotCalculation(z, c), c,

iterations - 1)

* + 1. **Radius Limit Check**

#-----------checkRadiusLimit-----------------------

#It checks if Z went further than as a radius from the origin which therefore indicates which values are within the Mandelbrot set (radius of 2, diameter=4)

#This is the overall determinant for how many iterations are completed in each zoom before extending past radius of 2 where it is no longer part of the Mandelbrot set

#cRealSquared squares the constant c. Real and Imag(inary) are part of the complex **numbers.**

def \_\_checkRadiusLimit(self, c):

cRealSquared = c.real \* c.real

cImagSquared = c.imag \* c.imag

return cRealSquared + cImagSquared > 4

* + 1. **Linear Interpolation and Histogram Colouring/Assigning Colours**

#---------linearInterpolation-----------------------

#This is the standard linear interpolation calculation in the stasticial part of Mathematics for the colouring algorithm of the fractal

#The values are plugged into the equation whenever it is called upon by other functions in the class

def \_\_linearInterpolation(self, v0, v1, t):

linearPEquation = (1 - t) \* v0 + t \* v1

return linearPEquation

#---------colouring--------------------------------

#I used linear interpolation as part of my extension objectives for smooth colouring to render the normalised iteration count with bands of colours being replaced with

#--> a smoother colour gradient between each colourised pixel. #LinearP is carried out on the RGB values to determine the colour of pixels.

#Number of colours determines the detail of imagery of the colour. Higher the value, higher the processing time and colours become more enhanced.

#A nested for loop is used for the scaling and colouring of each and every x and y coordinate across the fractal according to the iterative values previously solved

#Looped through as a list where [i], [i+1]... to loop through all the overall sets of RGB values within the 2D array

#The values following such as [0], [1]... are used to plug in each RGB value within the overall sets in the 2D array into the linear interpolation calculation

def \_\_colouring(self):

numberOfColours = 17

colours = [(self.\_\_linearInterpolation(self.colourBoundaries [i][0], self.colourBoundaries [i + 1][0],

t / (numberOfColours / len(self.colourBoundaries ))),

self.\_\_linearInterpolation(self.colourBoundaries [i][1], self.colourBoundaries [i + 1][1],

t / (numberOfColours / len(self.colourBoundaries ))),

self.\_\_linearInterpolation(self.colourBoundaries [i][2], self.colourBoundaries [i + 1][2],

t / (numberOfColours / len(self.colourBoundaries ))))

for i in range(-1, len(self.colourBoundaries ) - 1) for t in range(numberOfColours // len(self.colourBoundaries ))]

return colours

#------assignColour--------------------------------

#The recursive algorithm now carried out for the colouring of the pixels to make sure the iterative values are under the maximum iterations value

#The initial recursiveZ variable is used to assign 0 to it for the start of the while loop of the 'mandelbrotCalculation' where the value of 0 and constant c are

#--> plugged in. recursiveZ used for recursion purpose.

#The RGB value of (0,0,0) is returned for when the value is exceeding the radius limit to colour the fractal black, indicating values not within the Mandelbrot

def \_\_assignColour(self, c, maximumItEntry, colours):

i = 0

recursiveZ = complex(0)

while i < self.maximumItEntry:

recursiveZ = self.\_\_mandelbrotCalculation(recursiveZ, c)

if self.\_\_checkRadiusLimit(recursiveZ):

return colours[i % len(colours)]

i += 1

return (0, 0, 0)

* + 1. **Smooth Colouring (Extension Objective Met)**

#---------smoothColouring-------------------------

#EXTENSION OBJECTIVE MET. #If the maximum iterations value is met, the overall function will halt and the value is retuned

#Once the while loop is finished, the standard log base 10 and log base 2 functions are applied to the recursive z value. Equation = 'potential function'

def \_\_smoothColouring(self, c, maximumItEntry):

z = 0

n = 0

while abs(z) <=2 and n < self.maximumItEntry:

z = z \* z + c

n += 1

if n == self.maximumItEntry:

return self.maximumItEntry

return n + 1 - log(log2(abs(z)))

* + 1. **Coordinate/Pixel Complex Number Conversions**

#-------convertToComplex-------------------------

#This is the standard mathmatical function used to convert the standard x and y coordinates to complex numbers e.g. (0 + 5j)

#The real part of the complex number is using linear interpolation with the x values and the width dimension of the image as the width corresponds to the x axis

#The equation assigned to the real variable calculates a resulting value for the displacement of the real part of the complex number

#The same equation is applied to the imaginary part of complex number for the displacement along the y axis

#Pixel coordinates are converted to complex number as part of the cartesian coordinate system

def \_\_convertToComplex(self, x, y, bounds=(-2.5, -1.5, 1, 1.5)):

minimumX = bounds[0]

maximumX = bounds[2]

minimumY = bounds[1]

maximumY = bounds[3]

real = self.\_\_linearInterpolation(minimumX, maximumX, x / self.WIDTH)

imaginary = self.\_\_linearInterpolation(minimumY, maximumY, y / self.HEIGHT)

return complex(real, imaginary)

#------convertFromComplex------------------------

#Used to convert the x and y coordinate values plotted sequentially across the coordinate plan back to the complex number

#Standardised mathematical algorithm of the reverse of 'convertToComplex' function

#The complex numbers represent the distance/displacement between each pixel and the minimum/maximum values represent the corners of the coordinate cartesian plane

#The minumum, real part of the complex number is the left border of the image and the maximum, real part is the right border of the image

#The coordinates are calculated manually with the algorithm to disallow the image from stretching no matter what the dimensions are - could distort real part

#The subtraction of height and width is used to calculate the corner pixels of the fractal

def \_\_convertFromComplex(self, real, imaginary, bounds=(-2.5, -1.5, 1, 1.5)):

minimumX = bounds[0]

maximumX = bounds[2]

minimumY = bpunds[1]

maximumY = bounds[3]

x = (real \* self.WIDTH - self.WIDTH \* minimumX) / (-minimumX + maximumX)

y = (imaginary \* self.HEIGHT - self.HEIGHT \* minimumY) / (-minimumY + maximumY)

return (x, y)

* + 1. **Main Mandelbrot Fractal Plot**

#------main--------------------------------------

#All the functions are called and pixels are plotted and colourised, also enabling the user to zoom in and out of the image

def main(self):

#----Initialising-variables---------------------

#mousePosition is the variable to retrieve the current state of the mouse device

#screen is the variable to create an instance of the pygame window.

#imageRep is the image representation used in pygame surface function. Surface has a fixed resolution and pixel format. In an 8-bit pixel format to map 24-bit colours

#currentBounds is the current boundaries of each zoom within the zoom box

mousePosition = pygame.mouse.get\_pos()

screen = pygame.display.set\_mode((self.WIDTH, self.HEIGHT))

imageRep = pygame.Surface((self.WIDTH, self.HEIGHT))

currentBounds = (-2.5, -1.5, 1, 1.5)

#The pygame image is drawn on using the current x and y coordinates. The assignColour function is called on the current x, y, and currentBounds. In addition, the

#--> the pixels are iterating and plotting onto the image according to the maximumItEntry. The colouring function is also called to colour the pixels

#zoomNumber is initialised to allow the user to zoom into the image

#The stack defined as the object 'zoomInOut' and the currentBounds are pushed on as the tracking of the bounds being used for the pixels and to call them once the user

#--> requests to zoom out.

for y in range(self.HEIGHT):

for x in range(self.WIDTH):

imageRep.set\_at((x, y), self.\_\_assignColour(self.\_\_convertToComplex(x, y, currentBounds), self.maximumItEntry,

self.\_\_colouring()))

self.\_\_smoothColouring(self.\_\_convertToComplex(x, y, currentBounds), self.maximumItEntry)

zoomNumber = 1

self.zoomInOut.push(currentBounds)

#sleepOver is defined as False to pause the program when needed (set as True) to minimise lag in case the user spams mouse clicks to frequently

#done is defined as False to indicate the interaction with the GUI from the user is not done yet (button clicking, zooming etc...)

sleepOver = False

done = False

#The functions to allow the user to interact with the pygame GUI is within the while loop until the user is done

#The mousePosition is once again initialised to track the movement of the current state of the mouse

while not done:

mousePosition = pygame.mouse.get\_pos()

#pygame.event.get() for loop is used to register all the events of the interaction of the user into an abstract data type of a queue (FIFO) where actions are

#-->internally queued on internally in the pygame module to keep track of the users' actions

#Each event type is enqueued to the queue. buttons variable uses the internal pygame function, this returns a list of all the states of the keys being pressed

#Internal list contains 0 for all the keys not pressed and 1 for all keys which are pressed

for event in pygame.event.get():

buttons = pygame.key.get\_pressed()

#EXIT program button

if event.type == pygame.QUIT:

done = True

sys.exit()

#Calls the event type which will call the mouseClick function when the mouse button is clicked

if event.type == pygame.MOUSEBUTTONDOWN:

self.mouseClick(mousePosition, currentBounds, zoomNumber, imageRep, False)

#'if' BACKSPACE is clicked the fractal will zoom out by popping off last boundaries and checking if stack is empty

#The given parameters are assigned to 'params' to track the values and return the last used values on the stack to peek the top of the stack

#The mouseClick function is called for the fractal boundaries to be pushed on

#'if' statement used to check if the stack is empty preventing'out of range' error or 'underflow' error occurring on stack when attempting to zoom out with empty stack

if buttons[pygame.K\_BACKSPACE]:

if self.zoomInOut.isEmpty():

pygame.display.set\_caption("Can Not Zoom Out...")

continue

else:

pygame.display.set\_caption("Zooming out...")

self.zoomInOut.pop()

params = self.zoomInOut.peek()

self.mouseClick(mousePosition, params, zoomNumber, imageRep, True, currentBounds)

#'if' the UP arrow key is pressed, this will increase the overall size of the zoom rectangle box of where the user wishes to zoom

#'zoomSize' is used to maintain the size of the rectangle as the overall size is strethched out in the x direction

if buttons[pygame.K\_UP]:

self.SCALE = min(1, self.SCALE + self.scalingSpeed)

self.zoomSize = self.WIDTH \* self.SCALE

#'if' the DOWN arrow key is pressed, the function is the similar concept to the UP arrow key being pressed but the reverse happens and the rectangle decreases the size

if buttons[pygame.K\_DOWN]:

self.SCALE = max(0.01, self.SCALE - self.scalingSpeed)

self.zoomSize = self.WIDTH \* self.SCALE

#'if' the SPACE key (SPACEBAR) is pressed, the conversion to complex is called to print current position of mouse in a complex format for the fractal plane.

if buttons[pygame.K\_SPACE]:

print(self.\_\_convertToComplex(\*mousePosition, currentBounds))

#'if' the 's' button is pressed (s for 'save') the current image of the fractal is saved to the user's default file directory of where pygame is installed

if buttons[pygame.K\_s]:

pygame.image.save(screen, "MandelbrotFractal.jpeg")

#.blit is used to draw one image onto another which will draw the next fractal image when the image is zoomed in on

#imageRep represents the image being drawn and sets 0 as the initial values needed for the 'blit'

#pygame.draw.rect will draw the rectangle used to draw the red rectangular zoom box

#(255, 0, 0)=red RGB for the clear indication of zoom box. The caption is set to show retrieve the current mouse position of the user and print on top of the window.

#pygame.display.update() will update the display of the pygame window in runtime

screen.blit(imageRep, (0, 0))

pygame.draw.rect(screen, (255, 0, 0),

(mousePosition[0] - self.zoomSize / 2, mousePosition[1] - (self.zoomSize / self.aspectRatio) / 2,

self.zoomSize, self.zoomSize / self.aspectRatio), 1)

pygame.display.set\_caption(str(self.\_\_convertToComplex(\*pygame.mouse.get\_pos(), currentBounds)))

pygame.display.update()

* + 1. **Mouse Click Zoom Stack Implementation**

#-----------mouseClick----------------------

#This function is the zooming into the fractal when the mouse button is clicked

#positionVar and 2 are the new boundaries of pixels being converted to complex numbers when zooming into the fractal as the boundaries and coordinates change

#Passes the calculation for obtaining the current mouse position and the size of the rectangular zooming box and dividing it by 2 as a parameter for the

#--> complex conversion. Current boundaries set to the last boundaries on stack using peek function.

def mouseClick(self, mousePosition, currentBoundsP, zoomNumber, imageRep, useParams=False, currentBounds=False):

currentBounds = self.zoomInOut.peek()

pygame.display.set\_caption("Processing...")

positionVar1 = self.\_\_convertToComplex(mousePosition[0] - self.zoomSize / 2, mousePosition[1] - (self.zoomSize / self.aspectRatio) / 2,

currentBounds)

positionVar2 = self.\_\_convertToComplex(mousePosition[0] + self.zoomSize / 2, mousePosition[1] + (self.zoomSize / self.aspectRatio) / 2,

currentBounds)

#useParams is set to the boolean value of False to indicate the parameters have not been used

#if useParams used to track progress of whether parameters have been used, if so, no zoom is made. Else, new boundaries calculated for zoom to track it with

#This is also evident of polymorphism as 'zoomInOut' is being used for different purposes such as zooming in and zooming out using push and pop in different cases

#The zoom number is then incremented and divided by the scale to finalise the zoom of the fractal and to readjust the zoom value of the image. Boundaries are pushed

#--> onto stack for later usage of zooming in/out

if useParams:

currentBounds = currentBoundsP

else:

currentBounds = (positionVar1.real, positionVar1.imag,

positionVar2.real, positionVar2.imag)

self.zoomInOut.push(currentBounds)

zoomNumber += 1 / self.SCALE

#For loop used as the height and width are finite which is complemented by the finite number of pixles within the fractal window for coordinated plotting on the plane

#The image is now set to the zoomed in fractal using the correct parameters and then it is saved to a list; '', for tracking purposes of values of parameters

#--> being used. pygame.event.clear used to clear the events of the user clicking the mouse button within the internal pygame module queue.

#This is used so that if the user double clicks, the program will not respond to the image being zooming in twice and therefore resulting in lag

for y in range(self.HEIGHT):

for x in range(self.WIDTH):

imageRep.set\_at((x, y), self.\_\_assignColour(self.\_\_convertToComplex(x, y, currentBounds),

self.maximumItEntry + 4 \* zoomNumber, self.\_\_colouring()))

imageSave = [(x, y), self.\_\_assignColour(self.\_\_convertToComplex(x, y, currentBounds),

self.maximumItEntry + 4 \* zoomNumber, self.\_\_colouring())]

pygame.event.clear(pygame.MOUSEBUTTONDOWN)

* + 1. **Running Main Mandelbrot Fractal**

#---------------------run-Subroutine--------------------------

#The 'get' functions are used to execute and return the required values needed to run the program in the different methods within the class. callFractal is the object

#--> of main Mandelbrot class. #callFractal.main() calls the main function within the class for the overall execution of the program

def run():

pygame.init()

callFractal = fractal()

callFractal.getBounds()

callFractal.getScale()

callFractal.getImageDimensions()

callFractal.getZoomSize()

callFractal.getAspectRatio()

callFractal.getMaximumIt()

callFractal.getScalingSpeed()

callFractal.getThemes()

callFractal.main()

#This specific function disallows the program from being run as soon as it is imported from another module (editing window and main menu)

#The run function is called upon when 'mandelbrotNEAOOP' is called from another module; the editing window

#Actual function: When the Python interpreter reads a file the \_\_name\_\_ variable is set as \_\_main\_\_ if the module is being run or as modules name if it's imported

if \_\_name\_\_ == "\_\_main\_\_":

run()

* 1. **RGB Slider**
     1. **Setting Up Sliders**

from tkinter import \*

#------------------------RGB-FUNCTIONS-----------------------------------

class mandelSlider:

def \_\_init\_\_(self, \*args):

#-------------------------MAIN-WIND0W-------------------------------------

#\_\_init\_\_ is used with \*args and \*kwargs to allow multiple keywords and arguments to be passed to functions later in the program

#The root window is constructed with the title, size (geometry) and resizeable disallows the user from changing the size of the window with boolean False value

#TopLevel must be used to be compatible with the main loginlogin window

root = Tk()

root.withdraw()

self.RGBWindow = Toplevel()

self.RGBWindow.title("RGB Slider Mandelbrot")

self.RGBWindow.geometry("800x300")

self.RGBWindow.resizable(False, False)

#--------------------------SLIDER-LABELS----------------------------------

#Using .place for the placement of the RGB sliders to show user which colour the slider is interacting with

self.redLabel = Label(self.RGBWindow, text="R").place(x=100, y=65)

self.greenLabel = Label(self.RGBWindow, text="G").place(x=100, y=115)

self.blueLabel = Label(self.RGBWindow, text="B").place(x=100, y=165)

#-----------------------------SLIDER-VALUES-------------------------------

#These are the slider value for red, green and blue. IntVar is used because integer values are being used to set the RGB values as integers from 1 to 255

self.redSliderValue = IntVar(name = 'redSliderValue')

self.greenSliderValue= IntVar(name = 'greenSliderValue')

self.blueSliderValue = IntVar(name = 'blueSliderValue')

#-----------------------------COLOUR-EXCEPTION-------------------------

#This is used to only allow the user to input the suitable values from 1 to 255 and only integers as an exception handling algorithm for the entry box

self.colouringCodes = StringVar()

checkerReg = self.RGBWindow.register(self.\_\_RGBLimits)

#------------------------------------SLIDER-ENTRIES----------------------

#These are entry values for each slider. The entries must be within the given range of 1 to 255 for the standard RGB values. Used to prevent range and type errors

#%P is the standard regex used for the validation of the exception handling in the entry box.

self.redEntry = Entry(self.RGBWindow, textvariable = self.redSliderValue)

self.redEntry.config(validate = "key", validatecommand = (checkerReg, "%P"))

self.greenEntry = Entry(self.RGBWindow, textvariable = self.greenSliderValue)

self.greenEntry.config(validate = "key", validatecommand = (checkerReg, "%P"))

self.blueEntry = Entry(self.RGBWindow, textvariable = self.blueSliderValue)

self.blueEntry.config(validate = "key", validatecommand = (checkerReg, "%P"))

#-----------------------------------COLOUR-BINDING-------------------------

#.bind allows the slider to be adjusted to where the user inputs the valid of the RGB value manually

#For example, if the redEntry is the 255 the slider will bind up and adjust itself to the end of the slider where 255 to show red colour in box

#Lambda is used as a nameless function for a short period of time between when the value is entered and when the slider itself is adjusted

self.redEntry.bind(lambda \_:self.\_\_inputLimits(self.redSliderValue))

self.greenEntry.bind(lambda \_:self.\_\_inputLimits(self.greenSlider))

self.blueEntry.bind(lambda \_:self.\_\_inputLimits(self.blueSliderValue))

self.redEntry.place(x=450, y=65, height=30, width=30)

self.greenEntry.place(x=450, y=115, height=30, width=30)

self.blueEntry.place(x=450, y=165, height=30, width=30)

#---------------------------------HEX COLOURS-----------------------------

#This is the rectangle at the top of the screen to show the hexidecimal equivalent of the colour shown in the box

self.hexColours = Entry(self.RGBWindow, textvariable = self.colouringCodes, state = 'readonly', foreground = "#777").pack()

#---------------------------------SLIDERS---------------------------------

#These are the sliders being constructed. The sliders can only go as far as the value of 255 as this is the standard RGB value allowed.

#from\_ is 1 which is where it starts and to =255 where it ends. The slider placement is horizontal

self.redSlider = Scale(self.RGBWindow, from\_ = 1, to = 255, length = 270, variable = self.redSliderValue, orient = HORIZONTAL)

self.redSlider.place(x=150, y=50)

self.greenSlider = Scale(self.RGBWindow, from\_ = 1, to = 255, length = 270, variable = self.greenSliderValue, orient = HORIZONTAL)

self.greenSlider.place(x=150, y=100)

self.blueSlider = Scale(self.RGBWindow, from\_ = 1, to = 255, length = 270, variable = self.blueSliderValue, orient = HORIZONTAL)

self.blueSlider.place(x=150, y=150)

#-----------------------------SWATCH-RGB-REPRESENTATION-------------------

#colourBox is a canvas Tkinter Widget to show that the colour is being changed using #ffffff as the hexidecimal value for the max RGB value

self.colourBox = Canvas(self.RGBWindow, background='#ffffff', height=95, width=95)

self.colourBox.place(x=500, y=80)

#-----------------------------TRACING-BACK-SLIDER-VALUE------------------

#.trace\_add is used in 'write' mode to set the colour and track it to change the colour of the swatch according to the adjustment of the slider

self.redSliderValue.trace\_add("write", self.\_\_settingFill)

self.greenSliderValue.trace\_add("write", self.\_\_settingFill)

self.blueSliderValue.trace\_add("write", self.\_\_settingFill)

* + 1. **Assigning Inputs to Colour**

#-----------------------------ASSIGNING-COLOURS--------------------------

#This is the main function used to assign the colour to the swatch canvas

#colourAssigned is the format specifier for the standard format of an RGB value. 02x is used to get 2 char outputs with mod function for each colour assignment

#swatch.configure will change the background of the Tkinter colour canvas according to the RGB value calculated.

#Try and except used to try the values being entered, if not valid then it is passed

def \_\_settingFill(self, \*args):

try:

redAssign = int(self.redSliderValue.get())

greenAssign = int(self.greenSliderValue.get())

blueAssign = int(self.blueSliderValue.get())

hexConvertor = ("#%02x%02x%02x")

colourAssigned = hexConvertor % (redAssign, greenAssign, blueAssign)

self.colourBox.configure(background = colourAssigned)

self.colouringCodes.set(colourAssigned)

except:

pass

* + 1. **Exception Handling of Hex RGB Values and Slider Entries**

#---------------------------VALIDATING-HEX-COLOUR-RANGE-----------------

#Function use to validate the range of which hex values are allowed: 1-255

#If the value is equal to or above 1 AND equal to or below 255, it will return True and the sliders can run as normal

#If the value is empty, it will be returned as True

def \_\_RGBLimits(self, inp):

if inp.isdigit():

if int(inp) >= 1:

if int(inp) <= 255:

return True

elif inp == "":

return True

elif inp == "":

return True

else:

return False

return False

#---------------------------VALIDATING-SLIDER-ENTRY---------------------

#The slider will 'focus' depending on the input by the user

def \_\_inputLimits(self, \*args):

textEntry = args[0]

inputRGB = str(textEntry)

if inputRGB == 'redSliderValue':

self.redEntry.focus()

elif inputRGB == 'blueSliderValue':

self.blueEntry.focus()

elif inputRGB == 'greenSliderValue':

self.greenEntry.focus()

* + 1. **Run RGB Slider**

#-------------------------RUN-------------------------------------------

#Used to run the main root window in a main loop until it is exited

def run(self, \*args):

self.RGBWindow.mainloop()

#--------------------------------MAIN-----------------------------------

#callSlider is used as an object constructor to call the RGBHexConversion class

#.run() is used to call the 'run' function from the class using the callSlider object

def main():

callSlider = mandelSlider()

callSlider.run()

#Important if statement to make sure this module does not run straight away when importing into another module (loginlogin)

if \_\_name\_\_ == '\_\_main\_\_':

main()

* 1. **Julia Set Fractal (Extension Objective Met)** 
     1. **Julia Fractal Setup**

#--------------EXTENSION-JULIA-SET----------------

#importing necessary libraries and initialising pygame for OOP

import pygame

import math

from pygame.locals import \*

import sys

pygame.init()

#--------------------------juliaFractal---------------------------

#jBounds is the boundaries for the X and Y values for the corners of plotting pixels and RES is the resolution value. Blend type is for colouring.

#jMaxIt is the maximum iterations for the Julia set which determines the resolution and how transparent the image is. C is the complex constant for the 'z' iteration

#--> equation; used to determine the iterations too in the standard calculations for the Julia set.

class juliaFractal():

def \_\_init\_\_(self):

self.width = 512

jBounds = [-1.6, -1, 1, 1.6]

self.minimumX = jBounds[0]

self.maximumX = jBounds[3]

self.minimumY = jBounds[1]

self.maximumY = jBounds[2]

self.RES = 1

self.jMaxIt = 2000

self.blendType = 1

self.C = complex(-0.79, 0.155)

* + 1. **Setter Functions**

#-------------------setBoundaries----------------------------

#Used to set the boundary values into float values for the purpose of formatting for calculation

def setBoundaries(self):

self.minimumX = float(self.minimumX)

self.maximumX = float(self.maximumX)

self.minimumY = float(self.minimumY)

self.maximumY = float(self.maximumY)

return self.minimumX, self.maximumX, self.minimumY, self.maximumY

#------------------setHeight-----------------------------------

#Standard calculation used for the height of the Julia set window as discovered in research of the Julia set fractal.

#Converted to integer for formatting purposes for later calculations

def setHeight(self):

height = (self.maximumY - self.minimumY) \* float(self.width) / (self.maximumX - self.minimumX)

self.height = int(height)

return self.height

#------------------setDisplay-----------------------------------

#dimensions used with width and height values calculated and set. Surface used to initiate the pygame window

#juliaScreen uses format of PixelArray from PyGame to plot the rows of data for the Julia set to speed up optimisation results

def setDisplay(self):

imageDimensions = self.width, self.height

surface = pygame.display.set\_mode(imageDimensions)

self.juliaScreen = pygame.PixelArray(surface)

return self.juliaScreen

#-----------------setIteration-----------------------------------

#Standard calculation from research for the number of increments according to other values in program when mapping the pixels to points in the complex Julia plane

def setIteration(self):

self.incrementNo = self.RES \* ((self.maximumX - self.minimumX)/(float(self.width)))

return self.incrementNo

* + 1. **Julia Set Pixel/Coordinates Complex Number Conversions**

#---------------complexToX---------------------------------------

#Calculation for converting the X coordinates for the mapping of the pixels in the complex Julia plane by calculating equivalent complex values for pixel conversion

#plotting is the variable when calling this function to plug in the current coordinate value for the pixel conversion

def complexToX(self, plotting):

return int(((self.width)/(self.maximumX - self.minimumX) \* plotting) + (self.width \* self.minimumX) / (self.minimumX - self.maximumX))

#--------------complexToY-----------------------------------------

#Same algorithm as the complexToX function but with the use of Y coordinates and using height rather than width

def complexToY(self, plotting):

return int(((self.height)/(self.minimumY - self.maximumY) \* plotting) + (self.height \* self.maximumY) / (self.maximumY - self.minimumY))

* + 1. **Julia Set Iteration Algorithm**

#--------------juliaIteration--------------------------------------

#This is the main calculation used for the iteration and the range of pixels being plotted

#%10.5f is the standard format for plotting the axes necessary for the julia set and the values are converted to float in itValues where the iteration values are

#--> stored in a list format

#the base is the beginning of the iteration and it stops once the condition (base case) is met in the while loop where the increments are calculated

def juliaIteration(self, initiate, halt, iteration):

base = initiate

axesPlot = ('%10.5f' % base)

itValues = [float(axesPlot)]

while base < halt:

base += iteration

itValues = itValues + [float('%10.5f' % base)]

return itValues

* + 1. **Julia Axes Plotting**

#-------------juliaPlot--------------------------------------------

#The positions are calculated to store the values being increments and therefore iterated for storing the necessary axis points by calling juliaIteration

#Position 1 is for the x coordinate and therefore the width of the axes. The conversion to X is called for the x coordinate of the pixels to convert to coordinates

#The axes in the x position (1) is then using the conversion and the RGB values of 255, 255, 255 to show the user the centre of the axis with the white line

#The same process is repeated for the 'for' loop in the y coordinates for position 2 but here the y coordinates and pixels are converted

#pygame.display.update() is used to plot the axes in run time on the pygame display

def juliaPlot(self):

position1 = self.juliaIteration(self.minimumX, self.maximumX, self.incrementNo)

position2 = self.juliaIteration(self.minimumY, self.maximumY, self.incrementNo)

for x in position1:

self.juliaScreen[self.complexToX(x) - 1, self.complexToY(0)] = (255, 255, 255)

for y in position2:

self.juliaScreen[self.complexToX(0), self.complexToY(y) - 1] = (255, 255, 255)

pygame.display.update()

* + 1. **Main Julia Set PyGame Window Plotting**

#-------------juliaPlot--------------------------------------------

#The positions are calculated to store the values being increments and therefore iterated for storing the necessary axis points by calling juliaIteration

#Position 1 is for the x coordinate and therefore the width of the axes. The conversion to X is called for the x coordinate of the pixels to convert to coordinates

#The axes in the x position (1) is then using the conversion and the RGB values of 255, 255, 255 to show the user the centre of the axis with the white line

#The same process is repeated for the 'for' loop in the y coordinates for position 2 but here the y coordinates and pixels are converted

#pygame.display.update() is used to plot the axes in run time on the pygame display

def juliaPlot(self):

position1 = self.juliaIteration(self.minimumX, self.maximumX, self.incrementNo)

position2 = self.juliaIteration(self.minimumY, self.maximumY, self.incrementNo)

for x in position1:

self.juliaScreen[self.complexToX(x) - 1, self.complexToY(0)] = (255, 255, 255)

for y in position2:

self.juliaScreen[self.complexToX(0), self.complexToY(y) - 1] = (255, 255, 255)

pygame.display.update()

#-----------drawJulia----------------------------------------------

#the positionX and positionY are the same as the position 1 and 2 in the juliaPlot function but the function is called upon again for the plotting of the pixels

#---> rather than the axes itself

#2 is the radius limit until the Julia set escapes and it is not counted anymore

#The plotting fill is the transparency of the set itself and 80 shows full visibility of the set with corresponding RGB values too

#The colourblend is the standard RGB values for the set

def drawJulia(self):

pygame.display.set\_caption("Julia Set Preview")

positionX = self.juliaIteration(self.minimumX, self.maximumX, self.incrementNo)

positionY = self.juliaIteration(self.minimumY, self.maximumY, self.incrementNo)

radiusLimit = 2

plottingFill = 80

colourBlend = (255, 90, 90)

#The x and y 'for' loops are for the plotting of the x and y coordinates with the conversion to pixels

#z is the standard iteration equation for the Julia set which is the same as the Mandelbrot set equation where complex values are used for the complex plan (x,y)

#currentP is the final value of the iteration which is calculated. i = 0 is to initialise the loop.

#The while loop is looping through the iteration until the radius limit is met and until the maximum iterations is met as it iterates through according to the limit

#--> of the radius until the pixels are not in the Julia set and until the iteration value is met

for x in positionX:

for y in positionY:

z = complex(x, y)

currentP = z

i = 0

while abs(z) < radiusLimit and i < self.jMaxIt:

z = z\*z + self.C

i += 1

#if the iteration value within the Julia set loop has met the maximum Iterations when the x and y coordinates are converted to pixels using the complex number format

#--> of imaginary and real numbers. If the pixel coordinates are less than correspinding height and width then they are coloured using the chosen colour blend RGB

#--> using the screen of pygame as the 2D array of x and y coordinates

if i == self.jMaxIt:

xCoordinate = self.complexToX(currentP.real)

yCoordinate = self.complexToY(currentP.imag)

if xCoordinate < self.width and yCoordinate < self.height:

if self.blendType == 1:

self.juliaScreen[xCoordinate, yCoordinate] = (colourBlend)

#if the iteration value within the Julia set loop is still under the value of the iterations when the x and y coordinates are still converted to pixels using the

#--> complex plane with the real and imaginary format of the complex numbers of the final iteration value

#the 'if' statement for blend type is used to colour the pixels as long as the iteration value is under the maximum iteration and the dotting blend is used to

#--> blend in the pixels when looping with the pixels that are used in the final calculation in the loop. background fill is the blue background fill used with RGB

#If the x and y coordinate positions are less than the height and width and they are over the value of 0, they are plotted using the pygame display 2d array according

#--> to the blue background colour. the pygame display is then updated

if i < self.jMaxIt:

xCoordinate = self.complexToX(currentP.real)

yCoordinate = self.complexToY(currentP.imag)

if self.blendType == 1:

dottingBlend = plottingFill \* math.log10(i + 1)

backgroundFill = (dottingBlend, 50 , 150)

if xCoordinate < self.width and yCoordinate < self.height:

if yCoordinate > 0 and xCoordinate > 0:

self.juliaScreen[xCoordinate, yCoordinate] = (backgroundFill)

pygame.display.update()

#This is the standard loop used to quit the pygame window when needed

done = True

while not done:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

* + 1. **Run Julia Set**

#---------------------Julia--------------------------

#Used to call all the functions to run the Julia set.

def Julia():

callJulia = juliaFractal()

callJulia.setBoundaries()

callJulia.setHeight()

callJulia.setDisplay()

callJulia.setIteration()

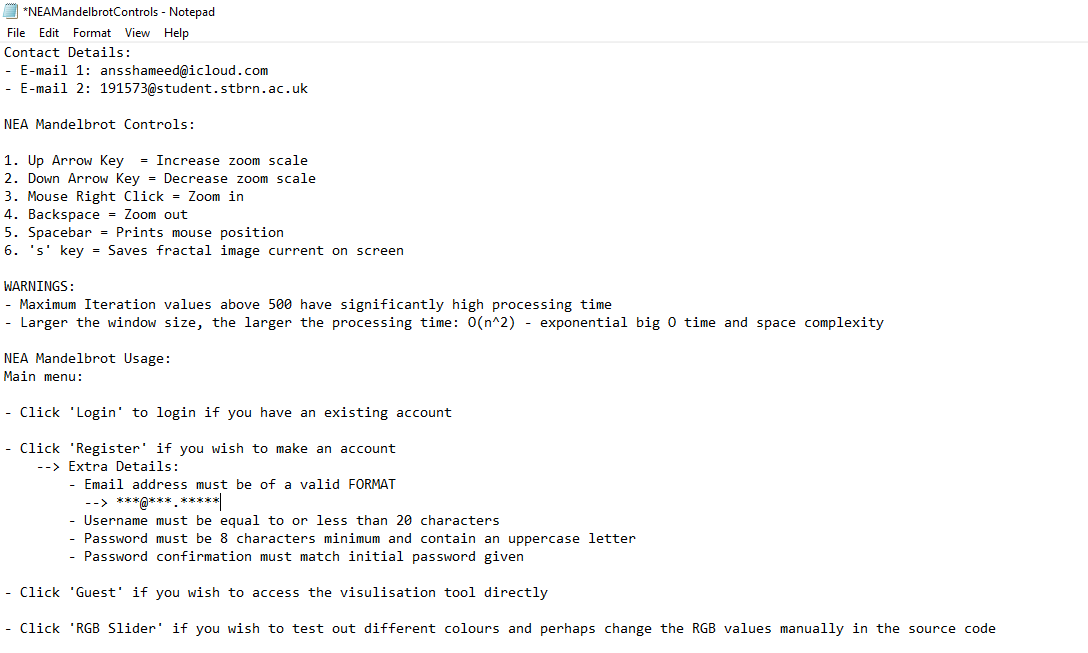
callJulia.juliaPlot()

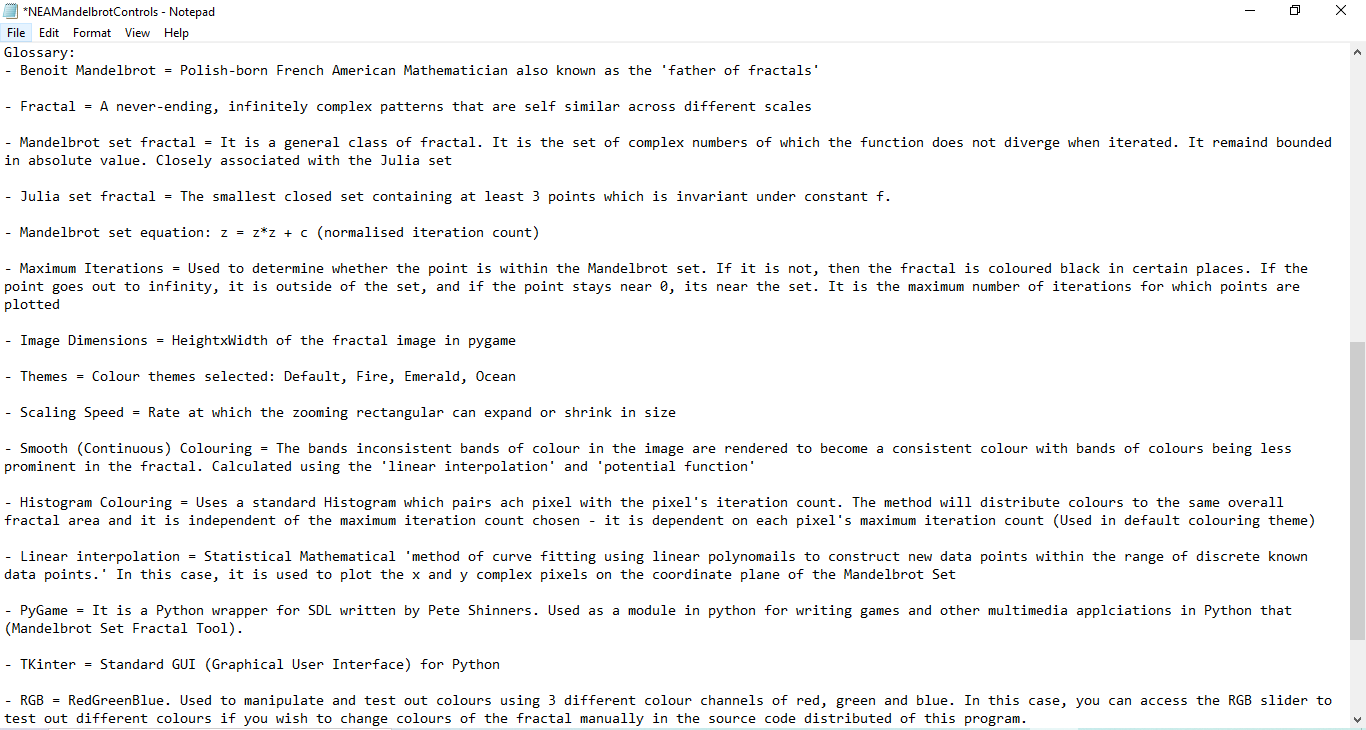
callJulia.drawJulia()

if \_\_name\_\_ == "\_\_main\_\_":

Julia()

* 1. **Help Menu Sample Screenshots**





1. **Testing**
   1. **Input Validation Testing**
      1. **Registration System**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Purpose of Test** | **Test Data** | **Expected Result** | **Actual Result** | **Actions Needed/Comments** |
| 1 | Check if username is equal to or less than 20 characters in register subclass using regex; usernameInfo variable – Range Check | HelloMyName  IsAnssHameedWhat  IsYours | Red warning message: ‘Please fill the required details’ and doesn’t process registration (username not written to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (username not written to text file) | Successful |
| 2 | Check if username is just above 20 characters in register subclass using regex; usernameInfo variable – Boundary Data Check (21 characters) | HelloIsMyNameIsAnssss | Red warning message: ‘Please fill the required details’ and doesn’t process registration (username not written to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (username not written to text file) | Successful |
| 3 | Check is username is within the range of 20 characters or below in register subclass using regex; usernameInfo variable (Range check) | Anss123 | Correctly processed as long as other fields are filled correctly too (username will be written to text file) | Correctly processed as long as other fields are filled correctly too (username will be written to text file) | Successful |
| 4 | Username must be of string format allowing digits 0-9, with no symbols apart from: @#$%^&+=, in register subclass using regex; usernameInfo variable (type check/erroneous data check) | Anss()\*%<> | Red warning message: ‘Please fill the required details’ and doesn’t process registration (username not written to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (username not written to text file) | Successful |
| 5 | Username meets all the criteria; usernameInfo variable in register subclass (Normal data check) | Anss123 | Correctly processed as long as other fields are filled correctly too (username will be written to text file) | Correctly processed as long as other fields are filled correctly too (username will be written to text file) | Successful |
| 6 | Username is present in username field entry box in register subclass; usernameInfo variable (Presence check) | ‘’ (empty) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (username not written to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (username not written to text file) | Successful |
| 7 | Password must contain at least one upper case letter which is checked using regex criteria of: A-Za-z – using passwordInfo variable in register subclass (Type check) | anssishere12 | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Successful |
| 8 | Password is checked at 7 characters (just below 8) which is checked against regex criteria using passwordInfo variable in register subclass (range/boundary data check) | Iphone1 | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Successful |
| 9 | Password must be at least 8 characters long which is checked by regex criteria: {8,} using passwordInfo in register subclass (range check) | Hi | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Successful |
| 10 | Password must contain some integers which is checked against regex criteria: 0-9, using passwordInfo variable in register subclass (type check) | HelloItsMe | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Successful |
| 11 | Password must contain appropriate symbols with regex criteria: @#$%^&+=, using passwordInfo variable in register subclass (type check) | HelloItsAnss<>\*£” | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Successful |
| 12 | Password field entry is empty using (presence check) | “” (empty) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (password not written to text file) | Successful |
| 13 | Password meets all the regex criteria in passwordInfo variable in register subclass: [A-Za-z0-9@#$%^&+=]{8,} (normal data check) | HelloItsAnss89! | Correctly processed as long as other fields are filled correctly too (password will be written to text file) | Correctly processed as long as other fields are filled correctly too (password will be written to text file) | Successful |
| 14 | Email must contain @ symbol in emailInfo variable in register subclass with regex criteria: [A-Za-z0-9\.\+\_-][+@[A-Za-z0-9\.\_-]+[a-zA-Z]\*$](mailto:+@[A-Za-z0-9\._-%5d+%5ba-zA-Z%5d*$), (type check) | Ansshameedgmail.com | Red warning message: ‘Please fill the required details’ and doesn’t process registration | Red warning message: ‘Please fill the required details’ and doesn’t process registration | Successful |
| 15 | Email must contain ‘.’ after name of email service provided in emailInfo variable in register subclass when meeting regex criteria (type check) | Ansshameed:icloudcom | Red warning message: ‘Please fill the required details’ and doesn’t process registration | Red warning message: ‘Please fill the required details’ and doesn’t process registration | Successful |
| 16 | Email must meet all regex criteria: [A-Za-z0-9\.\+\_-][+@[A-Za-z0-9\.\_-]+[a-zA-Z]\*$](mailto:+@[A-Za-z0-9\._-%5d+%5ba-zA-Z%5d*$) in emailInfo variable in register subclass (normal data check) | ansshameed@icloud.com | Correctly processed as long as other fields are filled correctly too | Correctly processed as long as other fields are filled correctly too | Successful |
| 17 | Email field must contain data for emailInfo variable in register subclass (presence check) | “” (empty) | Red warning message: ‘Please fill the required details’ and doesn’t process registration | Red warning message: ‘Please fill the required details’ and doesn’t process registration | Successful |
| 18 | Password confirmation field must match the original password input with passwordInfo2 variable (normal data check) | password = HelloItsAnss23!  Password confirmation = HelloItsAnss23! | Red warning message: ‘Please fill the required details’ and doesn’t process registration (will not write original password to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (will not write original password to text file) | Successful |
| 19 | Password confirmation field must contain data (presence check) | “” (empty) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (will not write original password to text file) | Red warning message: ‘Please fill the required details’ and doesn’t process registration (will not write original password to text file) | Successful |

* + 1. **Mandelbrot Editing Input Values**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Purpose of Test** | **Test Data** | **Expected Result** | **Actual Result** | **Actions Needed/Comments** |
| 1 | If maximum iterations value is above 500, red warning message should print (boundary testing) | 50, 51 | Red warning printed ‘Maximum iterations values over 500 may take more processing time’ – still processed | Red warning printed ‘Maximum iterations values over 500 may take more processing time’ – still processed | Successful |
| 2 | If maximum iterations value is below 0 (negative) should not be processed (erroneous testing) | -250 | Fractal must not process | Fractal must not process | Successful |
| 3 | If maximum iterations value is anything other than an integer or floating value (e.g. hello), it should not be processed (erroneous/type data testing) | Hello | Fractal must not process | Fractal must not process | Successful |
| 4 | Maximum iterations value entry field must contain value (presence check) | “” (empty) | Fractal must not process | Fractal must not process | Successful |
| 5 | When maximum iterations value is low (below default 50 value) it should be quicker process and lower resolution (normal data check) | 25 | Fractal is of lower quality and processed in shorter time in seconds | Fractal is of lower quality and processed in 20 seconds  **6.2.1.** | Successful |
| 6 | When maximum iterations value is high (above 500) it should be a significant processing time and much higher resolution (normal data check) | 600 | Fractal is of much higher resolution and processed after a longer time in seconds | Fractal is of much higher resolution and processed after 150 seconds – more detail on outer resolution of Mandelbrot fractal where each pixel is more colourised with increased detail.  **(6.2.2)** | Successful |
| 7 | Correct data is input and must be processed (normal data check) | 50 | Fractal is processed of normal standard | Fractal is processed of normal standard  **(6.2.3.)** | Successful |
| 8 | Negative scaling speed value must not be processed (normal data check/type check) | -0.5 | Fractal must not process | Fractal must not process | Successful |
| 9 | If scaling speed of a data type other than an integer or floating data type (string characters), it must not be processed (erroneous/data type checking) | Hello | Fractal must not process | Fractal must not process | Successful |
| 10 | If scaling speed is very high according to image dimensions, rectangular zoom box size should increase/decrease faster when pressing up or down arrow keys (normal data check) | 1.5 | Rectangular zoom box must change faster in size | Rectangular zoom box changes faster in size | Successful |
| 11 | If scaling speed is low according to image dimensions, rectangular zoom box size should increase/decrease slower when pressing up or down arrow keys (normal data check) | 0.0001 | Rectangular zoom box must change slower in size | Rectangular zoom box changes slower in size – Evidence of rectangular zoom box changing in size: **(6.2.4 🡪 6.2.5)** | Successful |
| 12 | Scaling speed entry field must contain a value (presence check) | “” (empty) | Fractal must not process | Fractal not processed | Successful |
| 13 | Default scaling speed value used to check zoom is working appropriately (normal data check) | 0.01 | Fractal processed with average speed of zoom box changing | Fractal processed with average speed of zoom box changing | Successful |

* + 1. **RGB Value Slider Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Purpose of Test** | **Test Data** | **Expected Result** | **Actual Result** | **Actions Needed/Comments** |
| 1 | Check if all sliders move on manual entry box input with colour change (Normal data check) | 60 | Slider should move to 60 value and colour change in swatch box | Slider moved to 60 value and colour change in swatch box  **(6.2.6.)** | Successful |
| 2 | Check result of inputting a string rather than an integer or float (Erroneous data check) | RGB | Slider should not respond to entry | Slider did not react on input of string | Successful |
| 3 | Check result of inputting value of 256 (erroneous/range/boundary data check) | 256 | Slider should move to maximum value of 255 | Slider moved to value of 255 and did not allow user to input value over 255 (256) in entry box | Successful |
| 4 | Check result of negative integer input (erroneous/range/boundary data check) | -1 | Convert the entry of negative number into positive number | Slider moved to positive equivalent of negative number (when inputting -1 the slider moved to value of 1) | Successful |
| 5 | Check result of inputting a float/decimal number (normal/type/erroneous data check) | 57.3, 60.7 | Should round down to nearest unit if below .5 and round up if above .5 | Input 57.3: Rounded down to 57  Input 60.7: Rounded up to 61 | Successful |
| 6 | Check result of an empty entry box or input of 0 (normal/presence data check) | 0, “” | Input ‘0’: Automatically change to 1  Input “”: Automatically changes to 1 | Input ‘0’: Automatically change to 1  Input “”: Automatically changes to 1 | Successful |

* 1. **File Handing Test**
     1. **Login and Registration System**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Purpose of Test** | **Test Data** | **Expected Result** | **Actual Result** | **Actions Needed/Comments** |
| 1 | Write username to login text file if it meets criteria (Normal data check) | AnssHameed | Written to login text file on a new line | Written to login text file on a new line | Successful |
| 2 | Write password to login text file if it meets criteria (Normal data check) | HelloItsMe123 | Written to the login text file on a new line (separate to username) | Written to the login text file on a new line (separate to username) | Successful |
| 3 | Read entered username from login text file when logging in if it meets criteria (normal data check) | Reads login text file for username input from user: AnssHameed | Grants user access to Mandelbrot Fractal editing window (if password correct too) | Grants user access to Mandelbrot Fractal editing window (if password correct too) | Successful |
| 4 | Read entered username from login text file if it doesn’t meet criteria/not registered (normal data check) | Reads login text file for username (incorrectly entered) input from user: HameedAnss | Prints red error message: ‘Invalid username or password’ | Prints red error message: ‘Invalid username or password’ | Successful |
| 5 | Read entered password from login text file when logging in if it meets criteria/is registered (normal data check) | Reads login text file for password entered correctly: HelloItsMe123 | Grants user access to Mandelbrot Fractal editing window (if username correct too) | Grants user access to Mandelbrot Fractal editing window (if username correct too) | Successful |
| 6 | Read entered password from login text file when logging in if it doesn’t meet criteria/not registered (normal data check) | Reads login text file for password entered incorrectly: helloItsnotMe293 | Prints red error message: ‘Invalid username or password’ | Prints red error message: ‘Invalid username or password’ | Successful |

* + 1. **Mandelbrot Editing Tool (Applying values)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No** | **Test Purpose** | **Test Data** | **Expected Result** | **Actual Result** | **Action Needed/Comments** |
| 1 | Writing maximum iterations value to ‘maximumIterations.txt’ if valid (normal data check) | 50 into ‘Maximum iterations’ entry field | Write 50 into ‘maximumIterations.txt’ on new line | Write 50 into ‘maximumIterations.txt’ on new line | Successful |
| 2 | Writing HEIGHT and WIDTH values to ‘imageDimensions.txt’ according to dimensions selected from dropdown menu (normal data check) | ‘512x512’ selected from dropdown menu | Write 512 (HEIGHT value) on one line in ‘imagedDimensions.txt’ and 512 (WIDTH) on next new line | Write 512 (HEIGHT value) on one line in ‘imagedDimensions.txt’ and 512 (WIDTH) on next new line | Successful |
| 3 | Writing scaling speed value to ‘SS.txt’ if it meets criteria (normal data check) | 0.1 | Write 0.1 in new line of ‘SS.txt’ | Write 0.1 in new line of ‘SS.txt’ | Successful |
| 4 | Accessing and reading the ‘maximumIterations.txt’ from ‘mandelbrotNEAOOP’ module (normal data check) | Reading 50 from text file | Value of 50 must be read from ‘maximumIterations.txt. using direct access and applying it to fractal with conversion to integer/float number | Value of 50 read from ‘maximumIterations.txt’  **(6.2.7.)** | Successful |
| 5 | Accessing and reading the ‘imageDimensions.txt’ from ‘mandelbrotNEAOOP’ model with height and width individually read  (normal data check) | Reading 512 from one line and 512 from next new line. | Value of 512 must be read from ‘imageDimensions.txt’ as HEIGHT value and value of 512 on next line read as WIDTH value and converted from text (string) to integer and applied | Value of 512 read from first line as HEIGHT and read from separate, new line with CSV as WIDTH and converted to integer values from text (string)  Example – 512x512:  **(6.2.8.)**  Example – 320x200 (HEIGHT = 320, WIDTH = 200):  **(6.2.9.)** | Successful |
| 6 | Accessing and reading ‘SS.txt’ for scalingSpeed value (normal data check) | 0.1 | Value of ‘scalingSpeed’ or in this case, 0.1, read from ‘SS.txt’ and applied | Value of ‘scalingSpeed’ or in this case, 0.1, read from ‘SS.txt’ and applied | Successful |
| 7 | Writing the values from RGB 2D array to ‘mandelbrotThemes’.txt | ‘Sunset’ selected from drop down menu | Values of ‘Sunset’ themed RGB values must be written to ‘mandelbrotThemes.txt’ text file with sets of RGB values written row by row | Values of ‘Sunset’ themed RGB values written to ‘mandelbrotThemes.txt’ with values from 2D array written row by row | Successful (Initially, values were written with the ‘[]’ 2D array brackets, now fixed) |
| 8 | Accessing 2D array RGB values written in ‘mandelbrotThemes.txt’ | Reading all of the RGB set values from ‘sunset’ theme | Values of 2D array ‘Sunset’ RGB values read and shows fractal with ‘Sunset’ theme | **(6.2.10)** | Successful (Initially, values not read in correct format but now read as integer and not float) |

* + 1. **Vernam Cipher**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No** | **Test Purpose** | **Test Purpose** | **Expected Result** | **Actual Result** | **Actions Needed/Comments** |
| 1 | One time pad written to oneTimePad.txt | Writing unique OTP which is only used once and is same length as password | Unique one time pad which is same length as password and random (to a certain extent as it is using ‘random’ library in Python) | **(6.2.11.)** | Successful |
| 2 | Encrypted password written to ciphertext.txt | Writing encrypted password according to Vernam cipher | Unique encryption of password | **(6.2.12.)** | Successful |
| 3 | Plaintext of password written to vernamPass.txt | Writing plaintext of password to file | Password written in first line of file | **(6.2.13.)** | Successful |
| 4 | Decrypted password written to decryptionRev.txt | Decrypted password must be written using decipher algorithm | Original plaintext (decrypted password) written to file | **(6.2.14)** | Successful |

* 1. **Interface Testing**
     1. **Main Menu**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Result** | **Actual Result** | **Actions Needed/Comments** |
| 1 | Testing login button on main menu window | Mouse click on Login button from main menu | Show login window instantly | Shows login window instantly  **(6.2.15.)** | Successful |
| 2 | Testing login button on the login window to open editing window | Mouse click in Login button from the login window. Data: Correct login criteria met and a failed login attempt | If login criteria met, shows editing window instantly. If login criteria not met (logs in with un-registered details), show error message. | Shows editing window if correct and shows error message when not met.  **(6.2.16.)**  **(6.2.17.)** | Successful |
| 3 | Testing register button from the main menu to load up registration window | Mouse click on register button from main menu | Open registration window instantly | Shows registration window instantly  **(6.2.18.)** | Successful |
| 4 | Testing register button from the registration window | Mouse click on the Register button from registration window | If registration criteria is met, display success message and write data to text file. If registration failed then show error message when Register button clicked. | When successful, success message displayed and values written to text file. When failed, error message displayed.  **(6.2.19.)**  **(6.2.20.)**  **(6.2.21.)** | Successful |
| 5 | Testing Guest button from main menu | Mouse click on the Guest button from the main menu | Instantly opens editing window. | Instantly opens editing window.  **(6.2.22.)** | Successful |
| 6 | Testing RGB Slider button from main menu | Mouse click on the RGB Slider button from the main menu | Instantly opens RGB slider tool | Instantly opens RGB slider tool  **(6.2.23.)** | Successful |
| 7 | Testing Guest button from main menu | Mouse click on the Guest button from the main menu | Instantly opens editing window | Instantly opened editing window  **(6.2.24.)** | Successful |
| 8 | Testing Help button from the main menu | Mouse click on Help button from the main menu | Instantly opens ‘Help’ menu (notepad text file) | Instantly opened Help menu  **(6.2.25.)** | Successful |
| 9 | Testing Julia Set Preview button from main menu | Mouse click on ‘Julia Set Preview’ button from the main menu | Instantly opens the PyGame window for the Julia set | **(6.2.26.)** | Successful |

* + 1. **Editing Window**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Purpose of Test** | **Test Data** | **Expected Result** | **Actual Result** | **Actions Needed/Comments** |
| 1 | Testing ‘Apply Values’ button in editing window | Maximum Iterations: 1000  Image Dimensions: 300x200  Colour Theme: Default  Scaling Speed: 0.01  Colour Theme: Default | Open fractal with fairly high resolution (1000 max its), window size of 300x200, default colour theme and resizing of zoom box with speed 0.01 and ‘Default’ colour theme. | **(6.2.27)** | Successful |

* + 1. **Fractal Interface Window**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No** | **Test Purpose** | **Test Data** | **Expected Result** | **Actual Result** | **Actions needed/comments** | |
| 1 | Up arrow key to increase size of rectangular zoom box | Click up arrow key and have scaling speed at 0.01 | Increase in size of rectangular zoom box at average speed of 0.01 | Increase in zoom box size  **(6.2.28.) 🡪 (6.2.29.)** | Successful | |
| 2 | Down arrow key to decrease size of rectangular zoom box | Click down arrow key and scaling speed: 0.01 | Decrease size in rectangular zoom box at average speed of 0.01 | Decrease in zoom box size  **(6.2.30.) 🡪 (6.2.31.)** | Successful | |
| 3 | Zooming into specific location of fractal | Mouse click at desired position to zoom into fractal | Zoomed in fractal image according to where the mouse has been clicked on pygame window. The zoom must be correct according to the placement of the borders of red, rectangular zoom box. | **(6.2.32.) 🡪 (6.2.33.) zoomed in** | Successful | |
| 4 | Zooming out of fractal to previous image prior to zooming in | Press BACKSPACE key | Print previous image of fractal prior to zooming in | Before zooming in:  **(6.2.34.)**  Zoomed in:  **(6.2.35.)**  Zoomed back out:  **(6.2.36.)** | Successful | |
| 5 | Print ‘Processing…’  Message when zooming in or out of fractal image. | Click anywhere on fractal image to zoom in | Printed ‘Processing…’ message when interacted with fractal window | **(6.2.37.)** | Successful | |
| 6 | Testing boundaries of rectangular zoom box to make sure image is being zoomed into in right area | Click anywhere on fractal image to zoom into image | Must zoom into right area according to boundaries of zoom box | **(6.2.38.) 🡪 (6.2.39.)** | Successful | |
| 7 | Testing if complex number coordinates change according to movement of mouse across screen | Move mouse around on fractal screen | Must print the complex number coordinates (imaginary and real) according to movement of mouse cursor | Correctly prints complex number of coordinates according to movement of mouse cursor across the fractal screen (Cartesian coordinate system) | | Successful |
| 8 | Printing current mouse coordinates in Python shell | Click SPACEBAR | Prints current mouse position on fractal image onto the Python shell | **(6.2.40.)** | | Successful |
| 9 | Testing result of double clicking or more consecutive clicks on fractal window (Erroneous data checking) | Click mouse button to zoom multiple times in a row | Suspends operation and must zoom in only once | Suspends operation and only zooms in once, although there is slight lag and ‘(Not Responding)’ message appears on Pygame window for 5-10 seconds. Operation then continues as normal (zooming in or out). | | Successful |
| 10 | Testing placing rectangular zoom box on edge of fractal window (Boundary data checking) | Place zoom box on edge of fractal window and click mouse to zoom in | Zooms in to edge of window by performing correct calculation (regardless of half of zoom box not being visible on fractal window) | **(6.2.41.) 🡪 (6.2.42.)** | | Successful |
| 11 | Capturing and saving current image in Pygame window (current fractal zoom) | Press ‘s’ on keyboard | Saves current fractal image | Saves current fractal image to directory depending on user’s chosen location on device | | Successful |

* + - 1. **Fractal Timing Testing**

(Note: These results will be dependent on the hardware being used – e.g. some devices can handle high maximum iteration values better than others.) I did 3 different tests on 3 different laptops for experimental testing purposes of optimisation and speed. I did a second round of timing tests and made a line graph for a visual representation of how quick the fractal loads depending on the two variables of the image dimensions (major determinant of how quick fractal loads) and the time (seconds) of how long it took to load the fractal. The optimisation was determined on which laptop was used and the processor power of each. – **(6.2.49.)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Result (seconds)** | **Actual Result (seconds)** | **Actions Needed/Comments** |
| 1 | Timing of zoom with average maximum iterations value and average window size (Normal data check) | Maximum iterations: 50  Image Dimensions: 512x512 | 30-40s | Tests: 39.80s, 37.81s, 38.90s  Average: 39.14s | Successful |
| 2 | Timing of zoom with low maximum iterations value and average window size  (Boundary/range data check) | Maximum iterations: 20  Image Dimensions:  512x512 | 25-35s | Tests: 32.18s, 31.01s, 31.57s  Average: 31.59s | Successful |
| 3 | Timing of zoom with high maximum iterations value and average window size (Boundary/range data check) | Maximum Iterations: 500  Image Dimensions: 512x512 | 50-70s | Tests: 62.02s, 59.75s, 55.05s  Average: 58.94 | Successful |
| 4 | Timing of zoom with small image dimensions and average maximum iterations (Boundary/range data check) | Maximum iterations: 50  Image Dimensions: 320x200 | 10-20s | Tests: 6.63s, 6.57s, 6.10s  Average: 6.43s | Successful (Quicker than predicted) |
| 5 | Timing of zoom with large image dimensions and average maximum iterations value (Boundary/range data check) | Maximum iterations: 50  Image Dimensions:  800x600 | 50-75s | Tests: 70.47s, 65.45s, 59.99s  Average: 65.30s | Successful |

* 1. **OOP (Object-Oriented Programming) Class Testing**
     1. **Main Menu**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Result** | **Actual Result** | **Actions Needed/Comments** |
| 1 | Test if super().\_\_init\_\_() works to access methods of ‘main’ class from ‘register’ and by using inheritance of ‘main’ class with ‘main’ parameter as object | Click register button | The register window opens by accessing \_\_pack\_\_ method from ‘main’ class | Register window opens from ‘Register’ button by accessing \_\_pack\_\_ method from ‘main’ class | Successful |
| 2 | Test if super().\_\_init\_\_() works to access methods of ‘main’ class from ‘login’ and by using inheritance of ‘main’ class with ‘main’ parameter as object | Click login button | The login window opens by accessing \_\_pack\_\_ method from ‘main’ class | Login window opens from ‘Register’ button by accessing \_\_pack\_\_ method from ‘main’ class | Successful |
| 3 | Test ‘mainCall’ object to call ‘main’ class | Run main menu module | The main menu must open | The main menu opens by running main menu module | Successful |
| 4 | Ensure whole program runs by accessing classes through ‘self’ parameter in each method | Run program and make sure all windows work | All windows work: login, register, editing, help, RGB slider | All windows work: login, register, editing, help, RGB slider | Successful |

* + 1. **Editing Window**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Result** | **Actual Result** | **Actions Needed/Comments** |
| 1 | Ensure entire program runs by accessing methods through ‘self’ parameter in each method | Run program and make sure all input values work | Every input must work | Every input works | Successful |
| 2 | Use ‘editCall’ object to call entire ‘editing’ class when program is run | Run program and ensure no errors occur | Program works with no errors | Editing window works with no errors | Successful |

* + 1. **Mandelbrot Fractal Window (PyGame)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Result** | **Actual Result** | **Actions Needed/Comments** |
| 1 | Ensure ‘Stack’ class is called through ‘self.zoomInOut’ as the object for ‘Stack’ class into ‘fractal’ class | Zoom out of program using BACKSPACE key | Fractal zooms out | Fractal zooms out and stack works | Successful |
| 2 | Ensure methods are acquiring the correct initialised variables from ‘\_\_init\_\_ method in ‘fractal’ class | Run program and zoom in and out | Program works | Fractal program works | Successful |
| 3 | Ensure ‘callFractal’ object works to call all necessary methods and ‘fractal’ class itself | Run Mandelbrot fractal program | Program works | Fractal program works | Successful |

* 1. **Colour Theme Testing (Screenshots)**

|  |  |  |
| --- | --- | --- |
| **Test No.** | **Test Purpose (Colour Theme)** | **Outcome (Success/Fail)** |
| 1 | Default | - Theme – Success  - Smooth – Success  - Histogram – Success  **(6.2.43.)** |
| 2 | Fire | - Theme – Success  - Smooth – Success  - Histogram – Success  **(6.2.44.)** |
| 3 | Sunset | - Theme – Success  - Smooth – Success  - Histogram – Success  **(6.2.45.)** |
| 4 | Ocean | - Theme – Success  - Smooth – Success  - Histogram – Success  **(6.2.46.)** |
| 5 | Emerald | - Theme – Success  - Smooth – Success  - Histogram – Success  **(6.2.47.)** |

* 1. **Third Party Testing:**

**Tester:** James Heavens

* The zooming is very swift
* Controls are easy to use
* Relatively quick zooming in time according to expected results
* Easy to navigate around menu
* ‘Help’ menu is very useful
* Program is correctly responsive according to what is clicked
* Could speed up zooming in time, not taking account of expected result
* Would be useful to have a palette of colours rather than selecting strict colour themes
* Encryption algorithm is very useful for passwords
* Useful that ‘Guest’ button is added to launch the visualisation tool directly

1. **Evaluation**
   1. **Objectives**
      1. **Core Objectives**
2. The user must be able undo actions by popping off the previously pushed on actions and manipulations in recent usage off a ‘stack’ (abstract data structure). Therefore, the user should be able to re-do actions using a stack by pushing item on stack.

This objective was met with clear purpose and the user is able to zoom out of the image with the use of the Stack data structure by pressing the ‘Backspace’ button on the keyboard to process the zooming out. However, a significant portion of the layout of the calculation had to be changed in order to complete this objectives – successful.

1. The program must allow the user to zoom in and out of the image (with limitations e.g. 10^32) until the image starts to deteriorate as the Mandelbrot set isn’t infinite as the orbit of 0 under the equation doesn’t go to infinity (value of c). The resolution of the image must be maintained with a high level of zooming in to allow the user to explore the image with ease and enjoyment (can only be done within the limit of the Mandelbrot set up to the point of disorientation of the image)

The program has allowed the user to zoom in and out of the image with the Pygame window being used as the GUI and the resolution has been maintained with a large number of zooms. 25 zooms were tested and the same level of resolution was maintained with no deterioration in the image resolution – successful

1. The program must be able to manipulate and use the inputted values correctly no matter how complex or simple they are, to be able to accordingly perform the user’s operations with full effect with the correct calculations e.g. the demanded ‘MaxIterations’ must be as accurate as possible. Input panel must be put into the system to allow the user to input values such as the maximum iterations.

The program has allowed the user to manipulate the Mandelbrot fractal with the given inputs. Very high (1000) and very low (5) maximum iterations values were tested for the output and the fractal was carried out with full effect and no errors with correct calculations for zooming of the image. The input panel allowed the user to input values such as the maximum iterations, colour themes, scaling speed and image dimensions - successful

1. In case the value inputted into any variables is unrealistic due to erroneous limits, the program should output an alert to notify the user of an error and suggest a solution e.g. let the user know the limitations of zooming in, or provide them with a warning message that image may take long to load if unrealistic value is inputted e.g. Maximum Iterations: 10,000

In the editing window alerts the user if a high value has been selected. The value is still processed for experimental purposes for the user, however, they are warned e.g. there is a warning if the maximum iterations value is above 500 and if the image dimensions are large as the processing time may be significantly longer. The high limits are stated in the ‘Help’ menu from the main menu – successful

1. The user interface MUST be simple and convenient to follow with clear options placed in the most appropriate positions as possible to make them easy to follow to please the user and convince them to use this tool again and making the whole interface readable. For example, the options must be easy to follow and the font and colour of the text should be easy to read. The user should be able to manipulate easy-to-use control keys with various controls (such as zooming in) in the complex system using abstraction – swift and easy-to-use user interface.

Every detail in the interface is readable in the login system, RGB slider, Mandelbrot tool, editing window, main menu and the help menu with a clear font and writing. All the buttons, titles and labels have been placed in the most convenient places using Tkinter making the whole interface readable with convenient font and colour. The fractal has easy-to-use control keys for the manipulation of the fractal and all the controls are stated within the help menu. Swift and easy control is enabled e.g. the zoom box moves swiftly in the fractal window – successful

1. The program should be able to provide a ‘help’ tab to open whenever the user needs a reminder of how to use the program or if the user is completely new to the program. This must be easy to find with significance within the user’s visualisation on the screen for easy access. Also can include a glossary of values for brand new users.

A ’Help’ tab is provided in the main menu which can be accessed with easy usage whenever it is needed to help the user. It is easy to find from the main menu. My contact details, glossary and controls are used to give every possible detail in the help menu.

1. The program should be able to output complex information to intrigue the user with knowledge such as the ‘Complex Number: Mouse Position Value’

A form of complex information I have used is the tracking of the mouse using its position value in the form of a complex number on the circadian coordinate system of the fractal which is shown in the top of the window – successful

1. The user must be able to manipulate the image visualisation to a certain extent of complexity such as allowing the colour theme to be changed. A suitable colouring algorithm must be used to colour the pixels rather than a simple greyscale representation of the image e.g. Histogram method or standardised colouring palette for colour themes; RGB values (smooth colouring as extension objective/acceptable limitation)

Different colouring algorithms were successfully carried out such as the Histogram colouring algorithm and colours were plotted and correctly calculated. RGB values were used for the calculation using 2D arrays with colour themes. Smooth colouring also used to a certain extent – successful. The colour theme is editable and an RGB slider is also provided if the user wishes to educate themselves and experiment with RGB values if they want to change the values manually via the source code. Overall, the colour theme is editable via the editing window – successful

1. The manipulation of the image must be swift with a minimisation of ‘lag’ in terms of zooming into different parts of the image (depending on hardware and given inputs)

The lag is minimised to a certain extent depending on how high or complex the values inputted are as it may be dependent on the hardware being used too. Overall, I have kept lag to a minimum using as much optimisation as possible within the given time frame of the NEA – successful

1. The program must be able to load and present the image within a short time in real-time for the user’s satisfaction. If the input values are changed (such as the MaxIterations or ImageSize) it must change within a given time frame of 0 – 150 seconds (150 seconds being the absolute maximum with REALISTIC values – most computers can’t process maximum iteration values of unrealistic values of 5000; processing time may vary with hardware).

The fractal image is loaded within a suitable time frame according to the input values and hardware being used. Many tests were conducted and on average from all tests, this objective has been met and zooms are loaded In a short time in real time – successful

1. The program must be able to convert the pixels to complex numbers in order for the circadian plane to process the pixels.

The algorithm has been correctly carried out and calculated for the process of complex numbers on the circadian plane – successful

1. Provide a method within the code to present the image in the user interface with the use of manipulation of the image – making it the most significant part of the interface. (e.g. Using the ‘matplotlib’ library which specialises in plotting and presenting images. This will be decided after the prototype investigations) – most likely to use Pygame.

After numerous prototype tests in my ‘Analysis’ I balanced the limitations and benefits of each interface method and I used ‘PyGame’ as my final decision for the manipulation of the fractal image. I tried matplotlib and PIL as they are very efficient in terms of timing and optimisation when rendering the Mandelbrot set. However, the major limitation was that it was very inefficient in terms of programming an interface allowing the user to zoom In and out through Matplotlib and PIL. – successful

1. Allow the user to log into the system, and if they have not got login details then they should be able to sign up.

The user can log into the system and can also sign up – successful

1. If the user wishes to enter as a guest, this must be allowed when they don’t sign up or login.

The user has the option of accessing the editing window and therefore the fractal, through the simple ‘Guest’ button via the main menu – successful

1. User must be able to quit the program instantly when required.

The user can quit the program through any window or close any window – successful

1. User must be able to change the scaling speed of the rectangular zoom box (recommended value: 0.001)

The user is able to change the scaling speed of the rectangular zoom box via the editing window with the default value being 0.0001

1. Create appropriate working slider for RGB values in case user wants to change RGB colour values of fractal manually through source code.

Working RGB slider is created and can be accessed via main menu – successful

1. User must be able to change image size (dimensions) from drop down meu of industry standard image dimensions.

The user can change the image dimensions via a drop down menu in the editing window. I could’ve potentially allowed the user to create their own image dimensions for further flexibility in manipulation of the settings. – successful

1. The user must be allowed to save the fractal image in runtime

The user can save the fractal image in runtime by pressing ‘s’ on their keyboard. However, I could’ve allowed the user to save the image to their chosen file directory rather than the default settings.

* + 1. **Acceptable Limitations/Extension Objectives**

1. There will inevitably always be a limitation in zooming into the image as there are boundaries selected such as the maximum iterations and image size; my program may not be able to zoom in as further in as the other programs I have investigated earlier on in my analysis – but I will make it acceptable for the purpose of the program and maximise flexibility of zooming in as far as possible.

With many tests carried out, the fractal can zoom in further than I initially predicted despite this limitation stated in my analysis. However, it can become quite slow when loading the newly zoomed image after a significant amount of zooms e.g. 25 zooms. Overall, I have overcame this limitation to a certain extent. To improve this, I would try to improve how fast the newly zoomed fractal images are loaded when the fractal has been zoomed in a significant amount of times to improve the overall speed and optimisation of the program.

1. Disallowing certain numbers when put into the system and an error message being outputted when these numbers are being inputted. I will make the user aware of what sort of numbers can be input e.g. complex numbers and what numbers can’t be input e.g. logarithmic functions

Limitations and input/exception handling has been used with the prompt of error messages when unsuitable inputs are used in entry boxes throughout every aspect of my program. In the ‘Help’ menu I have made it clear which types of inputs will be allowed. However, calculations can not be used in the input boxes such as logarithmic functions and this is the limitation.

1. There are certain colouring algorithms which may be better and more appealing in terms of viewing the Mandelbrot set e.g. continuous smooth colouring. The bands of colours will be rendered with higher quality and replaced with a smoother gradient with the use of logarithmic functions, as opposed to the Histogram colouring algorithm (algorithm I am most likely to use). However, it may be far too complex to implement within the time frame given with the combination of other objectives I am trying to achieve e.g. minimising time delay and processing of printing the image out on the screen.

I have overcame with limitation and met the extension objective of implementing the ‘smooth colouring’ algorithm to a certain extent. The algorithm is correctly carried out in my program. However, the render of certain bands of colours on the outer edge of the fractal is not completely successful and you can see subtle, vague bands of colours which have not been smoothly rendered. However, the algorithm has still been carried out correctly and next time I would improve the optimisation of the overall algorithm to make sure the bands of colours are rendered correctly to become ‘smooth.’

1. My project is specifically targeted at the Mandelbrot set of fractals, but some other systems allow the use of other fractal e.g. Julia set fractals, snowflakes, Lyapunov fractals. This may be a limitation as in my investigation I discovered that other projects have implemented both the Mandelbrot set with the Julia set and a wide range of other fractals. Implementing the Julia set fractal (closely associated with Mandelbrot)

According to my critical path analysis of the ‘Gantt’ chart I used in my analysis, I ended up having a few weeks of spare time and worked on the Julia set. I have implemented a preview of the Julia set as one of the extension objectives met. However, next time I would allow the user to zoom into the Julia set fractal and manipulate the visualisation and calculation inputs as I have done with the Mandelbrot set. I would also place the Mandelbrot set fractal and Julia set fractal next to each other in the GUI to allow the user to experiment and compare the 2 fractals. Overall, I met the extension to a certain extent and managed to carry out the Julia set fractal despite having minimal knowledge on it as seen in my analysis of my primary and only research on the Mandelbrot set. Next time I would also use more fractal sets for the experimental usage of the user e.g. Snowflake, Lypaunov fractals etc.

1. One well known limitation of implementing Mandelbrot set is the limitation in drawing external rays. This is a limitation in which the drawing tool for the set cannot be used to carry on. The boundary point is implemented by the resolution of the data type of floating-point computation used in the code. This is analysed via the floating-point computation with quadruple 128 bits. This has become a limitation because computer systems usually use 64-bit floating point numbers (double data type) and the program won’t let you zoom into certain areas which require 128 bit floating point communication, such as zooming into specific rectangles.

This limitation/extension objective was near to impossible to meet as a ‘high spec’ device is needed to carry out drawing external rays and using this high level of resolution where 128 bit floating point communication si needed to zoom into specific rectangles. I enabled standard zooming in and even with good levels of optimisation, however, this objective/limitation was unrealistic but I will keep this in mind next time and perhaps one day carry it out on a highly optimised and ‘high spec’ device when potentially programming in University or later on in life to further my research of the Mandelbrot set fractal.

1. Implement an encryption algorithm for the login/registry system to encrypt the password when registering and decrypting when logging in with the ‘key.’

I met this extension objective using the Vernam cipher which I learned earlier in the computer science Syllabus. I implemented through file organisation and accessing files through encrypting and decrypting the plain text to the ciphertext using the one time pad. When the user registers, a one time pad is generated using the ‘random’ library (it would be difficult to generate a truly random one time pad in Python and this is a possible limitation which could be further improved by using white noise as one example of a truly random format of the one time pad). The encryption algorithm is used with the ASCII equivalent of the characters and this algorithm is therefore reversed to decrypt the password for the user to log in to the system.

* 1. **Final Third Party Feedback/Analysis**

James said the user interface was excellent and easy to follow in terms of the controls, instructions and what to do overall. He said it was very fun to use and he would use it again when needed. In my opinion, this is considered to be positive feedback because the Mandelbrot Fractal program is a ‘tool’ meaning it must be used numerous times. The positive feedback from James tells me that users would like to reuse it which achieves my main objective of my program being used as a tool. James is a Maths student at my college and the overall mission of this tool is to allow Mathematicians to use this as an experimental tool to explore the Mandelbrot fractal set and it’s overall purpose.

Additionally, James praised my efforts of the user interface being easy to use which shows that my main core objective of making the program easy to use in terms of controls has been achieved to a high standard as James said the ease of usage was excellent. I feel like this is very good for my program if it has to be used later on because it allows the user to have an enjoyable experience despite the source code and mathematical algorithms being potentially difficult for a new user to understand…the interface allows them to have a fun and easy experience through the GUI hiding the complexities of the source code.

James stated that the overall visualisation of the fractal is very pleasing and it is nice for a new user even if they don’t understand Mathematics because it is a nice visualisation tool despite not being able to understand the Mathematical concepts behind the program. He also liked the fact that I added a login system if the user wishes to have the satisfaction of having their own ownership of the program and he stated it’s very useful how I added a ‘Guest’ option in case the user does not wish to go through a sign up process and the ‘Help’ tab helped James a lot because he was not familiar with some of the terminology used such as ‘Maximum Iterations.’ It was also very easy to use the input entry boxes due to the exception handling used with only being allowed to use numbers (integers and floating point numbers)

Overall, James said I have achieved all of my measurable core objectives. However, he has stated some improvements that can be made with some constructive criticism.

Firstly, James stated that the RGB slider should be linked so that the user can change the individual colours of the fractal such as the ‘edge’ and ‘fill’ colours manually through the RGB slider. In addition, James said it’s excellent that I included the Julia set fractal but I should’ve included the same functions with the Julia set as I did with the Mandelbrot set such as zooming in and changing the colour themes with different input values being used too. Furthermore, James stated that I should’ve used more effective optimisation algorithms so that the zooming of the fractal is slightly more smoother and so that the zooming in and loading of the fractal is sped up slightly because it can be time consuming for the user waiting for the new zoom of the fractal in certain situations. Also, James said that the rectangular zoom box should be able to be changed in order to transform into different shapes rather than just a rectangle to allow some further flexibility for the user. Interestingly, James said that I should’ve allowed the user to input mathematical functions such as logarithms, pi etc… to allow users that are more familiar with Maths to have a more experimental experience with the program rather than the simple floating point and integer numbers. Finally, James said that along with the up and down arrow keys used to change the size of the zoom box, It would be very nice if the user could change the size of the rectangular zoom box using the manual usage of the mouse for a more flexible panning of the zoom area.

In addition, I allowed all the interviewee’s (the 4 anonymous interviewees and James) in my analysis to test my program and fill out a survey, the average scores out of 5 are as follows:

* Visualisation: 4/5
* How easy it is to use: 5/5
* Optimisation performance in run time: 4/5
* Extent to how manipulative the fractal is/Enough features? : 3/5
* Performance of the fundamental features: 5/5
* Performance of the extension features: 3/5

From the scores above I have gathered the following:

* The overall visualisation is up to standard and could be improved through perhaps letting the user have more control in the colours they choose, such as the fill and line colours of the fractal rather than just the fractal themes. One of my interviewees said the resolution and quality of the fractal did exceed their expectations
* It is very easy to use and straight forward, especially with t he help menu
* The performance in run time is good, however, I feel like this can be based on the processor power as I investigated further the program does take the majority of power of the processor in runtime. This can be difficult for certain hardware to cope with. I will take this into account next time and perhaps investigate further into more optimisation algorithms to maintain a high resolution as well as speeding up the load time
* From the score of my program having enough features, I feel like I should’ve included more such as having more fractal sets and even broadening the features of the Julia set which I completed as an extension objective. I could also potentially set up an online system where users can upload their fractal images.
* The performance of the fundamental features is very good as seen in the score. The fundamental features include the ability to zoom in, zoom out and other features such as registering and logging in with an account.
* The performance of the extension features could’ve been extended further such as allowing the Julia set to be manipulated as much as the Mandelbrot set. In addition, for the Vernam cipher I could include a TRULY random key.
  1. **Possible Improvements/Extensions**

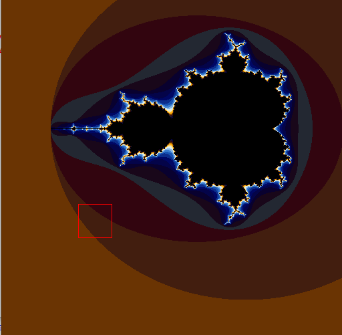
1. Include more fractal sets and place them next to the Mandelbrot set in the interface for easy comparison. Similarly, the Julia set extension objective should be broadened further by allowing the user to zoom in and use the same input functions as the Mandelbrot set.
2. Allow the user to manually change the colour of the fractal such as changing individual fill and edge colours rather than restricting the colour change to strict colour themes. This would allow the user to have more visual flexibility in how the fractal looks in the interface. I would do this by enabling more input boxes for the change in the RGB values of each, individual band of colour in the fractal. This could also be done using a palette of colours.
3. Perhaps include a sort of networking feature to send a confirmation email to the user through Flask or another API when using the login system. I have already experimented with this, however, it did not go as planned and I had to leave it due to the limited time frame I had on my Gantt chart.
4. Storing the username, password and other fractal details in a database with a hashing algorithm for encryption. This would allow the user to resume the fractal from the last zoom they left it at by using the last stored current boundaries used in the zoom of the fractal. This would make effective use of SQL and the queries made. (However, this may not be necessary for a fractal tool). The use of databases would’ve saved storage memory rather than deleting and creating files each time it needs to be accessed throughout the program. A normalised database would’ve been great for data integrity. Next time I design my program to save data (for example, in a login system), I will use a database to improve the efficiency of searching and querying different data items needed.
5. Allow the user to change the shape of the rectangular zoom box for a more flexible zooming feature. In addition, the user must be able to pan around with the zoom rather than restricting their usage to up and down arrow keys and instead allow them to manually use their mouse for more flexibility in the zooming boundaries. This could be done by using PyGame functions to allow the user to pan around the fractal as well as changing the boundaries of the zoom box in the source code to allow it to become different shapes rather than just a simple rectangle.
6. Allow the user to input complex mathematical functions in the entry boxes for users who are more familiar with Maths and therefore being able to take a more experimental approach with the tool. This could be done by using the ‘Maths’ library in Python or even using my own knowledge from A Level Mathematics to form algorithms to allow users to input logarithmic functions, pi, Euler’s number etc…
7. Most importantly, I should improve the optimisation of the program although this extension is very controversial due to it being dependent on the complexity of the input values entered by the user and the hardware being used to run the program. But, I can still improve the optimisation to a certain extent by including algorithms to increase the speed of loading up the newly zoomed fractal image. After doing further research on this extension objective, I have discovered that this may be a limitation on Python due to its slow loop times compared to other programming languages such as Java.
8. Additionally to my last extension, I could potentially program the Mandelbrot fractal set on another programming language such as Java where the speed and optimisation of the zooming of the fractal isn’t limited in terms of loops within the source code. This would improve the overall experience in the interface for the user.
9. I should have also included the progression of how much % of the fractal the program has generated so far in run time. For example, if the fractal is half way done processing it would print 50% in the top of the fractal window. This would be done with a % loading algorithm by keeping track of how many pixels have been plotted so far in run time. I carried this out in my prototype testing, however, It did not make it in my final program and this would be included as an extension objective if I were to program the Mandelbrot fractal set visualisation tool again.
10. In the Vernam cipher, a major improvement would be generating a truly random one time pad. This was difficult in Python and I used the ‘random’ library from the choice of any character in the ASCII table. From further research I have discovered that I could use white noise, radioactive decay or the timing of a hard disk read/write to offer perfect security which is the true purpose of the Vernam cipher. I would also need to further justify and use the Boolean XOR in my implementation of the Vernam cipher.
    1. **Obstacles Overcome**

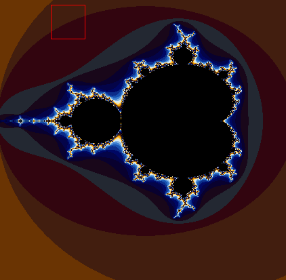
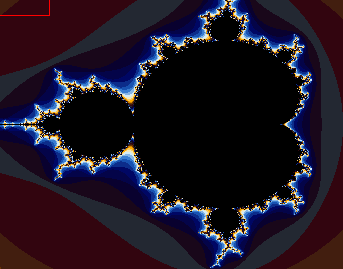
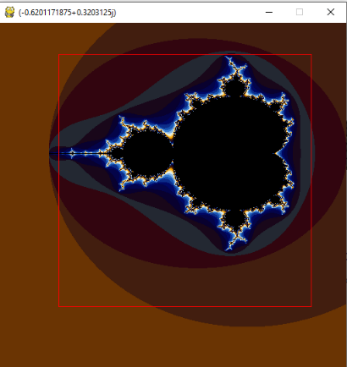
The uncertainty of COVID-19 was a major obstacle as there was an uncertainty of deadlines and I still maintained effective usage of the critical path analysis of the Gantt chart I used which helped me organise my time management. I used any extra time to organise my documentation and work further towards extension objectives and overcome the acceptable limitations stated in my analysis. Additionally, the stack implementation took more time than expected when tracking my Gantt chart. However, I overcame this and used the stack to its full effectiveness despite the challenges I faced such as the boundaries of the Mandelbrot set being calculated incorrectly when trying to implement the stack; I overcame this problem and my program works perfectly now. One obstacle which limited my implementation was the use of hashing. The idea of using hash tables and using the actual hashing algorithm would’ve in fact made my login/registry system less effective due to the weakness in my program that I used file organisation rather than databases. However, I overcame this problem using the ‘Vernam’ cipher with the goal of implementing an encryption algorithm for the password in the login/registry system. Furthermore, I also faced the obstacle of not having very hardware when implementing my program. My program consists of many loops in Python and it takes up a lot of the processing power which I discovered when investigating the actual hardware usage in the ‘usage’ section under the ‘settings’ of my laptop. I maintained focus although it took a significant amount of time running the program on each run of it. I overcame this by implementing optimisation algorithms to a certain extent, however, this was still a limitation when programming.

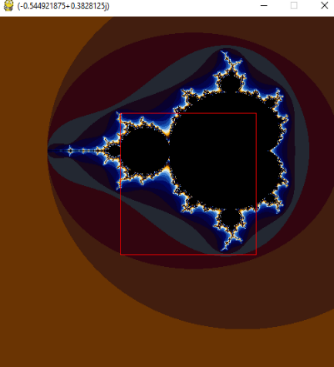
* 1. **What I have learned**

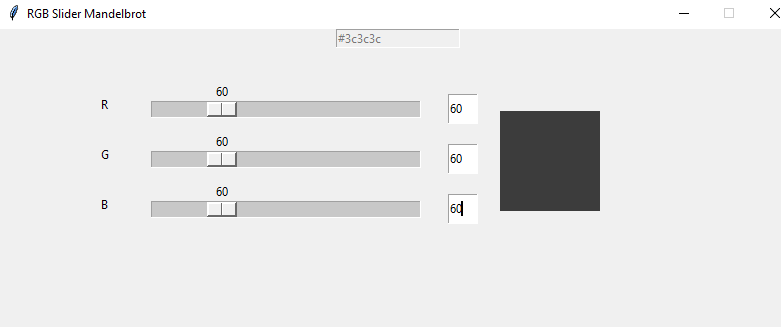
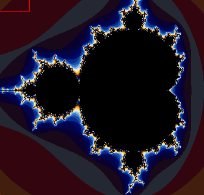
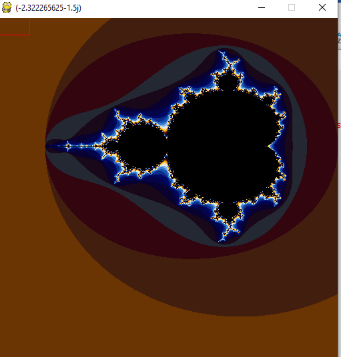
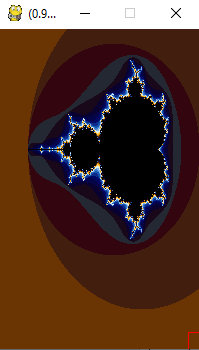
Mainly, I learned how to implement a fully working GUI rather than a standard CLI approach in Python. I did this through spending a significant amount of hours learning Python. A very big limitation when approaching the implementation I had was the fact that I had limited knowledge on OOP at the time. However, I overcame this by doing lots of research and practice on prototype and unrelated programming in Python to practice OOP, such as using inheritance to inherit the attributes and methods of another class which is shown in the login and registry system implementation of my program. I also learned about the Mandelbrot set which I had no prior knowledge of at all when approaching the NEA, but completing the ‘Analysis’ section really helped. I also learned how to implement lots of mathematical algorithms in the implementation which was really enjoyable as I am passionate about Maths and the opportunity to complete an NEA on a mathematical-related algorithm of the Mandelbrot set was enjoyable despite obstacles faced. I also learned how to maintain focus despite the pandemic, this was mainly through the Gantt chart I created in my analysis which allowed to track the timings of different parts of my project. This allowed me to have time to complete extension objectives as I had 1-2 weeks spare towards the end of the implementation section. In terms of programming, I learned lots of new skills such as using ‘lambda’ and these skills can be transferred over to university which I hope to study Computer Science at.

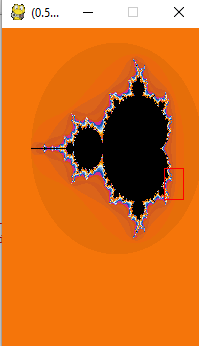
1. **Appendices**
   1. **Site References:**
2. <https://www.encyclopedia.com/science/encyclopedias-almanacs-transcripts-and-maps/fractal-theory-and-benoit-mandelbrot>
3. <https://plus.maths.org/content/what-mandelbrot-set#:~:text=In%20mathematics%20this%20process%20is,c%20is%20a%20constant%20number.&text=x1%20%3D%20x02%20%2B%20c.>
4. <https://brilliant.org/wiki/complex-numbers/>)
5. <http://warp.povusers.org/Mandelbrot/>
6. <https://tomchaplin.github.io/portfolio/Exploring-the-Mandelbrot-Set/>
7. <https://stackoverflow.com/questions/53658296/mandelbrot-set-color-spectrum-suggestions>
8. <https://tomchaplin.github.io/portfolio/Exploring-the-Mandelbrot-Set/>
9. <http://catilinejs.com/website/leaflet-fractal/#1000/-0.37/0.6/mandlebrot>
10. <http://catilinejs.com/website/leaflet-fractal/#1000/-0.37/0.6/julia>
11. <https://www.bbc.co.uk/news/magazine-11564766>
12. <http://math.hws.edu/eck/js/mandelbrot/MB.html>
13. <http://davidbau.com/mandelbrot/>
14. <https://sciencedemos.org.uk/index.php>
15. <https://www.ibiblio.org/e-notes/html5/fractals/mandelbrot.htm>l
16. <http://www.jakebakermaths.org.uk/maths/mandelbrot/canvasmandelbrotv12.html>
17. <https://en.wikipedia.org/wiki/Plotting_algorithms_for_the_Mandelbrot_set>
    1. **Testing Screenshots**



* + 1. 
    2. 
    3. 
    4. **(Joined with 2.2.4. where zoom box size has changed)**



* + 1. 
    2. 
    3. **(512x512)**
    4. **(320x200) – Follow from 2.2.8.**



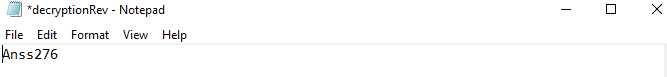
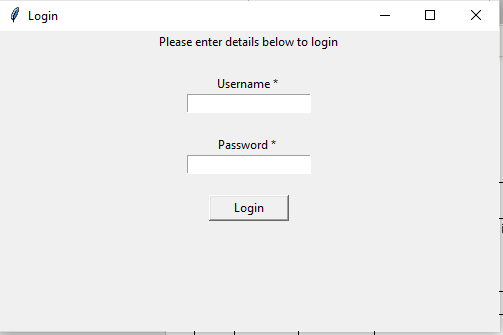
* + 1. **oneTimePad.txt**

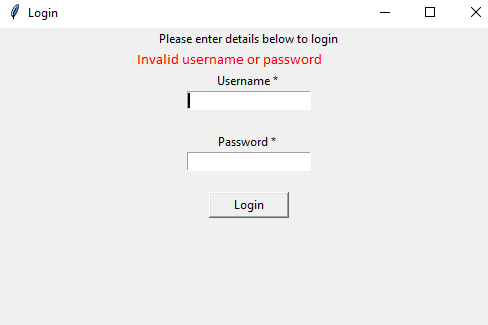


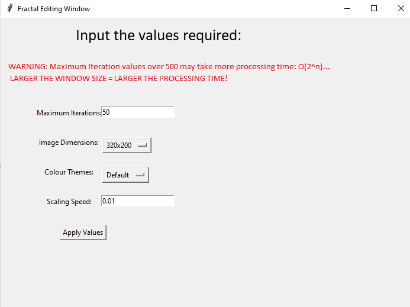
* + 1. **ciphertext.txt**
    2. **vernamPass.txt (plaintext)**

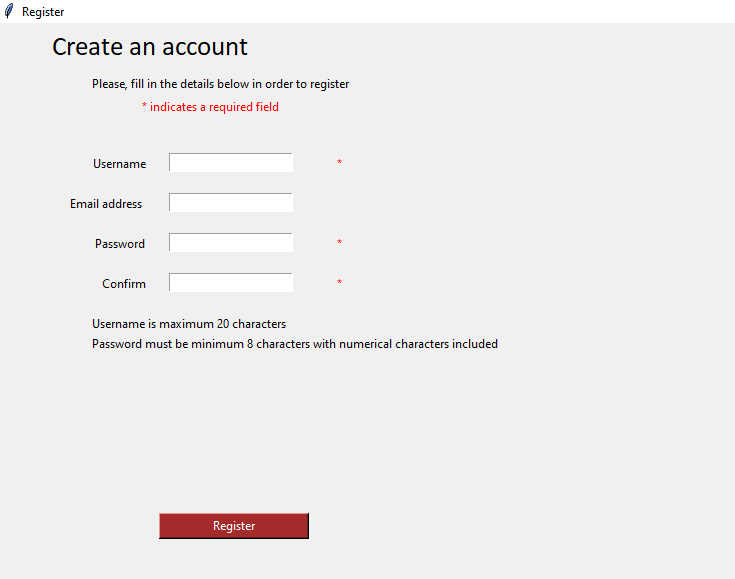


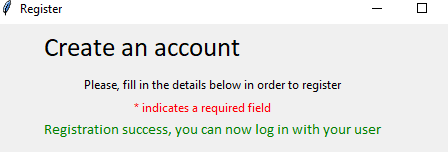
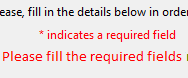
* + 1. **decryptionRev.txt (decrypted password)**

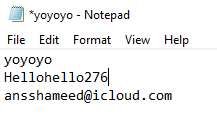


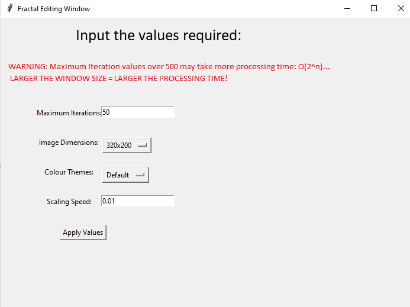


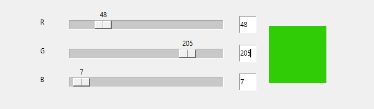
* + 1. 

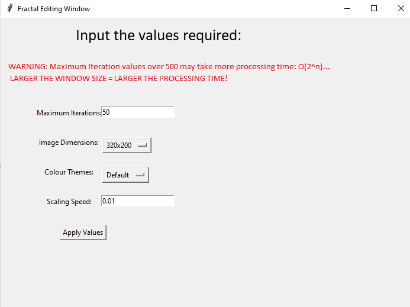


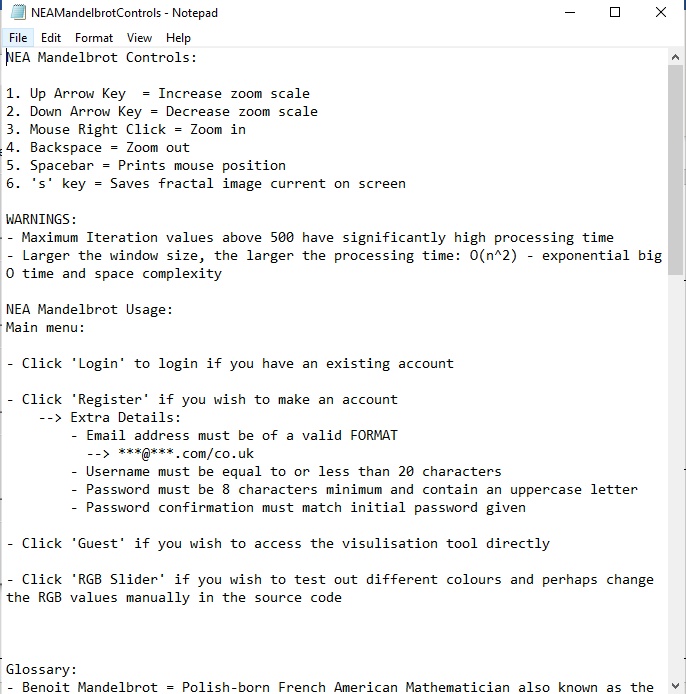
* + 1. 
    2. 

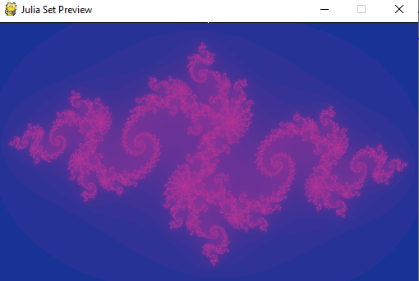


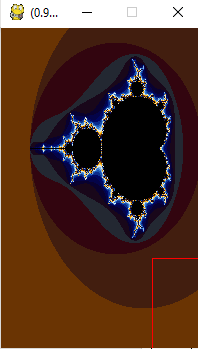




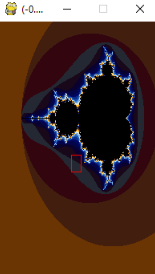
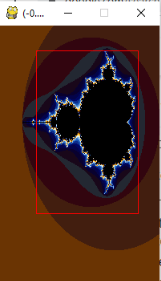


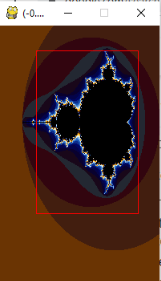
* + 1. 

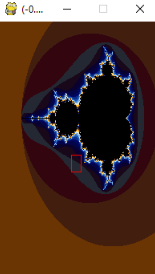


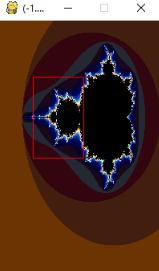


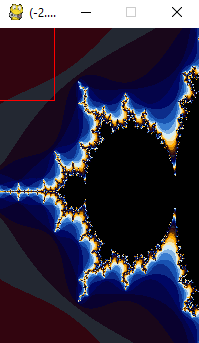


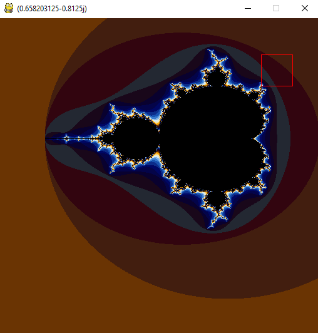
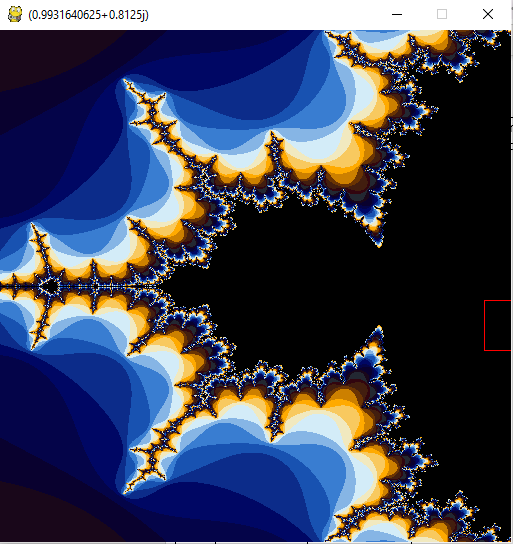
* + 1. 
    2. **– Increase in zoom box to 6.2.28.**



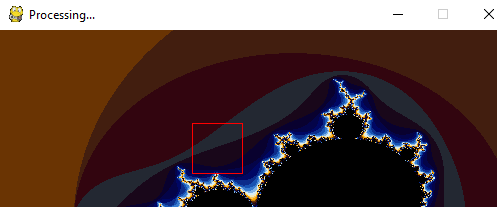
* + 1. **Decrease in zoom box from (6.2.30.)**

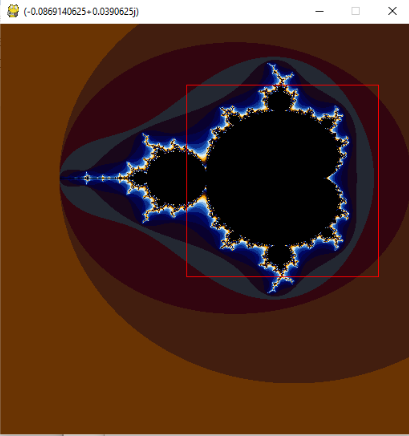


* + 1. **– Zoomed into dimensions of box in 6.2.32.**

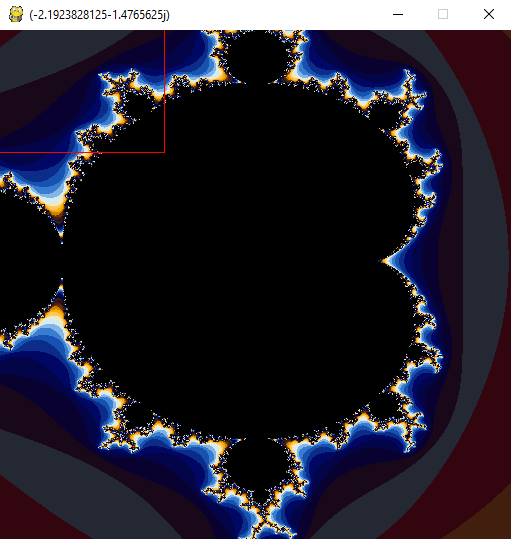
* + 1. **– Before zooming in**
    2. **– Zoomed in**
    3. **– Zoomed back out using stack**

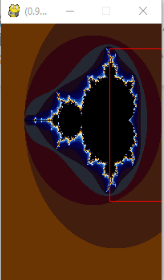
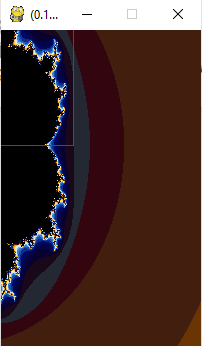


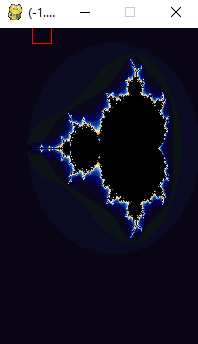
* + 1. 

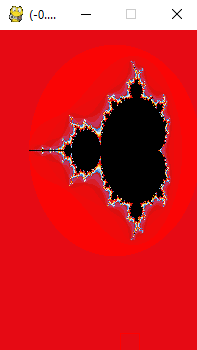
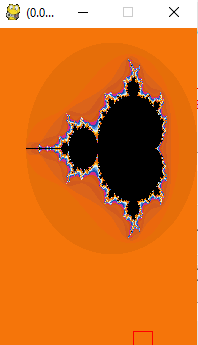
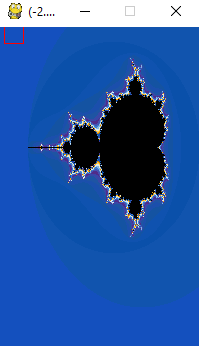


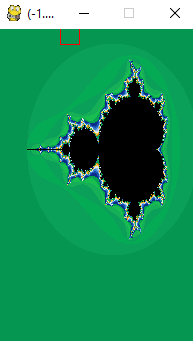
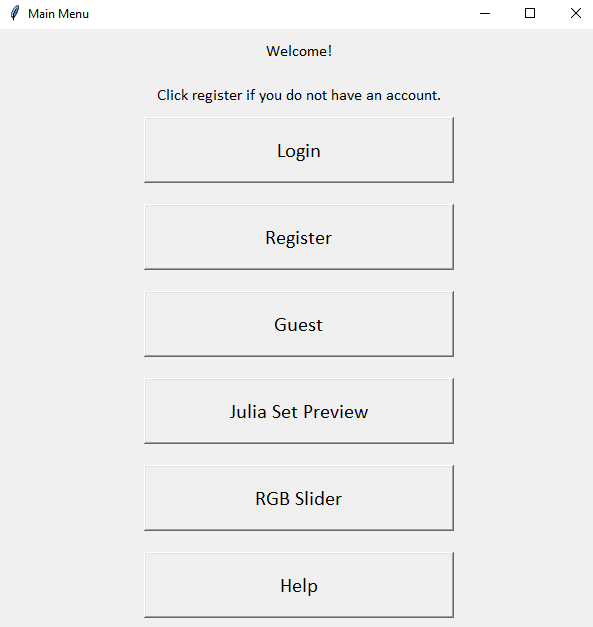
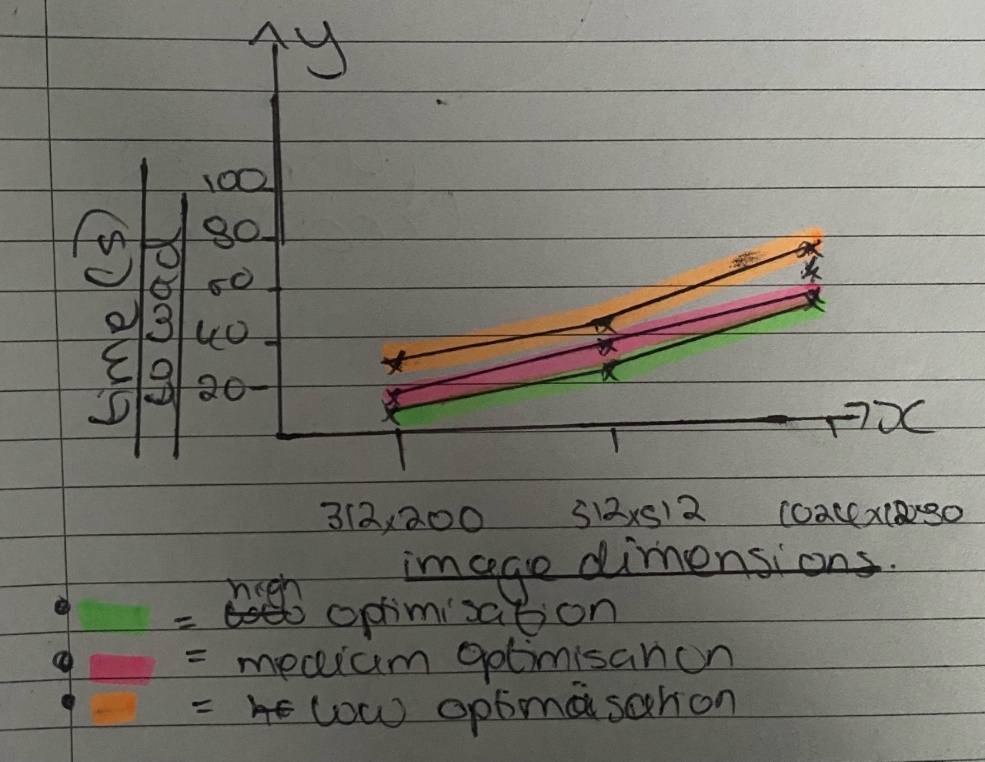
* + 1. **Zoomed in of 6.2.38.**

****

* + 1. 
    2.  **Boundary data check of zoom box**
    3. **Zoomed in boundary check to 6.2.41.**



* + 1. **Default Colour**
    2. **Fire**
    3.  **Sunset**
    4. **Ocean**

* + 1.  **Emerald**
    2. 
    3. 
  1. **Testing Video**
* This video was recorded on the slowest of my testing hardware devices for testing the absolute minimal optimisation (Heuristic) of the fractal set.
* Link: <https://youtu.be/mJ29v8bdKQ8>