

# **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

<b>1 Analysis</b>	<b>4</b>
1.1 Statement Of Problem . . . . .	4
1.2 Background . . . . .	4
1.3 End User . . . . .	5
1.3.1 First Interview . . . . .	5
1.3.2 Evaluation of First Interview . . . . .	6
1.4 Initial Research . . . . .	6
1.4.1 Existing Solutions . . . . .	6
Google Maps . . . . .	6
Bing Maps . . . . .	7
OS Maps . . . . .	7
Existing Solutions Conclusion . . . . .	8
1.4.2 Possible Algorithmic Solutions . . . . .	8
Edge Detection . . . . .	8
Graph Forming . . . . .	9
1.4.3 Key Components Required . . . . .	9
The Graphical User Interface . . . . .	9
Image Manipulation and Edge Detection . . . . .	10
Graph Creation and Representation . . . . .	10
Graph Traversal and Output . . . . .	10
1.5 Further Research . . . . .	10
1.5.1 Dive into Specific Algorithms . . . . .	10
Black and White Filter . . . . .	10
Gaussian Filter . . . . .	11
Convolution Operation . . . . .	11
1.5.2 Second Interview . . . . .	12
1.5.3 Evaluation of Second Interview . . . . .	13
1.6 Prototyping . . . . .	13
1.6.1 Prototype Objectives . . . . .	13
1.6.2 Edge Detection . . . . .	13
1. Converting to Black and White . . . . .	14
2. Gaussian Filter . . . . .	15
3. Calculation of XY Gradients . . . . .	17
4. Gradient Direction . . . . .	18
5. Gradient Magnitude Threshold . . . . .	19
6. Min Max Threshold and Potential Edge Calculations . . . . .	22
7. Edge tracking by Hysteresis . . . . .	23
8. Emboss Kernel . . . . .	24
9. Custom Hole Filling . . . . .	25
1.6.3 Graph Class and Graph Traversal . . . . .	26
1.6.4 Windows Forms with Images . . . . .	30
1.7 Objectives . . . . .	33
1.8 Modelling . . . . .	35
<b>2 Technical Design</b>	<b>36</b>
2.1 Programming Language Selection and Libraries Used . . . . .	36
<b>3 Program Testing</b>	<b>37</b>
3.1 Testing Tables . . . . .	37
3.1.1 Targeted Testing Areas . . . . .	37
3.1.2 User Inputs and Outputs Testing Table . . . . .	38
3.1.3 Canny Edge Detection Testing Table . . . . .	40
3.1.4 Road Detection and Graph Conversion Testing Table . . . . .	42
3.1.5 Graph Traversal Testing Table . . . . .	44

3.1.6	Logging and Saves Testing Table . . . . .	45
3.1.7	Miscellaneous Testing Table . . . . .	47
3.2	Testing Video . . . . .	50
<b>4</b>	<b>Evaluation</b>	<b>51</b>
<b>5</b>	<b>Code Base</b>	<b>52</b>
5.1	Prototypes . . . . .	52
5.1.1	Canny Edge Detection . . . . .	52
5.1.2	Graph Class and DFS / BFS . . . . .	62
5.1.3	Forms Interface . . . . .	65
5.2	Final Solution . . . . .	65
5.2.1	BackendLib . . . . .	65
5.2.1.1	Data . . . . .	65
	MapFile.cs . . . . .	65
	Traversals.cs . . . . .	68
5.2.1.2	Datatypes . . . . .	70
	Graph.cs . . . . .	70
	Matrix.cs . . . . .	71
	MaxPriorityQueue.cs . . . . .	73
	MinPriorityQueue.cs . . . . .	74
	Queue.cs . . . . .	76
	Stack.cs . . . . .	77
5.2.1.3	Exceptions . . . . .	77
	GraphException.cs . . . . .	77
	KernelException.cs . . . . .	78
	LoggerException.cs . . . . .	78
	MapFileException.cs . . . . .	78
	MatrixException.cs . . . . .	79
	PreprocessingException.cs . . . . .	79
	SettingsException.cs . . . . .	79
5.2.1.4	Interfaces . . . . .	80
	IHandler.cs . . . . .	80
5.2.1.5	Processing . . . . .	80
	CannyEdgeDetection.cs . . . . .	80
	Post.cs . . . . .	84
	Pre.cs . . . . .	86
	RoadDetection.cs . . . . .	87
5.2.1.6	Root . . . . .	89
	Extensions.cs . . . . .	89
	Kernel.cs . . . . .	91
	Logger.cs . . . . .	93
	Structures.cs . . . . .	94
	Utility.cs . . . . .	95
5.2.2	LocalApp . . . . .	98
5.2.2.1	Actions . . . . .	98
	NewImage.cs . . . . .	98
	SaveFile.cs . . . . .	99
	SettingsControl.cs . . . . .	100
5.2.2.2	CLI . . . . .	102
	Input.cs . . . . .	102
	Log.cs . . . . .	106
	Menu.cs . . . . .	108
	ProgressBar.cs . . . . .	113
	Settings.cs . . . . .	114
	TextWall.cs . . . . .	117
5.2.2.3	Processes . . . . .	118

AsyncEdgeDetection.cs . . . . .	118
Pathfinder.cs . . . . .	123
RoadSquence.cs . . . . .	124
SyncEdgeDetection.cs . . . . .	126
5.2.2.4 WindowsForms . . . . .	131
PathfindImageForm.cs (Partial) . . . . .	131
ViewImageForm.cs (Partial) . . . . .	136
5.2.2.5 Root . . . . .	137
Program.cs . . . . .	137

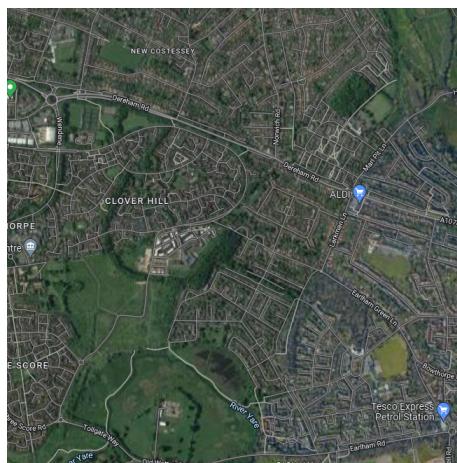
# 1 Analysis

## 1.1 Statement Of Problem

Maps, as you would think of them today, have been around since 6<sup>th</sup> 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. Therefore, 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<sup>©</sup>) 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.

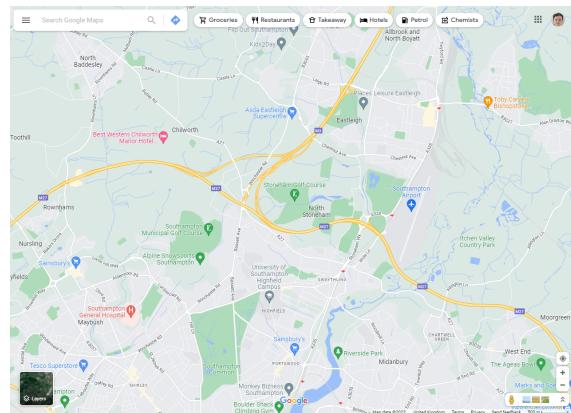
**1.4 Initial Research****1.4.1 Existing Solutions**

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 8<sup>th</sup> 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.



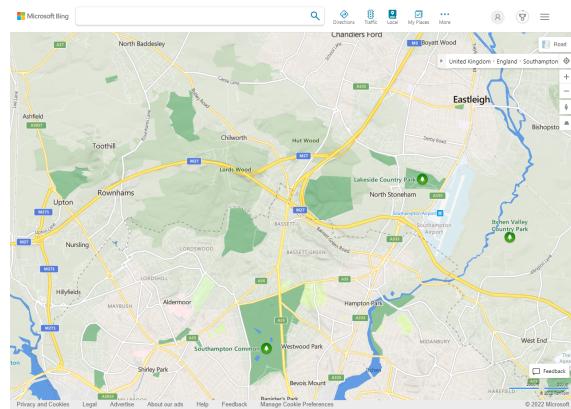
(a) Example of Google Maps' GUI

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

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



(b) Example of Bing Maps' GUI

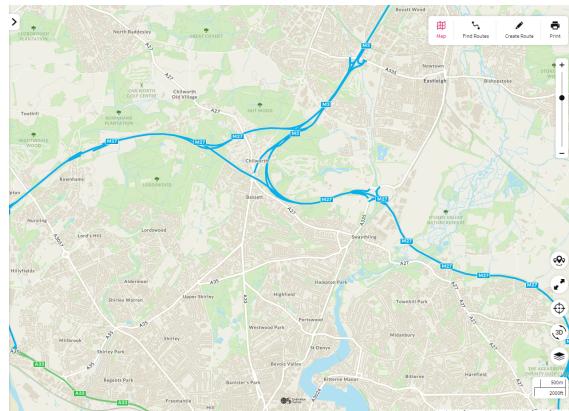
Sourced from Bing Maps®

### 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 therefore for the average user this is not a viable option and a hindrance. It is good to note however that the other variations on the map of the UK, and this holds true for all of the aforementioned maps, that the satellite view and other views are not necessary and could in fact be a hindrance.



(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 were 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 were 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.

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

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  $\sigma$  which denotes the amount of blur to apply and  $k$  is the kernel size. As well as being one of the key steps in canny edge detection it is also a vital component to most edge detection programs since noise can cause errors in the final image.

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

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

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

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

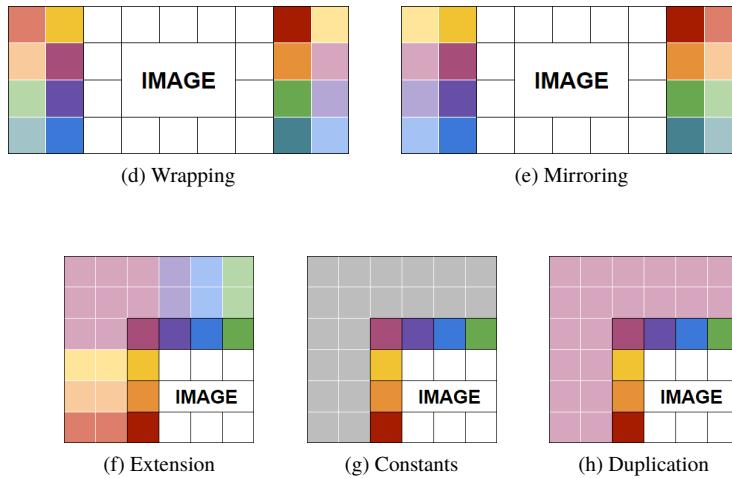
- A version of edge detection
- A graph class with basic traversal
- A forms interface for showing images

### 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 if 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.



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



Figure 1: Original Image

### 1. Converting to Black and White

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

The implementation in code of this is as below:

```

1 public double[,] BWFilter(Bitmap image)
2 {
3     double[,] result = new double[image.Height, image.Width];
4
5     for (int i = 0; i < image.Height; i++)
6     {
7         for (int j = 0; j < image.Width; j++)
8         {
9             int r = image.GetPixel(j, i).R;
10            int g = image.GetPixel(j, i).G;
11            int b = image.GetPixel(j, i).B;
12
13            double y = 0.299 * r + 0.587 * g + 0.114 * b;
14            double u = -0.169 * r - 0.331 * g + 0.5 * b;
15            double v = 0.5 * r - 0.419 * g - 0.081 * b;
16
17            result[i, j] = y;
18        }
19    }
20 }
```

```

8
9     Color c = image.GetPixel(j, i);
10    double value = c.R * 0.299 + c.G * 0.587 + c.B * 0.114;
11
12    result[i, j] = Bound(0, 255, value);
13}
14}
15
16 return result;
17}

```

This takes the original image in Bitmap form and then instantiates an array with the dimensions of the input image, this will serve going forward as the array as to which all changes will be based from. I learnt from this prototype early on that when calculating the values it is better to use the exact ones from the previous stage. This is because if all the values were compressed to within image specifications ( $0 \leq x \leq 255$ ) you would lose definition and precision causing later calculation to be incorrect. Once this section has run through every pixel in the image and converted it to a black and white value the subroutine returns the double array with the black and white values. The result of this on the input *figure 1* is:



Figure 2: Black and White Filter

## 2. Gaussian Filter

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

```

1 public double[,] GaussianFilter(double sigma, int kernelSize, double[,] imageArray)
2 {
3     double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];
4
5     Matrix gaussianKernel = GetGaussianKernel(kernelSize, sigma);
6
7     for (int i = 0; i < result.GetLength(0); i++)

```

```

8     {
9         for (int j = 0; j < result.GetLength(1); j++)
10        {
11            Matrix imageKernel = BuildKernel(j, i, kernelSize, imageArray);
12            double sum = Matrix.Convolution(imageKernel, gaussianKernel);
13            result[i, j] = sum;
14        }
15    }
16
17    return result;
18 }
19
20 public Matrix GetGaussianKernel(int k, double sigma)
21 {
22     double[,] result = new double[k, k];
23     int halfK = k / 2;
24
25     double sum = 0;
26
27     int cntY = -halfK;
28     for (int i = 0; i < k; i++)
29     {
30         int cntX = -halfK;
31         for (int j = 0; j < k; j++)
32         {
33             result[halfK + cntY, halfK + cntX] = GetGaussianDistribution(cntX, cntY, sigma);
34             sum += result[halfK + cntY, halfK + cntX];
35             cntX++;
36         }
37         cntY++;
38     }
39
40     for (int i = 0; i < k; i++) for (int j = 0; j < k; j++) result[i, j] /= sum;
41     return new Matrix(result);
42 }

```

Again this subroutine follows a similar layout to the rest in this prototype, it iterates through 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*.

```

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

```

18     }
19
20     return result;
21 }
```

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



(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 travelling. This is done via the use of the 2<sup>nd</sup> argument arc-tangent. The definition of the 2<sup>nd</sup> argument arc-tangent is defined as the angle in the Euclidean plane, given in radians, between the positive  $x$  axis and the ray from the origin to the point  $(x, y)$ . Once this is calculated this will allow the program to see in which direction the gradient is travelling in the image. As well as this it 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<sup>nd</sup> 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++)
5         result[i, j] = Math.Atan2(gradY[i, j], gradX[i, j]);
6     return result;
}
```

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



Figure 5: Gaussian Filter

## 5. Gradient Magnitude Threshold

Once both the combined gradient and gradient directions have been calculated 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 <
5         thetaArray.GetLength(1); j++) result[i, j] = 180 * Math.Abs(thetaArray[i, j]) / Math.PI;
6     }

```

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



Figure 6: Gradient Direction

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

```

1 public double[,] ApplyGradientMagnitudeThreshold(double[,] angles, double[,] magnitudes)
2 {
3     double[,] result = magnitudes;
4     double[,] anglesInDegrees = ConvertThetaToDegrees(angles);
5
6     for (int i = 0; i < anglesInDegrees.GetLength(0); i++)
7     {
8         for (int j = 0; j < anglesInDegrees.GetLength(1); j++)
9         {
10             double[,] magnitudeKernel = BuildKernel(j, i, 3, magnitudes).matrix;
11
12             if (anglesInDegrees[i, j] < 22.5 || anglesInDegrees[i, j] >= 157.5)
13             {
14                 if (magnitudes[i, j] < magnitudeKernel[1, 2] || magnitudes[i, j] <
15                     magnitudeKernel[1, 0])
16                 {

```

```

16             result[i, j] = 0;
17         }
18     }
19     else if (anglesInDegrees[i, j] >= 22.5 && anglesInDegrees[i, j] < 67.5)
20     {
21         if (magnitudes[i, j] < magnitudeKernel[0, 2] || magnitudes[i, j] <
→ magnitudeKernel[2, 0])
22         {
23             result[i, j] = 0;
24         }
25     }
26     else if (anglesInDegrees[i, j] >= 67.5 && anglesInDegrees[i, j] < 112.5)
27     {
28         if (magnitudes[i, j] < magnitudeKernel[0, 1] || magnitudes[i, j] <
→ magnitudeKernel[2, 1])
29         {
30             result[i, j] = 0;
31         }
32     }
33     else if (anglesInDegrees[i, j] >= 112.5 && anglesInDegrees[i, j] < 157.5)
34     {
35         if (magnitudes[i, j] < magnitudeKernel[0, 0] || magnitudes[i, j] <
→ magnitudeKernel[2, 2])
36         {
37             result[i, j] = 0;
38         }
39     }
40     else throw new Exception();
41 }
42 }
43
44 return result;
45 }
```

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



Figure 7: Magnitude Threshold

## 6. Min Max Threshold and Potential Edge Calculations

This part of the canny edge detection is also called the double threshold. This is where the image pixels will all be taken and their values considered. This is when it becomes necessary for us to use the black and white version of the image. If we did not then there would be no easy way to perform this. This is because unlike most of the 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.

```

1  public (double, bool)[,] ApplyDoubleThreshold(double l, double h, double[,] gradients)
2  {
3      double min = l * 255;
4      double max = h * 255;
5
6      (double, bool)[,] result = new (double, bool)[gradients.GetLength(0),
7      gradients.GetLength(1)];
8
9      for (int i = 0; i < gradients.GetLength(0); i++)
10     {
11         for (int j = 0; j < gradients.GetLength(1); j++)
12         {
13             if (gradients[i, j] < min) result[i, j] = (0, false);
14             else if (gradients[i, j] > min && gradients[i, j] < max) result[i, j] = (gradients[i,
15             j], false);
16             else if (gradients[i, j] > max) result[i, j] = (gradients[i, j], true);
17             else throw new Exception();
18         }
19     }
20
21     return result;
22 }
```

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 outright 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<sup>nd</sup> part of the tuple. And **false** for a "weak" pixel.

## 7. Edge tracking by Hysteresis

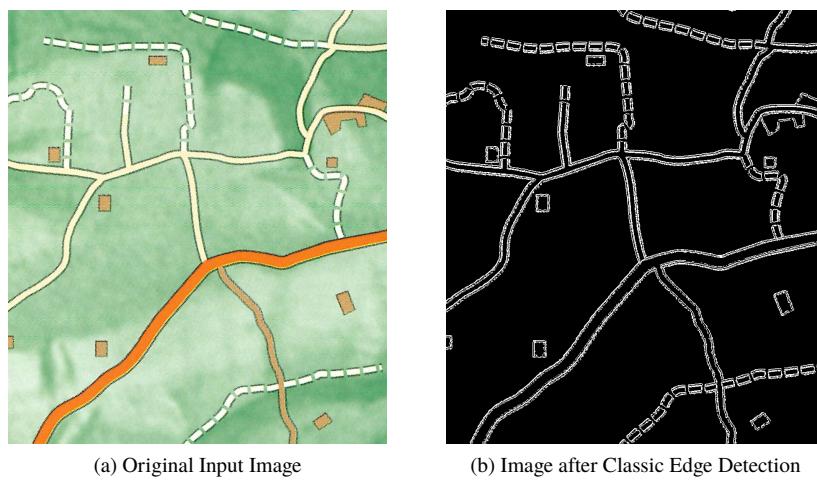
This is the final step of traditional canny edge detection. This will require the 2D array of tuples and will require kernels of the image as it loops over every pixel. This will cause a problem since the usual way of doing it would default to grey if the kernel overlapped with the edge of the image.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

```
1 public double[,] ApplyEdgeTrackingHysteresis((double, bool)[,] arrayOfValues)
2 {
3     double[,] result = new double[arrayOfValues.GetLength(0), arrayOfValues.GetLength(1)];
4
5     for (int i = 0; i < arrayOfValues.GetLength(0); i++)
6     {
7         for (int j = 0; j < arrayOfValues.GetLength(1); j++)
8         {
9             if (arrayOfValues[i, j].Item2 == false)
10             {
11                 (double, bool)[,] imageKernel = BuildKernel(j, i, 3, arrayOfValues);
12                 bool strong = false;
13                 for (int k = 0; k < 3 && !strong; k++)
14                 {
15                     for (int l = 0; l < 3 && !strong; l++)
16                     {
17                         if (imageKernel[k, l].Item2 == true) strong = true;
18                     }
19                 }
20             }
21         }
22     }
23 }
```

```

18         }
19     }
20
21     result[i, j] = strong ? 255 : 0;
22 }
23 else result[i, j] = 255;
24 }
25 }
26
27 return result;
28 }
```

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.



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.

```

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

```

3     double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];
4
5     Matrix embossMatrix = new Matrix(new double[,] [
6     {
7         { -2, -1, 0 },
8         { -1, 1, 1 },
9         { 0, 1, 2 },
10    });
11
12    for (int i = 0; i < imageArray.GetLength(0); i++)
13    {
14        for (int j = 0; j < imageArray.GetLength(1); j++)
15        {
16            Matrix imageKernel = BuildKernel(j, i, 3, imageArray);
17            result[i, j] = Math.Abs(Matrix.Convolution(imageKernel, embossMatrix));
18        }
19    }
20
21    return result;
22 }
```

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

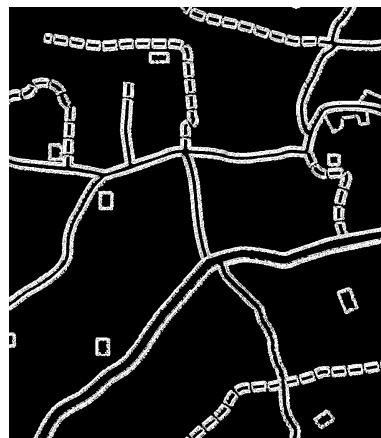


Figure 9: Magnitude Threshold

## 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  {
3      double[,] result = imageArray;
4
5      for (int i = 0; i < imageArray.GetLength(0); i++)
6      {
7          for (int j = 0; j < imageArray.GetLength(1); j++)
8          {
9              Matrix imageKernel = BuildKernel(j, i, 3, imageArray);
10             int count = 0;
11             foreach (double value in imageKernel.matrix)
12             {
13                 if (value >= 255) count++;
14             }
15
16             if (count > 4) result[i, j] = 255;
17         }
18     }
19
20     return result;
21 }
```

### 1.6.3 Graph Class and Graph Traversal

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

#### 1. Adjacency Matrix

- Advantages

Very fast when needing to lookup connections.

Inserting is also fast due to it being instantly accessible and not a dynamic structure.

- Disadvantages

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

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

## 2. Adjacency List

- Advantages

Easier to use pragmatically and implement

It is much easier to use Linq functions with to find graph connections

- Disadvantages

Relatively slower when it comes to accessing sections of the graph.

Would have to be a hybrid with a dictionary to allow for reasonable use

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

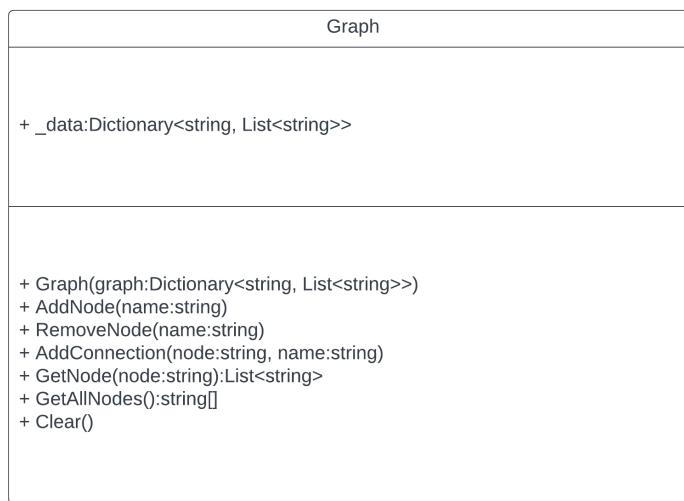


Figure 11: Graph UML Diagram

and in code

```

1  public class Graph
2  {
3      public Dictionary<string, List<string>> _data = new Dictionary<string, List<string>>();
4
5      public Graph(Dictionary<string, List<string>> graph)
6      {
  
```

```

7         _data = graph;
8     }
9
10    public void AddNode(string name)
11    {
12        if (_data.ContainsKey(name)) throw new GraphException($"Cannot add {name}, node already
13        exists.");
14        _data.Add(name, new List<string>());
15    }
16
17    public void RemoveNode(string name)
18    {
19        if (!_data.ContainsKey(name)) throw new GraphException($"Cannot remove {name}, node does
20        not exist.");
21        _data.Remove(name);
22    }
23
24    public void AddConnection(string node, string name)
25    {
26        if (!_data.ContainsKey(node)) throw new GraphException($"Cannot add connection {name} to
27        {node} original node does not exist.");
28        if (_data[node].Contains(name)) throw new GraphException($"Cannot add connection {name}
29        to {node} connection already exists.");
30        _data[node].Add(name);
31    }
32
33    public List<string> GetNode(string node)
34    {
35        if (!_data.ContainsKey(node)) throw new GraphException($"Node {node} does not exist.");
36        return _data[node];
37    }
38
39    public string[] GetAllNodes() => _data.Keys.ToArray();
40
41    public void Clear() => _data.Clear();
42 }
```

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        }
29    }
30
31
32    return path.ToArray();
33 }
34
35 public static string[] BFS(string start, Graph graph)
36 {
37    List<string> path = new List<string>();
38    Queue<string> stack = new Queue<string>();
39    Dictionary<string, bool> visited = new Dictionary<string, bool>();
40    foreach (string s in graph.GetAllNodes()) visited.Add(s, false);
41
42    // Kick Start
43    stack.Enqueue(start);
44
45    while (!stack.IsEmpty())
46    {
47
48        string node = stack.Dequeue();
49        path.Add(node);
50        visited[node] = true;
51
52        List<string> connections = graph.GetNode(node);
53
54        connections.Reverse();
```

```

55
56     foreach (string s in connections)
57     {
58         if (visited[s] == false)
59         {
60             stack.Enqueue(s);
61         }
62     }
63 }
64
65     return path.ToArray();
66 }

```

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

#### 1.6.4 Windows Forms with Images

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

#### Example

```

1 partial class ShowImage
2 {
3     /// <summary>
4     /// Required designer variable.
5     /// </summary>
6     private System.ComponentModel.IContainer components = null;
7
8     /// <summary>
9     /// Clean up any resources being used.
10    /// </summary>
11    /// <param name="disposing">true if managed resources should be disposed; otherwise,
12    ← false.</param>
13    protected override void Dispose(bool disposing)
14    {
15        if (disposing && (components != null))
16        {
17            components.Dispose();
18        }
19        base.Dispose(disposing);
20    }
21
22 #region Windows Form Designer generated code

```

```
23  /// <summary>
24  /// Required method for Designer support - do not modify
25  /// the contents of this method with the code editor.
26  /// </summary>
27  private void InitializeComponent()
28  {
29      this.pictureBox = new System.Windows.Forms.PictureBox();
30      this.next = new System.Windows.Forms.Button();
31      this.content = new System.Windows.Forms.RichTextBox();
32      ((System.ComponentModel.ISupportInitialize)(this.pictureBox)).BeginInit();
33      this.SuspendLayout();
34      // 
35      // pictureBox
36      // 
37      this.pictureBox.Location = new System.Drawing.Point(12, 12);
38      this.pictureBox.Name = "pictureBox";
39      this.pictureBox.Size = new System.Drawing.Size(500, 450);
40      this.pictureBox.TabIndex = 1;
41      this.pictureBox.TabStop = false;
42      // 
43      // next
44      // 
45      this.next.Font = new System.Drawing.Font("JetBrains Mono SemiBold", 24.75F,
→       System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
46      this.next.Location = new System.Drawing.Point(518, 384);
47      this.next.Name = "next";
48      this.next.Size = new System.Drawing.Size(354, 78);
49      this.next.TabIndex = 4;
50      this.next.Text = "Continue";
51      this.next.UseVisualStyleBackColor = true;
52      this.next.Click += new System.EventHandler(this.next_Click);
53      // 
54      // content
55      // 
56      this.content.AcceptsTab = true;
57      this.content.Font = new System.Drawing.Font("JetBrains Mono SemiBold", 15F,
→       System.Drawing.FontStyle.Bold);
58      this.content.Location = new System.Drawing.Point(518, 12);
59      this.content.Name = "content";
60      this.content.ReadOnly = true;
61      this.content.Size = new System.Drawing.Size(354, 366);
62      this.content.TabIndex = 5;
63      this.content.Text = "";
64      // 
65      // ShowImage
66      // 
67      this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
```

```
68     this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
69     this.ClientSize = new System.Drawing.Size(884, 474);
70     this.Controls.Add(this.content);
71     this.Controls.Add(this.next);
72     this.Controls.Add(this.pictureBox);
73     this.Name = "ShowImage";
74     this.Text = "ShowImage";
75     this.Load += new System.EventHandler(this.ShowImage_Load);
76     ((System.ComponentModel.ISupportInitialize)(this.pictureBox)).EndInit();
77     this.ResumeLayout(false);
78 }
79 }
80 #endregion
81
82 private System.Windows.Forms.PictureBox pictureBox;
83 private System.Windows.Forms.Button next;
84 private System.Windows.Forms.RichTextBox content;
85 }
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        pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;
106        pictureBox.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

2.2 Between each stage the user should be able to repeat the last step in order to change parameters.

*The user should be able to change (at various stages):*

2.2.1 The sigma value of the Gaussian elimination

2.2.2 The lower threshold value

2.2.3 The higher threshold value

2.2.4 The Gaussian kernel size

2.2.5 The black and white filter ratios

2.2.6 The amount of times embossing is performed

2.2.7 The times de-blocking should be performed

2.3 The edge detection must have the option to be multi threaded.

- 2.3.1 There should be presets to allow quicker processing
  - 2.3.1.1 There should be a preset for hand drawn images
  - 2.3.1.2 There should be a preset for photographed images
  - 2.3.1.3 There should be a preset for screen shot images
- 2.4 The edge detection must have the option to be single threaded
- 3. The Program must overlay the detected roads onto the original image
  - 3.1 The result of the edge detection will be shown to the user before road detection
  - 3.2 The program will perform road detection
    - 3.2.1 The image should have the option to be inverted
    - 3.2.2 A filling algorithm should be applied to the image
    - 3.2.3 The percentage threshold for non roads much be changeable by the user
    - 3.2.4 The total filled image can be displayed to the user
    - 3.2.5 The singled out roads and paths must be shown to the user
- 4. The Program must allow Map Traversal
  - 4.1 There should be Multiple Traversal Algorithms Available to be chosen from.
    - 4.1.1 The Program should implement Routing Algorithms
      - 4.1.1.1 This includes Dijkstra's algorithm
      - 4.1.1.2 This includes A\*
    - 4.1.2 The Program should Implement Searching Algorithms these do not have to be shown to the user.
      - 4.1.2.1 This includes BFS (Breadth-first search).
      - 4.1.2.2 This includes DFS (Depth-first search).
  - 4.2 Depending on the option that the user chooses they can either
    - 4.2.1 Decide a specific algorithm to use
      - 4.2.1.1 The general efficiency should be displayed.
      - 4.2.1.2 The general length of each should be displayed.
      - 4.2.1.3 The node count of each should be displayed if Dijkstra's is selected.
- 5. The Program must have a Clear and Simplistic GUI.
  - 5.1 At a glance the user should be able to ascertain which step they are at in the process.
  - 5.2 Whenever a forms is displayed it should not serve more than one purpose.
  - 5.3 There should be a setting so that if the user chooses more detail can be displayed.
  - 5.4 The main user window should not be cluttered with old information.
- 6. The program must implement abstract data types
  - 6.1 The program must implement a matrix class
    - 6.1.1 The program must be able to perform basic operations
      - 6.1.1.1 Perform matrix multiplication
      - 6.1.1.2 Perform matrix addition
      - 6.1.1.3 Perform matrix subtraction
      - 6.1.1.4 Perform scalar multiplication

- 6.1.1.5 Perform matrix minimization
- 6.1.2 The program must be able to find the determinant of a matrix
- 6.1.3 The program must be able to find the inverse of a matrix
- 6.1.4 The program must be able to apply the convolution operation
- 6.2 The program should implement a graph class
  - 6.2.1 The graph should be able to be modified by
    - 6.2.1.1 Inserting Nodes
    - 6.2.1.2 Accessing per node
    - 6.2.1.3 Access all nodes
    - 6.2.1.4 Inserting connections between nodes
  - 6.2.2 It should be implemented using an adjacency list.

### **Extension Objectives**

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

## **1.8 Modelling**

TODO

## 2 Technical Design

### 2.1 Programming Language Selection and Libraries Used

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

Find below a list of all libraries I used:

#### Linq

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

#### Bitmap

In order for my program to function a required part of it is that it is able to take an image as an input. In native C there is no set way to do this. Therefore I needed to use the Microsoft System.Drawing Namespace. This namespace provides access to GDI+ basic graphics functionality. This does limit this project as is to only working on Windows since the library requires access to the GDI+ native library which is only on windows services.

The only part of this library I will be using is the Bitmap class. This will allow me to accept all types of images without the need of parsing them myself since this is not the aim of my project.

#### Windows Forms

In order to complete my objectives my program will need to be easy to use and any user with some degree of technical competency should be able to use it. In order to achieve this objective I thought that instead of using some form of console input in order to get a starting and an end location, that it would be better to use some form of GUI. In order to do this I will use Windows Forms

### 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
<b>1.1.(2)</b> 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" 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
2	Entering a PNG	Enter the test image as a PNG 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
3	Entering a BMP	Enter the test image as a BMP 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
4	Entering a TIFF	Enter the test image as a TIFF 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
<b>1.1.1</b> 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
<b>1.1.3</b> 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
<b>1.4</b> 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 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
<b>2.1</b> 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
<b>2.3.1</b> ATThere 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
<b>2.3</b> 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
<b>2.2</b> 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.

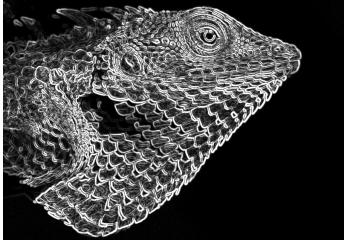
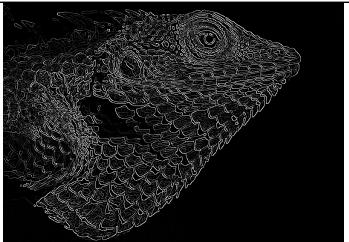
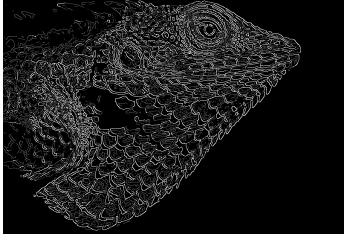
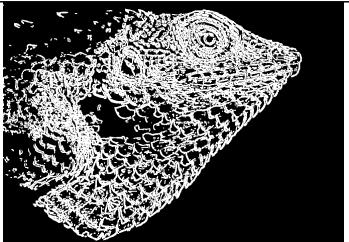
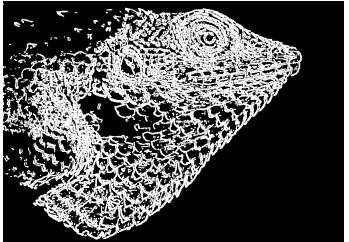


(a) Example Image Used

Sourced from Wikipedia<sup>©</sup>

[https://en.wikipedia.org/wiki/Canny\\_edge\\_detector#Walkthrough\\_of\\_the\\_algorithm](https://en.wikipedia.org/wiki/Canny_edge_detector#Walkthrough_of_the_algorithm)

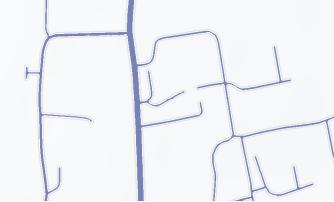
<b>2.4</b> The edge detection must have the option to be single threaded					
<b>2.2.1 - 2.2.7</b> Stages of edge detection.					
6	Black and White Method	Canny Edge Detection method should be ran with the original testing image.		Pass	TODO
7	Gaussian Filter Method	Canny Edge Detection method should 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
<b>3</b> 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 must 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 of 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
<b>4.1.1.2</b> This includes A* (a specialised Dijkstra)					
9	Run A* Image	Two nodes should be placed on points on the graph, then the "Pathfind" button should be clicked.	The algorithm will run the A* algorithm which using a heuristic algorithm will more efficiently find a path to the end node. It should run faster than Dijkstra's.	Pass	TODO

### 3.1.6 Logging and Saves Testing Table

Test No.	Name	Input Data / Description	Expected Output	Pass Fail	Test Evidence
<b>8</b> 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
<b>7.6</b> The saved binary file should be able to 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
<b>7.3</b> 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
<b>7.5</b> 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
<b>7.7</b> The saved binary file should 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
<b>6.1:</b> The program must implement a matrix class					
1	Matrix Constructor	b	c	Pass	TODO
2	Array Index Accessing of Matrix	b	c	Pass	TODO
3	Adding Matrices	b	c	Pass	TODO
4	Subtracting Matrices	b	c	Pass	TODO
5	Matrix Multiplication	b	c	Pass	TODO

6	Scalar Multiplication	b	c	Pass	TODO
7	Matrix Minimisation	b	c	Pass	TODO
8	Matrix Convolution	b	c	Pass	TODO

X.X: No set objective but contribute to the simplistic and user input objectives

9	Progress Bar Creation	b	c	Pass	TODO
10	Progress Bar Update Action	b	c	Pass	TODO
11	Coord Struct ToString	b	c	Pass	TODO
12	Coord Struct Equals Method	b	c	Pass	TODO
13	Coord Struct Equals Operator	b	c	Pass	TODO
14	Coord Struct Not Equals Operator	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

```
1  using System;
2  using System.Drawing;
3  using System.IO;
4  using System.Threading.Tasks;
5
6  namespace MultithreadedEdgeDetection
7  {
8      public class Program
9      {
10          public static void Main(string[] args)
11          {
12              Directory.CreateDirectory("./out");
13              var thing = System.Diagnostics.Stopwatch.StartNew();
14              Bitmap image = new Bitmap("./image.jpg");
15              if (image.Width < 400 || image.Width > 400)
16                  throw new Exception("Too small must be at least 400 x 400");
17              if (image.Width % 2 == 1 || image.Height % 2 == 1)
18                  throw new Exception("Must be of even dimensions");
19
20              Bitmap[] images = SplitImage(image);
21
22              Task<double[,]>[] tasks = new Task<double[,]>[4];
23
24              for (int i = 0; i < tasks.Length; i++)
25              {
26                  // To overcome the capture condition
27                  int copyI = i;
28                  CannyDetection item = new CannyDetection();
29                  Task<double[,]> task = new Task<double[,]>(() => item.DoDetect(images[copyI], copyI + 1));
30                  task.Start();
31                  tasks[i] = task;
32              }
33
34              Task.WaitAll(tasks);
35              thing.Stop();
36
37              double[,] partA = new double[image.Height / 2, image.Width];
38              double[,] partB = new double[image.Height / 2, image.Width];
39              for (int i = 0; i < tasks[0].Result.GetLength(0); i++)
40              {
41                  for (int j = 0; j < tasks[0].Result.GetLength(1); j++)
42                      partA[i, j] = tasks[0].Result[i, j];
43
44                  for (int y = 0; y < tasks[1].Result.GetLength(1); y++)
45                      partA[i, y + tasks[0].Result.GetLength(1)] = tasks[1].Result[i, y];
46              }
47
48              for (int i = 0; i < tasks[2].Result.GetLength(0); i++)
49              {
50                  for (int j = 0; j < tasks[2].Result.GetLength(1); j++)
51                      partB[i, j] = tasks[2].Result[i, j];
52
53                  for (int y = 0; y < tasks[3].Result.GetLength(1); y++)
54                      partB[i, y + tasks[2].Result.GetLength(1)] = tasks[3].Result[i, y];
55              }
56          }
57      }
58  }
```

```

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($"7. Beginning Secondary Min Max Thresholding ({id})");
182     (double, bool) [,] doubleThresholdArray = ApplyDoubleThreshold(0.1, 0.3, gradientMagnitudeThreshold);
183
184     double[,] doubleThresholdImageArray = new double[input.Height, input.Width];
185     for (int i = 0; i < input.Height; i++) for (int j = 0; j < input.Width; j++) doubleThresholdImageArray[i, j]
186     ← = doubleThresholdArray[i, j].Item1;
187     Bitmap doubleThresholdImage = DoubleArrayToBitmap(doubleThresholdImageArray);
188     doubleThresholdImage.Save($"./out/i{id}.jpg");
189     doubleThresholdImage.Dispose();
190
191     Console.WriteLine($"8. Applying Hysteresis ({id})");
192     double[,] edgeTrackingHystersis = ApplyEdgeTrackingHystersis(doubleThresholdArray);
193     Bitmap finalImage = DoubleArrayToBitmap(edgeTrackingHystersis);
194     finalImage.Save($"./out/j{id}.jpg");
195     finalImage.Dispose();
196
197     Console.WriteLine("9. Embossing out image");
198     double[,] embosArray = EmbosImage(edgeTrackingHystersis);
199     Bitmap embosImage = DoubleArrayToBitmap(embosArray);
200     embosImage.Save("./out/k.jpg");
201     embosImage.Dispose();
202
203     Console.WriteLine("10. Filling in the blanks");
204     double[,] filledArray = FillImage(embosArray);
205     Bitmap filledImage = DoubleArrayToBitmap(filledArray);
206     filledImage.Save("./out/l.jpg");
207     filledImage.Dispose();
208
209     Console.WriteLine($"Done {id}");
210
211     return edgeTrackingHystersis;
212 }
213
214 public double[,] FillImage(double[,] imageArray)
215 {
216     double[,] result = imageArray;
217
218     for (int i = 0; i < imageArray.GetLength(0); i++)
219     {
220         for (int j = 0; j < imageArray.GetLength(1); j++)
221         {
222             Matrix imageKernel = BuildKernel(j, i, 3, imageArray);
223             int count = 0;
224             foreach (double value in imageKernel.matrix)
225             {
226                 if (value >= 255) count++;
227             }
228
229             if (count > 4) result[i, j] = 255;
230         }
231     }
232
233     return result;
234 }
```

```

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

```

```

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

```

```

354     {
355         double min = l * 255;
356         double max = h * 255;
357
358         (double, bool)[,] result = new (double, bool)[gradients.GetLength(0), gradients.GetLength(1)];
359
360         for (int i = 0; i < gradients.GetLength(0); i++)
361         {
362             for (int j = 0; j < gradients.GetLength(1); j++)
363             {
364                 if (gradients[i, j] < min) result[i, j] = (0, false);
365                 else if (gradients[i, j] > min && gradients[i, j] < max) result[i, j] = (gradients[i, j], false);
366                 else if (gradients[i, j] > max) result[i, j] = (gradients[i, j], true);
367                 else throw new Exception();
368             }
369         }
370
371         return result;
372     }
373
374     public double[,] ConvertThetaToDegrees(double[,] thetaArray)
375     {
376         double[,] result = new double[thetaArray.GetLength(0), thetaArray.GetLength(1)];
377         for (int i = 0; i < thetaArray.GetLength(0); i++) for (int j = 0; j < thetaArray.GetLength(1); j++)
378             result[i, j] = 180 * Math.Abs(thetaArray[i, j]) / Math.PI;
379         return result;
380     }
381
382     public double[,] CalculateTheta(double[,] gradX, double[,] gradY)
383     {
384         double[,] result = new double[gradX.GetLength(0), gradX.GetLength(1)];
385         for (int i = 0; i < gradX.GetLength(0); i++) for (int j = 0; j < gradX.GetLength(1); j++) result[i, j] =
386             Math.Atan2(gradY[i, j], gradX[i, j]);
387         return result;
388     }
389
390     public double[,] CalculateGradientCombined(double[,] gradX, double[,] gradY)
391     {
392         double[,] result = new double[gradX.GetLength(0), gradX.GetLength(1)];
393         for (int i = 0; i < gradX.GetLength(0); i++) for (int j = 0; j < gradX.GetLength(1); j++) result[i, j] =
394             Math.Sqrt(Math.Pow(gradX[i, j], 2) + Math.Pow(gradY[i, j], 2));
395         return result;
396     }
397
398     public double[,] CalculateGradientY(double[,] imageArray)
399     {
400         double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];
401
402         Matrix sobelY = new Matrix(new double[,] {
403             { 1, 0, -1 },
404             { 2, 0, -2 },
405             { 1, 0, -1 },
406         });
407
408         for (int i = 0; i < imageArray.GetLength(0); i++)
409         {
410             for (int j = 0; j < imageArray.GetLength(1); j++)
411             {
412                 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             double distance = Math.Sqrt(Math.Pow(i - cntX, 2) + Math.Pow(j - cntY, 2));
473             if (distance >= sigma)
474                 result[i, j] = 0;
475             else
476                 result[i, j] = Math.Exp(-distance / (2 * sigma * sigma));
477             sum += result[i, j];
478         }
479         cntX++;
480     }
481
482     for (int i = 0; i < k; i++)
483     {
484         for (int j = 0; j < k; j++)
485         {
486             result[i, j] /= sum;
487         }
488     }
489
490     return result;
491 }
492
493 public void ShowImage(double[,] image)
494 {
495     for (int i = 0; i < image.GetLength(0); i++)
496     {
497         for (int j = 0; j < image.GetLength(1); j++)
498         {
499             Console.Write((int)(image[i, j] * 255));
500         }
501         Console.WriteLine();
502     }
503 }
504
505 public void ShowMatrix(Matrix matrix)
506 {
507     for (int i = 0; i < matrix.RowCount; i++)
508     {
509         for (int j = 0; j < matrix.ColumnCount; j++)
510         {
511             Console.Write(matrix[i, j]);
512         }
513         Console.WriteLine();
514     }
515 }
516
517 public void ShowMatrix(double[,] matrix)
518 {
519     for (int i = 0; i < matrix.GetLength(0); i++)
520     {
521         for (int j = 0; j < matrix.GetLength(1); j++)
522         {
523             Console.Write((int)(matrix[i, j] * 255));
524         }
525         Console.WriteLine();
526     }
527 }
528
529 public void ShowMatrix(Matrix matrix, double[,] image)
530 {
531     for (int i = 0; i < matrix.RowCount; i++)
532     {
533         for (int j = 0; j < matrix.ColumnCount; j++)
534         {
535             image[i, j] = matrix[i, j];
536         }
537     }
538 }
539
540 public void ShowImage(double[,] image)
541 {
542     for (int i = 0; i < image.GetLength(0); i++)
543     {
544         for (int j = 0; j < image.GetLength(1); j++)
545         {
546             Console.Write((int)(image[i, j] * 255));
547         }
548         Console.WriteLine();
549     }
550 }
551
552 public void ShowMatrix(Matrix matrix)
553 {
554     for (int i = 0; i < matrix.RowCount; i++)
555     {
556         for (int j = 0; j < matrix.ColumnCount; j++)
557         {
558             Console.Write(matrix[i, j]);
559         }
560         Console.WriteLine();
561     }
562 }
563
564 public void ShowMatrix(double[,] matrix)
565 {
566     for (int i = 0; i < matrix.GetLength(0); i++)
567     {
568         for (int j = 0; j < matrix.GetLength(1); j++)
569         {
570             Console.Write((int)(matrix[i, j] * 255));
571         }
572         Console.WriteLine();
573     }
574 }
575
576 public void ShowMatrix(Matrix matrix, double[,] image)
577 {
578     for (int i = 0; i < matrix.RowCount; i++)
579     {
580         for (int j = 0; j < matrix.ColumnCount; j++)
581         {
582             image[i, j] = matrix[i, j];
583         }
584     }
585 }
586
587 public void ShowImage(double[,] image)
588 {
589     for (int i = 0; i < image.GetLength(0); i++)
590     {
591         for (int j = 0; j < image.GetLength(1); j++)
592         {
593             Console.Write((int)(image[i, j] * 255));
594         }
595         Console.WriteLine();
596     }
597 }
598
599 public void ShowMatrix(Matrix matrix)
600 {
601     for (int i = 0; i < matrix.RowCount; i++)
602     {
603         for (int j = 0; j < matrix.ColumnCount; j++)
604         {
605             Console.Write(matrix[i, j]);
606         }
607         Console.WriteLine();
608     }
609 }
610
611 public void ShowMatrix(double[,] matrix)
612 {
613     for (int i = 0; i < matrix.GetLength(0); i++)
614     {
615         for (int j = 0; j < matrix.GetLength(1); j++)
616         {
617             Console.Write((int)(matrix[i, j] * 255));
618         }
619         Console.WriteLine();
620     }
621 }
622
623 public void ShowMatrix(Matrix matrix, double[,] image)
624 {
625     for (int i = 0; i < matrix.RowCount; i++)
626     {
627         for (int j = 0; j < matrix.ColumnCount; j++)
628         {
629             image[i, j] = matrix[i, j];
630         }
631     }
632 }
633
634 public void ShowImage(double[,] image)
635 {
636     for (int i = 0; i < image.GetLength(0); i++)
637     {
638         for (int j = 0; j < image.GetLength(1); j++)
639         {
640             Console.Write((int)(image[i, j] * 255));
641         }
642         Console.WriteLine();
643     }
644 }
645
646 public void ShowMatrix(Matrix matrix)
647 {
648     for (int i = 0; i < matrix.RowCount; i++)
649     {
650         for (int j = 0; j < matrix.ColumnCount; j++)
651         {
652             Console.Write(matrix[i, j]);
653         }
654         Console.WriteLine();
655     }
656 }
657
658 public void ShowMatrix(double[,] matrix)
659 {
660     for (int i = 0; i < matrix.GetLength(0); i++)
661     {
662         for (int j = 0; j < matrix.GetLength(1); j++)
663         {
664             Console.Write((int)(matrix[i, j] * 255));
665         }
666         Console.WriteLine();
667     }
668 }
669
670 public void ShowMatrix(Matrix matrix, double[,] image)
671 {
672     for (int i = 0; i < matrix.RowCount; i++)
673     {
674         for (int j = 0; j < matrix.ColumnCount; j++)
675         {
676             image[i, j] = matrix[i, j];
677         }
678     }
679 }
680
681 public void ShowImage(double[,] image)
682 {
683     for (int i = 0; i < image.GetLength(0); i++)
684     {
685         for (int j = 0; j < image.GetLength(1); j++)
686         {
687             Console.Write((int)(image[i, j] * 255));
688         }
689         Console.WriteLine();
690     }
691 }
692
693 public void ShowMatrix(Matrix matrix)
694 {
695     for (int i = 0; i < matrix.RowCount; i++)
696     {
697         for (int j = 0; j < matrix.ColumnCount; j++)
698         {
699             Console.Write(matrix[i, j]);
700         }
701         Console.WriteLine();
702     }
703 }
704
705 public void ShowMatrix(double[,] matrix)
706 {
707     for (int i = 0; i < matrix.GetLength(0); i++)
708     {
709         for (int j = 0; j < matrix.GetLength(1); j++)
710         {
711             Console.Write((int)(matrix[i, j] * 255));
712         }
713         Console.WriteLine();
714     }
715 }
716
717 public void ShowMatrix(Matrix matrix, double[,] image)
718 {
719     for (int i = 0; i < matrix.RowCount; i++)
720     {
721         for (int j = 0; j < matrix.ColumnCount; j++)
722         {
723             image[i, j] = matrix[i, j];
724         }
725     }
726 }
727
728 public void ShowImage(double[,] image)
729 {
730     for (int i = 0; i < image.GetLength(0); i++)
731     {
732         for (int j = 0; j < image.GetLength(1); j++)
733         {
734             Console.Write((int)(image[i, j] * 255));
735         }
736         Console.WriteLine();
737     }
738 }
739
740 public void ShowMatrix(Matrix matrix)
741 {
742     for (int i = 0; i < matrix.RowCount; i++)
743     {
744         for (int j = 0; j < matrix.ColumnCount; j++)
745         {
746             Console.Write(matrix[i, j]);
747         }
748         Console.WriteLine();
749     }
750 }
751
752 public void ShowMatrix(double[,] matrix)
753 {
754     for (int i = 0; i < matrix.GetLength(0); i++)
755     {
756         for (int j = 0; j < matrix.GetLength(1); j++)
757         {
758             Console.Write((int)(matrix[i, j] * 255));
759         }
760         Console.WriteLine();
761     }
762 }
763
764 public void ShowMatrix(Matrix matrix, double[,] image)
765 {
766     for (int i = 0; i < matrix.RowCount; i++)
767     {
768         for (int j = 0; j < matrix.ColumnCount; j++)
769         {
770             image[i, j] = matrix[i, j];
771         }
772     }
773 }
774
775 public void ShowImage(double[,] image)
776 {
777     for (int i = 0; i < image.GetLength(0); i++)
778     {
779         for (int j = 0; j < image.GetLength(1); j++)
780         {
781             Console.Write((int)(image[i, j] * 255));
782         }
783         Console.WriteLine();
784     }
785 }
786
787 public void ShowMatrix(Matrix matrix)
788 {
789     for (int i = 0; i < matrix.RowCount; i++)
790     {
791         for (int j = 0; j < matrix.ColumnCount; j++)
792         {
793             Console.Write(matrix[i, j]);
794         }
795         Console.WriteLine();
796     }
797 }
798
799 public void ShowMatrix(double[,] matrix)
800 {
801     for (int i = 0; i < matrix.GetLength(0); i++)
802     {
803         for (int j = 0; j < matrix.GetLength(1); j++)
804         {
805             Console.Write((int)(matrix[i, j] * 255));
806         }
807         Console.WriteLine();
808     }
809 }
810
811 public void ShowMatrix(Matrix matrix, double[,] image)
812 {
813     for (int i = 0; i < matrix.RowCount; i++)
814     {
815         for (int j = 0; j < matrix.ColumnCount; j++)
816         {
817             image[i, j] = matrix[i, j];
818         }
819     }
820 }
821
822 public void ShowImage(double[,] image)
823 {
824     for (int i = 0; i < image.GetLength(0); i++)
825     {
826         for (int j = 0; j < image.GetLength(1); j++)
827         {
828             Console.Write((int)(image[i, j] * 255));
829         }
830         Console.WriteLine();
831     }
832 }
833
834 public void ShowMatrix(Matrix matrix)
835 {
836     for (int i = 0; i < matrix.RowCount; i++)
837     {
838         for (int j = 0; j < matrix.ColumnCount; j++)
839         {
840             Console.Write(matrix[i, j]);
841         }
842         Console.WriteLine();
843     }
844 }
845
846 public void ShowMatrix(double[,] matrix)
847 {
848     for (int i = 0; i < matrix.GetLength(0); i++)
849     {
850         for (int j = 0; j < matrix.GetLength(1); j++)
851         {
852             Console.Write((int)(matrix[i, j] * 255));
853         }
854         Console.WriteLine();
855     }
856 }
857
858 public void ShowMatrix(Matrix matrix, double[,] image)
859 {
860     for (int i = 0; i < matrix.RowCount; i++)
861     {
862         for (int j = 0; j < matrix.ColumnCount; j++)
863         {
864             image[i, j] = matrix[i, j];
865         }
866     }
867 }
868
869 public void ShowImage(double[,] image)
870 {
871     for (int i = 0; i < image.GetLength(0); i++)
872     {
873         for (int j = 0; j < image.GetLength(1); j++)
874         {
875             Console.Write((int)(image[i, j] * 255));
876         }
877         Console.WriteLine();
878     }
879 }
880
881 public void ShowMatrix(Matrix matrix)
882 {
883     for (int i = 0; i < matrix.RowCount; i++)
884     {
885         for (int j = 0; j < matrix.ColumnCount; j++)
886         {
887             Console.Write(matrix[i, j]);
888         }
889         Console.WriteLine();
890     }
891 }
892
893 public void ShowMatrix(double[,] matrix)
894 {
895     for (int i = 0; i < matrix.GetLength(0); i++)
896     {
897         for (int j = 0; j < matrix.GetLength(1); j++)
898         {
899             Console.Write((int)(matrix[i, j] * 255));
900         }
901         Console.WriteLine();
902     }
903 }
904
905 public void ShowMatrix(Matrix matrix, double[,] image)
906 {
907     for (int i = 0; i < matrix.RowCount; i++)
908     {
909         for (int j = 0; j < matrix.ColumnCount; j++)
910         {
911             image[i, j] = matrix[i, j];
912         }
913     }
914 }
915
916 public void ShowImage(double[,] image)
917 {
918     for (int i = 0; i < image.GetLength(0); i++)
919     {
920         for (int j = 0; j < image.GetLength(1); j++)
921         {
922             Console.Write((int)(image[i, j] * 255));
923         }
924         Console.WriteLine();
925     }
926 }
927
928 public void ShowMatrix(Matrix matrix)
929 {
930     for (int i = 0; i < matrix.RowCount; i++)
931     {
932         for (int j = 0; j < matrix.ColumnCount; j++)
933         {
934             Console.Write(matrix[i, j]);
935         }
936         Console.WriteLine();
937     }
938 }
939
940 public void ShowMatrix(double[,] matrix)
941 {
942     for (int i = 0; i < matrix.GetLength(0); i++)
943     {
944         for (int j = 0; j < matrix.GetLength(1); j++)
945         {
946             Console.Write((int)(matrix[i, j] * 255));
947         }
948         Console.WriteLine();
949     }
950 }
951
952 public void ShowMatrix(Matrix matrix, double[,] image)
953 {
954     for (int i = 0; i < matrix.RowCount; i++)
955     {
956         for (int j = 0; j < matrix.ColumnCount; j++)
957         {
958             image[i, j] = matrix[i, j];
959         }
960     }
961 }
962
963 public void ShowImage(double[,] image)
964 {
965     for (int i = 0; i < image.GetLength(0); i++)
966     {
967         for (int j = 0; j < image.GetLength(1); j++)
968         {
969             Console.Write((int)(image[i, j] * 255));
970         }
971         Console.WriteLine();
972     }
973 }
974
975 public void ShowMatrix(Matrix matrix)
976 {
977     for (int i = 0; i < matrix.RowCount; i++)
978     {
979         for (int j = 0; j < matrix.ColumnCount; j++)
980         {
981             Console.Write(matrix[i, j]);
982         }
983         Console.WriteLine();
984     }
985 }
986
987 public void ShowMatrix(double[,] matrix)
988 {
989     for (int i = 0; i < matrix.GetLength(0); i++)
990     {
991         for (int j = 0; j < matrix.GetLength(1); j++)
992         {
993             Console.Write((int)(matrix[i, j] * 255));
994         }
995         Console.WriteLine();
996     }
997 }
998
999 public void ShowMatrix(Matrix matrix, double[,] image)
1000 {
1001     for (int i = 0; i < matrix.RowCount; i++)
1002     {
1003         for (int j = 0; j < matrix.ColumnCount; j++)
1004         {
1005             image[i, j] = matrix[i, j];
1006         }
1007     }
1008 }
1009
1010 public void ShowImage(double[,] image)
1011 {
1012     for (int i = 0; i < image.GetLength(0); i++)
1013     {
1014         for (int j = 0; j < image.GetLength(1); j++)
1015         {
1016             Console.Write((int)(image[i, j] * 255));
1017         }
1018         Console.WriteLine();
1019     }
1020 }
1021
1022 public void ShowMatrix(Matrix matrix)
1023 {
1024     for (int i = 0; i < matrix.RowCount; i++)
1025     {
1026         for (int j = 0; j < matrix.ColumnCount; j++)
1027         {
1028             Console.Write(matrix[i, j]);
1029         }
1030         Console.WriteLine();
1031     }
1032 }
1033
1034 public void ShowMatrix(double[,] matrix)
1035 {
1036     for (int i = 0; i < matrix.GetLength(0); i++)
1037     {
1038         for (int j = 0; j < matrix.GetLength(1); j++)
1039         {
1040             Console.Write((int)(matrix[i, j] * 255));
1041         }
1042         Console.WriteLine();
1043     }
1044 }
1045
1046 public void ShowMatrix(Matrix matrix, double[,] image)
1047 {
1048     for (int i = 0; i < matrix.RowCount; i++)
1049     {
1050         for (int j = 0; j < matrix.ColumnCount; j++)
1051         {
1052             image[i, j] = matrix[i, j];
1053         }
1054     }
1055 }
1056
1057 public void ShowImage(double[,] image)
1058 {
1059     for (int i = 0; i < image.GetLength(0); i++)
1060     {
1061         for (int j = 0; j < image.GetLength(1); j++)
1062         {
1063             Console.Write((int)(image[i, j] * 255));
1064         }
1065         Console.WriteLine();
1066     }
1067 }
1068
1069 public void ShowMatrix(Matrix matrix)
1070 {
1071     for (int i = 0; i < matrix.RowCount; i++)
1072     {
1073         for (int j = 0; j < matrix.ColumnCount; j++)
1074         {
1075             Console.Write(matrix[i, j]);
1076         }
1077         Console.WriteLine();
1078     }
1079 }
1080
1081 public void ShowMatrix(double[,] matrix)
1082 {
1083     for (int i = 0; i < matrix.GetLength(0); i++)
1084     {
1085         for (int j = 0; j < matrix.GetLength(1); j++)
1086         {
1087             Console.Write((int)(matrix[i, j] * 255));
1088         }
1089         Console.WriteLine();
1090     }
1091 }
1092
1093 public void ShowMatrix(Matrix matrix, double[,] image)
1094 {
1095     for (int i = 0; i < matrix.RowCount; i++)
1096     {
1097         for (int j = 0; j < matrix.ColumnCount; j++)
1098         {
1099             image[i, j] = matrix[i, j];
1100         }
1101     }
1102 }
1103
1104 public void ShowImage(double[,] image)
1105 {
1106     for (int i = 0; i < image.GetLength(0); i++)
1107     {
1108         for (int j = 0; j < image.GetLength(1); j++)
1109         {
1110             Console.Write((int)(image[i, j] * 255));
1111         }
1112         Console.WriteLine();
1113     }
1114 }
1115
1116 public void ShowMatrix(Matrix matrix)
1117 {
1118     for (int i = 0; i < matrix.RowCount; i++)
1119     {
1120         for (int j = 0; j < matrix.ColumnCount; j++)
1121         {
1122             Console.Write(matrix[i, j]);
1123         }
1124         Console.WriteLine();
1125     }
1126 }
1127
1128 public void ShowMatrix(double[,] matrix)
1129 {
1130     for (int i = 0; i < matrix.GetLength(0); i++)
1131     {
1132         for (int j = 0; j < matrix.GetLength(1); j++)
1133         {
1134             Console.Write((int)(matrix[i, j] * 255));
1135         }
1136         Console.WriteLine();
1137     }
1138 }
1139
1140 public void ShowMatrix(Matrix matrix, double[,] image)
1141 {
1142     for (int i = 0; i < matrix.RowCount; i++)
1143     {
1144         for (int j = 0; j < matrix.ColumnCount; j++)
1145         {
1146             image[i, j] = matrix[i, j];
1147         }
1148     }
1149 }
1150
1151 public void ShowImage(double[,] image)
1152 {
1153     for (int i = 0; i < image.GetLength(0); i++)
1154     {
1155         for (int j = 0; j < image.GetLength(1); j++)
1156         {
1157             Console.Write((int)(image[i, j] * 255));
1158         }
1159         Console.WriteLine();
1160     }
1161 }
1162
1163 public void ShowMatrix(Matrix matrix)
1164 {
1165     for (int i = 0; i < matrix.RowCount; i++)
1166     {
1167         for (int j = 0; j < matrix.ColumnCount; j++)
1168         {
1169             Console.Write(matrix[i, j]);
1170         }
1171         Console.WriteLine();
1172     }
1173 }
1174
1175 public void ShowMatrix(double[,] matrix)
1176 {
1177     for (int i = 0; i < matrix.GetLength(0); i++)
1178     {
1179         for (int j = 0; j < matrix.GetLength(1); j++)
1180         {
1181             Console.Write((int)(matrix[i, j] * 255));
1182         }
1183         Console.WriteLine();
1184     }
1185 }
1186
1187 public void ShowMatrix(Matrix matrix, double[,] image)
1188 {
1189     for (int i = 0; i < matrix.RowCount; i++)
1190     {
1191         for (int j = 0; j < matrix.ColumnCount; j++)
1192         {
1193             image[i, j] = matrix[i, j];
1194         }
1195     }
1196 }
1197
1198 public void ShowImage(double[,] image)
1199 {
1200     for (int i = 0; i < image.GetLength(0); i++)
1201     {
1202         for (int j = 0; j < image.GetLength(1); j++)
1203         {
1204             Console.Write((int)(image[i, j] * 255));
1205         }
1206         Console.WriteLine();
1207     }
1208 }
1209
1210 public void ShowMatrix(Matrix matrix)
1211 {
1212     for (int i = 0; i < matrix.RowCount; i++)
1213     {
1214         for (int j = 0; j < matrix.ColumnCount; j++)
1215         {
1216             Console.Write(matrix[i, j]);
1217         }
1218         Console.WriteLine();
1219     }
1220 }
1221
1222 public void ShowMatrix(double[,] matrix)
1223 {
1224     for (int i = 0; i < matrix.GetLength(0); i++)
1225     {
1226         for (int j = 0; j < matrix.GetLength(1); j++)
1227         {
1228             Console.Write((int)(matrix[i, j] * 255));
1229         }
1230         Console.WriteLine();
1231     }
1232 }
1233
1234 public void ShowMatrix(Matrix matrix, double[,] image)
1235 {
1236     for (int i = 0; i < matrix.RowCount; i++)
1237     {
1238         for (int j = 0; j < matrix.ColumnCount; j++)
1239         {
1240             image[i, j] = matrix[i, j];
1241         }
1242     }
1243 }
1244
1245 public void ShowImage(double[,] image)
1246 {
1247     for (int i = 0; i < image.GetLength(0); i++)
1248     {
1249         for (int j = 0; j < image.GetLength(1); j++)
1250         {
1251             Console.Write((int)(image[i, j] * 255));
1252         }
1253         Console.WriteLine();
1254     }
1255 }
1256
1257 public void ShowMatrix(Matrix matrix)
1258 {
1259     for (int i = 0; i < matrix.RowCount; i++)
1260     {
1261         for (int j = 0; j < matrix.ColumnCount; j++)
1262         {
1263             Console.Write(matrix[i, j]);
1264         }
1265         Console.WriteLine();
1266     }
1267 }
1268
1269 public void ShowMatrix(double[,] matrix)
1270 {
1271     for (int i = 0; i < matrix.GetLength(0); i++)
1272     {
1273         for (int j = 0; j < matrix.GetLength(1); j++)
1274         {
1275             Console.Write((int)(matrix[i, j] * 255));
1276         }
1277         Console.WriteLine();
1278     }
1279 }
1280
1281 public void ShowMatrix(Matrix matrix, double[,] image)
1282 {
1283     for (int i = 0; i < matrix.RowCount; i++)
1284     {
1285         for (int j = 0; j < matrix.ColumnCount; j++)
1286         {
1287             image[i, j] = matrix[i, j];
1288         }
1289     }
1290 }
1291
1292 public void ShowImage(double[,] image)
1293 {
1294     for (int i = 0; i < image.GetLength(0); i++)
1295     {
1296         for (int j = 0; j < image.GetLength(1); j++)
1297         {
1298             Console.Write((int)(image[i, j] * 255));
1299         }
1300         Console.WriteLine();
1301     }
1302 }
1303
1304 public void ShowMatrix(Matrix matrix)
1305 {
1306     for (int i = 0; i < matrix.RowCount; i++)
1307     {
1308         for (int j = 0; j < matrix.ColumnCount; j++)
1309         {
1310             Console.Write(matrix[i, j]);
1311         }
1312         Console.WriteLine();
1313     }
1314 }
1315
1316 public void ShowMatrix(double[,] matrix)
1317 {
1318     for (int i = 0; i < matrix.GetLength(0); i++)
1319     {
1320         for (int j = 0; j < matrix.GetLength(1); j++)
1321         {
1322             Console.Write((int)(matrix[i, j] * 255));
1323         }
1324         Console.WriteLine();
1325     }
1326 }
1327
1328 public void ShowMatrix(Matrix matrix, double[,] image)
1329 {
1330     for (int i = 0; i < matrix.RowCount; i++)
1331     {
1332         for (int j = 0; j < matrix.ColumnCount; j++)
1333         {
1334             image[i, j] = matrix[i, j];
1335         }
1336     }
1337 }
1338
1339 public void ShowImage(double[,] image)
1340 {
1341     for (int i = 0; i < image.GetLength(0); i++)
1342     {
1343         for (int j = 0; j < image.GetLength(1); j++)
1344         {
1345             Console.Write((int)(image[i, j] * 255));
1346         }
1347         Console.WriteLine();
1348     }
1349 }
1350
1351 public void ShowMatrix(Matrix matrix)
1352 {
1353     for (int i = 0; i < matrix.RowCount; i++)
1354     {
1355         for (int j = 0; j < matrix.ColumnCount; j++)
1356         {
1357             Console.Write(matrix[i, j]);
1358         }
1359         Console.WriteLine();
1360     }
1361 }
1362
1363 public void ShowMatrix(double[,] matrix)
1364 {
1365     for (int i = 0; i < matrix.GetLength(0); i++)
1366     {
1367         for (int j = 0; j < matrix.GetLength(1); j++)
1368         {
1369             Console.Write((int)(matrix[i, j] * 255));
1370         }
1371         Console.WriteLine();
1372     }
1373 }
1374
1375 public void ShowMatrix(Matrix matrix, double[,] image)
1376 {
1377     for (int i = 0; i < matrix.RowCount; i++)
1378     {
1379         for (int j = 0; j < matrix.ColumnCount; j++)
1380         {
1381             image[i, j] = matrix[i, j];
1382         }
1383     }
1384 }
1385
1386 public void ShowImage(double[,] image)
1387 {
1388     for (int i = 0; i < image.GetLength(0); i++)
1389     {
1390         for (int j = 0; j < image.GetLength(1); j++)
1391         {
1392             Console.Write((int)(image[i, j] * 255));
1393         }
1394         Console.WriteLine();
1395     }
1396 }
1397
1398 public void ShowMatrix(Matrix matrix)
1399 {
1400     for (int i = 0; i < matrix.RowCount; i++)
1401     {
1402         for (int j = 0; j < matrix.ColumnCount; j++)
1403         {
1404             Console.Write(matrix[i, j]);
1405         }
1406         Console.WriteLine();
1407     }
1408 }
1409
1410 public void ShowMatrix(double[,] matrix)
1411 {
1412     for (int i = 0; i < matrix.GetLength(0); i++)
1413     {
1414         for (int j = 0; j < matrix.GetLength(1); j++)
1415         {
1416             Console.Write((int)(matrix[i, j] * 255));
1417         }
1418         Console.WriteLine();
1419     }
1420 }
1421
1422 public void ShowMatrix(Matrix matrix, double[,] image)
1423 {
1424     for (int i = 0; i < matrix.RowCount; i++)
1425     {
1426         for (int j = 0; j < matrix.ColumnCount; j++)
1427         {
1428             image[i, j] = matrix[i, j];
1429         }
1430     }
1431 }
1432
1433 public void ShowImage(double[,] image)
1434 {
1435     for (int i = 0; i < image.GetLength(0); i++)
1436     {
1437         for (int j = 0; j < image.GetLength(1); j++)
1438         {
1439             Console.Write((int)(image[i, j] * 255));
1440         }
1441         Console.WriteLine();
1442     }
1443 }
1444
1445 public void ShowMatrix(Matrix matrix)
1446 {
1447     for (int i = 0; i < matrix.RowCount; i++)
1448     {
1449         for (int j = 0; j < matrix.ColumnCount; j++)
1450         {
1451             Console.Write(matrix[i, j]);
1452         }
1453         Console.WriteLine();
1454     }
1455 }
1456
1457 public void ShowMatrix(double[,] matrix)
1458 {
1459     for (int i = 0; i < matrix.GetLength(0); i++)
1460     {
1461         for (int j = 0; j < matrix.GetLength(1); j++)
1462         {
1463             Console.Write((int)(matrix[i, j] * 255));
1464         }
1465         Console.WriteLine();
1466     }
1467 }
1468
1469 public void ShowMatrix(Matrix matrix, double[,] image)
1470 {
1471     for (int i = 0; i < matrix.RowCount; i++)
1472     {
1473         for (int j = 0; j < matrix.ColumnCount; j++)
1474         {
1475             image[i, j] = matrix[i, j];
1476         }
1477     }
1478 }
1479
1480 public void ShowImage(double[,] image)
1481 {
1482     for (int i = 0; i < image.GetLength(0); i++)
1483     {
1484         for (int j = 0; j < image.GetLength(1); j++)
1485         {
1486             Console.Write((int)(image[i, j] * 255));
1487         }
1488         Console.WriteLine();
1489     }
1490 }
1491
1492 public void ShowMatrix(Matrix matrix)
1493 {
1494     for (int i = 0; i < matrix.RowCount; i++)
1495     {
1496         for (int j = 0; j < matrix.ColumnCount; j++)
1497         {
1498             Console.Write(matrix[i, j]);
1499         }
1500         Console.WriteLine();
1501     }
1502 }
1503
1504 public void ShowMatrix(double[,] matrix)
1505 {
1506     for (int i = 0; i < matrix.GetLength(0); i++)
1507     {
1508         for (int j = 0; j < matrix.GetLength(1); j++)
1509         {
1510             Console.Write((int)(matrix[i, j] * 255));
1511         }
1512         Console.WriteLine();
1513     }
1514 }
1515
1516 public void ShowMatrix(Matrix matrix, double[,] image)
1517 {
1518     for (int i = 0; i < matrix.RowCount; i++)
1519     {
1520         for (int j = 0; j < matrix.ColumnCount; j++)
1521         {
1522             image[i, j] = matrix[i, j];
1523         }
1524     }
1525 }
1526
1527 public void ShowImage(double[,] image)
1528 {
1529     for (int i = 0; i < image.GetLength(0); i++)
1530     {
1531         for (int j = 0; j < image.GetLength
```

```

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

```

```

531     }
532 
533     return kernel;
534 }
535 
536     public double[,] BWFilter(Bitmap image)
537 {
538         double[,] result = new double[image.Height, image.Width];
539 
540         for (int i = 0; i < image.Height; i++)
541         {
542             for (int j = 0; j < image.Width; j++)
543             {
544                 Color c = image.GetPixel(j, i);
545                 double value = c.R * 0.299 + c.G * 0.587 + c.B * 0.114;
546 
547                 result[i, j] = Bound(0, 255, value);
548             }
549         }
550 
551         return result;
552     }
553 
554     public static int Bound(int l, int h, double v) => v > h ? h : (v < l ? l : (int)v);
555 
556     public double GetGaussianDistribution(int x, int y, double sigma) =>
557         1 / (2 * Math.PI * sigma * sigma) * Math.Exp(-((Math.Pow(x, 2) + Math.Pow(y, 2)) / (2 * sigma * sigma)));
558 
559 
560     public static Bitmap DoubleArrayToBitmap(double[,] input)
561 {
562         Bitmap image = new Bitmap(input.GetLength(1), input.GetLength(0));
563         for (int i = 0; i < image.Height; i++)
564         {
565             for (int j = 0; j < image.Width; j++)
566             {
567                 int val = Bound(0, 255, input[i, j]);
568                 image.SetPixel(j, i, Color.FromArgb(val, val, val));
569             }
570         }
571         return image;
572     }
573 }
574 
575     public class Matrix
576 {
577         public int x { get; private set; }
578         public int y { get; private set; }
579         public double[,] matrix { get; private set; }
580 
581         public Matrix(double[,] inputMatrix)
582 {
583             x = inputMatrix.GetLength(1);
584             y = inputMatrix.GetLength(0);
585             matrix = inputMatrix;
586         }
587 
588         public static double Convolution(Matrix a, Matrix b)
589 {
590             if (a.x != a.y || b.x != a.x) throw new Exception();

```

```

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

### 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>
```

```

33
34         "B", "G"
35     });
36     temp.Add("G", new List<string>
37     {
38         "D", "F"
39     });
40     temp.Add("H", new List<string>
41     {
42         "E"
43     });
44
45     Graph myGraph = new Graph(temp);
46     Console.WriteLine(string.Join(", ", DFS("A", myGraph)));
47     Console.WriteLine(string.Join(", ", BFS("A", myGraph)));
48     Console.ReadLine();
49 }
50
51     public static string[] DFS(string start, Graph graph)
52     {
53         List<string> path = new List<string>();
54         Stack<string> stack = new Stack<string>();
55         Dictionary<string, bool> visited = new Dictionary<string, bool>();
56         foreach (string s in graph.GetAllNodes()) visited.Add(s, false);
57
58         // Kick Start
59         stack.Push(start);
60
61         while (!stack.IsEmpty())
62         {
63
64             string node = stack.Pop();
65             path.Add(node);
66             visited[node] = true;
67
68             List<string> connections = graph.GetNode(node);
69
70             connections.Reverse();
71
72             foreach (string s in connections)
73             {
74                 if (visited[s] == false)
75                 {
76                     stack.Push(s);
77                 }
78             }
79         }
80
81         return path.ToArray();
82     }
83
84
85     public static string[] BFS(string start, Graph graph)
86     {
87         List<string> path = new List<string>();
88         Queue<string> stack = new Queue<string>();
89         Dictionary<string, bool> visited = new Dictionary<string, bool>();
90         foreach (string s in graph.GetAllNodes()) visited.Add(s, false);
91
92         // Kick Start

```

```
93     stack.Enqueue(start);
94
95     while (!stack.IsEmpty())
96     {
97
98         string node = stack.Dequeue();
99         path.Add(node);
100        visited[node] = true;
101
102        List<string> connections = graph.GetNode(node);
103
104        connections.Reverse();
105
106        foreach (string s in connections)
107        {
108            if (visited[s] == false)
109            {
110                stack.Enqueue(s);
111            }
112        }
113    }
114
115    return path.ToArray();
116}
117}
118
119 public class Queue<T>
120 {
121     public List<T> _data = new List<T>();
122
123     public T Dequeue()
124     {
125         T val = _data[0];
126         _data.RemoveAt(0);
127         return val;
128     }
129
130     public void Enqueue(T val) => _data.Add(val);
131
132     public bool IsEmpty() => _data.Count == 0;
133 }
134
135 public class Stack<T>
136 {
137     public List<T> _data = new List<T>();
138
139     public T Pop()
140     {
141         T val = _data[_data.Count - 1];
142         _data.RemoveAt(_data.Count - 1);
143         return val;
144     }
145
146     public void Push(T val) => _data.Add(val);
147
148     public bool IsEmpty() => _data.Count == 0;
149 }
150
151
152 }
```

```

153     public class Graph
154     {
155         public Dictionary<string, List<string>> _data = new Dictionary<string, List<string>>();
156
157         public Graph(Dictionary<string, List<string>> graph)
158         {
159             _data = graph;
160         }
161
162         public void AddNode(string name)
163         {
164             if (_data.ContainsKey(name)) throw new GraphException($"Cannot add {name}, node already exists.");
165             _data.Add(name, new List<string>());
166         }
167
168         public void RemoveNode(string name)
169         {
170             if (!_data.ContainsKey(name)) throw new GraphException($"Cannot remove {name}, node does not exist.");
171             _data.Remove(name);
172         }
173
174         public void AddConnection(string node, string name)
175         {
176             if (!_data.ContainsKey(node)) throw new GraphException($"Cannot add connection {name} to {node} original
177             ↪ node does not exist.");
178             if (_data[node].Contains(name)) throw new GraphException($"Cannot add connection {name} to {node} connection
179             ↪ already exists.");
180             _data[node].Add(name);
181         }
182
183         public List<string> GetNode(string node)
184         {
185             if (!_data.ContainsKey(node)) throw new GraphException($"Node {node} does not exist.");
186             return _data[node];
187         }
188
189         public string[] GetAllNodes() => _data.Keys.ToArray();
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-zA-Z]:\\|\\"| [a-zA-Z]|\.\.(\|\|/)|\.(|\|/))(([a-zA-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
37        ← check the path to the file and that exists.");
38        if (!fileRegex.IsMatch(_filePath)) throw new MapFileException("The file which you entered does not appear to be
39        ← a map file. It should end in .vmap double check and try again.");
40
41    private void ReadMapFile(Action updateProgress)
42    {
43        using (BinaryReader br = new BinaryReader(File.Open(_filePath, FileMode.Open)))
44        {
45            string dateTime = br.ReadString();
46            DateTime dt = new DateTime(1970, 1, 1, 0, 0, 0, 0,
47            ← DateTimeKind.Utc).AddMilliseconds(double.Parse(dateTime)).ToLocalTime();
48            TimeCreated = new DateTimeOffset(dt);
49
50            Name = br.ReadString();
51            Description = br.ReadString();
52            Type = br.ReadInt32();
53            IsInverted = br.ReadBoolean();
54
55            int width = (int)br.ReadInt32();
56            int height = (int)br.ReadInt32();
57
58            for (int j = 0; j < 3; j++)
59            {
60                Structures.RGB[,] tempImage = new Structures.RGB[height, width];
61                for (int i = 0; i < 3; i++)
62                {
63                    tempImage[i, j] = new Structures.RGB(br.ReadByte(), br.ReadByte(), br.ReadByte());
64                }
65                Images.Add(tempImage);
66            }
67        }
68    }
69
70    public void Save()
71    {
72        using (BinaryWriter bw = new BinaryWriter(File.Create(_filePath)))
73        {
74            bw.WriteString(TimeCreated.ToString("yyyy-MM-dd HH:mm:ss"));
75            bw.Write(Name);
76            bw.Write(Description);
77            bw.Write(Type);
78            bw.Write(IsInverted);
79
80            bw.Write(OriginalImage.Width);
81            bw.Write(OriginalImage.Height);
82
83            for (int j = 0; j < 3; j++)
84            {
85                for (int i = 0; i < 3; i++)
86                {
87                    bw.Write(OriginalImage[i, j].R);
88                    bw.Write(OriginalImage[i, j].G);
89                    bw.Write(OriginalImage[i, j].B);
90                }
91            }
92        }
93    }
94
95    public void Dispose()
96    {
97        foreach (Bitmap image in Images)
98        {
99            image.Dispose();
100       }
101   }
102 }
```

```

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     if (j == 0) OriginalImage = tempImage.ToBitmap();
73     else if (j == 1) PathImage = tempImage.ToBitmap();
74     else if (j == 2) CombinedImage = tempImage.ToBitmap();
75 }
76 }
77 }
78 }

79 public string Save(Guid currentGuid)
80 {
81     using (BinaryWriter bw = new BinaryWriter(File.Open($"./saves/{currentGuid}.vmap", FileMode.OpenOrCreate)))
82     {
83         bw.Write(TimeCreated.ToUnixTimeMilliseconds().ToString());
84
85         bw.Write(Name);
86         bw.Write(Description);
87         bw.Write(Type);
88         bw.Write(IsInverted);
89
90         bw.Write((int)OriginalImage.Width);
91         bw.Write((int)OriginalImage.Height);
92
93         for (int j = 0; j < 3; j++)
94         {
95             for (int i = 0; i < 3; i++)
96             {
97                 for (int y = 0; y < OriginalImage.Height; y++)
98                 {
99                     for (int x = 0; x < OriginalImage.Width; x++)
100                     {
101                         if (j == 0)
102                         {
103                             if (i == 0) bw.Write(OriginalImage.GetPixel(x, y).R);
104                             else if (i == 1) bw.Write(OriginalImage.GetPixel(x, y).G);
105                             else if (i == 2) bw.Write(OriginalImage.GetPixel(x, y).B);
106                         }
107                         else if (j == 1)
108                         {
109                             if (i == 0) bw.Write(PathImage.GetPixel(x, y).R);
110                             else if (i == 1) bw.Write(PathImage.GetPixel(x, y).G);
111                             else if (i == 2) bw.Write(PathImage.GetPixel(x, y).B);
112                         }
113                         else if (j == 2)
114                         {
115                             if (i == 0) bw.Write(CombinedImage.GetPixel(x, y).R);
116                             else if (i == 1) bw.Write(CombinedImage.GetPixel(x, y).G);
117                             else if (i == 2) bw.Write(CombinedImage.GetPixel(x, y).B);
118                         }
119                     }
120                 }
121             }
122         }
123     }
124 }

```

```

120             }
121         }
122     }
123 }
124 }

126     return $"./saves/{currentGuid}.vmap";
127 }
128 }
129

```

**Traversal.cs**

```

1  public class Traversal<T>
2  {
3      private Graph<T> _graph;
4
5      public Traversal(Graph<T> graph)
6      {
7          _graph = graph;
8      }
9
10     public T[] DFS(T start)
11     {
12         List<T> path = new List<T>();
13         Datatypes.Stack<T> stack = new Datatypes.Stack<T>();
14         Dictionary<T, bool> visited = new Dictionary<T, bool>();
15         foreach (T s in _graph.GetAllNodes()) visited.Add(s, false);
16
17         // Kick Start
18         stack.Push(start);
19
20         while (!stack.IsEmpty())
21         {
22             T node = stack.Pop();
23             path.Add(node);
24             visited[node] = true;
25
26             List<T> connections = _graph.GetNode(node);
27
28             connections.Reverse();
29
30             foreach (T s in connections)
31             {
32                 if (visited[s] == false && !stack.Contains(s))
33                 {
34                     stack.Push(s);
35                 }
36             }
37         }
38
39
40         return path.ToArray();
41     }
42
43     public T[] BFS(T start)
44     {
45         List<T> path = new List<T>();
46         Datatypes.Queue<T> queue = new Datatypes.Queue<T>();
47         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;

```

```

108         else prev.Add(neighbor, current);
109         queue.ChangePriority(neighbor, tentative + weightFunction(neighbor, goal));
110     }
111 }
112 }
113
114
115 return new Dictionary<T, T>();
116 }
117
118 public Dictionary<T, T> Dijkstra(T start, T goal, bool endOnFind, Action nodeUpdate)
119 {
120     Dictionary<T, double> dist = new Dictionary<T, double>();
121     Dictionary<T, T> prev = new Dictionary<T, T>();
122     dist.Add(start, 0);
123
124     MinPriorityQueue<T> queue = new MinPriorityQueue<T>();
125
126     T[] nodes = _graph.GetAllNodes();
127     foreach (T node in nodes)
128     {
129         if (_graph.GetNode(node).Count > 0)
130         {
131             if (!Equals(start, node)) dist.Add(node, double.MaxValue);
132             queue.Enqueue(node, dist[node]);
133         }
134     }
135
136     while (queue.Size > 0)
137     {
138         T minVertex = queue.Dequeue();
139         nodeUpdate();
140         if (Equals(minVertex, goal) && endOnFind) return prev;
141
142         List<T> adjacent = _graph.GetNode(minVertex);
143
144         foreach (var neighbor in adjacent)
145         {
146
147             if (queue.Contains(neighbor))
148             {
149                 double alternateWeight = dist[minVertex] + 1;
150                 if (alternateWeight < dist[neighbor])
151                 {
152                     dist[neighbor] = alternateWeight;
153                     if (prev.ContainsKey(neighbor)) prev[neighbor] = minVertex;
154                     else prev.Add(neighbor, minVertex);
155                     queue.ChangePriority(neighbor, alternateWeight);
156                 }
157             }
158         }
159     }
160
161     return prev;
162 }
163 }
```

### 5.2.1.2 Datatypes

#### Graph.cs

```

1  public class Graph<T>
2  {
3      public Dictionary<T, List<T>> _data = new Dictionary<T, List<T>>();
4
5      public Graph() { }
6
7      public Graph(Dictionary<T, List<T>> graph)
8      {
9          _data = graph;
10     }
11
12     public void AddNode(T key)
13     {
14         if (_data.ContainsKey(key)) throw new GraphException($"Failed to add node {key} to the graph, the node already
15         exists.");
16         _data.Add(key, new List<T>());
17     }
18
19     public void RemoveNode(T key)
20     {
21         if (!_data.ContainsKey(key)) throw new GraphException($"Failed to remove node {key} from the graph, the node
22         does not exist.");
23         _data.Remove(key);
24     }
25
26     public void AddConnection(T key, T value)
27     {
28         if (!_data.ContainsKey(key)) throw new GraphException($"Cannot add connection between {value} and {key} the
29         parent node does not exist in the graph.");
30         if (_data[key].Contains(value)) throw new GraphException($"Cannot add connection between {value} and {key} the
31         connection already exists.");
32         _data[key].Add(value);
33     }
34
35     public List<T> GetNode(T key)
36     {
37         if (!_data.ContainsKey(key)) throw new GraphException($"Failed to get node {key} form graph because it does not
38         exist.");
39         return _data[key];
40     }
41
42     public T[] GetAllNodes() => _data.Keys.ToArray();
43
44     public bool ContainsNode(T node) => _data.ContainsKey(node);
45
46     public void Clear() => _data.Clear();
47 }

```

**Matrix.cs**

```

1  public class Matrix : IEnumerable
2  {
3      private readonly double[,] _matrix;
4      public int X { get; }
5      public int Y { get; }
6
7      public Matrix(double[,] matrix)
8      {
9          _matrix = matrix;
10         X = matrix.GetLength(1);
11     }
12
13     public void Print()
14     {
15         for (int i = 0; i < Y; i++)
16         {
17             for (int j = 0; j < X; j++)
18             {
19                 Console.Write(_matrix[i, j] + " ");
20             }
21             Console.WriteLine();
22         }
23     }
24
25     public double this[int i, int j]
26     {
27         get { return _matrix[i, j]; }
28         set { _matrix[i, j] = value; }
29     }
30
31     public static Matrix operator +(Matrix m1, Matrix m2)
32     {
33         if (m1.X != m2.X || m1.Y != m2.Y)
34             throw new ArgumentException("Matrices must have the same dimensions");
35
36         var result = new Matrix(m1._matrix);
37
38         for (int i = 0; i < m1.Y; i++)
39         {
40             for (int j = 0; j < m1.X; j++)
41             {
42                 result._matrix[i, j] += m2._matrix[i, j];
43             }
44         }
45
46         return result;
47     }
48
49     public static Matrix operator -(Matrix m1, Matrix m2)
50     {
51         if (m1.X != m2.X || m1.Y != m2.Y)
52             throw new ArgumentException("Matrices must have the same dimensions");
53
54         var result = new Matrix(m1._matrix);
55
56         for (int i = 0; i < m1.Y; i++)
57         {
58             for (int j = 0; j < m1.X; j++)
59             {
60                 result._matrix[i, j] -= m2._matrix[i, j];
61             }
62         }
63
64         return result;
65     }
66
67     public static Matrix operator *(Matrix m1, Matrix m2)
68     {
69         if (m1.X != m2.Y)
70             throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
71
72         var result = new Matrix(new double[m1.Y, m2.X]);
73
74         for (int i = 0; i < m1.Y; i++)
75         {
76             for (int j = 0; j < m2.X; j++)
77             {
78                 for (int k = 0; k < m1.X; k++)
79                 {
80                     result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
81                 }
82             }
83         }
84
85         return result;
86     }
87
88     public static Matrix operator /(Matrix m1, double scalar)
89     {
90         var result = new Matrix(m1._matrix);
91
92         for (int i = 0; i < m1.Y; i++)
93         {
94             for (int j = 0; j < m1.X; j++)
95             {
96                 result._matrix[i, j] /= scalar;
97             }
98         }
99
100        return result;
101    }
102
103    public static Matrix operator /(double scalar, Matrix m1)
104    {
105        var result = new Matrix(m1._matrix);
106
107        for (int i = 0; i < m1.Y; i++)
108        {
109            for (int j = 0; j < m1.X; j++)
110            {
111                result._matrix[i, j] /= scalar;
112            }
113        }
114
115        return result;
116    }
117
118    public static Matrix operator *(double scalar, Matrix m1)
119    {
120        var result = new Matrix(m1._matrix);
121
122        for (int i = 0; i < m1.Y; i++)
123        {
124            for (int j = 0; j < m1.X; j++)
125            {
126                result._matrix[i, j] *= scalar;
127            }
128        }
129
130        return result;
131    }
132
133    public static Matrix operator *(Matrix m1, double scalar)
134    {
135        var result = new Matrix(m1._matrix);
136
137        for (int i = 0; i < m1.Y; i++)
138        {
139            for (int j = 0; j < m1.X; j++)
140            {
141                result._matrix[i, j] *= scalar;
142            }
143        }
144
145        return result;
146    }
147
148    public static Matrix operator *(Matrix m1, Matrix m2)
149    {
150        if (m1.X != m2.Y)
151            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
152
153        var result = new Matrix(m1._matrix);
154
155        for (int i = 0; i < m1.Y; i++)
156        {
157            for (int j = 0; j < m2.X; j++)
158            {
159                for (int k = 0; k < m1.X; k++)
160                {
161                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
162                }
163            }
164        }
165
166        return result;
167    }
168
169    public static Matrix operator *(Matrix m1, int scalar)
170    {
171        var result = new Matrix(m1._matrix);
172
173        for (int i = 0; i < m1.Y; i++)
174        {
175            for (int j = 0; j < m1.X; j++)
176            {
177                result._matrix[i, j] *= scalar;
178            }
179        }
180
181        return result;
182    }
183
184    public static Matrix operator *(int scalar, Matrix m1)
185    {
186        var result = new Matrix(m1._matrix);
187
188        for (int i = 0; i < m1.Y; i++)
189        {
190            for (int j = 0; j < m1.X; j++)
191            {
192                result._matrix[i, j] *= scalar;
193            }
194        }
195
196        return result;
197    }
198
199    public static Matrix operator *(Matrix m1, long scalar)
200    {
201        var result = new Matrix(m1._matrix);
202
203        for (int i = 0; i < m1.Y; i++)
204        {
205            for (int j = 0; j < m1.X; j++)
206            {
207                result._matrix[i, j] *= scalar;
208            }
209        }
210
211        return result;
212    }
213
214    public static Matrix operator *(Matrix m1, float scalar)
215    {
216        var result = new Matrix(m1._matrix);
217
218        for (int i = 0; i < m1.Y; i++)
219        {
220            for (int j = 0; j < m1.X; j++)
221            {
222                result._matrix[i, j] *= scalar;
223            }
224        }
225
226        return result;
227    }
228
229    public static Matrix operator *(Matrix m1, double scalar)
230    {
231        var result = new Matrix(m1._matrix);
232
233        for (int i = 0; i < m1.Y; i++)
234        {
235            for (int j = 0; j < m1.X; j++)
236            {
237                result._matrix[i, j] *= scalar;
238            }
239        }
240
241        return result;
242    }
243
244    public static Matrix operator *(Matrix m1, decimal scalar)
245    {
246        var result = new Matrix(m1._matrix);
247
248        for (int i = 0; i < m1.Y; i++)
249        {
250            for (int j = 0; j < m1.X; j++)
251            {
252                result._matrix[i, j] *= scalar;
253            }
254        }
255
256        return result;
257    }
258
259    public static Matrix operator *(Matrix m1, Matrix m2)
260    {
261        if (m1.X != m2.Y)
262            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
263
264        var result = new Matrix(m1._matrix);
265
266        for (int i = 0; i < m1.Y; i++)
267        {
268            for (int j = 0; j < m2.X; j++)
269            {
270                for (int k = 0; k < m1.X; k++)
271                {
272                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
273                }
274            }
275        }
276
277        return result;
278    }
279
280    public static Matrix operator *(Matrix m1, long scalar)
281    {
282        var result = new Matrix(m1._matrix);
283
284        for (int i = 0; i < m1.Y; i++)
285        {
286            for (int j = 0; j < m1.X; j++)
287            {
288                result._matrix[i, j] *= scalar;
289            }
290        }
291
292        return result;
293    }
294
295    public static Matrix operator *(Matrix m1, float scalar)
296    {
297        var result = new Matrix(m1._matrix);
298
299        for (int i = 0; i < m1.Y; i++)
300        {
301            for (int j = 0; j < m1.X; j++)
302            {
303                result._matrix[i, j] *= scalar;
304            }
305        }
306
307        return result;
308    }
309
310    public static Matrix operator *(Matrix m1, double scalar)
311    {
312        var result = new Matrix(m1._matrix);
313
314        for (int i = 0; i < m1.Y; i++)
315        {
316            for (int j = 0; j < m1.X; j++)
317            {
318                result._matrix[i, j] *= scalar;
319            }
320        }
321
322        return result;
323    }
324
325    public static Matrix operator *(Matrix m1, decimal scalar)
326    {
327        var result = new Matrix(m1._matrix);
328
329        for (int i = 0; i < m1.Y; i++)
330        {
331            for (int j = 0; j < m1.X; j++)
332            {
333                result._matrix[i, j] *= scalar;
334            }
335        }
336
337        return result;
338    }
339
340    public static Matrix operator *(Matrix m1, Matrix m2)
341    {
342        if (m1.X != m2.Y)
343            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
344
345        var result = new Matrix(m1._matrix);
346
347        for (int i = 0; i < m1.Y; i++)
348        {
349            for (int j = 0; j < m2.X; j++)
350            {
351                for (int k = 0; k < m1.X; k++)
352                {
353                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
354                }
355            }
356        }
357
358        return result;
359    }
360
361    public static Matrix operator *(Matrix m1, Matrix m2)
362    {
363        if (m1.X != m2.Y)
364            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
365
366        var result = new Matrix(m1._matrix);
367
368        for (int i = 0; i < m1.Y; i++)
369        {
370            for (int j = 0; j < m2.X; j++)
371            {
372                for (int k = 0; k < m1.X; k++)
373                {
374                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
375                }
376            }
377        }
378
379        return result;
380    }
381
382    public static Matrix operator *(Matrix m1, Matrix m2)
383    {
384        if (m1.X != m2.Y)
385            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
386
387        var result = new Matrix(m1._matrix);
388
389        for (int i = 0; i < m1.Y; i++)
390        {
391            for (int j = 0; j < m2.X; j++)
392            {
393                for (int k = 0; k < m1.X; k++)
394                {
395                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
396                }
397            }
398        }
399
400        return result;
401    }
402
403    public static Matrix operator *(Matrix m1, Matrix m2)
404    {
405        if (m1.X != m2.Y)
406            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
407
408        var result = new Matrix(m1._matrix);
409
410        for (int i = 0; i < m1.Y; i++)
411        {
412            for (int j = 0; j < m2.X; j++)
413            {
414                for (int k = 0; k < m1.X; k++)
415                {
416                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
417                }
418            }
419        }
420
421        return result;
422    }
423
424    public static Matrix operator *(Matrix m1, Matrix m2)
425    {
426        if (m1.X != m2.Y)
427            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
428
429        var result = new Matrix(m1._matrix);
430
431        for (int i = 0; i < m1.Y; i++)
432        {
433            for (int j = 0; j < m2.X; j++)
434            {
435                for (int k = 0; k < m1.X; k++)
436                {
437                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
438                }
439            }
440        }
441
442        return result;
443    }
444
445    public static Matrix operator *(Matrix m1, Matrix m2)
446    {
447        if (m1.X != m2.Y)
448            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
449
450        var result = new Matrix(m1._matrix);
451
452        for (int i = 0; i < m1.Y; i++)
453        {
454            for (int j = 0; j < m2.X; j++)
455            {
456                for (int k = 0; k < m1.X; k++)
457                {
458                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
459                }
460            }
461        }
462
463        return result;
464    }
465
466    public static Matrix operator *(Matrix m1, Matrix m2)
467    {
468        if (m1.X != m2.Y)
469            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
470
471        var result = new Matrix(m1._matrix);
472
473        for (int i = 0; i < m1.Y; i++)
474        {
475            for (int j = 0; j < m2.X; j++)
476            {
477                for (int k = 0; k < m1.X; k++)
478                {
479                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
480                }
481            }
482        }
483
484        return result;
485    }
486
487    public static Matrix operator *(Matrix m1, Matrix m2)
488    {
489        if (m1.X != m2.Y)
490            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
491
492        var result = new Matrix(m1._matrix);
493
494        for (int i = 0; i < m1.Y; i++)
495        {
496            for (int j = 0; j < m2.X; j++)
497            {
498                for (int k = 0; k < m1.X; k++)
499                {
500                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
501                }
502            }
503        }
504
505        return result;
506    }
507
508    public static Matrix operator *(Matrix m1, Matrix m2)
509    {
510        if (m1.X != m2.Y)
511            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
512
513        var result = new Matrix(m1._matrix);
514
515        for (int i = 0; i < m1.Y; i++)
516        {
517            for (int j = 0; j < m2.X; j++)
518            {
519                for (int k = 0; k < m1.X; k++)
520                {
521                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
522                }
523            }
524        }
525
526        return result;
527    }
528
529    public static Matrix operator *(Matrix m1, Matrix m2)
530    {
531        if (m1.X != m2.Y)
532            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
533
534        var result = new Matrix(m1._matrix);
535
536        for (int i = 0; i < m1.Y; i++)
537        {
538            for (int j = 0; j < m2.X; j++)
539            {
540                for (int k = 0; k < m1.X; k++)
541                {
542                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
543                }
544            }
545        }
546
547        return result;
548    }
549
550    public static Matrix operator *(Matrix m1, Matrix m2)
551    {
552        if (m1.X != m2.Y)
553            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
554
555        var result = new Matrix(m1._matrix);
556
557        for (int i = 0; i < m1.Y; i++)
558        {
559            for (int j = 0; j < m2.X; j++)
560            {
561                for (int k = 0; k < m1.X; k++)
562                {
563                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
564                }
565            }
566        }
567
568        return result;
569    }
570
571    public static Matrix operator *(Matrix m1, Matrix m2)
572    {
573        if (m1.X != m2.Y)
574            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
575
576        var result = new Matrix(m1._matrix);
577
578        for (int i = 0; i < m1.Y; i++)
579        {
580            for (int j = 0; j < m2.X; j++)
581            {
582                for (int k = 0; k < m1.X; k++)
583                {
584                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
585                }
586            }
587        }
588
589        return result;
590    }
591
592    public static Matrix operator *(Matrix m1, Matrix m2)
593    {
594        if (m1.X != m2.Y)
595            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
596
597        var result = new Matrix(m1._matrix);
598
599        for (int i = 0; i < m1.Y; i++)
600        {
601            for (int j = 0; j < m2.X; j++)
602            {
603                for (int k = 0; k < m1.X; k++)
604                {
605                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
606                }
607            }
608        }
609
610        return result;
611    }
612
613    public static Matrix operator *(Matrix m1, Matrix m2)
614    {
615        if (m1.X != m2.Y)
616            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
617
618        var result = new Matrix(m1._matrix);
619
620        for (int i = 0; i < m1.Y; i++)
621        {
622            for (int j = 0; j < m2.X; j++)
623            {
624                for (int k = 0; k < m1.X; k++)
625                {
626                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
627                }
628            }
629        }
630
631        return result;
632    }
633
634    public static Matrix operator *(Matrix m1, Matrix m2)
635    {
636        if (m1.X != m2.Y)
637            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
638
639        var result = new Matrix(m1._matrix);
640
641        for (int i = 0; i < m1.Y; i++)
642        {
643            for (int j = 0; j < m2.X; j++)
644            {
645                for (int k = 0; k < m1.X; k++)
646                {
647                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
648                }
649            }
650        }
651
652        return result;
653    }
654
655    public static Matrix operator *(Matrix m1, Matrix m2)
656    {
657        if (m1.X != m2.Y)
658            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
659
660        var result = new Matrix(m1._matrix);
661
662        for (int i = 0; i < m1.Y; i++)
663        {
664            for (int j = 0; j < m2.X; j++)
665            {
666                for (int k = 0; k < m1.X; k++)
667                {
668                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
669                }
670            }
671        }
672
673        return result;
674    }
675
676    public static Matrix operator *(Matrix m1, Matrix m2)
677    {
678        if (m1.X != m2.Y)
679            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
680
681        var result = new Matrix(m1._matrix);
682
683        for (int i = 0; i < m1.Y; i++)
684        {
685            for (int j = 0; j < m2.X; j++)
686            {
687                for (int k = 0; k < m1.X; k++)
688                {
689                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
690                }
691            }
692        }
693
694        return result;
695    }
696
697    public static Matrix operator *(Matrix m1, Matrix m2)
698    {
699        if (m1.X != m2.Y)
700            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
701
702        var result = new Matrix(m1._matrix);
703
704        for (int i = 0; i < m1.Y; i++)
705        {
706            for (int j = 0; j < m2.X; j++)
707            {
708                for (int k = 0; k < m1.X; k++)
709                {
710                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
711                }
712            }
713        }
714
715        return result;
716    }
717
718    public static Matrix operator *(Matrix m1, Matrix m2)
719    {
720        if (m1.X != m2.Y)
721            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
722
723        var result = new Matrix(m1._matrix);
724
725        for (int i = 0; i < m1.Y; i++)
726        {
727            for (int j = 0; j < m2.X; j++)
728            {
729                for (int k = 0; k < m1.X; k++)
730                {
731                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
732                }
733            }
734        }
735
736        return result;
737    }
738
739    public static Matrix operator *(Matrix m1, Matrix m2)
740    {
741        if (m1.X != m2.Y)
742            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
743
744        var result = new Matrix(m1._matrix);
745
746        for (int i = 0; i < m1.Y; i++)
747        {
748            for (int j = 0; j < m2.X; j++)
749            {
750                for (int k = 0; k < m1.X; k++)
751                {
752                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
753                }
754            }
755        }
756
757        return result;
758    }
759
760    public static Matrix operator *(Matrix m1, Matrix m2)
761    {
762        if (m1.X != m2.Y)
763            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
764
765        var result = new Matrix(m1._matrix);
766
767        for (int i = 0; i < m1.Y; i++)
768        {
769            for (int j = 0; j < m2.X; j++)
770            {
771                for (int k = 0; k < m1.X; k++)
772                {
773                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
774                }
775            }
776        }
777
778        return result;
779    }
780
781    public static Matrix operator *(Matrix m1, Matrix m2)
782    {
783        if (m1.X != m2.Y)
784            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
785
786        var result = new Matrix(m1._matrix);
787
788        for (int i = 0; i < m1.Y; i++)
789        {
790            for (int j = 0; j < m2.X; j++)
791            {
792                for (int k = 0; k < m1.X; k++)
793                {
794                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
795                }
796            }
797        }
798
799        return result;
800    }
801
802    public static Matrix operator *(Matrix m1, Matrix m2)
803    {
804        if (m1.X != m2.Y)
805            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
806
807        var result = new Matrix(m1._matrix);
808
809        for (int i = 0; i < m1.Y; i++)
810        {
811            for (int j = 0; j < m2.X; j++)
812            {
813                for (int k = 0; k < m1.X; k++)
814                {
815                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
816                }
817            }
818        }
819
820        return result;
821    }
822
823    public static Matrix operator *(Matrix m1, Matrix m2)
824    {
825        if (m1.X != m2.Y)
826            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
827
828        var result = new Matrix(m1._matrix);
829
830        for (int i = 0; i < m1.Y; i++)
831        {
832            for (int j = 0; j < m2.X; j++)
833            {
834                for (int k = 0; k < m1.X; k++)
835                {
836                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
837                }
838            }
839        }
840
841        return result;
842    }
843
844    public static Matrix operator *(Matrix m1, Matrix m2)
845    {
846        if (m1.X != m2.Y)
847            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
848
849        var result = new Matrix(m1._matrix);
850
851        for (int i = 0; i < m1.Y; i++)
852        {
853            for (int j = 0; j < m2.X; j++)
854            {
855                for (int k = 0; k < m1.X; k++)
856                {
857                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
858                }
859            }
860        }
861
862        return result;
863    }
864
865    public static Matrix operator *(Matrix m1, Matrix m2)
866    {
867        if (m1.X != m2.Y)
868            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
869
870        var result = new Matrix(m1._matrix);
871
872        for (int i = 0; i < m1.Y; i++)
873        {
874            for (int j = 0; j < m2.X; j++)
875            {
876                for (int k = 0; k < m1.X; k++)
877                {
878                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
879                }
880            }
881        }
882
883        return result;
884    }
885
886    public static Matrix operator *(Matrix m1, Matrix m2)
887    {
888        if (m1.X != m2.Y)
889            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
890
891        var result = new Matrix(m1._matrix);
892
893        for (int i = 0; i < m1.Y; i++)
894        {
895            for (int j = 0; j < m2.X; j++)
896            {
897                for (int k = 0; k < m1.X; k++)
898                {
899                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
900                }
901            }
902        }
903
904        return result;
905    }
906
907    public static Matrix operator *(Matrix m1, Matrix m2)
908    {
909        if (m1.X != m2.Y)
910            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
911
912        var result = new Matrix(m1._matrix);
913
914        for (int i = 0; i < m1.Y; i++)
915        {
916            for (int j = 0; j < m2.X; j++)
917            {
918                for (int k = 0; k < m1.X; k++)
919                {
920                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
921                }
922            }
923        }
924
925        return result;
926    }
927
928    public static Matrix operator *(Matrix m1, Matrix m2)
929    {
930        if (m1.X != m2.Y)
931            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
932
933        var result = new Matrix(m1._matrix);
934
935        for (int i = 0; i < m1.Y; i++)
936        {
937            for (int j = 0; j < m2.X; j++)
938            {
939                for (int k = 0; k < m1.X; k++)
940                {
941                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
942                }
943            }
944        }
945
946        return result;
947    }
948
949    public static Matrix operator *(Matrix m1, Matrix m2)
950    {
951        if (m1.X != m2.Y)
952            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
953
954        var result = new Matrix(m1._matrix);
955
956        for (int i = 0; i < m1.Y; i++)
957        {
958            for (int j = 0; j < m2.X; j++)
959            {
960                for (int k = 0; k < m1.X; k++)
961                {
962                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
963                }
964            }
965        }
966
967        return result;
968    }
969
970    public static Matrix operator *(Matrix m1, Matrix m2)
971    {
972        if (m1.X != m2.Y)
973            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
974
975        var result = new Matrix(m1._matrix);
976
977        for (int i = 0; i < m1.Y; i++)
978        {
979            for (int j = 0; j < m2.X; j++)
980            {
981                for (int k = 0; k < m1.X; k++)
982                {
983                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
984                }
985            }
986        }
987
988        return result;
989    }
990
991    public static Matrix operator *(Matrix m1, Matrix m2)
992    {
993        if (m1.X != m2.Y)
994            throw new ArgumentException("The number of columns in the first matrix must be equal to the number of rows in the second matrix");
995
996        var result = new Matrix(m1._matrix);
997
998        for (int i = 0; i < m1.Y; i++)
999        {
1000            for (int j = 0; j < m2.X; j++)
1001            {
1002                for (int k = 0; k < m1.X; k++)
1003                {
1004                    result._matrix[i, j] += m1._matrix[i, k] * m2._matrix[k, j];
1005                }
1006            }
1007        }
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
46 public static Matrix operator *(Matrix a, Matrix b)
47 {
48     if (a.X != b.X || a.Y != b.Y) throw new MatrixException("Matrices must be the same dimensions to multiply.");
49
50     Matrix m = new Matrix(a.X, a.Y);
51     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];
52     return m;
53 }
54
55 public static Matrix operator *(int a, Matrix b)
56 {
57     Matrix m = new Matrix(b.X, b.Y);
58     for (int i = 0; i < b.Y; i++) for (int j = 0; j < b.X; j++) m[i, j] = a * b[i, j];
59     return m;
60 }
61
62 public void Minimize()
63 {
64     double sum = 0;
65     foreach (double val in _matrix) sum += val;
66
67     for (int i = 0; i < Y; i++)
68     {
69         for (int j = 0; j < X; j++)
70         {
71             _matrix[i, j] /= sum;
72         }
73     }
74 }

```

```

71         }
72     }
73 }
74
75     public static double Convolution(Matrix a, Matrix b)
76     {
77         if (a.X != b.X || b.Y != a.Y) throw new MatrixException("Matrices must be the same dimensions to apply convolution.");
78
79         double[,] flippedB = new double[b.Y, b.X];
80         int l = b.X;
81         for (int i = l - 1; i >= 0; i--) for (int j = l - 1; j >= 0; j--) flippedB[b.Y - (i + 1), b.X - (j + 1)] = b[i,
82             j];
83
84         double sum = 0;
85         for (int i = 0; i < a.Y; i++) for (int j = 0; j < a.X; j++) sum += a[i, j] * flippedB[i, j];
86         return sum;
87     }
88
89     public IEnumerator GetEnumerator() => _matrix.GetEnumerator();
90 }

```

**MaxPriorityQueue.cs**

```

1  public class MaxPriorityQueue<T>
2  {
3      private List<int> _priorityQueue = new List<int>();
4      private List<T> _queue = new List<T>();
5
6      public int Size => _priorityQueue.Count;
7      private int _size => _priorityQueue.Count - 1;
8
9      public MaxPriorityQueue() { }
10
11     private T GetParent(int index) => _queue[Parent(index)];
12     private int Parent(int index) => (index - 1) / 2;
13
14     private T GetLeftChild(int index) => _queue[LeftChild(index)];
15     private int LeftChild(int index) => (index * 2) + 1;
16
17     private T GetRightChild(int index) => _queue[RightChild(index)];
18     private int RightChild(int index) => (index * 2) + 2;
19
20     private void ShiftNodeUp(int index)
21     {
22         while (index > 0 && _priorityQueue[Parent(index)] < _priorityQueue[index])
23         {
24             Swap(Parent(index), index);
25             index = Parent(index);
26         }
27     }
28
29     public void ChangePriority(T item, int newPriority)
30     {
31         int index = _queue.FindIndex(i => Equals(i, item));
32         int oldPriority = _priorityQueue[index];
33         _priorityQueue[index] = newPriority;
34
35         if (newPriority > oldPriority) ShiftNodeUp(index);

```

```

36         else ShiftNodeDown(index);
37     }
38
39     private void ShiftNodeDown(int index)
40     {
41         int maxIndex = index;
42
43         int left = LeftChild(index);
44         if (left <= _size && _priorityQueue[left] > _priorityQueue[maxIndex]) maxIndex = left;
45
46         int right = RightChild(index);
47         if (right <= _size && _priorityQueue[right] > _priorityQueue[maxIndex]) maxIndex = right;
48
49         if (index != maxIndex)
50         {
51             Swap(index, maxIndex);
52             ShiftNodeDown(maxIndex);
53         }
54     }
55
56     public void Enqueue(T item, int priority)
57     {
58         _queue.Add(item);
59         _priorityQueue.Add(priority);
60
61         ShiftNodeUp(_size);
62     }
63
64     public T Dequeue() => RemoveMax().Item1;
65
66     private (T, int) RemoveMax()
67     {
68         int res = _priorityQueue[0];
69         T result = _queue[0];
70         _priorityQueue.RemoveAt(0);
71         _queue.RemoveAt(0);
72
73         ShiftNodeDown(0);
74
75         return (result, res);
76     }
77
78     private void Swap(int indexX, int indexY)
79     {
80         T tempValue = _queue[indexX];
81         _queue[indexX] = _queue[indexY];
82         _queue[indexY] = tempValue;
83
84         int tempPriority = _priorityQueue[indexX];
85         _priorityQueue[indexX] = _priorityQueue[indexY];
86         _priorityQueue[indexY] = tempPriority;
87     }
88
89     public bool Contains(T neighbor) => _queue.Contains(neighbor);
90 }

```

**MinPriorityQueue.cs**

```

1  public class MinPriorityQueue<T>
2  {

```

```

3   private List<double> _priorityQueue = new List<double>();
4   private List<T> _queue = new List<T>();
5
6   public int Size => _priorityQueue.Count;
7   private int _size => _priorityQueue.Count - 1;
8
9   public MinPriorityQueue() { }
10
11  private int Parent(int index) => (index - 1) / 2;
12  private int Left(int index) => (2 * index) + 1;
13  private int Right(int index) => (2 * index) + 2;
14
15  public void Enqueue(T value, double priority)
16  {
17      int oldSize = Size;
18
19      _queue.Add(value);
20      _priorityQueue.Add(priority);
21
22      while (oldSize != 0 && _priorityQueue[oldSize] < _priorityQueue[Parent(oldSize)])
23      {
24          Swap(oldSize, Parent(oldSize));
25          oldSize = Parent(oldSize);
26      }
27  }
28
29  public void ChangePriority(T item, double newPriority)
30  {
31      int index = _queue.FindIndex(i => Equals(i, item));
32      if (index > -1)
33      {
34          if (_priorityQueue[index] > newPriority)
35          {
36              _priorityQueue[index] = newPriority;
37
38              while (index != 0 && _priorityQueue[index] < _priorityQueue[Parent(index)])
39              {
40                  Swap(index, Parent(index));
41                  index = Parent(index);
42              }
43          }
44          else
45          {
46              _priorityQueue[index] = newPriority;
47              MinifyHeap(index);
48          }
49      }
50  }
51
52  public T Dequeue()
53  {
54      if (Size == 1)
55      {
56          T val = _queue[0];
57
58          _queue.RemoveAt(0);
59          _priorityQueue.RemoveAt(0);
60
61          return val;
62      }
63  }

```

```

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

```

**Queue.cs**

```

1  public class Queue<T>
2  {
3      private List<T> _queue = new List<T>();
4      public int Size => _queue.Count;
5
6      public Queue() { }
7
8      public Queue(IEnumerable<T> input)
9      {
10          foreach (var item in input) _queue.Add(item);
11      }
12

```

```

13     public void Enqueue(T item) => _queue.Add(item);
14
15     public T Dequeue()
16     {
17         T item = _queue[0];
18         _queue.RemoveAt(0);
19         return item;
20     }
21
22     public bool IsEmpty() => _queue.Count == 0;
23
24     public bool Contains(T item) => _queue.Contains(item);
25 }
```

**Stack.cs**

```

1  public class Stack<T>
2  {
3      private List<T> _stack = new List<T>();
4      public int Size => _stack.Count;
5
6      public Stack() { }
7
8      public Stack(IEnumerable<T> input)
9      {
10         foreach (var item in input) _stack.Add(item);
11     }
12     public T Peek() => _stack[_stack.Count - 1];
13
14     public void Push(T item) => _stack.Add(item);
15
16     public T Pop()
17     {
18         T item = _stack[_stack.Count - 1];
19         _stack.RemoveAt(_stack.Count - 1);
20         return item;
21     }
22
23     public bool IsEmpty() => _stack.Count == 0;
24
25     public bool Contains(T item) => _stack.Contains(item);
26 }
```

**5.2.1.3 Exceptions****GraphException.cs**

```

1  [Serializable]
2  public class GraphException : Exception
3  {
4      public GraphException()
5      {
6      }
7
8      public GraphException(string message) : base(message)
9      {
10 }
11
12     public GraphException(string message, Exception innerException) : base(message, innerException)
13 {
```

```

14     }
15
16     protected GraphException(SerializationInfo info, StreamingContext context) : base(info, context)
17     {
18     }
19 }
```

**KernelException.cs**

```

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

**LoggerException.cs**

```

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

**MapFileException.cs**

```

1 [Serializable]
2 public class MapFileException : Exception
3 {
4     public MapFileException()
5     {
6     }
7 }
```

```

8     public MapFileException(string message) : base(message)
9     {
10 }
11
12    public MapFileException(string message, Exception innerException) : base(message, innerException)
13    {
14 }
15
16   protected MapFileException(SerializationInfo info, StreamingContext context) : base(info, context)
17   {
18 }
19 }
```

**MatrixException.cs**

```

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

**PreprocessingException.cs**

```

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

**SettingsException.cs**

```

1 [Serializable]
2 public class SettingsException : Exception
```

```

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

#### 5.2.1.4 Interfaces

##### IHandler.cs

```

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

#### 5.2.1.5 Processing

##### CannyEdgeDetection.cs

```

1  public class CannyEdgeDetection
2  {
3      public int KernelSize { get; set; } = 5;
4      public double RedRatio { get; set; } = 0.299;
5      public double GreenRatio { get; set; } = 0.587;
6      public double BlueRatio { get; set; } = 0.114;
7      public double Sigma { get; set; } = 1.4;
8      public double LowerThreshold { get; set; } = 0.1;
9      public double UpperThreshold { get; set; } = 0.3;
10
11     public CannyEdgeDetection() { }
12
13     public CannyEdgeDetection(int kernelSize, double redRatio, double greenRatio, double blueRatio, double sigma, double
14     ↪ lowerThreshold, double upperThreshold)
15     {
16         KernelSize = kernelSize;
17         RedRatio = redRatio;
18         GreenRatio = greenRatio;
19         BlueRatio = blueRatio;
20         Sigma = sigma;
21         LowerThreshold = lowerThreshold;
22         UpperThreshold = upperThreshold;
23     }
24
25     /// <summary>
26     /// Convert a given image in the form of a RGB double array will convert it to a single double array of black and
27     ↪ white pixels.
28     /// </summary>
29     /// <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) !=
126     → grads.GradientY.GetLength(1))
127         throw new ArgumentException("Canny edge detection failed due to arrays not being of the same size.");
128
129     double[,] output = new double[grads.GradientX.GetLength(0), grads.GradientX.GetLength(1)];
130
131     for (int y = 0; y < grads.GradientX.GetLength(0); y++)
132     {
133         for (int x = 0; x < grads.GradientX.GetLength(1); x++)
134         {
135             output[y, x] = Math.Sqrt(Math.Pow(grads.GradientX[y, x], 2) + Math.Pow(grads.GradientY[y, x], 2));
136         }
137     }
138
139     return output;
140 }
141
142 public double[,] GradientAngle(Structures.Gradients grads)
143 {
144     if (grads.GradientX.GetLength(0) != grads.GradientY.GetLength(0) || grads.GradientX.GetLength(1) !=
145     → grads.GradientY.GetLength(1))
146         throw new ArgumentException("Canny edge detection failed due to arrays not being of the same size.");

```

```

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 =
223             false, Value = input[y, x] };
224             else if (input[y, x] > max) output[y, x] = new Structures.ThresholdPixel { Strong = true, Value =
225             input[y, x] };
226             else throw new Exception("Critical unknown error occurred, please try again.");
227         }
228     }
229
230     return output;
231 }
232
233 public double[,] EdgeTrackingHysteresis(Structures.ThresholdPixel[,] input)
234 {
235     double[,] output = new double[input.GetLength(0), input.GetLength(1)];
236
237     Kernel<Structures.ThresholdPixel> masterKernel = new Kernel<Structures.ThresholdPixel>(input);
238
239     for (int i = 0; i < input.GetLength(0); i++)
240     {
241         for (int j = 0; j < input.GetLength(1); j++)
242         {
243             if (input[i, j].Strong == false)
244             {
245                 Structures.ThresholdPixel[,] imageKernel = masterKernel.Duplication(j, i, 3);
246                 bool strong = false;
247                 for (int k = 0; k < 3 && !strong; k++)
248                 {
249                     for (int l = 0; l < 3 && !strong; l++)
250                     {
251                         if (imageKernel[k, l].Strong) strong = true;
252                     }
253                 }
254                 output[i, j] = strong ? 255 : 0;
255             }
256             else output[i, j] = 255;
257         }
258     }
259
260     return output;
261 }
262 }
```

**Post.cs**

```

1  public class Post
2  {
3      private double[,] _imageDoubles;
4
5      public Post(double[,] input)
6      {
7          _imageDoubles = input;
8      }
9
10     public void Start(int embossCount)
11     {
12         if (embossCount <= 0) _imageDoubles = FillPixelGaps(_imageDoubles);
13         else
14         {
15             for (int i = 0; i < embossCount; i++)
16             {
17                 _imageDoubles = FillPixelGaps(EmbossImage(_imageDoubles));
18             }
19         }
20     }
21
22     private double[,] EmbossImage(double[,] input)
23     {
24         double[,] result = new double[input.GetLength(0), input.GetLength(1)];
25
26         Matrix embossMatrix = new Matrix(new double[,] { { -2, -1, 0 }, { -1, 1, 1 }, { 0, 1, 2 } });
27         Kernel<double> masterKernel = new Kernel<double>(input);
28
29         for (int y = 0; y < input.GetLength(0); y++)
30         {
31             for (int x = 0; x < input.GetLength(1); x++)
32             {
33                 Matrix imageKernel = new Matrix(masterKernel.Duplication(x, y, 3));
34                 result[y, x] = Math.Abs(Matrix.Convolution(imageKernel, embossMatrix));
35             }
36         }
37
38         return result;
39     }
40
41     private double[,] FillPixelGaps(double[,] input)
42     {
43         double[,] output = new double[input.GetLength(0), input.GetLength(1)];
44         Kernel<double> masterKernel = new Kernel<double>(input);
45
46
47         for (int y = 0; y < input.GetLength(0); y++)
48         {
49             for (int x = 0; x < input.GetLength(1); x++)
50             {
51                 Matrix imageKernel = new Matrix(masterKernel.Duplication(x, y, 3));
52                 int count = imageKernel.Cast<double>().Count(value => value >= 255);
53                 if (count > 4) output[y, x] = 255;
54             }
55         }
56
57         return output;
58     }
59
60

```

```
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 =
8     @"^([a-z]:\\|\|\\|[a-z]|\.\.\.(\\|\|/)|\.\.(\\|\|/))((\w|(\|\|/))+)\.(jpg|bmp|exif|png|tiff)$";
9
10    public Pre(string imagePath)
11    {
12        _imagePath = imagePath;
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,
33        double check the path to the file and that exists.");
34        if (!fileRegex.IsMatch(_imagePath)) throw new PreprocessingException("The file which you entered does not appear
35        to be an image file. It should end in .jpg, .bmp, .exif, .png or .tiff double check and try again.");
36
37    private void ReadImage()
38    {
39        _imageBitmap = new Bitmap(_imagePath, true);
40        _imageRgb = new Structures.RGB[_imageBitmap.Height, _imageBitmap.Width];
41
42        for (int y = 0; y < _imageBitmap.Height; y++)
43        {
44            for (int x = 0; x < _imageBitmap.Width; x++)
45            {
46                Color tempPixel = _imageBitmap.GetPixel(x, y);
47                _imageRgb[y, x] = new Structures.RGB
48                {
49                    R = tempPixel.R,
50                    G = tempPixel.G,
51                    B = tempPixel.B
52                };
53            }
54        }
55    }
56}
```

```

52         }
53     }
54 }
55
56     private void CheckDimensions()
57 {
58     if (_imageRgb.GetLength(0) < 200 || _imageRgb.GetLength(1) < 200)
59         throw new PreprocessingException("The image you supplied is too small to work properly it must be at least
→ 200x200. Try a larger image.");
60
61     if (_imageRgb.GetLength(0) % 2 != 0 || _imageRgb.GetLength(1) % 2 != 0)
62     {
63         Structures.RGB[,] resizedRgb =
64             new Structures.RGB[_imageRgb.GetLength(0) / 2 * 2, _imageRgb.GetLength(1) / 2 * 2];
65
66         for (int y = 0; y < _imageRgb.GetLength(0) / 2 * 2; y++)
67         {
68             for (int x = 0; x < _imageRgb.GetLength(1) / 2 * 2; x++)
69             {
70                 resizedRgb[y, x] = _imageRgb[y, x];
71             }
72         }
73
74         _imageRgb = resizedRgb;
75     }
76 }
77
78     public Structures.RawImage Result() => new Structures.RawImage
79 {
80     Original = _imageBitmap,
81     Pixels = _imageRgb,
82     Path = _imagePath,
83     Height = _imageBitmap.Height,
84     Width = _imageBitmap.Width
85 };
86 }
```

**RoadDetection.cs**

```

1  public class RoadDetection
2  {
3      private Bitmap _filledBitmap;
4      private Bitmap _pathBitmap;
5      private double[,] _pathDoubles;
6      private readonly double[,] _imageDoubles;
7      private readonly double _threshold;
8      private Random _gen = new Random();
9
10     public RoadDetection(double[,] imageDoubles, double threshold)
11     {
12         _imageDoubles = imageDoubles;
13         _threshold = threshold;
14     }
15
16     public void Start(Action updateAction)
17     {
18         List<Color> toRemoveColors = FillImage(updateAction);
19         RemoveColor(toRemoveColors, updateAction);
20
21         _pathDoubles = new double[_pathBitmap.Height, _pathBitmap.Width];
```

```

22     for (int y = 0; y < _pathBitmap.Height; y++)
23     {
24         for (int x = 0; x < _pathBitmap.Width; x++)
25         {
26             Color pixel = _pathBitmap.GetPixel(x, y);
27             if (pixel == Color.FromArgb(0, 0, 0)) _pathDoubles[y, x] = 0;
28             else _pathDoubles[y, x] = 255;
29         }
30     }
31 }
32
33 private List<Color> FillImage(Action updateAction)
34 {
35     Color[,] tempImage = new Color[_imageDoubles.GetLength(0), _imageDoubles.GetLength(1)];
36
37     for (int y = 0; y < _imageDoubles.GetLength(0); y++)
38         for (int x = 0; x < _imageDoubles.GetLength(1); x++)
39             tempImage[y, x] = Color.FromArgb((int)_imageDoubles[y, x], (int)_imageDoubles[y, x],
40             (int)_imageDoubles[y, x]);
41
42     List<Color> toReplaceColors = new List<Color>();
43     List<Color> usedColors = new List<Color>();
44
45     _filledBitmap = _imageDoubles.ToBitmap();
46
47     for (int y = 0; y < _imageDoubles.GetLength(0); y++)
48     {
49         for (int x = 0; x < _imageDoubles.GetLength(1); x++)
50         {
51             if (((y + 1) * (x + 1)) / 100 % 100 == 0) updateAction();
52
53             int minX = _imageDoubles.GetLength(1), maxX = 0, minY = _imageDoubles.GetLength(0), maxY = 0;
54             int filled = 0;
55
56             Color randCol = Color.FromArgb(_gen.Next(56, 256), _gen.Next(56, 256), _gen.Next(56, 256));
57             while (usedColors.Contains(randCol))
58                 randCol = Color.FromArgb(_gen.Next(56, 256), _gen.Next(56, 256), _gen.Next(56, 256));
59
60             Datatypes.Queue<(int, int)> queue = new Datatypes.Queue<(int, int)>();
61             queue.Enqueue((y, x));
62
63             while (queue.Size > 0)
64             {
65                 (int, int) cord = queue.Dequeue();
66                 if (tempImage[cord.Item1, cord.Item2] == Color.FromArgb(0, 0, 0))
67                 {
68                     tempImage[cord.Item1, cord.Item2] = randCol;
69                     _filledBitmap.SetPixel(cord.Item2, cord.Item1, tempImage[cord.Item1, cord.Item2]);
70
71                     if (cord.Item1 > 0) queue.Enqueue((cord.Item1 - 1, cord.Item2));
72                     if (cord.Item2 > 0) queue.Enqueue((cord.Item1, cord.Item2 - 1));
73                     if (cord.Item1 < _filledBitmap.Height - 1) queue.Enqueue((cord.Item1 + 1, cord.Item2));
74                     if (cord.Item2 < _filledBitmap.Width - 1) queue.Enqueue((cord.Item1, cord.Item2 + 1));
75
76                     if (!usedColors.Contains(randCol)) usedColors.Add(randCol);
77
78                     filled++;
79                 }
80                 else if (tempImage[cord.Item1, cord.Item2] == Color.FromArgb(255, 255, 255))
81             {

```

```

81         tempImage[cord.Item1, cord.Item2] = Color.FromArgb(1, 1, 1);
82         _filledBitmap.SetPixel(cord.Item2, cord.Item1, tempImage[cord.Item1, cord.Item2]);
83     }
84
85     if (cord.Item1 > maxY) maxY = cord.Item1;
86     if (cord.Item2 > maxX) maxX = cord.Item2;
87     if (cord.Item1 < minY) minY = cord.Item1;
88     if (cord.Item2 < minX) minX = cord.Item2;
89   }
90
91   double totalSquares = (maxX - minX) * (maxY - minY);
92   if (filled / totalSquares > _threshold || filled == 1) toReplaceColors.Add(randCol);
93 }
94 }
95
96 return toReplaceColors;
97 }
98
99 private void RemoveColor(List<Color> toRemove, Action updateAction)
100 {
101   _pathBitmap = new Bitmap(_filledBitmap);
102
103   for (int y = 0; y < _pathBitmap.Height; y++)
104   {
105     for (int x = 0; x < _pathBitmap.Width; x++)
106     {
107       if (((y + 1) * (x + 1)) / 100 % 100 == 0) updateAction();
108       if (toRemove.Contains(_pathBitmap.GetPixel(x, y)))
109       {
110         _pathBitmap.SetPixel(x, y, Color.FromArgb(1, 1, 1));
111       }
112     }
113   }
114
115   for (int i = 0; i < _pathBitmap.Height; i++)
116   {
117     for (int j = 0; j < _pathBitmap.Width; j++)
118     {
119       if (((i + 1) * (j + 1)) / 100 % 100 == 0) updateAction();
120       if (_pathBitmap.GetPixel(j, i) == Color.FromArgb(1, 1, 1))
121         _pathBitmap.SetPixel(j, i, Color.FromArgb(0, 0, 0));
122     }
123   }
124 }
125
126 public Structures.RoadResult Result() => new Structures.RoadResult
127 {
128   FilledBitmap = _filledBitmap,
129   PathBitmap = _pathBitmap,
130   PathDoubles = _pathDoubles
131 };
132 }

```

### 5.2.1.6 Root

#### Extensions.cs

```

1  public static class Extensions
2  {
3    public static Bitmap ToBitmap(this double[,] array)
4    {

```

```

5     Bitmap output = new Bitmap(array.GetLength(1), array.GetLength(0));
6
7     for (int y = 0; y < array.GetLength(0); y++)
8     {
9         for (int x = 0; x < array.GetLength(1); x++)
10        {
11            int boundedPixel = (int)Utility.Bound(0, 255, array[y, x]);
12            output.SetPixel(x, y, Color.FromArgb(boundedPixel, boundedPixel, boundedPixel));
13        }
14    }
15
16    return output;
17 }
18
19 public static double[,] ToDoubles(this Bitmap image, Func<Color, double> getPixelFunction)
20 {
21     double[,] result = new double[image.Height, image.Width];
22
23     for (int y = 0; y < image.Height; y++)
24     {
25         for (int x = 0; x < image.Width; x++)
26         {
27             result[y, x] = getPixelFunction(image.GetPixel(x, y));
28         }
29     }
30
31     return result;
32 }
33
34 public static Bitmap ToBitmap(this Structures.RGB[,] array)
35 {
36     Bitmap output = new Bitmap(array.GetLength(1), array.GetLength(0));
37
38     for (int y = 0; y < array.GetLength(0); y++)
39     {
40         for (int x = 0; x < array.GetLength(1); x++)
41         {
42             output.SetPixel(x, y, Color.FromArgb((int)array[y, x].R, (int)array[y, x].G, (int)array[y, x].B));
43         }
44     }
45
46     return output;
47 }
48
49 public static Graph<Structures.Coord> ToGraph(this double[,] doubles)
50 {
51     Graph<Structures.Coord> output = new Graph<Structures.Coord>();
52     Kernel<double> masterKernel = new Kernel<double>(doubles);
53
54     for (int y = 0; y < doubles.GetLength(0); y++)
55     {
56         for (int x = 0; x < doubles.GetLength(1); x++)
57         {
58             Structures.Coord tempCoord = new Structures.Coord { X = x, Y = y };
59             output.AddNode(tempCoord);
60
61             double[,] surroundingDoubles = masterKernel.Constant(x, y, 3, 0);
62
63             bool found = false;
64

```

```
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

```
1 public class Kernel<T>
2 {
3     private readonly T[,] _image;
4     private readonly int _width;
5     private readonly int _height;
6
7     public Kernel(T[,] image)
8     {
9         _image = image;
10        _height = image.GetLength(0);
11        _width = image.GetLength(1);
12    }
13
14    public T[,] Constant(int x, int y, int size, T constant = default)
15    {
16        if (size % 2 != 1) throw new KernelException("The image kernel supplied was of an odd size, check your settings
17 ← and try again.");
18        if (x >= _width || x < 0 || y >= _height || y < 0)
19            throw new KernelException("Your kernel must start within the image.");
20
21        T[,] kernel = new T[size, size];
22
23        int halfK = size / 2;
24
25        for (int i = 0; i < size; i++)
26            for (int j = 0; j < size; j++)
27                kernel[i, j] = constant;
28
29        int cnty = 0;
30        for (int j = y - halfK; j <= y + halfK; j++)
31        {
32            for (int i = x - halfK; i <= x + halfK; i++)
33            {
34                kernel[i, j] = constant;
35            }
36        }
37    }
38}
```

```

31     int cntX = 0;
32     for (int i = x - halfK; i <= x + halfK; i++)
33     {
34         if (j >= 0 && i >= 0 && j < _height && i < _image.GetLength(1))
35         {
36             kernel[cntY, cntX] = _image[j, i];
37         }
38         cntX++;
39     }
40     cntY++;
41 }
42
43     return kernel;
44 }
45
46 public T[,] Duplication(int x, int y, int size)
47 {
48     if (size % 2 != 1) throw new KernelException("The image kernel supplied was of an odd size, check your settings
49 ← and try again.");
50     if (x >= _width || x < 0 || y >= _height || y < 0)
51         throw new KernelException("Your kernel must start within the image.");
52
53     T[,] kernel = new T[size, size];
54
55     int halfK = size / 2;
56
57     for (int i = 0; i < size; i++) for (int j = 0; j < size; j++) kernel[i, j] = _image[y, x];
58
59     int cntY = 0;
60     for (int j = y - halfK; j <= y + halfK; j++)
61     {
62         int cntX = 0;
63         for (int i = x - halfK; i <= x + halfK; i++)
64         {
65             if (j >= 0 && i >= 0 && j < _height && i < _image.GetLength(1))
66             {
67                 kernel[cntY, cntX] = _image[j, i];
68             }
69             cntX++;
70         }
71         cntY++;
72     }
73
74     return kernel;
75 }
76
77 public static double[,] Gaussian(double sigma, int size)
78 {
79     double[,] result = new double[size, size];
80     int halfK = size / 2;
81
82     double sum = 0;
83
84     int cntY = -halfK;
85     for (int i = 0; i < size; i++)
86     {
87         int cntX = -halfK;
88         for (int j = 0; j < size; j++)
89         {
90             result[halfK + cntY, halfK + cntX] = Utility.GaussianDistribution(cntX, cntY, sigma);

```

```

90         sum += result[halfK + cntY, halfK + cntX];
91         cntX++;
92     }
93     cntY++;
94 }
95
96 for (int i = 0; i < size; i++) for (int j = 0; j < size; j++) result[i, j] /= sum;
97 return result;
98 }
99
100}

```

**Logger.cs**

```

1 public class Logger
2 {
3     private readonly bool _localApplication;
4     private static readonly object Lock = new object();
5     public Logger(bool local)
6     {
7         _localApplication = local;
8         CreateDirStructure();
9     }
10
11     private void CreateDirStructure()
12     {
13         Directory.CreateDirectory("./runs");
14         Directory.CreateDirectory("./logs");
15         Directory.CreateDirectory("./saves");
16
17         string mode = _localApplication ? "Local Application" : "Web Application";
18
19         lock (Lock)
20         {
21             using (StreamWriter sr = File.AppendText("./logs/master.txt"))
22             {
23                 sr.WriteLine("<===== New Instance =====>");
24                 sr.WriteLine($"Datetime: {DateTime.UtcNow:dd-MM-yyyy} {DateTime.UtcNow:HH:mm:ss}");
25                 sr.WriteLine($"Mode: {mode}");
26             }
27         }
28     }
29
30     public static Guid CreateRun()
31     {
32         Guid guidForRun = Uuid();
33
34         Directory.CreateDirectory($"./runs/{guidForRun.ToString("N").ToUpper()}");
35
36         WriteLineToRunFile(guidForRun, "<===== Begin New Run =====>");
37         WriteLineToRunFile(guidForRun, $"Datetime: {DateTime.UtcNow:dd-MM-yyyy} {DateTime.UtcNow:HH:mm:ss}");
38         WriteLineToRunFile(guidForRun, $"Run Object Guid: {guidForRun.ToString().ToUpper()}");
39
40         WriteLineToMaster($"New Run Started with GUID {guidForRun.ToString().ToUpper()}");
41
42         return guidForRun;
43     }
44
45     public static void WriteLineToRunFile(Guid currentGuid, string message)
46     {

```

```

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;

```

```

20     }
21
22     public struct RawImage
23     {
24         public Bitmap Original;
25         public string Path;
26         public RGB[,] Pixels;
27         public int Width;
28         public int Height;
29         public MapFile MapFile;
30     }
31
32     public struct RoadResult
33     {
34         public Bitmap FilledBitmap;
35         public Bitmap PathBitmap;
36         public double[,] PathDoubles;
37     }
38
39     public struct CannyResult
40     {
41         public Bitmap BitmapImage;
42         public double[,] DoubleImage;
43     }
44
45     public struct Coord
46     {
47         public int X;
48         public int Y;
49
50         public override string ToString() => $"({X}, {Y})";
51         public bool Equals(Coord other) => X == other.X && Y == other.Y;
52         public override bool Equals(object obj) => obj is Coord other && Equals(other);
53         public static bool operator ==(Coord lhs, Coord rhs) => lhs.X == rhs.X && lhs.Y == rhs.Y;
54         public static bool operator !=(Coord lhs, Coord rhs) => !(lhs == rhs);
55         public override int GetHashCode()
56         {
57             unchecked
58             {
59                 return (X * 397) ^ Y;
60             }
61         }
62     }
63 }
64

```

**Utility.cs**

```

1  public static class Utility
2  {
3      public static double GaussianDistribution(int x, int y, double sigma) =>
4          1 / (2 * Math.PI * sigma * sigma) * Math.Exp(-((Math.Pow(x, 2) + Math.Pow(y, 2)) / (2 * sigma * sigma)));
5
6      public static double Bound(int l, int h, double v) => v > h ? h : v < l ? l : v;
7
8      public static bool TryBound(int l, int h, double v, out double value)
9      {
10         if (v < h && v > l) value = v;
11         else value = v > h ? h : l;
12         return v < h && v > l;
13     }
14 }

```

```

13 }
14
15     public static double RadianToDegree(double input) => 180 * input / Math.PI;
16
17     public static double DegreeToRadian(double input) => input * Math.PI / 180;
18
19     public static double MapRadiansToPixel(double input) => (int)(128 / (2 * Math.PI) * input + 128);
20
21     public static Bitmap CombineBitmap(Bitmap a, Bitmap b)
22 {
23         if (a.Width != b.Width || a.Height != b.Height)
24             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.");
25
26         Bitmap result = new Bitmap(a);
27         for (int y = 0; y < a.Height; y++)
28         {
29             for (int x = 0; x < a.Width; x++)
30             {
31                 Color pixel = b.GetPixel(x, y);
32                 if (pixel != Color.FromArgb(0, 0, 0))
33                 {
34                     result.SetPixel(x, y, pixel);
35                 }
36             }
37         }
38
39         return result;
40     }
41
42     public static Structures.RGB[][] SplitImage(Structures.RGB[,] image)
43 {
44         Structures.RGB[,] one = new Structures.RGB[image.GetLength(0) / 2, image.GetLength(1) / 2];
45         Structures.RGB[,] beta = new Structures.RGB[image.GetLength(0) / 2, image.GetLength(1) / 2];
46         Structures.RGB[,] gamma = new Structures.RGB[image.GetLength(0) / 2, image.GetLength(1) / 2];
47         Structures.RGB[,] delta = new Structures.RGB[image.GetLength(0) / 2, image.GetLength(1) / 2];
48
49         for (int i = 0; i < image.GetLength(1) / 2; i++)
50         {
51             for (int j = 0; j < image.GetLength(0) / 2; j++)
52             {
53                 one.SetPixel(i, j, image.GetPixel(i, j));
54             }
55         }
56
57         for (int i = image.GetLength(1) / 2; i < image.GetLength(1); i++)
58         {
59             for (int j = 0; j < image.GetLength(0) / 2; j++)
60             {
61                 beta.SetPixel(i - (image.GetLength(1) / 2), j, image.GetPixel(i, j));
62             }
63         }
64
65         for (int i = 0; i < image.GetLength(1) / 2; i++)
66         {
67             for (int j = image.GetLength(0) / 2; j < image.GetLength(0); j++)
68             {
69                 gamma.SetPixel(i, j - (image.GetLength(0) / 2), image.GetPixel(i, j));
70             }
71         }
72     }

```

```

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     {

```

```

132     for (int x = 0; x < image.GetLength(1); x++)
133     {
134         image[y, x] = image[y, x] == 255 ? 0 : 255;
135     }
136 }
137
138 return image;
139 }
140
141 public static T[] RebuildPath<T>(Dictionary<T, T> prev, T goal)
142 {
143     if (prev == null) return new T[1];
144     List<T> sequence = new List<T>();
145     T u = goal;
146
147     while (prev.ContainsKey(u))
148     {
149         sequence.Insert(0, u);
150         u = prev[u];
151     }
152
153     return sequence.ToArray();
154 }
155
156
157 public static bool IsYes(string input) => new Regex(@"^y(es)?$", RegexOptions.IgnoreCase).IsMatch(input.Trim());
158 public static double GetRed(Color pixel) => pixel.R;
159 public static double GetGreen(Color pixel) => pixel.G;
160 public static double GetBlue(Color pixel) => pixel.B;
161 public static double GetAverage(Color pixel) => (pixel.R + pixel.G + pixel.B) / 3.0;
162 public static double GetIndustryAverage(Color pixel) => (pixel.R * 0.299) + (pixel.G * 0.586) + (pixel.B * 0.114);
163 public static double GetIfExists(Color pixel) => GetAverage(pixel) > 0 ? 255 : 0;
164
165 public static double GetDistanceBetweenNodes(Structures.Coord a, Structures.Coord b) =>
166     Math.Sqrt(Math.Pow(a.X - b.X, 2) + Math.Pow(a.Y - b.Y, 2));
167 }

```

## 5.2.2 LocalApp

### 5.2.2.1 Actions

#### NewImage.cs

```

1 internal class NewImage
2 {
3     private readonly Guid _runGuid;
4     private readonly Menu _menuInstance;
5     private readonly Log _logInstance;
6
7     public NewImage(Menu menu, Log logger, Guid runGuid)
8     {
9         _runGuid = runGuid;
10        _menuInstance = menu;
11        _logInstance = logger;
12    }
13
14     public Structures.RawImage Read()
15     {
16         Input inputHandle = new Input(_menuInstance);
17
18         string path =

```

```

19         inputHandel.GetInput(
20             "Please enter the path of the image you wish to process into a map (you can click and drag an image from
21             your file explorer here too):");
22         _logInstance.Event(_runGuid, $"Looking for image at {path}");
23
24         Pre preProcess = new Pre(path);
25
26         ProgressBar progressBar = new ProgressBar("Pre-processing your image", 4, _menuInstance);
27         progressBar.DisplayProgress();
28
29         try
30         {
31             preProcess.Start(progressBar.GetIncrementAction());
32             _logInstance.Event(_runGuid, "Completed pre processing of image.");
33         }
34         catch (PreprocessingException ex)
35         {
36             _logInstance.Error(_runGuid, ex.Message);
37             throw new Exception("An expected occurred while pre processing your image.", ex);
38         }
39         catch (Exception ex)
40         {
41             _logInstance.Error(ex.Message);
42             throw new Exception("An unexpected occurred while pre processing your image.", ex);
43         }
44
45         _menuInstance.ClearUserSection();
46
47         bool saveAsBinary =
48             Utility.IsTrue(
49                 inputHandel.TryGetInput(
50                     "Would you like to save this map afterwards in a file to be reused later (y/n)?"));
51         MapFile mapSave = saveAsBinary ? new MapFile() : null;
52
53         if (saveAsBinary)
54         {
55             mapSave.Type = inputHandel.GetOption("What type of image are you supplying:",
56                 new[] { "Screenshot", "Hand Drawn", "Photograph", "Other" });
57
58             mapSave.Name = inputHandel.TryGetInput("Enter a name for image, or leave blank for 'None':");
59             _menuInstance.WriteLine();
60
61             mapSave.Description = inputHandel.TryGetInput("Enter a brief description about this image, or leave blank
62             for 'None':");
63
64             Structures.RawImage result = preProcess.Result();
65             if (saveAsBinary) result.MapFile = mapSave;
66             else result.MapFile = null;
67             if (saveAsBinary) mapSave.OriginalImage = result.Pixels.ToBitmap();
68
69             return result;
70         }
71     }

```

**SaveFile.cs**

```

1  public class SaveFile
2  {
3      private readonly Guid _runGuid;

```

```

4     private readonly Menu _menuInstance;
5     private readonly Log _logInstance;
6
7     public SaveFile(Menu menu, Log logger, Guid runGuid)
8     {
9         _runGuid = runGuid;
10        _menuInstance = menu;
11        _logInstance = logger;
12    }
13
14    public MapFile Read()
15    {
16        Input inputHandle = new Input(_menuInstance);
17
18        string path = inputHandle.GetInput("Please enter the path of the map which you wish to recall:");
19        _logInstance.Event(_runGuid, $"Looking for map file at {path}");
20
21        ProgressBar progressBar = new ProgressBar("Recalling Saved Map File", 10, _menuInstance);
22        progressBar.DisplayProgress();
23
24        MapFile result = new MapFile(path);
25
26        try
27        {
28            result.Initialize(progressBar.GetIncrementAction());
29            _logInstance.Event(_runGuid, "Completed recollection.");
30        }
31        catch (MapFileNotFoundException ex)
32        {
33            _logInstance.Error(_runGuid, ex.Message);
34            throw new Exception("An expected occurred while recalling your save file.", ex);
35        }
36        catch (Exception ex)
37        {
38            _logInstance.Error(ex.Message);
39            throw new Exception("An unexpected occurred while recalling your save file.", ex);
40        }
41
42
43        return result;
44    }
45
46 }

```

**SettingsControl.cs**

```

1  public class SettingsControl
2  {
3      private readonly Settings _settings;
4      private readonly Menu _menuInstance;
5      private readonly Log _logInstance;
6      private readonly Input _inputHandle;
7
8      private readonly Dictionary<string, (string, Type)> _oldSettings;
9
10     public SettingsControl(Settings settings, Menu menuInstance, Log logInstance)
11     {
12         _settings = settings;
13         _menuInstance = menuInstance;
14         _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:",
55             new string[] {
56                 "Dijkstra",
57                 "AStar"
58             });
59
60             string newValue = algorithmOption == 0 ? "Dijkstra" : "AStar";
61
62             _settings.Change("pathfindingAlgorithm", newValue);
63             break;
64         default:
65             running = false;
66
67             _settings.Update(_oldSettings, Settings.UserSettings);
68
69             break;
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($" {option.Item1} [{Log.Green}x{Log.Blue}]");
117             else Console.WriteLine($" {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             {

```

```

141         Console.CursorLeft = 1;
142         Console.CursorTop = currentTop;
143         Console.Write(' ');
144         Console.CursorTop = ++currentTop;
145         Console.CursorLeft = 1;
146         Console.WriteLine('>');
147     }
148 }
149 else if (key.Key == ConsoleKey.UpArrow && currentTop > 3)
150 {
151     lock (_menuInstance.ScreenLock)
152     {
153         Console.CursorLeft = 1;
154         Console.CursorTop = currentTop;
155         Console.Write(' ');
156         Console.CursorTop = --currentTop;
157         Console.CursorLeft = 1;
158         Console.WriteLine('>');
159     }
160 }
161 else if (key.Key == ConsoleKey.Enter || key.Key == ConsoleKey.Spacebar)
162 {
163     if (result.Count + 2 == currentTop)
164     {
165         if (clear) _menuInstance.ClearUserSection();
166         Console.CursorVisible = false;
167
168         selected = true;
169     }
170     else
171     {
172         result[currentTop - 3] = (result[currentTop - 3].Item1, !result[currentTop - 3].Item2);
173         Console.SetCursorPosition(1, currentTop);
174         if (result[currentTop - 3].Item2) Console.WriteLine($"> {result[currentTop - 3].Item1}
→ [{Log.Green}x{Log.Blue}]{")});
175             else Console.WriteLine($"> {result[currentTop - 3].Item1} [ ]");
176         }
177     }
178 }
179
180 return result;
181 }
182
183
184 public string GetInput(string prompt)
185 {
186     while (Console.KeyAvailable) Console.ReadKey(true);
187     _menuInstance.WriteLine(prompt);
188
189     bool complete = false;
190     StringBuilder input = new StringBuilder();
191     int line = _menuInstance.CurrentLine;
192
193     while (!complete)
194     {
195         if (Console.KeyAvailable)
196         {
197             ConsoleKeyInfo key = Console.ReadKey(true);
198             switch (key.Key)
199             {

```

```

200     case ConsoleKey.Enter:
201         complete = true;
202         break;
203     case ConsoleKey.Backspace:
204     case ConsoleKey.Delete:
205     {
206         if (input.Length > 0)
207         {
208             lock (_menuInstance.ScreenLock)
209             {
210                 Console.SetCursorPosition((input.Length % (Console.WindowWidth * 3 / 4 - 1)), line);
211                 Console.Write(' ');
212             }
213
214             input.Remove(input.Length - 1, 1);
215         }
216
217         break;
218     }
219     default:
220     {
221         if (input.Length / (line - 1) > Console.WindowWidth * 3 / 4 - 2) line++;
222
223         lock (_menuInstance.ScreenLock)
224         {
225             Console.SetCursorPosition((input.Length % (Console.WindowWidth * 3 / 4 - 1)) + 1, line);
226             Console.Write(key.KeyChar);
227         }
228
229         input.Append(key.KeyChar);
230         break;
231     }
232 }
233 }
234 }
235
236 _menuInstance.WriteLine();
237
238 return input.ToString();
239 }
240
241 public string TryGetInput(string prompt)
242 {
243     string res = GetInput(prompt);
244     return res.Length == 0 ? "None" : res;
245 }
246
247 public double GetDouble(string prompt) => double.Parse(GetInput(prompt));
248
249 public bool TryGetDouble(string prompt, out double result) => double.TryParse(GetInput(prompt), out result);
250
251 public int GetInt(string prompt) => int.Parse(GetInput(prompt));
252
253 public bool TryGetInt(string prompt, out int result) => int.TryParse(GetInput(prompt), out result);
254 }
```

**Log.cs**

```

1 public class Log
2 {
```

```

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, $"Run Guid {runGuid} Deleted. See {Environment.CurrentDirectory}\\saves\\ for output(s) and
38      {Environment.CurrentDirectory}\\runs\\{runGuid.ToString("N").ToUpper()} for temp images.", true);
39      End($"Completed run {runGuid} successfully.");
40  }
41
42  public void EndSuccessSave(Guid runGuid)
43  {
44      End(runGuid, "Successfully completed recall and pathfinding of save file!", true);
45      Warn(runGuid, $"Run Guid {runGuid} Deleted. See {Environment.CurrentDirectory}\\saves\\ for output(s). Or just
46      go to where the save file was located.", true);
47      End($"Completed run {runGuid} successfully.");
48  }
49
50  public Log(Menu menuInstance)
51  {
52      _menuInstance = menuInstance;
53      _ = new Logger(true);
54  }
55
56  /// <summary>
57  /// </summary>
58  /// <param name="message"></param>
59  /// <param name="type">0 - Error, 1 - Warning, 2 - Event, 3 - End</param>
60  private void LogParent(Guid runGuid, string message, int type, bool detailed)
61  {

```

```

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}",
65     → $"{Purple}END{Log.Blank}" };
66     string[] filePrefix = { "[ERROR] ", "[WARN] ", "[EVENT] ", "[END] " };
67
68     lock (_menuInstance.ScreenLock)
69     {
70         CheckLogLineCount();
71
72         if (message.Length > Console.WindowWidth / 4 - 7)
73         {
74             Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 2, _logLineCount++);
75             int i = 10;
76
77             Console.Write($"{prefix[type]}: ");
78
79             foreach (char letter in message)
80             {
81                 Console.Write(letter);
82                 i++;
83                 if (i > Console.WindowWidth / 4)
84                 {
85                     if (CheckLogLineCount()) return;
86                     Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 9, _logLineCount++);
87                     i = 10;
88                 }
89             }
90         }
91         else
92         {
93             Console.SetCursorPosition(Console.WindowWidth * 3 / 4 + 2, _logLineCount++);
94             Console.Write($"{prefix[type]}: {message}");
95         }
96     }
97
98     Logger.WriteLineToRunFile(runGuid, $"{filePrefix[type]>{message}}");
99 }
100
101 // Make sure that the total log lines does not exceed the space given
102 private bool CheckLogLineCount()
103 {
104     if (_logLineCount >= Console.WindowHeight)
105     {
106         _logLineCount = 6;
107         _menuInstance.ClearLogSection();
108
109         return true;
110     }
111
112     return false;
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 >=
14    → Console.LargestWindowWidth - 3;
15
16    private readonly string _permLineA;
17    private readonly string _permLineB;
18
19    public const char VerticalChar = '|';
20    public const char HorizontalChar = '-';
21
22    public Menu(string permLineA, string permLineB)
23    {
24        IntPtr handle = GetStdHandle(-11);
25        GetConsoleMode(handle, out var mode);
26        SetConsoleMode(handle, mode | 0x4);
27
28        int width = Console.WindowWidth / 2;
29        int height = Console.WindowHeight / 4;
30        Console.SetWindowSize(width, height);
31        Console.SetBufferSize(width, height);
32
33        _permLineA = permLineA;
34        _permLineB = permLineB;
35
36        Console.Clear();
37        Console.CursorVisible = false;
38    }
39
40    public void Setup()
41    {
42        while (!IsWindowMax())
43        {
44            Console.SetCursorPosition(0, 0);
45            Console.WriteLine($"{Log.Red}Maximize Window To Continue{Log.Blue}");
46            System.Threading.Thread.Sleep(250);
47            Console.SetCursorPosition(0, 0);
48            Console.WriteLine($""\x1b[48;5;196mMaximize Window To Continue{Log.Blue}");
49            System.Threading.Thread.Sleep(250);
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.WriteLine($"{sw.Elapsed.Hours}:{sw.Elapsed.Minutes}:{sw.Elapsed.Seconds}".PadRight(10, ' '));
121                Console.CursorVisible = false;
122            }
123        }
124    }

```

```
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.WriteLine($"{Log.Red}Something went wrong, to see what take a look below.{Log.Blue}");
228     Console.SetCursorPosition((widthStart * 3) - 8, (int)(heightStart * 1.5) - 3);
229     Console.WriteLine("Reason for Error");
230     for (int i = 0; i < messages.Count; i++)
231     {
232         Console.SetCursorPosition((widthStart * 3) - messages[i].Count / 2, (int)(heightStart * 1.5) - (2 - i));
233         Console.WriteLine($"{Log.Blue}{string.Join("", messages[i])}{Log.Blue}");
234     }
235     Console.SetCursorPosition((widthStart * 3) - 18, heightStart * 2 - 2);
236     Console.WriteLine($"{Log.Grey}(Press Enter to Return to Main Menu){Log.Blue}");
237 }
238
239 }
240
241 public void WriteLine(string message)

```

```

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

```

**ProgressBar.cs**

```

1  public class ProgressBar
2  {
3      private readonly string _progressTitle;
4      private double _progressAmount;
5      private readonly double _progressInterval;
6      private readonly string _progressOutline;
7      private string _progressLine;
8
9      private readonly Menu _menuInstance;
10
11     public ProgressBar(string title, int totalSegments, Menu menuInstance)
12     {
13         _progressInterval = (double)1 / totalSegments;
14         _progressAmount = 0;
15
16         StringBuilder bar = new StringBuilder();
17         bar.Append('+');
18         for (int i = 0; i < (Console.WindowWidth * 3 / 4) - 4; i++) bar.Append(Menu.HorizontalChar);
19         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.WriteLine(_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.WriteLine($"{Log.Blue}{_progressLine}{Log.White}");
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
27     public Settings(Menu menu, Log log)
28     {
29         _menuInstance = menu;
30         _loggerInstance = log;
31     }
32
33     public void CheckIfExistsOrCreate()
34     {
35         if (!File.Exists("settings.conf"))
36         {
37             _loggerInstance.Event("Settings file did not exist. Creating...");
38             using (TextWriter tw = File.CreateText("settings.conf"))
39             {
40                 foreach (string line in defaultSettings)
41                 {
42                     tw.WriteLine(line);
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;

```

```

67     }
68
69     private Dictionary<string, (string, Type)> ConvertSettingsToPairs(List<string> parsedLines)
70     {
71         Dictionary<string, (string, Type)> pairs = new Dictionary<string, (string, Type)>();
72         foreach (string item in parsedLines)
73         {
74             string name = item.Split('=')[0].Trim();
75             string value = item.Split('=')[1].Trim();
76             if (bool.TryParse(value, out bool _)) pairs.Add(name, (value, typeof(bool)));
77             else if (int.TryParse(value, out int _)) pairs.Add(name, (value, typeof(int)));
78             else if (double.TryParse(value, out double _)) pairs.Add(name, (value, typeof(double)));
79             else pairs.Add(name, (value, typeof(string)));
80         }
81
82         return pairs;
83     }
84
85     public bool Change(string setting, bool value)
86     {
87         if (!UserSettings.ContainsKey(setting)) return false;
88         UserSettings[setting] = (value.ToString().ToLower(), typeof(bool));
89
90         return true;
91     }
92
93     public bool Change(string setting, int value)
94     {
95         if (!UserSettings.ContainsKey(setting)) return false;
96         UserSettings[setting] = (value.ToString(), typeof(int));
97
98         return true;
99     }
100
101    public bool Change(string setting, double value)
102    {
103        if (!UserSettings.ContainsKey(setting)) return false;
104        UserSettings[setting] = (value.ToString(), typeof(double));
105
106        return true;
107    }
108
109    public bool Change(string setting, string value)
110    {
111        if (!UserSettings.ContainsKey(setting)) return false;
112        UserSettings[setting] = (value.ToString(), typeof(string));
113
114        return true;
115    }
116
117    public void Read()
118    {
119        CheckIfExistsOrCreate();
120        List<string> parsedLines = ParseSettingsFile();
121        Dictionary<string, (string, Type)> settingValuePairs = ConvertSettingsToPairs(parsedLines);
122        UserSettings = settingValuePairs;
123    }
124
125    public void Update(Dictionary<string, (string, Type)> oldSettings, Dictionary<string, (string, Type)> newSettings)
126    {

```

```

127     if (oldSettings.Count != newSettings.Count) throw new SettingsException("Cannot set settings when the amount of
128     ← settings has changed, if this problem persists delete settings.conf and restart the program.");
129
130     foreach (KeyValuePair<string, (string, Type)> pair in newSettings)
131     {
132         int location = rawLines.FindIndex(toCheck => toCheck.Contains(pair.Key));
133         if (location == -1) throw new SettingsException($"You have an unknown setting {pair.Key}, if this problem
134         ← persists delete settings.conf and restart the program.");
135         else
136         {
137             if (!oldSettings.ContainsKey(pair.Key)) throw new SettingsException($"Setting {pair.Key} does not exist,
138             ← if this problem persists delete settings.conf and restart the program.");
139             if (!oldSettings[pair.Key].Equals(pair.Value)) rawLines[location] = $"{pair.Key}= {pair.Value.Item1}";
140         }
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
6          ← you will be asked to enter the file / the path to it. After that you will have several options open to you:");
7          menuInstance.WriteLine();
8          menuInstance.WriteLine("1. You can choose to modify the file parameters, i.e. Name, Description or Type");
9          menuInstance.WriteLine("2. Delete the file");
10         menuInstance.WriteLine("3. Clone the file");
11         menuInstance.WriteLine("4. Rename the file");
12         menuInstance.WriteLine("5. View current file stats");
13         menuInstance.WriteLine("6. Run pathfinding on the image");
14     }
15
16     public static void ImageWelcome(Menu menuInstance)
17     {
18         menuInstance.WriteLine("You have selected to read a new image and turn it into a route-able map, during this the
19         ← following steps will occur:");
20         menuInstance.WriteLine();
21         menuInstance.WriteLine("1. You will be asked to supply an image to process.");
22         menuInstance.WriteLine("2. The image will be checked to make sure it is valid, if it is not you will have to
23         ← pick another and start again.");
24         menuInstance.WriteLine("3. You will be shown the image to check if it is the right one, as well as some file
25         ← details about it. You can chose to end here if you wish.");
26         menuInstance.WriteLine("4. You will have some options as to how to pick out the roads. There are some presets as
27         ← well as a step by step version.");
28     }
29 }
```

```

23     menuInstance.WriteLine("5. After the roads have been picked out you will be able to click on different points");
24     menuInstance.WriteLine("and find the most efficient route through them.");
25   }
26
27   public static void FileDetails(Menu menuInstance, Structures.RawImage rawImage)
28   {
29     menuInstance.WriteLine("Your image file information:");
30     menuInstance.WriteLine($"      Name of image: {Log.Green}{Path.GetFileNameWithoutExtension(rawImage.Path)}{Log.Blue}");
31     menuInstance.WriteLine($"      Folder it's contained within: {Log.Green}{{Path.GetDirectoryName(rawImage.Path) == "" ? "/" : Path.GetDirectoryName(rawImage.Path)}}{Log.Blue}");
32     menuInstance.WriteLine($"      Type of image: {Log.Green}{Path.GetExtension(rawImage.Path)}{Log.Blue}");
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
18   lowerThreshold, double upperThreshold, int loopCount)
19   {
20     _logInstance.Event(_runGuid, $"Running preset with values {{Log.Orange}}{kernelSize}{Log.Blue}, {{Log.Orange}}{redRatio}{Log.Blue}, {{Log.Orange}}{greenRatio}{Log.Blue}, {{Log.Orange}}{blueRatio}{Log.Blue}, {{Log.Orange}}{sigma}{Log.Blue}, {{Log.Orange}}{lowerThreshold}{Log.Blue}, {{Log.Orange}}{upperThreshold}{Log.Blue}, {{Log.Orange}}{loopCount}{Log.Blue})");
21
22     CannyEdgeDetection detector = new CannyEdgeDetection(kernelSize, redRatio, greenRatio, blueRatio, sigma,
23               lowerThreshold, upperThreshold);
24
25     Input inputHandle = new Input(_menuInstance);
26
27     Structures.RGB[,] quads = Utility.SplitImage(_image.Pixels);
28     Task<double[,]>[] threads = new Task<double[,]>[quads.Length];
29
30     int continueOption = inputHandle.GetOption("Continue to Canny Edge Detection:", new[] { "Yes", "No" });
31     if (continueOption != 0) throw new Exception("You asked for the processing of your map to stop.");
32
33     bool saveTempOption = inputHandle.GetOption("Would you like to save images at each step of the edge detection?", new[] { "Yes", "No" }) == 0;
34

```

```

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

```

```

91         task.Start();
92         threads[i] = task;
93     }
94
95     Task.WaitAll(threads);
96     double[,] cannyImage = Utility.CombineQuadrants(threads[0].Result, threads[1].Result, threads[2].Result,
97     threads[3].Result);
98
99     PostProcessImage(cannyImage, inputHandel);
100 }
101
102 private void PostProcessImage(double[,] image, Input inputHandel)
103 {
104     int timeApproximation = 5;
105     Post postProcessor = new Post(image);
106
107     _menuInstance.ClearUserSection();
108     if (inputHandel.TryGetInt("How many times would you like to emboss the image (can be 0): ", out int loopCount)
109     && loopCount > 0)
110     {
111         _menuInstance.WriteLine();
112         _menuInstance.WriteLine($"Running image embossing this will take approximately {Log.Red}{timeApproximation * 
113         loopCount}{Log.Blank} seconds!");
114         postProcessor.Start(loopCount);
115     }
116     else
117     {
118         _menuInstance.WriteLine();
119         _menuInstance.WriteLine($"Running image embossing this will take approximately
120         {Log.Red}{timeApproximation}{Log.Blank} seconds!");
121         postProcessor.Start(0);
122     }
123 }
124
125 private double[,] RunDetectionOnQuadrant(CannyEdgeDetection detector, Structures.RGB[,] image, int id, Action
126 increment, bool saveTemp)
127 {
128     char letter = (char)('A' + id);
129     double[,] workingArray;
130     _logInstance.Event(_runGuid, $"Starting processing of quadrant {letter} ({id % 2}, {id / 2})");
131
132     workingArray = detector.BlackWhiteFilter(image);
133     if (saveTemp) Logger.SaveBitmap(_runGuid, workingArray, $"BlackWhiteFilterQuad{letter}");
134     increment();
135     _logInstance.Event(_runGuid, $"Completed Black and White Filter on Quadrant {letter}");
136
137     workingArray = detector.GaussianFilter(workingArray);
138     if (saveTemp) Logger.SaveBitmap(_runGuid, workingArray, $"GaussianFilterQuad{letter}");
139     increment();
140     _logInstance.Event(_runGuid, $"Applied Gaussian Filter on Quadrant {letter}");
141
142     Structures.Gradients grads = detector.CalculateGradients(workingArray, increment);
143     if (saveTemp)
144     {
145         Logger.SaveBitmap(_runGuid, grads.GradientX, $"GradientXQuad{letter}");
146         Logger.SaveBitmap(_runGuid, grads.GradientY, $"GradientYQuad{letter}");
147     }

```

```

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,
183                         Color.Red);
184                     else toSave.SetPixel(x, y, Color.Black);
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:
205         {cannyDetection.RedRatio}, Range: 0 <= x <= 1)",
```

```

205         out double newRedRatio) && newRedRatio <= 1 && newRedRatio >= 0 && newRedRatio != 
206     ← cannyDetection.RedRatio)
207     {
208         l.Warn(_runGuid, $"Changed red ratio {cannyDetection.RedRatio} -> {newRedRatio}");
209         m.WriteLine($"[{Log.Green}]Changed: {cannyDetection.RedRatio} -> {newRedRatio}[{Log.Blue}]");
210         cannyDetection.RedRatio = newRedRatio;
211     }
212     else m.WriteLine($"[{Log.Orange}]Kept Default: {cannyDetection.RedRatio}[{Log.Blue}]");
213     m.WriteLine();
214
215     if (i.TryGetDouble(
216         $"Enter a value for the ratio value for green for the Black and White filter (Default:
217     ← {cannyDetection.GreenRatio}, Range: 0 <= x <= 1)",
218         out double newGreenRatio) && newGreenRatio <= 1 && newGreenRatio >= 0 &&
219         newGreenRatio != cannyDetection.GreenRatio)
220     {
221         l.Warn(_runGuid, $"Changed green ratio {cannyDetection.GreenRatio} -> {newGreenRatio}");
222         m.WriteLine($"[{Log.Green}]Changed: {cannyDetection.GreenRatio} -> {newGreenRatio}[{Log.Blue}]");
223         cannyDetection.GreenRatio = newGreenRatio;
224     }
225     else m.WriteLine($"[{Log.Orange}]Kept Default: {cannyDetection.GreenRatio}[{Log.Blue}]");
226     m.WriteLine();
227
228     if (i.TryGetDouble(
229         $"Enter a value for the ratio value for blue for the Black and White filter (Default:
230     ← {cannyDetection.BlueRatio}, Range: 0 <= x <= 1)",
231         out double newBlueRatio) && newBlueRatio <= 1 && newBlueRatio >= 0 && newBlueRatio != 
232     ← cannyDetection.BlueRatio)
233     {
234         l.Warn(_runGuid, $"Changed blue ratio {cannyDetection.BlueRatio} -> {newBlueRatio}");
235         m.WriteLine($"[{Log.Green}]Changed: {cannyDetection.BlueRatio} -> {newBlueRatio}[{Log.Blue}]");
236         cannyDetection.BlueRatio = newBlueRatio;
237     }
238     else m.WriteLine($"[{Log.Orange}]Kept Default: {cannyDetection.BlueRatio}[{Log.Blue}]");
239     m.WriteLine();
240
241     if (i.TryGetDouble(
242         $"Enter a value for sigma for the Gaussian Filter stage (Default: {cannyDetection.Sigma}, Range: 0 <
243     ← x <= 10)",
244         out double newSigma) && newSigma <= 10 && newSigma > 0 && newSigma != cannyDetection.Sigma)
245     {
246         l.Warn(_runGuid, $"Changed sigma value {cannyDetection.Sigma} -> {newSigma}");
247         m.WriteLine($"[{Log.Green}]Changed: {cannyDetection.Sigma} -> {newSigma}[{Log.Blue}]");
248         cannyDetection.Sigma = newSigma;
249     }
250     else m.WriteLine($"[{Log.Orange}]Kept Default: {cannyDetection.Sigma}[{Log.Blue}]");
251     m.WriteLine();
252
253     if (i.TryGetInt(
254         $"Enter a value for kernel size for the Gaussian Filter stage, large values will take exponentially
255     ← longer (Default: {cannyDetection.KernelSize}, Range: x >= 3, x not a multiple of 2 and a whole number)",
256         out int newKernel) && newKernel >= 3 && newKernel % 2 == 1 && newKernel % 1 == 0 && newKernel != 
257     ← cannyDetection.KernelSize)
258     {
259         l.Warn(_runGuid, $"Changed kernel size {cannyDetection.KernelSize} -> {newKernel}");
260         m.WriteLine($"[{Log.Green}]Changed: {cannyDetection.KernelSize} -> {newKernel}[{Log.Blue}]");
261         cannyDetection.KernelSize = newKernel;
262     }
263     else m.WriteLine($"[{Log.Orange}]Kept Default: {cannyDetection.KernelSize}[{Log.Blue}]");
264     m.WriteLine();

```

```

258
259     if (i.TryGetDouble(
260         $"Enter a value for the lower threshold for the Min Max stage (Default:
261         {cannyDetection.LowerThreshold}, Range: 0 <= x < 1)",
262         out double newLowerThreshold) && newLowerThreshold > 0 && newLowerThreshold < 1 && newLowerThreshold != 
263         cannyDetection.LowerThreshold)
264     {
265         l.Warn(_runGuid, $"Changed lower threshold {cannyDetection.LowerThreshold} -> {newLowerThreshold}");
266         m.WriteLine($"{{Log.Green}}Changed: {cannyDetection.LowerThreshold} -> {newLowerThreshold}{{Log.Blue}}");
267         cannyDetection.LowerThreshold = newLowerThreshold;
268     }
269     else m.WriteLine($"{{Log.Orange}}Kept Default: {cannyDetection.LowerThreshold}{{Log.Blue}}");
270     m.WriteLine();
271
272     if (i.TryGetDouble(
273         $"Enter a value for the lower threshold for the Min Max stage (Default:
274         {cannyDetection.UpperThreshold}, Range: {cannyDetection.LowerThreshold} < x <= 1)",
275         out double newHigherThreshold) && newHigherThreshold > cannyDetection.LowerThreshold &&
276         newHigherThreshold <= 1 && newHigherThreshold != cannyDetection.UpperThreshold)
277     {
278         l.Warn(_runGuid, $"Changed upper threshold {cannyDetection.UpperThreshold} -> {newHigherThreshold}");
279         m.WriteLine($"{{Log.Green}}Changed: {cannyDetection.UpperThreshold} -> {newHigherThreshold}{{Log.Blue}}");
280         cannyDetection.UpperThreshold = newHigherThreshold;
281     }
282     else m.WriteLine($"{{Log.Orange}}Kept Default: {cannyDetection.UpperThreshold}{{Log.Blue}}");
283     m.WriteLine();
284
285     i.WaitInput($"{{Log.Grey}}(Enter to continue){Log.Blue}");
286     m.ClearUserSection();
287
288     m.WriteLine("For reference the variables which will be used are:");
289     m.WriteLine($"    Red Ratio: {{Log.Green}}{cannyDetection.RedRatio}{Log.Blue}");
290     m.WriteLine($"    Green Ratio: {{Log.Green}}{cannyDetection.GreenRatio}{Log.Blue}");
291     m.WriteLine($"    Blue Ratio: {{Log.Green}}{cannyDetection.BlueRatio}{Log.Blue}");
292     m.WriteLine($"    Gaussian Sigma Value: {{Log.Green}}{cannyDetection.Sigma}{Log.Blue}");
293     m.WriteLine($"    Gaussian Kernel Size: {{Log.Green}}{cannyDetection.KernelSize}{Log.Blue}");
294     m.WriteLine($"    Double Threshold Lower: {{Log.Green}}{cannyDetection.LowerThreshold}{Log.Blue}");
295     m.WriteLine($"    Double Threshold Upper: {{Log.Green}}{cannyDetection.UpperThreshold}{Log.Blue}");
296     m.WriteLine();
297
298     return cannyDetection;
299 }
300
301 public double[,] Result() => _resultArray;
302 }
```

**Pathfinder.cs**

```

1  public class Pathfinder
2  {
3      private readonly double[,] _input;
4      private readonly Bitmap _originalBitmap;
5
6      private Graph<Structures.Coord> _graph;
7      private Traversal<Structures.Coord> _traversal;
8
9      public Pathfinder(Bitmap originalImage, double[,] input)
10     {
11         _originalBitmap = originalImage;
12         _input = input;
13     }
14 }
```

```

13     }
14
15     public void Start()
16     {
17         InstanceClasses();
18
19         PathfindImageForm pathfindForm = new PathfindImageForm(_originalBitmap, _traversal, _graph);
20         pathfindForm.ShowDialog();
21     }
22
23     private void InstanceClasses()
24     {
25         _graph = _input.ToGraph();
26         _traversal = new Traversal<Structures.Coord>(_graph);
27     }
28 }
```

**RoadSequence.cs**

```

1 internal class RoadSequence
2 {
3     private readonly Menu _menuInstance;
4     private readonly Log _logInstance;
5     private readonly Guid _runGuid;
6     private double[,] _cannyEdgeDetectionResult;
7     private readonly MapFile _saveFile;
8     private Structures.RoadResult _roadResult;
9
10    public RoadSequence(Menu menuInstance, Log logInstance, Guid currentGuid, double[,] cannyResult, MapFile saveFile)
11    {
12        _menuInstance = menuInstance;
13        _logInstance = logInstance;
14        _runGuid = currentGuid;
15        _cannyEdgeDetectionResult = cannyResult;
16        _saveFile = saveFile;
17        _roadResult = new Structures.RoadResult();
18    }
19
20    public Structures.RoadResult Result() => _roadResult;
21
22    public void Start()
23    {
24        Input inputHandel = new Input(_menuInstance);
25
26        InvertImage(inputHandel);
27
28        DetectRoads(inputHandel);
29
30        if (_saveFile != null)
31        {
32            _saveFile.PathImage = new Bitmap(_roadResult.PathBitmap);
33            _saveFile.CombinedImage = Utility.CombineBitmap(_saveFile.OriginalImage, _roadResult.PathBitmap);
34            string path = _saveFile.Save(_runGuid);
35
36            string saveName = _runGuid.ToString();
37
38            if (bool.Parse(Settings.UserSettings["shortNames"]
39                           .Item1))
40            {
41
```

```

42         saveName = _saveFile.Name.Replace(' ', '_');
43         File.Move(path,
44             path.Replace(Path.GetFileName(path)
45                 .Split('.')[0],
46                 saveName));
47     }
48
49     if (bool.Parse(Settings.UserSettings["zipOnComplete"]
50         .Item1))
51     {
52         Directory.CreateDirectory("temp");
53         Directory.CreateDirectory("temp/images");
54
55         string[] files = Directory.GetFiles($"./runs/{_runGuid.ToString("N").ToUpper()}", "*.*",
56         SearchOption.AllDirectories);
57         foreach (string newPath in files)
58         {
59             File.Copy(newPath, newPath.Replace($"./runs/{_runGuid.ToString("N").ToUpper()}", "temp/images"));
60
61             File.Copy("./logs/{_runGuid}.txt", "temp/log.txt");
62             File.Copy("./saves/{saveName}.vmap", "temp/map.vmap");
63             ZipFile.CreateFromDirectory("temp", $"run-{_runGuid}.zip");
64             Directory.Delete("temp", true);
65         }
66     }
67 }
68
69 private void InvertImage(Input inputHandel)
70 {
71     bool invert = Utility.IsTrue(inputHandel.GetInput("Invert image (y/n)?"));
72     if (invert)
73     {
74         _cannyEdgeDetectionResult = Utility.InverseImage(_cannyEdgeDetectionResult);
75         ViewImageForm invertImageForm = new ViewImageForm(_cannyEdgeDetectionResult.ToBitmap());
76         invertImageForm.ShowDialog();
77         if (_saveFile != null) _saveFile.IsInverted = true;
78     }
79     if (_saveFile != null) _saveFile.IsInverted = false;
80
81     _menuInstance.WriteLine();
82 }
83
84 private void DetectRoads(Input inputHandel)
85 {
86     bool happy = true;
87
88     double threshold = 0.3;
89
90     while (happy)
91     {
92         if (inputHandel.TryGetDouble(
93             $"Value for Threshold (Default: {threshold}, Range: 0 <= x < 1)",
94             out double newThreshold) && newThreshold > 0 && newThreshold < 1 && newThreshold != threshold)
95         {
96             _logInstance.Warn(_runGuid, $"Changed threshold {threshold} -> {newThreshold}");
97             _menuInstance.WriteLine($"{Log.Green}Changed: {threshold} -> {newThreshold}{Log.Blue}");
98             threshold = newThreshold;
99         }
100        else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {threshold}{Log.Blue}");

```

```

101     _menuInstance.WriteLine();
102
103     RoadDetection roadDetector = new RoadDetection(_cannyEdgeDetectionResult, threshold);
104     ProgressBar pb = new ProgressBar("Road Detection", _cannyEdgeDetectionResult.Length / 100 * 3,
105     ↪ _menuInstance);
106
107     pb.DisplayProgress();
108     roadDetector.Start(pb.GetIncrementAction());
109
110     _roadResult = roadDetector.Result();
111     ViewImageForm roadForm = new ViewImageForm(_roadResult.PathBitmap);
112     roadForm.ShowDialog();
113
114     _menuInstance.ClearUserSection();
115
116     if (Utility.IsYes(inputHandle.GetInput("Are you happy with this lower threshold you should see your roads,
117     ↪ if you don't try decreasing the threshold if you see too much then increase the threshold. (y/n)?")))
118     ↪ happy = false;
119   }
120 }
```

**SyncEdgeDetection.cs**

```

1  internal class SyncEdgeDetection : IHandler
2  {
3    private readonly Menu _menuInstance;
4    private readonly Log _logInstance;
5    private Input _classInputHandle;
6    private readonly Structures.RawImage _image;
7    private readonly Guid _runGuid;
8    private double[,] _workingArray;
9    private double[,] _resultArray;
10   private CannyEdgeDetection _detector;
11
12   public SyncEdgeDetection(Menu menu, Log logger, Structures.RawImage image, Guid currentGuid)
13   {
14     _menuInstance = menu;
15     _logInstance = logger;
16     _image = image;
17     _runGuid = currentGuid;
18   }
19
20   public void Start()
21   {
22     _classInputHandle = new Input(_menuInstance);
23     _detector = new CannyEdgeDetection();
24
25     ShowDialog();
26     BlackWhiteStep();
27     GaussianStep();
28
29     _menuInstance.WriteLine("The next 5 steps don't require any parameters, you will still see the result of each
30     ↪ step however, in the order of:");
31     _menuInstance.WriteLine("  1. Gradient in X");
32     _menuInstance.WriteLine("  2. Gradient in Y");
33     _menuInstance.WriteLine("  3. Combined Gradients");
34     _menuInstance.WriteLine("  4. Gradient Directions");
35     _menuInstance.WriteLine("  5. Magnitude Threshold");
36     _menuInstance.WriteLine();
37     _classInputHandle.WaitForInput($"{Log.Grey}(Enter to Continue){Log.Blue}");
```

```

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
67     it occurs.");
68     _menuInstance.WriteLine();
69     _classInputHandel.WaitForInput("${Log.Grey}(Enter to Continue){Log.Blank}");
70
71     _workingArray = _detector.EdgeTrackingHysteresis(_thresholdPixels);
72     ViewImageForm edgeTrackingForm = new ViewImageForm(_workingArray.ToBitmap());
73     edgeTrackingForm.ShowDialog();
74
75     PostProcessImage(_workingArray);
76 }
77
78 private void PostProcessImage(double[,] image)
79 {
80     Post postProcessor = new Post(image);
81
82     _menuInstance.ClearUserSection();
83     if (_classInputHandel.TryGetInt("How many times would you like to emboss the image (can be 0): ", out int
84     loopCount) &&
85         loopCount > 0)
86     {
87         _menuInstance.WriteLine();
88         _menuInstance.WriteLine($"Running image embossing this will take approximately {Log.Red}{10 * 
89         loopCount}{Log.Blank} seconds!");
90         postProcessor.Start(loopCount);
91     }
92     else
93     {
94         _menuInstance.WriteLine();
95         _menuInstance.WriteLine($"Running image embossing this will take approximately {Log.Red}10{Log.Blank}
96         seconds!");
97     }

```

```

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
105     two parameters a lower and upper threshold.");
106
107     while (!happy)
108     {
109         if (_classInputHandle.TryGetDouble(
110             $"Value for Lower Threshold (Default: {_detector.LowerThreshold}, Range: 0 <= x < 1)",
111             out double newLowerThreshold) && newLowerThreshold > 0 && newLowerThreshold < 1 && newLowerThreshold
112             != _detector.LowerThreshold)
113         {
114             _logInstance.Warn(_runGuid, $"Changed lower threshold {_detector.LowerThreshold} ->
115             {newLowerThreshold}");
116             _menuInstance.WriteLine($"'{Log.Green}Changed: {_detector.LowerThreshold} ->
117             {newLowerThreshold}{Log.Blank}'");
118             _detector.LowerThreshold = newLowerThreshold;
119         }
120         else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.LowerThreshold}{Log.Blank}");
121         _menuInstance.WriteLine();
122
123         if (_classInputHandle.TryGetDouble(
124             $"Value for Upper Threshold (Default: {_detector.UpperThreshold}, Range: {_detector.LowerThreshold}
125             < x <= 1}",
126             out double newHigherThreshold) && newHigherThreshold > _detector.LowerThreshold &&
127             newHigherThreshold <= 1 && newHigherThreshold != _detector.UpperThreshold)
128         {
129             _logInstance.Warn(_runGuid, $"Changed upper threshold {_detector.UpperThreshold} ->
130             {newHigherThreshold}");
131             _menuInstance.WriteLine($"'{Log.Green}Changed: {_detector.UpperThreshold} ->
132             {newHigherThreshold}{Log.Blank}'");
133             _detector.UpperThreshold = newHigherThreshold;
134         }
135         else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.UpperThreshold}{Log.Blank}");
136         _menuInstance.WriteLine();
137         _menuInstance.WriteLine();
138         _menuInstance.WriteLine("Applying Double Threshold. This may take some time...");
```

```

139
140         _workingThresholdPixels = _detector.DoubleThreshold(_workingArray);
141         Bitmap toView = new Bitmap(_workingThresholdPixels.GetLength(1), _workingThresholdPixels.GetLength(0));
142         for (int y = 0; y < _workingThresholdPixels.GetLength(0); y++)
143         {
144             for (int x = 0; x < _workingThresholdPixels.GetLength(1); x++)
145             {
146                 if (_workingThresholdPixels[y, x].Strong) toView.SetPixel(x, y, Color.Green);
147                 else if (!_workingThresholdPixels[y, x].Strong && _workingThresholdPixels[y, x].Value != 0)
148                     toView.SetPixel(x, y, Color.Red);
149                 else toView.SetPixel(x, y, Color.Black);
150             }
151         }
152         ViewImageForm gaussianForm = new ViewImageForm(toView);
153     }
154 }
```

```

144     _menuInstance.ClearUserSection();
145     gaussianForm.ShowDialog();
146
147     _menuInstance.WriteLine("Current values for thresholds");
148     _menuInstance.WriteLine($"Lower: {_detector.LowerThreshold}");
149     _menuInstance.WriteLine($"Upper: {_detector.UpperThreshold}");
150     _menuInstance.WriteLine();
151
152     string opt = _classInputHandel.GetInput("Are you happy with these values for the upper and lower threshold
153     ↪ (y/n)?");
154
155     if (opt.ToLower().StartsWith("y")) happy = true;
156     else
157     {
158         _menuInstance.ClearUserSection();
159         _menuInstance.WriteLine($"{Log.Pink}Please re-enter your values.{Log.Blue}");
160     }
161
162     _menuInstance.ClearUserSection();
163     return _workingThresholdPixels;
164 }
165
166 private void GaussianStep()
167 {
168     bool happy = false;
169
170     _menuInstance.WriteLine($"The second stage of Canny Edge Detection is applying a Gaussian filter. It is made up
171     ↪ of two parameters sigma and kernel size.");
172
173     while (!happy)
174     {
175         if (_classInputHandel.TryGetDouble(
176             $"Value for Sigma (Default: {_detector.Sigma}, Range: 0 < x <= 10)",
177             out double newSigma) && newSigma <= 10 && newSigma > 0 && newSigma != _detector.Sigma)
178         {
179             _logInstance.Warn(_runGuid, $"Changed Sigma value {_detector.Sigma} -> {newSigma}");
180             _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.Sigma} -> {newSigma}{Log.Blue}");
181             _detector.Sigma = newSigma;
182         }
183         else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.Sigma}{Log.Blue}");
184         _menuInstance.WriteLine();
185
186         if (_classInputHandel.TryGetInt(
187             $"Value for Kernel Size (Default: {_detector.KernelSize}, Range: x >= 3, x not a multiple of 2 and a
188             ↪ whole number)",
189             out int newKernel) && newKernel >= 3 && newKernel % 2 == 1 && newKernel % 1 == 0 && newKernel !=
190             _detector.KernelSize)
191         {
192             _logInstance.Warn(_runGuid, $"Changed Kernel Size {_detector.KernelSize} -> {newKernel}");
193             _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.KernelSize} -> {newKernel}{Log.Blue}");
194             _detector.KernelSize = newKernel;
195         }
196         else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.KernelSize}{Log.Blue}");
197         _menuInstance.WriteLine();
198         _menuInstance.WriteLine("Applying Gaussian Filter. This may take some time...");
```

```

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($"{Log.Pink}Please re-enter your values.{Log.Blue}");
214 }
215 }
216
217 _menuInstance.ClearUserSection();
218 }
219
220 private void BlackWhiteStep()
221 {
222     bool happy = false;
223
224     _menuInstance.WriteLine($"The first stage of Canny Edge Detection is the Black and White filter. It is made up
225     of 3 parameters {Log.Red}Red{Log.Blue}, {Log.Green}Green{Log.Blue}, {Log.Blue}Blue{Log.Blue} Ratios.");
226
227     while (!happy)
228     {
229         if (_classInputHandel.TryGetDouble(
230             $"Value for {Log.Red}Red{Log.Blue} (Old: {_detector.RedRatio}, Range: 0 <= x <= 1)",
231             out double newRedRatio) && newRedRatio <= 1 && newRedRatio >= 0 && newRedRatio !=
232             _detector.RedRatio)
233         {
234             _logInstance.Warn(_runGuid, $"Changed {Log.Red}Red{Log.Blue} ratio {_detector.RedRatio} ->
235             {newRedRatio}");
236             _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.RedRatio} -> {newRedRatio}{Log.Blue}");
237             _detector.RedRatio = newRedRatio;
238         }
239         else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.RedRatio}{Log.Blue}");
240         _menuInstance.WriteLine();
241
242         if (_classInputHandel.TryGetDouble(
243             $"Value for {Log.Green}Green{Log.Blue} (Old: {_detector.GreenRatio}, Range: 0 <= x <= 1)",
244             out double newGreenRatio) && newGreenRatio <= 1 && newGreenRatio >= 0 &&
245             newGreenRatio != _detector.GreenRatio)
246         {
247             _logInstance.Warn(_runGuid, $"Changed {Log.Green}Green{Log.Blue} ratio {_detector.GreenRatio} ->
248             {newGreenRatio}");
249             _menuInstance.WriteLine($"{Log.Green}Changed: {_detector.GreenRatio} -> {newGreenRatio}{Log.Blue}");
250             _detector.GreenRatio = newGreenRatio;
251         }
252         else _menuInstance.WriteLine($"{Log.Orange}Kept Default: {_detector.GreenRatio}{Log.Blue}");
253         _menuInstance.WriteLine();
254
255         if (_classInputHandel.TryGetDouble(
256             $"Value for {Log.Blue}Blue{Log.Blue} (Old: {_detector.BlueRatio}, Range: 0 <= x <= 1)",
257             out double newBlueRatio) && newBlueRatio <= 1 && newBlueRatio >= 0 && newBlueRatio !=
258             _detector.BlueRatio)
259         {

```

```

255         _logInstance.Warn(_runGuid, $"Changed {Log.Blue}Blue{Log.Blank} ratio {_detector.BlueRatio} ->
256         {newBlueRatio}");
257         _menuInstance.WriteLine($"'{Log.Green}Changed: {_detector.BlueRatio} -> {newBlueRatio}{Log.Blank}'");
258         _detector.BlueRatio = newBlueRatio;
259     }
260     else _menuInstance.WriteLine($"'{Log.Orange}Kept Default: {_detector.BlueRatio}{Log.Blank}'");
261     _menuInstance.WriteLine();
262     _menuInstance.WriteLine("Converting to black and white. This may take some time...");

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

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

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

273
274     if (opt.ToLower().StartsWith("y")) happy = true;
275     else
276     {
277         _menuInstance.ClearUserSection();
278         _menuInstance.WriteLine($"{Log.Pink}Please re-enter your values.{Log.Blank}");
279     }
280 }
281
282     _menuInstance.ClearUserSection();
283 }

284 private void ShowDialog()
285 {
286     _menuInstance.ClearUserSection();
287     _menuInstance.WriteLine("You have selected to run edge detection steps one after another, this means that at the
288     end of every step you will be shown your image and then have the option to continue to the next step or change
289     variables.");
290     _classInputHandel.WaitInput($"{Log.Grey}(Enter to Continue){Log.Blank}");
291     _menuInstance.WriteLine();
292 }

293 public double[,] Result() => _resultArray;
294 }
```

#### 5.2.2.4 WindowsForms

##### PathfindImageForm.cs (Partial)

```

1  public partial class PathfindImageForm : Form
2  {
3      private static readonly Structures.Coord invalidCoord = new Structures.Coord { X = -1, Y = -1 };
4
5      private Bitmap _image;
6      private readonly Bitmap _originalImage;
7      private int _width;
8      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         pictureBox.Width = _width * 2 / 3 - 12;
54         pictureBox.Height = _height - 24;
55         pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;
56         pictureBox.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 / pictureBox.Width) * location.X;
101     int y = (int)((double)_image.Height / pictureBox.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     pictureBox.Image = _image;
153 }
154
155 private void DrawCross(Structures.Coord center, Color colour)
156 {
157     double xRatio = (double)_image.Width / pictureBox.Width;
158     double yRatio = (double)_image.Height / pictureBox.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 pictureBox_Click(object sender, EventArgs e)
184 {
185     MouseEventArgs mouseEvent = (MouseEventArgs)e;
186     Structures.Coord clickCord = ConvertImageBoxToBitmapCoord(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 ->
205             convertToMST)\n";
206
207         string endWarning = string.Empty;
208         if (bool.Parse(Settings.UserSettings["endOnFind"].Item1))
209             endWarning = "(Warning causes longer times from different start nodes. To change goto settings ->
210             pathfinding -> endOnFind)\n";
211
212         runningBox.Text = "Current Pathfinding Settings\n\n" +
213                         $"\\nAlgorithm: {Settings.UserSettings["pathfindingAlgorithm"].Item1}" +
214                         $"\\n\\nUsing Minimum Spanning Tree: {({Settings.UserSettings["convertToMST"].Item1 == "true" ?
215             "Yes" : "No"})}" +
216                         $"\\n{mstWarning}" +
217                         $"\\nSnapping to grid: {({Settings.UserSettings["snapToGrid"].Item1 == "true" ? "Yes" : "No"})}" +
218                         $"\\n{snapWarning}" +
219                         $"\\nEnd pathfinding on Finding End (Dijkstra Only): {({Settings.UserSettings["endOnFind"].Item1 ==
220             "true" ? "Yes" : "No"})}" +
221                         $"\\n{endWarning}";
222     }
223
224     private int GetDistanceBetweenNodes(Structures.Coord start, Structures.Coord goal) =>
225     (int)Utility.GetDistanceBetweenNodes(start, goal);
226
227     private int nodes;
228
229     private void UpdateNodes()
230     {
231         nodes++;
232         nodeBox.Text = $"Progress {(nodes / (double)_graph.GetAllNodes().Length * 100):f2}% complete\\nNode {nodes} out
233         of {_graph.GetAllNodes().Length}";
234         if (nodes % 2 == 0) Update();
235     }
236
237     private void goButton_Click(object sender, EventArgs e)
238     {
239         nodes = 0;
240
241         workingButton.Visible = true;
242         textBox.Visible = false;
243         runningBox.Visible = true;
244         if (Settings.UserSettings["pathfindingAlgorithm"].Item1.ToLower() == "dijkstra") nodeBox.Visible = true;
245
246         Update();
247
248         try { if (startNode != invalidCord && endNode != invalidCord)
249             {
250                 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     else if (Settings.UserSettings["pathfindingAlgorithm"].Item1.ToLower() == "astar")
273     {
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         pictureBox.Width = _width * 2 / 3 - 12;
35         pictureBox.Height = _height - 24;
36         pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;
37         pictureBox.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.Blue}");
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 inputHandle = new Input(menuInstance);
20
21     bool running = true;
22
23     while (running)
24     {
25         menuInstance.SetPage("Welcome Menu");
26         int opt = inputHandle.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                 inputHandle.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
39                 menuInstance.WriteLine();
40
41                 RunNewImage(menuInstance, CLILoggingInstance);
42                 break;
43             // Recall
44             case 1:
45                 menuInstance.SetPage("Recall Old Image");
46                 TextWall.SaveWelcome(menuInstance);
47                 inputHandle.WaitInput($"{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,
57                     CLILoggingInstance);
58                     settingsControl.Start();
59                 }
59                 catch (Exception ex)
60                 {
61                     menuInstance.ClearUserSection();
62                     menuInstance.Error("Your settings file is corrupt, please delete the 'settings.conf' file and
63                     restart. The program will now exit.");
64                     new Input(menuInstance).WaitInput("");
65                     Environment.Exit(0);
66                 }
67

```

```

67         menuInstance.ClearUserSection();
68         break;
69     // Exit
70     case 3:
71         menuInstance.SetPage("Exit");
72         running = false;
73         break;
74     }
75 }
76
77 private static void RunSaveFile(Menu menu, Log logger)
78 {
79     Input inputHandel = new Input(menu);
80     Guid runGuid = Logger.CreateRun();
81
82     menu.ClearUserSection();
83     logger.Event(runGuid, $"Beginning recall of map file (Run Id: {runGuid})");
84
85     SaveFile saveFile = new SaveFile(menu, logger, runGuid);
86
87     try
88     {
89         MapFile recalledMap = saveFile.Read();
90
91         bool running = true;
92
93         while (running)
94         {
95             menu.SetPage("Recalled Options");
96
97             int opt = inputHandel.GetOption("What would you like to do with your recalled map?",
98                 new[]
99                 {
100                     "View File / Map Information",
101                     "Change File Information",
102                     "Clone File",
103                     "Rename File",
104                     "Delete File",
105                     "Pathfind Through Image",
106                     "Exit"
107                 });
108         );
109
110         switch (opt)
111         {
112             case 0:
113                 menu.SetPage("Image Details");
114                 string[] items = { "Screenshot", "Hand Drawn", "Photograph", "Other" };
115                 menu.ClearUserSection();
116                 menu.WriteLine("Your current save file information:");
117                 menu.WriteLine($"Name: {recalledMap.Name}");
118                 menu.WriteLine($"Description: {recalledMap.Description}");
119                 menu.WriteLine();
120                 menu.WriteLine($"Type of image: {Log.Orange}{items[recalledMap.Type]}{Log.Blue}");
121                 menu.WriteLine($"Was it inverted: {Log.Purple}{{recalledMap.IsInverted ? "Yes" :
122                   "No"} }{Log.Blue}");
123                 menu.WriteLine($"Time Created: {Log.Green}{recalledMap.TimeCreated}{Log.Grey}");
124                 inputHandel.WaitInput($"{Log.Grey}(Enter to Continue){Log.Blue}");
125                 break;
126             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, $"Changing file settings, see current run save folder for the save
130     ↵ file.");
131     switch (option)
132     {
133         case 0:
134             string newName =
135                 inputHandel.GetInput("What do you want to change the title of the save to?");
136             recalledMap.Name = newName;
137             break;
138         case 1:
139             string newDescription =
140                 inputHandel.GetInput("What do you want to change the title of the save to?");
141             recalledMap.Description = newDescription;
142             break;
143         case 2:
144             recalledMap.Type = inputHandel.GetOption("What type of image is this save?",
145                 new[] { "Screenshot", "Hand Drawn", "Photograph", "Other" });
146             break;
147     }
148
149     string path = recalledMap.Save(runGuid);
150     if (bool.Parse(Settings.UserSettings["shortNames"]
151         .Item1))
152         File.Move(path,
153             path.Replace(Path.GetFileName(path)
154                 .Split('.')[0],
155                 recalledMap.Name));
156         break;
157     case 2:
158         menu.SetPage("Clone Image");
159         File.Copy(recalledMap._filePath,
160             ↵ recalledMap._filePath.Replace(Path.GetFileName(recalledMap._filePath).Split('.')[0],
161             ↵ Path.GetFileName(recalledMap._filePath).Split('.')[0] + "-CLONE"));
162         logger.Event($"Cloned {recalledMap._filePath}.");
163         break;
164     case 3:
165         menu.SetPage("Rename Image");
166         string name = inputHandel.GetInput("What would you like to rename the file too?");
167         logger.Event(runGuid, $"Renamed {Path.GetFileName(recalledMap._filePath).Split('.')[0]} to
168             ↵ {name}.");
169         File.Move(recalledMap._filePath,
170             recalledMap._filePath.Replace(Path.GetFileName(recalledMap._filePath).Split('.')[0], name));
171         break;
172     case 4:
173         menu.SetPage($"{Log.Red}DANGER: Delete Image{Log.Blue}");
174         if (inputHandel.GetOption("Are you sure you want to delete the save?",
175             new[] { $"{Log.Red}No{Log.Blue}", $"{Log.Red}Yes{Log.Blue}" }) == 2)
176         {
177             logger.Warn(runGuid, $"Save file at path {recalledMap._filePath} deleted.");
178             File.Delete(recalledMap._filePath);
179             running = false;
180         }
181         break;
182     case 5:
183         menu.SetPage("Pathfinding Window");
184         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, $"Begin processing of new image (Run Id: {runGuid}).");
208
209     NewImage newImage = new NewImage(menu, logger, runGuid);
210
211     try
212     {
213         Structures.RawImage rawImage = newImage.Read();
214
215         menu.WriteLine();
216         menu.WriteLine("Successfully managed to read in your image, please look carefully at the next popup and make");
217         ← sure it is your image.");
218         i.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
219         menu.WriteLine();
220
221         logger.Event(runGuid, $"Confirming is correct file.");
222         ViewImageForm beforeForm = new ViewImageForm(rawImage.Pixels.ToBitmap());
223         beforeForm.ShowDialog();
224         menu.ClearUserSection();
225
226         TextWall.FileDetails(menu, rawImage);
227         menu.WriteLine();
228
229         bool correctImage = utility.IsYes(i.GetInput("Is this the correct image (y/n)?"));
230         if (!correctImage) throw new Exception("You asked for the processing of your map to stop.");
231
232         int opt = i.GetOption("Select a version of edge detection to run:", new[] {
233             "Preset - Hand Drawn Map",
234             "Preset - Screenshot",
235             "Preset - Photograph",
236             "Multi-threaded - Fast, all options decided at the start which allows for faster processing.",
237             "Synchronous - Slow, options can be changed after each step and steps can be repeated." });
238
239         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");
287     ← 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
288     ← select invert at the next prompt.");
289     menu.WriteLine();
290     i.WaitInput($"{Log.Grey}(Enter to continue){Log.Blank}");
291     menu.WriteLine();
292
293     ViewImageForm edgeImageForm = new ViewImageForm(resultOfEdgeDetection.ToBitmap());
294     edgeImageForm.ShowDialog();
295
296     MapFile saveMapFile = rawImage.MapFile;
297
298     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 }
```