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

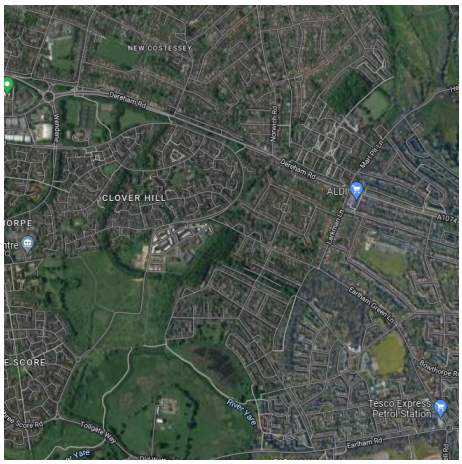
Contents

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.

(a) Map without labels on roads



(b) 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. Therefor 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, **NEEDS TO BE ADDED TO**"

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

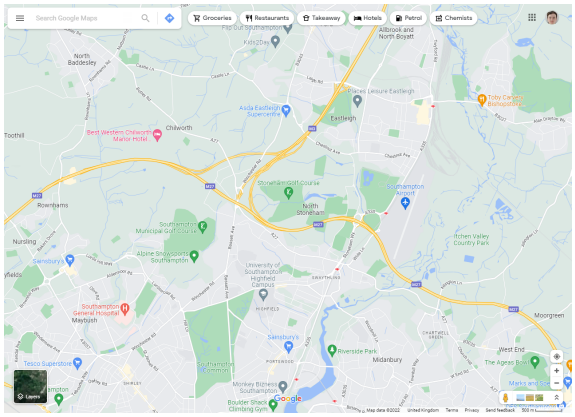
1.4.1 Existing Solutions

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.



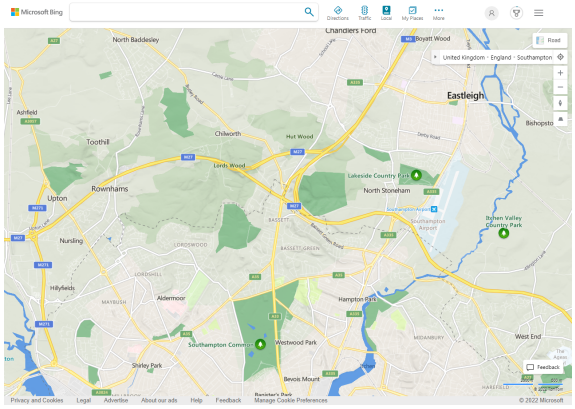
(a) Example of Google Maps' GUI

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 routable map.



(b) Example of Bing Maps' GUI

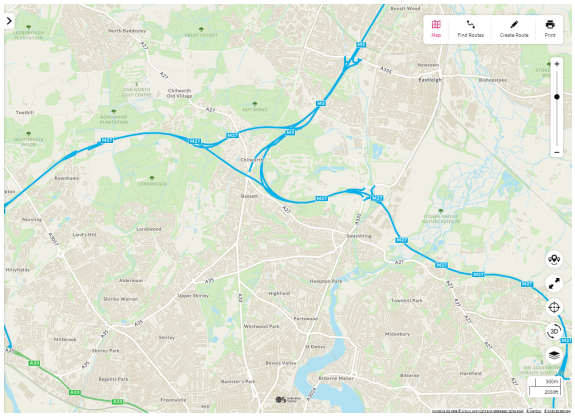
Sourced from Bing Maps©

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 therefor 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.



(c) 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 caise 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 centered 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-centered 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.

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

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.

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

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

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

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

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 $[0, 0]$.

1.5.2 Second Interview

Now that I have done some more research into the various ways there are to complete this task I have formed some more questions to ask my end user to get a solid and defined list of objectives for the program. AI will couple this with my research to form a complete plan to form said objectives. As well as this however the second interview will allow me to correct any inaccurate questions that where asked in the initial interview. This is because after I received my initial responses I realised that I needed to be more clear with what I was asking and the information that I wanted back.

HAS BEEN ASKED WAITING FOR RESPONSES

1. **Bobbert?**
   bobbert.
2. **Cobbert?**
   cobbert.
3. **Dobbert?**
   dobbert.
4. **Fobbert?**
   Fobbert.
5. **Norbert?**
   norbert
6. **Dilbert?**
   dilbert.
7. **Bobbert?**
   bobbert.

1.5.3 Evaluation of Second Interview

After conducting this second interview I feel I now have a firm understanding of what I need to achieve with this program. I will also take this opportunity to create a prototype of the the different parts of the program to gauge the difficulty of the program and any problems I may encounter before moving onto the final solution.

Apart from that however I feel the interview went...

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:

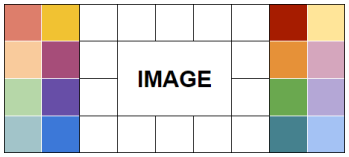
- A version of edge detection
- A graph class with basic traversal
- A forms interface for showing images

1.6.2 Edge Detection

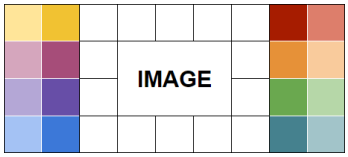
For the example of edge detection which I am going to prototype I have chosen Canny Edge Detection, this is the most common of the types of edge detection and is relatively simple. It is also widely documented which allows me to focus more on the application and less on the finding of resources.

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.

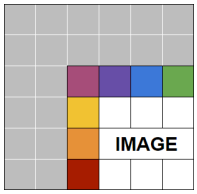


(d) Wrapping

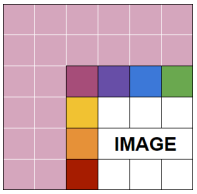


(e) Mirroring

(f) Extension



(g) Constants



(h) 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.

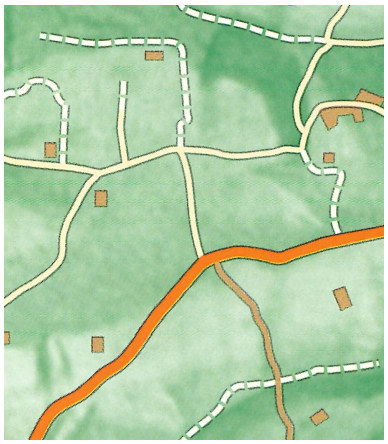


Figure 1: Original Image

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:

```
public double[,] BWFilter(Bitmap image)
{
    double[,] result = new double[image.Height, image.Width];

    for (int i = 0; i < image.Height; i++)
    {
        for (int j = 0; j < image.Width; j++)
        {
            Color c = image.GetPixel(j, i);
            double value = c.R * 0.299 + c.G * 0.587 + c.B * 0.114;

            result[i, j] = Bound(0, 255, value);
        }
    }

    return result;
}
```

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:

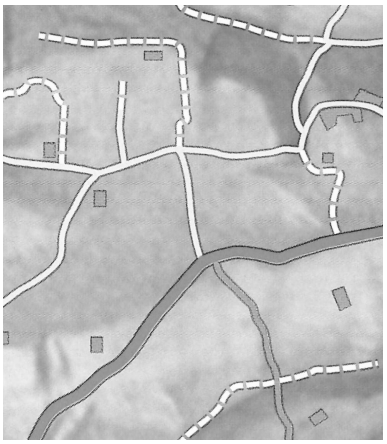


Figure 2: Black and White Filter

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:

```
public double[,] GaussianFilter(double sigma, int kernelSize, double[,] imageArray)
{
    double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];

    Matrix gaussianKernel = GetGaussianKernel(kernelSize, sigma);

    for (int i = 0; i < result.GetLength(0); i++)
    {
        for (int j = 0; j < result.GetLength(1); j++)
        {
            Matrix imageKernel = BuildKernel(j, i, kernelSize, imageArray);
            double sum = Matrix.Convolution(imageKernel, gaussianKernel);
            result[i, j] = sum;
        }
    }

    return result;
}

public Matrix GetGaussianKernel(int k, double sigma)
{
    double[,] result = new double[k, k];
    int halfK = k / 2;

    double sum = 0;

    int cntY = -halfK;
    for (int i = 0; i < k; i++)
    {
        int cntX = -halfK;
        for (int j = 0; j < k; j++)
        {
            result[halfK + cntY, halfK + cntX] = GetGaussianDistribution(cntX, cntY, sigma);
            sum += result[halfK + cntY, halfK + cntX];
            cntX++;
        }
        cntY++;
    }

    for (int i = 0; i < k; i++) for (int j = 0; j < k; j++) result[i, j] /= sum;
    return new Matrix(result);
```

```
}
```

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.

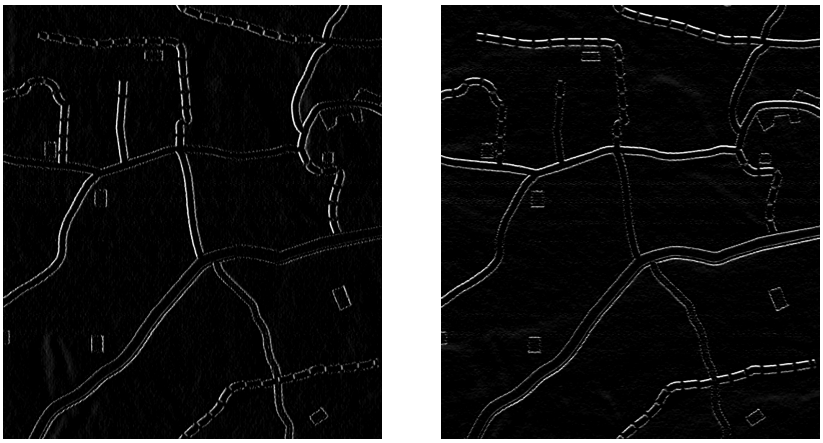
$$M_y = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad \text{and} \quad M_x = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

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*.

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

    Matrix sobelX = new Matrix(new double[,] {
```

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

    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] = Matrix.Convolution(imageKernel, sobelX);
        }
    }

    return result;
}
```

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:



(a) Gradient in X (b) 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.

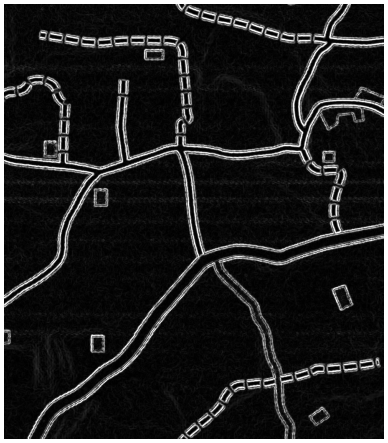


Figure 4: Gaussian Filter

4. Gradient Direction

Now that the gradient values have been calculated we can move onto working out which direction the gradient is traveling. 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 traveling 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 therefor 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 traveling. 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 therefor the process should not continue. After this has been applied to our image we are left with:

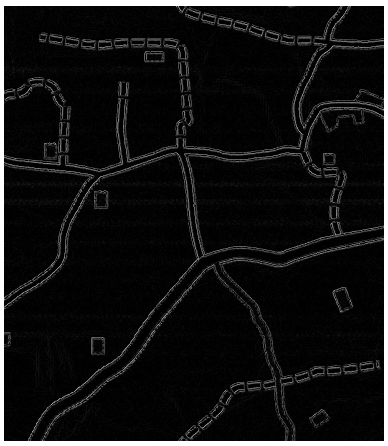


Figure 7: Magnitude Threshold

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();
        }
    }

    return result;
}
```

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.

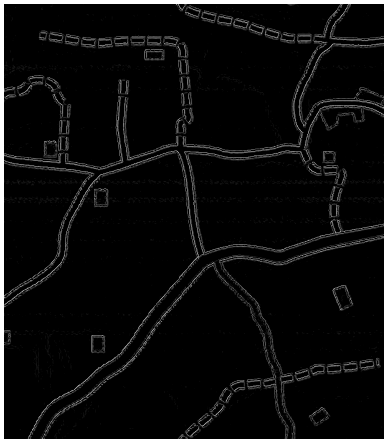


Figure 8: Magnitude Threshold

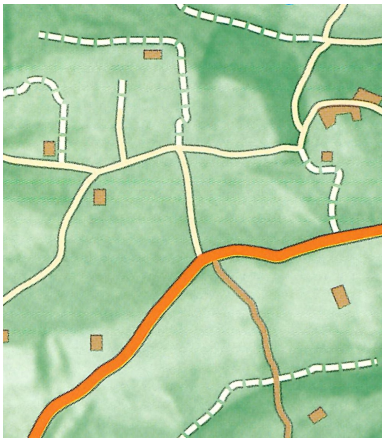
As you can see lots of noise from the scan lines of the image have been removed in this step as they would have been too small to make it past the lower threshold. Now we have an 2D array of pixel values and whether they are considered "strong" or not. If they are strong this is represented by **true**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

7. Edge tracking by Hysteresis

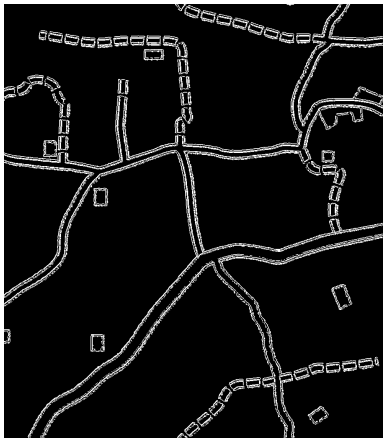
This is the final step of traditional canny edge detection. This will require the 2D array of tuples and will require kernels of the image as it loops over every pixel. This will cause a problem since the usual way fo doing it would default to grey if the kernel overlapped with the edge of the image. 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 arround it. If any of the pixels which surround it are "strong" then this pixel is made "strong". The code for this is as follows.

```
public double[,] ApplyEdgeTrackingHystersis((double, bool)[,] arrayOfValues)
{
    double[,] result = new double[arrayOfValues.GetLength(0), arrayOfValues.GetLength(1)];

    for (int i = 0; i < arrayOfValues.GetLength(0); i++)
    {
        for (int j = 0; j < arrayOfValues.GetLength(1); j++)
        {
            if (arrayOfValues[i, j].Item2 == false)
            {
                (double, bool)[,] imageKernel = BuildKernel(j, i, 3, arrayOfValues);
                bool strong = false;
                for (int k = 0; k < 3 && !strong; k++)
                {
                    for (int l = 0; l < 3 && !strong; l++)
                    {
                        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.



(a) Original Input Image



(b) 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[,] EmbosImage(double[,] imageArray)
{
    double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];

    Matrix embosMatrix = 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);
```

```
17              result[i, j] = Math.Abs(Matrix.Convolution(imageKernel, embosMatrix));
18          }
19      }
20
21      return result;
22  }
```

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

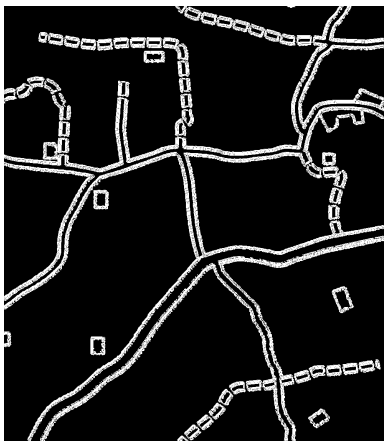


Figure 9: Magnitude Threshold

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.

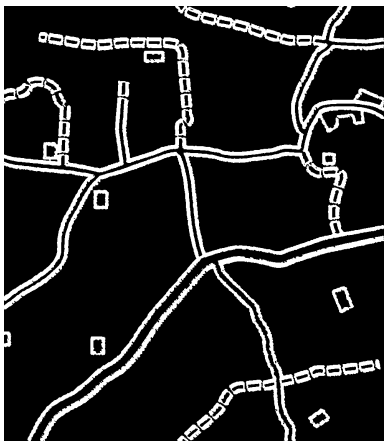


Figure 10: Magnitude Threshold

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.

```
1  public double[,] FillImage(double[,] imageArray)
2  {
```

```
    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:

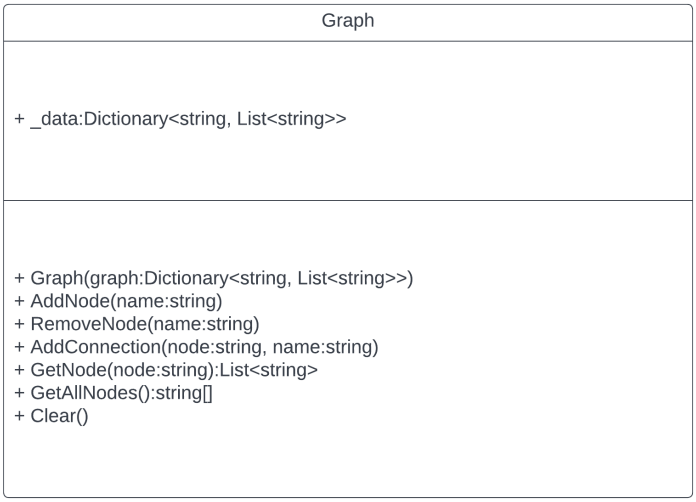


Figure 11: Graph UML Diagram

and in code

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

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

    public void AddNode(string name)
    {
        if (_data.ContainsKey(name)) throw new GraphException($"Cannot add {name}, node already
        ↪  exists.");
        _data.Add(name, new List<string>());
    }

    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.");
```

```
25          if (_data[node].Contains(name)) throw new GraphException($"Cannot add connection {name}
            ↪  to {node} connection already exists.");
26          _data[node].Add(name);
27      }
28
29      public List<string> GetNode(string node)
30      {
31          if (!_data.ContainsKey(node)) throw new GraphException($"Node {node} does not exist.");
32          return _data[node];
33      }
34
35      public string[] GetAllNodes() => _data.Keys.ToArray();
36
37      public void Clear() => _data.Clear();
38  }
```

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).

```
1   public static string[] DFS(string start, Graph graph)
2   {
3       List<string> path = new List<string>();
4       Stack<string> stack = new Stack<string>();
5       Dictionary<string, bool> visited = new Dictionary<string, bool>();
6       foreach (string s in graph.GetAllNodes()) visited.Add(s, false);
7
8       // Kick Start
9       stack.Push(start);
10
11      while (!stack.IsEmpty())
12      {
13
14          string node = stack.Pop();
15          path.Add(node);
16          visited[node] = true;
17
18          List<string> connections = graph.GetNode(node);
19
20          connections.Reverse();
21
22          foreach (string s in connections)
23          {
24              if (visited[s] == false)
25              {
26                  stack.Push(s);
27              }
28          }
```

```
    }


    return path.ToArray();
}

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

    // Kick Start
    stack.Enqueue(start);

    while (!stack.IsEmpty())
    {

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

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

        connections.Reverse();

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

    return path.ToArray();
}
```

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

programatic part to it which can be used to display the image.

Example

```
partial class ShowImage
{
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.IContainer components = null;

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    /// <param name="disposing">true if managed resources should be disposed; otherwise,
    ↪  false.</param>
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.imageBox = new System.Windows.Forms.PictureBox();
        this.next = new System.Windows.Forms.Button();
        this.content = new System.Windows.Forms.RichTextBox();
        ((System.ComponentModel.ISupportInitialize)(this.imageBox)).BeginInit();
        this.SuspendLayout();
        //
        // imageBox
        //
        this.imageBox.Location = new System.Drawing.Point(12, 12);
        this.imageBox.Name = "imageBox";
        this.imageBox.Size = new System.Drawing.Size(500, 450);
        this.imageBox.TabIndex = 1;
        this.imageBox.TabStop = false;
        //
        // next
        //
```

```
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 }
```

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.

1.7 Objectives

After conducting the initial and second interviews 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
2.1.1 The user should be able to save the intermidiate images.
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 minimisation
- 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 Modeling

TODO

2 Technical Design

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 video 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 video 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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
13	Try to Break inputs through pre-clicking enter.	When going through menus repeatably 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	TODO
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	TODO
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	TODO
2.3.1 AThere 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	TODO
2.3 The edge detection must have the option to be multi threaded.					

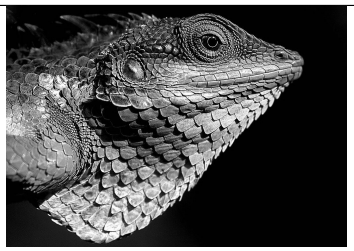
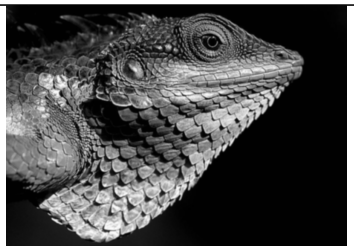
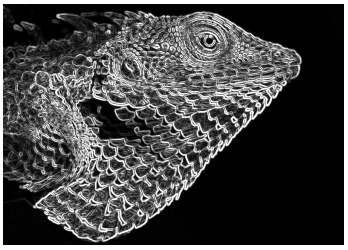
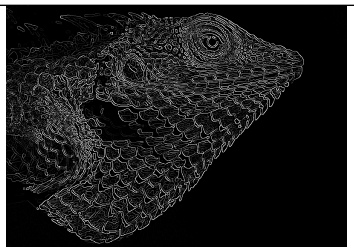
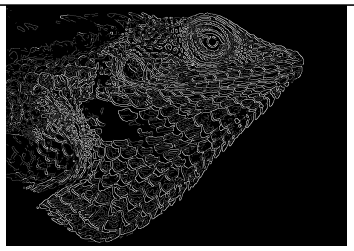
3	Cancel A Run	As above the test image should be entered. Both when it comes to the edge picking "Multi-threaded" then entering values then when the program confirms to continue select "No", and when the image is first read selecting "No" when the "Correct Image" prompt shows.	The program should stop 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	TODO
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	TODO
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	TODO

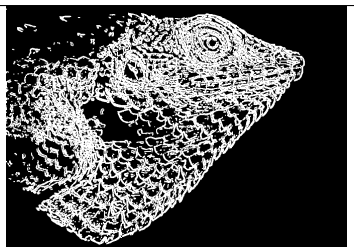
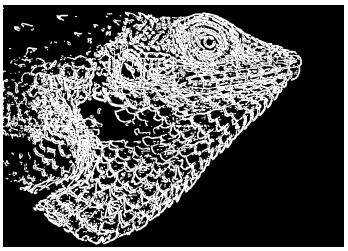
The following tests ending in "method" are run one at a time during the slow, single threaded version of canny edge detection with the exception of the Gradient calculation with error, these are used to test that each stage of the canny edge detection algorithm are correct and functioning correctly.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

(a) Example Image Used

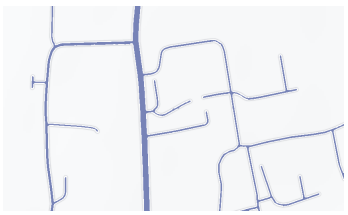
Sourced from Wikipedia©
`https://en.wikipedia.org/wiki/Canny_edge_detector#Walkthrough_of_the_algorithm`

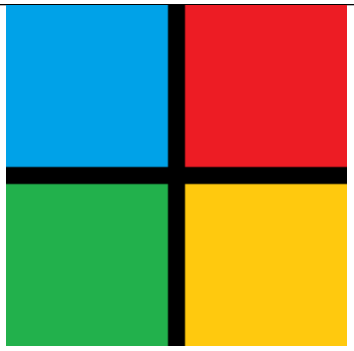
2.4 The edge detection must have the option to be single threaded

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	TODO
7	Gaussian Filter Method	Canny Edge Detection method should be run with the output of the previous step, Black and White conversion.		Pass	TODO
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	TODO
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	TODO
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	TODO

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	TODO
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	TODO

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	TODO
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	TODO
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	TODO

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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
4.1.1.2 This includes A* (a specialised Dijkstra)					
9	Run A* Image	Two nodees 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	TODO

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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO

10	Delete Some Folders and Re-run	In the directory where the program file is contained the programatically created folders should be deleted. Not all but some.	When the program is restarted the files should be recreated	Pass	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO
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	TODO

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	Marix 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	TODO
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	TODO
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	TODO
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	TODO
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	TODO

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	TODO
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 Evaluation

5 Code Base

5.1 Prototypes

5.1.1 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;
```

```
                        }
                    }

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

        return result;
    }

    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;
    }


    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++;
```

```
            }

            return kernel;
        }

        public double[,] BWFilter(Bitmap image)
        {
            double[,] result = new double[image.Height, image.Width];

            for (int i = 0; i < image.Height; i++)
            {
                for (int j = 0; j < image.Width; j++)
                {
                    Color c = image.GetPixel(j, i);
                    double value = c.R * 0.299 + c.G * 0.587 + c.B * 0.114;

                    result[i, j] = Bound(0, 255, value);
                }
            }

            return result;
        }

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

        public double GetGaussianDistribution(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 Bitmap DoubleArrayToBitmap(double[,] input)
        {
            Bitmap image = new Bitmap(input.GetLength(1), input.GetLength(0));
            for (int i = 0; i < image.Height; i++)
            {
                for (int j = 0; j < image.Width; j++)
                {
                    int val = Bound(0, 255, input[i, j]);
                    image.SetPixel(j, i, Color.FromArgb(val, val, val));
                }
            }
            return image;
        }
    }

    public class Matrix
    {
        public int x { get; private set; }
        public int y { get; private set; }
        public double[,] matrix { get; private set; }

        public Matrix(double[,] inputMatrix)
        {
            x = inputMatrix.GetLength(1);
            y = inputMatrix.GetLength(0);
            matrix = inputMatrix;
        }

        public static double Convolution(Matrix a, Matrix b)
        {
            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 }
```

5.1.2 Graph Class and DFS / BFS

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  namespace GraphStuff
6  {
7      internal class Program
8      {
9          static void Main(string[] args)
10         {
11             Dictionary<string, List<string>> temp = new Dictionary<string, List<string>>();
12             temp.Add("A", new List<string>
13             {
14                 "D"
15             });
16             temp.Add("B", new List<string>
17             {
18                 "C", "F"
19             });
20             temp.Add("C", new List<string>
21             {
22                 "B"
23             });
24             temp.Add("D", new List<string>
25             {
26                 "A", "E", "G"
27             });
28             temp.Add("E", new List<string>
29             {
30                 "D", "H"
31             });
32             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
            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);
                    }
                }
            }


            return path.ToArray();
        }

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

            // 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 }
```

5.1.3 Forms Interface

5.2 Final Solution

5.2.1 BackendLib

5.2.1.1 Data

MapFile.cs

```
1 public class MapFile
2 {
```

```
3       public readonly string _filePath;
4       private const string FileExtensionRegex = @"^([a-z]:\\|\\|[a-z]|\.\.(\\|\/)|\.(\\|\/))(([a-z]|(\\|\/))+)\.(vmap)$";
5
6       public string Name { get; set; }
7       public string Description { get; set; }
8       public int Type { get; set; }
9       public bool IsInverted { get; set; }
10      public DateTimeOffset TimeCreated { get; set; }
11      public Bitmap PathImage { get; set; }
12      public Bitmap OriginalImage { get; set; }
13      public Bitmap CombinedImage { get; set; }
14
15      public MapFile()
16      {
17          TimeCreated = DateTimeOffset.Now;
18      }
19
20      public MapFile(string filePath)
21      {
22          _filePath = filePath;
23      }
24
25      public void Initialize(Action updateProgress)
26      {
27          ValidateImage();
28          updateProgress();
29          ReadMapFile(updateProgress);
30      }
31
32      private void ValidateImage()
33      {
34          Regex fileRegex = new Regex(FileExtensionRegex, RegexOptions.IgnoreCase);
35
36          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.");
37          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.");
38      }
39
40      private void ReadMapFile(Action updateProgress)
41      {
42          using (BinaryReader br = new BinaryReader(File.Open(_filePath, FileMode.Open)))
43          {
44              string dateTime = br.ReadString();
45              DateTime dt = new DateTime(1970, 1, 1, 0, 0, 0, 0,
                ↪  DateTimeKind.Utc).AddMilliseconds(double.Parse(dateTime)).ToLocalTime();
46              TimeCreated = new DateTimeOffset(dt);
47
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                           }
108                           else if (j == 1)
109                           {
110                               if (i == 0) bw.Write(PathImage.GetPixel(x, y).R);
111                               else if (i == 1) bw.Write(PathImage.GetPixel(x, y).G);
112                               else if (i == 2) bw.Write(PathImage.GetPixel(x, y).B);
113                           }
114                           else if (j == 2)
115                           {
116                               if (i == 0) bw.Write(CombinedImage.GetPixel(x, y).R);
117                               else if (i == 1) bw.Write(CombinedImage.GetPixel(x, y).G);
118                               else if (i == 2) bw.Write(CombinedImage.GetPixel(x, y).B);
119                           }
```

```
                        }
                    }
                }
            }
        }

        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>();
```

```
48          foreach (T s in _graph.GetAllNodes()) visited.Add(s, false);
49
50          // Kick Start
51          queue.Enqueue(start);
52
53          while (!queue.IsEmpty())
54          {
55
56              T node = queue.Dequeue();
57              path.Add(node);
58              visited[node] = true;
59
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;
```

```
                    else prev.Add(neighbor, current);
                    queue.ChangePriority(neighbor, tentative + weightFunction(neighbor, goal));
                }
            }
        }


        return new Dictionary<T, T>();
    }

    public Dictionary<T, T> Dijkstra(T start, T goal, bool endOnFind, Action nodeUpdate)
    {
        Dictionary<T, double> dist = new Dictionary<T, double>();
        Dictionary<T, T> prev = new Dictionary<T, T>();
        dist.Add(start, 0);

        MinPriorityQueue<T> queue = new MinPriorityQueue<T>();

        T[] nodes = _graph.GetAllNodes();
        foreach (T node in nodes)
        {
            if (_graph.GetNode(node).Count > 0)
            {
                if (!Equals(start, node)) dist.Add(node, double.MaxValue);
                queue.Enqueue(node, dist[node]);
            }
        }

        while (queue.Size > 0)
        {
            T minVertex = queue.Dequeue();
            nodeUpdate();
            if (Equals(minVertex, goal) && endOnFind) return prev;

            List<T> adjacent = _graph.GetNode(minVertex);

            foreach (var neighbor in adjacent)
            {

                if (queue.Contains(neighbor))
                {
                    double alternateWeight = dist[minVertex] + 1;
                    if (alternateWeight < dist[neighbor])
                    {
                        dist[neighbor] = alternateWeight;
                        if (prev.ContainsKey(neighbor)) prev[neighbor] = minVertex;
                        else prev.Add(neighbor, minVertex);
                        queue.ChangePriority(neighbor, alternateWeight);
                    }
                }
            }
        }

        return prev;
    }
}
```

5.2.1.2 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);
```

```
11          Y = matrix.GetLength(0);
12      }
13
14      public Matrix(int x, int y)
15      {
16          _matrix = new double[y, x];
17          X = x;
18          Y = y;
19      }
20
21
22      public double this[int y, int x]
23      {
24          get => _matrix[y, x];
25          private set => _matrix[y, x] = value;
26      }
27
28      public static Matrix operator +(Matrix a, Matrix b)
29      {
30          if (a.X != b.X || a.Y != b.Y) throw new MatrixException("Matrices must be the same dimensions to add.");
31
32          Matrix m = new Matrix(a.X, a.Y);
33          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];
34          return m;
35      }
36
37      public static Matrix operator -(Matrix a, Matrix b)
38      {
39          if (a.X != b.X || a.Y != b.Y) throw new MatrixException("Matrices must be the same dimensions to subtract.");
40
41          Matrix m = new Matrix(a.X, a.Y);
42          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];
43          return m;
44      }
45      public static Matrix operator *(Matrix a, Matrix b)
46      {
47          if (a.X != b.X || a.Y != b.Y) throw new MatrixException("Matrices must be the same dimensions to multiply.");
48
49          Matrix m = new Matrix(a.X, a.Y);
50          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];
51          return m;
52      }
53
54      public static Matrix operator *(int a, Matrix b)
55      {
56          Matrix m = new Matrix(b.X, b.Y);
57          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;
```

```
            }
        }
    }

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

        double[,] flippedB = new double[b.Y, b.X];
        int l = b.X;
        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];

        double sum = 0;
        for (int i = 0; i < a.Y; i++) for (int j = 0; j < a.X; j++) sum += a[i, j] * flippedB[i, j];
        return sum;
    }

    public IEnumerator GetEnumerator() => _matrix.GetEnumerator();

}
```

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);

        ShiftNodeUp(_size);
    }

    public T Dequeue() => RemoveMax().Item1;

    private (T, int) RemoveMax()
    {
        int res = _priorityQueue[0];
        T result = _queue[0];
        _priorityQueue.RemoveAt(0);
        _queue.RemoveAt(0);

        ShiftNodeDown(0);

        return (result, res);
    }

    private void Swap(int indexX, int indexY)
    {
        T tempValue = _queue[indexX];
        _queue[indexX] = _queue[indexY];
        _queue[indexY] = tempValue;

        int tempPriority = _priorityQueue[indexX];
        _priorityQueue[indexX] = _priorityQueue[indexY];
        _priorityQueue[indexY] = tempPriority;
    }

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

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);
            _priorityQueue.RemoveAt(0);

            return val;
```

```
        }

        T res = _queue[0];

        int oldSize = _size;

        _queue[0] = _queue[oldSize];
        _queue.RemoveAt(oldSize);
        _priorityQueue[0] = _priorityQueue[oldSize];
        _priorityQueue.RemoveAt(oldSize);

        MinifyHeap(0);

        return res;
    }

    private void MinifyHeap(int index)
    {
        int left = Left(index);
        int right = Right(index);

        int smallest = index;

        if (left < Size && _priorityQueue[left] < _priorityQueue[smallest]) smallest = left;
        if (right < Size && _priorityQueue[right] < _priorityQueue[smallest]) smallest = right;
        if (smallest != index)
        {
            Swap(index, smallest);
            MinifyHeap(smallest);
        }
    }

    private void Swap(int indexX, int indexY)
    {
        T tempValue = _queue[indexX];
        _queue[indexX] = _queue[indexY];
        _queue[indexY] = tempValue;

        double tempPriority = _priorityQueue[indexX];
        _priorityQueue[indexX] = _priorityQueue[indexY];
        _priorityQueue[indexY] = tempPriority;
    }

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

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);
}
```

5.2.1.3 Exceptions

GraphException.cs

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

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

    public GraphException(string message, Exception innerException) : base(message, innerException)
    {
```

```
    }

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

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)
    {
    }
}
```

5.2.1.4 Interfaces

IHandler.cs

```
public interface IHandler
{
    void Start();
    double[,] Result();
}
```

5.2.1.5 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;
    }

    /// <summary>
    /// Convert a given image in the form of a RGB double array will convert it to a single double array of black and
    ↪ white pixels.
    /// </summary>
    /// <param name="input">The image to be converted to black and white</param>
```

```
28      /// <returns>The processed double array</returns>
29      public double[,] BlackWhiteFilter(Structures.RGB[,] input)
30      {
31          double[,] output = new double[input.GetLength(0), input.GetLength(1)];
32
33          for (int y = 0; y < input.GetLength(0); y++)
34          {
35              for (int x = 0; x < input.GetLength(1); x++)
36              {
37                  output[y, x] = (input[y, x].R * RedRatio) + (input[y, x].G * GreenRatio) + (input[y, x].B * BlueRatio);
38              }
39          }
40
41          return output;
42      }
43
44      public double[,] GaussianFilter(double[,] input)
45      {
46          double[,] output = new double[input.GetLength(0), input.GetLength(1)];
47
48          Matrix gaussianKernel = new Matrix(Kernel<double>.Gaussian(Sigma, KernelSize));
49          Kernel<double> masterKernel = new Kernel<double>(input);
50
51          for (int y = 0; y < input.GetLength(0); y++)
52          {
53              for (int x = 0; x < input.GetLength(1); x++)
54              {
55                  Matrix subKernel = new Matrix(masterKernel.Duplication(x, y, KernelSize));
56                  double sum = Matrix.Convolution(subKernel, gaussianKernel);
57                  output[y, x] = sum;
58              }
59          }
60
61          return output;
62      }
63
64      public Structures.Gradients CalculateGradients(double[,] input, Action updateMenu)
65      {
66          Task<double[,]>[] tasks =
67          {
68              new Task<double[,]>(() => CalculateGradientX(input, updateMenu)),
69              new Task<double[,]>(() => CalculateGradientY(input, updateMenu))
70          };
71
72          foreach (var task in tasks) task.Start();
73
74          Task.WaitAll(tasks);
75
76          return new Structures.Gradients
77          {
78              GradientX = tasks[0].Result,
79              GradientY = tasks[1].Result
80          };
81      }
82
83      private double[,] CalculateGradientX(double[,] input, Action updateMenu)
84      {
85          double[,] output = new double[input.GetLength(0), input.GetLength(1)];
86
87          Matrix sobelMatrixY = new Matrix(new double[,] { { 1, 0, -1 }, { 2, 0, -2 }, { 1, 0, -1 } });
```

```
88          Kernel<double> masterKernel = new Kernel<double>(input);
89
90          for (int y = 0; y < input.GetLength(0); y++)
91          {
92              for (int x = 0; x < input.GetLength(1); x++)
93              {
94                  Matrix imageKernel = new Matrix(masterKernel.Duplication(x, y, 3));
95                  output[y, x] = Matrix.Convolution(imageKernel, sobelMatrixY);
96              }
97          }
98
99          updateMenu();
100         return output;
101     }
102
103     private double[,] CalculateGradientY(double[,] input, Action updateMenu)
104     {
105         double[,] output = new double[input.GetLength(0), input.GetLength(1)];
106
107         Matrix sobelMatrixY = new Matrix(new double[,] { { 1, 2, 1 }, { 0, 0, 0 }, { -1, -2, -1 } });
108         Kernel<double> masterKernel = new Kernel<double>(input);
109
110         for (int y = 0; y < input.GetLength(0); y++)
111         {
112             for (int x = 0; x < input.GetLength(1); x++)
113             {
114                 Matrix imageKernel = new Matrix(masterKernel.Duplication(x, y, 3));
115                 output[y, x] = Matrix.Convolution(imageKernel, sobelMatrixY);
116             }
117         }
118
119         updateMenu();
120         return output;
121     }
122
123     public double[,] CombineGradients(Structures.Gradients grads)
124     {
125         if (grads.GradientX.GetLength(0) != grads.GradientY.GetLength(0) || grads.GradientX.GetLength(1) !=
        ↪  grads.GradientY.GetLength(1))
126             throw new ArgumentException("Canny edge detection failed due to arrays not being of the same size.");
127
128         double[,] output = new double[grads.GradientX.GetLength(0), grads.GradientX.GetLength(1)];
129
130         for (int y = 0; y < grads.GradientX.GetLength(0); y++)
131         {
132             for (int x = 0; x < grads.GradientX.GetLength(1); x++)
133             {
134                 output[y, x] = Math.Sqrt(Math.Pow(grads.GradientX[y, x], 2) + Math.Pow(grads.GradientY[y, x], 2));
135             }
136         }
137
138         return output;
139     }
140
141     public double[,] GradientAngle(Structures.Gradients grads)
142     {
143         if (grads.GradientX.GetLength(0) != grads.GradientY.GetLength(0) || grads.GradientX.GetLength(1) !=
        ↪  grads.GradientY.GetLength(1))
144             throw new ArgumentException("Canny edge detection failed due to arrays not being of the same size.");
145
```

```
146         double[,] output = new double[grads.GradientX.GetLength(0), grads.GradientX.GetLength(1)];
147 
148         for (int y = 0; y < grads.GradientX.GetLength(0); y++)
149         {
150             for (int x = 0; x < grads.GradientX.GetLength(1); x++)
151             {
152                 output[y, x] = Math.Atan2(grads.GradientY[y, x], grads.GradientX[y, x]);
153             }
154         }
155 
156         return output;
157     }
158 
159     public double[,] MagnitudeThreshold(double[,] gradCombined, double[,] gradAngle)
160     {
161         if (gradCombined.GetLength(0) != gradAngle.GetLength(0) || gradCombined.GetLength(1) != gradAngle.GetLength(1))
162             throw new ArgumentException("Canny edge detection failed due to arrays not being of the same size.");
163 
164         double[,] output = gradCombined;
165         double[,] anglesInDegrees = new double[gradCombined.GetLength(0), gradCombined.GetLength(1)];
166 
167         for (int y = 0; y < anglesInDegrees.GetLength(0); y++)
168         {
169             for (int x = 0; x < anglesInDegrees.GetLength(1); x++)
170             {
171                 anglesInDegrees[y, x] = Utility.RadianToDegree(gradAngle[y, x]);
172             }
173         }
174 
175         Kernel<double> masterKernel = new Kernel<double>(gradCombined);
176 
177         for (int y = 0; y < anglesInDegrees.GetLength(0); y++)
178         {
179             for (int x = 0; x < anglesInDegrees.GetLength(1); x++)
180             {
181                 double[,] magnitudeKernel = masterKernel.Duplication(x, y, 3);
182 
183                 if (anglesInDegrees[y, x] < 22.5 || anglesInDegrees[y, x] >= 157.5)
184                 {
185                     if (gradCombined[y, x] < magnitudeKernel[1, 2] || gradCombined[y, x] < magnitudeKernel[1, 0])
186                         output[y, x] = 0;
187                 }
188                 else if (anglesInDegrees[y, x] >= 22.5 && anglesInDegrees[y, x] < 67.5)
189                 {
190                     if (gradCombined[y, x] < magnitudeKernel[0, 2] || gradCombined[y, x] < magnitudeKernel[2, 0])
191                         output[y, x] = 0;
192                 }
193                 else if (anglesInDegrees[y, x] >= 67.5 && anglesInDegrees[y, x] < 112.5)
194                 {
195                     if (gradCombined[y, x] < magnitudeKernel[0, 1] || gradCombined[y, x] < magnitudeKernel[2, 1])
196                         output[y, x] = 0;
197                 }
198                 else if (anglesInDegrees[y, x] >= 112.5 && anglesInDegrees[y, x] < 157.5)
199                 {
200                     if (gradCombined[y, x] < magnitudeKernel[0, 0] || gradCombined[y, x] < magnitudeKernel[2, 2])
201                         output[y, x] = 0;
202                 }
203                 else throw new Exception("Critical unknown error occurred, please try again.");
204             }
205         }
```

```
206
207         return output;
208     }
209
210     public Structures.ThresholdPixel[,] DoubleThreshold(double[,] input)
211     {
212         double min = LowerThreshold * 255;
213         double max = UpperThreshold * 255;
214
215         Structures.ThresholdPixel[,] output = new Structures.ThresholdPixel[input.GetLength(0), input.GetLength(1)];
216
217         for (int y = 0; y < input.GetLength(0); y++)
218         {
219             for (int x = 0; x < input.GetLength(1); x++)
220             {
221                 if (input[y, x] < min) output[y, x] = new Structures.ThresholdPixel { Strong = false, Value = 0 };
222                 else if (input[y, x] > min && input[y, x] < max) output[y, x] = new Structures.ThresholdPixel { Strong =
                    ↪  false, Value = input[y, x] };
223                 else if (input[y, x] > max) output[y, x] = new Structures.ThresholdPixel { Strong = true, Value =
                    ↪  input[y, x] };
224                 else throw new Exception("Critical unknown error occurred, please try again.");
225             }
226         }
227
228         return output;
229     }
230
231     public double[,] EdgeTrackingHysteresis(Structures.ThresholdPixel[,] input)
232     {
233         double[,] output = new double[input.GetLength(0), input.GetLength(1)];
234
235         Kernel<Structures.ThresholdPixel> masterKernel = new Kernel<Structures.ThresholdPixel>(input);
236
237         for (int i = 0; i < input.GetLength(0); i++)
238         {
239             for (int j = 0; j < input.GetLength(1); j++)
240             {
241                 if (input[i, j].Strong == false)
242                 {
243                     Structures.ThresholdPixel[,] imageKernel = masterKernel.Duplication(j, i, 3);
244                     bool strong = false;
245                     for (int k = 0; k < 3 && !strong; k++)
246                     {
247                         for (int l = 0; l < 3 && !strong; l++)
248                         {
249                             if (imageKernel[k, l].Strong) strong = true;
250                         }
251                     }
252                     output[i, j] = strong ? 255 : 0;
253                 }
254                 else output[i, j] = 255;
255             }
256         }
257
258         return output;
259     }
260 }
```

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;
    }


```

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

Pre.cs

```
1   public class Pre
2   {
3       private readonly string _imagePath;
4       private Bitmap _imageBitmap;
5       private Structures.RGB[,] _imageRgb;
6
7       private const string FileExtensionRegex =
        ↪  @"^([a-z]:\\|\\|[a-z]|\.\.(\\|\/)|\.(\\|\/))((\w|(\\|\/))+)\.(jpg|bmp|exif|png|tiff)$";
8
9       public Pre(string imagePath)
10      {
11          _imagePath = imagePath;
12      }
13
14
15      /// <exception cref="PreprocessingException"></exception>
16      /// <exception cref="Exception"></exception>
17      public void Start(Action updateProgressAction)
18      {
19          updateProgressAction();
20          ValidatePath();
21          updateProgressAction();
22          ReadImage();
23          updateProgressAction();
24          CheckDimensions();
25          updateProgressAction();
26      }
27
28      private void ValidatePath()
29      {
30          Regex fileRegex = new Regex(FileExtensionRegex, RegexOptions.IgnoreCase);
31
32          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.");
33          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.");
34      }
35
36      private void ReadImage()
37      {
38          _imageBitmap = new Bitmap(_imagePath, true);
39          _imageRgb = new Structures.RGB[_imageBitmap.Height, _imageBitmap.Width];
40
41          for (int y = 0; y < _imageBitmap.Height; y++)
42          {
43              for (int x = 0; x < _imageBitmap.Width; x++)
44              {
45                  Color tempPixel = _imageBitmap.GetPixel(x, y);
46                  _imageRgb[y, x] = new Structures.RGB
47                  {
48                      R = tempPixel.R,
49                      G = tempPixel.G,
50                      B = tempPixel.B
51                  };
```

```
            }
        }
    }

    private void CheckDimensions()
    {
        if (_imageRgb.GetLength(0) < 200 || _imageRgb.GetLength(1) < 200)
            throw new PreprocessingException("The image you supplied is too small to work properly it must be at least
            ↪  200x200. Try a larger image.");

        if (_imageRgb.GetLength(0) % 2 != 0 || _imageRgb.GetLength(1) % 2 != 0)
        {
            Structures.RGB[,] resizedRgb =
                new Structures.RGB[_imageRgb.GetLength(0) / 2 * 2, _imageRgb.GetLength(1) / 2 * 2];

            for (int y = 0; y < _imageRgb.GetLength(0) / 2 * 2; y++)
            {
                for (int x = 0; x < _imageRgb.GetLength(1) / 2 * 2; x++)
                {
                    resizedRgb[y, x] = _imageRgb[y, x];
                }
            }

            _imageRgb = resizedRgb;
        }
    }

    public Structures.RawImage Result() => new Structures.RawImage
    {
        Original = _imageBitmap,
        Pixels = _imageRgb,
        Path = _imagePath,
        Height = _imageBitmap.Height,
        Width = _imageBitmap.Width
    };
}
```

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));

                Datatypes.Queue<(int, int)> queue = new Datatypes.Queue<(int, int)>();
                queue.Enqueue((y, x));

                while (queue.Size > 0)
                {
                    (int, int) cord = queue.Dequeue();
                    if (tempImage[cord.Item1, cord.Item2] == Color.FromArgb(0, 0, 0))
                    {
                        tempImage[cord.Item1, cord.Item2] = randCol;
                        _filledBitmap.SetPixel(cord.Item2, cord.Item1, tempImage[cord.Item1, cord.Item2]);

                        if (cord.Item1 > 0) queue.Enqueue((cord.Item1 - 1, cord.Item2));
                        if (cord.Item2 > 0) queue.Enqueue((cord.Item1, cord.Item2 - 1));
                        if (cord.Item1 < _filledBitmap.Height - 1) queue.Enqueue((cord.Item1 + 1, cord.Item2));
                        if (cord.Item2 < _filledBitmap.Width - 1) queue.Enqueue((cord.Item1, cord.Item2 + 1));

                        if (!usedColors.Contains(randCol)) usedColors.Add(randCol);

                        filled++;
                    }
                    else if (tempImage[cord.Item1, cord.Item2] == Color.FromArgb(255, 255, 255))
                    {
```

```
                        tempImage[cord.Item1, cord.Item2] = Color.FromArgb(1, 1, 1);
                        _filledBitmap.SetPixel(cord.Item2, cord.Item1, tempImage[cord.Item1, cord.Item2]);
                    }

                    if (cord.Item1 > maxY) maxY = cord.Item1;
                    if (cord.Item2 > maxX) maxX = cord.Item2;
                    if (cord.Item1 < minY) minY = cord.Item1;
                    if (cord.Item2 < minX) minX = cord.Item2;
                }

                double totalSquares = (maxX - minX) * (maxY - minY);
                if (filled / totalSquares > _threshold || filled == 1) toReplaceColors.Add(randCol);
            }
        }

        return toReplaceColors;
    }

    private void RemoveColor(List<Color> toRemove, Action updateAction)
    {
        _pathBitmap = new Bitmap(_filledBitmap);

        for (int y = 0; y < _pathBitmap.Height; y++)
        {
            for (int x = 0; x < _pathBitmap.Width; x++)
            {
                if ((((y + 1) * (x + 1)) / 100) % 100 == 0) updateAction();
                if (toRemove.Contains(_pathBitmap.GetPixel(x, y)))
                {
                    _pathBitmap.SetPixel(x, y, Color.FromArgb(1, 1, 1));
                }
            }
        }

        for (int i = 0; i < _pathBitmap.Height; i++)
        {
            for (int j = 0; j < _pathBitmap.Width; j++)
            {
                if ((((i + 1) * (j + 1)) / 100) % 100 == 0) updateAction();
                if (_pathBitmap.GetPixel(j, i) == Color.FromArgb(1, 1, 1))
                    _pathBitmap.SetPixel(j, i, Color.FromArgb(0, 0, 0));
            }
        }
    }

    public Structures.RoadResult Result() => new Structures.RoadResult
    {
        FilledBitmap = _filledBitmap,
        PathBitmap = _pathBitmap,
        PathDoubles = _pathDoubles
    };
}
```

5.2.1.6 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++)
            {
                Structures.Coord tempCord = new Structures.Coord { X = x, Y = y };
                output.AddNode(tempCord);

                double[,] surroundingDoubles = masterKernel.Constant(x, y, 3, 0);

                bool found = false;

```

```
                if (doubles[y, x] == 255)
                {
                    for (int i = 0; i < 9; i++)
                    {
                        if (surroundingDoubles[i / 3, i % 3] != 0 && i != 4)
                        {
                            output.AddConnection(tempCord, new Structures.Coord { X = (x + (i % 3)) - 1, Y = (y + (i /
                            ↪  3)) - 1 });
                            found = true;
                        }
                    }
                }

                if (!found) output.RemoveNode(tempCord);
            }
        }

        return output;
    }

    // To ensure compatibility with BITMAP
    public static void SetPixel(this Structures.RGB[,] pixels, int x, int y, Structures.RGB toSetPixel) => pixels[y, x]
    ↪  = toSetPixel;

    public static Structures.RGB GetPixel(this Structures.RGB[,] pixels, int x, int y) => pixels[y, x];
}
```

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];

        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 static double[,] Gaussian(double sigma, int size)
    {
        double[,] result = new double[size, size];
        int halfK = size / 2;

        double sum = 0;

        int cntY = -halfK;
        for (int i = 0; i < size; i++)
        {
            int cntX = -halfK;
            for (int j = 0; j < size; j++)
            {
                result[halfK + cntY, halfK + cntX] = Utility.GaussianDistribution(cntX, cntY, sigma);
```

```
                sum += result[halfK + cntY, halfK + cntX];
                cntX++;
            }
            cntY++;
        }

        for (int i = 0; i < size; i++) for (int j = 0; j < size; j++) result[i, j] /= sum;
        return result;
    }

}
```

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)
    {
```

```
47          lock (Lock)
48          {
49              using (StreamWriter sr = File.AppendText($"./logs/{currentGuid}.txt"))
50                  sr.WriteLine($"{message}");
51          }
52      }
53
54      public static void WriteLineToMaster(string message)
55      {
56          lock (Lock)
57          {
58              using (StreamWriter sr = File.AppendText("./logs/master.txt"))
59                  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

```
1   public class Structures
2   {
3       public struct ThresholdPixel
4       {
5           public bool Strong;
6           public double Value;
7       }
8
9       public struct RGB
10      {
11          public double R;
12          public double G;
13          public double B;
14      }
15
16      public struct Gradients
17      {
18          public double[,] GradientX;
19          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;
            }
        }
    }
}

```

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));
            }
        }
```

```
72
73          for (int i = image.GetLength(1) / 2; i < image.GetLength(1); i++)
74          {
75              for (int j = image.GetLength(0) / 2; j < image.GetLength(0); j++)
76              {
77                  delta.SetPixel(i - (image.GetLength(1) / 2), j - (image.GetLength(0) / 2), image.GetPixel(i, j));
78              }
79          }
80
81          return new[] { one, beta, gamma, delta };
82      }
83
84      public static double[,] CombineQuadrants(double[,] alpha, double[,] beta, double[,] gamma, double[,] delta)
85      {
86          double[,] partA = new double[alpha.GetLength(0), alpha.GetLength(1) * 2];
87          double[,] partB = new double[alpha.GetLength(0), alpha.GetLength(1) * 2];
88          for (int i = 0; i < alpha.GetLength(0); i++)
89          {
90              for (int j = 0; j < alpha.GetLength(1); j++)
91                  partA[i, j] = alpha[i, j];
92
93              for (int y = 0; y < beta.GetLength(1); y++)
94                  partA[i, y + alpha.GetLength(1)] = beta[i, y];
95          }
96
97          for (int i = 0; i < gamma.GetLength(0); i++)
98          {
99              for (int j = 0; j < gamma.GetLength(1); j++)
100                 partB[i, j] = gamma[i, j];
101
102             for (int y = 0; y < delta.GetLength(1); y++)
103                 partB[i, y + gamma.GetLength(1)] = delta[i, y];
104         }
105
106         double[,] final = new double[alpha.GetLength(0) * 2, alpha.GetLength(1) * 2];
107         for (int i = 0; i < alpha.GetLength(0) * 2; i++)
108         {
109             if (i < alpha.GetLength(0) * 2 / 2)
110             {
111                 for (int j = 0; j < alpha.GetLength(1) * 2; j++)
112                 {
113                     final[i, j] = partA[i, j];
114                 }
115             }
116             else
117             {
118                 for (int j = 0; j < alpha.GetLength(1) * 2; j++)
119                 {
120                     final[i, j] = partB[i - alpha.GetLength(0) * 2 / 2, j];
121                 }
122             }
123         }
124
125         return final;
126     }
127
128     public static double[,] InverseImage(double[,] image)
129     {
130         for (int y = 0; y < image.GetLength(0); y++)
131         {
```

```
            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));
}
```

5.2.2 LocalApp

5.2.2.1 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:",
                new[] { "Screenshot", "Hand Drawn", "Photograph", "Other" });

            mapSave.Name = inputHandel.TryGetInput("Enter a name for image, or leave blank for 'None':");
            _menuInstance.WriteLine();

            mapSave.Description = inputHandel.TryGetInput("Enter a brief description about this image, or leave blank for 'None':");
        }

        Structures.RawImage result = preProcess.Result();
        if (saveAsBinary) result.MapFile = mapSave;
        else result.MapFile = null;
        if (saveAsBinary) mapSave.OriginalImage = result.Pixels.ToBitmap();

        return result;
    }
}
```

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;
```

```
15          _oldSettings = new Dictionary<string, (string, Type)>(Settings.UserSettings);
16          _inputHandel = new Input(_menuInstance);
17      }
18
19      public void Start()
20      {
21          bool running = true;
22
23          while (running)
24          {
25              _menuInstance.SetPage("Settings Home Page");
26              int opt = _inputHandel.GetOption("Whcih settings would you like to change?",
27                  new[]
28                  {
29                      "General",
30                      "Pathfinding",
31                      "Save",
32                      "Algorithm",
33                      "Exit"
34                  }
35              );
36
37              switch (opt)
38              {
39                  case 0:
40                      _menuInstance.SetPage("Settings -> General Settings");
41                      General();
42                      break;
43                  case 1:
44                      _menuInstance.SetPage("Settings -> Pathfinding Settings");
45                      Pathfinding();
46                      break;
47                  case 2:
48                      _menuInstance.SetPage("Settings -> Save Settings");
49                      Save();
50                      break;
51                  case 3:
52                      _menuInstance.SetPage("Settings -> Pathfinding Algorithm");
53
54                      int algorithmOption = _inputHandel.GetOption("Select which pathfinding algorithm you wish to use:",
                        ↪  new string[] {
55                          "Dijkstra",
56                          "AStar"
57                      });
58
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 }
```

5.2.2.2 CLI

Input.cs

```
1   public class Input
2   {
3       private readonly Menu _menuInstance;
4
5       public Input(Menu menuInstance)
6       {
7           _menuInstance = menuInstance;
8       }
9
10      /// <summary>
11      /// A function to easily display a menu and get an option from a supplied list.
12      /// </summary>
13      /// <param name="title">Title of the menu to be displayed</param>
14      /// <param name="options">Options to be displayed</param>
15      /// <param name="clear">Clear the screen on function call</param>
16      /// <returns>0 based index for the option which was selected</returns>
17      public int GetOption(string title, IEnumerable<string> options, bool clear = true)
18      {
19          while (Console.KeyAvailable) Console.ReadKey(true);
20          _menuInstance.ClearUserSection();
21          _menuInstance.WriteLine(title);
```

```
22
23          int j = 3;
24
25          lock (_menuInstance.ScreenLock)
26          {
27              foreach (var option in options)
28              {
29                  Console.SetCursorPosition(1, j++);
30                  Console.WriteLine($"  {option}");
31              }
32          }
33
34          bool selected = false;
35          int currentTop;
36
37          lock (_menuInstance.ScreenLock)
38          {
39              Console.SetCursorPosition(1, 3);
40              Console.Write('>');
41
42              currentTop = Console.CursorTop;
43          }
44
45          while (!selected)
46          {
47              Console.CursorVisible = false;
48
49              ConsoleKeyInfo key = Console.ReadKey(true);
50              if (key.Key == ConsoleKey.DownArrow && currentTop < options.Count() + 2)
51              {
52                  lock (_menuInstance.ScreenLock)
53                  {
54                      Console.CursorLeft = 1;
55                      Console.CursorTop = currentTop;
56                      Console.Write(' ');
57                      Console.CursorTop = ++currentTop;
58                      Console.CursorLeft = 1;
59                      Console.Write('>');
60                  }
61              }
62              else if (key.Key == ConsoleKey.UpArrow && currentTop > 3)
63              {
64                  lock (_menuInstance.ScreenLock)
65                  {
66                      Console.CursorLeft = 1;
67                      Console.CursorTop = currentTop;
68                      Console.Write(' ');
69                      Console.CursorTop = --currentTop;
70                      Console.CursorLeft = 1;
71                      Console.Write('>');
72                  }
73              }
74              else if (key.Key == ConsoleKey.Enter)
75              {
76                  if (clear) _menuInstance.ClearUserSection();
77                  Console.CursorVisible = false;
78
79                  selected = true;
80              }
81          }
```

```
82
83          return currentTop - 3;
84      }
85
86      public void WaitInput(string prompt)
87      {
88          while (Console.KeyAvailable) Console.ReadKey(true);
89          _menuInstance.WriteLine(prompt);
90          bool complete = false;
91
92          while (!complete)
93          {
94              if (!Console.KeyAvailable) continue;
95              ConsoleKeyInfo key = Console.ReadKey(true);
96              if (key.Key == ConsoleKey.Enter) complete = true;
97          }
98      }
99
100     public IEnumerable<(string, bool)> OptionSelector(string title, IEnumerable<(string, bool)> options, bool clear =
    ↪  true)
101     {
102         List<(string, bool)> result = new List<(string, bool)>(options);
103         result.Add(("EXIT", false));
104
105         while (Console.KeyAvailable) Console.ReadKey(true);
106         _menuInstance.ClearUserSection();
107         _menuInstance.WriteLine(title);
108
109         int j = 3;
110
111         lock (_menuInstance.ScreenLock)
112         {
113             foreach (var option in result)
114             {
115                 Console.SetCursorPosition(1, j++);
116                 if (option.Item2) Console.WriteLine($"  {option.Item1} [{Log.Green}x{Log.Blank}]");
117                 else Console.WriteLine($"  {option.Item1} [ ]");
118             }
119         }
120
121         bool selected = false;
122         int currentTop;
123
124         lock (_menuInstance.ScreenLock)
125         {
126             Console.SetCursorPosition(1, 3);
127             Console.Write('>');
128
129             currentTop = Console.CursorTop;
130         }
131
132         while (!selected)
133         {
134             Console.CursorVisible = false;
135
136             ConsoleKeyInfo key = Console.ReadKey(true);
137             if (key.Key == ConsoleKey.DownArrow && currentTop < result.Count() + 2)
138             {
139                 lock (_menuInstance.ScreenLock)
140                 {
```

```
                    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)
                {
                    Console.CursorLeft = 1;
                    Console.CursorTop = currentTop;
                    Console.Write(' ');
                    Console.CursorTop = --currentTop;
                    Console.CursorLeft = 1;
                    Console.Write('>');
                }
            }
            else if (key.Key == ConsoleKey.Enter || key.Key == ConsoleKey.Spacebar)
            {
                if (result.Count + 2 == currentTop)
                {
                    if (clear) _menuInstance.ClearUserSection();
                    Console.CursorVisible = false;

                    selected = true;
                }
                else
                {
                    result[currentTop - 3] = (result[currentTop - 3].Item1, !result[currentTop - 3].Item2);
                    Console.SetCursorPosition(1, currentTop);
                    if (result[currentTop - 3].Item2) Console.WriteLine($"> {result[currentTop - 3].Item1}
                    ↪  [{Log.Green}x{Log.Blank}]");
                    else Console.WriteLine($"> {result[currentTop - 3].Item1} [ ]");
                }
            }
        }

        return result;
    }


    public string GetInput(string prompt)
    {
        while (Console.KeyAvailable) Console.ReadKey(true);
        _menuInstance.WriteLine(prompt);

        bool complete = false;
        StringBuilder input = new StringBuilder();
        int line = _menuInstance.CurrentLine;

        while (!complete)
        {
            if (Console.KeyAvailable)
            {
                ConsoleKeyInfo key = Console.ReadKey(true);
                switch (key.Key)
                {
```

```
                    case ConsoleKey.Enter:
                        complete = true;
                        break;
                    case ConsoleKey.Backspace:
                    case ConsoleKey.Delete:
                        {
                            if (input.Length > 0)
                            {
                                lock (_menuInstance.ScreenLock)
                                {
                                    Console.SetCursorPosition((input.Length % (Console.WindowWidth * 3 / 4 - 1)), line);
                                    Console.Write(' ');
                                }

                                input.Remove(input.Length - 1, 1);
                            }

                            break;
                        }
                    default:
                        {
                            if (input.Length / (line - 1) > Console.WindowWidth * 3 / 4 - 2) line++;

                            lock (_menuInstance.ScreenLock)
                            {
                                Console.SetCursorPosition((input.Length % (Console.WindowWidth * 3 / 4 - 1)) + 1, line);
                                Console.Write(key.KeyChar);
                            }

                            input.Append(key.KeyChar);
                            break;
                        }
                }
            }
        }

        _menuInstance.WriteLine();

        return input.ToString();
    }

    public string TryGetInput(string prompt)
    {
        string res = GetInput(prompt);
        return res.Length == 0 ? "None" : res;
    }

    public double GetDouble(string prompt) => double.Parse(GetInput(prompt));

    public bool TryGetDouble(string prompt, out double result) => double.TryParse(GetInput(prompt), out result);

    public int GetInt(string prompt) => int.Parse(GetInput(prompt));

    public bool TryGetInt(string prompt, out int result) => int.TryParse(GetInput(prompt), out result);
}
```

Log.cs

```
public class Log
{
```

```
3       private int _logLineCount = 6;
4       private readonly Menu _menuInstance;
5
6       public const string Red = "\x1b[38;5;196m";
7       public const string Orange = "\x1b[38;5;184m";
8       public const string Purple = "\x1b[38;5;129m";
9       public const string Green = "\x1b[38;5;2m";
10      public const string Blue = "\x1b[38;5;27m";
11      public const string Pink = "\x1b[38;5;200m";
12      public const string Grey = "\x1b[38;5;243m";
13      public const string Blank = "\x1b[0m";
14
15      public void Error(string message) => Logger.WriteLineToMaster($"ERROR {message}");
16      public void Warn(string message) => Logger.WriteLineToMaster($"WARNING {message}");
17      public void Event(string message) => Logger.WriteLineToMaster($"EVENT {message}");
18      public void End(string message) => Logger.WriteLineToMaster($"END {message}");
19
20      public void Error(Guid runGuid, string message, bool detailed = false) => LogParent(runGuid, message, 0, detailed);
21      public void Warn(Guid runGuid, string message, bool detailed = false) => LogParent(runGuid, message, 1, detailed);
22      public void Event(Guid runGuid, string message, bool detailed = false) => LogParent(runGuid, message, 2, detailed);
23      public void End(Guid runGuid, string message, bool detailed = false) => LogParent(runGuid, message, 3, detailed);
24
25      public void EndError(Guid runGuid, Exception ex)
26      {
27          Error($"Run ({runGuid}) terminated due to an error.");
28          Error($"Exception: {ex.Message}");
29          if (ex.InnerException != null) Error($"Inner Exception: {ex.InnerException.Message}");
30          Error(runGuid, ex.Message);
31          End(runGuid, $"Run ({runGuid}) terminated.", true);
32      }
33
34      public void EndSuccessRun(Guid runGuid)
35      {
36          End(runGuid, "Successfully completed processing and pathfinding of new image!", true);
37          Warn(runGuid, $"Run Guid {runGuid} Deleted. See {Environment.CurrentDirectory}\\saves\\ for output(s) and
            ↪  {Environment.CurrentDirectory}\\runs\\{runGuid.ToString("N").ToUpper()} for temp images.", true);
38          End($"Completed run {runGuid} successfully.");
39      }
40
41      public void EndSuccessSave(Guid runGuid)
42      {
43          End(runGuid, "Successfully completed recall and pathfinding of save file!", true);
44          Warn(runGuid, $"Run Guid {runGuid} Deleted. See {Environment.CurrentDirectory}\\saves\\ for output(s). Or just
            ↪  go to where the save file was located.", true);
45          End($"Completed run {runGuid} successfully.");
46      }
47
48      public Log(Menu menuInstance)
49      {
50          _menuInstance = menuInstance;
51          _ = new Logger(true);
52      }
53
54      /// <summary>
55      ///
56      /// </summary>
57      /// <param name="message"></param>
58      /// <param name="type">0 - Error, 1 - Warning, 2 - Event, 3 - End</param>
59      private void LogParent(Guid runGuid, string message, int type, bool detailed)
60      {
```

```
61          if (!bool.Parse(Settings.UserSettings["detailedLogging"].Item1) && !detailed) return;
62
63          Console.CursorVisible = false;
64          string[] prefix = { $"{Red}ERROR{Log.Blank}", $"{Orange}WARN{Log.Blank}", $"{Green}EVENT{Log.Blank}",
            ↪  $"{Purple}END{Log.Blank}" };
65          string[] filePrefix = { "[ERROR] ", "[WARN] ", "[EVENT] ", "[END] " };
66
67          lock (_menuInstance.ScreenLock)
68          {
69              CheckLogLineCount();
70
71              if (message.Length > Console.WindowWidth / 4 - 7)
72              {
73                  Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 2, _logLineCount++);
74                  int i = 10;
75
76                  Console.Write($"{prefix[type]}: ");
77
78                  foreach (char letter in message)
79                  {
80                      Console.Write(letter);
81                      i++;
82                      if (i > Console.WindowWidth / 4)
83                      {
84                          if (CheckLogLineCount()) return;
85                          Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 9, _logLineCount++);
86                          i = 10;
87                      }
88                  }
89              }
90              else
91              {
92                  Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 2, _logLineCount++);
93                  Console.Write($"{prefix[type]}: {message}");
94              }
95          }
96
97          Logger.WriteLineToRunFile(runGuid, $"{filePrefix[type]}{message}");
98      }
99
100     // Make sure that the total log lines does not exceed the space given
101     private bool CheckLogLineCount()
102     {
103         if (_logLineCount >= Console.WindowHeight)
104         {
105             _logLineCount = 6;
106             _menuInstance.ClearLogSection();
107
108             return true;
109         }
110
111         return false;
112     }
113 }
```

Menu.cs

```
1   public class Menu
2   {
3       public object ScreenLock { get; } = new object();
```

```
4       public int CurrentLine { get; private set; } = 1;
5
6       [DllImport("kernel32.dll", SetLastError = true)]
7       private static extern bool SetConsoleMode(IntPtr hConsoleHandle, int mode);
8       [DllImport("kernel32.dll", SetLastError = true)]
9       private static extern bool GetConsoleMode(IntPtr handle, out int mode);
10      [DllImport("kernel32.dll", SetLastError = true)]
11      private static extern IntPtr GetStdHandle(int handle);
12
13      public bool IsWindowMax() => Console.WindowHeight >= Console.LargestWindowHeight && Console.WindowWidth >=
        ↪  Console.LargestWindowWidth - 3;
14
15      private readonly string _permLineA;
16      private readonly string _permLineB;
17
18      public const char VerticalChar = '';
19      public const char HorizontalChar = '';
20
21      public Menu(string permLineA, string permLineB)
22      {
23          IntPtr handle = GetStdHandle(-11);
24          GetConsoleMode(handle, out var mode);
25          SetConsoleMode(handle, mode | 0x4);
26
27          int width = Console.WindowWidth / 2;
28          int height = Console.WindowHeight / 4;
29          Console.SetWindowSize(width, height);
30          Console.SetBufferSize(width, height);
31
32          _permLineA = permLineA;
33          _permLineB = permLineB;
34
35          Console.Clear();
36          Console.CursorVisible = false;
37      }
38
39      public void Setup()
40      {
41          while (!IsWindowMax())
42          {
43              Console.SetCursorPosition(0, 0);
44              Console.WriteLine($"{Log.Red}Maximize Window To Continue{Log.Blank}");
45              System.Threading.Thread.Sleep(250);
46              Console.SetCursorPosition(0, 0);
47              Console.WriteLine($"\x1b[48;5;196mMaximize Window To Continue{Log.Blank}");
48              System.Threading.Thread.Sleep(250);
49
50          }
51
52          Console.Clear();
53
54          DisplayInfoBox();
55          DisplayLogBox();
56
57          Console.SetCursorPosition(0, 0);
58          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)
```

```
    {
        Console.CursorVisible = false;

        if (message.Length > Console.WindowWidth * 3 / 4)
        {
            int maxLength = Console.WindowWidth * 3 / 4;

            List<string> words = message.Split(' ').ToList();
            StringBuilder sb = new StringBuilder();

            foreach (string word in words)
            {
                if ($"{sb} {word}".Length > maxLength)
                {
                    WriteLine(sb.ToString());
                    sb.Remove(0, sb.Length);
                }
                else
                {
                    sb.Append($"{word} ");
                }
            }

            WriteLine(sb.ToString());
        }
        else
        {
            lock (ScreenLock)
            {
                if (CurrentLine > Console.WindowHeight * 5 / 6) ClearUserSection();

                Console.SetCursorPosition(1, CurrentLine++);
                Console.Write(message);
            }
        }
    }

}
```

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('+');
```

```
20
21          _progressOutline = bar.ToString();
22          _progressLine = "";
23          _progressTitle = title;
24          _menuInstance = menuInstance;
25      }
26
27      public void DisplayProgress()
28      {
29          int middle = Console.WindowHeight * 5 / 12;
30
31          lock (_menuInstance.ScreenLock)
32          {
33              Console.SetCursorPosition((Console.WindowWidth * 3 / 8) - (_progressTitle.Length / 2), middle - 3);
34              Console.Write(_progressTitle);
35
36              Console.SetCursorPosition(1, middle - 1);
37              Console.Write(_progressOutline);
38              Console.SetCursorPosition(1, middle);
39              Console.Write(Menu.VerticalChar);
40              Console.SetCursorPosition(Console.WindowWidth * 3 / 4 - 2, middle);
41              Console.Write(Menu.VerticalChar);
42              Console.SetCursorPosition(1, middle + 1);
43              Console.Write(_progressOutline);
44          }
45      }
46
47      public Action GetIncrementAction() => new Action(IncrementProgress);
48
49      private void IncrementProgress()
50      {
51          lock (_menuInstance.ScreenLock)
52          {
53              _progressAmount = _progressAmount + _progressInterval > 1 ? 1 : _progressAmount + _progressInterval;
54
55              int middle = Console.WindowHeight * 5 / 12;
56              double possibleLength = (Console.WindowWidth * 3 / 4) - 4;
57              possibleLength *= _progressAmount;
58
59              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

```
1   public class Settings
2   {
3       private readonly Menu _menuInstance;
4       private readonly Log _loggerInstance;
5
6       private List<string> rawLines;
```

```
7       public static Dictionary<string, (string, Type)> UserSettings { get; private set; }
8
9       private readonly string[] defaultSettings = {
10          "# Manually Edit At Own Risk",
11          "# General Settings",
12          "detailedLogging=false",
13          "forceFormsFront=true",
14          "",
15          "# Pathfinding Settings",
16          "convertToMST=false",
17          "pathfindingAlgorithm=AStar",
18          "snapToGrid=true",
19          "endOnFind=false",
20          "",
21          "# Save Settings",
22          "shortNames=false",
23          "zipOnComplete=false",
24      };
25
26      public Settings(Menu menu, Log log)
27      {
28          _menuInstance = menu;
29          _loggerInstance = log;
30      }
31
32      public void CheckIfExistsOrCreate()
33      {
34          if (!File.Exists("settings.conf"))
35          {
36              _loggerInstance.Event("Settings file did not exist. Creating...");
37              using (TextWriter tw = File.CreateText("settings.conf"))
38              {
39                  foreach (string line in defaultSettings)
40                  {
41                      tw.WriteLine(line);
42                  }
43              }
44          }
45      }
46
47      public List<string> ParseSettingsFile()
48      {
49          List<string> lines = new List<string>();
50          using (StreamReader sr = File.OpenText("settings.conf"))
51          {
52              while (!sr.EndOfStream)
53              {
54                  lines.Add(sr.ReadLine());
55              }
56          }
57
58          rawLines = lines;
59
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;
```

```
    }

    private Dictionary<string, (string, Type)> ConvertSettingsToPairs(List<string> parsedLines)
    {
        Dictionary<string, (string, Type)> pairs = new Dictionary<string, (string, Type)>();
        foreach (string item in parsedLines)
        {
            string name = item.Split('=')[0].Trim();
            string value = item.Split('=')[1].Trim();
            if (bool.TryParse(value, out bool _)) pairs.Add(name, (value, typeof(bool)));
            else if (int.TryParse(value, out int _)) pairs.Add(name, (value, typeof(int)));
            else if (double.TryParse(value, out double _)) pairs.Add(name, (value, typeof(double)));
            else pairs.Add(name, (value, typeof(string)));
        }

        return pairs;
    }

    public bool Change(string setting, bool value)
    {
        if (!UserSettings.ContainsKey(setting)) return false;
        UserSettings[setting] = (value.ToString().ToLower(), typeof(bool));

        return true;
    }

    public bool Change(string setting, int value)
    {
        if (!UserSettings.ContainsKey(setting)) return false;
        UserSettings[setting] = (value.ToString(), typeof(int));

        return true;
    }

    public bool Change(string setting, double value)
    {
        if (!UserSettings.ContainsKey(setting)) return false;
        UserSettings[setting] = (value.ToString(), typeof(double));

        return true;
    }

    public bool Change(string setting, string value)
    {
        if (!UserSettings.ContainsKey(setting)) return false;
        UserSettings[setting] = (value.ToString(), typeof(string));

        return true;
    }

    public void Read()
    {
        CheckIfExistsOrCreate();
        List<string> parsedLines = ParseSettingsFile();
        Dictionary<string, (string, Type)> settingValuePairs = ConvertSettingsToPairs(parsedLines);
        UserSettings = settingValuePairs;
    }

    public void Update(Dictionary<string, (string, Type)> oldSettings, Dictionary<string, (string, Type)> newSettings)
    {
```

```
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
```

5.2.2.3 Processes

AsyncEdgeDetection.cs

```
1   public class AsyncEdgeDetection : IHandler
2   {
3       private readonly Menu _menuInstance;
4       private readonly Log _logInstance;
5       private readonly Guid _runGuid;
6       private readonly Structures.RawImage _image;
7       private double[,] _resultArray;
8
9       public AsyncEdgeDetection(Menu menu, Log log, Structures.RawImage image, Guid currentGuid)
10      {
11          _menuInstance = menu;
12          _logInstance = log;
13          _image = image;
14          _runGuid = currentGuid;
15      }
16
17      public void Preset(int kernelSize, double redRatio, double greenRatio, double blueRatio, double sigma, double
        ↪  lowerThreshold, double upperThreshold, int loopCount)
18      {
19          _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})");
20
21          CannyEdgeDetection detector = new CannyEdgeDetection(kernelSize, redRatio, greenRatio, blueRatio, sigma,
22              lowerThreshold, upperThreshold);
23
24          Input inputHandel = new Input(_menuInstance);
25
26          Structures.RGB[][,] quads = Utility.SplitImage(_image.Pixels);
27          Task<double[,]>[] threads = new Task<double[,]>[quads.Length];
28
29          int continueOption = inputHandel.GetOption("Continue to Canny Edge Detection:", new[] { "Yes", "No" });
30          if (continueOption != 0) throw new Exception("You asked for the processing of your map to stop.");
31
32          bool saveTempOption = inputHandel.GetOption("Would you like to save images at each step of the edge detection?",
            ↪  new[] { "Yes", "No" }) == 0;
```

```
33
34          ProgressBar pb = new ProgressBar("Canny Edge Detection", 36, _menuInstance);
35          pb.DisplayProgress();
36
37          for (int i = 0; i < quads.Length; i++)
38          {
39              // Overcome Capture Condition
40              int copyI = i;
41              Task<double[,]> task = new Task<double[,]>(() => RunDetectionOnQuadrant(detector, quads[copyI], copyI,
                ↪  pb.GetIncrementAction(), saveTempOption));
42              task.Start();
43              threads[i] = task;
44          }
45
46          Task.WaitAll(threads);
47          double[,] cannyImage = Utility.CombineQuadrants(threads[0].Result, threads[1].Result, threads[2].Result,
            threads[3].Result);
48
49          Post postProcessor = new Post(cannyImage);
50          postProcessor.Start(loopCount);
51          _resultArray = postProcessor.Result();
52      }
53
54      public void Start()
55      {
56          Input inputHandel = new Input(_menuInstance);
57
58          _logInstance.Event(_runGuid, "Started Multi Threaded Canny Edge Detection");
59          bool confirmOptions = false;
60          CannyEdgeDetection detector;
61
62          _logInstance.Event(_runGuid, "Getting Multi Thread Options");
63
64          do
65          {
66              detector = GetDetector(_menuInstance, inputHandel, _logInstance);
67
68              string opt = inputHandel.GetInput("Are you happy with those edge detection variables (y/n): ");
69              if (opt.ToLower() == "y") confirmOptions = true;
70              else _menuInstance.ClearUserSection();
71          } while (!confirmOptions);
72
73
74          Structures.RGB[][,] quads = Utility.SplitImage(_image.Pixels);
75          Task<double[,]>[] threads = new Task<double[,]>[quads.Length];
76
77          int continueOption = inputHandel.GetOption("Continue to Canny Edge Detection:", new[] { "Yes - Continue", "No -
            ↪  Return to main menu" });
78          if (continueOption != 0) throw new Exception("You asked for the processing of your map to stop.");
79
80          bool saveTempOption = inputHandel.GetOption("Would you like to save images at each step of the edge detection?",
            ↪  new[] { "Yes", "No" }) == 0;
81
82          ProgressBar pb = new ProgressBar("Canny Edge Detection", 36, _menuInstance);
83          pb.DisplayProgress();
84
85          for (int i = 0; i < quads.Length; i++)
86          {
87              // Overcome Capture Condition
88              int copyI = i;
89
```

```
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,
                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;
```

```
        }
        else m.WriteLine($"{Log.Orange}Kept Default: {cannyDetection.KernelSize}{Log.Blank}");
        m.WriteLine();

        if (i.TryGetDouble(
                    $"Enter a value for the lower threshold for the Min Max stage (Default:
                    ↪  {cannyDetection.LowerThreshold}, Range: 0 <= x < 1)",
                out double newLowerThreshold) && newLowerThreshold > 0 && newLowerThreshold < 1 && newLowerThreshold !=
                ↪  cannyDetection.LowerThreshold)
        {
            l.Warn(_runGuid, $"Changed lower threshold {cannyDetection.LowerThreshold} -> {newLowerThreshold}");
            m.WriteLine($"{Log.Green}Changed: {cannyDetection.LowerThreshold} -> {newLowerThreshold}{Log.Blank}");
            cannyDetection.LowerThreshold = newLowerThreshold;
        }
        else m.WriteLine($"{Log.Orange}Kept Default: {cannyDetection.LowerThreshold}{Log.Blank}");
        m.WriteLine();

        if (i.TryGetDouble(
                    $"Enter a value for the lower threshold for the Min Max stage (Default:
                    ↪  {cannyDetection.UpperThreshold}, Range: {cannyDetection.LowerThreshold} < x <= 1)",
                out double newHigherThreshold) && newHigherThreshold > cannyDetection.LowerThreshold &&
                ↪  newHigherThreshold <= 1 && newHigherThreshold != cannyDetection.UpperThreshold)
        {
            l.Warn(_runGuid, $"Changed upper threshold {cannyDetection.UpperThreshold} -> {newHigherThreshold}");
            m.WriteLine($"{Log.Green}Changed: {cannyDetection.UpperThreshold} -> {newHigherThreshold}{Log.Blank}");
            cannyDetection.UpperThreshold = newHigherThreshold;
        }
        else m.WriteLine($"{Log.Orange}Kept Default: {cannyDetection.UpperThreshold}{Log.Blank}");
        m.WriteLine();

        i.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
        m.ClearUserSection();

        m.WriteLine("For reference the variables which will be used are:");
        m.WriteLine($"    Red Ratio: {Log.Green}{cannyDetection.RedRatio}{Log.Blank}");
        m.WriteLine($"    Green Ratio: {Log.Green}{cannyDetection.GreenRatio}{Log.Blank}");
        m.WriteLine($"    Blue Ratio: {Log.Green}{cannyDetection.BlueRatio}{Log.Blank}");
        m.WriteLine($"    Gaussian Sigma Value: {Log.Green}{cannyDetection.Sigma}{Log.Blank}");
        m.WriteLine($"    Gaussian Kernel Size: {Log.Green}{cannyDetection.KernelSize}{Log.Blank}");
        m.WriteLine($"    Double Threshold Lower: {Log.Green}{cannyDetection.LowerThreshold}{Log.Blank}");
        m.WriteLine($"    Double Threshold Upper: {Log.Green}{cannyDetection.UpperThreshold}{Log.Blank}");
        m.WriteLine();

        return cannyDetection;
    }

    public double[,] Result() => _resultArray;
}
```

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($"./runs/{_runGuid.ToString("N").ToUpper()}", "*.*",
                ↪  SearchOption.AllDirectories);
                foreach (string newPath in files)
                {
                    File.Copy(newPath, newPath.Replace($"./runs/{_runGuid.ToString("N").ToUpper()}", "temp/images"));
                }

                File.Copy($"./logs/{_runGuid}.txt", "temp/log.txt");
                File.Copy($"./saves/{saveName}.vmap", "temp/map.vmap");
                ZipFile.CreateFromDirectory("temp", $"run-{_runGuid}.zip");
                Directory.Delete("temp", true);
            }
        }
    }

    private void InvertImage(Input inputHandel)
    {
        bool invert = Utility.IsYes(inputHandel.GetInput("Invert image (y/n)?"));
        if (invert)
        {
            _cannyEdgeDetectionResult = Utility.InverseImage(_cannyEdgeDetectionResult);
            ViewImageForm invertImageForm = new ViewImageForm(_cannyEdgeDetectionResult.ToBitmap());
            invertImageForm.ShowDialog();
            if (_saveFile != null) _saveFile.IsInverted = true;
        }
        if (_saveFile != null) _saveFile.IsInverted = false;

        _menuInstance.WriteLine();
    }

    private void DetectRoads(Input inputHandel)
    {
        bool happy = true;

        double threshold = 0.3;

        while (happy)
        {
            if (inputHandel.TryGetDouble(
                    $"Value for Threshold (Default: {threshold}, Range: 0 <= x < 1)",
                    out double newThreshold) && newThreshold > 0 && newThreshold < 1 && newThreshold != threshold)
            {
                _logInstance.Warn(_runGuid, $"Changed threshold {threshold} -> {newThreshold}");
                _menuInstance.WriteLine($"{Log.Green}Changed: {threshold} -> {newThreshold}{Log.Blank}");
```

```
                threshold = newThreshold;
            }
            else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {threshold}{Log.Blank}");
            _menuInstance.WriteLine();

            RoadDetection roadDetector = new RoadDetection(_cannyEdgeDetectionResult, threshold);
            ProgressBar pb = new ProgressBar("Road Detection", _cannyEdgeDetectionResult.Length / 100 * 3,
            ↪  _menuInstance);

            pb.DisplayProgress();
            roadDetector.Start(pb.GetIncrementAction());

            _roadResult = roadDetector.Result();
            ViewImageForm roadForm = new ViewImageForm(_roadResult.PathBitmap);
            roadForm.ShowDialog();

            _menuInstance.ClearUserSection();

            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;
        }
    }
}
```

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");
```

```
33          _menuInstance.WriteLine("   4. Gradient Directions");
34          _menuInstance.WriteLine("   5. Magnitude Threshold");
35          _menuInstance.WriteLine();
36          _classInputHandel.WaitInput($"{Log.Grey}(Enter to Continue){Log.Blank}");
37          _menuInstance.WriteLine("This may take some time to process each step.");
38
39          Structures.Gradients grads = _detector.CalculateGradients(_workingArray, () => { });
40          ViewImageForm gradXForm = new ViewImageForm(grads.GradientX.ToBitmap());
41          gradXForm.ShowDialog();
42
43          ViewImageForm gradYForm = new ViewImageForm(grads.GradientY.ToBitmap());
44          gradYForm.ShowDialog();
45
46          _workingArray = _detector.CombineGradients(grads);
47          ViewImageForm combinedGradientForm = new ViewImageForm(_workingArray.ToBitmap());
48          combinedGradientForm.ShowDialog();
49
50          double[,] gradientDirections = _detector.GradientAngle(grads);
51          double[,] gradCopy = gradientDirections;
52          for (int y = 0; y < gradientDirections.GetLength(0); y++)
53              for (int x = 0; x < gradientDirections.GetLength(1); x++)
54                  gradCopy[y, x] = Utility.MapRadiansToPixel(gradientDirections[y, x]);
55          ViewImageForm gradientDirectionForm = new ViewImageForm(gradCopy.ToBitmap());
56          gradientDirectionForm.ShowDialog();
57
58          _workingArray = _detector.MagnitudeThreshold(_workingArray, gradientDirections);
59          ViewImageForm magnitudeForm = new ViewImageForm(_workingArray.ToBitmap());
60          magnitudeForm.ShowDialog();
61
62          _menuInstance.ClearUserSection();
63
64          Structures.ThresholdPixel[,] _thresholdPixels = DoubleThresholdStep();
65
66          _menuInstance.WriteLine("From here on out stages are automated, however as before you will see each step after
            ↪  it occurs.");
67          _menuInstance.WriteLine();
68          _classInputHandel.WaitInput($"{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);
```

```
                    else toView.SetPixel(x, y, Color.Black);
                }
            }
            ViewImageForm gaussianForm = new ViewImageForm(toView);
            _menuInstance.ClearUserSection();
            gaussianForm.ShowDialog();

            _menuInstance.WriteLine("Current values for thresholds");
            _menuInstance.WriteLine($"Lower: {_detector.LowerThreshold}");
            _menuInstance.WriteLine($"Upper: {_detector.UpperThreshold}");
            _menuInstance.WriteLine();

            string opt = _classInputHandel.GetInput("Are you happy with these values for the upper and lower threshold
            ↪  (y/n)?");

            if (opt.ToLower().StartsWith("y")) happy = true;
            else
            {
                _menuInstance.ClearUserSection();
                _menuInstance.WriteLine($"{Log.Pink}Please re-enter your values.{Log.Blank}");
            }
        }

        _menuInstance.ClearUserSection();
        return _workingThresholdPixels;
    }

    private void GaussianStep()
    {
        bool happy = false;

        _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.");

        while (!happy)
        {
            if (_classInputHandel.TryGetDouble(
                    $"Value for Sigma (Default: {_detector.Sigma}, Range: 0 < x <= 10)",
                    out double newSigma) && newSigma <= 10 && newSigma > 0 && newSigma != _detector.Sigma)
            {
                _logInstance.Warn(_runGuid, $"Changed Sigma value {_detector.Sigma} -> {newSigma}");
                _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.Sigma} -> {newSigma}{Log.Blank}");
                _detector.Sigma = newSigma;
            }
            else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.Sigma}{Log.Blank}");
            _menuInstance.WriteLine();

            if (_classInputHandel.TryGetInt(
                    $"Value for Kernel Size (Default: {_detector.KernelSize}, Range: x >= 3, x not a multiple of 2 and a
                    ↪  whole number)",
                    out int newKernel) && newKernel >= 3 && newKernel % 2 == 1 && newKernel % 1 == 0 && newKernel !=
                    ↪  _detector.KernelSize)
            {
                _logInstance.Warn(_runGuid, $"Changed Kernel Size {_detector.KernelSize} -> {newKernel}");
                _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.KernelSize} -> {newKernel}{Log.Blank}");
                _detector.KernelSize = newKernel;
            }
            else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.KernelSize}{Log.Blank}");
            _menuInstance.WriteLine();
            _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(
```

```
                        $"Value for {Log.Blue}Blue{Log.Blank} (Old: {_detector.BlueRatio}, Range: 0 <= x <= 1)",
                    out double newBlueRatio) && newBlueRatio <= 1 && newBlueRatio >= 0 && newBlueRatio !=
                    ↪ _detector.BlueRatio)
            {
                _logInstance.Warn(_runGuid, $"Changed {Log.Blue}Blue{Log.Blank} ratio {_detector.BlueRatio} ->
                ↪ {newBlueRatio}");
                _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.BlueRatio} -> {newBlueRatio}{Log.Blank}");
                _detector.BlueRatio = newBlueRatio;
            }
            else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.BlueRatio}{Log.Blank}");
            _menuInstance.WriteLine();
            _menuInstance.WriteLine("Converting to black and white. This may take some time...");

            _workingArray = _detector.BlackWhiteFilter(_image.Pixels);
            ViewImageForm blackWhiteForm = new ViewImageForm(_workingArray.ToBitmap());
            _menuInstance.ClearUserSection();
            blackWhiteForm.ShowDialog();

            _menuInstance.WriteLine("Current values for ratios");
            _menuInstance.WriteLine($"Red: {Log.Red}{_detector.RedRatio}{Log.Blank}");
            _menuInstance.WriteLine($"Green: {Log.Green}{_detector.GreenRatio}{Log.Blank}");
            _menuInstance.WriteLine($"Blue: {Log.Blue}{_detector.BlueRatio}{Log.Blank}");
            _menuInstance.WriteLine();

            string opt = _classInputHandel.GetInput("Are you happy with these values and the result (y/n)?");

            if (opt.ToLower().StartsWith("y")) happy = true;
            else
            {
                _menuInstance.ClearUserSection();
                _menuInstance.WriteLine($"{Log.Pink}Please re-enter your values.{Log.Blank}");
            }
        }
        _menuInstance.ClearUserSection();
    }

    private void ShowDialog()
    {
        _menuInstance.ClearUserSection();
        _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.");
        _classInputHandel.WaitInput($"{Log.Grey}(Enter to Continue){Log.Blank}");
        _menuInstance.WriteLine();
    }

    public double[,] Result() => _resultArray;
}
```

5.2.2.4 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;
```

```
9
10      private readonly Graph<Structures.Coord> _graph;
11
12      private readonly Traversal<Structures.Coord> _traversalObject;
13
14      private Structures.Coord prevStartNode;
15      private Structures.Coord startNode = invalidCord;
16      private Structures.Coord endNode = invalidCord;
17
18      private Dictionary<Structures.Coord, Structures.Coord> _preCalculatedTree;
19
20      public PathfindImageForm(Bitmap image, Traversal<Structures.Coord> traversal, Graph<Structures.Coord> graph)
21      {
22          _image = image;
23          _originalImage = image;
24          _traversalObject = traversal;
25          _graph = graph;
26
27          InitializeComponent();
28      }
29
30      private void ViewImageForm_Load(object sender, EventArgs e)
31      {
32          // Define size
33          _width = Console.WindowWidth * 3 / 4 * 8;
34          _height = Console.WindowHeight * 5 / 6 * 16;
35
36
37          // Styling
38          ControlBox = false;
39          FormBorderStyle = FormBorderStyle.None;
40          Text = "Pathfinding Window";
41
42          // set window to size of user area
43          MinimumSize = new Size(_width, _height);
44          MaximumSize = new Size(_width, _height);
45
46          // account for window bar
47          Location = new Point(0, 25);
48
49          // Always on top
50          if (bool.Parse(Settings.UserSettings["forceFormsFront"].Item1)) TopMost = true;
51
52          // set picture frame
53          imageBox.Width = _width * 2 / 3 - 12;
54          imageBox.Height = _height - 24;
55          imageBox.SizeMode = PictureBoxSizeMode.StretchImage;
56          imageBox.Image = _image;
57
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)

```
1  public partial class ViewImageForm : Form
2  {
3      private readonly Bitmap _image;
4      private int _width;
5      private int _height;
6      public ViewImageForm(Bitmap image)
7      {
8          this._image = image;
9          InitializeComponent();
10     }
11
12     private void ViewImageForm_Load(object sender, System.EventArgs e)
13     {
14         // Define size
15         _width = Console.WindowWidth * 3 / 4 * 8;
16         _height = Console.WindowHeight * 5 / 6 * 16;
17
18         // Styling
19         ControlBox = false;
20         FormBorderStyle = FormBorderStyle.None;
21         Text = "Preview Window";
22
23         // set window to size of user area
24         MinimumSize = new Size(_width, _height);
25         MaximumSize = new Size(_width, _height);
26
27         // account for window bar
28         Location = new Point(0, 25);
29
30         // Always on top
31         if (bool.Parse(Settings.UserSettings["forceFormsFront"].Item1)) TopMost = true;
32
33         // set picture frame
34         imageBox.Width = _width * 2 / 3 - 12;
35         imageBox.Height = _height - 24;
36         imageBox.SizeMode = PictureBoxSizeMode.StretchImage;
37         imageBox.Image = _image;
38
39         nextButton.Width = _width / 3 - 24;
40         nextButton.Height = _height - 24;
41         nextButton.Left = _width * 2 / 3 + 12;
42     }
43
44     private void nextButton_Click(object sender, System.EventArgs e)
45     {
46         Close();
47     }
48 }
```

5.2.2.5 Root

Program.cs

```
1  public class Program
2  {
```

```
3       private static void Main()
4       {
5           Menu menu = new Menu("Author: Rubens Pirie", $"{Log.Grey}Production Mode{Log.Blank}");
6           Log logger = new Log(menu);
7
8           Settings settings = new Settings(menu, logger);
9           settings.Read();
10
11          menu.Setup();
12          logger.Event("Program has started and menu has been created successfully.");
13
14          Run(menu, logger, settings);
15      }
16
17      private static void Run(Menu menuInstance, Log CLILoggingInstance, Settings settingsInstance)
18      {
19          Input inputHandel = new Input(menuInstance);
20
21          bool running = true;
22
23          while (running)
24          {
25              menuInstance.SetPage("Welcome Menu");
26              int opt = inputHandel.GetOption("Please select an option to continue:",
27                  new[]
28                  {
29                      "Process New Image Into Map Data File", "Recall Map From Data File", "Settings", "Exit Program"
30                  });
31
32              switch (opt)
33              {
34                  // New
35                  case 0:
36                      menuInstance.SetPage("Process New Image");
37                      TextWall.ImageWelcome(menuInstance);
38                      inputHandel.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
39                      menuInstance.WriteLine();
40
41                      RunNewImage(menuInstance, CLILoggingInstance);
42                      break;
43                  // Recall
44                  case 1:
45                      menuInstance.SetPage("Recall Old Image");
46                      TextWall.SaveWelcome(menuInstance);
47                      inputHandel.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
48                      menuInstance.WriteLine();
49
50                      RunSaveFile(menuInstance, CLILoggingInstance);
51                      break;
52                  // Settings
53                  case 2:
54                      try
55                      {
56                          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 }
```