Algorithmic Map Recognition and Edge Detection with Point to Point Pathfinding

Computer Science NEA

Name: Rubens Pirie

Candidate Number: 1749
Centre Number: 58231

Centre Name: Barton Peveril Sixth Form College

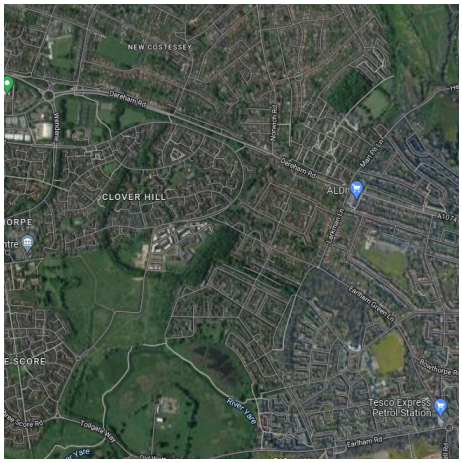
Contents

1 Analysis

1.1 Statement Of Problem

Maps, as you would think of them today, have been around since 6th century BC and since then have been in constant use by people in their day to day lives. The more modern version of maps, for example Google maps or Bing maps have only been around since the late 1990's. The problem that I am going to be solving is map path finding. Currently not all roads and paths are logged and entered into a searchable format. The only way some people have to navigate terrain is through the use of old style paper maps. The problem with paper maps is that they are not easily, at a glance, used to find a path from point to point. As well as this sometimes are not easy to comprehend just by looking at them with various terrain features.

(1) Map without labels on roads



(2) Map with labels on roads

Examples of maps with and without labels taken from Google Maps©

This can cause issues for people who live out in areas which have not been mapped. This is because they cannot create easy to follow routes with the click of a button. Therefor, causing people who live in rural areas to waste time getting used to the routes they have to take to go anywhere. Overall, the problem I am going to be creating a solution for is how people are unable to easily go from point to point at the click of a button and be easily able to, at a glance, interpret the map without prior experience.

1.2 Background

When people usually want to go about planning a journey they will use a service, for example Google Maps to get from one location to another. This usually takes the form of clicking a location and then selecting an origin. This isn't always possible however, this can be for a multitude of reasons it seems however I will briefly go over some below:

1. **Either the destination or origin location(s) are not in the service's database.**
2. **The destination and origin have no clear defined path between them.**
3. **Either the destination or origin are off any predefined track.**
4. **The travel method the user has selected is not able to traverse the terrain between the origin and destination.**

Some of these I believe are out of the scope of this project however once the interview has been conducted with the end user I will have a better idea of the needs that my program needs to for-fill.

Finally, I feel that the point of my final solution should be to fix all of the flaws which I find during my research as well as from the end user. As well as improving where the end user feels it needs to be.

1.3 End User

1.3.1 First Interview

In order to get a better feel for the objectives and functions that my program should complete I interviewed with an end user, Mrs Mandy T. I believed that she was an appropriate candidate for this project due to the fact that she has to drive into work every morning. Along her route she has to deal with Google Maps which do not cover all of the roads in her area. Therefore in the following questions I asked her some questions gauge her priorities when it comes to web mapping.

1. **When using web maps (e.g. Google Maps©) what are the key features you look for?**

   "A scale! WHY is it lost so often when Google Maps is embedded?! Then it depends what type of map I'm looking at... if it's a road map then....roads! Size/type of road is important and things like one-way restrictions. If it's for e.g. walking...footpaths/bridleways and parking are important. "

2. **Have you ever experienced a faulty or mislabeled part of an web map or has said map ever been inaccurate?**

   "Yes"

3. **Do you often use web maps in your day to day life, if so in what capacity?**

   "Yes, I tend to use them when travelling around to places that I have not been to recently or at all. They can be very handy if I need to navigate to somewhere that I have never been too and there are multiple ways to get there with no obvious faster route."

4. **In your opinion do you feel that web maps are vital to every day life if so why or why not?**

   "No. I passed my driving test before we had sat-nav or internet, so clearly they're not vital - we survived without them!
   They are quite helpful though as we used to have to buy a new road map every year, whereas web maps can be updated as things change, instead of only annually!"

5. **What makes a good user interface for a web map?**

   "Clarity and simplicity. Nothing needlessly complicated."

6. **How do you use web maps (e.g. long journeys, short journeys, school runs)?**

   "Route functionality on long or unfamiliar journeys. Using them a lot at the moment as am planning a holiday overseas. The maps are useful to see whether accommodation and restaurants will be walking distance, and what options there are in each location etc."

7. **Do you feel a tutorial would be beneficial to aid in the use of the map or should the focus more be spent on intuitive ease of use?**

   "If they're easy to use, a tutorial would be surplus to requirements, so ease of use is more important. "

8. **Would it be beneficial to store old routes?**

"Not really (is this a routing question?). I don't know what purpose that would give, unless I was being accused of something and needed to use the route as evidence of being in a certain location! It could be use full in the context of frequently traveled routes however if this was the case I would know the route by heart anyway."

9. **What forms of transport should the map include?**

   "(I think this is a routing question not a map question) Walking, bike/horse, car, bus, plane, ferry. If just a map question, then the map should include footpaths, bridle paths, roads, ferry routes"

10. **If there was one feature you could have implemented in an existing solution what would it be?**

    "To be able to post a question about a specific area and have a person who is local to that area answer it."

1.3.2 Evaluation of First Interview

Overall I feel that this interview gave me valuable insight into the requirements of my end user. As well as this my end user made it clear to me that there are two overriding parts of this solution. The map recognition aspect of it and the path finding aspect. Going deeper into the path finding part of this project I will need to do research on the different methods that will be used to achieve this and some of the possible data structures I could use.

1.4 Initial Research

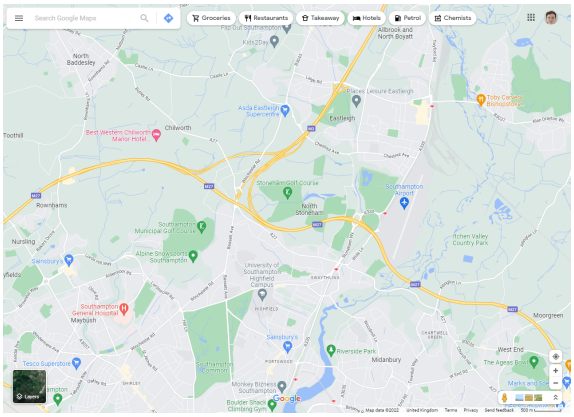
1.4.1 Existing Solutions

Below each overview passage I have included an image of each map for comparison of their GUI's. These will be used as inspiration as to how my final solution will look as well as serving as examples of how the GUI can sometimes become overly complicated. This is especially the case with Bing Maps as when you initially access it you are flooded with popups and extra options.

Google Maps

As aforementioned this is one of the most used forms of interactive web mapping in use at the moment. It has been in use since 8th February 2005. As it exists now it is an interactive world map with routing features built in. It provides detailed information about geographical places and regions around the world. Unlike some of its competitors it also offers aerial and satellite images of places around the world aiding in navigation of terrain.

As well as its map viewing capabilities it also offers partial route planning and live route tracking for cars, bikes, walkers and public transport. It provides instantaneous and real time feedback while you are moving however the one big caveat to this is the fact that it will require an internet connection to run, something that is not always available.

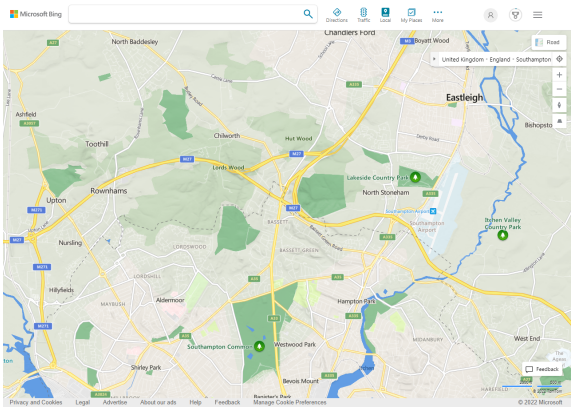


(1) Example of Google Maps' GUI

Sourced from Google Maps©

Bing Maps
This is another form of interactive web mapping. This is a more plain version of Google Maps at first glace. This is due to the fact that it does not have as many features as Google Maps. This does have its advantages due to the UI seeming less cluttered and more accessible. Similar to the Google Maps it also offers route planning and map traversal as well as live traffic updating. Bing maps unlike Google Maps boasts a more open API and easier programmatic interface for developers to be able to interface with their program.

Bing maps also still includes the feature which allows users to create their own maps based on their own data. Unlike google which did have this feature until they discontinued it. I believe that this could be something that would be beneficial to my program, allowing people to take a photo of their own map and have my solution compute it into a rotatable map.

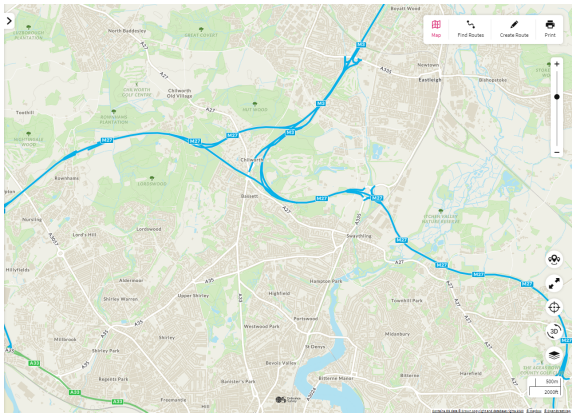


(2) Example of Bing Maps' GUI

Sourced from Bing Maps©

OS Maps
This is a different take in web mapping compared to Bing and Google Maps. With Ordnance Survey their focus was on the accuracy of their maps hence they do not have as an extensive routing system. If you wanted to go from point to point on an OS map you would have to plot it by hand. However if you wanted to go on an exercise trail on the other hand they are very well suited for this and as such have an extensive list of pre-planned routes.

Similar to Google Maps, and in a limited capacity, Bing maps; OS Maps allow you to view their maps in different forms such as 3D and topographic however in order to access these you will have to access their premium plan therefore for the average user this is not a viable option and a hindrance. It is good to note however that the other variations on the map of the UK, and this holds true for all of the aforementioned maps, that the satellite view and other views are not necessary and could in fact be a hindrance.



(3) Example of Ordnance Survey's Map GUI

Sourced from OSMaps.com©

Existing Solutions Conclusion
In conclusion, I have found that the existing solutions that are available are all very well designed and well implemented. I have found that they are easy to use and rather intuitive however, for the average user who just needs to get from A to B in the most economic way possible they are overly complicated. As well as this I have found that with the exception of OS maps both of the other solutions require an internet connection to get the best use out of their maps, this is something which I believe I should avoid. This will mean that all calculations will have to occur self contained within the program, not allowing the use of external API's.

1.4.2 Possible Algorithmic Solutions

There are, as aforementioned many existing solutions which work in various ways, in order to make my solution unique and functioning I am going to have to incorporate many different algorithms and theories.

Edge Detection
First of all I will need some way of recognising a map and parsing it in some way. The way that first springs to mind is edge detection. This is a way of taking an image and computing where there are changes in contrast or brightness which could be considered an edge. There are many forms of edge detection out there all of which work in various ways, the main things they look for however are discontinuities in depth, discontinuities in surface orientation, changes in material properties and variations in scene illumination. All of these factors combine and allow a program to decide if there is an edge in an image.

A simple edge detection model can be extremely effected by natural blur or artifacts in an image. In order to mitigate this there are smoothing algorithms that can be used to blur and smooth edges causing the impact of artifacts to be avoided. The common term when referring to artifacts and erroneous data in an image is *noise*. I believe it will be beneficial to include some of these in my solution, this will be something to look into in the **Further Research** section.

Taking a quick look at one form of edge detection, Canny edge detection, it is relatively simple in its implementation. It has only 5 steps, first removing noise with a Gaussian filter then applying bounding to the image and finally performing hysteresis threshold. This is the most common form of edge detection that I have come across in my research however there are others. A rather different example of edge detection is Kovalevsky edge detection. Unlike Canny edge detection this does not care about the luminosity of the image and goes based of the colour intensity in each of the channels.

Graph Forming
This is not so much a possible algorithmic solution but more of something that my solution will have to achieve. Once the image of the map has been altered and the edge detection has been performed, I will be left with an image which has white lines where there "edges". From this I will need to create a weighted graph as well as an unweighted graph.

During my research I have failed to come across an existing solution to this problem. As well as this looking through some examples that people have uploaded it seems that sometimes the edge detection does not yield a fully connected image. This could prove to be an issue as it would add the possibility of isolating certain roads.

I feel that I need to look more into this and come up with my own solution during the prototyping stage, and come up with my own algorithmic way of generating it.

1.4.3 Key Components Required

After doing my initial research and a brief look at the existing solutions I have come up with, what I feel, is the main 4 Components that I will need to build my solution.

The Graphical User Interface
Talking to my end user made it clear to me that in order for the program to be usable by the wider population it would need to be clean and uncluttered. This leaves me in a difficult position due to there being a limited amount of frameworks that are available to me. I have two sets of possibilities:

1. A Local App Run on Device
2. A Web Based Application

Each of these have their advantages, if I where to go with a locally run app I could make it in the console keeping it simplistic and easy to use. However if I do use the console it would limit this solution to a computer which could be seen as going against the idea of this problem. On the other hand, if I where to go with a web server based application this would yield much better compatibility with all devices since all you would need is access to a web browser. This, by its very nature, means that you would need an internet connection which is also a problem which I was hoping to fix.

The solution then I believe is to make it both a locally based program with the option for it to run a web server. However I will need to specify one over the other to begin with to make sure that the program is working either way.

Regardless of which one I choose I will conduct some form of testing where I will allow, through a survey, people to specify what makes an easy to use and intuitive.

Image Manipulation and Edge Detection
This is perhaps the most important part of the project since without this I would not be able to continue to path find the image of the map. Looking at my research I feel that there will be a combination of

Graph Creation and Representation
From lessons which we have had in class I have been shown that there are 2 reasonable ways of representing a graph in code, this includes an adjacency matrix and an adjacency list. Both have their advantages and disadvantages. An adjacency matrix is good when you have a reasonably connected graph which has weights, this is due to it being easy to access and traverse. As soon as you have a sparse graph however it becomes very memory intensive which is unnecessary considering that there will be very few of the cells with actual data in them. This is when the adjacency list comes into play, the reason that I am reluctant to use this form of representing a graph is that when performing some of the various graph traversal algorithms it can incredibly difficult and pointless to adapt them when by adapting them you effectively generate the adjacency matrix.

Graph Traversal and Output

1.5 Further Research

1.5.1 Dive into Specific Algorithms

After doing some research it seems that there needs to be a set of definitions before I go any further to avoid confusion. This is because during my time on Wikipedia there are sections where several terms are used interchangeable where I feel they are not the same. Each of these definitions are as defined by me and are not necessarily the official definitions since they do not explicitly exist. They are as follows:

1. Graph Traversal: The act of routing or searching through a graph from one node to another, either using an algorithm or by another means.
2. Graph Routing: Graph traversal in a *weighted undirected* graph.
3. Graph Searching: Graph traversal in a *unweighted undirected* graph.

The difference is slight however the key takeaway from this is that when I am referring to a Routing algorithm I am referring to one which works on a weighted graph. And vice versa if I am talking about a searching algorithm this is referring to graph traversal on an unweighted graph.

Black and White Filter
In order to allow the program to function, assuming that the Canny edge detection was chosen we do not need the colour data of the image. In order to remove this a filter is used, this one is the industry standard since it takes into account how prevalent red, green and blue are rather than taking an average which could become non representing of the real case.

$$\beta = 0.299 * \alpha_b + 0.587 * \alpha_g + 0.114 * \alpha_b; \begin{cases} 255 & \beta > 255 \\ 0 & \beta < 0 \\ \beta & \beta \in [0, 255] \end{cases}$$

If an averaging was used it would just be, this is also known colloquially as the "quick and dirty" method.

$$\beta = \frac{(\alpha_b + \alpha_g + \alpha_b)}{3}$$

Gaussian Filter
This is the first step of 5 in terms of performing Canny Edge Detection. Applying the Gaussian filter to the image will smooth out the image and remove any noise. It does this by taking a section of the image, sometimes referred to as a kernel and performing an equation on it. Once it has computed the equation it sets all of the pixels inside the kernel to this value. The following is true for a kernel size of $(2k+1)*(2k+1)$. It takes two changeable parameters σ which denotes the amount of blur to apply and k is the kernel size. As well as being one of the key steps in Canny edge detection it is also a vital component to most edge detection programs since noise can cause errors in the final image.

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(- \frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2} \right); 1 \leq i, j \leq (2k+1)$$

Since the Gaussian kernel I would be using would always be centred around the origin $(0,0)$ I can use a simplified version of the Gaussian distribution equation. This is as follows:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp \frac{-(x^2+y^2)}{\sigma^2}$$

I can afford to remove the $(i-(k+1))$ section due to the fact that I am not having to calculate the Gaussian distribution at a non-centred location. One notable thing to mention is that in many cases it is not necessary to calculate the Gaussian kernel by hand and an approximation can be used. The example below is the approximation when σ has a value of 1.

$$B = \frac{1}{159} \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix} * A$$

Convolution Operation
Convolution is the method at which most image manipulation is achieved. It evolves taking a altering kernel and a kernel of the original image and then combines the two through convolution. The generalised equation for this is as follows.

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix} * \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{pmatrix} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{(m-i)(n-j)} y_{(1+i)(1+j)}$$

To give a more comprehensive example this can be simplified down to:

$$\left(\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} * \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}\right)$$

$$\rightarrow (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9)$$

The simplest way of thinking of this is that you are performing matrix multiplication on a two matrices except one of them has been flipped both vertically and horizontally. Mapping the point [2, 2] to [0, 0].

1.6 Prototyping

1.6.1 Prototype Objectives

Before I begin the creation of my prototypes I will create a list of sections I wish to complete by the end. This will allow me to keep perspective and make sure that the prototype remains on track. I have decided that the parts of my final solution are:

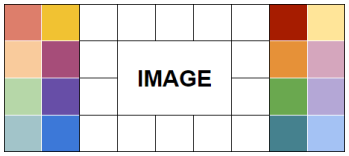
- A version of edge detection
- A graph class with basic traversal
- A forms interface for showing images

1.6.2 Edge Detection

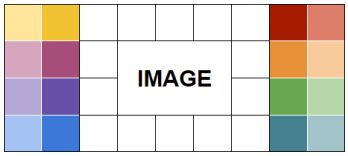
For the example of edge detection which I am going to prototype I have chosen Canny Edge Detection, this is the most common of the types of edge detection and is relatively simple. It is also widely documented which allows me to focus more on the application and less on the finding of resources.

Before I begin, there are a couple of key features that need to be mentioned. The first is how I handle building the image kernel. For example when the center pixel is on the edge of the image, you will have some non existent pixels as part of the image kernel. To combat this there are several methods:

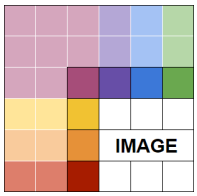
1. Extension - The nearest border pixels to the chosen pixel are extended in order to fill the gaps. The corner pixels are extended at 90 deg. Others are extended in straight lines.
2. Wrapping - The pixels for the unknown ones are taken from the opposite side of the image. For example it it was 1 off the top the first pixel from the bottom would be used.
3. Mirroring - The image is mirrored at the edges doubling up the total image.
4. Constants - Any pixels in the kernel which are not contained in the image are given a default value, this is usually grey or black depending on the application.
5. Duplication - Similar to above any pixels which are not contained are set to the value of the center pixel in the kernel.



(4) Wrapping

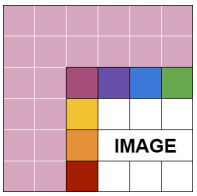


(5) Mirroring



(6) Extension

(7) Constants



(8) Duplication

For this part of the prototype I have decide to go with the duplication option, this is due to the fact that it is one of the easier and quicker methods to implement as well as being suitable for the edge detection use case.

Figure 1: Original Image

1. Converting to Black and White
The first part of the edge detection is to convert the image to black and white. This is because if the image is in colour then you would have to either perform edge detection on each of the colour sections and then somehow combine them, or take a single colour value to base the conversion off of. As previously mentioned this can be accomplished through many means, the most common as explained in *1.5.1 Black and White Filter*. The version which I have decided to use for this prototype is the industry standard Y'UV conversion.

The implementation in code of this is as below:

```
1  public double[,] BWFilter(Bitmap image)
2  {
3      double[,] result = new double[image.Height, image.Width];
4
5      for (int i = 0; i < image.Height; i++)
6      {
7          for (int j = 0; j < image.Width; j++)
8          {
9              Color c = image.GetPixel(j, i);
10             double value = c.R * 0.299 + c.G * 0.587 + c.B * 0.114;
```

```
11
12              result[i, j] = Bound(0, 255, value);
13          }
14      }
15
16      return result;
17  }
```

This takes the original image in Bitmap form and then instantiates an array with the dimensions of the input image, this will serve going forward as the array as to which all changes will be based from. I learnt from this prototype early on that when calculating the values is is better to use the exact ones from the previous stage. This is because if all the values where compressed to within image specifications ($0 \leq x \leq 25$) you would loose definition and precision causing later calculation so be incorrect. Once this section has run through every pixel in the image and converted it to a black and white value the subroutine returns the double array with the black and white values. The result of this on the input *figure 1* is:

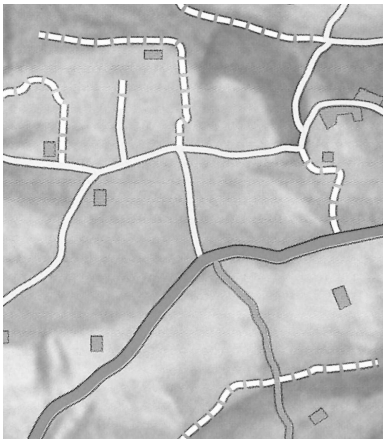


Figure 2: Black and White Filter

2. Gaussian Filter

The next step of Canny edge detection is applying the Gaussian filter. This is to ensure that any noise that is contained within the image is removed. This is because if there are stray pixels in the center of the image this can cause an edge to form when in fact there isn't one. This is the first operation in edge detection which requires convolution as explained in *1.5.1 Gaussian Filter*. To accomplish this the following code was used:

```
1   public double[,] GaussianFilter(double sigma, int kernelSize, double[,] imageArray)
2   {
3       double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];
4
5       Matrix gaussianKernel = GetGaussianKernel(kernelSize, sigma);
6
7       for (int i = 0; i < result.GetLength(0); i++)
8       {
9           for (int j = 0; j < result.GetLength(1); j++)
10          {
11              Matrix imageKernel = BuildKernel(j, i, kernelSize, imageArray);
12              double sum = Matrix.Convolution(imageKernel, gaussianKernel);
13              result[i, j] = sum;
14          }
15      }
```

```
16
17      return result;
18  }
19
20  public Matrix GetGaussianKernel(int k, double sigma)
21  {
22      double[,] result = new double[k, k];
23      int halfK = k / 2;
24
25      double sum = 0;
26
27      int cntY = -halfK;
28      for (int i = 0; i < k; i++)
29      {
30          int cntX = -halfK;
31          for (int j = 0; j < k; j++)
32          {
33              result[halfK + cntY, halfK + cntX] = GetGaussianDistribution(cntX, cntY, sigma);
34              sum += result[halfK + cntY, halfK + cntX];
35              cntX++;
36          }
37          cntY++;
38      }
39
40      for (int i = 0; i < k; i++) for (int j = 0; j < k; j++) result[i, j] /= sum;
41      return new Matrix(result);
42  }
```

Again this subroutine follows a similar layout to the rest in this prototype, it iterates though each pixel in the image and apply some equation. In this case as stated above it is performing convolution of a matrix which is a sub section of the original image. It is convoluting this with the Gaussian kernel though the means described in *1.5.1 Convolution Operation*. The code for the convolution operation can be seen at *5.1.1 Lines 586 through 612* and the Gaussian distribution lambda function can be found *5.1.1 Line 554*. Another learning experience here was how if the image is sufficiently large then the kernel does not have as much of an effect at blurring the image and removing noise. It may be beneficial in the final program to reduce the image to a smaller size or perhaps change the sigma and kernel size. The output of this subroutine is:

Figure 3: Gaussian Filter

3. Calculation of XY Gradients

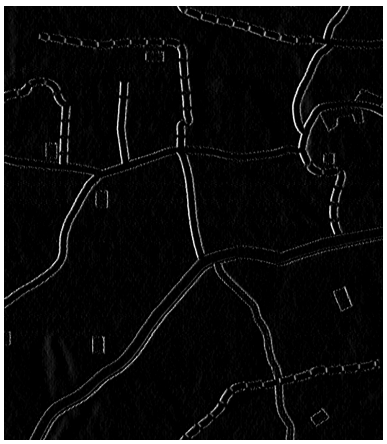
The first edge picking stage of Canny edge detection is the calculation of the gradients of the image in both the X axis and the Y axis. In order to achieve this two more kernels are used. They are known as the Sobel operators.

$$M_y = \begin{pmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad M_x = \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

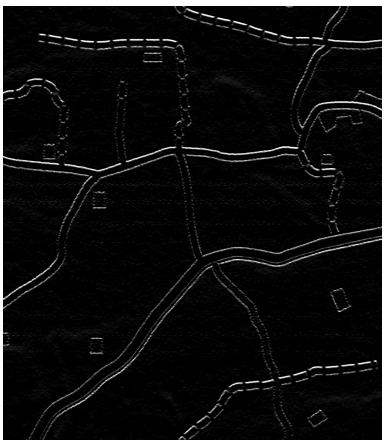
The code which is used to perform this section of the Canny edge detection is as follows, note that for the gradient in Y the matrix is replaced with the Y matrix and its code can be seen at *5.1.1 Lines 416 through 432*.

```
1  public double[,] CalculateGradientX(double[,] imageArray)
2  {
3      double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];
4
5      Matrix sobelX = new Matrix(new double[,] {
6          { 1, 2, 1 },
7          { 0, 0, 0 },
8          { -1, -2, -1 },
9      });
10
11     for (int i = 0; i < imageArray.GetLength(0); i++)
12     {
13         for (int j = 0; j < imageArray.GetLength(1); j++)
14         {
15             Matrix imageKernel = BuildKernel(j, i, 3, imageArray);
16             result[i, j] = Matrix.Convolution(imageKernel, sobelX);
17         }
18     }
19
20     return result;
21 }
```

Same as the Gaussian filter the convolution operation is applied to both of these matrices. The kernels that are used are build from the image with the center (i, j) same as the previous step. This is when it becomes beneficial to use the duplication method for the kernel building. Since the gradient is dependent on the surrounding pixels using the pixel itself prevents false edges from appearing. The two separate gradient kernels produce the following images:



(1) Gradient in X



(2) Gradient in Y

These two images represent the cases where in the image there is a change in the value of the pixels. The brighter the white the more different two given pixels are. We can combine these two to give a total image of all gradient changes. Find image below, while this is useful to look at from a human perspective it is not the most useful in edge detection and in fact we will need both the raw 2D double arrays from each gradient calculation to move onto the next step.

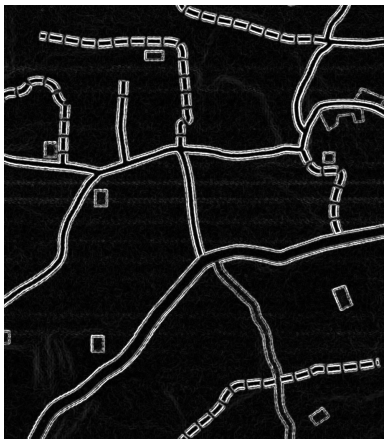


Figure 4: Gaussian Filter

4. Gradient Direction
Now that the gradient values have been calculated we can move onto working out which direction the gradient is travelling. This is done via the use of the 2^{nd} argument arc-tangent. The definition of the 2^{nd} argument arc-tangent is defined as the angle in the Euclidean plane, given in radians, between the positive x axis and the ray from the origin to the point (x, y). Once this is calculated this will allow the program to see in which direction the gradient is travelling in the image. As well as this is also allows us to see how sharp the change is from one to the other, this is how we can decide if there is an edge there. The code to calculate the 2^{nd} argument arc-tangent is simple since all is needed is to iterate over the entire image. The code for this can be seen at *5.1.1 Lines 379 through 384.*

```
1 public double[,] CalculateTheta(double[,] gradX, double[,] gradY)
2 {
3     double[,] result = new double[gradX.GetLength(0), gradX.GetLength(1)];
4     for (int i = 0; i < gradX.GetLength(0); i++) for (int j = 0; j < gradX.GetLength(1); j++) result[i, j] =
  ↪  Math.Atan2(gradY[i, j], gradX[i, j]);
5     return result;
6 }
```

This however will return an array with values which are in the range of -π to π therefore in order to create an image to visualise the result a linear transformation must be used which can be calculated as the equation of a line. The derived equation is $\frac{128}{2\pi}x + 128$ where x is the value of theta. Once converted the output of this stage is as follows.

Figure 5: Gaussian Filter

5. Gradient Magnitude Threshold
Once both the combined gradient and gradient directions have been calculate the next step in the process is working out which parts of the edge detected image are noise and which are not. In order to do this the combined gradients and the direction are taken into account and similar to before we build a kernel of the surrounding pixels of the image. The first part of this however is to convert the values in radians to values in degrees, to do this we run all through all values and convert them first. This can be seen *Lines 372 through 377*.

```
1  public double[,] ConvertThetaToDegrees(double[,] thetaArray)
2  {
3      double[,] result = new double[thetaArray.GetLength(0), thetaArray.GetLength(1)];
4      for (int i = 0; i < thetaArray.GetLength(0); i++) for (int j = 0; j < thetaArray.GetLength(1); j++) result[i,
   ↪  j] = 180 * Math.Abs(thetaArray[i, j]) / Math.PI;
5      return result;
6  }
```

Once all values are in degrees this becomes easier to deal with since there is less data lost to floating point arithmetic. Now that the angles are in degrees they are compared to predefined values as shown in the code. Depending which if the categories the pixel in question falls into the kernel is then used to decide whether that pixel will be set to black or not. Since this is the first filtering pass it is rather blunt and will not remove all of the noise in the image, this will come at a later stage through the use of min max threshold. Just so that the gradients can be visualised this is what is generated (adjusted to be visible and comprehensible for a human) see above *figure 5*.

Figure 6: Gradient Direction

The part of the edge detection that this portion of the code is performing is removing parts of the image which have random lines and sporadic noise. This is due to us having a "direction" of where the gradient of the image is travelling. From this we can create a image kernel of our processed image so far. Depending on what the direction is it will fall into several categories. These can be seen in the code as follows:

```
public double[,] ApplyGradientMagnitudeThreshold(double[,] angles, double[,] magnitudes)
{
    double[,] result = magnitudes;
    double[,] anglesInDegrees = ConvertThetaToDegrees(angles);

    for (int i = 0; i < anglesInDegrees.GetLength(0); i++)
    {
        for (int j = 0; j < anglesInDegrees.GetLength(1); j++)
        {
            double[,] magnitudeKernel = BuildKernel(j, i, 3, magnitudes).matrix;

            if (anglesInDegrees[i, j] < 22.5 || anglesInDegrees[i, j] >= 157.5)
            {
                if (magnitudes[i, j] < magnitudeKernel[1, 2] || magnitudes[i, j] < magnitudeKernel[1, 0])
                {
                    result[i, j] = 0;
                }
            }
            else if (anglesInDegrees[i, j] >= 22.5 && anglesInDegrees[i, j] < 67.5)
            {
                if (magnitudes[i, j] < magnitudeKernel[0, 2] || magnitudes[i, j] < magnitudeKernel[2, 0])
                {
                    result[i, j] = 0;
                }
            }
            else if (anglesInDegrees[i, j] >= 67.5 && anglesInDegrees[i, j] < 112.5)
            {
                if (magnitudes[i, j] < magnitudeKernel[0, 1] || magnitudes[i, j] < magnitudeKernel[2, 1])
                {
                    result[i, j] = 0;
                }
            }
            else if (anglesInDegrees[i, j] >= 112.5 && anglesInDegrees[i, j] < 157.5)
            {
                if (magnitudes[i, j] < magnitudeKernel[0, 0] || magnitudes[i, j] < magnitudeKernel[2, 2])
                {
```

```
                    result[i, j] = 0;
                }
            }
            else throw new Exception();
        }
    }

    return result;
}
```

The use of the exception at the end is because the code above should catch all values however if it doesn't then something has gone wrong and therefore the process should not continue. After this has been applied to our image we are left with:

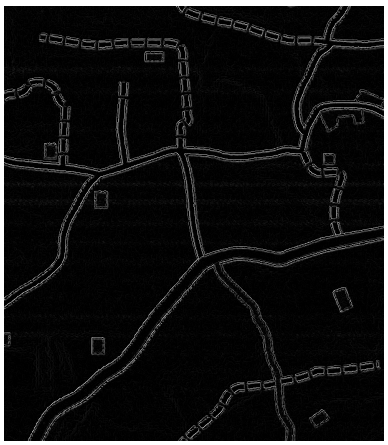


Figure 7: Magnitude Threshold

6. Min Max Threshold and Potential Edge Calculations
This part of the Canny edge detection is also called the double threshold. This is where the image pixels will all be taken and their values considered. This is when it becomes necessary for us to use the black and white version of the image. If we did not then there would be no easy way to perform this. This is because unlike most of he other steps of the edge detection we are not interested yet at the pixels which are surrounding the ones we are looking at. We are just interested in its specific value. The code to perform this is as follows.

```
public (double, bool)[,] ApplyDoubleThreshold(double l, double h, double[,] gradients)
{
    double min = l * 255;
    double max = h * 255;

    (double, bool)[,] result = new (double, bool)[gradients.GetLength(0), gradients.GetLength(1)];

    for (int i = 0; i < gradients.GetLength(0); i++)
    {
        for (int j = 0; j < gradients.GetLength(1); j++)
        {
            if (gradients[i, j] < min) result[i, j] = (0, false);
            else if (gradients[i, j] > min && gradients[i, j] < max) result[i, j] = (gradients[i, j], false);
            else if (gradients[i, j] > max) result[i, j] = (gradients[i, j], true);
            else throw new Exception();
        }
    }

```

```
19      return result;
20  }
```

The function takes two important parameters. The lower bound and the upper bound. These are the values at which we decide if a pixel is too weak and is to be set to black, if it is a "weak" pixel or a "strong" pixel. These are not important at the moment however will be used when it comes to hysteresis. Some pixels will be out right removed however and we can see the result of this double threshold is.

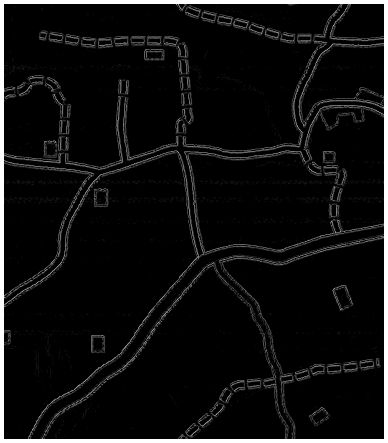


Figure 8: Magnitude Threshold

As you can see lots of noise from the scan lines of the image have been removed in this step as they would have been too small to make it past the lower threshold. Now we have an 2D array of pixel values and whether they are considered "strong" or not. If they are strong this is represented by **true** in the 2^{nd} part of the tuple. And **false** for a "weak" pixel.

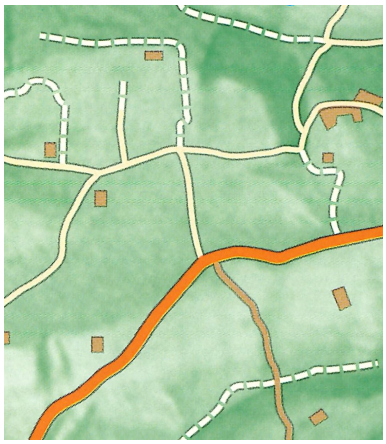
7. Edge tracking by Hysteresis

This is the final step of traditional Canny edge detection. This will require the 2D array of tuples and will require kernels of the image as it loops over every pixel. This will cause a problem since the usual way fo doing it would default to grey if the kernel overlapped with the edge of the image. So in this case we default to the pixel itself because any other value could cause us to get an erroneous edge. The way that this works is if the pixel is a "strong" pixel then it is defaulted to an edge since it was above the previous threshold. If the pixel is "weak" then it will build a kernel of all of the images around it. If any of the pixels which surround it are "strong" then this pixel is made "strong". The code for this is as follows.

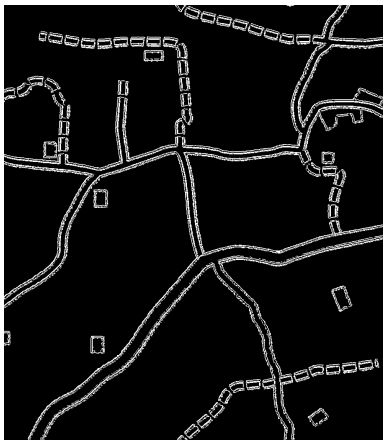
```
1   public double[,] ApplyEdgeTrackingHysteresis((double, bool)[,] arrayOfValues)
2   {
3       double[,] result = new double[arrayOfValues.GetLength(0), arrayOfValues.GetLength(1)];
4
5       for (int i = 0; i < arrayOfValues.GetLength(0); i++)
6       {
7           for (int j = 0; j < arrayOfValues.GetLength(1); j++)
8           {
9               if (arrayOfValues[i, j].Item2 == false)
10              {
11                  (double, bool)[,] imageKernel = BuildKernel(j, i, 3, arrayOfValues);
12                  bool strong = false;
13                  for (int k = 0; k < 3 && !strong; k++)
14                  {
15                      for (int l = 0; l < 3 && !strong; l++)
16                      {
17                          if (imageKernel[k, l].Item2 == true) strong = true;
```

```
                    }
                }

                result[i, j] = strong ? 255 : 0;
            }
            else result[i, j] = 255;
        }
    }

    return result;
}
```

After this has been completed we are left with a classically edge detected image which looks as follows. The left image is the original for comparison purposes.



(1) Original Input Image



(2) Image after Classic Edge Detection

As is visible in the final image we can see that after the edge detection there are holes in the lines. As well as this there are occasional gaps this is where I came up with a extra couple of steps. This allows the image to be properly formed and connect any miscellaneous roads which have small gaps.

8. Emboss Kernel
This stage isn't strictly needed for more than the reasons stated above, this will make it so that the some roads which are slightly separated, or artifacts left over from the edge detection are removed. This is done thought the use of an image kernel which is as follows:

$$\begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

The code for this is very simple and involved convolution across the entire image using this code.

```
public double[,] EmbossImage(double[,] imageArray)
{
    double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];

    Matrix embossMatrix = new Matrix(new double[,]
```

```
    {
        { -2, -1, 0 },
        { -1, 1, 1 },
        { 0, 1, 2 },
    });

    for (int i = 0; i < imageArray.GetLength(0); i++)
    {
        for (int j = 0; j < imageArray.GetLength(1); j++)
        {
            Matrix imageKernel = BuildKernel(j, i, 3, imageArray);
            result[i, j] = Math.Abs(Matrix.Convolution(imageKernel, embossMatrix));
        }
    }

    return result;
}
```

This results in, as you can imagine, an embossed image.

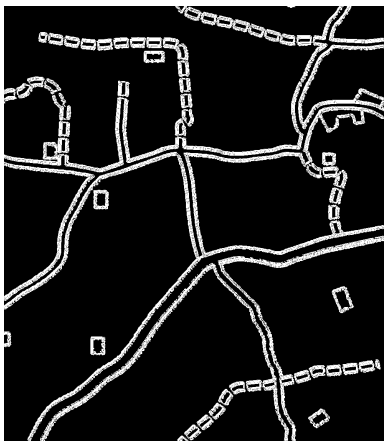


Figure 9: Magnitude Threshold

9. Custom Hole Filling
Now that the lines of the image have been increased then the only step which remains is to make the lines full and complete, this means that in the future when this is Incorporated into my final solution when a filling algorithm is applied it wont pick up erroneous roads.

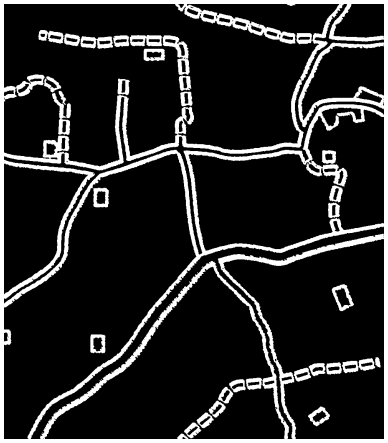


Figure 10: Magnitude Threshold

This is completed with the following code, the way that it works is that it takes a kernel of the surrounding image. If there is a certain amount of pixels in the surrounding kernel which are white then the center pixel is set to white. This threshold can be changed but 4 works well.

```
public double[,] FillImage(double[,] imageArray)
{
    double[,] result = imageArray;

    for (int i = 0; i < imageArray.GetLength(0); i++)
    {
        for (int j = 0; j < imageArray.GetLength(1); j++)
        {
            Matrix imageKernel = BuildKernel(j, i, 3, imageArray);
            int count = 0;
            foreach (double value in imageKernel.matrix)
            {
                if (value >= 255) count++;
            }

            if (count > 4) result[i, j] = 255;
        }
    }

    return result;
}
```

1.6.3 Graph Class and Graph Traversal

The graph data structure is well documented and has two main ways of being represented. One of which is a Adjacency List and the other is an Adjacency Matrix, each have their advantages and disadvantages so I will start with those.

1. Adjacency Matrix
    - Advantages

      Very fast when needing to lookup connections.
      Inserting is also fast due to it being instantly accessible and not a dynamic structure.
    - Disadvantages

      Very memory inefficient and will need to grow exponentially in each dimension with the amount of pixels in the image.

When you have a sparse graph it is even more inefficient.

2. Adjacency List
   - Advantages

     Easier to use pragmatically and implement
     It is much easier to use Linq functions with to find graph connections
   - Disadvantages

     Relatively slower when it comes to accessing sections of the graph.
     Would have to be a hybrid with a dictionary to allow for reasonable use

.

With all of this being said I decided to go for a Dictionary List since this was the easiest way to programmatically manipulate it. It also makes it easier to enter a new graph. This compared to a matrix where it would get into extreme values quickly. The structure of my prototype graph is:

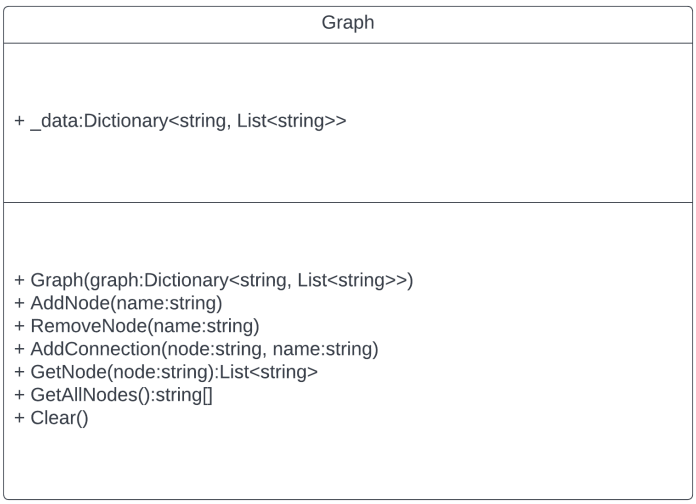


Figure 11: Graph UML Diagram

and in code

```
1  public class Graph
2  {
3      public Dictionary<string, List<string>> _data = new Dictionary<string, List<string>>();
4
5      public Graph(Dictionary<string, List<string>> graph)
6      {
7          _data = graph;
8      }
9
10     public void AddNode(string name)
11     {
12         if (_data.ContainsKey(name)) throw new GraphException($"Cannot add {name}, node already exists.");
13         _data.Add(name, new List<string>());
14     }
15
16     public void RemoveNode(string name)
```

```
    {
        if (!_data.ContainsKey(name)) throw new GraphException($"Cannot remove {name}, node does not exist.");
        _data.Remove(name);
    }

    public void AddConnection(string node, string name)
    {
        if (!_data.ContainsKey(node)) throw new GraphException($"Cannot add connection {name} to {node} original
↪   node does not exist.");
        if (_data[node].Contains(name)) throw new GraphException($"Cannot add connection {name} to {node}
↪   connection already exists.");
        _data[node].Add(name);
    }

    public List<string> GetNode(string node)
    {
        if (!_data.ContainsKey(node)) throw new GraphException($"Node {node} does not exist.");
        return _data[node];
    }

    public string[] GetAllNodes() => _data.Keys.ToArray();

    public void Clear() => _data.Clear();
}
```

This is the most basic of graph structures and may need to be changed as I develop the final solution however for the moment it serves as a good prototype. With this graph I also went on to program basic DFS (Depth-First Search) and BFS (Breadth First Search).

```
public static string[] DFS(string start, Graph graph)
{
    List<string> path = new List<string>();
    Stack<string> stack = new Stack<string>();
    Dictionary<string, bool> visited = new Dictionary<string, bool>();
    foreach (string s in graph.GetAllNodes()) visited.Add(s, false);

    // Kick Start
    stack.Push(start);

    while (!stack.IsEmpty())
    {

        string node = stack.Pop();
        path.Add(node);
        visited[node] = true;

        List<string> connections = graph.GetNode(node);

        connections.Reverse();

        foreach (string s in connections)
        {
            if (visited[s] == false)
            {
                stack.Push(s);
            }
        }
    }


```

```
32      return path.ToArray();
33  }
34
35  public static string[] BFS(string start, Graph graph)
36  {
37      List<string> path = new List<string>();
38      Queue<string> stack = new Queue<string>();
39      Dictionary<string, bool> visited = new Dictionary<string, bool>();
40      foreach (string s in graph.GetAllNodes()) visited.Add(s, false);
41
42      // Kick Start
43      stack.Enqueue(start);
44
45      while (!stack.IsEmpty())
46      {
47
48          string node = stack.Dequeue();
49          path.Add(node);
50          visited[node] = true;
51
52          List<string> connections = graph.GetNode(node);
53
54          connections.Reverse();
55
56          foreach (string s in connections)
57          {
58              if (visited[s] == false)
59              {
60                  stack.Enqueue(s);
61              }
62          }
63      }
64
65      return path.ToArray();
66  }
```

Both of these I ran through by hand and they came out correct. It was useful to see how they are calculated and how the implementation is different depending on whether you use a stack or a queue for the graph traversal.

1.6.4 Windows Forms with Images

To allow the user to easily be able to see the output of the edge detection. In order to do this the project needed to be created in dot-Net Framework. Once this is done a basic mock up of what the prompt to the user will see is made in the user interface. This creates backend XML which is interpreted by the framework to be presented to the user. As well as this there is also the programmatic part to it which can be used to display the image.

Example

```
1   partial class ShowImage
2   {
3       /// <summary>
4       /// Required designer variable.
5       /// </summary>
6       private System.ComponentModel.IContainer components = null;
7
8       /// <summary>
9       /// Clean up any resources being used.
10      /// </summary>
```

```
11      /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
12      protected override void Dispose(bool disposing)
13      {
14          if (disposing && (components != null))
15          {
16              components.Dispose();
17          }
18          base.Dispose(disposing);
19      }
20
21      #region Windows Form Designer generated code
22
23      /// <summary>
24      /// Required method for Designer support - do not modify
25      /// the contents of this method with the code editor.
26      /// </summary>
27      private void InitializeComponent()
28      {
29          this.imageBox = new System.Windows.Forms.PictureBox();
30          this.next = new System.Windows.Forms.Button();
31          this.content = new System.Windows.Forms.RichTextBox();
32          ((System.ComponentModel.ISupportInitialize)(this.imageBox)).BeginInit();
33          this.SuspendLayout();
34          //
35          // imageBox
36          //
37          this.imageBox.Location = new System.Drawing.Point(12, 12);
38          this.imageBox.Name = "imageBox";
39          this.imageBox.Size = new System.Drawing.Size(500, 450);
40          this.imageBox.TabIndex = 1;
41          this.imageBox.TabStop = false;
42          //
43          // next
44          //
45          this.next.Font = new System.Drawing.Font("JetBrains Mono SemiBold", 24.75F, System.Drawing.FontStyle.Bold,
    ↪ System.Drawing.GraphicsUnit.Point, ((byte)(0)));
46          this.next.Location = new System.Drawing.Point(518, 384);
47          this.next.Name = "next";
48          this.next.Size = new System.Drawing.Size(354, 78);
49          this.next.TabIndex = 4;
50          this.next.Text = "Continue";
51          this.next.UseVisualStyleBackColor = true;
52          this.next.Click += new System.EventHandler(this.next_Click);
53          //
54          // content
55          //
56          this.content.AcceptsTab = true;
57          this.content.Font = new System.Drawing.Font("JetBrains Mono SemiBold", 15F, System.Drawing.FontStyle.Bold);
58          this.content.Location = new System.Drawing.Point(518, 12);
59          this.content.Name = "content";
60          this.content.ReadOnly = true;
61          this.content.Size = new System.Drawing.Size(354, 366);
62          this.content.TabIndex = 5;
63          this.content.Text = "";
64          //
65          // ShowImage
66          //
67          this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
68          this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
69          this.ClientSize = new System.Drawing.Size(884, 474);
```

```
70          this.Controls.Add(this.content);
71          this.Controls.Add(this.next);
72          this.Controls.Add(this.imageBox);
73          this.Name = "ShowImage";
74          this.Text = "ShowImage";
75          this.Load += new System.EventHandler(this.ShowImage_Load);
76          ((System.ComponentModel.ISupportInitialize)(this.imageBox)).EndInit();
77          this.ResumeLayout(false);
78
79      }
80
81      #endregion
82
83      private System.Windows.Forms.PictureBox imageBox;
84      private System.Windows.Forms.Button next;
85      private System.Windows.Forms.RichTextBox content;
86  }
87
88  public partial class ShowImage : Form
89  {
90      private Bitmap _image;
91      private string _content;
92
93      public ShowImage(Bitmap image, string content)
94      {
95          this.ControlBox = false;
96
97          _image = image;
98          _content = content;
99
100         InitializeComponent();
101     }
102
103     private void ShowImage_Load(object sender, EventArgs e)
104     {
105         imageBox.SizeMode = PictureBoxSizeMode.StretchImage;
106         imageBox.Image = _image;
107         content.Text = _content;
108     }
109
110     private void next_Click(object sender, EventArgs e)
111     {
112         Close();
113     }
114 }
```

1.6.5 Analysis of Prototype

These two partial classes come together to form the final form. One thing which I learned from this prototype is that there are several ways that the image can be made to fill the text box and that needs to be carefully considered. I also learned from these prototypes, especially with the edge detection is that if they are not optimised they can cause a computer to really slow down and become unresponsive. This was evident with the multithreaded version which cause the computer to run out of ram and crash. This turned out to be an object reference issue where the bitmap images where being recreated every ime instead fo transferring the reference. This caused the memory to fill and slow down the program.

The forms prototype was very interesting since it showed me how I could go about getting th euler inputs from when they clicked on an image. This will allow me in the main program to pathfind from point to point with minimal knowledge of the program. This is because

otherwise I will have to have the user put in coordinates of points on th map.

1.7 Objectives

After conducting the initial interview and reflecting upon the results of my research I have formed a list of objectives that the program must meet to be considered complete. As well as the base objectives I have also, with help from my end user, come up with extensions which will increase the effectiveness of my solution overall.

1. The Program must have way to input a Map
   - 1.1 The Program should be able to parse a map from a file, including
     - 1.1.1 A photograph of an map
     - 1.1.2 A screenshot of an existing map
     - 1.1.3 A hand drawing of suitable quality (if it is not a message should be shown)
   - 1.2 When the user inputs a map, the program will ask them
     - 1.2.1 What type of map they are inputting
     - 1.2.2 Whether this is the correct image
     - 1.2.3 Whether they want the image deleted after edge detection
     - 1.2.4 Whether they would like the image to be stored in a binary file,
       - 1.2.4.1 If selected then the programs should ask for a name
       - 1.2.4.2 It should ask for a description of the image
       - 1.2.4.3 It should ask for the type of image.
       - 1.2.4.4 The time and date of the image should be automatically calculated.

       These are just some examples of prompts
   - 1.3 The inputted map should be converted into a graph
     - 1.3.1 The map (in graph form) should be able to be traversed
     - 1.3.2 The map in graph form should be simplified to ensure that redundant nodes are not recorded.
   - 1.4 If any error occurs during the map input process an appropriate error should be displayed and the program should continue to run
2. The Program must perform Canny edge detection
   - 2.1 At each stage of the edge detection an image should be produced
   - 2.2 Between each stage the user should be able to repeat the last step in order to change parameters.
     The user should be able to change (at various stages):
     - 2.2.1 The sigma value of the Gaussian elimination
     - 2.2.2 The lower threshold value
     - 2.2.3 The higher threshold value
     - 2.2.4 The Gaussian kernel size
     - 2.2.5 The black and white filter ratios
     - 2.2.6 The amount of times embossing is performed
     - 2.2.7 The times de-blocking should be performed
   - 2.3 The edge detection must have the option to be multi threaded.
     - 2.3.1 There should be presets to allow quicker processing
       - 2.3.1.1 There should be a preset for hand drawn images
       - 2.3.1.2 There should be a preset for photographed images
       - 2.3.1.3 There should be a preset for screen shot images
   - 2.4 The edge detection must have the option to be single threaded

3. The Program must overlay the detected roads onto the original image
   - 3.1 The result of the edge detection will be shown to the user before road detection
   - 3.2 The program will perform road detection
     - 3.2.1 The image should have the option to be inverted
     - 3.2.2 A filling algorithm should be applied to the image
     - 3.2.3 The percentage threshold for non roads much be changeable by the user
     - 3.2.4 The total filled image can be displayed to the user
     - 3.2.5 The singled out roads and paths must be shown to the user
4. The Program must allow Map Traversal
   - 4.1 There should be Multiple Traversal Algorithms Available to be chosen from.
     - 4.1.1 The Program should implement Routing Algorithms
       - 4.1.1.1 This includes Dijkstra's algorithm
       - 4.1.1.2 This includes A*
     - 4.1.2 The Program should Implement Searching Algorithms these do not have to be shown to the user.
       - 4.1.2.1 This includes BFS (Breadth-first search).
       - 4.1.2.2 This includes DFS (Depth-first search).
   - 4.2 Depending on the option that the user chooses they can either
     - 4.2.1 Decide a specific algorithm to use
       - 4.2.1.1 The general efficiency should be displayed.
       - 4.2.1.2 The general length of each should be displayed.
       - 4.2.1.3 The node count of each should be displayed if Dijkstra's is selected.
5. The Program must have a Clear and Simplistic GUI.
   - 5.1 At a glance the user should be able to ascertain which step they are at in the process.
   - 5.2 Whenever a forms is displayed it should not serve more than one purpose.
   - 5.3 There should be a setting so that if the user chooses more detail can be displayed.
   - 5.4 The main user window should not be cluttered with old information.
6. The program must implement abstract data types
   - 6.1 The program must implement a matrix class
     - 6.1.1 The program must be able to perform basic operations
       - 6.1.1.1 Perform matrix multiplication
       - 6.1.1.2 Perform matrix addition
       - 6.1.1.3 Perform matrix subtraction
       - 6.1.1.4 Perform scalar multiplication
       - 6.1.1.5 Perform matrix minimization
     - 6.1.2 The program must be able to find the determinant of a matrix
     - 6.1.3 The program must be able to find the inverse of a matrix
     - 6.1.4 The program must be able to apply the convolution operation
   - 6.2 The program should implement a graph class
     - 6.2.1 The graph should be able to be modified by
       - 6.2.1.1 Inserting Nodes
       - 6.2.1.2 Accessing per node
       - 6.2.1.3 Access all nodes
       - 6.2.1.4 Inserting connections between nodes
     - 6.2.2 It should be implemented using an adjacency list.

Extension Objectives

7. The program should be able to output
   - 7.1 The map in a binary file format
     - 7.1.1 This file can be saved
     - 7.1.2 This file can be re-read and re-routed
   - 7.2 The saved images from the processing of the map should be able to be saved in a compressed format.
   - 7.3 The routed map with path drawn on it
   - 7.4 The saved binary file should be able to be cloned
   - 7.5 The saved binary file should be able to be renamed
   - 7.6 The saved binary file should be able to be have its description changed
   - 7.7 The saved binary file should be able to be able to be deleted
8. The program should have re-callable settings
   - 8.1 Map Algorithm
   - 8.2 Random Save Names
   - 8.3 Map Approximations
9. The program settings should be easily movable.
10. The program save files should be easily movable.

1.8 Modelling

I believe that most of the modelling has been completed in terms of how the algorithms will be used and how they will function above. Here I will briefly explain how I will structure the program.

1.8.1 General Overview of Structure

Through research and discussion with other students about how best to aly out the program I have decided to split the front end and the back end of my program into two separate pieces. I am not sure how this will be accomplished at the moment, however there is a way to have a library project and an application project. The other option is that i develop a server version which holds the backend then there are HTML requests between the client and the server which does all of the processing. The main reason I feel a separated structure is vital is that from talking to my end user having multiple ways to use this application would be useful and having a separated back end and front end would mean that the back end could be used in the front end with no modification.

1.8.2 User Interface

The user interface with the local run portion will be simple. From talking to my end user I have found that something graphical is a necessity. There is the option of using Windows Forms for this however the one downside to using Forms is that it is a legacy technology and can sometimes cause issues when it comes down to scaling. The other drawback which is not an issue in this case is that windows forms only works on windows, this means that this application

wouldn't work on anything but a windows computer. This however is not an issue in this project due to it being out of the scope.

As for the design of the user interface I think that I will go with a console application with a windows form showing when the program gets to the pathfinding stage. I believe that this will give the best ballance between inputs and giving enough information.

1.8.3 File Storage

I feel that I will need some way of storing information, in order to do this I will have to find some medium to do this. I have not done any prototyping on this section however I believe that having a basic directory structure with sub files depending on the stuff that needs to be saved. As for the layout of this I am not sure at this time.

I will be using a custom binary file to store some of the map information, this will be saved somewhere by the program, it may be useful to have this accessible through a normal text editor to allow the user to see whats going. Again this is a bit of a push since this program is not meant for an expert and is just meant to work without any caveats. The allowing the user to edit their own files would open up issues with file validation which is again is out of the scope of this program.

2 Technical Design

2.1 Programming Language Selection and Libraries Used

I selected C as my programming language for several reasons. Currently, it is the language that I am most familiar with. In addition, I conducted research on which languages are best for fast processing, and found that C, C++, and C are among the top contenders. Considering my skill set and the importance of speed in this situation, I concluded that C would be a good fit. Furthermore the object orientated nature of the language means that I will be able to separate the front end and the back end processing into separate bll files keeping the code clean and easily maintainable.

Find below a list of all libraries I used:

2.1.1 Linq

In order to manipulate lists and create the data structures that I need I will need to use some Linq methods. During the prototyping stage I found that using some Linq methods such as the Select statement allowed the program to be easier to read and make logical sense. As well as this there have been optimisations made in the iterative Linq methods which will make my program faster. Similar to some of the following libraries this is a Microsoft Library which is open source.

2.1.2 Bitmap

In order for my program to function a required part of it is that it is able to take an image as an input. In native C there is no set way to do this. Therefore I needed to use the Microsoft System.Drawing Namespace. This namespace provides access to GDI+ basic graphics functionality. This does limit this project as is to only working on Windows since the library requires access to the GDI+ native library which is only on windows services.
The only part of this library I will be using is the Bitmap class. This will allow me to accept all types of images without the need of parsing them myself since this is not the aim of my project.

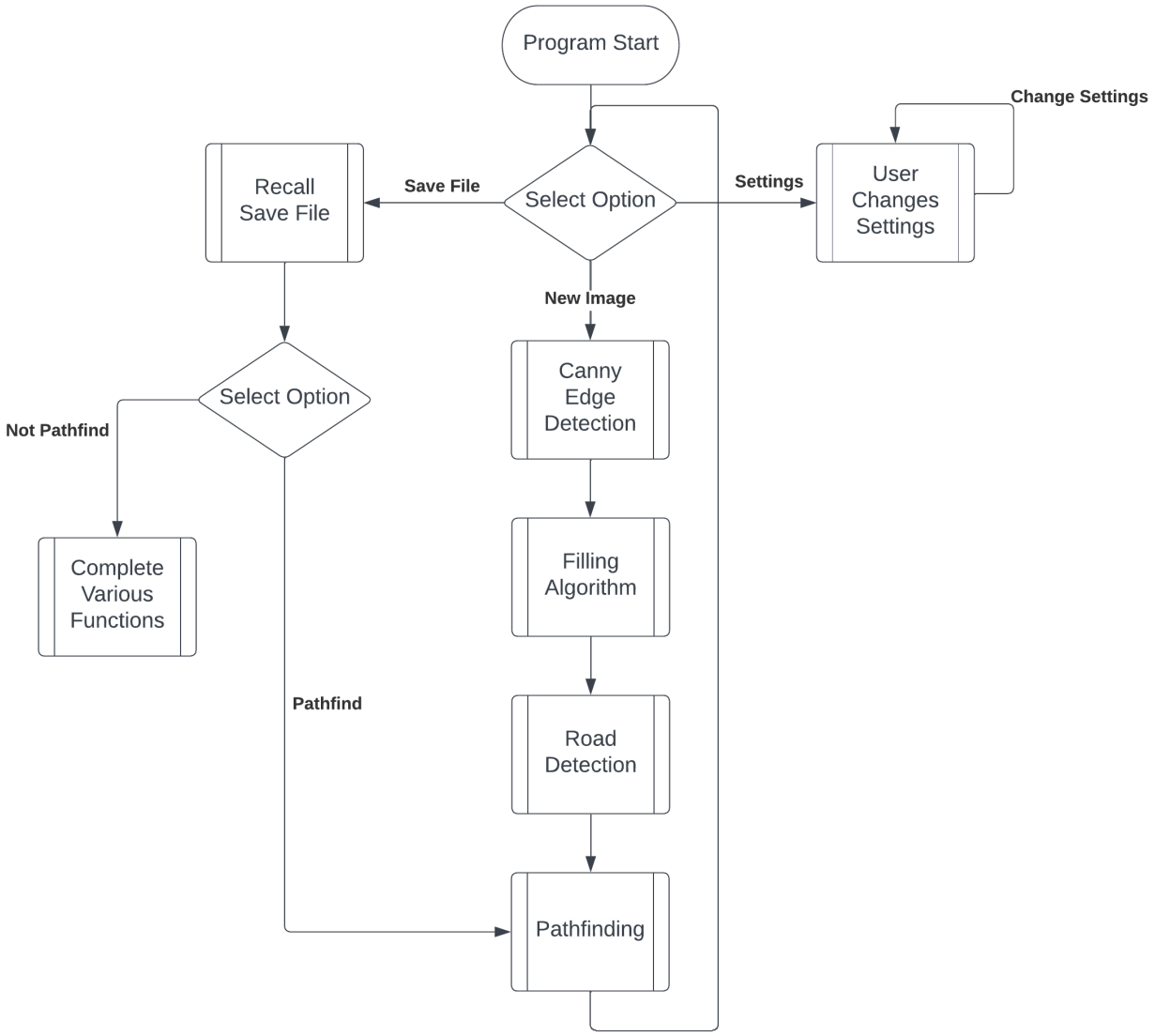
2.1.3 Windows Forms

In order to complete my objectives my program will need to be easy to use and any user with some degree of technical competency should be able to use it. In order to achieve this objective I though that instead of using some form of console input in order to get a starting and an end location, that it would be better to use some form of GUI. In order to do this I will use Windows Forms. This will allow me to make a simple GUI which will allow the end user to interact with the user and easily understand.
The things which I will end up using the windows forms are the map traversal, allowing the user to select a start and an end node with a click instead of having to enter a coordinate. As well as this I will also use forms to show the user the stages of, for example, the Canny edge detection.

2.2 High Level Overview

The general purpose of my project is to allow a user to take a map and input it into my program, then subsequently convert it into a routable map.

In order to achieve this goal my program will first take an input, the users map. It will then take this map and convert it into an machine readable format, a Bitmap. Canny edge detection will then be performed on it causing the edges and the surroundings of the paths on the image to be found. Using these edges a filling algorithm will fill the spaces encapsulated by the lines. Finally these filled spaces will be used to convert the whole image to a graph which can then be traversed using graph traversal algorithms such as A* or Dijkstra's algorithm.

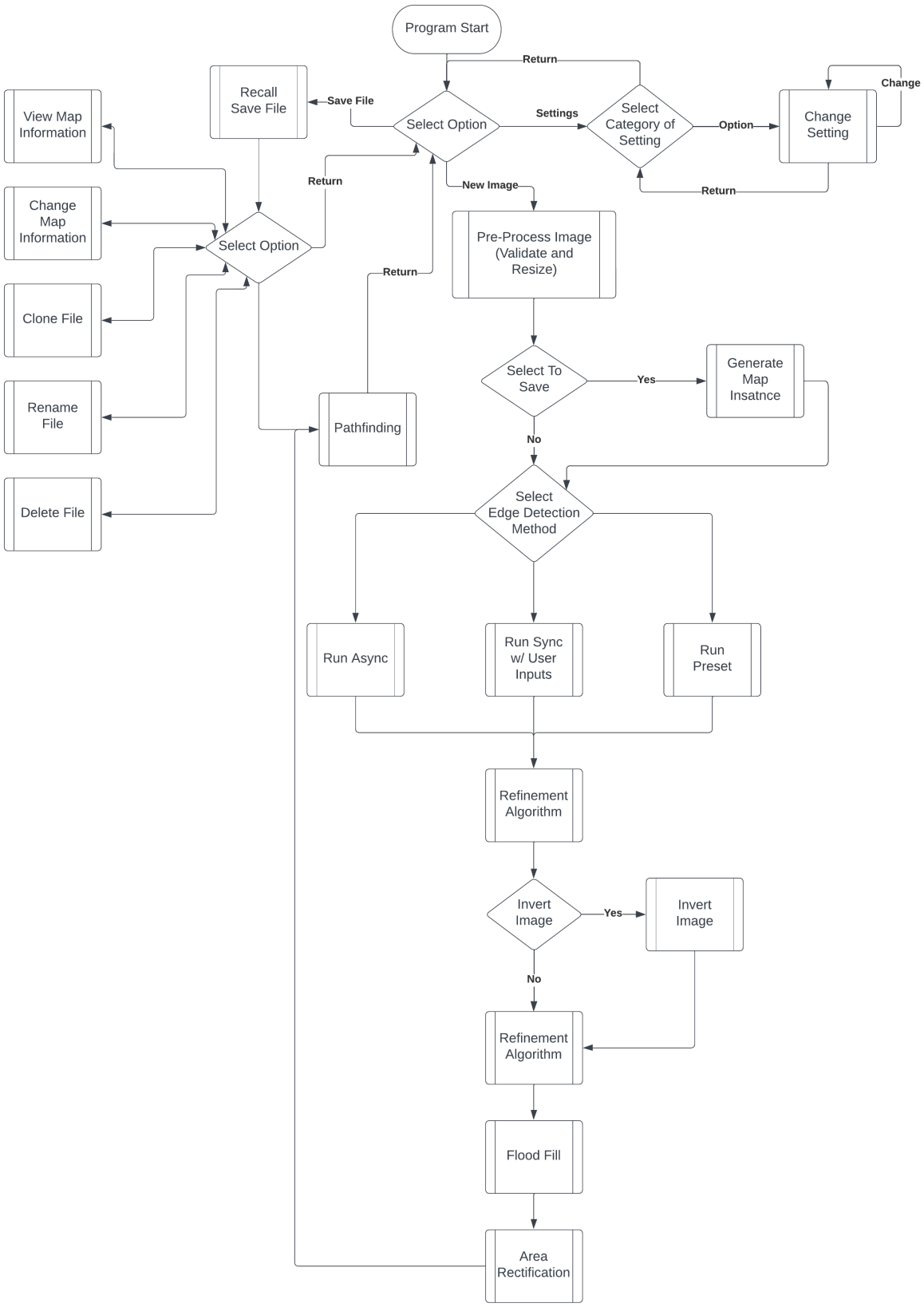


(1) Data Flow Diagram

This is a high level overview of the flow of data through the program, find a more detailed one below.

The version of Edge Detection I will be using as previously stated will be Canny Edge Detection, this is as opposed to Sobel Edge Detection. The main version of filling I will be using is flood fill due to it simple nature to implement ad due to the fact that it does not take much memory and can be made recursive so it performs well. The final main algorithm I will need to use is image kernels and convolution, this will allow me to manipulate the inputted image.

2.2.1 Full DFD (Data Flow Diagram)



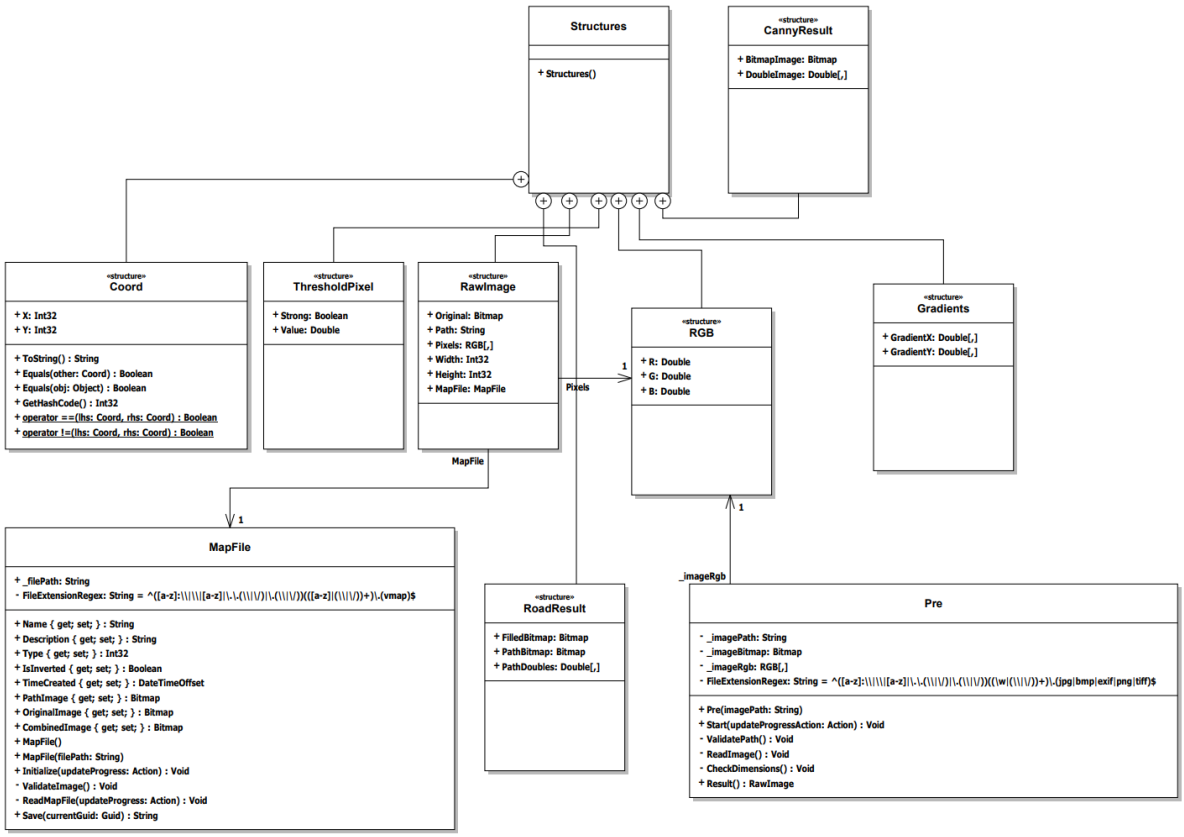
(2) Full DFD

2.2.2 Backend Library

For my project to ensure that I conform to the OOP principle of encapsulate what varies. I will accomplish this through the use of classes and encapsulation. Furthermore I have also made the decision to split up my solution into two separate projects, this means that my program will produce two files in order to run, one of these will be the DLL for the backend library and the other will be the executable for the front end.

Contained within this backend section of my program will be contained the edge detection, road detection, complex data types and graph traversal algorithms as well as various utilities that are frequently used throughout the program.

One of the main features of the backend library are the custom structures that have been created in order to allow for easier processing of data. Find below the image of the structure class layout and the classes which link within.



(3) Overview of Backend Structures

As can be seen form the class diagram of the backend library, there is very little dependency within the library itself. This allows the backend to function independently of the program which is using it. This allows the backend to be split out and moved to another program if needed. Summarised there are four main reasons to do this:

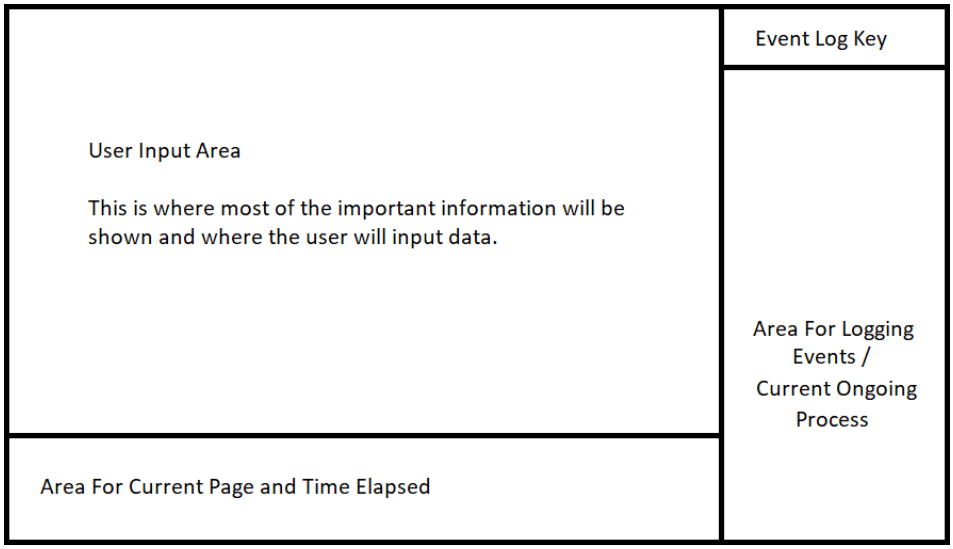
- **Modularity** - By separating the backend from the frontend one is able to be built without the other. This means that when working on my project I can take time to perfect one without impacting the other.
- **Reusability** - As previously stated being able to be reused is a large reason as to why to separating the elements is a good idea. Since if I wanted to expand this project for example and make a web interface for it, I could take the maths of the backend and recreate the front end in a web framework like Razor Pages.

- **Maintainability** - It is allot easier to maintain code when it has been organised into classes and by extension into libraries where a library is a collection of classes. It means that should something throw an error in the backend I would be able to easily isolate the issue and be able to fix it.
- **Testability** - In a similar vain to the Maintainability of the program being modular also means that it is very easy to implement testing. This means that as I go thorough making my program it will make it allot easier to separate variables and make isolated testing conditions. Furthermore it means that I can test the maths of the Canny Detection without having to worry about making an interface to it using the UI.

2.2.3 Local Application

The local application part of this program will be responsible for the tying if the various algorithms of the backend together along with providing the user with a way to interact with them, whether this is through the use of windows forms of the console for text inputs. As stated by objective 5 the design of the UI should be simplistic and easy to understand at a glance, therefore I will only be using the methods as stated above for interacting with the user. I also believe that it will be best to keep the changes between the two to a minimum and when there is a change make sure that the user is aware of it before hand.

Design of User Interface (Console)
In order to keep the user interface as easy to use as possible the console will remain static while the program is being run. This means that once it has been started and set to its correct size it will form itself to fit the screen and will only run if it has been maximized. This will allow me to make sure that the interface is clear and easy to use. Find below a mock-up of the console design.



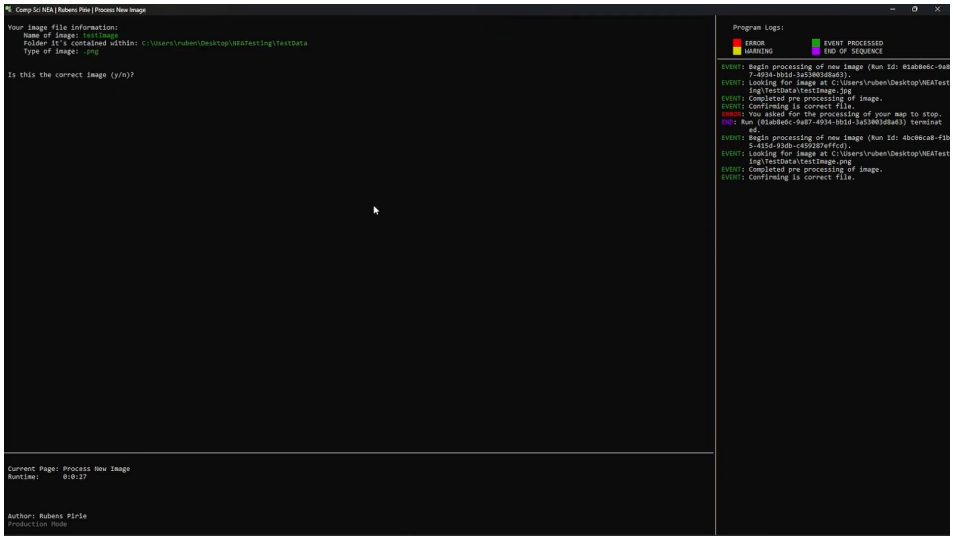
(4) Mock-up of Console Interface

As can be seen in this mock-up fo the console interface it can be seen that there is a large section for the user to enter and view important in. As will be expanded on in the second section about the Windows Forms interface, due to the static nature of the console I will be able to make a Form conform to the shape of this area. See the next paragraph for more information.

As for the other elements of the console UI, as part of objective 5, this must be easy to see at a glance what is going on and which step you are in. To accomplish this on the right hand side of the console there will be a log which, should the user select to do so in settings, will display

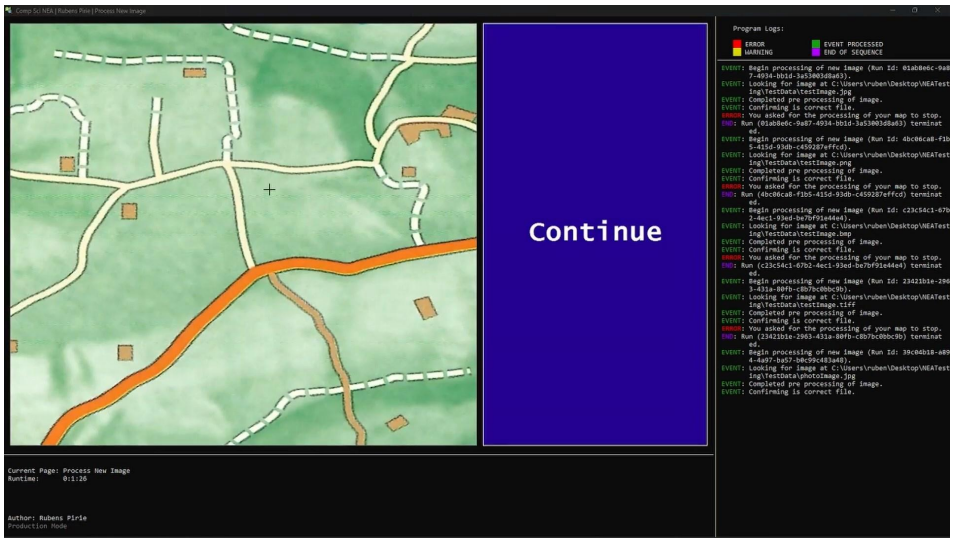
each method call and the result of that call allowing them to see exactly where they are in the process.

At the bottom of the console in the section labelled "Area for Current Page and Time Elapsed" this will be used for, as the name suggests, the current page and time elapsed. What this means is that at a glance a non-technical user or one who has opted not to have the advanced logging will still be able to see where they area at in the current process.

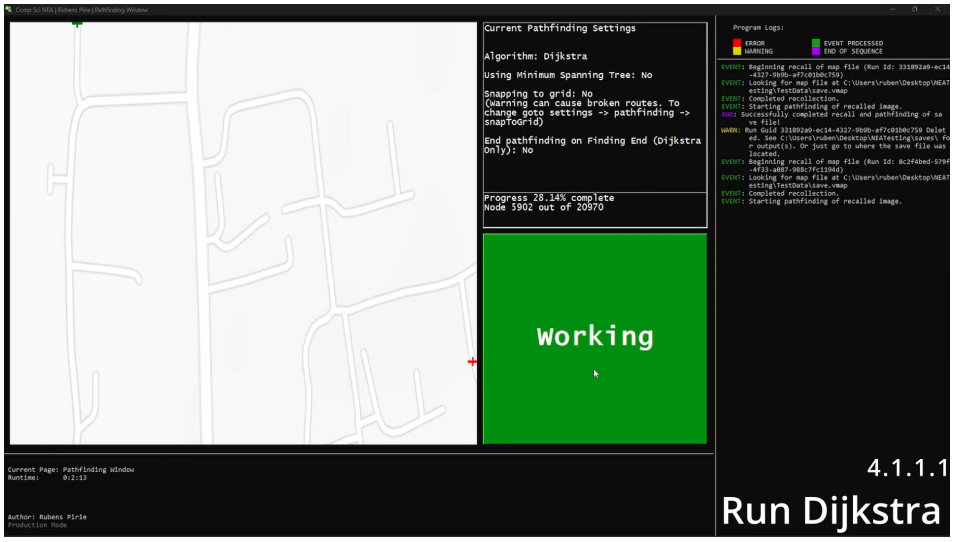


(5) Mock-up of Console Interface

Design of User Interface (Windows Forms)
For the forms interface I have chosen to keep the use of the opening of new windows to a minimum, as explained above I want this part of the program to be as simple as possible to avoid over stimulating the user and confusing them.

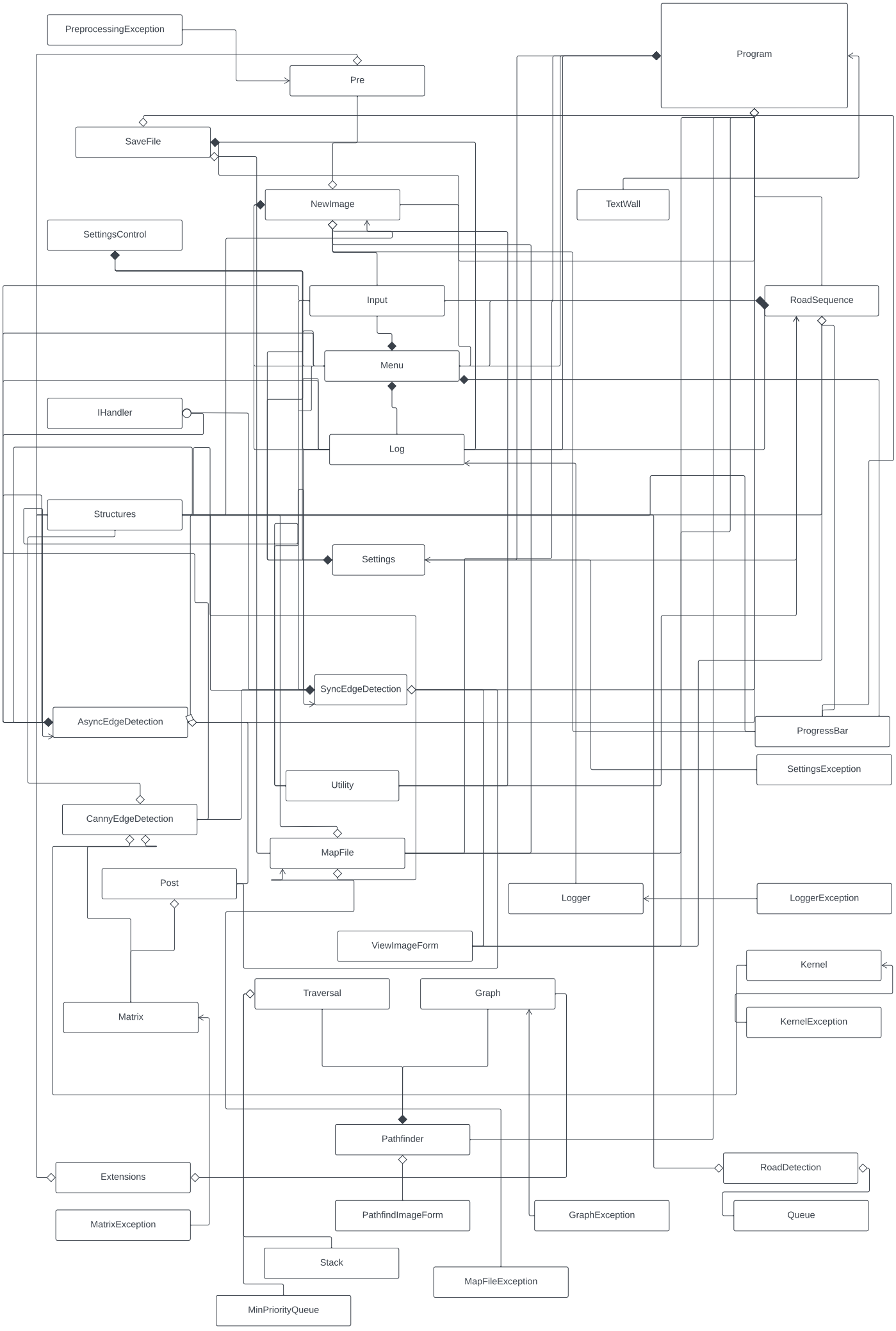


(6) Windows Form For Confirming Image



(7) Windows Form For Pathfinding Image

2.2.4 Entire Class Diagram


PreprocessingException
Pre
Program
SaveFile
NewImage
TextWall
SettingsControl
Input
RoadSequence
Menu
IHandler
Log
Structures
Settings
SyncEdgeDetection
AsyncEdgeDetection
ProgressBar
SettingsException
Utility
CannyEdgeDetection
MapFile
Post
Logger
LoggerException
ViewImageForm
Kernel
Traversal
Graph
Matrix
KernelException
Pathfinder
Extensions
RoadDetection
PathfindImageForm
GraphException
Queue
MatrixException
Stack
MapFileException
MinPriorityQueue

2.3 Binary File Structure

There will be an option during the process of the program for the user to save what they are doing with a binary file. Inside this file will be contained all of the information relating to the map. It will store, the name of the map, description, whether it needed to be inverted, date of creation, type of image, and 3 various versions of the image.

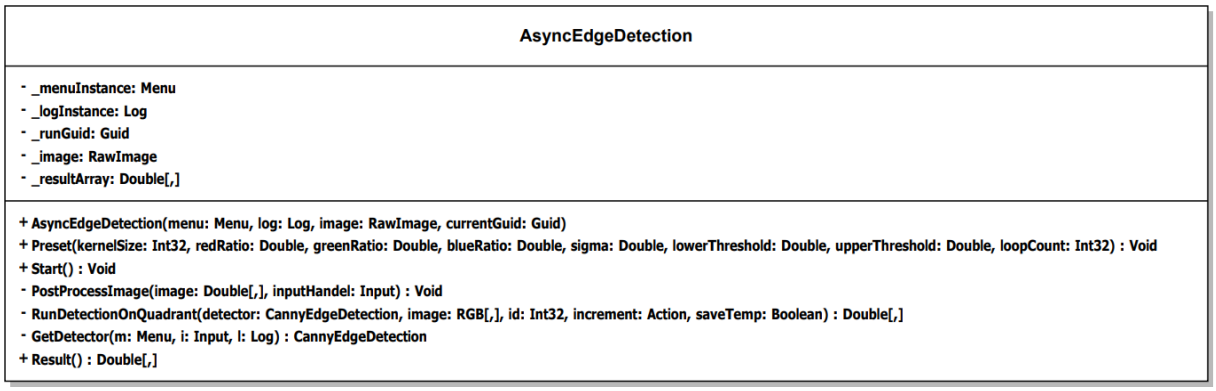
The program will ask the user if they want to save the image, if they select yes to this then it will go on to ask them what the name of the map should be. A brief description to allow them to remember what it was for, and finally the type of map which can aid in future processing. These are all stored as strings apart from the type of map which is stored as a integer.

The 3 images which are stored, these are the original image, the path found image, and the combined image which is the path overlays onto the original. These are all stored as bytes, this is to keep the size of the image to a minimum. This does raise the issue of there being no way to tell the size of the image. To overcome this there are two Int32's stored at the very beginning which are the dimensions of all the images meaning that it can be read without any guesswork. This has the effect of compressing all information relating to the images since each image's pixel is only 3 bytes.

2.4 Class Overviews

In this following part of the write up will go briefly over every class in the program first stating its function, which section it is in (backend library or front end application) and finally how it plays a part in the program.

Async Edge Detection *(Class)*



(8) Async Edge Detection Class Diagram

This class is located in the front end section of my application, its main function is to coordinate the method calls of the Canny Edge Detection class. Due to the separated nature of my program I did not want there to be any user inputs in the accrual processing section. In order to do this I needed to have a separate hander on the local application side which would ask the user for their inputs and handel all of the validation of them.

This class also implements the IHandler interface, this is to allow the main front end application to switch between the Async version and the Synchronous version of the edge detection without having a mess of IF statements. Part of the IHandler interface means that this class must contain the methods, Start() and Result(). What these methods do is what they say in the name. The start methods begins the process of getting user inputs and then starting the edge detection. The Result method will, as the name suggests return the result of the edge detection.

The PostProcessImage method in the class is responsible for the custom embossing which runs through the result of the Canny edge detection and fills in any gaps that may have appeared. It

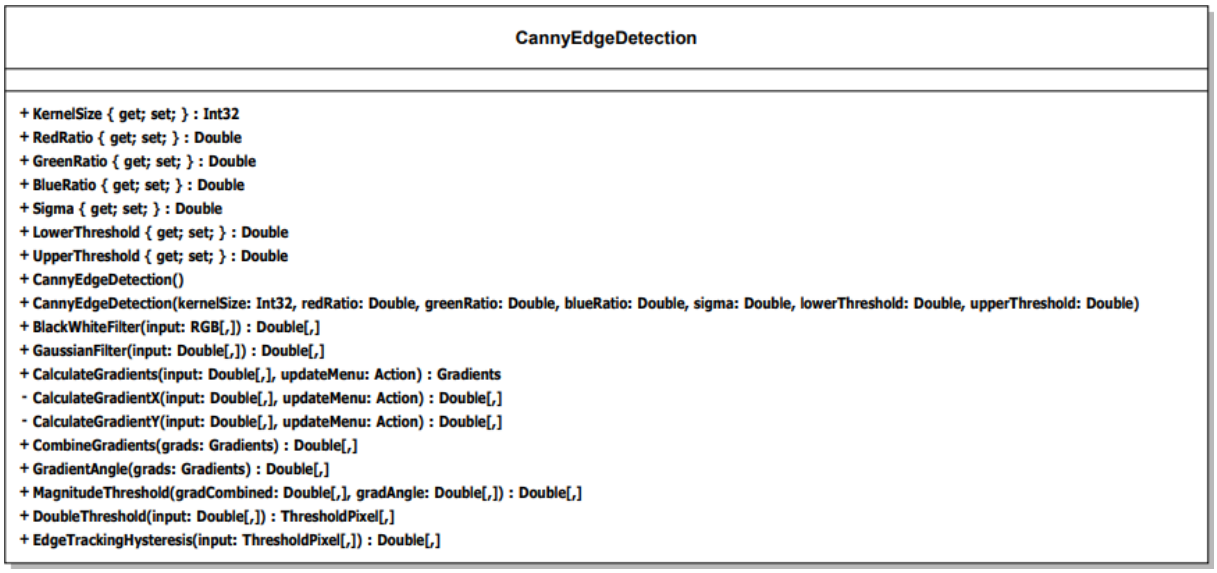
also applies an embossing kernel on the image to embolden the lines. It will also prompt the user to enter the amount of times they want to run the embossing process.

The GetDetectorMethod is used to get the variables for the Canny edge detector. This section handles all of the validation and checking that the values supplied are valid. The result of this method is that a new Canny Edge Detector object is created with the user inputted variables, this is then passed down the chain to be used in processing each quadrant.

The main differentiator between this asynchronously method and the synchronous method is that this one will split the input image into 4 distinct quadrants, this will allow the program to run each at the same time using threading. Through my prototyping stage I found that using this method of threading greatly improves the speed even on lower specification computers.

Finally the Preset method is used in the front end application in the eventuality of the user selecting preset values. An example of this is that when it gets to the selection of the edge detection method they select the "Photograph" method, the async method will be used due to its speed however instead of prompting the user it will run the various stages with predefined values.

Canny Edge Detection *(Class)*



(9) Canny Edge Detection Class Diagram

This class is located in the backend library portion of my project, its main function is to house the methods which contain the maths to perform the Canny edge detection. Since this method is in the backend library of my project there is no user input here however in many of the methods an action is passed in, this is used to update the progress bar on the front end without having the two intrinsically integrated.

The first set of methods shown in the class diagram are in essence properties on the class which are used in the various calculations. The reason that I will go with getter and setter variables is that if at some point in the future I wish to restrict access to the properties on the class or perhaps mutate the way in which they are stored this could be easily done without the need to change many different things.

The first method and subsequently the first stage in Canny edge detection is the black and white filter. The default behaviour is that it uses the industry standard for getting a single value from a RGB pixel which is as follows

$$Y' = 0.299R' + 0.587G' + 0.114B'$$

. It will perform the same calculation for every pixel in the supplied image causing it to be converted to black and white. This is used since if the detection was run on each colour chanel seperatly this would not give one unified result and could be very inaccurate.

The next method in this class is the Gaussian Filter this will pass over the image and apply a Gaussian Filter kernel to each pixel. This is mainly used in order to make sure that any noise in the image is suppressed to avoid false edges. The maths for the kernel implementation and matrix convolutions are covered in their respective classes as well as the gaussian equation. A rough approximation for the gaussian kernel is as follows:

$$\frac{1}{159}\begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

It is stated that the larger the kernel is the less effect noise will have on the result however it will

impact the performance of the detector, for this reason the Canny Edge Detection class defaults to a kernel size of 5x5.

The next stage of Canny edge detection involve calculating gradients and gradient angles and for this reason I will be combining the CalculateGradients, CalculateGradientX and CalculateGradientY into one. Again another way in which I will optimise the Canny edge detection is by using threading. In order to calculate the gradient direction I will need to use Atan2, this requires an X and Y component which are independent of each other, a perfect use of threading. When a image gets to this point it will be processed at the same time by each method. The process completed by each method is vastly the same they will just be applying different image kernels, both are the sobel edge kernels keeping with the scheme of Canny edge detection.

$$M_x = \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \text{ and } M_y = \begin{pmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{pmatrix}$$

Once the gradient magnitudes in each dimension have been calculated the result of these can be fed into the CombineGradients method. This method will finally extract the usefully data which has been created from applying the two sobel gradient kernels. The first of these is working out the combined gradient magnitudes which is simply calculated with the equation:

$$G = \sqrt{G_x^2 + G_y^2}$$

We then also want to calculate the direction in which the gradient is travelling to work out if it is extreme enough to quantify an edge. This is achieved though the use of Atan2 as follows:

$$\Theta = \text{Atan2}(G_x, G_y)$$

The next stage in Canny edge detection is gradient magnitude threshold's, this is a method of removing lines which would not be "thick enough" to be a propped edge. This is accomplished by the use of the MagnitudeThreshold method and the bullet point description:
At every pixel N in a given image, it will be suppressed (removed) if,

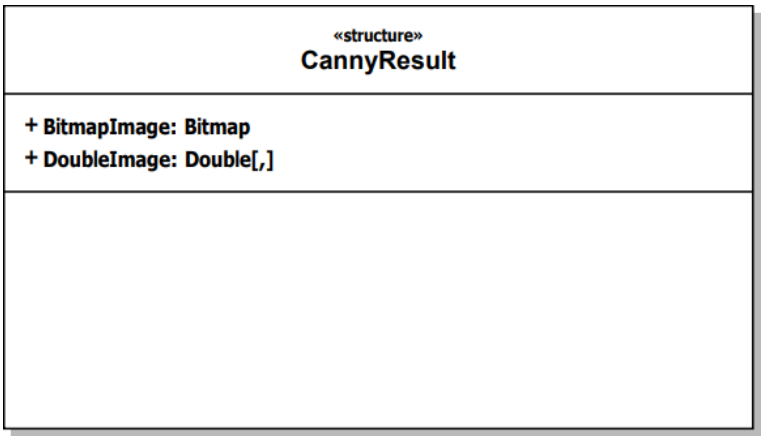
- the rounded gradient angle is 0° (i.e. the edge is in the north-south direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the east and west directions.
- the rounded gradient angle is 90° (i.e. the edge is in the east-west direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the north and south directions.
- the rounded gradient angle is 135° (i.e. the edge is in the northeast-southwest direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the north-west and south-east directions.
- the rounded gradient angle is 45° (i.e. the edge is in the northwest-southeast direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the north-east and south-west directions.

The penultimate method call is to Double Threshold suppression. After the magnitude threshold, remaining edge pixels provide a more accurate representation of real edges in an image. However, some edge pixels remain that are caused by noise and color variation which didn't get removed by the gaussian filter stage. To account for these spurious responses, it is essential to filter out edge pixels with a weak gradient value and preserve edge pixels with a high gradient value.

The way in which this is implemented is that a user threshold from the beginning is used and if the pixel is greater than the threshold then it is included, if it is below the max threshold but greater than the min then it is set to a strong pixel and retains its value. In any other case it is removed and its value is set to 0.

Finally edge tracking by hysteresis is used, to track the edge connection, blob analysis is applied by looking at a weak edge pixel and its 8-connected neighbourhood pixels. As long as there is one strong edge pixel that is involved in the blob, that weak edge point can be identified as one that should be preserved. These weak edge pixels become strong edges that can then cause their neighbouring weak edge pixels to be preserved otherwise they are not preserved and are removed. After each of these sages Canny edge detection is complete.

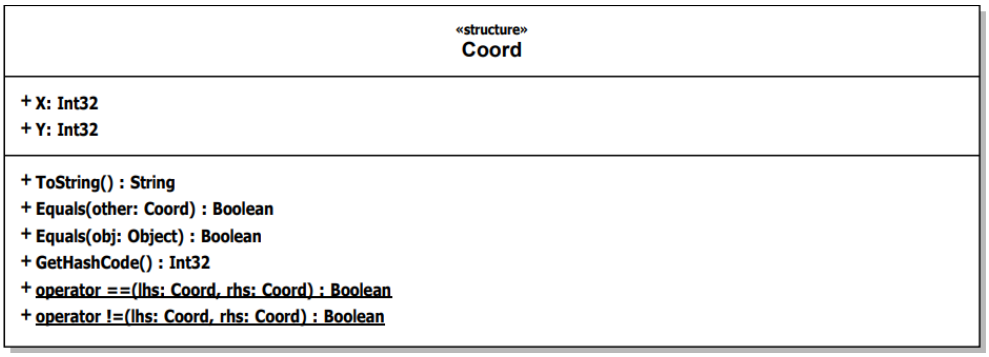
Canny Result *(Structure)*



(10) Canny Result Class Diagram

This is contained within the static class of structures in the backend library. There is no methods or functions contained within this class since it is in fact a structure. Its main function is to contain any relevant data from Canny Edge Detection.

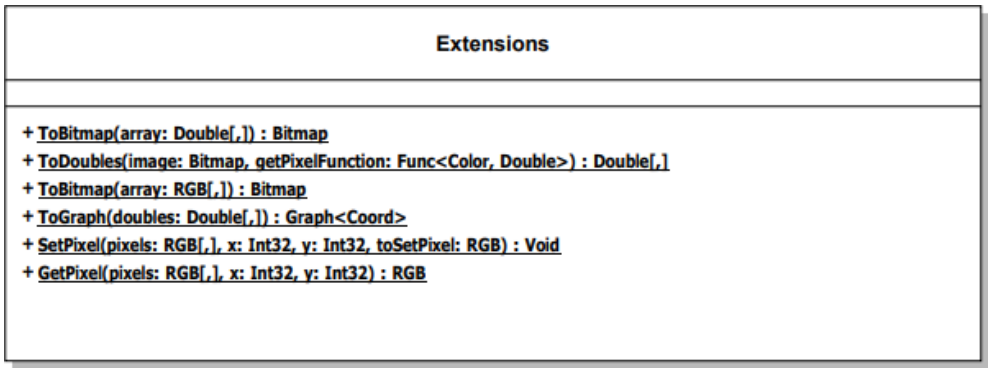
Coord *(Structure)*



(11) Coord Class Diagram

Similar to above this is contained within the static class of structures. It is used to represent a coordinate, and unlike the Canny result structure, it does contain methods. The main of which are the operator overloads. This allows me to directly compare two different coordinates instead of constantly comparing each dimensions. The other hash codes and alike are used when a dictionary is required in order for them to be converted into a hash code. Finally the ToString method is mainly used during testing however it can also be used to inform the user of a coordinate that they are interacting with.

Extensions *(Class)*



(12) Extensions Class Diagram

This class is responsible for altering the default behaviors of certain inbuilt classes in C as well as extending functionality to make it easier to maintain my code. The way in which this is achieved is through the use of the this keyword. It will allow me for example to be to .ToBitmap on a 2D array speeding up the process instead of calling separate methods.

The first of the extensions that will be implemented is the ToBitmap extension. As stated above it allows me to call ToBitmap on any given 2D double array, it then converts it to a bitmap and returns said bitmap. It does this through taking each pixel in turn and then using the Utility method to bound them within the allowed range of a pixel.

Another of the extensions contained within this class is the ToDoubles extension which is essentially the inverse of the ToBitmap extension. This acts upon a Bitmap to convert it to a 2D double array. This extension however requires a method to be passed in which is then used to work out how to get the values of the pixel to a single 0 -> 255 value. The function which is passed in must take a Color object and return a single double value.

The third extension contained is essentially a clone of the first one however this acts upon the EGB structure. This is important due to the fact that the RGB structure contains more than 1 value which means that using it I can reconstruct the original image using the R, G and B channels in the structure. Apart from this difference the two extensions are the same.

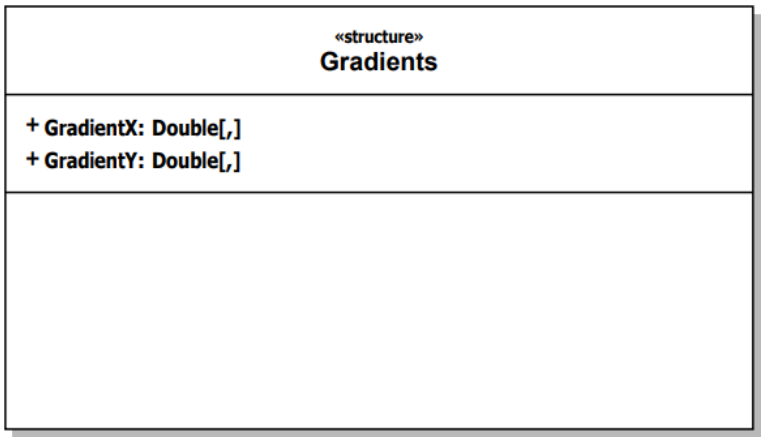
The ToGraph extension contained within this class is very important as it is how after the Canny edge detection I will convert the result to a graph which can then have a traversal algorithm used upon it. This extension acts upon a 2D double array. In order to gain the pixels around a given pixel in a double array the GetKernel function is used, this is explained in the Kernel class. Once the kernel of the pixel is created we can work out the relative coordinates through the use of modulus arithmetic.

$$\text{Where } X = x + (i\%3) - 1 \text{ and } Y = y + (i/3) - 1$$

Once this has been calculated more maths is done however this is contained within the graph class so see there for more information.

The next two structures go hand in hand so I will talk about both of them together, due to the image processing and the way in which I am accessing the arrays which contain the data I feel that it would be easier if I could alter my RGB[,] and directly access it with the .setPixel and .getPixel methods since this is how it is used in the bitmap class. Therefore it would make it easier to use the two interchangeably instead of accessing it using direct accessing then changing the value. The way in which they both work is that in the function call an x and y coordinate is passed as well as a colour if it is the set pixel version. Then it changes or returns the value.

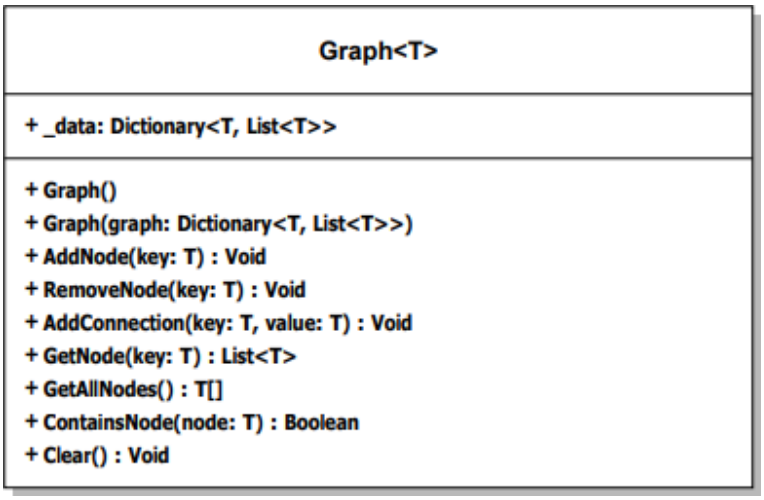
Gradients *(Structure)*



(13) Gradients Class Diagram

This is a structure located in the back end library of my project, its main function in the program is store the result of the gradient calculations. As with many of my structures in this code, there is no methods contained within.

Graph *(Class)*



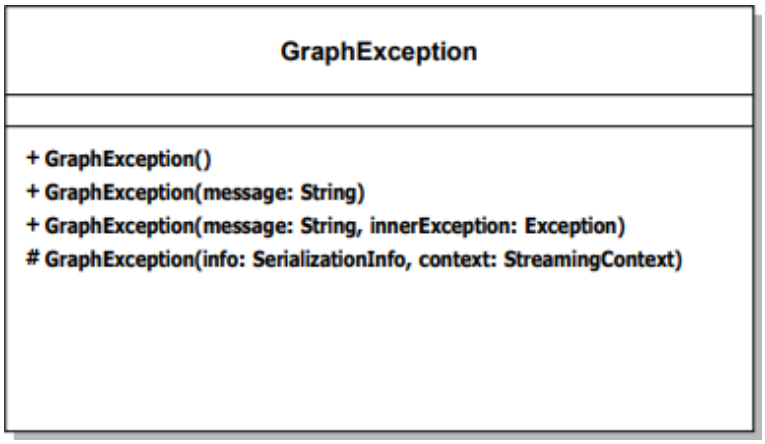
(14) Graph Class Diagram

The graph class is located in the backend portion of the project. Its responsible for holding all node data corresponding to a graph. The general implementation of this graph class is rather basic however it is responsible for allowing the majority of the program to work. All of the methods inside the class are generics. This means that a graph of any data type can be created. This especially usefully in my case because it allows me to have the nodes as Coordinate structures meaning that I can easily get its corresponding point on a map image.

The methods included in this class are as follows:

- AddNode - As the name suggests this is responsible for adding a node to the graph. You cannot add neighbors in the same function, you pass in the node name.
- RemoveNode - This is the partner method to the one above which removes a node from the graph deleting all of its neighbors connections.
- AddConnection - This ties an existing node to the on the graph to an existing node. This allows us to tell where a node can go to from its current one.
- GetNode - This allows the program to fetch a single node from the graph. It will only return connections on the node.
- GetAllNodes - Returns all nodes in the graph including those, if they exist, which have no neighbors.
- ContainsNode - Allows the program to easily check if a node exists, this is an optimisation to allow the program to run faster and at a higher level of abstraction.
- Clear - Clears the node

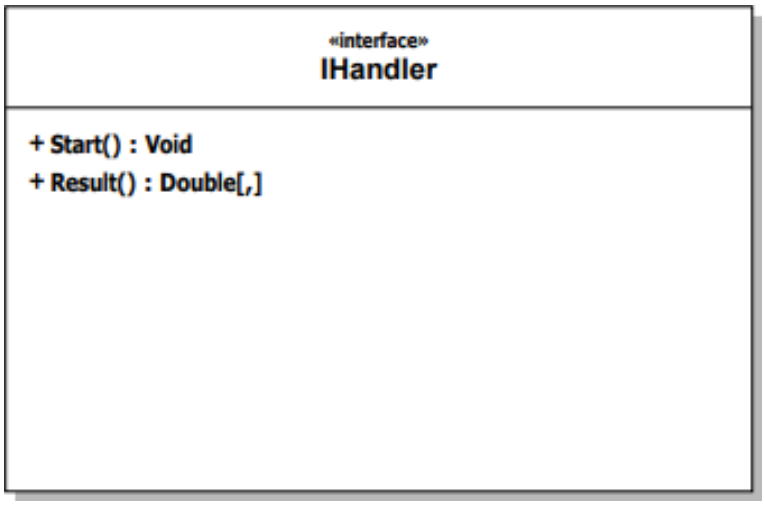
Graph Exception *(Exception)*



(15) Graph Exception Class Diagram

Located in the backend of the project this is a specific exception used to differentiate errors in the front end. This inherits the methods of the base exception class. This is the same as all other custom exception classes. They exist to allow me in the front end to show special error messages.

IHandler *(Interface)*

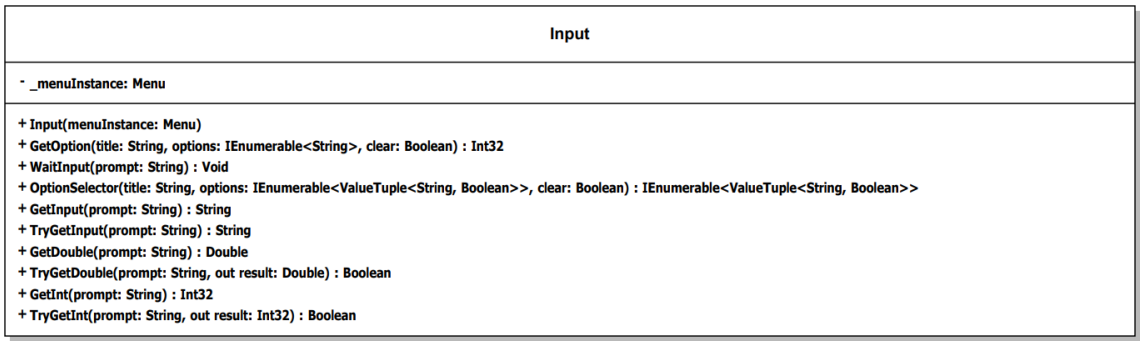


(16) IHandler UML Diagram

This is located in the backend portion of the program. IT itself has no code due to the fact that it is an interface and is soule use is to cut down on code during the selection between async or synchronous edge detection. It requires all child classes to implement the functions, Start and Result. These will be used to as the names suggest start the process and get the result of the process.

The advantage to using an interface here means that the object can be cast to a simplified version meaning that two different classes can be assigned to the same variable removing the need for guard clauses.

Input *(Class)*



(17) Input Class Diagram

This is responsible for allowing the user to interact with the program and is an integral part. It is located in the front end of the project due to the fact that it is dealing with the user input. There are several functions in this class which act in different ways.

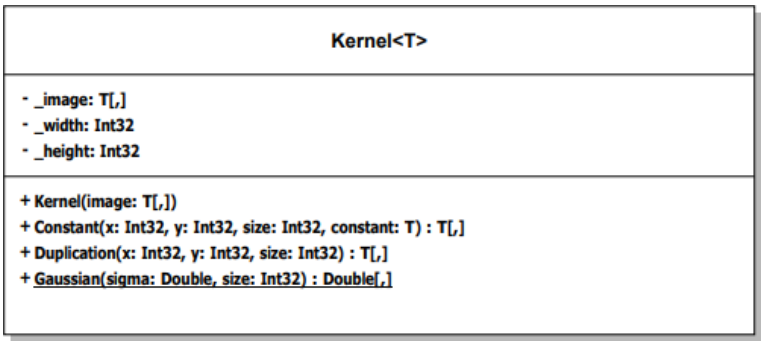
One of the integral functions of the program is to be able to get the user to select an option from a list of options. The function works by taking an IEnumerable of strings which are the options. Once the user has selected an option th function returns the 0-indexed reference to the option that was selected.

The WaitInput method is soul's used for waiting for the ser to press any key to move on. This can be used when waiting for the user to read something without there having to be an arbitrary time delay.

The OptionSelector method is used in the setting section of the program. This method will take an IEnumerable of tuples which are used for settings. It allows the user to press the enter or space-bar key which changes the boolean value of a string. This can be used to change the "SaveToZip" file. Once the user has completed all of their setting changes and choses to exit the function will return the same IEnumerable with the changed boolean values.

The following functions GetInput, TryGetInput, TryGetInt, TryGetDouble, GetInt and GetDouble are all used as the name suggests to get user input. The Try version of the methods instead return a boolean if the getting of the input was successful. These echo the double.TryGet in c however instead they are tied into the custom menu system of the program. The non Try versions will throw an exception if they are unsuccessful in getting the input which will need to be handled further up the call stack.

Kernel *(Class)*



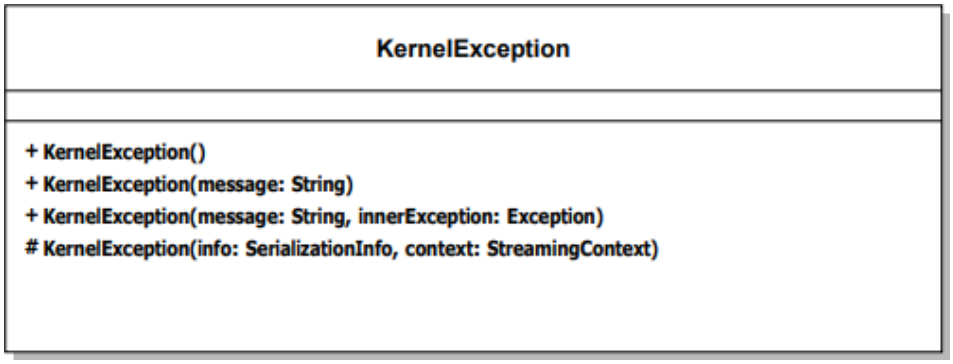
(18) Kernel Class Diagram

This is the class which is responsible for generating image kernels from a supplied image. This is also writer with generics meaning that the kernel will be able to be generated from and 2D array. This was especially usefully when it came down to using both doubles to represent black and white pixels and Color objects which represent an RGB pixel.

This class implements two types of fetching the kernel, the first of which is Constant and the second of which is Duplication. The first method, duplication runs the exact same as the second method, it will take in the center of the kernel and the size. It will run a loop over the "area" of the desired kernel if the coordinate is inside the bound of the image then it will use that in the kernel however if it is not in the duplication version it will take the initial kernel and duplicate that. This is explained in more detail in the analysis section when I was investigation this field. The constants version will just use a constant value which in this case is 128 which is a grey colour in black and white.

The resulting kernel is then converted into a matrix and retuned in order to be convoluted with the processing kernel. The only exception to this is the gaussian kernel. This does not take in an image, instead it applies the 3D statistical gaussian distribution to a matrix producing a 3D Gaussian distribution matrix of a given size. This is then returned to be used in the Canny edge detection section.

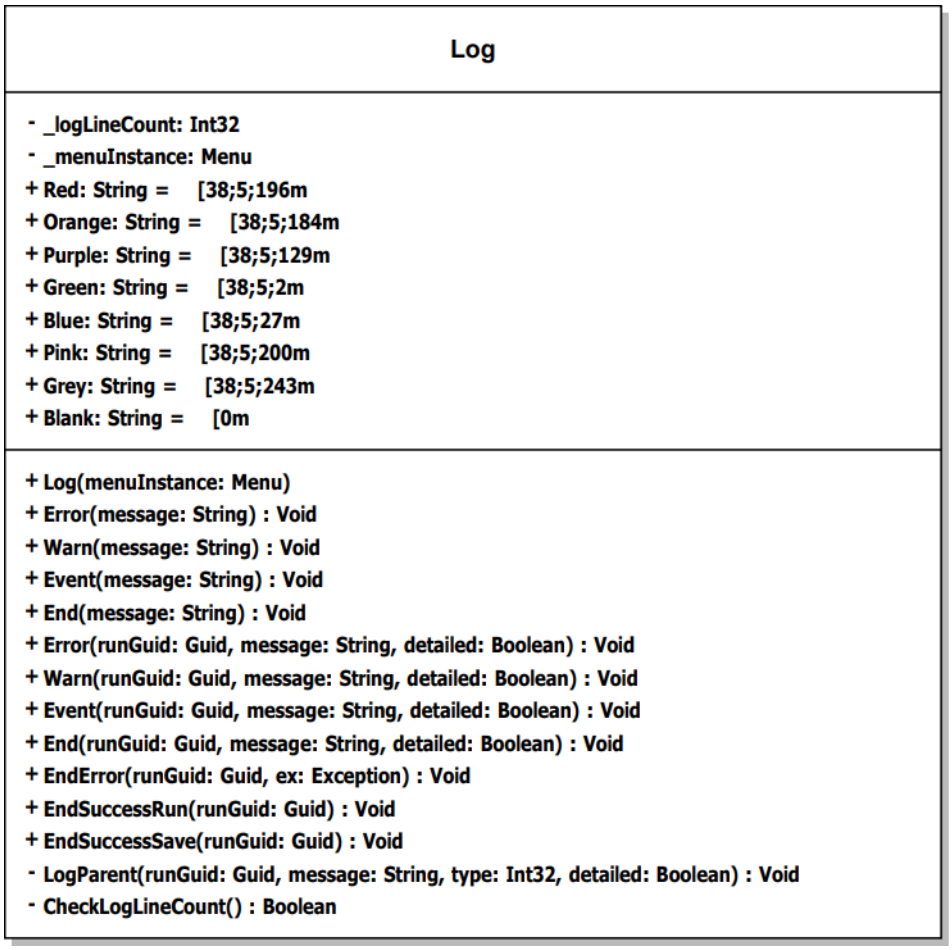
Kernel Exception *(Exception)*



(19) Kernel Exception Class Diagram

Located in the backend of the project this is a specific exception used to differentiate errors in the front end. This inherits the methods of the base exception class. This is the same as all other custom exception classes. They exist to allow me in the front end to show special error messages.

Log *(Class)*



(20) Log Class Diagram

The Log class is located in the front end of the program due to the fact that it is interacting with the user, there is also a Logger class, this is responsible for the outputs and is located back end and is explained below. Most of the methods contained within this class do the same thing by printing a log message however there are two distinct versions.

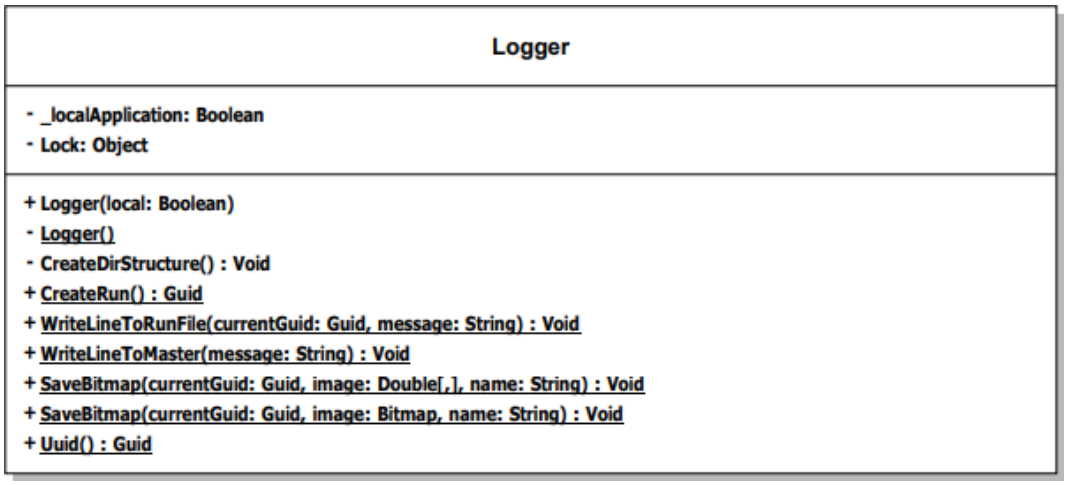
The first of are the methods which contain only a message as an argument, these logs are not displayed to the user and are instead sent directly to a save file to be logged. I chose to do this because it will allow me to hide some complexity to the user.

The second version of the log is the one which if detailed logging is enabled will be shown to the user. There is a case in this method however in which the setting can be overridden. This means that for important messages like the beginning and end will be shown even if the user wants it or not.

The end success and end error functions are used to end a run which as stated above overrides the logging and forces it to the side. It also saves the outputs and manages all of the file structures.

There are several static properties of the class these are the ascii colour codes which are used to make the user interface more palatable to the user. The over very important property is the current line, this allows the program to tell how far down the log side it is and to see if it needs to be cleared. It is incremented every time a new line is logged.

Logger *(Class)*

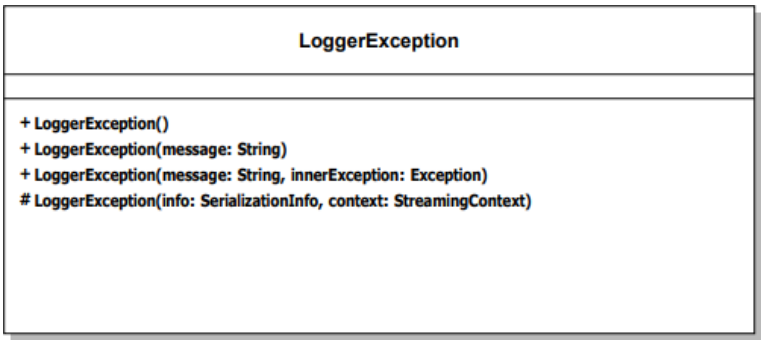


(21) Logger Class Diagram

This is the part of the code which is responsible for saving all of the users files which have been generated though the program. Since the methods contained within are rather generic in how they could function I will bullet point the general idea behind each one.

- CreateDirStructure - As the name suggests this creates the desired directly structure for the user depending on how the designer wishes it to be laid out. This checks if the directories exist and if they don't then it creates them. This is not responsible for creating any of the files used in the program like settings.conf or master.log
- CreateRun - The use for this is to create a run, this will create a directory, in this program, inside the runs folder. It also will create a run log file with a random guid. This guid is then returned to be used in the rest of the program and when logging.
- WriteLineToLogFile - Writes a line to a log file given a UUID / Guid. It will locate the file and append it to the end.
- WriteLineToMaster - This will append a line to the master log file which is located in this case inside the logs directory.
- SaveBitmap - Both the functions here will save a bitmap into a folder given a GUID, if the double array is supplied then it converts to a bitmap then it is saved. If a bitmap is supplied the bitmap is saved.
- Uuid - Returns a new UUID also known as a GUID

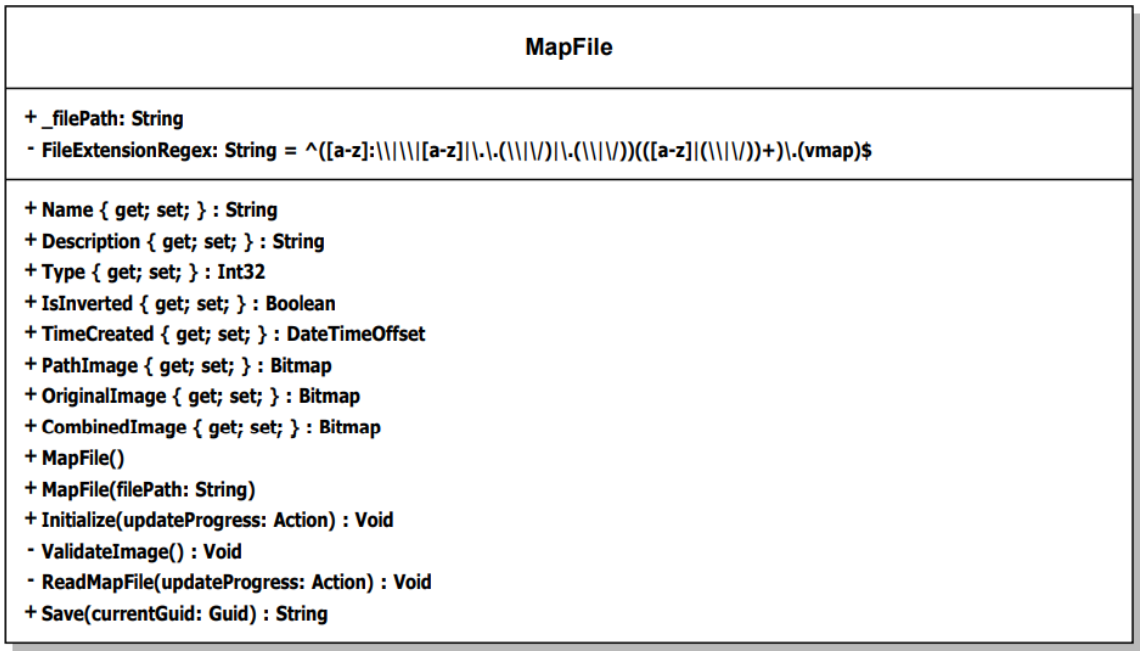
Logger Exception *(Exception)*



(22) Logger Exception Class Diagram

Located in the backend of the project this is a specific exception used to differentiate errors in the front end. This inherits the methods of the base exception class. This is the same as all other custom exception classes. They exist to allow me in the front end to show special error messages.

Map File *(Class)*



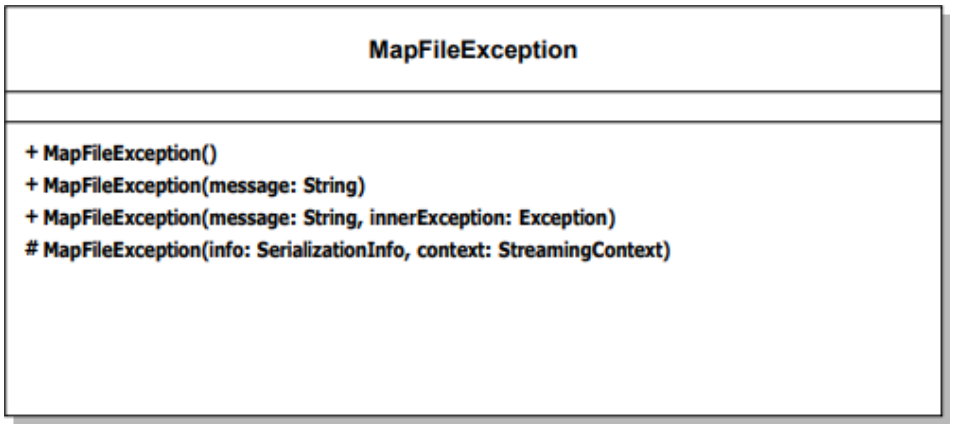
(23) Map File Class Diagram

This is located in the backend library of the project. Its function is to store and create map save files.All of the methods contained within with the suffix get; set; are actual properties on the class which are used to store information about the save file.

The file extension regex is part of the properties is used to validate whether a given file is actually a map file. This comes in usefully when recalling a map from a save file. This is when the method Initialize comes in useful, it takes in the file path, validates it and then according to a set of rules will read in the custom binary file, the actual reading of the map fle is done in the ReadMapFile method. However this is called from within the validate function since we only want this to run if the file is valid.

If at any point it encounters an error it will throw an exception which tells the user what the issue is. The final function of this class is saving the map file. This is done once all processing has been done, it saves it using a binary file. It is important to me that the file was all in one place and not split up allowing it to be transferred easily.

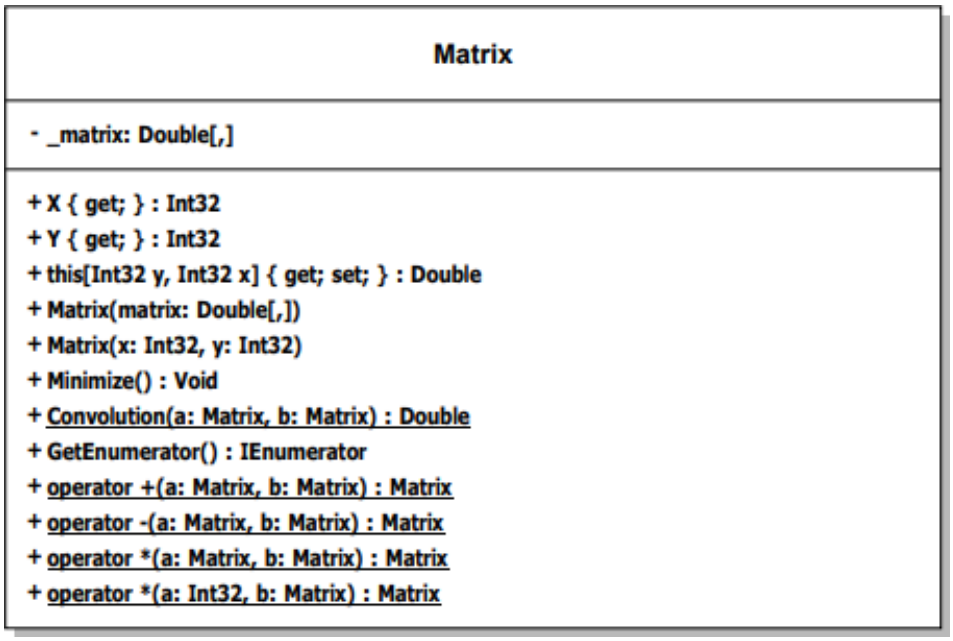
Map File Exception *(Exception)*



(24) Map File Exception Class Diagram

Located in the backend of the project this is a specific exception used to differentiate errors in the front end. This inherits the methods of the base exception class. This is the same as all other custom exception classes. They exist to allow me in the front end to show special error messages.

Matrix *(Class)*



(25) Matrix Class Diagram

This is located in the backend portion of my program. Again as above the methods of the class which are responsible for storing data to do with the program are denoted with get; set; . The X and Y properties of the matrix class denote the dimensions of the matrix.

There is a direct array access to the data in the matrix. This was so that when assigning the gaussian distribution to a new matrix I would have to have stored all of the values in a temporary 2D array just for it to be assigned to a matrix. Therefore I made the decision to allow my program to directly access the cells of the matrix through the [] notation. It is both Get and Set however it could be made internal to keep the library and the front end even more separated.

There is a rather large amount of operator overloading in this section which perform the various numerical operations of the matrix. These are as follows:

Matrix Addition

$$\mathbf{A}+\mathbf{B}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

Matrix Multiplication

$$\text{Let } \mathbf{A}=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B}=\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

$$C = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & a_{11}b_{12} + \cdots + a_{1n}b_{n2} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ a_{21}b_{11} + \cdots + a_{2n}b_{n1} & a_{21}b_{12} + \cdots + a_{2n}b_{n2} & \cdots & a_{21}b_{1p} + \cdots + a_{2n}b_{np} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & a_{m1}b_{12} + \cdots + a_{mn}b_{n2} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

Scalar Multiplication

$$\lambda\mathbf{A} = \lambda \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix} = \begin{pmatrix} \lambda A_{11} & \lambda A_{12} & \cdots & \lambda A_{1m} \\ \lambda A_{21} & \lambda A_{22} & \cdots & \lambda A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda A_{n1} & \lambda A_{n2} & \cdots & \lambda A_{nm} \end{pmatrix}$$

Matrix Convolution

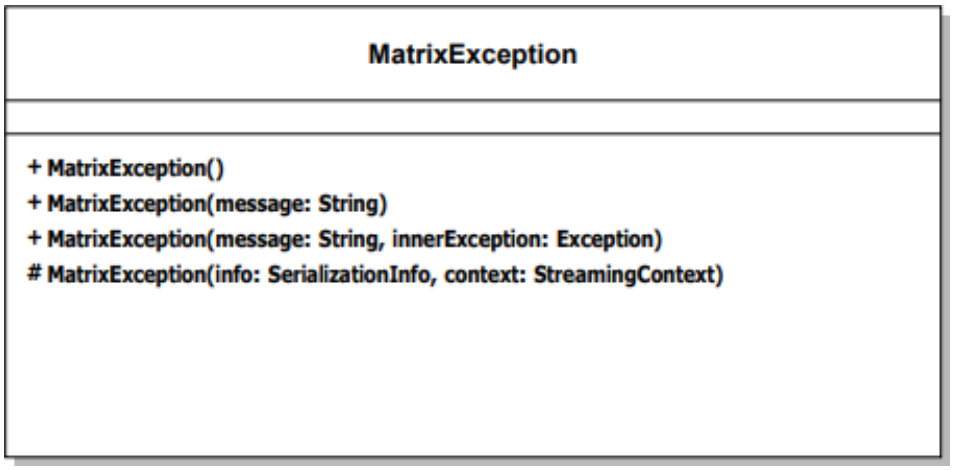
This is very important for my program since this is the section which allows all of the image processing to function. The general form of matrix convolution is as follows:

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} * \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{(m-i)(n-j)} y_{(1+i)(1+j)}$$

Matrix Minimisation

This is more of a custom made algorithm. It runs over every entry in the matrix summing their values together. This is then used to divide every element in the matrix by this sum. This means that all of the elements will sum to zero if added which can help with matrix convolution.

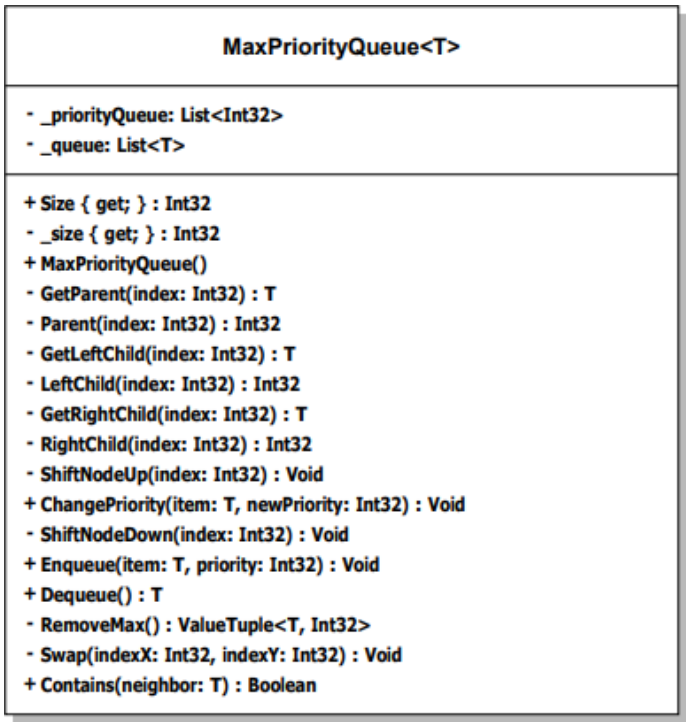
Matrix Exception *(Exception)*



(26) Matrix Exception Class Diagram

Located in the backend of the project this is a specific exception used to differentiate errors in the front end. This inherits the methods of the base exception class. This is the same as all other custom exception classes. They exist to allow me in the front end to show special error messages.

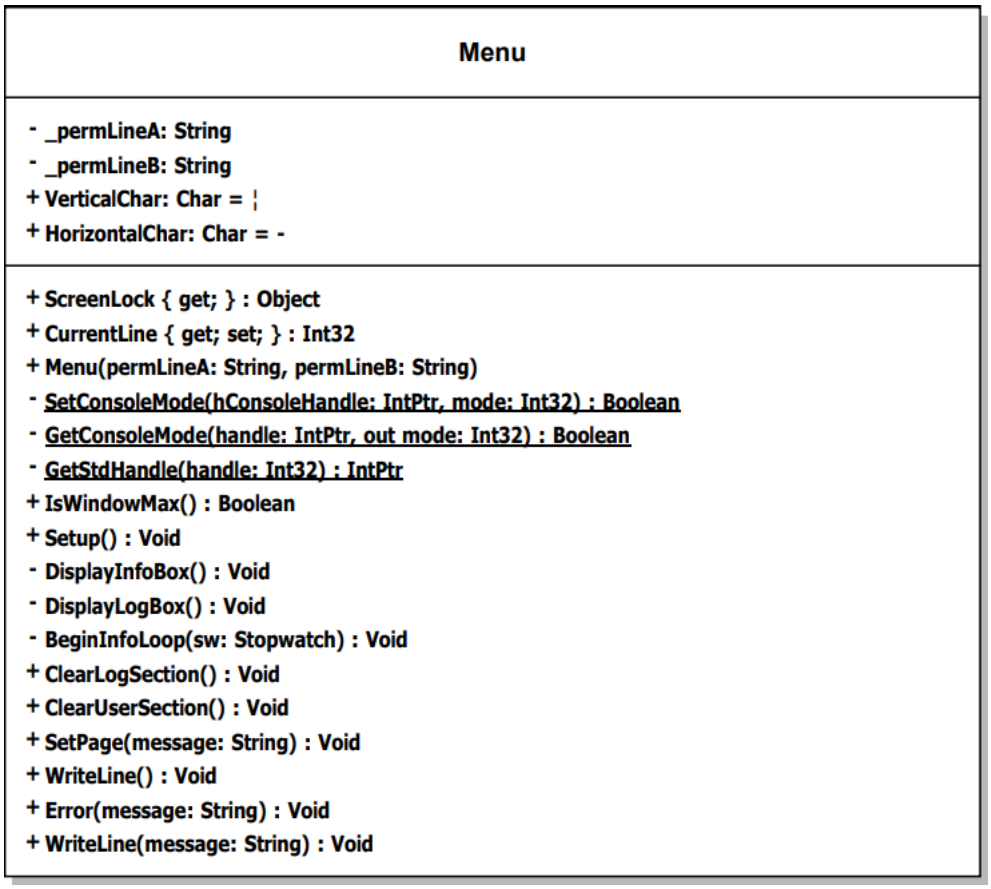
Max Priority Queue *(Class)*



(27) Max Priority Queue Class Diagram

Again located in the back end library of the project, this is a data type used in the processing and pathfinding of the map. This is identical to the Min priority queue explained below apart from the ordering of the elements in the queue. Both use a binary heap. Please refer to the min priority queue for an explication of the methods involved.

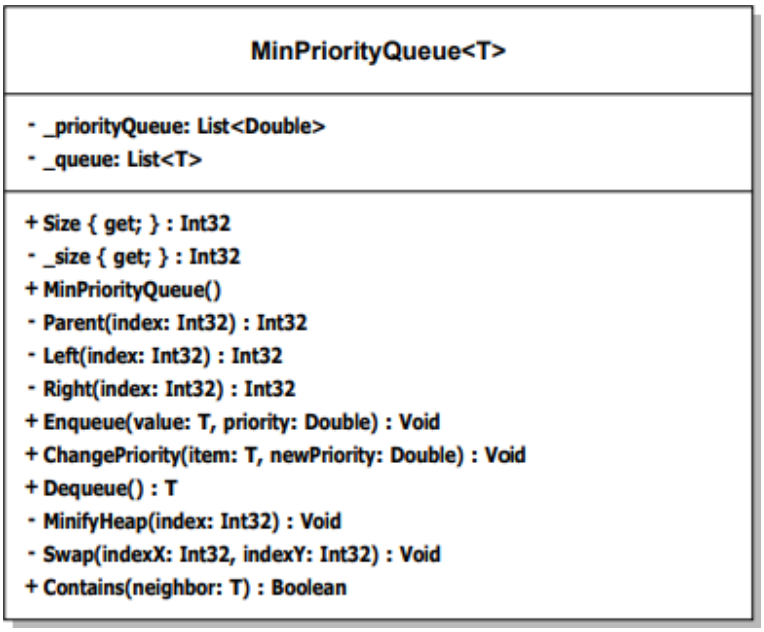
Menu *(Class)*



(28) Menu Class Diagram

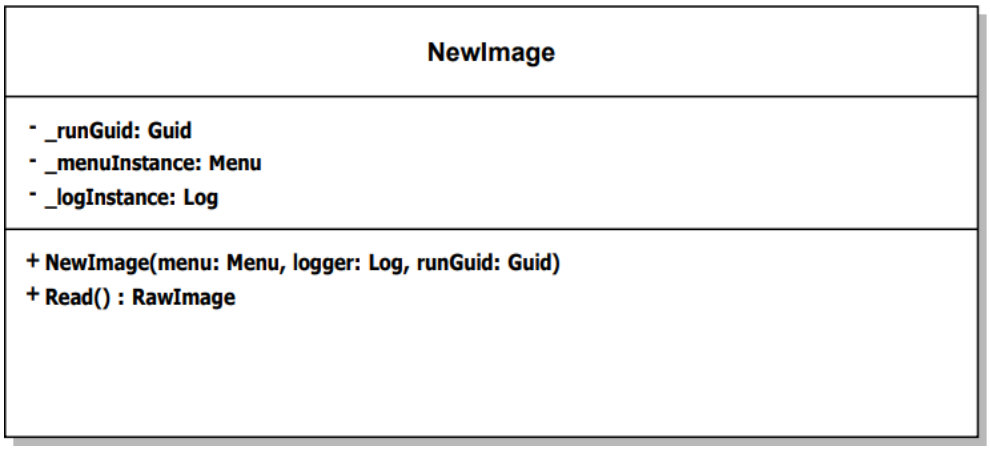
This is located in the front end portion of my project and is responsible for most

Min Priority Queue *(Class)*



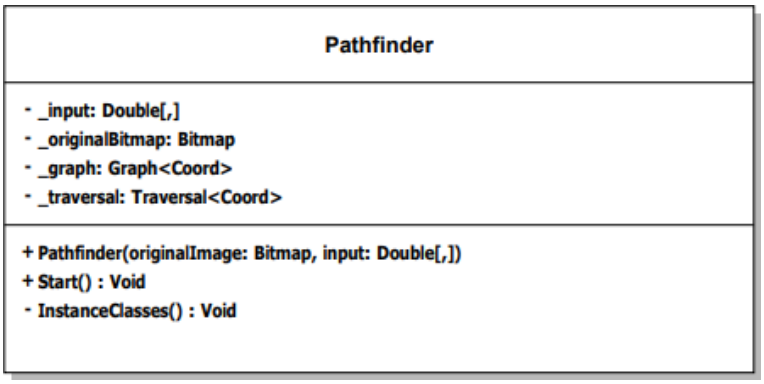
(29) Min Priority Queue Class Diagram

New Image *(Class)*



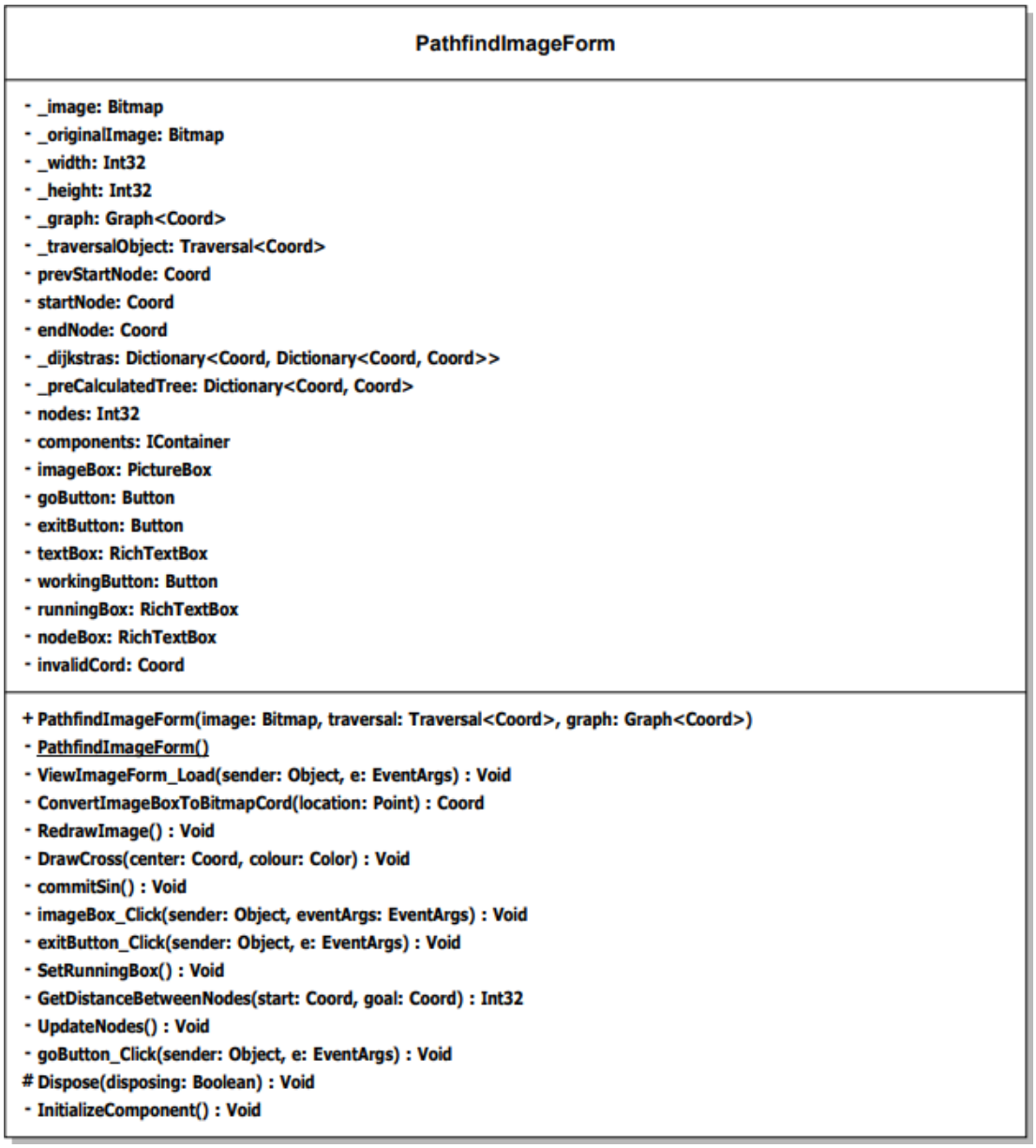
(30) New Image Class Diagram

Pathfinder *(Class)*



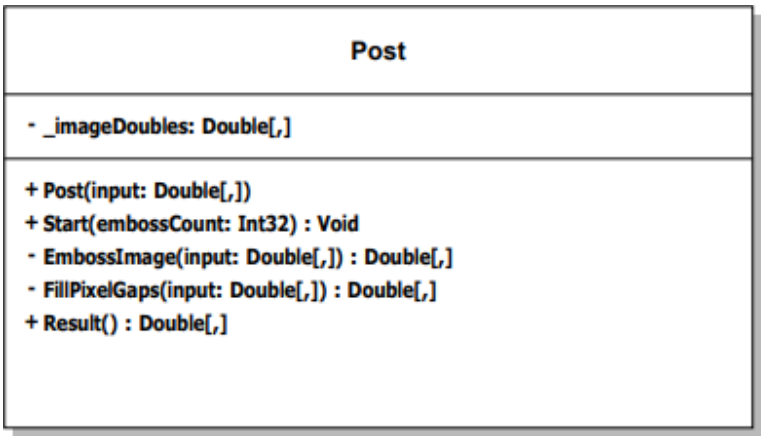
(31) Pathfinder Class Diagram

Pathfind Image Form *(Windows Form)*



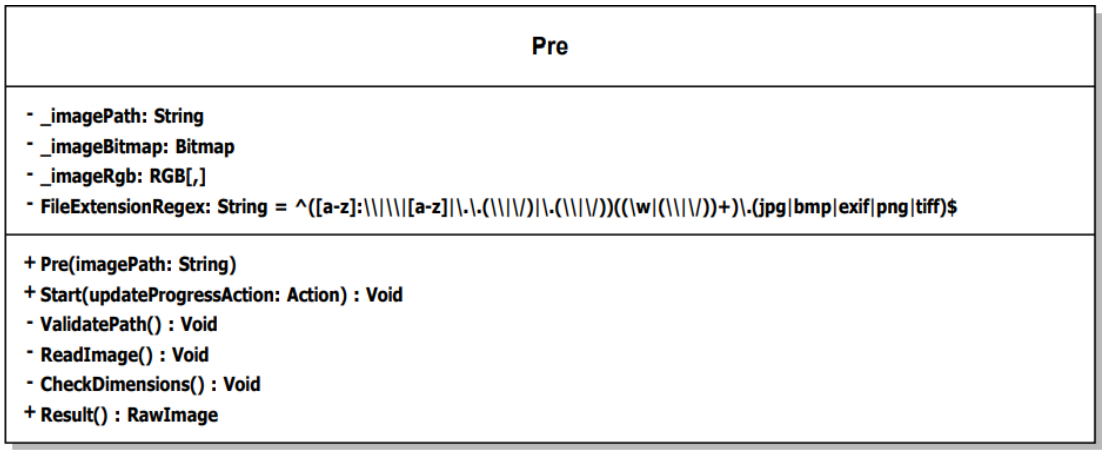
(32) Pathfind Image Form UML Diagram

Post *(Class)*



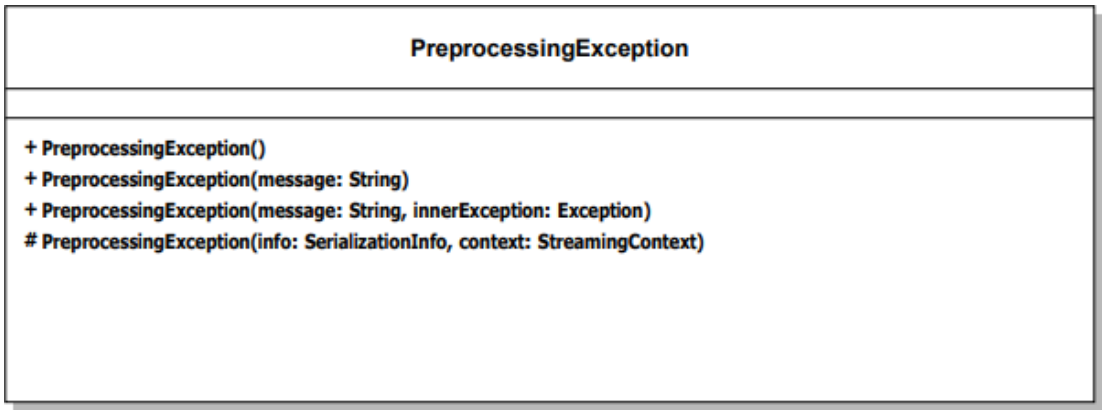
(33) Post Class Diagram

Pre *(Class)*



(34) Pre Class Diagram

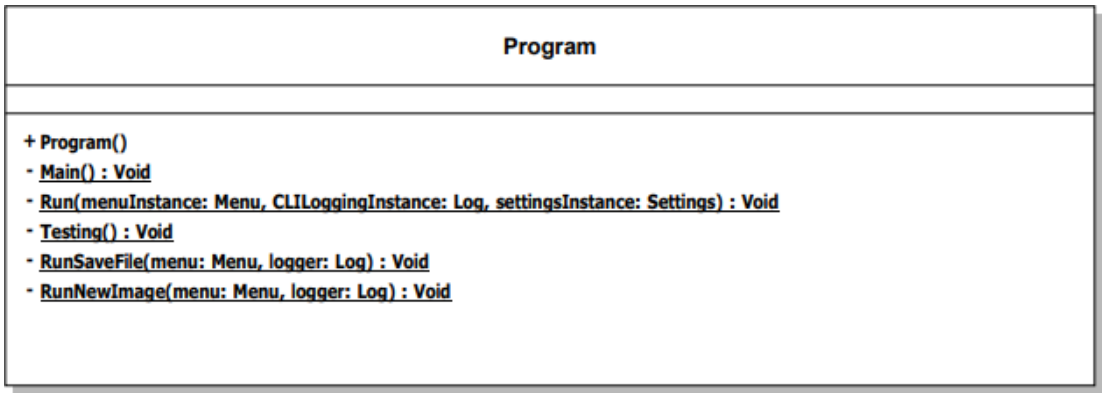
Preprocessing Exception *(Exception)*



(35) Preprocessing Exception Class Diagram

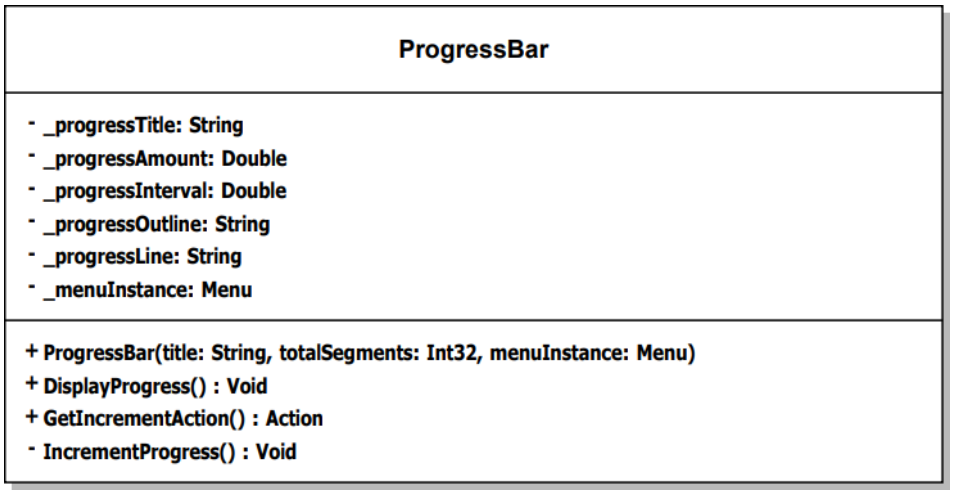
Located in the backend of the project this is a specific exception used to differentiate errors in the front end. This inherits the methods of the base exception class. This is the same as all other custom exception classes. They exist to allow me in the front end to show special error messages.

Program *(Class)*



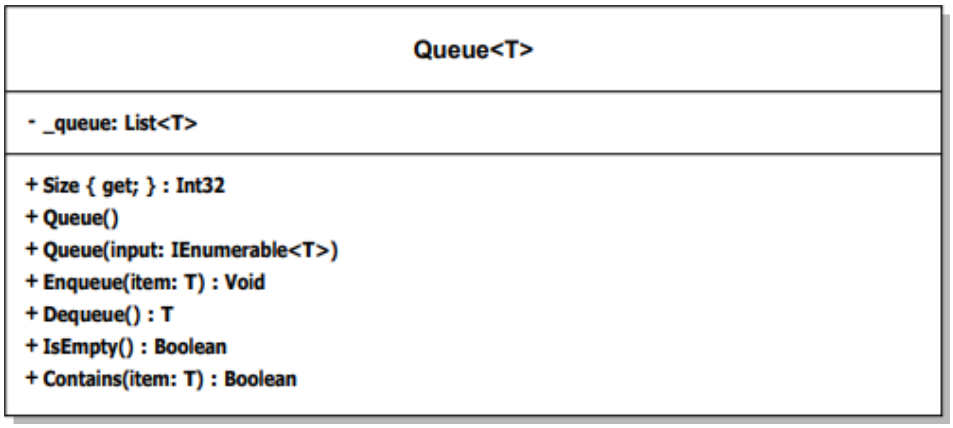
(36) Program Class Diagram

Progress Bar *(Class)*



(37) Progress Bar Class Diagram

Queue *(Class)*



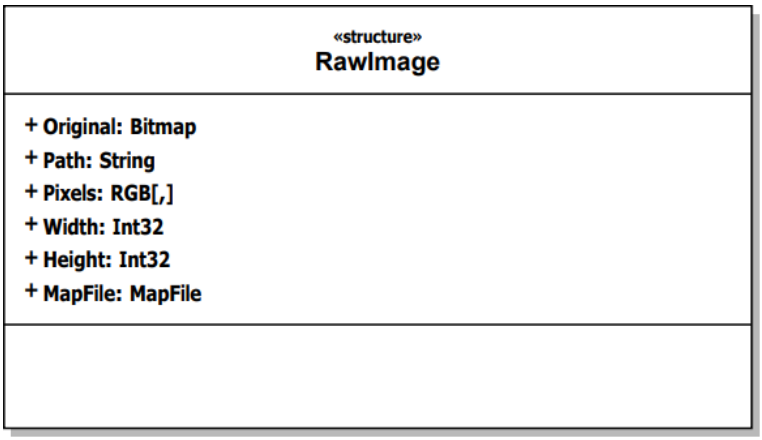
(38) Queue Class Diagram

This ADT is located in the backend library of my project. The reason which I created my own Queue class is that I needed to be able to directly access parts of it which might not be considered by the inbuilt class. It still contains the same methods as the inbuilt queue:

- Enqueue: Adds an element to the end of the queue.
- Dequeue: Removes the element from the front of the queue and returns it.
- Peek: Returns the element at the front of the queue without removing it.
- Is Empty: Returns a boolean indicating whether the queue is empty or not.
- Size: Returns the number of elements in the queue.
- Contains: Returns a boolean indicating whether the queue contains an element or not.

I allowed the queue to be instantiated with a pre existing enumerable, this allows me to create queues with nodes of a graph for example, this is not possible with the queue which was available to me by default. This is used in the traversal class when performing BFS.

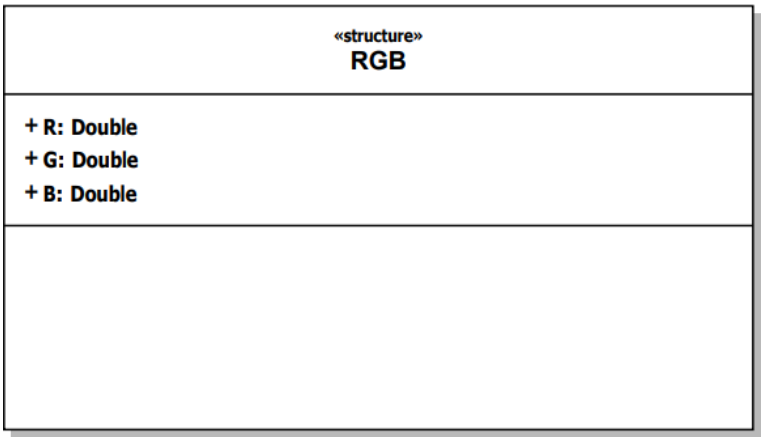
Raw Image *(Structure)*



(39) Raw Image Class Diagram

This is a structure and therefore located in the backend portion of the project. Its function is to move the results of the image about. It contains the original image which was processed, the string representation of the file path as well as the dimensions of said image. These are included to speed up mathematical calculations. It also contains a RGB 2D array which is a more mutable representation of the image. This allows it to be easily converted into other forms and have various algorithms applied to it. Finally a MapFile can be attached to this structure however in the event that no save is selected this will be null to allow the program to see that no save needs to be made.

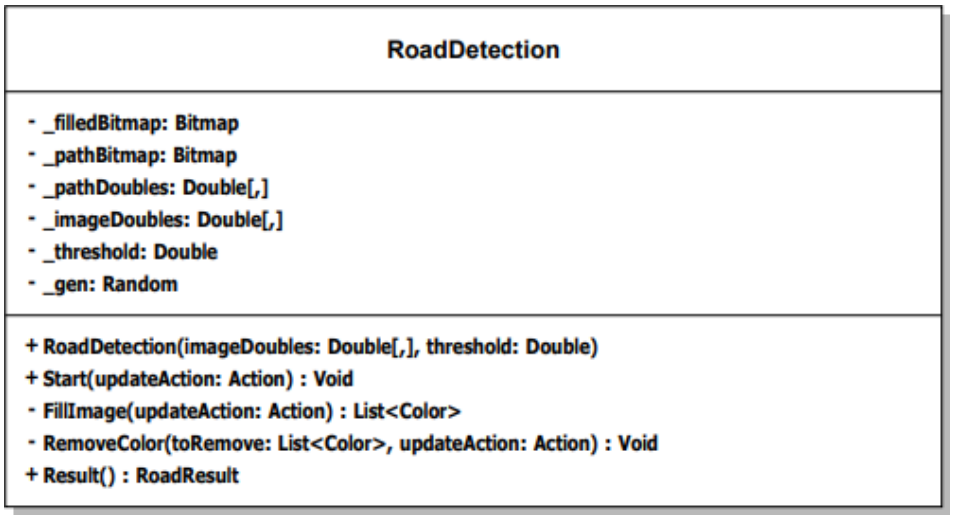
RGB *(Structure)*



(40) RGB Class Diagram

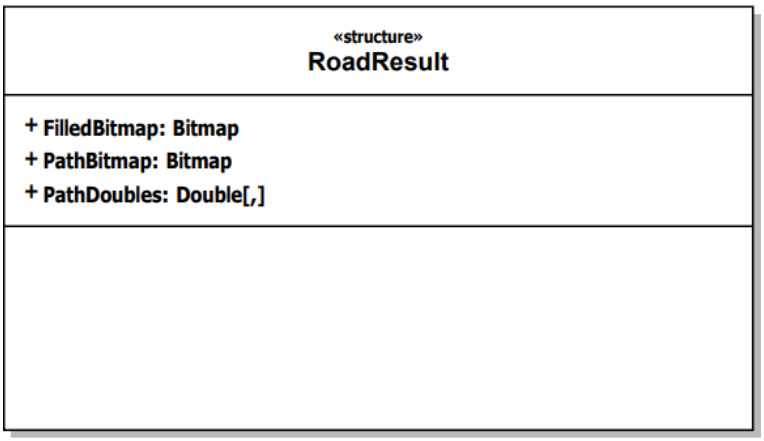
This basic structure is located in the backend library of my final solution. Whilst it does not contain much information this is used in place of the inbuilt Color class due to the limitation that the inbuilt Color class has. It contains just 3 doubles, one for each colour channel. There are overloads involving this class however they are covered by the extensions class so see there for those functions.

Road Detection *(Class)*



(41) Road Detection Class Diagram

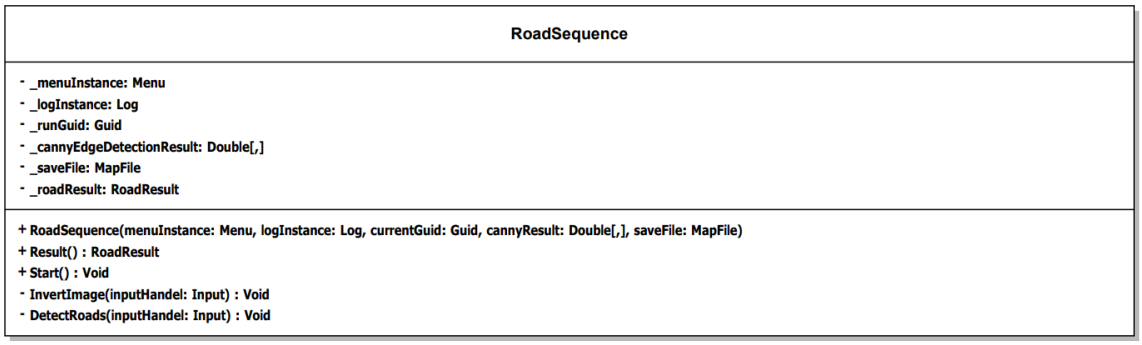
Road Result *(Structure)*



(42) Road Result Class Diagram

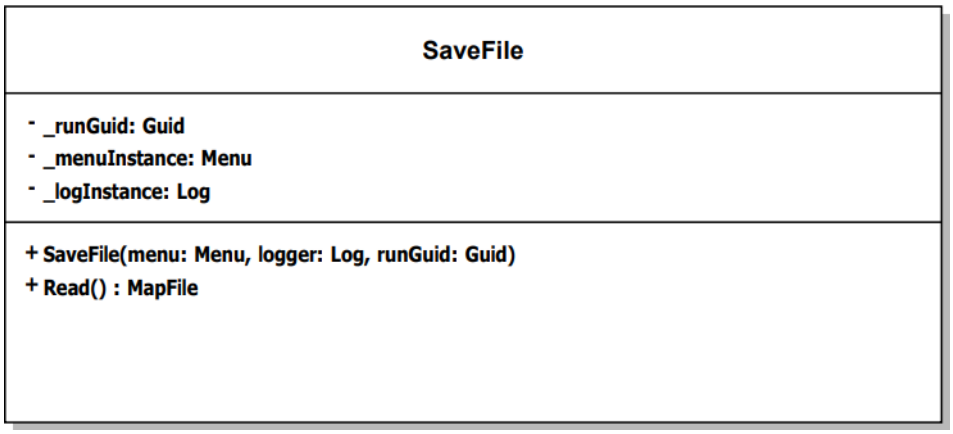
This structure is located in the backend portion of the solution, as the name might suggest it is responsible for the result of the road detection. By the stage that this is used the picking out of the roads has occurred and been calculated. The next stage is pathfinding. The filled bitmap is returned so that the user can see how the image was filled and the different section that where picked out. The path bitmap is returned to be stored in the save file if the user requests it. Finally the 2D double array is ued in the pathfinding to tell whether the selected nodes are on the path. It is also used when creating the graph to be routed.

Road Sequence *(Class)*



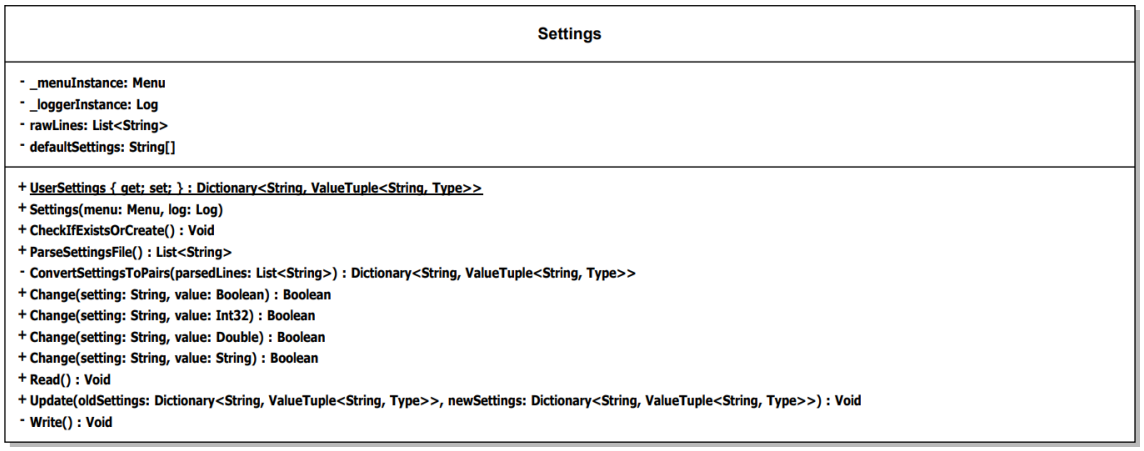
(43) Road Sequence Class Diagram

Save File *(Class)*



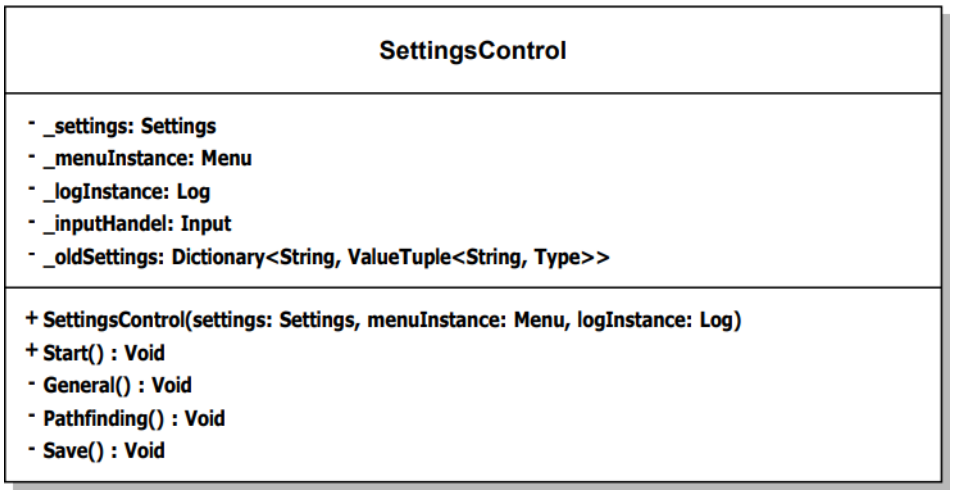
(44) Save File Class Diagram

Settings *(Class)*



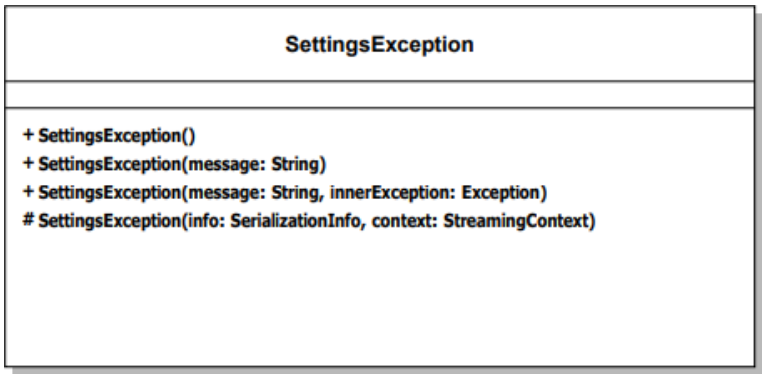
(45) Settings Class Diagram

Settings Control *(Class)*



(46) Settings Control Class Diagram

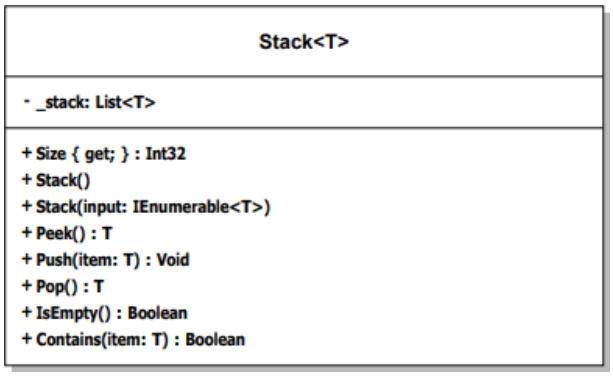
Settings Exception *(Exception)*



(47) Settings Exception Class Diagram

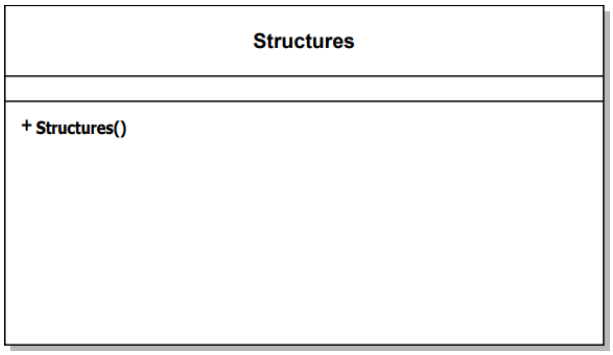
Located in the backend of the project this is a specific exception used to differentiate errors in the front end. This inherits the methods of the base exception class. This is the same as all other custom exception classes. They exist to allow me in the front end to show special error messages.

Stack *(Class)*



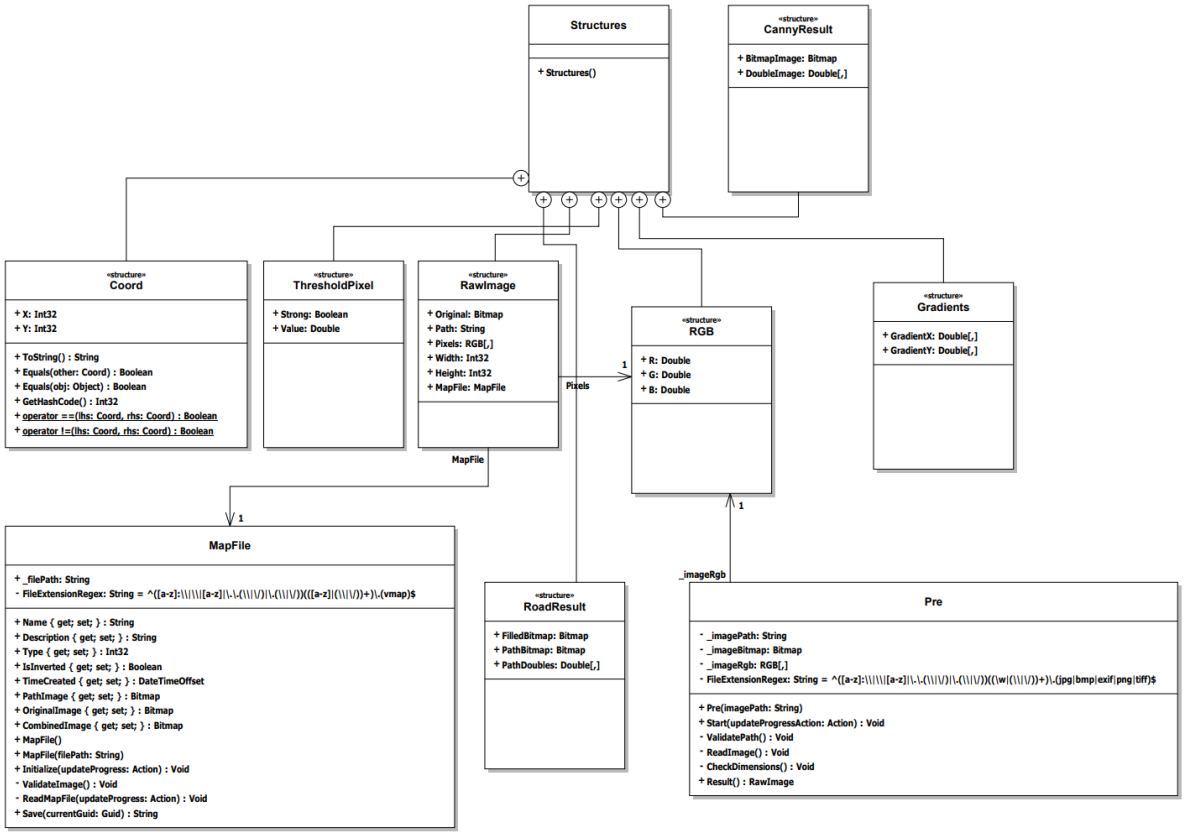
(48) Stack Class Diagram

Structures *(Class)*



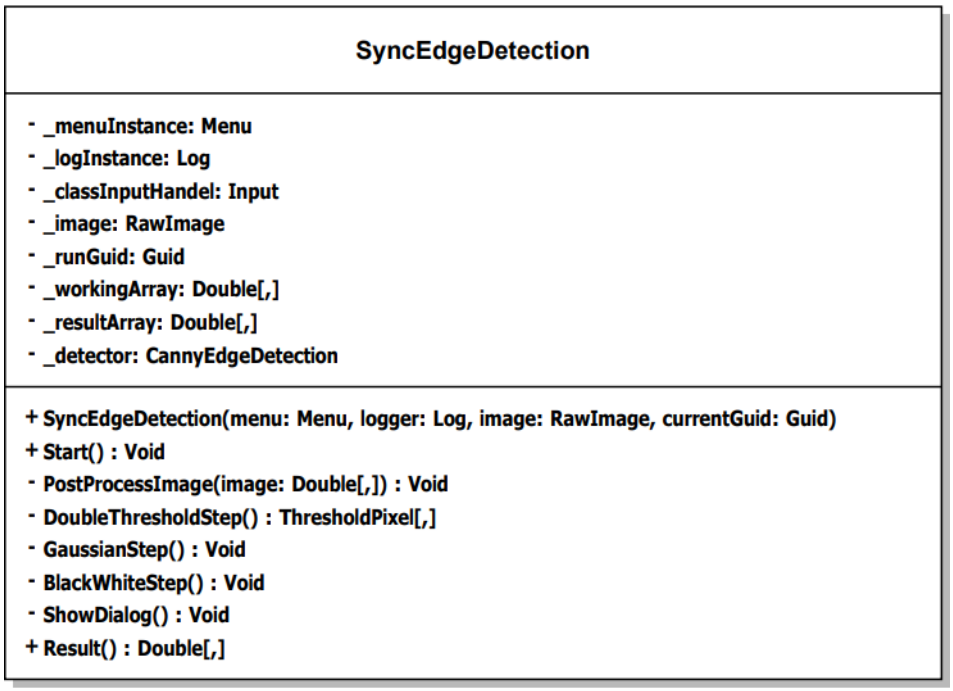
(49) Structures Class Diagram

This is located in the backend of the project, this is due to the fact that all it contains is the structures which can be seen written out below. Each of them contains public properties which can then be used in the program to move large amounts of data around without the need for massive tuples.



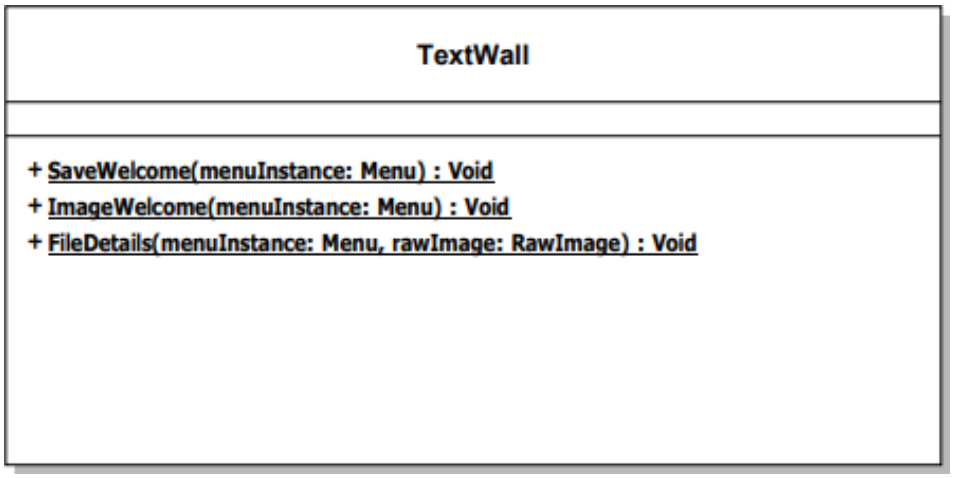
(50) Structures Class Diagram

Sync Edge Detection *(Class)*



(51) Sync Edge Detection Class Diagram

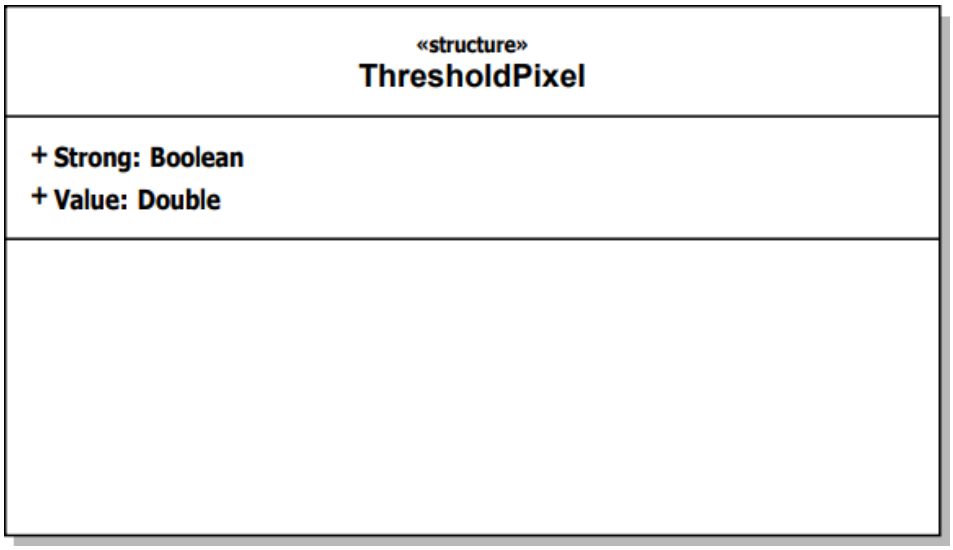
Text Wall *(Class)*



(52) Text Wall Class Diagram

This is located in the front end section of the program. It is used to send blocks of text which may be used more than once to keep the code clean and editable. The functions in here are self explanatory and send messages to the user they are not integral to the function of the program and are just separated for ease of use.

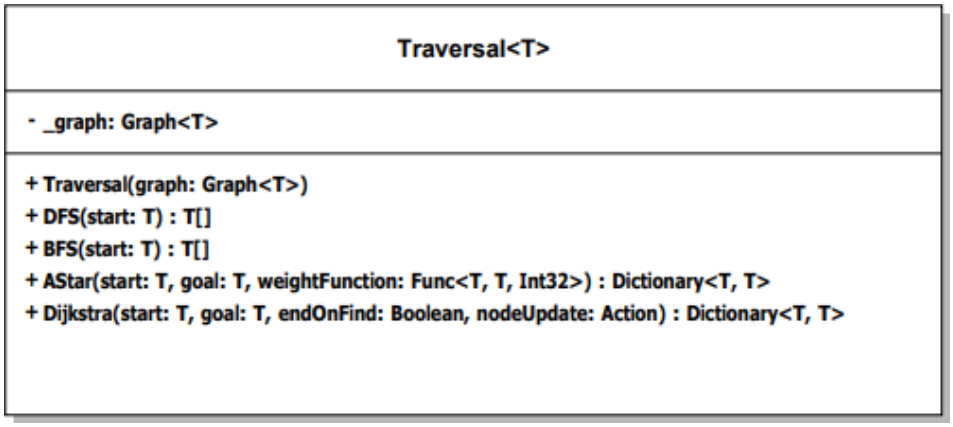
Threshold Pixel *(Structure)*



(53) Threshold Pixel Class Diagram

This is a structure and located in the backend library of the project. It is only used in the edge detection section however it is vital in the final hysteresis stage. It allows the program to tell if a pixel needs to be made strong or not. This structure therefore contains the greyscale value of the pixel it represents as a double and whether it is considered strong or not. This is represented by a boolean. The general way that this is implemented is in a 2D array removing the need to have coordinates stored in the structure itself.

Traversal *(Class)*



(54) Traversal Class Diagram

This is the class which is responsible for the functionality relating to the pathfinding of the map. This is located in the backend library of the project. It contains 4 different traversal and searching algorithms each of which will be explained below.

BFS / DFS
I have chosen to combine these two together due to the fact that they are essentially identical bar from the fact that one of them uses a stack and the other a queue. The pseudocode for each is below:

```
// DFS
procedure DFS_iterative(G, v) is
let S be a stack
S.push(v)
while S is not empty do
    v = S.pop()
    if v is not labelled as discovered then
        label v as discovered
        for all edges from v to w in G.adjacentEdges(v) do
            S.push(w)

// BFS
procedure BFS_iterative(G, v) is
let S be a queue
S.push(v)
while S is not empty do
    v = S.pop()
    if v is not labelled as discovered then
        label v as discovered
        for all edges from v to w in G.adjacentEdges(v) do
            S.push(w)
```

Dijkstra's

```
function Dijkstra(Graph, source):
    dist[source] ← 0                           // Initialization
    create vertex priority queue Q
    for each vertex v in Graph.Vertices:
        if v  source
            dist[v] ← INFINITY                 // Unknown distance from source to v
            prev[v] ← UNDEFINED                // Predecessor of v

        Q.add_with_priority(v, dist[v])


    while Q is not empty:                      // The main loop
        u ← Q.extract_min()                    // Remove and return best vertex
        for each neighbor v of u:              // Go through all v neighbors of u
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
```

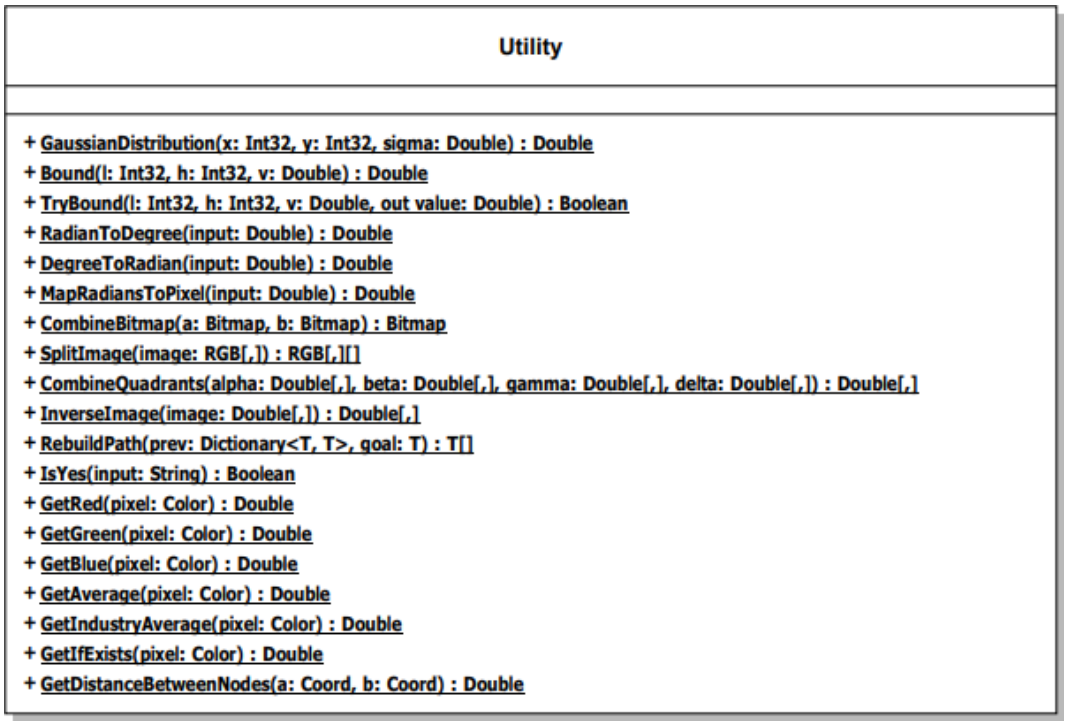
```
                dist[v] ← alt
                prev[v] ← u
                Q.decrease_priority(v, alt)

    return dist, prev
```

A* Search

```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path

// A* finds a path from start to goal.
// h is the heuristic function. h(n) estimates the cost to reach goal from node n.
function A_Star(start, goal, h)
    // The set of discovered nodes that may need to be (re-)expanded.
    // Initially, only the start node is known.
    // This is usually implemented as a min-heap or priority queue rather than a hash-set.
    openSet := {start}

    // For node n, cameFrom[n] is the node immediately preceding it on the cheapest path from start
    // to n currently known.
    cameFrom := an empty map

    // For node n, gScore[n] is the cost of the cheapest path from start to n currently known.
    gScore := map with default value of Infinity
    gScore[start] := 0

    // For node n, fScore[n] := gScore[n] + h(n). fScore[n] represents our current best guess as to
    // how cheap a path could be from start to finish if it goes through n.
    fScore := map with default value of Infinity
    fScore[start] := h(start)

    while openSet is not empty
        // This operation can occur in O(Log(N)) time if openSet is a min-heap or a priority queue
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        for each neighbor of current
            // d(current,neighbor) is the weight of the edge from current to neighbor
            // tentative_gScore is the distance from start to the neighbor through current
            tentative_gScore := gScore[current] + d(current, neighbor)
            if tentative_gScore < gScore[neighbor]
                // This path to neighbor is better than any previous one. Record it!
                cameFrom[neighbor] := current
                gScore[neighbor] := tentative_gScore
                fScore[neighbor] := tentative_gScore + h(neighbor)
                if neighbor not in openSet
                    openSet.add(neighbor)

    // Open set is empty but goal was never reached
    return failure
```

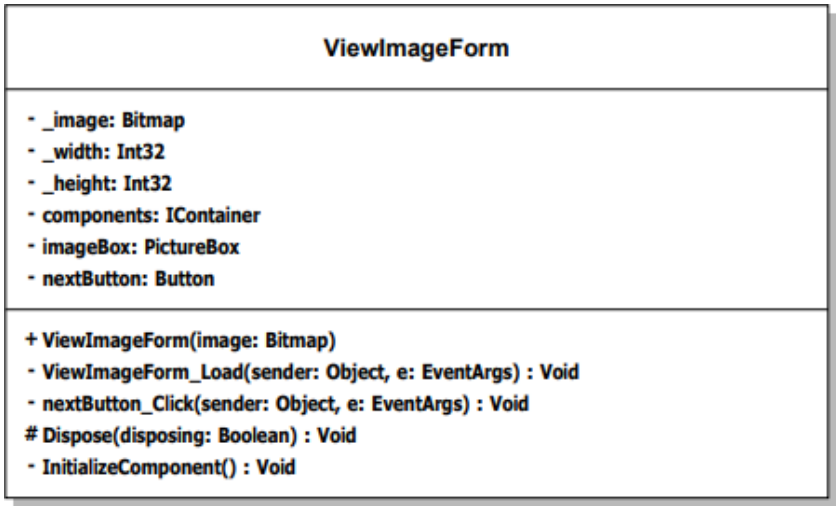
This is all that is contained within the traversal algorithm, the only properties on this class is te graph which is having the algorithms applied to it. I chose to instantiate the class with the graph instead of passing it in as a reference to keep the classes couples and to avoid data from getting altered unexpectedly.

Utility *(Class)*



(*55*) Utility Class Diagram

View Image Form *(Windows Form)*



(56) View Image Form UML Diagram

This is located in the front end of the program and is responsible for showing the image to the user. This is all that the form does and has the image on the left and the continue button on the right. Inside the form is code to dynamically resize the elements of the form to fit the screen. The only function written by me is the nextButton which closes the form and resumes the program.

2.5 File Structure

2.5.1 Program Files

I have taken the decision of splitting my project into a front end and a back end. This came from the interview talking to my end user. They felt that having multiple different ways of interacting with the program. In order to do this I will separate the front end and the back end processing. This is also considered best practice when writing software since it keeps it maintainable.

One of the requirements if I am going to follow this is that there are no user input calls made in the back end portion of the code. However sometimes I may want the user to be kept update as to what is going on in the program and give them a rough estimation of how long is lest, to do this I will employ the use of actions. With actions this means that should I want to extend this in the future to use razor pages for example this can be done with minimal amounts of effort.

Contained within the front section of the program in this case will be all of the console UI and various classes which will interact with the backend. The one slight issue with this is that the program will then require the DLL of the backend to function which would make the program more complicated to just move to another computer. To get around this I will be using ILmerge and the script below:

```
C:\\Users\\ruben\\.nuget\\packages\\ilmerge\\3.0.41\\tools\\net452\\ILMerge.exe
    /out:RubensPirieNEA.exe
    ..\\bin\\Release\\LocalApp.exe
    ..\\bin\\Release\\BackendLib.dll
```

This allows the program to be executed as one single .exe file making it easy of the user to use.

2.5.2 Generated Files

When the program is run there will need to be certain directories created in order to store the outputs of the files. The requirement for this will need to be every time the program runs it will

check to see if the directories and files exist, if they do not then it will create them. If they do then it will not create one or override the existing files.

The directory structure will be as follows:

```
.
Program/
    saves/
       <run-id>
    runs/
       <run-id>/
           image(s).png
    logs/
       <run-id>.log
       master.log
    settings.conf
```

The main reason which I am going with this structure is that I feel it makes navigating the program easy and intuitive. Since the logs of what are happening will be stored in the log directory, run files in the runs directory and so on and so forth.

The reason as to why I have chosen to move the settings file out to the root of the program files is that this is going to be the thing that needs to be most quickly accessed by the user and therefore should be easy to get too. I did consider using a database for the settings of the project however have decided against it due to the fact that it would add extra overhead which is not needed.

I did consider using some other form of storing the user outputs from the program, however as one of my extension objectives is to compress all of the outputs into one format which can be drag and dropped I felt that files where the easiest way to go. The custom binary file is also compatible with being windows zipped so I saw no issues with using standard binary and text files in this case.

2.6 Algorithms

2.6.1 Edge Detection

This has been explained in detail in the class breakdown and analysis sections of this report. Here I will explain the general idea behind edge detection as a whole and what method it plays here.

The purpose of edge detection is to allow an application to pick out the edges of a common example of this is in parcel courier services where when a parcel is being shipped they will need to be able to find the exact dimensions of the parcel but cannot afford to have someone there to measure every one. In this case edge detection can be used in order to find the outline and therefore allow it to be processed. The relevance of this to my program is that I need to be able to pick out, on a given image, where the roads are. This I feel is a perfect application of edge detection since I will be picking out roads which are characterised by edges on an image.

There are several different methods of edge detection, the ones which I came across during my research are all examples of searching. Examples of the methods I came across are, Canny (the method I chose), Kovalevsky which takes a rather different approach to Canny where this one uses the second differential of regions in the image to work iut where there are changes in intensity and finally Sobel detection. This is somewhat incorporated into Canny edge detection however the theory behind this is purley mathematical. It involve taking remade matrices and multiplying the image with them.

The advantage to Canny edge detection is that it has two major advantages:

- High Quality Maps: The way in which the image is processed allows it to preserve the resolution as it flows through the various steps. I want to be able to keep the image as high quality as possible since this is presented to a user and not being used in a machine learning environment.

- Accuracy: This method in particular with the first step of the Gaussian filtering, allows the inputs to be noisy, such as a photographed image. This means that I don't have to worry about de-noising the image first.

2.6.2 Refinement Algorithm (Custom)

This is part of the processing the map into a a computer recognisable form. The stage after this will be filling the image. In order to do this the program must have a closed shape in which it can fill. In order to do this I needed to take the output of the edge detection and make sure that all of the roads where enclosed.

I have been unable to find a suitable existing algorithm to complete this for me. There are several different types of filling algorithm which will be explained soon. What I did come across in my research is that there are many different types of image kernel and how they can be used to alter image properties. One useful one which I came across was was the embossing kernel. This goes over the images and essentially "bolds" all of the lines in the image.

With these "bolded lines" it can cause gaps which previously existed to be filled. However there is one final issue which is where the lines have been connected there are intermittent black pixels which haven't been fixed by the embossing kernel. In order to remedy this I will employ a similar method as the hysteresis algorithm in Canny edge detection.

The solution I have come up with is to run over every pixel in the edge map. If there are more than a set number of white "strong" pixels around a black pixel then the black will be replaced with white. What this accomplishes is a "filling" the black pot holes. Overall this does not accomplish much however it means that if an image needs more than one pass of this "fortification" algorithm then this makes the result more reliable in the long run.

I have also decided that this will need to have the option of being run more than once. I found that during the testing if there where large gaps then the embossing would need to occurs more than once.

2.6.3 Flood Fill

This is an algorithm as old as time, I decided to go with th 8 direction flood fill. The premise behind the flood filling algorithm is to start from a seed point in the image and then spread the fill color to all connected pixels in the same region. The algorithm uses a queue to store the pixels that need to be filled, and it works by visiting each pixel in the queue, checking its 8 neighbouring pixels, and adding any unvisited pixels that belong to the same region to the queue. This process continues until all pixels in the region have been visited and filled.

The algorithm can be implemented in a variety of ways, and it is often used as a building block for more complex image processing, as is being used in this case, algorithms, such as object recognition, image segmentation, and feature extraction. It is also commonly used in video games and user interface design to fill shapes and create smooth animations.

In my version the other advantage to using the flood fill colours is that it can be shown to the user to clearly outline where the regions where which would allow them to change the thresholds and behaviour of the filling algorithm. Certain things that can be changed are the method of storing the pixels. Either in a queue of a stack.

2.6.4 Area Rectification (Custom)

This came about due to an issue with the flood filling, the issue with the flood fill is that it fills every single area that there is in the chance that the user will pathfind within a pond marked on the map for example. The solution which I have arrived to to solve this is to take the two extremes of every filled area.

Using these two extremes we can calculate the square boundary of the filled area.

$$(X_{max} - X_{min}) * (Y_{max} - Y_{min}) = \text{Square Area}$$

With this area we can take the actual filled area of the shape and work out the ratio of one to the other by doing

$$\frac{\text{Actual Area}}{\text{Calculated Area}} = \text{Area Ratio}$$

The user will then enter a ratio and if the ratio is greater than the calculated ratio then the area us rejected as not a road. The reason that this works is roads are characteristic's long and winding which means that they will have a small ratio making it easy to single them out. The one downside is that this can produce false positives if there is a thick straight road however in testing this did not seem to be an issue.

2.6.5 Graph Conversion (Custom)

This is not an algorithm as such but more of a method to complete a goal. Once the image has been processed to the fullest degree, this is after the road detection and area rectification has taken place.

The way that this algorithm functions is that it takes the corrected filled image, it will pass over the entire image converting all "non white" pixels to white and keeping black ones black. After it has done this the program will iterate over all of the pixels in the image and if it is white it will take all of the neighbors and add a connection.

As an optimisation step I took the decision to only add nodes to the graph which "exist", this cuts down on the nodes in the image drastically. This is because without making this optimisation the amount of neighbors nodes are always 8 which causes the program complexity to drastically increase. Therefore I am making the decision to remove all of those nodes from the graph in the first place.

The identifier of the nodes will be a Coordinate object which contains the X and Y. These are the references which are stored as the node's children which can then be used to edit the final image to show the user the path.

2.6.6 Dijkstra's Search Algorithm

Dijkstra's search algorithm is a graph search algorithm that is commonly used to find the shortest path between two nodes in a graph. It was first proposed by the Dutch computer scientist Edsger W. Dijkstra in 1956, and it is widely used in computer science, especially in the fields of computer networking, transportation systems, and geographic information systems (similar to my program here).

The algorithm works by maintaining a priority queue of nodes that have been visited and continually updating the distance to the nearest unvisited node until the destination node has been reached. The algorithm operates as follows:

- Start at the source node and set the distance to the source node to zero.
- Visit the neighbors of the current node and update their distances based on the distance to the current node and the edge weight between the current node and the neighbors.
- Choose the node with the smallest distance as the next node to visit, and mark it as visited. Repeat the process until the destination node has been reached or all nodes have been visited.

Dijkstra's algorithm is guaranteed to find the shortest path between two nodes in a graph if there are no negative edge weights. The algorithm is efficient and can be implemented in a variety of programming languages, including C.

In this map processing project, Dijkstra's algorithm can be used to find the shortest path between two points on a map, which can be useful for navigation, route planning, and other applications. The algorithm can be adapted to work with different types of maps, such as road networks or public transportation systems, by representing the map as a graph and defining the edge weights between nodes based on the specific requirements of this application.

As I mentioned really usefully for this application because if I could expand the application to work out the weights of different roads based on width.

2.6.7 A-Star Search Algorithm

A* is a graph search algorithm that is commonly used to find the shortest path between two nodes in a graph. It was first proposed by Peter Hart, Nils Nilsson, and Bertram Raphael in 1968 and is considered an extension of Dijkstra's algorithm.

Like Dijkstra's algorithm, A* operates by maintaining a priority queue of nodes that have been visited, and it continually updates the distance to the nearest unvisited node until the destination node has been reached. The key difference between A* and Dijkstra's is that A* incorporates a heuristic function to estimate the remaining distance from a node to the destination node. This heuristic function allows A* to prioritize nodes that are likely to be closer to the destination node, making it more efficient than Dijkstra's algorithm in many cases.

The algorithm operates as follows:

- Start at the source node and set the distance to the source node to zero.
- Visit the neighbors of the current node and update their distances based on the distance to the current node, the edge weight between the current node and the neighbors, and the estimated remaining distance to the destination node.
- Choose the node with the smallest estimated total distance (the sum of the distance to the current node and the estimated remaining distance to the destination node) as the next node to visit, and mark it as visited. Repeat the process until the destination node has been reached or all nodes have been visited.

A* is often used in computer science and engineering applications that require finding the shortest path between two nodes in a graph, such as computer graphics, video game development, robotics, and artificial intelligence. In this map processing project, A* can be used to find the shortest path between two points on a map, taking into account factors such as road conditions, traffic, and elevation, to provide an optimal route for navigation and route planning which adapts to given conditions.

Then overriding main reason which I am choosing to go with a validation of A* with cartesian distance is that it is significantly faster and more efficient than Dijkstra's. Although Dijkstra has the advantage of post routing meaning that as long as the start node is the same the route can be found instantly.

2.6.8 Binary Heap Min Priority Queue

A binary heap is a type of data structure that can be used to implement a priority queue, where elements are kept in order based on their priority. A binary heap can be either a min-heap or a max-heap, depending on whether the smallest or largest element is considered to have the highest priority. In a min-heap, the parent node of the heap has a value that is less than or equal to the values of its children.

A binary heap is implemented as a complete binary tree, where all levels of the tree are filled except possibly the last one, which is filled from left to right. In a min-heap, the smallest element is stored at the root of the tree, and the two children of a node with a smaller value are stored in the left and right child nodes of that node.

To implement a min-priority queue using a binary heap, we can use the following operations;

Insertion: To insert an element into the binary heap, it is added to the next available position in the last level of the tree, and then "bubbled up" the tree by swapping it with its parent until the min-heap property is restored.

Extraction of minimum element: To extract the minimum element from the binary heap, we remove the root node and replace it with the last element in the tree. The new root node is then "sunk down" the tree by swapping it with its smaller child until the min-heap property is restored.

Decrease key: To decrease the priority of an element in the binary heap, we first find the element and update its value, and then "bubble up" the node by swapping it with its parent until the min-heap property is restored.

Binary heaps are used in many algorithms, such as Dijkstra's shortest path algorithm and the A* algorithm which happen to be the exact algorithms I am using, because they have the property of being able to efficiently maintain a priority queue in a sorted order. They have a time complexity of O(log n) for inserting and extracting elements, which makes them a fast and efficient data structure for implementing priority queues, keeping the delay of processing low.

3 Program Testing

3.1 Testing Tables

3.1.1 Targeted Testing Areas

In order to ensure that my NEA conforms to my objectives this following section will test each of them one at a time. As well as this I will test to make sure that each part of the final solution works together and produces the desired and expected output.
An overview of the sections I will test are:

1. User Map Inputs and Subsequent Outputs
    1.1 Loading In Image Files
    1.2 Creating The Save File
    1.3 Options Given To User
    1.4 Conversion To Graph
    1.5 Error Handling

2. Canny Edge Detection Operations
    2.1 User Variables
    2.2 Constructor Arguments
    2.3 Full Flow Thorough
    2.4 Individual Method Calls
    2.5 Exceptions

3. Road Detection
    3.1 User Variables
    3.2 Constructor Arguments
    3.3 Full Flow Through
    3.4 Individual Method Calls
    3.5 Exceptions

4. Graph Traversal
    4.1 Different Node Placements
    4.2 Different Algorithms
    4.3 Other Graph Settings

5. Logging and Saves
    5.1 Validity Of Save Files
    5.2 Contents of Log Files
    5.3 Save Settings

6. Miscellaneous Items + GUI
    6.1 GUI Elements
    6.2 Matrix Functions
    6.3 Extensions and Utilities

6.4 Structures

It should be noted that in the following tests do not explicitly test objective *5* however it can be seen through out the that this objective has been met. From the icon being clear to the user interface clearing. I believe this combined and the constant evidence shown through the allows me to come to the conclusion that objective *5* has been met.

3.1.2 User Inputs and Outputs Testing Table

Test No.	Name	Input Data / Description	Expected Output	Pass Fail	Test Evidence
1.1.(2) The program should be able to parse a map from a file including...					
1	Entering a JPG	Enter the test image as a JPG into the "New Image" promp.	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	0:10
2	Entering a PNG	Enter the test image as a PNG into the "New Image" promp.	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	0:23
3	Entering a BMP	Enter the test image as a BMP into the "New Image" promp.	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	0:33
4	Entering a TIFF	Enter the test image as a TIFF into the "New Image" promp	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	0:46
1.1.1 A photograph of an map					
5	Entering a Photograph	Enter a photograph into the "new image prompt"	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	0:54
1.1.3 A hand drawing of suitable quality (if it is not a message should be shown)					
6	Entering a Hand Drawing	Enter a hand drawing into the "new image prompt"	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	1:05
1.4 A hand drawing of suitable quality (if it is not a message should be shown)					

7	Entering a Small Image (less than 200x200)	Resize test image to be less than 200x200 and then input that into the "New Image" prompt	The program should reject the image and instruct the user as to how to fix the issue.	Pass	1:17
8	Entering an Invalid Image Path	At the "New Image" prompt an invalid file path should be entered. This test should be repeated with different invalid paths to make sure that all cases are accounted for.	The program should reject all of these inputs without crashing.	Pass	1:35
9	Entering an Local Path	The test described described here would consist of a path in the form "../../image.png" for example.	The program should be able to process this path and show the image to the user in the "Preview Image" form.	Pass	1:52
10	Entering a Valid Save Path	A valid save file path should be entered, use the test image save "save.vmap".	the program should accept this input and show the "Recalled Image" options.	Pass	12:16
11	Entering an Invalid Save Path	An invalid save file path should be entered. This can be any path ending with "/<something>.vmap"	The program should error and instruct the user how to fix the issue.	Pass	13:05
12	Try to Escape Bounds of Option Selector	When in the main menu attempt to go out of bounds of the menu and then select a non existent element.	The option function should not allow the user to go out of the options presented.	Pass	2:11
13	Try to Break inputs through pre-clicking enter.	When going through menus repeatable click the enter key in order to attempt to get the program to error. This can include clicking misc keys as well as enter.	The program should handle all of these inputs before it then waits for non spammed inputs. It should not error.	Pass	2:11
14	Remove Characters from Input	When a text input is required, for example the new image prompt when a path is entered, there is a chance that the user could have entered a mistake. Enter random characters then click "Backspace" to remove characters.	The characters should be removed and no error should occur if the backspace is clicked when the carat is at the end it should not error,	Pass	2:33
1.3 The inputted map should be converted into a graphs					

15	Graph Constructor	Inside the testing menu run the test "Manual Graph", this should generate a predefined graph which contains the nodes and connections as follows.	A: D B: F, C C: B D: A, E, G E: D, H F: B, G G: D, F H: E	Pass	TODO
16	ToGraph Method	On a small test image the function extension .ToGraph should be run.	The outputted graph should contain the following nodes, (0,2), (1,2), (2,0), (2,1), (2,2), (2,3), (2,4), (2,5), (3,2), (4,2), (5,2)	Pass	TODO

3.1.3 Canny Edge Detection Testing Table

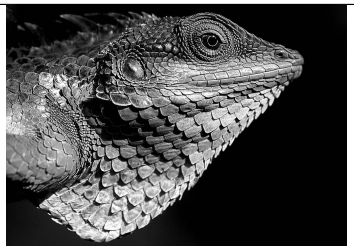
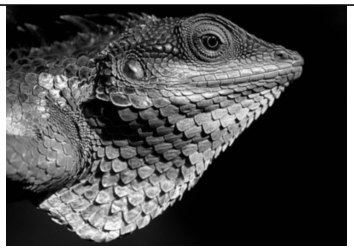
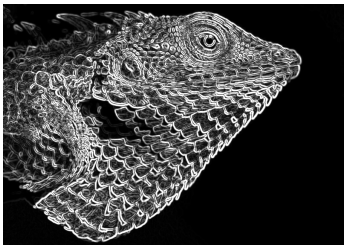
Test No.	Name	Input Data / Description	Expected Output	Pass Fail	Test Evidence
2.1 At each stage of the edge detection an image should be produced					
1	Canny Edge Detect Save Images	Run through a full map detection and at the prompt when it asks if the user would like to save an image at each stage of the Canny edge detection select yes then run the Canny edge detection.	Each stage of the edge detection will have an image saved in the runs/<id> folder.	Pass	2:58
2.3.1 There should be presets to allow quicker processing					
2	Run A Preset	The test image should be input at the "New Image" prompt. When it comes to picking how the edges should be picked the preset "Screenshot" should be selected.	The program should perform Canny Edge Detection without prompting the user for variables. It should return to user control at the "Invert Image" stage.	Pass	3:03
2.3 The edge detection must have the option to be multi threaded.					

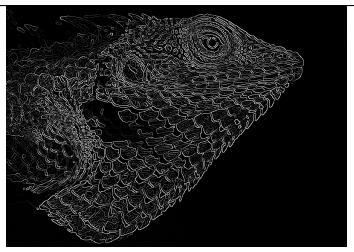
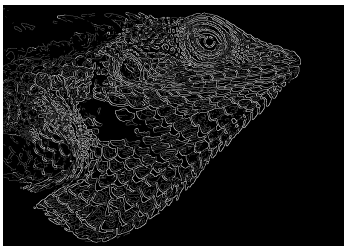
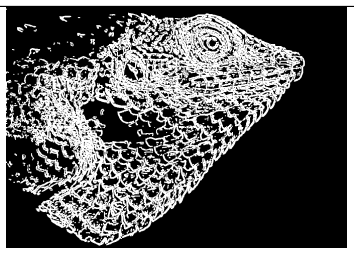
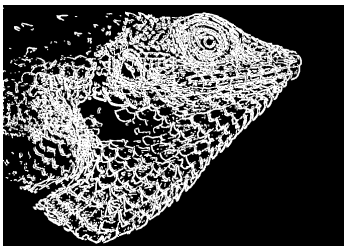
3	Cancel A Run	As above the test image should be entered. Both when it comes to the edge picking "Multi-threaded" then entering values then when the program confirms to continue select "No", and when the image is first read selecting "No" when the "Correct Image" prompt shows.	The program should stop running the current image and error with the reason "You asked for the processing of your map to stop." Then it should return to the main menu.	Pass	3:49
2.2 Between each stage the user should be able to repeat the last step in order to change parameters.					
4	Enter Invalid Values	During the selection of Canny edge detection variations "Multi-threaded" should be chosen. When the program prompts for user inputs a variety of invalid ones should be provided. For example "test", "999999", "1s", "newline", "zero" etc...	The program should check to see if these inputs are within the bounds of the required variables and if they are not it will assume a default value and inform the user.	Pass	3:16
5	Enter Valid Values	Same prompt as above, in the multi-threaded Canny edge detection variables. However this time valid values should be input, these should test the bounds of the inputs as prompted by the program.	The program should accept these changed values and notify the user of what they have changed too.	Pass	3:10

The following tests ending in "method" are run one at a time during the slow, single threaded version of Canny edge detection with the exception of the Gradient calculation with error, these are used to test that each stage of the Canny edge detection algorithm are correct and functioning correctly. A full slow run is included afterwards to show that all of the methods work together. The test image is taken from wikipedia.

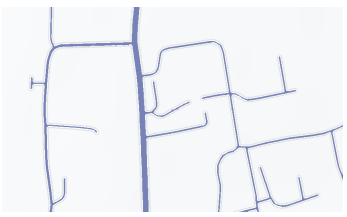
(57) Example Image Used

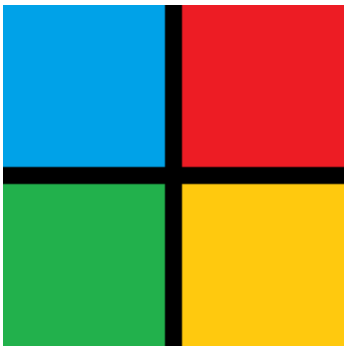
Sourced from Wikipedia©
https://en.wikipedia.org/wiki/Canny_edge_detector#Walkthrough_of_the_algorithm

2.4 The edge detection must have the option to be single threaded					
2.2.1 - 2.2.7 Stages of edge detection.					
6	Black and White Method	Canny Edge Detection method should be ran with the original testing image.		Pass	3:59
7	Gaussian Filter Method	Canny Edge Detection method should be run with the output of the previous step, Black and White conversion.		Pass	4:10
8	Gradient Calculation Method(s)	This test describes a series of method calls which will all combine to form the image to the right. During this test, the outputs of each individual method call should be shown. The input into the initial methods should be the output from the Gaussian filter.		Pass	4:23
9	Gradient Calculation Method	This test describes a series of method calls, the initial calls should be run with the output from the Gaussian filter.	The program should not start the gradient calculations, it should not run any further and should throw an ArgumentException.	Pass	TODO

10	Threshold Method(s)	Canny Edge Detection method should be run with the output of the previous successful step, the non-error gradient calculations.		Pass	4:30
11	Hysteresis Method	Canny Edge Detection method should be run with the output of the previous step, the gradient calculation methods. After this test the image will be in its final edge detected form.		Pass	4:38
12	Run Full Custom Run (Quick)	Using the "RunQuadrant" method in order to quickly process an image. The default values should be used and the result file should be compared to the image to the right.		Pass	11:15
13	Run Full Custom Run (Slow)	The slow single threaded version should be used, this should allow the user to change and go back on variables if they do not like the output. The final expected result is seen to the left. At each stage however the processed images should be shown.		Pass	3:57

3.1.4 Road Detection and Graph Conversion Testing Table

Test No.	Name	Input Data / Description	Expected Output	Pass Fail	Test Evidence
3 The Program must overlay the detected roads onto the original imaged **3.2.4 - 3.2.5** The total filled image can be displayed to the user					
1	Full Run of Road Detection	Using the test image, after the run of Canny edge detection the result should not be inverted and the road threshold should be set to 0.3 and then the road detection run.		Pass	5:06

3.2.3 The percentage threshold for non roads much be changeable by the user					
2	Enter Valid Threshold	When the road prompt is shown a number within the shown range should be entered.	The program should accept this new input and use it in the following process. It should also clearly show the user that the value has been changed.	Pass	4:54
3	Enter Invalid Threshold	When the road prompt is shown a number out the shown range should be entered as well as this invalid strings should be entered. Examples include "test", "ds@13=kle3q" etc...	The program should use the default value and not error. It should clearly show the user that the default value has been used.	Pass	5:05
4	Redo Threshold	After the road detection has been performed the user is prompted whether the result is as they like, at this prompt "No" should be entered.	The program should exit with an error message "You asked for the processing of your map to stop.". It should then return to the main menu.	Pass	5:00
3.2.1 The image should have the option to be inverted					
5	Invert Image Method	An all black image 100x100 image should be fed into this method and then the output should be a 100x100 white square.		Pass	4:52
3.2.2 A filling algorithm should be applied to the image					
6	Fill Image Method	An image with 4 white quadrants should be fed into the function. This image should be 200x200. The colours used are pseudo randomly generated so they may not be identical to the expected output, the 4 quadrants should still be filled however.		Pass	TODO

3.1.5 Graph Traversal Testing Table

Test No.	Name	Input Data / Description	Expected Output	Pass Fail	Test Evidence

The following are all performed on the test image unless otherwise stated, some of the tests are conducted separate to the main program but still using the same methods and functions. This is due to the fact that some of these traversal algorithms are never shown to the user.

4.1.2 The Program should Implement Searching Algorithms these do not have to be shown to the user.					
4.1.2.2 This includes DFS (Depth-first search).					
1	Run DFS	Using the test image run depth first search. Since this test is not shown the user use the premed video.	To the human eye it should look like the path is going "down" more than it is going across, in essence it should look like the image is "filling up".	Pass	5:25
4.1.2.1 This includes BFS (Breadth-first search).					
2	Run BFS Location 1	Using the test image run breadth first search. Since this test is not shown the user use the premed video.	To the human eye it should look like the path is going "across" more than it is going down, in essence it should look like something is spreading from a single point source out to the rest of the image.	Pass	5:47
3	Run BFS Location 2	Using the test image run breadth first search. Since this test is not shown the user use the premed video.	To the human eye it should look like the path is going "across" more than it is going down, in essence it should look like something is spreading from a single point source out to the rest of the image.	Pass	6:10
4.1.1 The Program should implement Routing Algorithms					
4.1.1.1 This includes Dijkstra's algorithm.					
4	Run Dijkstra	Using the save.vmap perform graph traversal using the algorithm "Dijkstra's" setting the start node and end node anywhere on the graph then clicking "Pathfind"	The program should perform Dijkstra's algorithm on the image before drawing the path which it found as the most optimal route.	Pass	6:35
5	Run Dijkstra Same Start Different End	Using the same start node as the previous test the end node should be moved, then "pathfind" should be clicked	The program should instantly draw the new path without having to re-perform Dijkstra's	Pass	6:55

6	Run Dijkstra Different Start Same End	With the same end node as above, the start node should be moved to another point on the image then the "Pathfind" button should be clicked.	The program should perform Dijkstra's again due to the start node being moved.	Pass	7:27
7	Run Dijkstra Different Start Different End	Move both the start and end nodes from the ones above and then click "Pathfind"	As above the program will have to recalculate the entire path since the start node has moved.	Pass	7:04
8	Run Dijkstra End on Find	Enable the setting "endOnFind" and then perform Dijkstra's on two nodes which are relatively spatially close to each other. Then click "Pathfind".	The program will perform Dijkstra's however if it locates the end node it will pause pathfinding there and stop. It should be faster than regular Dijkstra's	Pass	7:54
4.1.1.2 This includes A* (a specialised Dijkstra)					
9	Run A* Image	Two nodes should be placed on points on the graph, then the "Pathfind" button should be clicked.	The algorithm will run the A* algorithm which using a heuristic algorithm will more efficiently find a path to the end node. It should run faster than Dijkstra's.	Pass	8:05

3.1.6 Logging and Saves Testing Table

Test No.	Name	Input Data / Description	Expected Output	Pass Fail	Test Evidence
8 The program should have re-callable settings					
1	Read Normal Settings File	Start the program and navigate to "Settings"	No error should occur and settings should be able to be changed.	Pass	8:27
2	Read Corrupt Settings File	Remove and rename sections of settings file. Then as above.	The program should error and instruct the user how to correct the fault.	Pass	8:47
3	Programmatically Alter Normal Settings File	Navigate to "Settings" and change settings in each sub menu and show altered settings.conf	settings.conf should show the changed settings. Before and after should be shown side by side.	Pass	8:35, 9:14
4	Programmatically Alter Corrupt Settings File	Remove entry from settings then attempt to alter settings similar to above.	The program should error and instruct the user how to correct the fault.	Pass	8:47

5	Save Corrupt Settings File	Attempt to enter the settings menu, alter a setting and the exit. Upon the "exit" condition the file will be saved.	The program should not let the user alter the settings and should error and instruct the user how to proceed.	Pass	8:47
6	Save Normal Settings File	Enter the settings menu, alter a setting and then exit. Upon the "exit" condition the file will be saved.	The file should save without issue and a side by side of the programmatically altered file should be shown.	Pass	8:44
7	Manually Alter Settings File	Open the settings.conf file and change settings values then save and restart the program. Once the program has been restarted check the settings in the menu to see if they have been changed.	The changed settings state should be mirrored in the settings menu.	Pass	9:24
9 / 10 The program settings / save files should be easily movable.					
8	Run Program Fresh	Run the executable of the program.	In the file directory 3 folders should be created. Runs, Saves, Logs. And inside of the log file there should be a file called master.txt Inside the master log a startup message should be recorded. There should also be a config file created.	Pass	10:00
9	Re-run Program	Close the program which was just started. Then run the executable.	No files should be created or deleted however there should be a new entry in the master.log	Pass	10:07
10	Delete Some Folders and Re-run	In the directory where the program file is contained the programmatically created folders should be deleted. Not all but some.	When the program is restarted the files should be recreated	Pass	10:10
11	Full Run and Check Master Log	After the previous tests have been completed (ones involving a raw image being processed) the master.log should be checked	when checking the master log there should be a message saying that a run has started and that it ends. Furthermore it should contain the ID of the run.	Pass	10:19

12	Full Run and Check Individual Log	After the previous tests have been completed (ones involving a raw image being processed) the individual unique run log should be checked	Inside the per run log there should be each step of the edge detection and others depending on pathfinding.	Pass	11:08
13	Cause Error and Check Log	Check the log after one of the input validation tests.	There should be a line in the master file referencing the error.	Pass	TODO
7.1 The map in a binary file format					
7.2 The saved images from the processing of the map should be able to be saved in a compressed format.					
14	Full Run with Save To Zip	Process a whole image asking for it to be saved. The setting "zipOnComplete" enabled. This will ensure that after the processing the file is saved.	After the run has completed in the root directory a zip file will be created containing any partial images, save file and logs.	Pass	11:18
15	Run with Detailed Logging	Enable the setting "detailedLogging" and run through a full process of map recognition.	To the side of the main screen during the process detailed log messages of what exactly is going on should be shown.	Pass	11:18
16	View Save File From Program	Using the test image save attempt to read it into the program.	The program should accept the test image save and take the user to the save image file.	Pass	12:16
7.6 The saved binary file should be able to be have its description changed					
17	Change Save File Information	First the file information should be viewed by selecting "View File Information" then once what you know what you wish to change the "Change File Information". Then any of the details may be changed.	Once a change has been made the program should create a copy of the save file with the new info contained within and the rest of the old data.	Pass	13:21
7.3 The saved binary file should be able to be cloned					
18	Clone Save File	On the save file info page select "Clone"	The program should create a copy of the save file with all of its details exactly the same.	Pass	12:25
7.5 The saved binary file should be able to be renamed					

19	Rename Save File	In the save file menu, "Rename" should be selected. A new name should be entered.	The program will be renamed to the value which the user entered.	Pass	12:36
7.7 The saved binary file should be able to be able to be deleted					
20	Delete Save File	As above select "Delete" and then follow the prompts to delete the file.	Once the user has navigated to the confirm button the program will delete the save file.	Pass	12:51
21	Recall to Pathfind Save File	In the recalled options select the pathfind option.	When this option is selected the program will turn over to the pathfinding image form. From there the user can perform graph traversal on it.	Pass	13:09

3.1.7 Miscellaneous Testing Table

Test No.	Name	Input Data / Description	Expected Output	Pass Fail	Test Evidence
6.1:The program must implement a matrix class					
1	Matrix Constructor	b	c	Pass	TODO
2	Array Index Accessing of Matrix	b	c	Pass	TODO
3	Adding Matrices	b	c	Pass	TODO
4	Subtracting Matrices	b	c	Pass	TODO
5	Matrix Multiplication	b	c	Pass	TODO
6	Scalar Multiplication	b	c	Pass	TODO
7	Matrix Minimisation	b	c	Pass	TODO
8	Matrix Convolution	b	c	Pass	TODO
X.X: No set objective but contribute to the simplistic and user input objectives					
9	Progress Bar Creation	b	c	Pass	TODO

10	Progress Bar Update Action	b	c	Pass	TODO
11	Coord Struct ToString	b	c	Pass	TODO
12	Coord Struct Equals Method	b	c	Pass	TODO
13	Coord Struct Equals Opperator	b	c	Pass	TODO
14	Coord Struct Not Equals Opperator	b	c	Pass	TODO
15	2D Double Array ToBitmap Extension	b	c	Pass	TODO
16	Bitmap ToDoubles Extension	b	c	Pass	TODO
17	2D RGB Structure ToBitmap Extension	b	c	Pass	TODO
18	2D Doubles ToGraph	b	c	Pass	TODO
19	SetPixel Extension	b	c	Pass	TODO
20	GetPixel Extension	b	c	Pass	TODO
21	Gaussian Distribution Utility	b	c	Pass	TODO
22	Bound Utility	b	c	Pass	TODO
23	TryBound Utility	b	c	Pass	TODO
24	Degree to Radian Utility	b	c	Pass	TODO
25	Radian to Degree Utility	b	c	Pass	TODO
26	Map Radian To Pixel Utility	b	c	Pass	TODO

27	Combine Bitmap Utility	b	c	Pass	TODO
28	Split Image Utility	b	c	Pass	TODO
29	combine Quadrants Utility	b	c	Pass	TODO
30	Inverse Image Utility	b	c	Pass	TODO
31	Generic Rebuild Path Utility	b	c	Pass	TODO
32	Is Yes Utility	b	c	Pass	TODO
33	Get Red Utility	b	c	Pass	TODO
34	Get Green Utility	b	c	Pass	TODO
35	Get Blue Utility	b	c	Pass	TODO
36	Get Average Utility	b	c	Pass	TODO
37	Get Industry Average Utility	b	c	Pass	TODO
38	Get If Exists Utility	b	c	Pass	TODO
39	Get Distance Between Nodes Utility	b	c	Pass	TODO

The following tests refer to pathfinding through any given map using A-Star, this is testing the "Pathfind Image Form". The test image recalled from a save file will be used for all of these tests unless otherwise specified.

5 **|** The Program must have a Clear and Simplistic GUI. ***5*** **|** (The following show that it is easy to use and hard to break the user inputs.)				
40	Select No Nodes	Neither left or right mouse buttons should be clicked and then the "Pathfind" button should be clicked.	The program should not run and instantly go back to waiting for input.	Pass	13:53
41	Select One Node	Only one left or right mouse button should be clicked and then the "Pathfind" button should be clicked.	The program should not run and instantly go back to waiting for input.	Pass	13:57

42	Select Two Nodes	Neither left or right mouse buttons should be clicked and then the "Pathfind" button should be clicked.	The program should run and after some time should then wait for input.	Pass	14:00
43	Select One Node Off Path	First the "snapToGrid" setting to false. Set one node off the path and one on and then click the "Pathfind" button.	The program should not run and instantly go back to waiting for input.	Pass	14:08
44	Select Two Nodes Off Path	First the "snapToGrid" setting to false. Set both nodes off the path and then click the "Pathfind" button.	The program should not run and instantly go back to waiting for input.	Pass	14:23
45	Select One Node Off Path One On with Dijkstras	First the "snapToGrid" setting to false. Set one node off the path and one on and then click the "Pathfind" button.	The program should run momentarily and allow then return to waiting. If the end node is then placed back on the road the pathfinding should be instant.	Pass	14:33
46	Click Continue Button in View Image Form	Get to a situation where the "View Image" form is shown. This can be during Canny Edge Detection or when a new image is processed. Then click the continue button.	The button should cause the form to close itself and allow the program to continue.	Pass	TODO

3.2 Testing Video

Please find below several links to the NEA testing Video: as well as a QR code. The timestamps from the table refer to points in this video. Timestamps are also contained within the description.

Raw URL: `https://youtu.be/cJqFovg27Bo`
charlie JULIET quebec FOXTROT oscar victor golf two seven BRAVO oscar

Short URL: `https://shorturl.at/dT158`
delta TANGO one five eight

4 Technical Solution

4.1 Code Table of Contents

5 Evaluation

5.1 Objective Completion

Objective Number	Completion Details
1.1	In its current form my program is able to take an image of a map and convert it successfully into routable data. It accepts most common image formats.
1.2	Should the user select to do so the program will prompt them to enter all of the information about the image that was supplied. Once it collects this information it stores the data in a temporary class in memory and then writes it to a binary save after it has finished processing.
1.3	Once Canny edge detection has been performed upon the image and the set roads have been picked out it, it converts it to a graph inside a windows form to allow the user to pathfind through it.
1.4	When an error occurs the program logs it to a log file allowing the user to go back and see what exactly caused the error. It also shows a large box in the middle of the screen making it clear what just happened. It also contains a small message about how to remedy the error.
2.1	Should the user choose to do so the program can save a image of the current edge detection stage. As well as this if the user selects to run a specific way the program will show a preview of the image while it is running.
2.2	Similar to above should the user chose to do so they can have the program pause at every stage and allow them to input different variables and see the effect that this has.
2.3	If the user choses to do so there are 3 presets they can pick from which all they need to do is select then the program will run through all stages autonomously, alternatively they can choose to input values at the very beginning and run from there.
2.4	This is completed should the user select the single threaded option at the Canny edge detection stage.
3.1	After the Canny edge detection has finished, a windows form is shown to the user, this contains the output of the edge detection after it has gone through the embossing kernel and pixel filling.
3.2	With the result of the Canny edge detection the paths can be detected using a combination of filling and coordinate maths.
4.1	The user can change the map traversal algorithm in the program settings in order to change the behaviour of the algorithms used, two algorithms are .
4.2	Depending on the algorithm which the user has selected to use, when the program reaches the pathfinding windows form, the details about that specific algorithm are displayed. Examples incudes the complexity and amount of nodes processed.

5.1	At the bottom of the user interface there is a small window title telling the user exactly at what stage they are. As well as this the title of the window also updates dynamically depending on which stage the user is at.
5.2	There are only two forms in the final version of my program. The soul purpose of the first form is to display images to the user, it contains nothing but the image itself and a continue button. The second form is used to allows the user to interact with the image and click on points to pathfind from one to another.
5.3	In the settings, should the user chose to select it there is the option for logging to be shown to the user as it is generated or to be hidden.
5.4	Every time that the user moves to the next stage there is a screen clear meaning that no data is left behind to clutter up the GUI
6.1	In order for the Canny edge detection to work the program must be able to perform matrix operations. The matrix class can perform all of the operations outlined in this objective.
6.2	The program implements the graph class and it is heavily used in the pathfinding image form where it has pathfinding algorithms performed on it. It is seen to work due to paths being able to be drawn on the image.
7.x	As outlined in all the sub objectives the program should be able to save the map as a binary file. This is seen where if the user choses so a .vmap file will be produced containing all the information related to the pathfinding and processing of said image. There are also options within the program such as deletion and renaming as required by the extension objectives.
8.x	There is a save file which contains all the information relating to settings inside the program. This file persists over program instances and also can be moved manuals to a new version of the program. As stated in the objectives it contains settings for all the functions listed and more.
9	As stated above the settings file is a normal windows file and therefore can be moved manuals or even the text inside it copied to another settings file.
10	All save files are stored in a .vmap file which is a custom binary file. This means that they can be emailed, put on a USB or any other method of transporting files.

5.2 End User Feedback

1. What do you think about the overall program?

   " The overall program is very impressive with the given examples, when it comes down to the core features I would want from a program like this it comes away with all of them. Some things do feel slightly rough around the edges and if this where to be a tool I would recommend it would need ot be a little more modern, perhaps a website.

   From an ease of use stand point I think that this program excels at making it easy for a user to tell where they are in the whole process. The addition of the small page title at the bottom of the screen makes it intuitive and easy to work out whats going on. The addition of user settings is also nice since it allows me to change various aspects of the program without having to be an expert.

   I really like the way in which when there is a visual thing the program seamlessly floats to the front and lets you see the image. The one thing which is not amazing about it is that by default it forces itself to the front. This makes it very difficult to do other things while it is processing. There is a setting to change this however it is not easily accessible. What might be nice is a minimise button on the window.

   Finally, the way in which I can move the files around from one folder to another is very useful. The nice thing about the program in particular is how I can just click and drag a file from file explorer into the program, it makes it intuitive as to how to use the program. "

2. What do you think of the pathfinding aspect of the program?

   " I think that this has been pulled off perfectly, the way in which the points will jump to the roads on the map makes it easy to just click and not risk it being not on the road if you get what I mean. The way in which the pathfinding works by drawing the line makes it easy to see where it is going.
   Furthermore if the Dijkstra one is used it lets me test lots of different places which is really useful. "

3. Do you believe that my program accomplished my objectives?

   " I believe that all of the objectives that you outlined in your objectives chapter have been completed. I also feel that in some aspects the program you have made has exceeded expectations. However there are some things which I mentioned in the initial interview which you did not manage to incorporate. However on the whole I feel that this program meets and exceeds all of the written objectives. "

4. What could be better about the program what are your Criticisms/Improvements?

   " Firstly the lack of a scale on the routing section is a shame. It is something that I know that I would use and I have a feeling that it would benefit many people. The reason which I feel that it is acceptable in this case is that this is not a fully fledged product and this was not one of the main features that was outlined at the beginning.

   Secondly, when it comes down to the user settings it would be nice if there was a small description to go along with them. While the names are very self explanatory and make logical sense, to someone who just wants to use it as a tool it could be a little of putting. What I suggest is that you make it so that the descriptions pop up in the side part of the screen where it displays the rest of the information.

   Finally, a way to stop the Dijkstra algorithm, it was frustrating when I made a mistake and clicked in the wrong point and had to wait for it to finish before I could click another. It would be nice if there was a STOP button somewhere. "

5.3 Reflection on Feedback

From what the end user has said I feel that most of the improvements are relating to the user interface and not the actual function of the program itself. The most interesting piece of feedback for me is the request of being able to have a selector for the type of algorithm to use when doing the pathfinding. This was not something that I had imagined when making the program, however looking at it form a less technical perspective I can see how this would be useful.

The way in which I could see this being implemented is through the use of a dropdown menu in the pathfinding menu. In there would be the options for which algorithm to use. Once a user has selected one it could give them a quick description to allow them to see how it differs to the first one.

The second piece of key feedback that I saw was adding descriptions of the settings when the user clicks on them. I think this is a great idea and during development had considered it. The main issue was how to show the user what the description of the settings was. I think the way that I would do it is have the settings contained within a form again. Similar to above the advantage to forms is that it requires less skill to navigate, most final users will be coming from phones or computers where the default way of interacting with devices is through clicking on a GUI.

Adding a stop button is defiantly worth adding to the dijkstra's as I feel this is a rather large limitation of my current program. This is when you run the program it will, not crash, but hang and do nothing else while it is pathfinding, this is especially an issue where the algorithm cannot find the end node since it will then run the entire map. The main issue with implementing this is that it would have to send messages across threads which can cause other issues. Overall I think that this is something to come back to.

As for not being able to show a scale on the map, since this was not an objective I still feel that it was not crucial that it was incorporated however it would be a nice feature to have. I would end up with a setting where the user could enter the scale of the map at the beginning and then that data be carried through the data flow of the program.

5.4 System Improvements

I believe that there are several areas to improve upon including some which where not outlined in the end user feedback.

Graph Traversal
When it came down to the ways in which the user could use the map and then traverse it the program did work and achieved the objectives however it is not always what is wanted. I believe that if I could improve the system more I would implement a better system to select the traversal algorithm. Perhaps if the user could select which one they wanted at the time rather than having to do it before hand could relieve some of the irritation when the user goes to pathfind through the map and then cant choose the algorithm they want.

A little more on the visual side of the graph traversal, I feel that if I could improve more I would add an option to change the way the path is drawn on the image. This is from feedback from testing my program is that if you have an image which is rather dark then it can be hard to pick out the line. Whilst the purple colour is pretty good having the option to make it thicker or change its colour I think would be a valid feature.

User Settings
Looking at my end users feedback on the user settings I understand what they are saying with regard to the settings interface. While it serves its purpose and allows the user to change the functionality of the program it is not the most user friendly interface and could be daunting to

someone trying to use it. One way I can think of overcoming this is perhaps turning it into a windows form meaning that it would be more intuitive to use.

More on the adding descriptions to the settings, I feels that this is a great idea and was overlooked in the initial design of the program and how the user interface functions itself. The issue with adding descriptions to settings however is that these descriptions would have to be added somehow into the config file while keeping that user editable. Either that or the settings be hard coded which I feel would be a mistake.

What I would do differently next time
If I where to write this program again I would use a pre made user interface. This is because of the massive hea ache that came half way through making the program which was the update to windows 11. This caused my program to stop working because it replied on the API of windows framework. The way in which I overcame this was manual calculations of the window dimensions. However I feel that I could have avoided all of this if I had just used windows forms or some other similar UI framework. The main reason that I did not in the first place was that I wanted to keep this project as simple as possible in terms of the user interface however in my aim to do this you could argue that it had the opposite effect.

The only other real thing I would change is the class layout. This is not so much of a user interface decision however even with my best efforts this first iteration of the program still has a significant amount of class coupling which could cause issues if I where to make it into a web application.

Overall, there is very little in terms of the core principles of the program I would change, the main downfall is the design of the user interface and even then I believe it was designed well and accomplished its goal.

6 Code Base

6.1 Prototypes

6.1.1 Canny Edge Detection

```
using System;
using System.Drawing;
using System.IO;
using System.Threading.Tasks;

namespace MultithreadedEdgeDetection
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Directory.CreateDirectory("./out");
            var thing = System.Diagnostics.Stopwatch.StartNew();
            Bitmap image = new Bitmap("./image.jpg");
            if (image.Width < 400 || image.Width < 400)
                throw new Exception("Too small must be at least 400 x 400");
            if (image.Width % 2 == 1 || image.Height % 2 == 1)
                throw new Exception("Must be of even dimensions");

            Bitmap[] images = SplitImage(image);

            Task<double[,]>[] tasks = new Task<double[,]>[4];

            for (int i = 0; i < tasks.Length; i++)
            {
                // To overcome the capture condition
                int copyI = i;
                CannyDetection item = new CannyDetection();
                Task<double[,]> task = new Task<double[,]>(() => item.DoDetect(images[copyI], copyI + 1));
                task.Start();
                tasks[i] = task;
            }

            Task.WaitAll(tasks);
            thing.Stop();

            double[,] partA = new double[image.Height / 2, image.Width];
            double[,] partB = new double[image.Height / 2, image.Width];
            for (int i = 0; i < tasks[0].Result.GetLength(0); i++)
            {
                for (int j = 0; j < tasks[0].Result.GetLength(1); j++)
                    partA[i, j] = tasks[0].Result[i, j];

                for (int y = 0; y < tasks[1].Result.GetLength(1); y++)
                    partA[i, y + tasks[0].Result.GetLength(1)] = tasks[1].Result[i, y];
            }

            for (int i = 0; i < tasks[2].Result.GetLength(0); i++)
            {
                for (int j = 0; j < tasks[2].Result.GetLength(1); j++)
                    partB[i, j] = tasks[2].Result[i, j];

                for (int y = 0; y < tasks[3].Result.GetLength(1); y++)
                    partB[i, y + tasks[2].Result.GetLength(1)] = tasks[3].Result[i, y];
```

```
55              }
56
57              double[,] final = new double[image.Height, image.Width];
58              for (int i = 0; i < image.Height; i++)
59              {
60                  if (i < image.Height / 2)
61                  {
62                      for (int j = 0; j < image.Width; j++)
63                      {
64                          final[i, j] = partA[i, j];
65                      }
66                  }
67                  else
68                  {
69                      for (int j = 0; j < image.Width; j++)
70                      {
71                          final[i, j] = partB[i - image.Height / 2, j];
72                      }
73                  }
74              }
75
76              Bitmap finalImage = CannyDetection.DoubleArrayToBitmap(final);
77              finalImage.Save("./final.jpg");
78
79              Console.WriteLine($"Done, took {thing.ElapsedMilliseconds}ms");
80              Console.ReadLine();
81          }
82
83          public static Bitmap[] SplitImage(Bitmap image)
84          {
85              Bitmap one = new Bitmap(image.Width / 2, image.Height / 2);
86              Bitmap two = new Bitmap(image.Width / 2, image.Height / 2);
87              Bitmap three = new Bitmap(image.Width / 2, image.Height / 2);
88              Bitmap four = new Bitmap(image.Width / 2, image.Height / 2);
89
90              for (int i = 0; i < image.Width / 2; i++)
91              {
92                  for (int j = 0; j < image.Height / 2; j++)
93                  {
94                      one.SetPixel(i, j, image.GetPixel(i, j));
95                  }
96              }
97
98              for (int i = image.Width / 2; i < image.Width; i++)
99              {
100                 for (int j = 0; j < image.Height / 2; j++)
101                 {
102                     two.SetPixel(i - (image.Width / 2), j, image.GetPixel(i, j));
103                 }
104             }
105
106             for (int i = 0; i < image.Width / 2; i++)
107             {
108                 for (int j = image.Height / 2; j < image.Height; j++)
109                 {
110                     three.SetPixel(i, j - (image.Height / 2), image.GetPixel(i, j));
111                 }
112             }
113
114             for (int i = image.Width / 2; i < image.Width; i++)
```

```
115             {
116                 for (int j = image.Height / 2; j < image.Height; j++)
117                 {
118                     four.SetPixel(i - (image.Width / 2), j - (image.Height / 2), image.GetPixel(i, j));
119                 }
120             }
121 
122             return new[] { one, two, three, four };
123 
124         }
125     }
126 
127     public class CannyDetection
128     {
129         public double[,] DoDetect(Bitmap masterImage, int id)
130         {
131             Console.WriteLine("Beginning Edge Detection...");
132             Bitmap input = new Bitmap(masterImage);
133             input.Save($"./out/a{id}.jpg");
134 
135             Console.WriteLine($"1. Converting to Black and White ({id})");
136             double[,] bwArray = BWFilter(input);
137             Bitmap bwImage = DoubleArrayToBitmap(bwArray);
138             bwImage.Save($"./out/b{id}.jpg");
139             bwImage.Dispose();
140 
141             Console.WriteLine($"2. Beginning Gaussian Filter ({id})");
142             double[,] gaussianArray = GaussianFilter(1.4, 7, bwArray);
143             Bitmap gaussianImage = DoubleArrayToBitmap(gaussianArray);
144             gaussianImage.Save($"./out/c{id}.jpg");
145             gaussianImage.Dispose();
146 
147             Console.WriteLine($"3. Beginning Gradient Calculations ({id})");
148 
149             Task<double[,]>[] tasks = new Task<double[,]>[2];
150             tasks[0] = new Task<double[,]>(() => CalculateGradientX(gaussianArray));
151             tasks[1] = new Task<double[,]>(() => CalculateGradientY(gaussianArray));
152 
153             foreach (var task in tasks) task.Start();
154             Task.WaitAll(tasks);
155 
156             Bitmap gradientXImage = DoubleArrayToBitmap(tasks[0].Result);
157             Bitmap gradientYImage = DoubleArrayToBitmap(tasks[1].Result);
158             gradientXImage.Save($"./out/d{id}.jpg");
159             gradientYImage.Save($"./out/e{id}.jpg");
160             gradientXImage.Dispose();
161             gradientYImage.Dispose();
162 
163             Console.WriteLine($"4. Beginning Total Gradient Calculations ({id})");
164             double[,] gradientCombined = CalculateGradientCombined(tasks[0].Result, tasks[1].Result);
165             Bitmap gradientCombinedImage = DoubleArrayToBitmap(gradientCombined);
166             gradientCombinedImage.Save($"./out/f{id}.jpg");
167             gradientCombinedImage.Dispose();
168 
169             Console.WriteLine($"5. Calculating Gradient Angles Calculations ({id})");
170             double[,] thetaArray = CalculateTheta(tasks[0].Result, tasks[1].Result);
171             Bitmap thetaImage = ConvertThetaToBitmap(thetaArray);
172             thetaImage.Save($"./out/g{id}.jpg");
173             thetaImage.Dispose();
174 
```

```
175             Console.WriteLine($"6. Beginning Initial Gradient Magnitude Thresholding ({id})");
176             double[,] gradientMagnitudeThreshold = ApplyGradientMagnitudeThreshold(thetaArray, gradientCombined);
177             Bitmap gradientMagnitudeThresholdImage = DoubleArrayToBitmap(gradientMagnitudeThreshold);
178             gradientMagnitudeThresholdImage.Save($"./out/h{id}.jpg");
179             gradientMagnitudeThresholdImage.Dispose();
180
181             Console.WriteLine($"7. Beginning Secondary Min Max Thresholding ({id})");
182             (double, bool)[,] doubleThresholdArray = ApplyDoubleThreshold(0.1, 0.3, gradientMagnitudeThreshold);
183
184             double[,] doubleThresholdImageArray = new double[input.Height, input.Width];
185             for (int i = 0; i < input.Height; i++) for (int j = 0; j < input.Width; j++)
    ↪ doubleThresholdImageArray[i, j] = doubleThresholdArray[i, j].Item1;
186             Bitmap doubleThresholdImage = DoubleArrayToBitmap(doubleThresholdImageArray);
187             doubleThresholdImage.Save($"./out/i{id}.jpg");
188             doubleThresholdImage.Dispose();
189
190             Console.WriteLine($"8. Applying Hystersis ({id})");
191             double[,] edgeTrackingHystersis = ApplyEdgeTrackingHystersis(doubleThresholdArray);
192             Bitmap finalImage = DoubleArrayToBitmap(edgeTrackingHystersis);
193             finalImage.Save($"./out/j{id}.jpg");
194             finalImage.Dispose();
195
196             Console.WriteLine("9. Embossing out image");
197             double[,] embosArray = EmbosImage(edgeTrackingHystersis);
198             Bitmap embosImage = DoubleArrayToBitmap(embosArray);
199             embosImage.Save("./out/k.jpg");
200             embosImage.Dispose();
201
202             Console.WriteLine("10. Filling in the blanks");
203             double[,] filledArray = FillImage(embosArray);
204             Bitmap filledImage = DoubleArrayToBitmap(filledArray);
205             filledImage.Save("./out/l.jpg");
206             filledImage.Dispose();
207
208             Console.WriteLine($"Done {id}");
209
210             return edgeTrackingHystersis;
211         }
212
213         public double[,] FillImage(double[,] imageArray)
214         {
215             double[,] result = imageArray;
216
217             for (int i = 0; i < imageArray.GetLength(0); i++)
218             {
219                 for (int j = 0; j < imageArray.GetLength(1); j++)
220                 {
221                     Matrix imageKernel = BuildKernel(j, i, 3, imageArray);
222                     int count = 0;
223                     foreach (double value in imageKernel.matrix)
224                     {
225                         if (value >= 255) count++;
226                     }
227
228                     if (count > 4) result[i, j] = 255;
229                 }
230             }
231
232             return result;
233         }
```

```
234
235         public double[,] EmbosImage(double[,] imageArray)
236         {
237             double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];
238
239             Matrix embosMatrix = new Matrix(new double[,] {
240                 { -2, -1, 0 },
241                 { -1,  1, 1 },
242                 {  0,  1, 2 },
243             });
244
245             for (int i = 0; i < imageArray.GetLength(0); i++)
246             {
247                 for (int j = 0; j < imageArray.GetLength(1); j++)
248                 {
249                     Matrix imageKernel = BuildKernel(j, i, 3, imageArray);
250                     result[i, j] = Math.Abs(Matrix.Convolution(imageKernel, embosMatrix));
251                 }
252             }
253
254             return result;
255         }
256
257         public static Bitmap ConvertThetaToBitmap(double[,] angles)
258         {
259             Bitmap image = new Bitmap(angles.GetLength(1), angles.GetLength(0));
260
261             for (int i = 0; i < angles.GetLength(0); i++)
262             {
263                 for (int j = 0; j < angles.GetLength(1); j++)
264                 {
265                     int x = (int)(
266                         ((128 / (2 * Math.PI)) * angles[i, j]) + 128
267                     );
268
269                     image.SetPixel(j, i, Color.FromArgb(x, x, x));
270                 }
271             }
272
273             return image;
274
275         }
276
277         public double[,] ApplyEdgeTrackingHystersis((double, bool)[,] arrayOfValues)
278         {
279             double[,] result = new double[arrayOfValues.GetLength(0), arrayOfValues.GetLength(1)];
280
281             for (int i = 0; i < arrayOfValues.GetLength(0); i++)
282             {
283                 for (int j = 0; j < arrayOfValues.GetLength(1); j++)
284                 {
285                     if (arrayOfValues[i, j].Item2 == false)
286                     {
287                         (double, bool)[,] imageKernel = BuildKernel(j, i, 3, arrayOfValues);
288                         bool strong = false;
289                         for (int k = 0; k < 3 && !strong; k++)
290                         {
291                             for (int l = 0; l < 3 && !strong; l++)
292                             {
293                                 if (imageKernel[k, l].Item2 == true) strong = true;
```

```
294                             }
295                         }
296 
297                         result[i, j] = strong ? 255 : 0;
298                     }
299                     else result[i, j] = 255;
300                 }
301             }
302 
303             return result;
304         }
305 
306         public double[,] ApplyGradientMagnitudeThreshold(double[,] angles, double[,] magnitudes)
307         {
308             double[,] result = magnitudes;
309             double[,] anglesInDegrees = ConvertThetaToDegrees(angles);
310 
311             for (int i = 0; i < anglesInDegrees.GetLength(0); i++)
312             {
313                 for (int j = 0; j < anglesInDegrees.GetLength(1); j++)
314                 {
315                     double[,] magnitudeKernel = BuildKernel(j, i, 3, magnitudes).matrix;
316 
317                     if (anglesInDegrees[i, j] < 22.5 || anglesInDegrees[i, j] >= 157.5)
318                     {
319                         if (magnitudes[i, j] < magnitudeKernel[1, 2] || magnitudes[i, j] < magnitudeKernel[1, 0])
320                         {
321                             result[i, j] = 0;
322                         }
323                     }
324                     else if (anglesInDegrees[i, j] >= 22.5 && anglesInDegrees[i, j] < 67.5)
325                     {
326                         if (magnitudes[i, j] < magnitudeKernel[0, 2] || magnitudes[i, j] < magnitudeKernel[2, 0])
327                         {
328                             result[i, j] = 0;
329                         }
330                     }
331                     else if (anglesInDegrees[i, j] >= 67.5 && anglesInDegrees[i, j] < 112.5)
332                     {
333                         if (magnitudes[i, j] < magnitudeKernel[0, 1] || magnitudes[i, j] < magnitudeKernel[2, 1])
334                         {
335                             result[i, j] = 0;
336                         }
337                     }
338                     else if (anglesInDegrees[i, j] >= 112.5 && anglesInDegrees[i, j] < 157.5)
339                     {
340                         if (magnitudes[i, j] < magnitudeKernel[0, 0] || magnitudes[i, j] < magnitudeKernel[2, 2])
341                         {
342                             result[i, j] = 0;
343                         }
344                     }
345                     else throw new Exception();
346                 }
347             }
348 
349             return result;
350         }
351 
352 
353         public (double, bool)[,] ApplyDoubleThreshold(double l, double h, double[,] gradients)
```

```
354         {
355             double min = l * 255;
356             double max = h * 255;
357 
358             (double, bool)[,] result = new (double, bool)[gradients.GetLength(0), gradients.GetLength(1)];
359 
360             for (int i = 0; i < gradients.GetLength(0); i++)
361             {
362                 for (int j = 0; j < gradients.GetLength(1); j++)
363                 {
364                     if (gradients[i, j] < min) result[i, j] = (0, false);
365                     else if (gradients[i, j] > min && gradients[i, j] < max) result[i, j] = (gradients[i, j],
    ↪  false);
366                     else if (gradients[i, j] > max) result[i, j] = (gradients[i, j], true);
367                     else throw new Exception();
368                 }
369             }
370 
371             return result;
372         }
373 
374         public double[,] ConvertThetaToDegrees(double[,] thetaArray)
375         {
376             double[,] result = new double[thetaArray.GetLength(0), thetaArray.GetLength(1)];
377             for (int i = 0; i < thetaArray.GetLength(0); i++) for (int j = 0; j < thetaArray.GetLength(1); j++)
    ↪  result[i, j] = 180 * Math.Abs(thetaArray[i, j]) / Math.PI;
378             return result;
379         }
380 
381         public double[,] CalculateTheta(double[,] gradX, double[,] gradY)
382         {
383             double[,] result = new double[gradX.GetLength(0), gradX.GetLength(1)];
384             for (int i = 0; i < gradX.GetLength(0); i++) for (int j = 0; j < gradX.GetLength(1); j++) result[i, j]
    ↪  = Math.Atan2(gradY[i, j], gradX[i, j]);
385             return result;
386         }
387 
388         public double[,] CalculateGradientCombined(double[,] gradX, double[,] gradY)
389         {
390             double[,] result = new double[gradX.GetLength(0), gradX.GetLength(1)];
391             for (int i = 0; i < gradX.GetLength(0); i++) for (int j = 0; j < gradX.GetLength(1); j++) result[i, j]
    ↪  = Math.Sqrt(Math.Pow(gradX[i, j], 2) + Math.Pow(gradY[i, j], 2));
392             return result;
393         }
394 
395         public double[,] CalculateGradientY(double[,] imageArray)
396         {
397             double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];
398 
399             Matrix sobelY = new Matrix(new double[,] {
400                 { 1, 0, -1 },
401                 { 2, 0, -2 },
402                 { 1, 0, -1 },
403             });
404 
405 
406             for (int i = 0; i < imageArray.GetLength(0); i++)
407             {
408                 for (int j = 0; j < imageArray.GetLength(1); j++)
409                 {
```

```
410                 Matrix imageKernel = BuildKernel(j, i, 3, imageArray);
411                 result[i, j] = Matrix.Convolution(imageKernel, sobelY);
412             }
413         }
414 
415         return result;
416     }
417 
418     public double[,] CalculateGradientX(double[,] imageArray)
419     {
420         double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];
421 
422         Matrix sobelX = new Matrix(new double[,] {
423             {  1,  2,  1 },
424             {  0,  0,  0 },
425             { -1, -2, -1 },
426         });
427         for (int i = 0; i < imageArray.GetLength(0); i++)
428         {
429             for (int j = 0; j < imageArray.GetLength(1); j++)
430             {
431                 Matrix imageKernel = BuildKernel(j, i, 3, imageArray);
432                 result[i, j] = Matrix.Convolution(imageKernel, sobelX);
433             }
434         }
435 
436 
437         return result;
438     }
439 
440     public double[,] GaussianFilter(double sigma, int kernelSize, double[,] imageArray)
441     {
442         double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];
443 
444         Matrix gaussianKernel = GetGaussianKernel(kernelSize, sigma);
445 
446         for (int i = 0; i < result.GetLength(0); i++)
447         {
448             for (int j = 0; j < result.GetLength(1); j++)
449             {
450                 Matrix imageKernel = BuildKernel(j, i, kernelSize, imageArray);
451                 double sum = Matrix.Convolution(imageKernel, gaussianKernel);
452                 result[i, j] = sum;
453             }
454         }
455 
456         return result;
457     }
458 
459     public Matrix GetGaussianKernel(int k, double sigma)
460     {
461         double[,] result = new double[k, k];
462         int halfK = k / 2;
463 
464         double sum = 0;
465 
466         int cntY = -halfK;
467         for (int i = 0; i < k; i++)
468         {
469             int cntX = -halfK;
```

```
470                 for (int j = 0; j < k; j++)
471                 {
472                     result[halfK + cntY, halfK + cntX] = GetGaussianDistribution(cntX, cntY, sigma);
473                     sum += result[halfK + cntY, halfK + cntX];
474                     cntX++;
475                 }
476                 cntY++;
477             }
478 
479             for (int i = 0; i < k; i++) for (int j = 0; j < k; j++) result[i, j] /= sum;
480             return new Matrix(result);
481         }
482 
483 
484         public Matrix BuildKernel(int x, int y, int k, double[,] grid)
485         {
486             double[,] kernel = new double[k, k];
487 
488             int halfK = k / 2;
489 
490             for (int i = 0; i < k; i++) for (int j = 0; j < k; j++) kernel[i, j] = grid[y, x];
491 
492             int cntY = 0;
493             for (int j = y - halfK; j <= y + halfK; j++)
494             {
495                 int cntX = 0;
496                 for (int i = x - halfK; i <= x + halfK; i++)
497                 {
498                     if (j >= 0 && i >= 0 && j < grid.GetLength(0) && i < grid.GetLength(1))
499                     {
500                         kernel[cntY, cntX] = grid[j, i];
501                     }
502                     cntX++;
503                 }
504                 cntY++;
505             }
506 
507             return new Matrix(kernel);
508         }
509 
510         public (double, bool)[,] BuildKernel(int x, int y, int k, (double, bool)[,] grid)
511         {
512             (double, bool)[,] kernel = new (double, bool)[k, k];
513 
514             int halfK = k / 2;
515 
516             for (int i = 0; i < k; i++) for (int j = 0; j < k; j++) kernel[i, j] = grid[y, x];
517 
518             int cntY = 0;
519             for (int j = y - halfK; j <= y + halfK; j++)
520             {
521                 int cntX = 0;
522                 for (int i = x - halfK; i <= x + halfK; i++)
523                 {
524                     if (j >= 0 && i >= 0 && j < grid.GetLength(0) && i < grid.GetLength(1))
525                     {
526                         kernel[cntY, cntX] = grid[j, i];
527                     }
528                     cntX++;
529                 }
```

```
530                 cntY++;
531             }
532 
533             return kernel;
534         }
535 
536         public double[,] BWFilter(Bitmap image)
537         {
538             double[,] result = new double[image.Height, image.Width];
539 
540             for (int i = 0; i < image.Height; i++)
541             {
542                 for (int j = 0; j < image.Width; j++)
543                 {
544                     Color c = image.GetPixel(j, i);
545                     double value = c.R * 0.299 + c.G * 0.587 + c.B * 0.114;
546 
547                     result[i, j] = Bound(0, 255, value);
548                 }
549             }
550 
551             return result;
552         }
553 
554         public static int Bound(int l, int h, double v) => v > h ? h : (v < l ? l : (int)v);
555 
556         public double GetGaussianDistribution(int x, int y, double sigma) =>
557             1 / (2 * Math.PI * sigma * sigma) * Math.Exp(-((Math.Pow(x, 2) + Math.Pow(y, 2)) / (2 * sigma *
    ↪  sigma)));
558 
559 
560         public static Bitmap DoubleArrayToBitmap(double[,] input)
561         {
562             Bitmap image = new Bitmap(input.GetLength(1), input.GetLength(0));
563             for (int i = 0; i < image.Height; i++)
564             {
565                 for (int j = 0; j < image.Width; j++)
566                 {
567                     int val = Bound(0, 255, input[i, j]);
568                     image.SetPixel(j, i, Color.FromArgb(val, val, val));
569                 }
570             }
571             return image;
572         }
573     }
574 
575     public class Matrix
576     {
577         public int x { get; private set; }
578         public int y { get; private set; }
579         public double[,] matrix { get; private set; }
580 
581         public Matrix(double[,] inputMatrix)
582         {
583             x = inputMatrix.GetLength(1);
584             y = inputMatrix.GetLength(0);
585             matrix = inputMatrix;
586         }
587 
588         public static double Convolution(Matrix a, Matrix b)
```

```
589         {
590             if (a.x != a.y || b.x != a.x) throw new Exception();
591 
592             double[,] flippedB = new double[b.y, b.x];
593             int l = b.x;
594             for (int i = l - 1; i >= 0; i--)
595             {
596                 for (int j = l - 1; j >= 0; j--)
597                 {
598                     flippedB[b.y - (i + 1), b.x - (j + 1)] = b.matrix[i, j];
599                 }
600             }
601 
602 
603             double sum = 0;
604             for (int i = 0; i < a.y; i++)
605             {
606                 for (int j = 0; j < a.x; j++)
607                 {
608                     sum += a.matrix[i, j] * flippedB[i, j];
609                 }
610             }
611 
612             return sum;
613         }
614     }
615 }
```

6.1.2 Graph Class and DFS / BFS

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace GraphStuff
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Dictionary<string, List<string>> temp = new Dictionary<string, List<string>>();
            temp.Add("A", new List<string>
            {
                "D"
            });
            temp.Add("B", new List<string>
            {
                "C", "F"
            });
            temp.Add("C", new List<string>
            {
                "B"
            });
            temp.Add("D", new List<string>
            {
                "A", "E", "G"
            });
            temp.Add("E", new List<string>
            {
                "D", "H"
            });
            temp.Add("F", new List<string>
            {
                "B", "G"
            });
            temp.Add("G", new List<string>
            {
                "D", "F"
            });
            temp.Add("H", new List<string>
            {
                "E"
            });

            Graph myGraph = new Graph(temp);
            Console.WriteLine(string.Join(", ", DFS("A", myGraph)));
            Console.WriteLine(string.Join(", ", BFS("A", myGraph)));
            Console.ReadLine();
        }

        public static string[] DFS(string start, Graph graph)
        {
            List<string> path = new List<string>();
            Stack<string> stack = new Stack<string>();
            Dictionary<string, bool> visited = new Dictionary<string, bool>();
            foreach (string s in graph.GetAllNodes()) visited.Add(s, false);

            // Kick Start
```

```
59              stack.Push(start);
60
61              while (!stack.IsEmpty())
62              {
63
64                  string node = stack.Pop();
65                  path.Add(node);
66                  visited[node] = true;
67
68                  List<string> connections = graph.GetNode(node);
69
70                  connections.Reverse();
71
72                  foreach (string s in connections)
73                  {
74                      if (visited[s] == false)
75                      {
76                          stack.Push(s);
77                      }
78                  }
79              }
80
81
82              return path.ToArray();
83          }
84
85          public static string[] BFS(string start, Graph graph)
86          {
87              List<string> path = new List<string>();
88              Queue<string> stack = new Queue<string>();
89              Dictionary<string, bool> visited = new Dictionary<string, bool>();
90              foreach (string s in graph.GetAllNodes()) visited.Add(s, false);
91
92              // Kick Start
93              stack.Enqueue(start);
94
95              while (!stack.IsEmpty())
96              {
97
98                  string node = stack.Dequeue();
99                  path.Add(node);
100                 visited[node] = true;
101
102                 List<string> connections = graph.GetNode(node);
103
104                 connections.Reverse();
105
106                 foreach (string s in connections)
107                 {
108                     if (visited[s] == false)
109                     {
110                         stack.Enqueue(s);
111                     }
112                 }
113             }
114
115             return path.ToArray();
116         }
117     }
118
```

```
119     public class Queue<T>
120     {
121         public List<T> _data = new List<T>();
122 
123         public T Dequeue()
124         {
125             T val = _data[0];
126             _data.RemoveAt(0);
127             return val;
128         }
129 
130         public void Enqueue(T val) => _data.Add(val);
131 
132         public bool IsEmpty() => _data.Count == 0;
133     }
134 
135     public class Stack<T>
136     {
137         public List<T> _data = new List<T>();
138 
139         public T Pop()
140         {
141             T val = _data[_data.Count - 1];
142             _data.RemoveAt(_data.Count - 1);
143             return val;
144         }
145 
146         public void Push(T val) => _data.Add(val);
147 
148         public bool IsEmpty() => _data.Count == 0;
149 
150     }
151 
152 
153     public class Graph
154     {
155         public Dictionary<string, List<string>> _data = new Dictionary<string, List<string>>();
156 
157         public Graph(Dictionary<string, List<string>> graph)
158         {
159             _data = graph;
160         }
161 
162         public void AddNode(string name)
163         {
164             if (_data.ContainsKey(name)) throw new GraphException($"Cannot add {name}, node already exists.");
165             _data.Add(name, new List<string>());
166         }
167 
168         public void RemoveNode(string name)
169         {
170             if (!_data.ContainsKey(name)) throw new GraphException($"Cannot remove {name}, node does not exist.");
171             _data.Remove(name);
172         }
173 
174         public void AddConnection(string node, string name)
175         {
176             if (!_data.ContainsKey(node)) throw new GraphException($"Cannot add connection {name} to {node}
↪  original node does not exist.");
```

```
177             if (_data[node].Contains(name)) throw new GraphException($"Cannot add connection {name} to {node}
    ↪  connection already exists.");
178             _data[node].Add(name);
179         }
180 
181         public List<string> GetNode(string node)
182         {
183             if (!_data.ContainsKey(node)) throw new GraphException($"Node {node} does not exist.");
184             return _data[node];
185         }
186 
187         public string[] GetAllNodes() => _data.Keys.ToArray();
188 
189         public void Clear() => _data.Clear();
190     }
191 
192     public class GraphException : Exception
193     {
194         public GraphException(string message) : base(message)
195         {
196         }
197     }
198 }
```

6.1.3 Forms Interface

6.2 Final Solution

6.2.1 BackendLib

6.2.1.1 Data

MapFile.cs

```
public class MapFile
{
    public readonly string _filePath;
    private const string FileExtensionRegex =
↪   @"^([a-z]:\\|\\[a-z]|\.\.(\\|\/)|\.(\\|\/))(([a-z]|(\\|\/))+)\.(vmap)$";

    public string Name { get; set; }
    public string Description { get; set; }
    public int Type { get; set; }
    public bool IsInverted { get; set; }
    public DateTimeOffset TimeCreated { get; set; }
    public Bitmap PathImage { get; set; }
    public Bitmap OriginalImage { get; set; }
    public Bitmap CombinedImage { get; set; }

    public MapFile()
    {
        TimeCreated = DateTimeOffset.Now;
    }

    public MapFile(string filePath)
    {
        _filePath = filePath;
    }

    public void Initialize(Action updateProgress)
    {
        ValidateImage();
        updateProgress();
        ReadMapFile(updateProgress);
    }

    private void ValidateImage()
    {
        Regex fileRegex = new Regex(FileExtensionRegex, RegexOptions.IgnoreCase);

        if (!File.Exists(_filePath)) throw new MapFileException("The virtual map that you entered does not exist,
↪   double check the path to the file and that exists.");
        if (!fileRegex.IsMatch(_filePath)) throw new MapFileException("The file which you entered does not appear
↪   to be a map file. It should end in .vmap double check and try again.");
    }

    private void ReadMapFile(Action updateProgress)
    {
        using (BinaryReader br = new BinaryReader(File.Open(_filePath, FileMode.Open)))
        {
            string dateTime = br.ReadString();
            DateTime dt = new DateTime(1970, 1, 1, 0, 0, 0, 0,
↪   DateTimeKind.Utc).AddMilliseconds(double.Parse(dateTime)).ToLocalTime();
            TimeCreated = new DateTimeOffset(dt);

```

```
48              Name = br.ReadString();
49              Description = br.ReadString();
50              Type = br.ReadInt32();
51              IsInverted = br.ReadBoolean();
52
53              int width = (int)br.ReadInt32();
54              int height = (int)br.ReadInt32();
55
56              for (int j = 0; j < 3; j++)
57              {
58                  Structures.RGB[,] tempImage = new Structures.RGB[height, width];
59                  for (int i = 0; i < 3; i++)
60                  {
61                      for (int y = 0; y < height; y++)
62                      {
63                          for (int x = 0; x < width; x++)
64                          {
65                              if (i == 0) tempImage[y, x].R = br.ReadByte();
66                              else if (i == 1) tempImage[y, x].G = br.ReadByte();
67                              else if (i == 2) tempImage[y, x].B = br.ReadByte();
68                          }
69                      }
70                      updateProgress();
71                  }
72
73                  if (j == 0) OriginalImage = tempImage.ToBitmap();
74                  else if (j == 1) PathImage = tempImage.ToBitmap();
75                  else if (j == 2) CombinedImage = tempImage.ToBitmap();
76              }
77          }
78      }
79
80      public string Save(Guid currentGuid)
81      {
82          using (BinaryWriter bw = new BinaryWriter(File.Open($"./saves/{currentGuid}.vmap", FileMode.OpenOrCreate)))
83          {
84              bw.Write(TimeCreated.ToUnixTimeMilliseconds().ToString());
85
86              bw.Write(Name);
87              bw.Write(Description);
88              bw.Write(Type);
89              bw.Write(IsInverted);
90
91              bw.Write((int)OriginalImage.Width);
92              bw.Write((int)OriginalImage.Height);
93
94              for (int j = 0; j < 3; j++)
95              {
96                  for (int i = 0; i < 3; i++)
97                  {
98                      for (int y = 0; y < OriginalImage.Height; y++)
99                      {
100                         for (int x = 0; x < OriginalImage.Width; x++)
101                         {
102                             if (j == 0)
103                             {
104                                 if (i == 0) bw.Write(OriginalImage.GetPixel(x, y).R);
105                                 else if (i == 1) bw.Write(OriginalImage.GetPixel(x, y).G);
106                                 else if (i == 2) bw.Write(OriginalImage.GetPixel(x, y).B);
107                             }
```

```
                            else if (j == 1)
                            {
                                if (i == 0) bw.Write(PathImage.GetPixel(x, y).R);
                                else if (i == 1) bw.Write(PathImage.GetPixel(x, y).G);
                                else if (i == 2) bw.Write(PathImage.GetPixel(x, y).B);
                            }
                            else if (j == 2)
                            {
                                if (i == 0) bw.Write(CombinedImage.GetPixel(x, y).R);
                                else if (i == 1) bw.Write(CombinedImage.GetPixel(x, y).G);
                                else if (i == 2) bw.Write(CombinedImage.GetPixel(x, y).B);
                            }
                        }
                    }
                }
            }
        }

        return $"./saves/{currentGuid}.vmap";
    }
}

```

Traversal.cs

```
public class Traversal<T>
{
    private Graph<T> _graph;

    public Traversal(Graph<T> graph)
    {
        _graph = graph;
    }

    public T[] DFS(T start)
    {
        List<T> path = new List<T>();
        Datatypes.Stack<T> stack = new Datatypes.Stack<T>();
        Dictionary<T, bool> visited = new Dictionary<T, bool>();
        foreach (T s in _graph.GetAllNodes()) visited.Add(s, false);

        // Kick Start
        stack.Push(start);

        while (!stack.IsEmpty())
        {
            T node = stack.Pop();
            path.Add(node);
            visited[node] = true;

            List<T> connections = _graph.GetNode(node);

            connections.Reverse();

            foreach (T s in connections)
            {
                if (visited[s] == false && !stack.Contains(s))
                {
                    stack.Push(s);
                }
            }
        }


        return path.ToArray();
    }

    public T[] BFS(T start)
    {
        List<T> path = new List<T>();
        Datatypes.Queue<T> queue = new Datatypes.Queue<T>();
        Dictionary<T, bool> visited = new Dictionary<T, bool>();
        foreach (T s in _graph.GetAllNodes()) visited.Add(s, false);

        // Kick Start
        queue.Enqueue(start);

        while (!queue.IsEmpty())
        {

            T node = queue.Dequeue();
            path.Add(node);
            visited[node] = true;

```

```
60              List<T> connections = _graph.GetNode(node);
61
62              connections.Reverse();
63
64              foreach (T s in connections)
65              {
66                  if (visited[s] == false && !queue.Contains(s))
67                  {
68                      queue.Enqueue(s);
69                  }
70              }
71          }
72
73          return path.ToArray();
74      }
75
76      public Dictionary<T, T> AStar(T start, T goal, Func<T, T, int> weightFunction)
77      {
78          Dictionary<T, double> dist = new Dictionary<T, double>();
79          Dictionary<T, T> prev = new Dictionary<T, T>();
80
81          MinPriorityQueue<T> queue = new MinPriorityQueue<T>();
82
83          queue.Enqueue(start, weightFunction(start, goal));
84          dist.Add(start, 0);
85
86          foreach (T node in _graph.GetAllNodes())
87          {
88              if (!Equals(node, start))
89              {
90                  dist.Add(node, double.MaxValue);
91                  queue.Enqueue(node, double.MaxValue);
92              }
93          }
94
95
96          while (queue.Size > 0)
97          {
98              T current = queue.Dequeue();
99              if (Equals(current, goal)) return prev;
100
101             foreach (T neighbor in _graph.GetNode(current))
102             {
103                 double tentative = dist[current] + 1;
104                 if (tentative < dist[neighbor])
105                 {
106                     dist[neighbor] = tentative;
107                     if (prev.ContainsKey(neighbor)) prev[neighbor] = current;
108                     else prev.Add(neighbor, current);
109                     queue.ChangePriority(neighbor, tentative + weightFunction(neighbor, goal));
110                 }
111             }
112         }
113
114
115         return new Dictionary<T, T>();
116     }
117
118     public Dictionary<T, T> Dijkstra(T start, T goal, bool endOnFind, Action nodeUpdate)
119     {
```

```
120         Dictionary<T, double> dist = new Dictionary<T, double>();
121         Dictionary<T, T> prev = new Dictionary<T, T>();
122         dist.Add(start, 0);
123 
124         MinPriorityQueue<T> queue = new MinPriorityQueue<T>();
125 
126         T[] nodes = _graph.GetAllNodes();
127         foreach (T node in nodes)
128         {
129             if (_graph.GetNode(node).Count > 0)
130             {
131                 if (!Equals(start, node)) dist.Add(node, double.MaxValue);
132                 queue.Enqueue(node, dist[node]);
133             }
134         }
135 
136         while (queue.Size > 0)
137         {
138             T minVertex = queue.Dequeue();
139             nodeUpdate();
140             if (Equals(minVertex, goal) && endOnFind) return prev;
141 
142             List<T> adjacent = _graph.GetNode(minVertex);
143 
144             foreach (var neighbor in adjacent)
145             {
146 
147                 if (queue.Contains(neighbor))
148                 {
149                     double alternateWeight = dist[minVertex] + 1;
150                     if (alternateWeight < dist[neighbor])
151                     {
152                         dist[neighbor] = alternateWeight;
153                         if (prev.ContainsKey(neighbor)) prev[neighbor] = minVertex;
154                         else prev.Add(neighbor, minVertex);
155                         queue.ChangePriority(neighbor, alternateWeight);
156                     }
157                 }
158             }
159         }
160 
161         return prev;
162     }
163 }
```

6.2.1.2 Datatypes

Graph.cs

```
public class Graph<T>
{
    public Dictionary<T, List<T>> _data = new Dictionary<T, List<T>>();

    public Graph() { }

    public Graph(Dictionary<T, List<T>> graph)
    {
        _data = graph;
    }

    public void AddNode(T key)
    {
        if (_data.ContainsKey(key)) throw new GraphException($"Failed to add node {key} to the graph, the node already exists.");
        _data.Add(key, new List<T>());
    }

    public void RemoveNode(T key)
    {
        if (!_data.ContainsKey(key)) throw new GraphException($"Failed to remove node {key} from the graph, the node does not exist.");
        _data.Remove(key);
    }

    public void AddConnection(T key, T value)
    {
        if (!_data.ContainsKey(key)) throw new GraphException($"Cannot add connection between {value} and {key} the parent node does not exist in the graph.");
        if (_data[key].Contains(value)) throw new GraphException($"Cannot add connection between {value} and {key} the connection already exists.");
        _data[key].Add(value);
    }

    public List<T> GetNode(T key)
    {
        if (!_data.ContainsKey(key)) throw new GraphException($"Failed to get node {key} form graph because it does not exist.");
        return _data[key];
    }

    public T[] GetAllNodes() => _data.Keys.ToArray();

    public bool ContainsNode(T node) => _data.ContainsKey(node);

    public void Clear() => _data.Clear();
}
```

Matrix.cs

```
public class Matrix : IEnumerable
{
    private readonly double[,] _matrix;
    public int X { get; }
    public int Y { get; }

    public Matrix(double[,] matrix)
    {
        _matrix = matrix;
        X = matrix.GetLength(1);
        Y = matrix.GetLength(0);
    }

    public Matrix(int x, int y)
    {
        _matrix = new double[y, x];
        X = x;
        Y = y;
    }


    public double this[int y, int x]
    {
        get => _matrix[y, x];
        private set => _matrix[y, x] = value;
    }

    public static Matrix operator +(Matrix a, Matrix b)
    {
        if (a.X != b.X || a.Y != b.Y) throw new MatrixException("Matrices must be the same dimensions to add.");

        Matrix m = new Matrix(a.X, a.Y);
        for (int i = 0; i < a.Y; i++) for (int j = 0; j < a.X; j++) m[i, j] = a[i, j] + b[i, j];
        return m;
    }

    public static Matrix operator -(Matrix a, Matrix b)
    {
        if (a.X != b.X || a.Y != b.Y) throw new MatrixException("Matrices must be the same dimensions to
↪   subtract.");

        Matrix m = new Matrix(a.X, a.Y);
        for (int i = 0; i < a.Y; i++) for (int j = 0; j < a.X; j++) m[i, j] = a[i, j] - b[i, j];
        return m;
    }
    public static Matrix operator *(Matrix a, Matrix b)
    {
        if (a.X != b.X || a.Y != b.Y) throw new MatrixException("Matrices must be the same dimensions to
↪   multiply.");

        Matrix m = new Matrix(a.X, a.Y);
        for (int i = 0; i < a.Y; i++) for (int j = 0; j < a.X; j++) m[i, j] = a[i, j] * b[i, j];
        return m;
    }

    public static Matrix operator *(int a, Matrix b)
    {
        Matrix m = new Matrix(b.X, b.Y);
        for (int i = 0; i < b.Y; i++) for (int j = 0; j < b.X; j++) m[i, j] = a * b[i, j];
```

```
58          return m;
59      }
60
61      public void Minimize()
62      {
63          double sum = 0;
64          foreach (double val in _matrix) sum += val;
65
66          for (int i = 0; i < Y; i++)
67          {
68              for (int j = 0; j < X; j++)
69              {
70                  _matrix[i, j] /= sum;
71              }
72          }
73      }
74
75      public static double Convolution(Matrix a, Matrix b)
76      {
77          if (a.X != b.X || b.Y != a.Y) throw new MatrixException("Matrices must be the same dimensions to apply
    ↪  convolution.");
78
79          double[,] flippedB = new double[b.Y, b.X];
80          int l = b.X;
81          for (int i = l - 1; i >= 0; i--) for (int j = l - 1; j >= 0; j--) flippedB[b.Y - (i + 1), b.X - (j + 1)] =
    ↪  b[i, j];
82
83          double sum = 0;
84          for (int i = 0; i < a.Y; i++) for (int j = 0; j < a.X; j++) sum += a[i, j] * flippedB[i, j];
85          return sum;
86      }
87
88      public IEnumerator GetEnumerator() => _matrix.GetEnumerator();
89
90  }
```

MaxPriorityQueue.cs

```
public class MaxPriorityQueue<T>
{
    private List<int> _priorityQueue = new List<int>();
    private List<T> _queue = new List<T>();

    public int Size => _priorityQueue.Count;
    private int _size => _priorityQueue.Count - 1;

    public MaxPriorityQueue() { }

    private T GetParent(int index) => _queue[Parent(index)];
    private int Parent(int index) => (index - 1) / 2;

    private T GetLeftChild(int index) => _queue[LeftChild(index)];
    private int LeftChild(int index) => (index * 2) + 1;

    private T GetRightChild(int index) => _queue[RightChild(index)];
    private int RightChild(int index) => (index * 2) + 2;

    private void ShiftNodeUp(int index)
    {
        while (index > 0 && _priorityQueue[Parent(index)] < _priorityQueue[index])
        {
            Swap(Parent(index), index);
            index = Parent(index);
        }
    }

    public void ChangePriority(T item, int newPriority)
    {
        int index = _queue.FindIndex(i => Equals(i, item));
        int oldPriority = _priorityQueue[index];
        _priorityQueue[index] = newPriority;

        if (newPriority > oldPriority) ShiftNodeUp(index);
        else ShiftNodeDown(index);
    }

    private void ShiftNodeDown(int index)
    {
        int maxIndex = index;

        int left = LeftChild(index);
        if (left <= _size && _priorityQueue[left] > _priorityQueue[maxIndex]) maxIndex = left;

        int right = RightChild(index);
        if (right <= _size && _priorityQueue[right] > _priorityQueue[maxIndex]) maxIndex = right;

        if (index != maxIndex)
        {
            Swap(index, maxIndex);
            ShiftNodeDown(maxIndex);
        }
    }

    public void Enqueue(T item, int priority)
    {
        _queue.Add(item);
        _priorityQueue.Add(priority);
```

```
60
61          ShiftNodeUp(_size);
62      }
63
64      public T Dequeue() => RemoveMax().Item1;
65
66      private (T, int) RemoveMax()
67      {
68          int res = _priorityQueue[0];
69          T result = _queue[0];
70          _priorityQueue.RemoveAt(0);
71          _queue.RemoveAt(0);
72
73          ShiftNodeDown(0);
74
75          return (result, res);
76      }
77
78      private void Swap(int indexX, int indexY)
79      {
80          T tempValue = _queue[indexX];
81          _queue[indexX] = _queue[indexY];
82          _queue[indexY] = tempValue;
83
84          int tempPriority = _priorityQueue[indexX];
85          _priorityQueue[indexX] = _priorityQueue[indexY];
86          _priorityQueue[indexY] = tempPriority;
87      }
88
89      public bool Contains(T neighbor) => _queue.Contains(neighbor);
90  }
```

MinPriorityQueue.cs

```
public class MinPriorityQueue<T>
{
    private List<double> _priorityQueue = new List<double>();
    private List<T> _queue = new List<T>();

    public int Size => _priorityQueue.Count;
    private int _size => _priorityQueue.Count - 1;

    public MinPriorityQueue() { }

    private int Parent(int index) => (index - 1) / 2;
    private int Left(int index) => (2 * index) + 1;
    private int Right(int index) => (2 * index) + 2;

    public void Enqueue(T value, double priority)
    {
        int oldSize = Size;

        _queue.Add(value);
        _priorityQueue.Add(priority);

        while (oldSize != 0 && _priorityQueue[oldSize] < _priorityQueue[Parent(oldSize)])
        {
            Swap(oldSize, Parent(oldSize));
            oldSize = Parent(oldSize);
        }
    }

    public void ChangePriority(T item, double newPriority)
    {
        int index = _queue.FindIndex(i => Equals(i, item));
        if (index > -1)
        {
            if (_priorityQueue[index] > newPriority)
            {
                _priorityQueue[index] = newPriority;

                while (index != 0 && _priorityQueue[index] < _priorityQueue[Parent(index)])
                {
                    Swap(index, Parent(index));
                    index = Parent(index);
                }
            }
            else
            {
                _priorityQueue[index] = newPriority;
                MinifyHeap(index);
            }
        }

    }

    public T Dequeue()
    {
        if (Size == 1)
        {
            T val = _queue[0];

            _queue.RemoveAt(0);
```

```
60              _priorityQueue.RemoveAt(0);
61
62              return val;
63          }
64
65          T res = _queue[0];
66
67          int oldSize = _size;
68
69          _queue[0] = _queue[oldSize];
70          _queue.RemoveAt(oldSize);
71          _priorityQueue[0] = _priorityQueue[oldSize];
72          _priorityQueue.RemoveAt(oldSize);
73
74          MinifyHeap(0);
75
76          return res;
77      }
78
79      private void MinifyHeap(int index)
80      {
81          int left = Left(index);
82          int right = Right(index);
83
84          int smallest = index;
85
86          if (left < Size && _priorityQueue[left] < _priorityQueue[smallest]) smallest = left;
87          if (right < Size && _priorityQueue[right] < _priorityQueue[smallest]) smallest = right;
88          if (smallest != index)
89          {
90              Swap(index, smallest);
91              MinifyHeap(smallest);
92          }
93      }
94
95      private void Swap(int indexX, int indexY)
96      {
97          T tempValue = _queue[indexX];
98          _queue[indexX] = _queue[indexY];
99          _queue[indexY] = tempValue;
100
101         double tempPriority = _priorityQueue[indexX];
102         _priorityQueue[indexX] = _priorityQueue[indexY];
103         _priorityQueue[indexY] = tempPriority;
104     }
105
106     public bool Contains(T neighbor) => _queue.Contains(neighbor);
107 }
```

Queue.cs

```
public class Queue<T>
{
    private List<T> _queue = new List<T>();
    public int Size => _queue.Count;

    public Queue() { }

    public Queue(IEnumerable<T> input)
    {
        foreach (var item in input) _queue.Add(item);
    }

    public void Enqueue(T item) => _queue.Add(item);

    public T Dequeue()
    {
        T item = _queue[0];
        _queue.RemoveAt(0);
        return item;
    }

    public bool IsEmpty() => _queue.Count == 0;

    public bool Contains(T item) => _queue.Contains(item);
}
```

Stack.cs

```
public class Stack<T>
{
    private List<T> _stack = new List<T>();
    public int Size => _stack.Count;

    public Stack() { }

    public Stack(IEnumerable<T> input)
    {
        foreach (var item in input) _stack.Add(item);
    }
    public T Peek() => _stack[_stack.Count - 1];

    public void Push(T item) => _stack.Add(item);

    public T Pop()
    {
        T item = _stack[_stack.Count - 1];
        _stack.RemoveAt(_stack.Count - 1);
        return item;
    }

    public bool IsEmpty() => _stack.Count == 0;

    public bool Contains(T item) => _stack.Contains(item);
}
```

6.2.1.3 Exceptions

GraphException.cs

```
1  [Serializable]
2  public class GraphException : Exception
3  {
4      public GraphException()
5      {
6      }
7
8      public GraphException(string message) : base(message)
9      {
10     }
11
12     public GraphException(string message, Exception innerException) : base(message, innerException)
13     {
14     }
15
16     protected GraphException(SerializationInfo info, StreamingContext context) : base(info, context)
17     {
18     }
19 }
```

KernelException.cs

```
[Serializable]
public class KernelException : Exception
{
    public KernelException()
    {
    }

    public KernelException(string message) : base(message)
    {
    }

    public KernelException(string message, Exception innerException) : base(message, innerException)
    {
    }

    protected KernelException(SerializationInfo info, StreamingContext context) : base(info, context)
    {
    }
}
```

LoggerException.cs

```
[Serializable]
public class LoggerException : Exception
{
    public LoggerException()
    {
    }

    public LoggerException(string message) : base(message)
    {
    }

    public LoggerException(string message, Exception innerException) : base(message, innerException)
    {
    }

    protected LoggerException(SerializationInfo info, StreamingContext context) : base(info, context)
    {
    }
}
```

MapFileException.cs

```
[Serializable]
public class MapFileException : Exception
{
    public MapFileException()
    {
    }

    public MapFileException(string message) : base(message)
    {
    }

    public MapFileException(string message, Exception innerException) : base(message, innerException)
    {
    }

    protected MapFileException(SerializationInfo info, StreamingContext context) : base(info, context)
    {
    }
}
```

MatrixException.cs

```
[Serializable]
public class MatrixException : Exception
{
    public MatrixException()
    {
    }

    public MatrixException(string message) : base(message)
    {
    }

    public MatrixException(string message, Exception innerException) : base(message, innerException)
    {
    }

    protected MatrixException(SerializationInfo info, StreamingContext context) : base(info, context)
    {
    }
}
```

PreprocessingException.cs

```
[Serializable]
public class PreprocessingException : Exception
{
    public PreprocessingException()
    {
    }

    public PreprocessingException(string message) : base(message)
    {
    }

    public PreprocessingException(string message, Exception innerException) : base(message, innerException)
    {
    }

    protected PreprocessingException(SerializationInfo info, StreamingContext context) : base(info, context)
    {
    }
}
```

SettingsException.cs

```
[Serializable]
public class SettingsException : Exception
{
    public SettingsException()
    {
    }

    public SettingsException(string message) : base(message)
    {
    }

    public SettingsException(string message, Exception innerException) : base(message, innerException)
    {
    }

    protected SettingsException(SerializationInfo info, StreamingContext context) : base(info, context)
    {
    }
}
```

6.2.1.4 Interfaces

IHandler.cs

```
1  public interface IHandler
2  {
3      void Start();
4      double[,] Result();
5  }
```

6.2.1.5 Processing

CannyEdgeDetection.cs

```
public class CannyEdgeDetection
{
    public int KernelSize { get; set; } = 5;
    public double RedRatio { get; set; } = 0.299;
    public double GreenRatio { get; set; } = 0.587;
    public double BlueRatio { get; set; } = 0.114;
    public double Sigma { get; set; } = 1.4;
    public double LowerThreshold { get; set; } = 0.1;
    public double UpperThreshold { get; set; } = 0.3;

    public CannyEdgeDetection() { }

    public CannyEdgeDetection(int kernelSize, double redRatio, double greenRatio, double blueRatio, double sigma,
↪   double lowerThreshold, double upperThreshold)
    {
        KernelSize = kernelSize;
        RedRatio = redRatio;
        GreenRatio = greenRatio;
        BlueRatio = blueRatio;
        Sigma = sigma;
        LowerThreshold = lowerThreshold;
        UpperThreshold = upperThreshold;
    }

    public double[,] BlackWhiteFilter(Structures.RGB[,] input)
    {
        double[,] output = new double[input.GetLength(0), input.GetLength(1)];

        for (int y = 0; y < input.GetLength(0); y++)
        {
            for (int x = 0; x < input.GetLength(1); x++)
            {
                output[y, x] = (input[y, x].R * RedRatio) + (input[y, x].G * GreenRatio) + (input[y, x].B *
↪   BlueRatio);
            }
        }

        return output;
    }

    public double[,] GaussianFilter(double[,] input)
    {
        double[,] output = new double[input.GetLength(0), input.GetLength(1)];

        Matrix gaussianKernel = new Matrix(Kernel<double>.Gaussian(Sigma, KernelSize));
        Kernel<double> masterKernel = new Kernel<double>(input);

        for (int y = 0; y < input.GetLength(0); y++)
        {
            for (int x = 0; x < input.GetLength(1); x++)
            {
                Matrix subKernel = new Matrix(masterKernel.Duplication(x, y, KernelSize));
                double sum = Matrix.Convolution(subKernel, gaussianKernel);
                output[y, x] = sum;
            }
        }

```

```
56          return output;
57      }
58
59      public Structures.Gradients CalculateGradients(double[,] input, Action updateMenu)
60      {
61          Task<double[,]>[] tasks =
62          {
63              new Task<double[,]>(() => CalculateGradientX(input, updateMenu)),
64              new Task<double[,]>(() => CalculateGradientY(input, updateMenu))
65          };
66
67          foreach (var task in tasks) task.Start();
68
69          Task.WaitAll(tasks);
70
71          return new Structures.Gradients
72          {
73              GradientX = tasks[0].Result,
74              GradientY = tasks[1].Result
75          };
76      }
77
78      private double[,] CalculateGradientX(double[,] input, Action updateMenu)
79      {
80          double[,] output = new double[input.GetLength(0), input.GetLength(1)];
81
82          Matrix sobelMatrixY = new Matrix(new double[,] { { 1, 0, -1 }, { 2, 0, -2 }, { 1, 0, -1 } });
83          Kernel<double> masterKernel = new Kernel<double>(input);
84
85          for (int y = 0; y < input.GetLength(0); y++)
86          {
87              for (int x = 0; x < input.GetLength(1); x++)
88              {
89                  Matrix imageKernel = new Matrix(masterKernel.Duplication(x, y, 3));
90                  output[y, x] = Matrix.Convolution(imageKernel, sobelMatrixY);
91              }
92          }
93
94          updateMenu();
95          return output;
96      }
97
98      private double[,] CalculateGradientY(double[,] input, Action updateMenu)
99      {
100         double[,] output = new double[input.GetLength(0), input.GetLength(1)];
101
102         Matrix sobelMatrixY = new Matrix(new double[,] { { 1, 2, 1 }, { 0, 0, 0 }, { -1, -2, -1 } });
103         Kernel<double> masterKernel = new Kernel<double>(input);
104
105         for (int y = 0; y < input.GetLength(0); y++)
106         {
107             for (int x = 0; x < input.GetLength(1); x++)
108             {
109                 Matrix imageKernel = new Matrix(masterKernel.Duplication(x, y, 3));
110                 output[y, x] = Matrix.Convolution(imageKernel, sobelMatrixY);
111             }
112         }
113
114         updateMenu();
115         return output;
```

```
116     }
117
118     public double[,] CombineGradients(Structures.Gradients grads)
119     {
120         if (grads.GradientX.GetLength(0) != grads.GradientY.GetLength(0) || grads.GradientX.GetLength(1) !=
    ↪ grads.GradientY.GetLength(1))
121             throw new ArgumentException("Canny edge detection failed due to arrays not being of the same size.");
122
123         double[,] output = new double[grads.GradientX.GetLength(0), grads.GradientX.GetLength(1)];
124
125         for (int y = 0; y < grads.GradientX.GetLength(0); y++)
126         {
127             for (int x = 0; x < grads.GradientX.GetLength(1); x++)
128             {
129                 output[y, x] = Math.Sqrt(Math.Pow(grads.GradientX[y, x], 2) + Math.Pow(grads.GradientY[y, x], 2));
130             }
131         }
132
133         return output;
134     }
135
136     public double[,] GradientAngle(Structures.Gradients grads)
137     {
138         if (grads.GradientX.GetLength(0) != grads.GradientY.GetLength(0) || grads.GradientX.GetLength(1) !=
    ↪ grads.GradientY.GetLength(1))
139             throw new ArgumentException("Canny edge detection failed due to arrays not being of the same size.");
140
141         double[,] output = new double[grads.GradientX.GetLength(0), grads.GradientX.GetLength(1)];
142
143         for (int y = 0; y < grads.GradientX.GetLength(0); y++)
144         {
145             for (int x = 0; x < grads.GradientX.GetLength(1); x++)
146             {
147                 output[y, x] = Math.Atan2(grads.GradientY[y, x], grads.GradientX[y, x]);
148             }
149         }
150
151         return output;
152     }
153
154     public double[,] MagnitudeThreshold(double[,] gradCombined, double[,] gradAngle)
155     {
156         if (gradCombined.GetLength(0) != gradAngle.GetLength(0) || gradCombined.GetLength(1) !=
    ↪ gradAngle.GetLength(1))
157             throw new ArgumentException("Canny edge detection failed due to arrays not being of the same size.");
158
159         double[,] output = gradCombined;
160         double[,] anglesInDegrees = new double[gradCombined.GetLength(0), gradCombined.GetLength(1)];
161
162         for (int y = 0; y < anglesInDegrees.GetLength(0); y++)
163         {
164             for (int x = 0; x < anglesInDegrees.GetLength(1); x++)
165             {
166                 anglesInDegrees[y, x] = Utility.RadianToDegree(gradAngle[y, x]);
167             }
168         }
169
170         Kernel<double> masterKernel = new Kernel<double>(gradCombined);
171
172         for (int y = 0; y < anglesInDegrees.GetLength(0); y++)
```

```
173         {
174             for (int x = 0; x < anglesInDegrees.GetLength(1); x++)
175             {
176                 double[,] magnitudeKernel = masterKernel.Duplication(x, y, 3);
177 
178                 if (anglesInDegrees[y, x] < 22.5 || anglesInDegrees[y, x] >= 157.5)
179                 {
180                     if (gradCombined[y, x] < magnitudeKernel[1, 2] || gradCombined[y, x] < magnitudeKernel[1, 0])
181                         output[y, x] = 0;
182                 }
183                 else if (anglesInDegrees[y, x] >= 22.5 && anglesInDegrees[y, x] < 67.5)
184                 {
185                     if (gradCombined[y, x] < magnitudeKernel[0, 2] || gradCombined[y, x] < magnitudeKernel[2, 0])
186                         output[y, x] = 0;
187                 }
188                 else if (anglesInDegrees[y, x] >= 67.5 && anglesInDegrees[y, x] < 112.5)
189                 {
190                     if (gradCombined[y, x] < magnitudeKernel[0, 1] || gradCombined[y, x] < magnitudeKernel[2, 1])
191                         output[y, x] = 0;
192                 }
193                 else if (anglesInDegrees[y, x] >= 112.5 && anglesInDegrees[y, x] < 157.5)
194                 {
195                     if (gradCombined[y, x] < magnitudeKernel[0, 0] || gradCombined[y, x] < magnitudeKernel[2, 2])
196                         output[y, x] = 0;
197                 }
198                 else throw new Exception("Critical unknown error occurred, please try again.");
199             }
200         }
201 
202         return output;
203     }
204 
205     public Structures.ThresholdPixel[,] DoubleThreshold(double[,] input)
206     {
207         double min = LowerThreshold * 255;
208         double max = UpperThreshold * 255;
209 
210         Structures.ThresholdPixel[,] output = new Structures.ThresholdPixel[input.GetLength(0),
    ↪ input.GetLength(1)];
211 
212         for (int y = 0; y < input.GetLength(0); y++)
213         {
214             for (int x = 0; x < input.GetLength(1); x++)
215             {
216                 if (input[y, x] < min) output[y, x] = new Structures.ThresholdPixel { Strong = false, Value = 0 };
217                 else if (input[y, x] > min && input[y, x] < max) output[y, x] = new Structures.ThresholdPixel {
    ↪ Strong = false, Value = input[y, x] };
218                 else if (input[y, x] > max) output[y, x] = new Structures.ThresholdPixel { Strong = true, Value =
    ↪ input[y, x] };
219                 else throw new Exception("Critical unknown error occurred, please try again.");
220             }
221         }
222 
223         return output;
224     }
225 
226     public double[,] EdgeTrackingHysteresis(Structures.ThresholdPixel[,] input)
227     {
228         double[,] output = new double[input.GetLength(0), input.GetLength(1)];
229 
```

```
230         Kernel<Structures.ThresholdPixel> masterKernel = new Kernel<Structures.ThresholdPixel>(input);
231 
232         for (int i = 0; i < input.GetLength(0); i++)
233         {
234             for (int j = 0; j < input.GetLength(1); j++)
235             {
236                 if (input[i, j].Strong == false)
237                 {
238                     Structures.ThresholdPixel[,] imageKernel = masterKernel.Duplication(j, i, 3);
239                     bool strong = false;
240                     for (int k = 0; k < 3 && !strong; k++)
241                     {
242                         for (int l = 0; l < 3 && !strong; l++)
243                         {
244                             if (imageKernel[k, l].Strong) strong = true;
245                         }
246                     }
247                     output[i, j] = strong ? 255 : 0;
248                 }
249                 else output[i, j] = 255;
250             }
251         }
252 
253         return output;
254     }
255 }
```

Post.cs

```
public class Post
{
    private double[,] _imageDoubles;

    public Post(double[,] input)
    {
        _imageDoubles = input;
    }

    public void Start(int embossCount)
    {
        if (embossCount <= 0) _imageDoubles = FillPixelGaps(_imageDoubles);
        else
        {
            for (int i = 0; i < embossCount; i++)
            {
                _imageDoubles = FillPixelGaps(EmbossImage(_imageDoubles));
            }
        }
    }

    private double[,] EmbossImage(double[,] input)
    {
        double[,] result = new double[input.GetLength(0), input.GetLength(1)];

        Matrix embossMatrix = new Matrix(new double[,] { { -2, -1, 0 }, { -1, 1, 1 }, { 0, 1, 2 } });
        Kernel<double> masterKernel = new Kernel<double>(input);

        for (int y = 0; y < input.GetLength(0); y++)
        {
            for (int x = 0; x < input.GetLength(1); x++)
            {
                Matrix imageKernel = new Matrix(masterKernel.Duplication(x, y, 3));
                result[y, x] = Math.Abs(Matrix.Convolution(imageKernel, embossMatrix));
            }
        }

        return result;
    }

    private double[,] FillPixelGaps(double[,] input)
    {
        double[,] output = new double[input.GetLength(0), input.GetLength(1)];
        Kernel<double> masterKernel = new Kernel<double>(input);


        for (int y = 0; y < input.GetLength(0); y++)
        {
            for (int x = 0; x < input.GetLength(1); x++)
            {
                Matrix imageKernel = new Matrix(masterKernel.Duplication(x, y, 3));
                int count = imageKernel.Cast<double>().Count(value => value >= 255);
                if (count > 4) output[y, x] = 255;
            }
        }

        return output;
    }

```

```
60
61      public double[,] Result() => _imageDoubles;
62
63  }
```

Pre.cs

```
public class Pre
{
    private readonly string _imagePath;
    private Bitmap _imageBitmap;
    private Structures.RGB[,] _imageRgb;

    private const string FileExtensionRegex =
    @"^([a-z]:\\|\\|[a-z]|\.\.(\\|\/)|\.(\\|\/))((\w|(\\|\/))+)\.(jpg|bmp|exif|png|tiff)$";

    public Pre(string imagePath)
    {
        _imagePath = imagePath;
    }

    public void Start(Action updateProgressAction)
    {
        updateProgressAction();
        ValidatePath();
        updateProgressAction();
        ReadImage();
        updateProgressAction();
        CheckDimensions();
        updateProgressAction();
    }

    private void ValidatePath()
    {
        Regex fileRegex = new Regex(FileExtensionRegex, RegexOptions.IgnoreCase);

        if (!File.Exists(_imagePath)) throw new PreprocessingException("The image that you entered does not exist,
    double check the path to the file and that exists.");
        if (!fileRegex.IsMatch(_imagePath)) throw new PreprocessingException("The file which you entered does not
    appear to be an image file. It should end in .jpg, .bmp, .exif, .png or .tiff double check and try again.");
    }

    private void ReadImage()
    {
        _imageBitmap = new Bitmap(_imagePath, true);
        _imageRgb = new Structures.RGB[_imageBitmap.Height, _imageBitmap.Width];

        for (int y = 0; y < _imageBitmap.Height; y++)
        {
            for (int x = 0; x < _imageBitmap.Width; x++)
            {
                Color tempPixel = _imageBitmap.GetPixel(x, y);
                _imageRgb[y, x] = new Structures.RGB
                {
                    R = tempPixel.R,
                    G = tempPixel.G,
                    B = tempPixel.B
                };
            }
        }
    }

    private void CheckDimensions()
    {
        if (_imageRgb.GetLength(0) < 200 || _imageRgb.GetLength(1) < 200)
```

```
56              throw new PreprocessingException("The image you supplied is too small to work properly it must be at
↪  least 200x200. Try a larger image.");
57
58          if (_imageRgb.GetLength(0) % 2 != 0 || _imageRgb.GetLength(1) % 2 != 0)
59          {
60              Structures.RGB[,] resizedRgb =
61                  new Structures.RGB[_imageRgb.GetLength(0) / 2 * 2, _imageRgb.GetLength(1) / 2 * 2];
62
63              for (int y = 0; y < _imageRgb.GetLength(0) / 2 * 2; y++)
64              {
65                  for (int x = 0; x < _imageRgb.GetLength(1) / 2 * 2; x++)
66                  {
67                      resizedRgb[y, x] = _imageRgb[y, x];
68                  }
69              }
70
71              _imageRgb = resizedRgb;
72          }
73      }
74
75      public Structures.RawImage Result() => new Structures.RawImage
76      {
77          Original = _imageBitmap,
78          Pixels = _imageRgb,
79          Path = _imagePath,
80          Height = _imageBitmap.Height,
81          Width = _imageBitmap.Width
82      };
83  }
```

RoadDetection.cs

```
public class RoadDetection
{
    private Bitmap _filledBitmap;
    private Bitmap _pathBitmap;
    private double[,] _pathDoubles;
    private readonly double[,] _imageDoubles;
    private readonly double _threshold;
    private Random _gen = new Random();

    public RoadDetection(double[,] imageDoubles, double threshold)
    {
        _imageDoubles = imageDoubles;
        _threshold = threshold;
    }

    public void Start(Action updateAction)
    {
        List<Color> toRemoveColors = FillImage(updateAction);
        RemoveColor(toRemoveColors, updateAction);

        _pathDoubles = new double[_pathBitmap.Height, _pathBitmap.Width];
        for (int y = 0; y < _pathBitmap.Height; y++)
        {
            for (int x = 0; x < _pathBitmap.Width; x++)
            {
                Color pixel = _pathBitmap.GetPixel(x, y);
                if (pixel == Color.FromArgb(0, 0, 0)) _pathDoubles[y, x] = 0;
                else _pathDoubles[y, x] = 255;
            }
        }
    }

    private List<Color> FillImage(Action updateAction)
    {
        Color[,] tempImage = new Color[_imageDoubles.GetLength(0), _imageDoubles.GetLength(1)];

        for (int y = 0; y < _imageDoubles.GetLength(0); y++)
            for (int x = 0; x < _imageDoubles.GetLength(1); x++)
                tempImage[y, x] = Color.FromArgb((int)_imageDoubles[y, x], (int)_imageDoubles[y, x],
↪  (int)_imageDoubles[y, x]);

        List<Color> toReplaceColors = new List<Color>();
        List<Color> usedColors = new List<Color>();

        _filledBitmap = _imageDoubles.ToBitmap();

        for (int y = 0; y < _imageDoubles.GetLength(0); y++)
        {
            for (int x = 0; x < _imageDoubles.GetLength(1); x++)
            {
                if ((((y + 1) * (x + 1)) / 100) % 100 == 0) updateAction();

                int minX = _imageDoubles.GetLength(1), maxX = 0, minY = _imageDoubles.GetLength(0), maxY = 0;
                int filled = 0;

                Color randCol = Color.FromArgb(_gen.Next(56, 256), _gen.Next(56, 256), _gen.Next(56, 256));
                while (usedColors.Contains(randCol))
                    randCol = Color.FromArgb(_gen.Next(56, 256), _gen.Next(56, 256), _gen.Next(56, 256));

```

```
59                  Datatypes.Queue<(int, int)> queue = new Datatypes.Queue<(int, int)>();
60                  queue.Enqueue((y, x));
61
62                  while (queue.Size > 0)
63                  {
64                      (int, int) cord = queue.Dequeue();
65                      if (tempImage[cord.Item1, cord.Item2] == Color.FromArgb(0, 0, 0))
66                      {
67                          tempImage[cord.Item1, cord.Item2] = randCol;
68                          _filledBitmap.SetPixel(cord.Item2, cord.Item1, tempImage[cord.Item1, cord.Item2]);
69
70                          if (cord.Item1 > 0) queue.Enqueue((cord.Item1 - 1, cord.Item2));
71                          if (cord.Item2 > 0) queue.Enqueue((cord.Item1, cord.Item2 - 1));
72                          if (cord.Item1 < _filledBitmap.Height - 1) queue.Enqueue((cord.Item1 + 1, cord.Item2));
73                          if (cord.Item2 < _filledBitmap.Width - 1) queue.Enqueue((cord.Item1, cord.Item2 + 1));
74
75                          if (!usedColors.Contains(randCol)) usedColors.Add(randCol);
76
77                          filled++;
78                      }
79                      else if (tempImage[cord.Item1, cord.Item2] == Color.FromArgb(255, 255, 255))
80                      {
81                          tempImage[cord.Item1, cord.Item2] = Color.FromArgb(1, 1, 1);
82                          _filledBitmap.SetPixel(cord.Item2, cord.Item1, tempImage[cord.Item1, cord.Item2]);
83                      }
84
85                      if (cord.Item1 > maxY) maxY = cord.Item1;
86                      if (cord.Item2 > maxX) maxX = cord.Item2;
87                      if (cord.Item1 < minY) minY = cord.Item1;
88                      if (cord.Item2 < minX) minX = cord.Item2;
89                  }
90
91                  double totalSquares = (maxX - minX) * (maxY - minY);
92                  if (filled / totalSquares > _threshold || filled == 1) toReplaceColors.Add(randCol);
93              }
94          }
95
96          return toReplaceColors;
97      }
98
99      private void RemoveColor(List<Color> toRemove, Action updateAction)
100     {
101         _pathBitmap = new Bitmap(_filledBitmap);
102
103         for (int y = 0; y < _pathBitmap.Height; y++)
104         {
105             for (int x = 0; x < _pathBitmap.Width; x++)
106             {
107                 if ((((y + 1) * (x + 1)) / 100) % 100 == 0) updateAction();
108                 if (toRemove.Contains(_pathBitmap.GetPixel(x, y)))
109                 {
110                     _pathBitmap.SetPixel(x, y, Color.FromArgb(1, 1, 1));
111                 }
112             }
113         }
114
115         for (int i = 0; i < _pathBitmap.Height; i++)
116         {
117             for (int j = 0; j < _pathBitmap.Width; j++)
118             {
```

```
119                 if ((((i + 1) * (j + 1)) / 100) % 100 == 0) updateAction();
120                 if (_pathBitmap.GetPixel(j, i) == Color.FromArgb(1, 1, 1))
121                     _pathBitmap.SetPixel(j, i, Color.FromArgb(0, 0, 0));
122             }
123         }
124     }
125 
126     public Structures.RoadResult Result() => new Structures.RoadResult
127     {
128         FilledBitmap = _filledBitmap,
129         PathBitmap = _pathBitmap,
130         PathDoubles = _pathDoubles
131     };
132 }
```

6.2.1.6 Root

Extensions.cs

```
public static class Extensions
{
    public static Bitmap ToBitmap(this double[,] array)
    {
        Bitmap output = new Bitmap(array.GetLength(1), array.GetLength(0));

        for (int y = 0; y < array.GetLength(0); y++)
        {
            for (int x = 0; x < array.GetLength(1); x++)
            {
                int boundedPixel = (int)Utility.Bound(0, 255, array[y, x]);
                output.SetPixel(x, y, Color.FromArgb(boundedPixel, boundedPixel, boundedPixel));
            }
        }

        return output;
    }

    public static double[,] ToDoubles(this Bitmap image, Func<Color, double> getPixelFunction)
    {
        double[,] result = new double[image.Height, image.Width];

        for (int y = 0; y < image.Height; y++)
        {
            for (int x = 0; x < image.Width; x++)
            {
                result[y, x] = getPixelFunction(image.GetPixel(x, y));
            }
        }

        return result;
    }

    public static Bitmap ToBitmap(this Structures.RGB[,] array)
    {
        Bitmap output = new Bitmap(array.GetLength(1), array.GetLength(0));

        for (int y = 0; y < array.GetLength(0); y++)
        {
            for (int x = 0; x < array.GetLength(1); x++)
            {
                output.SetPixel(x, y, Color.FromArgb((int)array[y, x].R, (int)array[y, x].G, (int)array[y, x].B));
            }
        }

        return output;
    }

    public static Graph<Structures.Coord> ToGraph(this double[,] doubles)
    {
        Graph<Structures.Coord> output = new Graph<Structures.Coord>();
        Kernel<double> masterKernel = new Kernel<double>(doubles);

        for (int y = 0; y < doubles.GetLength(0); y++)
        {
            for (int x = 0; x < doubles.GetLength(1); x++)
            {
```

```
58                  Structures.Coord tempCord = new Structures.Coord { X = x, Y = y };
59                  output.AddNode(tempCord);
60
61                  double[,] surroundingDoubles = masterKernel.Constant(x, y, 3, 0);
62
63                  bool found = false;
64
65                  if (doubles[y, x] == 255)
66                  {
67                      for (int i = 0; i < 9; i++)
68                      {
69                          if (surroundingDoubles[i / 3, i % 3] != 0 && i != 4)
70                          {
71                              output.AddConnection(tempCord, new Structures.Coord { X = (x + (i % 3)) - 1, Y = (y +
    ↪  (i / 3)) - 1 });
72                              found = true;
73                          }
74                      }
75                  }
76
77                  if (!found) output.RemoveNode(tempCord);
78              }
79          }
80
81          return output;
82      }
83
84      // To ensure compatibility with BITMAP
85      public static void SetPixel(this Structures.RGB[,] pixels, int x, int y, Structures.RGB toSetPixel) =>
    ↪  pixels[y, x] = toSetPixel;
86
87      public static Structures.RGB GetPixel(this Structures.RGB[,] pixels, int x, int y) => pixels[y, x];
88  }
```

Kernel.cs

```
public class Kernel<T>
{
    private readonly T[,] _image;
    private readonly int _width;
    private readonly int _height;

    public Kernel(T[,] image)
    {
        _image = image;
        _height = image.GetLength(0);
        _width = image.GetLength(1);
    }

    public T[,] Constant(int x, int y, int size, T constant = default)
    {
        if (size % 2 != 1) throw new KernelException("The image kernel supplied was of an odd size, check your settings and try again.");
        if (x >= _width || x < 0 || y >= _height || y < 0)
            throw new KernelException("Your kernel must start within the image.");

        T[,] kernel = new T[size, size];

        int halfK = size / 2;

        for (int i = 0; i < size; i++)
            for (int j = 0; j < size; j++)
                kernel[i, j] = constant;

        int cntY = 0;
        for (int j = y - halfK; j <= y + halfK; j++)
        {
            int cntX = 0;
            for (int i = x - halfK; i <= x + halfK; i++)
            {
                if (j >= 0 && i >= 0 && j < _height && i < _image.GetLength(1))
                {
                    kernel[cntY, cntX] = _image[j, i];
                }
                cntX++;
            }
            cntY++;
        }

        return kernel;
    }

    public T[,] Duplication(int x, int y, int size)
    {
        if (size % 2 != 1) throw new KernelException("The image kernel supplied was of an odd size, check your settings and try again.");
        if (x >= _width || x < 0 || y >= _height || y < 0)
            throw new KernelException("Your kernel must start within the image.");

        T[,] kernel = new T[size, size];

        int halfK = size / 2;

        for (int i = 0; i < size; i++) for (int j = 0; j < size; j++) kernel[i, j] = _image[y, x];

```

```
58          int cntY = 0;
59          for (int j = y - halfK; j <= y + halfK; j++)
60          {
61              int cntX = 0;
62              for (int i = x - halfK; i <= x + halfK; i++)
63              {
64                  if (j >= 0 && i >= 0 && j < _height && i < _image.GetLength(1))
65                  {
66                      kernel[cntY, cntX] = _image[j, i];
67                  }
68                  cntX++;
69              }
70              cntY++;
71          }
72
73          return kernel;
74      }
75
76      public static double[,] Gaussian(double sigma, int size)
77      {
78          double[,] result = new double[size, size];
79          int halfK = size / 2;
80
81          double sum = 0;
82
83          int cntY = -halfK;
84          for (int i = 0; i < size; i++)
85          {
86              int cntX = -halfK;
87              for (int j = 0; j < size; j++)
88              {
89                  result[halfK + cntY, halfK + cntX] = Utility.GaussianDistribution(cntX, cntY, sigma);
90                  sum += result[halfK + cntY, halfK + cntX];
91                  cntX++;
92              }
93              cntY++;
94          }
95
96          for (int i = 0; i < size; i++) for (int j = 0; j < size; j++) result[i, j] /= sum;
97          return result;
98      }
99
100 }
```

Logger.cs

```
public class Logger
{
    private readonly bool _localApplication;
    private static readonly object Lock = new object();
    public Logger(bool local)
    {
        _localApplication = local;
        CreateDirStructure();
    }

    private void CreateDirStructure()
    {
        Directory.CreateDirectory("./runs");
        Directory.CreateDirectory("./logs");
        Directory.CreateDirectory("./saves");

        string mode = _localApplication ? "Local Application" : "Web Application";

        lock (Lock)
        {
            using (StreamWriter sr = File.AppendText("./logs/master.txt"))
            {
                sr.WriteLine("<====================== New Instance ======================>");
                sr.WriteLine($"Datetime: {DateTime.UtcNow:dd-MM-yyyy} {DateTime.UtcNow:HH:mm:ss}");
                sr.WriteLine($"Mode: {mode}");
            }
        }
    }

    public static Guid CreateRun()
    {
        Guid guidForRun = Uuid();

        Directory.CreateDirectory($"./runs/{guidForRun.ToString("N").ToUpper()}");

        WriteLineToRunFile(guidForRun, "<====================== Begin New Run ======================>");
        WriteLineToRunFile(guidForRun, $"Datetime: {DateTime.UtcNow:dd-MM-yyyy} {DateTime.UtcNow:HH:mm:ss}");
        WriteLineToRunFile(guidForRun, $"Run Object Guid: {guidForRun.ToString().ToUpper()}");

        WriteLineToMaster($"New Run Started with GUID {guidForRun.ToString().ToUpper()}");

        return guidForRun;
    }

    public static void WriteLineToRunFile(Guid currentGuid, string message)
    {
        lock (Lock)
        {
            using (StreamWriter sr = File.AppendText($"./logs/{currentGuid}.txt"))
                sr.WriteLine($"{message}");
        }
    }

    public static void WriteLineToMaster(string message)
    {
        lock (Lock)
        {
            using (StreamWriter sr = File.AppendText("./logs/master.txt"))
                sr.WriteLine($"{DateTime.UtcNow:HH:mm:ss} || {message}");
```

```
60          }
61
62      }
63
64      public static void SaveBitmap(Guid currentGuid, double[,] image, string name)
65      {
66          Bitmap toSaveBitmap = image.ToBitmap();
67          if (!Directory.Exists($"./runs/{currentGuid.ToString("N").ToUpper()}"))
68              throw new LoggerException("Run directory not found, logger not created correctly, please restart the
    ↪  program.");
69
70          toSaveBitmap.Save($"./runs/{currentGuid.ToString("N").ToUpper()}/{name}.png");
71      }
72
73      public static void SaveBitmap(Guid currentGuid, Bitmap image, string name)
74      {
75          if (!Directory.Exists($"./runs/{currentGuid.ToString("N").ToUpper()}"))
76              throw new LoggerException("Run directory not found, logger not created correctly, please restart the
    ↪  program.");
77
78          image.Save($"./runs/{currentGuid.ToString("N").ToUpper()}/{name}.png");
79      }
80
81      public static Guid Uuid() => Guid.NewGuid();
82  }
```

Structures.cs

```
public class Structures
{
    public struct ThresholdPixel
    {
        public bool Strong;
        public double Value;
    }

    public struct RGB
    {
        public double R;
        public double G;
        public double B;
    }

    public struct Gradients
    {
        public double[,] GradientX;
        public double[,] GradientY;
    }

    public struct RawImage
    {
        public Bitmap Original;
        public string Path;
        public RGB[,] Pixels;
        public int Width;
        public int Height;
        public MapFile MapFile;
    }

    public struct RoadResult
    {
        public Bitmap FilledBitmap;
        public Bitmap PathBitmap;
        public double[,] PathDoubles;
    }

    public struct CannyResult
    {
        public Bitmap BitmapImage;
        public double[,] DoubleImage;
    }

    public struct Coord
    {
        public int X;
        public int Y;

        public override string ToString() => $"({X}, {Y})";
        public bool Equals(Coord other) => X == other.X && Y == other.Y;
        public override bool Equals(object obj) => obj is Coord other && Equals(other);
        public static bool operator ==(Coord lhs, Coord rhs) => lhs.X == rhs.X && lhs.Y == rhs.Y;
        public static bool operator !=(Coord lhs, Coord rhs) => !(lhs == rhs);
        public override int GetHashCode()
        {
            unchecked
            {
                return (X * 397) ^ Y;
```

```
60                  }
61              }
62          }
63      }
64
```

Utility.cs

```
public static class Utility
{
    public static double GaussianDistribution(int x, int y, double sigma) =>
        1 / (2 * Math.PI * sigma * sigma) * Math.Exp(-((Math.Pow(x, 2) + Math.Pow(y, 2)) / (2 * sigma * sigma)));

    public static double Bound(int l, int h, double v) => v > h ? h : v < l ? l : v;

    public static bool TryBound(int l, int h, double v, out double value)
    {
        if (v < h && v > l) value = v;
        else value = v > h ? h : l;
        return v < h && v > l;
    }

    public static double RadianToDegree(double input) => 180 * input / Math.PI;

    public static double DegreeToRadian(double input) => input * Math.PI / 180;

    public static double MapRadiansToPixel(double input) => (int)(128 / (2 * Math.PI) * input + 128);

    public static Bitmap CombineBitmap(Bitmap a, Bitmap b)
    {
        if (a.Width != b.Width || a.Height != b.Height)
            throw new ArgumentException($"An error has occurred somewhere in the map images aren't of the same size
↪  ({a.Width}x{a.Height} vs {b.Width}x{b.Height}) please try again.");

        Bitmap result = new Bitmap(a);
        for (int y = 0; y < a.Height; y++)
        {
            for (int x = 0; x < a.Width; x++)
            {
                Color pixel = b.GetPixel(x, y);
                if (pixel != Color.FromArgb(0, 0, 0))
                {
                    result.SetPixel(x, y, pixel);
                }
            }
        }

        return result;
    }

    public static Structures.RGB[][,] SplitImage(Structures.RGB[,] image)
    {
        Structures.RGB[,] one = new Structures.RGB[image.GetLength(0) / 2, image.GetLength(1) / 2];
        Structures.RGB[,] beta = new Structures.RGB[image.GetLength(0) / 2, image.GetLength(1) / 2];
        Structures.RGB[,] gamma = new Structures.RGB[image.GetLength(0) / 2, image.GetLength(1) / 2];
        Structures.RGB[,] delta = new Structures.RGB[image.GetLength(0) / 2, image.GetLength(1) / 2];

        for (int i = 0; i < image.GetLength(1) / 2; i++)
        {
            for (int j = 0; j < image.GetLength(0) / 2; j++)
            {
                one.SetPixel(i, j, image.GetPixel(i, j));
            }
        }

        for (int i = image.GetLength(1) / 2; i < image.GetLength(1); i++)
        {
```

```
            for (int j = 0; j < image.GetLength(0) / 2; j++)
            {
                beta.SetPixel(i - (image.GetLength(1) / 2), j, image.GetPixel(i, j));
            }
        }

        for (int i = 0; i < image.GetLength(1) / 2; i++)
        {
            for (int j = image.GetLength(0) / 2; j < image.GetLength(0); j++)
            {
                gamma.SetPixel(i, j - (image.GetLength(0) / 2), image.GetPixel(i, j));
            }
        }

        for (int i = image.GetLength(1) / 2; i < image.GetLength(1); i++)
        {
            for (int j = image.GetLength(0) / 2; j < image.GetLength(0); j++)
            {
                delta.SetPixel(i - (image.GetLength(1) / 2), j - (image.GetLength(0) / 2), image.GetPixel(i, j));
            }
        }

        return new[] { one, beta, gamma, delta };
    }

    public static double[,] CombineQuadrants(double[,] alpha, double[,] beta, double[,] gamma, double[,] delta)
    {
        double[,] partA = new double[alpha.GetLength(0), alpha.GetLength(1) * 2];
        double[,] partB = new double[alpha.GetLength(0), alpha.GetLength(1) * 2];
        for (int i = 0; i < alpha.GetLength(0); i++)
        {
            for (int j = 0; j < alpha.GetLength(1); j++)
                partA[i, j] = alpha[i, j];

            for (int y = 0; y < beta.GetLength(1); y++)
                partA[i, y + alpha.GetLength(1)] = beta[i, y];
        }

        for (int i = 0; i < gamma.GetLength(0); i++)
        {
            for (int j = 0; j < gamma.GetLength(1); j++)
                partB[i, j] = gamma[i, j];

            for (int y = 0; y < delta.GetLength(1); y++)
                partB[i, y + gamma.GetLength(1)] = delta[i, y];
        }

        double[,] final = new double[alpha.GetLength(0) * 2, alpha.GetLength(1) * 2];
        for (int i = 0; i < alpha.GetLength(0) * 2; i++)
        {
            if (i < alpha.GetLength(0) * 2 / 2)
            {
                for (int j = 0; j < alpha.GetLength(1) * 2; j++)
                {
                    final[i, j] = partA[i, j];
                }
            }
            else
            {
                for (int j = 0; j < alpha.GetLength(1) * 2; j++)
```

```
                {
                    final[i, j] = partB[i - alpha.GetLength(0) * 2 / 2, j];
                }
            }
        }

        return final;
    }

    public static double[,] InverseImage(double[,] image)
    {
        for (int y = 0; y < image.GetLength(0); y++)
        {
            for (int x = 0; x < image.GetLength(1); x++)
            {
                image[y, x] = image[y, x] == 255 ? 0 : 255;
            }
        }

        return image;
    }

    public static T[] RebuildPath<T>(Dictionary<T, T> prev, T goal)
    {
        if (prev == null) return new T[1];
        List<T> sequence = new List<T>();
        T u = goal;

        while (prev.ContainsKey(u))
        {
            sequence.Insert(0, u);
            u = prev[u];
        }

        return sequence.ToArray();
    }


    public static bool IsYes(string input) => new Regex(@"^y(es)?$",
↪   RegexOptions.IgnoreCase).IsMatch(input.Trim());
    public static double GetRed(Color pixel) => pixel.R;
    public static double GetGreen(Color pixel) => pixel.G;
    public static double GetBlue(Color pixel) => pixel.B;
    public static double GetAverage(Color pixel) => (pixel.R + pixel.G + pixel.B) / 3.0;
    public static double GetIndustryAverage(Color pixel) => (pixel.R * 0.299) + (pixel.G * 0.586) + (pixel.B *
↪   0.114);
    public static double GetIfExists(Color pixel) => GetAverage(pixel) > 0 ? 255 : 0;

    public static double GetDistanceBetweenNodes(Structures.Coord a, Structures.Coord b) =>
        Math.Sqrt(Math.Pow(a.X - b.X, 2) + Math.Pow(a.Y - b.Y, 2));
}
```

6.2.2 LocalApp

6.2.2.1 Actions

NewImage.cs

```
internal class NewImage
{
    private readonly Guid _runGuid;
    private readonly Menu _menuInstance;
    private readonly Log _logInstance;

    public NewImage(Menu menu, Log logger, Guid runGuid)
    {
        _runGuid = runGuid;
        _menuInstance = menu;
        _logInstance = logger;
    }

    public Structures.RawImage Read()
    {
        Input inputHandel = new Input(_menuInstance);

        string path =
            inputHandel.GetInput(
                "Please enter the path of the image you wish to process into a map (you can click and drag an image from your file explorer here too):");
        _logInstance.Event(_runGuid, $"Looking for image at {path}");

        Pre preProcess = new Pre(path);

        ProgressBar progressBar = new ProgressBar("Pre-processing your image", 4, _menuInstance);
        progressBar.DisplayProgress();

        try
        {
            preProcess.Start(progressBar.GetIncrementAction());
            _logInstance.Event(_runGuid, "Completed pre processing of image.");
        }
        catch (PreprocessingException ex)
        {
            _logInstance.Error(_runGuid, ex.Message);
            throw new Exception("An expected occurred while pre processing your image.", ex);
        }
        catch (Exception ex)
        {
            _logInstance.Error(ex.Message);
            throw new Exception("An unexpected occurred while pre processing your image.", ex);
        }

        _menuInstance.ClearUserSection();

        bool saveAsBinary =
            Utility.IsYes(
                inputHandel.TryGetInput(
                    "Would you like to save this map afterwards in a file to be reused later (y/n)?"));
        MapFile mapSave = saveAsBinary ? new MapFile() : null;

        if (saveAsBinary)
        {
            mapSave.Type = inputHandel.GetOption("What type of image are you supplying:",
```

```
55                  new[] { "Screenshot", "Hand Drawn", "Photograph", "Other" });
56
57              mapSave.Name = inputHandel.TryGetInput("Enter a name for image, or leave blank for 'None':");
58              _menuInstance.WriteLine();
59
60              mapSave.Description = inputHandel.TryGetInput("Enter a brief description about this image, or leave
    ↪  blank for 'None':");
61          }
62
63          Structures.RawImage result = preProcess.Result();
64          if (saveAsBinary) result.MapFile = mapSave;
65          else result.MapFile = null;
66          if (saveAsBinary) mapSave.OriginalImage = result.Pixels.ToBitmap();
67
68          return result;
69      }
70  }
```

SaveFile.cs

```
public class SaveFile
{
    private readonly Guid _runGuid;
    private readonly Menu _menuInstance;
    private readonly Log _logInstance;

    public SaveFile(Menu menu, Log logger, Guid runGuid)
    {
        _runGuid = runGuid;
        _menuInstance = menu;
        _logInstance = logger;
    }

    public MapFile Read()
    {
        Input inputHandel = new Input(_menuInstance);

        string path = inputHandel.GetInput("Please enter the path of the map which you wish to recall:");
        _logInstance.Event(_runGuid, $"Looking for map file at {path}");

        ProgressBar progressBar = new ProgressBar("Recalling Saved Map File", 10, _menuInstance);
        progressBar.DisplayProgress();

        MapFile result = new MapFile(path);

        try
        {
            result.Initialize(progressBar.GetIncrementAction());
            _logInstance.Event(_runGuid, "Completed recollection.");
        }
        catch (MapFileException ex)
        {
            _logInstance.Error(_runGuid, ex.Message);
            throw new Exception("An expected occurred while recalling your save file.", ex);
        }
        catch (Exception ex)
        {
            _logInstance.Error(ex.Message);
            throw new Exception("An unexpected occurred while recalling your save file.", ex);
        }


        return result;
    }

}
```

SettingsControl.cs

```
public class SettingsControl
{
    private readonly Settings _settings;
    private readonly Menu _menuInstance;
    private readonly Log _logInstance;
    private readonly Input _inputHandel;

    private readonly Dictionary<string, (string, Type)> _oldSettings;

    public SettingsControl(Settings settings, Menu menuInstance, Log logInstance)
    {
        _settings = settings;
        _menuInstance = menuInstance;
        _logInstance = logInstance;
        _oldSettings = new Dictionary<string, (string, Type)>(Settings.UserSettings);
        _inputHandel = new Input(_menuInstance);
    }

    public void Start()
    {
        bool running = true;

        while (running)
        {
            _menuInstance.SetPage("Settings Home Page");
            int opt = _inputHandel.GetOption("Whcih settings would you like to change?",
                new[]
                {
                    "General",
                    "Pathfinding",
                    "Save",
                    "Algorithm",
                    "Exit"
                }
            );

            switch (opt)
            {
                case 0:
                    _menuInstance.SetPage("Settings -> General Settings");
                    General();
                    break;
                case 1:
                    _menuInstance.SetPage("Settings -> Pathfinding Settings");
                    Pathfinding();
                    break;
                case 2:
                    _menuInstance.SetPage("Settings -> Save Settings");
                    Save();
                    break;
                case 3:
                    _menuInstance.SetPage("Settings -> Pathfinding Algorithm");

                    int algorithmOption = _inputHandel.GetOption("Select which pathfinding algorithm you wish to
↪  use:", new string[] {
                        "Dijkstra",
                        "AStar"
                    });

```

```
59                      string newValue = algorithmOption == 0 ? "Dijkstra" : "AStar";
60
61                      _settings.Change("pathfindingAlgorithm", newValue);
62                      break;
63                  default:
64                      running = false;
65
66                      _settings.Update(_oldSettings, Settings.UserSettings);
67
68                      break;
69
70              }
71          }
72      }
73
74      private void General()
75      {
76          (string, bool)[] settings = new (string, bool)[] {
77              ( "detailedLogging", bool.Parse(Settings.UserSettings["detailedLogging"].Item1)),
78              ( "forceFormsFront", bool.Parse(Settings.UserSettings["forceFormsFront"].Item1)),
79          };
80
81          IEnumerable<(string, bool)> result = _inputHandel.OptionSelector("General Settings:", settings);
82          foreach (var item in result) _settings.Change(item.Item1, item.Item2);
83      }
84
85      private void Pathfinding()
86      {
87          (string, bool)[] settings = new (string, bool)[] {
88              ( "convertToMST", bool.Parse(Settings.UserSettings["convertToMST"].Item1)),
89              ( "snapToGrid", bool.Parse(Settings.UserSettings["snapToGrid"].Item1)),
90              ( "endOnFind", bool.Parse(Settings.UserSettings["endOnFind"].Item1)),
91          };
92
93          IEnumerable<(string, bool)> result = _inputHandel.OptionSelector("Save File Settings:", settings);
94          foreach (var item in result) _settings.Change(item.Item1, item.Item2);
95      }
96
97      private void Save()
98      {
99          (string, bool)[] settings = new (string, bool)[] {
100             ( "shortNames", bool.Parse(Settings.UserSettings["shortNames"].Item1)),
101             ( "zipOnComplete", bool.Parse(Settings.UserSettings["zipOnComplete"].Item1)),
102         };
103
104         IEnumerable<(string, bool)> result = _inputHandel.OptionSelector("Save File Settings:", settings);
105         foreach (var item in result) _settings.Change(item.Item1, item.Item2);
106     }
107
108 }
```

6.2.2.2 CLI

Input.cs

```
public class Input
{
    private readonly Menu _menuInstance;

    public Input(Menu menuInstance)
    {
        _menuInstance = menuInstance;
    }

    public int GetOption(string title, IEnumerable<string> options, bool clear = true)
    {
        while (Console.KeyAvailable) Console.ReadKey(true);
        _menuInstance.ClearUserSection();
        _menuInstance.WriteLine(title);

        int j = 3;

        lock (_menuInstance.ScreenLock)
        {
            foreach (var option in options)
            {
                Console.SetCursorPosition(1, j++);
                Console.WriteLine($"  {option}");
            }
        }

        bool selected = false;
        int currentTop;

        lock (_menuInstance.ScreenLock)
        {
            Console.SetCursorPosition(1, 3);
            Console.Write('>');

            currentTop = Console.CursorTop;
        }

        while (!selected)
        {
            Console.CursorVisible = false;

            ConsoleKeyInfo key = Console.ReadKey(true);
            if (key.Key == ConsoleKey.DownArrow && currentTop < options.Count() + 2)
            {
                lock (_menuInstance.ScreenLock)
                {
                    Console.CursorLeft = 1;
                    Console.CursorTop = currentTop;
                    Console.Write(' ');
                    Console.CursorTop = ++currentTop;
                    Console.CursorLeft = 1;
                    Console.Write('>');
                }
            }
            else if (key.Key == ConsoleKey.UpArrow && currentTop > 3)
            {
                lock (_menuInstance.ScreenLock)
```

```
58                  {
59                      Console.CursorLeft = 1;
60                      Console.CursorTop = currentTop;
61                      Console.Write(' ');
62                      Console.CursorTop = --currentTop;
63                      Console.CursorLeft = 1;
64                      Console.Write('>');
65                  }
66              }
67              else if (key.Key == ConsoleKey.Enter)
68              {
69                  if (clear) _menuInstance.ClearUserSection();
70                  Console.CursorVisible = false;
71
72                  selected = true;
73              }
74          }
75
76          return currentTop - 3;
77      }
78
79      public void WaitInput(string prompt)
80      {
81          while (Console.KeyAvailable) Console.ReadKey(true);
82          _menuInstance.WriteLine(prompt);
83          bool complete = false;
84
85          while (!complete)
86          {
87              if (!Console.KeyAvailable) continue;
88              ConsoleKeyInfo key = Console.ReadKey(true);
89              if (key.Key == ConsoleKey.Enter) complete = true;
90          }
91      }
92
93      public IEnumerable<(string, bool)> OptionSelector(string title, IEnumerable<(string, bool)> options, bool clear
    ↪  = true)
94      {
95          List<(string, bool)> result = new List<(string, bool)>(options);
96          result.Add(("EXIT", false));
97
98          while (Console.KeyAvailable) Console.ReadKey(true);
99          _menuInstance.ClearUserSection();
100         _menuInstance.WriteLine(title);
101
102         int j = 3;
103
104         lock (_menuInstance.ScreenLock)
105         {
106             foreach (var option in result)
107             {
108                 Console.SetCursorPosition(1, j++);
109                 if (option.Item2) Console.WriteLine($"  {option.Item1} [{Log.Green}x{Log.Blank}]");
110                 else Console.WriteLine($"  {option.Item1} [ ]");
111             }
112         }
113
114         bool selected = false;
115         int currentTop;
116
```

```
117         lock (_menuInstance.ScreenLock)
118         {
119             Console.SetCursorPosition(1, 3);
120             Console.Write('>');
121 
122             currentTop = Console.CursorTop;
123         }
124 
125         while (!selected)
126         {
127             Console.CursorVisible = false;
128 
129             ConsoleKeyInfo key = Console.ReadKey(true);
130             if (key.Key == ConsoleKey.DownArrow && currentTop < result.Count() + 2)
131             {
132                 lock (_menuInstance.ScreenLock)
133                 {
134                     Console.CursorLeft = 1;
135                     Console.CursorTop = currentTop;
136                     Console.Write(' ');
137                     Console.CursorTop = ++currentTop;
138                     Console.CursorLeft = 1;
139                     Console.Write('>');
140                 }
141             }
142             else if (key.Key == ConsoleKey.UpArrow && currentTop > 3)
143             {
144                 lock (_menuInstance.ScreenLock)
145                 {
146                     Console.CursorLeft = 1;
147                     Console.CursorTop = currentTop;
148                     Console.Write(' ');
149                     Console.CursorTop = --currentTop;
150                     Console.CursorLeft = 1;
151                     Console.Write('>');
152                 }
153             }
154             else if (key.Key == ConsoleKey.Enter || key.Key == ConsoleKey.Spacebar)
155             {
156                 if (result.Count + 2 == currentTop)
157                 {
158                     if (clear) _menuInstance.ClearUserSection();
159                     Console.CursorVisible = false;
160 
161                     selected = true;
162                 }
163                 else
164                 {
165                     result[currentTop - 3] = (result[currentTop - 3].Item1, !result[currentTop - 3].Item2);
166                     Console.SetCursorPosition(1, currentTop);
167                     if (result[currentTop - 3].Item2) Console.WriteLine($"> {result[currentTop - 3].Item1}
    ↪  [{Log.Green}x{Log.Blank}]");
168                     else Console.WriteLine($"> {result[currentTop - 3].Item1} [ ]");
169                 }
170             }
171         }
172 
173         return result;
174     }
175 
```

```
176
177     public string GetInput(string prompt)
178     {
179         while (Console.KeyAvailable) Console.ReadKey(true);
180         _menuInstance.WriteLine(prompt);
181
182         bool complete = false;
183         StringBuilder input = new StringBuilder();
184         int line = _menuInstance.CurrentLine;
185
186         while (!complete)
187         {
188             if (Console.KeyAvailable)
189             {
190                 ConsoleKeyInfo key = Console.ReadKey(true);
191                 switch (key.Key)
192                 {
193                     case ConsoleKey.Enter:
194                         complete = true;
195                         break;
196                     case ConsoleKey.Backspace:
197                     case ConsoleKey.Delete:
198                         {
199                             if (input.Length > 0)
200                             {
201                                 lock (_menuInstance.ScreenLock)
202                                 {
203                                     Console.SetCursorPosition((input.Length % (Console.WindowWidth * 3 / 4 - 1)),
    ↪  line);
204                                     Console.Write(' ');
205                                 }
206
207                                 input.Remove(input.Length - 1, 1);
208                             }
209
210                             break;
211                         }
212                     default:
213                         {
214                             if (input.Length / (line - 1) > Console.WindowWidth * 3 / 4 - 2) line++;
215
216                             lock (_menuInstance.ScreenLock)
217                             {
218                                 Console.SetCursorPosition((input.Length % (Console.WindowWidth * 3 / 4 - 1)) + 1,
    ↪  line);
219                                 Console.Write(key.KeyChar);
220                             }
221
222                             input.Append(key.KeyChar);
223                             break;
224                         }
225                 }
226             }
227         }
228
229         _menuInstance.WriteLine();
230
231         return input.ToString();
232     }
233
```

```
234     public string TryGetInput(string prompt)
235     {
236         string res = GetInput(prompt);
237         return res.Length == 0 ? "None" : res;
238     }
239 
240     public double GetDouble(string prompt) => double.Parse(GetInput(prompt));
241 
242     public bool TryGetDouble(string prompt, out double result) => double.TryParse(GetInput(prompt), out result);
243 
244     public int GetInt(string prompt) => int.Parse(GetInput(prompt));
245 
246     public bool TryGetInt(string prompt, out int result) => int.TryParse(GetInput(prompt), out result);
247 }
```

Log.cs

```
public class Log
{
    private int _logLineCount = 6;
    private readonly Menu _menuInstance;

    public const string Red = "\x1b[38;5;196m";
    public const string Orange = "\x1b[38;5;184m";
    public const string Purple = "\x1b[38;5;129m";
    public const string Green = "\x1b[38;5;2m";
    public const string Blue = "\x1b[38;5;27m";
    public const string Pink = "\x1b[38;5;200m";
    public const string Grey = "\x1b[38;5;243m";
    public const string Blank = "\x1b[0m";

    public void Error(string message) => Logger.WriteLineToMaster($"ERROR {message}");
    public void Warn(string message) => Logger.WriteLineToMaster($"WARNING {message}");
    public void Event(string message) => Logger.WriteLineToMaster($"EVENT {message}");
    public void End(string message) => Logger.WriteLineToMaster($"END {message}");

    public void Error(Guid runGuid, string message, bool detailed = false) => LogParent(runGuid, message, 0,
↪   detailed);
    public void Warn(Guid runGuid, string message, bool detailed = false) => LogParent(runGuid, message, 1,
↪   detailed);
    public void Event(Guid runGuid, string message, bool detailed = false) => LogParent(runGuid, message, 2,
↪   detailed);
    public void End(Guid runGuid, string message, bool detailed = false) => LogParent(runGuid, message, 3,
↪   detailed);

    public void EndError(Guid runGuid, Exception ex)
    {
        Error($"Run ({runGuid}) terminated due to an error.");
        Error($"Exception: {ex.Message}");
        if (ex.InnerException != null) Error($"Inner Exception: {ex.InnerException.Message}");
        Error(runGuid, ex.Message);
        End(runGuid, $"Run ({runGuid}) terminated.", true);
    }

    public void EndSuccessRun(Guid runGuid)
    {
        End(runGuid, "Successfully completed processing and pathfinding of new image!", true);
        Warn(runGuid, $"Run Guid {runGuid} Deleted. See {Environment.CurrentDirectory}\\saves\\ for output(s) and
↪   {Environment.CurrentDirectory}\\runs\\{runGuid.ToString("N").ToUpper()} for temp images.", true);
        End($"Completed run {runGuid} successfully.");
    }

    public void EndSuccessSave(Guid runGuid)
    {
        End(runGuid, "Successfully completed recall and pathfinding of save file!", true);
        Warn(runGuid, $"Run Guid {runGuid} Deleted. See {Environment.CurrentDirectory}\\saves\\ for output(s). Or
↪   just go to where the save file was located.", true);
        End($"Completed run {runGuid} successfully.");
    }

    public Log(Menu menuInstance)
    {
        _menuInstance = menuInstance;
        _ = new Logger(true);
    }

```

```
54      // 0 - Error, 1 - Warning, 2 - Event, 3 - End
55      private void LogParent(Guid runGuid, string message, int type, bool detailed)
56      {
57          if (!bool.Parse(Settings.UserSettings["detailedLogging"].Item1) && !detailed) return;
58
59          Console.CursorVisible = false;
60          string[] prefix = { $"{Red}ERROR{Log.Blank}", $"{Orange}WARN{Log.Blank}", $"{Green}EVENT{Log.Blank}",
    ↪  $"{Purple}END{Log.Blank}" };
61          string[] filePrefix = { "[ERROR] ", "[WARN] ", "[EVENT] ", "[END] " };
62
63          lock (_menuInstance.ScreenLock)
64          {
65              CheckLogLineCount();
66
67              if (message.Length > Console.WindowWidth / 4 - 7)
68              {
69                  Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 2, _logLineCount++);
70                  int i = 10;
71
72                  Console.Write($"{prefix[type]}: ");
73
74                  foreach (char letter in message)
75                  {
76                      Console.Write(letter);
77                      i++;
78                      if (i > Console.WindowWidth / 4)
79                      {
80                          if (CheckLogLineCount()) return;
81                          Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 9, _logLineCount++);
82                          i = 10;
83                      }
84                  }
85              }
86              else
87              {
88                  Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 2, _logLineCount++);
89                  Console.Write($"{prefix[type]}: {message}");
90              }
91          }
92
93          Logger.WriteLineToRunFile(runGuid, $"{filePrefix[type]}{message}");
94      }
95
96      // Make sure that the total log lines does not exceed the space given
97      private bool CheckLogLineCount()
98      {
99          if (_logLineCount >= Console.WindowHeight)
100         {
101             _logLineCount = 6;
102             _menuInstance.ClearLogSection();
103
104             return true;
105         }
106
107         return false;
108     }
109 }
```

Menu.cs

```
public class Menu
{
    public object ScreenLock { get; } = new object();
    public int CurrentLine { get; private set; } = 1;

    [DllImport("kernel32.dll", SetLastError = true)]
    private static extern bool SetConsoleMode(IntPtr hConsoleHandle, int mode);
    [DllImport("kernel32.dll", SetLastError = true)]
    private static extern bool GetConsoleMode(IntPtr handle, out int mode);
    [DllImport("kernel32.dll", SetLastError = true)]
    private static extern IntPtr GetStdHandle(int handle);

    public bool IsWindowMax() => Console.WindowHeight >= Console.LargestWindowHeight && Console.WindowWidth >=
↪   Console.LargestWindowWidth - 3;

    private readonly string _permLineA;
    private readonly string _permLineB;

    public const char VerticalChar = '';
    public const char HorizontalChar = '';

    public Menu(string permLineA, string permLineB)
    {
        IntPtr handle = GetStdHandle(-11);
        GetConsoleMode(handle, out var mode);
        SetConsoleMode(handle, mode | 0x4);

        int width = Console.WindowWidth / 2;
        int height = Console.WindowHeight / 4;
        Console.SetWindowSize(width, height);
        Console.SetBufferSize(width, height);

        _permLineA = permLineA;
        _permLineB = permLineB;

        Console.Clear();
        Console.CursorVisible = false;
    }

    public void Setup()
    {
        while (!IsWindowMax())
        {
            Console.SetCursorPosition(0, 0);
            Console.WriteLine($"{Log.Red}Maximize Window To Continue{Log.Blank}");
            System.Threading.Thread.Sleep(250);
            Console.SetCursorPosition(0, 0);
            Console.WriteLine($"\x1b[48;5;196mMaximize Window To Continue{Log.Blank}");
            System.Threading.Thread.Sleep(250);

        }

        Console.Clear();

        DisplayInfoBox();
        DisplayLogBox();

        Console.SetCursorPosition(0, 0);
        Console.CursorVisible = false;
```

```
59
60          new Task(() => BeginInfoLoop(Stopwatch.StartNew())).Start();
61      }
62
63      private void DisplayInfoBox()
64      {
65          for (int i = 0; i < Console.WindowWidth * 3 / 4; i++)
66          {
67              Console.SetCursorPosition(i, Console.WindowHeight * 5 / 6);
68              Console.Write(HorizontalChar);
69          }
70
71          Console.SetCursorPosition(1, Console.WindowHeight * 5 / 6 + 2);
72          Console.WriteLine("Current Page: ????? ??? ??????");
73          Console.SetCursorPosition(1, Console.WindowHeight * 5 / 6 + 3);
74          Console.WriteLine("Runtime:       ??:??:??");
75
76          Console.SetCursorPosition(1, Console.WindowHeight * 5 / 6 + 8);
77          Console.WriteLine(_permLineA);
78          Console.SetCursorPosition(1, Console.WindowHeight * 5 / 6 + 9);
79          Console.WriteLine(_permLineB);
80      }
81
82      private void DisplayLogBox()
83      {
84          for (int i = 0; i < Console.WindowHeight; i++)
85          {
86              if (i > 5)
87              {
88                  for (int j = Console.WindowWidth * 3 / 4; j < Console.WindowWidth; j++)
89                  {
90                      Console.SetCursorPosition(j, i);
91                      Console.Write(' ');
92                  }
93              }
94
95              Console.SetCursorPosition(Console.WindowWidth * 3 / 4, i);
96              Console.Write(VerticalChar);
97          }
98
99          for (int i = Console.WindowWidth * 3 / 4 + 1; i < Console.WindowWidth; i++)
100         {
101             Console.SetCursorPosition(i, 5);
102             Console.Write(HorizontalChar);
103         }
104
105         Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 5, 1);
106         Console.WriteLine("Program Logs:");
107         Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 5, 3);
108         Console.WriteLine($"\x1b[48;5;196m  {Log.Blank} ERROR            \x1b[48;5;2m  {Log.Blank} EVENT
    ↪  PROCESSED");
109         Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 5, 4);
110         Console.WriteLine($"\x1b[48;5;184m  {Log.Blank} WARNING          \x1b[48;5;129m  {Log.Blank} END OF
    ↪  SEQUENCE");
111     }
112
113     private void BeginInfoLoop(Stopwatch sw)
114     {
115         while (true)
116         {
```

```
117             lock (ScreenLock)
118             {
119                 Console.SetCursorPosition(15, Console.WindowHeight * 5 / 6 + 3);
120                 Console.Write($"{sw.Elapsed.Hours}:{sw.Elapsed.Minutes}:{sw.Elapsed.Seconds}".PadRight(10, ' '));
121                 Console.CursorVisible = false;
122             }
123             System.Threading.Thread.Sleep(1000);
124         }
125     }
126 
127     public void ClearLogSection()
128     {
129         for (int i = 6; i < Console.WindowHeight; i++)
130         {
131             for (int j = Console.WindowWidth * 3 / 4 + 1; j < Console.WindowWidth; j++)
132             {
133                 Console.SetCursorPosition(j, i);
134                 Console.Write(' ');
135             }
136         }
137     }
138 
139     public void ClearUserSection()
140     {
141         CurrentLine = 1;
142         StringBuilder sb = new StringBuilder();
143         for (int i = 0; i < Console.WindowWidth * 3 / 4; i++) sb.Append(' ');
144 
145         string line = sb.ToString();
146 
147         lock (ScreenLock)
148         {
149             for (int i = 0; i < Console.WindowHeight * 5 / 6; i++)
150             {
151                 Console.SetCursorPosition(0, i);
152                 Console.Write(line);
153             }
154         }
155 
156         Console.SetCursorPosition(0, 0);
157     }
158 
159     public void SetPage(string message)
160     {
161         lock (ScreenLock)
162         {
163             Console.CursorVisible = false;
164             Console.SetCursorPosition(15, Console.WindowHeight * 5 / 6 + 2);
165             Console.Write(message.PadRight(Console.WindowWidth * 3 / 4 - 15));
166         }
167 
168         Console.Title = $"Comp Sci NEA | Rubens Pirie | {message}";
169     }
170 
171     public void WriteLine()
172     {
173         if (CurrentLine > Console.WindowHeight * 5 / 6) ClearUserSection();
174         CurrentLine++;
175     }
176 
```

```
177     public void Error(string message)
178     {
179         int widthStart = ((Console.WindowWidth * 3 / 4) / 3) / 2;
180         int heightStart = (Console.WindowHeight * 5 / 6) / 3;
181         for (int i = 0; i < widthStart * 4; i++)
182         {
183             lock (ScreenLock)
184             {
185                 string toPrint = i == 0 || i == widthStart * 4 - 1 ? "+" : HorizontalChar.ToString();
186                 Console.SetCursorPosition(widthStart + i, heightStart);
187                 Console.Write($"{toPrint}");
188                 Console.SetCursorPosition(widthStart + i, heightStart * 2);
189                 Console.Write($"{toPrint}");
190             }
191         }
192 
193         for (int i = heightStart + 1; i < heightStart * 2; i++)
194         {
195             lock (ScreenLock)
196             {
197                 Console.SetCursorPosition(widthStart, i);
198                 Console.Write($"{VerticalChar}");
199                 Console.SetCursorPosition(widthStart + widthStart * 4 - 1, i);
200                 Console.Write($"{VerticalChar}");
201             }
202         }
203 
204         List<List<char>> messages = new List<List<char>>();
205         messages.Add(new List<char>());
206         List<char> messageChars = message.ToCharArray().ToList();
207         messageChars.Reverse();
208 
209         int e = 0;
210         while (messageChars.Count > 0)
211         {
212             if (messages[e].Count < widthStart * 3)
213             {
214                 messages[e].Add(messageChars[messageChars.Count - 1]);
215                 messageChars.RemoveAt(messageChars.Count - 1);
216             }
217             else
218             {
219                 e++;
220                 messages.Add(new List<char>());
221             };
222         }
223 
224         lock (ScreenLock)
225         {
226             Console.SetCursorPosition((widthStart * 3) - 26, heightStart + 2);
227             Console.Write($"{Log.Red}Something went wrong, to see what take a look below.{Log.Blank}");
228             Console.SetCursorPosition((widthStart * 3) - 8, (int)(heightStart * 1.5) - 3);
229             Console.Write("Reason for Error");
230             for (int i = 0; i < messages.Count; i++)
231             {
232                 Console.SetCursorPosition((widthStart * 3) - messages[i].Count / 2, (int)(heightStart * 1.5) - (2 -
    ↪  i));
233                 Console.Write($"{Log.Blue}{string.Join("", messages[i])}{Log.Blank}");
234             }
235             Console.SetCursorPosition((widthStart * 3) - 18, heightStart * 2 - 2);
```

```
236             Console.Write($"{Log.Grey}(Press Enter to Return to Main Menu){Log.Blank}");
237         }
238 
239 
240     }
241 
242     public void WriteLine(string message)
243     {
244         Console.CursorVisible = false;
245 
246         if (message.Length > Console.WindowWidth * 3 / 4)
247         {
248             int maxLength = Console.WindowWidth * 3 / 4;
249 
250             List<string> words = message.Split(' ').ToList();
251             StringBuilder sb = new StringBuilder();
252 
253             foreach (string word in words)
254             {
255                 if ($"{sb} {word}".Length > maxLength)
256                 {
257                     WriteLine(sb.ToString());
258                     sb.Remove(0, sb.Length);
259                 }
260                 else
261                 {
262                     sb.Append($"{word} ");
263                 }
264             }
265 
266             WriteLine(sb.ToString());
267         }
268         else
269         {
270             lock (ScreenLock)
271             {
272                 if (CurrentLine > Console.WindowHeight * 5 / 6) ClearUserSection();
273 
274                 Console.SetCursorPosition(1, CurrentLine++);
275                 Console.Write(message);
276             }
277         }
278     }
279 
280 }
```

ProgressBar.cs

```
public class ProgressBar
{
    private readonly string _progressTitle;
    private double _progressAmount;
    private readonly double _progressInterval;
    private readonly string _progressOutline;
    private string _progressLine;

    private readonly Menu _menuInstance;

    public ProgressBar(string title, int totalSegments, Menu menuInstance)
    {
        _progressInterval = (double)1 / totalSegments;
        _progressAmount = 0;

        StringBuilder bar = new StringBuilder();
        bar.Append('+');
        for (int i = 0; i < (Console.WindowWidth * 3 / 4) - 4; i++) bar.Append(Menu.HorizontalChar);
        bar.Append('+');

        _progressOutline = bar.ToString();
        _progressLine = "";
        _progressTitle = title;
        _menuInstance = menuInstance;
    }

    public void DisplayProgress()
    {
        int middle = Console.WindowHeight * 5 / 12;

        lock (_menuInstance.ScreenLock)
        {
            Console.SetCursorPosition((Console.WindowWidth * 3 / 8) - (_progressTitle.Length / 2), middle - 3);
            Console.Write(_progressTitle);

            Console.SetCursorPosition(1, middle - 1);
            Console.Write(_progressOutline);
            Console.SetCursorPosition(1, middle);
            Console.Write(Menu.VerticalChar);
            Console.SetCursorPosition(Console.WindowWidth * 3 / 4 - 2, middle);
            Console.Write(Menu.VerticalChar);
            Console.SetCursorPosition(1, middle + 1);
            Console.Write(_progressOutline);
        }
    }

    public Action GetIncrementAction() => new Action(IncrementProgress);

    private void IncrementProgress()
    {
        lock (_menuInstance.ScreenLock)
        {
            _progressAmount = _progressAmount + _progressInterval > 1 ? 1 : _progressAmount + _progressInterval;

            int middle = Console.WindowHeight * 5 / 12;
            double possibleLength = (Console.WindowWidth * 3 / 4) - 4;
            possibleLength *= _progressAmount;

            if (_progressLine.Length != (int)possibleLength)
```

```
60              {
61                  StringBuilder sb = new StringBuilder();
62                  for (int i = 0; i < possibleLength; i++) sb.Append(Menu.VerticalChar);
63                  _progressLine = sb.ToString();
64
65                  Console.SetCursorPosition(2, middle);
66                  Console.Write($"{Log.Blue}{_progressLine}{Log.Blank}");
67              }
68          }
69      }
70  }
```

Settings.cs

```
public class Settings
{
    private readonly Menu _menuInstance;
    private readonly Log _loggerInstance;

    private List<string> rawLines;
    public static Dictionary<string, (string, Type)> UserSettings { get; private set; }

    private readonly string[] defaultSettings = {
        "# Manually Edit At Own Risk",
        "# General Settings",
        "detailedLogging=false",
        "forceFormsFront=true",
        "",
        "# Pathfinding Settings",
        "convertToMST=false",
        "pathfindingAlgorithm=AStar",
        "snapToGrid=true",
        "endOnFind=false",
        "",
        "# Save Settings",
        "shortNames=false",
        "zipOnComplete=false",
    };

    public Settings(Menu menu, Log log)
    {
        _menuInstance = menu;
        _loggerInstance = log;
    }

    public void CheckIfExistsOrCreate()
    {
        if (!File.Exists("settings.conf"))
        {
            _loggerInstance.Event("Settings file did not exist. Creating...");
            using (TextWriter tw = File.CreateText("settings.conf"))
            {
                foreach (string line in defaultSettings)
                {
                    tw.WriteLine(line);
                }
            }
        }
    }

    public List<string> ParseSettingsFile()
    {
        List<string> lines = new List<string>();
        using (StreamReader sr = File.OpenText("settings.conf"))
        {
            while (!sr.EndOfStream)
            {
                lines.Add(sr.ReadLine());
            }
        }

        rawLines = lines;

```

```
60          List<string> validLines = new List<string>();
61          for (int i = 0; i < lines.Count; i++)
62          {
63              if (lines[i].Trim() != "" && !lines[i].Trim().StartsWith("#")) validLines.Add(lines[i]);
64          }
65
66          return validLines;
67      }
68
69      private Dictionary<string, (string, Type)> ConvertSettingsToPairs(List<string> parsedLines)
70      {
71          Dictionary<string, (string, Type)> pairs = new Dictionary<string, (string, Type)>();
72          foreach (string item in parsedLines)
73          {
74              string name = item.Split('=')[0].Trim();
75              string value = item.Split('=')[1].Trim();
76              if (bool.TryParse(value, out bool _)) pairs.Add(name, (value, typeof(bool)));
77              else if (int.TryParse(value, out int _)) pairs.Add(name, (value, typeof(int)));
78              else if (double.TryParse(value, out double _)) pairs.Add(name, (value, typeof(double)));
79              else pairs.Add(name, (value, typeof(string)));
80          }
81
82          return pairs;
83      }
84
85      public bool Change(string setting, bool value)
86      {
87          if (!UserSettings.ContainsKey(setting)) return false;
88          UserSettings[setting] = (value.ToString().ToLower(), typeof(bool));
89
90          return true;
91      }
92
93      public bool Change(string setting, int value)
94      {
95          if (!UserSettings.ContainsKey(setting)) return false;
96          UserSettings[setting] = (value.ToString(), typeof(int));
97
98          return true;
99      }
100
101     public bool Change(string setting, double value)
102     {
103         if (!UserSettings.ContainsKey(setting)) return false;
104         UserSettings[setting] = (value.ToString(), typeof(double));
105
106         return true;
107     }
108
109     public bool Change(string setting, string value)
110     {
111         if (!UserSettings.ContainsKey(setting)) return false;
112         UserSettings[setting] = (value.ToString(), typeof(string));
113
114         return true;
115     }
116
117     public void Read()
118     {
119         CheckIfExistsOrCreate();
```

```
120         List<string> parsedLines = ParseSettingsFile();
121         Dictionary<string, (string, Type)> settingValuePairs = ConvertSettingsToPairs(parsedLines);
122         UserSettings = settingValuePairs;
123     }
124 
125     public void Update(Dictionary<string, (string, Type)> oldSettings, Dictionary<string, (string, Type)>
    ↪ newSettings)
126     {
127         if (oldSettings.Count != newSettings.Count) throw new SettingsException("Cannot set settings when the
    ↪ amount of settings has changed, if this problem persists delete settings.conf and restart the program.");
128 
129         foreach (KeyValuePair<string, (string, Type)> pair in newSettings)
130         {
131             int location = rawLines.FindIndex(toCheck => toCheck.Contains(pair.Key));
132             if (location == -1) throw new SettingsException($"You have an unknown setting {pair.Key}, if this
    ↪ problem persists delete settings.conf and restart the program.");
133             else
134             {
135                 if (!oldSettings.ContainsKey(pair.Key)) throw new SettingsException($"Setting {pair.Key} does not
    ↪ exist, if this problem persists delete settings.conf and restart the program.");
136                 if (!oldSettings[pair.Key].Equals(pair.Value)) rawLines[location] = $"{pair.Key}=
    ↪ {pair.Value.Item1}";
137             }
138         }
139 
140         Write();
141     }
142 
143     private void Write()
144     {
145         using (TextWriter tw = File.CreateText("settings.conf"))
146         {
147             foreach (string line in rawLines)
148             {
149                 tw.WriteLine(line);
150             }
151         }
152     }
153 }
```

TextWall.cs

```
1  public static class TextWall
2  {
3      public static void SaveWelcome(Menu menuInstance)
4      {
5          menuInstance.WriteLine("You have chosen to re-call a map file which has been previously used. At the next
   ↪  prompt you will be asked to enter the file / the path to it. After that you will have several options open to
   ↪  you:");
6          menuInstance.WriteLine();
7          menuInstance.WriteLine("1. You can choose to modify the file parameters, i.e. Name, Description or Type");
8          menuInstance.WriteLine("2. Delete the file");
9          menuInstance.WriteLine("3. Clone the file");
10         menuInstance.WriteLine("4. Rename the file");
11         menuInstance.WriteLine("5. View current file stats");
12         menuInstance.WriteLine("6. Run pathfinding on the image");
13     }
14
15     public static void ImageWelcome(Menu menuInstance)
16     {
17         menuInstance.WriteLine("You have selected to read a new image and turn it into a route-able map, during
   ↪  this the following steps will occur:");
18         menuInstance.WriteLine();
19         menuInstance.WriteLine("1. You will be asked to supply an image to process.");
20         menuInstance.WriteLine("2. The image will be checked to make sure it is valid, if it is not you will have
   ↪  to pick another and start again.");
21         menuInstance.WriteLine("3. You will be shown the image to check if it is the right one, as well as some
   ↪  file details about it. You can chose to end here if you wish.");
22         menuInstance.WriteLine("4. You will have some options as to how to pick out the roads. There are some
   ↪  presets as well as a step by step version.");
23         menuInstance.WriteLine("5. After the roads have been picked out you will be able to click on different
   ↪  points and find the most efficient root through them.");
24         menuInstance.WriteLine("6. You can chose to save that map or not.");
25     }
26
27     public static void FileDetails(Menu menuInstance, Structures.RawImage rawImage)
28     {
29         menuInstance.WriteLine("Your image file information:");
30         menuInstance.WriteLine($"    Name of image:
   ↪  {Log.Green}{Path.GetFileNameWithoutExtension(rawImage.Path)}{Log.Blank}");
31         menuInstance.WriteLine($"    Folder it's contained within:
   ↪  {Log.Green}{(Path.GetDirectoryName(rawImage.Path) == "" ? "/" :
   ↪  Path.GetDirectoryName(rawImage.Path))}{Log.Blank}");
32         menuInstance.WriteLine($"    Type of image: {Log.Green}{Path.GetExtension(rawImage.Path)}{Log.Blank}");
33         menuInstance.WriteLine();
34     }
35 }
36
```

6.2.2.3 Processes

AsyncEdgeDetection.cs

```
public class AsyncEdgeDetection : IHandler
{
    private readonly Menu _menuInstance;
    private readonly Log _logInstance;
    private readonly Guid _runGuid;
    private readonly Structures.RawImage _image;
    private double[,] _resultArray;

    public AsyncEdgeDetection(Menu menu, Log log, Structures.RawImage image, Guid currentGuid)
    {
        _menuInstance = menu;
        _logInstance = log;
        _image = image;
        _runGuid = currentGuid;
    }

    public void Preset(int kernelSize, double redRatio, double greenRatio, double blueRatio, double sigma, double
    ↪ lowerThreshold, double upperThreshold, int loopCount)
    {
        _logInstance.Event(_runGuid, $"Running preset with values ({Log.Orange}{kernelSize}{Log.Blank},
    ↪ {Log.Orange}{redRatio}{Log.Blank}, {Log.Orange}{greenRatio}{Log.Blank}, {Log.Orange}{blueRatio}{Log.Blank},
    ↪ {Log.Orange}{sigma}{Log.Blank}, {Log.Orange}{lowerThreshold}{Log.Blank},
    ↪ {Log.Orange}{upperThreshold}{Log.Blank}, {Log.Orange}{loopCount}{Log.Blank})");

        CannyEdgeDetection detector = new CannyEdgeDetection(kernelSize, redRatio, greenRatio, blueRatio, sigma,
            lowerThreshold, upperThreshold);

        Input inputHandel = new Input(_menuInstance);

        Structures.RGB[][,] quads = Utility.SplitImage(_image.Pixels);
        Task<double[,]>[] threads = new Task<double[,]>[quads.Length];

        int continueOption = inputHandel.GetOption("Continue to Canny Edge Detection:", new[] { "Yes", "No" });
        if (continueOption != 0) throw new Exception("You asked for the processing of your map to stop.");

        bool saveTempOption = inputHandel.GetOption("Would you like to save images at each step of the edge
    ↪ detection?", new[] { "Yes", "No" }) == 0;

        ProgressBar pb = new ProgressBar("Canny Edge Detection", 36, _menuInstance);
        pb.DisplayProgress();

        for (int i = 0; i < quads.Length; i++)
        {
            // Overcome Capture Condition
            int copyI = i;
            Task<double[,]> task = new Task<double[,]>(() => RunDetectionOnQuadrant(detector, quads[copyI], copyI,
    ↪ pb.GetIncrementAction(), saveTempOption));
            task.Start();
            threads[i] = task;
        }

        Task.WaitAll(threads);
        double[,] cannyImage = Utility.CombineQuadrants(threads[0].Result, threads[1].Result, threads[2].Result,
        threads[3].Result);

        Post postProcessor = new Post(cannyImage);
        postProcessor.Start(loopCount);
```

```
52          _resultArray = postProcessor.Result();
53      }
54
55      public void Start()
56      {
57          Input inputHandel = new Input(_menuInstance);
58
59          _logInstance.Event(_runGuid, "Started Multi Threaded Canny Edge Detection");
60          bool confirmOptions = false;
61          CannyEdgeDetection detector;
62
63          _logInstance.Event(_runGuid, "Getting Multi Thread Options");
64
65          do
66          {
67              detector = GetDetector(_menuInstance, inputHandel, _logInstance);
68
69              string opt = inputHandel.GetInput("Are you happy with those edge detection variables (y/n): ");
70              if (opt.ToLower() == "y") confirmOptions = true;
71              else _menuInstance.ClearUserSection();
72          } while (!confirmOptions);
73
74
75          Structures.RGB[][,] quads = Utility.SplitImage(_image.Pixels);
76          Task<double[,]>[] threads = new Task<double[,]>[quads.Length];
77
78          int continueOption = inputHandel.GetOption("Continue to Canny Edge Detection:", new[] { "Yes - Continue",
    ↪  "No - Return to main menu" });
79          if (continueOption != 0) throw new Exception("You asked for the processing of your map to stop.");
80
81          bool saveTempOption = inputHandel.GetOption("Would you like to save images at each step of the edge
    ↪  detection?", new[] { "Yes", "No" }) == 0;
82
83          ProgressBar pb = new ProgressBar("Canny Edge Detection", 36, _menuInstance);
84          pb.DisplayProgress();
85
86          for (int i = 0; i < quads.Length; i++)
87          {
88              // Overcome Capture Condition
89              int copyI = i;
90              Task<double[,]> task = new Task<double[,]>(() => RunDetectionOnQuadrant(detector, quads[copyI], copyI,
    ↪  pb.GetIncrementAction(), saveTempOption));
91              task.Start();
92              threads[i] = task;
93          }
94
95          Task.WaitAll(threads);
96          double[,] cannyImage = Utility.CombineQuadrants(threads[0].Result, threads[1].Result, threads[2].Result,
97              threads[3].Result);
98
99          PostProcessImage(cannyImage, inputHandel);
100     }
101
102     private void PostProcessImage(double[,] image, Input inputHandel)
103     {
104         int timeApproximation = 5;
105         Post postProcessor = new Post(image);
106
107         _menuInstance.ClearUserSection();
```

```
108         if (inputHandel.TryGetInt("How many times would you like to emboss the image (can be 0): ", out int
    ↪ loopCount) &&
109             loopCount > 0)
110         {
111             _menuInstance.WriteLine();
112             _menuInstance.WriteLine($"Running image embossing this will take approximately
    ↪ {Log.Red}{timeApproximation * loopCount}{Log.Blank} seconds!");
113             postProcessor.Start(loopCount);
114         }
115         else
116         {
117             _menuInstance.WriteLine();
118             _menuInstance.WriteLine($"Running image embossing this will take approximately
    ↪ {Log.Red}{timeApproximation}{Log.Blank} seconds!");
119             postProcessor.Start(0);
120         }
121 
122         _resultArray = postProcessor.Result();
123     }
124 
125     private double[,] RunDetectionOnQuadrant(CannyEdgeDetection detector, Structures.RGB[,] image, int id, Action
    ↪ increment, bool saveTemp)
126     {
127         char letter = (char)('A' + id);
128         double[,] workingArray;
129         _logInstance.Event(_runGuid, $"Starting processing of quadrant {letter} ({id % 2}, {id / 2})");
130 
131         workingArray = detector.BlackWhiteFilter(image);
132         if (saveTemp) Logger.SaveBitmap(_runGuid, workingArray, $"BlackWhiteFilterQuad{letter}");
133         increment();
134         _logInstance.Event(_runGuid, $"Completed Black and White Filter on Quadrant {letter}");
135 
136         workingArray = detector.GaussianFilter(workingArray);
137         if (saveTemp) Logger.SaveBitmap(_runGuid, workingArray, $"GaussianFilterQuad{letter}");
138         increment();
139         _logInstance.Event(_runGuid, $"Applied Gaussian Filter on Quadrant {letter}");
140 
141         Structures.Gradients grads = detector.CalculateGradients(workingArray, increment);
142         if (saveTemp)
143         {
144             Logger.SaveBitmap(_runGuid, grads.GradientX, $"GradientXQuad{letter}");
145             Logger.SaveBitmap(_runGuid, grads.GradientY, $"GradientYQuad{letter}");
146         }
147         _logInstance.Event(_runGuid, $"Calculated Gradients for Quadrant {letter}");
148 
149         double[,] combinedGrads = detector.CombineGradients(grads);
150         if (saveTemp) Logger.SaveBitmap(_runGuid, combinedGrads, $"CombinedGradientsQuad{letter}");
151         increment();
152         _logInstance.Event(_runGuid, $"Calculated Combined Gradients for Quadrant {letter}");
153 
154         double[,] angleGrads = detector.GradientAngle(grads);
155         increment();
156 
157         if (saveTemp)
158         {
159             for (int y = 0; y < angleGrads.GetLength(0); y++)
160                 for (int x = 0; x < angleGrads.GetLength(1); x++)
161                     workingArray[y, x] = Utility.MapRadiansToPixel(angleGrads[y, x]);
162 
163             Logger.SaveBitmap(_runGuid, workingArray, $"AngleGradientsQuad{letter}");
```

```
164         }
165         _logInstance.Event(_runGuid, $"Calculated Gradient Angles for Quadrant {letter}");
166 
167         workingArray = detector.MagnitudeThreshold(combinedGrads, angleGrads);
168         if (saveTemp) Logger.SaveBitmap(_runGuid, workingArray, $"MagnitudeThresholdQuad{letter}");
169         increment();
170         _logInstance.Event(_runGuid, $"Applied Magnitude Threshold on Quadrant {letter}");
171 
172         Structures.ThresholdPixel[,] thresholdArray = detector.DoubleThreshold(workingArray);
173         increment();
174         if (saveTemp)
175         {
176             Bitmap toSave = new Bitmap(thresholdArray.GetLength(1), thresholdArray.GetLength(0));
177             for (int y = 0; y < thresholdArray.GetLength(0); y++)
178             {
179                 for (int x = 0; x < thresholdArray.GetLength(1); x++)
180                 {
181                     if (thresholdArray[y, x].Strong) toSave.SetPixel(x, y, Color.Green);
182                     else if (!thresholdArray[y, x].Strong && thresholdArray[y, x].Value != 0) toSave.SetPixel(x, y,
    ↪  Color.Red);
183                     else toSave.SetPixel(x, y, Color.Black);
184                 }
185             }
186             Logger.SaveBitmap(_runGuid, toSave, $"ThresholdPixelsQuad{letter}");
187         };
188 
189         _logInstance.Event(_runGuid, $"Calculated Threshold Pixels for Quadrant {letter}");
190 
191         workingArray = detector.EdgeTrackingHysteresis(thresholdArray);
192         if (saveTemp) Logger.SaveBitmap(_runGuid, workingArray, $"EdgeTrackingHysteresisQuad{letter}");
193         increment();
194         _logInstance.Event(_runGuid, $"Applied Edge Tracking by Hysteresis on Quadrant {letter}");
195 
196         return workingArray;
197     }
198 
199     private CannyEdgeDetection GetDetector(Menu m, Input i, Log l)
200     {
201         CannyEdgeDetection cannyDetection = new CannyEdgeDetection();
202 
203         if (i.TryGetDouble(
204                     $"Enter a value for the ratio value for red for the Black and White filter (Default:
    ↪  {cannyDetection.RedRatio}, Range: 0 <= x <= 1)",
205                 out double newRedRatio) && newRedRatio <= 1 && newRedRatio >= 0 && newRedRatio !=
    ↪  cannyDetection.RedRatio)
206         {
207             l.Warn(_runGuid, $"Changed red ratio {cannyDetection.RedRatio} -> {newRedRatio}");
208             m.WriteLine($"{Log.Green}Changed: {cannyDetection.RedRatio} -> {newRedRatio}{Log.Blank}");
209             cannyDetection.RedRatio = newRedRatio;
210         }
211         else m.WriteLine($"{Log.Orange}Kept Default: {cannyDetection.RedRatio}{Log.Blank}");
212         m.WriteLine();
213 
214         if (i.TryGetDouble(
215                     $"Enter a value for the ratio value for green for the Black and White filter (Default:
    ↪  {cannyDetection.GreenRatio}, Range: 0 <= x <= 1)",
216                 out double newGreenRatio) && newGreenRatio <= 1 && newGreenRatio >= 0 &&
217             newGreenRatio != cannyDetection.GreenRatio)
218         {
219             l.Warn(_runGuid, $"Changed green ratio {cannyDetection.GreenRatio} -> {newGreenRatio}");
```

```
220             m.WriteLine($"{Log.Green}Changed: {cannyDetection.GreenRatio} -> {newGreenRatio}{Log.Blank}");
221             cannyDetection.GreenRatio = newGreenRatio;
222         }
223         else m.WriteLine($"{Log.Orange}Kept Default: {cannyDetection.GreenRatio}{Log.Blank}");
224         m.WriteLine();
225 
226         if (i.TryGetDouble(
227                     $"Enter a value for the ratio value for blue for the Black and White filter (Default:
    ↪ {cannyDetection.BlueRatio}, Range: 0 <= x <= 1)",
228                 out double newBlueRatio) && newBlueRatio <= 1 && newBlueRatio >= 0 && newBlueRatio !=
    ↪ cannyDetection.BlueRatio)
229         {
230             l.Warn(_runGuid, $"Changed blue ratio {cannyDetection.BlueRatio} -> {newBlueRatio}");
231             m.WriteLine($"{Log.Green}Changed: {cannyDetection.BlueRatio} -> {newBlueRatio}{Log.Blank}");
232             cannyDetection.BlueRatio = newBlueRatio;
233         }
234         else m.WriteLine($"{Log.Orange}Kept Default: {cannyDetection.BlueRatio}{Log.Blank}");
235         m.WriteLine();
236 
237         if (i.TryGetDouble(
238                     $"Enter a value for sigma for the Gaussian Filter stage (Default: {cannyDetection.Sigma},
    ↪ Range: 0 < x <= 10)",
239                 out double newSigma) && newSigma <= 10 && newSigma > 0 && newSigma != cannyDetection.Sigma)
240         {
241             l.Warn(_runGuid, $"Changed sigma value {cannyDetection.Sigma} -> {newSigma}");
242             m.WriteLine($"{Log.Green}Changed: {cannyDetection.Sigma} -> {newSigma}{Log.Blank}");
243             cannyDetection.Sigma = newSigma;
244         }
245         else m.WriteLine($"{Log.Orange}Kept Default: {cannyDetection.Sigma}{Log.Blank}");
246         m.WriteLine();
247 
248         if (i.TryGetInt(
249                     $"Enter a value for kernel size for the Gaussian Filter stage, large values will take exponentially
    ↪ longer (Default: {cannyDetection.KernelSize}, Range: x >= 3, x not a multiple of 2 and a whole number)",
250                 out int newKernel) && newKernel >= 3 && newKernel % 2 == 1 && newKernel % 1 == 0 && newKernel !=
    ↪ cannyDetection.KernelSize)
251         {
252             l.Warn(_runGuid, $"Changed kernel size {cannyDetection.KernelSize} -> {newKernel}");
253             m.WriteLine($"{Log.Green}Changed: {cannyDetection.KernelSize} -> {newKernel}{Log.Blank}");
254             cannyDetection.KernelSize = newKernel;
255         }
256         else m.WriteLine($"{Log.Orange}Kept Default: {cannyDetection.KernelSize}{Log.Blank}");
257         m.WriteLine();
258 
259         if (i.TryGetDouble(
260                     $"Enter a value for the lower threshold for the Min Max stage (Default:
    ↪ {cannyDetection.LowerThreshold}, Range: 0 <= x < 1)",
261                 out double newLowerThreshold) && newLowerThreshold > 0 && newLowerThreshold < 1 &&
    ↪ newLowerThreshold != cannyDetection.LowerThreshold)
262         {
263             l.Warn(_runGuid, $"Changed lower threshold {cannyDetection.LowerThreshold} -> {newLowerThreshold}");
264             m.WriteLine($"{Log.Green}Changed: {cannyDetection.LowerThreshold} -> {newLowerThreshold}{Log.Blank}");
265             cannyDetection.LowerThreshold = newLowerThreshold;
266         }
267         else m.WriteLine($"{Log.Orange}Kept Default: {cannyDetection.LowerThreshold}{Log.Blank}");
268         m.WriteLine();
269 
270         if (i.TryGetDouble(
271                     $"Enter a value for the lower threshold for the Min Max stage (Default:
    ↪ {cannyDetection.UpperThreshold}, Range: {cannyDetection.LowerThreshold} < x <= 1)",
```

```
272                 out double newHigherThreshold) && newHigherThreshold > cannyDetection.LowerThreshold &&
    ↪ newHigherThreshold <= 1 && newHigherThreshold != cannyDetection.UpperThreshold)
273         {
274             l.Warn(_runGuid, $"Changed upper threshold {cannyDetection.UpperThreshold} -> {newHigherThreshold}");
275             m.WriteLine($"{Log.Green}Changed: {cannyDetection.UpperThreshold} -> {newHigherThreshold}{Log.Blank}");
276             cannyDetection.UpperThreshold = newHigherThreshold;
277         }
278         else m.WriteLine($"{Log.Orange}Kept Default: {cannyDetection.UpperThreshold}{Log.Blank}");
279         m.WriteLine();
280 
281         i.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
282         m.ClearUserSection();
283 
284         m.WriteLine("For reference the variables which will be used are:");
285         m.WriteLine($"    Red Ratio: {Log.Green}{cannyDetection.RedRatio}{Log.Blank}");
286         m.WriteLine($"    Green Ratio: {Log.Green}{cannyDetection.GreenRatio}{Log.Blank}");
287         m.WriteLine($"    Blue Ratio: {Log.Green}{cannyDetection.BlueRatio}{Log.Blank}");
288         m.WriteLine($"    Gaussian Sigma Value: {Log.Green}{cannyDetection.Sigma}{Log.Blank}");
289         m.WriteLine($"    Gaussian Kernel Size: {Log.Green}{cannyDetection.KernelSize}{Log.Blank}");
290         m.WriteLine($"    Double Threshold Lower: {Log.Green}{cannyDetection.LowerThreshold}{Log.Blank}");
291         m.WriteLine($"    Double Threshold Upper: {Log.Green}{cannyDetection.UpperThreshold}{Log.Blank}");
292         m.WriteLine();
293 
294         return cannyDetection;
295     }
296 
297     public double[,] Result() => _resultArray;
298 }
```

Pathfinder.cs

```
public class Pathfinder
{
    private readonly double[,] _input;
    private readonly Bitmap _originalBitmap;

    private Graph<Structures.Coord> _graph;
    private Traversal<Structures.Coord> _traversal;

    public Pathfinder(Bitmap originalImage, double[,] input)
    {
        _originalBitmap = originalImage;
        _input = input;
    }

    public void Start()
    {
        InstanceClasses();

        PathfindImageForm pathfindForm = new PathfindImageForm(_originalBitmap, _traversal, _graph);
        pathfindForm.ShowDialog();
    }

    private void InstanceClasses()
    {
        _graph = _input.ToGraph();
        _traversal = new Traversal<Structures.Coord>(_graph);
    }
}
```

RoadSquence.cs

```
internal class RoadSequence
{
    private readonly Menu _menuInstance;
    private readonly Log _logInstance;
    private readonly Guid _runGuid;
    private double[,] _cannyEdgeDetectionResult;
    private readonly MapFile _saveFile;
    private Structures.RoadResult _roadResult;

    public RoadSequence(Menu menuInstance, Log logInstance, Guid currentGuid, double[,] cannyResult, MapFile
↪   saveFile)
    {
        _menuInstance = menuInstance;
        _logInstance = logInstance;
        _runGuid = currentGuid;
        _cannyEdgeDetectionResult = cannyResult;
        _saveFile = saveFile;
        _roadResult = new Structures.RoadResult();
    }

    public Structures.RoadResult Result() => _roadResult;

    public void Start()
    {
        Input inputHandel = new Input(_menuInstance);

        InvertImage(inputHandel);

        DetectRoads(inputHandel);

        if (_saveFile != null)
        {
            _saveFile.PathImage = new Bitmap(_roadResult.PathBitmap);
            _saveFile.CombinedImage = Utility.CombineBitmap(_saveFile.OriginalImage, _roadResult.PathBitmap);
            string path = _saveFile.Save(_runGuid);

            string saveName = _runGuid.ToString();

            if (bool.Parse(Settings.UserSettings["shortNames"]
                    .Item1))
            {

                saveName = _saveFile.Name.Replace(' ', '_');
                File.Move(path,
                    path.Replace(Path.GetFileName(path)
                            .Split('.')[0],
                        saveName));
            }

            if (bool.Parse(Settings.UserSettings["zipOnComplete"]
                    .Item1))
            {
                Directory.CreateDirectory("temp");
                Directory.CreateDirectory("temp/images");

                string[] files = Directory.GetFiles($"./runs/{_runGuid.ToString("N").ToUpper()}", "*.*",
↪   SearchOption.AllDirectories);
                foreach (string newPath in files)
                {
```

```
58                      File.Copy(newPath, newPath.Replace($"./runs/{_runGuid.ToString("N").ToUpper()}",
    ↪ "temp/images"));
59                  }
60
61                  File.Copy($"./logs/{_runGuid}.txt", "temp/log.txt");
62                  File.Copy($"./saves/{saveName}.vmap", "temp/map.vmap");
63                  ZipFile.CreateFromDirectory("temp", $"run-{_runGuid}.zip");
64                  Directory.Delete("temp", true);
65              }
66          }
67      }
68
69      private void InvertImage(Input inputHandel)
70      {
71          bool invert = Utility.IsYes(inputHandel.GetInput("Invert image (y/n)?"));
72          if (invert)
73          {
74              _cannyEdgeDetectionResult = Utility.InverseImage(_cannyEdgeDetectionResult);
75              ViewImageForm invertImageForm = new ViewImageForm(_cannyEdgeDetectionResult.ToBitmap());
76              invertImageForm.ShowDialog();
77              if (_saveFile != null) _saveFile.IsInverted = true;
78          }
79          if (_saveFile != null) _saveFile.IsInverted = false;
80
81          _menuInstance.WriteLine();
82      }
83
84      private void DetectRoads(Input inputHandel)
85      {
86          bool happy = true;
87
88          double threshold = 0.3;
89
90          while (happy)
91          {
92              if (inputHandel.TryGetDouble(
93                      $"Value for Threshold (Default: {threshold}, Range: 0 <= x < 1)",
94                      out double newThreshold) && newThreshold > 0 && newThreshold < 1 && newThreshold != threshold)
95              {
96                  _logInstance.Warn(_runGuid, $"Changed threshold {threshold} -> {newThreshold}");
97                  _menuInstance.WriteLine($"{Log.Green}Changed: {threshold} -> {newThreshold}{Log.Blank}");
98                  threshold = newThreshold;
99              }
100             else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {threshold}{Log.Blank}");
101             _menuInstance.WriteLine();
102
103             RoadDetection roadDetector = new RoadDetection(_cannyEdgeDetectionResult, threshold);
104             ProgressBar pb = new ProgressBar("Road Detection", _cannyEdgeDetectionResult.Length / 100 * 3,
    ↪ _menuInstance);
105
106             pb.DisplayProgress();
107             roadDetector.Start(pb.GetIncrementAction());
108
109             _roadResult = roadDetector.Result();
110             ViewImageForm roadForm = new ViewImageForm(_roadResult.PathBitmap);
111             roadForm.ShowDialog();
112
113             _menuInstance.ClearUserSection();
114
```

```
115             if (Utility.IsYes(inputHandel.GetInput("Are you happy with this lower threshold you should see your
    ↪ roads, if you don't try decreasing the threshold if you see too much then increase the threshold. (y/n)?")))
    ↪ happy = false;
116         }
117     }
118 }
```

SyncEdgeDetection.cs

```
internal class SyncEdgeDetection : IHandler
{
    private readonly Menu _menuInstance;
    private readonly Log _logInstance;
    private Input _classInputHandel;
    private readonly Structures.RawImage _image;
    private readonly Guid _runGuid;
    private double[,] _workingArray;
    private double[,] _resultArray;
    private CannyEdgeDetection _detector;

    public SyncEdgeDetection(Menu menu, Log logger, Structures.RawImage image, Guid currentGuid)
    {
        _menuInstance = menu;
        _logInstance = logger;
        _image = image;
        _runGuid = currentGuid;
    }

    public void Start()
    {
        _classInputHandel = new Input(_menuInstance);
        _detector = new CannyEdgeDetection();

        ShowDialog();
        BlackWhiteStep();
        GaussianStep();

        _menuInstance.WriteLine("The next 5 steps don't require any parameters, you will still see the result of
↪  each step however, in the order of:");
        _menuInstance.WriteLine("   1. Gradient in X");
        _menuInstance.WriteLine("   2. Gradient in Y");
        _menuInstance.WriteLine("   3. Combined Gradients");
        _menuInstance.WriteLine("   4. Gradient Directions");
        _menuInstance.WriteLine("   5. Magnitude Threshold");
        _menuInstance.WriteLine();
        _classInputHandel.WaitInput($"{Log.Grey}(Enter to Continue){Log.Blank}");
        _menuInstance.WriteLine("This may take some time to process each step.");

        Structures.Gradients grads = _detector.CalculateGradients(_workingArray, () => { });
        ViewImageForm gradXForm = new ViewImageForm(grads.GradientX.ToBitmap());
        gradXForm.ShowDialog();

        ViewImageForm gradYForm = new ViewImageForm(grads.GradientY.ToBitmap());
        gradYForm.ShowDialog();

        _workingArray = _detector.CombineGradients(grads);
        ViewImageForm combinedGradientForm = new ViewImageForm(_workingArray.ToBitmap());
        combinedGradientForm.ShowDialog();

        double[,] gradientDirections = _detector.GradientAngle(grads);
        double[,] gradCopy = gradientDirections;
        for (int y = 0; y < gradientDirections.GetLength(0); y++)
            for (int x = 0; x < gradientDirections.GetLength(1); x++)
                gradCopy[y, x] = Utility.MapRadiansToPixel(gradientDirections[y, x]);
        ViewImageForm gradientDirectionForm = new ViewImageForm(gradCopy.ToBitmap());
        gradientDirectionForm.ShowDialog();

        _workingArray = _detector.MagnitudeThreshold(_workingArray, gradientDirections);
```

```
59          ViewImageForm magnitudeForm = new ViewImageForm(_workingArray.ToBitmap());
60          magnitudeForm.ShowDialog();
61
62          _menuInstance.ClearUserSection();
63
64          Structures.ThresholdPixel[,] _thresholdPixels = DoubleThresholdStep();
65
66          _menuInstance.WriteLine("From here on out stages are automated, however as before you will see each step
    ↪  after it occurs.");
67          _menuInstance.WriteLine();
68          _classInputHandel.WaitInput($"{Log.Grey}(Enter to Continue){Log.Blank}");
69
70          _workingArray = _detector.EdgeTrackingHysteresis(_thresholdPixels);
71          ViewImageForm edgeTrackingForm = new ViewImageForm(_workingArray.ToBitmap());
72          edgeTrackingForm.ShowDialog();
73
74          PostProcessImage(_workingArray);
75      }
76
77      private void PostProcessImage(double[,] image)
78      {
79          Post postProcessor = new Post(image);
80
81          _menuInstance.ClearUserSection();
82          if (_classInputHandel.TryGetInt("How many times would you like to emboss the image (can be 0): ", out int
    ↪  loopCount) &&
83              loopCount > 0)
84          {
85              _menuInstance.WriteLine();
86              _menuInstance.WriteLine($"Running image embossing this will take approximately {Log.Red}{10 *
    ↪  loopCount}{Log.Blank} seconds!");
87              postProcessor.Start(loopCount);
88          }
89          else
90          {
91              _menuInstance.WriteLine();
92              _menuInstance.WriteLine($"Running image embossing this will take approximately {Log.Red}10{Log.Blank}
    ↪  seconds!");
93              postProcessor.Start(0);
94          }
95
96          _resultArray = postProcessor.Result();
97      }
98
99      private Structures.ThresholdPixel[,] DoubleThresholdStep()
100     {
101         bool happy = false;
102         Structures.ThresholdPixel[,] _workingThresholdPixels = new Structures.ThresholdPixel[0, 0];
103
104         _menuInstance.WriteLine($"The 8th stage of Canny Edge Detection is applying a double threshold. It is made
    ↪  up of two parameters a lower and upper threshold.");
105
106         while (!happy)
107         {
108             if (_classInputHandel.TryGetDouble(
109                     $"Value for Lower Threshold (Default: {_detector.LowerThreshold}, Range: 0 <= x < 1)",
110                     out double newLowerThreshold) && newLowerThreshold > 0 && newLowerThreshold < 1 &&
    ↪  newLowerThreshold != _detector.LowerThreshold)
111             {
```

```
112                 _logInstance.Warn(_runGuid, $"Changed lower threshold {_detector.LowerThreshold} ->
    ↪ {newLowerThreshold}");
113                 _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.LowerThreshold} ->
    ↪ {newLowerThreshold}{Log.Blank}");
114                 _detector.LowerThreshold = newLowerThreshold;
115             }
116             else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.LowerThreshold}{Log.Blank}");
117             _menuInstance.WriteLine();
118 
119             if (_classInputHandel.TryGetDouble(
120                     $"Value for Upper Threshold  (Default: {_detector.UpperThreshold}, Range:
    ↪ {_detector.LowerThreshold} < x <= 1)",
121                     out double newHigherThreshold) && newHigherThreshold > _detector.LowerThreshold &&
    ↪ newHigherThreshold <= 1 && newHigherThreshold != _detector.UpperThreshold)
122             {
123                 _logInstance.Warn(_runGuid, $"Changed upper threshold {_detector.UpperThreshold} ->
    ↪ {newHigherThreshold}");
124                 _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.UpperThreshold} ->
    ↪ {newHigherThreshold}{Log.Blank}");
125                 _detector.UpperThreshold = newHigherThreshold;
126             }
127             else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.UpperThreshold}{Log.Blank}");
128             _menuInstance.WriteLine();
129             _menuInstance.WriteLine();
130             _menuInstance.WriteLine("Applying Double Threshold. This may take some time...");
131 
132             _workingThresholdPixels = _detector.DoubleThreshold(_workingArray);
133             Bitmap toView = new Bitmap(_workingThresholdPixels.GetLength(1), _workingThresholdPixels.GetLength(0));
134             for (int y = 0; y < _workingThresholdPixels.GetLength(0); y++)
135             {
136                 for (int x = 0; x < _workingThresholdPixels.GetLength(1); x++)
137                 {
138                     if (_workingThresholdPixels[y, x].Strong) toView.SetPixel(x, y, Color.Green);
139                     else if (!_workingThresholdPixels[y, x].Strong && _workingThresholdPixels[y, x].Value != 0)
    ↪ toView.SetPixel(x, y, Color.Red);
140                     else toView.SetPixel(x, y, Color.Black);
141                 }
142             }
143             ViewImageForm gaussianForm = new ViewImageForm(toView);
144             _menuInstance.ClearUserSection();
145             gaussianForm.ShowDialog();
146 
147             _menuInstance.WriteLine("Current values for thresholds");
148             _menuInstance.WriteLine($"Lower: {_detector.LowerThreshold}");
149             _menuInstance.WriteLine($"Upper: {_detector.UpperThreshold}");
150             _menuInstance.WriteLine();
151 
152             string opt = _classInputHandel.GetInput("Are you happy with these values for the upper and lower
    ↪ threshold (y/n)?");
153 
154             if (opt.ToLower().StartsWith("y")) happy = true;
155             else
156             {
157                 _menuInstance.ClearUserSection();
158                 _menuInstance.WriteLine($"{Log.Pink}Please re-enter your values.{Log.Blank}");
159             }
160         }
161 
162         _menuInstance.ClearUserSection();
163         return _workingThresholdPixels;
```

```
164     }
165
166     private void GaussianStep()
167     {
168         bool happy = false;
169
170         _menuInstance.WriteLine($"The second stage of Canny Edge Detection is applying a Gaussian filter. It is
    ↪  made up of two parameters sigma and kernel size.");
171
172         while (!happy)
173         {
174             if (_classInputHandel.TryGetDouble(
175                     $"Value for Sigma (Default: {_detector.Sigma}, Range: 0 < x <= 10)",
176                     out double newSigma) && newSigma <= 10 && newSigma > 0 && newSigma != _detector.Sigma)
177             {
178                 _logInstance.Warn(_runGuid, $"Changed Sigma value {_detector.Sigma} -> {newSigma}");
179                 _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.Sigma} -> {newSigma}{Log.Blank}");
180                 _detector.Sigma = newSigma;
181             }
182             else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.Sigma}{Log.Blank}");
183             _menuInstance.WriteLine();
184
185             if (_classInputHandel.TryGetInt(
186                     $"Value for Kernel Size (Default: {_detector.KernelSize}, Range: x >= 3, x not a multiple of 2
    ↪  and a whole number)",
187                     out int newKernel) && newKernel >= 3 && newKernel % 2 == 1 && newKernel % 1 == 0 && newKernel
    ↪  != _detector.KernelSize)
188             {
189                 _logInstance.Warn(_runGuid, $"Changed Kernel Size {_detector.KernelSize} -> {newKernel}");
190                 _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.KernelSize} -> {newKernel}{Log.Blank}");
191                 _detector.KernelSize = newKernel;
192             }
193             else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.KernelSize}{Log.Blank}");
194             _menuInstance.WriteLine();
195             _menuInstance.WriteLine("Applying Gaussian Filter. This may take some time...");
196
197             _workingArray = _detector.GaussianFilter(_workingArray);
198             ViewImageForm gaussianForm = new ViewImageForm(_workingArray.ToBitmap());
199             _menuInstance.ClearUserSection();
200             gaussianForm.ShowDialog();
201
202             _menuInstance.WriteLine("Current values");
203             _menuInstance.WriteLine($"Sigma: {_detector.Sigma}");
204             _menuInstance.WriteLine($"Kernel Size: {_detector.KernelSize}");
205             _menuInstance.WriteLine();
206
207             string opt = _classInputHandel.GetInput("Are you happy with this value of sigma and the result
    ↪  (y/n)?");
208
209             if (opt.ToLower().StartsWith("y")) happy = true;
210             else
211             {
212                 _menuInstance.ClearUserSection();
213                 _menuInstance.WriteLine($"{Log.Pink}Please re-enter your values.{Log.Blank}");
214             }
215         }
216
217         _menuInstance.ClearUserSection();
218     }
219
```

```
220     private void BlackWhiteStep()
221     {
222         bool happy = false;
223 
224         _menuInstance.WriteLine($"The first stage of Canny Edge Detection is the Black and White filter. It is made
    ↪ up of 3 parameters {Log.Red}Red{Log.Blank}, {Log.Green}Green{Log.Blank}, {Log.Blue}Blue{Log.Blank} Ratios.");
225 
226         while (!happy)
227         {
228             if (_classInputHandel.TryGetDouble(
229                     $"Value for {Log.Red}Red{Log.Blank} (Old: {_detector.RedRatio}, Range: 0 <= x <= 1)",
230                     out double newRedRatio) && newRedRatio <= 1 && newRedRatio >= 0 && newRedRatio !=
    ↪ _detector.RedRatio)
231             {
232                 _logInstance.Warn(_runGuid, $"Changed {Log.Red}Red{Log.Blank} ratio {_detector.RedRatio} ->
    ↪ {newRedRatio}");
233                 _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.RedRatio} -> {newRedRatio}{Log.Blank}");
234                 _detector.RedRatio = newRedRatio;
235             }
236             else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.RedRatio}{Log.Blank}");
237             _menuInstance.WriteLine();
238 
239             if (_classInputHandel.TryGetDouble(
240                     $"Value for {Log.Green}Green{Log.Blank} (Old: {_detector.GreenRatio}, Range: 0 <= x <= 1)",
241                     out double newGreenRatio) && newGreenRatio <= 1 && newGreenRatio >= 0 &&
242                 newGreenRatio != _detector.GreenRatio)
243             {
244                 _logInstance.Warn(_runGuid, $"Changed {Log.Green}Green{Log.Blank} ratio {_detector.GreenRatio} ->
    ↪ {newGreenRatio}");
245                 _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.GreenRatio} ->
    ↪ {newGreenRatio}{Log.Blank}");
246                 _detector.GreenRatio = newGreenRatio;
247             }
248             else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.GreenRatio}{Log.Blank}");
249             _menuInstance.WriteLine();
250 
251             if (_classInputHandel.TryGetDouble(
252                     $"Value for {Log.Blue}Blue{Log.Blank} (Old: {_detector.BlueRatio}, Range: 0 <= x <= 1)",
253                     out double newBlueRatio) && newBlueRatio <= 1 && newBlueRatio >= 0 && newBlueRatio !=
    ↪ _detector.BlueRatio)
254             {
255                 _logInstance.Warn(_runGuid, $"Changed {Log.Blue}Blue{Log.Blank} ratio {_detector.BlueRatio} ->
    ↪ {newBlueRatio}");
256                 _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.BlueRatio} -> {newBlueRatio}{Log.Blank}");
257                 _detector.BlueRatio = newBlueRatio;
258             }
259             else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.BlueRatio}{Log.Blank}");
260             _menuInstance.WriteLine();
261             _menuInstance.WriteLine("Converting to black and white. This may take some time...");
262 
263             _workingArray = _detector.BlackWhiteFilter(_image.Pixels);
264             ViewImageForm blackWhiteForm = new ViewImageForm(_workingArray.ToBitmap());
265             _menuInstance.ClearUserSection();
266             blackWhiteForm.ShowDialog();
267 
268             _menuInstance.WriteLine("Current values for ratios");
269             _menuInstance.WriteLine($"Red: {Log.Red}{_detector.RedRatio}{Log.Blank}");
270             _menuInstance.WriteLine($"Green: {Log.Green}{_detector.GreenRatio}{Log.Blank}");
271             _menuInstance.WriteLine($"Blue: {Log.Blue}{_detector.BlueRatio}{Log.Blank}");
272             _menuInstance.WriteLine();
```

```
273
274             string opt = _classInputHandel.GetInput("Are you happy with these values and the result (y/n)?");
275
276             if (opt.ToLower().StartsWith("y")) happy = true;
277             else
278             {
279                 _menuInstance.ClearUserSection();
280                 _menuInstance.WriteLine($"{Log.Pink}Please re-enter your values.{Log.Blank}");
281             }
282         }
283         _menuInstance.ClearUserSection();
284     }
285
286     private void ShowDialog()
287     {
288         _menuInstance.ClearUserSection();
289         _menuInstance.WriteLine("You have selected to run edge detection steps one after another, this means that
    ↪  at the end of every step you will be shown your image and then have the option to continue to the next step or
    ↪  change variables.");
290         _classInputHandel.WaitInput($"{Log.Grey}(Enter to Continue){Log.Blank}");
291         _menuInstance.WriteLine();
292     }
293
294     public double[,] Result() => _resultArray;
295 }
```

6.2.2.4 WindowsForms

PathfindImageForm.cs (Partial)

```
public partial class PathfindImageForm : Form
{
    private static readonly Structures.Coord invalidCord = new Structures.Coord { X = -1, Y = -1 };

    private Bitmap _image;
    private readonly Bitmap _originalImage;
    private int _width;
    private int _height;

    private readonly Graph<Structures.Coord> _graph;

    private readonly Traversal<Structures.Coord> _traversalObject;

    private Structures.Coord prevStartNode;
    private Structures.Coord startNode = invalidCord;
    private Structures.Coord endNode = invalidCord;

    private Dictionary<Structures.Coord, Structures.Coord> _preCalculatedTree;

    public PathfindImageForm(Bitmap image, Traversal<Structures.Coord> traversal, Graph<Structures.Coord> graph)
    {
        _image = image;
        _originalImage = image;
        _traversalObject = traversal;
        _graph = graph;

        InitializeComponent();
    }

    private void ViewImageForm_Load(object sender, EventArgs e)
    {
        // Define size
        _width = Console.WindowWidth * 3 / 4 * 8;
        _height = Console.WindowHeight * 5 / 6 * 16;


        // Styling
        ControlBox = false;
        FormBorderStyle = FormBorderStyle.None;
        Text = "Pathfinding Window";

        // set window to size of user area
        MinimumSize = new Size(_width, _height);
        MaximumSize = new Size(_width, _height);

        // account for window bar
        Location = new Point(0, 25);

        // Always on top
        if (bool.Parse(Settings.UserSettings["forceFormsFront"].Item1)) TopMost = true;

        // set picture frame
        imageBox.Width = _width * 2 / 3 - 12;
        imageBox.Height = _height - 24;
        imageBox.SizeMode = PictureBoxSizeMode.StretchImage;
        imageBox.Image = _image;

```

```
58          // Set Pathfind Button
59          goButton.Width = _width / 3 - 24;
60          goButton.Height = (_height / 4 - 24) / 2;
61          goButton.Left = _width * 2 / 3 + 12;
62          goButton.Top = _height * 3 / 4;
63
64          // Set Exit Button
65          exitButton.Width = _width / 3 - 24;
66          exitButton.Height = (_height / 4 - 24) / 2;
67          exitButton.Left = _width * 2 / 3 + 12;
68          exitButton.Top = (_height * 3 / 4 + ((_height / 4 - 24) / 2)) + 12;
69          //exitButton.Top = _height * 9 / 10 - 12;
70
71          // Set instruction box
72          textBox.Width = _width / 3 - 24;
73          textBox.Height = _height * 3 / 4 - 24;
74          textBox.Left = _width * 2 / 3 + 12;
75
76          // Set running box
77          runningBox.Width = _width / 3 - 24;
78          runningBox.Height = _height * 2 / 4 - 24;
79          runningBox.Left = _width * 2 / 3 + 12;
80          runningBox.Visible = false;
81          SetRunningBox();
82
83          // Set working button
84          workingButton.Width = _width / 3 - 24;
85          workingButton.Height = _height / 2 - 12;
86          workingButton.Left = _width * 2 / 3 + 12;
87          workingButton.Top = _height / 2;
88          workingButton.Visible = false;
89
90          // Set Node Progress
91          nodeBox.Width = _width / 3 - 24;
92          nodeBox.Height = _height / 12;
93          nodeBox.Left = _width * 2 / 3 + 12;
94          nodeBox.Top = _height / 2 - 84;
95          nodeBox.Visible = false;
96      }
97
98      private Structures.Coord ConvertImageBoxToBitmapCord(Point location)
99      {
100         int x = (int)(((double)_image.Width / imageBox.Width) * location.X);
101         int y = (int)(((double)_image.Height / imageBox.Height) * location.Y);
102
103         return new Structures.Coord { X = x, Y = y };
104     }
105
106     private void RedrawImage()
107     {
108         _image = new Bitmap(_originalImage);
109         if (startNode != invalidCord)
110         {
111             if (!_graph.ContainsNode(startNode) && bool.Parse(Settings.UserSettings["snapToGrid"].Item1))
112             {
113                 double value = double.MaxValue;
114                 Structures.Coord smallest = new Structures.Coord { X = int.MaxValue, Y = int.MaxValue };
115                 foreach (Structures.Coord node in _graph.GetAllNodes())
116                 {
```

```
117                     double compare = Math.Sqrt(Math.Pow(startNode.X - node.X, 2) + Math.Pow(startNode.Y - node.Y,
    ↪  2));
118                     if (compare < value && _graph.GetNode(node).Count != 0)
119                     {
120                         smallest = node;
121                         value = compare;
122                     }
123                 }
124 
125                 startNode = smallest;
126             }
127 
128             DrawCross(startNode, Color.Green);
129         }
130 
131         if (endNode != invalidCord)
132         {
133             if (!_graph.ContainsNode(endNode) && bool.Parse(Settings.UserSettings["snapToGrid"].Item1))
134             {
135                 double value = double.MaxValue;
136                 Structures.Coord smallest = new Structures.Coord { X = int.MaxValue, Y = int.MaxValue };
137                 foreach (Structures.Coord node in _graph.GetAllNodes())
138                 {
139                     double compare = Math.Sqrt(Math.Pow(endNode.X - node.X, 2) + Math.Pow(endNode.Y - node.Y, 2));
140                     if (compare < value && _graph.GetNode(node).Count != 0)
141                     {
142                         smallest = node;
143                         value = compare;
144                     }
145                 }
146 
147                 endNode = smallest;
148             }
149             DrawCross(endNode, Color.Red);
150         }
151 
152         imageBox.Image = _image;
153     }
154 
155     private void DrawCross(Structures.Coord center, Color colour)
156     {
157         double xRatio = (double)_image.Width / imageBox.Width;
158         double yRatio = (double)_image.Height / imageBox.Height;
159 
160         for (int x = center.X - (int)(2 * xRatio); x <= center.X + (int)(2 * xRatio); x++)
161         {
162             for (int y = center.Y - (int)(10 * yRatio); y <= center.Y + (int)(10 * yRatio); y++)
163             {
164                 if (y >= 0 && y < _image.Height && x >= 0 && x < _image.Width)
165                 {
166                     _image.SetPixel(x, y, colour);
167                 }
168             }
169         }
170 
171         for (int y = center.Y - (int)(2 * yRatio); y <= center.Y + (int)(2 * yRatio); y++)
172         {
173             for (int x = center.X - (int)(10 * xRatio); x <= center.X + (int)(10 * xRatio); x++)
174             {
175                 if (x >= 0 && x < _image.Width && y >= 0 && y < _image.Height)
```

```
176                 {
177                     _image.SetPixel(x, y, colour);
178                 }
179             }
180         }
181     }
182 
183     private void imageBox_Click(object sender, EventArgs eventArgs)
184     {
185         MouseEventArgs mouseEvent = (MouseEventArgs)eventArgs;
186         Structures.Coord clickCord = ConvertImageBoxToBitmapCord(mouseEvent.Location);
187 
188         if (mouseEvent.Button == MouseButtons.Left) if (startNode != clickCord) startNode = clickCord;
189         if (mouseEvent.Button == MouseButtons.Right) if (endNode != clickCord) endNode = clickCord;
190 
191         RedrawImage();
192     }
193 
194     private void exitButton_Click(object sender, EventArgs e) => Close();
195 
196     private void SetRunningBox()
197     {
198         string snapWarning = string.Empty;
199         if (!bool.Parse(Settings.UserSettings["snapToGrid"].Item1))
200             snapWarning = "(Warning can cause broken routes. To change goto settings -> pathfinding ->
    ↪  snapToGrid)\n";
201 
202         string mstWarning = string.Empty;
203         if (bool.Parse(Settings.UserSettings["convertToMST"].Item1))
204             mstWarning = "(Warning can cause non-optimal routes. To change goto settings -> pathfinding ->
    ↪  convertToMST)\n";
205 
206         string endWarning = string.Empty;
207         if (bool.Parse(Settings.UserSettings["endOnFind"].Item1))
208             endWarning = "(Warning causes longer times from different start nodes. To change goto settings ->
    ↪  pathfinding -> endOnFind)\n";
209 
210 
211         runningBox.Text = "Current Pathfinding Settings\n\n" +
212                           $"\nAlgorithm: {Settings.UserSettings["pathfindingAlgorithm"].Item1}" +
213                           $"\n\nUsing Minimum Spanning Tree: {(Settings.UserSettings["convertToMST"].Item1 ==
    ↪  "true" ? "Yes" : "No")}" +
214                           $"\n{mstWarning}" +
215                           $"\nSnapping to grid: {(Settings.UserSettings["snapToGrid"].Item1 == "true" ? "Yes" :
    ↪  "No")}" +
216                           $"\n{snapWarning}" +
217                           $"\nEnd pathfinding on Finding End (Dijkstra Only):
    ↪  {(Settings.UserSettings["endOnFind"].Item1 == "true" ? "Yes" : "No")}" +
218                           $"\n{endWarning}";
219     }
220 
221     private int GetDistanceBetweenNodes(Structures.Coord start, Structures.Coord goal) =>
    ↪  (int)Utility.GetDistanceBetweenNodes(start, goal);
222 
223     private int nodes;
224 
225     private void UpdateNodes()
226     {
227         nodes++;
```

```
228         nodeBox.Text = $"Progress {(nodes / (double)_graph.GetAllNodes().Length * 100):f2}% complete\nNode {nodes}
    ↪ out of {_graph.GetAllNodes().Length}";
229         if (nodes % 2 == 0) Update();
230     }
231 
232     private void goButton_Click(object sender, EventArgs e)
233     {
234         nodes = 0;
235 
236         workingButton.Visible = true;
237         textBox.Visible = false;
238         runningBox.Visible = true;
239         if (Settings.UserSettings["pathfindingAlgorithm"].Item1.ToLower() == "dijkstra") nodeBox.Visible = true;
240 
241         Update();
242 
243         try { if (startNode != invalidCord && endNode != invalidCord)
244             {
245                 if (Settings.UserSettings["pathfindingAlgorithm"].Item1.ToLower() == "dijkstra")
246                 {
247                     if (prevStartNode != startNode && startNode != endNode ||
248                         bool.Parse(Settings.UserSettings["endOnFind"].Item1) == true)
249                     {
250 
251                         Dictionary<Structures.Coord, Structures.Coord> tree = _traversalObject.Dijkstra(startNode,
252                             endNode, bool.Parse(Settings.UserSettings["endOnFind"].Item1), UpdateNodes);
253                         Structures.Coord[] path = Utility.RebuildPath(tree, endNode);
254                         foreach (Structures.Coord node in path)
255                         {
256                             _image.SetPixel(node.X, node.Y, Color.BlueViolet);
257                             imageBox.Image = _image;
258                         }
259 
260                         _preCalculatedTree = tree;
261                     }
262                     else if (prevStartNode == startNode && startNode != endNode)
263                     {
264                         Structures.Coord[] path = Utility.RebuildPath(_preCalculatedTree, endNode);
265                         foreach (Structures.Coord node in path)
266                         {
267                             _image.SetPixel(node.X, node.Y, Color.BlueViolet);
268                             imageBox.Image = _image;
269 
270                         }
271                     }
272                 }
273                 else if (Settings.UserSettings["pathfindingAlgorithm"].Item1.ToLower() == "astar")
274                 {
275                     Dictionary<Structures.Coord, Structures.Coord> tree =
276                         _traversalObject.AStar(startNode, endNode, GetDistanceBetweenNodes);
277                     Structures.Coord[] path = Utility.RebuildPath(tree, endNode);
278                     foreach (Structures.Coord node in path)
279                     {
280                         _image.SetPixel(node.X, node.Y, Color.BlueViolet);
281                         imageBox.Image = _image;
282                     }
283 
284                     _preCalculatedTree = tree;
285 
286                 }
```

```
287
288                 prevStartNode = startNode;
289             }
290
291             workingButton.Visible = false;
292             textBox.Visible = true;
293             runningBox.Visible = false;
294             nodeBox.Visible = false;
295         } catch (Exception _)
296         {
297             workingButton.Visible = false;
298             textBox.Visible = true;
299             runningBox.Visible = false;
300             nodeBox.Visible = false;
301         }
302     }
303 }
```

ViewImageForm.cs (Partial)

```
public partial class ViewImageForm : Form
{
    private readonly Bitmap _image;
    private int _width;
    private int _height;
    public ViewImageForm(Bitmap image)
    {
        this._image = image;
        InitializeComponent();
    }

    private void ViewImageForm_Load(object sender, System.EventArgs e)
    {
        // Define size
        _width = Console.WindowWidth * 3 / 4 * 8;
        _height = Console.WindowHeight * 5 / 6 * 16;

        // Styling
        ControlBox = false;
        FormBorderStyle = FormBorderStyle.None;
        Text = "Preview Window";

        // set window to size of user area
        MinimumSize = new Size(_width, _height);
        MaximumSize = new Size(_width, _height);

        // account for window bar
        Location = new Point(0, 25);

        // Always on top
        if (bool.Parse(Settings.UserSettings["forceFormsFront"].Item1)) TopMost = true;

        // set picture frame
        imageBox.Width = _width * 2 / 3 - 12;
        imageBox.Height = _height - 24;
        imageBox.SizeMode = PictureBoxSizeMode.StretchImage;
        imageBox.Image = _image;

        nextButton.Width = _width / 3 - 24;
        nextButton.Height = _height - 24;
        nextButton.Left = _width * 2 / 3 + 12;
    }

    private void nextButton_Click(object sender, System.EventArgs e)
    {
        Close();
    }
}
```

6.2.2.5 Root

Program.cs

```
public class Program
{
    private static void Main()
    {
        Menu menu = new Menu("Author: Rubens Pirie", $"{Log.Grey}Production Mode{Log.Blank}");
        Log logger = new Log(menu);

        Settings settings = new Settings(menu, logger);
        settings.Read();

        menu.Setup();
        logger.Event("Program has started and menu has been created successfully.");

        Run(menu, logger, settings);
    }

    private static void Run(Menu menuInstance, Log CLILoggingInstance, Settings settingsInstance)
    {
        Input inputHandel = new Input(menuInstance);

        bool running = true;

        while (running)
        {
            menuInstance.SetPage("Welcome Menu");
            int opt = inputHandel.GetOption("Please select an option to continue:",
                new[]
                {
                    "Process New Image Into Map Data File", "Recall Map From Data File", "Settings", "Exit Program"
                });

            switch (opt)
            {
                // New
                case 0:
                    menuInstance.SetPage("Process New Image");
                    TextWall.ImageWelcome(menuInstance);
                    inputHandel.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
                    menuInstance.WriteLine();

                    RunNewImage(menuInstance, CLILoggingInstance);
                    break;
                // Recall
                case 1:
                    menuInstance.SetPage("Recall Old Image");
                    TextWall.SaveWelcome(menuInstance);
                    inputHandel.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
                    menuInstance.WriteLine();

                    RunSaveFile(menuInstance, CLILoggingInstance);
                    break;
                // Settings
                case 2:
                    try
                    {
                        SettingsControl settingsControl = new SettingsControl(settingsInstance, menuInstance,
↪   CLILoggingInstance);
```

```
57                          settingsControl.Start();
58                      }
59                      catch (Exception ex)
60                      {
61                          menuInstance.ClearUserSection();
62                          menuInstance.Error("Your settings file is corrupt, please delete the 'settings.conf' file
    ↪  and restart. The program will now exit.");
63                          new Input(menuInstance).WaitInput("");
64                          Environment.Exit(0);
65                      }
66
67                      menuInstance.ClearUserSection();
68                      break;
69                  // Exit
70                  case 3:
71                      menuInstance.SetPage("Exit");
72                      running = false;
73                      break;
74              }
75          }
76      }
77
78      private static void RunSaveFile(Menu menu, Log logger)
79      {
80          Input inputHandel = new Input(menu);
81          Guid runGuid = Logger.CreateRun();
82
83          menu.ClearUserSection();
84          logger.Event(runGuid, $"Beginning recall of map file (Run Id: {runGuid})");
85
86          SaveFile saveFile = new SaveFile(menu, logger, runGuid);
87
88          try
89          {
90              MapFile recalledMap = saveFile.Read();
91
92              bool running = true;
93
94              while (running)
95              {
96                  menu.SetPage("Recalled Options");
97
98                  int opt = inputHandel.GetOption("What would you like to do with your recalled map?",
99                      new[]
100                     {
101                         "View File / Map Information",
102                         "Change File Information",
103                         "Clone File",
104                         "Rename File",
105                         "Delete File",
106                         "Pathfind Through Image",
107                         "Exit"
108                     });
109
110                 switch (opt)
111                 {
112                     case 0:
113                         menu.SetPage("Image Details");
114                         string[] items = { "Screenshot", "Hand Drawn", "Photograph", "Other" };
115                         menu.ClearUserSection();
```

```
116                         menu.WriteLine("Your current save file information:");
117                         menu.WriteLine($"Name: {recalledMap.Name}");
118                         menu.WriteLine($"Description: {recalledMap.Description}");
119                         menu.WriteLine();
120                         menu.WriteLine($"Type of image: {Log.Orange}{items[recalledMap.Type]}{Log.Blank}");
121                         menu.WriteLine($"Was it inverted: {Log.Purple}{(recalledMap.IsInverted ? "Yes" :
    ↪  "No")}{Log.Blank}");
122                         menu.WriteLine($"Time Created: {Log.Green}{recalledMap.TimeCreated}{Log.Grey}");
123                         inputHandel.WaitInput($"{Log.Grey}(Enter to Continue){Log.Blank}");
124                         break;
125                     case 1:
126                         menu.SetPage("Change Image Details");
127                         int option = inputHandel.GetOption("What part of the tile information do you wish to
    ↪  change:",
128                             new[] { "1. Name", "2. Description", "3. Type of image" });
129                         logger.Event(runGuid, $"Changing file settings, see current run save folder for the save
    ↪  file.");
130                         switch (option)
131                         {
132                             case 0:
133                                 string newName =
134                                     inputHandel.GetInput("What do you want to change the title of the save to?");
135                                 recalledMap.Name = newName;
136                                 break;
137                             case 1:
138                                 string newDescription =
139                                     inputHandel.GetInput("What do you want to change the title of the save to?");
140                                 recalledMap.Description = newDescription;
141                                 break;
142                             case 2:
143                                 recalledMap.Type = inputHandel.GetOption("What type of image is this save?",
144                                     new[] { "Screenshot", "Hand Drawn", "Photograph", "Other" });
145                                 break;
146                         }
147
148                         string path = recalledMap.Save(runGuid);
149                         if (bool.Parse(Settings.UserSettings["shortNames"]
150                                 .Item1))
151                             File.Move(path,
152                                 path.Replace(Path.GetFileName(path)
153                                         .Split('.')[0],
154                                     recalledMap.Name));
155                         break;
156                     case 2:
157                         menu.SetPage("Clone Image");
158                         File.Copy(recalledMap._filePath,
    ↪  recalledMap._filePath.Replace(Path.GetFileName(recalledMap._filePath).Split('.')[0],
    ↪  Path.GetFileName(recalledMap._filePath).Split('.')[0] + "-CLONE"));
159                         logger.Event($"Cloned {recalledMap._filePath}.");
160                         break;
161                     case 3:
162                         menu.SetPage("Rename Image");
163                         string name = inputHandel.GetInput("What would you like to rename the file too?");
164                         logger.Event(runGuid, $"Renamed {Path.GetFileName(recalledMap._filePath).Split('.')[0]} to
    ↪  {name}.");
165                         File.Move(recalledMap._filePath,
    ↪  recalledMap._filePath.Replace(Path.GetFileName(recalledMap._filePath).Split('.')[0], name));
166                         break;
167                     case 4:
168                         menu.SetPage($"{Log.Red}DANGER: Delete Image{Log.Blank}");
```

```
169                         if (inputHandel.GetOption("Are you sure you want to delete the save?",
170                                 new[] { $"{Log.Red}No{Log.Blank}", $"{Log.Red}No{Log.Blank}",
    ↪ $"{Log.Green}Yes{Log.Blank}", $"{Log.Red}No{Log.Blank}", $"{Log.Red}No{Log.Blank}" }) == 2)
171                         {
172                             logger.Warn(runGuid, $"Save file at path {recalledMap._filePath} deleted.");
173                             File.Delete(recalledMap._filePath);
174                             running = false;
175                         }
176                         break;
177                     case 5:
178                         menu.SetPage("Pathfinding Window");
179                         logger.Event(runGuid, $"Starting pathfinding of recalled image.");
180                         double[,] doubles = recalledMap.PathImage.ToDoubles(Utility.GetIfExists);
181                         new Pathfinder(recalledMap.OriginalImage, doubles).Start();
182                         break;
183                     default:
184                         running = false;
185                         break;
186                 }
187             }
188 
189             logger.EndSuccessSave(runGuid);
190         }
191         catch (Exception ex)
192         {
193             menu.ClearUserSection();
194             menu.Error(ex.InnerException != null ? ex.InnerException.Message : ex.Message);
195             new Input(menu).WaitInput("");
196             logger.EndError(runGuid, ex);
197         }
198     }
199 
200     private static void RunNewImage(Menu menu, Log logger)
201     {
202         Input i = new Input(menu);
203 
204         Guid runGuid = Logger.CreateRun();
205         menu.ClearUserSection();
206 
207         logger.Event(runGuid, $"Begin processing of new image (Run Id: {runGuid}).");
208 
209         NewImage newImage = new NewImage(menu, logger, runGuid);
210 
211         try
212         {
213             Structures.RawImage rawImage = newImage.Read();
214 
215             menu.WriteLine();
216             menu.WriteLine("Successfully managed to read in your image, please look carefully at the next popup and
    ↪ make sure it is your image.");
217             i.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
218             menu.WriteLine();
219 
220             logger.Event(runGuid, $"Confirming is correct file.");
221             ViewImageForm beforeForm = new ViewImageForm(rawImage.Pixels.ToBitmap());
222             beforeForm.ShowDialog();
223             menu.ClearUserSection();
224 
225             TextWall.FileDetails(menu, rawImage);
226             menu.WriteLine();
```

```
227
228             bool correctImage = Utility.IsYes(i.GetInput("Is this the correct image (y/n)?"));
229             if (!correctImage) throw new Exception("You asked for the processing of your map to stop.");
230
231             int opt = i.GetOption("Select a version of edge detection to run:", new[] {
232                     "Preset - Hand Drawn Map",
233                     "Preset - Screenshot",
234                     "Preset - Photograph",
235                     "Multi-threaded - Fast, all options decided at the start which allows for faster processing.",
236                     "Synchronous - Slow, options can be changed after each step and steps can be repeated." });
237
238             menu.SetPage("Edge Detection");
239             double[,] resultOfEdgeDetection = null;
240
241             IHandler handler = opt <= 3
242                         ? new AsyncEdgeDetection(menu,
243                             logger,
244                             rawImage,
245                             runGuid)
246                         : (IHandler)new SyncEdgeDetection(menu,
247                             logger,
248                             rawImage,
249                             runGuid);
250
251             switch (opt)
252             {
253                 case 0:
254                     AsyncEdgeDetection handPreset = new AsyncEdgeDetection(menu,
255                         logger,
256                         rawImage,
257                         runGuid);
258                     handPreset.Preset(5, 0.299, 0.587, 0.114, 2, 0.07, 0.25, 2);
259                     handler = handPreset;
260
261                     break;
262                 case 1:
263                     AsyncEdgeDetection screenPreset = new AsyncEdgeDetection(menu,
264                         logger,
265                         rawImage,
266                         runGuid);
267                     screenPreset.Preset(5, 0.299, 0.587, 0.114, 1.4, 0.05, 0.15, 0);
268                     handler = screenPreset;
269                     break;
270                 case 2:
271                     AsyncEdgeDetection photoPreset = new AsyncEdgeDetection(menu,
272                         logger,
273                         rawImage,
274                         runGuid);
275                     photoPreset.Preset(7, 0.299, 0.587, 0.114, 2, 0.1, 0.3, 1);
276                     handler = photoPreset;
277                     break;
278                 default:
279                     handler.Start();
280                     break;
281             }
282
283             resultOfEdgeDetection = handler.Result();
284
285             menu.ClearUserSection();
```

```
286             menu.WriteLine("In order for the road detection to function properly there must be a outline
    ↪  encapsulating the road. It should look like an outline of the road, if there isn't one, and there is just a big
    ↪  white blob then select invert at the next prompt.");
287             menu.WriteLine();
288             i.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
289             menu.WriteLine();
290 
291             ViewImageForm edgeImageForm = new ViewImageForm(resultOfEdgeDetection.ToBitmap());
292             edgeImageForm.ShowDialog();
293 
294             MapFile saveMapFile = rawImage.MapFile;
295 
296             menu.SetPage("Road Detection");
297             RoadSequence roadDetector = new RoadSequence(menu, logger, runGuid, resultOfEdgeDetection,
    ↪  saveMapFile);
298             roadDetector.Start();
299 
300             menu.SetPage("Pathfinding Window");
301             new Pathfinder(rawImage.Original, roadDetector.Result().PathDoubles).Start();
302 
303             logger.EndSuccessRun(runGuid);
304         }
305         catch (Exception ex)
306         {
307             menu.ClearUserSection();
308             menu.Error(ex.InnerException != null ? ex.InnerException.Message : ex.Message);
309             new Input(menu).WaitInput("");
310             logger.EndError(runGuid, ex);
311         }
312     }
313 }
```