OCR Computer Science H446

NEA Submission May 2023

Ace Harvey

Centre No 20153

Candidate No idk

## Project Overview

My project is a piece of software designed to visualise complicated mathematical systems, such as the Lorenz attractor, and other similarly chaotic and interesting systems. It will take an input “project file” which contains information such as a system of differential equations, viewport settings and color settings and produce output images that show the phase space of that system. These project files can be generated by the program through use of a GUI made in Python.

Contents

[Project Overview 1](#_Toc116546625)

[Analysis – Problem Identification 4](#_Toc116546626)

[Analysis – Stakeholders 5](#_Toc116546627)

[Identifying Stakeholders 5](#_Toc116546628)

[Survey Questions 6](#_Toc116546629)

[Response: Stakeholder Mark 6](#_Toc116546630)

[Response: Stakeholder Isabelle 6](#_Toc116546631)

[Analysis of responses 6](#_Toc116546632)

[Analysis – Computational Methods 7](#_Toc116546633)

[Abstraction 7](#_Toc116546634)

[Decomposition 7](#_Toc116546635)

[Logical nature 7](#_Toc116546636)

[Procedural nature 7](#_Toc116546637)

[Input and output 7](#_Toc116546638)

[Saving configurations and outputs 7](#_Toc116546639)

[Why is this project suited to a computational approach? 8](#_Toc116546640)

[Analysis – Existing Solutions 9](#_Toc116546641)

[*Visions of Chaos* 9](#_Toc116546642)

[Summary 9](#_Toc116546643)

[Features 10](#_Toc116546644)

[Technological Analysis 10](#_Toc116546645)

[*Chaoscope* 11](#_Toc116546646)

[Summary 11](#_Toc116546647)

[Features 11](#_Toc116546648)

[Technological Analysis 12](#_Toc116546649)

[*glChAoS.P* 13](#_Toc116546650)

[Summary 13](#_Toc116546651)

[Features 14](#_Toc116546652)

[Technological Analysis 14](#_Toc116546653)

[Analysis - Features 15](#_Toc116546654)

[Analysis - Limitations 16](#_Toc116546655)

[Dimensionality 16](#_Toc116546656)

[Efficiency 16](#_Toc116546657)

[Mathematical 16](#_Toc116546658)

[Rendering 16](#_Toc116546659)

[Analysis - Requirements 17](#_Toc116546660)

[Stakeholder Requirements 17](#_Toc116546661)

[Functionality Requirements 17](#_Toc116546662)

[Hardware and Software Requirements 17](#_Toc116546663)

[Analysis – Success Criteria 18](#_Toc116546664)

[Design - Decomposition 21](#_Toc116546665)

[Design – Structure 22](#_Toc116546666)

[Design – Algorithms 23](#_Toc116546667)

[Design – Features 24](#_Toc116546668)

[Design – Key Things 25](#_Toc116546669)

[Design – Testing Data 26](#_Toc116546670)

[Design – Further Data 27](#_Toc116546671)

# Analysis – Problem Identification

Currently, no software exists to visualise any given system of equations relevant phase space. There exists much software made to visual phase spaces of common, known chaotic or interesting systems, however these pieces of software mostly only function in 3D and do not have support for entering custom, user defined equations. Entering custom equations if important as it may help to discover more strange or mathematically interesting attractors and help further the field of chaos theory and gain a better understanding of chaos and phase spaces.

A computer would need standard periphery to visualise these phase spaces, and this is suited to a computational approach as interacting with and creating images is much easier using a computational approach.

Such software is needed as the field of chaos theory is hard-to-access and is also very new in mathematics. Allowing people to experience this though interesting or soothing visuals could potentially help the field grow and progress, as well as help existing researchers visualise new chaotic attractors without needed knowledge of complicating programming and rendering. Furthermore, the interesting visuals can be used for their calming and smoothing effects by end users who are less interested in the mathematics.

# Analysis – Stakeholders

## Identifying Stakeholders

The stakeholders for the project are either people interested in the mathematics of chaos, looking to visualise and/or analyse chaotic maps, or people that want to use the software for the soothing visuals often created by similar software. I would like the software to meet the requirements of both stakeholders.

The software should have inbuilt methods for mathematical analysis, such as searching through parameters and XY phase-space to find attractors and potentially identify their nature. The software should also have the ability to input custom systems of equations so that new attractors can be discovered and analysed. As such, the software would be useful to the first kind of stakeholder as they would be able to use it for mathematical visualisation and analysis. Also, batch and video rendering could be useful to analyse how a system evolves as parameters change or over time.

The software should also be able to render maps with custom colouring and high-quality rendering, potentially in real-time so that users looking to use the software for the soothing nature of the generated images and videos It should also include built-in examples and an easy-to-understand interface without the need of complicated mathematics to use at basic level.

I will survey both types of stakeholders to gain more of an insight into what those stakeholders individually want out of the software, and how it can be more suited to their needs.

## Survey Questions

1. What would you use the software described for, and why?
2. What existing solutions have you heard of / used in the past?
3. If you have used existing solutions, what did you like/dislike about those solutions?
4. How often have you used said existing solutions?
5. Are there any specific features you would like to be implemented?

## Response: Stakeholder Mark

1. **What would you use the software described for, and why?**
   1. I would use the software for visualisation purposes, as existing software doesn’t allow me to visualise systems that aren’t pre-loaded into the software. I would also use it to search through and analyse the phase space, looking for attractors to classify.
2. **What existing solutions have you heard of / used in the past?**
   1. I have used glChAoS.P and Chaoscope.
3. **If you have used existing solutions, what did you like/dislike about those solutions?**
   1. I really liked the user interface on glChAoS.P but it didn’t allow me to add my own systems or render out videos as a parameter is changed. I also felt the same about Chaoscope, but that had a lot of other features I liked such as batch rendering, and the search window which helped find interesting parameter combinations and points in phase space.
4. **How often have you used said existing solutions?**
   1. I did not often use the existing solutions because of the aforementioned issues.
5. **Are there any specific features you would like to be implemented?**
   1. Aside from what I mentioned earlier, I’d like to be able be able to use the program on Linux because that’s my main operating system.

## Response: Stakeholder Isabelle

1. **What would you use the software described for, and why?**
   1. I would use the software in a therapeutic way to make cool visuals in order to relax.
2. **What existing solutions have you heard of / used in the past?**
   1. I haven’t heard of anything exactly like it, but the closest thing I can think of is like blender or Photoshop, something related to 3d modelling.
3. **If you have used existing solutions, what did you like/dislike about those solutions?**
   1. N/A
4. **How often have you used said existing solutions?**
   1. N/A
5. **Are there any specific features you would like to be implemented?**
   1. I would like to be able to change the color scheme. I would also like the interface to be easy to use, maybe with a little information button telling me what each slider does.

## Analysis of responses

From these responses, I deduced that the creation of the project is justified as Mark had issues with existing solutions and Isabelle was unaware of similar existing solutions. The project meets the needs of both Mark and Isabelle I also deduced that the following features will be important to implement:

* Render a video as a parameter is gradually changed
* Batch rendering
* Search window
* Linux Compatibility
* Ability to change color scheme
* Tooltips or similar on the user interface

# Analysis – Computational Methods

## Abstraction

The project should take a config file as an input or take input from a UI to change said config file, then output a picture or video in the format selected by the user. However, processing of data into images is a complicated problem, and one that has been solved by many before me. As such, for some elements of output processing (mostly pertaining to encoding raw pixels into filetypes) I will be using existing libraries such as PIL (Python Image Library) and openCV for video encoding. This usage of existing libraries is abstraction, a computational method.

Furthermore, I could use abstraction to hide away the complicated workings of the project, only showing the user the parameters that affect the look of the final image rather than all of the parameters for image rendering which, when adjusted, could cause unexpected results.

## Decomposition

The project decomposes into various tasks easily, such as dealing with the mathematical expressions, the image processing and rendering, etc. By using decomposition in this way, I can break down the problem into less complicated problems and solve those individually, which would be easier than tackling the whole problem at once.

## Logical nature

The project requires the use of logic in many distinct aspects – such as validating mathematical expressions, evaluating said mathematical expressions and validating configuration files.

## Procedural nature

The project is also highly suited to procedural programming techniques, as the program can be decomposed into smaller programs and then recombined at the end. Doing so would also allow me to reuse various parts of the project in other parts of the project, or potentially in future projects. Building smaller subprograms and combining them allows for a smoother workflow and also allows me to isolate problems or bottlenecks easily.

## Input and output

Presenting output is a problem that leads itself to the use of computational methods as the simplest way to present an output diagram is through an image or video on a computer screen – attempting to do otherwise would be complicated, requiring some form of physical image creation which could be expensive due to the need for physical materials and would be inconvenient if the user made a mistake in input parameters and had to re-create the image again, costing double. As such, a computational approach is best to present the output.

Furthermore, getting input to the project would be difficult without a computational approach as I would like to present the parameters in an easy to use and understand way. This leads itself to using a computational approach as I could easily create a graphical or command line interface that labels and explains various parameters and allows the easy entry of numbers or equations through use of a mouse and/or keyboard.

## Saving configurations and outputs

In addition, the project should be able to save and load inputs via the use of configuration files, which is easy to implement via computational methods such as file handling (such as pythons `open` syntax) but would be difficult to do without a computer. Using a computer avoids the need to either remember the desired configuration or have a physical way of storing or inputting parameters.

Also, the project should be able to save its outputs, which again would be easy to do via file handling on a computer. Other approaches could be inconvenient for the user, for example, saving images using a computer would allow the user to then analyse or process them further using other computer software, in a way that would be difficult if not impossible should a computational approach not be used for saving outputs.

## Why is this project suited to a computational approach?

As I have detailed, there are many reasons I believe that make this problem solvable via use of a computational approach; input, output and processing would all be challenging to implement without use of computational methods, and as such I believe that the whole project should be done via a computational approach. Furthermore, the usage of libraries to process raw image and video data would not be possible without the use of a computer, and as such the project is highly suited to a computational approach.

# Analysis – Existing Solutions

## *Visions of Chaos*

### Summary

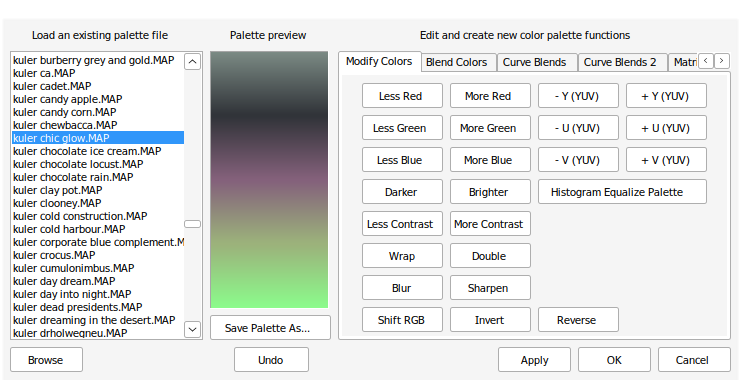
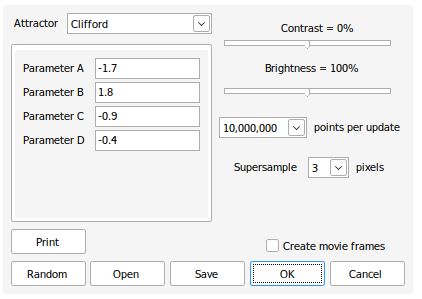
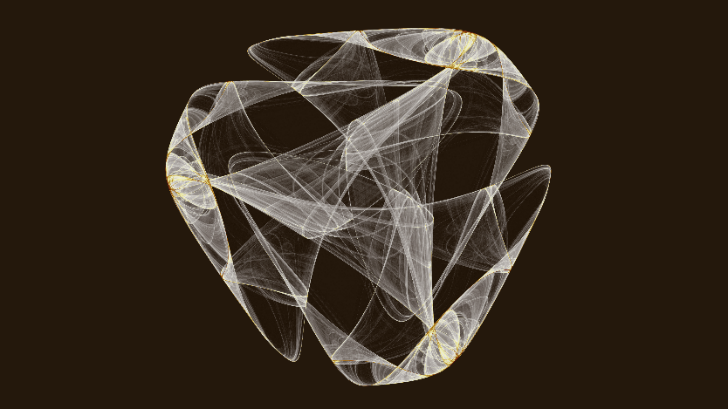
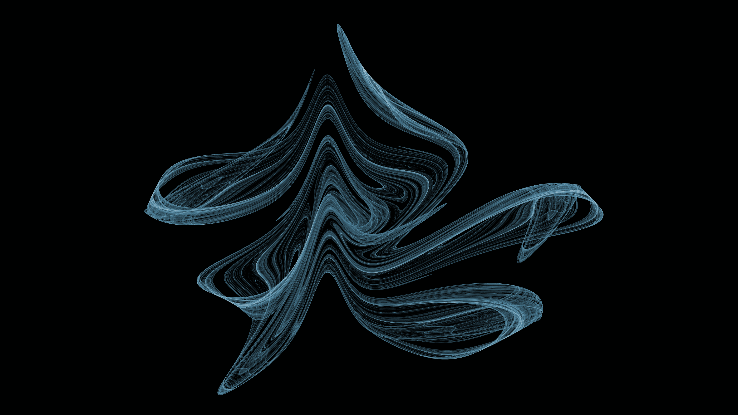
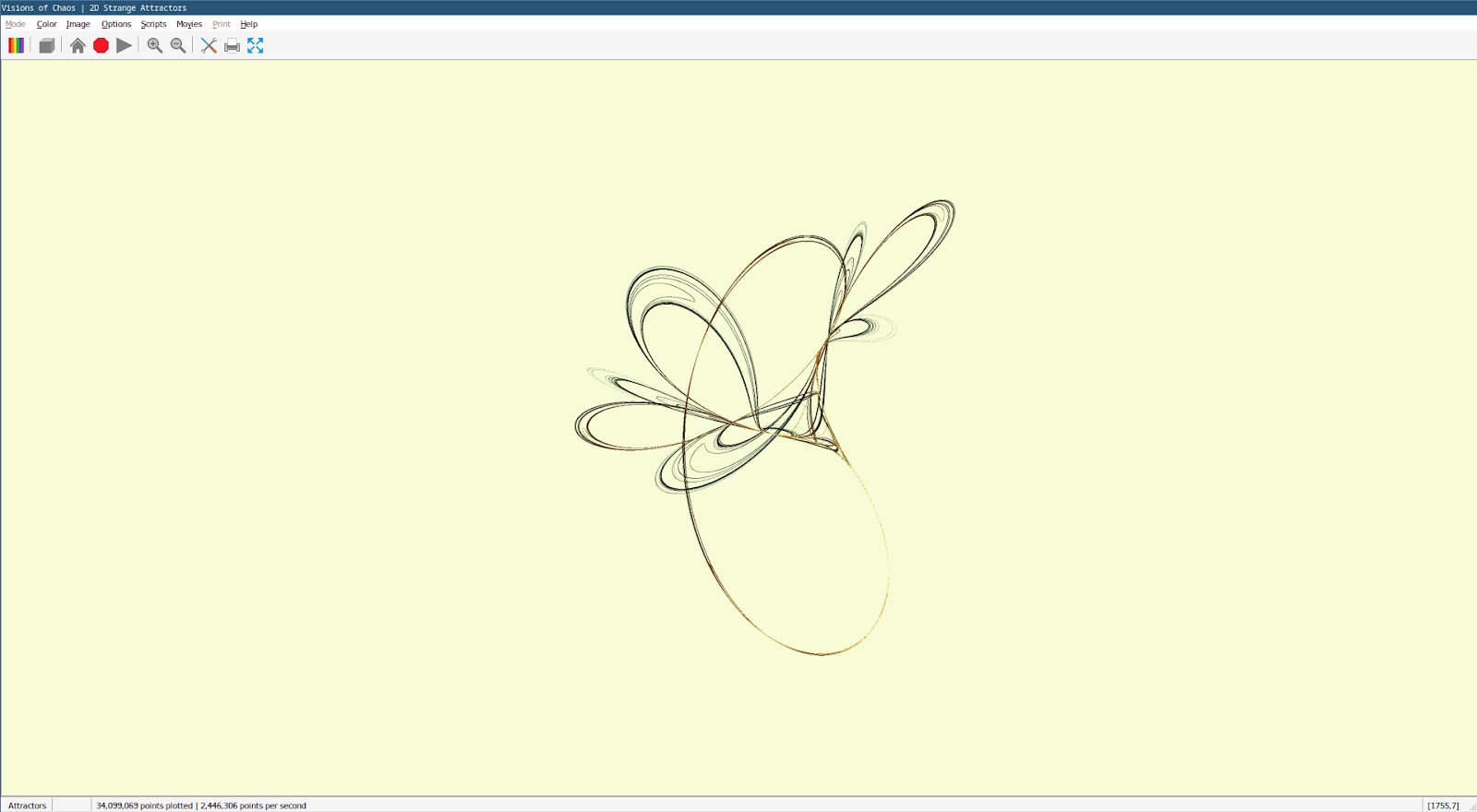
*Visions of Chaos* is a windows program made by *Softology* focused on simulating various mathematical models. The software is available for all versions of windows and focuses on being an all-in-one tool for rendering chaotic models. The software is kept up to date, but the attractors module does not get frequent updates. The software accomplishes some of the goals of the project but is more focused on creating soothing images rather than providing tools for mathematical analysis or searching through parameter combinations.

Figure 1: Example image generated by Visions of Chaos

Figure 2: Example image generated by Visions of Chaos

Figure 3: Attractor selection and config window

Figure 4: The color gradient editor

Figure 5: The program running in Wine

### Features

The software, however, has features the stakeholders expressed interest in:

* A range of built in colormaps makes customization easy
* Powerful colormap editor that makes creating complex colormaps easier.
* The viewport settings are intuitive and easy to use, utilising the mouse rather than number or sliders.
* Creating videos is simple and fast via use of the move maker dialogue.
* Supersampling is effective at increasing detail and fidelity.
* Program runs well on low-end machines.
* Images can be exported in a range of ways, including directly to the wallpaper in windows.

The stakeholders, however, had some issues.

* Changing parameters is awkward and clicking through menus to do so is time-consuming.
* Some settings are not explained well in tooltips.
* Program can freeze up when trying to stop rendering.
* Cannot enter custom equation.
* Linux support under wine is slow and difficult to use.

### Technological Analysis

This software first generates a list of points based on the formulas for the attractor, with some xstart and ystart value where x and y first start. Then, a list of millions of points is created, ordered by each iteration of the xnew and ynew formula. Then, for 3d models, cylinders are drawn between each consecutive point – i.e., in ordered list [A,B,C] the points A and B, and B and C will be joined. For 2d attractors, points are simply plotted on a lattice at least 3x the width and height of the final image, then a supersampling algorithm is applied to the larger grid to downscale the resulting image by 3x. This leads to a higher fidelity, antialiased image with less “jaggies” and with more detail. Note that the larger lattice is clamped between 0.5 and -0.5 in both X and Y dimensions, and all points scaled into this window so that the window always contains all points. This means there is no need for a viewport or viewport settings but could lead to some attractors being squished or stretched in one or both dimensions. Then, the coloring of each cylinder in 3d or point in 2d is based on the initial co-ordinates of the point or start of the cylinder by converting each co-ordinate axis to a gradient colormap, usually XYZ to RGB but for 2d maps this could be XY to any gradient map. Then, the black background color is replaced with the specified background color. Custom attractors are not explicitly supported; however, the author takes suggestions on attractors to add, and adding attractors is possible but difficult even with some knowledge of programming, via the built-in code editor.

The source code and explanations also reference the idea of the delta hyper-parameter, which is an exceedingly small scalar all points are multiplied by to keep them from “exploding” or going to infinity. Depending on the value of delta, the attractor may look different than expected, through no fault of the program. This is something I need to consider when evaluating the success of the program, as I will compare the images generated by my program to existing images of the attractors to determine the success of the program. I would like the user to be able to edit this delta value in the preferences window.

The code also discards the first couple thousand points of an attractor (a different amount per attractor) as these points form a “tail” outside the basin of the attractor. This tail is **not** part of the basin of attraction, so should not usually be included in the output image of a solution. This is something I should consider when creating my solution, however I would like the user to be able to choose to include or not include the tail in the preferences.

Overall, I think this rendering method could be highly effective to generate detailed images, however due to the high space complexity of the algorithm it could fail on machines with less than the required ram to store a list of all points, especially when points for attractors are in the billions. This issue, however, is only present because **all** points are generated then simultaneously rendered. A similar method where a point is generated, rendered and then that point is used to generate the next point and the first point dropped out of memory could potentially remedy this issue, at the cost of time complexity.

## *Chaoscope*

### Summary

*Chaoscope* is a windows program made by *Nicolas Desprez* with a Linux-compatible command-line version. The software focuses on rendering 3d attractors, rather than 2d models, but is still a powerful piece of software. The software was last updated on the 31st of Oct 2010, so is relatively outdated now. The software does not have support for user defined attractors, but instead uses a group of predefined attractors that potentially have undiscovered parameter combinations that lead to mathematically interesting behaviour.

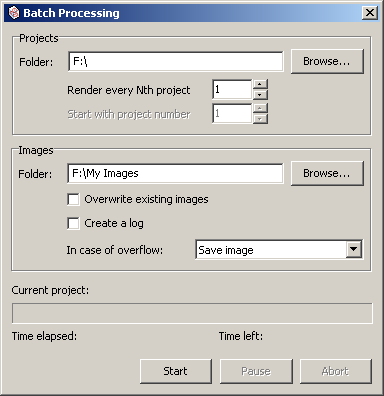
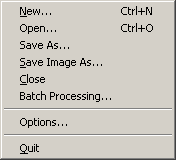
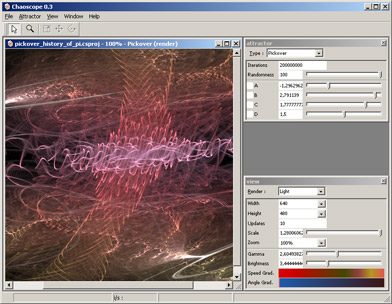
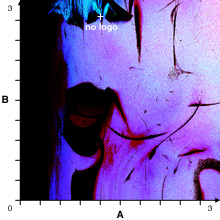


Figure 10: The File menu

Figure 10: The program in use.

Figure 10: The batch rendering menu

Figure 10: The search window with custom parameter axis.

Figure 10: An example output file

### Features

I presented the software to my two stakeholders. The software has the following features that the stakeholders expressed interest in, or liked:

* Command line and graphical batch rendering
* Project files that save parameters, viewports, etc
* Search window for searching for parameter combinations that lead to interesting behaviour
  + Graph based method, where parameters are mapped to an XY plane
  + Automated method that randomizes parameters
  + Randomness and parameter exclusion to help automate searching
  + Sliders to change parameters slightly
* Level parameter for root-based attractors (i.e., Julia attractor)
* Preview window.

And they disliked the following:

* Inability to use user defined equations
* Colormaps are hard to use and are light based.
* Complicated UI that is difficult to use without mathematical background.

### Technological Analysis

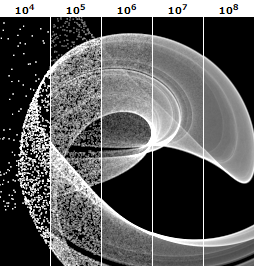


Figure 11: The gas rendering mode at different numbers of iterations

The program has 5 rendering modes to choose from. The “Gas” rendering mode (Fig. 6) is based on “pixel accumulation”. The pixel grid is projected into space as a lattice and extended into the Z dimension infinitely. Whenever the orbit is updated, the “brightness” of the lattice point (corresponding to a pixel) is increased by one. Then all pixel values are normalised, and a gradient is applied, usually black to white as in Figure 6. Each iteration is one update of the orbit and one pixel brightness increase. The gas rendering mode was designed to be used on old CRT monitors, and as such has parameters that make the images clearer specifically on those CRT monitors, such as “Gamma” and “Contrast.” However, these parameters are no longer as necessary as modern monitors do not suffer from low brightness and contrast and therefore do not need to be adjusted for.

The “Liquid” rendering mode is the same as the Gas rendering mode, except adds depth and opacity via the use of a “Z-Buffer.” The Z-Buffer rendering splits the 3D space into a lattice of width and height being the resolution of the screen, and the depth being some value. Then, the usual pixel accumulation algorithm is run, except pixels are discarded if a pixel has a lower Z value (i.e., is in front of that pixel from the cameras perspective.). Opacity means that each pixel is given an opacity value, proportional to the brightness value. When this value is less than one, pixels behind this pixel contribute some proportional amount to the brightness of the pixel in front rather than being discarded.

The “Light” rendering mode is similar to the Gas mode, but instead of assigning brightness's to pixels, we assign colors to points in space, and then to the relevant pixels. This means the color of a pixel encodes data about the points that pass through that space, usually being the speed and angle. This can be set to various gradients to change the look of the final image.

The “Plasma” rendering mode is similar to the “Light” rendering mode but makes use of a Z buffer and opacity.

Finally, the “Solid” rendering mode is similar to the Liquid mode, except the accumulation value defines only opacity, and the Z-Buffer is rendered using a ray-tracing method. This allows the render to look like a solid surface, with roughness, diffusion, highlights, reflections and more.

The Solid rendering mode is not useful for my project as there will be no Z-Buffer as the project will only deal with the X and Y dimensions. The same therefore goes to the Liquid, Light and Plasma modes too, as they also utilize a Z-Buffer which would not have any effect on the rendering of images by my program. However, the Gas rendering mode could be a highly effective approach for rendering in my project. The algorithm is simple and slowly increases in accuracy and clarity as it is run and can be used in 2D easily. The method, due to its iterative nature, would also work on slower machines as it can simply be rendered at a lower number of iterations, as well as being able to effectively render previews by limiting the resources available and iterating slowly. This method also circumvents the space complexity issues presented by the algorithm used in *Visions of Chaos*. As such, I will give consideration to this as a rendering method.

## *glChAoS.P*

### Summary

*glChAoS.P* is a piece of software designed to utilise openGL to visualise and explore various 3d fractals. It runs in Linux, Windows, MacOS and webGL. The project was created and is maintained by *Michele Morrone*. The software is not longer actively maintained, with the last update in 2020. The software had a focus on fast rendering and having a range of preset systems, ranging from attractors to hypercomplex fractals.

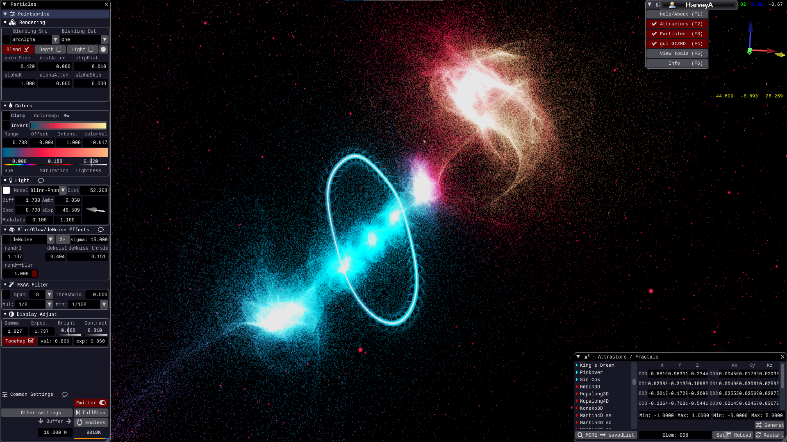
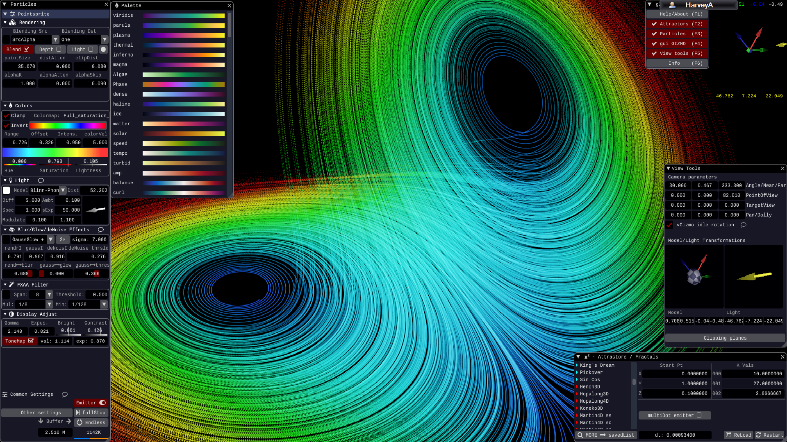


Figure 13: wglChAoS.P rendering the Lorenz attractor. Rendering, colour, and viewport settings are shown.

Figure 12: wglChAoS.P rendering a fractal. On the left is shown the rendering settings

### Features

I presented the software to my two stakeholders. The software has the following features that the stakeholders expressed interest in, or liked:

* The ability to change, edit and save colormaps makes fine tuning the look and feel of a render easier.
* Compatibility with multiple operating systems, so the end user can use the software on Linux.
* Easy to use camera orbiting using the mouse is very intuitive.
* Multiple emitter modes, allowing the user to choose the particle size and the overall look.
* De-noise and glow features allow for better-looking images.
* FXAA Anti-Aliasing allows for higher fidelity images.
* Easy to use parameter UI with the ability to type or drag parameters.
* Window-based UI with a preview in the background makes adjusting settings extremely easy.

And they disliked the following:

* Inability to input custom systems of equations.
* No support for 2d systems, only 3d systems.
* Does not utilize the GPU as well as it could.
* Lighting system seems unnecessary and complex.

### Technological Analysis

The software uses OpenGL for rendering and is programmed in C++. This also allows the program to use webGL and also can utilise the speed of OpenGL. The fractals and attractors are rendered using a dot emitter that emits particles that lie along the path of a point as it evolves over time. The colormap represents when in time the point would have evolved. This approach is effective because as the rendering continues the picture gets increasingly accurate. The billboard mode also allows for the illusion of solid surfaces from solid spheres around the emitted points. This rendering method, however, strains the GPU as millions of points must be rendered simultaneously every frame, with more being added every second as the emitter functions. This approach might not be effective in Python due to its slow speed, but further testing will be required. This is similar to the Gas approach in *Chaoscope*, but does not rasterize a point into its closet pixel, but rather renders the point as a sphere with some radius. Due to the high resolution of modern screens, this may be a better approach as the human eye struggles to differentiate pixels on a modern monitor, so colouring pixels could lead to eye strain as the user cannot properly see individual pixels, but could easily see a particle of fixed radius, spanning multiple pixels.

Furthermore, the approach is suited to both high-end and low-end machines as lower emitting rates and less emitters can speed up rendering (at the cost of lower clarity and slower increase in clarity) on low end machines, and higher emitting rates and more emitters can be used on higher end machines to utilise the more powerful graphics processors and therefore speed up rendering.

As such, this emitter rendering approach could be highly effective for my project and may be well suited to the stakeholders needs. The emitter approach is similar to the approach in *Chaoscope*, and also circumvents the memory issue in *Visions of Chaos*, while also being able to reduce time complexity using multiple emitters and multithreading. As such, this is the most suited approach to the rendering problem.

# Analysis - Features

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | | | **Source** | **Description** |
| Interface and user experience | Graphical User Interface (GUI) | Sliders to adjust values | Me | Sliders can be used to adjust values accurately and easily. |
| Text boxes for data entry | Text boxes can be used for entry of equations and precise values. |
| Radio buttons for Boolean choices | Radio buttons can be used for easily toggling Boolean choices. |
| Camera rotating and panning via mouse and keyboard | Existing solutions | Click-and-drag can be used to move the viewport around in space and the Q and E keys can be used to rotate the viewport. The scroll wheel can be used to zoom in/out. |
| Tooltips | Isabelle | Tooltips describe parts of the GUI. |
| Preview Window | Existing solutions | A preview window is present in the GUI. |
| Command Line Interface (CLI) | Command line operation of the program | The program can be operated from the command line. |
| Command line batch rendering | Mark | Multiple images can be rendered at once using the command line |
| Rendering | Rendering Stills | Changing Colormaps | Isabelle | The colormap of the image can be changed by the user. |
| Changing Viewport | Existing solutions | The viewport can be moved, zoomed, or rotated by the user. |
| Rendering Videos | Rendering videos with time as the time dimension | Me | The user can choose to watch as a system evolves by rendering multiple images at different time values and stitching them together. |
| Rendering videos with parameters as the time dimension | Instead of at different time values, the rendering can be done using different parameter values. |
| Render Previews | Render still previews | Existing solutions | The GUI has a preview window that renders low-quality fast preview stills. |
| Render video previews | The GUI has a preview window that renders low-quality fast preview gifs. |
| High quality/fidelity | Super sampling | Me | The image is supersampled via a supersampling algorithm. |
| Anti-aliasing | The image is anti-aliased in some way. |
| High res | The image can be up to 4k resolution, |
| Saving and loading | Saving and loading project files | Saving | Existing solutions | The user can save and load a whole project |
| Loading |
| Saving and loading config files | Saving | The user can save and load just config settings. |
| Loading |
| Mathematical / Analysis | Range of built-in mathematical functions | Trigonometric | Me | Trig functions can be used in an equation. |
| Exponential | Exponential functions can be used in an equation. |
| Matrices/vectors | Matrices/vectors can be used in an equation. |
| Piecewise | Piecewise functions can be used in an equation. |
| Hyperbolic | Hyperbolic functions can be used in an equation |
| Search function | Parameter searching | Me | Search through parameter space or phase space via a search window with customisable axis. |
| Space searching | Mark |
| Entering custom equations | The user can enter their own custom system equations | The user can, using the above functions) define equations for and and render images for those systems. |

Key: Orange text represents optional features.

# Analysis - Limitations

## Dimensionality

The software will have some limitations. The software will only deal with 2-dimensional systems as 3d rendering is outside of the scope of the project as it would be difficult to develop and implement. As such, certain “classic” examples of chaos will not be able to be rendered by the program without losing a dimension. This limitation is not major as programs already exist to visualise 3d systems, whereas programs to deal with 2d systems do not, so this limitation could be considered a benefit or limitation.

## Efficiency

Another limitation is the slow speed at which python, a high-level programming language, runs at compared to lower-level languages such as Rust or C++. The rendering of higher quality images will take much longer, and the real-time rendering of the preview window may be very slow. This could be remedied by using a lower level, faster language however I am not comfortable enough in those languages to develop a solution using them. However, this limitation is only present in the proof-of-concept that I will be developing, and a fully developed solution might be coded in a lower-level language and therefore avoid this limitation.

## Mathematical

There will be some mathematical limitations on the solution. For example, the libraries I use for the various mathematical functions might not be as precise as they could be. This could lead to significant variations in output due to the chaotic nature of the maths – small changes in input lead to substantial changes in output. This limitation could be significant as it may cause unexpected mathematical behaviour and lead to incorrect analysis. This limitation cannot be entirely circumvented either, but I could use more precise libraries and lessen the error margin, however that could require using libraries I am not comfortable using, or potentially coding my own.

## Rendering

Modern monitors can only show so much detail. Strange attractors have an infinite amount of detail due to their fractal nature. As such, modern computers will never be able to fully capture all of the detail of a strange attractor and could potentially lose out on clarity and detail. My project will suffer from this limitation as it also is intended for use on a modern computer. However, this limitation is unavoidable, so is not an issue with my program but rather with the problem itself.

## Numerical Stability

The program could be limited by the numerical stability of floating-point numbers. Since the system is inherently chaotic, the difference between 0.3 and the binary representation of 0.3 could cause massive changes in output.

# Analysis - Requirements

## Stakeholder Requirements

|  |  |
| --- | --- |
| **Requirement** | **Explanation** |
| Lightweight, graphical interface | The GUI needs to be lightweight and easy to understand so that less technical users looking to use it to relax rather than for maths can easily operate the program without needed mathematical or technical knowledge. |
| Running on both Linux and Windows | One of my stakeholders uses Windows, the other uses Linux, so the program will have to run on both. |
| Output of common filetypes such as PNG. | The output files will need to be viewed, edited, and shared easily so use of a common filetype is required |
| Saving and loading projects. | Saving and loading projects is especially important to Mark as he may want to revisit areas of interest and it would be difficult to write down or remember the considerable number of parameters the system may be using. |

## Functionality Requirements

|  |  |
| --- | --- |
| **Requirement** | **Explanation** |
| Rendering still images of a system. | The program should be able to render still images of a system’s phase space given its equations and a viewport. |
| Entering custom equations for a system. | The user should be able to enter their own custom governing equations |
| Moving around the viewport to render different areas. | The user should be able to move, rotate and scale the viewport as they see fit. |
| Editing parameters of a system. | The user should be able to edit the parameters of a system of equations. |
| Settings menu. | There should be a built-in graphical settings menu. |
| Changing colours of the image. | The user should be able to change the colormap of the image. |
| Search window. | The user should be able to use a search window to find areas of interest in phase space. |
| Range of built-in mathematical functions. | The user should be able to utilise a range of different mathematical functions when entering their custom equations. |

## Hardware and Software Requirements

|  |  |
| --- | --- |
| **Requirement** | **Explanation** |
| A computer with standard periphery | The computer needs to have a mouse and keyboard to use the GUI. |
| Python 3+ interpreter with libraries | The program will be written in python and will need libraries: |
| **PyOpenGl –** for image processing |
| **TKinter** – for GUI rendering. |
| **scikit-video** – to process videos. |
| Image displaying software | The user will need software such as nomacs to view the output images. |
| Video displaying software | The user will need software such as VLC to view the output videos |
| Windows or Linux | The user will need either a windows operating system or a Linux operating system. |

# Analysis – Success Criteria

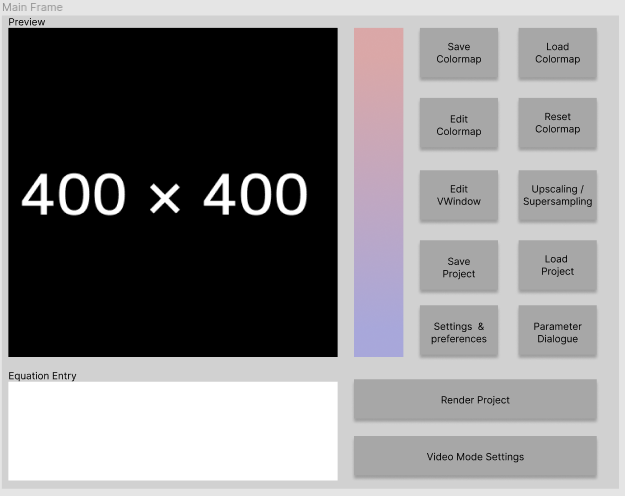
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SC** | | **Name** | | **SRC** | **Description** | **Justification** | **Evidence** |
| 1 | a | GUI Windows | Main window | Me | The main window where parameters, colormaps, equations and such will be entered, saved, and loaded. | The program needs the main window to function. | Screenshots of the different windows, and videos of their usage. |
| b | Settings window | The settings window where output filetype, rendering settings and such will be entered, saved, and loaded. | A settings window allows for decluttering and easy setting changing. |
| c | Preview window | SH | The preview window where a lower resolution preview of the render will be shown. | A preview window allows the user to see the effect of their changes before rendering and was requested by a stakeholder. |
| d | Search window | The search window that allows the user to search through either parameter or XY space and find points of interest. | A search window allows the user to find interesting viewports with ease and was requested by Mark. |
| 2 | a | GUI Usage | Tooltips | SH | Tooltips and lightweight, intuitive GUI controls used so that the user can easily understand and use elements of the GUI. | A new end user will feel lost in a cluttered, undocumented UI. Also, Isabelle requested tooltips. | Video of usage showing a lack of clutter and clear controls with tooltips. |
| b | Ease of use | Me |
| 3 | a | Command Line Interface | CLI operation of the program | SH | The program should also be able to be operated entirely from the command line or terminal. | Stakeholder Mark expressed interest in being able to batch render. | Screenshots of command-line operation working. |
| b | CLI batch rendering | The user should be able to utilise batch rendering while operating the program from the command line. |
| 4 | a | Saving/Loading | Saving project files | SH | Saving and loading project files that restore parameters, equations, viewport, colormap, etc. | Saving and loading project files was talked about with stakeholders. | Videos of saving and then loading project and config files. |
| b | Loading project files | SH |
| c | Saving config files | Me | Saving and loading config files that restores rendering settings, preferences, etc. | Saving and loading config files allows for users to share and transfer their rendering settings and preferences. |
| d | Loading config files |
| 5 | a | Rendering | Rendering Stills | Me | Rendering still images given a viewport, colormap and a system of equations. | This is required for the basic function of the program. | Example inputs and outputs. |
| b | Rendering Videos | Rendering videos given a viewport, colormap and system of equations and a variable to change with time. | This would allow for more interesting and soothing visuals. |
| c | Supersampling | Rendering stills at a higher resolution then downscaling for a higher fidelity | This allows for higher fidelity and clarity in images, for analysis. |
| d | Anti-aliasing | Using an anti-aliasing algorithm on the rendered stills to reduce or remove “jaggies” |
| e | High-res mode | An optional “High-res” mode that renders in 4k and up. |
| f | Filetype support | Support for at least PNG and JPG output filetypes. | End users may want to share their outputs and having support for multiple filetypes makes that easier. |
| g | Colormaps | SH | Colormaps can be applied to a render that colors points differently based on a metric that applies to all points. | Isabelle asked for this feature. |
| 6 | a | Maths | Entering custom equations. | SH | Entering custom equations to govern a system. | Stakeholder Mark asked for this feature for analysis purposes. | Example custom equations, screenshots of them being entered and their outputs |
| b | Support for a variety of mathematical functions. | Me | Support for functions such as sine, cosine, natural log, etc. | This would allow Mark to enter any equations that he would want to analyse. |
| 7 | | Parameters | | Me | Support for attractors with parameters, as well as GUI elements that allow them to be changed. | Some attractors have parameters. Being able to change them allows for those attractors to be used. | Screenshots of changes in parameters via GUI then a change in the output. |
| 8 | | Built-in example projects | | Me | Some project files are built into the software. | Inexperienced users can look at these example projects to learn the software or because they create soothing visuals | Screenshots of example project files and their default outputs. |
| 9 | | Support for multiple operating systems. | | SH  Me | Support for Windows-based and Linux-based operating systems | Stakeholder Mark uses Linux, I use both and Isabelle uses windows. | Screenshots of the program running on both operating systems. |
| 10 | | Accuracy | | Me | The program should accurately create images of the attractors, with the same image being produced given the same equation and parameters. For example, the Lorenz attractor should look like the Lorenz attractor and be mathematically identical to other renders of the Lorenz attractor, every time it is rendered. | Without accuracy, the program would be pointless for use by Mark, and could confuse inexperienced users wanting to test “classical” attractors. | Screenshots of attractors, then compared to existing images of those attractors. Note that changes in rendering settings and delta value may lead to slight variations, but this is acceptable as they are mathematically identical. |

# Design – User Interface

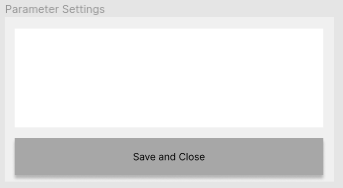
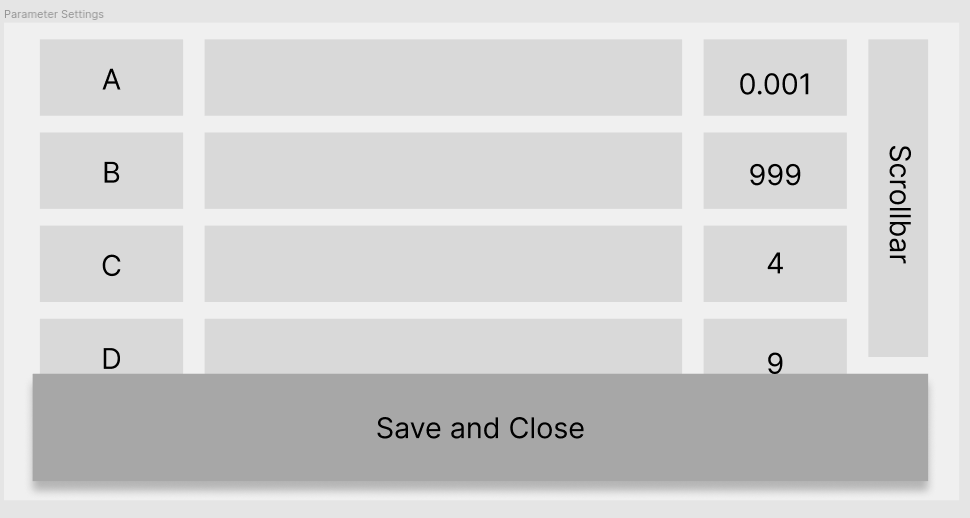
I decided to start by designing the look and feel of the program. This was a self-contained task that I could do before decomposing the problem, and since it required back-and-forth between me and the end users, I wanted to get it finished before moving on to the other parts of the problem.

In order to design the user interface, I used the online design tool Figma.

I first designed the main landing page:



This main page design has a large preview window so that the user can see the effect of their changes. Also included is a preview of the current colormap, so that the user knows what colormap is loaded and can see their changes to the colormap. I have opted to abstract lots of the more complicated options for the program behind extra windows so that the user does not feel overwhelmed and a new user can use the program more easily. For example, the supersampling settings are behind a button which is clearly labelled. This links to 2b as it keeps the function of each button clear and easy to use. Furthermore, I will include tooltips to each button to meet criteria 2a. The equation entry box is also on this page so that the user can easily see any issues with the equations they have entered. Finally, large buttons for rendering and video mode are present as these will likely be the most used buttons as this is the main purpose of this page.



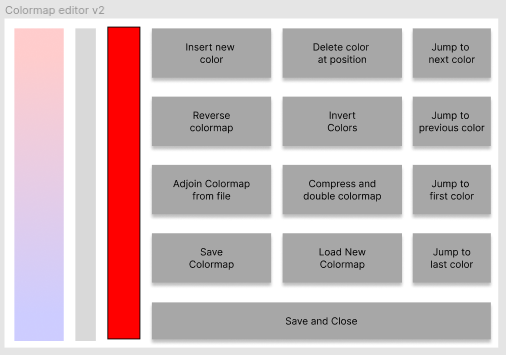
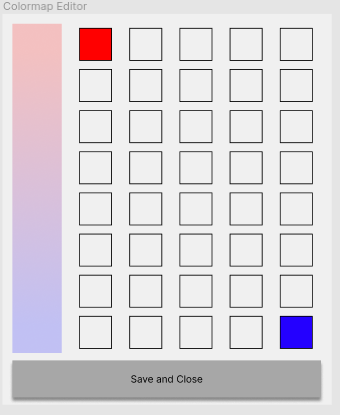
On the left is my initial design for the parameter settings window, and on the right is my revised design for the parameter settings window. Originally I planned for the user to enter expressions to define each parameter, as so:

X=3.14159267

Y=12345678

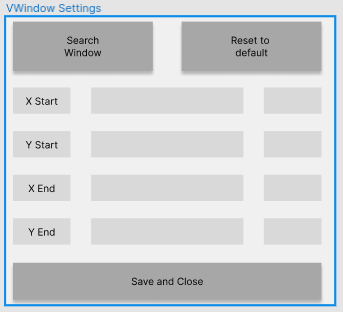
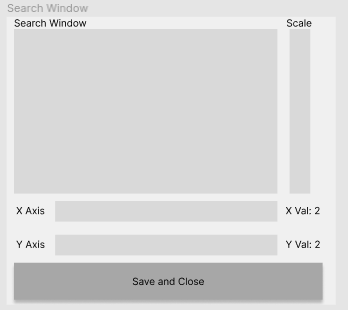
Z=sin(X/4)

However after a dialogue with the end users, we decided a slider and numerical entry box would be more suited to the program, especially to success criteria 2b. As such, the parameter settings page will include a scrollbar to scroll between parameters if there are many parameters to configure. Each parameter will have a slider ranging from -10 to 10, and a text entry box that can take any value, which will overwrite the slider. This allows for rough adjustment by slider for end users such as Isabelle, and precise adjustment for end users such as Mark. Tooltips will be used here to explain this overwriting behaviour.

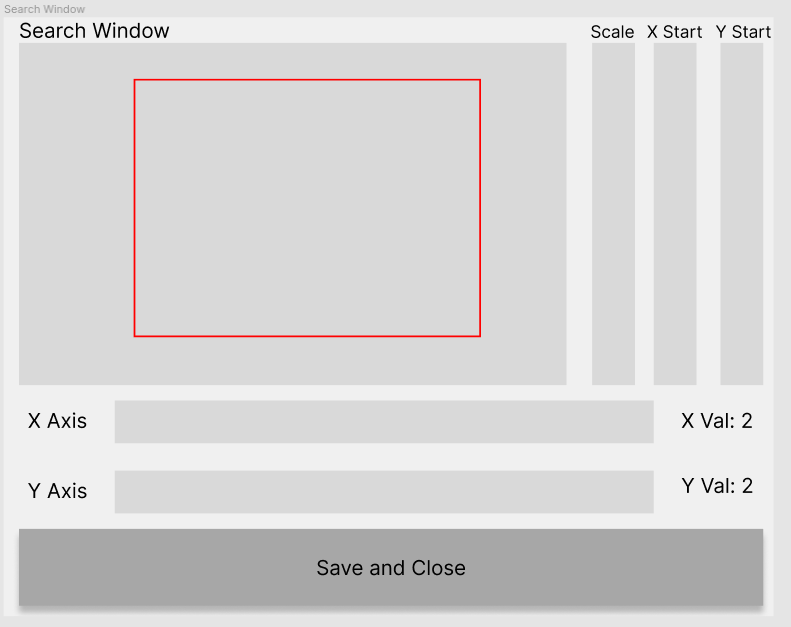


Another window that changed designed in this dialogue was the colormap editor window. I struggled with the initial design of this, using a multitude of color selector buttons to add colors at positions. However both end users struggled to understand how this worked and asked for a new design, so I came back with the design on the right, which utilised a slider to adjust the position that the user was editing, and a multitude of button functions. The jump functions can be used to jump to a “color peak” – where a color is most intense. Buttons allow for colors to be inserted and deleted, colormaps can be reversed, inverted and doubled. Also, colormaps can be loaded, adjoined from file and saved. These functions were agreed on by end users and myself to include all functions that the user of the program might want to edit a colormap. A small window here is also used to preview the color at the position of slider so that the user can see clearly where the slider is and what color that position is. The colormap can be previewed in both designs so that the user can clearly see their changes. Tooltips will also be applied here to explain each function in detail.

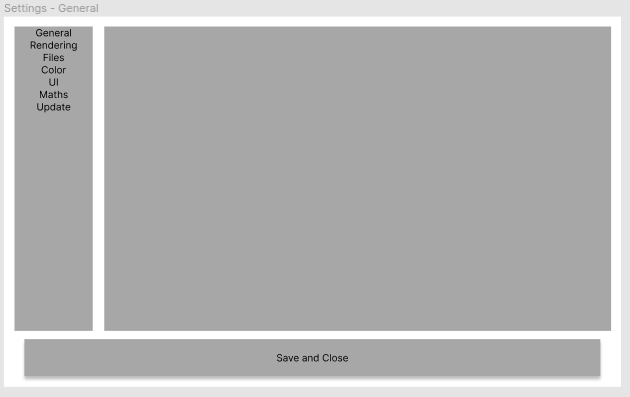
The user will use the system color picker dialogue when inserting a new color, as both Linux and Windows have an effective built in colorpicker. This saves programming time for me, and also makes sure that the user interface is friendly as most people can use their operating system’s built in color picker as they are familiar with it.



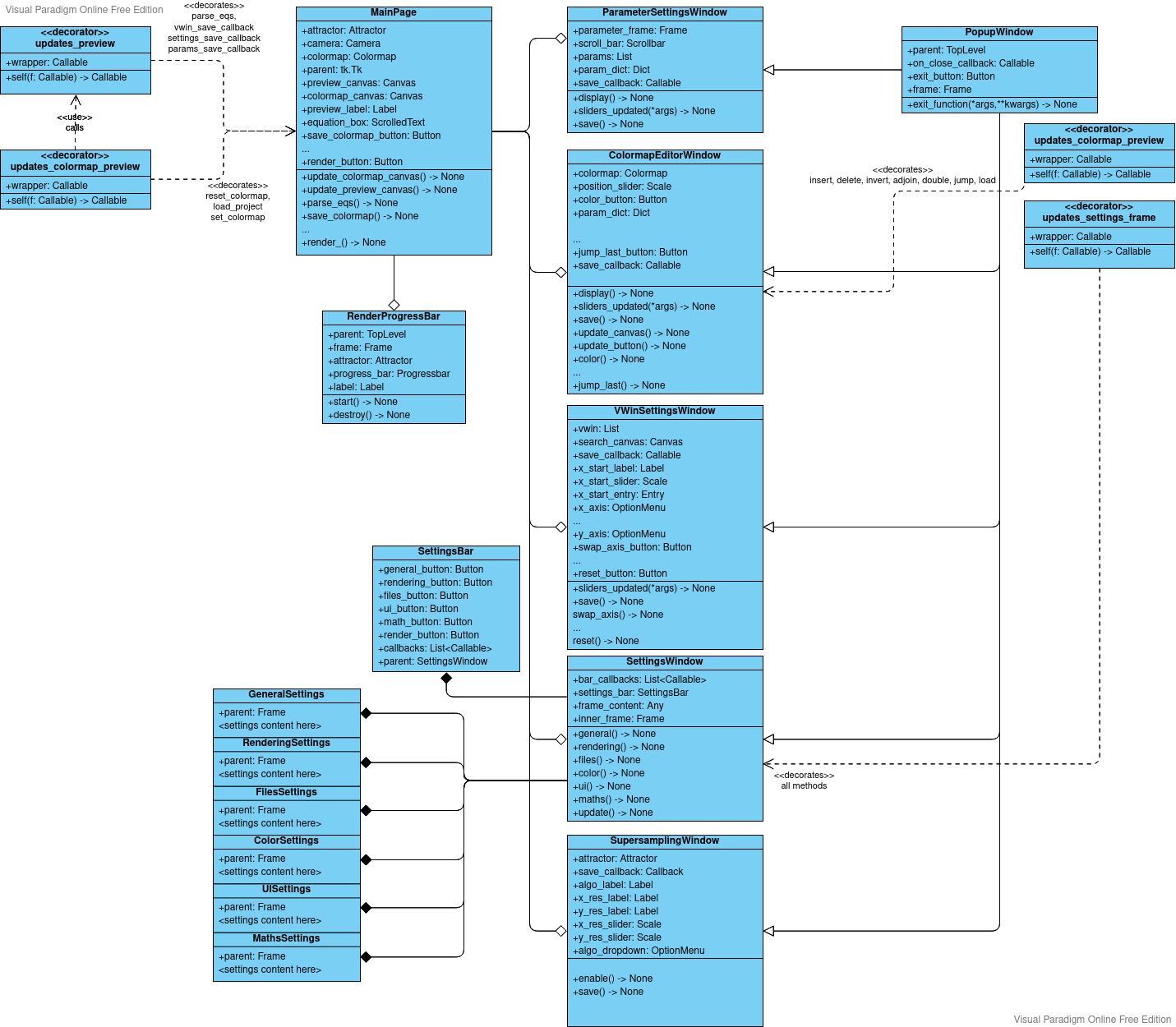
These were my original designs for the VWindow settings and search window. These featured sliders on the VWindow settings to choose the start and end cords, and also text entry boxes for specific values. The search window could be opened with the Search Window button, which used 2 dropdowns to select what each axis displayed, labels to show where the mouse is in XY space, and a scale slider. I decided that the purpose of this scale slider was ambiguous as it was unknown where the scale slider would zoom into. I redesigned the search window as follows so that the program was easier to use:



This was my revised design for the search window, which allows for zooming on a specific point using the red rectangle. The distance from the edges of the red rectangle and the edges of the search canvas are constant – the scale, xstart and ystart sliders the size and start of this rectangle. The aspect ratio of this rectangle is fixed and is based on the v-window settings from the previous window. This was done to actually represent what was and wasn’t being shown to avoid ambiguity.



This was the design I decided on for the settings window. A bar on the side would be used to change the tab – or the content in the central frame. I showed no content here as I am still unsure on what settings I will include in each tab, but a variety of sliders, radio buttons and text boxes could be used to edit each setting. I will add settings as I develop the program – as some variables may lead themselves to being implemented as settings, and some may not.

I decided to use Tkinter to develop this user interface, as it is a standard module for UI and is well documented – and includes features to code all of the widgets I would need. I decided an OOP approach to this would be most appropriate (see Design – Justification) and created this UML Diagram:

The UML Diagram shows the structure of the front-end of the program. Firstly, the application will employ the object-oriented features of Python to encapsulate windows, and in some cases, frames. The application will launch to the MainPage class, which has attributes for the objects needed for the backend of the program. This page will also contain the buttons shown in the UI diagrams. Some of these buttons will open new windows - which are encapsulated into classes. This pseudocode shows how the program will achieve this:

method settings(self)

settings\_window = tk.Toplevel(self.parent)

settings\_app = SettingsWindow(settings\_window)

end method

The program will use callback methods and initialisation arguments in order to pass variables between classes. For example:

CLASS MainPage

…

PUBLIC METHOD vwin\_save\_callback(self, start: Vector, end: Vector)

    self.camera = new Camera(start, end, self.cam\_rotation)

END METHOD

…

END CLASS

CLASS VWinSettingsWindow(PopupWindow)

PUBLIC METHOD new(self, parent: TopLevel, save\_callback: Callable)

super.init(parent, exit\_callback = save\_callback)

self.save\_callback = save\_callback

self.start = Vector

self.end = Vector

END METHOD

PUBLIC METHOD save(self)

self.save\_callback(self.start, self.end)

END METHOD

END CLASS

This example shows a callback method in MainPage being passed to a VWinSettingsWindow class, and being called with attributes as it's arguments. When VWinSettings:save is called, the MainPage class will initialise a new Camera with the start and end vectors from the VWinSettingsWindow instance, and set the MainPage class instance variable camera to that newly initialised Camera object. This is done with callbacks for three reasons:

1. It is clear what the program is doing
2. It does not involve the large overhead of globals
3. It works with the asynchronous nature of tkinter

The program will also use the Pythonic decorators to wrap functions that cause an update to a different element of the UI, which this pseudocode shows.

DECORATOR updates\_preview(f):

LOCAL FUNCTION wrapper(\*args):

return\_value = f(\*args)

self = args[0]

self.update\_canvas()

return return\_value

END FUNCTION

return wrapper

END DECORATOR

In this pseudocode, the decorator updates\_preview calls the function which changes the UI, and then once it has executed, updates the canvas. This could be done as so:

FUNCTION change\_ui(self):

change\_ui()

self.update\_canvas()

END FUNCTION

However, in my experience, using decorators creates much clearer code as it becomes in my opinion much more obvious when a function is updating the canvas unnecessarily and creating an overhead.

One notable point about the update\_preview and update\_colormap\_preview decorators is that the update\_colormap\_preview decorator should also invoke the function to update the preview canvas if it is called by the MainPage. I had two possible solutions to this: wrapping the function twice; or having the callback for updating the canvas invoke the decorator to update the canvas. I found the second to make more logical sense, as it prevents potential errors where the function is only decorated once.

Another notable feature of the structure I will be using is the PopupWindow class. This class is a parent class for all windows that are not the root window. This contains a parent TopLevel object, an exit button and an option on\_close callback. This is done as this code would be common to all of the popup windows, so I decided to use inheritance to save writing out the same code for each class.

Finally, the SettingsWindow class has some special features. It is a composition of multiple classes - a base SettingsWindow class that contains callbacks for switching the content of an inner frame, classes for each possible inner frame, and a SettingsBar class that contains buttons to switch between the various possible inner frames. The SettingBar class is used for the same reason as the PopupWindow class - the buttons in the class are common to all settings windows, so they are placed into a class and composed to save writing extra lines of code. It also allows for the easy addition of new settings pages. Finally, the inner frame content classes are used as this allows for each settings page to be encapsulated, and also allows for a seamless user experience as the inner frame content simply switches between classes.

Overall, my design for the UI should fully accomplish the following success criteria:

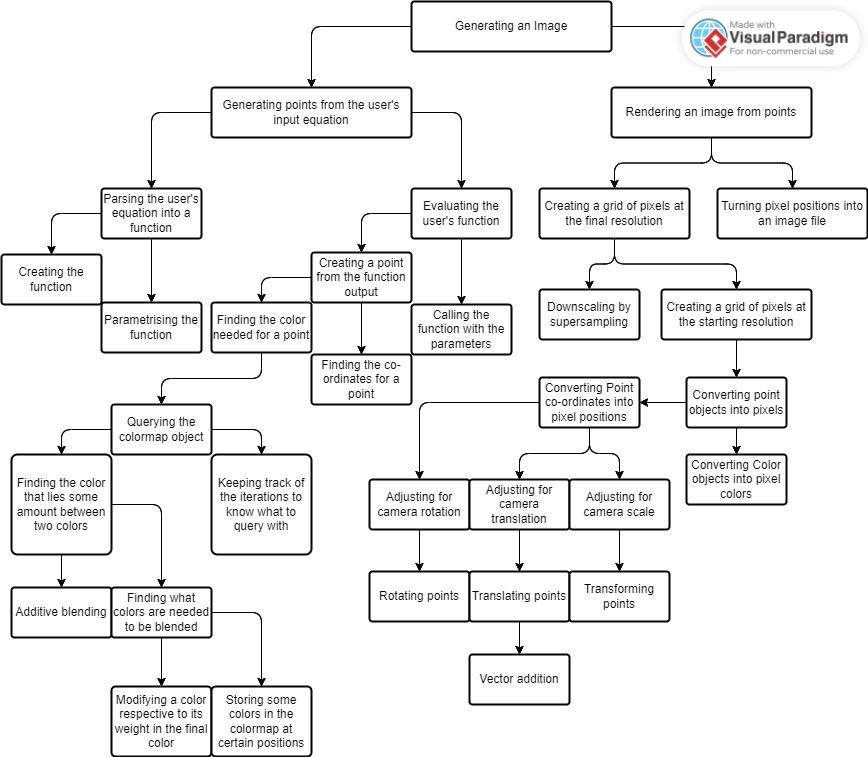
* 1a,b,c,d
* 2a,b
* 4a,b,c,d
* 6a

And partially accomplish:

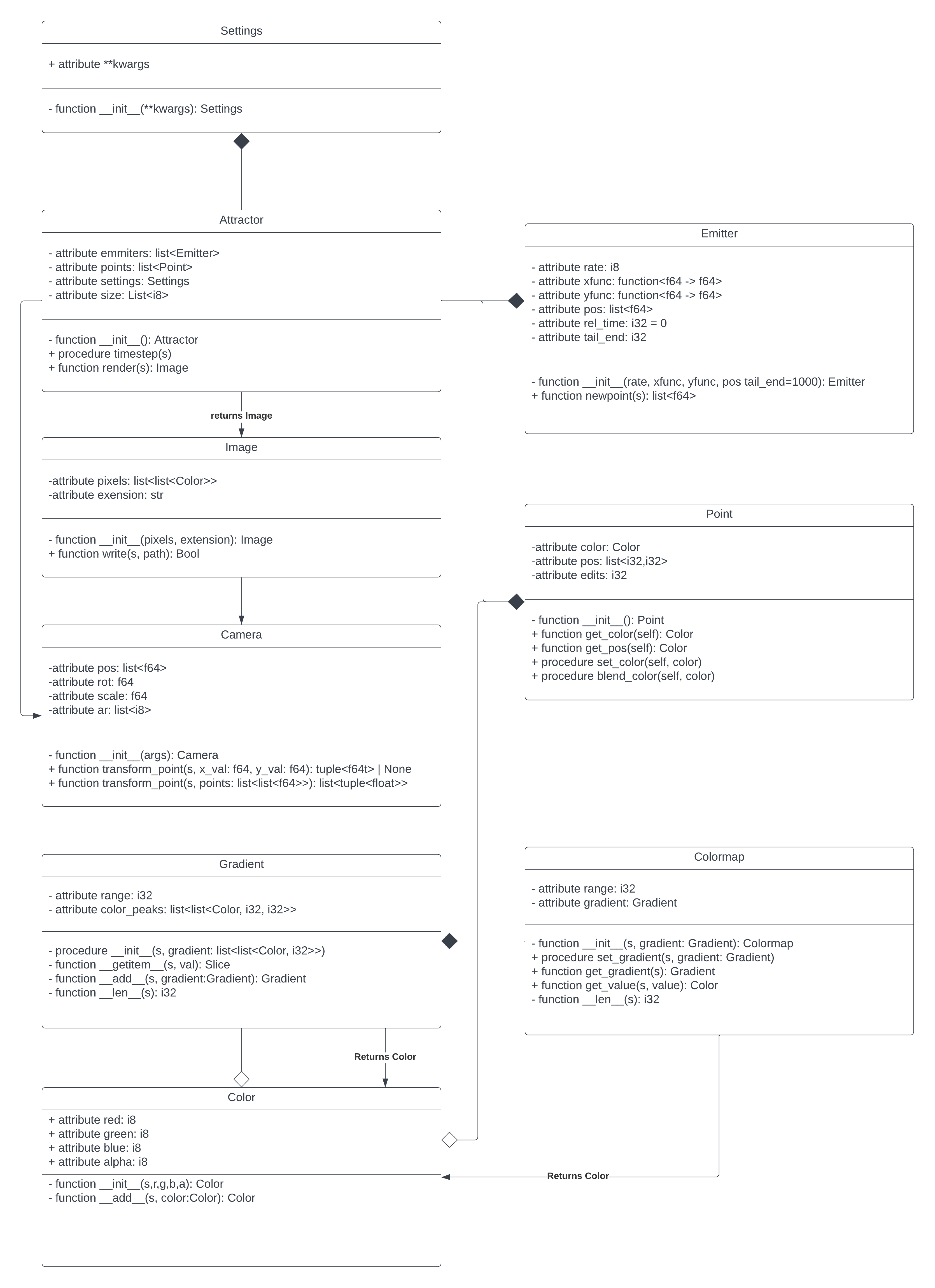
* 7

# Design – Rendering Algorithms

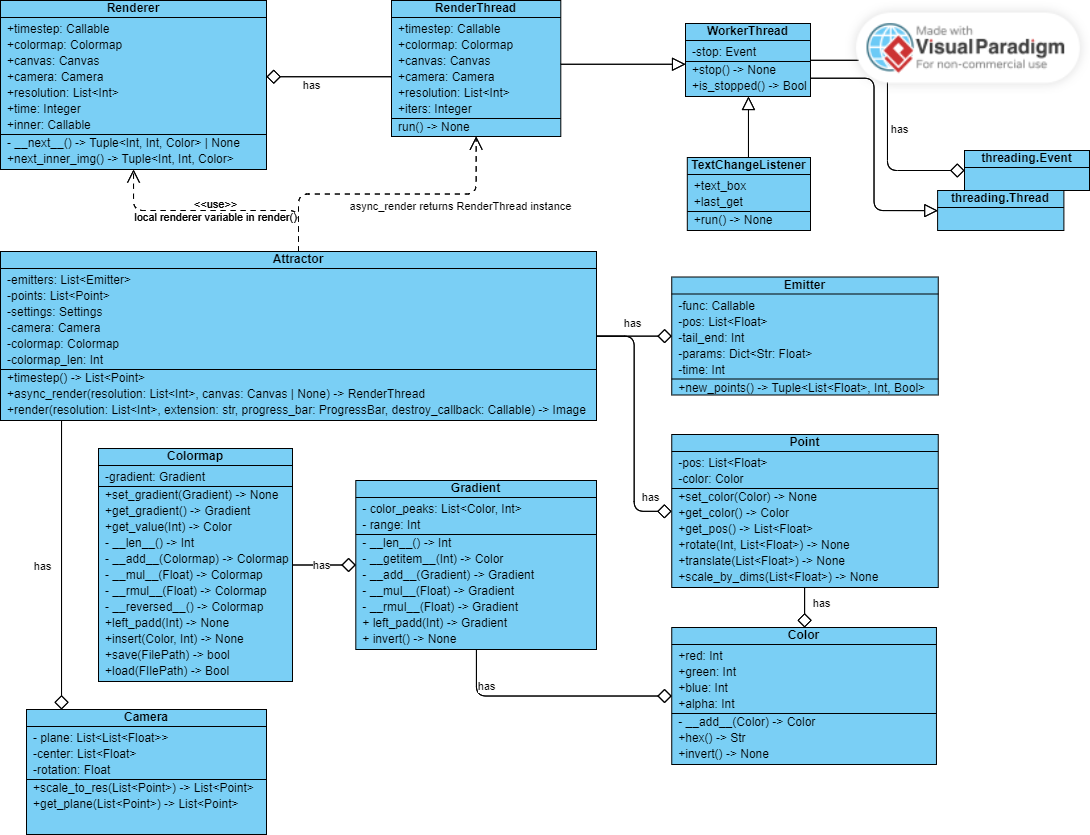
Aside from the user interface, rendering is the other problem my solution needs to solve. Since this is a highly complex problem, I first broke it down into smaller, more manageable problems using a decomposition algorithm.



And also, this was my original UML diagram.



However, I found that, once I was developing a solution, this UML diagram was missing lots of features (see development part 3) so I recreated a new one, with the features I realised I would need.



**Equation Parsing**

Equation parsing will be done by writing the custom user equations in Python. Python is both easy to understand for the user as well as easy to parse into a function for generating new values using `exec`. The parameters set by the user will be compiled as a parameter passed to the function generated by parsing the equation. This is done as it allows for the program to easily switch out functions and parameters, as the equation will be a class attribute that can be set or changed, and the parameters a dictionary of keyword arguments that can also be set and changed. My algorithm for equation parsing is as follows:

Pseudocode and flowchart:

FUNCTION get\_params(rawtext)

loc = {"x":1,"y":1,"t":0,"dx":None,"dy":None}

glo = {}

exec("from math import\*", glo, loc)

params = {}

FOR line in rawtext.split("\n"):

FUNCTION process\_line(loc, params):

x=dict\_join(params, loc)

TRY

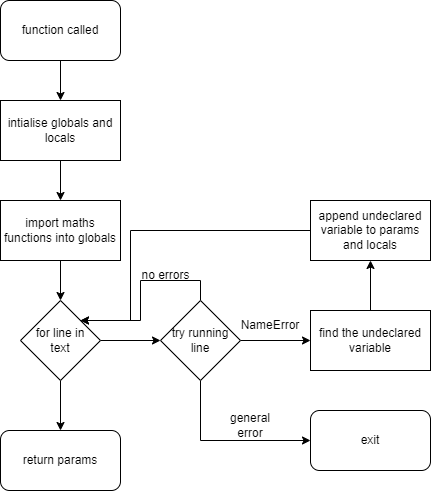
exec(line, glo, x)

CATCH NameError with error as e:

new\_param = str(e).split("'")[1]

params = dict\_join(params, {new\_param: 1})

return process\_line(x, params)

 return x, params

ENDFUNCTION

loc, params = process\_line(loc, params)

ENDFOR

return params.keys()

ENDFUNCTION

FUNCTION parse\_eq(rawtext: string) -> Dict, Callable

params = get\_params(rawtext)

LOCAL FUNCTION gen\_params(\*\*kwargs)

return new Dict(kwargs)

END FUNCTION

LOCAL FUNCTION func(\*\*kwargs):

glo = new Dict

loc = gen\_params(\*\*kwargs)

exec("from math import\*", glo, loc)

exec(rawtext, glo, loc)

return loc["dx"], loc["dy"]

END FUNCTION

return params, func

END FUNCTION

CLASS MainPage

…

METHOD parse\_eqs(self) DECORATED BY updates\_preview

rawtext = self.equation\_box.get()

self.params, self.func = parse\_eq(rawtext)

self.params\_dict = self.process\_new\_params()

del self.attractor

emitter = new Emitter(self.func, self.params\_dict, self.start\_pos, self.tail\_end)

self.attractor = new Attractor([emitter], [], self.camera, self.settings)

ENDMETHOD

END CLASS

**Emitters**

Each emitter will be an instance of the Emitter class. Information on classes can be seen in the UML diagram. Each emitter can start at a different starting position, enabling each emitter to function at various positions, or with some configuring by the user generate images more quickly using multithreading. This is also done to separate parts of code into different classes – with the code for generating points in Emitter rather than Attractor to keep the code clear and readable.

**Points**

Points will be generated by Emitters and stored in a list of points which is a class attribute of the attractor class. New points generated will be returned to be processed by the Renderer object as this allows some concurrency. This concurrency may or may not speed up the program, as the python global interpreter lock might prevent the benefit of concurrency. However, processing points like this also means that the preview window can generate iteratively while the GUI is running. This is a major advantage as otherwise the program would “hang” or stop responding every time the preview needed to update, which is something I would like to avoid to ensure a smooth user experience. Finally, rendering of actual final images will be synchronous, as there would be no benefit to asyncronosity due to the global interpreter lock and the sequential nature of point generation.

**Colors**  
Colors, colormaps and gradients will be managed by the respective classes. Gradients can be edited using the operators built into the Gradient class and colors can be blended using the built-in operators in the Color class. Blending will be done via an additive RGBA algorithm. This is done as this blending algorithm is common across similar projects, so users will probably be familiar with it. Finally, I will use an RGBA algorithm over and RBG algorithm as having it is not particularly more work to implement, and could be functionality required by a user. As such, I feel it is justified to use an RGBA algorithm over an RBG one. I will not use HSV algorithms as many users would be inexperienced using these algorithms and this could cause confusion, which I would like to avoid. In future, I could add optional support for HSV color picking if needed, so the lack of HSV algorithms is not an issue.

**Time**

An instance time variable will increment every time a point is added to the space. This keeps track of the number of iterations the algorithm has done. This instance variable may need to be passed between various class instances to make sure they are all in time – which could cause some overhead processing time. However, the alternative is a global time variable which could cause even more overhead processing time as global variables are very inefficient, so this method is preferable.

**Downscaling**

There will be various supersampling downscaling algorithms available to the user to pick from:

*Poisson Disc*

*Uniform*

*Quincunx*

*Rotated grid*

These algorithms all have upsides and downsides, so the user will be left to pick their favourite. I used these visual references to understand each algorithm:

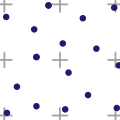
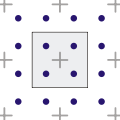
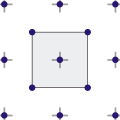
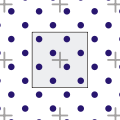
   

Image Credit [Vierge\_Marie](file:///\\ke5-mbr-store1\HarveyA$\My%20Settings\Application%20Data\Microsoft\Word\Vierge_Marie) on <https://en.wikipedia.org/wiki/Supersampling>

**Converting points to images**

This will be done using the render function in the Attractor class.

This will be done by use of the Python Image Library. PIL allows for the pixel data of an image to be edited, which will accomplish the rendering of points, to pixels, and then to an image. However, colors in PIL are represented by RGBA hex strings, so I will need a subroutine in the Color class to convert the data in the color class to an RGBA hex string.

PIL also has the ability to support multiple file extensions, which sis in accordance with my project’s success criteria, so it is the most appropriate library to use. The use of a library is justified here as the algorithms involved in writing file types such as JPEGs are very complex and would take much longer to write than is possible in my timeframe and been implemented in python by many people much more efficient way than I would be able to due to my lack of experience writing file encoding algorithms.

**Worker Threads**

The program will utilise my stoppable worker thread as this allows for rendering to be stopped which is important as rendering might have a set end point or be terminated by the user. Since multithreading code is often complex, I created pseudocode to assist with the development:

CLASS WorkerThread(Thread)

PUBLIC METHOD new(self, \*args, \*\*kwargs)

self.super.new(\*args, \*\*kwargs)

PRIVATE self.stop = Event()

END METHOD

PUBLIC METHOD stop(self) -> None

self.stop.set()

END METHOD

PUBLIC METHOD is\_stopped(self) -> bool

return self.stop.is\_set()

END METHOD

END CLASS

**Render Threads**

As per the issues discussed in the Points section. I will do rendering for the preview thread asynchronously aside the main thread and gui threads. As such, I will require a RenderThread that inherits from the WorkerThread. I created pseudocode for render thread:

CLASS RenderThread INHERITS FROM WorkerThread

PUBLIC METHOD new(self, timestep, colormap, camera, canvas\_inner, resolution\_inner, \*args, \*\*kwargs)

self.super.new(\*args, \*\*kwargs)

PUBLIC self.timestep = timestep

PUBLIC self.colormap = colormap

PUBLIC self.camera = camera

PUBLIC self.canvas = canvas\_inner

PUBLIC self.resolution = resolution\_inner

PUBLIC self.iters = 0

END METHOD

PROTECTED METHOD run(self) -> None:

renderer = new Renderer(self.resolution, self.timestep, self.canvas, self.colormap, self.camera)

WHILE not self.is\_stopped() and self.iters < 25000:

next(renderer)

self.iters += 1

END WHILE

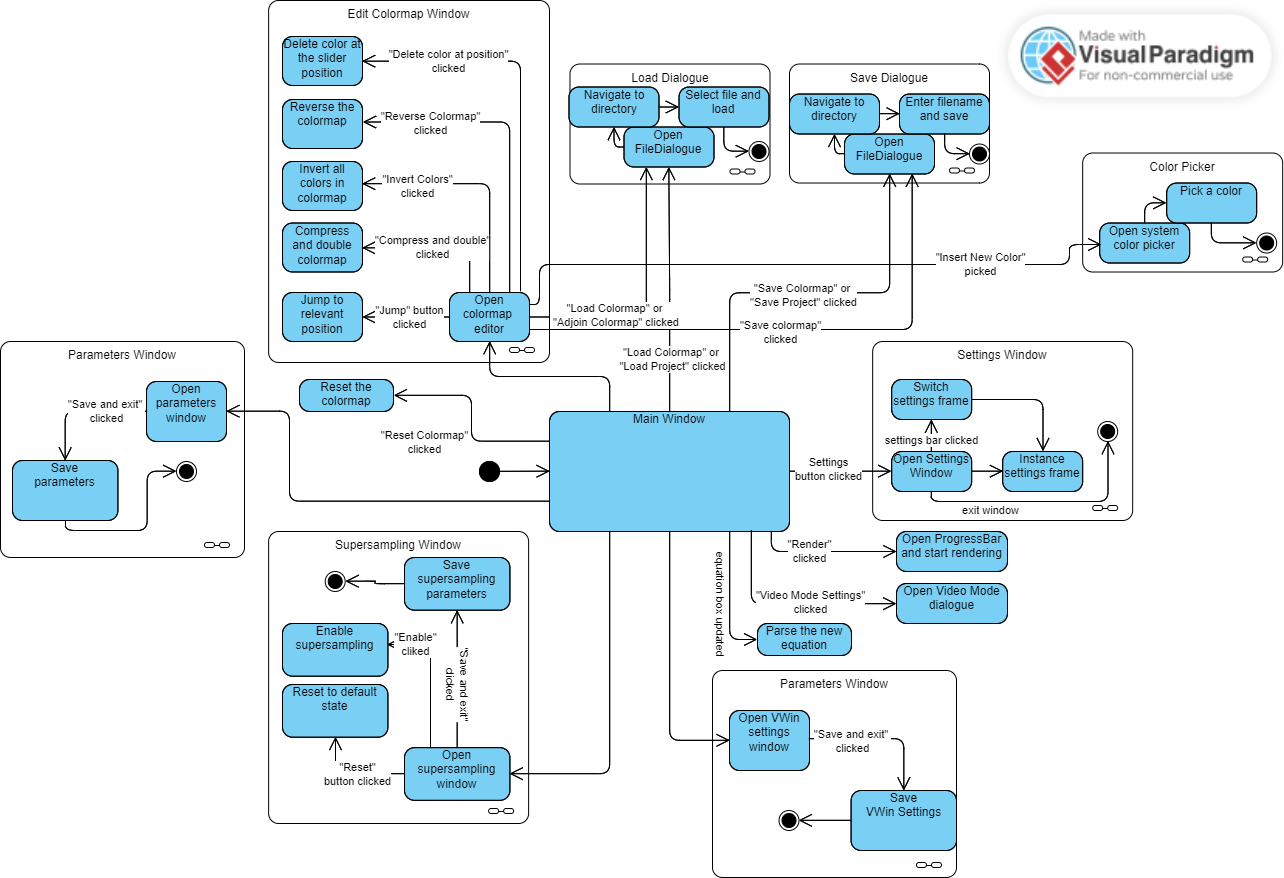
END METHOD

END CLASS

**Renderer Object**

A renderer class will be used to generate new points for both canvas and image rendering. The basic structure of this class can be seen in the UML class diagram. It is used to link the Attractor object, Camera object and Canvas object or PIL together for the creation of either a preview or a final image.

# Design – UI Algorithms

Diagram showing how subroutines link in the UI:

Some subroutines will be commonly used across classes, with varying specifics based on the needs to the class. Pseudocode for these subroutines is found here:

Pseudocode for creating new windows (colormap window shown here):

METHOD edit\_colormap(self)

colormap\_editor\_window = new Toplevel(self.parent)

colormap\_editor\_app = new ColormapEditor(colormap\_editor\_window)

END METHOD

This method creates a new TopLevel object, opening a window, then instantiates the relevant window class as a child of the TopLevel. Neither of these are saved as attributes, and are only kept as locally bound varaibles. This is done to preserve memory as after the window is closed, an attribute would keep taking up memory whereas the locally bound variable has its scope removed from memory, destroying the data and freeing up memory.

Many classes might need to use a callback to get information saved by a child window:

METHOD vwin\_save\_callback(self, xs, ys, xe, ye) DECORATED BY updates\_preview

self.vwin\_params = xs, ys, xe, ye

self.camera = new Camera([xs, ye], [xe, ys], self.cam\_rotation)

del self.attractor

emitter = new Emitter(self.func, self.params\_dict, self.start\_pos, self.tail\_end)

self.attractor = new Attractor([emitter], [], self.camera, self.settings)

ENDMETHOD

Shown here is an example for how this would look to save the VWin settings in the MainPage class after the VWin settings window is closed. Note that the camera and attractor have to be reinstantatied as their components change.

Pseudocode for saving and loading objects might be commonly used too (colormaps shown here):

METHOD save\_colormap(self)

filetypes =[("Colormap File", "\*.cmp"),("Raw Text Colormap", "\*.txt")]

filename = savedialogue(filetypes=filetypes, defaultextension=".cmp")

self.colormap.save(filename)

ENDMETHOD

METHOD load\_colormap(self) DECORATED BY updates\_colormap\_preview

filetypes =[("Colormap File", "\*.cmp"),("Raw Text Colormap", "\*.txt")]

filename = loaddialogue(filetypes=filetypes, defaultextension=".cmp")

temp\_colormap = new Colormap

temp\_colormap.load(filename)

self.set\_colormap(temp\_colormap)

ENDMETHOD

In these functions, the system filediagloues can be used to get user input. This is done as the user is usually familiar with these and it seems frivolous to re-code something that is tried and tested.

Finally, from experience I am aware the Textbox from tkinter does not have a command callback that is called when the text is changed, so I designed a listener for doing so:

METHOD equation\_box\_change\_listener(self)

t = new TextChangeListener(self.equation\_box, self.parse\_eqs)

t.start()

ENDMETHOD

# Design – Linking Program Halves

The user interface and the rendering algorithms will need to link together. Also, individual parts of the program halves will need to link together. This will be done by the flow of data around objects in the program. I created a communication diagram to represent this:

# Design – Justification

My task contained a lot of complexity and was not easy to approach – however – the use of decomposition has made the problem seem far more approachable and has led to having the ability to deal with the complexity one part at a time. Not only has this made design of the program much easier, but during development the modular nature of the solution will assist in maintenance, coding, debugging and much more.

First, a clarification on how I have decomposed the problem. I have split my proposed solution into two areas: the user interface, which is tricky as the data the user needs to input is non-trivial, and the creation of images. The both rendering and UI halves of the solution will take a multi-threaded object-oriented approach. The multi-threading is done to make sure that the relatively slow python can utilise the processor as much as possible; a lack of multi-threading could lead to a very slow and near unusable program due to the computational complexity of the rendering process. I took an object-oriented approach as it allows me to test individual components (classes) and their functionality individually, and then simply link them together using methods such as callbacks. However, a functional approach would also have allowed this. When choosing which approach to use, I evaluated how much code would need to be repeated in each approach, whether either approach would have an advantage over the other in terms of implementation in python, and how other solutions have approached the problem.

Overall, I realised I could use inheritance, aggregation and composition of classes to avoid repeating code and speed up the development process. A great example of this is the PopupWindow class which contains code used in nearly all UI classes - which, in a functional approach, would have to be repeated. Also, use of these object-oriented methods allows debugging to be easier - for example, the Attractor class aggregates the Emitter class. If there is an issue in the Emitter class, I can separate it from the Attractor class and debug it on it's own, and visa versa. Secondly, an object-oriented approach also allows me to use \_\_add\_\_, \_\_mul\_\_, \_\_len\_\_ and similar pythonic “magic methods” that make my code significantly more readable. For example, an object-oriented approach to adding colors can be seen in this pseudcode:

CLASS Color

…

METHOD \_\_add\_\_(self, other):

return ColorMixingExpression(self, other)

END METHOD

END CLASS

>>> Color(r1,g1,b1,a1) + Color(r2,g2,b2,a2)

<”Color” object>

Whereas in a functional approach

FUNCTION AddColors(C1: Tuple[Int],C2: Tuple[Int]) -> Tuple[Int]:

return ColorMixingExpression(C1, C2)

END FUNCTION

>>> AddColors((r1,g1,b1,a1),(r2,g2,b2,a2))

(r3,g3,b3,a3)

Though the code in a functional approach is shorter, the syntax for adding colors is much, much clearer and can even use functions like sum. Also note that in a functional approach, Colors would be represented as tuples of integers, so accessing an individual color (via color[1] for example) is much less readable than in an object-oriented approach (color.green) which is another reason an object-oriented approach is superior.

Finally, all other solutions used an object-oriented approach and do so effectively, so it is likely that my solution can utilise an object-oriented approach effectively too.

The simplifications in coding created by the multi-threaded object-oriented approach I have chosen to take justify my choice of approach in my opinion.

# Design – Key Variables

## Table of key variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable Name** | **Type** | **Binding** | **Usage** |
| App | MainPage | Local | The instance for the MainPage. |
| attractor | Attractor | App | The attractor instance contains data on how the attractor should be rendered as well as rendering methods |
| camera | Camera | App | The camera instance contains VWin information and how it should be applied |
| settings | Settings | App | The settings instance contains a variety of user-set settings |
| colormap | Colormap | App | The colormap instance in use refers to the colormap the program is using |
| func | Callable | App | The parsed equation for the attractor |
| colormap\_editor\_app | ColormapEditor | App::edit\_colormap::Local | The ColormapEditor instance for when the colormap editing window is opened. |
| vwin\_config\_app | VWinConfigWindow | App::edit\_vwin::Local | The VWinConfigWindow instance for when the vwin editing window is opened. |
| supersampling\_config\_app | SupersamplingWindow | App:: supersampling::Local | The SupersamplingWindow instance for when the supersampling settings editing window is opened. |
| settings\_app | SettingsWindow | App:: settings::Local | The SettingsWindow instance for when the settings window is opened. |
| param\_app | ParameterSettingsWindow | App:: parameters::Local | The ParameterSettingsWindow instance for when the parameter editing window is opened. |

## Tables of inputs and outputs

|  |  |  |  |
| --- | --- | --- | --- |
| **What is being input?** | **Type of input** | | **Usage** |
| Equation | text box | | Parsed into the func variable |
| Colormap settings | ColormapEditor, buttons, color. | | Used to generate a colormap |
| VWin Settings | VWinConfigWindow, slider, numeric, button | | Used to generate a Camera instance |
| Parameter values | ParameterSettingsWindow, slider, numeric | | Used to pass parameters to the renderer |
| Project, Colormap, etc files | File | | Used to save/load aspects of the program |
| Supersampling Settings | Dropdown, numeric | | Also used to generate Camera instance |
| The user is ready to render | Button | | Pressed when the user wants to render |
| **What is being output?** | | **How is it output?** | **Why is it output?** |
|  | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |

## Data Validation

# Design – Testing Methodology

# Development

## Code Style

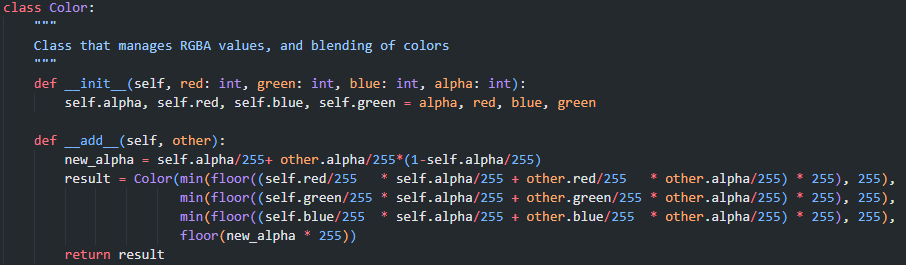
All common code written for this project should adhere to a personal styleguide. This styleguide is very basic:

* All parameters should be typehinted unless the required type is the self-type.
* All functions should have a specific return type unless the required type is the self-type.
* All methods, functions, Modules etc should have a simple descriptive docstring
* All variable and function names should be snakecase
* All class names should be upper camel case.

All GUI Code should be written similarly, aside from lack of type hints/docstrings for more rapid development.

## Part 1: Colors and Gradients.

I decided to start by developing the Color, Gradient and Colormap classes, as these could be tested via a library to display colors such as Processing, rather than tested in conjunction with other parts of my code. This made debugging easier as I was only testing one class at a time.



The Color class has two methods. The \_\_init\_\_ method is a constructor which assigns the public instance variables red, green, blue, and alpha to parameters passed to the constructor. The \_\_add\_\_ method is called whenever the class is the operand in an + operator expression. In my solution, Color + Color expression refer to the blending of two colors, which will be used in the gradient creator and rendering process. This is done by a standard additive color mixing algorithm.

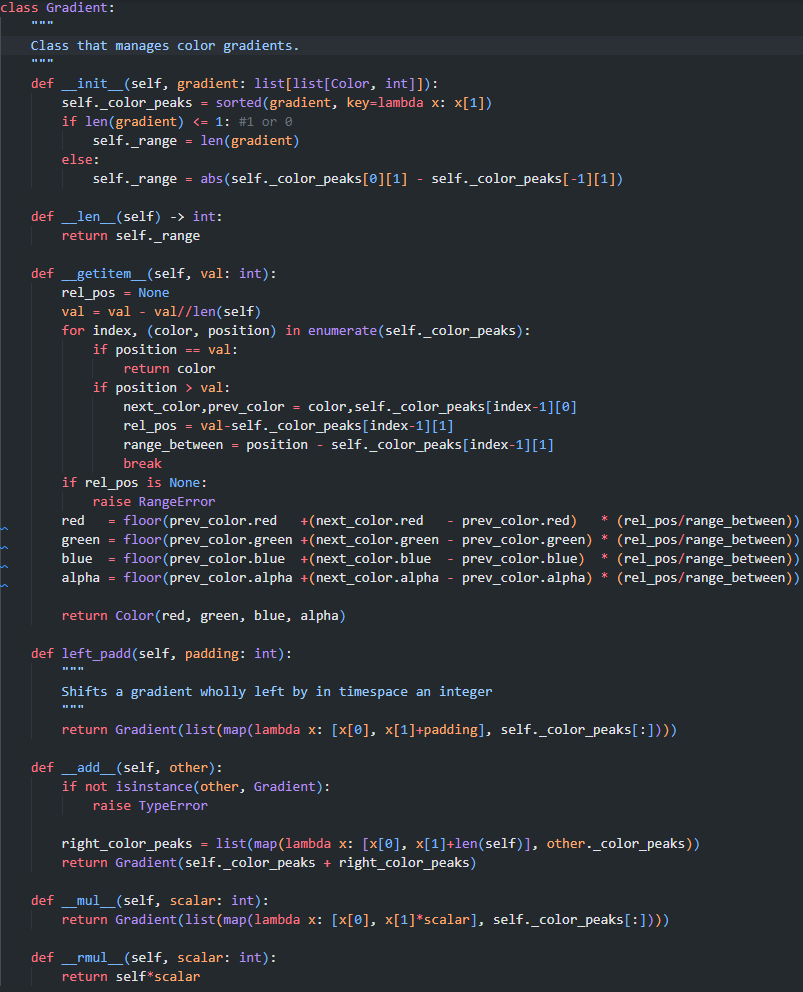
At first, I had issues with this outputting decimal RGBA values. I did not want that, as those values should be integers clamped between 0 and 255. I fixed this via the inclusion of floor statements.

I tested this class using the Processing library. I wrote a function to cast my Color class to the Processing color class, then rendered it onto a processing canvas to see the color. I wrote code to blend two colors and display them, and check that it appeared to be working correctly. Screenshots of this testing are in figure 15 and 14, at the top. Screenshots for the blending testing are in figure 13.





Figure 13: Two colors on the left blended form the color on the right (x2)

After this, I began work on the Gradient class:

The gradient class has 7 methods; firstly, the \_\_init\_\_ constructor which takes a list[list[color, int]] as a parameter, then sets the protected instance variable \_color\_peaks to that value but sorted with respect to the position of peaks. This value is used to define where the “peak” of a color is within a gradient – i.e., where that color is at full intensity. Then, the length of this value is checked, to find the “range” of the gradient. The range of the gradient is stored in a protected instance variable. The \_\_len\_\_ method is used as a getter for this range, as this function is called when an instance is passed to the len built-in.

The \_\_getitem\_\_ function is called when an instance is sliced. The slice value is passed to the function as a parameter (here it is called val). First, I clip the value to be in the range of the gradient, then use an enumerated for loop to find the next color peak along from the position passed, as well as the previous color peak. I also find the position relative to the previous color peak, and the range between these two peaks. If no valid color peaks are found, a RangeError is raised, and if the position is a color peak, that color is returned. Then, I use 4 linear equations to smoothly transform the RGBA values of the previous color to the RGBA values of the next color. I get the RGBA values at the position relative to the previous color, and return a Color object with those RGBA values.

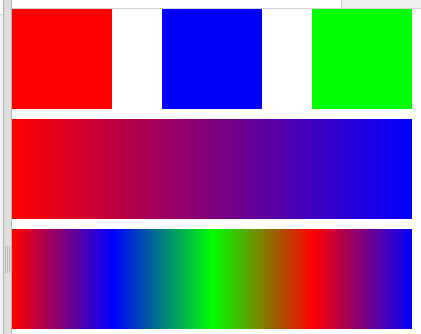
The left\_padd function shifts all the color peaks of a gradient along, in order to create a padding of no color at the start. This is used in gradient addition, to set a space to blend the left gradient final color to the right gradient first color.

The \_\_add\_\_ function is used to join two gradients end to end, by offsetting the gradient right on the operand so that there is no intersection between the two gradients, and then returning a Gradient with color peaks equal to the conjoined color peaks of the gradient and the offset gradient.

The \_\_mul\_\_ function is called when an instance is used in a \* expression as an operand. Since this is used to multiply the Gradient not by another Gradient, but rather by an integer, both \_\_mul\_\_ and \_rmul\_\_ must be used. The former is for when the instance is the left operand, and the latter is for when the instance is the right operand. These functions scale the whole gradient up by an integer by multiplying all of the color peak positions by the scalar.

Initially I did not implement the \_\_rmul\_\_ function, which lead to error when I tried to have a Gradient instance on the right of the operator. Also, I discovered an edge-case error when testing this class, where passing None to the constructor would result in an RangeError, which was fixed by adding the code “if len(gradient) <= 1:” and the following line.

I tested this class in a comparable way to the Color class – I wrote code to sample a gradient and render the color observed on a Processing canvas. I took some screenshots of this testing (fig 15)



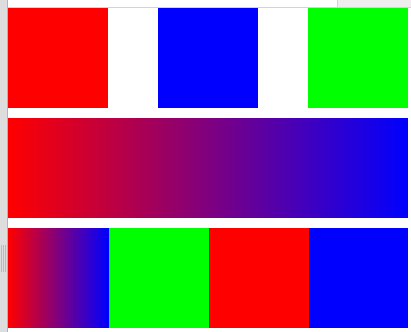
This screenshot shows the Color class working as intended at the top, the Gradient object working in the middle, (the gradient here has been multiplied by four, stretching the green part of the gradient out of the frame as intended) and the Gradient class with addition working at the bottom.

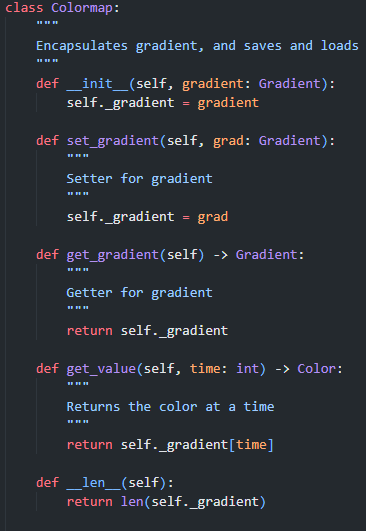
Figure 14: Incorrect test result

Figure 15: Correct test result

However, when testing in this way I also encountered an error, seen in fig 14. This error was caused by the \_\_getitem\_\_ function, which at the time contained this code snippet:



This code snipped used four Immediately Invoked Function Expressions (IIFYs) where the variable val was passed. Here, rel\_val should be passed instead. However, I also realised that the IIFYs here are unclear and would be harder to debug if I found further problems, so I changed the expressions to execute inline instead, fixing the error.



I then coded the Colormap class:

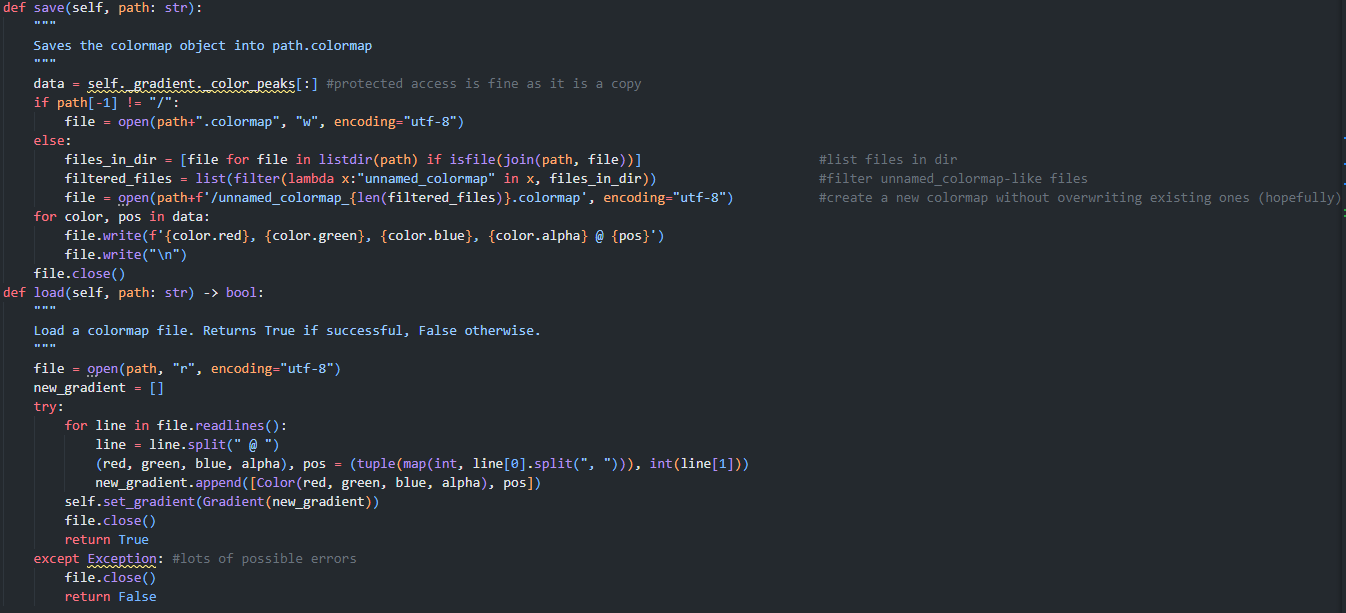
The Colormap class encapsulates the Gradient class. This is done to save and load a parent of the Gradient, as well as for passing by reference.

It has 7 methods, starting with the \_\_init\_\_ constructor. This takes a Gradient instance as a parameter, which it then saves as a protected instance variable for encapsulation. The set\_gradient and get\_gradient methods are setters and getters for this protected instance variable.

The get\_value method encapsulates the \_\_get\_item\_\_ function of Gradient, returning the color at the parameter.

The \_\_len\_\_ function encapsulates the \_\_len\_\_ function of the gradient

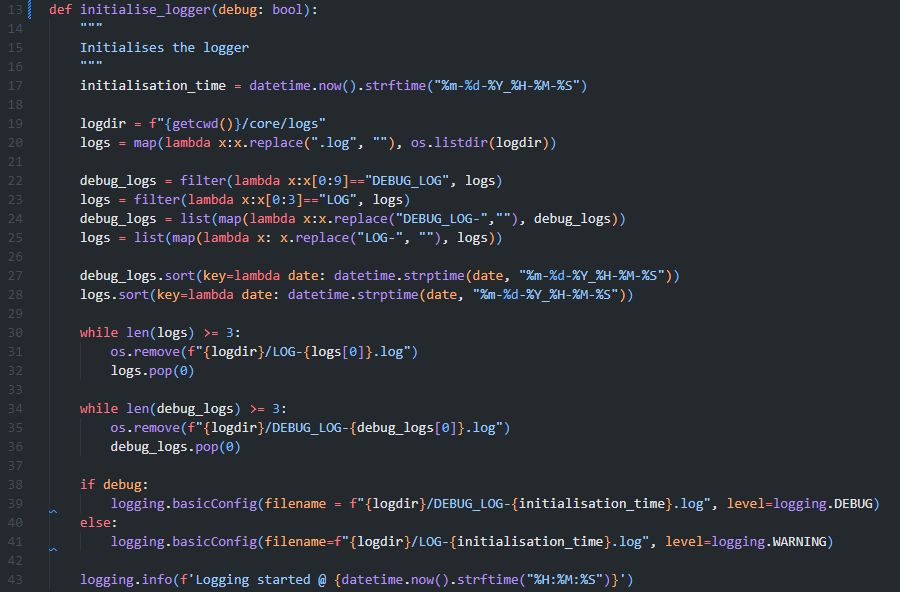
Also in this class are the saving and loading functions for gradients. These use my own filetype, called .colormap files, to avoid end user confusion between files.

 The save function makes a copy of the protected color\_peaks of its gradient. This is okay as the actual data is copied, rather than passed by reference, meaning no data can be altered in the client class. The program checks if a filename was specified or just a directory, then if a filename was specified, creates that file with UTF-8 encoding. If a directory is passed as an argument, the program generates a filename of the format unnamed\_colormap\_N where N is a number. This is done to prevent overwriting existing colormaps in that directory. Then, it writes the color peaks data to the file and closes it

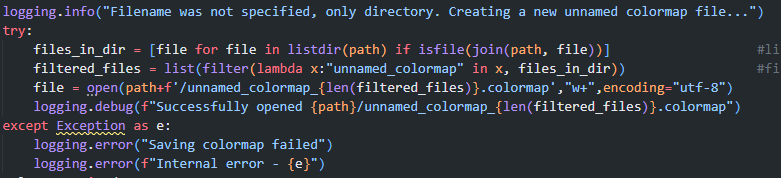
The load function opens the file at the specified path, iterates over each line, reconstructing the saved color\_peaks list. If this function errors, it returns False as it was unable to load from the path. This could be due to malformed input file, so should not error and break the program, hence the try catch. The color peaks recovered from the file is set as self.\_gradient.

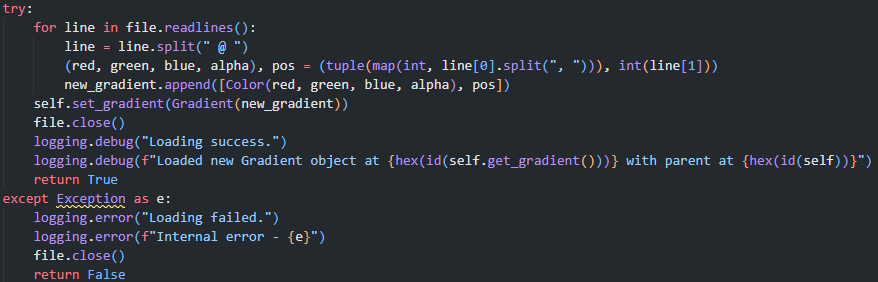
## Part 2: Logging and file structure

Around this point, I realised a good, reliable logging system for debugging errors and for maintenance of the program. First, I wrote code to initialise and configure the logging module:

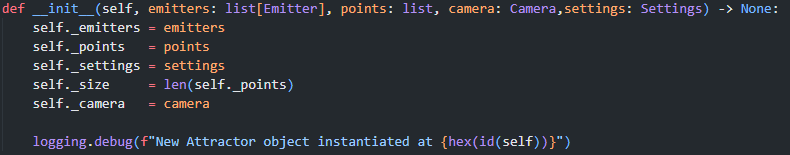
This code finds the initialisation time and date in a format that is valid for filenames, then gets the Log directory. It then uses some string parsing and sorting in order to sort existing debug logs and standard logs by the time of their creation. It then deletes the oldest log if there are greater than 3 or 3 logs in the folder, until there are only 2 logs in the folder. This means the new log generated is the third log in the folder of that type. It then creates a new log with the correct level based on the debug parameter.

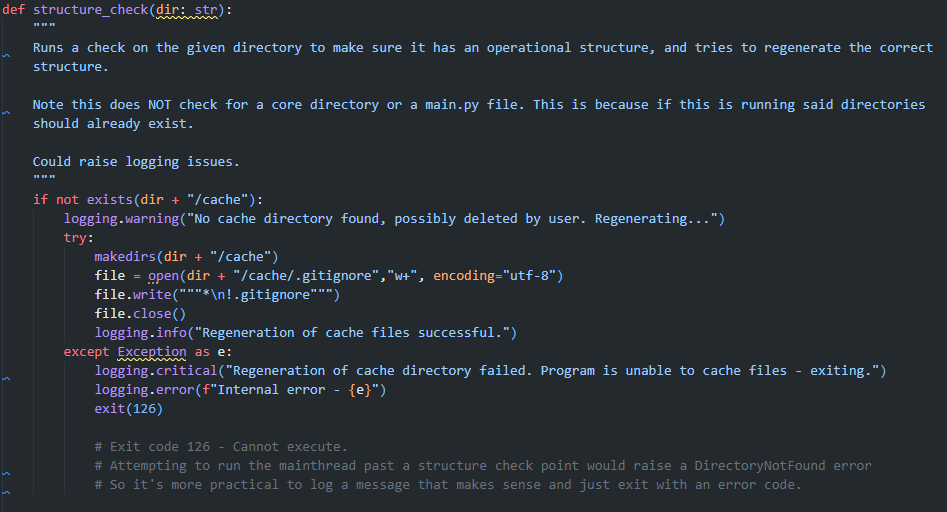
I then added logging statements to all of the code I had written where I thought appropriate. Here are some examples:

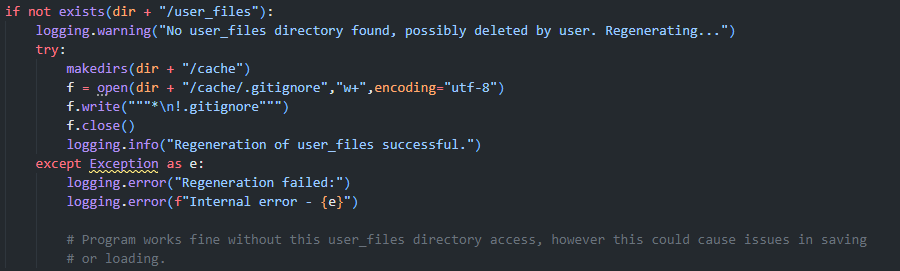
Logging for the creation of unnamed colormaps



Logging for saving colormaps

And logging for instantiation

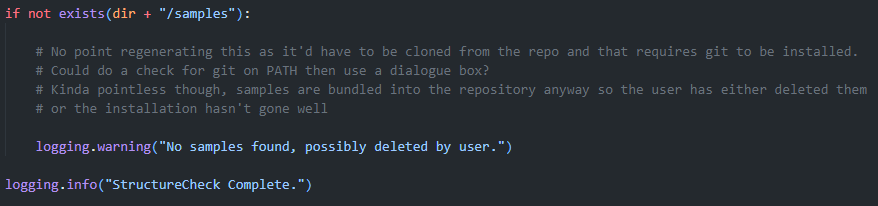
I also realised that this code would cause issues if the directory /core/logs did not have write/read access or did not exist. As such I wrote a function to make sure all necessary directories were able to be used:

This code first checks if the cache directory exists. If it does not, It tries to regenerate this directory and it’s gitignore. It raises a critical error if it cannot do so as the program would not function unless it is able to cache certain objects such as colormaps. It exits with code 126. This is also explained in a comment.

The function also checks the user files directory. It does not raise a critical error or exit here as the user will simply have to specify the save location rather than use the default one.

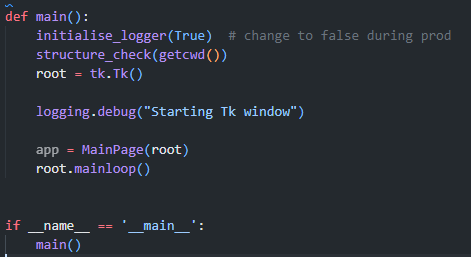


WRITE THIS UP AFTER IVE FINISHED IT

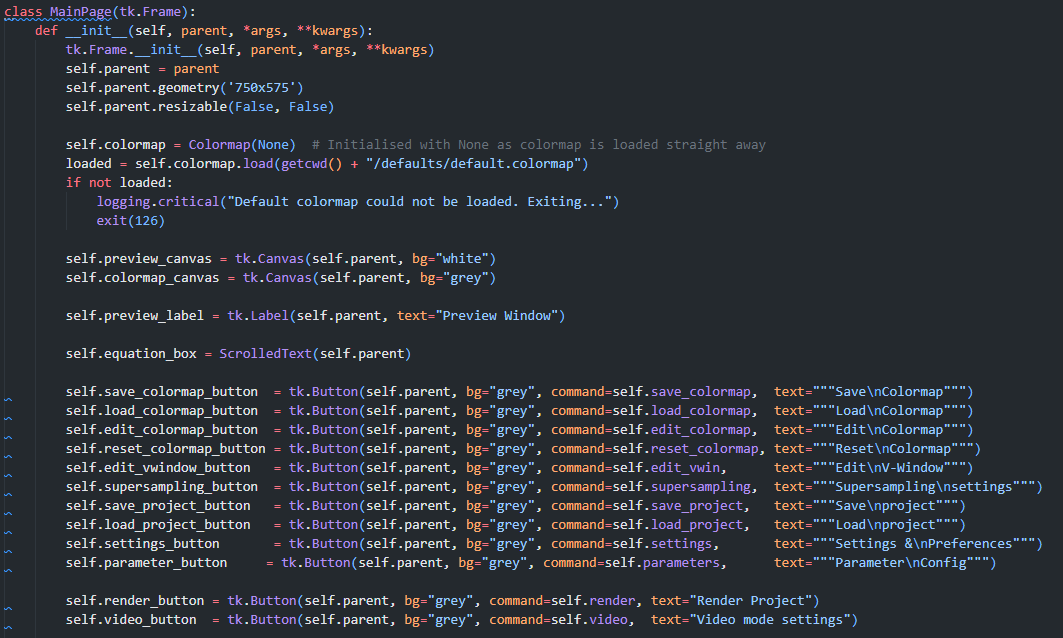
Finally, the program logs a warning if there is no samples directory. I determined that it was pointless to host these files to regenerate them or to try and clone them from the git source, so just a warning is logged.

## Part 3: User Interface.

At this point, I coded the user interface based on the design I set out earlier. I did this before continuing as it allowed me to test the backend of the program.

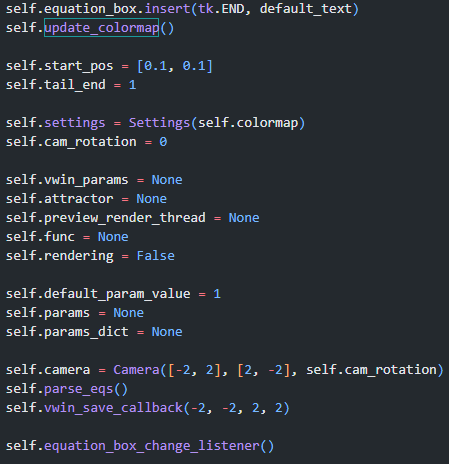


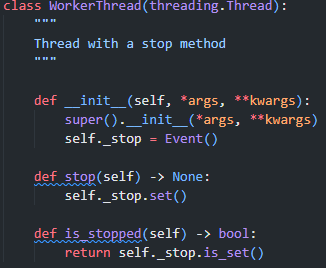
This code starts the program. If the thread is the main thread, it initialises the logger, checks the structure and creates a root object. It then creates an app using the MainPage class, and beings the loop.

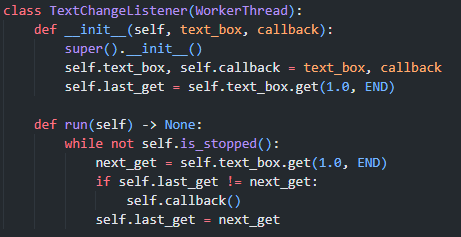
This is the code for initialising the MainPage object. It initialises its parent Frame object (which is always a tk.Tk) and sets its geometry. It loads the default colormap, and exits if it cannot be loaded.

Then, the UI attribute definitions begin. These lines show the created of all of the UI elements found on the root page. This includes the colormap and preview canvases, the label for the preview box, the equation box and all of the buttons. The buttons are initialised with callbacks to the class methods.

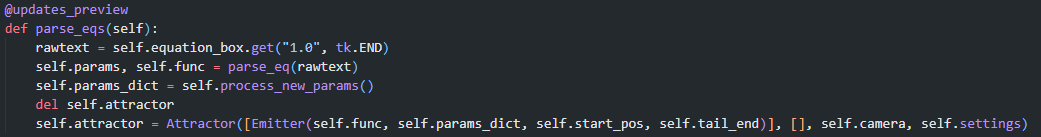
Next, the UI elements are placed based on the X, Y, width and height. The program then attempts to open the default equation text file, and inserts it into the equation box. The opening is done in a try-catch loop as I had an issue where if the user deleted the defaults, the code would not run. Instead, this falls back to a hardcoded default.

The snippet on the left initialises some important instance attributes, and calls some functions to update everything using the newly initialised instance attributes.





The code snippet at the top, containing equation\_box\_change\_listener required the use of a second class.This class, TextChangeListener, is a child class of WorkerThread. This class is used to overcome a significant problem I had with the text box - I would have to manually submit a new equation with a button, or check it regularly to see if it had changed, as there was no callback function in ScrolledText for when the contents changed. The WorkerThread class is a class I have used in the past, which creates a thread with a stop method using a Threading.Event(). Then, the TextChangeListener checks if the text in the box has changed, while the thread is not stopped. If the text has changed, it invokes the callback function passed to the TextChangeListener class (which is invoked asynchronously to the main GUI thread preventing it from hanging) and runs the method parse\_eqs.



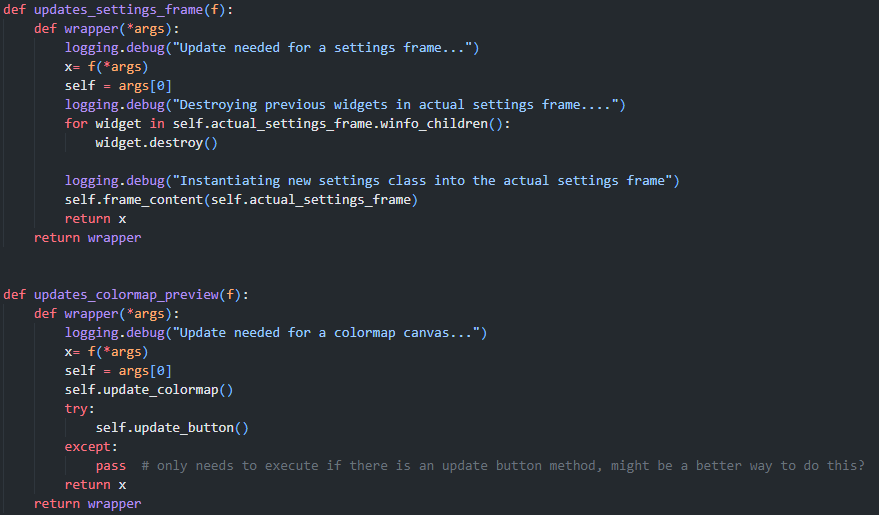
The parse\_eqs method of MainPage is used to parse the user text and create a new Emitter object and a new Attractor object with this new equation. At this point I was happy with the basic structure of the core classes, so I knew which arguments to pass into the classes to initialise them, but had not coded them. First it gets the text from the equation box, then it parses it and gets and processes the new parameters. For space efficiency reasons, it deletes the old attractor instance and creates a new one.

This class is decorated with the updates\_preview decorator:

This decorator is used for any MainPage methods that update the preview canvas. First, it invokes the wrapped function and stores the return value in x. Then, it grabs the self (which is a MainPage instance) pointer from the \*args of the wrapped method, and invokes the method start\_prevew\_render\_thread, which creates a new preview rendering thread. The reason this is used is it makes sure that the preview canvas is updated after the wrapped method’s call, so that any instance variables that update during the execution of the wrapped method are taken into account when the preview is re-rendered. The wrapper also returns the return value of the function. Originally, I had an error where in testing, using a wrapped method as a callback function would always return none (as I was not returning the wrapper pointer at the end of the decorator) and was only returning the return value of the function, which was never getting called. I fixed this after realising the mistake, but when creating the other required decorator functions I will make sure to include both return statements.

There are other ways to solve the updating problem, as mentioned in the design section, however after using this decorator I think I was justified in using decorators as a solution to my updating problem as they have created clear and readable code, and have allowed me to avoid errors where the canvas isn't updated when it should be.

Also note at this time, the start\_preview\_render \_thread method is equivalent to a pass statement, since I have not yet coded the Attractor object.

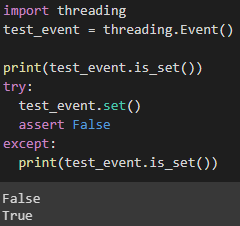
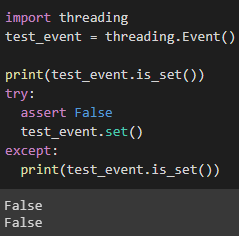
At this time, I decided it made sense to also create the other decorators I would need:

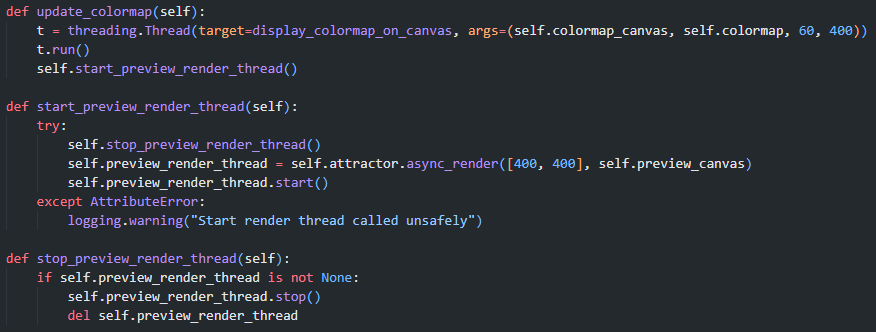
The updates\_settings\_frame decorator invokes the wrapped function, then destroys all widgets in the inner frame (actual\_settings\_frame, renamed for clarity as self.inner I felt made more sense for inner functions rather than frames) then instantiates the new settings class (for which frame\_content is a pointer to) with the inner frame as the parent.

The colormap preview updater works similarly to the preview updater, except that it has a try-catch loop for the method update\_button. This is because there is a button in the colormap window that needs updating, but not in the main window. This code means that the same decorator can be used in both instances, saving around 20 lines of code.

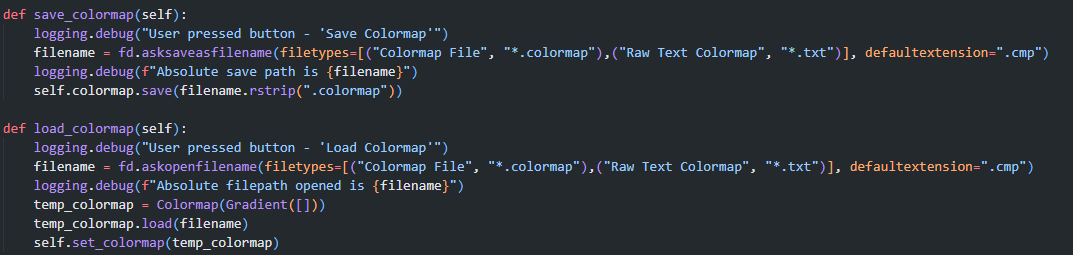
Next, I created these 3 functions. Though I had not coded display\_colormap\_on\_canvas or Attractor:async\_render, I knew how I wanted them to function and had pseudocode for them both, so I felt I could effectively create the methods used to invoke those functions. Firstly, update\_colormap is a simple method used to create a thread to display the instance attribute colormap onto the pages colormap preview canvas.

The start\_preview\_render\_thread works slightly differently, with a try-catch loop to avoid unsafe calls and uses a child class of my WorkerThread class. Based on previous experience, I knew that I could run into errors where self.attractor has not been defined yet, and trying to call the async\_render method of the attribute would result in an AttributeError. This is an unsafe call of the function, but can occur in many situations. One such instance is if a different part of the program raises a non-exit error that causes self.attractor to be deleted but not regenerated (i.e. in parse\_eqs where del self.attractor is called and then the attractor initialization is called). This could be avoided by making sure no functions can cause the class to be in this state, however this could mean that the program starts rendering the incorrect attractor, which could lead to the user being confused, or mistaking one attractor for another. This needs to be avoided as this could lead to a user such as Mark being unable to analyse an attractor effectively, or a less experienced user assuming they did something wrong when attempting to recreate the results of another user. When this function passes (by the catch section not invoking anything other than a logging function), nothing changes except that the render thread is stopped, clearing the preview canvas. This avoids the user confusion. Originally, I expected this code to not clear the canvas, and that I would have to call the stop function in the except clause. However, this test code shows this is not the case and I do not need to call the stop function in the except clause:

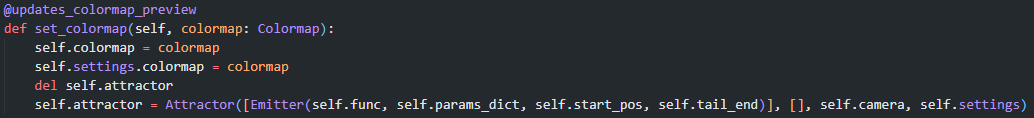


This shows that the asynchronous nature of Events means events set in a try loop will still be set if the try loop errors, unless the set is invoked after the exception is raised. The former is the case in my program, so the stop call is not required in the except clause.

The async\_render method of Attractor returns a RenderThread that can be started by the run method, and stopped by the stop method. The stop method stop\_preview\_render\_thread also deletes the old thread for space efficiency and to make sure that the thread frees up to avoid filling all of the CPU thread space with unnecessary threads.



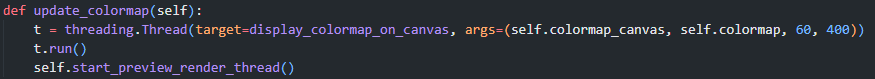
These two methods are responsible for saving and loading colormaps using the tk.FileDialogue module. They, respectively, open a save or load dialogue, invoke the save or load method on either the in use colormap for save functionality, or a blank one for the load colormap, then exit. Note that the self.set\_colormap method is used rather than simply setting the colormap. This is because the setter method is as followings:



This does not just update the colormap attribute of the MainPage instance, but also of the settings instance, and of the attractor instance. It is also decorated to update the colormap preview. Here was also where I encountered an interesting error with my colormap updating logic - updating the colormap also requires updating the preview window. As such I realised I had to change the update\_colormap method from this:

https://lh4.googleusercontent.com/PfmG7o6XlRl6Jt47ufiHF0USxe8pAajrpFIWA2aN3d9FFJO1ESByVquoPYoGYc7-Ulw4mg4GS2l1L36TG661Xw0oXX2VI7VFnlGFwiG9s7YyomojxZ7XuwQW1TKKLHX3OEEPKwXjoiOvbysBJIpyxJiUB3RLkPFlUZOY_hIKD0uszN-0ygQYs43EMCSrIA

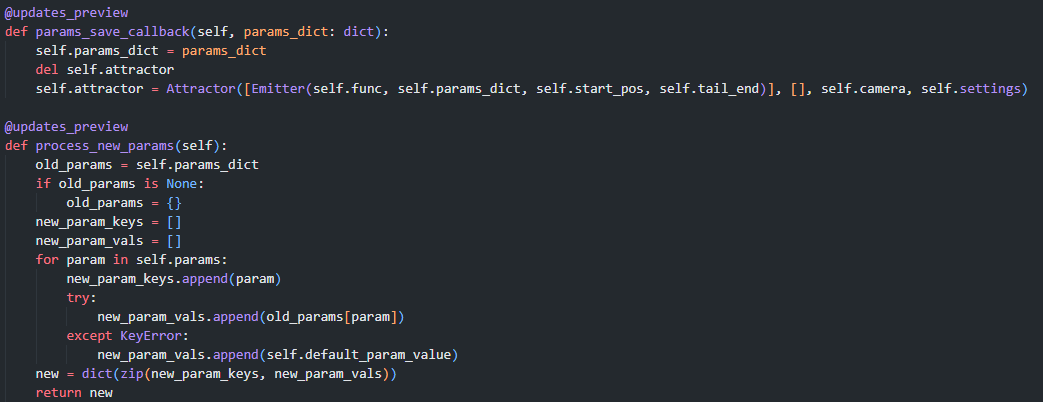
to



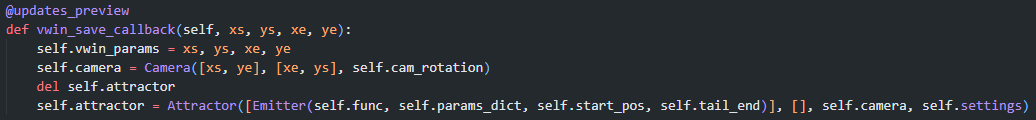
such that updating the colormap via the decorator would also update the preview canvas.



This code is used to reset the colormap. First, it creates a new temporary colormap which attempts to load the default. It logs an error if it could not be loaded, and if not invokes the colormap setter to set the colormap as the default.

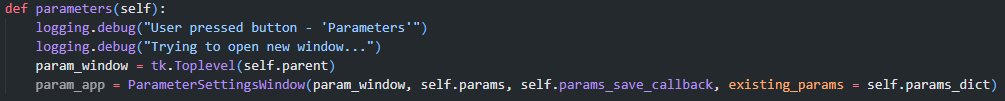
This code snipped includes two methods – params\_save\_callback, which functions as a simple callback to set an instance attribute and regenerate the attractor attribute based on that new instance attribute, and process\_new\_params. Process\_new\_params is only called when new parmeters are loaded into self.params but not self.params\_dict. This method is used to carry over values of previous parameters.

There is good reason for this code – during testing of the UI I found that it was cumbersome for the user to have their parameters reset if they slightly change an equation – which is what happened before I implemented this method. Now, if a user slightly changes the equation, implements a new parameter or deletes a parameter, the parameter values are carried over from the last saved values.

Similarly:

The vwin\_save\_callback has the same purpose, but for saving view window information into the camera attribute through use of a Camera object

The code for opening windows follows the same format as this methods:



This code shows a method that instantiates the ParameterSettingsWindow in order to open the window that class represents. It creates a new TopLevel object and passes this as the parent of a ParameterSettingsWindow instance.

It also passes relevant arguments and keyword arguments, as required per class. Some other examples can be found throughout the MainPage class, as multiple buttons cause windows to be opened:

